SYMBIOSIS
INTERNATIONAL (DEEMED UNIVERSITY)

EMBEDDED SYSTEMS AND DESIGN MINI PROJECT

SUPERVISED BY
Dr. Bhaskar Thakkar

Name- Akash Singh, PRN- 16070123010
Mihiir Prabhu, PRN-16070123057
Arjun Patidar, PRN-16070123023

Certificate

This is to certify that Students - **Akash Singh, Mihiir Prabhu & Arjun Patidar** of PRN - **16070123010, 16070123057 & 16070123023** of the class Third Year & Branch/ Div - **Electronics & Telecommunications / Division 'A'** has completed all the above tasks mentioned in this **Problem Based Learning** in **Embedded Systems and Design** of the **Semester VI** as prescribed by the **SYMBIOSIS INTERNATIONAL UNIVERSITY** in the Academic Year **2019**.

....

Sign **Date**

ACKNOWLEDGEMENT

I would like to express my special thanks of gratitude to our professor Dr. Bhaskar Thakkar who gave me the golden opportunity to do this wonderful project (Problem Based Learning) on the Embedded Systems - Interfacing MPU 6050 using LPC2148, which also helped me in doing a lot of Research and i came to know about so many new things I am really thankful to them.

Secondly i would also like to thank my parents and friends who helped me a lot in finalising this project within the limited time frame.

Embedded System Design

TY B. Tech. E&TC | Batch 2016-20

Mini Project

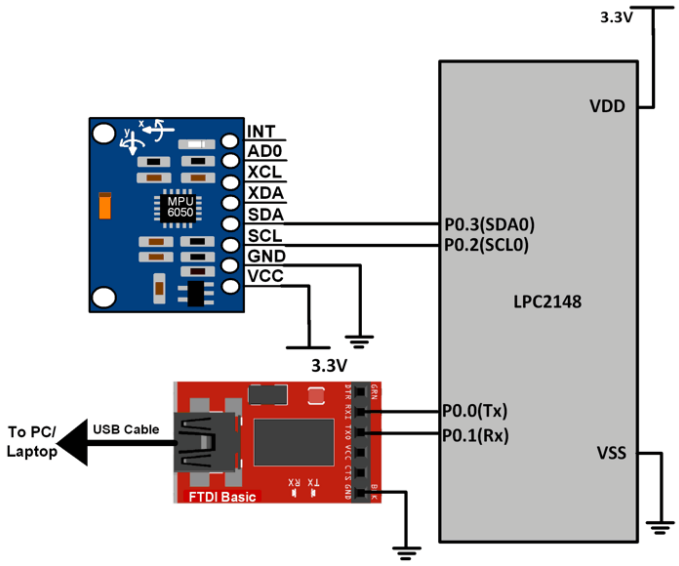
Name of the Student: Akash Singh		**PRN: 16070123010**
Name of the Student: Mihiir Prabhu		**PRN: 16070123057**
Name of the Student: Arjun Patidar		**PRN: 16070123023**
Academic Year : 2017-18	**Division: A**	

Title: Interfacing MPU 6050 using LPC2148.

Objective: -

- In this project, our objective is to make a program on Keil uVision to interface MPU 6050 using LPC2148 from Rhydolabz to display the data of Accelerometer, Gyroscope & Temperature Sensor on the Flash Magic Terminal Window.

Interfacing Diagram



Peripherals used:-

1. **MPU 6050 Sensor**
2. **LPC 2148 Stick**
3. **BreadBoard**
4. **Male to Female wires- 2**
5. **Male to Male wires- 2**

ARM Peripherals Description:-

1. **MPU 6050 Sensor**

MPU6050 sensor module is an integrated 6-axis Motion tracking device.

- It has a 3-axis Gyroscope, 3-axis Accelerometer, Digital Motion Processor and a Temperature sensor, all in a single IC.
- It can accept inputs from other sensors like 3-axis magnetometer, pressure sensor using its Auxiliary I2C bus.
- If external 3-axis magnetometer is connected, it can provide complete 9-axis Motion Fusion output.
- A microcontroller can communicate with this module using I2C communication protocol.
- Gyroscope and accelerometer reading along X, Y and Z axes are available in 2's complement form. Temperature reading is available in signed integer form.
- Gyroscope readings are in degrees per second (dps) unit; Accelerometer readings are in g unit; and Temperature reading is in degrees Celsius.

