UNIT-IV

Microcontroller Fundamentals for Basic Programming

GPIOs:

General Purpose Input/output (GPIO) refers to pins on a board which are connected to the microcontroller in a special configuration. Users can control the activities of these pins in real-time.

- GPIOs are used in devices like SoC, PLDs, and FPGAs, which inherit problems of pin scarcity.
- They are used in multifunction chips like audio codecs and video cards for connectivity
- They are extensively used in embedded systems designs to interface the microcontroller to external sensors and driver circuits.

GPIO pins can be configured as both input and output. There are generally two states in a GPIO pin, High=1 and Low=0. These pins can be easily enabled and disabled by the user. A GPIO pin can be configured as input and used as an interrupt pin typically for wakeup events. We will see this later in this chapter when we use a switch to force the system wake from hibernation. GPIO peripherals vary quite widely. In some cases, they can exist as a group of pins that can be switched as a group to either input or output. In others, each pin can be set up adaptable to either accept or act as a source for different logic voltages, with configurable drive strengths and pull ups. Pin states of the GPIOs can be accessed using software instructions. These instructions can be represented by one or more types of interfaces. Memory mapped peripheral or a dedicated I/O port instruction can be used in this regard.

Voltage levels of GPIOs are critical and it is necessary that users take note of these voltages before interfacing. Tolerant voltages at GPIO pins are not same as the board supply voltage. Some GPIOs have 5 V tolerant inputs: even if the device has a low supply voltage (say 2 V), it can accept 5 V without damage. However, a higher voltage may cause damage to the circuitry or may even fry the board.

GPIO Pins in Tiva Launchpad:

In the Tiva Launchpad, the GPIO module is composed of six physical GPIO blocks. Each of these blocks corresponds to an individual GPIO port. There are six ports in Tiva C series microcontrollers namely, Port A through F. This GPIO module supports up to 43 programmable input/output pins. (Although it depends on the peripherals being used)

The GPIO module has the following features:

- The GPIO pins are flexibly multiplexed. This allows it to be also used as peripheral functions.

- The GPIO pins are 5-V-tolerant in input configuration
- Ports A-G are accessed through the Advanced Peripheral Bus (APB)
- Fast toggle capable of a change every clock cycle for ports on AHB, every two clock cycles for ports on APB.

Most of the GPIO functions can operate on more than one GPIO pin (within a single module) at a time. Can be configured to be a GPIO or a peripheral pin. On reset, the default is GPIO. Note that not all pins on all parts have peripheral functions, in which case, e the pin is only useful as a GPIO.

Advanced features of GPIO in Tiva Launchpad:

The GPIO module in Tiva Launch Pad can be used in advanced configurations also. They can be used for programmable control through interrupts. These interrupts can be triggered on rising, falling or both edges of the clock. They can also be levelled sensitive for both high and low states. The state of these pins is retained during hibernate mode. The programmable control for GPIO pad configuration includes

- 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

Programming System Registers:

Direction and Data Registers:

Generally, every microcontroller has a minimum of two registers associated with each of I/O ports, namely Data Register and Direction Register. As the name suggests, Direction Register decides which way the data will flow; Input or Output. Data register stores the data coming from the microcontroller or from the pin.

The value assigned to Direction register is configuring the pin as either input or output. When the direction register is properly configured, the Data register can be used to write to the pin or read data from the pin. When the Direction register is configured as output, the information on the Data register is driven to the microcontroller pin. Similarly, when Direction register is configured as input, the information on the microcontroller pin is written to the Data register.

Data Direction Operation: In Tiva C series Launchpad, the GPIO Direction (GPIODIR) register is used to configure each individual pin as an input or output. When the data direction bit is cleared, the GPIO is configured as an input, and the corresponding data register bit captures and stores the value on the GPIO port. When the data direction bit is set, the GPIO is configured as an output, and the corresponding data register bit is driven out on the GPIO port.

Data Register Operation: In Tiva C Series Launchpad, GPIODATA register is the data register in which the values written in this register are transferred onto the GPIO port pins if the respective pins have been configured as outputs through the GPIO Direction (GPIODIR)

register. The GPIO ports allow for the modification of individual bits in the GPIO Data (GPIODATA) register by using bits of the address bus as a mask. In this manner, we can modify individual GPIO pins in a single instruction without affecting the state of the other pins.

Toggling Multicolor LED:

In this section we will understand how GPIOs are accessed. Embedded engineers have been using on-board LEDs for a long time in debugging hardware programs. Thus blinking and LED serves a much higher purpose beneath than it appears. The most famous example, called 'Blinky,' is used just to light up a particular onboard LED.

Switches and RGB LED: The Tiva C Series LaunchPad comes with an RGB LED. This LED is used in the preloaded RGB quick-start application and can be configured for use in custom applications. And two switch buttons are included on the board, used in the preloaded quick-start application to adjust the light spectrum of the RGB LED as well as go into and out of hibernation.

Three GPIO pins connected to the LEDs as outputs and two GPIO pins are connected to switches. The LaunchPad board schematic and below table shows GPIO pins PF1, PF2 and PF3 are connected to the red, blue and green color of RGB LEDs and PF4, PF0 are connected to the SW1, SW2 of user switches respectively.

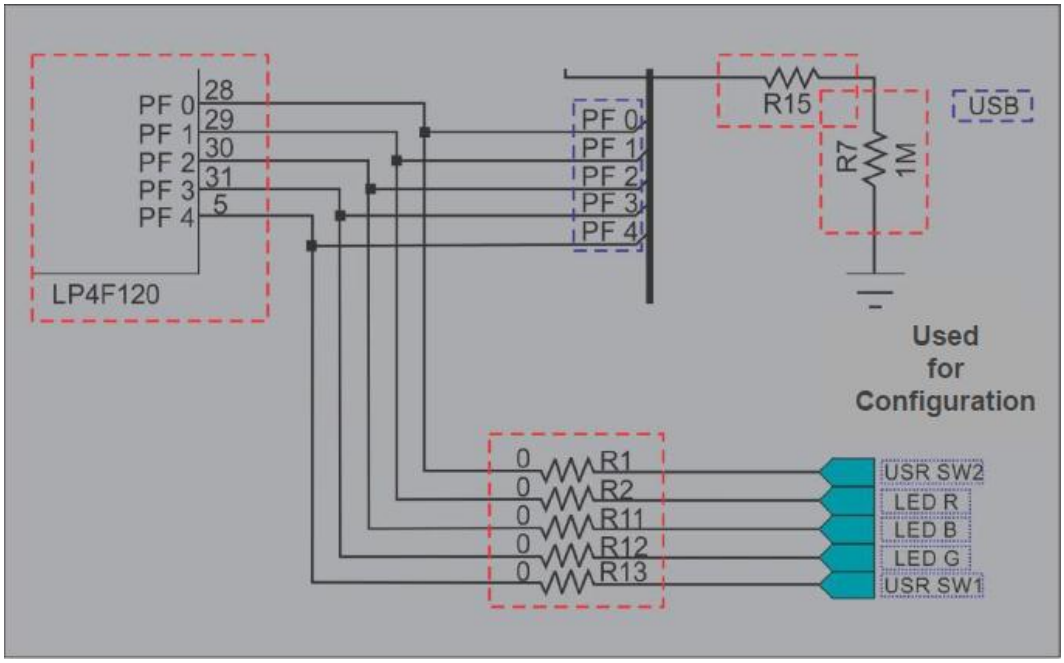

Figure: Switches and RGB Schematic Connection to GPIO Pins

Table: Switches – RGB and GPIO pin connection to the microcontroller

GPIO Pin	Pin Function	USB Device
PF4	GPIO	SW1
PF0	GPIO	SW2
PF1	GPIO	RGB LED (Red)
PF2	GPIO	RGB LED (Blue)
PF3	GPIO	RGD LED (Green)

Watchdog Timer:

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

A watchdog timer counter enters a counter lapse or timeout after it reaches certain count. Under normal operation, the program running the system continuously resets the watchdog timer. When the system enters an infinite loop or stops responding, it fails to reset the watchdog timer. In due time, the watchdog timer enters counter lapse. This timeout will trigger a reset signal to the system or call for an interrupt service routine (ISR).

