LPC2148 I2C Programming Tutorial

Posted By Umang Gajera Posted date: April 10, 2017 in: Embedded, LPC2148 Tutorials No Comments

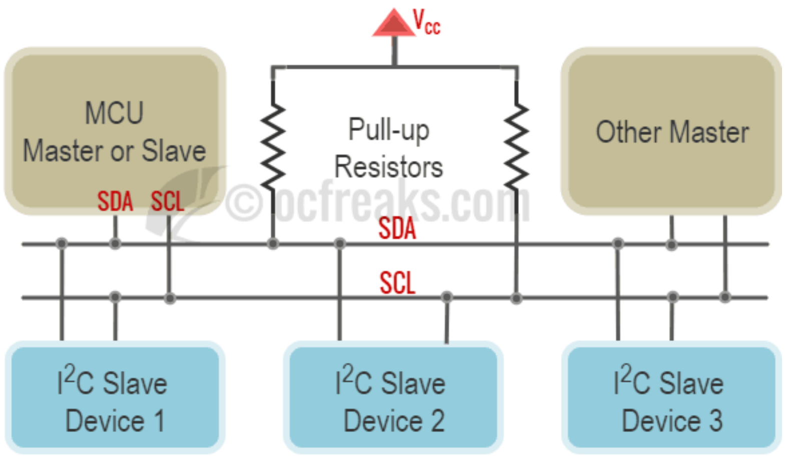
In this tutorial we will go through LPC2148 I2C programming and learn how to program it for interfacing various I2C modules, sensors and other slave devices. For those who are new to I2C Bus & Protocol I have posted an I2C Basics Tutorial @ http://www.ocfreaks.com/i2c-tutorial/

Introduction

A quick Recap of I2C

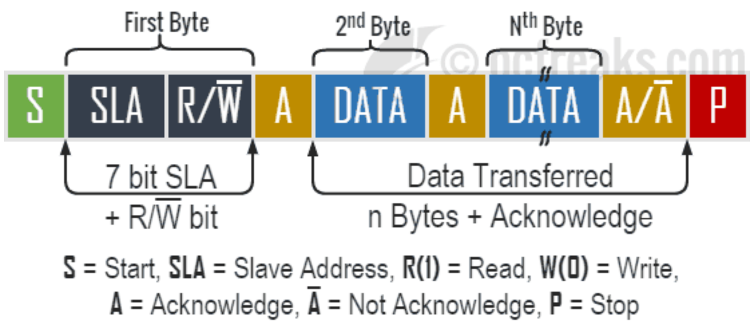
I2C was invented by Philips in 1980s. I2C stands for Inter-Integrated Circuit and also sometimes also referred as TWI i.e. Two Wire Interface since it uses only 2 wires for data transmission and synchronization. The two wires of I2C Bus consists of:
1. Data Line which is **SDA i.e. Serial Data**
2. Clock Line which is **SCL i.e. Serial Clock**

I2C uses 7bit and 10bit addresses for each device connected to the bus. 10bit addressing was introduced later. In this tutorial we will use 7bit addressing since its common with sensors, eeproms, etc. A general I2C bus topology with multiple masters and multiple slaves connected to the bus at the same time is shown below:



I2C bus is a **Byte Oriented bus**. Only a byte can be transferred at a time. Communication(Write to & Read from) is always initiated by a Master. The Master first sends a **START** condition and then writes the **Slave Address(SLA)** and the **Direction bit(Read=1/Write=0)** on bus and the corresponding Slave responds accordingly.

Format for I2C communication protocol is given as:


First Byte
2nd Byte
Nth Byte
S
SLA
R/W̄
A
DATA
A
DATA
A/Ā
P
7 bit SLA + R/W̄ bit
Data Transferred n Bytes + Acknowledge
S = Start, SLA = Slave Address, R(1) = Read, W(0) = Write,
A = Acknowledge, Ā = Not Acknowledge, P = Stop

I2C module in LPC2148 ARM7 Microcontrollers

The I2C block in LPC2148 and other LPC2100 series ARM7 MCUs can be configured as either Master, Slave or both Master & Slave. It also features a programmable clock which aids in using different transfer rates as required. The I2C block in LPC214x supports speeds up to 400kHz.

