EMBEDDED SYSTEMS

BASED ON CORTEX-M4 AND THE RENESAS SYNERGY PLATFORM

2020
PROF. DOUGLAS RENAUX, PHD
PROF. ROBSON LINHARES, DR.
UTFPR / ESYSTECH

RENESAS ELECTRONICS CORPORATION

12 WEEK COURSE OUTLINE (1/2)

1) Introduction
   - What are embedded systems
   - Characteristics
   - Sample Market Segments
   - The IoT Era
2) Computer Architecture
   - RISC vs CISC
3) ARM Cortex-M Architecture
   - Block Diagram
   - Registers
   - Instruction set
   - Memory access
   - Exception handling
4) Memory
   - SRAM
   - DRAM (SDRAM, DDR)
   - ROM/EEPROM/Flash
5) Timer and GPIO
   - Timer
   - PWM
   - GPIO
   - Simple drivers (e.g. LED, Relay)
   - Power drivers (motors)
6) Interrupt Controller

12 WEEK COURSE OUTLINE (2/2)

7) Analog Interfacing
   - ADC / DAC
8) Serial Communication
   - UART
   - SPI
   - I2C
9) CAN
   - Physical interface
   - Stack
10) USB
   - Physical interface
   - Stack
11) Ethernet
   - Physical interface
   - Stack
12) Software Development
   - Software Process
   - UML Class Diagram
   - UML State Machine Diagram
13) Concurrent Programming
   - Tasks / Context Switching, Scheduling
   - Semaphores, Signals / Messages
   - Common problems to avoid: deadlock, priority inversion
14) RTOS
   - Thread Management
   - Inter-thread communication and synchronization
   - Timing Services
   - Memory Management

LIST OF LABS – BASED ON SK-S7G2

- Lab1 – Synergy Installation - try demo program on the S7G2 board. Requirements: none.
- Lab2 – Sample C program - means to access hardware peripherals; memory organization of a C program. Requirements: Section 5.
- Lab3 – Assembly Programming ATPCS - access from C a function written in assembly. Requirements: Section 5.
- Lab4 – Peripheral Sample device driver. Requirements: Section 6.
- Lab5 – Serial Communication. Requirements: Section 8.
- Lab6 – Display and Touch. Requirements: Section 8.
- Lab7 – RTOS. Requirements: Section 14.
- Lab8 – USB Device. Requirements: Section 14.
- Lab9 – IoT. Requirements: Section 14.

DISCLAIMER

- This course material was developed to contribute to the several forms of training in the area of Embedded Systems, but particularly with undergraduate courses such as Electrical Engineering, Computer Engineering and Computer Science.
- Contents can be freely copied and distributed to students both for commercial and non-commercial purposes, as long as:
  - Credit to original work mentioning authors and Renesas as the distributor of this work.
  - The contents can be freely modified to suit the needs of specific courses, all figures made by the authors may be freely used without modification as long as credited; likewise, all figures authored by Renesas may be freely used without modification as long as credited. All figures from other sources, if used in derived work or in other works must request authorization from the original author/copyright holder.

AUTHORS

- The authors: Douglas Renaux and Robson Linhares are faculty at UTFPR in the subjects of Embedded Real-Time Systems and Computer Architecture and Organization.

  UTFPR is the Brazilian Federal University of Technology.
- eSysTech – Embedded Systems Technologies is a company providing engineering and training services in the area of Embedded Systems. It is a spin-off of the Laboratory of Innovation and Technology in Embedded Systems of UTFPR.
- Renesas is a major player in the semiconductor market. They have been providing significant support for educational material such as this one. Renesas has worldwide non-exclusive distribution rights for this material.

OVERVIEW AND PREREQUISITES

- This Embedded Systems course is organized into theory and practice parts. There are 12 theory sections and 9 labs. The labs solutions can be made available to instructors. All labs are conceived to be developed on the Renesas SK-S7G2 board, based on an ARM Cortex-M4F MCU.
- The course assumes that the students have previous knowledge on:
  - C programming for embedded systems
  - Microcontrollers and assembly programming (on an architecture other than ARM)
  - Digital Systems
  - Digital communications and networks

1 – INTRODUCTION

1. What are Embedded Systems
2. Characteristics
3. Market Segments
4. The IoT Era

WHAT ARE EMBEDDED SYSTEMS?

An Embedded System (a.k.a. Embedded Computing System) is a computing system that is built-into (i.e. embedded) a larger device, such as an equipment, a system, or a vehicle.

Embedded Systems (ES) are usually application-specific and have real-time constraints; thus, many ES are also real-time systems. Often, ES are used in control loops: reading sensors, processing data, and generating outputs that control the device they are embedded into. Finite State Machines are commonly used to model the behavior of ES.

CHARACTERISTICS OF EMBEDDED SYSTEMS (1/4)

Typical characteristics of an Embedded System are:

1. **Microcontroller based system** consisting of a processor,

   non-volatile memory (Flash), volatile memory (RAM), and a large number of inputs and output interfaces as well as communication channels.

2. **Cost effective implementations** as many device architectures are cost-driven.

3. **Energy efficient solutions** as many devices are battery powered. Current trend is to develop battery-less devices that harvest energy from the environment.

CHARACTERISTICS OF EMBEDDED SYSTEMS (2/4)

4. **Heterogeneous.** While desktop computers are based on standard platforms (Windows PC, Apple IOS, ...) there is a large variety of hardware and software for embedded systems.
5. **Variety of restrictions** to the design solutions, such as:
   a) Physical: Size, Weight, Temperature Range, Vibration, Dust, Spills, Water;
   b) Computational resources: processing speed, non-volatile memory, RAM, available I/O;
   c) Response time.
6. **Interconnected.** Embedded devices and systems are ever more interconnected to each other. Trend is to increase the interconnection rate (IoT).

CHARACTERISTICS OF EMBEDDED SYSTEMS (3/4)

7. **Reliability**

   The ability of a system or component to function under stated conditions for a specified period of time.

8. **Availability**

   The ability of a system or component to function at a specific moment of interval of time.

9. **Maintainability**

   Measures how easily and how fast a system can be restored to operational status after a failure.

10. **Testability**

    The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met.

CHARACTERISTICS OF EMBEDDED SYSTEMS (4/4)

11. Scalability

The ability of a system to handle increased workload by repeatedly and cost-effectively adding components to extend the system´s capacity.

12. Safety

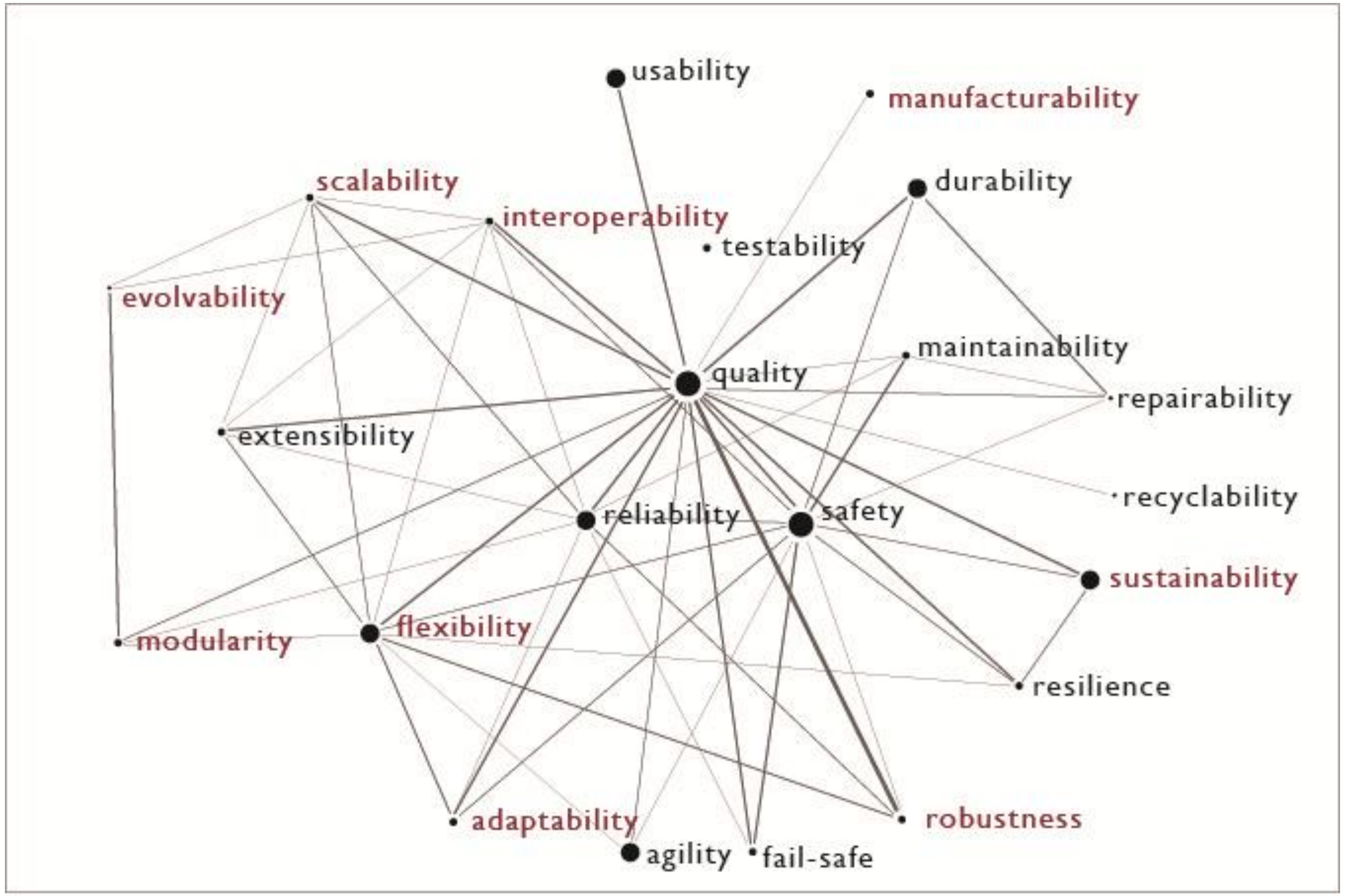
Concerns the requirement: not to harm people, the environment or other assets.

13. Security

The ability of a system to protect information and system resources with respect to integrity and confidentiality.

CHARACTERISTICS OF EMBEDDED SYSTEMS

Relationship among 22 of the most common ilities:



to read about this graph: Book Chapter about the Ilities: Chapter 4 from "Engineering Systems: Meeting Human Needs in a Complex Technological World" by de Weck O., Roos D. and Magee C, MIT Press, January 2012 (http://strategic.mit.edu)

SAMPLE MARKET SEGMENTS

- Consumer Electronics
- Telecommunications
- Home Automation
- Industrial Automation
- Transportation
  - Avionics
  - Navigation
  - Electric Vehicles
- Defense
- Medical Equipment
- ??? (many new areas to come)

Smart House

Smart Factory

Drone

Connected Care

Cool Gadget

Robots

source: Renesas DevCon2015

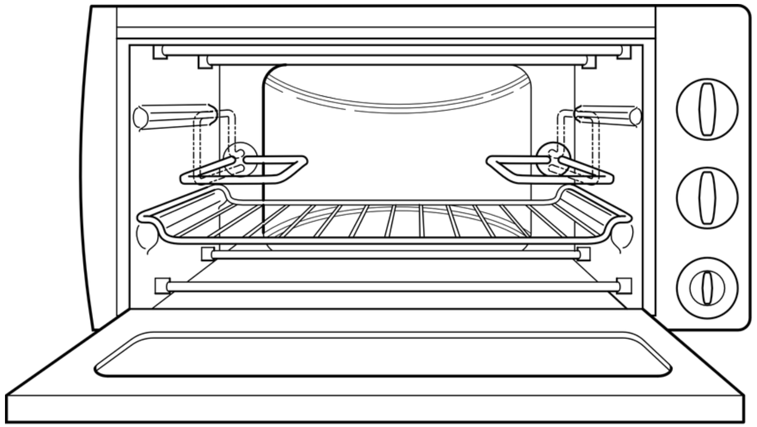
CONSUMER ELECTRONICS

- Phones
- Videogame consoles
- Printers
- Digital cameras
- Audio/Video:
  - Television
  - Music Players
  - Home Entertainment Systems
  - BRD players

source: pixabay.com (CC)

HOUSEHOLD APPLIANCES

- Washing Machines
- Dishwasher
- Air Conditioners
- Microwave Oven

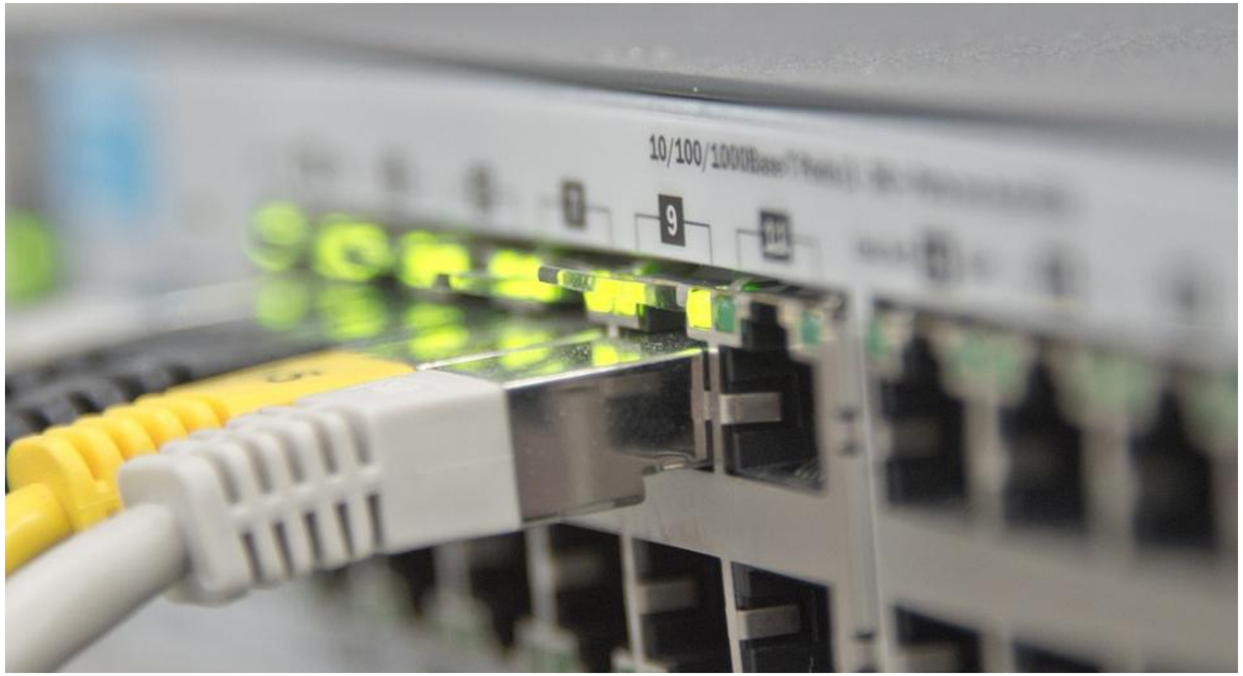


source: pixabay.com (CC)

TELECOMMUNICATIONS

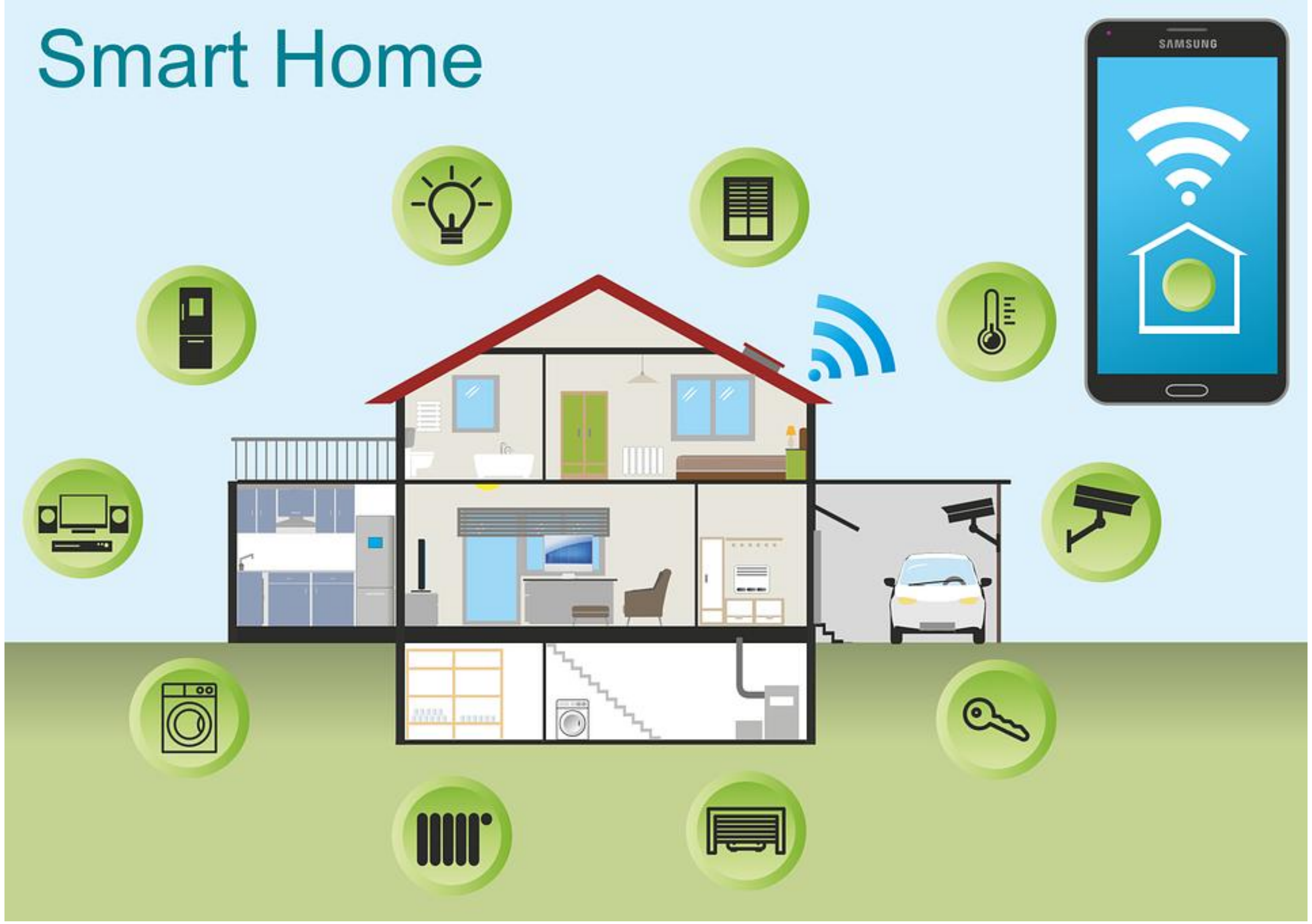
- Routers
- Satellite Phones
- Switches

source: wikimedia.org (CC)



source: pixabay.com (CC)

HOME AUTOMATION

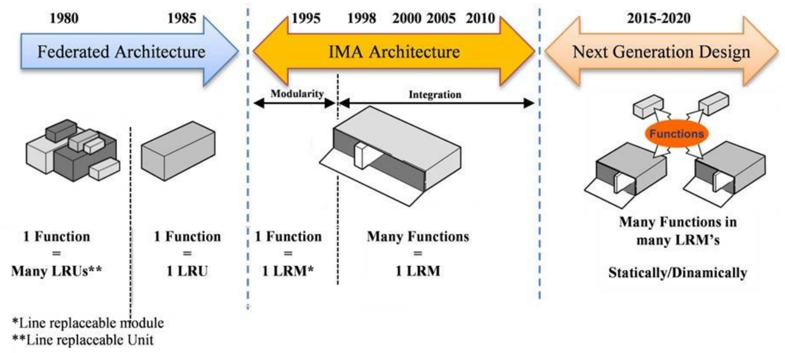


source: pixabay.com (CC)

TRANSPORTATION – AVIONICS

Glass cockpit of the

Airbus A350 XWB



1980 1985

Federated Architecture

1995 1998 2000 2005 2010

IMA Architecture

2015-2020

Next Generation Design

Modularity

Integration

Functions

1 Function = Many LRUs**

1 Function = 1 LRU

1 Function = 1 LRM*

Many Functions = 1 LRM

Many Functions in many LRM's

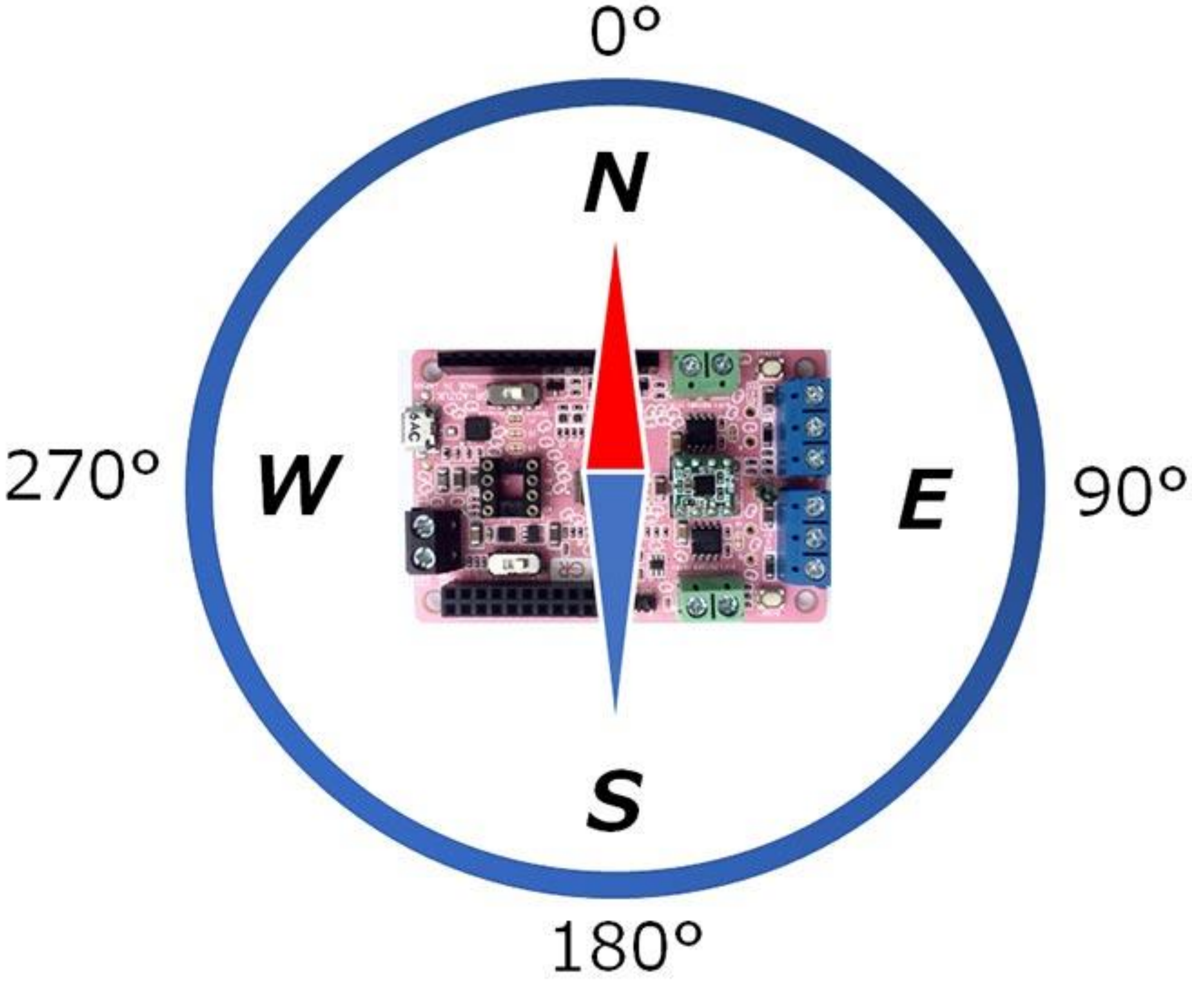
Statically/Dinamically

*Line replaceable module
**Line replaceable Unit

source: flickr (CC)

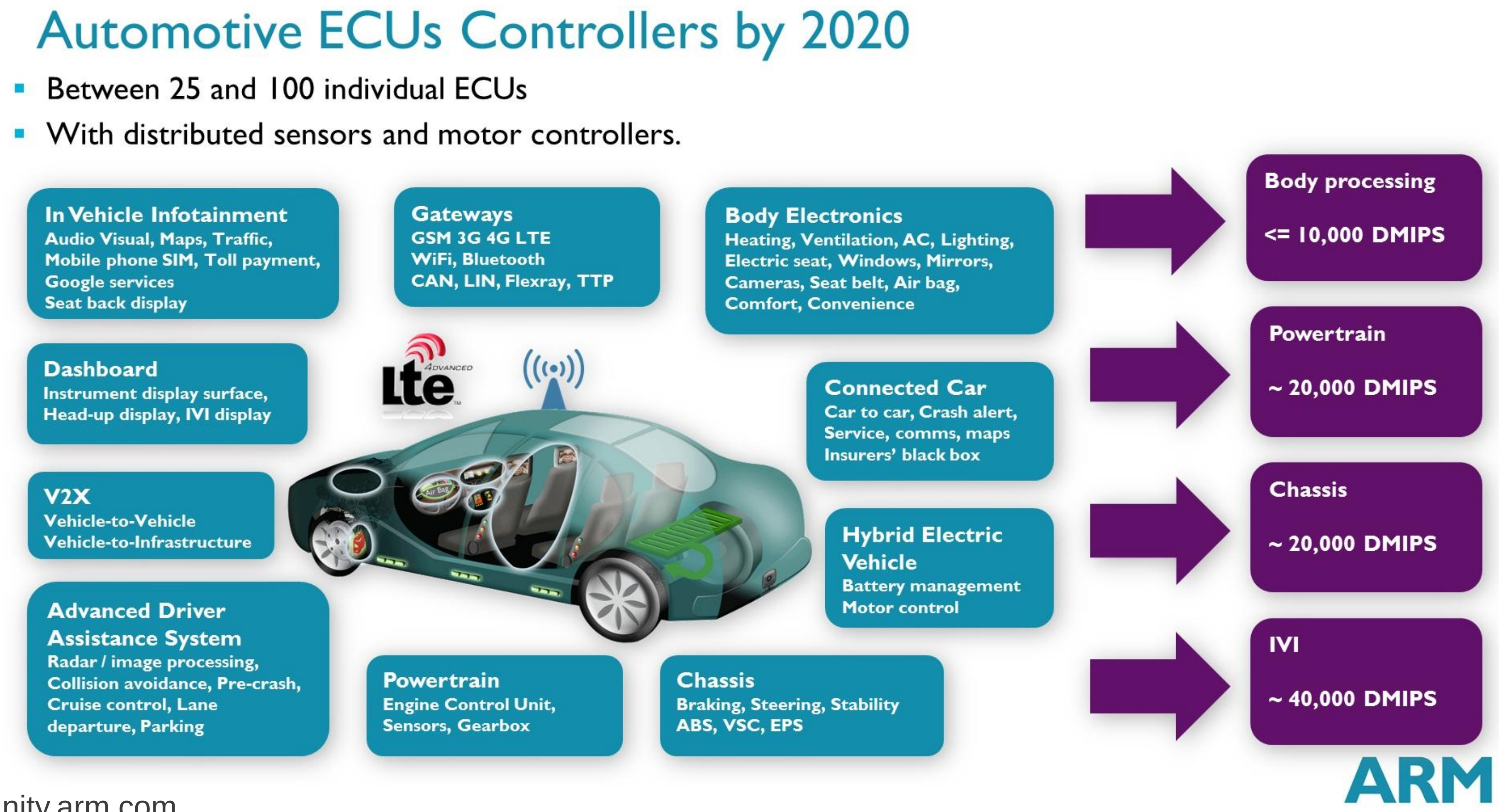
TRANSPORTATION – NAVIGATION

- Automotive GPS
- Electronic Compass



source: Renesas

TRANSPORTATION – AUTOMOTIVE



Automotive ECUs Controllers by 2020

- Between 25 and 100 individual ECUs
- With distributed sensors and motor controllers.

In Vehicle Infotainment
Audio Visual, Maps, Traffic, Mobile phone SIM, Toll payment, Google services
Seat back display

Gateways
GSM 3G 4G LTE
WiFi, Bluetooth
CAN, LIN, Flexray, TTP

Body Electronics
Heating, Ventilation, AC, Lighting, Electric seat, Windows, Mirrors, Cameras, Seat belt, Air bag, Comfort, Convenience

Dashboard
Instrument display surface, Head-up display, IVI display

Connected Car
Car to car, Crash alert, Service, comms, maps
Insurers' black box

V2X
Vehicle-to-Vehicle
Vehicle-to-Infrastructure

Hybrid Electric Vehicle
Battery management
Motor control

Advanced Driver Assistance System
Radar / image processing, Collision avoidance, Pre-crash, Cruise control, Lane departure, Parking

Powertrain
Engine Control Unit, Sensors, Gearbox

Chassis
Braking, Steering, Stability
ABS, VSC, EPS

Body processing
<= 10,000 DMIPS

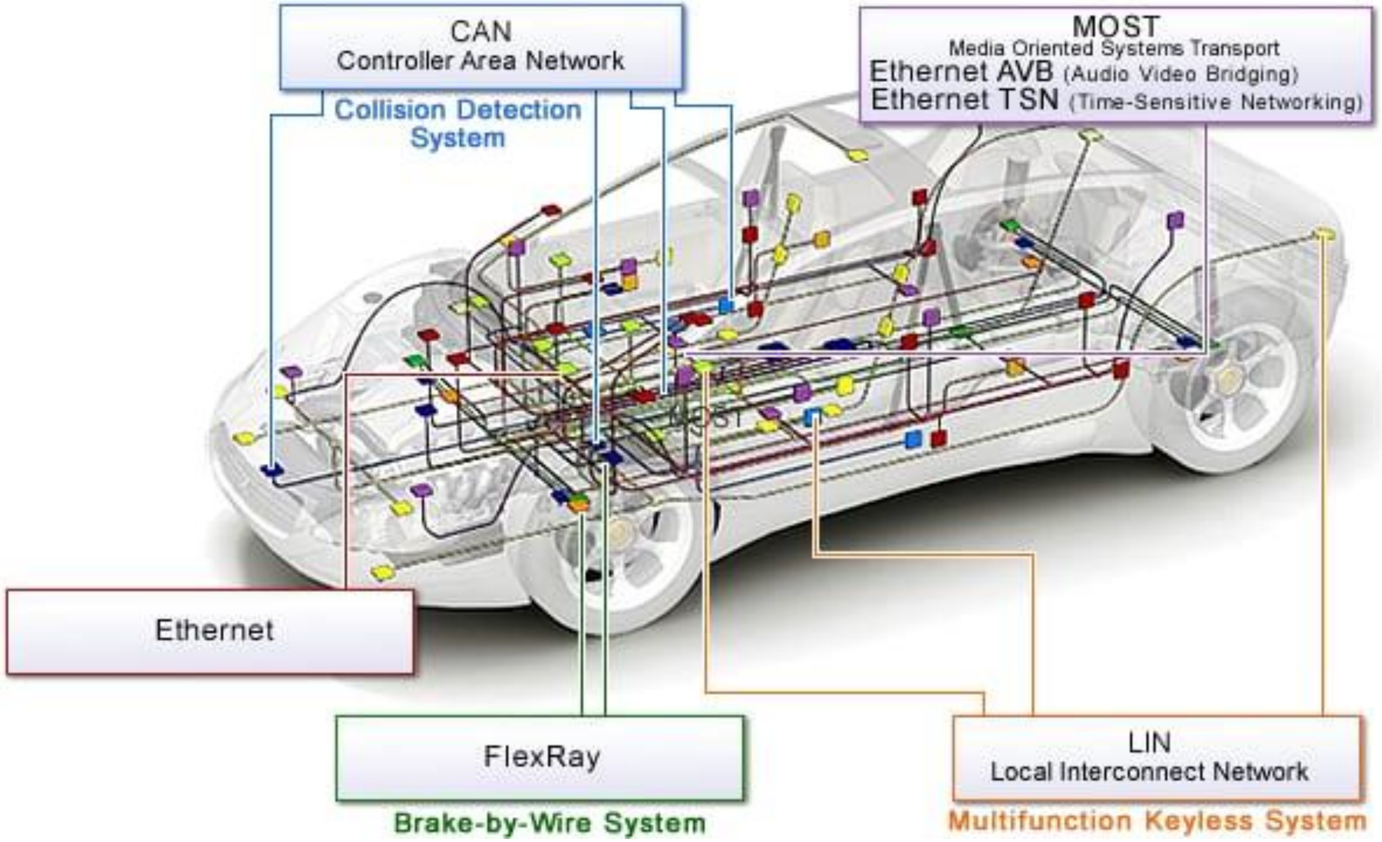
Powertrain
~ 20,000 DMIPS

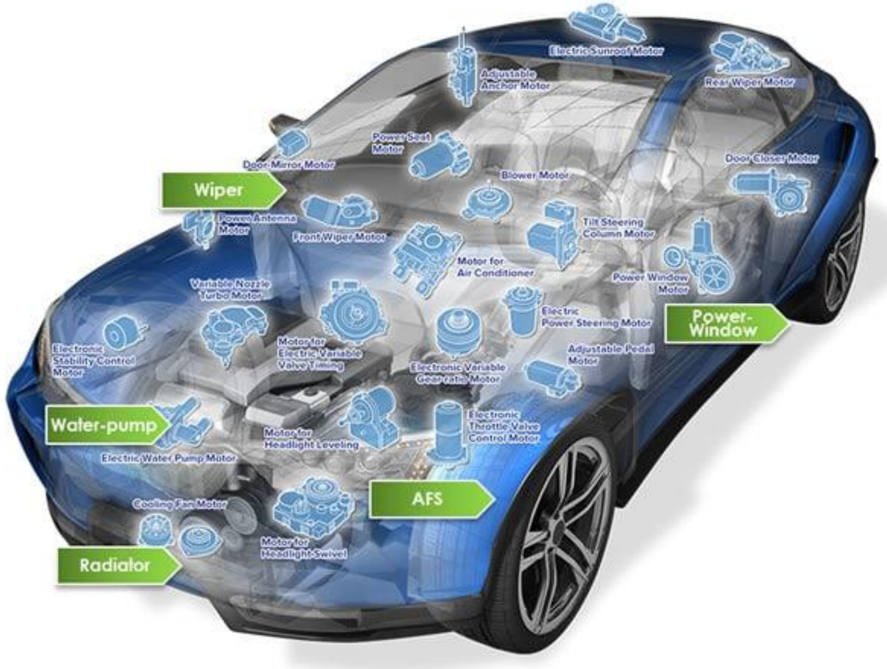
Chassis
~ 20,000 DMIPS

IVI
~ 40,000 DMIPS

source: community.arm.com

TRANSPORTATION – AUTOMOTIVE


CAN
Controller Area Network
Collision Detection System
MOST
Media Oriented Systems Transport
Ethernet AVB (Audio Video Bridging)
Ethernet TSN (Time-Sensitive Networking)
MOST
Ethernet
FlexRay
Brake-by-Wire System
LIN
Local Interconnect Network
Multifunction Keyless System


Electric Sunroof Motor
Adjustable Anchor Motor
Rear Wiper Motor
Power Seat Motor
Door Mirror Motor
Door Closer Motor
Wiper
Blower Motor
Power Antenna Motor
Front Wiper Motor
Tilt Steering Column Motor
Motor for Air Conditioner
Power Window Motor
Variable Nozzle Turbo Motor
Electric Power Steering Motor
Power-Window
Electronic Stability Control Motor
Motor for Electric Variable Valve Timing
Electronic Variable Gear ratio Motor
Adjustable Pedal Motor
Water-pump
Motor for Headlight Leveling
Electronic Throttle Valve Control Motor
Electric Water Pump Motor
AFS
Cooling Fan Motor
Motor for Headlight Swivel
Radiator

source: Renesas

MEDICAL EQUIPMENT

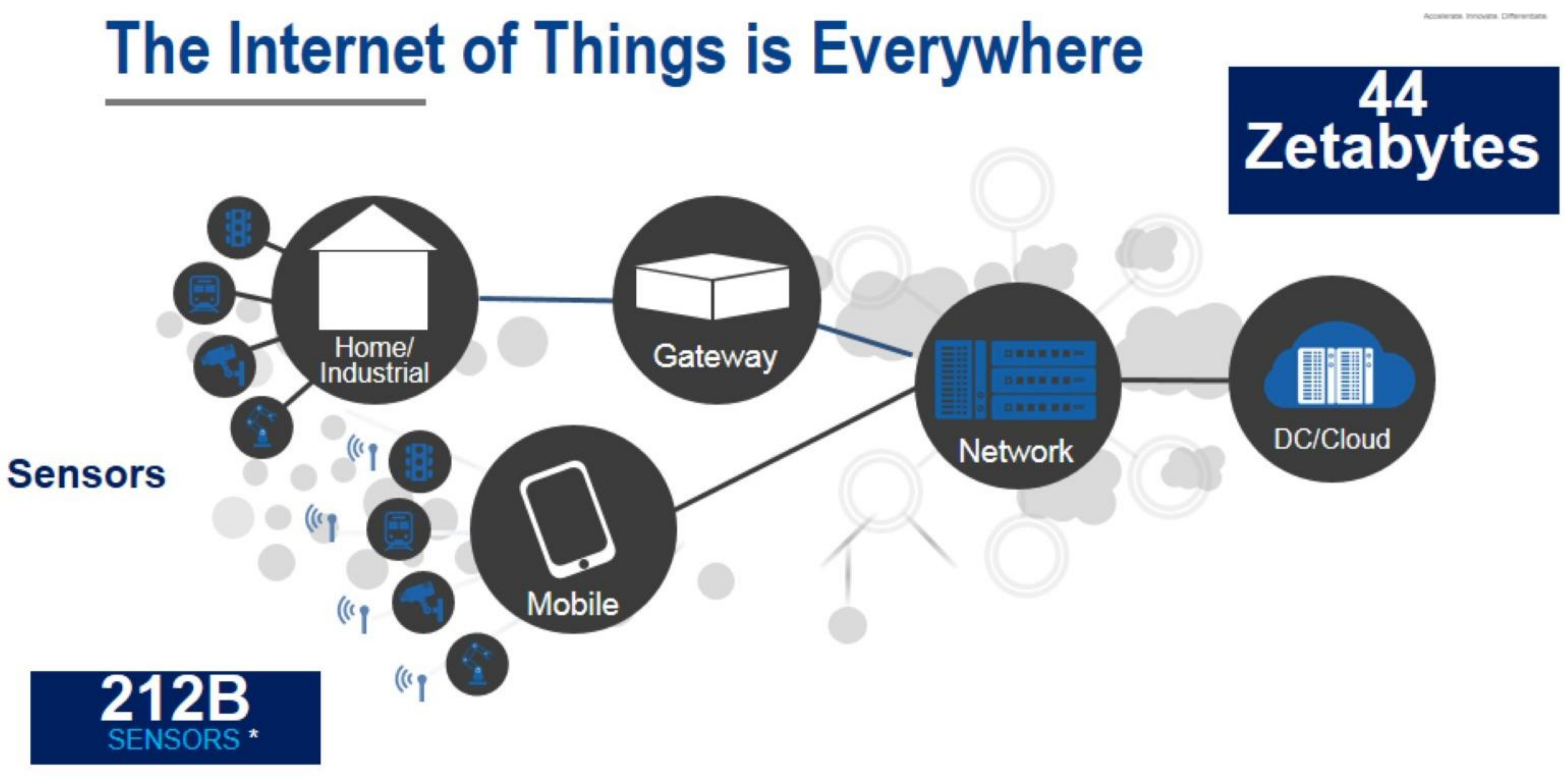
- CT Scanners
- ECG (Electrocardiogram)
- Blood Glucose Monitor
- Blood Pressure Monitor
- Body Composition Analyzer

HEALTHCARE SOLUTION USING RENESAS SYNERGY™

Accelerate Medical Device Design with IEC62304 Class C Pre-Certified Renesas Synergy™ Platform Safety Solution

source: Renesas

THE IOT ERA



The Internet of Things is Everywhere

Accelerate. Innovate. Differentiate.

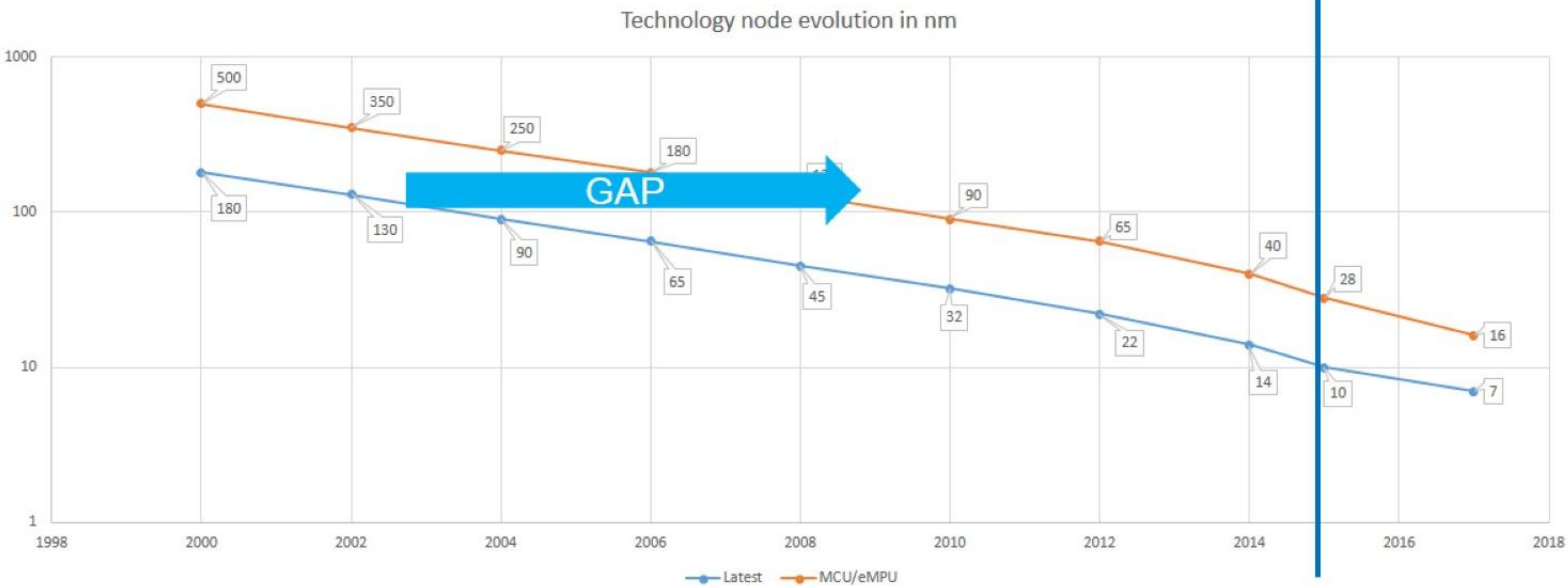
44 Zetabytes

212B SENSORS *

source: Renesas

TRENDS

Five-year gap from state of the art to MCU technology



Technology node evolution in nm

2015

Technology node = minimum transistor gate length

Source: ITRS, Renesas Electronics – based on product announcement

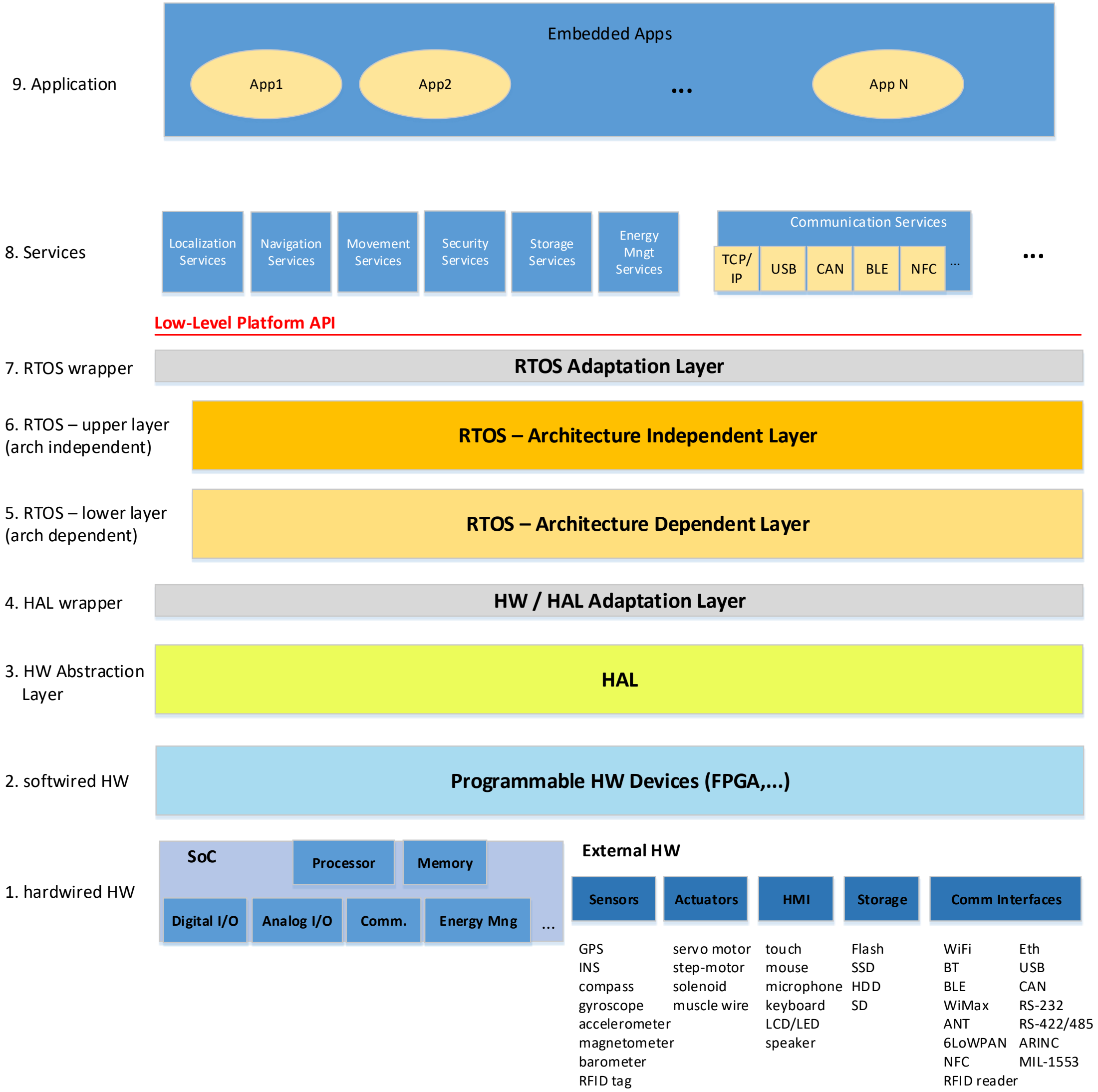
source: Renesas

EMBEDDED SYSTEMS ARCHITECTURE – GENERIC MODEL

- One of the important characteristics of Embedded Systems (ES) is its diversity. Hence, a truly generic model for an ES does not exist.
- The model presented here attempts to represent a large set of the existing ES. Hence, it is adequate to understand Embedded Systems concepts.

suggested reading: (see IEEEXplore.ieee.org)
D. Renaux, F. Pottker, "Applicability of the CMSIS-RTOS Standard to the Internet of Things", *10th Workshop on Software Technologies for Future Embedded and Ubiquitous Systems - SEUS/ISORC 2014*, June 2014.

EMBEDDED SYSTEMS ARCHITECTURE GENERIC MODEL



9. Application

Embedded Apps

App1 · App2 · … · App N

8. Services

Localization Services · Navigation Services · Movement Services · Security Services · Storage Services · Energy Mngt Services

Communication Services: TCP/IP · USB · CAN · BLE · NFC · …

…

Low-Level Platform API

7. RTOS wrapper — RTOS Adaptation Layer

6. RTOS – upper layer (arch independent) — RTOS – Architecture Independent Layer

5. RTOS – lower layer (arch dependent) — RTOS – Architecture Dependent Layer

4. HAL wrapper — HW / HAL Adaptation Layer

3. HW Abstraction Layer — HAL

2. softwired HW — Programmable HW Devices (FPGA,...)

1. hardwired HW

SoC: Processor · Memory · Digital I/O · Analog I/O · Comm. · Energy Mng · …

External HW

Sensors	Actuators	HMI	Storage	Comm Interfaces	
GPS	servo motor	touch	Flash	WiFi	Eth
INS	step-motor	mouse	SSD	BT	USB
compass	solenoid	microphone	HDD	BLE	CAN
gyroscope	muscle wire	keyboard	SD	WiMax	RS-232
accelerometer		LCD/LED		ANT	RS-422/485
magnetometer		speaker		6LoWPAN	ARINC
barometer				NFC	MIL-1553
RFID tag				RFID reader	

source: Authors

1. HARDWIRED HW

- This level is composed of hardware devices (MCU, Memory, I/O, connectors) whose connection is determined by the copper traces on a PCB, hence, not changeable after fabrication.
- Currently, a significant portion of the HW functionality of an ES is integrated into a SoC (System on Chip). External HW consists of the remaining HW not in the SoC and comprises, among others, sensors, actuators, Human-Machine Interface devices, Storage devices, and a large variety of communication interfaces.

Skywire Renesas Synergy PMOD Kit

2. SOFT WIRED HW

- In contrast to Level 1, the soft wired HW level consists of components whose connection is programmable, hence, can be modified at any time.
- Currently, the most common programmable devices are FPGAs, however, a variety of programmable logic devices (PLD) are available: FPGA, CPLD, GAL, PAL, PLA, and even ROM.

- The two hardware layers (1 and 2) compose the physical part of an embedded system. The remaining layers (3 to 9) are software layers.

SOFTWARE LAYERS

The upper layers (3 to 9) are software layers.

Embedded Systems Software have two distinct characteristics:

1. Typically the development environment (compiler toolchain) generates a single binary file that integrates all software components: device drivers, libraries (RTOS, Services, ...) and the Application. Hence, avoiding the process of reading an executable file and loading it on memory.
2. While on desktops applications are changed and upgraded quite frequently, in embedded systems, typically a single multitasking application is executed along the life of the device. Upgrades may occur but are much less frequent.

3. HARDWARE ABSTRACTION LAYER

- The HAL is comprised of a set of functions that directly access the hardware devices. These functions are also called device drivers.
- A well-designed HAL provides to the upper levels a standardized interface, providing an easy interchange of devices. For instance, if all communication devices have the same API then replacing a comm. interface (such as SPI) for another (such as I2C) is straightforward. The Renesas SSP (Synergy Software Package) is an example of such.
- Developing a device driver for the HAL requires expertise in both hardware and software. Such a development is a complex and time-consuming task typically performed in C and sometimes mixing with assembly language.

4. ADAPTATION LAYER

- An Adaptation Layer (or Wrapper) is a means of providing a common interface for different device drivers.

- If a HAL is not carefully designed, or if device drivers from different vendors are integrated in the same solution, then the software interface to the upper level may not be regular, meaning that different devices have access functions with different signatures. Such a scenario poses severe difficulties for portability and changes.
- The adaptation layer is a simple translation layer aiming at converting the non-standard interface to a standardized one. Such a translation can often be done at compile time, hence, not imposing any runtime penalty.

5. AND 6. RTOS

- Embedded Operating Systems have significant differences to O.S. used in desktop computers, including a very small memory footprint and a reduced amount of functionality. Most embedded O.S. have to provide support for real-time systems, hence, they are termed RTOS (Real-Time Operating System).
- The implementation of a well designed RTOS has at least two layers, so that different software modules implement the architectural dependent code and the architectural independent code. This approach improves modularity and improves the portability to other architectures.

7. RTOS ADAPTATION LAYER

This is a wrapper that translates the API of a given RTOS to a standard API, such as CMSIS-RTOS. Once a standard API is provided to the upper levels, all of the software in the Services and Application layers can be reused in different platforms without rewriting the calls to the RTOS.

8. SERVICES

- To cope with the large amount of functionality implement by software in current Embedded Systems, code reuse is almost mandatory. Thus, off-the-shelf software components are integrated to form the final solution. Typically, these components come in the form of libraries that are linked at compile time.
- A large variety of software components is available providing functionality for: TCP/IP stacks, USB stacks, communication protocols, georeferencing, navigation, security, storage, among many others.

9. APPLICATION LAYER

- The top layer of the model is the Application Layer. This is the software layer that implements the specific functionality of each embedded system.
- In this layer, several concurrent tasks cooperate to provide the required functionality. Concurrent programming is the most common approach to cope with the software complexity of current Embedded Systems.
- The RTOS provides the management of the concurrency, among many other services.

2 – COMPUTER ARCHITECTURE – RISC VS CISC

- Computer Generations
- The RISC Paradigm

DEFINING “COMPUTER”

Computer =

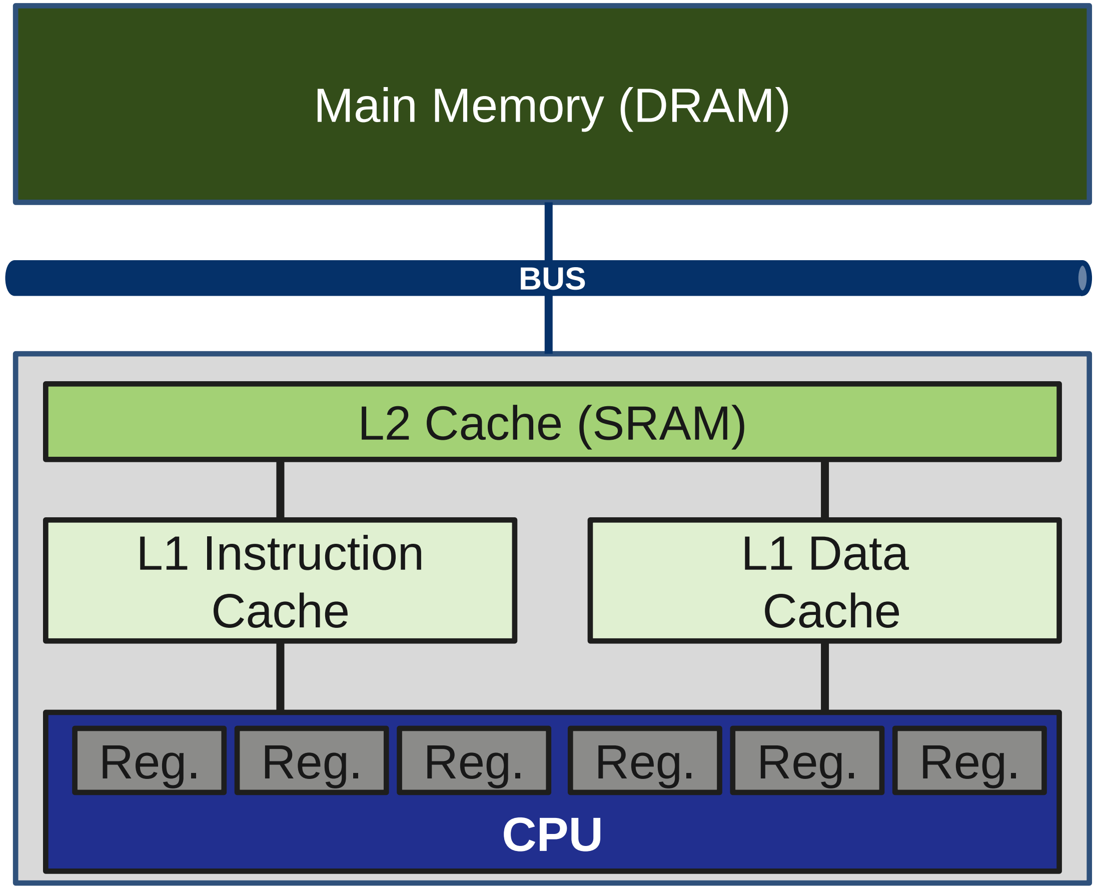
a device, or person, who performs a computation, i.e. a sequence of calculations according to an algorithm.

Hence, we consider Generation 0 of computers the generation that precedes the electronic device currently known as computer.

COMPUTER GENERATIONS

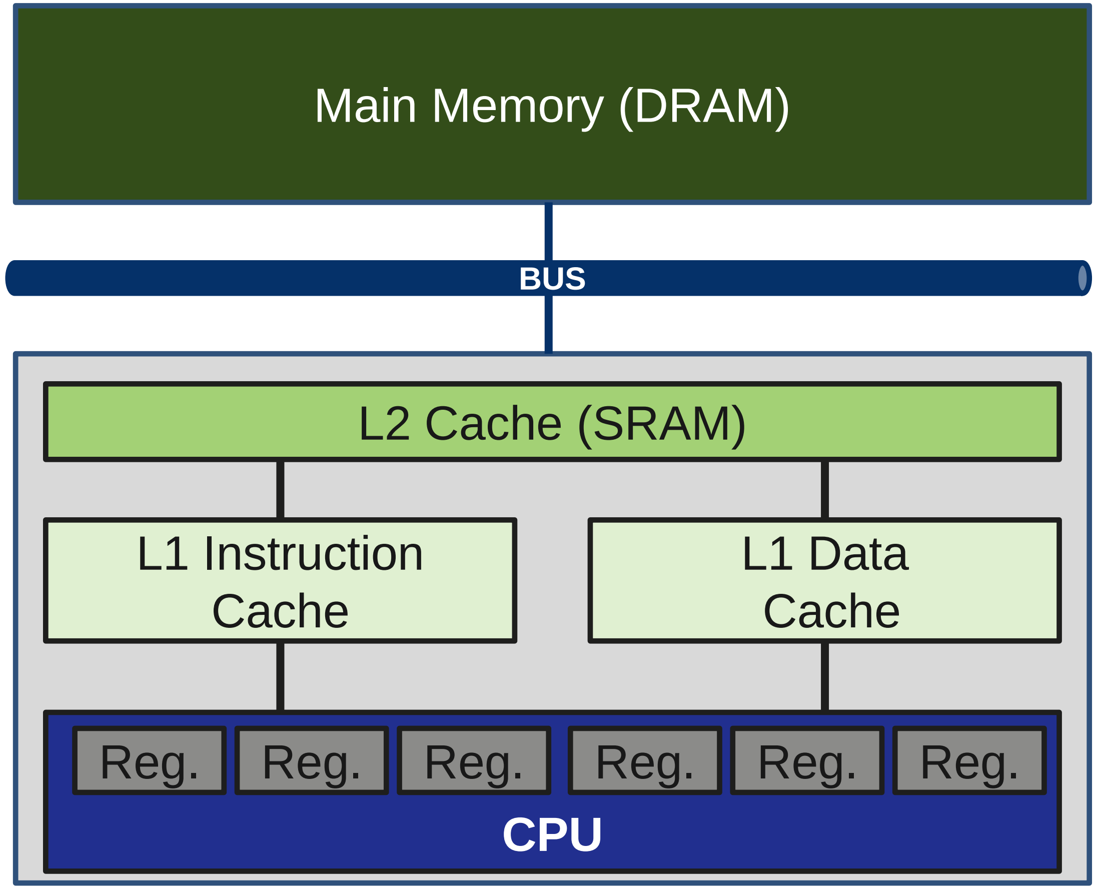
Generation	Description
0	Mechanical and Electromechanical Devices
1	40’s – Vacuum tubes ENIAC, Zuse
2	50’s – Transistors Manchester University, IBM 350
3	60’s – SSI Integrated Circuits (logic gates) Apollo Guidance Computer IBM System/360, Digital VAX
4	70’s – Microprocessor
5	2010’s? – quantic / organic / optical??? AI

COMPUTER ORGANIZATION (SIMPLIFIED)



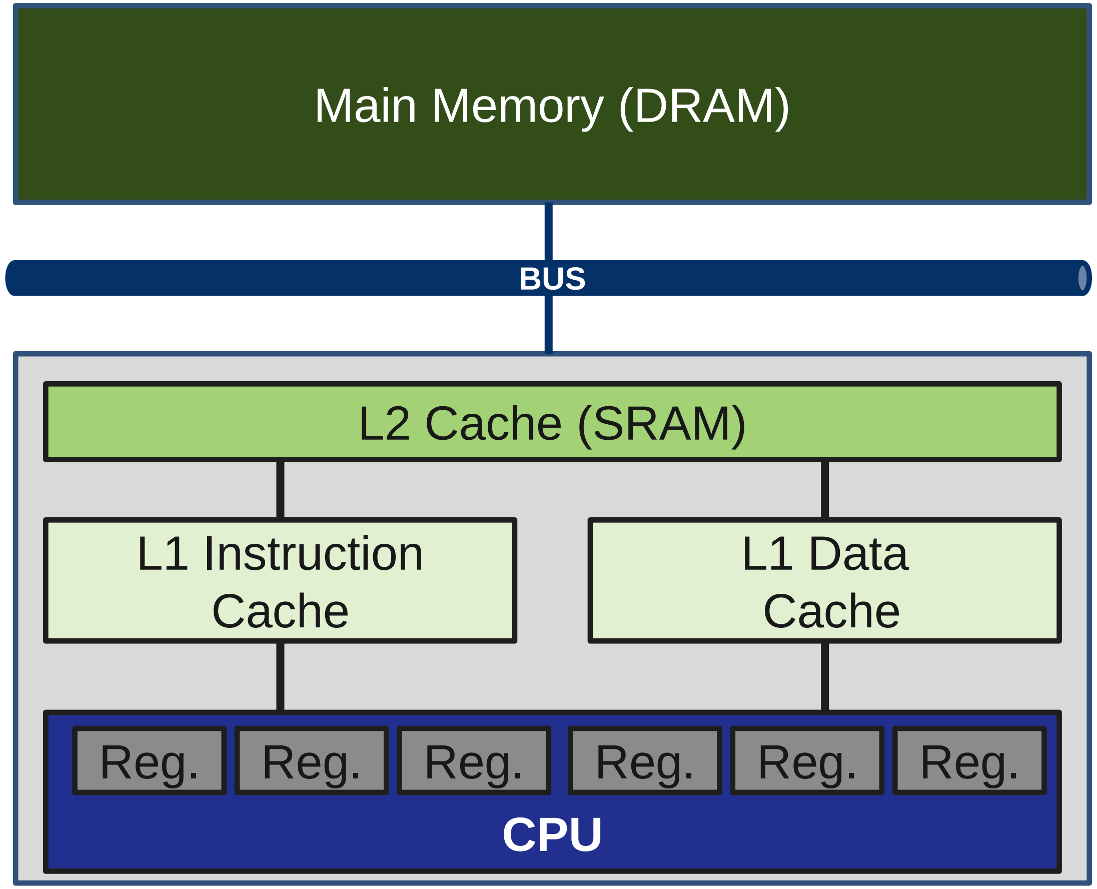
COMPLEX INSTRUCTION EXAMPLE

```
ADD [RAX+RBX*4+8], ECX
```



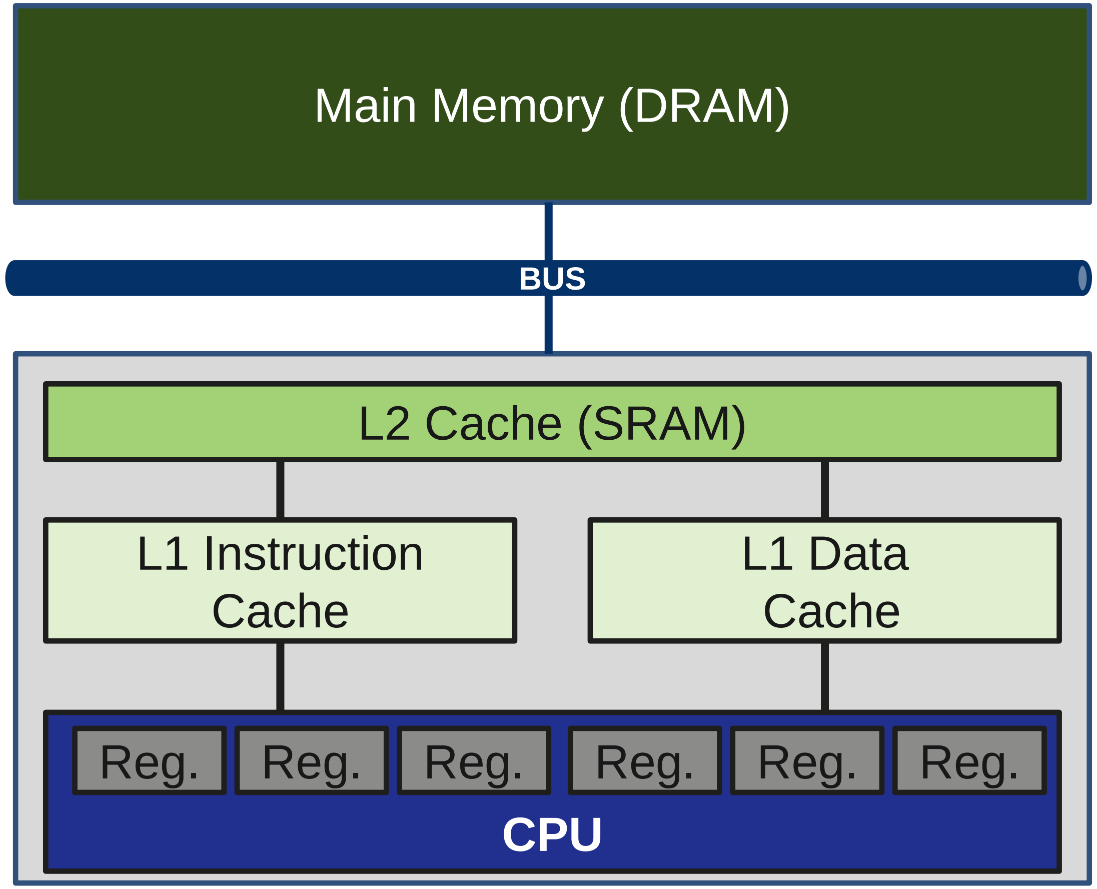
COMPLEX INSTRUCTION EXAMPLE

```
ADD [RAX+RBX*4+8], ECX
```



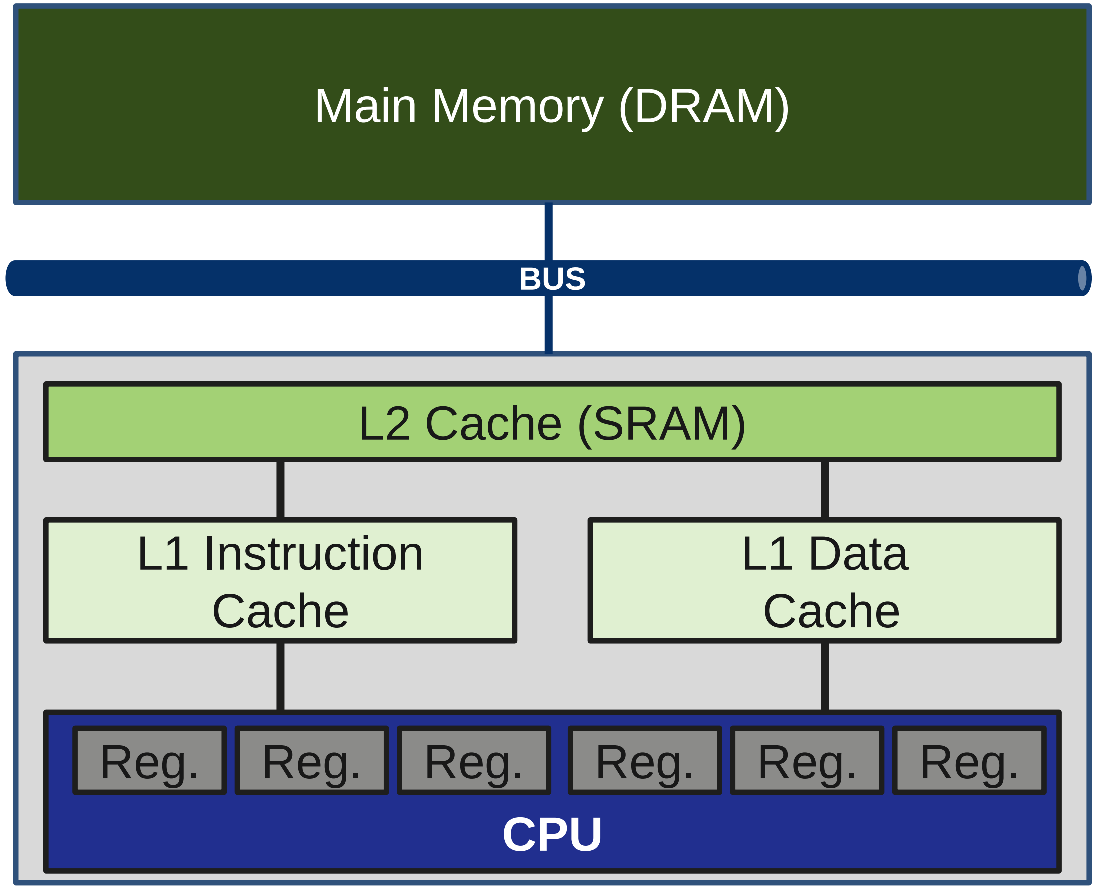
COMPLEX INSTRUCTION EXAMPLE

```
ADD [RAX+RBX*4+8], ECX
```



COMPLEX INSTRUCTION EXAMPLE

```
ADD [RAX+RBX*4+8], ECX
```

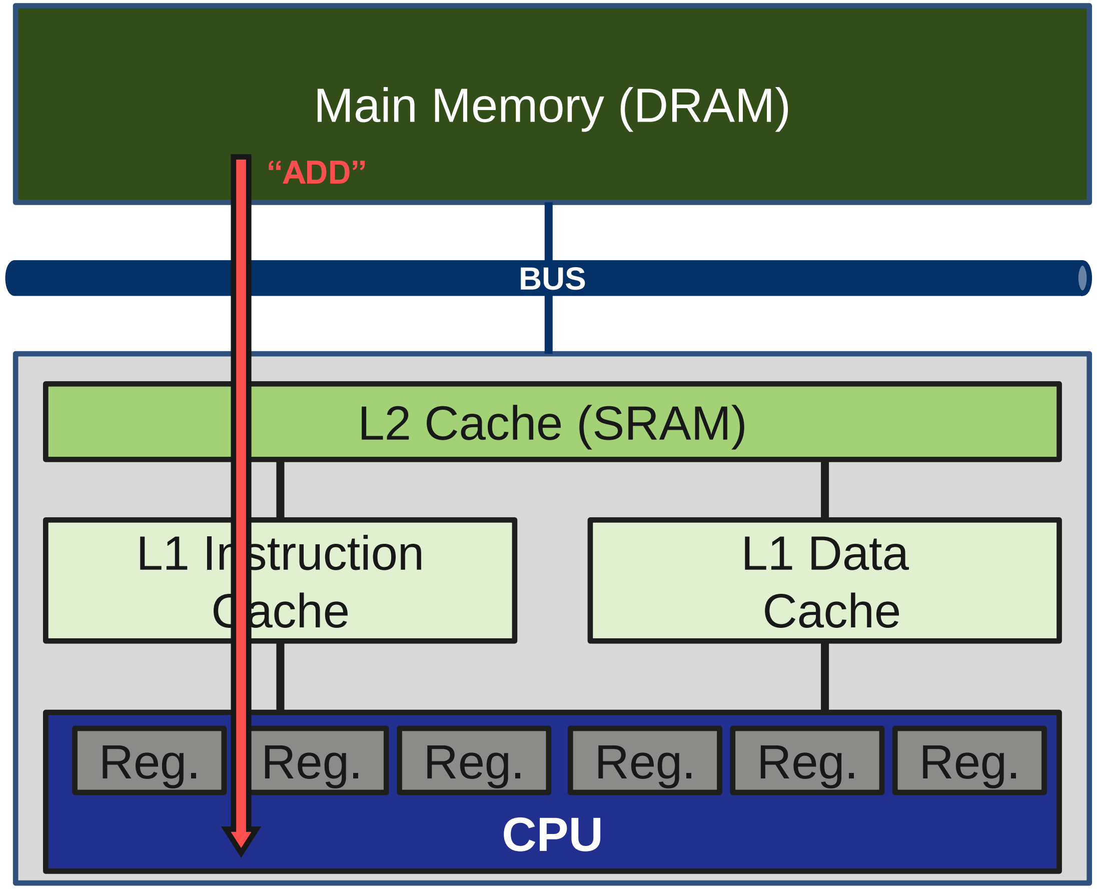


COMPLEX INSTRUCTION EXAMPLE

```
ADD [RAX+RBX*4+8], ECX
```

Steps to execute this command:

1. Fetch instruction from memory (or cache)



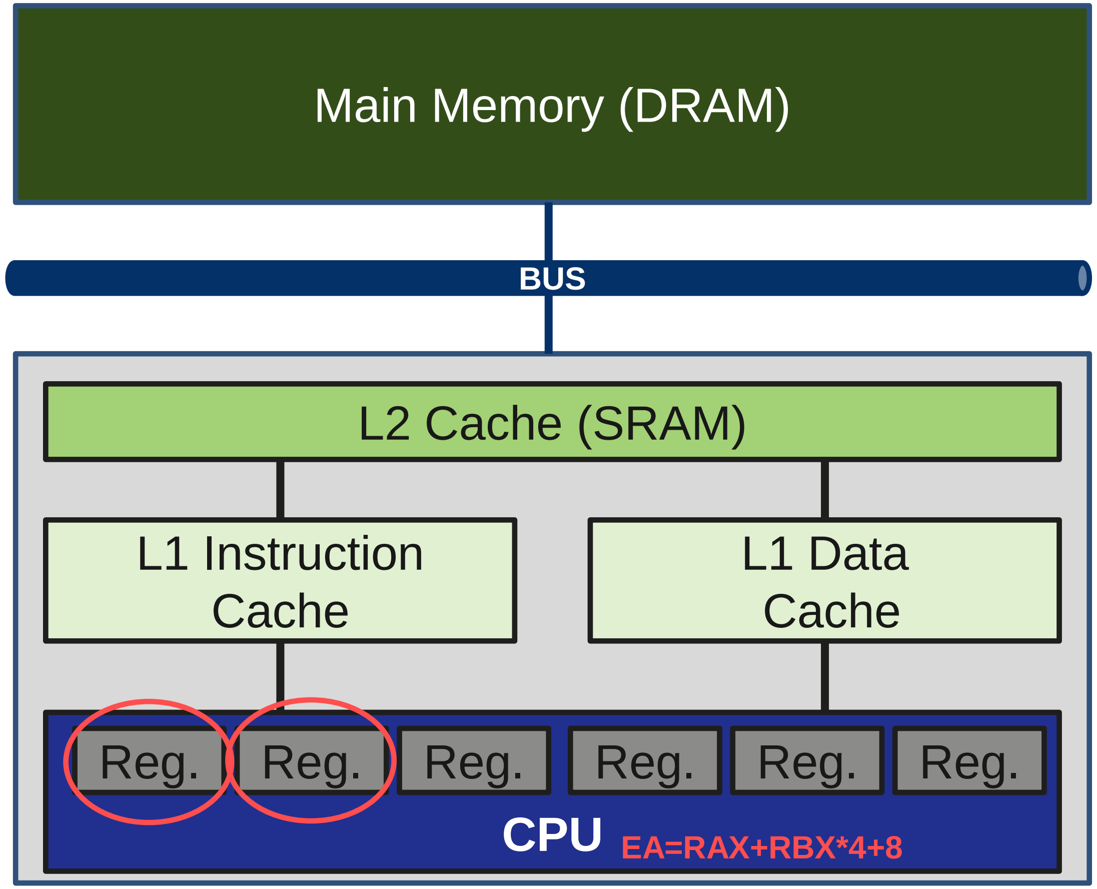
© 2020 Renesas Electronics Corporation. All rights reserved.

COMPLEX INSTRUCTION EXAMPLE

```
ADD [RAX+RBX*4+8], ECX
```

Steps to execute this command:

1. Fetch instruction from memory (or cache)
2. Generate target address

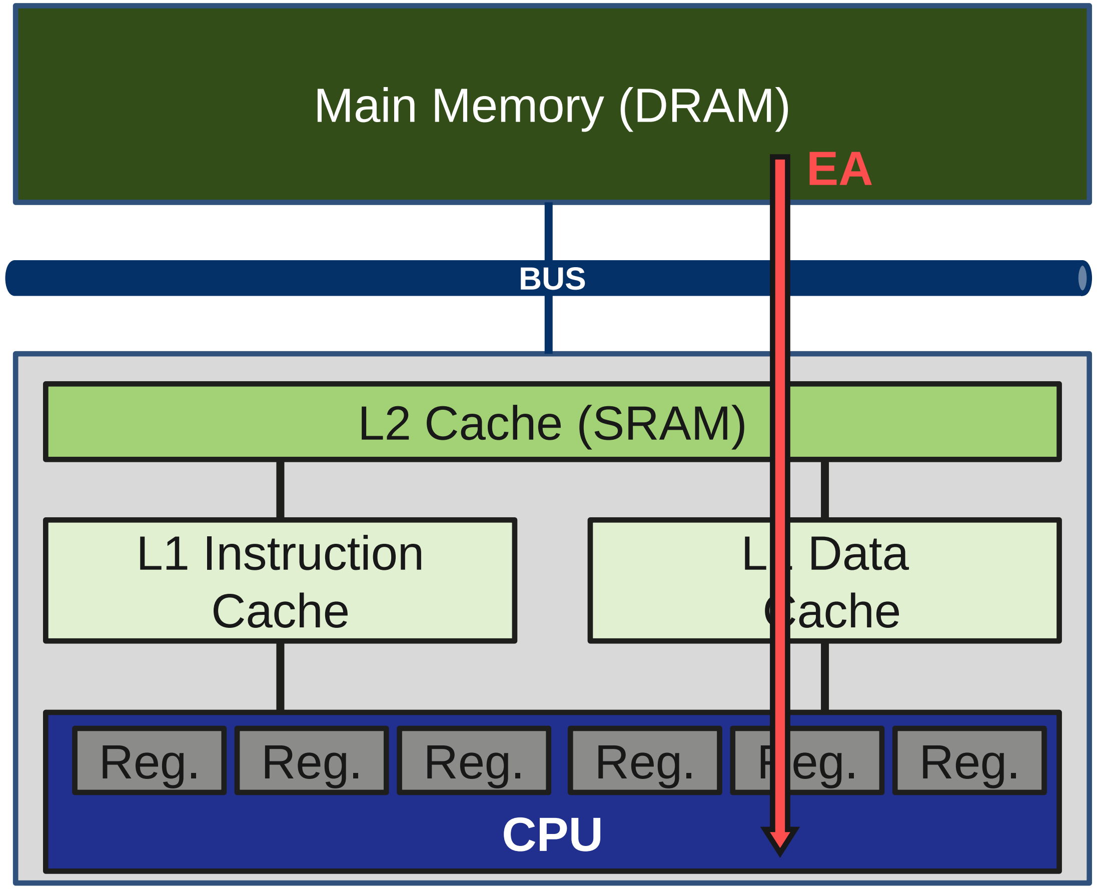


COMPLEX INSTRUCTION EXAMPLE

```
ADD [RAX+RBX*4+8], ECX
```

Steps to execute this command:

1. Fetch instruction from memory (or cache)
2. Generate target address
3. Load data from memory

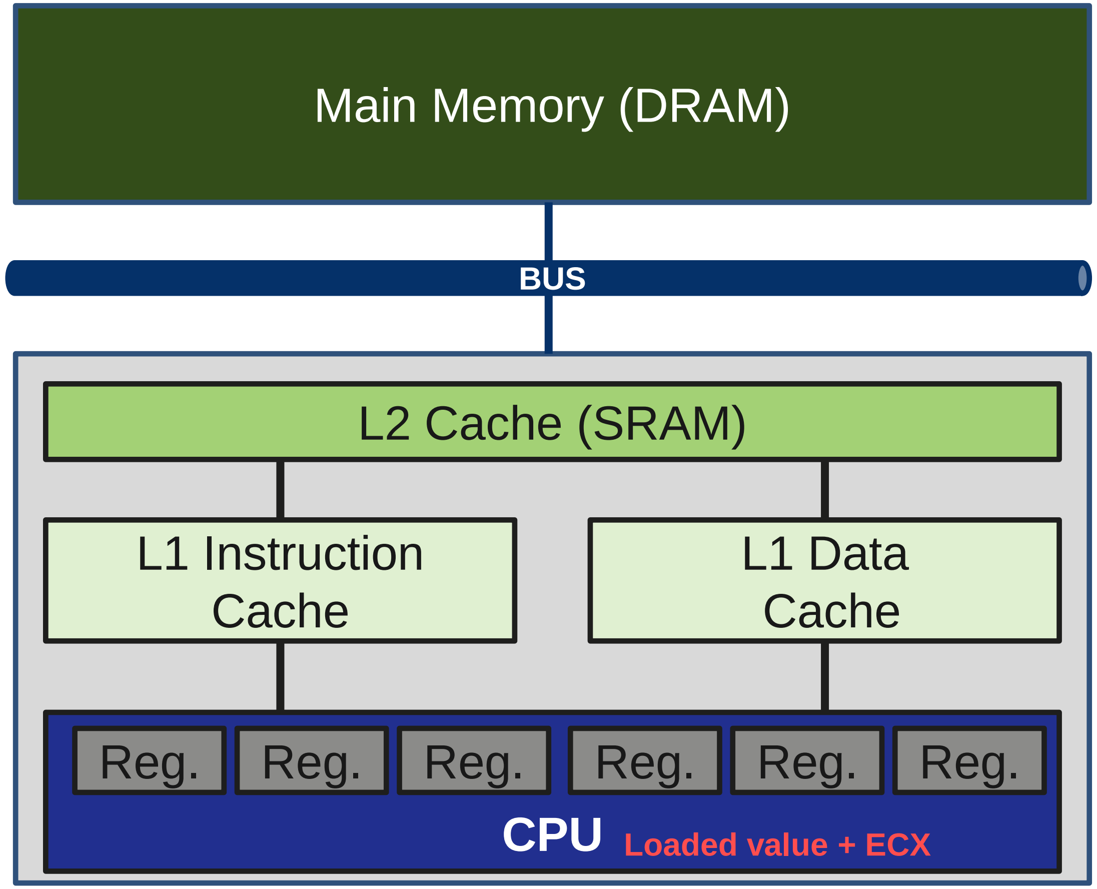


COMPLEX INSTRUCTION EXAMPLE

```
ADD [RAX+RBX*4+8], ECX
```

Steps to execute this command:

1. Fetch instruction from memory (or cache)
2. Generate target address
3. Load data from memory
4. Add Operation

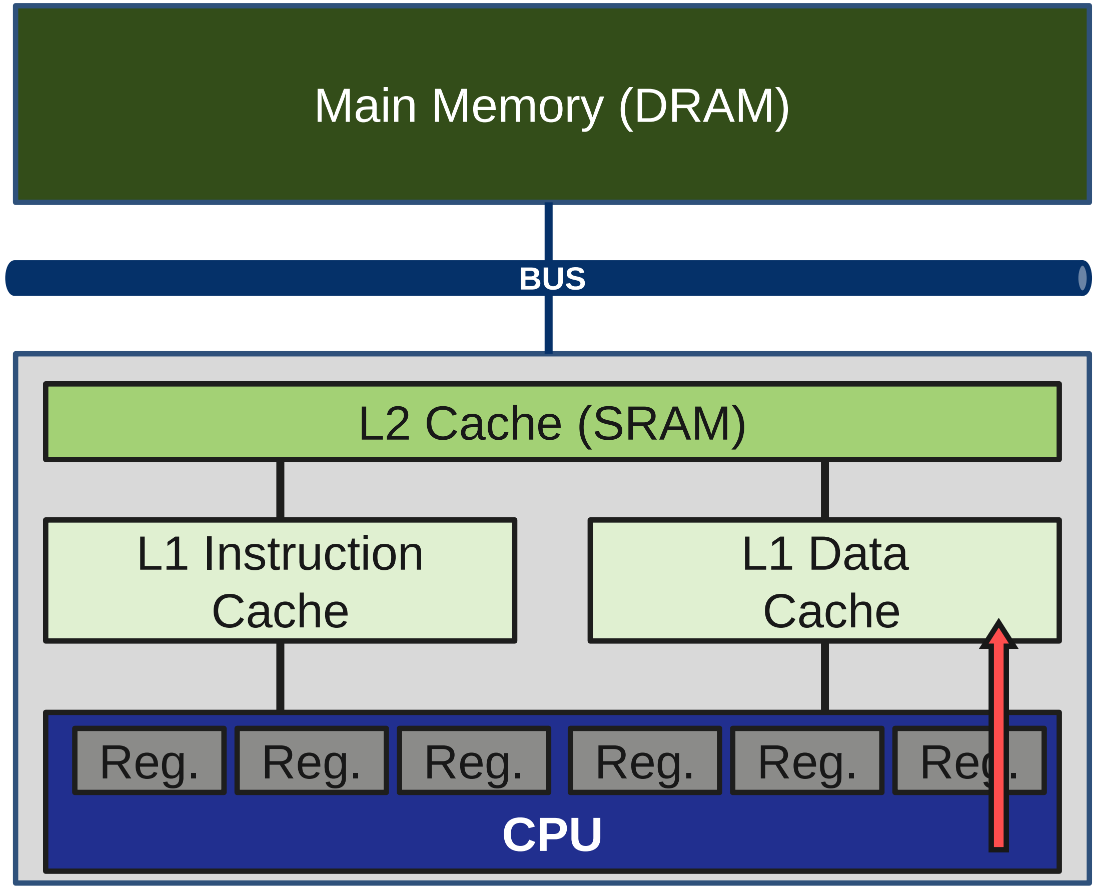


COMPLEX INSTRUCTION EXAMPLE

```
ADD [RAX+RBX*4+8], ECX
```

Steps to execute this command:

1. Fetch instruction from memory (or cache)
2. Generate target address
3. Load data from memory
4. Add Operation
5. Store results



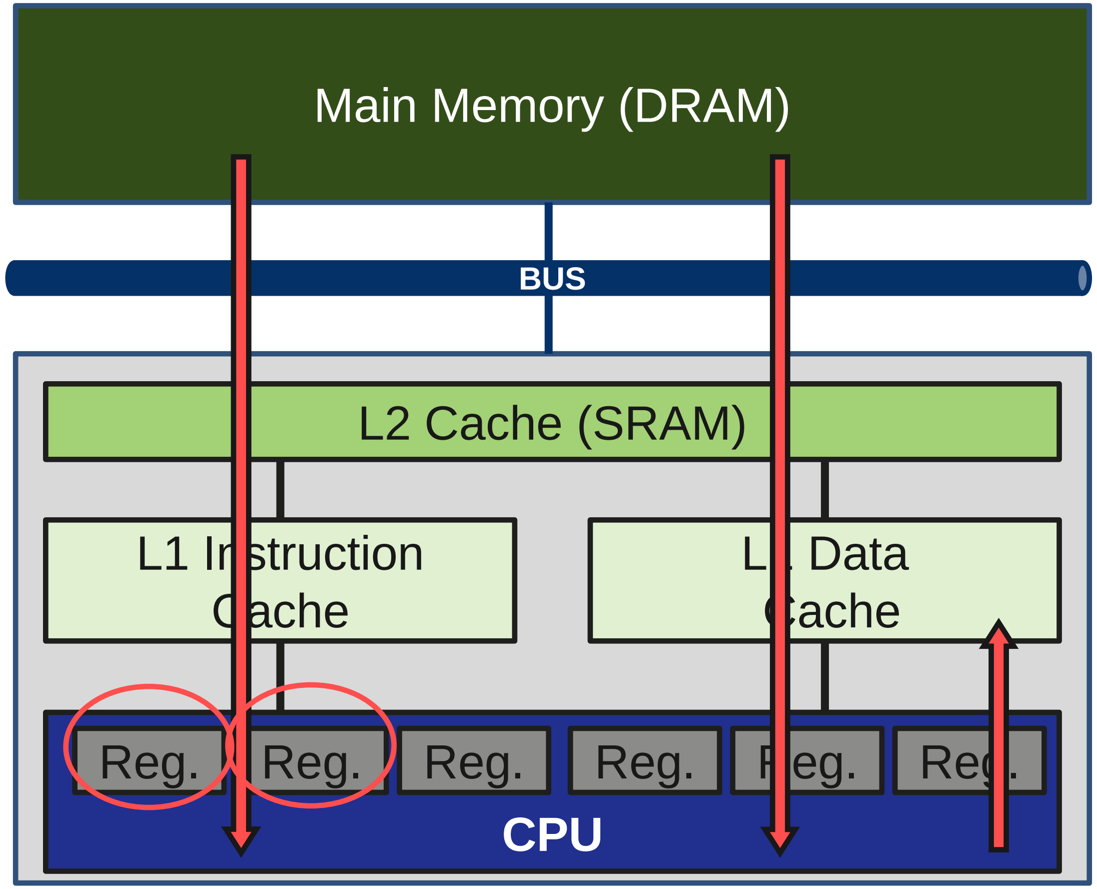
COMPLEX INSTRUCTION EXAMPLE

```
ADD [RAX+RBX*4+8], ECX
```

Steps to execute this command:

1. Fetch instruction from memory (or cache)
2. Generate target address
3. Load data from memory
4. Add Operation
5. Store results

A single instruction could load/store from/to memory and also perform computations! ➔ *complex*



RESEARCH PRECEDING THE RISC PARADIGM SHIFT

- Careful examination of the execution of actual programs concluded that:
  - Often, complex instructions were not used and the equivalent effect was obtained by a sequence of simpler instructions.
  - The inclusion of a single complex instruction in the instruction set could impact overall performance by imposing a lower clock rate.
  - Since the instructions did not have a regular execution sequence (i.e. each one had its execution determined by its microcode) the implementation of a pipeline was close to impossible.

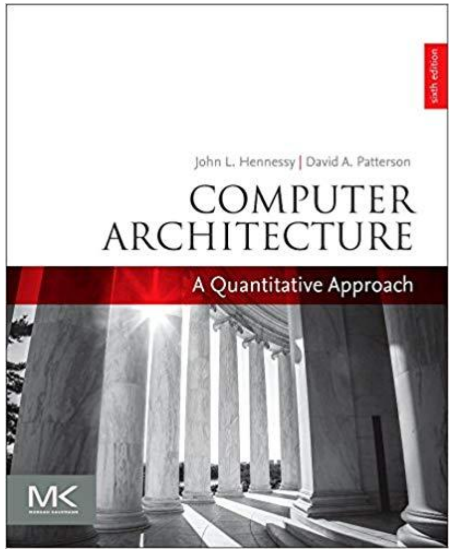
RESEARCH PRECEDING THE RISC PARADIGM SHIFT

What is the final goal?

- From the user's perspective, the goal is **performance of the application programs**

- **RISC thesis**: a processor whose instruction set is made of simple instructions with a regular execution, allowing the implementation
  of a pipeline, will have a higher performance than a CISC
  (Complex Instruction Set Computer)

recommended reading:
Computer Architecture: A Quantitative Approach 6th Edition
by John L. Hennessy, David A. Patterson



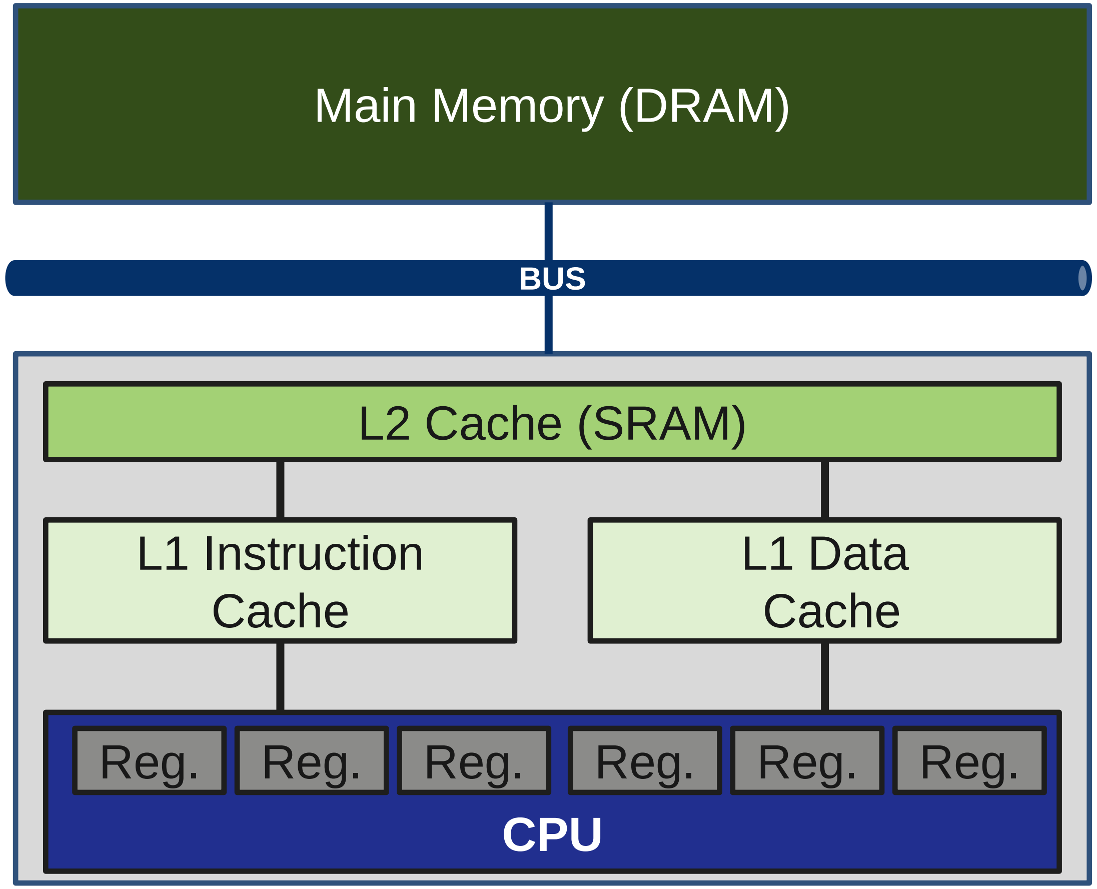
RISC ARCHITECTURAL FEATURES

Features that characterize the RISC paradigm are:

- An instruction set with a reduced number of very simple instructions.
- LD/ST architecture, meaning that only two instructions (and their variants) access memory: LD (load) reads data from memory, and ST (store) saves data to memory.
- All other instructions operate on registers. There is a large number of general-purpose registers, avoiding access to memory.
- All instructions follow the same logical execution sequence. Hence, a pipelined architecture can be implemented, significantly improving performance. Typically 1 instruction is executed every clock cycle.

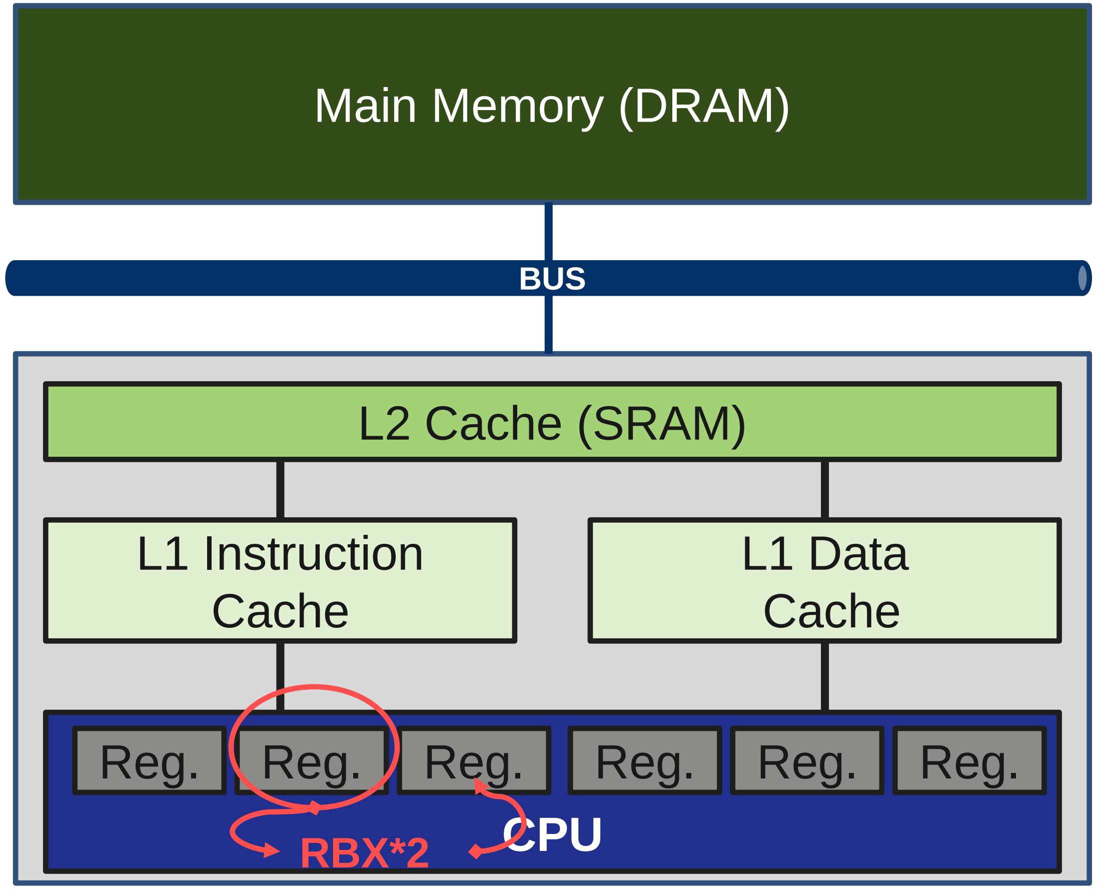
CISC TO RISC

```
CISC:
       ADD [RAX+RBX*4+8], ECX

RISC:
       SLL  R1, RBX, 2    # R1=RBX*4
       ADD  R1, R1, RAX   # R1=RAX+RBX*4
       ADDI R1, R1, 8     # R1=RAX+RBX*4+8

       LW   R2, 0(R1)     # R2=Memory[R1]
       ADD  R2, R2, ECX   # R2=R2+ECX
       SW   R2, 0(R1)     # Memory[R1]=R2
```



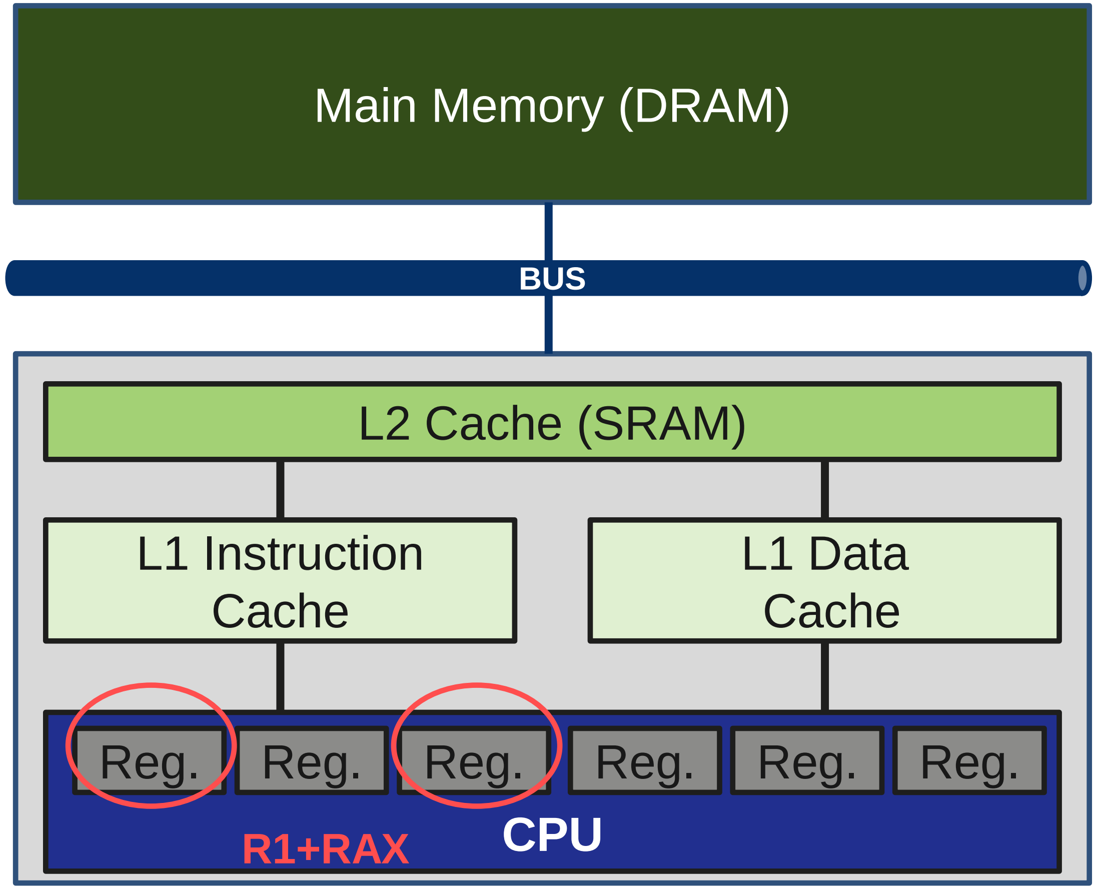
CISC TO RISC

```
CISC:
        ADD [RAX+RBX*4+8], ECX

RISC:
        SLL  R1, RBX, 2   # R1=RBX*4
```



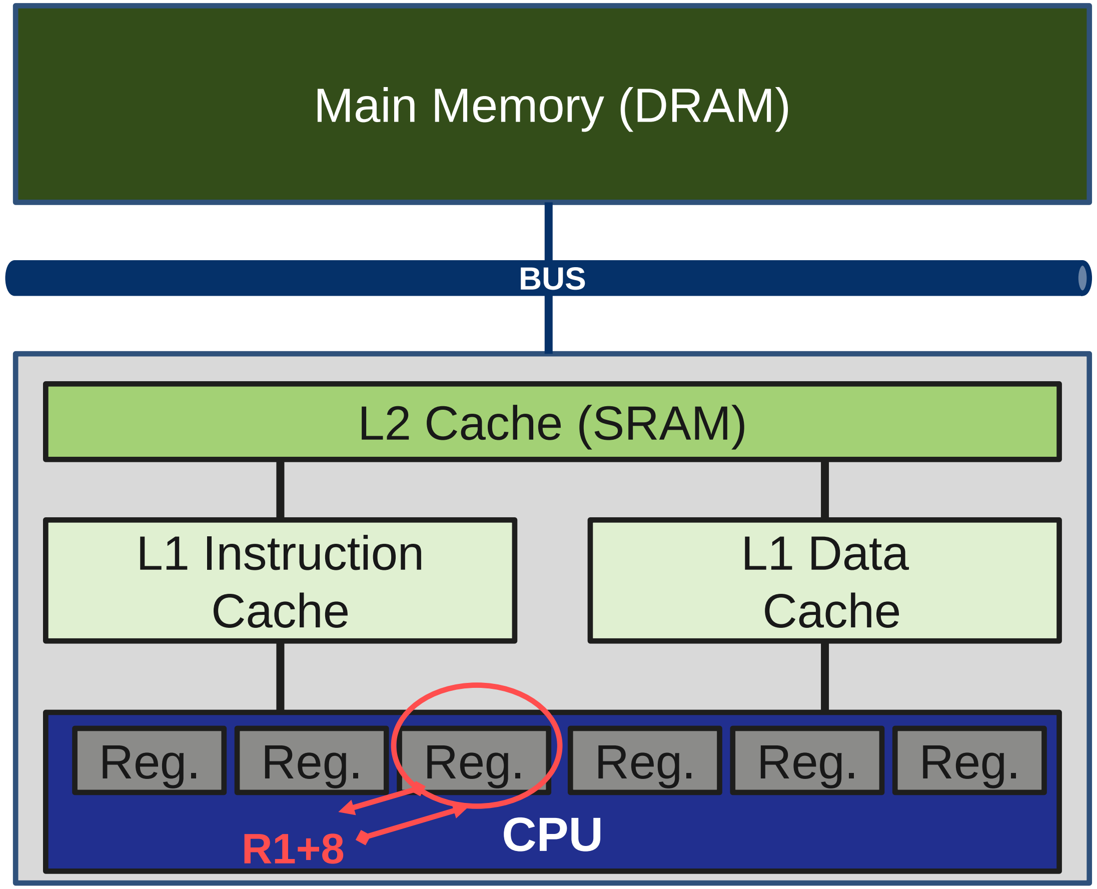
CISC TO RISC

```
CISC:
        ADD [RAX+RBX*4+8], ECX

RISC:
        SLL  R1, RBX, 2    # R1=RBX*4
        ADD  R1, R1, RAX   # R1=RAX+RBX*4
```



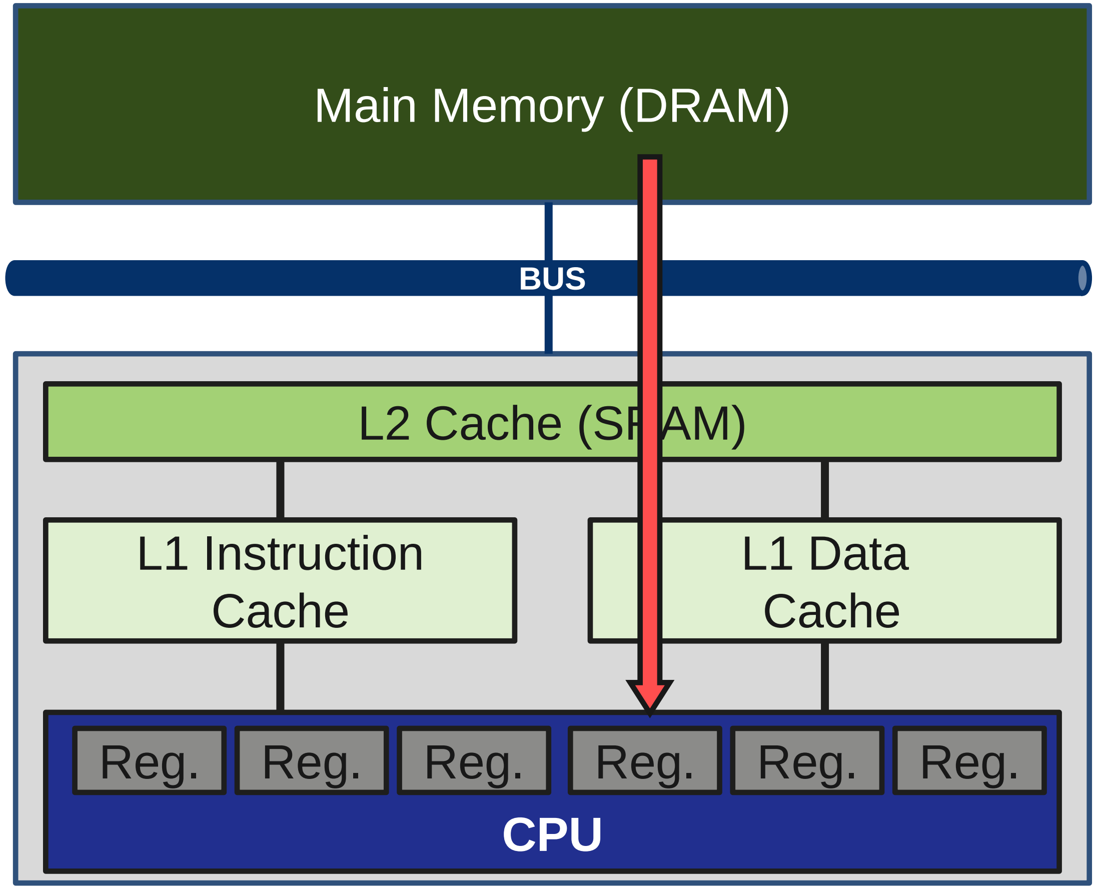
CISC TO RISC

```
CISC:
       ADD [RAX+RBX*4+8], ECX

RISC:
       SLL  R1, RBX, 2    # R1=RBX*4
       ADD  R1, R1, RAX   # R1=RAX+RBX*4
       ADDI R1, R1, 8     # R1=RAX+RBX*4+8
```



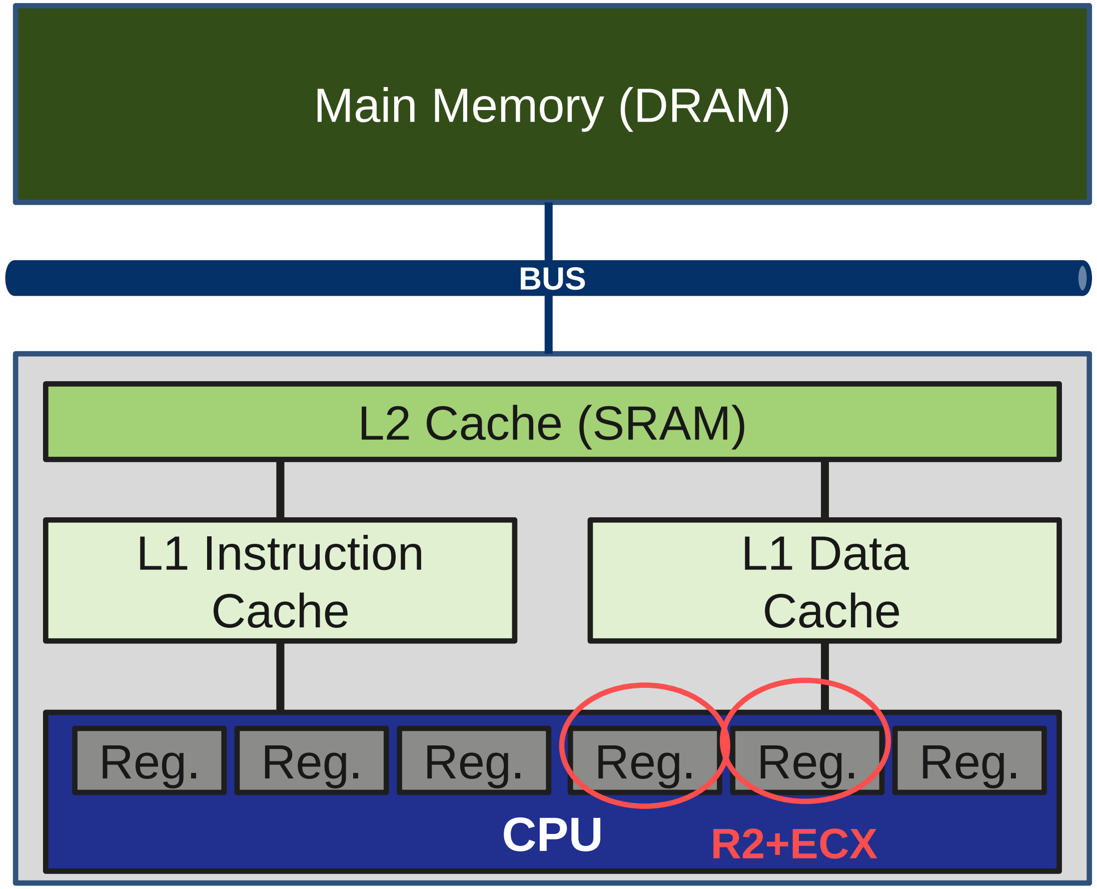
CISC TO RISC

```
CISC:
        ADD [RAX+RBX*4+8], ECX

RISC:
        SLL  R1, RBX, 2    # R1=RBX*4
        ADD  R1, R1, RAX   # R1=RAX+RBX*4
        ADDI R1, R1, 8     # R1=RAX+RBX*4+8

        LW   R2, 0(R1)     # R2=Memory[R1]
```



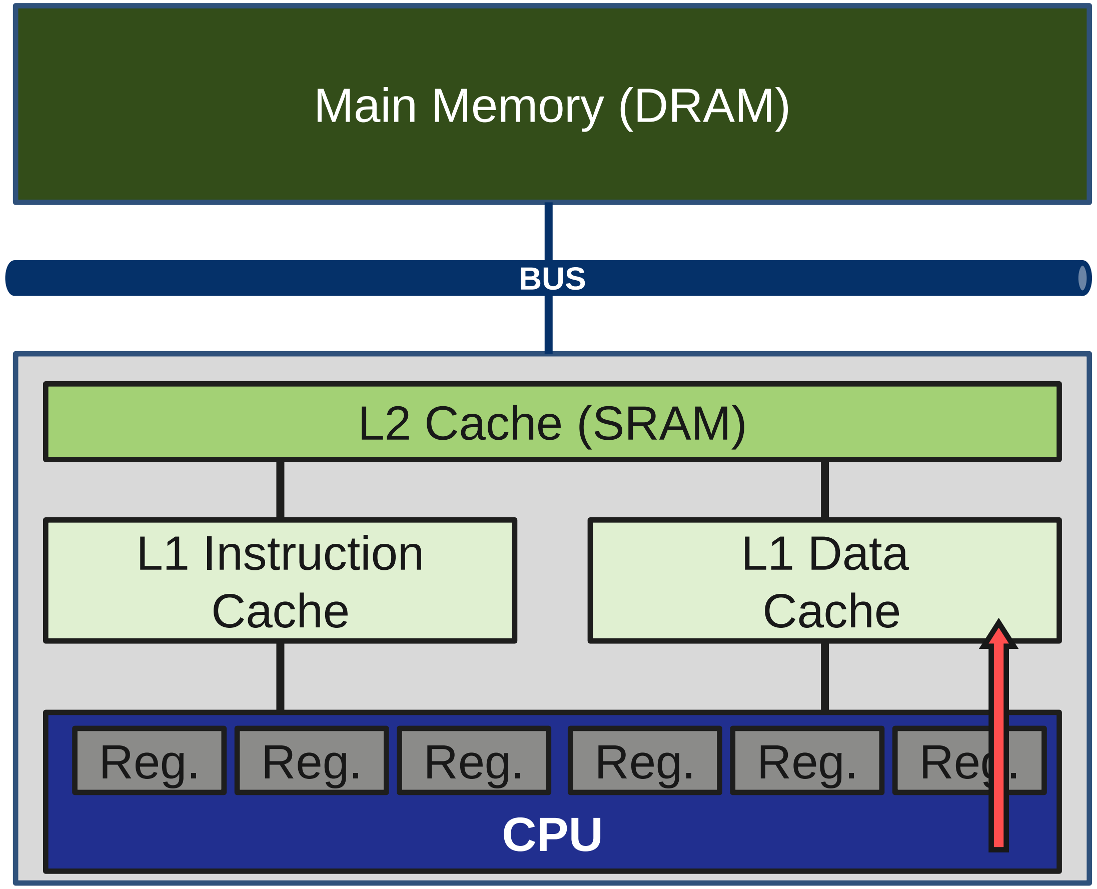
CISC TO RISC

```
CISC:
        ADD [RAX+RBX*4+8], ECX

RISC:
        SLL  R1, RBX, 2    # R1=RBX*4
        ADD  R1, R1, RAX   # R1=RAX+RBX*4
        ADDI R1, R1, 8     # R1=RAX+RBX*4+8

        LW   R2, 0(R1)     # R2=Memory[R1]
        ADD  R2, R2, ECX   # R2=R2+ECX
```



CISC TO RISC

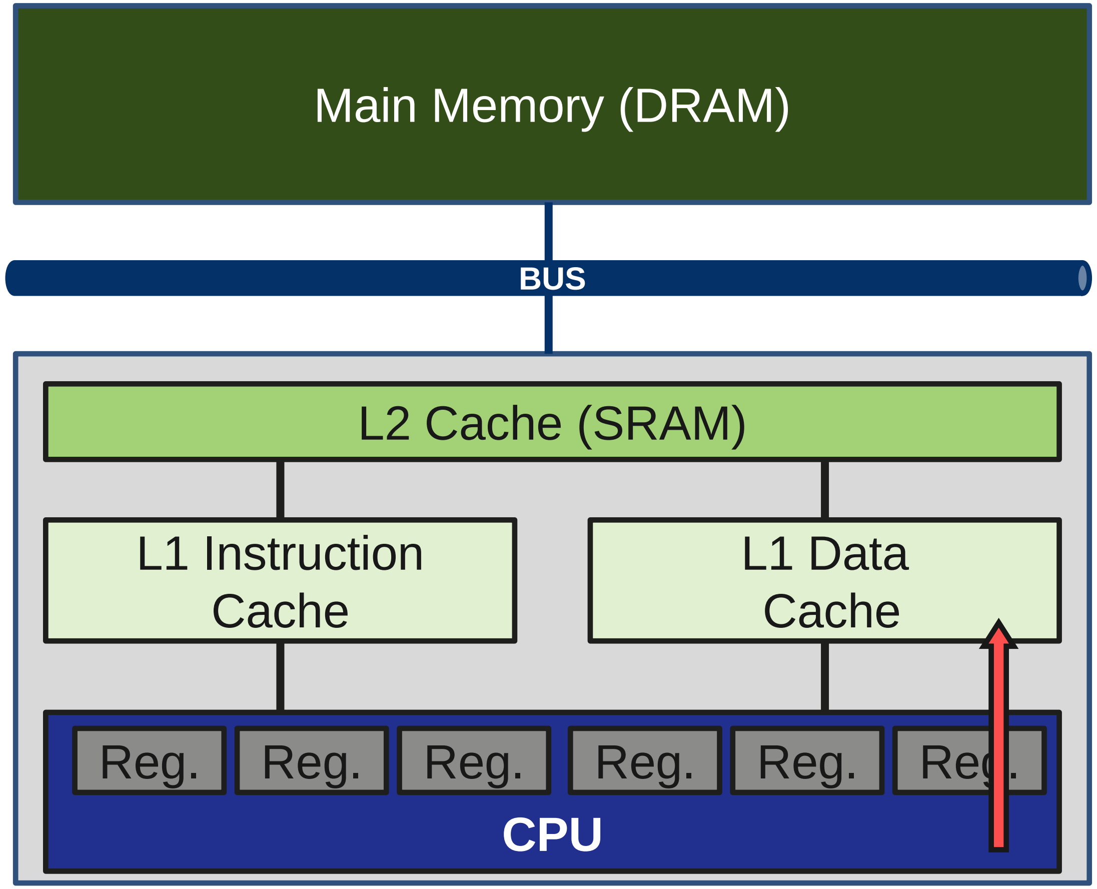
```
CISC:
       ADD [RAX+RBX*4+8], ECX

RISC:
       SLL  R1, RBX, 2    # R1=RBX*4
       ADD  R1, R1, RAX   # R1=RAX+RBX*4
       ADDI R1, R1, 8     # R1=RAX+RBX*4+8

       LW   R2, 0(R1)     # R2=Memory[R1]
       ADD  R2, R2, ECX   # R2=R2+ECX
       SW   R2, 0(R1)     # Memory[R1]=R2
```



CISC TO RISC

```
CISC:
        ADD [RAX+RBX*4+8], ECX

RISC:
        SLL  R1, RBX, 2    # R1=RBX*4
        ADD  R1, R1, RAX   # R1=RAX+RBX*4
        ADDI R1, R1, 8     # R1=RAX+RBX*4+8

        LW   R2, 0(R1)     # R2=Memory[R1]
        ADD  R2, R2, ECX   # R2=R2+ECX
        SW   R2, 0(R1)     # Memory[R1]=R2
```

A single instruction performs only one simple operation (either load/store or computation) ➔ ***simple and regular***



PIPELINING

Requirements for the implementation of a Pipeline in a processor:

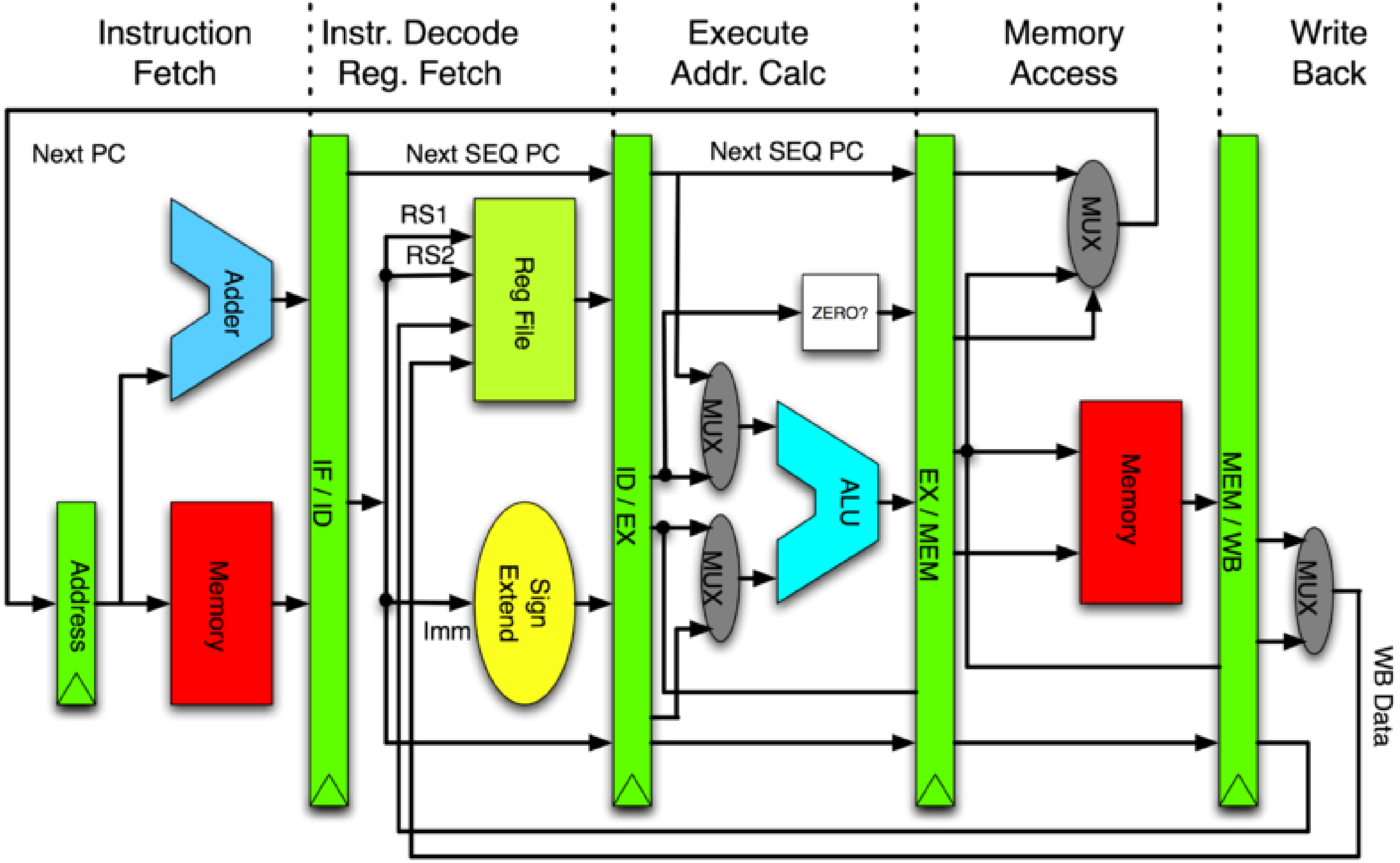
- The instruction set must be designed so that all instructions have the same execution steps
- Instructions must be regular with respect to:
  - Size
  - Addressing modes
  - Decoding
  - Operands

PIPELINING – OVERVIEW

Example of a Pipeline with 5 stages:

1. Fetch
2. Decode and read operands in registers
3. Use ALU to execute instruction or to calculate memory address
4. Read or Write to memory
5. Write result back to Register file

Remark: not all instructions use step 4 and 5



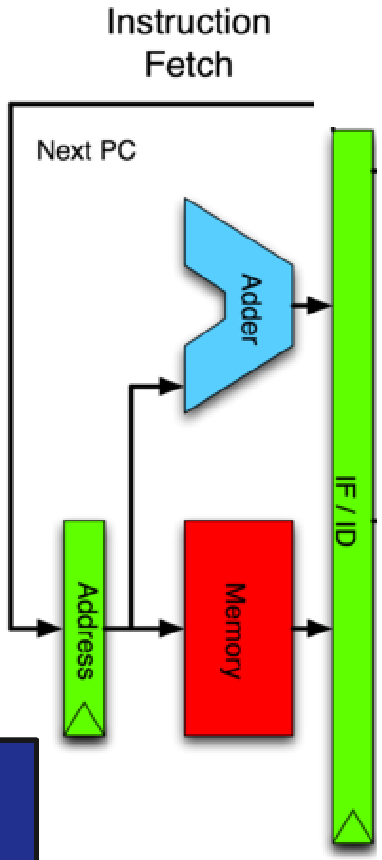
source: wikimedia.org (CC)

PIPELINING – STAGE 1: INSTRUCTION FETCH

1. The Program Counter (PC) is a register that holds the address of the current instruction
2. PC is sent to instruction memory
3. Instruction (bits) is loaded from instruction memory (icache or DRAM).
4. The Adder sets the PC to the next command (typically PC+4)

Outputs:

- ***Instruction bits***
- ***Next instruction (PC+4)***



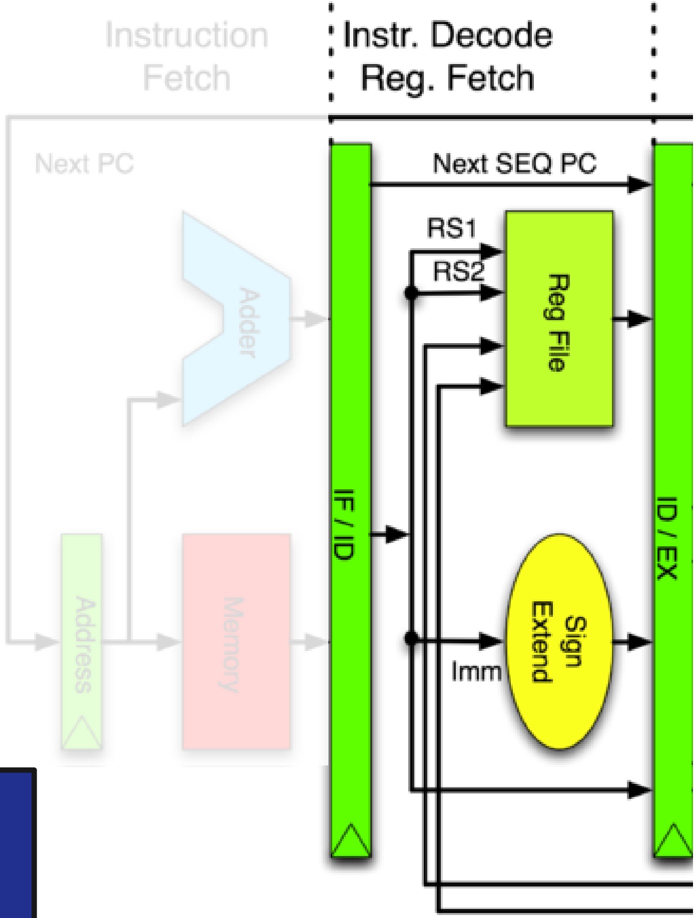
source: wikimedia.org (CC)

PIPELINING – STAGE 2: INSTRUCTION DECODE / REGISTER FETCH

1. Instruction read from IF/ID pipeline register.
2. Instruction fields extracted
   - Opcode, registers, immediate, etc.
3. Decodes the opcode and generates control signals for later stages (EX, MEM, WB).
4. The Reg. File reads the values of the source registers (RS1, RS2).
5. If the instruction contains an immediate value, it is sign-extended to the processor word size.

Outputs:

- ***Register values RS1, RS2***
- ***Sign-extended immediate***



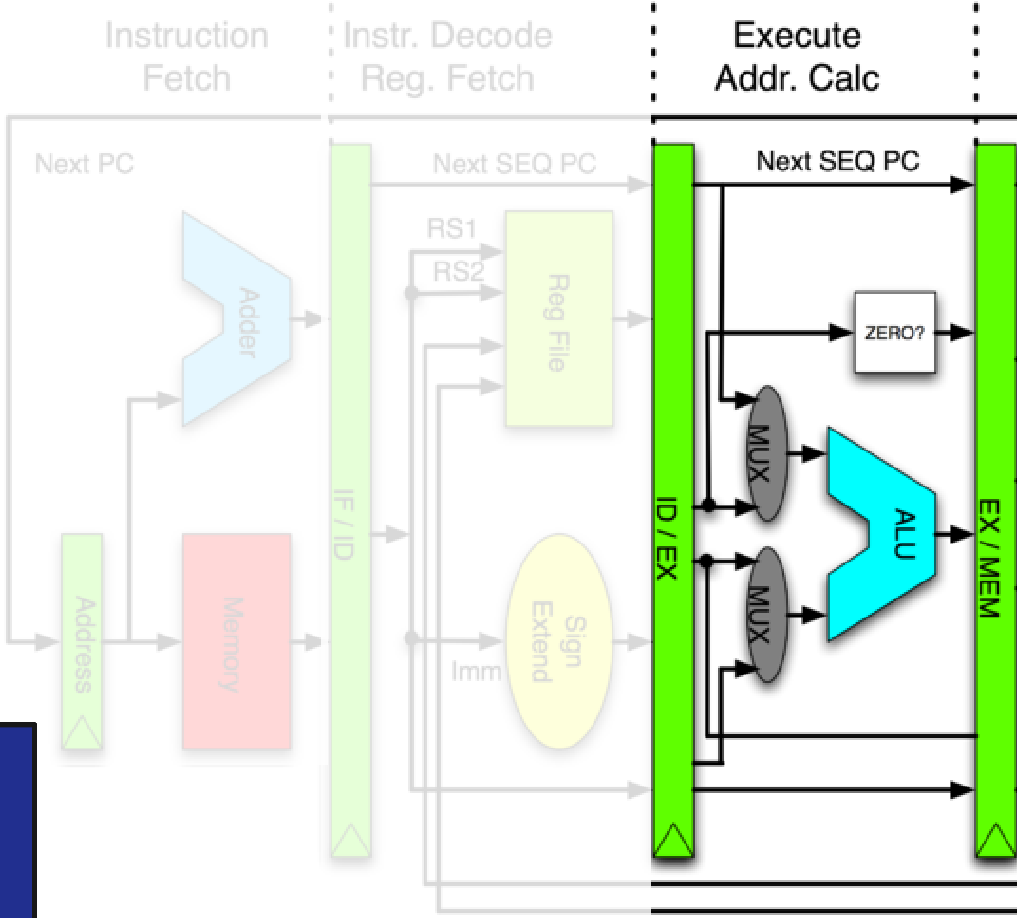
source: wikimedia.org (CC)

PIPELINING – STAGE 3: EXECUTE / ADDRESS CALCULATION

1. Instruction inputs read from the ID/EX register
   - RS1, RS2, Immediate
2. Operand selection using multiplexers
   - Select between RS2 and immediate
3. The ALU performs the required operation
   - Arithmetic (ADD, SUB, …)
   - Logical (AND, OR, …)
   - Comparison (for branches)
4. Zero signal for branch instructions

Outputs:

- ***ALU result / Effective Address***
- ***Zero flag***
- ***Branch target address***



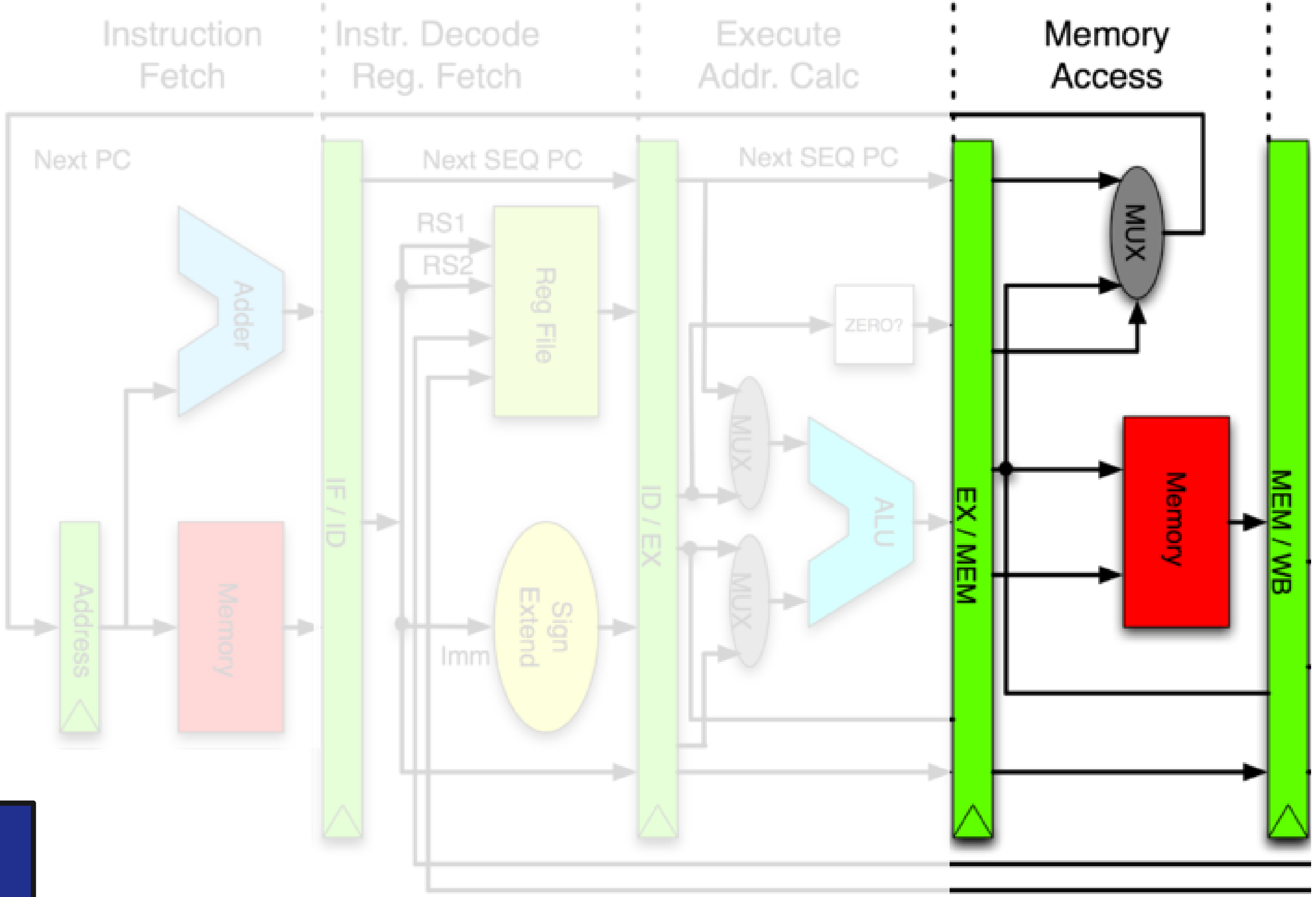
source: wikimedia.org (CC)

PIPELINING – STAGE 4: MEMORY ACCESS

1. Inputs read from the EX/MEM register
   - Effective Address, Register values, Control signals
2. For Load (LW) instructions
   - Data memory is read using the effective address
   - Loaded data is forwarded to the MEM/WB register
3. For Store (SW) instructions:
   - Value of source register is written at the effective address
4. For arithmetic/logic instructions
   - No memory access is performed
   - ALU result simply passes through

Outputs:

- *__Data loaded from memory (for LW)__*
- *__ALU result (for arithmetic operation)__*
- *__Destination register identifier__*



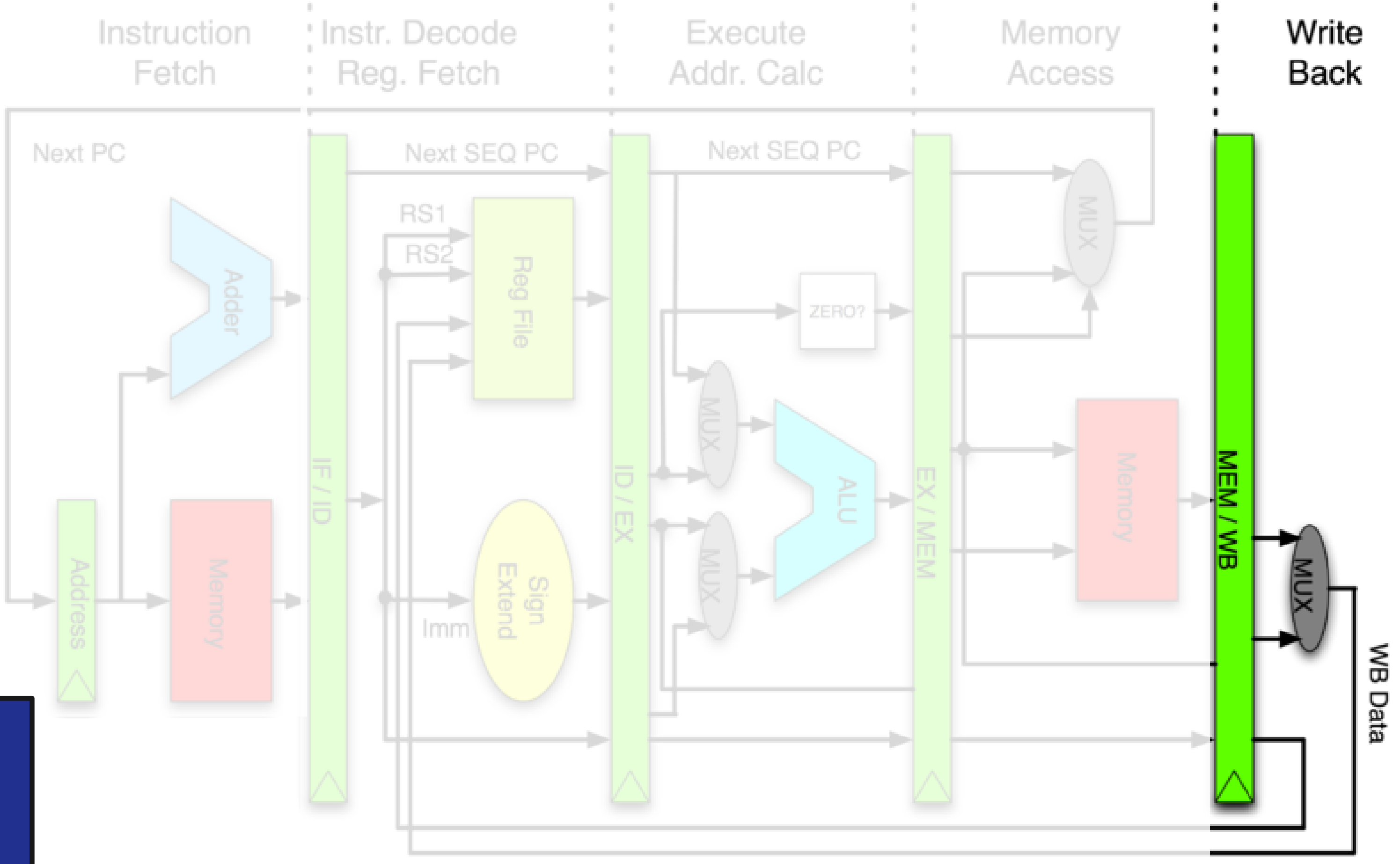
source: wikimedia.org (CC)

PIPELINING – STAGE 5: WRITE BACK

1. Inputs read from the MEM/WB register:
   - Memory read data, ALU result, Destination Register Identifier, Control Signals
2. Multiplexer selects the value to write back
   - Memory data (for load instructions)
   - ALU result (for arithmetic/logic instructions)
3. If the instruction requires register write
   - Value is written into the destination register.
4. For store (SW) and branch instructions:
   - No register write is performed.

Outputs:

- ***Updated register file (architectural state updated)***



source: wikimedia.org (CC)

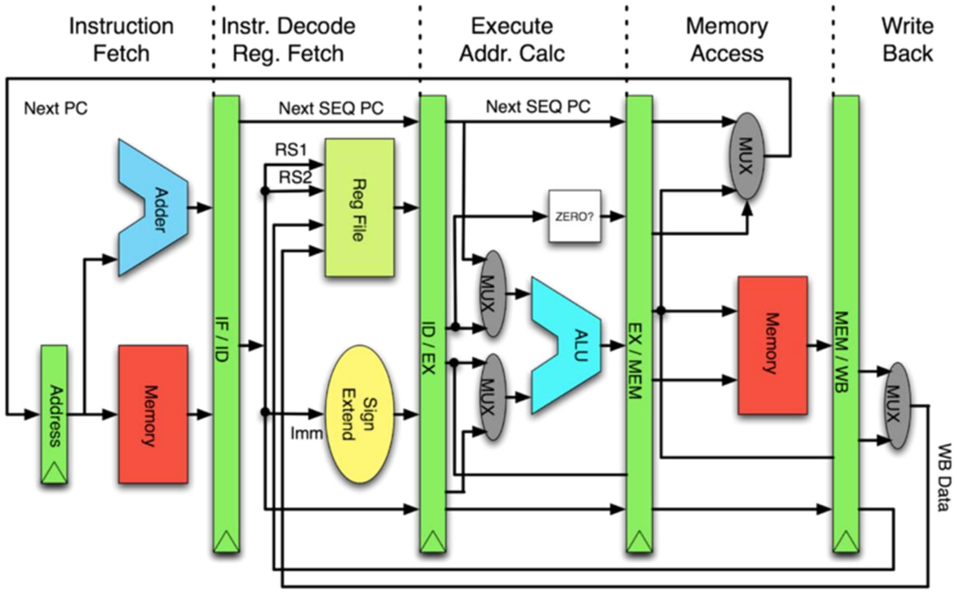
BENEFITS OF PIPELINING

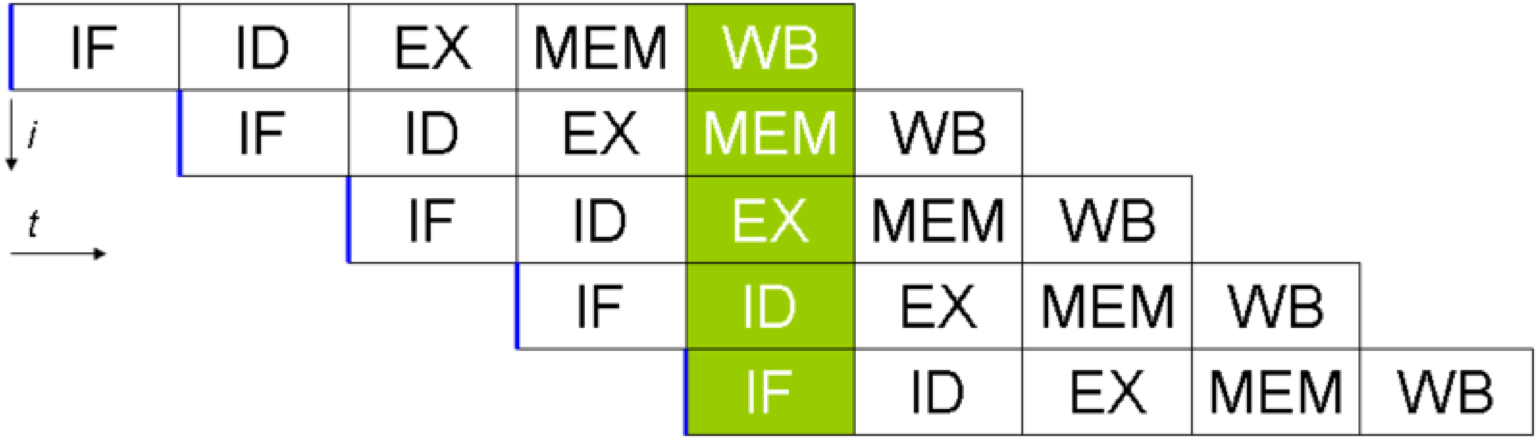
For the previous example of a 5 stage pipeline, the

execution over time is presented in this diagram.

Note that although each instruction takes 5 clock cycles to execute, due to pipelining, after filling the pipeline, on every clock another instruction finished its execution.

Effectively, the number of instructions executed per second is the same as the clock rate.





IF	ID	EX	MEM	WB				
	IF	ID	EX	MEM	WB			
		IF	ID	EX	MEM	WB		
			IF	ID	EX	MEM	WB	
				IF	ID	EX	MEM	WB

source: wikimedia.org (CC)

LIMITATIONS OF PIPELINING

Consequence:
the average number of instructions executed per second is lower than the clock rate.

- Data Hazard:

  if an instruction requires as input a value produced by the previous instruction, this value may not be available at stage 2 (decode and fetch operands) since the previous instruction only saves its results at stage 5.

  Solutions: stall the pipeline (instruction must wait for result of previous)

  OR use a technique called bypassing to forward the result of the previous instruction at stage 3 of its execution

  OR reorder instructions to avoid this dependency

- Control Hazard:

  a change of control flow (e.g. a branch instruction) causes the pipeline to be flushed, meaning that following instructions that already were fetched and decoded will be discarded.

DATA HAZARDS

- **Read After Write (RAW) – True dependency**

  An instruction needs a value that a previous instruction produces.

```
ADD R1, R2, R3
SUB R4, R1, R5
```

DATA HAZARDS

- **Read After Write (RAW) – True dependency**

  An instruction needs a value that a previous instruction produces.

```
ADD R1, R2, R3
SUB R4, R1, R5
```

The value of R1 is written to the register file in Stage 5.

SUB needs the value in the execution stage (Stage 3).

Without forwarding, SUB must stall until ADD completes WB (2 stall cycles).

DATA HAZARDS

- **Read After Write (RAW) – True dependency**

  An instruction needs a value that a previous instruction produces.

```
ADD R1, R2, R3

SUB R4, R1, R5
```

- **Write After Read (WAR) – Anti-dependency**

  A later instruction writes to a register that a previous instruction still needs to read.

```
SUB R4, R1, R5
ADD R1, R2, R3
```

DATA HAZARDS

- **Read After Write (RAW) – True dependency**

  An instruction needs a value that a previous instruction produces.

```
ADD R1, R2, R3

SUB R4, R1, R5
```

- **Write After Read (WAR) – Anti-dependency**

  A later instruction writes to a register that a previous instruction still needs to read.

```
SUB R4, R1, R5
ADD R1, R2, R3
```

- **Write After Write (WAW) – Output dependency**

  Two instructions write to the same register.

```
ADD R1, R2, R3
SUB R1, R4, R5
```

DATA HAZARDS

- **Read After Write (RAW) – True dependency**

  An instruction needs a value that a previous instruction produces.

```
ADD R1, R2, R3
SUB R4, R1, R5
```

- **Write After Read (WAR) – Anti-dependency**

  A later instruction writes to a register that a previous instruction still needs to read.

```
SUB R4, R1, R5
ADD R1, R2, R3
```

- **Write After Write (WAW) – Output dependency**

  Two instructions write to the same register.

```
ADD R1, R2, R3
SUB R1, R4, R5
```

In the classic 5-stage RISC pipeline, only RAW hazards are possible!

CONTROL HAZARD

- **The next instruction address depends on a branch/jump outcome, so the pipeline may fetch the wrong instructions.**

```
BEQ R1, R2, LABEL
ADD R3, R4, R5      ; may be wrong-path
SUB R6, R7, R8      ; may be wrong-path
LABEL:...
```

CONTROL HAZARD

- **The next instruction address depends on a branch/jump outcome, so the pipeline may fetch the wrong instructions.**

```
BEQ R1, R2, LABEL
ADD R3, R4, R5      ; may be wrong-path
SUB R6, R7, R8      ; may be wrong-path
LABEL:...
```

- **What happens in a 5-stage pipeline?**
  - The pipeline fetches BEQ and continues fetching the next sequential instructions (ADD, SUB).
  - The branch decision is only known later (typically in EX stage, Stage 3).
  - If the branch is taken, the already-fetched instructions after BEQ are wrong-path.
  - The pipeline must flush those wrong-path instructions and fetch from LABEL.

STRUCTURAL HAZARD

- **Two pipeline stages require the same hardware resource at the same time.**

Example (single memory for instructions and data):

```
LDR R1, [R2]      ; load data from memory
ADD R3, R4, R5
```

During execution:

- The Fetch stage needs memory to fetch the next instruction.
- The Execute stage of LDR needs memory to read data.

What happens in a simple pipeline?

- The Fetch stage tries to read the next instruction.
- The LDR instruction simultaneously accesses data memory.
- If there is only one memory, both cannot proceed.
- The processor inserts a stall cycle.

COMPARING RISC X CISC

RISC	CISC
Instruction set has a reduced number of instructions	Instruction set has a large number of instructions
Instructions are very simple	Instructions are complex, i.e. have a high semantic content
Large number of general purpose register	Small number of registers
Instruction codes have fixed size	Instructions are coded in a variable number of bytes
Instruction decoding is very simple and typically performed by a table in ROM	Instruction decoding is complex
Instruction execution is simple, typically requiring a single clock cycle	Instruction execution is complex and typically uses microprogramming, i.e. interpretation by microcode
Regular execution of the instructions	Execution steps varies from instruction to instruction
Execution time is regular, typically a single clock cycle	Execution time varies significantly among instructions
Architecture is prone to the use of a pipeline in the implementations, resulting in N times performance improvement (N is the number of pipeline stages).	Architecture is not prone to the use of Pipeline.

COMPARING RISC X CISC

When the same source program is compiled to a RISC processor and to a CISC processor:

- The number of machine instructions of the compiled program is typically larger on the RISC
- The size of the compiled program (measured in bytes) is typically larger on the RISC
- The performance (inverse of the time to execute the program) is typically better on the RISC

COMPARING RISC X CISC

From a HW perspective, the design of a RISC processor is significantly simpler than that of a CISC:

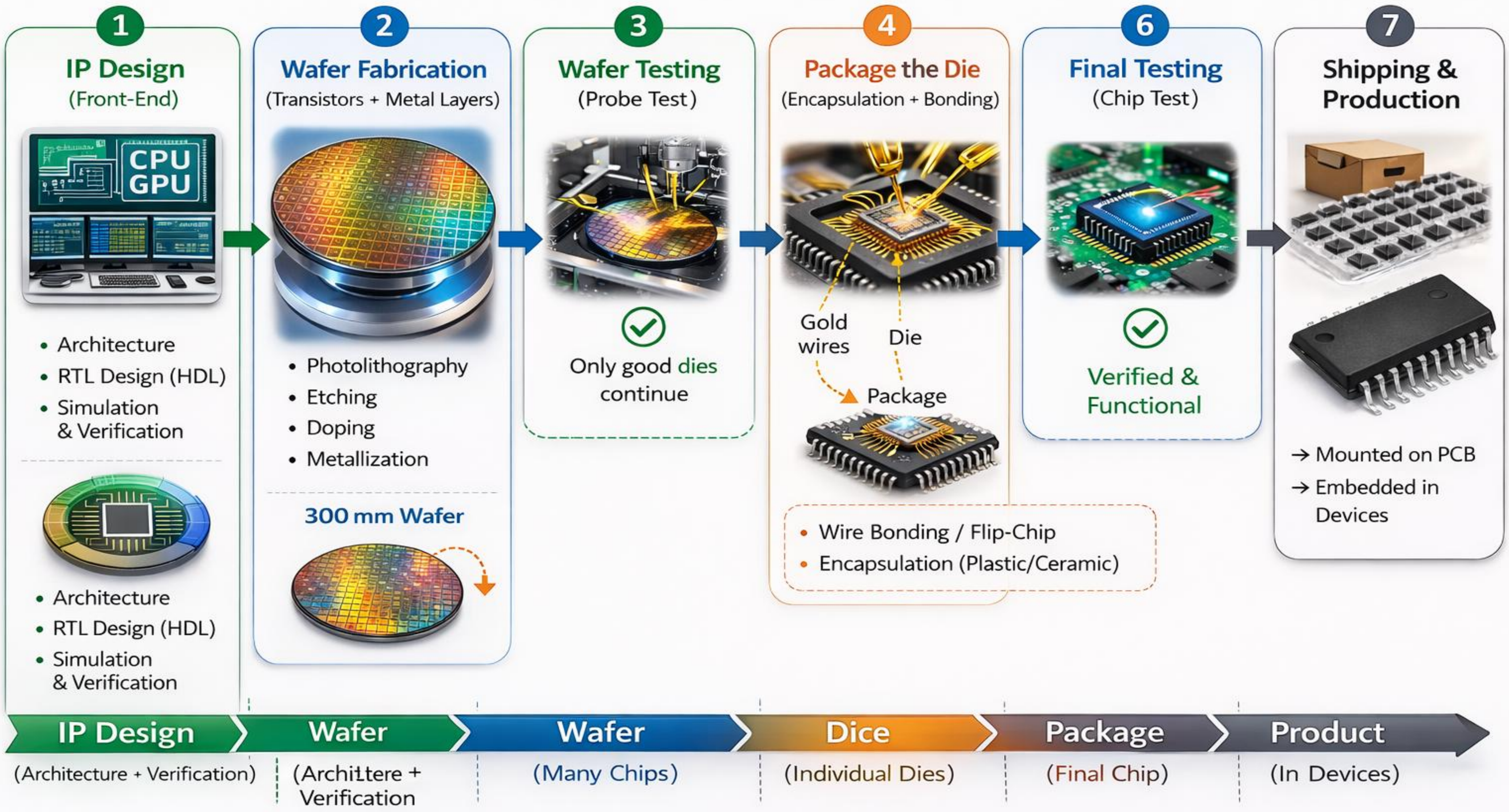
- Shorter design cycle,
- Lower number of transistors to implement,
- Less area on silicon die.

ECONOMICS OF INTEGRATED CIRCUITS MANUFACTURING

The cost of a chip is determined mainly by:

- Cost of the die (cost of manufacturing a wafer divided by the number of good dies per wafer)
- Testing costs (both for die testing and testing after packaging)
- Packaging costs
- Yield (percentage of functional chips).

From IP Design to Final Chip – Manufacturing Process



1 IP Design
(Front-End)

- Architecture
- RTL Design (HDL)
- Simulation & Verification

- Architecture
- RTL Design (HDL)
- Simulation & Verification

2 Wafer Fabrication
(Transistors + Metal Layers)

- Photolithography
- Etching
- Doping
- Metallization

300 mm Wafer

3 Wafer Testing
(Probe Test)

Only good dies continue

4 Package the Die
(Encapsulation + Bonding)

Gold wires

Die

Package

- Wire Bonding / Flip-Chip
- Encapsulation (Plastic/Ceramic)

6 Final Testing
(Chip Test)

Verified & Functional

7 Shipping & Production

→ Mounted on PCB

→ Embedded in Devices

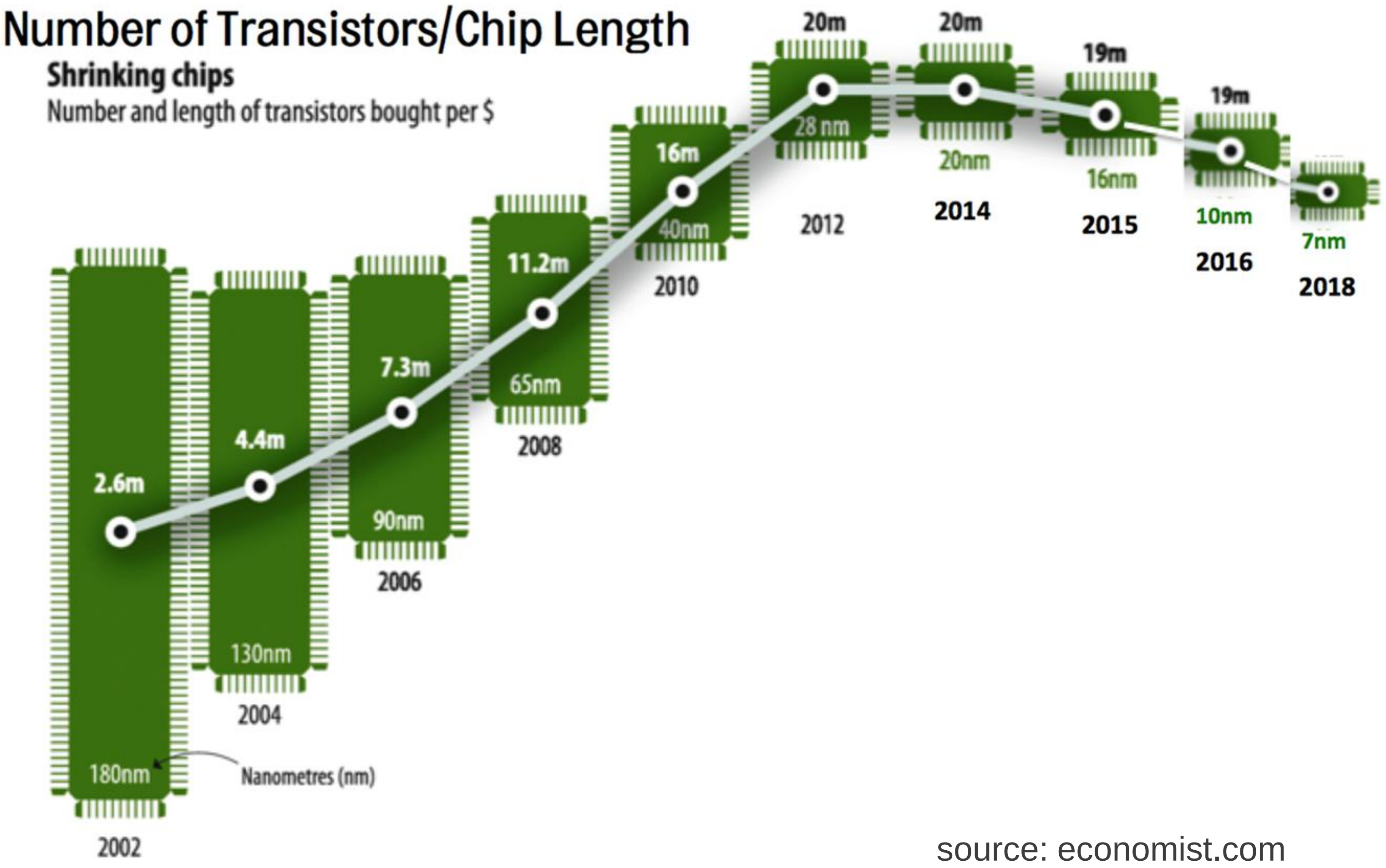
IP Design (Architecture + Verification) → Wafer (Archiltere + Verification → Wafer (Many Chips) → Dice (Individual Dies) → Package (Final Chip) → Product (In Devices)

ECONOMICS OF INTEGRATED CIRCUITS MANUFACTURING

Curve shows how many transistors can be purchased with one dollar for each manufacturing technology over time.

This graph represents the situation in 2018. At the time, the most cost-effective technology was 20 nm.

Over time, as the processes mature, transistors cost lowers. Until a given technology enters obsolescence.



source: economist.com

ECONOMICS OF INTEGRATED CIRCUITS MANUFACTURING

Key design decision:

If currently the “sweet spot” (best value for money) is a given manufacturing technology and a transistor count of T millions transistors, **what is the best use for this asset?**

1. CISC core + little Cache/Memory/Peripherals
2. RISC core + large amount of Cache/Memory/Peripherals

Most silicon vendors select the second alternative

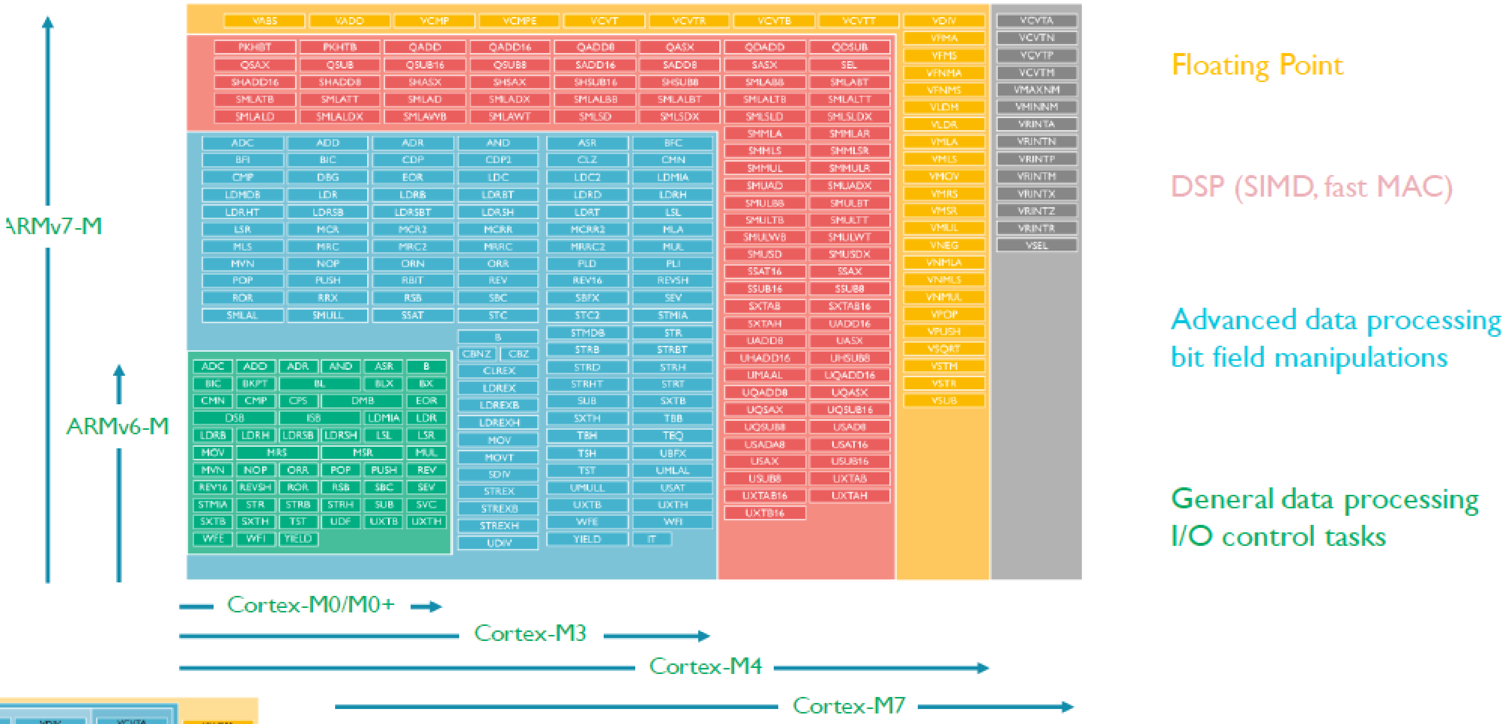
RISC VS CISC TODAY

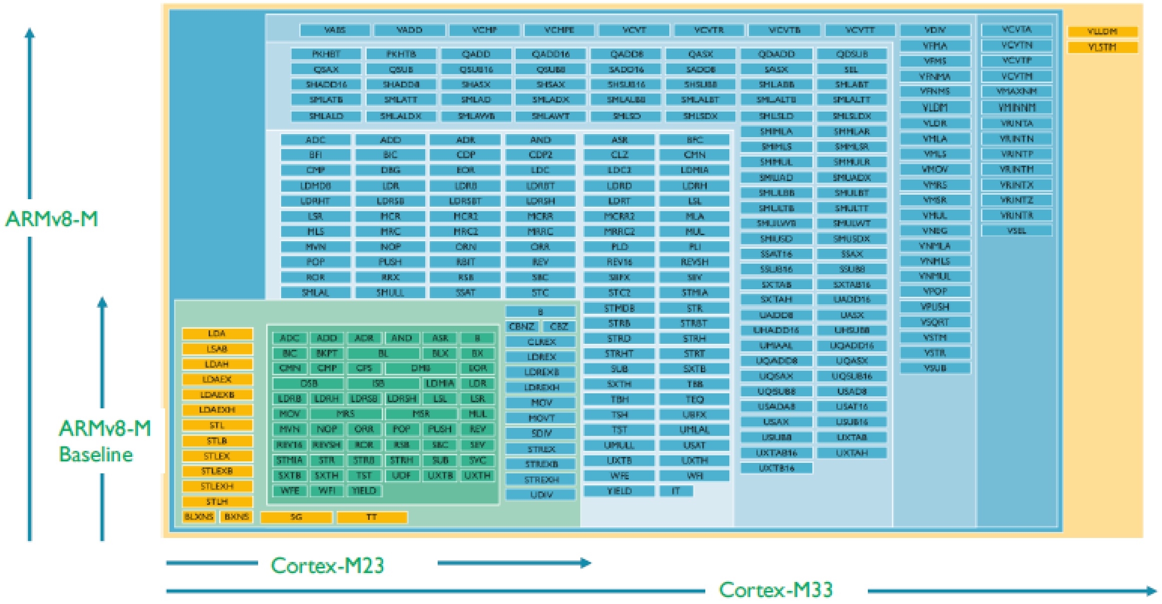
Currently, there is a trend to mix the best features of both paradigms. The convergence of RISC and CISC has not been named yet, but the characteristics of novel processors are:

- Large instruction set, but execution model is usually still load-store and regular
- Regular instructions, prone to pipelining
- Pipelines from 3 to 20 stages
- No microcode

RISC VS CISC TODAY

Instruction set of Cortex-M including Cortex-M23 and Cortex-M33





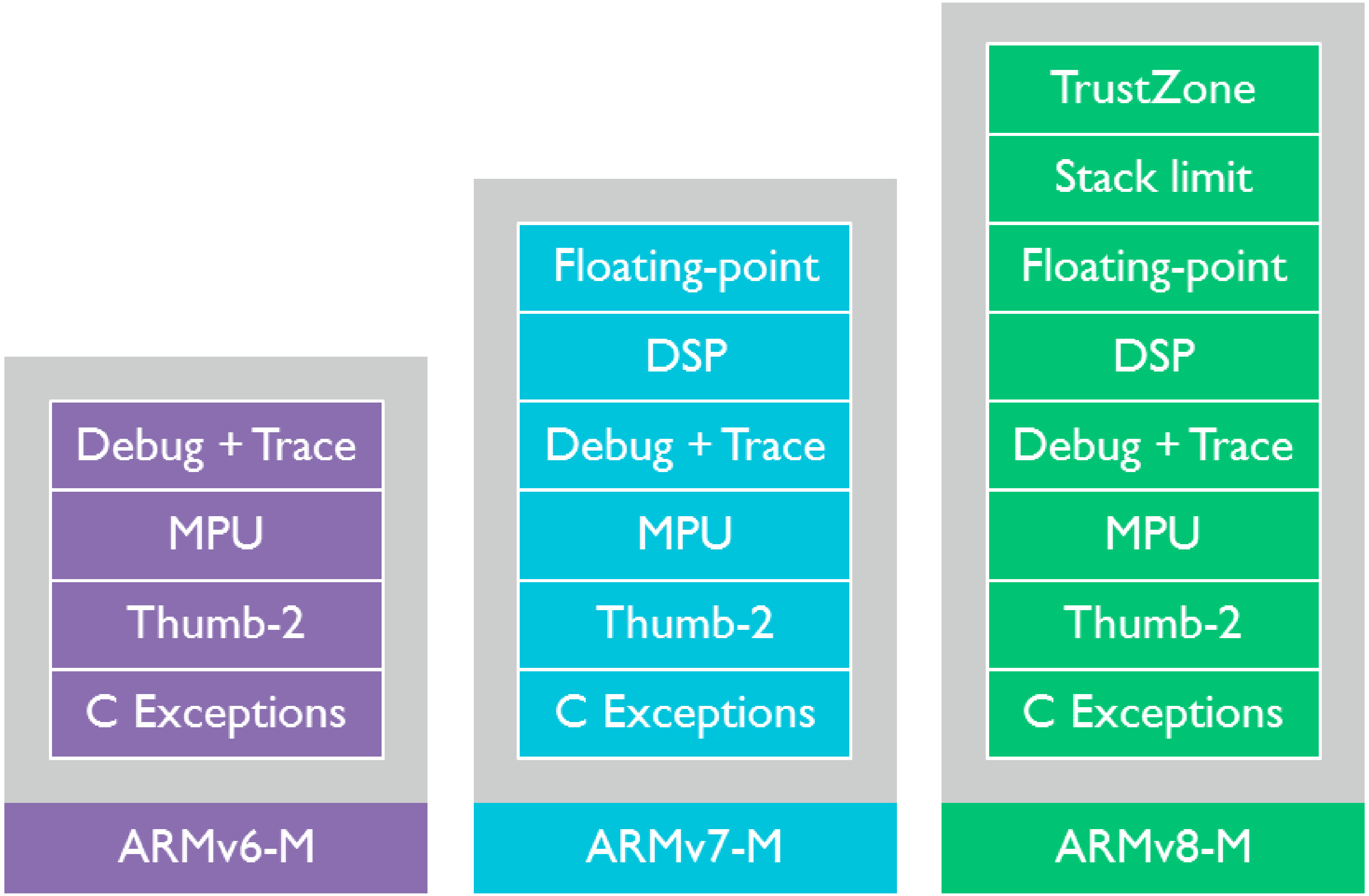
source: ARM
ARM Cortex-M for Beginners - Yiu

3 – ARM CORTEX-M ARCHITECTURE

- History of ARM
- Cortex-M Features
- Cortex-M Instruction Set Architecture
- Instruction Set
- Memory Access
  - Memory-mapped I/O
- Exception Handling

3.1 – ARM HISTORY

- Over time, several ARM architecture versions were released. From the seminal ARMv1 to the ARMv8.3-A, released in 2017. The documentation of these standards is available on the arm.com website.
- For each architectural version there are several implementations



Debug + Trace
MPU
Thumb-2
C Exceptions
ARMv6-M

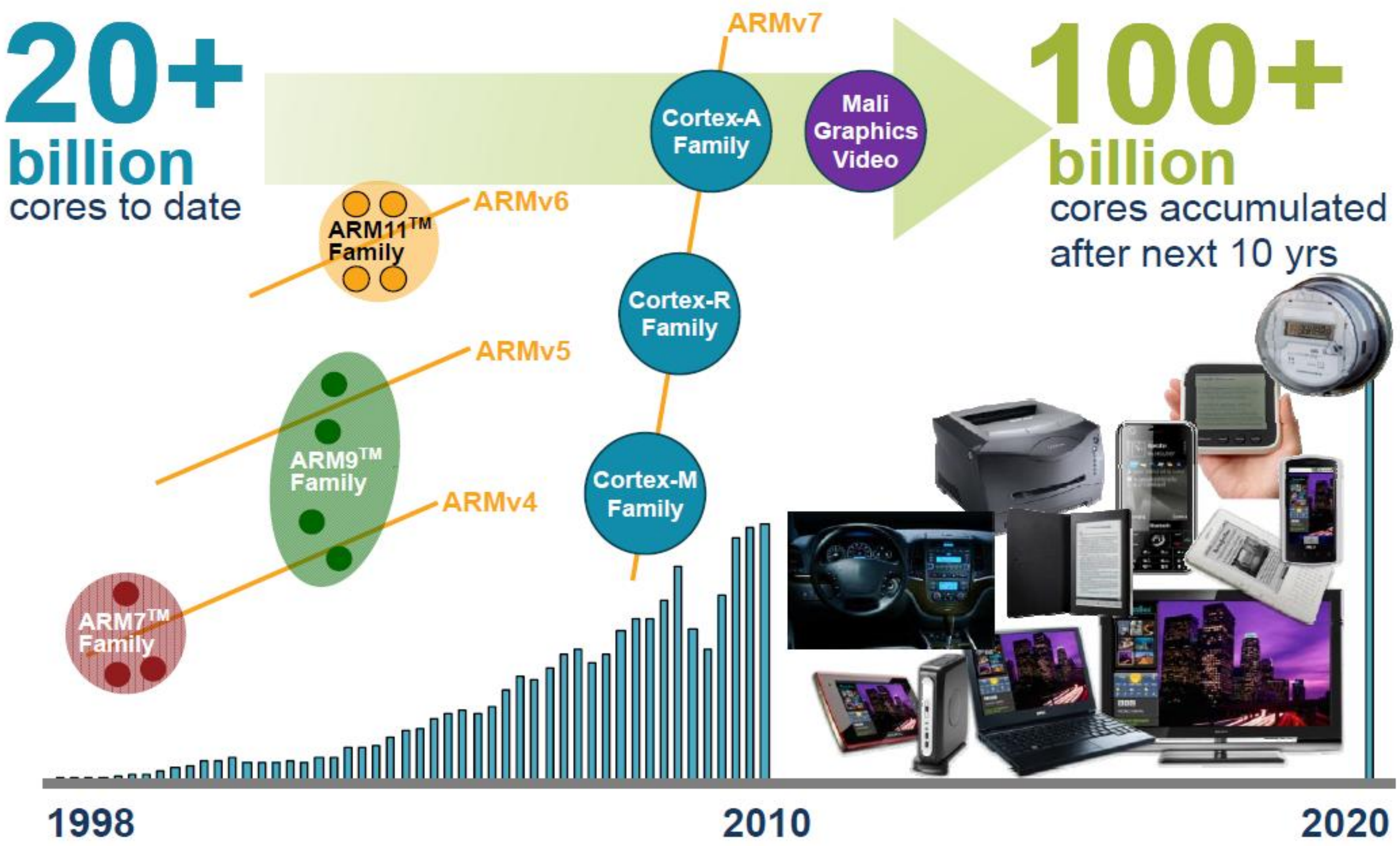
Floating-point
DSP
Debug + Trace
MPU
Thumb-2
C Exceptions
ARMv7-M

TrustZone
Stack limit
Floating-point
DSP
Debug + Trace
MPU
Thumb-2
C Exceptions
ARMv8-M

Cortex-M0
Cortex-M0+

Cortex-M3
Cortex-M4
Cortex-M7

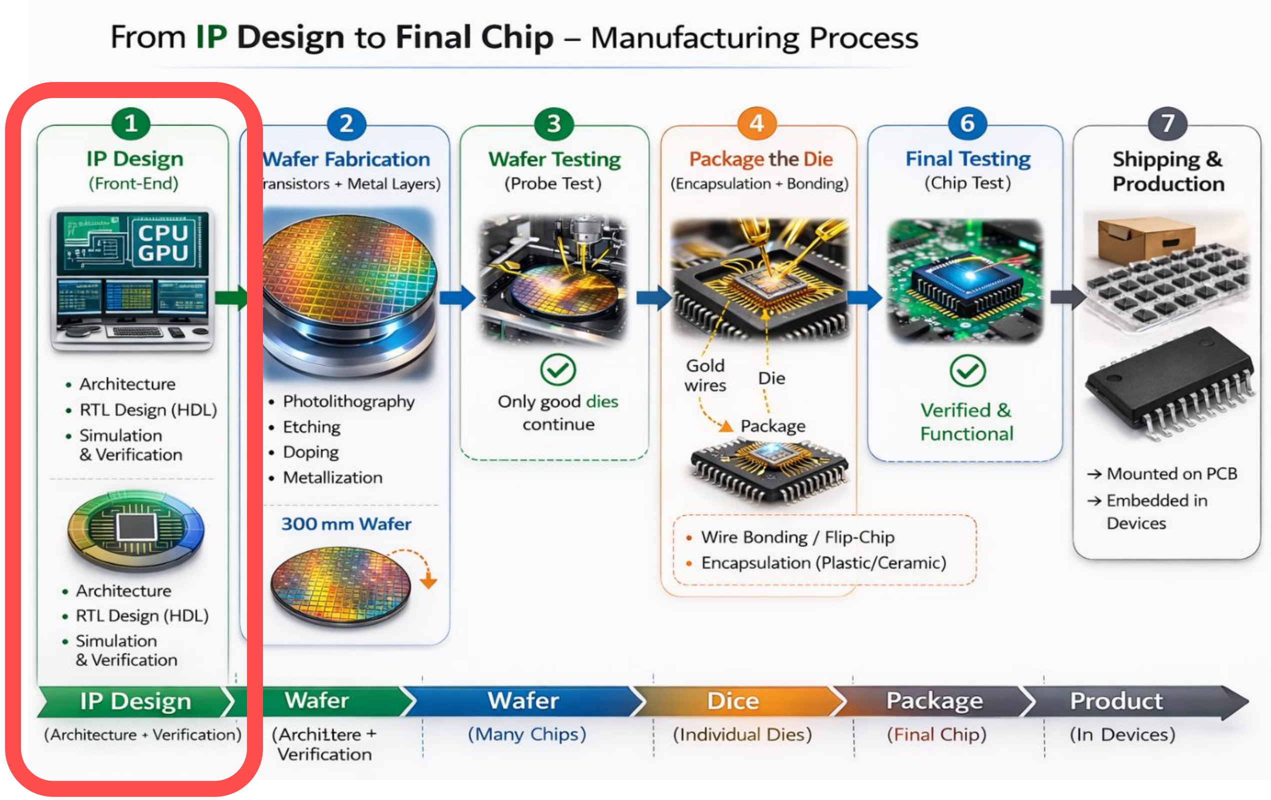
Cortex-M23
Cortex-M33



source: ARM
The Future is in
Your Hands -
Peter Middleton

ARM PRICING MODEL

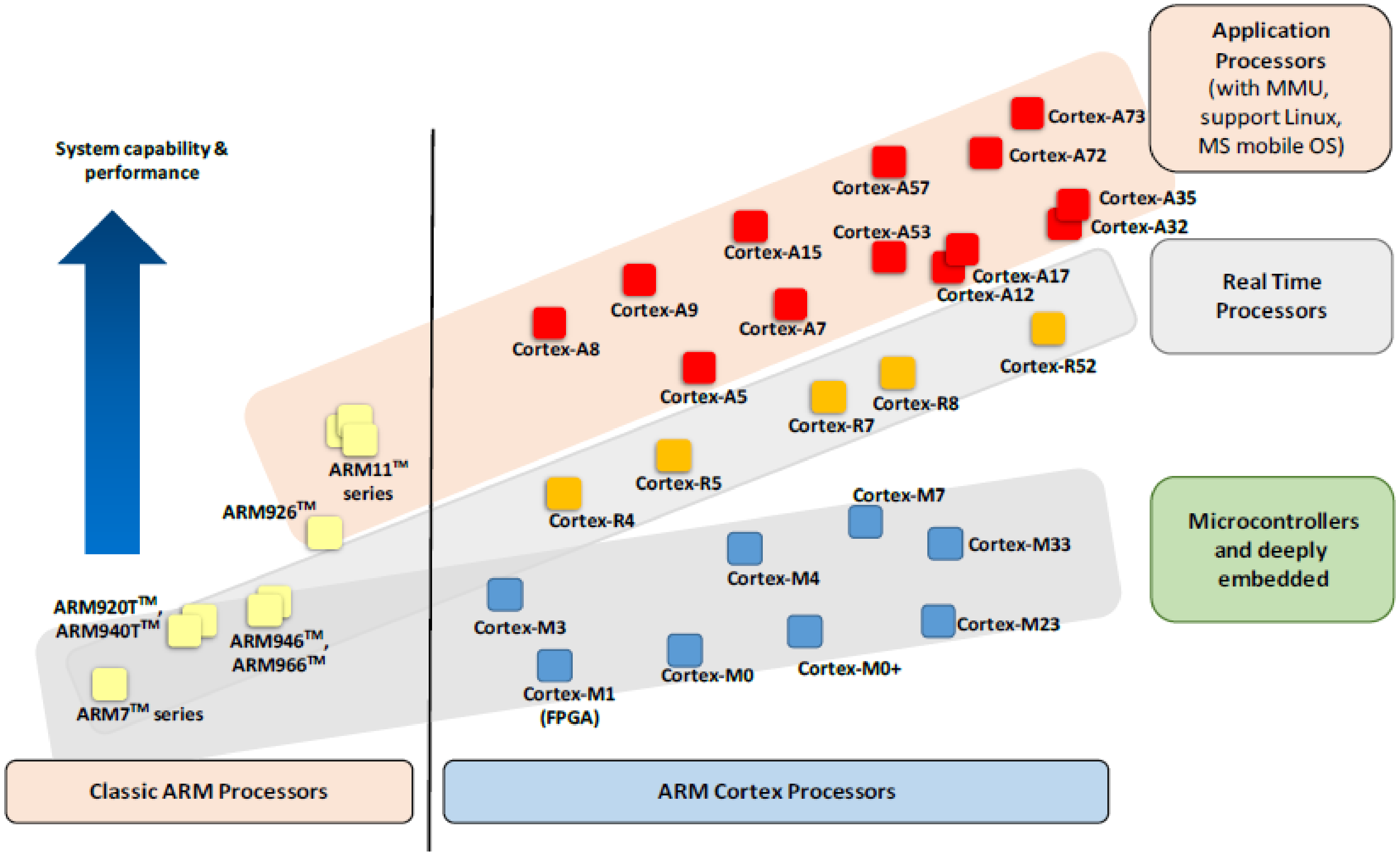
- ARM only provides the IP Design of a processor!
- No fabrication, no testing
- This is very clever, because you save a lot of money related to fab
- Fabrication is performed by 3rd party organizations (e.g., TSMC)



COMPANIES UTILIZING ARM ARCHITECTURE

ARM	System-On-a-Chip (SoC)	Products Containing	Type of ARM processor	Number of Cores
Apple	A4	iPhone 4, iPod Touch (4th Gen), iPad (1st Gen), AppleTV (2nd Gen)	Cortex-A8	1
	A5	iPhone 4S,iPad 2, AppleTV (3rd Gen)	Cortex-A9	2
	A5X	iPad (3rd Gen, Retina Display)	Cortex-A9	2
Samsung	Exynos 3 Single	Samsung Galaxy S, Samsung Galaxy Nexus S	Cortex-A8	1
	Exynos 4 Dual	Samsung Galaxy SII, Samsung Galaxy Note	Cortex-A9	2
	Exynos 4 Quad	Samsung Galaxy SIII	Cortex-A9	4
	Exynos 5 Dual	N/A	Cortex-A15	2
Nvidia	Tegra	Microsoft Zune HD	ARM11	1
	Tegra 2	Samsung Galaxy Tab 10.1, Motorola Xoom, Dell Streak 7 & Pro, Sony Tablet S	Cortex-A9	2
	Tegra X1	Motorola ATRIX, Asus Transformer TF101, Microsoft Kin One	Cortex-A57	4
Texas Instruments	OMAP 3	Barnes and Noble Nook Color	Cortex-A8	1
	OMAP 4	Amazon Kindle Fire, Samsung Galaxy Tab 2, Blackberry Playbook, Samsung Galaxy Nexus	Cortex-A9	2
	OMAP 5	N/A	Cortex-A15	2

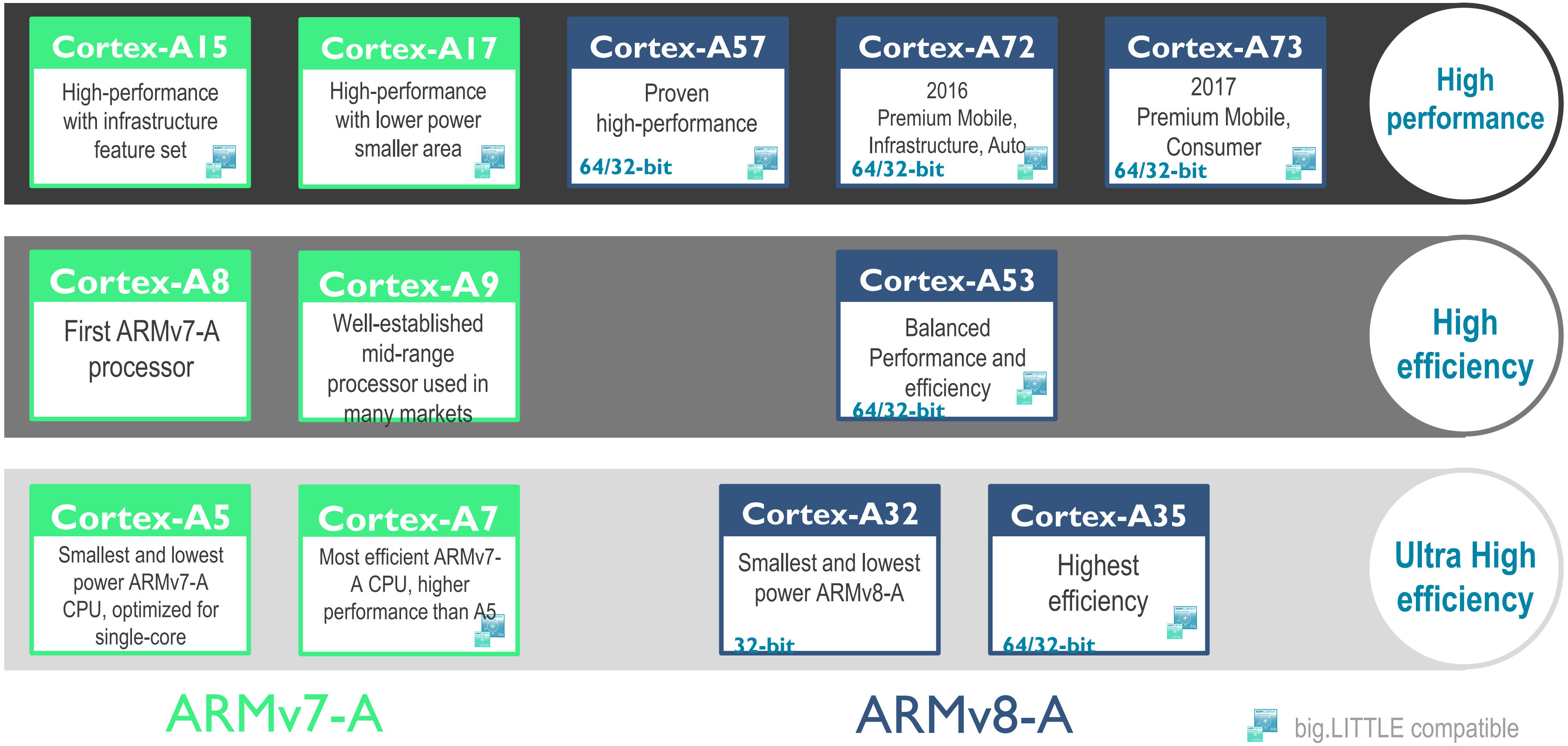
ARM PROCESSOR FAMILY



source: ARM
ARM Cortex-M for Beginners - Yiu

ARM CORTEX-A PORTFOLIO

- **Efficient application processors for every level of performance**



Cortex-A15
High-performance with infrastructure feature set

Cortex-A17
High-performance with lower power smaller area

Cortex-A57
Proven high-performance
64/32-bit

Cortex-A72
2016
Premium Mobile, Infrastructure, Auto
64/32-bit

Cortex-A73
2017
Premium Mobile, Consumer
64/32-bit

High performance

Cortex-A8
First ARMv7-A processor

Cortex-A9
Well-established mid-range processor used in many markets

Cortex-A53
Balanced Performance and efficiency
64/32-bit

High efficiency

Cortex-A5
Smallest and lowest power ARMv7-A CPU, optimized for single-core

Cortex-A7
Most efficient ARMv7-A CPU, higher performance than A5

Cortex-A32
Smallest and lowest power ARMv8-A
32-bit

Cortex-A35
Highest efficiency
64/32-bit

Ultra High efficiency

ARMv7-A

ARMv8-A

big.LITTLE compatible

ARM’S BIG.LITTLE TECHNOLOGY

What is big.LITTLE?