The TM4C123GH6PM microcontroller has two Watchdog Timer modules, one module is clocked by the system clock (Watchdog Timer 0) and the other (Watchdog Timer 1) is clocked by the PIOSC therefore it requires synchronizers.

Features of Watchdog Timer in TM4C123GH6PM controller:

- 32-bit down counter with a programmable load register
- Separate watchdog clock with an enable
- Programmable interrupt generation logic with interrupt masking & optional NMI function
- Lock register protection from runaway software
- Reset generation logic with an enable/disable
- User-enabled stalling when the microcontroller asserts the CPU halt flag during debug

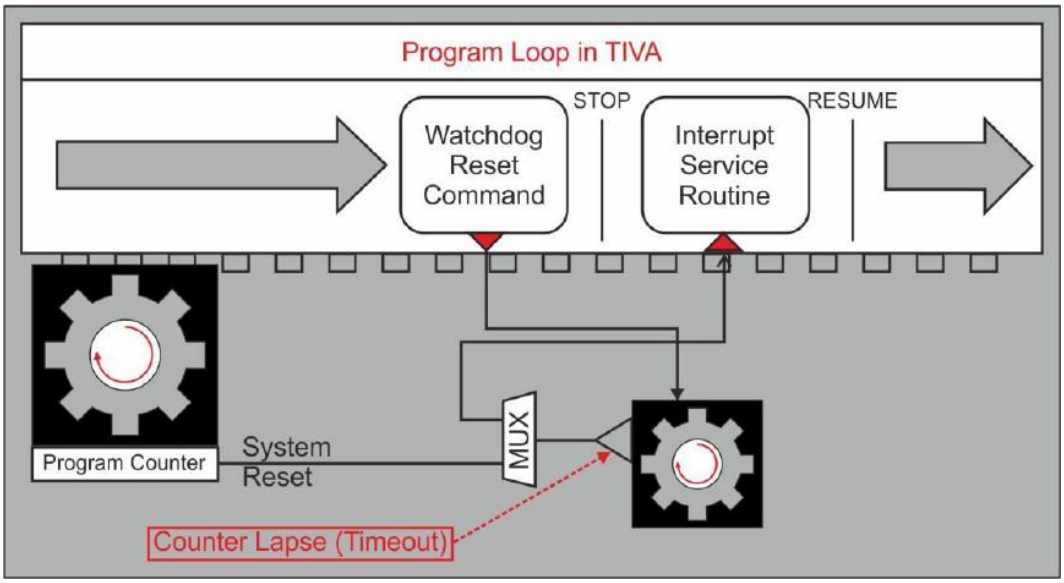

Figure: Operation of Watchdog Timer

The watchdog timer can be configured to generate an interrupt to the controller on its first time out, and to generate a reset signal on its second time-out. Once the watchdog timer has been configured, the lock register can be written to prevent the timer configuration from being inadvertently altered.

Hibernation Module on TM4C:

The watchdog timer can be configured to generate an interrupt to the controller on its first time out, and to generate a reset signal on its second time-out. Once the watchdog timer has been configured, the lock register can be written to prevent the timer configuration from being inadvertently altered.

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

(i) A Real-Time Clock (RTC) to be used for wake events

(ii) A battery backed SRAM for storing and restoring processor state. The SRAM consists of 16 32-bit word memory.

The RTC is a 32- bit seconds counter and 15- bit sub second counter. It also has an add-in trim capability for precision control over time. The Microprocessor has a dedicated pin for waking using external signal. The RTC and the SRAM are operational only if there is a valid battery voltage. There is a VDD30N mode, which provides GPIO pin state during hibernation of the device.

Thus we are actually shutting the power off for the device or part at the lowest power mode. Under such circumstances, it is safe to assume that in the wake up we are actually coming out of reset. But this will allow the device to the keep the GPIO pins in their state without resetting them. A mechanism for power control is used to shut down the part. In TM4C123GH6PM we have an on-chip power controller which controls power for the CPU only. There is also a pin output from the microcontroller which is used for system power control.

It should be duly noted that in TIVA Launchpad, the battery voltage is directly connected to the processor voltage and it is always valid. But in a custom design with TM4C123GH6PM microcontroller running on a battery, if the battery voltage is not valid, it will not go into hibernation mode.

The Hibernation module of TM4C123GH6PM provides two mechanisms for power control:

- ⇨ The first mechanism uses internal switches to control power to the Cortex-M4F.
- ⇨ The second mechanism controls the power to the microcontroller with a control signal (HIB) that signals an external voltage regulator to turn on or off.

The Hibernation module power source is determined dynamically. The supply voltage of the Hibernation module is the larger of the main voltage source (VDD) or the battery voltage source (VBAT).

Hibernate mode can be entered through one of two ways:

- ⇨ The user initiates hibernation by setting the HIBREQ bit in the Hibernation Control (HIBCTL) register.
- ⇨ Power is arbitrarily removed from VDD while a valid VBAT is applied

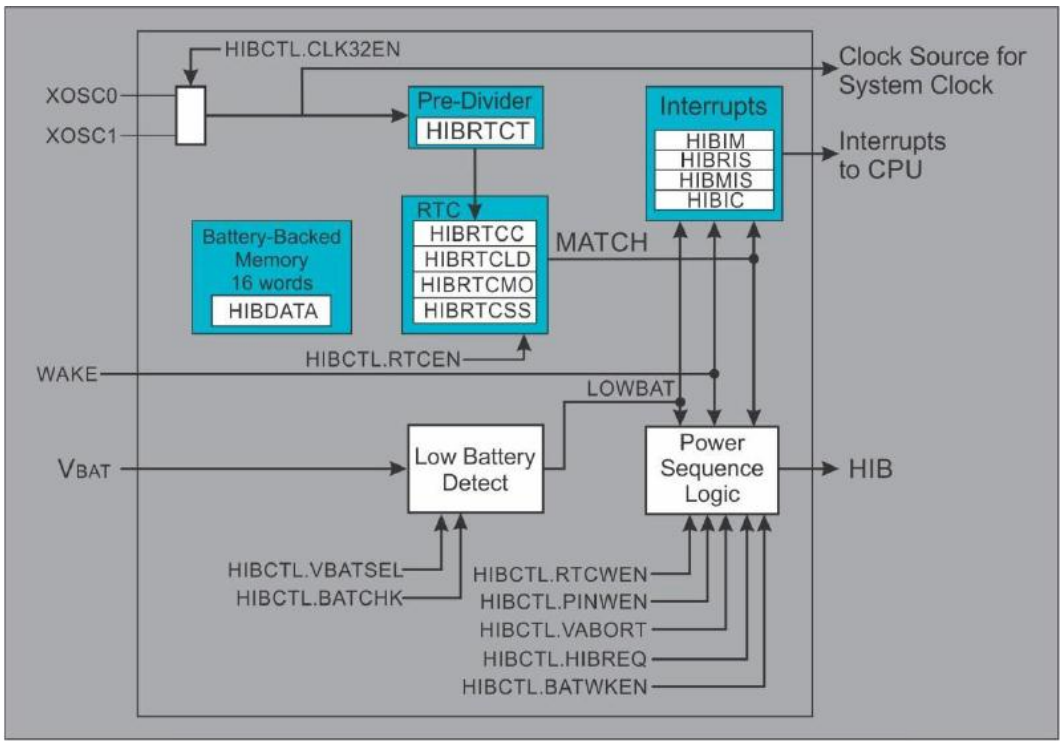

Figure: Block diagram of Hibernation module

Active Vs Standby Current Consumption:

Power Modes

There are six power modes in which TM4C123GH6PM operates as shown in the below table. They are Run, Sleep, Deep Sleep, Hibernate with VDD3ON, Hibernate with RTC, and Hibernate without RTC. To understand all these modes and compare them, it is necessary to analyze them under a condition.

Let us consider that the device is operating at 40 MHz system clock with PLL.

Table: Power Modes of Tiva

Mode → / Parameter ↓	Run Mode	Sleep Mode	Deep Sleep Mode	Hibernation (VDD3ON)	Hibernation (RTC)	Hibernation (no RTC)
I_{DD}	32 mA	10 mA	1.05 mA	5 µA	1.7 µA	1.6 µA
V_{DD}	3.3 V	3.3 V	3.3 V	3.3 V	0 V	0 V
V_{BAT}	N.A.	N.A.	N.A.	3 V	3 V	3 V
System Clock	40 MHz with PLL	40 MHz with PLL	30 kHz	Off	Off	Off
Core	Powered On	Powered On	Powered On	Off	Off	Off
	Clocked	Not Clocked	Not Clocked	Not Clocked	Not Clocked	Not Clocked
Peripherals	All On	All Off	All Off	All Off	All Off	All Off
Code	while{1}	N.A.	N.A.	N.A.	N.A.	N.A.

Introduction to Interrupts:

The reader is aware that a microprocessor is connected to several input and output devices. It is important at this point for us to know how a microprocessor manages these devices efficiently.

Introduction to Interrupts and Polling:

A microprocessor executes instructions sequentially. Alongside, it is also connected to several devices. Dataflow between these devices and the microprocessor has to be managed effectively. There are two ways it is done in a microprocessor: either by using interrupts or by using polling.

Polling

Polling is a simple method of I/O access. In this method, the microcontroller continuously probes whether the device requires attention, i.e. if there is data to be exchanged. A polling function or subroutine is called repeatedly while a program is being executed. When the status of the device being polled responds to the interrogation, a data exchange is initiated. The polling subroutine consumes processing time from the presently executing task. This is a very inefficient way because I/O devices do not always crave for attention from the microprocessor. But the microprocessor wastes valuable processing time in unnecessarily polling of the devices.

Interrupts

However, in interrupt method, whenever a device requires the attention from the microprocessors, it pings the microprocessor. This ping is called interrupt signal or sometimes interrupt request (IRQ). Every IRQ is associated with a subroutine that needs to be executed within the microprocessor. This subroutine is called interrupt service routine (ISR) or sometimes interrupt handler. The microprocessor halts current program execution and attends to the IRQ by executing the ISR. Once execution of ISR completes, the microprocessor resumes the halted task.

The current state of the microprocessor must be saved before it attends the IRQ in order to be able to continue from where it was before the interrupt. To achieve this, the contents of all of its internal registers, both general purpose and special registers, are required to be saved to a memory section called the stack. On completion of the interrupt call, these register contents will be reinstated from the stack. This allows the microprocessor to resume its originally halted task.

There are two types of interrupts namely software driven interrupts (SWI) and hardware driven interrupts (HWI). SWIs are generated from within a currently executing program. They are triggered by the interrupt opcode. A SWI will call a subroutine that allows a program to access certain lower level service. HWIs are signals from a device to the microprocessor. The device sets an interrupt line in the control bus high. Microprocessors have two types of hardware interrupts namely, non-maskable interrupt (NMI) and interrupt request (INTR). An NMI has a very high priority and they demand immediate execution.

There is no option to ignore an NMI. NMI is exclusively used for events that are regarded as having a higher priority or tragic consequences for the system operation. For example, NMI can be initiated due to an interruption of power supply, a memory fault or pressing of the reset button. An INTR may be generated by a number of different devices all of which are connected to the single INTR control line. An INTR may or may not be attended by the microprocessor. If the microprocessor is attending an interrupt, then no further interrupts, other than an NMI, will be entertained until the current interrupt has been completed. A control signal is used by the microprocessor to acknowledge an INTR. This control signal is called ACK or sometimes INTA.

Interrupt Vector Table:

It is discussed in the previous section that when an interrupt occurs, the microprocessor runs an associated ISR. IRQ is an input signal to the microprocessor. When a microprocessor receives an IRQ, it pushes the PC register onto the stack and load address of the ISR onto the PC register. This makes the microprocessor execute the ISR. These associated ISRs, corresponding to every interrupt, become a part of the executable program. This executable is loaded in the memory of the device. Under such circumstances, it becomes easier to manage the ISRs if there is a lookup table where address locations of all ISRs are listed. This lookup table is called Interrupt vector table. The below table shows an interrupt vector table for ARM cortex-M microcontroller. In ARM microcontroller, there exist 256 interrupts. Out of these, some are hardware or peripheral generated IRQs and some are software generated IRQs. However, first 15 interrupts, INT0 to INT15 are called the predefined interrupts. In ARM Cortex-M microcontrollers, Interrupt vector table is an on-chip module, called as Nested Vector Interrupt Controller (NVIC).

NVIC is an on-chip interrupt controller for ARM Cortex-M series microcontrollers. No other ARM series has this on-chip NVIC. This means that the interrupt handling is primarily different in ARM Cortex-M microcontrollers compared to other ARM microcontrollers

Table: Interrupt Vector Table for ARM Cortex M4

VECTOR NO.	PRIORITY	EXCEPTION TYPE	VECTOR ADDRESS
0	-	SP initial Value	0x0000.0000
1	-3	RESET	0x0000.0004
2	-2	NMI	0x0000.0008
3	-1	Hard Fault	0x0000.000C
4	Programmable	Memory Management Fault	0x0000.0010
5	Programmable	BUS Fault	0x0000.0014
6	Programmable	Usage Fault (undefined instructions, divide by zero, unaligned memory access, etc.)	0x0000.0018

VECTOR NO.	PRIORITY	EXCEPTION TYPE	VECTOR ADDRESS
7 - 10	-	Reserved	0x0000.001C to 0x0000.0028
11	Programmable	SVCall	0x0000.002C
12	Programmable	Debug Monitor	0x0000.0030
13	-	Reserved	0x0000.0034
14	Programmable	PendSV	0x0000.0038
15	Programmable	SysTick	0x0000.003C
16 -255	Programmable	User Interrupts (interrupts generated from peripherals and software)	0x0000.0040 to 0x0000.03FC

On system reset, the vector table is fixed at address 0x0000.0000.