2. **LPC 2148 Stick**

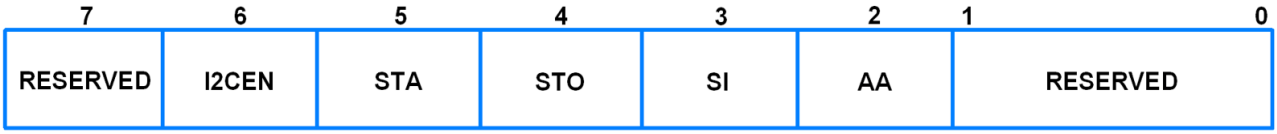
LPC2148 Stick consists of all basic components required to quickly evaluate and demonstrate the capabilities of NXP LPC2148 (ARM7TDMI) microcontroller. LPC2148 is a single-chip 16/32-bit RISC Microcontroller with 512KB on-chip Flash ROM with In-System Programming (ISP) and In-Application Programming (IAP) 32KB RAM having Interrupt Controller, Two 10bit ADCs with 14 channels, USB 2.0 Full Speed Device Controller, 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.

The Stick board is designed and populated with USB connector, voltage regulator, RTC crystal, Main Crystal and necessary de-coupling capacitors in a breadboard compatible form factor with access to all Port pins for external connection. The voltage regulation circuitry of the stick is designed on LD1117 LDO regulator, generating 3V3 for the controller. Power supply input can be either from the VCC pinout or the USB Supply (Use J1 for appropriate power input selection). The design also includes a reset switch to avoid programming pitfalls and to manually bring back the system to the initialisation mode. LPC2148 has inbuilt ISP which means we can program it within the system using the USB interface.

Registers Description:-

I2C Registers-

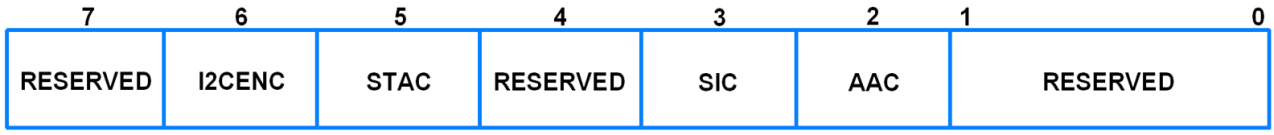
1. I2C0CONSET (I2C0 Configuration Set Register)
   - It is an 8-bit read-write register.
   - It is used to control the operation of the I2C0 interface.



I2C0CONSET (I2C0 Configuration Set Register)

- Writing a 1 to a bit in this register causes corresponding bit in the I2C control register to set.
- Writing a 0 has no effect.
- Bit 2 – AA (Assert Acknowledge Flag)
- When set to 1, Acknowledge (SDA LOW) is returned during acknowledge clock pulse on SCL. Otherwise, Not acknowledge (SDA HIGH) is returned.
- Bit 3 – SI (I2C Interrupt Flag)
- This bit is set when the I2C state changes. Else it remains reset.
- Bit 4 – STO (Stop Flag)
- In Master Mode, setting this bit causes the I2C interface to transmit a STOP condition. In Slave Mode, setting this bit causes recover from an error condition if any.
- This bit is cleared by hardware automatically.
- Bit 5 – STA (Start Flag)
- Setting this bit causes the I2C interface to enter in master mode and transmit a START condition or transmit a repeated START condition if it is already in master mode.
- Bit 6 – I2CEN (I2C Interface Enable)
- Set this bit to enable I2C interface.

2. I2C0CONCLR (I2C0 Configuration Clear Register)
   - It is an 8-bit write only register.

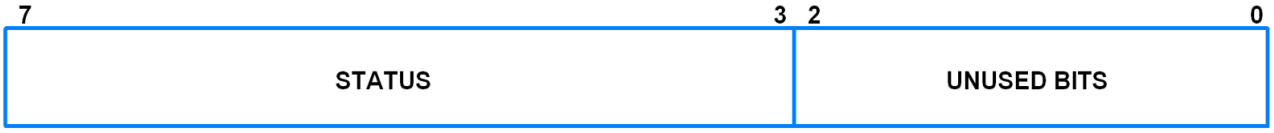


I2C0CONCLR (I2C0 Configuration Clear Register)

- Writing a 1 to a bit in this register causes corresponding bit in the I2C control register to clear.
- Writing a 0 has no effect.
- Bit 2 – AAC (Assert Acknowledge Flag Clear)
- Setting this bit clears the AA bit in I2C0CONSET register.
- Bit 3 – SIC (I2C Interrupt Flag Clear)
- Setting this bit clears the SI bit in I2C0CONSET register.
- Bit 5 – STAC (Start Flag Clear)
- Setting this bit clears the STA bit in I2C0CONSET register.
- Bit 6 – I2CENC (I2C Interface Disable)

- Setting this bit clears the I2CEN bit in I2C0CONSET register.