I2C has 4 operating modes:

1. Master Transmitter mode
2. Master Receiver mode
3. Slave Transmitter mode
4. Slave Receiver mode

LPC2148 ARM7 Microcontroller supports all of these 4 modes, but in this tutorial we will go through Master Transmitter and Master Receiver modes only since implementing the Slave modes is easy once you understand the Master modes and also since Master mode is used for interfacing with Sensors, LCD Displays, and other I2C slave devices.

Pins relating to I2C Module of LPC2148

For I2C0 block the **SLC(Clock)** pin is **P0.2** and **SDA(Data)** Pin is **P0.3**, while for I2C1 block the **SCL** pin in **P0.11** and **SDA** pin is **P0.14**.

Registers used for programming LPC2148 I2C block

Before we get into Operating mode details lets go through the registers used in I2C block of LPC214x:

(Replace 0 with 1 for I2C1 block registers)

1) **I2C0CONSET (8 bit) – I2C control set register:** The bits in this register control the operation of the I2C interface. Writing a 1 to a bit of this register causes the corresponding bit in the I2C control register(inside I2C block) to be set. Writing a 0 has no effect. This is a Read-Write register.

1. **Bits[0 & 1] :** Reserved
2. **Bit 2 – AA – Assert Acknowledge Flag :** When this bit is set to 1, an acknowledge (Logic low on SDA) will be returned when a data byte has been received in the master receiver mode. Similarly when AA is set to 0, a not acknowledge (Logic low on SDA) will be returned when a data byte has been received in the master receiver mode.
3. **Bit 3 – SI – I2C Interrupt Flag :** This bit is set whenever the I2C state changes(Except for state code 0xF8). When SI is set the Low Period of the serial clock is stretched which is also termed as clock stretching. When SCL is HIGH, its not affected by the state of SI flag. SI must be reset using I2CONCLR register everytime.
4. **Bit 4 – STO – STOP Flag :** When this bit is set to 1 the I2C interface will send a STOP condition.
5. **Bit 5 – STA – START Flag :** When this bit is set to 1 the I2C interface is forced to enter Master mode and send a START Condition or send a Repeated START if its already in Master mode.
6. **Bit 6 – I2EN – I2C interface Enable :** This bit is used to Enabled or Disable the I2C interface. When set to 1 the I2C interface is enabled and when set to 0 the I2C interface is disabled.
7. **Bit 7** – Reserved.

2) **I2C0CONCLR (8 bit) – I2C control clear register.** This register is used to clear bits in I2C0CONSET register. Writing 0 no effect. The bit locations are same as that of I2C0CONSET register given above. Its a Write only register.

3) **I2C0STAT (8 bit)** – This gives the current state of I2C interface in form of state codes. This is a read only register.

4) **I2C0DAT (8 bit)** – This register contains the data that is to be transmitted or the latest received data. Data in this register is always shifted from right to left i.e. the first bit to be transmitted is the MSB (bit 7), and after a byte has been received, the first bit of received data is located at the MSB of I2C0DAT.

5) **I20SCLH (16 bit)** – This register is used to store the High time period of the SCL pulse.

6) **I20SCLL (16 bit)** – This register is used to store the Low time period of the SCL pulse.

7) **I2C0ADR (8 bit)** – I2C Slave Address register : Not applicable for master mode. Used to store the address in slave mode.

LPC2148 I2C Status Codes

Before we start coding, first lets go through some status codes. Whenever an event occurs on the I2C bus a corresponding I2C status code will be set in I2CxSTAT register.

Status Codes Common to Master Transmitter & Receiver Mode:

0x08	: A **START condition** has been transmitted. Load Slave Address + Read/Write (SLA+R/W) into I2CDAT to transmit it.
0x10	: A **REPEAT START condition** has been transmitted. Load Slave Address + Read/Write (SLA+R/W) into I2CDAT to transmit it.
0x18	: Previous state was State 0x08 or State 0x10, **SLA+R/W** has been transmitted, **ACK** has been received. The first data byte will be transmitted, an **ACK[Acknowledgment](AA=0) bit will be received.**
0x20	: **SLA+R/W** has been transmitted, **NOT ACK(AA=1)** has been received. A **STOP** condition will be transmitted.

