18ECE204J-**ARM-based Embedded System Design**

UNIT 1

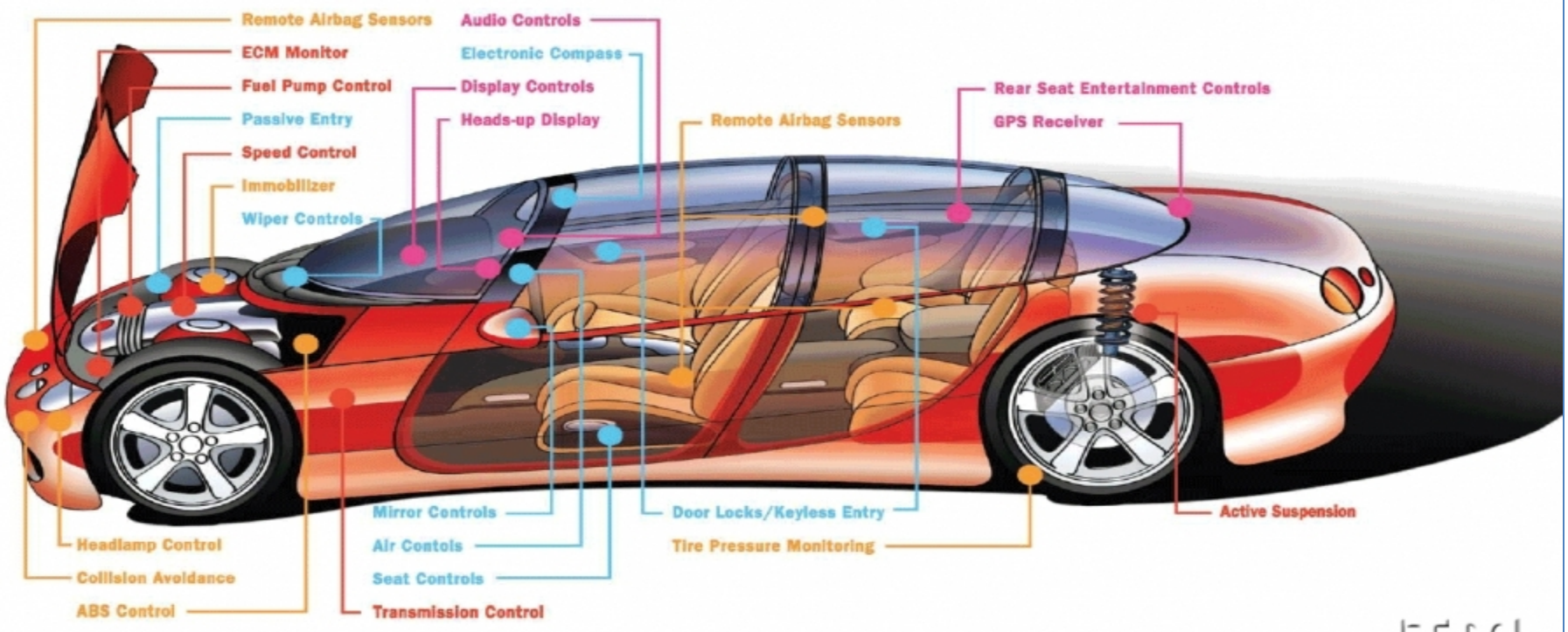
- **Introducing embedded systems and mbed**
- **ARM Cortex assembly language basics.**
- **Lab-1:Assembly language program, simulation -1**
- **Cortex-M processor architecture and Basics : Programming exercises**
- **Development Environment using the mbed**
- **Lab 2: Assembly language program, simulation-2**
- **Keil IDE and Debugging tools**
- **C- language review**
- **Embedded C , introduction**
- **Lab 3: Parallel port programming, simulation**

Introducing embedded systems and mbed

- Instead of putting a microprocessor in a general-purpose computer, it can also be placed inside a product which has nothing to do with computing, like a washing machine, toaster, or camera.
- The microprocessor is then customized to control that product. The computer is there, inside the product; but it can't be seen, and the user probably doesn't even know it's there.
- Moreover, those addons which we normally associate with a computer, like a keyboard, screen, or mouse, are also nowhere to be seen.
- We call such products embedded systems, because the microprocessor that controls them is embedded right inside.
- Because such a microprocessor is developed to control the device, in many cases, those used in embedded systems have different characteristics from the ones used in more general purpose computing machines.
- We end up calling these embedded ones microcontrollers.

Introducing embedded systems and mbed

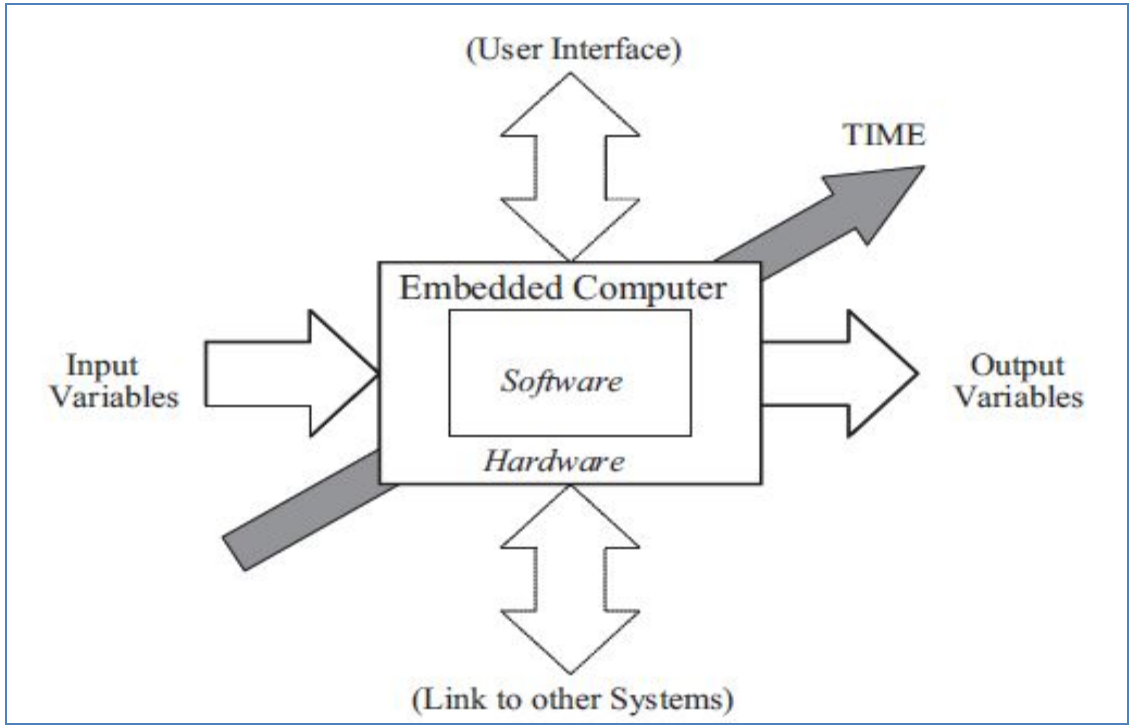
- Most modern domestic appliances, like a washing machine, dishwasher, oven, central heating, and burglar alarm, are embedded systems.
- The motor car is full of them, in engine management, security (for example, locking and antitheft devices), air-conditioning, brakes, radio, and so on
- They are found across industry and commerce, in machine control, factory automation, robotics, electronic commerce, and office equipment.



4. Today's vehicles feature many more MCUs, which control numerous functions. *(Courtesy of Microchip Technology)*

Introducing embedded systems and mbed

- The actual electronic circuit, along with any electromechanical components, is often called the hardware;
- the program running on it is often called the software.
- Aside from all of this, there may also be interaction with a user, for example via a keypad and display, and there may be interaction with other subsystems elsewhere



An Example Embedded System

A snack vending machine

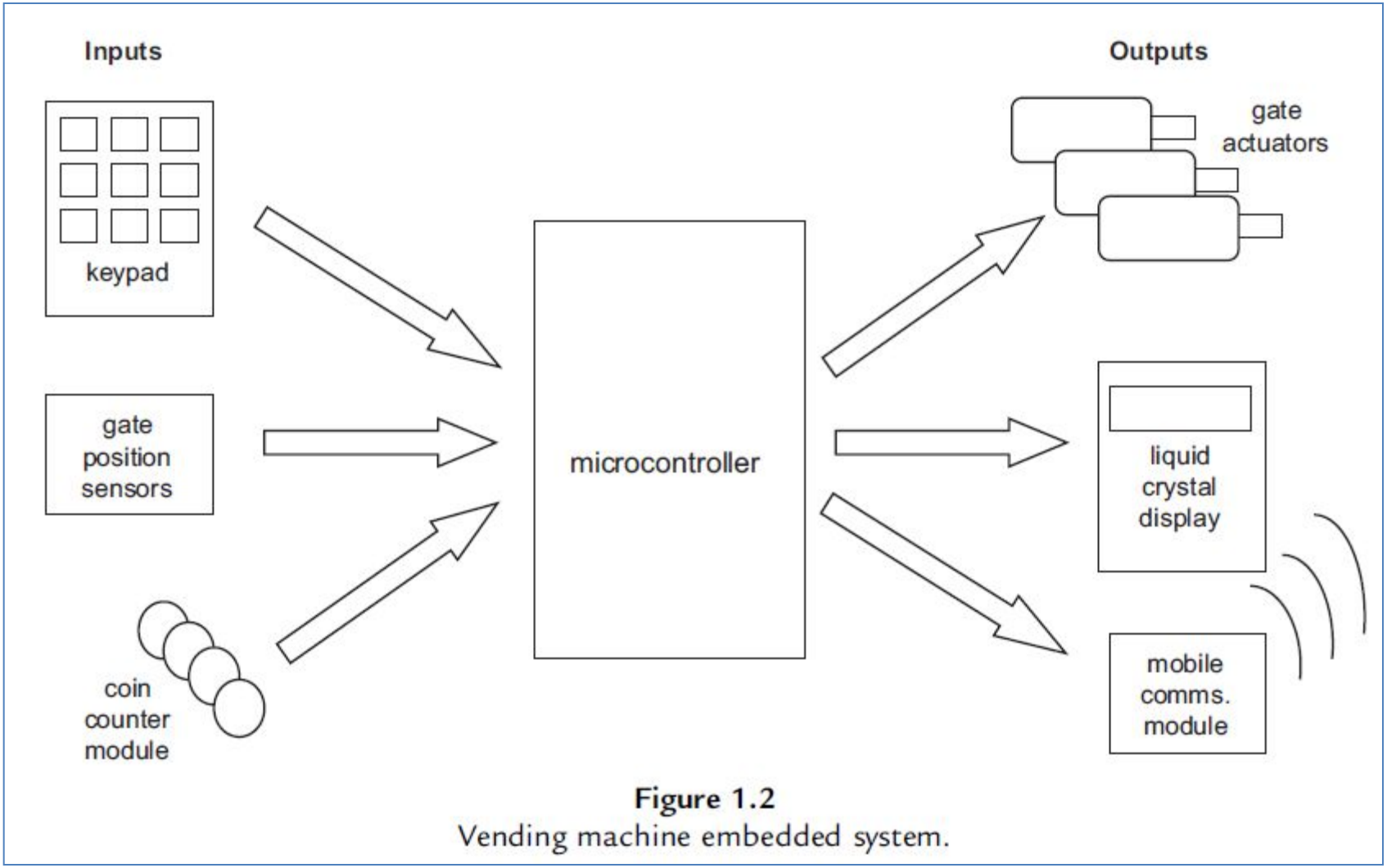
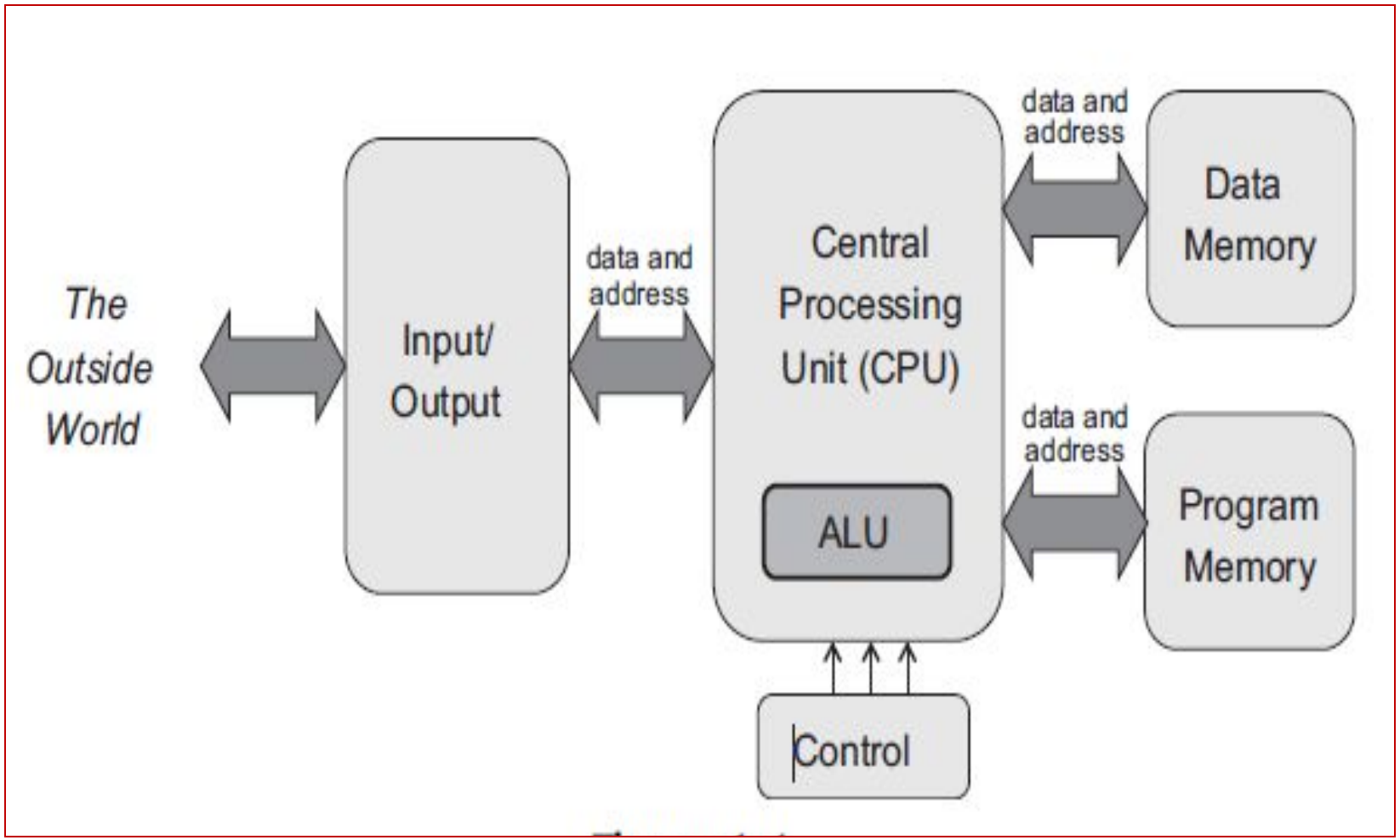


Figure 1.2
Vending machine embedded system.

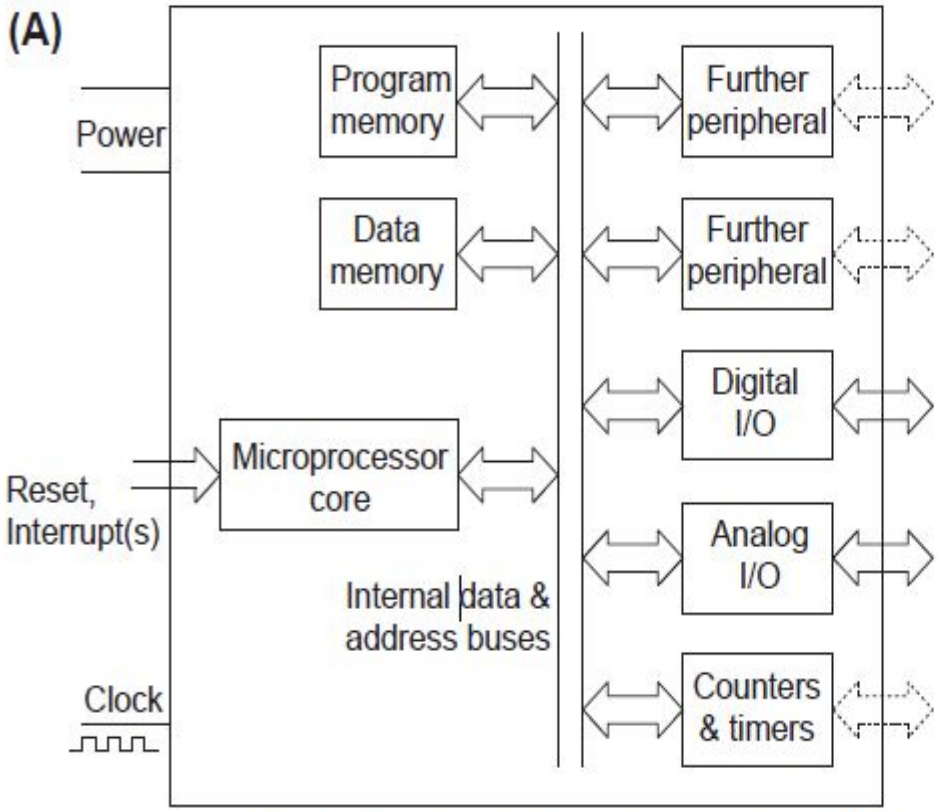
An Example Embedded control System

- Closed loop systems built around fast microcontrollers can also allow advanced control of systems which were previously thought uncontrollable.
- For example, the Segway personal transporter shown in fig uses sensitive gyroscopes to ensure that the standing platform always remains horizontal.
- If weight shifts forward (and the gyroscope shows an imbalance), then the motor in the wheels moves forwards marginally to compensate and stops moving when a horizontal position has been achieved again.
- The microcontroller, sensors, and actuators inside read and compute so rapidly that this process can be performed faster than a human's motion can displace their weight, so the controller can always ensure that the platform stays stable.

