Work book-ARM7(LPC2148)

Learn-Practice-Grow

www.pantechsolutions.net

8/19/2021

Contents

Getting started with ARM LPC2148 using Keil uVision IDE

There are various development environments available in the market for ARM processors.

Some of these are mentioned below :

- CrossWorks for Arm
- Keil µVision
- IAR Embedded Workbench

We will see how to install and setup the µVision IDE by Keil.

We will see the steps that need to be followed for installing this software correctly.

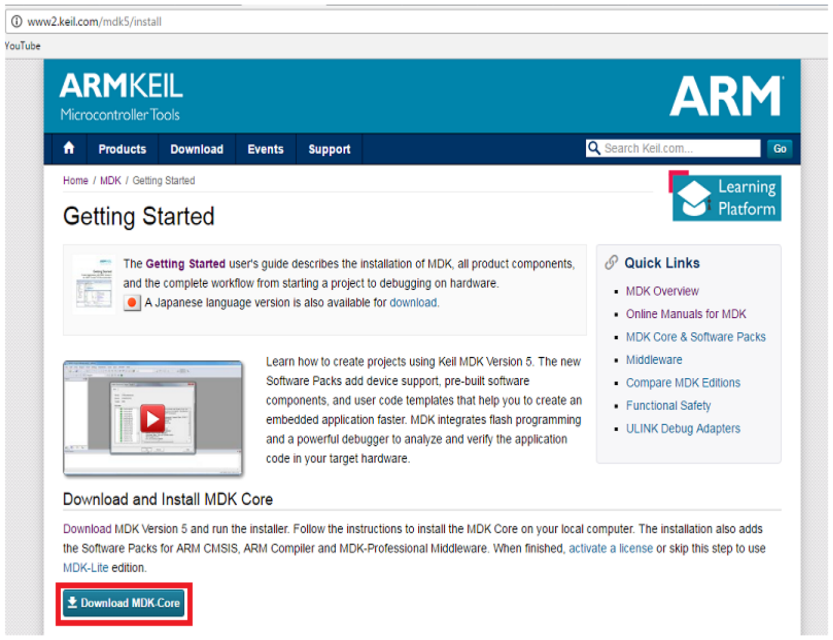
When this is done, we will setup the environment for LPC2148 and write a basic code for LED blinking.

Downloading and installation

Follow the steps given below:

1. Download the MDK-lite (Microcontroller Development Kit) by Keil from their website. Here is the link to the page from where you can download this: http://www2.keil.com/mdk5/install

Click on **Download MDK-Core**.



Getting Started

The Getting Started user's guide describes the installation of MDK, all product components, and the complete workflow from starting a project to debugging on hardware.
A Japanese language version is also available for download.

Learn how to create projects using Keil MDK Version 5. The new Software Packs add device support, pre-built software components, and user code templates that help you to create an embedded application faster. MDK integrates flash programming and a powerful debugger to analyze and verify the application code in your target hardware.

Download and Install MDK Core

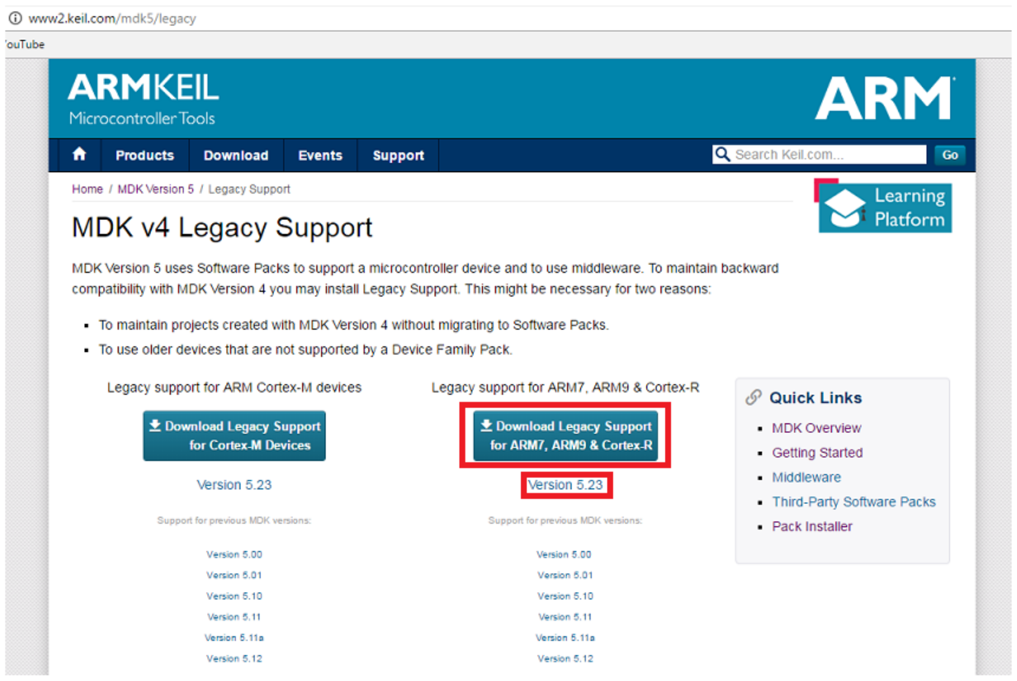
Download MDK Version 5 and run the installer. Follow the instructions to install the MDK Core on your local computer. The installation also adds the Software Packs for ARM CMSIS, ARM Compiler and MDK-Professional Middleware. When finished, activate a license or skip this step to use MDK-Lite edition.

Install the software by following the simple instructions provided during installation process.

2. The new version µVision5 does not support many of the devices that were supported in the older versions yet. LPC2148 is one of the devices that are not supported. Hence, we need to add this device after successfully installing µVision5.

To do this, go to the following link and download the executable file for Legacy Support for ARM7, ARM9, and Cortex-R: http://www2.keil.com/mdk5/legacy

Download the Legacy support for the version of MDK downloaded and installed.



www2.keil.com/mdk5/legacy
ARMKEIL
Microcontroller Tools
ARM
Products
Download
Events
Support
Search Keil.com...
Go
Home / MDK Version 5 / Legacy Support
Learning Platform
Download Legacy Support for Cortex-M Devices
Download Legacy Support for ARM7, ARM9 & Cortex-R
Quick Links
MDK Overview
Getting Started
Middleware
Third-Party Software Packs
Pack Installer

MDK v4 Legacy Support

MDK Version 5 uses Software Packs to support a microcontroller device and to use middleware. To maintain backward compatibility with MDK Version 4 you may install Legacy Support. This might be necessary for two reasons:

- To maintain projects created with MDK Version 4 without migrating to Software Packs.
- To use older devices that are not supported by a Device Family Pack.

Legacy support for ARM Cortex-M devices	Legacy support for ARM7, ARM9 & Cortex-R
Version 5.23	Version 5.23
Support for previous MDK versions:	Support for previous MDK versions:
Version 5.00	Version 5.00
Version 5.01	Version 5.01
Version 5.10	Version 5.10
Version 5.11	Version 5.11
Version 5.11a	Version 5.11a
Version 5.12	Version 5.12

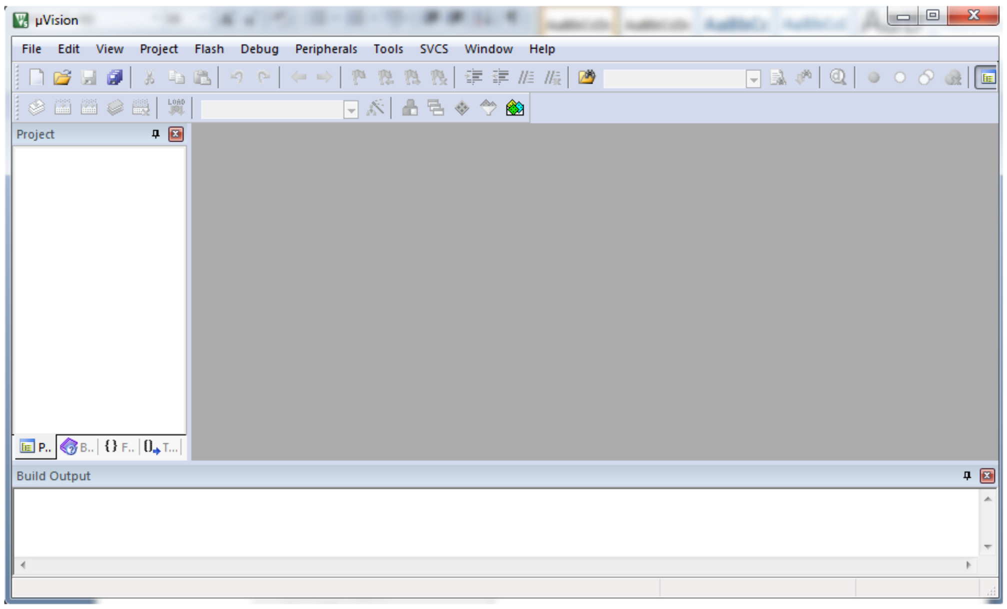
Install the executable file that will be downloaded. Follow the simple instructions provided during the installation process.

When the above described steps are completed, we will have the IDE installed and ready to use with support for the device we intend to use, i.e. LPC2148.

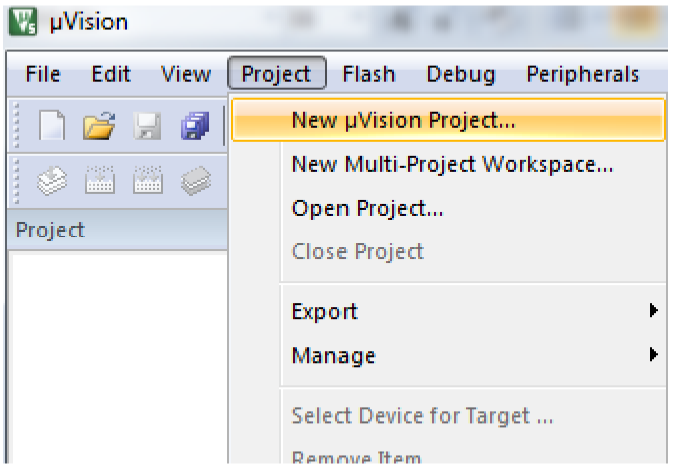
Using µVision IDE

We will create a simple LED blinking project. Following are steps which show how to create and built project using the Keil uVision IDE:

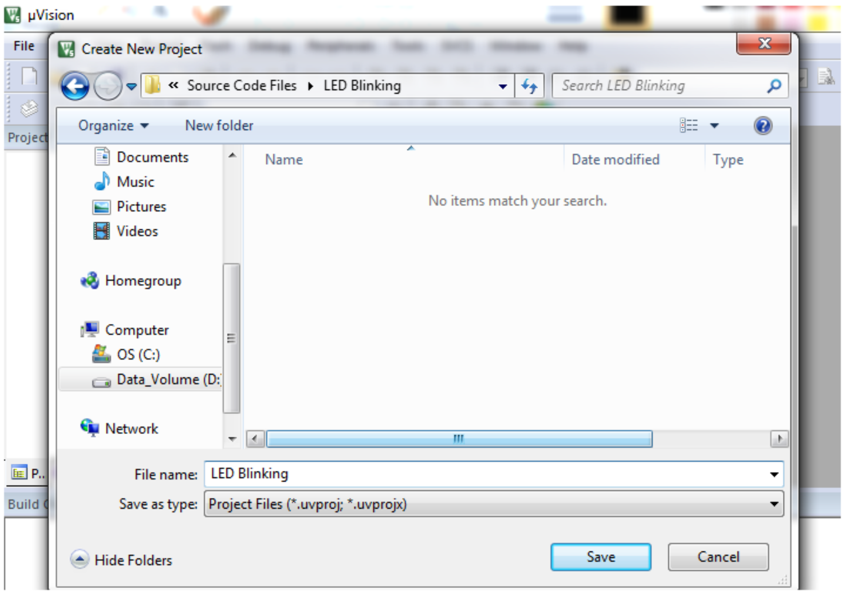
1. Open Keil µVision from the icon created on your desktop.


µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Project
Build Output

2. Go to the **Project** tab. Select New **µVision Project ...**from that menu.

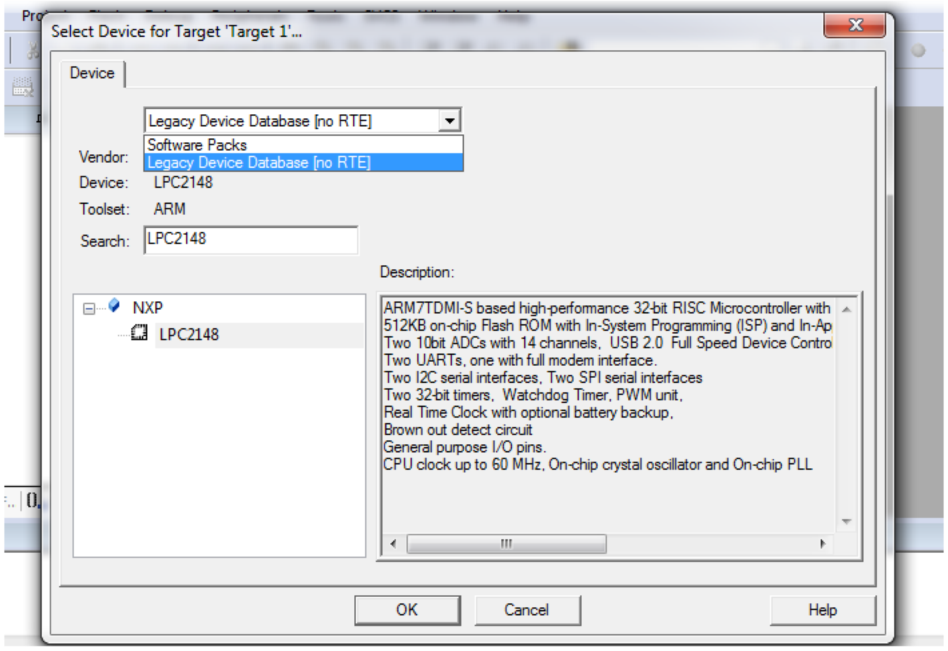

µVision
File Edit View Project Flash Debug Peripherals
New µVision Project...
New Multi-Project Workspace...
Open Project...
Close Project
Export
Manage
Select Device for Target ...
Remove Item
Project

3. **Create New Project** window will pop up. Select the folder where you want to create project and give a suitable name to the project. Then click on **Save**.



File name: LED Blinking
Save as type: Project Files (*.uvproj; *.uvprojx)

4. **Select Device for Target: 'Target1'...**window will pop up next. It has a select window to choose between Software Packs or Legacy Device Database. As LPC2148 is in Legacy Device Database, choose Legacy Device Database.

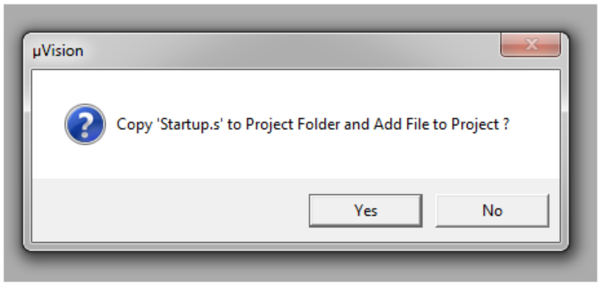

Select Device for Target 'Target 1'...
Device
Software Packs
Legacy Device Database [no RTE]
NXP
LPC2148
OK
Cancel
Help

Legacy Device Database [no RTE]
Vendor:
Device: LPC2148
Toolset: ARM
Search: LPC2148

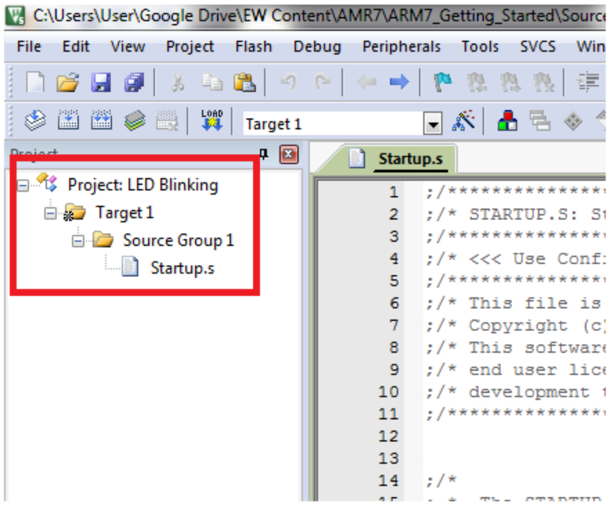
Description:
ARM7TDMI-S based high-performance 32-bit RISC Microcontroller with
512KB on-chip Flash ROM with In-System Programming (ISP) and In-Ap
Two 10bit ADCs with 14 channels, USB 2.0 Full Speed Device Contro
Two UARTs, one with full modem interface.
Two I2C serial interfaces, Two SPI serial interfaces
Two 32-bit timers, Watchdog Timer, PWM unit,
Real Time Clock with optional battery backup,
Brown out detect circuit
General purpose I/O pins.
CPU clock up to 60 MHz, On-chip crystal oscillator and On-chip PLL

Type in LPC2148 in search and select the device under NXP with the name LPC2148 and click on OK.