- Complex multicore CPU architecture combining…
  - Several high performance “big” cores
  - Several lower power “little” cores
- Cores should be architecturally compatible
- Cores may be…
  - Of 2 different architectures
  - Of the same architecture but with different highest frequency or cache size

ARM’S BIG.LITTLE TECHNOLOGY

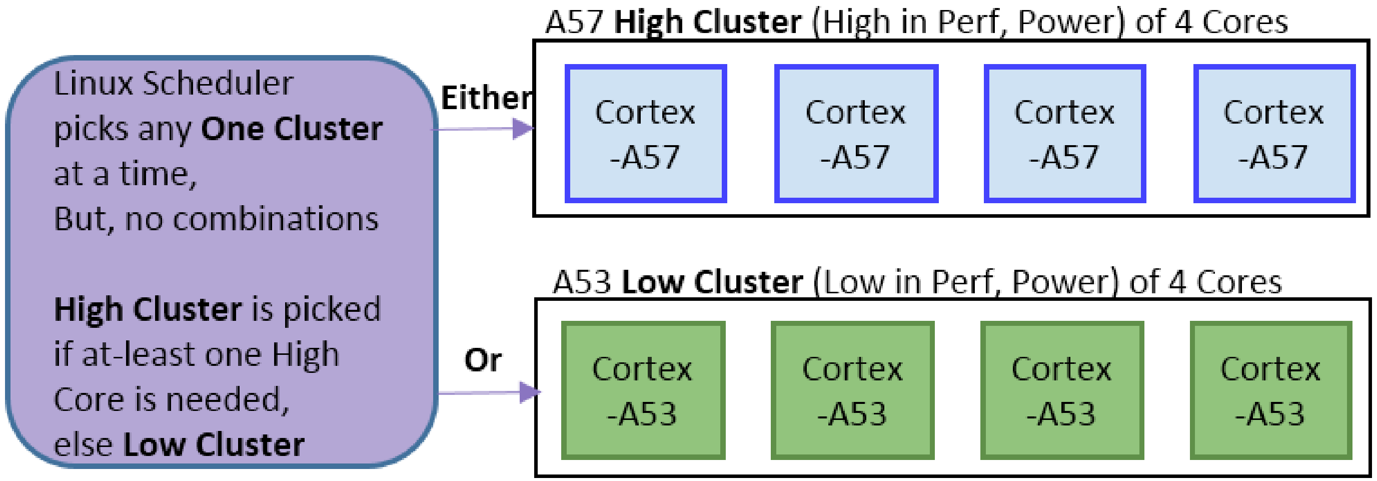
Why big.LITTLE?

- Targeting optimal power saving / performance balance
  - Real life CPU load is bursty
    - big.LITTLE allows for running power hungry cores only when bursts are coming
  - Peak performance only when it’s needed
  - Power optimized cores run most of the time
- More options for fine-tuning compared to standard Symmetric Multiprocessing (SMP)

ARM'S BIG.LITTLE TECHNOLOGY

Clustered Switching

- Simple implementation
- Identically-sized clusters of "big" or "little" cores
- The operating system scheduler can only see one cluster at a time
- No combinations
- High cluster is picked if at least one High core is needed, otherwise Low cluster



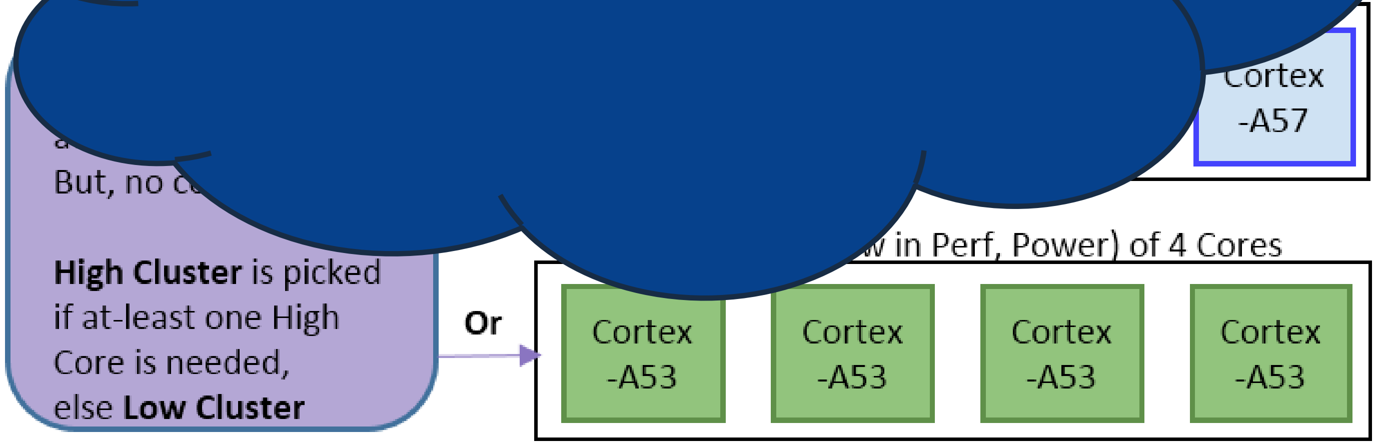
Linux Scheduler picks any **One Cluster** at a time,
But, no combinations
High Cluster is picked if at-least one High Core is needed, else **Low Cluster**
Either
Or
A57 **High Cluster** (High in Perf, Power) of 4 Cores
Cortex -A57
Cortex -A57
Cortex -A57
Cortex -A57
A53 **Low Cluster** (Low in Perf, Power) of 4 Cores
Cortex -A53
Cortex -A53
Cortex -A53
Cortex -A53

ARM’S BIG.LITTLE TECHNOLOGY

Clustered Switching

- Simple implementation
- Identically-sized clusters of “big” or “little”
- The operating system sch
- No combinations
- High cluster is pick

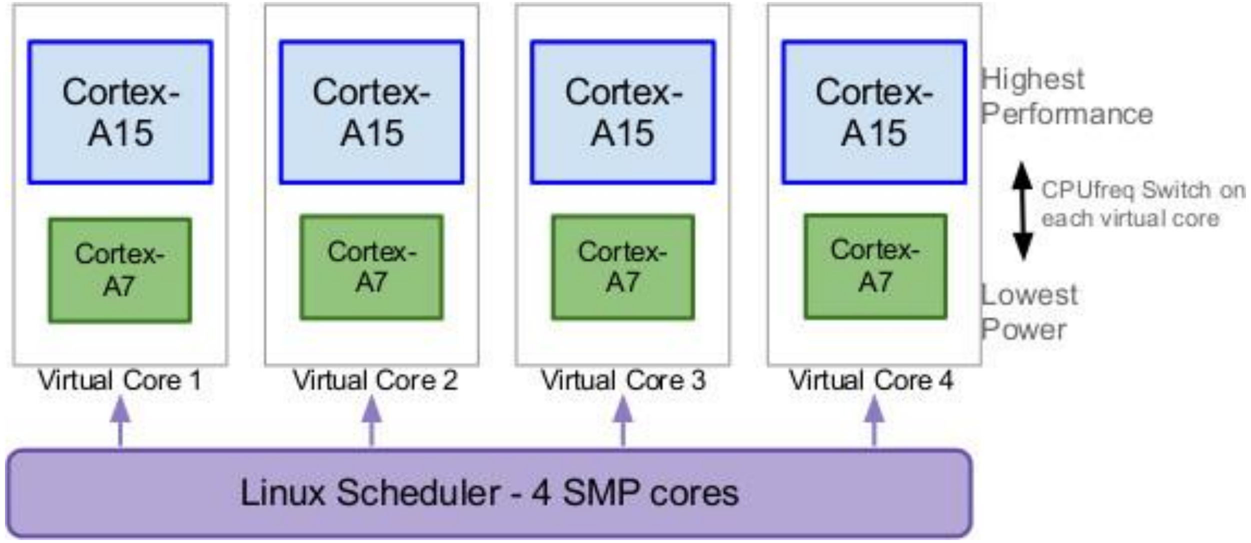
Depreceated in modern big.liitle conifgurations


Cortex -A57
a
But, no c
High Cluster is picked if at-least one High Core is needed, else **Low Cluster**
Or
w in Perf, Power) of 4 Cores
Cortex -A53
Cortex -A53
Cortex -A53
Cortex -A53

ARM’S BIG.LITTLE TECHNOLOGY

In-kernel switcher

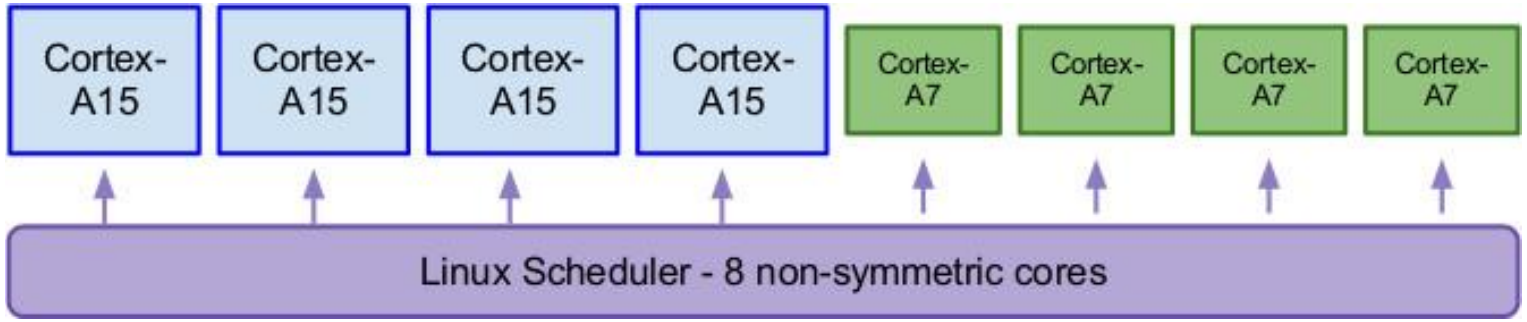
- Pairing up a “big” core with a “little” core
- Each pair operates as one virtual core
- Only one real core is (fully) powered up and running at a time
- 'Big' core is used when the demand is high 'Little' core is employed when demand is low



ARM’S BIG.LITTLE TECHNOLOGY

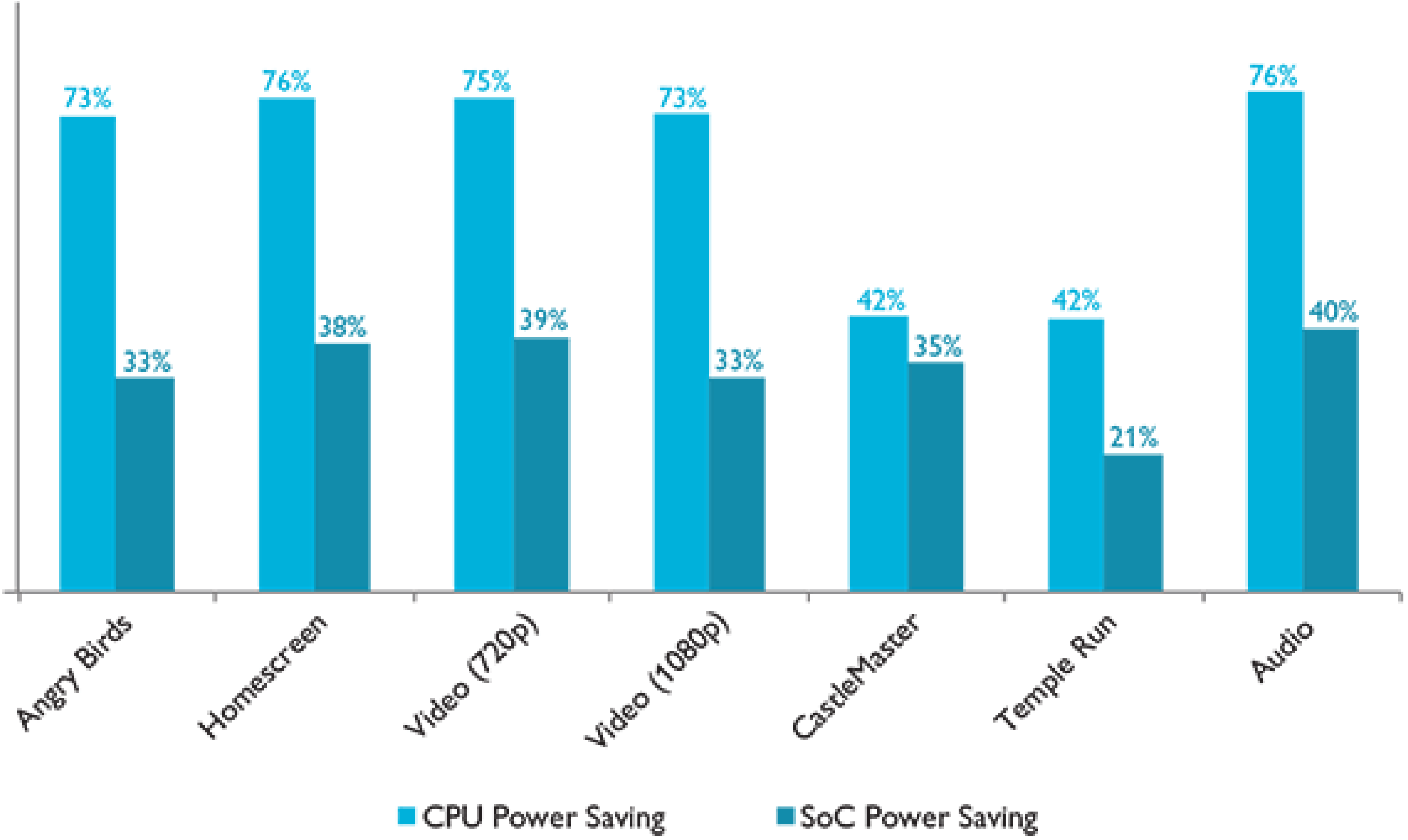
Heterogeneous Multi-processing

- Most powerful use of big.LITTLE architecture
- Use of all physical cores at the same time
- Threads with high priority or computational intensity are allocated to “big" cores, rest in “little” cores



ARM'S BIG.LITTLE TECHNOLOGY

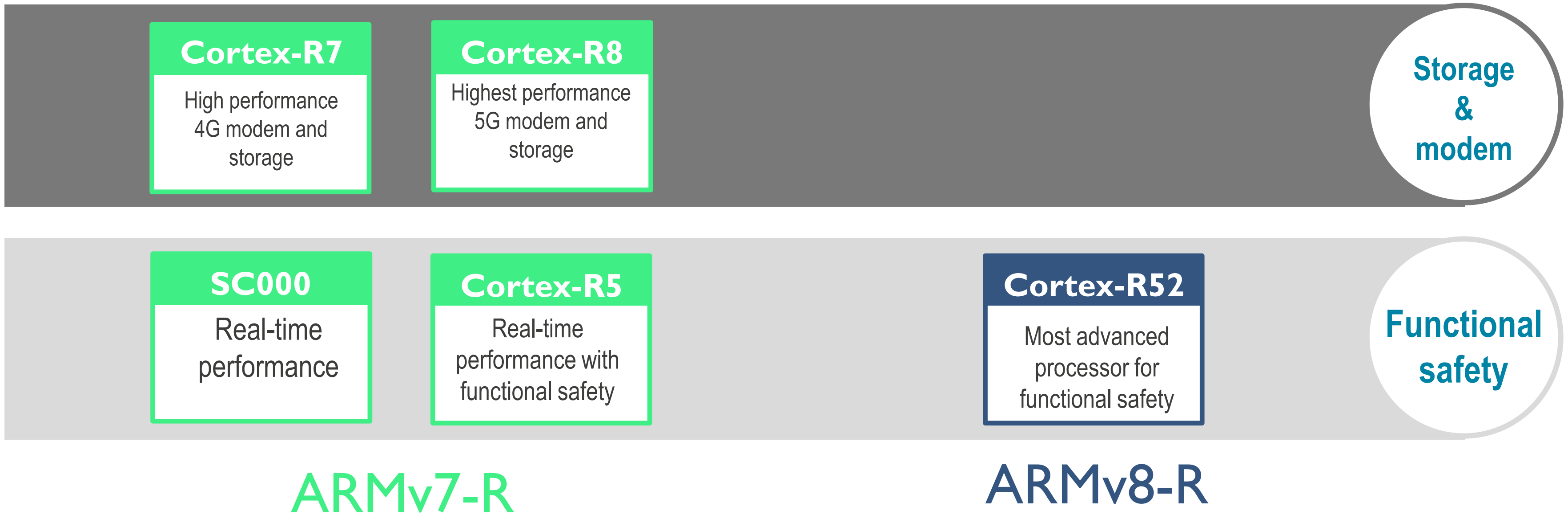
Measured CPU and SoC power savings



Measured CPU and SoC power savings on a **Cortex-A15 MP4 MP4 big.LITTLE** relative to a **Cortex-A15 MP4 SoC**

ARM CORTEX-R PORTFOLIO

- **Fast response - optimized for high performance, hard real-time applications**



Cortex-R7

High performance 4G modem and storage

Cortex-R8

Highest performance 5G modem and storage

Storage & modem

SC000

Real-time performance

Cortex-R5

Real-time performance with functional safety

Cortex-R52

Most advanced processor for functional safety

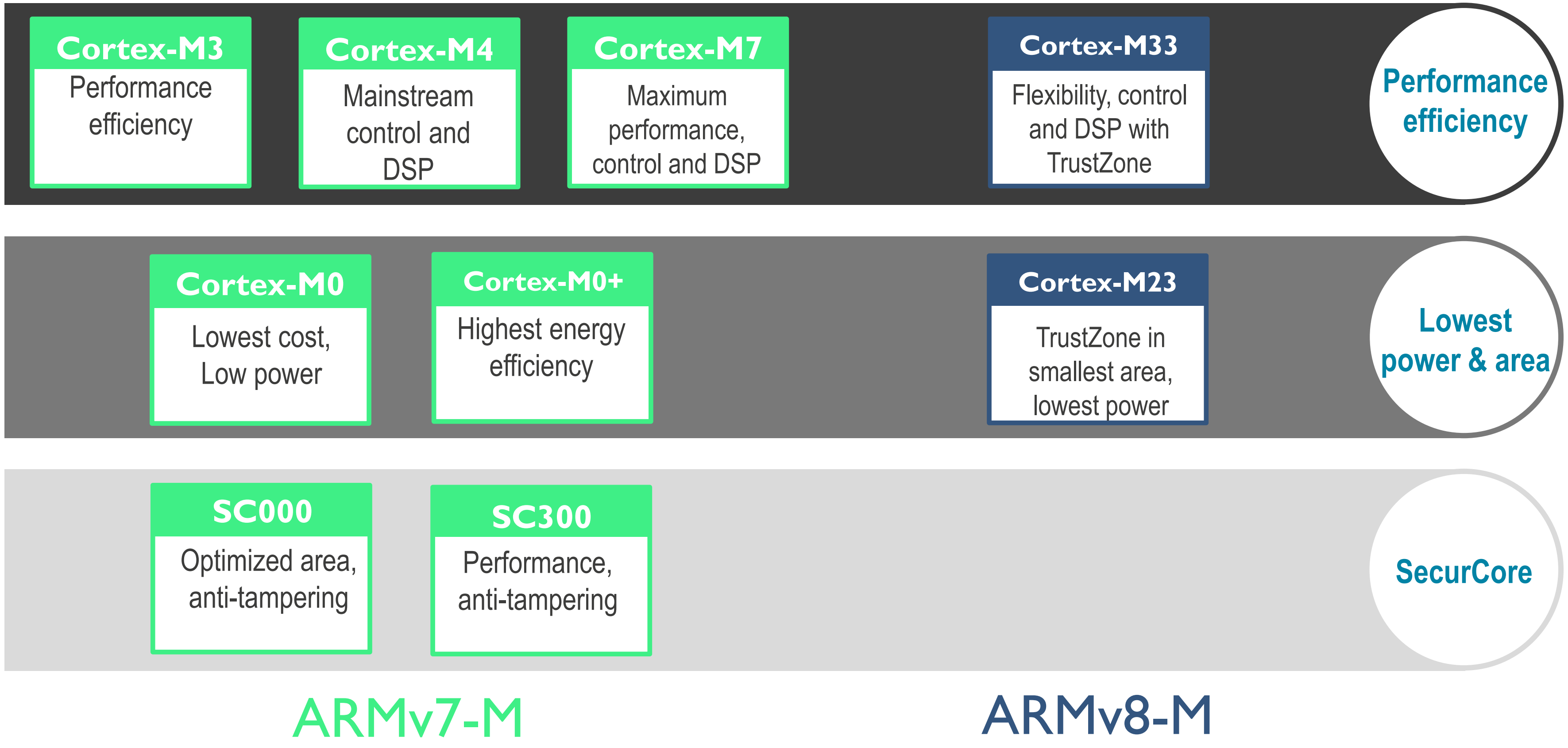
Functional safety

ARMv7-R

ARMv8-R

ARM CORTEX-M PORTFOLIO

- **Fast response - optimized for high performance, hard real-time applications**



Cortex-M3
Performance efficiency

Cortex-M4
Mainstream control and DSP

Cortex-M7
Maximum performance, control and DSP

Cortex-M33
Flexibility, control and DSP with TrustZone

Performance efficiency

Cortex-M0
Lowest cost, Low power

Cortex-M0+
Highest energy efficiency

Cortex-M23
TrustZone in smallest area, lowest power

Lowest power & area

SC000
Optimized area, anti-tampering

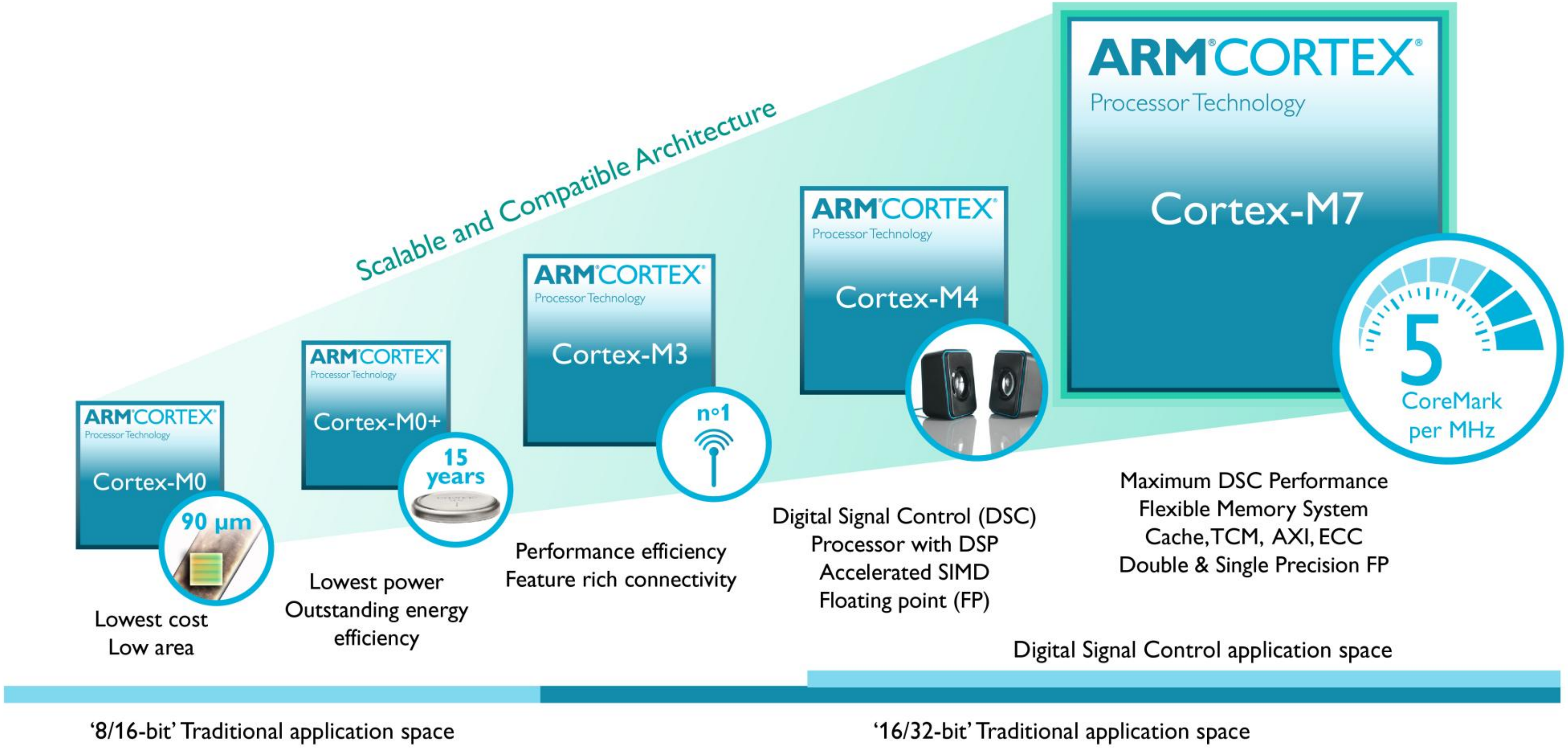
SC300
Performance, anti-tampering

SecurCore

ARMv7-M

ARMv8-M

THE CORTEX-M FAMILY



Scalable and Compatible Architecture
ARM CORTEX Processor Technology
Cortex-M0
90 µm
ARM CORTEX Processor Technology
Cortex-M0+
15 years
ARM CORTEX Processor Technology
Cortex-M3
n°1
ARM CORTEX Processor Technology
Cortex-M4
ARM CORTEX Processor Technology
Cortex-M7
5
CoreMark per MHz

Lowest cost
Low area

Lowest power
Outstanding energy efficiency

Performance efficiency
Feature rich connectivity

Digital Signal Control (DSC)
Processor with DSP
Accelerated SIMD
Floating point (FP)

Maximum DSC Performance
Flexible Memory System
Cache, TCM, AXI, ECC
Double & Single Precision FP

Digital Signal Control application space

'8/16-bit' Traditional application space

'16/32-bit' Traditional application space

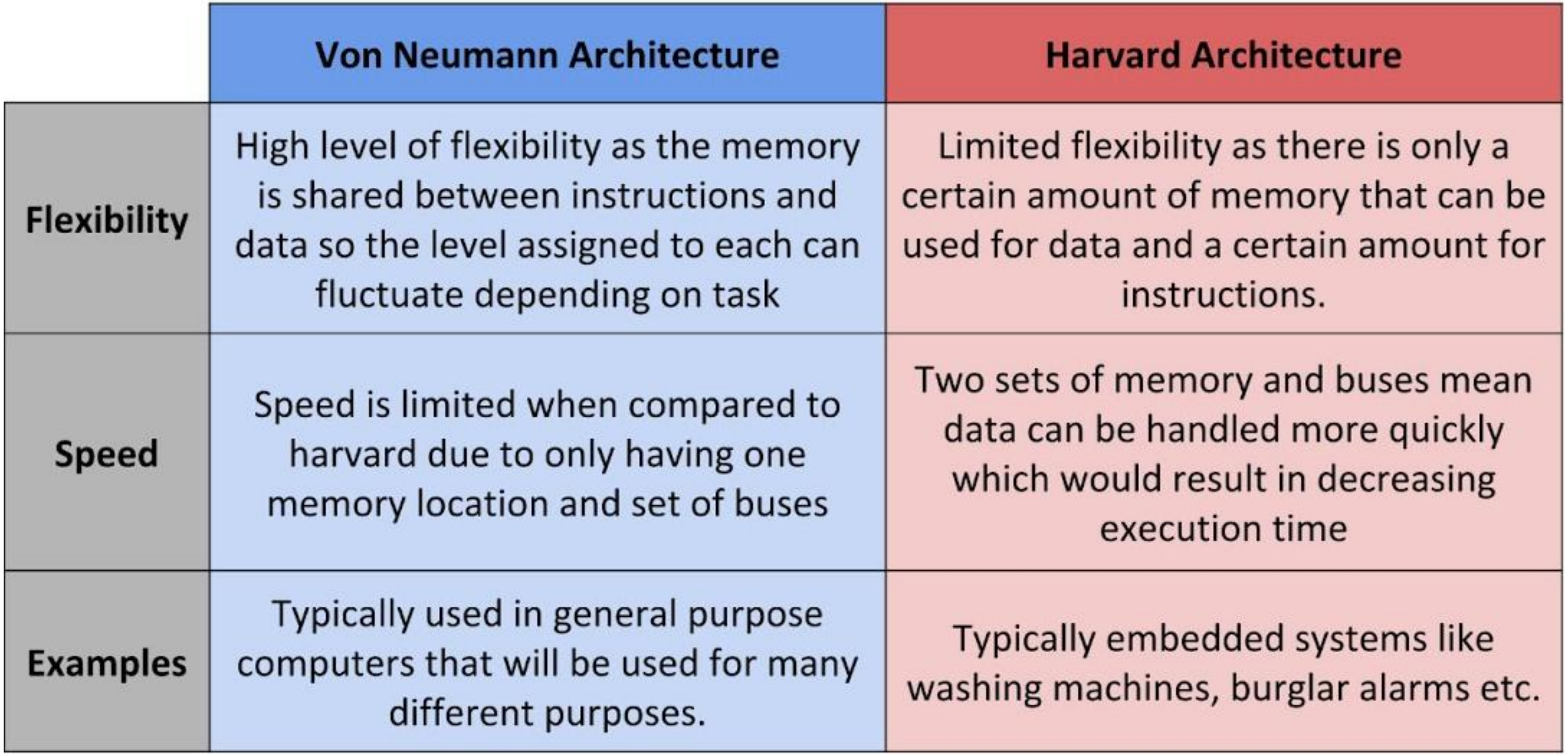
source: community.arm.com

3.2 – CORTEX-M FEATURES

Among the distinguishing features of the Cortex-M architecture are:

- 3-stage pipeline (except for Cortex-M7): Fetch, Decode, Execute,
- Harvard architecture (except for Cortex-M0 and M0+),
- Designed for power efficiency (includes an ultra low-power deep sleep),
- Thumb-2 instruction set, combining ARM performance and Thumb code density,
- Interrupt Controller (NVIC) is defined in the architecture; low latency vectored interrupt servicing,
- Interrupt servicing with tail-chaining and late arrival functionalities,
- Bit-banding to provide faster bit operations in memory and memory mapped I/O,
- MPU = Memory Protection Unit,
- Most instructions can be conditional.

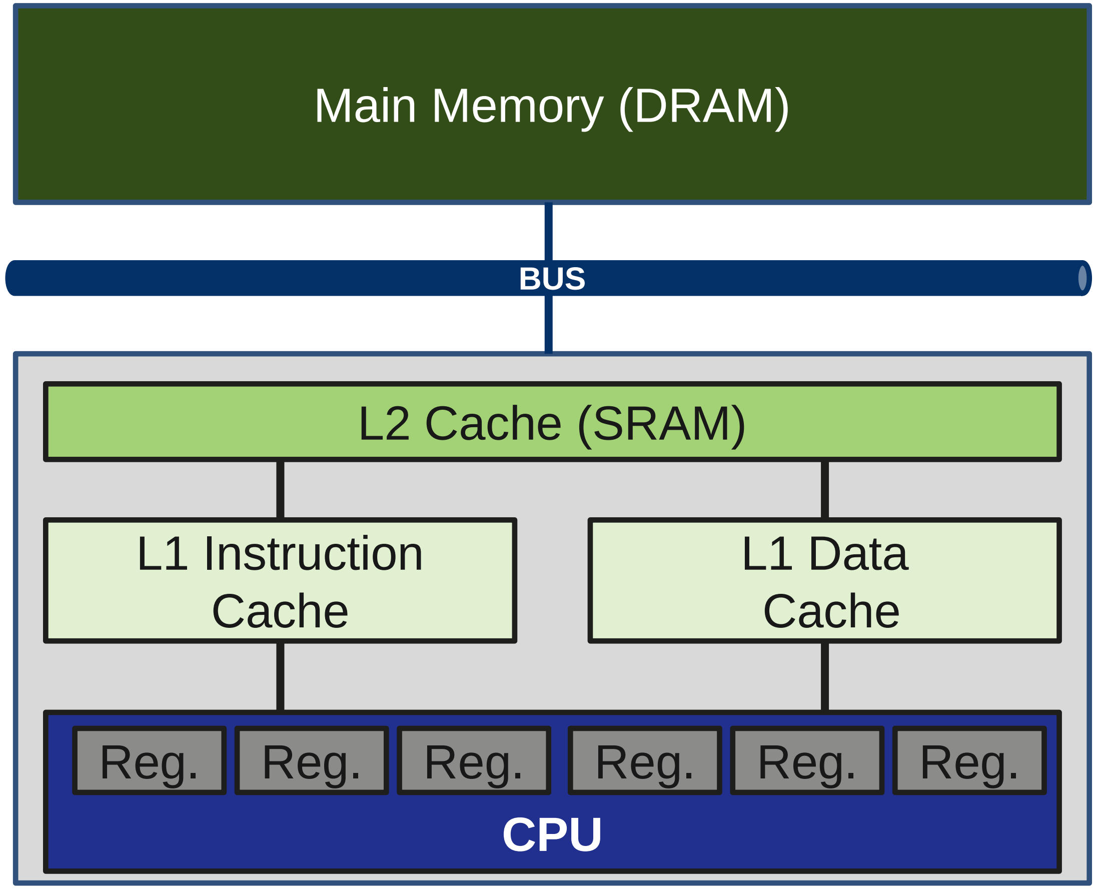
3.2 – CORTEX-M FEATURES: HARVARD ARCHITECTURE



	Von Neumann Architecture	Harvard Architecture
Flexibility	High level of flexibility as the memory is shared between instructions and data so the level assigned to each can fluctuate depending on task	Limited flexibility as there is only a certain amount of memory that can be used for data and a certain amount for instructions.
Speed	Speed is limited when compared to harvard due to only having one memory location and set of buses	Two sets of memory and buses mean data can be handled more quickly which would result in decreasing execution time
Examples	Typically used in general purpose computers that will be used for many different purposes.	Typically embedded systems like washing machines, burglar alarms etc.

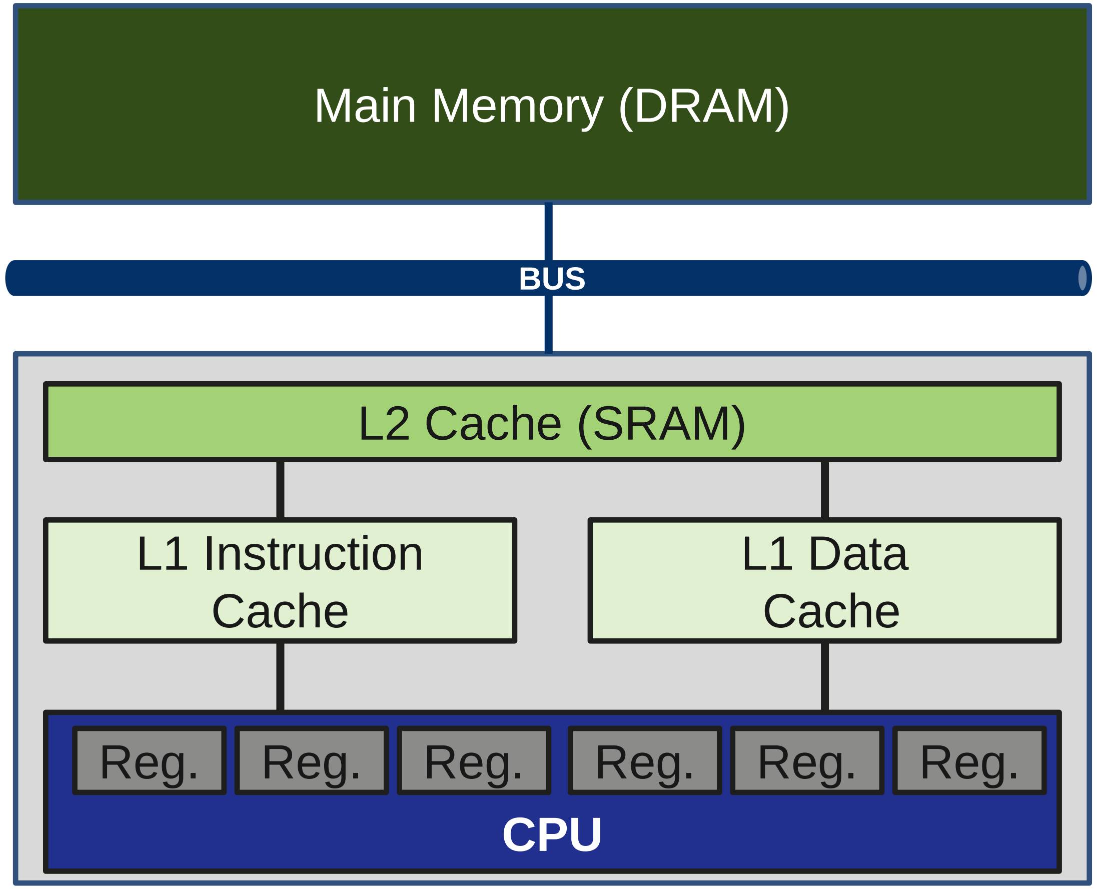
3.2 – CORTEX-M FEATURES: HARVARD ARCHITECTURE

Last time we discussed this architecture. Is this Von Neumann or Harvard?



3.2 – CORTEX-M FEATURES: HARVARD ARCHITECTURE

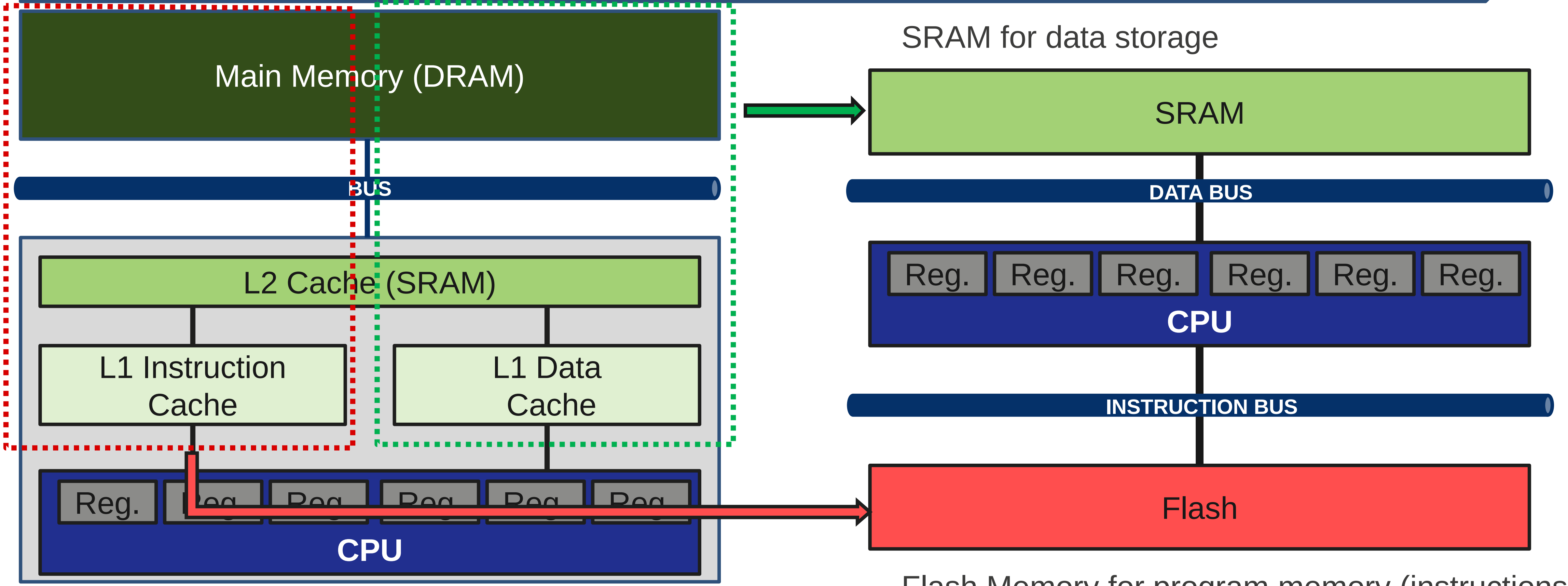
Last time we discussed this architecture. Is this Von Neumann or Harvard?



Best of the two worlds!
Modified Harvard Architecture

3.2 – CORTEX-M FEATURES: HARVARD ARCHITECTURE

How about Cortex-M architecture?

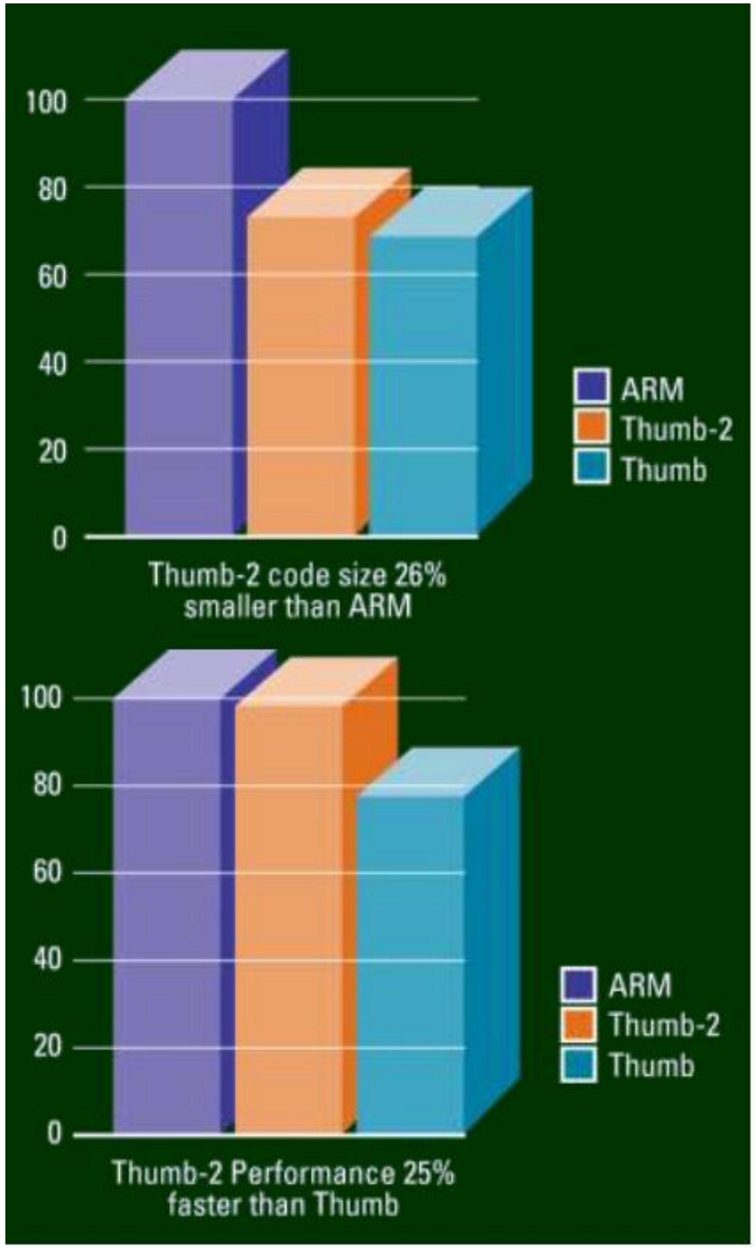


SRAM for data storage

Flash Memory for program memory (instructions)

3.2 – CORTEX-M FEATURES: THUMB

- The ARM7TDMI had two instructions sets:
- ARM - with 32-bit instructions, higher performance and lower code density,
- Thumb - with 16-bit instructions, lower performance and higher code density.
- Cortex-M has a single instruction set: Thumb-2
- It mixes 16-bit and 32-bit instructions. Its code density is similar to Thumb and its performance is similar to ARM.

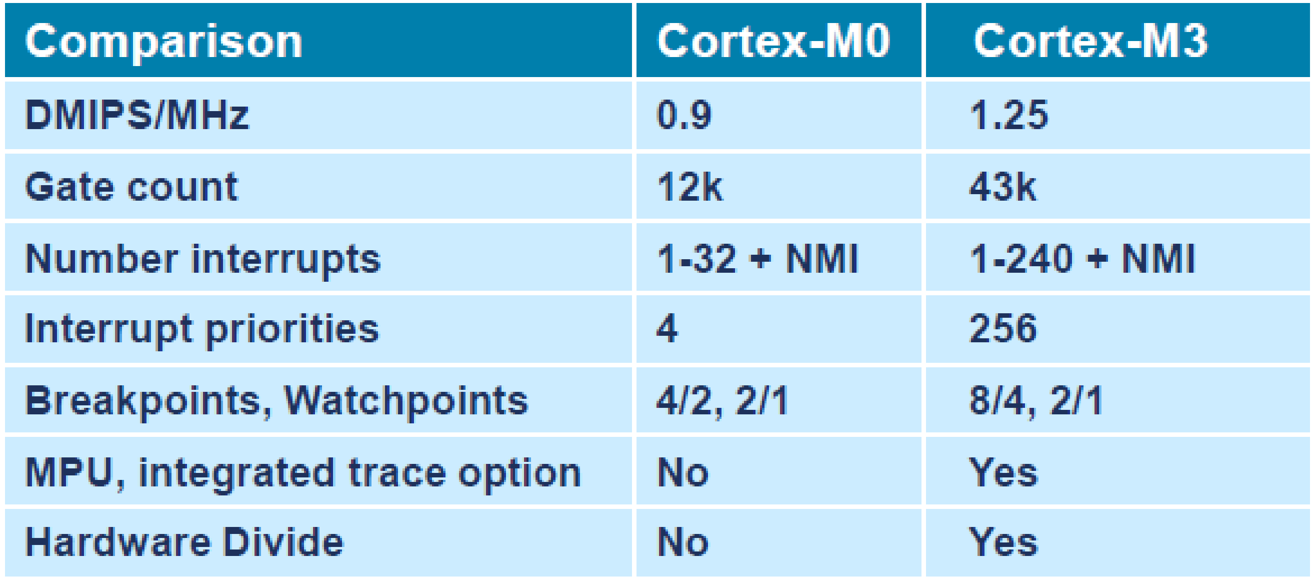


Thumb-2 code size 26% smaller than ARM

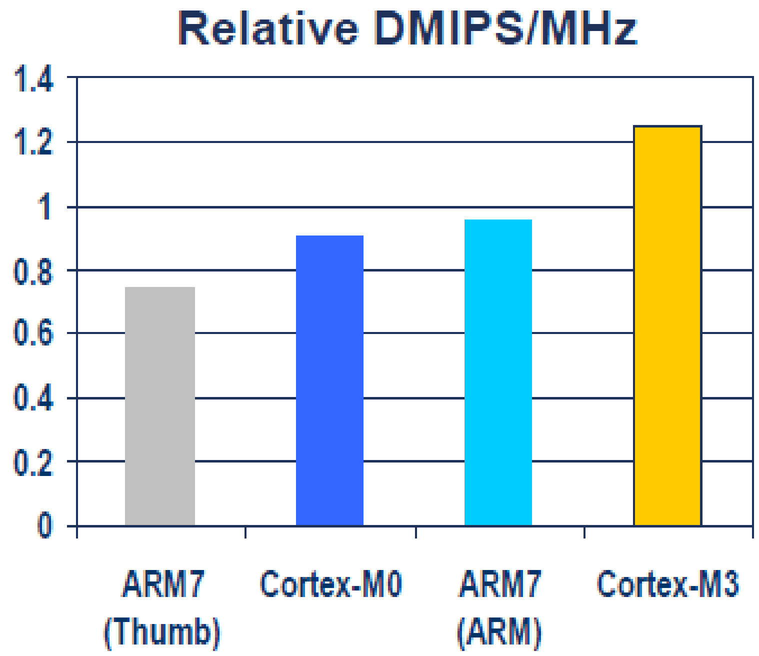
Thumb-2 Performance 25% faster than Thumb

source: ARM
The ARM Architecture - Joe Bungo

3.2 – CORTEX-M FEATURES: CHARACTERISTICS

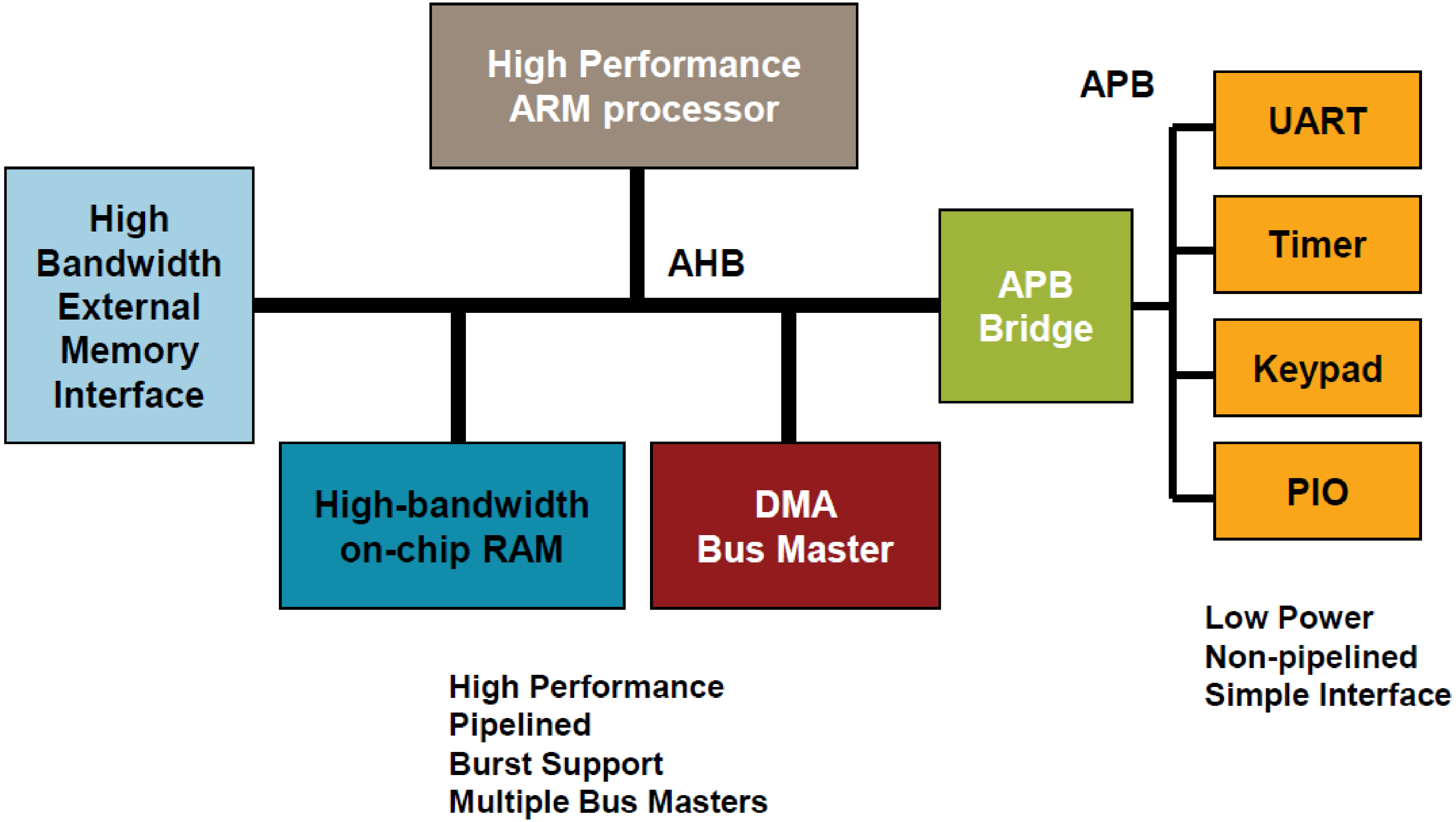


Comparison	Cortex-M0	Cortex-M3
DMIPS/MHz	0.9	1.25
Gate count	12k	43k
Number interrupts	1-32 + NMI	1-240 + NMI
Interrupt priorities	4	256
Breakpoints, Watchpoints	4/2, 2/1	8/4, 2/1
MPU, integrated trace option	No	Yes
Hardware Divide	No	Yes



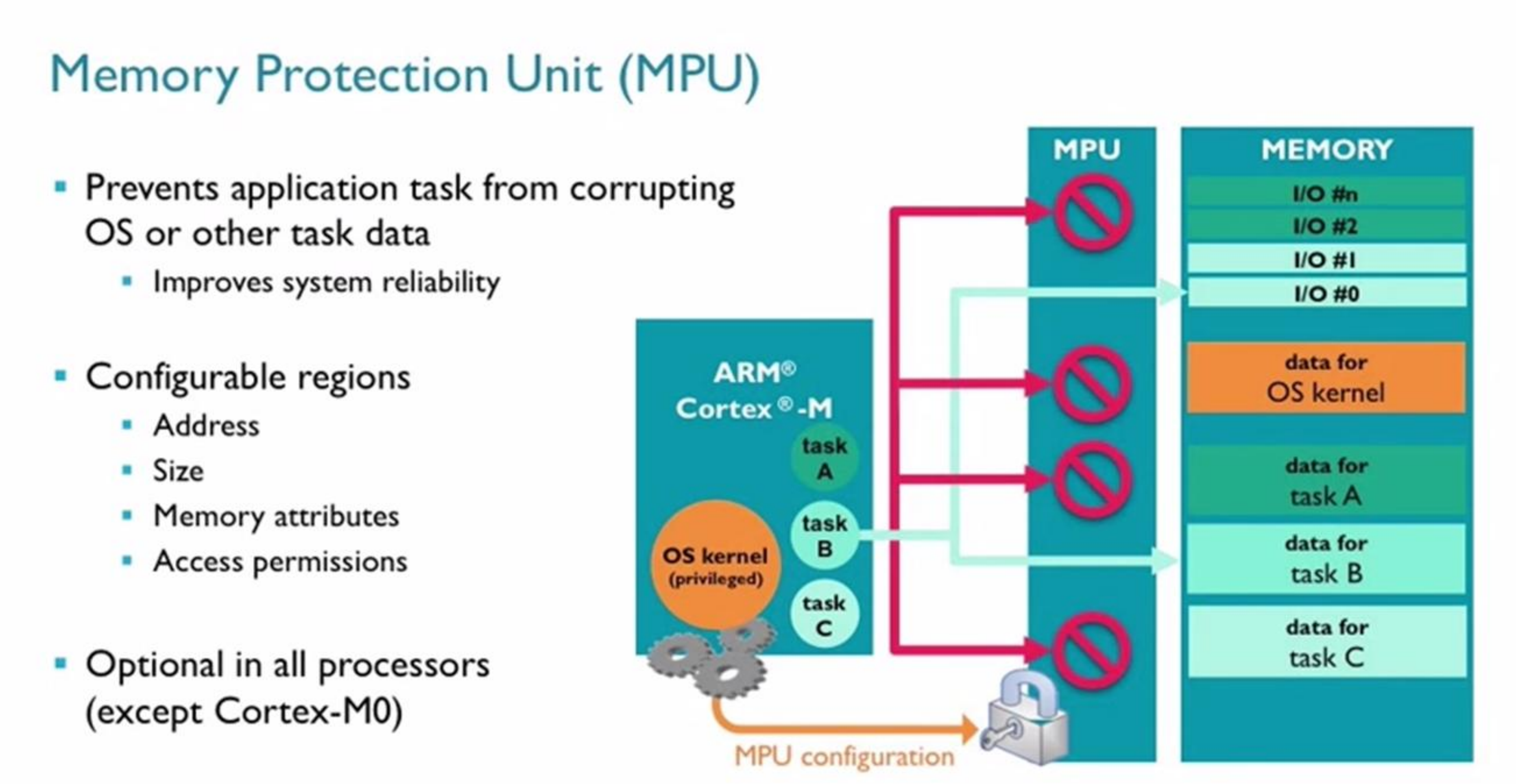
source: ARM
ARM Cortex-M Processor Family

3.2 – CORTEX-M FEATURES: BUSES



source: ARM
The ARM Architecture - Joe Bungo

3.2 – CORTEX-M FEATURES: MEMORY PROTECTION UNIT



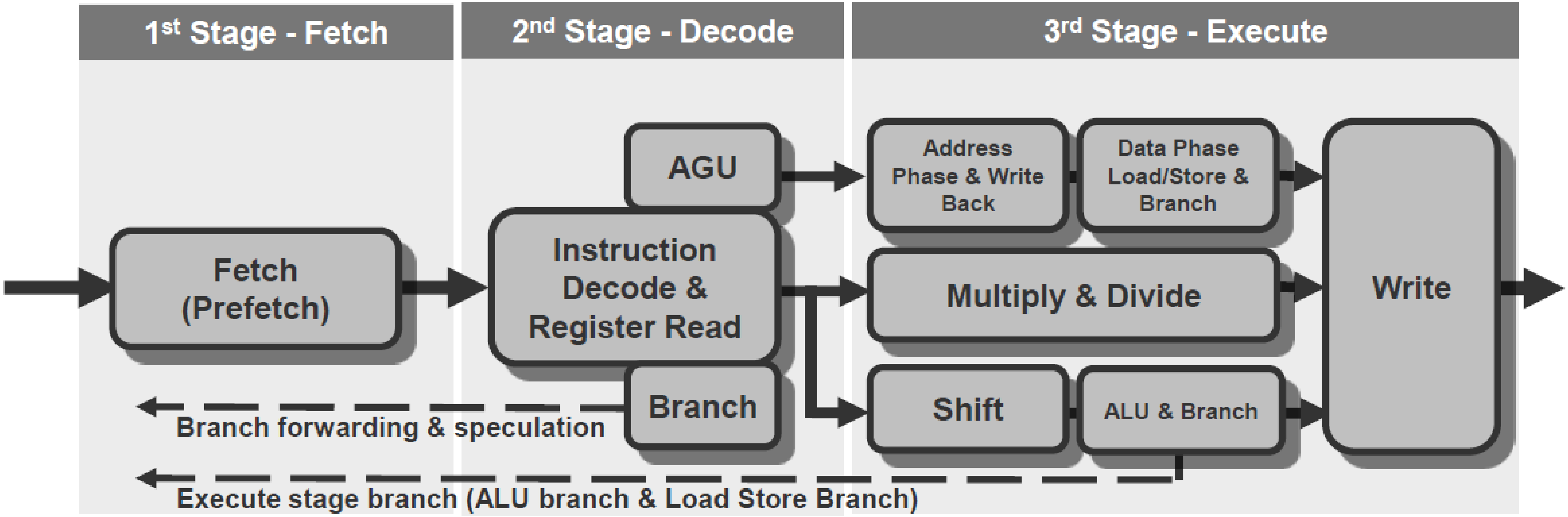
Memory Protection Unit (MPU)

- Prevents application task from corrupting OS or other task data
  - Improves system reliability
- Configurable regions
  - Address
  - Size
  - Memory attributes
  - Access permissions
- Optional in all processors (except Cortex-M0)

source: ARM
How to Choose Your
ARM Cortex-M Processor
- Tim Menasveta

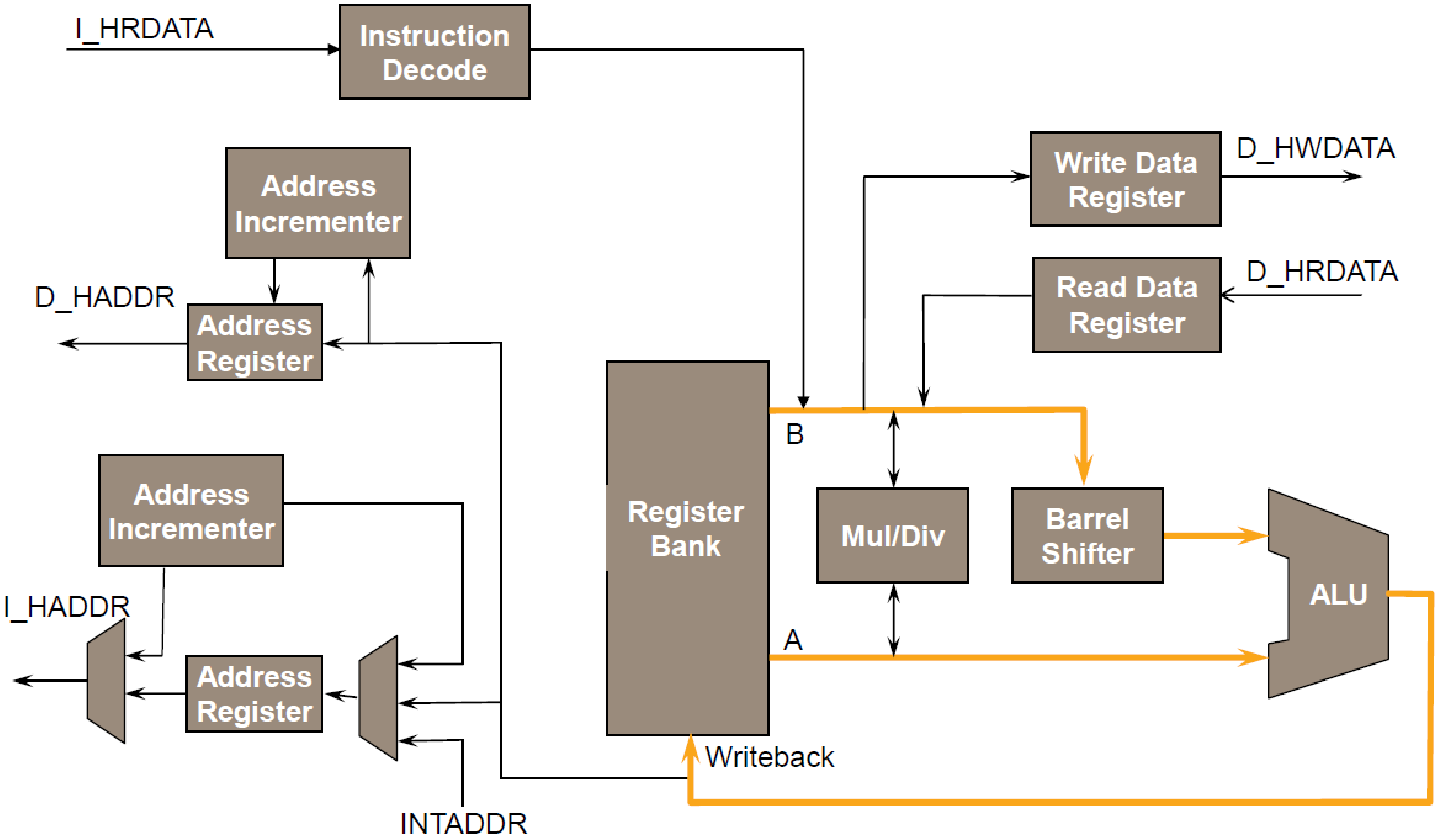
3.2 – CORTEX-M FEATURES: 3 STAGE PIPELINE

Cortex-M4 Pipeline


1st Stage - Fetch
2nd Stage - Decode
3rd Stage - Execute
AGU
Fetch (Prefetch)
Instruction Decode & Register Read
Branch
Address Phase & Write Back
Data Phase Load/Store & Branch
Multiply & Divide
Shift
ALU & Branch
Write
Branch forwarding & speculation
Execute stage branch (ALU branch & Load Store Branch)

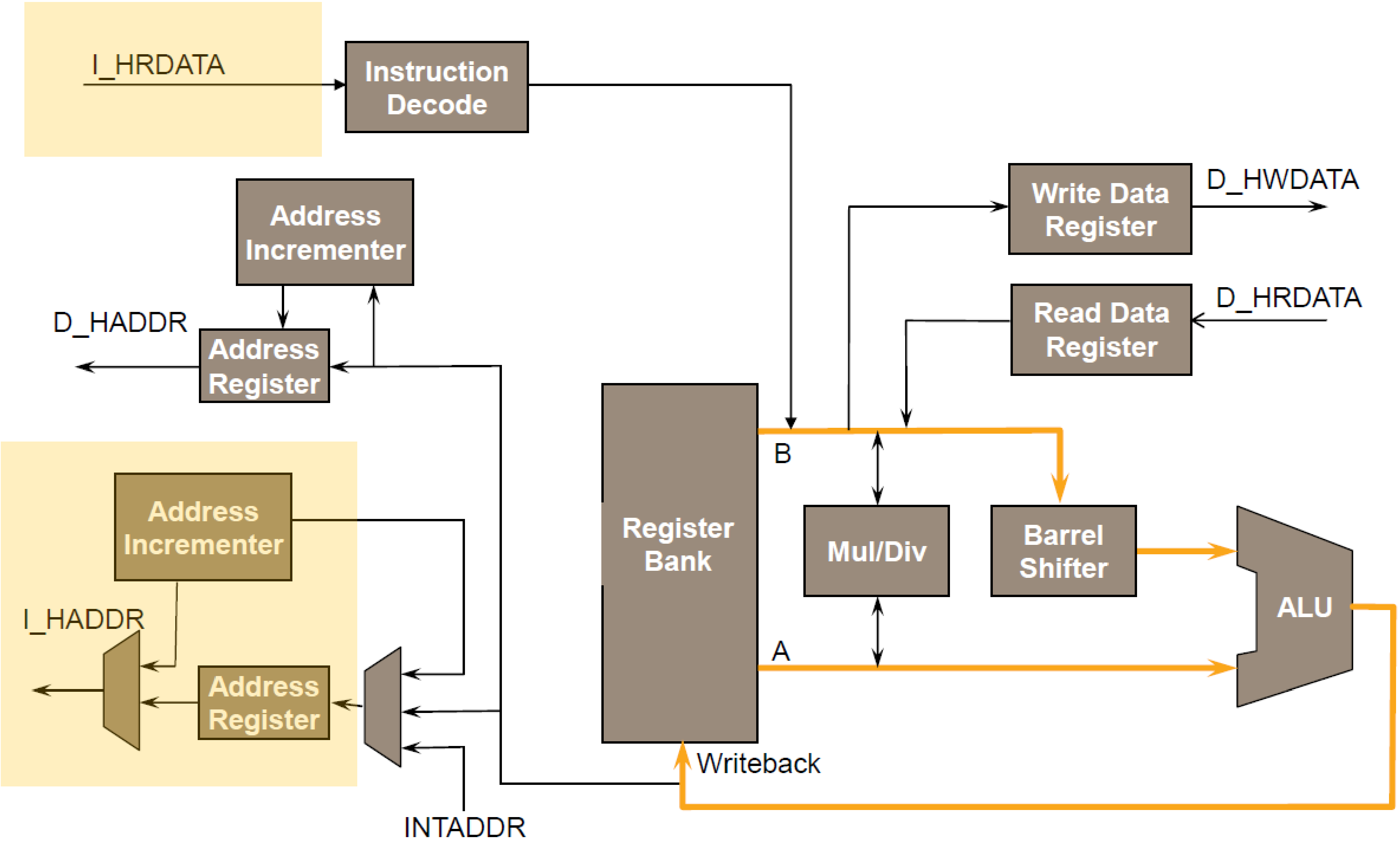
source: ARM
The ARM Architecture - Joe Bungo

Cortex-M3 Datapath



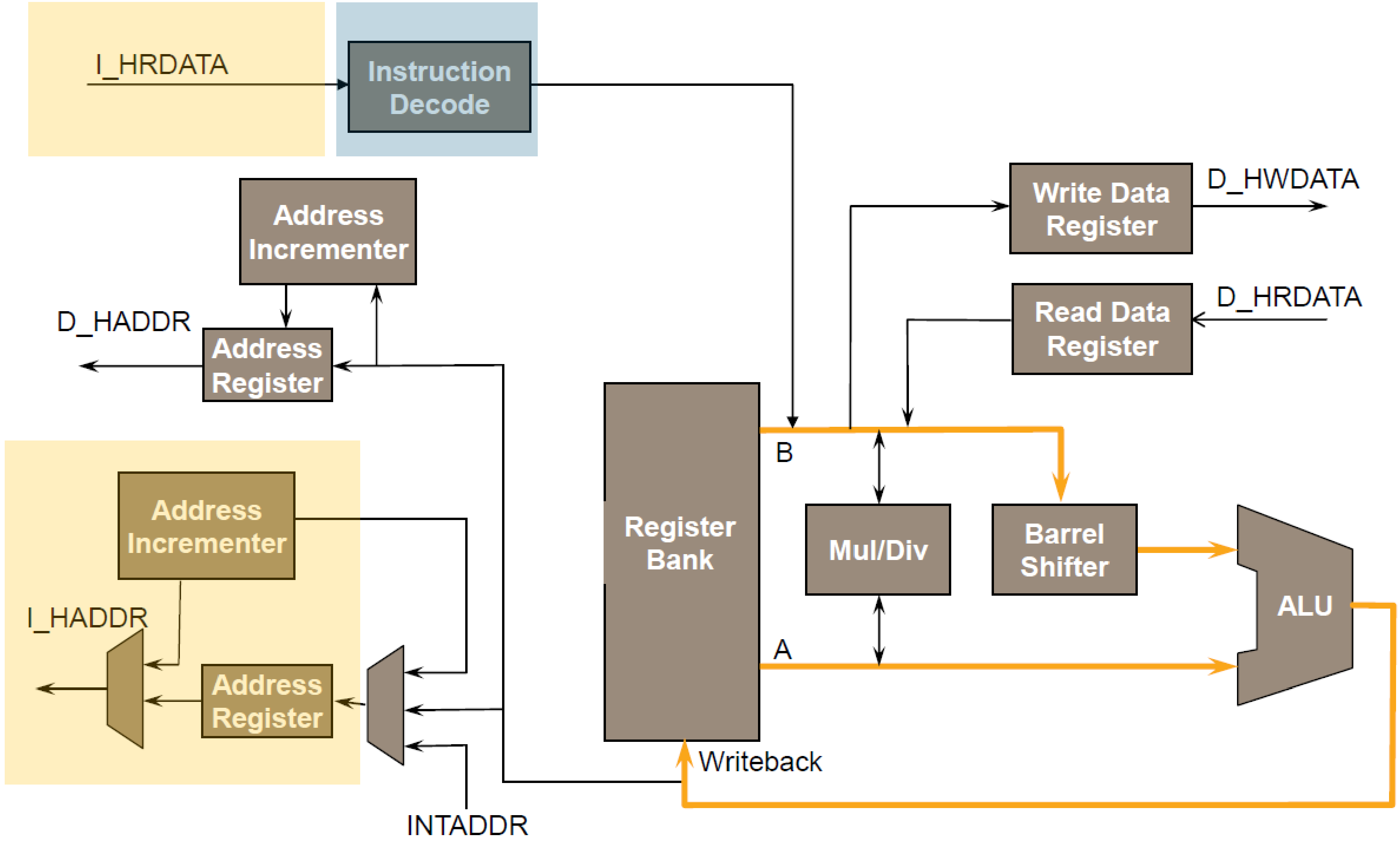
source: ARM
The ARM
Architecture - Joe
Bungo

Cortex-M3 Datapath FETCH



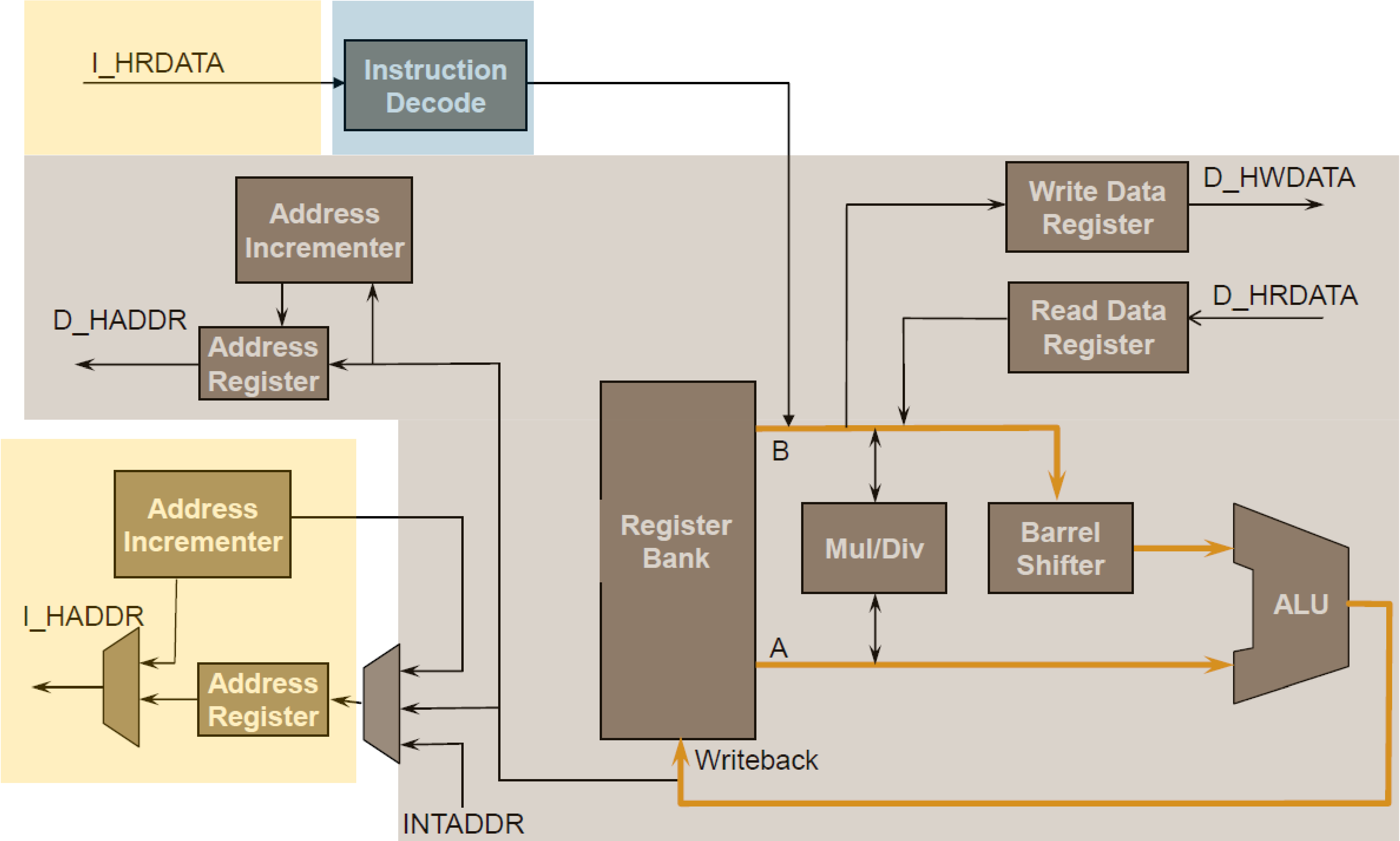
source: ARM The ARM Architecture - Joe Bungo

Cortex-M3 Datapath DECODE



source: ARM The ARM Architecture - Joe Bungo

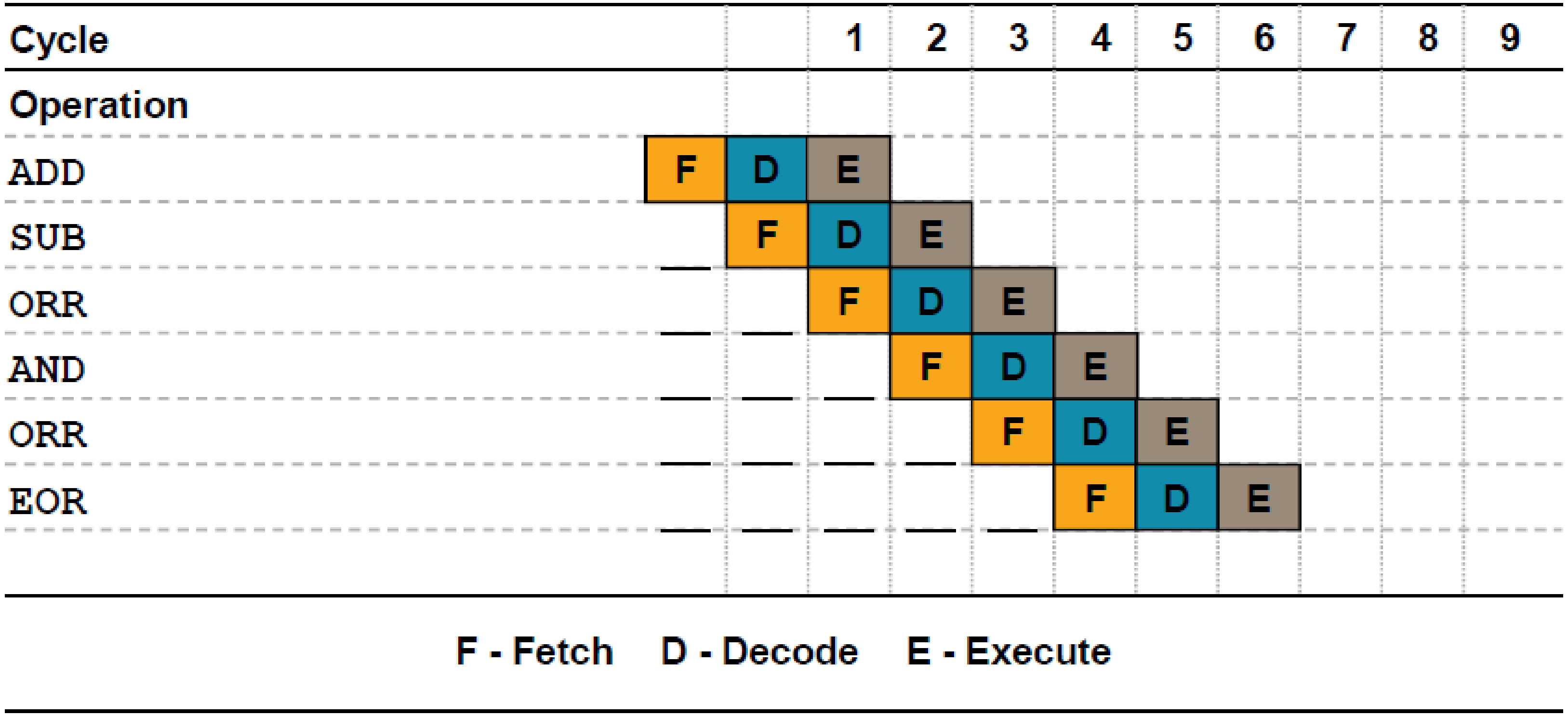
Cortex-M3 Datapath EXECUTE (+MEM +WB)



source: ARM
The ARM
Architecture - Joe
Bungo

3.2 – CORTEX-M FEATURES: PIPELINE EXAMPLE

Cortex-M4 Pipeline – example for optimal execution



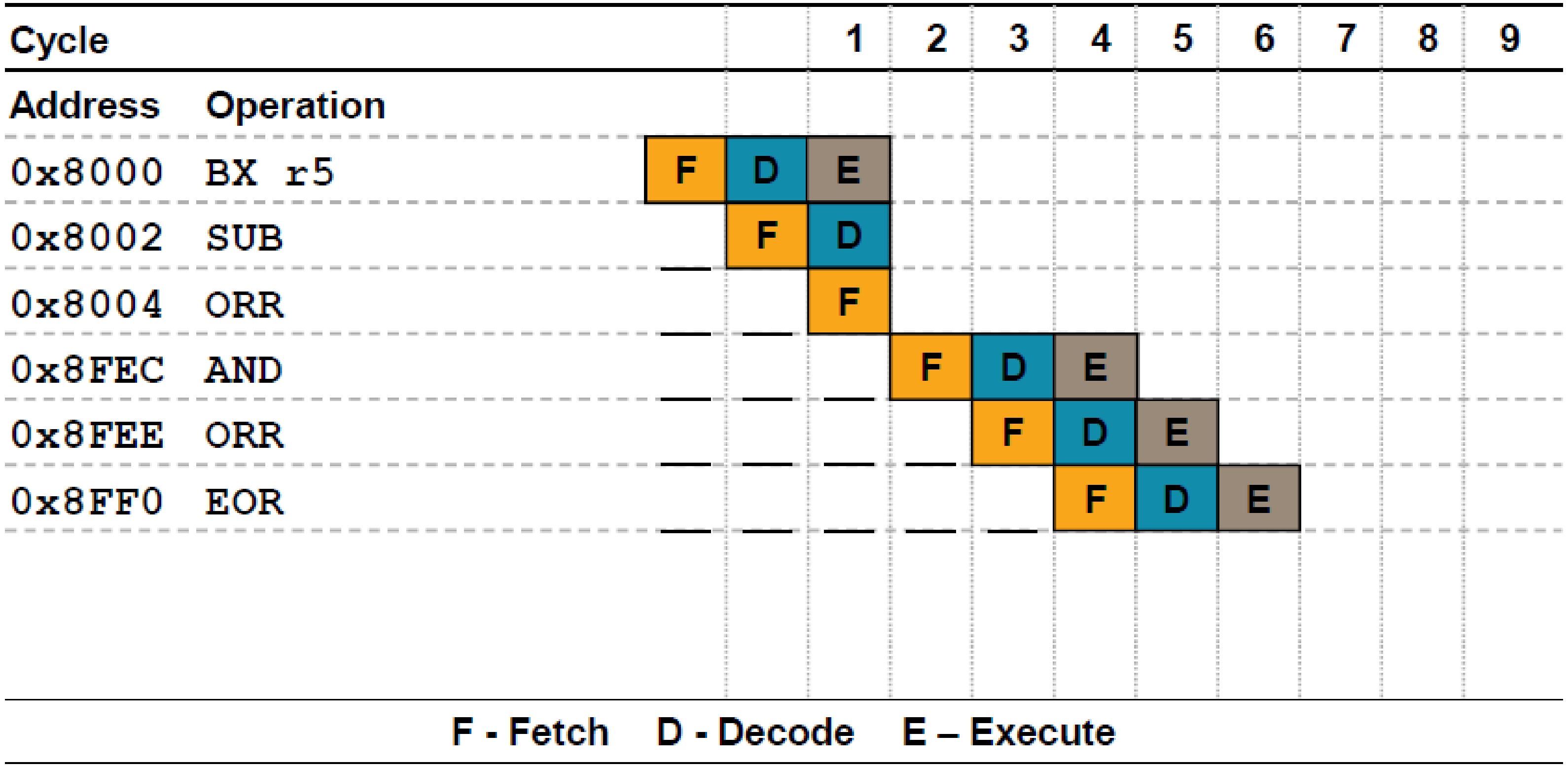
Cycle			1	2	3	4	5	6	7	8	9
Operation											
ADD	F	D	E								
SUB		F	D	E							
ORR			F	D	E						
AND				F	D	E					
ORR					F	D	E				
EOR						F	D	E			

F - Fetch D - Decode E - Execute

source: ARM
ARM Cortex-M3 Introduction
- ARM University Relations

3.2 – CORTEX-M FEATURES: PIPELINE EXAMPLE

Cortex-M4 Pipeline – pipeline flush due to indirect branch (no forwarding)



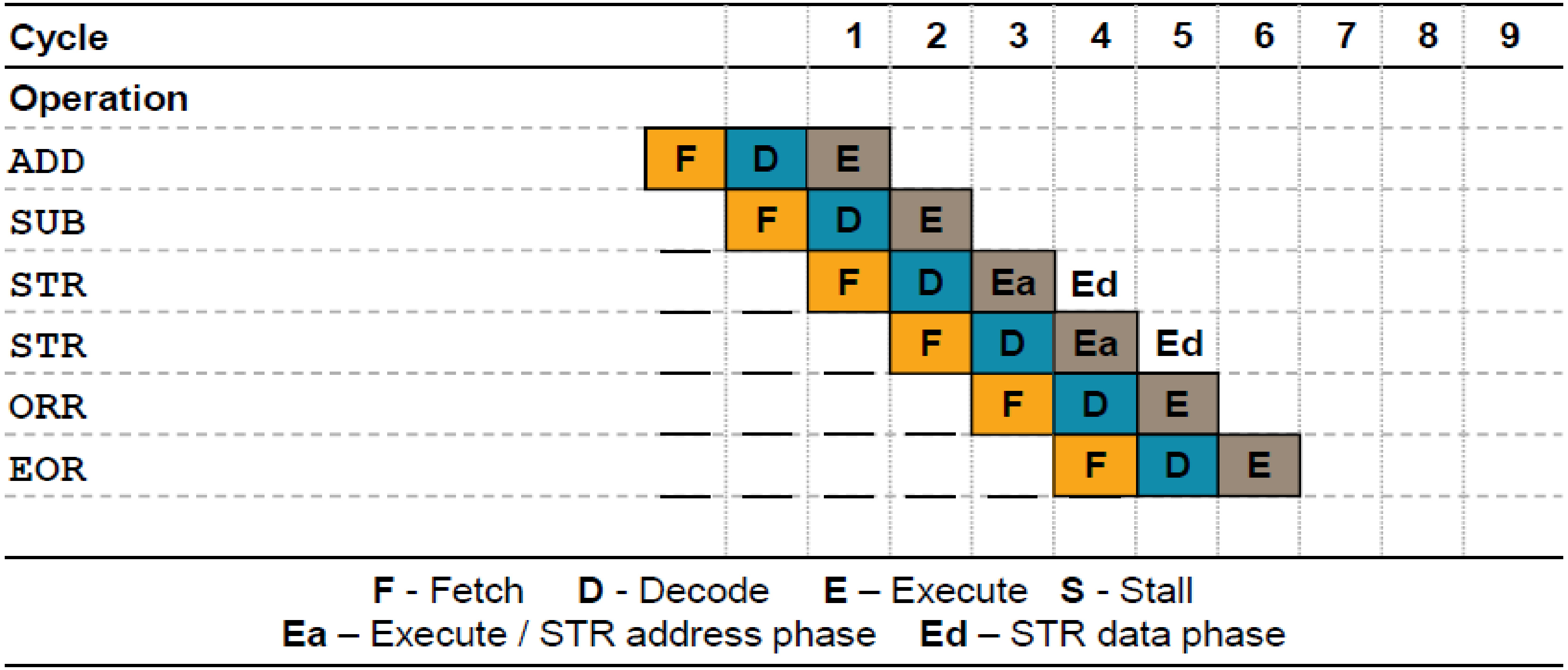
Cycle				1	2	3	4	5	6	7	8	9
Address	**Operation**											
0x8000	BX r5	F	D	E								
0x8002	SUB		F	D								
0x8004	ORR			F								
0x8FEC	AND				F	D	E					
0x8FEE	ORR					F	D	E				
0x8FF0	EOR						F	D	E			

F - Fetch **D - Decode** **E – Execute**

source: ARM
ARM Cortex-M3 Introduction
- ARM University Relations

3.2 – CORTEX-M FEATURES: PIPELINE EXAMPLE

Cortex-M4 Pipeline – Harvard architecture allows for concurrent access to code and data memory.



Cycle			1	2	3	4	5	6	7	8	9
Operation											
ADD	F	D	E								
SUB		F	D	E							
STR			F	D	Ea	Ed					
STR				F	D	Ea	Ed				
ORR					F	D	E				
EOR						F	D	E			

F - Fetch **D** - Decode **E** – Execute **S** - Stall
Ea – Execute / STR address phase **Ed** – STR data phase

source: ARM
ARM Cortex-M3 Introduction
- ARM University Relations

3.3 – CORTEX-M4 INSTRUCTION SET ARCHITECTURE

The Instruction Set Architecture (ISA) presents the Programmer’s View of the processor, including:

- Data types
- Processor Modes
- Processor Registers
- Instruction Set
- Memory Accessing
- Exception Processing

3.3 – CORTEX-M4 ISA: ADDRESSABILITY

The Cortex-M4 instruction set can operate on the following data types:

- **bit:** stores a single bit of information (0 or 1).

  Bit banding instructions can set or clear bits in specific memory regions.
- **byte:** 8-bit. Each byte in memory is individually addressable.
- **half-word:** 16-bit. The address of a half-word in memory is the address of its least significant byte.
- **word:** 32-bit. The address of a word in memory is the address of its least significant byte.
- **double-word:** 64-bit. Requires a register pair to be stored, such as R1:R0 (concatenation of R1 and R0 with R1 being the most significant word). The address of a double-word in memory is the address of its least significant byte.

Byte-addressable memory, but instructions decide how many bytes we read or write

3.3 – CORTEX-M4 ISA: ADDRESSABILITY (EXAMPLE)

```
r1 = 0x1000

LDRB r0, [r1]  # Load byte. r0 = ?

LDRH r0, [r1]  # Load half-word. r0 = ?

LDR  r0, [r1]  # Load word. r0 = ?

LDRD  r0, r2, [r1]  # Load double-word.
                    # r0 = ?
                    # r2 = ?
```

SRAM		
Address	Data	ASCII Char
⋮	⋮	⋮
0x1000	0x44	“D”
0x1001	0x41	“A”
0x1002	0x54	“T”
0x1003	0x41	“A”
0x1004	0x46	“F”
0x1005	0x4c	“L”
0x1006	0x4f	“O”
0x1007	0x57	“W”
⋮	⋮	⋮

Cortex-M has 32-bit registers, so a 64-bit value needs two registers.

3.3 – CORTEX-M4 ISA: EXECUTION MODES

The Cortex-M4 processor modes are:

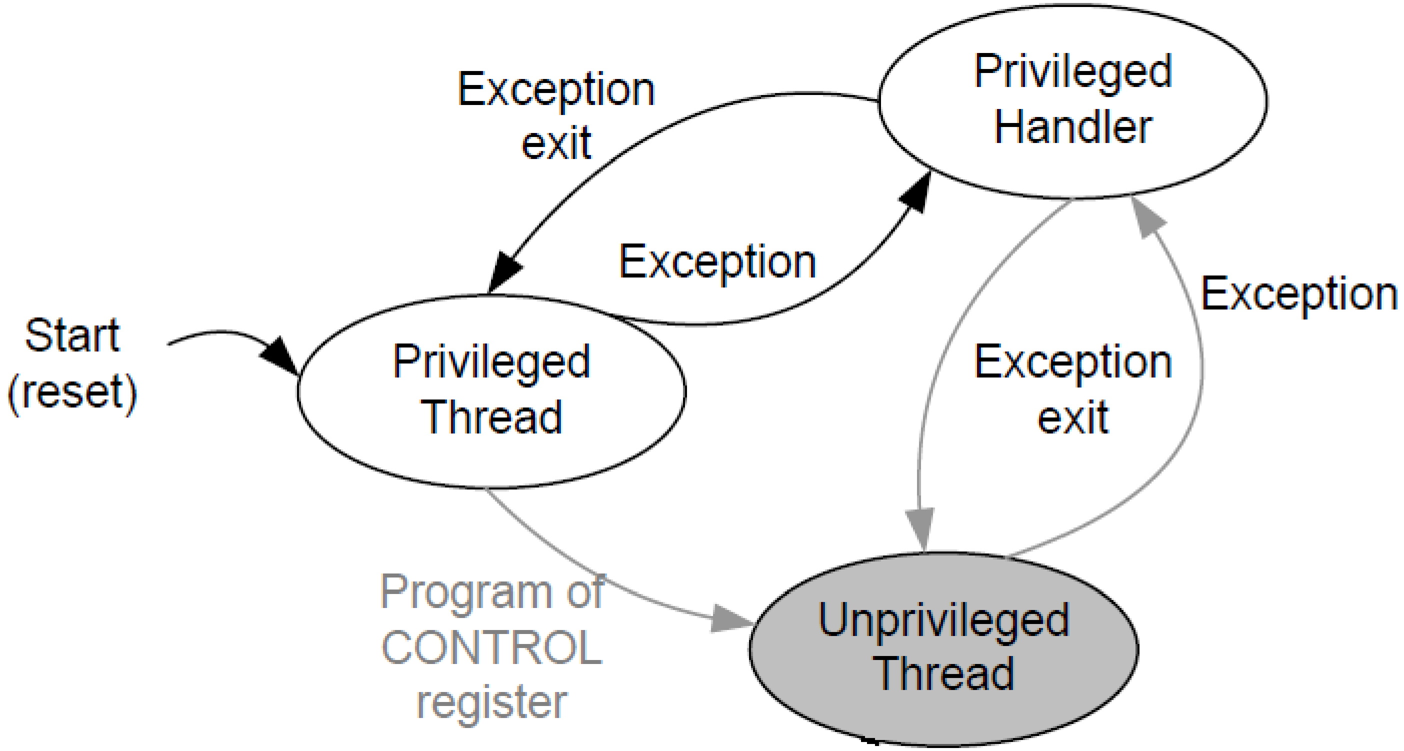
- **Thread Mode:** Normal program execution
- **Handler Mode:** Interrupt and exception handling

Privilege levels:

- **Privileged:** full access to system resources
- **Unprivileged:** restricted access

Possible Execution States:

- **Privileged Thread:** OS/kernel or system-level code
- **Unprivileged Thread:** Application code
- **Privileged Handler:** Interrupt and exception handlers



source: ARM
ARM Cortex-M for Beginners - Yiu

3.3 – CORTEX-M4 ISA: EXECUTION MODES

The Cortex-M4 has two stacks:

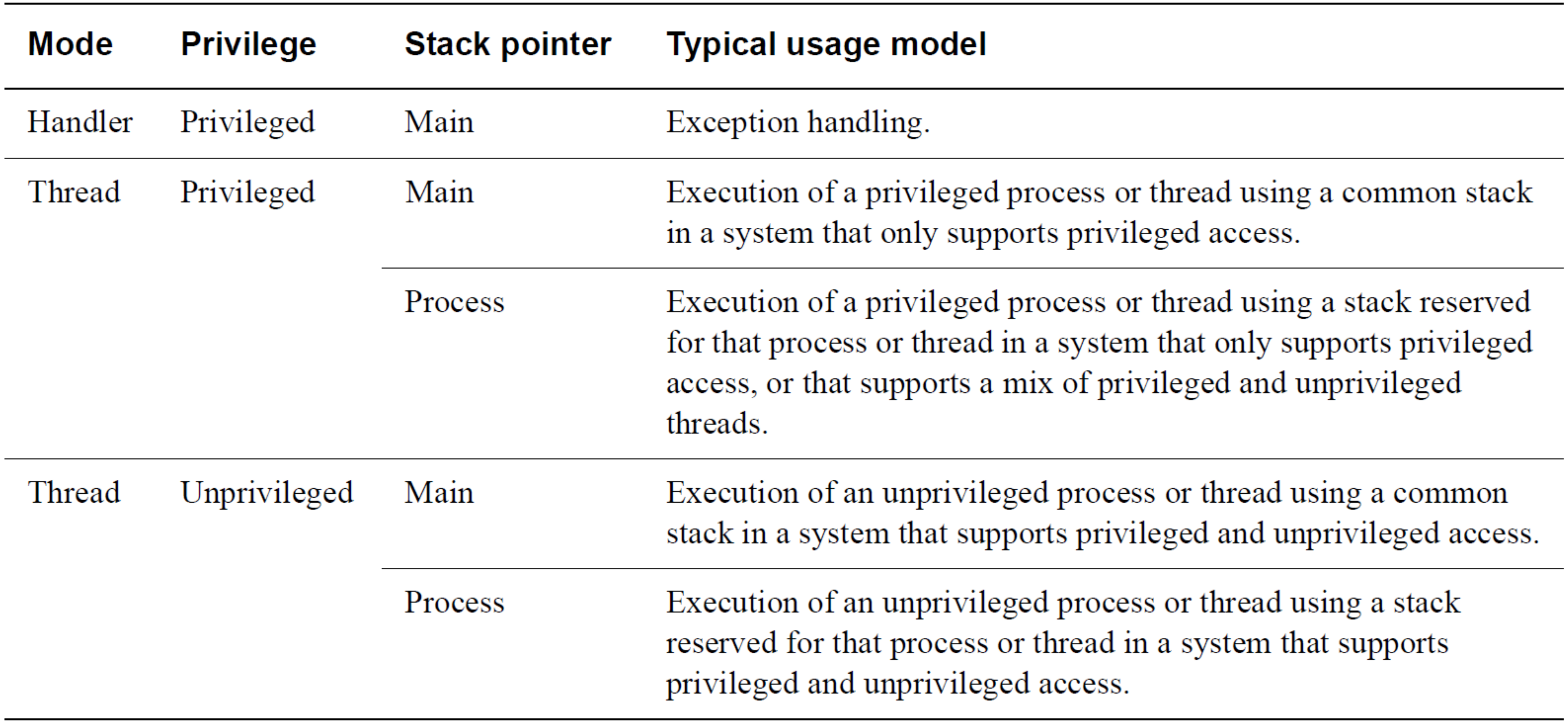
- **Main stack - addressed by MSP (Main Stack Pointer)**
  - Used automatically in Handler mode (interrupts & exceptions)
  - Typically used by the OS / kernel
- **Process stack - addressed by PSP (Process Stack Pointer)**
  - Used in Thread mode
  - Typically used by application tasks or RTOS threads

Stack selection

- Only one stack pointer is active at a time
- The CONTROL register selects the stack used in Thread mode
- R13 (SP) always refers to the currently active stack pointer

3.3 – CORTEX-M4 ISA

Table B1-1 Mode, privilege and stack relationship



Mode	Privilege	Stack pointer	Typical usage model
Handler	Privileged	Main	Exception handling.
Thread	Privileged	Main	Execution of a privileged process or thread using a common stack in a system that only supports privileged access.
		Process	Execution of a privileged process or thread using a stack reserved for that process or thread in a system that only supports privileged access, or that supports a mix of privileged and unprivileged threads.
Thread	Unprivileged	Main	Execution of an unprivileged process or thread using a common stack in a system that supports privileged and unprivileged access.
		Process	Execution of an unprivileged process or thread using a stack reserved for that process or thread in a system that supports privileged and unprivileged access.

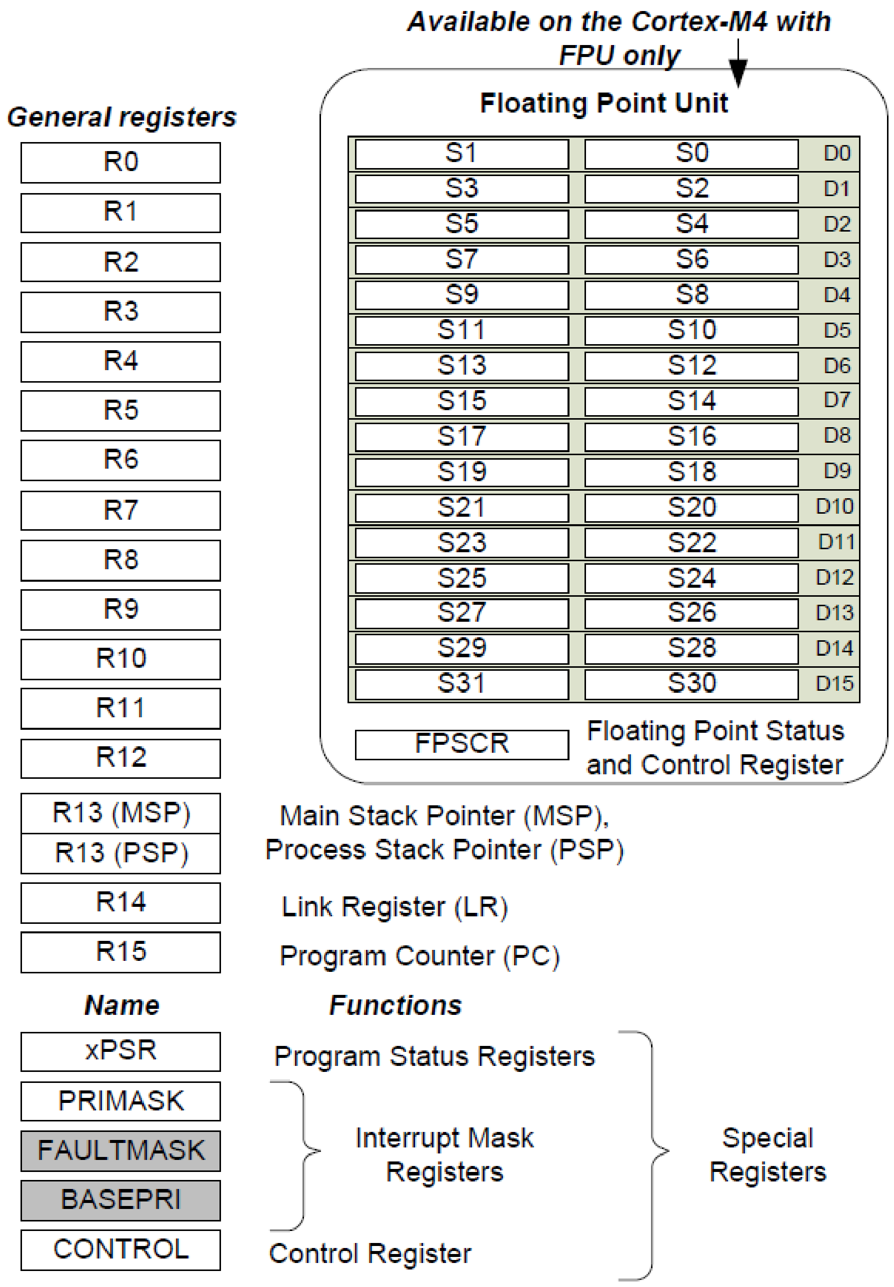
source: ARM
ARMv7-M
Architecture
Reference Manual

3.3 – CORTEX-M4 ISA – REGISTERS

The register set of the Cortex-M4 consists of:

- General Purpose Registers (R0-R15)
- Floating Point Registers (S0-S31)
- Special Registers (xPSR, PRIMASK, FAULTMASK, BASEPRI, CONTROL)

All registers are 32-bit wide. Not all bits of the Special Registers are implemented.



source: ARM
ARM Cortex-M for Beginners - Yiu

3.3 – CORTEX-M4 ISA – REGISTERS

General Purpose Registers:

- 32-bit wide
- Low registers: R0 .. R7
- High registers: R8 .. R15
- Special usage:

  R13 or SP is the Stack Pointer - points to the top of the stack

  R14 or LR is the Link Register - stores the procedure return address

  R15 or PC is the Program Counter - stores the address of the next instruction fetch

b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0

3.3 – CORTEX-M4 ISA – REGISTERS

General Purpose Registers:

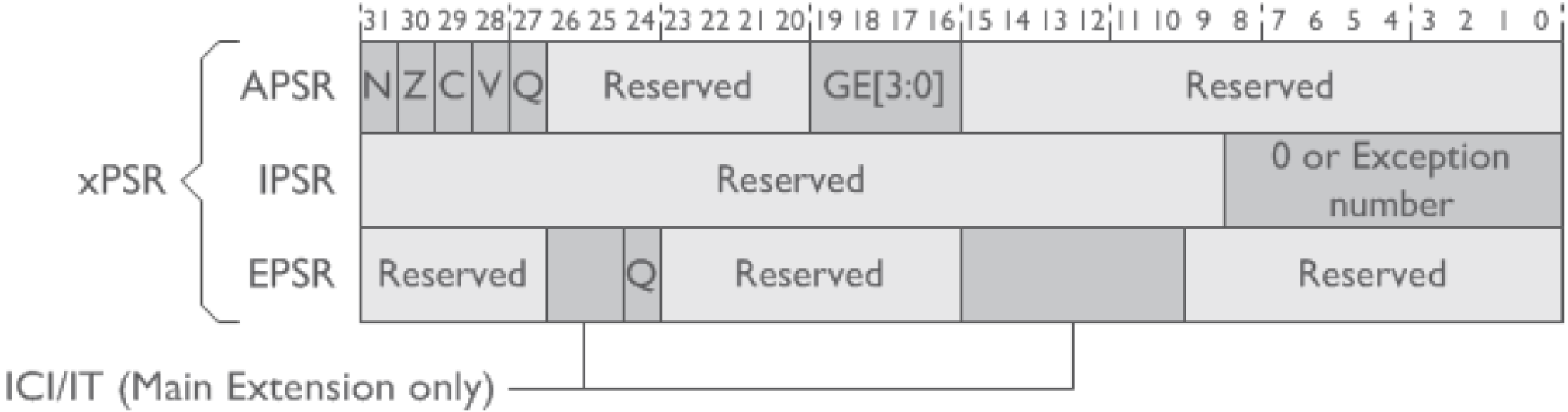
- the general-purpose registers are accessible by most instructions;
- b0 is the Least Significant Bit (LSb) and b31 is the Most Significant Bit (MSb);
- a register can hold an unsigned integer with values from 0 to 4,294,967,295
  or a signed integer with values from -2,147,483,648 to 2,147,483,647;
- when using hexadecimal notation, a 32-bit register holds 8 hex digits. E.g. 0x1234 5678;
- Cortex-M architecture uses 32-bit memory addresses, hence, a single general-purpose register can hold a memory address.

b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0

3.3 – CORTEX-M4 ISA – REGISTERS

Special Purpose Registers - XPSR

- The three registers APSR, IPSR and EPSR can be accessed individually or combined as XPSR.

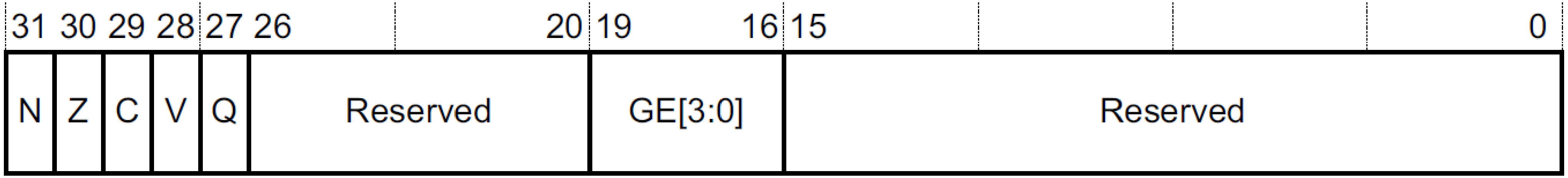


The Combined Program Status Register

source: ARM
ARMv7-M Architecture Reference Manual

3.3 – CORTEX-M4 ISA – APSR

APSR = Application Program Status Register



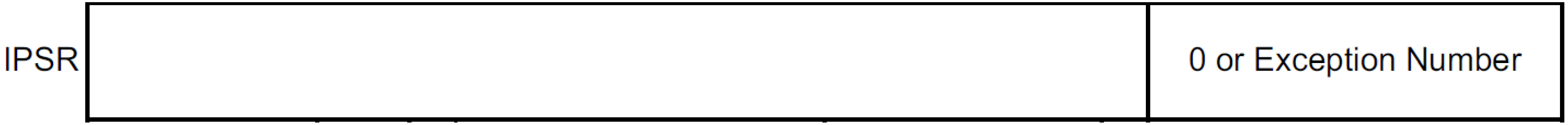
31	30	29	28	27	26 – 20	19 – 16	15 – 0
N	Z	C	V	Q	Reserved	GE[3:0]	Reserved

Flag	bit	Description
N	31	Negative. For two’s complement results this bit is set to indicate that the result is negative.
Z	30	Zero. This bit is set when the result is zero. After a comparison, this bit is set to indicate that the compared values are equal.
C	29	Carry. Set to indicate that the result of an unsigned addition produced overflow. Also set to indicate that un unsigned subtraction underflowed.
V	28	Overflow. Set to indicate overflow in a signed arithmetic operation.
Q	27	Saturation bit. For DSP extension instructions.
GE	19:16	Greater than or Equal. For SIMD instructions.

source: ARM
ARMv7-M Architecture Reference Manual

3.3 – CORTEX-M4 ISA – IPSR

IPSR = Interrupt Program Status Register



IPSR

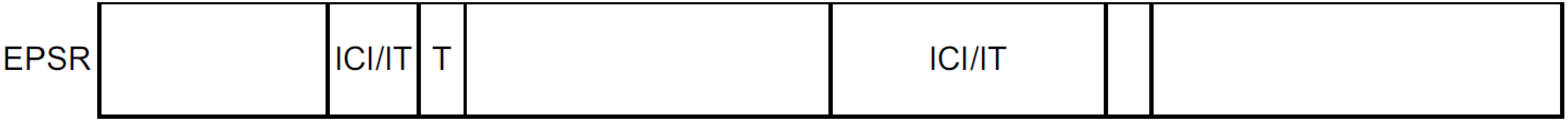
0 or Exception Number

Field	bit	Description
Exception Number	8 : 0	This field identifies the exception that is currently active (being serviced). The value 0 indicates that no exception is currently active. If this value is non-zero the processor is in Handler mode. If this value is zero the processor is in Thread mode.

source: ARM
ARMv7-M Architecture Reference Manual

3.3 – CORTEX-M4 ISA – EPSR

EPSR = Execution Program Status Register

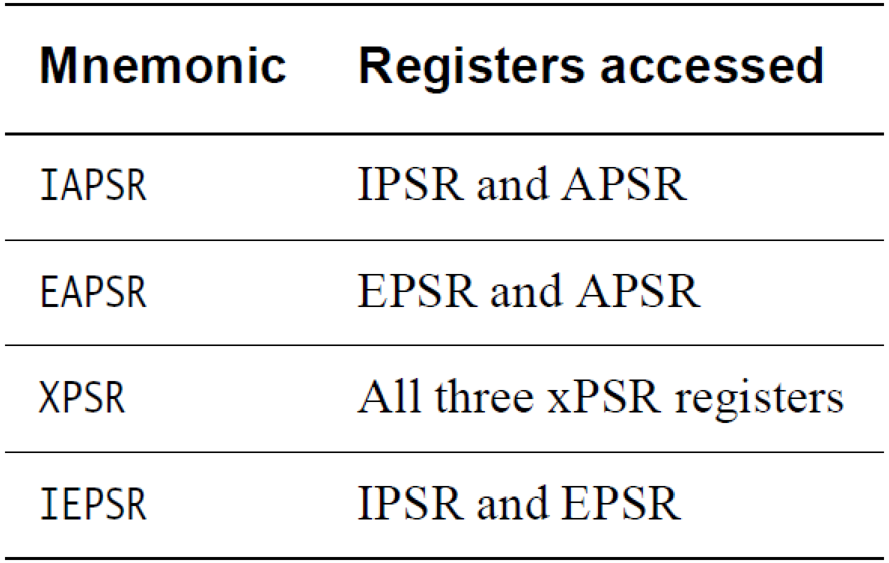


Field	bit	Description
T	24	Thumb. This bit is set to indicate that the instruction set in use is Thumb. On Cortex-M this bit must be set all the time or an exception occurs. On ARM7TDMI, Cortex-R and Cortex-A this bit is 0 when the ARM instruction set is in use.
ICI/IT	26:25 15:10	ICI - used for a interrupted exception-continuable multi-cycle load or store. IT - provide context information for instructions in an IT block.

source: ARM
ARMv7-M Architecture Reference Manual

3.3 – CORTEX-M4 ISA – XPSR

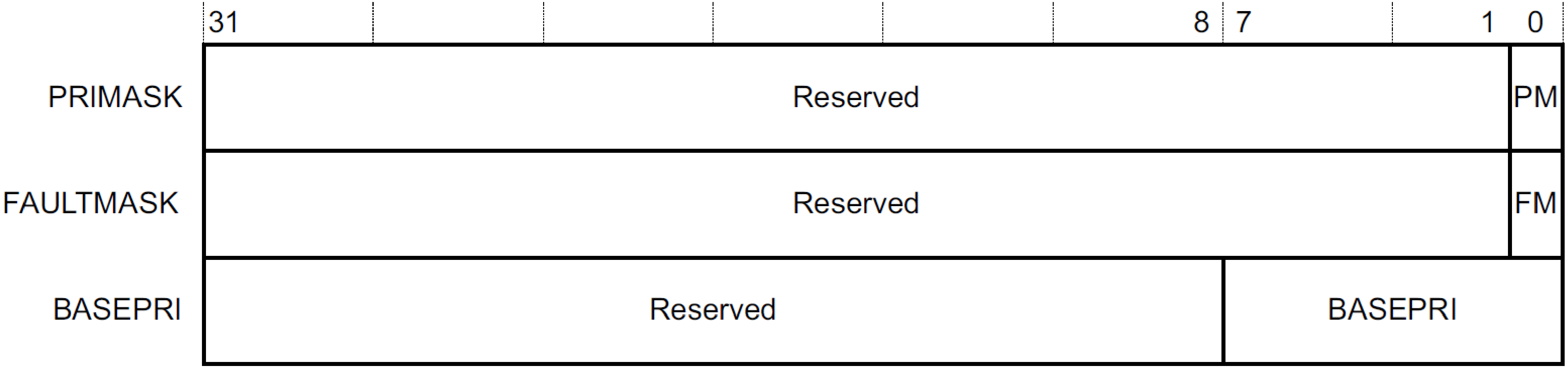
Mnemonics used to combine the XPSR component registers:



Mnemonic	Registers accessed
IAPSR	IPSR and APSR
EAPSR	EPSR and APSR
XPSR	All three xPSR registers
IEPSR	IPSR and EPSR

source: ARM
ARMv7-M Architecture Reference Manual

3.3 – CORTEX-M4 ISA – SPECIAL REGISTERS

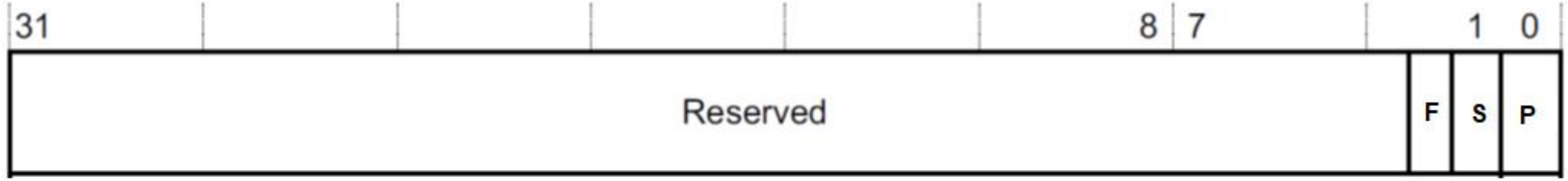


Field	bit	Description
PM	PRIMASK[0]	Set to mask exceptions with configurable priority (priority 0 and lower). Reset to unmask.
FM	FAULTMASK[0]	Set to mask the HardFault exceptions and the configurable priorities (prio -1 and lower).
BASE PRI	BASEPRI[7:0]	Changes the priority level required for exception preemption. Affects only the currently executing code with lower priority than BASEPRI.

source: ARM
ARMv7-M Architecture Reference Manual

3.3 – CORTEX-M4 ISA – CONTROL REGISTER

CONTROL Register



Field	bit	Description
nPRIV	0	When the processor is in Thread mode. 0 = privileged mode; 1 = unprivileged mode.
SPSEL	1	Stack selection. 0 = use MSP (Main Stack); 1 = use PSP (Process Stack). In Handler mode this bit is always 0.
FPCA	2	Implemented only when floating point is available. 0 = do not save floating point registers on exception; 1 = save floating point context on exception.

source: ARM
ARMv7-M Architecture Reference Manual

3.3 – CORTEX-M4 ISA – REGISTERS

Floating Point Registers:

- The Cortex-M4 with optional floating-point extension, implements 32 32-bit floating point registers named S0 to S31.
- These can be combined two by two forming 16 double precision (64-bit) floating point registers named D0 to D15. D0 is formed by S1:S0 (the concatenation of the registers S1 and S0 where S1 is the most significant word, i.e. the leftmost word).

3.4 – INSTRUCTION SET

Before presenting the instruction set, lets examine:

- Assembly syntax
- 3-operand instructions
- Conditional instructions
- Instructions that affect the flags

3.4 – INSTRUCTION SET

The most common instruction formats are: (format)

label: MNEMONIC   Destination, Operand1, Operand2   ;comment

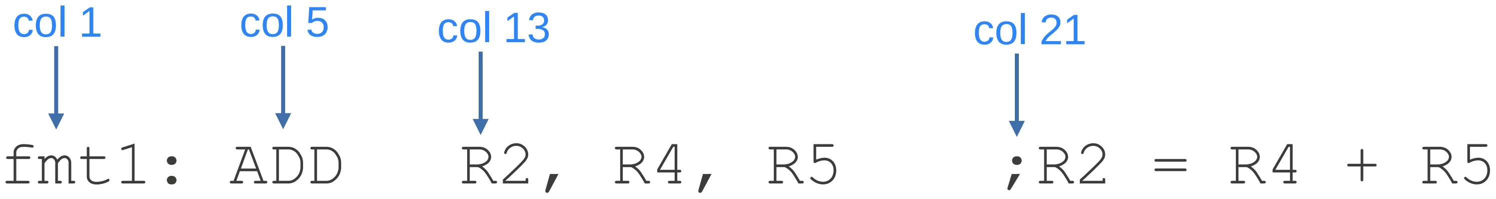
label: MNEMONIC   Destination, Operand2   ;comment

Examples:

```
fmt1: ADD  R2, R4, R5          ;R2 = R4 + R5
fmt2: ADD  R2, R4              ;R2 = R2 + R4
```

3.4 – INSTRUCTION SET

Suggested assembly source file layout:



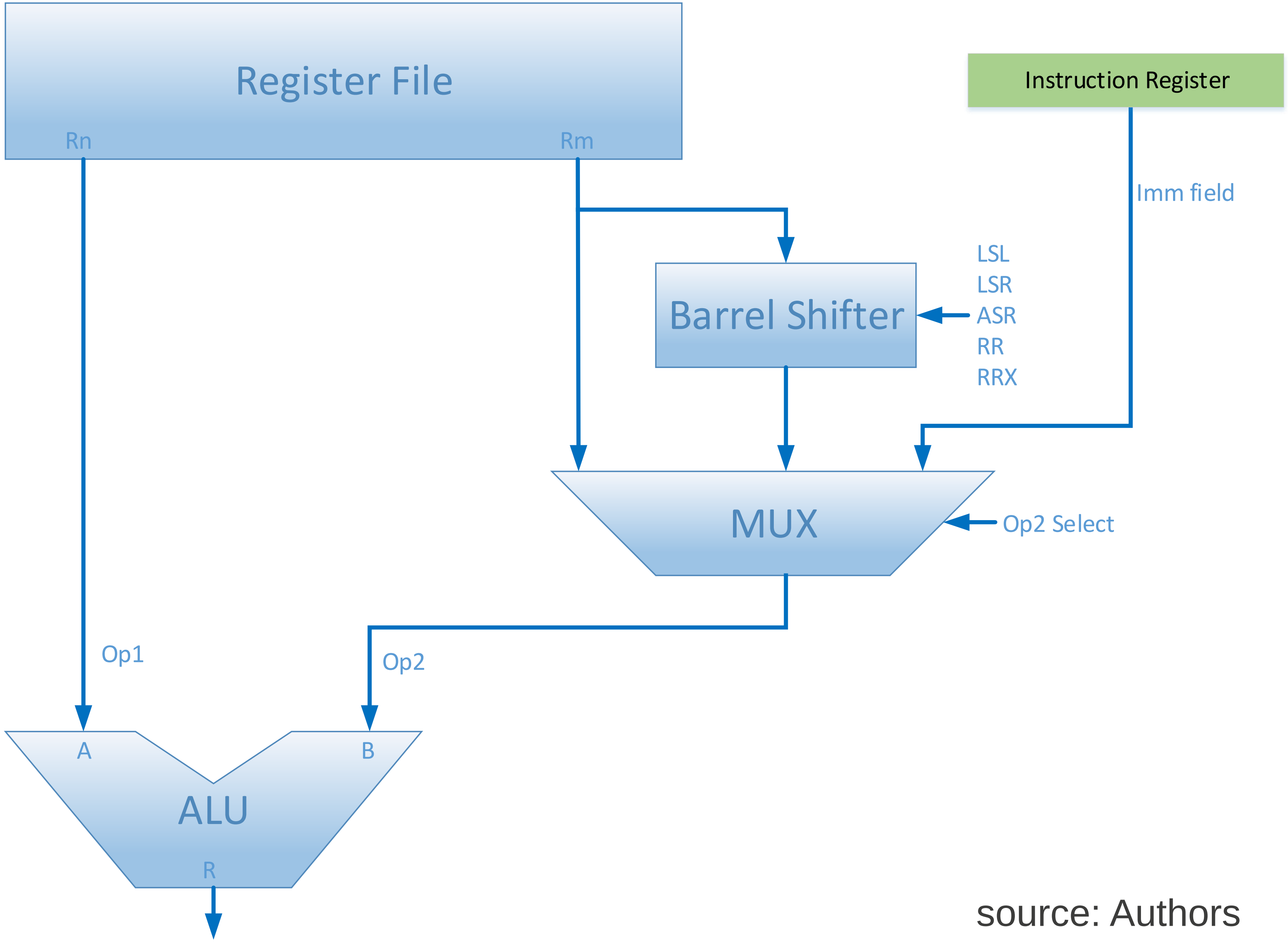
```
fmt1: ADD   R2, R4, R5     ;R2 = R4 + R5
```

By setting the TABs to 4 spaces, these positions can be easily obtainable.

Only labels should begin on column 1.

Only mnemonics are compulsory. Labels, operands and comments are optional, although they are all very frequent.

For an instruction with two operands, the first operand (Op1) is always a register. The second operand (Op2) may be a register, or a register that had its contents shifted or rotated, or an immediate value coded in the instruction.



source: Authors

3.4 – INSTRUCTION SET

Examples of Operand 2

```
ADD  R2, R4, R5          ;Operand 2 is a register (R5)

ADD  R2, R4, R5,LSL #2   ;Operand 2 is a shifted register
                         ;R5 is shifted left by 2 bits
                         ;this corresponds to multiplying its value by 4

ADD  R2, R4, #0xFF       ;Operand 2 is an immediate value
                         ;the hexadecimal value 0xFF
```

Example

```
BEFORE      r5 = 5
            r7 = 8

            MOV r7,r5, LSL #2

AFTER       r5 =
            r7 =
```

3.4 – INSTRUCTION SET

In the Cortex-M4 instruction set, the programmer explicitly controls if the result of an instruction should affect the flags: N,Z,C,V.

Most instructions have a variant with the letter S appended to the mnemonic. The S variant means: “set the flags”.

```
ADD  R2, R4, R5        ;the result of this addition does
                       ;not affect the flags.

ADDS R2, R4, R5        ;the result of this addition
                       ;affects the N,Z,C and V flags.
```

3.4 – INSTRUCTION SET

Many Cortex-M4 instruction can be **conditional**, meaning that the instruction only executes if the flags are in a given state. Except for branch instructions, an instruction must be in an IT block to be conditional. The condition is specified by two letters appended after the mnemonic (see condition table on next slide).

```
ADD    R2,R4,R5        ;the result of this addition does
                       ;not affect the flags.
ADDS   R2,R4,R5        ;the result of this addition
                       ;affects the N,Z,C and V flags.
ITT    EQ              ;start of an IT block with 2 instructions
ADDEQ  R2,R4,R5        ;if Z is set, execute the ADD
ADDSEQ R2,R4,R5        ;if Z is set, execute the ADD and change
                       ;flags according to result of this instruction
```

3.4 – INSTRUCTION SET

Condition codes mnemonics suffixes

Suffix	Flags	Meaning
EQ	Z = 1	Equal
NE	Z = 0	Not equal
CS or HS	C = 1	Higher or same, unsigned
CC or LO	C = 0	Lower, unsigned
MI	N = 1	Negative
PL	N = 0	Positive or zero
VS	V = 1	Overflow
VC	V = 0	No overflow
HI	C = 1 and Z = 0	Higher, unsigned
LS	C = 0 or Z = 1	Lower or same, unsigned
GE	N = V	Greater than or equal, signed
LT	N != V	Less than, signed
GT	Z = 0 and N = V	Greater than, signed
LE	Z = 1 and N != V	Less than or equal, signed
AL	Can have any value	Always. This is the default when no suffix is specified.

3.4 – INSTRUCTION SET

Code examples for conditional instructions.

```
        cmp     R12,R10     ;compare the unsigned values in R12 and R10, change flags
        beq     op1         ;branch to op1 if the values of R12 and R10 are equal
        ite     hi          ;two-instruction IT block with HI condition
        addhi   R12,R12,#1  ;if R12 > R10 then increment R12
        addls   R10,R10,#1  ;else increment R10
op1:
        ...
```

3.4 – INSTRUCTION SET

An immediate value is a constant whose value is encoded in the instruction. Hence, a limited range of values is allowed.

The notation for immediate values is #value.

The notation for negative values is `#-15`.

The notation for hexadecimal values is `#0xFA0`.

Example:

```
        ADD     R2,R4,#5        ;R2 = R4 + 5
```

3.4 – INSTRUCTION SET

- Cortex-M4 instruction codes are either 16-bit or 32-bit.
- 16-bit instructions are called narrow and may have a .N suffix.
- 32-bit instructions are called wide and may have a .W suffix.
- Some mnemonics may be coded either in narrow or wide format, for example:

```
0x37a: 0x1840        ADDS.N    R0, R0, R1     //16-bit code
0x37c: 0xeb10 0x0001 ADDS.W    R0, R0, R1     //32-bit code
```

- 16-bit and 32-bit instruction code can be freely intermixed in a program.
- All instructions must be halfword aligned, i.e. must be stored on an even address.

3.4 – INSTRUCTION SET

- Hence, PC will never hold an odd address => bit 0 of PC is always 0.
- When writing a 32-bit value to PC, bit 0 is ignored => can be used for other purpose.
- In other ARM processors, use bit 0 for interworking (i.e. change of instruction set).
- On Cortex-M, bit 0 must be a 1. This value is stored to the T flag in XPSR.
- Instructions that can be used for interworking (i.e. write to T flag):
  - BX
  - BLX
  - pop {PC}
- Instructions that have as destination register the PC, cause a branch
  - MOV PC, LR
  - ADD PC, PC,R1
- There are restrictions on which instructions may write to PC.

3.4 – INSTRUCTION SET

Arithmetic instructions

Instruction	Description	Action
ADD Rd, Rn, Op2	Add a register to Operand2	Rd = Rn + Op2
ADC Rd, Rn, Op2	Add a register to Operand2 and to Carry	Rd = Rn + Op2 + CY
SUB Rd, Rn, Op2	Subtract from a register the Operand2	Rd = Rn - Op2
SBC Rd, Rn, Op2	Subtract from a register the Operand2 and the Borrow (negation of Carry)	Rd = Rn - Op2 - /CY
RSB Rd, Rn, Op2	Subtract from Operand2 a register	Rd = Op2 - Rn
RSC Rd, Rn, Op2	Subtract from Operand2 a register and the Borrow	Rd = Op2 - Rn - /CY
MOV Rd, Op2	Move to Rd from Operand2 (put a copy of Operand2 into Rd)	Rd = Op2
MVN Rd, Op2	Move to Rd /Operand2	Rd = /Op2
MOVT Rd,<imm16>	Move to Rd[31:16] from imm16. Lower bits of Rd are unaffected	Rd[31:16] = imm16 Rd[15:0] unchanged

3.4 – INSTRUCTION SET

Compare and Test

Instruction	Description	
CMP Rn, Op2	Compare: Subtract from Rn the Operand2, discard result, change flags	
CMN Rn, Op2	Compare negative: Add Rn to Operand2, discard result, change flags	
TST Rn, Op2	Test: Rn AND Operand2, discard result, change flags	
TEQ Rn, Op2	Test equivalence: Rn EOR Operand2, discard result, change flags	

3.4 – INSTRUCTION SET

Logical

Instruction	Description	Action
AND Rd, Rn, Op2	AND: bitwise logical AND a register to Operand2	Rd = Rn AND Op2
ORR Rd, Rn, Op2	OR: bitwise logical OR a register to Operand2	Rd = Rn OR Op2
EOR Rd, Rn, Op2	Exclusive OR: bitwise logical XOR a register to Operand2	Rd = Rn XOR Op2
ORN Rd, Rn, Op2	OR NOT: bitwise logical OR a register to NOT(Operand2)	Rd = Rn OR /Op2

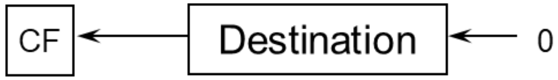
3.4 – INSTRUCTION SET

Shift Instructions

Instruction	Description	#imm Sh range
ASR Rd, Rn, Sh	Arithmetic Shift Right (preserves signal)	1..32
LSL Rd, Rn, Sh	Logical Shift Left	0..31
LSR Rd, Rn, Sh	Logical Shift Right	1..32
ROR Rd, Rn, Sh	Rotate Right	0..31
RRX Rd, Rn	Rotate Right Extended	
Remark: Sh is either	the lower 8 bits of a register (value from 0..255)	
	a 5-bit immediate value representing either 1..32 or 0..31	

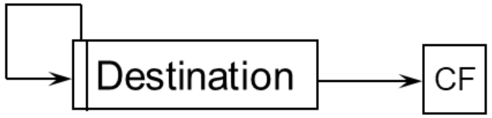
3.4 – INSTRUCTION SET

LSL : Logical Left Shift



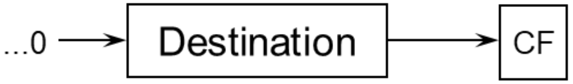
Multiplication by a power of 2

ASR: Arithmetic Right Shift



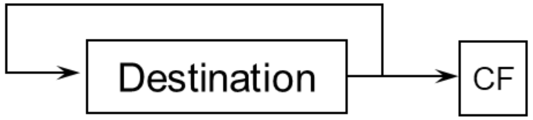
Division by a power of 2,
preserving the sign bit

LSR : Logical Shift Right



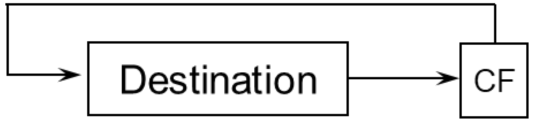
Division by a power of 2

ROR: Rotate Right



Bit rotate with wrap around
from LSB to MSB

RRX: Rotate Right Extended



Single bit rotate with wrap around
from CF to MSB

source: ARM
The ARM Architecture

3.4 – INSTRUCTION SET

Shift Operators to be used in Operand2

Operator	Description	#imm Sh range
ASR Sh	Arithmetic Shift Right (preserves signal)	1..32
LSL Sh	Logical Shift Left	0..31
LSR Sh	Logical Shift Right	1..32
ROR Sh	Rotate Right	0..31
RRX	Rotate Right Extended	
Remark: Sh is either	the lower 8 bits of a register (value from 0..255)	
	a 5-bit immediate value representing either 1..32 or 0..31	
Usage:	R4, LSL #3 (Operand2 is R4 << 3)	

3.4 – INSTRUCTION SET

Multiply

Instruction	Description	Action
Instructions that multiply 32-bit by 32-bit resulting 32-bit with wrapping (LSW is preserved and higher bits are discarded)		
MUL Rd, Rm, Rs	Multiply	Rd = Rm * Rs
MLA Rd, Rm, Rs, Rn	Multiply and accumulate	Rd = Rm * Rs + Rn
MLS Rd, Rm, Rs, Rn	Multiply and subtract	Rd = Rm * Rs - Rn
Long multiplication: multiply 32-bit by 32-bit resulting 64-bit		
UMULL RdLo, RdHi, Rm, Rs	Unsigned long multiply	RdHi:RdLo = unsigned(Rm*Rs)
UMLAL RdLo, RdHi, Rm, Rs	Unsigned long multiply and accumulate	RdHi:RdLo = unsigned(RdHi:RdLo + Rm*Rs)
UMAAL RdLo, RdHi, Rm, Rs	Unsigned long multiply and accumulate double	RdHi:RdLo = unsigned(RdHi+RdLo + Rm*Rs)
SMULL RdLo, RdHi, Rm, Rs	Signed long multiply	RdHi:RdLo = signed(Rm*Rs)
SMLAL RdLo, RdHi, Rm, Rs	Signed long multiply and accumulate	RdHi:RdLo = signed(RdHi:RdLo + Rm*Rs)

3.4 – INSTRUCTION SET

Divide

Instruction	Description	Action
UDIV Rd, Rn, Rm	Unsigned divide	Rd = Rn / Rm
SDIV Rd, Rn, Rm	Signed divide	Rd = Rn / Rm

3.4 – INSTRUCTION SET

Bit field operations

A bit field is a sequence of bits in a register.

A bit field is characterized by two values:

- Width: the number of bits in the bit field (1..32);
- lsb: the position of the least significant bit in the bitfield (0..31).

Instruction	Description	Action
BFC Rd,#<lsb>,#<width>	Bit field clear	clear Rd[(width+lsb-1)..lsb], others unchanged
BFI Rd, Rn,#<lsb>,#<width>	Bit field insert. Copy the <width> LSb of Rn to Rd	Rd[(width+lsb-1)..lsb] = Rn[(width-1)..0]
SBFX Rd, Rn,#<lsb>,#<width>	Signed bit field extract.	Copy bitfield from Rn to LSb of Rd and sign extend.
UBFX Rd, Rn,#<lsb>,#<width>	Unsigned bit field extract.	Copy bitfield from Rn to LSb of Rd and zero extend.

3.4 – INSTRUCTION SET

Memory access instructions

Instruction type	Description
LDRB	Load byte. Read a byte from memory and store in the LSB of a register.
LDRH	Load half word. Read a half-word from memory and store in the lower half-word of a register.
LDR	Load register. Read a word from memory and store in a register.
LDRD	Load double. Read a double word form memory and store in two registers.
STRB	Store byte. Store the LSB of a register into memory.
STRH	Store half-word. Store the lower half of a register into memory.
STR	Store register. Store a register into memory.
STRD	Store double. Store the two registers into memory.
LDM	Load multiple. Read several (up to 16) registers from memory.
STM	Store multiple. Store several (up to 16) registers into memory.

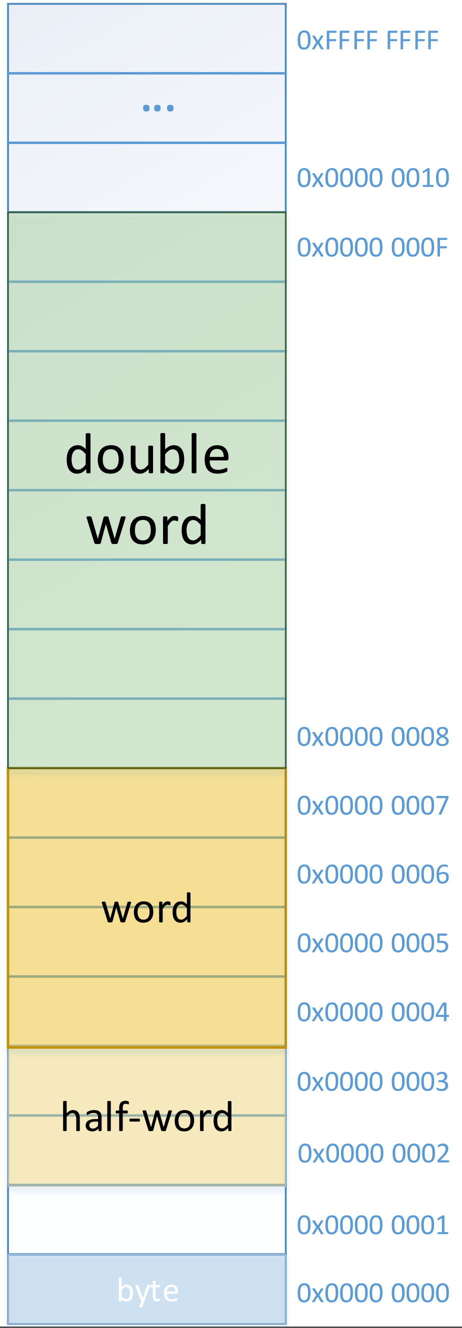
3.4 – INSTRUCTION SET

Memory access instructions – addressing

For the purposes of addressing, memory is just a very large vector of bytes. For Cortex-M, it is a vector with 4G entries.

The four data types that can be accesses with LDR/STR instructions are shown here. In a little-endian memory system, when a data type occupies more than 1 byte in memory, its address is the address of the LSB (Least Significant Byte).

shown: byte at 0x0, half-word at 0x2, word at 0x4, and double-word at 0x8



0xFFFF FFFF

…

0x0000 0010

0x0000 000F

double word

0x0000 0008

0x0000 0007

0x0000 0006

word

0x0000 0005

0x0000 0004

0x0000 0003

half-word

0x0000 0002

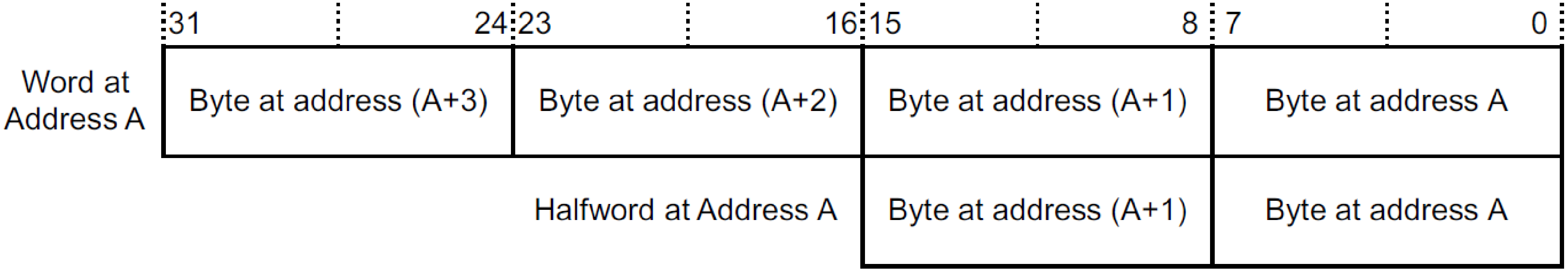
0x0000 0001

byte

0x0000 0000

source: Authors

3.4 – INSTRUCTION SET



31 24 23 16 15 8 7 0

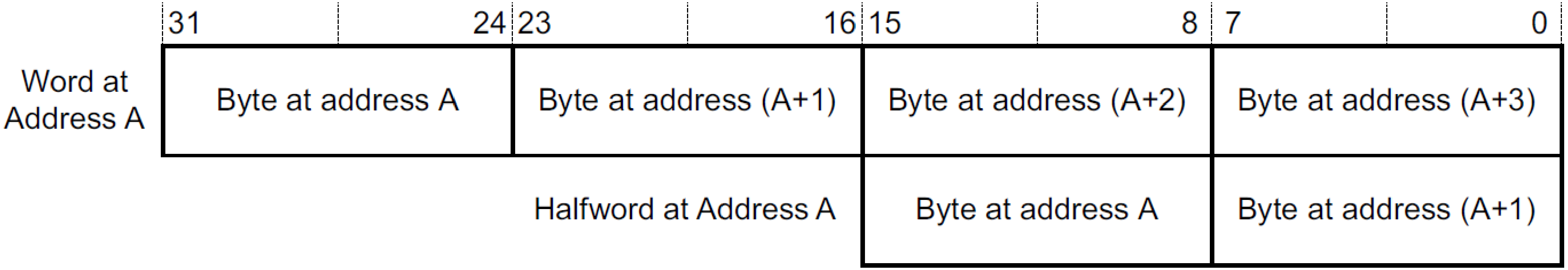
Word at Address A

Byte at address (A+3) | Byte at address (A+2) | Byte at address (A+1) | Byte at address A

Halfword at Address A

Byte at address (A+1) | Byte at address A

Figure A3-1 Little-endian byte format



31 24 23 16 15 8 7 0

Word at Address A

Byte at address A | Byte at address (A+1) | Byte at address (A+2) | Byte at address (A+3)

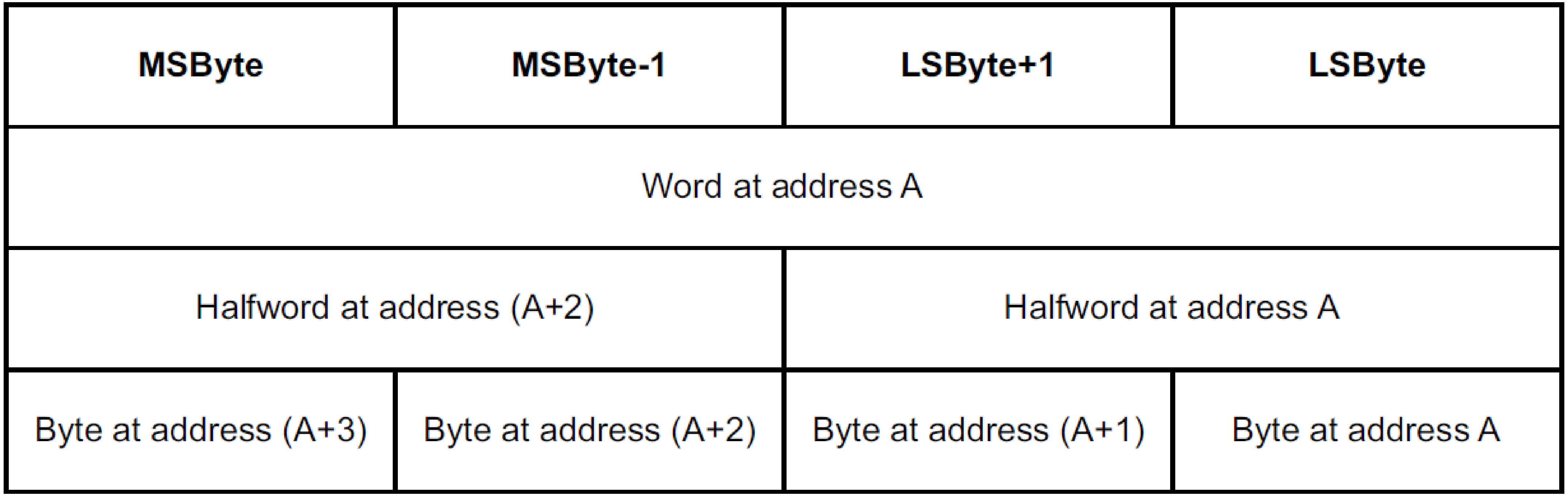
Halfword at Address A

Byte at address A | Byte at address (A+1)

Figure A3-2 Big-endian byte format

source: DDI0403E.B ARMv7-M Architecture Reference Manual

3.4 – INSTRUCTION SET



<table>
<tr><th>MSByte</th><th>MSByte-1</th><th>LSByte+1</th><th>LSByte</th></tr>
<tr><td colspan="4">Word at address A</td></tr>
<tr><td colspan="2">Halfword at address (A+2)</td><td colspan="2">Halfword at address A</td></tr>
<tr><td>Byte at address (A+3)</td><td>Byte at address (A+2)</td><td>Byte at address (A+1)</td><td>Byte at address A</td></tr>
</table>

Figure A3-3 Little-endian memory system

source: DDI0403E.B ARMv7-M Architecture Reference Manual

3.4 – INSTRUCTION SET: ADDRESSING MODES

ARM provides three addressing modes:

- Preindex with writeback
- Preindex
- Postindex

Preindex mode useful for accessing a single element in a data structure

Postindex and preindex with writeback useful for traversing an array

3.4 – INSTRUCTION SET: ADDRESSING MODES

Preindex with writeback

- Calculates address from a base register plus address offset
- Updates the address in the base register with the new address
- This is the address used to access memory
- Example: LDR r0, [r1, #4]!

Example

```
BEFORE     r0 = 0x00000000
           r1 = 0x00009000
           mem32[0x00009000] = 0x01010101
           mem32[0x00000004] = 0x02020202

           LDR r0, [r1, #4]!

AFTER      r0 =
           r1 =
```

3.4 – INSTRUCTION SET: ADDRESSING MODES

Preindex with writeback

- Calculates address from a base register plus address offset
- Updates the address in the base register with the new address
- This is the address used to access memory
- Example: LDR r0, [r1, #4]!

Example

```
BEFORE     r0 = 0x00000000
           r1 = 0x00009000
           mem32[0x00009000] = 0x01010101
           mem32[0x00000004] = 0x02020202

           LDR r0, [r1, #4]!

AFTER      r0 = 0x02020202
           r1 = 0x00009004
```

3.4 – INSTRUCTION SET: ADDRESSING MODES

Preindex

- Same as preindex with writeback, but does not update the base register

- Example: LDR r0, [r1, #4]

Example

```
BEFORE     r0 = 0x00000000
           r1 = 0x00009000
           mem32[0x00009000] = 0x01010101
           mem32[0x00000004] = 0x02020202

           LDR r0, [r1, #4]!

AFTER      r0 = 0x02020202
           r1 = 0x00009000
```

3.4 – INSTRUCTION SET: ADDRESSING MODES

Postindex

- Only updates the base register after the address is used

- Example: LDR r0, [r1], #4

Example

```
BEFORE     r0 = 0x00000000
           r1 = 0x00009000
           mem32[0x00009000] = 0x01010101
           mem32[0x00000004] = 0x02020202

           LDR r0, [r1, #4]!

AFTER      r0 = 0x01010101
           r1 = 0x00009004
```

3.4 – INSTRUCTION SET

Memory access instructions – addressing modes

Indexing Mode	Example	Action	Change in base register
Pre-index with writeback (!)	LDR R0,[R1,#4] !	R0 = [R1 + 4] (R0 gets the contents of memory location at address R1+4)	R1 = R1 + 4
	LDR R0,[R1,R2] !	R0 = [R1+R2]	R1 = R1 + R2
	LDR R0,[R1,R2,LSL #2] !	R0 = [R1 + (R2 << 2)]	R1 = R1 + R2 << 2
Pre-index	LDR R0,[R1,#4]	R0 = [R1 + 4]	no change
	LDR R0,[R1,R2]	R0 = [R1+R2]	no change
	LDR R0,[R1,R2,LSL #2]	R0 = [R1 + (R2 << 2)]	no change
Pre-index	LDR R0,[R1],#4	R0 = [R1]	R1 = R1 + 4
	LDR R0,[R1],R2	R0 = [R1]	R1 = R1 + R2
	LDR R0,[R1],R2,LSL #2	R0 = [R1]	R1 = R1 + R2 << 2

3.4 – INSTRUCTION SET

Execution flow control instructions

Instruction	Usage	Branch Range
B.N <label>	16-bit Branch to target address.	-256 to 254 bytes
B.W <label>	32-bit Branch to target address.	+/–1 MB
CBNZ <label> *CBZ* <label>	Compare and Branch on Nonzero. Compare and Branch on Zero.	0-126 B
BL <label>	Call a subroutine.	+/–16 MB
BLX <register>	Call a subroutine, optionally change instruction set.	Any
BX <register>	Branch to target address, optionally change instruction set.	Any
TBB	TBB: Table Branch, byte offsets.	0-510 B
TBH	TBH: Table Branch, halfword offsets.	0-131070 B

3.4 – INSTRUCTION SET

Miscellaneous instructions

Instruction	Usage
CPSID	Change Processor State, Disable Interrupts.
CPSIE	Change Processor State, Enable Interrupts.
DMB	Data Memory Barrier.
DSB	Data Synchronization Barrier.
ISB	Instruction Synchronization Barrier.
MRS	Move to Register from Special Register.
MSR	Move to Special Register from Register.
NOP	No Operation.
SVC	Supervisor Call.
WFI	Wait for Interrupt.

3.5 – EXCEPTIONS

The normal flow of execution of a program is to execute the next instruction in memory, unless a Branch, Subroutine Call or Return is executed. Hence, a human processor could execute the same program in the same order.

An exception if a break in this normal flow of execution. Such a break can be caused by:

- Hardware interrupt,
- Fault (e.g. memory access error, divide by 0, invalid instruction code),
- Software generate exception.

3.5 – EXCEPTIONS

Exceptions occur **asynchronousl**y, this is, at any point of the execution. They may occur many time and on successive executions of the program they usually occur at different places of this program.

When an exception occurs it must be **serviced**. Meaning that a software routine must either respond to the interrupt request or take steps to resolve or mitigate the fault.

This routine is called: **exception handler routine, interrupt service routine,** or **interrupt handler**.

3.5 – EXCEPTIONS – INTERRUPTS

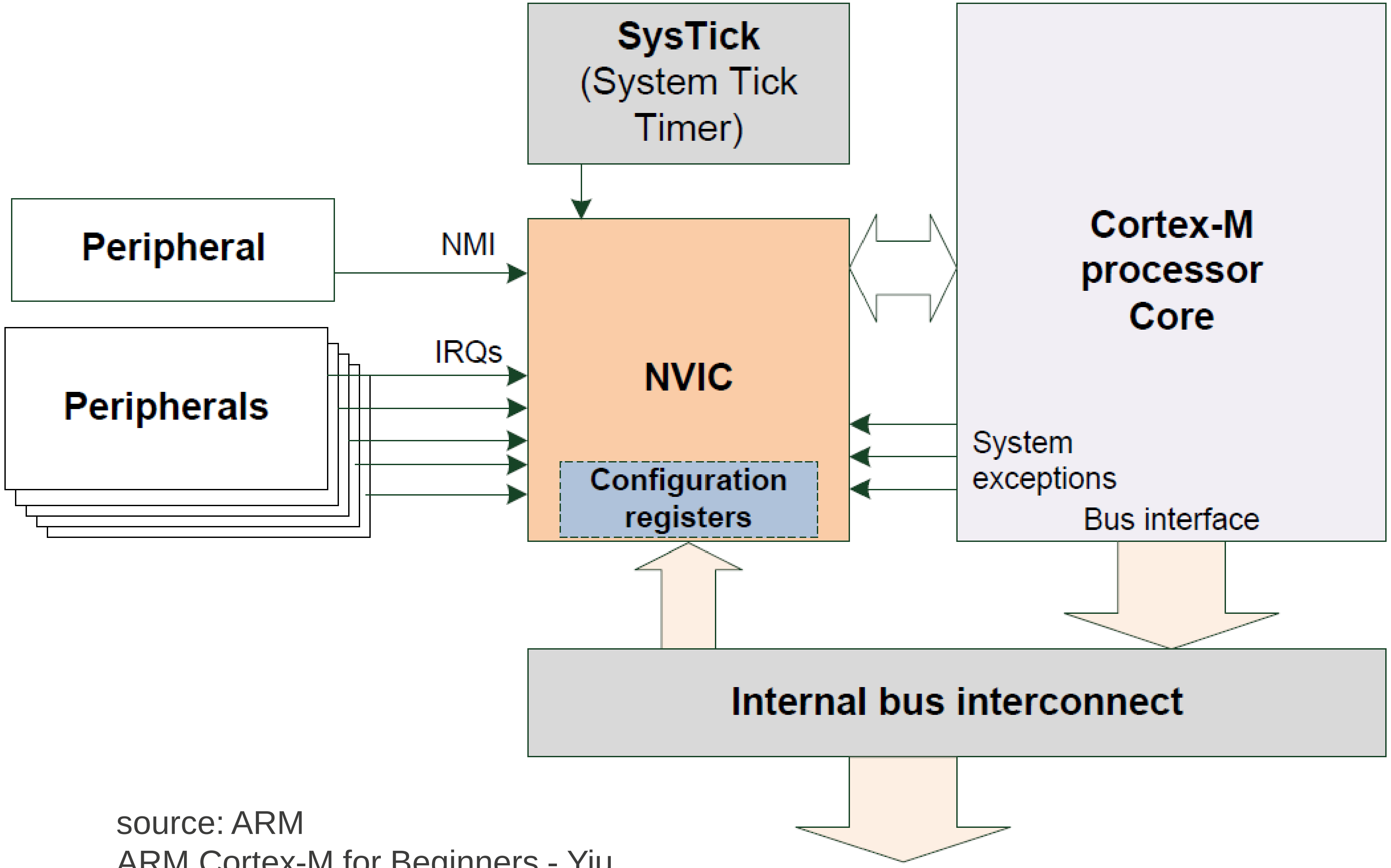
Hardware Interrupts are one of the kind of exceptions.

Interrupts are an efficient way for a peripheral to inform the processor that it requires servicing. If interrupts were not available, the processor would have to periodically poll the peripherals (thus termed **polling**) to check if service is required. Polling is an inefficient technique.

3.5 – EXCEPTIONS – INTERRUPTS

Basic concepts of interrupts:

1. Peripheral sends an Interrupt Request (IRQ) to an Interrupt Controller
2. Interrupt Controller selects the highest priority non-masked interrupt request and informs the core.
3. If the priority of the IRQ is sufficiently high, when the instruction currently in execution finishes then the IRQ is serviced.



source: ARM
ARM Cortex-M for Beginners - Yiu

3.5 – EXCEPTIONS – INTERRUPTS – DETAILED PROCESS

1- An external device, such as a peripheral, requests an interrupt (IRQ) by signaling to the interrupt controller.

The input lines of the interrupt controller (240 in the NVIC of a Cortex-M4) can be either level sensitive of edge sensitive.

3.5 – EXCEPTIONS – INTERRUPTS – DETAILED PROCESS

2- Upon receiving an IRQi (hardware signal on input i of the interrupt controller - IC) then the IC performs two checks:

a) if input i is masked or not;

b) if there is another request (IRQj on input j) already being sent to the processor.

If IRQi is not masked and if its priority is higher than IRQj's priority (or no request is currently being sent to the processor)

then IRQi is forwarded to the processor.

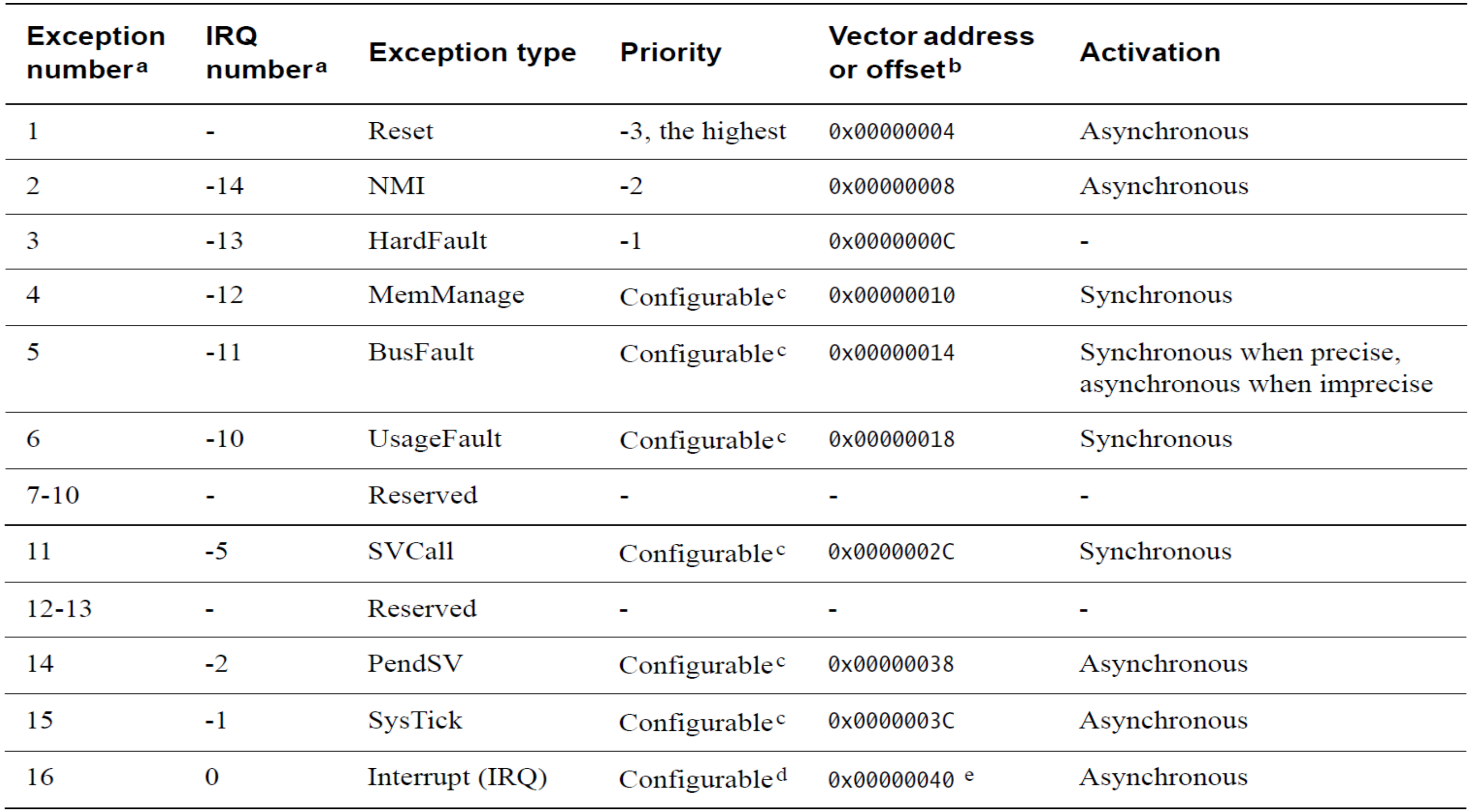
3.5 – EXCEPTIONS – INTERRUPTS – DETAILED PROCESS

3- The processor, upon receiving an IRQ verifies if its priority is sufficiently high:

a) PRIMASK and FAULTMASK, when set, impose a priority level of 0 or -1 respectively. Hence, when FAULTMASK is set, all exceptions from 3 on are masked.

b) If an exception is active (being serviced) then only a higher priority exception may preempt its handler.

If both these conditions are met, servicing starts at the end of the current instruction.



Exception number[a]	IRQ number[a]	Exception type	Priority	Vector address or offset[b]	Activation
1	-	Reset	-3, the highest	0x00000004	Asynchronous
2	-14	NMI	-2	0x00000008	Asynchronous
3	-13	HardFault	-1	0x0000000C	-
4	-12	MemManage	Configurable[c]	0x00000010	Synchronous
5	-11	BusFault	Configurable[c]	0x00000014	Synchronous when precise, asynchronous when imprecise
6	-10	UsageFault	Configurable[c]	0x00000018	Synchronous
7-10	-	Reserved	-	-	-
11	-5	SVCall	Configurable[c]	0x0000002C	Synchronous
12-13	-	Reserved	-	-	-
14	-2	PendSV	Configurable[c]	0x00000038	Asynchronous
15	-1	SysTick	Configurable[c]	0x0000003C	Asynchronous
16	0	Interrupt (IRQ)	Configurable[d]	0x00000040 [e]	Asynchronous

source: ARM
Cortex™-M4 Devices Generic User Guide

3.5 – EXCEPTIONS – INTERRUPTS – DETAILED PROCESS

4- Eight registers are pushed onto the Stack

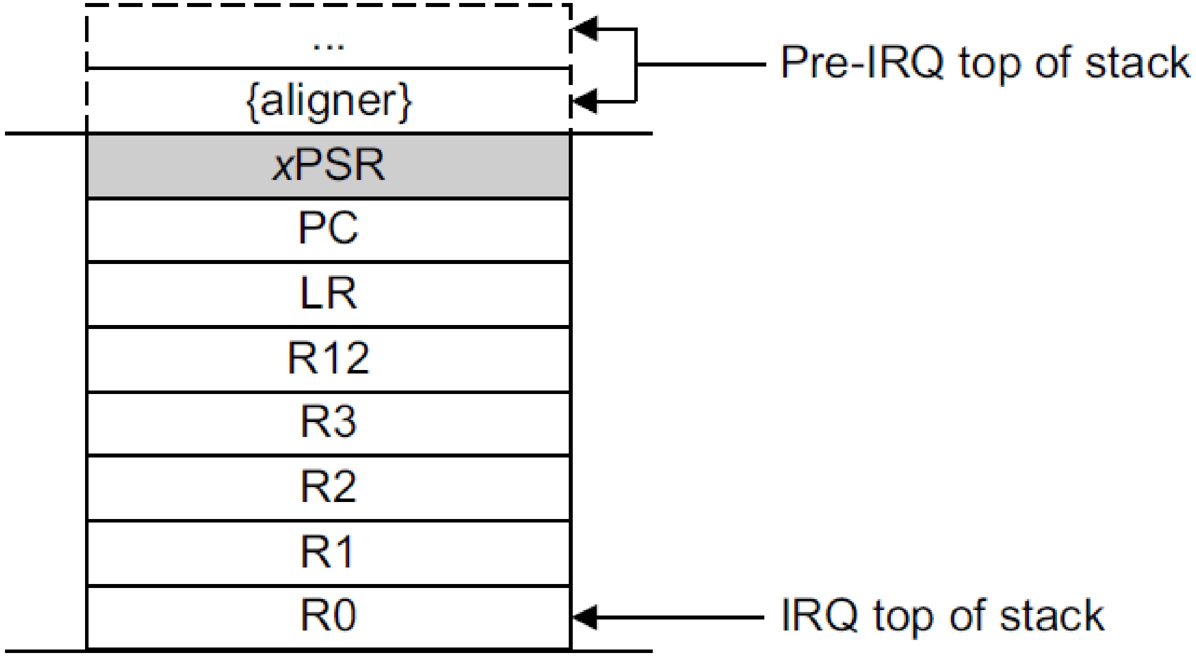
The Interrupt changes state to

Active.

Since R0-R3 and R12 are stacked,

any C procedure following ATPCS

can be registered as a handler.

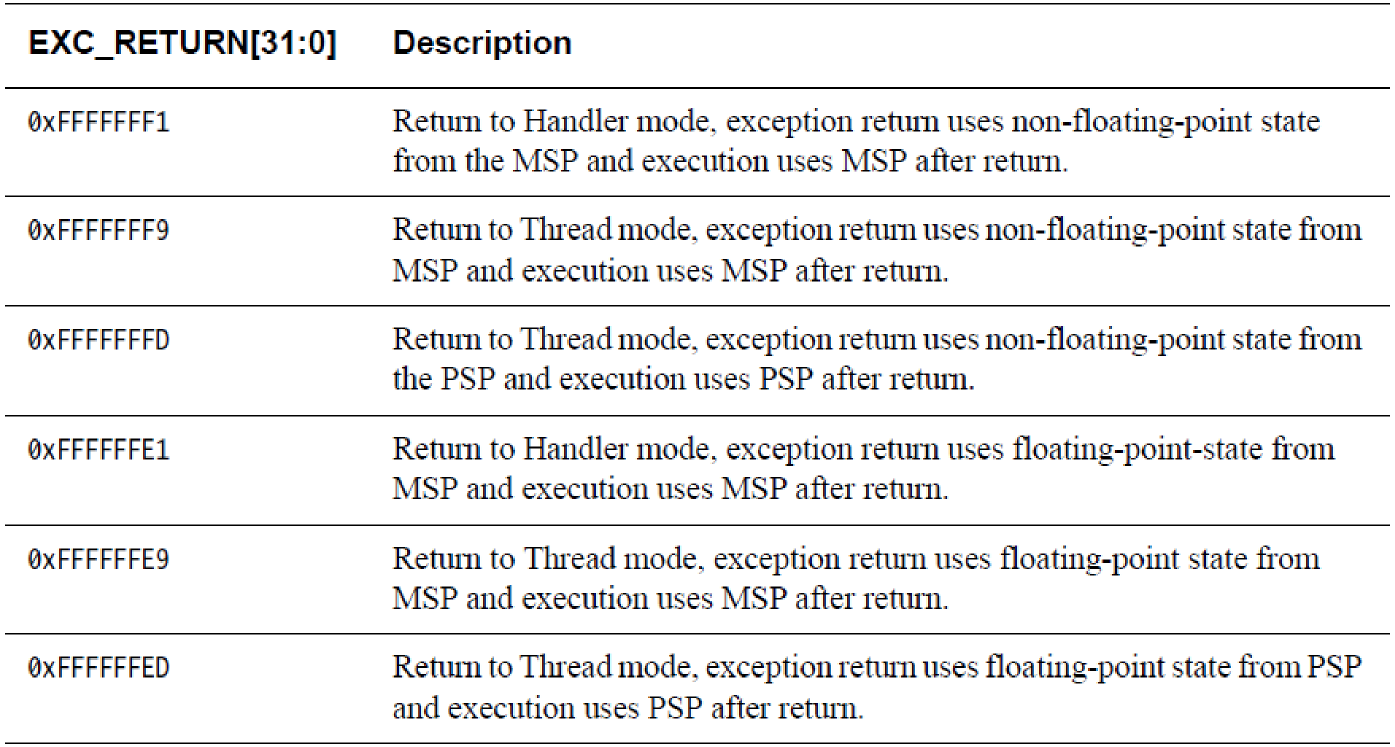


Exception frame without
floating-point storage

source: ARM
Cortex™-M4 Devices Generic User Guide

3.5 – EXCEPTIONS – INTERRUPTS – DETAILED PROCESS

5- Register LR is loaded with one of the EXC_RETURN values, depending on the current state of the processor. Note that EXC_RETURN represents memory addresses in a region where code is not allowed.



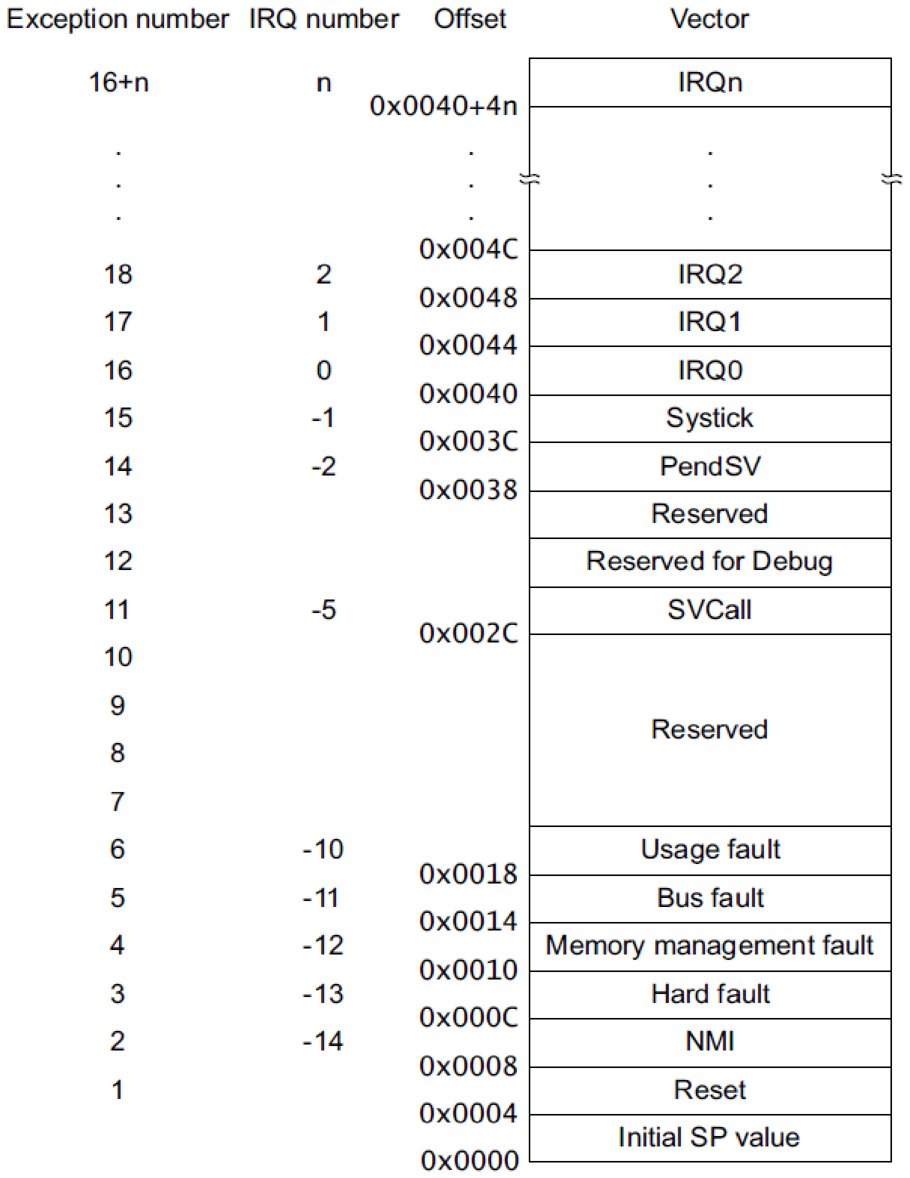
EXC_RETURN[31:0]	Description
`0xFFFFFFF1`	Return to Handler mode, exception return uses non-floating-point state from the MSP and execution uses MSP after return.
`0xFFFFFFF9`	Return to Thread mode, exception return uses non-floating-point state from MSP and execution uses MSP after return.
`0xFFFFFFFD`	Return to Thread mode, exception return uses non-floating-point state from the PSP and execution uses PSP after return.
`0xFFFFFFE1`	Return to Handler mode, exception return uses floating-point-state from MSP and execution uses MSP after return.
`0xFFFFFFE9`	Return to Thread mode, exception return uses floating-point state from MSP and execution uses MSP after return.
`0xFFFFFFED`	Return to Thread mode, exception return uses floating-point state from PSP and execution uses PSP after return.

source: ARM
Cortex™-M4 Devices Generic User Guide

3.5 – EXCEPTIONS – INTERRUPTS – DETAILED PROCESS

6- The processor reads from the vector table the initial address of the handler for the Interrupt. This value is loaded to PC and the execution of the handler starts.

Important: since the handler is Thumb-2 code the addresses in the vector table must have its LSb set to 1.



Exception number	IRQ number	Offset	Vector
16+n	n	0x0040+4n	IRQn
. . .		. . . 0x004C	. . .
18	2	0x0048	IRQ2
17	1	0x0044	IRQ1
16	0	0x0040	IRQ0
15	-1	0x003C	Systick
14	-2	0x0038	PendSV
13			Reserved
12			Reserved for Debug
11	-5	0x002C	SVCall
10			Reserved
9			
8			
7			
6	-10	0x0018	Usage fault
5	-11	0x0014	Bus fault
4	-12	0x0010	Memory management fault
3	-13	0x000C	Hard fault
2	-14	0x0008	NMI
1		0x0004	Reset
		0x0000	Initial SP value

source: ARM
Cortex™-M4 Devices Generic User Guide

Figure 2-2 Vector table

3.5 – EXCEPTIONS – SERVICING

Exception Servicing

The handler must service the interrupt request, in this process at least three actions must be taken:

1. The interrupt request signal must be deactivated, otherwise the processor would continuously be servicing this interrupt.
2. Any volatile data (such as a byte that arrived on the UART and is available in the Receiving Register) must be saved.
3. Apart from the registers saved in Step 4 of the entry process, any other register must be saved by the handler before modifying it and these registers must be restored before returning to the interrupted code.

3.5 – EXCEPTIONS – RETURN

Exception Return

The instruction that causes the return from the handler to the interrupted code is either:

`BX LR` or

```
POP {...,PC}    // if this instruction is used the the entry of the handler must be PUSH {...,LR}
```

3.5 – EXCEPTIONS – RETURN

Exception Return

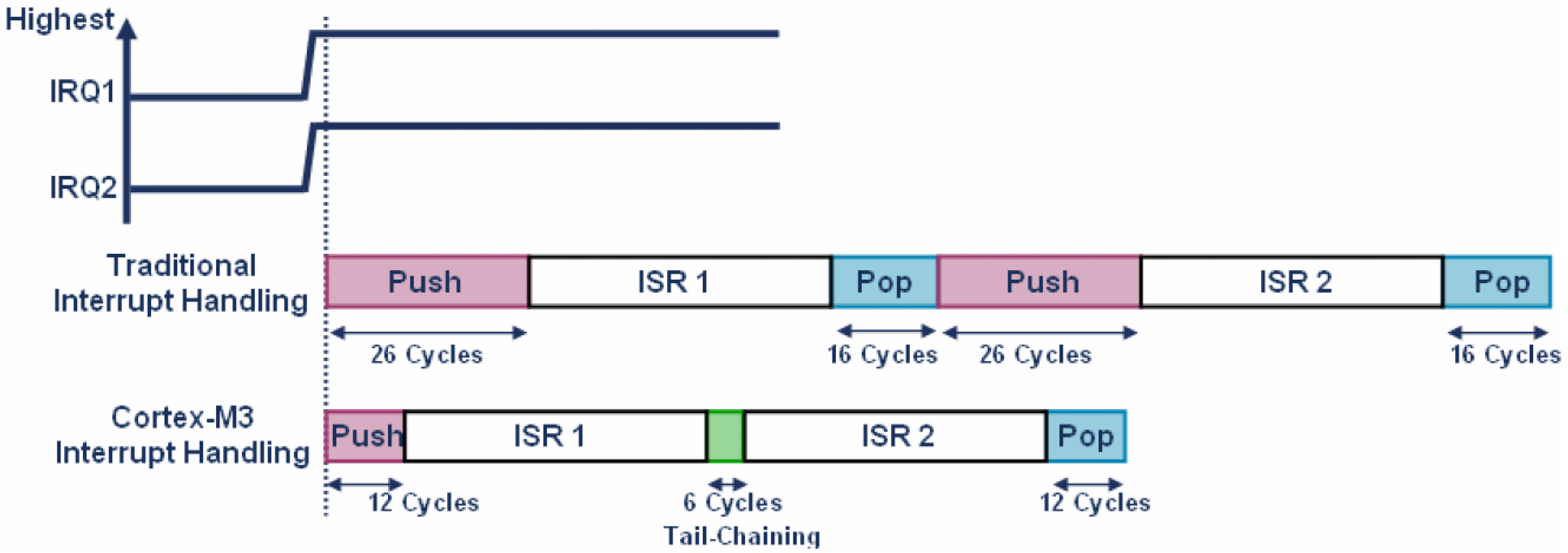
What happens when a value such as 0xFFFF FFF1 is loaded to the PC?

Being an invalid code address, the processor detects that this is an EXC_RETURN code and proceeds with the actions described in the table in the slide Step 5 of the Interrupt Entry.

3.5 – EXCEPTION HANDLING

Tail Chaining: optimization that avoids registers pop followed by registers push when one exception is handled right after another.

