**UNIT\_4**

Input/Output ports of LPC2148:

* LPC2148 is a 32 -bit microcontroller based on ARM7 family (ARM7TDMI-S to be specific) and is available in a 64 pin LQFP Package (Low Profile Quad Flat Package).
* It has an on-chip static RAM of 32kB and an on-chip flash memory of 512kB.
* LPC2148 has an on-chip boot loader software that can be used to program the microcontroller using the feature of In-System Programming.
* It also has the feature of in-Application Programming using which the user can program the microcontroller without using the boot loader software.
* It has an embedded in-circuit emulator that helps in real time debugging.
* LPC2148 has two 10-bit ADCs (ADC0 and ADC1) of Successive Approximation type.
* It also has a 10-bit DAC of Resistor String type.

General Purpose Input Output (GPIO) pins of a microcontroller are the first thing we need to learn before starting its embedded programming as input/output pins are the only way to interface with the microcontroller. GPIO pins can be used for driving loads, reading [digital and analog](https://www.electronicshub.org/analog-circuits-and-digital-circuits/) signal, controlling external components, generating triggers for external devices etc.

LPC2148 has two IO ports namely PORT0 (P0) and PORT1 (P1). These two IO ports are of 32-bit wide and are provided by the 64 pins of the microcontroller.

The naming convention of the I/O pins on the LPC2148 Microcontroller is Pa.bc where ‘a’ is the number of the port i.e. 0 or 1 (as LPC2148 has only two ports) and ‘bc’ is the number of the pin in the port a. For example, P0.1 indicates pin number 1 of PORT0 and P1.10 indicates pin number 10 of PORT1.

PORT0 is a 32-bit wide input/output port with dedicated direction control bits for each physical pin. Out of the 32 pins, 28 pins can be used as general purpose bidirectional I/O pins. Pin P0.31 is an output only pin. Pins P0.24, P0.26 and P0.27 of PORT0 are not available.

PORT1 is also a 32-bit wide input/output port. In PORT1, pins P1.0 to P1.15 are not available and pins P1.16 to P1.31 are the available general purpose input/output pins.

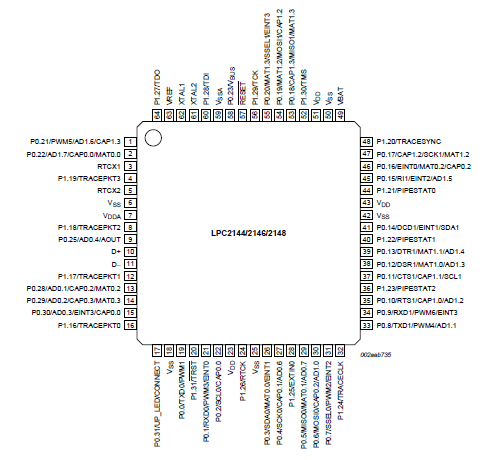
Most of the pins in both the I/O ports of the LPC2148 have more than one function i.e. they are multiplexed with different functions. For example, Pin 19 of the LPC2148 has three functions namely P0.0, a general purpose I/O pin, TXD0, the transmitter O/P for UART0 and PWM1, the pulse width modulator O/P 1.

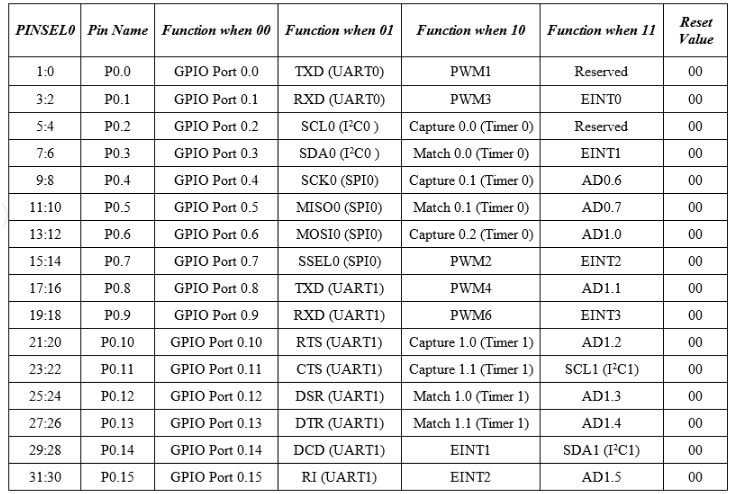
At any point of operation, each pin can have a single function and the function can be selected with the help of three Configuration Registers which control the multiplexers to allow connection between the external pin and the on-chip peripheral.

The configuration register is called PINSEL and is classified in to three registers: PINSEL0, PINSEL1 and PINSEL2. These configuration registers are of 32-bit wide. Any pin on the LPC2148 can have a maximum of 4 functions. Hence in order to select one of the four functions, two corresponding bits of the PINSEL register are needed. So, a 32-bit PINSEL register can control 16 pins with 2-bits to control each pin.

PINSEL0 controls PORT0 pins P0.0 to P0.15, PINSEL1 controls PORT0 pins P0.16 to P0.31 and PINSEL2 controls PORT1 pins P1.16 to P1.31.

The following table shows the PINSEL0 and corresponding functions on the PORT0.





The default function of all the Pins is GPIO. But it is a good programming practice to mention “PINSEL0=0” in order to select the GPIO function of the Pins.

GPIO function is the most frequently used functionality of the microcontroller. The GPIO function in both the Ports are controlled by a set of 4 registers: IOPIN, IODIR, IOSET and IOCLR.

**IOPIN**: It is a GPIO Port Pin Value register and can be used to read or write values directly to the pin. The status of the Pins that are configured as GPIO can always be read from this register irrespective of the direction set on the pin (Input or Output).

The syntax for this register is IOxPIN, where ‘x’ is the port number i.e. IO0PIN for PORT0 and IO1PIN for PORT1.

**IODIR**: It is a GPIO Port Direction Control register and is used to set the direction i.e. either input or output of individual pins. When a bit in this register is set to ‘0’, the corresponding pin in the microcontroller is configured as Input. Similarly, when a bit is set as ‘1’, the corresponding pin is configured as Output.

The syntax for this register is IODIRx, where ‘x’ is the port number i.e. IODIR0 for PORT0 and IODIR1 for PORT1.

**IOSET**: It is a GPIO Port Output Set Register and can be used to set the value of a GPIO pin that is configured as output to High (Logic 1). When a bit in the IOSET register is set to ‘1’, the corresponding pin is set to Logic 1. Setting a bit ‘0’ in this register has no effect on the pin.

The syntax for this register is IOSETx, where ‘x’ is the port number i.e. IO0SET for PORT0 and IO1SET for PORT1.

**IOCLR**: It is a GPIO Port Output Clear Register and can be used to set the value of a GPIO pin that is configured as output to Low (Logic 0). When a bit in the IOCLR register is set to ‘1’, the corresponding pin in the respective Port is set to Logic 0 and at the same time clears the corresponding bit in the IOSET register. Setting ‘0’ in the IOCLR has no effect on the pin.

The syntax for this register is IOCLRx, where ‘x’ is the port number i.e. IO0CLR for PORT0 and IO1CLR for PORT1.

An important note to remember is that since the LPC2148 is a 32-bit microcontroller, the length of all the registers mentioned is also 32-bits. Each bit in the above mentioned registers is directly linked to the corresponding pin in the microcontroller i.e. bit ‘a’ in IOSET0 corresponds to Pin ‘a’ in the PORT0.

Registers in LPC2148 follow Big Endian format i.e. bit 0 is the LSB on the extreme right of the register and bit 31 is the MSB on the extreme left of the register.

Another important note is that when reset, all the pins are set as GPIO pins and the direction of each pin is set as Input.

Now, we’ll see how to use the above mentioned registers in programming. First, is to set the direction of a pin. For example, if we want to set 4th pin of PORT0 i.e. P0.3 as output, then it can be set in various ways as shown below.

**Method 1: IO0DIR = (1<<3);**

This is a direct assignment method where the binary value (1) is set directly on the pin. All the other pins are set to 0. This method should be avoided as the value is directly being assigned in the register and while P0.3 is assigned as ‘1’, all the other pins are forced to be assigned ‘0’.

An alternative to this method is ORing the register and then assigning the value. This can be done in two ways.

**Method 2: IO0DIR | = 0x00000008;**

In this method, the hexadecimal value of the register is assigned after ORing the register with itself. In this way, the other pins other than the desired pin (P0.3 in this case) are not affected. This method is useful if we want to assign many pins without affecting the other pins.

**Method 3: IO0DIR | = (1<<3);**

This is similar to the above method except that only a single pin is affected.

Other registers can also be set using the same methods. Now, we’ll see an example of setting pin 15 of PORT0 i.e. P0.15 as output and drive the pin High.

For this we need to use two registers: IODIR and IOSET.

**IO0DIR | = (1<<15);** // Configuring P0.15 as output.

**IO0SET | = (1<<15);** // Make the O/P pin P0.15 as High

We’ll see another example where pin 11 of PORT0 is set as output and the output of this pin is set to logic 1 and then to logic 0.

For this we need to use three registers: IODIR, IOSET and IOCLR.