Some Computer Essentials



The Microcontroller

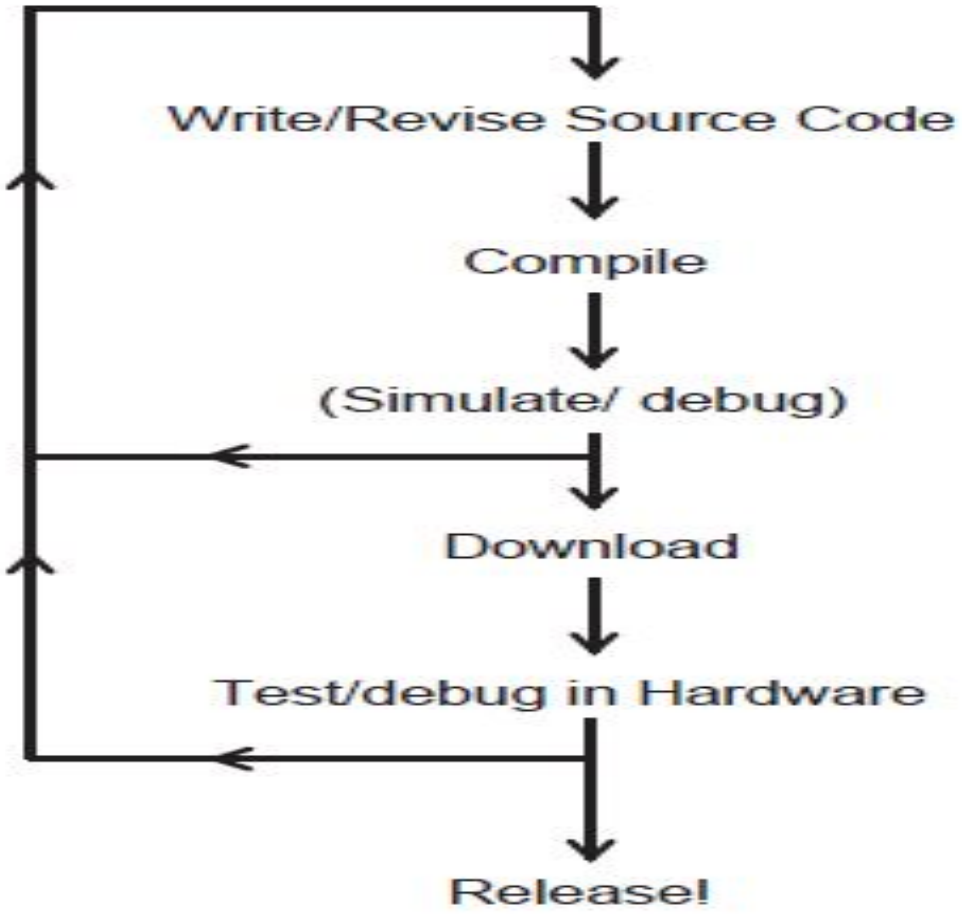


Generic features: core + memory + peripherals

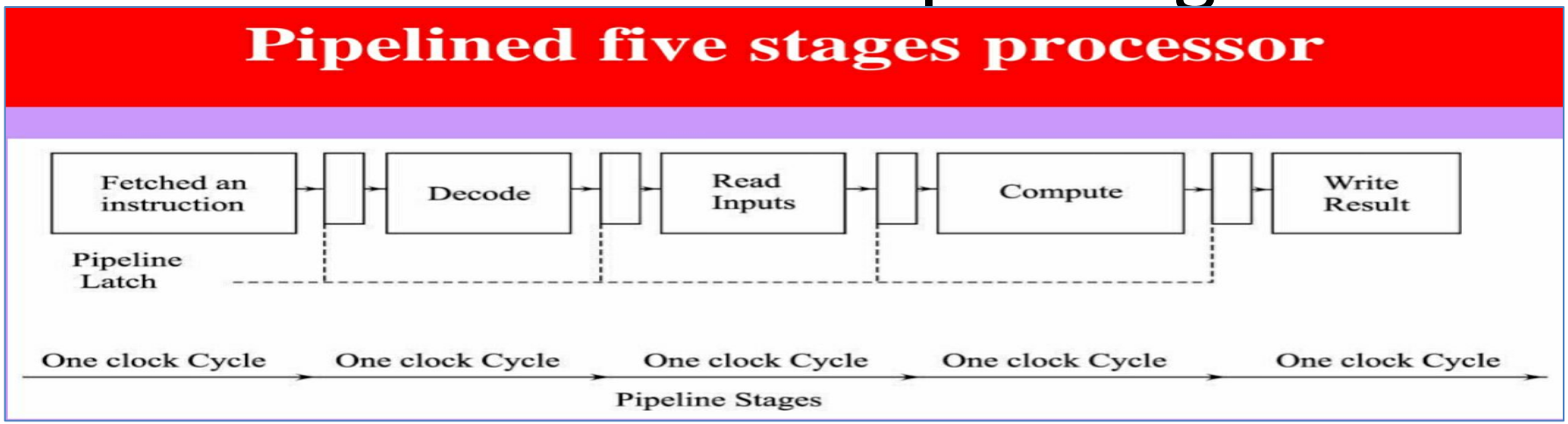
(B)

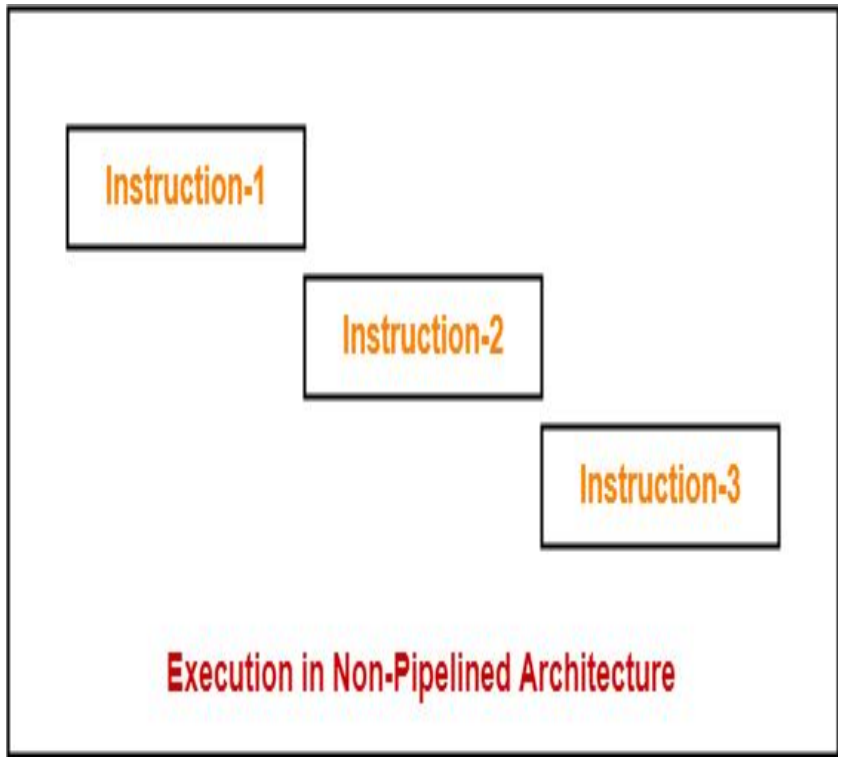
an actual microcontroller, the LPC1768

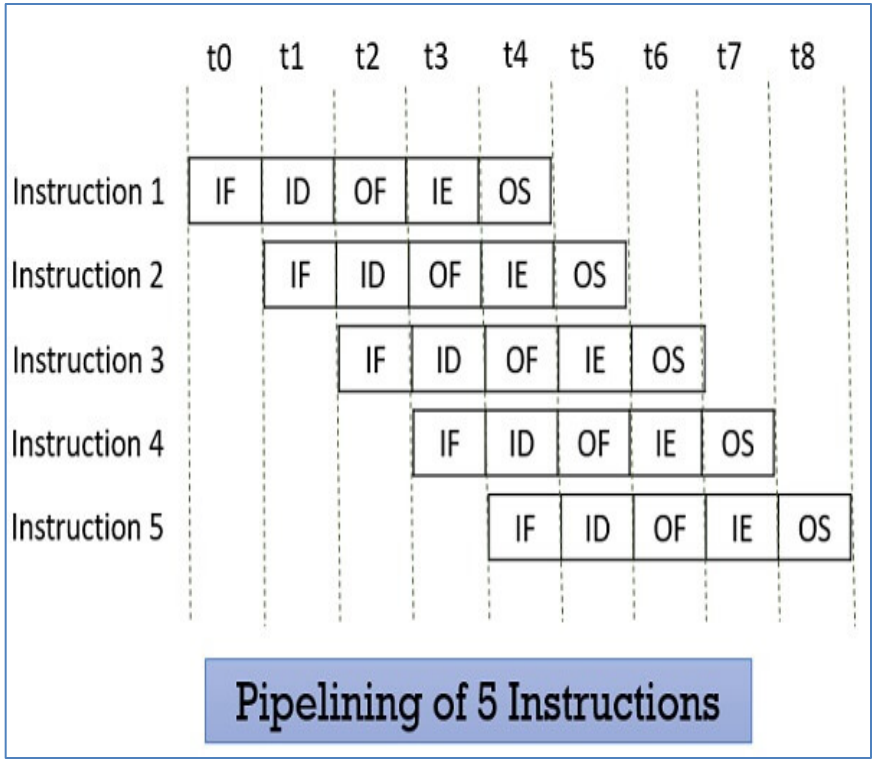
The Development Cycle



Instruction Pipelining







The World of ARM

- Advanced RISC Machines-ARM
- In the earlier days of microprocessor development, designers were trying to make the instruction set as advanced and sophisticated as possible.
- The price they were paying was that this was also making the computer hardware more complex, expensive, and slower. Such a microprocessor is called a Complex Instruction Set Computer (CISC)
- Another approach to CPU design is therefore to insist on keeping things simple and have a limited instruction set. This leads to the RISC approached the Reduced Instruction Set Computer.

The World of ARM

- A simple RISC CPU can execute code fast, but it may need to execute more instructions to complete a given task(memory efficiency)
- One characteristic of the RISC approach is that each instruction is contained within a single binary word. That word must hold all information necessary, including the instruction code itself, as well as any address or data information also needed.
- A further characteristic, an outcome of the simplicity of the approach, is that every instruction normally takes the same amount of time to execute.
- A good example is pipelining - as one instruction is being executed, the next is already being fetched from memory. It's easy to do this with a RISC architecture, where all (or most) instructions take the same amount of time to complete.
- An interesting subplot to the RISC concept is the fact that, due to its simplicity, RISC designs tend to lead to low power consumption and battery power is less and helps to explain why ARM products find their way into so many mobile phones and tablets.

CISC	RISC
Emphasis on hardware	Emphasis on software
Multiple instruction sizes and formats	Instructions of same set with few formats
Less registers	Uses more registers
More addressing modes	Fewer addressing modes
Extensive use of microprogramming	Complexity in compiler
Instructions take a varying amount of cycle time	Instructions take one cycle time
Pipelining is difficult	Pipelining is easy

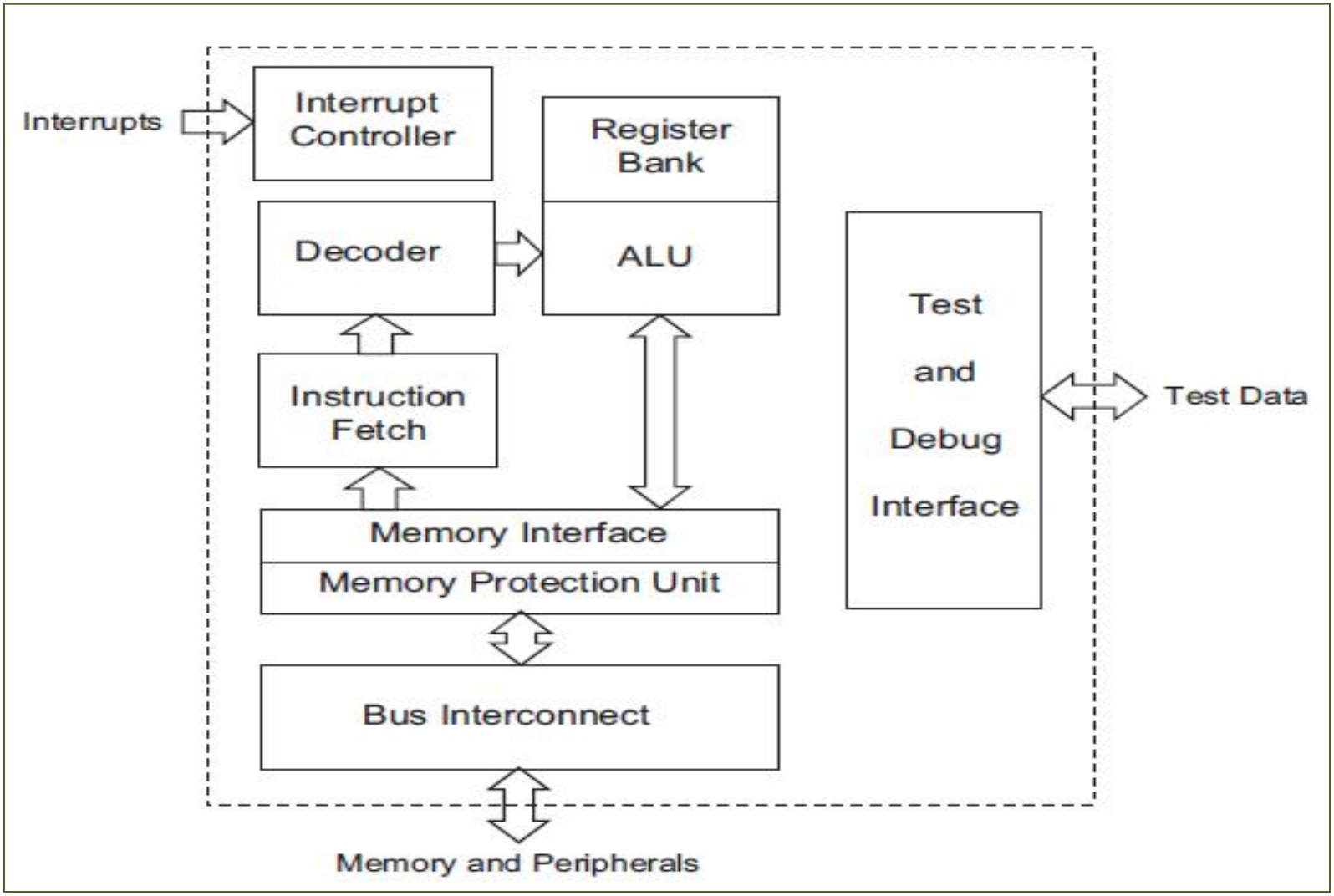
Table 1. ARM7TDMI-S and Cortex-M3 comparison (100MHz frequency on TSMC 0.18G)

Features	ARM7TDMI-S	Cortex-M3
Architecture	ARMv4T (von Neumann)	ARMv7-M (Harvard)
ISA Support	Thumb / ARM	Thumb / Thumb-2
Pipeline	3-Stage	3-Stage + branch speculation
Interrupts	FIQ / IRQ	NMI + 1 to 240 Physical Interrupts
Interrupt Latency	24-42 Cycles	12 Cycles
Sleep Modes	None	Integrated
Memory Protection	None	8 region Memory Protection Unit
Dhrystone	0.95 DMIPS/MHz (ARM mode)	1.25 DMIPS/MHz
Power Consumption	0.28mW/MHz	0.19mW/MHz
Area	0.62mm2 (Core Only)	0.86mm2 (Core & Peripherals)*

Comparison of ARM Cortex A, M & R CPU

Parameters	Cortex A	Cortex R	Cortex M
Performance	❖ Highest	❖ Very Good	❖ Medium
Response Time	❖ Very Good	❖ Best	❖ Medium
Power Consumption	❖ 80µW/MHz	❖ 120µW/MHz	❖ 8µW/MHz
Processor	❖ Application Based	❖ RTOS based	❖ Embedded System based
Pipeline	❖ Long Pipeline	❖ Medium Pipeline	❖ Short Pipeline
Clock	❖ High	❖ High	❖ Less relatively
Memory	❖ Cache Memory with more size.	❖ Cache Memory + Tightly Coupled Memory	❖ Cache Memory with less size.
ISA	❖ ARM	❖ ARM	❖ Thumb

The Cortex Core



overview

• An embedded system contains one or more tiny computers, which control it and give it a sense of intelligence.

• The embedded computer usually takes the form of a microcontroller, which combines microprocessor core, memory, and peripherals.

•Embedded system design combines hardware (electronic, electrical, and electromechanical) and software (program) design.

•The embedded microcontroller has an instruction set. It is the ultimate goal of the programmer to develop code which is made up of instructions from this instruction set.

•Most programming is done in a HLL, with a compiler being used to convert that program into the binary code, drawn from the instruction set and recognized by the microcontroller.

•ARM has developed a range of effective microprocessor and microcontroller designs, widely applied in embedded systems.

Quiz

1. Explain the following acronyms: IC, ALU, CPU.
2. Describe an embedded system in less than 100 words.
3. What are the differences between a microprocessor and a microcontroller?
4. What range of numbers can be represented by a 16-bit ALU?
5. What is a “bus” in the context of embedded systems and describe two types of busses that might be found in an embedded system?
6. Describe the term “instruction set” and explain how use of the instruction set differs for high- and low-level programming

Cortex-M processor architecture

- The LPC1768/66/65/64 are ARM Cortex-M3 based microcontrollers for embedded applications featuring a high level of integration and low power consumption.
- The ARM Cortex-M3 is a next generation core that offers system enhancements such as enhanced debug features and a higher level of support block integration.
- The LPC1768/66/65/64 operate at CPU frequencies of up to 100 MHz.
- The ARM Cortex-M3 CPU incorporates a 3-stage pipeline and uses a Harvard architecture with separate local instruction and data buses as well as a third bus for peripherals.
- The ARM Cortex-M3 CPU also includes an internal prefetch unit that supports speculative branching.
- The peripheral complement of the LPC1768/66/65/64 includes up to 512 kB of flash memory, up to 64 kB of data memory,
- Ethernet MAC, USB Device/Host/OTG interface, 8-channel general purpose DMA controller, 4 UARTs, 2 CAN channels, 2 SSP controllers, SPI interface, 3 I2C-bus interfaces, 2-input plus 2-output I2S-bus interface, 8-channel 12-bit ADC, 10-bit DAC, motor control PWM, Quadrature Encoder interface, 4 general purpose timers, 6-output general purpose PWM, ultra-low power Real-Time Clock (RTC) with separate battery supply, and up to 70 general purpose I/O pins.

Cortex-M processor architecture

- **Features**

- Running at frequencies of up to 100 MHz.
- A Memory Protection Unit (MPU) supporting eight regions is included.
- ARM Cortex-M3 built-in Nested Vectored Interrupt Controller (NVIC).
- Up to 512 kB on-chip flash programming memory.
- In-System Programming (ISP) and In-Application Programming (IAP) via on-chip boot loader software.(In-System Programming means that the device can be programmed in the circuit by using an utility such as the ULINK Debug Adapter. In Application Programming means that the application itself can re-program the on-chip Flash ROM.)
- On-chip SRAM
- Eight channel General Purpose DMA controller
- Split Advanced Peripheral Bus (APB)and Advanced High performance Bus (AHB)bus
- Serial interfaces:
    - Ethernet MAC
    - USB 2.0
    - Four and RS-485 support.
    - CAN 2.0B controller with two channels.
    - SPI controller with synchronous, serial, full duplex
    - Two SSP controllers with FIFO and multi-protocol capabilities.
    - Two I_2C-bus interfaces supporting fast mode with a data rate of 400 kbits/s
    - One I_2C-bus interface supporting full I_2C-bus specification and fast mode plus with a data rate of 1 Mbit/s with multiple address recognition and monitor mode.
    - I_2S (Inter-IC Sound) interface for digital audio input or output, with fractional rate control.