Predefined Interrupts (INT0-INT15)

RESET

All ARM devices have a RESET pin which is invoked on device power-up or in case of warm reset. This exception is a special exception and has the highest priority. On the assertion of Reset signal, the execution stops immediately. When the Reset signal stops, execution starts from the address provided by the Reset entry in the vector table i.e. 0x0000.0004. Hereby, to run a program on Reset, it is necessary to place the program in 0x0000.0004 memory address.

NMI

In the ARM microcontroller, some pins are associated with hardware interrupts. They are often called IRQs (interrupt request) and NMI (non-maskable interrupt). IRQ can be controlled by software masking and unmasking. Unlike IRQ, NMI cannot be masked by software. This is why I is named as nonmaskable interrupt. As shown in above Table, "INT 02" in ARM Cortex-M is used only for NMI. On activation of NMI, the microcontroller load memory location 0x0000008 to program counter.

Hard Fault

All the classes of fault corresponding to a fault handler cannot be activated. This may be a result of the fault handler being disabled or masked.

Memory Management Fault

It is caused by a memory protection unit violation. The violation can be caused by attempting to write into a read only memory. An instruction fetch is invalid when it is fetched from non-executable region of memory. In an ARM microcontroller with an on-chip MMU, the page fault can also be mapped into the memory management fault.

Bus Fault

A bus fault is an exception that arises due to a memory-related fault for an instruction or data memory transaction, such as a pre-fetch fault or a memory access fault. This fault can be enabled or disabled.

Usage Fault

Exception that occurs due to a fault associated with instruction execution. This includes undefined instruction, illegal unaligned access, invalid state on instruction execution, or an error on exception return may termed as usage fault. An unaligned address of a word or half-word memory access or division by zero can cause a usage fault.

SVCall

A supervisor call (SVC) is an exception that is activated by the SVC instruction. In an operating system, applications can use SVC instructions to contact OS kernel functions and device drivers. This is a software interrupt since it was raised from software, and not from a Hardware or peripheral exception.

PendSV

PendSV is pendable service call and interrupt-driven request for system-level service. PendSV is used for framework switching when no other exception is active. The Interrupt Control and State (INTCTRL) register is used to trigger PendSV. The PendSV is an interrupt and can wait until NVIC has time to service it when other urgent higher priority interrupts are being taken care.

SysTick

A SysTick exception is generated by the system timer when it reaches zero and is enabled to generate an interrupt. The software can also produce a SysTick exception using the Interrupt Control and State (INTCTRL) register.

User Interrupts

This interrupt is an exception signaled either by a peripheral or by software request and fed through the NVIC based on their priority. All interrupts are asynchronous to instruction execution. In the system, peripherals use interrupts to communicate with the processor. An ISR can be also propelled as a result of an event at the peripheral devices. This may include timer timeout or completion of analog-to-digital converter (ADC) conversion. Each peripheral device has a group of special function registers that must be used to access the device for configuration. For a given peripheral interrupt to take effect, the interrupt for that peripheral must be enabled.

Interrupt Programming:

```
/**************************************************************************
Aim: To understand how exceptions/interrupts work
**************************************************************************/
#include <stdint.h>
#include <stdbool.h>
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"
int main(void)
uint32_t ui32Period;
```

```
SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_M
AIN);
// SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,
GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0); TimerConfigure(TIMER0_BASE,
TIMER_CFG_PERIODIC); ui32Period = (SysCtlClockGet() / 10) / 2;
TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period -1);
IntEnable(INT_TIMER0A);
TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
IntMasterEnable();
TimerEnable(TIMER0_BASE, TIMER_A);
while(1)
{
}
}
void Timer0IntHandler(void)
{
// Clear the timer interrupt
// Read the current state of the GPIO pin and
// write back the opposite state
if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_2)) {
GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);
}
Else {
GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4);
}
}
```

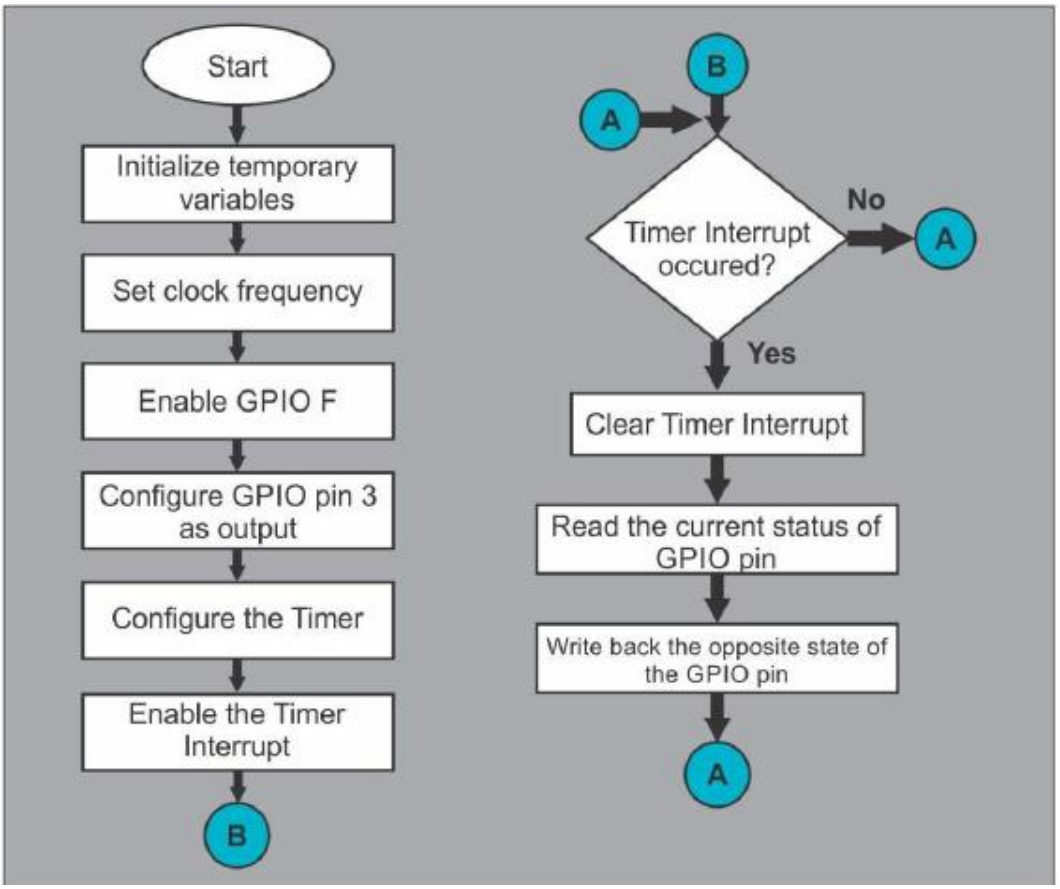

Figure: Flowchart for servicing timer interrupts

In the above program, we have commented following line of the code.
// SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

This means that the code is accessing the peripheral without the peripheral clock being enabled. This should generate an exception. We can see that when this code is executed, upon suspension, execution has trapped inside the "FaultISR()" interrupt routine.

All of the exceptions ISRs are trapped in infinite loop in the provided code. This behavior of the program should be avoided.

If the same code is run, only this time uncommenting un-commenting the "SysCtlPeripheralEnable" while building, then we can see the BLUE LED glow.

