Australian National University

Week 7 Introduction to microcontrollers and microprocessors Video lecture 9

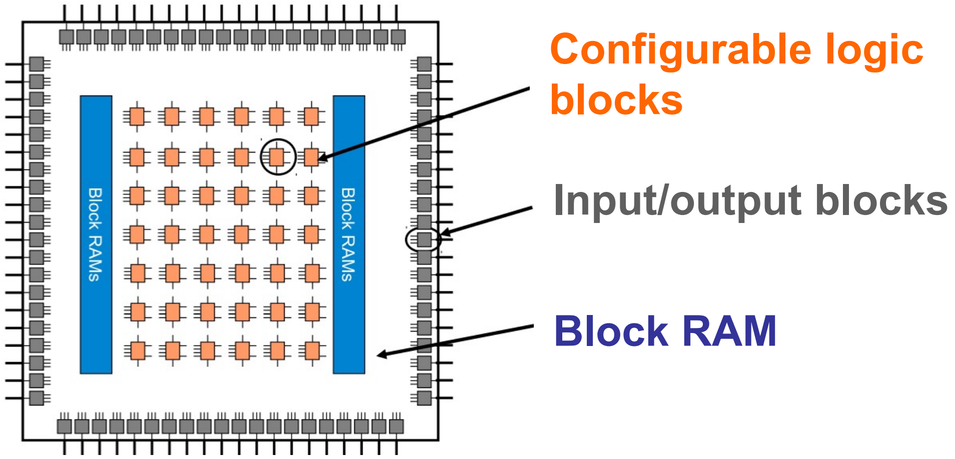
ENGN4213/6213 Digital Systems and Microprocessors

What is in this lecture

- From FPGA to microprocessors
  - When to use one and when to use the other
- Microprocessors and microcontrollers
  - Explanation of these often interchangeably used terms – their differences
- Introduction to the second part of the course
  - Basics of microprocessors/ micro-controllers
  - STM32 ARM micro-controller
  - C language – to program micro-controllers

FPGA (as we know it)

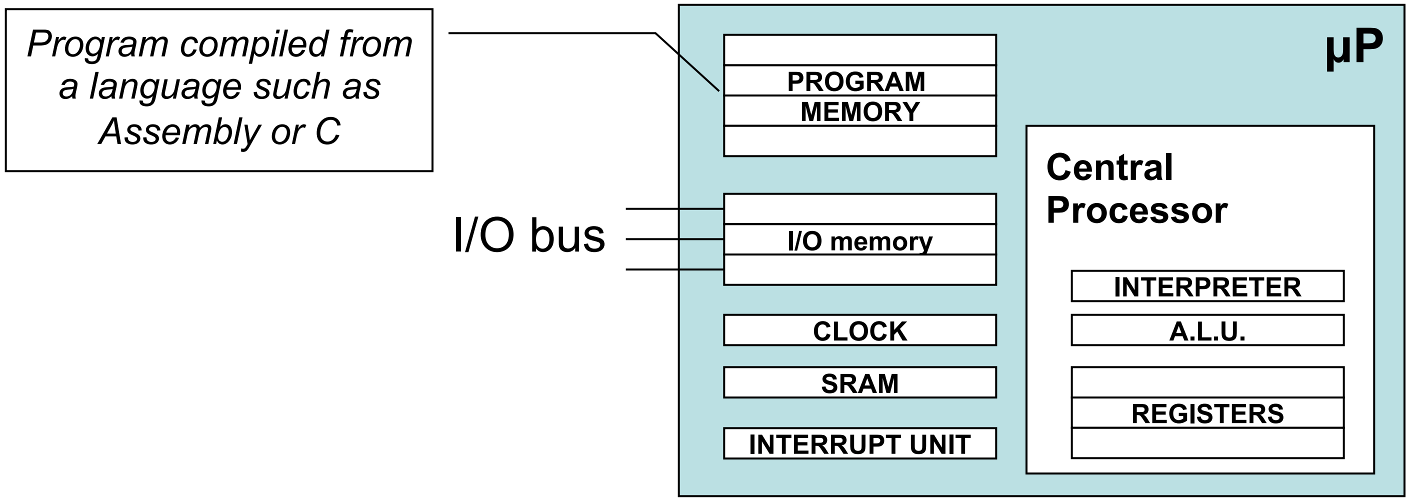
- A chip containing programmable logic



- A bit loader application (e.g., Vivado) programs the LUTs in the configurable logic blocks, the I/O blocks and routes the programmable interconnect using the information in the *bit file* in order to implement a particular design.
  - ***The design architecture can be determined at gate level***