5. A window will pop up asking whether to copy Startup.s to project folder and add file to project. Click on **Yes**.

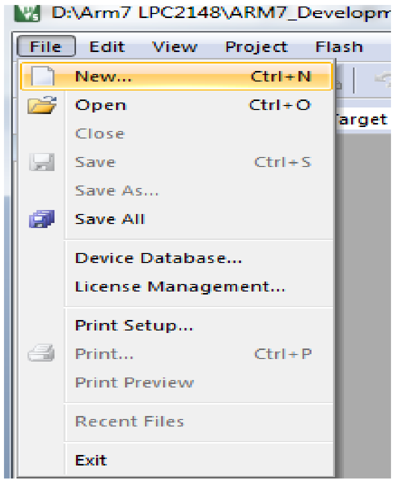

µVision
Copy 'Startup.s' to Project Folder and Add File to Project ?
Yes
No

6. The project name and its folders can be seen on the left side in the project window after the previous step is completed as shown below.

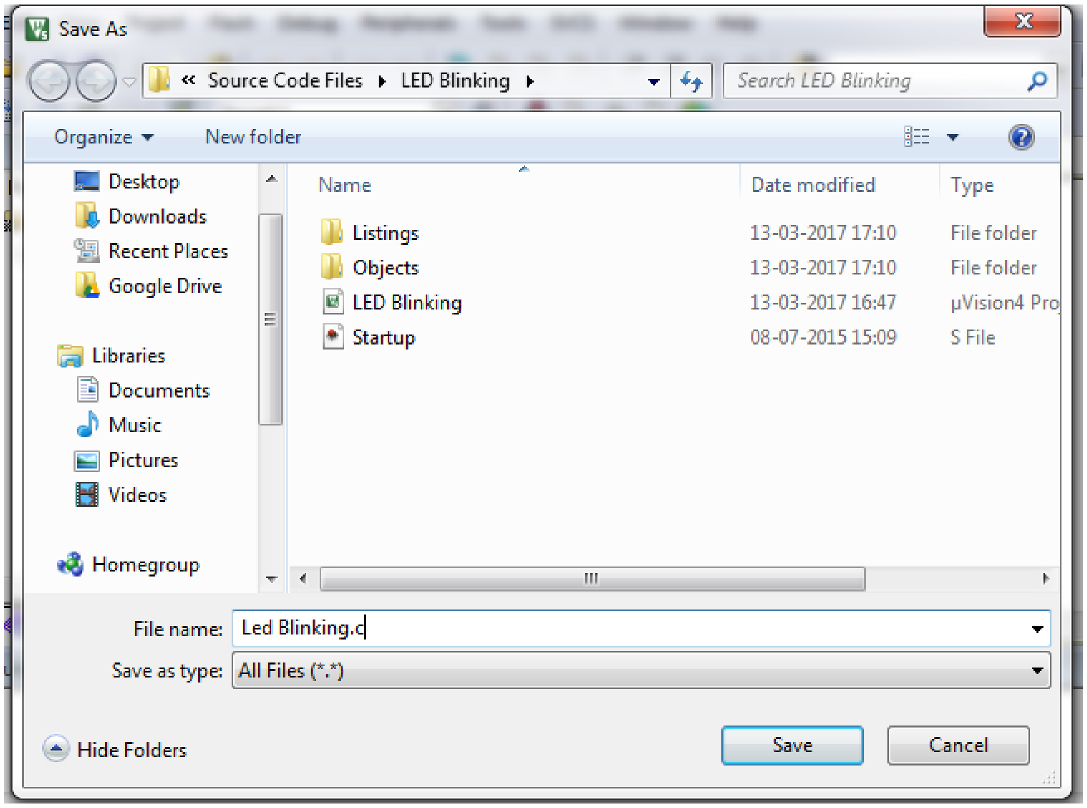

C:\Users\User\Google Drive\EW Content\AMR7\ARM7_Getting_Started\Sourc
File Edit View Project Flash Debug Peripherals Tools SVCS Win
Target 1
Project: LED Blinking
Target 1
Source Group 1
Startup.s
Startup.s

```
;/***************
;/* STARTUP.S: St
;/***************
;/* <<< Use Conf:
;/***************
;/* This file is
;/* Copyright (c)
;/* This software
;/* end user lice
;/* development t
;/***************


;/*
```

7. Now go to File tab and add **New** file from the menu.



8. Save the file from the previous step with a specific name. Add .c extension to the file name.

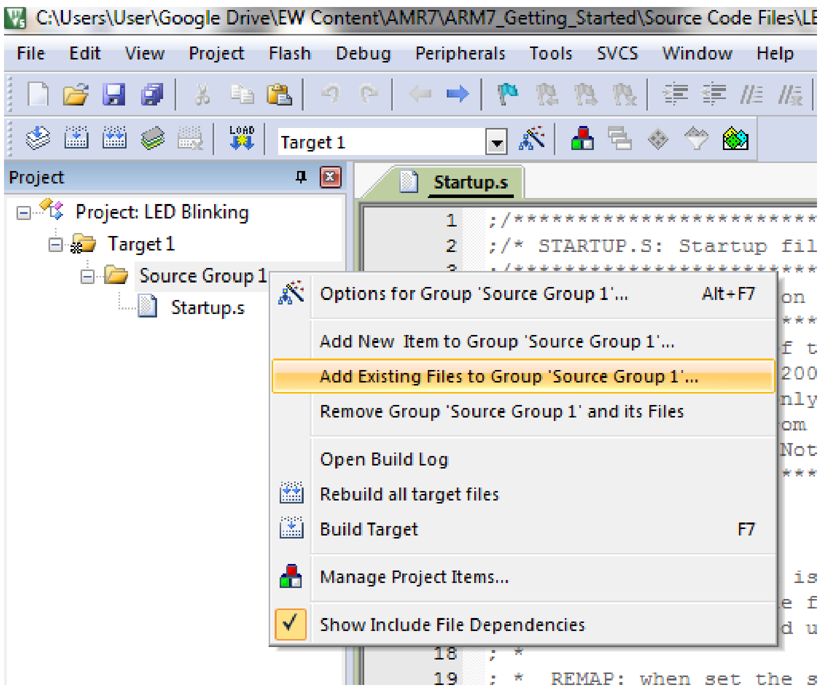


Name	Date modified	Type
Listings	13-03-2017 17:10	File folder
Objects	13-03-2017 17:10	File folder
LED Blinking	13-03-2017 16:47	µVision4 Pro
Startup	08-07-2015 15:09	S File

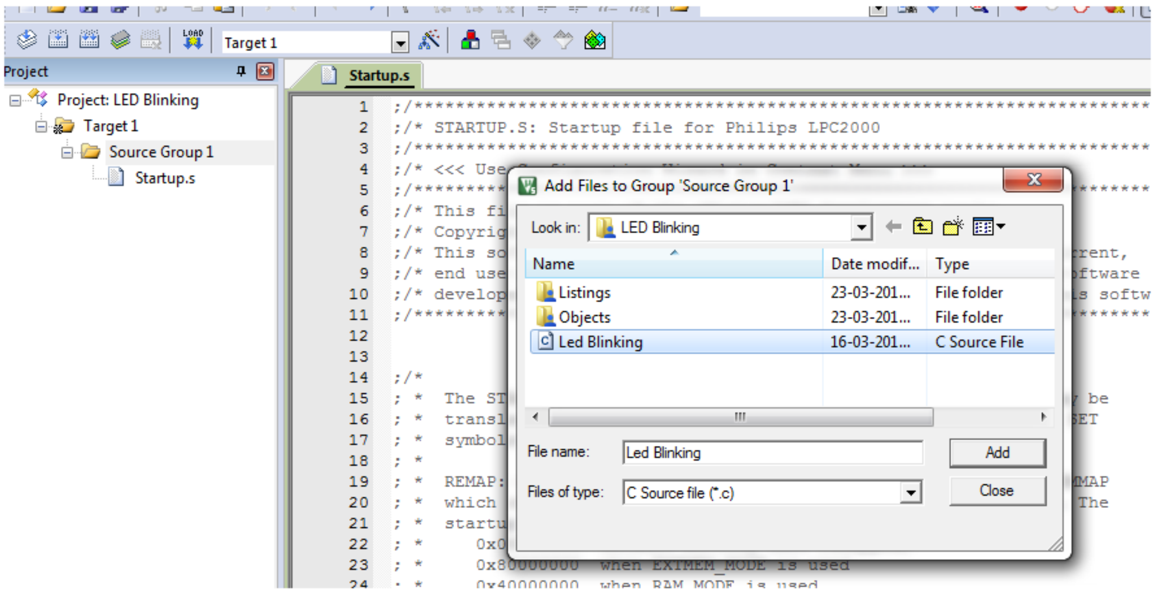
File name: Led Blinking.c

Save as type: All Files (*.*)

9. Add this file to Source Group folder in the project window by right clicking on Source Group1 folder and selecting **Add Existing Files to Group 'Source Group1'...**


C:\Users\User\Google Drive\EW Content\AMR7\ARM7_Getting_Started\Source Code Files\LE
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Target 1
Project
Project: LED Blinking
Target 1
Source Group 1
Startup.s
Options for Group 'Source Group 1'... Alt+F7
Add New Item to Group 'Source Group 1'...
Add Existing Files to Group 'Source Group 1'...
Remove Group 'Source Group 1' and its Files
Open Build Log
Rebuild all target files
Build Target F7
Manage Project Items...
Show Include File Dependencies

```
;/*****************************
;/* STARTUP.S: Startup fil
```

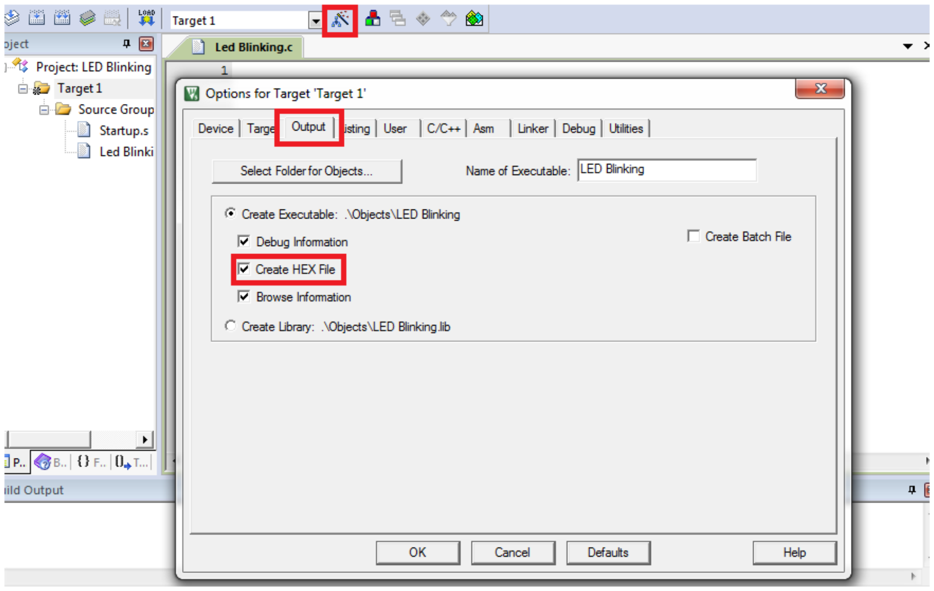

Target 1
Project
Project: LED Blinking
Target 1
Source Group 1
Startup.s
Add Files to Group 'Source Group 1'
Look in: LED Blinking
Name
Date modif...
Type
Listings 23-03-201... File folder
Objects 23-03-201... File folder
Led Blinking 16-03-201... C Source File
File name: Led Blinking
Files of type: C Source file (*.c)
Add
Close

```
;/*****************************************************************
;/* STARTUP.S: Startup file for Philips LPC2000
;/*****************************************************************
```

Select the previously saved file from the window that pops up and add it to the Source Group1. In our case, LED Blinking.c

10. Now click on the **Options for Target ‘Target1’...** symbol shown in red box in the image below or press **Alt+F7** or right click on Target1 and click on **Options for Target ‘Target1’...**.

Options for target window will open. Go to the **Output** tab in that window. Tick '√' **Create HEX File** option. We need to produce HEX file to burn it into the microcontroller.



Name of Executable: LED Blinking

(●) Create Executable: .\Objects\LED Blinking

[✓] Debug Information

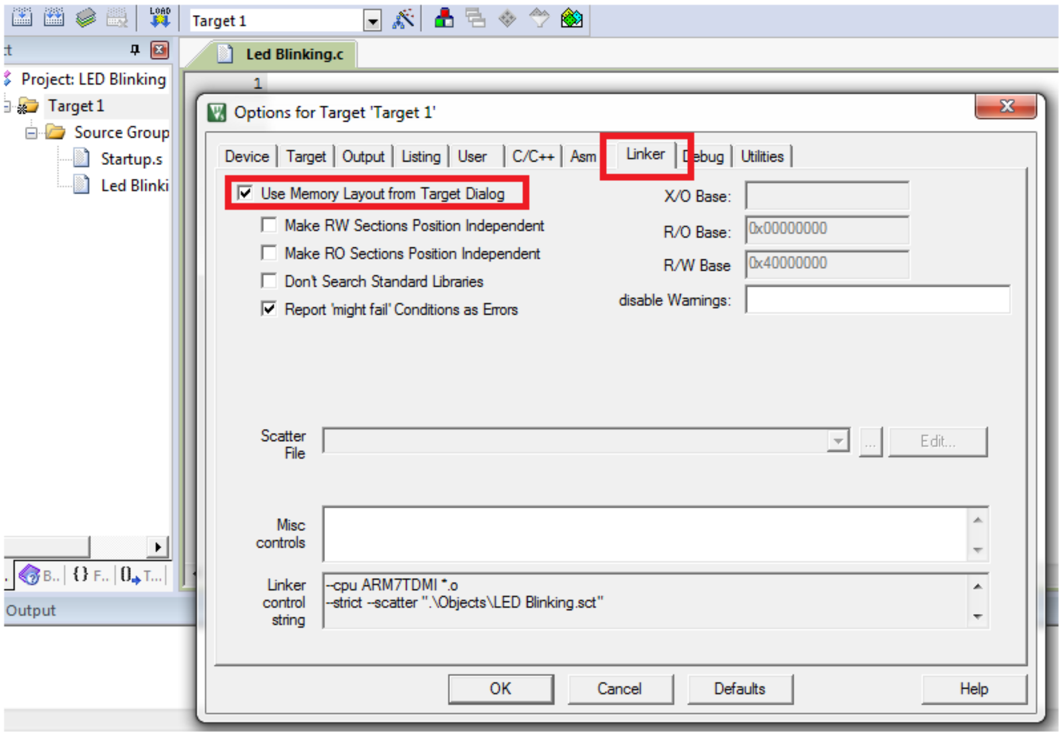
[✓] Create HEX File

[✓] Browse Information

[ ] Create Batch File

( ) Create Library: .\Objects\LED Blinking.lib

In the options for target window, go to the **Linker** tab. Select the **Use Memory Layout from Target Dialogue** option.



[✓] Use Memory Layout from Target Dialog

[ ] Make RW Sections Position Independent

[ ] Make RO Sections Position Independent

[ ] Don't Search Standard Libraries

[✓] Report 'might fail' Conditions as Errors

X/O Base:

R/O Base: 0x00000000

R/W Base 0x40000000

disable Warnings:

Scatter File

Misc controls

Linker control string:

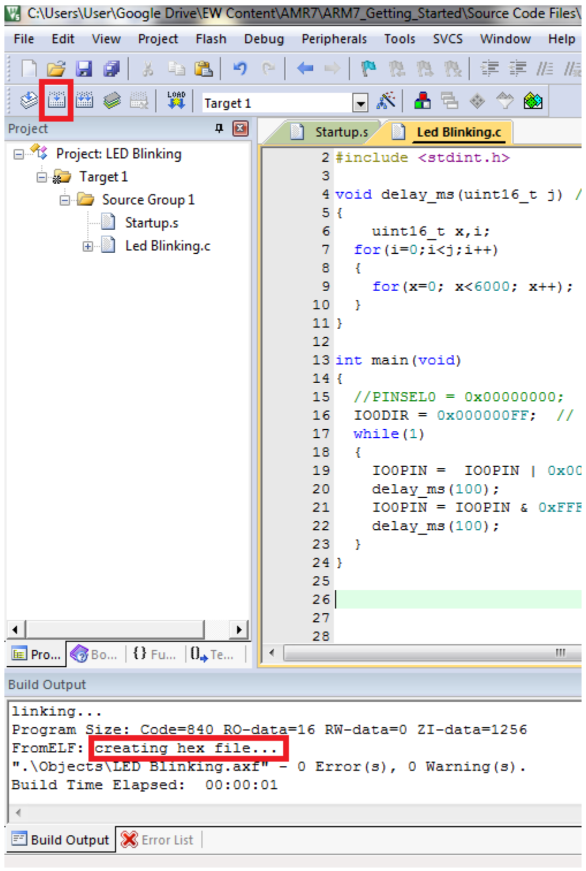
```
--cpu ARM7TDMI *.o
--strict --scatter ".\Objects\LED Blinking.sct"
```

Then click on OK.