Cortex-M processor architecture

•**Features**

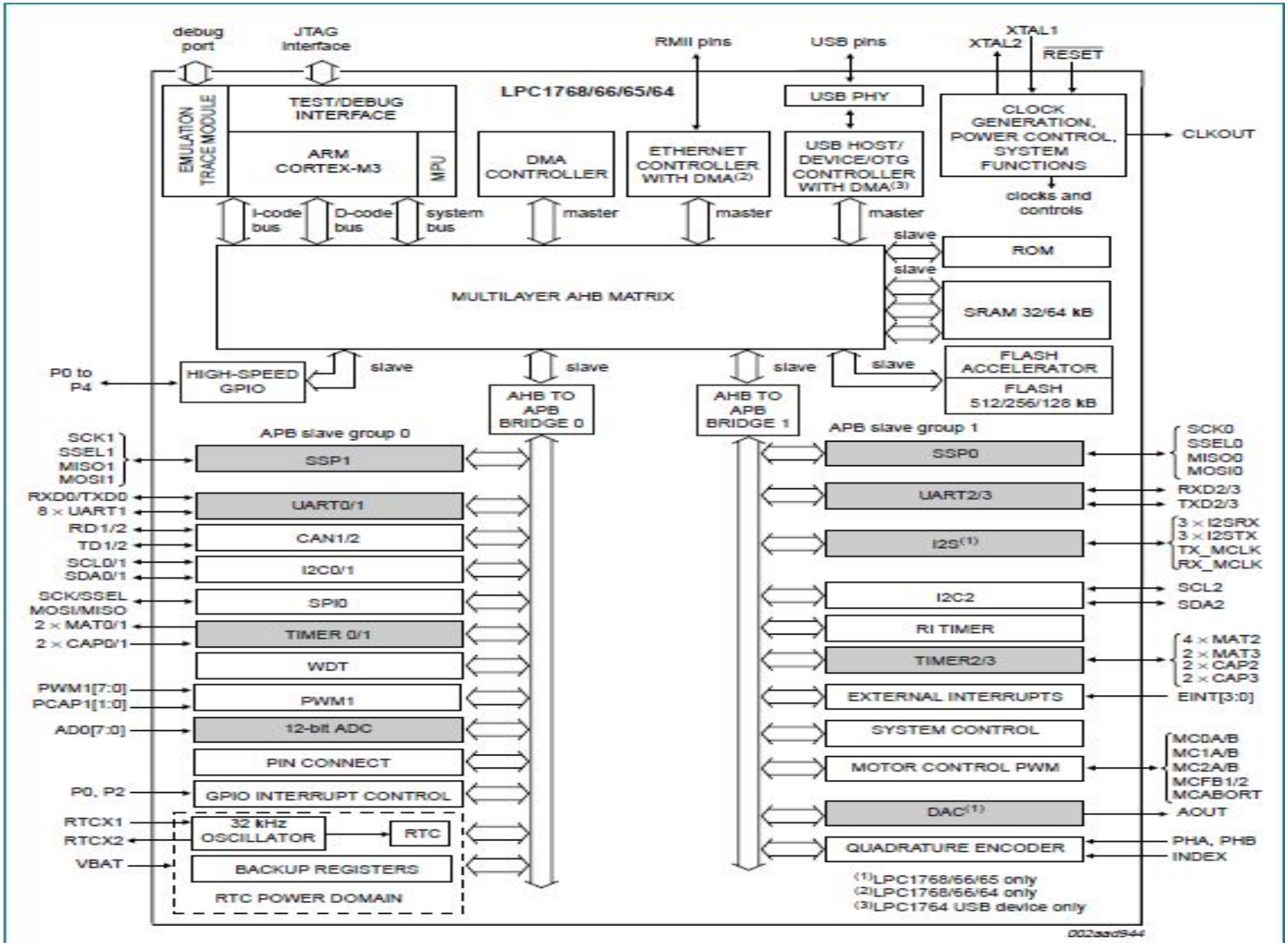
•Other peripherals
•70 General Purpose I/O (GPIO) pins.
•12-bit Analog-to-Digital Converter (ADC)
• 10-bit Digital-to-Analog Converter (DAC)
• Four general purpose timers/counters
• One motor control PWM with support for three-phase motor control.
• Quadrature encoder interface that can monitor one external quadrature encoder.
• One standard PWM/timer block with external count input.
• RTC with a separate **power** domain and dedicated RTC oscillator.
• Watchdog Timer (WDT) resets the microcontroller within a reasonable amount of time if it enters an erroneous states
•System tick timer, including an external clock input option.
• Repetitive interrupt timer provides programmable and repeating timed interrupts.
• Each peripheral has its own clock divider for further power savings

Cortex-M processor architecture

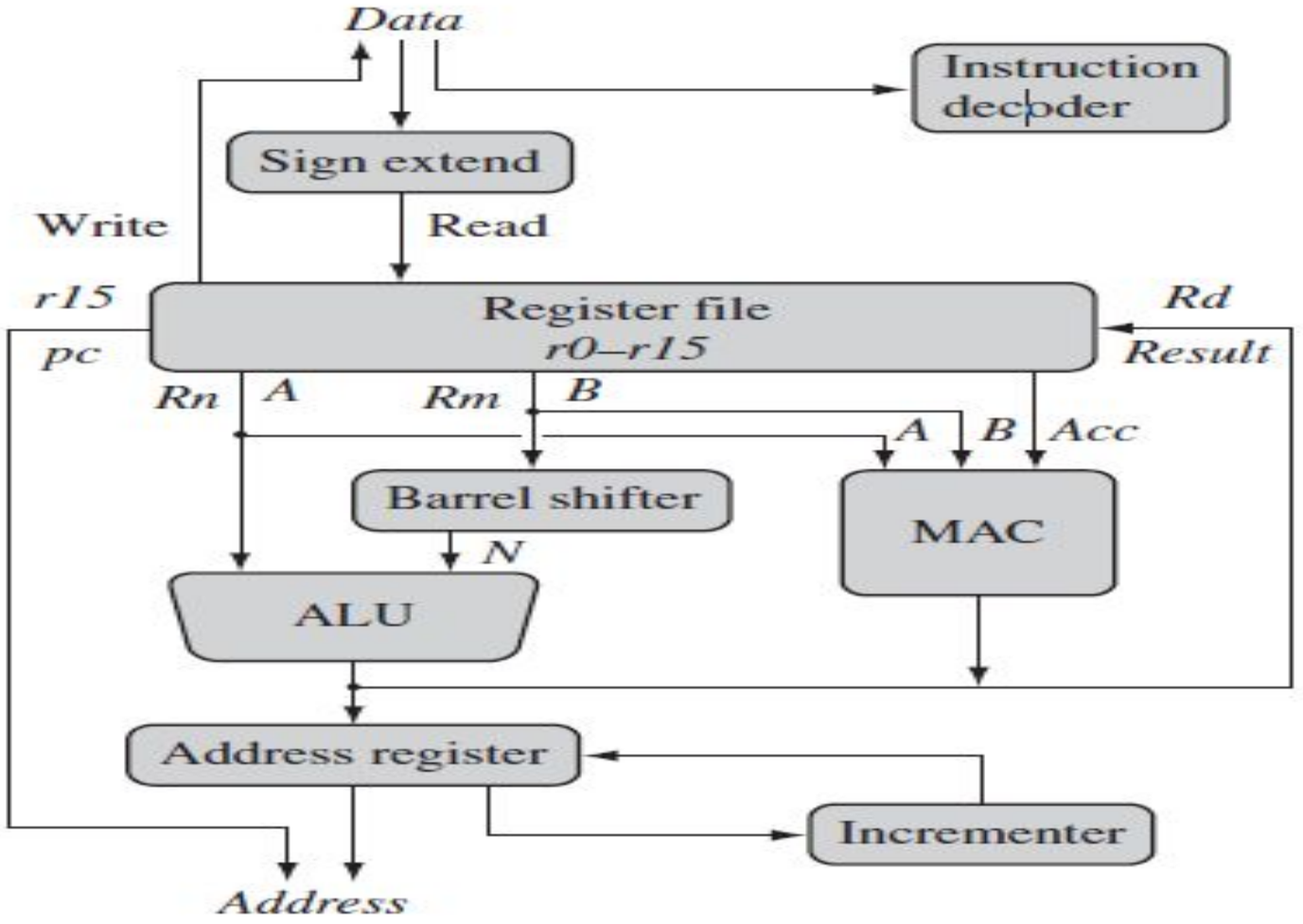
•**Features**

•Standard JTAG test/debug interface
• Emulation trace module enables non-intrusive, high-speed real-time tracing of
•instruction execution.
•Integrated PMU (Power Management Unit) to minimize power consumption
•Four reduced power modes: Sleep, Deep-sleep, Power-down, and Deep power-down.
• Single 3.3 V power supply (2.4 V to 3.6 V).
• Four external interrupt inputs configurable as edge/level sensitive.
• Non-maskable Interrupt (NMI) input.
• Clock output function.
• The Wakeup Interrupt Controller (WIC) allows the CPU to automatically wake up
• Processor wake-up from Power-down mode via interrupts from various peripherals.
• Brownout detect with separate threshold for interrupt and forced reset.
• Power-On Reset (POR).
• Crystal oscillator with an operating range of 1 MHz to 25 MHz.
• 4 MHz internal RC oscillator optionally be used as a system clock.
• PLL allows CPU operation up to the maximum CPU rate without the need for a
high-frequency crystal.
• USB PLL for added flexibility.
• Code Read Protection (CRP) with different security levels.
• Available as 100-pin LQFP package (14 ´ 14 ´ 1.4 mm).

Cortex-M processor architecture



ARM core

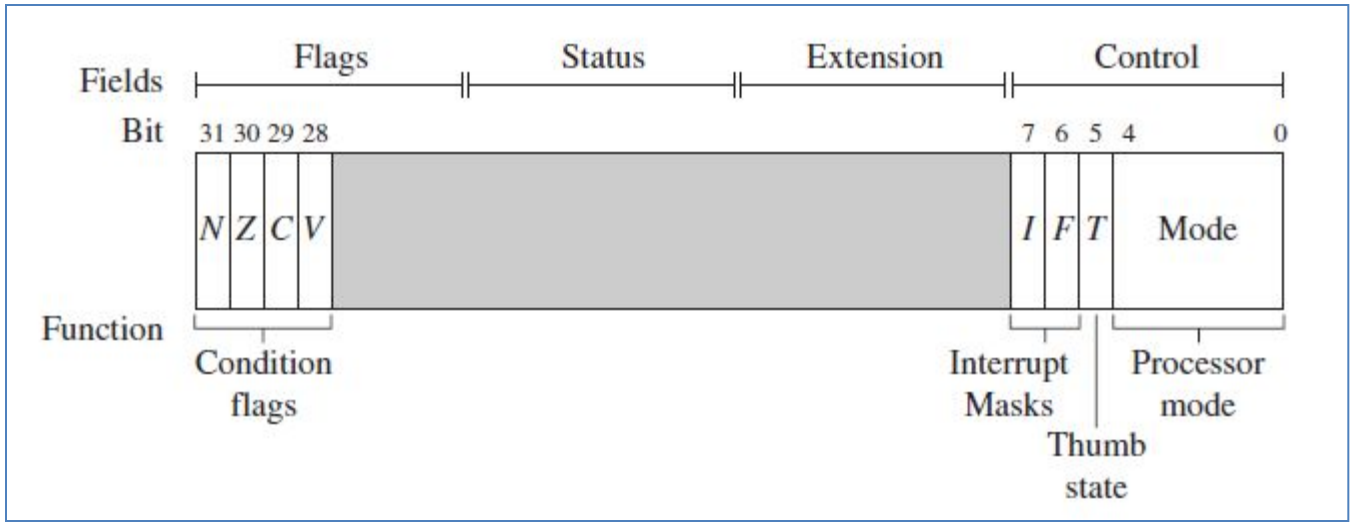

Data
Instruction decoder
Sign extend
Write
Read
r15
pc
Register file
r0–r15
Rd
Result
Rn
A
Rm
B
A
B
Acc
Barrel shifter
MAC
N
ALU
Address register
Incrementer
Address

Registers

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 sp
r14 lr
r15 pc

cpsr
-

Current Program Status Register



The cpsr is divided into four fields, each 8 bits wide: flags, status, extension, and control.

Processor Modes

There are seven processor modes in total: six privileged modes (abort, fast interrupt request, interrupt request, supervisor, system, and undefined) and one non-privileged mode (user).

The processor enters abort mode when there is a failed attempt to access memory. Fast interrupt request and interrupt request modes correspond to the two interrupt levels available on the ARM processor. Supervisor mode is the mode that the processor is in after reset and is generally the mode that an operating system kernel operates in. System mode is a special version of user mode that allows full read-write access to the cpsr. Undefined mode is used when the processor encounters an instruction that is undefined or not supported by the implementation. User mode is used for programs and applications.

Processor mode.

Mode	Abbreviation	Privileged	Mode[4:0]
Abort	abt	yes	10111
Fast interrupt request	fiq	yes	10001
Interrupt request	irq	yes	10010
Supervisor	svc	yes	10011
System	sys	yes	11111
Undefined	und	yes	11011
User	usr	no	10000

Processor Modes

Processor Modes

- The ARM has seven basic operating modes:
  - Each mode has access to:
    - Its own stack space and a different subset of registers
  - Some operations can only be carried out in a privileged mode

Exception modes

Mode	Description	
Supervisor (SVC)	Entered on reset and when a Software Interrupt instruction (SWI) is executed	Privileged modes
FIQ	Entered when a high priority (fast) interrupt is raised	
IRQ	Entered when a low priority (normal) interrupt is raised	
Abort	Used to handle memory access violations	
Undef	Used to handle undefined instructions	
System	Privileged mode using the same registers as User mode	
User	Mode under which most Applications / OS tasks run	Unprivileged mode

Processor Modes

Operating Modes

User mode:

- Normal program execution mode
- System resources unavailable
- Mode changed by exception only

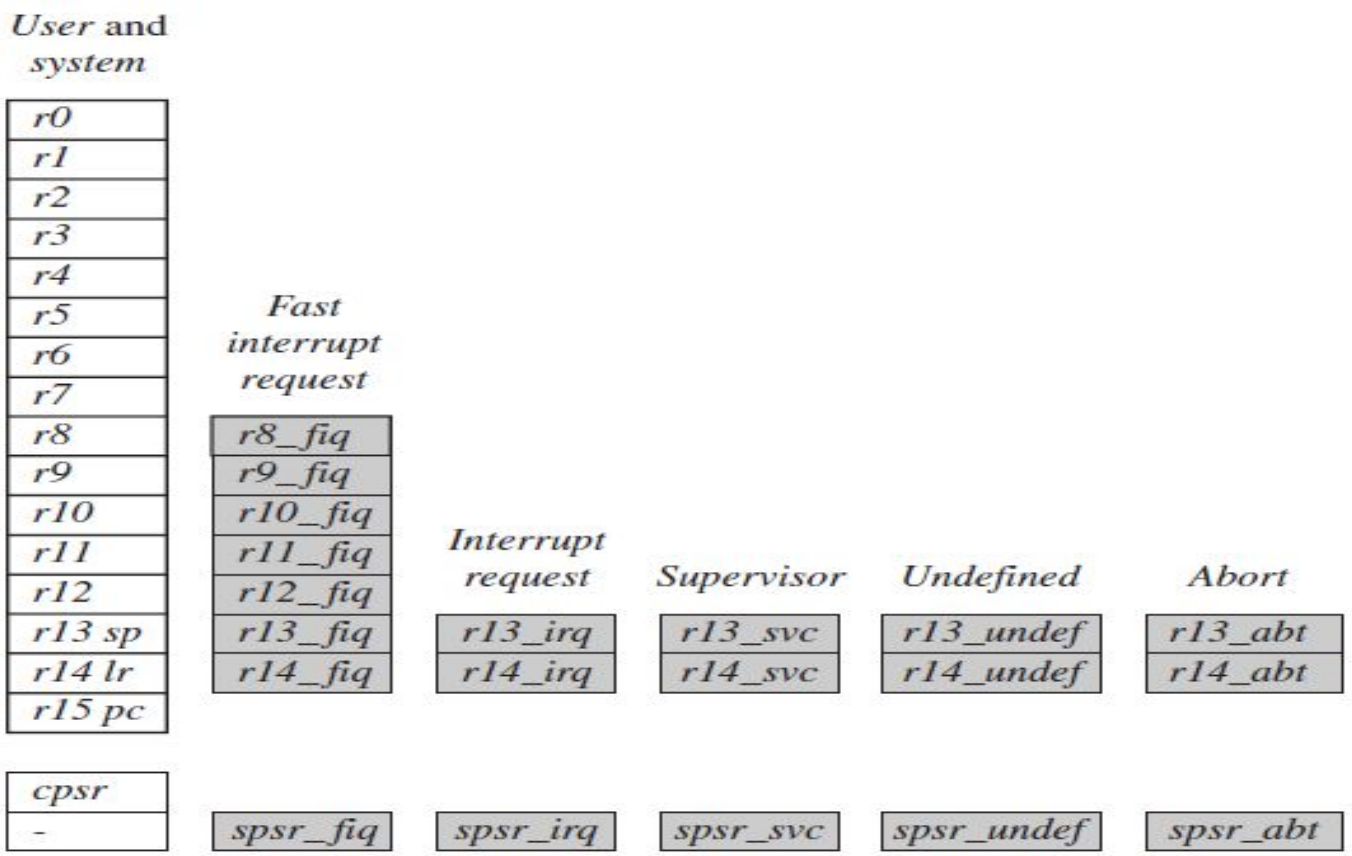
Exception modes:

- Entered upon exception
- Full access to system resources
- Mode changed freely

Modes (Thread out of reset)		Operations (privilege out of reset)	Stacks (Main out of reset)
	Handler - An exception is being processed	Privileged execution Full control	Main Stack Used by OS and Exceptions
	Thread - no exception is being processed - Normal code is executing	Privileged/Unprivileged	Main/Process

Banked Registers

37 registers in the register file. Of those, 20 registers are hidden from a program at different times.



State and Instruction Sets

The state of the core determines which instruction set is being executed. There are three instruction sets: ARM, Thumb, and Jazelle

ARM and Thumb instruction set features.

	ARM (*cpsr* $T = 0$)	Thumb (*cpsr* $T = 1$)
Instruction size	32-bit	16-bit
Core instructions	58	30
Conditional execution[a]	most	only branch instructions
Data processing instructions	access to barrel shifter and ALU	separate barrel shifter and ALU instructions
Program status register	read-write in privileged mode	no direct access
Register usage	15 general-purpose registers +*pc*	8 general-purpose registers +7 high registers +*pc*

Jazelle instruction set features.

	Jazelle (*cpsr* $T = 0, J = 1$)
Instruction size	8-bit
Core instructions	Over 60% of the Java bytecodes are implemented in hardware; the rest of the codes are implemented in software.

Memory map

- The LPC17xx incorporates several distinct memory regions, shown in the following figures.
- Figure below shows the overall map of the entire address space from the user program viewpoint following reset.
- The interrupt vector area supports address remapping.
- The AHB peripheral area is 2 MB in size, and is divided to allow for up to 128 peripherals.
- The APB peripheral area is 1 MB in size and is divided to allow for up to 64 peripherals.
- Each peripheral of either type is allocated 16 kB of space. This allows simplifying the address decoding for each peripheral.