**IO0DIR | = (1<<11);** // Configuring P0.11 as output.

**IO0SET | = (1<<11);** // Make the O/P pin P0.11 as High.

**IO0CLR | = (1<<11);** // Make the O/P pin P0.11 as Low.

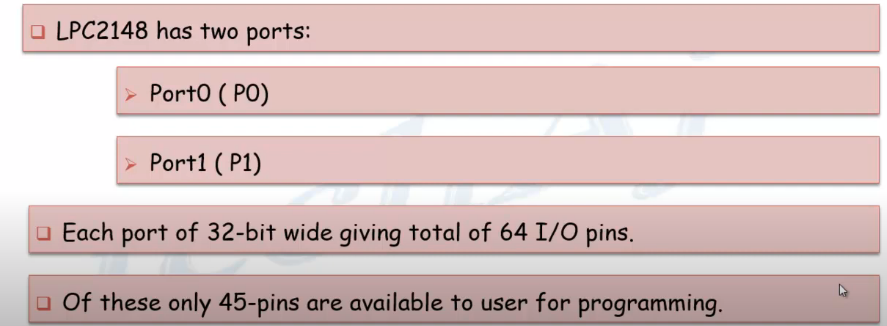
Now, we’ll see an example where more than one pin has to be set as output and the value of that pin must be HIGH. Consider pins 7 and 14 of PORT0 (P0.7 and P0.14).

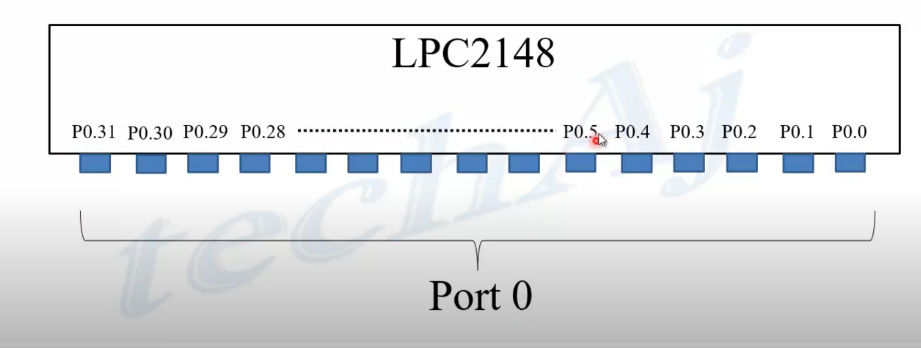
**IO0DIR | = (1<<7) | (1<<14);** // Configuring pins P0.7 and P0.14 as output.

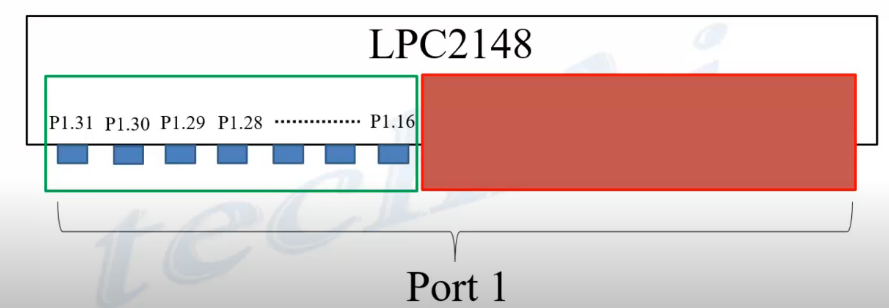
**IO0SET | = (1<<7) | (1<<14);** // Make the O/P of pins P0.7 and P0.14 as High.

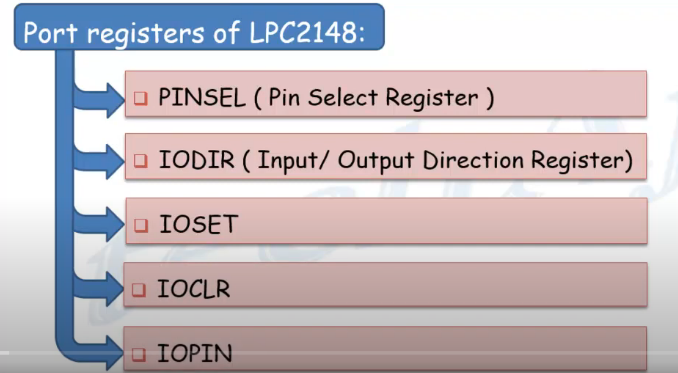
Once we have seen how to set the direction of pins, setting the pin as High or Low, now we’ll jump into real time embedded programming. Similar to “Hello, World” program in C Language, blinking an LED is the basic program in embedded system.

The IDE we are going to use for the development of LPC2148 based projects is Keil µVision. All the registers for LPC214X series of microcontrollers are defined in the “lpc214x.h” header file.