Basic Timer:

A standard timer will comprise a pre-scaler, an N-bit timer/counter register, one or more N-bit capture and compare registers. Usually N is 8, 16 or 32 bits. Along with these, there will also be registers for control and status units responsible to configure and monitor the timer.

To count the incoming pulses, an up-counter is deployed as fundamental hardware. A counter can be converted to a timer by fixing incoming pulses and setting a known frequency. Also note that the size in bits of a timer should not be related directly to the size in bits of the CPU architecture. An 8-bit microcontroller can have 16-bit timers (in fact mostly do), and a 32-bit microcontroller can have 16-bit timers (and some do).

Pre-scaler

The pre-scaler takes the basic timer clock frequency as an input and divides it by some value depending upon the circuit requirements before feeding it to the timer, to configure the pre-scaler register(s). This configuration might be limited to a few fixed values (powers of 2), or integers from 1 to 2^m, where m is the number of pre-scaler bits.

Pre-scaler is used to set the clock rate of the timer as per your desire. This provides a flexibility in resolution (high clock rate implies better resolution) and range (high clock rate causes quicker overflow of timer). For instance, we cannot get 1us resolution and a 1sec maximum period using a 16-bit timer. If we want 1us resolution we are restricted to about 65ms maximum period. If we want 1sec maximum period, we are bounded to about 16us resolution. The pre-scaler allows us to manage resolution and maximum period to fit your needs.

Timer Register

The timer register can be defined as hardware with an N-bit up-counter, which has accessibility of read and write command rights for the current count value, and to stop or reset the counter. As discussed, the timer is driven by the pre-scaler output. The regular pulses which drive the timer, irrespective of their source are often called "ticks". We may understand now that it is not necessary for a timer to time in seconds or milliseconds, they do time in ticks. This enables us the elasticity to control the rate of these ticks, depending upon the hardware and software configuration. We may construct our design to some human-

friendly value such as e.g. 1 millisecond or 1 microsecond, or any other design specified units.

Capture Registers

A capture registers are those hardware which can be routinely loaded with the current counter value upon the occurrence of some event, usually a change on an input pin. Therefore the capture register is used to capture a "snapshot" of the timer at the instant when the event occurs. A capture event can also be constructed to produce an interrupt, and the Interrupt Service Routines (ISR) can save or else use the just-captured timer snapshot.

There is no latency problem in snapshot value as the capture occurs in hardware, which would be if the capture was done in software. Capture registers can be used to time intervals between pulses or input signals, to determine the high and low times of input signals.

Compare/Match Registers

Compare or match registers hold a value against which the current timer value is routinely compared and shoots to trigger an event when the value in two registers matches.

- ⇨ If the timer/counter is configured as a timer, we can generate events at known and precise times. Events can be like output pin changes and/or interrupts and/or timer resets.
- ⇨ If the timer/counter is configured as a counter, the compare registers can generate event based on preset counts being achieved.

For instance, the compare registers can be used to generate a timer "tick", a fixed timer interrupt used for system software timing. For example, if a 2ms tick is desired, and the timer is configured with a 0.5us clock, setting a compare register to 4000 will cause a compare event after 2ms. If we set the compare event to generate an interrupt as well as to reset the timer to 0, the result will be an endless stream of 2ms interrupts.

Another notable use of a compare register can be to generate a pulse with variable width. Set an output high/low when the timer is at 0, configure the compare register with value of pulse width, and on the compare event set the output low/high. We may use a second compare register with a larger value, to set the pulse interval by retuning the timer on compare.

Real Time Clock (RTC):

The RTC module is designed to keep wall time. RTC is a mainframe clock that keeps track of the current time. RTCs are present in approximately every electronic device which needs to maintain accurate time. The term RTC came into picture to avoid confusion with regular hardware clocks which are merely signals that administer digital electronics, and do not count time in human units.

Benefits of using RTC:

- ⇨ Low power consumption
- ⇨ Liberates the main system for time-critical tasks
- ⇨ Increases accuracy if compared to other methods

A GPS receiver can cut down its startup time by comparing the current time as per its RTC, with the moment of last valid signal. If it has been less than a few hours, then the previous ephemeris is still usable.

With the option of alternative power source with RTCs, they can continue to keep time while the primary power source being unavailable. This alternate source may be a lithium battery or a super capacitor.

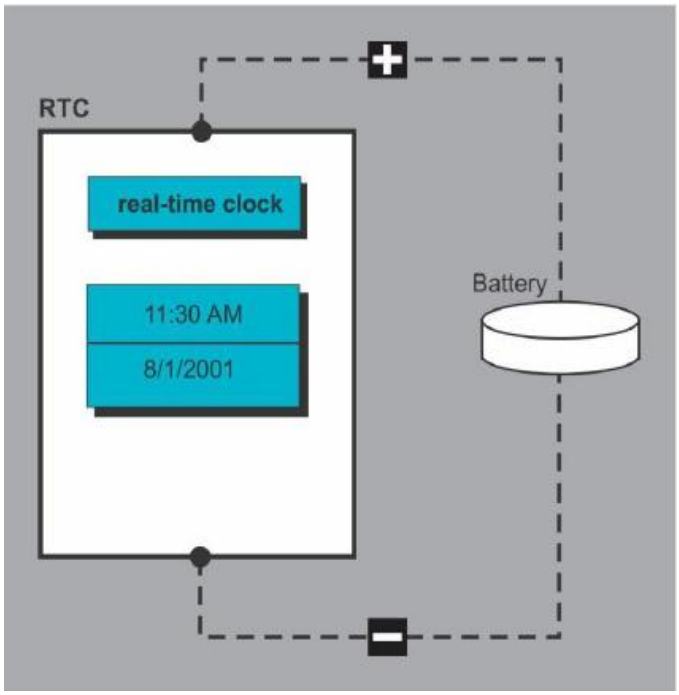

Figure: Real Time Clock with External Power Source

Motion Control Peripherals – PWM Module & Quadrature Encoder Interface (QEI):

PWM Module:

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

PWM using TIVA TM4C123HG6PM

TM4C123GH6PM PWM module provides a great deal of flexibility and can generate simple PWM signals, such as those required by a simple charge pump as well as paired PWM signals with deadband delays, such as those required by a half-H bridge driver. Three generator blocks can also generate the full six channels of gate controls required by a 3-phase inverter bridge.

Each PWM generator block has the following features:

- ⇨ One fault-condition handling inputs to quickly provide low-latency shutdown and prevent damage to the motor being controlled, for a total of two inputs

⇨ One 16-bit counter
- Runs in Down or Up/Down mode
- Output frequency controlled by a 16-bit load value
- Load value updates can be synchronized
- Produces output signals at zero and load value

⇨ Two PWM comparators
- Comparator value updates can be synchronized
- Produces output signals on match

⇨ PWM signal generator
- Output PWM signal is constructed based on actions taken as a result of the counter and PWM comparator output signals
- Produces two independent PWM signals

⇨ Dead-band generator
- Produces two PWM signals with programmable dead-band delays suitable for driving a half-H bridge.
- Can be bypassed, leaving input PWM signals unmodified.

⇨ Can initiate an ADC sample sequence

The control block determines the polarity of the PWM signals and which signals are passed through to the pins. The output of the PWM generation blocks are managed by the output control block before being passed to the device pins.

Block Diagram:

TM4C123GH6PM controller contains two PWM modules, each with four generator blocks that generate eight independent PWM signals or four paired PWM signals with dead-band delays inserted.