Memory map

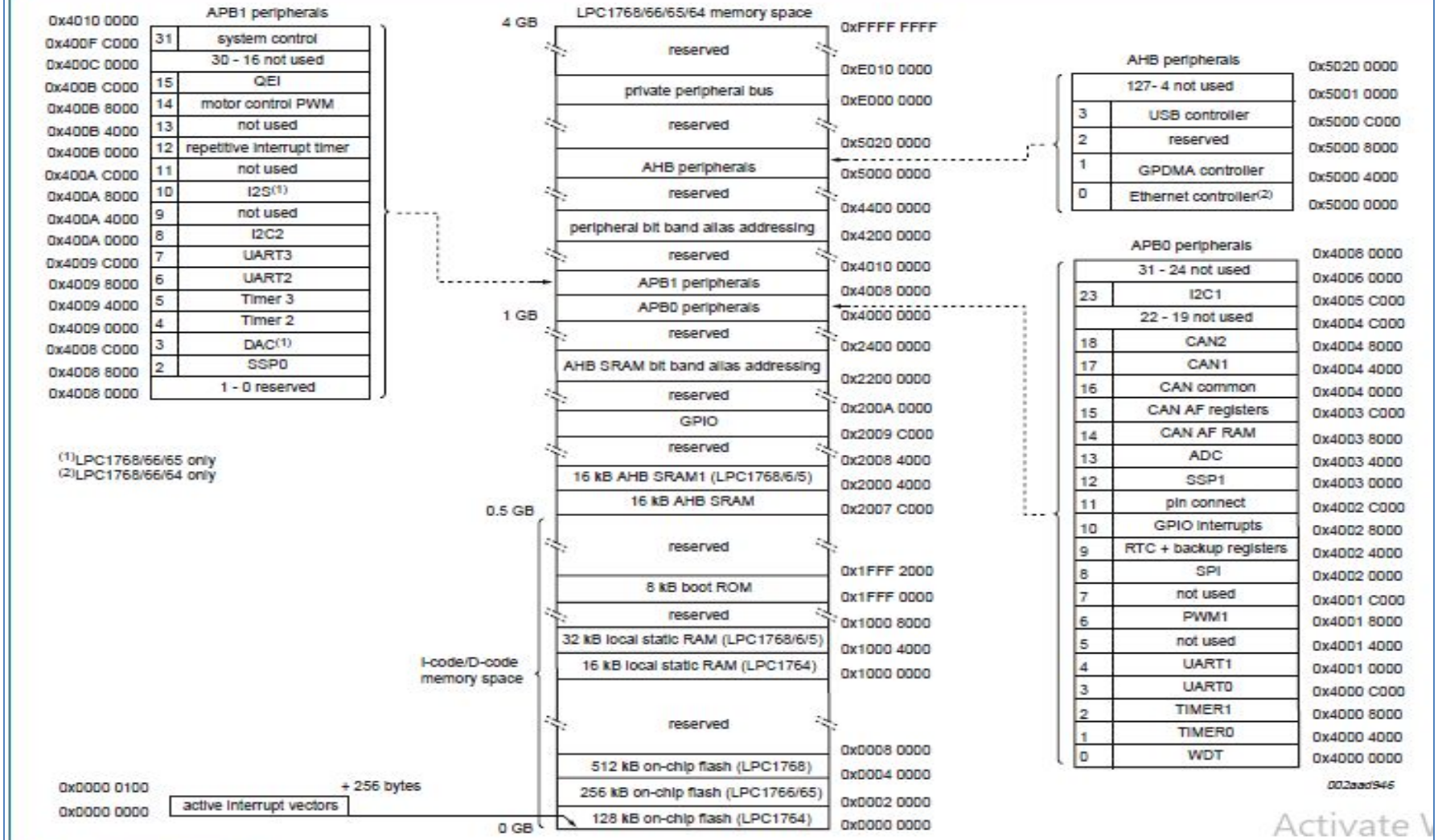
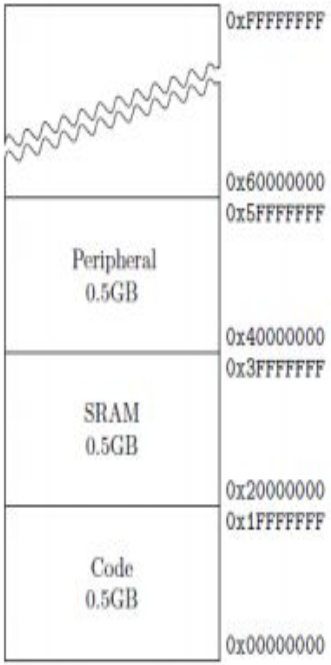


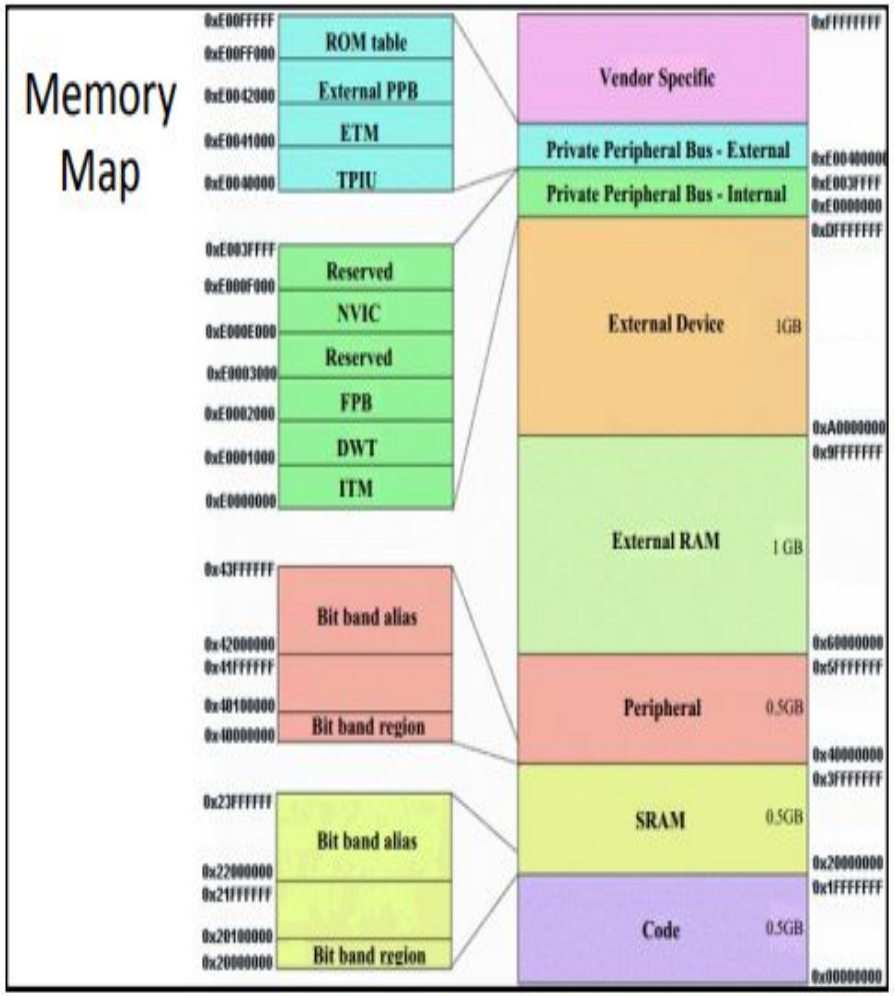
Fig 3. LPC1768/66/65/64 memory map

Memory map

Cortex-M3 Memory Address Space

- ARM Cortex-M3 processor has a single 4 GB address space
- The **SRAM and Peripheral** areas are accessed through the System bus
- The **"Code" region** is accessed through the ICode (instructions) and DCode (constant data) buses





I/O PINS

Fast general purpose parallel I/O

Device pins that are not connected to a specific peripheral function are controlled by the GPIO registers.

Pins may be dynamically configured as inputs or outputs.

Separate registers allow setting or clearing any number of outputs simultaneously.

The value of the output register may be read back as well as the current state of the port pins.

LPC1768/66/65/64 use accelerated GPIO functions:

- **GPIO registers are a dedicated AHB peripheral and are accessed through the AHB** multilayer bus so that the fastest possible I/O timing can be achieved.
- **Mask registers allow treating sets of port bits as a group, leaving other bits** unchanged.
- **All GPIO registers are byte and half-word addressable.**
- **Entire port value can be written in one instruction.**

Additionally, any pin on PORT0 and PORT2 (total of 42 pins) providing a digital function can be programmed to generate an interrupt on a rising edge, a falling edge, or both.

ADC

12-bit ADC

The LPC1768/66/65/64 contain one ADC. It is a single 12-bit successive approximation ADC with eight channels and DMA support.

Features

- 12-bit successive approximation ADC.
- Input multiplexing among 8 pins.
- Power-down mode.
- Measurement range VREFN to Vi(VREFP).
- 12-bit conversion rate: 1 MHz.
- Individual channels can be selected for conversion.
- Burst conversion mode for single or multiple inputs.
- Optional conversion on transition of input pin or Timer Match signal.
- Individual result registers for each ADC channel to reduce interrupt overhead.
- DMA support.

I2C-bus

I2C-bus serial I/O controllers

The LPC1768/66/65/64 each contain three I2C-bus controllers.

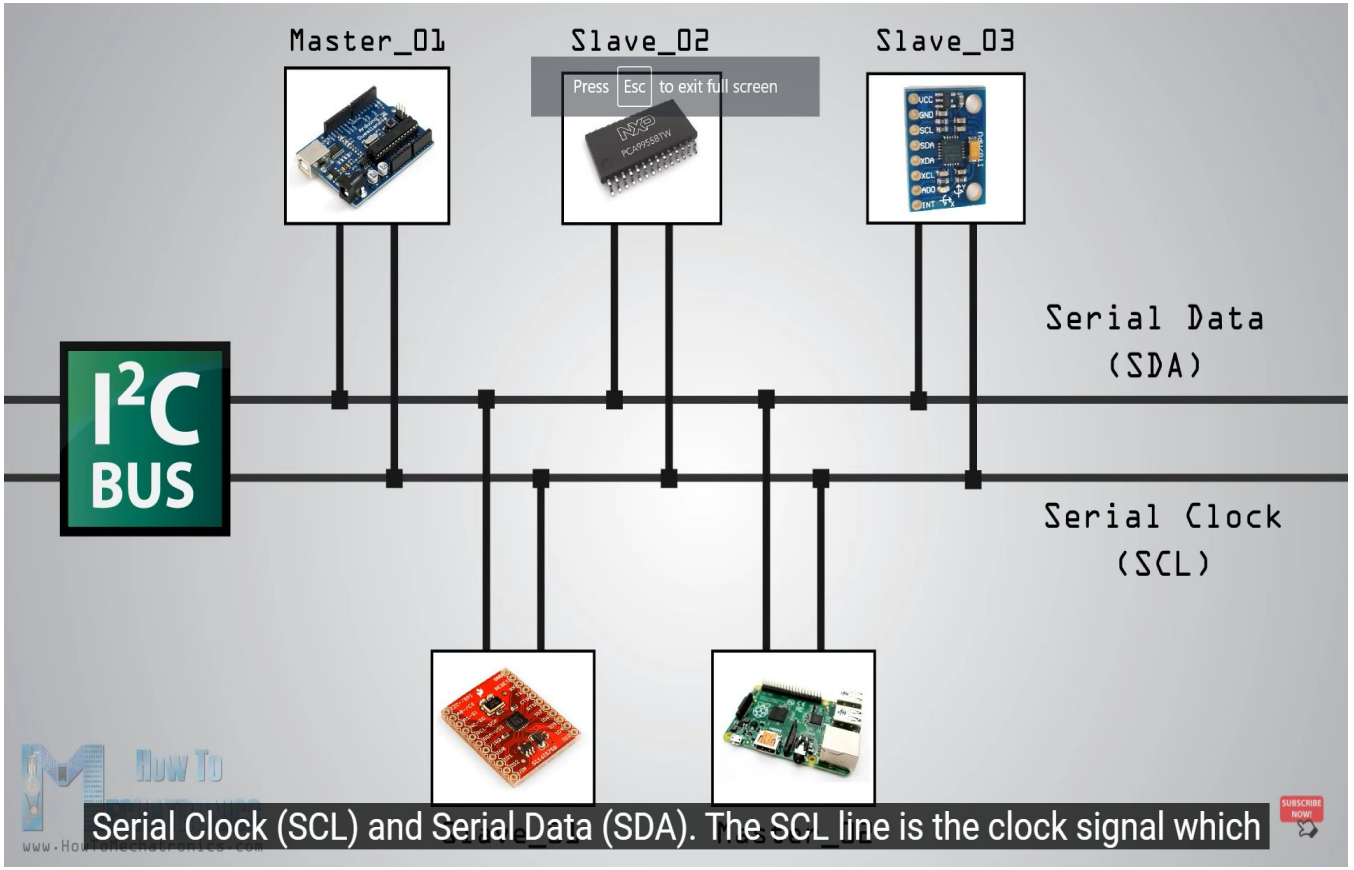
The I2C-bus is bidirectional for inter-IC control using only two wires: a Serial Clock line (SCL) and a Serial Data Line (SDA).

Each device is recognized by a unique address and can operate as either a receiver-only device (e.g., an LCD driver) or a transmitter with the capability to both receive and send information (such as memory).

Transmitters and/or receivers can operate in either master or slave mode, depending on whether the chip has to initiate a data transfer or is only addressed.

The I2C is a multi-master bus and can be controlled by more than one bus master connected to it.

I2C-bus



https://www.youtube.com/watch?v=RPHP4fAisz8

I2C-bus

Features

- **I2C0 is a standard I2C compliant bus interface with open-drain pins. I2C0 also** supports Fast mode plus with bit rates up to 1 Mbit/s.
- **I2C1 and I2C2 use standard I/O pins with bit rates of up to 400 kbit/s (Fast I2C-bus).**
- **Easy to configure as master, slave, or master/slave.**
- **Programmable clocks allow versatile rate control.**
- **Bidirectional data transfer between masters and slaves.**
- **Multi-master bus (no central master).**
- **Arbitration between simultaneously transmitting masters without corruption of serial** data on the bus.
- **Serial clock synchronization allows devices with different bit rates to communicate via** one serial bus.
- **Serial clock synchronization can be used as a handshake mechanism to suspend and** resume serial transfer.
- **The I2C-bus can be used for test and diagnostic purposes.**
- **All I2C-bus controllers support multiple address recognition and a bus monitor mode.**

PWM

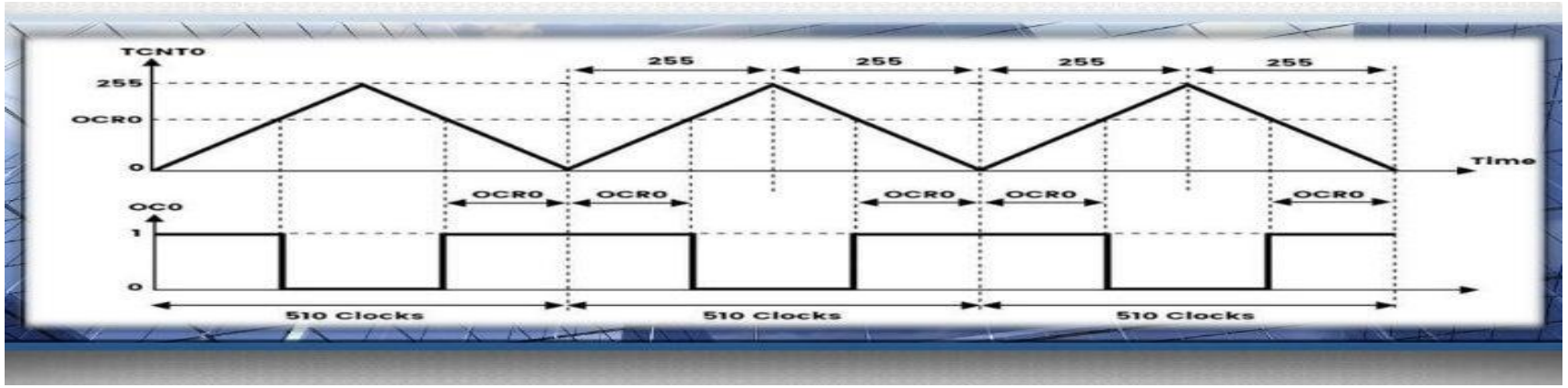
The PWM is based on the standard Timer block and inherits all of its features, although only the PWM function is pinned out on the LPC1768/66/65/64.

The Timer is designed to count cycles of the system derived clock and optionally switch pins, generate interrupts or perform other actions when specified timer values occur, based on seven match registers.

The PWM function is in addition to these features, and is based on match register events.

The ability to separately control rising and falling edge locations allows the PWM to be used for more applications.

For instance, multi-phase motor control typically requires three non-overlapping PWM outputs with individual control of all three pulse widths and Positions.



https://www.youtube.com/watch?v=2XjqS1clY_E

https://www.youtube.com/watch?v=5nwNKPs2gco

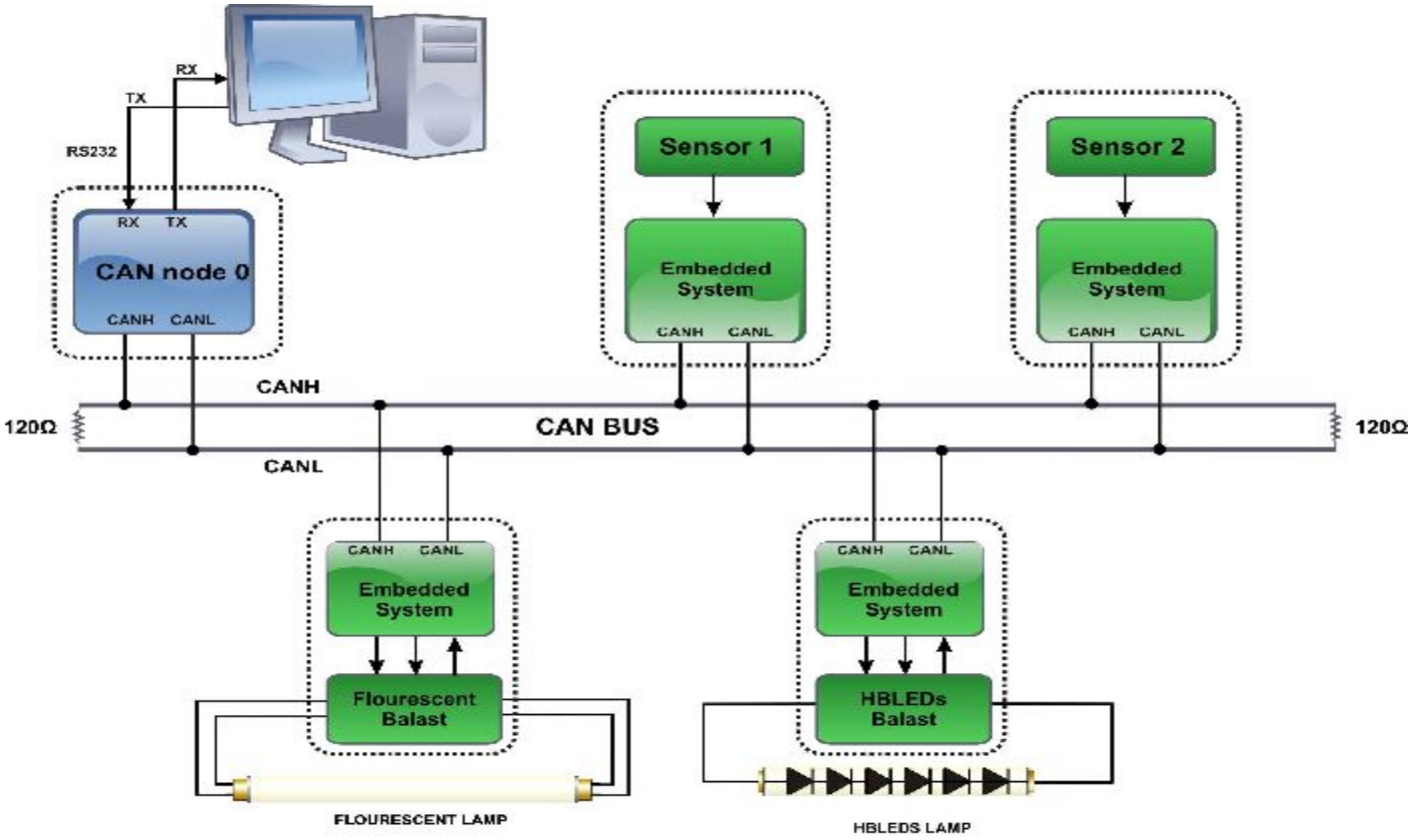
PWM

Two match registers can be used to provide a single edge controlled PWM output.
One match register (PWMMR0) controls the PWM cycle rate, by resetting the count upon match.
The other match register controls the PWM edge position.
Three match registers can be used to provide a PWM output with both edges controlled.
Again, the PWMMR0 match register controls the PWM cycle rate.
The other match registers control the two PWM edge positions.
Additional double edge controlled PWM outputs require only two match registers each, since the repetition rate is the same for all PWM outputs.
With double edge controlled PWM outputs, specific match registers control the rising and falling edge of the output.
This allows both positive going PWM pulses (when the rising edge occurs prior to the falling edge), and negative going PWM pulses (when the falling edge occurs prior to the rising edge).