Figure 7. Tail chaining in the NVIC

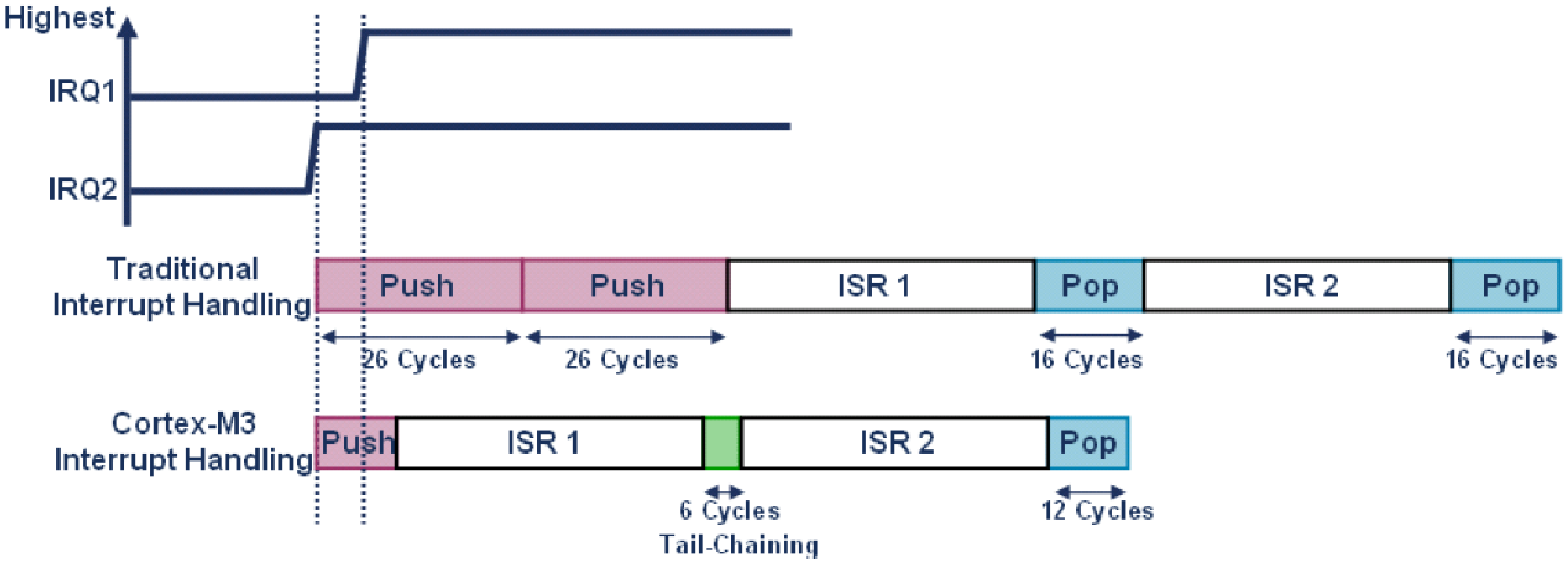


source: ARM
An Introduction to the ARM Cortex-M3 Processor - Shyam Sadasivan

3.5 – EXCEPTION HANDLING

Late arrival: optimization where a higher priority interrupt is serviced first even if it arrives while a prior lower priority interrupt is already in the stage of pushing registers.

Figure 11. Response of the NVIC to late arrival of higher priority interrupts



source: ARM
An Introduction to the ARM Cortex-M3 Processor - Shyam Sadasivan

4 – MEMORY

- Introduction
- Non-Volatile Memory
- Static RAM
- Dynamic RAM

MEMORY – INTRODUCTION

Semiconductor Memory

- Electronic components that store information.
- Memory is an essential part of a microprocessor-based system.

MEMORY – INTRODUCTION

Types of Memory devices:

- **Volatile:** memory devices that do not retain information when power is removed.

  Example: PC main memory

- **Non-volatile:** memory devices that retain information even when not powered.

  Example: flash drive

MEMORY – INTRODUCTION

Types of Volatile Memory:

- **Static:** retain information as long as the device is powered.
- **Dynamic:** do not retain information, even when powered. Hence, dynamic memories do need to be constantly “remembered” of the information they store. This process is called refresh. It must occur every few milliseconds in order not to loose information.

TYPES OF NON-VOLATILE MEMORY

Some of the types of non-volatile memory are:

- ROM: (or masked ROM) - Read Only Memory,
- EPROM: Erasable Programmable Read Only Memory,
- EEPROM: Electrically Erasable Programmable Read Only Memory,
- NOR Flash,
- NAND Flash,
- FeRAM: Ferro Electric Random Access Memory.

TYPES OF VOLATILE MEMORY

Some of the types of volatile memory are:

- SRAM: Static Random Access Memory
- DRAM: Dynamic Random Access Memory
- DDR: Double Data Rate

MEMORY – CONCEPT

A memory device behaves like an array in C.

The memory device that is pictured here has size 2^{20} and every addressable location can store 8 bits. The three control lines indicate when the memory is being addressed, if it is being read or written.

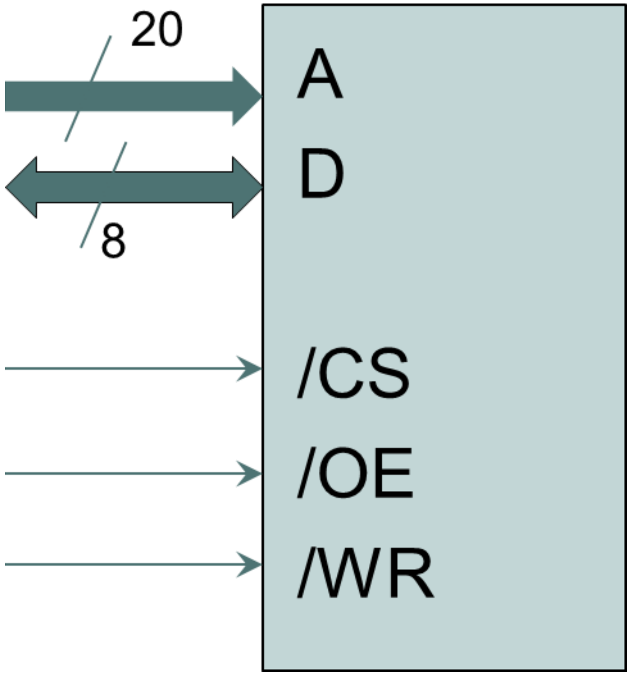
A – address bus. Input. With N address lines it is possible to index 2^N unique locations.

D – data bus. Bidirectional. Width corresponds to the number of bits stored in each addressable location.

/CS – active low chip select. When active, this device is selected.

/OE – active low output enable. When active, indicates the device is being read.

/WR – active low write. When active, indicates the device is being written to.

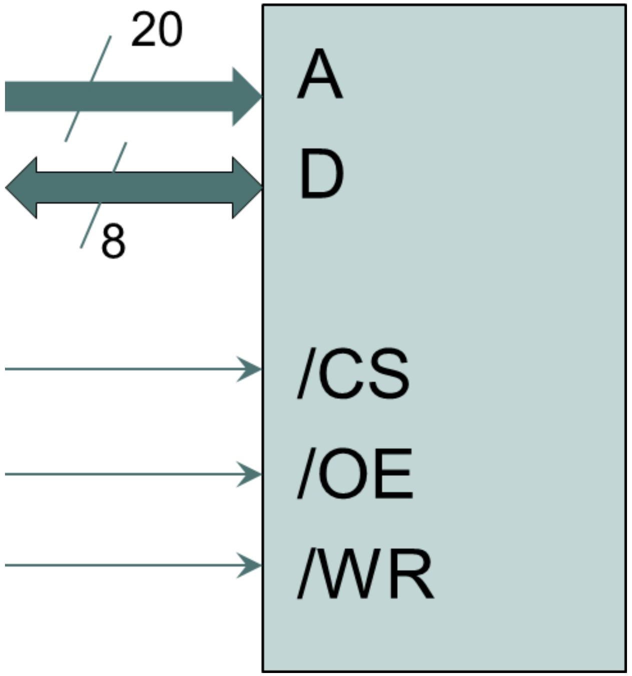


source: Authors

MEMORY – CONCEPT

A memory device behaves like an array in C

Physical view

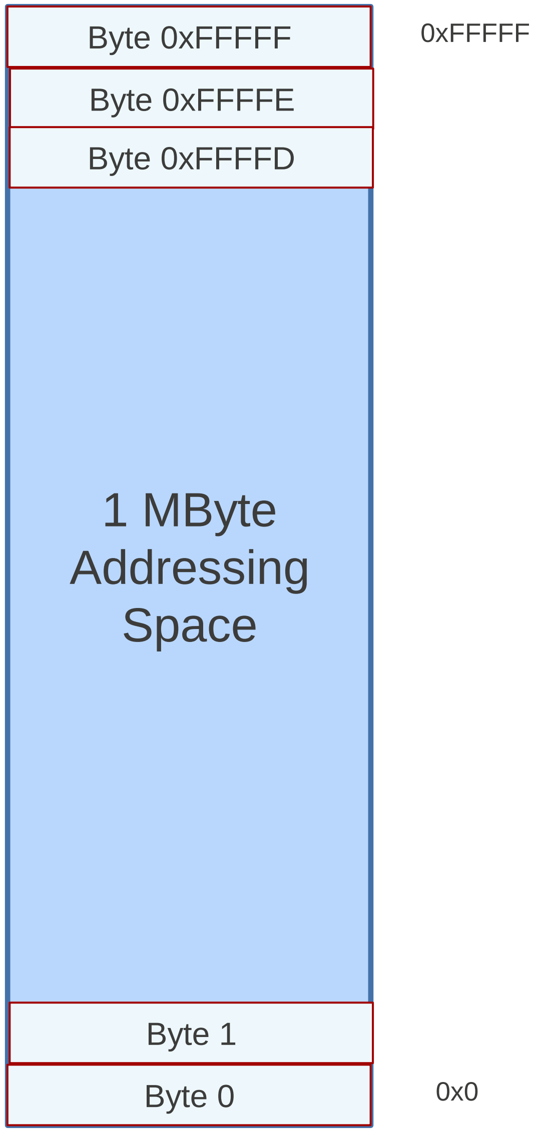


2^{20} = 1,048,576
= 1 Mega
addressable locations

each location holds 8 bits

hence, total storage is 1 Mbyte corresponding to a vector of 1,048,576 8-bit cells

Logical view



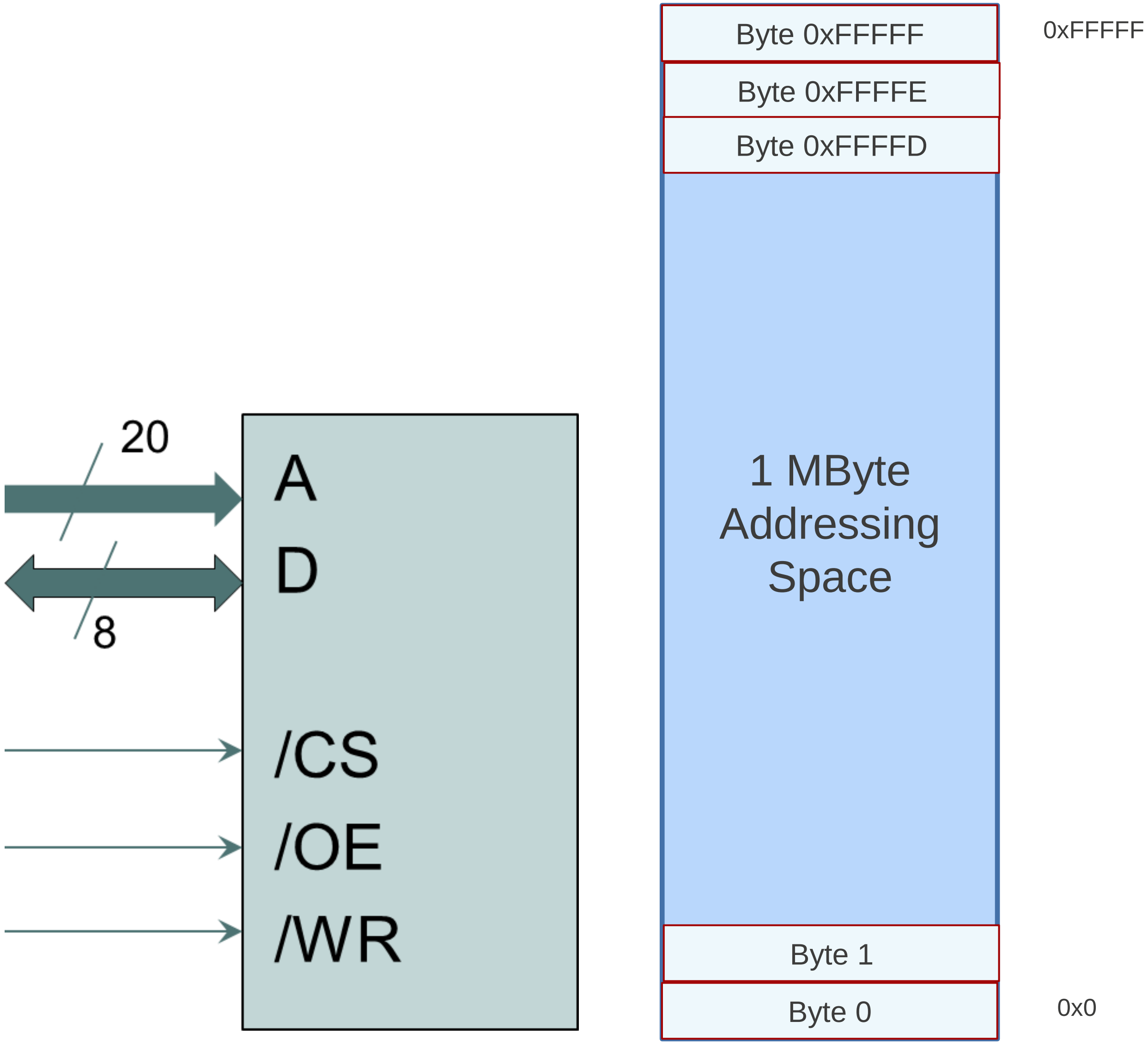
source: Authors

MEMORY – CONCEPT

From an external perspective, e.g. the perspective of the processor. This device is an array of 2^{20} locations of 8 bits each.

Hence, the size of the memory is 1 MByte (1,048,576) and the width is 8 bits. Or, 1 M x 8 bits resulting in 8 Mbits of total storage capacity.

Internally the organization is different. Possibly this memory is organized as a matrix of 1024 line and 1024 columns with 8 bits in each position. Resulting in the capacity of 1024 x 1024 x 8 = 8 Mbits.



source: Authors

SAMPLE PROBLEM

Assume the only memory chip available is 1 M x 8 bits. How to implement 2M x 32bits = 8 MBytes of memory to a Cortex-M4 with data bus width of 32-bits?

Memory should be located from address 0x1000 0000 on.

SAMPLE PROBLEM

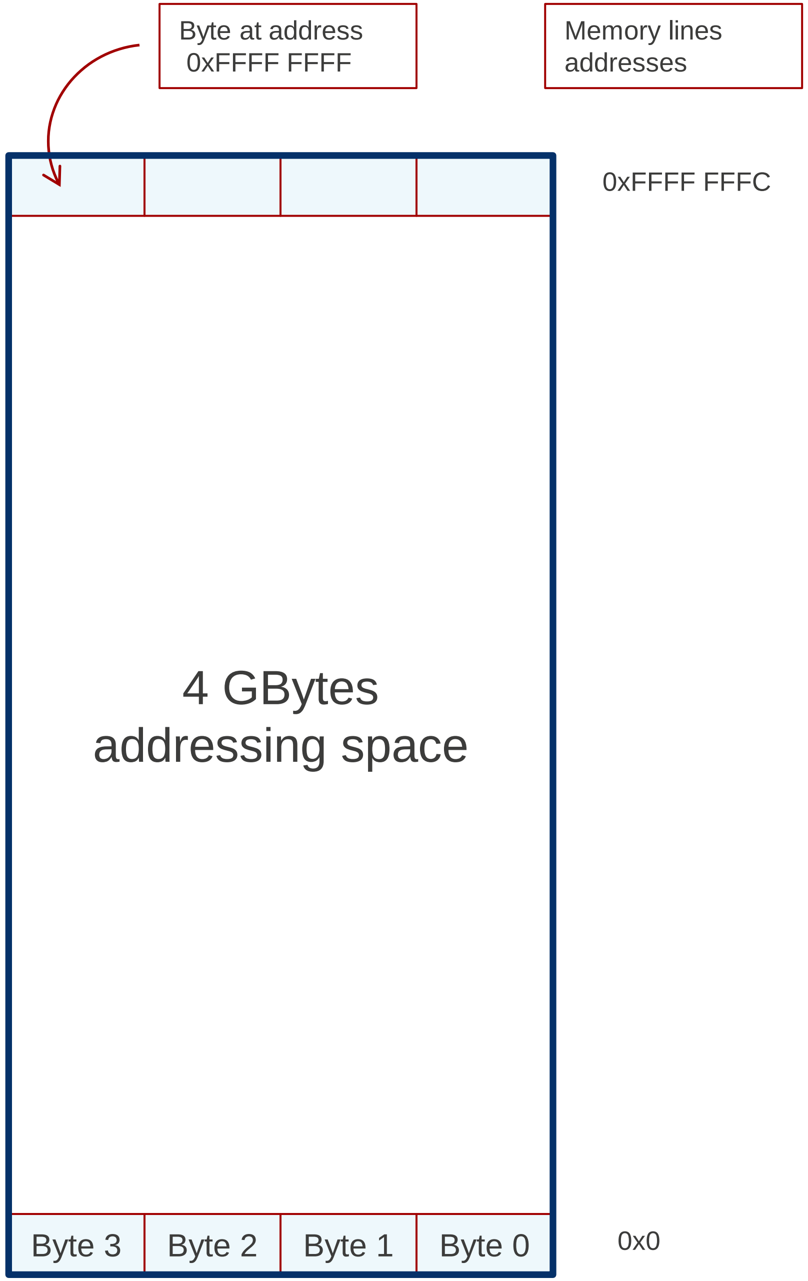
The addressing space of a Cortex-M is 4 GBytes, i.e. 2^{32} = 4,294,967,296.

If physical memory is 32-bit wide (4 bytes) then there are 1 G lines (1,073,741,824). Each of these lines is addressable by its LSByte address.

Hence, 30 address lines are required to select a single memory line (as 2^{30} = 1 G)

The address lines A31-A2 are used to select on of the 1G lines while address lines A1 and A0 are used to select a byte in the memory line

Memory address lines have their A1-A0 addresses set to 0.
Hence, the first memory line is at address 0x0000 0000, the second at address 0x0000 0004 and so on.



source: Authors

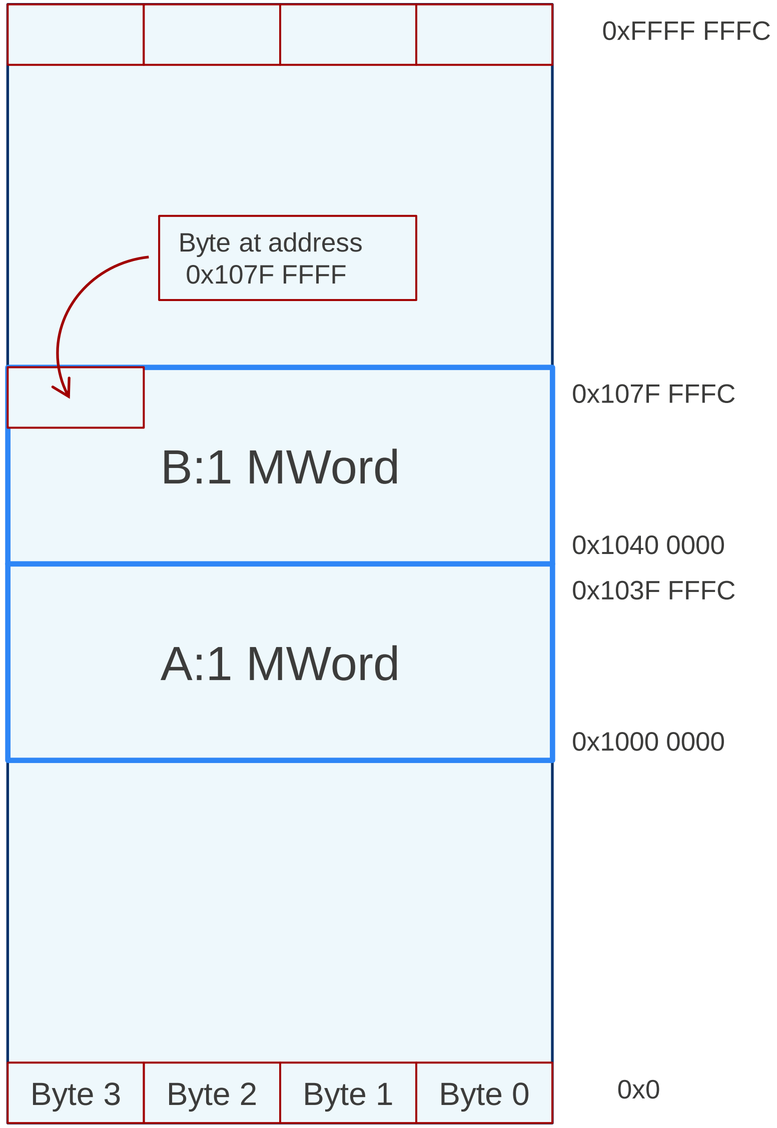
SAMPLE PROBLEM

Since the capacity of each chips is 1 Mbit and 8 Mbits are required, 8 chips must be used.

Since the processor data bus is 32-bit wide and the memory is 8-bit wide, 4 chips are put side-by-side to complete one 32-bit wide memory line.

The first set of 4 chips have a combined capacity of 8 MWords and are mapped from address 0x1000 0000 to 0x103F FFFF.

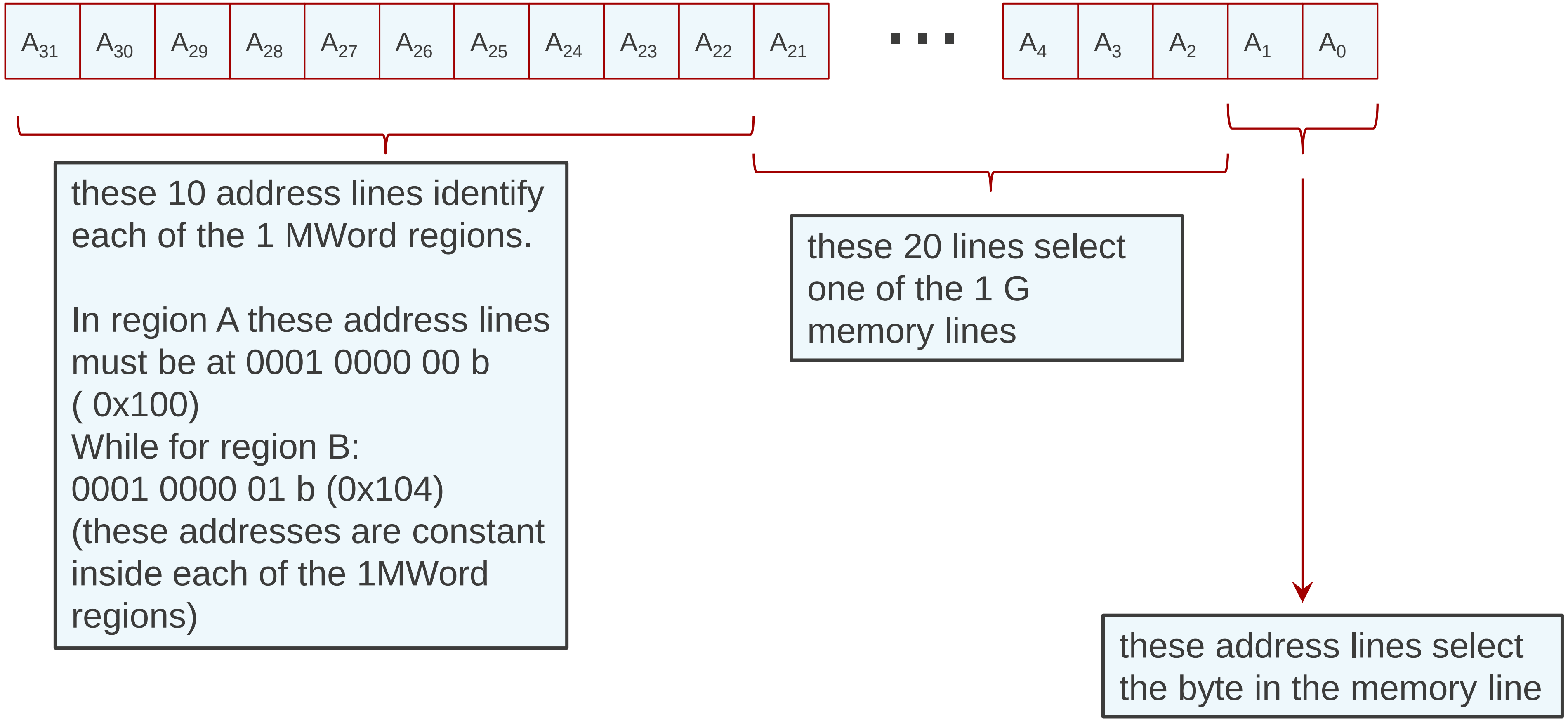
The second set of 4 chips are mapped from 0x1040 0000 to 0x107F FFFF.



source: Authors

SAMPLE PROBLEM

Selecting the address lines for CS (Chip Select) and for memory line addressing. These are the 32-bit addresses from the Cortex-M processor:



A_{31} A_{30} A_{29} A_{28} A_{27} A_{26} A_{25} A_{24} A_{23} A_{22} A_{21} . . . A_4 A_3 A_2 A_1 A_0

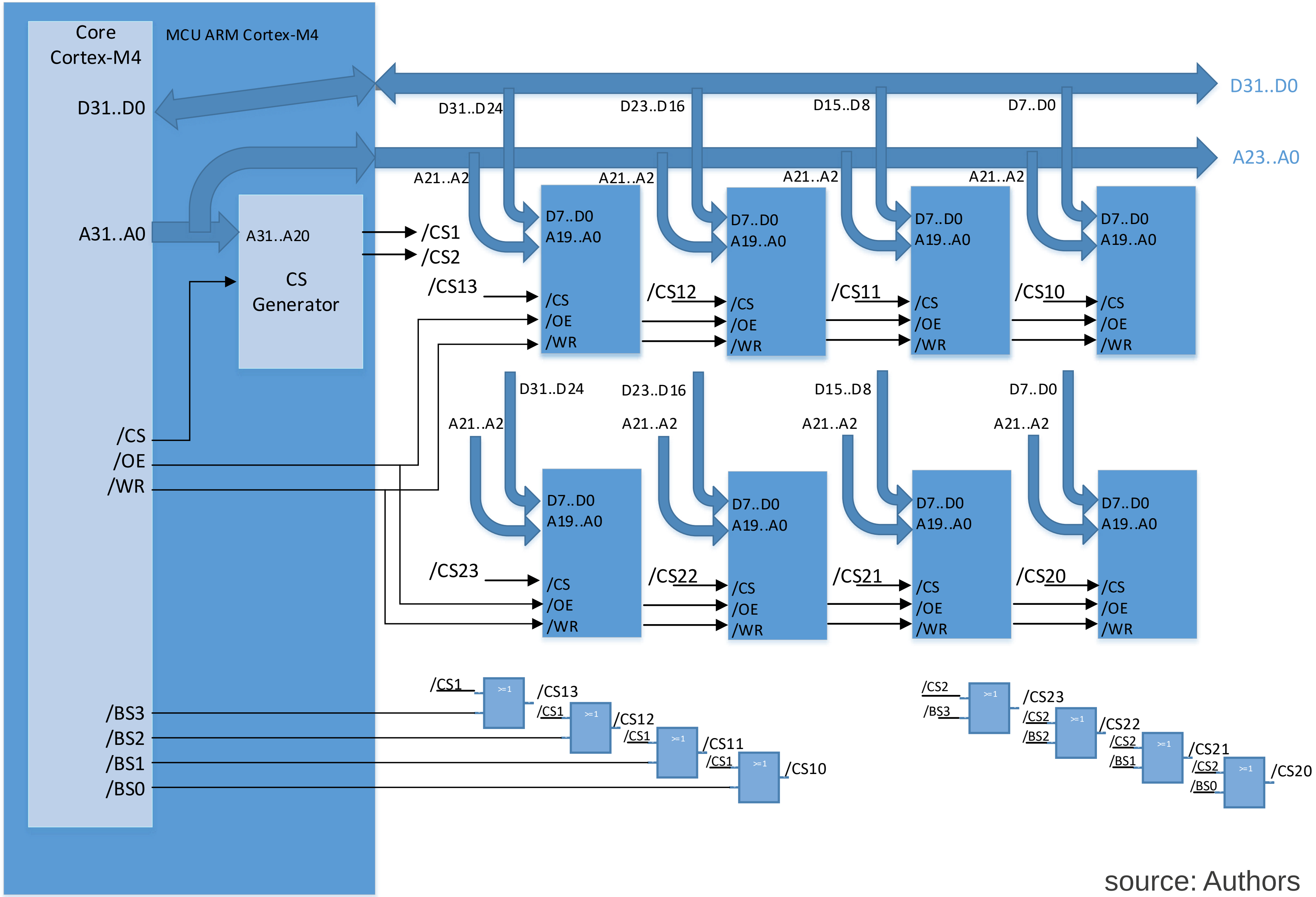
these 10 address lines identify each of the 1 MWord regions.

In region A these address lines must be at 0001 0000 00 b (0x100)
While for region B:
0001 0000 01 b (0x104)
(these addresses are constant inside each of the 1MWord regions)

these 20 lines select one of the 1 G memory lines

these address lines select the byte in the memory line

source: Authors



/CS1 must be active
in the range
0x1000 0000 to
0x103F FFFF

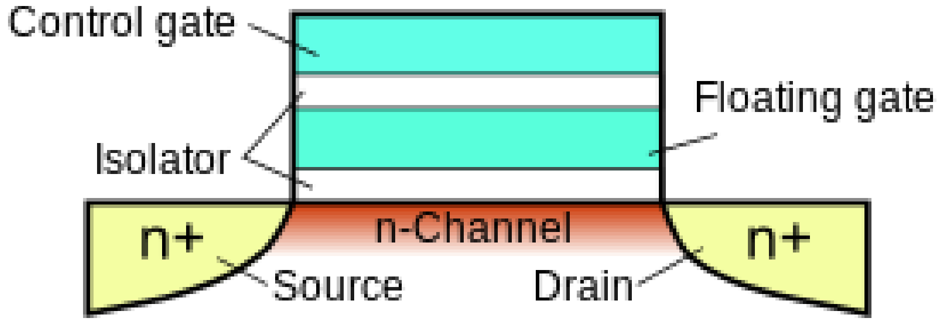
/CS2 must be active
in the range
0x1040 0000 to
0x107F FFFF

source: Authors

NOR FLASH MEMORY

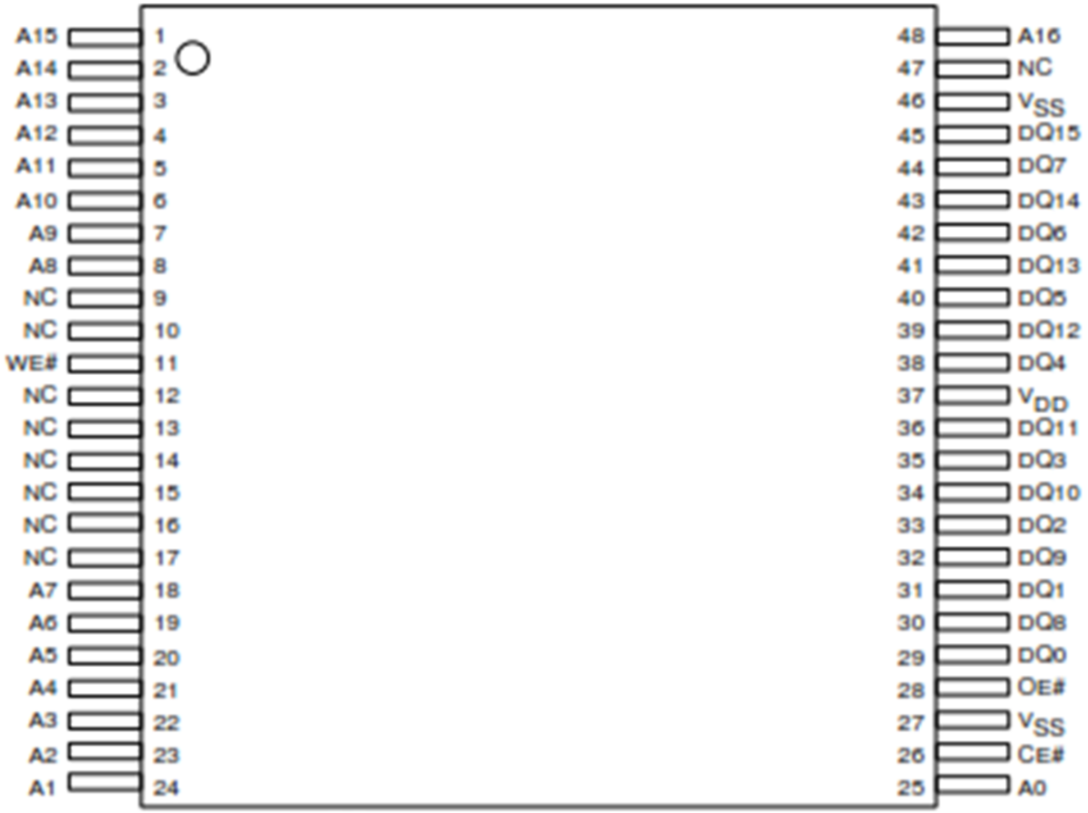
NOR Flash memory is frequently used in embedded systems to store non-volatile data, particularly code and constants.

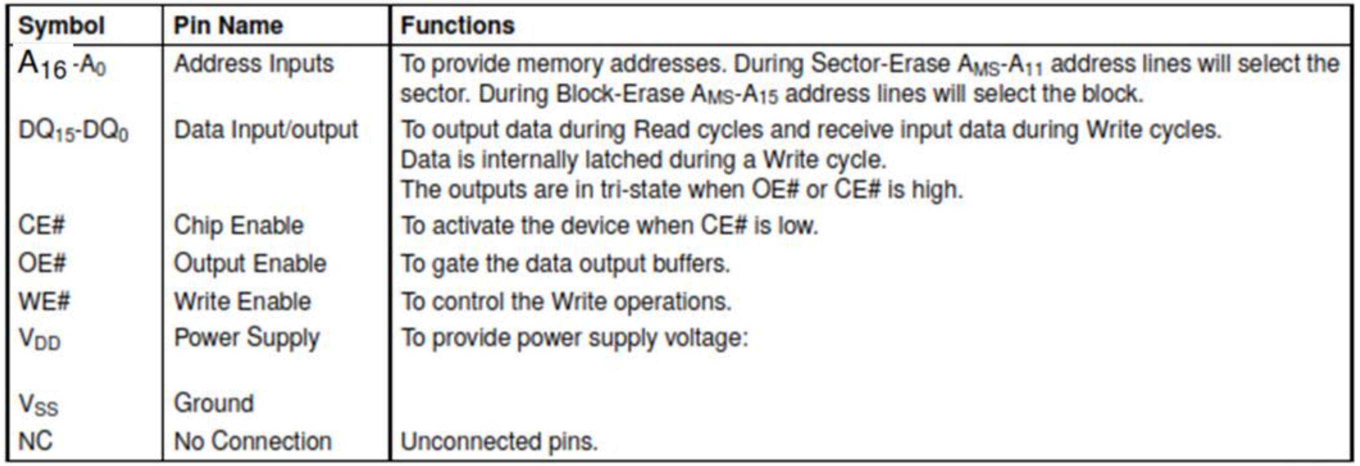
Each bit of a NOR Flash memory is implemented by a single floating gate MOS. The floating gate may be charged and, since the floating gate is isolated, the charges are trapped into de gate. Hence, the two possible states are: charges trapped in the floating gate vs no charges trapped in the floating gate.



source: wikimedia.org (CC)

EXAMPLE OF A FLASH MEMORY DEVICE 128K X 16 BITS

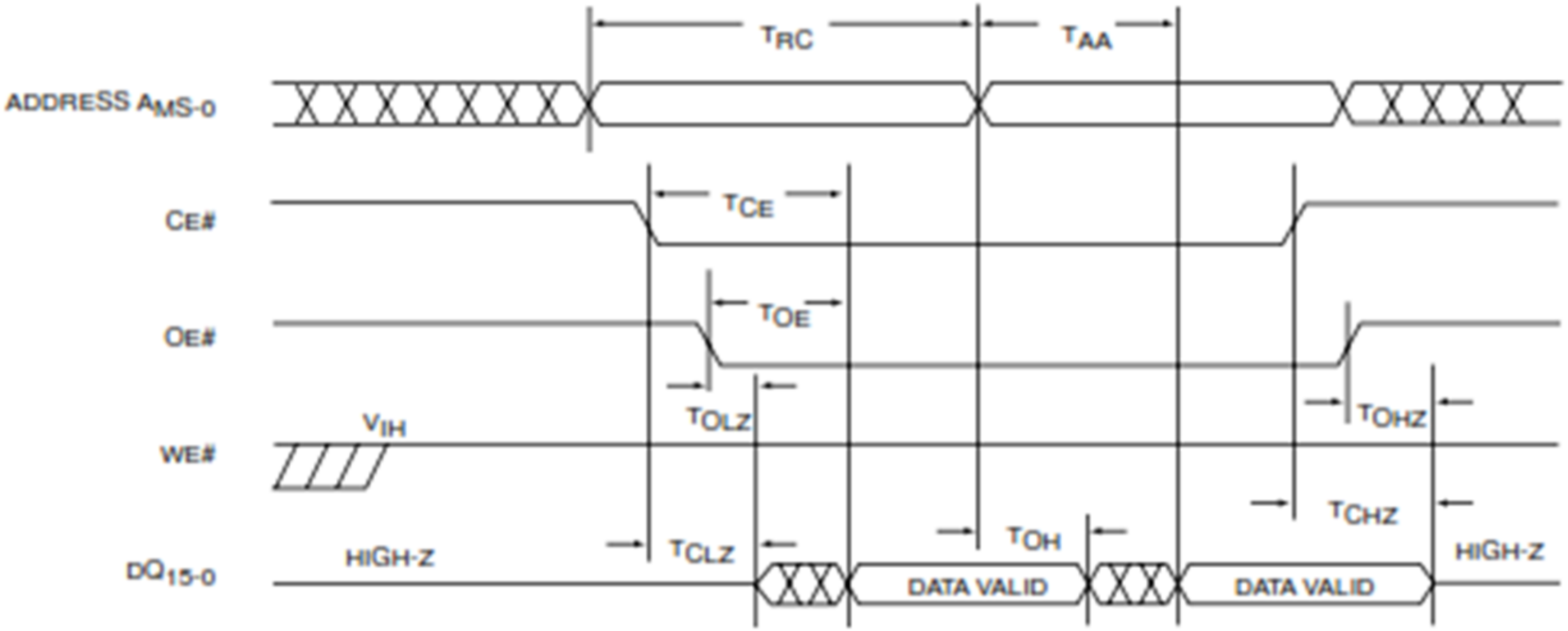




Symbol	Pin Name	Functions
A_{16}-A_0	Address Inputs	To provide memory addresses. During Sector-Erase A_{MS}-A_{11} address lines will select the sector. During Block-Erase A_{MS}-A_{15} address lines will select the block.
DQ_{15}-DQ_0	Data Input/output	To output data during Read cycles and receive input data during Write cycles. Data is internally latched during a Write cycle. The outputs are in tri-state when OE# or CE# is high.
CE#	Chip Enable	To activate the device when CE# is low.
OE#	Output Enable	To gate the data output buffers.
WE#	Write Enable	To control the Write operations.
V_{DD}	Power Supply	To provide power supply voltage:
V_{SS}	Ground	
NC	No Connection	Unconnected pins.

EXAMPLE OF A FLASH MEMORY DEVICE 128K X 16 BITS

The operation of a NOR Flash memory is straightforward: the processor sets the address to be read, and activates Chip Select (CE#) and Output Enable (OE#) after a delay (TCE of TOE) valid data is presented on the data bus.

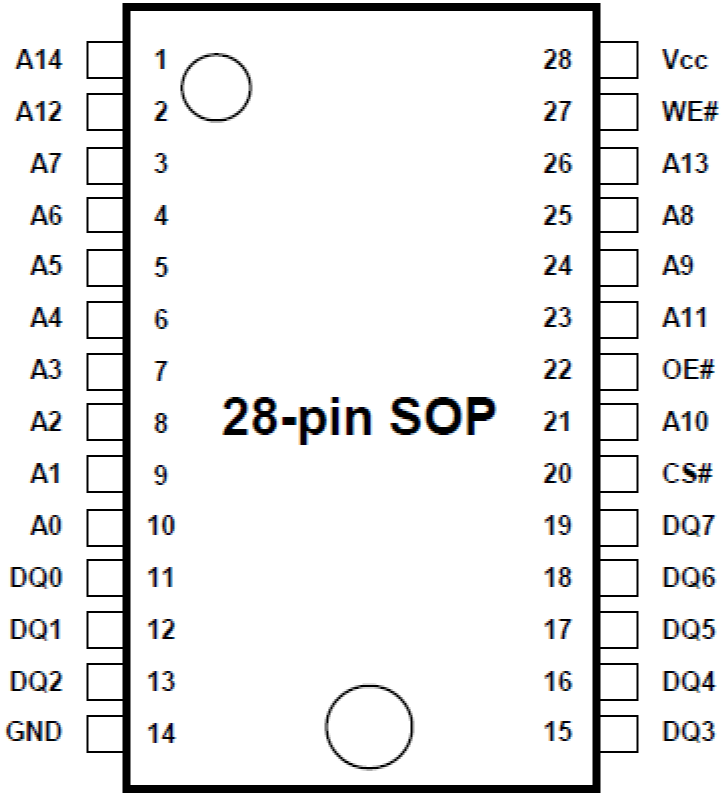


SRAM EXAMPLE

R1LV5256E Series

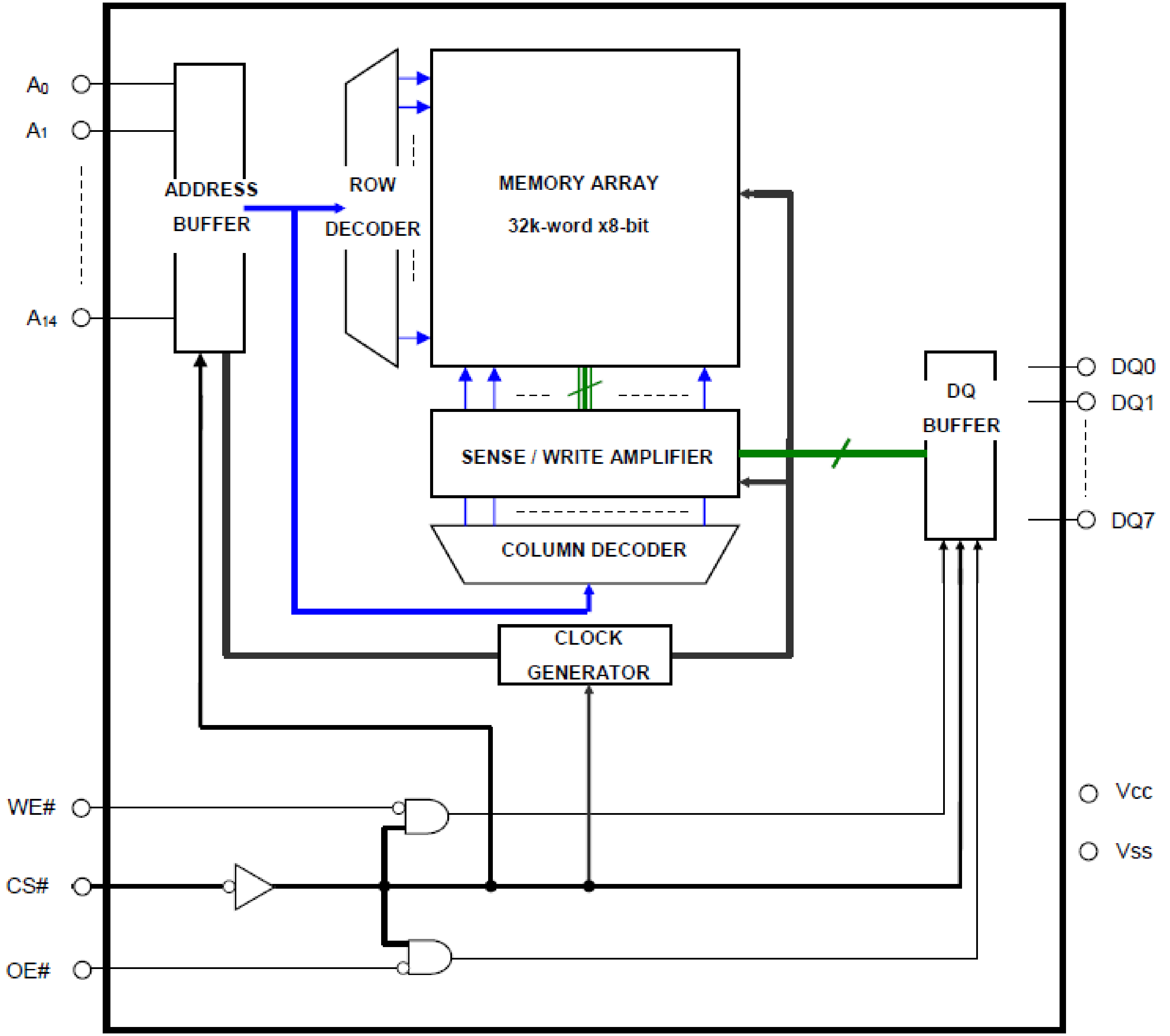
256Kb Advanced LPSRAM (32k word x 8bit)

Pin name	
Vcc	Power supply
Vss (GND)	Ground
A0 to A14	Address input
DQ0 to DQ7	Data input/output
CS#	Chip select
WE#	Write enable
OE#	Output enable



source: Renesas

INTERNAL ORGANIZATION



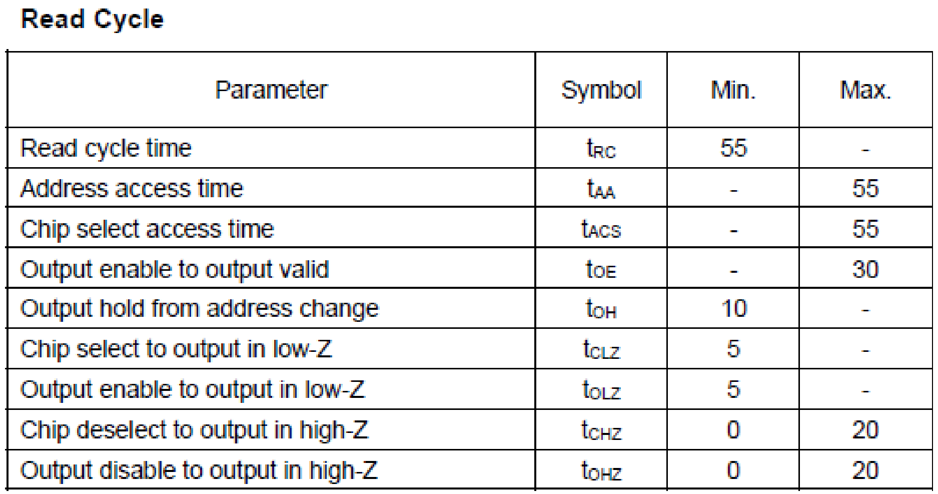
Operation Table

CS#	WE#	OE#	DQ0~7	Operation
H	X	X	High-Z	Stand-by
L	L	X	Din	Write
L	H	L	Dout	Read
L	H	H	High-Z	Output disable

Note 1. H: V_{IH} L:V_{IL} X: V_{IH} or V_{IL}

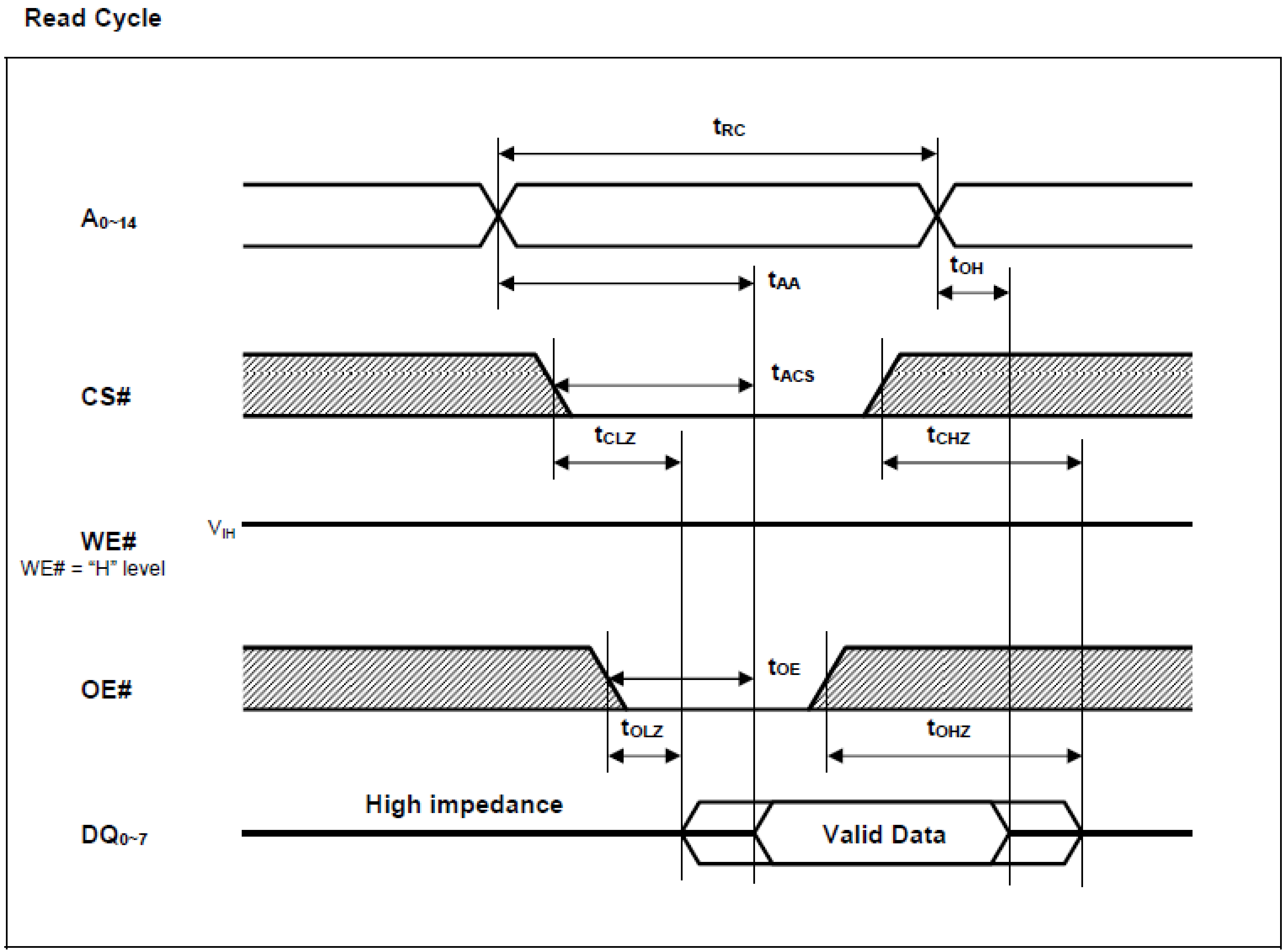
READ CYCLE

Read Cycle timing diagram



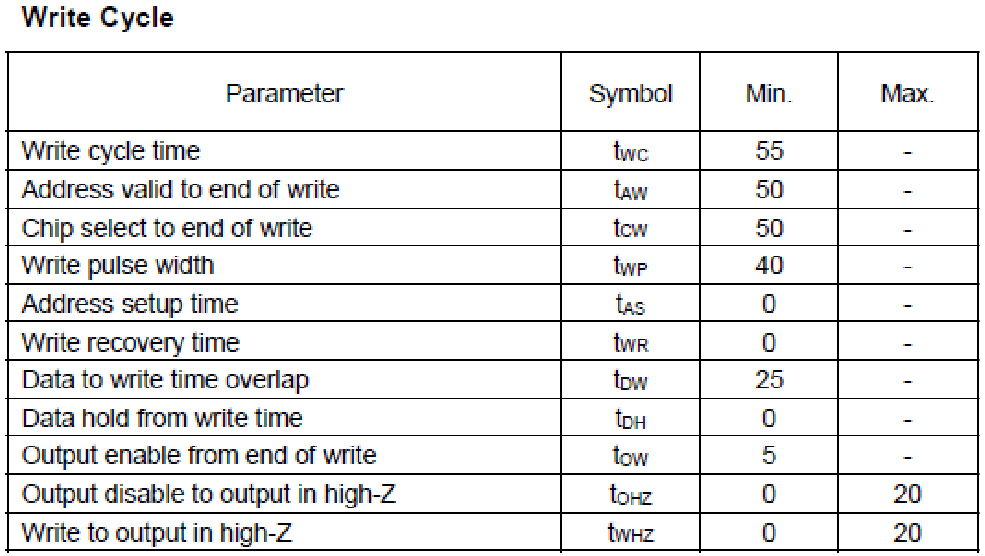
Read Cycle

Parameter	Symbol	Min.	Max.
Read cycle time	t_{RC}	55	-
Address access time	t_{AA}	-	55
Chip select access time	t_{ACS}	-	55
Output enable to output valid	t_{OE}	-	30
Output hold from address change	t_{OH}	10	-
Chip select to output in low-Z	t_{CLZ}	5	-
Output enable to output in low-Z	t_{OLZ}	5	-
Chip deselect to output in high-Z	t_{CHZ}	0	20
Output disable to output in high-Z	t_{OHZ}	0	20



WRITE CYCLE

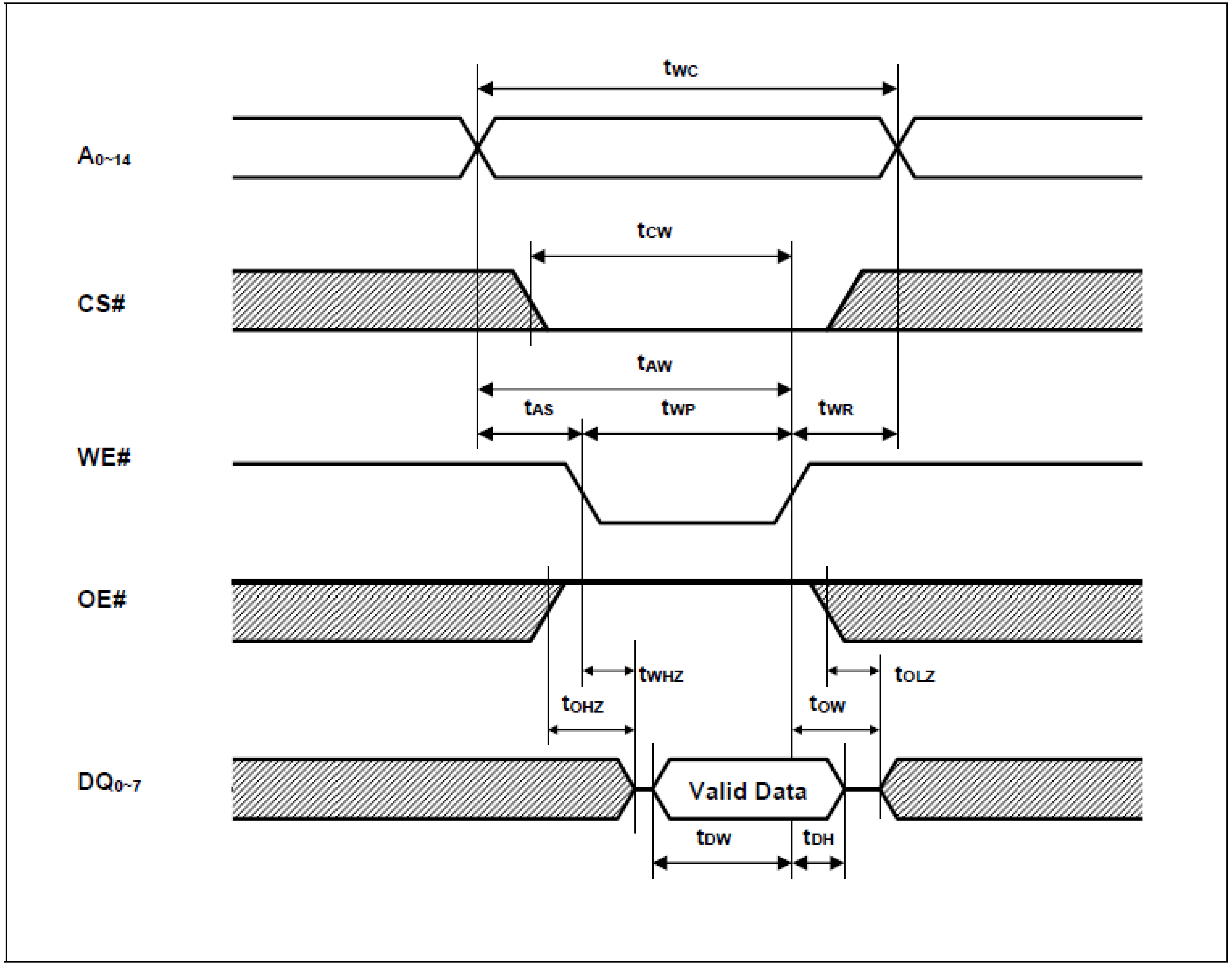
Write Cycle timing diagram



Write Cycle

Parameter	Symbol	Min.	Max.
Write cycle time	t_{WC}	55	-
Address valid to end of write	t_{AW}	50	-
Chip select to end of write	t_{CW}	50	-
Write pulse width	t_{WP}	40	-
Address setup time	t_{AS}	0	-
Write recovery time	t_{WR}	0	-
Data to write time overlap	t_{DW}	25	-
Data hold from write time	t_{DH}	0	-
Output enable from end of write	t_{OW}	5	-
Output disable to output in high-Z	t_{OHZ}	0	20
Write to output in high-Z	t_{WHZ}	0	20

Write Cycle (1) (WE# CLOCK)

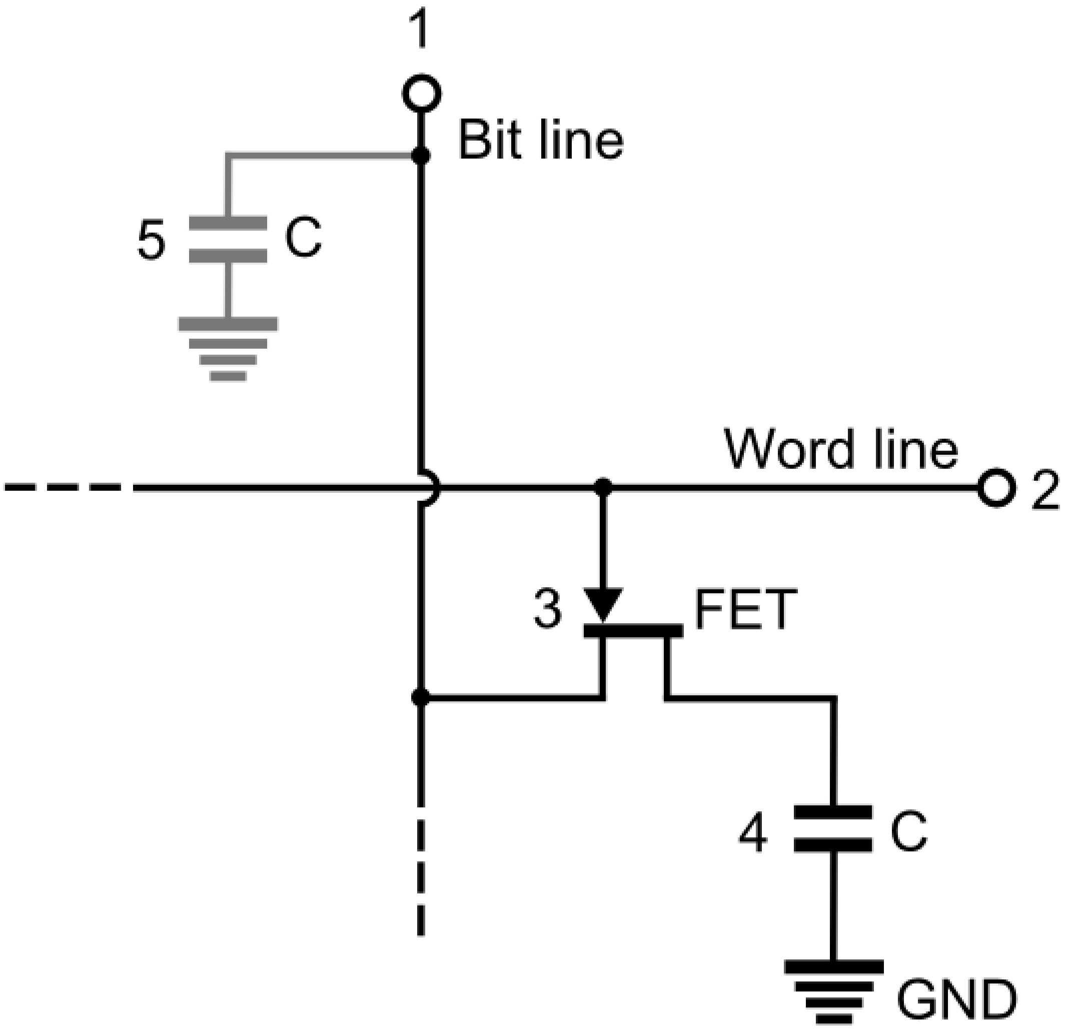


DYNAMIC RAM

In a dynamic RAM, each bit is implemented by a circuit consisting of a capacitor and a transistor.
The capacitor stores de information (charged vs discharged) and the transistor acts as a switch that connects the capacitor to the Bit Line when that particular bit is accessed.

Because the capacitance is very small and because of leakage, the charge of the capacitor only holds reliably for a few milliseconds, hence, this memory bit must be constantly refreshed.

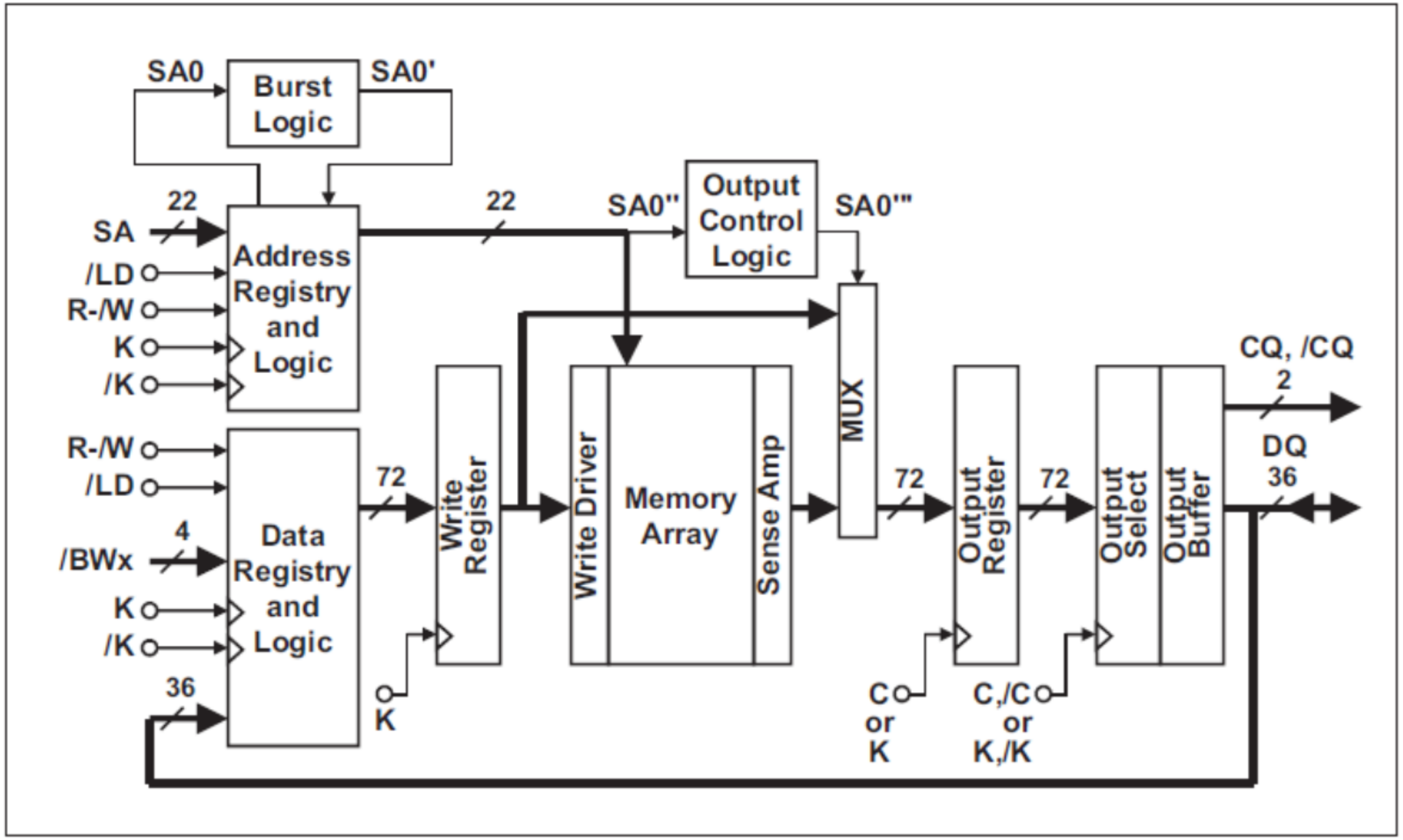
Also, every read of a memory cell is destructive, hence, after a read the value must be rewritten.



source: wikimedia.org (CC)

DDR – DOUBLE DATA RATE

[R1Q4A4436RBG]



5 – TIMER AND GPIO

1. Introduction - Peripherals
2. Timer
3. PWM – Pulse Width Modulations
4. GPIO
5. Low-power Drivers (LED, Relay)
6. Power Drivers (DC Motor)

5.1 – PERIPHERALS

A Microprocessor-based system consists of three types of components:

1. **Microprocessor** (the **core** of the MCU) - executes the instructions of a program;
2. **Memory** - store code and data;
3. **Peripherals** - perform specific functions particularly related to I/O. Peripherals provide the means for the system to sense, actuate and communicate with the world.

A microprocessor-based system without peripherals would be unable to interact with the rest of the world!

PERIPHERALS

Classes of peripherals in a microprocessor-based system:

- Digital I/O: input/output of digital signals,
- Analog I/O: input/output of analog signals,
- Timing: pulse generation, pulse measurement, PWM, ...
- Storage: non-volatile memory, file system, ...
- Communications: RS-232, SPI, I2C, CAN, USB, Ethernet, ...
- HMI: touch-screen, keyboard, ...
- Imaging: camera interface, ...
- System management: clock generation, watchdog, power management, ...

5.2 – TIMERS/COUNTERS

Timers/Counters are essential components of a Microcontroller Unit (MCU). They consists of a **digital counter circuit** that counts pulses on their input. **Timers** count clock pulses and Counters count pulses of a signal present on their input pin. **Timers/Counters** are used for:

- Measuring time, particularly time intervals,
- Counting events,
- Keeping track of current date and time (Real-Time Clock - RTC),
- Generating digital waveforms (PWM - Pulse-Width Modulation),
- Generating periodic interrupts to the MCU (Operating System Clock),
- Generating periodic signals to other peripheral, such as the command to start an Analog-to-Digital Conversion,
- Restart the MCU if software is unable to periodically restart a WATCHDOG timer.

TIMERS

- Timers/Counters vary in their characteristics, such as:
  - edge of the input signal used for counting: positive, negative, or both edges;
  - count direction: up, down, or both;
  - what is being counted: clock pulses (timers) or other input signal (counters);
  - periodic timers vs single-shot;
  - a number of actions can be taken at the end of the timing period: change state of GPIO pin, generate IRQ, stop the timer, reload the timer;
  - number of bits of the counter circuit: typically 32 bits for ARM MCUs;
  - since the frequency of the input signal and number of bits of the counter determine the maximum timing period, a prescaler may be used to reduce the input frequency.

TIMING SAMPLE PROBLEM 1

Consider the problem where an LED, connected to a pin on an MCU, must be on during 0.9 second.

The first solution to be presented is a software-based solution:

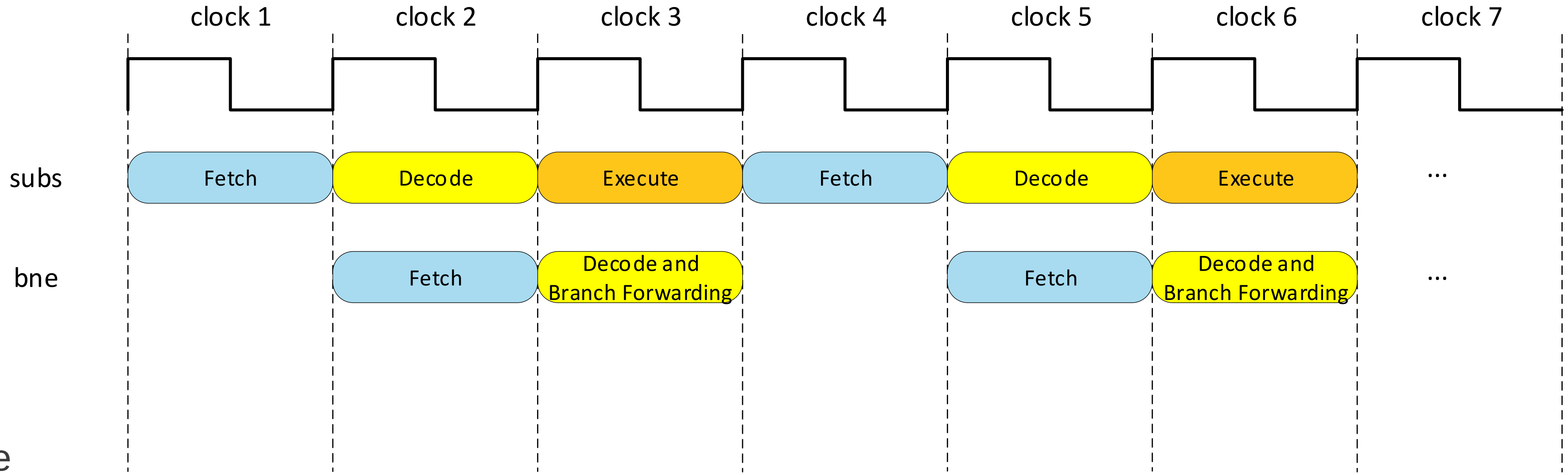
1. Turn the LED on by activating the MCU pin connected to it;
2. Execute a loop N times, where N is chosen so that the execution of the loop takes 0.9 second;
3. Turn the LED off by deactivating the MCU pin connected to the LED

SOFTWARE-BASED SOLUTION

```
void hal_entry(void)
{
    // pins 0,1,2 are output, reset pin 0 to 0 to turn green led ON
    R_IOPORT6->PCNTR1 = 0x00060007;

    __asm("     ldr r4,=72000000-2");
    __asm("loop:                ");
    __asm("     subs r4,r4,#1   ");
    __asm("     bne loop        ");

    // pins 0,1,2 are output, set pin 0 to 1 to turn green led OFF
    R_IOPORT6->PCNTR1 = 0x00070007;
}
```

SOFTWARE-BASED SOLUTION

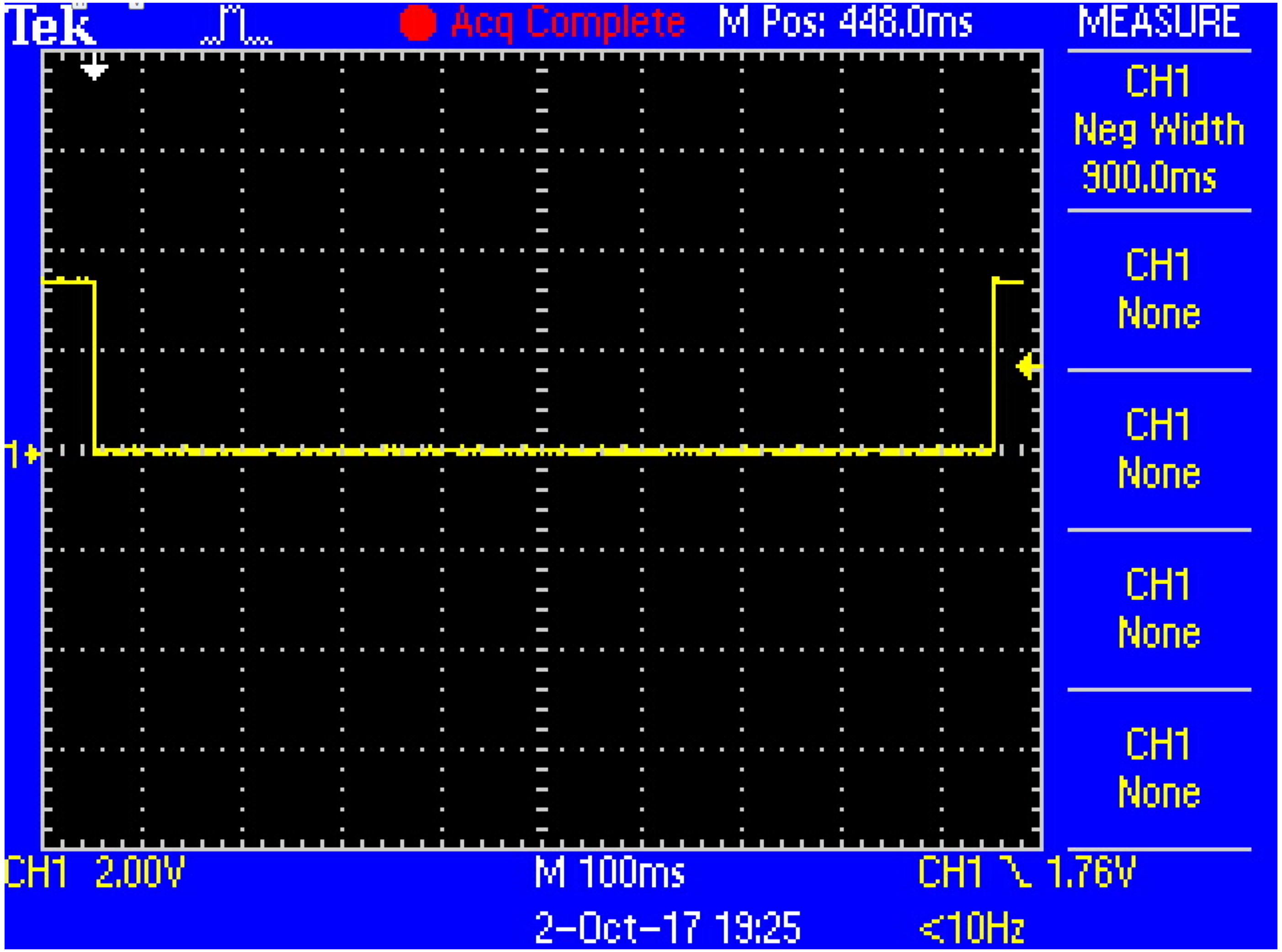
- This is the Timing Diagram for the execution of the code of the previous slide.
- Due to branch forwarding at the decode stage of the BNE instruction, each loop takes 3 clock cycles.
- At 240 MHz (period of 4.16ns), it takes 12.5 ns for each loop; hence, 0.9 second takes 72,000,000 loops.
- Remark: the loop_count must be reduced by 2 to compensate for GPIO and load instructions.



source: Authors

SOFTWARE-BASED SOLUTION

- Connecting a scope to the LED allows us to verify the timing of the software-based solution.



source: Authors

SOFTWARE-BASED SOLUTION

Evaluation of the software-based solution:

- 100% of the MCU processing capability was used for counting so that the exact timing of 0.9 seconds was obtained.
- No other activities where executed by the processor during this time.
- If interrupts where allowed, there would have been an error in timing.
  - This software-based solution is called busy-wait. It should be avoided as it wastes processor cycles and energy.

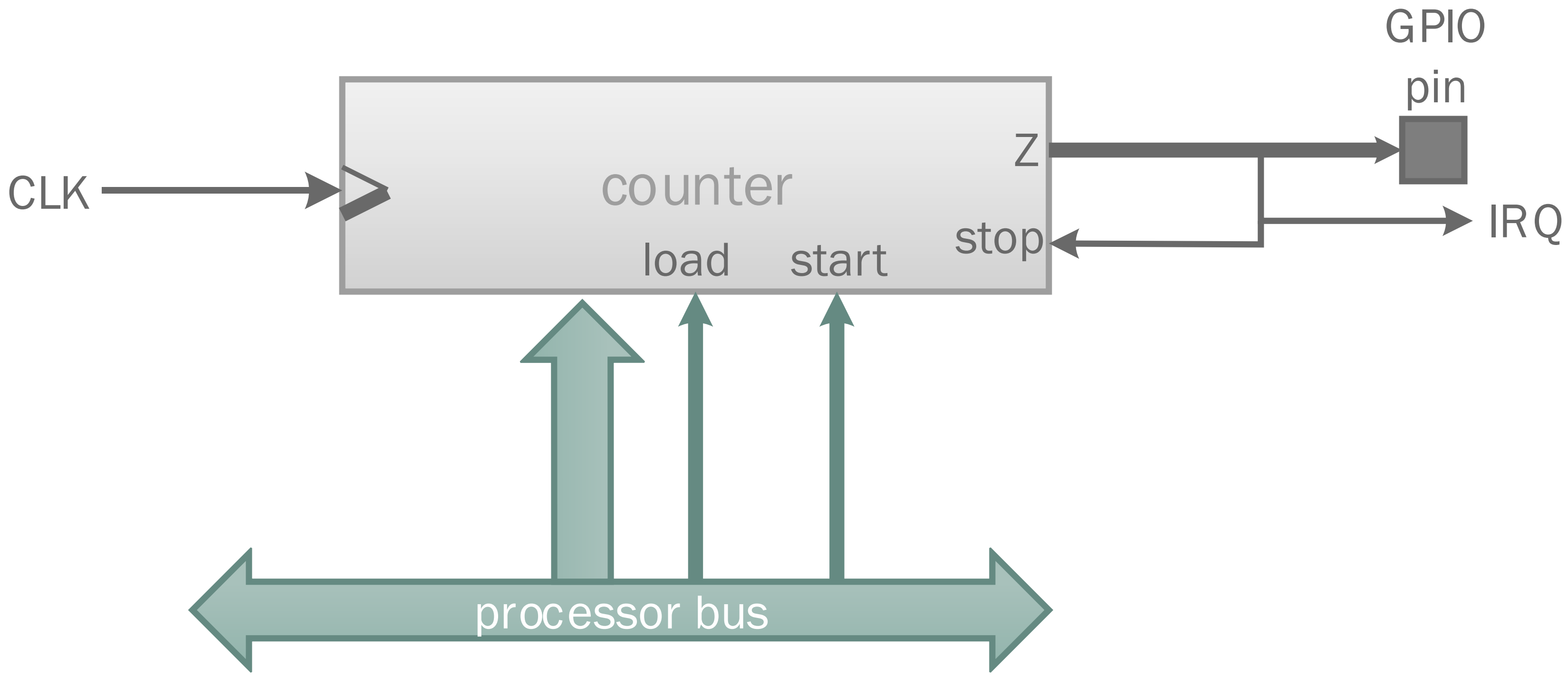
A VERY SIMPLE TIMER

- To avoid the drawbacks of the software-based solution, a small piece of hardware is added to an MCU: a counter.
- This circuit performs the function of counting clock cycles, freeing the processor to perform other tasks.

A VERY SIMPLE TIMER

Description:

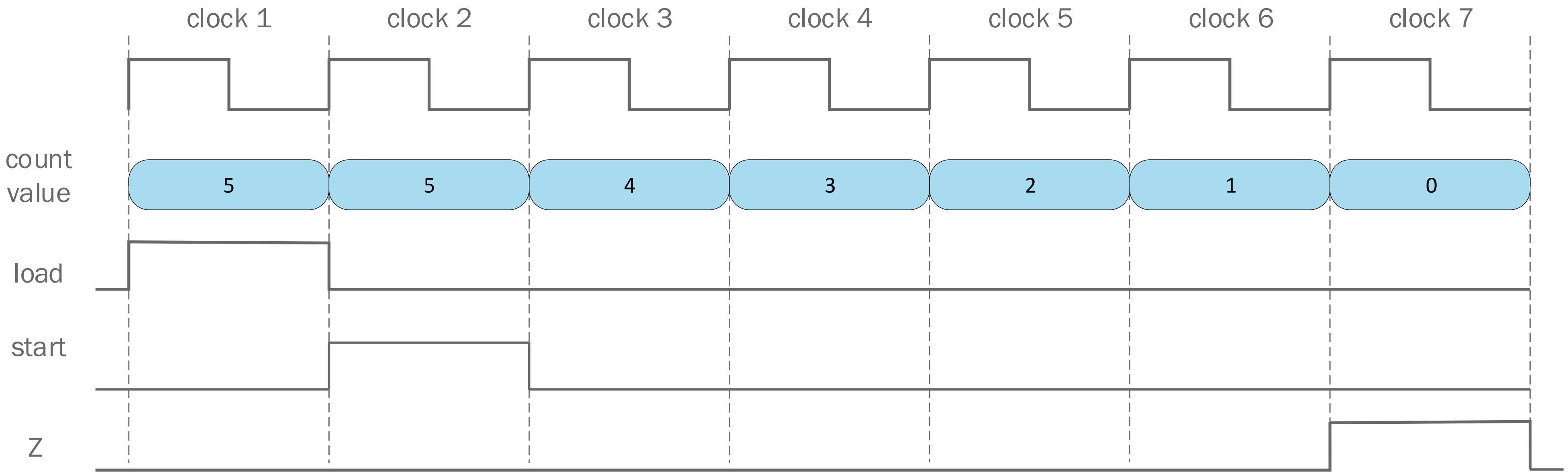
- The counter decrements its value on every positive edge of the CLK.
- The processor can write an initial count value by performing a load followed by a start command.
- When the count reaches 0 the Z output is activated.
- The Z output may command a change to the GPIO pin, generate an Interrupt Request (IRQ), and stop the counter.



source: Authors

A VERY SIMPLE TIMER

- clk 1 - the value 5 is loaded to the counter.
- clk 2 - the processor commands the start of the counting.
- clk 3 to 7 - count value is decremented on every positive edge of the CLK.
- clk 7 - count reaches 0 activating the Z output.



source: Authors

SOLVING THE TIMING SAMPLE PROBLEM 1 USING THE SIMPLE TIMER

Solution:

- Load the value 72,000,000 to the timer and start it.
- When the count reaches zero:
  - Option 1: timer commands LED pin to turn LED off;
  - Option 2: generate an IRQ and its service routine turns the LED off.
- Remark: for option 2, the value loaded to the timer should be lower to compensate for the entry into the interrupt service routine.

SOLVING THE TIMING SAMPLE PROBLEM 1 USING THE SIMPLE TIMER

Evaluation:

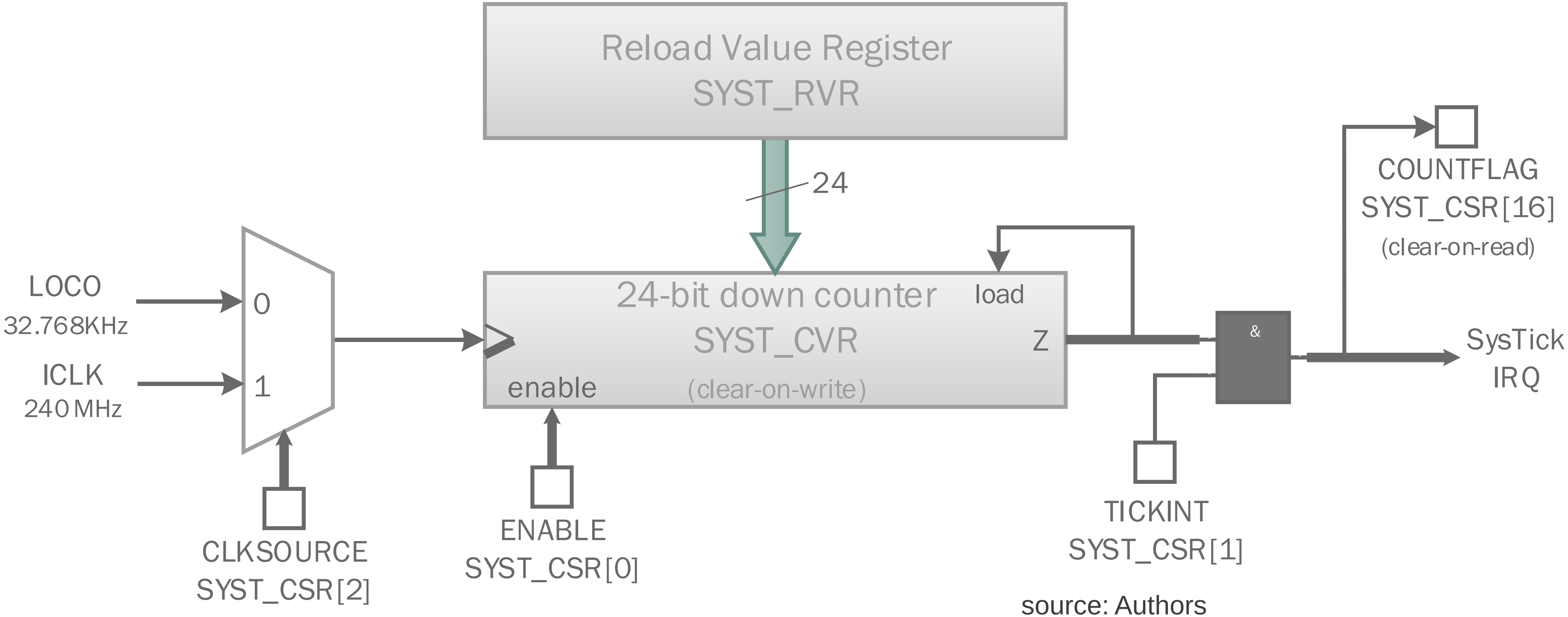
- The solution using a timer requires significantly less processing power, just a few cycles to configure the timer and to perform an action when the timing period finishes.
- Processor is free to perform other tasks OR
  processor can be put into a low energy sleep state.
- If interrupts are serviced during the timing period, this will not affect the timing precision.

TIMER CASE STUDY 1 – SYSTICK

- The SYSTICK is a timer available in all Cortex-M processors.
- It is a simple 24-bit counter with auto-reload.
- Its main use is to generate periodic interrupts required by most Embedded Operating Systems.
- Since its structure and operation is defined by ARM, its interface is standard, regardless of MCU supplier.

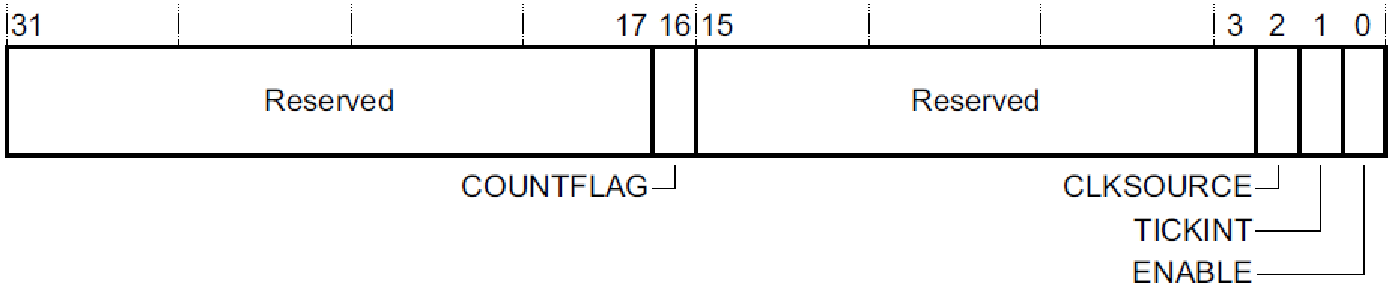
TIMER CASE STUDY 1 – SYSTICK

SysTick auto-reload has a period of SYST_RVR + 1 clock cycles.

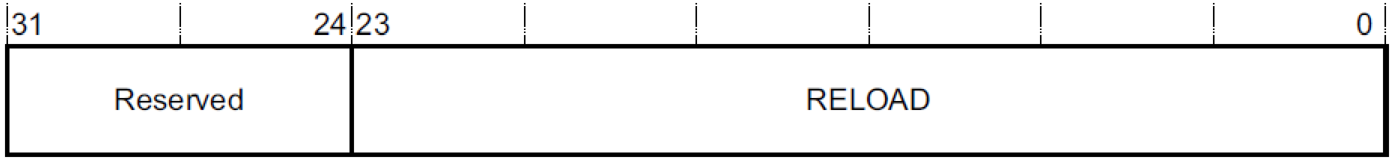


source: Authors
(based on ARM documentation:
DDI0403D ARMv7-M Architecture Reference Manual)

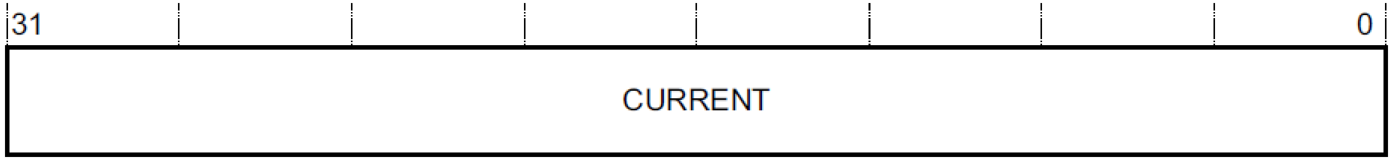
TIMER CASE STUDY 1 – SYSTICK



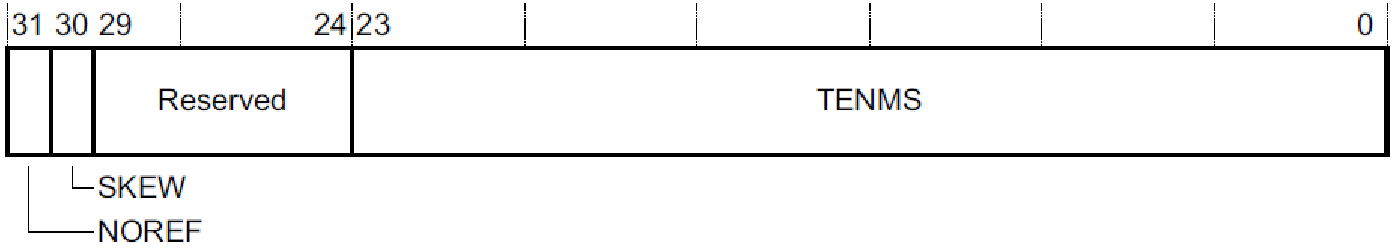
SYST_CSR controls de operation of SysTick. When this register is read, COUNTFLAG is cleared.



SYST_RVR holds de 24-bit reload value. Writing a 0 to this register disables SysTick.



SYST_CVR is the current count value. By writing any value to it the register is cleared.



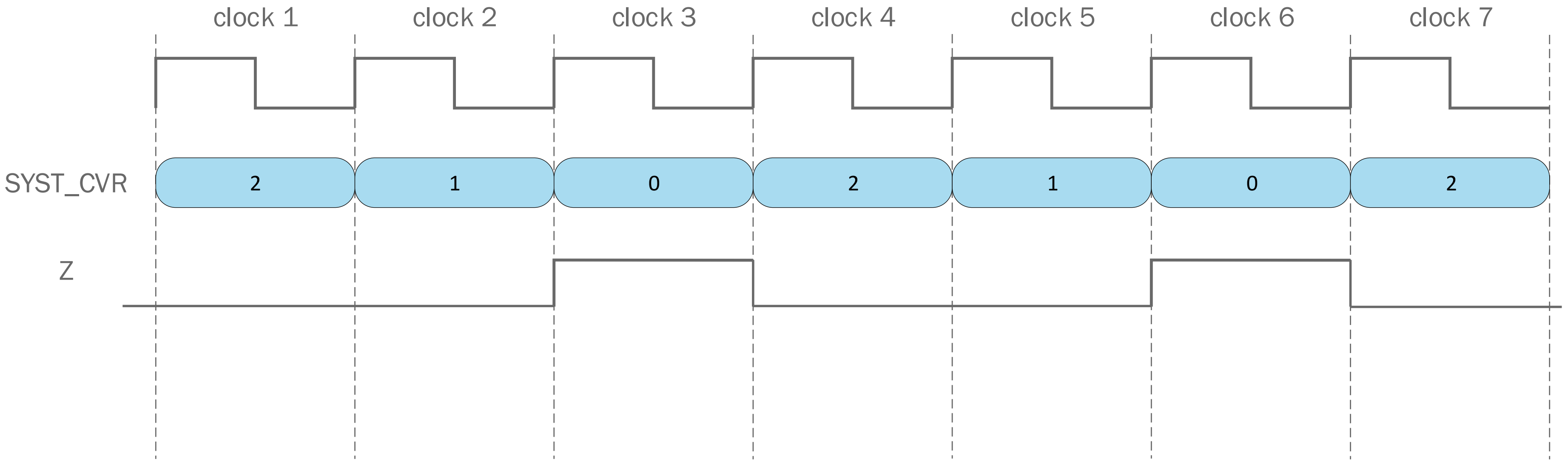
SYST_CALIB values are factory defined. TENMS, if available, holds the reload value corresponding to 10 ms for the reference clock.

source: DDI0403D ARMv7-M Architecture Reference Manual

TIMER CASE STUDY 1 – SYSTICK

- Timing diagram for the scenario where SYST_RVR holds the value 2.
- As the reload occurs on the next positive edge of the clock, the SysTick period is of 3 clock cycles.

  If enabled, every reload generates a SysTick interrupt request (IRQ).



source: Authors

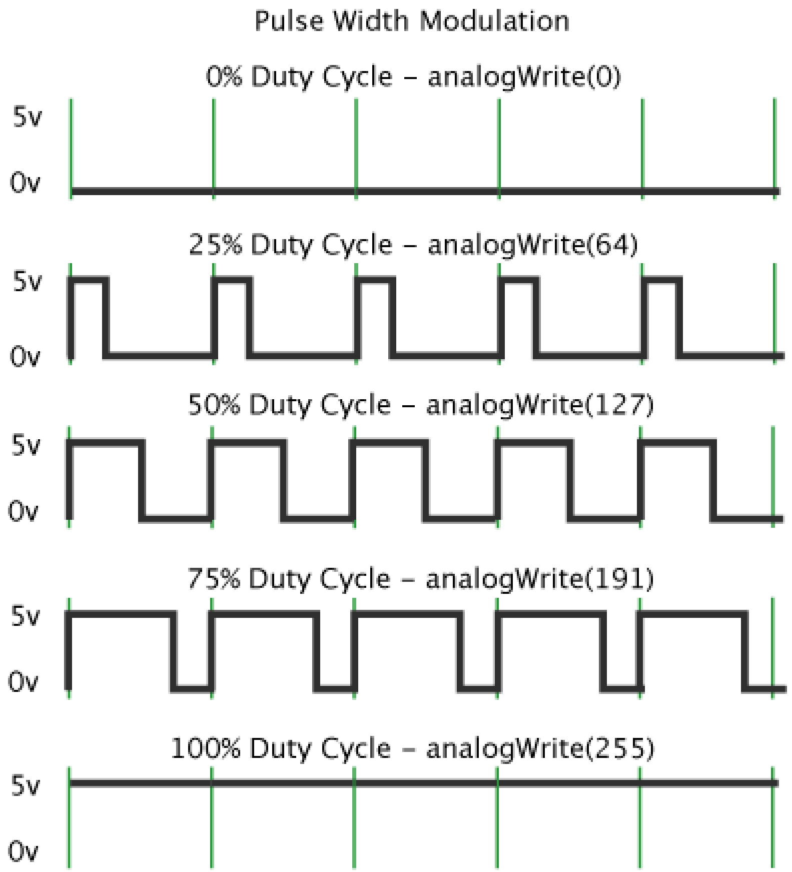
5.3 – PWM

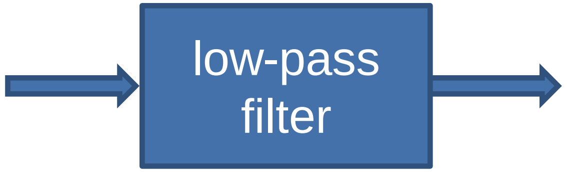
- **Pulse Width Modulation** (PWM) is a modulation technique that allows the encoding of analog information in a binary digital signal.

  This is achieved by encoding the information in width of a pulse while maintaining the pulse frequency constant.
- A low pass filter with cut-off frequency way below the PWM frequency restores the analog information.
- PWM has several applications including: control of power electronics including switched power supplies, motor control, temperature control and light dimming; audio power-amplifiers; and analog signal generation.
- Often, a low-pass filter is not required as the load itself performs this function.

PWM

The effect of 5 different duty cycles after passing a low-pass filter:




low-pass filter

0 V

1.25 V

2.5 V

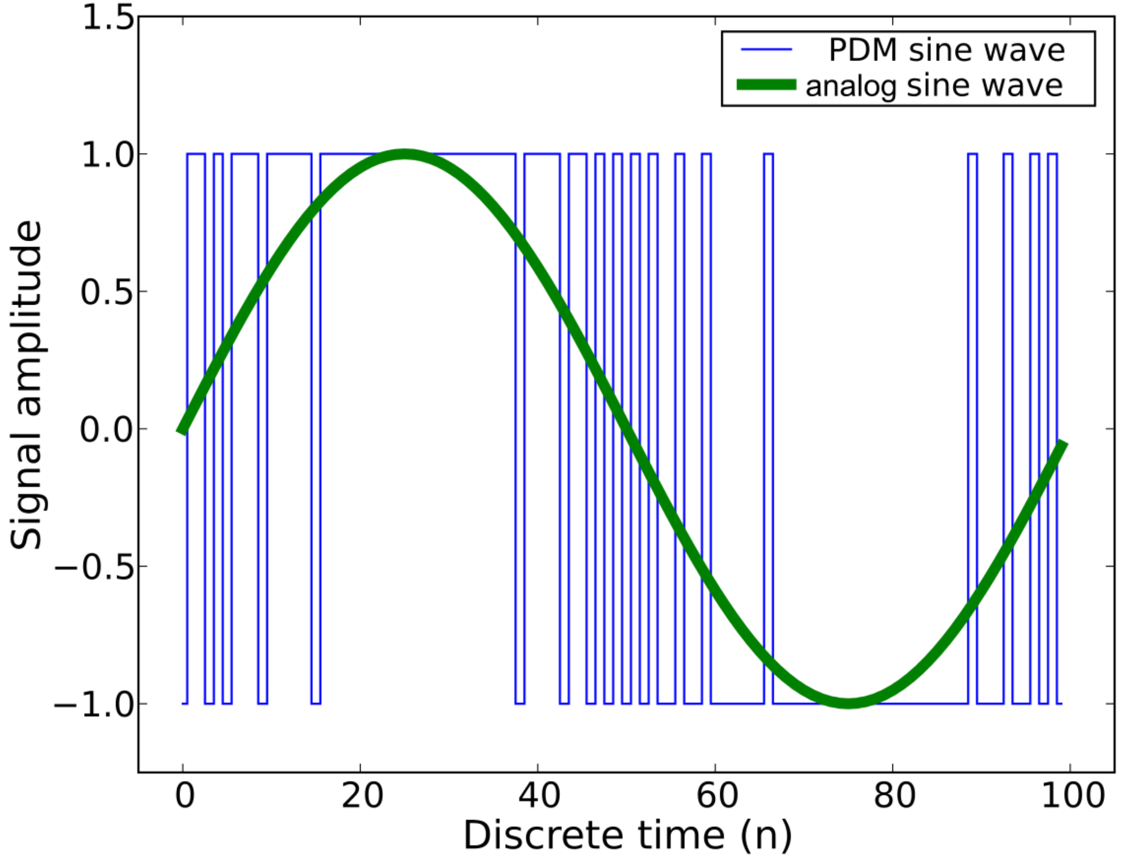
3.75 V

5V

source: commons.wikimedia.org (CC)

PWM

- If the duty cycle of a PWM signal is varied on every PWM period, an analog signal can be produced (after a low-pass filter).
- The higher the frequency of the PWM signal when compared to the frequency of the analog signal, the better (less noisy) the analog signal will be.



source: commons.wikimedia.org (CC)

A PWM TIMER

Operation:

- at start of period set the GPIO pin.
- counts up from 0.
- when counter matches Match Reg

  then reset GPIO pin.
- when counter matches Limit Reg

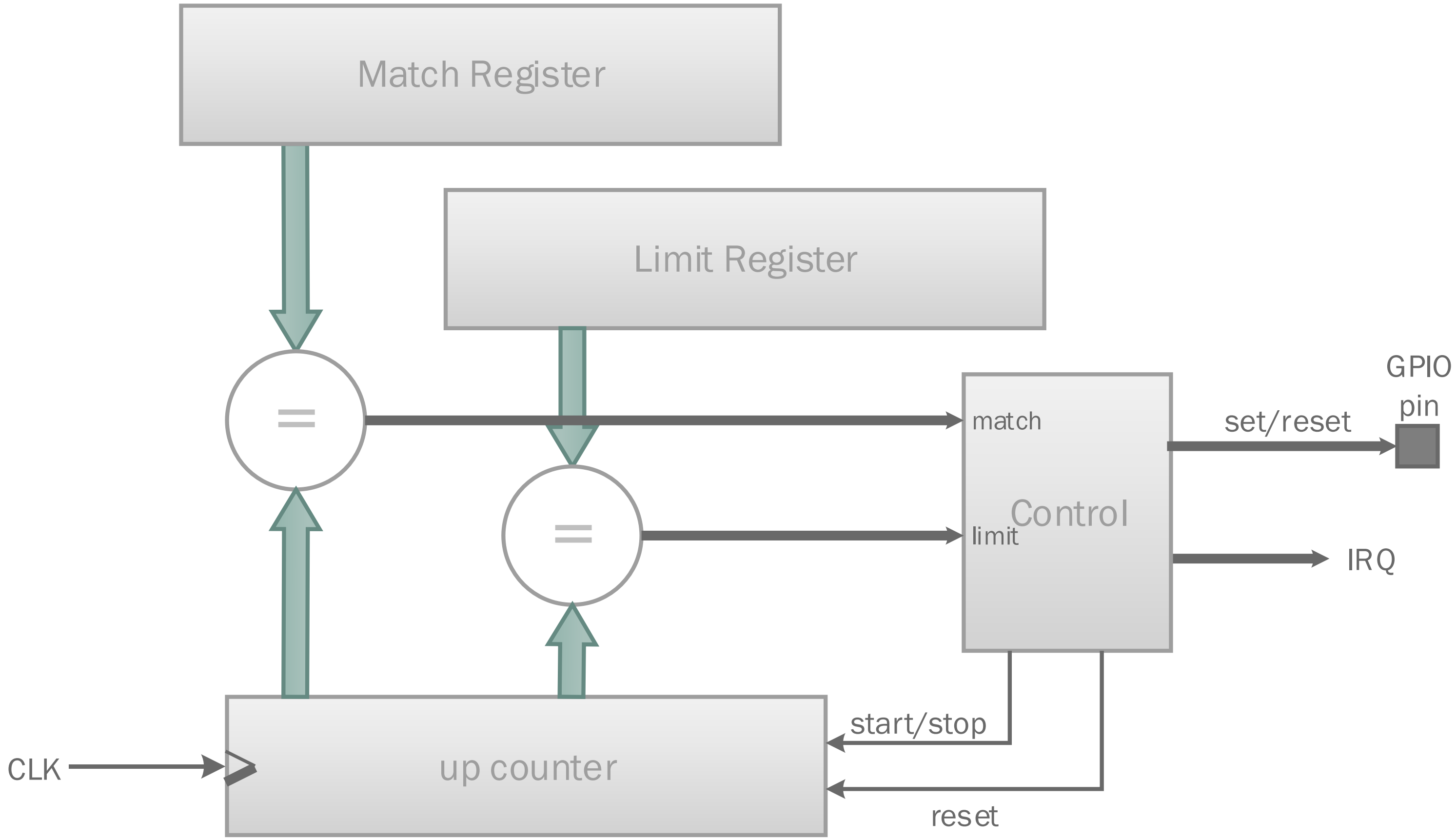
  then:

  set GPIO pin;

  generate IRQ;

  reset counter.
- interrupt service routine may

  reprogram the match register.



source: Authors

TIMER CASE STUDY 2 – S7G2 GPT

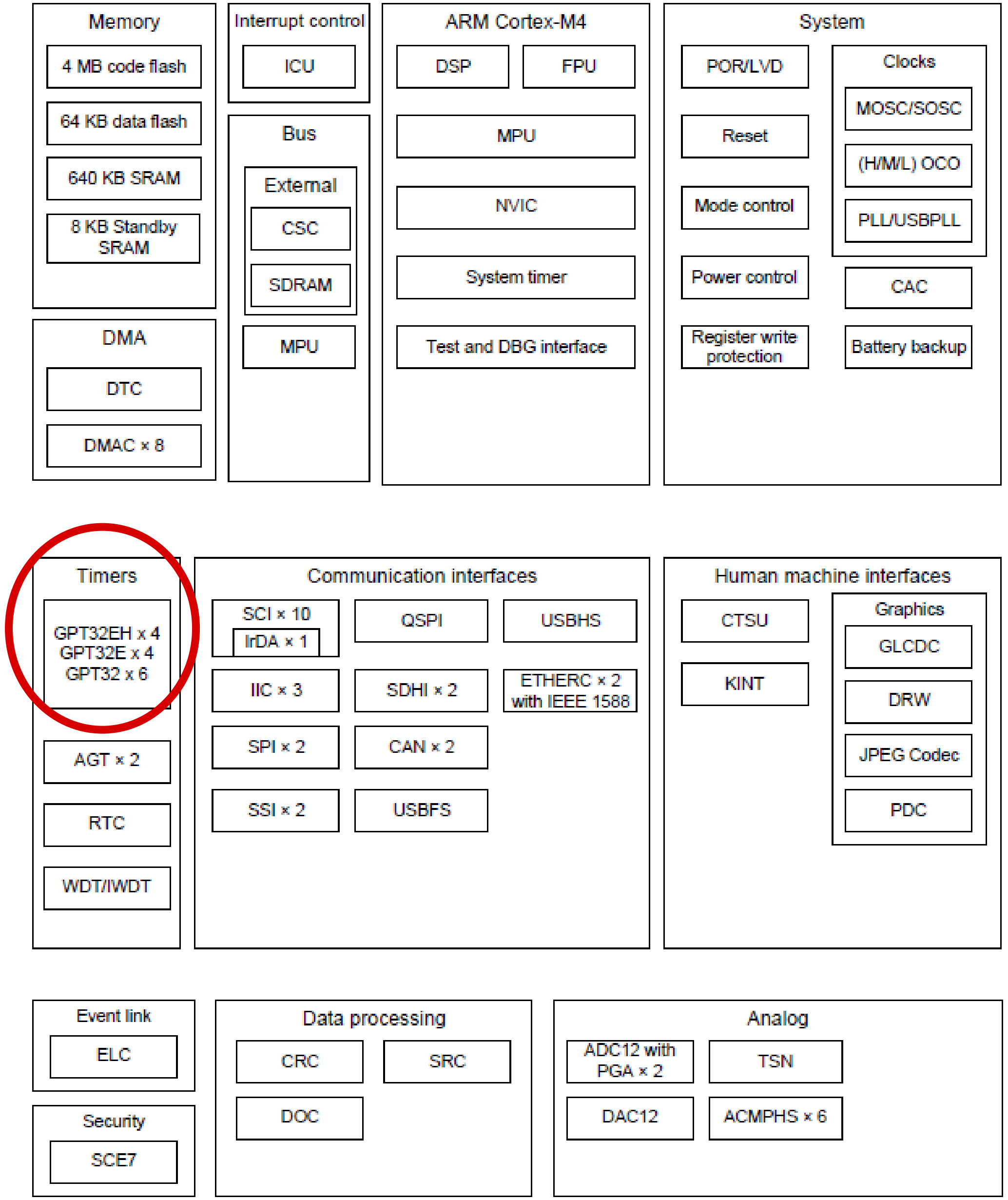
The Renesas S7G2 MCU has 14 timers of the type GPT (General PWM Timer).

Each one is a complex circuit that provides significant flexibility for a variety of applications.

TIMER CASE STUDY 2 – S7G2 GPT

Diagram shows the hardware modules of the S7G2 MCU.

General PWM Timers (GPT) are identified.



Memory
4 MB code flash
64 KB data flash
640 KB SRAM
8 KB Standby SRAM

Interrupt control
ICU

Bus
External
CSC
SDRAM
MPU

ARM Cortex-M4
DSP
FPU
MPU
NVIC
System timer
Test and DBG interface

System
POR/LVD
Reset
Mode control
Power control
Register write protection
Clocks
MOSC/SOSC
(H/M/L) OCO
PLL/USBPLL
CAC
Battery backup

DMA
DTC
DMAC × 8

Timers
GPT32EH x 4
GPT32E x 4
GPT32 x 6
AGT × 2
RTC
WDT/IWDT

Communication interfaces
SCI × 10
IrDA × 1
QSPI
USBHS
IIC × 3
SDHI × 2
ETHERC × 2 with IEEE 1588
SPI × 2
CAN × 2
SSI × 2
USBFS

Human machine interfaces
CTSU
KINT
Graphics
GLCDC
DRW
JPEG Codec
PDC

Event link
ELC

Security
SCE7

Data processing
CRC
SRC
DOC

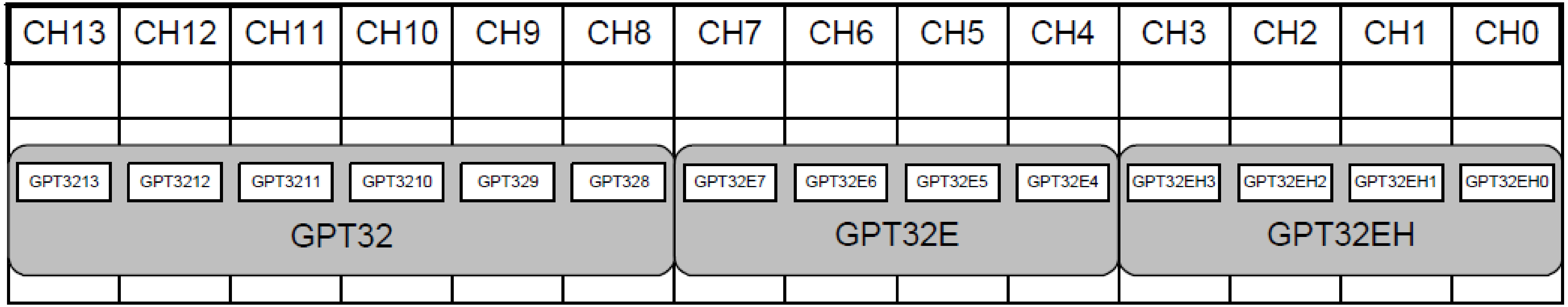
Analog
ADC12 with PGA × 2
TSN
DAC12
ACMPHS × 6

source: Renesas S7 Series Microcontrollers User's Manual

TIMER CASE STUDY 2 – S7G2 GPT

There are 14 timers, grouped into:

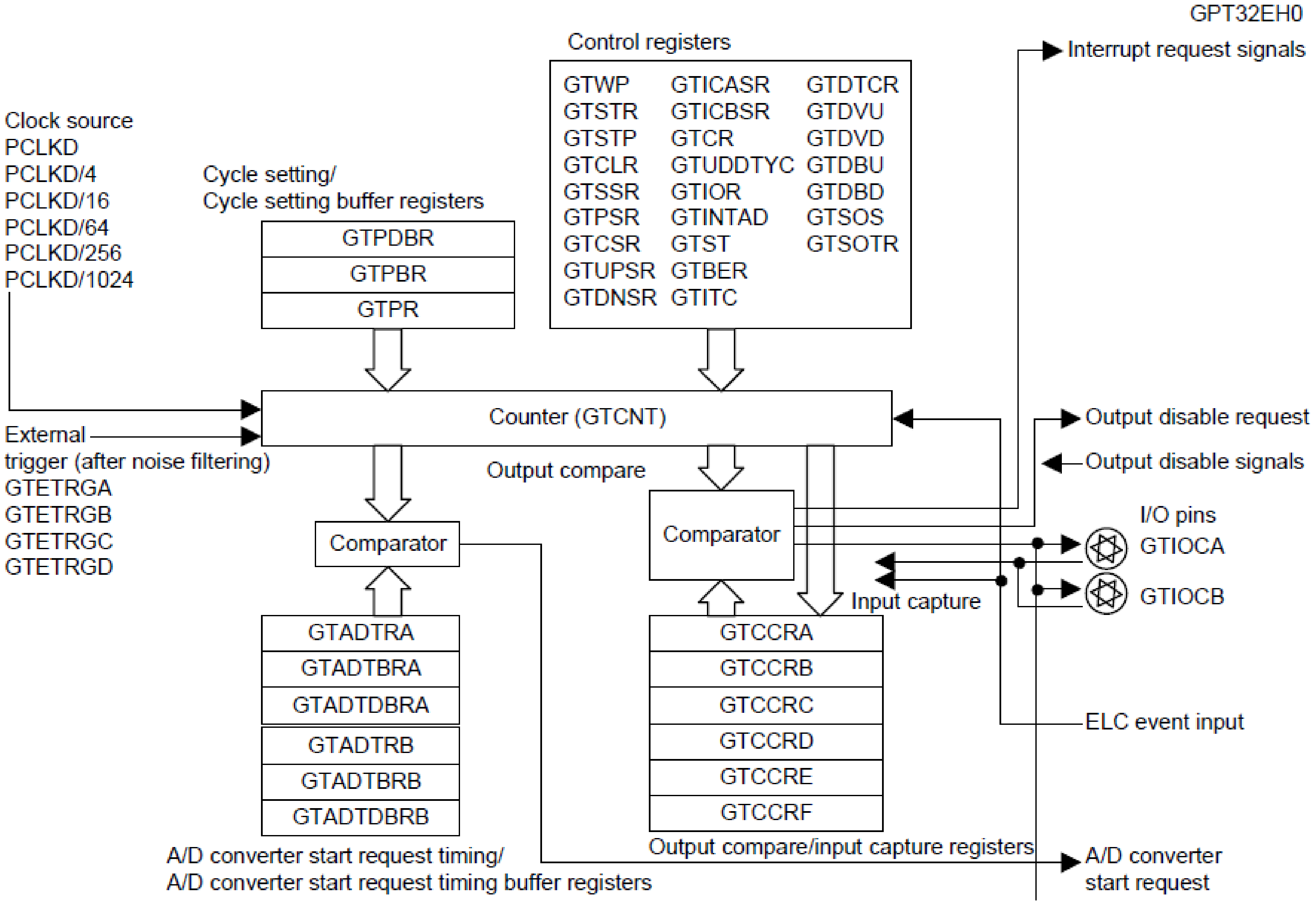
- 4x EH - enhanced high resolution
- 4x E - enhanced
- 6x - conventional



CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
GPT3213	GPT3212	GPT3211	GPT3210	GPT329	GPT328	GPT32E7	GPT32E6	GPT32E5	GPT32E4	GPT32EH3	GPT32EH2	GPT32EH1	GPT32EH0
GPT32						GPT32E				GPT32EH			

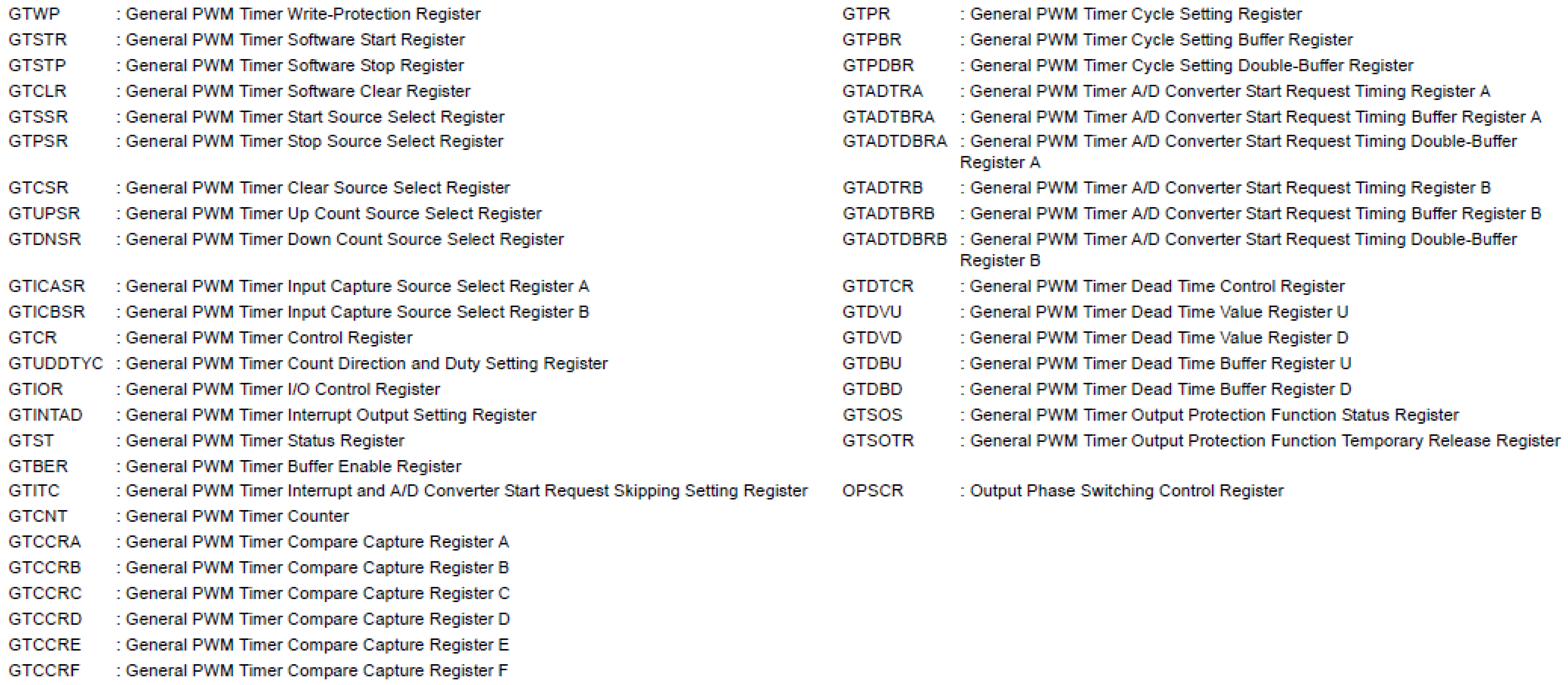
source: Renesas S7 Series Microcontrollers User’s Manual

TIMER CASE STUDY 2 – S7G2 GPT



source: Renesas S7 Series Microcontrollers User's Manual

TIMER CASE STUDY 2 – S7G2 GPT

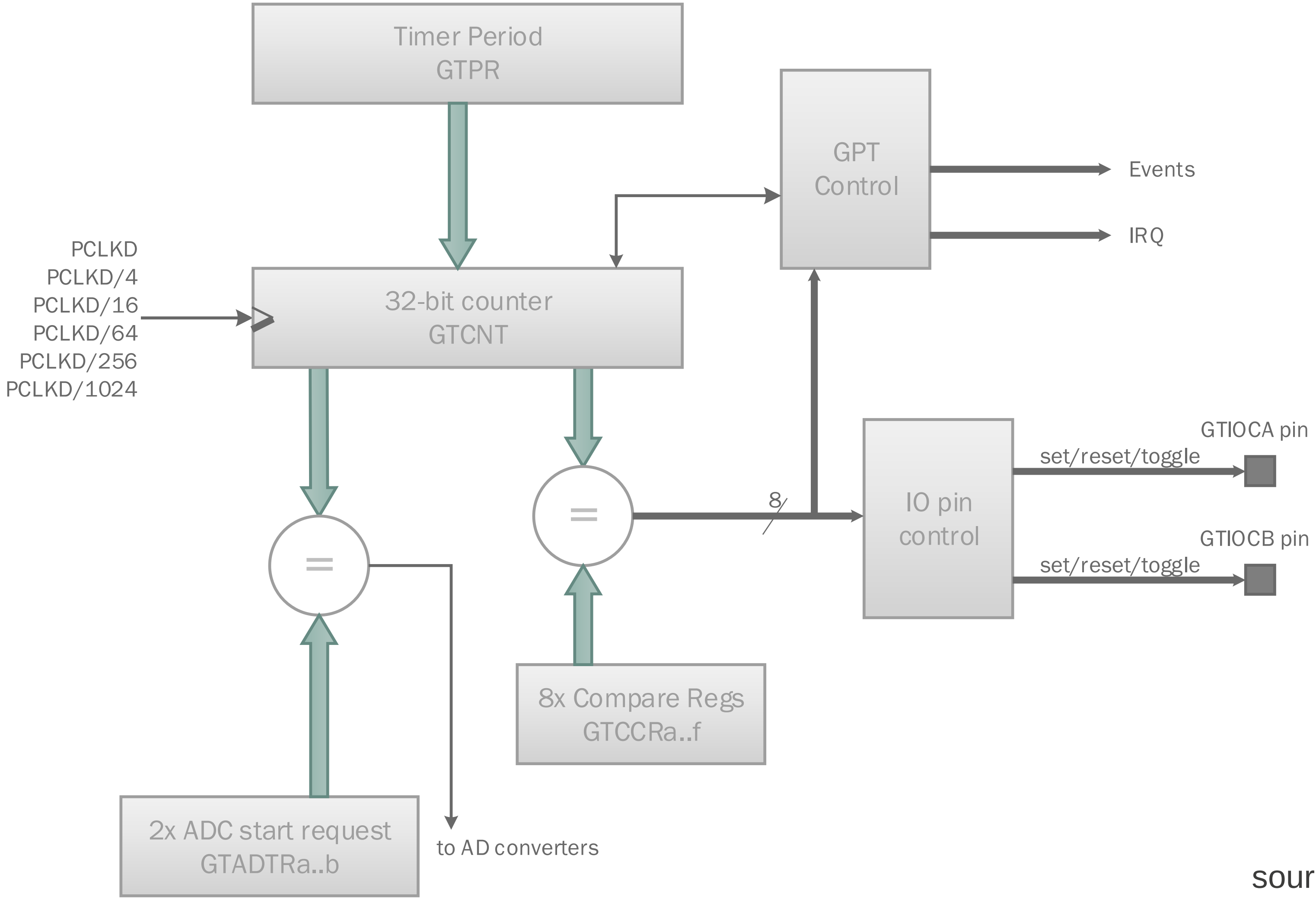


GTWP	: General PWM Timer Write-Protection Register
GTSTR	: General PWM Timer Software Start Register
GTSTP	: General PWM Timer Software Stop Register
GTCLR	: General PWM Timer Software Clear Register
GTSSR	: General PWM Timer Start Source Select Register
GTPSR	: General PWM Timer Stop Source Select Register
GTCSR	: General PWM Timer Clear Source Select Register
GTUPSR	: General PWM Timer Up Count Source Select Register
GTDNSR	: General PWM Timer Down Count Source Select Register
GTICASR	: General PWM Timer Input Capture Source Select Register A
GTICBSR	: General PWM Timer Input Capture Source Select Register B
GTCR	: General PWM Timer Control Register
GTUDDTYC	: General PWM Timer Count Direction and Duty Setting Register
GTIOR	: General PWM Timer I/O Control Register
GTINTAD	: General PWM Timer Interrupt Output Setting Register
GTST	: General PWM Timer Status Register
GTBER	: General PWM Timer Buffer Enable Register
GTITC	: General PWM Timer Interrupt and A/D Converter Start Request Skipping Setting Register
GTCNT	: General PWM Timer Counter
GTCCRA	: General PWM Timer Compare Capture Register A
GTCCRB	: General PWM Timer Compare Capture Register B
GTCCRC	: General PWM Timer Compare Capture Register C
GTCCRD	: General PWM Timer Compare Capture Register D
GTCCRE	: General PWM Timer Compare Capture Register E
GTCCRF	: General PWM Timer Compare Capture Register F
GTPR	: General PWM Timer Cycle Setting Register
GTPBR	: General PWM Timer Cycle Setting Buffer Register
GTPDBR	: General PWM Timer Cycle Setting Double-Buffer Register
GTADTRA	: General PWM Timer A/D Converter Start Request Timing Register A
GTADTBRA	: General PWM Timer A/D Converter Start Request Timing Buffer Register A
GTADTDBRA	: General PWM Timer A/D Converter Start Request Timing Double-Buffer Register A
GTADTRB	: General PWM Timer A/D Converter Start Request Timing Register B
GTADTBRB	: General PWM Timer A/D Converter Start Request Timing Buffer Register B
GTADTDBRB	: General PWM Timer A/D Converter Start Request Timing Double-Buffer Register B
GTDTCR	: General PWM Timer Dead Time Control Register
GTDVU	: General PWM Timer Dead Time Value Register U
GTDVD	: General PWM Timer Dead Time Value Register D
GTDBU	: General PWM Timer Dead Time Buffer Register U
GTDBD	: General PWM Timer Dead Time Buffer Register D
GTSOS	: General PWM Timer Output Protection Function Status Register
GTSOTR	: General PWM Timer Output Protection Function Temporary Release Register
OPSCR	: Output Phase Switching Control Register

source: Renesas S7 Series Microcontrollers User's Manual

TIMER CASE STUDY 2 – S7G2 GPT

Simplified view of a GPT



source: Authors

TIMER CASE STUDY 2 – S7G2 GPT

GPT characteristics and operation:

- The counter (GTCNT) can count up (from 0 to GTPR), count down (from GTPR to 0) or up and down (from 0 to GTPR and back to 0).

  Hence, the period is either GTPR+1 or 2x GTPR;
- Due to a prescaler, the clock source can be selected from PCLKD to PLCKD/1024;
- A GPT can be used to trigger the start of an AD conversion, registers GTADTR are used to determine the timing to command the start;
- 8 compare registers are available. On match, a number of actions can take place: set, reset, toggle an IO pin, generate an interrupt request, generate an event, ...

5.4 – GPIO

General Purpose Input/Output (GPIO) is possibly the simplest form of I/O in a system.

An input pin can be connected to a key, a push-button, or any other digital signal. By reading the pin the program can detect if it is on high or low level and take appropriate action.

Input pins can generate interrupt requests (IRQ) when a required transition or level is detected.

An output pin can be connected to an LED, to a switch (transistor, relay) or any other device to be controlled by.

GPIO CASE STUDY – S7G2 I/O PORTS

The MCU used in the lab experiments (kit SK-S7G2) is the R7FS7G27H3A01CFC. Its packaging is a 176 pins LQFP (Low-profile Quad Flat Package).

Of its 176 pins, 126 are available either for I/O or to be used by the integrated peripherals.

The I/O pins are grouped into 12 ports (P0 to PB) each with up to 16 pins.

Port	pins		Port	pins
P0	13		P6	16
P1	16		P7	8
P2	10		P8	7
P3	16		P9	6
P4	16		PA	5
P5	11		PB	2

GPIO CASE STUDY – S7G2 I/O PORTS

source: Renesas S7 Series Microcontrollers User's Manual

I/O pins can be configured in several ways:

- inputs may have an internal pull-up;
- outputs may be open-drain;
- output current capacity may be selectable: 2, 4, 16/20 mA;
- some inputs are 5V tolerant.

Remarks:

- total current provided by the MCU is restricted to 80 mA;
- configuration capabilities vary from pin to pin, see table on the right.

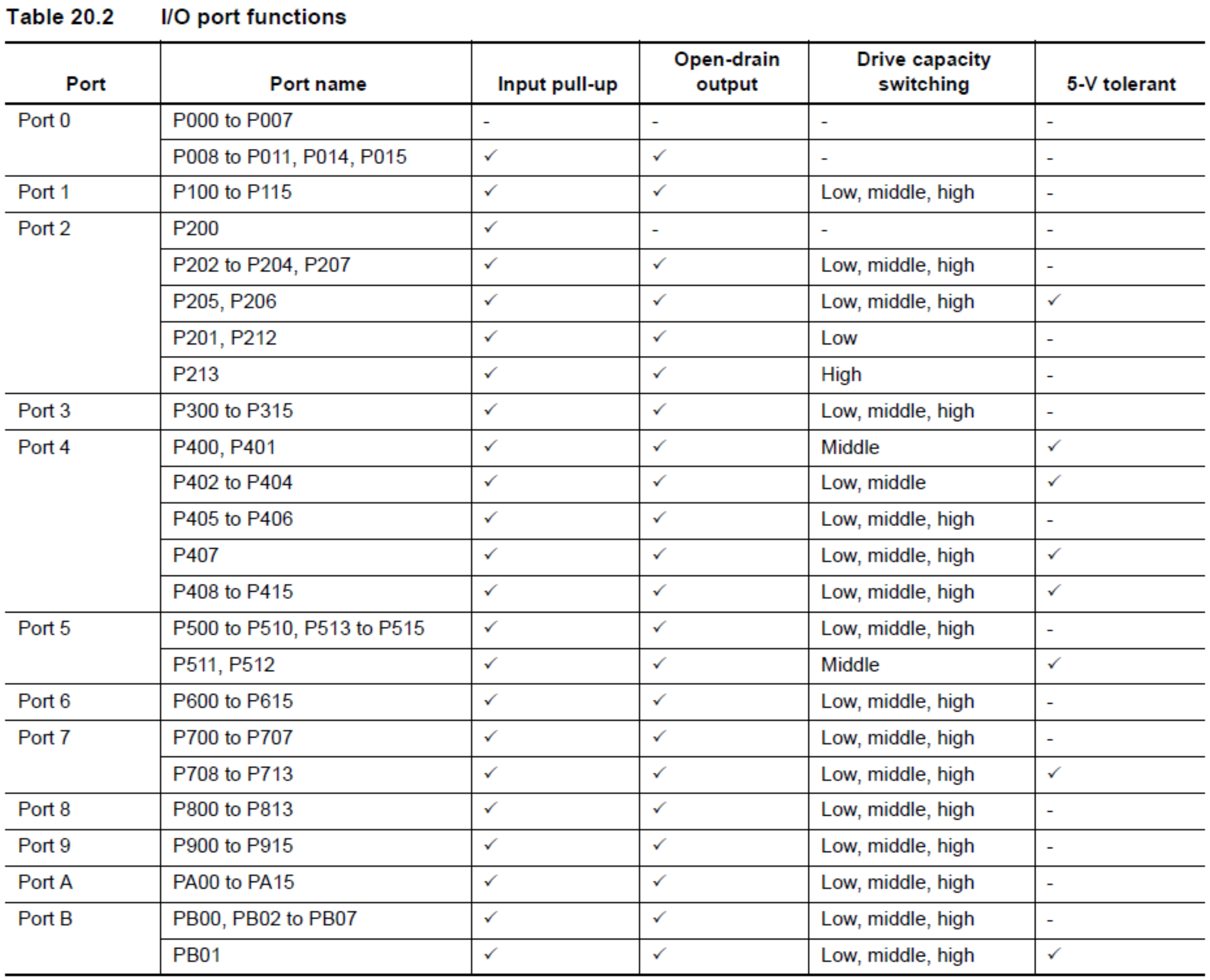


Table 20.2 I/O port functions

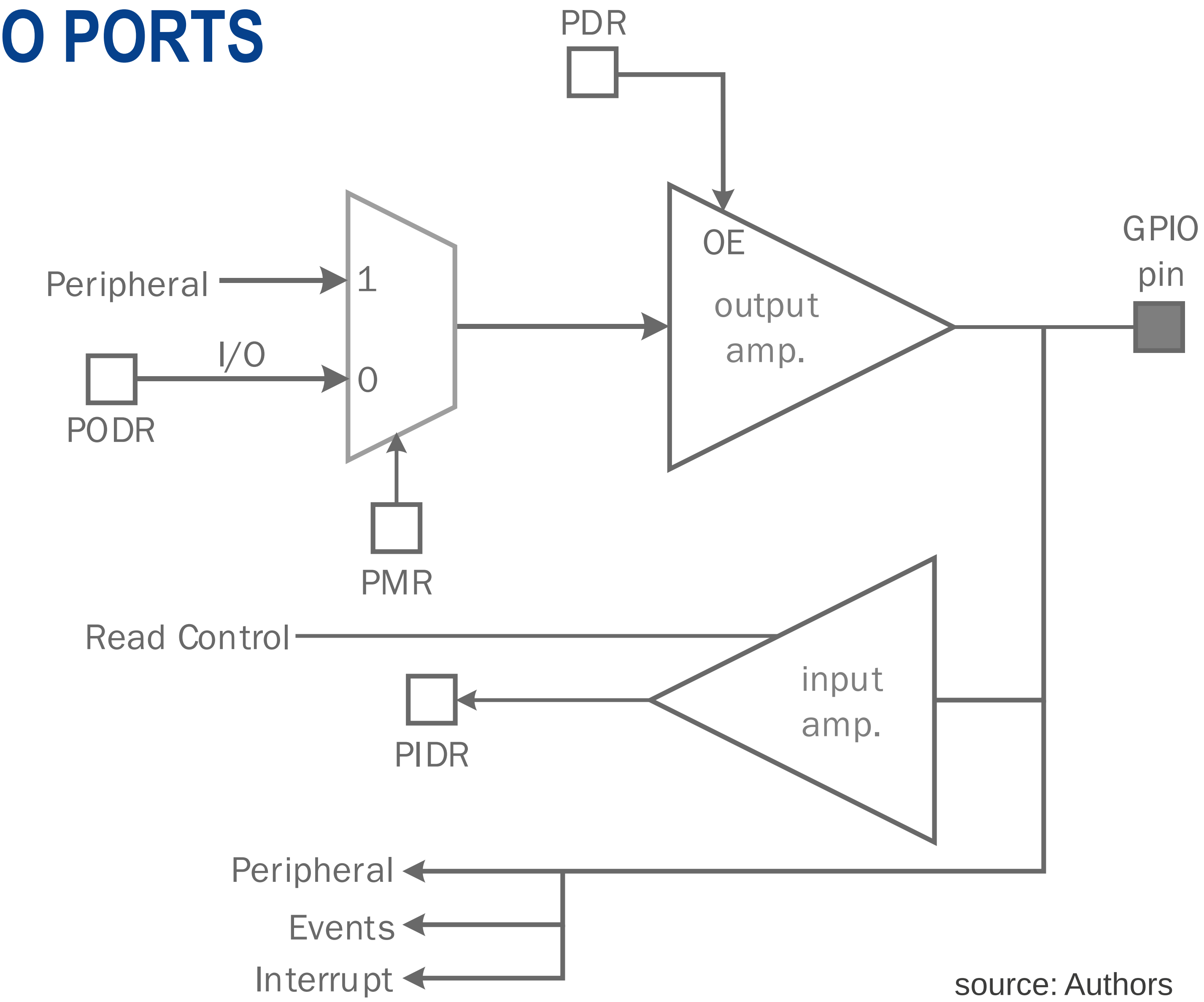
Port	Port name	Input pull-up	Open-drain output	Drive capacity switching	5-V tolerant
Port 0	P000 to P007	-	-	-	-
	P008 to P011, P014, P015	✓	✓	-	-
Port 1	P100 to P115	✓	✓	Low, middle, high	-
Port 2	P200	✓	-	-	-
	P202 to P204, P207	✓	✓	Low, middle, high	-
	P205, P206	✓	✓	Low, middle, high	✓
	P201, P212	✓	✓	Low	-
	P213	✓	✓	High	-
Port 3	P300 to P315	✓	✓	Low, middle, high	-
Port 4	P400, P401	✓	✓	Middle	✓
	P402 to P404	✓	✓	Low, middle	✓
	P405 to P406	✓	✓	Low, middle, high	-
	P407	✓	✓	Low, middle, high	✓
	P408 to P415	✓	✓	Low, middle, high	✓
Port 5	P500 to P510, P513 to P515	✓	✓	Low, middle, high	-
	P511, P512	✓	✓	Middle	✓
Port 6	P600 to P615	✓	✓	Low, middle, high	-
Port 7	P700 to P707	✓	✓	Low, middle, high	-
	P708 to P713	✓	✓	Low, middle, high	✓
Port 8	P800 to P813	✓	✓	Low, middle, high	-
Port 9	P900 to P915	✓	✓	Low, middle, high	-
Port A	PA00 to PA15	✓	✓	Low, middle, high	-
Port B	PB00, PB02 to PB07	✓	✓	Low, middle, high	-
	PB01	✓	✓	Low, middle, high	✓

rem: not all pins listed above are available on the part number R7FS7G27H3A01CFC

GPIO CASE STUDY – S7G2 I/O PORTS

Simplified block diagram

of an I/O pin



Reg	Description
PODR	Port output data
PDR	Port direction
PIDR	Port input data
PMR	Port mode: I/O vs periph.

source: Authors

GPIO CASE STUDY – S7G2 I/O PORTS

source: Renesas S7 Series Microcontrollers User’s Manual

Block diagram for an I/O pin

Reg	Description		Reg	Description
PCR	Pull-up control		PODR	Port output data
PDR	Port direction		PSEL	Peripheral sel.
DSCR	Drive capabil.		PIDR	Port input data
NCODR	open-drain ctr		ISEL	interrupt enable
EOSR	evt output set		ASEL	analog select
EORR	event output reset		PMR	Port mode: I/O vs periph.
POSR	port output set		EIDR	event input data
PORR	port output reset		EOF/ EOR	event on falling/ rising edge

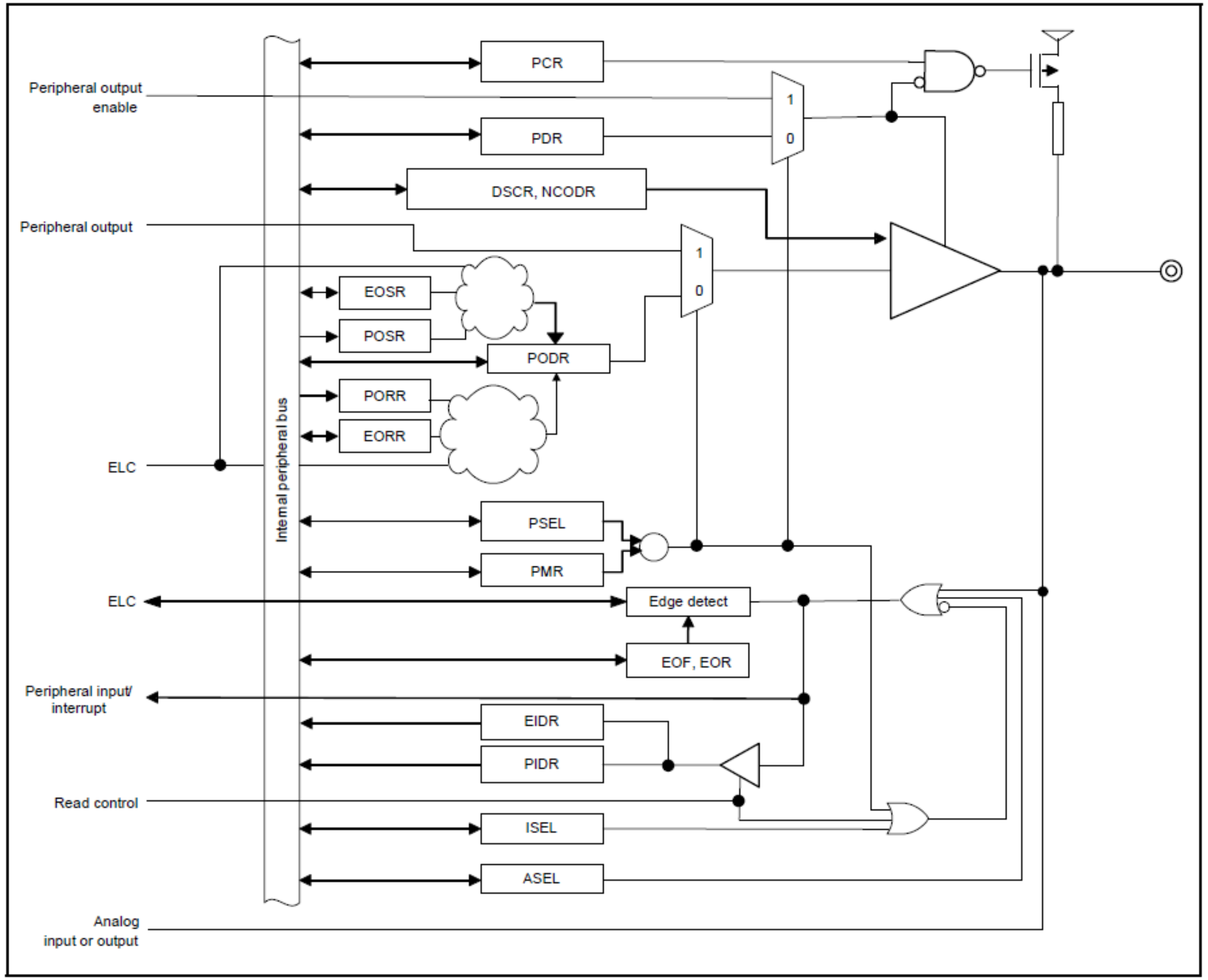


Figure 20.1 Connection diagram for I/O port registers

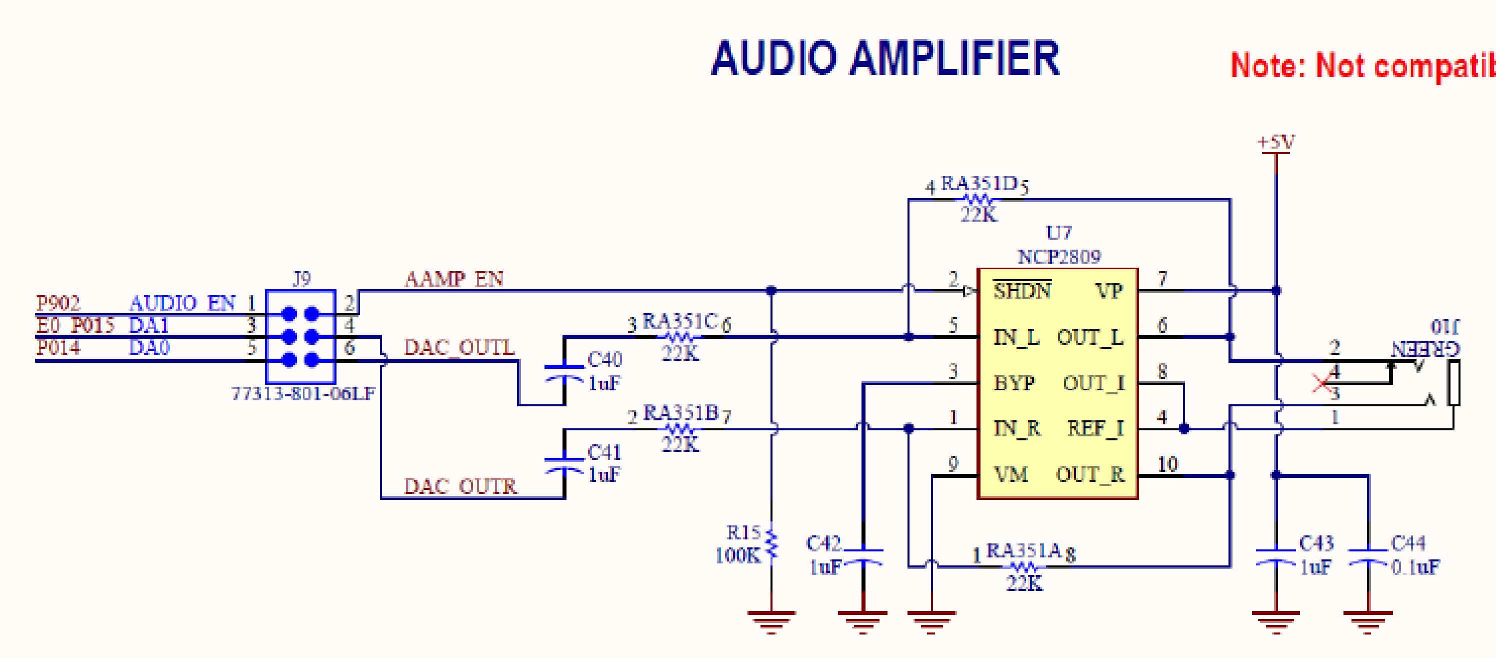
5.5 – LOW-POWER DRIVERS

When a device connected to an output pin has low input current requirements (low typically means below 1 mA) then it can be directly connected to the I/O pin.

Example: the Renesas Development Kit for S7G2 has an audio amplifier and a speaker connector.

This audio amplifier is controlled by an output pin:

- P902 is an I/O pin of the S7G2 microcontroller that controls if the amplifier is enabled or not.



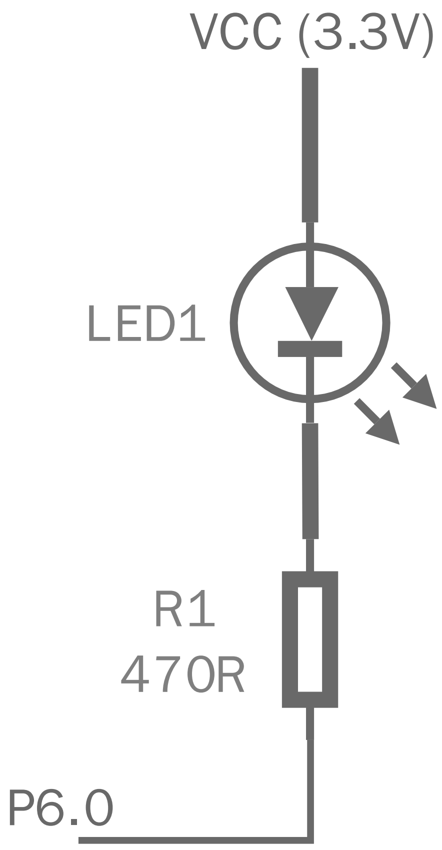
source: Renesas Development Kit S7G2 (DK-S7G2) User’s Manual

LOW-POWER DRIVERS

In the case of a small LED, which typically requires current around 2 to 5 mA to light up, a limited number can still be connected directly to output pins.

The S7G2 MCU has the same source and sink capability, meaning that an output pin configured for middle drive capacity can source 4mA when its output is high and can sink 4mA when its output is low.

This symmetry is not a rule, many digital logic ICs are capable to sink much higher currents than to source them. Hence, quite often, devices such as LEDs are turned on by an output pin sinking current, i.e. with a logic level 0.



P6.0 = 0 (0V) LED is ON
P6.0 = 1 (3.3V) LED is OFF

source: Authors

5.6 – POWER DRIVERS

Loads that require higher currents than in the previous examples are controlled by output pins connected to interface circuits, such as:

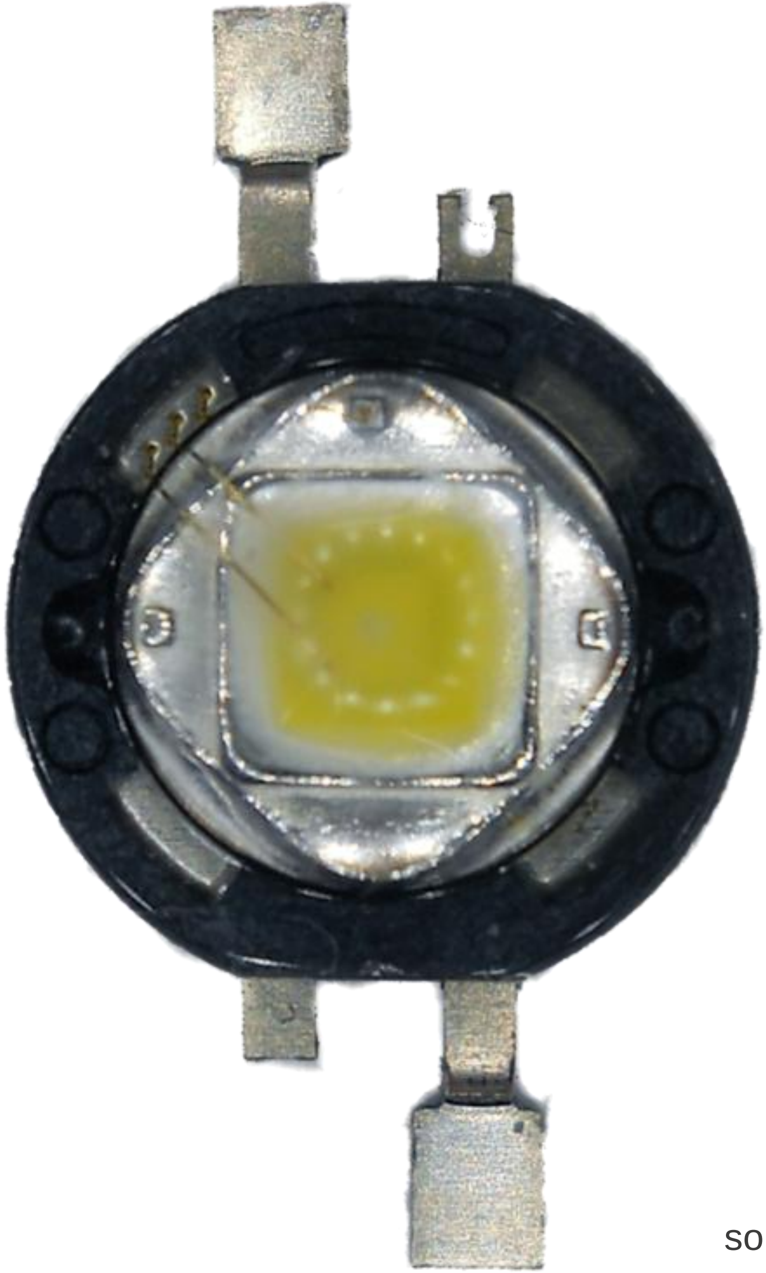
- Transistor (bipolar, MOS, or BICMOS),
- Relay,
- H-Bridge,
- Other power electronics circuits.

Such interface circuits also allow the use of much higher voltages on the load.

POWER DRIVERS

Example 1 - Power LED

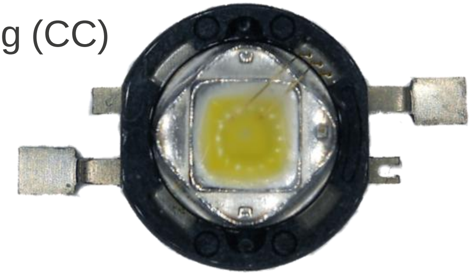
Consider a Power LED requiring a forward current of 500mA with a forward voltage of 3.2V ± 0.15V (varies with ambient temperature and component sample).



source: wikimedia.org (CC)

POWER DRIVERS

source: wikimedia.org (CC)



Example 1 - Power LED

A low-cost solution consists of a transistor, acting as a power switch, and a series resistance, to limit the forward current.

When P6.0 is at logic level 1, Q1 is on and LED is on.

When P6.0 is at logic level 0, Q1 and LED are off.

Drawbacks:

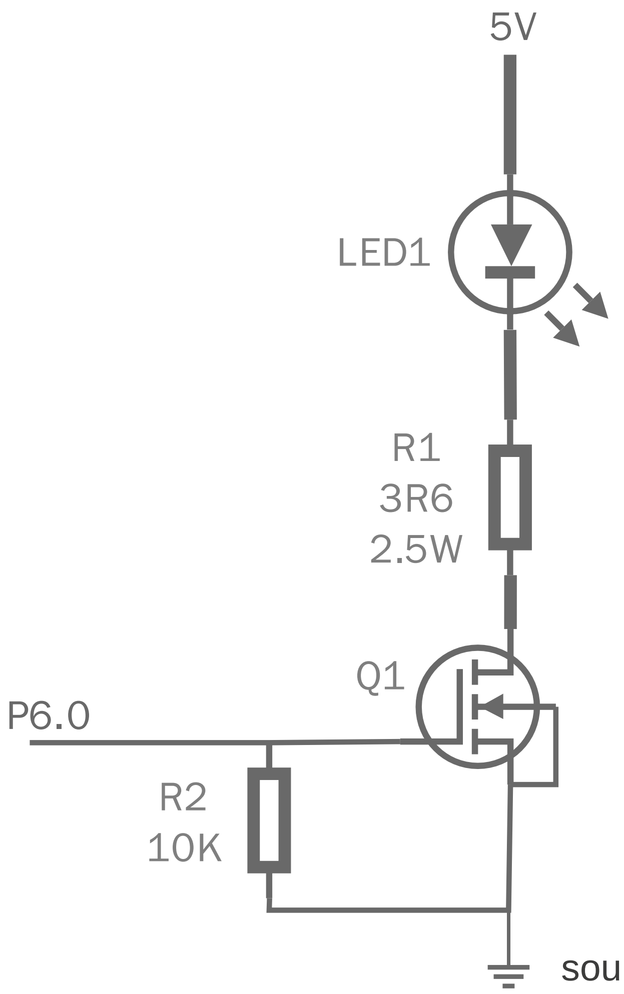
R1 dissipates 1W when LED is on.

LED current, and brightness, vary with LED forward voltage.

Advantages:

Load can operate from a power source with different voltage.

(in this example, MCU VDD is 3.3V and LED is powered from 5V)



source: Authors

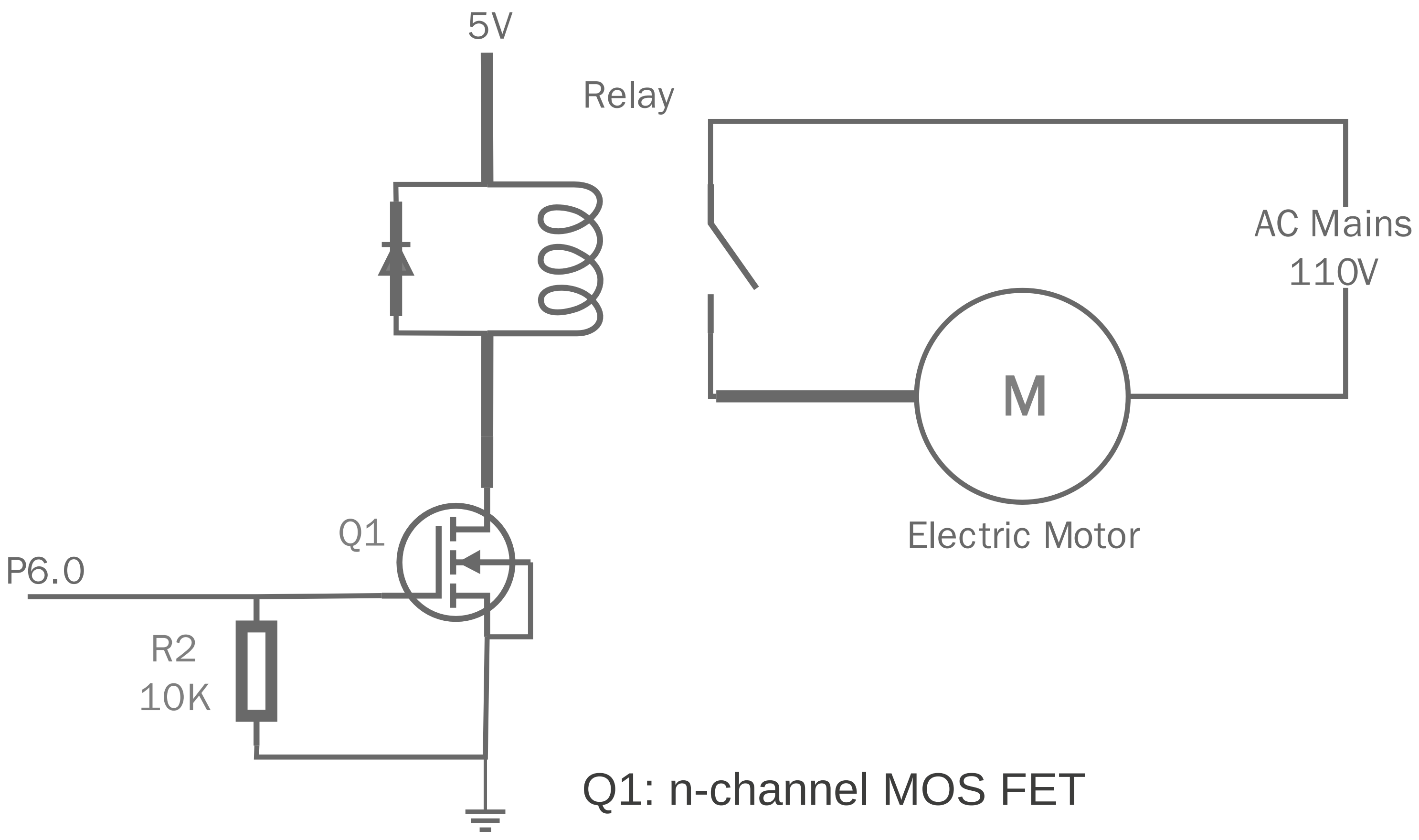
POWER DRIVERS

Example 2 - Relay

If galvanic isolation between the MCU and the load is required, a solution is the use of a relay.

When P6.0 is at logic level 1, the transistor is ON, current flows through the relay coil and the contact closes, turning the electric motor ON. When P6.0 is at 0, the motor is off.

Since the relay coil is an inductive load to the transistor, the diode is required to avoid voltage spikes when the transistor switches off.



Q1: n-channel MOS FET

source: Authors

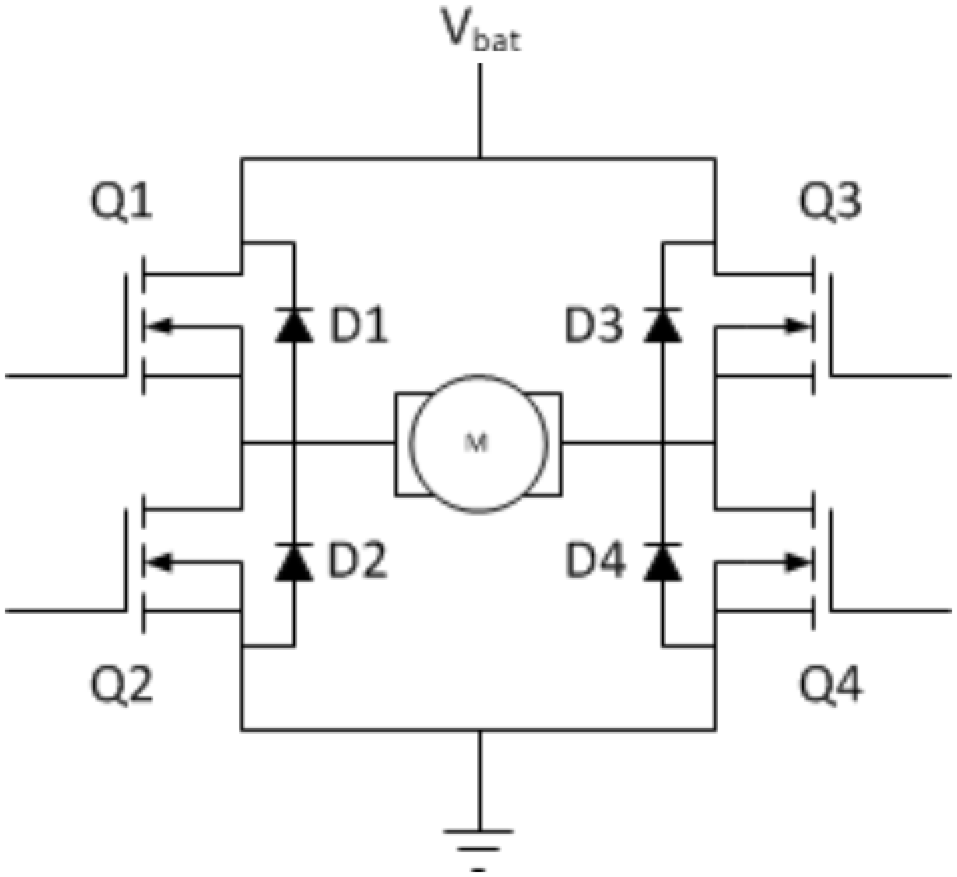
POWER DRIVERS

Example 3 - H-Bridge

An H-Bridge consists of four switches (in this case four n-mos). The load is a DC motor. When Q1 and Q4 are ON, the motor is energized. When Q2 and Q3 are ON the polarity is reversed.

Q1 and Q2 should never be ON at the same time as this would short circuit the power supply. For the same reason, Q3 and Q4 should never be ON at the same time.

Typically, the four switches are driven by PWM signals to control the current and the speed of the motor.



POWER DRIVERS

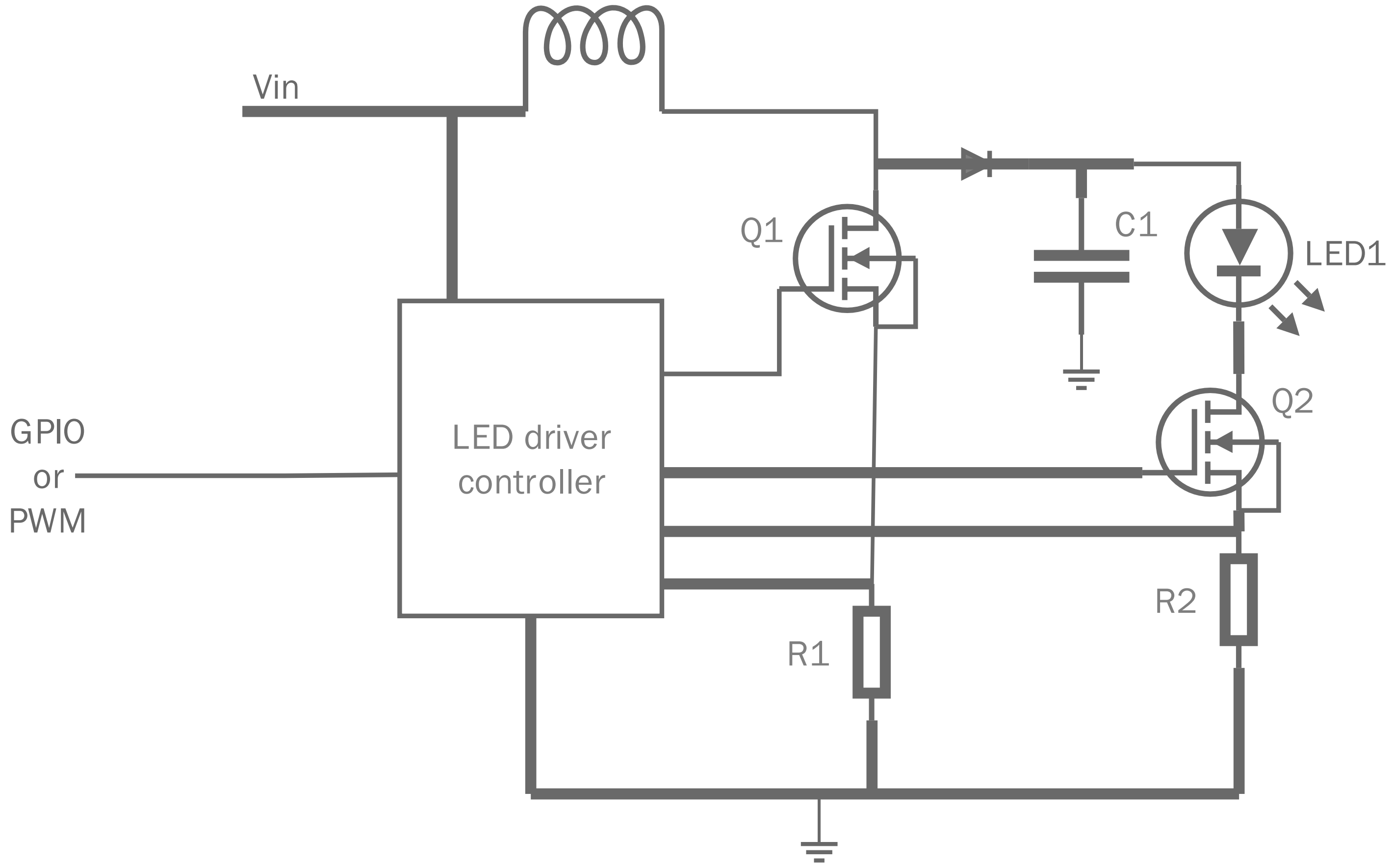
Example 4 -

Power LED driver using a switching power supply.

GPIO can turn LED on or OFF.

Alternatively, PWM can dimmer the LED.

When compared to Example 1, this circuit does not waste energy on a current limiting resistor, hence, it is energy efficient.



source: Authors

6 – INTERRUPT CONTROLLER

- NVIC
  - Structure
  - Registers
- CMSIS interface for NVIC operations

NVIC

The Nested Vectored Interrupt Controller (NVIC) is an integral part of the Cortex-M4 architecture. Hence, all Cortex-M4 have the same interrupt controller.

Exception handling was examined in Section 3 (link)

Functionality of the NVIC:

- **Detect** interrupt requests (IRQ) at its inputs and **combine** them into a single interrupt request to the microprocessor core
- Capability to **mask** any given input
- Associate inputs to interrupt **priority** levels

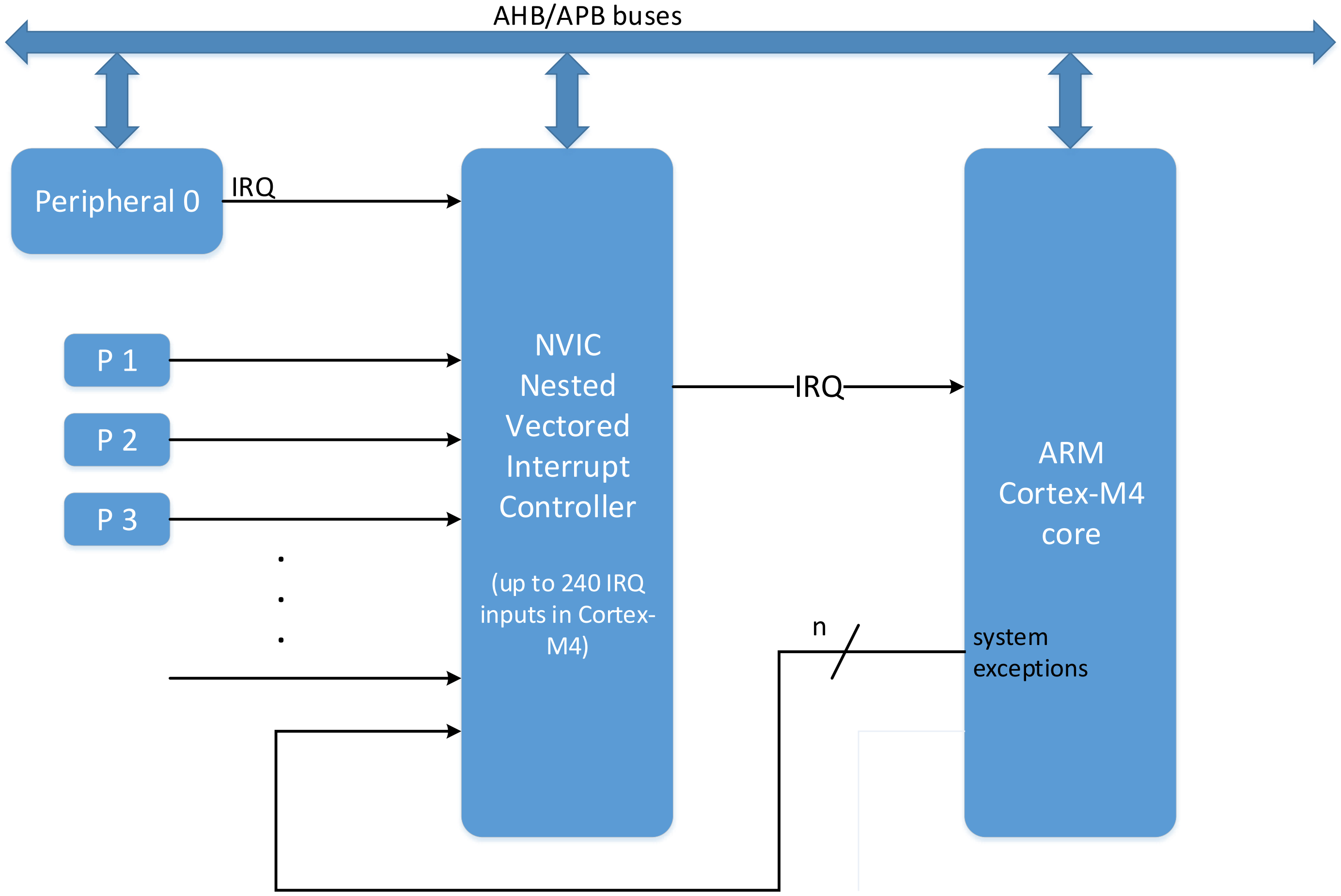
NVIC

The Cortex-M4 NVIC may have up to 240 IRQ inputs.

Each input is an interrupt request coming from an integrated peripheral, an external pin, or even from the core itself to inform of a system exception.

The operation of the NVIC is configured by the core via a set of NVIC registers that provide the functionality of:

- input masking
- IRQ priority assignment
- check status of input
  (is there a pending IRQ?)



source: Authors

NVIC CHARACTERISTICS

The Nested Vectored Interrupt Controller provides support for:

- **Nested exceptions** – each exception has an associated priority level. Nesting means that when an exception is being serviced, it can be interrupted by a higher priority exception that was activated. Once the higher priority exception has its service completed, the lower priority service resumes.
- **Vectored exceptions** – the starting addresses of the service routines (handlers) are stored in a table (vector table). When IRQi is detected the processor simply reads entry i of the table and starts servicing, avoiding delays to start the handler.
- **Interrupt masking** – each IRQi can be individually masked, i.e. ignored.

VECTOR TABLE

By default, its start address is 0x0.

The table has a 31-bit address for each of the possible exceptions (number is implementation dependent).

The 32th bit (the LSb) is used to identify the instruction set (ARM or Thumb) and must be always set in a Cortex-M processor

The first entry in the table is the initial value of the SP.

The next 15 entries are for the system exceptions
(Reset, NMI, … SysTick).

Then follow up to 240 entries for IRQ0 to IRQ239.

VECTOR TABLE FOR THE RENESAS S7G2

On the S7G2 the interrupt sources are managed by the ICU – Interrupt Control Unit. The ICU sits between the peripherals and the NVIC, preprocessing the interrupt requests before sending them to the NVIC.

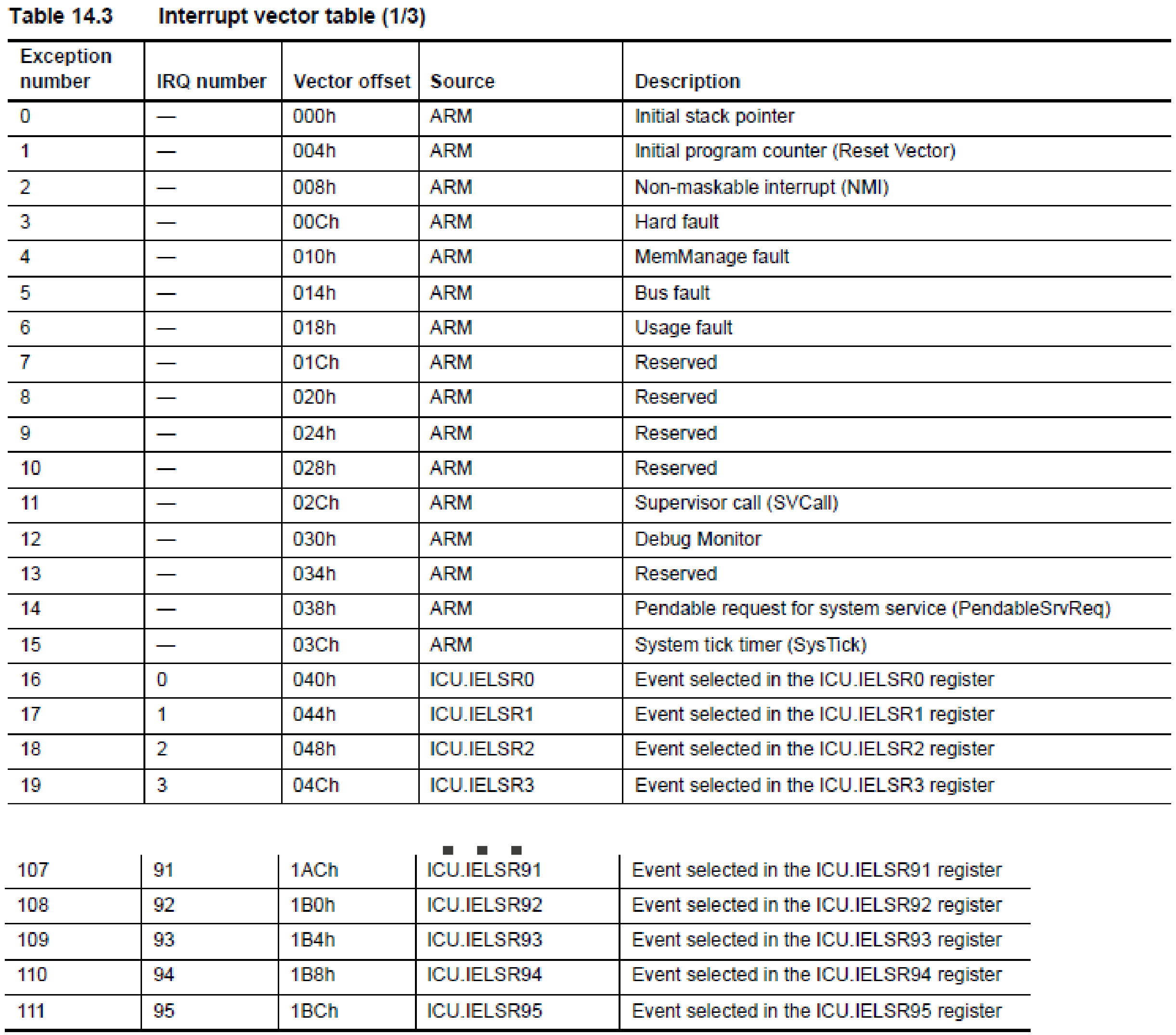
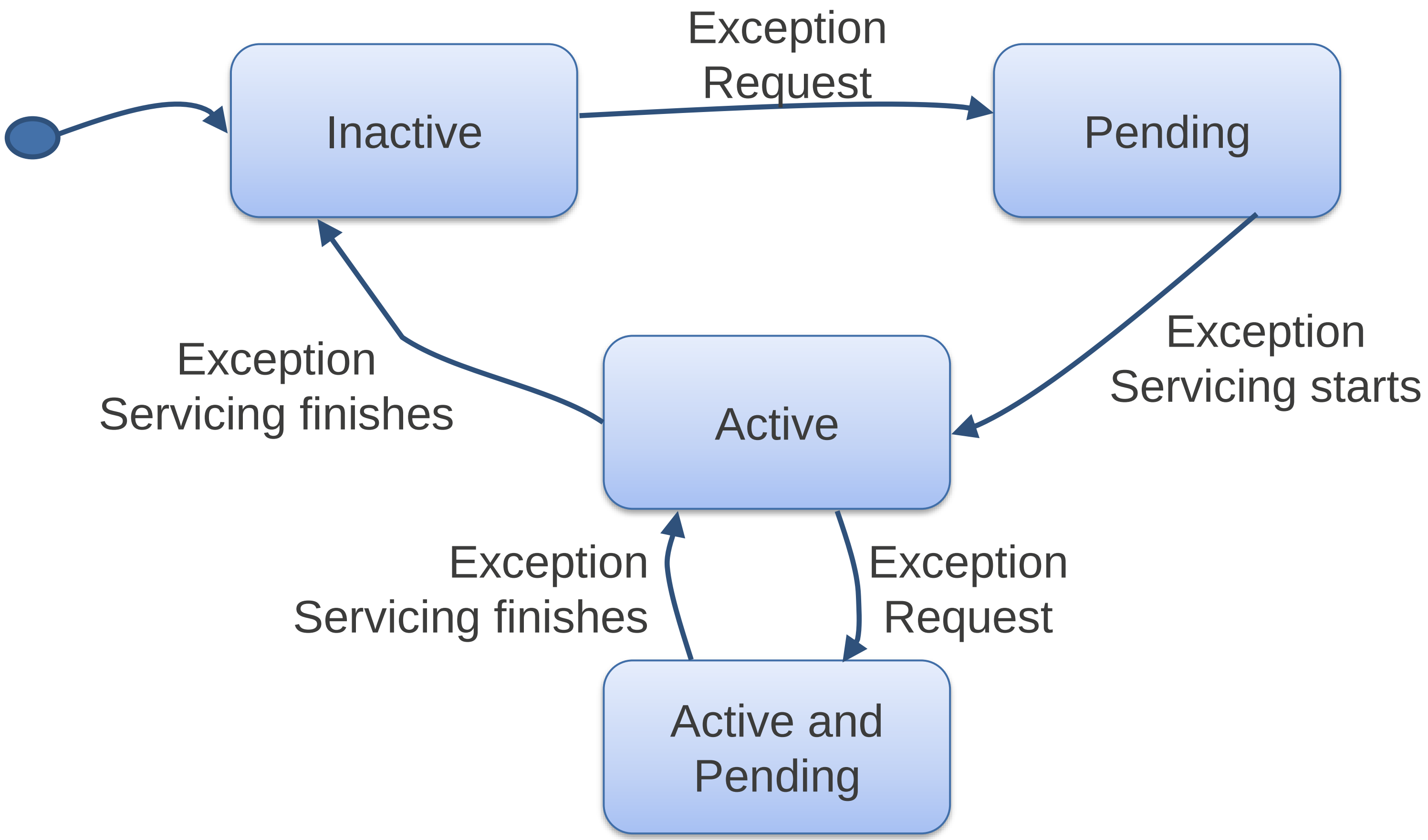


Table 14.3 Interrupt vector table (1/3)

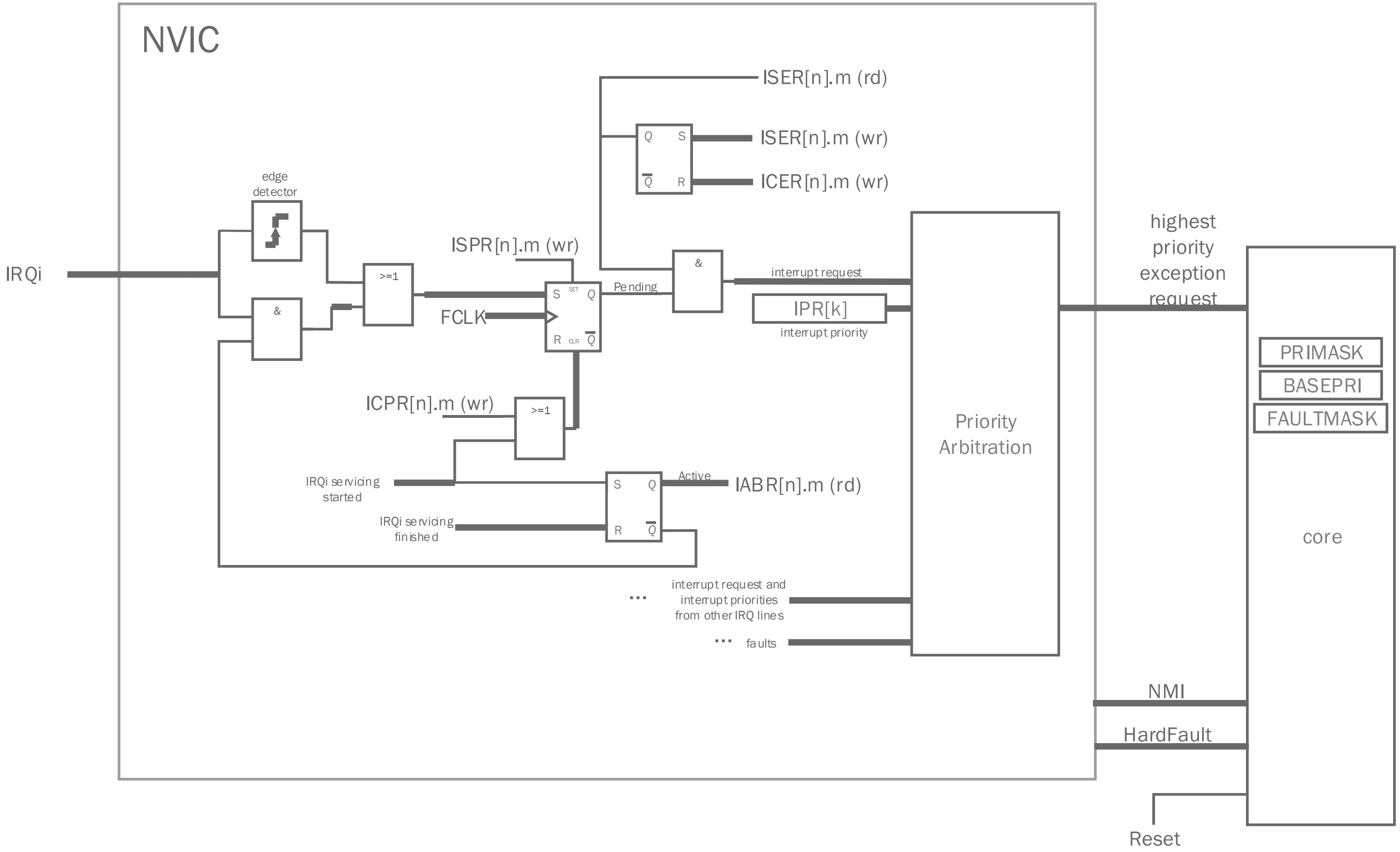
Exception number	IRQ number	Vector offset	Source	Description
0	—	000h	ARM	Initial stack pointer
1	—	004h	ARM	Initial program counter (Reset Vector)
2	—	008h	ARM	Non-maskable interrupt (NMI)
3	—	00Ch	ARM	Hard fault
4	—	010h	ARM	MemManage fault
5	—	014h	ARM	Bus fault
6	—	018h	ARM	Usage fault
7	—	01Ch	ARM	Reserved
8	—	020h	ARM	Reserved
9	—	024h	ARM	Reserved
10	—	028h	ARM	Reserved
11	—	02Ch	ARM	Supervisor call (SVCall)
12	—	030h	ARM	Debug Monitor
13	—	034h	ARM	Reserved
14	—	038h	ARM	Pendable request for system service (PendableSrvReq)
15	—	03Ch	ARM	System tick timer (SysTick)
16	0	040h	ICU.IELSR0	Event selected in the ICU.IELSR0 register
17	1	044h	ICU.IELSR1	Event selected in the ICU.IELSR1 register
18	2	048h	ICU.IELSR2	Event selected in the ICU.IELSR2 register
19	3	04Ch	ICU.IELSR3	Event selected in the ICU.IELSR3 register
			■ ■ ■	
107	91	1ACh	ICU.IELSR91	Event selected in the ICU.IELSR91 register
108	92	1B0h	ICU.IELSR92	Event selected in the ICU.IELSR92 register
109	93	1B4h	ICU.IELSR93	Event selected in the ICU.IELSR93 register
110	94	1B8h	ICU.IELSR94	Event selected in the ICU.IELSR94 register
111	95	1BCh	ICU.IELSR95	Event selected in the ICU.IELSR95 register

EXCEPTION HANDLING STATE MACHINE



source: Authors

FUNCTIONAL MODEL OF THE NVIC



source: Authors

NVIC – ARM DOCUMENTATION

- ARMv7-M Architecture Reference Manual (2014)

  ARM DDI 0403E.b

- ARM® Cortex® -M4 Processor Revision: r0p1 (2015)

  Technical Reference Manual

  ARM 100166_0001_00_en

- Cortex™-M4 Devices Generic User Guide (2011)

  ARM DUI 0553A

NVIC REGISTERS

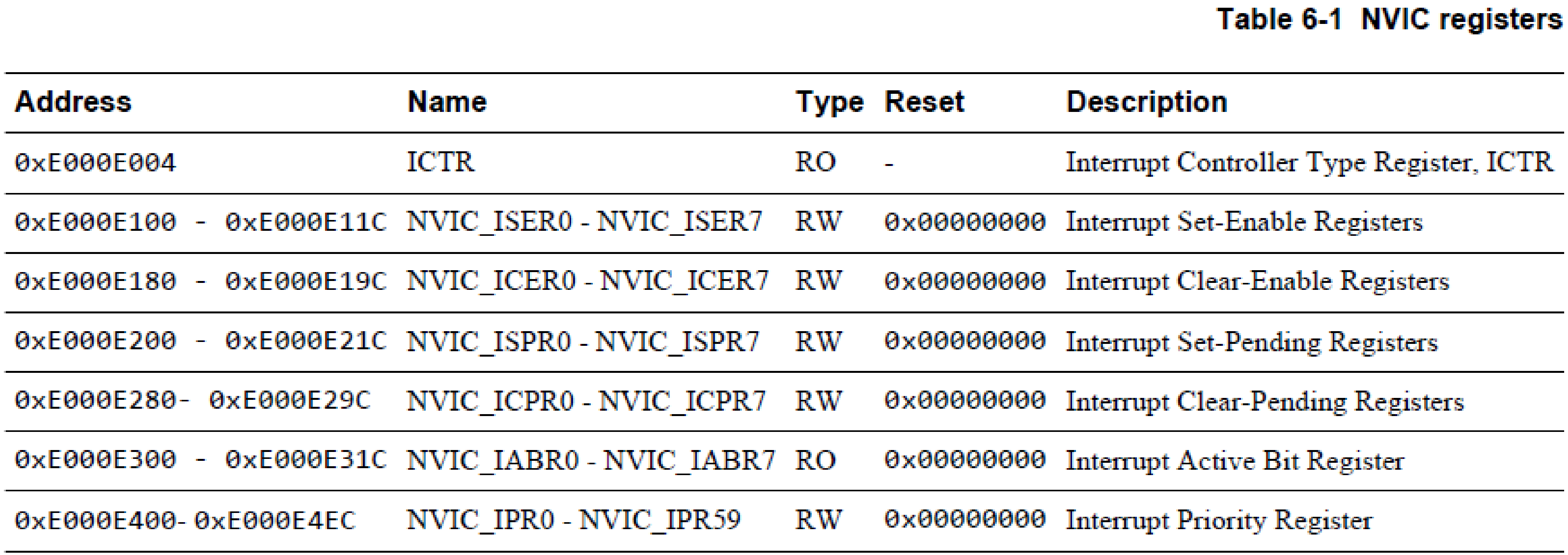


Table 6-1 NVIC registers

Address	Name	Type	Reset	Description
0xE000E004	ICTR	RO	-	Interrupt Controller Type Register, ICTR
0xE000E100 - 0xE000E11C	NVIC_ISER0 - NVIC_ISER7	RW	0x00000000	Interrupt Set-Enable Registers
0xE000E180 - 0xE000E19C	NVIC_ICER0 - NVIC_ICER7	RW	0x00000000	Interrupt Clear-Enable Registers
0xE000E200 - 0xE000E21C	NVIC_ISPR0 - NVIC_ISPR7	RW	0x00000000	Interrupt Set-Pending Registers
0xE000E280- 0xE000E29C	NVIC_ICPR0 - NVIC_ICPR7	RW	0x00000000	Interrupt Clear-Pending Registers
0xE000E300 - 0xE000E31C	NVIC_IABR0 - NVIC_IABR7	RO	0x00000000	Interrupt Active Bit Register
0xE000E400- 0xE000E4EC	NVIC_IPR0 - NVIC_IPR59	RW	0x00000000	Interrupt Priority Register

source: ARM® Cortex® -M4 Technical Reference Manual

NVIC REGISTERS – ICTR

Since the number of input lines (IRQi) to the NVIC is implementation dependent, so is the number of registers to control the NVIC.