3. I2C0STAT (I2C0 Status Register)
   - It is an 8-bit read only register.
   - It reflects the present status of the I2C interface.



I2C0STAT (I2C0 Status Register)
- Bit 2:0 – Unused bits
- These are unused bits and are always 0.
- Bit 7:3 – Status
- These are status bits which provide the status of the current event on I2C bus.
- There are 26 possible status codes.
- Example, I2C0STAT = 0x08. This code indicates that a start condition has been transmitted.
- For other status codes, refer Table 232 (Page 228) to table 235 of the datasheet given in the attachments section below.

4. I2C0DAT (I2C0 Data Register)
   - It is an 8-bit read-write register.

I2C0DAT (I2C0 Data Register)
- It contains the data to be transmitted or received.
- This register can be read or written to only when SI = 1.

5. I2C0SCLL (I2C0 SCL Low Duty Cycle Register)
   - It is a 16-bit register.

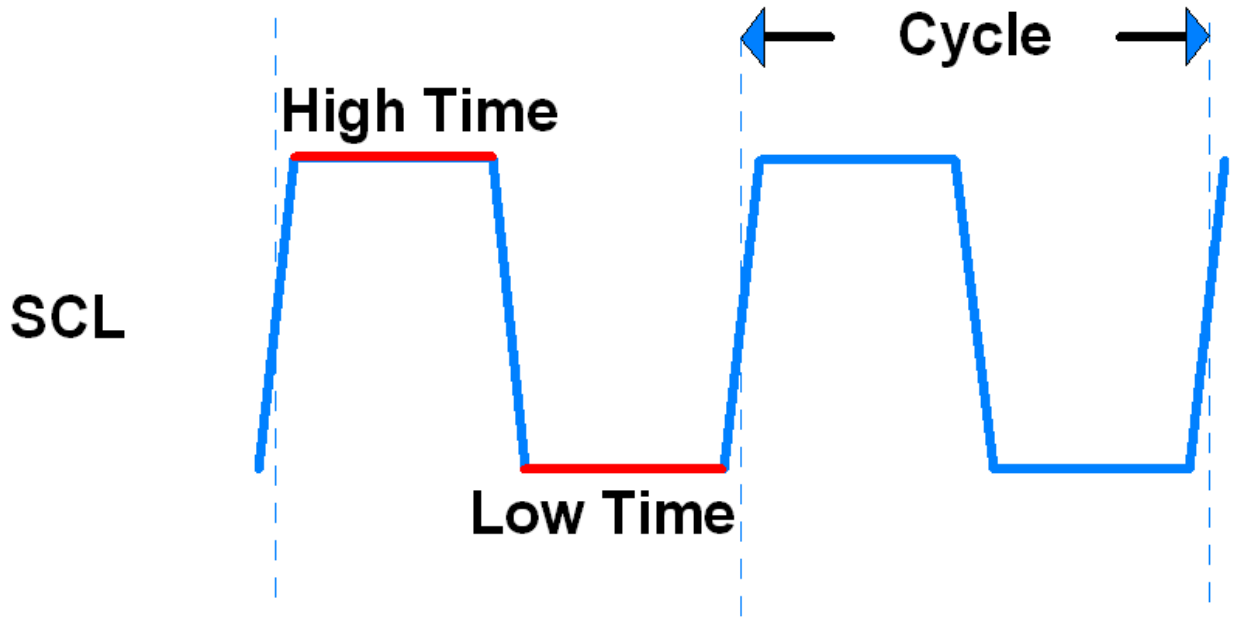
I2C0SCLL (I2C0 SCL Low Duty Cycle Register)
- This register contains the value for SCL Low time of the cycle.

6. I2C0SCLH (I2C0 SCL High Duty Cycle Register)

- It is a 16-bit register.

I2C0SCLH (I2C0 SCL High Duty Cycle Register)

- This register contains the value for SCL High time of the cycle.