TM4C123GH6PM controller contains two PWM modules, each with four generator blocks that generate eight independent PWM signals or four paired PWM signals with dead-band delays inserted.

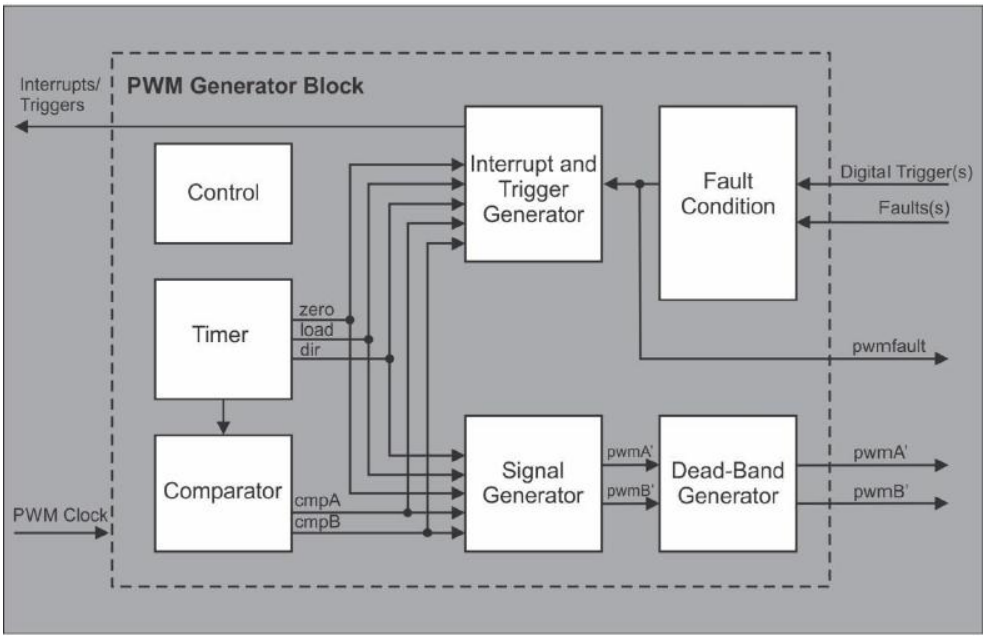

Figure: PWM Module Block Diagram

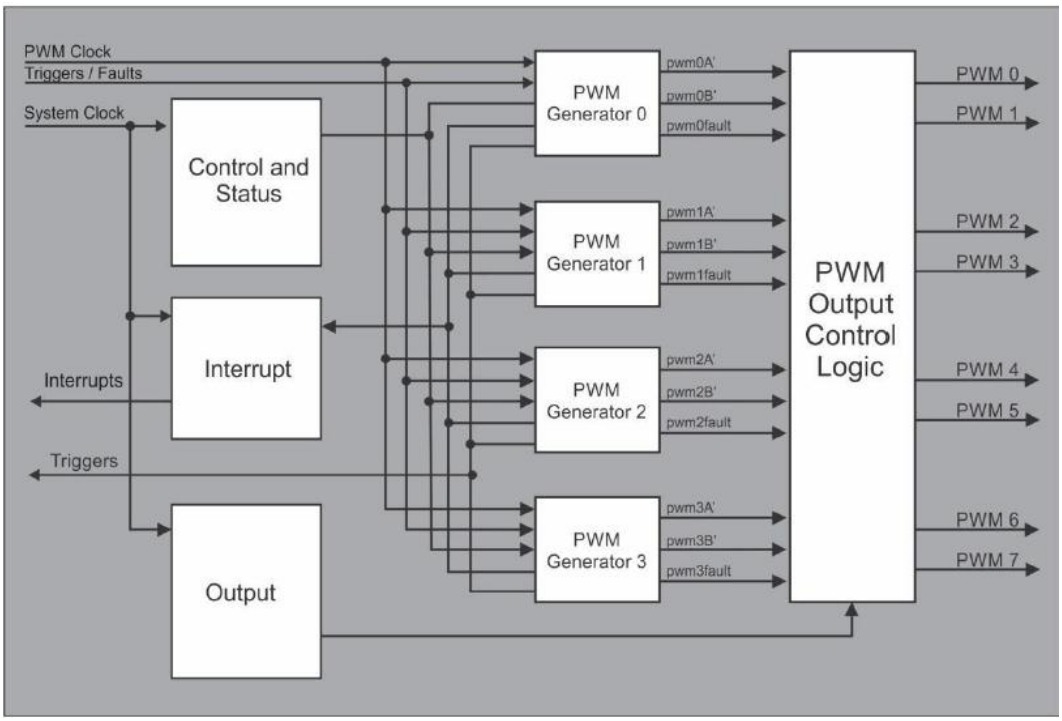

Figure: PWM Generator Block Diagram

Functional Description:

Clock Configuration

The PWM has two clock source options:

- The System Clock
- A pre divided System Clock

The clock source is selected by programming the USPWMDIV bit in the Run-Mode Clock Configuration (RCC) register. The PWMDIV bit field specifies the divisor of the system clock that is used to create the PWM Clock.

PWM Timer

The timer in each PWM generator runs in one of two modes: Count-Down mode or Count-Up/Down mode. In Count-Down mode, the timer counts from the load value to zero, goes back to the load value, and continues counting down. In Count-Up/Down mode, the timer counts from zero up to the load value, back down to zero, back up to the load value, and so on. Generally, Count-Down mode is used for generating left- or right-aligned PWM signals, while the Count-Up/Down mode is used for generating center-aligned PWM signals. The timers output three signals that are used in the PWM generation process: the direction signal (this is always Low in Count-Down mode, but alternates between low and high in Count-Up/Down mode), a single-clock-cycle-width High pulse when the counter is zero, and a single-clock-cycle-width High pulse when the counter is equal to the load value. Note that in Count-Down mode, the zero pulse is immediately followed by the load pulse. In the figures in this chapter, these signals are labelled "dir," "zero," and "load."

PWM Comparators:

Each PWM generator has two comparators that monitor the value of the counter, when either comparator matches the counter, they output a single-clock-cycle-width High pulse, labeled "cmpA" and "cmpB" in the figures in this chapter. When in Count-Up/Down

mode, these comparators match both when counting up and when counting down, and thus are qualified by the counter direction signal. These qualified pulses are used in the PWM generation process. If either comparator match value is greater than the counter load value, then that comparator never outputs a High pulse.

