

UNIT-II

Embedded Processor Architecture

CISC Vs RISC Design Philosophy:

Instruction Set Architecture:

Instruction set can be defined as the communication interface between the processor and the programmer. Every processor has its own instruction set implemented in the hardware to execute instructions such as move, add or multiply data in a definite way. Programmers can either use any high level language such as C, C ++, Java etc. or assembly language to write the program. Accordingly, a compiler or assembler can be used to translate the program into machine understandable language following the processor instruction set. There are two classic architectures of instruction set implementation, the Complex Instruction Set Computer (CISC) and the Reduced Instruction Set Computer (RISC). Each has its own advantages and disadvantages. The CISC architecture has more complexity in the hardware itself while RISC architecture offers more complexity to the software. The features of each architecture are summarized as below.

S.No.	CISC	RISC
1	Most of the instructions are complex in type	Most of the instructions are simple in nature.
2	"LOAD" and "STORE" incorporated in Instructions Ex: MOV	"LOAD" and "STORE" are independent instructions Ex: LOAD, STORE
3	It has more addressing modes	It has less addressing modes
4	Instruction sizes are different	Instruction sizes are same
5	Number of machine cycles for instruction are different	Number of machine cycles for instruction are same
6	Fewer working registers and more frequent memory access.	Higher number of working registers so less frequent memory access.
7	Pipeline implementation is difficult.	Pipeline implementation is easier.
8	Programming is easy	Programming is complex
9	Hardware is complex	Hardware is simple
10	Small code size	Large code size
11	More complexity is given to the hardware design.	More complexity is offered to the compiler design.
12	Ex: Intel, IBM, Atmel 805x	Ex: MSP430, AMD, ARM, Atmel AVR

Von-Neumann Vs Harvard architecture:

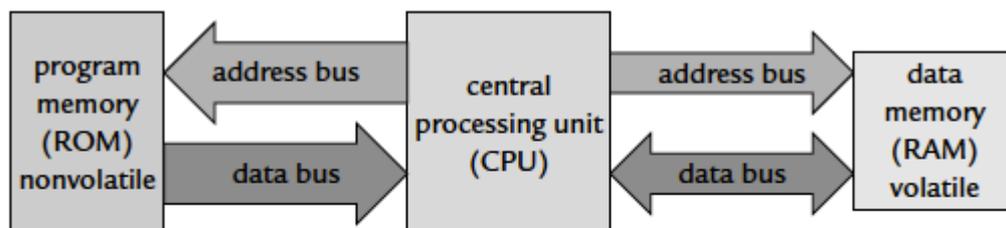
Memory Block:

The memory block consists of program and data memory. ROM is used as the program memory and RAM is used as the data memory. **There are two memory architectures: Harvard and Von-Neumann.**

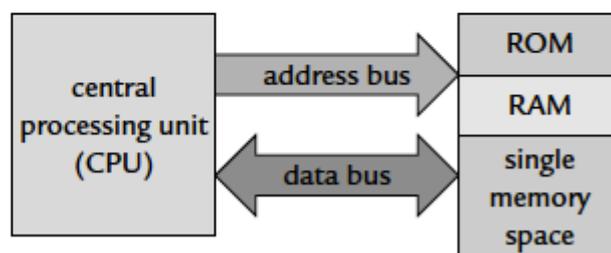
In *Harvard architecture*, the program and data memories are segregated with separate address and data bus drawn to each. So there can be parallel access to both and performance of the system can be improved at the cost of hardware complexity. On the other-hand, the *Von-Neumann architecture* has one unified memory used for both program and data. The system is comparatively slower, but the design implementation is simple and cost effective for an embedded system. The Von-Neumann also called as Princeton Architecture.

S.No.	Von-Neumann	Harvard
1	It has single data/address bus to fetch both program and data	It has separate data/address bus to fetch both program and data
2	Here, both program and data are stored in a same (single) memory.	Here separate memory for program and data.
3	Slower execution	Faster execution.
4	It first fetches an instruction, and then it fetches the data to process the instruction.	Here, both instruction and data are fetches simultaneously
5	More instruction cycles are required	Less instruction cycles are required
6	Program can be easily modified by itself because it is stored in read-write memory	It is difficult for program contents to be modified by the program itself
7	Widely used in microprocessors	Widely used in microcontrollers
8	Ex: Intel 8086, MSP430, ARM7	Ex: Intel 8051, PIC, ARM9

(a) Harvard architecture



(b) von Neumann architecture



Harvard and von Neumann architectures for memory.

I/O mapped I/O vs. Memory mapped I/O

There are two techniques for addressing an I/O device by CPU:

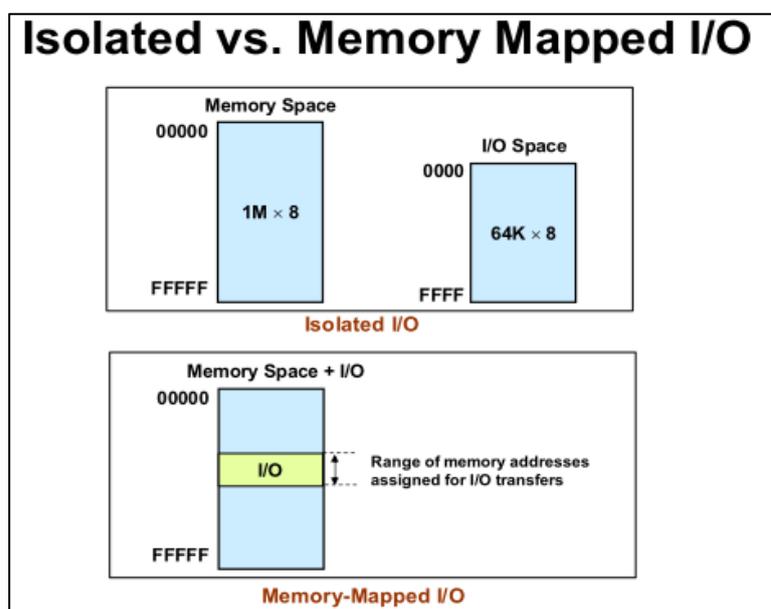
- I/O mapped I/O (Isolated I/O or Port mapped I/O)
- Memory mapped I/O

I/O mapped I/O:

- Here two separate address spaces are used - one for memory and other for I/O devices.
- The I/O devices are addressed with dedicated address space.
- Hence there are two separate control lines for memory and I/O transfer.
 - I/O read and I/O write lines for I/O transfer
 - Memory Write and Memory Read for memory transfer
- Hence IN and OUT instruction deals with I/O transfer and MOV instruction deals with memory transfer.
- Ex: Intel's 8085, 8086.

Memory mapped I/O:

- The technique in which CPU addresses an I/O device just like a memory i.e., address space is same for both I/O and Memory is called memory mapped I/O scheme.
- In this scheme only one address space is used by CPU. Some addresses of the address space are assigned to memory location and other are assigned to I/O devices.
- There is only one set of read and write lines. Hence there is no separate IN, OUT instructions. MOVE instruction can be used to accomplish both the transfer.
- The instructions used to manipulate the memory can be used for I/O devices.
- Ex: Intel's 8051, TI's MSP430



Little-Endian vs. Big-Endian

This is just a way of saying the order of storing bytes in memory. This order is categorized into two i.e., **little-endian** and **big-endian**.

- The **little-endian machines** like Intel x86, the low-order byte is stored at the lower address and the high-order byte at the higher address.
 - ❖ Ex: Intel's 8086, 80x86, TI's MSP430
- The **big-endian machines** like Motorola MC680xx, the high-order byte is stored at the lower address and the low-order byte at the higher address.
 - ❖ Ex: Motorola MC680x0, Freescale HCS08



Introduction to ARM Architecture and Cortex – M series:

ARM Architecture:

ARM cores are designed specifically for embedded systems. The needs of embedded systems can be satisfied only if features of RISC and CISC are considered together for processor design. So ARM architecture is not a pure RISC architecture. It has a blend of both RISC and CISC features.

S.No.	Features	Benefits to Embedded System
1	High Performance	Ensures the system has a fast response
2	Low power consumption	Makes the system more energy efficient
3	Low silicon area	Reduces the size and also consumes less power
4	High Code density	Helps embedded system to have less memory footprint
5	Load/store architecture	Used to load data from the memory to the ARM CPU register or store data from the CPU register to the memory; enables the memory access when required
6	Register bank with large number of working registers	Required to perform most of the operations within the CPU and provides faster context switch in a multitasking applications

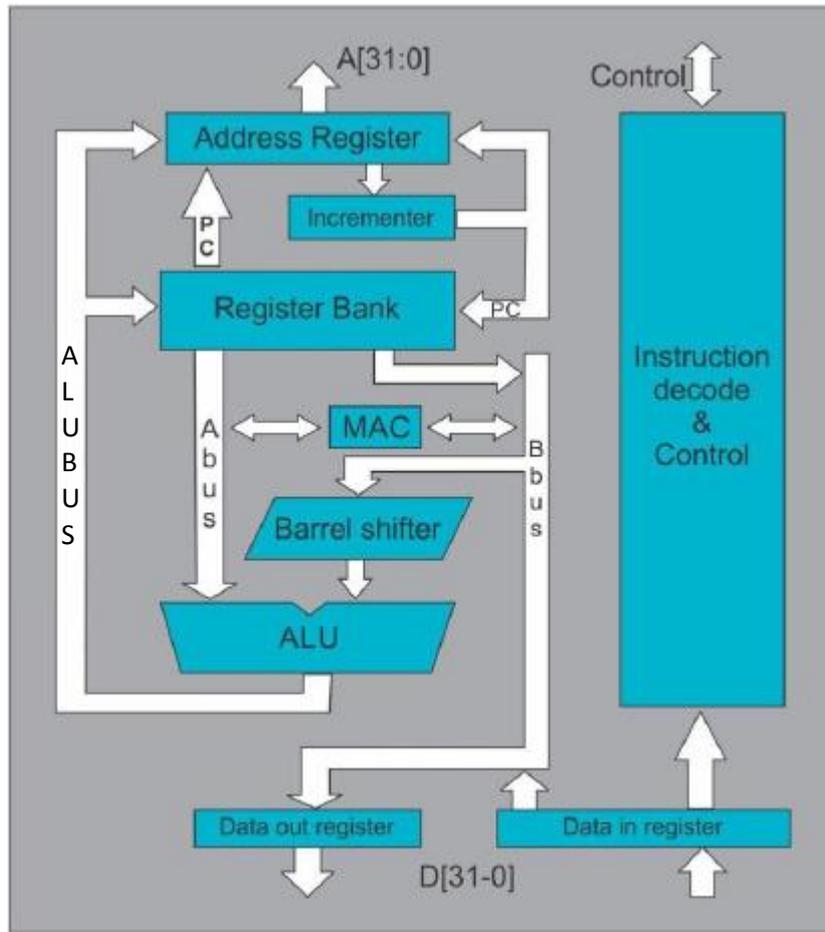


Figure: ARM7 Architecture

A Basic architecture of the ARM7 core:

ARM7, the basic architecture of ARM series of cores, is introduced here in this section. A brief introduction about each functional block of the architecture of ARM7 core shown in above Figure is presented below.