ICTR informs how many control registers are actually implemented in any given NVIC.

For the Renesas S7G2, ICTR.INTLINESNUM holds the value 2, meaning that there are 3 registers of each type (ISER, ICER, ISPR, ICPR, IABR, with indexes 0..2; and 24 registers IPR, with indexes 0..23). The number of input lines IRQi on this processor is limited to 96.

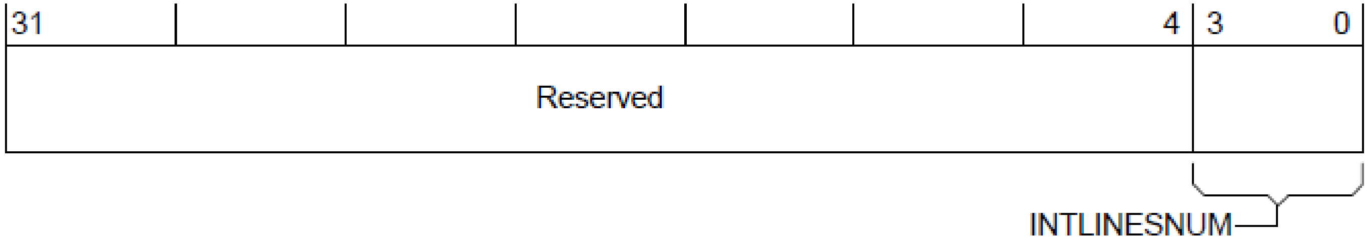
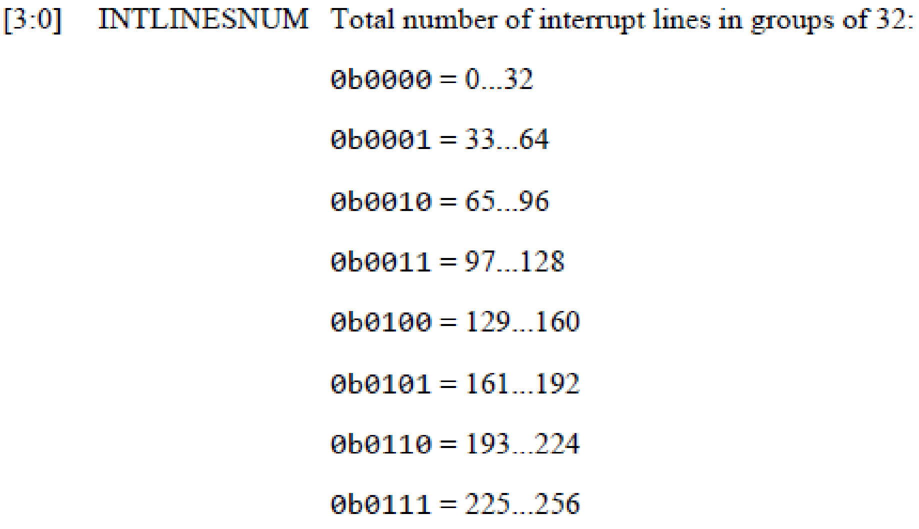


Figure 6-1 ICTR bit assignments



[3:0]	INTLINESNUM	Total number of interrupt lines in groups of 32: 0b0000 = 0...32 0b0001 = 33...64 0b0010 = 65...96 0b0011 = 97...128 0b0100 = 129...160 0b0101 = 161...192 0b0110 = 193...224 0b0111 = 225...256

source: ARM® Cortex® -M4 Technical Reference Manual

NVIC REGISTERS – ISER – S7G2

	bit31					bit0
ISER[0]	IRQ31	IRQ30	…	IRQ2	IRQ1	IRQ0
ISER[1]	IRQ63	IRQ62	…	IRQ34	IRQ33	IRQ32
ISER[2]	IRQ95	IRQ94	…	IRQ66	IRQ65	IRQ64

In a Renesas S7G2, each bit of registers ISER[0] to ISER[2] corresponds to one of the 96 IRQi lines.

on write: 1 enables the interrupt line, 0 has no effect

on read: returns the current state of each IRQi line

0 = disabled, 1 = enabled

NVIC REGISTERS – ICER/ISPR/ICPR/IABR – S7G2

Same bit assignment as for ISER.

ICER: on write: 1 **disables** the interrupt line, 0 has no effect

on read: report the current state of each IRQi line, 0 = disabled, 1 = enabled

ISPR: on write: 1 sets the interrupt line to **pending**, 0 has no effect

on read: report the current state of each IRQi line, 0 = not-pending, 1 = pending

ICPR: on write: 1 clears the pending state of the interrupt line, 0 has no effect

on read: report the current state of each IRQi line, 0 = not-pending, 1 = pending

IABR: read only: report the current state of each IRQi line, 0 = not-active, 1 = active

NVIC PRIORITIES

On the Cortex-M4, each IRQi line has a register that defines its priority level in relation to the other IRQi lines. This register may have up to 8 bits (implementation defined).

On the S7G2, 4-bit registers are implemented, allowing the definition of up to 16 priority levels. 0 is the highest priority and 15 is the lowest. Since these bits are left aligned, the actual values are 0, 0x10, 0x20, 0x30, … 0xF0.

A higher priority interrupt preempts a lower priority interrupt whose handler is being executed.

NVIC PRIORITIES

The bits of the priority register are divided into:

- Priority Group
- Priority Subgroup

The bits in the Priority Group define the number of priority levels for the purpose of preemption.

The bits in the Priority Subgroup define the number of sub-levels for the purpose of selecting which IRQi will be serviced first if two IRQi are simultaneously pending when the core accepts an interrupt.

NVIC PRIORITIES

The PRIGROUP bits of the AIRCR register are used to configure the division of bits among Priority Group and Subgroup.

AIRCR.PRIGROUP may be written with values in the range of 0 .. 7

On the S7G2, programming PRIGROUP with 0, 1, 2 or 3 has the effect of using the four bits for Priority Group and none for the Subgroup.

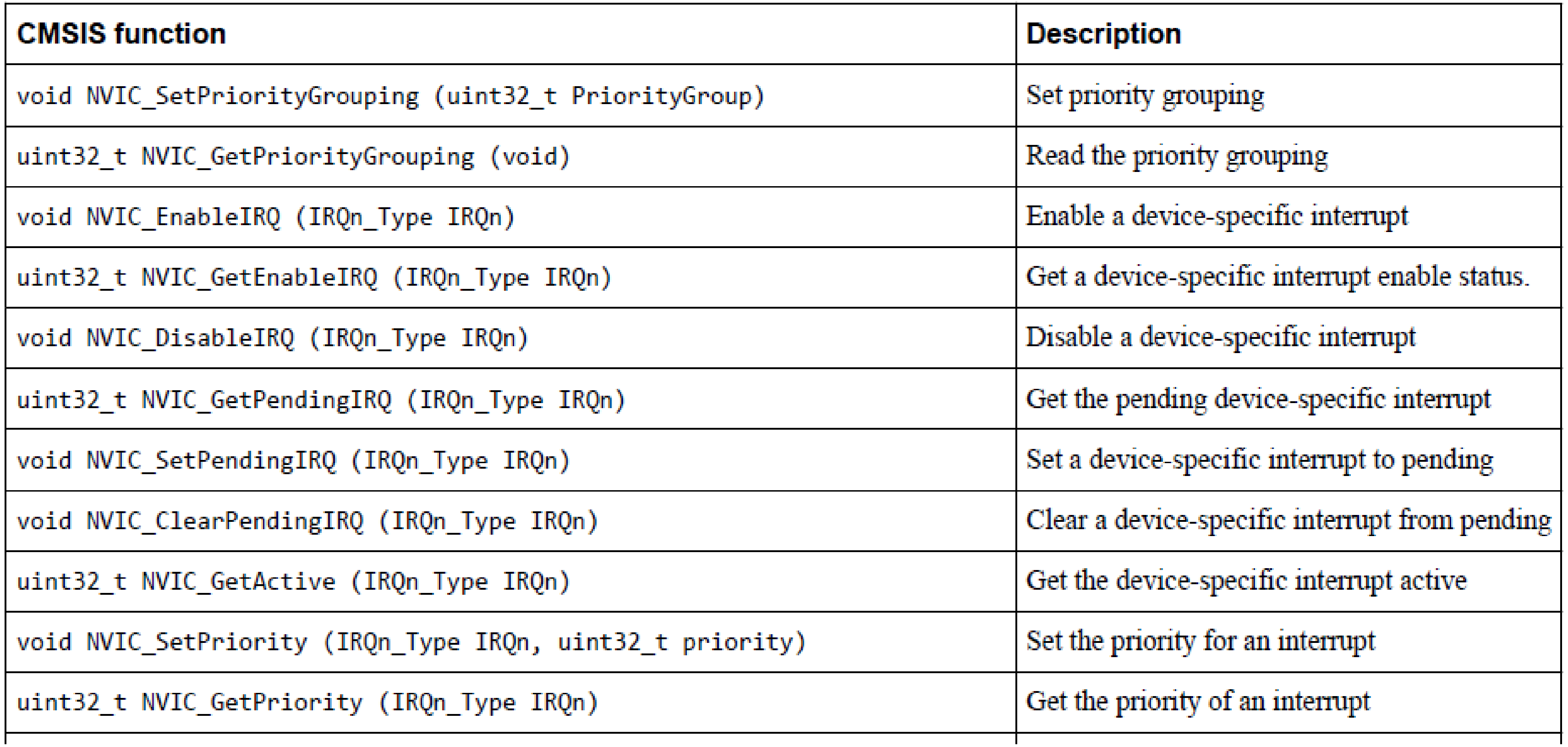
For PRIGROUP = 4 the division is 3.1 (3 for Group and 1 for Subgroup)

For PRIGROUP = 5 the division is 2.2, and so on…

CMSIS-CORE INTERRUPT FUNCTIONS

These is a partial list of the functions available in CMSIS-CORE that provide access to the NVIC as well as to other interrupt functionality.

Next slides detail these functions



CMSIS function	Description
`void NVIC_SetPriorityGrouping (uint32_t PriorityGroup)`	Set priority grouping
`uint32_t NVIC_GetPriorityGrouping (void)`	Read the priority grouping
`void NVIC_EnableIRQ (IRQn_Type IRQn)`	Enable a device-specific interrupt
`uint32_t NVIC_GetEnableIRQ (IRQn_Type IRQn)`	Get a device-specific interrupt enable status.
`void NVIC_DisableIRQ (IRQn_Type IRQn)`	Disable a device-specific interrupt
`uint32_t NVIC_GetPendingIRQ (IRQn_Type IRQn)`	Get the pending device-specific interrupt
`void NVIC_SetPendingIRQ (IRQn_Type IRQn)`	Set a device-specific interrupt to pending
`void NVIC_ClearPendingIRQ (IRQn_Type IRQn)`	Clear a device-specific interrupt from pending
`uint32_t NVIC_GetActive (IRQn_Type IRQn)`	Get the device-specific interrupt active
`void NVIC_SetPriority (IRQn_Type IRQn, uint32_t priority)`	Set the priority for an interrupt
`uint32_t NVIC_GetPriority (IRQn_Type IRQn)`	Get the priority of an interrupt

source: infocenter.arm.com

CMSIS-CORE INTERRUPT FUNCTIONS

```
typedef enum {
/* --------------------  Cortex-M4 Processor Exceptions Numbers  ------------------- */
  Reset_IRQn                = -15,      /*!<   1  Reset Vector, invoked on Power up and warm reset                  */
  NonMaskableInt_IRQn       = -14,      /*!<   2  Non maskable Interrupt, cannot be stopped or preempted           */
  HardFault_IRQn            = -13,      /*!<   3  Hard Fault, all classes of Fault                                 */
  MemoryManagement_IRQn     = -12,      /*!<   4  Memory Management, MPU mismatch, including Access Violation
                                                  and No Match                                                     */
  BusFault_IRQn             = -11,      /*!<   5  Bus Fault, Pre-Fetch-, Memory Access Fault, other address/memory
                                                  related Fault                                                    */
  UsageFault_IRQn           = -10,      /*!<   6  Usage Fault, i.e. Undef Instruction, Illegal State Transition    */
  SVCall_IRQn               =  -5,      /*!<  11  System Service Call via SVC instruction                          */
  DebugMonitor_IRQn         =  -4,      /*!<  12  Debug Monitor                                                    */
  PendSV_IRQn               =  -2,      /*!<  14  Pendable request for system service                              */
  SysTick_IRQn              =  -1,      /*!<  15  System Tick Timer                                                */
} IRQn_Type;
```

This is the IRQn_Type enumeration defined in file S7G2.h

CMSIS-CORE INTERRUPT FUNCTIONS

`void __enable_irq(void)`

Resets PRIMASK in the core, allowing interrupts from NVIC to be serviced.

Example: `__enable_irq( );`

`void __disable_irq(void)`

Sets PRIMASK in the core, preventing exceptions with configurable priorities (exceptions 4 and up) to be serviced.

Example: `__disable_irq( );`

CMSIS-CORE INTERRUPT FUNCTIONS

```
void NVIC_EnableIRQ(IRQn_Type)
```

Enables (**unmask**) a specific IRQi line

Example: `NVIC_Enable(SysTick_IRQn);`

```
void NVIC_DisableIRQ(IRQn_Type)
```

Disables (**mask**) a specific IRQi line

Example: `NVIC_Disable(SysTick_IRQn);`

CMSIS-CORE INTERRUPT FUNCTIONS

```
uint32_t NVIC_GetPendingIRQ(IRQn_Type)
```

Reads the Pending status (P_FF) returning 0 (not pending) or 1 (pending)

Example: `uint32_t pend = NVIC_GetPendingIRQ(SysTick_IRQn);`

```
void NVIC_SetPendingIRQ(IRQn_Type)
```

Sets the Pending status (P-FF) of a specific IRQi line

Example: `NVIC_SetPendingIRQ(SysTick_IRQn);`

```
void NVIC_ClearPendingIRQ(IRQn_Type)
```

Clears (resets) the Pending status (P-FF) of a specific IRQi line

Example: `NVIC_ClearPendingIRQ(SysTick_IRQn);`

CMSIS-CORE INTERRUPT FUNCTIONS

```
void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)
```

Sets the priority of a specific IRQi line

Example: `NVIC_SetPriority(SysTick_IRQn,4);`

```
uint32_t NVIC_GetPriority(IRQn_Type IRQn)
```

Get the priority level of a specific IRQi line

Example: `uint32_t prio = NVIC_GetPriority(SysTick_IRQn);`

Rem: The integer value of the priority must be in the range of 0 .. 2N-1, where

N is the number of priority bits implemented. For S7G2 the range is 0..15.

CMSIS-CORE INTERRUPT FUNCTIONS

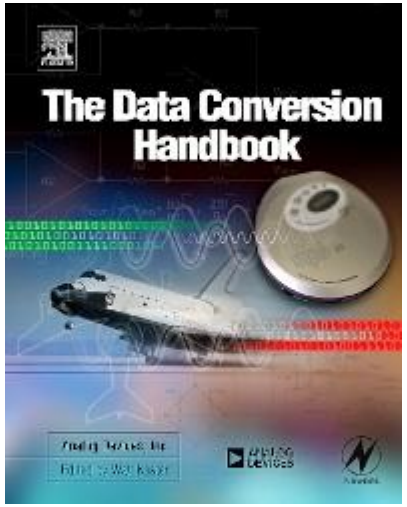
`uint32_t NVIC_GetActive(IRQn_Type)`

Reads the Active status (ACTIVE_FF) returning 0 (not active) or 1 (active)

Example: `uint32_t act = NVIC_ GetActive(SysTick_IRQn);`

7 – ANALOG INTERFACING

- ADC – Analog to Digital Converter
- DAC – Digital to Analog Converter



recommended readings:
1- http://www.analog.com/en/analog-dialogue/articles/the-right-adc-architecture.html
2- The Data Conversion Handbook, Edited by Walt Kester, Analog Devices Inc.

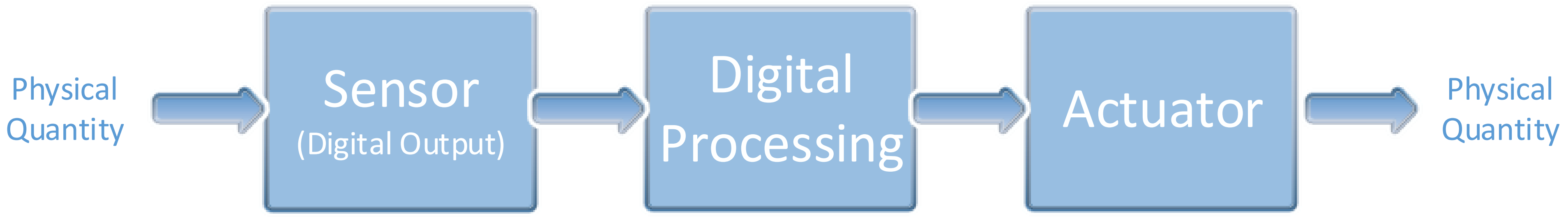
ANALOG VS DIGITAL

- Real world phenomena, such as audio, images, temperature, forces, pressure, and so, can be represented by waveforms that are continuous both in time and amplitude with an infinite resolution.
- Sensors are able to convert these physical quantities into analog electrical signals that can be processed by analog circuits. This was the most common case a few decades ago: radio, audio amplifiers, television, ...
- Nowadays, these physical quantities are converted to a sequence of numbers, i.e. they were digitized, so that they can be processed by a computer (digital processor).

ANALOG VS DIGITAL

Analog	Digital
Physical quantities are converted to electrical signals that are **continuous** both in time and amplitude.	Physical quantities are converted into numeric codes after being **discretized** both in the time (**sampling**) and in amplitude (**quantization**).
Analog signal are prone to **noise** and **distortion** during processing and transmission. Hence, **signal quality decreases** as the signal travels through a system.	Digital signals are much more **robust** to noise and distortion. They can be **restored** to their original value after a noisy system stage.
Analog processing is typically done by a hardwired circuit that performs a **predefined processing** function.	Digital signal processing is typically performed in **software**, thus, its function can be changed **dynamically.**

INTERACTING WITH AN ANALOG WORLD



source: Authors

To interact with the physical quantities in the real world, sensors and actuators are required.

Sensors perceive (“read”) a phenomena and translate it to an electrical signal. The output of a sensor can be an analog signal or a digital signal.

Analog sensors require an Analog-to-Digital conversion before the information is digitally processed.

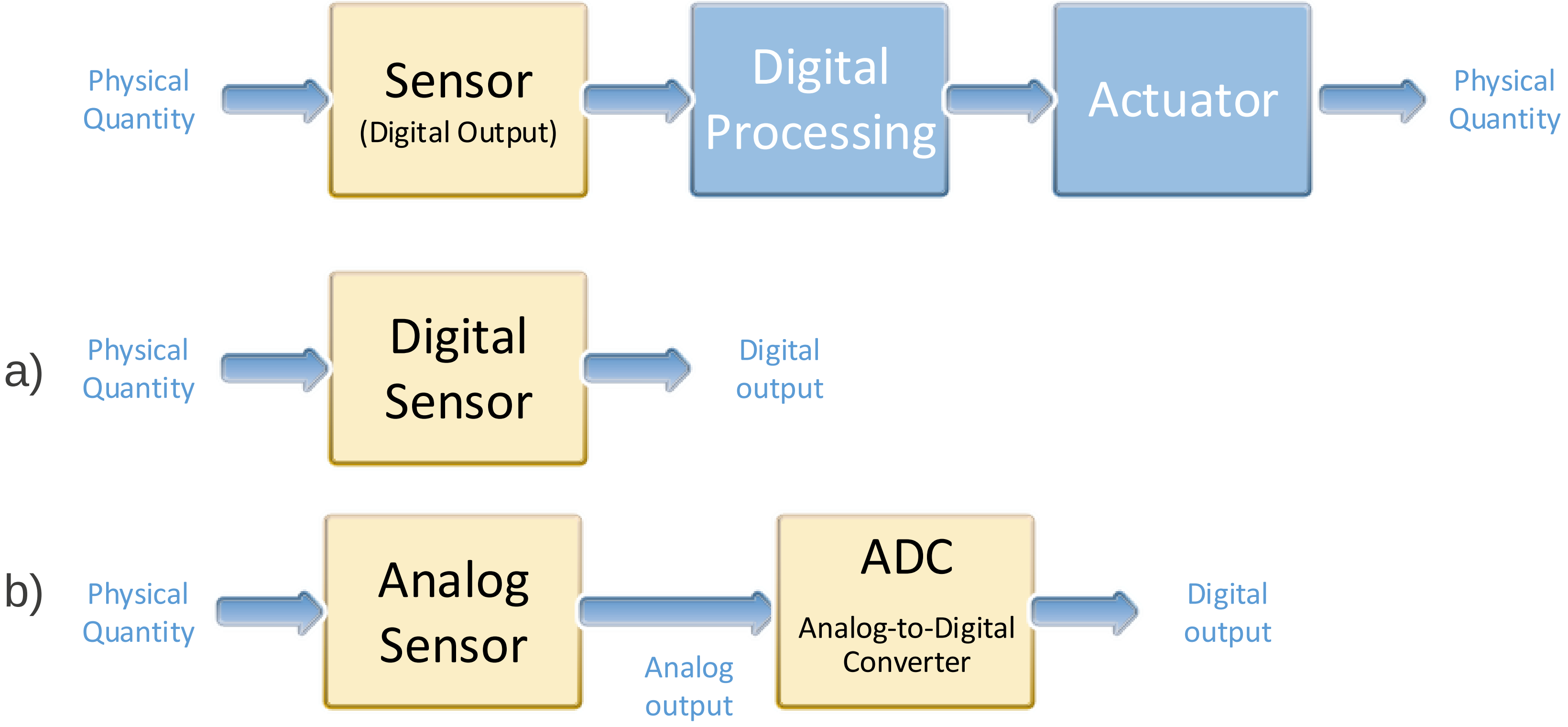
Actuators act upon (“write to”) the environment. The input of an actuator may be an analog or digital signal. To connect a digital processor to an analog actuator a Digital-to-Analog conversion is required.

SENSORS

Two alternatives to the implementation to a sensor in a microprocessed system:

a) Digital Sensor

b) Analog Sensor + ADC



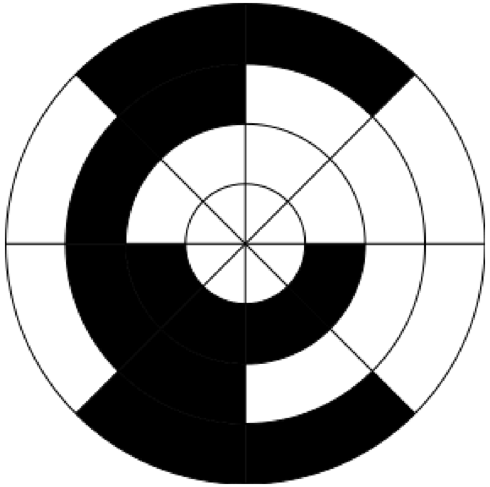
Physical Quantity → Sensor (Digital Output) → Digital Processing → Actuator → Physical Quantity

a) Physical Quantity → Digital Sensor → Digital output

b) Physical Quantity → Analog Sensor → Analog output → ADC Analog-to-Digital Converter → Digital output

source: Authors

EXAMPLE OF A DIGITAL SENSOR - ENCODER



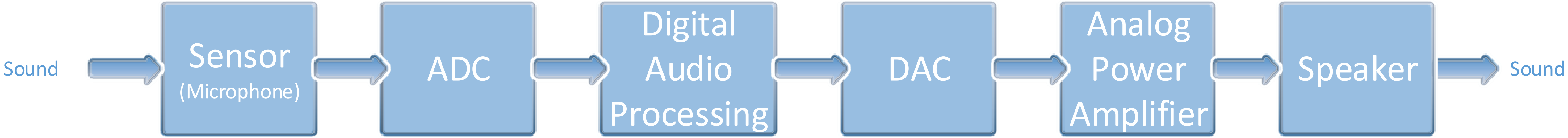
A rotary optical encoder is an angular position sensor. It consists of a disc with transparent and opaque areas that are detected by a photodetector.

The disc show is of an absolute rotary encoder.

Alternatively, an incremental (or relative) rotary encoder generates pulses to represent angular movement of its axis. A set of two pulse stream, shifted by 90 degrees, allows the detection of the direction of movement in an incremental rotary encoder.

source: wikimedia.org (CC)

EXAMPLE OF AN AUDIO PROCESSING SYSTEM



source: Authors

In this example, the microphone is an analog sensor;

the speaker is an analog actuator and its driver is the amplifier.

THE ANALOG-TO-DIGITAL CONVERSION PROCESS

The conversion of an analog signal to digital requires several steps:

1. Low-pass filter to guarantee that the input signal spectrum is limited to a given frequency (f_{signal})
2. Sampling (time discretization) - at periodic intervals take samples of the analog signal. The signal amplitude is still an analog value.
3. Quantization (amplitude discretization) - mapping of the continuous amplitude range into a set of discrete values.

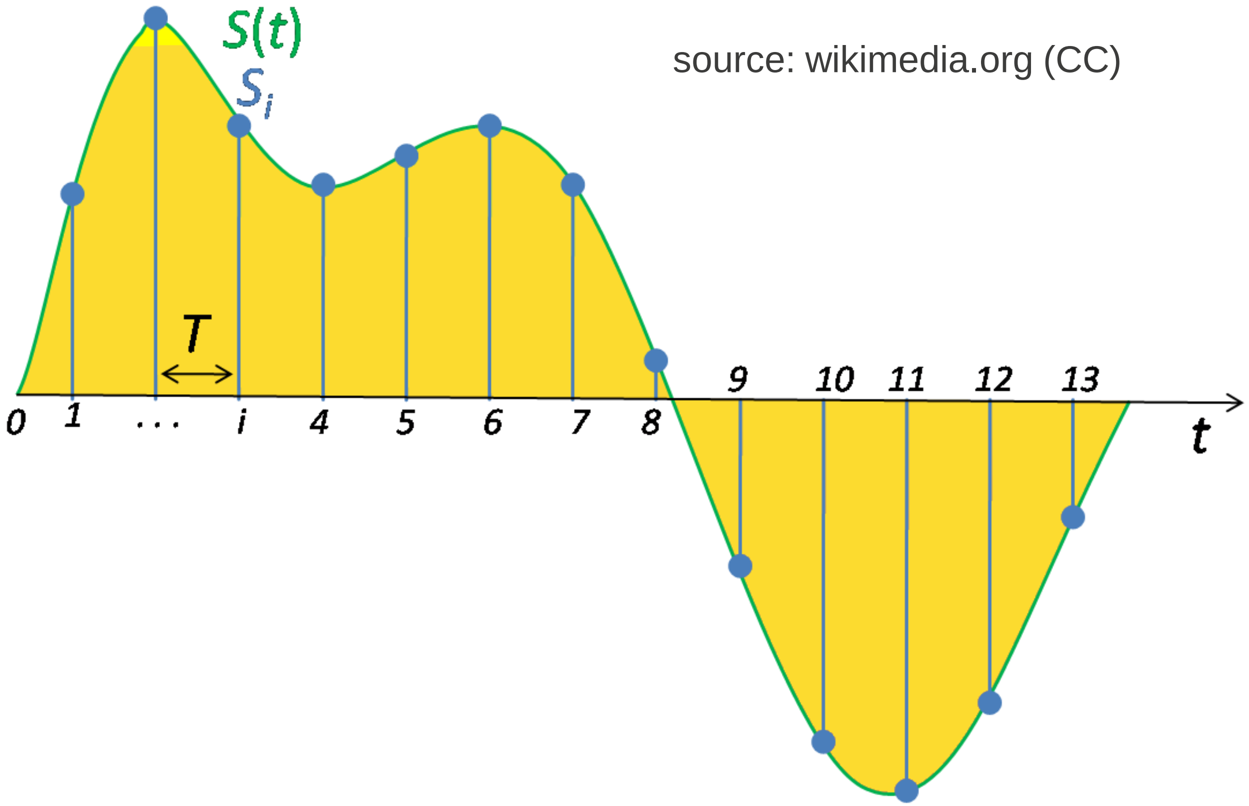
SAMPLING

The green curve represents an analog signal S(t).

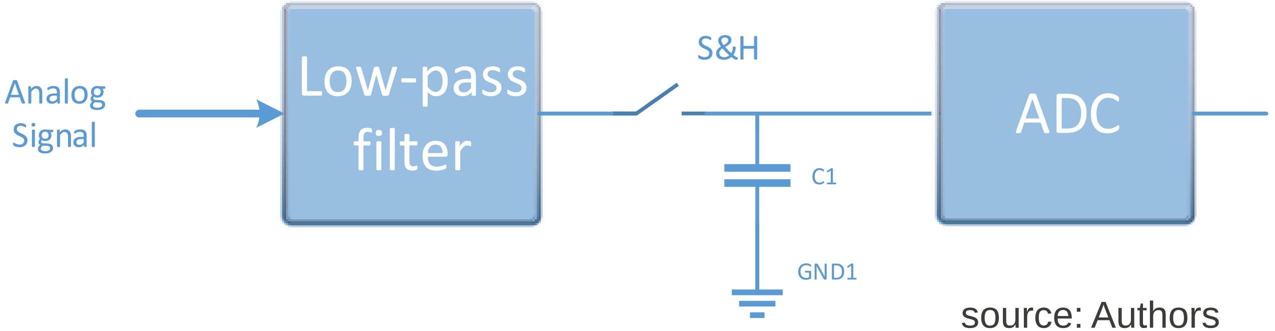
This signal is sampled periodically, T being the sampling period, resulting in a discrete sequence of samples Si (i = 1,2,3,...).

The amplitude of each sample is an analog value.

Sampling is performed by a **Sample-and-Hold** (S&H) circuit, represented by a switch and a capacitor. The switch closes momentarily, the capacitor is charged with the current value of the input signal, the switch opens and the value remains “memorized” by the capacitor.



source: wikimedia.org (CC)

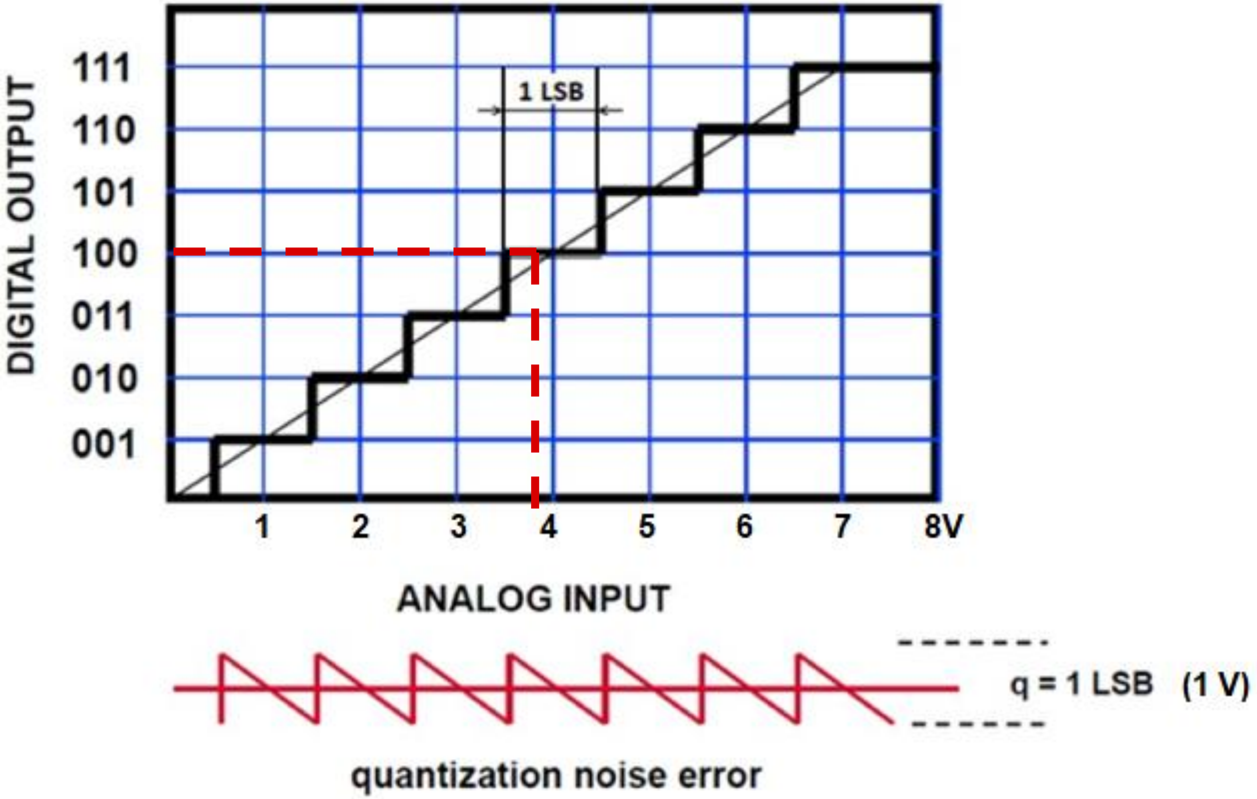


source: Authors

QUANTIZATION

To illustrate de quantization performed by an ADC, consider the transfer function on the right. The horizontal axis is the analog input values to the ADC and the vertical axis are the digital codes produced by the ADC.
In this example the ADC has 3 bits, hence, it is able to represent 8 binary values, from 0 (000) to 7 (111). The input range for this example is from 0 to 8 volts.

The bold line in the graph is the transfer function. If the input voltage is 3.8 volts, the output code will be 4 (100), representing 4 volts. The difference between the actual input value (3.8) and the output value (4) is the quantization error and it is due to the output assuming only a discrete set of values.



source: Renesas DevCon2015
Mitch Ferguson - ADC Specifications

OUTPUT VALUE CALCULATION

a) Unipolar: the quantization levels are distributed from 0 to Vref.

For an ADC with N-bit resolution, there are 2^N quantization levels.

Each quantization level corresponds to an input range q, where

$$q = \frac{Vref}{2^N}$$

Hence, q corresponds to the input range of the LSb (Least Significant bit) of the output code.

The output code (n) of an unipolar ADC, for Vin in the range of 0..(Vref-q) is given by:

$$n = int\left(\frac{Vin}{Vref} * (2^N) + \frac{1}{2}\right)$$

Example: N = 10 bits, Vref = 5V, Vin = 2.5V

q is 4.88 mV and the output code is 512 (10 0000 0000b)

int(x) results in the integer part of x by truncation.
Hence, for x ≥ 0, int(x) = floor(x)
and for x < 0, int(x) = ceiling(x)

OUTPUT VALUE CALCULATION

b) Bipolar: the quantization levels are distributed from V_{-ref} to V_{+ref}.

Where typically, $V_{-ref} = - V_{+ref}$

Each quantization level corresponds to an input range q, where

$$q = \frac{V_{+ref} - V_{-ref}}{2^N}$$

The output code (n) of a bipolar ADC, for Vin in the range of V_{-ref} .. (V_{+ref} -q) is given by:

$$n = int\left(\frac{Vin - V_{-ref}}{V_{+ref} - V_{-ref}} * (2^N) + \frac{1}{2}\right)$$

Example: N = 10 bits, V_{+ref} = 5V, V_{-ref} = -5V, Vin = 0V

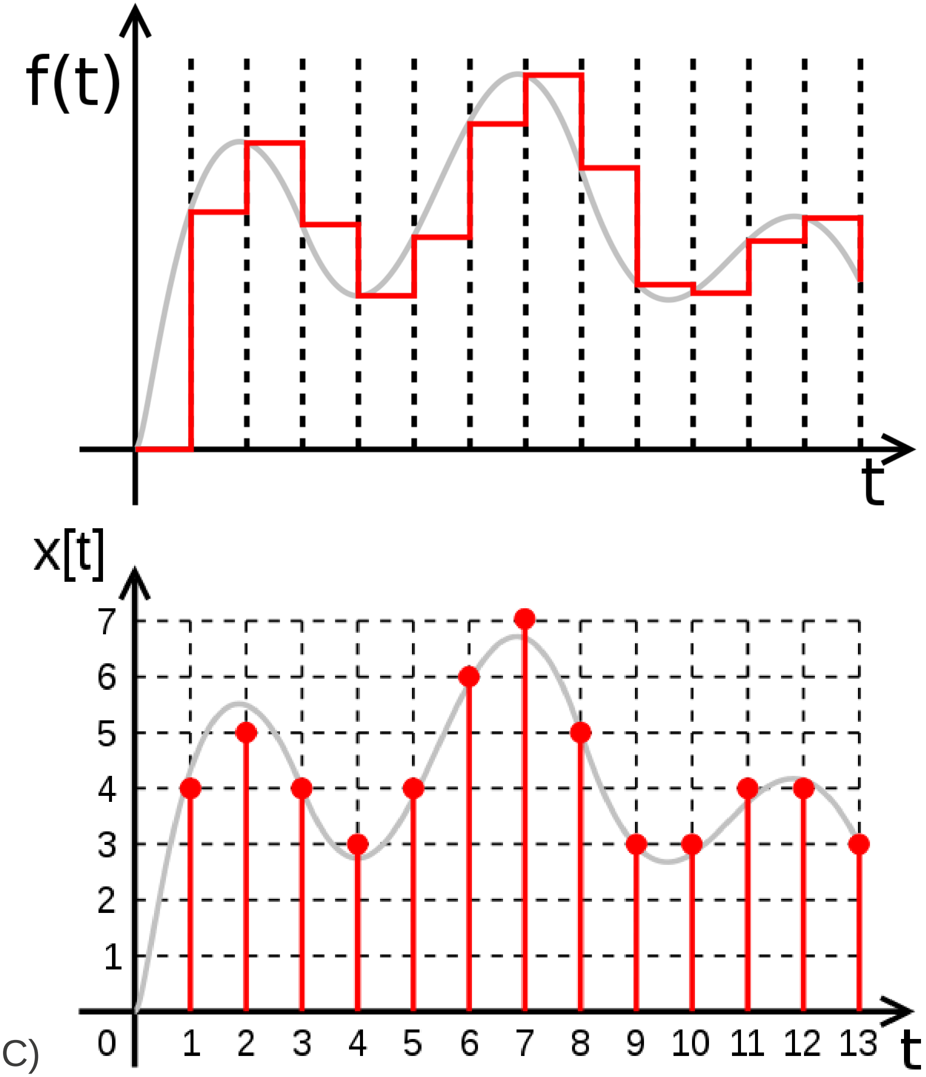
q is 9.76 mV and the output code is 512 (10 0000 0000b)

ANALOG-TO-DIGITAL CONVERSION EXAMPLE

On the upper figure, the grey line represents the input analog signal. The dashed lines indicate the sampling times. The red line is the output of the sample-and-hold. It changes value exactly at the sampling times.

In the lower figure, the red dots represent the output of the ADC. The effect of the quantization error is noticeable as the distance between the red dot and the input signal.

The output of the ADC is the following numeric sequence: 4,5,4,3,4,6,7,5,3,3,4,4,3.



source: wikimedia.org (CC)

SIMPLE ADC (3-BIT FLASH)

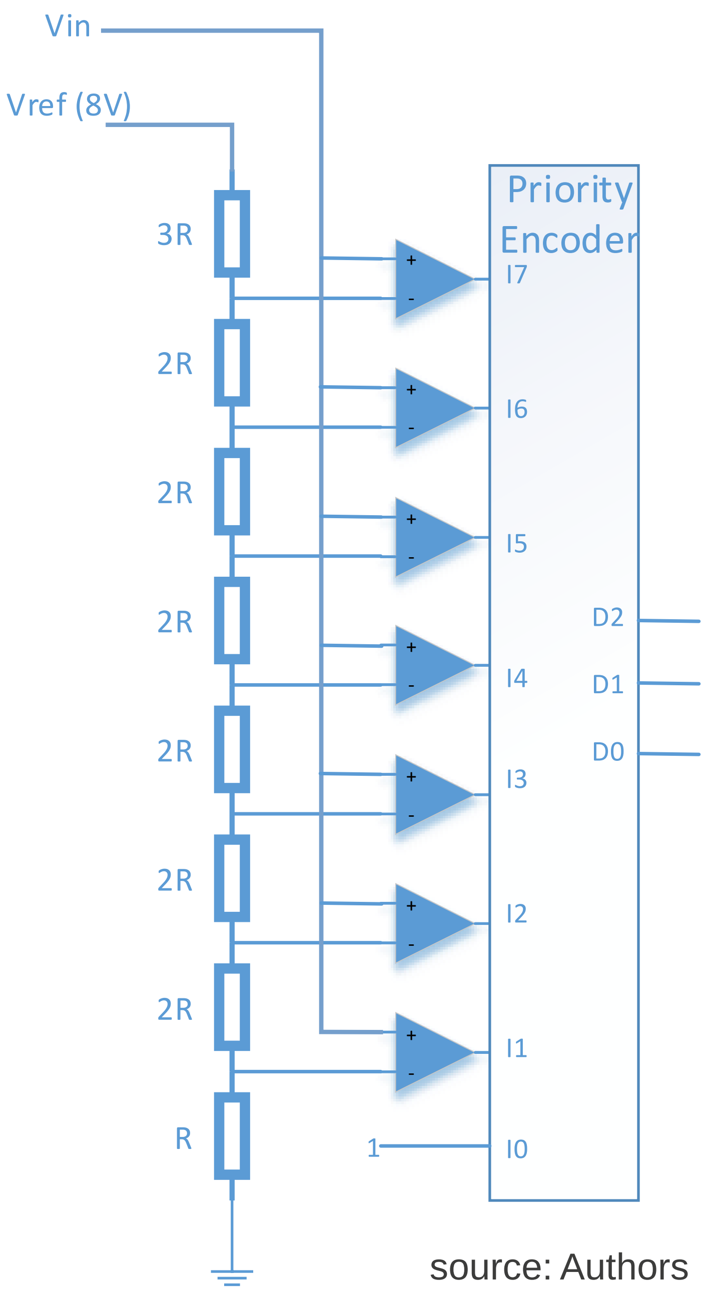
This circuit implements a 3-bit Flash ADC. It is the fastest ADC topology. Vin is in the range 0..Vref and Vref is 8V.

For an ADC with 2N possible output values, 2N -1 comparators are required. Thus, Flash ADCs are usually implemented for a small number of bits.

The resistor ladder provides the appropriate reference voltages for each comparator. In this case: 0.5V, 1.5V, 2.5V, ... 6.5V.

When a comparator detects that the input voltage (Vin) is higher than its reference voltage, its output changes to level 1.

The priority encoder (I7 is highest priority and I0 is lowest) generates the binary code corresponding to the highest priority active input.



source: Authors

ADC CHARACTERISTICS

Resolution: the number of bits (N) of the output code of the ADC. The number of quantization levels is given by 2^N. A 10-bit ADC has 1024 quantization levels. Hence, for a reference voltage of 1V, each quantization level is 0.97mV (1V/1024). The quantization error is up to ± 0.485mV (.97/2).

Conversion time: how long does it take to the ADC to perform a conversion. Currently, most ADCs integrated in MCUs take from 0.1µs to 1µs. Flash ADCs may take less than 10 ns. The conversion time determines the maximum **sampling frequency** ($f_{sampling}$). By the Nyquist theorem, the sampling frequency should be larger than twice the highest frequency in the input signal, i.e. $f_{sampling} > 2 * f_{signal}$.

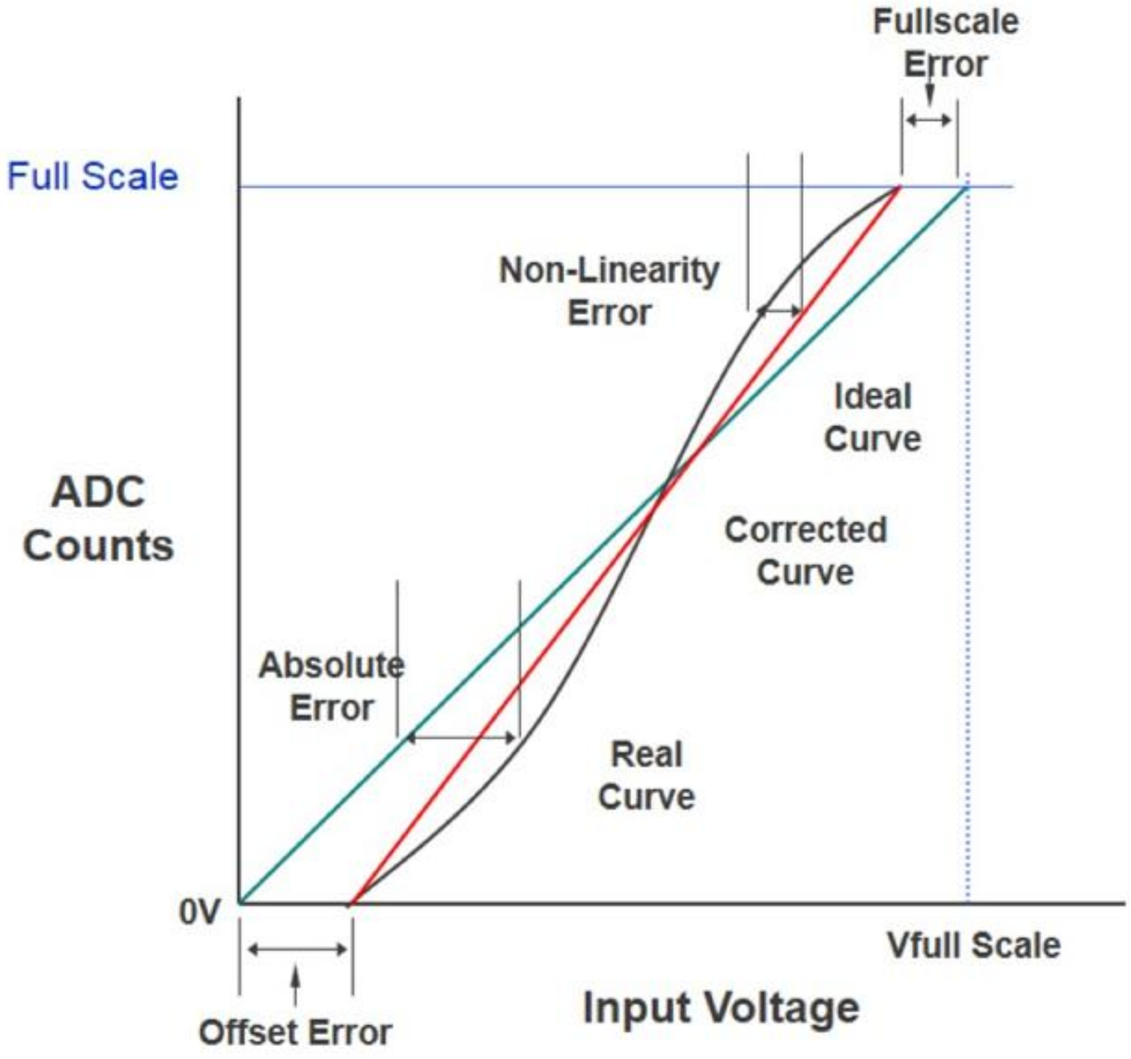
ADC CHARACTERISTICS

DC (and low frequency) errors.

The green line represents the ideal transfer function.

The black line represents the actual transfer function including **offset** errors and **non-linearity** errors.

The red line corrects non-linearities but offset errors are still present.



source: Renesas DevCon2015
Mitch Ferguson - ADC Specifications

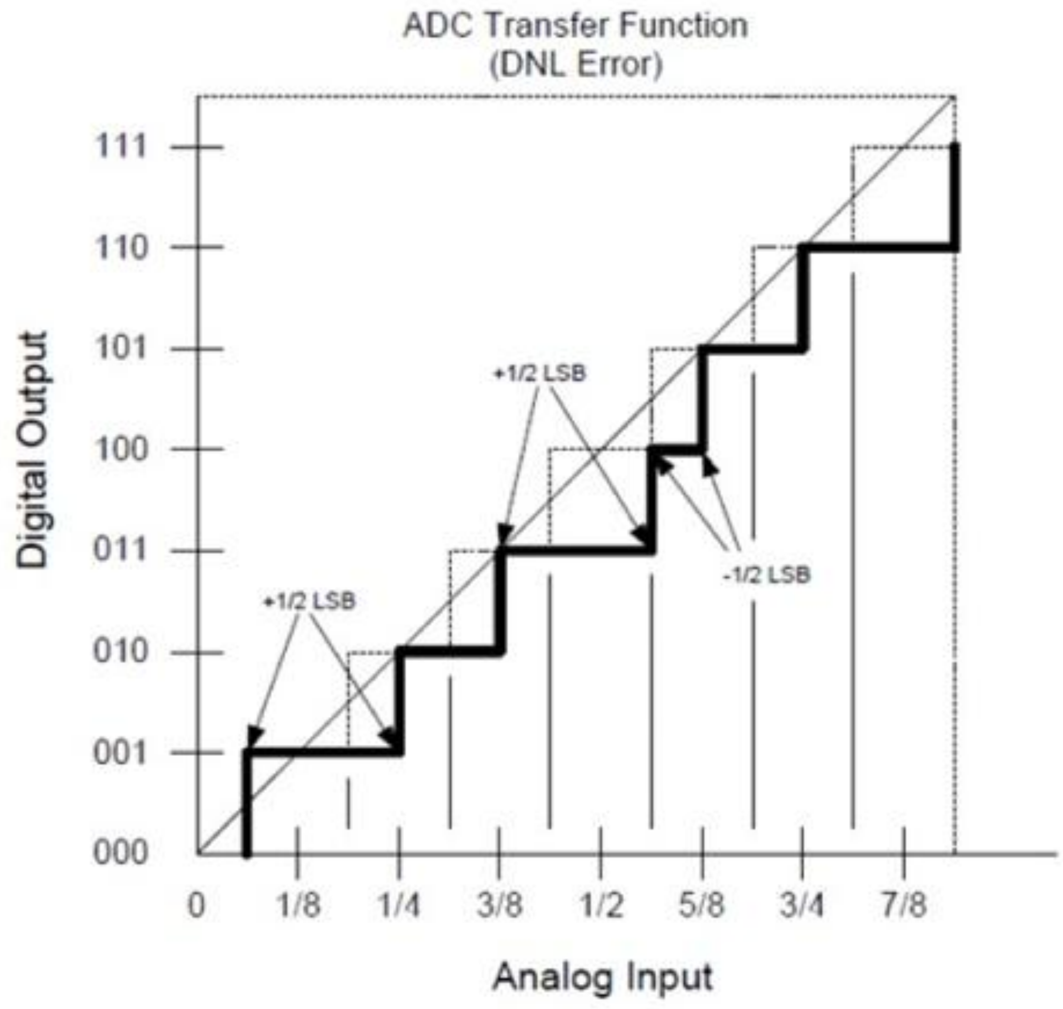
ADC CHARACTERISTICS

Differential non-linearity (DNL)

The ideal transfer function is shown as the dotted line.

Actual transfer function is shown in bold line.

DNL causes wider or narrower code widths. Also, increases quantization noise.



source: Renesas DevCon2015
Mitch Ferguson - ADC Specifications

ADC IMPLEMENTATIONS

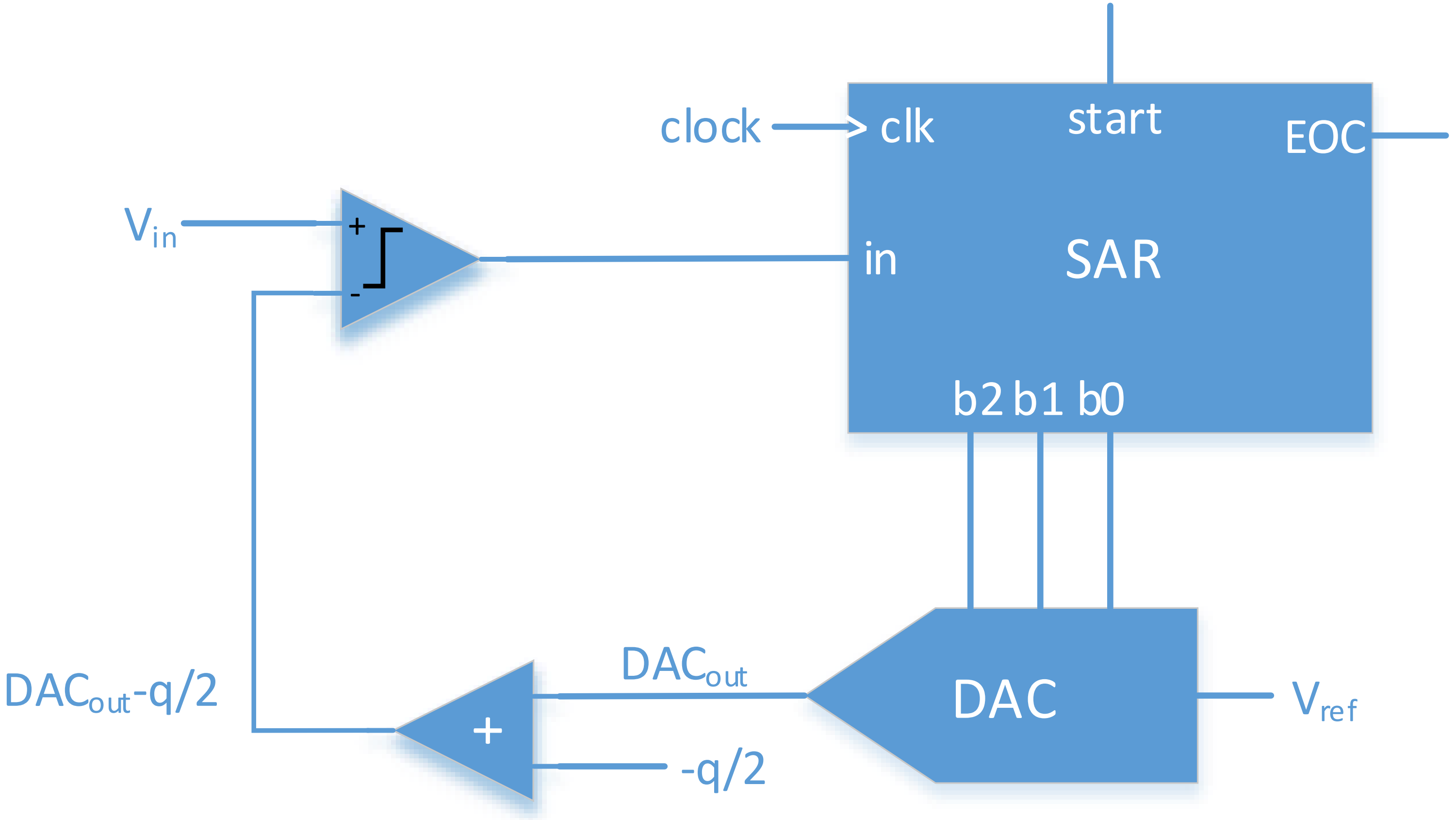
The most common ADC architectures (topologies) are:

Description
It is the fastest but also the one that requires the most circuitry (2^N comparators and resistors).
Two ADCs in sequence, the first resolves the MSb and the second the LSb.
Resolves bit-by-bit, thus, requiring a single comparator. Takes N clock cycles to generate the result.
Based on sigma-delta modulation, it is a 1-bit ADC that tracks the signal. It is based on oversampling, digital filtering and decimation.
Uses a single comparator whose reference voltage is a ramp. Counts the number of clock pulses to the ramp to reach the value of the input signal.
Integrates the input signal then integrates -Vref until the result of the integration reaches 0. Measures the time for the -Vref integration which is proportional to the amplitude of Vin.

SAR ADC

A SAR ADC (Successive Approximation Register Analog-to-Digital Converter) is based on a SAR, a Digital-to-Analog Converter and a Comparator.

The basic operation is to sequentially compare in input value to half of the analog range, decide if the input is in the upper or lower half, store this bit of information and move to the next comparison.



source: Authors

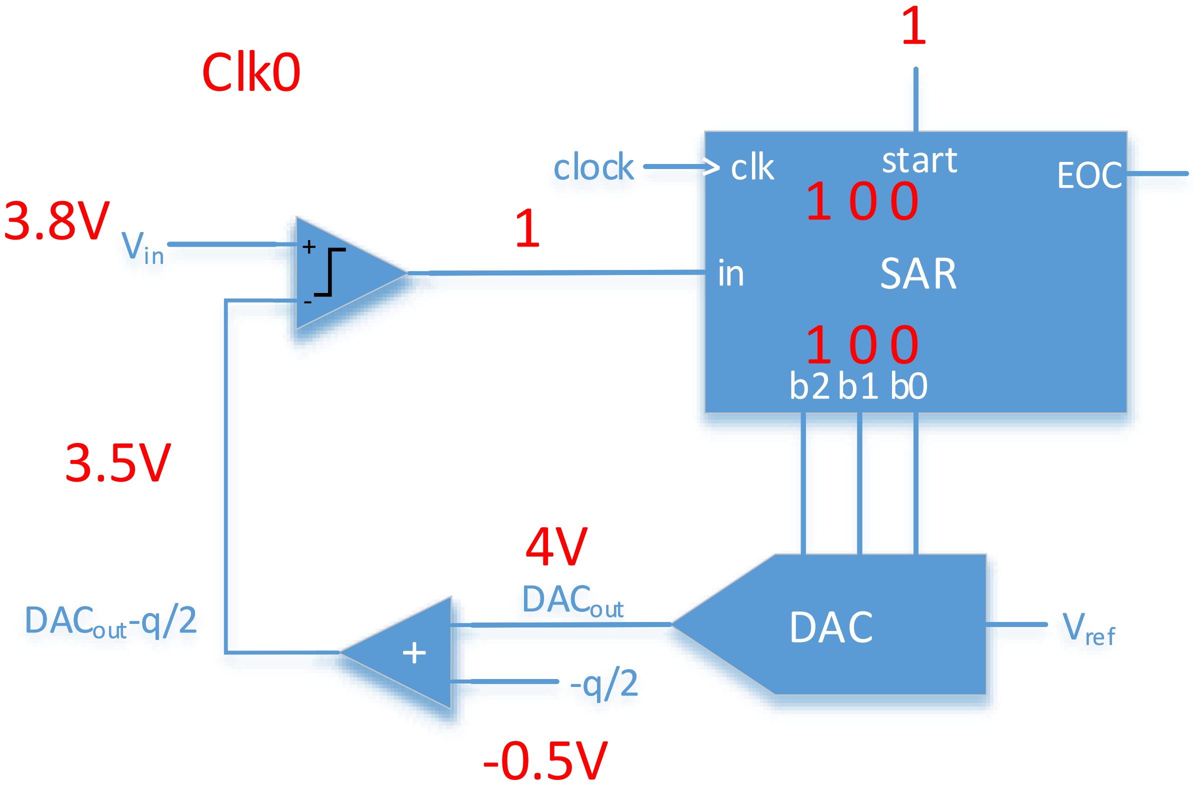
OPERATION OF THE SAR ADC

The same example as in the slide Quantization.
Vin = 3.8V, Vref = 8V, 3-bit ADC.

On the first clock cycle (Clk0):

- The SAR consists of a Shift Register (top) and a register holding the partial results (bottom).
- The start input of SAR is 1, this sets the first bit of the Shift Register in the SAR.
- The bit of the shift-register that is set, is used to compose the current reference value.
- The output 100 of the SAR is converted by the DAC to 4V, producing a 3.5V reference.
- The 3.8V input is compared to the 3.5V reference producing a 1 at the output of the comparator.

This is the MSb of the result that is latched in the SAR at the start of Clk1.



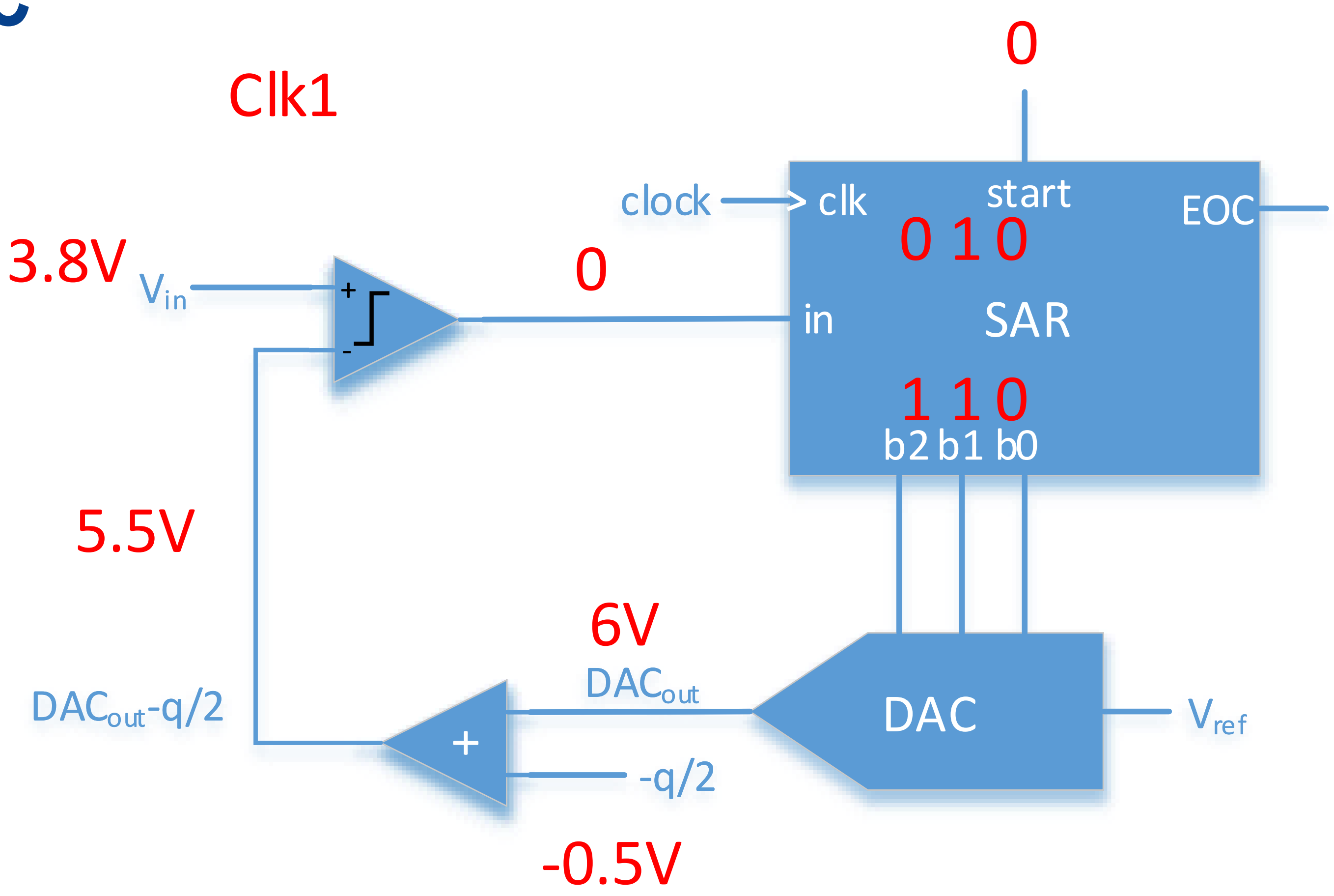
source: Authors

OPERATION OF THE SAR ADC

On the second clock cycle (Clk1):

- The start input of SAR is 0.
- The SARs Shift Register shifted the 1 bit to the next position.
- The partial result register holds a 1 at b2 that was latched at the start of the cycle, while b1 holds a 1 from the middle bit of the shift-register.
- The output 110 of the SAR is converted by the DAC to 6V, producing a 5.5V reference.
- The 3.8V input is compared to the 5.5V reference producing a 0 at the output of the comparator.

This is the next bit of the result that is latched in the SAR at the start of Clk2.



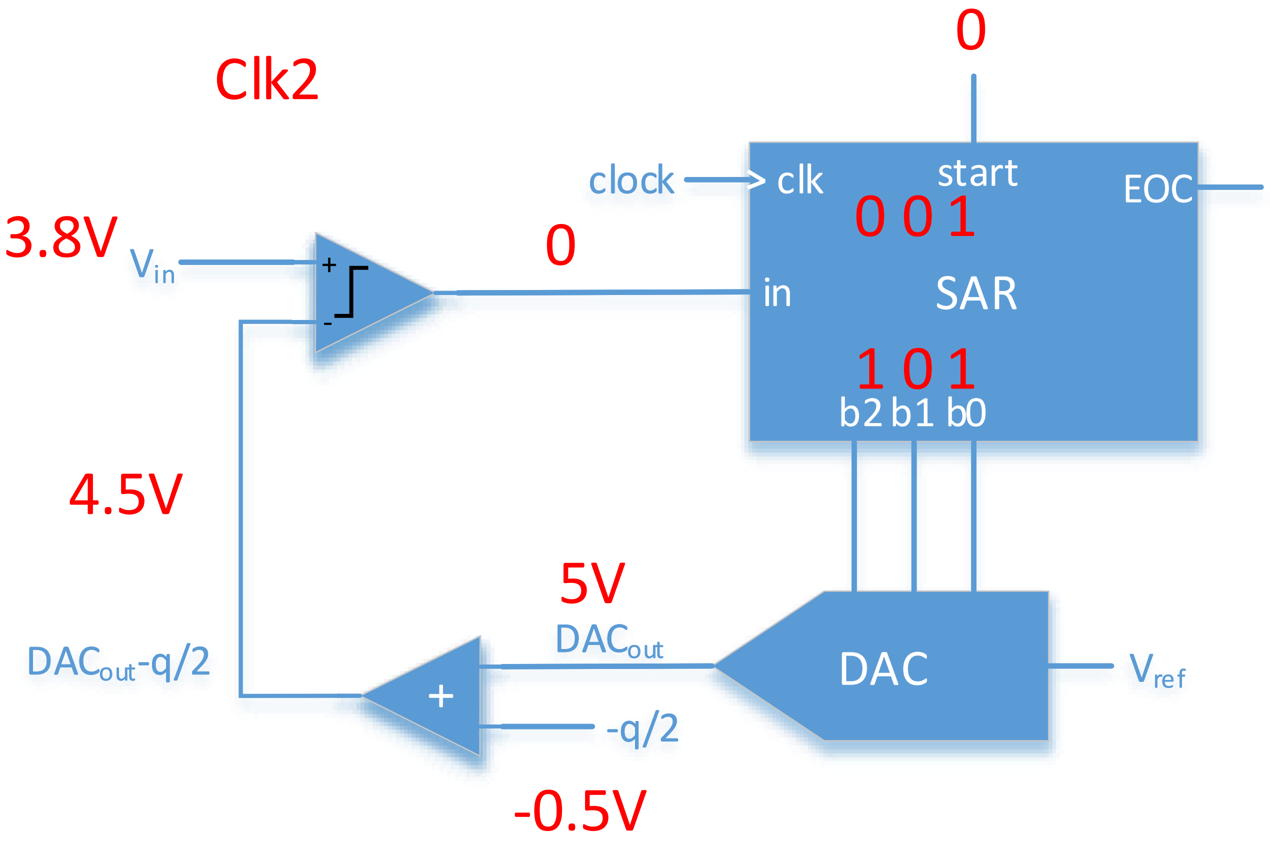
source: Authors

OPERATION OF THE SAR ADC

On the third clock cycle (Clk2):

- The start input of SAR is 0.
- The SARs Shift Register shifted the 1 bit to the next position.
- The partial result register holds a 10 at b2 b1 from the two previous comparisons, while b0 holds a 1 from the last bit of the shift-register.
- The output 101 of the SAR is converted by the DAC to 5V, producing a 4.5V reference.
- The 3.8V input is compared to the 4.5V reference producing a 0 at the output of the comparator.

This is the next bit of the result that is latched in the SAR at the start of the next clock

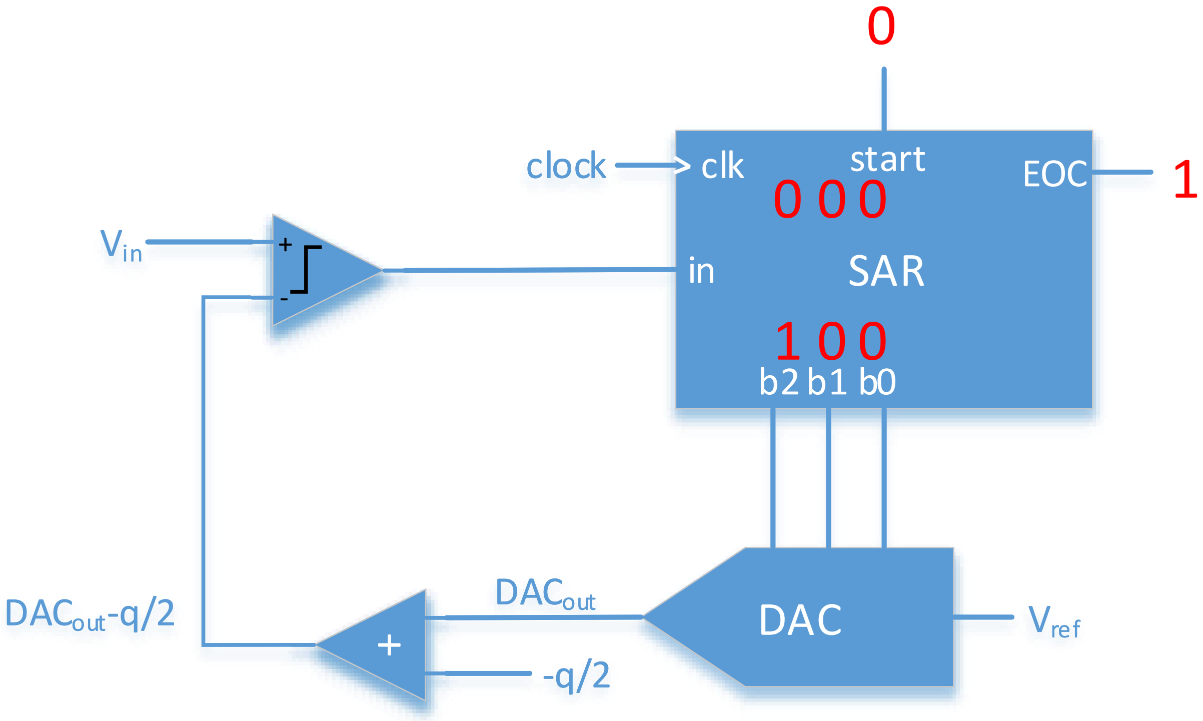


source: Authors

OPERATION OF THE SAR ADC

On the next clock cycle the result is available.

- The last bit of the shift-register is shifted out to EOC (end-of-conversion).
- The result of the conversion is presented at b2 b1 b0 that hold the results of the three previous comparisons.



source: Authors

DIGITAL TO ANALOG CONVERSION

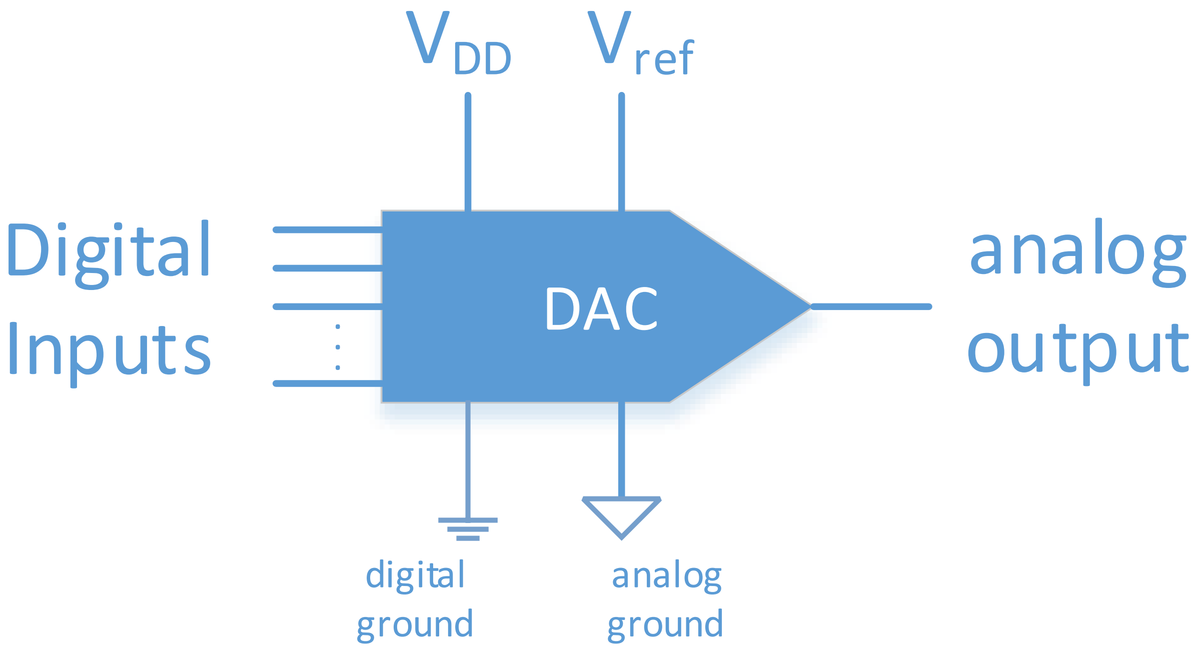
The Digital to Analog Converter (DAC) performs the opposite conversion of the ADC, i.e. it converts a digital value into the corresponding analog value according to the formula:

$$Analog\ output\ (V) = \frac{Digital\ Value}{2^N} V_{ref}$$

for an N-bit DAC whose input value is *Digital Value* and its analog reference voltage is $\mathrm{V_{ref}}$

DIGITAL TO ANALOG CONVERSION

The schematics symbol for a DAC is:

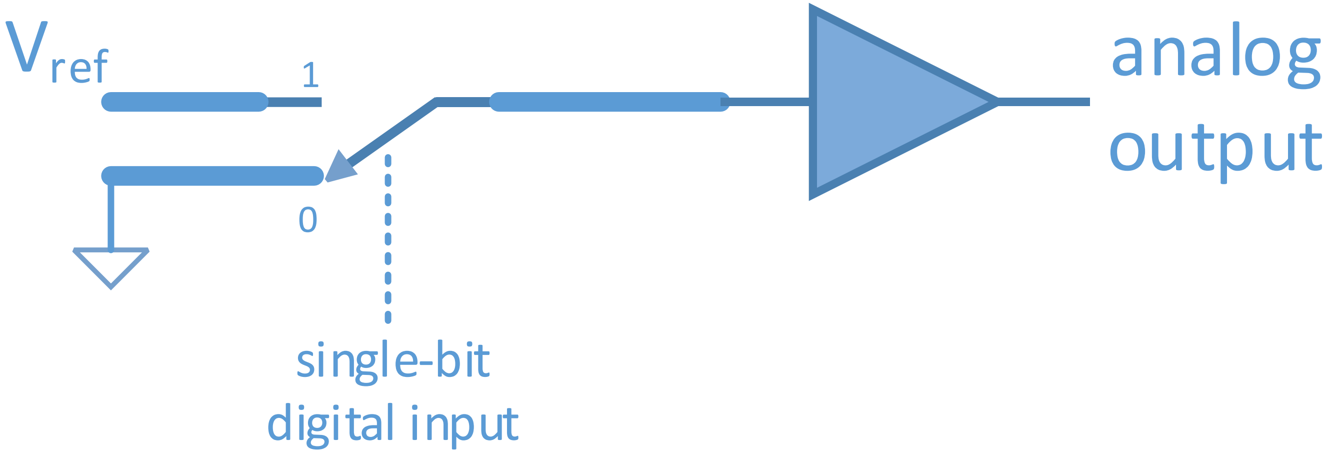


The number of digital input lines is N for an N-bit DAC.

source: Authors

SINGLE-BIT DIGITAL TO ANALOG CONVERTER

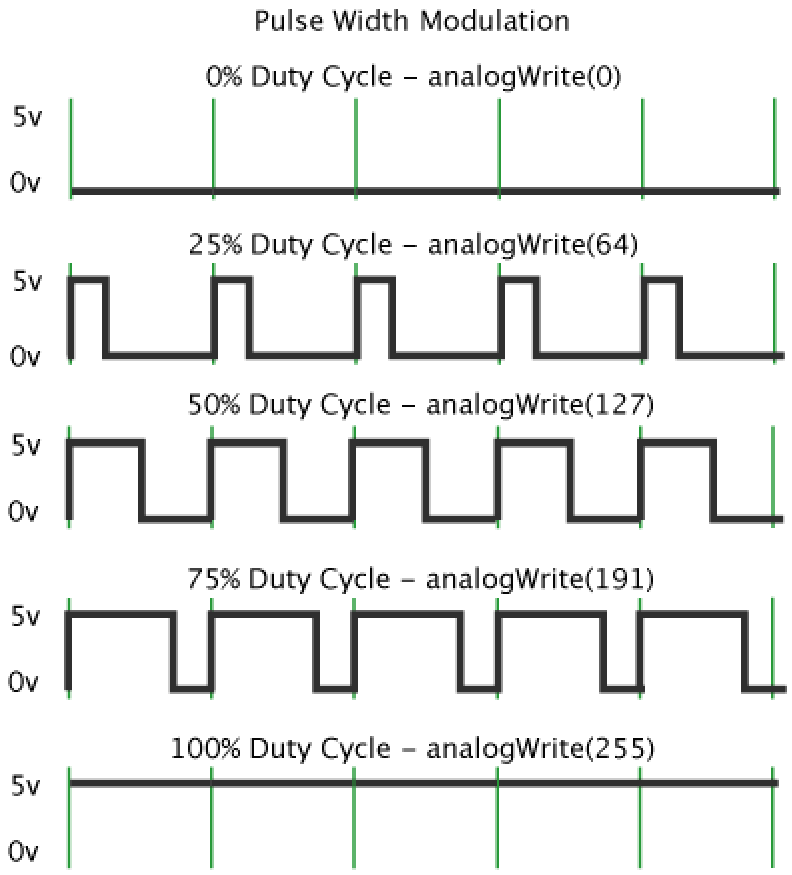
A single-bit DAC is the simplest form of a DAC. A single switch, which is controlled by the digital input, either connects the input of the analog amplifier to Vref or to ground. Hence, the possible analog output values are either 0 or Vref.

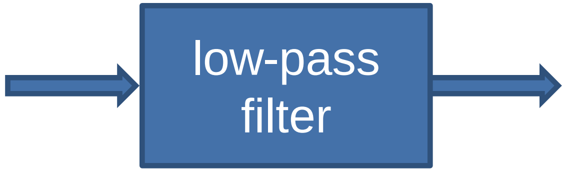


source: Authors

USING A PWM AS A DAC

The effect of 5 different duty cycles after passing a low-pass filter whose cutoff frequency is much lower than the PWM frequency.




low-pass filter

0 V

1.25 V

2.5 V

3.75 V

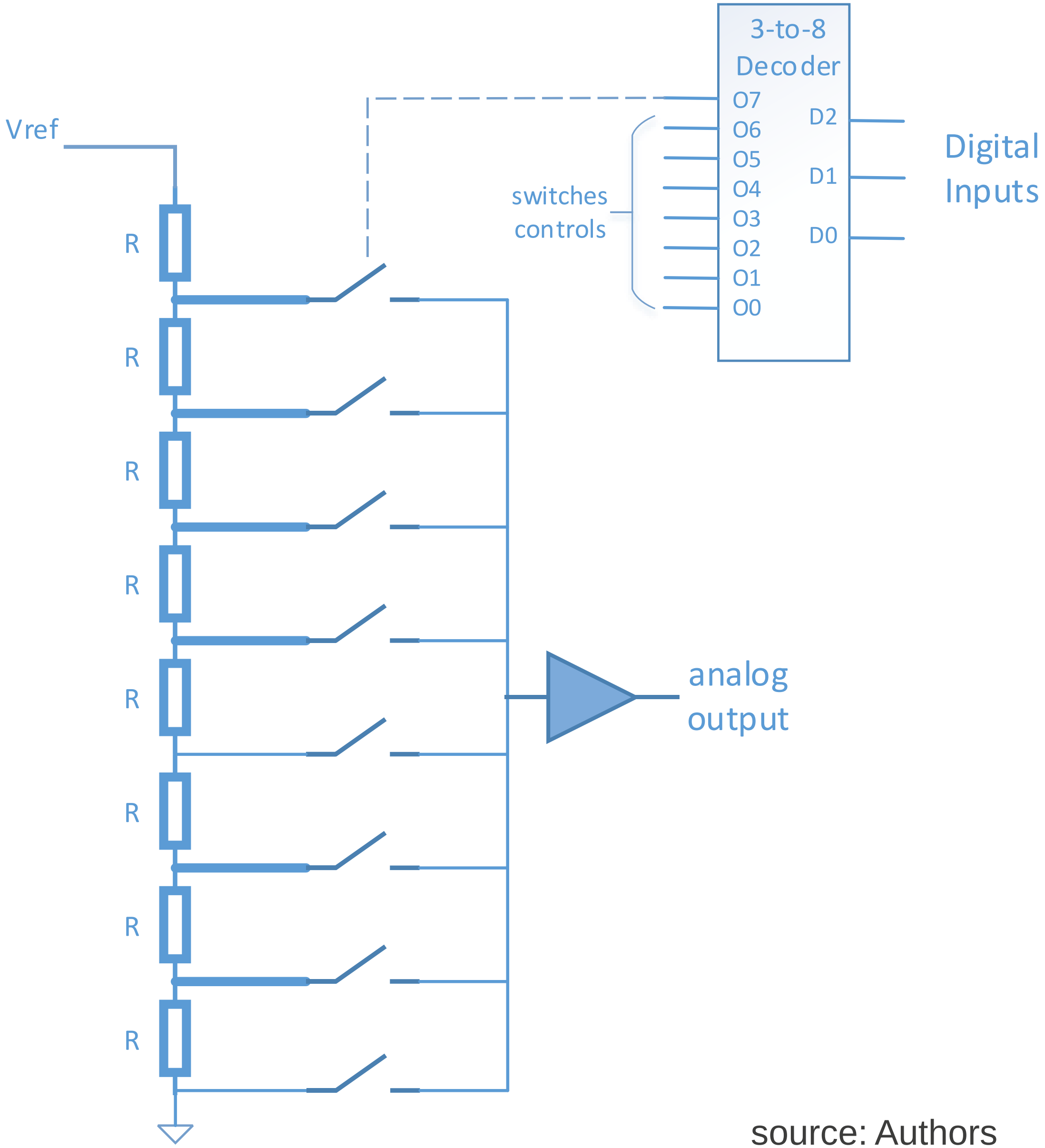
5V

source: commons.wikimedia.org (CC)

USING A PWM AS A DAC

The topology presented is a Kelvin Divider DAC, also called a string DAC. Among its advantages are its monotonicity and low-glitch.

It requires 2^N resistors and switches for an N-bit DAC, which makes it impractical for a larger number of bits.



source: Authors

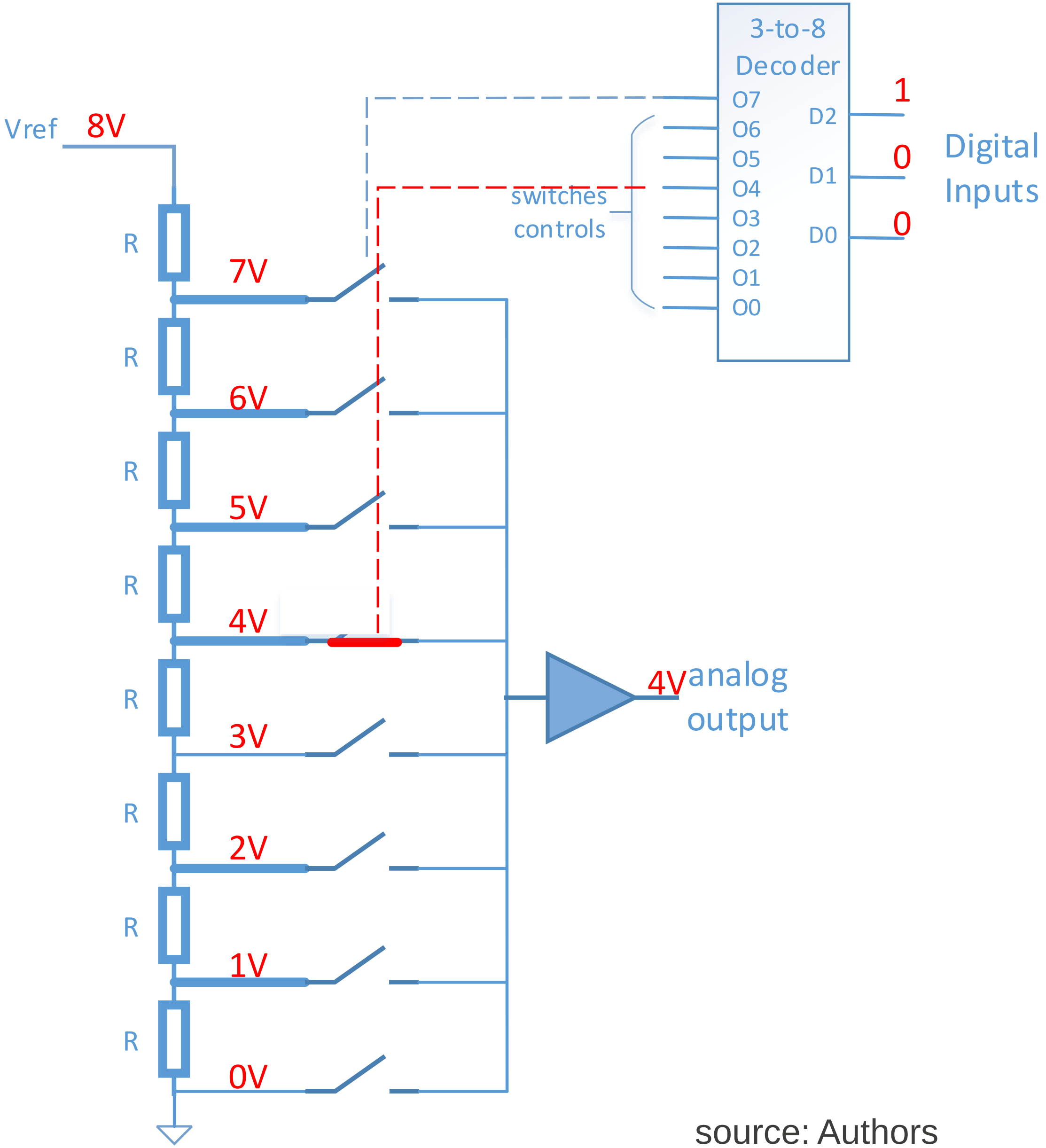
OPERATION OF THE 3-BIT DAC

In this example, Vref is 8V and the digital input has the value 4 (100b).

Output O4 of the decoder is active and commands the corresponding switch. The resistor string has values 1V apart

at each of its taps.

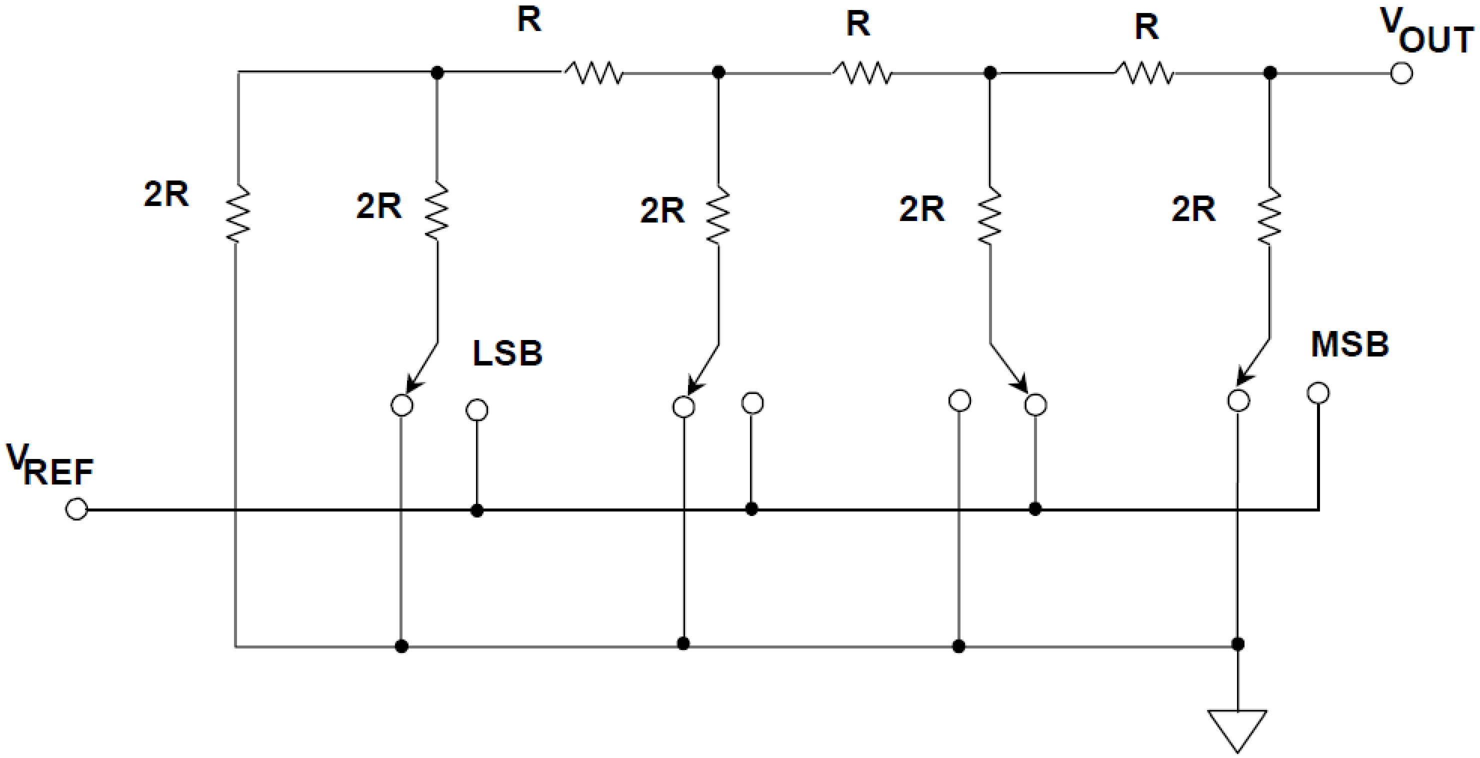
The output analog amplifier simply provides isolation, avoiding that the impedance of the load affects the analog value at the taps of the resistor string.



source: Authors

R2R LADDER DAC

The R2R resistor ladder topology has the advantages of requiring only 2N resistors in the ladder and the values of these resistors are either R or 2R, avoiding components with a significant difference in value, as would be the case for the binary-weighted DAC that requires values of R, 2R, 4R, 8R, 16R, ...



source:
The Data Conversion Handbook, Edited by Walt Kester, Analog Devices Inc.

DAC CASE STUDY THE S7G2 DAC

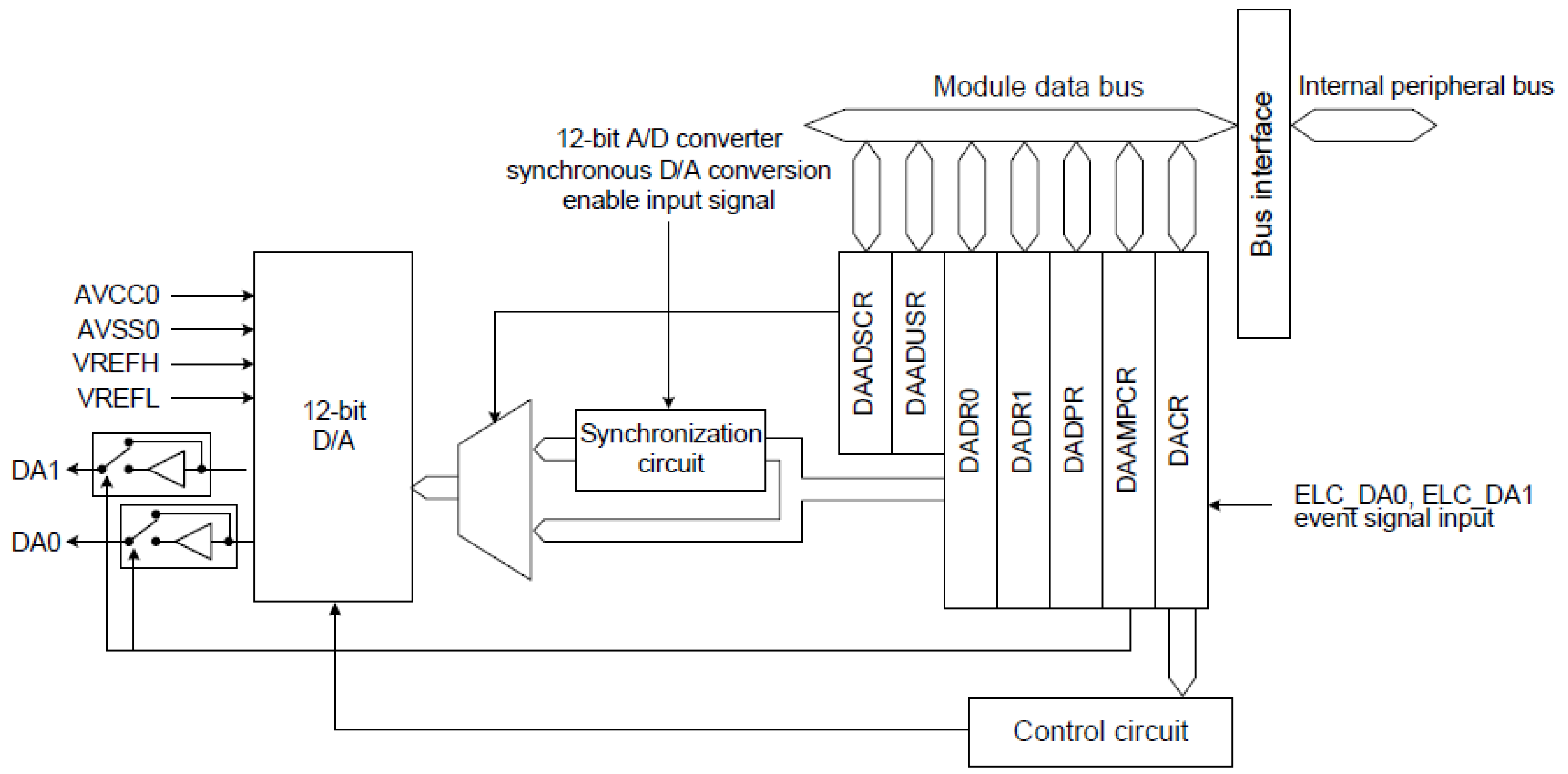
Characteristics:

- Two 12-bit DACs available: DAC0 and DAC1,
- The 12-bit value present at the DADRi register is converted to an analog value of $V\_(ref\ )\ DADRi/4096$,
- Output amplifier is available, may be enabled under SW control,
- DAC operation may be synchronized to ADC1,
- DAC conversion may be started by an ELC event.

DAC CASE STUDY THE S7G2 DAC

DAC0 and DAC1

block diagram



DADR0: D/A data register 0
DADR1: D/A data register 1
DACR: D/A control register
DADPR: DADRm format select register

DAADSCR: D/A A/D synchronous start control register
DAAMPCR: D/A output amplifier control register
DAADUSR: D/A A/D synchronous unit select register

source: Renesas S7G2 user’s manual

DAC CASE STUDY THE S7G2 DAC

Renesas S7G2 DAC characteristics:

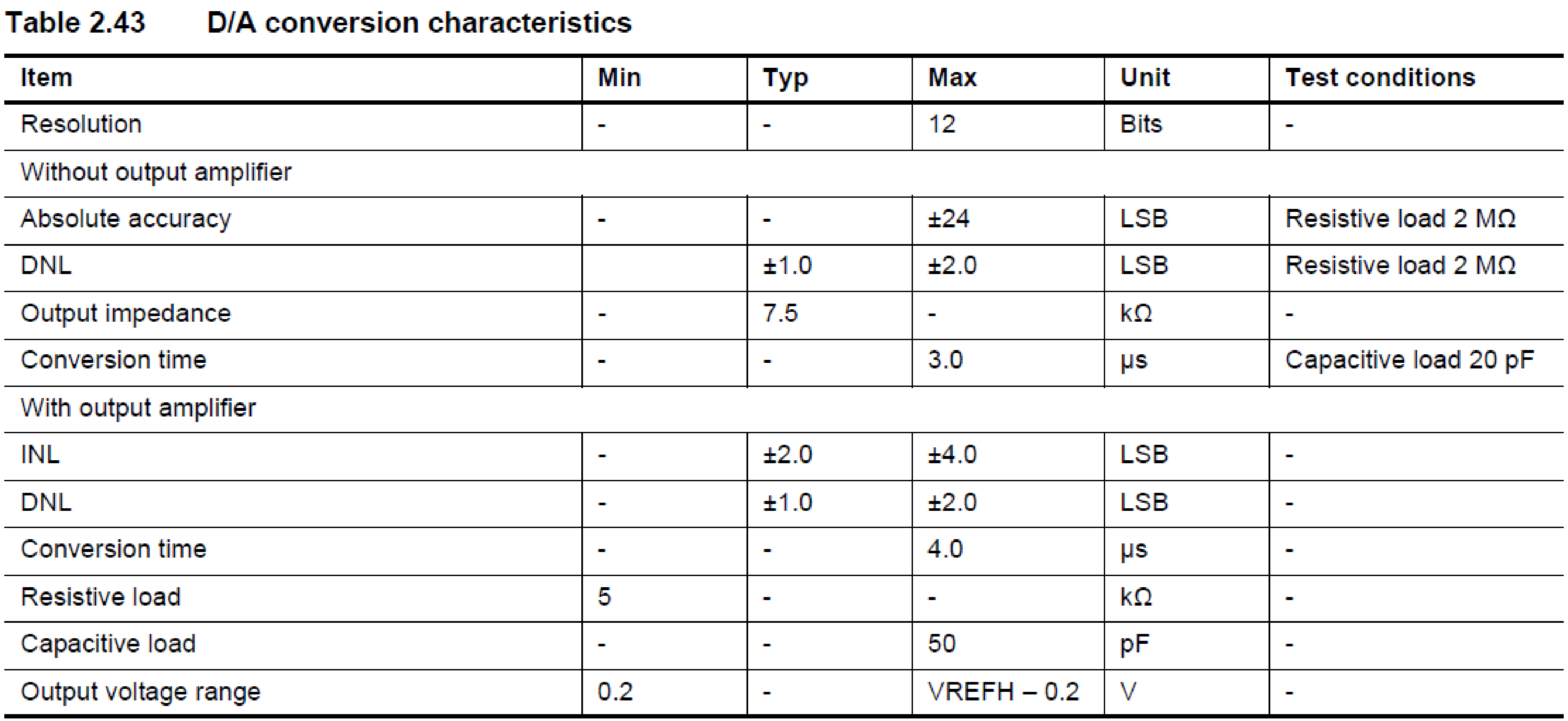


Table 2.43 D/A conversion characteristics

Item	Min	Typ	Max	Unit	Test conditions
Resolution	-	-	12	Bits	-
Without output amplifier					
Absolute accuracy	-	-	±24	LSB	Resistive load 2 MΩ
DNL		±1.0	±2.0	LSB	Resistive load 2 MΩ
Output impedance	-	7.5	-	kΩ	-
Conversion time	-	-	3.0	μs	Capacitive load 20 pF
With output amplifier					
INL	-	±2.0	±4.0	LSB	-
DNL	-	±1.0	±2.0	LSB	-
Conversion time	-	-	4.0	μs	-
Resistive load	5	-	-	kΩ	-
Capacitive load	-	-	50	pF	-
Output voltage range	0.2	-	VREFH – 0.2	V	-

source: Renesas S7G2 datasheet

8 – SERIAL COMMUNICATIONS

- Serial Communications - Introduction
- UART
  - Concepts, Block Diagram, Registers
- SPI
  - Concepts, Block Diagram, Registers
- I2C
  - Concepts, Block Diagram, Registers

8.1 – INTRODUCTION TO SERIAL COMMUNICATIONS

Concept:

- In serial communications ONE bit is transmitted at a time, from the transmitter device (TX) to the receiver device (RX). As opposed to parallel communications where several bits, e.g. 8 bits or 1 byte, are transmitted concurrently.
- In serial communications a reduced number of wires are required.
- Long distance wired communications typically use serial communication.

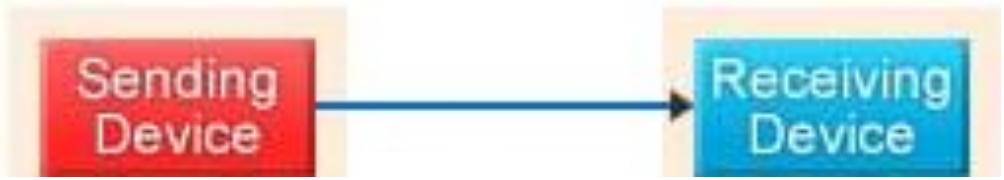
EXAMPLES OF SERIAL COMMUNICATIONS STANDARDS

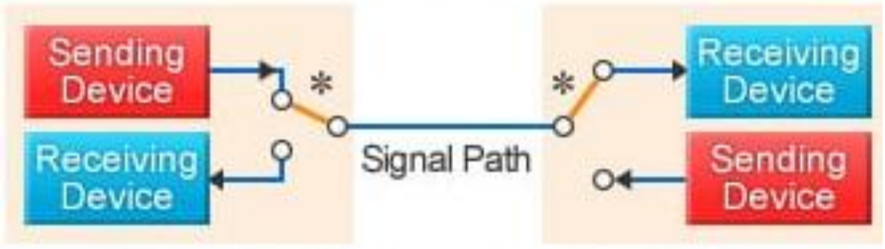
- UART: Universal Asynchronous Receiver Transmitter
- SPI: Serial Peripheral Interface
- I2C: Inter-Integrated Circuit
- USB: Universal Serial Bus
- Ethernet

DIRECTION OF COMMUNICATION

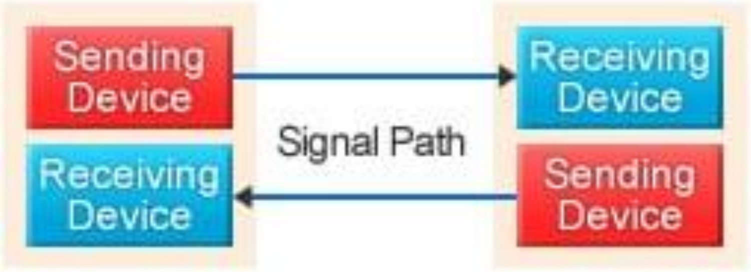
- **Simplex:** the communication occurs in a single direction – one device transmits and the other receives.
- **Half-Duplex:** the communication occurs in both directions but not simultaneously. Both communicating devices (DevA and DevB) have transmitters and receivers. At a given time, either DevA transmits and DevB receives or vice-versa. A single wire is needed to carry the communication.
- **Full-Duplex:** the communications occurs in both directions and can be simultaneous. Usually two wires are used: one to transmit from DevA to DevB and another to transmit from DevB to DevA.

  rem: a transceiver consists of a transmitter and a receiver.





*Use switch to change transfer direction.



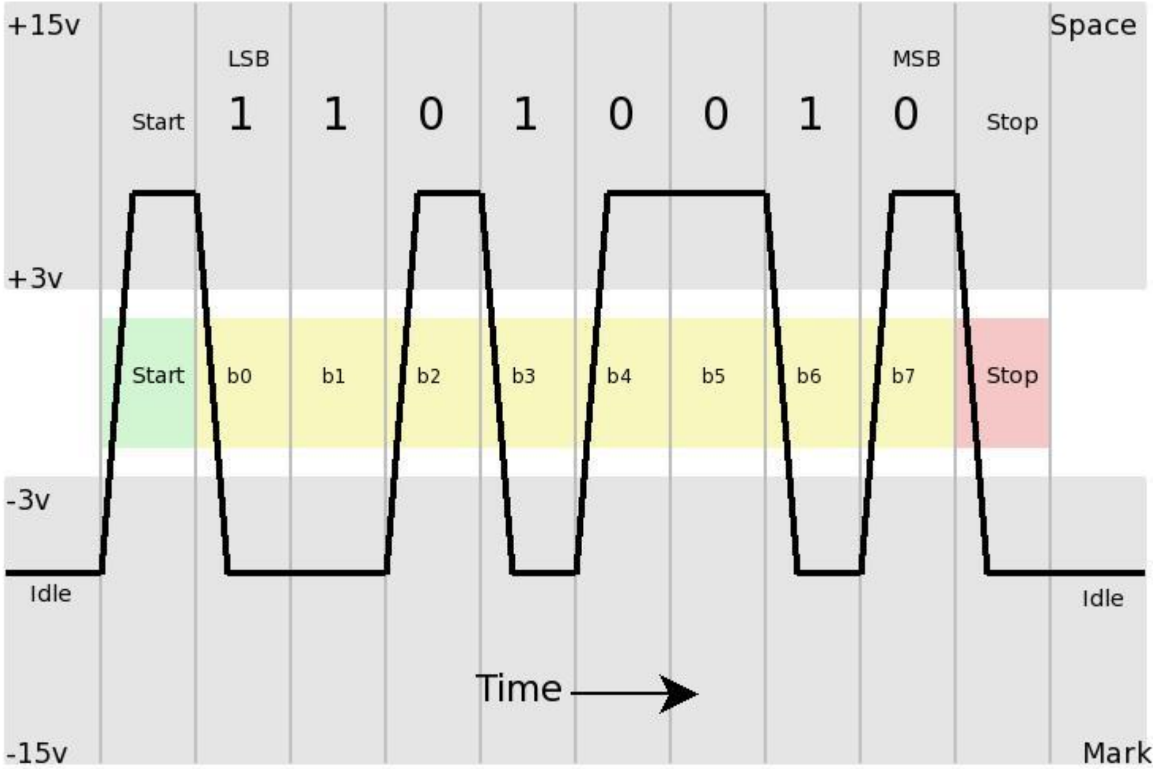
source: Renesas

SYNCHRONOUS VS ASYNCHRONOUS COMMUNICATIONS

- **Synchronous:** there is clock signal that identifies the time instances when a data bit is valid. Thus, the receiver uses the clock signal to recover the transmitted information. SPI and I2C are examples of synchronous communication protocols.
- **Asynchronous:** there is no common clock signal, thus, the two communicating devices must previously agree on a mechanism to identify each bit in the data stream. Therefore, the synchronization information must be embedded in the data signal. Quite often there are transitions in the data signal to identify the time slot of each bit in the data stream. RS-232 is an example of asynchronous communication.

BIT RATE VS BAUD RATE

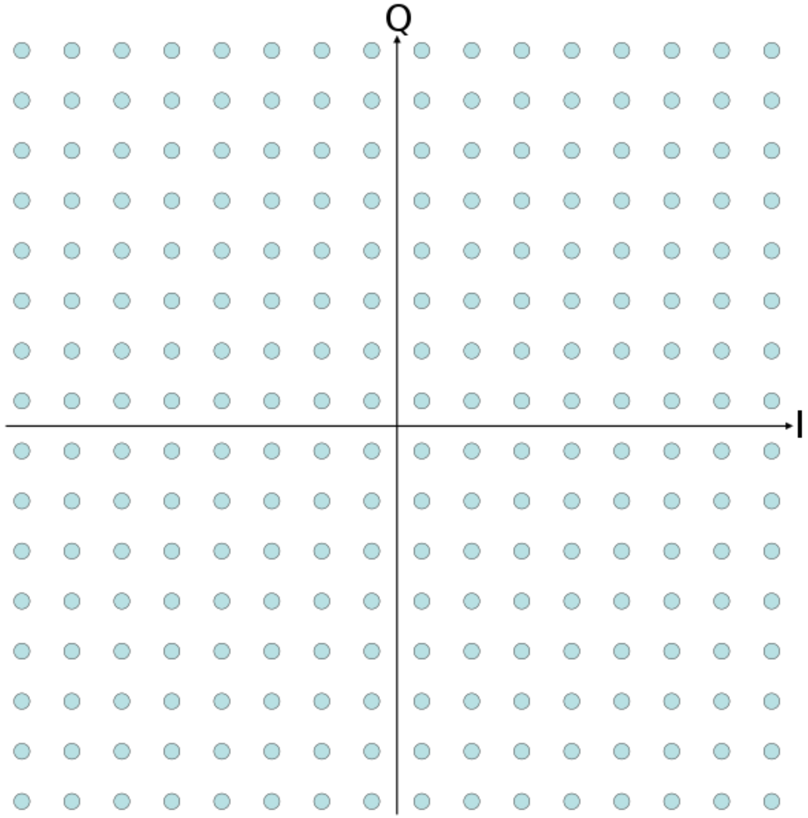
- **Symbols:** when transmitting signals over a wire, each possible combination of amplitude, phase and frequency is called a symbol. Simple schemes use only two symbols: for instance 0V and 3.3V to represent 0 and 1, or the coding used by RS-232 where amplitudes from -3 to -15 represent a 1 while +3 to +15 represent a 0.



source: commons.wikimedia.org (CC)

BIT RATE VS BAUD RATE

- **Symbols(cont):** a much larger set of symbols is also possible, for instance, 256QAM, one of the many encodings used for high speed ethernet, has 256 possible symbols, hence, each symbol encodes 8 bits.
- **Bit rate:** is the number of bits that are transmitted per time unit. Expressed in bits/s.
- **Baud rate:** is the number of symbols that are transmitted per time unit. Expressed in baud/s.
- If a symbol encodes a single bit, such as the case for RS-232, then the baud rate and the bit rate are the same.
  Yet, for 256QAM, the bit rate is 8 times higher than the baud rate.



source: commons.wikimedia.org (CC)

8.2 – UART

A UART is a peripheral present in most MCUs. Typically it receives data from the processor over the bus, hence, in parallel format, and handles the serialization and frame formatting.

A UART (Universal Asynchronous Receiver Transmitter) is capable of transmitting asynchronous data frames to another device as well as receiving. Both devices must be configured for the same speed and frame format.

Typical speeds used for asynchronous serial communication are: 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 bits per second. Higher speeds may also be used as long as agreed among transmitter and receiver.

Typically, the bit encoding defined by the RS-232 standard is used.

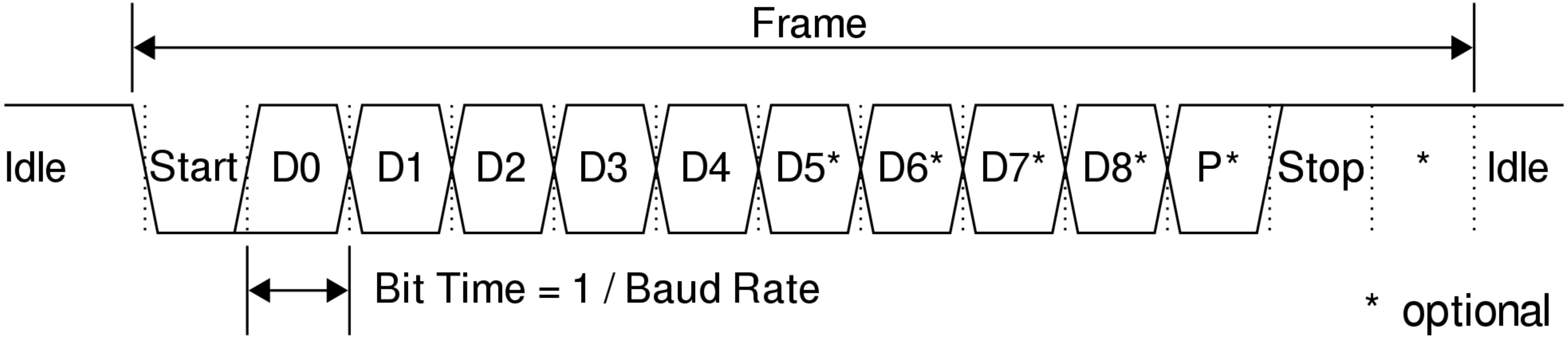
RS-232 defines separate lines for transmission and reception, thus, it is **full duplex** communication.

UART

Asynchronous frame format – consists of a start bit, data bits, an optional parity bit and stop bits.

The frame format is configurable:

- number of data bits: 5 to 8,
- number of stop bits: 1, 1.5 or 2,
- parity: none, odd, even.



source: commons.wikimedia.org (CC)

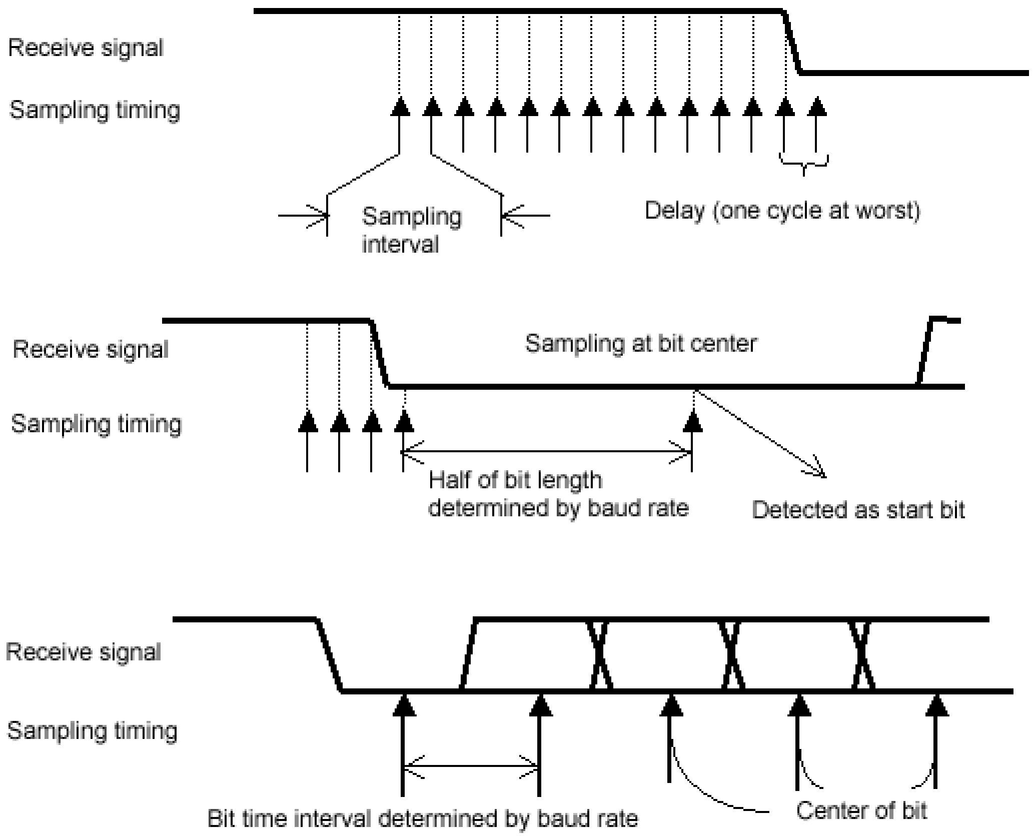
UART

How does the receiver synchronize with the transmitter?

1. Both the transmitter and the receiver are configured in the same way: speed and frame format.
2. Maximum clock skew allowed between the two sides is typically lower than 2%.
3. Receiver samples at a higher rate (e.g. 16 times de baud rate) for the start-bit transition. A delay of half-bit period determines the mid-bit position. From then on, sample every one-bit time.

UART

How does the receiver synchronize with the transmitter?



source: Renesas

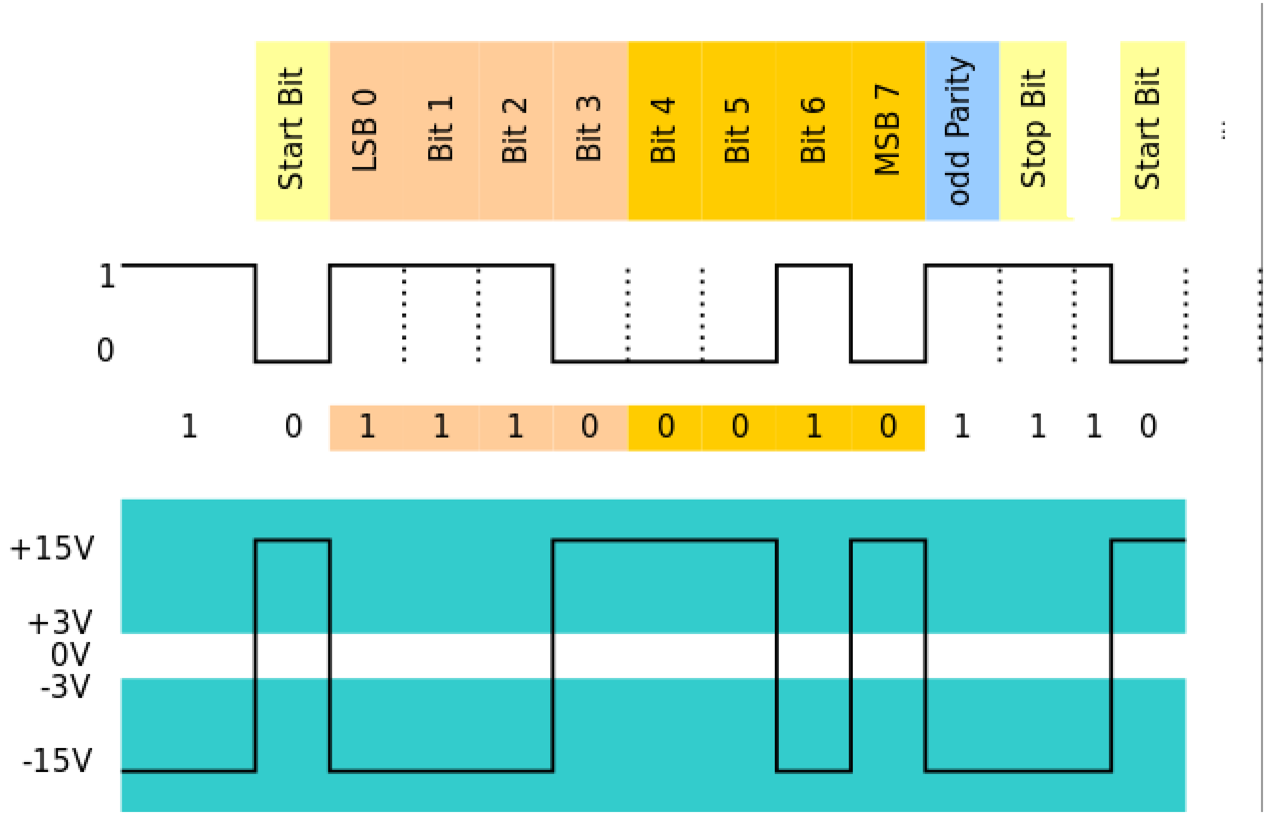
UART

Example of the transmission of the character G (ASCII code 0x47) over an asynchronous line. Note that the LSb is transmitted first.

Upper figure shows UART levels while lower figure shows RS-232 levels.

Configuration: 8O1.5 (eight bits, odd parity, 1.5 stop bits).

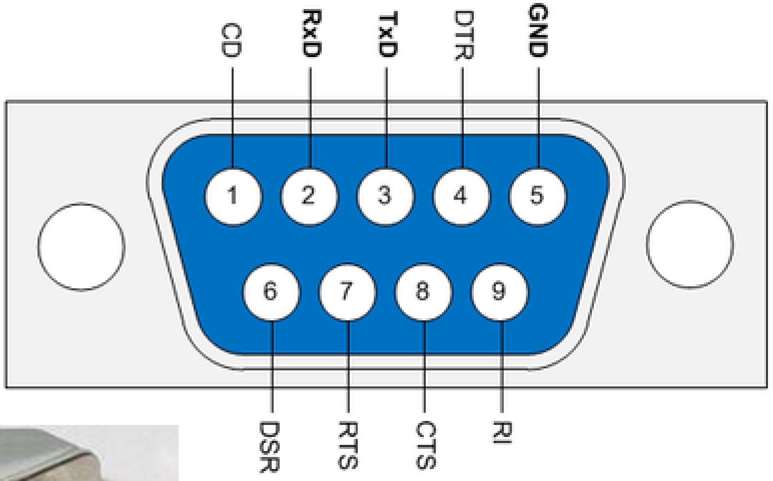
For a 9600 bps, the duration of each bit is 104.16 us.

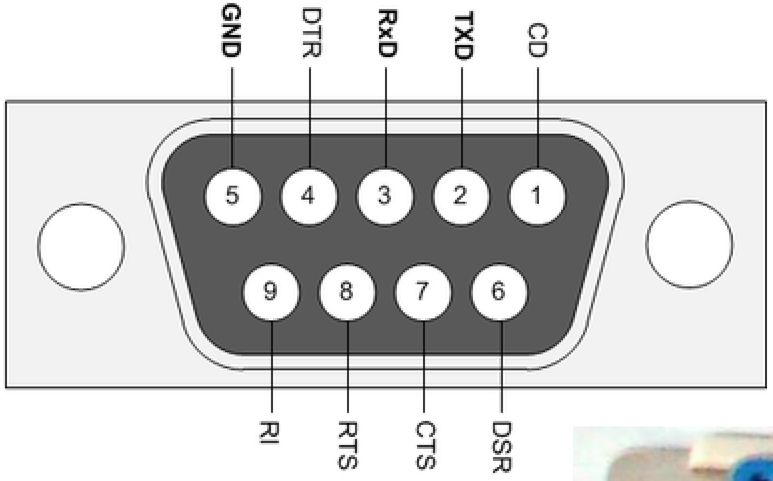


source: commons.wikimedia.org (CC)

UART

A frequently used connector for RS-232 is the DB-9. Shown here are the signals on each pin. TxD carries the transmitted data and RxD carries incoming data to the receiver.





male

female

source: commons.wikimedia.org (CC)

8.3 – SPI: SERIAL PERIPHERAL INTERFACE

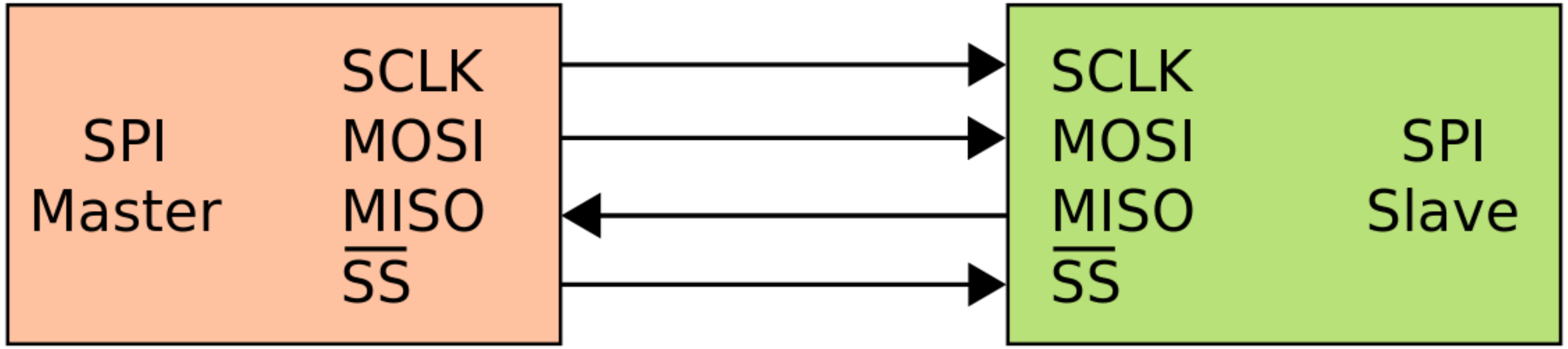
SPI = Serial Peripheral Interface

- Is a **synchronous** serial communication intended to be used to connect an MCU to external memory devices and peripherals such as: Flash EEPROM, ADC, DAC, temperature sensor, digital potentiometer, Real-Time Clock, …
- The **topology** is based in master and slave devices. There can be a single master, but multiple slaves are allowed.
- It uses 4 wires: data from master to slave, data from slave to master, clock and slave select. It is **full duplex** communication.
- There are many possible configurations; it has been reported that not all SPI devices are compatible, i.e. have a common configuration.

SPI

Single-slave connection:

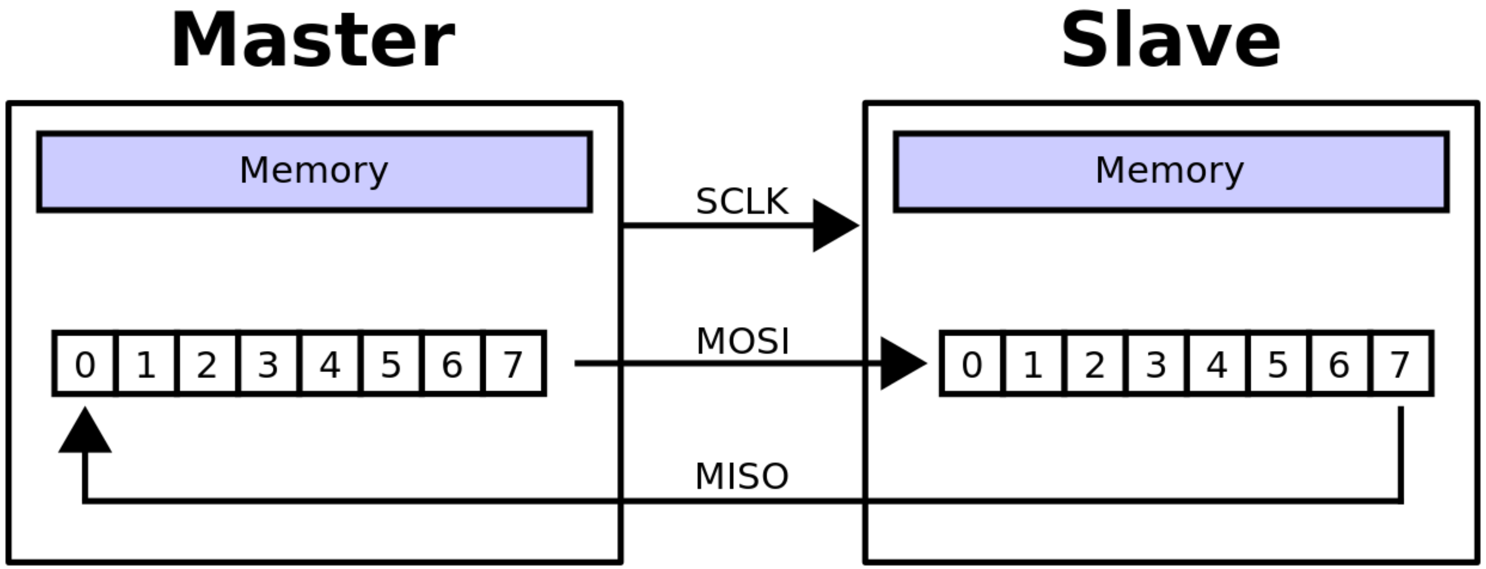
- SCLK: Serial Clock. Generated by the master;
- MOSI: Master Out Slave In;
- MISO: Master In Slave Out;
- $\overline{\text{SS}}$: Slave Select (active low).

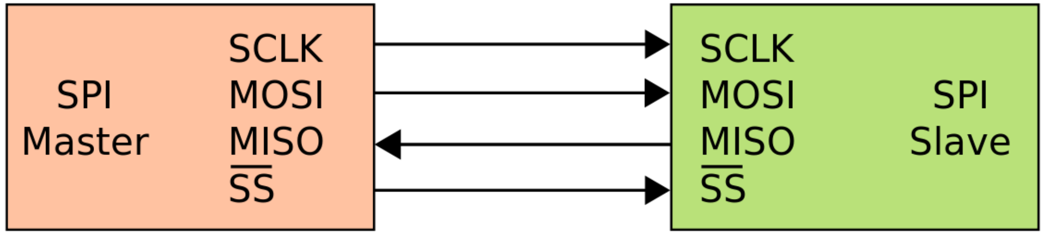


source: commons.wikimedia.org (CC)

SPI – OPERATION

- The Master selects a Slave by activating the /SS line.
- On every clock cycle, one bit is transferred from the master to the slave and another bit is transferred from the slave to the master. Not every transfer is significant.
- Typically the transfers occur in multiples of 8 bits.





source: commons.wikimedia.org (CC)

SPI – CONFIGURATION

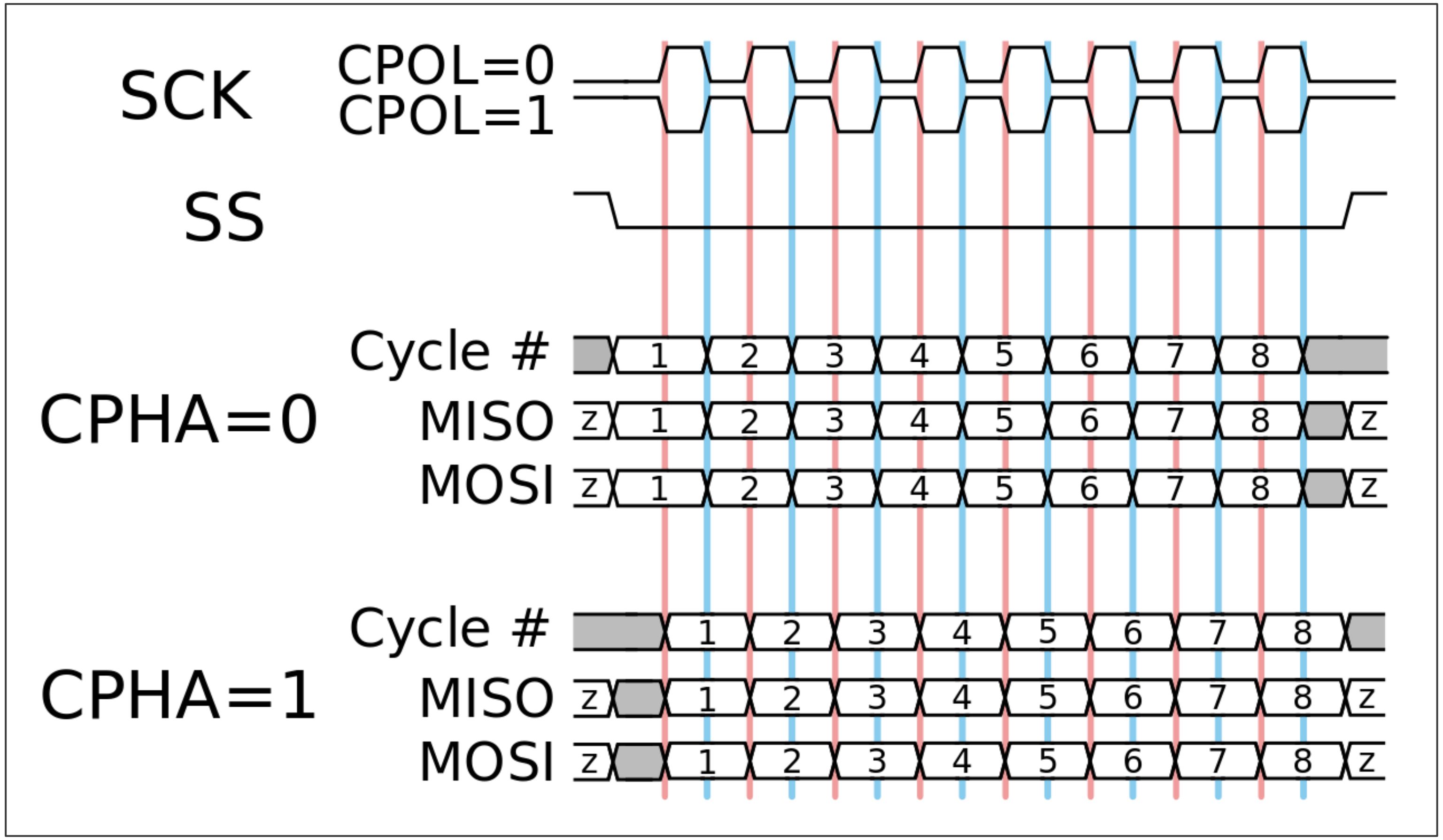
- CPOL:

  clock polarity selection

- CPHA:

  clock phase selection

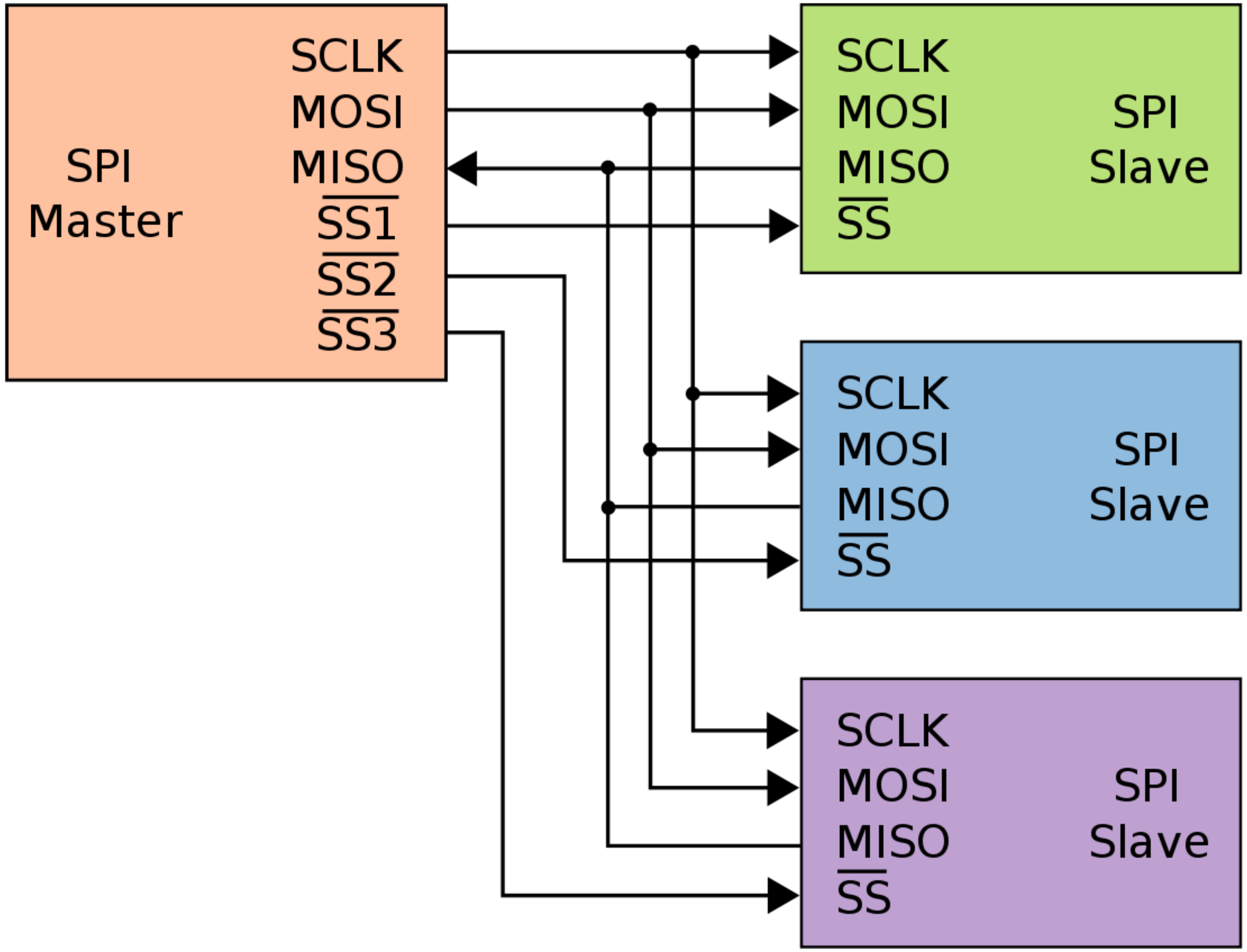
Observe that bits change on one clock edge and they are sampled on the other.



source: commons.wikimedia.org (CC)

SPI – MULTI-SLAVE

- MISO lines must be tri-state to be interconnected. They must go to high-impedance when the /SS line is not active.
- The Master selects one of the slaves to exchange data with it. Hence, separate Slave Select lines are required.



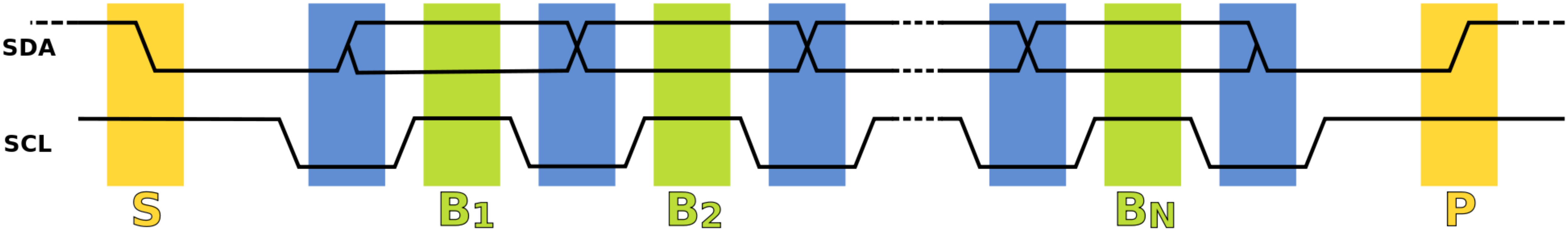
source: commons.wikimedia.org (CC)

8.4 – I2C: INTER-INTEGRATED CIRCUIT

- Synchronous half-duplex communications among multiple devices on a bus.
- Developed by a division of Philips in the 80’s. This division is now NXP.
- Needs only two wires: data and clock.
- Bus drivers are open-drain with pull-up resistors, allows for multiple transmitters connected to the same line.
- Multiple masters are allowed on a bus. A master is the one that initiates a data transfer. Addressing modes use 7-bit and 10-bit addressing.
- Data rates: 100kbps, 400 kbps, 1 Mbps, 3.4 Mbps. (version 4 added a 5 Mbps data rate using push-pull drivers on a unidirectional bus).
- Recommended reading: UM10204 – I2C-bus specification and user manual. Rev 6 – 4-April-2014. NXP.

I2C – OPERATION

- The SCL line (Serial Clock) is driven by the bus master. When SCL is high the data line (SDA) is stable and can be read. When SCL is low then the data line can change.
- The exception of this rule is the signaling of the start-bit (negative edge of SDA while SCL is high) and stop-bit (positive edge of SDA while SCL is high).



source: commons.wikimedia.org (CC)

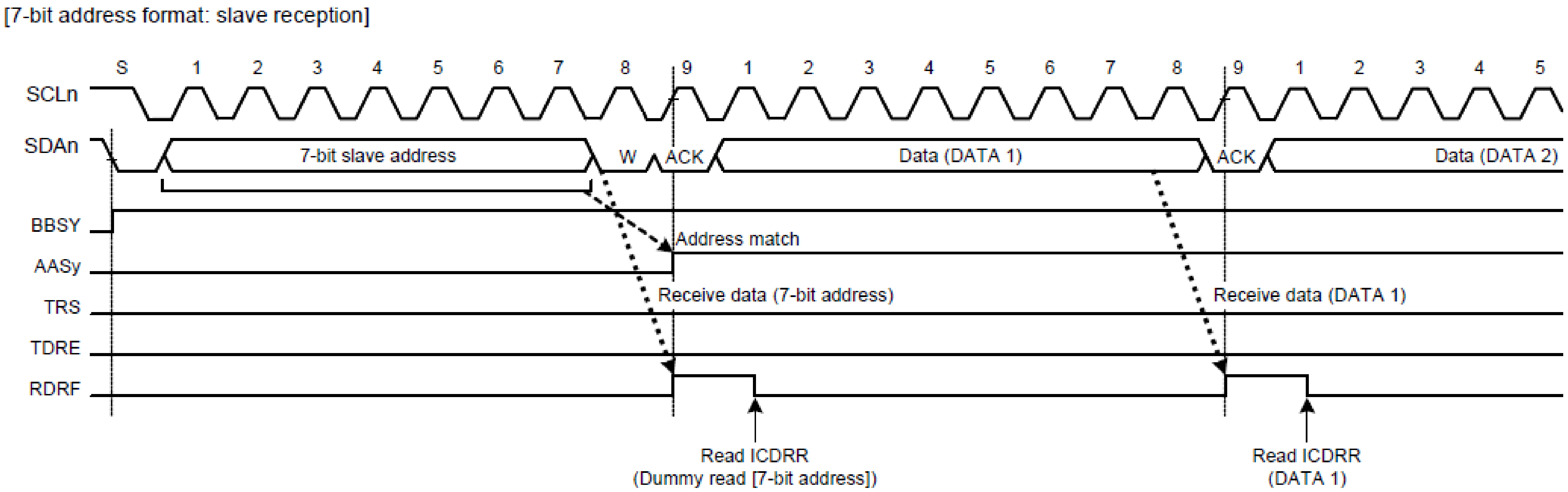
I2C – OPERATION (7-BIT ADDRESSING MODE, MASTER WRITE)

1. Master: sends the start bit.
2. Master: sends 7-bit slave address.
3. Master: sends the direction bit (0 = write).
4. Addressed slave: sends an ACK bit (0 = acknowledge).
5. Master: sends 8-bit data.
6. Addressed slave: sends ACK bit.
7. repeat steps 5 and 6 while there is data to transmit.
8. Master: sends stop bit.

I2C – OPERATION (7-BIT ADDRESSING MODE, MASTER WRITE)

- **I2C peripheral of Renesas S7G2 MCU**

  Timing diagram of a master sending data to a slave:



source: Renesas S7 Series Microcontrollers User's Manual

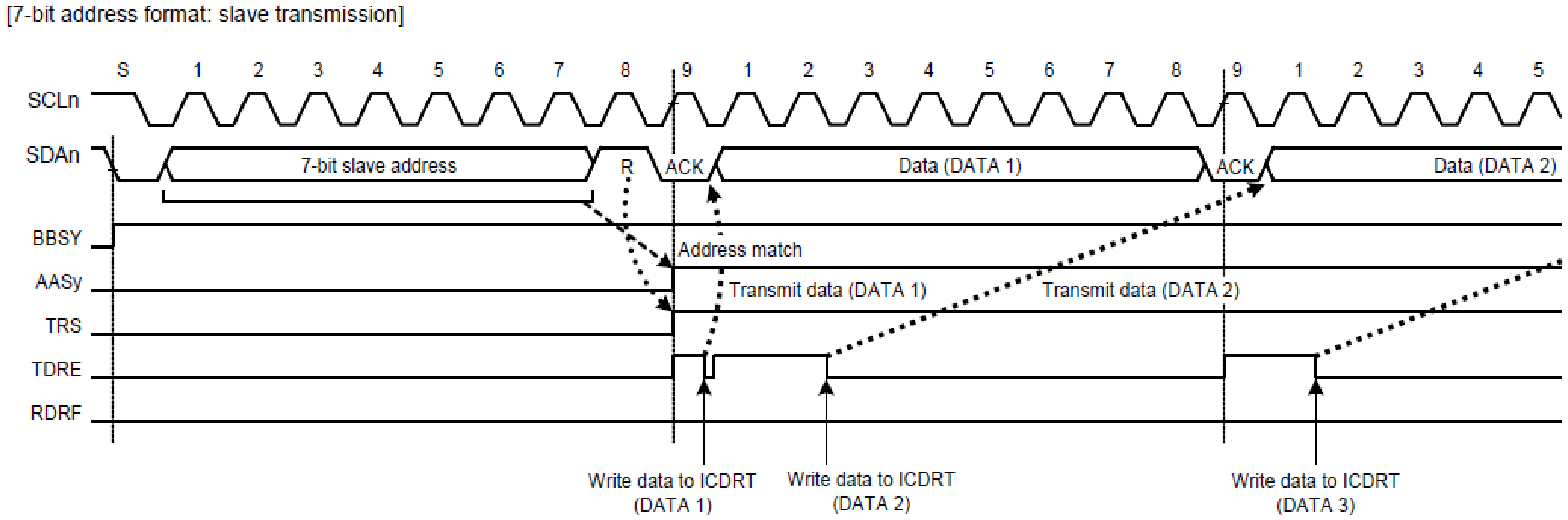
I2C – OPERATION (7-BIT ADDRESSING MODE, MASTER READ)

1. Master: sends the start bit.
2. Master: sends 7-bit slave address.
3. Master: sends the direction bit (1 = read).
4. Addressed slave: sends an ACK bit (0 = acknowledge).
5. Addressed slave: sends 8-bit data.
6. Master: sends ACK bit.
7. repeat steps 5 and 6 while there is data to transmit.
8. Master: sends stop bit.

I2C – OPERATION (7-BIT ADDRESSING MODE, MASTER READ)

- **I2C peripheral of Renesas S7G2 MCU**

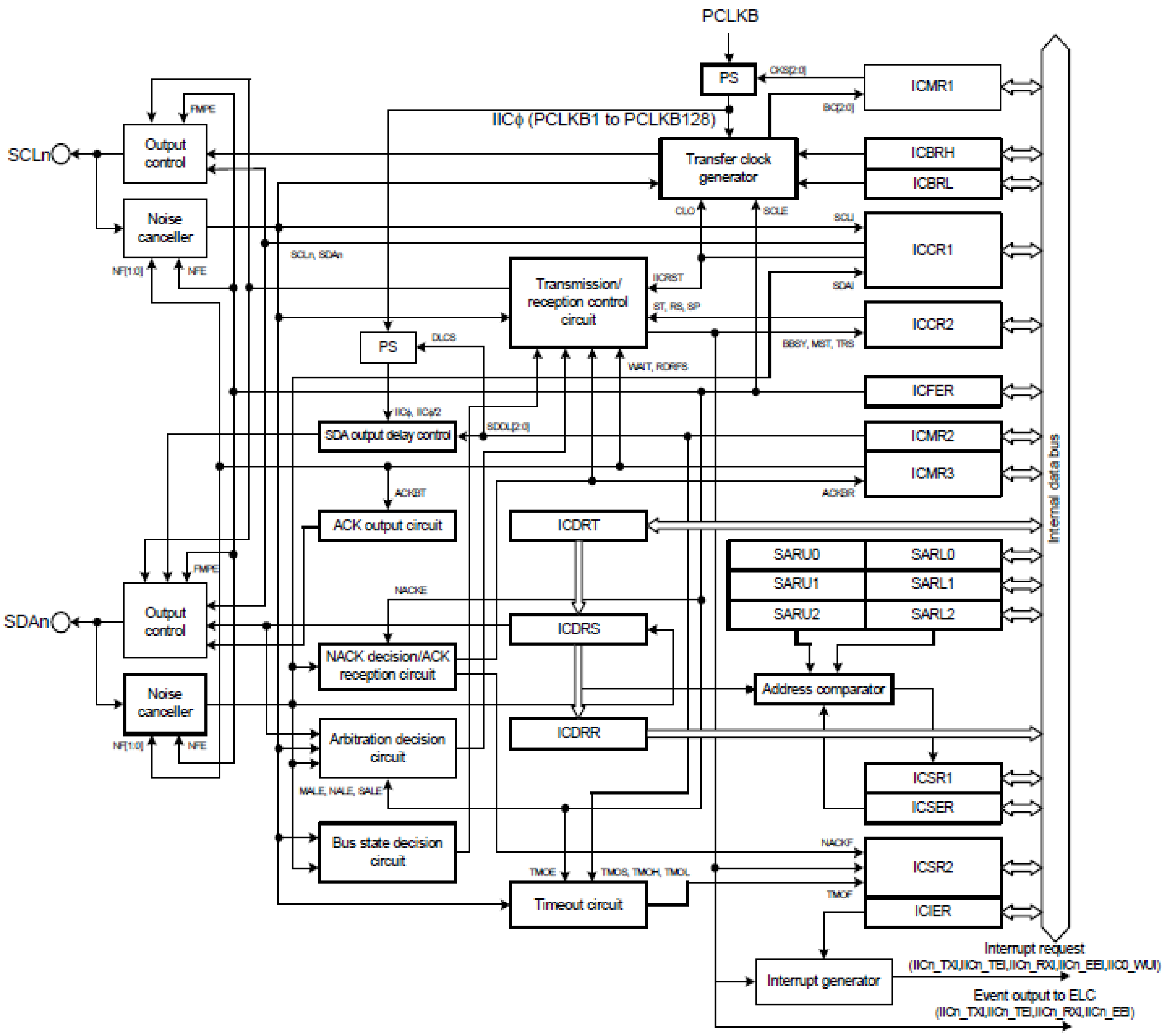
  Timing diagram of a slave sending data to a master:



source: Renesas S7 Series Microcontrollers User's Manual

I2C CASE STUDY – S7G2 IIC

Register	Name
ICCRx	I2C Control Register 1,2
ICMRx	I2C Mode Register 1,2,3
ICFER	I2C Function Enable Register
ICSER	I2C Status Enable Register
ICIER	I2C Interrupt Enable Register
ICSRx	I2C Status Register 1,2
ICWUR	I2C Wakeup Unit Register
SARLx	Slave Address Register L 0,1,2
SARUx	Slave Address Register U 0,1,2
ICBRL	I2C Bit Rate Low-Level Register
ICBRH	I2C Bit Rate High-Level Register
ICDRT	I2C Transmit Data Register
ICDRR	I2C Receive Data Register
ICDRS	I2C Shift Register



source: Renesas S7 Series Microcontrollers User’s Manual

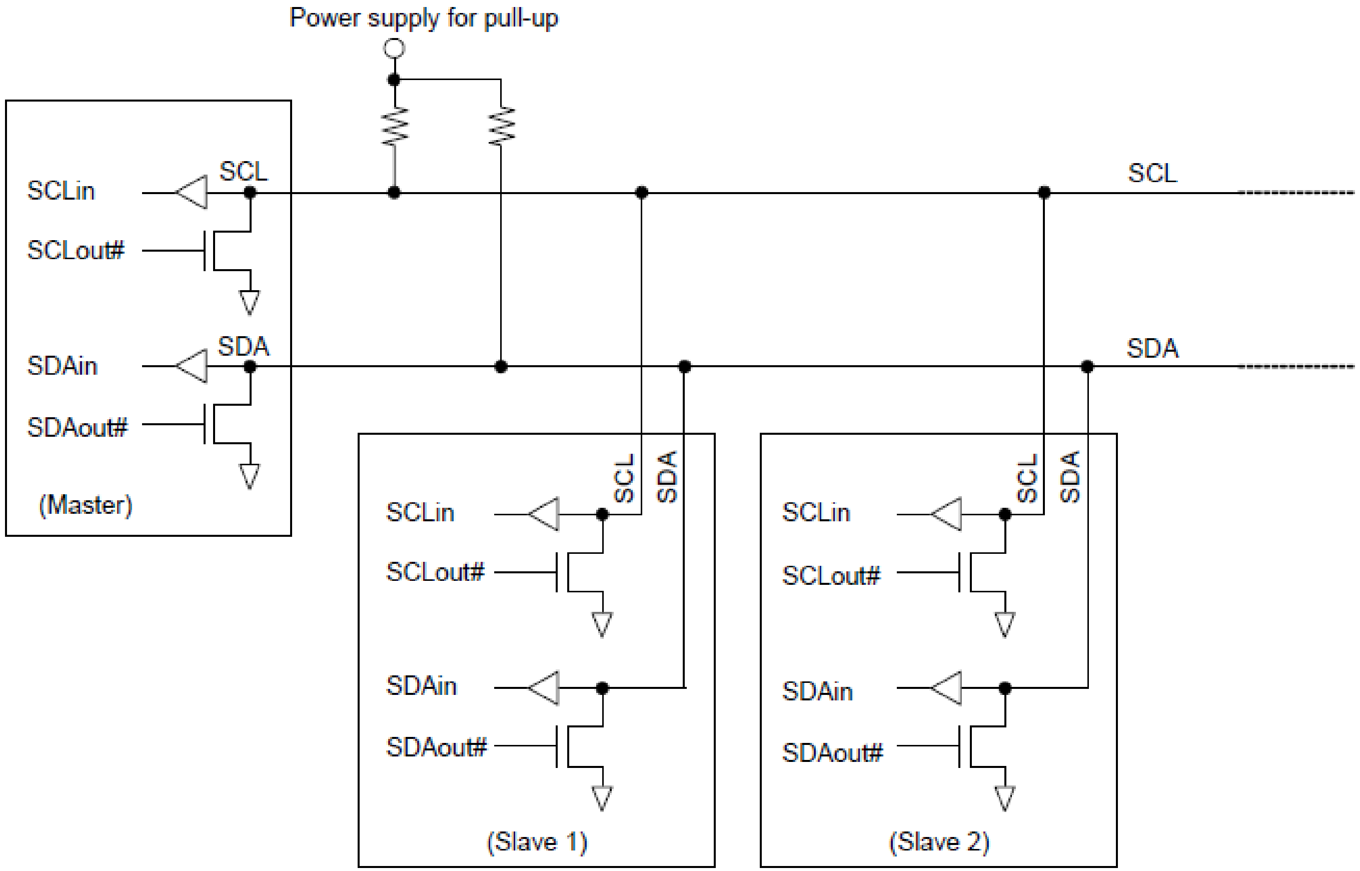
I2C CASE STUDY – S7G2 IIC

- Characteristics of the I2C Bus Interface of the Renesas S7G2 MCU:
- May operate as master or as slave of an I2C bus with data rates up to 1 Mbps.
- Up to 3 different slave addresses may be configured, 7-bit or 10-bit.
- Digital noise filters for SCL and SDA signals.
- Four interrupt sources: Receive data full, Transmit data empty, Transmit end, Error (NACK, timeout, …).

I2C CASE STUDY – S7G2 IIC

Connection of multiple devices on the I2C bus.

Notice open-drain outputs and pull-up resistors forming a wired AND.



source: Renesas S7 Series Microcontrollers User’s Manual

9 – CAN

- Introduction
- Block Diagram
- Registers
- SW Stack

9.1 – INTRODUCTION

CAN is an acronym for Controller Area Network. It is defined by the ISO-11808: 2003 standard and has been mainly motivated by the needs of the automotive industry, such as the ever increasing use of embedded sensors into the vehicles and the need to optimize the internal space and reduce costs with cabling.

Characteristics of CAN:

- Two-wire multi-master serial bus
- Message-based protocol
- Contention resolution via decentralized arbitration
- All messages are broadcast and processed by the nodes only if needed
- Speeds up to 1 Mbps

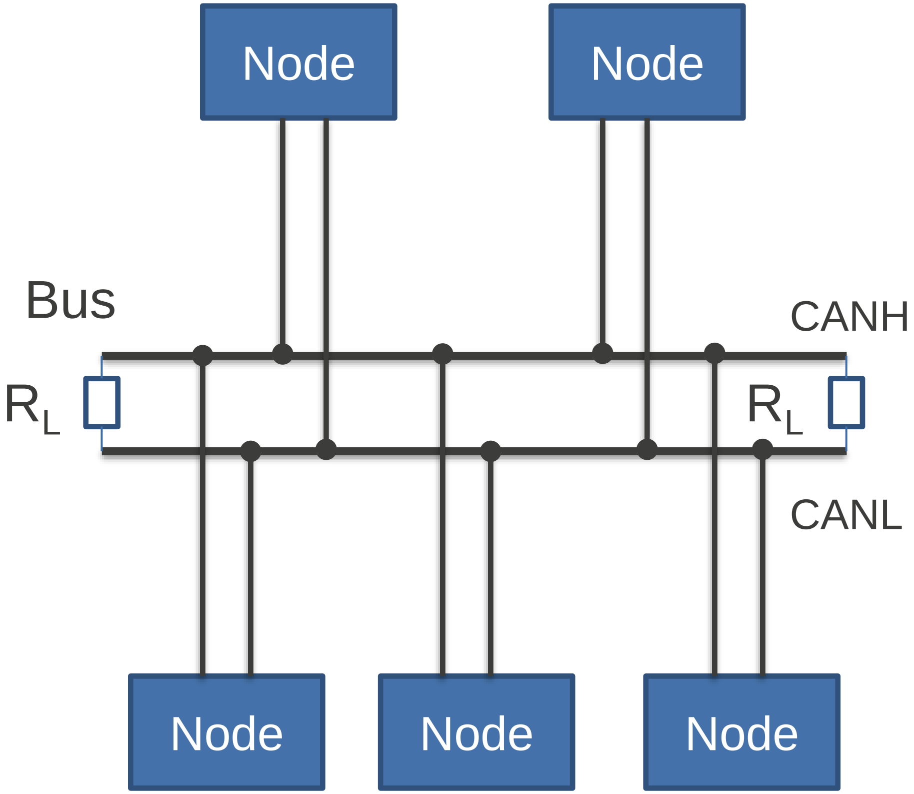
CAN TOPOLOGY

CAN nodes are interconnected in a bus topology.

CAN physical layer is implemented with two wires (CANH and CANL).

A logical 0 (called “dominant”) is obtained when CANH is approx. 3.5V and CANL is approx. 1.5V.

A logical 1 (called “recessive”) is obtained when both CANH and CANL are at approx. 2.5V.

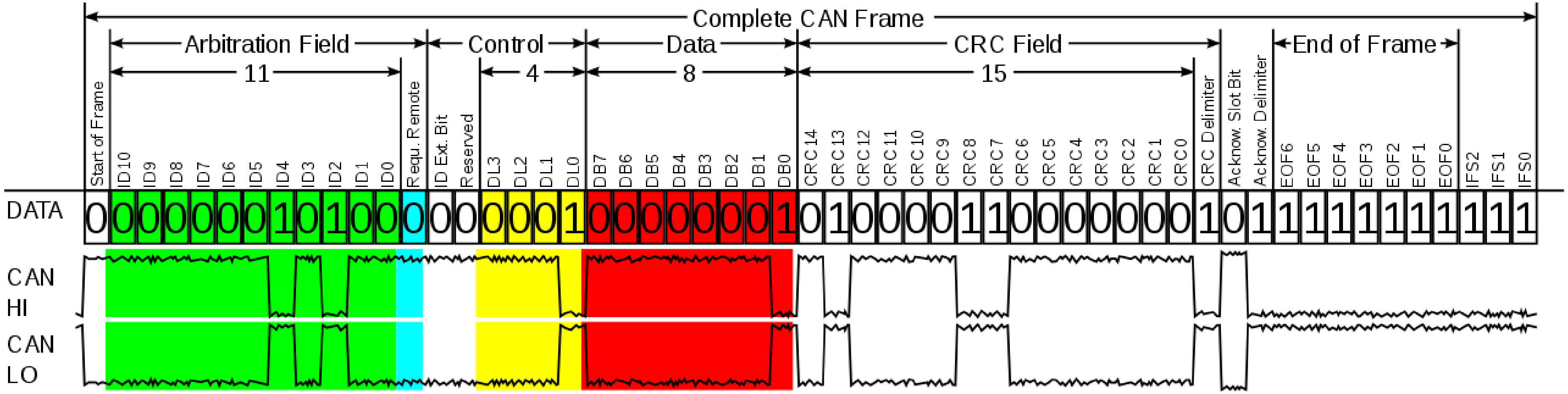


source: Authors

CAN BUS FRAME

The standard CAN frame is defined as follows:

- Arbitration field (Identifier) → defines the message identifier and its priority;
- Data → data to be transmitted (0 to 64 bits);
- SOF, CRC, ACK and End of Frame → error checking and synchronization;
- IFS (Interframe Space) → idle time used to process buffers.



	Start of Frame	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0	Requ. Remote	ID Ext. Bit	Reserved	DL3	DL2	DL1	DL0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	CRC14	CRC13	CRC12	CRC11	CRC10	CRC9	CRC8	CRC7	CRC6	CRC5	CRC4	CRC3	CRC2	CRC1	CRC0	CRC Delimiter	Acknow. Slot Bit	Acknow. Delimiter	EOF6	EOF5	EOF4	EOF3	EOF2	EOF1	EOF0	IFS2	IFS1	IFS0
DATA	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1

Source: CAN Bus (https://en.wikipedia.org/wiki/CAN_bus)

CAN BUS FRAME (CONT.)

Control → define the following sub-fields:

- Data length (0 to 8 bytes),
- Requ. Request (RTR) → used to identify a Remote Frame → see following slides,
- ID Ext. (IDE) → used to identify a Standard or Extended Frame:
  - Standard Frame → IDE is dominant “0”, 11-bit identifier as shown in picture,
  - Extended Frame → IDE is recessive “1”, 29-bit identifier. The remaining 18 bits of the identifier are placed right after the IDE bit, followed by an extra RTR bit. The original RTR is called SRR (Substitute Remote Request) and acts as a placeholder.

CAN MESSAGE TYPES

- **Data** Frame → carries data sent by a Node:
  - The RTR bit is dominant “0” to identify a Data Frame;
  - It is always preceded by an Interframe Space.
- **Remote** Frame → carries a request for a transmission of data from another Node:
  - The Arbitration/Identifier field carries the Identifier of the requested Node;
  - The RTR bit is recessive “1” to identify a Remote Frame;
  - The Data field is empty and the Data Length part of Control field determines the length of the requested message;
  - It is always preceded by an Interframe Space.

CAN MESSAGE TYPES

- **Error** frame → special format used to signal an error;
  - Not preceded by an Interframe Space.
- **Overload** frame → special format used to provide an extra delay between messages (receiver too busy);
  - Not preceded by an Interframe Space.

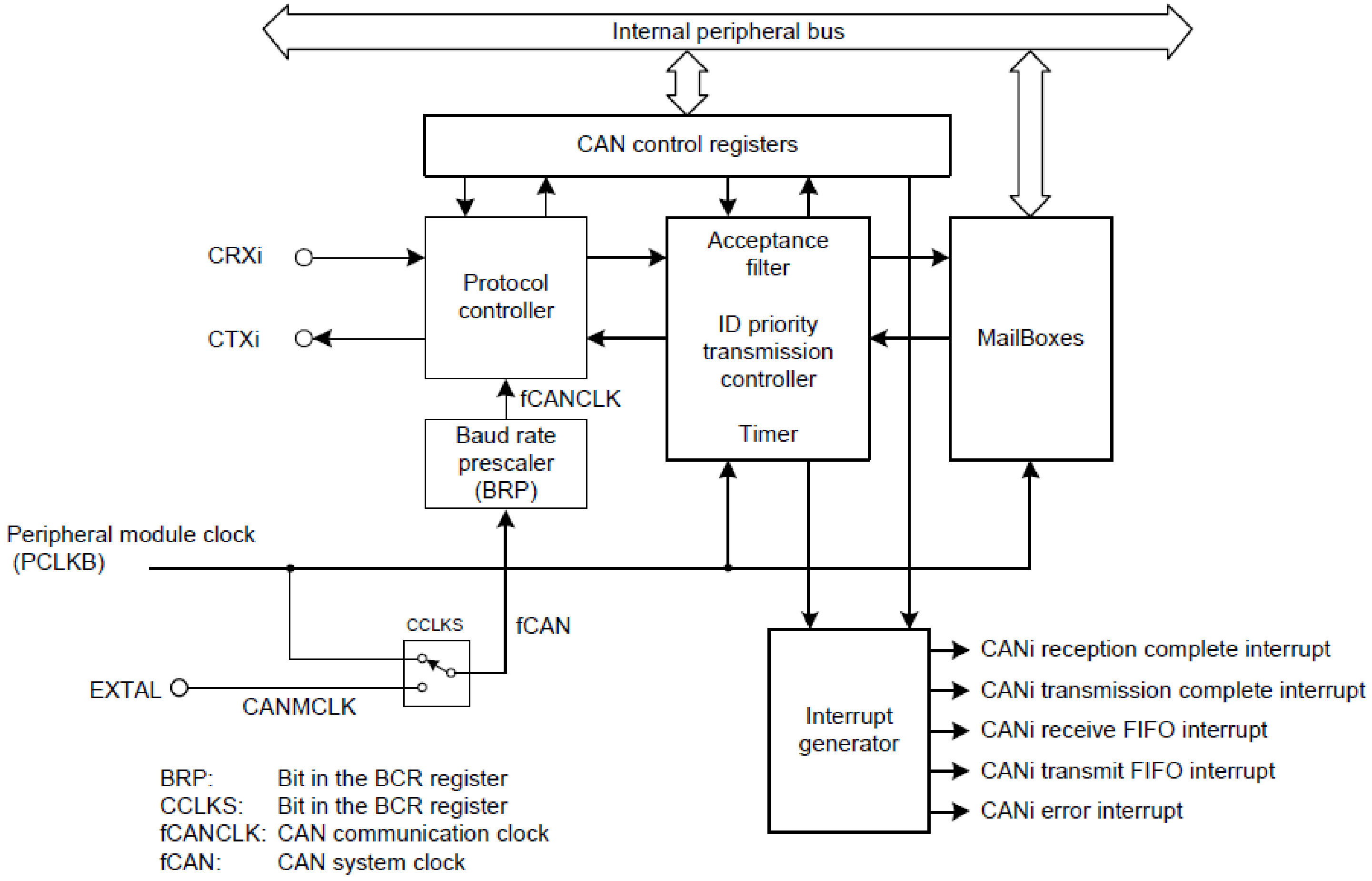
CAN ARBITRATION PROCESS

- Every message has a priority level corresponding to its identifier (arbitration field) → the lower the value, the higher the priority.
- When two or more nodes try to transmit at the same frame time:
  - Identifier bits 0 are “dominant” over identifier bits 1 “recessive”;
  - The node that sends a 1 and reads back a 0 stops transmitting on that frame → retries on the next frame;
  - The node that sends a 0 and reads back a 0 retains control and goes on to send the next identifier bit;
  - After all the identifier bits are tested, the node that keeps on retaining control (i. e. the node whose message identifier has the highest priority) sends the message contents.

9.2 – BLOCK DIAGRAM

Implementation for the CAN Module of the S7G2 MCU.

Message reception and transmission organized in *mailboxes* → configurable as single or FIFOs for different types of messages.


Internal peripheral bus
CAN control registers
CRXi
CTXi
Protocol controller
Acceptance filter
ID priority transmission controller
Timer
MailBoxes
fCANCLK
Baud rate prescaler (BRP)
Peripheral module clock (PCLKB)
CCLKS
fCAN
EXTAL
CANMCLK
Interrupt generator
CANi reception complete interrupt
CANi transmission complete interrupt
CANi receive FIFO interrupt
CANi transmit FIFO interrupt
CANi error interrupt
BRP: Bit in the BCR register
CCLKS: Bit in the BCR register
fCANCLK: CAN communication clock
fCAN: CAN system clock

Source: Renesas Synergy MCUs User's Manual: Hardware

9.3 – REGISTERS – CASE STUDY

Implementation for the CAN Module of the R7FS7G27H3A01CFC Renesas ARM Cortex-M4 MCU:

- CTLR → mailbox mode, bus operation and/or reset, timestamp configuration;
- BCR → data transfer rate configuration;
- MKR[0…7] → define masks for reception of specific messages (IDs) into specific mailboxes (depending on MKR index);
- FIDCR[0..1] → similar to MKR but for FIFO mailboxes;
- MKIVLR → enables or disables masking (via MKR) for message acceptance;
- MIER → enable/disable mailbox interrupts;
- MCTL_TX[0..31] → transmission control for each mailbox;
- MCTL_RX[0..31] → reception control for each mailbox;

9.3 – REGISTERS – CASE STUDY

- MB[0..31] → register groups for each mailbox:
  - MB[0..31].ID → received or transmitted message identifiers;
  - MB[0..31].DL → data length;
  - MB[0..31].D[0..7] → received or transmitted data;
  - MB[0..31].TS → timestamp for received messages.
- RFCR, TFCR → receive and transmit FIFO control;
- RFPCR, TFPCR → increment of the CPU-controlled pointer for receive and transmit FIFOs;
- STR → global CAN status register (new data received, receive and transmit FIFO status, CAN mode status and error status);
- EIER → enable / disable error interrupts;
- EIFR → status of error detection.

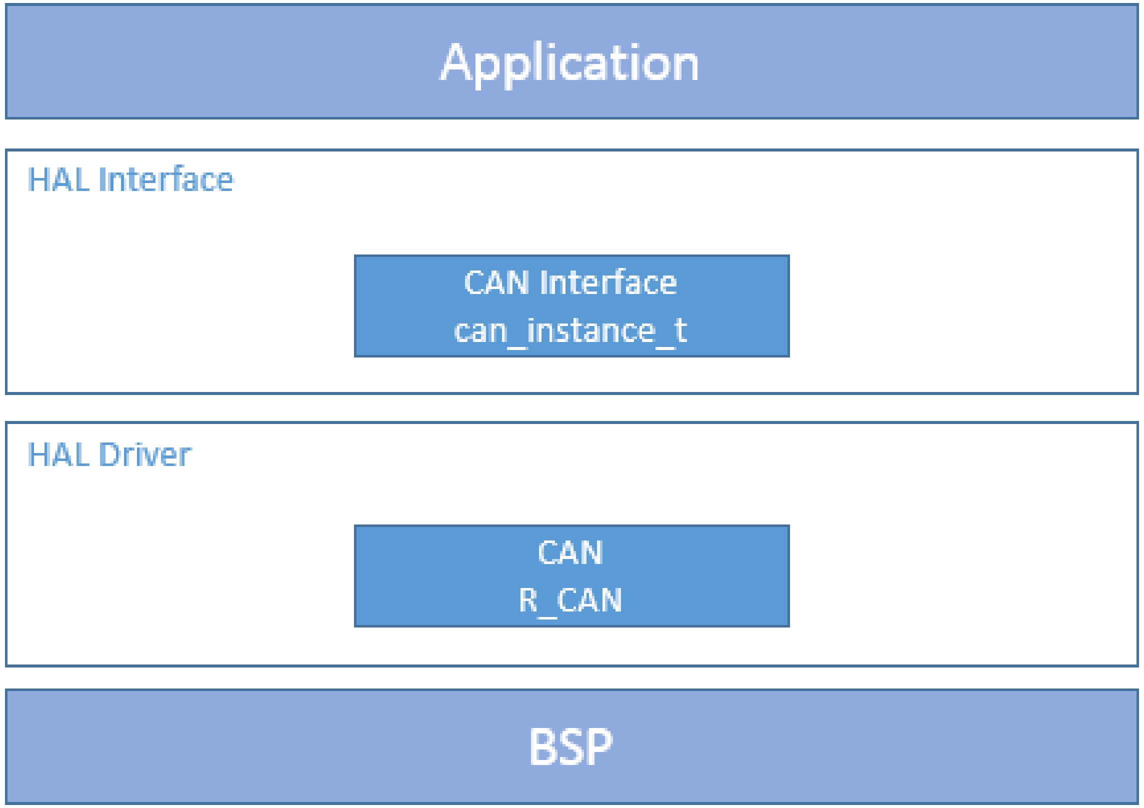
9.3 – REGISTERS – CASE STUDY

- RECR, TECR → receive / transmit error count;
- ECSR → status of CAN bus errors;
- TSR → stores the timestamp;
- TCR → test control.

9.4 – SOFTWARE STACK – CASE STUDY

Example of CAN stack for Renesas Synergy microcontroller hardware.

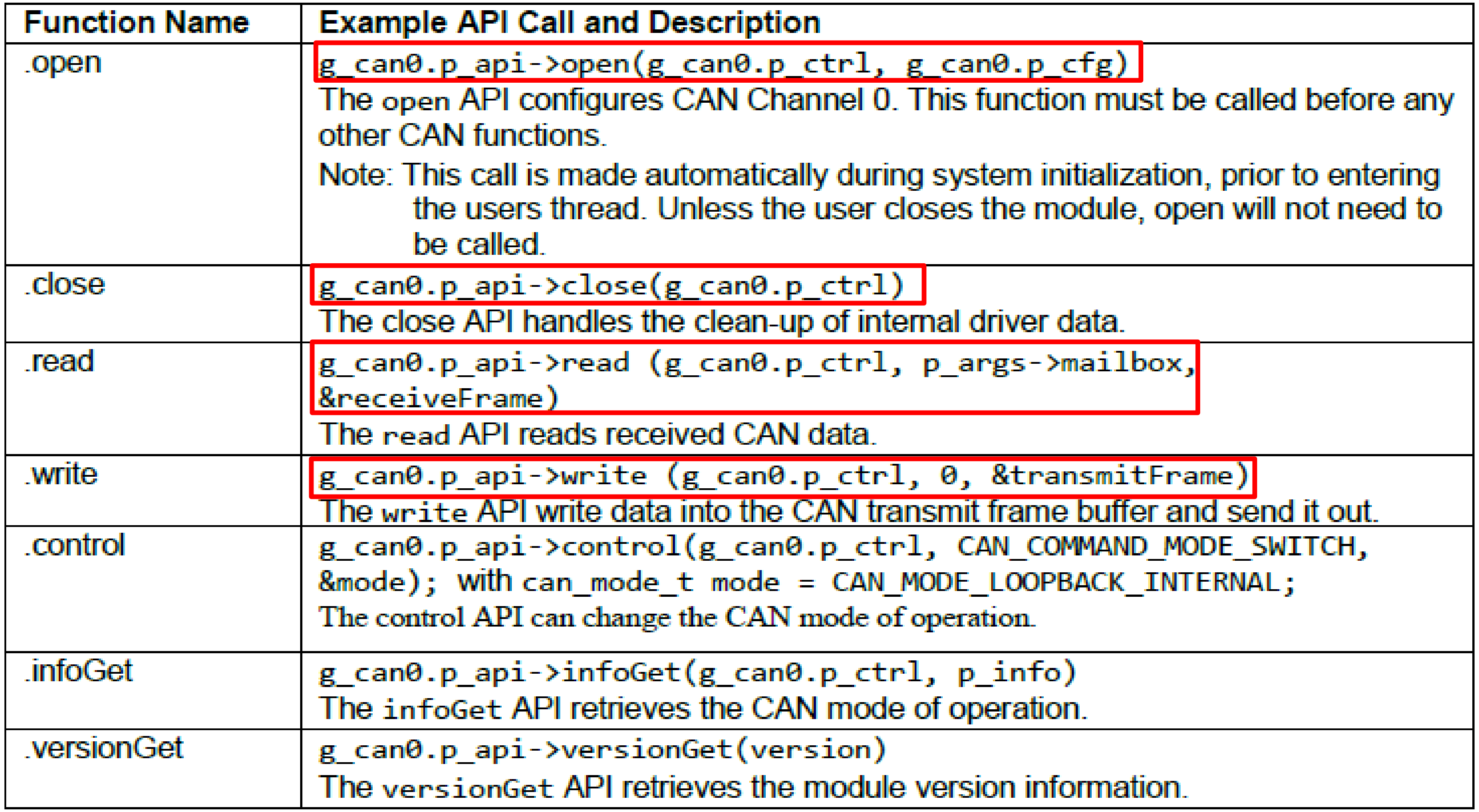
(https://www.renesas.com/en-us/software/D6001427.html)



Source: Renesas Synergy CAN HAL Driver Module Guide
r11an0065eu0101-synergy-can-hal-mod-guide

9.4 – SOFTWARE STACK – CASE STUDY

Example of CAN API for Renesas Synergy microcontroller hardware



Function Name	Example API Call and Description
.open	g_can0.p_api->open(g_can0.p_ctrl, g_can0.p_cfg) The open API configures CAN Channel 0. This function must be called before any other CAN functions. Note: This call is made automatically during system initialization, prior to entering the users thread. Unless the user closes the module, open will not need to be called.
.close	g_can0.p_api->close(g_can0.p_ctrl) The close API handles the clean-up of internal driver data.
.read	g_can0.p_api->read (g_can0.p_ctrl, p_args->mailbox, &receiveFrame) The read API reads received CAN data.
.write	g_can0.p_api->write (g_can0.p_ctrl, 0, &transmitFrame) The write API write data into the CAN transmit frame buffer and send it out.
.control	g_can0.p_api->control(g_can0.p_ctrl, CAN_COMMAND_MODE_SWITCH, &mode); with can_mode_t mode = CAN_MODE_LOOPBACK_INTERNAL; The control API can change the CAN mode of operation.
.infoGet	g_can0.p_api->infoGet(g_can0.p_ctrl, p_info) The infoGet API retrieves the CAN mode of operation.
.versionGet	g_can0.p_api->versionGet(version) The versionGet API retrieves the module version information.