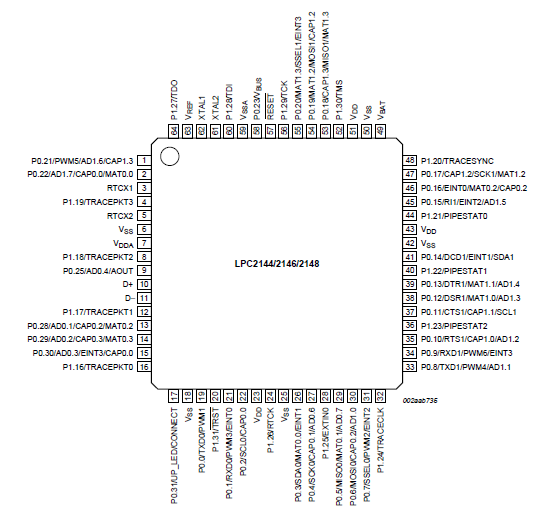
**LPC 2148 I/O PORTS**



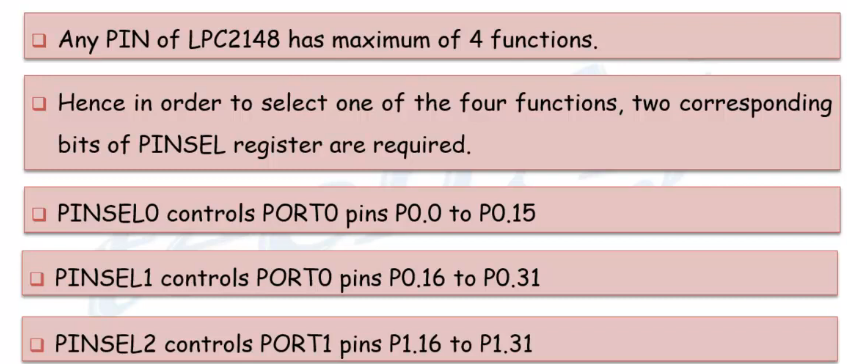




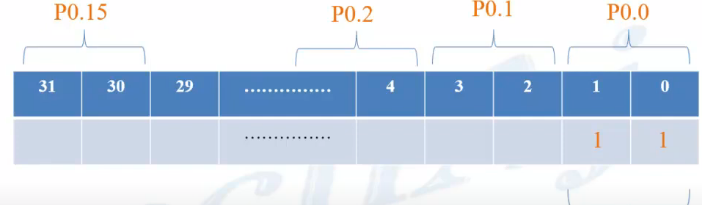












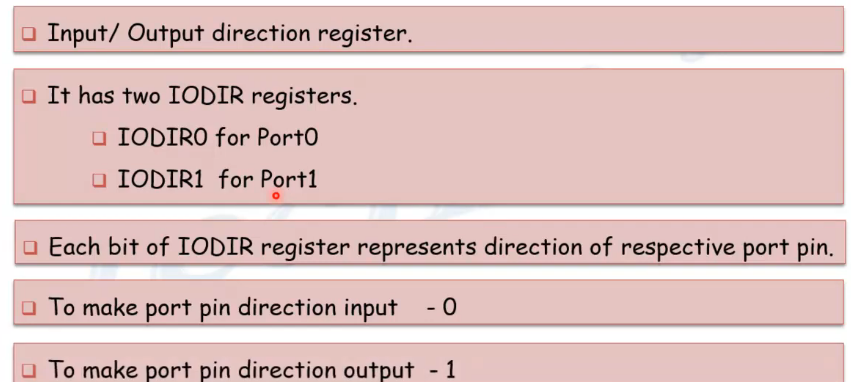


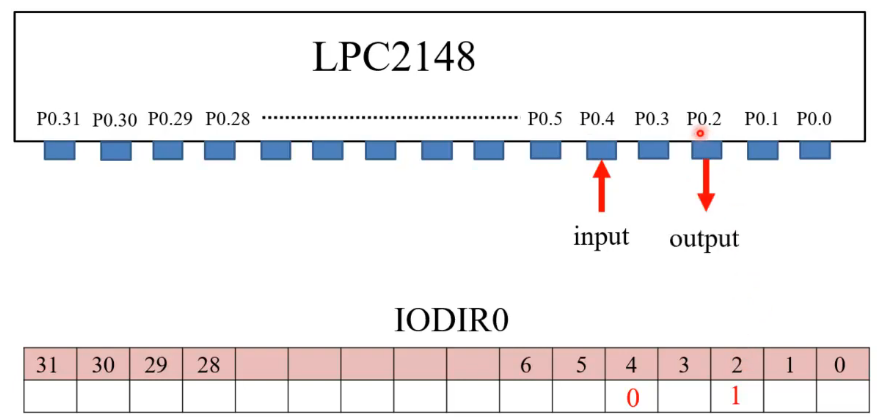




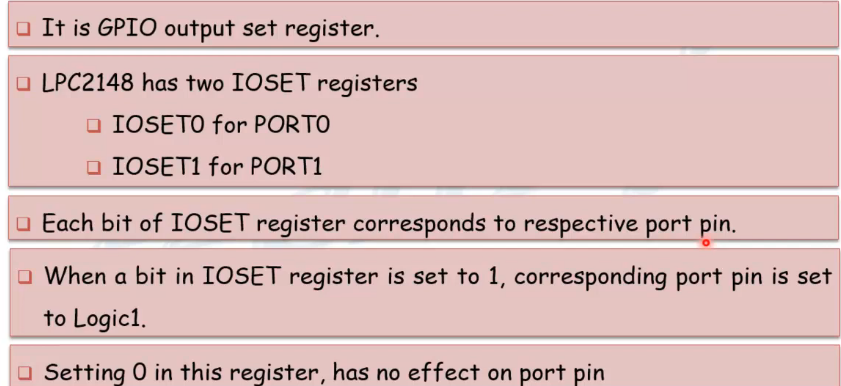


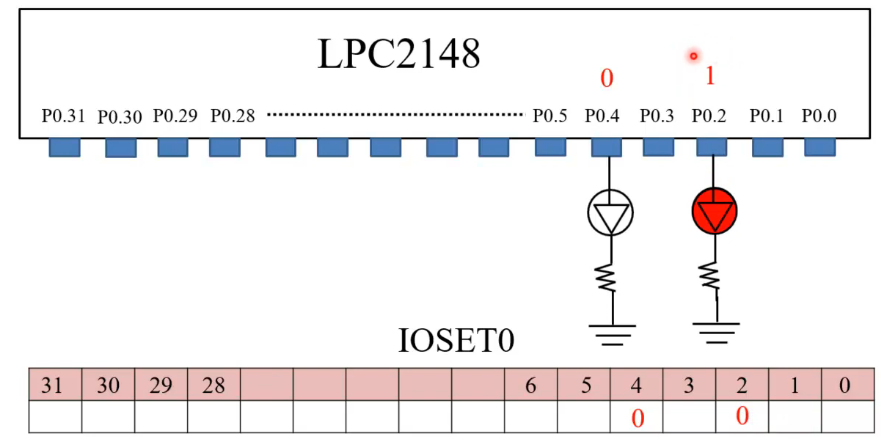


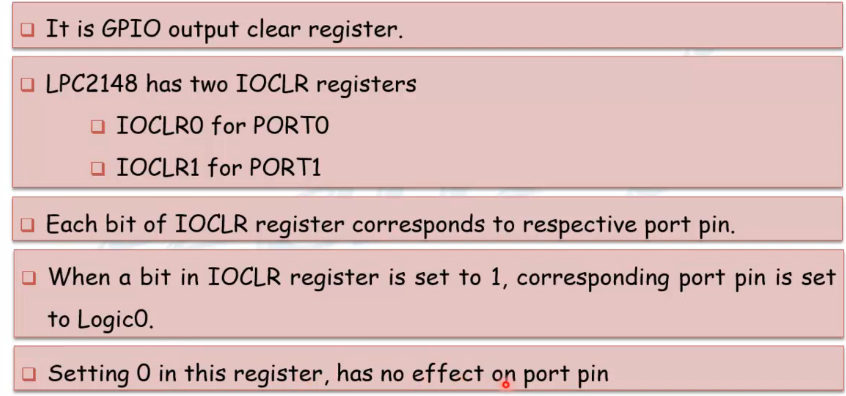


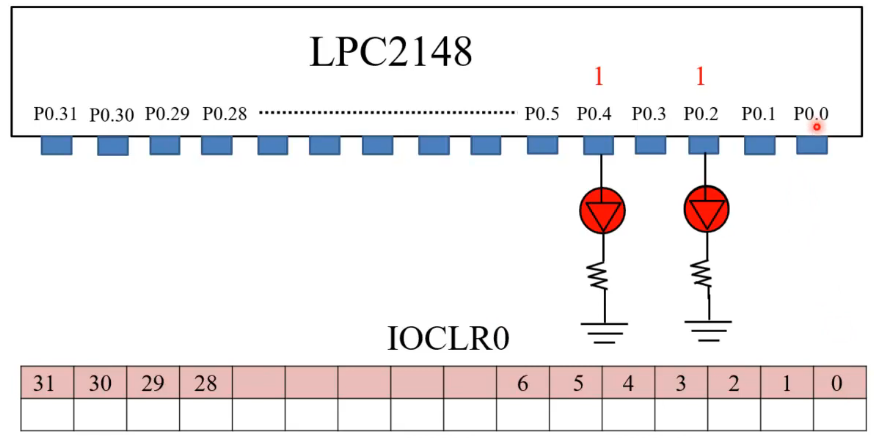


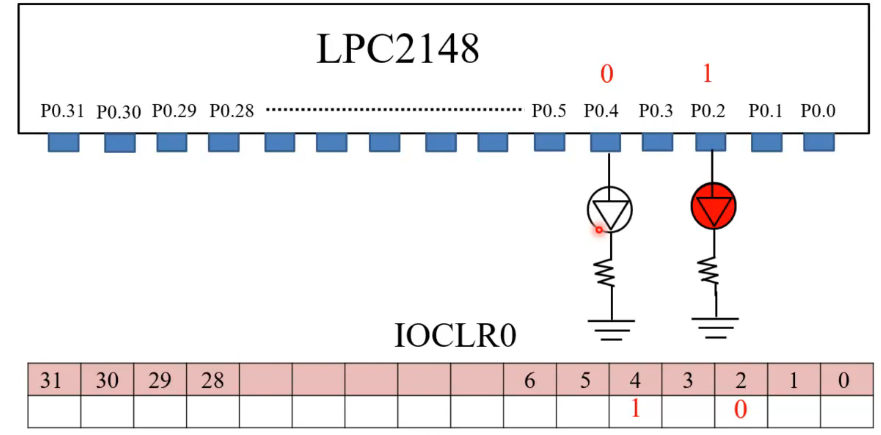




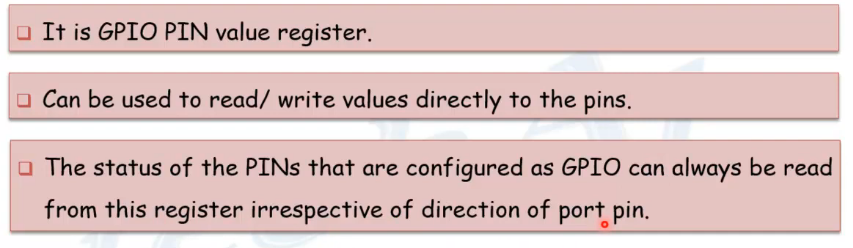


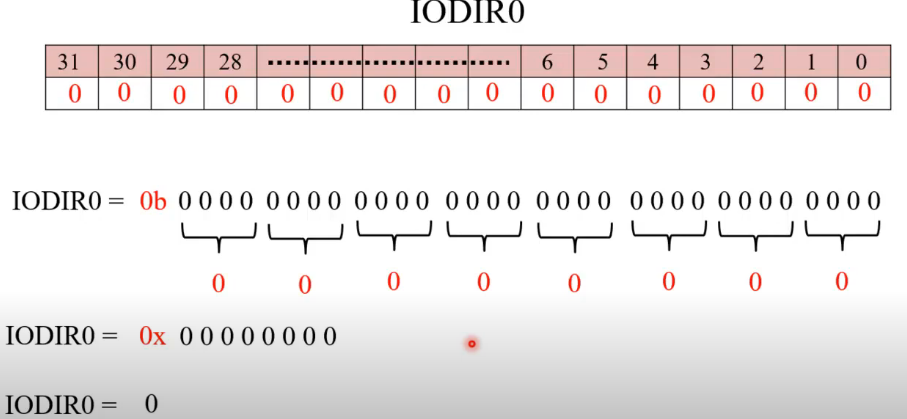


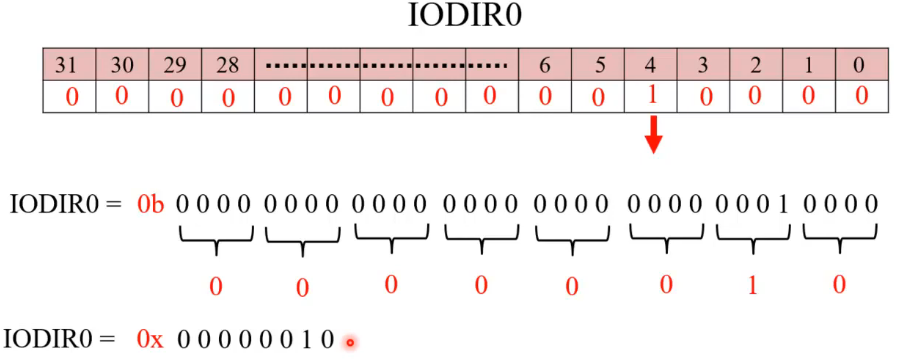


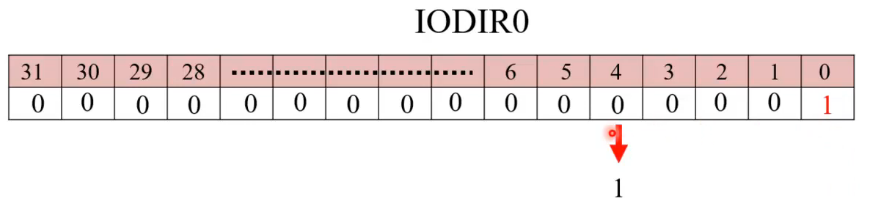


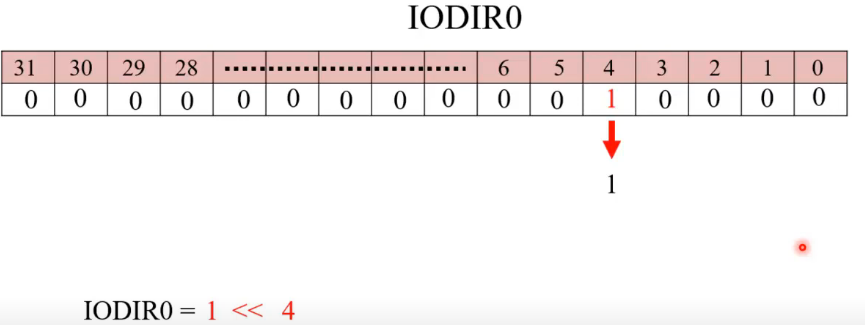


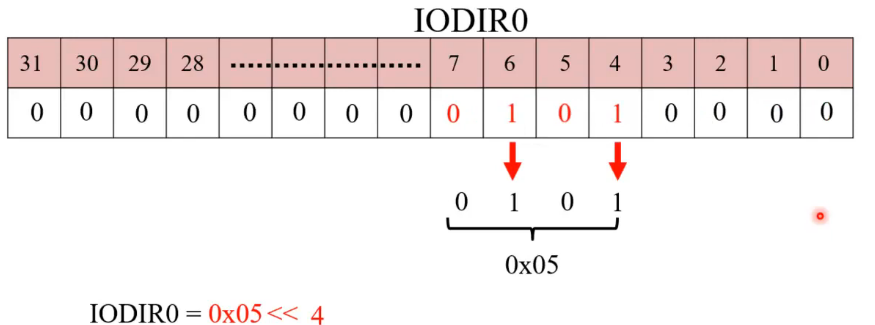












**LPC2148 TIMERS**

Features

Timer/Counter0 and Timer/Counter1 are functionally identical except for the peripheral base address.

• A 32-bit Timer/Counter with a programmable 32-bit Prescaler.

• Counter or Timer operation

• Up to four 32-bit capture channels per timer, that can take a snapshot of the timer value when an input signal transitions. A capture event may also optionally generate an interrupt.