Master Transmitter Status Codes:

0x28	: Data has been transmitted, **ACK(AA=0)** has been received. If the transmitted data was the last data byte then transmit a **STOP** condition, otherwise transmit the next data byte.
0x30	: Data has been transmitted, **NOT ACK(AA=1)** received. A **STOP** condition will be transmitted.
0x38	: Arbitration has been lost while sending **Slave Address + Write or Data**. The bus has been released and not addressed Slave mode is entered. A new **START** condition will be transmitted when the bus is free again.

Master Receiver Status Codes:

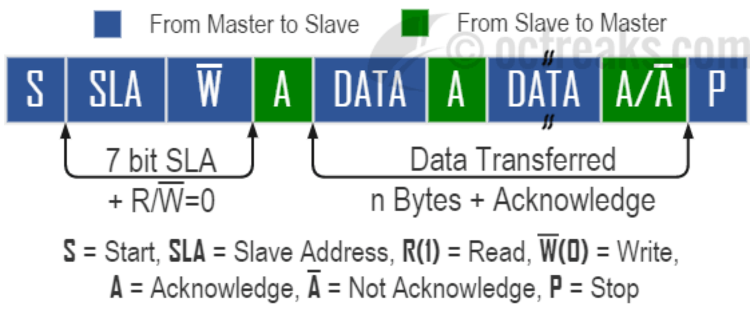
0x40	: Previous state was State 0x08 or State 0x10. **Slave Address + Read (SLA+R)** has been transmitted, **ACK** has been received. Data will be received and **ACK** returned.
0x48	: **Slave Address + Read (SLA+R)** has been transmitted, **NOT ACK** has been received. A **STOP** condition will be transmitted.
0x50	: Data has been received, **ACK** has been returned. Data will be read from **I2DAT**. Additional data will be received. If this is the last data byte then **NOT ACK** will be returned, otherwise **ACK** will be returned.
0x58	: Data has been received, **NOT ACK** has been returned. Data will be read from I2DAT. A **STOP** condition will be transmitted.

A Complete List of Status Codes, software response and what action is taken next by the I2C module on lpc214x is given in the **Datasheet(UM10120 Rev. 02) from Page 152+ onwards**. Refer the same whenever in doubt.

I2C Master Modes in LPC2148 ARM7 MCU:

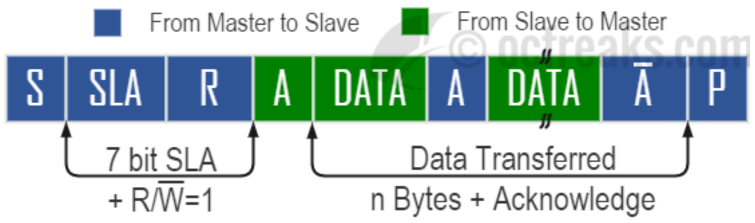
I2C Master Transmitter Mode:

To enter the Master Transmitted mode we set the **STA** bit to 1. After this the master will output a **START** condition(as soon as the bus is free) and the first byte is sent on the bus that will contain the address(7 bits) of the slave device along with the R/W bit. Here we set the R/W to 0 which means Write. After this the data sent a byte at a time. For each byte sent by master, the slave device sends a corresponding **Acknowledgement bit(AA=0)**. When slave is finished sending data or has no more data to send it will send a **'Not Acknowledge'(AA=1) bit** to indicate this. After this the master outputs a **STOP** condition. This is shown the figure below:


From Master to Slave
From Slave to Master
S
SLA
W̄
A
DATA
A
DATA
A/Ā
P
7 bit SLA + R/W̄=0
Data Transferred n Bytes + Acknowledge
S = Start, SLA = Slave Address, R(1) = Read, W̄(0) = Write,
A = Acknowledge, Ā = Not Acknowledge, P = Stop
© ocfreaks.com

I2C Master Receiver Mode:

In this mode a slave transmitter sends data to a Master Receiver. The initialization is same that we saw for Master Transmitter above, except here we set the R/W bit to 1 which means Read. The slave acknowledges it and sends data byte(s). For each byte sent by slave, the master device sends a corresponding **Acknowledgement bit(AA=0)**. If master wants to continue it can send an Acknowledge bit to salve or if master want to stop receiving data it will send a **'Not Acknowledge'(AA=1) bit** to indicate this. After this master will output a **STOP** condition. This is shown the figure below:



Each time any activity occurs on the bus the **I2C0STAT** will be loaded with a corresponding status code and the **SI** bit is also set which triggers the ISR if defined. Using these status codes we can check for successful transfers, errors on the bus or any other conditions and proceed accordingly. After taking appropriate actions(either inside ISR or outside of ISR) we must also clear the **SI** bit every time.