The Register Bank has sixteen general purpose registers (R0-R15) and a current program status register (CPSR) which are accessible by user applications. In addition to that, it has twenty numbers of banked registers specifically used for different operating modes of ARM core. These are invisible to user applications. The register bank has two read ports to read operand1 and operand2 and one write port to write back the result of operation to the any register specified in the instruction. It has an additional bidirectional port to update the program counter with address register and incrementer. Address register content is incremented at every sequential byte access by the incrementer but the program counter is incremented by four in ARM state of the core or is incremented by 2 in Thumb state of the core at every instruction access. Address register is directly connected to the address bus.

- ❑ The barrel shifter can shift or rotate operand 2 by specified number of bits prior to arithmetic or logic operations.
- ❑ The 32 bit ALU performs the arithmetic and logic functions.
- ❑ The data in and data out registers hold the input and output data from and to the memory.

- ❑ The instruction decoder and associated control logic generates appropriate control signals for the data path after decoding the fetched instruction.
- ❑ The MAC unit is to multiply two register operands and accumulate with another register holding the partial sum of the products.

The encoded instruction byte of the program saved in the code memory is fetched through the data bus and first enters into the data-in register of the ARM architecture from where it is delivered to the instruction decoder. After the instruction is decoded, appropriate control signals are generated for the data path. The required registers are activated in the register bank and the operands flow out from two read ports of register bank to the ALU: operand1 through A-bus and operand2 through B-bus after preprocessing at barrel shifter. The result of operation at ALU is written back to the result register through a write port at register bank. For Load/Store instructions, after decoding the instruction, the data memory address is first calculated at ALU as specified in the instruction and the pointer register is updated at the register bank. The address in the pointer register is given to the address register to access the memory and transfer data. If it is a load multiple or store multiple instruction, the core does not halt before completing the required number of data transfers unless it is a reset exception.

Migration to Cortex Series:

In the path of architectural evolution, ARM has contributed many versions of IP cores to the embedded computing world. ARM pioneered embedded products are excelling in every visible spectrum. Since its inception, ARM has migrated over a long meaningful road map starting from v4T ARM7TDMI to v7 Cortex series of architectures achieving many strong milestones in between. It is currently the new era of feature rich ARM Cortex series architectures truly empowering the embedded computing world.

ARM Architecture Evolution:

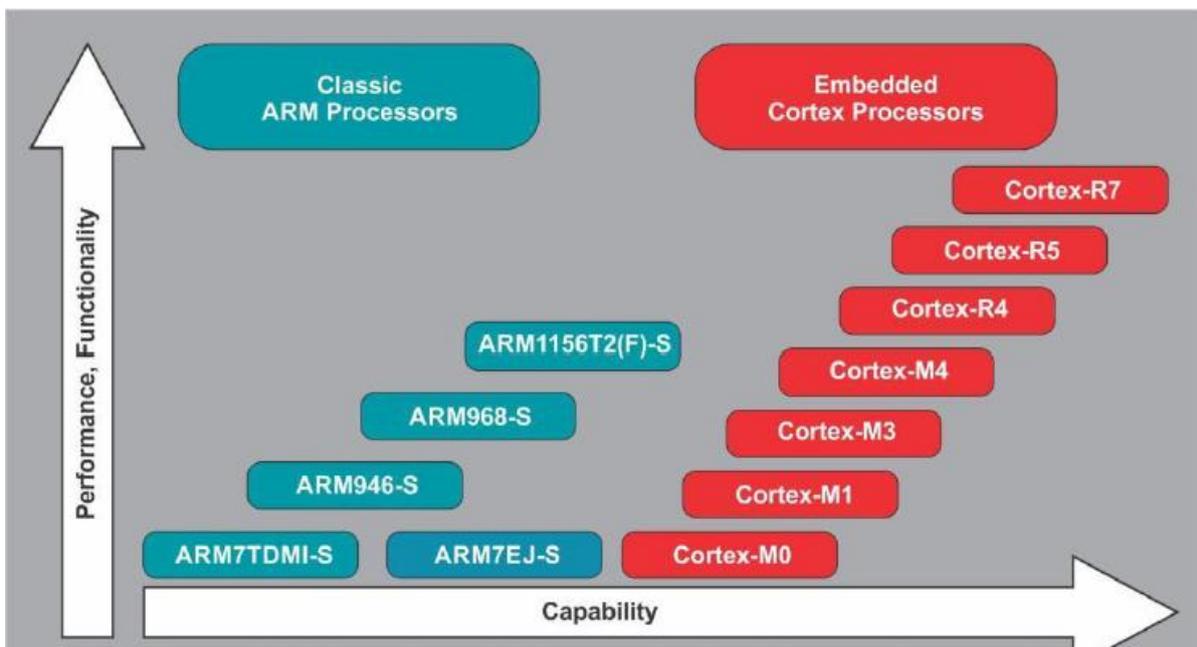


Figure: Performance and capability graph of Classic ARM and Cortex embedded processors.

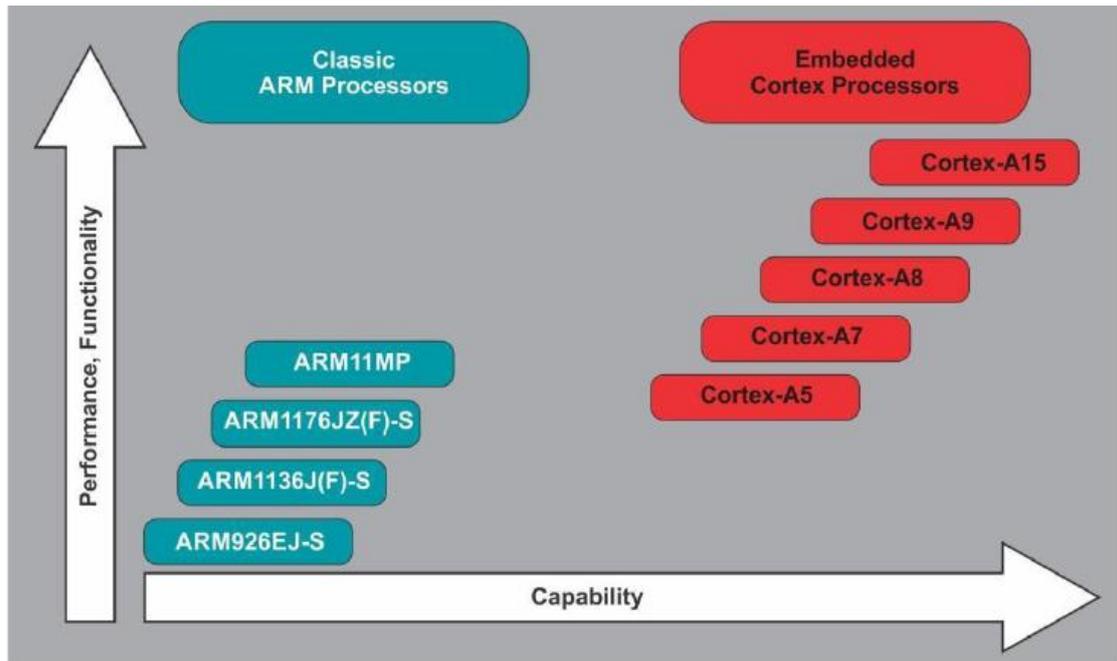


Figure: Performance and capability graph of Classic ARM and Cortex application processors.

ARM architecture has been improved a lot in the road map from classic ARM to ARM Cortex. The above figure depicts the performance and capability comparison of classic ARM with embedded cortex and application cortex series of processors. Even though ARM had earlier versions of products i.e. v1, v2, v3 and v4, the classic group of ARM starts with v4T. The classic group is divided into four basic families called ARM7, ARM9, ARM10 and ARM11.

- ❑ ARM7 has three-stage (fetch, decode, execute) pipeline, Von-Neumann architecture where both address and data use the same bus. It executes v4T instruction set. T stands for Thumb.
- ❑ ARM9 has five-stage (fetch, decode, execute, memory, write) pipeline with higher performance, Harvard architecture with separate instruction and data bus. ARM9 executes v4T and v5TE instruction sets. E stands for enhanced instructions.
- ❑ ARM10 has six-stage (fetch, issue, decode, execute, memory, write) pipeline with optional vector floating point unit and delivers high floating point performance. ARM10 executes v5TE instruction sets.
- ❑ ARM11 has eight-stage pipeline, high performance and power efficiency and it executes v6 instructions set. With the addition of vector floating point unit, it performs fast floating point operations

Nomenclature:

ARM processor implementation is described by the product nomenclature as given below

ARM [x][y][z][T][D][M][I][E][J][F][-S]

- x - Family
- y - Memory Management.
- z - Cache size
- T- Thumb state
- D - JTAG Debug option
- M - Fast multiplier
- I - Embedded ICE Macrocell

- E - Enhanced instructions
- J - Jazelle state (Java)
- F - Vector floating point unit
- S - Synthesizable version

Referring to the nomenclature, ARM7TDMI can be understood as an ARM7 processor with thumb implementation, JTAG debug, multiplier and ICE macro cell. Similarly ARM926EJ-S is an ARM9 processor with MMU and cache implementation, enhanced instructions, Jazelle state and has a synthesizable core.