The frequency and duty cycle of SCL is decided using I2C0SCLL and I2C0SCLH. I2C0SCLH contains the TON (High) time and I2C0SCLL contains the TOFF (Low) time.
The frequency is calculated as follows:

$$I2CBitFrequency = \frac{Fpclk}{I2C0SCLL + I2C0SCLH}$$

Example: Assume FPCLK = 30MHz, I2CBitFrequency = 300KHz, Duty Cycle = 50%.
As duty cycle is 50%, I2C0SCLL = I2C0SCLH.

Hence, $$300000 = \frac{30000000}{2xI2C0SCLH}$$

Hence, I2C0SCLL = I2C0SCLH = 50

7. I2C0ADR (I2C0 Slave Address Register)

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

I2C0ADR (I2C0 Slave Address Register)

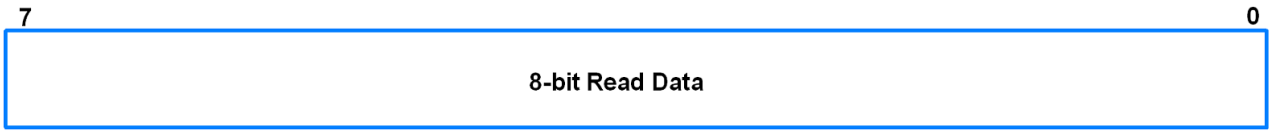
- It is only used when device is in Slave Mode.

- Bit 0 – GC (General Call)
- When this bit is set, the general call address (0x00) is recognized.
- Bit 7:1 – Address
- It contains the I2C device address for slave mode.

UART Registers-

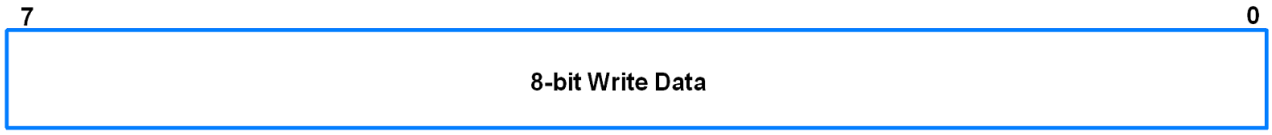
UART0 Registers

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)



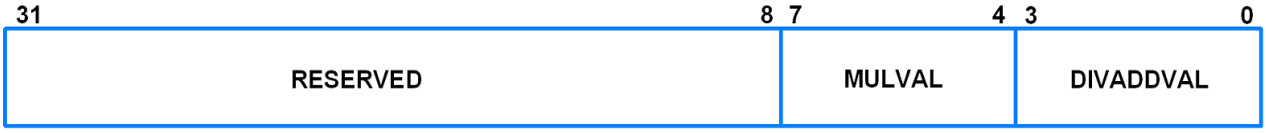
U0THR (UART0 Transmit Holding Register)

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.

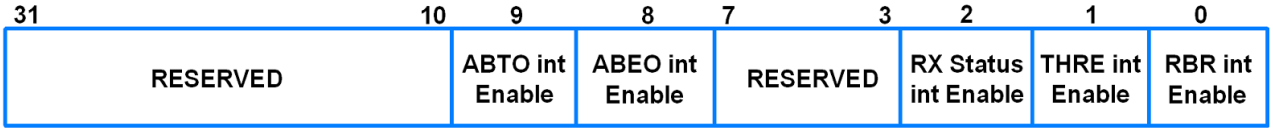


U0FDR (UART0 Fractional Divider Register)
   - If DIVADDVAL is 0, the fractional baudrate generator will not impact the UART0 baudrate.
   - Reset value of DIVADDVAL is 0.
   - MULVAL must be greater than or equal to 1 for UART0 to operate properly, regardless of whether the fractional baudrate generator 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.

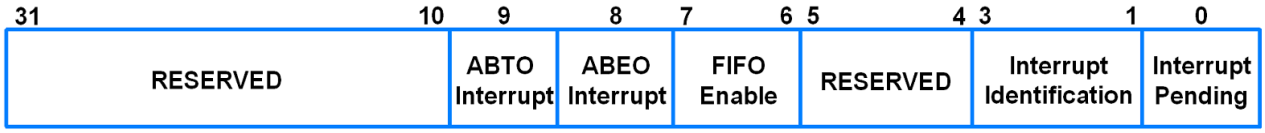
5. U0IER (UART0 Interrupt Enable Register)
   - It is a 32-bit read-write register.
   - It is used to enable UART0 interrupt sources.
   - DLAB should be zero (DLAB = 0).



