Inter-Processor Communication

Sashank Malladi, 010466651
Computer Engineering Department, College of Engineering
San Jose State University, San Jose, CA 94303
Email: sashank.malladi@sjsu.edu

Abstract

Inter-processor communication is transfer of data from one processor to other processor. The purpose of this lab is to establish communication between two microprocessor units using interrupt based GPIO technique. The system under consideration is a module with two microcontrollers connected through a GPIO ports. The objective of this lab is achieved by successfully establishing communication between two LPC 1769 microcontrollers.

Keywords: LPCXpresso 1769, GPIO

1. Introduction

This lab is focused on establishing communication between two LPC 1769 modules. This CPU module uses LPC1769 CPU, based on the ARM cortex M0 core that is provided with a user-friendly IDE called LPCXpresso. The main objective of this lab experiment is to understand GPIO features and interrupt concepts then transfer data between two LPC modules. It is essential to have prior programming knowledge of C/C++ languages to program the microcontroller circuit.

2. Methodology

This section includes the objectives, technical details, challenges, results and design of the project explained with block diagrams, flowcharts and layouts.

2.1 Objectives and Technical Challenges

The objectives of the lab includes:

1. Interface the LPC1769 with other LPC1769 CPU module using GPIO ports.
2. To build a power unit for the CPU module using voltage regulator.
3. Develop a program that sends the data from one LPC microcontroller to the other microcontroller connected through GPIO ports.

There were certain technical challenges faced while performing the experiment:

1. Connecting the berg strips to the LPC board.
2. Mounting and soldering the LPC micro controller onto the breakout board.
3. Connection between two LPC 1769 micro controllers.

When the berg strips were connected to the CPU module, due to the small width of the pins, it was not connected to the CPU module, so the pins were soldered.

2.2 Problem Formulation and Design

This section will provide the detailed design. It includes the block diagram and also the schematics and pin connection between the components used for this lab assignment. The hardware used for this lab is connected to the wire wrapping board using soldering technique.

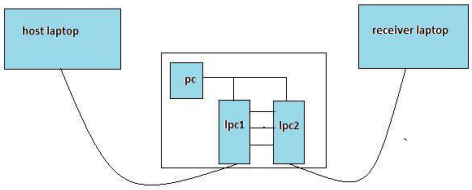

Figure 1. Wire wrap board layout

The hardware design includes two LPCXpresso 1769 CPU modules. The critical part of this lab is establishing communication between the two microcontrollers.

2.2.1 LPC 1769 Overview

The LPC1769 is an ARM Cortex-M3 based microcontrollers for embedded applications featuring a high level of integration and low power consumption. It offers system enhancements such as enhanced debug features and a higher level of support block integration.

The LPC1769 operates at CPU frequencies of up to 120 MHz. The ARM Cortex-M3 CPU incorporates a 3-stage pipeline and uses a Harvard architecture with separate local instruction and data buses as well as a third bus for peripherals.

Detailed technical statistics are given in following table

CPU	Core	32 bit ARM Cortex M3
	Speed	Up to 120 MHz
	Power	120 – 140 mA
MEMORY	RAM	64 kB SRAM
	ROM	512 kB on chip flash

Table 1. Technical statistics of LPC 1769 microcontroller unit.

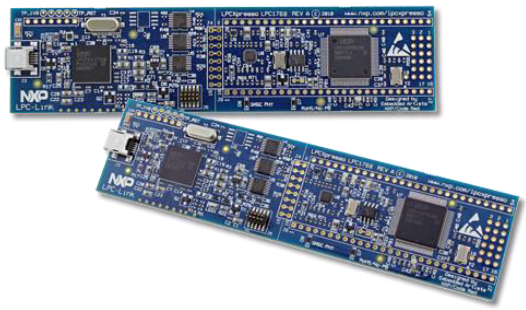

Figure 2. LPC 1769 unit

[1] LPC 1769 microcontroller unit, https://www.embeddedartists.com/sites/default/files/image/product/xpr_lpc176x_banner.png