11. Now write the code for LED Blinking.

```
/*   LED Blinking on LPC2148(ARM7)*/
#include <lpc214x.h>
#include <stdint.h>
void delay_ms(uint16_t j) /* Function for delay in milliseconds */
{
   uint16_t x,i;
         for(i=0;i<j;i++)
         {
   for(x=0; x<6000; x++);    /* loop to generate 1 millisecond delay with 12MHz Fosc. */
         }
}

int main(void)
{
         IO0DIR = 0x000000FF;         /* Set P0.0 to P0.7 bits as output bits by writing 1 in
IO0DIR register corresponding to those bits. */
         while(1)
         {
                  IO0PIN =  IO0PIN | 0x000000FF; /* Make P0.0 to P0.7 HIGH while
keeping other bits unchanged. */
                  delay_ms(300);
                  IO0PIN = IO0PIN & 0xFFFFFF00; /* Make P0.0 to P0.7 LOW while
keeoing other bits unchanged. */
                  delay_ms(300);
         }
}
```

12. Once the code is written, **Build** the code by clicking on the button shown in red in the image below. You can also build the project from the **Build Target** option in the Project tab or by pressing **F7** on the keyboard.



```
#include <stdint.h>

void delay_ms(uint16_t j) /
{
   uint16_t x,i;
  for(i=0;i<j;i++)
  {
    for(x=0; x<6000; x++);
  }
}

int main(void)
{
  //PINSEL0 = 0x00000000;
  IO0DIR = 0x000000FF;  //
  while(1)
  {
    IO0PIN =  IO0PIN | 0x0C
    delay_ms(100);
    IO0PIN = IO0PIN & 0xFFF
    delay_ms(100);
  }
}
```

```
linking...
Program Size: Code=840 RO-data=16 RW-data=0 ZI-data=1256
FromELF: creating hex file...
".\Objects\LED Blinking.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:01
```

You can see **creating hex file ...** in the Build Output window as shown in the image.

13. Once the project is built, a **hex file** is created in the **Objects** folder inside the folder of your project. Use **Flash Magic** software to burn this hex file in your microcontroller.

Download Flash Magic from the following Link: http://www.flashmagictool.com

LPC2148 (32-bit ARM7TDMI-S processor) GPIO Ports and Registers

Introduction

General-purpose input/output (GPIO) is a pin on an IC (Integrated Circuit). It can be either input pin or output pin, whose behaviour can be controlled at the run time. A group of these pins is called a port (Example, Port 0 of LPC2148 has 32 pins).

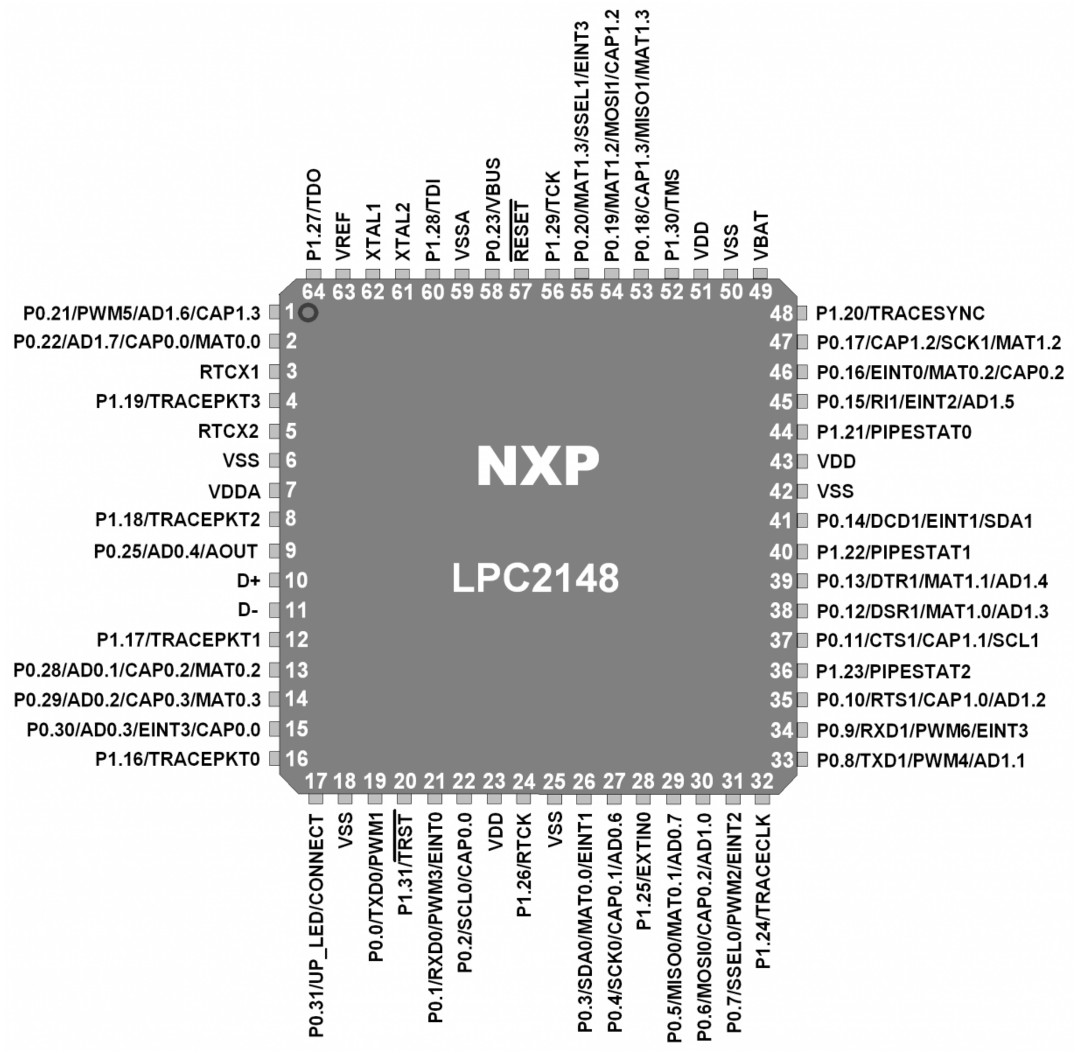
LPC2148 has two 32-bit General Purpose I/O ports.

1. **PORT0**
2. **PORT1**

PORT0 is a 32-bit port

- Out of these 32 pins, 28 pins can be configured as either general purpose input or output.
- 1 of these 32 pins (P0.31) can be configured as general-purpose output only.
- 3 of these 32 pins (P0.24, P0.26 and P0.27) are reserved. Hence, they are not available for use. Also, these pins are not mentioned in pin diagram.

PORT1 is also a 32-bit port. Only 16 of these 32 pins (P1.16 – P1.31) are available for use as general-purpose input or output.



LPC2148 Pin Diagram

Almost every pin of these two ports has some alternate function available. For example, P0.0 can be configured as the TXD pin for UART0 or as PWM1 pin as well. The functionality of each pin can be selected using the **Pin Function Select Registers**.

Note : The Port 0 pins do not have built-in pull-up or pull-down resistors. Hence, while using GPIOs on Port 0, in some cases, we need to connect pull-up or pull-down resistors externally.

Pin Function Select Registers

Pin Function Select Registers are 32-bit registers. These registers are used to select or configure specific pin functionality.

There are 3 Pin Function Select Registers in LPC2148:

1. **PINSEL0 : -** PINSEL0 is used to configure PORT0 pins P0.0 to P0.15.
2. **PINSEL1 : -** PINSEL1 is used to configure PORT0 pins P0.16 to P0.31.
3. **PINSEL2 : -** PINSEL2 is used to configure PORT1 pins P1.16 to P1.31.

Let's see GPIO registers that control the GPIO operations.

Fast and Slow GPIO Registers

There are 5 Fast (also called Enhanced GPIO Features Registers) GPIO Registers and 4 Slow (also called Legacy GPIO Registers) GPIO Registers available to control PORT0 and PORT1.

The Slow Registers allow backward compatibility with earlier family devices using the existing codes.

Slow GPIO Registers

There are 4 Slow GPIO registers :

1. **IOxPIN (GPIO Port Pin value register):** This is a 32-bit wide register. This register is used to read/write the value on Port (PORT0/PORT1). But care should be taken while writing. Masking should be used to ensure write to the desired pin.

Examples :

a) Writing 1 to P0.4 using IO0PIN

```
IO0PIN = IO0PIN | (1<<4)
```

b) Writing 0 to P0.4 using IO0PIN

```
IO0PIN = IO0PIN & (~(1<<4) )
```

c) Writing F to P0.7-P0.4

```
IO0PIN = IO0PIN | (0x000000F0)
```

2. **IOxSET (GPIO Port Output Set register) :** This is a 32-bit wide register. This register is used to make pins of Port (PORT0/PORT1) HIGH. Writing one to specific bit makes that pin HIGH. Writing zero has no effect.

3. **IOxDIR (GPIO Port Direction control register) :** This is a 32-bit wide register. This register individually controls the direction of each port pin. Setting a bit to '1' configures the corresponding pin as an output pin. Setting a bit to '0' configures the corresponding pin as an input pin.

4. **IOxCLR (GPIO Port Output Clear register) :** This is a 32-bit wide register. This register is used to make pins of Port LOW. Writing one to specific bit makes that pin LOW. Writing zeroes has no effect.

Examples :

a) Configure pin P0.0 to P0.3 as input pins and P0.4 to P0.7 as output pins.

```
IO0DIR = 0x000000F0;
```

b) Configure pin P0.4 as an output. Then set that pin HIGH.

```
IO0DIR = 0x00000010; OR IO0DIR = (1<<4);
IO0SET = (1<<4);
```

c) Configure pin P0.4 as an output. Then set that pin LOW.

```
IO0DIR = 0x00000010; OR IO0DIR = (1<<4);
IO0CLR = (1<<4);
```

Fast GPIO Registers

There are 5 fast GPIO registers :

1. **FIOxDIR (Fast GPIO Port Direction control register) :** This is a 32-bit wide register.This register individually controls the direction of each port pin. Setting a bit to ‘1’ configures the corresponding pin as an output pin. Setting a bit to ‘0’ configures the corresponding pin as an input pin.

2. **FIOxMASK (Fast Mask register for port) :** This is a 32-bit wide register. This register controls the effect of fast registers (FIOxPIN, FIOxSET, FIOxCLR) on port pins. Setting a bit to ‘0’ configures the corresponding pin access to the fast registers i.e. we can write/read the corresponding pin in fast mode using fast registers. Setting a bit to ‘1’ configures the corresponding pin unaffected by fast registers.

3. **FIOxPIN (Fast Port Pin value register using FIOMASK) :** This is a 32-bit wide register. This register is used to read/write the value on port pins, only if that corresponding port pins have access to fast registers (Access to port pins using fast registers is enabled using Zeroes in FIOxMASK register).

4. **FIOxSET (Fast Port Output Set register using FIOMASK) :** This is a 32-bit wide register. This register is used to make pins of Port HIGH. Writing one to specific bit makes that pin HIGH. Writing zero has no effect. Reading this register returns the current contents of the port output register. Only bits enabled by ZEROES in FIOMASK can be altered.

5. **FIOxCLR (Fast Port Output Clear register using FIOMASK) :** This is a 32-bit wide register.This register is used to make pins of Port LOW. Writing one to specific bit makes that pin LOW. Writing zeroes has no effect. Only bits enabled by ZEROES in FIOMASK can be altered.

Aside from the 32-bit long and word only accessible registers mentioned above, every fast GPIO port can also be controlled via several byte and half-word accessible registers. Refer chapter 8 (Page 83) on GPIO in the datasheet of LPC2148 provided in the attachments section for their use.

Examples :

a) Configure pin P0.0 to P0.3 as input pins and P0.4 to P0.7 as output pins.

```
FIO0DIR = 0x000000F0;
```

b) Configure pin P0.4 as an output. Then set that pin HIGH.

```
FIO0DIR = 0x00000010; OR FIO0DIR = (1<<4);
FIO0SET = (1<<4);
```

c) Configure pin P0.4 as an output. Then set that pin LOW.

```
FIO0DIR = 0x00000010; OR FIO0DIR = (1<<4);
FIO0CLR = (1<<4);
```

Why this is called Fast GPIO :

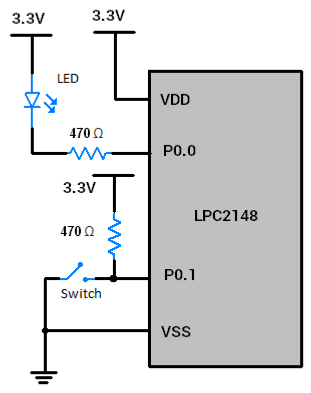
Fast GPIO registers are relocated to the ARM local bus. This makes software access to GPIO pins 3.5 times faster than access through Slow GPIO registers. This effect will not always be visible when a program is written in c code. It may be more evident in an assembly code than in a c code.

Example

Now, let’s write a simple program for turning LED ON or OFF depending on the status of the pin.

Here, LED is interfaced to P0.0.

A switch is interfaced to P0.1 to change the status of the pin.



Program

```
#include <lpc214x.h>

#include <stdint.h>


int main(void)

{

        //PINSEL0 = 0x00000000;          /* Configuring P0.0 to P0.15 as GPIO */

        /* No need for this as PINSEL0 reset value is 0x00000000 */

        IO0DIR = 0x00000001;              /* Make P0.0 bit as output bit, P0.1 bit as an input pin  */

        while(1)

        {

                if ( IO0PIN & (1<<1) )     /* If switch is open, pin is HIGH */

                {

                        IO0CLR = 0x00000001;  /* Turn on LED */

                }

                else /* If switch is closed, pin is LOW */

                {

                        IO0SET = 0x00000001;   /* Turn off LED */

                }

        }

}
```

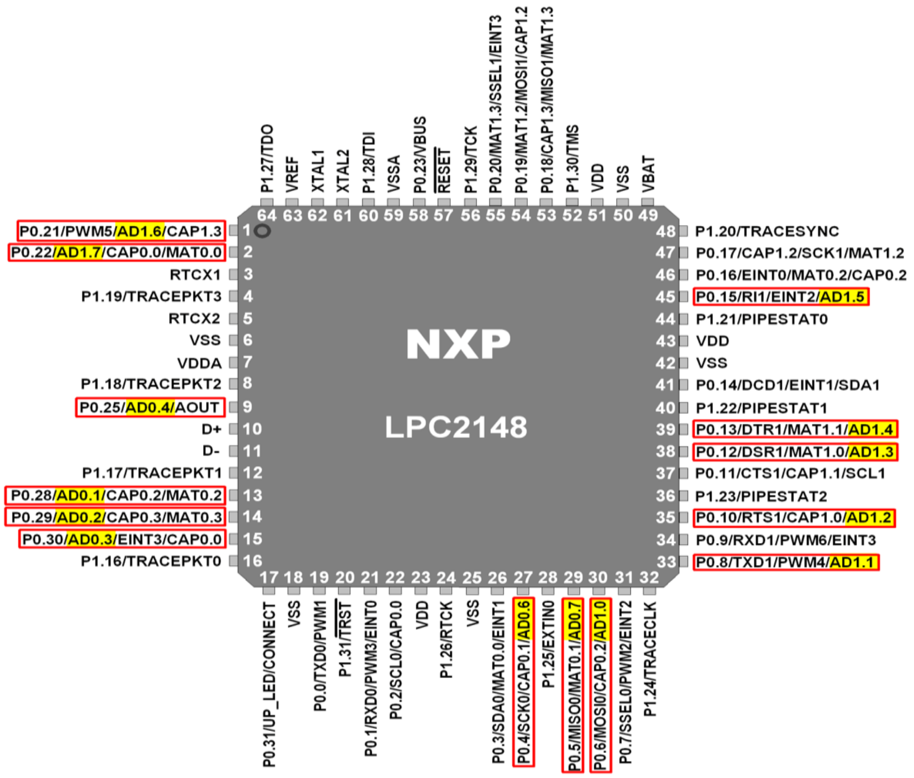
ADC (Analog to Digital Converter) in ARM LPC2148

Introduction

Analog to Digital Converter(ADC) is used to convert analog signal into digital form. LPC2148 has two inbuilt 10-bit ADC i.e. ADC0 & ADC1.

- ADC0 has 6 channels &ADC1 has 8 channels.
- Hence, we can connect 6 distinct types of input analog signals to ADC0 and 8 distinct types of input analog signals to ADC1.
- ADCs in LPC2148 use Successive Approximation technique to convert analog signal into digital form.
- This Successive Approximation process requires a clock less than or equal to 4.5 MHz. We can adjust this clock using clock divider settings.
- Both ADCs in LCP2148 convert analog signals in the range of 0V to VREF (typically 3V; not to exceed VDDA voltage level).

LPC4128 ADC Pins



LPC4128 ADC Pins

AD0.1:4, AD0.6:7 & AD1.7:0 (Analog Inputs)

These are Analog input pins of ADC. If ADC is used, signal level on analog pins must not be above the level of VDDA; otherwise, ADC readings will be invalid. If ADC is not used, then the pins can be used as 5V tolerant digital I/O pins.

VREF (Voltage Reference)

Provide Voltage Reference for ADC.

VDDA& VSSA (Analog Power and Ground)

These are the power and ground pins for ADC. These should be same as VDD & VSS.

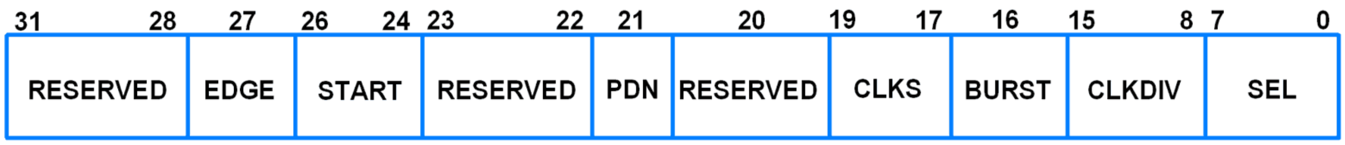
Let's see the ADC registers which are used to control and monitors the ADC operation.