A Microprocessor

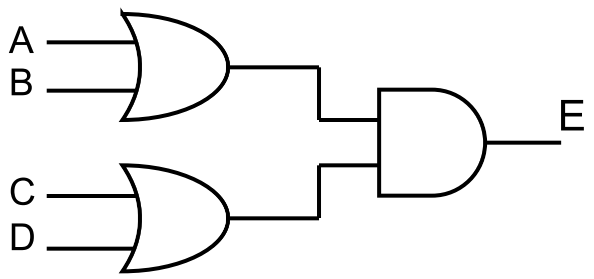
- A digital system with a **fixed** **architecture**
  - *Note: different micros still have different architectures*
- A microprocessor executes a program stored in the *program memory.* Execution is sequential.
  - Data, either stored elsewhere in the system memory or obtained from I/O peripherals, may be processed as part of the program.



FPGA vs Microprocessor

- Each has advantages and disadvantages
- Imagine I want to implement the following operation:

```
E = (A | B) & (C | D)
```

FPGA Programmable logic implementation	MICROPROCESSOR Algorithmic implementation	
	1. Read A 2. Read B 3. Compute A | B 4. Store result r1 5. Read C 6. Read D 7. Compute C | D	8. Store result r2 9. Read r1 10. Read r2 11. Compute r1&r2 12. Store final result E

FPGA vs Microprocessor (2)

- **When is it best to use a FPGA?**
  - When you are interested in ***maximising real-time performance***, e.g., you need a parallel architecture and/or need to operate at high frequencies with hard timing constraints. With a FPGA you can create a truly parallel architecture.
  - When the project does ***not require large memory usage*** (addressing on multi-MB memories may become a challenge for complex designs).
  - When the design does ***not need to interface with too many peripherals***. Designing for FPGA implementation means creating a digital circuit which interfaces with a peripheral's I/O. While a custom design can achieve best performance, peripheral manufacturers won't provide much support to assist with FPGA interfacing.
  - When ***configuration volatility is not an issue***.
- **In a FPGA design you trade silicon (complex, custom logic) for speed**

FPGA vs Microprocessor (3)

- **When is it best to use a Microprocessor?**
  - When you do not have hard timing constraints or when any performance requirements can be met with a less optimised hardware platform.
  - When you need quick up-time (no volatility).
  - When the design needs to be interfaced to large memories (we will see how straightforward RAM addressing is in C!) or other peripherals, for which the manufacturer may provide standard libraries.
  - When the design needs to run / support existing complex softwares.
- **Microprocessors use a fixed architecture. Versatility is achieved through the use of sequential algorithms. This comes at a cost in terms of performance.**

