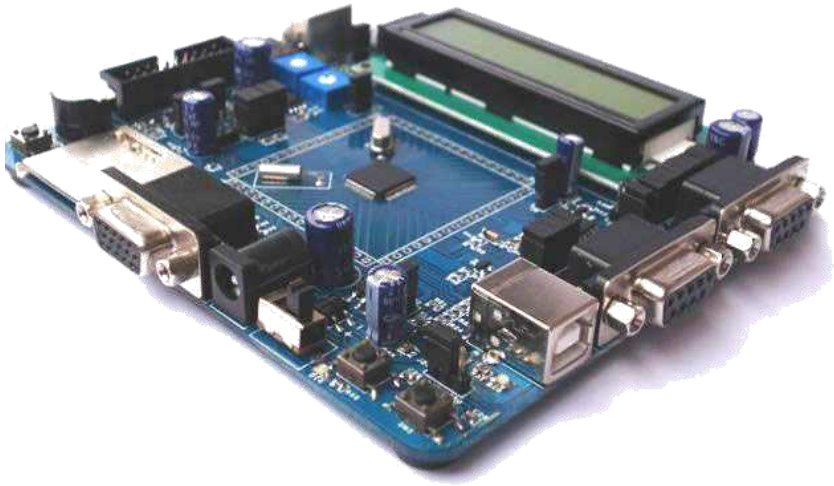
AMIT
Beyond Education

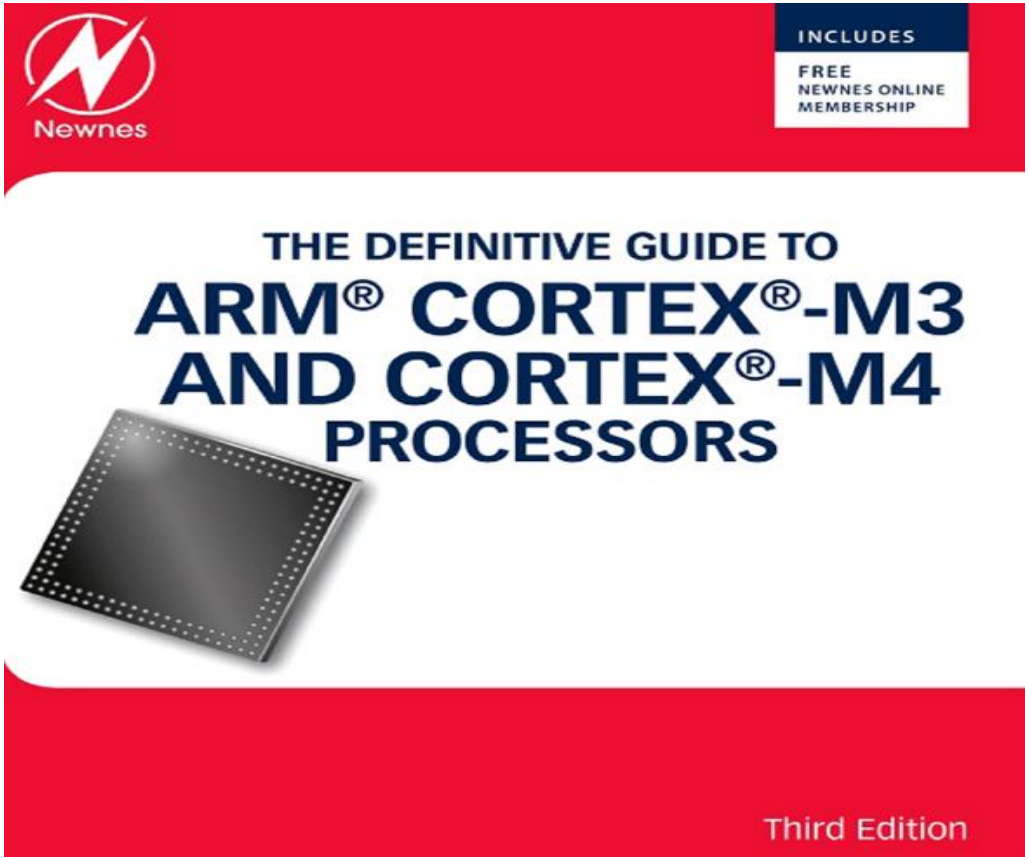
# ARM based Microcontrollers & Peripherals



# Agenda

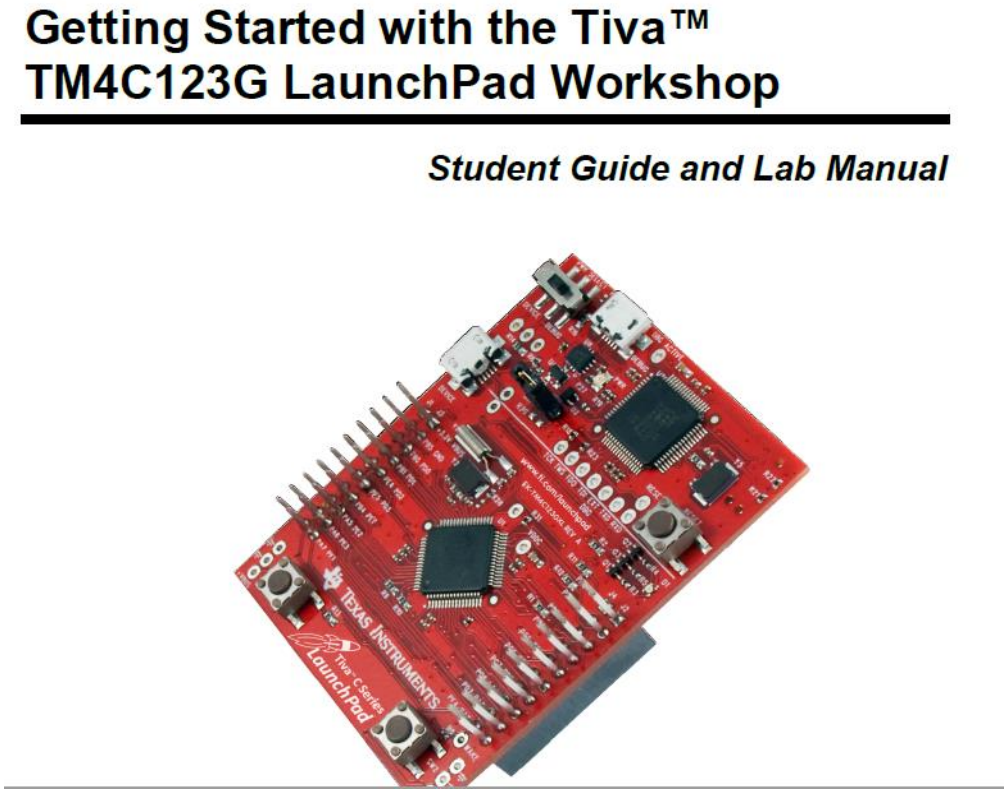
1. Overview on ARM architecture
2. ARM Cortex-M4 and ARM Cortex-M3 Specifications
3. TM4C123GH6PM Microcontroller Peripherals.
4. TIVA TM4C123GH6PM Launchpad kit specifications
5. GPIO Interface with applications
6. ADC Interface with applications
7. Interrupts and exceptions of TM4C123GH6PM.
8. SPI Interface with applications.
9. I2C Interface with applications.
10. UART Interface.
11. DMA and its applications.
12. Timers and PWM interfacing .

# References




Download Link: https://www.mediafire.com/?c7ajy6if2mxsp37

# References




Download Link: https://www.mediafire.com/?3i7y6eun3ewawao

# References


Tiva™ TM4C123GH6PM Microcontroller

DATA SHEET

Download Link: https://www.mediafire.com/?rno9q0eog0o4976

# Intro Video

# Agenda

1. Overview on ARM architecture
2. ARM Cortex-M4 and ARM Cortex-M3 Specifications
3. TM4C123GH6PM Microcontroller Peripherals.
4. TIVA TM4C123GH6PM Launchpad kit specifications
5. GPIO Interface with applications
6. ADC Interface with applications
7. Interrupts and exceptions of TM4C123GH6PM.
8. SPI Interface with applications.
9. I2C Interface with applications.
10. UART Interface.
11. DMA and its applications.
12. Timers and PWM interfacing .

# 1. Overview on ARM architecture

# Outline

- The point consists of the following topics:
  - What is ARM architecture?
  - Development of the ARM Architecture
  - ARM Licensing.
  - ARMv7 architecture profiles

# What is ARM architecture?

- Acronym of **A**dvanced **R**ISC **M**achines.
- It is a 32 – bit microprocessors based on RISC architecture.
- This approach reduces costs, heat and power use by stripping out unneeded instructions and optimizing pathways.
- Later versions of the architecture also support a variable-length instruction set that provides both 32- and 16-bit wide instructions for improved code density.

# History of ARM architecture

- The British computer manufacturer Acorn Computers first developed the Acorn RISC Machine architecture (ARM) in the 1980s to use in its personal computers.

# History of ARM architecture

- Its first ARM-based products were coprocessor modules for the BBC Micro series of computers.

# History of ARM architecture

- After the success of the first BBC Microprocessor they developed the second processor in the series ARM1.


•Clock: 2MHz
•Registers: 16 general purpose, selectively banked
•Pipeline: three-stage
•Cache: none
•Addressing: 26-bit

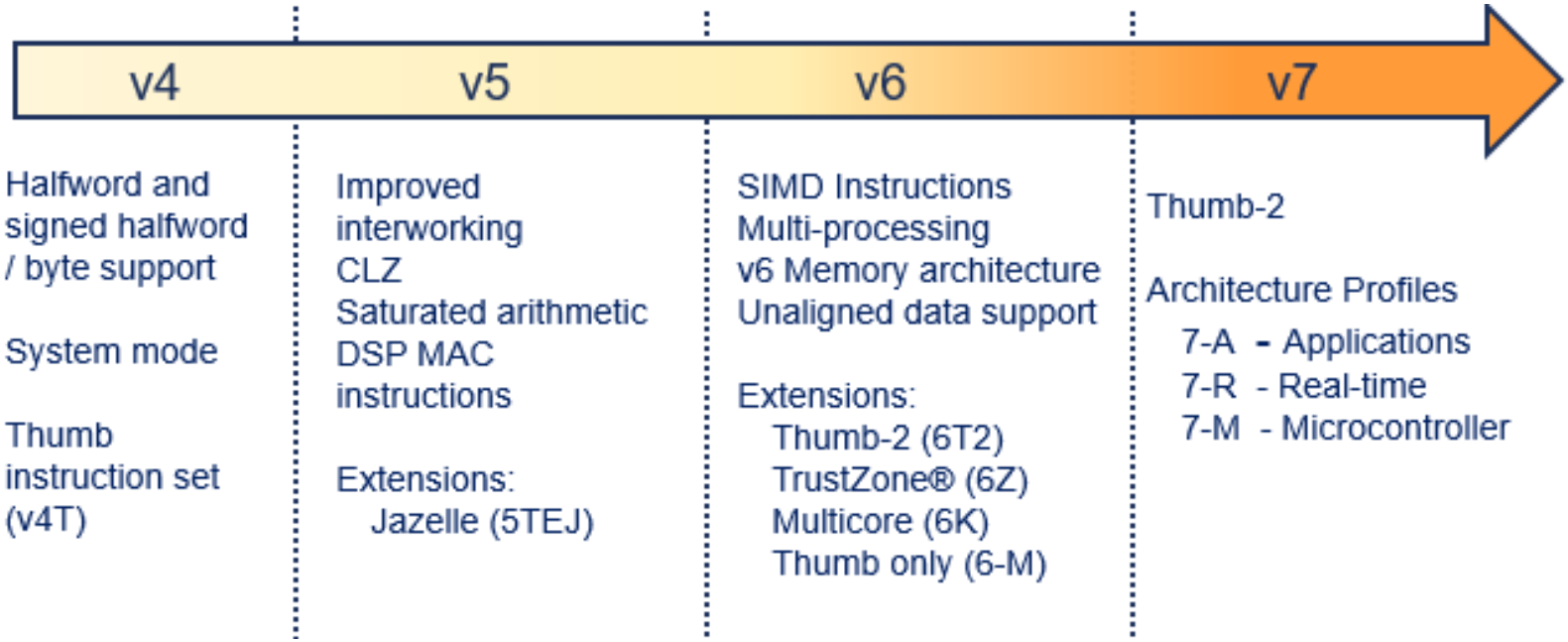
# History of ARM architecture

- The ARM2 featured a 32-bit data bus, 26-bit address space and 27 32-bit registers.
- Eight bits from the program counter register were available for other purposes;
- the top six bits (available because of the 26-bit address space) served as status flags, and the bottom two bits (available because the program counter was always word-aligned) were used for setting modes.

# History of ARM architecture

- In 1990, Acorn spun off the design team into a new company named Advanced RISC Machines Ltd developing ARM6 architecture.
- The address bus was extended to 32 bits in the ARM6, but program code still had to lie within the first 64 MB of memory in 26-bit compatibility mode, due to the reserved bits for the status flags

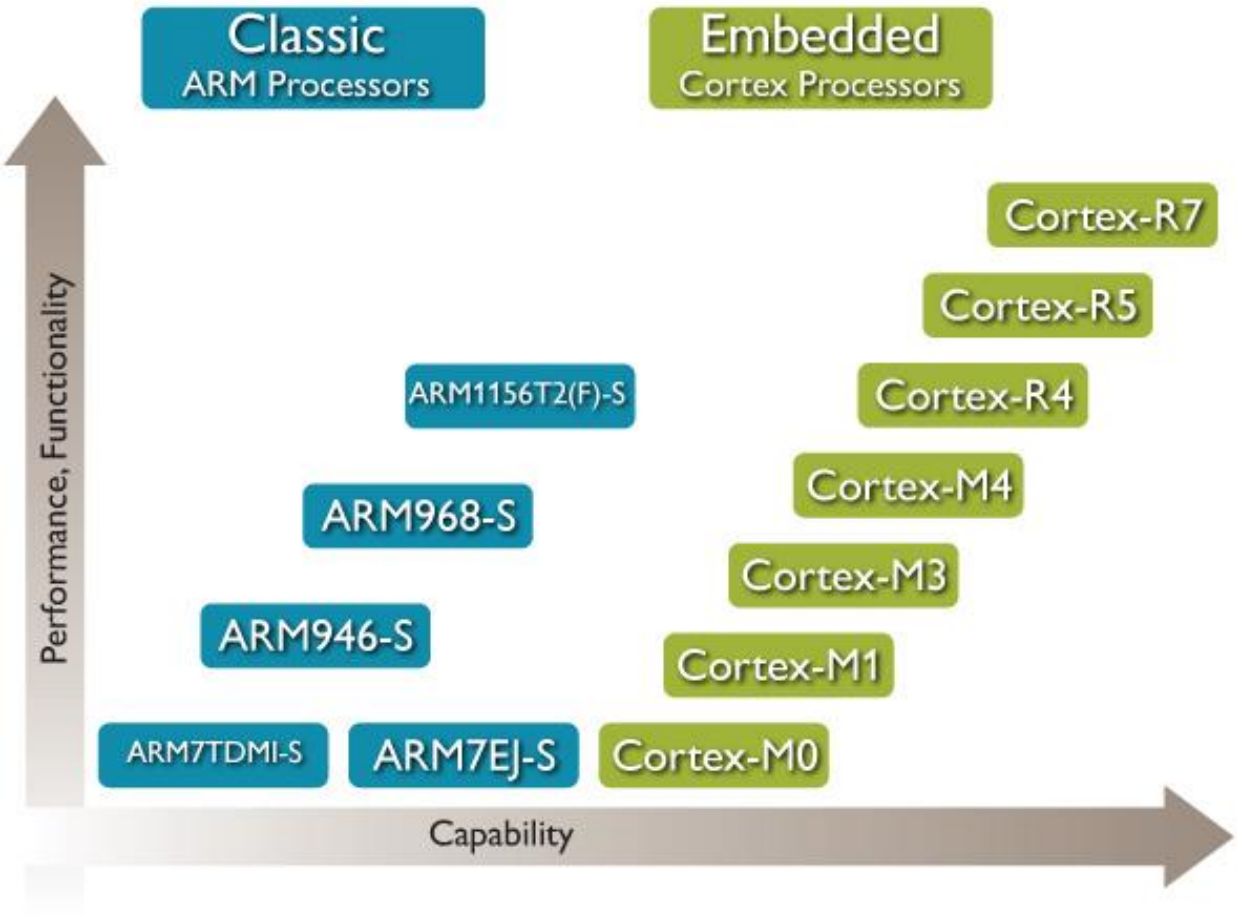
# Development of the ARM Architecture




See the link below for more information:
https://en.wikipedia.org/wiki/List_of_ARM_microarchitectures

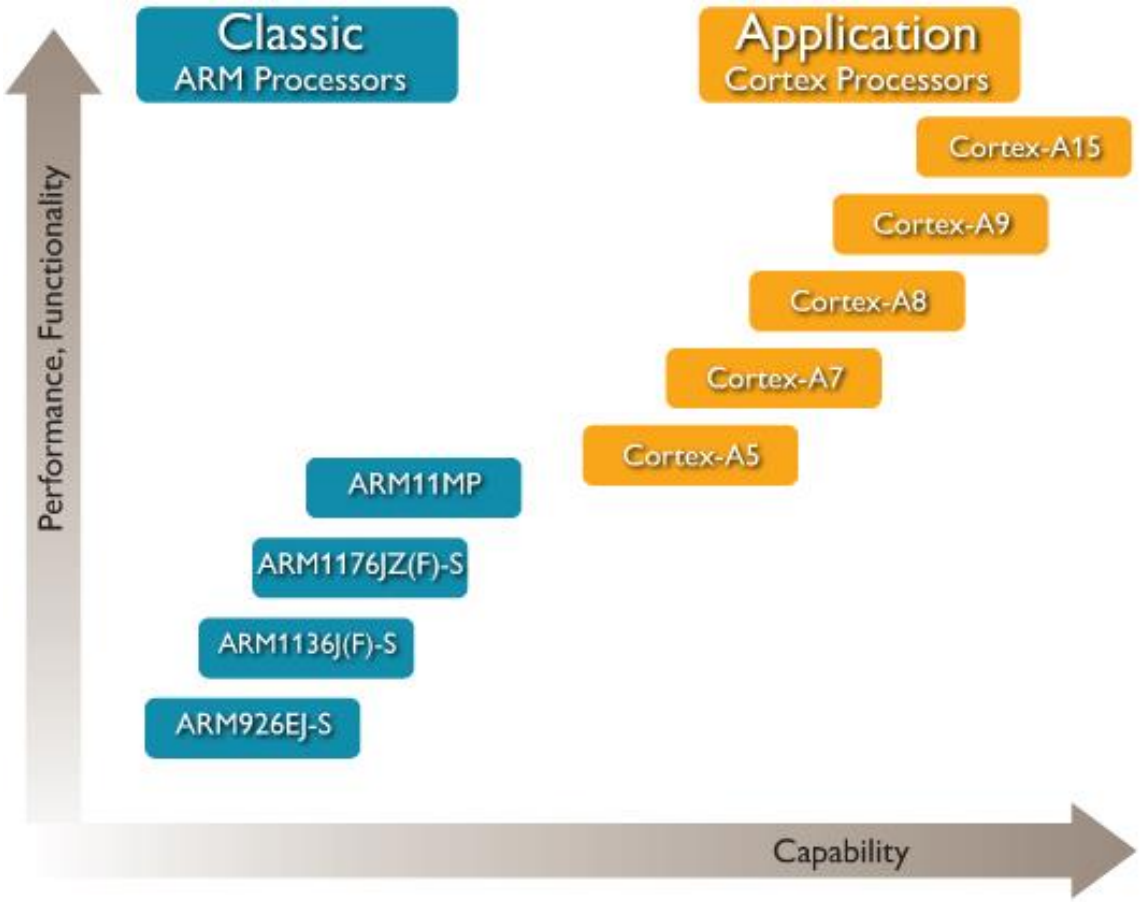
# Development of the ARM Architecture

## Embedded Processors



# Development of the ARM Architecture

## Application Processors

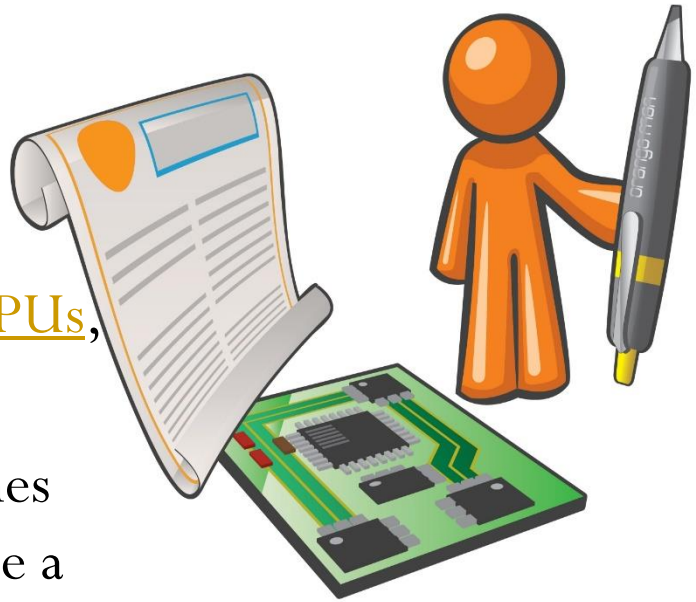


# ARM Licensing

- ARM Holdings Ltd develops the architecture and licenses it to other companies.
- It provides two types of Licenses:

1- Core License:

- which licensees use to create (MCUs), CPUs, and SOCs on those cores.
- The original design manufacturer combines the ARM core with other parts to produce a complete device.



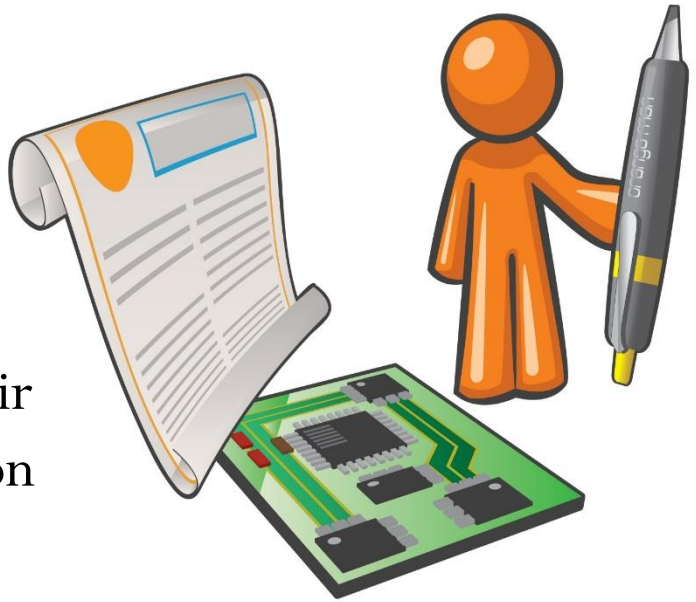
ARM

# ARM Licensing

- ARM Holdings Ltd develops the architecture and licenses it to other companies.
- It provides two types of Licenses:

2- Architecture License:

- Companies can also obtain an ARM *architectural license* for designing their own CPU cores using the ARM instruction sets.
- These cores must comply fully with the ARM architecture.



ARM

# ARMv7 architecture profiles

- Starting of ARMv7, ARM defined three architecture "profiles based on their capabilities and provided applications

1- A-Profile:

- the "Application" profile, implemented by 32-bit cores in the Cortex-A series and by some non-ARM cores.
- Application profiles implement a traditional ARM architecture with multiple modes and support a virtual memory system architecture based on an MMU.
- These profiles support both ARM and Thumb instruction sets.

# ARMv7 architecture profiles

- Starting of ARMv7, ARM defined three architecture "profiles based on their capabilities and provided applications

2- R-Profile:

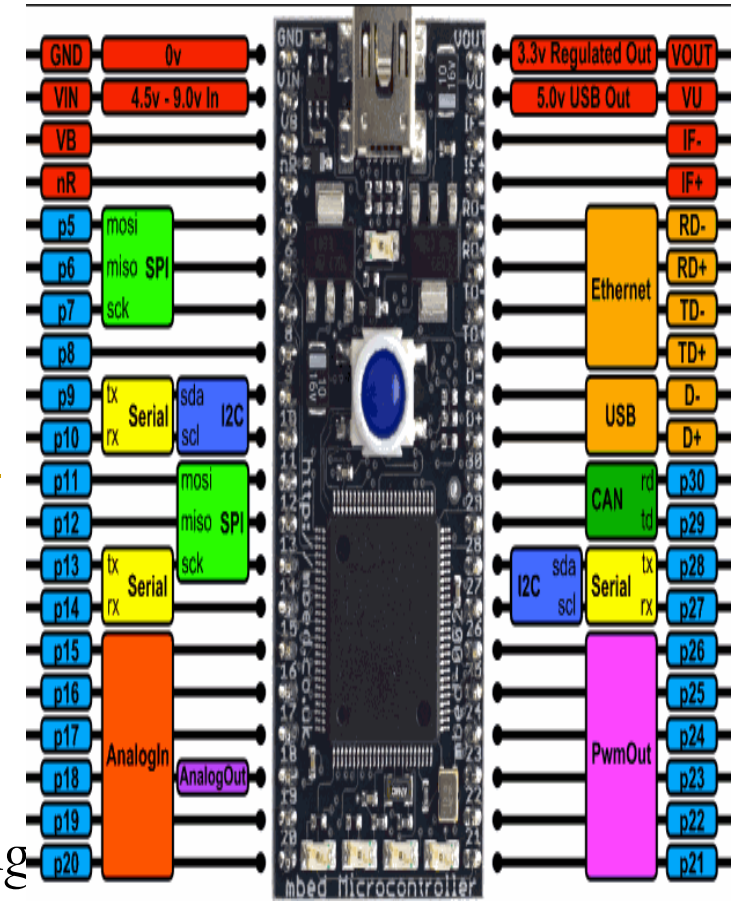
- Real-time profiles implement a traditional ARM architecture with multiple modes and support a protected memory system architecture based on an MPU.
- implemented by cores in the Cortex-R series

# ARMv7 architecture profiles

- Starting of ARMv7, ARM defined three architecture "profiles based on their capabilities and provided applications

3- M-Profile:

- the "Microcontroller" profile, implemented by most cores in the Cortex-M series.
- Microcontroller profiles implement a programmers' model designed for fast interrupt processing, with hardware stacking of registers and support for writing interrupt handlers in high-level languages.



# Agenda

1. Overview on ARM architecture
2. ARM Cortex-M4 and ARM Cortex-M3 Specifications
3. TM4C123GH6PM Microcontroller Peripherals.
4. TIVA TM4C123GH6PM Launchpad kit specifications
5. GPIO Interface with applications
6. ADC Interface with applications
7. Interrupts and exceptions of TM4C123GH6PM.
8. SPI Interface with applications.
9. I2C Interface with applications.
10. UART Interface.
11. DMA and its applications.
12. Timers and PWM interfacing .

# 2. ARM Cortex-M4 and ARM Cortex-M3 Specifications

# Outline

- The point consists of the following topics:
  - The Cortex-M Processor family
  - The Cortex-M3 and Cortex-M4 processors features
  - Cortex-M4 block diagram
  - Cortex-M4 Architecture
  - Cortex-M4 Exceptions and interrupts

# The Cortex-M Processor family

- For general data processing and I/O control tasks, the Cortex-M0 and Cortex-M0þ processors have excellent energy efficiency
- But for applications with complex data processing requirements, they may take more instructions and clock cycles.

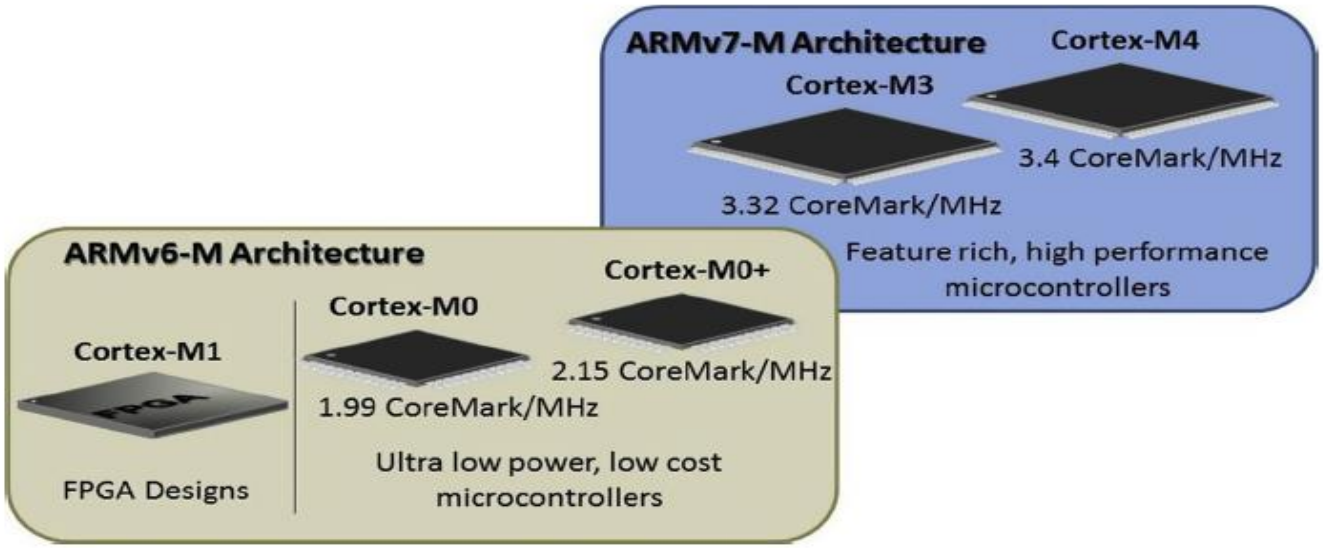



**FIGURE 1.1**

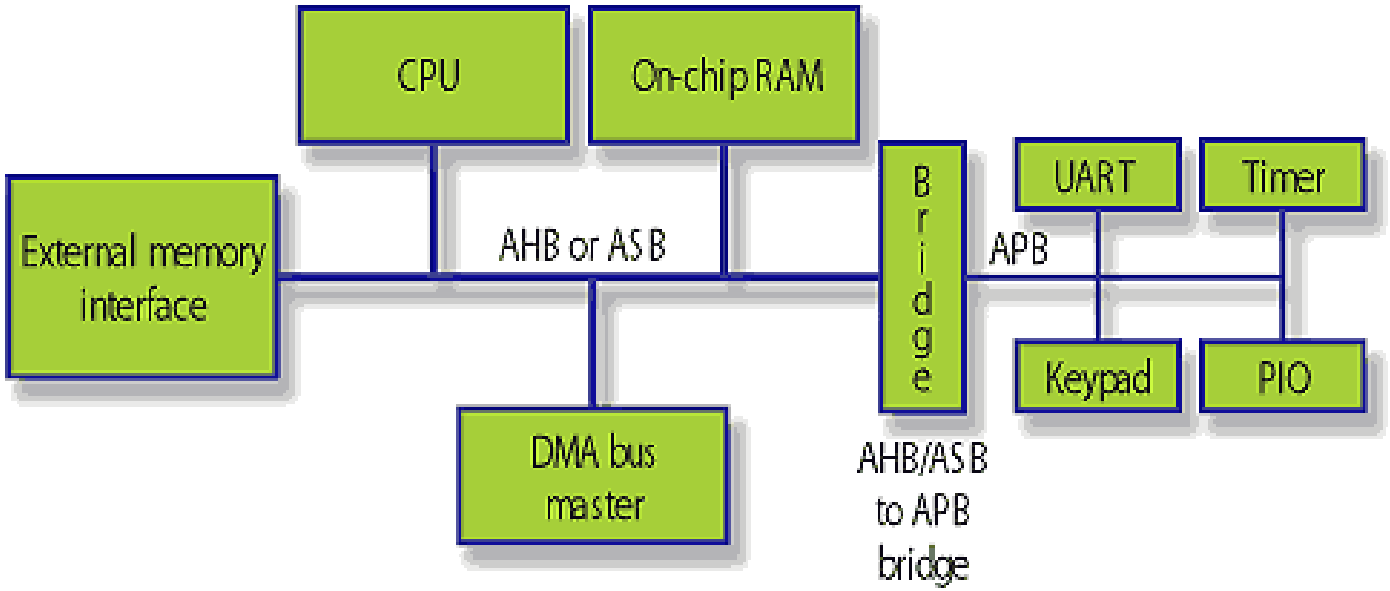
The Cortex-M processor family

# The Cortex-M3 and Cortex-M4 processors features

- Cortex-M3 was released in 2005 and Cortex – M4 was released in 2010.
- 32-bit registers
- 32-bit internal data path
- 32-bit bus interface
- The Instruction Set Architecture (ISA) is called Thumb ISA.
- Three-stage pipeline design
- Harvard bus architecture with unified memory space: instructions and data use the same address space
- 32-bit addressing, supporting 4GB of memory space.

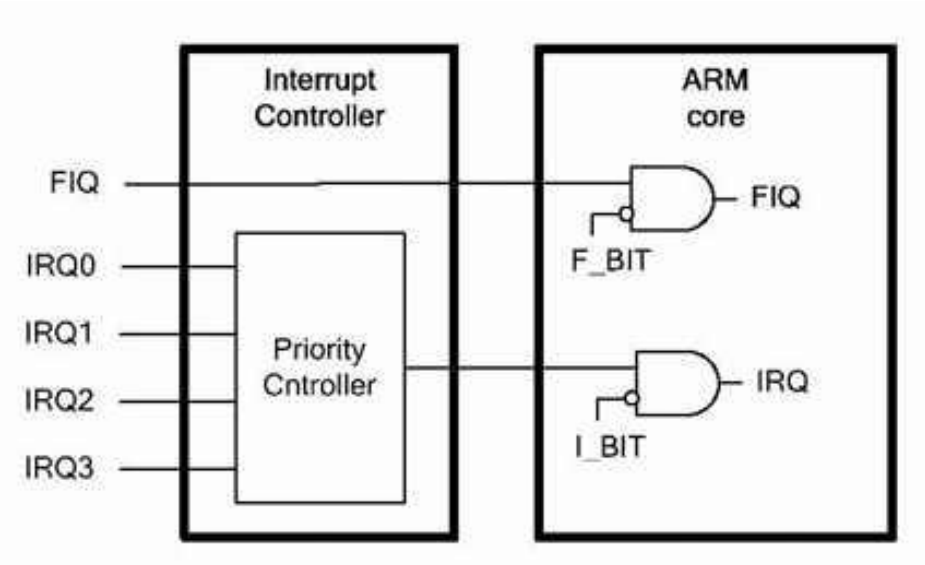
# The Cortex-M3 and Cortex-M4 processors features

- On-chip bus interfaces based on ARM AMBA (Advanced Microcontroller Bus Architecture)Technology, which allow pipelined bus operations for higher throughput



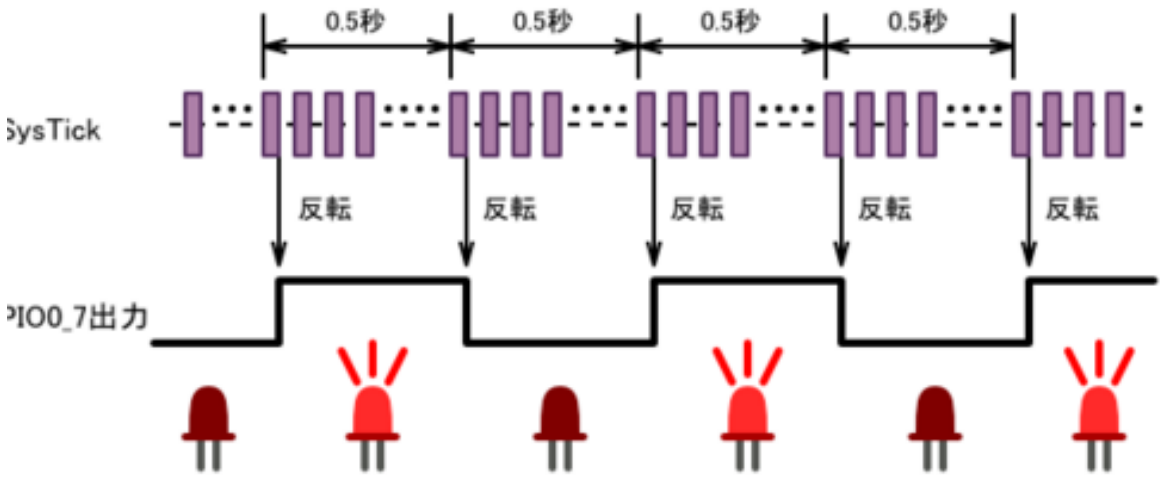
# The Cortex-M3 and Cortex-M4 processors features

- An interrupt controller called NVIC (Nested Vectored Interrupt Controller)
- supporting up to 240 interrupt requests and from 8 to 256 interrupt priority levels (dependent on the actual device implementation)



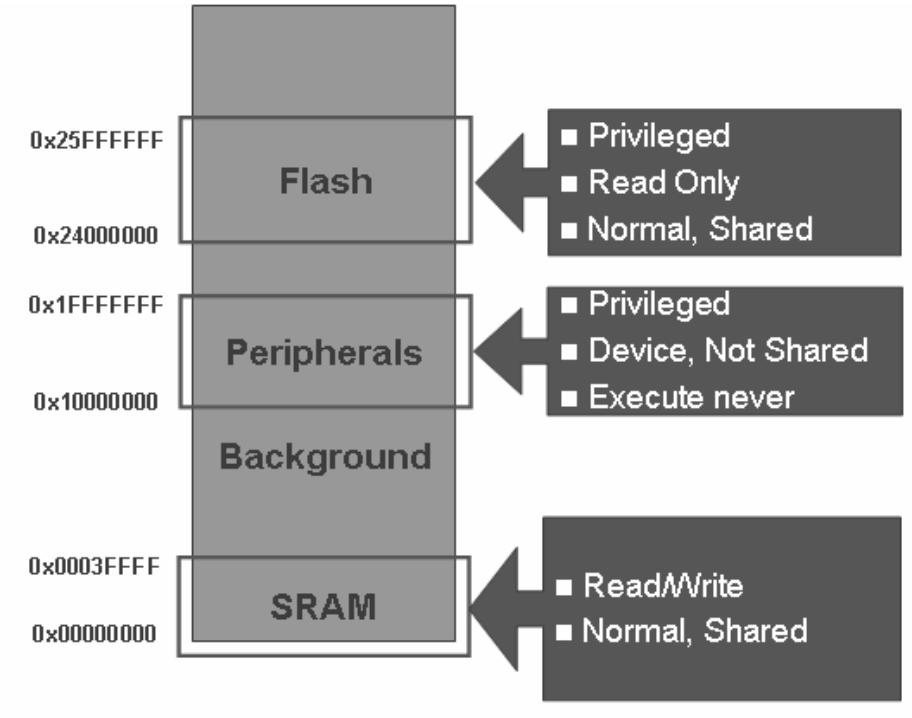
# The Cortex-M3 and Cortex-M4 processors features

- • Support for various features for OS (Operating System) implementation such as a system tick timer, shadowed stack pointer



# The Cortex-M3 and Cortex-M4 processors features

- Sleep mode support and various low power features.
- Support for an optional MPU (Memory Protection Unit) to provide memory protection features like programmable memory, or access permission control.



# Cortex-M4 block diagram

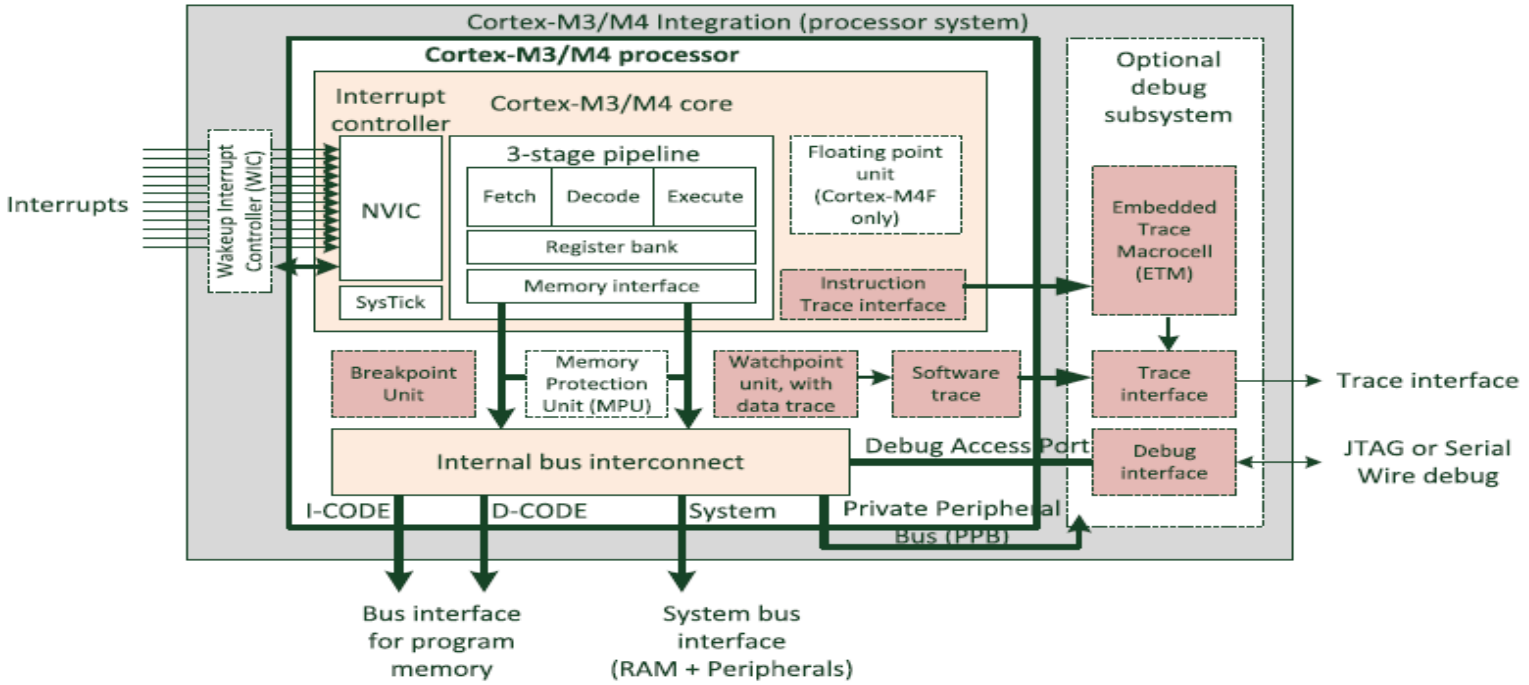



**FIGURE 3.3**

Block diagram of the Cortex-M3 and Cortex-M4 processor

The Cortex-M3 and Cortex-M4 processors are highly configurable. allowing system-on-chip designers to remove any optional components (Like debug support if not required to be supported in MCU).