• Four 32-bit match registers that allow:

–Continuous operation with optional interrupt generation on match.

– Stop timer on match with optional interrupt generation.

– Reset timer on match with optional interrupt generation.

• Up to four external outputs corresponding to match registers, with the following capabilities:

– Set low on match.

– Set high on match.

– Toggle on match.

– Do nothing on match.

Applications

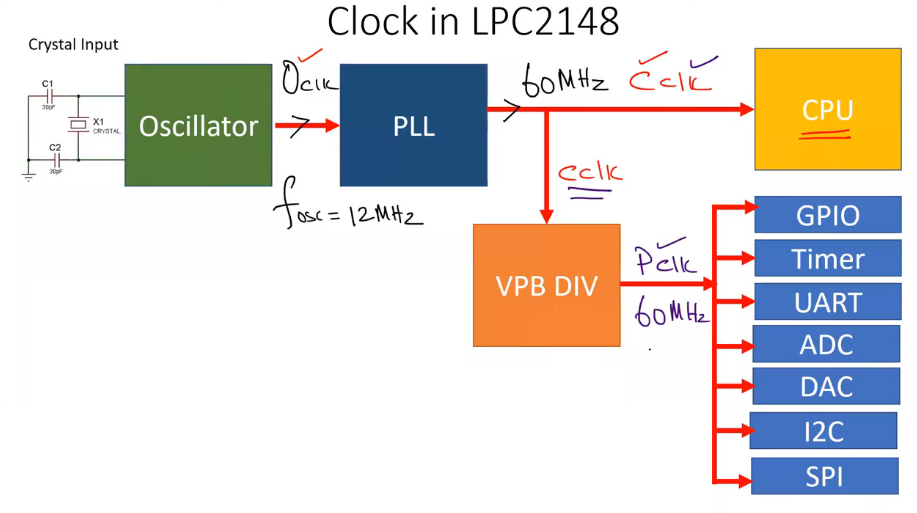
•Interval Timer for counting internal events.

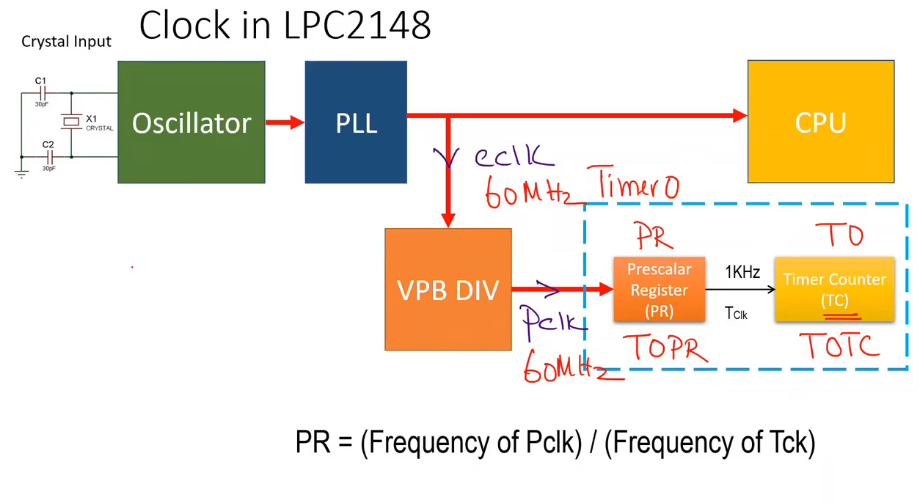
• Pulse Width Demodulator via Capture inputs.

• Free running timer.

Description

The Timer/Counter is designed to count cycles of the peripheral clock (PCLK) or an externally-supplied clock, and can optionally generate interrupts or perform other actions at specified timer values, based on four match registers. It also includes four capture inputs to trap the timer value when an input signal transitions, optionally generating an interrupt.

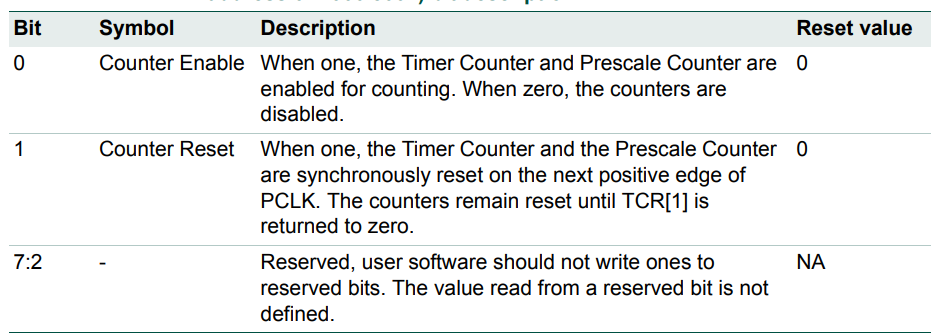




**Register description**

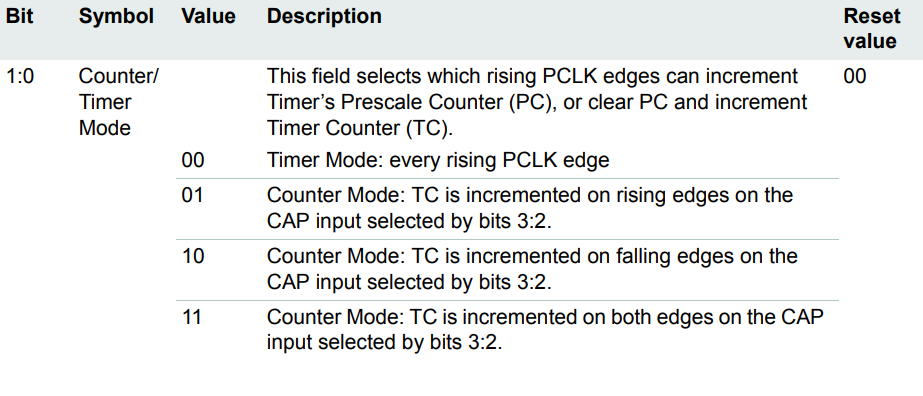
Each Timer/Counter contains the following registers