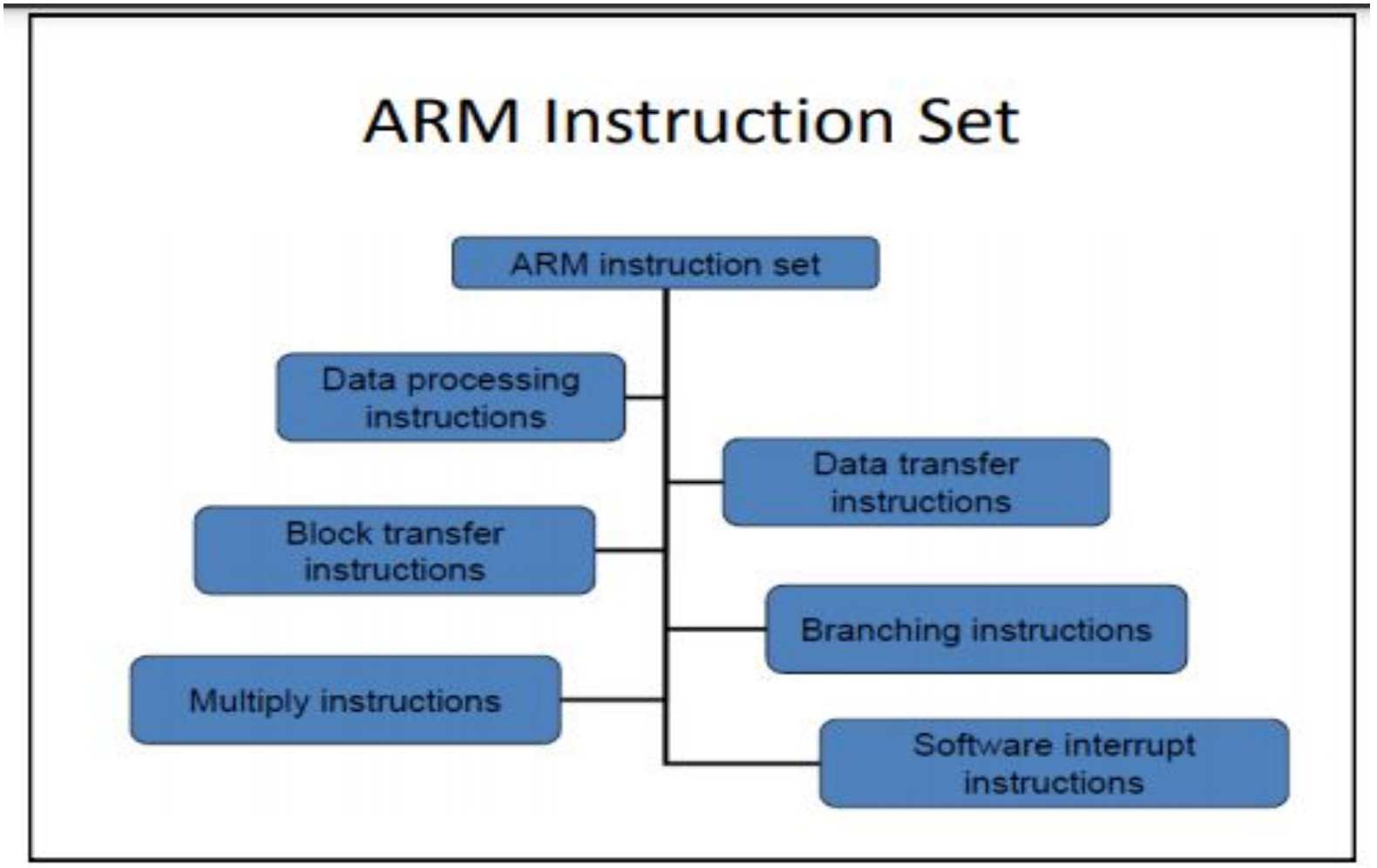
PWM

- **One PWM block with Counter or Timer operation (may use the peripheral clock or one** of the capture inputs as the clock source).
- **Seven match registers allow up to 6 single edge controlled or 3 double edge** controlled PWM outputs, or a mix of both types. The match registers also allow:
  - **Continuous operation with optional interrupt generation on match.**
  - **Stop timer on match with optional interrupt generation.**
  - **Reset timer on match with optional interrupt generation.**
- **Supports single edge controlled and/or double edge controlled PWM outputs. Single** edge controlled PWM outputs all go high at the beginning of each cycle unless the output is a constant low. Double edge controlled PWM outputs can have either edge occur at any position within a cycle. This allows for both positive going and negative going pulses.
- **Pulse period and width can be any number of timer counts. This allows complete** flexibility in the trade-off between resolution and repetition rate. All PWM outputs will occur at the same repetition rate.
- **Double edge controlled PWM outputs can be programmed to be either positive going** or negative going pulses.
- **Match register updates are synchronized with pulse outputs to prevent generation of** erroneous pulses. Software must ‘release’ new match values before they can become effective.
- **May be used as a standard 32-bit timer/counter**

CAN bus


RX
TX
RS232
RX TX
CAN node 0
CANH CANL
Sensor 1
Embedded System
CANH CANL
Sensor 2
Embedded System
CANH CANL
CANH
120Ω
CAN BUS
120Ω
CANL
CANH CANL
Embedded System
Flourescent Balast
CANH CANL
Embedded System
HBLEDs Balast
FLOURESCENT LAMP
HBLEDS LAMP

Instruction set



Instruction set

Data Processing Instructions

- Arithmetic and logical operations
- 3-address format:
  - Two 32-bit operands (op1 is register, op2 is register or immediate)
  - 32-bit result placed in a register
- Barrel shifter for op2 allows full 32-bit shift within instruction cycle

Data Processing Instructions

- Arithmetic operations:
  - ADD, ADDC, SUB, SUBC, RSB, RSC
- Bit-wise logical operations:
  - AND, EOR, ORR, BIC
- Register movement operations:
  - MOV, MVN
- Comparison operations:
  - TST, TEQ, CMP, CMN

Instruction Set

Data Processing Instructions

Conditional codes

+

Data processing instructions

+

Barrel shifter

=

Powerful tools for efficient coded programs

Data Processing Instructions

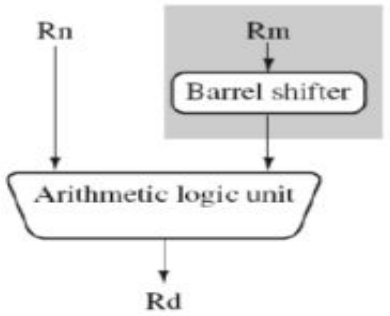
e.g.:

if (z==1) R1=R2+(R3*4)

compiles to

EQADDS R1,R2,R3, LSL #2

(SINGLE INSTRUCTION !)



Instruction Set

Multiply Instructions

- Integer multiplication (32-bit result)
- Long integer multiplication (64-bit result)
- Built in Multiply Accumulate Unit (MAC)
- Multiply and accumulate instructions add product to running total

Multiply Instructions

MUL	Multiply	32-bit result
MULA	Multiply accumulate	32-bit result
UMULL	Unsigned multiply	64-bit result
UMLAL	Unsigned multiply accumulate	64-bit result
SMULL	Signed multiply	64-bit result
SMLAL	Signed multiply accumulate	64-bit result

Instruction Set

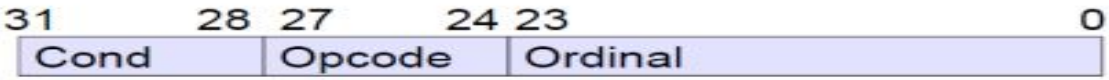
Data Transfer Instructions

- Load/store instructions
- Used to move signed and unsigned
- Word, Half Word and Byte to and from registers
- Can be used to load PC (if target address is beyond branch instruction range)

LDR	Load Word	STR	Store Word
LDRH	Load Half Word	STRH	Store Half Word
LDRSH	Load Signed Half Word	STRSH	Store Signed Half Word
LDRB	Load Byte	STRB	Store Byte
LDRSB	Load Signed Byte	STRSB	Store Signed Byte

Software Interrupt

- *SWI* instruction
  - Forces CPU into supervisor mode
  - Usage: SWI #n

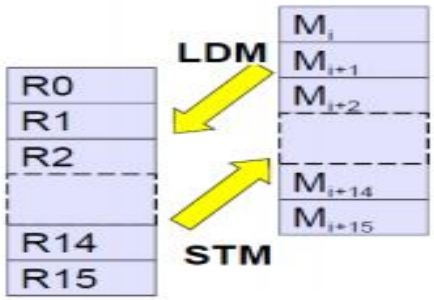


- Maximum 2^{24} calls
- Suitable for running privileged code and making OS calls

Instruction Set

Block Transfer Instructions

- Load/Store Multiple instructions (*LDM/STM*)
- Whole register bank or a subset copied to memory or restored with single instruction

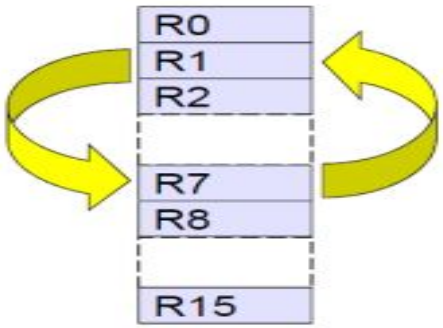


Swap Instruction

- Exchanges a word between registers
  - Two cycles

    but

    single atomic action
  - Support for RT semaphores



Instruction Set

Branching Instructions

- *Branch (B):*
  - jumps forwards/backwards up to 32 MB
- *Branch link (BL):*
  - same + saves (PC+4) in LR
- Suitable for function call/return
- Condition codes for conditional branches

Branching Instructions

Table A4-1 Branch instructions

Instruction	Usage	Range
B on page A6-40	Branch to target address	+/–1 MB
CBNZ, CBZ on page A6-52	Compare and Branch on Nonzero, Compare and Branch on Zero	0-126 B
BL on page A6-49	Call a subroutine	+/–16 MB
BLX (register) on page A6-50	Call a subroutine, optionally change instruction set	Any
BX on page A6-51	Branch to target address, change instruction set	Any
TBB, TBH on page A6-258	Table Branch (byte offsets)	0-510 B
	Table Branch (halfword offsets)	0-131070 B

Instruction Set

IF-THEN Instruction

- Another alternative to execute conditional code is the new 16-bit IF-THEN (IT) instruction
  - no change in program flow
  - no branching overhead
- Can use with 32-bit Thumb-2 instructions that do not support the 'S' suffix
- Example:

```
CMP R1, R2          ; If R1 = R2
IT EQ               ; execute next (1st)
                    ; instruction
ADDEQ R2, R1, R0    ; 1st instruction
```

- The conditional codes can be extended up to 4 instructions

Barrier instructions

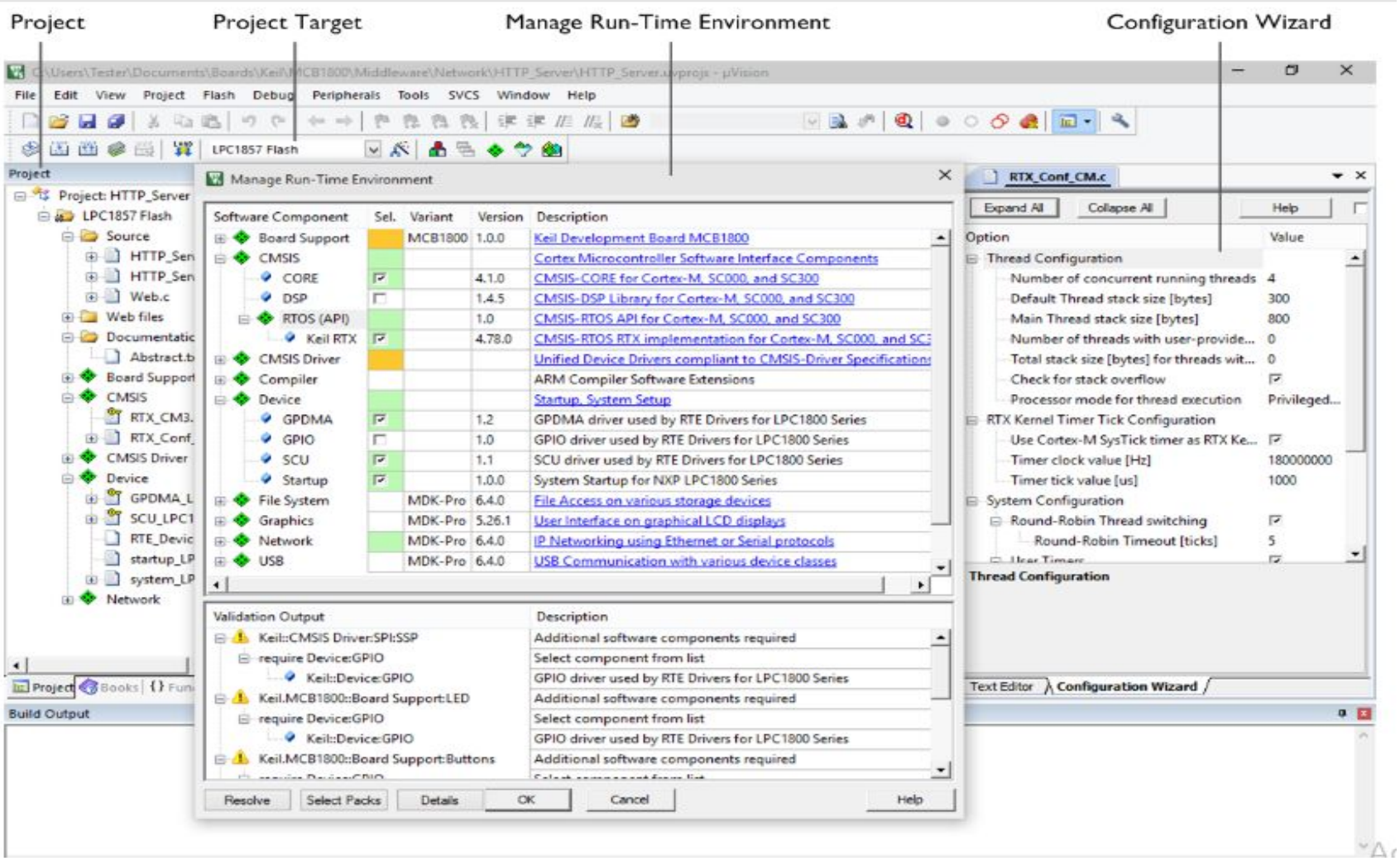
- Useful for multi-core & Self-modifying code

Instruction	Description
DMB	Data memory barrier; ensures that all memory accesses are completed before new memory access is committed
DSB	Data synchronization barrier; ensures that all memory accesses are completed before next instruction is executed
ISB	Instruction synchronization barrier; flushes the pipeline and ensures that all previous instructions are completed before executing new instructions

Keil IDE and Debugging tools

The µVision IDE combines project management, run-time environment, build facilities, source code editing, and program debugging in a single powerful environment.
µVision is easy-to-use and accelerates your embedded software development.
µVision supports multiple screens and allows you to create individual window layouts anywhere on the visual surface.
The µVision Debugger provides a single environment in which you may test, verify, and optimize your application code.
The debugger includes traditional features like simple and complex breakpoints, watch windows, and execution control and provides full visibility to device peripherals.
µVision Project Manager and Run-Time Environment
With the µVision Project Manager and Run-Time Environment you create software application using pre-build software components and device support from Software Packs. The software components contain libraries, source modules, configuration files, source code templates, and documentation. Software components can be generic to support a wide range of devices and applications

Keil IDE and Debugging tools



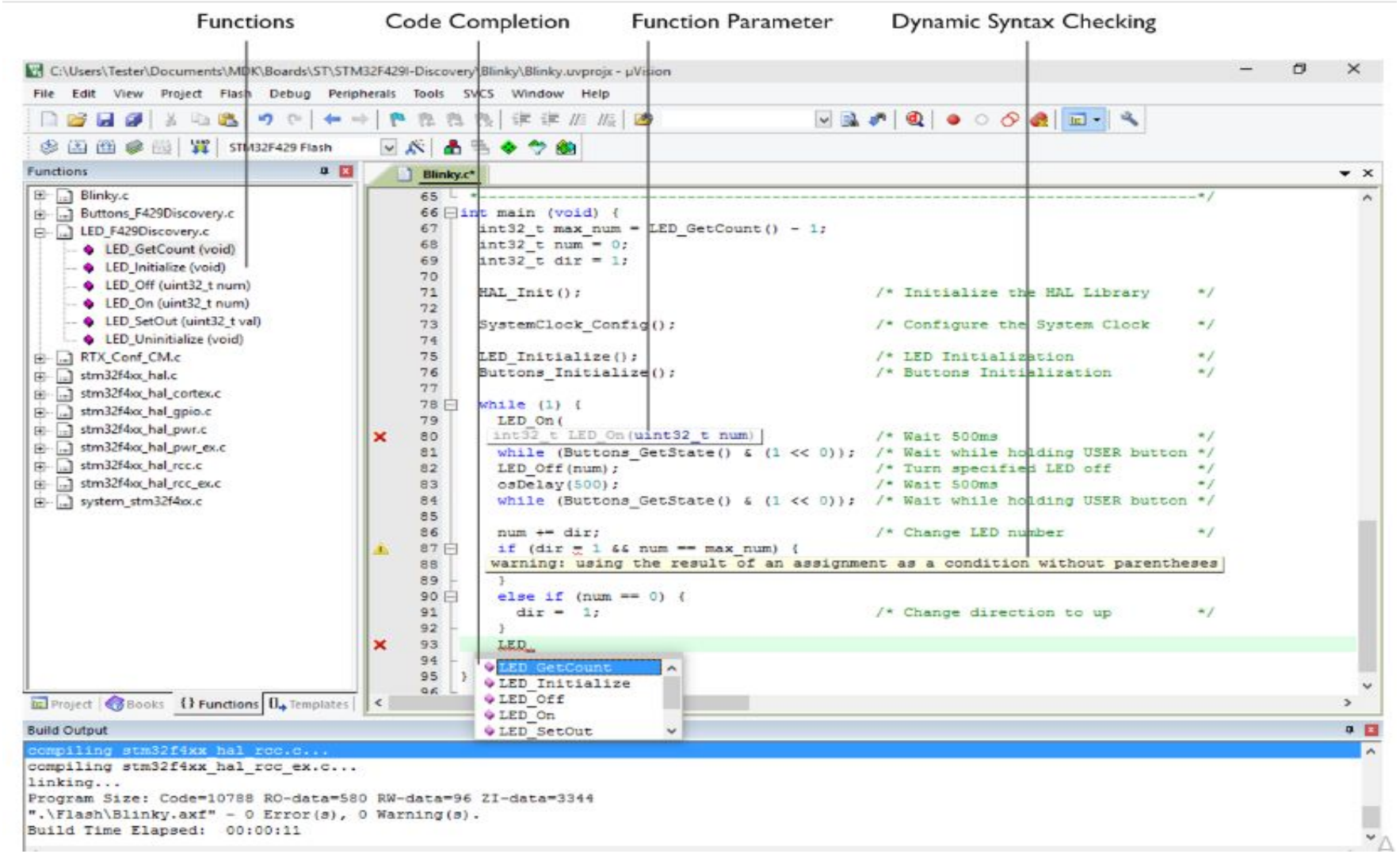
Keil IDE and Debugging tools

- The **Project** window shows application source files and selected software components. Below the components you will find corresponding library and configuration files.
- **Projects** support multiple **targets**. They ease configuration management and may be used to generate debug and release builds or adoptions for different hardware platforms.
- The **Manage Run-Time Environment** window shows all software components that are compatible with the selected device. Inter-dependencies of software components are clearly identified with validation messages.
- The **Configuration Wizard** is an integrated editor utility for generating GUI-like configuration controls in assembler, C/C++, or initialization files.