# Cortex-M4 block diagram

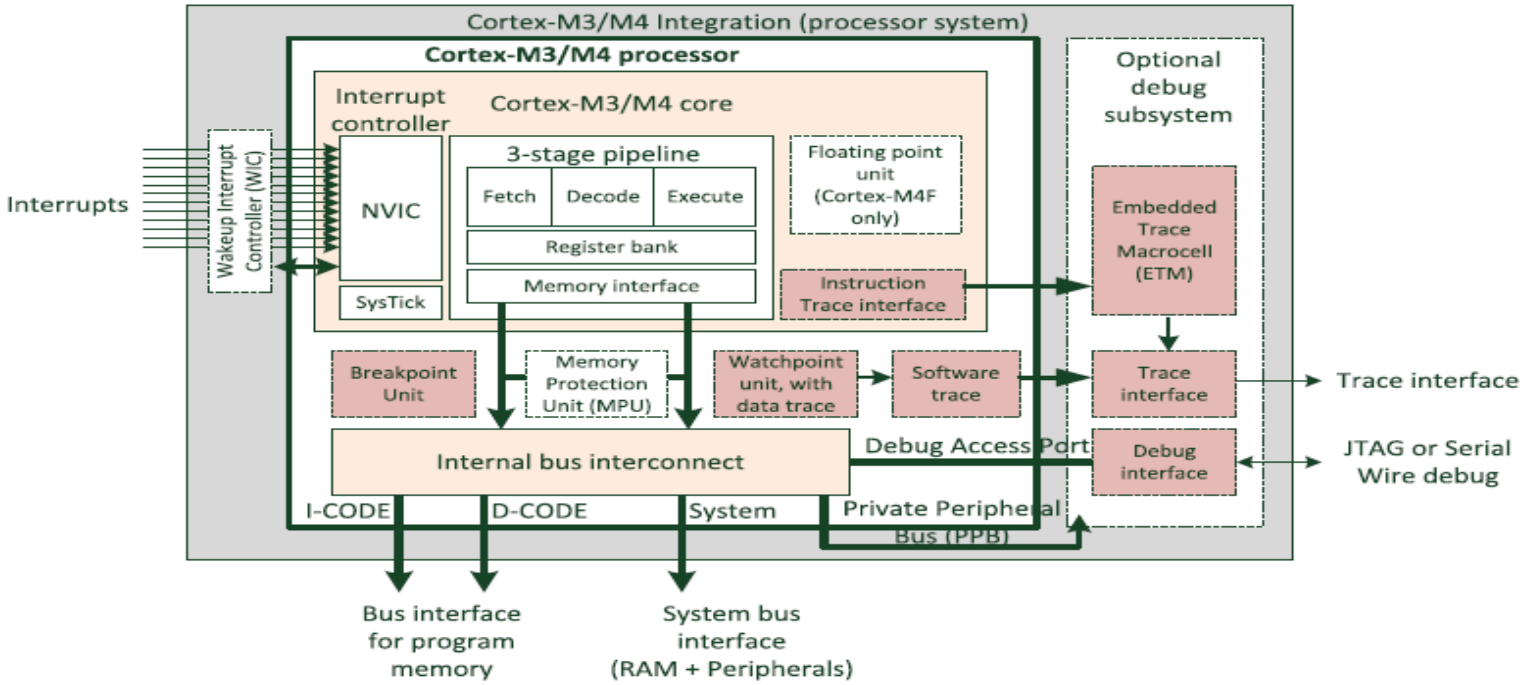



**FIGURE 3.3**

Block diagram of the Cortex-M3 and Cortex-M4 processor

- choose to reduce the number of hardware instruction breakpoint and data watchpoint comparators to reduce the gate count.
- Many system features like the number of interrupt inputs, number of interrupt priority levels supported, and the MPU are also configurable.

# Cortex-M4 block diagram

**Table 3.2** Various Bus Interfaces on the Cortex-M3 and Cortex-M4 Processors

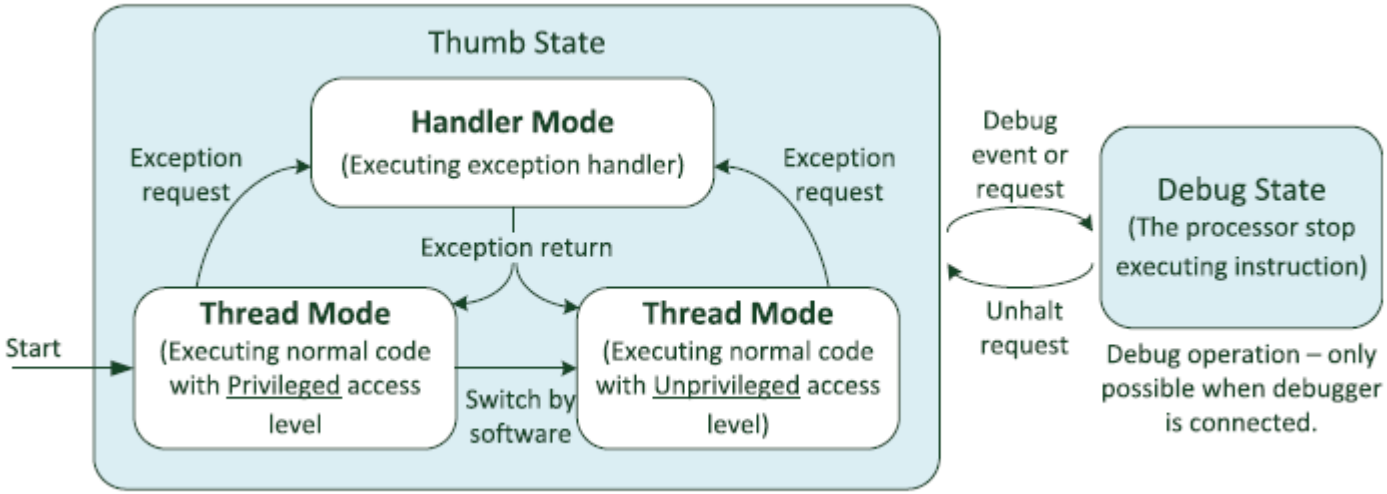
| Bus Interface | Descriptions |
|---|---|
| **I-CODE** | Primarily for program memory: Instruction fetch and vector fetch for address 0x0 to 0x1FFFFFFF. Based on AMBA 3.0 AHB Lite bus protocol. |
| **D-CODE** | Primarily for program memory: Data and debugger accesses for address 0x0 to 0x1FFFFFFF. Based on AMBA 3.0 AHB Lite bus protocol. |
| **System** | Primarily for RAM and peripherals: Any accesses from address 0x20000000 to 0xFFFFFFFF (apart from PPB regions). Based on AMBA 3.0 AHB Lite bus protocol. |
| **PPB** | External Private Peripheral Bus (PPB): For private debug components on system level from address 0xE0040000 to 0xE00FFFFF. Based on AMBA 3.0 APB protocol. |
| **DAP** | Debug Access Port (DAP) interface: For debugger accesses generated from the debug interface module to any memory locations including system memory and debug components. Based on the ARM CoreSight™ debug architecture. |

# Cortex-M4 Architecture

- Programmer model
- the processors can have privileged and unprivileged access levels.
- The privileged access level can access all resources in the processor.
- unprivileged access level means some memory regions are inaccessible, and a few operations cannot be used.
- The processor is also has two operational states:
- Debug state: When the processor is halted (e.g., by the debugger, or after hitting a breakpoint), it enters debug state and stops executing instructions.
- Thumb state: If the processor is running program code (Thumb), it is in the Thumb state.

# Cortex-M4 Architecture

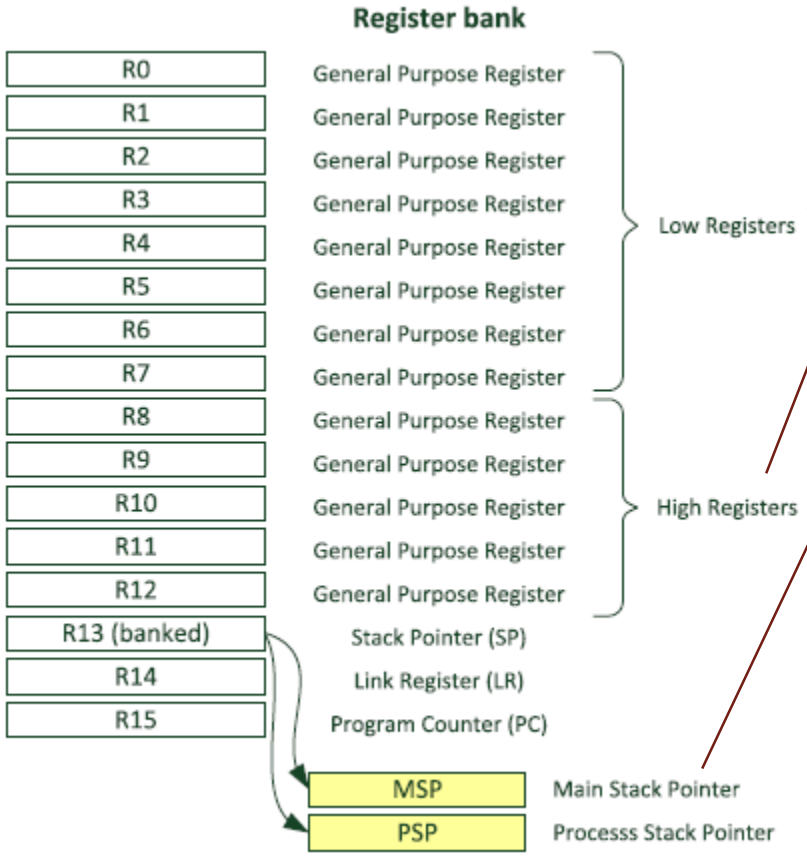
- Programmer model
- In the thumb state, the processor has two operational modes:




- The separation of privileged and unprivileged access levels allows system designers to develop robust embedded systems by providing a mechanism to safeguard memory accesses to critical regions and by providing a basic security model.

# Cortex-M4 Architecture

- Register file




Due to the limited available space in the instruction set, many 16-bit instructions can only access the low registers.

can be used with 32-bit instructions, and a few with 16-bit instructions

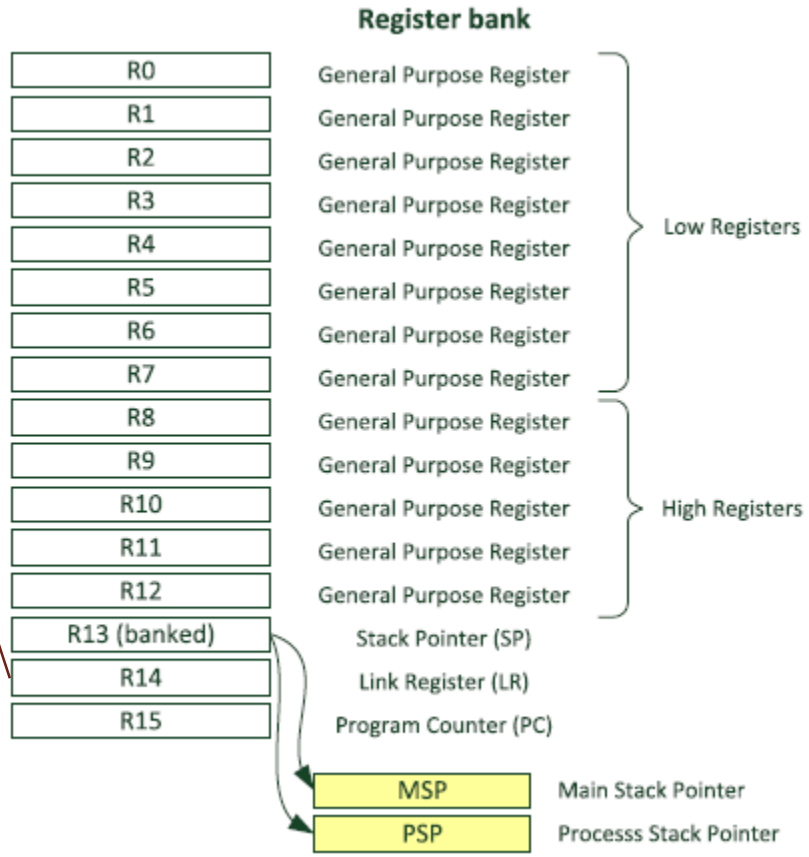
MSP is the default Stack Pointer.
It is selected after reset, or when the processor is in Handler Mode.
PSP can only be used in Thread Mode.
The PSP is normally used when an embedded OS is involved, where the stack for the OS kernel and application tasks are separated.

# Cortex-M4 Architecture

- Register file

Link Register (LR) is used for holding the return address when calling a function or subroutine.
At the end of the function or subroutine, the program control can return to the calling program and resume by loading the value of LR into the Program Counter (PC).

Program Counter (PC) is readable and writeable: a read returns the current instruction address plus 4.
Writing to PC causes a branch operation.



# Cortex-M4 Architecture

- Instruction Set:
- Classic ARM processors were supporting both ARM-32 bit Instructions and Thum16 bit instructions

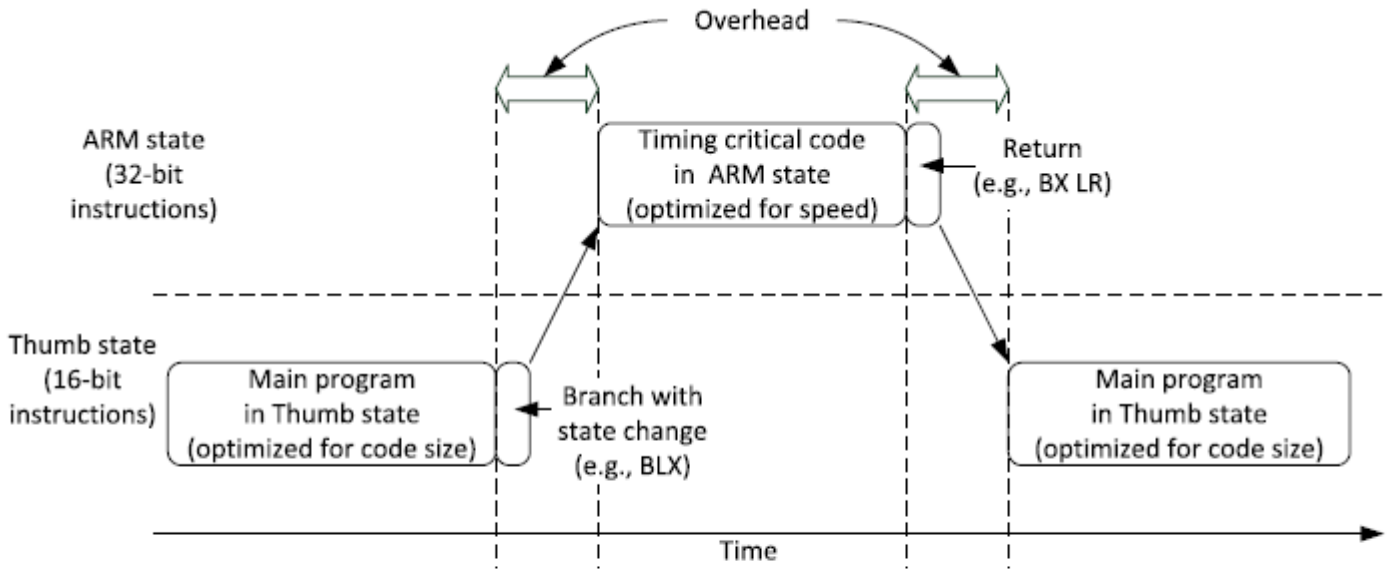



**FIGURE 3.1**

Switching between ARM code and Thumb code in class ARM processors such as the ARM7TDMI

# Cortex-M4 Architecture

- Instruction Set:
- ARM Cortex Instruction set introduced Thumb-2 technology, the Thumb instruction set has been extended to support both 16-bit and 32-bit instruction encoding.

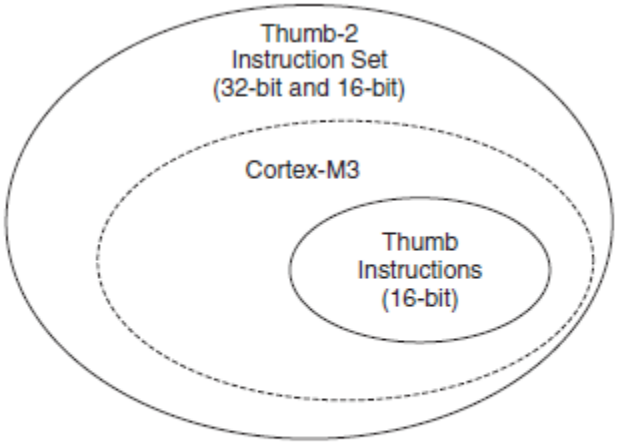



**Figure 1.4 The Relationship Between the Thumb-2 Instruction Set and the Thumb Instruction Set**

# Cortex-M4 Architecture

- Instruction Set:
- With Thumb-2 technology, the Cortex-M processor has a number of advantages over classic ARM processors:

1. No state switching overhead, saving both execution time and instruction space.
2. No need to specify ARM state or Thumb state in source files, making software development easier.
3. It is easier to get the best code density, efficiency, and performance at the same time.
4. With Thumb-2 technology, the Thumb instruction set has been extended by a wide margin when compared to a classic processor

# Cortex-M4 Architecture

- Memory System
- The Cortex-M3 and M4 processors themselves do not include memories.
- they come with a generic on-chip bus interface.
- the microcontroller vendor will need to add the following items to the memory system:

1. Program memory, typically flash
2. Data memory, typically SRAM
3. Peripherals

# Cortex-M4 Architecture

- Memory System
- The provided bus interfaces on the Cortex-M processors are 32-bit, and based on the Advanced Microcontroller Bus Architecture (AMBA) standard.

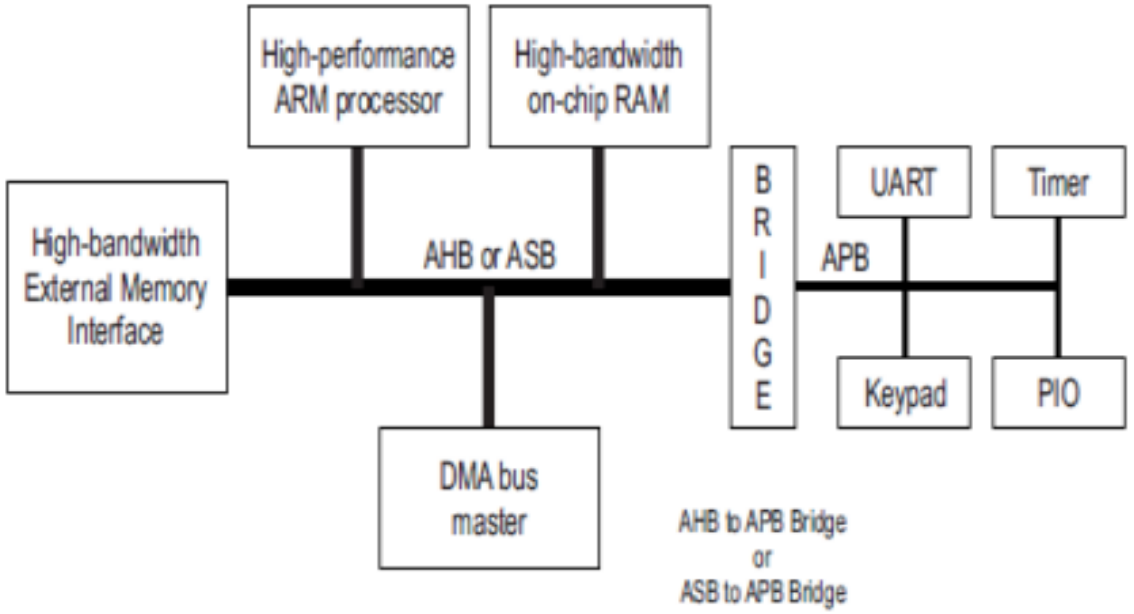
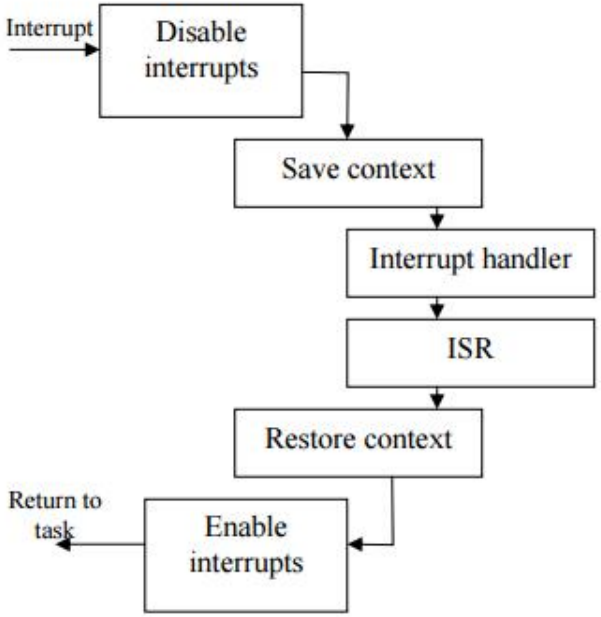



**Fig. 1**. AMBA based Simple Microcontroller

# Cortex-M4 Exceptions and interrupts

- Exceptions are events that cause changes to program flow.
- When one happens, the processor suspends the current executing task and executes a part of the program called the exception handler.
- After the execution of the exception handler is completed, the processor then resumes normal program execution.



# Cortex-M4 Exceptions and interrupts

**Table 4.9** Exception Types

| Exception Number | CMSIS Interrupt Number | Exception Type | Priority | Function |
|---|---|---|---|---|
| 1 | — | Reset | −3 (Highest) | Reset |
| 2 | −14 | NMI | −2 | Non-Maskable interrupt |
| 3 | −13 | HardFault | −1 | All classes of fault, when the corresponding fault handler cannot be activated because it is currently disabled or masked by exception masking |
| 4 | −12 | MemManage | Settable | Memory Management fault; caused by MPU violation or invalid accesses (such as an instruction fetch from a non-executable region) |
| 5 | −11 | BusFault | Settable | Error response received from the bus system; caused by an instruction prefetch abort or data access error |
| 6 | −10 | Usage fault | Settable | Usage fault; typical causes are invalid instructions or invalid state transition attempts (such as trying to switch to ARM state in the Cortex-M3) |
| 7–10 | — | — | — | Reserved |
| 11 | −5 | SVC | Settable | Supervisor Call via SVC instruction |
| 12 | −4 | Debug monitor | Settable | Debug monitor – for software based debug (often not used) |
| 13 | — | — | — | Reserved |
| 14 | −2 | PendSV | Settable | Pendable request for System Service |
| 15 | −1 | SYSTICK | Settable | System Tick Timer |
| 16–255 | 0–239 | IRQ | Settable | IRQ input #0–239 |

# Cortex-M4 Exceptions and interrupts

- Nested Interrupt Vector Controller. (NIVC)
- The NVIC handles the exceptions and interrupt configurations, prioritization, and interrupt masking.
- The NVIC has the following features:

1. Flexible exception and interrupt management
2. Nested exception/interrupt support.
3. Vectored exception/interrupt entry.
4. Interrupt masking.

# Agenda

1. Overview on ARM architecture
2. ARM Cortex-M4 and ARM Cortex-M3 Specifications
3. TM4C123GH6PM Microcontroller Peripherals.
4. TIVA TM4C123GH6PM Launchpad kit specifications
5. GPIO Interface with applications
6. ADC Interface with applications
7. Interrupts and exceptions of TM4C123GH6PM.
8. SPI Interface with applications.
9. I2C Interface with applications.
10. UART Interface.
11. DMA and its applications.
12. Timers and PWM interfacing .

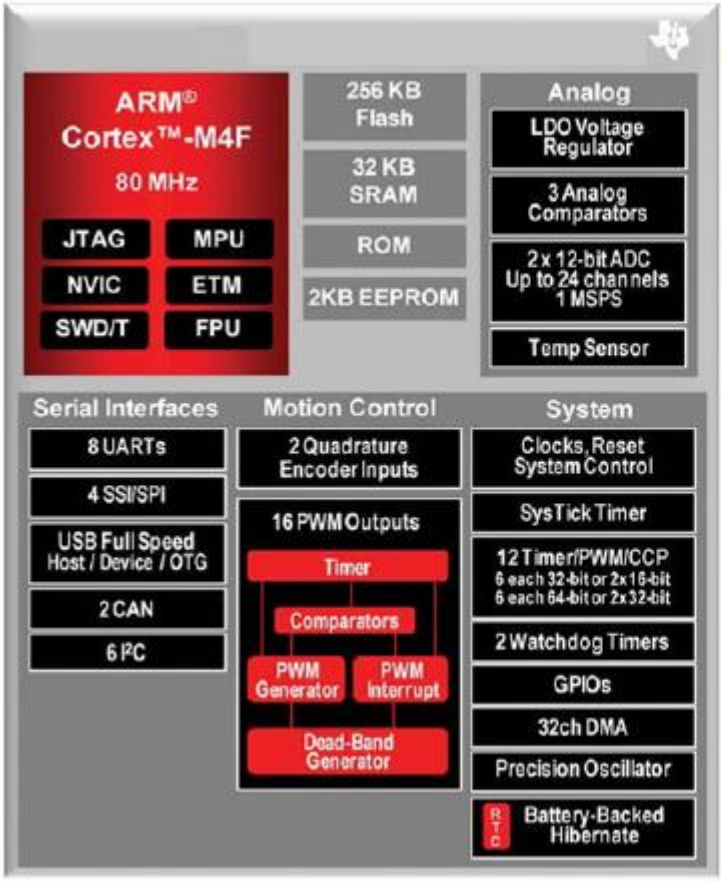
# 3. TM4C123GH6PM Microcontroller Peripherals.

# Outline

- The point consists of the following topics:
  - Microcontroller Features
  - Microcontroller High level block diagram

# TM4C123GH6PM Microcontroller Overview

## Microcontroller Features




**Low power consumption**

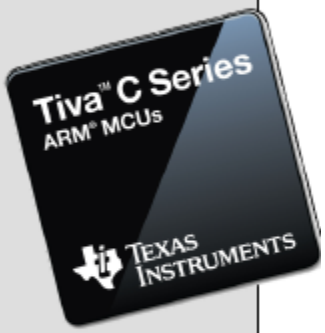
- As low as 370 µA/MHz
- 500µs wakeup from low-power modes
- RTC currents as low as 1.7µA
- Internal and external power control

# TM4C123GH6PM Microcontroller Overview

## Microcontroller Features

### M4 Core and Floating-Point Unit

- 32-bit ARM® Cortex™-M4 core
- Thumb2 16/32-bit code: 26% less memory & 25 % faster than pure 32-bit
- System clock frequency up to 80 MHz
- 100 DMIPS @ 80MHz
- Flexible clocking system
  - Internal precision oscillator
  - External main oscillator with PLL support
  - Internal low frequency oscillator
  - Real-time-clock through Hibernation module
- Saturated math for signal processing
- Atomic bit manipulation. Read-Modify-Write using bit-banding
- Single Cycle multiply and hardware divider
- Unaligned data access for more efficient memory usage
- IEEE754 compliant single-precision floating-point unit
- JTW and Serial Wire Debug debugger access
  - ETM (Embedded Trace Macrocell) available through Keil and IAR emulators



# TM4C123GH6PM Microcontroller Overview

Microcontroller Features

## TM4C123GH6PM Memory

**256KB Flash memory**
- Single-cycle to 40MHz
- Pre-fetch buffer and speculative branch improves performance above 40 MHz

**32KB single-cycle SRAM with bit-banding**

**Internal ROM loaded with TivaWare software**
- Peripheral Driver Library
- Boot Loader
- Advanced Encryption Standard (AES) cryptography tables
- Cyclic Redundancy Check (CRC) error detection functionality

**2KB EEPROM (fast, saves board space)**
- Wear-leveled 500K program/erase cycles
- Thirty-two 16-word blocks
- Can be bulk or block erased
- 10 year data retention
- 4 clock cycle read time


| |
|---|
| 0x00000000 Flash |
| 0x01000000 ROM |
| 0x20000000 SRAM |
| 0x22000000 Bit-banded SRAM |
| 0x40000000 Peripherals & EEPROM |
| 0x42000000 Bit-banded Peripherals |
| 0xE0000000 Instrumentation, ETM, etc. |

# TM4C123GH6PM Microcontroller Overview

Microcontroller Features

## TM4C123GH6PM Peripherals

**Battery-backed Hibernation Module**

- Internal and external power control (through external voltage regulator)
- Separate real-time clock (RTC) and power source
- VDD3ON mode retains GPIO states and settings
- Wake on RTC or Wake pin
- Sixteen 32-bit words of battery backed memory
- 5 µA Hibernate current with GPIO retention. 1.7 µA without

**Serial Connectivity**

- USB 2.0 (OTG/Host/Device)
- 8 - UART with IrDA, 9-bit and ISO7816 support
- 6 - $I^2C$
- 4 - SPI, Microwire or TI synchronous serial interfaces
- 2 - CAN

# TM4C123GH6PM Microcontroller Overview

Microcontroller Features

## TM4C123GH6PM Peripherals

**Two 1MSPS 12-bit SAR ADCs**

- Twelve shared inputs
- Single ended and differential measurement
- Internal temperature sensor
- 4 programmable sample sequencers
- Flexible trigger control: SW, Timers, Analog comparators, GPIO
- VDDA/GNDA voltage reference
- Optional hardware averaging
- 3 analog and 16 digital comparators
- µDMA enabled

**0 - 43 GPIO**

- Any GPIO can be an external edge or level triggered interrupt
- Can initiate an ADC sample sequence or µDMA transfer directly
- Toggle rate up to the CPU clock speed on the Advanced High-Performance Bus
- 5-V-tolerant in input configuration (except for PB0/1 and USB data pins when configured as GPIO)
- Programmable Drive Strength (2, 4, 8 mA or 8 mA with slew rate control)
- Programmable weak pull-up, pull-down, and open drain

# TM4C123GH6PM Microcontroller Overview

Microcontroller Features

## TM4C123GH6PM Peripherals

**Memory Protection Unit (MPU)**

- Generates a Memory Management Fault on incorrect access to region

**Timers**

- 2 Watchdog timers with separate clocks
- SysTick timer. 24-bit high speed RTOS and other timer
- Six 32-bit and Six 64-bit general purpose timers
- PWM and CCP modes
- Daisy chaining
- User enabled stalling on CPU Halt flag from debugger for all timers

**32 channel µDMA**

- Basic, Ping-pong and scatter-gather modes
- Two priority levels
- 8,16 and 32-bit data sizes
- Interrupt enabled

# TM4C123GH6PM Microcontroller Overview

Microcontroller Features

## TM4C123GH6PM Peripherals

**Nested-Vectored Interrupt Controller (NVIC)**

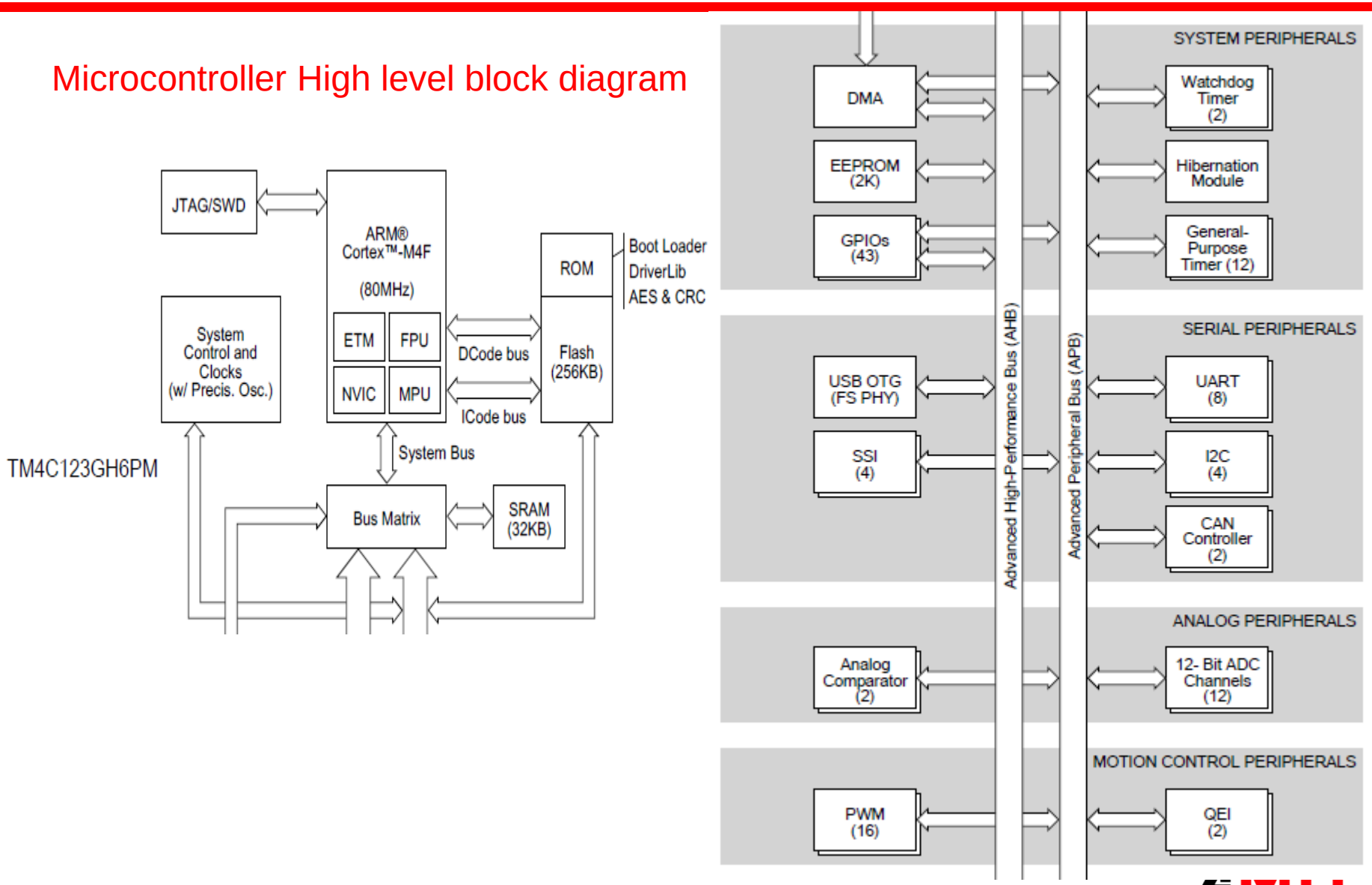
- 7 exceptions and 71 interrupts with 8 programmable priority levels
- Tail-chaining and other low-latency features
- Deterministic: always 12 cycles or 6 with tail-chaining
- Automatic system save and restore

**Two Motion Control modules. Each with:**

- 8 high-resolution PWM outputs (4 pairs)
- H-bridge dead-band generators and hardware polarity control
- Fault input for low-latency shutdown
- Quadrature Encoder Inputs (QEI)
- Synchronization in and between the modules

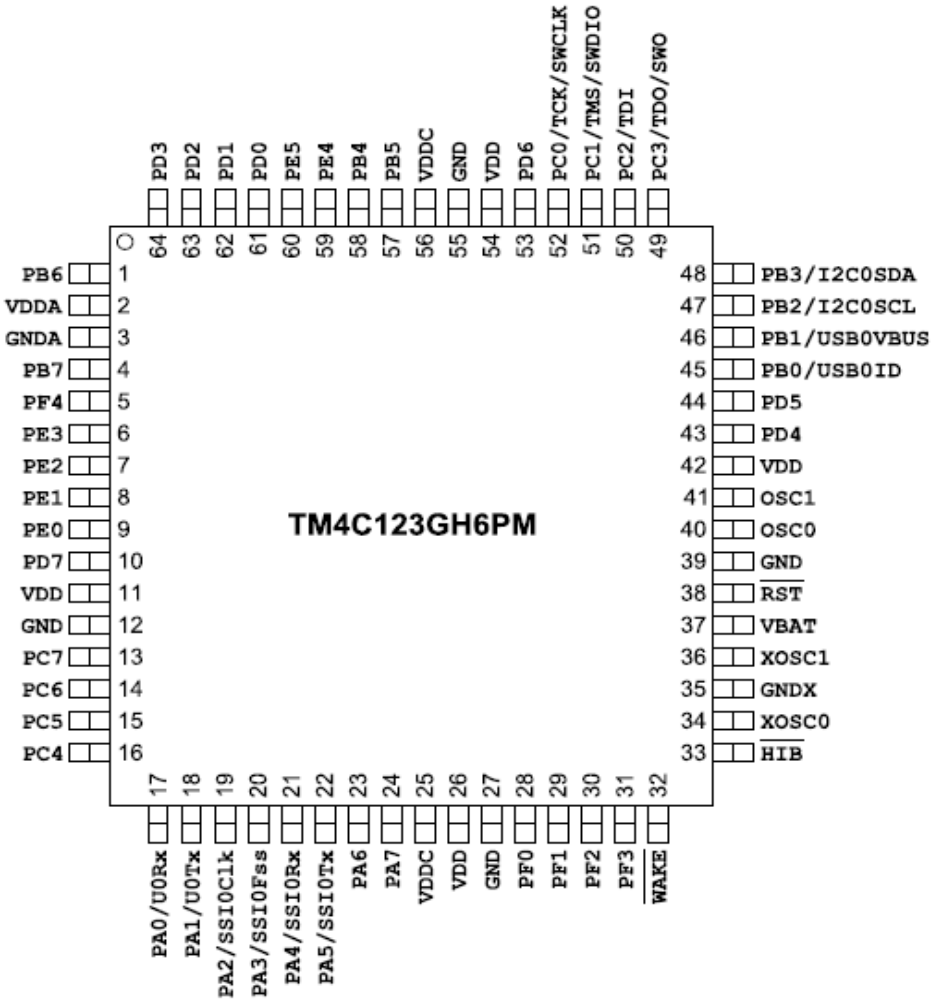
# TM4C123GH6PM Microcontroller Overview

Microcontroller High level block diagram



# TM4C123GH6PM Microcontroller Overview

## Microcontroller Pin Diagram



# Agenda

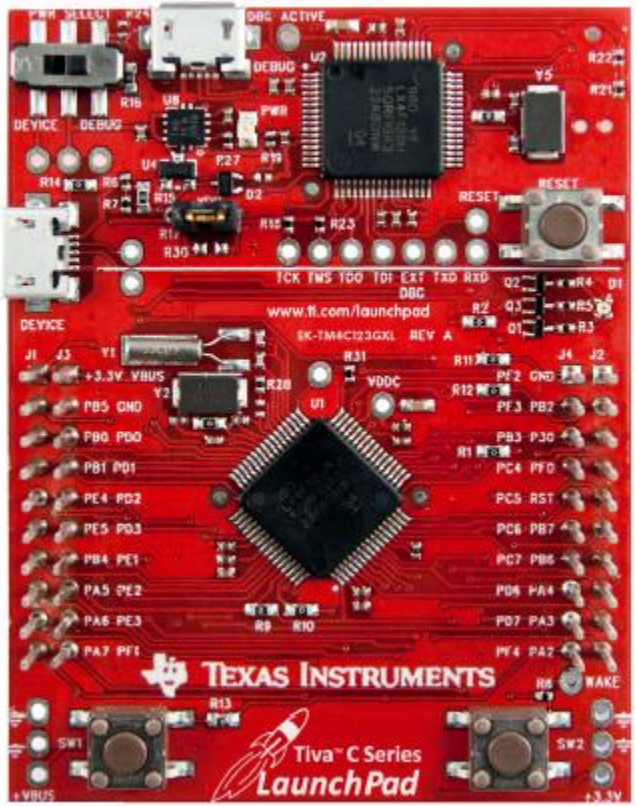
1. Overview on ARM architecture
2. ARM Cortex-M4 and ARM Cortex-M3 Specifications
3. TM4C123GH6PM Microcontroller Peripherals.
4. TIVA TM4C123GH6PM Launchpad kit specifications
5. GPIO Interface with applications
6. ADC Interface with applications
7. Interrupts and exceptions of TM4C123GH6PM.
8. SPI Interface with applications.
9. I2C Interface with applications.
10. UART Interface.
11. DMA and its applications.
12. Timers and PWM interfacing .