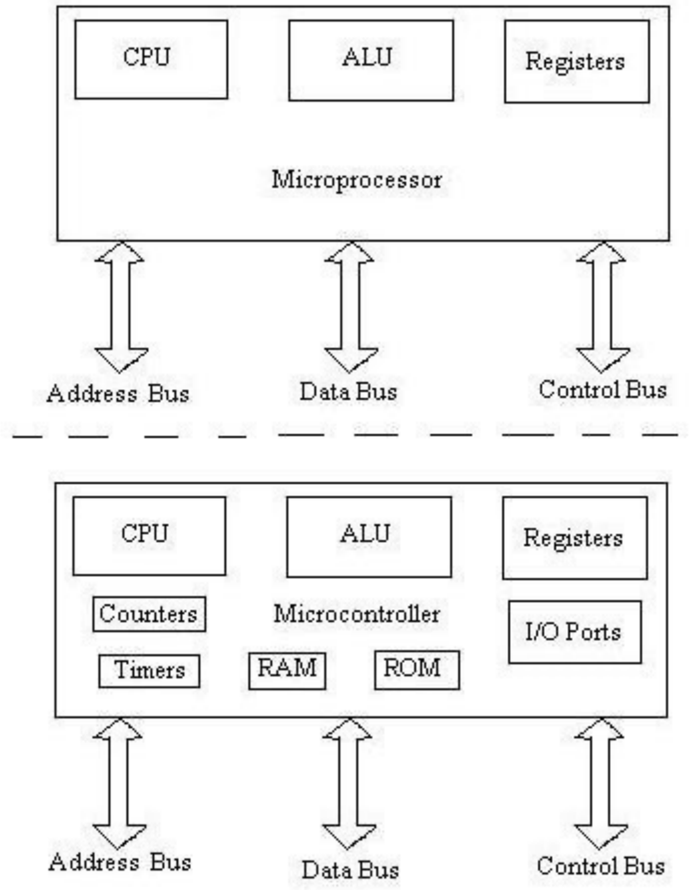
Soft cores

- An in-between concept which can be interpreted as *“the best of both worlds”*
  - Although one could equally say it is *“the worst of both worlds”*

- A *microprocessor soft core* means an implementation of a microprocessor on a FPGA chip.
  - You can then run software written and compiled for a specific micro on a FPGA platform
  - However, the performance is lower than the equivalent chip-based micro
  - Still you get the “hardware flexibility” of FPGA
    - you can for example try different microprocessors without changing the surrounding hardware
    - you could in theory customise your core to tailor it to your application
  - Some names of Xilinx cores: Picoblaze, Pacoblaze, Microblaze

Microprocessors vs Microcontrollers

- **Microprocessors, Microcontrollers (MCUs)** all have a **central processing unit**.
- A **Central Processing Unit (CPU)** consists of the **Arithmetic Logic Unit (ALU)** and **Controller** that provide the processing power.
- **Microcontrollers** are small low cost computers with a range of built in peripherals: SRAM, Flash, IO ports, Watchdog timer, UARTs - They are complete systems.
- **Microprocessors** have higher performance CPUs that interface to external peripherals.



Australian National University

General structure of a microprocessor

- We look here at a simplified general model of a CPU, as each manufacturer/model uses a unique design:

- An ***arithmetic-logic unit (ALU)***
  - performs arithmetic or logic operations on data
- A **control unit**
  - directs the flow of information
- A number of **registers**
  - store interim data, program execution info, memory addresses, i/o data flow…
- **Buses**
  - internal connections for the transfer of data
- **Memory**
  - not strictly part of the CPU. Stores data.

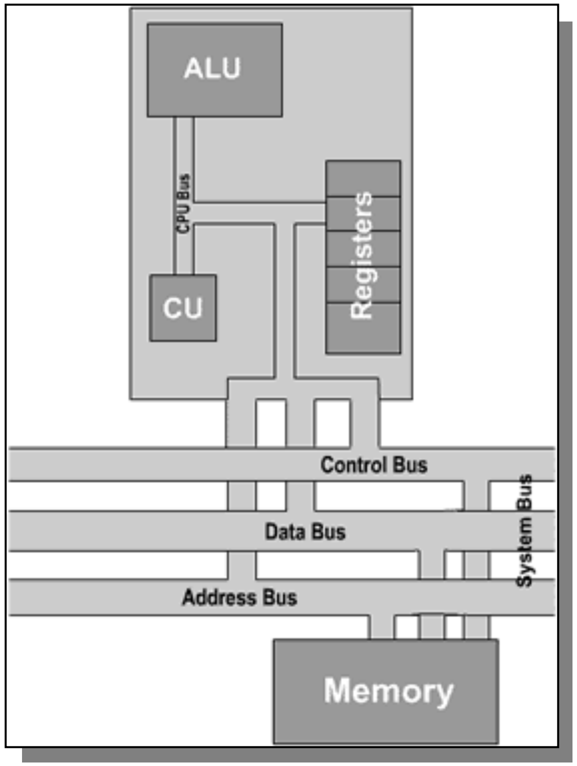


Image from http://www.eastaughs.fsnet.co.uk/cpu/structure-index.htm

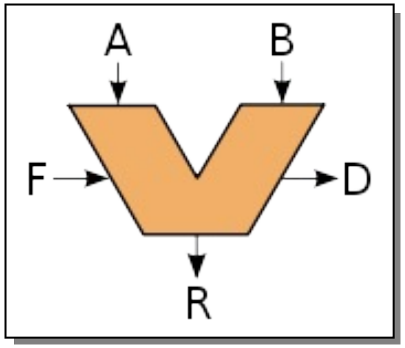
Other general microprocessor information