Here, we will see ADC0 registers and their configurations. ADC1 has similar registers and can be configured in a similar manner.

ADC0 Registers

1. **AD0CR (ADC0 Control Register)**

- AD0CR is a 32-bit register.
- This register must be written to select the operating mode before A/D conversion can occur.
- It is used for selecting channel of ADC, clock frequency for ADC, number of clocks or number of bits in result, start of conversion and few other parameters.



AD0CR (ADC0 Control Register)

- **Bits7:0–SEL**
  These bits select ADC0 channel as analog input. In software-controlled mode, only one of these bits should be 1.e.g. bit 7 (10000000) selects AD0.7 channel as analog input.

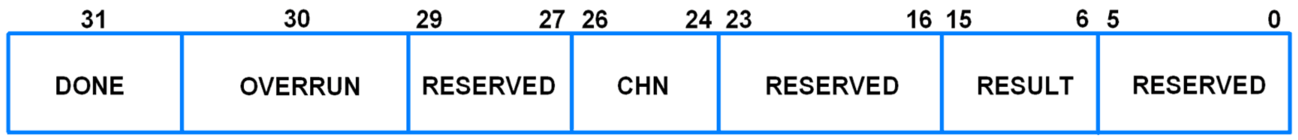
- **Bits15:8–CLKDIV**
  The APB(ARM Peripheral Bus)clock is divided by this value plus one, to produce the clock for ADC. This clock should be less than or equal to 4.5MHz.
- **Bit16–BURST**
  0=Conversions are software controlled and require 11 clocks
  1 = In Burst mode ADC does repeated conversions at the rate selected by the **CLKS** field for the analog inputs selected by **SEL** field. It can be terminated by clearing this bit, but the conversion that is in progress will be completed.
  When Burst = 1, the **START** bits must be 000, otherwise the conversions will not start.
- **Bits 19:17 – CLKS**
  Selects the number of clocks used for each conversion in burst mode and the number of bits of accuracy of Result bits of AD0DR.
  e.g. 000 uses 11 clocks for each conversion and provide 10 bits of result in corresponding **ADDR** register.
  000 = 11 clocks / 10 bits
  001 = 10 clocks / 9 bits
  010 = 9 clocks / 8 bits
  011 = 8 clocks / 7 bits
  100 = 7 clocks / 6 bits
  101 = 6 clocks / 5 bits
  110 = 5 clocks / 4 bits
  111 = 4 clocks / 3 bits
- **Bit 20 – RESERVED**
- **Bit 21 – PDN**
  0 = ADC is in Power Down mode
  1 = ADC is operational
- **Bit 23:22 – RESERVED**
- **Bit 26:24 – START**
  When **BURST** bit is 0, these bits control whether and when A/D conversion is started
  000 = No start (Should be used when clearing PDN to 0)
  001 = Start conversion now
  010 = Start conversion when edge selected by bit 27 of this register occurs on CAP0.2/MAT0.2 pin
  011= Start conversion when edge selected by bit 27 of this register occurs on CAP0.0/MAT0.0 pin
  100 = Start conversion when edge selected by bit 27 of this register occurs on MAT0.1

pin
101 = Start conversion when edge selected by bit 27 of this register occurs on MAT0.3 pin
110 = Start conversion when edge selected by bit 27 of this register occurs on MAT1.0 pin
111 = Start conversion when edge selected by bit 27 of this register occurs on MAT1.1 pin

- **Bit 27 – EDGE**
  This bit is significant only when the Start field contains 010-111. In these cases,
  0 = Start conversion on a rising edge on the selected CAP/MAT signal
  1 = Start conversion on a falling edge on the selected CAP/MAT signal
- **Bit 31:28 – RESERVED**

2. **AD0GDR (ADC0 Global Data Register)**

- AD0GDR is a 32-bit register.
- This register contains the ADC's DONE bit and the result of the most recent A/D conversion.



AD0GDR (ADC0 Global Data Register)

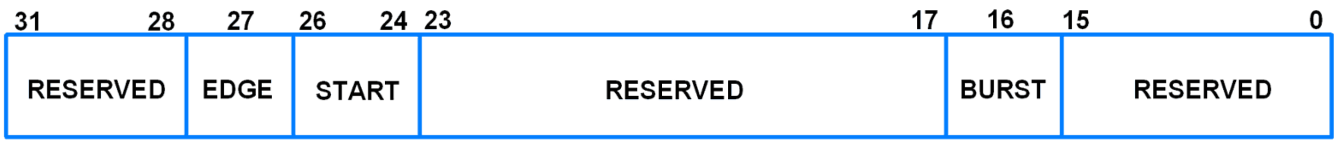
- **Bit 5:0 – RESERVED**
- **Bits 15:6 – RESULT**
  When **DONE** bit is set to 1, this field contains 10-bit ADC result that has a value in the range of 0 (less than or equal to VSSA) to 1023 (greater than or equal to VREF).
- **Bit 23:16 – RESERVED**
- **Bits 26:24 – CHN**
  These bits contain the channel from which ADC value is read.
  e.g. 000 identifies that the **RESULT** field contains ADC value of channel 0.
- **Bit 29:27 – RESERVED**
- **Bit 30 – Overrun**
  This bit is set to 1 in burst mode if the result of one or more conversions is lost and

overwritten before the conversion that produced the result in the **RESULT** bits.
This bit is cleared by reading this register.

- **Bit 31 – DONE**
  This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read and when the AD0CR is written.
  If AD0CR is written while a conversion is still in progress, this bit is set and new conversion is started.

3. **ADGSR (A/D Global Start Register)**

- ADGSR is a 32-bit register.
- Software can write to this register to simultaneously start conversions on both ADC.

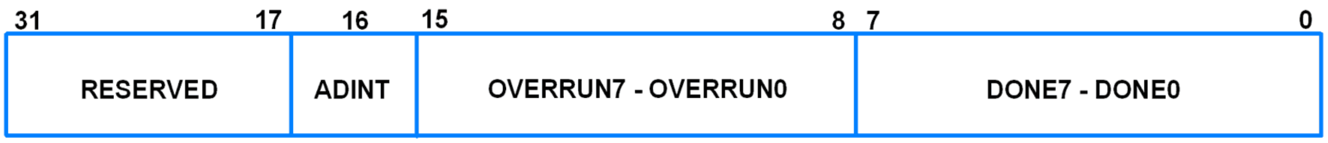


ADGSR (A/D Global Start Register)

- **BURST (Bit 16), START (Bit <26:24>) & EDGE (Bit 27)**
  These bits have same function as in the individual ADC control registers i.e. AD0CR & AD1CR. Only difference is that we can use these function for both ADC commonly from this register.

4. **AD0STAT (ADC0 Status Register)**

- AD0STAT is a 32-bit register.
- It allows checking of status of all the A/D channels simultaneously.

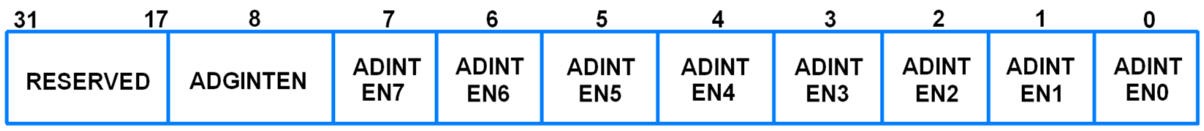


AD0STAT (ADC0 Status Register)

- **Bit 7:0 – DONE7:DONE0**
  These bits reflect the **DONE** status flag from the result registers for A/D channel 7 - channel 0.
- **Bit 15:8 – OVERRUN7:OVERRUN0**
  These bits reflect the **OVERRUN** status flag from the result registers for A/D channel 7 - channel 0.
- **Bit 16 – ADINT**
  This bit is 1 when any of the individual A/D channel DONE flags is asserted and enables ADC interrupt if any of interrupt is enabled in AD0INTEN register.
- **Bit 31:17 – RESERVED**

5. **AD0INTEN (ADC0 Interrupt Enable)**

- AD0INTEN is a 32-bit register.
- It allows control over which channels generate an interrupt when conversion is completed.

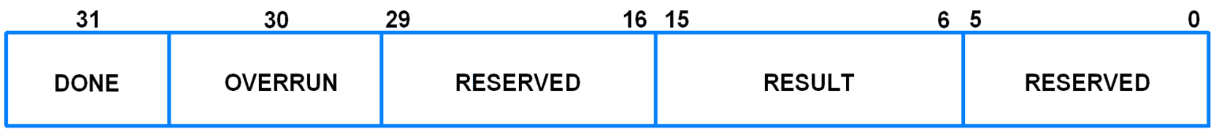


AD0INTEN (ADC0 Interrupt Enable)

- **Bit 0 – ADINTEN0**
  0 = Completion of a A/D conversion on ADC channel 0 will not generate an interrupt
  1 = Completion of a conversion on ADC channel 0 will generate an interrupt
- Remaining **ADINTEN** bits have similar description as given for **ADINTEN0**.
- **Bit 8 – ADGINTEN**
  0 = Only the individual ADC channels enabled by **ADINTEN7:0** will generate interrupts
  1 = Only the global **DONE** flag in A/D Data Register is enabled to generate an interrupt

6. **AD0DR0-AD0DR7 (ADC0 Data Registers)**

- These are 32-bit registers.
- They hold the result when A/D conversion is completed.
- They also include flags that indicate when a conversion has been completed and when a conversion overrun has occurred.



AD0 Data Registers Structure

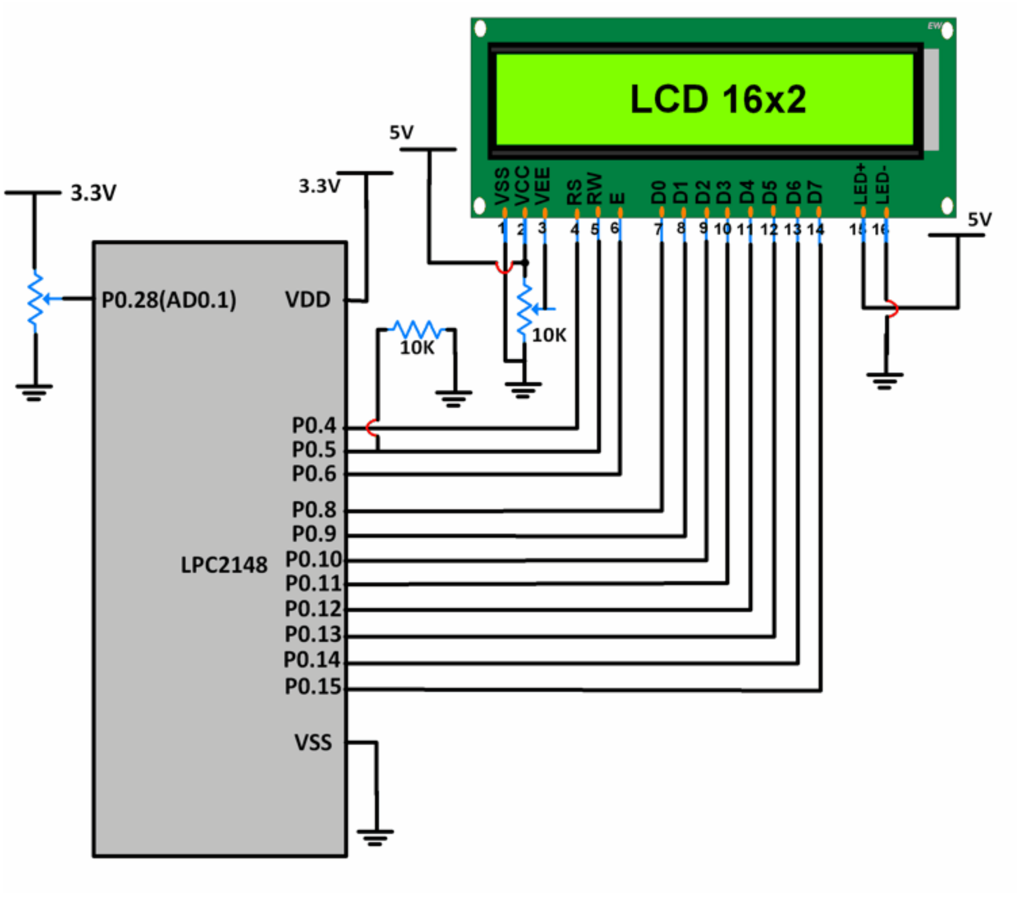
- **Bit 5:0 – RESERVED**
- **Bits 15:6 – RESULT**
  When **DONE** bit is set to 1, this field contains 10-bit ADC result that has a value in the range of 0 (less than or equal to VSSA) to 1023 (greater than or equal to VREF).
- **Bit 29:16 – RESERVED**
- **Bit 30 – Overrun**
  This bit is set to 1 in burst mode if the result of one or more conversions is lost and overwritten before the conversion that produced the result in the **RESULT** bits.
  This bit is cleared by reading this register.
- **Bit 31 – DONE**
  This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read.

Steps for Analog to Digital Conversion

1. Configure the ADxCR (ADC Control Register) according to the need of application.
2. Start ADC conversion by writing appropriate value to START bits in ADxCR. (Example, writing 001 to START bits of the register 26:24, conversion is started immediately).
3. Monitor the DONE bit (bit number 31) of the corresponding ADxDRy (ADC Data Register) till it changes from 0 to 1. This signals completion of conversion. We can also monitor DONE bit of ADGSR or the DONE bit corresponding to the ADC channel in the ADCxSTAT register.
4. Read the ADC result from the corresponding ADC Data Register. ADxDRy. E.g. AD0DR1 contains ADC result of channel 1 of ADC0.

Example

Let’s write a program to convert input voltage signal on AD0.1 (P0.28) into digital signal. We will convert the digital value to equivalent voltage value and display it on an LCD. We can compare this voltage with actual voltage measured on a digital multimeter.



Here, input signal is a DC signal which varies from 0 to 3.3V via a potentiometer.

Signal is given on P0.28. P0.28 is configured as AD0.1 using PINSEL register.

10-bit ADC result is stored in a variable and its lower 8 bits are given to P0.8-P0.15. These pins are connected to the data pins of an LCD.

P0.4,5,6 are used as RS, RW, EN pins of the LCD.

As the potentiometer is varied, we can see the variation in equivalent voltage value on the LCD.

Program

```
#include <lpc214x.h>
#include <stdint.h>
#include "LCD-16x2-8bit.h"
#include <stdio.h>
#include <string.h>
int main(void)
{
        uint32_t result;
        float voltage;
        char volt[18];
        LCD_Init();
        PINSEL1 = 0x01000000; /* P0.28 as AD0.1 */
        AD0CR = 0x00200402; /* ADC operational, 10-bits, 11 clocks for
conversion */
        while(1)
        {
                AD0CR = AD0CR | (1<<24); /* Start Conversion */
                while ( !(AD0DR1 & 0x80000000) ); /* Wait till DONE */
                result = AD0DR1;
                result = (result>>6);
                result = (result & 0x000003FF);
                voltage = ( (result/1023.0) * 3.3 ); /* Convert ADC value to
equivalent voltage */
                LCD_Command(0x80);
                sprintf(volt, "Voltage=%.2f V  ", voltage);
                LCD_String(volt);
                memset(volt, 0, 18);
        }
}
```

PWM in ARM LPC2148

Introduction

Pulse Width Modulation (PWM) is a technique by which width of a pulse is varied while keeping the frequency constant.

A period of a pulse consists of an **ON** cycle (HIGH) and an **OFF** cycle (LOW). The fraction for which the signal is ON over a period is known as **duty cycle**.

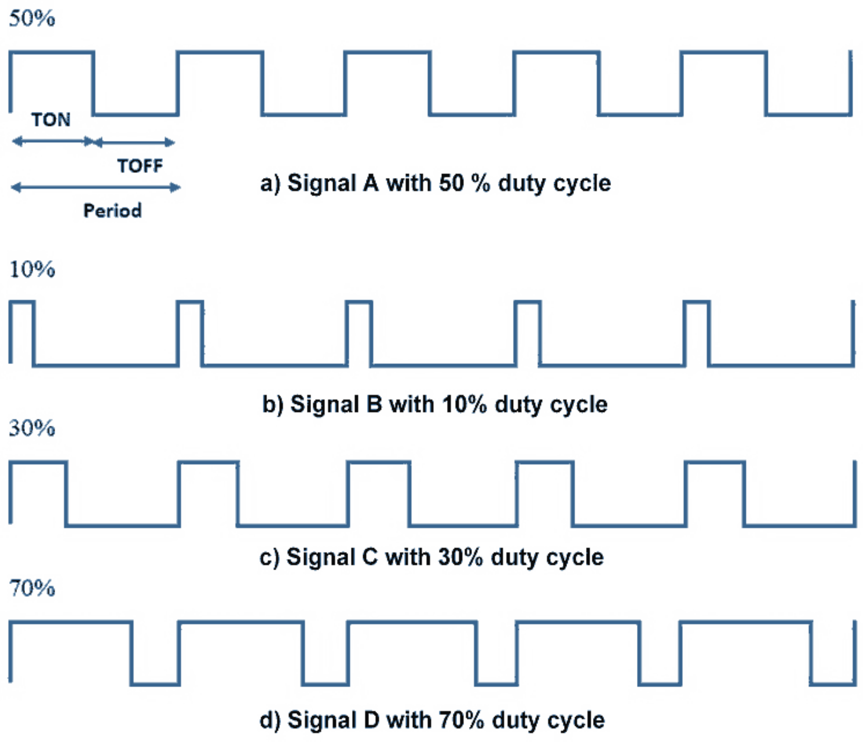
$$\text{Duty Cycle (In \%)} = \frac{Ton}{Ton + Toff} \times 100$$

E.g. Consider a pulse with a period of 10ms which remains ON (high) for 2ms.The duty cycle of this pulse will be

D = (2ms / 10ms) x 100 = 20%

Through PWM technique, we can control the power delivered to the load by using ON-OFF signal.