# 4. TIVA TM4C123GH6PM Launchpad kit specifications

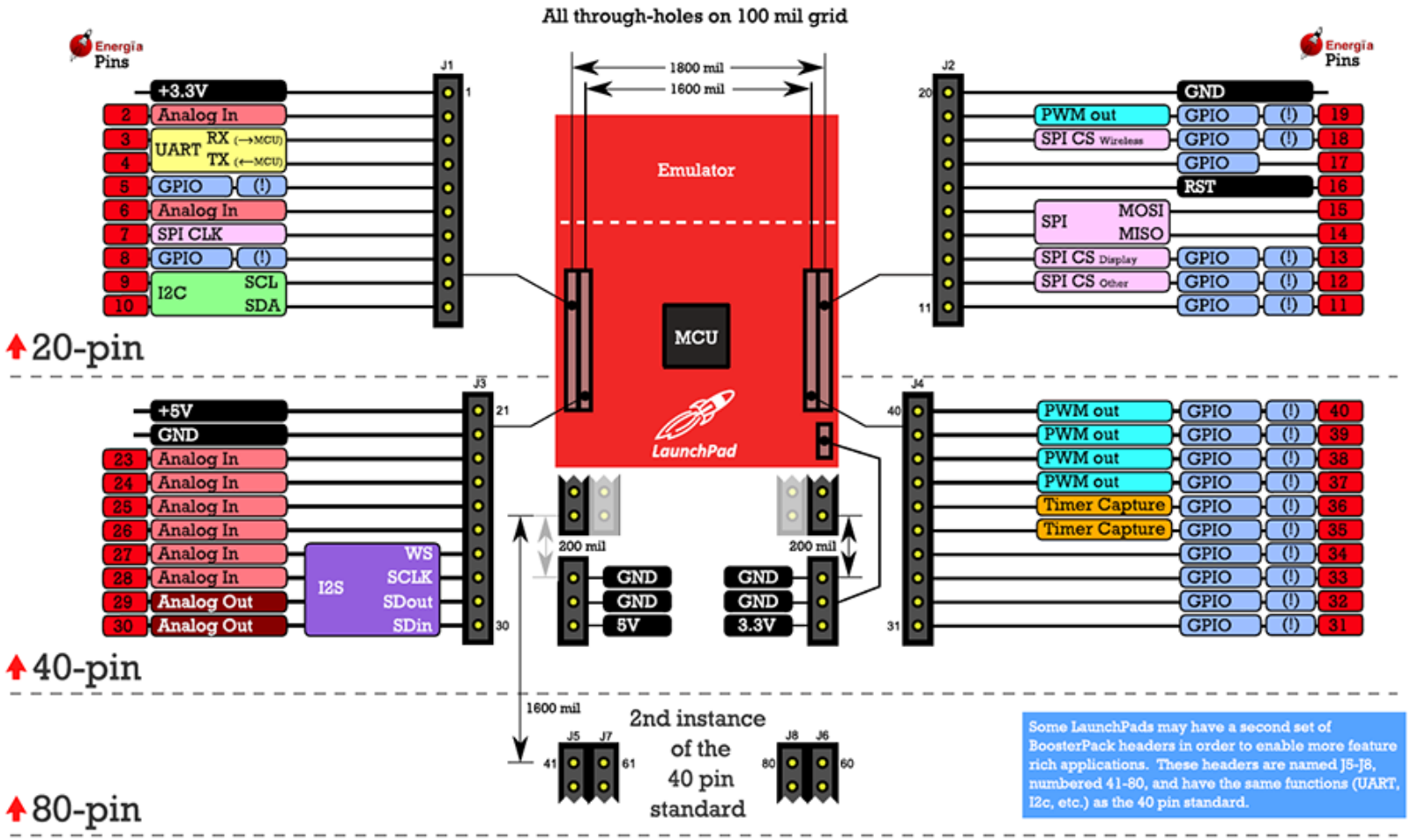
# TIVA TM4C123GH6PM Launchpad kit specifications

## Tiva™ EK-TM4C123GXL LaunchPad

- ARM® Cortex™-M4F 64-pin 80MHz TM4C123GH6PM
- On-board USB ICDI (In-Circuit Debug Interface)
- Micro AB USB port
- Device/ICDI power switch
- BoosterPack XL pinout also supports legacy BoosterPack pinout
- 2 user pushbuttons (SW2 is connected to the WAKE pin)
- Reset button
- 3 user LEDs (1 tri-color device)
- Current measurement test points
- 16MHz Main Oscillator crystal
- 32kHz Real Time Clock crystal
- 3.3V regulator
- Support for multiple IDEs:



# TIVA TM4C123GH6PM Launchpad kit specifications



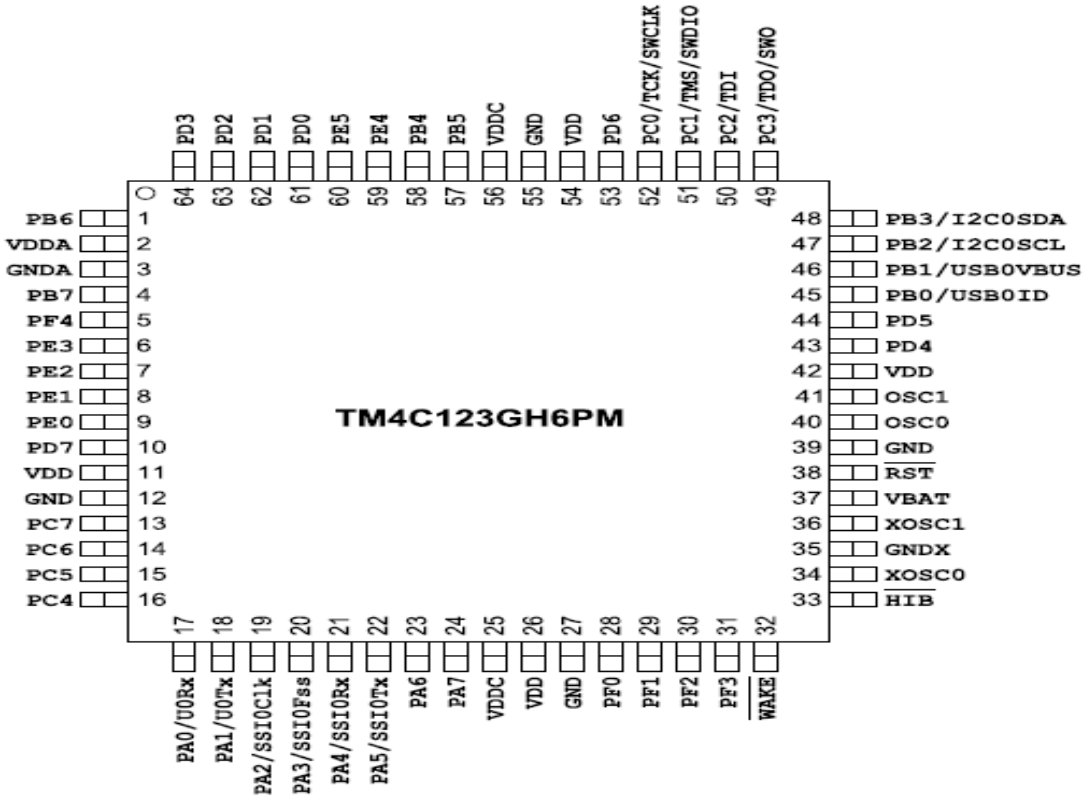
# Agenda

1. Overview on ARM architecture
2. ARM Cortex-M4 and ARM Cortex-M3 Specifications
3. TM4C123GH6PM Microcontroller Peripherals.
4. TIVA TM4C123GH6PM Launchpad kit specifications
5. GPIO Interface with applications
6. ADC Interface with applications
7. Interrupts and exceptions of TM4C123GH6PM.
8. SPI Interface with applications.
9. I2C Interface with applications.
10. UART Interface.
11. DMA and its applications.
12. Timers and PWM interfacing .

# 5. GPIO Interface with applications

# GPIO Interface with applications

The GPIO module is composed of six physical GPIO blocks, each corresponding to an individual GPIO port (Port A, Port B, Port C, Port D, Port E, Port F).



# GPIO Interface with applications

- Up to 43 GPIOs, depending on configuration.
- Highly flexible pin muxing allows use as GPIO or one of several peripheral functions.
- 5-V-tolerant in input configuration.
- Ports A-G accessed through the Advanced Peripheral Bus (APB)

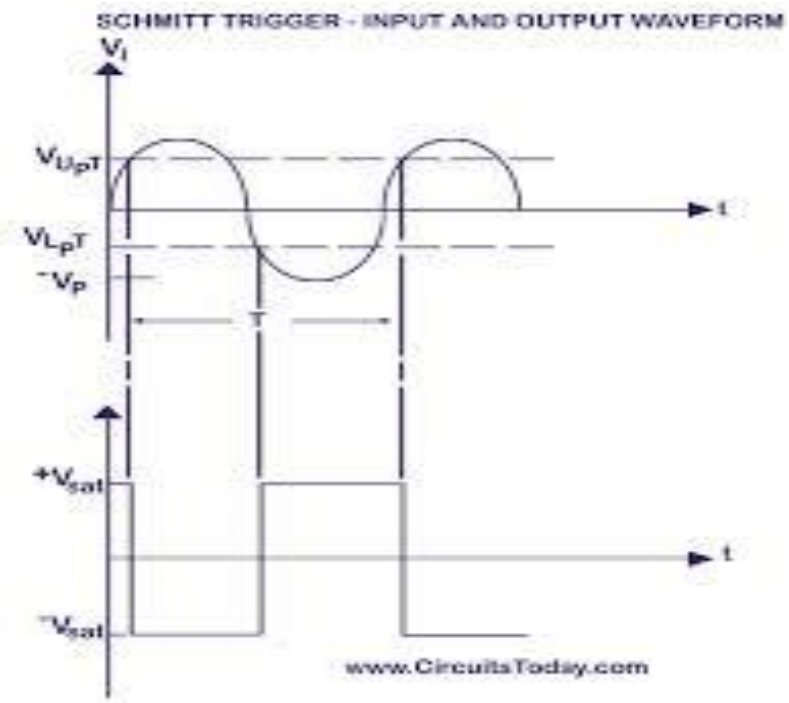
# GPIO Interface with applications

**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

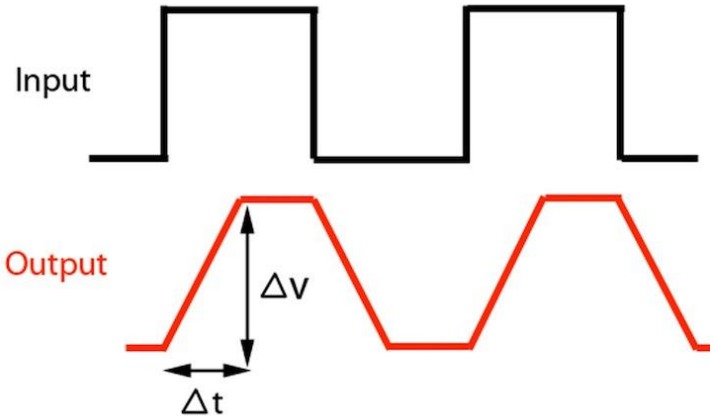
# GPIO Interface with applications

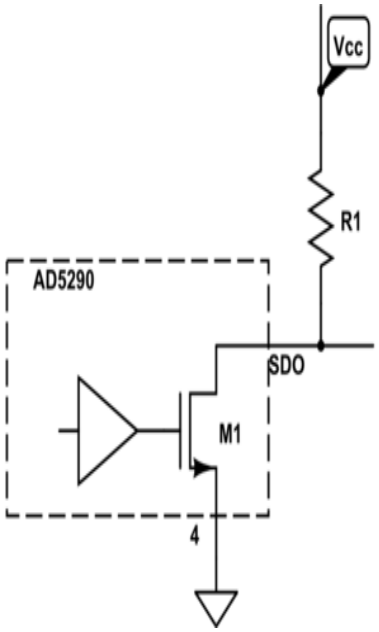
- Can be used to initiate an ADC sample sequence or a µDMA transfer
- Pin state can be retained during Hibernation mode
- Pins configured as digital inputs are Schmitt-triggered



# GPIO Interface with applications

- **Programmable control for GPIO pad configuration**
- Weak pull-up or pull-down resistors
- 2-mA, 4-mA, and 8-mA pad drive for digital communication; up to four pads can sink 18-mA for high-current applications.
- Slew rate control for 8-mA pad drive.
- Open drain enables.
- Digital input enables.



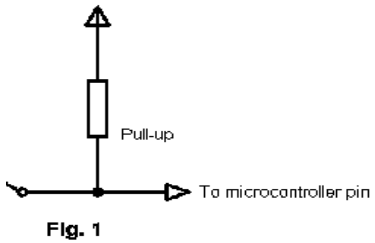
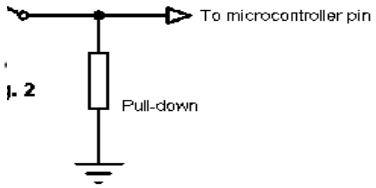




Fig. 1



# GPIO Interface with applications

## GPIO IO Registers related to DIO

### 1- Activate the PORT Clock signal

General-Purpose Input/Output Run Mode Clock Gating Control (RCGCGPIO)

Base 0x400F.E000
Offset 0x608
Type RW, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | R5 | R4 | R3 | R2 | R1 | R0 |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# GPIO Interface with applications

## GPIO IO Registers related to DIO

### 1- Activate the PORT Clock signal

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:6 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 5 | R5 | RW | 0 | GPIO Port F Run Mode Clock Gating Control<br>Value Description<br>0 GPIO Port F is disabled.<br>1 Enable and provide a clock to GPIO Port F in Run mode. |
| 4 | R4 | RW | 0 | GPIO Port E Run Mode Clock Gating Control<br>Value Description<br>0 GPIO Port E is disabled.<br>1 Enable and provide a clock to GPIO Port E in Run mode. |

# GPIO Interface with applications

## GPIO IO Registers related to DIO

### 1- Activate the PORT Clock signal

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3 | R3 | RW | 0 | GPIO Port D Run Mode Clock Gating Control<br>Value Description<br>0 GPIO Port D is disabled.<br>1 Enable and provide a clock to GPIO Port D in Run mode. |
| 2 | R2 | RW | 0 | GPIO Port C Run Mode Clock Gating Control<br>Value Description<br>0 GPIO Port C is disabled.<br>1 Enable and provide a clock to GPIO Port C in Run mode. |
| 1 | R1 | RW | 0 | GPIO Port B Run Mode Clock Gating Control<br>Value Description<br>0 GPIO Port B is disabled.<br>1 Enable and provide a clock to GPIO Port B in Run mode. |
| 0 | R0 | RW | 0 | GPIO Port A Run Mode Clock Gating Control<br>Value Description<br>0 GPIO Port A is disabled.<br>1 Enable and provide a clock to GPIO Port A in Run mode. |

# GPIO Interface with applications

## GPIO IO Registers related to DIO

### 2-Digital function enable

GPIO Digital Enable (GPIODEN)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
GPIO Port F (APB) base: 0x4002.5000
GPIO Port F (AHB) base: 0x4005.D000
Offset 0x51C
Type RW, reset -

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | DEN | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | - | - | - | - | - | - | - | - |

# GPIO Interface with applications

## GPIO IO Registers related to DIO

### 3- PIN Direction configuration

GPIO Direction (GPIODIR)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
GPIO Port F (APB) base: 0x4002.5000
GPIO Port F (AHB) base: 0x4005.D000
Offset 0x400
Type RW, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | DIR | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# GPIO Interface with applications

## GPIO IO Registers related to DIO

**4- PAD Drive current configuration. (if a drive selection was set the others are cleared)**

GPIO 2-mA Drive Select (GPIODR2R)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
GPIO Port F (APB) base: 0x4002.5000
GPIO Port F (AHB) base: 0x4005.D000
Offset 0x500
Type RW, reset 0x0000.00FF

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | DRV2 | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

# GPIO Interface with applications

## GPIO IO Registers related to DIO

**4- PAD Drive current configuration. (if a drive selection was set the others are cleared)**

GPIO 4-mA Drive Select (GPIODR4R)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
GPIO Port F (APB) base: 0x4002.5000
GPIO Port F (AHB) base: 0x4005.D000
Offset 0x504
Type RW, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | DRV4 | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# GPIO Interface with applications

## GPIO IO Registers related to DIO

**4- PAD Drive current configuration. (if a drive selection was set the others are cleared)**

GPIO 8-mA Drive Select (GPIODR8R)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
GPIO Port F (APB) base: 0x4002.5000
GPIO Port F (AHB) base: 0x4005.D000
Offset 0x508
Type RW, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | DRV8 | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# GPIO Interface with applications

## GPIO IO Registers related to DIO

**5- Internal Resistor selection (if an internal resistor type is selected the other is disabled by default)**

GPIO Pull-Up Select (GPIOPUR)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
GPIO Port F (APB) base: 0x4002.5000
GPIO Port F (AHB) base: 0x4005.D000
Offset 0x510
Type RW, reset -

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | PUE | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | - | - | - | - | - | - | - | - |

# GPIO Interface with applications

## GPIO IO Registers related to DIO

**5- Internal Resistor selection (if an internal resistor type is selected the other is disabled by default)**

GPIO Pull-Down Select (GPIOPDR)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
GPIO Port F (APB) base: 0x4002.5000
GPIO Port F (AHB) base: 0x4005.D000
Offset 0x514
Type RW, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | PDE | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# GPIO Interface with applications

## GPIO IO Registers related to DIO

**6- DIO writing and reading data (This Register is written using bit banding) (Bits address = Port base + Register offset + Bits mask << 2)**

GPIO Data (GPIODATA)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
GPIO Port F (APB) base: 0x4002.5000
GPIO Port F (AHB) base: 0x4005.D000
Offset 0x000
Type RW, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | DATA | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x0000.00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | DATA | RW | 0x00 | GPIO Data<br>This register is virtually mapped to 256 locations in the address space. To facilitate the reading and writing of data to these registers by independent drivers, the data read from and written to the registers are masked by the eight address lines [9:2]. Reads from this register return its current state. Writes to this register only affect bits that are not masked by ADDR[9:2] and are configured as outputs. See “Data Register Operation” on page 654 for examples of reads and writes. |

# Agenda

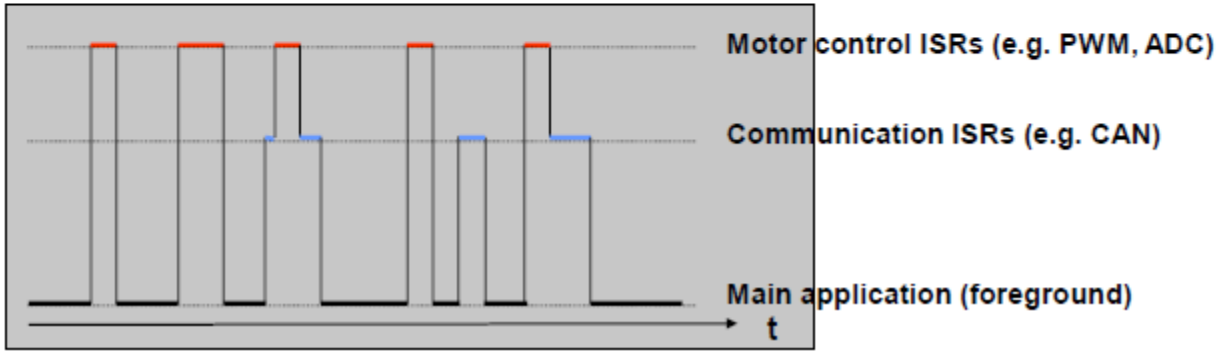
1. Overview on ARM architecture
2. ARM Cortex-M4 and ARM Cortex-M3 Specifications
3. TM4C123GH6PM Microcontroller Peripherals.
4. TIVA TM4C123GH6PM Launchpad kit specifications
5. GPIO Interface with applications
6. ADC Interface with applications
7. Interrupts and exceptions of TM4C123GH6PM.
8. SPI Interface with applications.
9. I2C Interface with applications.
10. UART Interface.
11. DMA and its applications.
12. Timers and PWM interfacing .

# 5. Interrupts and exceptions of TM4C123GH6PM

# Interrupts and exceptions of TM4C123GH6PM

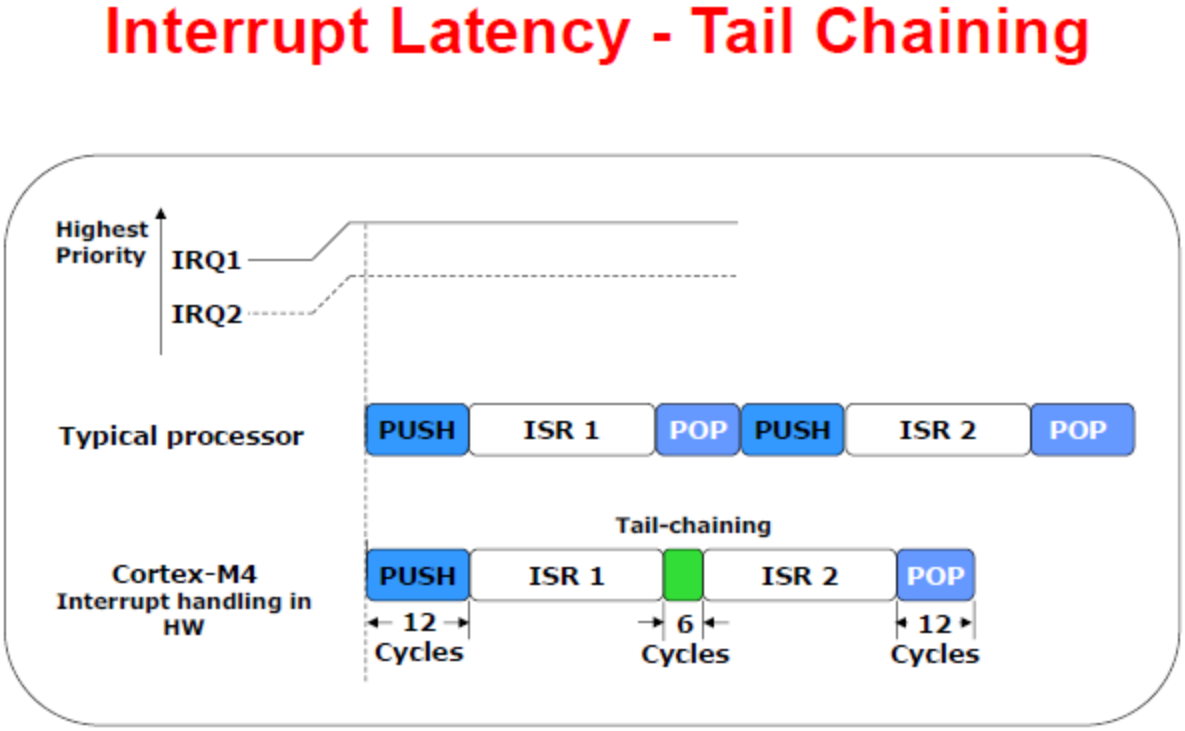
## NIVC Capabilities

- Handles exceptions and interrupts
- 8 programmable priority levels, priority grouping
- 7 exceptions and 71 Interrupts
- Automatic state saving and restoring
- Automatic reading of the vector table entry
- Pre-emptive/Nested Interrupts
- Tail-chaining
- Deterministic: always 12 cycles or 6 with tail-chaining



# Interrupts and exceptions of TM4C123GH6PM

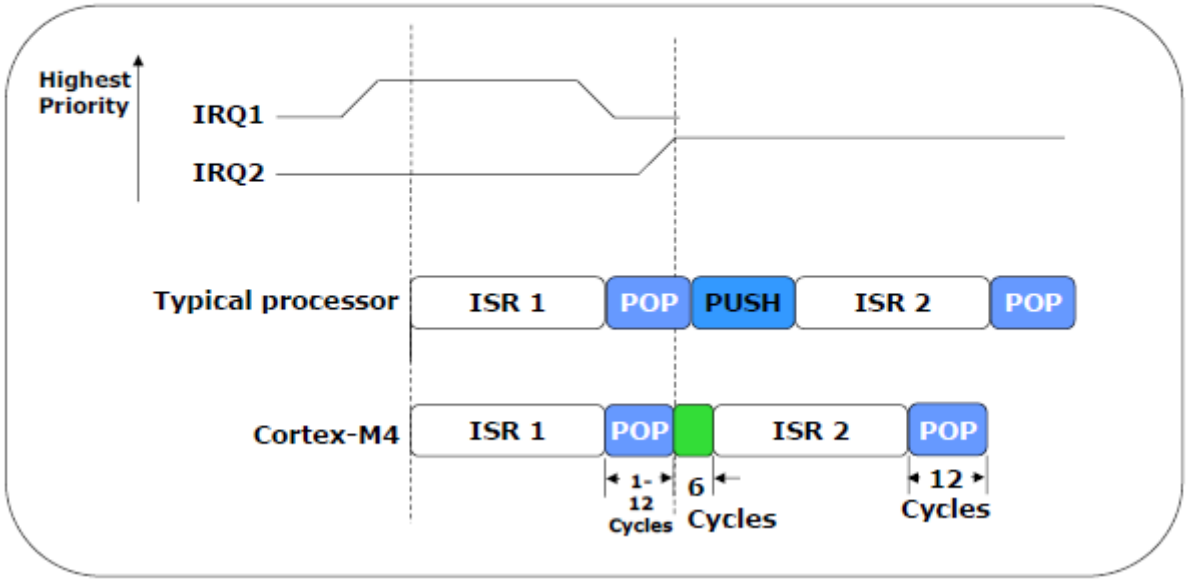
**Effect of Tail chaining on interrupt latency**




- In most processors, interrupt handling is fairly simple and each interrupt will start a PUSH PROCESSOR STATE – RUN ISR – POP PROCESSOR STATE process.
- If the interrupt handler could have seen that a second interrupt was pending, it could have “tail-chained” into the next ISR, saving power and cycles.

# Interrupts and exceptions of TM4C123GH6PM

## Effect of Tail chaining on interrupt latency

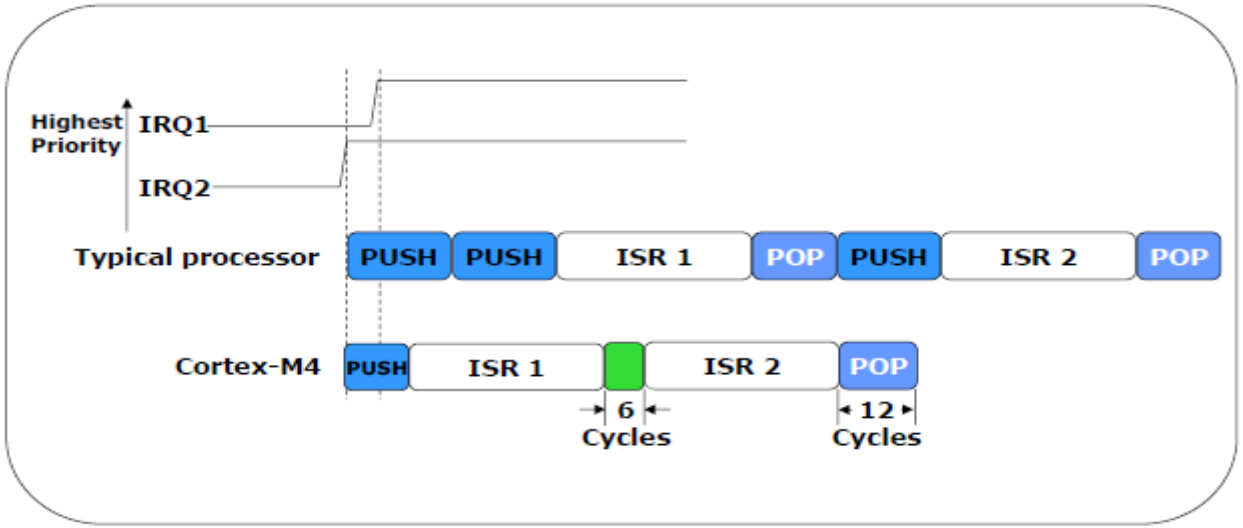



- In most processors, the interrupt controller would complete the process before starting the entire PUSH-ISR-POP process over again, wasting precious cycles and power doing so.
- The Tiva C NVIC is able to stop the POP process, return the stack pointer to the proper location and “tail-chain” into the next ISR with only 6 cycles.

# Interrupts and exceptions of TM4C123GH6PM

## Effect of Tail chaining on interrupt latency

### Interrupt Latency – Late Arrival




- In most processors, the interrupt controller is smart enough to recognize the late arrival of a higher priority interrupt and restart the interrupt procedure accordingly.
- In Tiva C NIVC The PUSH is the same process regardless of the ISR, so the Tiva C NVIC simply changes the fetched ISR. In between the ISRs, “tail chaining” is done to save cycles.

# Interrupts and exceptions of TM4C123GH6PM

## Sources of exceptions in ARM-Cortex M4F

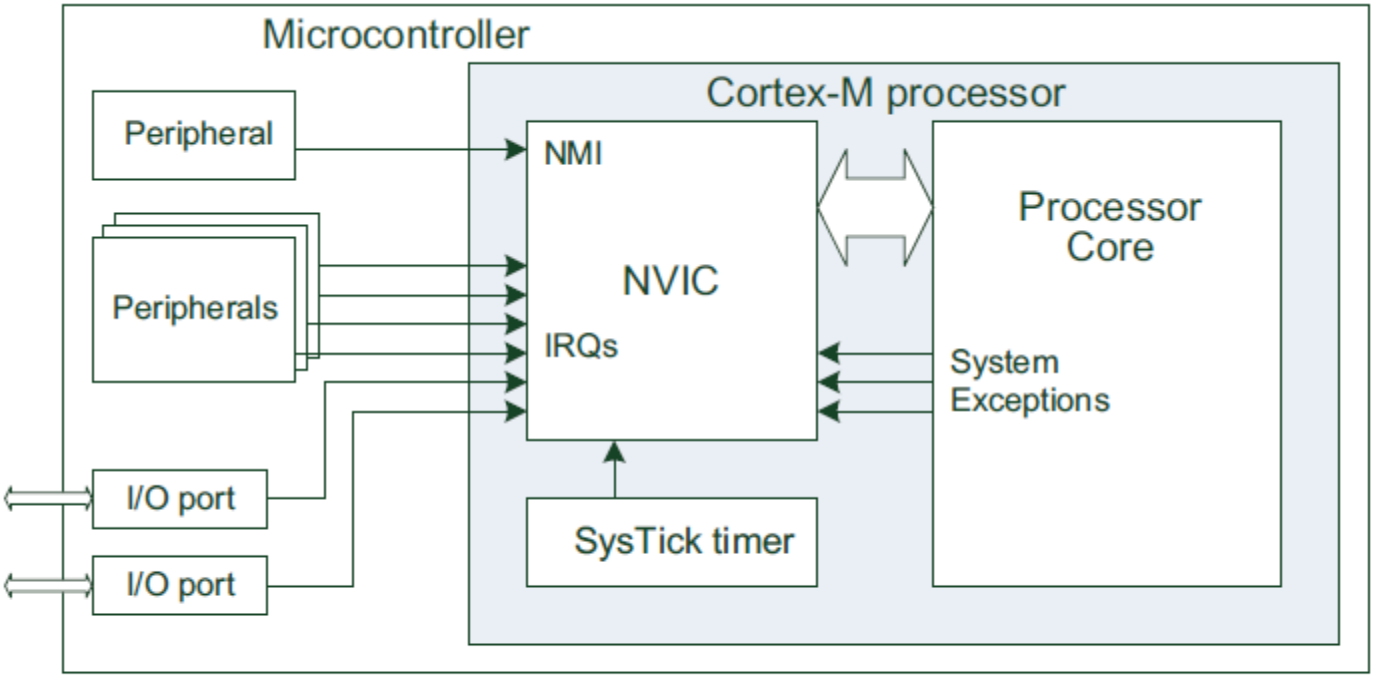



**FIGURE 7.1**

Various sources of exceptions in a typical microcontroller

# Interrupts and exceptions of TM4C123GH6PM

## Sources of exceptions in ARM-Cortex M4F

**Table 2-8. Exception Types**

| Exception Type | Vector Number | Priority[a] | Vector Address or Offset[b] | Activation |
|---|---|---|---|---|
| - | 0 | - | 0x0000.0000 | Stack top is loaded from the first entry of the vector table on reset. |
| Reset | 1 | -3 (highest) | 0x0000.0004 | Asynchronous |
| Non-Maskable Interrupt (NMI) | 2 | -2 | 0x0000.0008 | Asynchronous |
| Hard Fault | 3 | -1 | 0x0000.000C | - |
| Memory Management | 4 | programmable[c] | 0x0000.0010 | Synchronous |
| Bus Fault | 5 | programmable[c] | 0x0000.0014 | Synchronous when precise and asynchronous when imprecise |
| Usage Fault | 6 | programmable[c] | 0x0000.0018 | Synchronous |
| - | 7-10 | - | - | Reserved |
| SVCall | 11 | programmable[c] | 0x0000.002C | Synchronous |
| Debug Monitor | 12 | programmable[c] | 0x0000.0030 | Synchronous |
| - | 13 | - | - | Reserved |
| PendSV | 14 | programmable[c] | 0x0000.0038 | Asynchronous |
| SysTick | 15 | programmable[c] | 0x0000.003C | Asynchronous |
| Interrupts | 16 and above | programmable[d] | 0x0000.0040 and above | Asynchronous |

# Interrupts and exceptions of TM4C123GH6PM

**Exception states in ARM-Cortex M4F**

- **Inactive.**

The exception is not active and not pending.

- **Pending**.

The exception is waiting to be serviced by the processor. An interrupt request from a peripheral or from software can change the state of the corresponding interrupt to pending.

- **Active**.

An exception that is being serviced by the processor but has not completed.

Note:

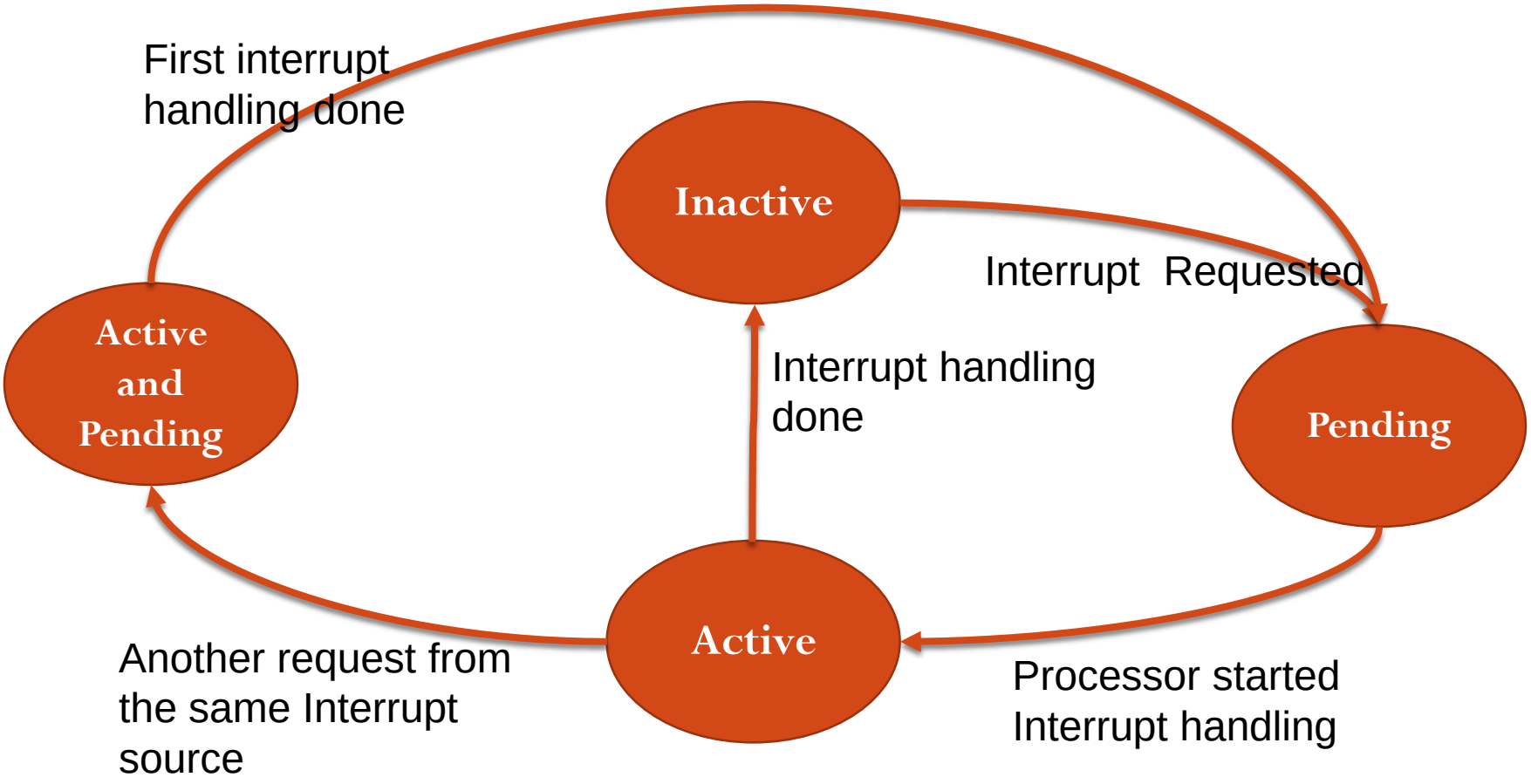
- **Active and Pending.**

The exception is being serviced by the processor, and there is a pending exception from the same source.

**if an exception is preempted by another, what is the preempted exception state?**

# Interrupts and exceptions of TM4C123GH6PM

## Exception states in ARM-Cortex M4F



# Interrupts and exceptions of TM4C123GH6PM

## Sources of interrupts in TM4C123GH6PM (see table 2-9 page 104)

| Vector Number | Interrupt Number (Bit in Interrupt Registers) | Vector Address or Offset | Description |
|---|---|---|---|
| 0-15 | - | 0x0000.0000 - 0x0000.003C | Processor exceptions |
| 16 | 0 | 0x0000.0040 | GPIO Port A |
| 17 | 1 | 0x0000.0044 | GPIO Port B |
| 18 | 2 | 0x0000.0048 | GPIO Port C |
| 19 | 3 | 0x0000.004C | GPIO Port D |
| 20 | 4 | 0x0000.0050 | GPIO Port E |
| 21 | 5 | 0x0000.0054 | UART0 |
| 22 | 6 | 0x0000.0058 | UART1 |
| 23 | 7 | 0x0000.005C | SSI0 |
| 24 | 8 | 0x0000.0060 | $I^2C0$ |
| 25 | 9 | 0x0000.0064 | PWM0 Fault |
| 26 | 10 | 0x0000.0068 | PWM0 Generator 0 |
| 27 | 11 | 0x0000.006C | PWM0 Generator 1 |
| 28 | 12 | 0x0000.0070 | PWM0 Generator 2 |
| 29 | 13 | 0x0000.0074 | QEI0 |
| 30 | 14 | 0x0000.0078 | ADC0 Sequence 0 |
| 31 | 15 | 0x0000.007C | ADC0 Sequence 1 |
| 32 | 16 | 0x0000.0080 | ADC0 Sequence 2 |
| 33 | 17 | 0x0000.0084 | ADC0 Sequence 3 |
| 34 | 18 | 0x0000.0088 | Watchdog Timers 0 and 1 |

# Interrupts and exceptions of TM4C123GH6PM

## Sources of interrupts in TM4C123GH6PM

| | | | |
|---|---|---|---|
| 35 | 19 | 0x0000.008C | 16/32-Bit Timer 0A |
| 36 | 20 | 0x0000.0090 | 16/32-Bit Timer 0B |
| 37 | 21 | 0x0000.0094 | 16/32-Bit Timer 1A |
| 38 | 22 | 0x0000.0098 | 16/32-Bit Timer 1B |
| 39 | 23 | 0x0000.009C | 16/32-Bit Timer 2A |
| 40 | 24 | 0x0000.00A0 | 16/32-Bit Timer 2B |
| 41 | 25 | 0x0000.00A4 | Analog Comparator 0 |
| 42 | 26 | 0x0000.00A8 | Analog Comparator 1 |
| 43 | 27 | - | Reserved |
| 44 | 28 | 0x0000.00B0 | System Control |
| 45 | 29 | 0x0000.00B4 | Flash Memory Control and EEPROM Control |
| 46 | 30 | 0x0000.00B8 | GPIO Port F |
| 47-48 | 31-32 | - | Reserved |
| 49 | 33 | 0x0000.00C4 | UART2 |
| 50 | 34 | 0x0000.00C8 | SSI1 |
| 51 | 35 | 0x0000.00CC | 16/32-Bit Timer 3A |
| 52 | 36 | 0x0000.00D0 | 16/32-Bit Timer 3B |
| 53 | 37 | 0x0000.00D4 | $I^2C1$ |
| 54 | 38 | 0x0000.00D8 | QEI1 |
| 55 | 39 | 0x0000.00DC | CAN0 |
| 56 | 40 | 0x0000.00E0 | CAN1 |
| 57-58 | 41-42 | - | Reserved |

# Interrupts and exceptions of TM4C123GH6PM

## Sources of interrupts in TM4C123GH6PM

| | | | |
|---|---|---|---|
| 59 | 43 | 0x0000.00EC | Hibernation Module |
| 60 | 44 | 0x0000.00F0 | USB |
| 61 | 45 | 0x0000.00F4 | PWM Generator 3 |
| 62 | 46 | 0x0000.00F8 | µDMA Software |
| 63 | 47 | 0x0000.00FC | µDMA Error |
| 64 | 48 | 0x0000.0100 | ADC1 Sequence 0 |
| 65 | 49 | 0x0000.0104 | ADC1 Sequence 1 |
| 66 | 50 | 0x0000.0108 | ADC1 Sequence 2 |
| 67 | 51 | 0x0000.010C | ADC1 Sequence 3 |
| 68-72 | 52-56 | - | Reserved |
| 73 | 57 | 0x0000.0124 | SSI2 |
| 74 | 58 | 0x0000.0128 | SSI3 |
| 75 | 59 | 0x0000.012C | UART3 |
| 76 | 60 | 0x0000.0130 | UART4 |
| 77 | 61 | 0x0000.0134 | UART5 |
| 78 | 62 | 0x0000.0138 | UART6 |
| 79 | 63 | 0x0000.013C | UART7 |
| 80-83 | 64-67 | 0x0000.0140 - 0x0000.014C | Reserved |
| 84 | 68 | 0x0000.0150 | $I^2C2$ |
| 85 | 69 | 0x0000.0154 | $I^2C3$ |
| 86 | 70 | 0x0000.0158 | 16/32-Bit Timer 4A |
| 87 | 71 | 0x0000.015C | 16/32-Bit Timer 4B |
| 88-107 | 72-91 | 0x0000.0160 - 0x0000.01AC | Reserved |

# Interrupts and exceptions of TM4C123GH6PM

## Sources of interrupts in TM4C123GH6PM

| | | | |
|---|---|---|---|
| 108 | 92 | 0x0000.01B0 | 16/32-Bit Timer 5A |
| 109 | 93 | 0x0000.01B4 | 16/32-Bit Timer 5B |
| 110 | 94 | 0x0000.01B8 | 32/64-Bit Timer 0A |
| 111 | 95 | 0x0000.01BC | 32/64-Bit Timer 0B |
| 112 | 96 | 0x0000.01C0 | 32/64-Bit Timer 1A |
| 113 | 97 | 0x0000.01C4 | 32/64-Bit Timer 1B |
| 114 | 98 | 0x0000.01C8 | 32/64-Bit Timer 2A |
| 115 | 99 | 0x0000.01CC | 32/64-Bit Timer 2B |
| 116 | 100 | 0x0000.01D0 | 32/64-Bit Timer 3A |
| 117 | 101 | 0x0000.01D4 | 32/64-Bit Timer 3B |
| 118 | 102 | 0x0000.01D8 | 32/64-Bit Timer 4A |
| 119 | 103 | 0x0000.01DC | 32/64-Bit Timer 4B |
| 120 | 104 | 0x0000.01E0 | 32/64-Bit Timer 5A |
| 121 | 105 | 0x0000.01E4 | 32/64-Bit Timer 5B |
| 122 | 106 | 0x0000.01E8 | System Exception (imprecise) |
| 123-149 | 107-133 | - | Reserved |
| 150 | 134 | 0x0000.0258 | PWM1 Generator 0 |
| 151 | 135 | 0x0000.025C | PWM1 Generator 1 |
| 152 | 136 | 0x0000.0260 | PWM1 Generator 2 |
| 153 | 137 | 0x0000.0264 | PWM1 Generator 3 |
| 154 | 138 | 0x0000.0268 | PWM1 Fault |

# Interrupts and exceptions of TM4C123GH6PM

**Exception Priorities**

- There are some sources with fixed priority and others with configurable priority.
- As the priority number decreases, the priority of the exception increased.
- If two interrupts are pending , the interrupt with the higher priority will be handled.
- If both of pending interrupts have the same priority, the interrupt with the lower interrupt number will be handled.
- If a higher priority interrupt becomes pending while a lower priority interrupt is active, the lower one will be preempted.
- If the newly pending interrupt is the same priority of the currently active interrupt, the active interrupt will not be preempted regardless of both interrupts numbers.