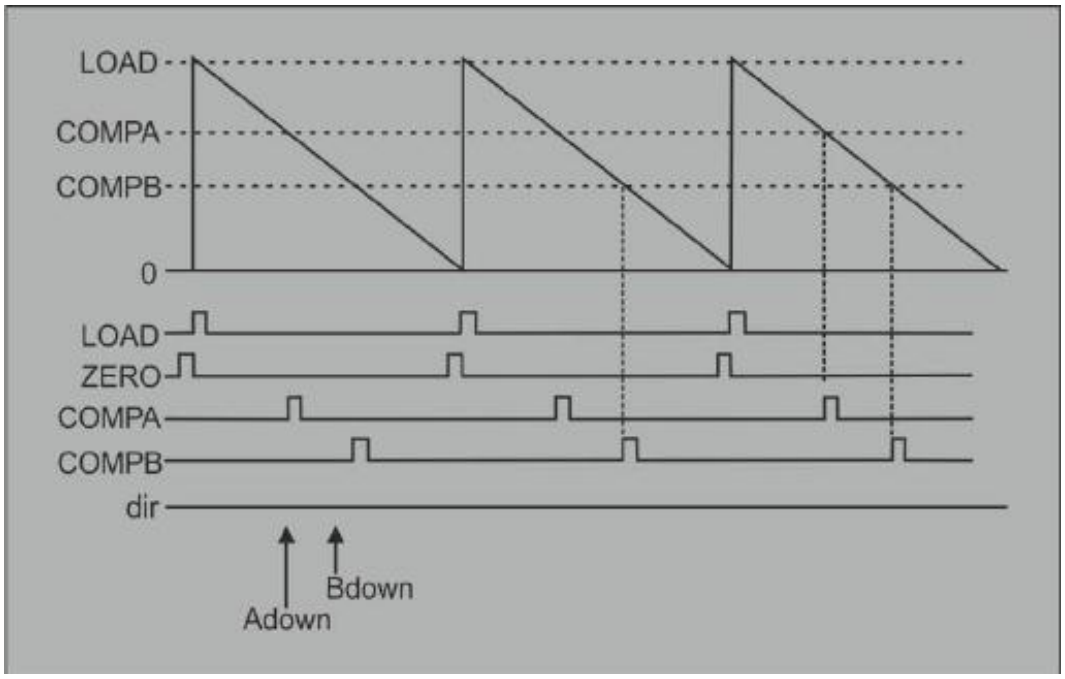

Figure: PWM Count-Down Mode

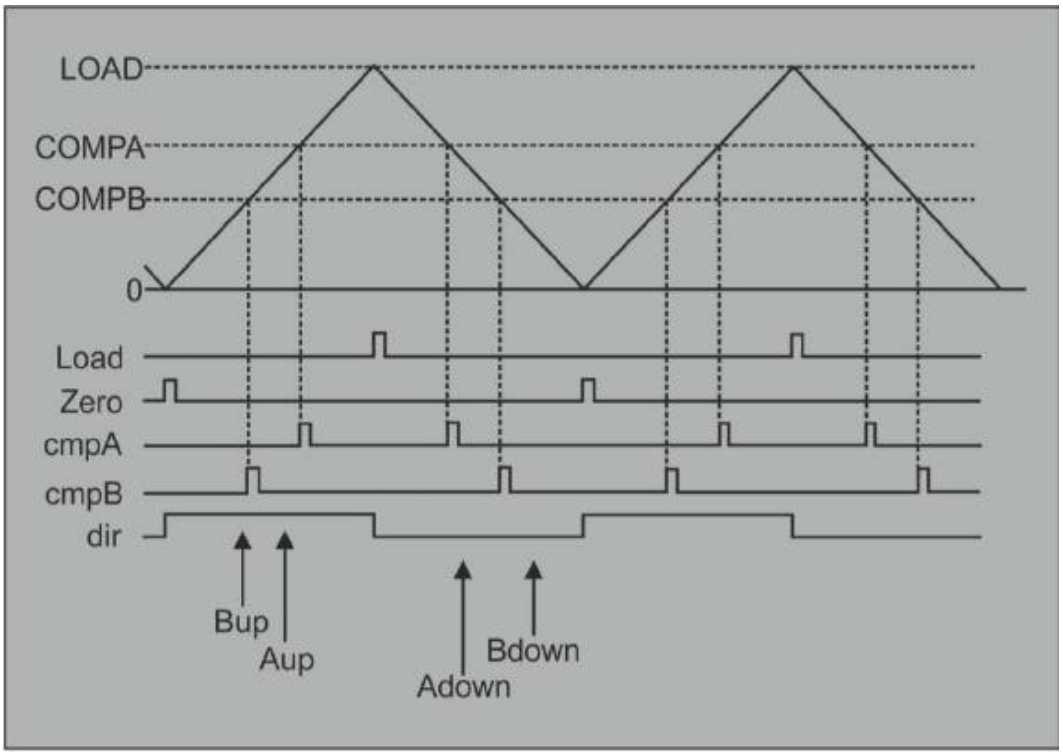

Figure: PWM Count- Up/Down Mode

PWM Signal Generator:

Each PWM generator takes the load, zero, cmpA, and cmpB pulses (qualified by the dir signal) and generates two internal PWM signals, pwmA and pwmB. In Count-Down mode, there are four events that can affect these signals: zero, load, match A down, and match B down. In Count-Up/Down mode, there are six events that can affect these signals: zero, load, match A down, match A up, match B down, and match B up. The match A or match B events are ignored when they coincide with the zero or load events. If the match A and match B events coincide, the first signal, pwmA, is generated based only on the match A event, and the second signal, pwmB, is generated based only on the match B event.

Dead-Band Generator:

The pwmA and pwmB signals produced by each PWM generator are passed to the dead-band generator. If the dead-band generator is disabled, the PWM signals simply pass through to the pwmA' and pwmB' signals unmodified. If the dead-band generator is enabled, the pwmB signal is lost and two PWM signals are generated based on the pwmA signal. The first output PWM signal, pwmA' is the pwmA signal with the rising edge delayed by a programmable amount. The second output PWM signal, pwmB', is the inversion of the pwmA signal with a programmable delay added between the falling edge of the pwmA signal and the rising edge of the pwmB' signal. The resulting signals are a pair of active high signals where one is always high, except for a programmable amount of time at transitions where both are low. These signals are therefore suitable for driving a half-H bridge, with the dead-band delays preventing shoot-through current from damaging the power electronics.

Quadrature Encoder Interface (QEI):

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

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

Interfacing QEI using Tiva TM4C123GH6PM

The TM4C123GH6PM microcontroller includes two quadrature encoder interface (QEI) modules. Each QEI module interprets the code produced by a quadrature encoder wheel to integrate position over time and determine direction of rotation. In addition, it can capture a running estimate of the velocity of the encoder wheel.

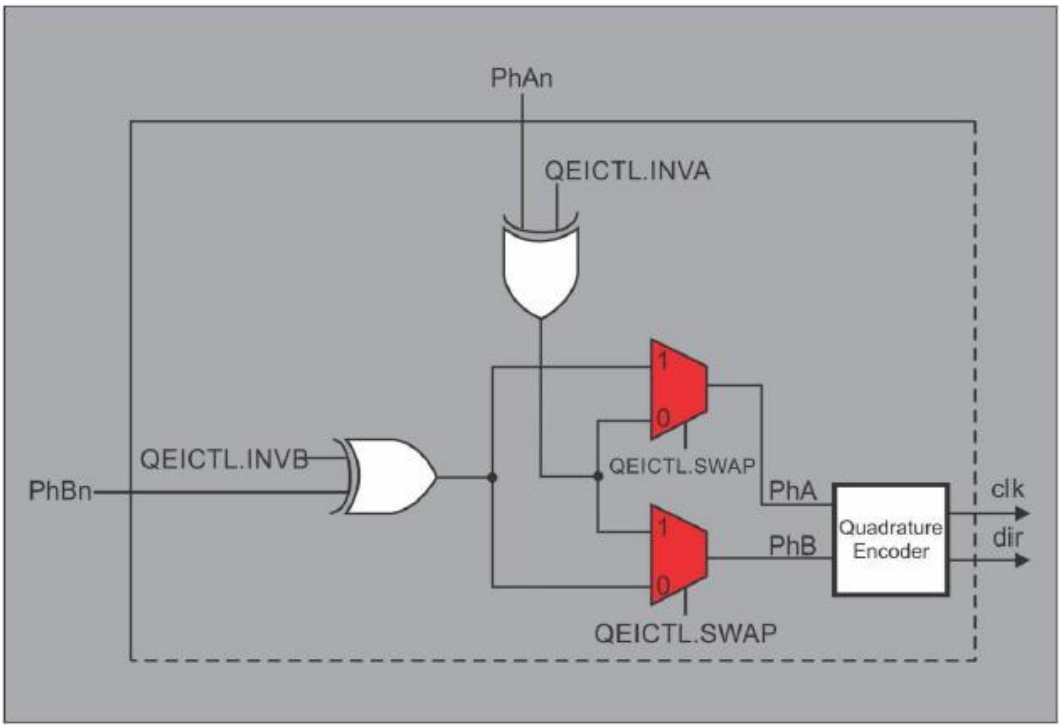

Figure: QEI Input Signal Logic

The TM4C123GH6PM microcontroller includes two QEI modules providing control of two motors at the same time with the following features:

- ⇨ Position integrator that tracks the encoder position
- ⇨ Programmable noise filter on the inputs
- ⇨ Velocity capture using built-in timer
- ⇨ The input frequency of the QEI inputs may be as high as 1/4 of the processor frequency (for example, 12.5 MHz for a 50-MHz system)
- ⇨ Interrupt generation on:
  - Index pulse
  - Velocity-timer expiration
  - Direction change
  - Quadrature error detection

Functional Description:

The QEI module interprets the two-bit gray code produced by a quadrature encoder wheel to integrate position over time and determine direction of rotation. In addition, it can capture a running estimate of the velocity of the encoder wheel. The position integrator and velocity capture can be independently enabled, though the position integrator must be enabled before the velocity capture can be enabled. The two phase signals, **PhAn** and **PhBn**, can be swapped before being interpreted by the QEI module to change the meaning of forward and backward and to correct for misfiring of the system. Alternatively, the phase signals can be interpreted as a clock and direction signal as output by some encoders.

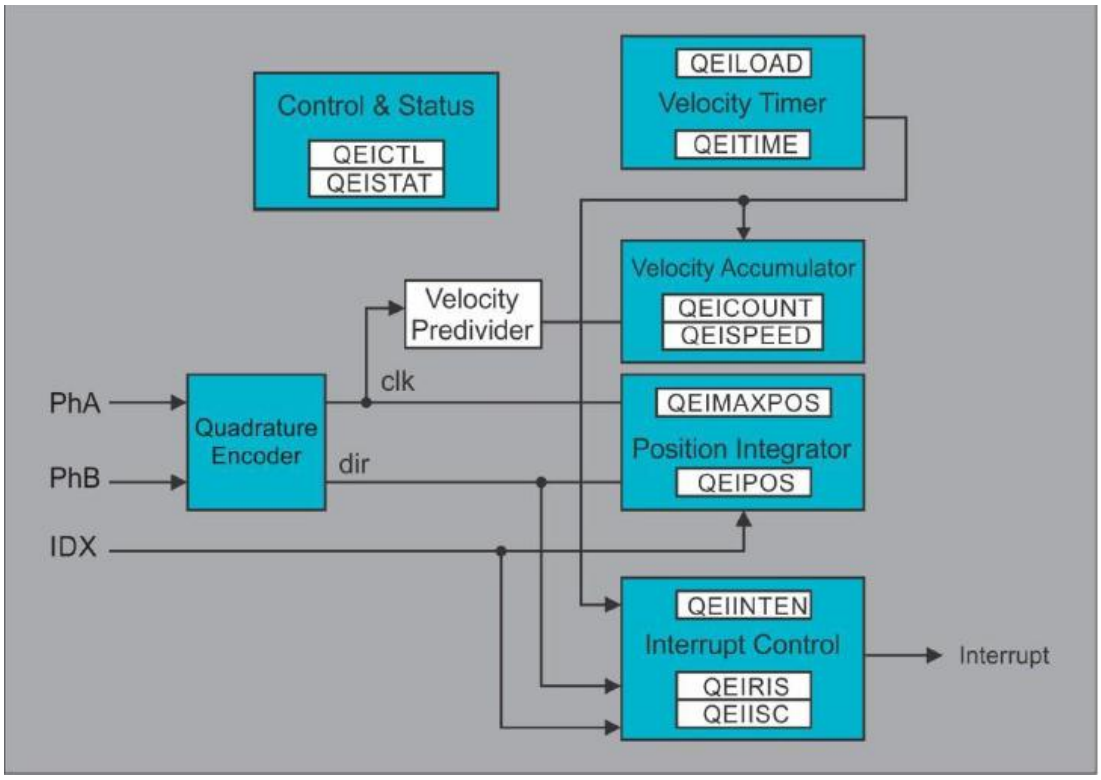

Figure: QEI Block Diagram

The QEI module input signals have a digital noise filter on them that can be enabled to prevent spurious operation. The noise filter requires that the inputs be stable for a specified number of consecutive clock cycles before updating the edge detector. The filter is enabled by the **FILTEN** bit in the **QEI Control (QEICTL)** register. The frequency of the input update is programmable using the
FILTCNT bit field in the **QEICTL** register.

The QEI module supports two modes of signal operation:

- **Quadrature phase mode**, the encoder produces two clocks that are 90 degrees out of phase, the edge relationship is used to determine the direction of rotation.
- Clock/direction mode, the encoder produces a clock signal to indicate steps and a direction signal to indicate the direction of rotation. This mode is determined by the SIGMODE bit of the QEICTL register.

When the QEI module is set to use the quadrature phase mode (SIGMODE bit is clear), the capture mode for the position integrator can be set to update the position counter on every edge of the PhA signal or to update on every edge of both PhA and PhB. Updating the position counter on every PhA and PhB edge provides more positional resolution at the cost of less range in the positional counter. When edges on PhA lead edges on PhB, the position counter is incremented. When edges on PhB lead edges on PhA, the position counter is decremented. When a rising and falling edge pair is seen on one of the phases without any edges on the other, the direction of rotation has changed.

The positional counter is automatically reset on one of two conditions:

- Sensing the index pulse or
- Reaching the maximum position value.

The reset mode is determined by the RESMODE bit of the QEICTL register.

- When RESMODE is set, the positional counter is reset when the index pulse is sensed. This mode limits the positional counter to the values [0: N-1], where N is the number of phase edges in a full revolution of the encoder wheel. The QEI Maximum Position (QEIMAXPOS) register must be programmed with N-1 so that the reverse direction from position 0 can move the position counter to N-1. In this mode, the position register contains the absolute position of the encoder relative to the index (or home) position once an index pulse has been seen.
- When RESMODE is clear, the positional counter is constrained to the range [0: M], where M is the programmable maximum value. The index pulse is ignored by the positional counter in this mode. Velocity capture uses a configurable timer and a count register. The timer counts the number of phase edges (using the same configuration as for the position integrator) in a given time period.

The edge count from the previous time period is available to the controller via the QEI Velocity (QEISPEED) register, while the edge count for the current time period is being accumulated in the QEI Velocity Counter (QEICOUNT) register. As soon as the current time period is complete, the total number of edges counted in that time period is made available in the QEISPEED register (overwriting the previous value), the QEICOUNT register is cleared, and counting commences on a new time period. The number of edges counted in a given time period is directly proportional to the velocity of the encoder.