ARM Architecture v7 profile:

In order to provide a wide coverage of different application domains, addressing their specific requirements, ARM core is evolved into architecture version7 which has three different profiles: the Application profile(Cortex-A), Real Time profile(Cortex-R) and Microcontroller profile(Cortex-M). Architecture v7 should not be confused with ARM7 which has been explained before under architecture v4.

Application profile (Cortex -A):

Cortex A series of architectures are multicores with power efficiency and high performance. Every Cortex-A implementation is intended for highest performance at ultralow power design. It supports with, in-built memory management unit. Being influenced by multitasking OS system requirements, it has virtualization extensions and provides a trust zone for a safe and extensible system. It has enhanced Java support and provides a secure program execution environment. These architectures are typically designed for high end real time safety critical applications like automotive powertrain system. Some Cortex-A application products are smart phones, tablets, televisions and even high end computing servers.

Real-Time profile (Cortex -R):

Cortex R series of architectures are designed for deeply embedded real time multitasking applications. They have low interrupt latency and predictability features for real time needs. It provides memory protection for supervisory OS tasks being in privileged mode. It also provides tightly coupled memories for fast deterministic access. Typical application examples are: hard disk drive controller and base band controller for mobile applications and engine management unit where high performance and reliability at very low interrupt latency and determinism are critical requirements.

Microcontroller profile (Cortex -M):

Cortex M series of architectures have v6-M as cortex M0, M0+ and M1 and v7-M with Cortex M3, M4 and other successors. This series of architectures developed for deeply embedded microcontroller profile, offer lowest gate count so smallest silicon area. These are flexible and powerful designs with completely predictable and deterministic interrupt handling capabilities by introducing the nested vector interrupt controller (NVIC). The small instruction sets support for high code density and simplified software development. Developers are able to achieve 32-bit performance at 8-bit price. The very low gate count of Cortex M0 facilitates its deployment in analog and mixed mode devices. Due to further

demanding applications requiring even better energy efficiency, Cortex M0+ was designed with two stage pipeline and achieved high performance with very low dynamic power consumption, reduced branch shadow and reduced number of flash memory access. Cortex M1 was designed for implementation in FPGA. It is functionally a subset of Cortex M3 and runs ARM v6 instruction set with OS extension options. It has 32-bit AHB lite bus interface; separate tightly coupled memory interface and JTAG interface to facilitate debug options. It has three stage pipeline implementation and configurable NVIC for reducing interrupt latency.

ARMv7-M architecture:

Key features for ARMv7-M architectures are:

- ❑ Enable implementations with industry excelling power, performance and silicon area constraints with simple pipeline design.
- ❑ Highly predictable and deterministic operation with Single/low cycle instruction execution and minimum interrupt latency with cache less memory design.
- ❑ Exception handlers are standard C/C++ functions align with ARM's programming standard.
- ❑ Debug and software profiling support.
- ❑ Cortex M3 is the first architecture of ARMv7-M profile. Subsequently the architecture was enhanced by DSP extensions and named as Cortex M4. Cortex M3 a general purpose CPU, has optimized debug options for microcontroller applications. It has only Thumb-2 processing core which has blend of ARM 32-bits and Thumb 16 bits instructions which removes the need of ARM-Thumb interworking and offers high code density at high energy efficiency. A hardware divide instruction was introduced in the instruction set and a number of multiply instructions are also available to improve data processing performance. It supports only two modes of operation called thread and handler mode. User programs run in thread mode and exceptions are handled in handler mode which is privileged. All exceptions could be programmed in C/C++. NVIC is part of Cortex-M3 macrocell. It is a 32 bit core with 18 working registers: r0-r7 as low registers, r8-r12 as high registers, three special purpose registers, r13 stack pointer, r14 link register, r15 the program counter, one program status register xPSR and one more stack pointer banked for handler mode. The Cortex-M3 and Cortex-M4 processors also support unaligned data accesses, a feature previously available only in high- end processors. Cortex M4 comes under the nomenclature of ARMv7E-M. It was developed as a high performance digital signal controller with 72 DSP instructions implemented along with Cortex M3 instruction set retained. Single cycle execution of multiply and accumulate instructions provides 45% speed improvement compared to Cortex M3.

Cortex M4 Features:

- ❑ Thumb2 instruction set delivers the significant benefits of high code density of Thumb with 32-bit performance of ARM.
- ❑ Optional IEEE754-compliant single-precision Floating Point Unit.
- ❑ Code-patch ability for memory system updates.
- ❑ Power control optimization by integrating sleep and deep sleep modes.
- ❑ Hardware division and fast multiply and accumulate for SIMD DSP instructions.
- ❑ Saturating arithmetic for noise cancellation in signal processing.
- ❑ Deterministic, low latency interrupt handling for real time-critical applications.
- ❑ Optional Memory Protection Unit(MPU) for safety-critical applications
- ❑ Extensive implementation of debug, trace and code profiling capabilities.

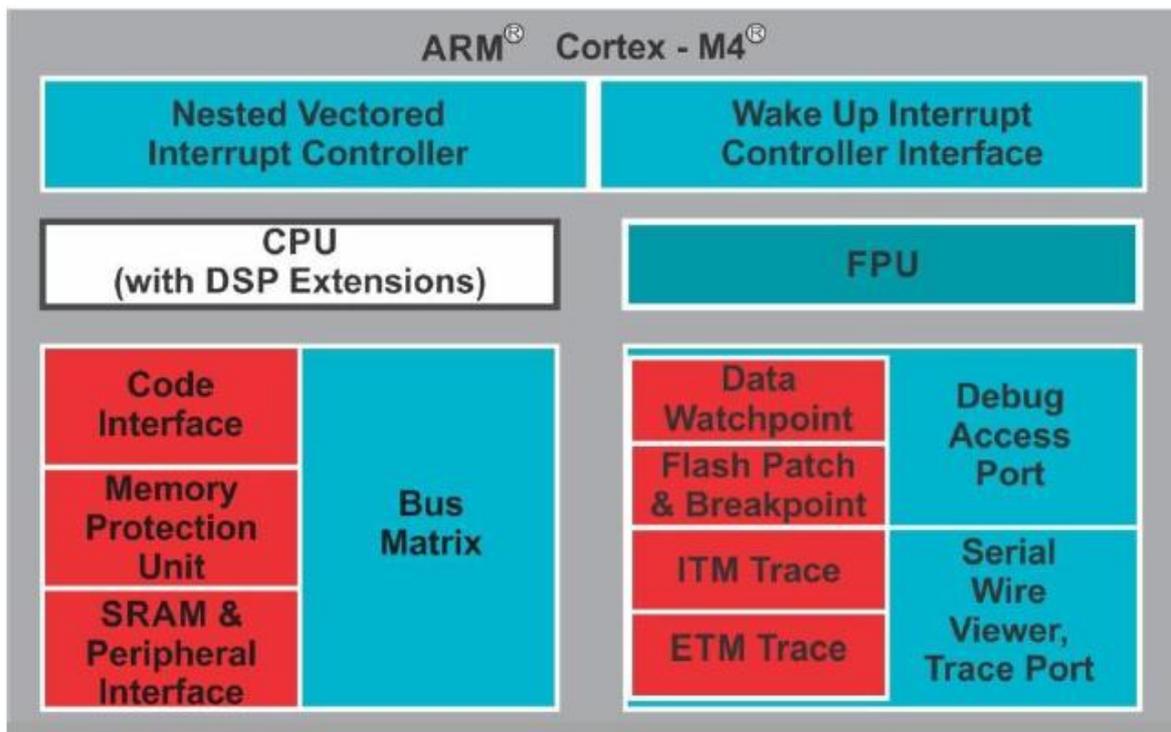


Figure: Cortex M4 Core Architecture

The ARM Cortex-M4 architecture is built on a high-performance processing core, with a 3-stage pipeline. Harvard architecture, optional IEEE754-compliant single-precision floating-point computation, a range of single-cycle and SIMD multiplication and multiply-with-accumulate capabilities, saturating arithmetic and dedicated hardware division features make it typically suitable for high precision digital signal processing applications. The processor delivers excellent energy efficiency at high code density and significantly improving interrupt handling and system debug capabilities. A generic system on chip architecture of Cortex M4 is shown in above figure. The brief description of each functional block is given below.

Nested Vectored Interrupt Controller (NVIC):

Tightly integrated with the processor core, NVIC is a configurable Interrupt Controller used to deliver excellent real time interrupt performance. Very low interrupt latency is achieved through its hardware stacking registers. The processor automatically saves and retrieves its state on exception entry and exit removing the code overhead from ISRs. It also has the ability of interrupting the load and stores multiple atomic instructions that provides faster interrupt response. The NVIC includes a Non Maskable Interrupt (NMI) and can provide up to 256 interrupt priority levels for each of 240 interrupts it supports. A higher priority interrupt can preempt the currently running ISR facilitating interrupt nesting.

Wake Up Interrupt Controller (WIC):

To optimize low-power designs, the NVIC integrates with an optional peripheral called Wake up interrupt controller to implement sleep modes and an optional deep sleep function. When the WIC is enabled, the power management unit powers down the processor and makes it enter deep sleep mode. When the WIC receives an interrupt, it takes few clock cycles to wake-up the processor and restore its state. So it adds to interrupt latency in deep sleep mode. WIC is not programmable and operates completely with hardware signals.