# Interrupts and exceptions of TM4C123GH6PM

**Interrupt priority grouping**

- To increase priority control in systems with interrupts, the NVIC supports priority grouping.
- This grouping divides each interrupt priority register entry into two fields:
  - An upper field that defines the group priority
  - A lower field that defines a subpriority within the group

- Only the group priority determines preemption of interrupt exceptions.

- If the two pending interrupts have the same group priority, the subpriority order will determine the order of execution.
- If the two pending interrupts have the same group priority and the same subpriority, the interrupt number will determine the order of execution.

# Interrupts and exceptions of TM4C123GH6PM

**Configuring interrupt registers**
**1- Enable Maskable interrupts**

- **Register 4: Interrupt 0-31 Set Enable (EN0), offset 0x100**
- **Register 5: Interrupt 32-63 Set Enable (EN1), offset 0x104**
- **Register 6: Interrupt 64-95 Set Enable (EN2), offset 0x108**
- **Register 7: Interrupt 96-127 Set Enable (EN3), offset 0x10C**
- **Register 8: Interrupt 128-138 Set Enable (EN4), offset 0x110**

- Bit 0 of EN0 corresponds to Interrupt 0; bit 31 corresponds to Interrupt 31.
- Bit 0 of EN1 corresponds to Interrupt 32; bit 31 corresponds to Interrupt 63.
- Bit 0 of EN2 corresponds to Interrupt 64; bit 31 corresponds to Interrupt 95.
- Bit 0 of EN3 corresponds to Interrupt 96; bit 31 corresponds to Interrupt 127.
- Bit 0 of EN4 corresponds to Interrupt 128; bit 10 corresponds to Interrupt 138.

# Interrupts and exceptions of TM4C123GH6PM

## Configuring interrupt registers
## 1- Enable Maskable Interrupts

Interrupt 0-31 Set Enable (EN0)

Base 0xE000.E000
Offset 0x100
Type RW, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | INT | | | | | | | | | | | | | | | |
| Type | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | INT | | | | | | | | | | | | | | | |
| Type | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | INT | RW | 0x0000.0000 | Interrupt Enable |

| Value | Description |
|---|---|
| 0 | On a read, indicates the interrupt is disabled.<br>On a write, no effect. |
| 1 | On a read, indicates the interrupt is enabled.<br>On a write, enables the interrupt. |

A bit can only be cleared by setting the corresponding `INT[n]` bit in the **DISn** register.

# Interrupts and exceptions of TM4C123GH6PM

**Configuring interrupt registers**

**2- Disable Maskable Interrupts**

- **Register 9: Interrupt 0-31 Clear Enable (DIS0), offset 0x180**
- **Register 10: Interrupt 32-63 Clear Enable (DIS1), offset 0x184**
- **Register 11: Interrupt 64-95 Clear Enable (DIS2), offset 0x188**
- **Register 12: Interrupt 96-127 Clear Enable (DIS3), offset 0x18C4**
- **Register 13: Interrupt 128-138 Clear Enable (DIS4), offset 0x190**

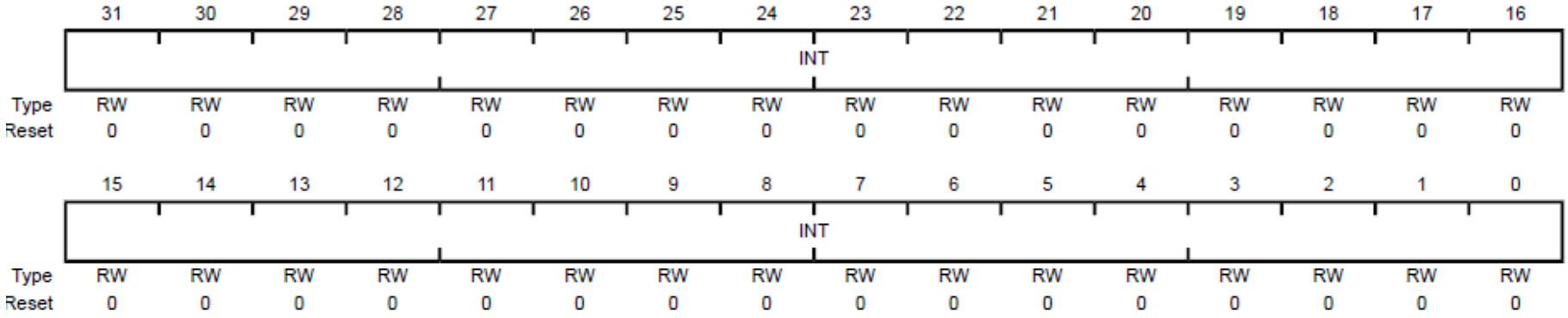
- Bit 0 of DIS0 corresponds to Interrupt 0; bit 31 corresponds to Interrupt 31.
- Bit 0 of DIS1 corresponds to Interrupt 32; bit 31 corresponds to Interrupt 63.
- Bit 0 of DIS2 corresponds to Interrupt 64; bit 31 corresponds to Interrupt 95.
- Bit 0 of DIS3 corresponds to Interrupt 96; bit 31 corresponds to Interrupt 127.
- Bit 0 of DIS4 corresponds to Interrupt 128; bit 10 corresponds to Interrupt 138.

# Interrupts and exceptions of TM4C123GH6PM

## Configuring interrupt registers
## 2- Disable Maskable Interrupts

Interrupt 0-31 Clear Enable (DIS0)

Base 0xE000.E000
Offset 0x180
Type RW, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | INT | RW | 0x0000.0000 | Interrupt Disable |

| Value | Description |
|---|---|
| 0 | On a read, indicates the interrupt is disabled.<br>On a write, no effect. |
| 1 | On a read, indicates the interrupt is enabled.<br>On a write, clears the corresponding `INT[n]` bit in the **EN0** register, disabling interrupt [n]. |

# Interrupts and exceptions of TM4C123GH6PM

**Configuring interrupt registers**

**3- Put the interrupt in pending state and check if it was pended**

- **Register 14: Interrupt 0-31 Set Pending (PEND0), offset 0x200**
- **Register 15: Interrupt 32-63 Set Pending (PEND1), offset 0x204**
- **Register 16: Interrupt 64-95 Set Pending (PEND2), offset 0x208**
- **Register 17: Interrupt 96-127 Set Pending (PEND3), offset 0x20C**
- **Register 18: Interrupt 128-138 Set Pending (PEND4), offset 0x210**

- Bit 0 of PEND0 corresponds to Interrupt 0; bit 31 corresponds to Interrupt 31.
- Bit 0 of PEND1 corresponds to Interrupt 32; bit 31 corresponds to Interrupt 63.
- Bit 0 of PEND2 corresponds to Interrupt 64; bit 31 corresponds to Interrupt 95.
- Bit 0 of PEND3 corresponds to Interrupt 96; bit 31 corresponds to Interrupt 127
- Bit 0 of PEND4 corresponds to Interrupt 128; bit 10 corresponds to Interrupt 138.

.

# Interrupts and exceptions of TM4C123GH6PM

## Configuring interrupt registers
## 3- Put the interrupt in pending state and check if t was pended

Interrupt 0-31 Set Pending (PEND0)

Base 0xE000.E000
Offset 0x200
Type RW, reset 0x0000.0000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| INT | | | | | | | | | | | | | | | |
| Type RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| INT | | | | | | | | | | | | | | | |
| Type RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | INT | RW | 0x0000.0000 | Interrupt Set Pending |

| Value | Description |
|---|---|
| 0 | On a read, indicates that the interrupt is not pending.<br>On a write, no effect. |
| 1 | On a read, indicates that the interrupt is pending.<br>On a write, the corresponding interrupt is set to pending even if it is disabled. |

If the corresponding interrupt is already pending, setting a bit has no effect.

A bit can only be cleared by setting the corresponding `INT[n]` bit in the **UNPEND0** register.

# Interrupts and exceptions of TM4C123GH6PM

**Configuring interrupt registers**

**4- Unpend maskable interrupts**

- **Register 19: Interrupt 0-31 Set Pending (UNPEND0), offset 0x280**
- **Register 20: Interrupt 32-63 Set Pending (UNPEND1), offset 0x284**
- **Register 21: Interrupt 64-95 Set Pending (UNPEND2), offset 0x288**
- **Register 22: Interrupt 96-127 Set Pending (UNPEND3), offset 0x28C**
- **Register 23: Interrupt 128-138 Set Pending (UNPEND4), offset 0x290**

- Bit 0 of UNPEND0 corresponds to Interrupt 0; bit 31 corresponds to Interrupt 31.
- Bit 0 of UNPEND1 corresponds to Interrupt 32; bit 31 corresponds to Interrupt 63.
- Bit 0 of UNPEND2 corresponds to Interrupt 64; bit 31 corresponds to Interrupt 95.
- Bit 0 of UNPEND3 corresponds to Interrupt 96; bit 31 corresponds to Interrupt 127
- Bit 0 of UNPEND4 corresponds to Interrupt 128; bit 10 corresponds to Interrupt 138.

# Interrupts and exceptions of TM4C123GH6PM

## Configuring interrupt registers
## 4- Unpend maskable interrupts

Interrupt 0-31 Clear Pending (UNPEND0)

Base 0xE000.E000
Offset 0x280
Type RW, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | INT | | | | | | | | | | | | | | | |
| Type | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | INT | | | | | | | | | | | | | | | |
| Type | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | INT | RW | 0x0000.0000 | Interrupt Clear Pending |

| Value | Description |
|---|---|
| 0 | On a read, indicates that the interrupt is not pending.<br>On a write, no effect. |
| 1 | On a read, indicates that the interrupt is pending.<br>On a write, clears the corresponding `INT[n]` bit in the **PEND0** register, so that interrupt [n] is no longer pending.<br>Setting a bit does not affect the active state of the corresponding interrupt. |

# Interrupts and exceptions of TM4C123GH6PM

**Configuring interrupt registers**

**5– Checking which interrupt is in being currently served(in active state)**

- **Register 24: Interrupt 0-31 Active Bit (ACTIVE0), offset 0x300**
- **Register 25: Interrupt 32-63 Active Bit (ACTIVE1), offset 0x304**
- **Register 26: Interrupt 64-95 Active Bit (ACTIVE2), offset 0x308**
- **Register 27: Interrupt 96-127 Active Bit (ACTIVE3), offset 0x30C**
- **Register 28: Interrupt 128-138 Active Bit (ACTIVE4), offset 0x310**

- Bit 0 of ACTIVE0 corresponds to Interrupt 0; bit 31 corresponds to Interrupt 31.
- Bit 0 of ACTIVE1 corresponds to Interrupt 32; bit 31 corresponds to Interrupt 63.
- Bit 0 of ACTIVE2 corresponds to Interrupt 64; bit 31 corresponds to Interrupt 95.
- Bit 0 of ACTIVE3 corresponds to Interrupt 96; bit 31 corresponds to Interrupt 127.
- Bit 0 of ACTIVE4 corresponds to Interrupt 128; bit 10 corresponds to Interrupt 138.

.

# Interrupts and exceptions of TM4C123GH6PM

## Configuring interrupt registers
## 5– Checking which interrupt is in being currently served(in active state)

**Caution – Do not manually set or clear the bits in this register.**

Interrupt 0-31 Active Bit (ACTIVE0)

Base 0xE000.E000
Offset 0x300
Type RO, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | INT | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | INT | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | INT | RO | 0x0000.0000 | Interrupt Active |

| Value | Description |
|---|---|
| 0 | The corresponding interrupt is not active. |
| 1 | The corresponding interrupt is active, or active and pending. |

# Interrupts and exceptions of TM4C123GH6PM

**Configuring interrupt registers**

**6- Setting interrupt priority**

- **Register 29: Interrupt 0-3 Priority (PRI0), offset 0x400**
- **Register 30: Interrupt 4-7 Priority (PRI1), offset 0x404**
- **Register 31: Interrupt 8-11 Priority (PRI2), offset 0x408**
- **Register 32: Interrupt 12-15 Priority (PRI3), offset 0x40C**
- **Register 33: Interrupt 16-19 Priority (PRI4), offset 0x410**
- **Register 34: Interrupt 20-23 Priority (PRI5), offset 0x414**
- **Register 35: Interrupt 24-27 Priority (PRI6), offset 0x418**
- **Register 36: Interrupt 28-31 Priority (PRI7), offset 0x41C**
- **Register 37: Interrupt 32-35 Priority (PRI8), offset 0x420**
- **Register 38: Interrupt 36-39 Priority (PRI9), offset 0x424**
- **Register 39: Interrupt 40-43 Priority (PRI10), offset 0x428**
- **Register 40: Interrupt 44-47 Priority (PRI11), offset 0x42C**
- **Register 41: Interrupt 48-51 Priority (PRI12), offset 0x430**
- **Register 42: Interrupt 52-55 Priority (PRI13), offset 0x434**
- **Register 43: Interrupt 56-59 Priority (PRI14), offset 0x438**
- **Register 44: Interrupt 60-63 Priority (PRI15), offset 0x43C**
- **Register 45: Interrupt 64-67 Priority (PRI16), offset 0x440**
- **Register 46: Interrupt 68-71 Priority (PRI17), offset 0x444**

# Interrupts and exceptions of TM4C123GH6PM

**Configuring interrupt registers**

**6- Setting interrupt priority**

- **Register 47: Interrupt 72-75 Priority (PRI18), offset 0x448**
- **Register 48: Interrupt 76-79 Priority (PRI19), offset 0x44C**
- **Register 49: Interrupt 80-83 Priority (PRI20), offset 0x450**
- **Register 50: Interrupt 84-87 Priority (PRI21), offset 0x454**
- **Register 51: Interrupt 88-91 Priority (PRI22), offset 0x458**
- **Register 52: Interrupt 92-95 Priority (PRI23), offset 0x45C**
- **Register 53: Interrupt 96-99 Priority (PRI24), offset 0x460**
- **Register 54: Interrupt 100-103 Priority (PRI25), offset 0x464**
- **Register 55: Interrupt 104-107 Priority (PRI26), offset 0x468**
- **Register 56: Interrupt 108-111 Priority (PRI27), offset 0x46C**
- **Register 57: Interrupt 112-115 Priority (PRI28), offset 0x470**
- **Register 58: Interrupt 116-119 Priority (PRI29), offset 0x474**
- **Register 59: Interrupt 120-123 Priority (PRI30), offset 0x478**
- **Register 60: Interrupt 124-127 Priority (PRI31), offset 0x47C**
- **Register 61: Interrupt 128-131 Priority (PRI32), offset 0x480**
- **Register 62: Interrupt 132-135 Priority (PRI33), offset 0x484**
- **Register 63: Interrupt 136-138 Priority (PRI34), offset 0x488**

# Interrupts and exceptions of TM4C123GH6PM

## Configuring interrupt registers

## 6- Setting interrupt priority

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

| PRIn Register Bit Field | Interrupt |
|---|---|
| Bits 31:29 | Interrupt [4n+3] |
| Bits 23:21 | Interrupt [4n+2] |
| Bits 15:13 | Interrupt [4n+1] |
| Bits 7:5 | Interrupt [4n] |

Interrupt 0-3 Priority (PRI0)

Base 0xE000.E000
Offset 0x400
Type RW, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | INTD | | | reserved | | | | | INTC | | | reserved | | | | |
| Type | RW | RW | RW | RO | RO | RO | RO | RO | RW | RW | RW | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | INTB | | | reserved | | | | | INTA | | | reserved | | | | |
| Type | RW | RW | RW | RO | RO | RO | RO | RO | RW | RW | RW | RO | RO | RO | RO | RO |

# Interrupts and exceptions of TM4C123GH6PM

## Configuring interrupt registers

## 7- Setting group priority

- The APINT register provides priority grouping control for the exception model, endian status for data accesses, and reset control of the system.
- To write to this register, 0x05FA must be written to the VECTKEY field, otherwise the write is ignored.
- The PRIGROUP field indicates the position of the binary point that splits the INTx fields in the Interrupt Priority (PRIx) registers into separate group priority and subpriority fields.

| PRIGROUP Bit Field | Binary Point[a] | Group Priority Field | Subpriority Field | Group Priorities | Subpriorities |
|---|---|---|---|---|---|
| 0x0 - 0x4 | bxxx. | [7:5] | None | 8 | 1 |
| 0x5 | bxx.y | [7:6] | [5] | 4 | 2 |
| 0x6 | bx.yy | [7] | [6:5] | 2 | 4 |
| 0x7 | b.yyy | None | [7:5] | 1 | 8 |

a. INTx field showing the binary point. An x denotes a group priority field bit, and a y denotes a subpriority field bit.

- Field columns in the table refer to the bits in the INTA field. For the INTB field, the corresponding bits are 15:13; for INTC, 23:21; and for INTD, 31:29.

# Interrupts and exceptions of TM4C123GH6PM

## Configuring interrupt registers
## 7- Setting group priority

Application Interrupt and Reset Control (APINT)

Base 0xE000.E000
Offset 0xD0C
Type RW, reset 0xFA05.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | VECTKEY | | | | | | | | | | | | | | | |
| Type | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ENDIANESS | reserved | | | | PRIGROUP | | | reserved | | | | | SYSRESREQ | VECTCLRACT | VECTRESET |
| Type | RO | RO | RO | RO | RO | RW | RW | RW | RO | RO | RO | RO | RO | WO | WO | WO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# Interrupts and exceptions of TM4C123GH6PM

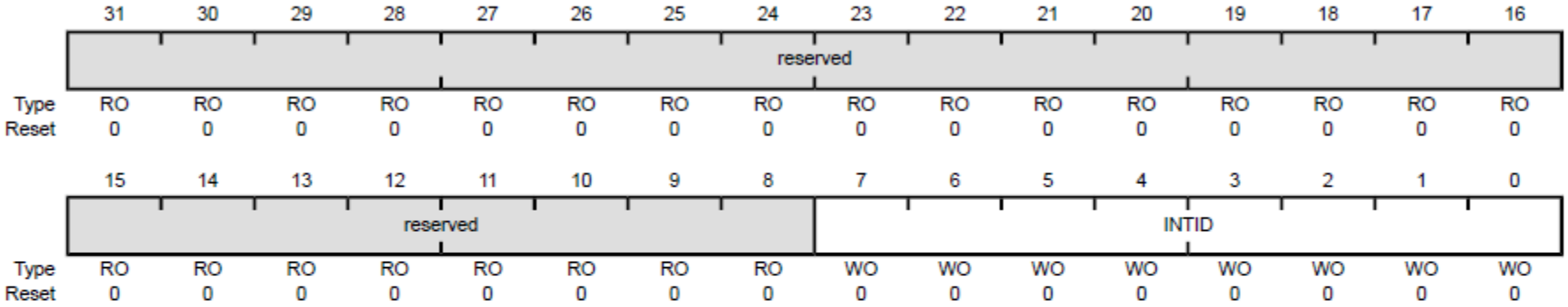
## Configuring interrupt registers

## 8- Triggering a software interrupt

- Writing an interrupt number to the SWTRIG register generates a Software Generated Interrupt (SGI).

Software Trigger Interrupt (SWTRIG)

Base 0xE000.E000
Offset 0xF00
Type WO, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x0000.00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | INTID | WO | 0x00 | Interrupt ID<br>This field holds the interrupt ID of the required SGI. For example, a value of 0x3 generates an interrupt on IRQ3. |

# Agenda

1. Overview on ARM architecture
2. ARM Cortex-M4 and ARM Cortex-M3 Specifications
3. TM4C123GH6PM Microcontroller Peripherals.
4. TIVA TM4C123GH6PM Launchpad kit specifications
5. GPIO Interface with applications
6. ADC Interface with applications
7. Interrupts and exceptions of TM4C123GH6PM.
8. SPI Interface with applications.
9. I2C Interface with applications.
10. UART Interface.
11. DMA and its applications.
12. Timers and PWM interfacing .

# 6. DMA (Direct Memory Access)

# DMA (Direct Memory Access)

**Introduction**

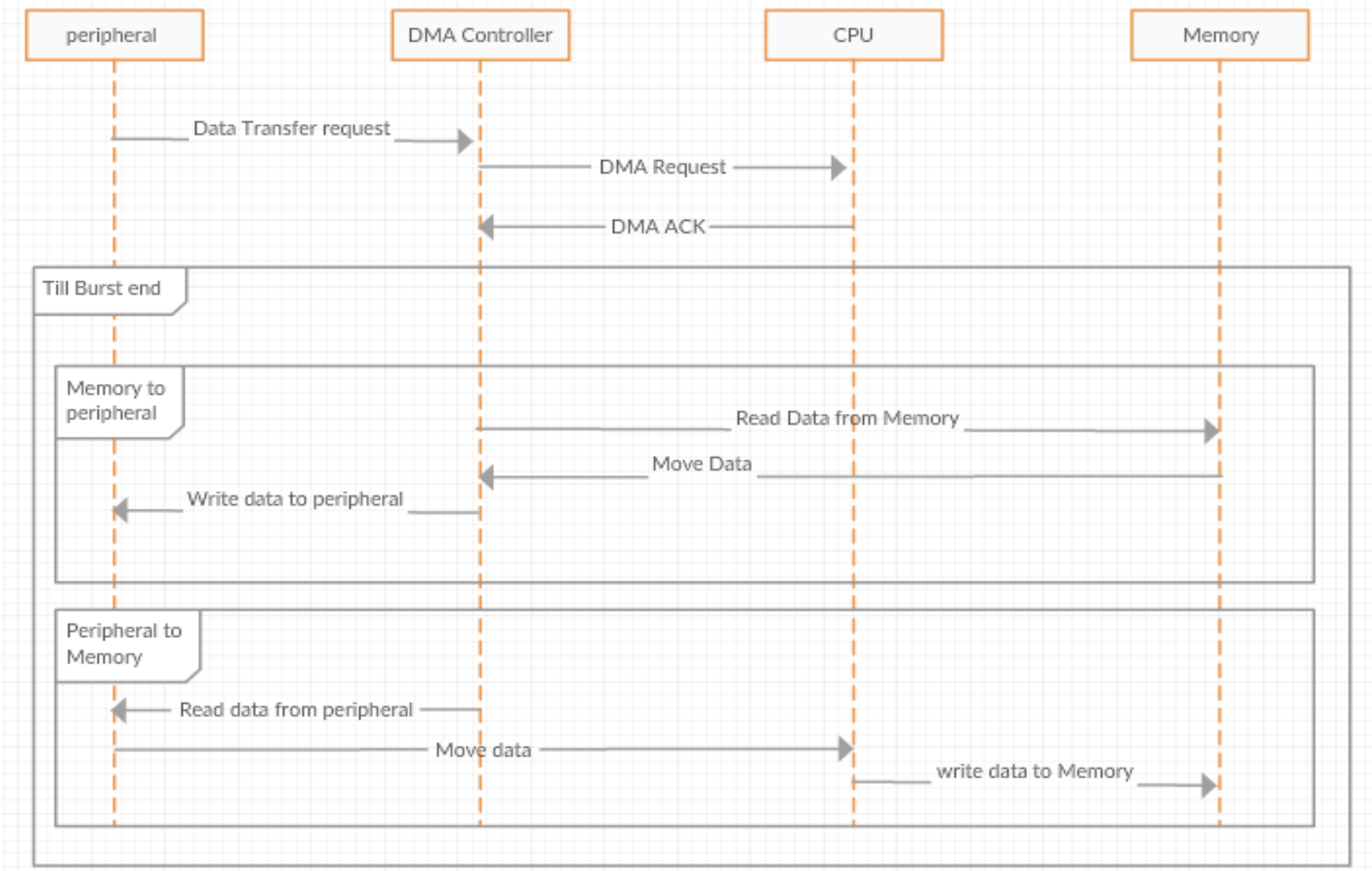
- Direct memory access (DMA) is a means of having a peripheral device control a processor's memory bus directly. DMA permits the peripheral, such as a UART, to transfer data directly to or from memory without having each byte (or word) handled by the processor.

**Benefits:**

- DMA enables more efficient use of interrupts.
- increases data throughput.
- potentially reduces hardware costs by eliminating the need for peripheral-specific FIFO buffers.

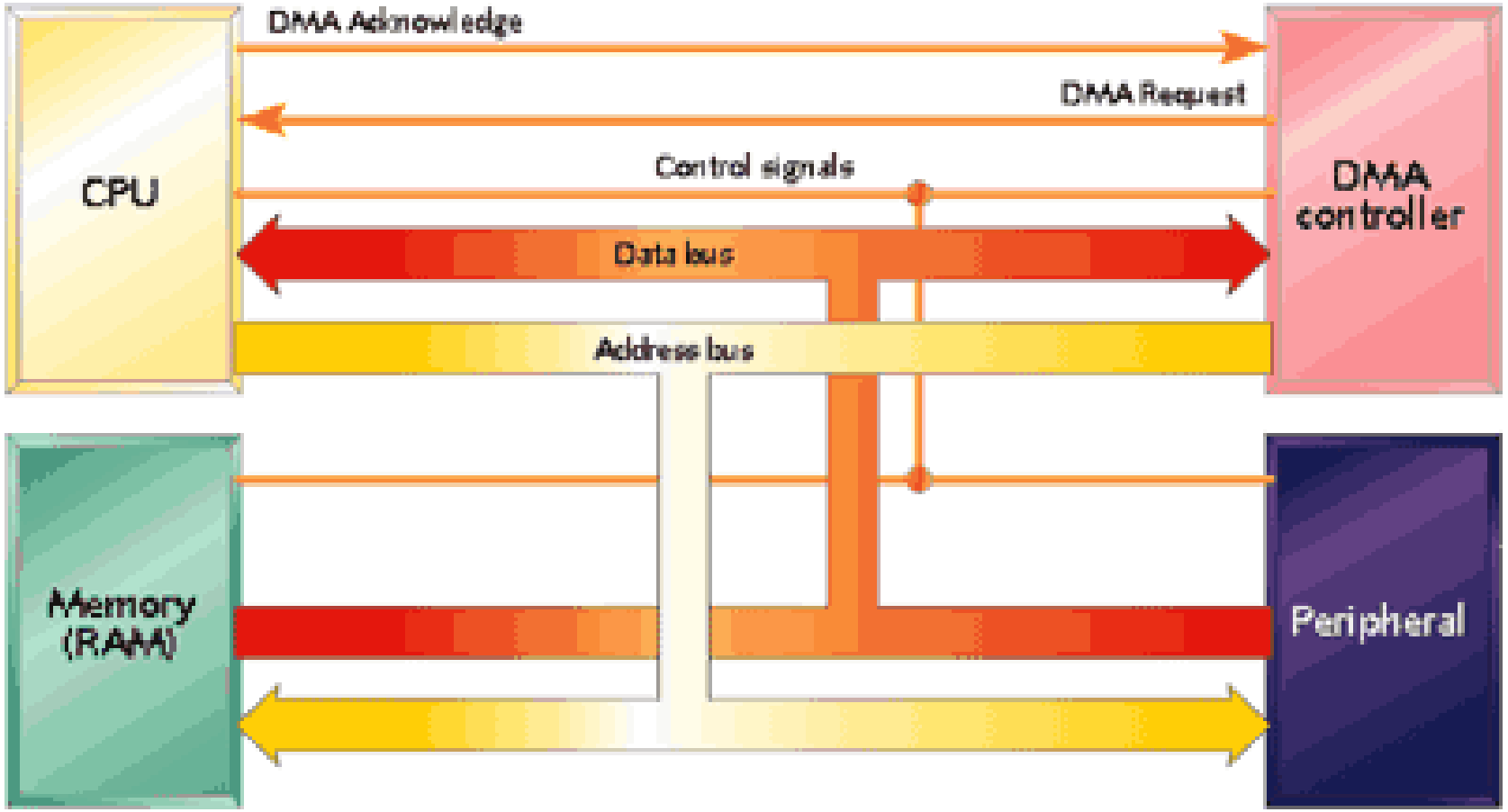
# DMA (Direct Memory Access)

**How does it work?**



# DMA (Direct Memory Access)

**How does it work?**



# DMA (Direct Memory Access)

**ARM TM4C123GH6PM features:**

- ARM® PrimeCell® 32-channel configurable µDMA controller.
- Support for memory-to-memory, memory-to-peripheral, and peripheral-to-memory in multiple transfer modes.
- Highly flexible and configurable channel operation.
- Two levels of priority.
- Design optimizations for improved bus access performance between µDMA controller and the processor core.
- Data sizes of 8, 16, and 32 bits.
- Transfer size is programmable in binary steps from 1 to 1024.
- Source and destination address increment size of byte, half-word, word, or no increment.
- Maskable peripheral requests.
- Interrupt on transfer completion, with a separate interrupt per channel

# DMA (Direct Memory Access)

## Functional Description:

## 1- Channel assignments:

- Each DMA channel has up to five possible assignments.
- The Type indicates if a particular peripheral uses a single request (S), burst request (B) or either (SB).

| Enc. | 0 | | 1 | | 2 | | 3 | | 4 | |
|---|---|---|---|---|---|---|---|---|---|---|
| Ch # | Peripheral | Type | Peripheral | Type | Peripheral | Type | Peripheral | Type | Peripheral | Type |
| 0 | USB0 EP1 RX | SB | UART2 RX | SB | Software | B | GPTimer 4A | B | Software | B |
| 1 | USB0 EP1 TX | B | UART2 TX | SB | Software | B | GPTimer 4B | B | Software | B |
| 2 | USB0 EP2 RX | B | GPTimer 3A | B | Software | B | Software | B | Software | B |
| 3 | USB0 EP2 TX | B | GPTimer 3B | B | Software | B | Software | B | Software | B |
| 4 | USB0 EP3 RX | B | GPTimer 2A | B | Software | B | GPIO A | B | Software | B |
| 5 | USB0 EP3 TX | B | GPTimer 2B | B | Software | B | GPIO B | B | Software | B |
| 6 | Software | B | GPTimer 2A | B | UART5 RX | SB | GPIO C | B | Software | B |
| 7 | Software | B | GPTimer 2B | B | UART5 TX | SB | GPIO D | B | Software | B |
| 8 | UART0 RX | SB | UART1 RX | SB | Software | B | GPTimer 5A | B | Software | B |
| 9 | UART0 TX | SB | UART1 TX | SB | Software | B | GPTimer 5B | B | Software | B |

# DMA (Direct Memory Access)

## Functional Description:
## 1- Channel assignments:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 10 | SSI0 RX | SB | SSI1 RX | SB | UART6 RX | SB | GPWideTimer 0A | B | Software | B |
| 11 | SSI0 TX | SB | SSI1 TX | SB | UART6 TX | SB | GPWideTimer 0B | B | Software | B |
| 12 | Software | B | UART2 RX | SB | SSI2 RX | SB | GPWideTimer 1A | B | Software | B |
| 13 | Software | B | UART2 TX | SB | SSI2 TX | SB | GPWideTimer 1B | B | Software | B |
| 14 | ADC0 SS0 | B | GPTimer 2A | B | SSI3 RX | SB | GPIO E | B | Software | B |
| 15 | ADC0 SS1 | B | GPTimer 2B | B | SSI3 TX | SB | GPIO F | B | Software | B |
| 16 | ADC0 SS2 | B | Software | B | UART3 RX | SB | GPWideTimer 2A | B | Software | B |
| 17 | ADC0 SS3 | B | Software | B | UART3 TX | SB | GPWideTimer 2B | B | Software | B |
| 18 | GPTimer 0A | B | GPTimer 1A | B | UART4 RX | SB | GPIO B | B | Software | B |
| 19 | GPTimer 0B | B | GPTimer 1B | B | UART4 TX | SB | Software | B | Software | B |
| 20 | GPTimer 1A | B | Software | B | UART7 RX | SB | Software | B | Software | B |

# DMA (Direct Memory Access)

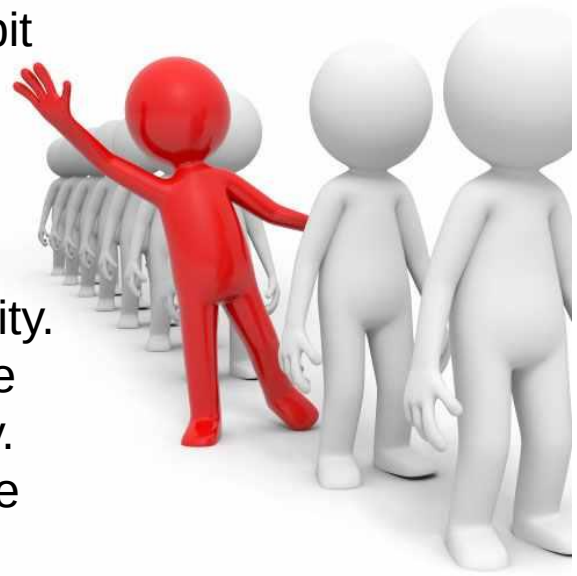
## Functional Description:
## 1- Channel assignments:

| Enc. | 0 | | 1 | | 2 | | 3 | | 4 | |
|---|---|---|---|---|---|---|---|---|---|---|
| Ch # | Peripheral | Type | Peripheral | Type | Peripheral | Type | Peripheral | Type | Peripheral | Type |
| 21 | GPTimer 1B | B | Software | B | UART7 TX | SB | Software | B | Software | B |
| 22 | UART1 RX | SB | Software | B | Software | B | Software | B | Software | B |
| 23 | UART1 TX | SB | Software | B | Software | B | Software | B | Software | B |
| 24 | SSI1 RX | SB | ADC1 SS0 | B | Software | B | GPWideTimer 3A | B | Software | B |
| 25 | SSI1 TX | SB | ADC1 SS1 | B | Software | B | GPWideTimer 3B | B | Software | B |
| 26 | Software | B | ADC1 SS2 | B | Software | B | GPWideTimer 4A | B | Software | B |
| 27 | Software | B | ADC1 SS3 | B | Software | B | GPWideTimer 4B | B | Software | B |
| 28 | Software | B | Software | B | Software | B | GPWideTimer 5A | B | Software | B |
| 29 | Software | B | Software | B | Software | B | GPWideTimer 5B | B | Software | B |
| 30 | Software | B | Software | B | Software | B | Software | B | Software | B |
| 31 | Reserved | B | Reserved | B | Reserved | B | Reserved | B | Reserved | B |

# DMA (Direct Memory Access)

**Functional Description:**
**2- Channel Priority:**

- The µDMA controller assigns priority to each channel based on the channel number and the priority level bit for the channel.
- Each channel has a priority level bit to provide two levels of priority: default priority and high priority.
- If the priority level bit is set, then that channel has higher priority than all other channels at default priority.
- If multiple channels are set for same priority, then the channel number is used to determine relative priority.
- Channel number 0 has the highest priority and as the channel number increases, the priority of a channel decreases.



# DMA (Direct Memory Access)

**Functional Description:**
**3- Arbitration size:**

- The arbitration size can also be thought of as a burst size. It is the maximum number of items that are transferred at any one time in a burst.
- The arbitration size can be configured for each channel, ranging from 1 to 1024 item transfers.
- When a µDMA channel requests a transfer, the µDMA controller arbitrates among all the channels making a request and services the µDMA channel with the highest priority.
- Once a transfer begins, it continues for a selectable number of transfers before rearbitrating among the requesting channels again.
- lower priority channels should not use a large arbitration size for best response on high priority channels.

# DMA (Direct Memory Access)

**Functional Description:**

**4- Channel Configuration structure:**

- The µDMA controller uses an area of system memory to store a set of channel control structures in a table.
- Each entry in the table structure contains source and destination pointers, transfer size, and transfer mode.
- The control table can be located anywhere in system memory, but it must be contiguous and aligned on a 1024-byte boundary.

**Table 9-4. Channel Control Structure**

| Offset | Description |
|---|---|
| 0x000 | Source End Pointer |
| 0x004 | Destination End Pointer |
| 0x008 | Control Word |
| 0x00C | Unused |

# DMA (Direct Memory Access)

**Functional Description:**
**4- Channel Configuration structure:**

The control word contains the following fields:

- Source and destination data sizes
- Source and destination address increment size
- Number of transfers before bus arbitration
- Total number of items to transfer
- Useburst flag
- Transfer mode

# DMA (Direct Memory Access)

## Functional Description:
## 4- Channel Configuration structure:

- Each channel may have one or two control structures in the control table: a primary control structure and an optional alternate control structure.
- The table is organized so that all of the primary entries are in the first half of the table, and all the alternate structures are in the second half of the table.
- The primary entry is used for simple transfer modes where transfers can be reconfigured and restarted after each transfer is complete.

**Table 9-3. Control Structure Memory Map**

| Offset | Channel |
|---|---|
| 0x0 | 0, Primary |
| 0x10 | 1, Primary |
| ... | ... |
| 0x1F0 | 31, Primary |
| 0x200 | 0, Alternate |
| 0x210 | 1, Alternate |
| ... | ... |
| 0x3F0 | 31, Alternate |

# DMA (Direct Memory Access)

**Functional Description:**

**5- Types of requests:**

**1- Single Request:**

- When a single request is detected, and not a burst request, the µDMA controller transfers one item and then stops to wait for another request.

**2- Burst Request:**

- When a burst request is detected, the µDMA controller transfers the number of items that is the lesser of the arbitration size or the number of items remaining in the transfer.
- the arbitration size should be the same as the number of data items that the peripheral can accommodate when making a burst request.
- A burst transfer runs to completion once it is started, and cannot be interrupted, even by a higher priority channel.
- Burst transfers complete in a shorter time than the same number of non-burst transfers.

# DMA (Direct Memory Access)

## Functional Description:
## 5- Types of requests:

**Table 9-2. Request Type Support**

| Peripheral | Event that generates Single Request | Event that generates Burst Request |
|---|---|---|
| ADC | None | FIFO half full |
| General-Purpose Timer | None | Trigger event |
| GPIO | Raw interrupt pulse | None |
| SSI TX | TX FIFO Not Full | TX FIFO Level (fixed at 4) |
| SSI RX | RX FIFO Not Empty | RX FIFO Level (fixed at 4) |
| UART TX | TX FIFO Not Full | TX FIFO Level (configurable) |
| UART RX | RX FIFO Not Empty | RX FIFO Level (configurable) |
| USB TX | None | FIFO TXRDY |
| USB RX | None | FIFO RXRDY |

# DMA (Direct Memory Access)

## Functional Description:
## 6- Transfere modes:

**- Stop Mode**

- The µDMA controller does not perform any transfers and disables the channel if it is enabled.
- At the end of a transfer, the µDMA controller updates the control word to set the mode to Stop.

# DMA (Direct Memory Access)

**Functional Description:**
**6- Transfere modes:**

- **Basic Mode**

- the µDMA controller performs transfers as long as there are more items to transfer,  and a transfer request is present.
- Only the number of transfers specified by the ARBSIZE field in the **DMA Channel Control Word (DMACHCTL)** register is transferred on a software request, even if there is more data to transfer.
- This mode is used with peripherals that assert a µDMA request signal whenever the peripheral is ready for a data transfer.
- Basic mode should not be used in any situation where the request is momentary even though the entire transfer should be completed.
- When all of the items have been transferred using Basic mode, the µDMA controller sets the mode for that channel to Stop.

# DMA (Direct Memory Access)

**Functional Description:**
**6- Transfere modes:**

- **Auto Mode**

- Auto mode is similar to Basic mode, except that once a transfer request is received, the transfer runs to completion, even if the µDMA request is removed.
- This mode is suitable for software-triggered transfers. Generally, Auto mode is not used with a peripheral.
- When all the items have been transferred using Auto mode, the µDMA controller sets the mode for that channel to Stop.

# DMA (Direct Memory Access)

**Functional Description:**
**6- Transfere modes:**

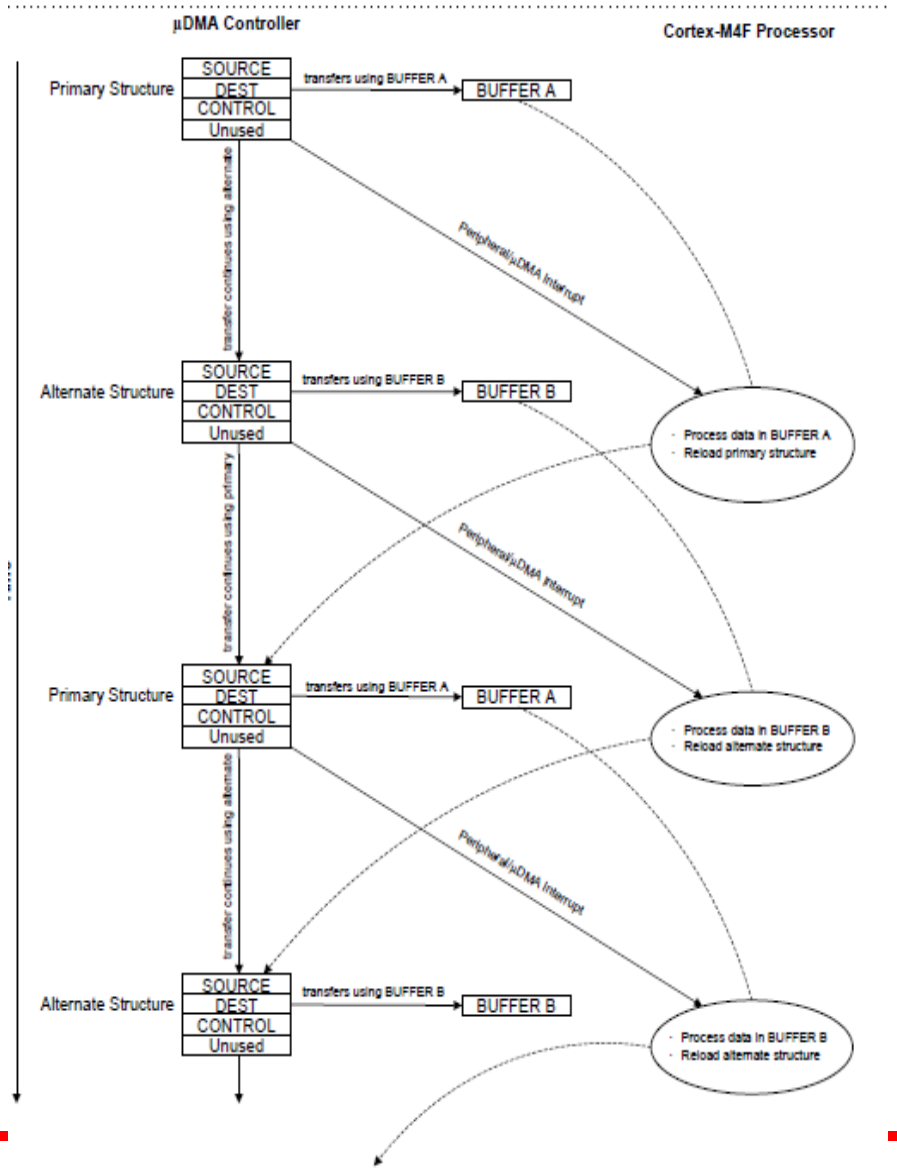
- **Ping-Pong**

- Ping-Pong mode is used to support a continuous data flow to or from a peripheral.
- To use Ping-Pong mode, both the primary and alternate data structures must be implemented.
- The transfer is started using the primary control structure and after complete the µDMA controller reads the alternate control structure for that channel to continue the transfer.
- Each time this happens, an interrupt is generated, and the processor can reload the control structure for the just-completed transfer.