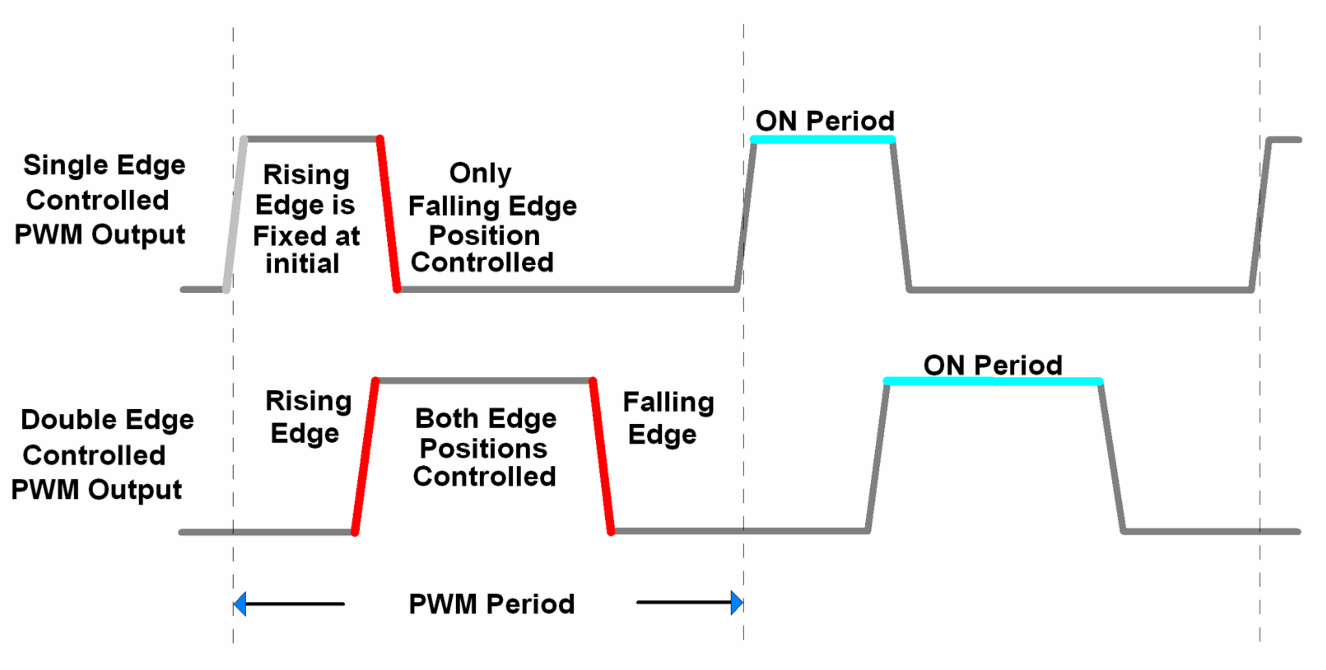
Pulse Width Modulated signals with different duty cycle are shown below.


50%
TON
TOFF
Period
a) Signal A with 50 % duty cycle
10%
b) Signal B with 10% duty cycle
30%
c) Signal C with 30% duty cycle
70%
d) Signal D with 70% duty cycle

PWM Duty Cycle Waveforms

LPC2148 has PWM peripheral through which we can generate multiple PWM signals on PWM pins. Also, LPC2148 supports two types of controlled PWM outputs as,

- **Single Edge Controlled PWM Output**
  Only falling edge position can be controlled.
- **Double Edge Controlled PWM Output**
  Both Rising and Falling edge positions can be controlled.

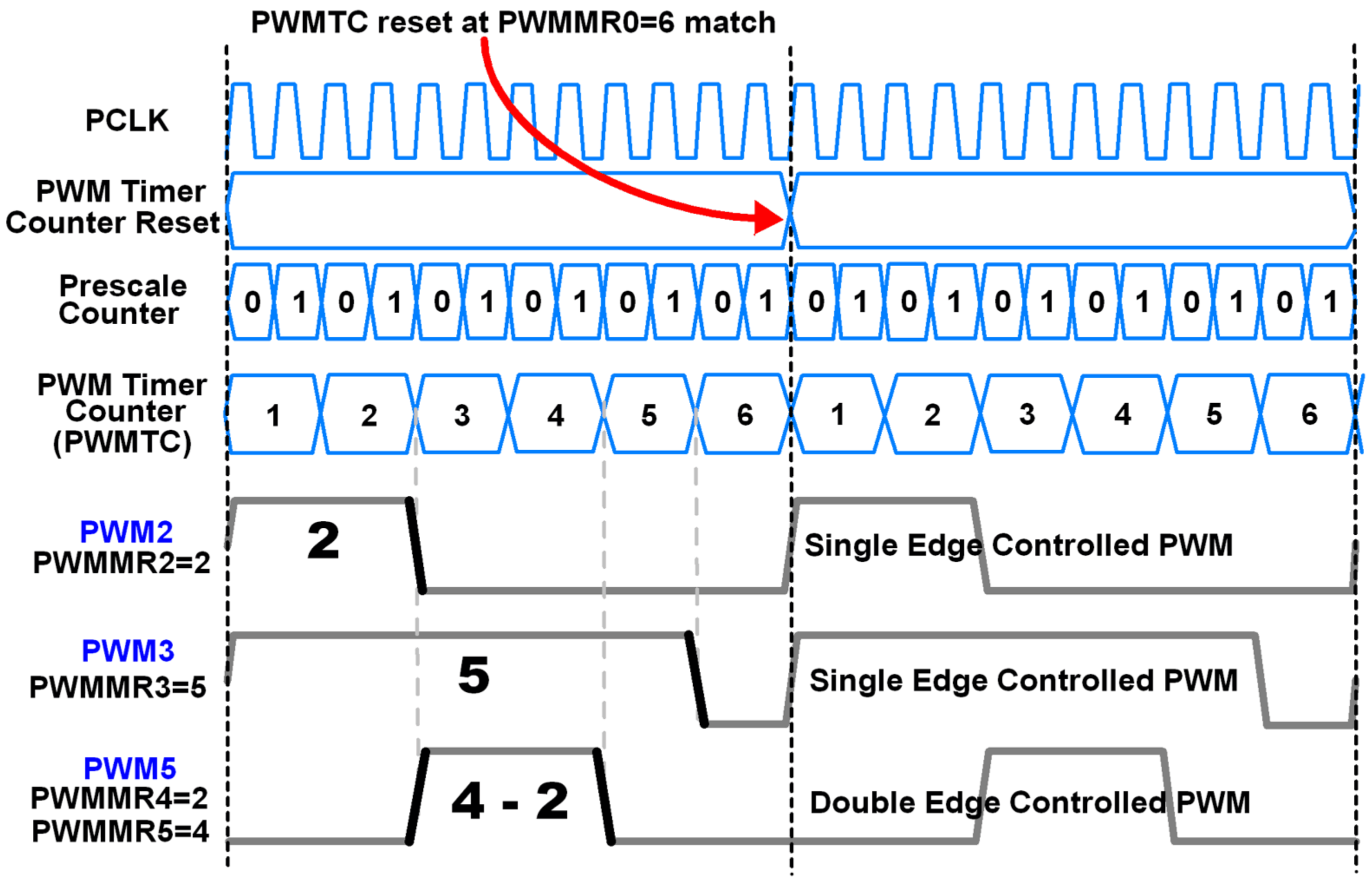

Single Edge Controlled PWM Output
Rising Edge is Fixed at initial
Only Falling Edge Position Controlled
ON Period
Double Edge Controlled PWM Output
Rising Edge
Both Edge Positions Controlled
Falling Edge
ON Period
PWM Period

Single Edge Controlled PWM : All the rising (positive going) edges of the output waveform are positioned/fixed at the beginning of the PWM period. Only falling (negative going) edge position can be controlled to vary the pulse width of PWM.

Double Edge Controlled PWM : All the rising (positive going) and falling (negative going) edge positions can be controlled to vary the pulse width of PWM. Both the rising as well as the falling edges can be positioned anywhere in the PWM period.

LPC2148 PWM

- The PWM in LPC2148 is based on standard 32-bit Timer Counter, i.e. PWMTC (PWM Timer Counter). This Timer Counter counts the cycles of peripheral clock (PCLK).
- Also, we can scale this timer clock counts using 32-bit PWM Prescale Register (PWMPR).
- LPC2148 has 7 PWM match registers (PWMMR0 – PWMMR06).
- One match register (PWMMR0) is used to set PWM frequency.
- Remaining 6 match registers are used to set PWM width for 6 different PWM signals in Single Edge Controlled PWM or 3 different PWM signals in Double Edge Controlled PWM.
- Whenever PWM Timer Counter (PWMTC) matches with these Match Registers then, PWM Timer Counter resets, or stops, or generates match interrupt, depending upon settings in PWM Match Control Register(PWMMCR).



As shown in above figure, PWMMR0 = 6 i.e. PWM period is 6 counts, after which PWM Timer Counter resets.

PWM2 & PWM3 are configured as Single Edge Controlled PWM and PWM5 is configured as Double Edge Controlled PWM.

Prescaler is set to increment PWM Timer Counter after every two Peripheral clocks (PCLK).

Match registers (PWMMR2 & PWMMR3) are used to set falling edge position for PWM2 & PWM3.

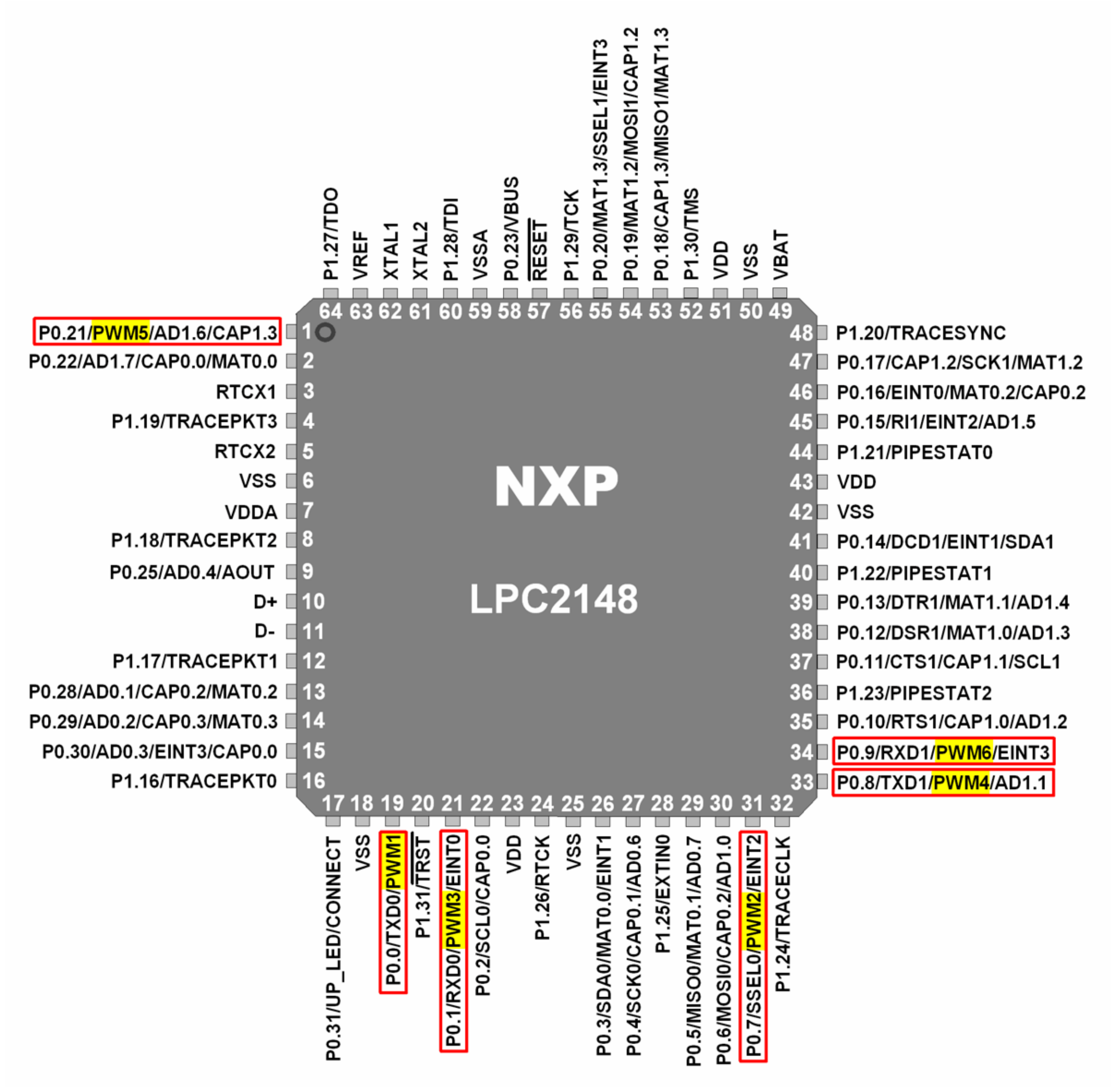
PWMMR4 & PWMMR5 are used to set rising & falling edge positions respectively for PWM5.

Let's see the different PWM that can be generated using LPC2148

The table given below shows when the PWM is Set (Rising Edge) and Reset (Falling Edge) for different PWM channels using 7 Match Register.

PWM Channel	**Single Edge Controlled**		**Double Edge Controlled**	
	Set by	**Reset by**	**Set by**	**Reset by**
1	Match 0	Match 1	Match 0	Match 1
2	Match 0	Match 2	Match 1	Match 2
3	Match 0	Match 3	Match 2	Match 3
4	Match 0	Match 4	Match 3	Match 4
5	Match 0	Match 5	Match 4	Match 5
6	Match 0	Match 6	Match 5	Match 6

LPC2148 PWM Pins



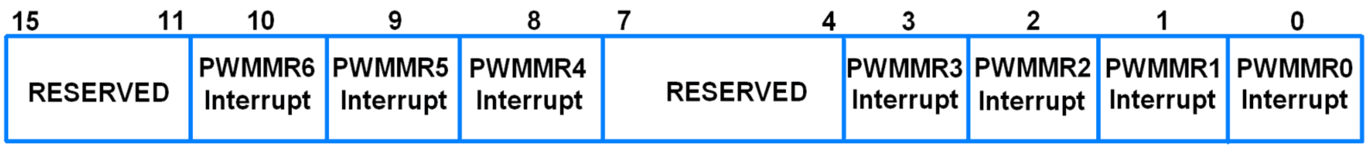
LPC2148 PWM Pins

Let’s see the various PWM registers that are useful in controlling and generating PWM.

PWM Registers

1. **PWMIR (PWM Interrupt Register)**

- It is a 16-bit register.



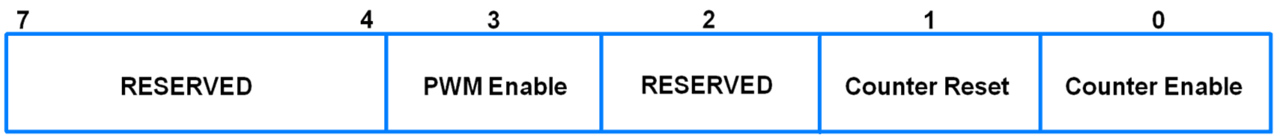
15 11	10	9	8	7 4	3	2	1	0
RESERVED	PWMMR6 Interrupt	PWMMR5 Interrupt	PWMMR4 Interrupt	RESERVED	PWMMR3 Interrupt	PWMMR2 Interrupt	PWMMR1 Interrupt	PWMMR0 Interrupt

PWMIR (PWM Interrupt Register)

- It has 7 interrupt bits corresponding to the 7 PWM match registers.
- If an interrupt is generated, then the corresponding bit in this register becomes HIGH.
- Otherwise the bit will be LOW.
- Writing a 1 to a bit in this register clears that interrupt.
- Writing a 0 has no effect.

2. **PWMTCR (PWM Timer Control Register)**

- It is an 8-bit register.
- It is used to control the operation of the PWM Timer Counter.



7 4	3	2	1	0
RESERVED	PWM Enable	RESERVED	Counter Reset	Counter Enable

PWMTCR (PWM Timer Control Register)

- **Bit 0 – Counter Enable**
  When 1, PWM Timer Counter and Prescale Counter are enabled.
  When 0, the counters are disabled.

- **Bit 1 – Counter Reset**
  When 1, the PWM Timer Counter and PWM Prescale Counter are synchronously reset on next positive edge of PCLK.
  Counter remains reset until this bit is returned to 0.

- **Bit 3 – PWM Enable**
  This bit always needs to be 1 for PWM operation. Otherwise PWM will operate as a normal timer.
  When 1, PWM mode is enabled and the shadow registers operate along with match registers.
  A write to a match register will have no effect as long as corresponding bit in PWMLER is not set.
  Note : PWMMR0 must always be set before PWM is enabled, otherwise match event will not occur to cause shadow register contents to become effective.