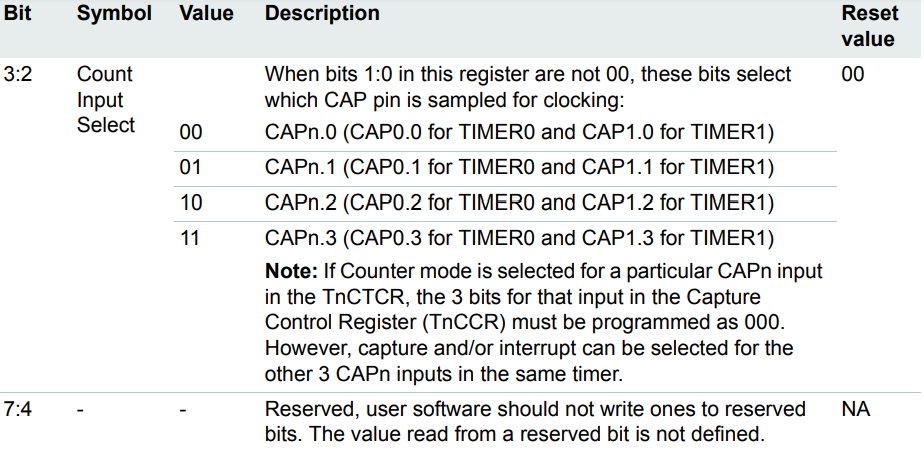
1)Timer Control Register (TCR):Timer Control register used to control the timer control functions. We’ll enable, disable and reset Timer Counter (TC) through this register



2) Count Control Register (CTCR):

The Count Control Register (CTCR) is used to select between Timer and Counter mode, and in Counter mode to select the pin and edge(s) for counting.





3) Timer Counter (TC)

The 32-bit Timer Counter is incremented when the Prescale Counter reaches its terminal count. Unless it is reset before reaching its upper limit, the TC will count up through the value 0xFFFF FFFF and then wrap back to the value 0x0000 0000. This event does not cause an interrupt, but a Match register can be used to detect an overflow if needed.

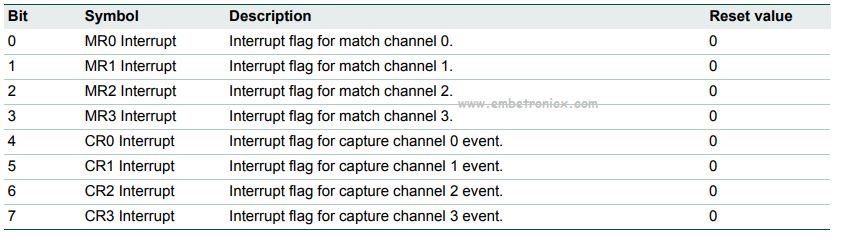
4). Prescale Register (PR)

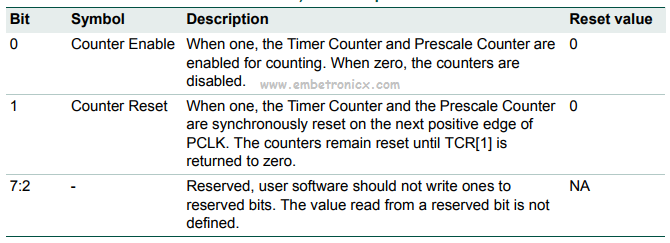
The 32-bit Prescale Register specifies the maximum value for the Prescale Counter.

5) Prescale Counter Register (PC)

The 32-bit Prescale Counter controls division of PCLK by some constant value before it is applied to the Timer Counter. This allows control of the relationship of the resolution of the timer versus the maximum time before the timer overflows. The Prescale Counter is incremented on every PCLK. When it reaches the value stored in the Prescale Register, the Timer Counter is incremented and the Prescale Counter is reset on the next PCLK. This causes the TC to increment on every PCLK when PR = 0, every 2 PCLKs when PR = 1, etc.

**Interrupt Register (IR):** Interrupt Register consists of flag bits for Match Interrupts and Capture Interrupts. It contains four bits each for match and capture interrupts. Bits 0 to 3 in the IR register are for Match Register Interrupts i.e. Bit 0 for MR0 interrupt, Bit 1 for MR1 interrupt, etc. Bits 4 to 7 are for Capture Register Interrupts Bit 4 for CR0 interrupt, Bit 5 for CR1 interrupts, etc. If an interrupt is triggered, then the corresponding bit in the IR register is set to 1. Manually writing a logic 1 on the bit of the IR register will reset the interrupt. Writing a zero has no effect.





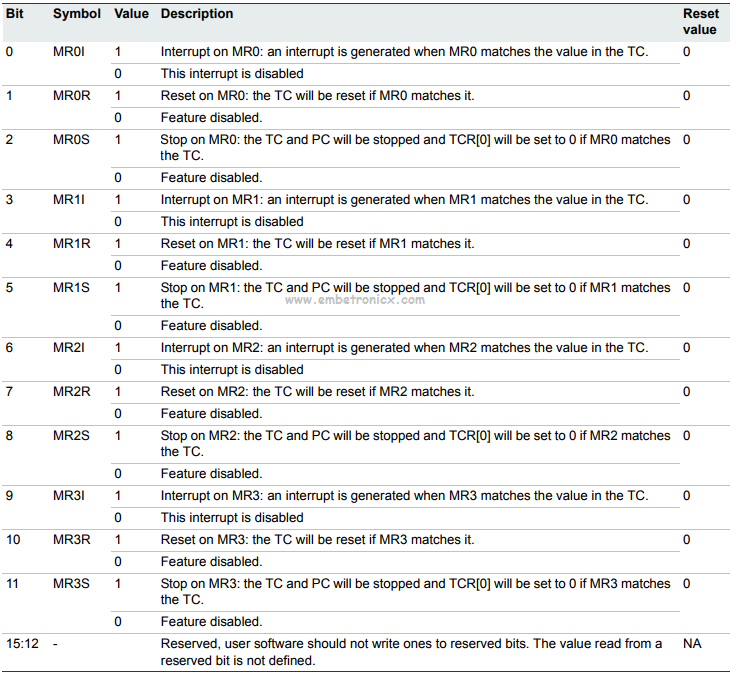
Match Control Register (MCR)

The Match Control Register is used to control the actions to be performed when the value in the Match Register (MR) matches with the value in the Timer Counter (TC). Bits 0 to 2 in MCR (i.e. MCR [0], MCR [1] and MCR [2]) are used to control the actions for MR0 register. Similarly, bits 3 to 5 for MR1, bits 6 to 8 for MR2, and bits 9 to 11 for MR3 respectively. We will see the function associated with MR0.

Bit 0: When this bit is 1, an interrupt is triggered when MR0 is equal to TC. When this bit is 0, the interrupts are disabled.

Bit 1: When this bit is 1, TC is reset when MR0 is equal to TC. When this bit is 0, this feature is disabled.

Bit 2: When this bit is 1, the Timer Counter (TC) and Prescale Counter (PC) are stopped when the value in MR0 is equal to TC. Also, the TC is reset to 0.



**Match Registers (MR0 – MR3)**

The Match register values are continuously compared to the Timer Counter value. When the two values are equal, actions can be triggered automatically. The action possibilities are to generate an interrupt, reset the Timer Counter, or stop the timer. Actions are controlled by the settings in the MCR register.

**Capture Control Registers (CCR)**

The Capture Control Register is used to control whether one of the four Capture Registers is loaded with the value in the Timer Counter when the capture event occurs and whether an interrupt is generated by the capture event.

**Capture Registers (CR0 -CR3)**

Each Capture register is associated with a device pin and may be loaded with the Timer Counter value when a specified event occurs on that pin. The settings in the Capture Control Register determine whether the capture function is enabled and whether a capture event happens on the rising edge of the associated pin, the falling edge, or on both edges.

**Configuring Timer**

To use timers we need to first configure them. I would like to encourage the readers to use the following sequence for Setting up Timers:

**1.Set appropriate value in TxCTCR**

**2.Define the Prescale value in TxPR**

**3.Set Value(s) in Match Register(s) if required**

**4.Set appropriate value in TxMCR if using Match registers / Interrupts**

**5.Reset Timer – Which resets PR and TC**

**6.Set TxTCR to 0x01 to Enable the Timer when required**

**7.Reset TxTCR to 0x00 to Disable the Timer when required**

**Prescale Register Value Calculation**

The delay or time required for 1 clock cycle at ‘X’ MHz is given by :

**Delay = 1/(X\*1000000) Seconds**

Hence in our case when PR=0 i.e TC increments at every PCLK the delay required for TC to increment by 1 is:

**Delay = (0+1)/(60\*1000000) Seconds**

Similarly, when we set PR = 59999 the delay will be:

**Delay = (599999+1)/(60\*1000000) Seconds**

**Delay = 1mS**

… which boils down to**1/1000 = 0.001** Seconds which is nothing but **1 Milli-Second i.e mS**. Hence the delay required for TC to increment by 1 will be 1mS.