Memory Protection Unit:

In embedded OS, MPU is used for safeguarding memory used for kernel functions from unauthorized access by user program. In OS environment, when any untrusted user program tries to access memory protected by MPU, the processor generates a memory manage fault causing a fault exception. MPU divides the memory map into a number of regions defining memory attributes for each. MPU separates and protects the code, data and stack for each task required for safety critical embedded systems. MPU can be implemented to enforce privilege access rules and separate tasks. It is an optional block in Cortex M4.

Bus Matrix:

The processor contains a bus matrix that arbitrates the processor core and optional Debug Access Port (DAP) memory accesses to both the external memory system, the internal System Control Spaces and to various debug components. It arbitrates requests from different bus masters in the system. Bus matrix is connected to the code interface for accessing the code memory, SRAM and peripheral interface for data memory and other peripherals and the optional MPU for managing different memory regions.

Debug Access Port (DAP):

DAP, the implementation of ARM debug interface enables debug access to various master ports on the ARM SoC. It provides system access for the debugger tool using AHB-AP, APB-AP and JTAG-AP without halting the processor. Embedded Trace Macrocell (ETM) generates instruction trace. Instrumentation Trace Macrocell (ITM) allows software-generated debug messages and also to generate timestamp information. Data Watch point and Trace (DWT) unit can be used to generate data trace, event trace, and profiling trace information. Flash patch and break point (FPB) implements hardware breakpoints, patches code and data from Code space to System space. Serial wire viewer (SWV) is one bit ETM port. SWV provides different types of information like program counter values, data read and write cycles, peripheral values, event counters and exceptions.

Floating Point Unit (FPU):

Cortex M4 architecture suggests an optional FPU which is IEEE 754 single precision compliant. The core instruction set supports various signal processing operations. It executes single instruction multiple data (SIMD) instructions with 16 bit data types. Floating point core supports addition, multiplication and hardware division. It has a 32X32 multiply and accumulate (MAC) unit that produces 64 bit results. Embedded signal processing applications that involve data compression, statistical signal processing, measuring, filtering and compressing real world analog signals can use Cortex M4 with FPU.

Floating point unit supports:

- Conversions between fixed point and floating point data formats and instructions with floating point immediate data.
- Saturation math.
- Decouple 3-stage pipeline.
- Three modes of operations: full compliance mode, flush-to-zero mode and default NaN mode.
- To be disabled when it is not in use to conserve energy.

Introduction to the TM4C family viz. TM4C123x & TM4C129x and its targeted applications:

TIVA TM4C123GH6PM Microcontroller:

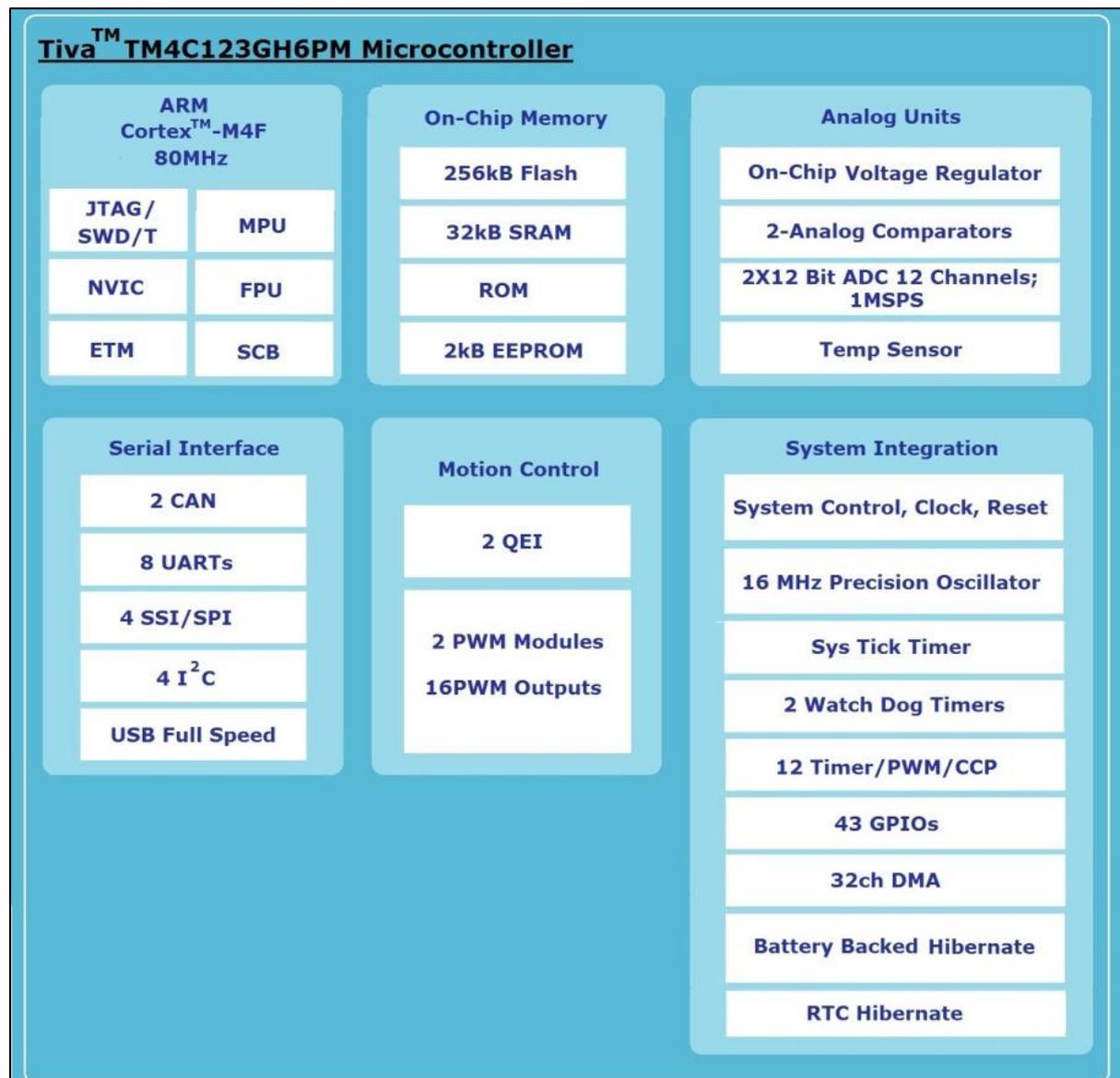


Figure: TIVA TM4C123GH6PM Microcontroller Block Diagram

The TM4C microcontroller block diagram shown in above figure has six functional units. The cortex M4F core, on-chip memory, analog block, serial interface, motion control and system integration.

Features:

- ❑ TM4C123GH6PM microcontroller has 32 bit ARM Cortex M4 CPU core with 80 MHz clock rate.

- ❑ Memory protection unit (MPU) provides protected operating system functionality and floating point unit (FPU) supports IEEE 754 single precision operations.
- ❑ JTAG/SWD/ETM for serial wire debugs and traces (SWD/T).
- ❑ Nested vector interrupt controller (NVIC) reduces interrupt response latency.
- ❑ System control block (SCB) holds the system configuration information.
- ❑ The microcontroller has a set of memory integrated in it: 256 KB flash memory, 32 KB SRAM, 2 KB EEPROM and ROM loaded with TIVA software library and boot loader.
- ❑ Serial communications peripherals such as: 2 CAN controllers, full speed USB controller, 8 UARTs, 4 I2C modules and 4 Synchronous Serial Interface (SSI) modules.
- ❑ On chip voltage regulator, two analog comparators and two 12 channel 12-bit analog to digital converter with sample rate 1 million samples per second (1MSPS) are the analog functions in built to the device.
- ❑ Two quadrature encoder interface (QEI) with index module and two PWM modules are the advanced motion control functions integrated into the device that facilitate wheel and motor controls.
- ❑ Various system functions integrated into the device are: Direct Memory Access controller, clock and reset circuitry with 16 MHz precision oscillator, six 32-bit timers, six 64-bit timers, twelve 32/64 bit captures compare PWM (CCP), battery backed hibernation module and RTC hibernation module, 2 watchdog timers and 43 GPIOs.

Few Applications:

- ❑ Building automation system
- ❑ Lighting control system
- ❑ Data acquisition system
- ❑ Motion control
- ❑ IoT and Sensor networks.