µVision Editor

The integrated µVision Editor includes all standard features of a modern source code editor and is also available during debugging. Color syntax highlighting, text indentation, and source outlining are optimized for C/C++.

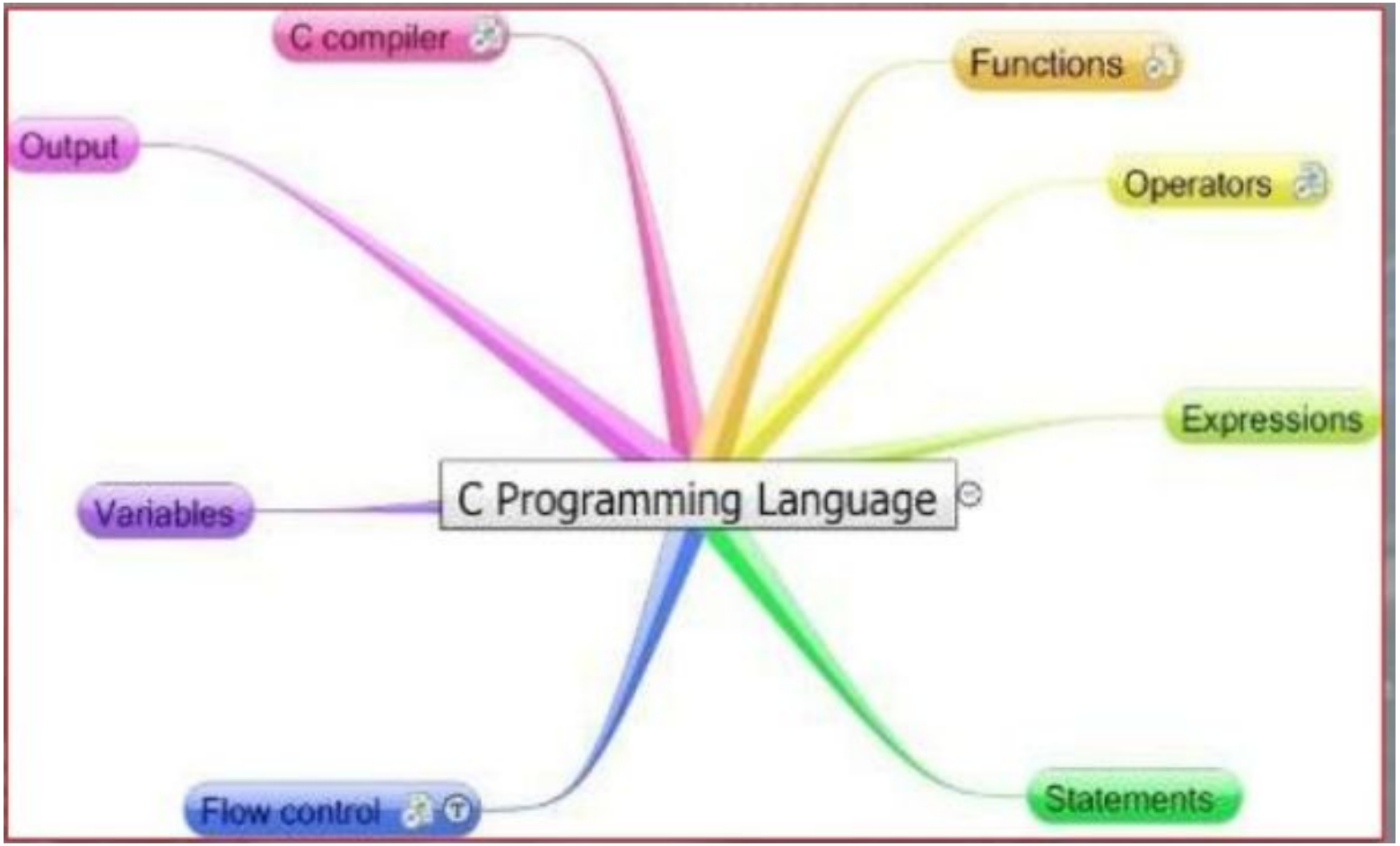
Keil IDE and Debugging tools



Keil IDE and Debugging tools

The **Functions** window gives fast access to the functions in each C/C++ source code module.
The **Code Completion** list and **Function Parameter** information helps you to keep track of symbols, functions, and parameters.
Dynamic Syntax Checking validates the program syntax while you are typing and provides real-time alerts to potential code violations before compilation.

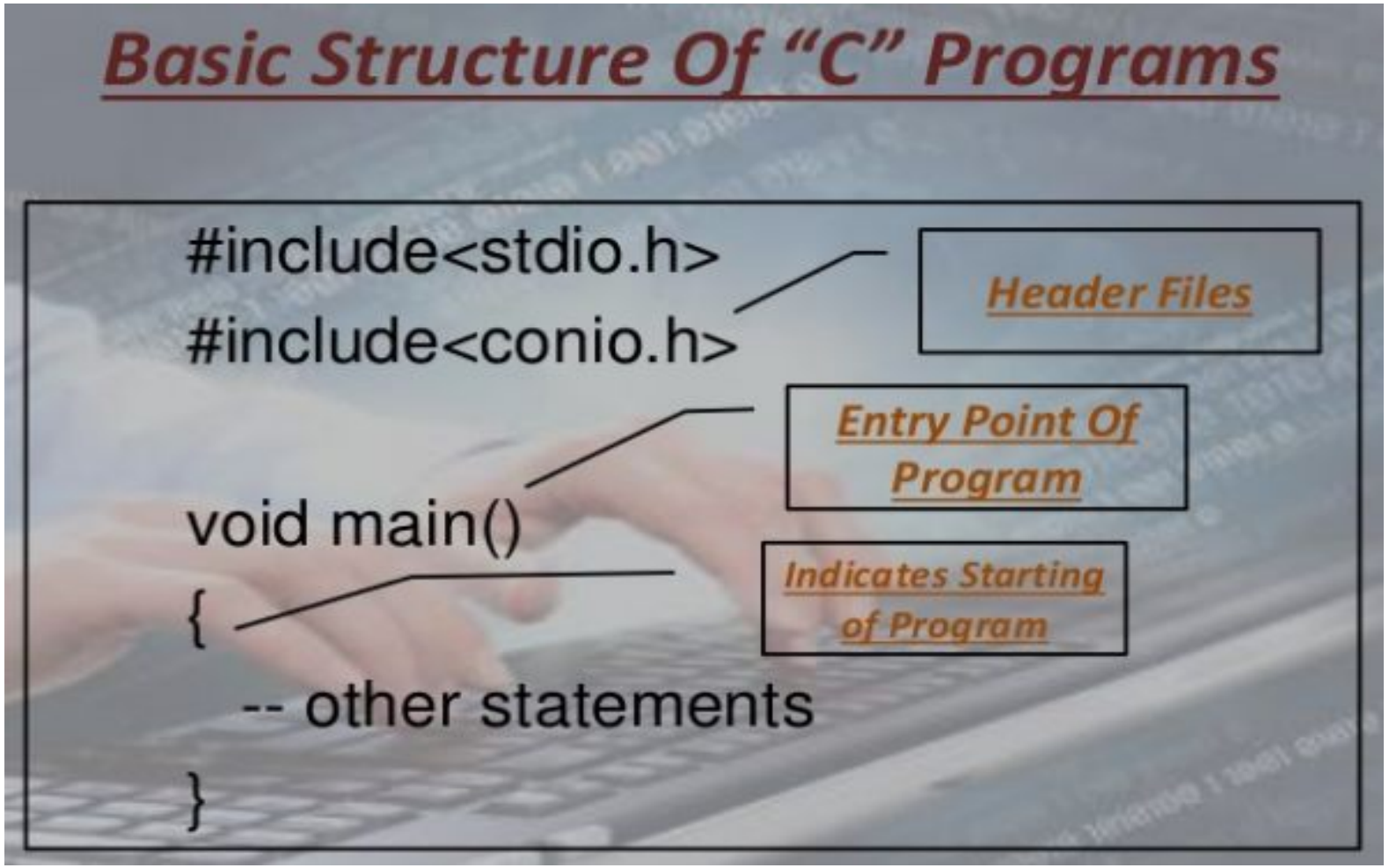
C- language review



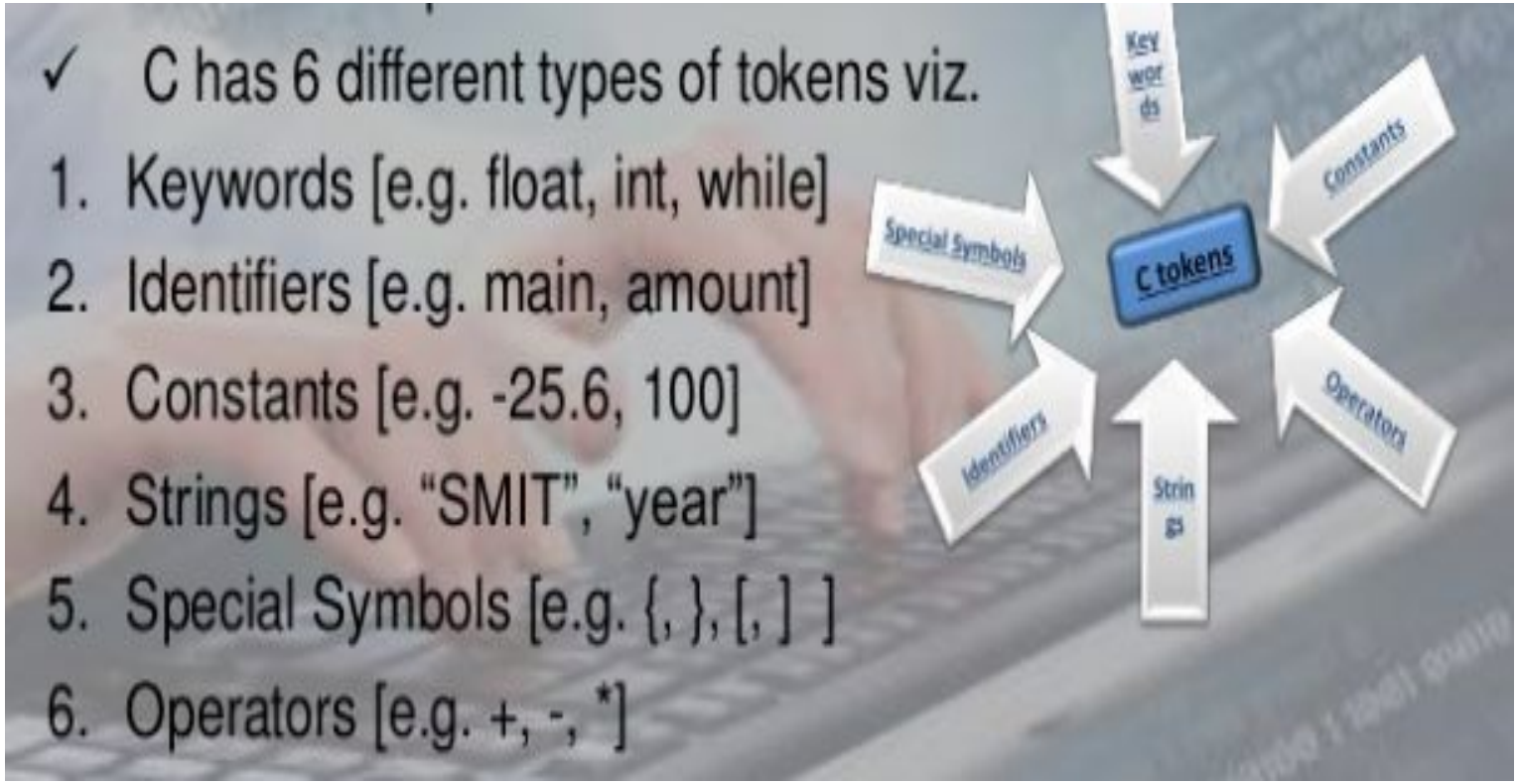
C- language review

- Introduction to C
- Variables and Data Types
- Blocks and Compound Statements
- Control Flow
- Modular Programming
- Variable Scope
- I/O
- Pointers and Memory Addresses
- Arrays and Pointers Arithmetic
- Search and Sorting Algorithms
- User Defined Data Types
- Data Structures
- Callbacks
- Hash Tables
- Using External Libraries
- Creating Libraries
- Standard Library

C- language review

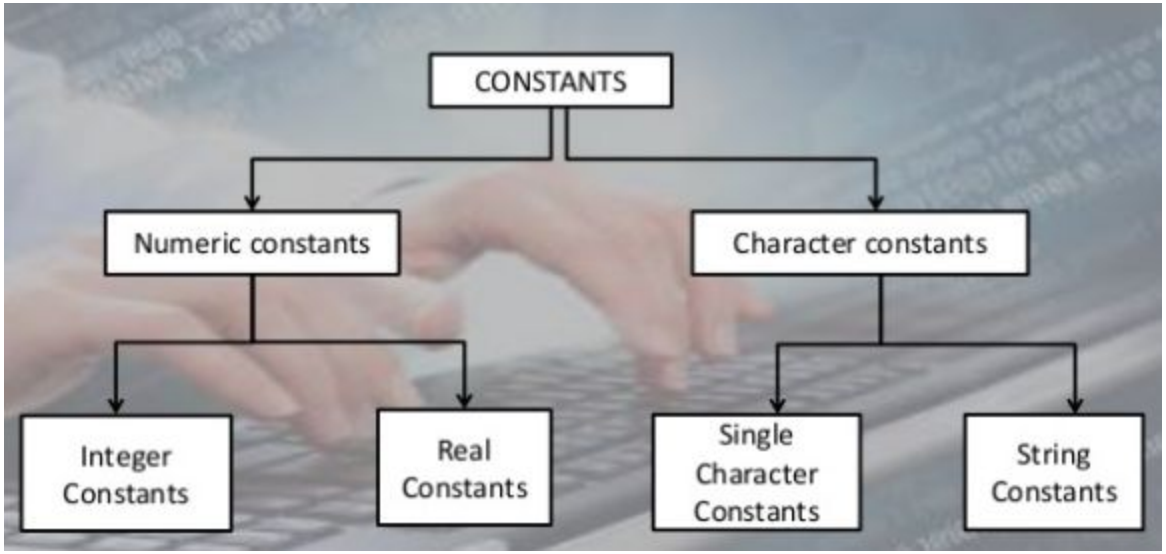


C- language review



C- language review

C- language review



$$x = a + b$$
$$z + 2 = 3(y - 5)$$

C- language review

Data types in 'ansi c'

- There are three classes of data types here::
- Primitive data types
  - int, float, double, char
- Aggregate OR derived data types
  - Arrays come under this category
  - Arrays can contain collection of int or float or char or double data
- User defined data types
  - Structures and enum fall under this category.

Type	Size	Representation	Minimum range	Maximum range
char, signed char	8 bits	ASCII	-128	127
unsigned char	bool 8 bits	ASCII	0	255
short, signed short	16 bits	2's complement	-32768	32767
unsigned short	16 bits	Binary	0	65535
int, signed int	16 bits	2's complement	-32768	32767
unsigned int	16 bits	Binary	0	65535
long, signed long	32 bits	2's complement	-2,147,483,648	2,147,483,647
unsigned long	32 bits	Binary	0	4,294,967,295
float	32 bits	IEEE 32-bit	1.175495e-38	3.4028235e+38
double	32 bits	IEEE 32-bit	1.175495e-38	3.4028235e+38
long double	32 bits	IEEE 32-bit	1.175495e-38	3.4028235e+38

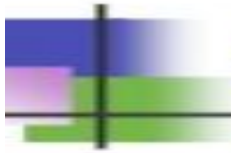
C- language review

- Orders of Operation

Operator	Evaluation Direction
+, - (sign)	Right-to-left
*, /, %	Left-to-right
+, -	Left-to-right
=, +=, -=, *=, %=	Right-to-left

- Use parentheses to override order of evaluation

C- language review



Operators

- Arithmetic
  - +, -, *, /, %
- Relational
  - >, >=, <, <=, ==, !=
- Logical
  - &&, ||, !
- Increment and decrement
  - X++, Y—
  - --X, ++X
- Bitwise
  - &, |, ^, ~, >>, <<
- Assignment
  - +=, &=...

C- language review

Conditional Expression

```
if(cond){

} else if(another cond){

} else {

}
```

```
(cond) ? true statement : false statement;
/* Same as
 * if (cond){
 *  true statement
 * } else
 *  fasle statement
 * }
 */
```

C- language review

The switch Statement

```
switch(ch){
    case 'Y' : /* ch == 'Y' */
               /* do something */
               break;
    case 'N' : /* ch == 'N' */
               /* do something else */
               break;
    default : /* otherwise */
              /* do a third thing */
              break;
}
```

```
switch(ch){
    case 'Y' : /* ch == 'Y' */
               /* do something */

    case 'N' : /* ch == 'N' */
               /* do something if 'Y' or 'N' */
               break;
    default : /* otherwise */
              /* do a third thing */
              break;
}
```


The do-while Loop

```
do{
    /* Loop body */
}while(cond);

/* Loop body */
while(cond){
    /* Loop body */
}
```

C- language review

The while Loop

```
while(condition){
    /* loop body
     * is executed
     * if condition
     * is true
     */
}
```

The for Loop

```
for(initialization; condition; termination){
    /* Loop body */
}

initialization
while(condition){
    /* Loop body */
    termination
}
```

C- language review

goto Keyword

- Allows you to jump unconditionally to arbitrary part of your code (within the same function)
- The location is identified using a label.
  - A label is a named location in the code. It has the same form as a variable followed by a ':' .
- Do not use it.

```
start:
    {
        if(cond)
            goto outside;
        /* some code */
        goto start;
    }

outside :
    /* outside block */
```

C- language review

Function Prototypes

- Functions also must be declared before use
- Declaration called function prototype
- Prototypes for many common functions in header files for C Standard Library
- General form:
  - return_type function_name(arg1, arg2, ...)

```
/*Examples of function prototypes */

int factorial (int);

int factorial (int n);
```

C- language review

The main() Function

- C Program entry point
- Can be one of:

```
int main(void);

int main(int argc, char ** argv);
```

C- language review

Function Definition

- Must match prototype (if there is one)
- Variable names don't have to match
- Curly braces define a block

```
Function declaration{
    declare variables;
    program statements;
}
```

- Variables declared in a block exist only in that block
- Variable declarations must be before any other statements

C- language review

More About Strings

- Strings stored as character array
- Null-terminated
  - Last character in array is '\0'
- Not written explicitly in string literals
- Special characters specified using \ (escape character):
  - \\ - backslash, \' - apostrophe, \" - quotation mark
  - \b, \t, \r, \n - backspace, tab, carriage return, linefeed
  - \ooo, \xhh - octal and hexadecimal ASCII character codes
    - \x41 – 'A',
    - \060 – '0'

C- language review

Console IO

- stdout, stdin: console output and input streams
- puts(string): print string to stdout
- putchar(char): print character to stdout
- char = getchar(): return character from stdin
- string = gets(string): read line from stdin into string

C- language review

Preprocessor Macros

- They begin with #.
- They can take arguments.
- Parentheses ensure order of operations.
- Compiler performs inline replacement.

```
#define msg "Hello world!\n"

#define add3(x,y,z) ((x)+(y)+(z))
```

C- language review

File IO

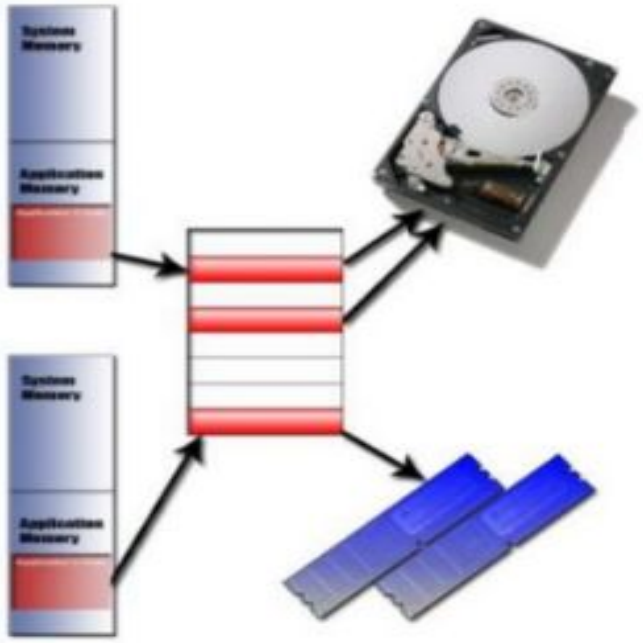
- C allows us to read/write data from text/binary files.
- We can:
  - Open a file
  - Close a file
  - Read a single character
  - Read a single line
  - Write a single character
  - Write a single line
  - Read formatted line
  - Write formatted line

```
FILE * fopen(char * filename, char * mode)
int fclose(FILE * fp)
int getc(FILE * fp)
char * fgets(char * line, int maxlen, FILE * fp)
int putc(int c, FILE * fp)
int fputs(char * line, FILE * fp)
int fscanf(FILE * fp, char * format, ...)
int fprintf(FILE * fp, char * format, ...)
```

C- language review

Physical vs. Virtual Memory

- Physical memories are physical resources where data can be stored.
  - Caches
  - RAMs
  - Hard disks
  - Removable storage
- Virtual memory is an OS abstracted addressable space accessible by your code.