Basic API functions

Source: Renesas Synergy CAN HAL Driver Module Guide r11an0065eu0101-synergy-can-hal-mod-guide

10 – USB

- Introduction
- Block Diagram
- Registers
- SW Stack

10.1 – INTRODUCTION

USB is an acronym for **Universal Serial Bus**. It has been proposed by a consortium of companies, such as Microsoft, Intel, IBM, Compaq and NEC and is designed to support a wide range of applications that require communication with distinct characteristics (real-time, high or low bandwidth, with or without message delivery guarantee etc.).

Current specification is 3.2 (Sep, 2017).

10.1 – INTRODUCTION

Examples of devices that make use of USB:

- Printers
- Cameras → interface for photo and video upload
- Smartphones → interface for battery charging and file transfer
- Human Interface Devices → keyboard, mouse etc.
- Development electronic boards → debug interface (JTAG emulation)
- Game joysticks
- …

10.1 – INTRODUCTION

Characteristics of USB:

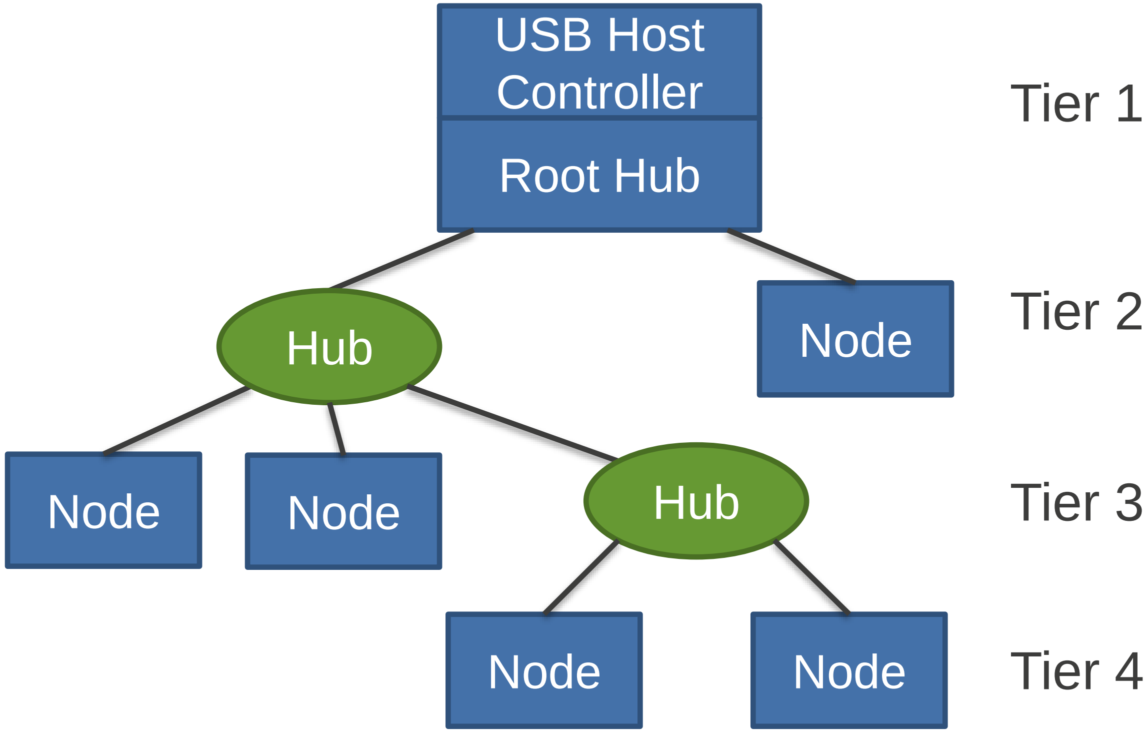
- Four (or five)-wire serial bus with single master (host) and up to 127 slaves (devices);
  - Exception → USB On-The-Go (OTG) → allows negotiation between two devices (point to point) to be a temporary host;
  - Example of OTG use: a camera device connected to a printer device to print photos;
- Defines low speed (1.5 Mbps), full speed (12 Mbps) and high speed (up to 5 Gbps at version 3.0) bandwidth
- Rem: USB 3.0 uses a 9-pin connector (USB-A 3.0 connector) or a 24-pin USB-C connector.

10.1 – INTRODUCTION

- Four types of data transfers → meet the requirements of different communication types
- Device class identification by the host via enumeration protocol → allows plug-and-play and hot swap capabilities, as well as instantiation of the proper class driver software by the host
- Bus power capabilities → some USB devices do not need an extra power source
  - USB host is able to detect overcurrent conditions, so that power can be removed from the device causing the problem without affecting the other devices already connected

USB TOPOLOGY

- USB **Host** Controller is the master and generates transactions (via Root Hub).
- Each Hub is physically connected (by wire) to a Node or another Hub.
- Nodes are slaves which perform the functions → also known as **Devices**.
- Each level defines a tier → maximum of 7 as USB 2.0 Specification.
- Nodes can be inserted or removed when necessary → upon insertion, the enumeration process is executed to identify the device class and configure it.



source: Authors

USB PHYSICAL INTERFACE

USB 1.1 and 2.0 → 4 shielded wires.

- 2 wires for data → differential
- 2 wires for power (5 Vdc and GND)

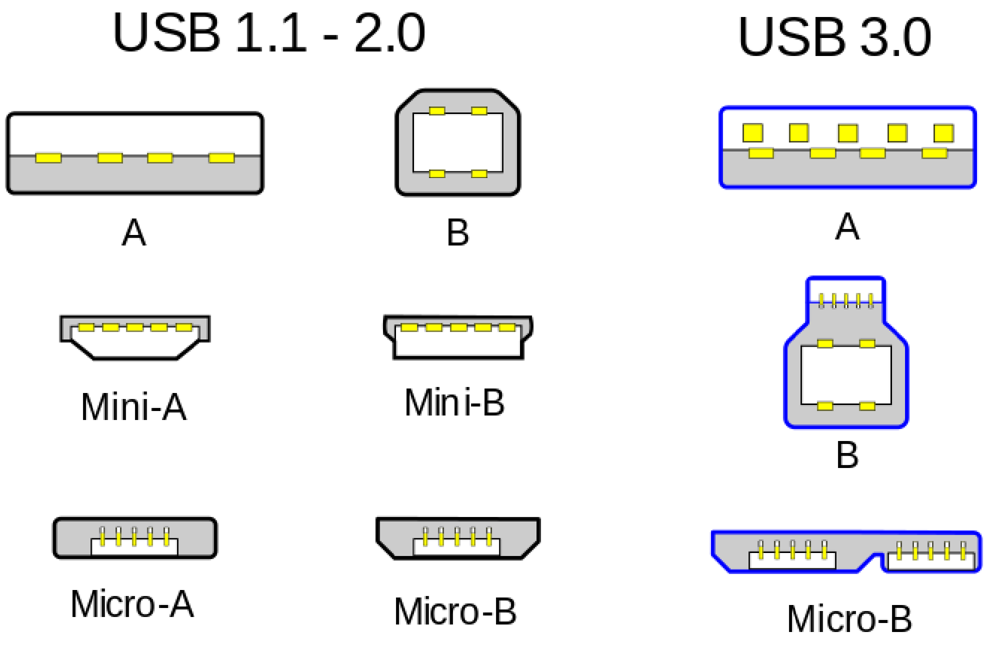
Type A → host.

Type B → device.

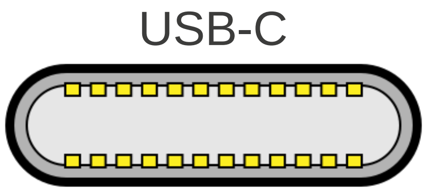
Mini and Micro variations use the same electrical interface in smaller form factors.

OTG → extra pin to identify the role of the device (A or B). The receptacle is called Micro-AB and accepts both Micro-A and Micro-B connectors.

USB 3.0 → 5 extra wires.



Source: By Milos.bmx (Own work) [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons https://commons.wikimedia.org/wiki/File%3AUSB3.0_connectors.svg



CC0
https://commons.wikimedia.org/wiki/File:USB_Type-C_icon.svg

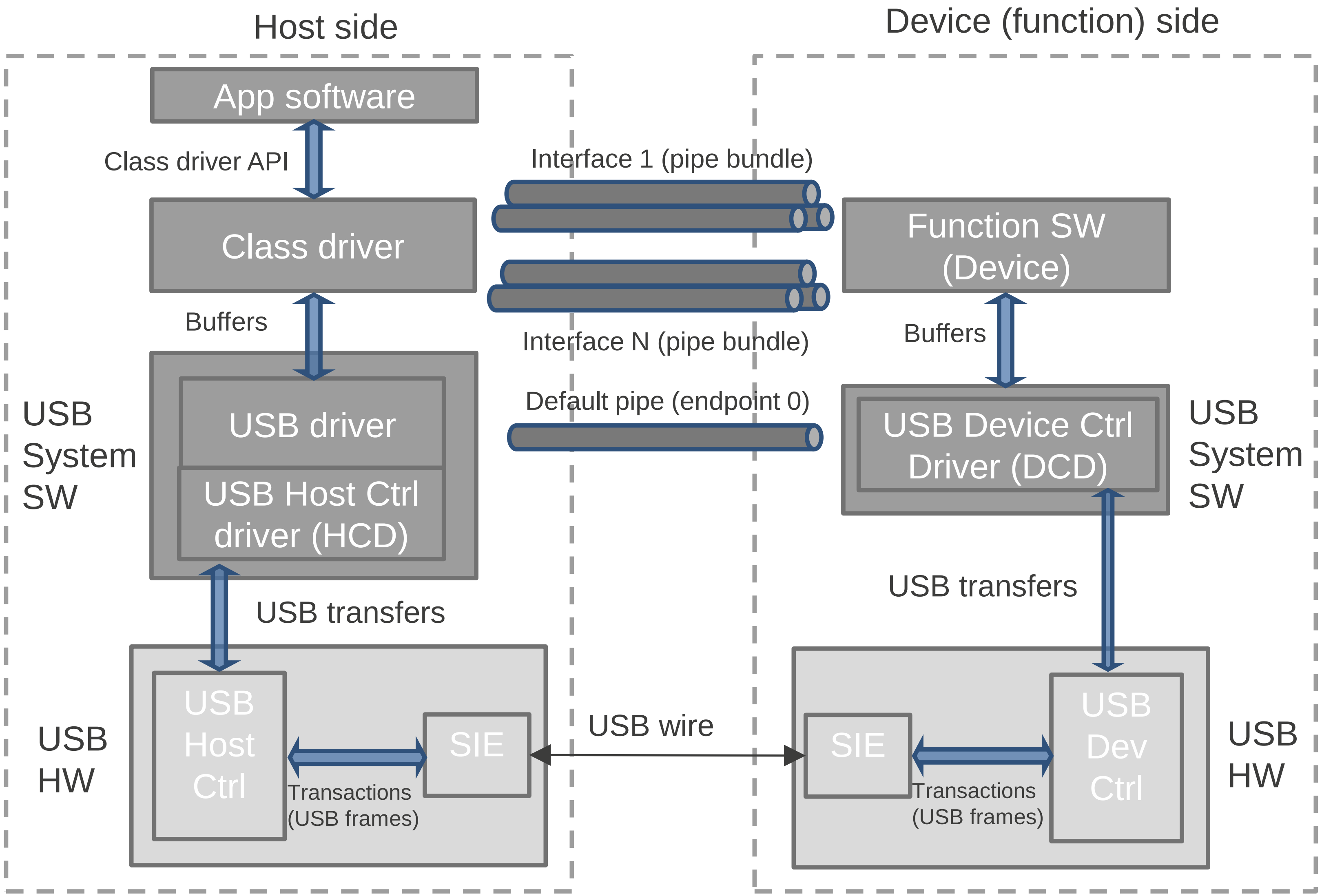
USB LOGICAL VIEW

Communication flows are performed via **pipes** → composed of **endpoints** → unidirectional data paths.

Endpoint / pipe bundles form an **interface** → a view to the function / device behavior as it is exposed to the host.

The host side instantiates a **class driver** during device enumeration → manages the interfaces and provides an API to the app level.

Default pipe is bidirectional (endpoint 0 in both directions) and is used for device configuration.



source: Authors

USB HOST CONTROLLER

- The USB Host Controller hardware layer offers an HCI (Host Controller Interface) to the Host Controller Driver in software → standardizes the register access and allows interoperability between the host OS and different hardware implementations.
- Some HCI standards have been historically defined:
  - **OHCI** (Open Host Controller Interface) → defined for USB 1.1, manages the USB bus mainly in hardware (internal FIFO descriptors management)
  - **UHCI** (Universal Host Controller Interface) → proprietary interface by Intel, defined for USB 1.1, manages most of the USB bus operation in software (HCD level)

USB HOST CONTROLLER

- **EHCI** (Enhanced Host Controller Interface) → defined for USB 2.0, manages high-speed communication on a USB bus. EHCI controllers have been usually implemented in PC motherboards in conjunction with UHCI or OHCI drivers (that managed the low and full-speed devices).
- **xHCI** (Extensible Host Controller Interface) → defined for USB 3.0, manages all the USB bus speeds. It is meant to replace the previous UHCI/OHCI/EHCI standards.

USB PACKETS

USB Transfers are performed by a sequence of transactions, which are composed of:

- A **Token** packet, carrying addressing, direction and packet type information (IN, OUT or SETUP). The token can be of PID type **Start-Of-Frame (SOF)**, issued every 1 ms (full-speed) or 125 us (high-speed) and used for synchronization.
  - Frame → interval during which a sequence of transactions is performed for the endpoints controlled by the host..

Field	PID	ADDR	ENDP	CRC5
Bits	8	7	4	5
Desc	Type of packet	Device address	Endpoint address	CRC of ADDR and ENDP

Token packet

Field	PID	Frame number	CRC5
Bits	8	11	5
Desc	Type of packet	Current frame	CRC of Frame Number

SOF packet

USB PACKETS

- A **Data** packet, carrying the effective data being transferred. Data packets are issued either by the host or the device, depending on the endpoint direction (identified by the previous Token packet). PID indicates type DATA0 or DATA1 (toggling for full-speed transfers) or DATA2 (high-speed transfers).
- A **Handshake** packet, used to report the status of a data transaction. Handshake packets are issued by the receiver of the Data packet. PID indicates ACK, NACK, a halt condition (STALL) or no response yet (NYET).

Field	PID	DATA	CRC16
Bits	8	0-8192	16
Desc	Type of packet	Data	CRC of DATA

Data packet

Field	PID
Bits	8
Desc	Type of packet

Handshake packet

USB TRANSFERS

USB defines four types of data transfers:

- **Control** → control commands to configure device, delivery guaranteed, low bandwidth required.
  - Control transfers are performed in three stages:
    - A *Setup* stage, starting with a token packet of type SETUP and a data packet containing a USB device request → see following slides.
    - An optional *Data* stage, starting with a token packet of type IN or OUT (depending on direction) and a data packet containing the data pertaining to the USB device request.
    - A *Status* stage, starting with a token packet of type IN or OUT (inverse of Data direction) and containing request status information.

USB TRANSFERS

- **Bulk** → large amounts of data, non real-time, delivery guaranteed, variable use of bandwidth → used for reliable data transfers, such as mass storage data.
  - Token packets for bulk transfers are of type IN or OUT, depending on transfer direction.
- **Interrupt** → real-time, small and periodic amounts of data → used for event notification (e.g. key typed on a keyboard).
  - Token packets for interrupt transfers are of type IN or OUT, depending on transfer direction.

USB TRANSFERS

- **Isochronous** → large amounts of data, delivery not guaranteed, steady rate of transmission and reception, bandwidth depending on sampling characteristics → used for streaming data, such as voice or video.
  - Token packets for isochronous transfers are of type IN or OUT, depending on transfer direction.
  - Isochronous transfers do not use Handshake packets.

Individual endpoints are configured to a specific type of data transfer, depending on the class driver loaded by the host during the enumeration process → see following slides.

USB TRANSFERS SCHEDULING

Rules for transfer scheduling:

- Periodic transfers (isochronous and interrupt) → limited to 90% of the bandwidth of a frame.
- Control → use as much as necessary of the remaining 10% (plus the remaining amount in the 90% of the bandwidth that is not used for periodic transfers).
- Bulk → use the bandwidth that is left.

USB ENUMERATION

The USB enumeration protocol is executed whenever a new device is inserted into the bus. This protocol comprises the following steps:

- USB root hub detects when a device is connected (D- or D+ are pulled up with resistors).
- USB host powers and resets the device.
- USB host issues device requests through the Default Control Pipe (default address 0) to get the Device Descriptor → see following slides.
- USB host assigns a unique address to the device.
- USB host issues device requests through the Default Control Pipe to get the Configuration Descriptors → see following slides.
- USB host enables a valid configuration → all corresponding interfaces and endpoints are configured, and the device may draw the current described in the descriptor for the selected configuration.

USB ENUMERATION

Some steps of the enumeration protocol require issuing USB device requests to the device being enumerated.

The USB device requests are issued during the Setup stage of a Control transfer. A device request is 8 bytes long and contains the following fields:

- *bmRequestType* (1 byte) → request direction, type (standard, class, vendor, reserved) and recipient (device, interface, endpoint, other).
- *bRequest* (1 byte) → specific request (set address, get and set configuration, get and set descriptor, get and set interface etc.).
- *wValue* (2 bytes), *wIndex* (2 bytes) → value and index that depend on request.
- *wLength* (2 bytes) → number of bytes to transfer if there is a data stage.

Refer to USB 2.0 Specification, Section 9.3 for more detailed information.

USB ENUMERATION

Descriptors sent by the device during enumeration process (in response to GET_DESCRIPTOR requests):

- **Device descriptor** → defines the device class, device subclass, device protocol, max packet size for default endpoint, vendor, product, release number, indices for manufacturer, product and serial number strings, and the number of configurations.

Device Descriptor

Field	Size (bytes)	Descr
bLength	1	Size of descriptor
bDescriptorType	1	DEVICE descriptor type
bcdUSB	2	USB Spec Relase Number in BCD
bDeviceClass	1	Class code
bDeviceSubClass	1	Subclass code
bDeviceProtocol	1	Protocol code
bMaxPacketSize0	1	Max packet size for endp 0
idVendor	2	Vendor ID
idProduct	2	Product ID
bcdDevice	2	Device release number in BCD
iManufacturer	1	Index of string desc for manufacturer
iProduct	1	Index of string desc for product
iSerialNumber	1	Index of string desc for serial
bNumConfigurations	1	Number of possible configurations

Source: USB 2.0 Specification, Table 9-8

USB ENUMERATION

- **Configuration descriptor** → defines the number of interfaces for this configuration, the configuration value and an index for this configuration’s string, if the device is self-powered when running that configuration and the max power consumption (in case it is bus powered).
  - A GET_DESCRIPTOR request to a Configuration Descriptor returns also the Interface and Endpoint descriptors pertaining to the given Configuration, in sequential order → see next slides.

Configuration Descriptor

Field	**Size (bytes)**	**Descr**
bLength	1	Size of descriptor
bDescriptorType	1	CONFIGURATION descriptor type
wTotalLength	2	Total length of configuration data (includes Interface and Endpoint descriptor sizes)
bNumInterfaces	1	Number of interfaces
bConfigurationValue	1	Configuration ID
iConfiguration	1	Index of string desc for this config
bmAttributes	1	Configuration characteristics
bMaxPower	1	Max power consumption in mA when operating on this configuration

Source: USB 2.0 Specification, Table 9-10

USB ENUMERATION

- **Interface descriptor** → defines the interface number, the number of endpoints, the interface class, subclass and protocol, and an index to a string describing this interface.

Interface Descriptor

Field	Size (bytes)	Descr
bLength	1	Size of descriptor
bDescriptorType	1	INTERFACE descriptor type
bInterfaceNumber	1	Zero-based number of this interface
bAlternateSetting	1	Value to select this alternate setting
bNumEndpoints	1	Number of endpoints used by this interface
bInterfaceClass	1	Class code for this interface
bInterfaceSubClass	1	Subclass code for this interface
bInterfaceProtocol	1	Protocol code for this interface
iInterface	1	Index of string desc for this interface

Source: USB 2.0 Specification, Table 9-12

USB ENUMERATION

- **Endpoint descriptor** → defines the endpoint address and direction, the endpoint type (control, isochronous, bulk or interrupt), the max packet size and the polling interval for periodic endpoints (isochronous and interrupt).

- Refer to USB 2.0 Specification, Section 9.6 for more detailed information.

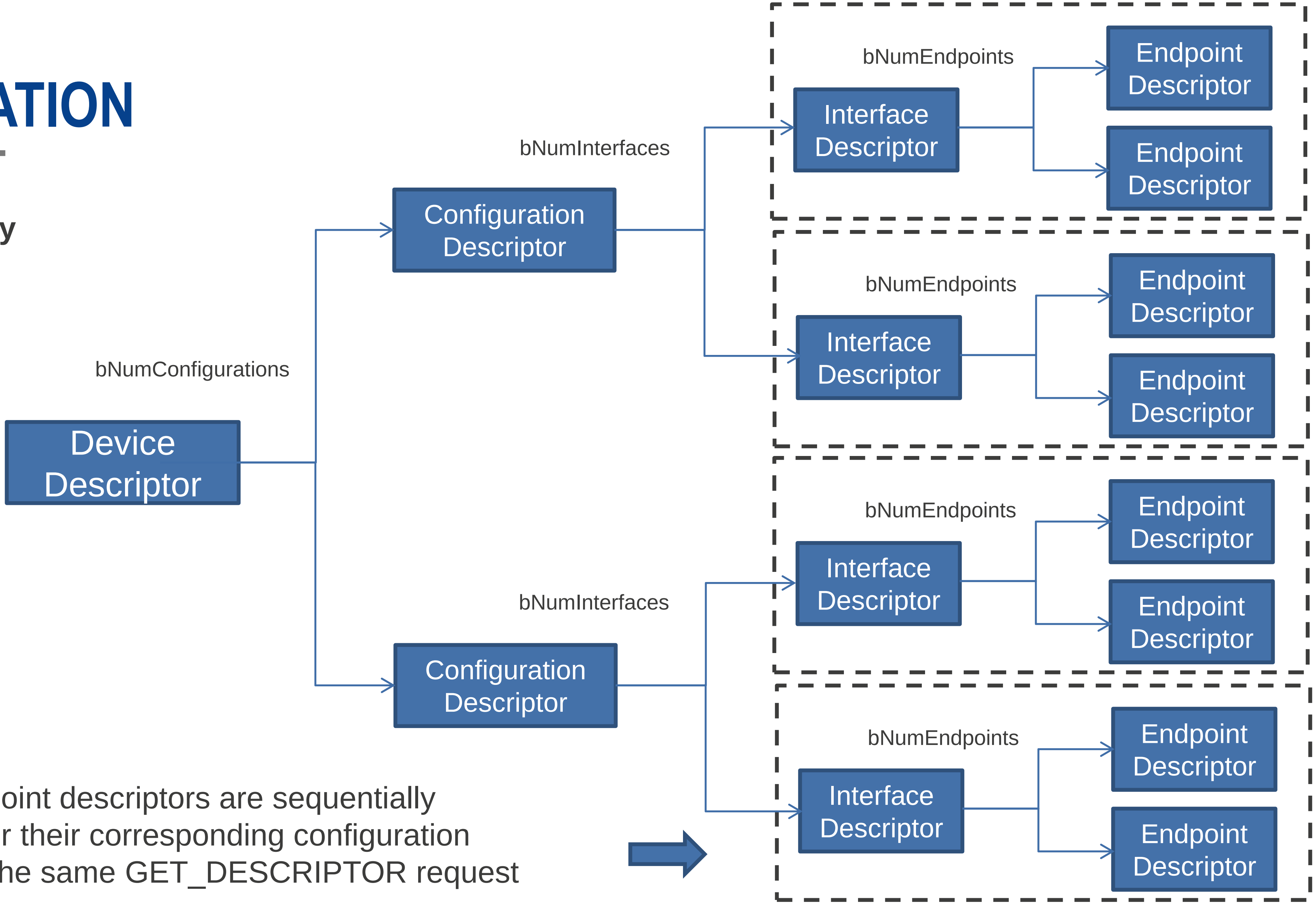
Endpoint Descriptor

Field	Size (bytes)	Descr
bLength	1	Size of descriptor
bDescriptorType	1	ENDPOINT descriptor type
bEndpointAddress	1	Address for this endpoint
bmAttributes	1	Endpoint attributes (type etc.)
wMaxPacketSize	2	Max packet size for this endpoint
bInterval	1	Polling interval in frames

Source: USB 2.0 Specification, Table 9-13

USB ENUMERATION

USB Descriptor hierarchy



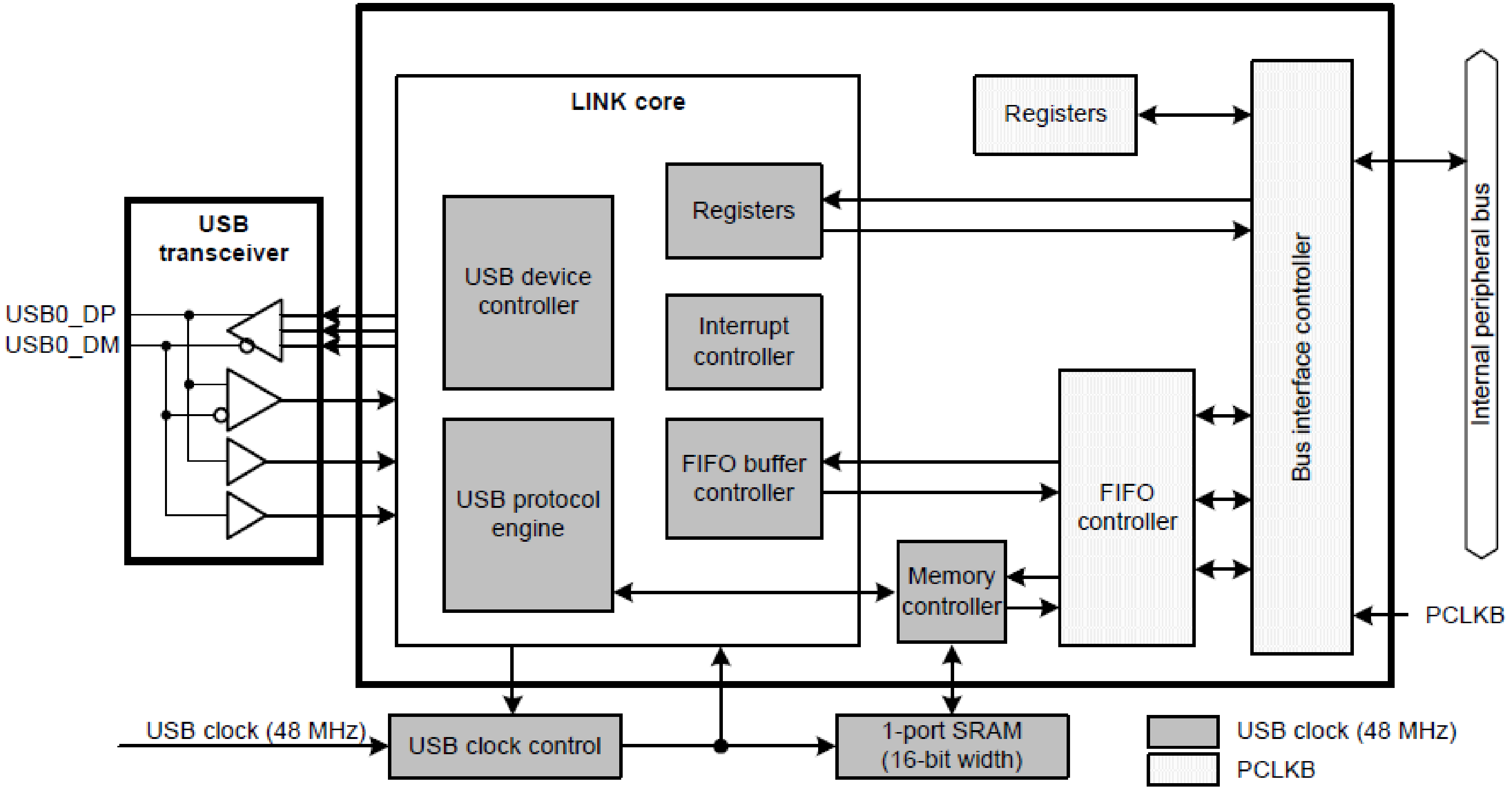
Interface and endpoint descriptors are sequentially retrieved, right after their corresponding configuration descriptor, during the same GET_DESCRIPTOR request

Source: Authors

10.2 – BLOCK DIAGRAM – CASE STUDY

The R7FS7G27H3A01CFC Renesas ARM Cortex-M4 MCU implements two USB modules:

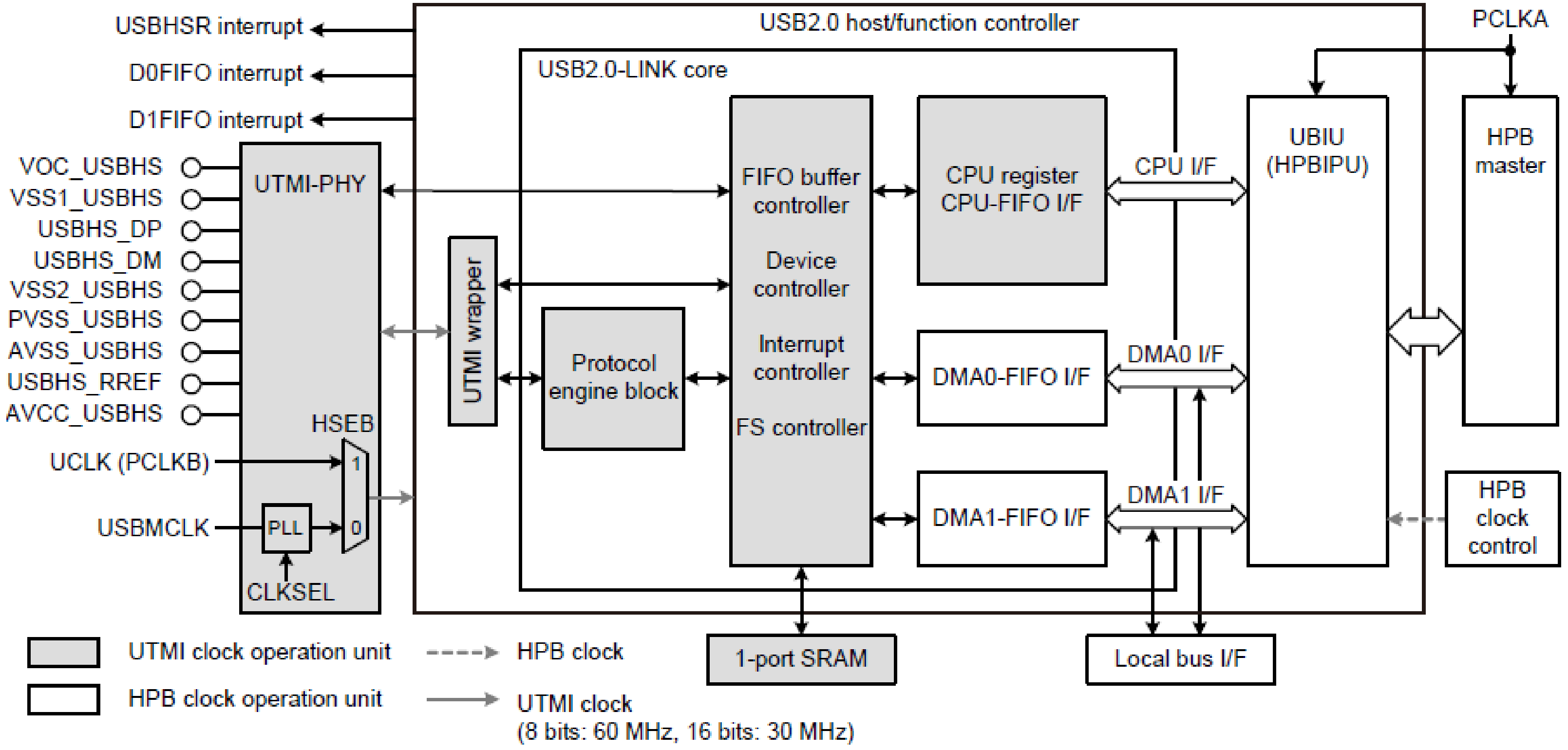
- **USB 2.0 FS** → operates only on low and full speed modes. Based on registers and a FIFO controller to manage buffers to be received / transmitted.



Source: Renesas Synergy MCUs User's Manual: Hardware

10.2 – BLOCK DIAGRAM – CASE STUDY

- USB 2.0 HS → operates in high speed mode (480 Mbps). Uses DMA FIFOs to maximize memory transfer speed.



Source: Renesas Synergy MCUs User’s Manual: Hardware

10.3 – REGISTERS – CASE STUDY

Implementation for the USB 2.0 FS Module of the R7FS7G27H3A01CFC Renesas ARM Cortex-M4 MCU:

- SYSCFG → enabling/disabling USB, pull up / pull down resistor config
- SYSSTS0 → line status, overcurrent status (from an external overcurrent detector), status bits for entering / exiting the "suspended" mode
- DVSTCTR0 → connection status (reset, low-speed or full-speed), wakeup detection, enable / resume / reset control
- CFIFO, D0FIFO and D1FIFO → read/write from/to FIFOs associated to control pipe and to other communication pipes
- CFIFOSEL → configure control pipe and associate to CFIFO
- D0FIFOSEL, D1FIFOSEL → associate pipes to D0FIFO and D1FIFO, configure DMA

10.3 – REGISTERS – CASE STUDY

- CFIFOCTR, D0FIFOCTR, D1FICOCTR → received data length, status of FIFO read
- INTENB0, INTENB1 → enable / disable USB interrupts
- BRDYENB → enable / disable BRDY (data transfer successful) interrupt for each USB pipe
- NRDYENB → enable / disable NRDY (data transfer not successful) interrupt for each USB pipe
- BEMPENB → enable / disable BEMP (buffer empty or incorrect packet size) interrupt for each USB pipe
- SOFCFG → configuration for SOF (start-of-frame) and frame timing (LS and FS)
- INTSTS0, INTSTS1 → status of several interrupt sources (SOF, resume, BRDY, NRDY, overcurrent, disconnection etc.)

10.3 – REGISTERS – CASE STUDY

- BRDYSTS → status of BRDY interrupt for each USB pipe
- NRDYSTS → status of NRDY interrupt for each USB pipe
- BEMPSTS → status of BEMP interrupt for each USB pipe
- FRNUM → frame number, status of CRC error and overrun/underrun in isochronous transfers
- DVCHGR → used when device recovers from deep software standby mode due to USB events
- USBADDR → USB device address, configuration for recovery from deep software standby mode
- USBREQ → fields of setup requests used for control transfers.
- USBVAL → stores the wValue field of setup transactions (received and for transmitting)

10.3 – REGISTERS – CASE STUDY

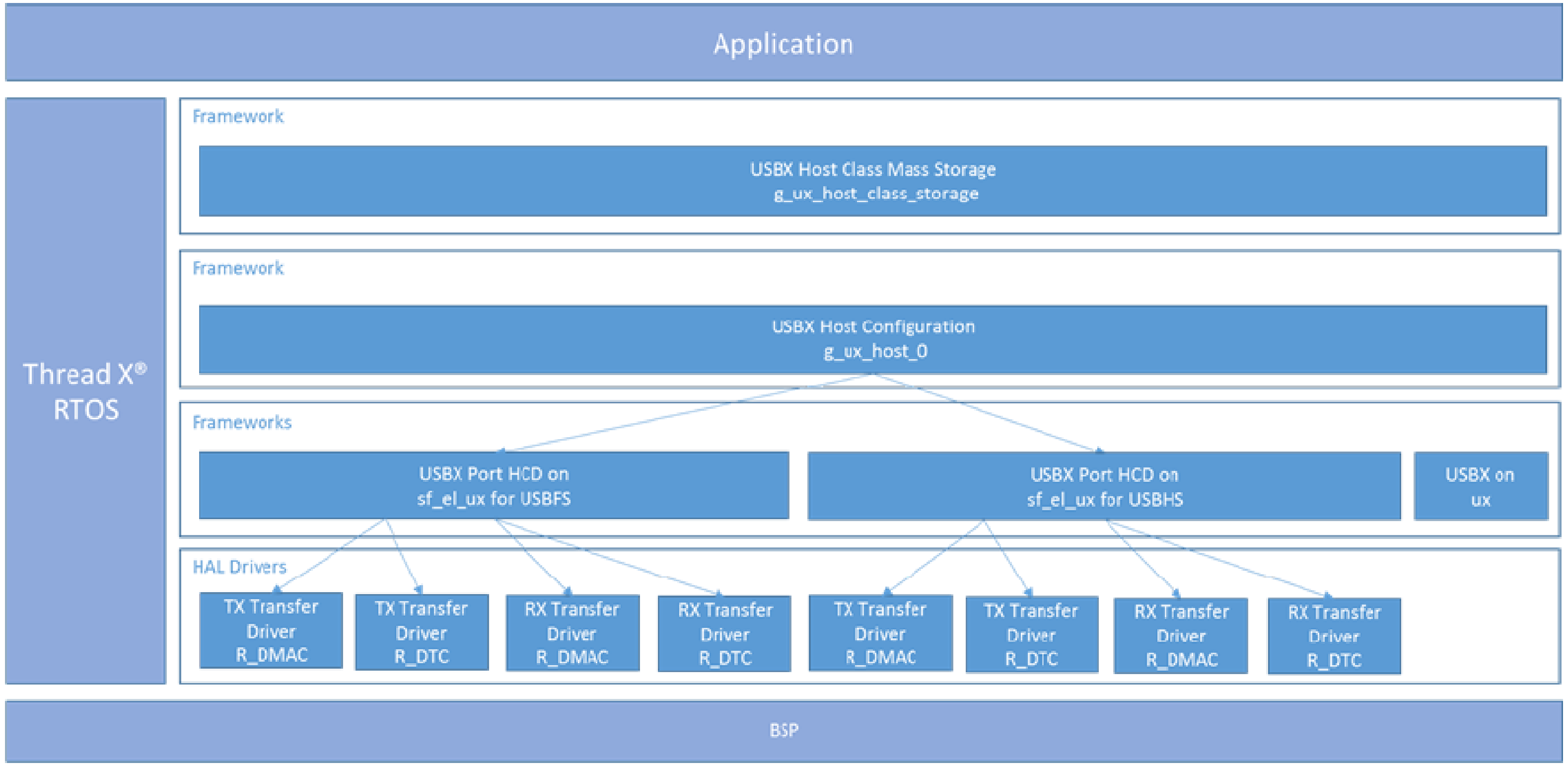
- USBLENG → stores the wLengths field of setup transactions (received and for transmitting)
- DCPCFG → enabling and direction of the Default Control Pipe
- DCPMAXP → maximum packet size for the Default Control Pipe
- DCPCTR → controls transfers for the Default Control Pipe
- PIPESEL → select pipe to be configured by PIPECFG, PIPEMAXP etc.
- PIPECFG → configures selected pipe (endp number, direction, transfer type etc.)
- PIPEMAXP → configures maximum packet size for selected pipe
- PIPEPERI → configures error detection interval for isochronous pipes
- PIPECTR[1..9] → controls transfers for the corresponding pipe
- PIPETRE[1..5] → enables / disables transaction counter

10.3 – REGISTERS – CASE STUDY

- PIPETRN[1..5] → transaction counters for the corresponding pipes
- DEVADD[0..5] → configures the transfer speed for the device to which the corresponding pipe is communicating
- PHYSLEW → adjust the physical driver to host or function operation
- DPUSR0R → configures pull-up / pull-down resistors, reads status of overcurrent and VBUS inputs
- DPUSR1R → configures and reads status concerning deep software standby mode
- USBMC → enables / disables battery charging mode and regulator circuit
- USBBCCTRL0 → configures parameters for battery charging mode

10.4 – SOFTWARE STACK – CASE STUDY

- Example of USB Host stack for Renesas microcontroller hardware (part of SSP – Synergy Sofware Package):
- Uses a Mass Storage Module on top as class driver.
- Uses Thread X RTOS to manage the threads concerning USB components.



Application

Thread X®
RTOS

Framework

USBX Host Class Mass Storage
g_ux_host_class_storage

Framework

USBX Host Configuration
g_ux_host_0

Frameworks

USBX Port HCD on
sf_el_ux for USBFS

USBX Port HCD on
sf_el_ux for USBHS

USBX on
ux

HAL Drivers

TX Transfer
Driver
R_DMAC

TX Transfer
Driver
R_DTC

RX Transfer
Driver
R_DMAC

RX Transfer
Driver
R_DTC

TX Transfer
Driver
R_DMAC

TX Transfer
Driver
R_DTC

RX Transfer
Driver
R_DMAC

RX Transfer
Driver
R_DTC

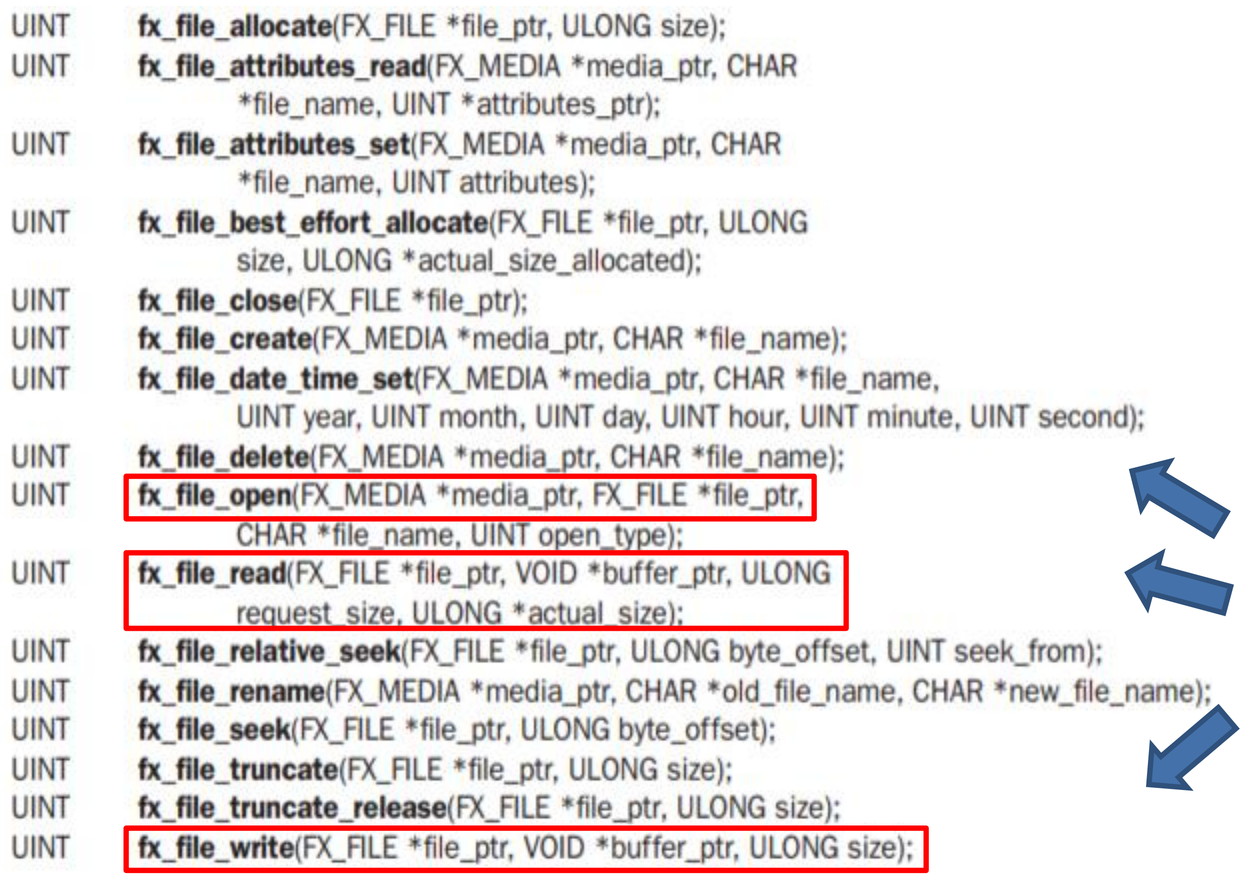
BSP

- (https://www.renesas.com/en-us/software/D6001255.html)

Source: Renesas Synergy USBX Host Class Mass Storage Module Guide
r11an0173eu0100-synergy-ux-host-class-mass-storage-mod-guide

10.4 – SOFTWARE STACK – CASE STUDY

- The application for the shown example uses the top-level API provided by the USBX Host Class Mass Storage component:
- This component instantiates a file manager (FileX) when a mass storage device (e. g. an USB memory) is inserted.
- The application uses the API provided by FileX to access the mass storage device contents → file open, close, read, write etc.



```
UINT    fx_file_allocate(FX_FILE *file_ptr, ULONG size);
UINT    fx_file_attributes_read(FX_MEDIA *media_ptr, CHAR
                *file_name, UINT *attributes_ptr);
UINT    fx_file_attributes_set(FX_MEDIA *media_ptr, CHAR
                *file_name, UINT attributes);
UINT    fx_file_best_effort_allocate(FX_FILE *file_ptr, ULONG
                size, ULONG *actual_size_allocated);
UINT    fx_file_close(FX_FILE *file_ptr);
UINT    fx_file_create(FX_MEDIA *media_ptr, CHAR *file_name);
UINT    fx_file_date_time_set(FX_MEDIA *media_ptr, CHAR *file_name,
                UINT year, UINT month, UINT day, UINT hour, UINT minute, UINT second);
UINT    fx_file_delete(FX_MEDIA *media_ptr, CHAR *file_name);
UINT    fx_file_open(FX_MEDIA *media_ptr, FX_FILE *file_ptr,
                CHAR *file_name, UINT open_type);
UINT    fx_file_read(FX_FILE *file_ptr, VOID *buffer_ptr, ULONG
                request_size, ULONG *actual_size);
UINT    fx_file_relative_seek(FX_FILE *file_ptr, ULONG byte_offset, UINT seek_from);
UINT    fx_file_rename(FX_MEDIA *media_ptr, CHAR *old_file_name, CHAR *new_file_name);
UINT    fx_file_seek(FX_FILE *file_ptr, ULONG byte_offset);
UINT    fx_file_truncate(FX_FILE *file_ptr, ULONG size);
UINT    fx_file_truncate_release(FX_FILE *file_ptr, ULONG size);
UINT    fx_file_write(FX_FILE *file_ptr, VOID *buffer_ptr, ULONG size);
```

Basic API functions

Source: FileX Services (https://rtos.com/wp-content/uploads/2017/10/EL-filex-programmers-guide.pdf)

11 – ETHERNET

- Introduction
- Block Diagram
- Registers
- SW Stack

11.1 – INTRODUCTION

Ethernet is a wired network technology used to interconnect devices in a Local Area Network (LAN). It has been standardized as IEEE802.3, comprising the physical and data link layers of the OSI model.

Main characteristics:

- Packet-based protocol,
- All messages are broadcast and processed by the nodes only if needed,
- Nodes can transmit at any time → Ethernet provides for automatic collision management (electrical or logical),
- Up to 10 Gbps (10GBASE-SR and further).

11.1 – INTRODUCTION

Ethernet standard has evolved across different versions:

- Ethernet (IEEE 802.3) standard → makes use of coaxial connections (named 10BASE2 and 10BASE5) to achieve bandwidths up to 10 Mbps;
- Fast Ethernet (IEEE 802.3u) → evolved from 10BASE-T (4-pair unshielded twisted pair) to 100BASE-TX and 100BASE-FX (fiber-optic cable) to achieve bandwidths up to 100 Mbps;
- Gigabit Ethernet (IEEE 802.3z) → variation of Fast Ethernet (1000BASE-T); that supports full duplex operation to provide higher data rates (up to 1 Gbps);
- 10 Gigabit Ethernet (IEEE 802.3ae) → entirely based on optical fiber (10GBASE-SR), up to 10 Gbps.

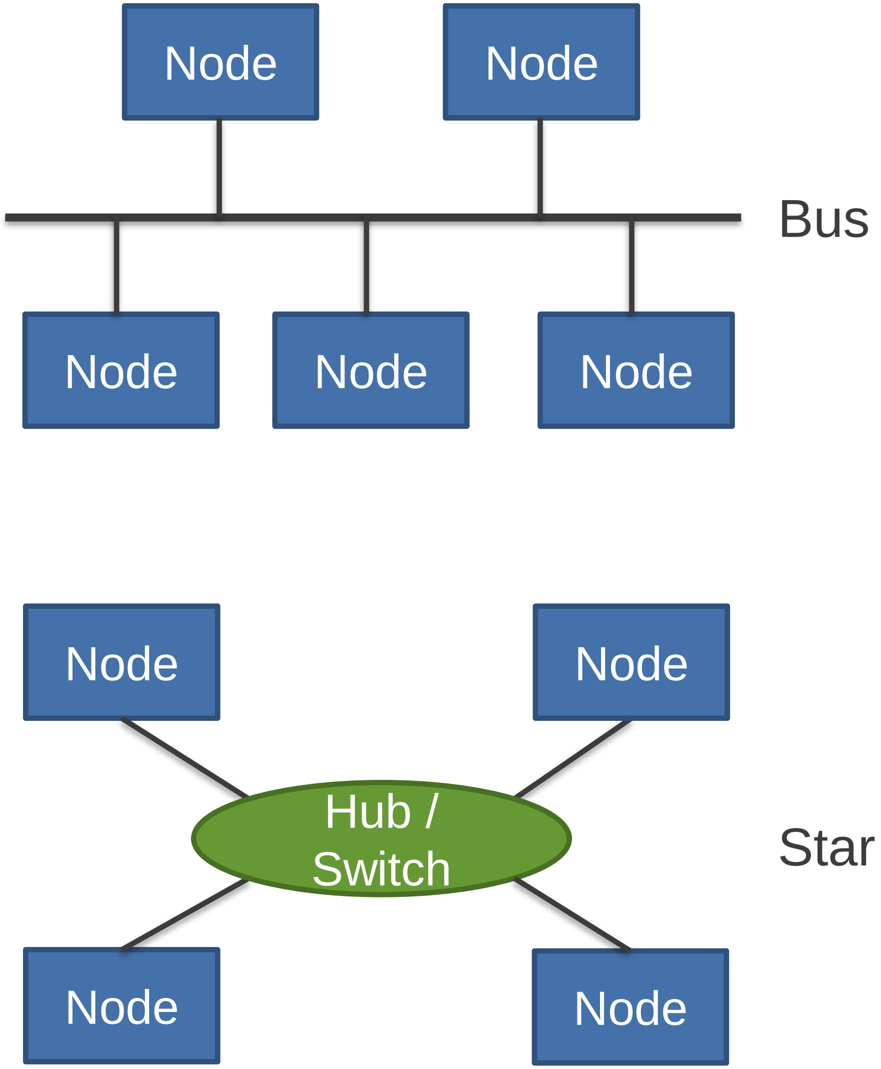
ETHERNET TOPOLOGY

Earlier Ethernet versions (10BASE2 and 10BASE5) used coaxial cables to interconnect nodes in a physical and logical bus topology.

Later versions (10BASE-T) rely on a physical star topology based on **hubs** → connected to nodes with twisted pair cabling.

Current versions (100BASE-TX and further) rely on a physical star topology based on **switches** → physically isolate the nodes so that a packet is delivered solely to its destination node → minimizes the network congestion due to packet collision.

Ethernet is always a bus topology from the logical point-of-view.



source: Authors

ETHERNET CONNECTION INFRASTRUCTURE

A Hub acts as a **repeater**:

- Data received by the hub from an Ethernet node is sent to all other Ethernet nodes connected to the hub.
- Therefore, multiple simultaneous transmissions are mixed and equally propagated to all connected nodes → possibility of collisions.

A Switch acts as a **filtered repeater**:

Destination address of every transmitted Ethernet packet (frame) is checked by the switch.

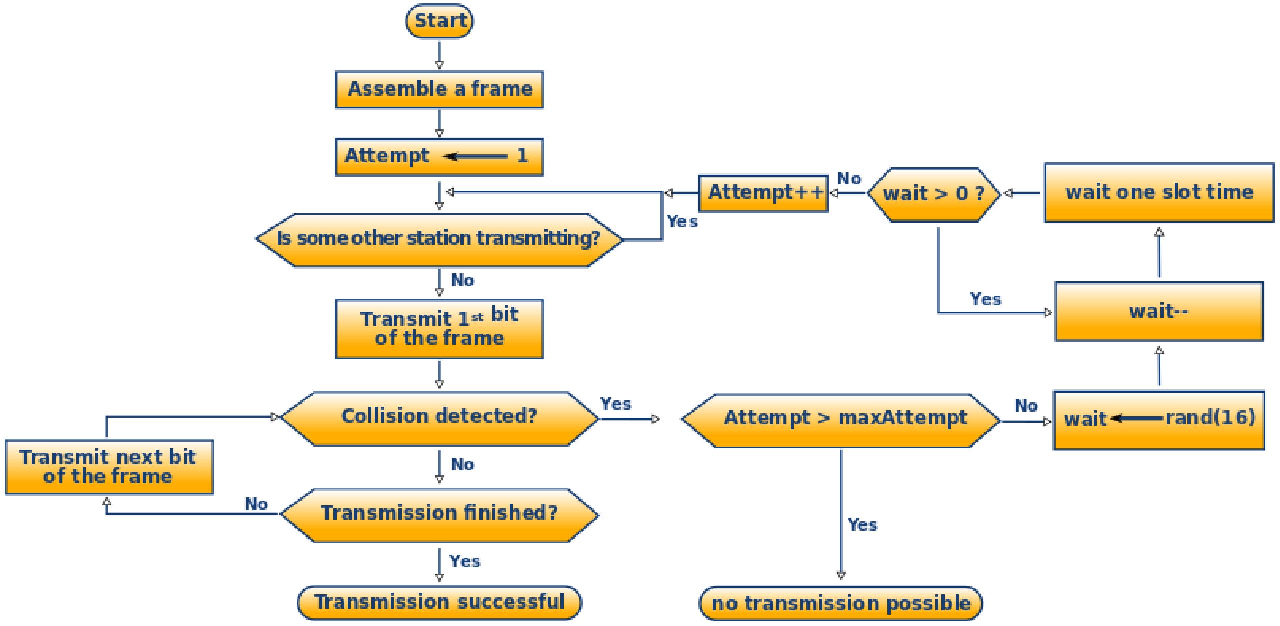
- The frame is forwarded only to the corresponding Ethernet node.
- This allows multiple simultaneous transmissions to succeed, provided that the pair of source-destination nodes for each of the transmissions is different.
- Packet collisions are avoided, as the switch is able to enqueue and serialize multiple frames addressed to the same destination node.

ETHERNET COLLISION MANAGEMENT

Because multiple Ethernet nodes share a logical bus, it is possible that more than one node try to transmit at the same time → collision.

To manage collisions, Ethernet uses the **Carrier Sense Multiple Access / Collision Detection (CSMA/CD)** protocol.

- The sender node starts transmitting a packet (frame) and uses carrier-sensing to detect if other nodes are trying to transmit at the same time.
- While no collision is detected, the sender node keeps on sending the frame bits until the end.
- If a collision is detected, a jam signal is sent to warn the other nodes about the collision, a retransmission counter is incremented, and the frame transmission is restarted after a random amount of time.
- If the retransmission counter reaches the maximum number of attempts, the transmission is aborted.



Source: https://upload.wikimedia.org/wikipedia/commons/3/37/CSMACD-Algorithm.svg

ETHERNET PACKET

An Ethernet frame encapsulates the data packet.

The frame includes addressing information (MAC) and error detection features.

- CRC checking is performed over all fields (except Preamble and SFD) and compared to FCS.

Dest addr defines special values for broadcast (all nodes receive and process the packet) and multicast (a group of nodes receives and processes the packet).

Field	Preamble	Start of Frame Delimiter	Dest MAC addr	Src MAC addr	Length / type	Data	Frame Check Seq
Bytes	8	1	6	6	2	46 to 1500	4

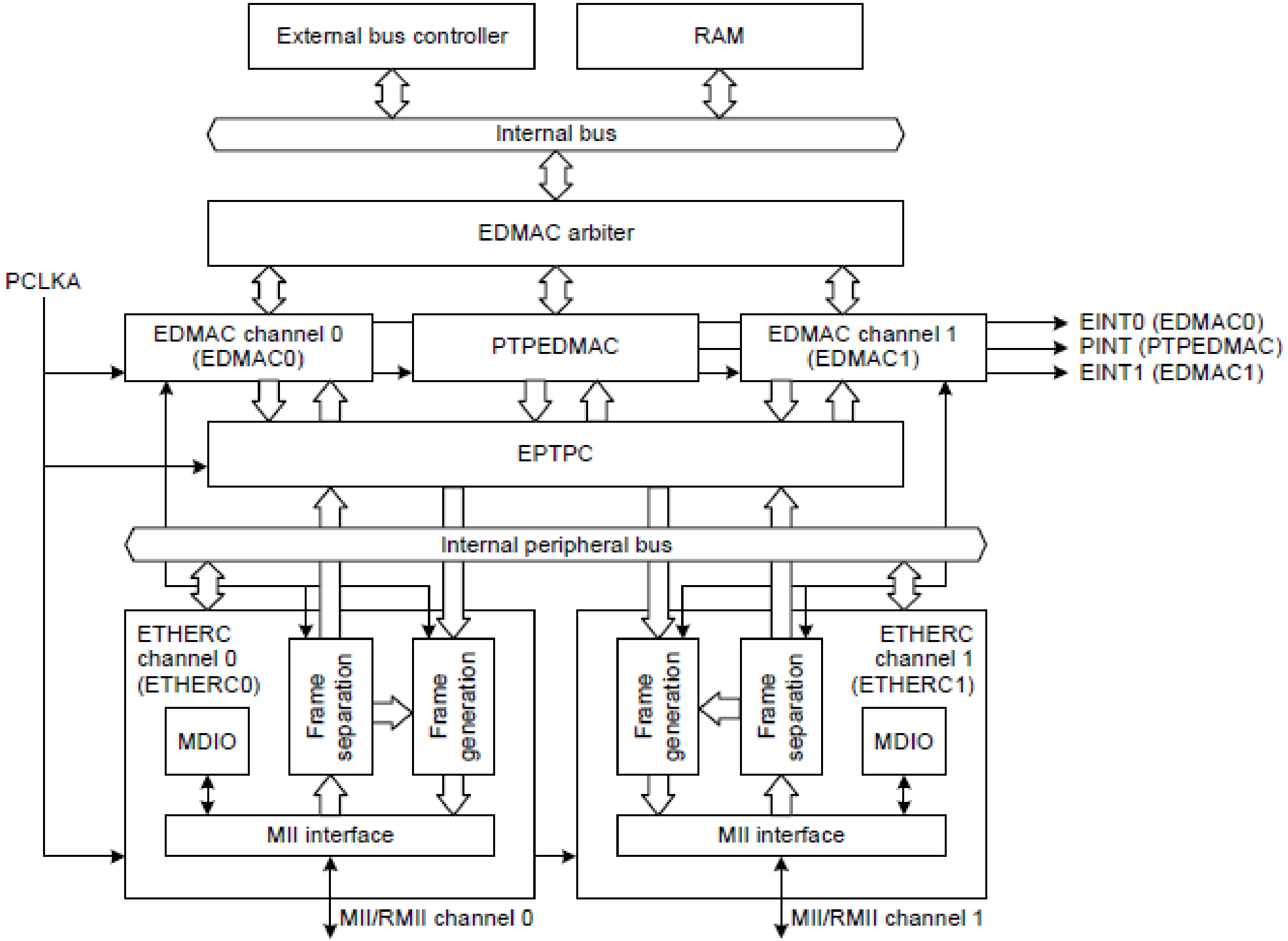
802.3 packet

11.2 – BLOCK DIAGRAM – CASE STUDY

Implementation for the Ethernet Controller Module of the R7FS7G27H3A01CFC Renesas ARM Cortex-M4 MCU.

- Two-channel controller → can operate two independent ETH interfaces.
- Depends on a DMA controller (EDMAC) to handle the TX and RX buffers without CPU intervention.

MII (Media Independent Interface) and RMII (Reduced MMI) are used to connect the ETH controller to the PHY hardware that implements the electrical interface.



Source: Renesas Synergy MCUs User’s Manual: Hardware

11.3 – REGISTERS – CASE STUDY

Implementation for the Ethernet Controller Module of the R7FS7G27H3A01CFC Renesas ARM Cortex-M4 MCU:

- ECMR → enable / disable, operation mode configuration
- RFLR → maximum frame length (between 1518 and 2048 bytes)
- ECSR → detection of line events (e. g. false carrier)
- ECSIPR → enable/disable line events interrupt
- PIR → access PHY registers
- PSR → status of PHY
- RDMLR → upper limit for random number generation
- IPGR → sets the interpacket gap (in bit times)

11.3 – REGISTERS – CASE STUDY

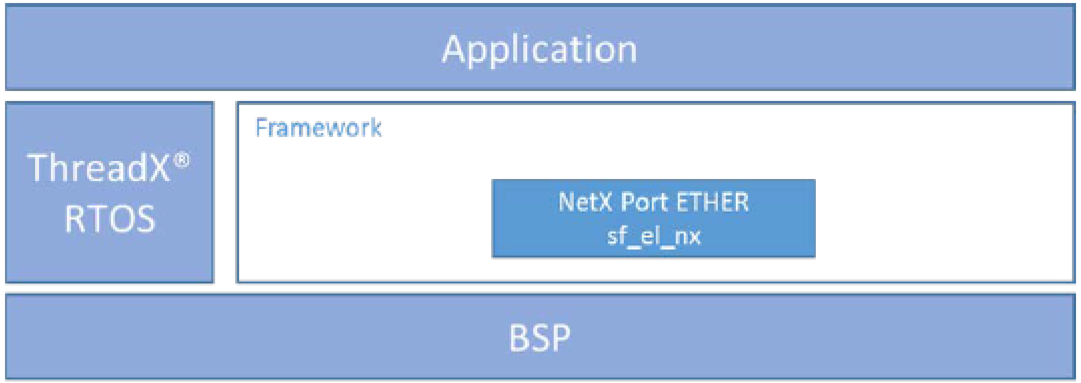
- APR → set pause time for an automatic PAUSE frame (used for flow control)
- MPR → set pause time for a manual PAUSE frame (used for flow control)
- RFCF → number of received PAUSE frames
- TPAUSER → max number of PAUSE frame retransmission
- TPAUSECR → PAUSE retransmit counter
- BCFRR → max number of received broadcast frames
- MAHR → upper bits of MAC address
- MALR → lower bits of MAC address
- TROCR → number of frames that failed to be retransmitted

11.3 – REGISTERS – CASE STUDY

- CDCR → number of late collisions detected
- LCCR → number of losses of carrier detected
- CNDCR → number of times a carrier is not detected
- CEFCR → number of received frames with CRC error
- FRECR → number of times a frame receive error has occurred
- TSFRCR → number of short frames received
- TLFRCR → number of long frames (longer than RFLR value) received
- RFCR → number of frames received with alignment error (not integral number of octets)
- MAFCR → number of multicast frames received

11.4 – SOFTWARE STACK – CASE STUDY

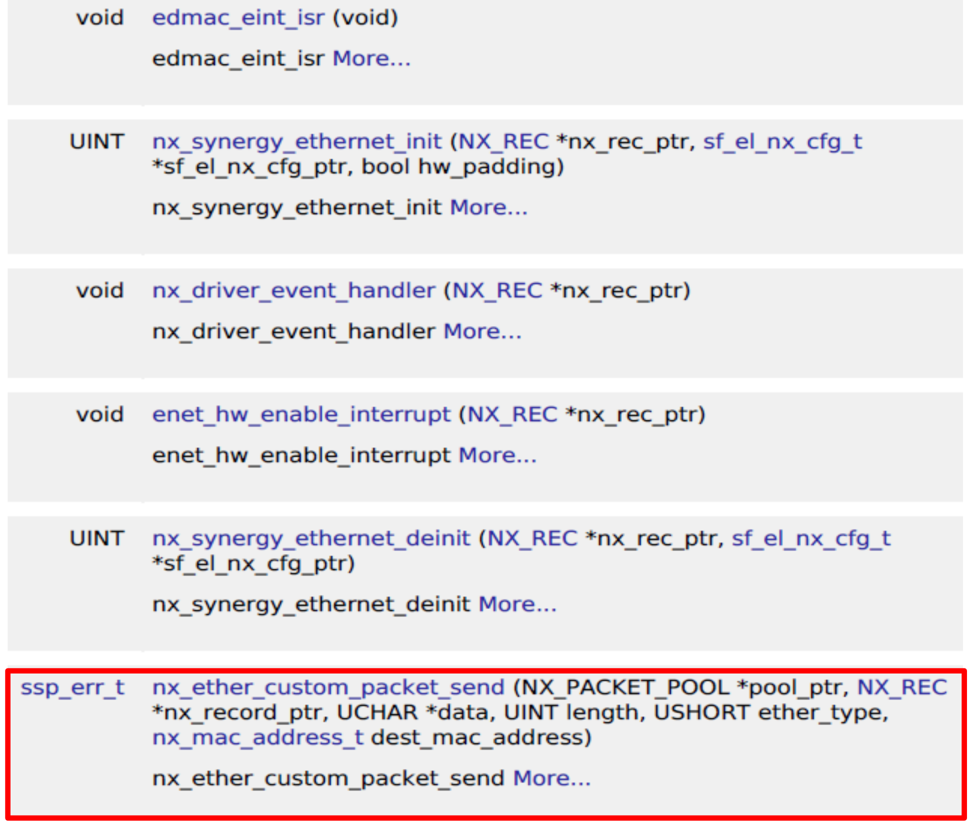
- Example of Ethernet stack for Renesas microcontroller hardware (part of the SSP → Synergy Software Package).
- The Ethernet layer depends on upper layers to manage the network protocols that generate or consume the data encapsulated into Ethernet packets → the “Application” layer.
- (https://www.renesas.com/en-us/software/D6001601.html)



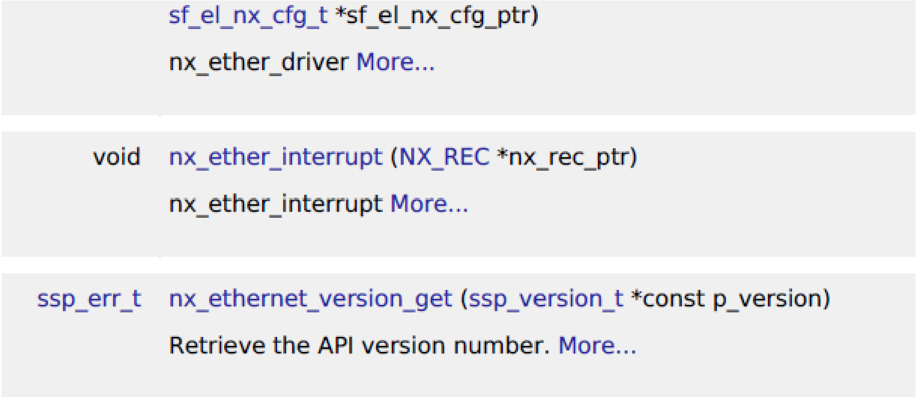
Source: Renesas Synergy NetX Port Module Guide
r11an0218eu0101-synergy-sf-el-nx-mod-guide

11.4 – SOFTWARE STACK – CASE STUDY

- Example of Ethernet API for Renesas microcontroller hardware □ part of NetX Framework for Renesas Synergy Software Package (SSP).



void	edmac_eint_isr (void) edmac_eint_isr More...
UINT	nx_synergy_ethernet_init (NX_REC *nx_rec_ptr, sf_el_nx_cfg_t *sf_el_nx_cfg_ptr, bool hw_padding) nx_synergy_ethernet_init More...
void	nx_driver_event_handler (NX_REC *nx_rec_ptr) nx_driver_event_handler More...
void	enet_hw_enable_interrupt (NX_REC *nx_rec_ptr) enet_hw_enable_interrupt More...
UINT	nx_synergy_ethernet_deinit (NX_REC *nx_rec_ptr, sf_el_nx_cfg_t *sf_el_nx_cfg_ptr) nx_synergy_ethernet_deinit More...
ssp_err_t	nx_ether_custom_packet_send (NX_PACKET_POOL *pool_ptr, NX_REC *nx_record_ptr, UCHAR *data, UINT length, USHORT ether_type, nx_mac_address_t dest_mac_address) nx_ether_custom_packet_send More...



	sf_el_nx_cfg_t *sf_el_nx_cfg_ptr) nx_ether_driver More...
void	nx_ether_interrupt (NX_REC *nx_rec_ptr) nx_ether_interrupt More...
ssp_err_t	nx_ethernet_version_get (ssp_version_t *const p_version) Retrieve the API version number. More...

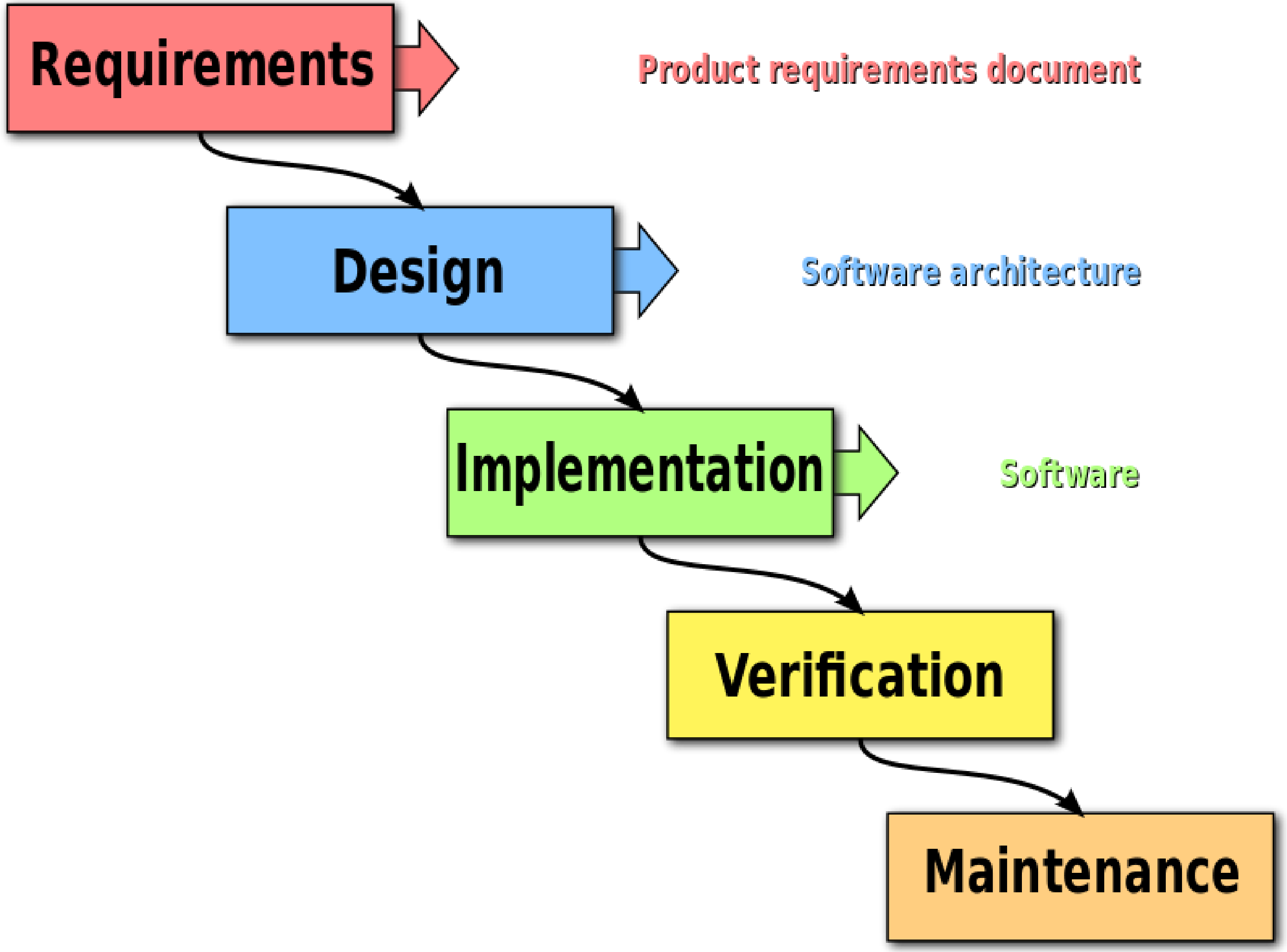
Basic comm API functions

Renesas Synergy Software Package v1.7.5 User's Manual
r11um0140eu0106-synergy-ssp-v175

12 – SOFTWARE DEVELOPMENT PROCESS

- Software Process Overview
- UML Class Diagram
- UML State Machine Diagram

WATERFALL PROCESS



Requirements

Product requirements document

Design

Software architecture

Implementation

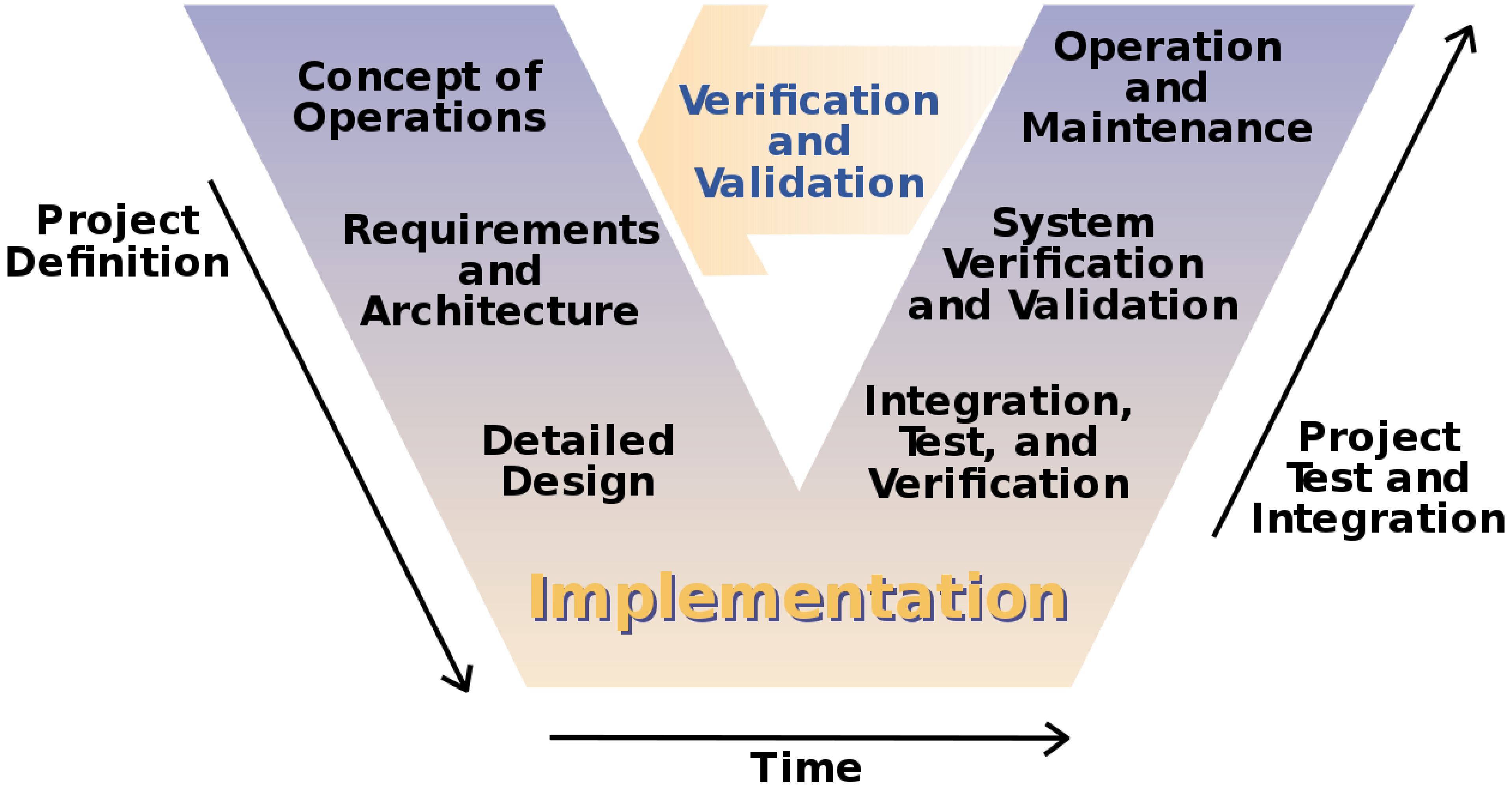
Software

Verification

Maintenance

SOFTWARE PROCESS

V-cycle


Concept of Operations
Requirements and Architecture
Detailed Design
Implementation
Integration, Test, and Verification
System Verification and Validation
Operation and Maintenance
Verification and Validation
Project Definition
Project Test and Integration
Time

UML

Unified Modeling Language

- Originally by Grady Booch, James Rumbaugh and Ivar Jacobson
- Based on the integration of several existing modeling languages
- OMG standard (www.omg.org) in 1998
- Language based on visual models
- Current version: 2.5 (March 2015)
- Non-proprietary language

MODELING

The basic elements of a structural model are:

- THINGS
- Interaction among THINGS

THINGS

Physical things:

- Coffee bean, processor, equipment , vehicle, planet ...

Logical things:

- Bank account, contract, variable stored in memory ...

INTERACTION AMONG THINGS

Examples:

processor is connected to memory

personA and personB are married

personX is responsible for bank_account_12345

CLASS / OBJECT

A cohesive entity that has attributes, behavior and (optionally) state.

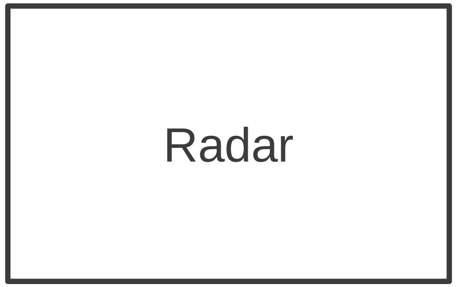
Characteristics of a Class / Object:

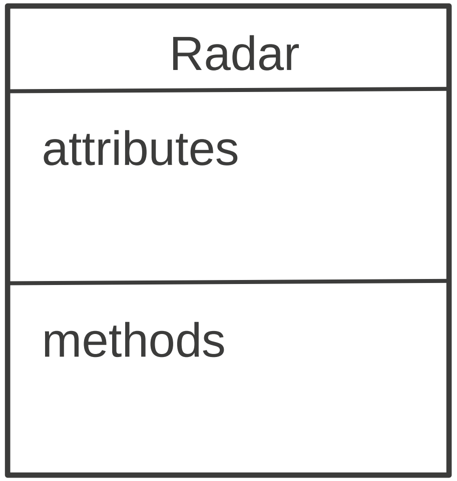
- Data - attributes
- Behavior - operations, methods, services, functions
- State - memory of its past
- Identity - unique identifier
- Responsibilities

CLASS / OBJECT IN PROGRAMMING LANGUAGES

Class	Object
example: a type in C	example: a variable in C
Compile time existence	runtime existence
	Ocupies memory
the design of an object	an instance of a class

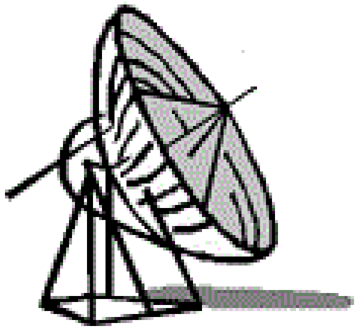
CLASS REPRESENTATION


Radar


Radar

attributes

methods

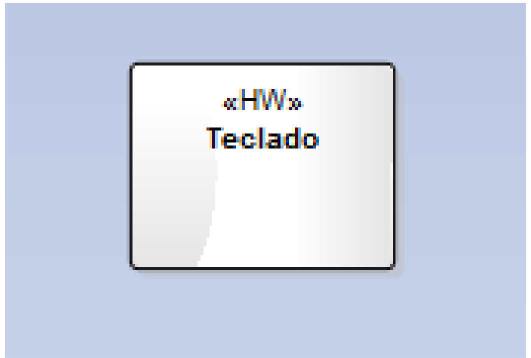


STEREOTYPE

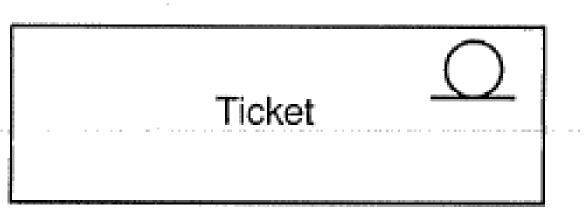
An additional (informal) form of classification.

Notation:

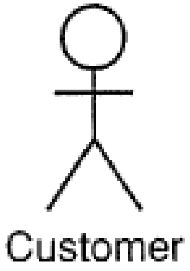
- text: <<stereotype_name>>



- icon



- class is represented by an icon

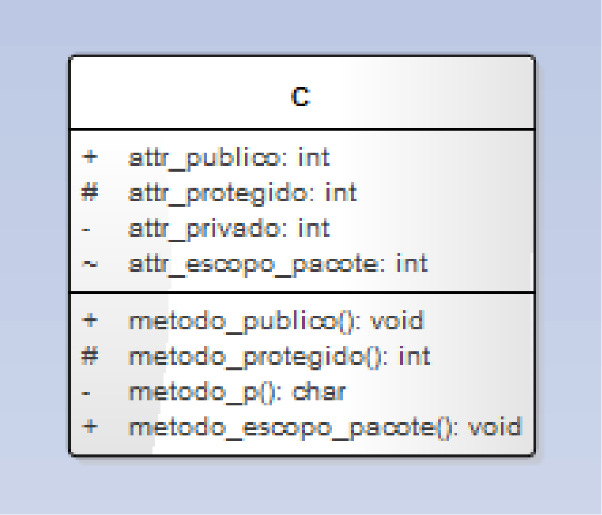


ATTRIBUTES / METHODS – VISIBILITY

A typical class representation has:

- class name (+ stereotype)
- attributes
- methods

The visibility of the methods and attributes is indicated by:



Symbol	Meaning	Visibility
+	public	accessible to all
#	protected	accessible by derived classes
-	private	accessible only by this class
~	package scope	accessible to the other classes in this package

INTERACTION AMONG THINGS

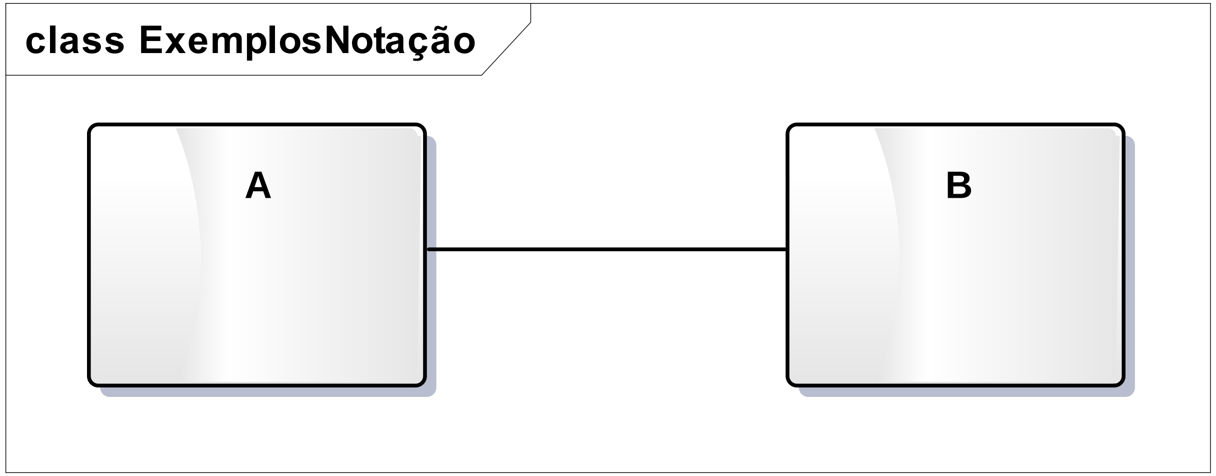
Relationships:

- Association
- Aggregation
- Composition
- Generalization
- Dependency

ASSOCIATION

A and B know of the existence of each other and may interact:

- access to public methods
- access to public attributes

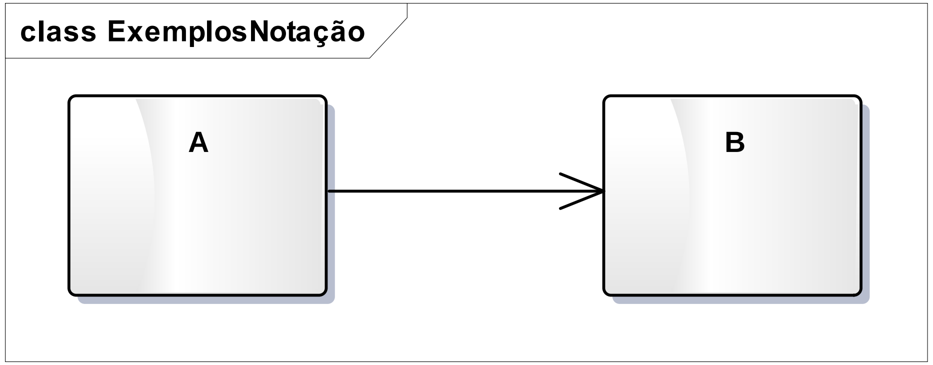


NAVIGABILITY

Who has access to whom?

A has access to B, but

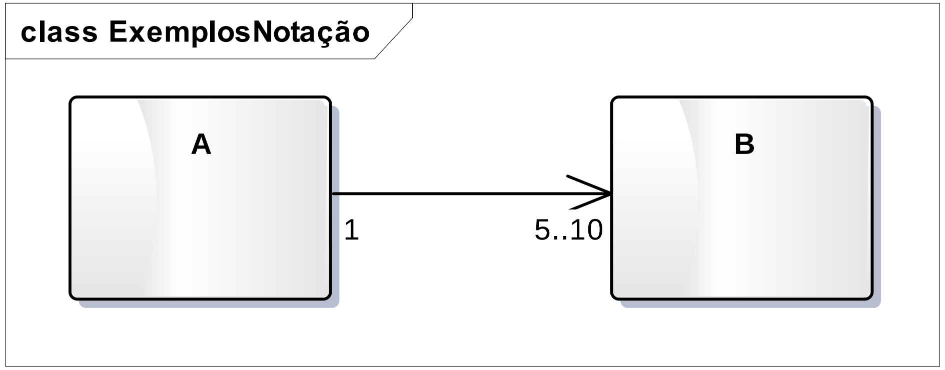
B does not have access to A



```
class A;
class B;

class A {
    B * pB;
    int a_attr;
};

class B {
 public:
    int b_attr;
};
```

MULTIPLICITY

How many objects of each class participate of this relationship?



1	exactly one
*	many (0 or more)
n	many (0 or more)
1..*	1 or more
3..20	from 3 to 20
4,6,8	4 or 6 or 8

```
class A;
class B;

class A {
    B * pB[10];
    int a_attr;
};

class B {
 public:
    int b_attr;
};
```

AGGREGATION

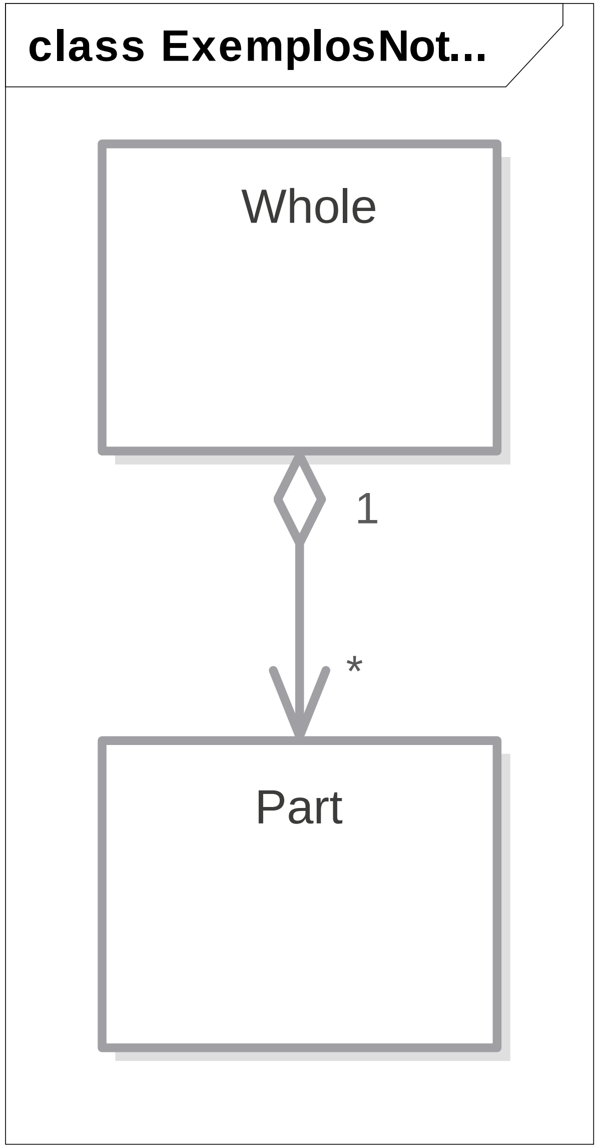
This relationship represents a weak part-whole or part-of.

The parts lifetime are different from the lifetime of the whole.

Read as:

Whole has a Part.

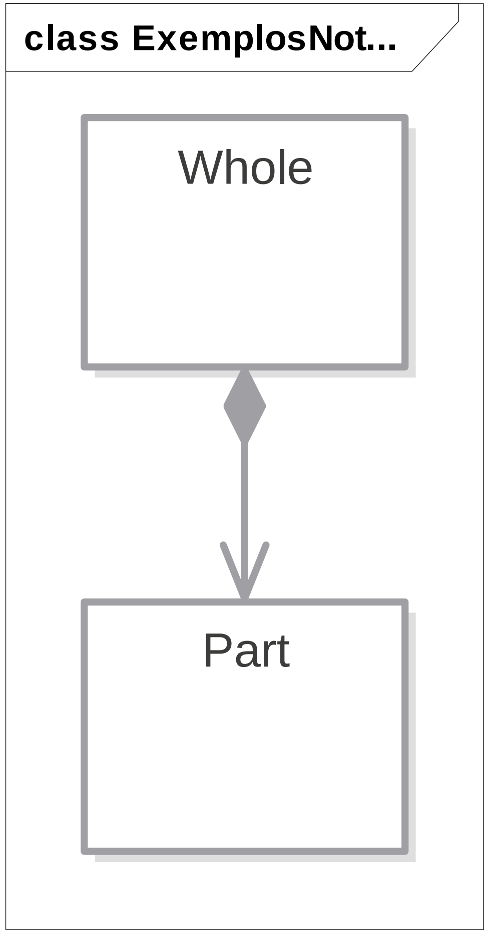
```
class Whole;
class Part;

class Whole {
  Part * pP[10];
  int a_attr;
};

class Part {
  public:
   int b_attr;
};
```

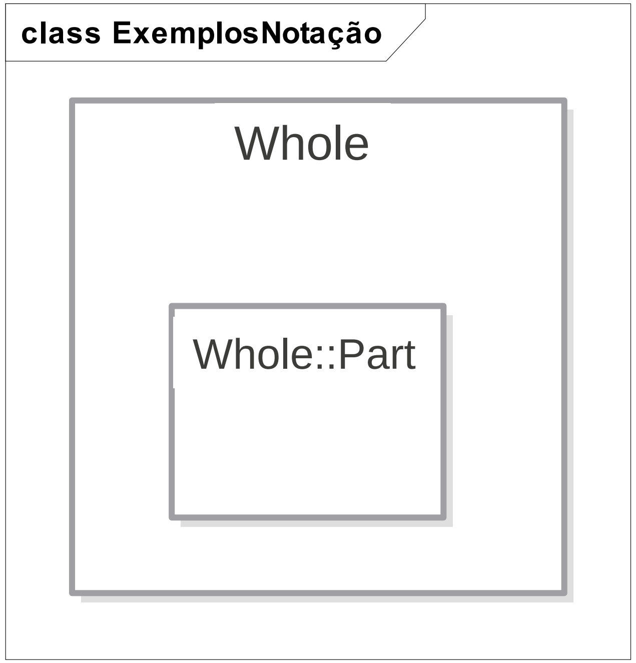


COMPOSITION

- Strong whole-part relationship.
- The whole and the parts form a single entity.
- Same lifespan for the whole and the parts.

```
class Part {
  public:
    int b_attr;
};

class Whole {
    Part obj;
    int a_attr;
};
```

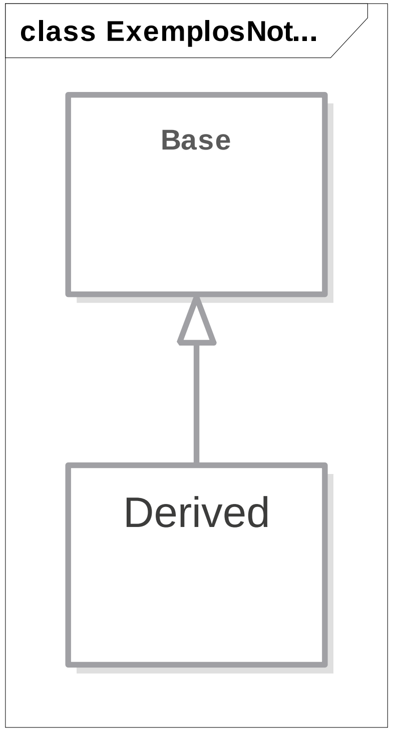




GENERALIZATION

- Represents inheritance.
- The derived class inherits all methods and attributes of the base class.
- Reading:

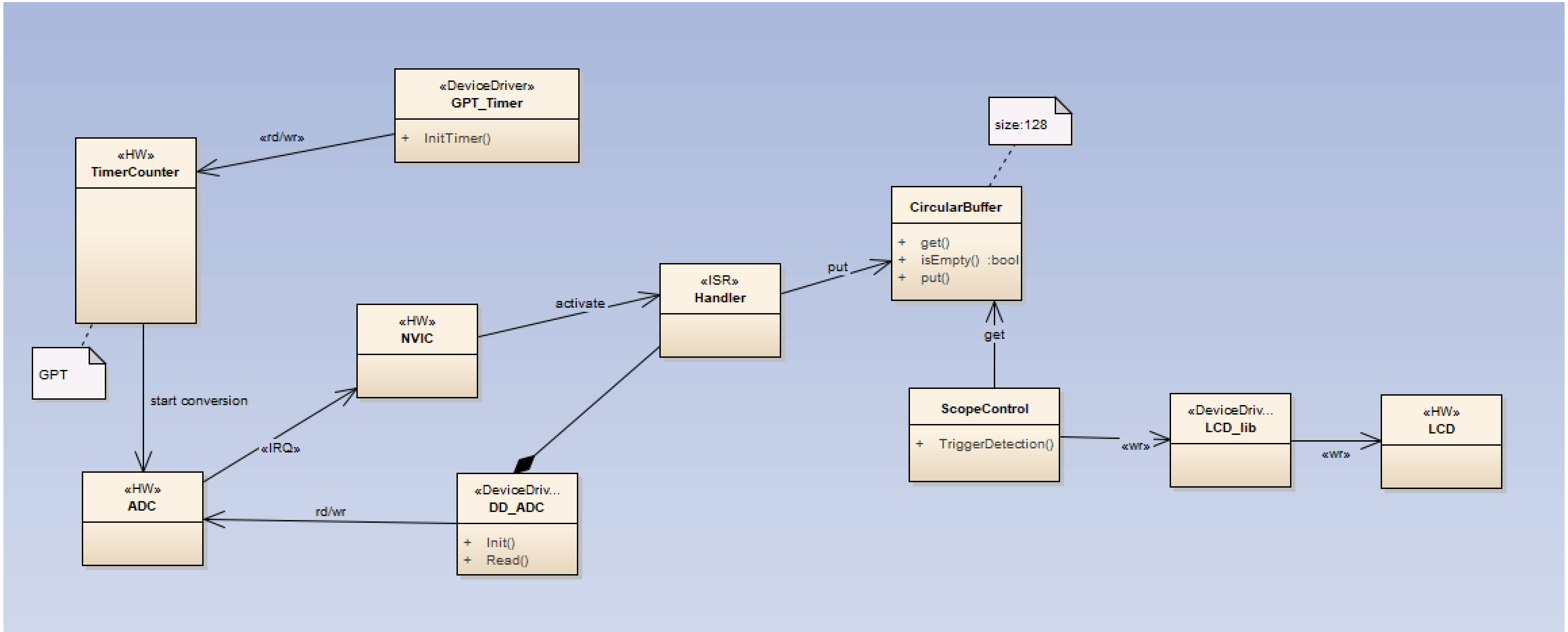
  Derived **is-a** Base.



```
class Base {
  protected:
   int base_attr;
   void base_meth(int);
};

class Derived : public Base {
   int der_attr;
};
```

USING UML TO DESIGN AN EMBEDDED SOLUTION

- Classes may represent: classes (e.g. C++), set of cohesive functions in C, hardware components, ...
- Since classes represent such a variety of things, the use of stereotypes is encouraged to inform the type: «HW», «ISR», «device driver»,...
- Associations also have several possible interpretations, from actual physical connections to pointers (e.g. in C). The use of stereotypes is also encouraged.
- A common mistake is to consider the navigation adornment as an indication of the direction of flow of data. It is not!



This is an example of a class diagram representing the components of a very simple digital scope.

Stereotypes are used to document what type of thing is being represented by each class: HW, Device Driver, ISR, ...

13 – CONCURRENT PROGRAMMING

- Tasks
  - Processes vs Threads
  - Context Switching
  - Scheduling
- Inter-task Communications and Synchronization
- Caveats

UNDERSTANDING CONCURRENCY

- Consider a single-person company. Suppose you could specify his activities by writing a program-like script. Wouldn’t it be very complex? Full of interleaved chores that would be hard to specify in a single script?

source: pixabay.com

UNDERSTANDING CONCURRENCY

Now, consider a simple embedded system with:

- 3 serial ports operating at 115 Kbps (aprox. 1 char every 87us);
- USB, requiring processing every 125 us for packets with about 1KBytes;
- IHM: touch screen plus LCD;
- activities implemented in SW:
  - A1: every 2 ms – process data from the touch screen;
  - A2: every 7 ms – process USB data;
  - A3: every 500 us – manage the USB protocol;
  - A4: every 100 ms – manage the menu system.

Wouldn’t it be very hard to program a single sequential code to execute all these tasks, even more if several possible interleavings could occur?

UNDERSTANDING CONCURRENCY

As embedded systems increase constantly in complexity and code size, this complexity can be better managed if a single program (responsible for all activities) could be divided into many smaller programs, each one responsible for a single activity. Each one of these small programs is called a **task**. The multiple tasks that compose a concurrent program execute concurrently and cooperate among them to achieve the desired functionality.

It is TEAM WORK!!

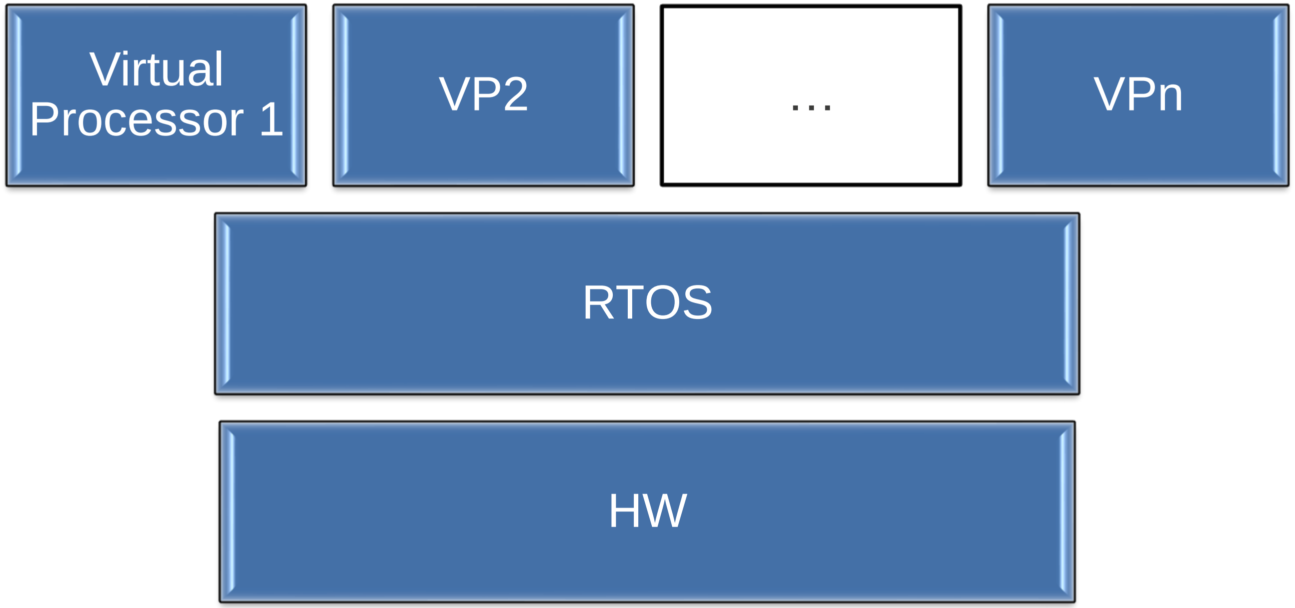
UNDERSTANDING CONCURRENCY

How can multiple tasks execute concurrently on a single processor?

Answer: each task will execute on a “virtual processor”. The combined performance of all virtual processors is about the same as the performance of the actual processor, as the virtual processor **share** the physical resources: processor, memory and peripherals.

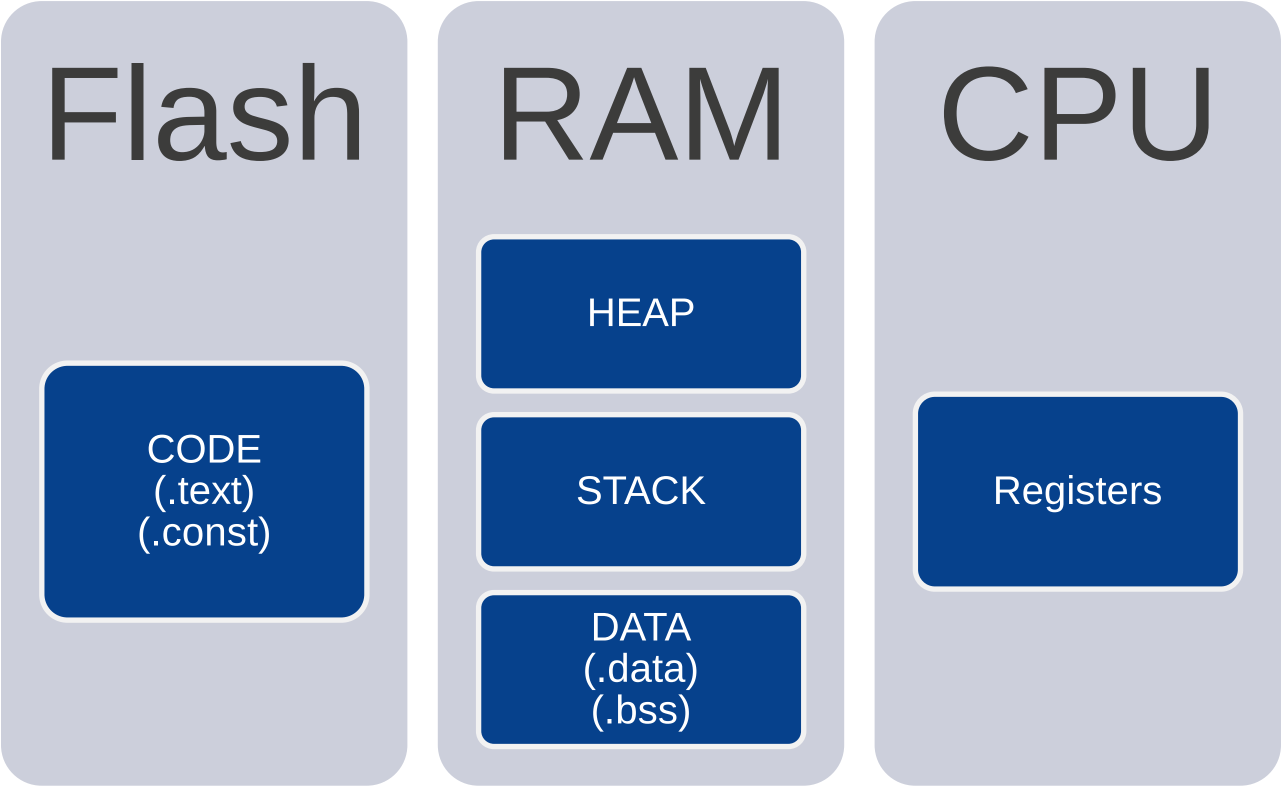
UNDERSTANDING CONCURRENCY

The sharing of the actual hardware (processor, memory, peripherals) is managed by an embedded operating system (or RTOS - Real-Time Operating System).



UNDERSTANDING CONCURRENCY

The storage regions of a single sequential program (e.g a C-program) are:



Flash

CODE
(.text)
(.const)

RAM

HEAP

STACK

DATA
(.data)
(.bss)

CPU

Registers

MULTITHREADING

For simplicity and to reduce the usage of computational resources, RTOS for MCUs typically rely on multithreading to implement concurrency.

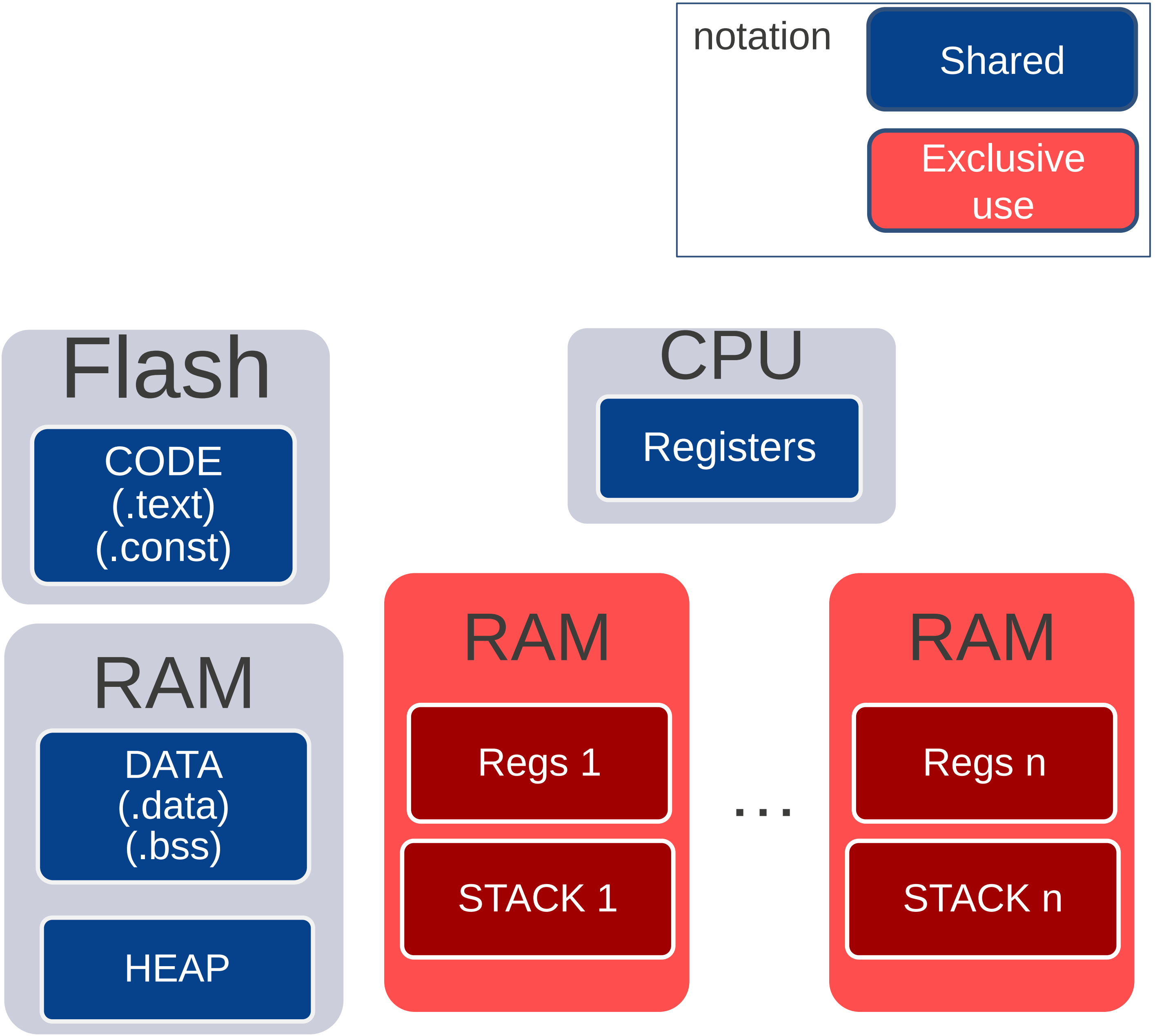
A **thread** is a program segment that executes concurrently to other threads (program segments). Hence, each thread is characterized by its own PC (program counter), its own set of processor registers and its own stack, while sharing the other memory regions with the other threads.

On MCUs, each task (abstract concept) is implement by one thread.

MULTITHREADING

Some sections of RAM (shown in red) are of **exclusive** use of a thread. These sections hold the stacks and a copy of the set of processor registers.

Other sections of Memory (shown in blue) are **shared** among all threads. While this sharing provides efficient access to shared data, it does not provide means of protection among threads.



MULTITHREADING

Differences between threads and processes:

- On processors with MMUs (Memory Management Units), it is possible to create an exclusive addressing space for each task. This type of implementation is called **process**. In a process, it is possible to host several threads, hence, there are single-threaded processes and multi-threaded processes.
- On processors without MMUs, all tasks share the available memory. This type of implementation is called **thread**.
- A **task** is a logical concept that can be implemented by a process or by a thread.

MULTITHREADING – CONCEPTS

Context Switch

How do multiple threads share a single processor?

- An RTOS manages the physical resources (processor, memory, peripherals).
- A context switch consists of saving the state of the processor when one thread is executing and restoring the state of another thread:

1. Task A is executing;
2. All CPU registers are saved onto the stack of Task A;
3. The RTOS executes and selects another task to execute: Task B. The criteria for selecting another task is defined by the scheduling policy;
4. The state of task B is restored from Task B stack onto the CPU registers. Execution proceeds on Task B's code and using Task B's stack.

MULTITHREADING – CONCEPTS

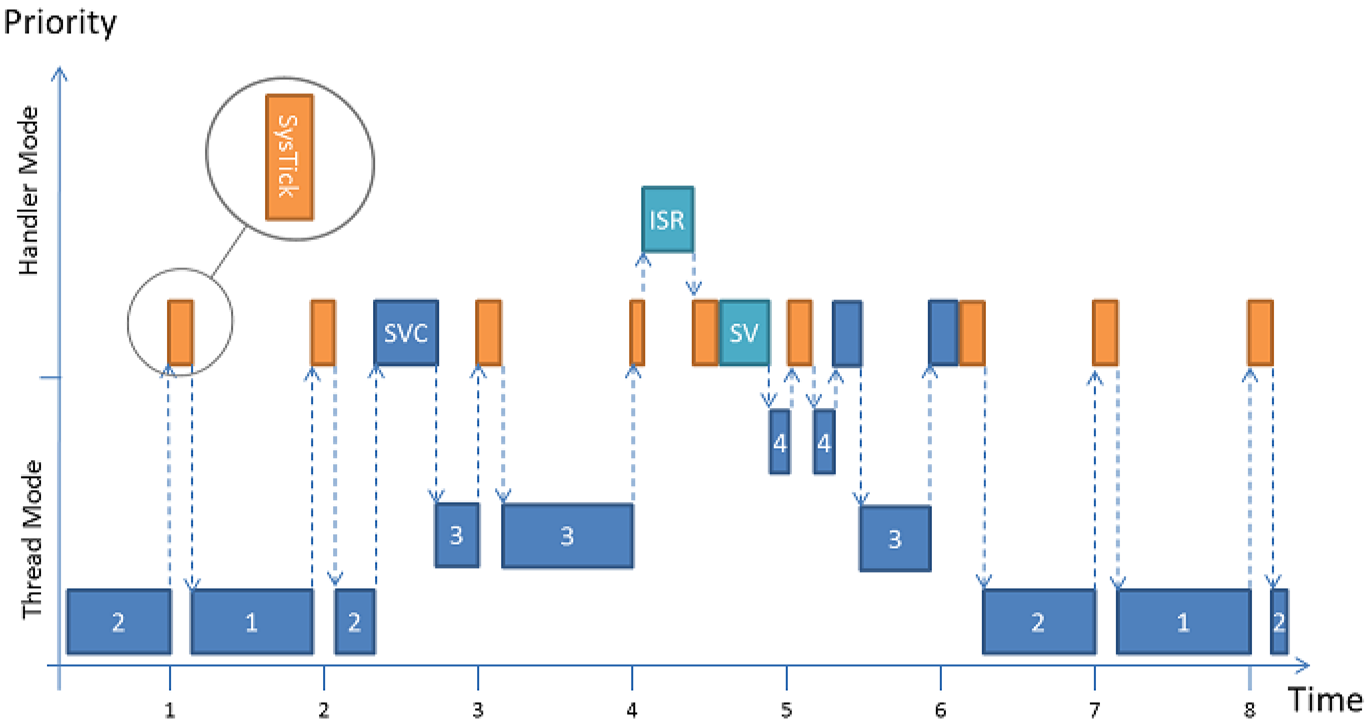
Preemptive vs Non-Preemptive RTOS

- A preemptive RTOS has control of the processor during all times. It releases the processor to a thread and at any time may get back the control of the processor to releases to another thread. This is the case when a higher priority thread becomes ready to run.
- When a non-preemptive RTOS releases control to a thread, it is unable to regain control of the processor until that thread, voluntarily, releases the processor back to the RTOS.

MULTITHREADING – CONCEPTS

Task Priority

- Each task is created with a defined priority level, which typically can be changed during execution. When a task of higher priority than the running task becomes ready, a priority-based preemptive scheduler will interrupt the running task and context switch to the higher priority task. Once this task releases the processor, the preempted task can resume its execution.



source: ARM, CMSIS-RTOS specs
http://www.keil.com/pack/doc/CMSIS_Dev/RTOS2/html/theory_of_operation.html

MULTITHREADING – CAVEATS

If all tasks in a concurrent program are independent, then the only control that is needed is the allocation of the processor to the tasks (**scheduling**).

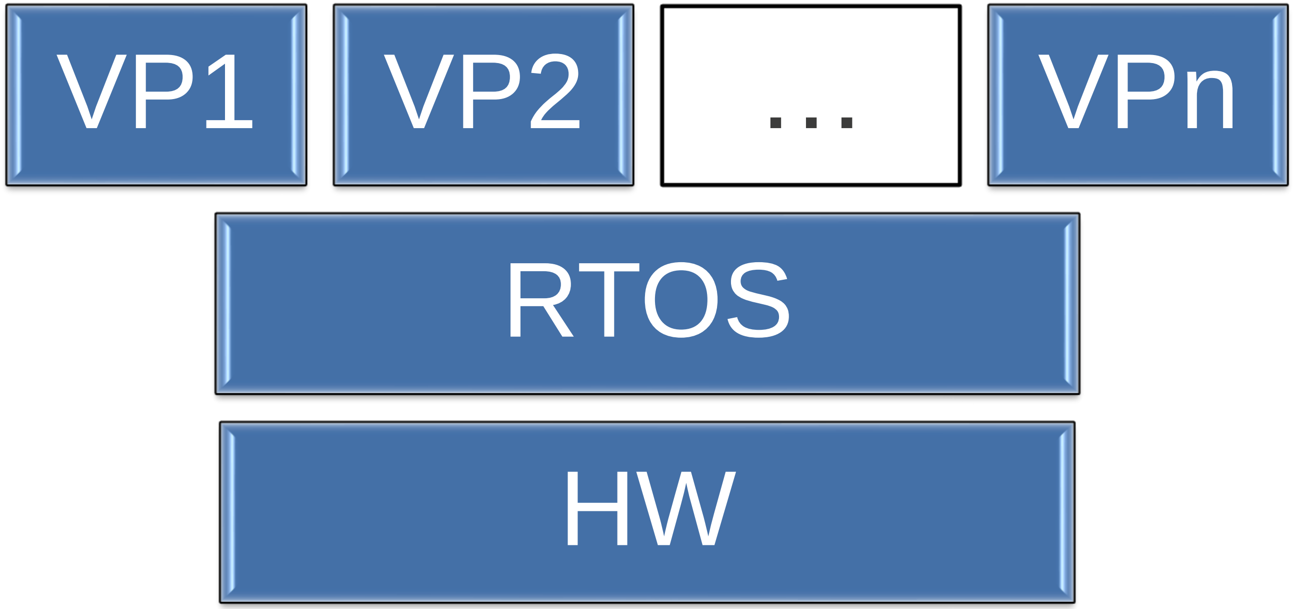
However, if tasks share resources (memory regions or peripherals) then an adequate control of resource sharing must be performed to guarantee **exclusive access to shared resources** and to avoid **deadlocks** and **priority inversion**.

14 – RTOS – REAL-TIME OPERATING SYSTEM

- Introduction
- Thread Management
- Inter-thread communication and synchronization
- Timing Services
- Memory Management

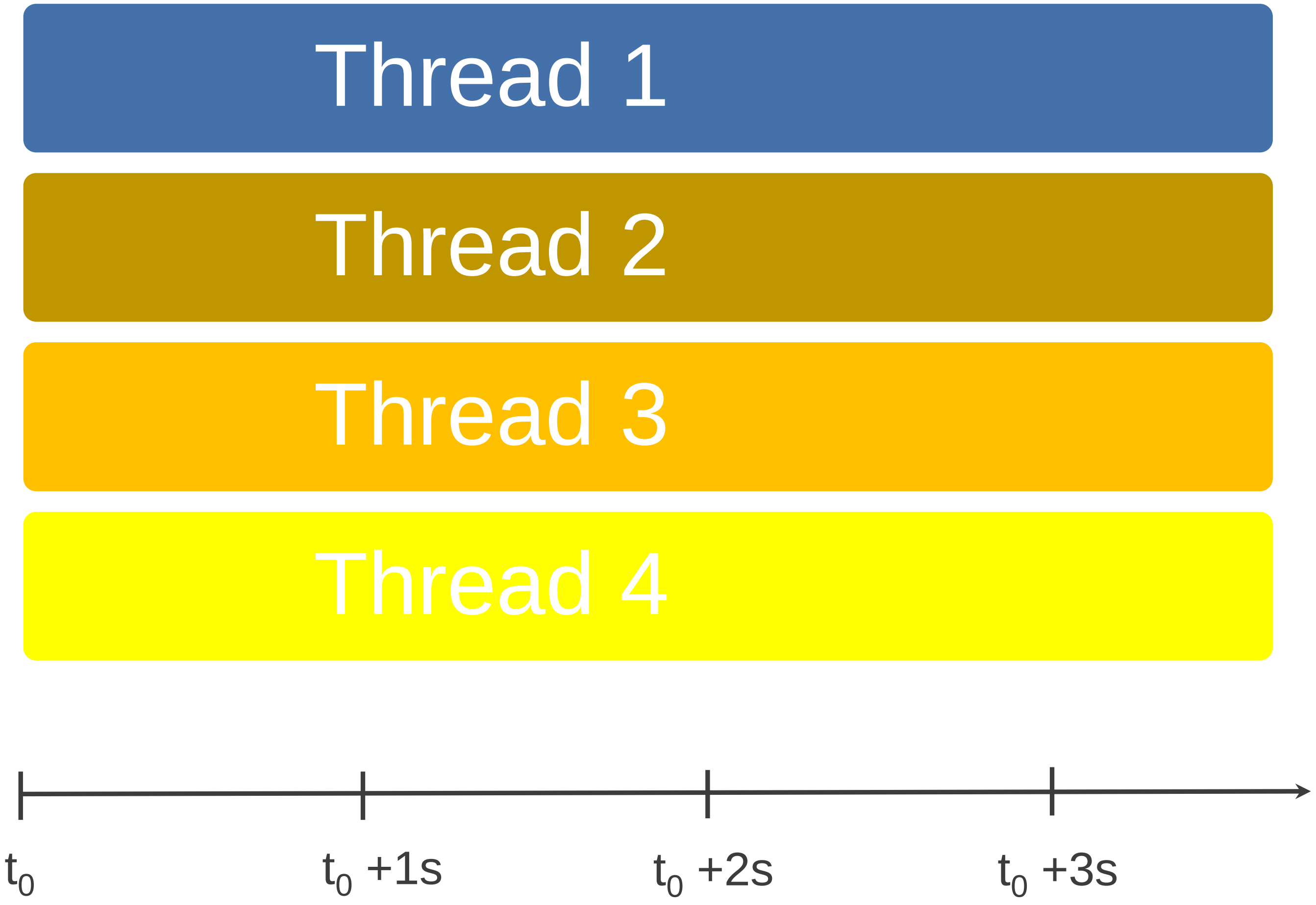
RTOS – INTRODUCTION

- An RTOS is a layer of Software between the hardware and the application software that allows the implementation of concurrent tasks implemented by multithreading.
- The RTOS manages the hardware, sharing resources among multiple “virtual processors”: VP1 to VPn so that each thread has the illusion of owning a processor and stack while sharing global data, heap, code area and peripherals.
- The RTOS implements sharing of the processor via context switching and provides several important functionalities for the threads: timing, inter-thread communication and synchronization, and thread management.



THREAD EXECUTION – LARGE GRAIN VIEW

Observing at a time scale of seconds, one has the impression that all threads execute in parallel.



Thread 1

Thread 2

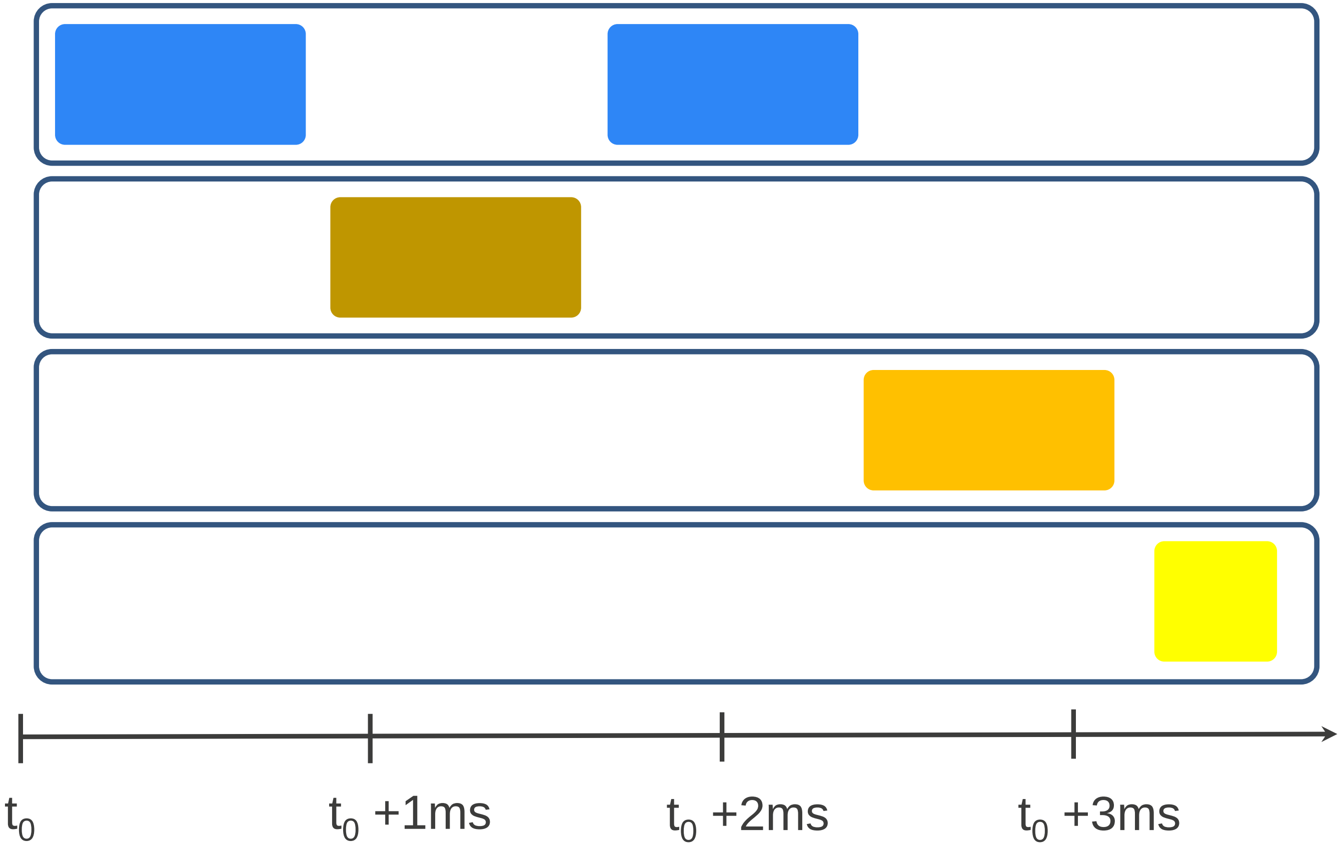
Thread 3

Thread 4

t_0 t_0 +1s t_0 +2s t_0 +3s

THREAD EXECUTION – FINE GRAIN VIEW

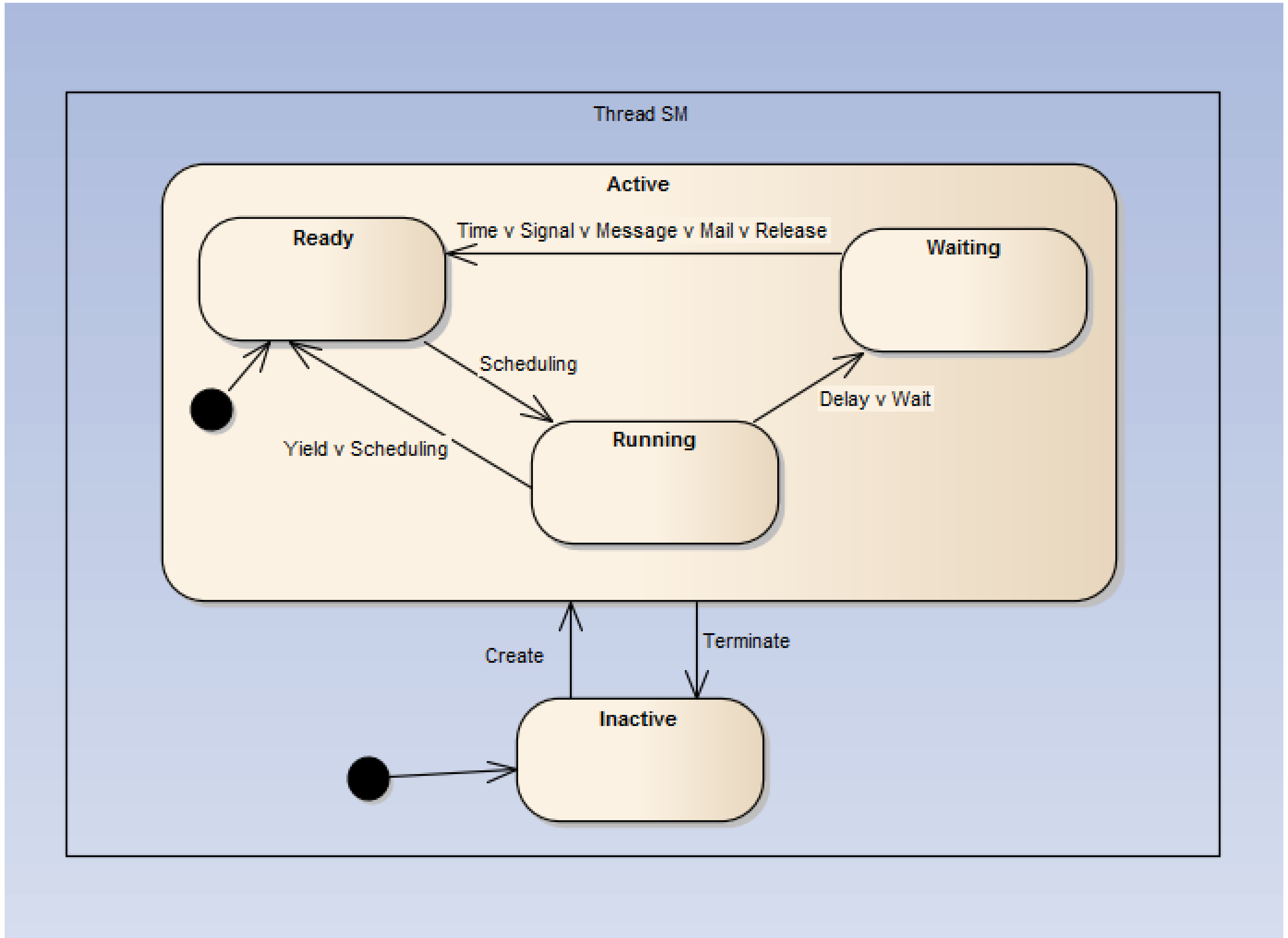
Observing on a fine grain scale (milliseconds) it is possible to realize that the tasks share the processor as the RTOS switches among them the utilization of the CPU.



TASK STATES

A typical state diagram of a task in a multithreaded environment:

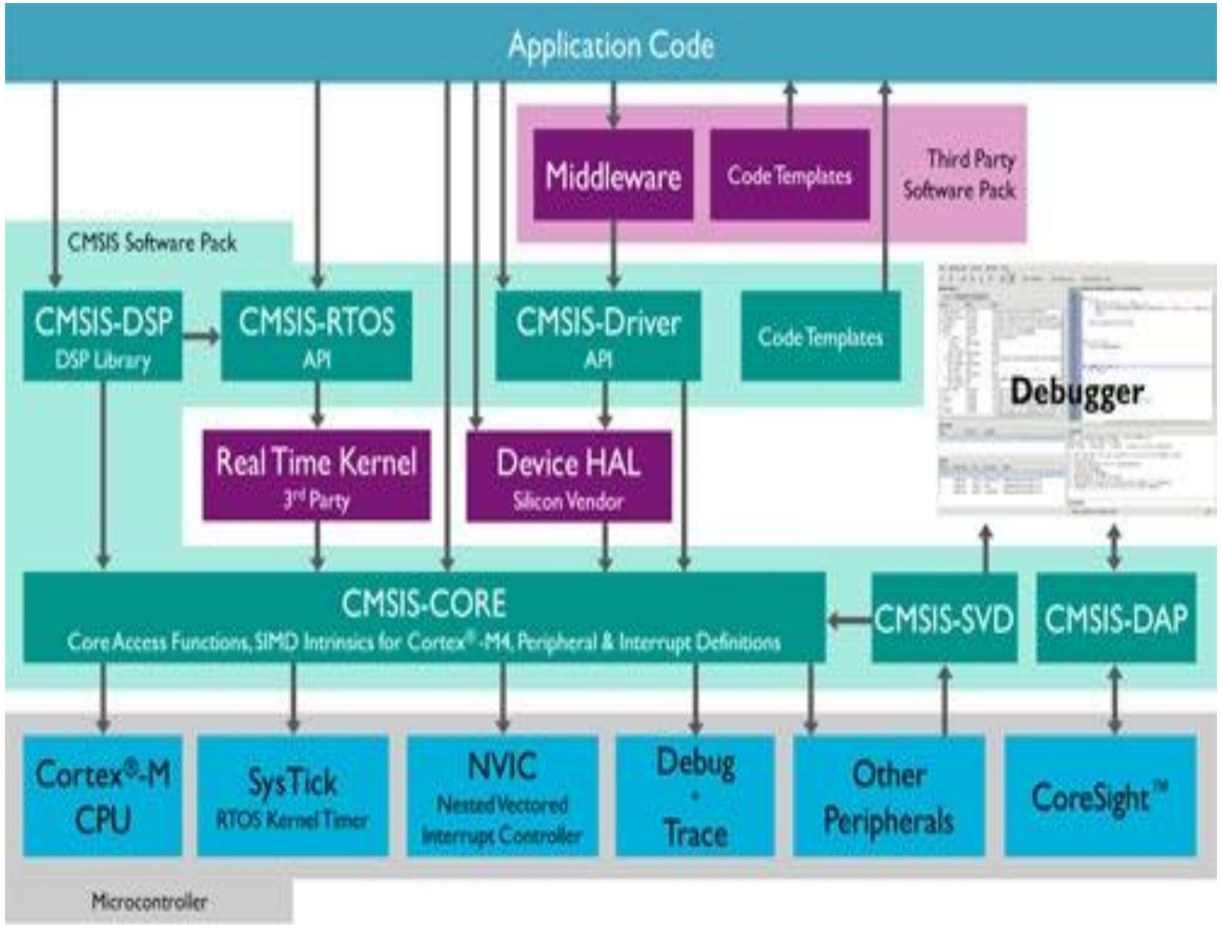
- **Ready:** tasks that is willing to use the processor and is expected to be scheduled.
- **Running:** the state of the task that is using the processor.
- **Waiting:** tasks whose execution is blocked due due to time, synchronization or communication.



CMSIS (CORTEX MICROCONTROLLER SOFTWARE INTERFACE STANDARD)

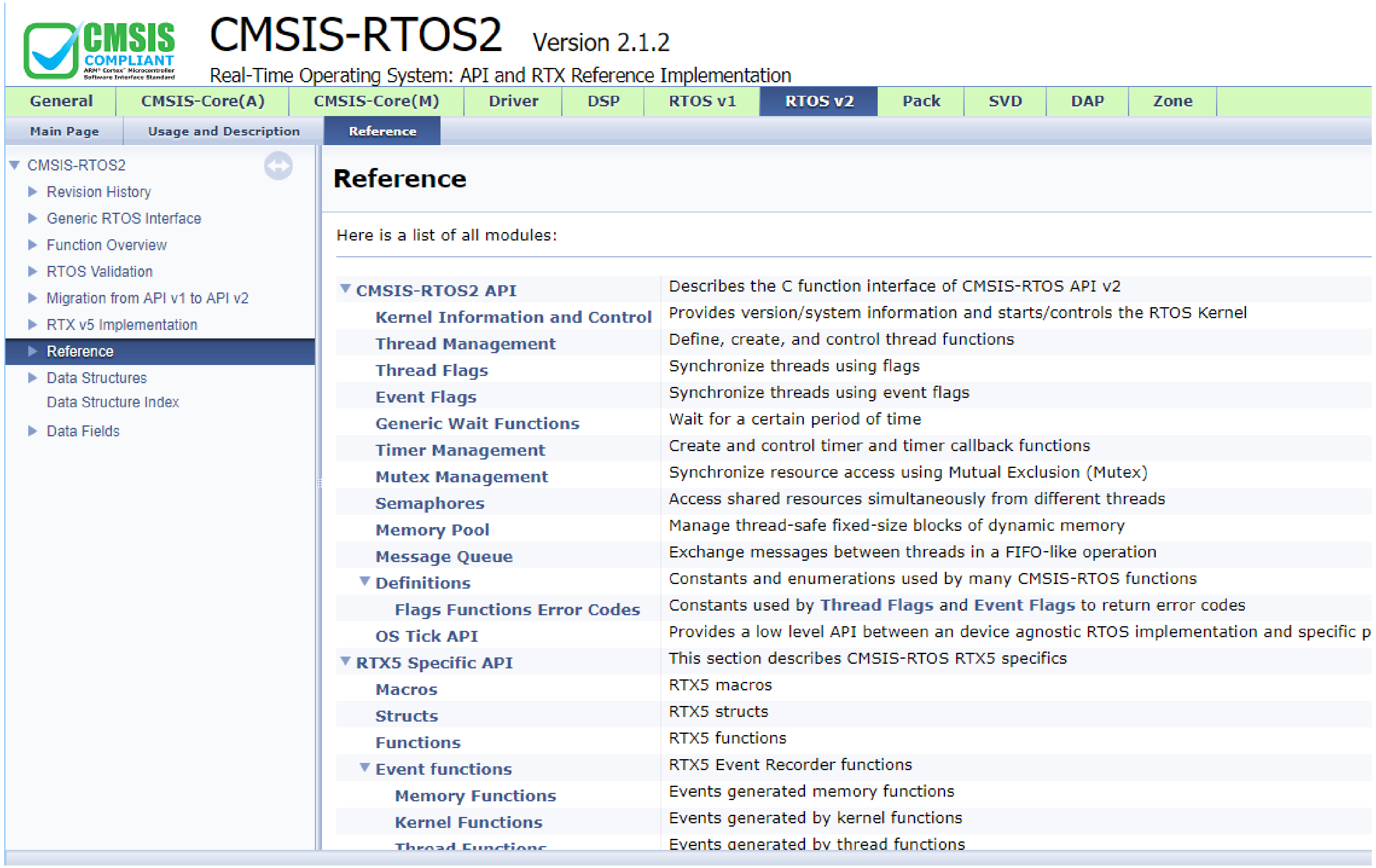
- v 1.0 (2008)
  - Drivers API
- v 2.0
  - DSP Lib
- v 3.0 (2012)
  - API RTOS
  - DAP (Debug)
- V 4.0 (2014)
  - CMSIS-Driver
  - CMSIS-Pack
- V 5.0 (2016) CMSIS-RTOS v2

As of March,2020, CMSIS is at version 5.6



CMSIS-RTOS DOCUMENTATION

Available online



CMSIS-RTOS2 Version 2.1.2

Real-Time Operating System: API and RTX Reference Implementation

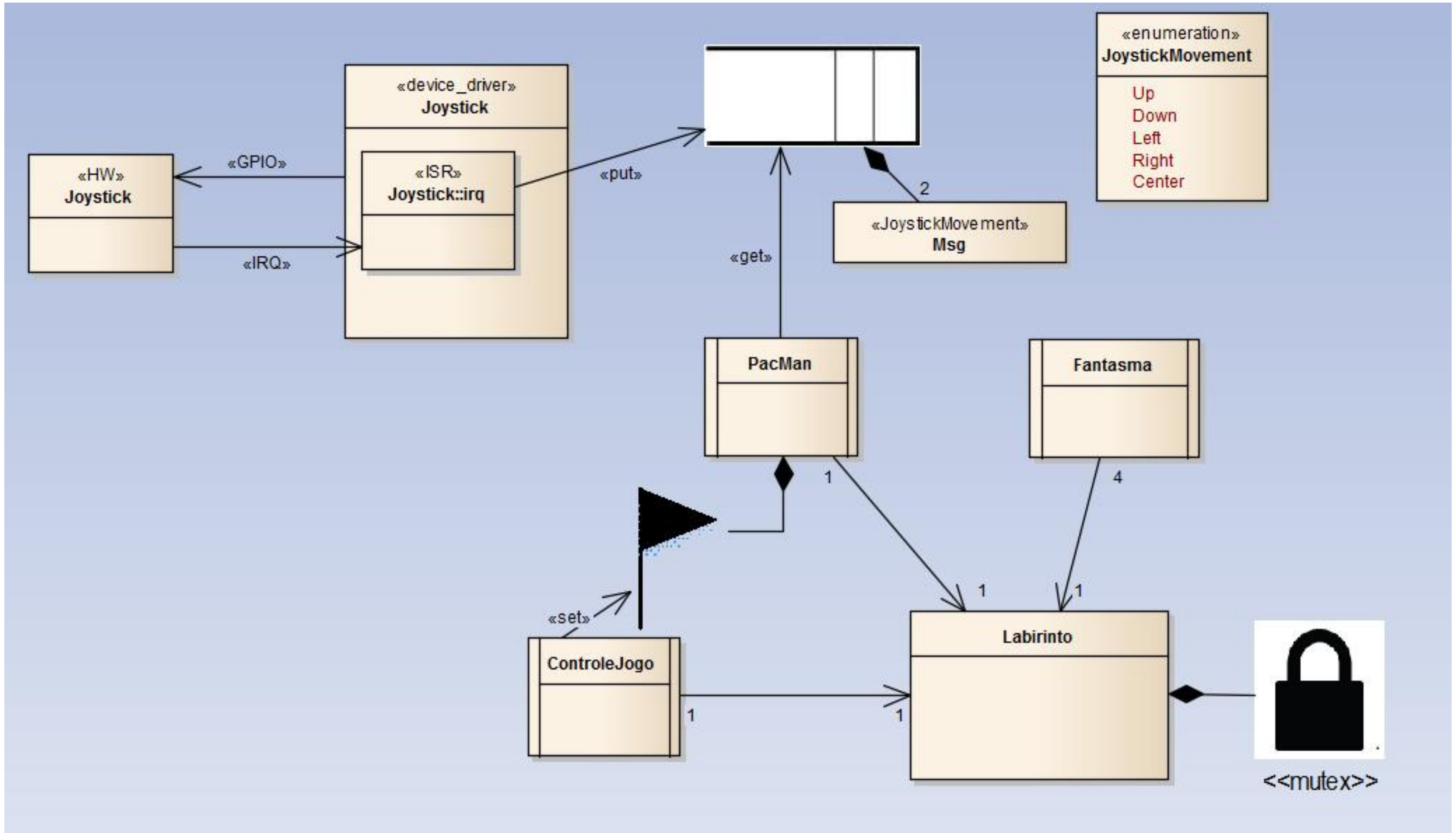
Reference

Here is a list of all modules:

▼ CMSIS-RTOS2 API	Describes the C function interface of CMSIS-RTOS API v2
Kernel Information and Control	Provides version/system information and starts/controls the RTOS Kernel
Thread Management	Define, create, and control thread functions
Thread Flags	Synchronize threads using flags
Event Flags	Synchronize threads using event flags
Generic Wait Functions	Wait for a certain period of time
Timer Management	Create and control timer and timer callback functions
Mutex Management	Synchronize resource access using Mutual Exclusion (Mutex)
Semaphores	Access shared resources simultaneously from different threads
Memory Pool	Manage thread-safe fixed-size blocks of dynamic memory
Message Queue	Exchange messages between threads in a FIFO-like operation
▼ Definitions	Constants and enumerations used by many CMSIS-RTOS functions
Flags Functions Error Codes	Constants used by **Thread Flags** and **Event Flags** to return error codes
OS Tick API	Provides a low level API between an device agnostic RTOS implementation and specific p
▼ RTX5 Specific API	This section describes CMSIS-RTOS RTX5 specifics
Macros	RTX5 macros
Structs	RTX5 structs
Functions	RTX5 functions
▼ Event functions	RTX5 Event Recorder functions
Memory Functions	Events generated memory functions
Kernel Functions	Events generated by kernel functions
Thread Functions	Events generated by thread functions

CMSIS code is available in GITHUB

PLANNING A MULTITHREADED APPLICATION



CMSIS-RTOS: OVERVIEW 1/4

Kernel Information and Control

- **osKernelInitialize** : Initialize the RTOS kernel.
- **osKernelStart** : Start the RTOS kernel.
- **osKernelRunning** : Query if the RTOS kernel is running.
- **osKernelSysTick** $: Get RTOS kernel system timer counter.
- **osKernelSysTickFrequency** $: RTOS kernel system timer frequency in Hz.
- **osKernelSysTickMicroSec** $: Convert microseconds value to RTOS kernel system timer value.

Thread Management

- **osThreadCreate** : Start execution of a thread function.
- **osThreadTerminate** : Stop execution of a thread function.
- **osThreadYield** : Pass execution to next ready thread function.
- **osThreadGetId** : Get the thread identifier to reference this thread.
- **osThreadSetPriority** : Change the execution priority of a thread function.
- **osThreadGetPriority** : Obtain the current execution priority of a thread function.

CMSIS-RTOS: OVERVIEW 2/4

Generic Wait Functions

- **osDelay** : Wait for a specified time.
- **osWait** $: Wait for any event of the type Signal, Message, or Mail.

Timer Management $

- **osTimerCreate** : Define attributes of the timer callback function.
- **osTimerStart** : Start or restart the timer with a time value.
- **osTimerStop** : Stop the timer.
- **osTimerDelete** : Delete a timer.

CMSIS-RTOS: OVERVIEW 3/4

Signal Management

- **osSignalSet** : Set signal flags of a thread.
- **osSignalClear** : Reset signal flags of a thread.
- **osSignalWait** : Suspend execution until specific signal flags are set.

Mutex Management $

- **osMutexCreate** : Define and initialize a mutex.
- **osMutexWait** : Obtain a mutex or Wait until it becomes available.
- **osMutexRelease** : Release a mutex.
- **osMutexDelete** : Delete a mutex.

Semaphore Management $

- **osSemaphoreCreate** : Define and initialize a semaphore.
- **osSemaphoreWait** : Obtain a semaphore token or Wait until it becomes available.
- **osSemaphoreRelease** : Release a semaphore token.
- **osSemaphoreDelete** : Delete a semaphore.

CMSIS-RTOS: OVERVIEW 4/4

Memory Pool Management $

osPoolCreate : Define and initialize a fix-size memory pool.

osPoolAlloc : Allocate a memory block.

osPoolCAlloc : Allocate a memory block and zero-set this block.

osPoolFree : Return a memory block to the memory pool.

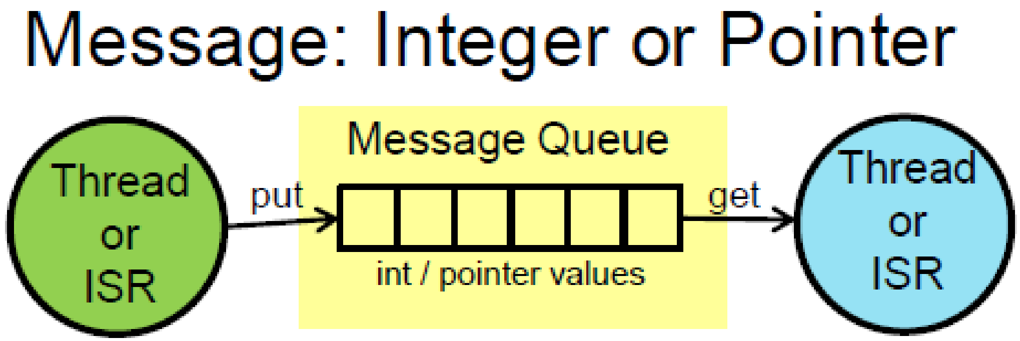
Message Queue Management $

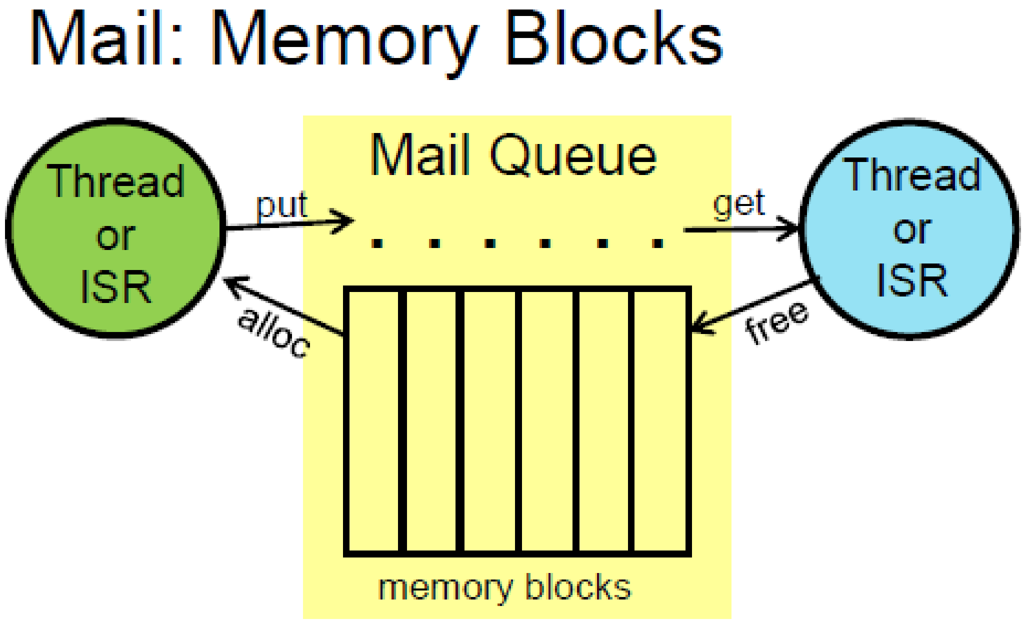
- osMessageCreate : Define and initialize a message queue.
- osMessagePut : Put a message into a message queue.
- osMessageGet : Get a message or suspend thread execution until message arrives.

Mail Queue Management $

- osMailCreate : Define and initialize a mail queue with fix-size memory blocks.
- osMailAlloc : Allocate a memory block.
- osMailCAlloc : Allocate a memory block and zero-set this block.
- osMailPut : Put a memory block into a mail queue.
- osMailGet : Get a mail or suspend thread execution until mail arrives.
- osMailFree : Return a memory block to the mail queue.

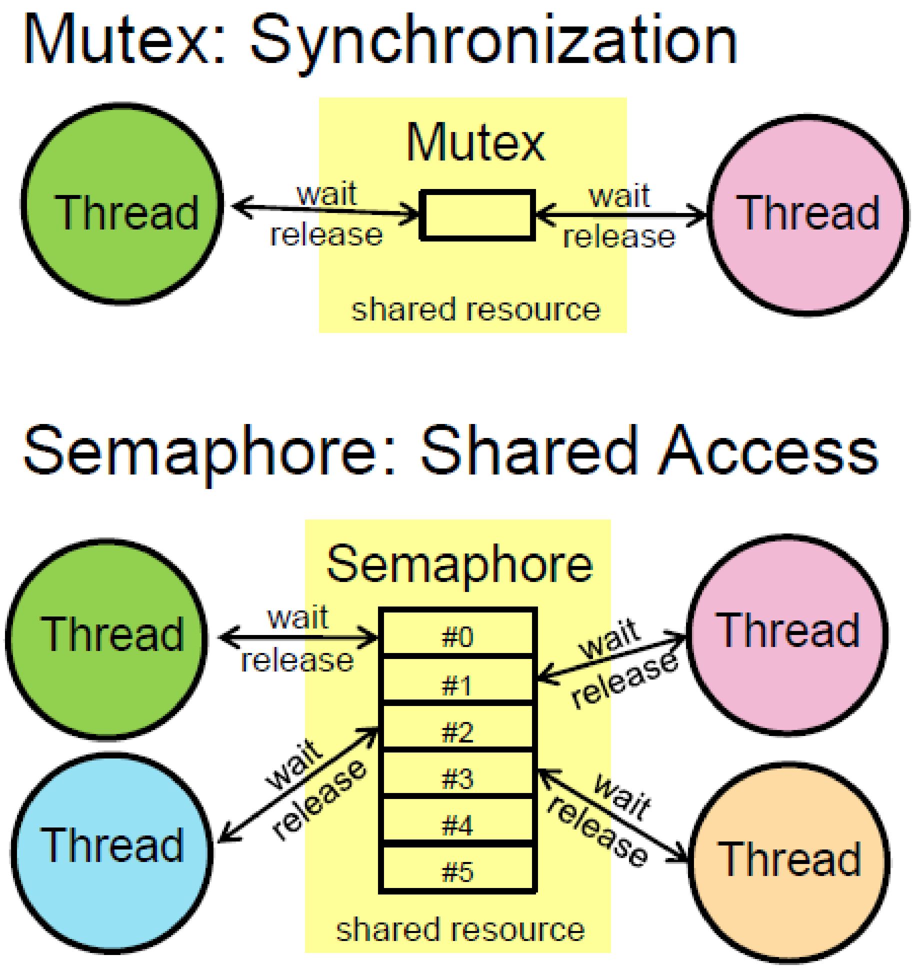
CMSIS-RTOS – INTER-TASK COMMUNICATIONS





Source: ARM, CMSIS documentation

CMSIS-RTOS – INTER-TASK COMMUNICATIONS



Source: ARM, CMSIS documentation

LAB1 – SYNERGY PLATFORM

Objectives:

- Execute the e2_studio and SSP installation procedures.
- Setup the development environment for the S7G2 kit.
- Run a sample program to verify the installation.
- Overview of e2_studio.
- Overview of SSP.

LAB1 – SYNERGY PLATFORM

Activities:

1. Create an account on Renesas, Install e2_studio (including SSP and tools);
2. Setup the SK-S7G2 board;

   Run a sample program to verify ISDE setup;
3. Overview of the Synergy Platform;
4. Overview of the e2 studio ISDE;
5. Overview of the SSP (Synergy Software Package).

LAB1 – SYNERGY PLATFORM

Activity 1 – Create an account on Renesas, Install e2_studio (including SSP and tools)

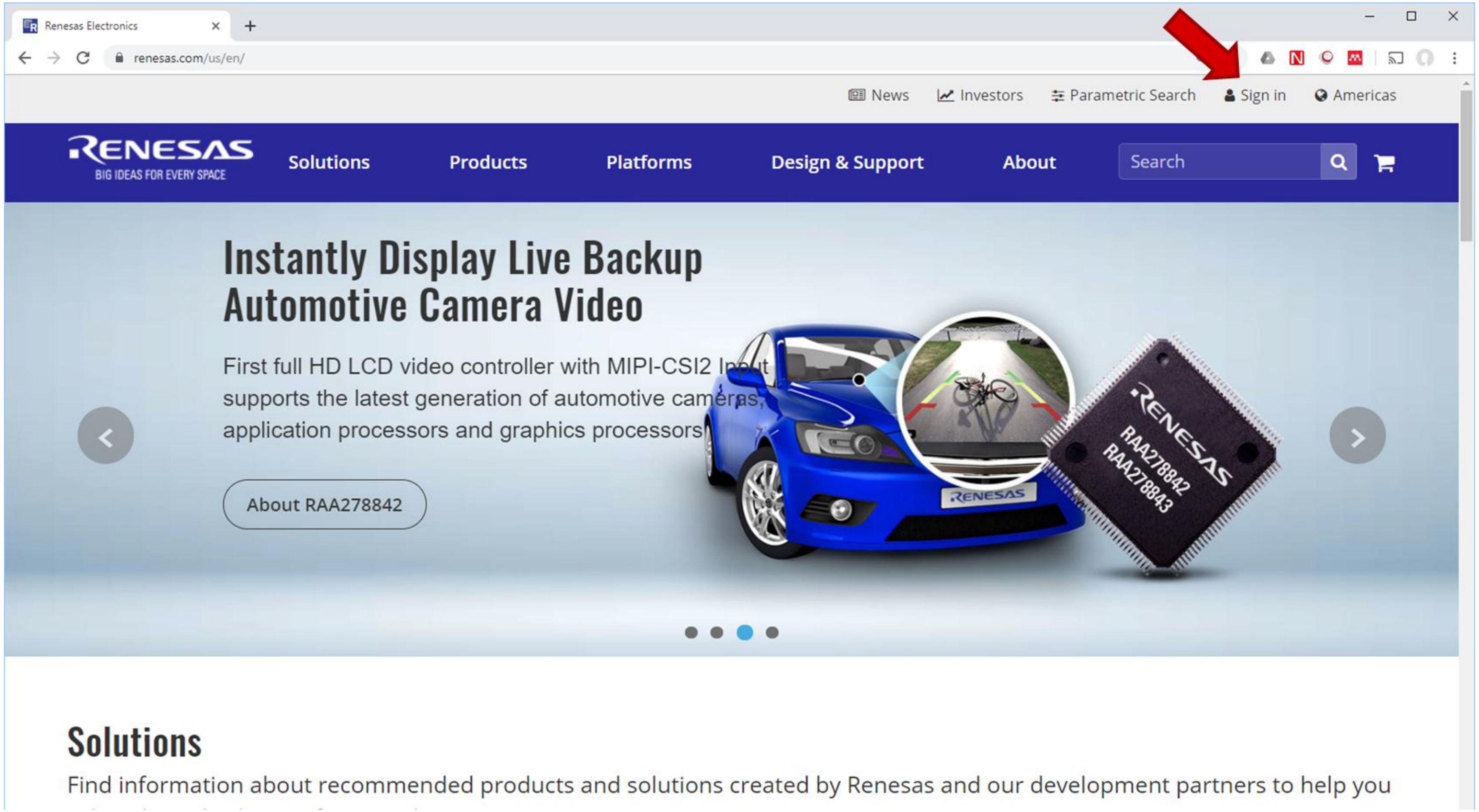
1. Register an account at https://www.renesas.com
2. Sign in
3. Download Platform Installer
4. Execute the installation
5. Execute e2_studio

LAB 1 – ACTIVITY 1 – STEP 1

Register

a) create an account with Renesas

- click on sign in



Instantly Display Live Backup Automotive Camera Video

First full HD LCD video controller with MIPI-CSI2 Input supports the latest generation of automotive cameras, application processors and graphics processors

About RAA278842

Solutions

Find information about recommended products and solutions created by Renesas and our development partners to help you

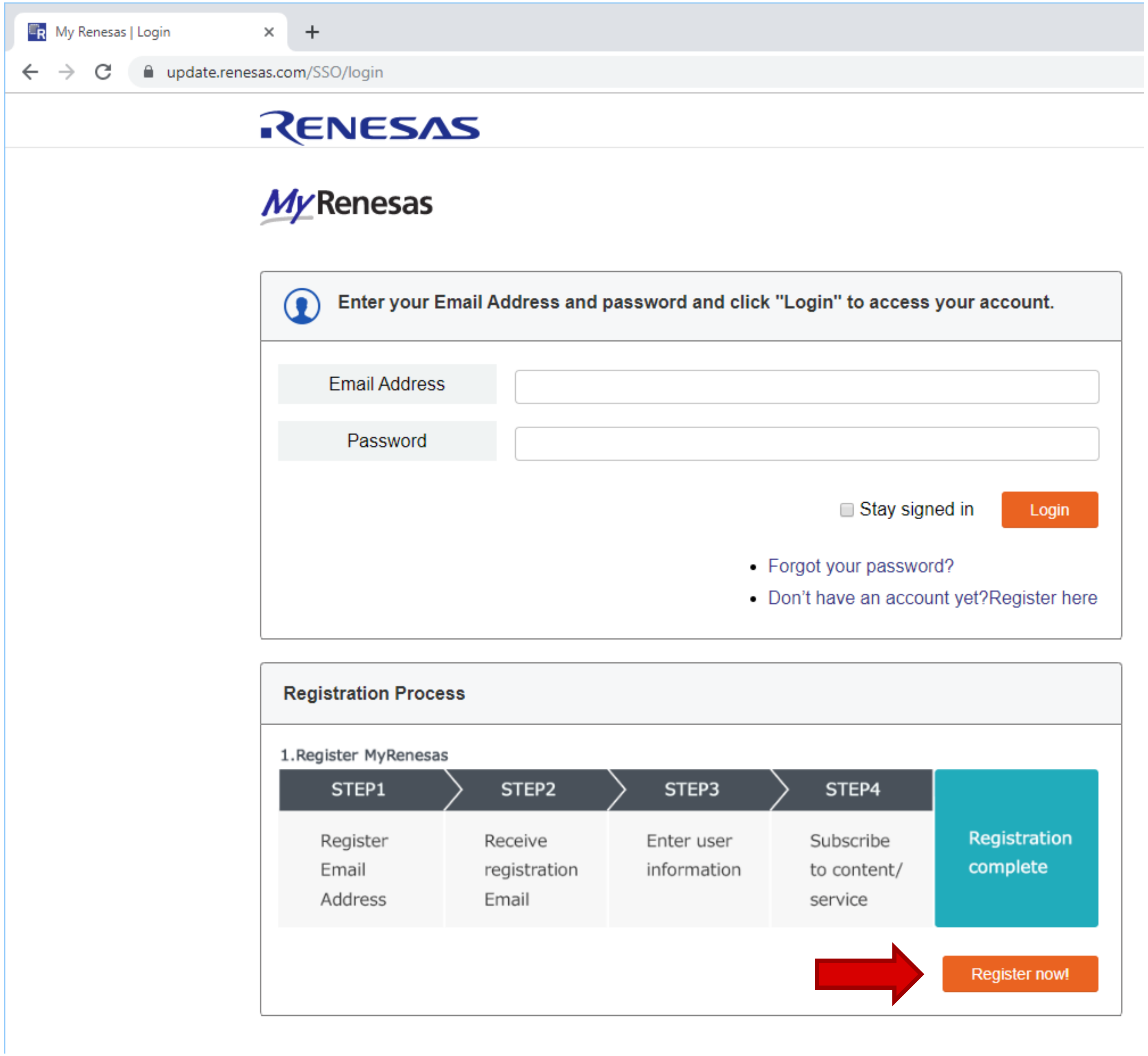
https://www.renesas.com/

LAB 1 – ACTIVITY 1 – STEP 1

Register

a) click on Register now

b) follow the registration process



Enter your Email Address and password and click "Login" to access your account.

Email Address	
Password	

□ Stay signed in Login

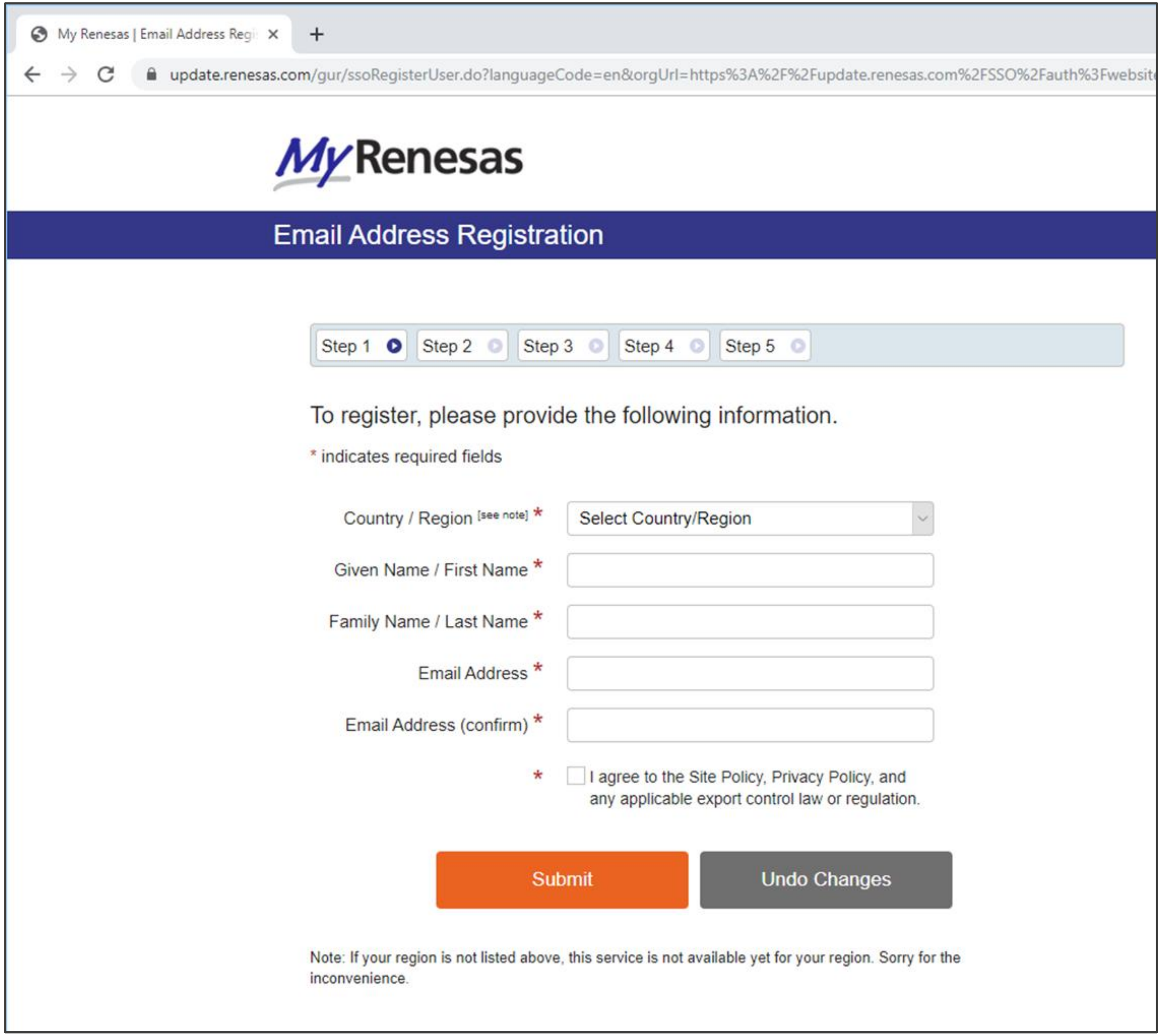
- Forgot your password?
- Don't have an account yet?Register here

Registration Process

1.Register MyRenesas

STEP1	STEP2	STEP3	STEP4	
Register Email Address	Receive registration Email	Enter user information	Subscribe to content/ service	Registration complete

Register now!


My Renesas | Email Address Regi
update.renesas.com/gur/ssoRegisterUser.do?languageCode=en&orgUrl=https%3A%2F%2Fupdate.renesas.com%2FSSO%2Fauth%3Fwebsite
MyRenesas

Email Address Registration

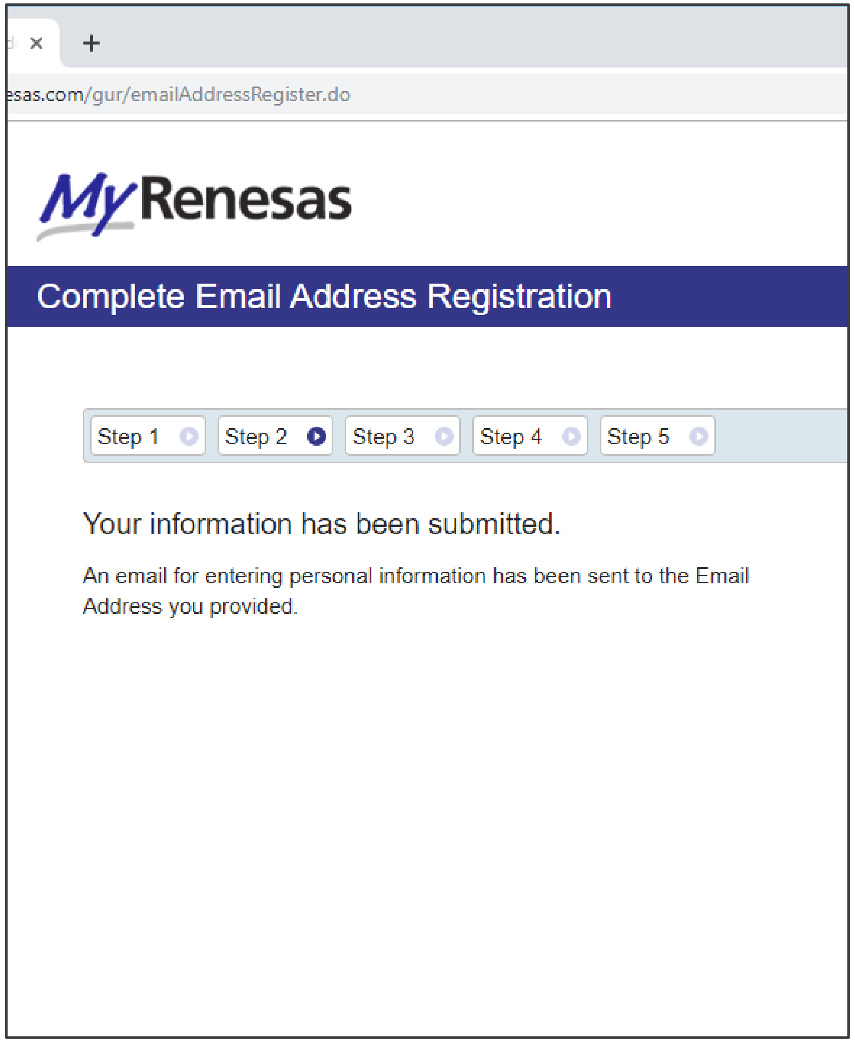
Step 1 ● Step 2 ○ Step 3 ○ Step 4 ○ Step 5 ○

To register, please provide the following information.

* indicates required fields

Country / Region [see note] *	Select Country/Region
Given Name / First Name *	
Family Name / Last Name *	
Email Address *	
Email Address (confirm) *	
*	□ I agree to the Site Policy, Privacy Policy, and any applicable export control law or regulation.

Note: If your region is not listed above, this service is not available yet for your region. Sorry for the inconvenience.



Complete Email Address Registration

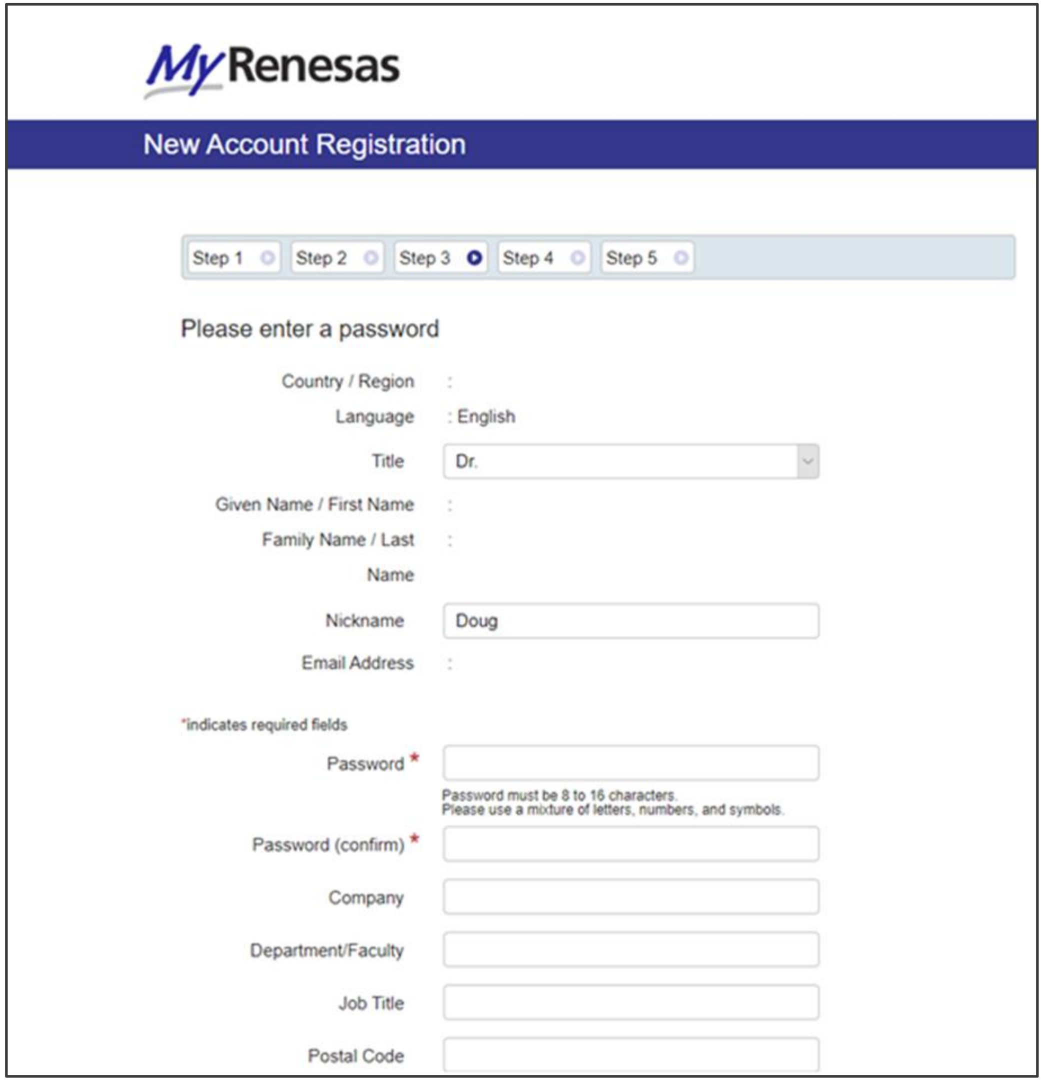
Step 1 ○ Step 2 ● Step 3 ○ Step 4 ○ Step 5 ○

Your information has been submitted.

An email for entering personal information has been sent to the Email Address you provided.

click on the link sent to your mail address to activate the account

by clicking on the link sent by mail,
the registration process proceeds



MyRenesas

New Account Registration

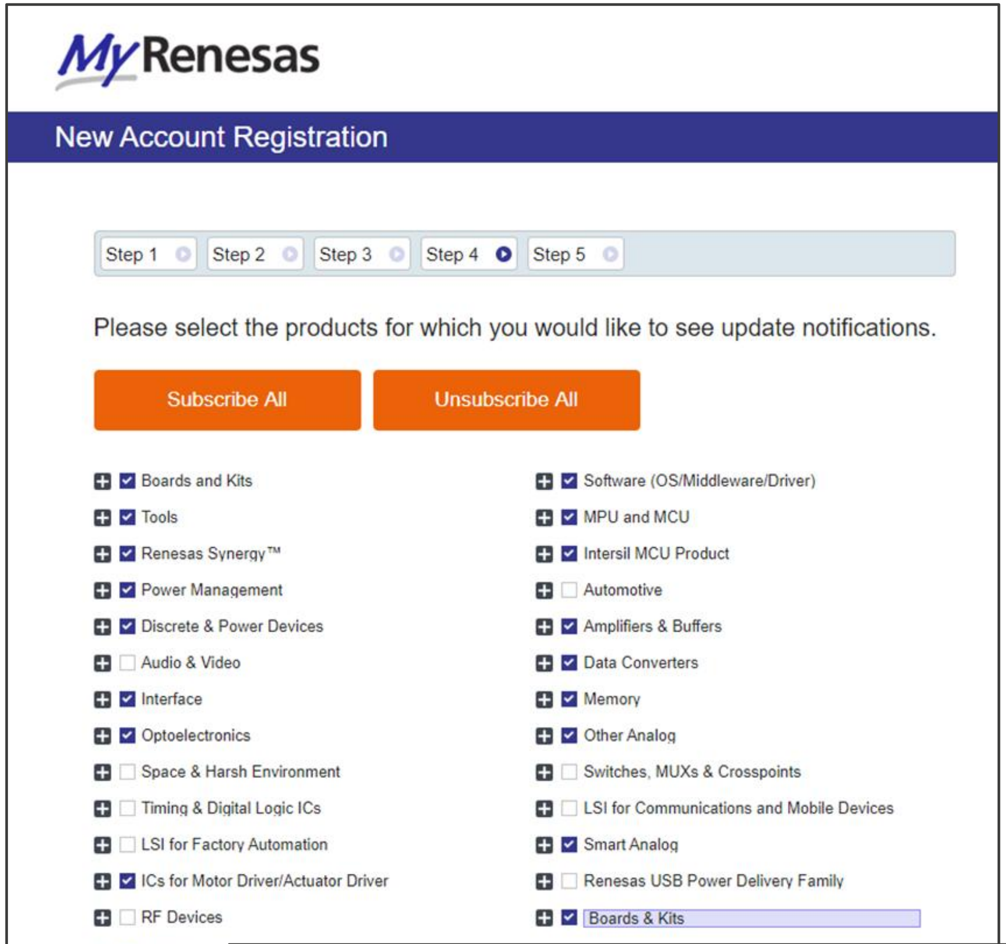
Step 1 ○ Step 2 ○ Step 3 ● Step 4 ○ Step 5 ○

Please enter a password

Country / Region	:
Language	: English
Title	Dr.
Given Name / First Name	:
Family Name / Last Name	:
Nickname	Doug
Email Address	:

*indicates required fields

Password *	
	Password must be 8 to 16 characters. Please use a mixture of letters, numbers, and symbols.
Password (confirm) *	
Company	
Department/Faculty	
Job Title	
Postal Code	



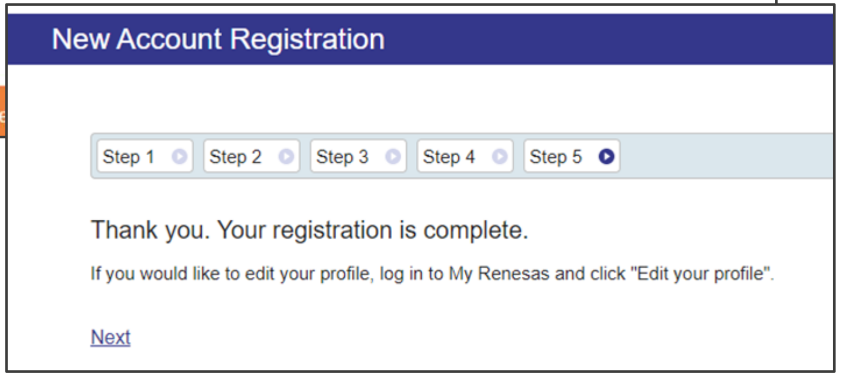
MyRenesas

New Account Registration

Step 1 ○ Step 2 ○ Step 3 ○ Step 4 ● Step 5 ○

Please select the products for which you would like to see update notifications.

- ☑ Boards and Kits
- ☑ Tools
- ☑ Renesas Synergy™
- ☑ Power Management
- ☑ Discrete & Power Devices
- □ Audio & Video
- ☑ Interface
- ☑ Optoelectronics
- □ Space & Harsh Environment
- □ Timing & Digital Logic ICs
- □ LSI for Factory Automation
- ☑ ICs for Motor Driver/Actuator Driver
- □ RF Devices
- ☑ Sensors
- ☑ Software (OS/Middleware/Driver)
- ☑ MPU and MCU
- ☑ Intersil MCU Product
- □ Automotive
- ☑ Amplifiers & Buffers
- ☑ Data Converters
- ☑ Memory
- ☑ Other Analog
- □ Switches, MUXs & Crosspoints
- □ LSI for Communications and Mobile Devices
- ☑ Smart Analog
- □ Renesas USB Power Delivery Family
- ☑ Boards & Kits



New Account Registration

Step 1 ○ Step 2 ○ Step 3 ○ Step 4 ○ Step 5 ●

Thank you. Your registration is complete.