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

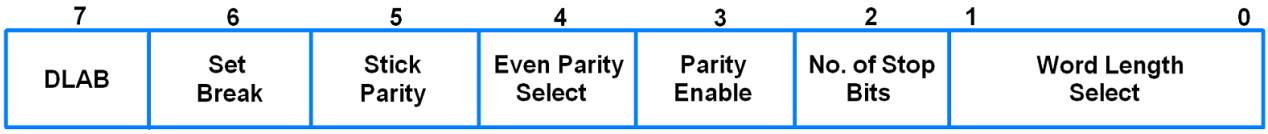
- 0 = Disable THRE interrupt
- 1 = Enable THRE interrupt
- Bit 2 - RX Line Interrupt Enable
- 0 = Disable UART0 RX line status interrupts
- 1 = EnableUART0 RX line status interrupts
- Bit 8 - ABEO Interrupt Enable
- 0 = Disable auto-baud time-out interrupt
- 1 = Enable auto-baud time-out interrupt
- 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 EnableThese 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.

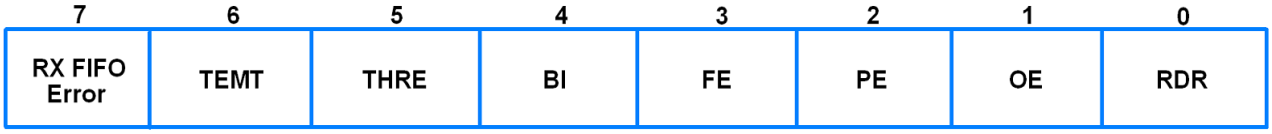


U0LCR (UART0 Line Control Register)

- 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.

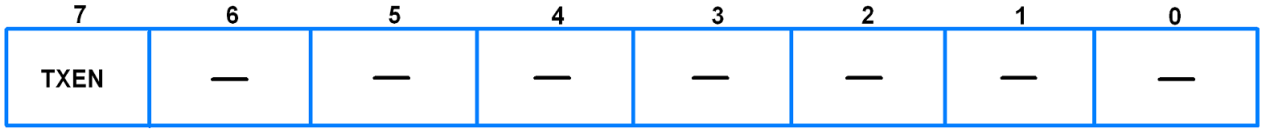


U0LSR (UART0 Line Status Register)

- 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.



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.