In this code LED will Blink with a 1-second delay. I’m using Timer 0 in this code. LEDs are connected to Port 0.

#include<lpc214x.h>

void delay**(**unsigned int z**)**;

void pll**()**;

int main**(**void**)**

**{**

IO0DIR=0xffffffff;

pll**()**; //Fosc=12Mhz,CCLK=60Mhz,PCLK=60MHz

while**(**1**)** **{**

IO0SET=0xffffffff;

delay**(**1000**)**; //1sec delay

IO0CLR=0xffffffff;

delay**(**1000**)**; //1sec delay

**}**

**}**

void pll**()** //Fosc=12Mhz,CCLK=60Mhz,PCLK=60MHz

**{**

PLL0CON=0x01;

PLL0CFG=0x24;

PLL0FEED=0xaa;

PLL0FEED=0x55;

while**(**!**(**PLL0STAT&**(**1**<<**10**)))**;

PLL0CON=0x03;

PLL0FEED=0xaa;

PLL0FEED=0x55;

VPBDIV=0x01;

**}**

void delay**(**unsigned int z**)**

**{**

T0CTCR=0x0; //Select Timer Mode

T0TCR=0x00; //Timer off

T0PR=59999; //Prescaler value for 1ms

T0TCR=0x02; //Timer reset

T0TCR=0x01; //Timer ON

while**(**T0TC**<**z**)**;

T0TCR=0x00; //Timer OFF

T0TC=0; //Clear the TC value. This is Optional.

**}**

---------------------------------------------------------------------------------------------------------------------**Phase Locked Loop (PLL)**

There are two PLL modules in the LPC2141/2/4/6/8 microcontroller. The PLL0 is used to generate the CCLK clock (system clock) while the PLL1 has to supply the clock for the USB at the fixed rate of 48 MHz. Structurally these two PLLs are identical with exception of the PLL interrupt capabilities reserved only for the PLL0.

The PLL0 and PLL1 accept an input clock frequency in the range of 10 MHz to 25 MHz only. The input frequency is multiplied up the range of 10 MHz to 60 MHz for the CCLK and 48 MHz for the USB clock using a Current Controlled Oscillators (CCO).

The multiplier can be an integer value from 1 to 32 (in practice, the multiplier value cannot be higher than 6 on the LPC2141/2/4/6/8 due to the upper frequency limit of the CPU).

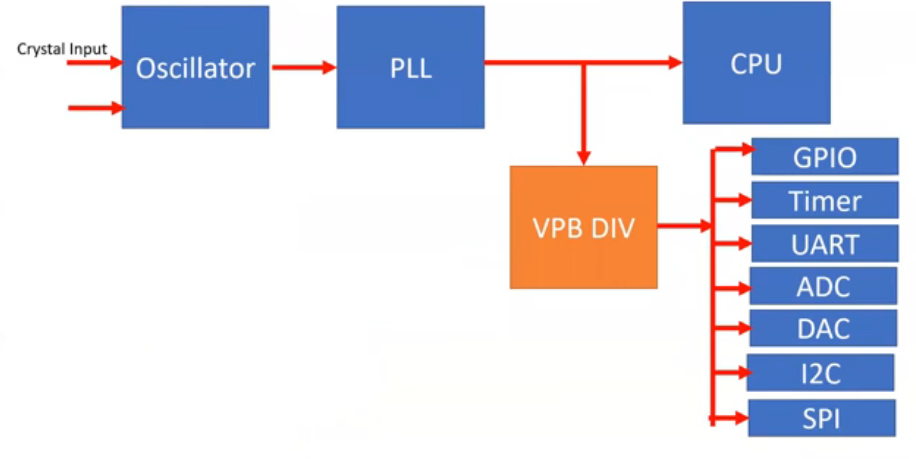
The CCO operates in the range of 156 MHz to 320 MHz, so there is an additional divider in the loop to keep the CCO within its frequency range while the PLL is providing the desired output frequency.

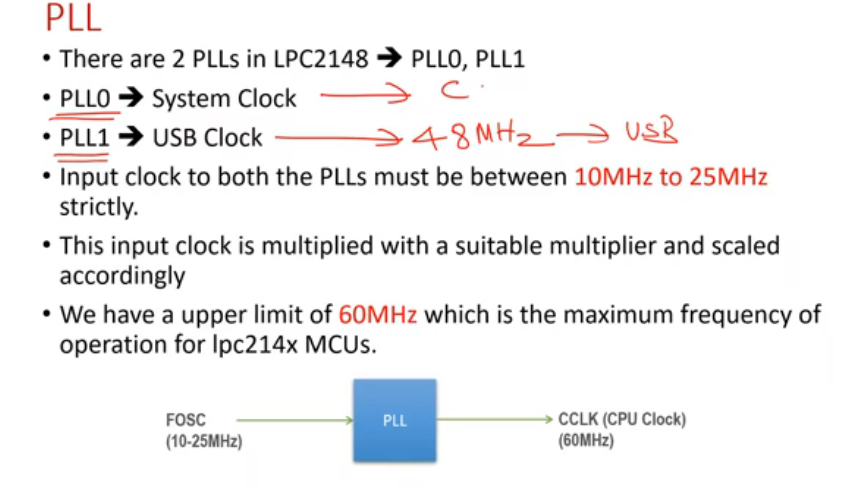
The output divider may be set to divide by 2, 4, 8, or 16 to produce the output clock. Since the minimum output divider value is 2, it is insured that the PLL output has a 50% duty cycle.

PLL activation is controlled via the PLLCON register. The PLL multiplier and divider values are controlled by the PLLCFG register. These two registers are protected in order to prevent accidental alteration of PLL parameters or deactivation of the PLL.

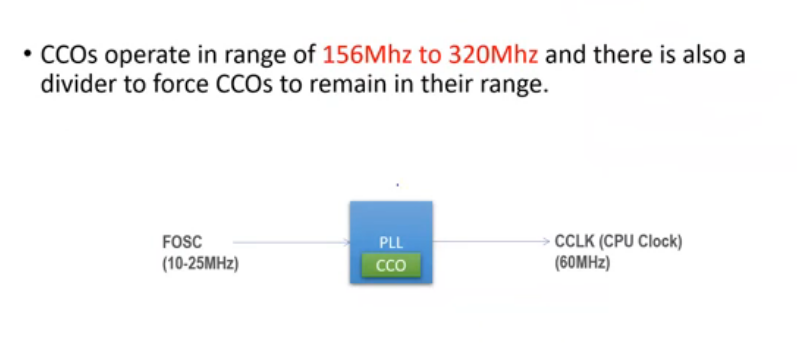
The same concern is present with the PLL1 and the USB. The protection is accomplished by a feed sequence similar to that of the Watchdog Timer. Details are provided in the description of the PLLFEED register.

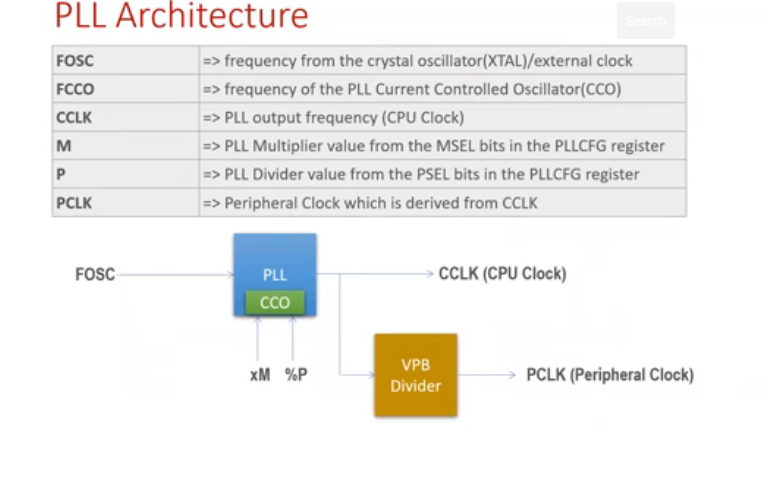
Both PLLs are turned off and bypassed following a chip Reset and when by entering Power-down mode. The PLL is enabled by software only. The program must configure and activate the PLL, wait for the PLL to Lock, then connect to the PLL as a clock source.