3. **PWMTC (PWM Timer Counter)**

- It is a 32-bit register.
- It is incremented when the PWM Prescale Counter (PWMPC) reaches its terminal count.

4. **PWMPR (PWM Prescale Register)**

- It is a 32-bit register.
- It holds the maximum value of the Prescale Counter.

5. **PWMPC (PWM Prescale Counter)**

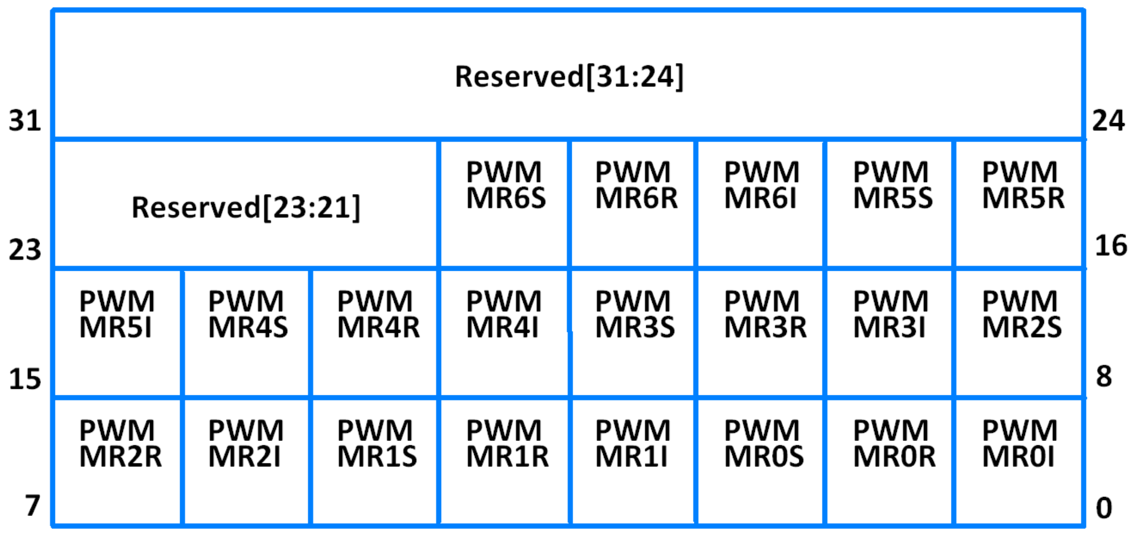
- It is a 32-bit register.
- It controls the division of PCLK by some constant value before it is applied to the PWM Timer Counter.
- It is incremented on every PCLK.
- When it reaches the value in PWM Prescale Register, the PWM Timer Counter is incremented and PWM Prescale Counter is reset on next PCLK.

6. **PWMMR0-PWMMR6 (PWM Match Registers)**

- These are 32-bit registers.
- The values stored in these registers are continuously compared with the PWM Timer Counter value.
- When the two values are equal, the timer can be reset or stop or an interrupt may be generated.
- The PWMMCR controls what action should be taken on a match.

7. **PWMMCR (PWM Match Control Register)**

- It is a 32-bit register.
- It controls what action is to be taken on a match between the PWM Match Registers and PWM Timer Counter.

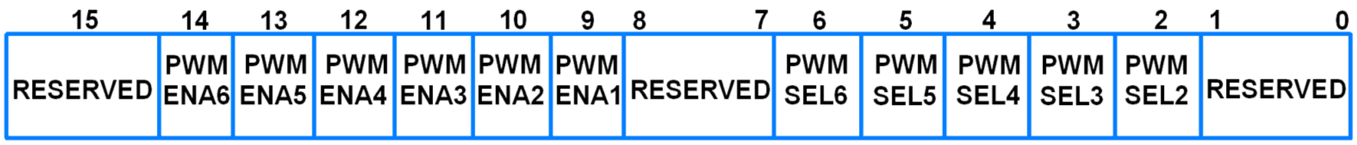


PWMMCR (PWM Match Control Register)

- **Bit 0 – PWMMR0I (PWM Match register 0 interrupt)**
  0 = This interrupt is disabled
  1 = Interrupt on PWMMR0. An interrupt is generated when PWMMR0 matches the value in PWMTC
- **Bit 1 – PWMMR0R (PWM Match register 0 reset)**
  0 = This feature is disabled
  1 = Reset on PWMMR0. The PWMTC will be reset if PWMMR0 matches it
- **Bit 2 – PWMMR0S (PWM Match register 0 stop)**
  0 = This feature is disabled
  1 = Stop on PWMMR0. The PWMTC and PWMPC is stopped and Counter Enable bit in PWMTCR is set to 0 if PWMMR0 matches PWMTC
- PWMMR1, PWMMR2, PWMMR3, PWMMR4, PWMMR5 and PWMMR6 has same function bits (stop, reset, interrupt) as in PWMMR0.

8. **PWMPCR (PWM Control Register)**

- It is a 16-bit register.
- It is used to enable and select each type of PWM.



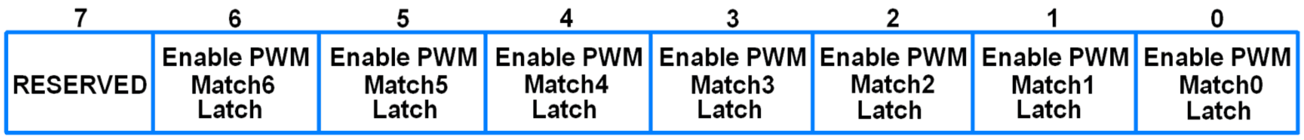
15	14	13	12	11	10	9	8–7	6	5	4	3	2	1–0
RESERVED	PWM ENA6	PWM ENA5	PWM ENA4	PWM ENA3	PWM ENA2	PWM ENA1	RESERVED	PWM SEL6	PWM SEL5	PWM SEL4	PWM SEL3	PWM SEL2	RESERVED

PWMPCR (PWM Control Register)

- **Bit 2 – PWMSEL2**
  0 = Single edge controlled mode for PWM2
  1 = Double edge controlled mode for PWM2
- All other PWMSEL bits have similar operation as PWMSEL2 above.
- **Bit 10 – PWMENA2**
  0 = PWM2 output disabled
  1 = PWM2 output enabled
- All other PWMENA bits have similar operation as PWMENA2 above.

9. **PWMLER (PWM Latch Enable Register)**

- It is an 8-bit register.



7	6	5	4	3	2	1	0
RESERVED	Enable PWM Match6 Latch	Enable PWM Match5 Latch	Enable PWM Match4 Latch	Enable PWM Match3 Latch	Enable PWM Match2 Latch	Enable PWM Match1 Latch	Enable PWM Match0 Latch

PWMLER (PWM Latch Enable Register)

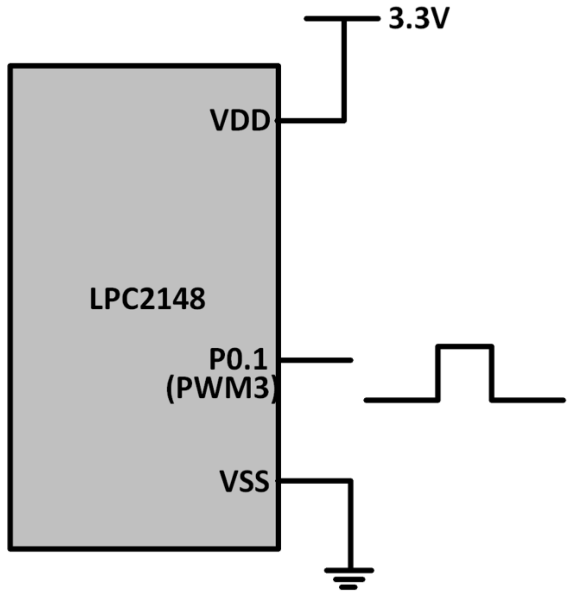
- It is used to control the update of the PWM Match Registers when they are used for PWM generation.
- When a value is written to a PWM Match Register while the timer is in PWM mode, the value is held in the shadow register. The contents of the shadow register are transferred to the PWM Match Register when the timer resets (PWM Match 0 event occurs) and if the corresponding bit in PWMLER is set.
- **Bit 6 – Enable PWM Match 6 Latch**
  Writing a 1 to this bit allows the last written value to PWMMR6 to become effective when timer next is reset by the PWM match event. Similar description as that of Bit 6 for the remaining bits.

Steps for PWM generation

- Reset and disable PWM counter using PWMTCR
- Load prescale value according to need of application in the PWMPR
- Load PWMMR0 with a value corresponding to the time period of your PWM wave
- Load any one of the remaining six match registers (two of the remaining six match registers for double edge controlled PWM) with the ON duration of the PWM cycle. (PWM will be generated on PWM pin corresponding to the match register you load the value with).
- Load PWMMCR with a value based on the action to be taken in the event of a match between match register and PWM timer counter.
- Enable PWM match latch for the match registers used with the help of PWMLER
- Select the type of PWM wave (single edge or double edge controlled) and which PWMs to be enabled using PWMPCR
- Enable PWM and PWM counter using PWMTCR

Example 1

Let’s write a Program to generate a double edge controlled PWM on PWM3 (P0.1).



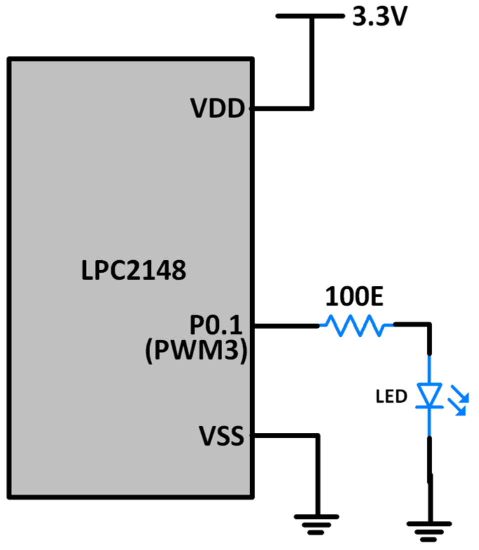
Program:

```
#include <lpc214x.h>

__irq void PWM_ISR (void)
{
        if ( PWMIR & 0x0001 )   /* If interrupt due to PWM0 */
        {
                PWMIR = 0x0001;         /* Clear PWM0 interrupt */
        }

        if ( PWMIR & 0x0002 )   /* If interrupt due to PWM1 */
        {
                PWMIR = 0x0002;         /* Clear PWM1 interrupt */
        }

        if ( PWMIR & 0x0004 )   /* If interrupt due to PWM2 */
        {
                PWMIR = 0x0004;         /* Clear PWM2 interrupt */
        }

        if ( PWMIR & 0x0008 )   /* If interrupt due to PWM3 */
        {
                PWMIR = 0x0008;         /* Clear PWM3 interrupt */
        }
        VICVectAddr = 0x00000000;
}


int main (void)
{
        VPBDIV          = 0x00000002;
        PINSEL0 = PINSEL0 | 0x00008008; /* Configure P0.1 and P0.7 as PWM3 and PWM2 respectively 
*/
        VICVectAddr0 = (unsigned) PWM_ISR; /* PWM ISR Address */
        VICVectCntl0 = (0x00000020 | 8); /* Enable PWM IRQ slot */
        VICIntEnable = VICIntEnable | 0x00000100; /* Enable PWM interrupt */
        VICIntSelect = VICIntSelect | 0x00000000; /* PWM configured as IRQ */


// For PWM3 double edge
        PWMTCR = 0x02;          /* Reset and disable counter for PWM */
```

```
	PWMPR = 0x1D;		/* Prescale value for 1usec, Pclk=30MHz*/
	PWMMR0 = 100000;	/* Time period of PWM wave, 100msec */
	PWMMR2 = 40000;	/* Rising edge of double edge controlled PWM */
	PWMMR3 = 80000;	/* Falling edge of double edge controlled PWM */
	PWMMCR = 0x00000243;		/* Reset and interrupt on MR0 match, interrupt on MR2 and 
MR3 match */
	PWMLER = 0x0D;		/* Latch enable for PWM3, PWM2 and PWM0 */
	PWMPCR = 0x0C08;	/* Enable PWM3, PWM2 and PWM0, double edge controlled PWM on 
PWM3 */
	PWMTCR = 0x09;		/* Enable PWM and counter */

	while (1);
}
```

Example 2

Let's write a Program to generate a single edge controlled PWM with varying duty cycle on PWM3 (P0.1). We can see the effect of the varying duty cycle by connecting an LED to the pin P0.1. The intensity of the LED varies with the duty cycle of PWM.



Program

```
#include <lpc214x.h>
#include <stdint.h>

void delay_ms(uint16_t j)
{
   uint16_t x,i;
         for(i=0;i<j;i++)
         {
   for(x=0; x<6000; x++);    /* loop to generate 1 milisecond delay with Cclk = 60MHz */
         }
}

__irq void PWM_ISR (void)
{
         if ( PWMIR & 0x0001 )   /* If interrupt due to PWM0 */
         {
                  PWMIR = 0x0001;          /* Clear PWM0 interrupt */
         }

         if ( PWMIR & 0x0002 )   /* If interrupt due to PWM1 */
         {
                  PWMIR = 0x0002;          /* Clear PWM1 interrupt */
         }

         if ( PWMIR & 0x0004 )   /* If interrupt due to PWM2 */
         {
                  PWMIR = 0x0004;          /* Clear PWM2 interrupt */
         }

         if ( PWMIR & 0x0008 )   /* If interrupt due to PWM3 */
         {
                  PWMIR = 0x0008;          /* Clear PWM3 interrupt */
         }
         VICVectAddr = 0x00000000;
}


int main (void)
{
         uint32_t value;
```

```
	value = 1;
	VPBDIV		= 0x00000002;
	PINSEL0 = PINSEL0 | 0x00000008; /* Configure P0.1 as PWM3 */
	VICVectAddr0 = (unsigned) PWM_ISR; /* PWM ISR Address */
	VICVectCntl0 = (0x00000020 | 8); /* Enable PWM IRQ slot */
	VICIntEnable = VICIntEnable | 0x00000100; /* Enable PWM interrupt */
	VICIntSelect = VICIntSelect | 0x00000000; /* PWM configured as IRQ */

// For single edge controlled PWM3
	PWMTCR = 0x02;		/* Reset and disable counter for PWM */
	PWMPR = 0x1D;		/* Prescale value for 1usec, Pclk=30MHz*/
	PWMMR0 = 1000;		/* Time period of PWM wave, 1msec */
	PWMMR3 = value;		/* Ton of PWM wave */
	PWMMCR = 0x00000203;		/* Reset and interrupt on MR0 match, interrupt on MR3 match
*/
	PWMLER = 0x09;		/* Latch enable for PWM3 and PWM0 */
	PWMPCR = 0x0800;	/* Enable PWM3 and PWM0, single edge controlled PWM on PWM3 */
	PWMTCR = 0x09;		/* Enable PWM and counter */

	while (1)
	{
		while (value != 999)
		{
			PWMMR3 = value;
			PWMLER = 0x08;
			delay_ms(5);
			value++;
		}
		while (value != 1)
		{
			PWMMR3 = value;
			PWMLER = 0x08;
			delay_ms(5);
			value--;
		}
	}
}
```

LPC2148 UART0

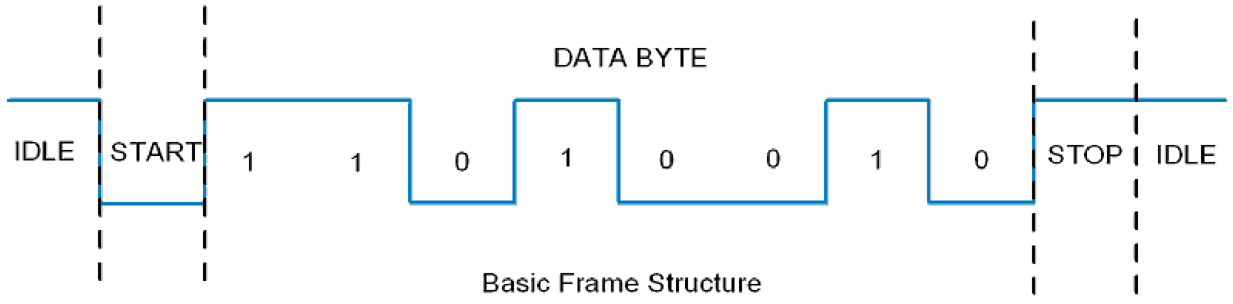
Introduction

UART (Universal Asynchronous Receiver/Transmitter) is a serial communication protocol in which data is transferred serially bit by bit at a time. Asynchronous serial communication is widely used for byte oriented transmission. In Asynchronous serial communication, a byte of data is transferred at a time.

UART serial communication protocol uses a defined frame structure for their data bytes.Frame structure in Asynchronous communication consists:

- **START bit:** It is a bit with which indicates that serial communication has started and it is always low.
- **Data bits packet**: Data bits can be packets of 5 to 9 bits. Normally we use 8 bit data packet, which is always sent after the START bit.

STOP bit: This usually is one or two bits in length. It is sent after data bits packet to indicate the end of frame. Stop bit is always logic high.



UART Frame structure

Usually, an asynchronous serial communication frame consists of a START bit (1 bit) followed by a data byte (8 bits) and then a STOP bit (1 bit), which forms a 10-bit frame as shown in the figure above. The frame can also consist of 2 STOP bits instead of a single bit, and there can also be a PARITY bit after the STOP bit.