# DMA (Direct Memory Access)

## Functional Description:
## 6- Transfere modes:

- **Ping-Pong**



# DMA (Direct Memory Access)

**Functional Description:**

**7- Transfer Size and Increment**

- The µDMA controller supports transfer data sizes of 8, 16, or 32 bits.
- The source and destination data size must be the same for any given transfer.
- The source and destination address can be auto-incremented **independently** by bytes, half-words, or words, or can be set to no increment.
- it is not necessary for the address increment to match the data size as long as the increment is the same or larger than the data size.

**Table 9-5. µDMA Read Example: 8-Bit Peripheral**

| Field | Configuration |
|---|---|
| Source data size | 8 bits |
| Destination data size | 8 bits |
| Source address increment | No increment |
| Destination address increment | Byte |
| Source end pointer | Peripheral read FIFO register |
| Destination end pointer | End of the data buffer in memory |

# DMA (Direct Memory Access)

**Functional Description:**
**8- Software Request and peripheral transfer:**

- One µDMA channel is dedicated to software-initiated transfers.
- A transfer is initiated by software by first configuring and enabling the transfer, and then issuing a software request using the DMA Channel Software Request (DMASWREQ) register.
- For software-based transfers, the Auto transfer mode should be used.

- Each peripheral that supports µDMA has a single request and/or burst request signal that is asserted when the peripheral is ready to transfer data
- The request signal can be disabled or enabled using the **DMA Channel Request Mask Set (DMAREQMASKSET)** and **DMA Channel Request Mask Clear (DMAREQMASKCLR)** registers.
- When a µDMA transfer is complete, the µDMA controller generates an interrupt,
- For more information on how a specific peripheral interacts with the µDMA controller, refer to the DMA Operation section in the chapter that discusses that peripheral.

# DMA (Direct Memory Access)

**Functional Description:**

**9 – Interrupts and errors:**

- When a µDMA transfer is complete, the µDMA controller generates a completion interrupt on the interrupt vector of the peripheral.
- if µDMA is used to transfer data for a peripheral and interrupts are used, then the interrupt handler for that peripheral must be designed to handle the µDMA transfer completion interrupt.
- If the transfer uses the software µDMA channel, then the completion interrupt occurs on the dedicated software µDMA interrupt vector.
- When µDMA is enabled for a peripheral, the µDMA controller stops the normal transfer interrupts for a peripheral from reaching the interrupt controller.
- when a large amount of data is transferred using µDMA, instead of receiving multiple interrupts from the peripheral as data flows, the interrupt controller receives only one interrupt when the transfer is complete.
- If the µDMA controller encounters a bus or memory protection error as it attempts to perform a data transfer, it disables the µDMA channel that caused the error and generates an interrupt on the µDMA error interrupt vector.

**Table 9-6. µDMA Interrupt Assignments**

| Interrupt | Assignment |
|---|---|
| 46 | µDMA Software Channel Transfer |
| 47 | µDMA Error |

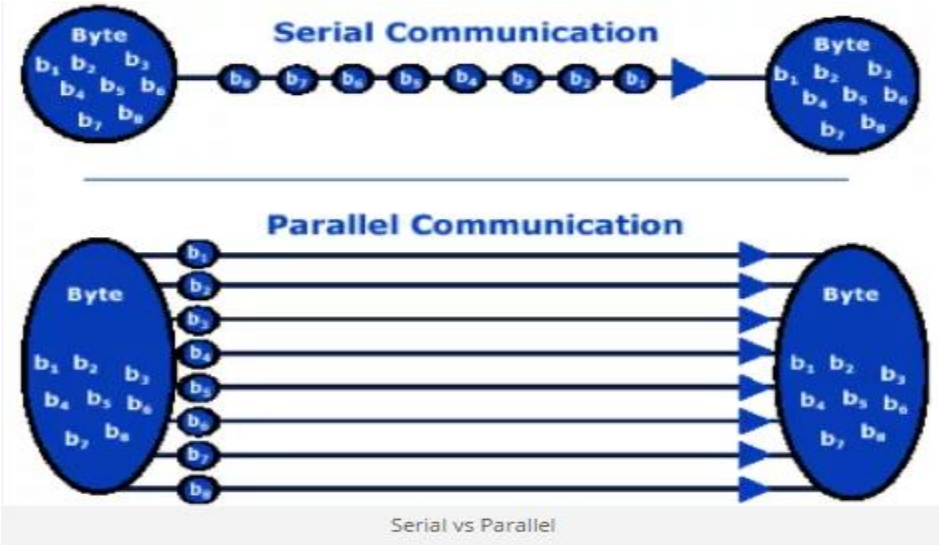
# DMA (Direct Memory Access)

**Initialization and configuration:**
**See datasheets page 600**

# 7. Serial Interface protocols

# Serial Interface Protocols

## Serial interface Vs. Parallel interface

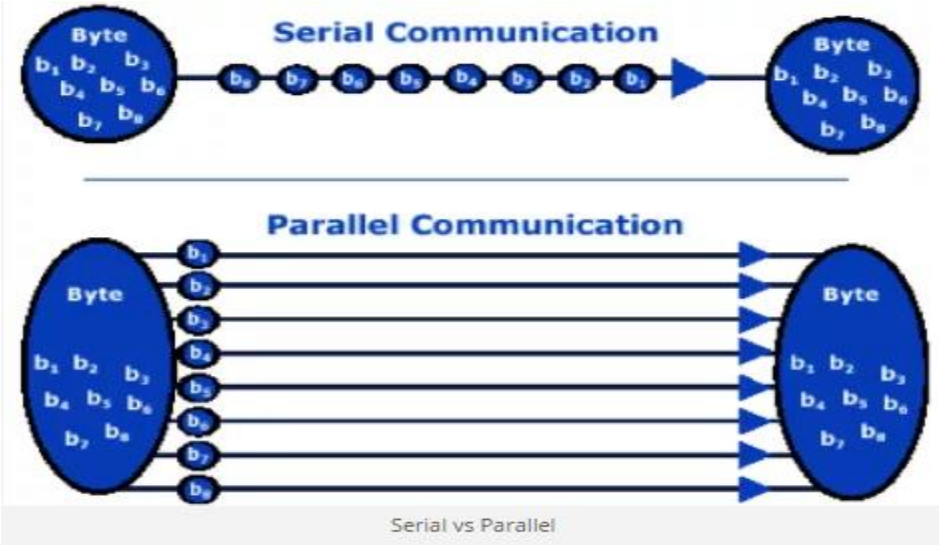


Serial vs Parallel

- In parallel mode, each bit has a single wire devoted to it and all the bits are transmitted at the same time.
- This method of transmission can move a significant amount of data in a given period of time.
- Its disadvantage is the large number of interconnecting cables between the two units.
- It also require special interfacing to minimize noise and distortion problems.

# Serial Interface Protocols

## Serial interface Vs. Parallel interface

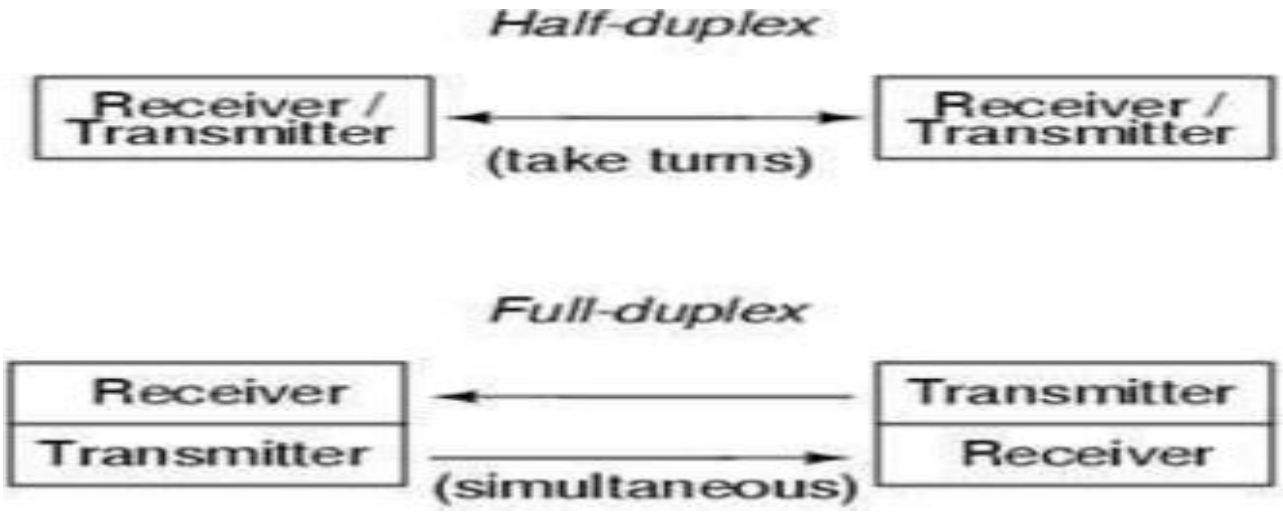


Serial vs Parallel

- Serial data transmission is the process of transmitting binary words a bit at a time.
- Since the bits time-share the transmission medium, only one interconnecting lead is required.
- It is much simpler and less expensive because of the use of a single interconnecting line.
- But it is a very slow method of data transmission. So is useful in systems where high speed is not a requirement.

# Serial Interface Protocols

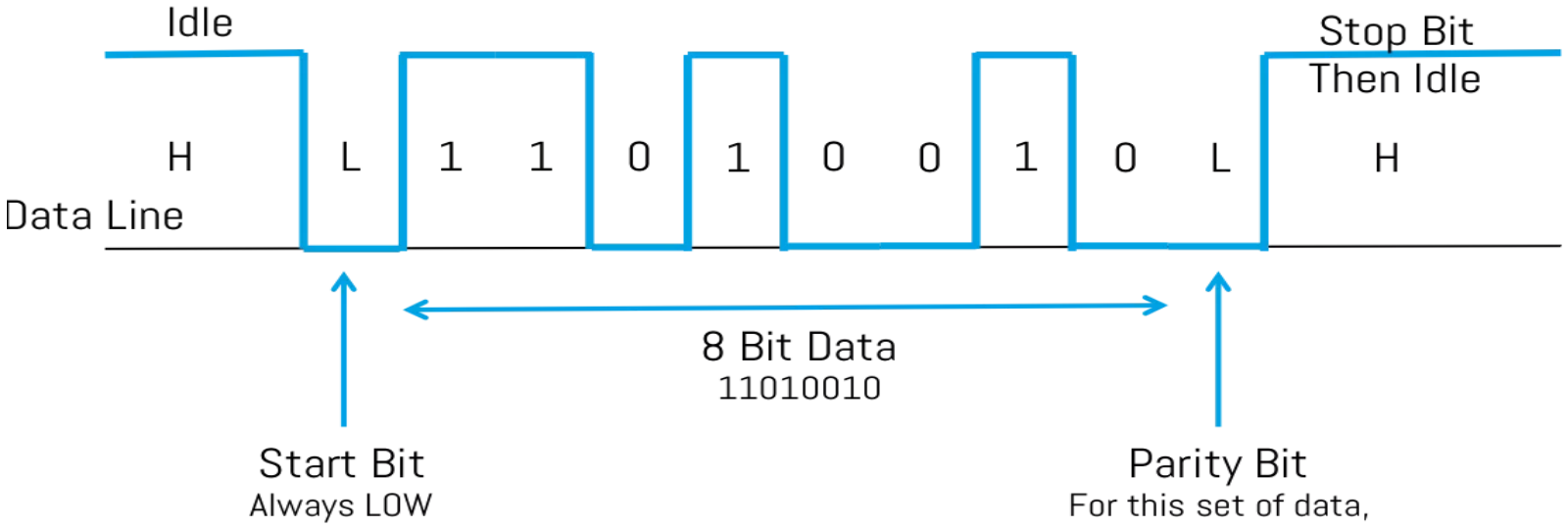
## Duplex techniques in serial communication




- Full-duplex communication between two components means that both can transmit and receive information between each other simultaneously.
- In half-duplex systems, the transmission and reception of information must happen alternately. While one point is transmitting, the other must only receive.

# Serial Interface Protocols

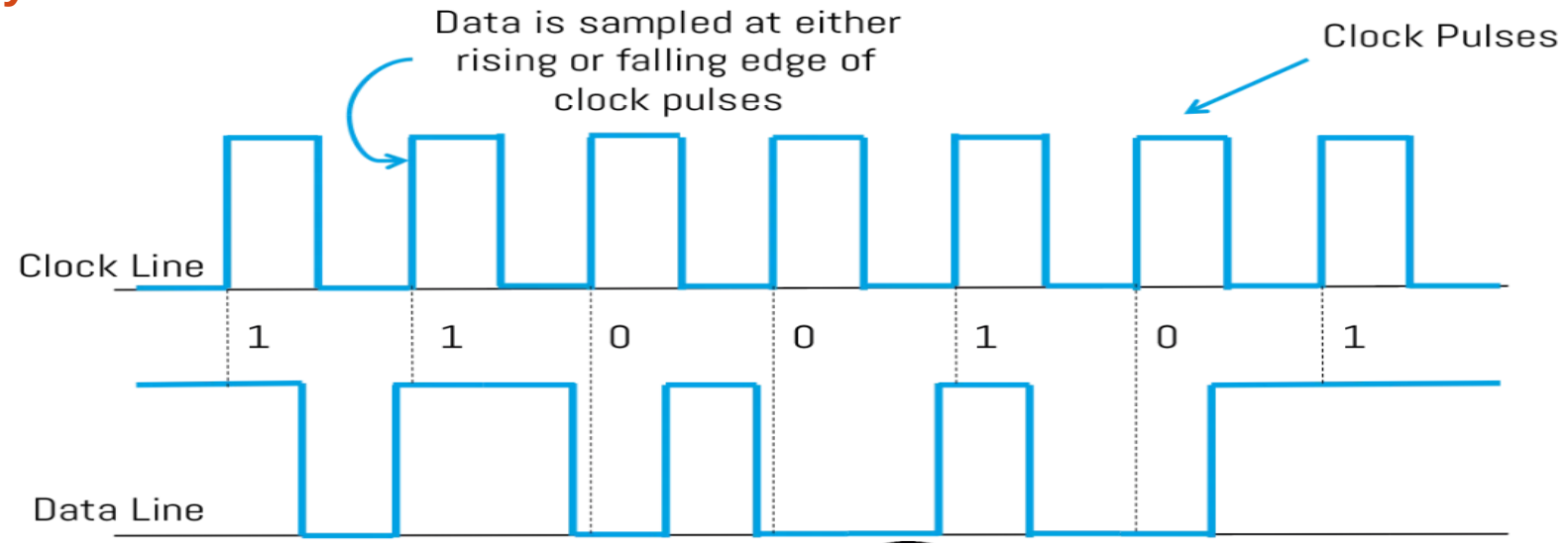
## Asynchrounous Data Transfer




- Data Transfer is called Asynchronous when data bits are not “synchronized” with a clock line, i.e. there is no clock line at all!
- Asynchronous data transfer has a protocol, which is *usually* as follows:
- The first bit is always the START bit ,followed by DATA bits followed by a STOP bit
- A way of error detection would be added.
- The START bit is always low (0) while the STOP bit is always high (1).

# Serial Interface Protocols

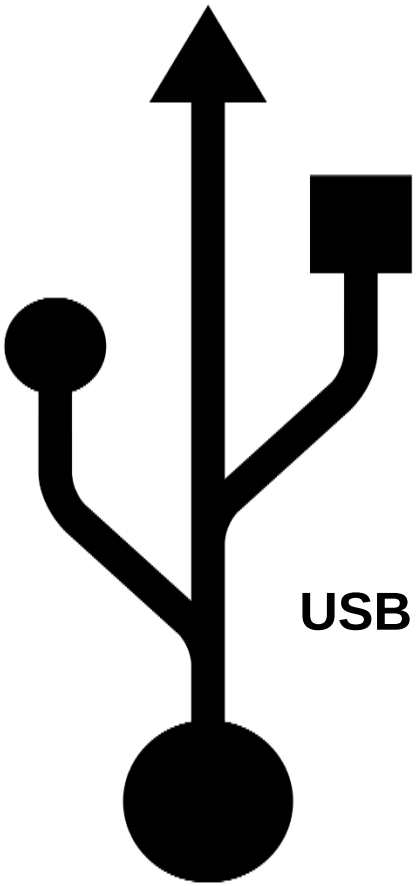
## Synchronous Data Transfer



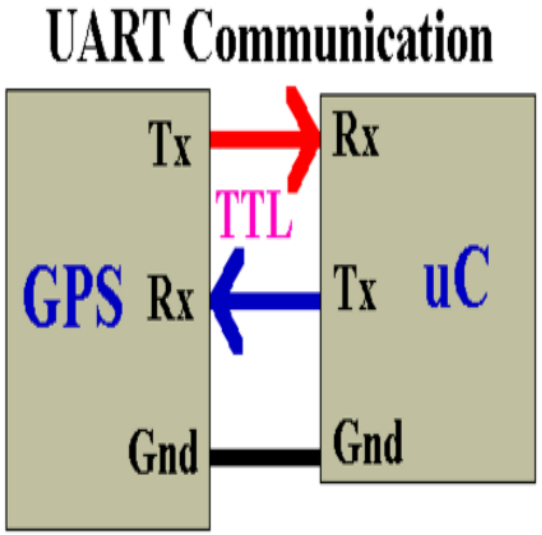

- Synchronous data transfer is when the data bits are “synchronized” with a clock pulse.
- The basic principle is that data bit sampling (or in other words, say, ‘recording’) is done with respect to clock pluses, as you can see in the timing diagrams.
- Since data is sampled depending upon clock pulses, and since the clock sources are very reliable, so there is much less error in synchronous as compared to asynchronous.

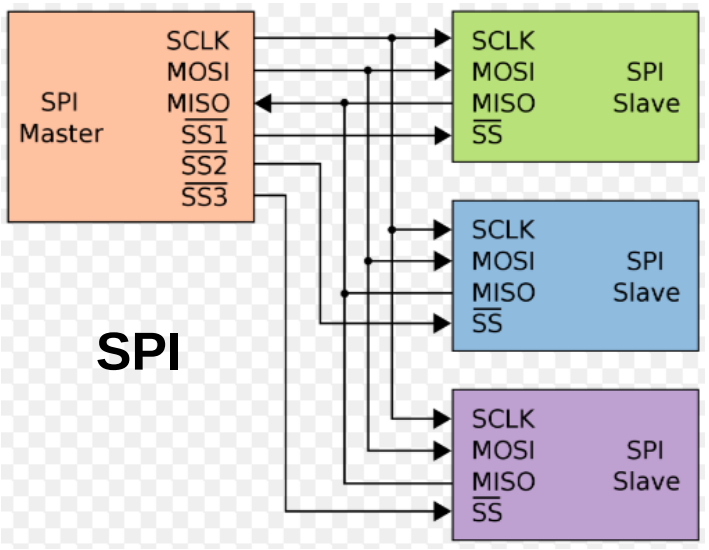
# Serial Interface Protocols

**Examples of serial interface protocols:**

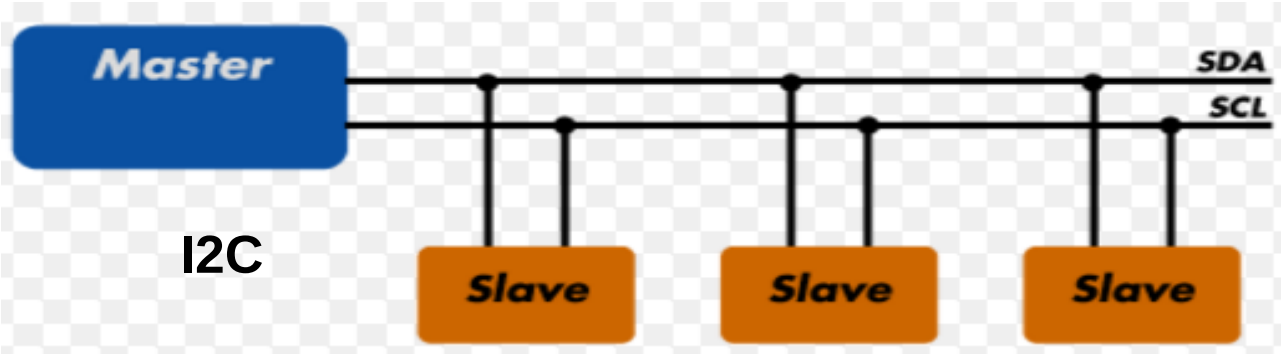


USB



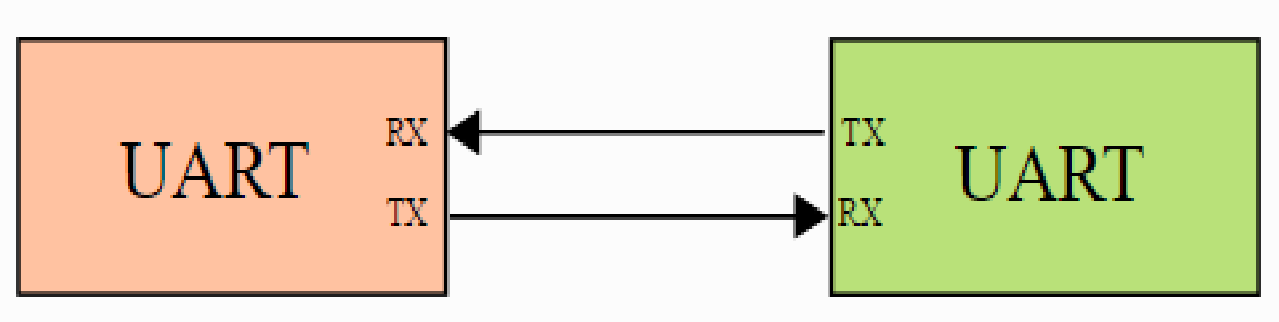

SPI




I2C

# Serial Interface Protocols: UART

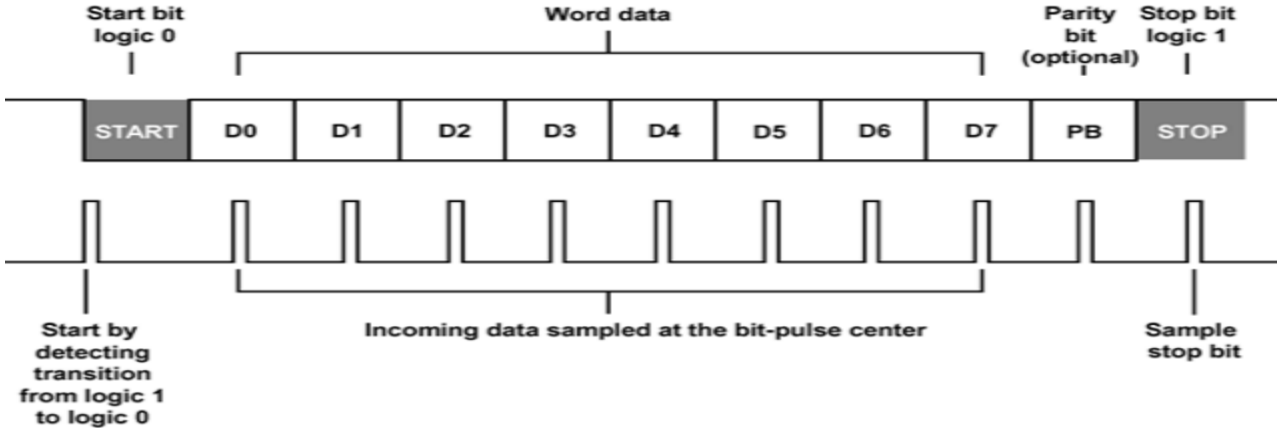
## What is UART?




- A Universal Asynchronous Receiver/Transmitter (UART) is a hardware component that can transmit and receive binary data (digital data) serially.
- A UART transmits bytes of data over a single wire one bit at a time to be received by another UART at the other end.
- The receiving UART reconstitutes the bits back into bytes.
- A UART communicates using a pair of wires, one for transmitting data (TX wire) and one for receiving it (RX wire).
- The TX wire of one UART gets connected to the RX wire of the other.

# Serial Interface Protocols: UART

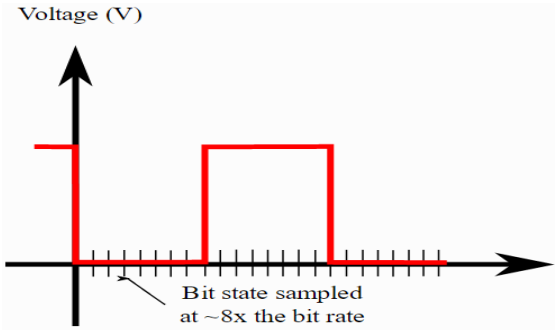
## UART byte framing:

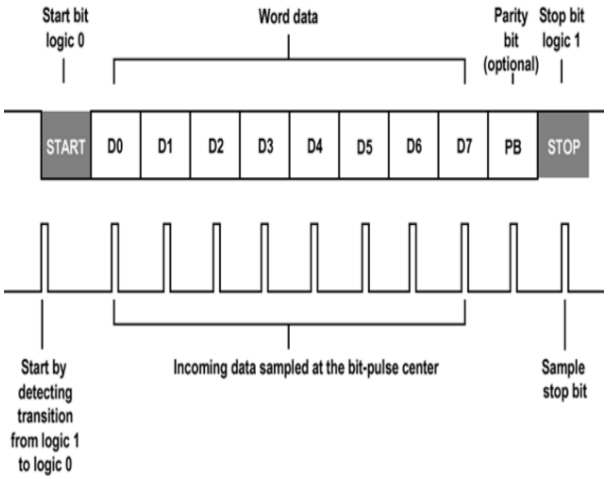


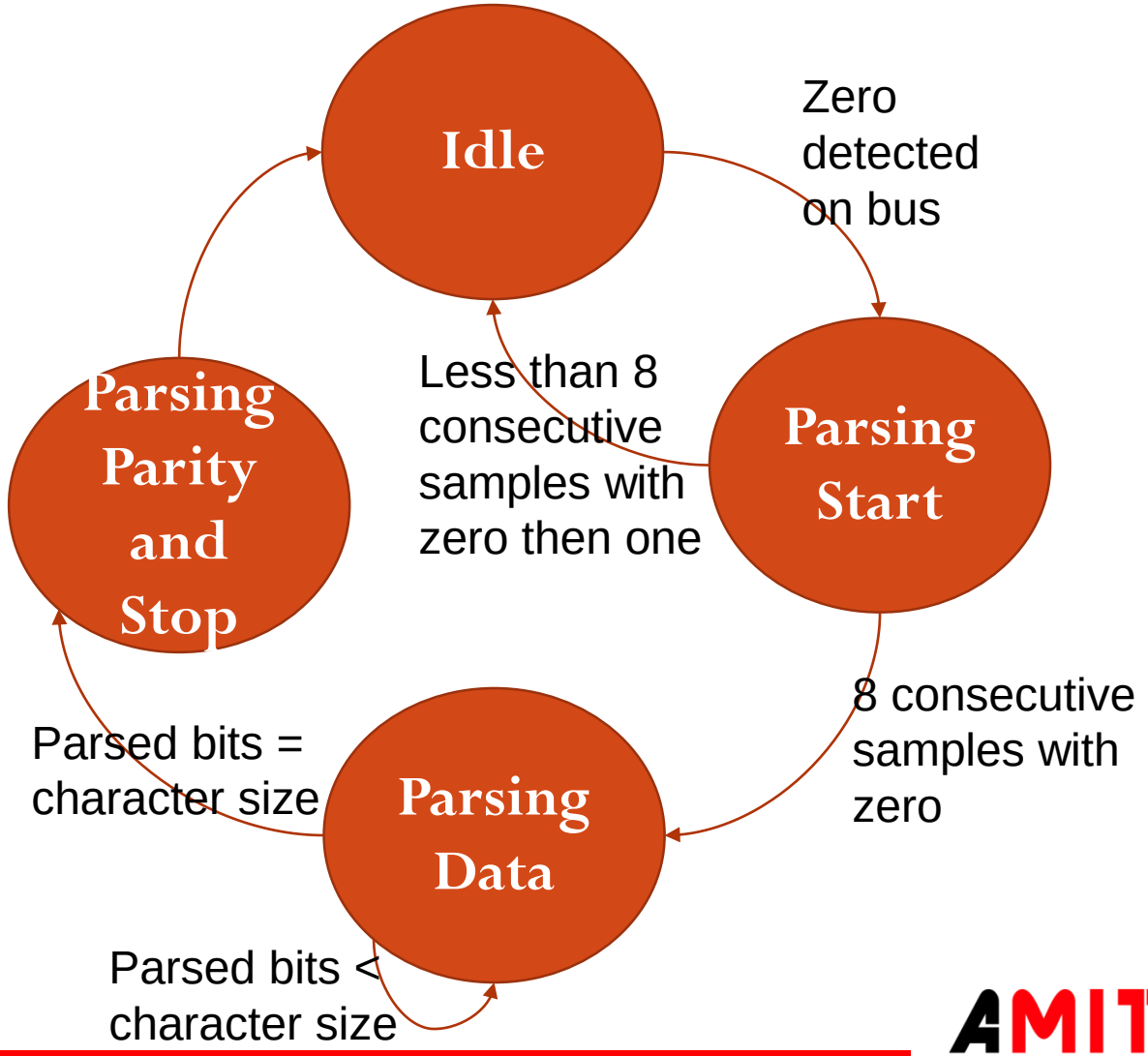

- When idling, or not transmitting any data, the wire voltage is held high.
- When the transmitting UART is ready to transmit a byte, it must first transmit a start The start bit is always 0. This tells the receiver that a byte is incoming, the next bit will be the first bit of the byte being transmitted.
- Bits are transmitted starting with the least significant digit finishing with the most significant..
- After the 8th bit, a stop bit is transmitted which always has a value of 1. At least one stop bit is required.
- If no more bytes are pending transmission, the transmitter will go idle with a high voltage level same as the stop bit.
- An optimal parity bit could be added before the stop bit and its value is dependent on the parity type (even or odd).

# Serial Interface Protocols: UART

## How UART Works?



# Serial Interface Protocols: UART

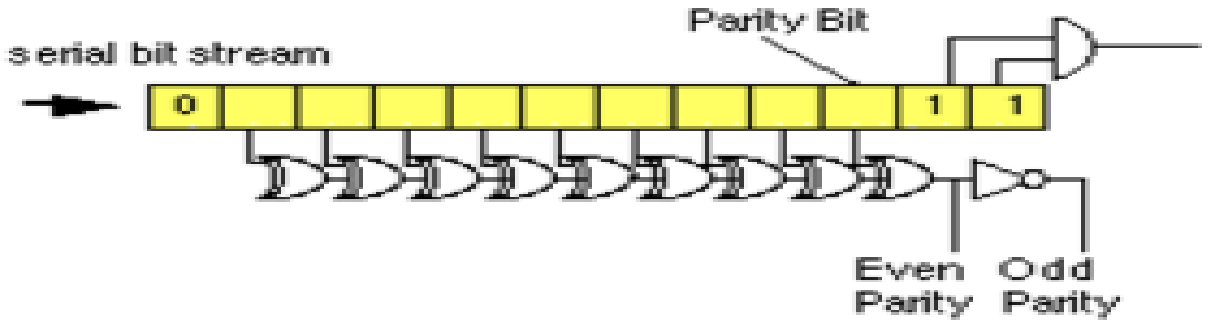
## Common UART Baud rates

| | |
|---|---|
| 2400 | 38.4k |
| 4800 | 57.6k |
| 9600 | 76.8k |
| 14.4k | 115.2k |
| 19.2k | 230.4k |
| 28.8k | 250k |

# Serial Interface Protocols: UART

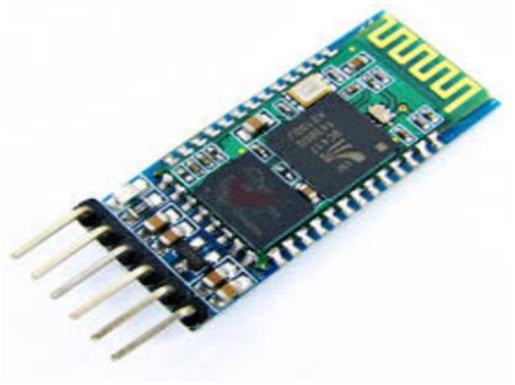
## Special UART receiver conditions

- **Overrun error:**
- occurs when the receiver cannot process the character that just came in before the next one arrives.
- **Framing error:**
- A "framing error" occurs when the designated "start" and "stop" bits are not found.
- **Parity error:**
- A Parity Error occurs when the parity of the number of 1 bits disagrees with that specified by the parity bit. Use of a parity bit is optional, so this error will only occur if parity-checking has been enabled.



# Serial Interface Protocols: UART

## Famous devices that interfaced by UART



# Serial Interface Protocols: UART

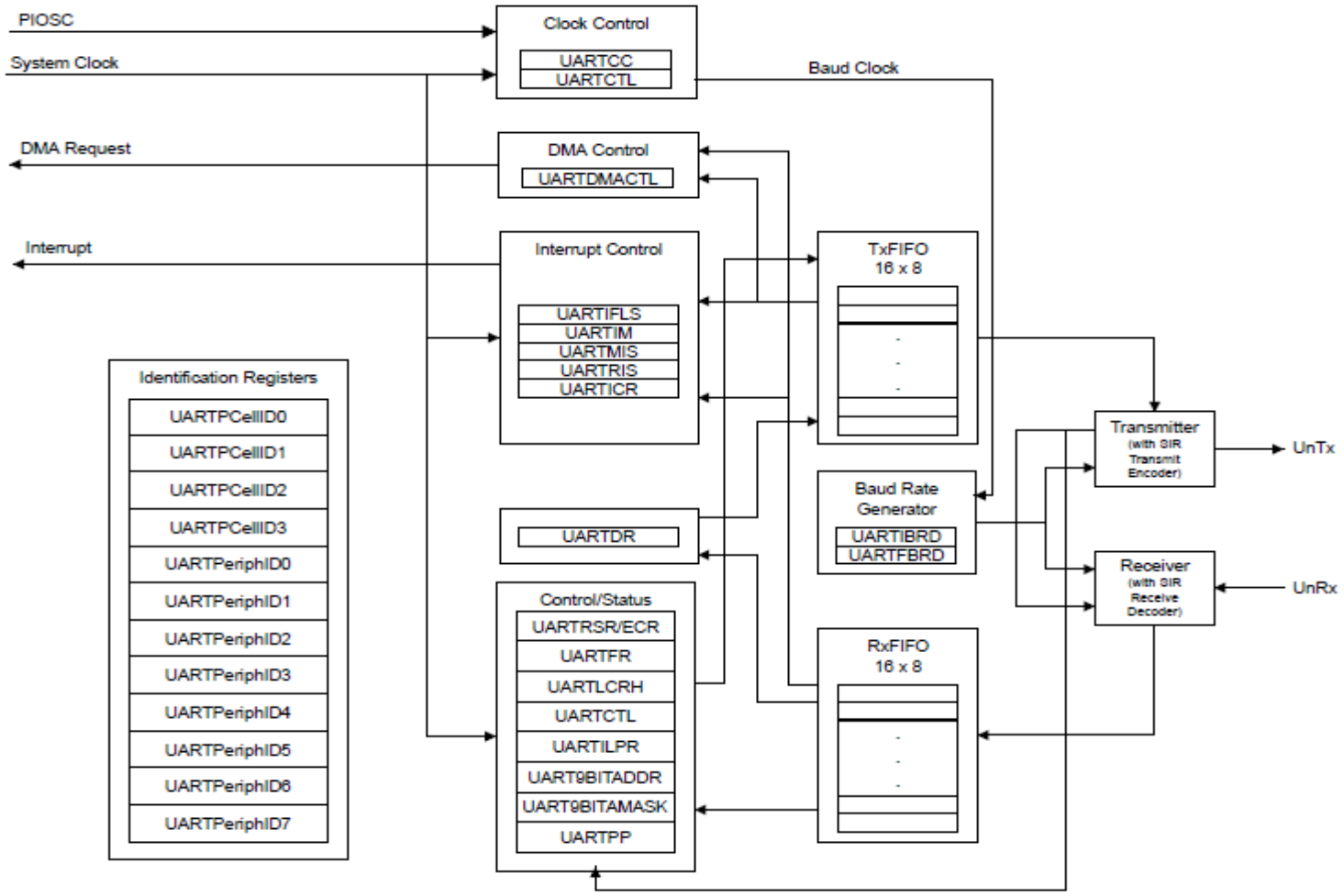
**The TM4C123GH6PM controller includes eight Universal Asynchronous Receiver/Transmitter (UART) with the following features:**

- Programmable baud-rate generator allowing speeds up to 5 Mbps for regular speed (divide by 16) and 10 Mbps for high speed (divide by 8)
- Separate 16x8 transmit (TX) and receive (RX) FIFOs to reduce CPU interrupt service loading
- Programmable FIFO length, including 1-byte deep operation providing conventional double-buffered interface
- FIFO trigger levels of 1/8, 1/4, 1/2, 3/4, and 7/8
- Standard asynchronous communication bits for start, stop, and parity
- Line-break generation and detection
- Fully programmable serial interface characteristics
  - 5, 6, 7, or 8 data bits
  - Even, odd, stick, or no-parity bit generation/detection
  - 1 or 2 stop bit generation
- IrDA serial-IR (SIR) encoder/decoder providing.
- Support for communication with ISO 7816 smart cards
- Modem flow control (on UART1)
- Standard FIFO-level and End-of-Transmission interrupts
- Efficient transfers using Micro Direct Memory Access Controller (µDMA)
- Loop back operation is supported for debugging

# Serial Interface Protocols: UART

## The TM4C123GH6PM UART Block diagram:

**Figure 14-1. UART Module Block Diagram**



# Serial Interface Protocols: UART

## The TM4C123GH6PM UART Signals:

Note that, we must set the GPIO alternative function and the GPIO port control registers to assign the signal to UART.

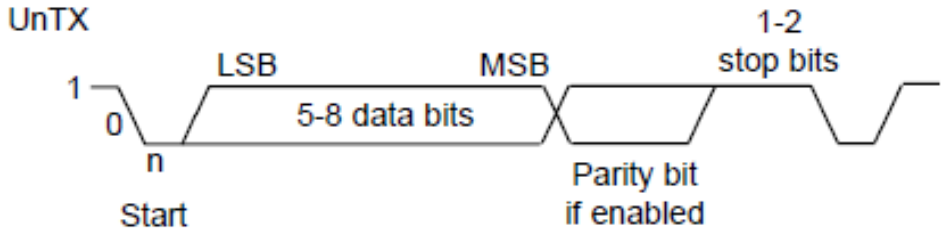
**Table 14-1. UART Signals (64LQFP)**

| Pin Name | Pin Number | Pin Mux / Pin Assignment | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|---|
| U0Rx | 17 | PA0 (1) | I | TTL | UART module 0 receive. |
| U0Tx | 18 | PA1 (1) | O | TTL | UART module 0 transmit. |
| U1CTS | 15<br>29 | PC5 (8)<br>PF1 (1) | I | TTL | UART module 1 Clear To Send modem flow control input signal. |
| U1RTS | 16<br>28 | PC4 (8)<br>PF0 (1) | O | TTL | UART module 1 Request to Send modem flow control output line. |
| U1Rx | 16<br>45 | PC4 (2)<br>PB0 (1) | I | TTL | UART module 1 receive. |
| U1Tx | 15<br>46 | PC5 (2)<br>PB1 (1) | O | TTL | UART module 1 transmit. |
| U2Rx | 53 | PD6 (1) | I | TTL | UART module 2 receive. |
| U2Tx | 10 | PD7 (1) | O | TTL | UART module 2 transmit. |
| U3Rx | 14 | PC6 (1) | I | TTL | UART module 3 receive. |
| U3Tx | 13 | PC7 (1) | O | TTL | UART module 3 transmit. |
| U4Rx | 16 | PC4 (1) | I | TTL | UART module 4 receive. |
| U4Tx | 15 | PC5 (1) | O | TTL | UART module 4 transmit. |
| U5Rx | 59 | PE4 (1) | I | TTL | UART module 5 receive. |
| U5Tx | 60 | PE5 (1) | O | TTL | UART module 5 transmit. |
| U6Rx | 43 | PD4 (1) | I | TTL | UART module 6 receive. |
| U6Tx | 44 | PD5 (1) | O | TTL | UART module 6 transmit. |
| U7Rx | 9 | PE0 (1) | I | TTL | UART module 7 receive. |
| U7Tx | 8 | PE1 (1) | O | TTL | UART module 7 transmit. |

# Serial Interface Protocols: UART

## The TM4C123GH6PM UART Transmit /Receive logic:

**Figure 14-2. UART Character Frame**




- **The transmit logic** performs parallel-to-serial conversion on the data read from the transmit FIFO.
- The control logic outputs the serial bit stream beginning with a start bit and followed by the data bits (LSB first), parity bit, and the stop bits according to the programmed configuration in the control registers.
- **The receive logic** performs serial-to-parallel conversion on the received bit stream after a valid start pulse has been detected.
- Overrun, parity, frame error checking, and line-break detection are also performed, and their status accompanies the data that is written to the receive FIFO.

# Serial Interface Protocols: UART

## The TM4C123GH6PM UART Baud rate generation:

- The baud-rate divisor is a 22-bit number consisting of a 16-bit integer and a 6-bit fractional part.
- The number formed by these two values is used by the baud-rate generator to determine the bit period.
- The 16-bit integer is loaded through the UART Integer Baud-Rate Divisor (UARTIBRD) register and the 6-bit fractional part is loaded with the UART Fractional Baud-Rate Divisor (UARTFBRD) register.
- The baud-rate divisor (BRD) has the following relationship to the system clock

```
BRD = BRDI + BRDF = UARTSysClk / (ClkDiv * Baud Rate)
```

- UARTSysClk is the system clock connected to the UART,
- ClkDiv is either 16 (if HSE in **UARTCTL** is clear) or 8 (if HSE is set).
- The 6-bit fractional number can be calculated by :

```
UARTFBRD[DIVFRAC] = integer(BRDF * 64 + 0.5)
```

- adding 0.5 to account for rounding errors.