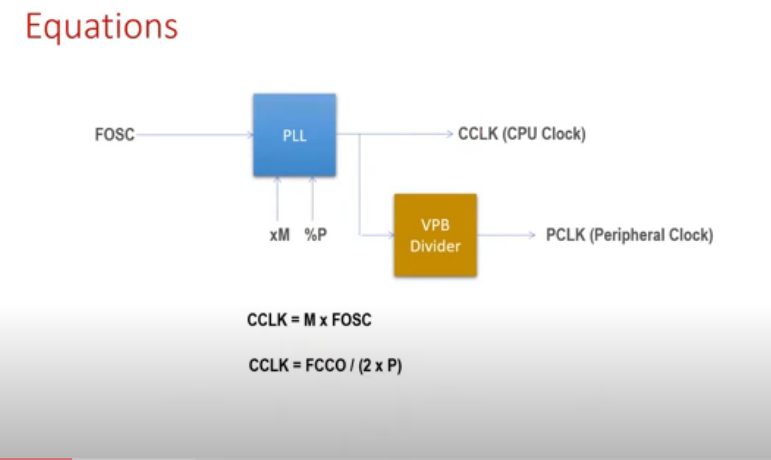
****

****

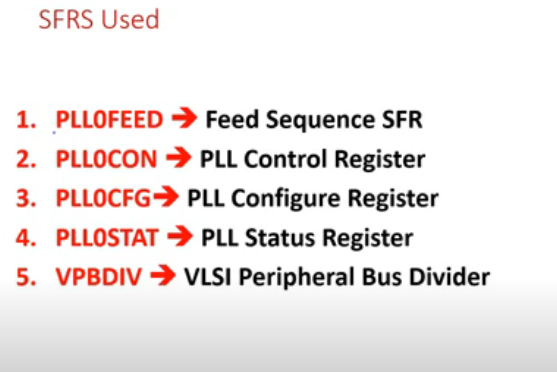
**Current controlled oscillators (CCO)**

****

****

****

**Register description**

****

**1.PLL Feed register (PLL0FEED):** A correct feed sequence must be written to the PLLFEED register in order for changes to the PLLCON and PLLCFG registers to take effect. The feed sequence is:

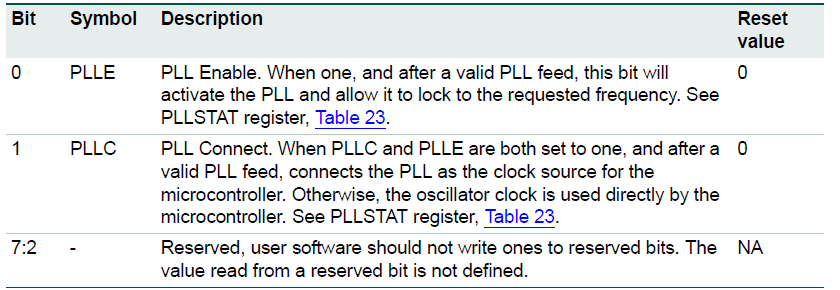
1. Write the value 0xAA to PLLFEED.

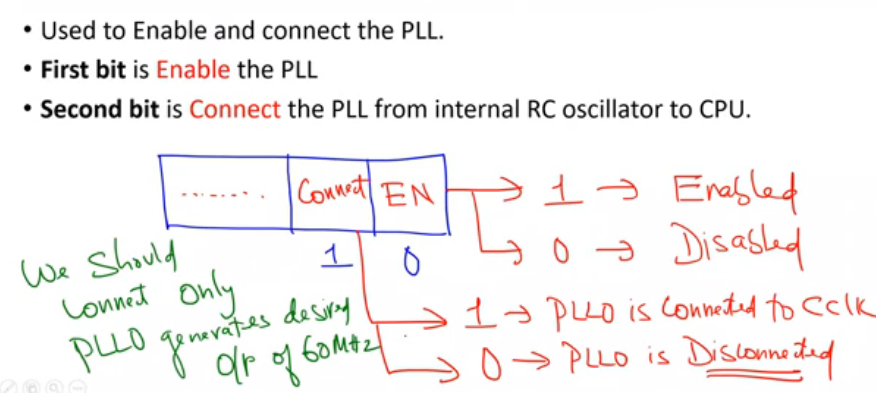
2. Write the value 0x55 to PLLFEED.

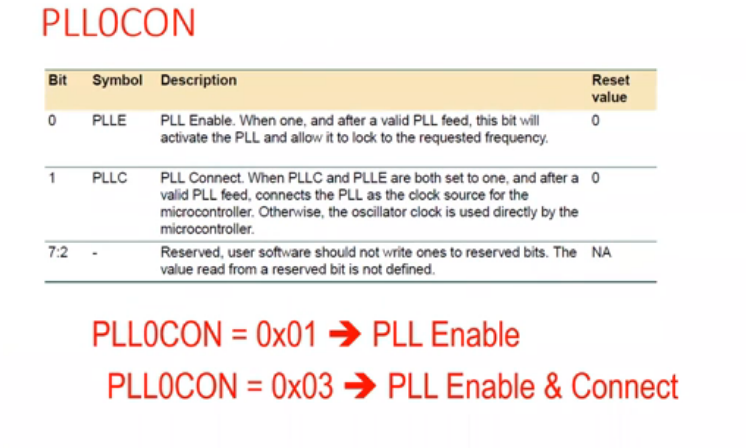




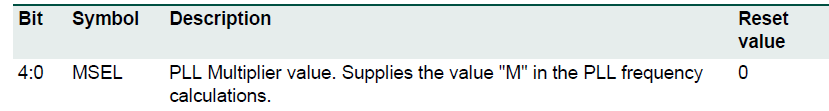
**2.PLL Control register (PLL0CON ):** The PLLCON register contains the bits that enable and connect the PLL. Enabling the PLL allows it to attempt to lock to the current settings of the multiplier and divider values. Connecting the PLL causes the processor and all chip functions to run from the PLL output clock. Changes to the PLLCON register do not take effect until a correct PLL feed sequence has been given

****

****

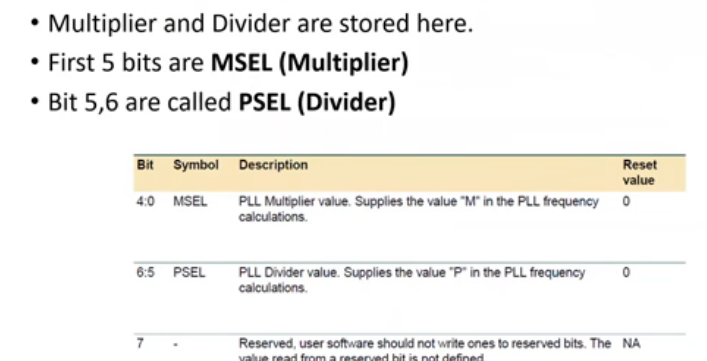
****

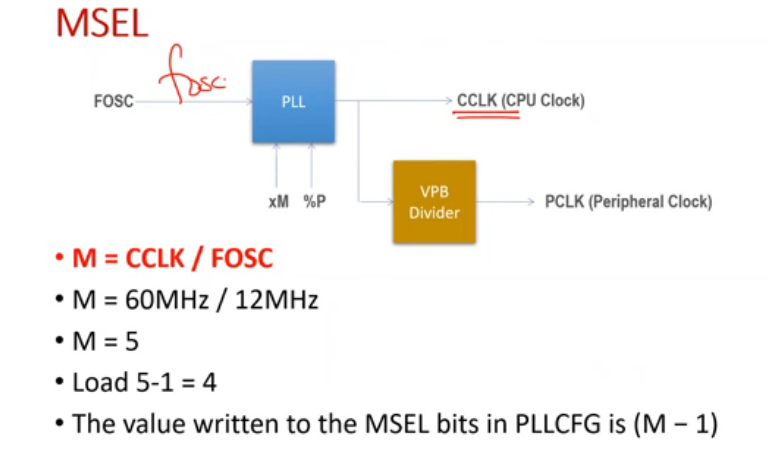
**3.PLL Configuration register (PLL0CFG ):** The PLLCFG register contains the PLL multiplier and divider values. Changes to the PLLCFG register do not take effect until a correct PLL feed sequence has been given