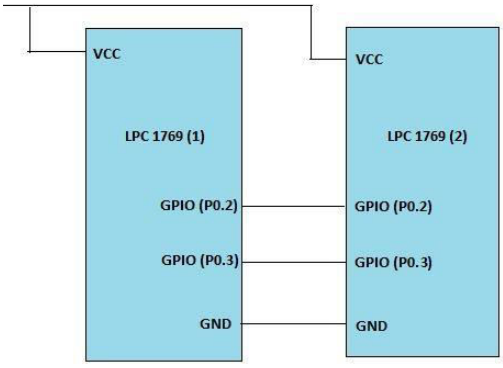

Figure 3. LPC 1769 interface with other LPC 1769 through GPIO

In our implementation, GPIO pins of port number 0 are used to connect two LPC CPU modules. The following table shows the required connection between the two CPU modules:

LPCXpresso 1769 CPU-1	**LPCXpresso 1769 CPU-2**
GPIO (P0.2)	GPIO (P0.2)
GPIO (P0.3)	GPIO (P0.3)
GND (J6-1)	GND (J6-1)

Table 2. Pin connection of interface between two LPC microcontrollers

The software design was implemented using graphics library provided by Adafruit on the LPCXpresso v6.1.0_164 IDE.
In this project, data is transferred from host CPU module to receiver module and then transferred data is shown on the console.

2.2.2 GPIO Overview

GPIO is what lets your microcontroller be something more than a weak auxiliary processor. With it you can interact with the environment, connecting up other devices and turning your microcontroller into something useful

GPIO has two fundamental operating modes, input and output. Input lets you read the voltage on a pin, to see whether it's held low (0v) or high (3v) and deal with that information programmatically. Output lets you set the voltage on a pin, again either high or low. Every pin on the LPC can be used as a GPIO pin, and can be independently set to act as an input or output.

For this lab we have considered GPIO pins 2 and 3 under port 0.

FIODIR special purpose register is used to set the direction of the GPIO port. Setting it 1 makes the port as output and 0 makes the port input.

FIOSET and FIOCLR are the two fast output control registers. Writing a 1 to a bit in FIOSET will enable the corresponding pin, and writing to FIOCLR will disable the pin

If the FIODIR is 0 for a particular GPIO port, the current state of that port can be read from FIOPIN pin.

Interrupts is one concept which provide as a way to get input from the GPIO pins. With the correct settings the processor can wait for an event in the background while letting your code run in the meantime. When an event occurs, the processor will stop your currently running code, run code to respond to the event, and restart the execution of your main program. This mode of communication is known as Interrupt Driven I/O.

We have utilized Interrupt Driven I/O, mentioned above for implementing this lab.

3. Implementation

The implementation consist two LPC 1769 modules one acting as a host and other as a receiver.

The data is transferred through the GPIO ports using concept of Interrupt driven I/O.

The following pictures represent implementation of the lab.

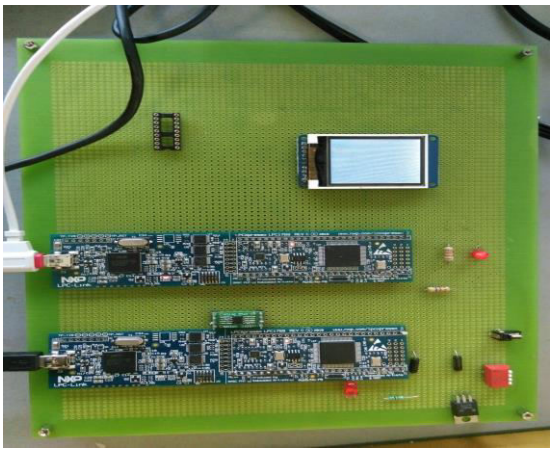

Figure 4. Embedded board design

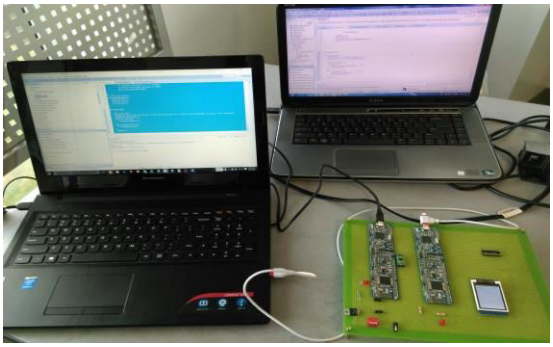

Figure 5. Project Overview

3.1 Algorithm

The following steps describe the algorithm for implementation

1. Set two GPIO pins in output direction at the transmitter module. Same set of two GPIO pins must be set as input at the receiver module.
2. One of the two GPIO ports connected is considered for transferring data and other is for triggering interrupts.
3. Convert data to be transferred into binary bits.
4. Set or clear the GPIO, which is used for transferring data based on the generated binary bits.
5. Trigger the interrupt to notify the receiver to read the data being transmitted.