TIVA TM4C129CNCZAD Microcontroller:

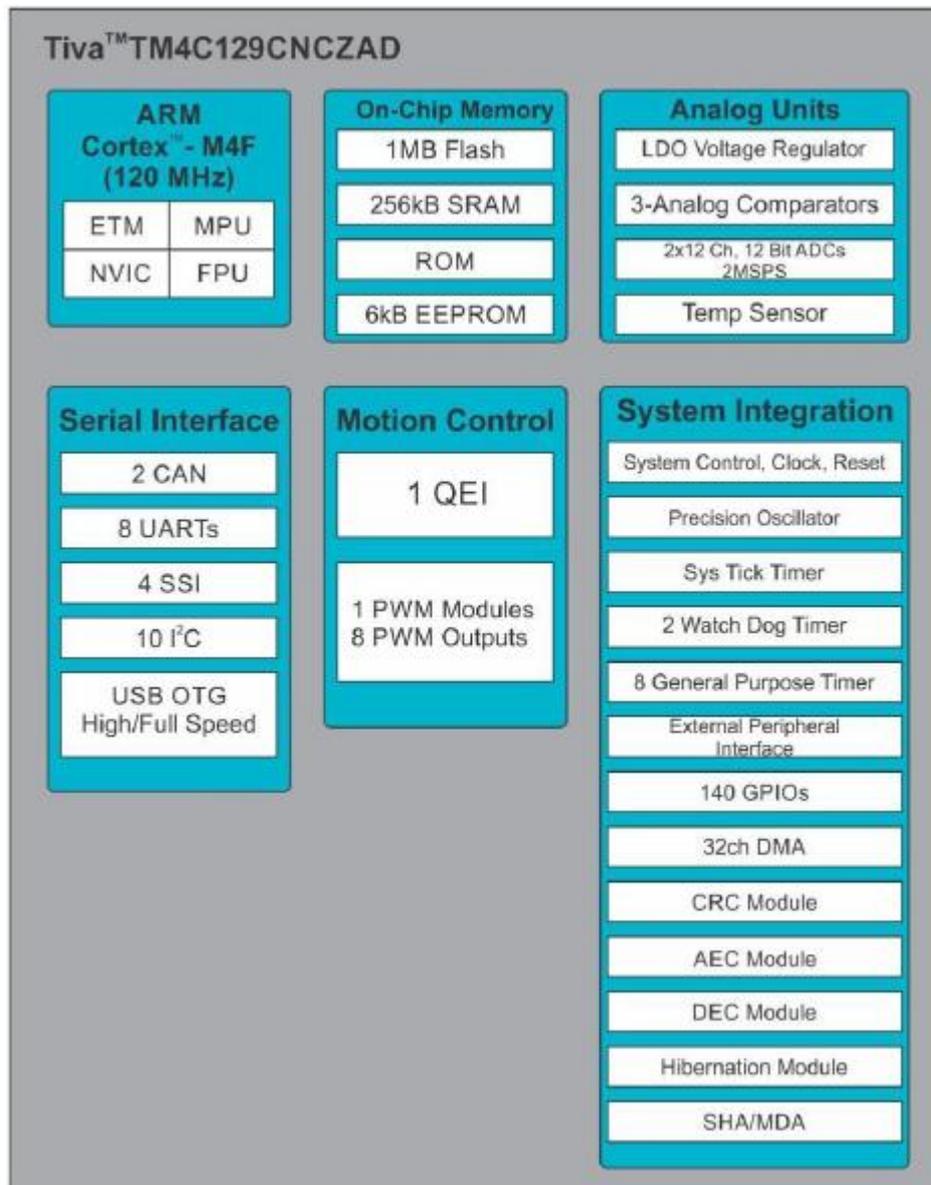


Figure: TIVA TM4C129CNCZAD Microcontroller Block Diagram

Features:

- ❑ TM4C129CNCZAD microcontroller has 32 bit ARM Cortex M4F CPU core with 120 MHz clock rate.
- ❑ Memory protection unit (MPU) provides a privileged mode for protected operating system functionality and floating point unit (FPU) supports IEEE 754 compliant single precision operations.
- ❑ JTAG/SWD/ETM for serial wire debug and trace.
- ❑ Nested vector interrupt controller (NVIC) reduces interrupt response latency and high performance interrupt handling for time critical applications.
- ❑ The microcontroller has a set of memory integrated in it: 1MB flash memory, 256 KB SRAM, 6 KB EEPROM and ROM loaded with TIVA ware software library and boot loader.

- ❑ Serial communications peripherals such as: 2 CAN controllers, full speed and high speed USB controller, 8 UARTs, 10 I2C modules and 4 Synchronous Serial Interface (SSI) modules.
- ❑ On chip voltage regulator, three analog comparators and two 12 channel 12-bit analog to digital converter with sample rate 2 million samples per second (2MSPS) and temperature sensor are the analog functions in built to the device.
- ❑ One quadrature encoder interface (QEI) and one PWM module with 8 PWM outputs are the advanced motion control functions integrated into the device that facilitate wheel and motor controls.
- ❑ Various system functions integrated into the device are: Micro Direct Memory Access controller, clock and reset circuitry with 16 MHz precision oscillator, eight 32-bit timers, and low power battery backed hibernation module and RTC hibernation module, 2 watchdog timers and 140 GPIOs.
- ❑ Cyclic Redundancy Check (CRC) computation module is used for message transfer and safety system checks. CRC module can be used in combination with AES and DES modules.
- ❑ Advanced Encryption Standard (AES) and Data Encryption Standard (DES) accelerator module provides hardware accelerated data encryption and decryption functions.
- ❑ Secure Hash Algorithm/ Message Digest Algorithm (SHA/MDA) provides hardware accelerated hash functions for secured data applications.

Address Space (Memory Map):

A TM4C123GH6PM chip consists of a 256 KB of Flash memory and 32 KB of SRAM. The following figure and table shows the memory map of a TM4C123GH6PM chip with addresses.

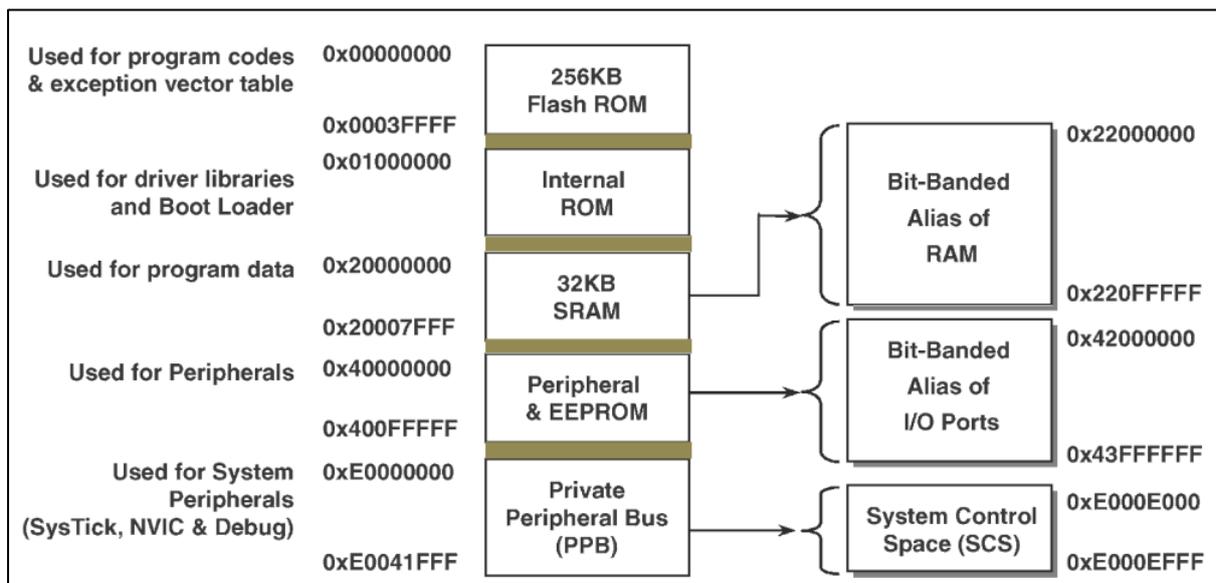


Figure: Memory Mapping in TM4C123GH6PM Chip

Flash Memory:

Flash memory is structured into multiple blocks of single KB size which can be individually written to and erased. Flash memory is used for store program code. Constant data used in a program can also be stored in this memory. Lookup tables are used in many designs for performance improvement. These lookup tables are stored in this memory.

Table: Memory Mapping in TM4C123GH6PM Chip

<i>Memory Type</i>	<i>Allocated size</i>	<i>Allocated address</i>
Flash	256KB	0x00000000 to 0x0003FFFF
Bit-banded on-chip SRAM	32 KB	0x20000000 to 0x20007FFF
Peripheral	All the peripherals	0x40000000 to 0x400FFFFFF

SRAM:

The on-chip SRAM starts at address 0x2000.0000 of the device memory map. ARM provides a technology to reduce occurrences of read-modify-write (RMW) operations called bit-banding. This technology allows address aliasing of SRAM and peripheral to allow access of individual bits of the same memory in single atomic operation. For SRAM, the bit-band base is located at address 0x2200.0000. Bit band alias are computed according to following formula.

$$\text{Bitband alias} = \text{bitband base} + \text{byte offset} * 32 + \text{bit number} * 4$$

Note: Bit banding is the technique to access and modifying content of bits in a register. It is helpful to finish the read-modify operation in single machine cycle.

The region of the memory which device consider for modification is known as bit band region and the region of memory to which device maps the selected memory is known as bit band alias.

The SRAM is implemented using two 32-bit wide SRAM banks (separate SRAM arrays). The banks are partitioned in a way that one bank contains all, even words (the even bank) and the other contains all odd words (the odd bank). A write access that is followed immediately by a read access to the same bank. This incurs a stall of a single clock cycle.

Internal ROM:

The internal ROM of the TM4C123GH6PM device is located at address 0x0100.0000 of the device memory map. The ROM contains:

- TivaWare™ Boot Loader and vector table
- TivaWare™ Peripheral Driver Library (DriverLib) release of product-specific peripherals and interfaces
- Advanced Encryption Standard (AES) cryptography tables

❑ Cyclic Redundancy Check (CRC) error detection functionality

The boot loader is used as an initial program loader (when the Flash memory is empty) as well as an application-initiated firmware upgrade mechanism (by calling back to the boot loader). The Peripheral Driver Library, APIs in ROM can be called by applications, reducing flash memory requirements and freeing the Flash memory to be used for other purposes (such as additional features in the application). Advance Encryption Standard (AES) is a publicly defined encryption standard used by the U.S. Government and Cyclic Redundancy Check (CRC) is a technique to validate if a block of data has the same contents as when previously checked.

Peripheral:

All Peripheral devices, timers, and ADCs are mapped as MMIO in address space 0x40000000 to 0x400FFFFFF. Since the number of supported peripherals is different among ICs of ARM families, the upper limit of 0x400FFFFFF is variant.

Private Peripheral Bus (PPB):

Private Peripheral Bus is used for System Peripheral like System Timer (SysTick), Nested Vectored Interrupt Controller (NVIC), System Control Block (SCB), Memory Protection Unit (MPU), & Floating Point Unit (FPU).