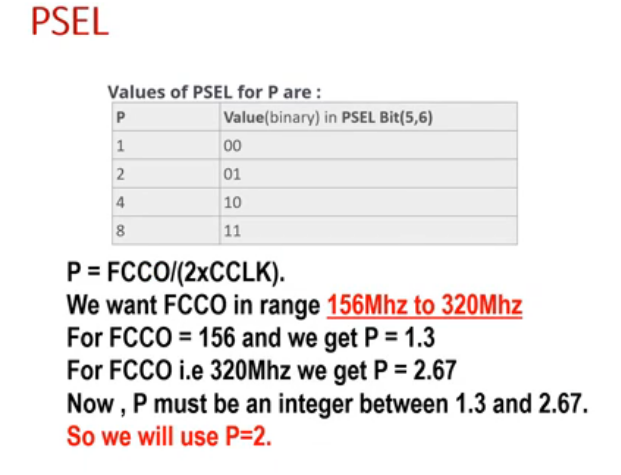
****

****

****

****

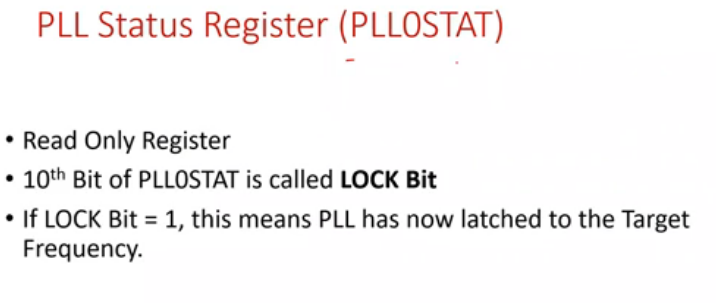
****

****

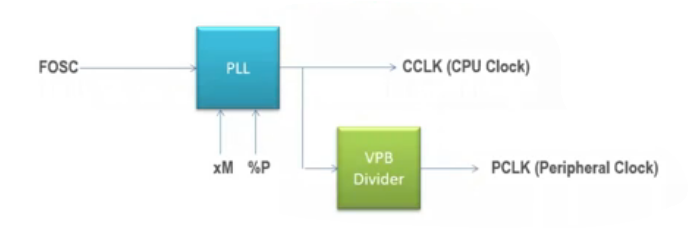
**PLL Status register (PLL0STAT):**

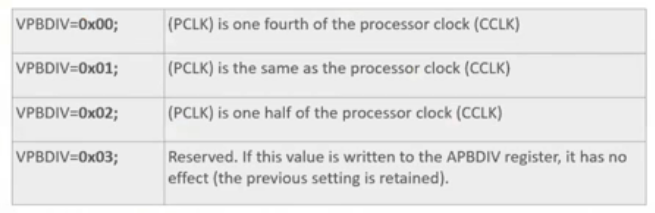
The read-only PLLSTAT register provides the actual PLL parameters that are in effect at the time it is read, as well as the PLL status. PLLSTAT may disagree with values found in PLLCON and PLLCFG because changes to those registers do not take effect until a proper PLL feed has occurred

****



**Deriving PCLK from CCLK**

****

****

**How to Configure PLL**

**Procedure for determining PLL settings:**

If a particular application uses the PLL0, its configuration may be determined as follows:

1.Choose the desired processor operating frequency (CCLK). This may be based on processor throughput requirements, need to support a specific set of UART baud rates, etc. Bear in mind that peripheral devices may be running from a lower clock than the processor.

2. Choose an oscillator frequency (FOSC). CCLK must be the whole (non-fractional) multiple of FOSC.

3. Calculate the value of M to configure the MSEL bits. M = CCLK / FOSC. M must be in the range of 1 to 32. The value written to the MSEL bits in PLLCFG is M -1

4. Find a value for P to configure the PSEL bits, such that FCCO is within its defined frequency limits. FCCO is calculated using the equation given above. P must have one of the values 1, 2, 4, or 8. The value written to the PSEL bits in PLLCFG is 00 for P = 1; 01 for P = 2; 10 for P = 4; 11 for P = 8

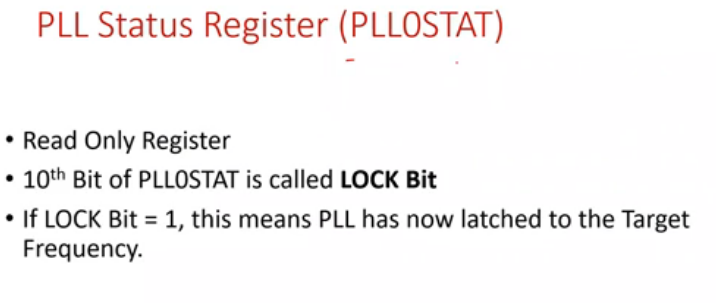
-------------------------------------------------------------------------------------------------------------------

1.Enable PLL ---------🡪 PLL0CON=0x01

2.Configure PLLCFG -------MSEL 4 for 5 and PSEL 2.

3.FEED sequesnce-------- PLL0FEED=0xAA; PLL0FEED=0x55;

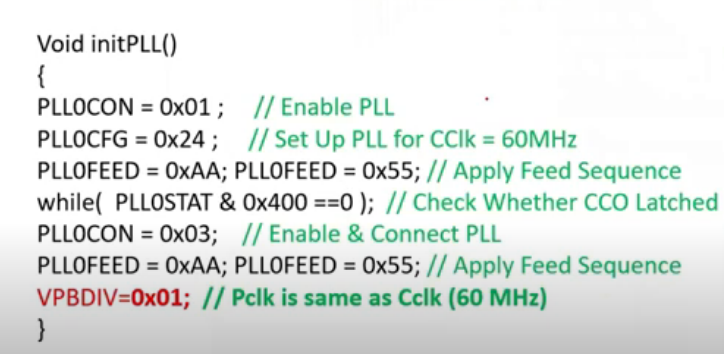
4.PLL Status register--- PLL0STAT

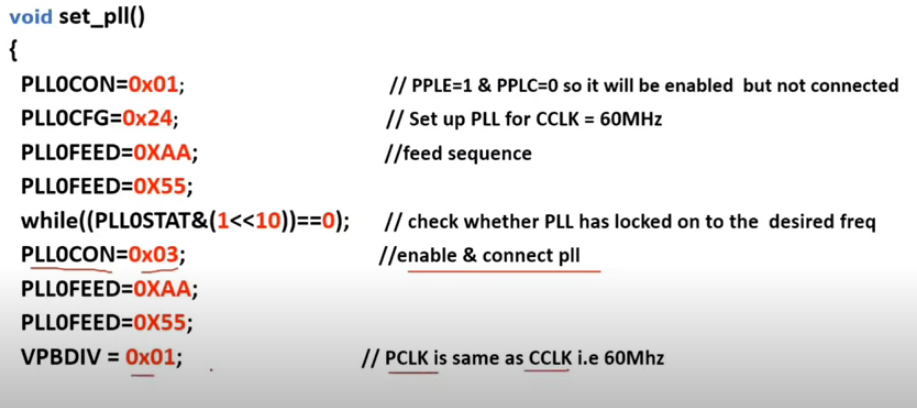


5. Connect the PLL-------PLL0CON-----00000011=0x03

6. FEED sequesnce-------- PLL0FEED=0xAA; PLL0FEED=0x55;

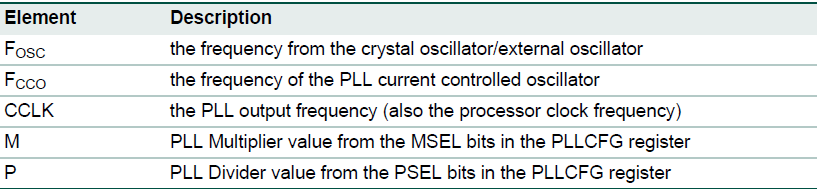
7.Derive PCLK--------- VPBDIV---PCLK.





**PLL frequency calculation:**

The PLL equations use the following parameters:



CCLK = M x FOSC

CCLK = FCCO / (2 x P)

The CCO frequency can be computed as: **FCCO = CCLK x 2 x P or FCCO = FOSC x M x 2 x P.**

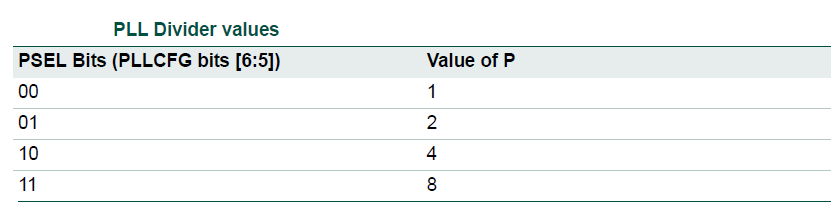
The PLL inputs and settings must meet the following:

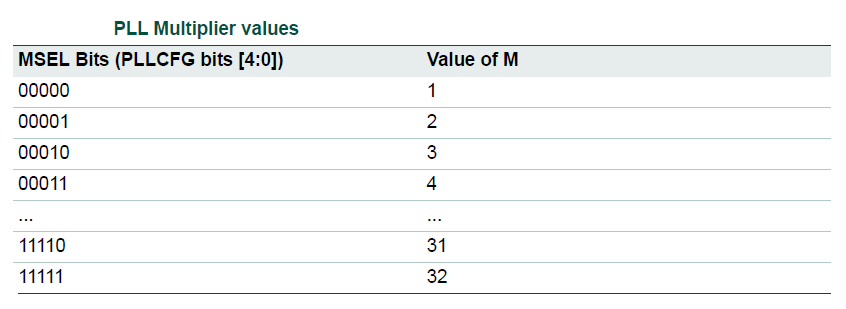
**•** FOSC is in the range of 10 MHz to 25 MHz.

**•** CCLK is in the range of 10 MHz to Fmax (the maximum allowed frequency for the

microcontroller - determined by the system microcontroller is embedded in).

**•** FCCO is in the range of 156 MHz to 320 MHz.





**Example 1:** an application not using the USB - configuring the PLL0

System design asks for FOSC= 10 MHz and requires CCLK = 60 MHz.Based on these specifications, M = CCLK / Fosc = 60 MHz / 10 MHz = 6. Consequently, M - 1 = 5 will be written as PLLCFG[4:0].

Value for P can be derived from P = FCCO / (CCLK x 2), using condition that FCCO must be

in range of 156 MHz to 320 MHz. Assuming the lowest allowed frequency for FCCO = 156 MHz, P = 156 MHz / (2 x 60 MHz) = 1.3. The highest FCCO frequency criteria produces P = 2.67

The only solution for P that satisfies both of these requirements and is listed above table,so is P = 2.