- The **number of bits**, which gives an idea of what degree of *parallel computation* the unit is capable of
  - This informs the size of the internal buses
  - For many years personal computers ran on 32-bit CPUs, lately 64-bit.
- The architecture type
  - ***Von Neumann architecture***, where program instructions and data are saved in the same memory
  - ***Harvard architecture*** where program instructions and data are stored in different memories

- Have you noticed that a microprocessor can be represented by **a number of functional blocks and registers connected by buses**?
  - We can use our knowledge of digital systems to understand the inner workings of microprocessors.
  - You could even design a new type if you needed/wanted to!

Arithmetic logic unit

- Usually depicted with a "V" shaped symbol to indicate that it takes data from two *n-bit*-long buses to create one *n-bit*-long result.
- It is a circuit which can be controlled by *control inputs* (*"F"* below) to perform a range of operations, e.g.
  - Sum (remember adders?)
  - Incrementing/Decrementing
  - Logical tests
  - Comparison operations
  - Bit shifting
  - …



- ALUs are generally **combinational circuits**. They can also have additional outputs (*"D"* above), usually single-bit flags, carry bits etc.
- Not all ALUs are the same, they can be made simple or extremely complex depending on the manufacturer's choice
  - Design / production cost vs performance trade-offs apply

The Control Unit

- Is a **state machine** running on the system clock
- Its role is to **interpret the program instructions** and **generate suitable control signals** to manage the flow of information throughout the rest of the CPU (the so-called *datapath*)
- Main features:
  - An **input decoder** (translates instructions)
  - **Synchronous** operations (dependable timing of operations)
  - An **output logic**
- The control unit does not do the calculations, it *just* enables the other subcomponents to “come together” to obtain the complex performance required.

The Registers

- Memory locations within the CPU which are functional to storing data requiring fast access.
- The number and type of registers varies for different CPUs, but the main ones are:
  - **Program counter (PC):** stores the memory address of the next instruction to be executed in the program algorithm.
  - **Instruction register (IR):** stores the current instruction which the Control Unit is to execute.
  - **Accumulator (Acc):** stores intermediate results from the ALU. Useful in multi-step calculations
  - **Memory address (MAR) and memory buffer registers (MBR)**
  - **Flag registers:** stores flags which can be raised by the ALU
  - Other **general purpose registers (**also called ***working registers*****):** used to store other data requiring fast access

Historical development of Microprocessors

- https://spectrum.ieee.org/tech-history/silicon-revolution/the-surprising-story-of-the-first-microprocessors
- (This article can be accessed from Reverse Proxy login to ANU library)
- Search the Web

Micro-controller families

- The 1st microcontroller TMS1000 is designed in 1971 by Smithsonian institution

- 8051: First manufactured in 1985. This is an 8-bit microcontroller. In this you can store numbers from 0 to 255. Coming to the instruction set 8051 has 250 instructions which take 1 to 4 machine cycles to execute. The speed of the 8051 microcontroller is 1 million instructions per second. The ALU of the 8051 makes computations simple. In 8051 family there is no inbuilt memory bus and A/D converters. 8051 microcontroller has 32 I/O pins, timers/counters, interrupts and UART's.