LPC2148 UART

LPC2148 has two inbuilt UARTs available i.e. UART0&UART1. So, we can connect two UART enabled devices (GSM module, GPS module, Bluetooth module etc.) with LPC2148 at a time. UART0 and UART1 are identical other than the fact that UART1 has modem interface included.

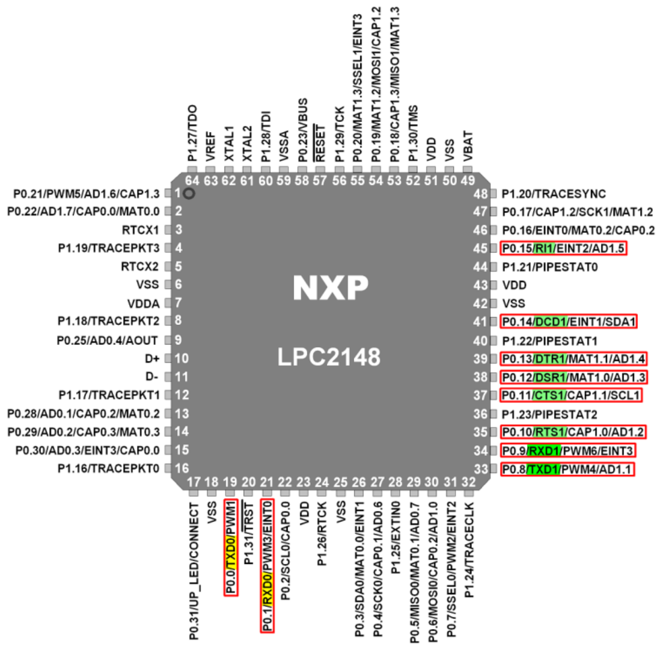
Features of UART0

- 16 byte Receive and Transmit FIFOs
- Built-in fractional baud rate generator with autobauding capabilities
- Software flow control through TXEN bit in Transmit Enable Register

Features of UART1

- 16 byte Receive and Transmit FIFOs
- Built-in fractional baud rate generator with autobauding capabilities
- Software and hardware flow control implementation possible
- Standard modem interface signals included with flow control (auto-CTS/RTS) fully supported in hardware

LPC2148 UART Pins



LPC2148 UART Pins

UART0 :

TXD0 (Output pin): Serial Transmit data pin.

RXD0 (Input pin): Serial Receive data pin.

UART1 :

TXD1 (Output pin): Serial Transmit data pin.

RXD1 (Input pin): Serial Receive data pin.

RTS1 (Output pin): Request To Send signal pin. Active low signal indicates that the UART1 would like to transmit data to the external modem.

CTS1 (Input pin): Clear To Send signal pin. Active low signal indicates if the external modem is ready to accept transmitted data via TXD1 from the UART1.

DSR1 (Input pin): Data Set Ready signal pin. Active low signal indicates if the external modem is ready to establish a communication link with the UART1.

DTR1 (Output pin): Data Terminal Ready signal pin. Active low signal indicates that the UART1 is ready to establish connection with external modem.

DCD1 (Input pin): Data Carrier Detect signal pin. Active low signal indicates if the external modem has established a communication link with the UART1 and data may be exchanged.

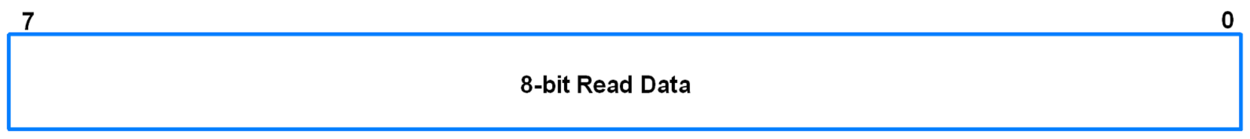
RI1 (Input pin): Ring Indicator signal pin. Active low signal indicates that a telephone ringing signal has been detected by the modem.

We will be seeing how to use UART0 here. UART1 can be used in a similar way by using the corresponding registers for UART1.

UART0 Registers

1. **U0RBR (UART0 Receive Buffer Register)**

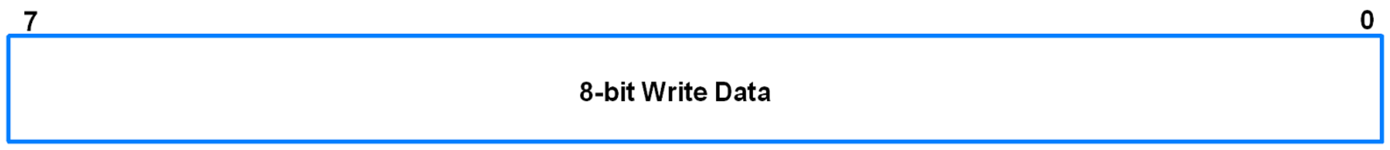
- It is an 8-bit read only register.
- This register contains the received data.
- It contains the "oldest" received byte in the receive FIFO.
- If the character received is less than 8 bits, the unused MSBs are padded with zeroes.
- The Divisor Latch Access Bit (DLAB) in U0LCR must be zero in order to access the U0RBR. (DLAB = 0)



U0RBR (UART0 Receive Buffer Register)

2. **U0THR (UART0 Transmit Holding Register)**

- It is an 8-bit write only register.
- Data to be transmitted is written to this register.
- It contains the "newest" received byte in the transmit FIFO.
- The Divisor Latch Access Bit (DLAB) in U0LCR must be zero in order to access theU0THR. (DLAB = 0)

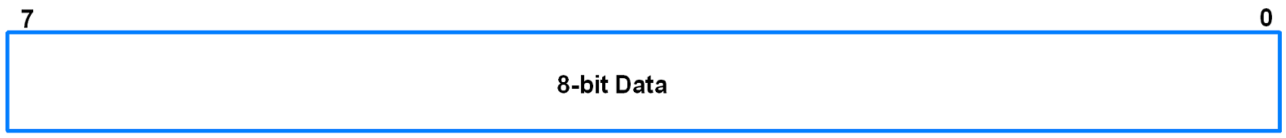


U0THR (UART0 Transmit Holding Register)

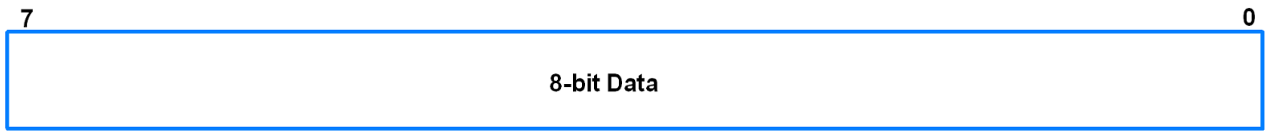
3. **U0DLL and U0DLM (UART0 Divisor Latch Registers)**

- U0DLL is the Divisor Latch LSB.
- U0DLM is the Divisor Latch MSB.
- These are 8-bit read-write registers.

- UART0 Divisor Latch holds the value by which the PCLK(Peripheral Clock) will be divided. This value must be 1/16 times the desired baud rate.
- A 0x0000 value is treated like a 0x0001 value as division by zero is not allowed.
- The Divisor Latch Access Bit (DLAB) in U0LCR must be one in order to access the UART0 Divisor Latches. (DLAB = 1)



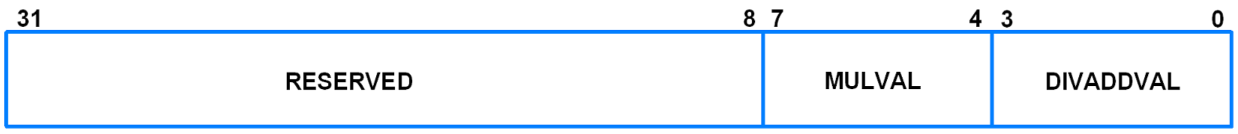
U0DLL



U0DLM

4. **U0FDR (UART0 Fractional Divider Register)**

- It is a 32-bit read write register.
- It decides the clock pre-scalar for baud rate generation.
- If fractional divider is active (i.e. DIVADDVAL>0) and DLM = 0, DLL must be greater than 3.



U0FDR (UART0 Fractional Divider Register)

- If DIVADDVAL is 0, the fractional baudrate generator will not impact the UART0 baudrate.
- Reset value of DIVADDVAL is 0.
- MULVAL must be greater than or equal to 1 for UART0 to operate properly, regardless of whether the fractional baudrate generator is used or not.
- Reset value of MULVAL is 1.
- The formula for UART0 baudrate is given below

$$UART0Baudrate = \frac{Pclk}{16 * (256 * U0DLM + U0DLL) * (1 + \frac{DIVADDVAL}{MULVAL})}$$

- MULVAL and DIVADDVAL should have values in the range of 0 to 15. If this is not ensured, the output of the fractional divider is undefined.
- The value of the U0FDR should not be modified while transmitting/receiving data. This may result in corruption of data.

5. **U0IER (UART0 Interrupt Enable Register)**

- It is a 32-bit read-write register.
- It is used to enable UART0 interrupt sources.
- DLAB should be zero (DLAB = 0).

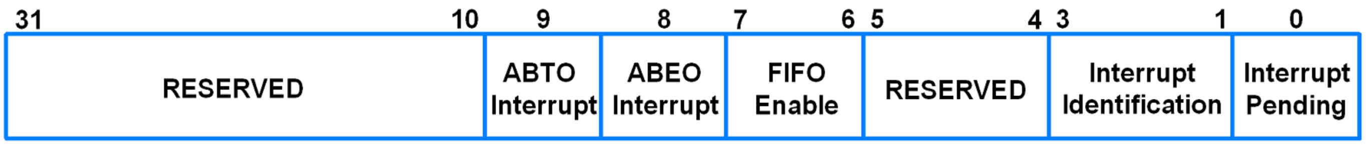
31 10	9	8	7 3	2	1	0
RESERVED	ABTO int Enable	ABEO int Enable	RESERVED	RX Status int Enable	THRE int Enable	RBR int Enable

U0IER (UART0 Interrupt Enable Register)

- **Bit 0 - RBR Interrupt Enable. It also controls the Character Receive Time-Out interrupt.**
  0 = Disable Receive Data Available interrupt
  1 = Enable Receive Data Available interrupt
- **Bit 1 - THRE Interrupt Enable**
  0 = Disable THRE interrupt
  1 = Enable THRE interrupt
- **Bit 2 - RX Line Interrupt Enable**
  0 = Disable UART0 RX line status interrupts
  1 = EnableUART0 RX line status interrupts
- **Bit 8 - ABEO Interrupt Enable**
  0 = Disable auto-baud time-out interrupt
  1 = Enable auto-baud time-out interrupt
- **Bit 9 - ABTO Interrupt Enable**
  0 = Disable end of auto-baud interrupt
  1 = Enable the end of auto-baud interrupt

6. **U0IIR (UART0 Interrupt Identification Register)**

- It is a 32-bit read only register.



U0IIR (UART0 Interrupt Identification Register)

- It provides a status code that denotes the priority and source of a pending interrupt.
- It must be read before exiting the Interrupt Service Routine to clear the interrupt.
- **Bit 0 - Interrupt Pending**
  0 = At least one interrupt is pending
  1 = No interrupts pending
- **Bit 3:1 - Interrupt Identification**
  Identifies an interrupt corresponding to theUART0 Rx FIFO.
  011 = Receive Line Status (RLS) Interrupt
  010 = Receive Data Available (RDA) Interrupt
  110 = Character Time-out Indicator (CTI) Interrupt
  001 = THRE Interrupt
- **Bit 7:6 - FIFO Enable**These bits are equivalent to FIFO enable bit in FIFO Control Register,
  0 = If FIFOs are disabled
  1 = FIFOs are enabled
- **Bit 8 - ABEO Interrupt**
  If interrupt is enabled,
  0 = No ABEO interrupt
  1 = Auto-baud has finished successfully
- **Bit 9 - ABTO Interrupt**
  If interrupt is enabled,
  0 = No ABTO interrupt
  1 = Auto-baud has timed out

7. **U0LCR (UART0 Line Control Register)**

- It is an 8-bit read-write register.
- It determines the format of the data character that is to be transmitted or received.

7	6	5	4	3	2	1 … 0
DLAB	Set Break	Stick Parity	Even Parity Select	Parity Enable	No. of Stop Bits	Word Length Select

U0LCR (UART0 Line Control Register)

- **Bit 1:0 - Word Length Select**
  00 = 5-bit character length
  01 = 6-bit character length
  10 = 7-bit character length
  11 = 8-bit character length
- **Bit 2 - Number of Stop Bits**
  0 = 1 stop bit
  1 = 2 stop bits
- **Bit 3 - Parity Enable**
  0 = Disable parity generation and checking
  1 = Enable parity generation and checking
- **Bit 5:4 - Parity Select**
  00 = Odd Parity
  01 = Even Parity
  10 = Forced “1” Stick Parity
  11 = Forced “0” Stick Parity
- **Bit 6 - Break Control**
  0= Disable break transmission
  1 = Enable break transmission
- **Bit 7 - Divisor Latch Access Bit (DLAB)**
  0 = Disable access to Divisor Latches
  1 = Enable access to Divisor Latches

8. U0LSR (UART0 Line Status Register)

- It is an 8-bit read only register.

7	6	5	4	3	2	1	0
RX FIFO Error	TEMT	THRE	BI	FE	PE	OE	RDR

U0LSR (UART0 Line Status Register)

- It provides status information on UART0 RX and TX blocks.
- **Bit 0 - Receiver Data Ready**
  0 = U0RBR is empty
  1 = U0RBR contains valid data
- **Bit 1 - Overrun Error**
  0 = Overrun error status inactive
  1 = Overrun error status active
  This bit is cleared when U0LSR is read.
- **Bit 2 - Parity Error**
  0 = Parity error status inactive
  1 = Parity error status active
  This bit is cleared when U0LSR is read.
- **Bit 3 - Framing Error**
  0 = Framing error status inactive
  1 = Framing error status active
  This bit is cleared when U0LSR is read.
- **Bit 4 - Break Interrupt**
  0 = Break interrupt status inactive
  1 = Break interrupt status active
  This bit is cleared when U0LSR is read.
- **Bit 5 - Transmitter Holding Register Empty**
  0 = U0THR has valid data
  1 = U0THR empty
- **Bit 6 - Transmitter Empty**
  0 = U0THR and/or U0TSR contains valid data
  1 = U0THR and U0TSR empty

- **Bit 7 - Error in RX FIFO (RXFE)**
  0 = U0RBR contains no UART0 RX errors
  1 = U0RBR contains at least one UART0 RX error
  This bit is cleared when U0LSR is read.

9. **U0TER (UART0 Transmit Enable Register)**

- It is an 8-bit read-write register.

7	6	5	4	3	2	1	0
TXEN	—	—	—	—	—	—	—

U0TER (UART0 Transmit Enable Register)

- The U0TER enables implementation of software flow control. When TXEn=1, UART0 transmitter will keep sending data as long as they are available. As soon as TXEn becomes 0, UART0 transmission will stop.
- Software implementing software-handshaking can clear this bit when it receives an XOFF character (DC3). Software can set this bit again when it receives an XON (DC1) character.
- **Bit 7 : TXEN**
  0 = Transmission disabled
  1 = Transmission enabled
- If this bit is cleared to 0 while a character is being sent, the transmission of that character is completed, but no further characters are sent until this bit is set again.

Programming of UART0

1. **Initialization of UART0**

- Configure P0.0 and P0.1 as TXD0 and RXD0 by writing 01 to the corresponding bits in PINSEL0.
- Using U0LCR register, make DLAB = 1. Also, select 8-bit character length and 1 stop bit.
- Set appropriate values in U0DLL and U0DLM depending on the PCLK value and the baud rate desired. Fractional divider can also be used to get different values of baudrate.
- Example,
  PCLK = 15MHz. For baud rate 9600, without using fractional divider register, from the baud rate formula, we have,
  $$9600 = \frac{15000000}{16 * (256 * U0DLM + U0DLL)} * \frac{MulVal}{MulVal + DivAddVal}$$
  On reset, MulVal = 1 and DivAddVal = 0 in the Fractional Divider Register.
  Hence, $(256xU0DLM + U0DLL) = \frac{15000000}{16 * 9600}$
  (256 x U0DLM + U0DLL) = 97.65
  We can consider it to be 98 or 97. It will make the baud rate slightly less or more than 9600. This small change is tolerable. We will consider 97.
  Since 97 is less than 256 and register values cannot contain fractions, we will take U0DLM = 0. This will give U0DLM = 97.
- Make DLA = 0 using U0LCR register.

```
void UART0_init(void)
{
        PINSEL0 = PINSEL0 | 0x00000005; /* Enable UART0 Rx0 and Tx0 pins of UART0 */
        U0LCR = 0x83;          /* DLAB = 1, 1 stop bit, 8-bit character length */
        U0DLM = 0x00;         /* For baud rate of 9600 with Pclk = 15MHz */
        U0DLL = 0x61;         /* We get these values of U0DLL and U0DLM from formula */
        U0LCR = 0x03;         /* DLAB = 0 */
}
```

2. **Receiving character**

- Monitor the RDR bit in U0LSR register to see if valid data is available in U0RBR register.

```
unsigned char UART0_RxChar(void) /*A function to receive a byte on UART0 */
{
       while( (U0LSR & 0x01) == 0); /*Wait till RDR bit becomes 1 which tells that receiver contains valid data */
                                return U0RBR;
}
```

3. **Transmitting character**

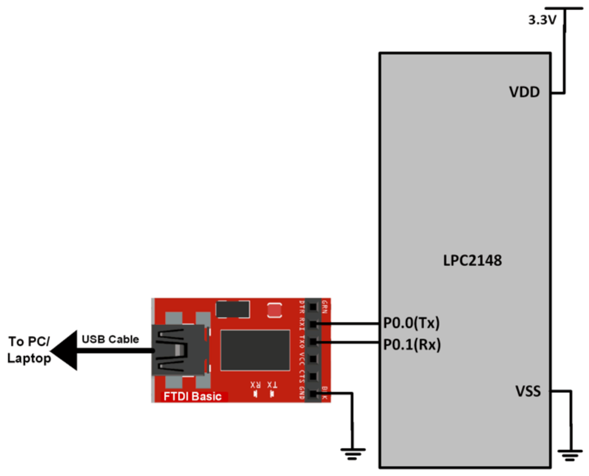
- Monitor the THRE bit in U0LSR register. When this bit becomes 1, it indicates that U0THR register is empty and the transmission is completed.

```
void UART0_TxChar(char ch) /*A function to send a byte on UART0 */
{
                U0THR = ch;
                while( (U0LSR & 0x40) == 0 );/* Wait till THRE bit becomes 1 which tells that transmission is completed */
}
```

Example

Program for echo test. LPC 2148 will receive a character through PC terminal and will transmit the same character back to the PC terminal.