On-Chip Peripherals (Analog and Digital):

SYSTEM PERIPHERALS:

- ⇒ Watchdog Timer
- ⇒ Hibernation Module
- ⇒ DMA
- ⇒ GPIOs
- ⇒ Timers
- ⇒ EEPROM

SERIAL PERIPHERALS:

- ⇒ UART
- ⇒ I2C
- ⇒ CAN
- ⇒ USB OTG
- ⇒ SPI/SSI

ANALOG PERIPHERALS:

- ⇒ Analog Comparator
- ⇒ 12-bit ADC

MOTION CONTROL PERIPHERALS:

- ⇒ PWM
- ⇒ QEI

Watchdog Timer:

Every CPU has a system clock which drives the program counter. In every cycle, the program counter executes instructions stored in the flash memory of a microcontroller. These instructions are executed sequentially. There exist possibilities where a remotely installed system may freeze or run into an unplanned situation which may trigger an infinite loop. On encountering such situations, system reset or execution of the interrupt subroutine remains the only option. Watchdog timer provides a solution to this.

The primary function of the Watchdog Timer is to perform a controlled system restart after a software problem occurs. If the selected time interval expires, a system reset is generated.

Hibernation Module:

This module manages to remove and restore power to the microcontroller and its associated peripherals. This provides a means for reducing system power consumption. When the processor and peripherals are idle, power can be completely removed if the Hibernation module is only the one powered.

To achieve this, the Hibernation (HiB) Module is added with following features:

- (i) A Real-Time Clock (RTC) to be used for wake events
- (ii) A battery backed SRAM for storing and restoring processor state. The SRAM consists of 16 32-bit word memory.

DMA:

Direct Memory Access is a way of streamlining transfers of large blocks of data between two different segments of memory or between an I/O device and memory. Reading from disk and store it in memory is a kind of operation we talking about. For this we may prefer either of two mentioned below:

- ❑ The processor can read each byte at a time from the memory into a register, then store the contents of the register to the suitable memory location. For each byte,
 - ⇒ processor must read an instruction,
 - ⇒ instruction decoding,
 - ⇒ read the data,
 - ⇒ execute read for next part of instruction,
 - ⇒ decode the instruction,
 - ⇒ Store the data.

Then the process starts over again for the next byte.

The second option is a special device, called a DMA controller (DMAC), performs high-speed transfers between memory and I/O devices. It is typically used in moving large sized data clusters around the system. Using DMAC, we can bypass the processor by creating a channel between the memory and the I/O device. Thus, data is read from the I/O device and written into memory without executing the code to perform the transfer on a byte-by-byte basis.

Programmable GPIOs:

General-purpose input/output (GPIO) pins offer flexibility for a variety of connections. The TM4C123GH6PM GPIO module is comprised of six physical GPIO blocks, each corresponding to an individual GPIO port. The GPIO module is FiRM-compliant (compliant to the ARM Foundation IP for Real-Time Microcontrollers specification) and supports 0-43 programmable input/output pins. The number of GPIOs available depends on the peripherals being used.

- Up to 43 GPIOs, depending on configuration
- Highly flexible pin muxing allows use as GPIO or one of several peripheral functions
- Fast toggle capable of a change every clock cycle for ports on Advanced High-Performance Bus (AHB), every two clock cycles for ports on Advanced Peripheral Bus (APB)
- Programmable control for GPIO interrupts
 - Interrupt generation masking
 - Edge-triggered on rising, falling, or both
 - Level-sensitive on High or Low values
- Bit masking in both read and write operations through address lines
- Can be used to initiate an ADC sample sequence or a μ DMA transfer
- Pin state can be retained during Hibernation mode

Timers:

Timers are basic constituents of most microcontrollers. Today, just about every microcontroller comes with one or more built-in timers. These are extremely useful to the embedded programmer – perhaps second in usefulness only to GPIO. The timer can be described as the counter hardware and can usually be constructed to count either regular or irregular clock pulses. Depending on the above usage, it can be a timer or a counter respectively.

The TM4C123GH6PM General-Purpose Timer Module (GPTM) contains six 16/32-bit GPTM blocks and six 32/64-bit Wide GPTM blocks. These programmable timers can be used to count or time external events that drive the Timer input pins. Timers can also be used to trigger μ DMA transfers, to trigger analog-to-digital conversions (ADC) when a time-out occurs in periodic and one-shot modes.

Serial Communications Peripherals:

The TM4C123GH6PM controller supports both asynchronous and synchronous serial communications with:

- Two CAN 2.0 A/B controllers
- USB 2.0 OTG/Host/Device
- Eight UARTs
- Four I2C modules with four transmission speeds including high-speed mode
- Four Synchronous Serial Interface modules (SSI)

Analog Comparators:

An analog comparator is a peripheral that compares two analog voltages and provides a logical output that signals the comparison result. The TM4C123GH6PM microcontroller provides two independent integrated analog comparators that can be configured to drive an output or generate an interrupt or ADC event.

The comparator can provide its output to a device pin, acting as a replacement for an analog comparator on the board, or it can be used to signal the application via interrupts or triggers to the ADC to cause it to start capturing a sample sequence. The interrupt generation and ADC triggering logic is separate. This means, for example, that an interrupt can be generated on a rising edge and the ADC triggered on a falling edge.

Analog to Digital Converter (ADC):

ADCs are peripherals that convert a continuous analog voltage to a discrete digital number. In order to convert to digital, the signal is sampled at higher frequencies to minimize the signal loss. Then the amplitude at those sampled moments is converted with respect to their quantization level. Finally these levels and moments are entitled to a unique code, which are simply the combinations of 0's and 1's – this is called encoding.

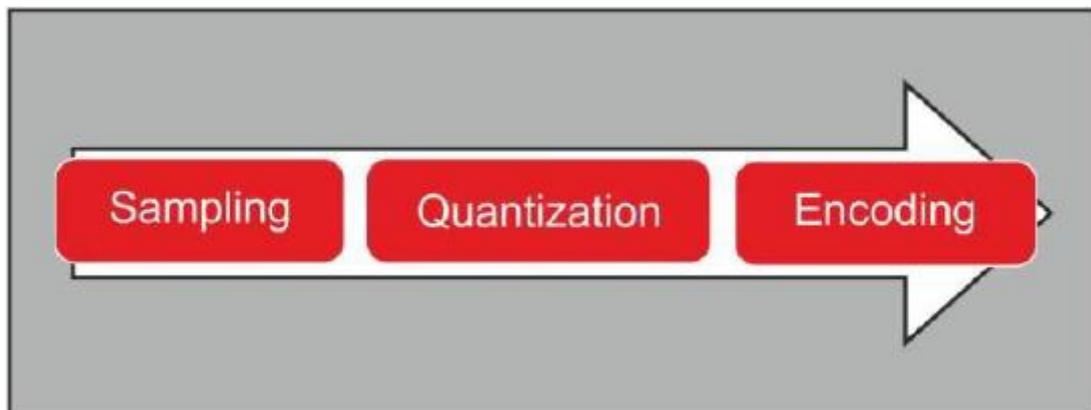


Figure: Block Diagram of working of an ADC

Pulse width modulation (PWM):

Pulse width modulation (PWM) is a simple but powerful technique of using a rectangular digital waveform to control an analog variable or simply controlling analog circuits with a microprocessor's digital outputs. PWM is employed in a wide variety of applications, from measurement & communications to power control and conversion.

Quadrature Encoder Interface (QEI):

A quadrature encoder, also known as a 2-channel incremental encoder, converts linear displacement into a pulse signal. By monitoring both the number of pulses and the relative phase of the two signals, you can track the position, direction of rotation, and speed. In addition, a third channel, or index signal, can be used to reset the position counter.

A classic quadrature encoder has a slotted wheel like structure, to which a shaft of the motor is attached and a detector module that captures the movement of slots in the wheel.

Register Set:

Registers are for temporary data storage within processor architecture. As shown in Figure, ARM processor has sixteen numbers of general purpose registers, R0-R15 and a current program status register (CPSR) defined for user mode of operation. Each of these registers is of 32-bits. Out of these registers, R13, R14 and R15 have special purposes.

R13: Used as the stack pointer that holds the address of the top of the stack in the current processor mode.

R14: Used as the link register that saves the content of program counter on control transfer due to the occurrence of exceptions or using the branch instructions in the program.

R15: Used as the program counter that points to the next instruction to be executed. In ARM state, all instructions are of 32-bits (four bytes) for which, PC is always aligned to a word boundary. This means that the least significant two bits of the PC are always zero. The PC can also be half word (16-bit) aligned for Thumb state (16 bit instructions) or byte aligned for Jazelle state (8-bit instructions) supported by different versions of ARM architecture.