# Serial Interface Protocols: UART

**The TM4C123GH6PM UART Data Transmission and Data reception:**

- Data received or transmitted is stored in two 16-byte FIFOs, though the receive FIFO has an extra four bits per character for status information.

- **For transmission,** data is written into the transmit FIFO by writing data on UART Data (UARTDR) register.
- If the UART is enabled, it causes a data frame to start transmitting.
- Data continues to be transmitted until there is no data left in the transmit FIFO.
- The BUSY bit in the UART Flag (UARTFR) register is asserted as soon as data is written to the transmit FIFO.
- It remains asserted while data is being transmitted.

- **For reception,** When the receiver is idle and a start bit has been received, the receive counter begins running and data is sampled depending on the setting of the HSE bit (bit 5) in **UARTCTL.**
- The parity bit is then checked if parity mode is enabled. Data length and parity are defined in the UARTLCRH register.
- When a full word is received, the data is stored in the receive FIFO along with any error bits associated with that word.

# Serial Interface Protocols: UART

## The TM4C123GH6PM UART FIFO Operation:

- The UART has two 16x8 FIFOs; one for transmit and one for receive.
- Both FIFOs are accessed via the UART Data (UARTDR).
- Out of reset, both FIFOs are disabled and act as 1-byte-deep holding registers.
- The FIFOs are enabled by setting the FEN bit in **UARTLCRH.**
- FIFO status can be monitored via the UART Flag (UARTFR) register and the UART Receive Status (UARTRSR) register.
- Hardware monitors empty, full and overrun conditions.(TXFE, TXFF, RXFE, RXFF and OE bits).
- The trigger points at which the FIFOs generate interrupts is controlled via the **UART Interrupt FIFO Level Select (UARTIFLS)** register.
- Both FIFOs can be individually configured to trigger interrupts at different levels. Available configurations include ⅛, ¼, ½, ¾, and ⅞.

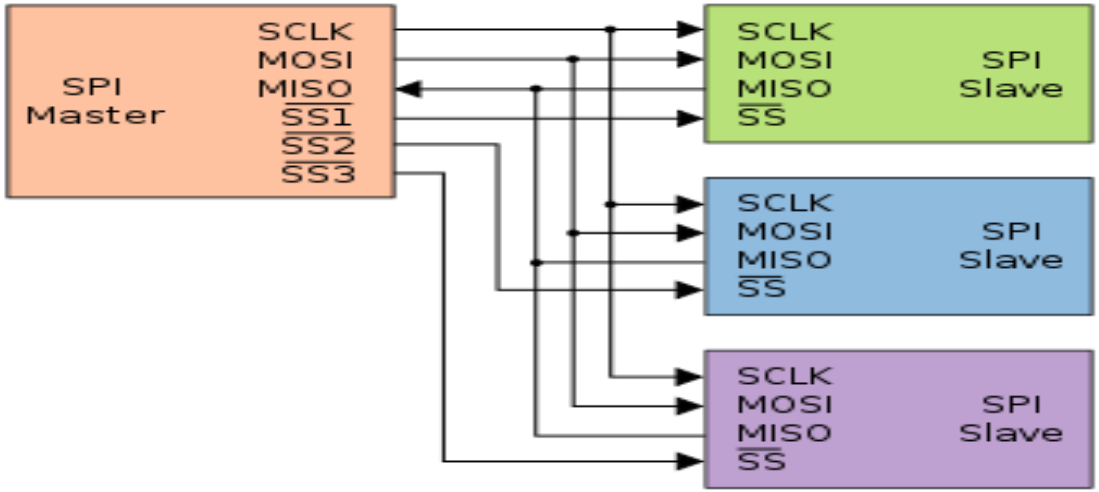
# Serial Interface Protocols: UART

## The TM4C123GH6PM UART Interrupts:

- The UART can generate interrupts when the following conditions are observed: Overrun error, break error, parity error, frame error, receive timeout, transmission condition met, reception condition met.
- All of the interrupt events are ORed together before being sent to the interrupt controller, so the UART can only generate a single interrupt request to the controller at any given time.(That means all sources UART interrupt will be handled by a single ISR).
- This could be done using by reading the UART Masked Interrupt Status (UARTMIS) register.
- The interrupt events that can trigger a controller-level interrupt are defined in the UART Interrupt Mask (UARTIM) register.
- If interrupts are not used, the raw interrupt status is visible via the UART Raw Interrupt Status (UARTRIS) register.

# Serial Interface Protocols: SPI

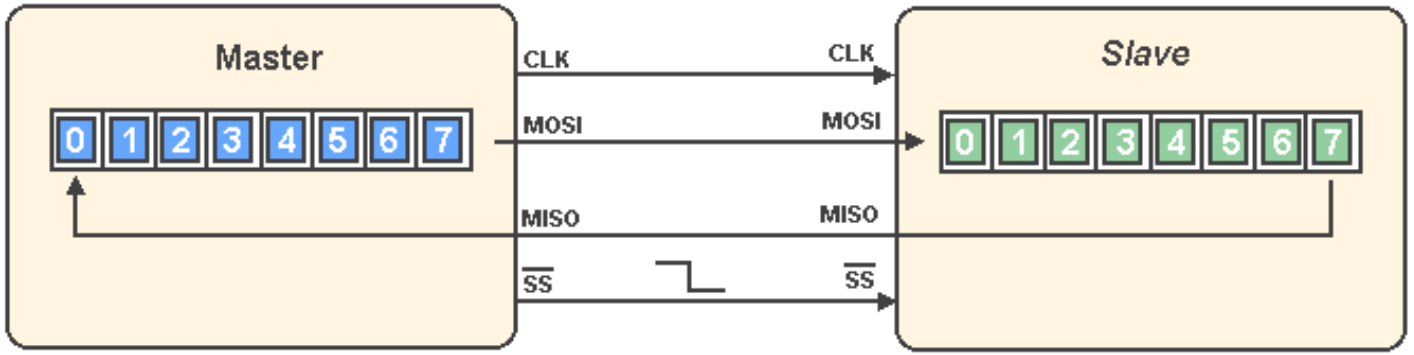
## What is SPI?




- The Serial Peripheral Interface (SPI) bus is asynchronous serial communication interface specification used for short distance communication, primarily in embedded systems.
- The interface was developed by Motorola and has become a de factostandard.
- SPI devices communicate in full duplex mode using a master-slave architecture with a single master.
- he master device originates the frame for reading and writing.
- Multiple slave devices are supported through selection with individualslave select (SS) lines.

# Serial Interface Protocols: SPI

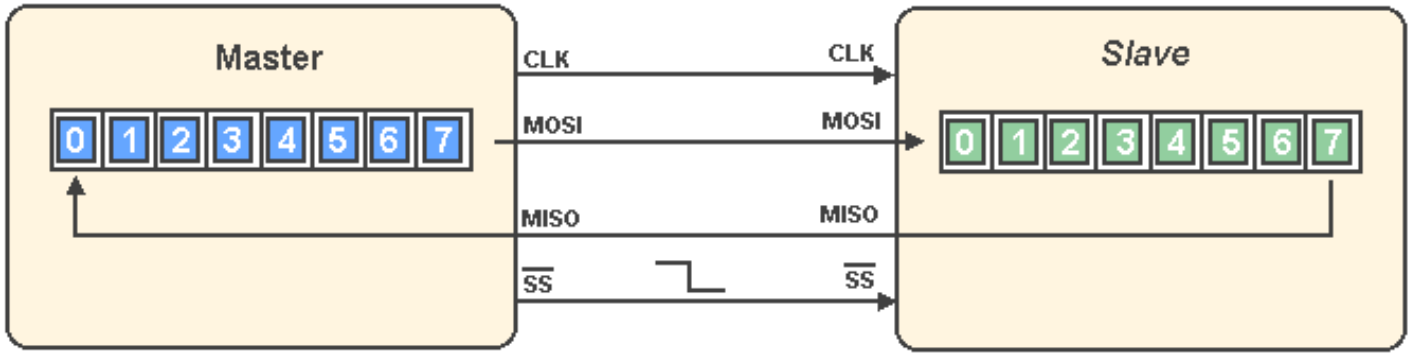
## Interface and operation




- The SPI bus specifies four logic signals:
- **SCLK(SCK,CLK)** : Serial Clock (output from master).
- **MOSI(SIMO,SDI,DI)** : Master Output, Slave Input (output from master).
- **MISO(SIMO,SDO,DO)** : Master Input, Slave Output (output from slave).
- **SS** (CS,CE,CEN): Slave Select (active low, output from master).

# Serial Interface Protocols: SPI

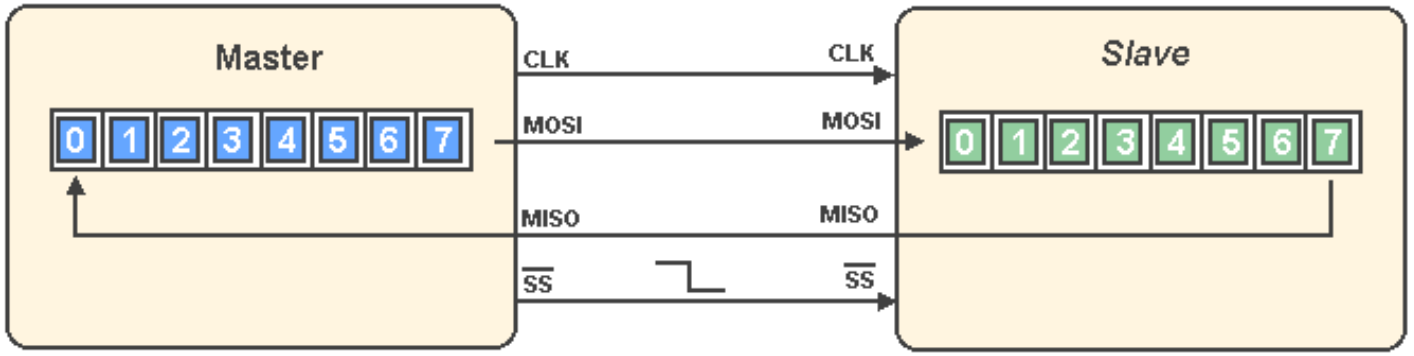
## Interface and operation




- To begin communication, the bus master configures the clock, using a frequency supported by the slave device, typically up to a few MHz.
- The master then selects the slave device with a logic level 0(If it was active low) on the select line.
- the master must wait for at least that period of time before issuing clock cycles.
- During each SPI clock cycle, The master sends a bit on the MOSI line and the slave reads it, while the slave sends a bit on the MISO line and the master reads it. This sequence is maintained even when only one-directional data transfer is intended.

# Serial Interface Protocols: SPI

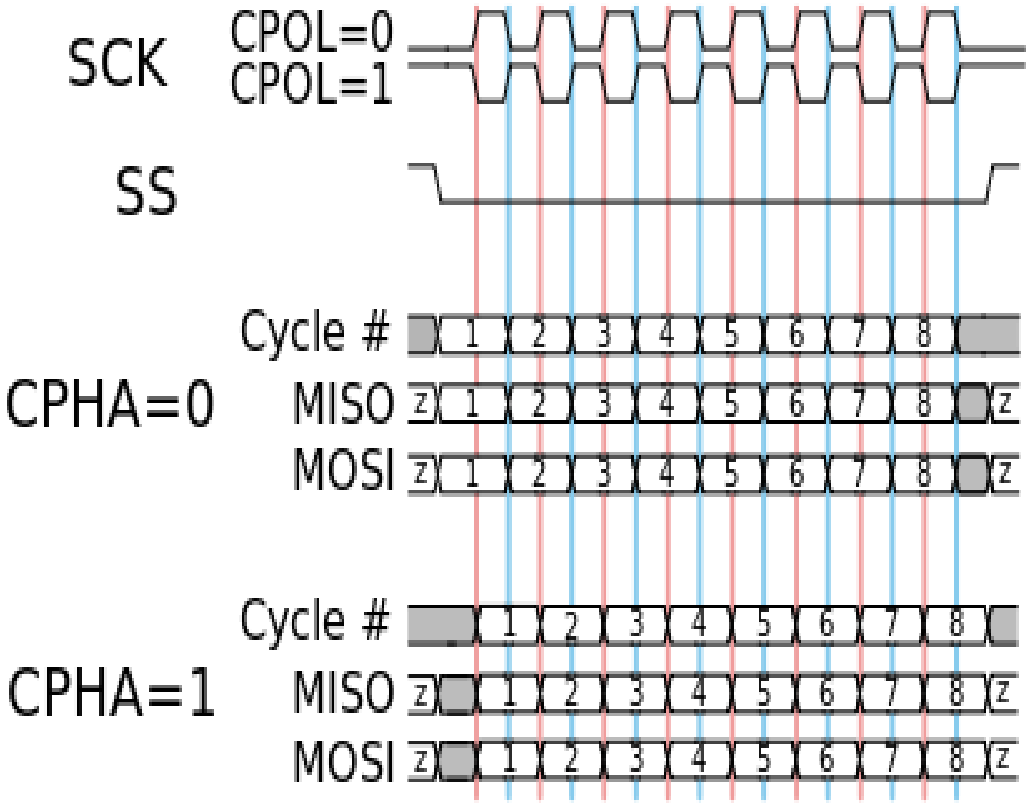
## Interface and operation




- Transmissions normally involve two shift registers of some given word size, such as eight bits, one in the master and one in the slave; they are connected in a virtual ring topology.
- After the register bits have been shifted out and in, the master and slave have exchanged register values.
- If more data needs to be exchanged, the shift registers are reloaded and the process repeats.

# Serial Interface Protocols: SPI

## Clock phase and clock polarity (SPI modes)

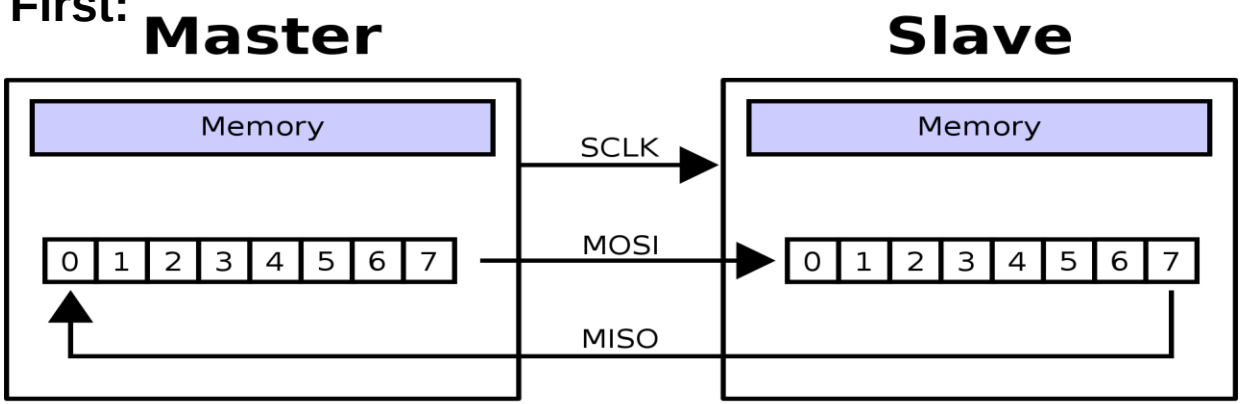



| Mode | CPOL | CPHA |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 2 | 1 | 0 |
| 3 | 1 | 1 |

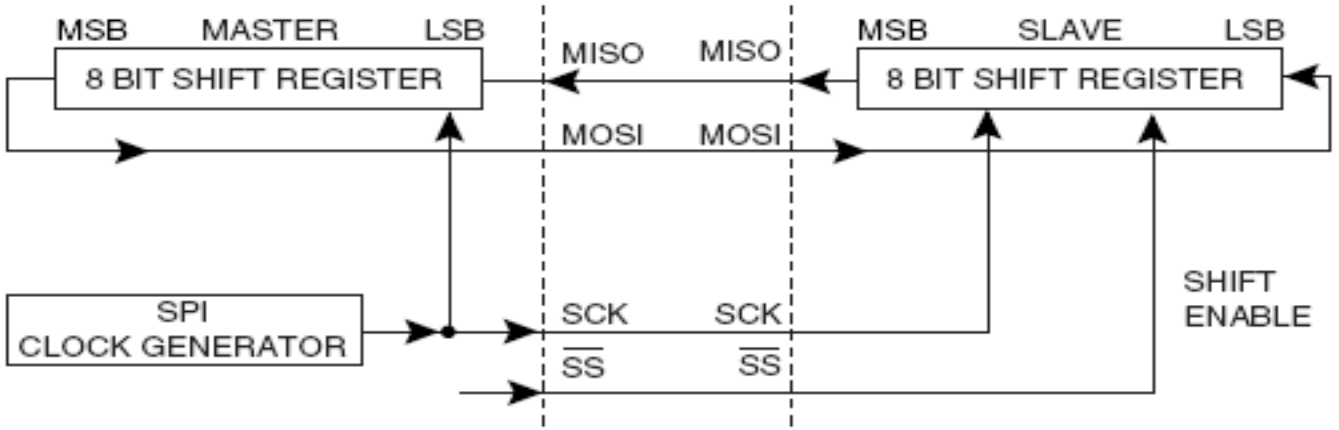
# Serial Interface Protocols: SPI

## Data Order

- **LSB First:**

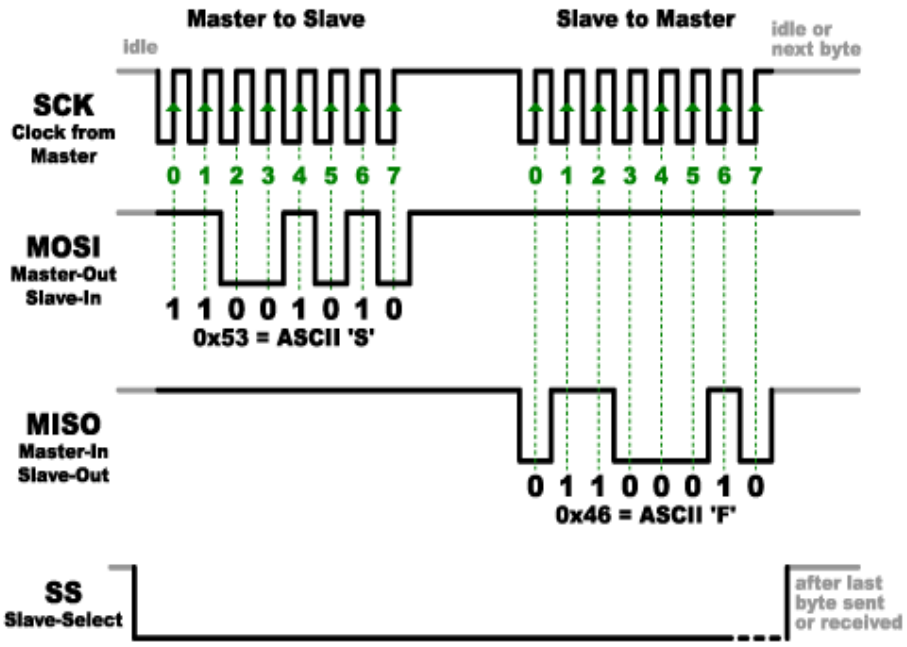


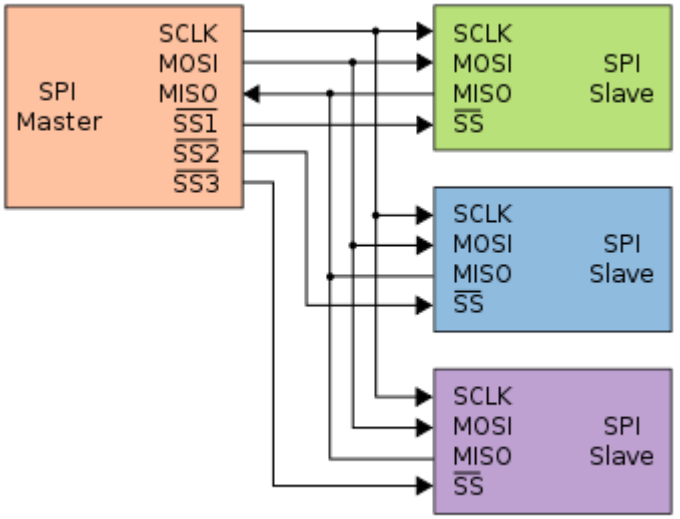

- **MSB First:**



# Serial Interface Protocols: SPI

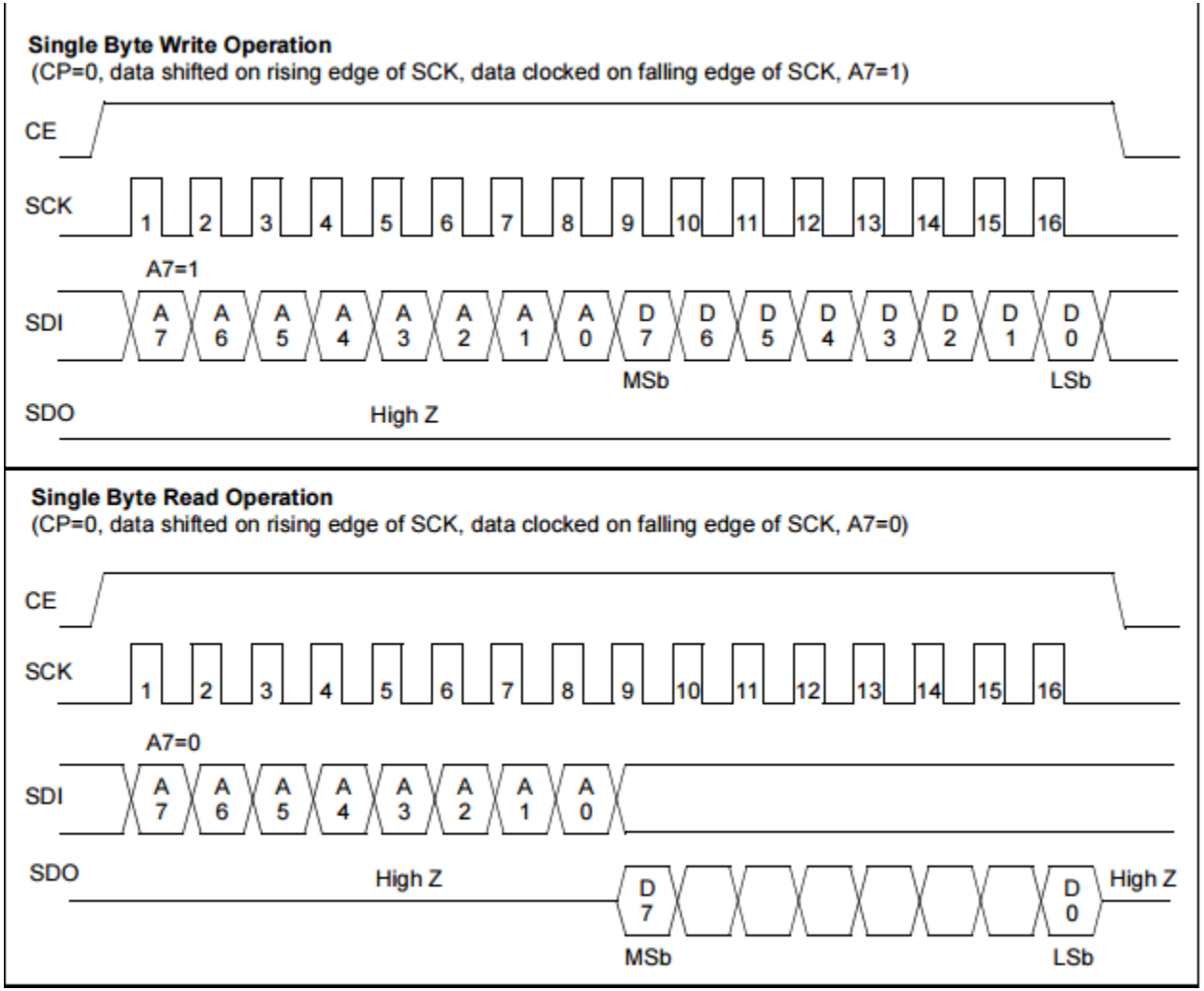
**Typical Device interface: (Independent slave configuration)**

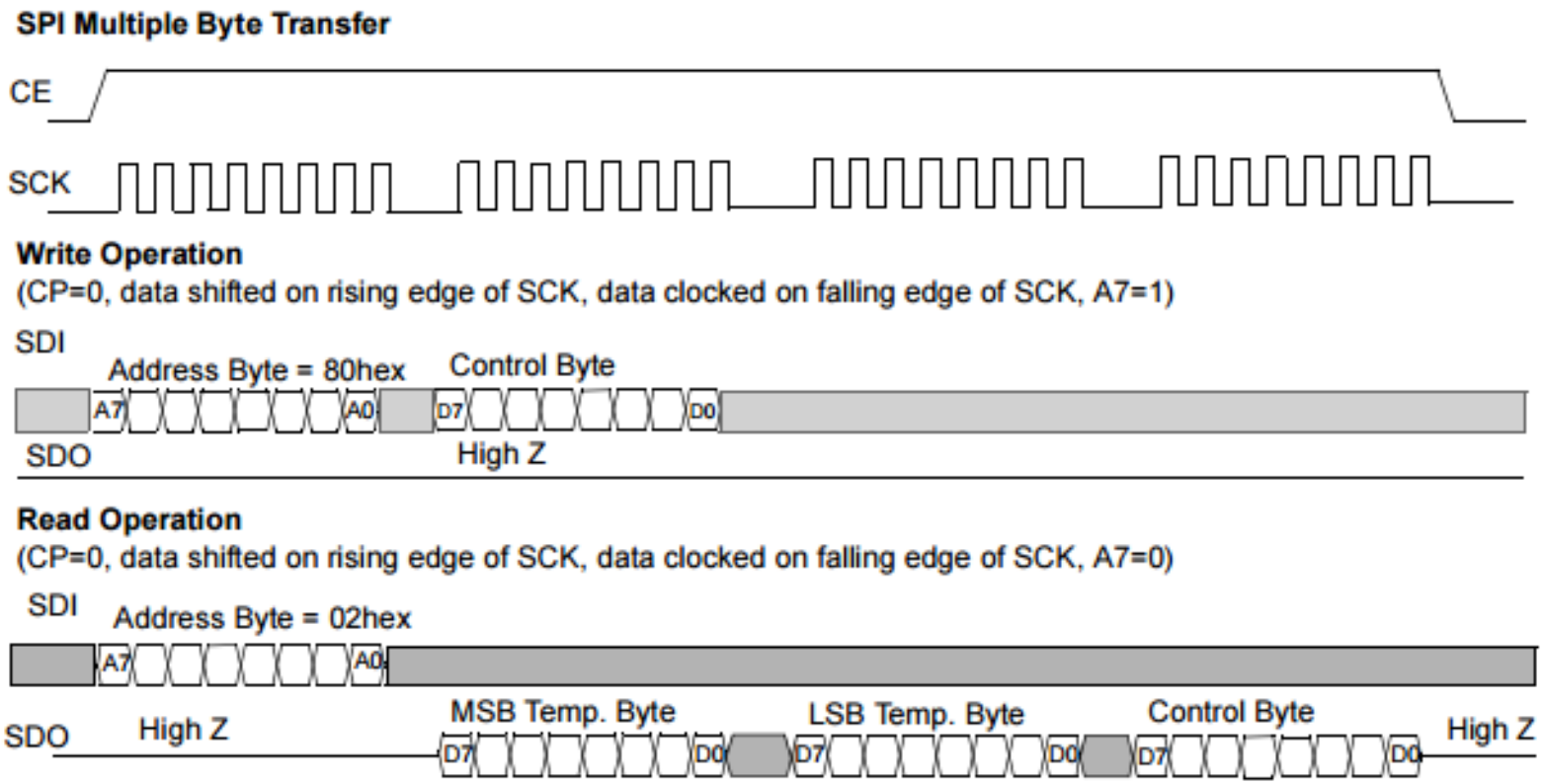
# Serial Interface Protocols: SPI

**Typical Device interface: (Independent slave configuration)**



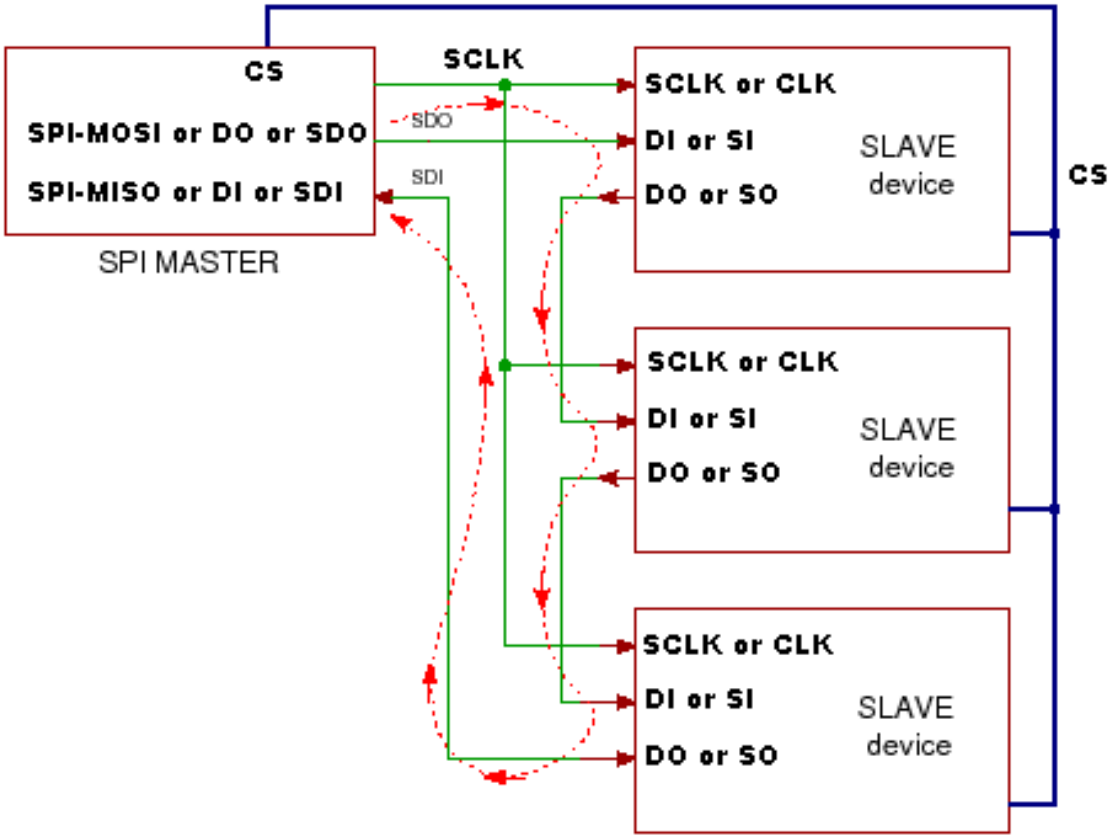
# Serial Interface Protocols: SPI

**Typical Device interface: (Independent slave configuration)**



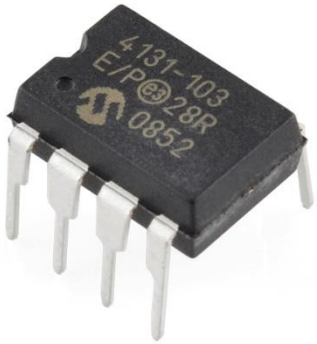
# Serial Interface Protocols: SPI

**Typical Device interface: (Daisy chain configuration)**

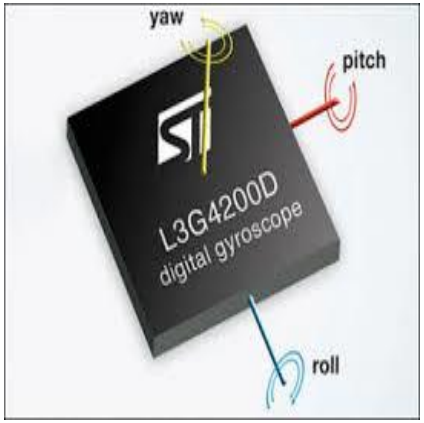


# Serial Interface Protocols: SPI

## Famous devices that use SPI interface

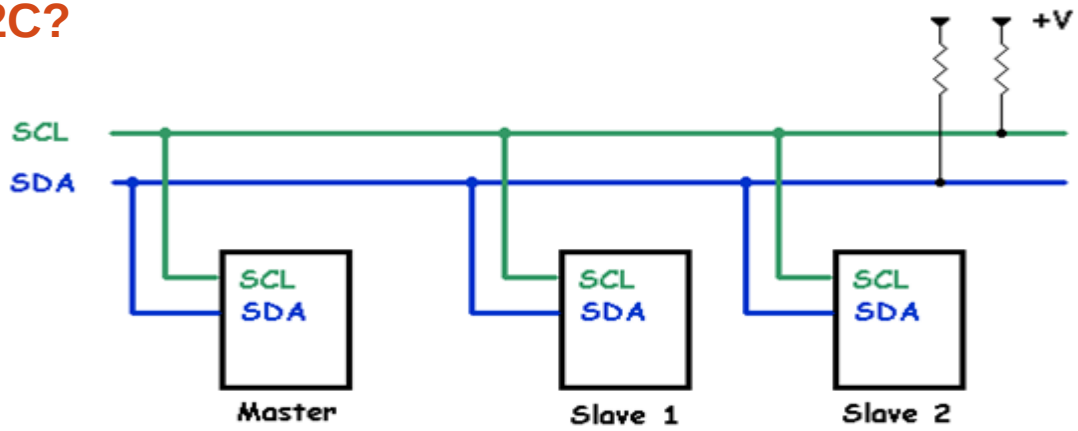






# Serial Interface Protocols: I2C

## What is I2C?



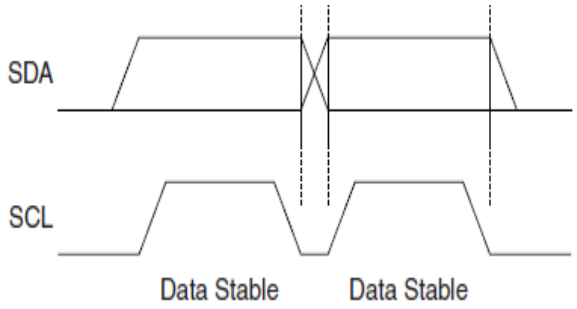

- I²C (Inter-Integrated Circuit), is a multi-master, multi-slave, single-ended, serial computer bus invented by Philips Semiconductor (now NXP Semiconductors).
- It is typically used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication.
- Since October 10, 2006, no licensing fees are required to implement the I²C protocol. However, fees are still required to obtain I²C slave addresses allocated by NXP
- I2C devices are connected through only two wires: SDA for Data and SCL for clock.
- Both signals are externally pulled up.

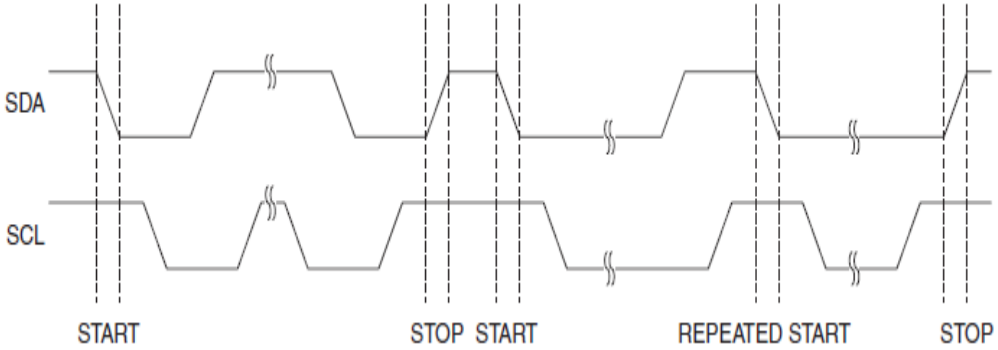
# Serial Interface Protocols: I2C

## I2C Terminologies:

| Term | Description |
|---|---|
| Master | The device that initiates and terminates a transmission. The master also generates the SCL clock. |
| Slave | The device addressed by a master. |
| Transmitter | The device placing data on the bus. |
| Receiver | The device reading data from the bus. |

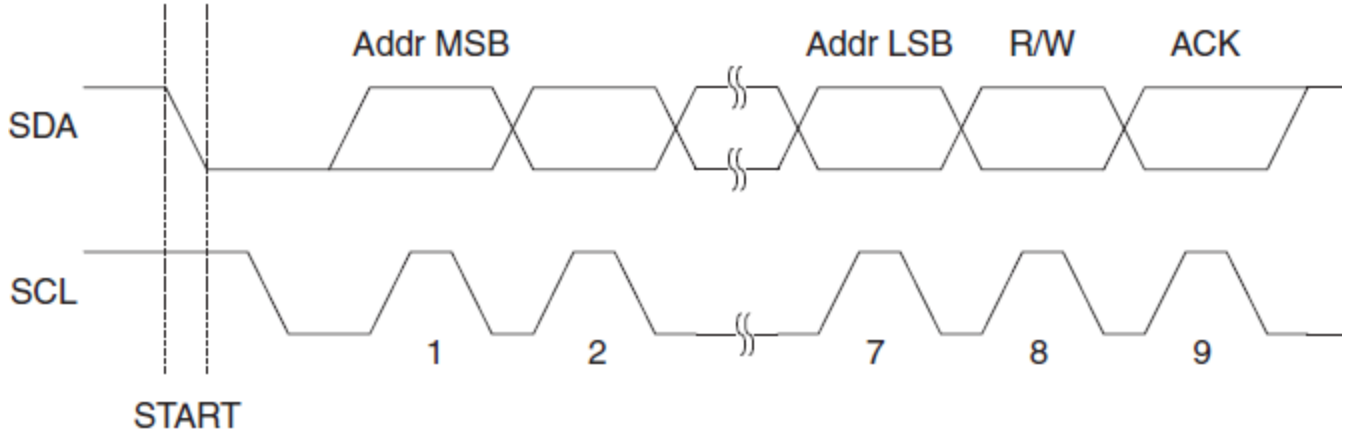
## I2C Data validity, Start and Stop condition



# Serial Interface Protocols: I2C

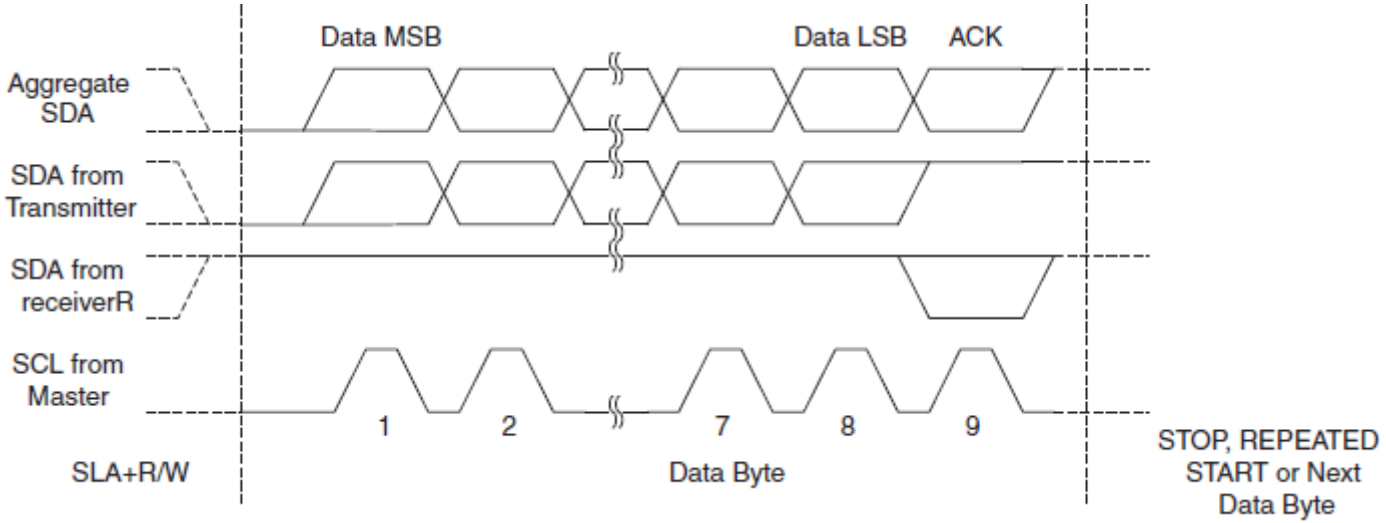
## I2C Address Packet format:




- All address packets transmitted on the TWI bus are nine bits long, consisting of seven address bits, one READ/WRITE control bit and an acknowledge bit.
- If the READ/WRITE bit is set, a read operation is to be performed, otherwise a write operation should be performed.
- When a slave recognizes that it is being addressed, it should acknowledge by pulling SDA low in the ninth SCL (ACK) cycle.
- the address 0000 000 is reserved for a general call.