3.2 Flow Chart

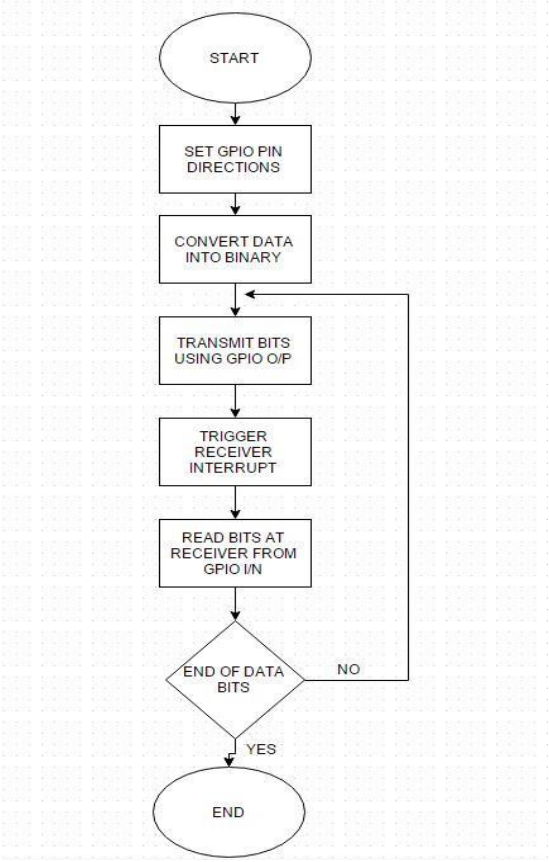

Figure 6. Implementation flowchart

3.3 Pseudo Code

3.3.1 Pseudo code for Transmitting bits

```
void transmitter(){
        //Set pin 0.2 as output
        GPIOinitOut(0,2);
        //Set pin 0.3 as output
        GPIOinitOut(0,3);
        int bits[]={0,1,1,1};
        int
sizeOfArray=sizeof(bits)/sizeof(int);
        int i=0;
          while(i<sizeOfArray)
          {
                clearGPIO(0,3);
```

```
                printf("%d", bits[i]);
                if (bits[i] == 1){
                //Activate pin 0.2 to transmit 1
                        setGPIO(0,2);
                        }
                        else
                        {
                //deactivate pin 0.2 to transmit 1
                        clearGPIO(0,2);
                        }
                setGPIO(0,3);
                int delay=0;

        for(delay=0;delay<1000000;delay++){}
                        i++;
                }

        }
```

3.3.2 Pseudo code for receiving bits

Setting up the interrupt

```
void receiver(uint8_t portNum, uint32_t pinNum)
{
        if (portNum == 0)
        {
        // Enable Rising Edge Interrupt on P0[3]
        LPC_GPIOINT->IO0IntEnR |= (1 << pinNum);
        // Enable Rising Edge Interrupt on P0[3]
        NVIC_EnableIRQ(EINT3_IRQn);
                }
}
```

Interrupt handler

```
void EINT3_IRQHandler() {
        if(firstTime>0){
        int PinState;
        if((LPC_GPIOINT->IO0IntStatR >> 3) & 1){

//Read the pin state if triggered from transmitter
PinState = (LPC_GPIO0->FIOPIN >> 2) & 1;
// clear interrupt after reading the input so that it can be triggered later
LPC_GPIOINT->IO0IntClr |= (1 << 3);
}
```

4. Testing and Verification

This section details the testing and verification procedures used.

4.1 Testing Procedure

Following are the steps to follow for debugging purposes in case of errors.