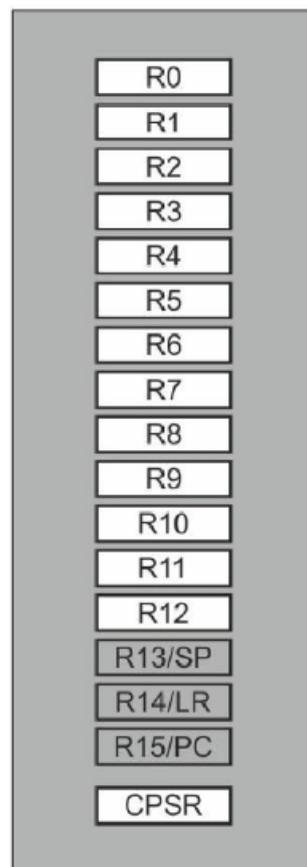


Figure: User Mode Register Set

Current Program Status Register (CPSR):

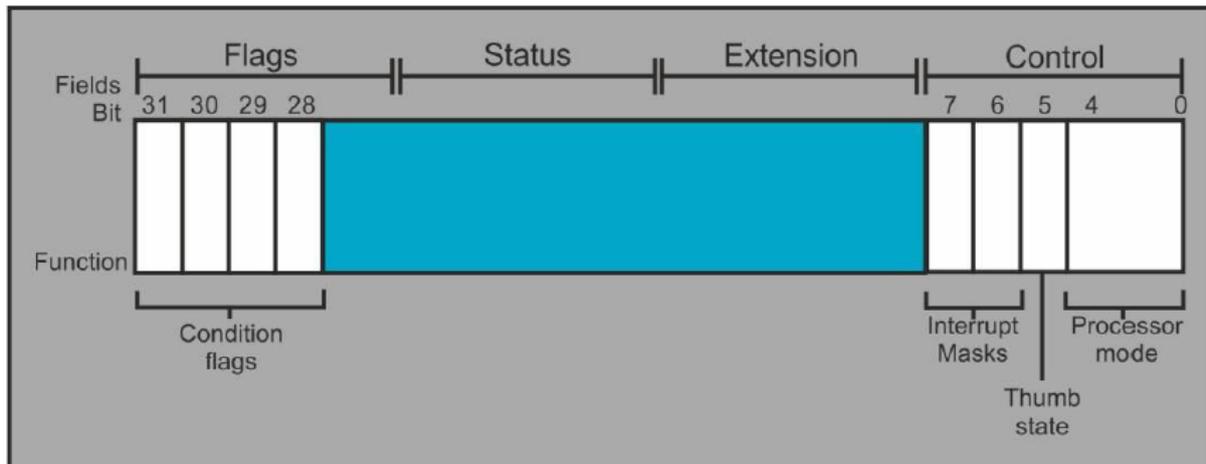


Figure: A Generic CPSR Format

CPSR, a 32-bit status register, holds the current state of the ARM core. As shown in Fig 1.4, the register is divided into four different fields- flags, status, extension and control; each of 8-bits. The flag field has the bit specification for four condition flags; N, Z, C and V and is used for arithmetic and logic instructions.

- N-(Negation flag) = 1 indicates negative result from ALU.
- Z- (Zero flag) = 1 indicates zero result from ALU.
- C- (Carry flag) = 1 indicates ALU operation generated carry.
- V- (Overflow flag) =1 indicates ALU operation overflowed.

Most of the ARM instructions are conditionally executed. Based on the status of these condition flags, condition codes are used along with instruction mnemonics to control whether or not the instruction will be executed. Status and extension fields are reserved for future usage. In the control field, the least significant five bits are used to save the modes of operation of ARM core. Processor mode can be changed by directly modifying these control bits. The most significant three bits I, F and T have significance as below:

- I = 1 indicates IRQ is disabled
0 indicates IRQ is enabled.
- F = 1 indicates FIQ is disabled
0 indicates FIQ is enabled.
- T = 1 indicates the Thumb state is active.
0 indicates ARM state is active.

Operating Modes

ARM core has seven operating modes basically used to isolate users programs from the protected memory or OS services. The operating modes are: user, system, fast interrupt request (FIQ), interrupt request (IRQ), abort, supervisor and undefined mode. Out of these, only user mode is unprivileged, remaining six are privileged modes. The basic difference between privileged and unprivileged mode is the access permission to protected area of the memory and write access permission to CPSR_c given to only privileged modes. All

application programs run in user mode. All operating system kernel functions and services run in system or supervisor mode. After reset, core enters to supervisor mode. FIQ mode is for interrupt requesting faster response and low latency and IRQ mode correspond to the low priority interrupt available on the processor itself. Processor enters abort mode to handle memory access violation. In the execution flow, when processor encounters an instruction that is not supported by the instruction set implementation, it enters to undefined mode. All exceptions are handled in privileged modes. Privileged modes have complete read and write access to both flags and control fields but unprivileged user mode has only read access to the control field while both read and write access to the flags field. Processor mode is changed automatically by the occurrence of exceptions or by modifying the control bits of CPSR by writing its binary pattern as shown in below table, being in a privileged mode.

- **User** : unprivileged mode under which most tasks run
- **FIQ** : entered when a high priority (fast) interrupt is raised
- **IRQ** : entered when a low priority (normal) interrupt is raised
- **Supervisor** : entered on reset and when a Software Interrupt instruction is executed
- **Abort** : used to handle memory access violations
- **Undef** : used to handle undefined instructions
- **System** : privileged mode using the same registers as user mode

Table: Processor mode with binary Pattern mode Control bits [4:0]

Abort (abt)	10111
Fast interrupt request(fiq)	10001
Interrupt request (irq)	10010
Supervisor(svc)	10011
System (sys)	11111
Undefined(und)	11011
User(usr)	10000

Programming Model:

Programming model of a processor is basically a set of working registers used to perform the operations defined in its instruction set. ARM programming model has total 37 registers in its register bank which are segmented for different modes of operation as shown in below figure. User mode register set is shared by the system mode also.

Each of the remaining privileged modes has a set of banked registers which are active and accessible to the programmer only when the core enters to the corresponding mode. Banked registers for a particular mode are physical replication of few of the user mode registers along with a saved program status register (SPSR) shown by shading in the figure.

If the processor mode is changed, for example from user to FIQ mode due to occurrence of hardware interrupt (fiq), the banked registers R8-R14 from the FIQ mode will replace the corresponding registers in user mode but the remaining user mode registers (R0-R7) can still be used in FIQ mode after saving the previous contents.

It means registers R8-R14 of user mode are unaffected by this mode change. The purpose of these banked registers is to reduce the context saving overhead. There is only one dedicated PC (R15) and one CPSR for all the operation modes.

When a mode is changed, the PC and CPSR contents are saved in the link register (R14) and SPSR of the new mode respectively. While returning back to previous mode, special instructions are used to restore back the saved register contents. There is no SPSR available in user mode and one important feature is that, when a mode change is forced, CPSR content is not saved in SPSR. It happens only when exception occurs.

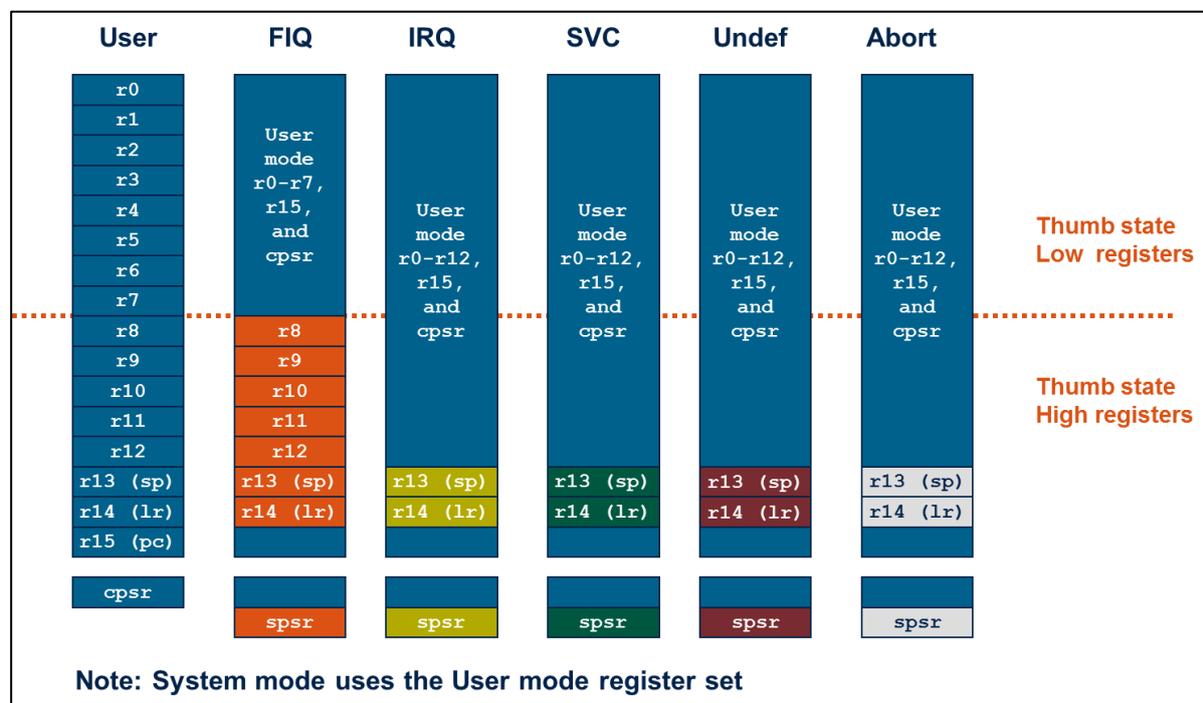


Figure: Complete Register Bank

Addressing Modes:

Addressing mode is the way of addressing data or operand in the instruction. Every processor instruction set offers different addressing modes to determine the address of operands. Some fundamental addressing modes used by most of the processors are: register addressing, immediate addressing, direct addressing and register indirect addressing. In register addressing mode, the operand is held in a register which is specified in the instruction. In immediate addressing mode, the operand is held in the instruction. In direct addressing mode, the operand resides in the memory whose address is specified in the instruction. Similarly in register indirect addressing mode, the operand is held in the memory whose address resides in a register that is specified in the instruction.

The ARM supports the following addressing modes:

- 1) Register Addressing Mode
- 2) Relative Addressing Mode
- 3) Immediate Addressing Mode
- 4) Register Indirect Addressing
- 5) Register Offset Addressing Mode
- 6) Register based with Offset Addressing Mode
 - Pre-Indexed Addressing
 - Pre-Indexed with write back
 - Post-Indexed

1) **Register Addressing:** The operands are in the registers.

Ex: MOV R1, R2 // move content of R2 to R1
SUB R0, R1, R2 //subtract content of R2 from R1 and move the result to R0

2) **Relative Addressing:** Address of the memory directly specified in the instruction.