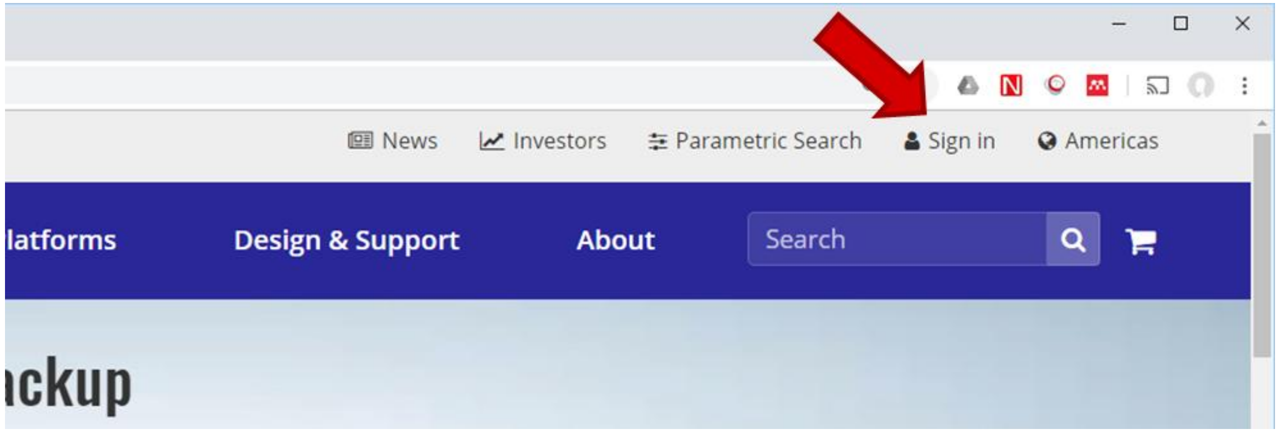
If you would like to edit your profile, log in to My Renesas and click "Edit your profile".

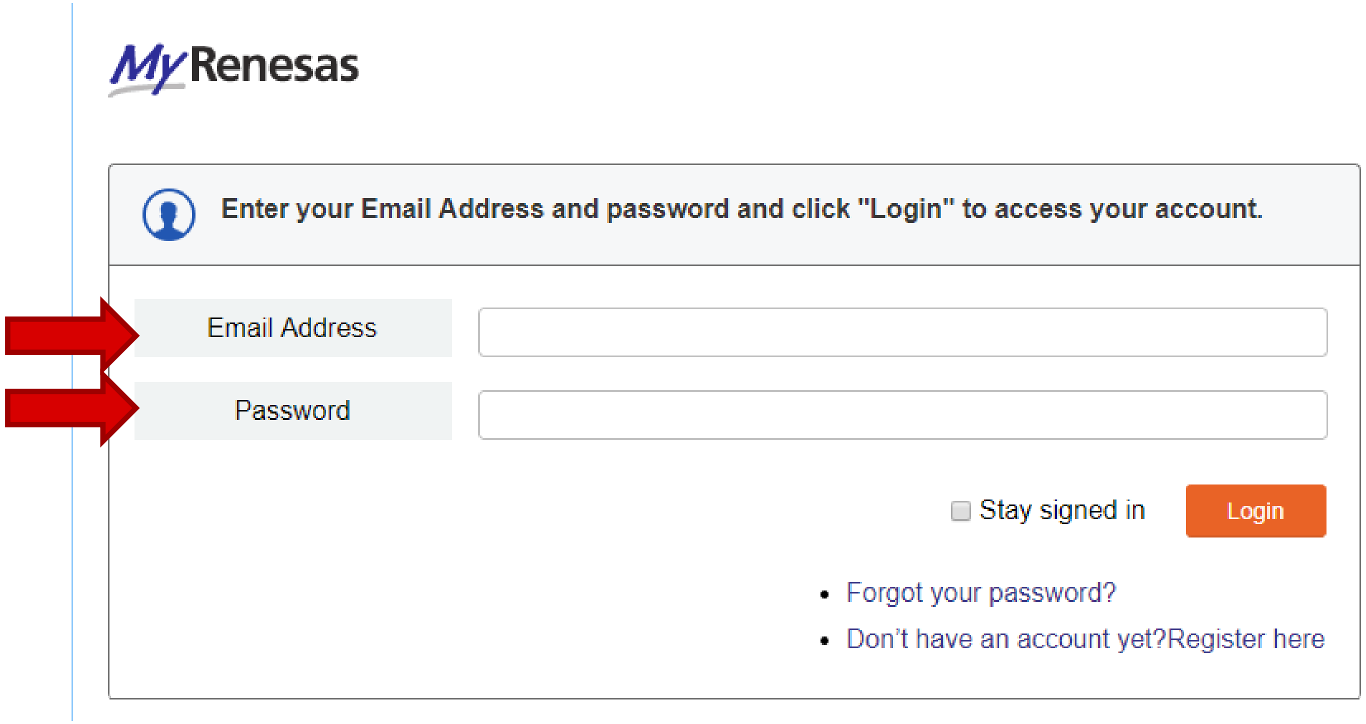
Next

LAB 1 – ACTIVITY 1 – STEP 2

Sign In:

return to renesas.com and sign in with your mail address and the password you defined





Enter your Email Address and password and click "Login" to access your account.

Email Address

Password

□ Stay signed in

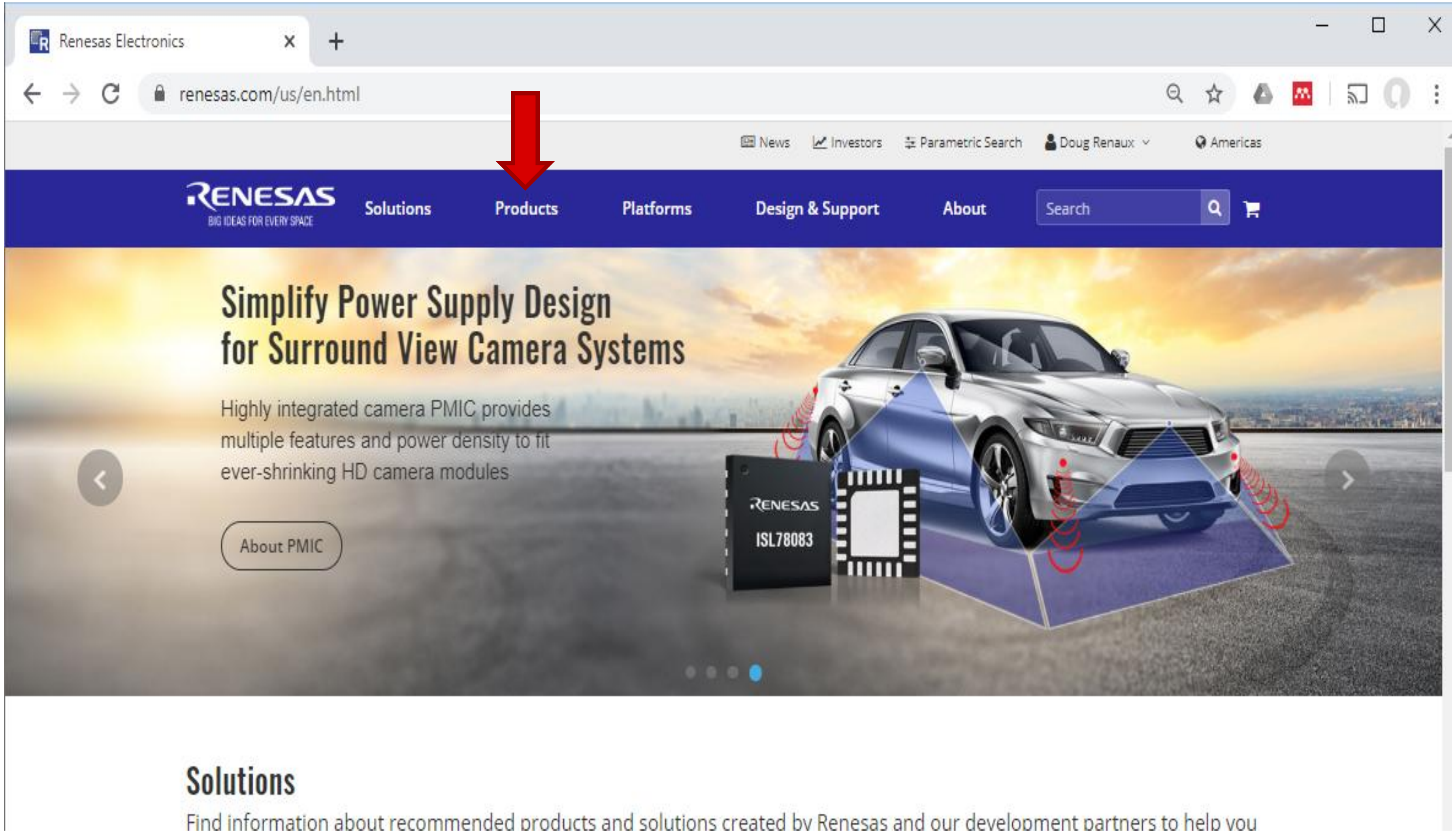
Login

- Forgot your password?
- Don't have an account yet?Register here

LAB 1 – ACTIVITY 1 – STEP 3

Download platform installer

(further registration info may be requested)



Simplify Power Supply Design for Surround View Camera Systems

Highly integrated camera PMIC provides multiple features and power density to fit ever-shrinking HD camera modules

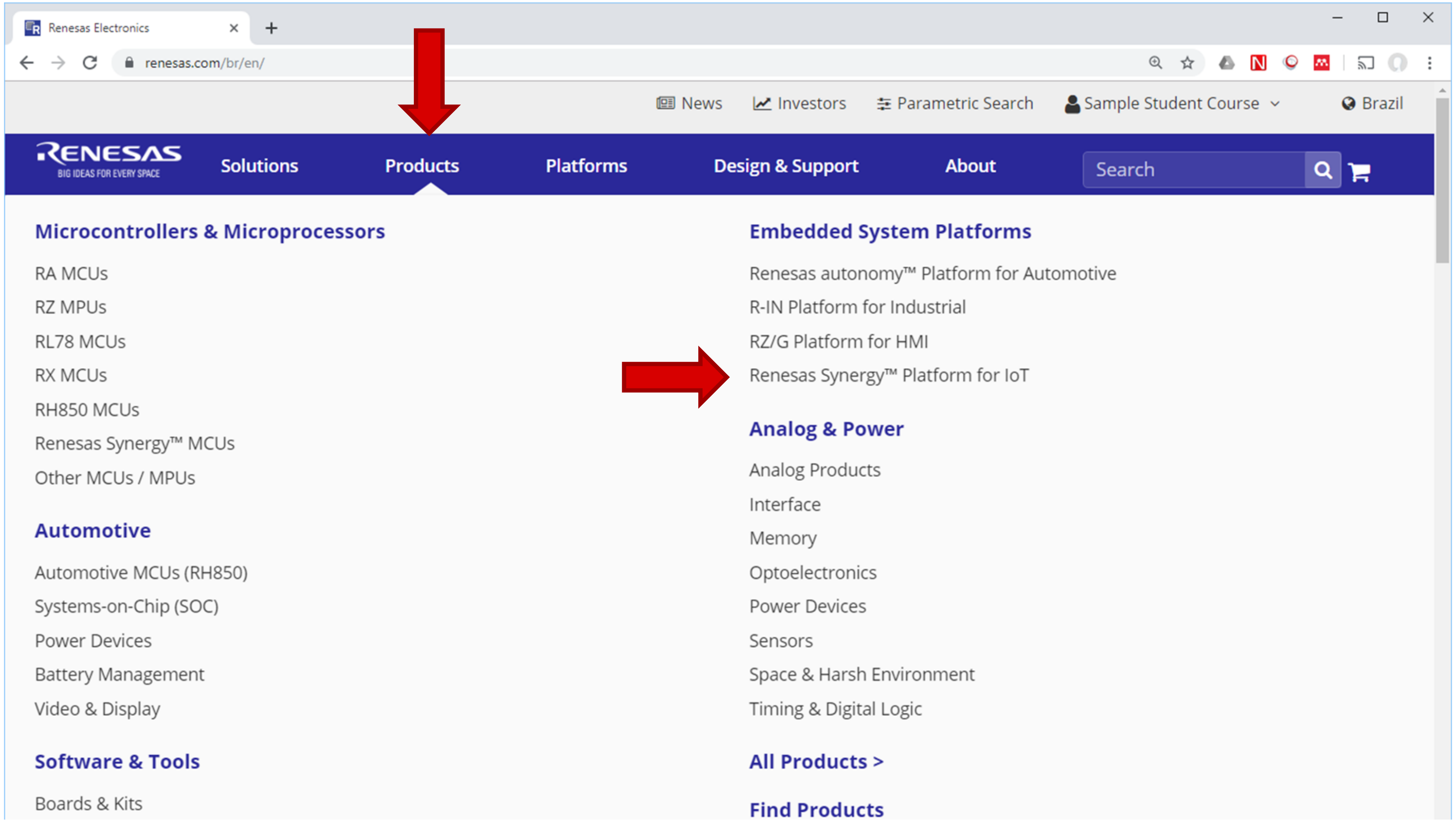
Solutions

Find information about recommended products and solutions created by Renesas and our development partners to help you

LAB 1 – ACTIVITY 1 – STEP 3

Download platform installer

(further registration info may be requested)



Renesas Electronics

renesas.com/br/en/

News | Investors | Parametric Search | Sample Student Course | Brazil

Solutions | Products | Platforms | Design & Support | About | Search

Microcontrollers & Microprocessors

- RA MCUs
- RZ MPUs
- RL78 MCUs
- RX MCUs
- RH850 MCUs
- Renesas Synergy™ MCUs
- Other MCUs / MPUs

Automotive

- Automotive MCUs (RH850)
- Systems-on-Chip (SOC)
- Power Devices
- Battery Management
- Video & Display

Software & Tools

- Boards & Kits

Embedded System Platforms

- Renesas autonomy™ Platform for Automotive
- R-IN Platform for Industrial
- RZ/G Platform for HMI
- Renesas Synergy™ Platform for IoT

Analog & Power

- Analog Products
- Interface
- Memory
- Optoelectronics
- Power Devices
- Sensors
- Space & Harsh Environment
- Timing & Digital Logic

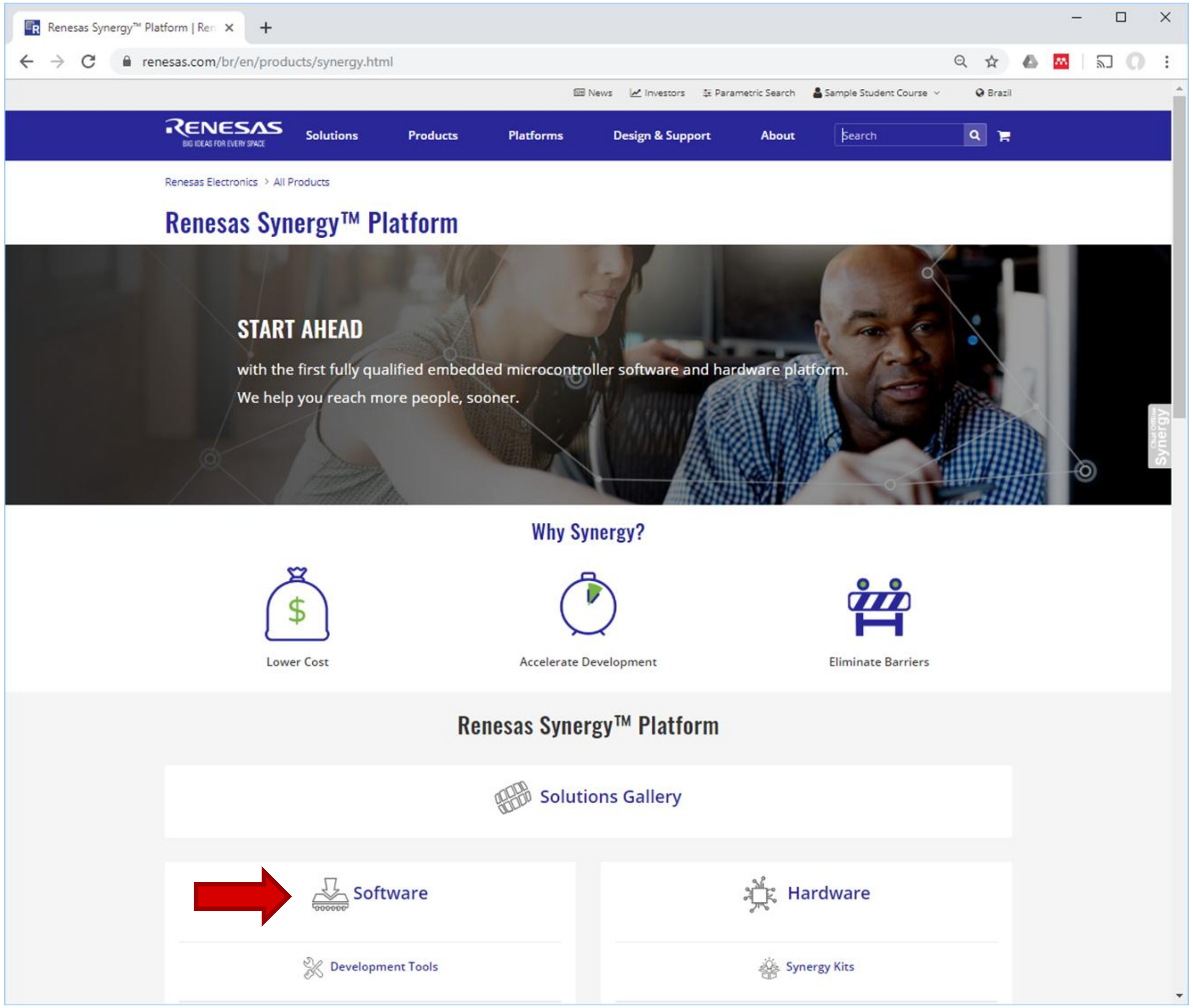
All Products >

Find Products

LAB 1 – ACTIVITY 1 – STEP 3

Download platform installer

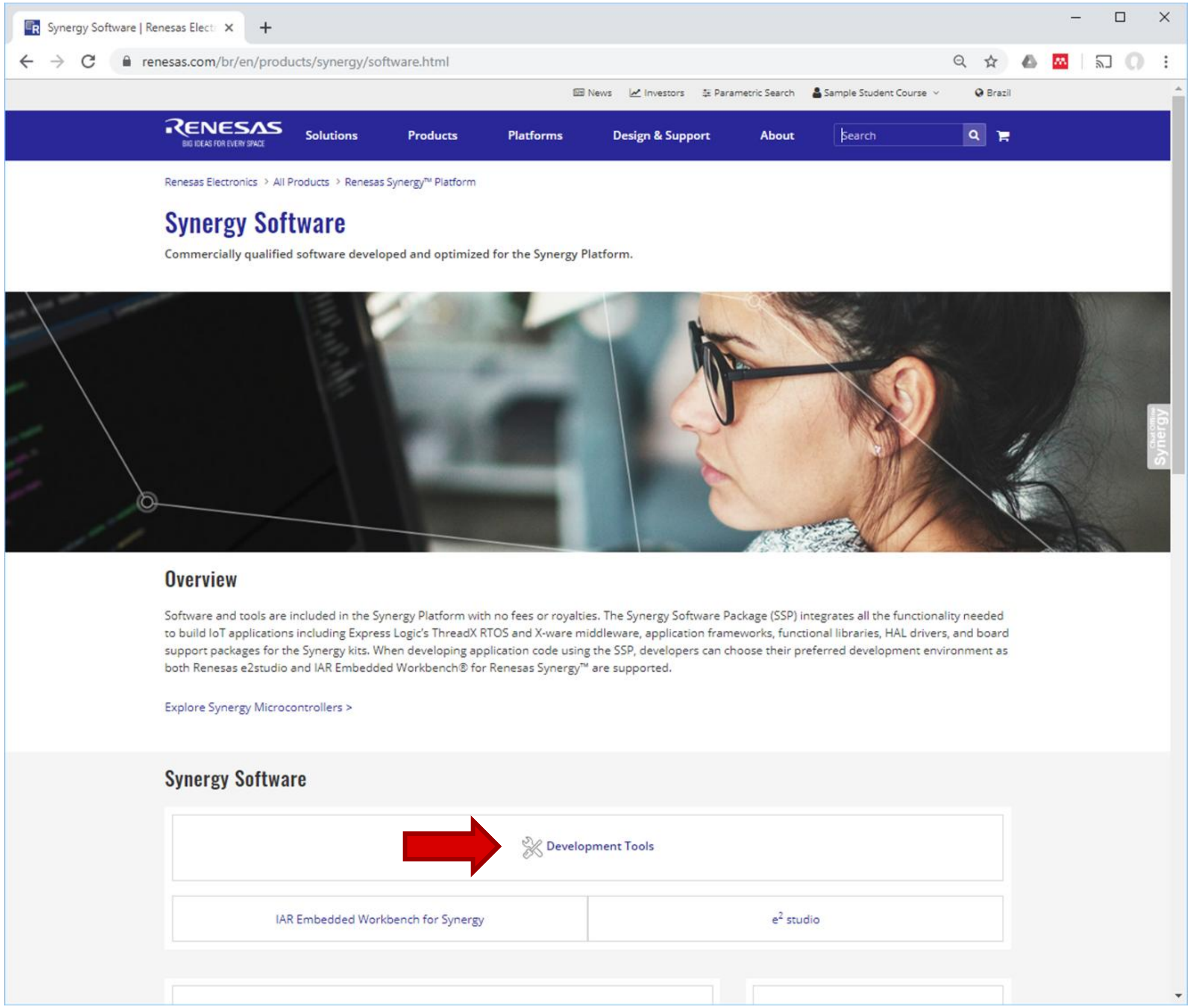
(further registration info may be requested)


Renesas Synergy™ Platform | Ren
renesas.com/br/en/products/synergy.html
News
Investors
Parametric Search
Sample Student Course
Brazil
RENESAS
BIG IDEAS FOR EVERY SPACE
Solutions
Products
Platforms
Design & Support
About
Search
Renesas Electronics > All Products
Renesas Synergy™ Platform
START AHEAD
with the first fully qualified embedded microcontroller software and hardware platform.
We help you reach more people, sooner.
Why Synergy?
Lower Cost
Accelerate Development
Eliminate Barriers
Renesas Synergy™ Platform
Solutions Gallery
Software
Hardware
Development Tools
Synergy Kits

LAB 1 – ACTIVITY 1 – STEP 3

Download platform installer

(further registration info may be requested)


Synergy Software | Renesas Elect
renesas.com/br/en/products/synergy/software.html
News
Investors
Parametric Search
Sample Student Course
Brazil
RENESAS
BIG IDEAS FOR EVERY SPACE
Solutions
Products
Platforms
Design & Support
About
Search
Renesas Electronics > All Products > Renesas Synergy™ Platform
Synergy

Synergy Software

Commercially qualified software developed and optimized for the Synergy Platform.

Overview

Software and tools are included in the Synergy Platform with no fees or royalties. The Synergy Software Package (SSP) integrates all the functionality needed to build IoT applications including Express Logic's ThreadX RTOS and X-ware middleware, application frameworks, functional libraries, HAL drivers, and board support packages for the Synergy kits. When developing application code using the SSP, developers can choose their preferred development environment as both Renesas e2studio and IAR Embedded Workbench® for Renesas Synergy™ are supported.

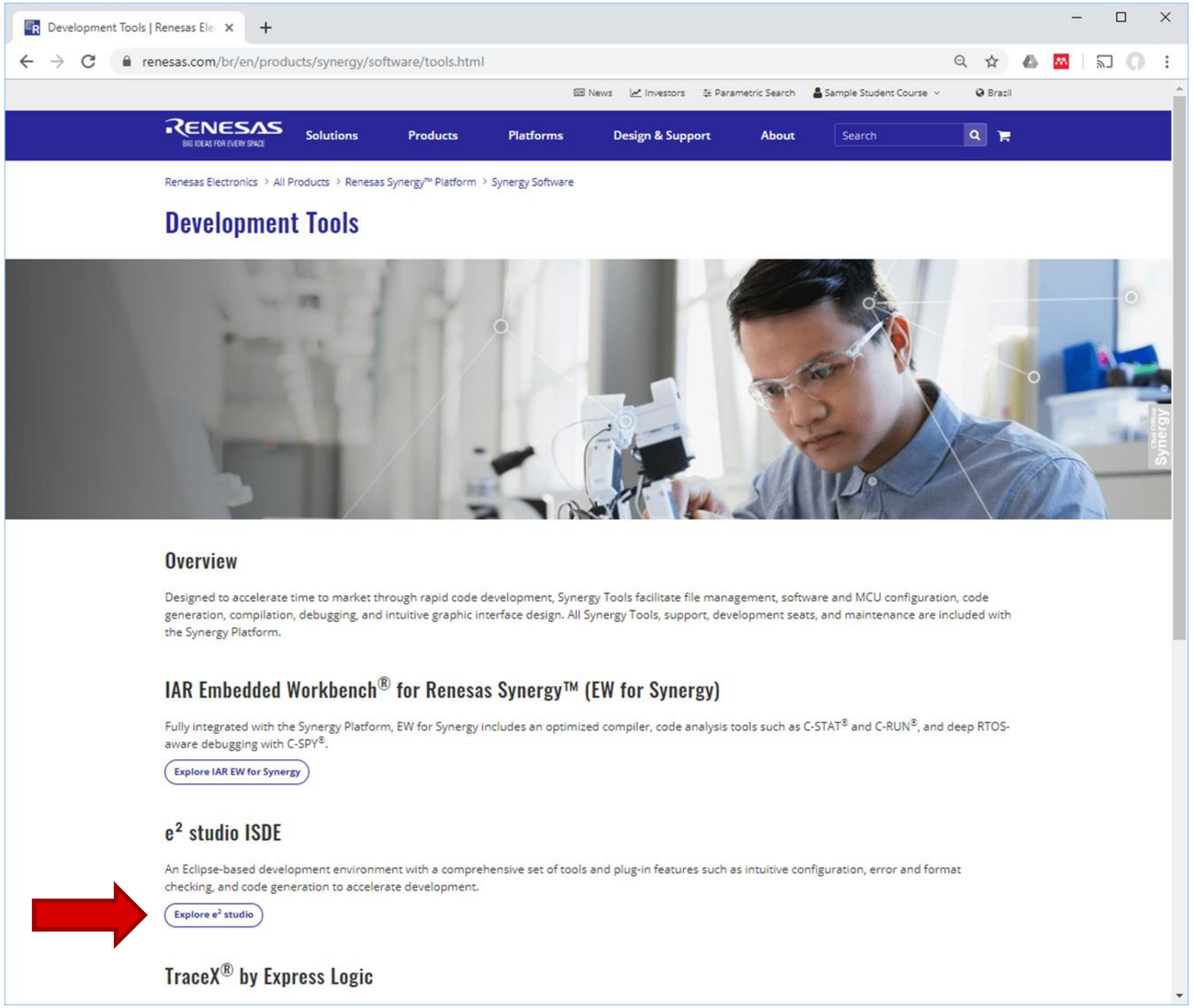
Explore Synergy Microcontrollers >

Synergy Software

LAB 1 – ACTIVITY 1 – STEP 3

Download platform installer

(further registration info may be requested)



Renesas Electronics > All Products > Renesas Synergy™ Platform > Synergy Software

Development Tools

Overview

Designed to accelerate time to market through rapid code development, Synergy Tools facilitate file management, software and MCU configuration, code generation, compilation, debugging, and intuitive graphic interface design. All Synergy Tools, support, development seats, and maintenance are included with the Synergy Platform.

IAR Embedded Workbench® for Renesas Synergy™ (EW for Synergy)

Fully integrated with the Synergy Platform, EW for Synergy includes an optimized compiler, code analysis tools such as C-STAT® and C-RUN®, and deep RTOS-aware debugging with C-SPY®.

e² studio ISDE

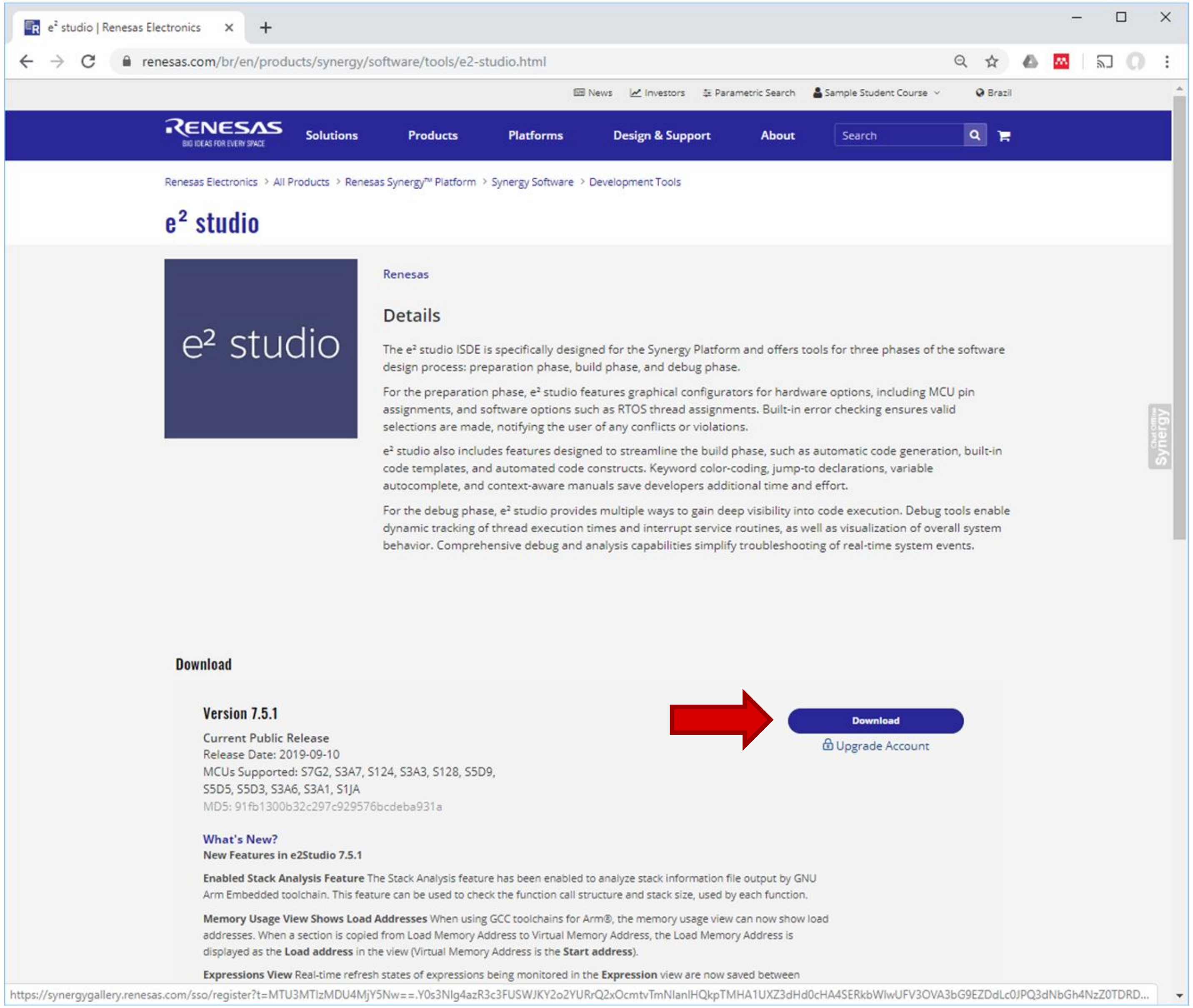
An Eclipse-based development environment with a comprehensive set of tools and plug-in features such as intuitive configuration, error and format checking, and code generation to accelerate development.

TraceX® by Express Logic

LAB 1 – ACTIVITY 1 – STEP 3

Download platform installer

(further registration info may be requested)



e² studio | Renesas Electronics

renesas.com/br/en/products/synergy/software/tools/e2-studio.html

Renesas Electronics > All Products > Renesas Synergy™ Platform > Synergy Software > Development Tools

e² studio

Renesas

Details

The e² studio ISDE is specifically designed for the Synergy Platform and offers tools for three phases of the software design process: preparation phase, build phase, and debug phase.

For the preparation phase, e² studio features graphical configurators for hardware options, including MCU pin assignments, and software options such as RTOS thread assignments. Built-in error checking ensures valid selections are made, notifying the user of any conflicts or violations.

e² studio also includes features designed to streamline the build phase, such as automatic code generation, built-in code templates, and automated code constructs. Keyword color-coding, jump-to declarations, variable autocomplete, and context-aware manuals save developers additional time and effort.

For the debug phase, e² studio provides multiple ways to gain deep visibility into code execution. Debug tools enable dynamic tracking of thread execution times and interrupt service routines, as well as visualization of overall system behavior. Comprehensive debug and analysis capabilities simplify troubleshooting of real-time system events.

Download

Version 7.5.1

Current Public Release
Release Date: 2019-09-10
MCUs Supported: S7G2, S3A7, S124, S3A3, S128, S5D9, S5D5, S5D3, S3A6, S3A1, S1JA
MD5: 91fb1300b32c297c929576bcdeba931a

What's New?

New Features in e2Studio 7.5.1

Enabled Stack Analysis Feature The Stack Analysis feature has been enabled to analyze stack information file output by GNU Arm Embedded toolchain. This feature can be used to check the function call structure and stack size, used by each function.

Memory Usage View Shows Load Addresses When using GCC toolchains for Arm®, the memory usage view can now show load addresses. When a section is copied from Load Memory Address to Virtual Memory Address, the Load Memory Address is displayed as the **Load address** in the view (Virtual Memory Address is the **Start address**).

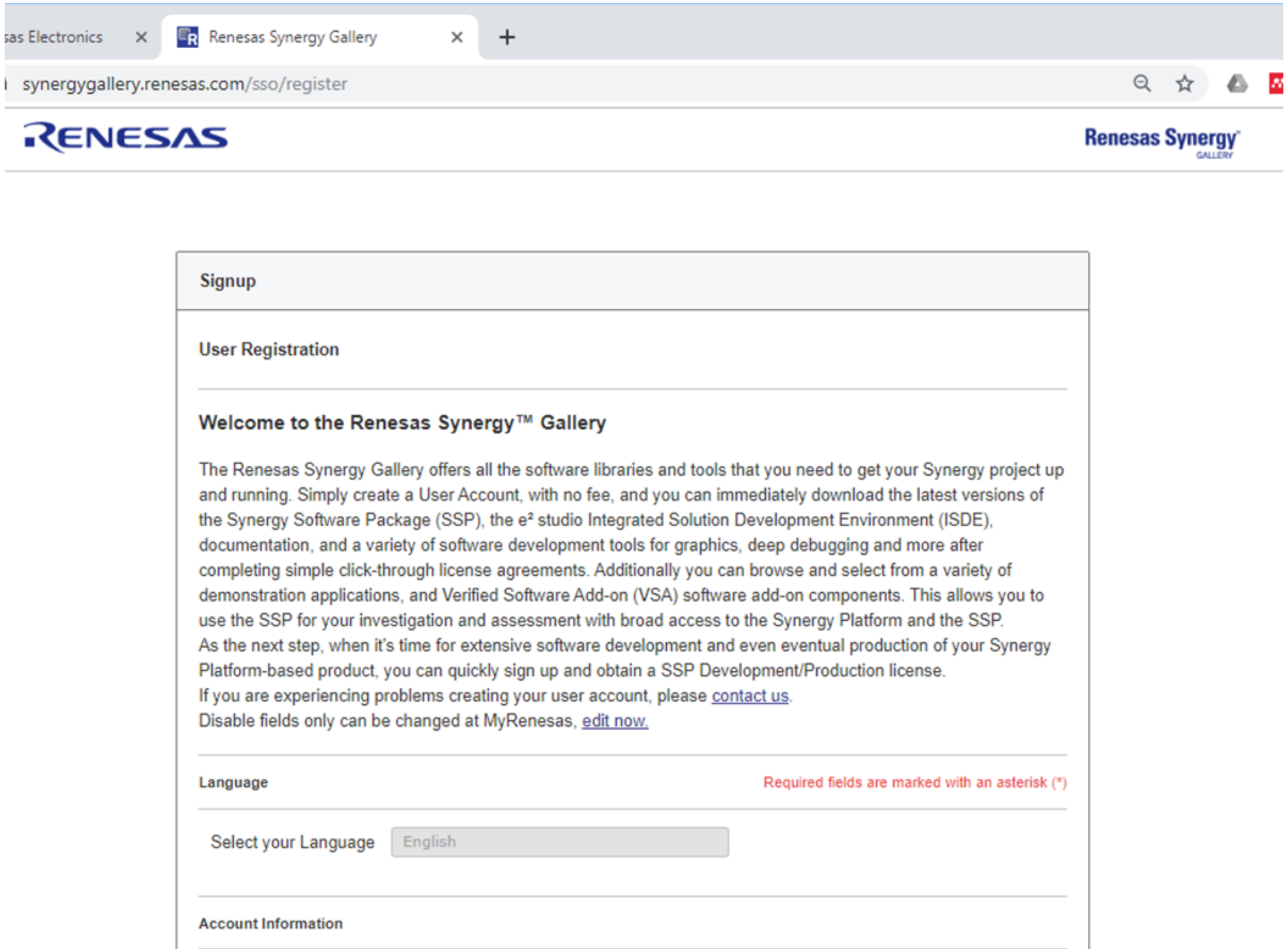
Expressions View Real-time refresh states of expressions being monitored in the **Expression** view are now saved between

https://synergygallery.renesas.com/sso/register?t=MTU3MTIzMDU4MjY5Nw==.Y0s3Nlg4azR3c3FUSWJKY2o2YURrQ2xOcmtvTmNlanlHQkpTMHA1UXZ3dHd0cHA4SERkbWlwUFV3OVA3bG9EZDdLc0JPQ3dNbGh4NzZ0TDRD...

LAB 1 – ACTIVITY 1 – STEP 3

Download platform installer

(further registration info may be requested)



Signup

User Registration

Welcome to the Renesas Synergy™ Gallery

The Renesas Synergy Gallery offers all the software libraries and tools that you need to get your Synergy project up and running. Simply create a User Account, with no fee, and you can immediately download the latest versions of the Synergy Software Package (SSP), the e² studio Integrated Solution Development Environment (ISDE), documentation, and a variety of software development tools for graphics, deep debugging and more after completing simple click-through license agreements. Additionally you can browse and select from a variety of demonstration applications, and Verified Software Add-on (VSA) software add-on components. This allows you to use the SSP for your investigation and assessment with broad access to the Synergy Platform and the SSP.
As the next step, when it's time for extensive software development and even eventual production of your Synergy Platform-based product, you can quickly sign up and obtain a SSP Development/Production license.
If you are experiencing problems creating your user account, please contact us.
Disable fields only can be changed at MyRenesas, edit now.

Language — Required fields are marked with an asterisk (*)

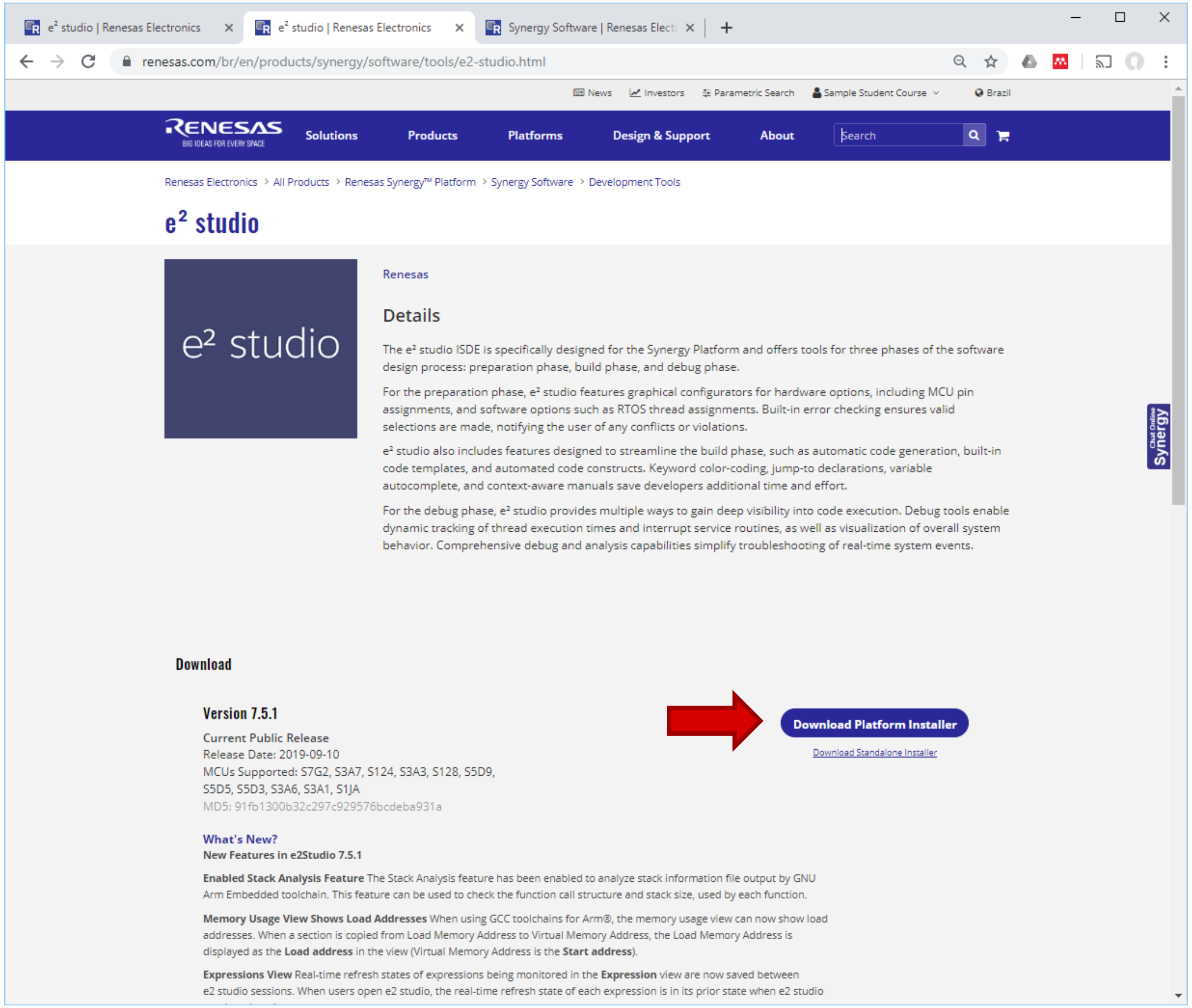
Select your Language: English

Account Information

LAB 1 – ACTIVITY 1 – STEP 3

Download platform installer

(further registration info may be requested)



e² studio | Renesas Electronics
e² studio | Renesas Electronics
Synergy Software | Renesas Electr
renesas.com/br/en/products/synergy/software/tools/e2-studio.html
News
Investors
Parametric Search
Sample Student Course
Brazil
RENESAS
BIG IDEAS FOR EVERY SPACE
Solutions
Products
Platforms
Design & Support
About
Search
Synergy

Renesas Electronics > All Products > Renesas Synergy™ Platform > Synergy Software > Development Tools

e² studio

Renesas

Details

The e² studio ISDE is specifically designed for the Synergy Platform and offers tools for three phases of the software design process: preparation phase, build phase, and debug phase.

For the preparation phase, e² studio features graphical configurators for hardware options, including MCU pin assignments, and software options such as RTOS thread assignments. Built-in error checking ensures valid selections are made, notifying the user of any conflicts or violations.

e² studio also includes features designed to streamline the build phase, such as automatic code generation, built-in code templates, and automated code constructs. Keyword color-coding, jump-to declarations, variable autocomplete, and context-aware manuals save developers additional time and effort.

For the debug phase, e² studio provides multiple ways to gain deep visibility into code execution. Debug tools enable dynamic tracking of thread execution times and interrupt service routines, as well as visualization of overall system behavior. Comprehensive debug and analysis capabilities simplify troubleshooting of real-time system events.

Download

Version 7.5.1

Current Public Release
Release Date: 2019-09-10
MCUs Supported: S7G2, S3A7, S124, S3A3, S128, S5D9, S5D5, S5D3, S3A6, S3A1, S1JA
MD5: 91fb1300b32c297c929576bcdeba931a

What's New?

New Features in e2Studio 7.5.1

Enabled Stack Analysis Feature The Stack Analysis feature has been enabled to analyze stack information file output by GNU Arm Embedded toolchain. This feature can be used to check the function call structure and stack size, used by each function.

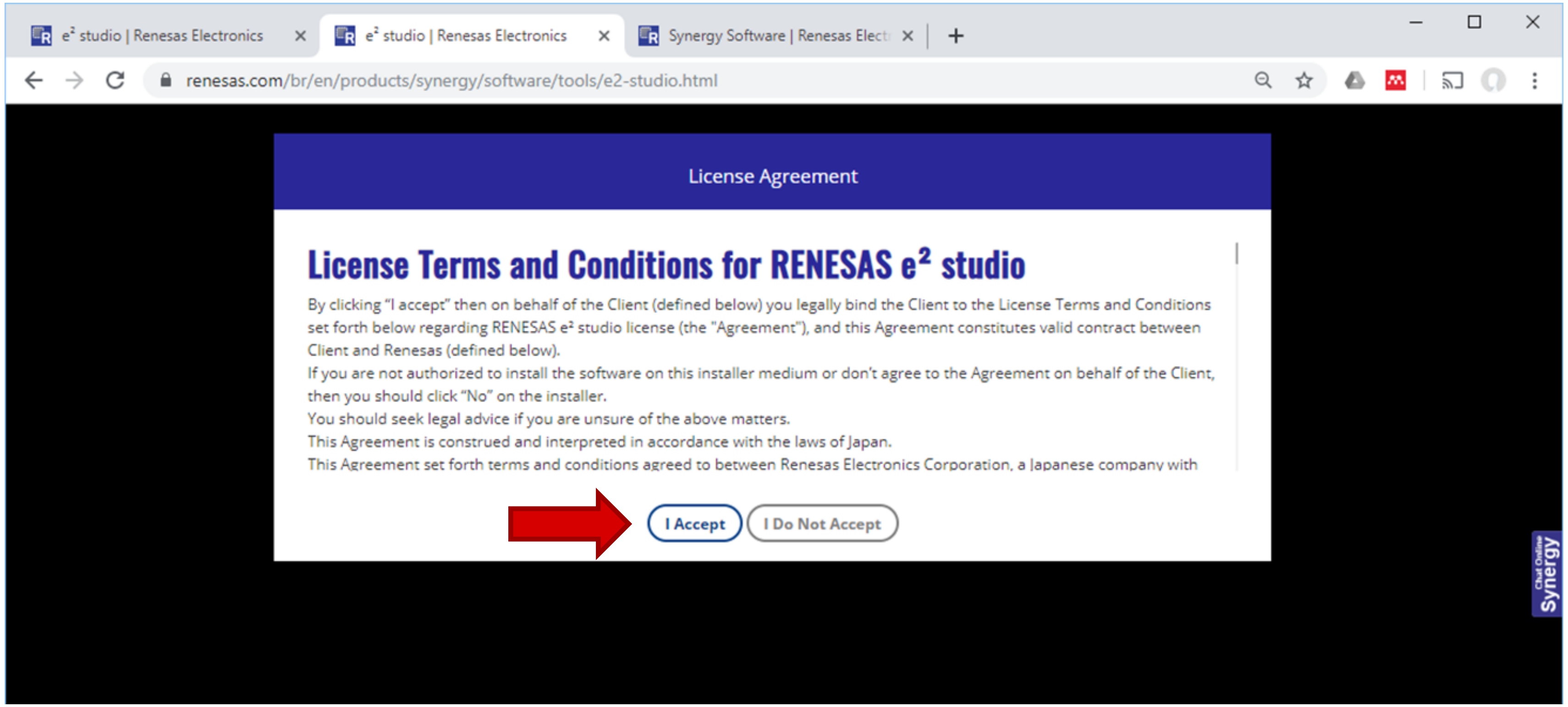
Memory Usage View Shows Load Addresses When using GCC toolchains for Arm®, the memory usage view can now show load addresses. When a section is copied from Load Memory Address to Virtual Memory Address, the Load Memory Address is displayed as the **Load address** in the view (Virtual Memory Address is the **Start address**).

Expressions View Real-time refresh states of expressions being monitored in the **Expression** view are now saved between e2 studio sessions. When users open e2 studio, the real-time refresh state of each expression is in its prior state when e2 studio

LAB 1 – ACTIVITY 1 – STEP 3

Download platform installer

(further registration info may be requested)


e² studio | Renesas Electronics
e² studio | Renesas Electronics
Synergy Software | Renesas Electr
renesas.com/br/en/products/synergy/software/tools/e2-studio.html
License Agreement
I Accept
I Do Not Accept
Synergy

License Terms and Conditions for RENESAS e² studio

By clicking "I accept" then on behalf of the Client (defined below) you legally bind the Client to the License Terms and Conditions set forth below regarding RENESAS e² studio license (the "Agreement"), and this Agreement constitutes valid contract between Client and Renesas (defined below).

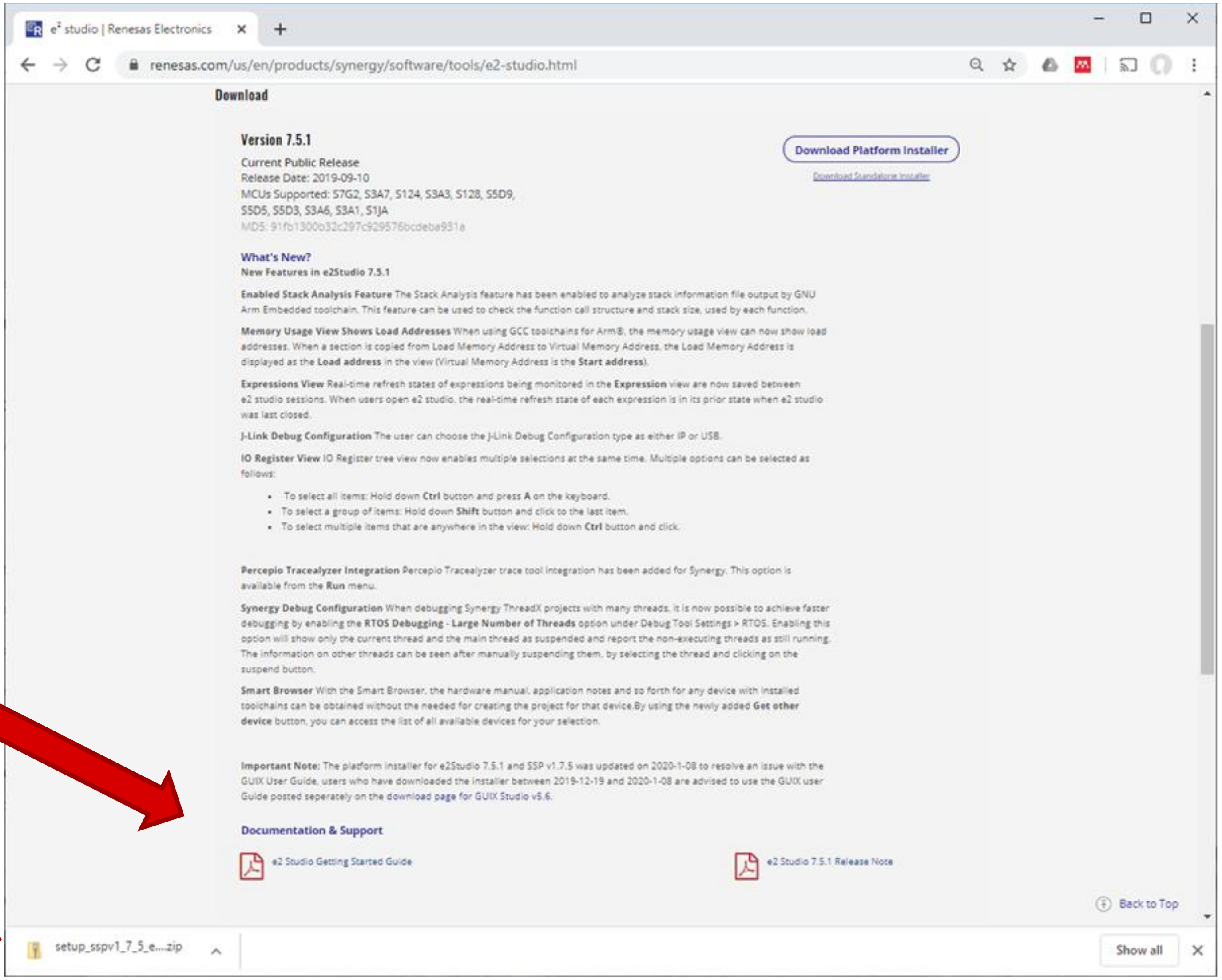
If you are not authorized to install the software on this installer medium or don't agree to the Agreement on behalf of the Client, then you should click "No" on the installer.

You should seek legal advice if you are unsure of the above matters.

This Agreement is construed and interpreted in accordance with the laws of Japan.

This Agreement set forth terms and conditions agreed to between Renesas Electronics Corporation, a Japanese company with

LAB 1 – ACTIVITY 1 – STEP 3



e² studio | Renesas Electronics
renesas.com/us/en/products/synergy/software/tools/e2-studio.html

Download

Version 7.5.1

Current Public Release
Release Date: 2019-09-10
MCUs Supported: S7G2, S3A7, S124, S3A3, S128, S5D9, S5D5, S5D3, S3A6, S3A1, S1JA
MD5: 91fb1300b32c297c929576bcdeba931a

Download Platform Installer
Download Standalone Installer

What's New?
New Features in e2Studio 7.5.1

Enabled Stack Analysis Feature The Stack Analysis feature has been enabled to analyze stack information file output by GNU Arm Embedded toolchain. This feature can be used to check the function call structure and stack size, used by each function.

Memory Usage View Shows Load Addresses When using GCC toolchains for Arm®, the memory usage view can now show load addresses. When a section is copied from Load Memory Address to Virtual Memory Address, the Load Memory Address is displayed as the **Load address** in the view (Virtual Memory Address is the **Start address**).

Expressions View Real-time refresh states of expressions being monitored in the **Expression** view are now saved between e2 studio sessions. When users open e2 studio, the real-time refresh state of each expression is in its prior state when e2 studio was last closed.

J-Link Debug Configuration The user can choose the J-Link Debug Configuration type as either IP or USB.

IO Register View IO Register tree view now enables multiple selections at the same time. Multiple options can be selected as follows:

- To select all items: Hold down **Ctrl** button and press **A** on the keyboard.
- To select a group of items: Hold down **Shift** button and click to the last item.
- To select multiple items that are anywhere in the view: Hold down **Ctrl** button and click.

Percepio Tracealyzer Integration Percepio Tracealyzer trace tool integration has been added for Synergy. This option is available from the **Run** menu.

Synergy Debug Configuration When debugging Synergy ThreadX projects with many threads, it is now possible to achieve faster debugging by enabling the **RTOS Debugging - Large Number of Threads** option under Debug Tool Settings > RTOS. Enabling this option will show only the current thread and the main thread as suspended and report the non-executing threads as still running. The information on other threads can be seen after manually suspending them, by selecting the thread and clicking on the suspend button.

Smart Browser With the Smart Browser, the hardware manual, application notes and so forth for any device with installed toolchains can be obtained without the needed for creating the project for that device.By using the newly added **Get other device** button, you can access the list of all available devices for your selection.

Important Note: The platform installer for e2Studio 7.5.1 and SSP v1.7.5 was updated on 2020-1-08 to resolve an issue with the GUIX User Guide, users who have downloaded the installer between 2019-12-19 and 2020-1-08 are advised to use the GUIX user Guide posted seperately on the download page for GUIX Studio v5.6.

Documentation & Support

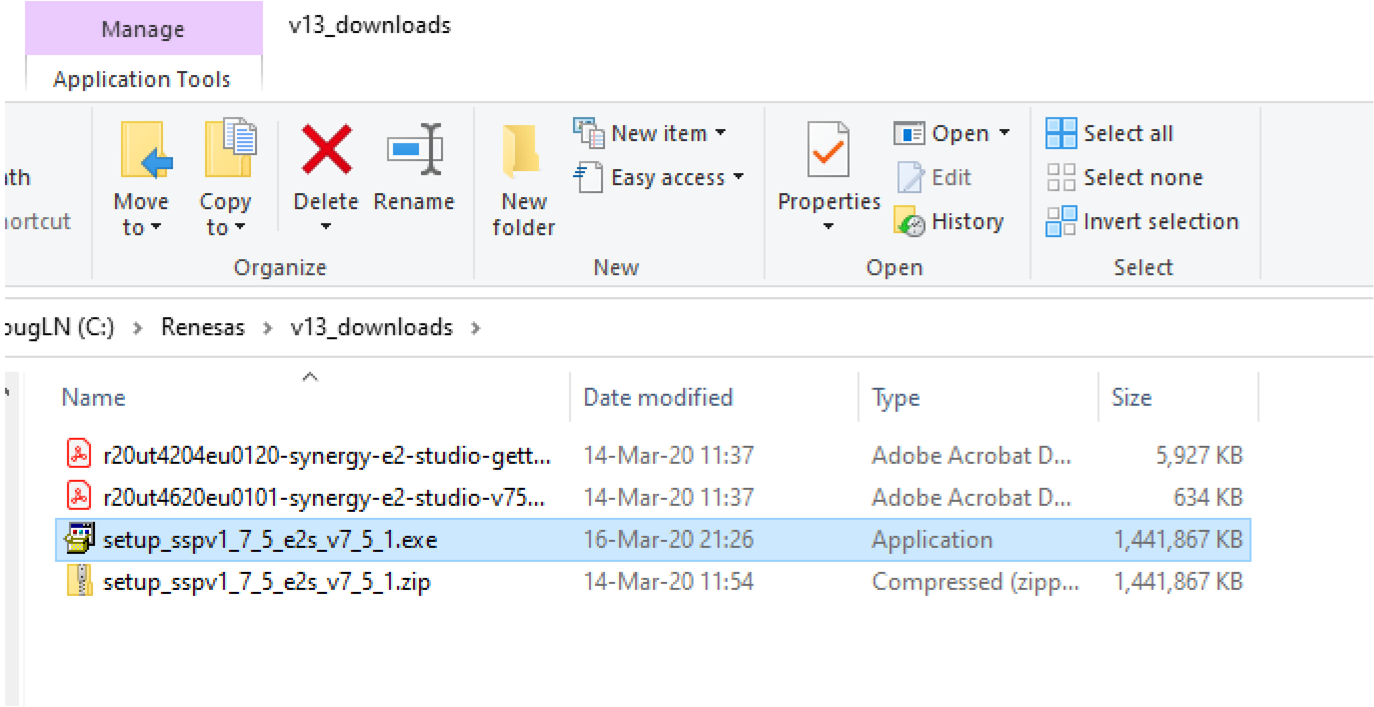
Download these two PDFs:
getting started and **release notes**
they contain important information about the tools and their installation.

platform installer download in progress

LAB 1 – ACTIVITY 1 – STEP 4

Installation process

Unzip the downloaded file to obtain the installation executable, run it.

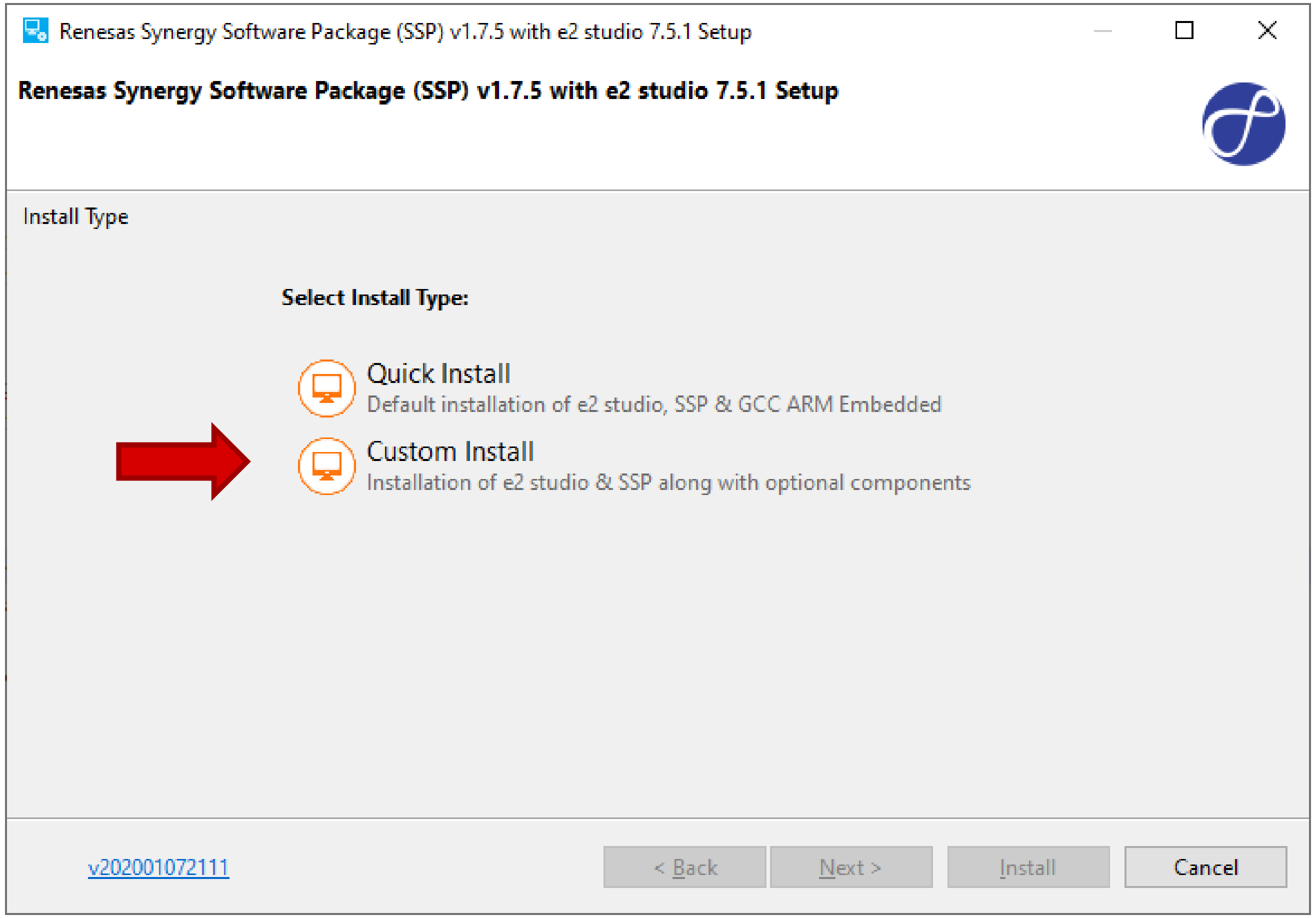


Name	Date modified	Type	Size
r20ut4204eu0120-synergy-e2-studio-gett...	14-Mar-20 11:37	Adobe Acrobat D...	5,927 KB
r20ut4620eu0101-synergy-e2-studio-v75...	14-Mar-20 11:37	Adobe Acrobat D...	634 KB
setup_sspv1_7_5_e2s_v7_5_1.exe	16-Mar-20 21:26	Application	1,441,867 KB
setup_sspv1_7_5_e2s_v7_5_1.zip	14-Mar-20 11:54	Compressed (zipp...	1,441,867 KB

LAB 1 – ACTIVITY 1 – STEP 4

Installation process

Note in red where the user must make appropriate selections



Renesas Synergy Software Package (SSP) v1.7.5 with e2 studio 7.5.1 Setup

Renesas Synergy Software Package (SSP) v1.7.5 with e2 studio 7.5.1 Setup

Install Type

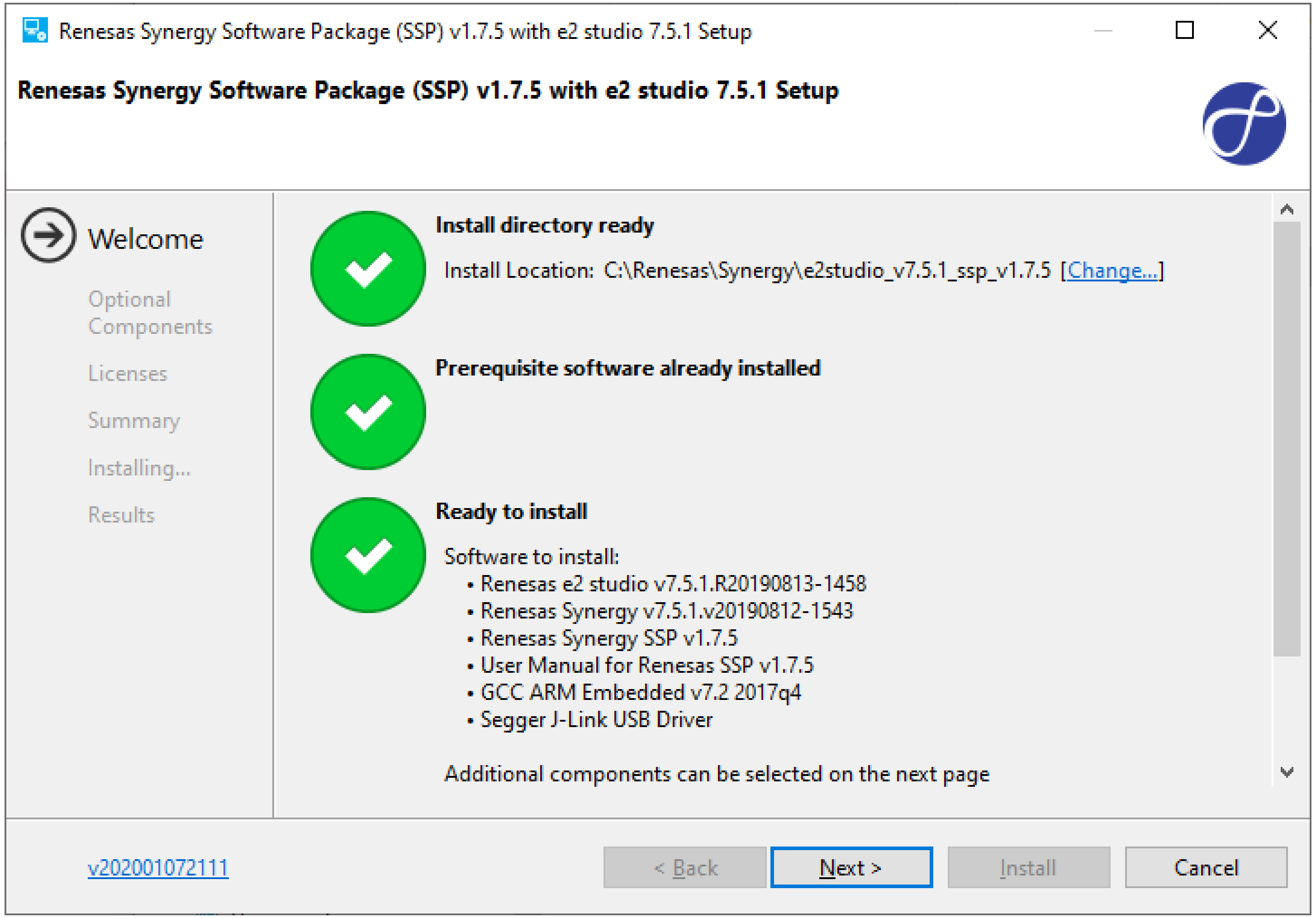
Select Install Type:

Quick Install
Default installation of e2 studio, SSP & GCC ARM Embedded

Custom Install
Installation of e2 studio & SSP along with optional components

v202001072111

LAB 1 – ACTIVITY 1 – STEP 4



Renesas Synergy Software Package (SSP) v1.7.5 with e2 studio 7.5.1 Setup

Renesas Synergy Software Package (SSP) v1.7.5 with e2 studio 7.5.1 Setup

Install directory ready

Install Location: C:\Renesas\Synergy\e2studio_v7.5.1_ssp_v1.7.5 [Change...]

Prerequisite software already installed

Ready to install

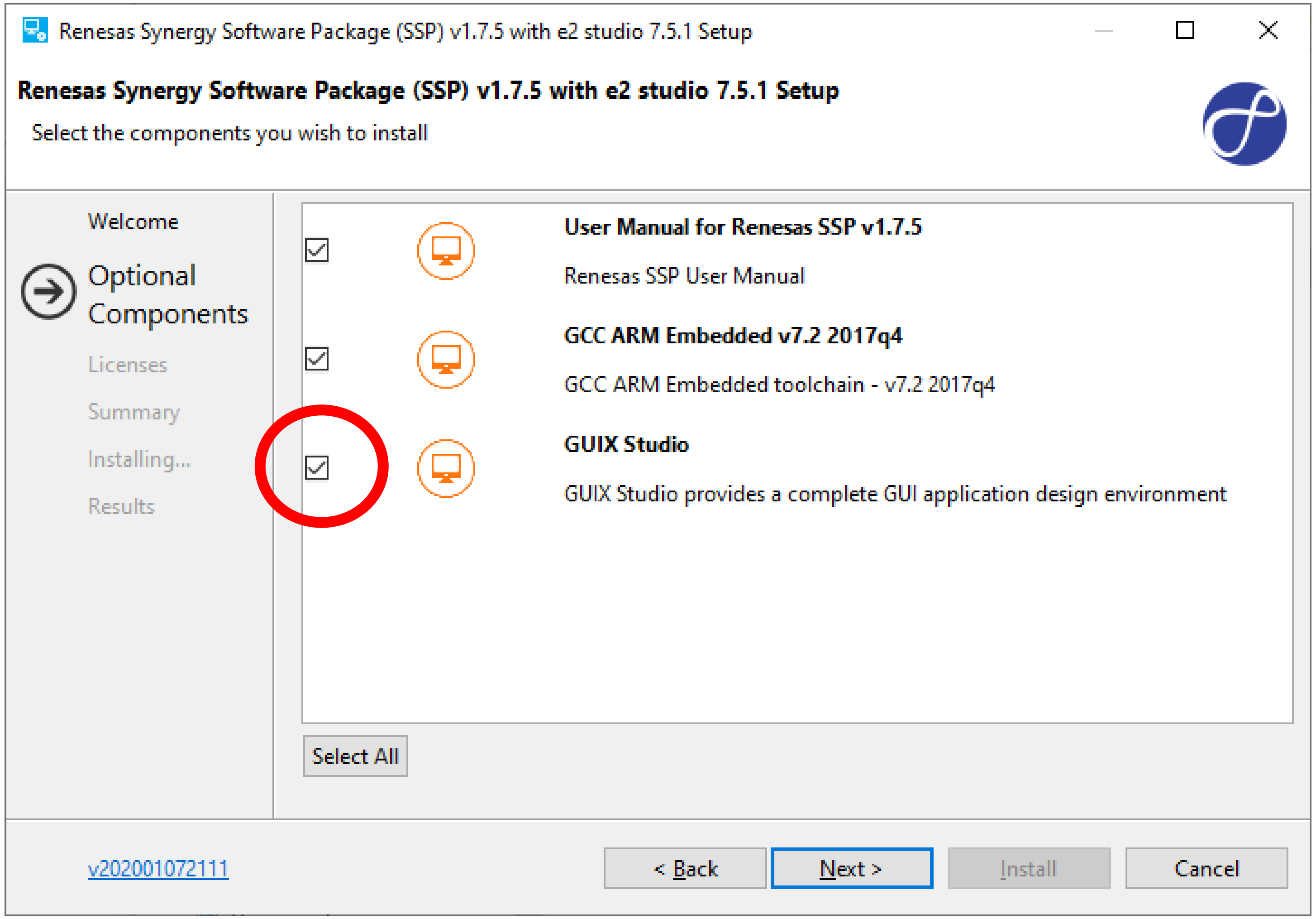
Software to install:

- Renesas e2 studio v7.5.1.R20190813-1458
- Renesas Synergy v7.5.1.v20190812-1543
- Renesas Synergy SSP v1.7.5
- User Manual for Renesas SSP v1.7.5
- GCC ARM Embedded v7.2 2017q4
- Segger J-Link USB Driver

Additional components can be selected on the next page

v202001072111

LAB 1 – ACTIVITY 1 – STEP 4



Renesas Synergy Software Package (SSP) v1.7.5 with e2 studio 7.5.1 Setup

Renesas Synergy Software Package (SSP) v1.7.5 with e2 studio 7.5.1 Setup

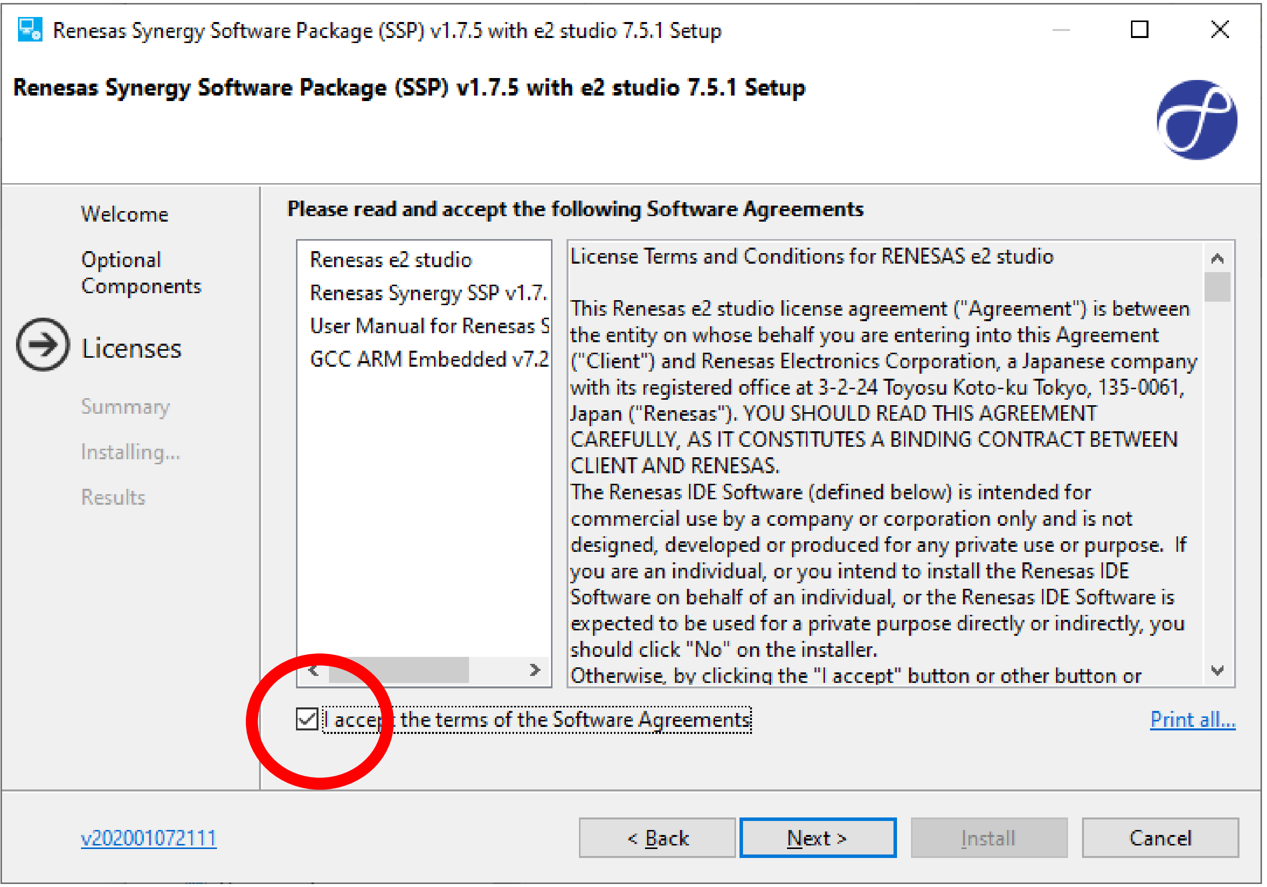
Select the components you wish to install

☑ **User Manual for Renesas SSP v1.7.5**
Renesas SSP User Manual

☑ **GCC ARM Embedded v7.2 2017q4**
GCC ARM Embedded toolchain - v7.2 2017q4

☑ **GUIX Studio**
GUIX Studio provides a complete GUI application design environment

LAB 1 – ACTIVITY 1 – STEP 4



Renesas Synergy Software Package (SSP) v1.7.5 with e2 studio 7.5.1 Setup

Renesas Synergy Software Package (SSP) v1.7.5 with e2 studio 7.5.1 Setup

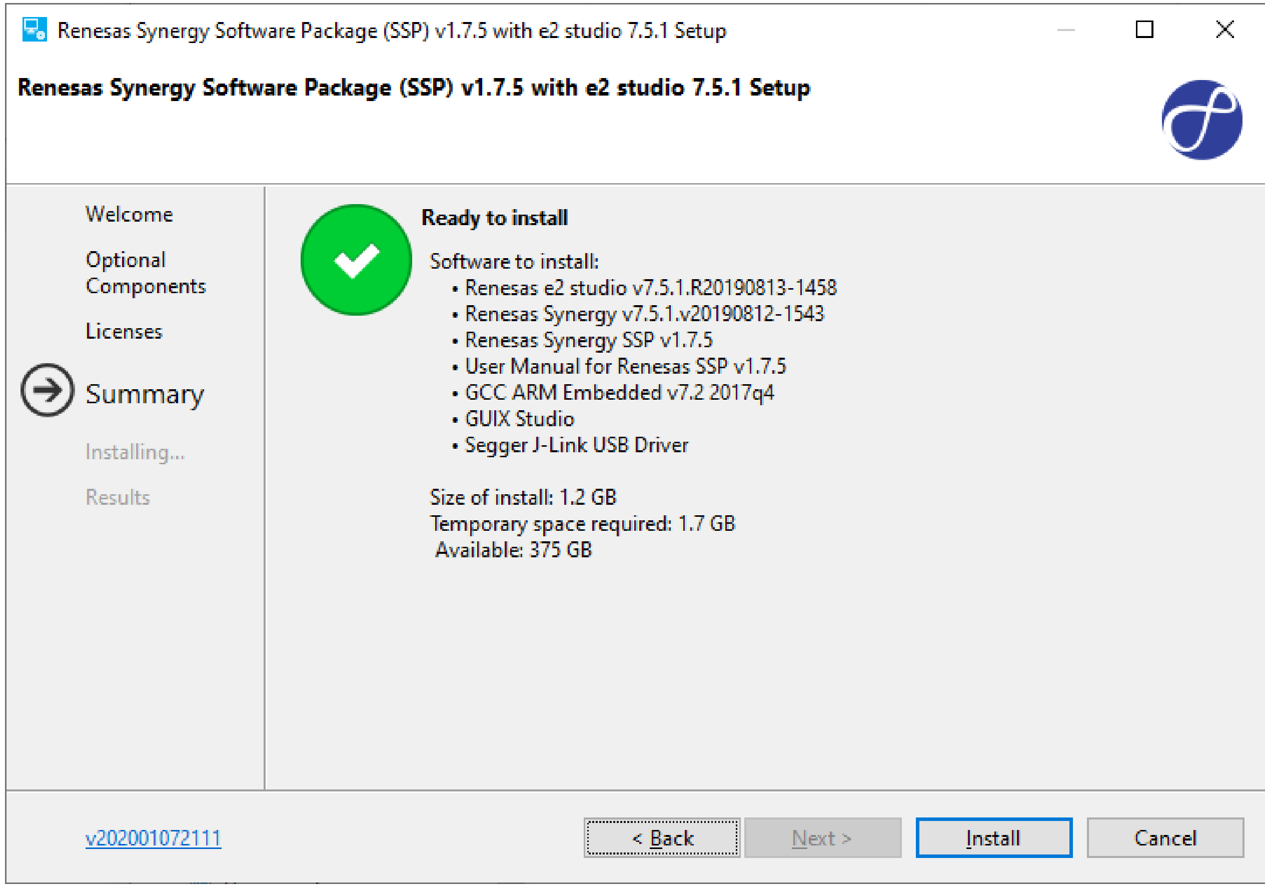
Please read and accept the following Software Agreements

- Renesas e2 studio
- Renesas Synergy SSP v1.7.
- User Manual for Renesas S
- GCC ARM Embedded v7.2

License Terms and Conditions for RENESAS e2 studio

This Renesas e2 studio license agreement ("Agreement") is between the entity on whose behalf you are entering into this Agreement ("Client") and Renesas Electronics Corporation, a Japanese company with its registered office at 3-2-24 Toyosu Koto-ku Tokyo, 135-0061, Japan ("Renesas"). YOU SHOULD READ THIS AGREEMENT CAREFULLY, AS IT CONSTITUTES A BINDING CONTRACT BETWEEN CLIENT AND RENESAS.
The Renesas IDE Software (defined below) is intended for commercial use by a company or corporation only and is not designed, developed or produced for any private use or purpose. If you are an individual, or you intend to install the Renesas IDE Software on behalf of an individual, or the Renesas IDE Software is expected to be used for a private purpose directly or indirectly, you should click "No" on the installer.
Otherwise, by clicking the "I accept" button or other button or

☑ I accept the terms of the Software Agreements



Renesas Synergy Software Package (SSP) v1.7.5 with e2 studio 7.5.1 Setup

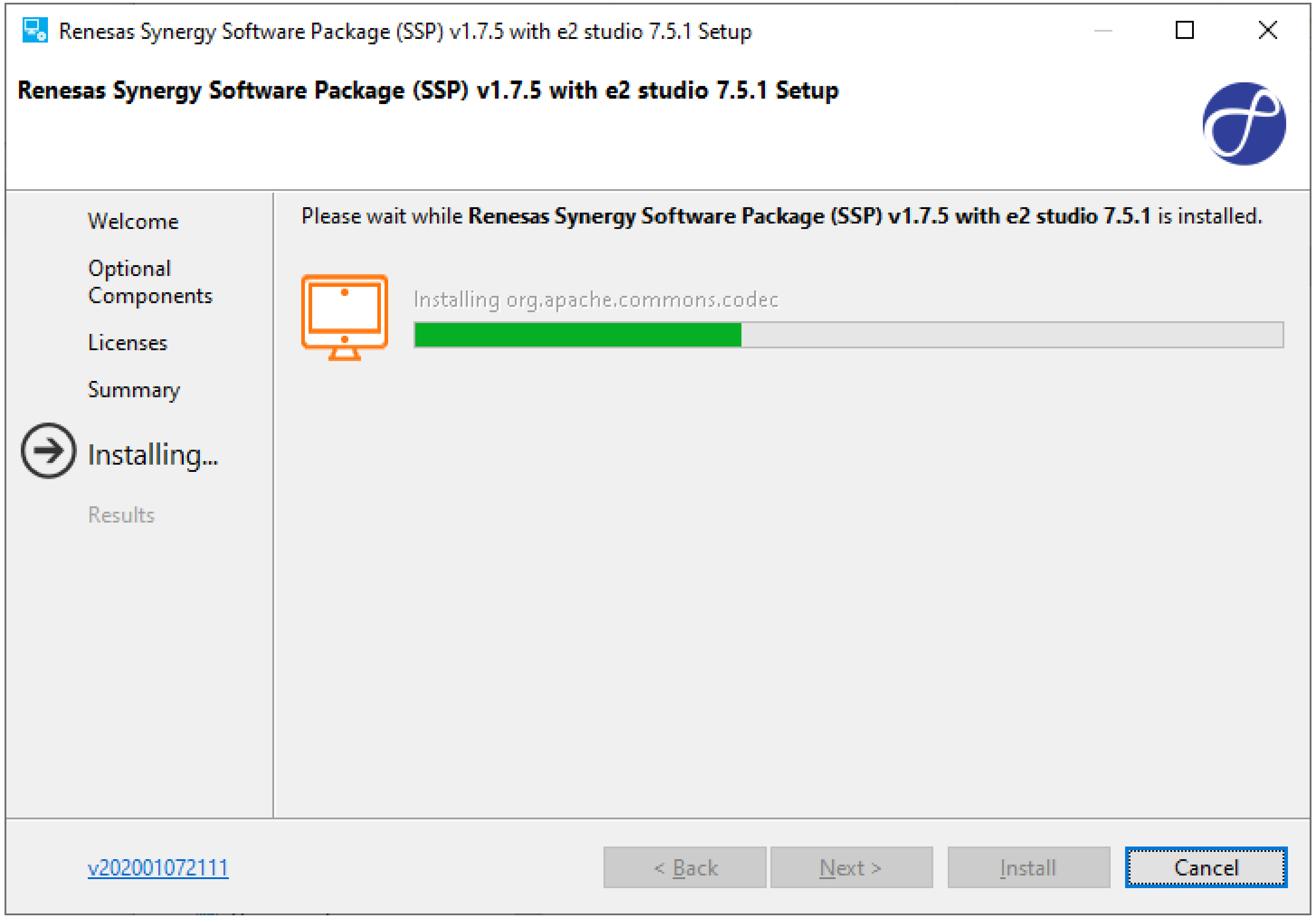
Renesas Synergy Software Package (SSP) v1.7.5 with e2 studio 7.5.1 Setup

Ready to install

Software to install:
- Renesas e2 studio v7.5.1.R20190813-1458
- Renesas Synergy v7.5.1.v20190812-1543
- Renesas Synergy SSP v1.7.5
- User Manual for Renesas SSP v1.7.5
- GCC ARM Embedded v7.2 2017q4
- GUIX Studio
- Segger J-Link USB Driver

Size of install: 1.2 GB
Temporary space required: 1.7 GB
Available: 375 GB

LAB 1 – ACTIVITY 1 – STEP 4



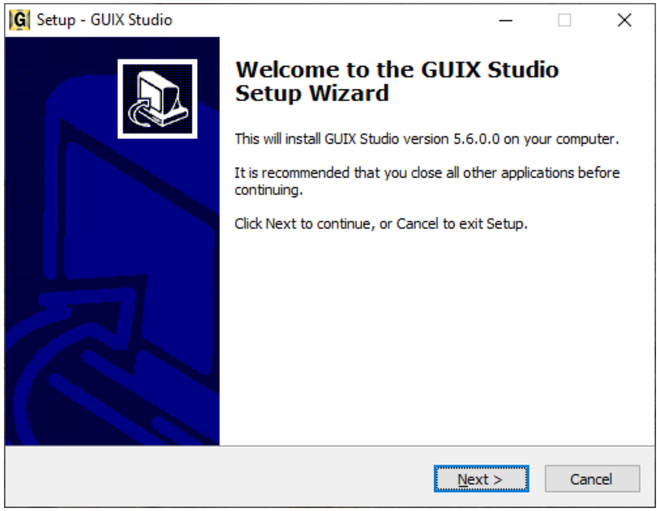
Renesas Synergy Software Package (SSP) v1.7.5 with e2 studio 7.5.1 Setup

Renesas Synergy Software Package (SSP) v1.7.5 with e2 studio 7.5.1 Setup

Please wait while **Renesas Synergy Software Package (SSP) v1.7.5 with e2 studio 7.5.1** is installed.

Installing org.apache.commons.codec

LAB 1 – ACTIVITY 1 – STEP 4



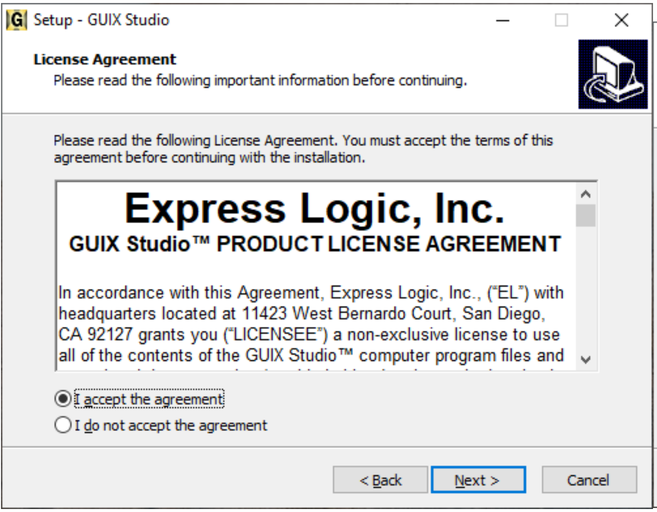
Setup - GUIX Studio

Welcome to the GUIX Studio Setup Wizard

This will install GUIX Studio version 5.6.0.0 on your computer.

It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.



Setup - GUIX Studio

License Agreement

Please read the following important information before continuing.

Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.

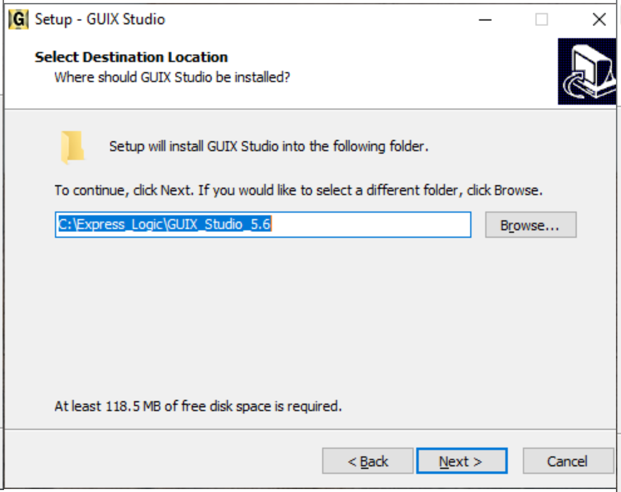
Express Logic, Inc.

GUIX Studio™ PRODUCT LICENSE AGREEMENT

In accordance with this Agreement, Express Logic, Inc., ("EL") with headquarters located at 11423 West Bernardo Court, San Diego, CA 92127 grants you ("LICENSEE") a non-exclusive license to use all of the contents of the GUIX Studio™ computer program files and

● I accept the agreement

○ I do not accept the agreement



Setup - GUIX Studio

Select Destination Location

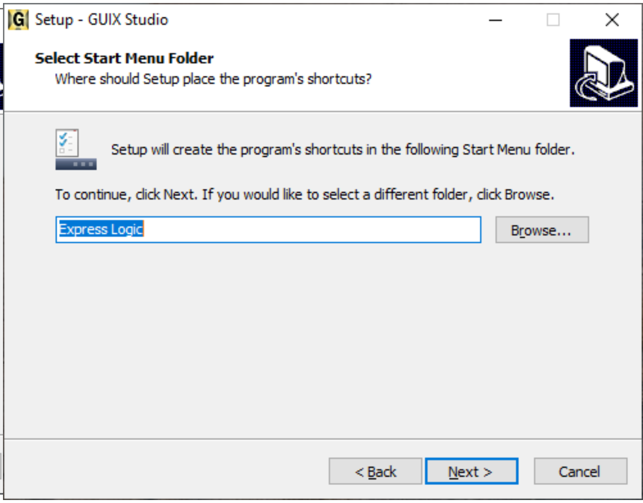
Where should GUIX Studio be installed?

Setup will install GUIX Studio into the following folder.

To continue, click Next. If you would like to select a different folder, click Browse.

C:\Express_Logic\GUIX_Studio_5.6

At least 118.5 MB of free disk space is required.



Setup - GUIX Studio

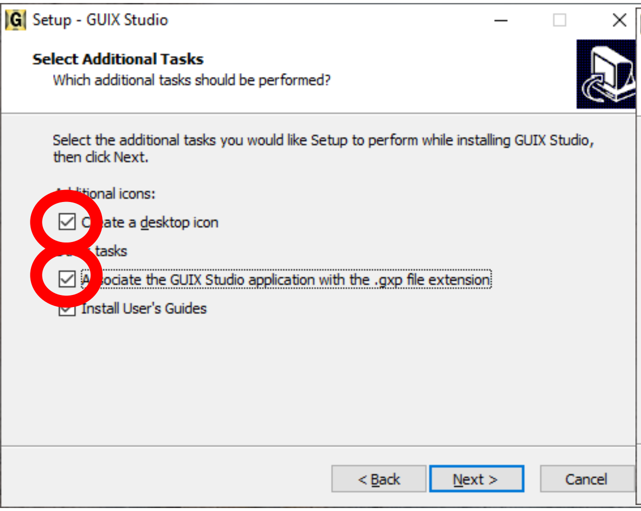
Select Start Menu Folder

Where should Setup place the program's shortcuts?

Setup will create the program's shortcuts in the following Start Menu folder.

To continue, click Next. If you would like to select a different folder, click Browse.

Express Logic



Setup - GUIX Studio

Select Additional Tasks

Which additional tasks should be performed?

Select the additional tasks you would like Setup to perform while installing GUIX Studio, then click Next.

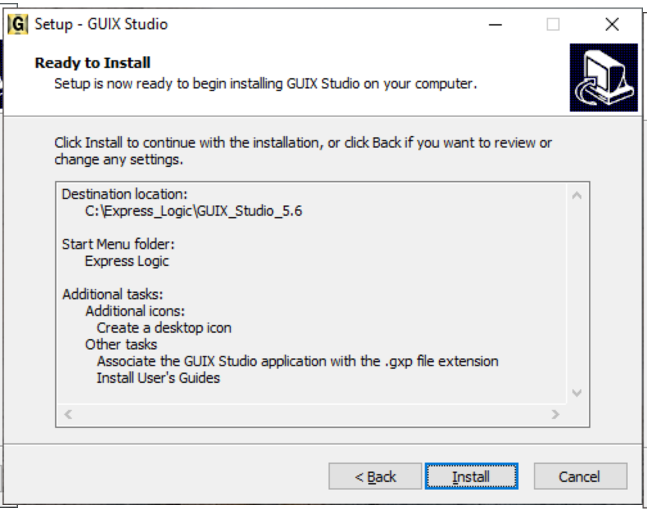
Additional icons:

☑ Create a desktop icon

Other tasks

☑ Associate the GUIX Studio application with the .gxp file extension

☑ Install User's Guides



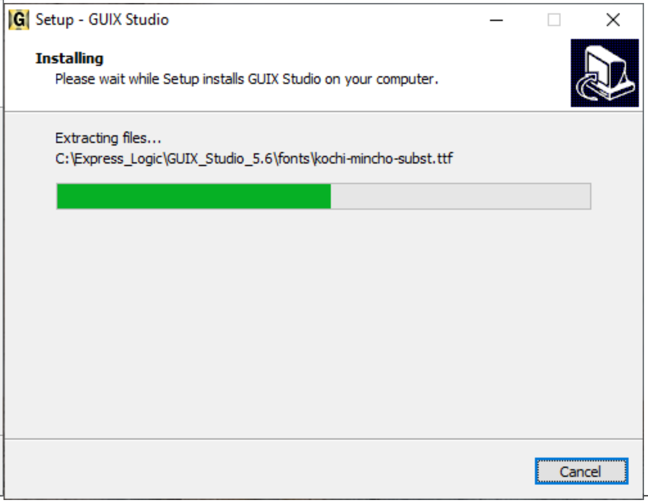
Setup - GUIX Studio

Ready to Install

Setup is now ready to begin installing GUIX Studio on your computer.

Click Install to continue with the installation, or click Back if you want to review or change any settings.

```
Destination location:
    C:\Express_Logic\GUIX_Studio_5.6

Start Menu folder:
    Express Logic

Additional tasks:
    Additional icons:
       Create a desktop icon
    Other tasks
       Associate the GUIX Studio application with the .gxp file extension
       Install User's Guides
```



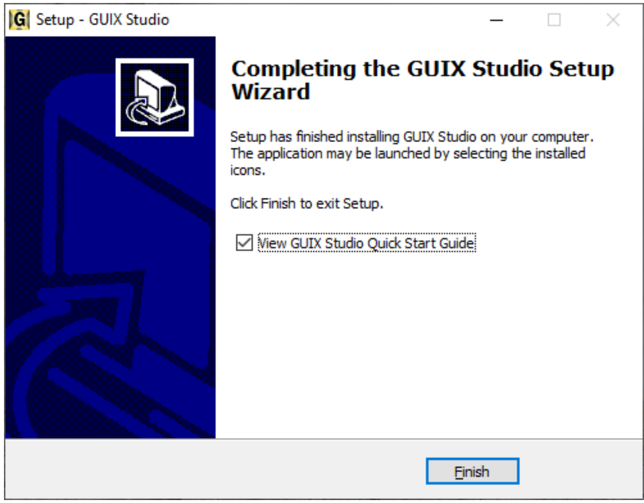
Setup - GUIX Studio

Installing

Please wait while Setup installs GUIX Studio on your computer.

Extracting files...

C:\Express_Logic\GUIX_Studio_5.6\fonts\kochi-mincho-subst.ttf



Setup - GUIX Studio

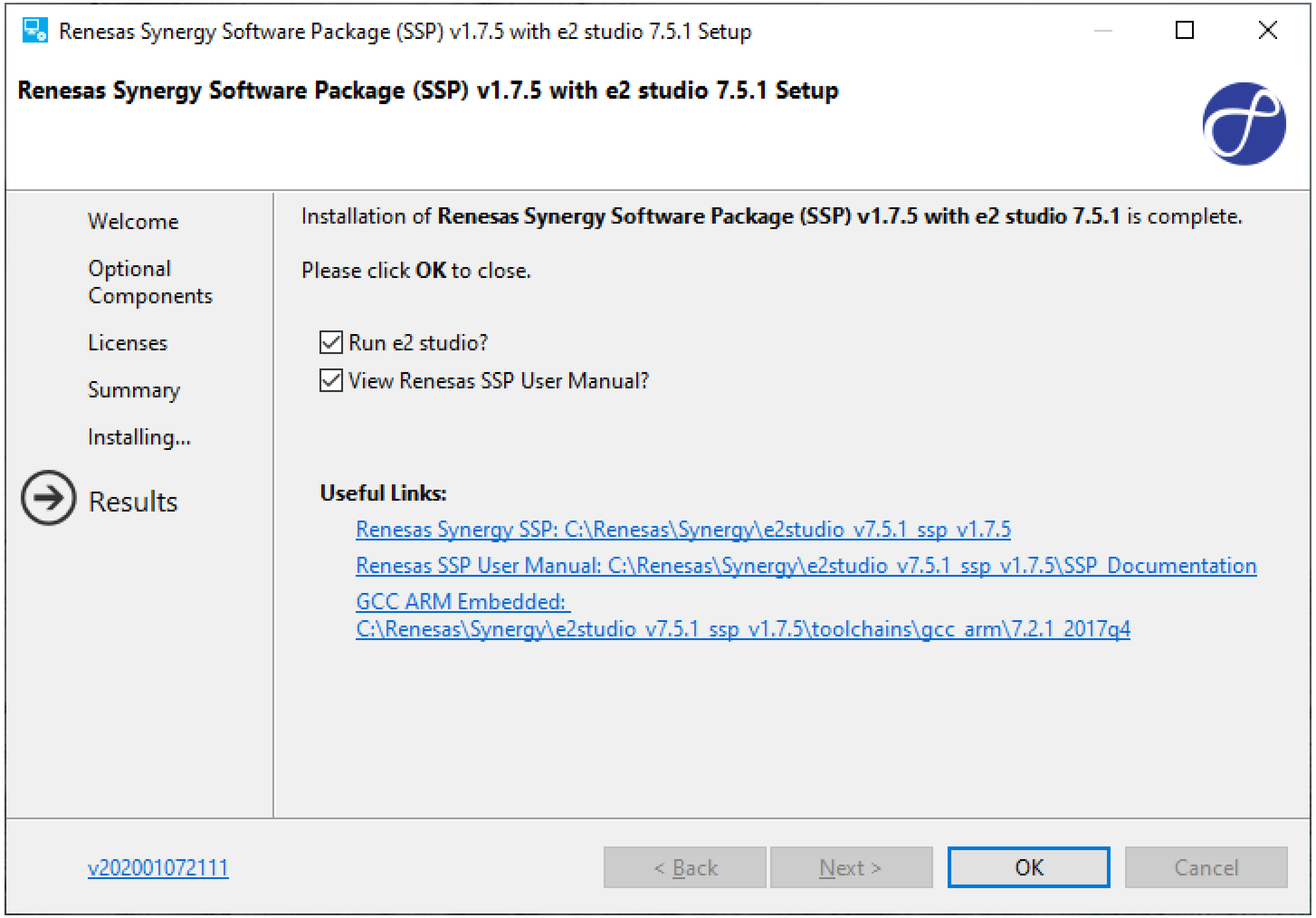
Completing the GUIX Studio Setup Wizard

Setup has finished installing GUIX Studio on your computer. The application may be launched by selecting the installed icons.

Click Finish to exit Setup.

☑ View GUIX Studio Quick Start Guide

LAB 1 – ACTIVITY 1 – STEP 4



Renesas Synergy Software Package (SSP) v1.7.5 with e2 studio 7.5.1 Setup

Renesas Synergy Software Package (SSP) v1.7.5 with e2 studio 7.5.1 Setup

Installation of **Renesas Synergy Software Package (SSP) v1.7.5 with e2 studio 7.5.1** is complete.

Please click **OK** to close.

☑ Run e2 studio?

☑ View Renesas SSP User Manual?

Useful Links:

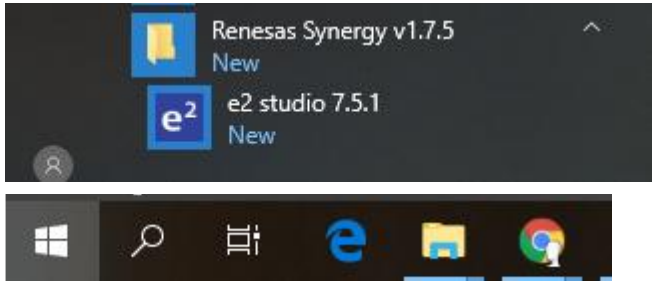
Renesas Synergy SSP: C:\Renesas\Synergy\e2studio_v7.5.1_ssp_v1.7.5

Renesas SSP User Manual: C:\Renesas\Synergy\e2studio_v7.5.1_ssp_v1.7.5\SSP_Documentation

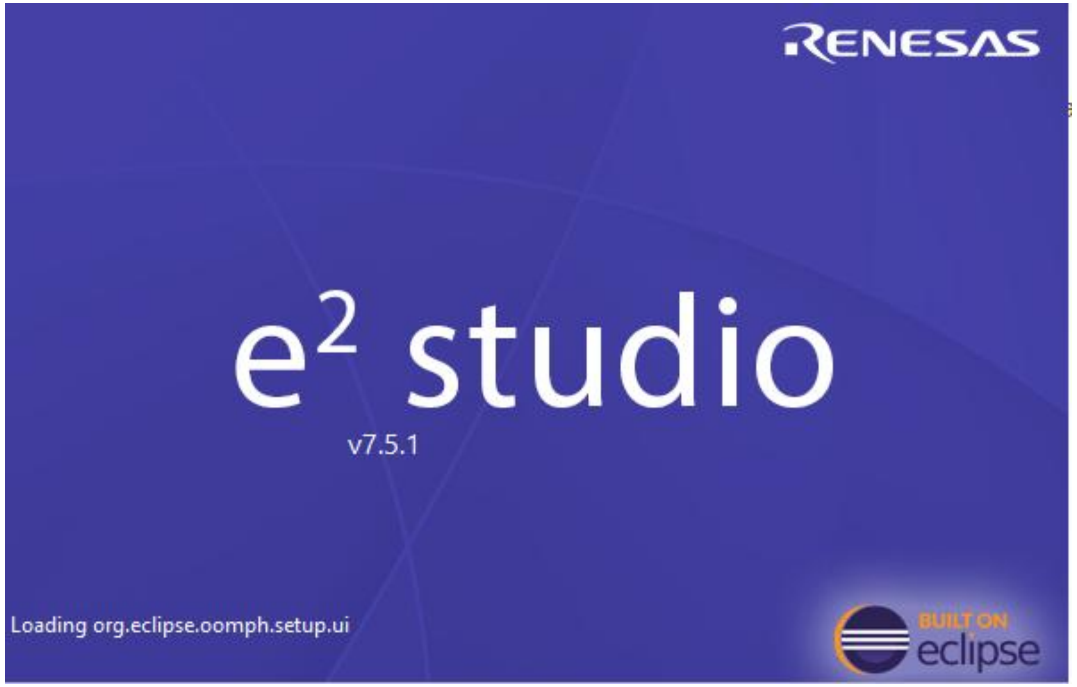
GCC ARM Embedded: C:\Renesas\Synergy\e2studio_v7.5.1_ssp_v1.7.5\toolchains\gcc_arm\7.2.1_2017q4

v202001072111

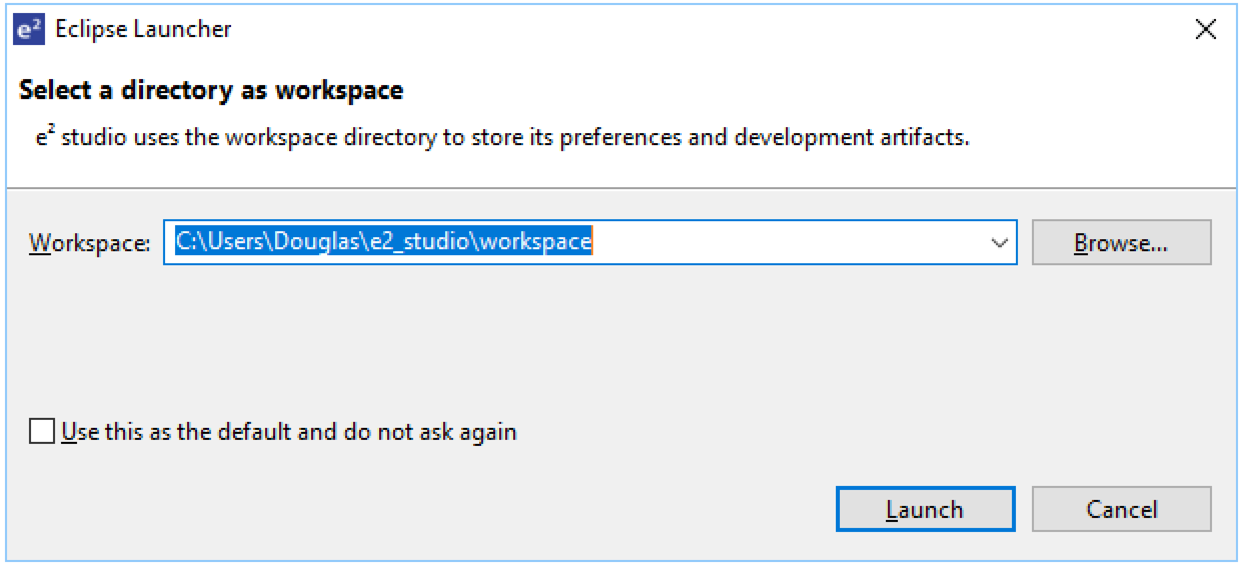
LAB 1 – ACTIVITY 1 – STEP 5



execute e2studio: C:\Renesas\Synergy\e2studio_v7.5.1_ssp_v1.7.5\eclipse\e2studio.exe



Loading org.eclipse.oomph.setup.ui



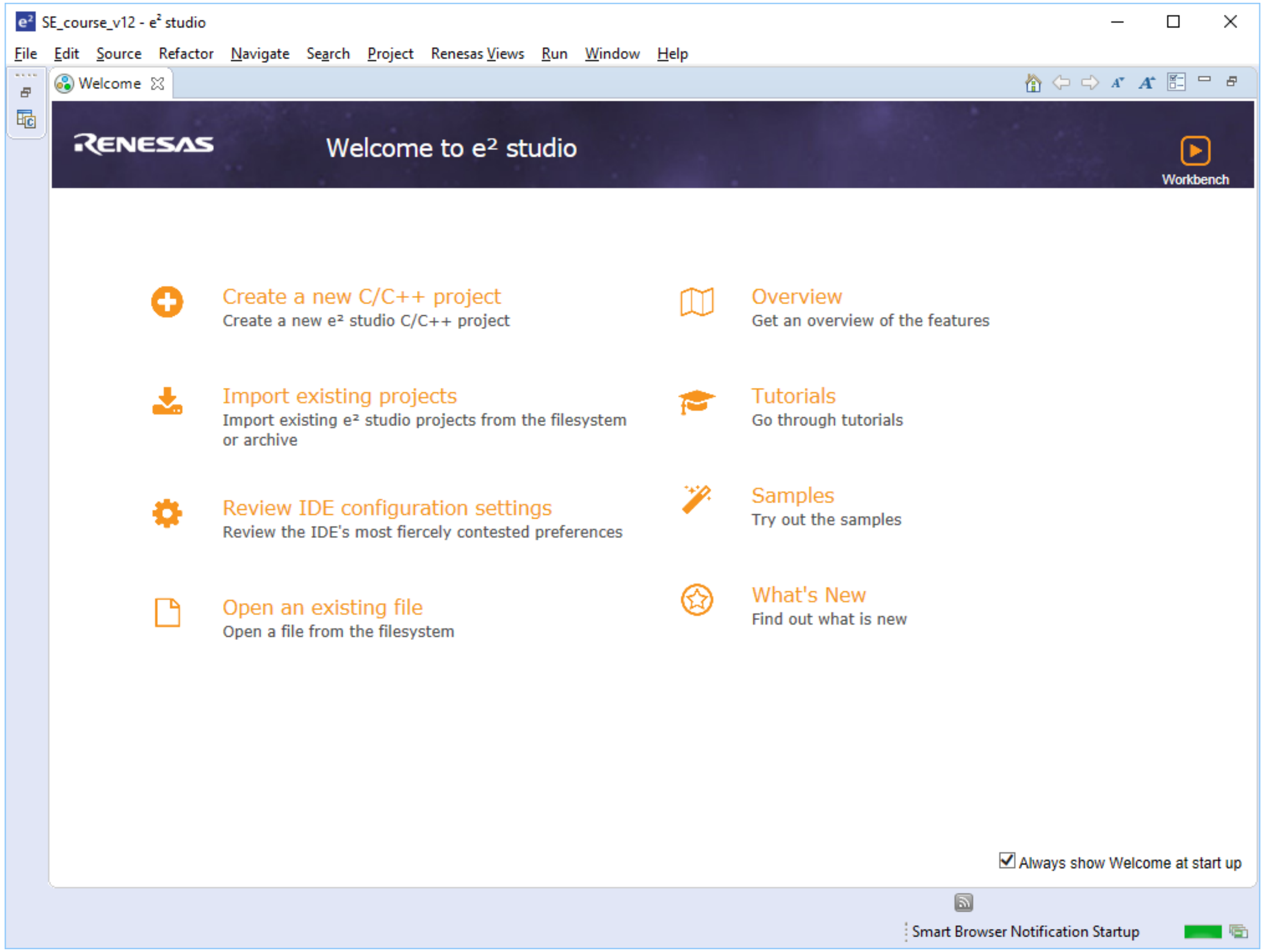
Select a directory as workspace

e² studio uses the workspace directory to store its preferences and development artifacts.

Workspace: C:\Users\Douglas\e2_studio\workspace

□ Use this as the default and do not ask again

LAB 1 – ACTIVITY 1 – STEP 5



SE_course_v12 - e² studio
File Edit Source Refactor Navigate Search Project Renesas Views Run Window Help
Welcome
RENESAS
Workbench

Welcome to e² studio

Create a new C/C++ project
Create a new e² studio C/C++ project

Import existing projects
Import existing e² studio projects from the filesystem or archive

Review IDE configuration settings
Review the IDE's most fiercely contested preferences

Open an existing file
Open a file from the filesystem

Overview
Get an overview of the features

Tutorials
Go through tutorials

Samples
Try out the samples

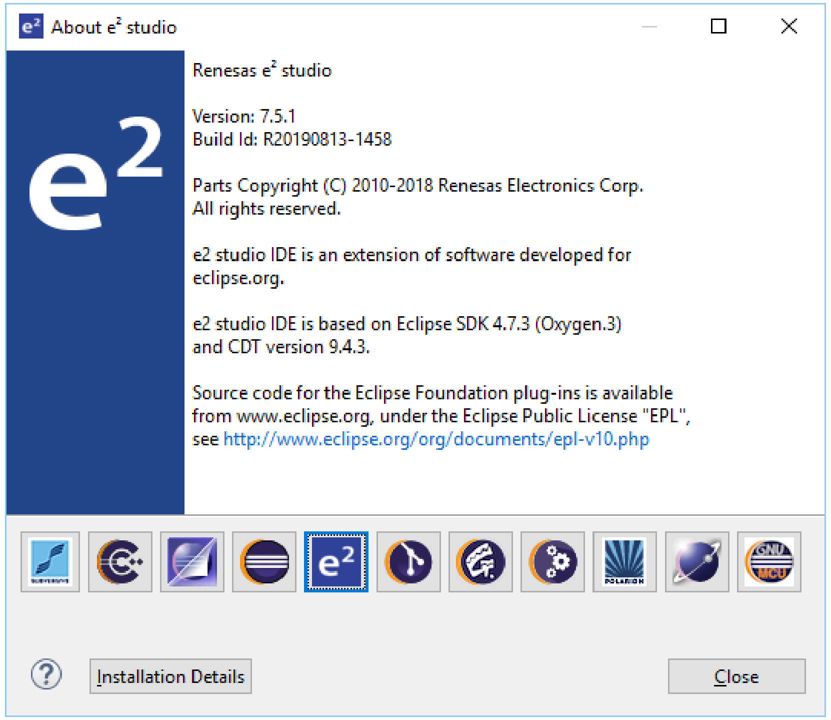
What's New
Find out what is new

☑ Always show Welcome at start up

Smart Browser Notification Startup

LAB 1 – ACTIVITY 1 – LEARN YOUR CONFIGURATION!

- From e2studio | Help | About - get the version of e2studio: 7.5.1
- In C:\Renesas\Synergy\e2studio_v7.5.1_ssp_v1.7.5 the folder name indicates the version of the SSP: 1.7.5
- In C:\Renesas\Synergy\e2studio_v7.5.1_ssp_v1.7.5\toolchains\gcc_arm see the version of the GCC tools: 7.2.1.2017
- From the microcontroller chip on the development board: S7G27H3CFC



Renesas e² studio

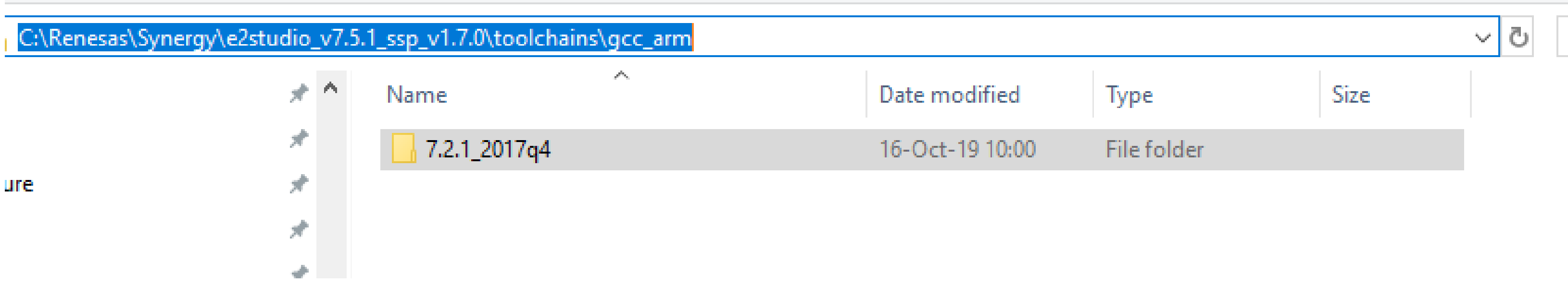
Version: 7.5.1
Build Id: R20190813-1458

Parts Copyright (C) 2010-2018 Renesas Electronics Corp.
All rights reserved.

e2 studio IDE is an extension of software developed for eclipse.org.

e2 studio IDE is based on Eclipse SDK 4.7.3 (Oxygen.3) and CDT version 9.4.3.

Source code for the Eclipse Foundation plug-ins is available from www.eclipse.org, under the Eclipse Public License "EPL", see http://www.eclipse.org/org/documents/epl-v10.php



C:\Renesas\Synergy\e2studio_v7.5.1_ssp_v1.7.0\toolchains\gcc_arm

Name	Date modified	Type	Size
7.2.1_2017q4	16-Oct-19 10:00	File folder	

LAB1 – SYNERGY PLATFORM

Activity 2 – Setup the SK-S7G2 board

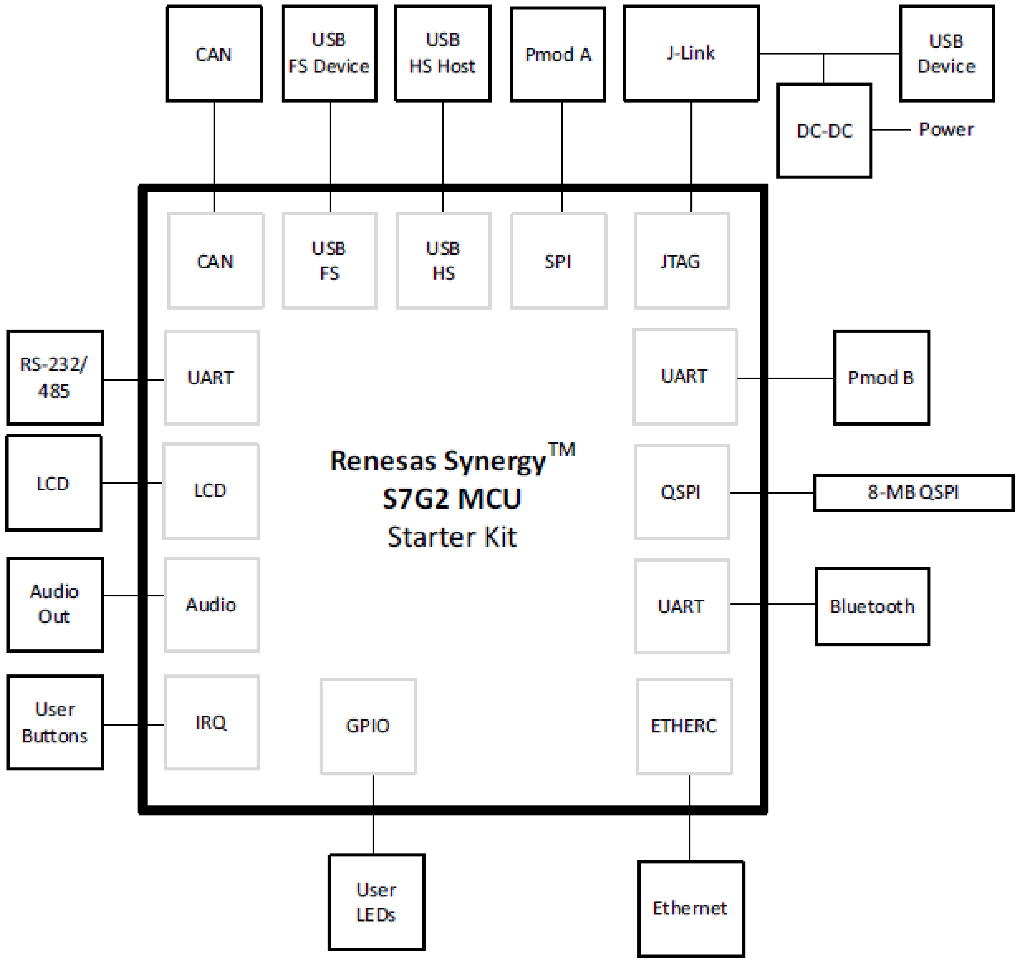
1. Verify the position of the jumpers on the board.
2. Connect the micro-USB cable to the DEBUG_USB port on the board and to your PC.
3. Verify the setup of the board and e2Studio by running a sample demo (Blinky).

source: Renesas.com

SK-S7G2 BOARD FEATURES

1. Uses a 176-pin LQFP MCU
   240 MHz S7G2 Cortex-M4 based microcontroller.
2. All pins of the MCU are accessible via header pins (access to CAN, SPI, UART, I2C).
3. Resistive touch TFT colour LCD (320 x 240 pixels).
4. Capacitive touch slider.
5. Connectors:
   1. USB Host - High speed
   2. USB Device - Full speed.
   3. Ethernet 10/100
   4. 2 PMOD type 2A
   5. audio connector
   6. CAN, RS-232/RS-485
   7. Arduino shields compatible connector
6. Wireless connectivity: on-board Bluetooth Low Energy (BLE) device
7. QSPI flash memory (8 MBytes).
8. USB - JTAG (on-board Segger JLink).
9. 3 user leds and 2 push buttons.
10. WiFi connectivity using add-on boards (e.g. AE-CLOUD1).

SK-S7G2 board block diagram



source: Renesas Starter Kit SK-S7G2 User's Manual

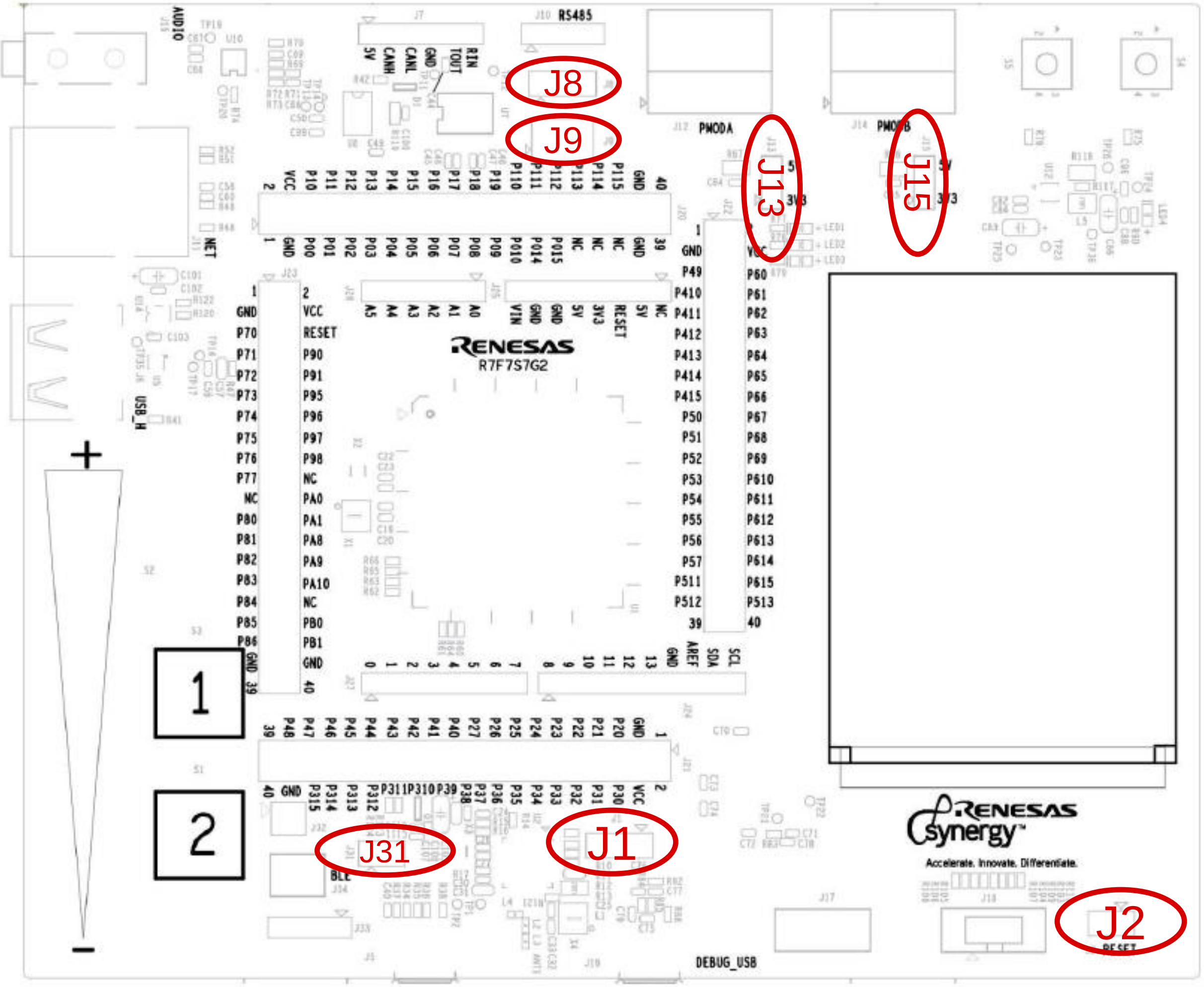
LAB 1 – ACTIVITY 2 – STEP 1

Verify the position of the jumpers on the board:

Jumper	Position
J1	1-2: normal boot
J2	open (reset released)
J8	1-2: signals a GPIO pin that RS-232 should be selected instead of RS-485
J9	1-3 and 2-4: use RS-232 driver
J13	1-2: output 3V3 to PMODA connector
J15	1-2: output 3V3 to PMODB connector
J31	closed - used for current measurement to MCU

pin 1 is marked by a triangle next to the connector

J9 (double row of pins is numbered as: 2 4 6
1 3 5



Suggested reading: Renesas Synergy Starter Kit SK-S7G2 User's Manual (r12um0004eu0100)

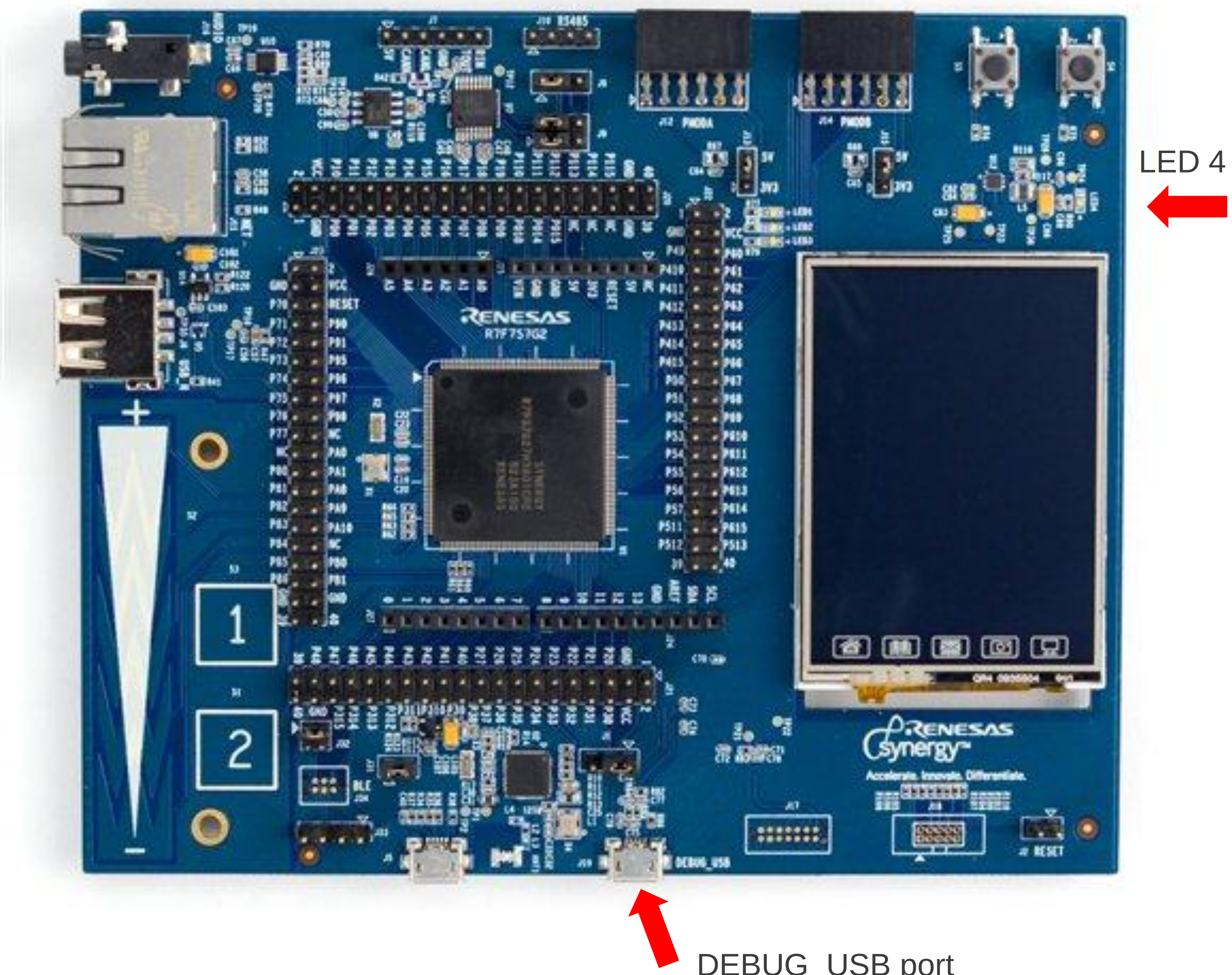
(https://www.renesas.com/us/en/doc/products/renesas-synergy/doc/r12um0004eu0100_synergy_sk_s7g2.pdf)

this manual has detailed explanation on the board functionality, schematics and configuration

LAB 1 – ACTIVITY 2 – STEP 2

Connect the micro-USB cable to the DEBUG_USB port on the board and to your PC.

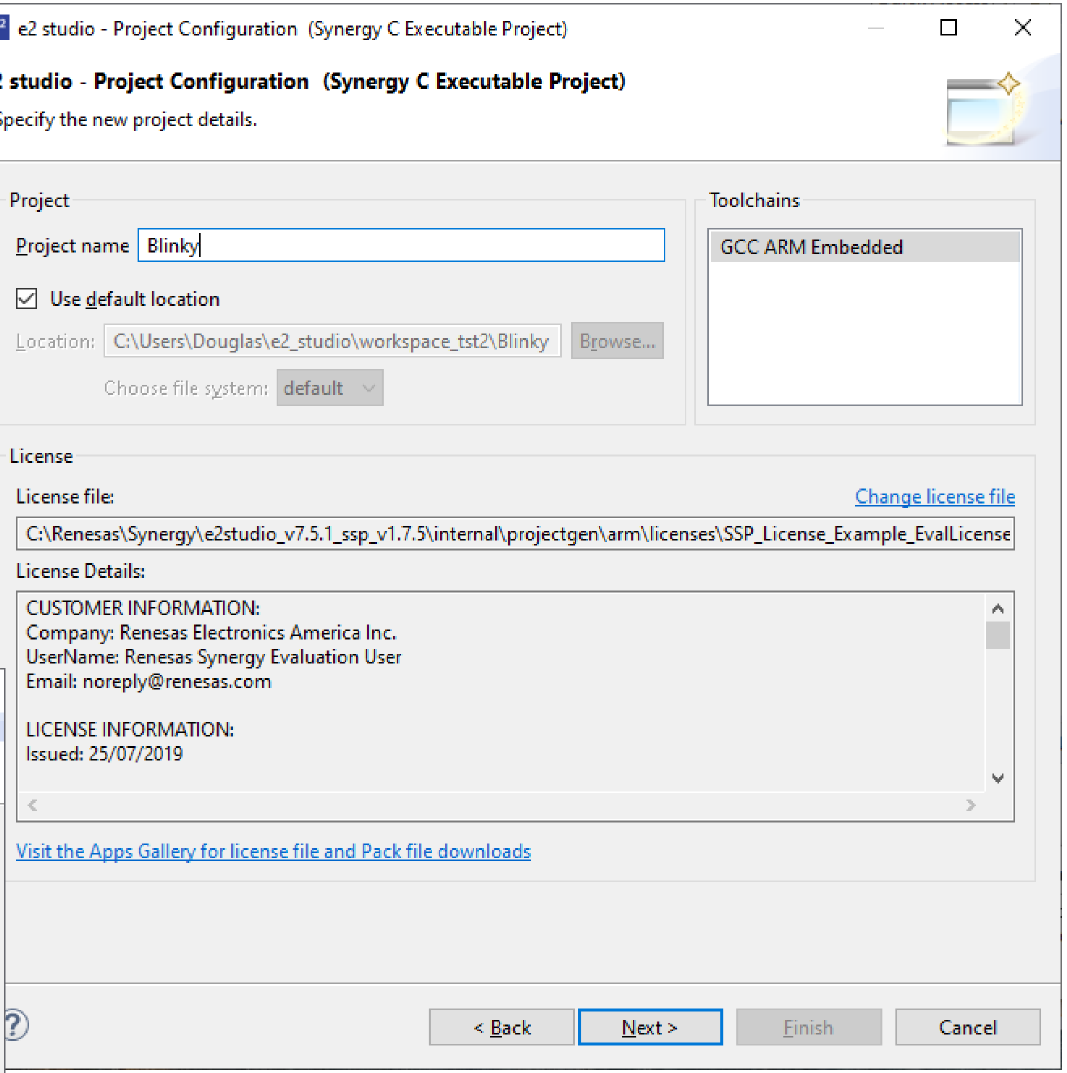
LED 4 should light up to indicate that the board powered up.



source: Renesas Starter Kit SK-S7G2 User’s Manual

LAB 1 – ACTIVITY 2 – STEP 3

1. Run e2studio
   (StartMenu | Renesas Synergy v1.7.5 | e2 studio 7.5.1)
   if the welcome screen shows up then click on Workbench.
2. File | New | Synergy C/C++ Project
   Renesas Synergy C Executable Project
   Project name: Blinky


e2 studio - Project Configuration (Synergy C Executable Project)

e2 studio - Project Configuration (Synergy C Executable Project)

Specify the new project details.

Project

Project name: Blinky

☑ Use default location

Location: C:\Users\Douglas\e2_studio\workspace_tst2\Blinky

Choose file system: default

Toolchains: GCC ARM Embedded

License

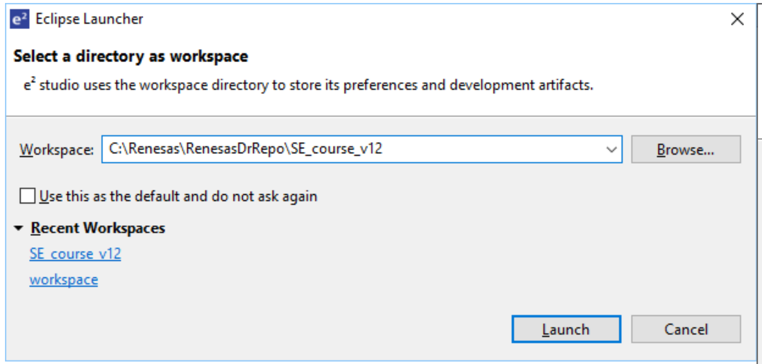
License file: C:\Renesas\Synergy\e2studio_v7.5.1_ssp_v1.7.5\internal\projectgen\arm\licenses\SSP_License_Example_EvalLicense

License Details:

CUSTOMER INFORMATION:
Company: Renesas Electronics America Inc.
UserName: Renesas Synergy Evaluation User
Email: noreply@renesas.com

LICENSE INFORMATION:
Issued: 25/07/2019

Visit the Apps Gallery for license file and Pack file downloads


Eclipse Launcher

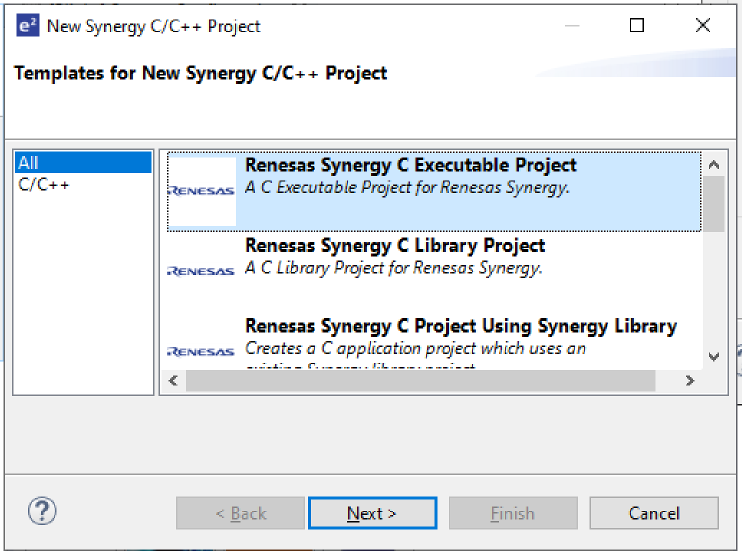
Select a directory as workspace

e2 studio uses the workspace directory to store its preferences and development artifacts.

Workspace: C:\Renesas\RenesasDrRepo\SE_course_v12

□ Use this as the default and do not ask again

Recent Workspaces
SE_course_v12
workspace


New Synergy C/C++ Project

Templates for New Synergy C/C++ Project

All
C/C++

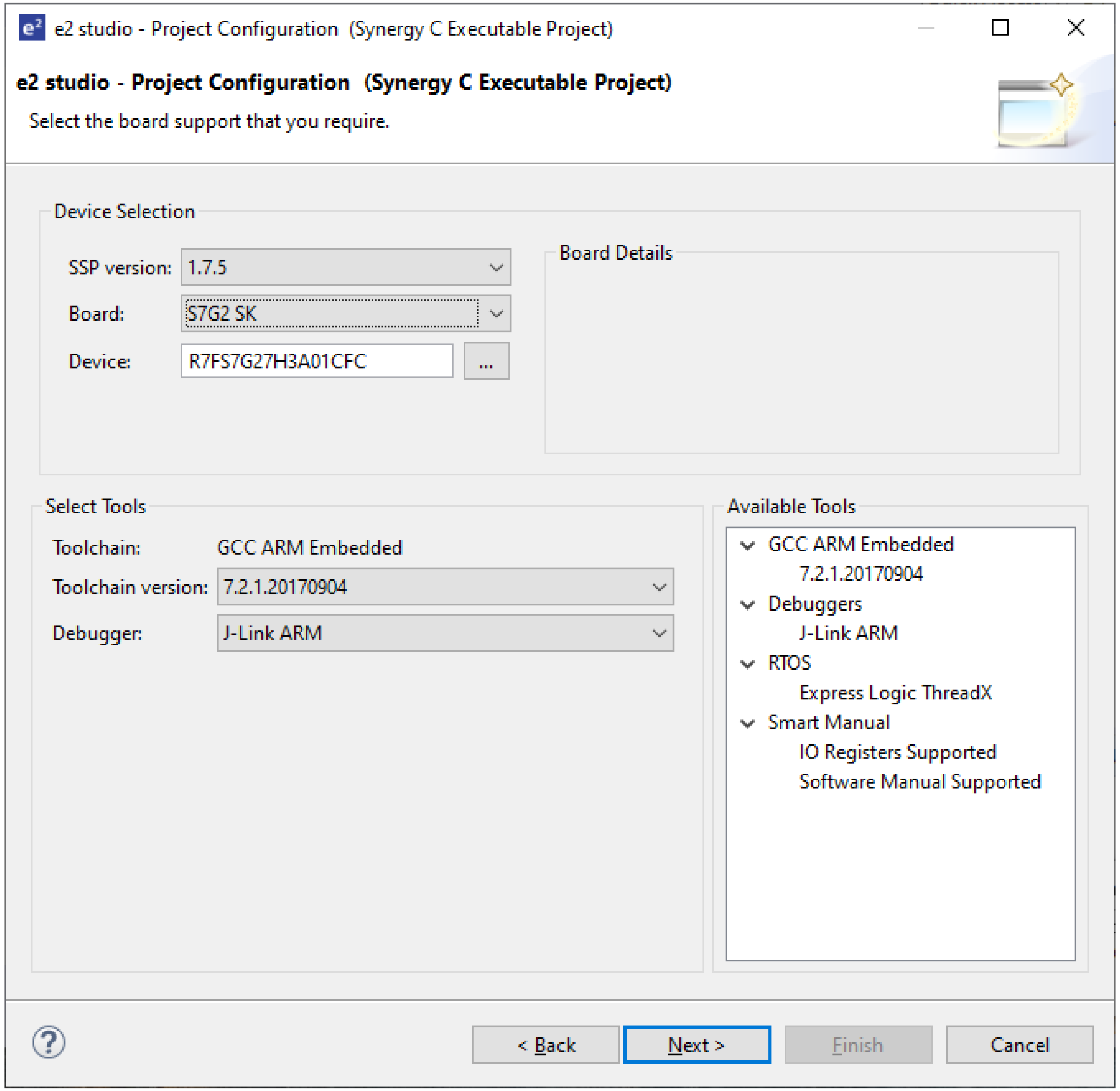
Renesas Synergy C Executable Project
A C Executable Project for Renesas Synergy.

Renesas Synergy C Library Project
A C Library Project for Renesas Synergy.

Renesas Synergy C Project Using Synergy Library
Creates a C application project which uses an existing Synergy library project

LAB 1 – ACTIVITY 2 – STEP 3

3. Select board: S7G2 SK
   Select the MCU that is used in your board
   Select the toolchain version that was installed during Activity 1 of this Lab
   Select J-Link ARM as the debugger interface



e2 studio - Project Configuration (Synergy C Executable Project)

e2 studio - Project Configuration (Synergy C Executable Project)

Select the board support that you require.

Device Selection

SSP version: 1.7.5

Board: S7G2 SK

Device: R7FS7G27H3A01CFC

Board Details

Select Tools

Toolchain: GCC ARM Embedded

Toolchain version: 7.2.1.20170904

Debugger: J-Link ARM

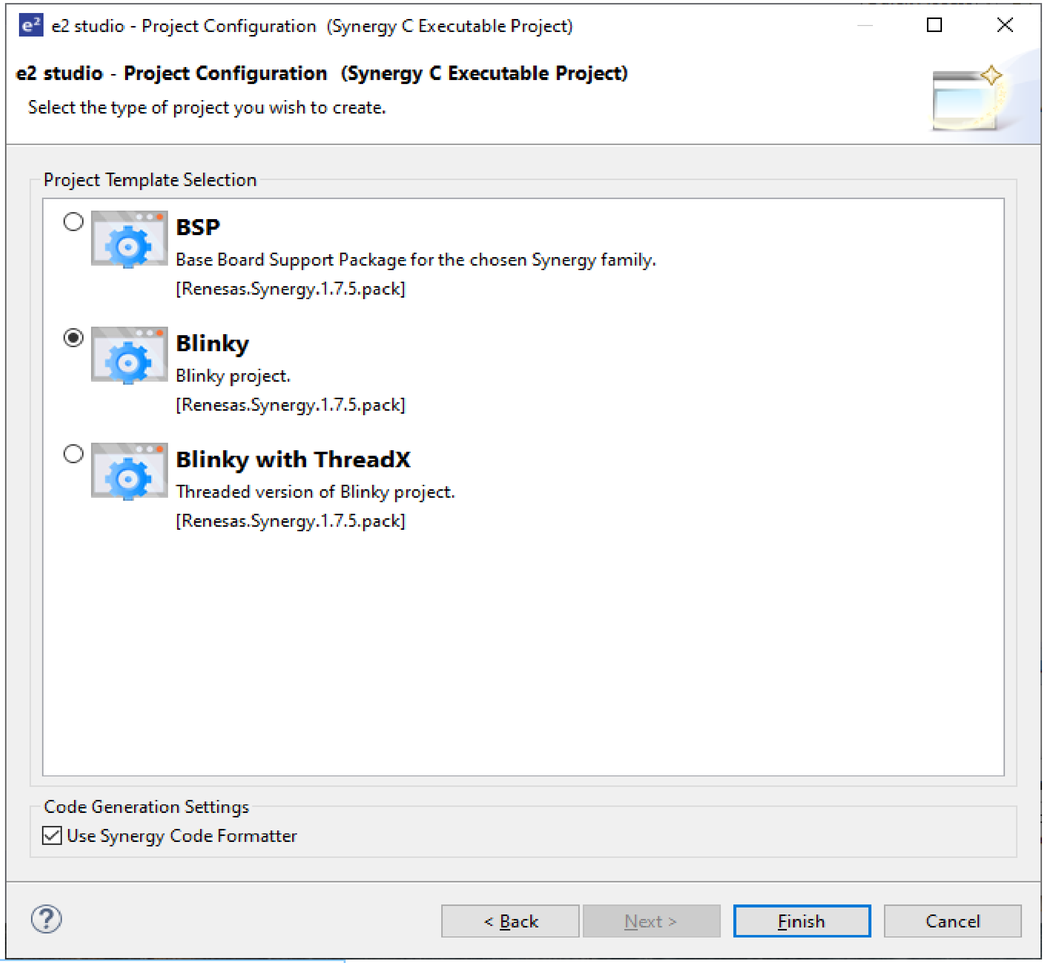
Available Tools

- GCC ARM Embedded
  - 7.2.1.20170904
- Debuggers
  - J-Link ARM
- RTOS
  - Express Logic ThreadX
- Smart Manual
  - IO Registers Supported
  - Software Manual Supported

LAB 1 – ACTIVITY 2 – STEP 3

4. Select **Blinky** as the Template
   Finish (press Finish button)

   accept the suggested perspective



e2 studio - Project Configuration (Synergy C Executable Project)

e2 studio - Project Configuration (Synergy C Executable Project)

Select the type of project you wish to create.

Project Template Selection

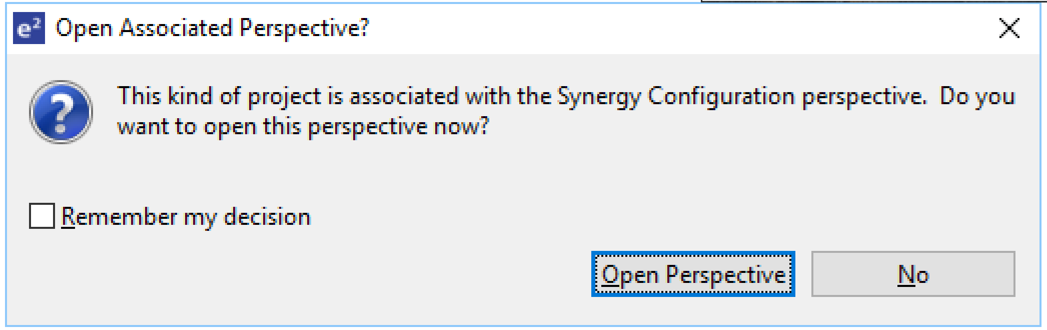
○ **BSP**
Base Board Support Package for the chosen Synergy family.
[Renesas.Synergy.1.7.5.pack]

● **Blinky**
Blinky project.
[Renesas.Synergy.1.7.5.pack]

○ **Blinky with ThreadX**
Threaded version of Blinky project.
[Renesas.Synergy.1.7.5.pack]

Code Generation Settings

☑ Use Synergy Code Formatter



Open Associated Perspective?

This kind of project is associated with the Synergy Configuration perspective. Do you want to open this perspective now?

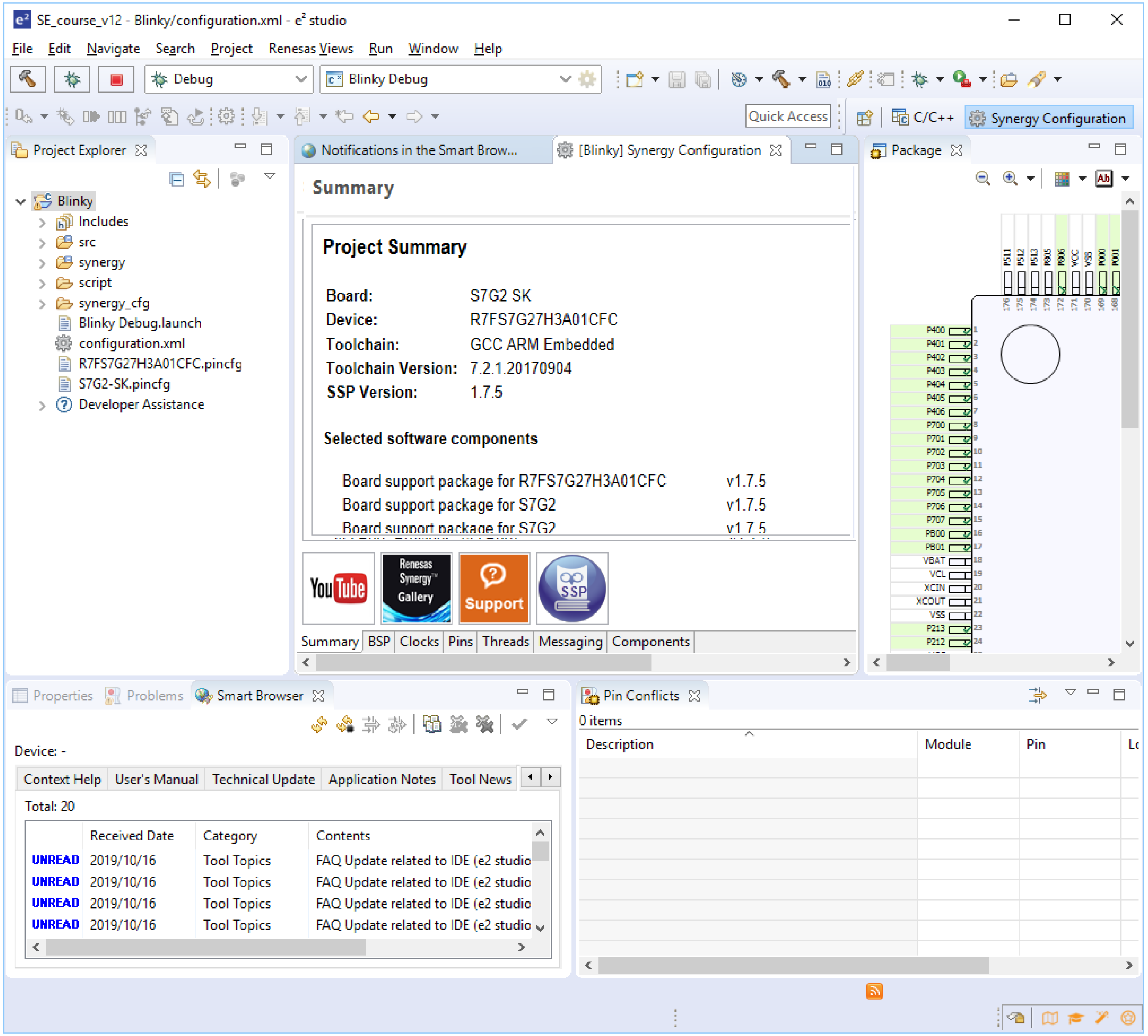
☐ Remember my decision

LAB 1 – ACTIVITY 2 – STEP 3

5. On the Project Explorer
   right click on the Blinky project
   Build Project
   (should compile without errors or warnings).

6. Right click again on Blinky project
   Debug As | Debug Configurations...

7. On the Debug Configurations Window
   expand Renesas GDB Hardware Debugging
   select Blinky Debug.
   Debug (press Debug button)
   if offered to upgrade the J-Link firmware, accept
   if offered to change perspective, accept.



Project Summary

Board:	S7G2 SK
Device:	R7FS7G27H3A01CFC
Toolchain:	GCC ARM Embedded
Toolchain Version:	7.2.1.20170904
SSP Version:	1.7.5

Selected software components

Board support package for R7FS7G27H3A01CFC	v1.7.5
Board support package for S7G2	v1.7.5
Board support package for S7G2	v1.7.5

Device: -

Context Help | User's Manual | Technical Update | Application Notes | Tool News

Total: 20

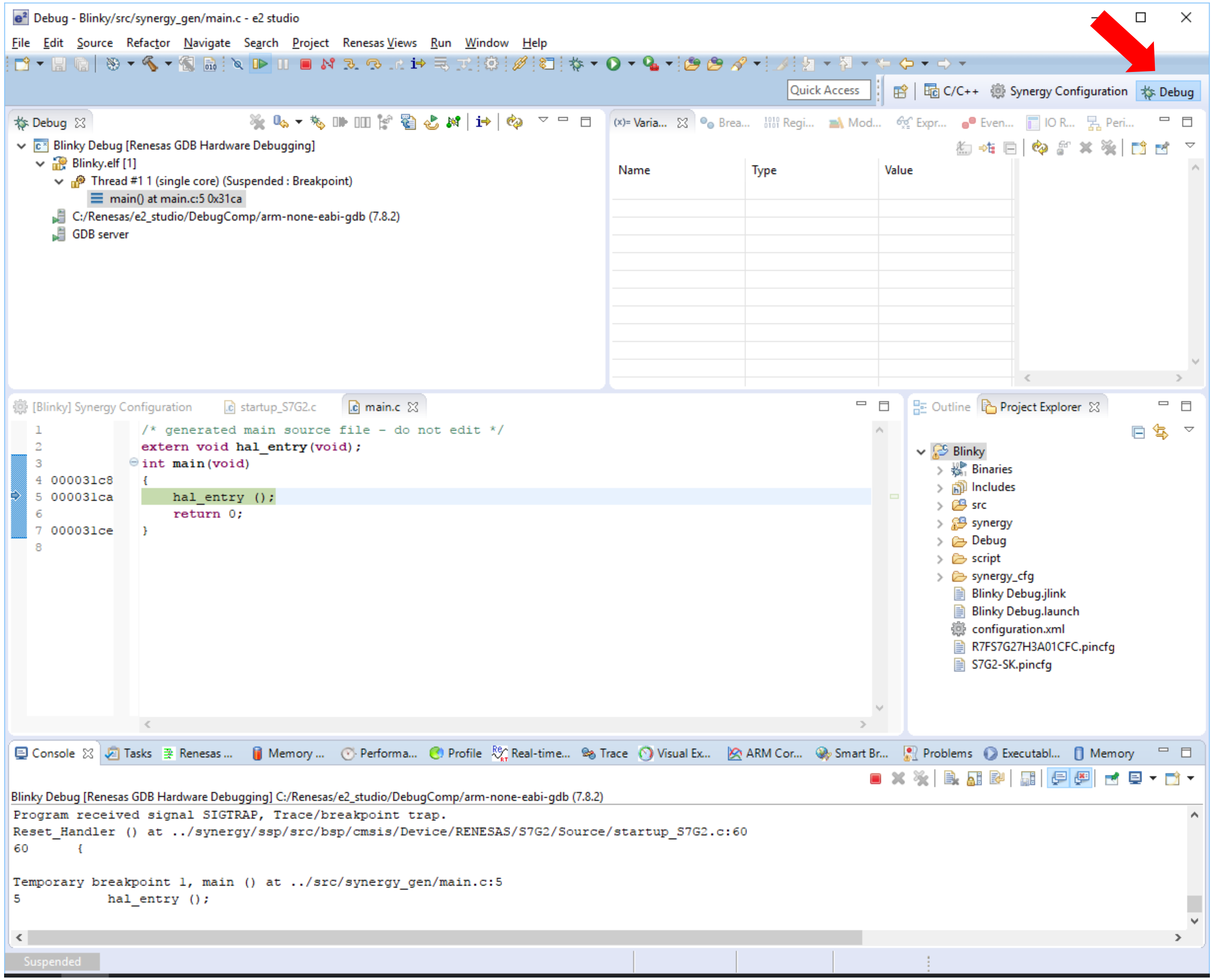
	Received Date	Category	Contents
UNREAD	2019/10/16	Tool Topics	FAQ Update related to IDE (e2 studio
UNREAD	2019/10/16	Tool Topics	FAQ Update related to IDE (e2 studio
UNREAD	2019/10/16	Tool Topics	FAQ Update related to IDE (e2 studio
UNREAD	2019/10/16	Tool Topics	FAQ Update related to IDE (e2 studio

LAB 1 – ACTIVITY 2 – STEP 3

8. The debug perspective should be presented at this time (see ⚙ Debug on upper right)

9. Press Resume (▷) and the code is executed up to the start of the main function - green line

10. Press Resume again and the code executes: three leds blink near the LCD of the board. You succeeded in the installation of e2studio!



```
Blinky Debug [Renesas GDB Hardware Debugging]
  Blinky.elf [1]
    Thread #1 1 (single core) (Suspended : Breakpoint)
      main() at main.c:5 0x31ca
  C:/Renesas/e2_studio/DebugComp/arm-none-eabi-gdb (7.8.2)
  GDB server
```

```
                /* generated main source file - do not edit */
                extern void hal_entry(void);
                int main(void)
  000031c8      {
  000031ca          hal_entry ();
                    return 0;
  000031ce      }
```

```
Blinky Debug [Renesas GDB Hardware Debugging] C:/Renesas/e2_studio/DebugComp/arm-none-eabi-gdb (7.8.2)
Program received signal SIGTRAP, Trace/breakpoint trap.
Reset_Handler () at ../synergy/ssp/src/bsp/cmsis/Device/RENESAS/S7G2/Source/startup_S7G2.c:60
60      {

Temporary breakpoint 1, main () at ../src/synergy_gen/main.c:5
5           hal_entry ();
```

LAB1 – SYNERGY PLATFORM

Activity 3 – Overview of the Synergy Platform

1. How platform is defined in this context
2. What is included in the Synergy Platform
3. Effect on the development process

LAB1 – ACTIVITY 3

Concept of the Synergy Platform

In this context, **Platform** is defined as a complete **set of hardware and software components** that can be easily combined to form the foundation upon which a solution is built. Furthermore, the Platform also provides the infrastructure for development (development tools, example of end-product design, technology building-block examples, web accessible repository of tools and software).

LAB1 – ACTIVITY 3

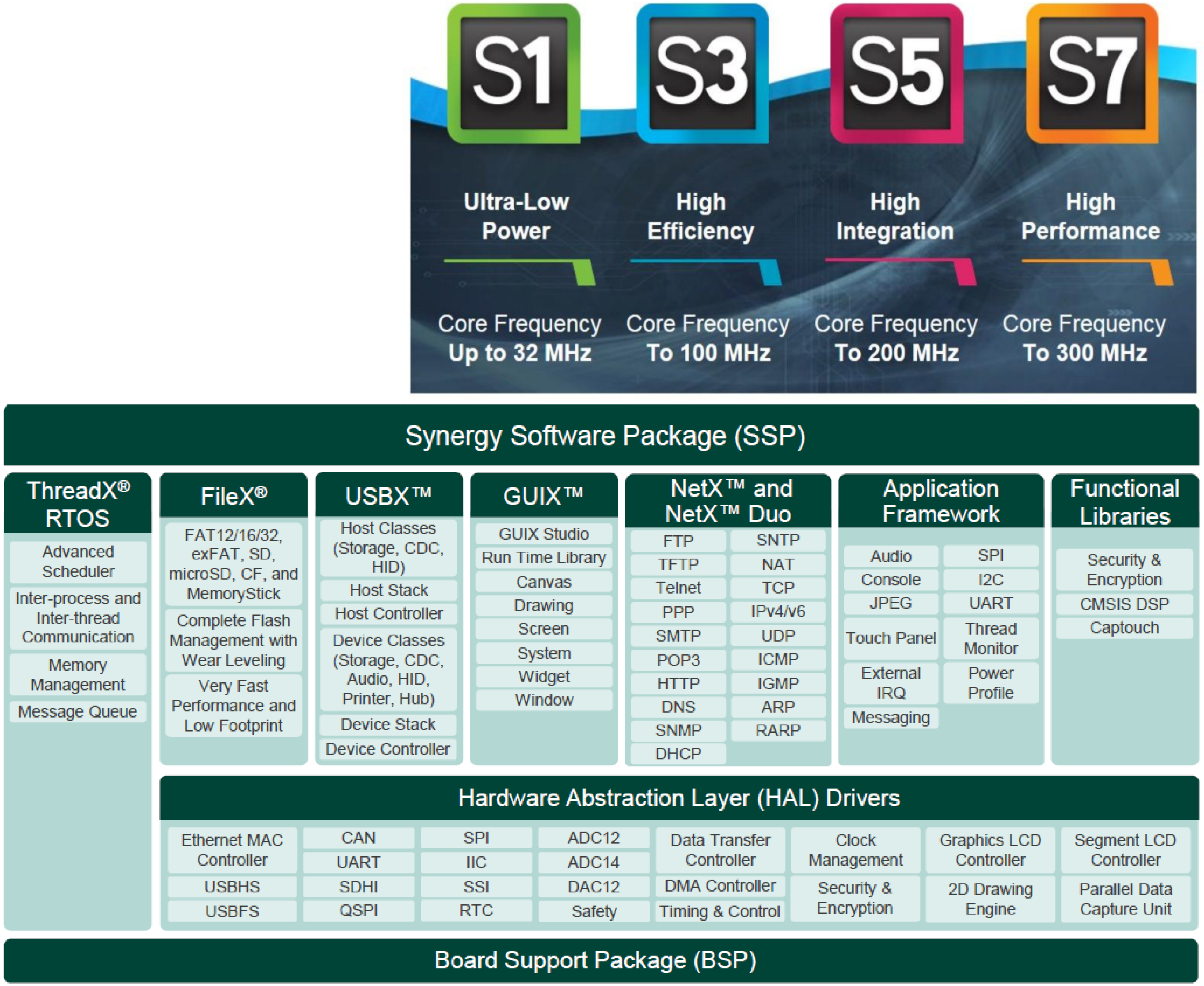
What is included in the Synergy Platform?

1. A family of Microcontrollers (S1, S3, S5, S7)
2. SSP (Synergy Software Package)

   complete package of qualified software
3. Development tools

   (e.g. the e2studio ISDE)
4. Development Boards (e.g. SK-S7G2)
5. Product examples
6. Application examples
7. Synergy Gallery

   (web accessible repository)



S1 – Ultra-Low Power – Core Frequency Up to 32 MHz

S3 – High Efficiency – Core Frequency To 100 MHz

S5 – High Integration – Core Frequency To 200 MHz

S7 – High Performance – Core Frequency To 300 MHz

Synergy Software Package (SSP)

ThreadX® RTOS: Advanced Scheduler; Inter-process and Inter-thread Communication; Memory Management; Message Queue

FileX®: FAT12/16/32, exFAT, SD, microSD, CF, and MemoryStick; Complete Flash Management with Wear Leveling; Very Fast Performance and Low Footprint

USBX™: Host Classes (Storage, CDC, HID); Host Stack; Host Controller; Device Classes (Storage, CDC, Audio, HID, Printer, Hub); Device Stack; Device Controller

GUIX™: GUIX Studio; Run Time Library; Canvas; Drawing; Screen; System; Widget; Window

NetX™ and NetX™ Duo: FTP; SNTP; TFTP; NAT; Telnet; TCP; PPP; IPv4/v6; SMTP; UDP; POP3; ICMP; HTTP; IGMP; DNS; ARP; SNMP; RARP; DHCP

Application Framework: Audio; SPI; Console; I2C; JPEG; UART; Touch Panel; Thread Monitor; External IRQ; Power Profile; Messaging

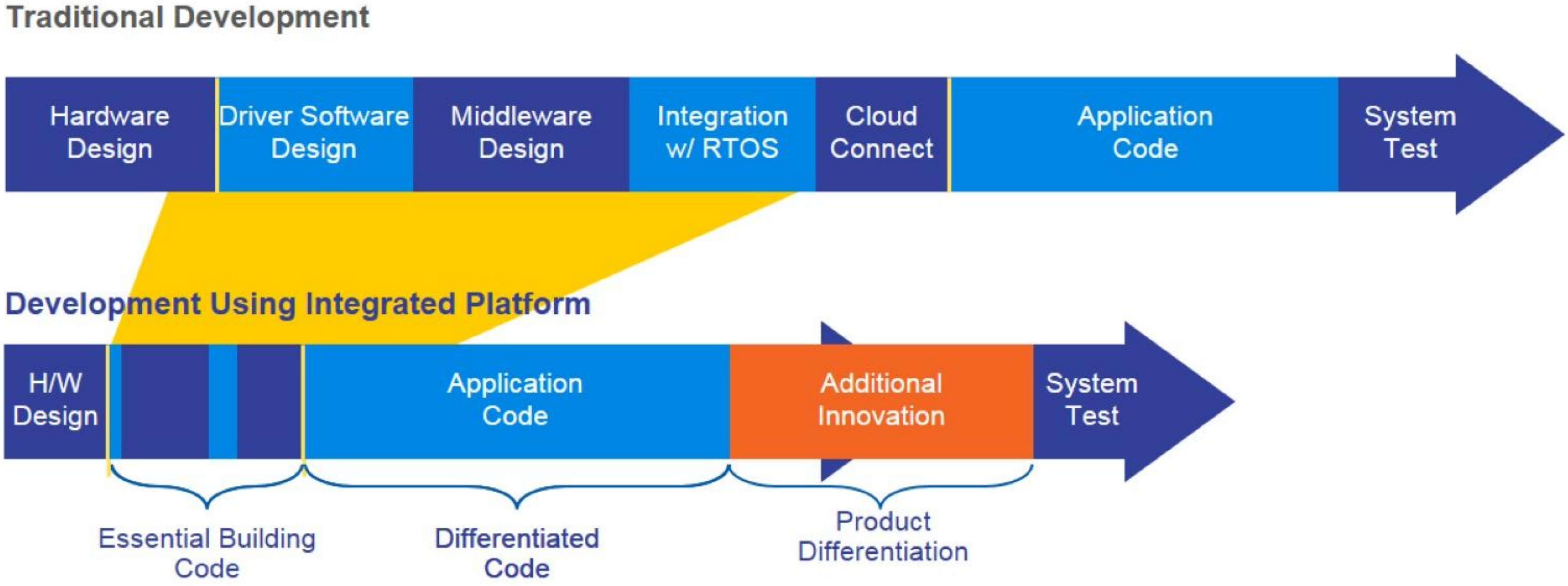
Functional Libraries: Security & Encryption; CMSIS DSP; Captouch

Hardware Abstraction Layer (HAL) Drivers: Ethernet MAC Controller; CAN; SPI; ADC12; Data Transfer Controller; Clock Management; Graphics LCD Controller; Segment LCD Controller; UART; IIC; ADC14; USBHS; SDHI; SSI; DAC12; DMA Controller; Security & Encryption; 2D Drawing Engine; Parallel Data Capture Unit; USBFS; QSPI; RTC; Safety; Timing & Control

Board Support Package (BSP)

source: Renesas, Synergy Platform Architecture

LAB1 – ACTIVITY 3



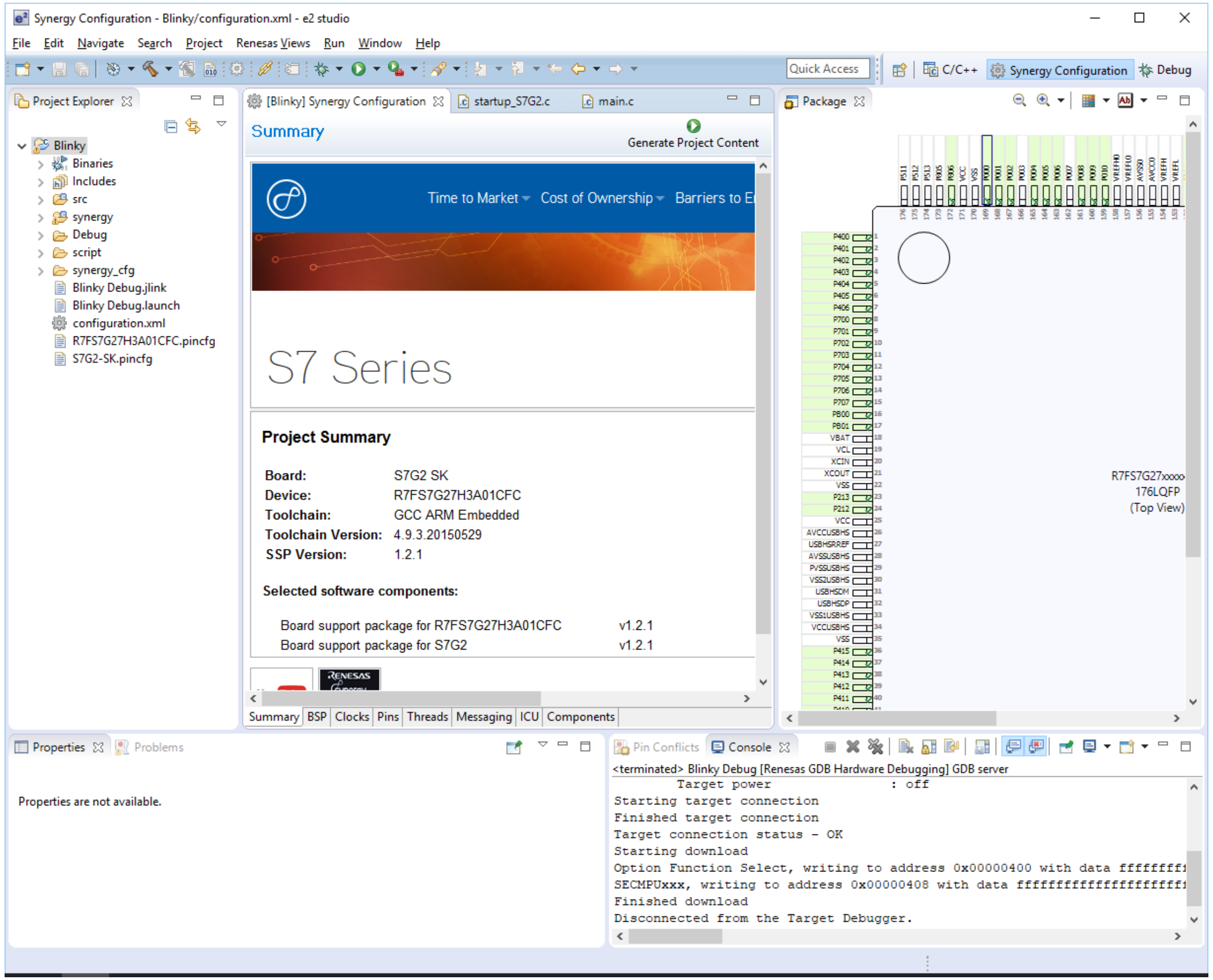
source: Renesas, Synergy Platform Architecture

LAB1 – SYNERGY PLATFORM

Activity 4 – Overview of e2studio

1. Understanding the organization of e2studio

2. Common tasks in e2studio



Project Summary

Board: S7G2 SK
Device: R7FS7G27H3A01CFC
Toolchain: GCC ARM Embedded
Toolchain Version: 4.9.3.20150529
SSP Version: 1.2.1

Selected software components:

Board support package for R7FS7G27H3A01CFC	v1.2.1
Board support package for S7G2	v1.2.1

```
<terminated> Blinky Debug [Renesas GDB Hardware Debugging] GDB server
        Target power                  : off
Starting target connection
Finished target connection
Target connection status - OK
Starting download
Option Function Select, writing to address 0x00000400 with data ffffffff
SECMPUxxx, writing to address 0x00000408 with data ffffffffffffffffffff
Finished download
Disconnected from the Target Debugger.
```

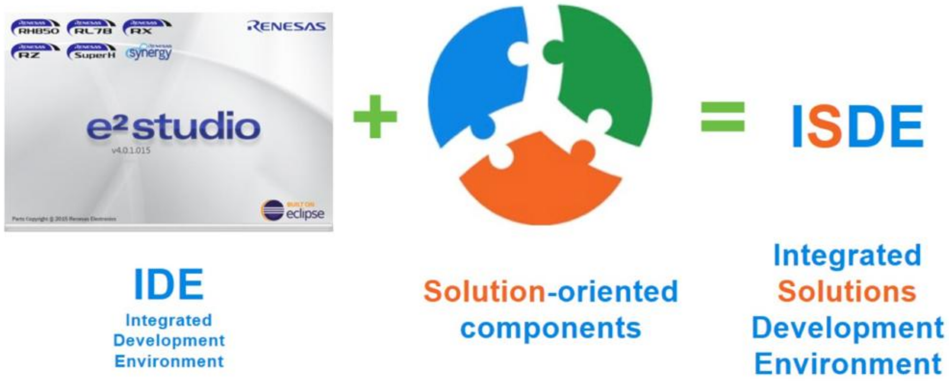
LAB1 – ACTIVITY 4

The e2studio ISDE

(Integrated Solutions Development Environment)

Features:

1. Built on the Eclipse Framework
2. Compiler/Linker - GNU or IAR
3. Editor (with syntax highlighting and auto-complete)
4. Configuration tools (pin, clock, interrupt, ...)
5. SSP module selector and configurator
6. RTOS awareness, execution profiler, tracing
7. JTAG debugger interface (J-Link)

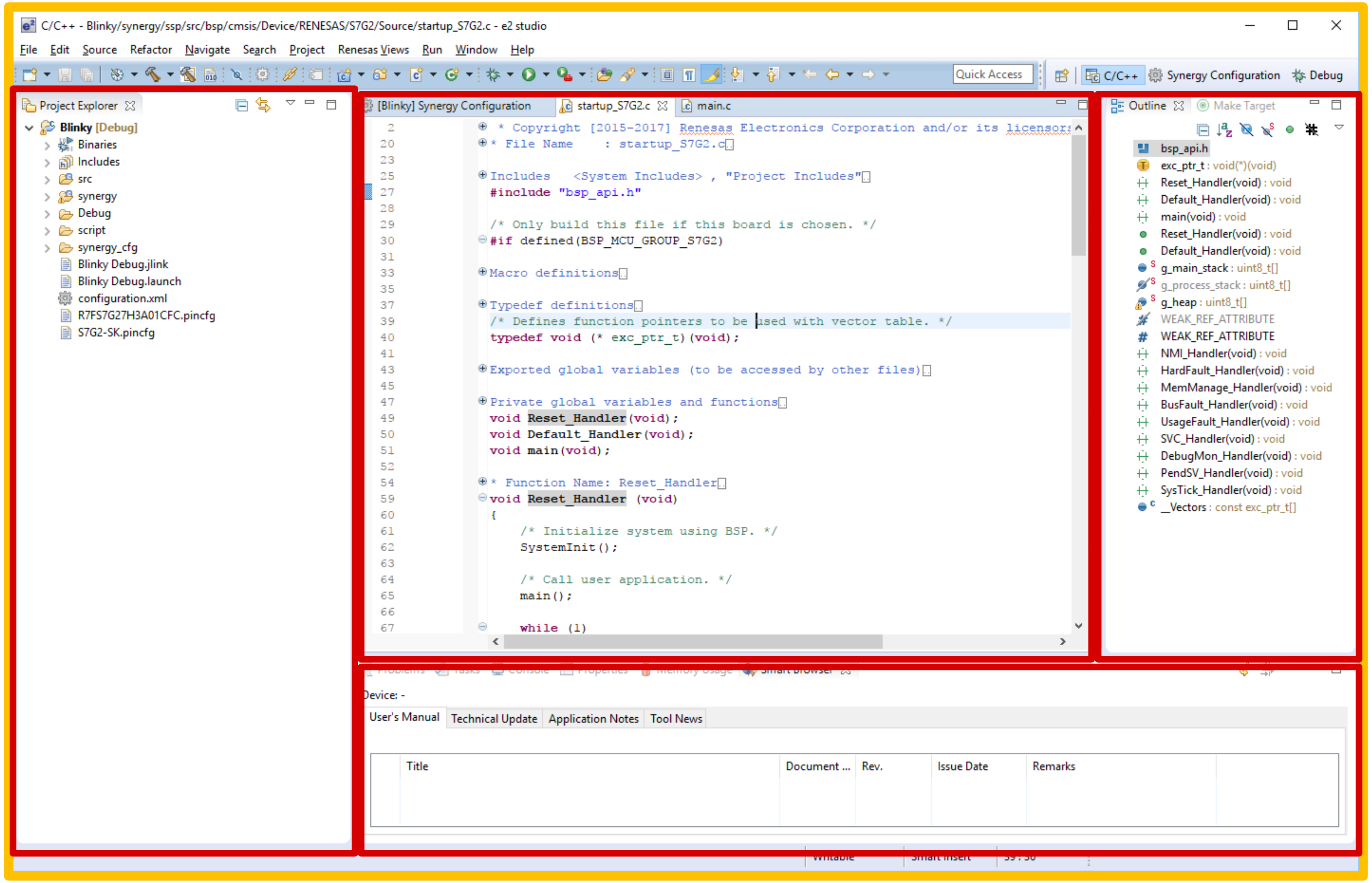

RH850
RL78
RX
RZ
SuperH
synergy
RENESAS
e² studio
v4.0.1.015
eclipse
+
=
ISDE
IDE
Integrated Development Environment
Solution-oriented components
Integrated Solutions Development Environment

LAB1 – ACTIVITY 4

Eclipse is a framework upon which many open source development environments are built.

Some important concepts of eclipse are:

- The **workbench** window (identified by an orange border)
- Several **areas** in the workbench, in this example the areas are identified in red
- In each area, several **parts** can be superimposed, selectable by a **tab**



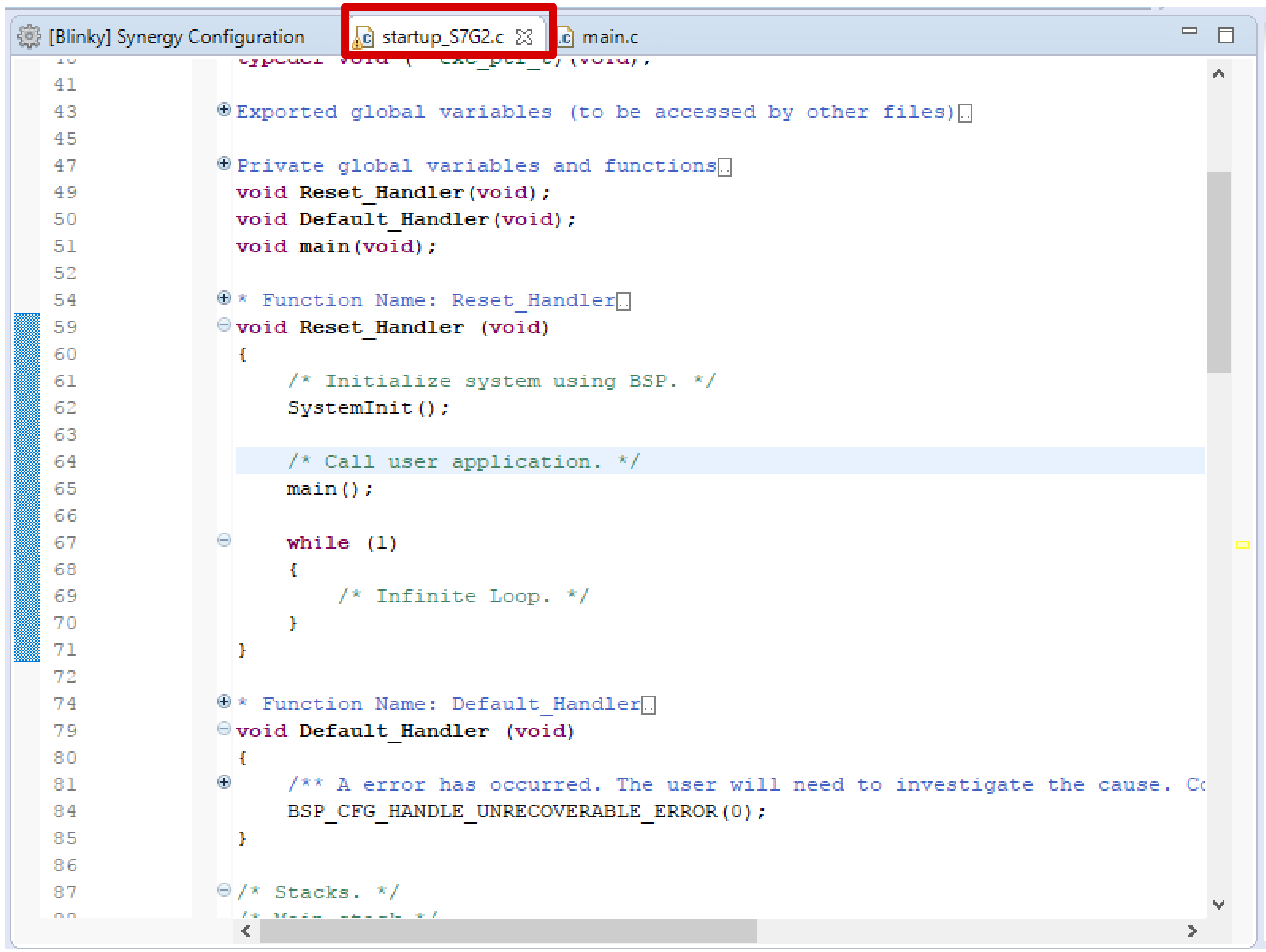
```
 * Copyright [2015-2017] Renesas Electronics Corporation and/or its licensors
* File Name    : startup_S7G2.c

Includes   <System Includes> , "Project Includes"
#include "bsp_api.h"

/* Only build this file if this board is chosen. */
#if defined(BSP_MCU_GROUP_S7G2)

Macro definitions

Typedef definitions
/* Defines function pointers to be used with vector table. */
typedef void (* exc_ptr_t)(void);

Exported global variables (to be accessed by other files)

Private global variables and functions
void Reset_Handler(void);
void Default_Handler(void);
void main(void);

* Function Name: Reset_Handler
void Reset_Handler (void)
{
    /* Initialize system using BSP. */
    SystemInit();

    /* Call user application. */
    main();

    while (1)
```

LAB1 – ACTIVITY 4

A part is either an editor or a view.