# Serial Interface Protocols: I2C

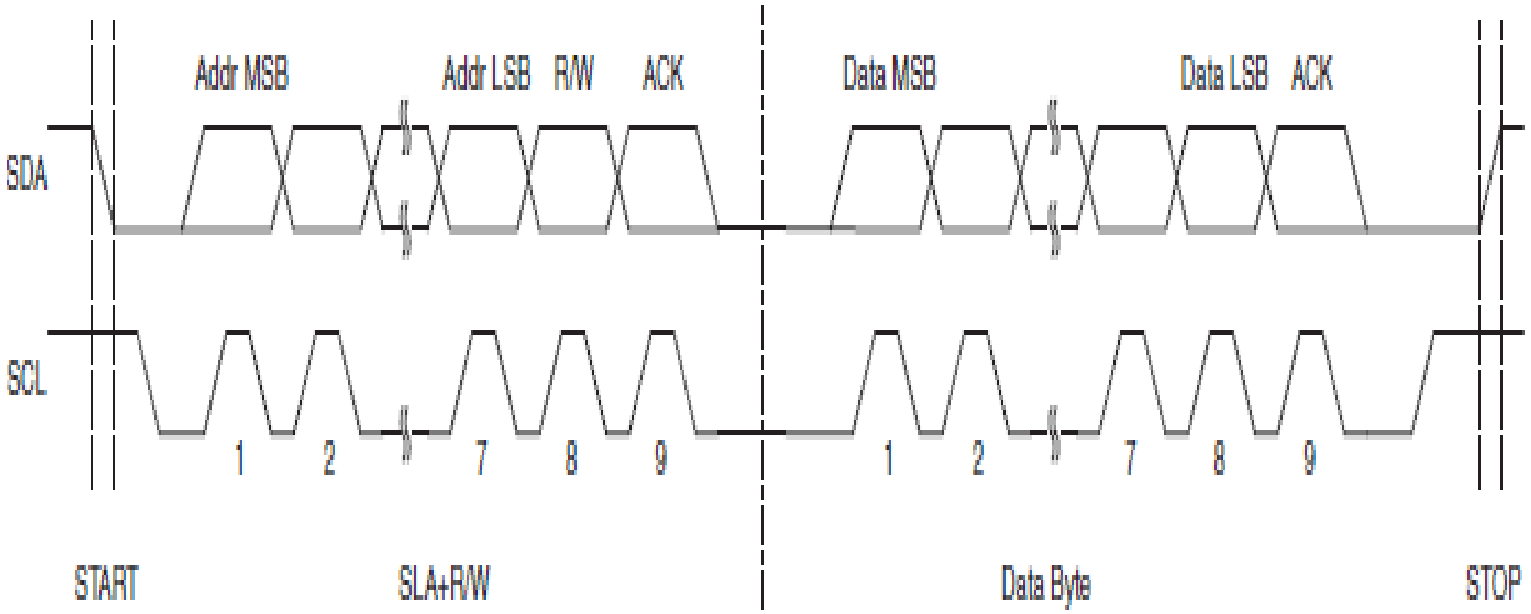
## I2C Data Packet format:




- All data packets transmitted on the TWI bus are nine bits long, consisting of one data byte and an acknowledge bit.
- The MSB of the data byte is transmitted first.

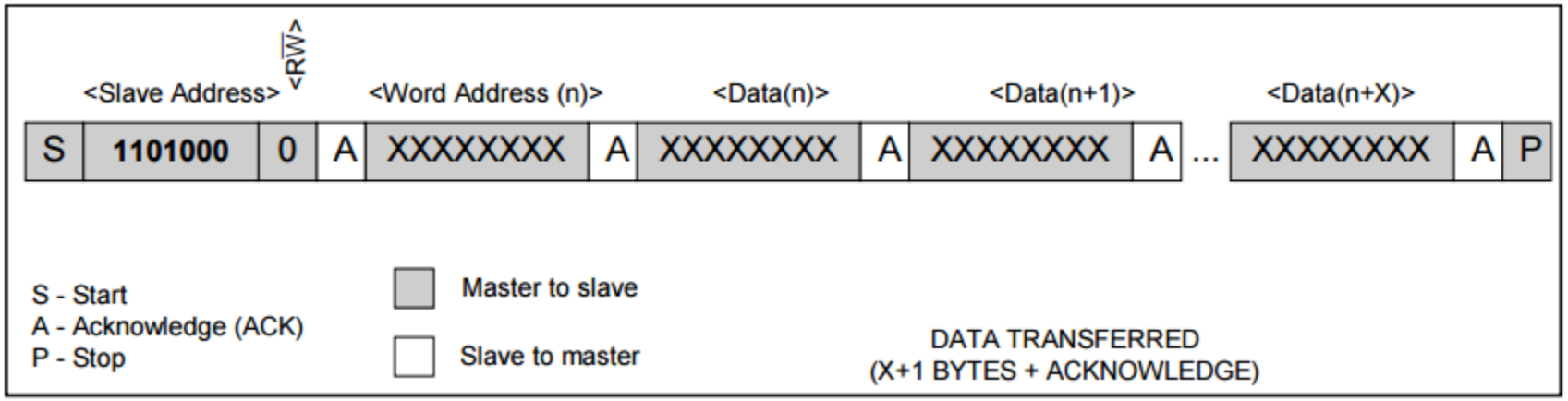
# Serial Interface Protocols: I2C

**Typical Data transmission:**

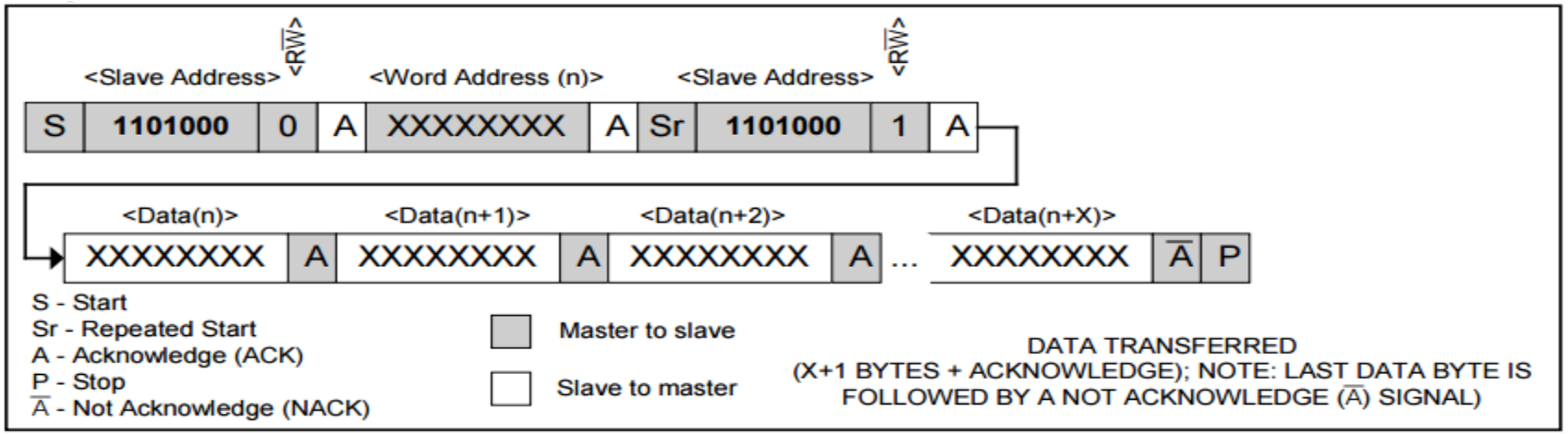


# Serial Interface Protocols: I2C

## Typical Device interface (data write):




## Typical Device interface (data read):

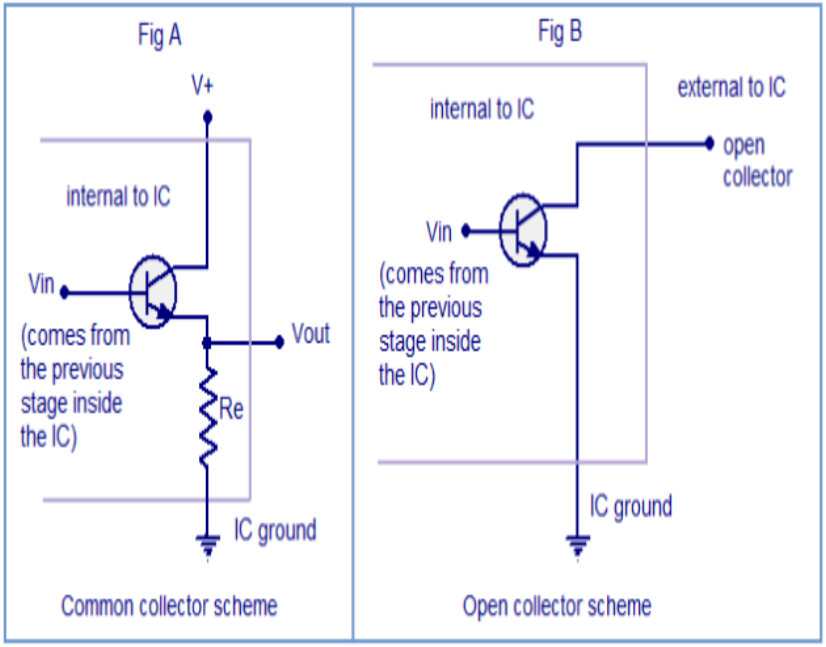


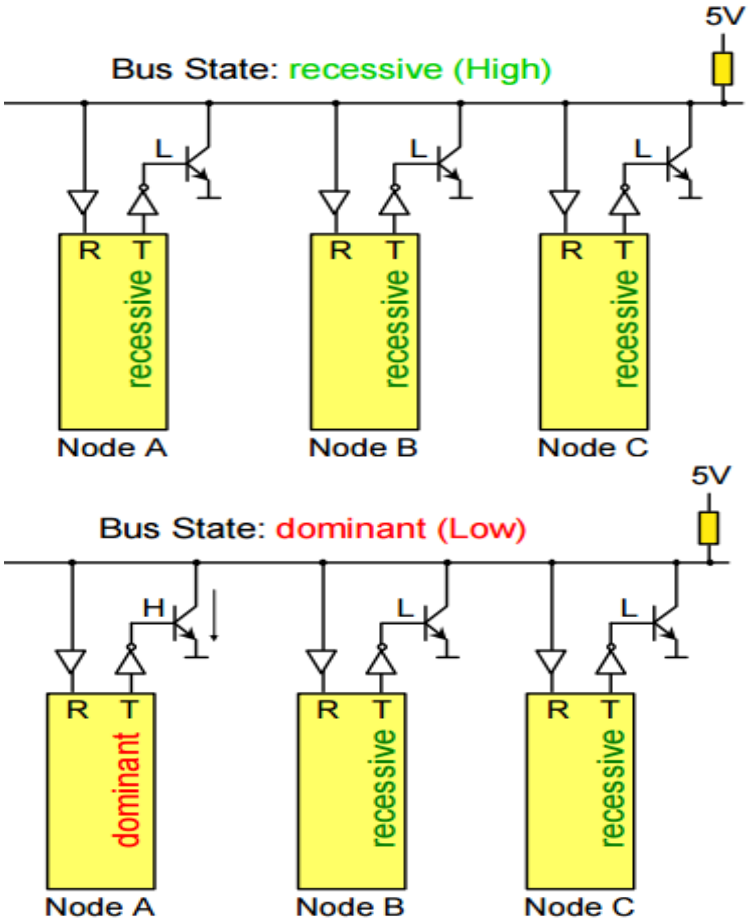
# Serial Interface Protocols: I2C

## I2C Bus arbitration and multimaster support:

**Concept of open drain connection:**

**Typically all signals on bus are ANDed together**

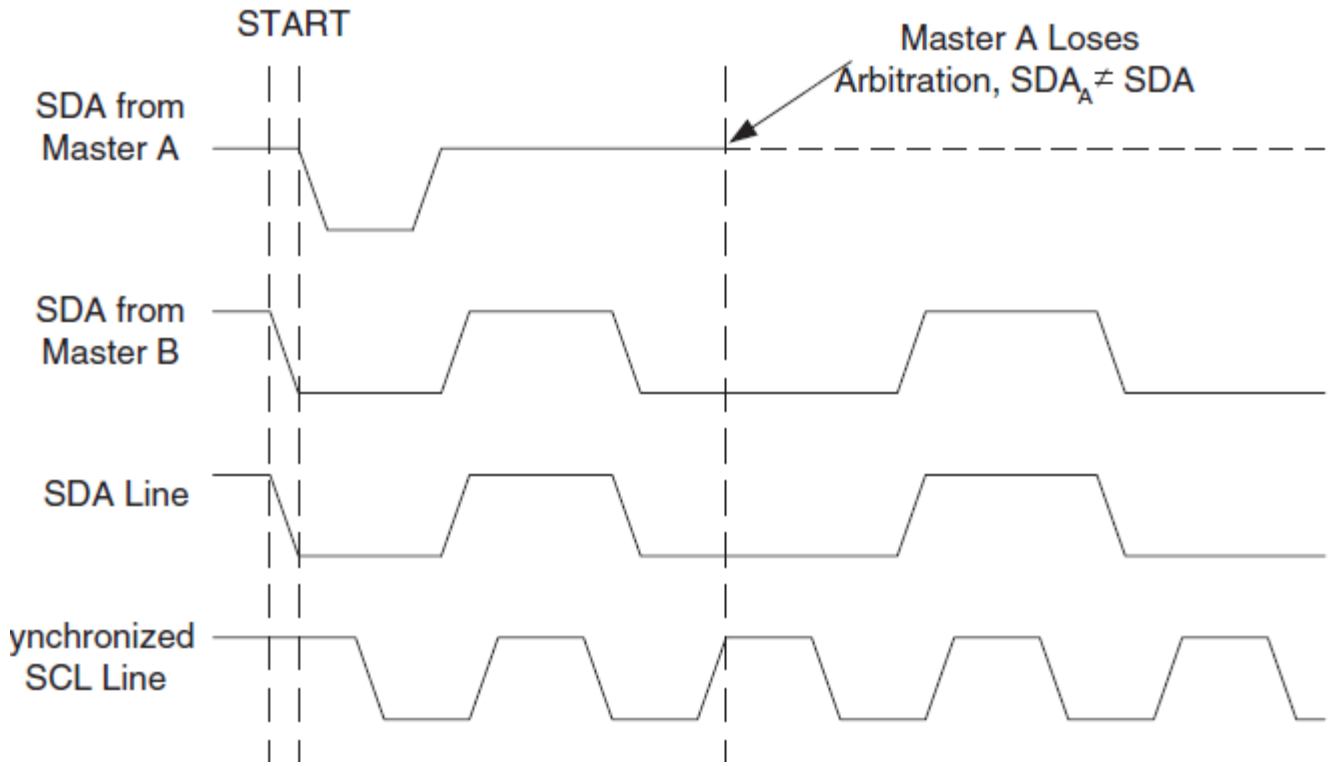


# Serial Interface Protocols: I2C

## I2C Bus arbitration and multimaster support:

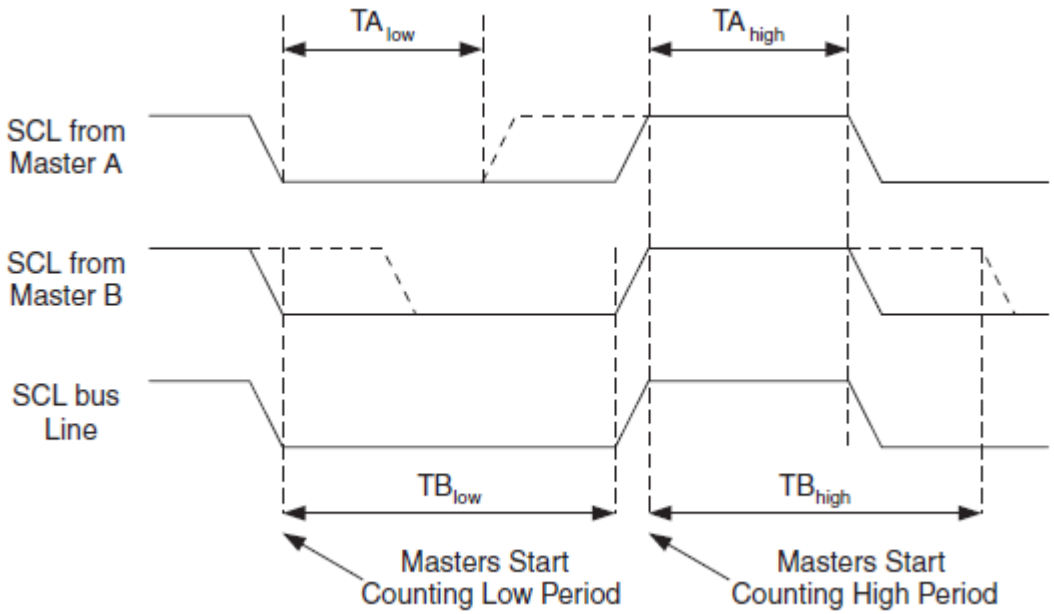
**Arbitration between two masters on the SDA line**



# Serial Interface Protocols: I2C

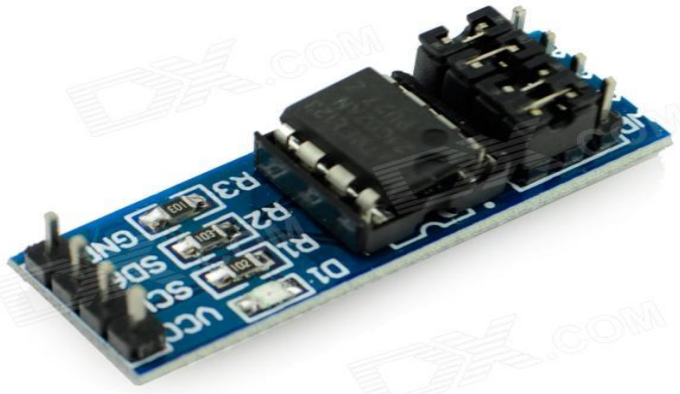
## I2C Bus arbitration and multimaster support:

**Arbitration between two masters on the SCL line**



# Serial Interface Protocols: I2C

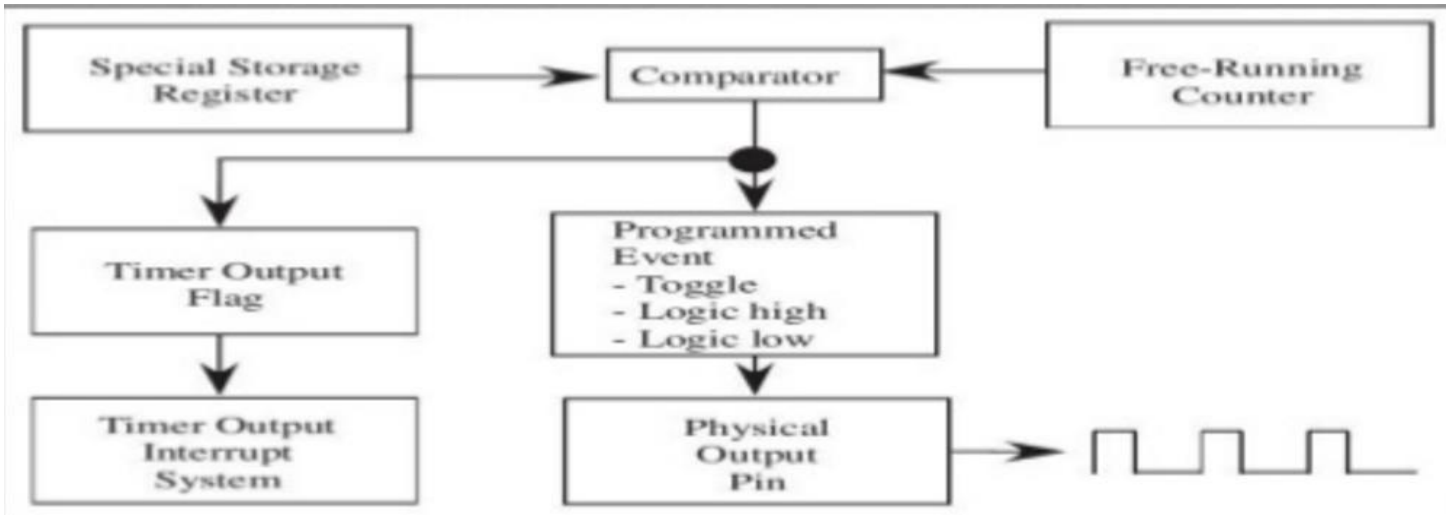
**Famous devices that use I2C interface**

# 8. General Purpose Timer and Pulse Width Modulation

# Introduction to Micrcontroller Free running timer

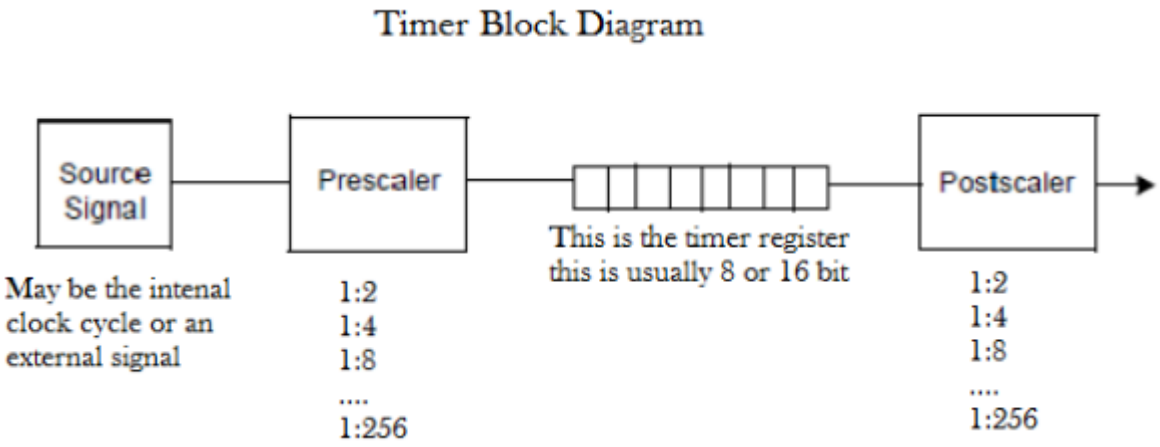
## What is a microcontroller timer?




- Timer is just a register that updates its value every trigger by increment or decrement.
- There are at least three data registers in any timer, one to hold the current Timer value, one to hold the max number of counts to be reached and the third is to hold a prescaler division factor.
- Any microcontroller will provide a group of modes to support the timer to perform the time related functionality.
- Any timer is used to support, Timeout events, external events counting and generation of time controlled signals (PWM).

# Introduction to Micrcontroller Free running timer

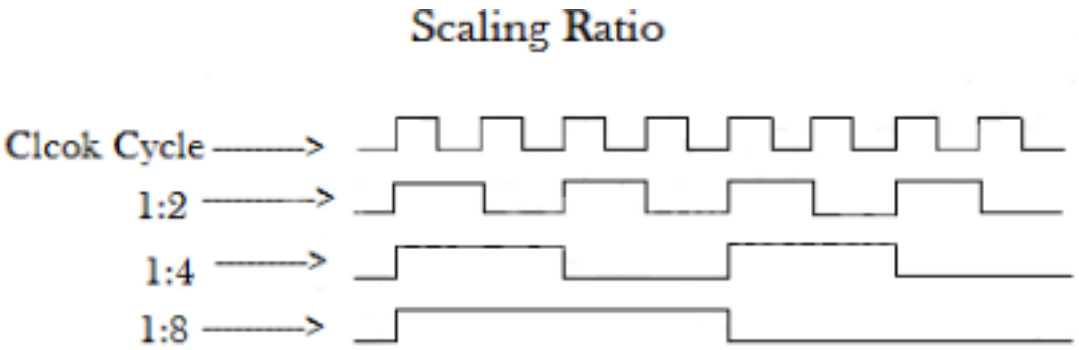
## Prescaler and Postscaler




The time consumed in one time iteration is calculated from the form:
Time per iteration = Number of counts per iteration X Timer per one count.
Number of counts per iteration is limited by the size of the free running timer register.
Time per one count is limited by the oscillator frequency.
To increase the overall iteration time we can increase the time per one count without changing the oscillator frequency.
This could be done using prescaler and post scaler.

# Introduction to Micrcontroller Free running timer

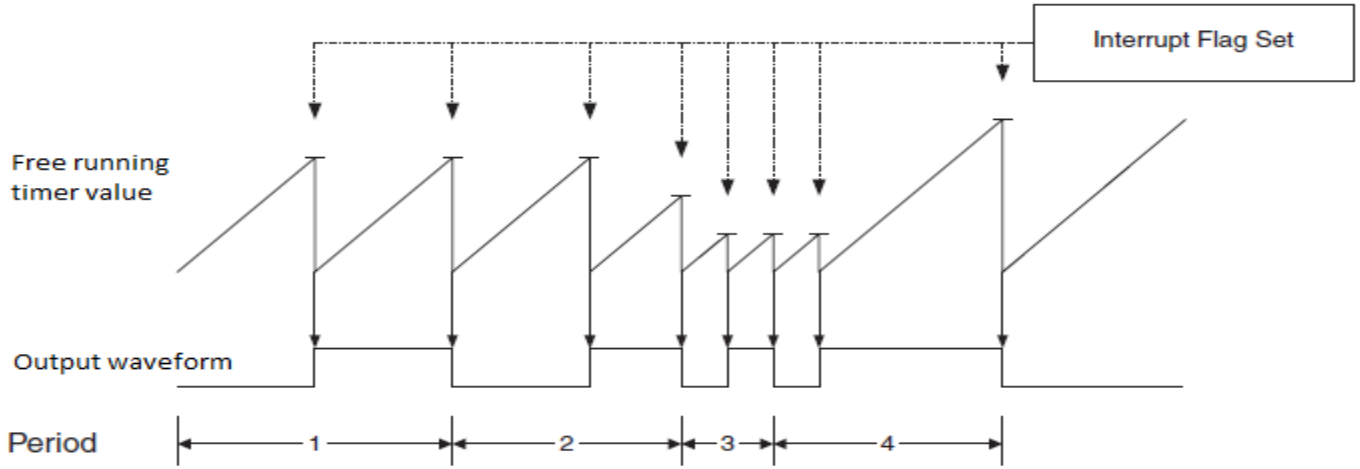
## Prescaler and Postscaler




- The the value in the prescaler register defines the number of times the clock has to tick before the timer register experiences a single pulse.
- The scaling ratio 1:2 means that the clock has to have a LOW to HIGH transition twice so that the output of the prescaler completes one HIGH pulse.
- The postscaler concept is much similar to the prescaler. Only here, the value of the prescaler determines the number of times the timer register has to overflow to produce an interrupt.

# Introduction to Micrcontroller Free running timer

## Timer applications: Timeout events



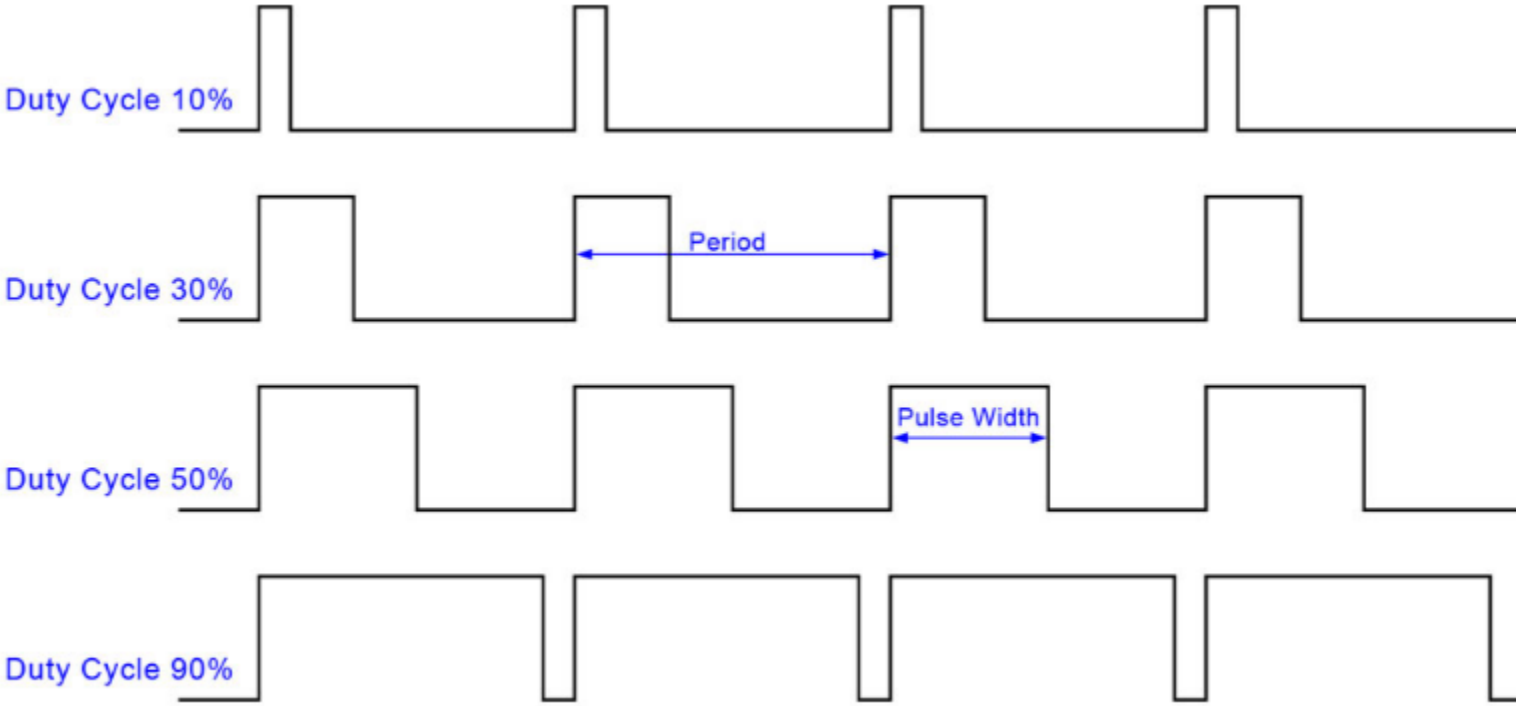

- Timer can be configured to reset after specific value saved in its compare register.
- The overflow time is calculated from the form:

  **Tov = (N / Fosc) X (Number of counts)**
- N is the prescaler division factor.
- Fosc is the input oscillator frequency
- Number of counts is dependent on the value saved in the compare register.
- The overflow events can be used to generate interrupts on a specific times.

# Introduction to Micrcontroller Free running timer

## Timer applications: Pulse Width Modulation

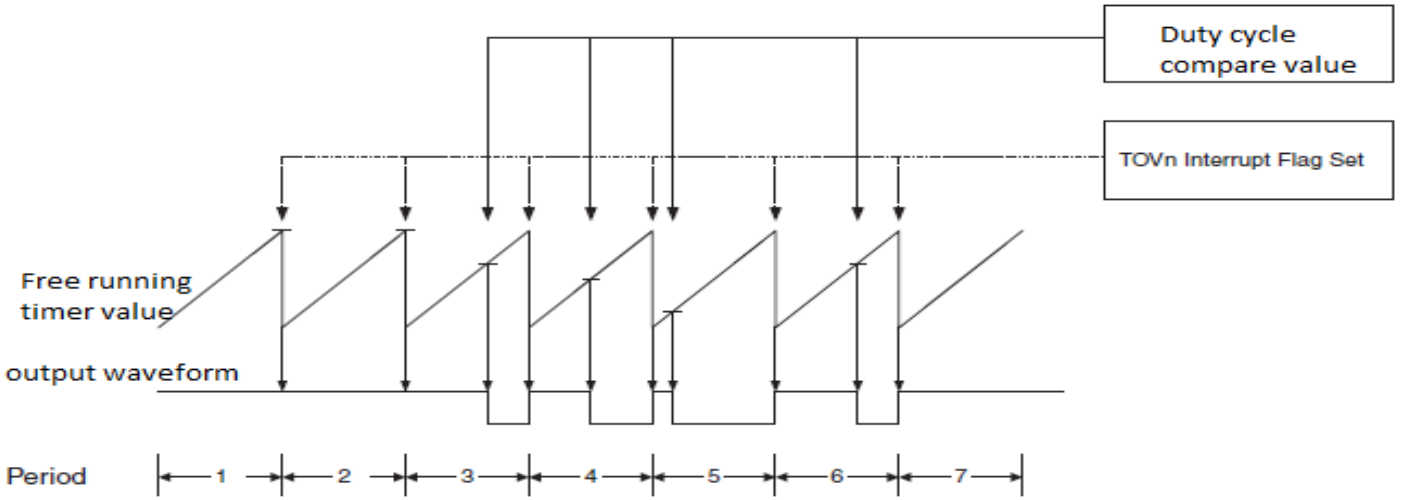
**Basic Terminologies: Period Vs Duty cycle**




Duty Cycle = Pulse Width x 100 / Period

# Introduction to Micrcontroller Free running timer

## Timer applications: Pulse Width Modulation

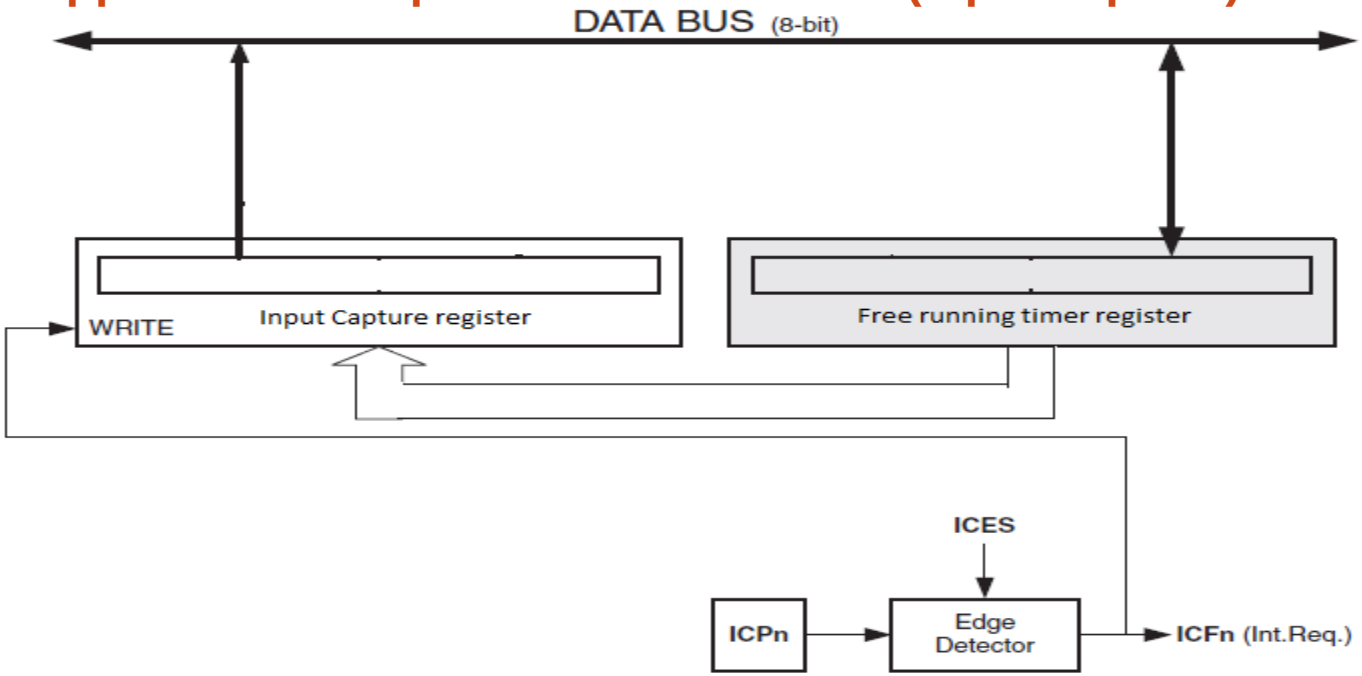



- Iteration number of counts is affecting the timer period.
- Number of counts until toggle is affecting the duty cycle.
- Duty cycle is calculated by:

**Duty cycle = (number of counts till toggle / total number of counts) X 100 %**

# Introduction to Micrcontroller Free running timer

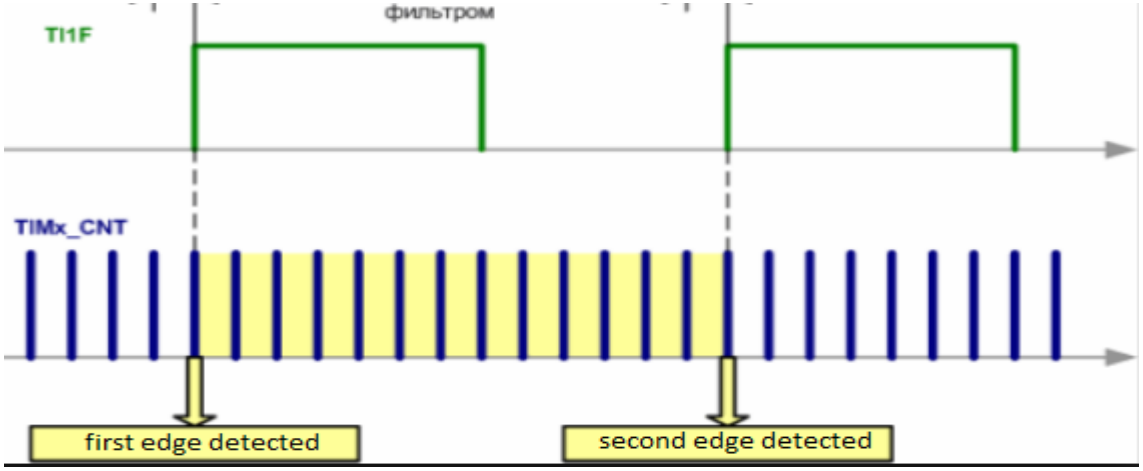
## Timer applications: Capture external events (Input capture)




- Embedded systems using input capture will record a timestamp in memory when an input signal is received.
- It will also set a flag indicating that an input has been captured.
- When a specific edge is detected, the value of the free running timer register is recorded to an additional input capture register.

# Introduction to Micrcontroller Free running timer

## Timer applications: Capture external events (Input capture)




- Input capture could be used to measure the periodicity and the frequency of external input signals.
- The measured time is calculated using the formula:
- **If Second edge counts > first edge counts**
- **Tperiod = (Second edge counts – first edge counts) X (Fosc / N) X (number of timer overflows – 1).**
- **If Second edge counts < first edge counts**
- **Tperiod = (Overflow counts – (first edge counts – second edge counts) X (Fosc / N) X (number of timer overflows – 1).**

# Introduction to Micrcontroller Free running timer

**Timer applications: Typical Application examples:**

**- Motor speed control (DC, AC, Stepper and Servo)**

# Introduction to Micrcontroller Free running timer

**Timer applications: Typical Application examples:**

**- Step mode power supply control and function generators**

# Introduction to Micrcontroller Free running timer

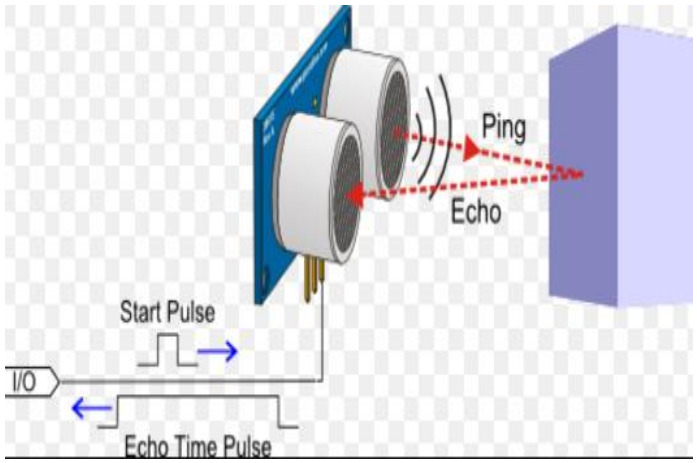
**Timer applications: Typical Application examples:**

**- Light dimming control:**

# Introduction to Micrcontroller Free running timer

**Timer applications: Typical Application examples:**

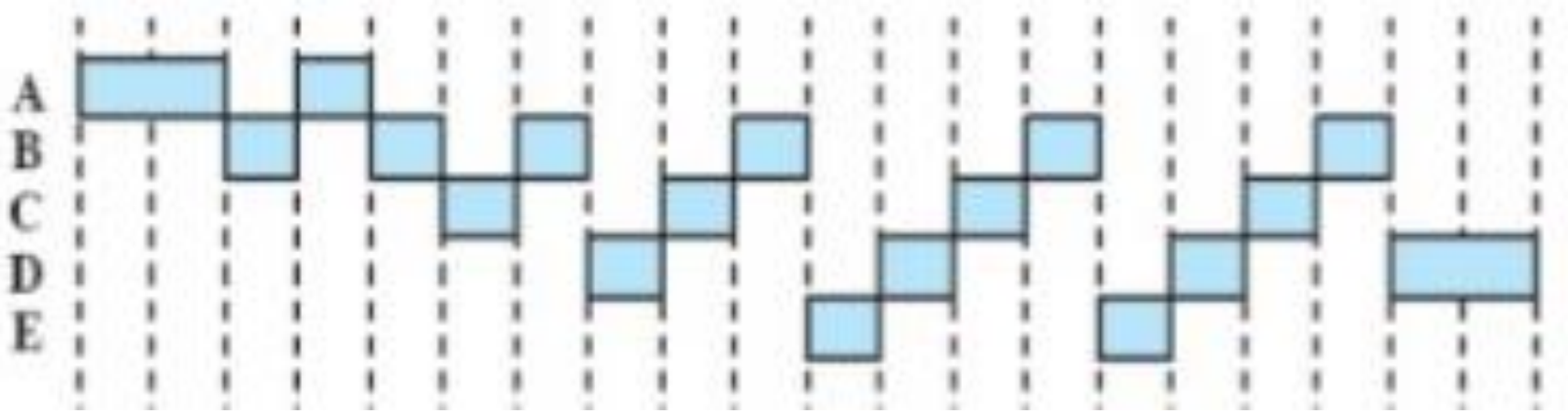
- **Distance measurement**




# Introduction to Micrcontroller Free running timer

**Timer applications: Typical Application examples:**

- **Operating System Scheduler**



# TM4C123GH6PM GPT

**The General-Purpose Timer Module (GPTM) contains six 16/32-bit GPTM blocks and six 32/64-bit Wide GPTM blocks with the following functional options:**

**16/32-bit operating modes:**

- 16- or 32-bit programmable one-shot timer
- 16- or 32-bit programmable periodic timer
- 16-bit general-purpose timer with an 8-bit prescaler
- 32-bit Real-Time Clock (RTC) when using an external 32.768-KHz clock as the input
- 16-bit input-edge count- or time-capture modes with an 8-bit prescaler
- 16-bit PWM mode with an 8-bit prescaler and software-programmable output inversion of the
- PWM signal

# TM4C123GH6PM GPT

**The General-Purpose Timer Module (GPTM) contains six 16/32-bit GPTM blocks and six 32/64-bit Wide GPTM blocks with the following functional options:**

**32/64-bit operating modes:**

- 32- or 64-bit programmable one-shot timer
- 32- or 64-bit programmable periodic timer
- 32-bit general-purpose timer with a 16-bit prescaler
- 64-bit Real-Time Clock (RTC) when using an external 32.768-KHz clock as the input
- 32-bit input-edge count- or time-capture modes with a16-bit prescaler
- 32-bit PWM mode with a 16-bit prescaler and software-programmable output inversion of the
- PWM signal

# TM4C123GH6PM GPT

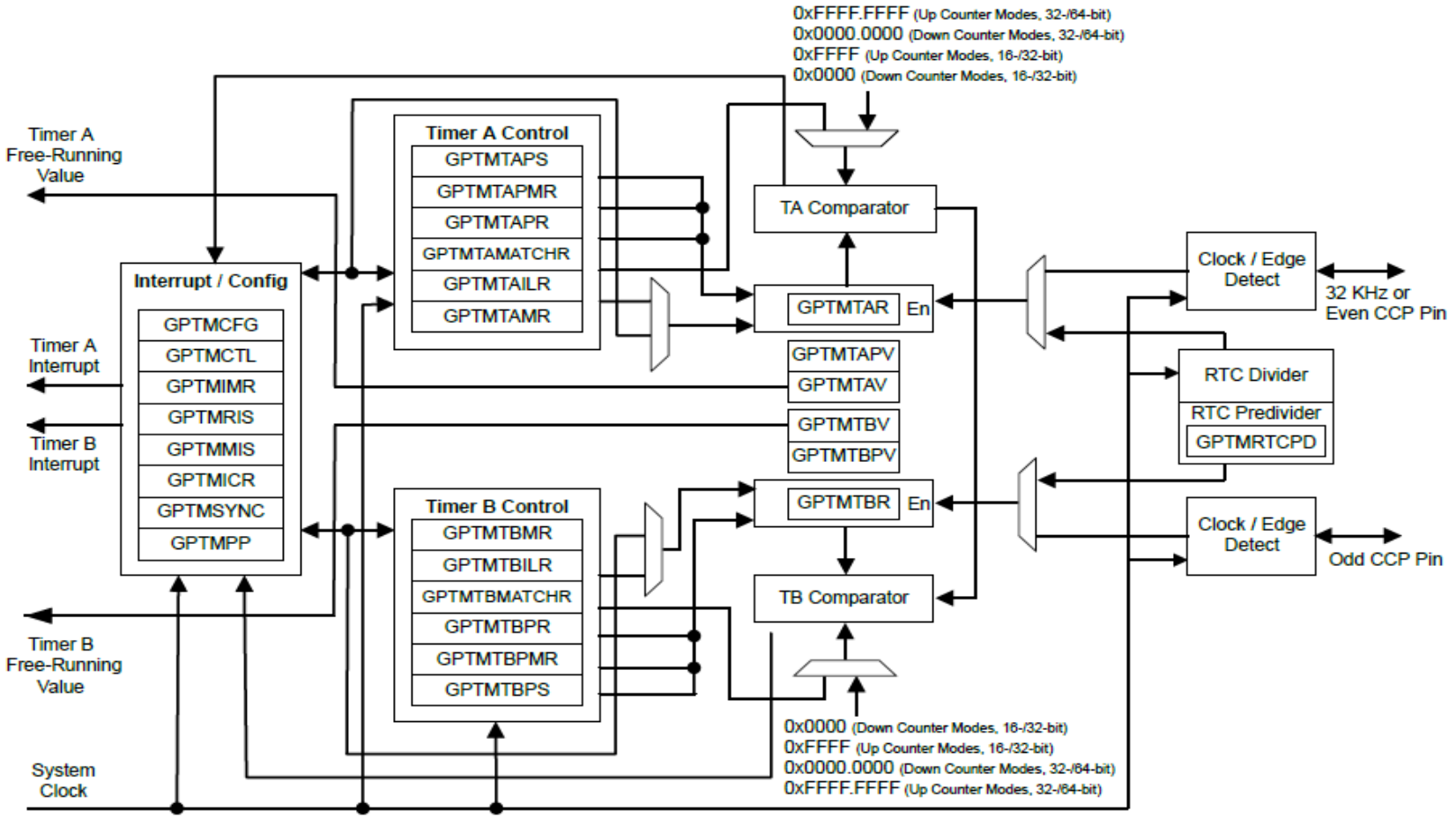
**The General-Purpose Timer Module (GPTM) contains six 16/32-bit GPTM blocks and six 32/64-bit Wide GPTM blocks with the following functional options:**

**Other features:**

- Count up or down
- Twelve 16/32-bit Capture Compare PWM pins (CCP)
- Twelve 32/64-bit Capture Compare PWM pins (CCP)
- Daisy chaining of timer modules to allow a single timer to initiate multiple timing events
- Timer synchronization allows selected timers to start counting on the same clock cycle
- ADC event trigger
- User-enabled stalling when the microcontroller asserts CPU Halt flag during debug (excluding RTC mode)
- Ability to determine the elapsed time between the assertion of the timer interrupt and entry into the interrupt service routine
- Efficient transfers using Micro Direct Memory Access Controller (µDMA)

# TM4C123GH6PM GPT

## Block Diagram:



# TM4C123GH6PM GPT

**Block Diagram:**

The main components of each GPTM block are:

**1- Two free running up/down counters known as timer A and Timer B.**

- Each one of them can work in 16 bit mode and can concatenated to together to make it 32 bit counter or 32 bit mode and can concatenated together to make it 64 bit counter.
- The timer timeout value is controlled via the Load initialization registers (GPTMTAILR and GPTMTBILR).
- The free running counter value is monitored by the timer shadow registers (GPTMTAV and GPTMTBV).