Ex: BEQ LOOP // branch to LOOP if previous instruction sets the zero flag i.e., Z=1

3) **Immediate Addressing:** Operand2 is an immediate value.

Ex: SUB R0, R0, #1 // Save (R0 -1) to R0
MOV R0, #0xFF00 // Put 0xFF00 to R0

4) **Register Indirect Addressing:** Address of the memory location that holds the operands there in a register.

Ex: LDR R1, [R2] //Load R1 with the data pointed by register R2.
ADD R0, R1, [R2] //add R1 with the data pointed by R2 and put the result into R0

5) **Register Offset Addressing:** Operand2 is in a register with some offset calculation.

Ex: MOV R0, R2, LSL #3 // (R2 << 3), then move to R0
AND R0, R1, R2, LSR R3 // (R2 >> R3), logically AND with R1 and move result to R0

6) **Register based with Offset Addressing:** Effective memory address has to be calculated from a base address and an offset. Offset can be an immediate offset, register offset or scaled register offset.

1) **Pre-Indexed Addressing**

Ex: LDR R2, [R3, #08] // Take value in R3, add to 08, use it as address and load data from that address to R2
STR R1, [R0, -R2] // Register offset // Use (R0-R2) as address of the memory and store data of R1 to that address.
LDR R3, [R1, R2 LSR #8] // Scaled register offset // Use (R1+ (R2>>8)) as address and load the data from that address to R3.

2) **Pre-Indexed with write back** also called auto-indexing with pre-indexed addressing. Symbol indicates that the instruction saves the calculated address in the base address register.

Ex: LDR R0, [R1, #4]! // Immediate offset // Use (R1+4) as address and load the data from that address to R0 and update R1 by (R1+4)

STR R1, [R2, R0]! // Register offset // Use (R2+R0) as address and store the data from R1 to that address. Update R2 by (R2+R0)

STR R3, [R1, R2 LSL #4]! // Scaled register offset // Use (R1+ (R2<<4)) as address and store the data from R3 to that address. Update R1 by (R1+ (R2<<4))

3) **Post-Indexed** also called auto-indexing with post-indexed addressing.

Ex: LDR R0, [R1], #4 // Immediate offset // Load the data pointed to by R1 to R0 and then update R1 by (R1+4).

STR R1, [R3], R4 // Register offset // Store the data in R1 to the memory location pointed to by R3 and then update R3 by (R3+R4)

LDR R2, [R0], -R3, LSR #4 // Scaled register offset // Load the data from the address pointed to by R0 to R2 and then update R0 to (R0-(R3>>4)).

Instruction Set Basics:

In any processor architecture, an instruction includes an opcode that specifies the operation to perform, such as add contents of two registers or move data from a register to memory etc., with specified operands, which may specify registers, memory locations, or immediate data.

ARM architecture has two instruction sets. The ARM instruction set and Thumb instruction set. In ARM instruction set, all instructions are 32 bits wide and are aligned at 4-bytes boundaries in memory. On the other hand, in thumb instruction set, all instructions are of 16 bits wide and are aligned at even or two bytes boundaries in memory.

ARM Instructions can be categorized into following broad classes:

- 1) Data movement instructions
- 2) Data Processing Instructions
 - ⇒ Arithmetic/logic Instructions
 - ⇒ Barrel shifting instructions
 - ⇒ Comparison Instructions
 - ⇒ Multiply Instructions
- 3) Branch Instructions
- 4) Load and store Instructions
 - ⇒ Load and Store register instruction
 - ⇒ Load and Store multiple register instructions
 - ⇒ Stack instructions
 - ⇒ Swap register and memory content
- 5) Program Status register Instructions
 - ⇒ Set the values of the conditional code flag
 - ⇒ Set the values of the interrupt enable bit
 - ⇒ Set the processor mode

- 6) Exception generating Instructions
 ⇒ Software Interrupt Instruction
 ⇒ Software Break Point instruction

Table: Instruction Set Table

Instruction Mnemonic	Description	Example	Working
1) Data Movement instructions			
Syntax: <instruction>{<condition>}{S} Rd, N			
MOV	Move a 32-bit value into a register	MOV r1, r2, LSL #4	Move (r2<<4) to r1.
MVN	Move the NOT of the 32-bit value into a register	MVN r1, r3	Move (~ r3) to r1.
2) Data Processing instructions			
i) Arithmetic Instructions; Syntax:<instruction>{<cond>}{S} Rd, Rn, N			
ADC	Add two 32-bit values and carry	ADC r1, r2, r3	r1= r2+r3+Carry
ADD	add two 32-bit values	ADD r4, r5, r3, LSR # r1	r4= r5+ (r3>> by r1)
RSC	Reverse subtract with carry of two 32-bit values	RSC r3, r2, r1	r3= r1- r2 - !Carry
RSB	Reverse subtract of two 32-bit values	RSB r3, r2, r1	r3= r1- r2
SBC	Subtract with carry of two 32-bit values	SBC r2,r4, r6	r2=r4-r6- !Carry
SUB	Subtract two 32-bit values	SUB r2,r4, r6	r2=r4-r6
ii) Logical Instructions; Syntax:<instruction>{<cond>}{S} Rd, Rn, N			
AND	logical bitwise AND of two 32-bit values	AND r7, r5, r2	r7= r5 & r2
ORR	logical bitwise OR of two 32-bit values	ORR r6, r4, r1, LSR r2	r6= r4 (r1>>r2)
EOR	logical exclusive OR of two 32-bit values	EOR r5, r1, r2	r5= r1 ^ r2
BIC	logical bit clear (AND NOT)	BIC r3, r1,r4	r3= r1 & ~ r4
iii) Comparison Instructions; Syntax:<instruction>{<cond>} Rn, N			
CMN	Compare negated	CMN r1, r2	Flags set as results of r1+r2
CMP	Compare	CMP r1, # 0XFF	Flags set as results of r1- 0XFF
TEQ	Test for equality of two 32-bit values	TEQ r3, r5	Flags set as results of r3 ^ r5
TST	Test bits of a 32-bit values	TST r1, r2	Flags set as results of r1& r2
iv) Multiply Instructions; Syntax:MLA{<cond>}{S} Rd, Rm, Rs, Rn; MUL{<cond>}{S} Rd, Rm, Rs			
MLA	Multiply and accumulate	MLA r1,r2,r3,r4	r1=(r2*r3)+r4
MUL	Multiply	MUL r3, r7, r6	R3= r7*r6

Instruction Mnemonic	Description	Example	Working
3) Branch instructions			
Syntax: B{<cond>} label; BL{<cond>} label; BX{<cond>} Rm; BLX{<cond>} label Rm			
B	Branch	B label	PC= label
BL	Branch with link	BL label	PC=label and Lr= Address of the next instruction after BL.
BX	Branch exchange	BX r5	PC=r5 & 0Xffffffe and T= r5 & 1
BLX	Branch exchange with link	BLX r6	PC=r6 & 0Xffffffe, T=r6 & 1 and lr= address of the next instruction after BLX.
4) Load/Store Instructions			
i) Single register transfer; Syntax:<LDR STR>{<cond>} Rd, Address			
LDR	Load register from memory	LDR r0, [r2, #0X8]	Load r0 with the content of memory address pointed to by [r2+0X8]
STR	Store register to memory	STR r1, [r4], #0X10	Store r1 into the memory address pointed to by r4 and update r4 by [r4+0X10]
ii) Multiple register transfer; Syntax:<LDM STM>{<cond>}<addressing mode> Rn{!},{registers}; Addressing modes: IA-Increment after; IB-Increment before; DA-Decrement after; DB-Decrement before:- Increment or decrement the memory pointer after or before the data transfer.			
LDM	Load multiple registers from memory	LDMIA r6!, {r2-r4}	r2=[r6]; r3=[r6+4]; r4=[r6+8] and update r6 by [r6+12]
STM	Store multiple registers to memory	STMDB r1!, {r3-r5}	[r1-4]=r5 [r1-8]=r4 [r1-12]=r3 and update r1 by [r1-12]
iii) Stack Operations ; Syntax:<LDM STM><addressing mode> SP{!},{registers}; Addressing modes: FA-Full ascending; FD-Full descending; EA-Empty ascending; ED –Empty descending;			
LDM	Load multiple registers from stack memory	LDMED Sp!, {r1, r3}	r1= [Sp+4] r2=[Sp+8] r3= [Sp+12] and Sp is updated by [Sp+12]

STM	Store multiple registers to stack memory	STMFD Sp!, {r4,r6}	[Sp-4]= r6 [Sp-8]= r5 [Sp-12]=r4 and Sp is updated by [Sp-12]
iv) Swap instruction ; Syntax: SWP{B}{<cond>} Rd,Rm,[Rn]			
SWP	swap a word between memory and a register	SWP/SWPB r0, r1, [r2]	Load a 32 bit word or a byte from the memory address in r2 into r0 and store the data in r1 to the memory address in r2.
SWPB	swap a byte between memory and a register		
5) Program status register instructions			
MRS{<cond>} Rd,<cpsr spsr>;MSR{<cond>} <cpsr spsr>_<fields>,Rm MSR{<cond>} <cpsr spsr>_<fields>,#immediate			
MRS	Move the content of cpsr or spsr to a register.	MOV r1, CPSR	Move the content of CPSR register to r1.
MSR	Move an immediate data or register to a specific field of cpsr or spsr.	MSR CPSR_f, r1	Update the flag field of CPSR by the content in r1.
6) Exception generating instructions			
Software interrupt instruction ; Syntax: SWI{<cond>} SWI_number (immediate 24 bit)			
SWI	Software interrupt for an operating system routine. Change to Supervisor mode. CPSR is saved in SPSR. Control branches to interrupt vector.	SWI 0X123456	Execute software interrupt at 0X123456 in ARM state of the core. T =0 in CPSR.