1. The connections are checked for any shorting or loose connections.
2. The wire wrapping and the soldering techniques are checked. The board is connected to the IDE and the program is built and debugged.
3. The VCC and Ground connections are mainly verified thoroughly.
4. The schematics are checked.
5. The code is checked.

4.2 Verification Procedure

If the steps given in the testing procedure are followed properly, the screenshot given below would be the correct output of the experiment:

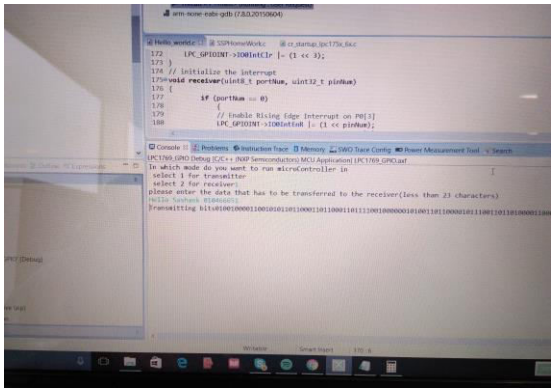

Figure 7. Program output at transmitter console

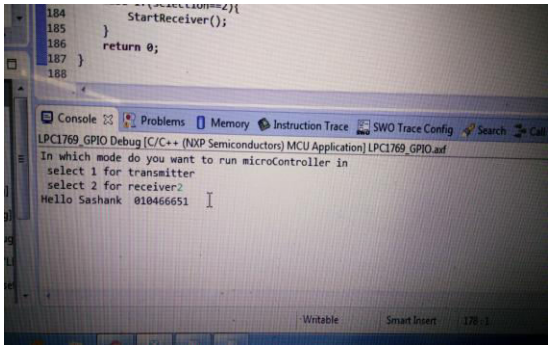

Figure 8. Program output at receiver console.

5. Conclusion

The objective of this lab was to interface two LPC1769 modules to demonstrate communication between two microprocessors using GPIO. The data is transferred between

transmitter and receiver successfully and displayed on console.

6. Acknowledgement

I would like to thank Dr Harry Li for instructing the various steps in this project in detail. Also, I would like to thank my classmates for the meaningful discussions about the project.

7. References

[1] H.Li, Guidelines for CMPE240 project and report, Computer Engineering Department, San Jose State University, San Jose 95112.
[2] H.Li, CMPE240 Lecture Notes, Computer Engineering Department, San Jose State University, San Jose 95112.
[3] NXP LPCXpresso1769 Discussion forums at www.lpcware.com/forum.
[4] LPC1769 User manual.
[5] http://www.cs.umd.edu/~rohit/ESDbook.pdf

8. Appendix

Appendix A: Bill of Materials

Number	Description
1	Wire wrapping board
2	2 in quantity of LPCXpresso 1769 CPU Module
3	Jumper cables x 6
4	Burg strips
5	Solder iron