Procedure for I2C communication in Master Transmitter Mode:

1. After Enabling I2C block Enter master mode by Setting **STA** bit which will send the **START** condition. If already in Master mode a **REPEAT START** will be send.
2. After **START** has been sent the **SI** bit in I2C0CON will be set to 1 and value of **I2C0STAT** will be 0x08 . For **REPEAT START I2C0STAT** will be 0x10 .
3. Now load **I2C0DAT** with 7 bit **Slave Address(SLA) + R/W bit i.e. SLA+RW**. Here it will be **SLA+W**(Note: W=0, R=1). Next clear **SI** to transfer this first byte.
4. After **SLA+R/W** has been sent, an **Acknowledge/ACK(AA=0)** bit is received and **SI** bit is set again. At this point status code(s) 0x28 , 0x30 & 0x38 are possible.
5. Now depending on the Status proceed further.
  5.1 In normal situation status code will be 0x18 which means **SLA+W** has been Transmitted and **ACK(A=0)** has been Received.
  5.2 Next, load the data to be sent into **I2C0DAT** and then clear **STA**, **STO** and **SI** bits using **I2C0CONCLR** to transmit data.
  5.3 When data has been sent and an **ACK** has been received **I2C0STAT** will be 0x28 . To keep transmitting data goto step 5.2.
    5.3.1 If slave sends a **Not Acknowledge/NACK(AA=1)** bit the Status code will be 0x38 and a **STOP** condition will be transmitted.
    5.3.2 Or you can stop the transmission with a **STOP** condition by setting **STO** bit in **I2C0CONSET**.

The **STO** bit in **I2C0CONSET** auto clears after the **STOP** condition is sent. So if you want to start the communication again you will need to wait till the **STO** bit clears. In code implementations not using ISR(like in our case) this is done by monitoring the **STO** bit. After **STO** bit is reset you send a **START** condition to transmit/receive data again.

Procedure for I2C communication in Master Receiver Mode:

1. Same as in Master Transmitter Mode.
2. Same as in Master Transmitter Mode.

3. Here we load **I2C0DAT** with **Slave Address + Read bit (SLA+R)**. Clear **SI** to continue.
4. After **SLA+R** has been sent, an **ACK(AA=0)** bit is received and **SI** bit is set again. At this point status code(s) 0x38 , 0x40 & 0x48 are possible.
5. Now depending on the Status proceed further.
   5.1 In normal situation **I2C0STAT** will be 0x40 which means **SLA+R** has been Transmitted and **ACK** has been Received.
   5.2 Now to receive data from Slave, set the **AA** bit and clear **SI** bit.
   5.3 If Data has been sent and **ACK(AA=0)** has been returned the status code will be 0x50 .
      5.3.1 To keep on receiving data, keep on setting **AA** bit everytime.
      5.3.2 To stop receiving data, send a **NOT ACK(AA=1)** by clearing **AA** bit after which status code will be 0x58 which means Data has been send and **NOT ACK(A=1)** has been returned.

LPC2148 I2C ARM7 Setup & Programming

Implementation Note: We can either implement the I2C code using an ISR which handles every thing =or= we can implement a state driven code in which we do not use ISR but instead use functions to handle the events by waiting for **SI** bit to be set and then clearing it to trigger next action. We can also implement it using a mix of ISR and event functions.
In our case we will NOT implement any ISR but will be implement a state driven code. In my opinion this makes it easier to code and understand.

In order to communicate with any I2C device we need to set the **I2C clock frequency**. The I2C Clock/bit frequency is set using 2 registers: **I2CxSCLH and I2CxSCLL**. I2CxSCLH defines the number of PCLK(Peripheral Clock) cycles for the I2C Clock(SCL) High time while I2CxSCLL defines the number of PCLK cycles for the I2C Clock(SCL) Low time. It is given a simple formula as given below :

Note that LPC214x ARM7 Microcontrollers support a maximum I2C frequency of 400KHz. In our case we will use a frequency of 100KHz. Given our PCLK is running at 60Mhz we need to set **I2C0SCLH** = **I2C0SCLL = 300 for IC20 Module**.