An **editor** area (shown here) may have many open files, selectable by a tab (in red). In this example the file *startup_S7G2.c* is shown on the editor panel.

The eclipse editor uses syntax highlighting, hence, comments are in green, types are in magenta, function names in bold, header blocks are in blue, ...



```
Exported global variables (to be accessed by other files)

Private global variables and functions
void Reset_Handler(void);
void Default_Handler(void);
void main(void);

* Function Name: Reset_Handler
void Reset_Handler (void)
{
    /* Initialize system using BSP. */
    SystemInit();

    /* Call user application. */
    main();

    while (1)
    {
        /* Infinite Loop. */
    }
}

* Function Name: Default_Handler
void Default_Handler (void)
{
    /** A error has occurred. The user will need to investigate the cause. Co
    BSP_CFG_HANDLE_UNRECOVERABLE_ERROR(0);
}

/* Stacks. */
```

LAB1 – ACTIVITY 4

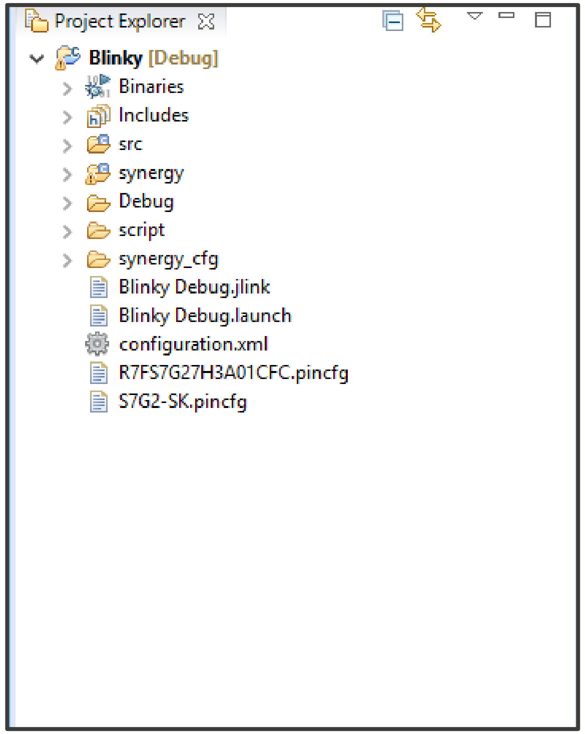
A **view** is used mainly to present information.

The **project explorer** view presents the files that compose the Blinky project.

The **outline** view presents the elements (variable, constants, functions, ...) that compose the file that is open in the currently active editor pane (in this example the startup_S7G2.c file of the previous slide).

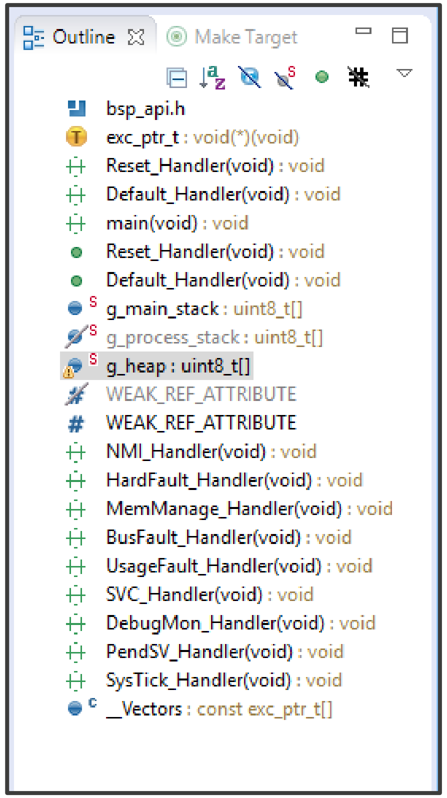
Each view may have its own menu. Accessible by the icon identified in a red box.

There are hundreds of views available in the submenu of Window | ShowView.



Project Explorer

- Blinky [Debug]
  - Binaries
  - Includes
  - src
  - synergy
  - Debug
  - script
  - synergy_cfg
  - Blinky Debug.jlink
  - Blinky Debug.launch
  - configuration.xml
  - R7FS7G27H3A01CFC.pincfg
  - S7G2-SK.pincfg



```
bsp_api.h
exc_ptr_t : void(*)(void)
Reset_Handler(void) : void
Default_Handler(void) : void
main(void) : void
Reset_Handler(void) : void
Default_Handler(void) : void
g_main_stack : uint8_t[]
g_process_stack : uint8_t[]
g_heap : uint8_t[]
WEAK_REF_ATTRIBUTE
WEAK_REF_ATTRIBUTE
NMI_Handler(void) : void
HardFault_Handler(void) : void
MemManage_Handler(void) : void
BusFault_Handler(void) : void
UsageFault_Handler(void) : void
SVC_Handler(void) : void
DebugMon_Handler(void) : void
PendSV_Handler(void) : void
SysTick_Handler(void) : void
__Vectors : const exc_ptr_t[]
```

LAB1 – ACTIVITY 4

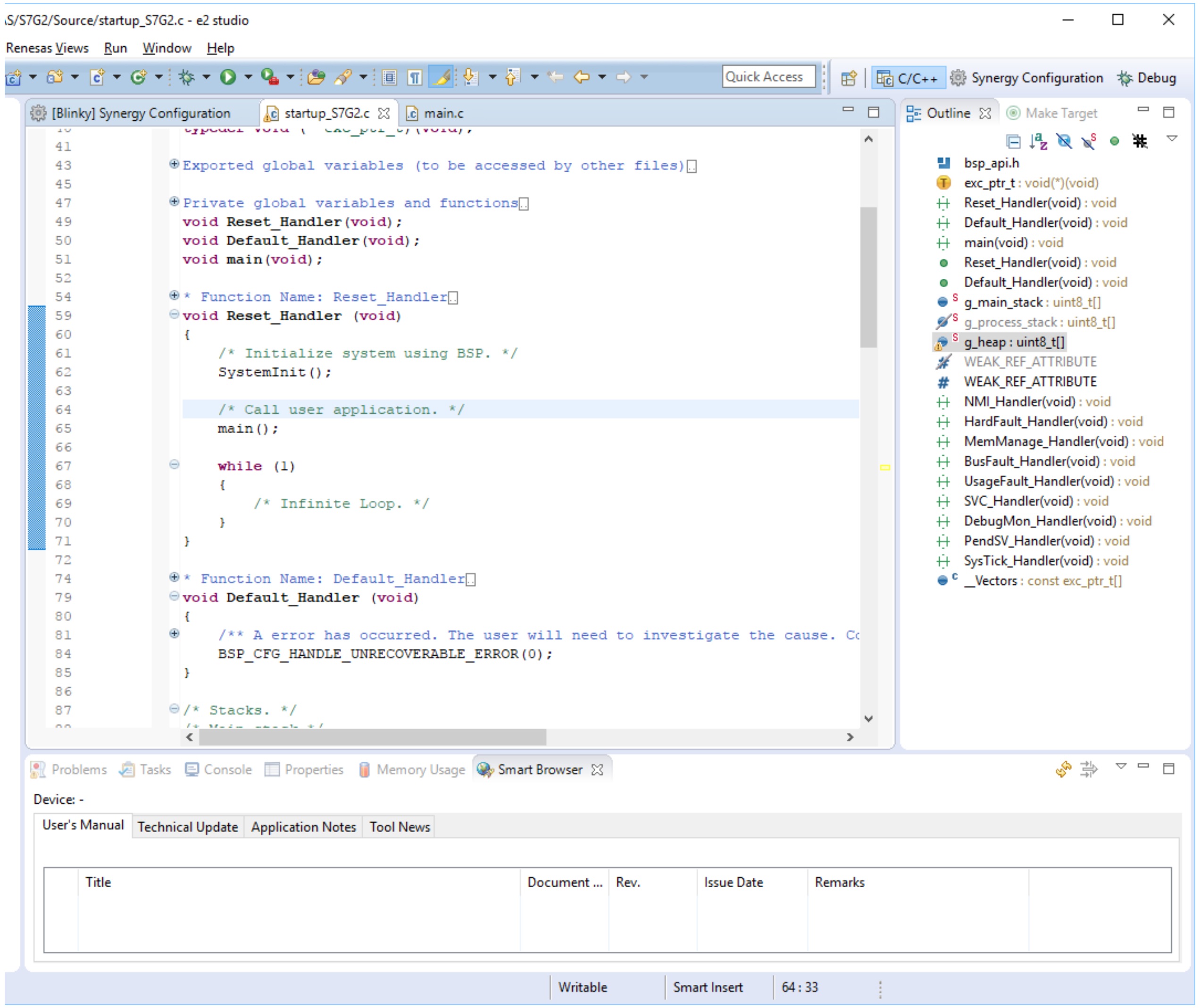
Parts (editors and views) can be freely rearranged and grouped to fit the user needs.

A given organization of parts is called a **perspective**. Some are predefined, but the user can create his own perspectives.

Perspectives are selected by buttons on the upper right (C/C++ perspective identified in red). Each perspective is conceived to fit a given activity, such as source file editing, synergy platform configuration, debugging, ...

Choose Window | Perspective | Reset Perspective...

to restore a perspective back to its default organization.



```
Exported global variables (to be accessed by other files)

Private global variables and functions
void Reset_Handler(void);
void Default_Handler(void);
void main(void);

* Function Name: Reset_Handler
void Reset_Handler (void)
{
    /* Initialize system using BSP. */
    SystemInit();

    /* Call user application. */
    main();

    while (1)
    {
        /* Infinite Loop. */
    }
}

* Function Name: Default_Handler
void Default_Handler (void)
{
    /** A error has occurred. The user will need to investigate the cause. Co
    BSP_CFG_HANDLE_UNRECOVERABLE_ERROR(0);
}

/* Stacks. */
```

- bsp_api.h
- exc_ptr_t : void(*)(void)
- Reset_Handler(void) : void
- Default_Handler(void) : void
- main(void) : void
- Reset_Handler(void) : void
- Default_Handler(void) : void
- g_main_stack : uint8_t[]
- g_process_stack : uint8_t[]
- g_heap : uint8_t[]
- WEAK_REF_ATTRIBUTE
- WEAK_REF_ATTRIBUTE
- NMI_Handler(void) : void
- HardFault_Handler(void) : void
- MemManage_Handler(void) : void
- BusFault_Handler(void) : void
- UsageFault_Handler(void) : void
- SVC_Handler(void) : void
- DebugMon_Handler(void) : void
- PendSV_Handler(void) : void
- SysTick_Handler(void) : void
- __Vectors : const exc_ptr_t[]

Device: -

User's Manual | Technical Update | Application Notes | Tool News

Title	Document ...	Rev.	Issue Date	Remarks

LAB1 – ACTIVITY 4

A **project** consists of a set of inputs files (source files and configuration files) as well as the output files generated during the compile/link process.

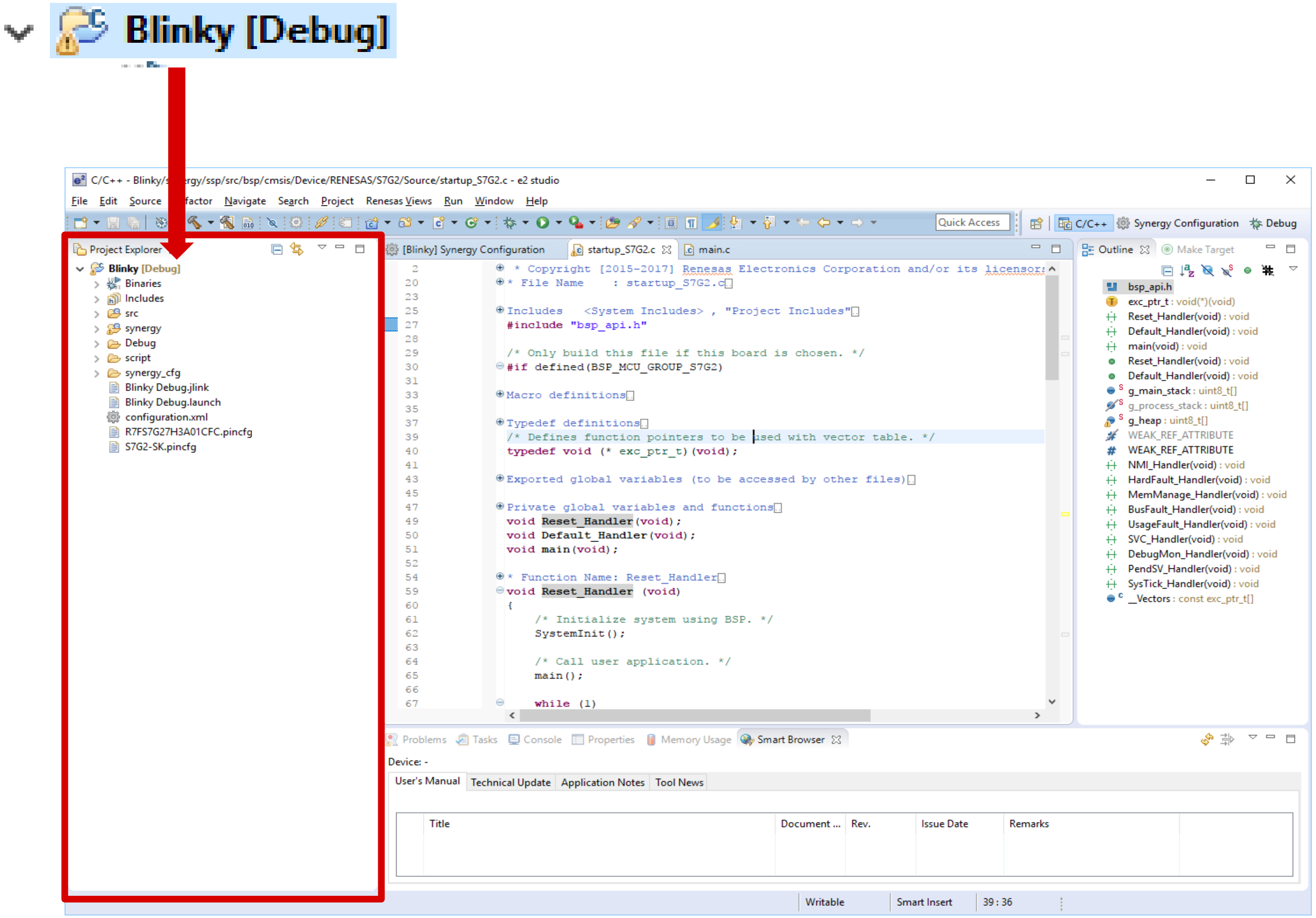
A project is presented in the **Project Explorer** window in the form of a tree with files and folders.

The user can create folders to better organize the files of a project

A project generates a **single binary executable** or a **library**.

Several configurations of a project may be stored for easy access. Such as: a configuration that generates debug info vs one that does not.

Here shown is the **Blinky** project in its Debug configuration.



Blinky [Debug]

C/C++ - Blinky/synergy/ssp/src/bsp/cmsis/Device/RENESAS/S7G2/Source/startup_S7G2.c - e2 studio

File Edit Source Refactor Navigate Search Project Renesas Views Run Window Help
Quick Access
C/C++
Synergy Configuration
Debug
Project Explorer
[Blinky] Synergy Configuration
startup_S7G2.c
main.c
Outline
Make Target

Project Explorer:

- Blinky [Debug]
  - Binaries
  - Includes
  - src
  - synergy
  - Debug
  - script
  - synergy_cfg
  - Blinky Debug.jlink
  - Blinky Debug.launch
  - configuration.xml
  - R7FS7G27H3A01CFC.pincfg
  - S7G2-SK.pincfg

startup_S7G2.c:

```
 * Copyright [2015-2017] Renesas Electronics Corporation and/or its licensors
* File Name    : startup_S7G2.c

Includes   <System Includes> , "Project Includes"
#include "bsp_api.h"

/* Only build this file if this board is chosen. */
#if defined(BSP_MCU_GROUP_S7G2)

Macro definitions

Typedef definitions
/* Defines function pointers to be used with vector table. */
typedef void (* exc_ptr_t)(void);

Exported global variables (to be accessed by other files)

Private global variables and functions
void Reset_Handler(void);
void Default_Handler(void);
void main(void);

* Function Name: Reset_Handler
void Reset_Handler (void)
{
    /* Initialize system using BSP. */
    SystemInit();

    /* Call user application. */
    main();

    while (1)
```

Outline:

- bsp_api.h
- exc_ptr_t : void(*)(void)
- Reset_Handler(void) : void
- Default_Handler(void) : void
- main(void) : void
- Reset_Handler(void) : void
- Default_Handler(void) : void
- g_main_stack : uint8_t[]
- g_process_stack : uint8_t[]
- g_heap : uint8_t[]
- WEAK_REF_ATTRIBUTE
- WEAK_REF_ATTRIBUTE
- NMI_Handler(void) : void
- HardFault_Handler(void) : void
- MemManage_Handler(void) : void
- BusFault_Handler(void) : void
- UsageFault_Handler(void) : void
- SVC_Handler(void) : void
- DebugMon_Handler(void) : void
- PendSV_Handler(void) : void
- SysTick_Handler(void) : void
- __Vectors : const exc_ptr_t[]

Device: -

Title	Document ...	Rev.	Issue Date	Remarks

Writable | Smart Insert | 39 : 36

LAB1 – ACTIVITY 4

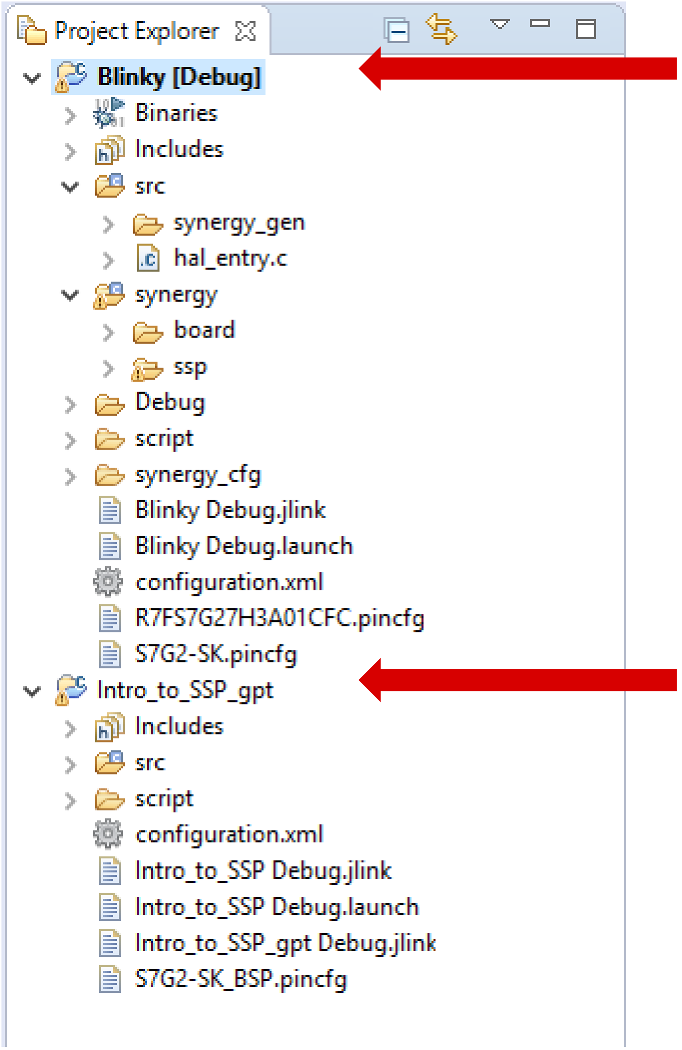
A **workspace** is a set of projects.

Typically, projects in the same workspace share common features such as libraries and hardware platforms.

For instance, you can create a single workspace for all projects of this Embedded Systems course, since they all will be running on the SK-S7G2 board and may share libraries.

On the file system a workspace corresponds to a folder and each of its projects is a folder therein.

(figure shows two projects in the same workspace)

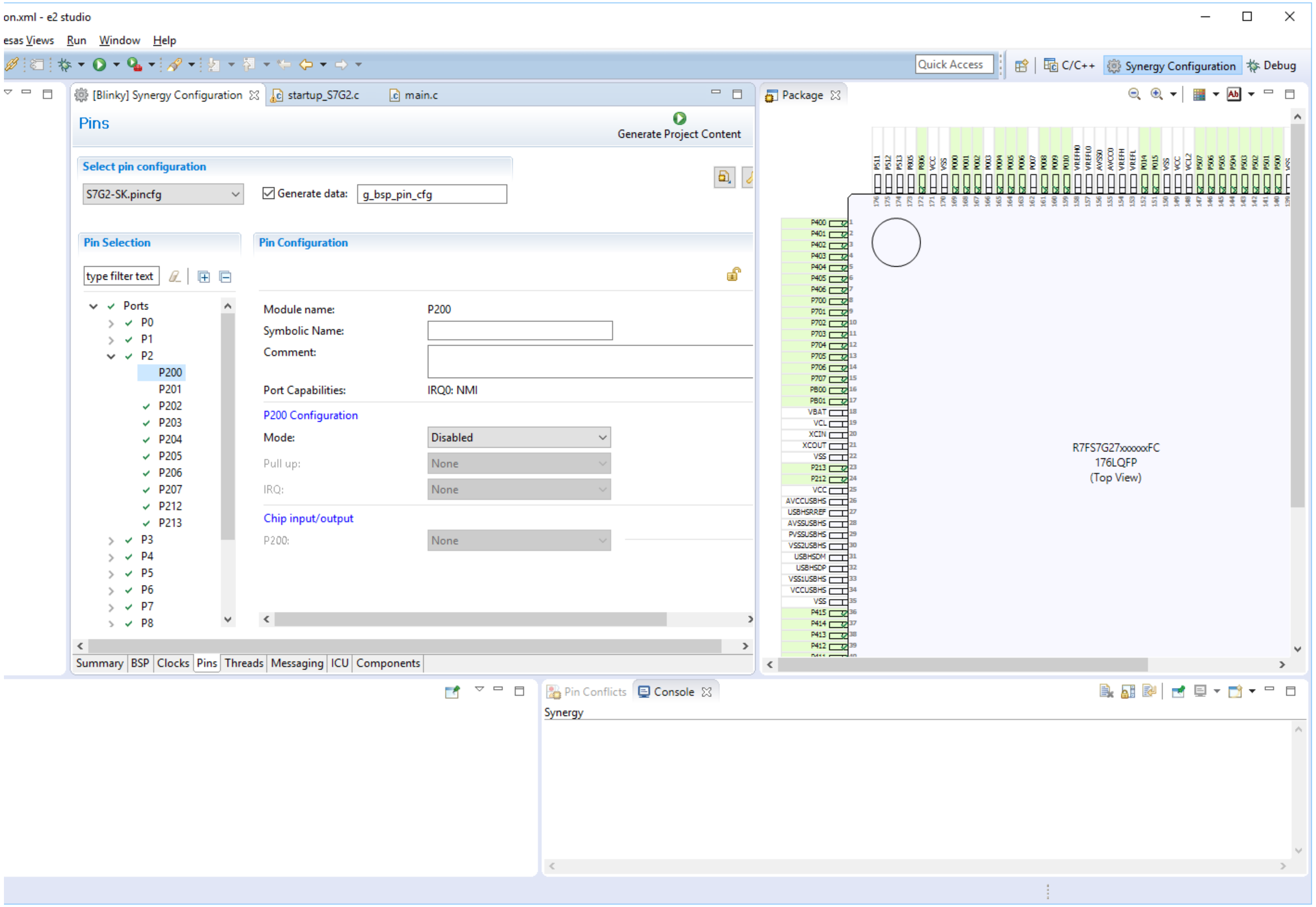


```
Project Explorer
Blinky [Debug]
  Binaries
  Includes
  src
    synergy_gen
    hal_entry.c
  synergy
    board
    ssp
  Debug
  script
  synergy_cfg
  Blinky Debug.jlink
  Blinky Debug.launch
  configuration.xml
  R7FS7G27H3A01CFC.pincfg
  S7G2-SK.pincfg
Intro_to_SSP_gpt
  Includes
  src
  script
  configuration.xml
  Intro_to_SSP Debug.jlink
  Intro_to_SSP Debug.launch
  Intro_to_SSP_gpt Debug.jlink
  S7G2-SK_BSP.pincfg
```

LAB1 – ACTIVITY 4

The **Synergy Configuration** perspective.

Select this perspective when using the **Synergy Configuration Tool** to configure the SSP, define pin functions, create threads, ...



Select pin configuration

S7G2-SK.pincfg ☑ Generate data: g_bsp_pin_cfg

- Ports
  - P0
  - P1
  - P2
    - P200
    - P201
    - P202
    - P203
    - P204
    - P205
    - P206
    - P207
    - P212
    - P213
  - P3
  - P4
  - P5
  - P6
  - P7
  - P8

Module name: P200

Symbolic Name:

Comment:

Port Capabilities: IRQ0: NMI

P200 Configuration

Mode: Disabled

Pull up: None

IRQ: None

Chip input/output

P200: None

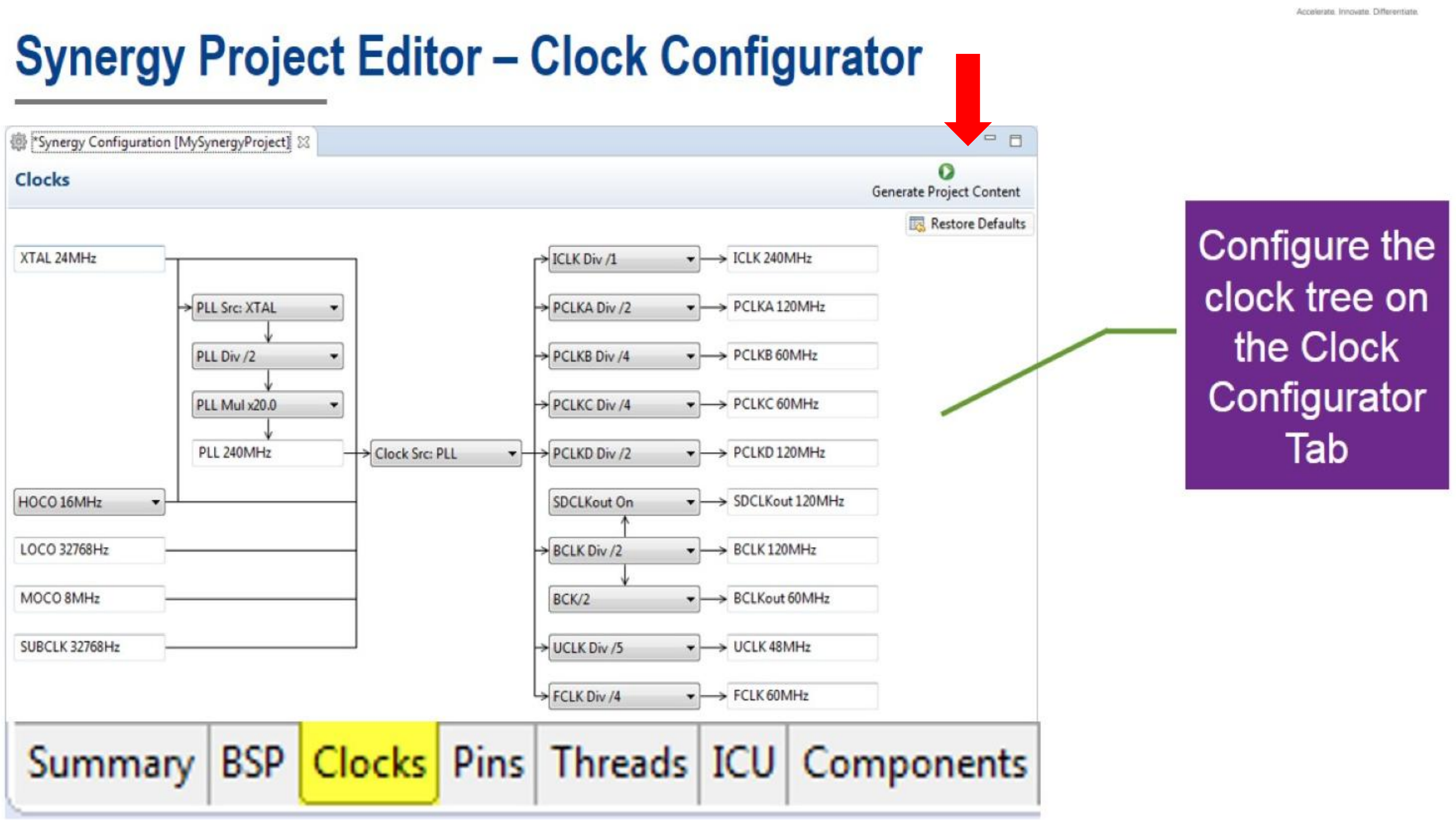
LAB1 – ACTIVITY 4

SSP Configuration Tool

(Synergy Configuration)

The user selects the desired configuration of a module of the SSP (in this example the **clock**) using a graphical tool.

Generate Project Content will produce the corresponding C code.



Accelerate. Innovate. Differentiate.

Synergy Project Editor – Clock Configurator

*Synergy Configuration [MySynergyProject]

Clocks

Generate Project Content

Restore Defaults

XTAL 24MHz
PLL Src: XTAL
PLL Div /2
PLL Mul x20.0
PLL 240MHz
Clock Src: PLL
HOCO 16MHz
LOCO 32768Hz
MOCO 8MHz
SUBCLK 32768Hz

ICLK Div /1 → ICLK 240MHz
PCLKA Div /2 → PCLKA 120MHz
PCLKB Div /4 → PCLKB 60MHz
PCLKC Div /4 → PCLKC 60MHz
PCLKD Div /2 → PCLKD 120MHz
SDCLKout On → SDCLKout 120MHz
BCLK Div /2 → BCLK 120MHz
BCK/2 → BCLKout 60MHz
UCLK Div /5 → UCLK 48MHz
FCLK Div /4 → FCLK 60MHz

Configure the clock tree on the Clock Configurator Tab

source: Renesas Synergy ISDE Tour

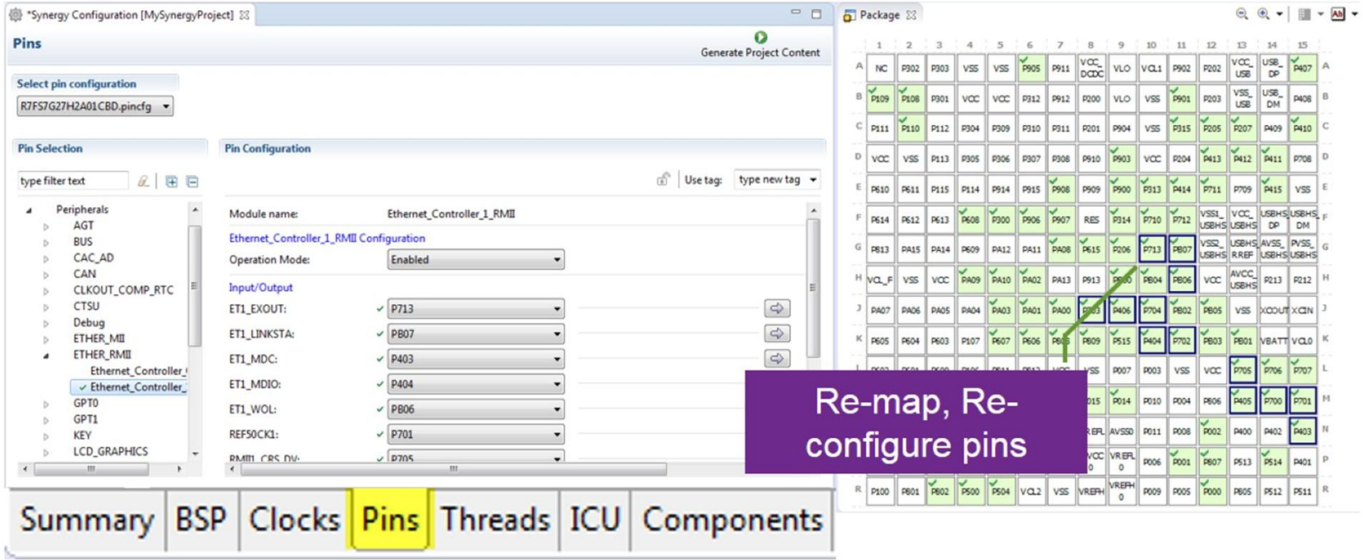
LAB1 – ACTIVITY 4

Another example of the usage of the Synergy Configuration Tool

The graphical tool is used to define the function of each pin.

Configuration errors (e.g. pin conflicts) are identified and can be corrected.

Again, Generate Project Content will generate the corresponding C code.



*Synergy Configuration [MySynergyProject]

Pins

Select pin configuration: R7FS7G27H2A01CBD.pincfg

Module name: Ethernet_Controller_1_RMII

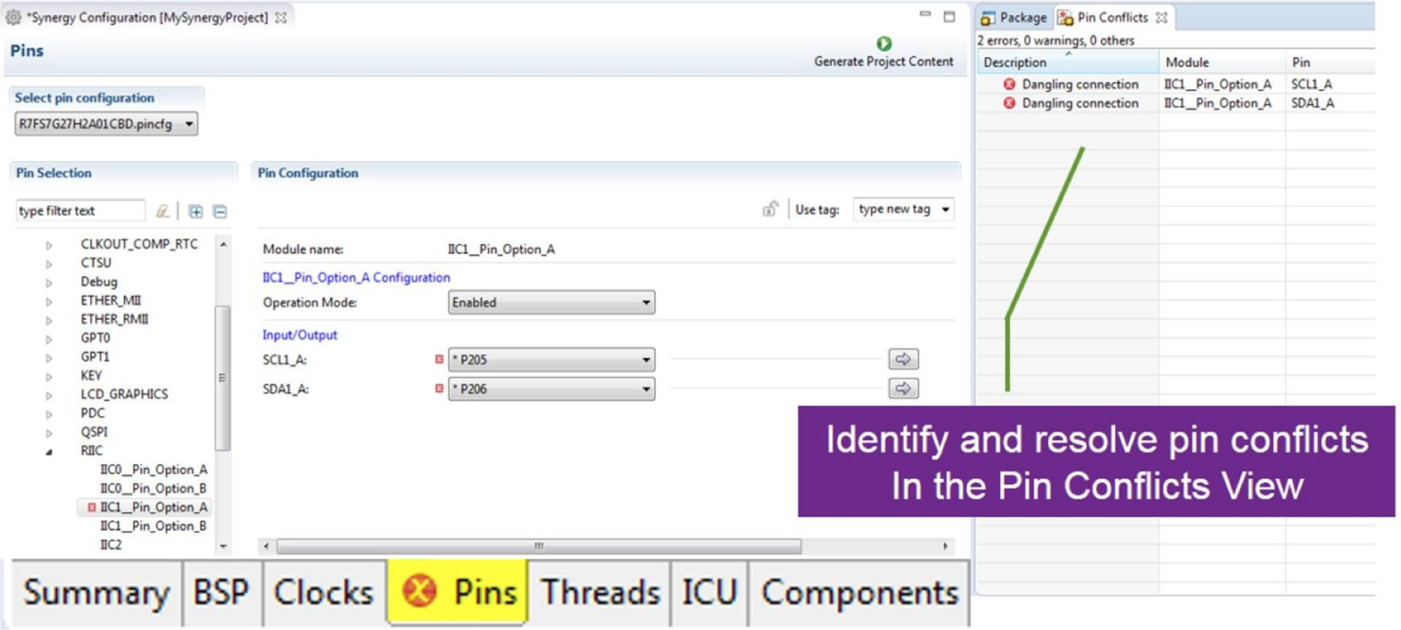
Ethernet_Controller_1_RMII Configuration

Operation Mode: Enabled

Input/Output

ET1_EXOUT:	✓	P713
ET1_LINKSTA:	✓	PB07
ET1_MDC:	✓	P403
ET1_MDIO:	✓	P404
ET1_WOL:	✓	PB06
REF50CK1:	✓	P701
RMII1_CRS_DV:	✓	P705

Re-map, Re-configure pins



*Synergy Configuration [MySynergyProject]

Pins

Select pin configuration: R7FS7G27H2A01CBD.pincfg

Module name: IIC1__Pin_Option_A

IIC1__Pin_Option_A Configuration

Operation Mode: Enabled

Input/Output

SCL1_A:	☒	* P205
SDA1_A:	☒	* P206

Pin Conflicts

2 errors, 0 warnings, 0 others

Description	Module	Pin
☒ Dangling connection	IIC1__Pin_Option_A	SCL1_A
☒ Dangling connection	IIC1__Pin_Option_A	SDA1_A

Identify and resolve pin conflicts
In the Pin Conflicts View

source: Renesas Synergy ISDE Tour

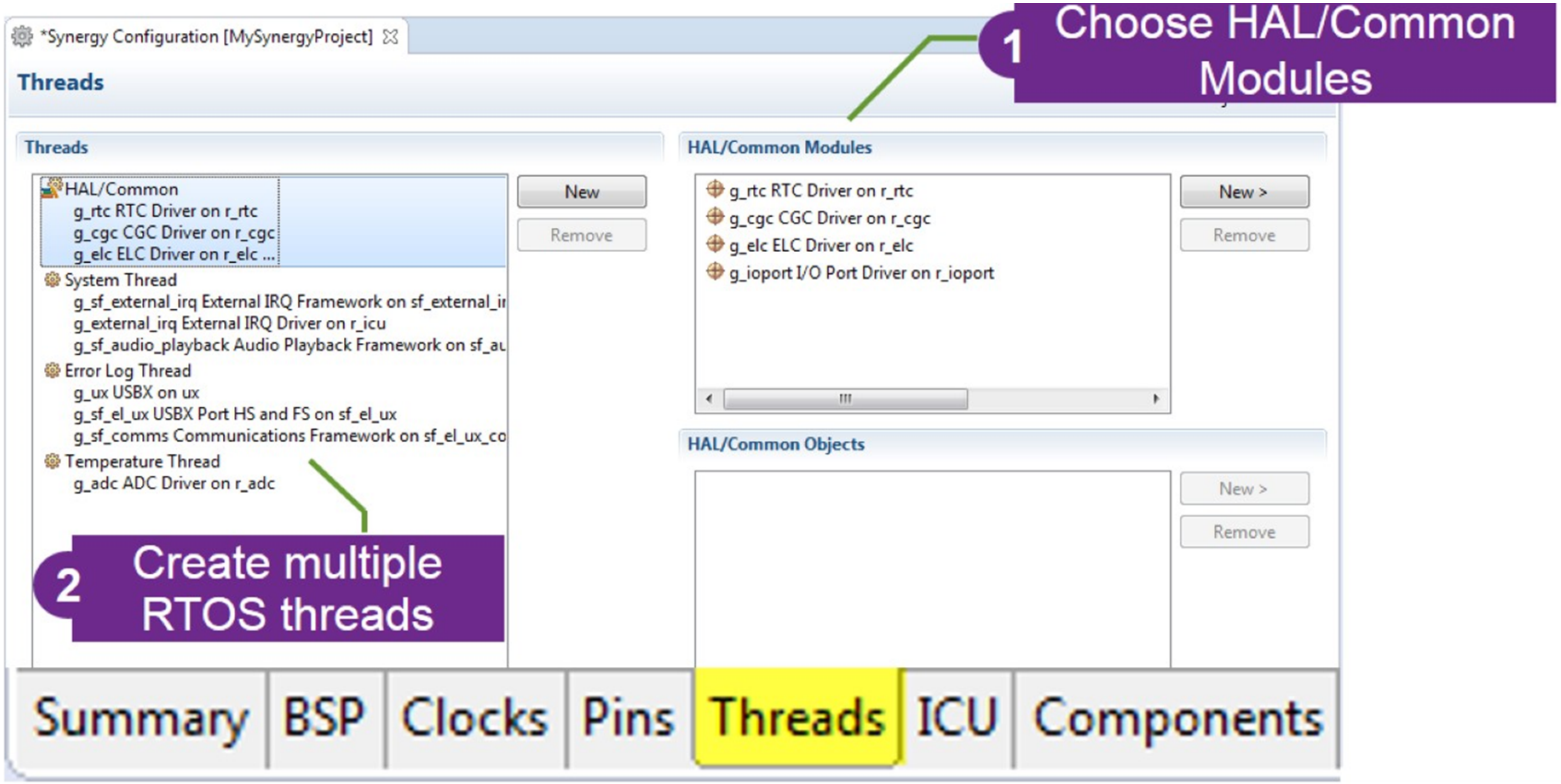
LAB1 – ACTIVITY 4

Use the Synergy Configuration tool to chose HAL modules, to create threads and configure them.

HAL =

Hardware Abstraction Layer

see Embedded Systems Architecture starting on slide [arch]



*Synergy Configuration [MySynergyProject]

Threads

Threads

- HAL/Common
  - g_rtc RTC Driver on r_rtc
  - g_cgc CGC Driver on r_cgc
  - g_elc ELC Driver on r_elc ...
- System Thread
  - g_sf_external_irq External IRQ Framework on sf_external_ir
  - g_external_irq External IRQ Driver on r_icu
  - g_sf_audio_playback Audio Playback Framework on sf_au
- Error Log Thread
  - g_ux USBX on ux
  - g_sf_el_ux USBX Port HS and FS on sf_el_ux
  - g_sf_comms Communications Framework on sf_el_ux_co
- Temperature Thread
  - g_adc ADC Driver on r_adc

HAL/Common Modules

- g_rtc RTC Driver on r_rtc
- g_cgc CGC Driver on r_cgc
- g_elc ELC Driver on r_elc
- g_ioport I/O Port Driver on r_ioport

HAL/Common Objects

1 Choose HAL/Common Modules

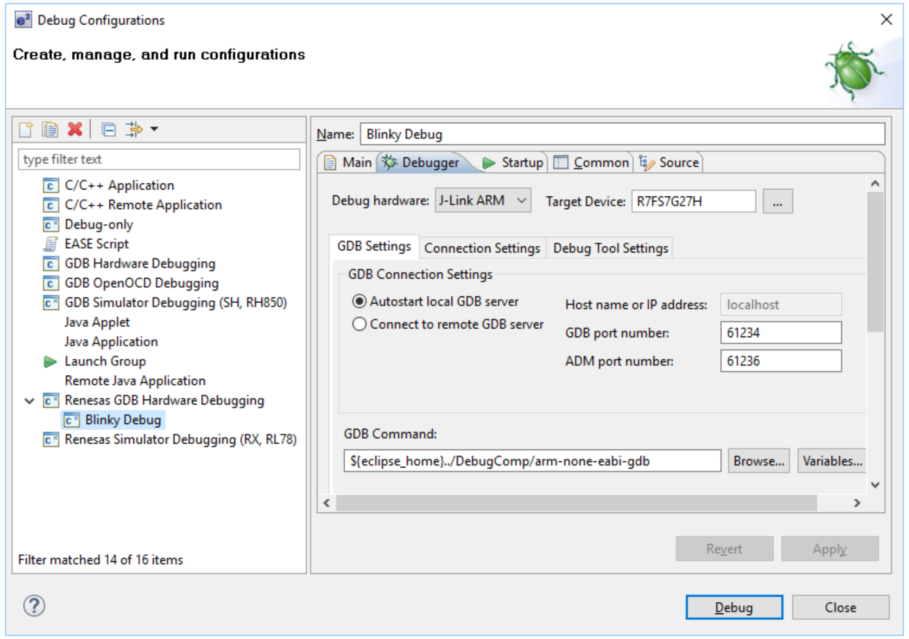
2 Create multiple RTOS threads

source: Renesas Synergy ISDE Tour

LAB1 – ACTIVITY 4

To start a debug session use the debug icon.

A debug configuration window is used to setup the debug configuration



Filter matched 14 of 16 items

Name: Blinky Debug

Debug hardware: J-Link ARM

Target Device: R7FS7G27H

● Autostart local GDB server

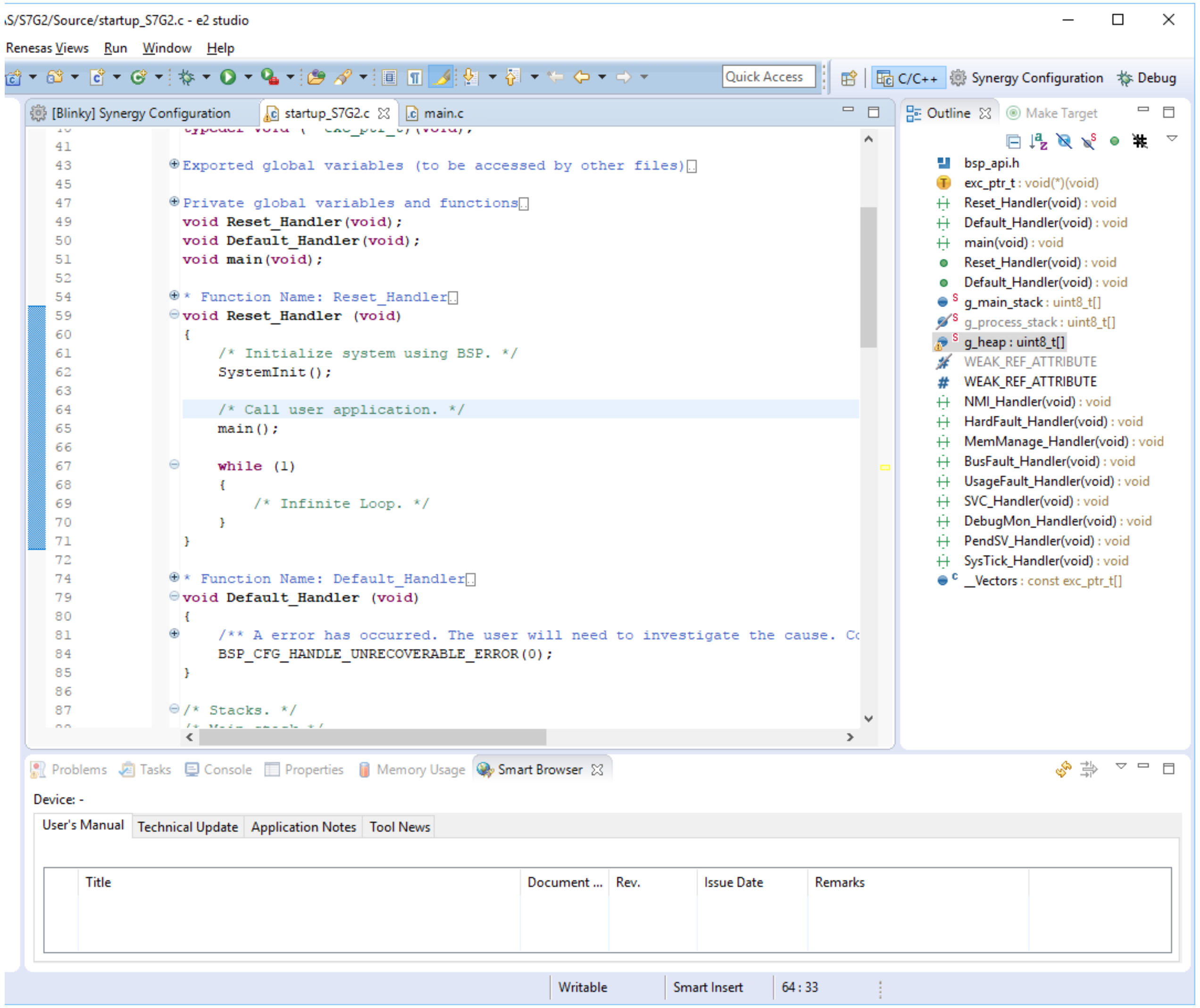
○ Connect to remote GDB server

Host name or IP address: localhost

GDB port number: 61234

ADM port number: 61236

GDB Command: ${eclipse_home}../DebugComp/arm-none-eabi-gdb


S/S7G2/Source/startup_S7G2.c - e2 studio
Renesas Views
Run
Window
Help
Quick Access
C/C++
Synergy Configuration
Debug
[Blinky] Synergy Configuration
startup_S7G2.c
main.c
Outline
Make Target
bsp_api.h
exc_ptr_t : void(*)(void)
Reset_Handler(void) : void
Default_Handler(void) : void
main(void) : void
Reset_Handler(void) : void
Default_Handler(void) : void
g_main_stack : uint8_t[]
g_process_stack : uint8_t[]
g_heap : uint8_t[]
WEAK_REF_ATTRIBUTE
WEAK_REF_ATTRIBUTE
NMI_Handler(void) : void
HardFault_Handler(void) : void
MemManage_Handler(void) : void
BusFault_Handler(void) : void
UsageFault_Handler(void) : void
SVC_Handler(void) : void
DebugMon_Handler(void) : void
PendSV_Handler(void) : void
SysTick_Handler(void) : void
__Vectors : const exc_ptr_t[]
Problems
Tasks
Console
Properties
Memory Usage
Smart Browser
Device: -
User's Manual
Technical Update
Application Notes
Tool News
Title
Document ...
Rev.
Issue Date
Remarks
Writable
Smart Insert
64 : 33

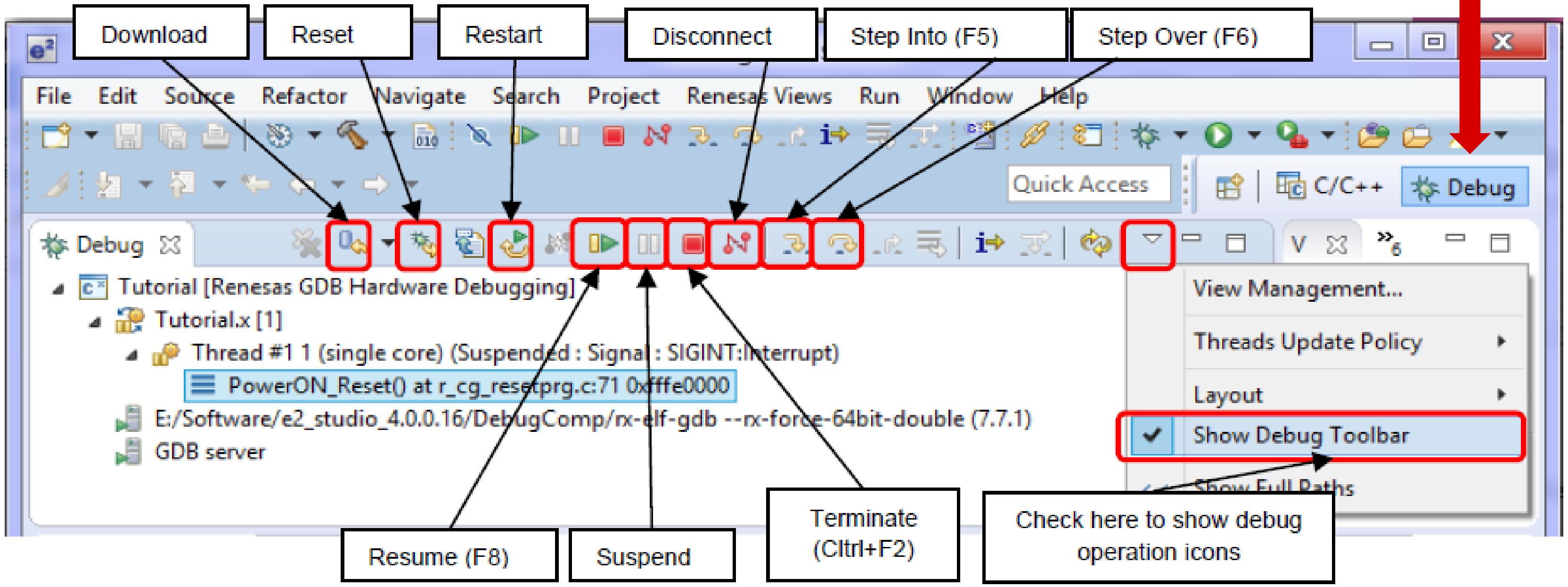
```
Exported global variables (to be accessed by other files)
Private global variables and functions
void Reset_Handler(void);
void Default_Handler(void);
void main(void);

* Function Name: Reset_Handler
void Reset_Handler (void)
{
    /* Initialize system using BSP. */
    SystemInit();

    /* Call user application. */
    main();

    while (1)
    {
        /* Infinite Loop. */
    }
}

* Function Name: Default_Handler
void Default_Handler (void)
{
    /** A error has occurred. The user will need to investigate the cause. Cc
    BSP_CFG_HANDLE_UNRECOVERABLE_ERROR(0);
}

/* Stacks. */
```

LAB1 – ACTIVITY 4

When entering a debug session,
the current perspective changes to Debug.

Several debug buttons become available.

debug perspective



Download
Reset
Restart
Disconnect
Step Into (F5)
Step Over (F6)
File Edit Source Refactor Navigate Search Project Renesas Views Run Window Help
Quick Access
C/C++
Debug
Tutorial [Renesas GDB Hardware Debugging]
Tutorial.x [1]
Thread #1 1 (single core) (Suspended : Signal : SIGINT:Interrupt)
PowerON_Reset() at r_cg_resetprg.c:71 0xfffe0000
E:/Software/e2_studio_4.0.0.16/DebugComp/rx-elf-gdb --rx-force-64bit-double (7.7.1)
GDB server
View Management...
Threads Update Policy
Layout
Show Debug Toolbar
Show Full Paths
Resume (F8)
Suspend
Terminate (Cltrl+F2)
Check here to show debug operation icons

source (also for some other figures in this section):
e² studio Integrated Development Environment User's Manual: Getting Started Guide
r20ut2771ej0400_e2_start_s.pdf
https://www.renesas.com/us/en/doc/products/tool/doc/006/r20ut2771ej0400_e2_start_s.pdf

LAB1 – SYNERGY PLATFORM

Activity 5 – Overview of the SSP

1. Synergy Software Package
2. How to use the SSP

LAB1 – ACTIVITY 5 – STEP 1

The Synergy Software Package (SSP) internal structure is presented inside the red box; the grey

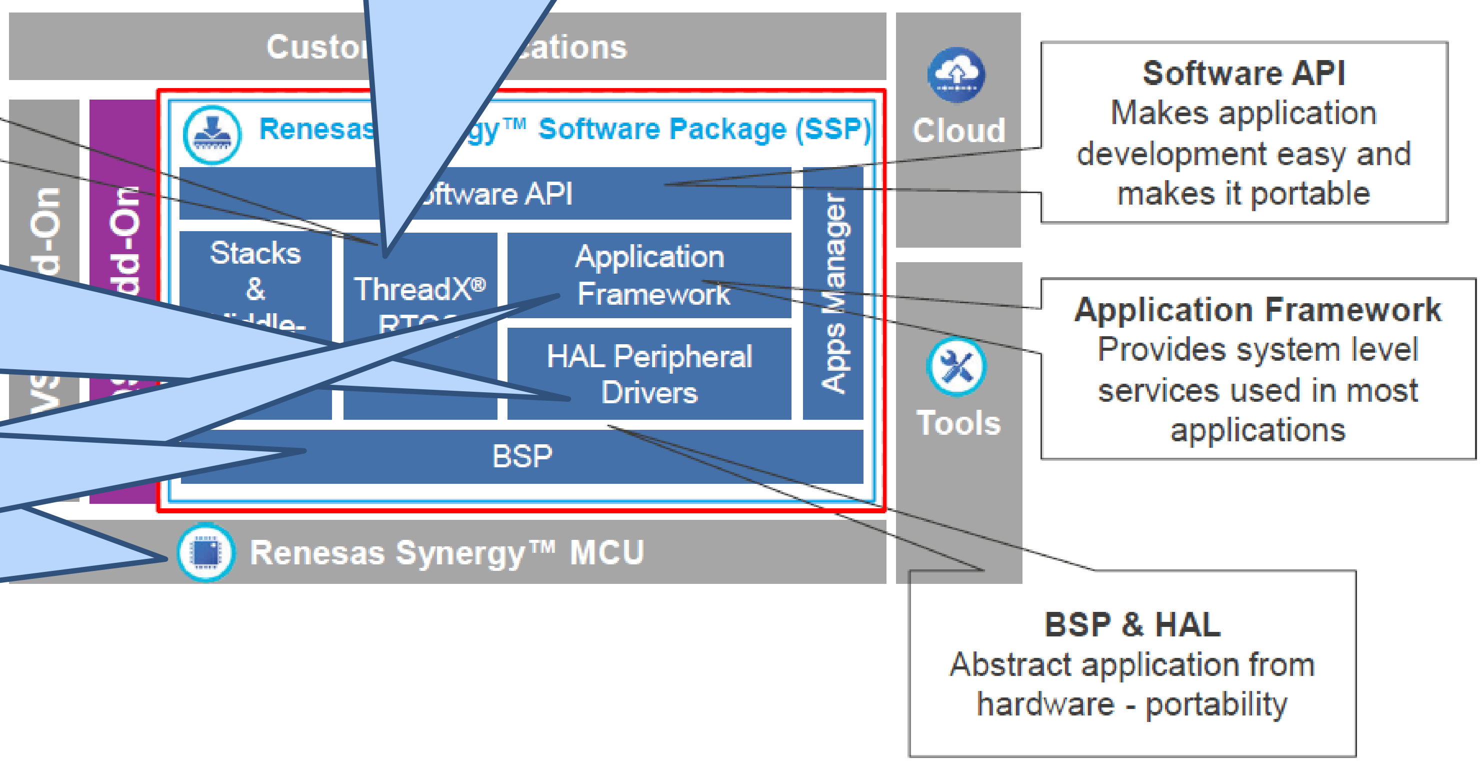
The ThreadX RTOS, and the aggregated FileX, USBX, GUIX, NetX are part of the SSP providing multitasking, communication, file system and user interface.

The Hardware Abstraction Layer provides an abstraction of the internal peripherals of the MCU: timer, communication interfaces, gpio, ADC, DAC, ...

BSP - Board Support Package is the part of SSP that provides an abstraction to the functionality implemented on a board, outside the MCU. Such as: leds, push-buttons, ...

to higher level functionality, such as: touch screen, audio, block media, ...

training.



Software API
Makes application development easy and makes it portable

Application Framework
Provides system level services used in most applications

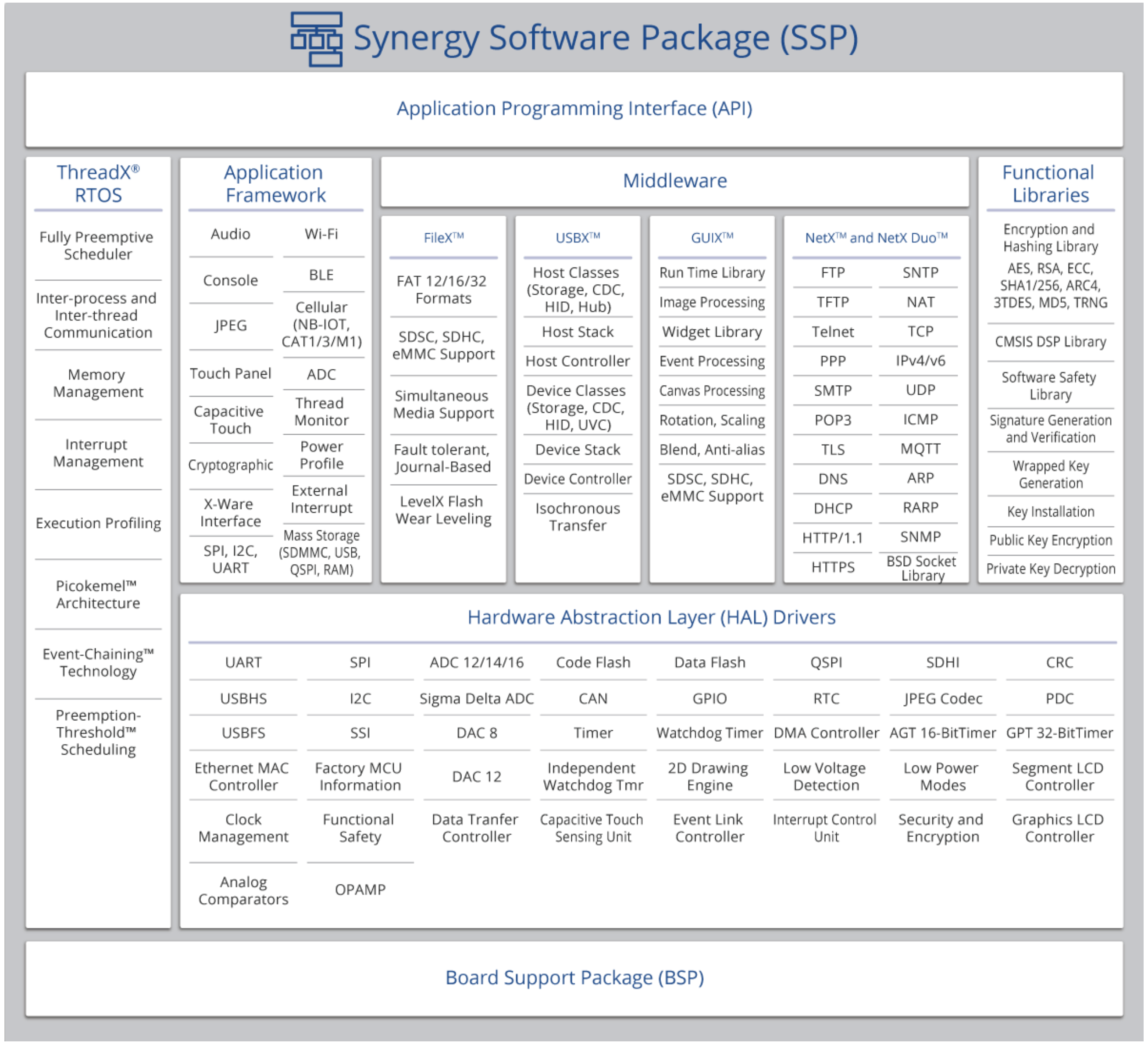
BSP & HAL
Abstract application from hardware - portability

source: Renesas

LAB1 – ACTIVITY 5 – STEP 1

The Synergy Software Package (SSP)

internal structure is further detailed here:



Synergy Software Package (SSP)

Application Programming Interface (API)

ThreadX® RTOS

- Fully Preemptive Scheduler
- Inter-process and Inter-thread Communication
- Memory Management
- Interrupt Management
- Execution Profiling
- Picokemel™ Architecture
- Event-Chaining™ Technology
- Preemption-Threshold™ Scheduling

Application Framework

- Audio
- Console
- JPEG
- Touch Panel
- Capacitive Touch
- Cryptographic
- X-Ware Interface
- SPI, I2C, UART
- Wi-Fi
- BLE
- Cellular (NB-IOT, CAT1/3/M1)
- ADC
- Thread Monitor
- Power Profile
- External Interrupt
- Mass Storage (SDMMC, USB, QSPI, RAM)

Middleware

FileX™

- FAT 12/16/32 Formats
- SDSC, SDHC, eMMC Support
- Simultaneous Media Support
- Fault tolerant, Journal-Based
- LevelX Flash Wear Leveling

USBX™

- Host Classes (Storage, CDC, HID, Hub)
- Host Stack
- Host Controller
- Device Classes (Storage, CDC, HID, UVC)
- Device Stack
- Device Controller
- Isochronous Transfer

GUIX™

- Run Time Library
- Image Processing
- Widget Library
- Event Processing
- Canvas Processing
- Rotation, Scaling
- Blend, Anti-alias
- SDSC, SDHC, eMMC Support

NetX™ and NetX Duo™

FTP	SNTP
TFTP	NAT
Telnet	TCP
PPP	IPv4/v6
SMTP	UDP
POP3	ICMP
TLS	MQTT
DNS	ARP
DHCP	RARP
HTTP/1.1	SNMP
HTTPS	BSD Socket Library

Functional Libraries

- Encryption and Hashing Library AES, RSA, ECC, SHA1/256, ARC4, 3TDES, MD5, TRNG
- CMSIS DSP Library
- Software Safety Library
- Signature Generation and Verification
- Wrapped Key Generation
- Key Installation
- Public Key Encryption
- Private Key Decryption

Hardware Abstraction Layer (HAL) Drivers

UART	SPI	ADC 12/14/16	Code Flash	Data Flash	QSPI	SDHI	CRC
USBHS	I2C	Sigma Delta ADC	CAN	GPIO	RTC	JPEG Codec	PDC
USBFS	SSI	DAC 8	Timer	Watchdog Timer	DMA Controller	AGT 16-BitTimer	GPT 32-BitTimer
Ethernet MAC Controller	Factory MCU Information	DAC 12	Independent Watchdog Tmr	2D Drawing Engine	Low Voltage Detection	Low Power Modes	Segment LCD Controller
Clock Management	Functional Safety	Data Tranfer Controller	Capacitive Touch Sensing Unit	Event Link Controller	Interrupt Control Unit	Security and Encryption	Graphics LCD Controller
Analog Comparators	OPAMP						

Board Support Package (BSP)

source: Renesas SSP v 1.7.5 documentation

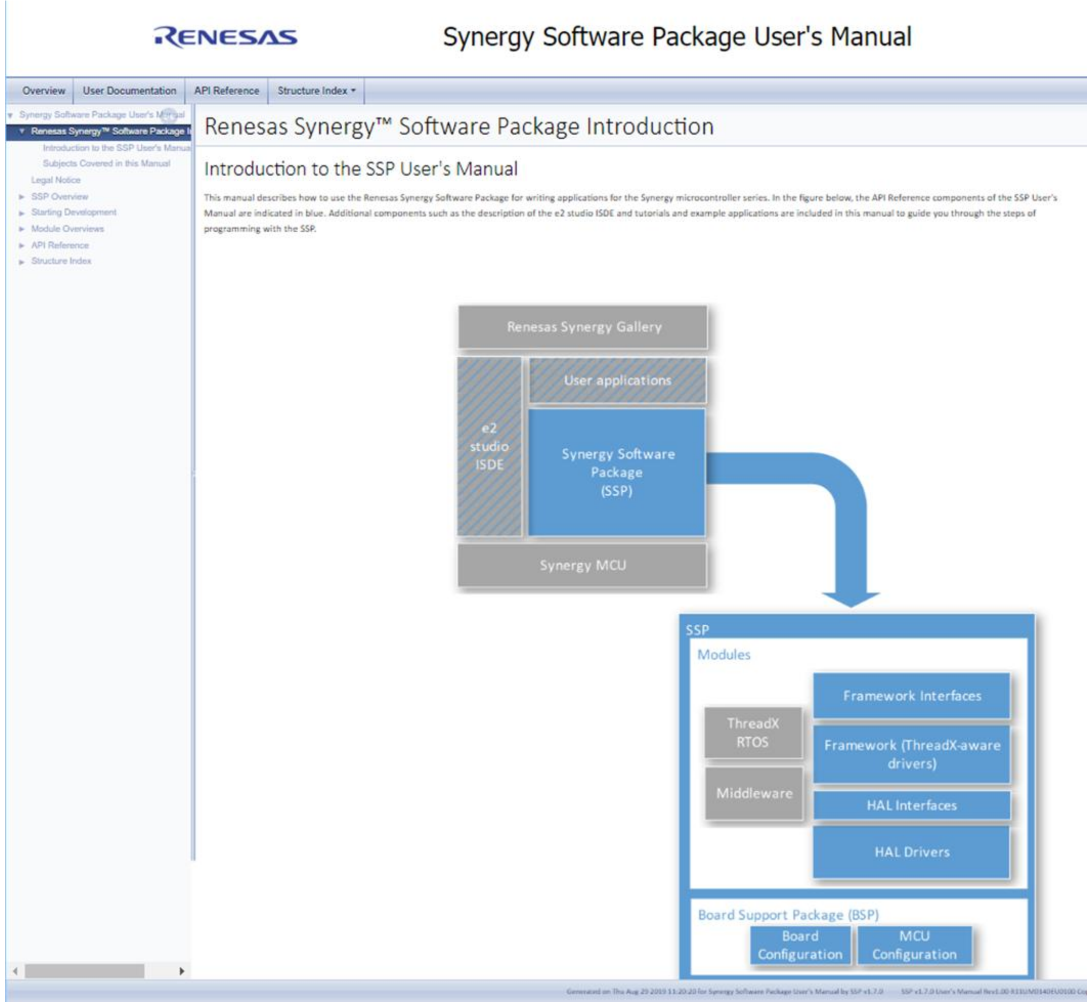
LAB1 – ACTIVITY 5 – STEP 1

The Synergy Software Package (SSP) is downloadable from the Renesas website (see Lab 1, Activity 1, Step 5).

It is installed in *C:\Renesas\Synergy\e2studio_v7.5.1_ssp_v1.7.5.*
The ISDE e2studio will access the SSP directly from there.

The HTML documentation for SSP is installed in
C:\Renesas\Synergy\e2studio_v7.5.1_ssp_v1.7.5\SSP_Documentation.
By opening in a browser the file
ssp-user-manual-html-v1.00-sspv1.7.5.html
the doxygen generated documentation is presented.

SSP documentation is also available in pdf from the Renesas website



Synergy Software Package User's Manual

Renesas Synergy™ Software Package Introduction

Introduction to the SSP User's Manual

This manual describes how to use the Renesas Synergy Software Package for writing applications for the Synergy microcontroller series. In the figure below, the API Reference components of the SSP User's Manual are indicated in blue. Additional components such as the description of the e2 studio ISDE and tutorials and example applications are included in this manual to guide you through the steps of programming with the SSP.

source: Renesas Synergy ISDE Tour

LAB1 – ACTIVITY 5 – STEP 2

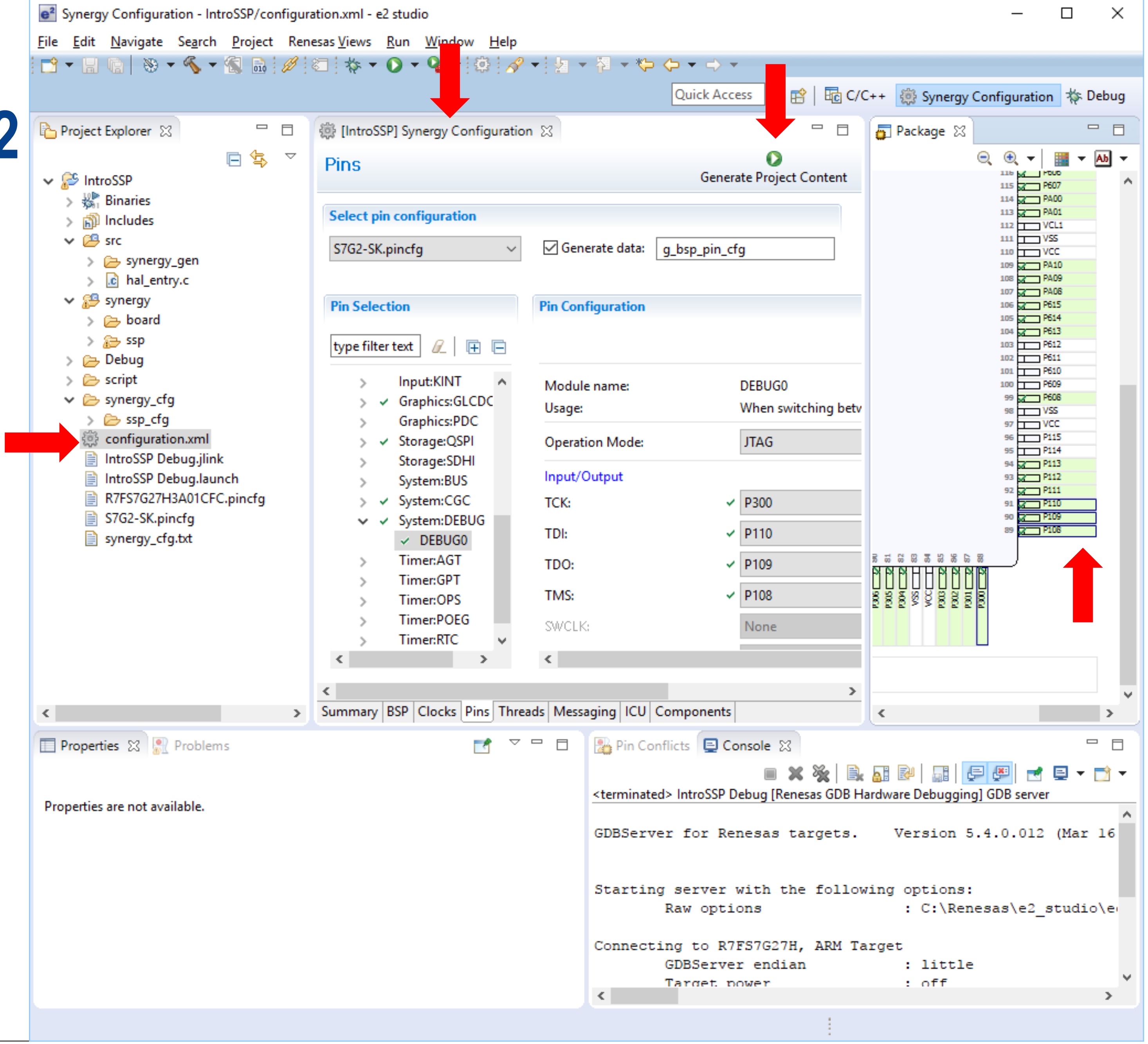
Using the Synergy Software Package (SSP)

The SSP is made available as a large set of source files that implement its functionality.

The SSP is also highly configurable to the application needs.

The task of configuring the SSP is significantly simplified by the use of the **Synergy Configuration Tool** available in e2studio.

This tool provides a graphical interface for the programmer to select the proper configuration. Once selected, the button Generate Project Contents will generate the appropriate C files that match the selected configuration.



Select pin configuration

S7G2-SK.pincfg ☑ Generate data: g_bsp_pin_cfg

Module name: DEBUG0

Usage: When switching betv

Operation Mode: JTAG

Input/Output

TCK: ✓ P300

TDI: ✓ P110

TDO: ✓ P109

TMS: ✓ P108

SWCLK: None

Properties are not available.

```
<terminated> IntroSSP Debug [Renesas GDB Hardware Debugging] GDB server

GDBServer for Renesas targets.    Version 5.4.0.012 (Mar 16

Starting server with the following options:
        Raw options                    : C:\Renesas\e2_studio\e

Connecting to R7FS7G27H, ARM Target
        GDBServer endian               : little
        Target power                   : off
```

LAB2 – SAMPLE C PROGRAM

Objectives:

- In Lab 2, the student will develop a simple game. The game's objective is simply to respond as fast as possible to a visual stimulus. A led goes on after a random time and the player has to press a button. The current response time is presented.
- The main objective of this Lab is to guide the student through a proposed development process that should be used in the following labs.

LAB2 – SAMPLE C PROGRAM

Learning Objectives:

- Project planning
- Problem definition, specification, hardware platform study, software framework study, design
- Project generation
- SSP Configuration
- Threads configuration
- Source code editing
- Compile/Link
- Debug

LAB2 – SAMPLE C PROGRAM

Activities:

1. Plan the phases of the development process
2. Problem definition
3. Specification
4. Hardware platform study
5. Software framework study
6. Design
7. Use the Wizard to generate a Synergy C Project
8. Configure the SSP
9. Use the Editor
10. Compile and link the project
11. Debug on the SK-S7G2 board

further reading for this Lab:
e^2 studio Integrated Development Environment User's Manual: Getting Started Guide
r20ut2771ej0400_e2_start_s.pdf
https://www.renesas.com/us/en/doc/products/tool/doc/006/r20ut2771ej0400_e2_start_s.pdf
(it is also the source of some figures of this section)

LAB2 – ACTIVITY 1

Activity 1 – Plan the phases of the development process

In Lab2, the student is faced with a simple problem to be implemented in C, using many of the functionalities made available in e2studio. The objectives in this lab are not limited to understanding the functionality of the tool, but also to follow a planned approach to a software development activity.

1. Problem definition: the problem statement should clearly define the scope of the problem.
2. Specification: a clear and precise set of statements that define the functionality of the resulting software as well as its non-functional characteristics (performance, ...)
3. Study of the hardware platform: one must have a clear understanding of the features of the MCU and of the board that will be used in this development
4. Software framework study: some of the functionality needed for the solution is available in the form of software components in the SSP, these must be identified and understood
5. Design: identification of the SW components that need to be developed, their interfaces, algorithms and interfaces to other software components.
6. Tool usage: project generation, SSP configuration, source code editing, compilation, linkage, debug

LAB2 – ACTIVITY 2

Activity 2 – Problem Definition

Game objective: to respond as fast as possible to a visual stimulus

Game operation: once the game is turned on, after 1 second an LED is turned on, the player must respond by pressing a button.

Game cycle repeats indefinitely.

Game presents the player response time.

Basic solution: response times are saved in variables and can be visualized in the debugger

LAB2 – ACTIVITY 3

Activity 3 – Specification

S1 - The ResponseTimeGame console shall provide an LED, a push-button and an LCD.

S2 - The operation of the ResponseTimeGame, after power up, is:

1. wait for 1 second
2. turn on LED
3. start measuring time
4. wait for player to press the push-button
5. stop measuring time, save measure as current response time
6. turn off LED
7. a variable should hold the current response time
8. go to step 1

S3 - discard response time measurements above 3 seconds

LAB2 – ACTIVITY 4

Activity 4 – Study the Hardware platform (SK-S7G2 board and S7G2 MCU)

The first manual to be studied is the Starter Kit SK-S7G2 User’s Manual.pdf

Relevant information in this manual concerning the current project is:

- Block diagram
- LEDs
- push-button
- LCD

LAB2 – ACTIVITY 4

Relevant blocks of the SK-S7G2 board are marked in red

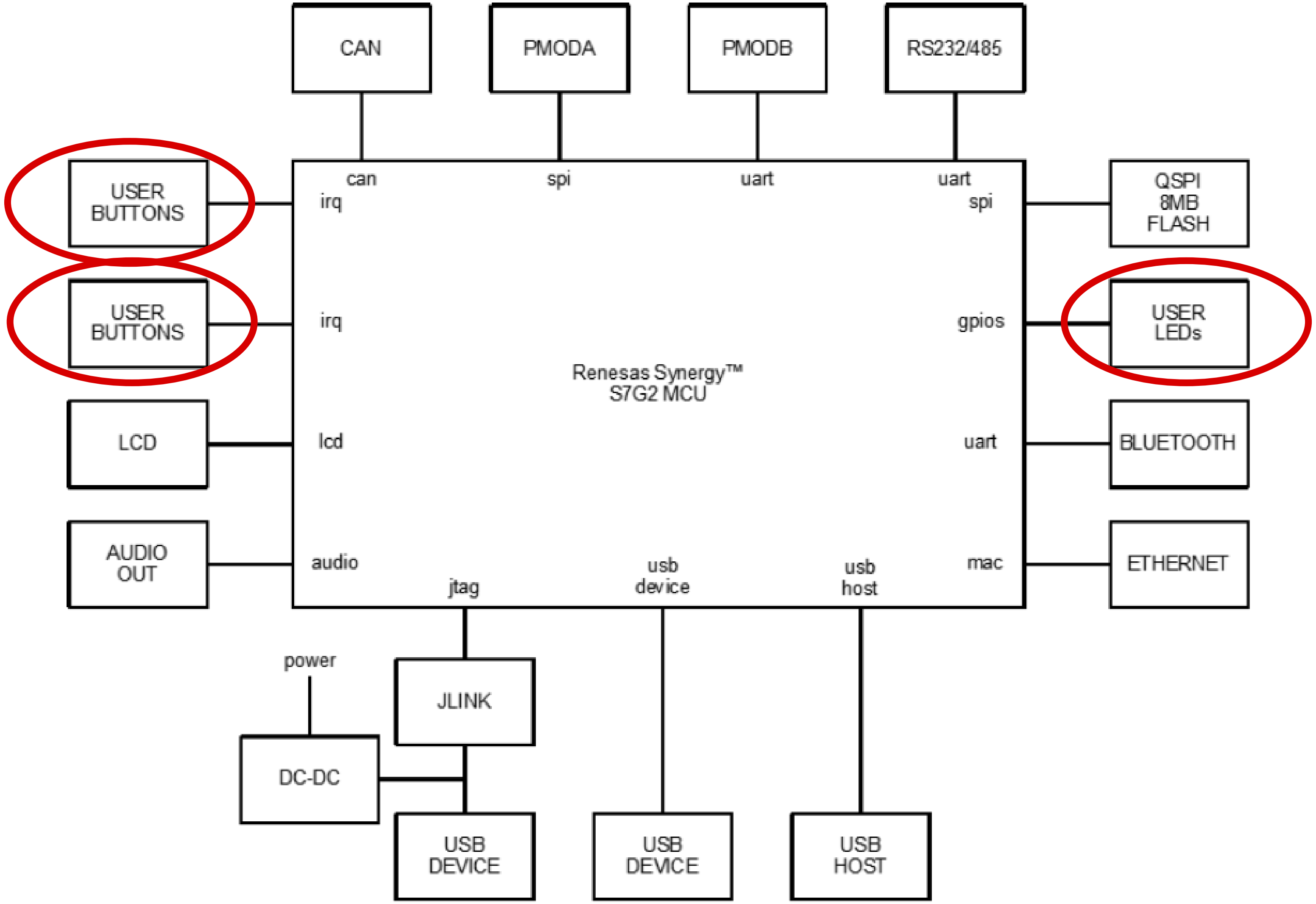


Figure 2: SK-S7G2 block diagram

source: Starter Kit SK-S7G2 User’s Manual

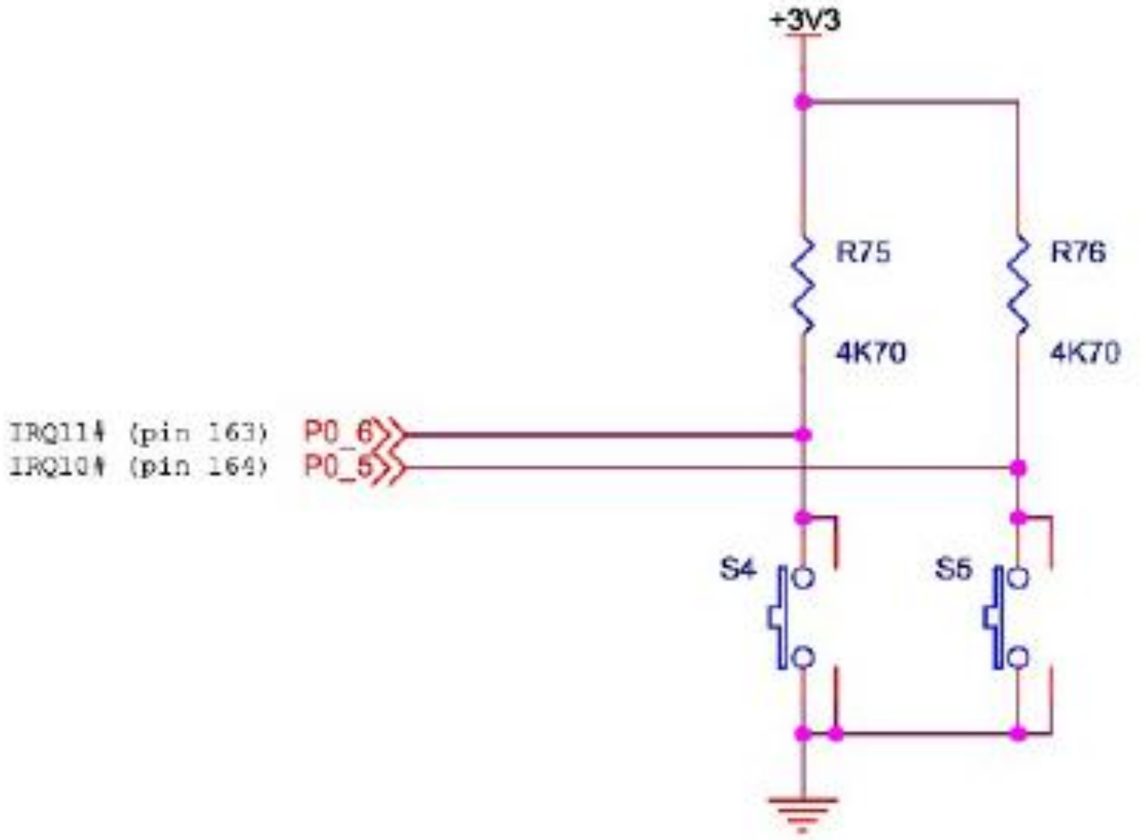
LAB2 – ACTIVITY 4

There are two push-buttons on the SK-S7G2: S4 and S5.

They are directly connected to GPIO pins: P0.6 and P0.5 respectively. There is no debounce circuit between the push-buttons and the MCU pins.

P0.6 (P006) can generate interrupts at IRQ11.

P0.5 (P005) can generate interrupts at IRQ10



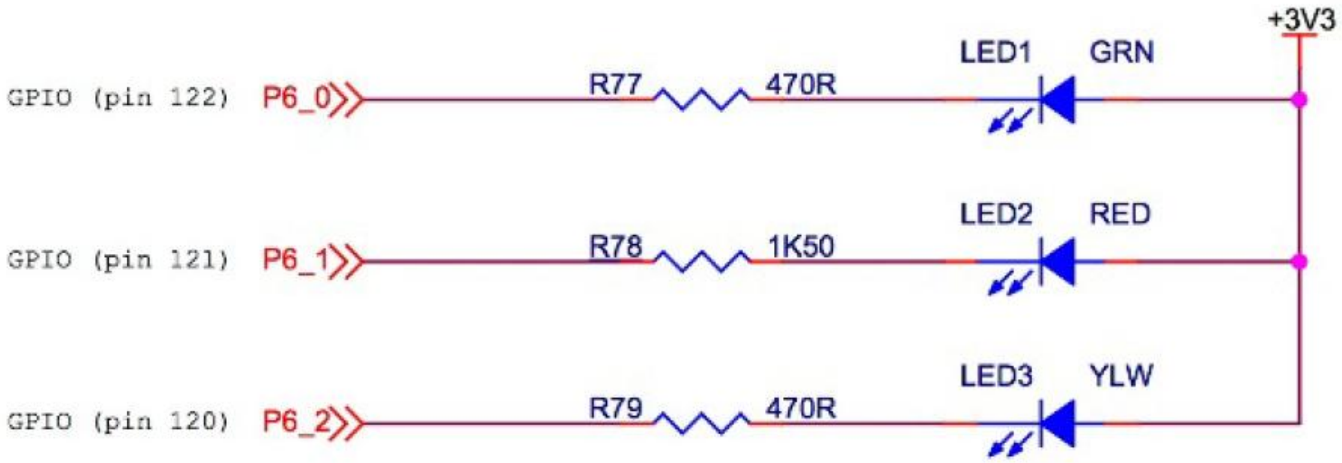
source: Starter Kit SK-S7G2 User’s Manual

LAB2 – ACTIVITY 4

There are three LEDs on the SK-S7G2 that are controllable by GPIO pins: LED1 (green), LED2 (red), and LED3 (yellow).

They are connected to GPIO pins P6.0, P6.1 and P6.2 respectively.

The LEDs turn on when a logic level 0 is written to the pin and they turn off writing a logic level 1.



source: Starter Kit SK-S7G2 User's Manual

LAB2 – ACTIVITY 4

Activity 4 – Study the Hardware platform (SK-S7G2 board and S7G2 MCU)

The next manual to be studied is the Renesas S7 Series Microcontrollers User’s Manual.pdf

Relevant information in this manual concerning the current project is:

- Overview
- CPU
- Clock Generation
- Event Link Controller
- I/O Ports
- Timers

LAB2 – ACTIVITY 4

S7G2 microcontroller’s block diagram with indication of blocks of interest.

Since the user’s manual of the S7G2 has more than 2000 pages, it is important to keep focus on what is relevant to this project.

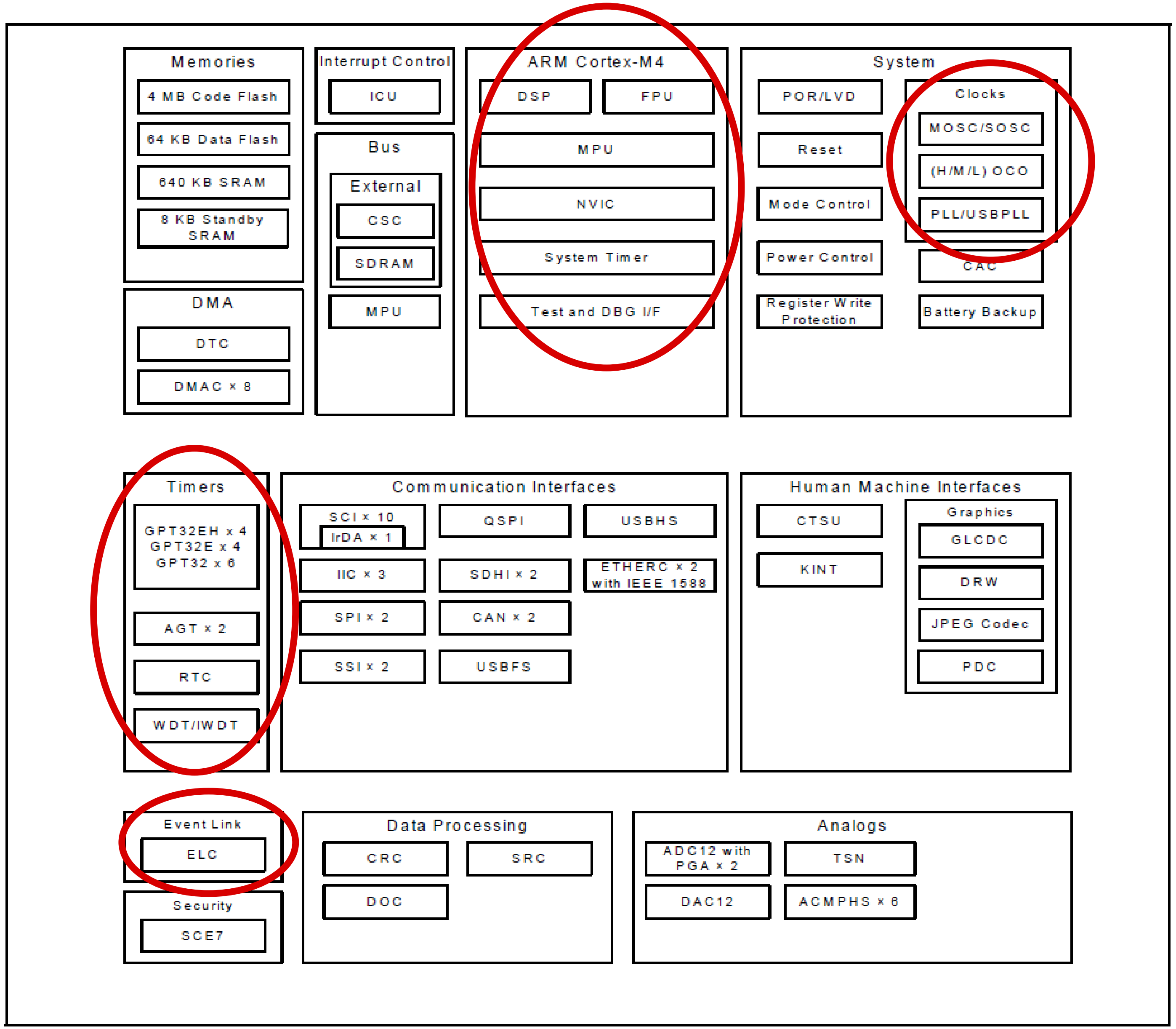


Figure 1.1 **Block diagram**

source: Renesas S7 Series Microcontrollers User’s Manual

LAB2 – ACTIVITY 4

The S7G2 microcontroller’s clock generation circuit is very flexible and can provide several different clock frequencies to different parts of the MCU. Following limits must be observed:

Flash clock (FCLOCK) - 60 MHz

Core clock (ICLOCK) - 240 MHz

PCLKA - 120 MHz

PCLKB - 60 MHz

PCLKC - 60 MHz

PCLKD - 120 MHz

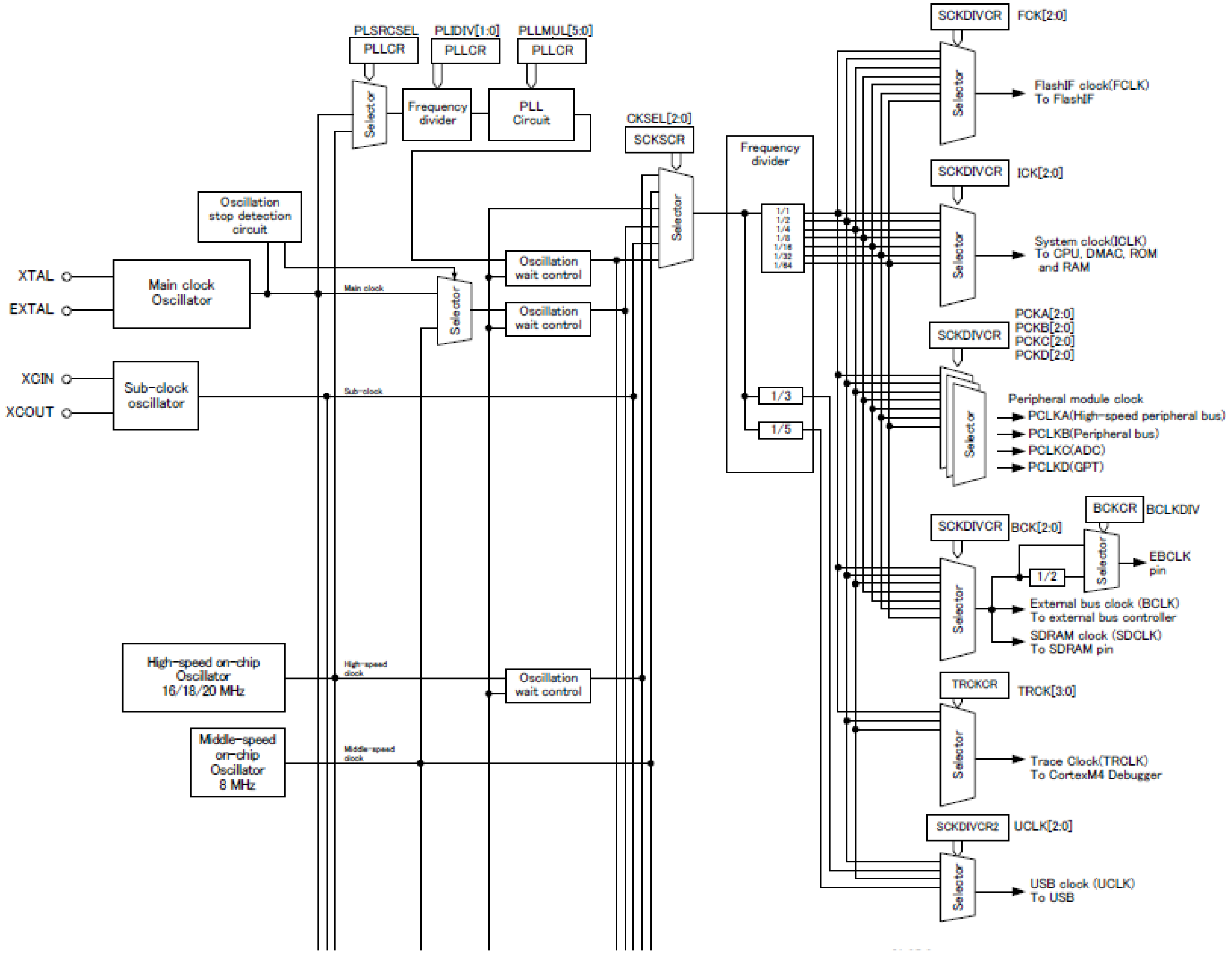


Fig 9.1 (partial)

source: Renesas S7 Series Microcontrollers User’s Manual

LAB2 – ACTIVITY 4

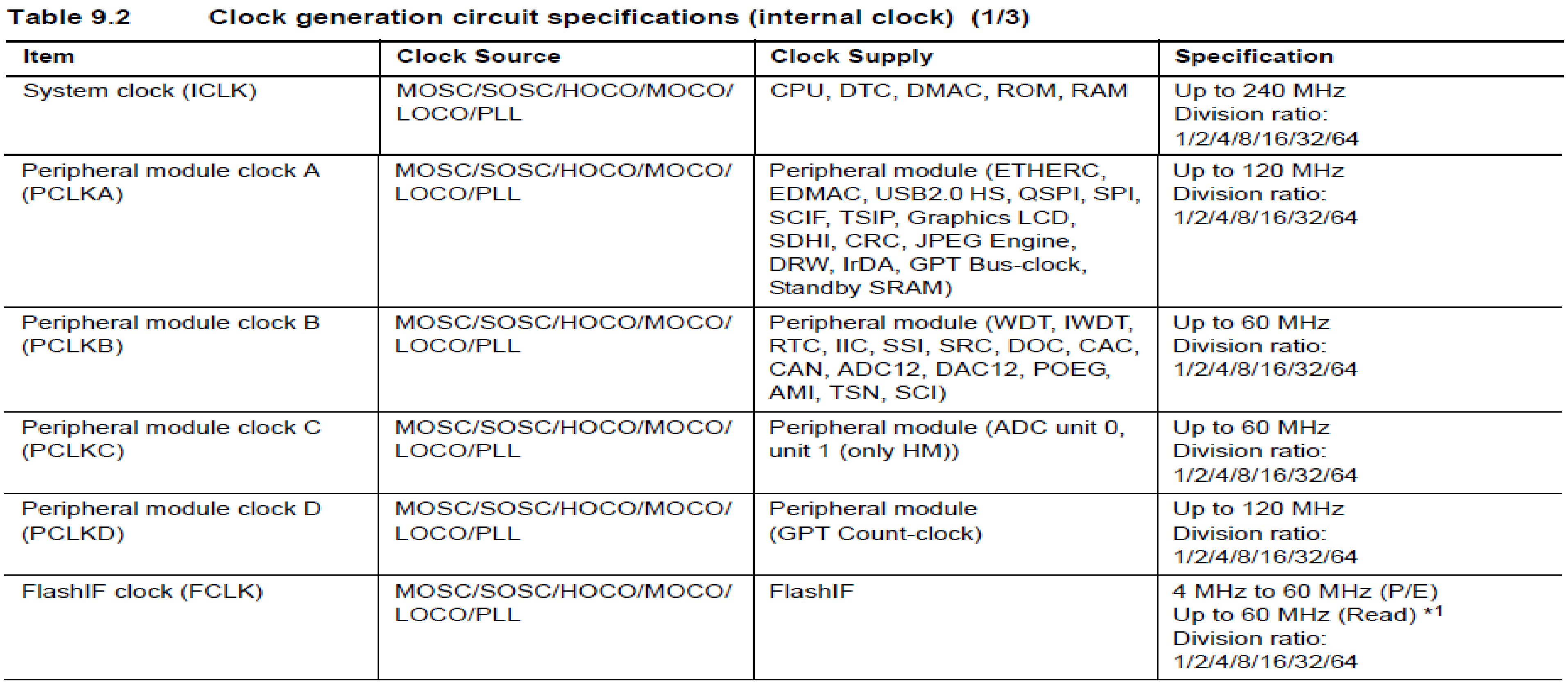


Table 9.2 Clock generation circuit specifications (internal clock) (1/3)

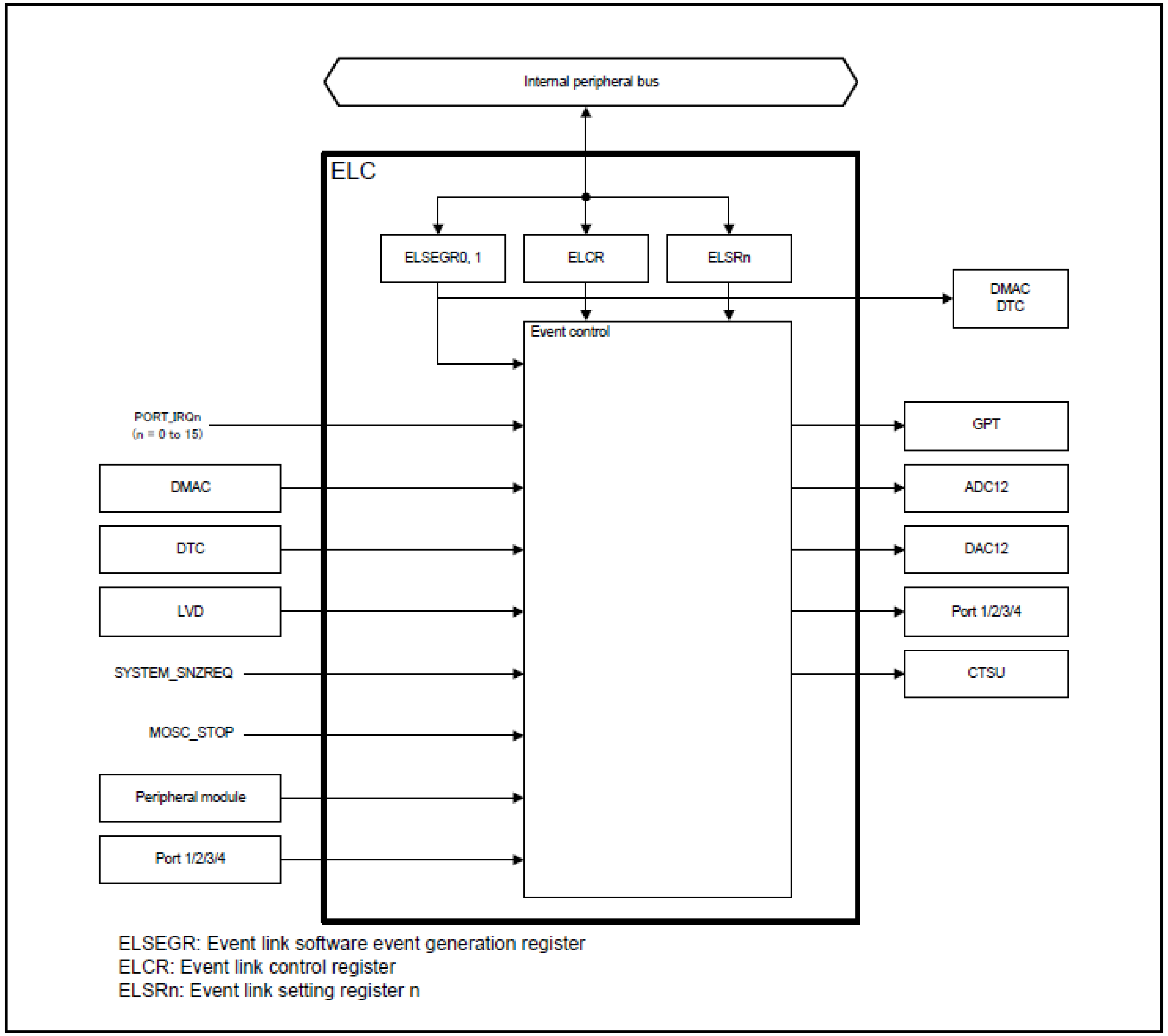
Item	Clock Source	Clock Supply	Specification
System clock (ICLK)	MOSC/SOSC/HOCO/MOCO/LOCO/PLL	CPU, DTC, DMAC, ROM, RAM	Up to 240 MHz Division ratio: 1/2/4/8/16/32/64
Peripheral module clock A (PCLKA)	MOSC/SOSC/HOCO/MOCO/LOCO/PLL	Peripheral module (ETHERC, EDMAC, USB2.0 HS, QSPI, SPI, SCIF, TSIP, Graphics LCD, SDHI, CRC, JPEG Engine, DRW, IrDA, GPT Bus-clock, Standby SRAM)	Up to 120 MHz Division ratio: 1/2/4/8/16/32/64
Peripheral module clock B (PCLKB)	MOSC/SOSC/HOCO/MOCO/LOCO/PLL	Peripheral module (WDT, IWDT, RTC, IIC, SSI, SRC, DOC, CAC, CAN, ADC12, DAC12, POEG, AMI, TSN, SCI)	Up to 60 MHz Division ratio: 1/2/4/8/16/32/64
Peripheral module clock C (PCLKC)	MOSC/SOSC/HOCO/MOCO/LOCO/PLL	Peripheral module (ADC unit 0, unit 1 (only HM))	Up to 60 MHz Division ratio: 1/2/4/8/16/32/64
Peripheral module clock D (PCLKD)	MOSC/SOSC/HOCO/MOCO/LOCO/PLL	Peripheral module (GPT Count-clock)	Up to 120 MHz Division ratio: 1/2/4/8/16/32/64
FlashIF clock (FCLK)	MOSC/SOSC/HOCO/MOCO/LOCO/PLL	FlashIF	4 MHz to 60 MHz (P/E) Up to 60 MHz (Read) *1 Division ratio: 1/2/4/8/16/32/64

source: Renesas S7 Series Microcontrollers User’s Manual

LAB2 – ACTIVITY 4

The ELC (Event Link Controller) connects events generated by peripheral modules to other peripheral modules. This direct communication among modules does not require intervention from the CPU.

The ELC is not required in this project, however, it is a compulsory module in the configuration of the SSP.



ELSEGR: Event link software event generation register
ELCR: Event link control register
ELSRn: Event link setting register n

Figure 19.1 ELC block diagram (n = 0 to 18)

source: Renesas S7 Series Microcontrollers User’s Manual

LAB2 – ACTIVITY 4

The block diagram of a GPIO pin.

Among the configurable features of a pin are:

- Input or output,
- Enable a pull-up on input,
- Output drive capability: low, medium, high
- Use pin for analog function (ADC or DAC);
- Use pin for peripheral function (e.g. SPI);
- Some inputs are 5V tolerant;
- Pins can generate interrupt on edges of input signal.

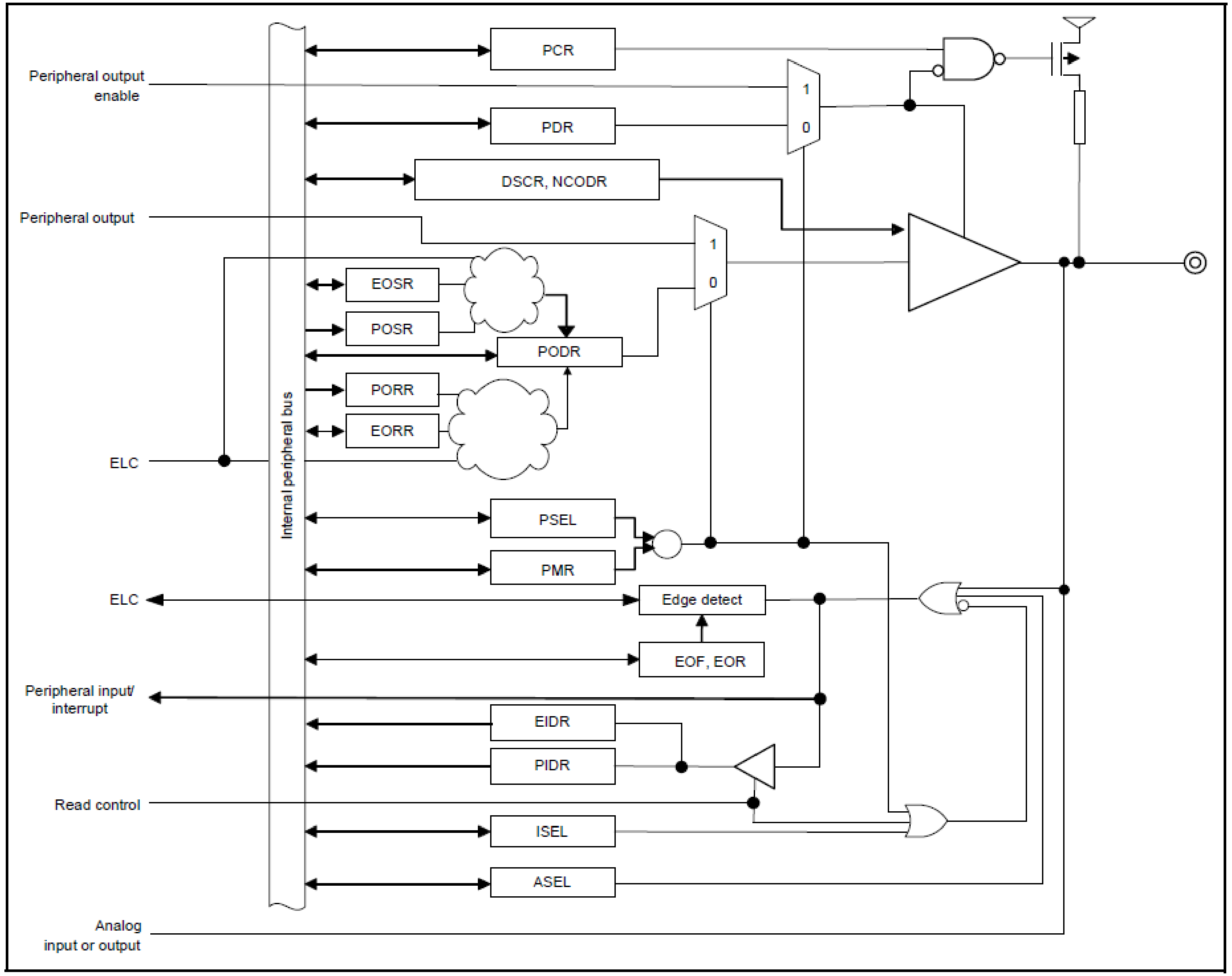


Figure 20.1 I/O Port registers connection diagram

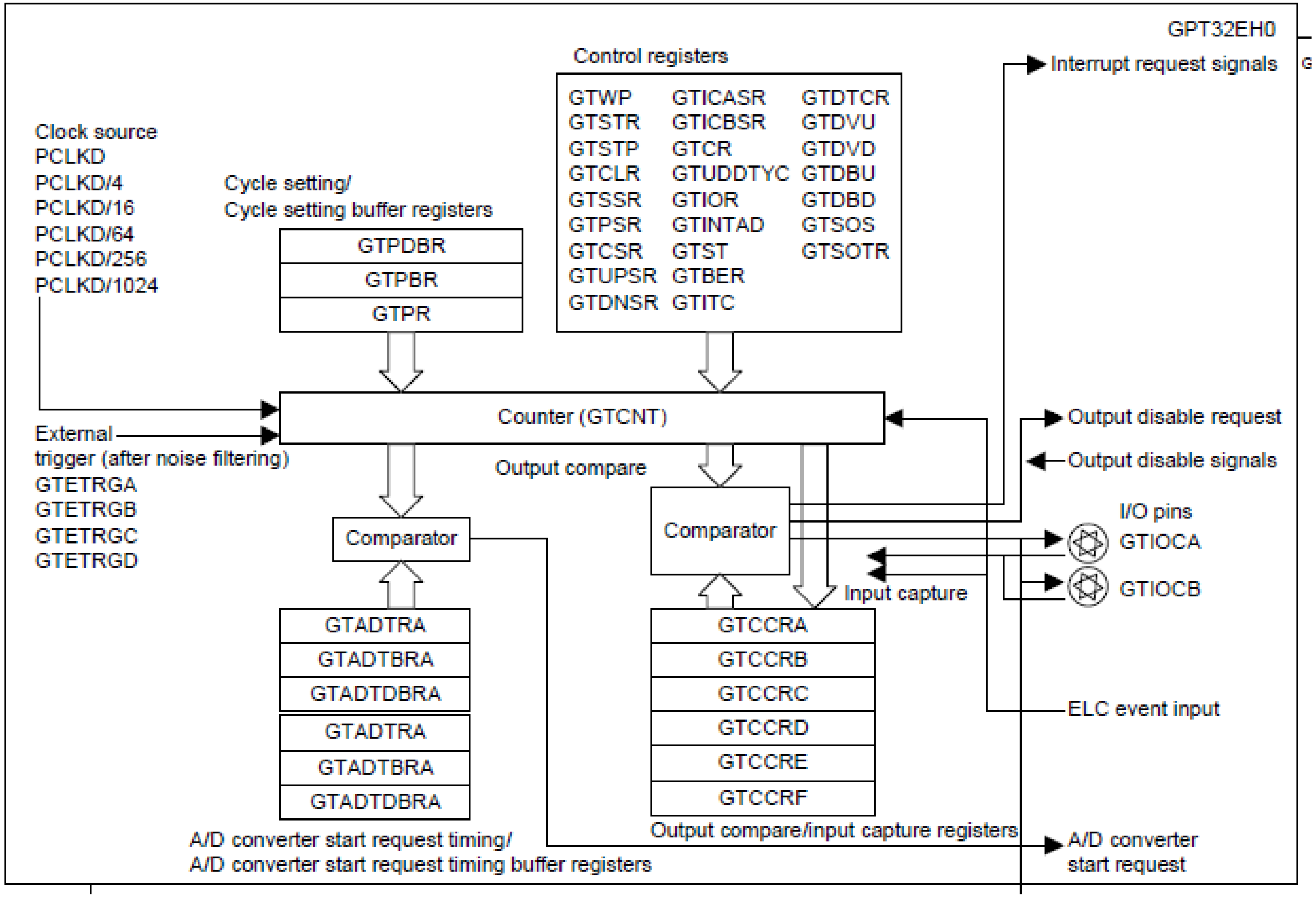
source: Renesas S7 Series Microcontrollers User’s Manual

LAB2 – ACTIVITY 4

The block diagram of GPT (General PWM Timer).

GPT characteristics:

- 32-bit counter
- Counts up or down
- Can generate interrupts
- Can start an ADC conversion
- Counts PCLKD pulses
- Periodic or single-shot
- 14 channels, each is a 32-bit counter

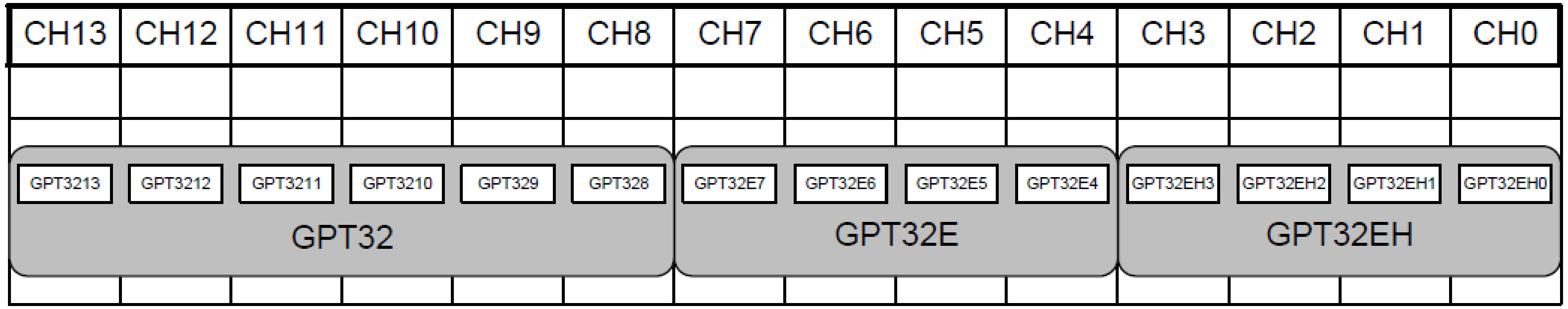


source: Renesas S7 Series Microcontrollers User's Manual

LAB2 – ACTIVITY 4

The 14 timer channels are grouped into:

- 4x EH – enhanced high resolution
- 4x E – enhanced
- 6x – conventional



CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
GPT3213	GPT3212	GPT3211	GPT3210	GPT329	GPT328	GPT32E7	GPT32E6	GPT32E5	GPT32E4	GPT32EH3	GPT32EH2	GPT32EH1	GPT32EH0
GPT32						GPT32E				GPT32EH			

source: Renesas S7 Series Microcontrollers User’s Manual

LAB2 – ACTIVITY 5

Activity 5 – Software Framework Study

Study the Renesas Synergy Software Package v1.7.5 (link) available in the Synergy Gallery; specifically Section 4.2.21 about the HAL GPT (General PWM Timer).

https://synergygallery.renesas.com/media/products/1/384/en-US/r11um0140eu0106-synergy-ssp-v175.pdf

Also, study the Module Guide for the GPT HAL (link).

https://www.renesas.com/us/en/doc/products/renesas-synergy/apn/r11an0091eu0101-synergy-gpt-hal-mod-guide.pdf

LAB2 – ACTIVITY 6

Activity 6 – Design

(identification of the SW components that need to be developed, their interfaces, algorithms and interfaces to other software components)

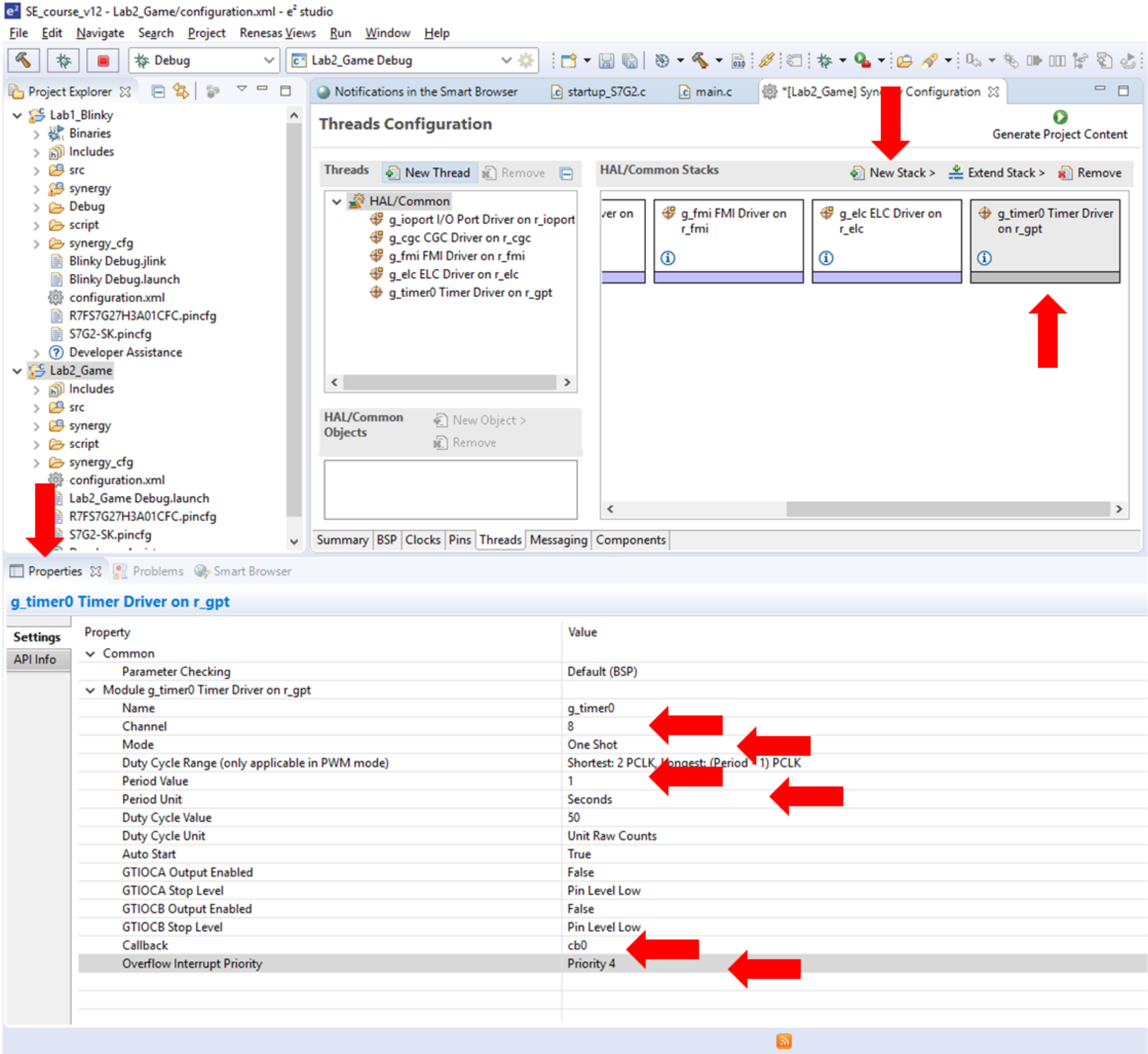
Components of the SSP to be used:

- Timer Driver on r_gpt. This component uses one of the 14 channels of the General PWM Timer. We selected channel 8 for no particular reason.
- External IRQ Driver on r_icu. This component manages the interrupt generated by one of the GPIO lines. Here we select IRQ 11 because on the SK-S7G2 board, the S5 push-button is connected to the IRQ11 pin.

LAB2 – ACTIVITY 6

Timer Driver

on r_gpt



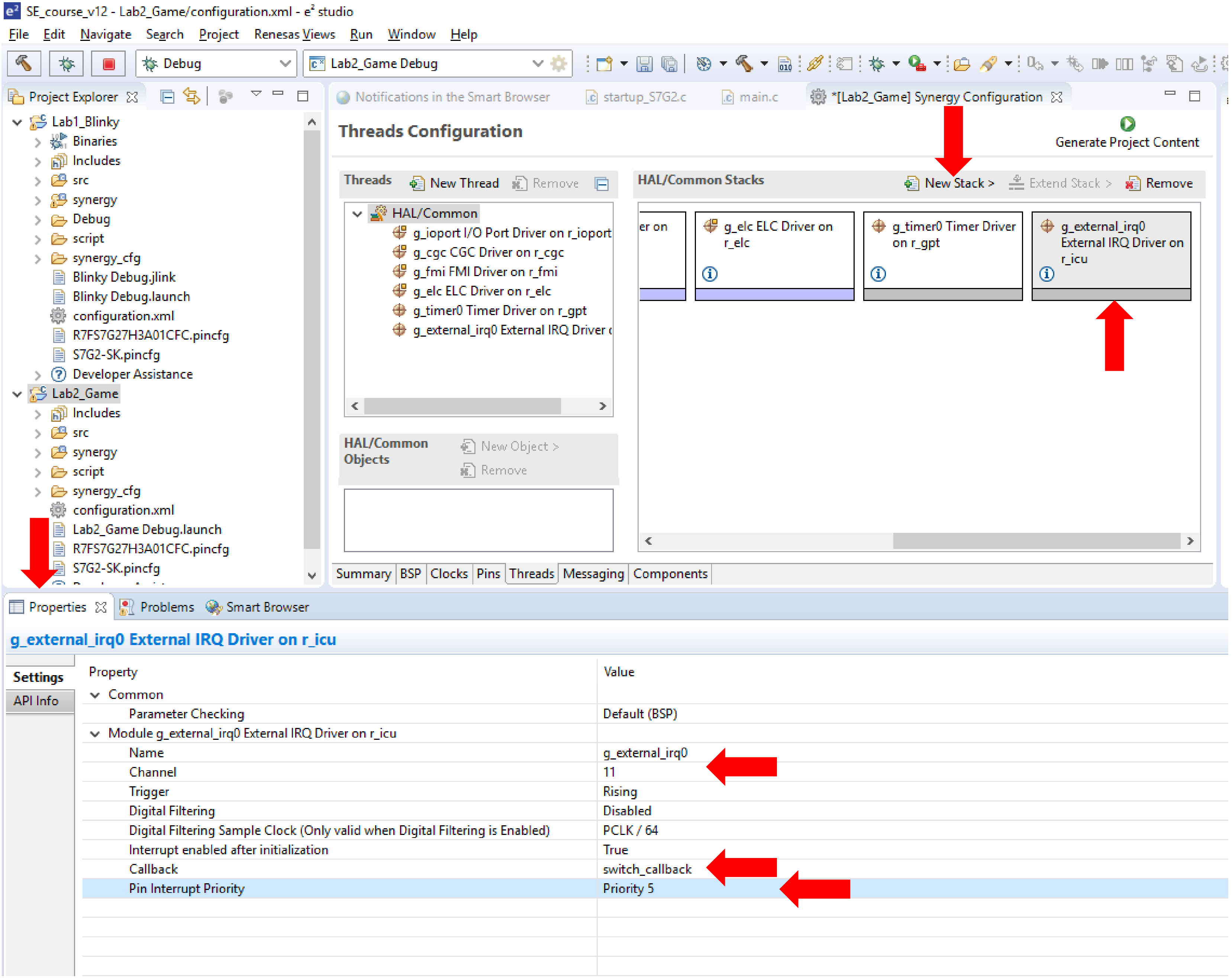
g_timer0 Timer Driver on r_gpt

Property	Value
Common	
Parameter Checking	Default (BSP)
Module g_timer0 Timer Driver on r_gpt	
Name	g_timer0
Channel	8
Mode	One Shot
Duty Cycle Range (only applicable in PWM mode)	Shortest: 2 PCLK, Longest: (Period - 1) PCLK
Period Value	1
Period Unit	Seconds
Duty Cycle Value	50
Duty Cycle Unit	Unit Raw Counts
Auto Start	True
GTIOCA Output Enabled	False
GTIOCA Stop Level	Pin Level Low
GTIOCB Output Enabled	False
GTIOCB Stop Level	Pin Level Low
Callback	cb0
Overflow Interrupt Priority	Priority 4

LAB2 – ACTIVITY 6

External IRQ Driver

on r_icu



g_external_irq0 External IRQ Driver on r_icu

Property	Value
Common	
Parameter Checking	Default (BSP)
Module g_external_irq0 External IRQ Driver on r_icu	
Name	g_external_irq0
Channel	11
Trigger	Rising
Digital Filtering	Disabled
Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	PCLK / 64
Interrupt enabled after initialization	True
Callback	switch_callback
Pin Interrupt Priority	Priority 5

LAB2 – ACTIVITY 6

Algorithm:

1. Initialize the components for the Timer Driver and the External IRQ Driver by calling their open API functions. This also starts the timer on its 1 second period as configured in the Synergy Configuration Tool.
2. Turn off the LEDs
3. Wait for the timer to expire. Its callback function informs via a flag (shared volatile global variable).
4. Reprogram the timer for 3 seconds and restart count (API functions `reset, periodSet, restart`).
5. Wait for the push-button to be pressed. Again, its callback function informs via another flag.
6. Get the current count of the timer (API functions `counterGet`) and save this value to a variable (this variable will be examined with the debugger).

This algorithm is to be implemented by modifying the function `hal_entry`, adding two callback functions and the required global variables.

LAB2 – ACTIVITIES 7 TO 11

Based on the previous study:

- After creating a new Synergy C project (lab2) based on the Blinky template, on the Threads tab of Synergy Configuration, add a Timer Driver (on r_gpt) and configure its properties to: channel 8, single shot, period of 1 second, callback `cb0`, interrupt priority 4.
- Modify the file hal_entry.c to:
  - Turn off all three LEDs, a LED is turned off by writing IOPORT_LEVEL_HIGH to the GPIO pin.
  - Call the GPT API function open(`g_timer0.p_ctrl,g_timer0.p_cfg`) to configure and start the timer. The two parameters of this function are created by the Synergy Configuration tool when the button “Generate Project Content” is pressed.
  - Create a callback function named `cb0`. This function must inform the `hal_entry` function that the timer expired by setting a flag. This flag must be a volatile global variable. Reminder: volatile informs the compiler that its value changes outside the control of the `hal_entry` function.

LAB2 – ACTIVITIES 7 TO 11

Modify the file hal_entry.c to (cont.):

- In function `hal_entry`, wait for the flag to change value, meaning 1 second has passed;
- Turn LEDs on;
- Reset the timer (API function reset);
- Set a new period of 3 seconds (API function `periodSet`);
- Start the timer again (API function `start`);
- Make sure the timer is operating by making successive calls to API function `counterGet`;
- Test if the program is operating correctly up to this point.

LAB2 – ACTIVITIES 7 TO 11

- In the Synergy Configuration tab, add an input driver of type external IRQ driver on r_icu. Configure its properties to: channel 11, callback function `switch_callback`, interrupt priority 5. Recall that, from the study of the board manual, we learned that push-button S4 is connected to IRQ 11.
- Modify the file hal_entry.c to:
  - In the `hal_entry` function, call the open API function of the `g_external_irq0` to configure this component;
  - Create a callback function named `switch_callback`. This function also informs the `hal_entry` function that the timer expired by setting another volatile flag;
  - In the `hal_entry` function, wait for this flag to change then get the number of ticks from the counter up to this point. Save it to a variable;
  - Put a breakpoint right after the variable is updated and play the game.

LAB3 – ASSEMBLY PROGRAMMING AND ATPCS

Objectives:

- Develop an assembly routine that is callable from a C program. The assembly routine must follow the ATPCS standard.
- The function to be implemented in assembly generates the histogram of an 8-bit grayscale image.

LAB3 – ASSEMBLY PROGRAMMING AND ATPCS

Learning Objectives:

- Apply the embedded software development process presented in Lab 2
- Define the interface between the C program (caller) and the Assembly function (callee)
- Plan the data structures to be used
- Devise an algorithm to generate a histogram
- Implement, Test, Debug

LAB3 – ASSEMBLY PROGRAMMING AND ATPCS

Activities:

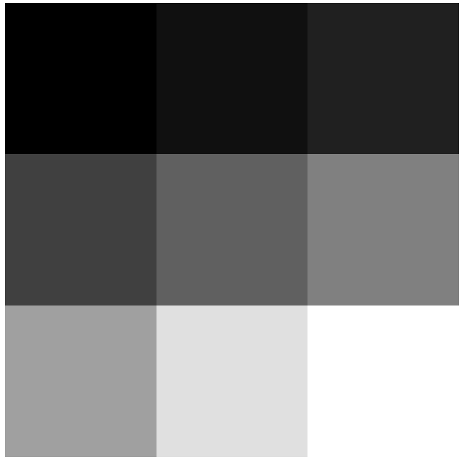
1. Understanding the Problem Domain
2. Problem definition
3. Designing the Data Structures
4. Parameter passing and return of the result
5. Algorithm
6. Implementation
7. Test cases

LAB3 – ACTIVITY 1

Activity 1 – Understanding the Problem Domain

- A raster image or bitmap is composed of dots, or pixels, lay out as a matrix. On a grayscale image, each pixel is represented by a number indicating the level of lighting of that pixel.
- On an 8-bit grayscale image, each pixel is represented by an 8-bit value. Hence, there are 256 levels of gray, ranging from 0 (black) to 255 (white).
- Shown below is a 3 x 3 8-bit grayscale image (9 pixels in total) and the corresponding 9-pixel image.

0	16	32
64	96	128
160	224	255

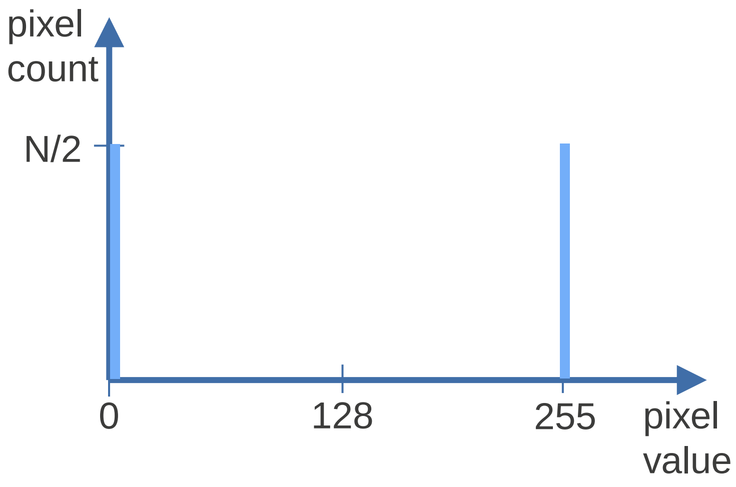


LAB3 – ACTIVITY 1

Activity 1 – Understanding the Problem Domain

A histogram is a graphical representation of the tonal distribution of an image. On the horizontal axis there are the possible values that a pixel can have (0-255 in this example) and the vertical axis presents the quantities of pixels with a given luminosity level.

An image with N pixels were half of them are white and half of them are black would have a histogram like this:



Histograms are very useful in digital image processing, to determine thresholds, to adjust brightness and contrast, to identify problems, and many more.

To construct a histogram, all pixels of an image have to be processed, hence, it is desirable to have efficient algorithms and implementations for better performance.

LAB3 – ACTIVITY 2

Activity 2 – Problem Definition 1/2

Develop a function, to be implemented in assembly, that constructs the histogram of an

8-bit grayscale bitmap image.

Input parameters:

- Image width - number of pixels across the image.
- Image height - height of image in pixels
- Starting address - address of the first pixel in memory. Each pixel occupies one byte. The image is represented by a matrix were image[0][0] is the upper left pixel of the image. The matrix is stored by rows, hence, the next address holds image[0][1] (next pixel on the upper row).
- Histogram - address of a 256-position vector holding 16-bit unsigned integers that hold the pixel counts. The histogram has invalid data when the function is called.

Output: 16-bit unsigned integer indicating the total number of pixels processed.

LAB3 – ACTIVITY 2

Activity 2 – Problem Definition 2/2

Restrictions:

The total number of pixels in the image (i.e. width x height) must be less than 64K (65,536)

Error codes:

Function returns 0 to indicate an error (e.g. image too large)

Function prototype

```
uint16_t EightBitHistogram(uint16_t width, uint16_t height, uint8_t * p_image, uint16_t *
p_histogram);
```

LAB3 – ACTIVITY 3

Activity 3 – Designing the Data Structures

The two important data structures for this problem are the matrix that holds the bitmap and the vector that stores the histogram.

The bitmap matrix has its number of columns equal to the width of the image and its number of rows equal to the height of the image. Each element stored in the matrix is an 8-bit unsigned integer (`uint8_t`) that represents the gray level of the corresponding pixel, 0 being black and 255 being white.

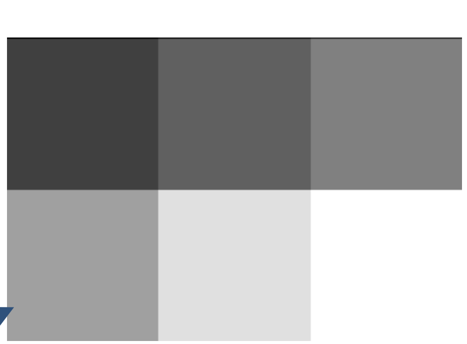
The histogram vector has size 256, hence, its indexes run from 0 to 255. The i_{th} element of the vector stores the count of pixels at value i.

`histogram[i]` = number of pixels whose value is i.

Hence, adding up all elements of the vector must result in a number equal to width x height.

For a 2-pixel high by 3-pixel wide image the matrix would be:

```
uint8_t bitmap[2][3] = {
       {64,96,128},
       {160,224,255}};
```



LAB3 – ACTIVITY 4

Activity 4 – Parameter passing and return of the result

Considering that the function prototype is:

```
uint16_t EightBitHistogram(uint16_t width, uint16_t height, uint16_t * p_image, uint8_t *
p_histogram);
```

By ATPCS the parameters are in registers R0 to R3:

`width` - in R0 (upper half of R0 is 0)

`height` - in R1 (upper half of R1 is 0)

`p_image` - in R2

`p_histogram` - in R3

The result is passed by R0 (upper half is 0)

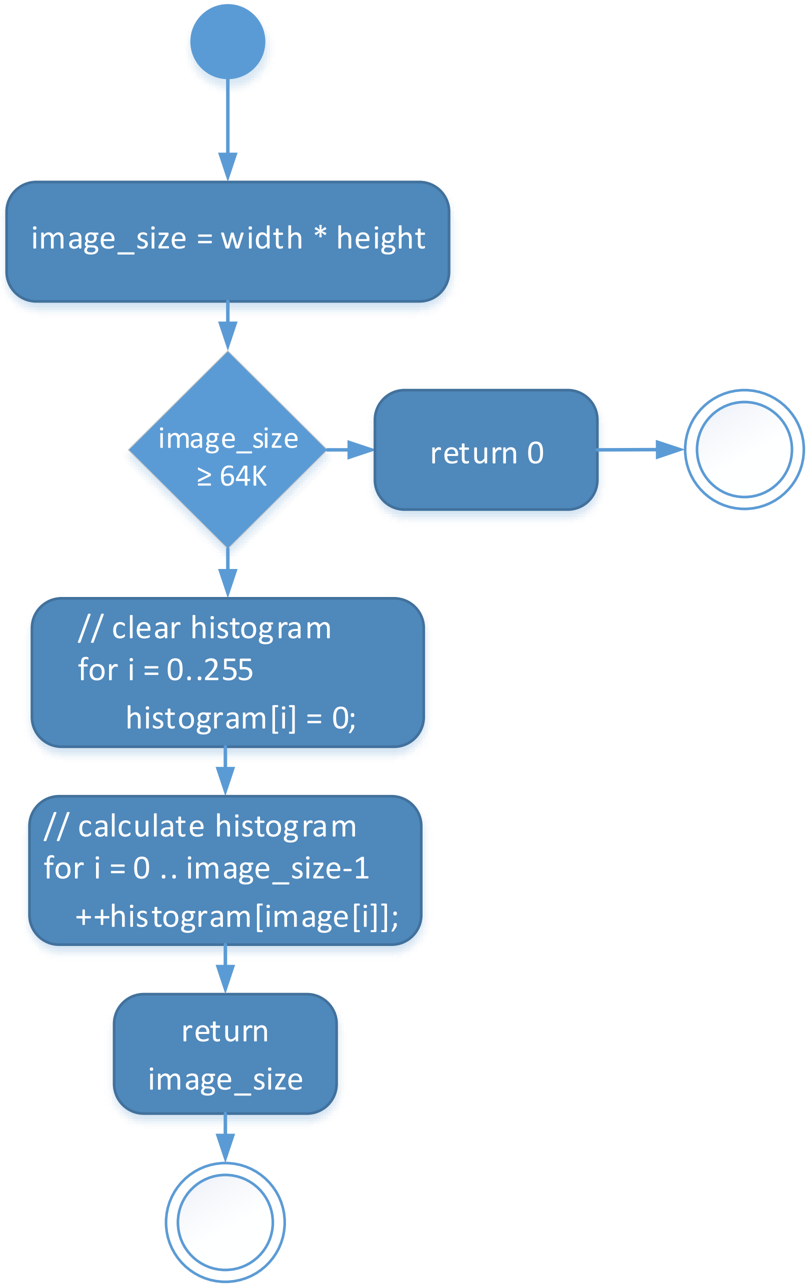
LAB3 – ACTIVITY 5

Activity 5 – Algorithm

One possible solution design is presented here using the UML 2.5 Activity diagram notation.

If the implementation requires registers other than R0..R3 and R12 then there is the need to push these registers at the start and restore them at the end.

Note that the histogram is constructed in a very efficient way by simply using the value of each pixel as an index to the position in the histogram that must be incremented.

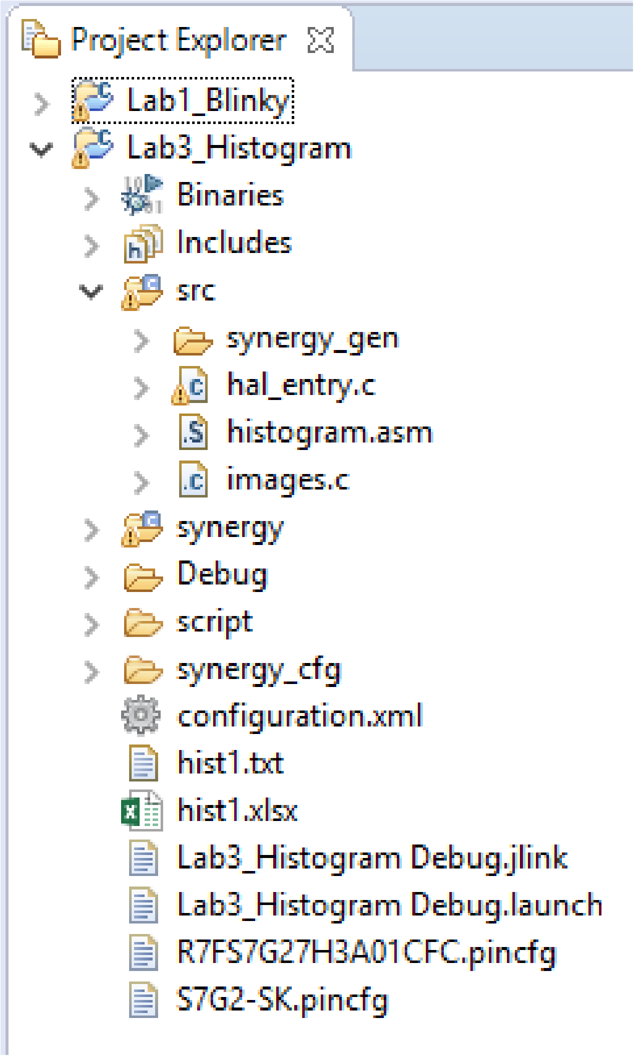


LAB3 – ACTIVITY 6

Activity 6 – Implementation

A possible organization of the source files is:

- hal_entry.c — this is the C program that calls the assembly function
  `hal_entry` is executed after initialization;
  it calls **`EightBitHistogram`**
  then presents the results on the virtual console.
- histogram.asm — this is the assembly source file where
  `EightBitHistogram` is defined.
- images.c — holds the matrices with the test images.



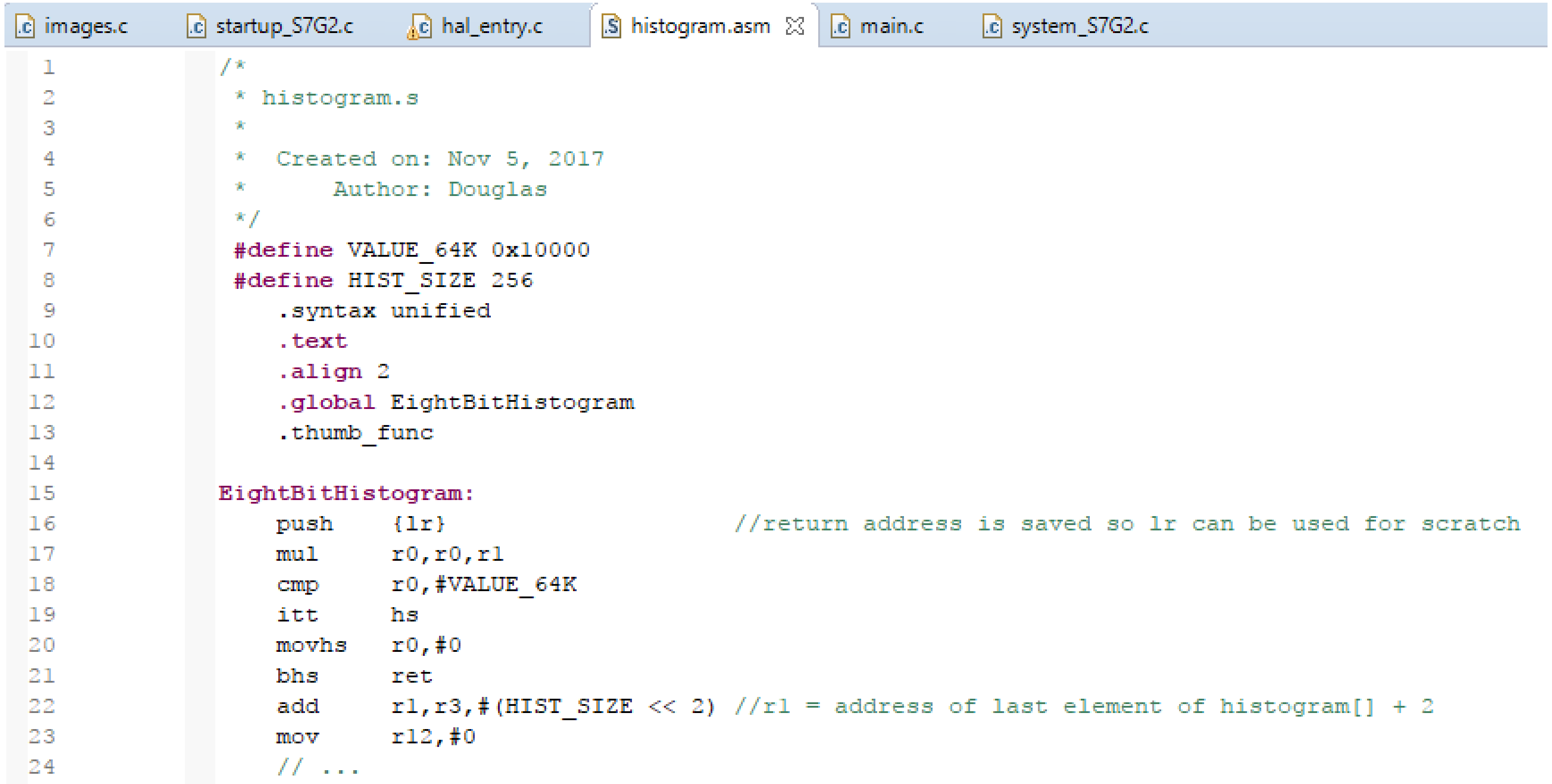
Project Explorer
- Lab1_Blinky
- Lab3_Histogram
  - Binaries
  - Includes
  - src
    - synergy_gen
    - hal_entry.c
    - histogram.asm
    - images.c
  - synergy
  - Debug
  - script
  - synergy_cfg
  - configuration.xml
  - hist1.txt
  - hist1.xlsx
  - Lab3_Histogram Debug.jlink
  - Lab3_Histogram Debug.launch
  - R7FS7G27H3A01CFC.pincfg
  - S7G2-SK.pincfg

Tips on how to use the Renesas Virtual Console are in slide: LAB 5 - Activity 3

LAB3 – ACTIVITY 6

Activity 6 – Implementation

The assembly source file requires assembler directives at the start, as shown here.



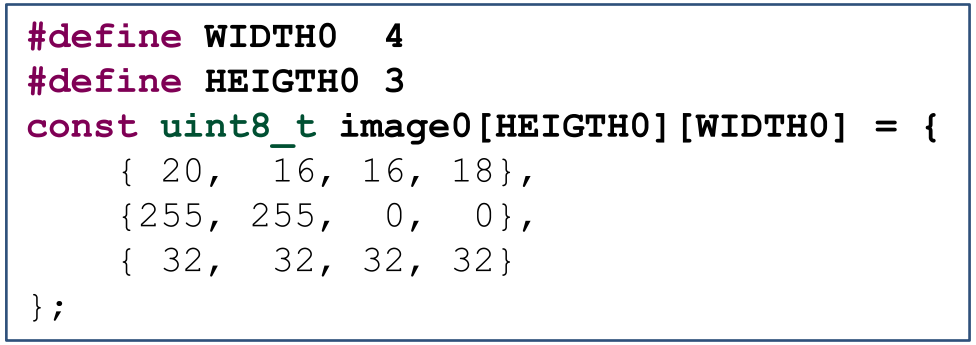
```
/*
 * histogram.s
 *
 *  Created on: Nov 5, 2017
 *      Author: Douglas
 */
 #define VALUE_64K 0x10000
 #define HIST_SIZE 256
    .syntax unified
    .text
    .align 2
    .global EightBitHistogram
    .thumb_func

EightBitHistogram:
    push    {lr}                    //return address is saved so lr can be used for scratch
    mul     r0,r0,r1
    cmp     r0,#VALUE_64K
    itt     hs
    movhs   r0,#0
    bhs     ret
    add     r1,r3,#(HIST_SIZE << 2) //r1 = address of last element of histogram[] + 2
    mov     r12,#0
    // ...
```

LAB3 – ACTIVITY 7

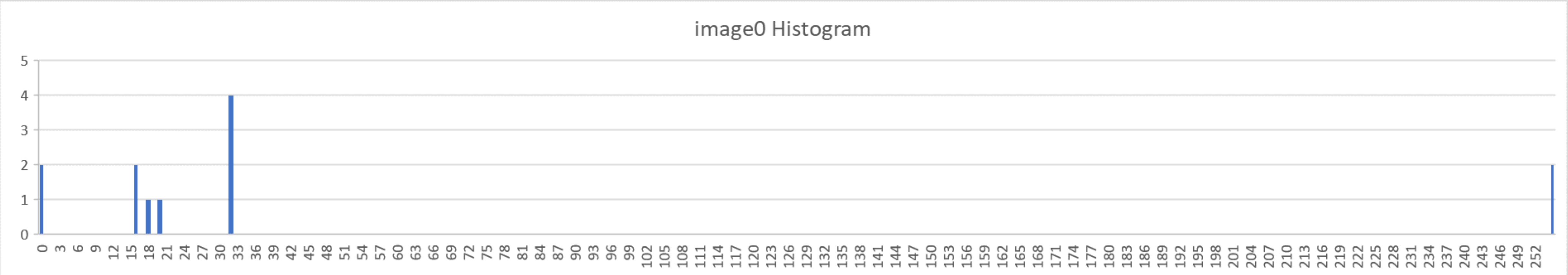
Activity 7 – Test Cases

Two test cases are provided. The first is matrix image0 that has only 3 lines and 4 columns. Such a small test case is important to debug the implementation on a step-by-step execution.

Shown here is the contents of image0 and the corresponding histogram.



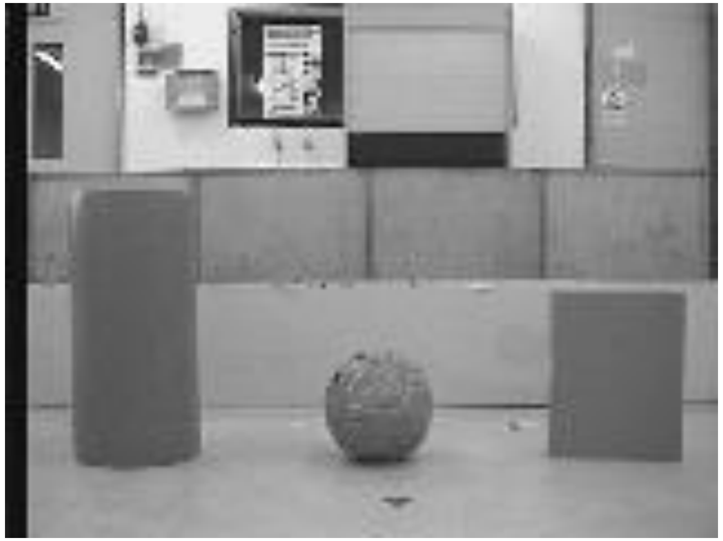
```
#define WIDTH0  4
#define HEIGTH0 3
const uint8_t image0[HEIGTH0][WIDTH0] = {
    { 20,  16, 16, 18},
    {255, 255,  0,  0},
    { 32,  32, 32, 32}
};
```



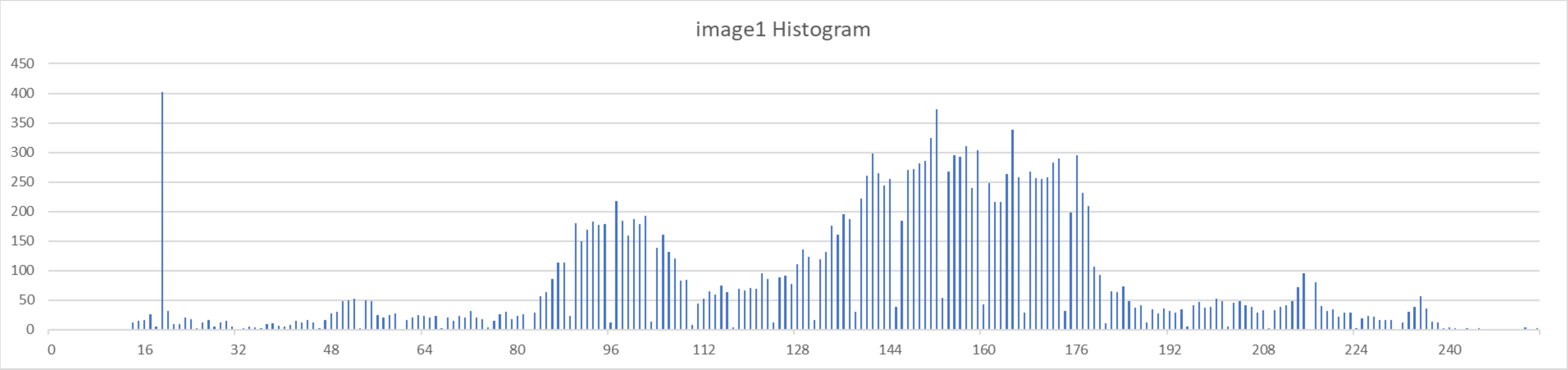
LAB3 – ACTIVITY 7

Activity 7 – Test Cases

The second test case is the test image presented here. Its pixels are encoded in the matrix image1. Its size is 160 x 120 pixels.



The corresponding histogram is presented below:



LAB4 – TIMER DEVICE DRIVER

Objectives:

Use a GPT (General PWM Timer) to generate a 100 Hz rectangular waveform with a 25% duty cycle. The SSP components for the GPT are NOT to be used. Here, the student is to exercise the direct interaction with the GPT hardware.

LAB4 – TIMER DEVICE DRIVER

Learning Objectives:

- Study a hardware peripheral with the objective to develop its device driver
- Identify the relevant registers of a peripheral for a given application
- Design an algorithm of a device driver
- Evaluate alternatives of means to interact with the registers of hardware peripheral

LAB4 – TIMER DEVICE DRIVER

Activities:

1. Study the GPT (General PWM Timer of the S7G2 MCU)
2. Determine the sequence that the GPT registers must be programmed and the respective values
3. Design an algorithm to achieve the purpose
4. Implement and test

further reading for this Lab:

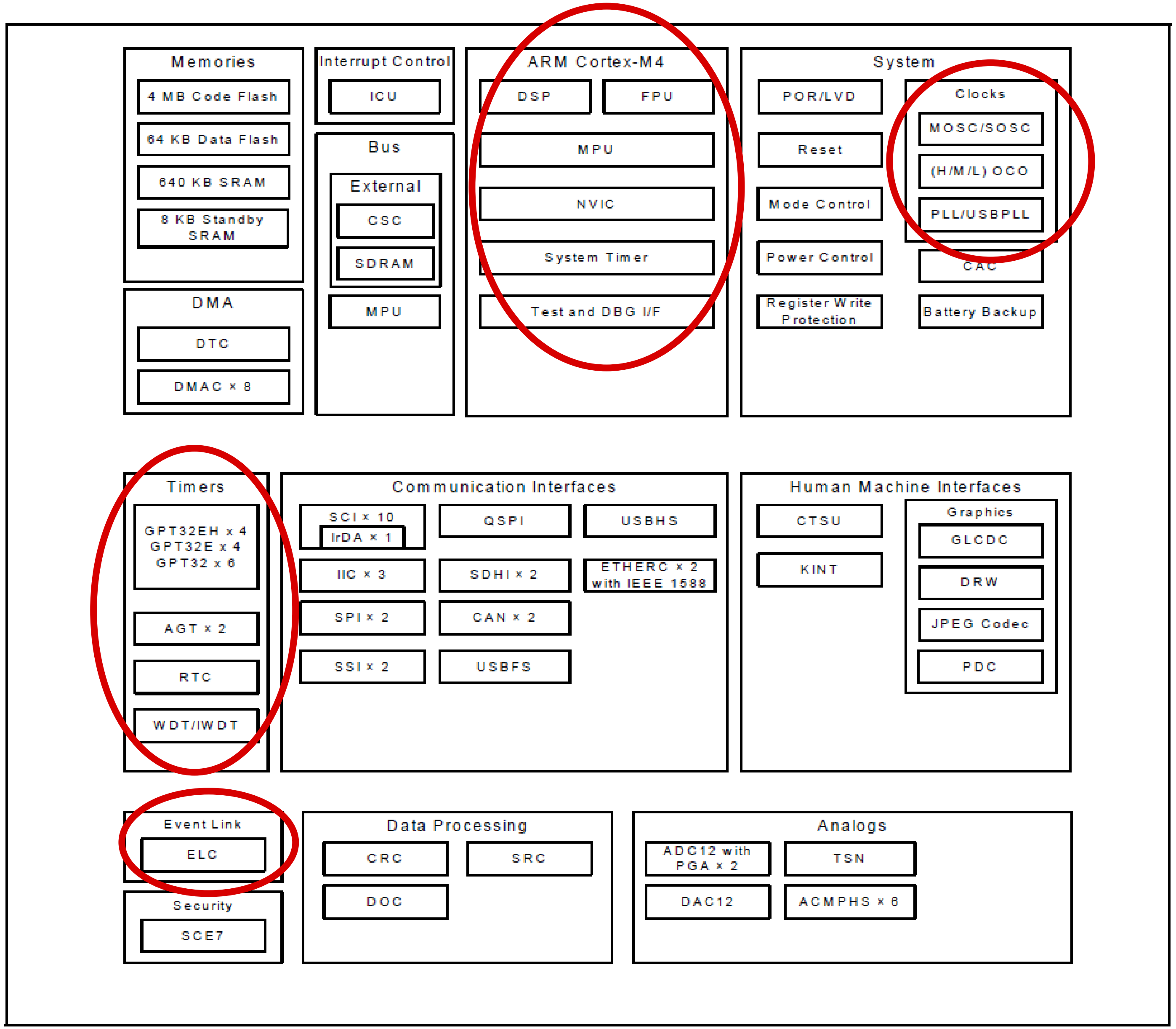
Renesas Synergy Software Package v1.7.5 User's Manual
r11um0140eu0106-synergy-ssp-v175

GPT HAL Module Guide
r11an0091eu0101-synergy-gpt-hal-mod-guide.pdf

LAB4 – ACTIVITY 1

Activities:

- S7G2 microcontroller’s block diagram with indication of blocks of interest.
- Since the user’s manual of the S7G2 has more than 2000 pages, it is important to keep focus on what is relevant to this project.



Memories
4 MB Code Flash
64 KB Data Flash
640 KB SRAM
8 KB Standby SRAM

DMA
DTC
DMAC × 8

Interrupt Control
ICU

Bus
External
CSC
SDRAM
MPU

ARM Cortex-M4
DSP
FPU
MPU
NVIC
System Timer
Test and DBG I/F

System
POR/LVD
Reset
Mode Control
Power Control
Register Write Protection
Clocks
MOSC/SOSC
(H/M/L) OCO
PLL/USBPLL
CAC
Battery Backup

Timers
GPT32EH × 4
GPT32E × 4
GPT32 × 6
AGT × 2
RTC
WDT/IWDT

Communication Interfaces
SCI × 10
IrDA × 1
QSPI
USBHS
IIC × 3
SDHI × 2
ETHERC × 2 with IEEE 1588
SPI × 2
CAN × 2
SSI × 2
USBFS

Human Machine Interfaces
CTSU
KINT
Graphics
GLCDC
DRW
JPEG Codec
PDC

Event Link
ELC

Security
SCE7

Data Processing
CRC
SRC
DOC

Analogs
ADC12 with PGA × 2
TSN
DAC12
ACMPHS × 6

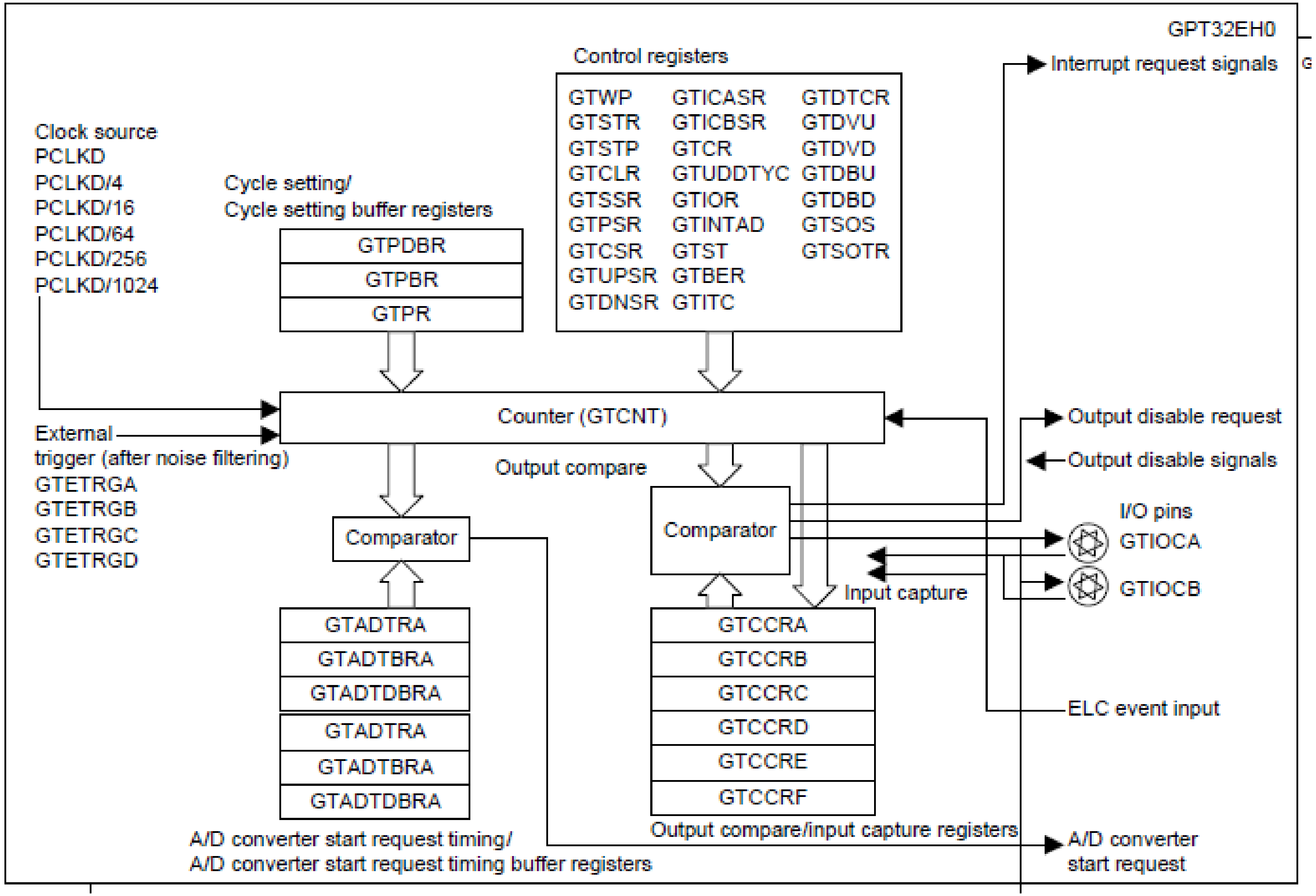
Figure 1.1 **Block diagram**

source: Renesas S7 Series Microcontrollers User’s Manual

LAB4 – ACTIVITY 1

The block diagram of GPT (General PWM Timer).

- GPT characteristics:
- 32-bit counter
- counts up or down
- can generate interrupts
- can start an ADC conversion
- counts PCLKD pulses
- periodic or single-shot
- 14 channels, each is a 32-bit counter

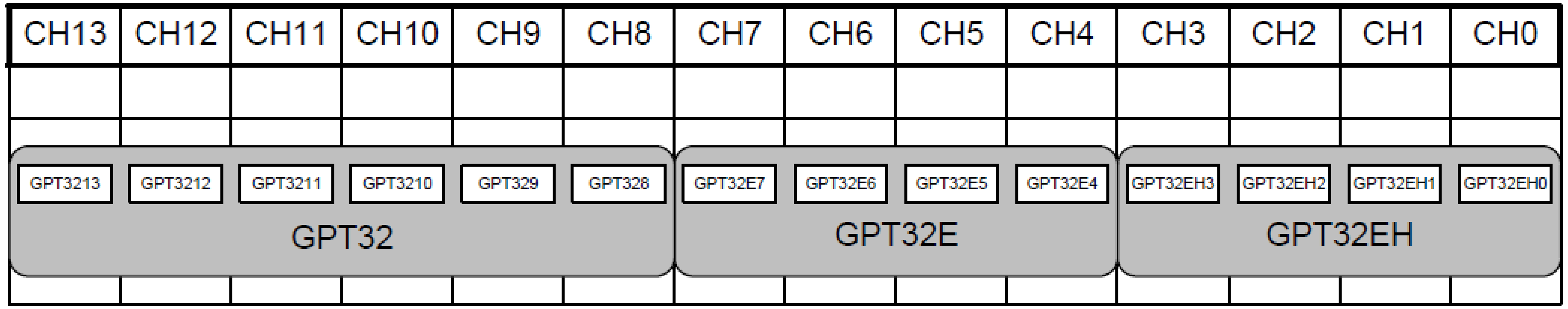


source: Renesas S7 Series Microcontrollers User's Manual

LAB4 – ACTIVITY 1

The 14 timer channels are grouped into:

- 4x EH – enhanced high resolution
- 4x E – enhanced
- 6x – conventional



CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
GPT3213	GPT3212	GPT3211	GPT3210	GPT329	GPT328	GPT32E7	GPT32E6	GPT32E5	GPT32E4	GPT32EH3	GPT32EH2	GPT32EH1	GPT32EH0
GPT32						GPT32E				GPT32EH			

source: Renesas S7 Series Microcontrollers User’s Manual

LAB4 – ACTIVITY 1

Registers of the GPT

GTWP	: General PWM Timer Write-Protection Register
GTSTR	: General PWM Timer Software Start Register
GTSTP	: General PWM Timer Software Stop Register
GTCLR	: General PWM Timer Software Clear Register
GTSSR	: General PWM Timer Start Source Select Register
GTPSR	: General PWM Timer Stop Source Select Register
GTCSR	: General PWM Timer Clear Source Select Register
GTUPSR	: General PWM Timer Up Count Source Select Register
GTDNSR	: General PWM Timer Down Count Source Select Register
GTICASR	: General PWM Timer Input Capture Source Select Register A
GTICBSR	: General PWM Timer Input Capture Source Select Register B
GTCR	: General PWM Timer Control Register
GTUDDTYC	: General PWM Timer Count Direction and Duty Setting Register
GTIOR	: General PWM Timer I/O Control Register
GTINTAD	: General PWM Timer Interrupt Output Setting Register
GTST	: General PWM Timer Status Register
GTBER	: General PWM Timer Buffer Enable Register
GTITC	: General PWM Timer Interrupt and A/D Converter Start Request Skipping Setting Register
GTCNT	: General PWM Timer Counter
GTCCRA	: General PWM Timer Compare Capture Register A
GTCCRB	: General PWM Timer Compare Capture Register B
GTCCRC	: General PWM Timer Compare Capture Register C
GTCCRD	: General PWM Timer Compare Capture Register D
GTCCRE	: General PWM Timer Compare Capture Register E
GTCCRF	: General PWM Timer Compare Capture Register F

GTPR	: General PWM Timer Cycle Setting Register
GTPBR	: General PWM Timer Cycle Setting Buffer Register
GTPDBR	: General PWM Timer Cycle Setting Double-Buffer Register
GTADTRA	: General PWM Timer A/D Converter Start Request Timing Register A
GTADTBRA	: General PWM Timer A/D Converter Start Request Timing Buffer Register A
GTADTDBRA	: General PWM Timer A/D Converter Start Request Timing Double-Buffer Register A
GTADTRB	: General PWM Timer A/D Converter Start Request Timing Register B
GTADTBRB	: General PWM Timer A/D Converter Start Request Timing Buffer Register B
GTADTDBRB	: General PWM Timer A/D Converter Start Request Timing Double-Buffer Register B
GTDTCR	: General PWM Timer Dead Time Control Register
GTDVU	: General PWM Timer Dead Time Value Register U
GTDVD	: General PWM Timer Dead Time Value Register D
GTDBU	: General PWM Timer Dead Time Buffer Register U
GTDBD	: General PWM Timer Dead Time Buffer Register D
GTSOS	: General PWM Timer Output Protection Function Status Register
GTSOTR	: General PWM Timer Output Protection Function Temporary Release Register
OPSCR	: Output Phase Switching Control Register

source: Renesas S7 Series Microcontrollers User's Manual

LAB4 – ACTIVITY 2

Some registers outside GPT must be configured first:

- PWPR must be set to 0 and then to 0x40 to write-enable the register P107PFS
  (PWPR is a byte-wide register at 0x4004085C).
- P107PFS must have its field PMR set to 1 and its field PSEL set to 00011b to enable the output signal GTIOCA to be available on P107. Hence, P107 is no longer a GPIO pin but it became a pin connected to the GPT channel 8 peripheral.
  (P107PFS is a word-wide register at 0x4004085C).
- bit-6 of MSTPCRD must be reset to 0 to enable GPT channel 8, otherwise it remains in low power state, hence, not operational
  (MSTPCRD is a word-wide register at 0x40047008);

source: Renesas S7 Series Microcontrollers User's Manual

LAB4 – ACTIVITY 2

GTWP - General PWM Timer Write-Protection Register

Most registers of the GPT are protected against accidental modification, these can only be written to after write-enabled by GTWP. After reset the registers are write-enabled.

To write-enable, GTWP must be written with (`0xA5 << 8 | 0`)

The affected registers are: GTSSR, GTPSR, GTCSR, GTUPSR, GTDNSR, GTICASR, GTIBCSR, GTCR, GTUDDTYC, GTIOR, GTINTAD,GTST, GTBER, GTITC, GTCNT, GTCCRA, GTCCRB, GTCCRC, GTCCRD, GTCCRE, GTCCRF, GTPR, GTPBR, GTPDBR, GTADTRA, GTADTBRA, GTADTDBRA, GTADTRB, GTADTBRB, GTADTDBRB, GTDTCR, GTDVU, GTDVD, GTDBU, GTDBD, GTSOS, GTSOTR

Since the default value of GTWP is write-enable, there is no need to change this register.

source: Renesas S7 Series Microcontrollers User’s Manual

LAB4 – ACTIVITY 2

GTSTR - General PWM Timer Software Start Register

One bit for each channel.

Write 1 to start that channel.

Write 0 has no effect.

Bit i controls channel i

A channel may be started by GTCR as well.

source: Renesas S7 Series Microcontrollers User’s Manual

LAB4 – ACTIVITY 2

GTUDDTYC - General PWM Timer Count Direction and Duty Setting Register

bit 0 - UD - set to count UP

bits 17,16 - OADTY - 00 = GTIOCA duty cycle depends on compare match

other bits must remain 0

GTUDDTYC is a word-wide register at 0x40078830.

source: Renesas S7 Series Microcontrollers User’s Manual

LAB4 – ACTIVITY 2

GTIOR - General PWM Timer I/O Control Register

bits 4..0 - 11001b - Initial output is high, low output at GTCCRA compare match,
high output at cycle end.

bit 8 - OAE - set to 1 to enable the GTIOCA pin output

other bits must remain at 0.

GTIOR is a word-wide register at 0x40078834.

source: Renesas S7 Series Microcontrollers User's Manual

LAB4 – ACTIVITY 2

GTCCRA - General PWM Timer Compare Capture Register A

Holds the number of PCLKD ticks at the moment when the PWM signal changes to low, this is, after 25% of the cycle, considering that the cycle starts at high (configured in GTIOR) and remains high for the first 25% of the cycle.

When PCLKD is configured for 120 MHz, and the desired PWM cycle is 10 ms, 25% corresponds to 7.5ms. It is required 300,000 PCLKD cycles for the high time and 1,200,000 PCLKD cycles for the PWM cycle. Since the counting starts at zero, the actual value programmed to GTCCRA is 299,999 (or 0x493DF).

GTCCRA is a word-wide register at 0x4007884C.

source: Renesas S7 Series Microcontrollers User’s Manual

LAB4 – ACTIVITY 2

GTPR - General PWM Timer Cycle Setting Register

Considering the calculation presented for GTCCRA, GTPR must be configured with the value 1.200,000 -1 (or 0x124F7F).

GTPR is a word-wide register at 0x40078864.

source: Renesas S7 Series Microcontrollers User’s Manual

LAB4 – ACTIVITY 2

GTCR - General PWM Timer Control Register

Bits 0 - CST - 1 means count in progress

Bits 18..16 - MD - 000 means saw-wave PWM

Bits 26..24 - TPCS - Timer Prescaler - 000 means PCLKD/1

GTCR is a word-wide register at 0x4007882C.

source: Renesas S7 Series Microcontrollers User’s Manual

LAB4 – ACTIVITY 3

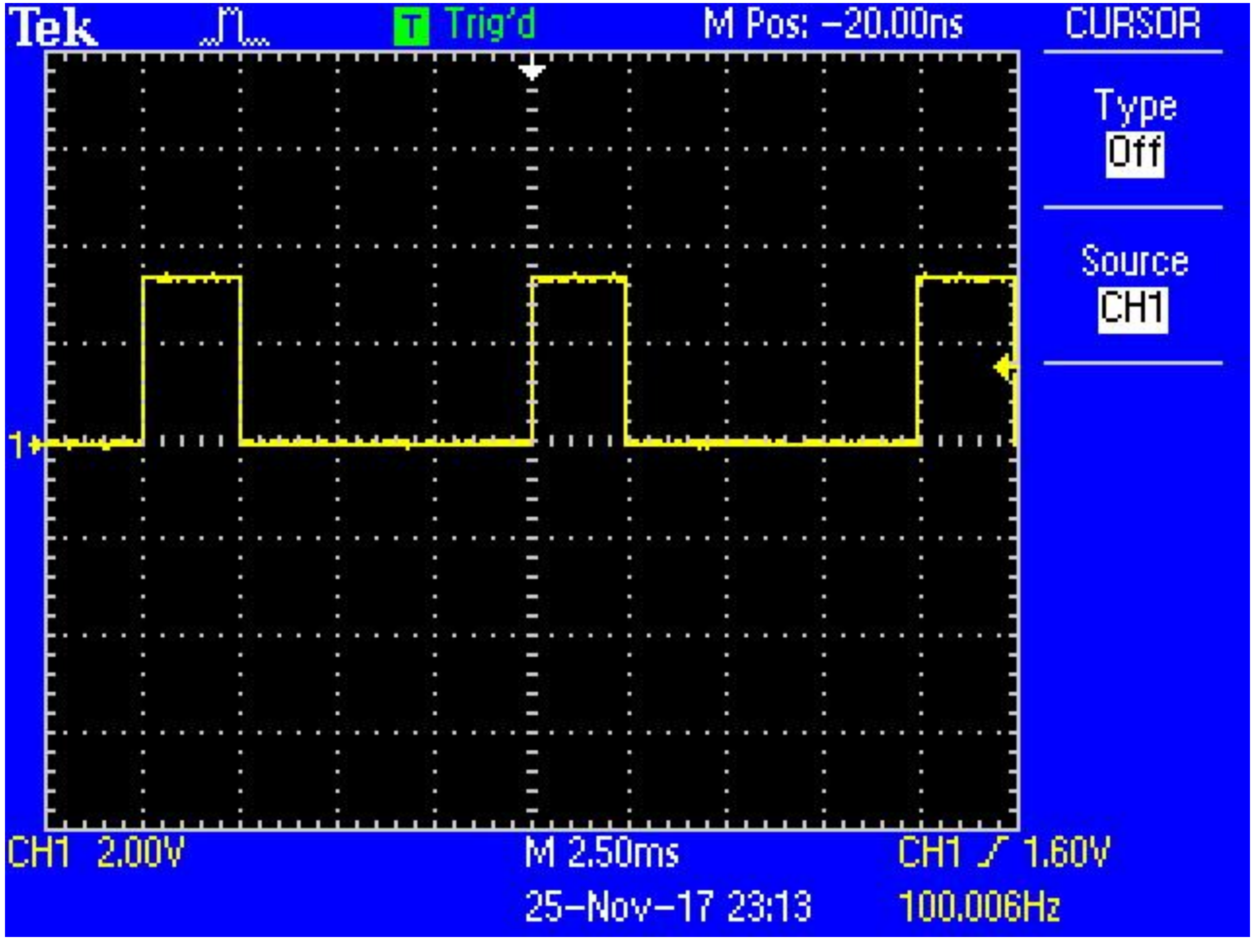
Algorithm

1. Program PWPR to write-enable P107PFS. Write 0 then 0x40 to PWPR.
2. Program P107PFS to put P107 into GTIOCA mode. PSEL = 3, PMR =1
3. Program MSTPCRD bit 6 to enable the power to GPT channel 8. MSTPD6 = 0.
4. Program GTUDDTYC so that the timer counts up and GTIOCA duty cycle depends on compare match to GTCCRA. GTUDDTYC = 1.
5. Program GTIOR so that the cycle starts high and changes to low when a match to GTCCRA occurs. Also, enable GTIOCA output. GTIOR = 0x119.
6. Program GTCCRA to 25% of the cycle (300,000 -1).
7. Program GTPR to the cycle period (1,200,000-1).
8. Start the timer in saw-wave mode with PCLKD/1.

LAB4 – ACTIVITY 4

Verify the operation of the PWM.

Connect a scope to pin P107 (labeled P17 on the board) and verify that a 100 Hz rectangular waveform with a 25% duty cycle is present.



LAB5 – SERIAL COMMUNICATIONS

Objectives:

In the Serial Communications lab, the objective is to utilize the software components of the SSP to build a simple application to transmit and receive via an UART. The actual communication signals are monitored with a scope.

LAB5 – SERIAL COMMUNICATIONS

Learning Objectives:

- Selecting software components in the SSP
- Planning a solution
- SSP Configuration
- Threads configuration
- Implementation/Test/Debug
- Checking serial comm signals with a scope

LAB5 – SERIAL COMMUNICATIONS

Activities:

1. Problem definition
2. Hardware platform study
3. Software framework study
4. Configure the SSP
5. Build/Test/Debug
6. Check signals with scope

further reading for this Lab:

Renesas Synergy Software Package v1.7.5 User’s Manual
r11um0140eu0106-synergy-ssp-v175

UART Communications Framework Module Guide
r11an0192eu0100-synergy-uart-comms-fw-mod-guide.pdf

LAB5 – ACTIVITY 1

Activity 1 – Problem Definition

Transmit the string “Lab5: Serial Comm over RS-232” using the SCI3 TX channel of S7G2

Receive bytes over the RX channel of the same SCI3 port.