Source: https://www.engineersgarage.com/tutorials/getting-started-with-microcontrollers/

Australian National University

Micro-controller families

- AVR is an 8-bit RISC architecture microcontroller (first produced in 1996). There are 16-bit and 32-bit microcontrollers also available in the same family. RISC means Reduced Instruction Set Computer.
- AVR has 140 instructions which are all 1 cycle based instructions. By default AVR microcontrollers operate with the 1 MHz clock cycle. The speed of AVR microcontroller is 12 million instructions per second. AVR controllers has number of I/O ports, timers/counters, interrupts, A/D converters, USART, I2C interfaces, PWM channels, on-chip analog comparators.
- **PIC**(Programmable interface controller): Available in 8-bit, 16-bit and 32-bit microcontrollers. PIC has nearly 40 instructions which all are take 4 clock cycles to execute. The speed of the PIC controller is 3 million instructions per second. It has on-chip peripherals like SPI, ADC, I2C, UART, analog comparator, internal RC oscillator, in-system programmability.

Source: https://www.engineersgarage.com/tutorials/getting-started-with-microcontrollers/

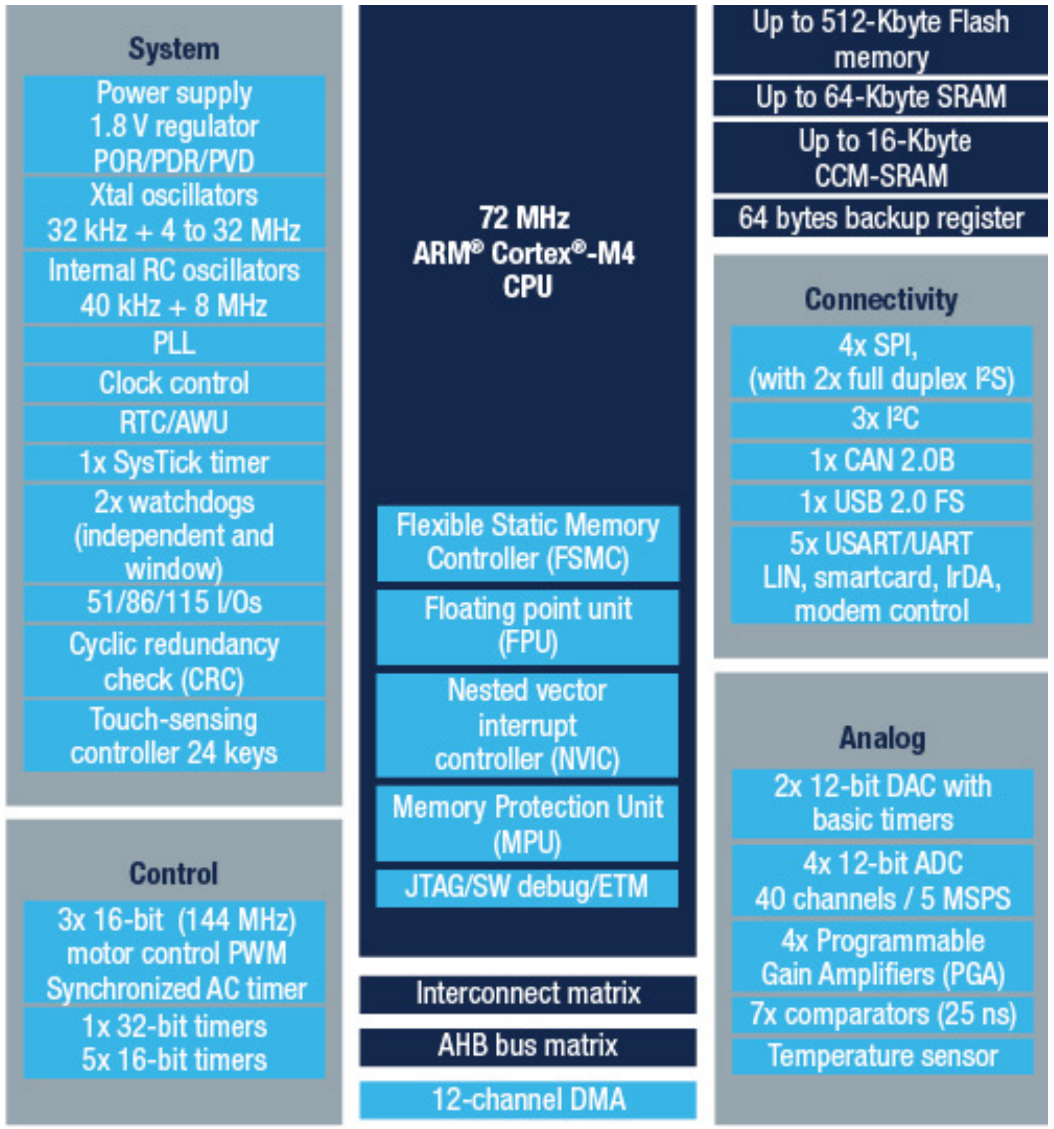
Micro-controller families

- **ARM** (Advanced RISC machine). This is the advanced RISC controller. ARM has the features like load-store architecture, fixed-length instruction set and 3-address instruction format. It has 32-bit ARM instruction set and 16-bit Thumb compressed instruction set. So many on-chip peripheral are there and on-chip debugger, on-chip boot loaders, on-chip RTC, DAC also available.
- https://en.wikipedia.org/wiki/ARM_architecture

Source: https://www.engineersgarage.com/tutorials/getting-started-with-microcontrollers/

STM32F411, a microcontroller

- In the labs, we will use **STM32F411RE Nucleo board.**
- STM32 has a family of micro-controllers for various applications and performance levels.
- Check out their website:
- https://www.st.com/en/evaluation-tools/stm32-nucleo-boards.html
- Manufacturer (STM) provides a comprehensive software
- **STM32CubeIDE** (compiler for all operating systems) https://www.st.com/en/development-tools/stm32cubeide.html