Now lets, define some bits which will help us setup the **I2C0ONCLR** and **I2C0CONSET** registers to initialize the I2C0 block before we can use it.

```
#define I2EN (1<<6) //Enable/Disable bit
#define STA  (1<<5) //Start Set/Clear bit
#define STO  (1<<4) //Stop bit
#define SI   (1<<3) //Serial Interrupt Flag Clear bit
#define AA   (1<<2) //Assert Acknowledge Set/Clear bit
```

Now, we will define some basic 'building block' functions which will help us in programming the I2C module without complicating it too much.

1. I2C initialization function – `I2C0Init()`: It first select the I2C function of the respective pins. Then it configures the I2C bit rate(I2C bus clock), clears I2C0CONCLR register and then finally enables the I2C0 block using I2C0CONSET register.

```
void I2C0Init(void)
{
   PINSEL0 |= (0<<7)|(1<<6)|(0<<5)|(1<<4); //Select SCL0(P0.2) and SDA0(P0.3)
   I2C0SCLL = 300;
   I2C0SCLH = 300; //I2C0 @ 100Khz, given PCLK @ 60Mhz
   I2C0CONCLR = STA | STO | SI | AA; //Clear these bits
   I2C0CONSET = I2EN; //Enable I2C0
   //After this we are ready to communicate with any other device connected to t
}
```

2. SI wait function – `I2C0WaitForSI(void)`: This functions waits for the SI bit to be set after any action taken by the I2C hardware. When this bit is set it indicates that the action taken by I2C module has been completed i.e. a new event has occurred which in turn changes the status code in I2C0STAT .

```
bool I2C0WaitForSI(void) //Wait till I2C0 block sets SI
{
    int timeout = 0;
    while ( !(I2C0CONSET & SI) ) //Wait till SI bit is set. This is important!
    {
        timeout++;
        if (timeout > 10000) return false; //In case we have some error on bus
    }
    return return; //SI has been set
}
```

3. Send START/Repeat-START function – `I2C0SendStart()`: This functions sends a START condition as soon as the bus becomes free =or= sends a Repeat START is already in master mode and then waits till SI bit set.

```
void I2C0SendStart(void)
{
    I2C0CONCLR = STA | STO | SI | AA; //Clear everything
    I2C0CONSET = STA; //Set start bit to send a start condition
    I2C0WaitForSI(); //Wait till the SI bit is set
}
```

4. Send STOP Function – `I2C0SendStop()`: This function sends a STOP condition and then waits for the STO bit in I2C0CONSET to auto clear which indicates was sent on the bus.

```
void I2C0SendStop(void)
{
    int timeout = 0;
    I2C0CONSET = STO ; //Set stop bit to send a stop condition
    I2C0CONCLR = SI;
    while (I2C0CONSET & STO) //Wait till STOP is send. This is important!
    {
        timeout++;
        if (timeout > 10000) //In case we have some error on bus
        {
            printf("STOP timeout!\n");
            return;
        }
    }
}
```

5. I2C Transmit Byte function – `I2C0TX_Byte(unsigned char)`: This function sends a byte on the I2C bus and then waits for SI to be set.

```
void I2C0TX_Byte(unsigned char data)
{
    I2C0DAT = data;
    I2C0CONCLR = STA | STO | SI; //Clear These to TX data
    I2C0WaitForSI(); //wait till TX is finished
}
```

6. I2C Receive Byte function – `I2C0RX_Byte(bool)`:

```
unsigned char I2C0RX_Byte(bool isLast)
{
    if(isLast) I2C0CONCLR = AA; //Send NACK to stop; I2C block will send a STOP a
    else       I2C0CONSET = AA; //Send ACK to continue
    I2C0CONCLR = SI; //Clear SI to Start RX
    I2C0WaitForSI(); //wait till RX is finished
    return I2C0DAT;
}
```

Now armed with the I2C communication building block functions we can interface LPC2148 in Master Transmitter or Master Receiver mode with any slave device.