Software like RealTerm, Putty can be used for this on PC.



Program

```
/*  UART0 in LPC2148(ARM7) */

#include <lpc214x.h>
#include <stdint.h>
#include "UART.h"

int main(void)
{
		char receive;
		UART0_init();
		while(1)
		{
				receive = UART0_RxChar();
				UART0_SendString("Received:");
				UART0_TxChar(receive);
				UART0_SendString("\r\n");
		}
}
```

Using Interrupt

In the above example, we have used the polling method to wait for setting of bits on completion of transmission or reception. We can use interrupts instead. Interrupts allow the controller to process other data while waiting on a certain task.

To use interrupts, we need to use the interrupt registers of UART0 and the VIC (Vectored Interrupt Controller) registers. The connections will be same as shown for above example.

Program using Interrupt

```
/*  UART0 in LPC2148(ARM7) using Interrupt */
#include <lpc214x.h>  #include <stdint.h>
char rx;
__irq void UART0_Interrupt(void);
void UART0_init(void)
{
        PINSEL0 = PINSEL0 | 0x00000005; /* Enable UART0 Rx0 and Tx0 pins of UART0 */
                U0LCR = 0x83;       /* DLAB = 1, 1 stop bit, 8-bit character length */
                U0DLM = 0x00;       /* For baud rate of 9600 with Pclk = 15MHz */
                U0DLL = 0x61;       /* We get these values of U0DLL and U0DLM from formula */
                U0LCR = 0x03; /* DLAB = 0 */
                U0IER = 0x00000003;         /* Enable THRE and RBR interrupt */
}
__irq void UART0_Interrupt(void)
{
                int iir_value; iir_value = U0IIR;   while ( !(iir_value & 0x01) );
                if( iir_value & 0x00000004 )
                {     rx = U0RBR;
                }
                                U0THR = rx;  while( (U0LSR & 0x40) == 0 );  VICVectAddr = 0x00;
}
int main(void)
{
                UART0_init();
                VICVectAddr0 = (unsigned) UART0_Interrupt;          /* UART0 ISR Address */
                VICVectCntl0 = 0x00000026;  /* Enable UART0 IRQ slot */
                VICIntEnable = 0x00000040;  /* Enable UART0 interrupt */
                VICIntSelect = 0x00000000;  /* UART0 configured as IRQ */
                while(1);
}
```

LED

HARDWARE PIN OUT		CONNECTIONS	OUTPUT
LED1	P1.16	Place a Jumper @JP3	LED’s will be Turned ON and OFF at 500ms interval.
LED2	P1.17		
LED3	P1.18		
LED4	P1.19		
LED5	P1.20		
LED6	P1.21		
LED7	P1.22		
LED8	P1.23		

LED SWITCH

HARDWARE PIN OUT				CONNECTIONS	OUTPU T
LED1	P1.16	SW5	P0.16	Place a Jumper @JP3	Turn ON and OFF the switches SW5-12, The Corresponding LED will be turned ON/OFF. I.e. LED displays the Switch Status..
LED2	P1.17	SW6	P0.17		
LED3	P1.18	SW7	P0.18		
LED4	P1.19	SW8	P0.19		
LED5	P1.20	SW9	P0.20		
LED6	P1.21	SW10	P0.21		
LED7	P1.22	SW11	P0.22		
LED8	P1.23	SW12	P0.23		

LCD

HARDWARE PIN OUT		CONNECTIONS	OUTPUT
CONTROL LINES		Turn ON DIP Switch (SW29) Pins 2-(5V-LCD) and 3-RS.	The Strings “ARM7 TYRO VER4” and " > LCD DEMO <" Will be displayed on LCD.
RS	P1.16		
RW	GND		
EN	P1.17		
DATA LINES			
D0	P1.18		
D1	P1.19		
D2	P1.20		
D3	P1.21		
D4	P1.22		
D5	P1.23		
D6	P1.24		
D7	P1.25		

KEYPAD_LCD

HARDWARE PIN OUT		CONNECTION	OUTPUT
ROW1	P0.16	Turn ON DIP Switch (SW29) Pins 2-(5V-LCD) and 3-RS. Ensure all the Slide switches (SW5-SW12) are pointing UP- WARD (HIGH side)	Press a Key (4x4 matrix key- pad) and the number will be displayed LCD
ROW2	P0.17		
ROW3	P0.18		
ROW4	P0.19		
COL1	P0.20		
COL2	P0.21		
COL3	P0.22		
COL4	P0.23		

BUZZER

HARDWARE PIN OUT		CONNECTIONS	OUTPUT
BUZZER	P0.15	Place all the jumpers (VCC and EN) of BUZZER SECTION	Buzzer sounds at an interval

ZIGBEE

Take Two ARM7 Tyro Boards. Load the Transmitter code in one board and Receiver code in an- other Board. Press RESET on both boards.

Transmitter

HARDWARE PIN OUT		CONNECTIONS	OUTPUT
U0TXD	P0.0	Turn ON DIP Switch SW29 pins ZBE+(4), U1TX(7) and U1RX(8) Alternatively ZBE+(4), U0TX(5) and U0RX(6) can be used to communicate via UART0 instead of UART1 The code should be modified to use UART0	Transmitter transmits the Slide Switch position over Zigbee
U0RXD	P0.1		
U1TXD	P0.8		
U1RXD	P0.9		

Receiver

HARDWARE PIN OUT		CONNECTIONS	OUTPUT
U0TXD	P0.0	Turn ON DIP Switch SW29 pins ZBE+(4), U1TX(7) and U1RX(8) Alternatively ZBE+(4), U0TX(5) and U0RX(6) can be used to communicate via UART0 instead of UART1 The code should be modified to use UART0	Receiver receives the switch posi- tions from transmitter and dis- plays its value in LED
U0RXD	P0.1		
U1TXD	P0.8		
U1RXD	P0.9		

STEPPER MOTOR

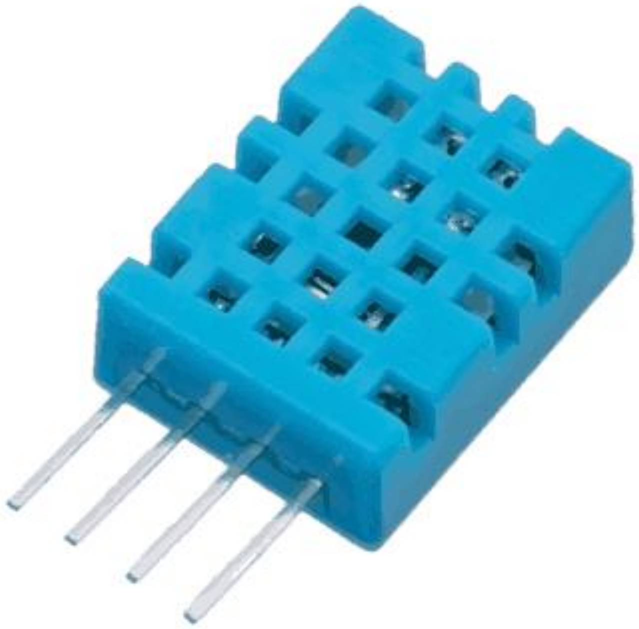
HARDWARE PIN OUT		CONNECTIONS	OUTPUT
PHASE.1	P0. 4	Connect a Stepper Motor @J10 (6 wire) Turn ON DIP Switch (SW29) Pin1 (5V-STEP).	The Stepper motor wills ro-Tate in both clockwise and counter clockwise. The direction will be changed after few rotations in either direction
PHASE.2	P0. 5		
PHASE.3	P0. 6		
PHASE.4	P0. 7		

ADC_LCD

HARDWARE PIN OUT		CONNECTIONS	OUTPUT
LM35	P0.30	Place all the Jumpers @J6 (ADC) Turn ON DIP Switch (SW29) Pins 2-(5V-LCD) and3-RS.	Digital Values of all the Three channels will be displayed in LCD Adjust the POT1 and POT2 or Apply temperature on LM35 to see the changes
POT1	P0.29		
POT2	P0.28		

DHT11 Sensor Interfacing with ARM LPC2148

Introduction



DHT11 Sensor

DHT11 is a single wire digital humidity and temperature sensor, which provides humidity and temperature values serially.

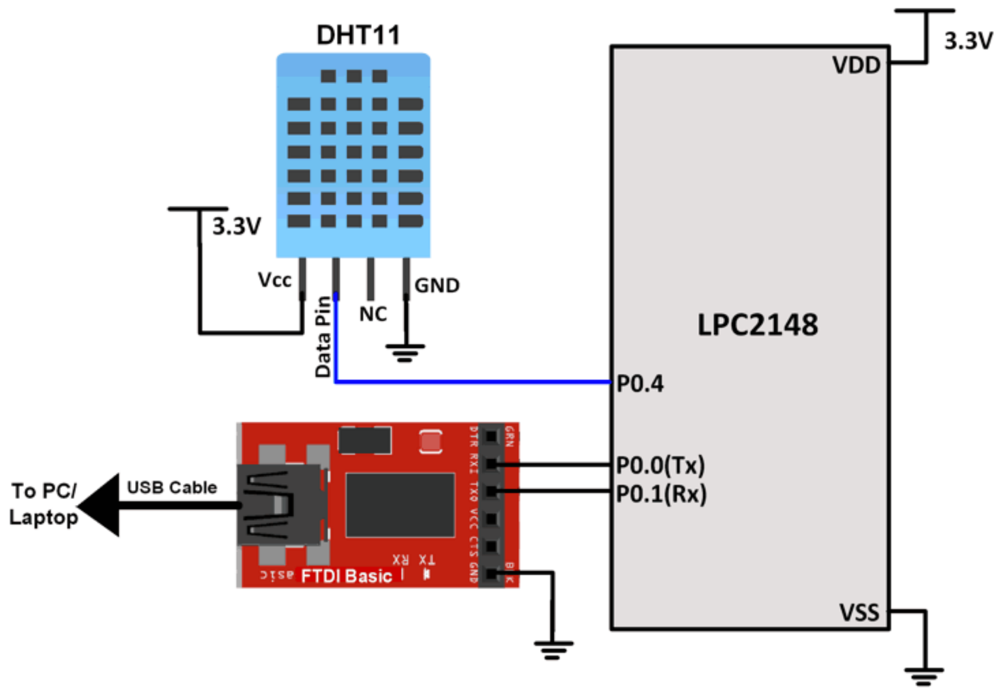
It can measure relative humidity in percentage (20 to 90% RH) and temperature in degree Celsius in the range of 0 to 50°C.

It has 4 pins of which 2 pins are used for supply, 1 is not used and the last one is used for data.

The data pin is the only pin used for communication. Pulses of different TON and TOFF are decoded as logic 1 or logic 0 or start pulse or end of the frame.

For more information about DHT11 sensor and how to use it, refer the topic DHT11 sensor in the sensors and modules topic.

Interfacing Diagram



Interfacing DHT11 Sensor with LPC2148

Example

Let’s see the example for Reading temperature and humidity from DHT11 sensor.

Here, the Data Pin of DHT11 is connected to P0.4 of LPC2148. UART0 is used for displaying data on a serial monitor on a PC/laptop.

Programming Steps

- Initialize UART0
- Make the pin connected to the sensor data pin as an output pin and transmit a Start pulse.
- Make the pin connected to the sensor data pin as an input pin and receive the Response pulse from the sensor.
- Once the response pulse is received, receive 40-bit data from the sensor.
- Display the received data on a serial terminal using UART0. In the case of checksum error, an error indication is also displayed.

Program

```
/*  DHT11 interfacing with LPC2148(ARM7)  */

#include <lpc214x.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>

void delay_ms(uint16_t j)
{
    uint16_t x,i;
        for(i=0;i<j;i++)
        {
  for(x=0; x<6000; x++);/* loop to generate 1 milisecond delay with 60MHz Cclk */
        }
}

void delay_us(uint16_t j)
{
    uint16_t x,i;
        for(i=0;i<j;i++)
        {
    for(x=0; x<7; x++);     /* loop to generate 1.04 microsecond delay with 60MHz Ccl
k */
        }
}

void UART0_init(void)
{
        PINSEL0 = PINSEL0 | 0x00000005; /* Enable UART0 Rx0 and Tx0 pins of UART
0 */
        U0LCR = 0x83;    /* DLAB = 1, 1 stop bit, 8-bit character length */
        U0DLM = 0x00;    /* For baud rate of 9600 with Pclk = 15MHz */
        U0DLL = 0x61;    /* We get these values of U0DLL and U0DLM from formula */
        U0LCR = 0x03;  /* DLAB = 0 */
}

void UART0_TxChar(char ch) /* A function to send a byte on UART0 */
{
        U0THR = ch;
        while( (U0LSR & 0x40) == 0 );     /* Wait till THRE bit becomes 1 which tells t
hat transmission is completed */
}

void UART0_SendString(char* str) /* A function to send string on UART0 */
{
        uint8_t i = 0;
        while( str[i] != '\0' )
        {
                UART0_TxChar(str[i]);
                i++;
        }
}

unsigned char UART0_RxChar(void) /* A function to receive a byte on UART0 */
{
        while( (U0LSR & 0x01) == 0);      /* Wait till RDR bit becomes 1 which tells th
at receiver contains valid data */
                return U0RBR;
}

void dht11_request(void)
{
        IO0DIR = IO0DIR | 0x00000010;   /* Configure DHT11 pin as output (P0.4 use
d here) */
        IO0PIN = IO0PIN & 0xFFFFFFEF; /* Make DHT11 pin LOW for minimum 18 seco
nds */
        delay_ms(20);
        IO0PIN = IO0PIN | 0x00000010; /* Make DHT11 pin HIGH and wait for respon
se */
}

void dht11_response(void)
{
        IO0DIR = IO0DIR & 0xFFFFFFEF;   /* Configure DHT11 pin as output */
        while( IO0PIN & 0x00000010 );   /* Wait till response is HIGH */
        while( (IO0PIN & 0x00000010) == 0 );     /* Wait till response is LOW */
        while( IO0PIN & 0x00000010 );   /* Wait till response is HIGH */     /* This is
end of response */
}

uint8_t dht11_data(void)
{
        int8_t count;
        uint8_t data = 0;
        for(count = 0; count<8 ; count++)/* 8 bits of data */
        {
                while( (IO0PIN & 0x00000010) == 0 );     /* Wait till response is LOW
*/
                delay_us(30);     /* delay greater than 24 usec */
                if ( IO0PIN & 0x00000010 ) /* If response is HIGH, 1 is received */
                        data = ( (data<<1) | 0x01 );
                else      /* If response is LOW, 0 is received */
                        data = (data<<1);
                while( IO0PIN & 0x00000010 );   /* Wait till response is HIGH (happe
ns if 1 is received) */
        }
        return data;
}

int main (void)
{
        uint8_t humidity_integer, humidity_decimal, temp_integer, temp_decimal, chec
ksum;
        char data[7];
        UART0_init();
        while(1)
        {
                dht11_request();
                dht11_response();
                humidity_integer = dht11_data();
                humidity_decimal = dht11_data();
                temp_integer = dht11_data();
                temp_decimal = dht11_data();
                checksum = dht11_data();
                if( (humidity_integer + humidity_decimal + temp_integer + temp_decima
l) != checksum )
                        UART0_SendString("Checksum Error\r\n");
                else
                {
                        UART0_SendString("Relative Humidity : ");
                        memset(data, 0, 7);
                        sprintf(data, "%d.", humidity_integer);
                        UART0_SendString(data);
                        memset(data, 0, 7);
                        sprintf(data, "%d\r\n", humidity_decimal);
                        UART0_SendString(data);
                        UART0_SendString("Temperature : ");
                        memset(data, 0, 7);
                        sprintf(data, "%d.", temp_integer);
                        UART0_SendString(data);
                        memset(data, 0, 7);
                        sprintf(data, "%d\r\n", temp_decimal);
                        UART0_SendString(data);
                        UART0_SendString("Checksum : ");
                        memset(data, 0, 7);
                        sprintf(data, "%d\r\n", checksum);
                        UART0_SendString(data);
                        delay_ms(1000);
                }
        }
}
```