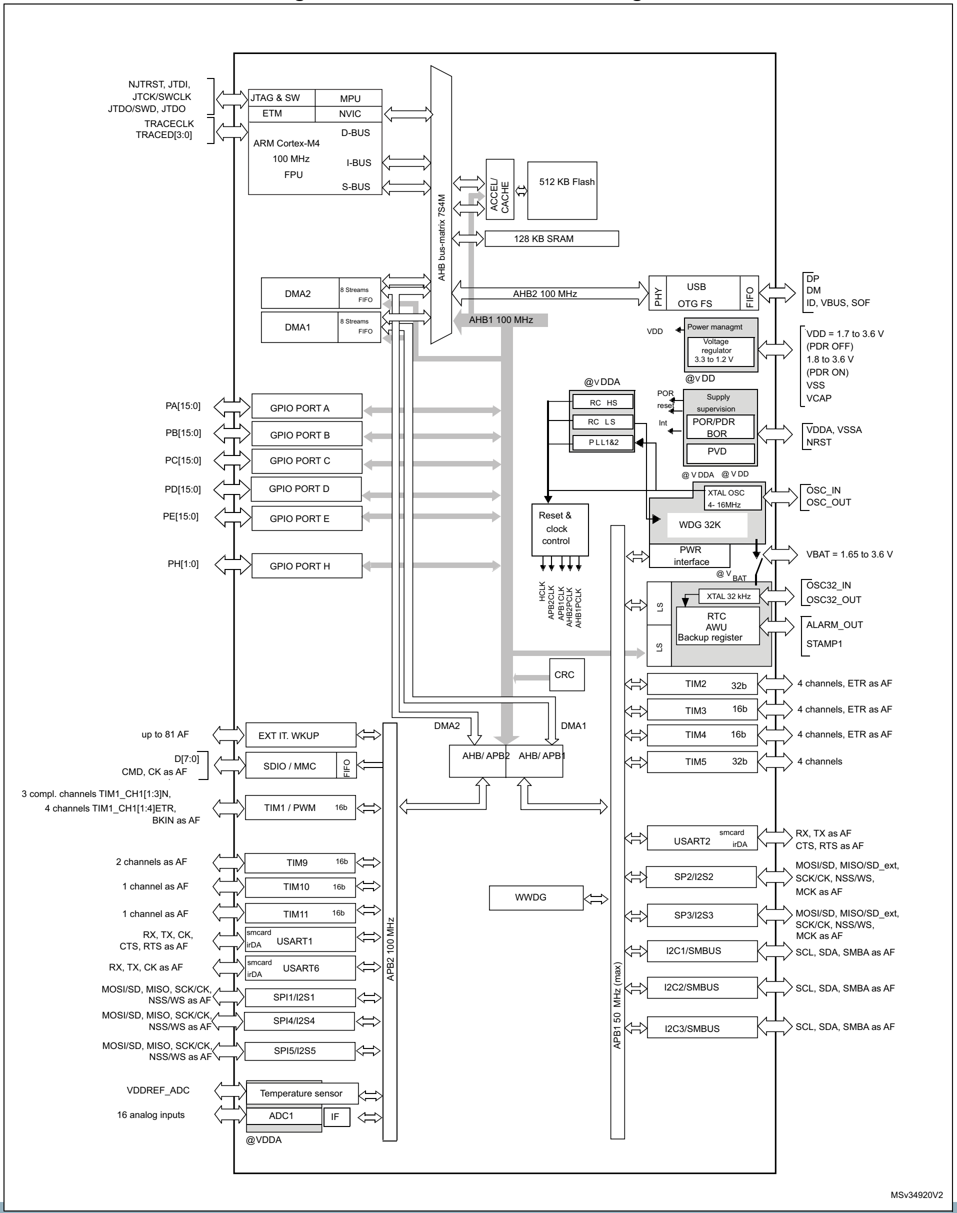
System
Power supply 1.8 V regulator POR/PDR/PVD
Xtal oscillators 32 kHz + 4 to 32 MHz
Internal RC oscillators 40 kHz + 8 MHz
PLL
Clock control
RTC/AWU
1x SysTick timer
2x watchdogs (independent and window)
51/86/115 I/Os
Cyclic redundancy check (CRC)
Touch-sensing controller 24 keys
Control
3x 16-bit (144 MHz) motor control PWM Synchronized AC timer
1x 32-bit timers 5x 16-bit timers
72 MHz ARM® Cortex®-M4 CPU
Flexible Static Memory Controller (FSMC)
Floating point unit (FPU)
Nested vector interrupt controller (NVIC)
Memory Protection Unit (MPU)
JTAG/SW debug/ETM
Interconnect matrix
AHB bus matrix
12-channel DMA
Up to 512-Kbyte Flash memory
Up to 64-Kbyte SRAM
Up to 16-Kbyte CCM-SRAM
64 bytes backup register
Connectivity
4x SPI, (with 2x full duplex I²S)
3x I²C
1x CAN 2.0B
1x USB 2.0 FS
5x USART/UART LIN, smartcard, IrDA, modem control
Analog
2x 12-bit DAC with basic timers
4x 12-bit ADC 40 channels / 5 MSPS
4x Programmable Gain Amplifiers (PGA)
7x comparators (25 ns)
Temperature sensor

Australian National University

STM32F411, a microcontroller

- Complex structure
- Datasheet is available from
- https://www.st.com/resource/en/datasheet/stm32f411ce.pdf

Figure 3. STM32F411xC/xE block diagram



STM32F411RE Nucleo board

- https://www.st.com/en/evaluation-tools/nucleo-f411re.html

- A useful video (Getting Started with STM32 and Nucleo)
- https://www.youtube.com/watch?v=hyZS2p1tW-g

C - Programming language

- **Assembly** is a language which closely reflects the *machine code* understood by the CPU.
  - Imagine human words to help memorise some sequence of 0s and 1s
  - Step-by-step programming
- **C** is a higher level language, which can abstract more complex behaviour into a single command.
  - Most operations we ask of our microprocessor require multiple steps
  - Higher-level programming speeds up the programming process
  - With C you still retain a lot of control over the microprocessor’s behaviour so you can really optimise your code for performance.

- The fundamental steps of programming:
  - Coding
  - Compiling (translating human words into machine code)
  - Saving the compiled file in the microprocessor’s program memory and executing!

Summing up

- FPGA vs Microprocessor
  - When to use one and when the other
  - Soft cores

- Introductions to Microprocessors and microcontrollers
- Rest of the course:
  - Self learn C
  - Use STM32 Nucleo board to learn to program a micro-controller and design embedded systems.