C- language review

Physical Memory Considerations

- Different sizes and access speeds
- Memory management is a major OS function.
- You have to optimize your code to make the best usage of the physical memory.
- OS moves data around physical memory during execution.
- In embedded systems, it may be very limited.

Addressing Variables

- Every variables has an address in memory.
- What does not have an address?
  - Register variables
  - Expressions unless result is a variable.
  - Constants, literals, and preprocessors
- The & operator finds the address of a variable.
- Address of a variable of type t has type t *.

```
int n = 4;
int * pn = &n; /* Address of integer n */
```

C- language review

Dereferencing Pointers

- Using the * operator, I can access and modify addressed variable.

```
/* prints "Value is 4" */
printf("Value is %d\n", *pn);

/* n equals 7 now */
*pn = *pn + 3;
```

- A pointer that dereferences nothing is called a NULL pointer.

Functions with Multiple Outputs

- Recall extended Ecluid, it calculates g = gcd(a,b) and sets the global variables x and y.
- Using pointers we can, extend the outputs of a function.

```
int ext_ecluid(int a, int b, int * x, int * y);

...

g = ext_ecluid(a, b, &x, &y);
```

C- language review

Pointer to Pointers

- Address stored by a pointer is also data in memory.
- A pointer to a pointer can address location of address in memory.
- Uses in C:
  - Pointer arrays
  - String arrays

```
int n= 3;
int * pn = &n; /* pointer to n */
int   ** ppn = &pn; /* pointer to address of n */
```

Arrays and Pointers

- Arrays in C are implemented using a pointer to block of contiguous memory.
- [] can be used for accessing array elements.
- Array name is a pointer to its 1st element.
  - Not modifiable/reassignable like any pointer.

```
int arr[8];
int a = arr[0];
int * pa = arr; /* int * pa = &arr[0]; */
```

C- language review

Structure

- A collection of related variables grouped under a single name.

```
struct point{
     int x;
     int y;
}; /* Note the ; @ the end */

struct rectangle{
     struct point tl;
     struct point br;
}; /* Memebers can be structures */

struct chain_element{
     int data;
     struct chain_element * next;
} /* Members can be self referential */
```

Union

- May hold objects of different types/sizes in the same memory location.
- Union size is equal to the size of its largest element.
- The compiler does not test if the data is being read in the correct format.

```
union data {
     int idata;
     float fdata;
     char * sdata;
} d1,d2,d3;

d1. idata=10;
d1. fdata=3.14F;
d1. sdata="hello world";

d2.idata=10;
float f=d2.fdata; /* will give junk */

/* Maintain a seperate variable to store
 * what is the latest assignment type */
enum dtype {INT ,FLOAT,CHAR};

struct variant {
     union data d;
     enum dtype t ;
};
```

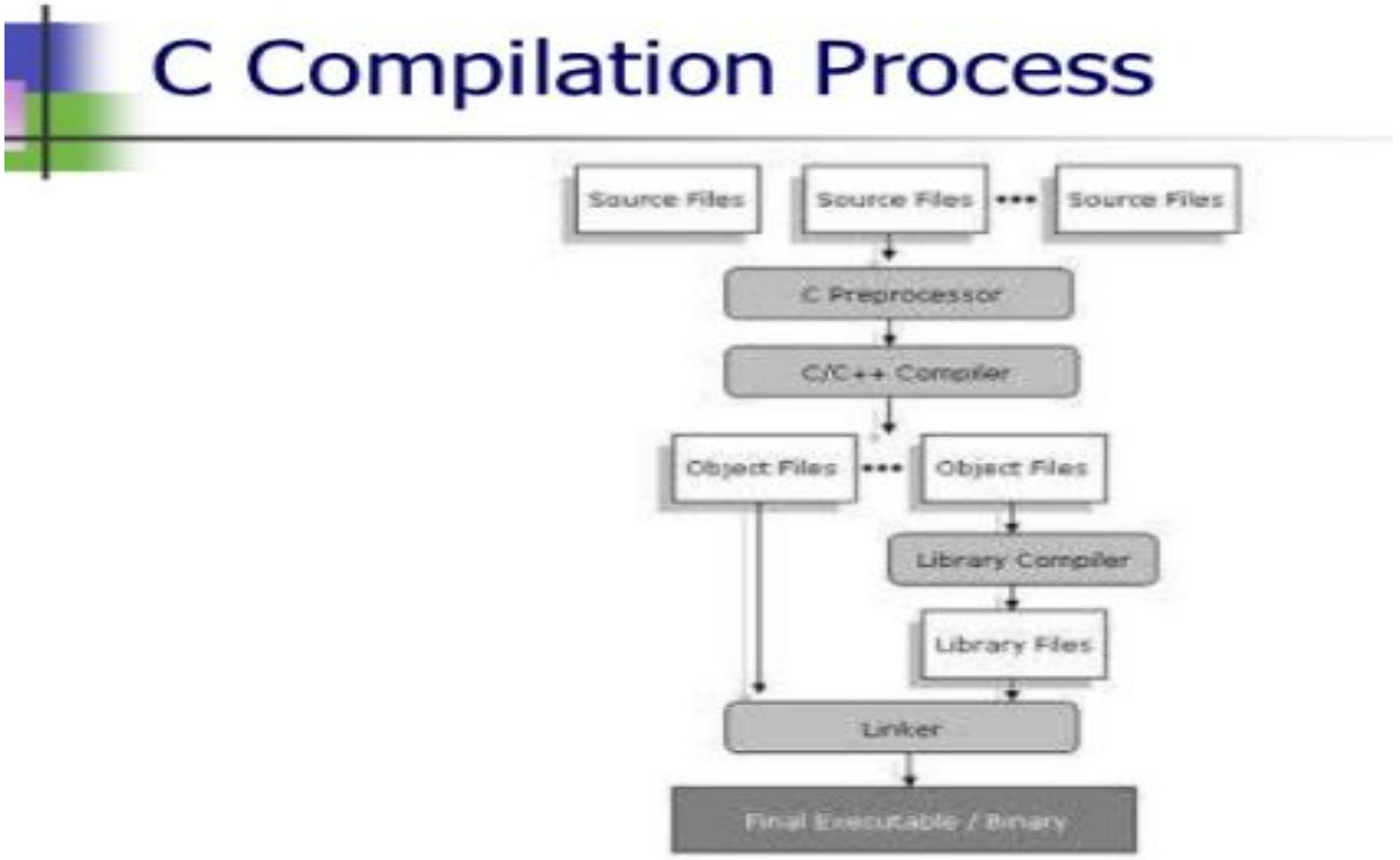
C- language review

Dynamic Memory Allocation

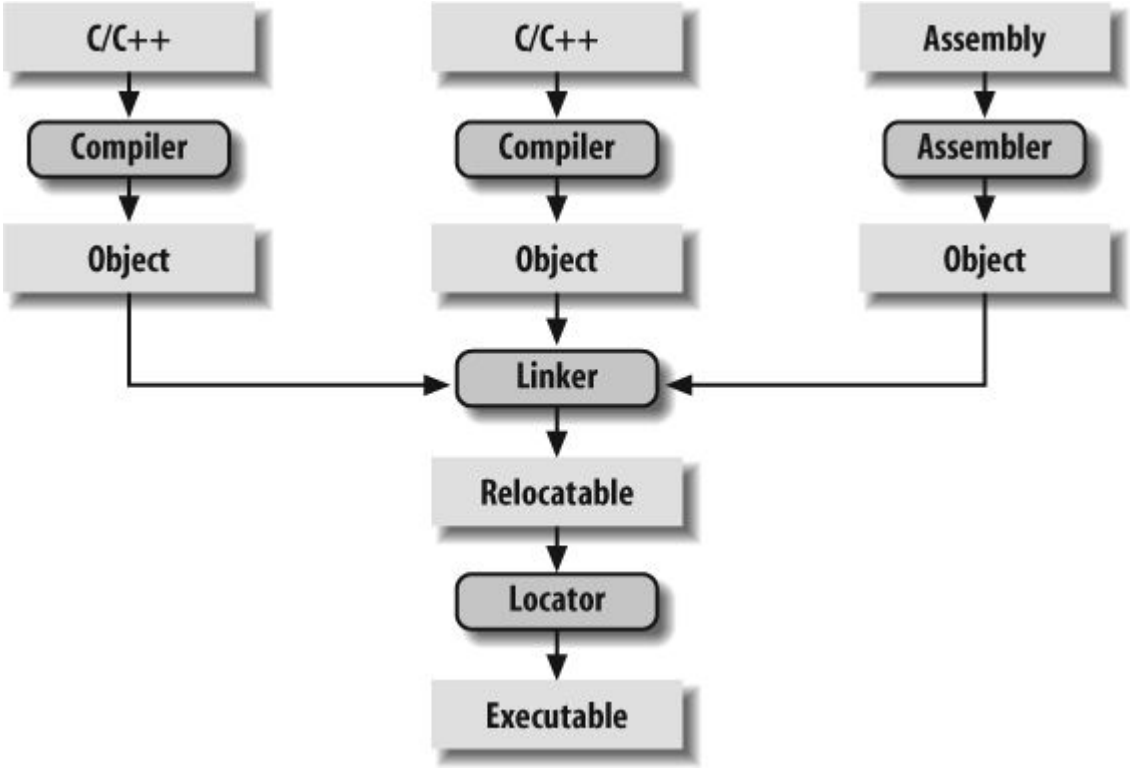
- malloc() allocates an uninitialized block of memory.
- calloc() allocates a zero initialized array of n elements.
- free() frees allocated memory.
- Common errors:
  - Accessing freed memory
  - Accessing uninitialized pointer
  - Memory leak

```
int * ip = (int *) malloc(sizeof(int) * 100);
int * ip2 = (int *) calloc(100, sizeof(int));
...
free(ip);
free(ip2);
```

C- language review



C- language review



Embedded C

WHAT IS EMBEDDED C?

Whenever the conventional 'C' language and its extensions are used for programming embedded systems, it is referred to as "Embedded C" programming.

C V/S EMBEDDED C

- 'C' is a well structured, well defined and standardised general purpose programming language.
- A platform specific application , known as , compiler is used for the conversion of programs written in C to the target processor specific binary files.
- Embedded 'C' can be considered as a subset of conventional 'C' language.
- A software program called 'Cross compiler' is used for the conversion of programs written in Embedded 'C' to target processor/controller specific instructions(machine language).

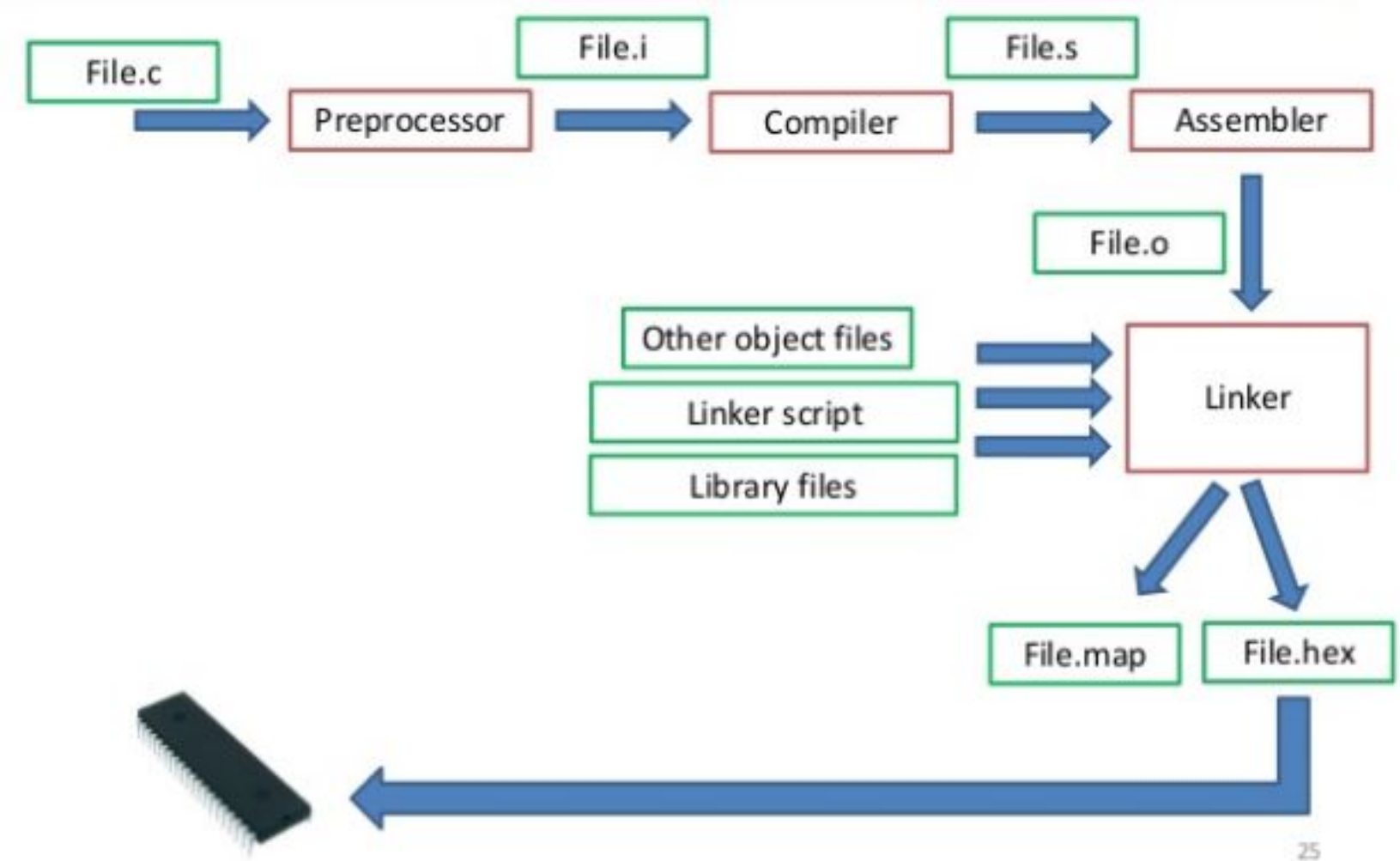
Embedded C

COMPILER V/S CROSS COMPILER

- Compiler is a software tool that converts a source code written in a high level language on top of a particular operating system running on a specific target processor architecture.

- Cross compilers are software tools used in cross platform development applications. (In cross platform development , the compiler running on a particular target processor/OS converts the source code to machine code for a target processor whose *architecture and instruction set is different* from the current development environment OS).

Embedded C


File.c
Preprocessor
File.i
Compiler
File.s
Assembler
File.o
Other object files
Linker script
Library files
Linker
File.map
File.hex

Development Environment using the mbed

Getting Started With the mbed:

- An mbed microcontroller with its USB lead,
- a computer running Windows, or Mac OS X, or GNU/Linux, and
- a web browser, for example, Internet Explorer, Safari, Chrome, or Firefox.

Step 1. Connecting the mbed to the PC

Connect the mbed to the PC using the USB lead

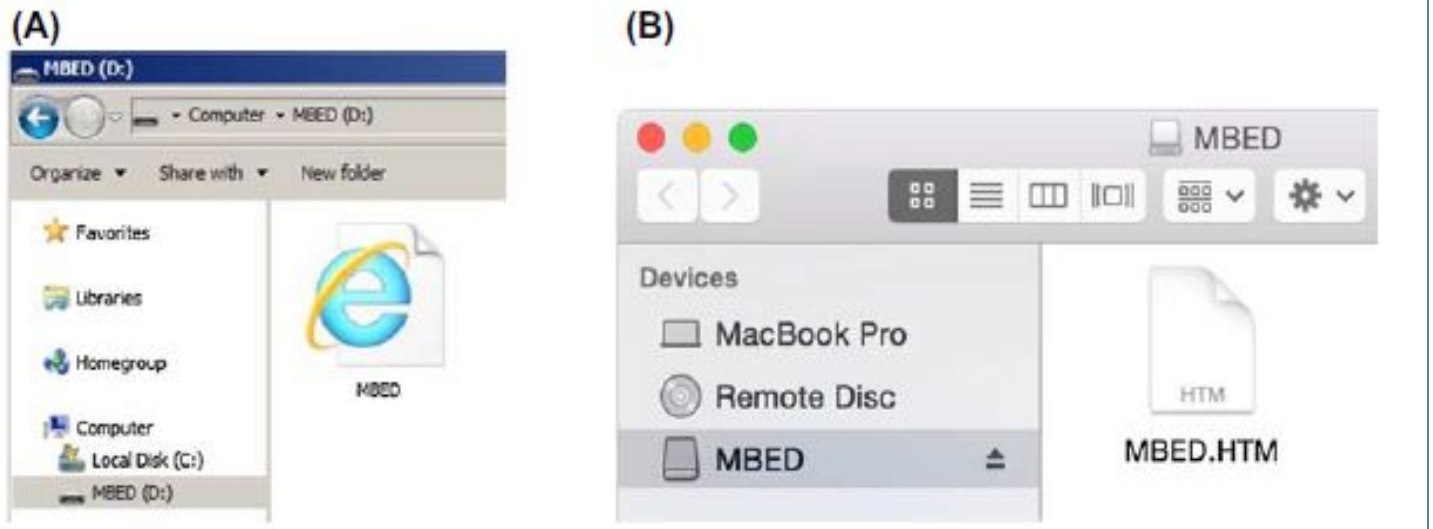


Figure 2.4

Locating the mbed (A) Windows example (B) Mac OS X example.