Table 3: Bill of materials

Appendix B. Source Code

```
#ifdef __USE_CMSIS
#include "LPC17xx.h"
#endif

#include <cr_section_macros.h>
#include <stdio.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

// grouping the 8 bits  for displaying on console
int receivingBits[8];
// counter for grouping the bits
int static counter=0;
// to check if the received is first bit
int firstBit=0;

//Initialize the port and pin as outputs.
void GPIOinitOut(uint8_t portNum, uint32_t
pinNum)
{
	if (portNum == 0)
	{
		LPC_GPIO0->FIODIR |= (1 <<
pinNum);
	}
	else if (portNum == 1)
	{
		LPC_GPIO1->FIODIR |= (1 <<
pinNum);
	}
	else if (portNum == 2)
	{
		LPC_GPIO2->FIODIR |= (1 <<
pinNum);
	}
	else
	{
		puts("Not a valid port!\n");
	}
}

//Initialize the port and pin as outputs.
void GPIOinitIn(uint8_t portNum, uint32_t
pinNum)
{
	if (portNum == 0)
	{
		LPC_GPIO0->FIODIR &= ~ (1
<< pinNum);
	}
	else if (portNum == 1)
	{
		LPC_GPIO1->FIODIR &= ~ (1
<< pinNum);
	}
	else if (portNum == 2)
	{
		LPC_GPIO2->FIODIR &= ~ (1
<< pinNum);
	}
	else
	{
		puts("Not a valid port!\n");
	}
}


// activate the pins
void setGPIO(uint8_t portNum, uint32_t
pinNum)
{
	if (portNum == 0)
	{
```

```
                LPC_GPIO0->FIOSET = (1 << pinNum);
        }
        //Can be used to set pins on other ports for future
modification
        else
        {
                puts("Only port 0 is used, try again!\n");
        }
}

//Deactivate the pin
void clearGPIO(uint8_t portNum, uint32_t pinNum)
{
        if (portNum == 0)
        {
                LPC_GPIO0->FIOCLR = (1 << pinNum);
        }
        //Can be used to clear pins on other ports for future
modification
        else
        {
                puts("Only port 0 is used, try again!\n");
        }
}


// start transmitting the bit after converting into binary
void trasmitData(char data[23]){
        //Converting the data in to binary
        int bits[184]={};
                int bitCount=0;
                int i;
                 for (i=0;i<strlen(data);i++){
                        int absNum=abs(data[i]);
                        int binary[8]={};
                        int itr;
                        for (itr=8;itr>0;itr--){
                                binary[itr-1]=(absNum%2);
                                absNum=absNum/2;
                        }
                        for (itr=0;itr<8;itr++){
                                 bits[bitCount]=binary[itr];
                                 bitCount++;
                        }
                       }
                 // transmitting the data on GPIO
                        int
sizeOfArray=sizeof(bits)/sizeof(int);
                        i=0;
                        printf("Transmitting bits");
                  while(i<sizeOfArray)
                  {
                        clearGPIO(0,3);
                        printf("%d", bits[i]);
                        if (bits[i] == 1){
                                        //Activate pin 0.2
                setGPIO(0,2);
                                        }
                                        else
                                        {       //De
activate pin

                clearGPIO(0,2);
                                        }
                                setGPIO(0,3);
                                int delay=0;

                for(delay=0;delay<2000000;delay++){}
                                        i++;
                        }
}
// point of start for the transmitter
void StartTransmitter(){
        //Set pin 0.2 as output
        GPIOinitOut(0,2);
        //Set pin 0.3 as output
        GPIOinitOut(0,3);
        // maintianing data in the array
        char data[23];
        printf("please enter the data that has to
be transferred to the receiver(less than 23
characters)\n");
        scanf(" %[^\n]s",data);
        trasmitData(data);
}
// interrupt handler
void EINT3_IRQHandler() {
        if(firstBit>0){
        int PinState;
        if((LPC_GPIOINT->IO0IntStatR >> 3) &
1){
                PinState = (LPC_GPIO0-
>FIOPIN >> 2) & 1;

                if(counter>6){

        receivingBits[counter]=PinState;
                        int itr=0;
                        int mulFactor=7;
                        int asval=0;
                        for(itr=0;itr<8;itr++){
                                //printf("%d
\n",receivingBits[itr]);

        asval=asval+receivingBits[itr]*pow(2,mu
lFactor);
                        mulFactor--;
                        }
                        printf("%c",asval);
                        counter=0;
                }
```

```
                else{

                        receivingBits[counter]=PinState;
                        counter++;
                }

                }
        }
        firstBit++;
        LPC_GPIOINT->IO0IntClr |= (1 << 3);
}
// initialize the interrupt
void receiver(uint8_t portNum, uint32_t pinNum)
{
                if (portNum == 0)
                        {
                        // Enable Rising Edge Interrupt on
P0[3]
                        LPC_GPIOINT->IO0IntEnR |= (1
<< pinNum);
                        // Enable Rising Edge Interrupt on
P0[3]
                        NVIC_EnableIRQ(EINT3_IRQn);
                        }
}
// start the receiver
void StartReceiver(){
        GPIOinitIn(0,2);
        GPIOinitIn(0,3);
        receiver(0,3);
}

int main(void)
{
        int selection;
        printf("In which mode do you want to run
microController in \n select 1 for transmitter \n select 2 for
receiver");
        scanf("%d",&selection);
        if(selection==1){
                StartTransmitter();
        }
        else if(selection==2){
                StartReceiver();
        }
   return 0;
}
```