LPC2148 I2C Example: Interfacing 24LC64 EEPROM

Now, lets do an I2C programming example where we Write and Read to an EEPROM. I'll be using 24LC64 for this example. Make sure you refer its datasheet- just in case 😉

Here is the connection diagram between LPC2148 Microcontroller and EEPROM:

In order to Read and Write to EEPROM we will define 1 macro and 2 functions as follows:

1. `checkStatus(int)` **Macro function:** This functions checks if the current value in I2C0STAT is same as argument supplied to. If not it will send STOP condition and make the calling function return false which indicates an error condition.

```
#define checkStatus(statusCode) \
if(I2C0STAT!=statusCode) \
{ \
   printf("Error! Expected status code: %i(decimal), Got: %i(decimal)\n",statusC
   I2C0SendStop(); return false; \
}
```

2. `I2C0WriteEEPROM(...)` **function:** This writes data from buffer to the I2C slave device. It takes 3 arguments : `startDataAddress` – the starting address inside EEPROM where the writes must begin from , `data` – a pointer to data buffer which contains data to be written to eeprom, `length` – length of the data buffer.

```
/*(C) Umang Gajera | Power_user_EX - www.ocfreaks.com 2011-17. LPC2148 I2C Tutor
More Embedded tutorials @ www.ocfreaks.com/cat/embedded*/
bool I2C0WriteEEPROM(unsigned int startDataAddress, unsigned char *data, int len
{
   for(int count=0 ; count< length ; count++ ) { I2C0SendStart(); //Send START o
      checkStatus(0x28); //High byte has been sent and ACK recevied

      I2C0TX_Byte(startDataAddress & 0xFF); //Now send the Low byte of word Addr
      checkStatus(0x28); //Low byte has been sent and ACK recevied

      I2C0TX_Byte(data[count]); //Finally send the data byte.
      checkStatus(0x28); //Data Byte has been sent and ACK recevied

      startDataAddress++; //Increment to next address
      I2C0SendStop(); //Send STOP since we are done.

      //Now initiate write acknowledge polling as given on page 9 of 24LC64's da
      const int retryTimeout = 100;
      for(int i=0; i < retryTimeout; i++)
      {
         I2C0SendStart();
         checkStatus(0x08);
         I2C0TX_Byte(I2CSlaveAddr & 0xFE);
         if(I2C0STAT == 0x18) //ACK recieved which indicates completion of write
         {
            I2C0SendStop();
            //printf("Write Completed! for data = %c\n",data[count]);
            goto OUT;
```

3. `I2C0ReadEEPROM(...)` **function:** This reads data to buffer from the I2C slave device. Arguments are similar to those of `I2C0WriteEEPROM();`

```
/*(C) Umang Gajera | Power_user_EX - www.ocfreaks.com 2011-17. LPC2148 I2C Tutor
More Embedded tutorials @ www.ocfreaks.com/cat/embedded*/
bool I2C0ReadEEPROM(unsigned int startDataAddress, unsigned char *data , int len
{
   unsigned char RXData = 0;
   for(int i=0; i < length;i++) { I2C0SendStart(); //Send START on the Bus to En
      checkStatus(0x28); //High byte has been sent and ACK recevied

      I2C0TX_Byte(startDataAddress & 0xFF); //Now send the Low byte of word Addr
      checkStatus(0x28); //Low byte has been sent and ACK recevied

      startDataAddress++; //Increment to next address

      I2C0SendStart(); //Send Repeat START, since we are already in Master mode
      checkStatus(0x10); //Repeat START sent

      I2C0TX_Byte(I2CSlaveAddr | 0x01); //This makes SLA-RW bit to 1 which indic
      checkStatus(0x40); //SLA-R has been Transmitted and ACK received.

      if(i != length-1) RXData = I2C0RX_Byte(false); //Send NACK for last byte t
      else RXData = I2C0RX_Byte(true); //Send ACK for byte other than last byte

      data[count++] = RXData; //Write recieved data to buffer
      printf("Data='%c' ",RXData);

   }
   return true;
}
```

Finally here is the code for `main()` function:

Note that I have used UART0 to send the `printf()` output to serial console hence in order to view the output you must connect the UART0 pins on your LPC214x to your computer/laptop using a suitable USB to Serial convertor (I am using FTDI(FT232) based USB to Serial module). The CPU clock and PCLK are both configured @ 60Mhz using `initClocks()` function. Source code for these functions including `initUART0()` are present in the header file `ocfreaks_sh.h` which is the support header file for all my LPC2148 tutorials.

```
/*(C) Umang Gajera | Power_user_EX - www.ocfreaks.com 2011-17. LPC2148 I2C Tutor
More Embedded tutorials @ www.ocfreaks.com/cat/embedded*/
#include <lpc214x.h>
#include "ocfreaks_sh.h" //This contains code for UART, printf(), initClocks()
#include <stdint.h>

#define I2EN (1<<6) //Enable/Disable bit
#define STA  (1<<5) //Start Set/Clear bit
#define STO  (1<<4) //Stop bit
#define SI   (1<<3) //Serial Interrupt Flag Clear bit
#define AA   (1<<2) //Assert Acknowledge Set/Clear bit

void I2C0Init(void);
bool I2C0WaitForSI(void);
void I2C0SendStart(void);
void I2C0SendStop(void);
void I2C0TX_Byte(unsigned char data);
unsigned char I2C0RX_Byte(int isLast);

bool I2C0ReadEEPROM(unsigned int addresss, unsigned char *data, int length);
bool I2C0WriteEEPROM(unsigned int address, unsigned char *data, int length);

unsigned char I2CSlaveAddr = 0xA0; //Address of EEPROM (A2=0,A1=0,A0=0)

int main(void)
{
   initClocks();
   initUART0();
```

Serial Output:

Download Project Source / I2C Sample code @ LPC214x I2C Tutorial.zip [Successfully tested on Keil UV5.23]

LPC2148 Development Boards that we Recommend:

Share this:

Share

Tags: lpc2148 tutorial

Like Tweet Share Share Share

« **Previous**
I2C Tutorial

Next »
DIY 3D Printer Build Chapter 2: The Body / Structure

ABOUT THE AUTHOR

Umang Gajera

9 Comments OCFreaks! 1 Login

Recommend Tweet Share Sort by Newest

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

Sagar Kokate • 6 months ago
Hello Sir,
nice explaination, can you explain same thing with , multimaster scenario how sync takes place in multi master can you explain
• Reply • Share ›

Vinayak • a year ago
Hello Sir,
Great post I was wondering if you have any article which explains the ATSAMG55 and I2C interface using physical address (without any in built function).

Thanks in Advance.
• Reply • Share ›

Sunil • 2 years ago
How can we know transmitted data is stored successfully in EEPROM?
• Reply • Share ›

Power_user_EX Mod → Sunil • 2 years ago • edited
Hi, Sunil. As mentioned in the tutorial, a status code of **0x28** (in Master Receiver Mode) denotes a condition where the data was sent and an acknowledge was received for the same (assuming there are no bugs in your code). A simple method to cross check is to read the data back. While reading always initialize the char variable (or array) to **NULL** so you can check if you indeed got the desired data back. I have done this in the example as:

```
unsigned char bufferRead[BUFF_SIZE] = {0};
```

Hope this helps,
-Regards.
• Reply • Share ›

Sunil → Power_user_EX • 2 years ago
so there is error handling mechanism in I2C right? in CAN protocol we error handling mechanism (CRC error)
• Reply • Share ›

Mohan Raj • 2 years ago
hi sir its very useful for me thanks alot to u
• Reply • Share ›

Power_user_EX Mod → Mohan Raj • 2 years ago
Thanks a lot Mohan!
• Reply • Share ›

Mayank Sharma • 2 years ago
Sir,
Please give a tutorial on CAN controller.
• Reply • Share ›

Power_user_EX Mod → Mayank Sharma • 2 years ago
Hi Mayank, I am myself new to CAN Module. I can't say for sure when I will be able to post CAN Tutorial.
• Reply • Share ›

ALSO ON **OCFREAKS!**

Interfacing LM35 Sensor with LPC2148
2 comments • 2 years ago
Power_user_EX — Hi Sachin. What error ? Did you download the complete project source and try? You need to add the files "lib_funcs.h" and

LPC1768 Timer Input Capture & Frequency Counter Tutorial
1 comment • a year ago
Min Peng — hello, thanks for your sharing. Can you show me the picture of the board that you used the code? I am using the Mbed LPC1768

QuantumZERO QZ-HD01 USB3.0 SATA Hard Drive Dock Review
1 comment • 2 years ago

LPC1114 and LPC1343 LPC_IOCON Register Tutorial
1 comment • 2 years ago

reachdpk — how much did you pay for it ?

kunaal saini — Dear sir ,
Greetings of the dayi am trying your GPIO code on LPC1112, and trying to simulate on proteus , but