Program:-

```
/*
  MPU6050 interfacing with LPC2148(ARM7)
  http://www.electronicwings.com/arm7/mpu6050-gyroscope-accelerometer-temperature-interfacing-with-lpc2148
*/

#include <lpc214x.h>
#include <stdint.h>
#include <stdio.h>
#define MPU_WRITE_ADDR 0xD0
#define MPU_READ_ADDR 0xD1

char receive_string[100];

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 Cclk = 60MHz */
         }
}

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 */
}

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 */
}

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++;
	}
}

void I2C_INIT(void)
{
	PINSEL0 = PINSEL0 | 0x00000050; /* P0.2 and P0.3 as SCL0 and SDA0 */
	I2C0CONSET = 0x40; /* I2C Enable */
	I2C0SCLL = 0x32; /* I2C bit frequency 300 kHz with 50% duty cycle */
	I2C0SCLH = 0x32;
}

void I2C_START(void)
{
	I2C0CONSET = 0x20; /* STA = 1 */
	while ( (I2C0CONSET & 0x08) == 0 ); /* Wait till SI = 1 */
	I2C0CONCLR = 0x28; /* Clear STA and SI */
}

void I2C_WRITE( char data )
{
	I2C0DAT = data;
	I2C0CONSET = 0x40; /* I2C Enable */
	while( (I2C0CONSET & 0x08) == 0 ); /* Wait till SI = 1 */
	I2C0CONCLR = 0x08; /* Clear SI */
}

unsigned char I2C_READ( void )
{
	I2C0CONSET = 0x44; /* I2C Enable with Acknowledge */
	while( (I2C0CONSET & 0x08) == 0 ); /* Wait till SI = 1 */
	I2C0CONCLR = 0x0C; /* Clear SI and Acknowledge */
	return I2C0DAT;
}

unsigned char I2C_READ1( void )
{
	I2C0CONSET = 0x40; /* I2C Enable */
	while( (I2C0CONSET & 0x08) == 0 ); /* Wait till SI = 1 */
	I2C0CONCLR = 0x08; /* Clear SI  */
	return I2C0DAT;
}

void I2C_MULTIREAD( char* arr , int bytes )
```

```
{
	uint8_t i = 0;
	while( ( bytes - 1 ) != 0 )
	{
		I2C0CONSET = 0x44; /* I2C Enable with Acknowledge */
		while( (I2C0CONSET & 0x08) == 0 ); /* Wait till SI = 1 */
		I2C0CONCLR = 0x0C; /* Clear SI and Acknowledge */
		*( arr + i ) = I2C0DAT ;
		bytes--;
		i++;
	}
	I2C0CONSET = 0x40; /* I2C Enable */
	while( (I2C0CONSET & 0x08) == 0 ); /* Wait till SI = 1 */
	I2C0CONCLR = 0x08; /* Clear SI */
	*( arr + i ) = I2C0DAT ;
}

void I2C_STOP( void )
{
	I2C0CONSET = 0x50; /* STO = 1 */
}

void I2C_MPU_CUSTOM ( char reg, char regval )
{
	I2C_START();
	I2C_WRITE(MPU_WRITE_ADDR);
	I2C_WRITE(reg);
	I2C_WRITE(regval);
	I2C_STOP();
}


void MPU_INIT( void )
{
	delay_ms(200);

	I2C_MPU_CUSTOM(0x6B, 0x02); /* PWR_MGMT_1 PLL with Y-axis gyroscope
reference  */

	I2C_MPU_CUSTOM(0x19, 0x07); /* SMPLRT_DIV Reg Select, SMPLRT = 1K */

	I2C_MPU_CUSTOM(0x1A, 0x00); /* CONFIG Reg Select */

	I2C_MPU_CUSTOM(0x1B, 0x00); /* CONFIG Reg Select, Full scale range +/- 250
degree/C */

	I2C_MPU_CUSTOM(0x1C, 0x00); /* ACCEL Reg Select, Full scale range +/- 2g */
```

```
	I2C_MPU_CUSTOM(0x23, 0x00); /* FIFO disabled */

	I2C_MPU_CUSTOM(0x24, 0x00); /* I2C Master Control, I2C Freq 348KHz */

	I2C_MPU_CUSTOM(0x37, 0x00); /* Interrupt pin configuration */

	I2C_MPU_CUSTOM(0x38, 0x01); /* Interrupt Enable, Data Ready Enable */

	I2C_MPU_CUSTOM(0x67, 0x00); /* Master delay reg */

	I2C_MPU_CUSTOM(0x68, 0x00); /* Signal path reset */

	I2C_MPU_CUSTOM(0x6A, 0x00); /* User Control */

	I2C_MPU_CUSTOM(0x6C, 0x00); /* PWR_MGMT_2 */

	I2C_MPU_CUSTOM(0x74, 0x00); /* FIFO R/W */
}

int main(void)
{
	int16_t  GX,GY,GZ,AX,AY,AZ,T;
	double XA,XG,YA,YG,ZA,ZG,TF;
	char data[14];
	char result[100];
	UART0_init();
	I2C_INIT();
	MPU_INIT();
  while(1)
  {
			I2C_START();
			I2C_WRITE(MPU_WRITE_ADDR);
			I2C_WRITE(0x3B);
			I2C_START();
			I2C_WRITE(MPU_READ_ADDR);
			I2C_MULTIREAD(data, 14);
			I2C_STOP();
			AX = (((int16_t)(data[0]<<8)) | ((int16_t)data[1]));
			AY = (((int16_t)(data[2]<<8)) | ((int16_t)data[3]));
			AZ = (((int16_t)(data[4]<<8)) | ((int16_t)data[5]));
			XA = (double)AX/16384.0;
			YA = (double)AY/16384.0;
			ZA = (double)AZ/16384.0;
			T = (((int16_t)(data[6]<<8)) | ((int16_t)data[7]));
			TF = ( ((double)T/340.00) + 36.53 ) ;
			GX = (((int16_t)(data[8]<<8)) | ((int16_t)data[9]));
			GY = (((int16_t)(data[10]<<8)) | ((int16_t)data[11]));
			GZ = (((int16_t)(data[12]<<8)) | ((int16_t)data[13]));
```

```
                        XG = (double)GX/131.0;
                        YG = (double)GY/131.0;
                        ZG = (double)GZ/131.0;
                        sprintf(result,"Ax=%lfg Ay=%lfg Az=%lfg T=%lf%cC Gx=%lf%c/s
Gy=%lf%c/s Gz=%lf%c/s",XA,YA,ZA,TF,0xB0,XG,0xB0,YG,0xB0,ZG,0xB0);
                        UART0_SendString(result);
                        UART0_SendString("\r\n");
                        delay_ms(1000);
        }
}
```