**2- Two prescaler units is counting till prescaler match every timer count.**

- Each timer prescaler value is initialized by the prescaler registers (GPTMTAPR and GPTMTBPR).
- The run time prescaler value is read via the prescaler value register.(GPTMTAPV and GPTMTBPV).

# TM4C123GH6PM GPT

**Block Diagram:**

The main components of each GPTM block are:

**3- Two Compare match units to control the compare match interrupts.**

- The value of the compare match is initialized via the compare match registers (GPTMTAMATCHR, GPTMTBMATCHR)

**4- Two interrupt units to generate interrupts at the following events:** Write update error, timeout event, compare match event. Capture mode event and RTC event.

- The Different interrupt types are enabled/disabled via the GPTM Interrupt Mask (GPTMIMR) register.
- Timer interrupts status (flags) are read via the GPTM Raw Interrupt Status (GPTMRIS) and GPTM Masked Interrupt Status (GPTMMIS) register.
- Timer interrupts flags are cleared via the GPTM Interrupt Clear (GPTMICR) register.

# TM4C123GH6PM GPT

## GPTM signals:

| Pin Name | Pin Number | Pin Mux / Pin Assignment | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|---|
| T0CCP0 | 1<br>28 | PB6 (7)<br>PF0 (7) | I/O | TTL | 16/32-Bit Timer 0 Capture/Compare/PWM 0. |
| T0CCP1 | 4<br>29 | PB7 (7)<br>PF1 (7) | I/O | TTL | 16/32-Bit Timer 0 Capture/Compare/PWM 1. |
| T1CCP0 | 30<br>58 | PF2 (7)<br>PB4 (7) | I/O | TTL | 16/32-Bit Timer 1 Capture/Compare/PWM 0. |
| T1CCP1 | 31<br>57 | PF3 (7)<br>PB5 (7) | I/O | TTL | 16/32-Bit Timer 1 Capture/Compare/PWM 1. |
| T2CCP0 | 5<br>45 | PF4 (7)<br>PB0 (7) | I/O | TTL | 16/32-Bit Timer 2 Capture/Compare/PWM 0. |
| T2CCP1 | 46 | PB1 (7) | I/O | TTL | 16/32-Bit Timer 2 Capture/Compare/PWM 1. |
| T3CCP0 | 47 | PB2 (7) | I/O | TTL | 16/32-Bit Timer 3 Capture/Compare/PWM 0. |
| T3CCP1 | 48 | PB3 (7) | I/O | TTL | 16/32-Bit Timer 3 Capture/Compare/PWM 1. |
| T4CCP0 | 52 | PC0 (7) | I/O | TTL | 16/32-Bit Timer 4 Capture/Compare/PWM 0. |
| T4CCP1 | 51 | PC1 (7) | I/O | TTL | 16/32-Bit Timer 4 Capture/Compare/PWM 1. |
| T5CCP0 | 50 | PC2 (7) | I/O | TTL | 16/32-Bit Timer 5 Capture/Compare/PWM 0. |
| T5CCP1 | 49 | PC3 (7) | I/O | TTL | 16/32-Bit Timer 5 Capture/Compare/PWM 1. |

# TM4C123GH6PM GPT

## GPTM signals:

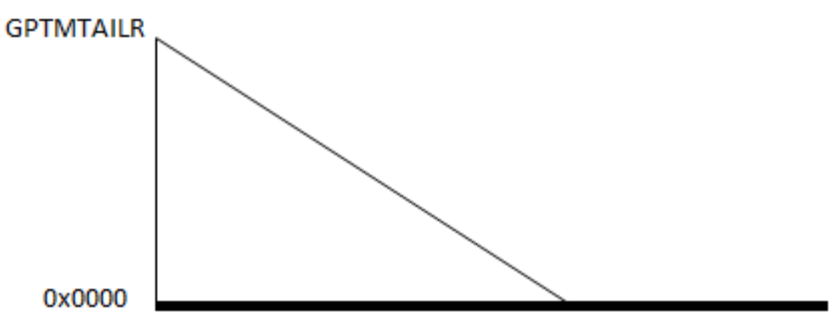
| Pin Name | Pin Number | Pin Mux / Pin Assignment | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|---|
| WT0CCP0 | 16 | PC4 (7) | I/O | TTL | 32/64-Bit Wide Timer 0 Capture/Compare/PWM 0. |
| WT0CCP1 | 15 | PC5 (7) | I/O | TTL | 32/64-Bit Wide Timer 0 Capture/Compare/PWM 1. |
| WT1CCP0 | 14 | PC6 (7) | I/O | TTL | 32/64-Bit Wide Timer 1 Capture/Compare/PWM 0. |
| WT1CCP1 | 13 | PC7 (7) | I/O | TTL | 32/64-Bit Wide Timer 1 Capture/Compare/PWM 1. |
| WT2CCP0 | 61 | PD0 (7) | I/O | TTL | 32/64-Bit Wide Timer 2 Capture/Compare/PWM 0. |
| WT2CCP1 | 62 | PD1 (7) | I/O | TTL | 32/64-Bit Wide Timer 2 Capture/Compare/PWM 1. |
| WT3CCP0 | 63 | PD2 (7) | I/O | TTL | 32/64-Bit Wide Timer 3 Capture/Compare/PWM 0. |
| WT3CCP1 | 64 | PD3 (7) | I/O | TTL | 32/64-Bit Wide Timer 3 Capture/Compare/PWM 1. |
| WT4CCP0 | 43 | PD4 (7) | I/O | TTL | 32/64-Bit Wide Timer 4 Capture/Compare/PWM 0. |
| WT4CCP1 | 44 | PD5 (7) | I/O | TTL | 32/64-Bit Wide Timer 4 Capture/Compare/PWM 1. |
| WT5CCP0 | 53 | PD6 (7) | I/O | TTL | 32/64-Bit Wide Timer 5 Capture/Compare/PWM 0. |
| WT5CCP1 | 10 | PD7 (7) | I/O | TTL | 32/64-Bit Wide Timer 5 Capture/Compare/PWM 1. |

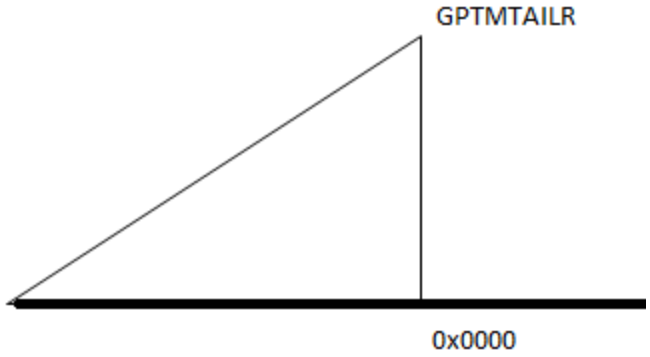
Note that:
The AFSEL bit in the (GPIOAFSEL) register should be set to choose the GP Timer function. The PMCn field in the (GPIOPCTL) register should selected to assign the GP Timer signal to the specified GPIO port pin. The GPIODEN register should be configured to make the GPIO pin digital.

# TM4C123GH6PM GPT

## GPTM Modes of operations:

### 1- One-Shot/Periodic Timer Mode:



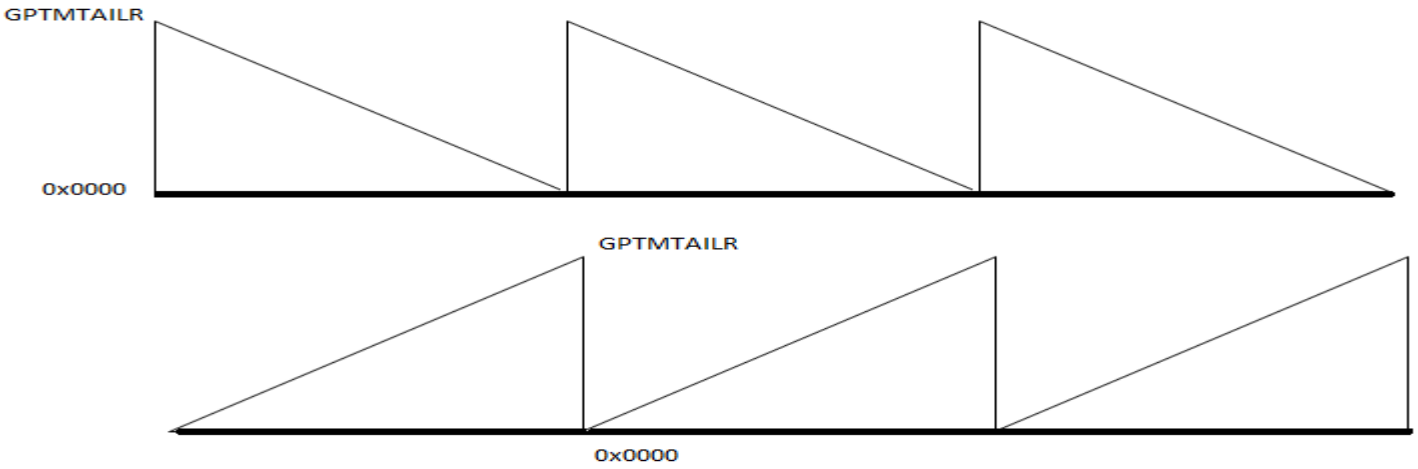




| Timer mode | Count direction | Start condition | Start value | End value | After timeout |
|---|---|---|---|---|---|
| One shot | Up | Timer is enabled | 0x0000 | GPTMTALIR | Stop counting, disable timer |
| One shot | Down | Timer is enabled | GPTMTALIR | 0x0000 | Stop counting, disable timer |

# TM4C123GH6PM GPT

## GPTM Modes of operations:

### 1- One-Shot/Periodic Timer Mode:




| Timer mode | Count direction | Start condition | Start value | End value | After timeout |
|---|---|---|---|---|---|
| Periodic | Up | Timer is enabled | 0x0000 | GPTMTALIR | Reload by 0x0000 and restart counting |
| Periodic | Down | Timer is enabled | GPTMTALIR | 0x0000 | Reload by GPTMTALIR and restart counting |

# TM4C123GH6PM GPT

## GPTM Modes of operations:

### 1- One-Shot/Periodic Timer Mode:

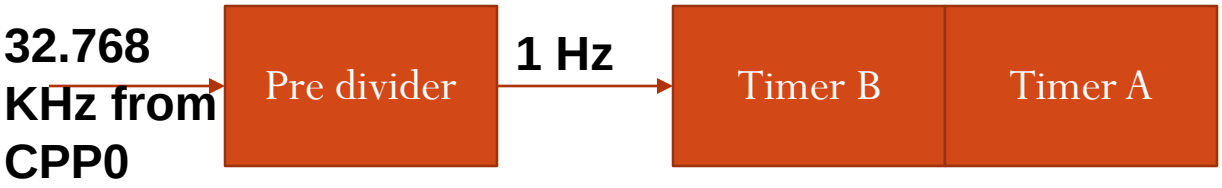
- CCP outputs and triggers when it reaches the time-out event.
- Two interrupts are supported, Timeout interrupt when Timer reaches 0 when downcounting or when timer reaches GPTMTALIR.
- The other one is a compare match interrupt when the timer value reaches the value loaded to the register GPTMTnMATCHR.
- Update of the timer load register GPTMTALIR or the prescaler register GPTMTnPR may configured take an immediate effect or after the next timeout.
- If the two timers are used in concatenated mode, only the control bits of timer A registers are used.
- Timer may be configured to freeze counting when a halt is inserted by debugger.

# TM4C123GH6PM GPT

## GPTM Modes of operations:

**2- Real time clock (RTC) mode:**

- Concatenated versions of the Timer A and Timer B registers are configured as an up-counter.



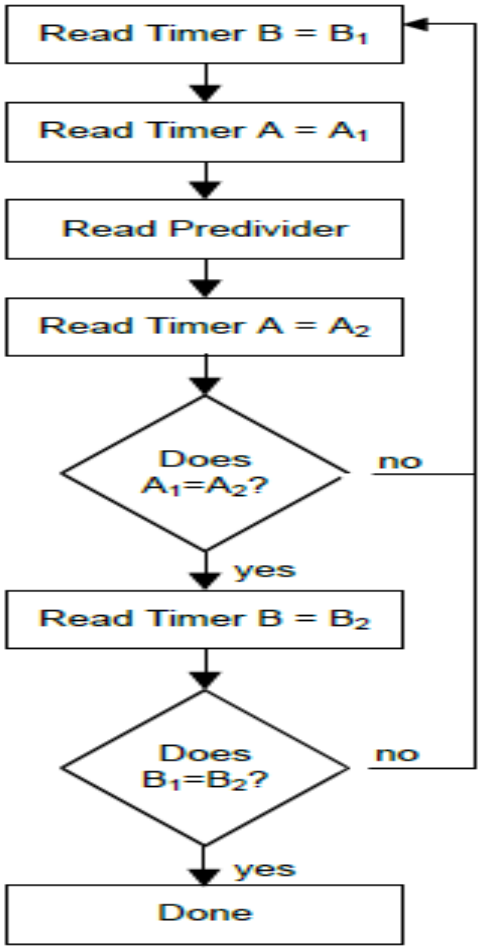

- The input clock on a CCP0 input is required to be 32.768 KHz in RTC mode.
- The clock signal is then divided down to a 1-Hz rate and is passed along to the input of the counter.
- When the timer is enabled it starting counting from a preloaded value equal 0x01.
- If the **GPTMTnILR** register is loaded with a new value, the counter begins counting at that value and rolls over at the fixed value of 0xFFFFFFFF
- When the counter value reach the value saved in the compare match register it generated RTC event interrupt.

# TM4C123GH6PM GPT

## GPTM Modes of operations:

**2- Real time clock (RTC) mode:**

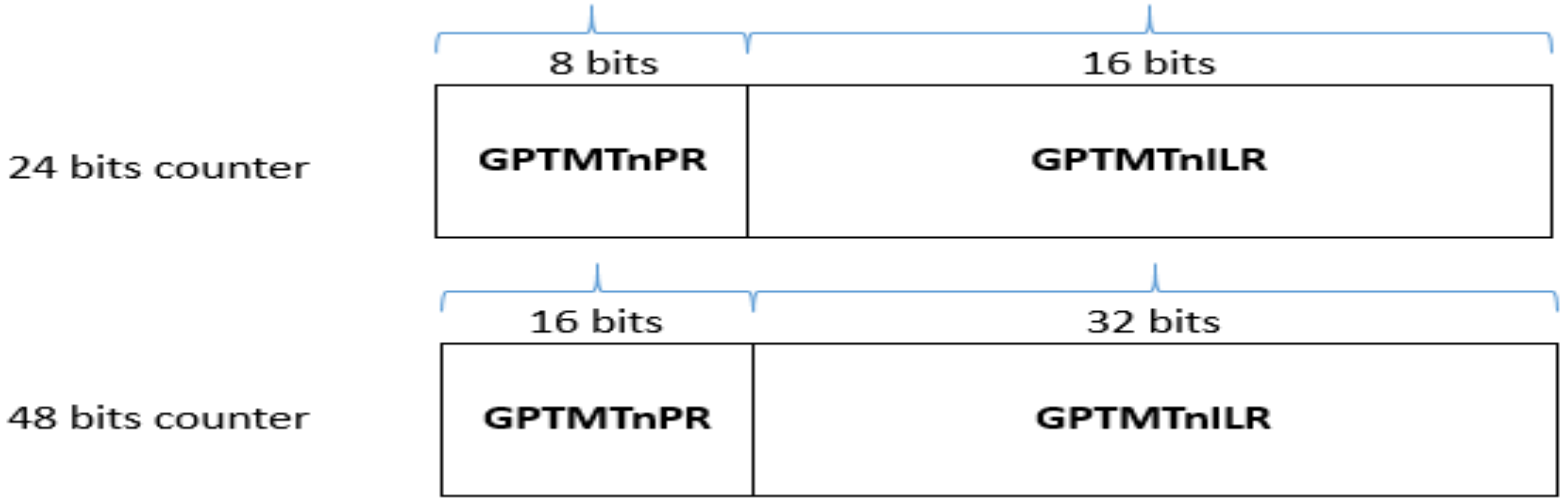
- To ensure that the RTC value is coherent, software should follow the process detailed in Figure



# TM4C123GH6PM GPT

## GPTM Modes of operations:

### 3- Input Edge-Count Mode:

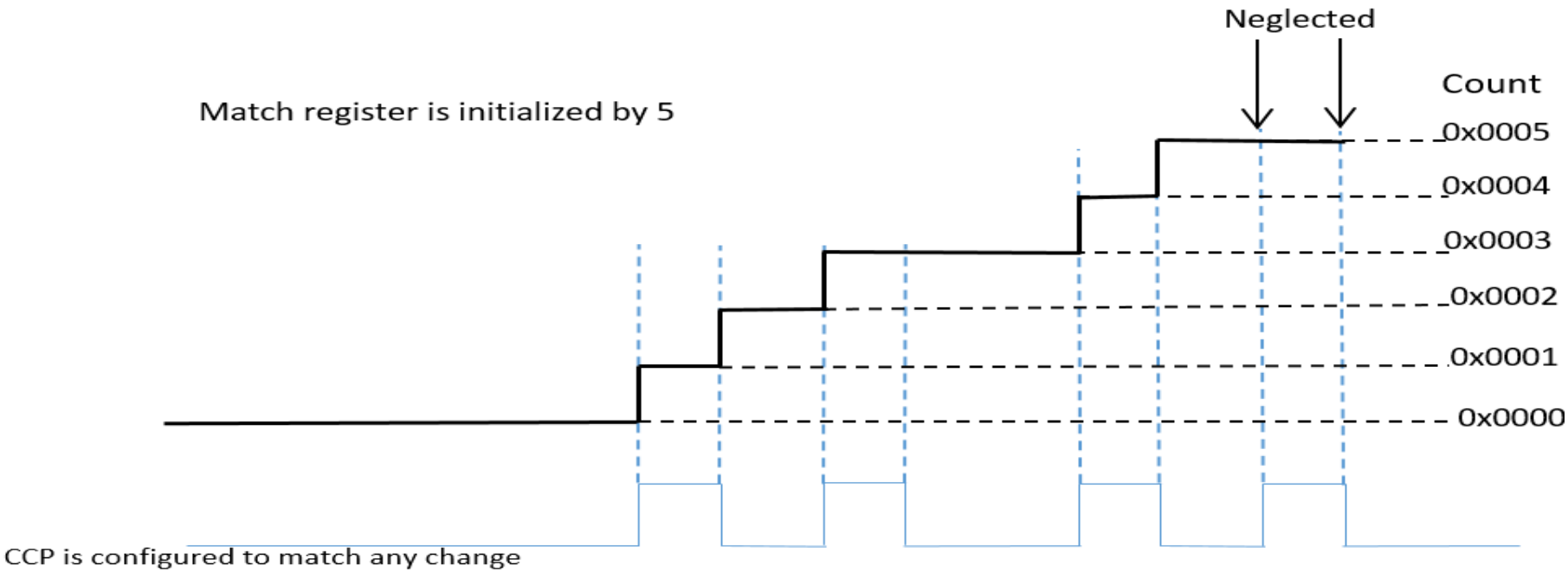



the timer is configured as a 24-bit or 48-bit up- or up- or down-counter including the optional prescaler.
the timer is capable of capturing three types of events: rising edge, falling edge, or both.

# TM4C123GH6PM GPT

## GPTM Modes of operations:

### 3- Input Edge-Count Mode:

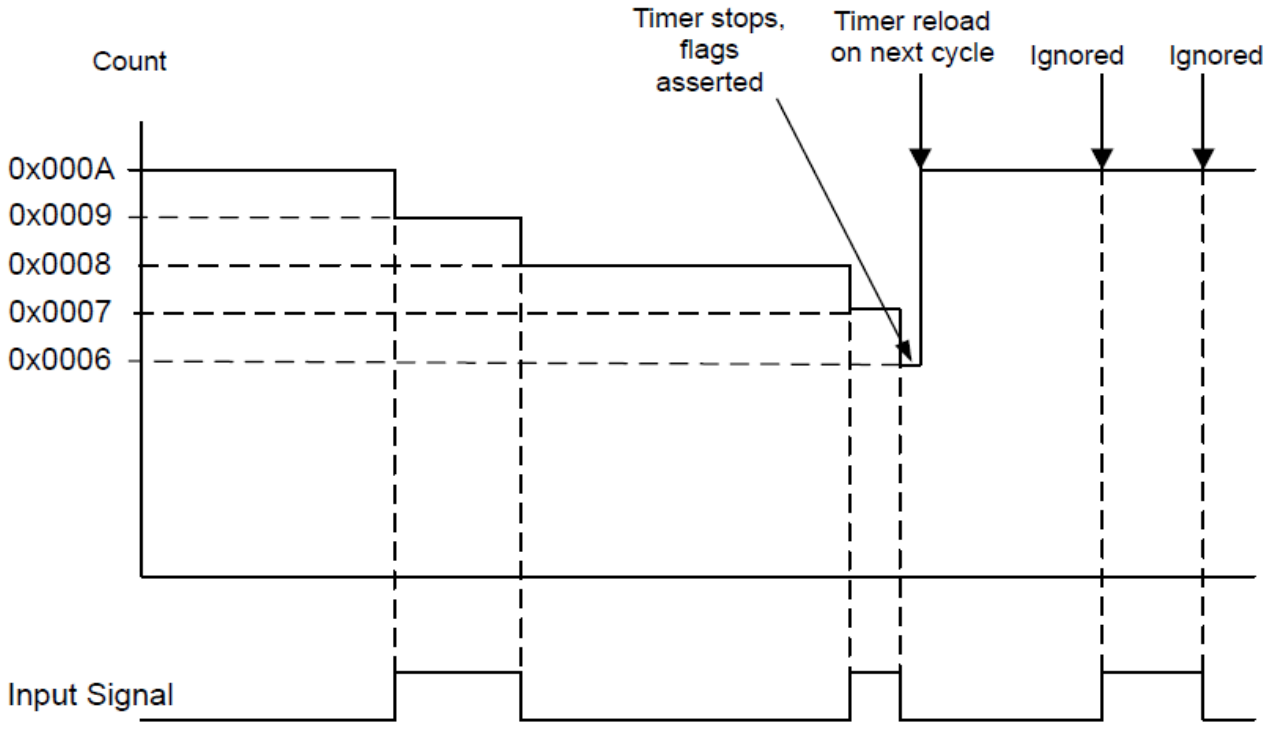



When up counting the number of detected counts will be the value saved in the match registers (**GPTMTnMATCHR** and **GPTMTnPMR**) as the counter stating from 0x00.

# TM4C123GH6PM GPT

## GPTM Modes of operations:

### 3- Input Edge-Count Mode:




When down counting, the timer is counting starting from the value saved in the load registers (**GPTMTnILR** and **GPTMTnPR**) and stopped when reaching the Match registers.

# TM4C123GH6PM GPT

**GPTM Modes of operations:**

**3- Input Edge-Count Mode:**

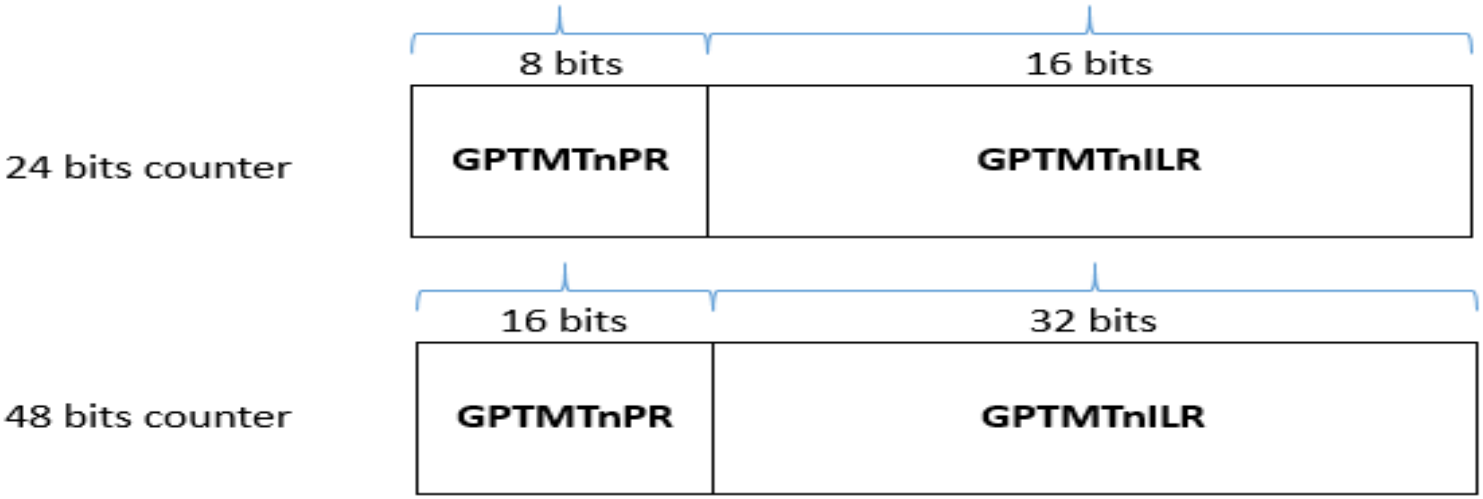
**Notes:**

- when executing an up-count, that the value of **GPTMTnPR** and **GPTMTnILR** must be greater than the value of **GPTMTnPMR** and **GPTMTnMATCHR**.
- When the counting matches, the timer sets the corresponding bit in the raw interrupt status and still high until cleared by setting the corresponding bit in the interrupt clear register.
- If interrupt is enabled, the timer also sets the masked interrupt status when match occurred.
- After match, the timer will be reloaded by zero in up counting and by the value saved in **GPTMTnILR** and **GPTMTnPR** registers when down counting.
- After match, the timer will stop counting as it will automatically disable the timer.

# TM4C123GH6PM GPT

## GPTM Modes of operations:

### 4- Input Edge-Time Mode:

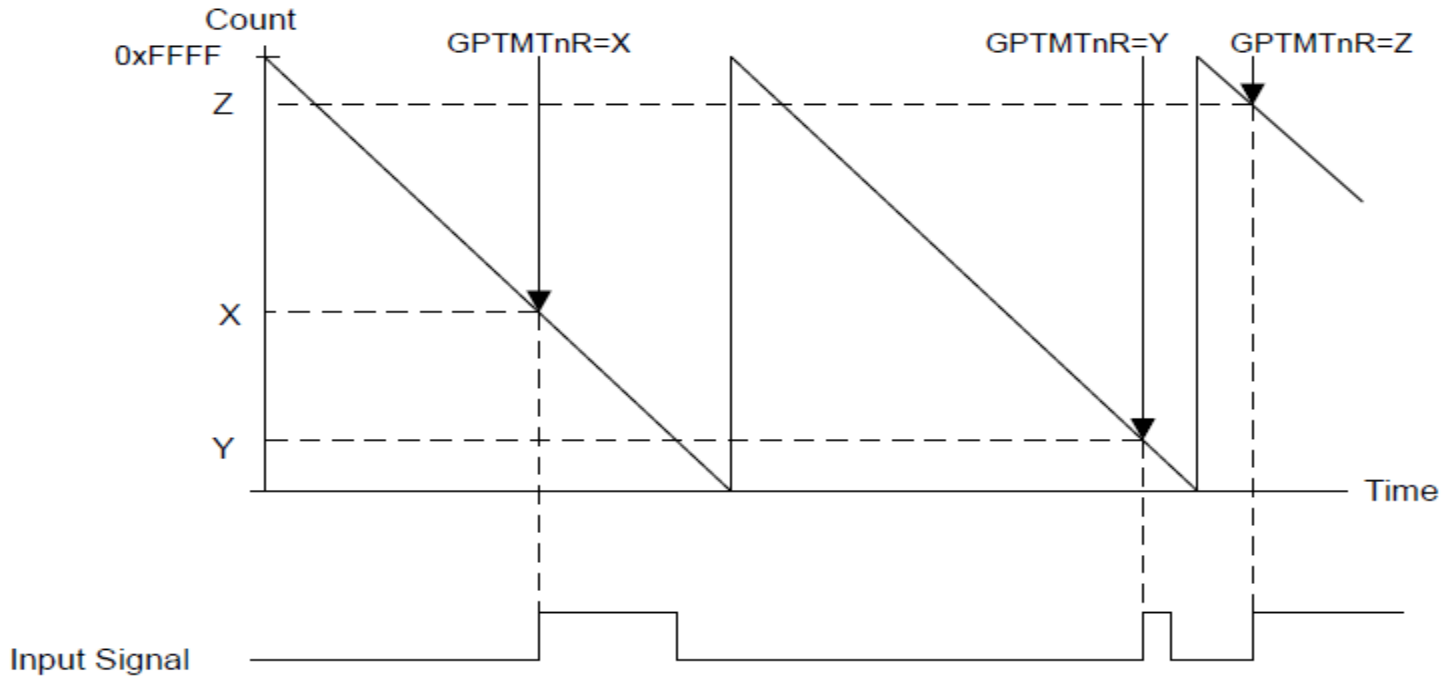



- the timer is configured as a 24-bit or 48-bit up- or down-counter including the optional prescaler.
- The timer is capable of capturing three types of events: rising edge, falling edge, or both.
- the timer is initialized to the value loaded in the **GPTMTnILR**
- and **GPTMTnPR** registers when counting down and 0x0 when counting up.
- The timer is counting till the timeout and then reload its registers with the starting value and continue counting.

# TM4C123GH6PM GPT

## GPTM Modes of operations:

### 4- Input Edge-Time Mode:




- When the selected input event is detected, the current timer counter value is captured in the **GPTMTnR** and **GPTMTnPS** register and is available to be read by the microcontroller.

# TM4C123GH6PM GPT

**GPTM Modes of operations:**

**4- Input Edge-Time Mode:**

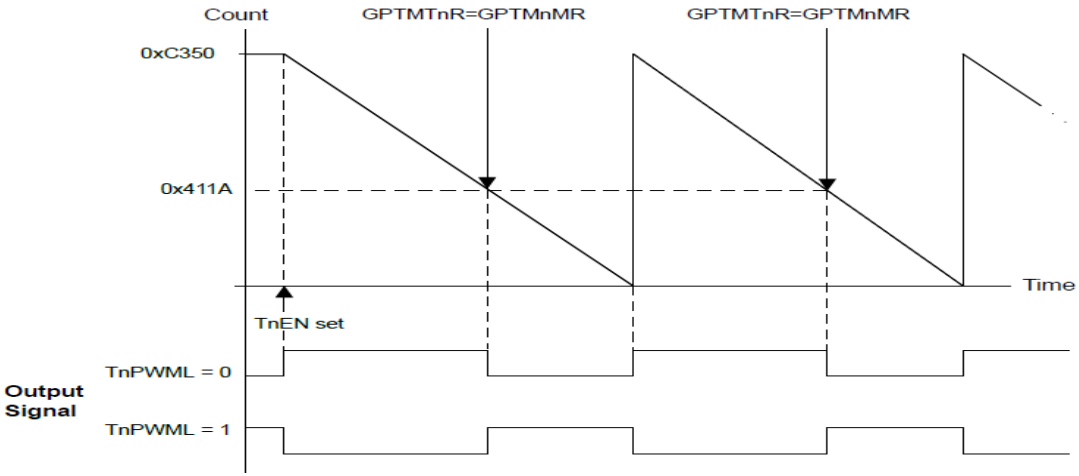
**Notes:**

- When the selected event detected, the timer sets the corresponding bit in the interrupt raw status register and still hold until cleared by setting the corresponding bit in the interrupt clear register.
- If the capture mode event interrupt is enabled, the timer will set the corresponding bit in the masked interrupt status register.

# TM4C123GH6PM GPT

## GPTM Modes of operations:

### 5- PWM mode:




- The timer is configured as a 24-bit or 48-bit down-counter with a start value (and thus period) defined by the **GPTMTnILR** and **GPTMTnPR** registers.
- When the timer enabled, the counter begins counting down from until it reaches the 0x0 state.
- On the next counter cycle in periodic mode, the counter reloads its start value from the GPTMTnILR and GPTMTnPR registers and continues counting until disabled by software.
- The output PWM signal asserts when the counter is at the value of the **PTMTnILR** and **GPTMTnPR** registers (its start state), and is deasserted when the counter value equals the value in the **GPTMTnMATCHR** and **GPTMTnPMR** registers.

# TM4C123GH6PM GPT

**GPTM Modes of operations:**

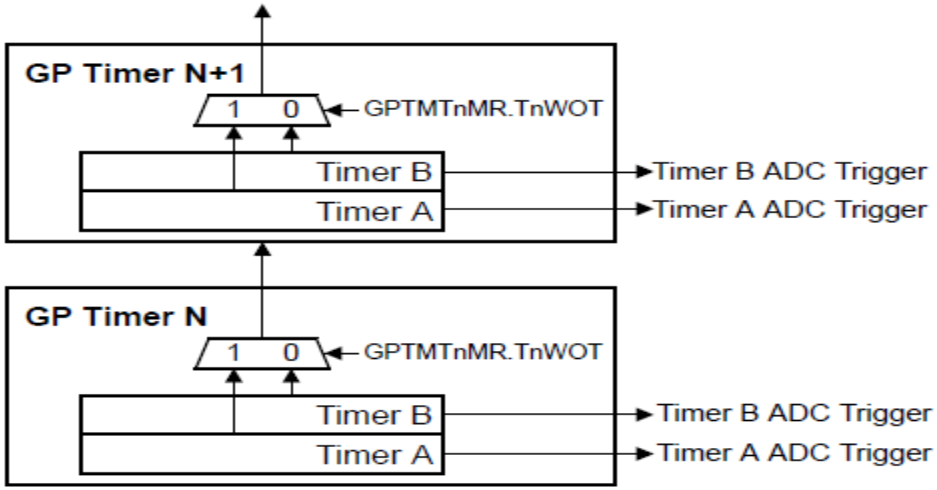
**5- PWM mode:**

**Notes:**

- Software has the capability of inverting the output PWM signal by setting the TnPWML bit in the **GPTMCTL** register.
- he timer is capable of generating interrupts based on three types of events: rising edge, falling edge, or both for the output signal.
- When event occurred, the timer sets the corresponding bit in the raw interrupt status register and holds until it is cleared by setting the corresponding bit in the interrupt clear register.
- If the capture mode event interrupt is enabled, the timer will also set the corresponding bit in the masked interrupt status register.
- If PWM output inversion is enabled, edge detection interrupt behavior is reversed.

# TM4C123GH6PM GPT

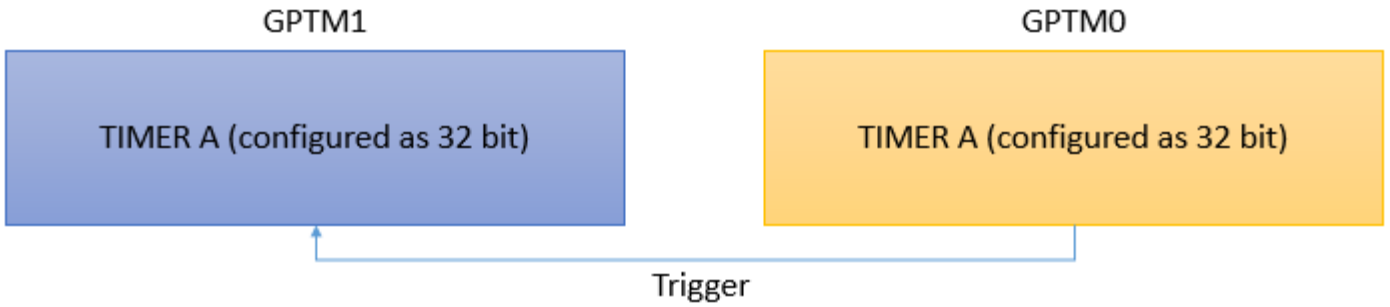
## GPTM timers Daisy chaining:



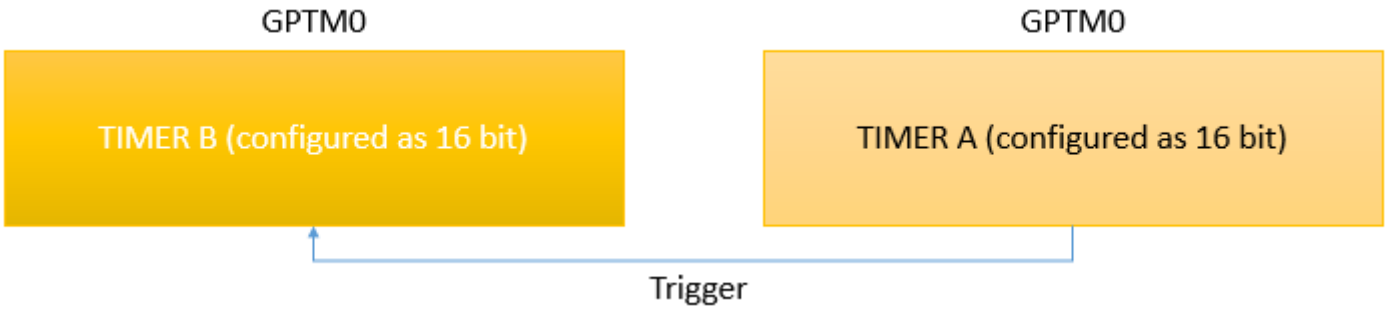

- The Wait-for-Trigger mode allows daisy chaining of the timer modules such that once configured, a single timer can initiate multiple timing events using the Timer triggers.
- Wait-for-Trigger mode is enabled by setting the TnWOT bit in the **GPTMTnMR** register.
- When the TnWOT bit is set, Timer N+1 does not begin counting until the timer in the previous position in the daisy chain (Timer N) reaches its time-out event.
- Care must be taken that the TAWOT bit is never set in GPTM0.

# TM4C123GH6PM GPT

## GPTM timers Daisy chaining:




- If Timer A is configured as a 32-bit (16/32-bit mode) or 64-bit (32/64-bit wide mode) timer, it triggers Timer A in the next module.




- If Timer A is configured as a 16-bit (16/32-bit mode) or 32-bit (32/64-bit wide mode) timer, it triggers Timer B in the same module, and Timer B triggers Timer A in the next module.

# Thank You