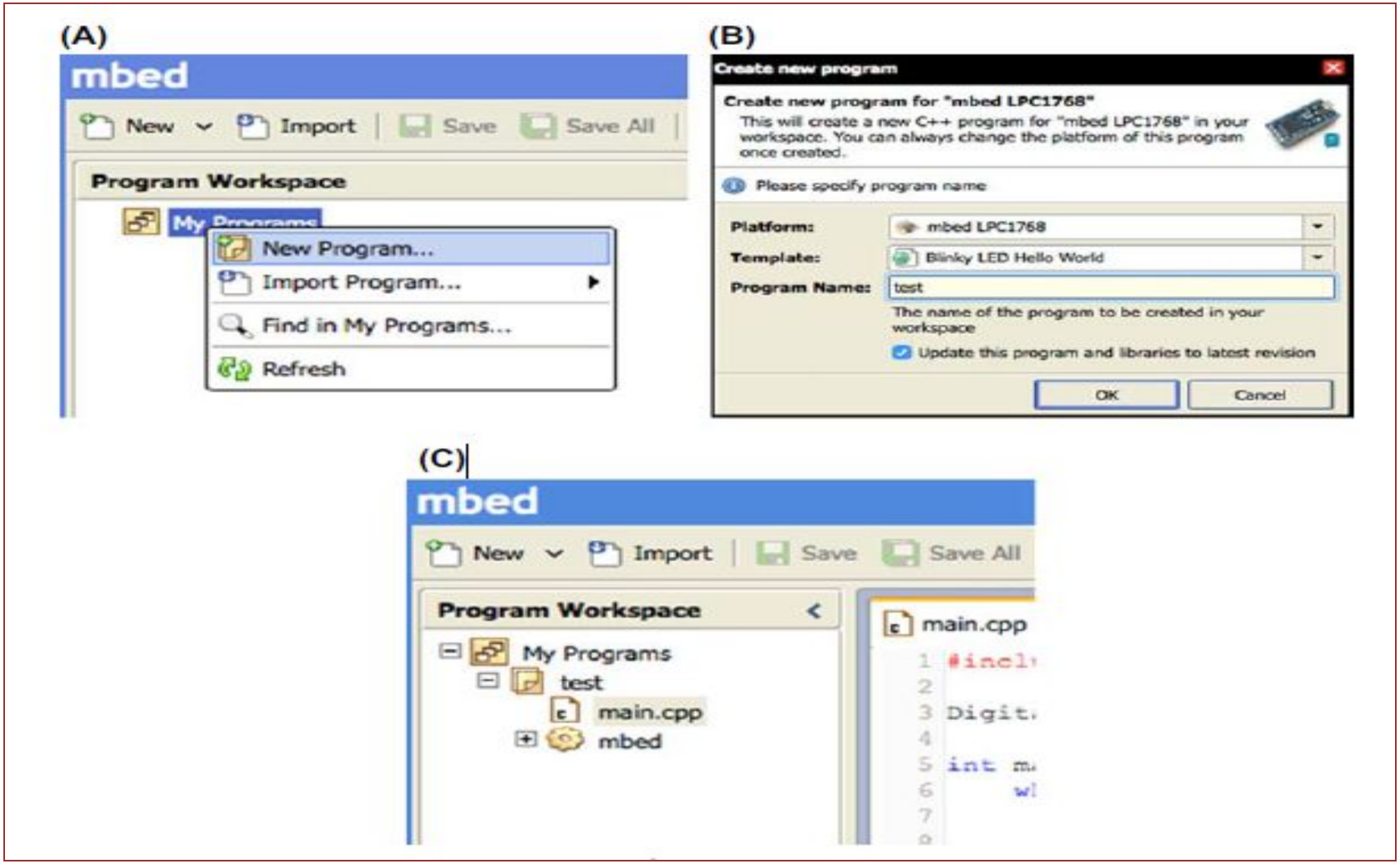
Development Environment using the mbed

Getting Started With the mbed:
Step 2. Creating an mbed Account

Development Environment using the mbed

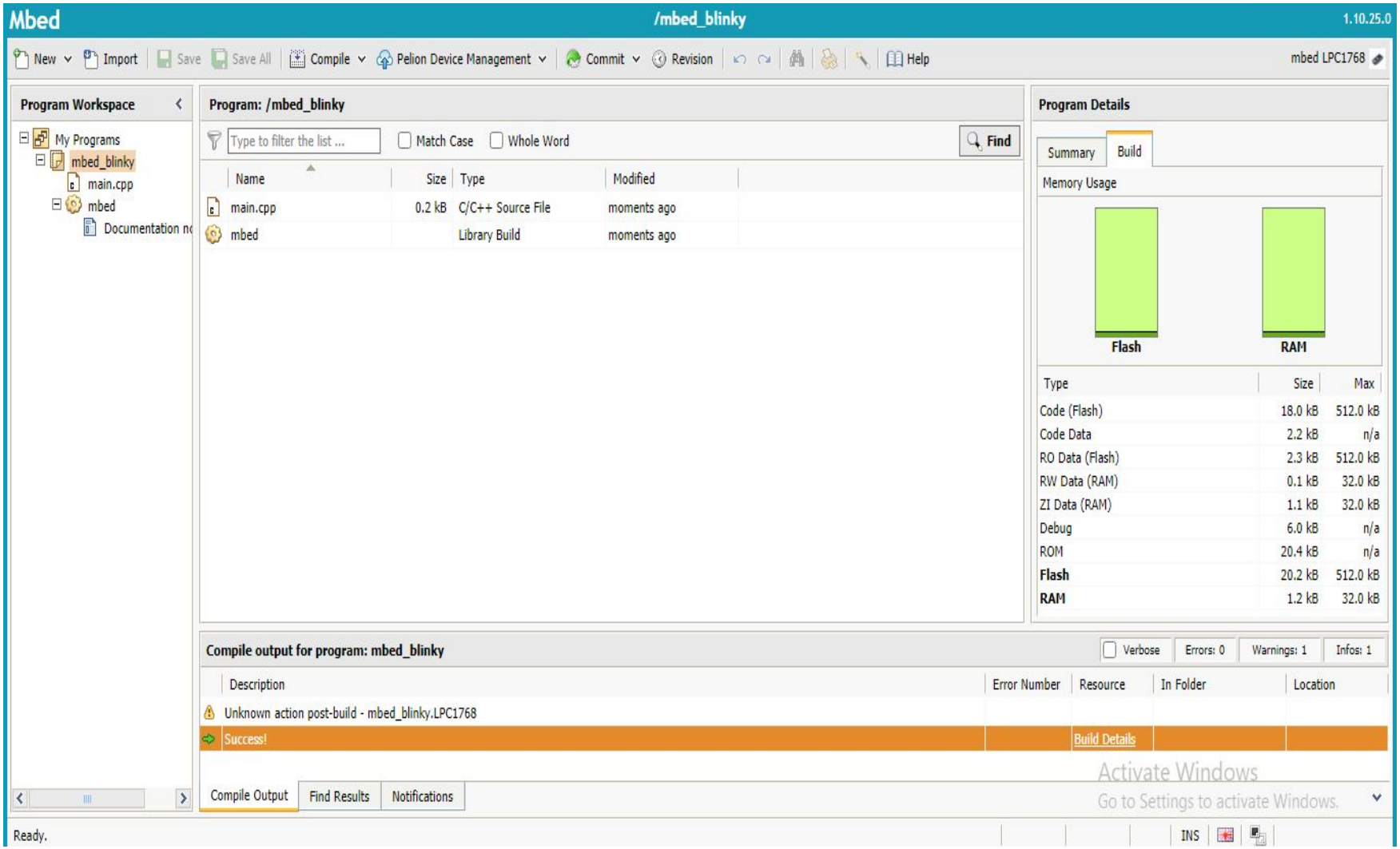
Getting Started With the mbed:

Step 3. Running a Program



Development Environment using the mbed

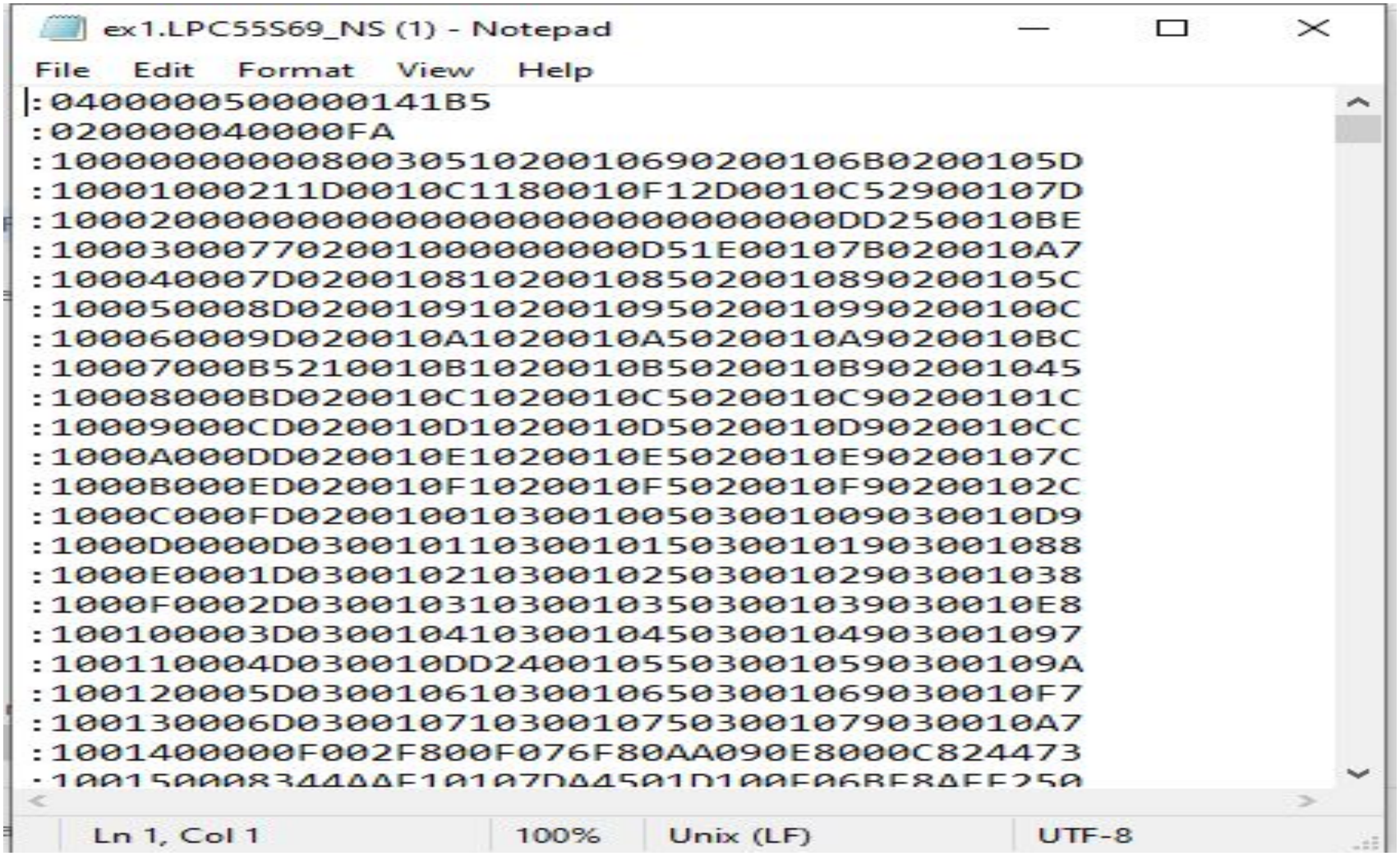
Getting Started With the mbed:
Step 4. Compiling the Program



Development Environment using the mbed

Getting Started With the mbed:

Step 5. Downloading the Program Binary Code



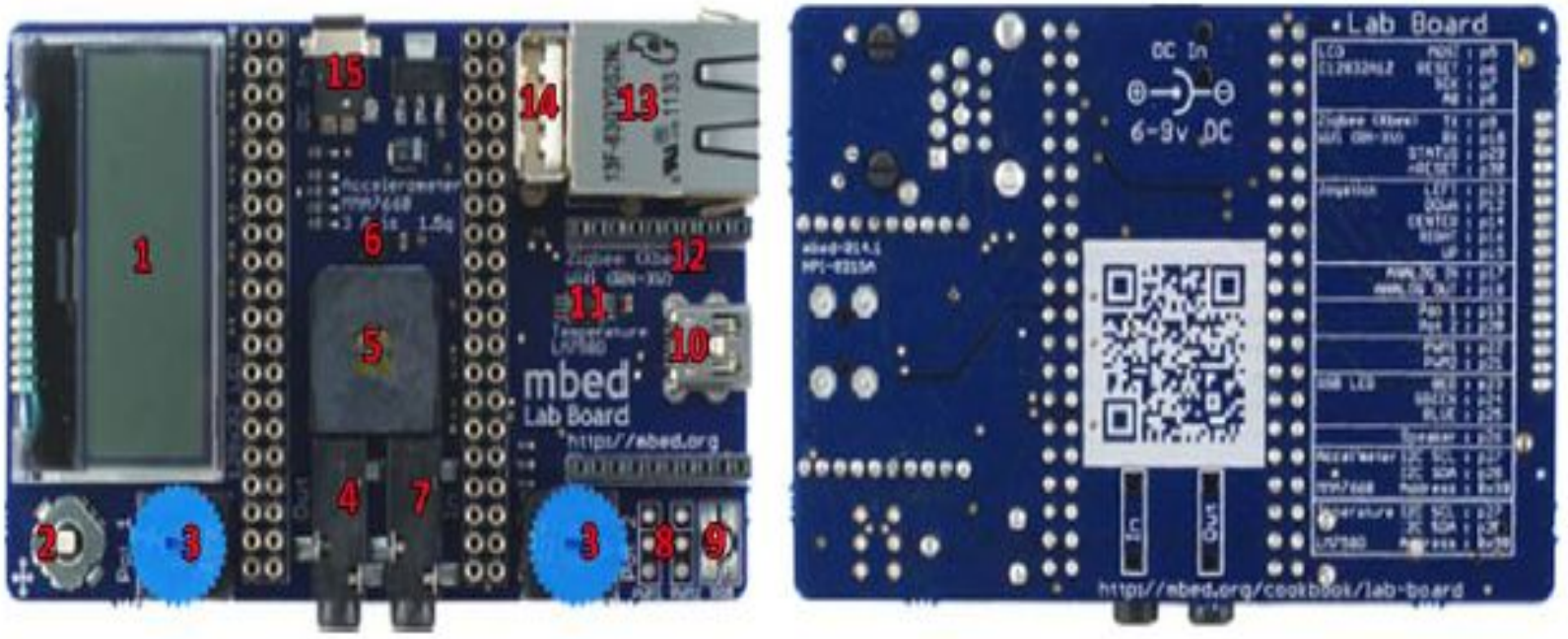
Development Environment using the mbed

Getting Started With the mbed:
Step 6. Modifying the Program Code

```
#include "mbed.h"
DigitalOut myled(LED1);
DigitalOut yourled(LED4);
int main() {
char i=0; //declare variable i, and set to 0
while(1){ //start endless loop
while(i<10) { //start first conditional while loop
myled = 1;
wait(0.2);
myled = 0;
wait(0.2);
i = i+1; //increment i
} //end of first conditional while loop
while(i>0) { //start second conditional loop
yourled = 1;
wait(0.2);
yourled = 0;
wait(0.2);
i = i-1;
}
} //end infinite loop block
} //end of main
```

Development Environment using the mbed

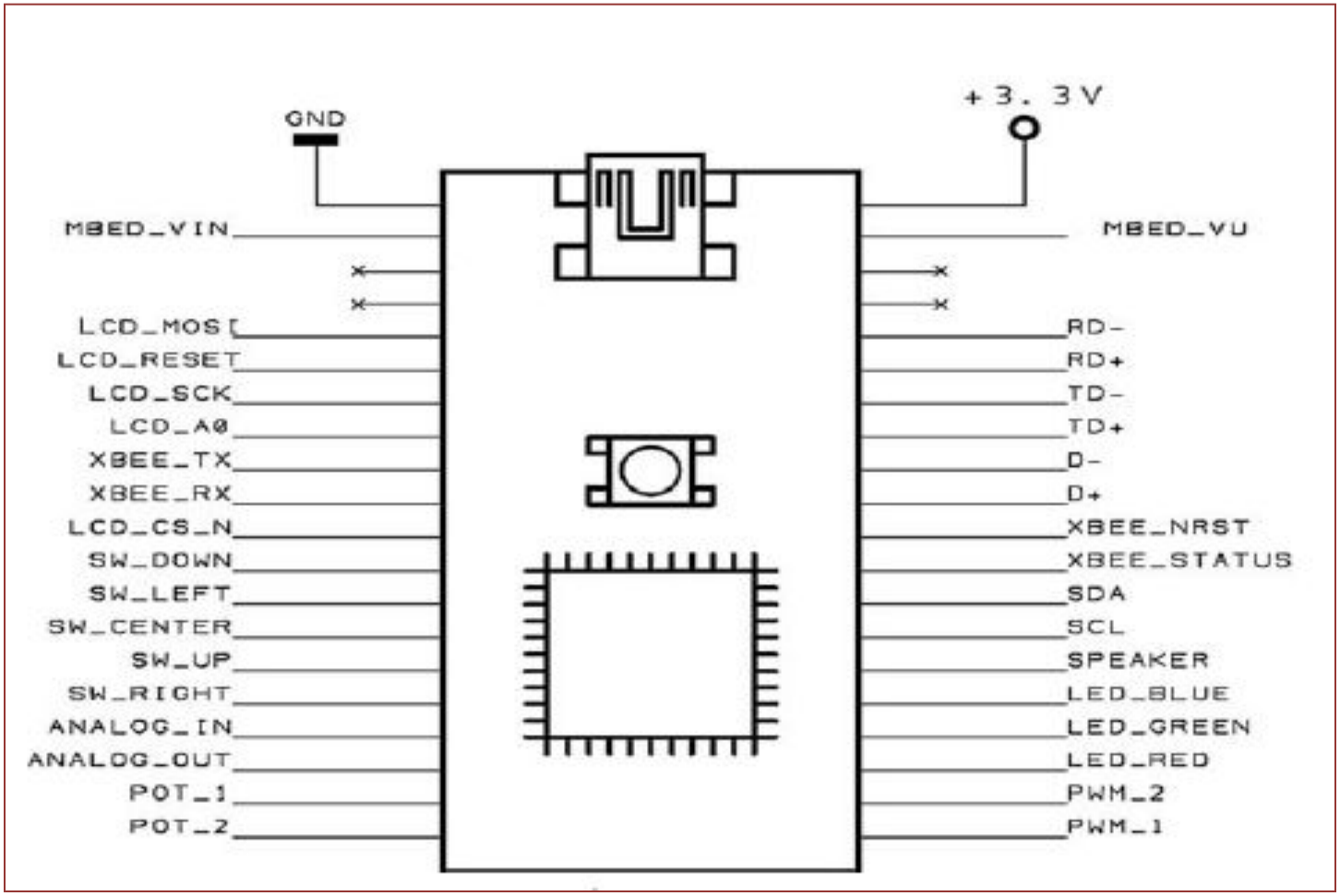
The mbed Application Board



1. Graphics LCD (128x32)
2. 5 Way Joystick
3. 2 x Potentiometers
4. 3.5mm Audio Jack (Analog Out)
5. Speaker, PWM Connected
6. 3 Axis +/1 1.5g Accelerometer
7. 3.5mm Audio Jack (Analog In)
8. 2 x Servo Motor Headers
9. RGB LED, PWM connected
10. USB-mini-B Connector
11. Temperature Sensor
12. Socket for Xbee (Zigbee) or RN-XV (Wifi)
13. RJ45 Ethernet Connector
14. USB-A Connector
15. 1.3mm DC Jack input

Development Environment using the mbed

The mbed Application Board



VOUT
VU
IF-
IF+
RD-
RD+
TD-
TD+
D-
D+
30
29
28
27
26
25
24
23
22
21
Ethernet
USB
CAN
I2C
Serial
PWM
https://mbed.com
Mbed microcontroller
Serial
Serial
SPI
I2C
SPI
Analog In
GND
VIN
VB
nR
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20