Present the received chars on the virtual comm console of e2studio

Non-functional requirements:

make use of the SSP components for UART communication

LAB5 – ACTIVITY 2

Activity 2 - Hardware Platform Study (SK-S7G2 board and S7G2 MCU)

Information obtained from the Starter Kit SK-S7G2 User’s Manual.pdf

- Block diagram
- RS-232 interface and related jumpers

LAB5 – ACTIVITY 2

relevant blocks of the SK-S7G2 board

are marked in red

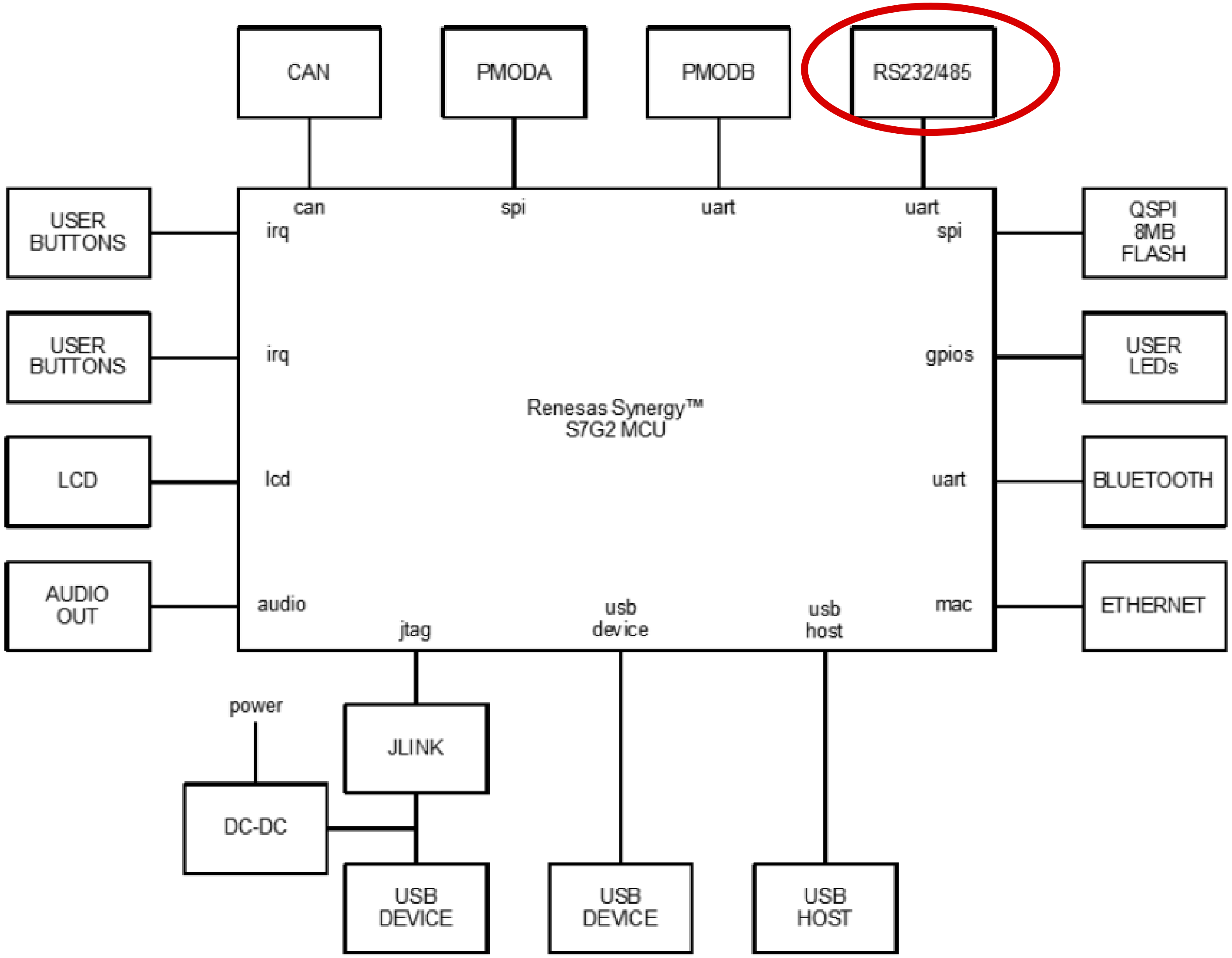


Figure 2: SK-S7G2 block diagram

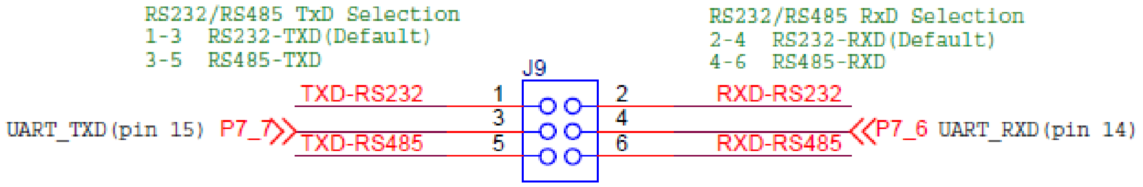
source: Starter Kit SK-S7G2 User's Manual

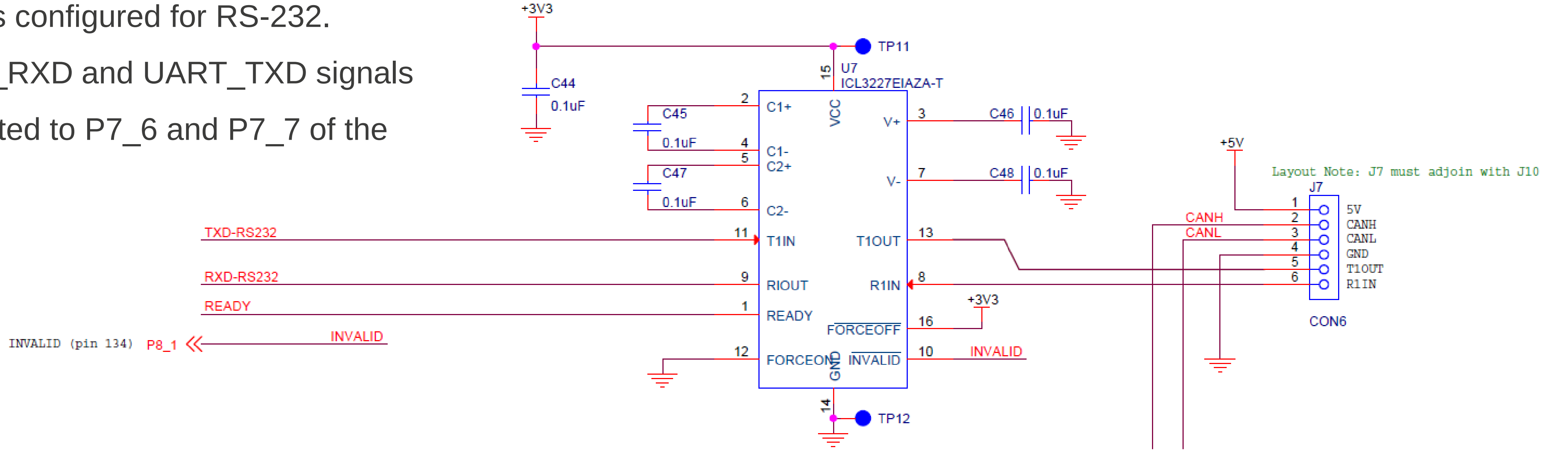
LAB5 – ACTIVITY 2

U7 converts TTL signals to RS-232, these signals are available on J7, pins 5 (TX) and 6 (RX).

J9 must have jumpers on 1-3 and 2-4 so that the board is configured for RS-232.

The UART_RXD and UART_TXD signals are connected to P7_6 and P7_7 of the MCU.





source: Starter Kit SK-S7G2 User's Manual

LAB5 – ACTIVITY 2

P7_6 and P7_7 are RX and TX for SCI 3 of the MCU

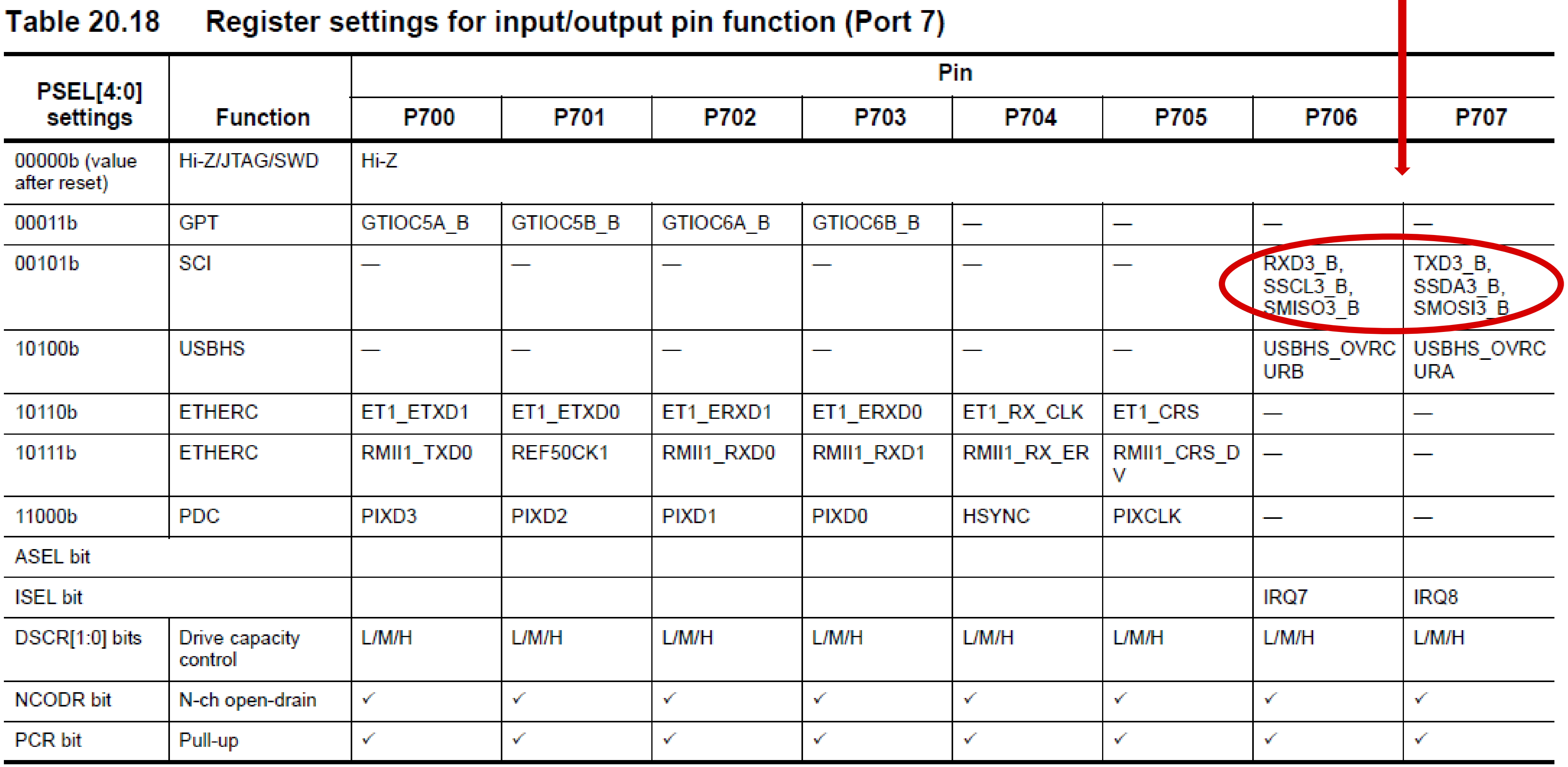


Table 20.18 Register settings for input/output pin function (Port 7)

PSEL[4:0] settings	Function	Pin							
		P700	P701	P702	P703	P704	P705	P706	P707
00000b (value after reset)	Hi-Z/JTAG/SWD	Hi-Z							
00011b	GPT	GTIOC5A_B	GTIOC5B_B	GTIOC6A_B	GTIOC6B_B	—	—	—	—
00101b	SCI	—	—	—	—	—	—	RXD3_B, SSCL3_B, SMISO3_B	TXD3_B, SSDA3_B, SMOSI3_B
10100b	USBHS	—	—	—	—	—	—	USBHS_OVRCURB	USBHS_OVRCURA
10110b	ETHERC	ET1_ETXD1	ET1_ETXD0	ET1_ERXD1	ET1_ERXD0	ET1_RX_CLK	ET1_CRS	—	—
10111b	ETHERC	RMII1_TXD0	REF50CK1	RMII1_RXD0	RMII1_RXD1	RMII1_RX_ER	RMII1_CRS_DV	—	—
11000b	PDC	PIXD3	PIXD2	PIXD1	PIXD0	HSYNC	PIXCLK	—	—
ASEL bit									
ISEL bit								IRQ7	IRQ8
DSCR[1:0] bits	Drive capacity control	L/M/H	L/M/H	L/M/H	L/M/H	L/M/H	L/M/H	L/M/H	L/M/H
NCODR bit	N-ch open-drain	✓	✓	✓	✓	✓	✓	✓	✓
PCR bit	Pull-up	✓	✓	✓	✓	✓	✓	✓	✓

source: Renesas S7 Series Microcontrollers User's Manual

LAB5 – ACTIVITY 2

One of the operating modes of an SCI is asynchronous communications (UART).

Among the possible interrupt sources of the SCI are end of transmission and char received (receive data full).

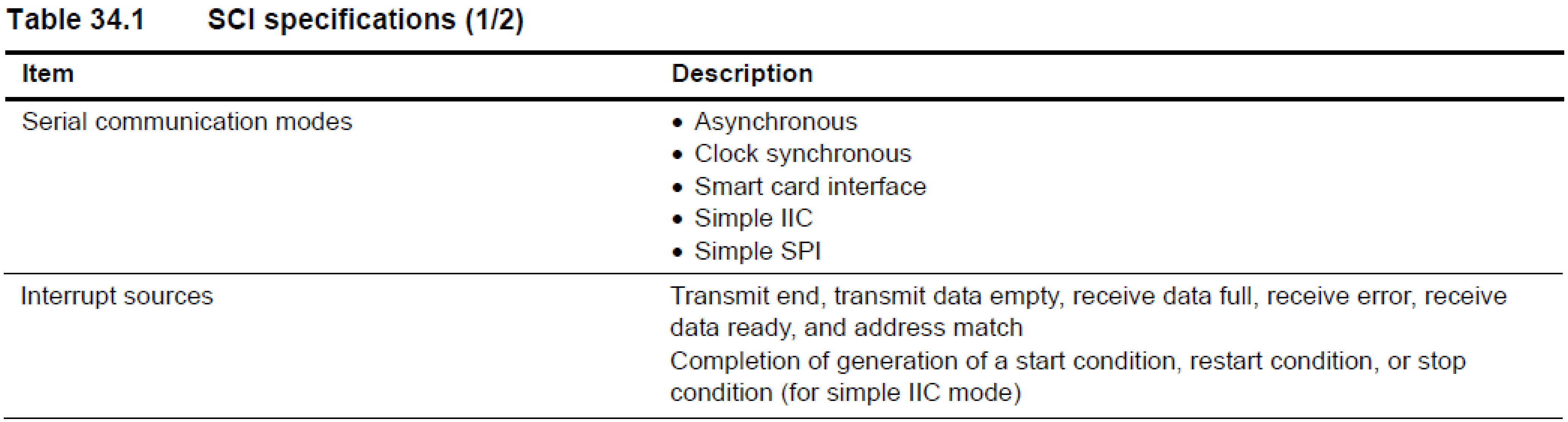


Table 34.1 SCI specifications (1/2)

Item	Description
Serial communication modes	• Asynchronous • Clock synchronous • Smart card interface • Simple IIC • Simple SPI
Interrupt sources	Transmit end, transmit data empty, receive data full, receive error, receive data ready, and address match Completion of generation of a start condition, restart condition, or stop condition (for simple IIC mode)

source: Renesas S7 Series Microcontrollers User's Manual

LAB5 – ACTIVITY 3 – STUDY

These are the services available in the UART Framework of the SSP.

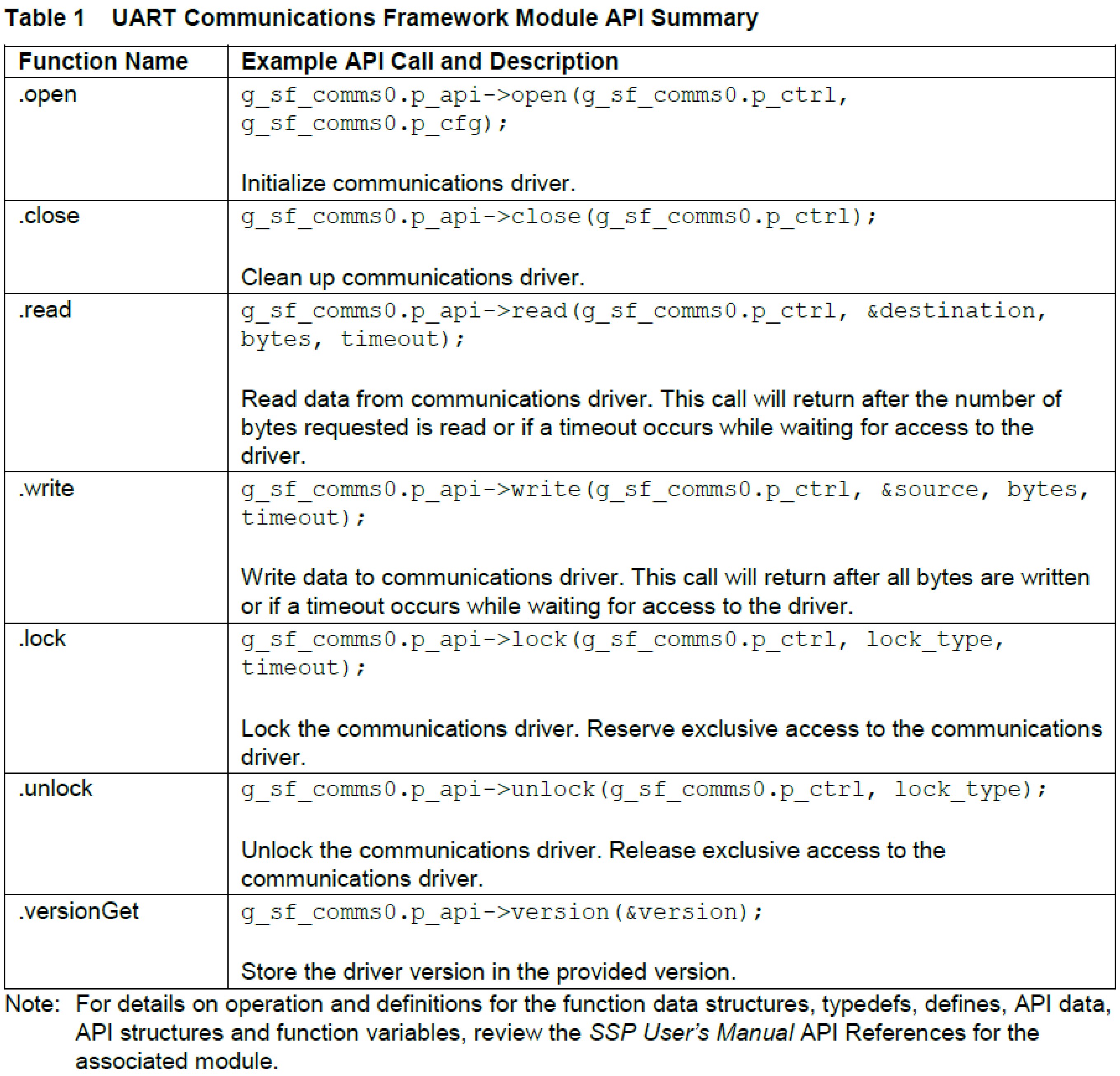


Table 1 UART Communications Framework Module API Summary

Function Name	Example API Call and Description
.open	`g_sf_comms0.p_api->open(g_sf_comms0.p_ctrl, g_sf_comms0.p_cfg);` Initialize communications driver.
.close	`g_sf_comms0.p_api->close(g_sf_comms0.p_ctrl);` Clean up communications driver.
.read	`g_sf_comms0.p_api->read(g_sf_comms0.p_ctrl, &destination, bytes, timeout);` Read data from communications driver. This call will return after the number of bytes requested is read or if a timeout occurs while waiting for access to the driver.
.write	`g_sf_comms0.p_api->write(g_sf_comms0.p_ctrl, &source, bytes, timeout);` Write data to communications driver. This call will return after all bytes are written or if a timeout occurs while waiting for access to the driver.
.lock	`g_sf_comms0.p_api->lock(g_sf_comms0.p_ctrl, lock_type, timeout);` Lock the communications driver. Reserve exclusive access to the communications driver.
.unlock	`g_sf_comms0.p_api->unlock(g_sf_comms0.p_ctrl, lock_type);` Unlock the communications driver. Release exclusive access to the communications driver.
.versionGet	`g_sf_comms0.p_api->version(&version);` Store the driver version in the provided version.

Note: For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the *SSP User's Manual* API References for the associated module.

source: UART Communications Framework Module Guide
r11an0192eu0100-synergy-uart-comms-fw-mod-guide.pdf

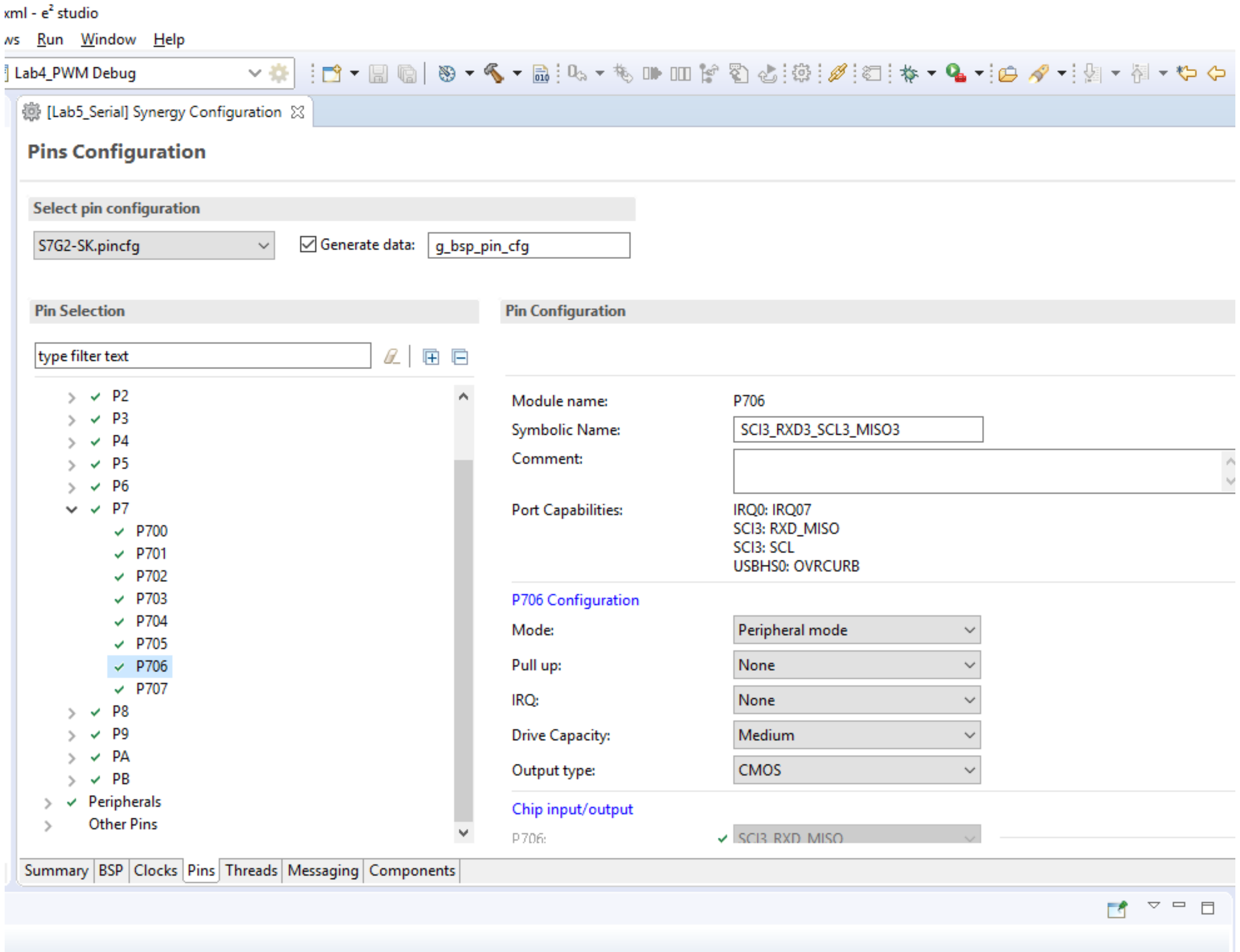
LAB5 – ACTIVITY 3

How to use the Renesas Debug Virtual Console so that printf() messages appear onto a console of e2studio?

- follow instructions from the Renesas Knowledge Base link
- basically:
  - `#include <stdio.h>`
  - `call  initialise_monitor_handles(); during thread inicialization`
  - `confirm that --specs=rdimon.specs is part of the linker flags`
- open a Renesas Debug Virtual Console during debugging and pin it (so that it is always visible).

LAB5 – ACTIVITY 4 – CONFIGURATION

1. Create new Synergy C project, use Blinky with ThreadX as template. Build and run to verify it is operational.
2. In the Synergy Configuration tab, add to the Blinky Thread a Connectivity Framework called sf_uart_comms
3. Configure g_uart0 to Channel 3 and all four interrupt priorities to Priority 3
4. Verify that the Pins for SCI3 are P707 (TXD) and P706 (RXD)



Pins Configuration

Select pin configuration: S7G2-SK.pincfg ☑ Generate data: g_bsp_pin_cfg

type filter text

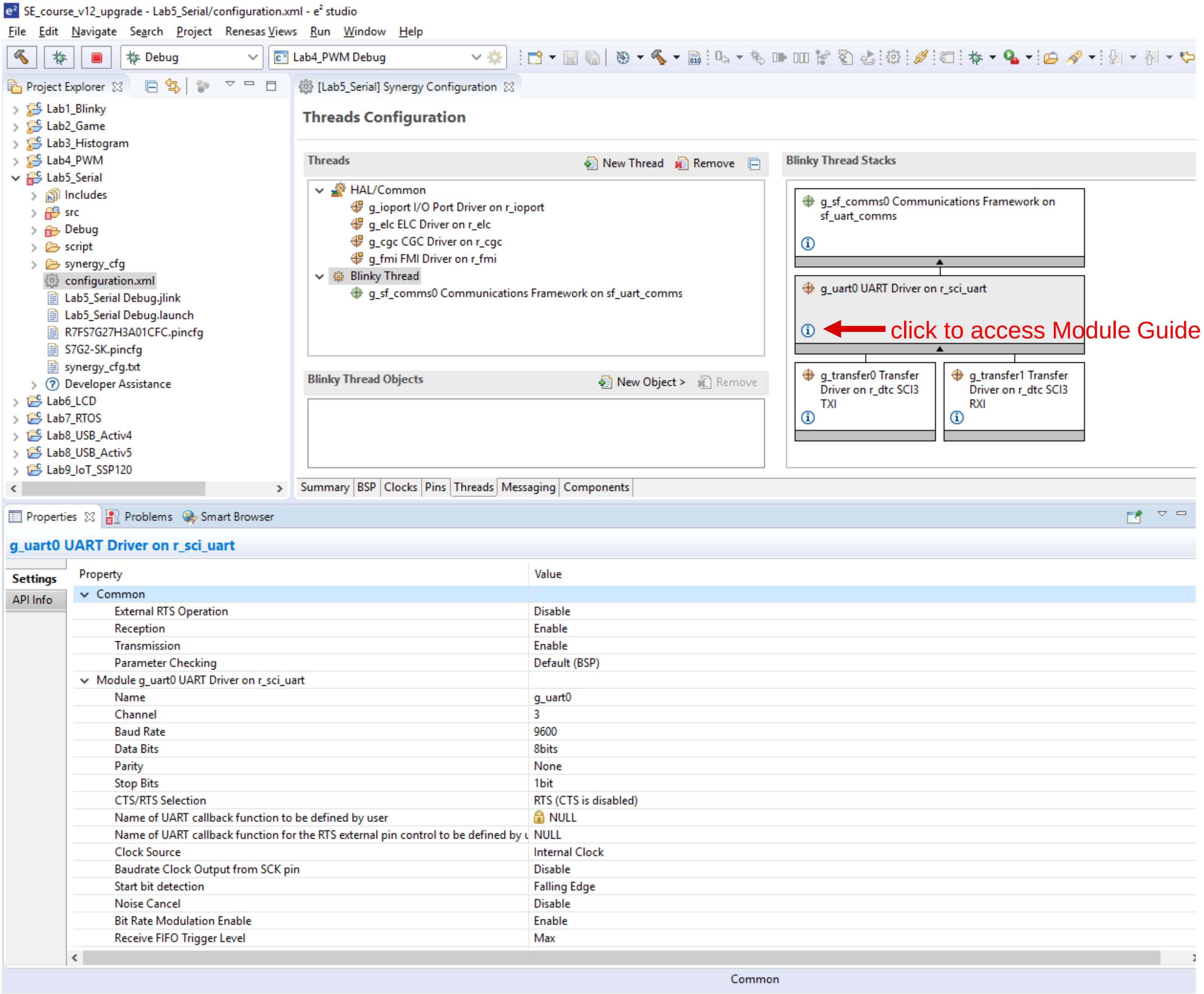
Module name: P706
Symbolic Name: SCI3_RXD3_SCL3_MISO3
Comment:
Port Capabilities: IRQ0: IRQ07
SCI3: RXD_MISO
SCI3: SCL
USBHS0: OVRCURB

P706 Configuration
Mode: Peripheral mode
Pull up: None
IRQ: None
Drive Capacity: Medium
Output type: CMOS

Chip input/output
P706: ✓ SCI3_RXD_MISO

LAB5 – ACTIVITY 4

The Module Guide has important configuration information for each SSP component


SE_course_v12_upgrade - Lab5_Serial/configuration.xml - e² studio
File Edit Navigate Search Project Renesas Views Run Window Help
Debug
Lab4_PWM Debug
Project Explorer
Lab1_Blinky
Lab2_Game
Lab3_Histogram
Lab4_PWM
Lab5_Serial
Includes
src
Debug
script
synergy_cfg
configuration.xml
Lab5_Serial Debug.jlink
Lab5_Serial Debug.launch
R7FS7G27H3A01CFC.pincfg
S7G2-SK.pincfg
synergy_cfg.txt
Developer Assistance
Lab6_LCD
Lab7_RTOS
Lab8_USB_Activ4
Lab8_USB_Activ5
Lab9_IoT_SSP120
[Lab5_Serial] Synergy Configuration
Threads Configuration
Threads
New Thread
Remove
HAL/Common
g_ioport I/O Port Driver on r_ioport
g_elc ELC Driver on r_elc
g_cgc CGC Driver on r_cgc
g_fmi FMI Driver on r_fmi
Blinky Thread
g_sf_comms0 Communications Framework on sf_uart_comms
Blinky Thread Stacks
g_sf_comms0 Communications Framework on sf_uart_comms
g_uart0 UART Driver on r_sci_uart
g_transfer0 Transfer Driver on r_dtc SCI3 TXI
g_transfer1 Transfer Driver on r_dtc SCI3 RXI
Blinky Thread Objects
New Object >
Remove
Summary BSP Clocks Pins Threads Messaging Components
Properties
Problems
Smart Browser
Settings
API Info
Common

click to access Module Guide

g_uart0 UART Driver on r_sci_uart

Property	Value
Common	
External RTS Operation	Disable
Reception	Enable
Transmission	Enable
Parameter Checking	Default (BSP)
Module g_uart0 UART Driver on r_sci_uart	
Name	g_uart0
Channel	3
Baud Rate	9600
Data Bits	8bits
Parity	None
Stop Bits	1bit
CTS/RTS Selection	RTS (CTS is disabled)
Name of UART callback function to be defined by user	NULL
Name of UART callback function for the RTS external pin control to be defined by u	NULL
Clock Source	Internal Clock
Baudrate Clock Output from SCK pin	Disable
Start bit detection	Falling Edge
Noise Cancel	Disable
Bit Rate Modulation Enable	Enable
Receive FIFO Trigger Level	Max

LAB5 – ACTIVITY 4

5. In `blinky_thread_entry.c` make the changes required for the Renesas Debug Virtual Console

6. Modify `blinky_thread_entry` to include the appropriate calls to the API of `g_sf_comms0`:

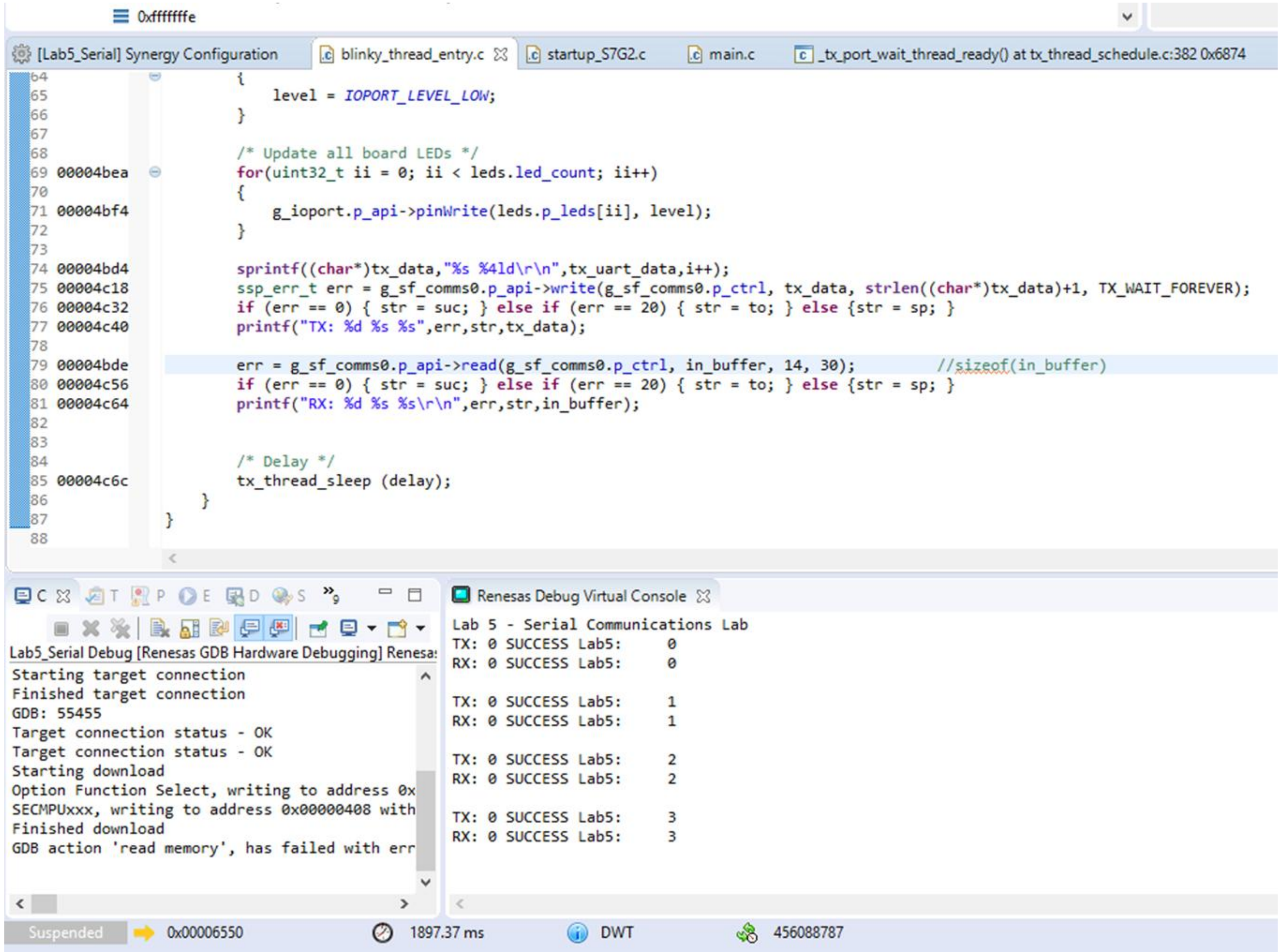
   write to send the string and read to receive it.

7. Use `printf()` to show the transmitted and received strings in the virtual console

LAB5 – ACTIVITIES 5 AND 6

Code running

Virtual Console in operation



0xfffffffe
[Lab5_Serial] Synergy Configuration
blinky_thread_entry.c
startup_S7G2.c
main.c
_tx_port_wait_thread_ready() at tx_thread_schedule.c:382 0x6874
C
T
P
E
D
S
Renesas Debug Virtual Console
Suspended
0x00006550
1897.37 ms
DWT
456088787

```
                          {
                              level = IOPORT_LEVEL_LOW;
                          }

                          /* Update all board LEDs */
00004bea                  for(uint32_t ii = 0; ii < leds.led_count; ii++)
                          {
00004bf4                      g_ioport.p_api->pinWrite(leds.p_leds[ii], level);
                          }

00004bd4                  sprintf((char*)tx_data,"%s %4ld\r\n",tx_uart_data,i++);
00004c18                  ssp_err_t err = g_sf_comms0.p_api->write(g_sf_comms0.p_ctrl, tx_data, strlen((char*)tx_data)+1, TX_WAIT_FOREVER);
00004c32                  if (err == 0) { str = suc; } else if (err == 20) { str = to; } else {str = sp; }
00004c40                  printf("TX: %d %s %s",err,str,tx_data);

00004bde                  err = g_sf_comms0.p_api->read(g_sf_comms0.p_ctrl, in_buffer, 14, 30);         //sizeof(in_buffer)
00004c56                  if (err == 0) { str = suc; } else if (err == 20) { str = to; } else {str = sp; }
00004c64                  printf("RX: %d %s %s\r\n",err,str,in_buffer);


                          /* Delay */
00004c6c                  tx_thread_sleep (delay);
                      }
                  }
```

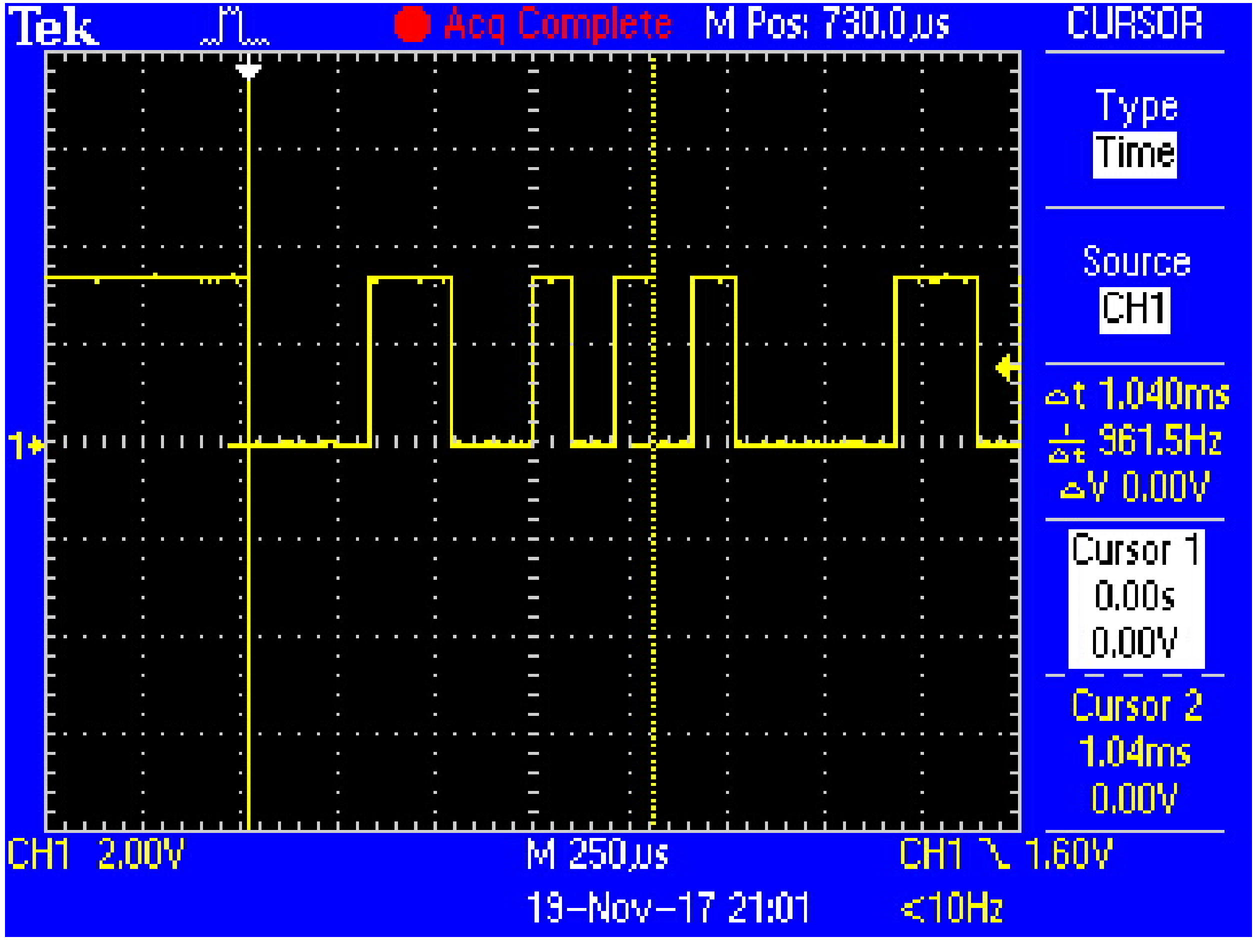
```
Lab5_Serial Debug [Renesas GDB Hardware Debugging] Renesas
Starting target connection
Finished target connection
GDB: 55455
Target connection status - OK
Target connection status - OK
Starting download
Option Function Select, writing to address 0x
SECMPUxxx, writing to address 0x00000408 with
Finished download
GDB action 'read memory', has failed with err
```

```
Lab 5 - Serial Communications Lab
TX: 0 SUCCESS Lab5:     0
RX: 0 SUCCESS Lab5:     0

TX: 0 SUCCESS Lab5:     1
RX: 0 SUCCESS Lab5:     1

TX: 0 SUCCESS Lab5:     2
RX: 0 SUCCESS Lab5:     2

TX: 0 SUCCESS Lab5:     3
RX: 0 SUCCESS Lab5:     3
```

LAB5 – ACTIVITIES 5 AND 6

8. With a scope examine the transmitted signal on J9-pin 3

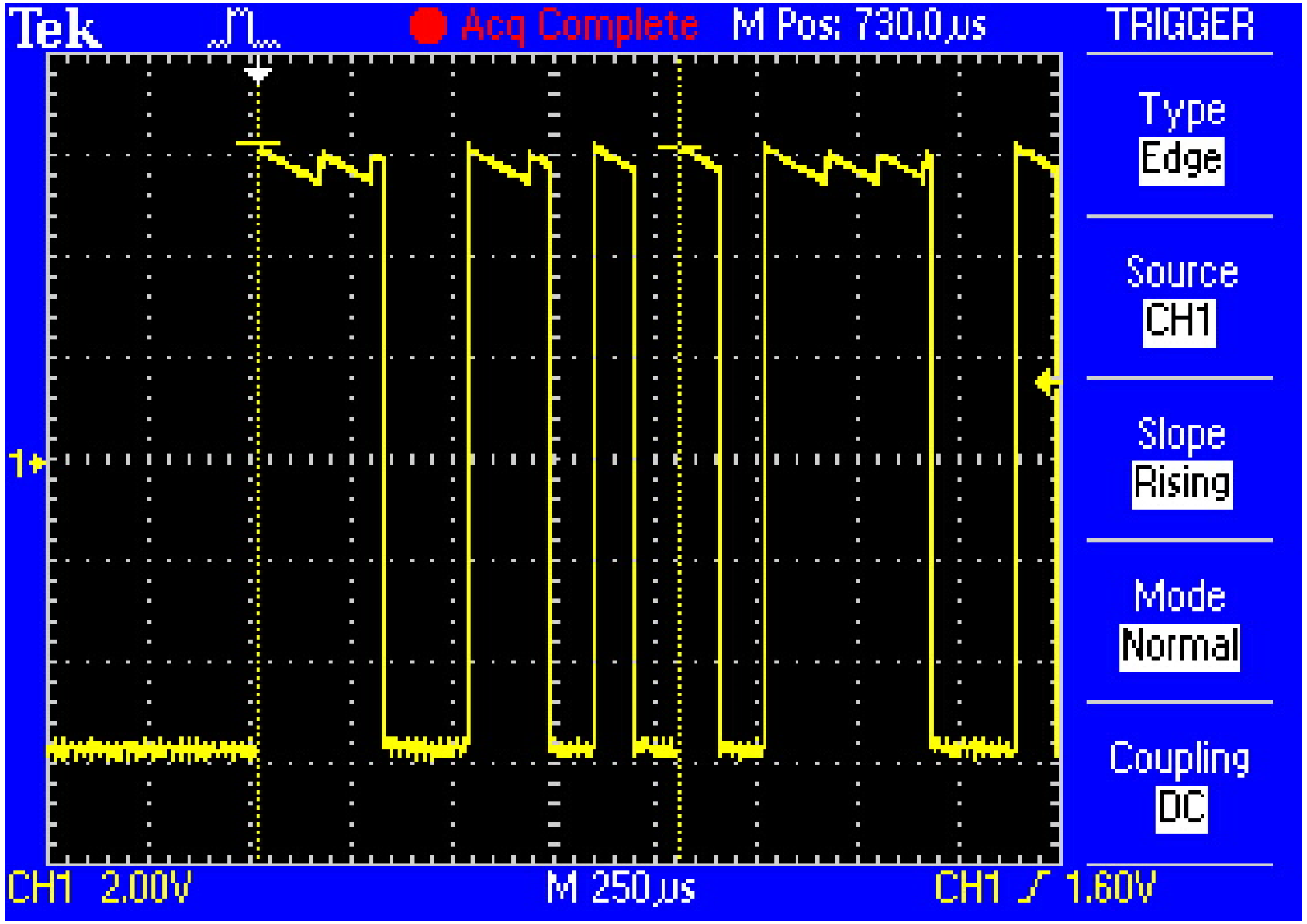
   The first transmitted char is shown between the two cursor lines. Takes 1.04ms to transmit 10 bits: start, 8 data bits and the stop bit.



LAB5 – ACTIVITIES 5 AND 6

9. With a scope examine the transmitted signal on J7-pin 5

On this pin the signal is at RS-232 levels. Logic 1 is represented by -6V and Logic 0 is represented by +6V.



LAB5 – ACTIVITIES 5 AND 6

10. Place a jumper on pins 5 and 6 of J7 so that the transmitted signal is received back.

11. The Virtual Console must present the same strings being transmitted and received

12. Include a counter in the transmitted string to verify that every transmitted message is different from the previous

LAB6 – DISPLAY AND TOUCH

Objectives:

Perform the process of creating a two screen Graphical User Interface making use of the graphical LCD and Touch Screen available on the SK-S7G2 board.

LAB6 – DISPLAY AND TOUCH

Activities:

1. Study the physical connection of the LCD and Touch Screen to the MCU
2. Study the SSP API for graphical interfaces and input via touch screen
3. Perform the experiment described in the Renesas Application Note: GUIX “Hello World” for the SK-S7G2.

ADDITIONAL READING FOR LAB6

- Renesas Synergy Starter Kit SK-S7G2 User's Manual (r12um0004eu0100) (https://www.renesas.com/en-eu/doc/products/renesas-synergy/doc/r12um0004eu0100_synergy_sk_s7g2.pdf)
- Renesas S7 Series Microcontrollers User's Manual
- Renesas Synergy Software Package v1.7.5 Manual (link) available in the Synergy Gallery (https://synergygallery.renesas.com/media/products/1/384/en-US/r11um0140eu0106-synergy-ssp-v175.pdf)
- Renesas Application Note R12AN0021EU0118 (link) (https://www.renesas.com/us/en/doc/products/renesas-synergy/apn/r12an0021eu0118-synergy-sk-s7g2-pk-s5d9-guix-hello-world.pdf)
- Sample Software for GUIX "Hello World" from:
  https://www.renesas.com/us/en/doc/products/renesas-synergy/apn/r12an0021eu0119-synergy-sk-s7g2-pk-s5d9-guix-hello-world.pdf
  https://www.renesas.com/us/en/software/D6003641.html

LAB6 – ACTIVITY 1

S7G2 microcontroller’s block diagram with indication of blocks of interest.

Since the user’s manual of the S7G2 has more than 2000 pages, it is important to keep focus on what is relevant to this project.

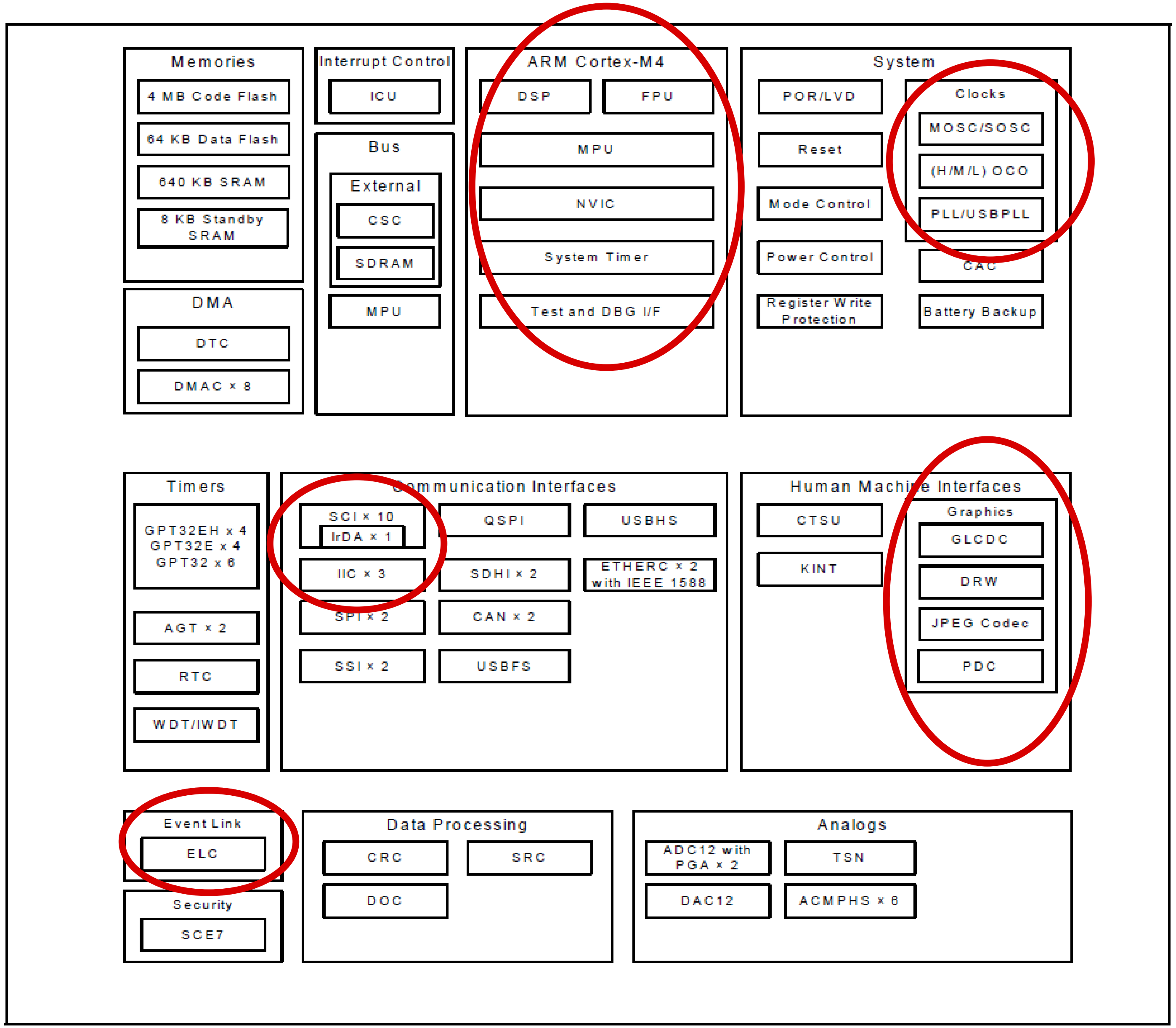


Figure 1.1 **Block diagram**

source: Renesas S7 Series Microcontrollers User’s Manual

LAB6 – ACTIVITY 1

Relevant blocks of the SK-S7G2 board are marked in red

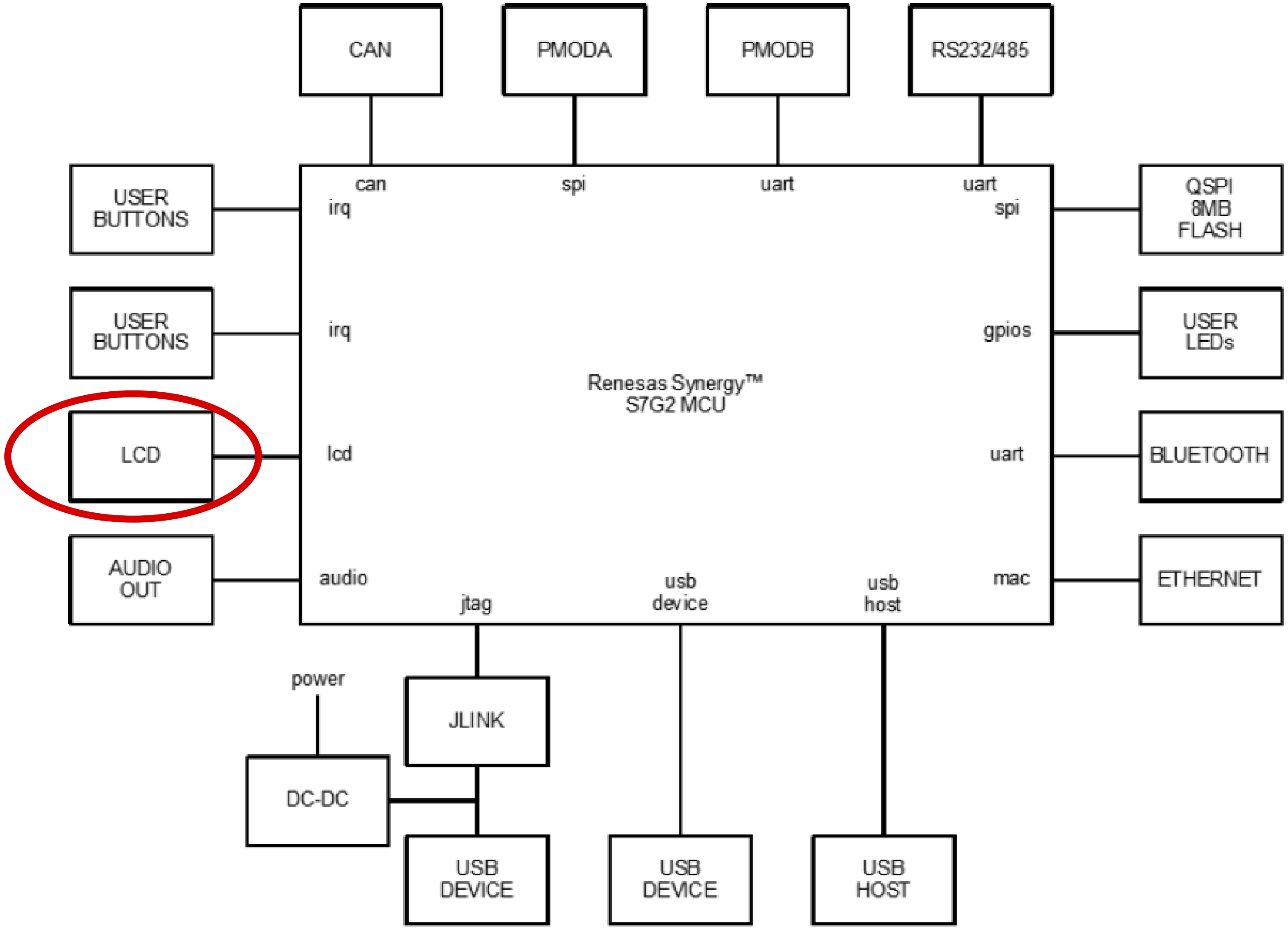


Figure 2: SK-S7G2 block diagram

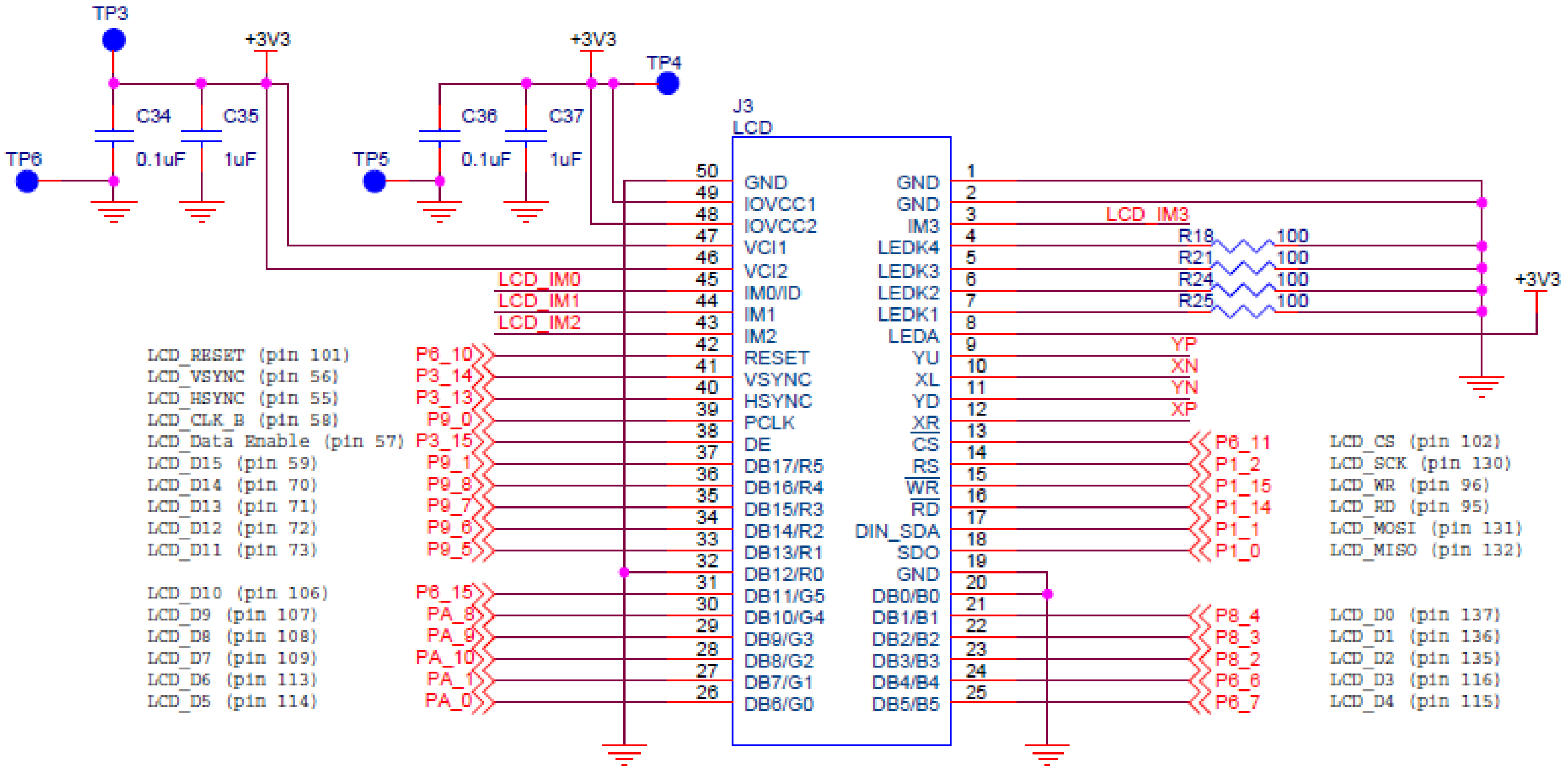
source: Starter Kit SK-S7G2 User's Manual

LAB6 – ACTIVITY 1

The LCD is connected

directly to the

LCD Interface of the MCU

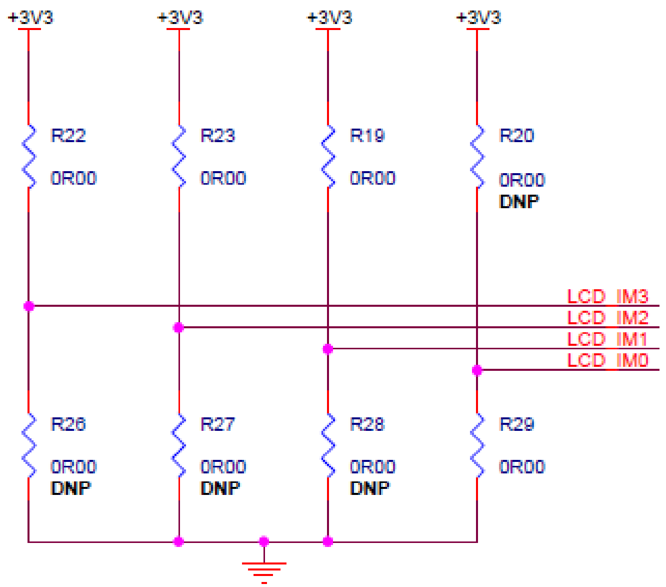


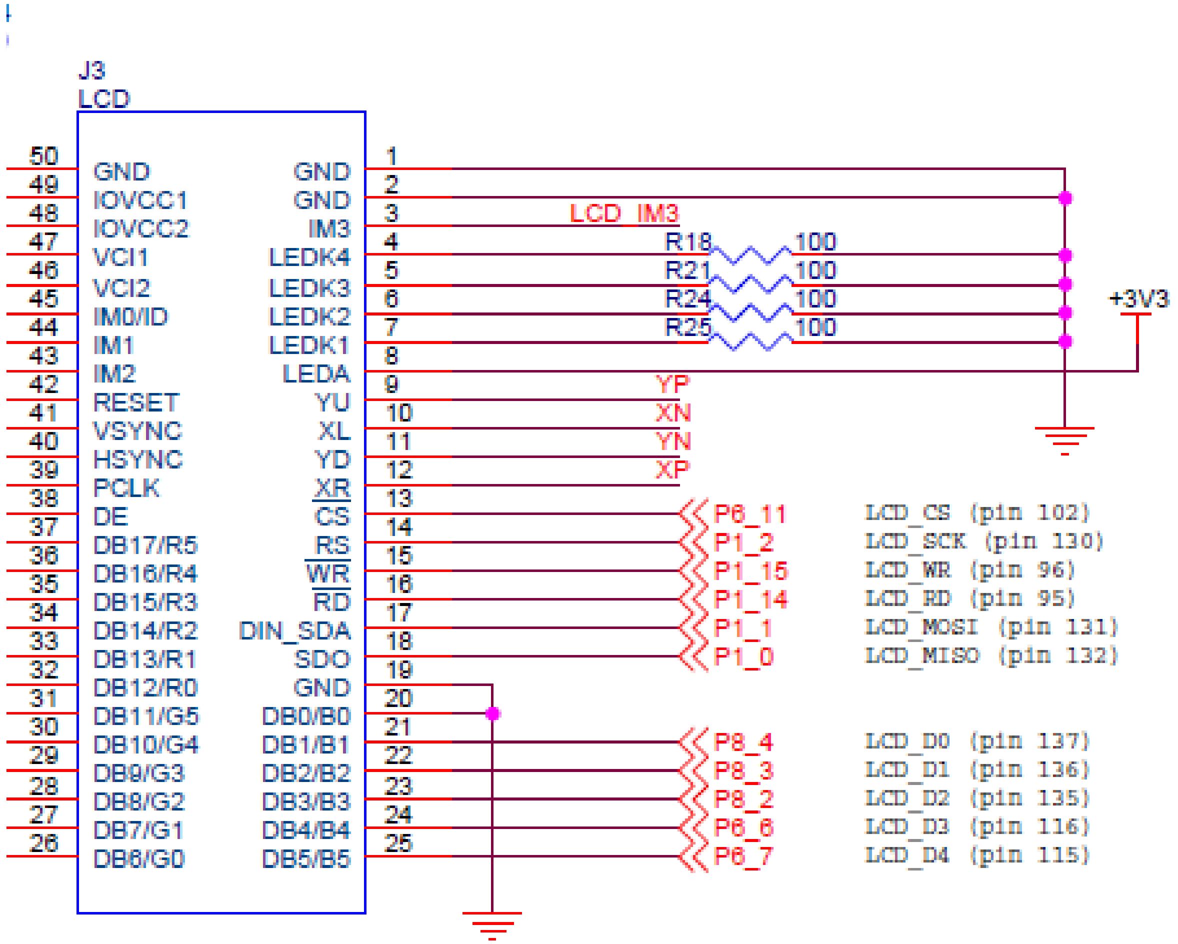
source: Starter Kit SK-S7G2 User's Manual

LAB6 – ACTIVITY 1

The LCD uses an Ilitek driver connected to an SCI (Serial Communications Interface) of the MCU. The serial communication is configured to 4-wire 8 bits.

NOTE: {IM0,IM1,IM2,IM3}={0,1,1,1},set the LCD mode to
4-wire 8-bit data serial interface.
Please refer to the ILI9341V datasheet for more information.

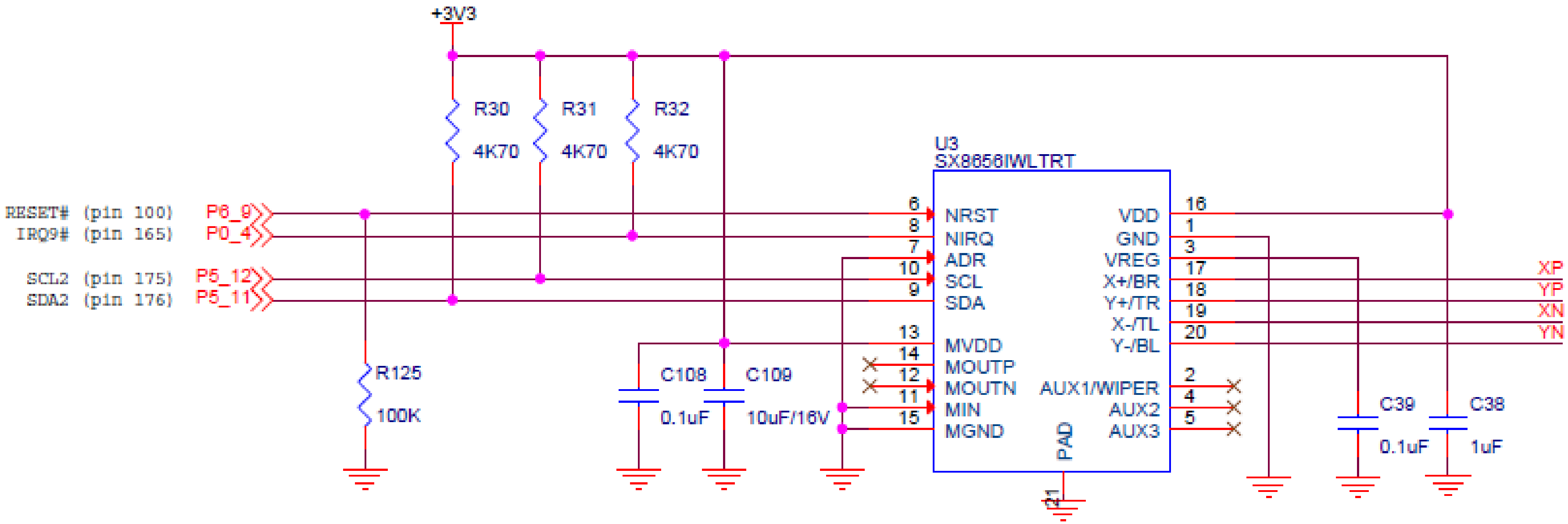




source: Starter Kit SK-S7G2 User's Manual

LAB6 – ACTIVITY 1

The Touch Screen is connected to the MCU via an I2C interface.



source: Starter Kit SK-S7G2 User’s Manual

LAB6 – ACTIVITY 1

The GLCD (Graphics LCD Controller) of the MCU is configurable to many different LCDs. Its physical interface may be up to 24 data bits plus several synchronization and clock signals.

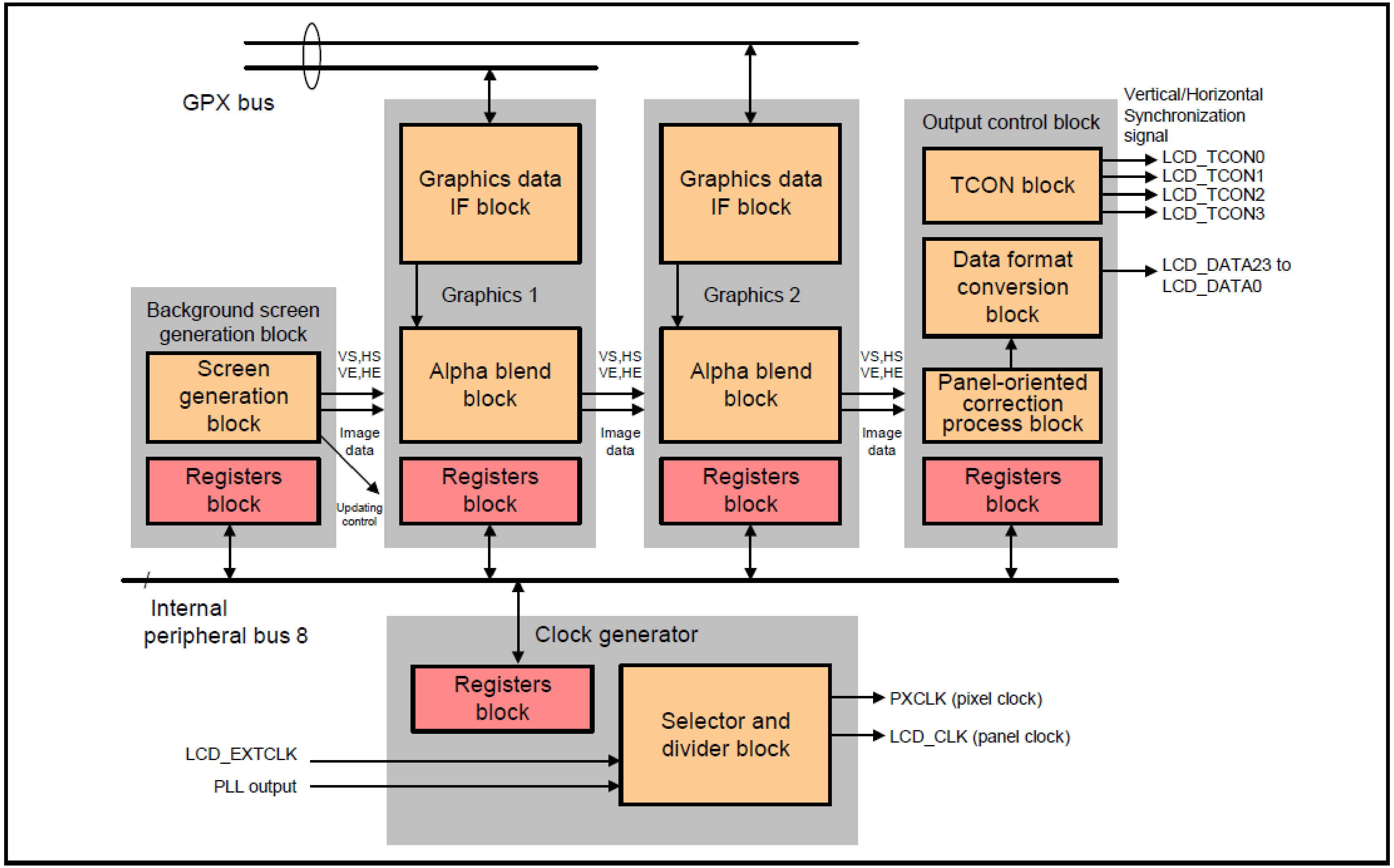


Figure 57.1 GLCDC block diagram

source: Renesas S7 Series Microcontrollers User's Manual

LAB6 – ACTIVITY 2

The Renesas Synergy Software Package v1.7.5 Manual (link), available in the Synergy Gallery, describes the API for:

- The Graphics Display, on section 4.2.14
- The Touch Panel, on section 4.1.19

LAB6 – ACTIVITY 3

- Download the zip file available at:

  https://www.renesas.com/us/en/software/D6004128.html

  this zip includes the pdf with the application note R12AN0021EU0118 and the source files for the corresponding lab experiment.

  The pdf is also available at: link

  The sample code may be also obtained by accessing page:

  https://www.renesas.com/us/en/products/synergy/hardware/microcontrollers.html#productinfo

  and searching for r12an0021

- The aim here is to go through the 48 pages describing in detail how to build a very simple Graphical User Interface.
- This lab requires the GUIX Studio, a tool that was installed as part of Lab1.

LAB7 – RTOS

Objectives:

Create a simple multithreaded application that uses mutex and message queue. Make use of RTOS-aware debugging.

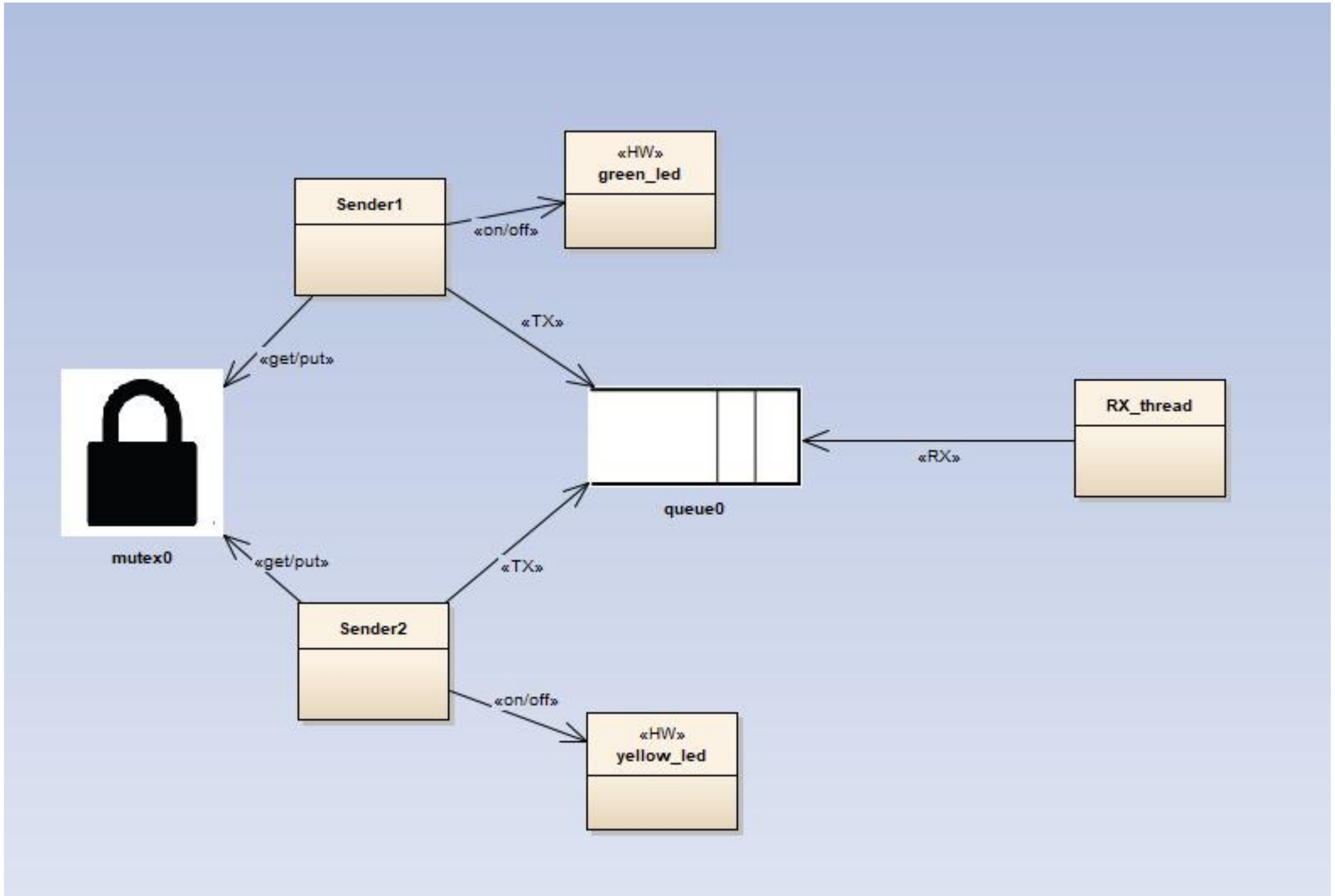
LAB7 – RTOS

The diagram (next slide) presents the desired application.

- Thread Sender1 - sends messages to RX_Thread via queue0, workload is simulated by 40 ticks delay and green led on.
- Thread Sender2 - sends messages to RX_Thread via queue0, workload is simulated by 60 ticks delay and yellow led on.
- Thread RX_Thread - receives messages from both sender threads
- mutex0 - prevents simultaneous execution of Sender 1 and Sender 2 (while LEDs are on)
- red led - indicates one of the sender threads is blocked on the mutex

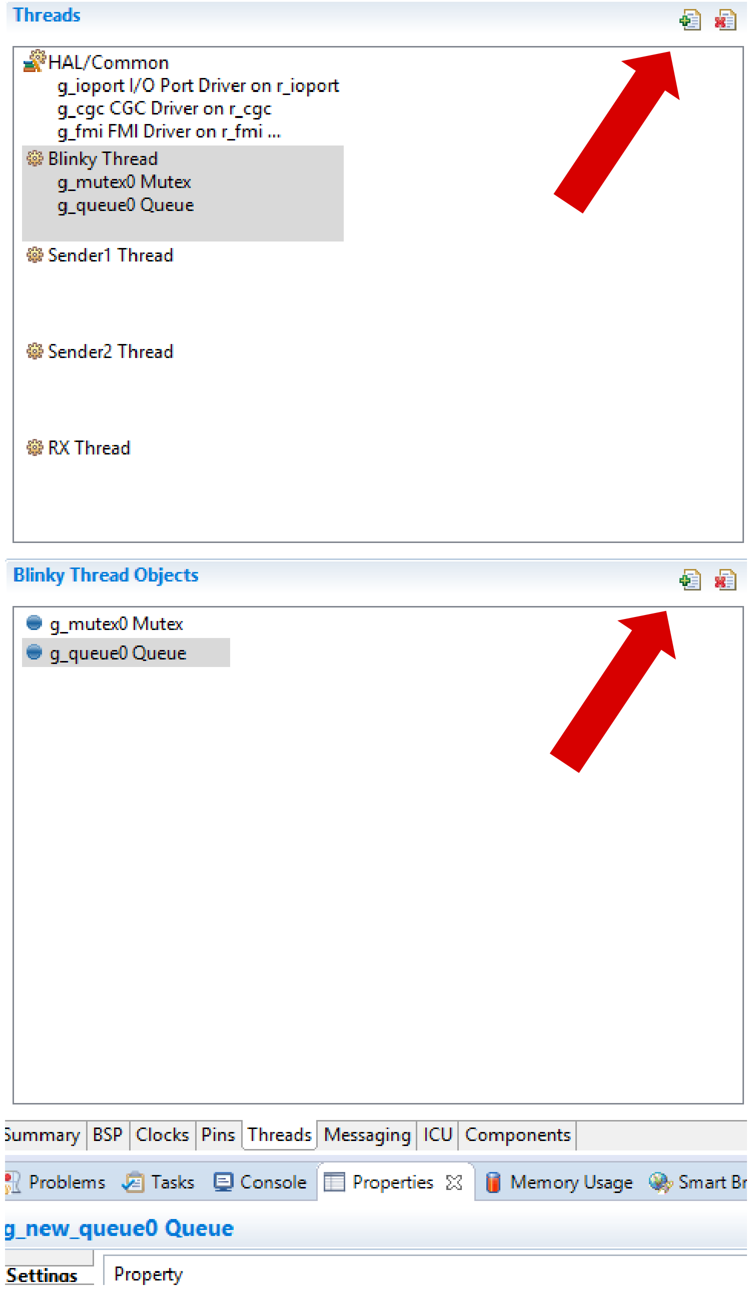
LAB7 – RTOS

Diagram presenting the architecture of Lab 7 (UML class diagram notation).



LAB7 – RTOS

Adding additional threads and ThreadX object (Mutex and Queue) using the Synergy Configuration tab.



Threads

HAL/Common
- g_ioport I/O Port Driver on r_ioport
- g_cgc CGC Driver on r_cgc
- g_fmi FMI Driver on r_fmi ...

Blinky Thread
- g_mutex0 Mutex
- g_queue0 Queue

Sender1 Thread

Sender2 Thread

RX Thread

Blinky Thread Objects
- g_mutex0 Mutex
- g_queue0 Queue

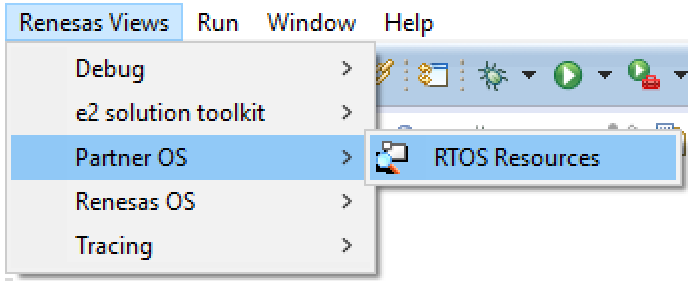
Summary | BSP | Clocks | Pins | Threads | Messaging | ICU | Components
Problems | Tasks | Console | Properties | Memory Usage | Smart Br

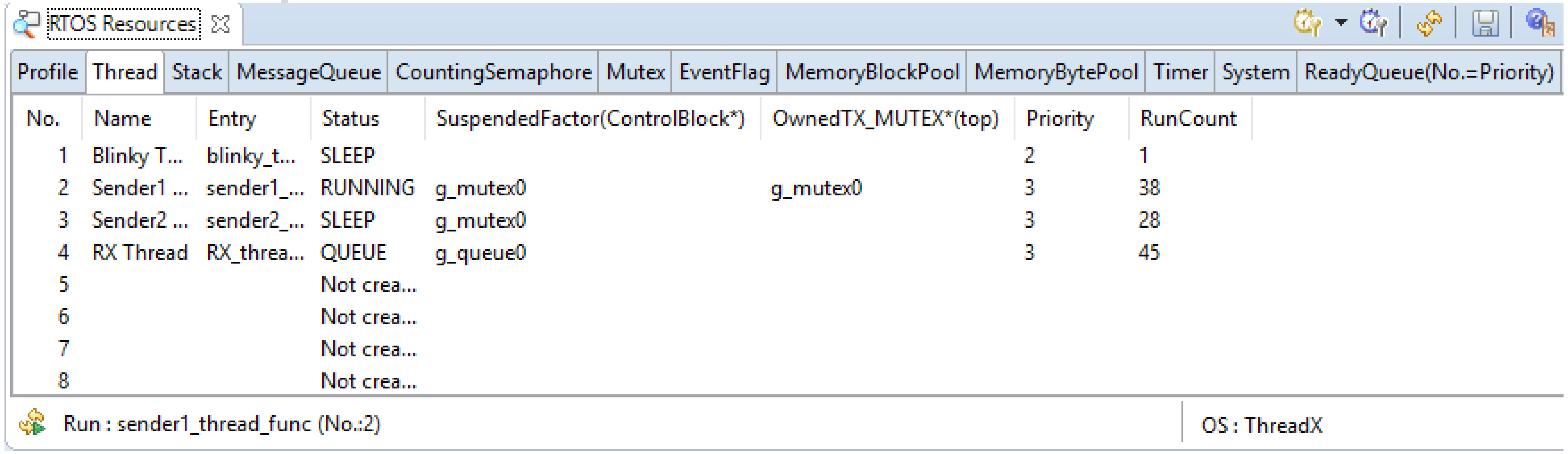
g_new_queue0 Queue

Settings | Property

LAB7 – RTOS

The Partner OS | RTOS Resources view allows the visualization of the state of the threads and of the objects (Queue and Mutex).

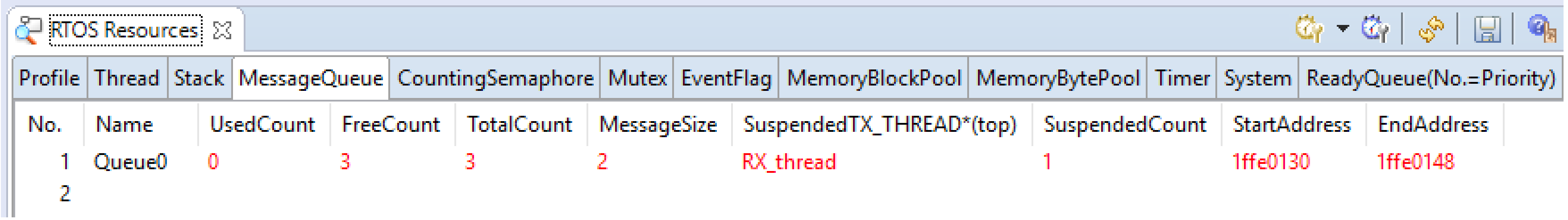




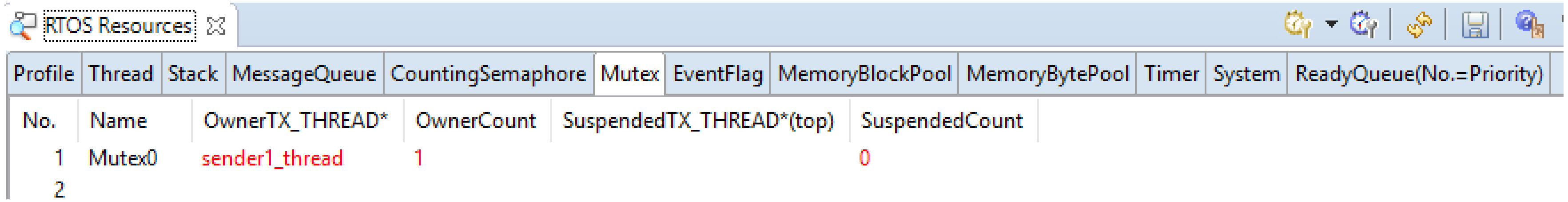
No.	Name	Entry	Status	SuspendedFactor(ControlBlock*)	OwnedTX_MUTEX*(top)	Priority	RunCount
1	Blinky T...	blinky_t...	SLEEP			2	1
2	Sender1 ...	sender1_...	RUNNING	g_mutex0	g_mutex0	3	38
3	Sender2 ...	sender2_...	SLEEP	g_mutex0		3	28
4	RX Thread	RX_threa...	QUEUE	g_queue0		3	45
5			Not crea...				
6			Not crea...				
7			Not crea...				
8			Not crea...				

Run : sender1_thread_func (No.:2)

OS : ThreadX



No.	Name	UsedCount	FreeCount	TotalCount	MessageSize	SuspendedTX_THREAD*(top)	SuspendedCount	StartAddress	EndAddress
1	Queue0	0	3	3	2	RX_thread	1	1ffe0130	1ffe0148
2									



No.	Name	OwnerTX_THREAD*	OwnerCount	SuspendedTX_THREAD*(top)	SuspendedCount
1	Mutex0	sender1_thread	1		0
2					

LAB7 – RTOS

Observe that due to the Mutex, it never happens that the green LED and the yellow LED are on at the same time.

If, however, you disable (comment out) the calls to the Mutex, then the behavior is quite different.

LAB8 – USB DEVICE

Objectives:

In Lab 8, the student will be presented to an e2 Studio project which uses the USBX and related components of the SSP to implement an USB Device.

The SK-S7G2 board is configured by this application to respond to USB requests as a Communication Device. That means, the OS installed in a PC, to which the SK-S7G2 board is connected, will handle the device as a virtual COM port, allowing a terminal application running on the PC to send and receive bytes to/from the device.

To do this lab you will need the following materials:

- SK-S7G2 kit with the USB Micro-B debug cable.
- An extra USB Micro-B cable to connect the kit to a PC.

LAB8 – USB DEVICE

Activities:

1. Tool setup and new project creation.
2. Setup of Synergy Configuration to use the Communications Framework on USB.
3. Low-level function prototyping.
4. Application requirements, coding and testing.
5. Challenge: multi-threaded version.

LAB8 – ACTIVITY 1

Activity 1 – Tool setup and new project creation

1. Ensure that the e2 Studio and the SSP are properly installed. If necessary, (re)run the activities 1 to 5 of Lab 1 to do so.

2. Create a new Renesas Synergy Project for the SK-S7G2 board and name it "Lab8_USB“.

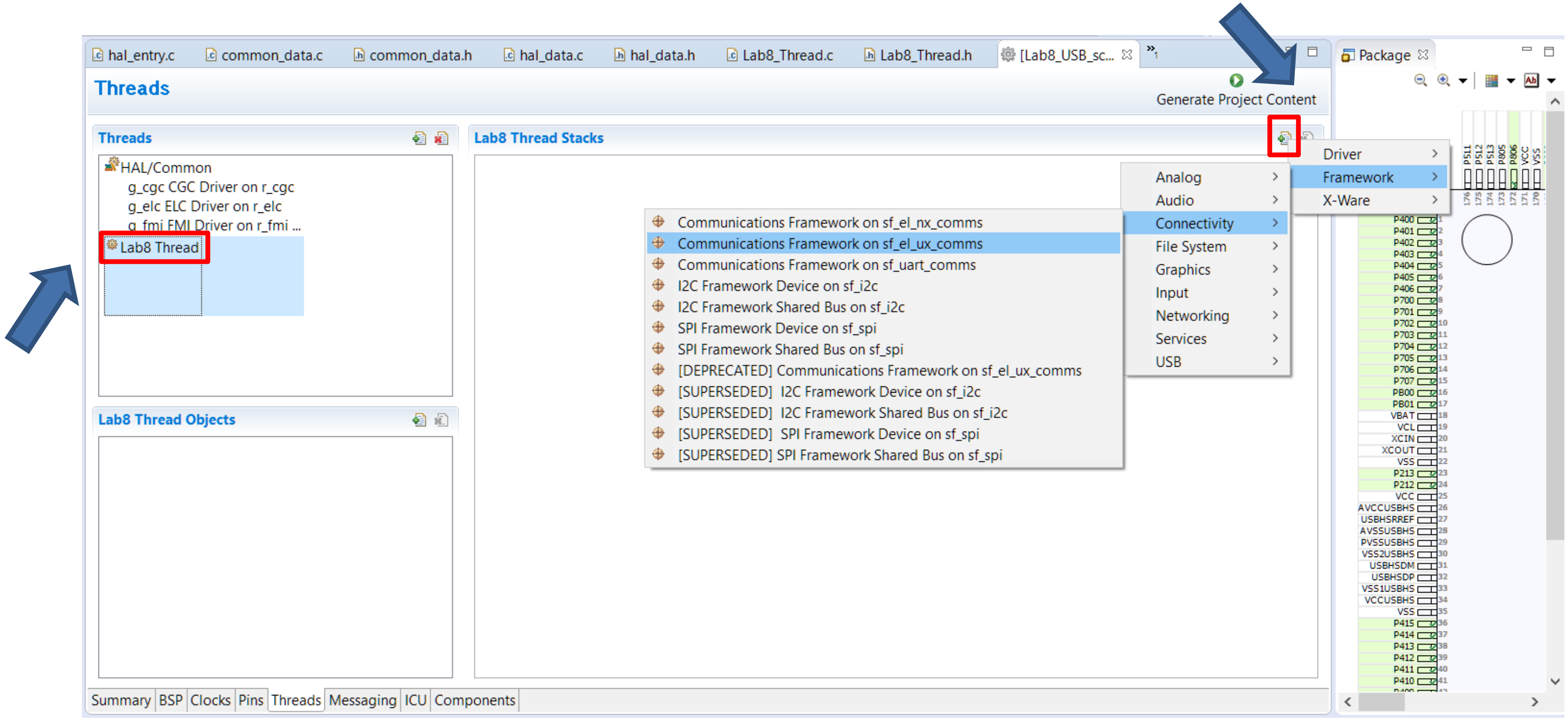
   - This project can be based on the “Blinky” template project

LAB8 – ACTIVITY 2

Activity 2 – Setup of Synergy Configuration to use the Communications Framework on USB

1. Create a new Thread using the Synergy Configuration and add the Communications Framework on *sf_el_ux_comms* (New Stack → Framework → Connectivity).

2. Rename the thread to "Lab8_Thread".



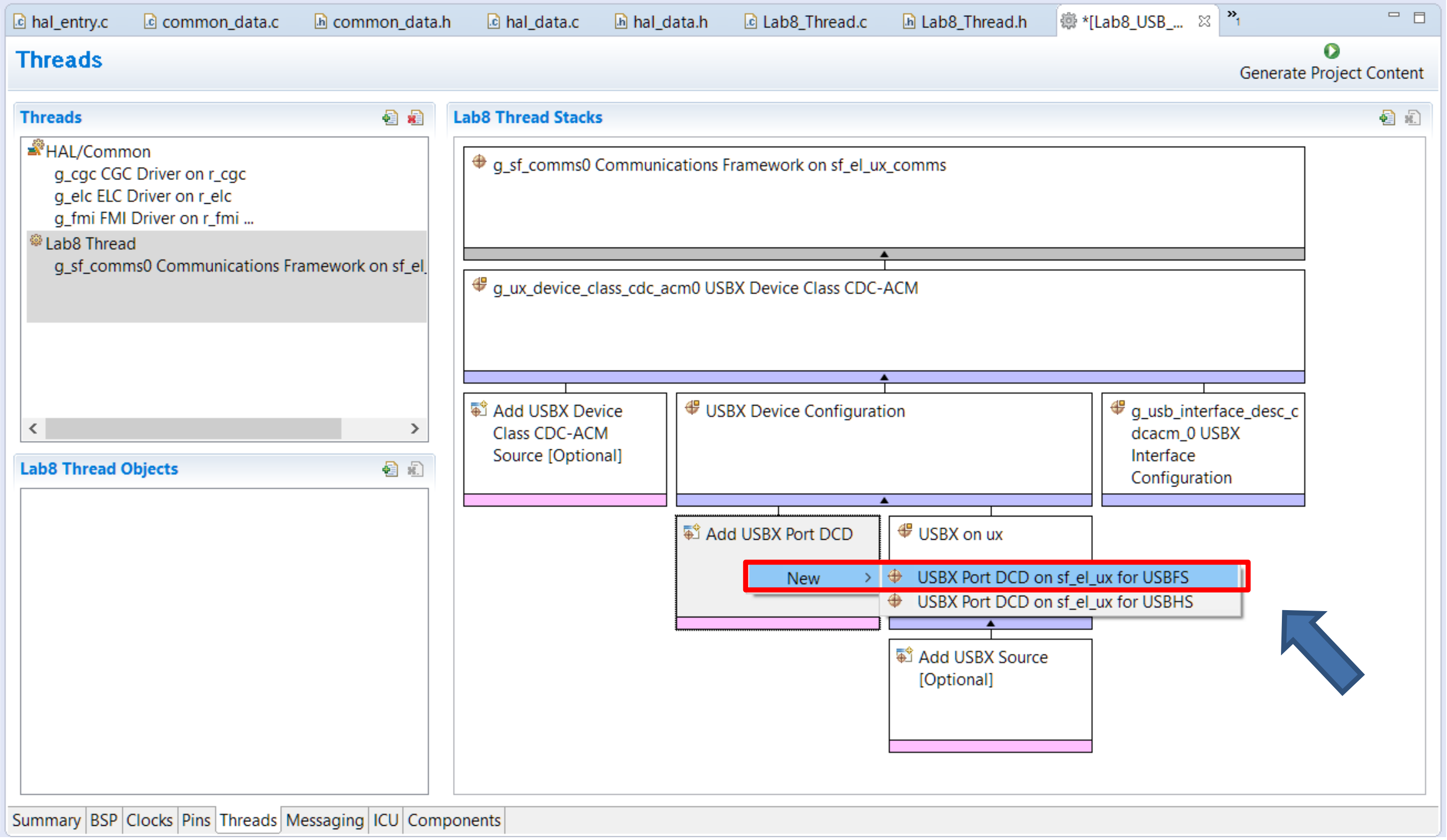
hal_entry.c
common_data.c
common_data.h
hal_data.c
hal_data.h
Lab8_Thread.c
Lab8_Thread.h
[Lab8_USB_sc...
Package
Threads
Generate Project Content
Threads
HAL/Common
g_cgc CGC Driver on r_cgc
g_elc ELC Driver on r_elc
g_fmi FMI Driver on r_fmi ...
Lab8 Thread
Lab8 Thread Stacks
Lab8 Thread Objects
Driver
Framework
X-Ware
Analog
Audio
Connectivity
File System
Graphics
Input
Networking
Services
USB
Communications Framework on sf_el_nx_comms
Communications Framework on sf_el_ux_comms
Communications Framework on sf_uart_comms
I2C Framework Device on sf_i2c
I2C Framework Shared Bus on sf_i2c
SPI Framework Device on sf_spi
SPI Framework Shared Bus on sf_spi
[DEPRECATED] Communications Framework on sf_el_ux_comms
[SUPERSEDED] I2C Framework Device on sf_i2c
[SUPERSEDED] I2C Framework Shared Bus on sf_i2c
[SUPERSEDED] SPI Framework Device on sf_spi
[SUPERSEDED] SPI Framework Shared Bus on sf_spi
Summary
BSP
Clocks
Pins
Threads
Messaging
ICU
Components

Source: Authors

LAB8 – ACTIVITY 2

3. Click on "Add USBX Port DCD" and select the "USBX Port DCD on sf_el_ux for USBFS". This will select the Full-Speed Device Controller Driver for the R7FS7G27H3A01CFC Renesas ARM Cortex-M4 MCU Device Controller.

   See Section 10, “USB Logical View”, to review the exact role of each of the component blocks included by Synergy Configuration.

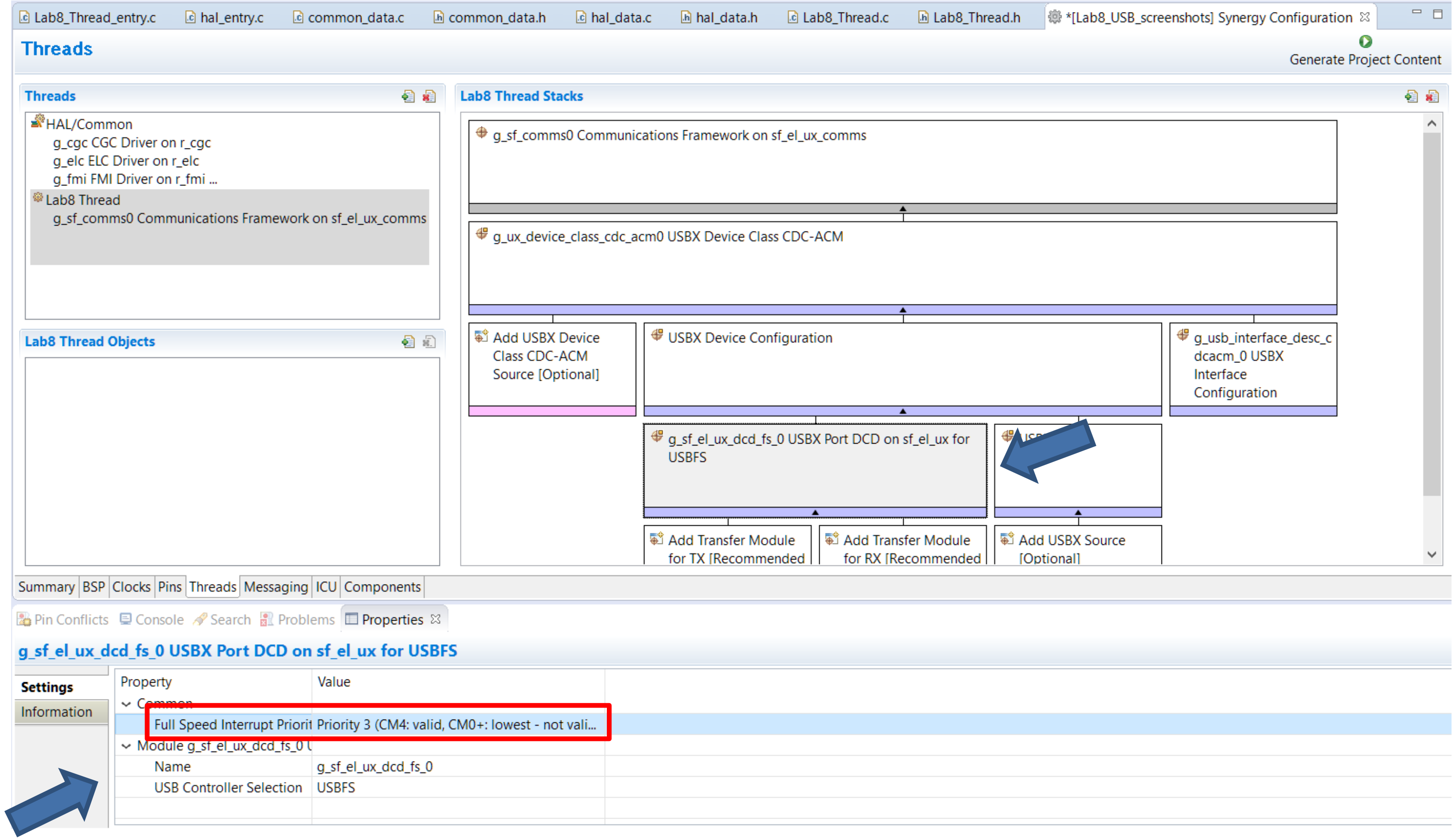


hal_entry.c
common_data.c
common_data.h
hal_data.c
hal_data.h
Lab8_Thread.c
Lab8_Thread.h
*[Lab8_USB_...
Threads
Generate Project Content
Threads
HAL/Common
g_cgc CGC Driver on r_cgc
g_elc ELC Driver on r_elc
g_fmi FMI Driver on r_fmi ...
Lab8 Thread
g_sf_comms0 Communications Framework on sf_el_
Lab8 Thread Objects
Lab8 Thread Stacks
g_sf_comms0 Communications Framework on sf_el_ux_comms
g_ux_device_class_cdc_acm0 USBX Device Class CDC-ACM
Add USBX Device Class CDC-ACM Source [Optional]
USBX Device Configuration
g_usb_interface_desc_cdcacm_0 USBX Interface Configuration
Add USBX Port DCD
USBX on ux
New
USBX Port DCD on sf_el_ux for USBFS
USBX Port DCD on sf_el_ux for USBHS
Add USBX Source [Optional]
Summary
BSP
Clocks
Pins
Threads
Messaging
ICU
Components

Source: Authors

LAB8 – ACTIVITY 2

4. Set the USBX Port DCD Property “Full Speed Interrupt Priority" to a priority level compatible with Cortex-M4 (e. g. Priority 3).



Lab8_Thread_entry.c
hal_entry.c
common_data.c
common_data.h
hal_data.c
hal_data.h
Lab8_Thread.c
Lab8_Thread.h
*[Lab8_USB_screenshots] Synergy Configuration
Threads
Generate Project Content
Threads
HAL/Common
g_cgc CGC Driver on r_cgc
g_elc ELC Driver on r_elc
g_fmi FMI Driver on r_fmi ...
Lab8 Thread
g_sf_comms0 Communications Framework on sf_el_ux_comms
Lab8 Thread Objects
Lab8 Thread Stacks
g_sf_comms0 Communications Framework on sf_el_ux_comms
g_ux_device_class_cdc_acm0 USBX Device Class CDC-ACM
Add USBX Device Class CDC-ACM Source [Optional]
USBX Device Configuration
g_usb_interface_desc_c dcacm_0 USBX Interface Configuration
g_sf_el_ux_dcd_fs_0 USBX Port DCD on sf_el_ux for USBFS
Add Transfer Module for TX [Recommended
Add Transfer Module for RX [Recommended
Add USBX Source [Optional]
Summary
BSP
Clocks
Pins
Threads
Messaging
ICU
Components
Pin Conflicts
Console
Search
Problems
Properties
Settings
Information

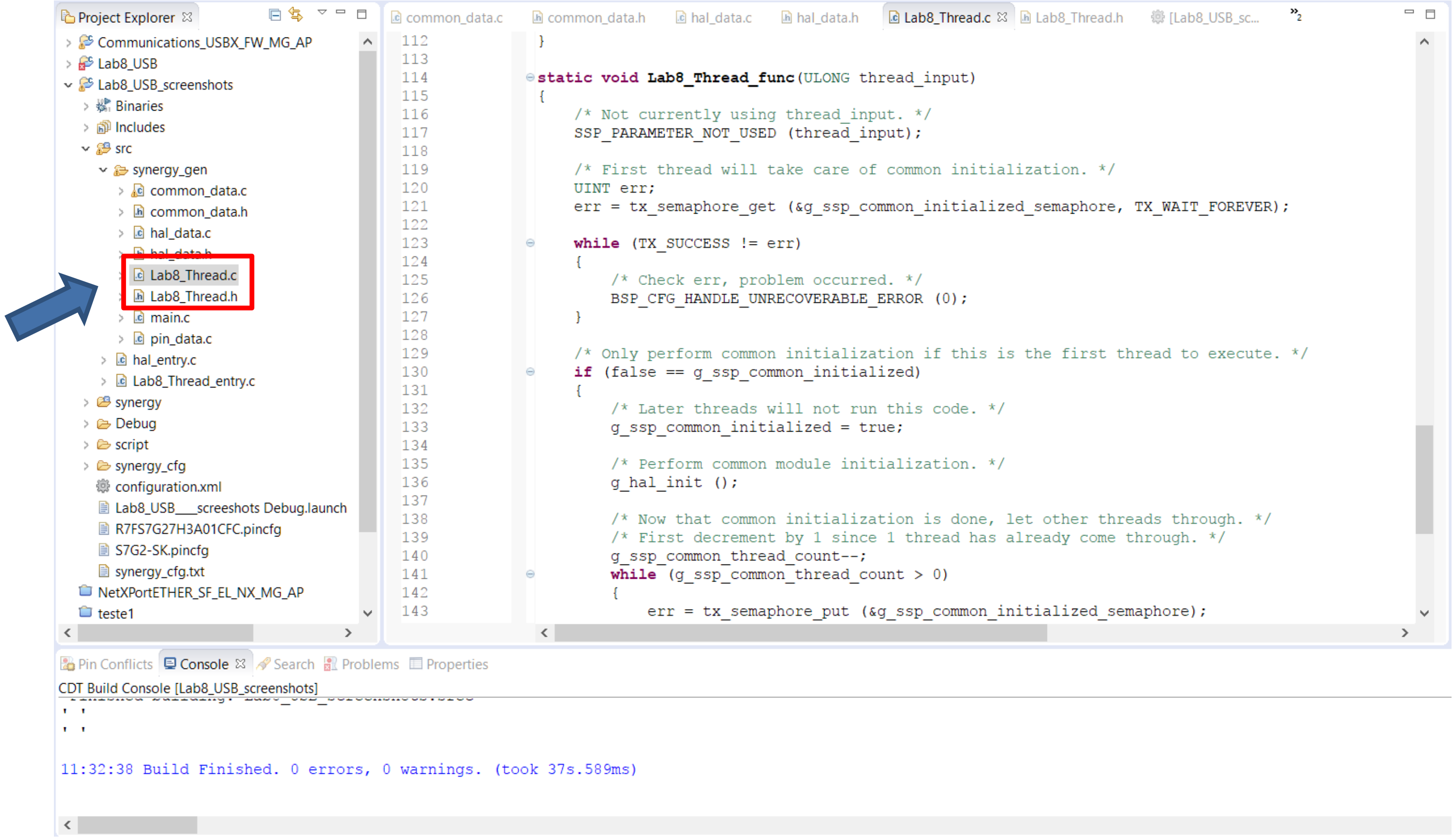
g_sf_el_ux_dcd_fs_0 USBX Port DCD on sf_el_ux for USBFS

Property	Value
Common	
Full Speed Interrupt Priorit	Priority 3 (CM4: valid, CM0+: lowest - not vali...
Module g_sf_el_ux_dcd_fs_0 U	
Name	g_sf_el_ux_dcd_fs_0
USB Controller Selection	USBFS

Source: Authors

LAB8 – ACTIVITY 2

5. Build the project. Verify that the files Lab8_Thread.c and Lab8_Thread.h were created into the src/synergy_gen folder. These files contain the initialization for the Communications Framework and the other common HAL modules.



```
}

static void Lab8_Thread_func(ULONG thread_input)
{
    /* Not currently using thread_input. */
    SSP_PARAMETER_NOT_USED (thread_input);

    /* First thread will take care of common initialization. */
    UINT err;
    err = tx_semaphore_get (&g_ssp_common_initialized_semaphore, TX_WAIT_FOREVER);

    while (TX_SUCCESS != err)
    {
        /* Check err, problem occurred. */
        BSP_CFG_HANDLE_UNRECOVERABLE_ERROR (0);
    }

    /* Only perform common initialization if this is the first thread to execute. */
    if (false == g_ssp_common_initialized)
    {
        /* Later threads will not run this code. */
        g_ssp_common_initialized = true;

        /* Perform common module initialization. */
        g_hal_init ();

        /* Now that common initialization is done, let other threads through. */
        /* First decrement by 1 since 1 thread has already come through. */
        g_ssp_common_thread_count--;
        while (g_ssp_common_thread_count > 0)
        {
            err = tx_semaphore_put (&g_ssp_common_initialized_semaphore);
```

CDT Build Console [Lab8_USB_screenshots]

```
' '
' '

11:32:38 Build Finished. 0 errors, 0 warnings. (took 37s.589ms)
```

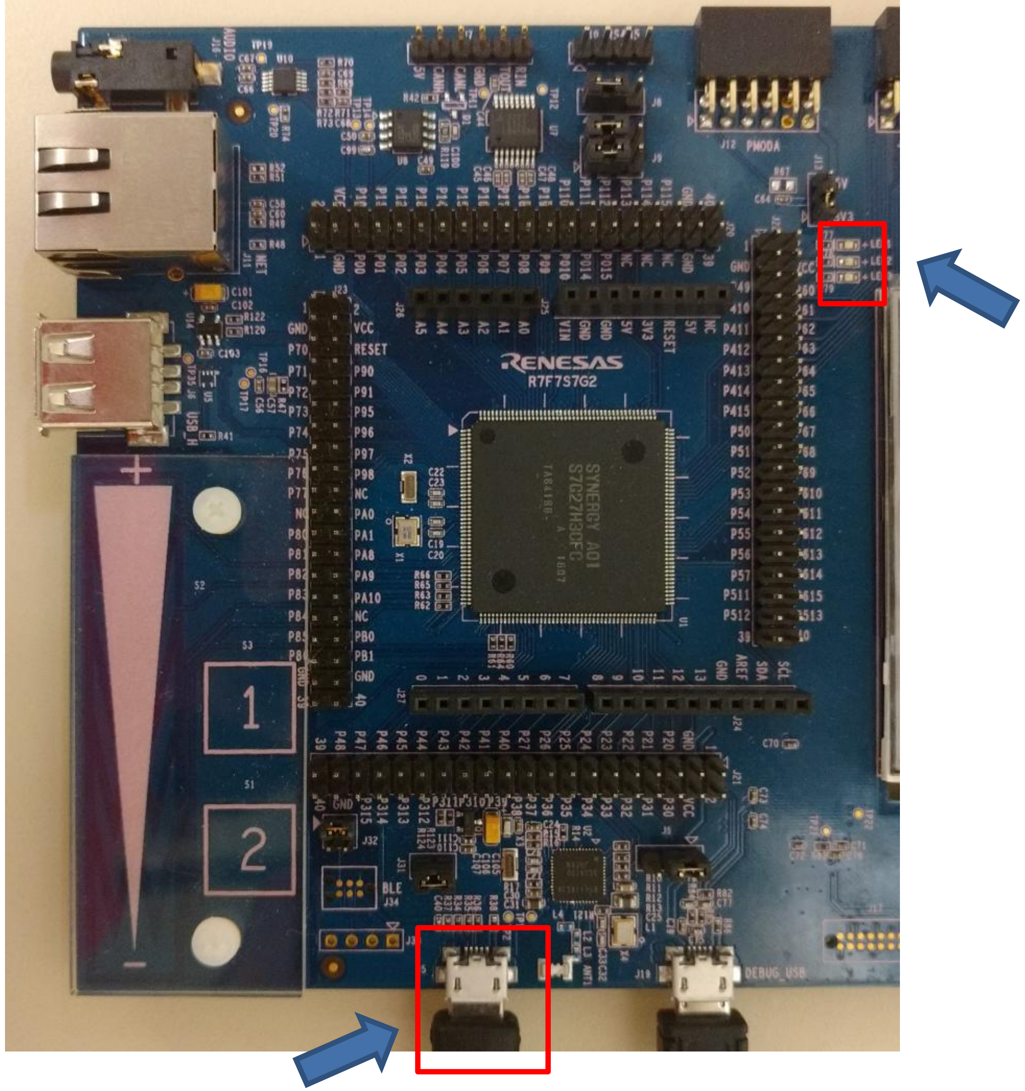
Source: Authors

LAB8 – ACTIVITY 3

Activity 3 – Low-level function prototyping

Our Lab8 application will make use of the SK_S7G2 onboard LEDs and communication port via USB.

The next step for this lab consists on prototyping the low-level functions to control the LEDs and USB and doing basic tests before dealing with the application requirements and full coding.



Source: Authors

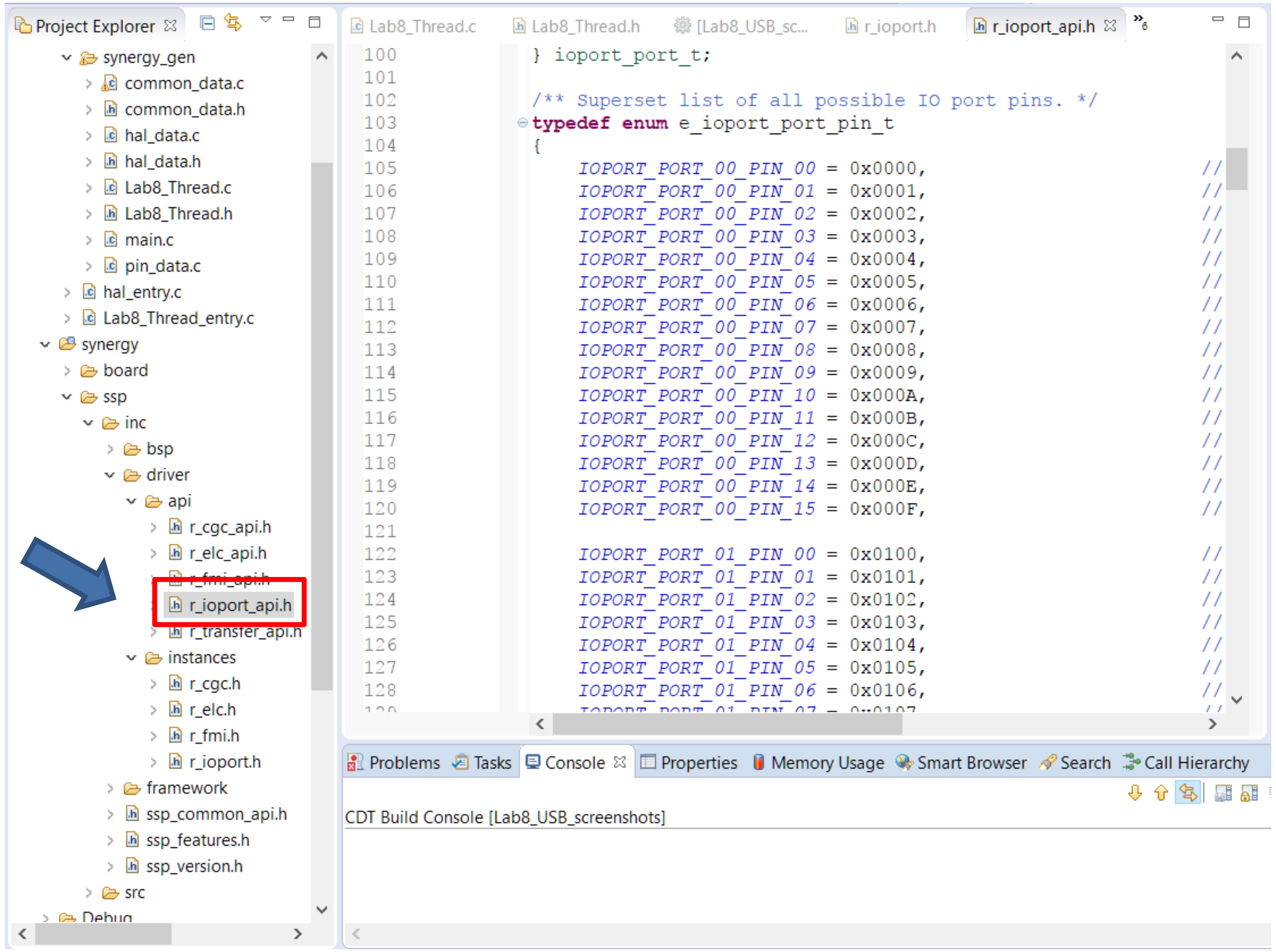
LAB8 – ACTIVITY 3

1. Create a function to turn the LEDs on/off at *Lab8_Thread_entry.c*.

   Hints:

   The hardware mapping for each LED can be done in two ways:

   a) The GPIO pin for each LED can be explicitly mapped to constants. Refer to Lab 2, Activity 4, to learn how to discover which of the GPIO pins are mapped to the LEDs on the SK-S7G2 board. Constant definitions for the GPIO can be found in the *r_ioport_api.h* file.



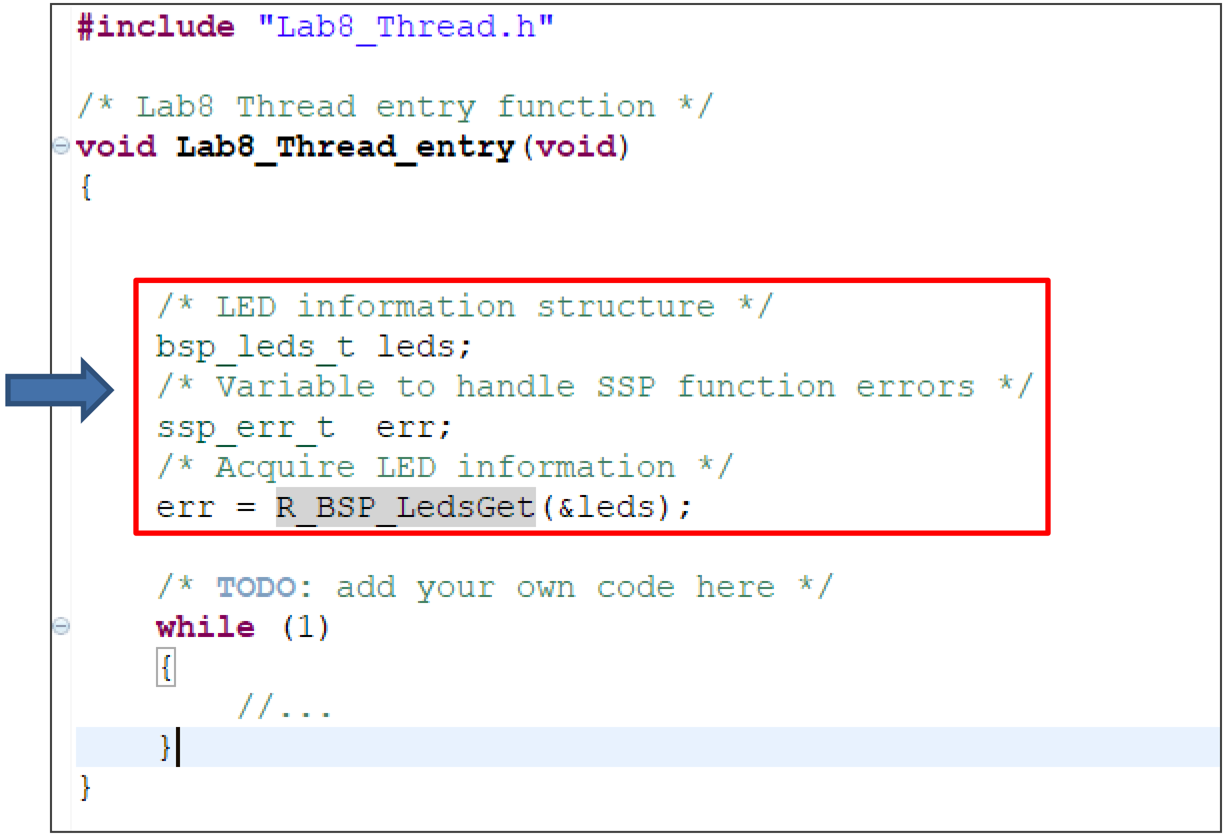
```
} ioport_port_t;

/** Superset list of all possible IO port pins. */
typedef enum e_ioport_port_pin_t
{
    IOPORT_PORT_00_PIN_00 = 0x0000,                    //
    IOPORT_PORT_00_PIN_01 = 0x0001,                    //
    IOPORT_PORT_00_PIN_02 = 0x0002,                    //
    IOPORT_PORT_00_PIN_03 = 0x0003,                    //
    IOPORT_PORT_00_PIN_04 = 0x0004,                    //
    IOPORT_PORT_00_PIN_05 = 0x0005,                    //
    IOPORT_PORT_00_PIN_06 = 0x0006,                    //
    IOPORT_PORT_00_PIN_07 = 0x0007,                    //
    IOPORT_PORT_00_PIN_08 = 0x0008,                    //
    IOPORT_PORT_00_PIN_09 = 0x0009,                    //
    IOPORT_PORT_00_PIN_10 = 0x000A,                    //
    IOPORT_PORT_00_PIN_11 = 0x000B,                    //
    IOPORT_PORT_00_PIN_12 = 0x000C,                    //
    IOPORT_PORT_00_PIN_13 = 0x000D,                    //
    IOPORT_PORT_00_PIN_14 = 0x000E,                    //
    IOPORT_PORT_00_PIN_15 = 0x000F,                    //

    IOPORT_PORT_01_PIN_00 = 0x0100,                    //
    IOPORT_PORT_01_PIN_01 = 0x0101,                    //
    IOPORT_PORT_01_PIN_02 = 0x0102,                    //
    IOPORT_PORT_01_PIN_03 = 0x0103,                    //
    IOPORT_PORT_01_PIN_04 = 0x0104,                    //
    IOPORT_PORT_01_PIN_05 = 0x0105,                    //
    IOPORT_PORT_01_PIN_06 = 0x0106,                    //
    IOPORT_PORT_01_PIN_07 = 0x0107,                    //
```

CDT Build Console [Lab8_USB_screenshots]

Source: Authors

LAB8 – ACTIVITY 3

b) The BSP function *R_BSP_LedsGet ()* (defined in *bsp_common_leds.c*) can be called to fill an instance of *bps_leds_t* with the LED information.



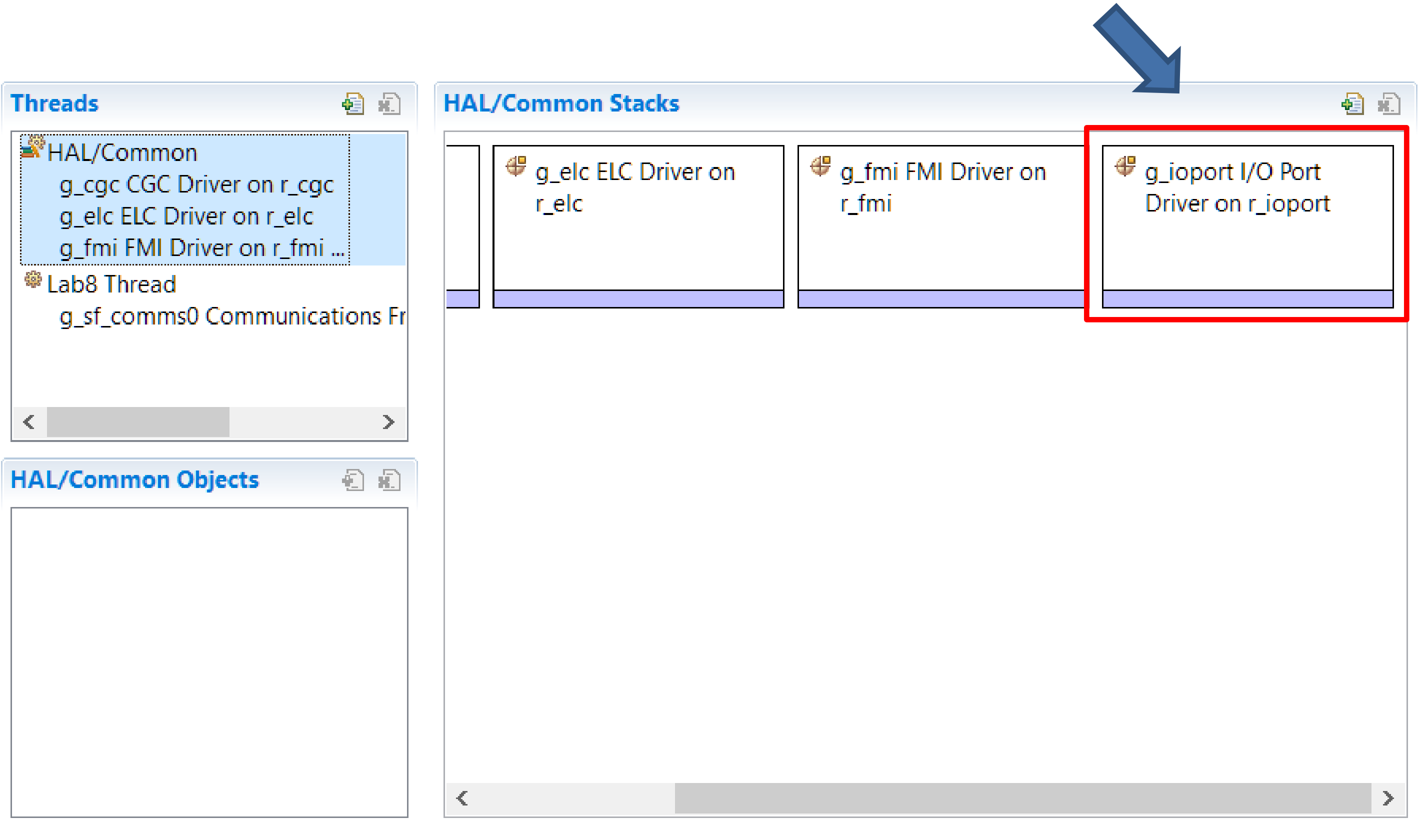
```
#include "Lab8_Thread.h"

/* Lab8 Thread entry function */
void Lab8_Thread_entry(void)
{


    /* LED information structure */
    bsp_leds_t leds;
    /* Variable to handle SSP function errors */
    ssp_err_t  err;
    /* Acquire LED information */
    err = R_BSP_LedsGet(&leds);

    /* TODO: add your own code here */
    while (1)
    {
        //...
    }
}
```

Source: Authors

LAB8 – ACTIVITY 3

- The *g_ioport* instance of the I/O port driver (included as a HAL component into the Synergy Configuration) should be used to command the GPIO pins concerning the LEDs.

  Reference the *p_api* field to access the API and then the *pinWrite()* function to write to the LED pin.



Threads

HAL/Common
g_cgc CGC Driver on r_cgc
g_elc ELC Driver on r_elc
g_fmi FMI Driver on r_fmi ...
Lab8 Thread
g_sf_comms0 Communications Fr

HAL/Common Objects

HAL/Common Stacks

g_elc ELC Driver on r_elc

g_fmi FMI Driver on r_fmi

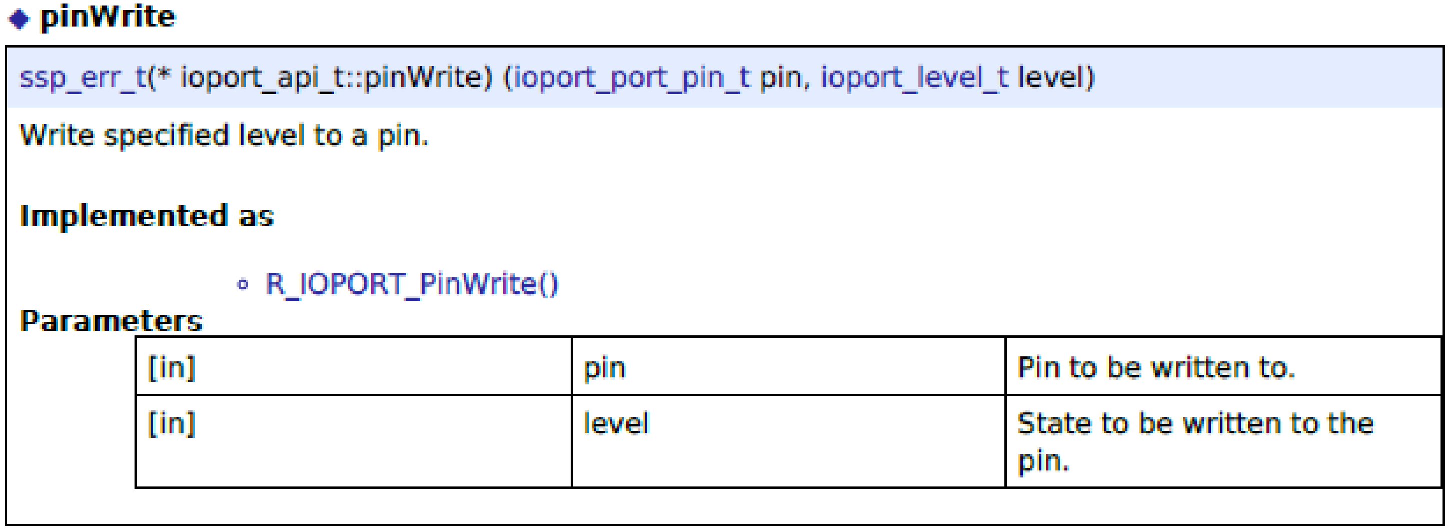
g_ioport I/O Port Driver on r_ioport

Source: Authors

LAB8 – ACTIVITY 3

- The *g_ioport* instance of the I/O port driver (included as a HAL component into the Synergy Configuration) should be used to command the GPIO pins concerning the LEDs.

  Reference the *p_api* field to access the API and then the *pinWrite()* function to write to the LED pin.



◆ **pinWrite**

ssp_err_t(* ioport_api_t::pinWrite) (ioport_port_pin_t pin, ioport_level_t level)

Write specified level to a pin.

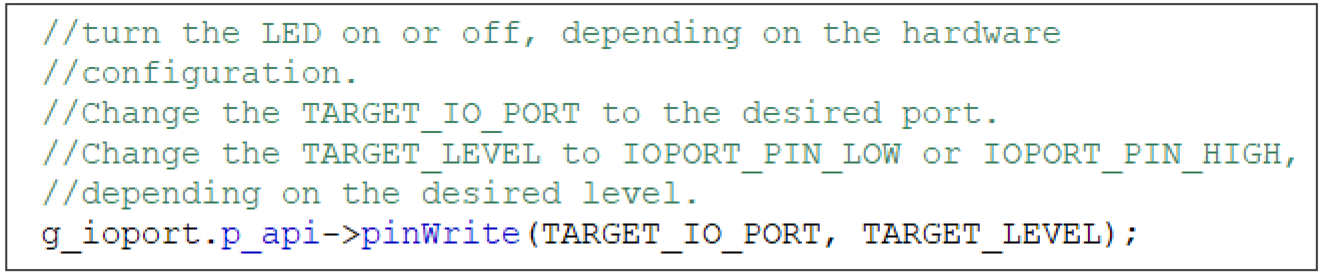
Implemented as

- R_IOPORT_PinWrite()

Parameters

[in]	pin	Pin to be written to.
[in]	level	State to be written to the pin.

Source: Renesas Synergy Software Package v1.7



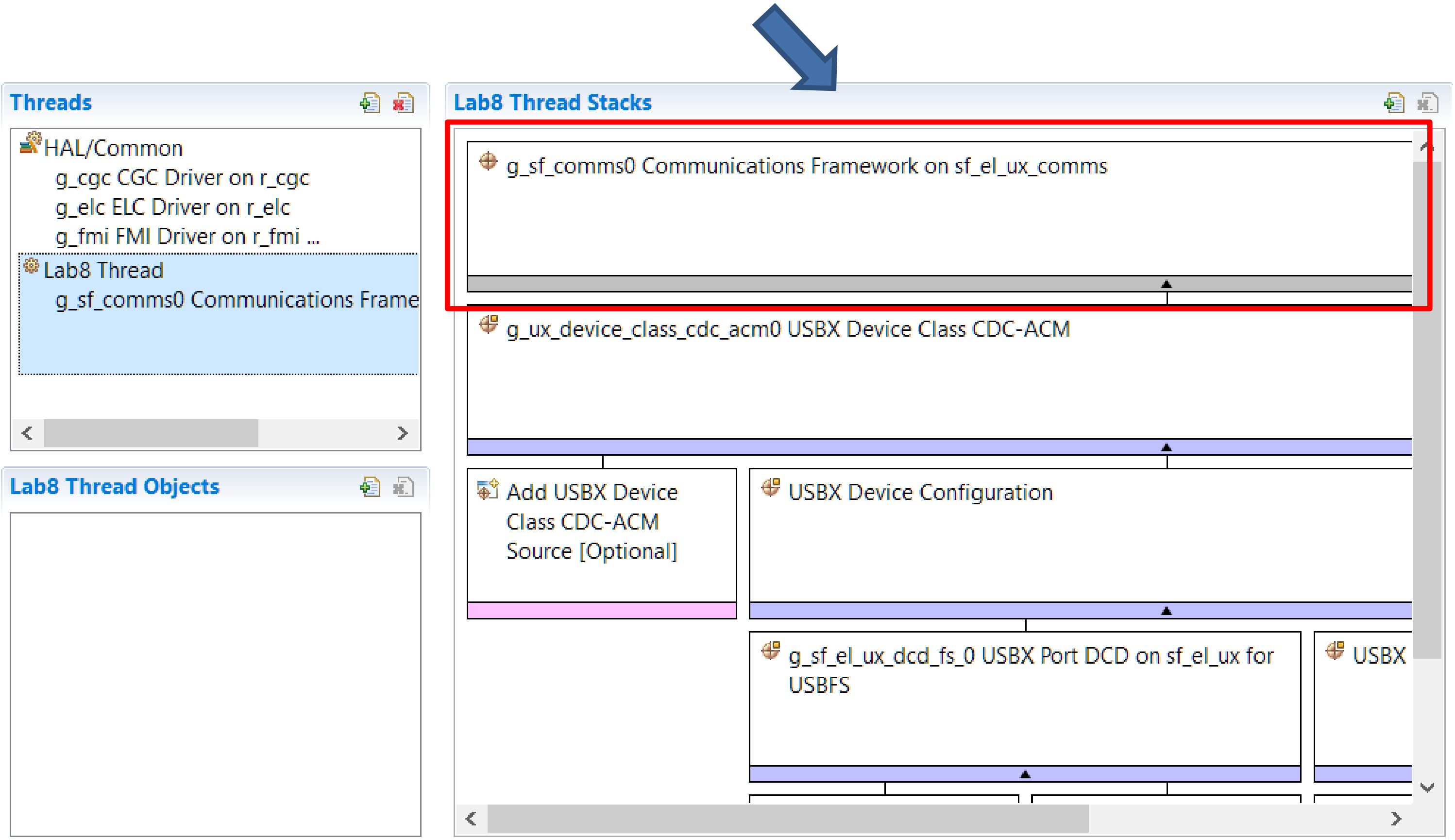
```
//turn the LED on or off, depending on the hardware
//configuration.
//Change the TARGET_IO_PORT to the desired port.
//Change the TARGET_LEVEL to IOPORT_PIN_LOW or IOPORT_PIN_HIGH,
//depending on the desired level.
g_ioport.p_api->pinWrite(TARGET_IO_PORT, TARGET_LEVEL);
```

Source: Authors

LAB8 – ACTIVITY 3

2. Insert at Lab8_Thread_entry() the API calls to send and receive data through the Communications port. Hints:

- use the instance g_sf_comms0 of the Communications Framework component that is configured in the Synergy Configuration Tool

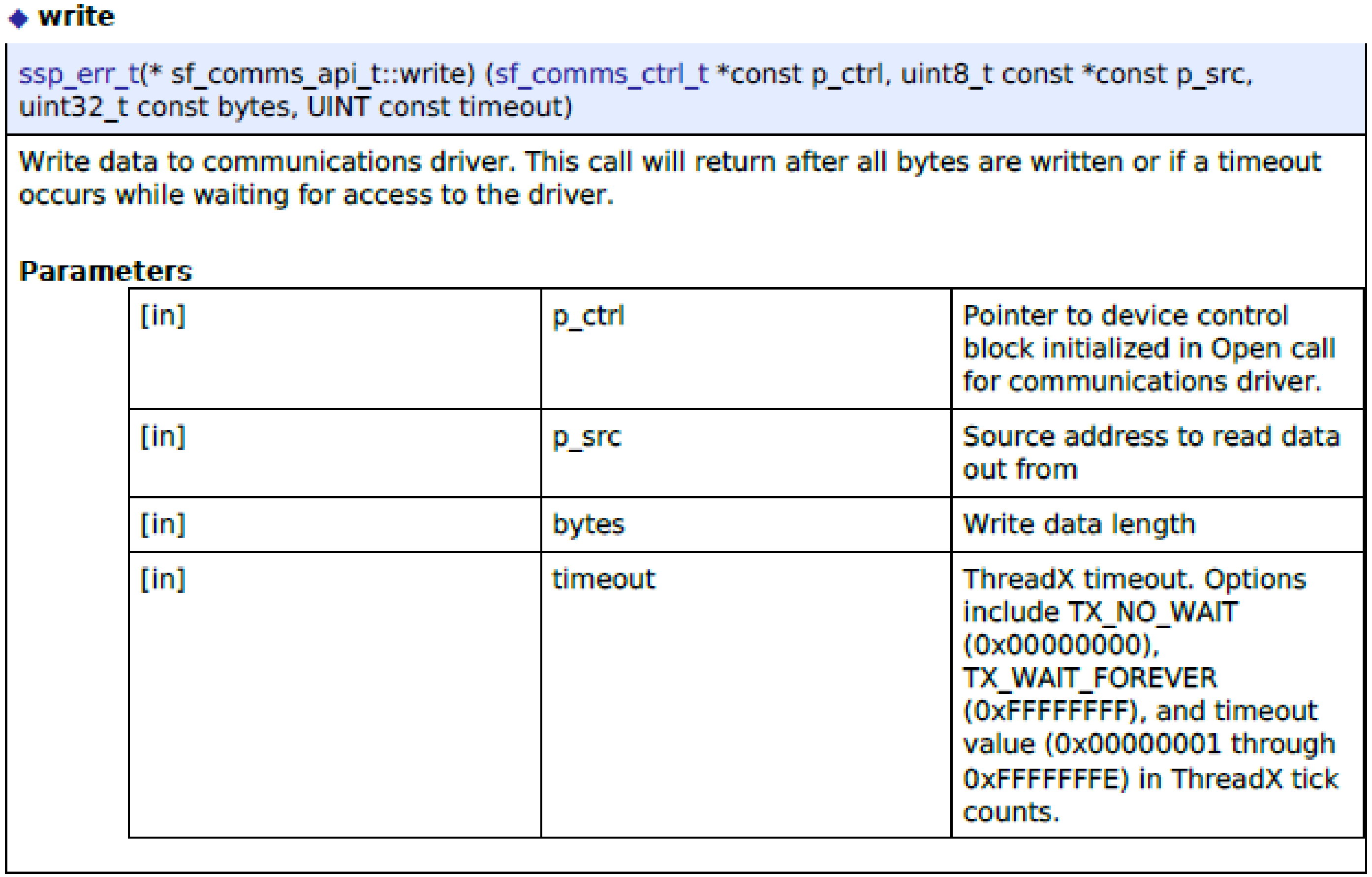


Threads
HAL/Common
g_cgc CGC Driver on r_cgc
g_elc ELC Driver on r_elc
g_fmi FMI Driver on r_fmi ...
Lab8 Thread
g_sf_comms0 Communications Frame
Lab8 Thread Objects
Lab8 Thread Stacks
g_sf_comms0 Communications Framework on sf_el_ux_comms
g_ux_device_class_cdc_acm0 USBX Device Class CDC-ACM
Add USBX Device Class CDC-ACM Source [Optional]
USBX Device Configuration
g_sf_el_ux_dcd_fs_0 USBX Port DCD on sf_el_ux for USBFS
USBX

Source: Authors

LAB8 – ACTIVITY 3

Reference the *p_api* field and then the functions *read()* and *write()* to receive and send data to the PC, respectively. The first parameter for the *read()* and *write()* functions is a pointer to a *sf_comms_ctrl_t* structure, which is a device control block previously initialized for communication. This pointer can be obtained from the *p_ctrl* member of the *g_sf_comms0* instance



◆ **write**

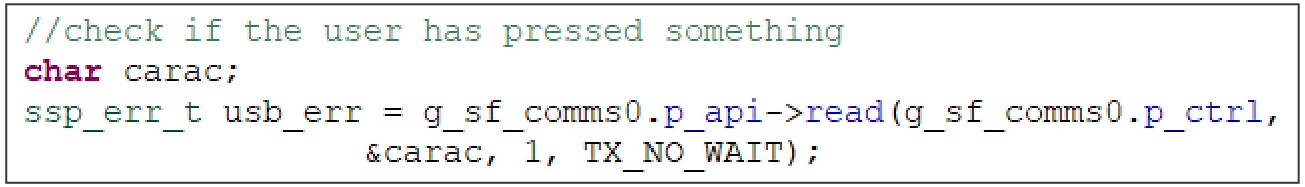
ssp_err_t(* sf_comms_api_t::write) (sf_comms_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes, UINT const timeout)

Write data to communications driver. This call will return after all bytes are written or if a timeout occurs while waiting for access to the driver.

Parameters

[in]	p_ctrl	Pointer to device control block initialized in Open call for communications driver.
[in]	p_src	Source address to read data out from
[in]	bytes	Write data length
[in]	timeout	ThreadX timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

Source: Renesas Synergy Software Package v1.7

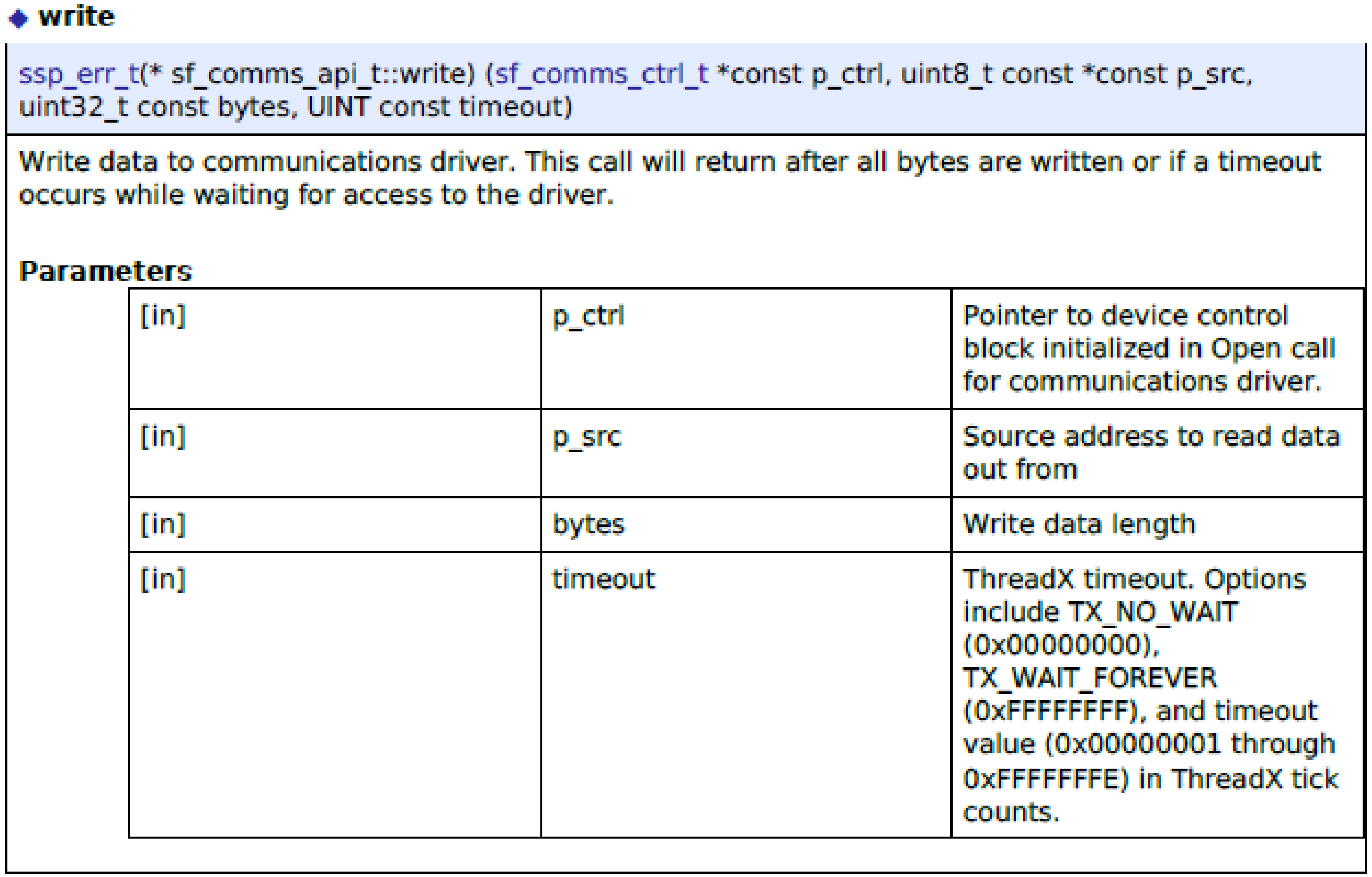


```
//check if the user has pressed something
char carac;
ssp_err_t usb_err = g_sf_comms0.p_api->read(g_sf_comms0.p_ctrl,
                 &carac, 1, TX_NO_WAIT);
```

Source: Authors

LAB8 – ACTIVITY 3

The last parameter for the read() and write() functions defines the timeout for the read and write operations. To command a transmission and wait as long as necessary for the driver access to be scheduled, use *TX_WAIT_FOREVER*. In case of reception, use *TX_NO_WAIT* to return immediately regardless of the number of received bytes.



◆ **write**

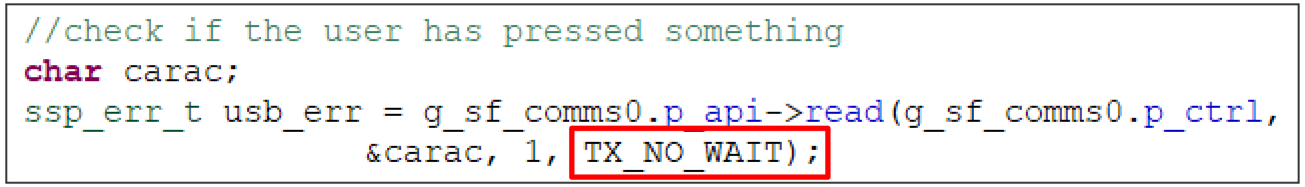
ssp_err_t(* sf_comms_api_t::write) (sf_comms_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes, UINT const timeout)

Write data to communications driver. This call will return after all bytes are written or if a timeout occurs while waiting for access to the driver.

Parameters

[in]	p_ctrl	Pointer to device control block initialized in Open call for communications driver.
[in]	p_src	Source address to read data out from
[in]	bytes	Write data length
[in]	timeout	ThreadX timeout. Options include TX_NO_WAIT (0x00000000), TX_WAIT_FOREVER (0xFFFFFFFF), and timeout value (0x00000001 through 0xFFFFFFFE) in ThreadX tick counts.

Source: Renesas Synergy Software Package v1.7



```
//check if the user has pressed something
char carac;
ssp_err_t usb_err = g_sf_comms0.p_api->read(g_sf_comms0.p_ctrl,
                 &carac, 1, TX_NO_WAIT);
```

Source: Authors

LAB8 – ACTIVITY 3

3. Start a debug session. Refer to Lab 1, Activity 4 for details on debugging.
4. Connect the SK-S7G2 board to the PC via USB interface (connector J5). This will allow the OS installed on the PC to enumerate the board as an USB Communication Device.
   - If the installed OS is Windows 10, the Device Manager will show a new “Communication Device” under “Ports (COM & LPT)”, after the proper USB enumeration and driver installation that follows the first execution of debugging application.
   - In this case, there is a driver issue that may prevent the USB COM port to be opened by a terminal application on the PC. To fix it, select the driver for the enumerated COM port, right-click on "Update driver", search for drivers installed on the computer, uncheck the “Show compatible hardware” option and select SEGGER □ JLink CDC UART Port.
5. Open a Terminal Application on the PC and connect it to the enumerated COM port.
6. Test the LED control and USB communications prototype functions.

LAB8 – ACTIVITY 4

Activity 4 – Application requirements, coding and testing

Use the prototyped low-level functions, implemented to control the LEDs and access the USB communications port, to implement an application logic that performs the following operations:

1. Turn on the LEDs in a binary pattern given by a counter (0 to 7 – 0b000 to 0b111)
2. Increment the counter and sleep for an interval of N ticks (hint: use *tx_thread_sleep(N)*).
3. For every increment of the counter, send the current counter value to the terminal application running on PC via the USB communications port.
4. Repeat steps 1 to 3 in 16 iterations.
5. Check if the user has typed and sent a character from the PC via the terminal application connected to the USB communications port. If so, test the character and increase or decrease the number N of ticks of the sleep interval, depending on the typed character.
6. Repeat this process (steps 1 to 5) in an infinite loop.

Test the application and check the resulting behavior.

LAB8 – ACTIVITY 4

HINTS:

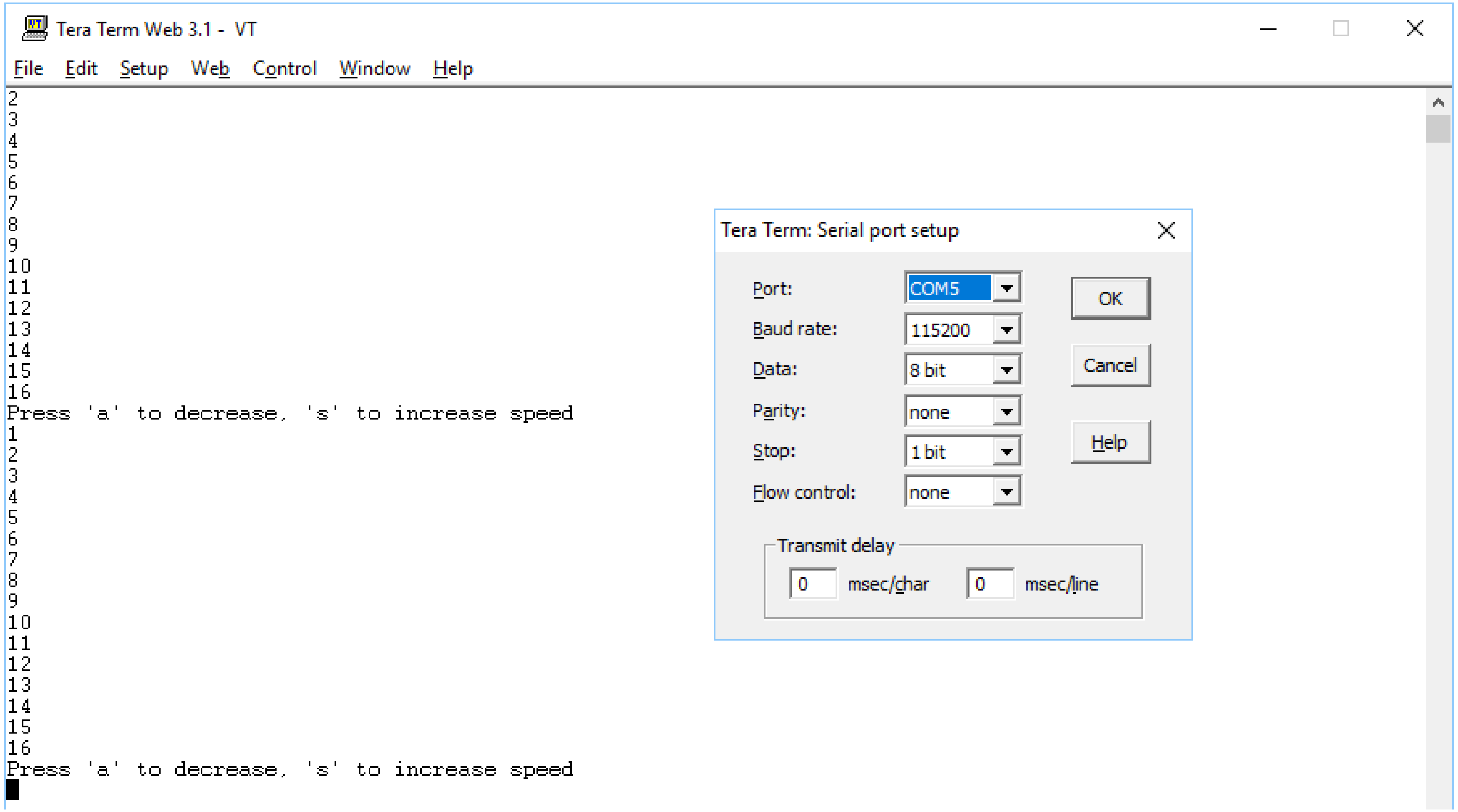
If an error condition occur, such as arriving on error_callback function, or on a call to BSP_CFG_HANDLE_UNRECOVERABLE_ERROR (0); or if nothings shows up on the terminal, then:

- both micro-USB ports of the S7G2 must be connected to the PC, one for debugging and the other for USB communications
- check is the COM port was correctly enumerated by the PC and if the driver in use is actually the SEGGER JLink CDC Uart.
- check if the terminal is configured for 115K baud and 8N1 frame format
- reset the terminal, possibly closing and reopening the terminal application

LAB8 – ACTIVITY 4

Example of output on a terminal emulator.

In this example, when the second USB cable was connected and the software executed on the S7G2, Windows identified the board as COM5.



```
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
Press 'a' to decrease, 's' to increase speed
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
Press 'a' to decrease, 's' to increase speed
```

Tera Term: Serial port setup

Port: COM5
Baud rate: 115200
Data: 8 bit
Parity: none
Stop: 1 bit
Flow control: none

Transmit delay
0 msec/char 0 msec/line

LAB8 – ACTIVITY 5

Activity 5 – Challenge: multi-threaded version

Modify the application structure to separate the logic into two different threads:

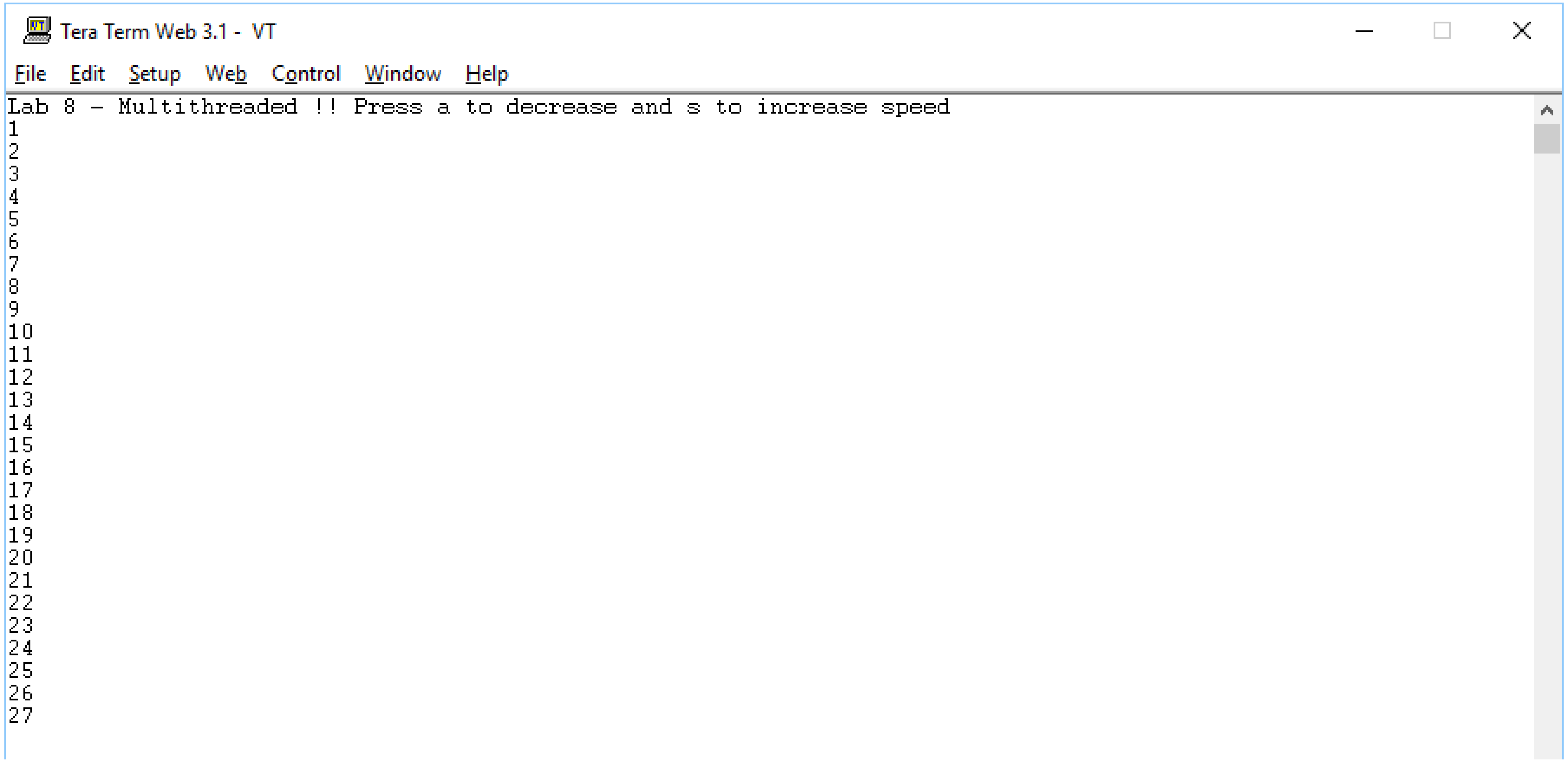
- Thread #1 → increments the counter and sends its current value via the USB Communications port.
  - Remove the limit of 16 iterations, allowing the counter to increase limited only by the size of the counter variable.
- Thread #2 → waits for user input and changes the sleep interval (counting period).

The sleep interval will be able to be changed while the counter is still running.

Hint: the *g_sf_comms0* instance is defined in one of the threads (depending on Synergy Configuration). To access this instance in the other thread, use the #include preprocessor directive to include the header file of the thread where the instance has been created and initialized.

LAB8 – ACTIVITY 5

Activity 5 – Challenge: multi-threaded version



```
Lab 8 - Multithreaded !! Press a to decrease and s to increase speed
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
```

LAB9 – IOT

Objectives:

In Lab 9, the student will be presented to an e2 Studio project that makes use of the Renesas Synergy Wi-Fi Framework to implement an example of IoT (Internet-Of-Things) application.

The SK-S7G2 board is configured to act as an IoT node that reads some sensors (internal CPU temperature) and writes to an actuator (a LED). The commands to read / write are sent to the node by a Remote Client application running on a PC or a smartphone, by means of the network infrastructure (LAN) to which both the IoT node and the Remote Client are connected.

To do this lab you will need the following materials / resources:

- SK-S7G2 kit with the USB Micro-B debug cable
- A GT202 Wi-Fi module with a PMOD plug-in
- A Local Area Network (LAN) with an accessible (i. e. known SSID and password) Wi-Fi AP/Router

LAB9 – IOT

Activities:

1. Hardware setup.
2. Follow Renesas application note R12AN0055EU0106 to build a thermostat application.
3. Based on Renesas application note R12AN0034EU0105, modify this project to include WiFi and access the temperature sensor and the LEDs from a web page.

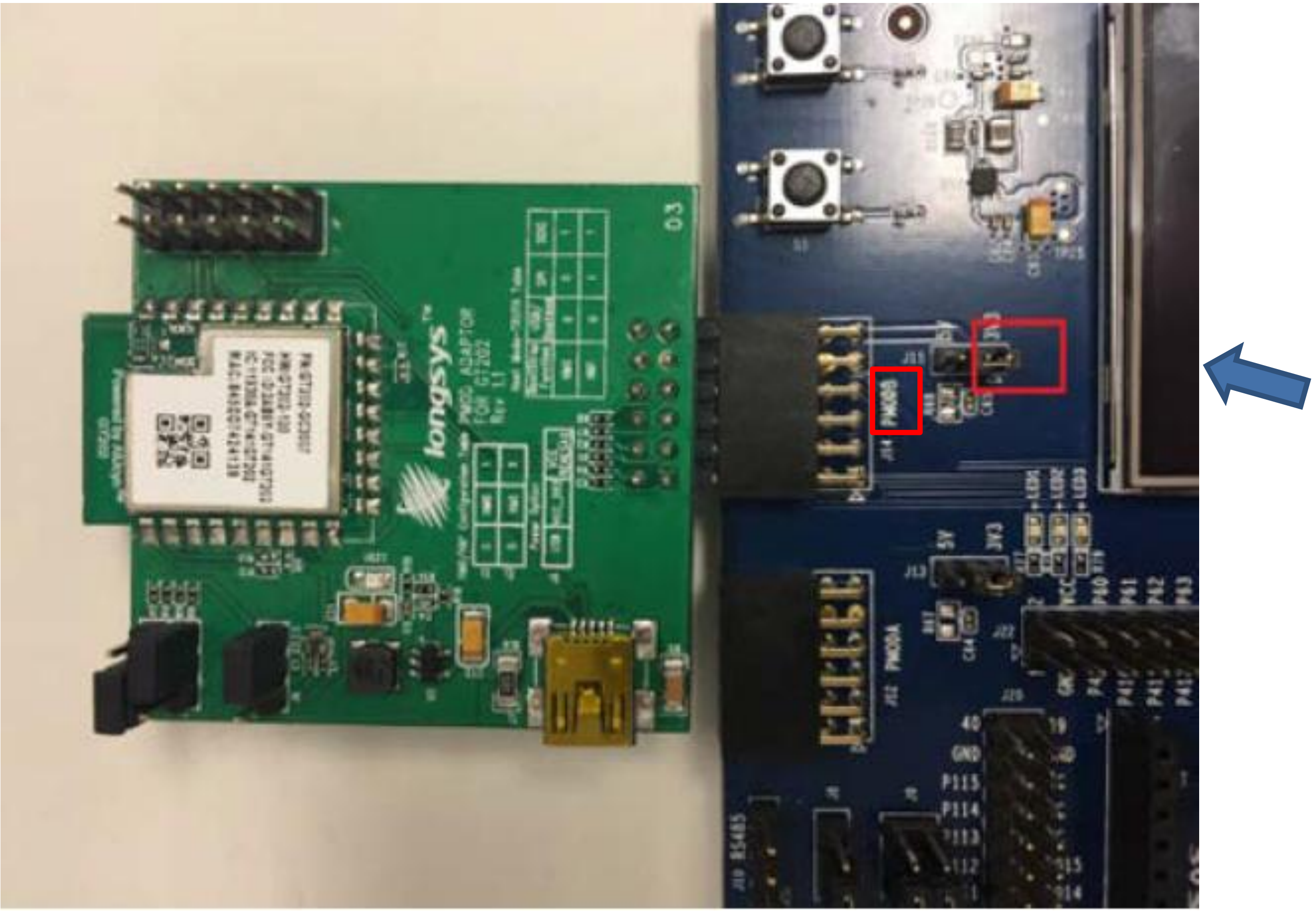
   Note: LAN SSID and password shall be set in the file src\wifi_app_thread_entry.c

LAB9 – ACTIVITY 1

Activity 1 – Hardware Setup:

1. Connect the GT202 Module to SK-S7G2 PMOD B interface.
2. Make sure that jumper J15 is at 3V3 position □ provides the correct voltage level (3.3V) to the module.

Caution! If J15 is not properly configured, the Wi-Fi module can be damaged!



Source: Synergy Wi-Fi Application Project for SK-S7G2
r11an0082eu0101-synergy-wifi-framework-app-note.pdf

LAB9 – ACTIVITY 2

Activity 2 –

1. The application note **R12AN0055EU0106** is provided as part of the lab files for this course in pdf format. Follow the instructions therein.

   Note: this AN is applicable to four different boards. Make sure to target it to the SK-S7G2.

LAB9 – ACTIVITY 3

Activity 3 –

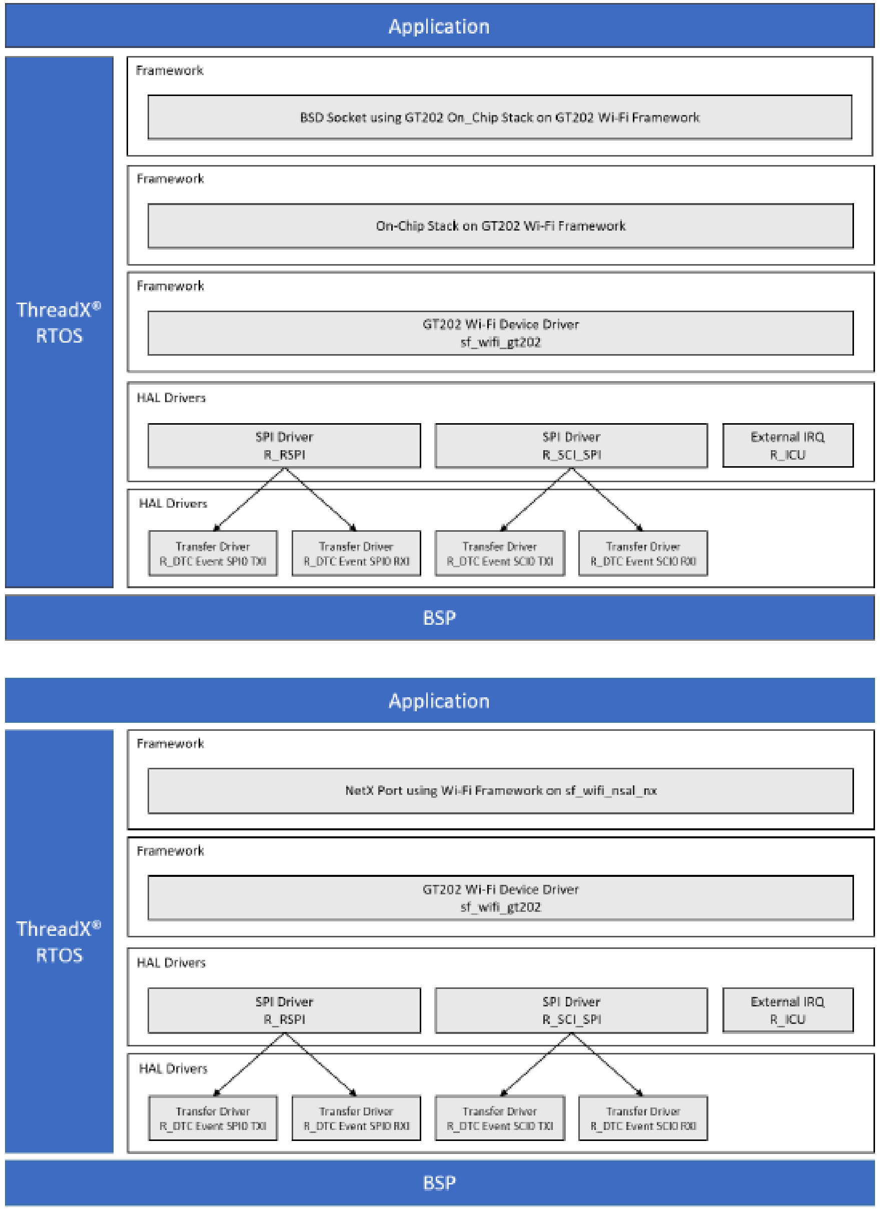
1. The application note **R12AN0034EU0105** is provided as part of the lab files for this course in pdf format. Based on its contents, modify your project to include a WiFi framework and the functionality of remote accessing a sensor and an actuator (LEDs).

LAB9

The Wi-Fi Framework is designed as a layered architecture that is integrated to other SSP components as shown in the figure.

Notice that it can provide a low-level network layer to NetX protocol stack, as well as be directly accessed by an application via its APIs (including on-chip stack implementations).

The IoT Example Project makes use of the NetX Protocol Stack, as shown in the next slides.



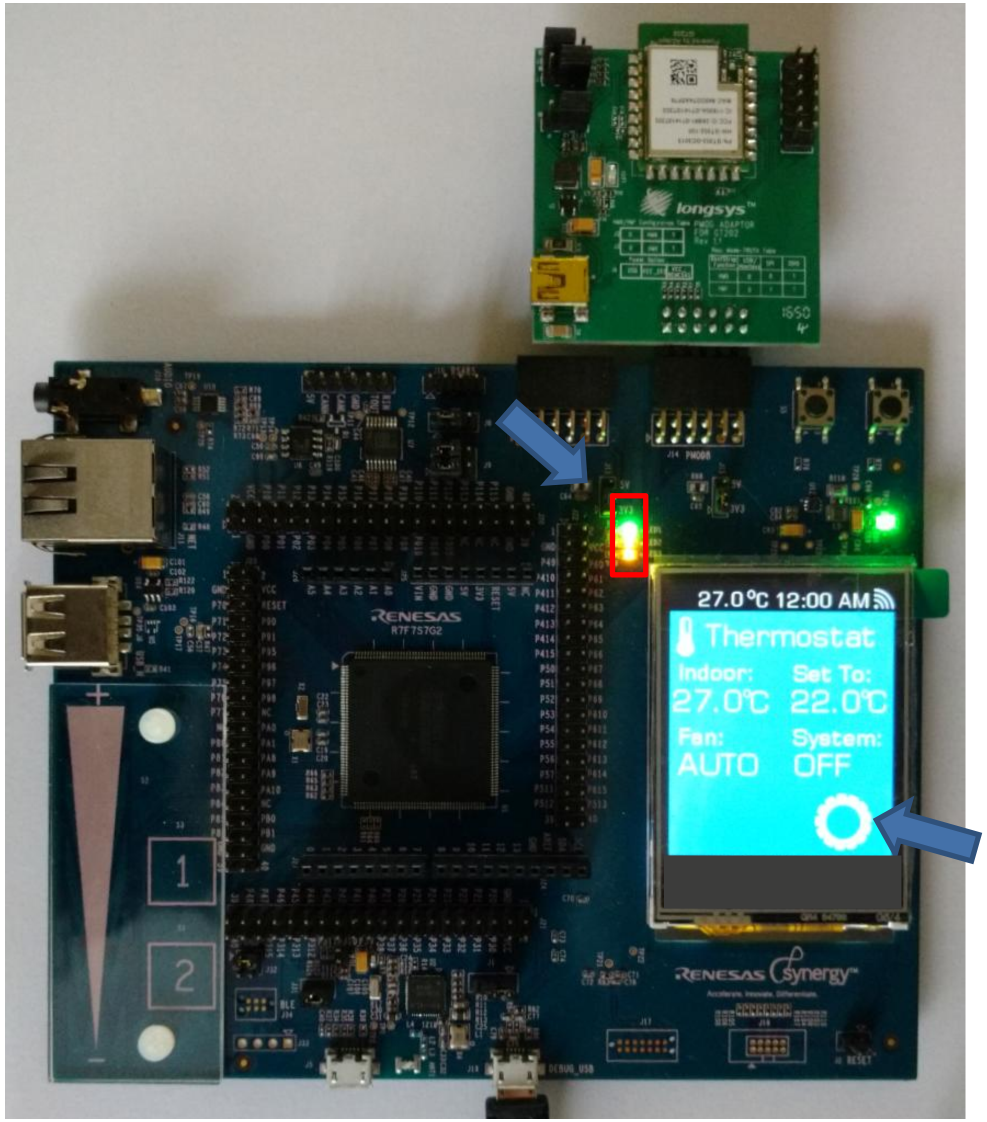
Application
ThreadX® RTOS
Framework
BSD Socket using GT202 On_Chip Stack on GT202 Wi-Fi Framework
Framework
On-Chip Stack on GT202 Wi-Fi Framework
Framework
GT202 Wi-Fi Device Driver sf_wifi_gt202
HAL Drivers
SPI Driver R_RSPI
SPI Driver R_SCI_SPI
External IRQ R_ICU
HAL Drivers
Transfer Driver R_DTC Event SPI0 TXI
Transfer Driver R_DTC Event SPI0 RXI
Transfer Driver R_DTC Event SCI0 TXI
Transfer Driver R_DTC Event SCI0 RXI
BSP
Application
ThreadX® RTOS
Framework
NetX Port using Wi-Fi Framework on sf_wifi_nsal_nx
Framework
GT202 Wi-Fi Device Driver sf_wifi_gt202
HAL Drivers
SPI Driver R_RSPI
SPI Driver R_SCI_SPI
External IRQ R_ICU
HAL Drivers
Transfer Driver R_DTC Event SPI0 TXI
Transfer Driver R_DTC Event SPI0 RXI
Transfer Driver R_DTC Event SCI0 TXI
Transfer Driver R_DTC Event SCI0 RXI
BSP

Source: Renesas Synergy Software Package v1.7 Manual (link)

LAB9

Build and run the application. Verify that the SK-S7G2 kit shows information on the display. If the kit is successfully connected to the Wi-Fi AP, the IP address leased by the DHCP server can be identified in the settings | Network menu entry.

The temperature sensor and the LEDs are configured as a “sensor” and an “actuator” that can be accessed / controlled by the Remote Client via Wi-Fi interface.



settings

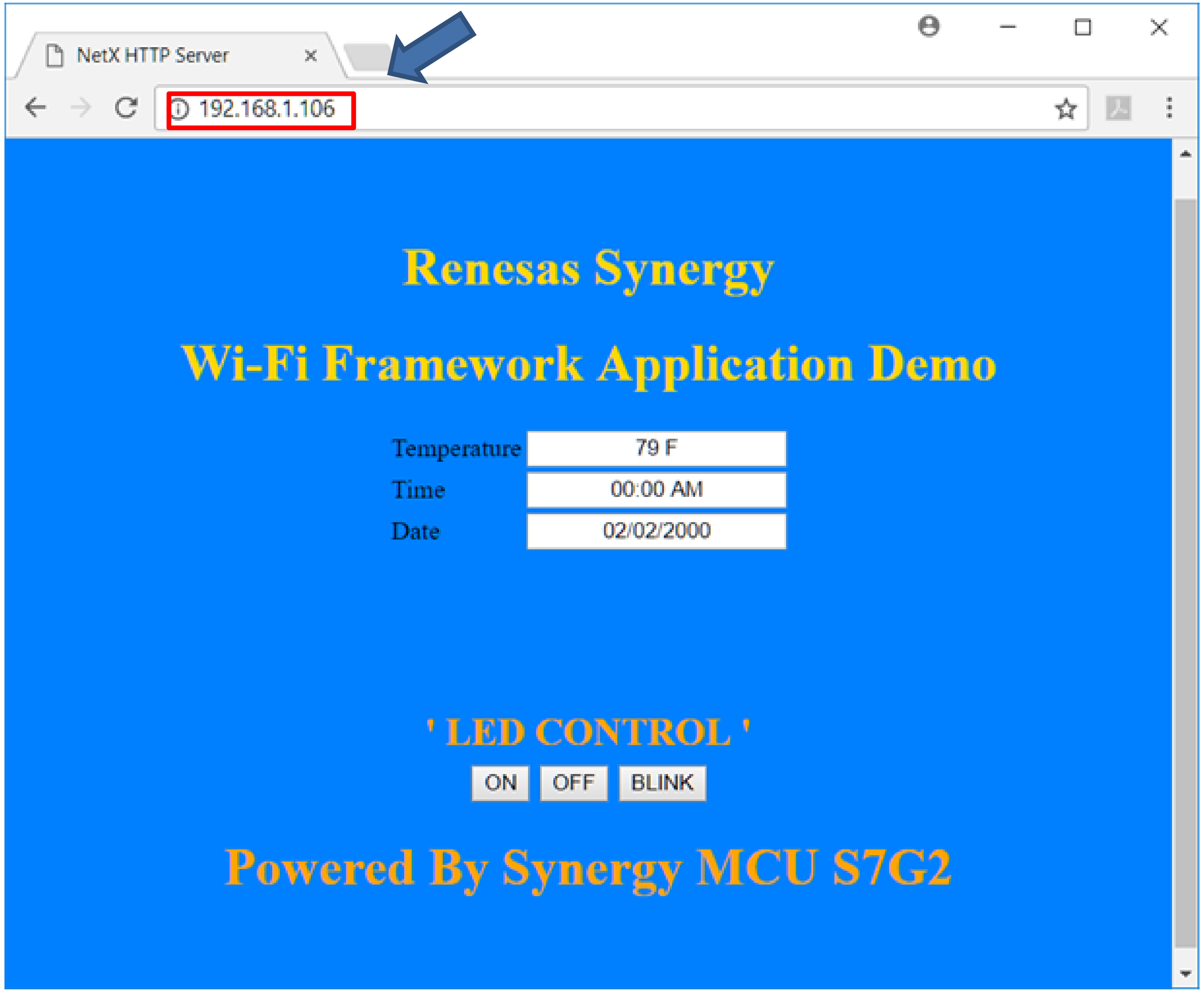
Source: Authors

LAB9

The IoT Node can be remotely accessed by the Remote Client, via HTTP, by browsing its IP address.

Use the web page to control the IoT "actuator" (the LED) and check the status of the "sensor" (thermostat).

The web page also shows date/time as provided by the IoT Node.



Renesas Synergy

Wi-Fi Framework Application Demo

Temperature	79 F
Time	00:00 AM
Date	02/02/2000

' LED CONTROL '

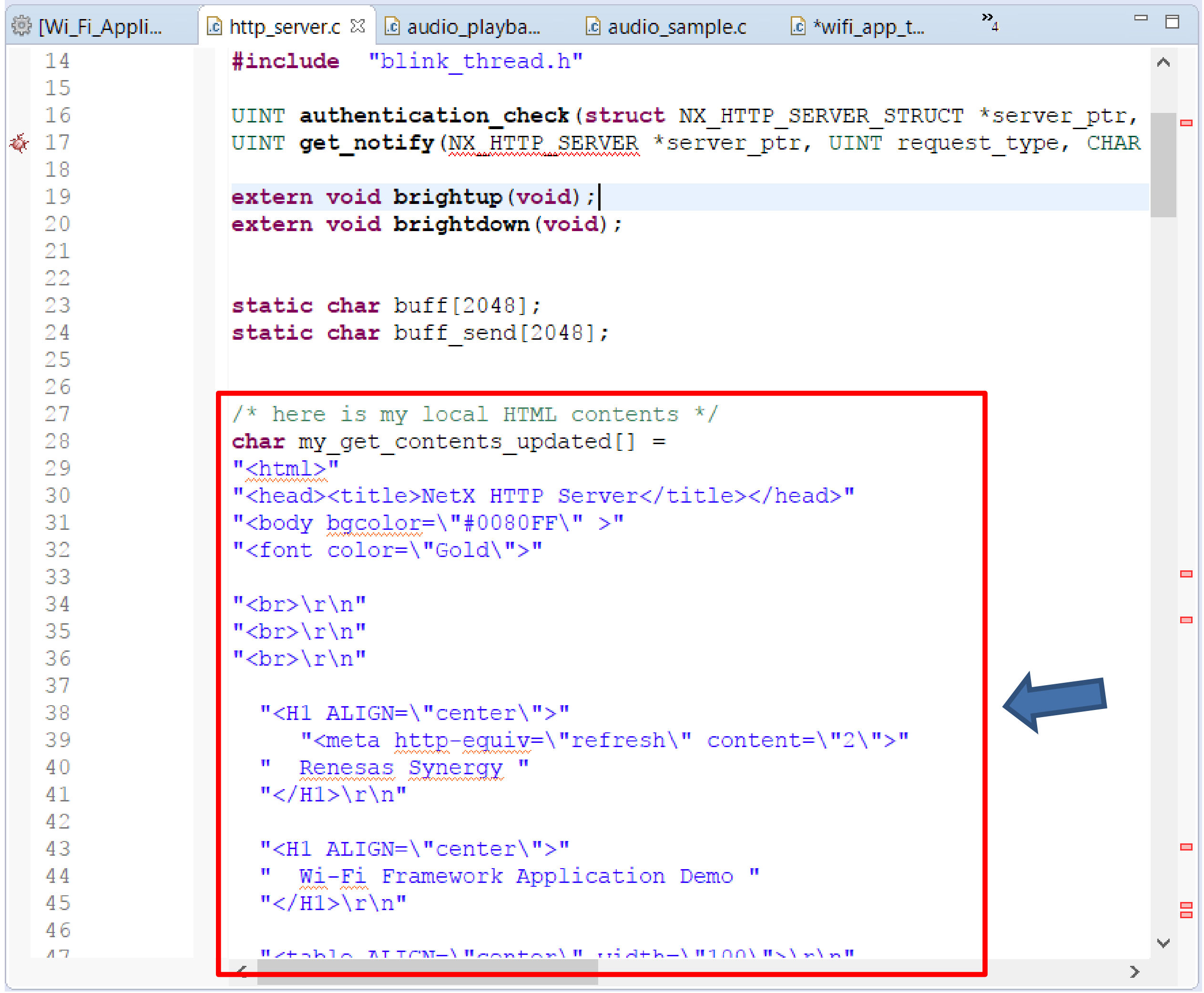
Powered By Synergy MCU S7G2

Source: Authors

LAB9

The HTTP server thread configured into the WiFI Application Project *(http_server_thread_entry.c)* sends the contents defined in *http_server.c* as an HTML page to the web browser.

Try editing the HTML contents in *http_server.c*, rebuilding and running the project, and check the results.



```
#include  "blink_thread.h"

UINT authentication_check(struct NX_HTTP_SERVER_STRUCT *server_ptr,
UINT get_notify(NX_HTTP_SERVER *server_ptr, UINT request_type, CHAR

extern void brightup(void);
extern void brightdown(void);


static char buff[2048];
static char buff_send[2048];


/* here is my local HTML contents */
char my_get_contents_updated[] =
"<html>"
"<head><title>NetX HTTP Server</title></head>"
"<body bgcolor=\"#0080FF\" >"
"<font color=\"Gold\">"

"<br>\r\n"
"<br>\r\n"
"<br>\r\n"

  "<H1 ALIGN=\"center\">"
     "<meta http-equiv=\"refresh\" content=\"2\">"
  "  Renesas Synergy "
  "</H1>\r\n"

  "<H1 ALIGN=\"center\">"
  "  Wi-Fi Framework Application Demo "
  "</H1>\r\n"

  "<table ALIGN=\"center\" width=\"100\">\r\n"
```

Source: Authors

Renesas.com