AT45 Flash Data Logger

Nadim Sarras
Computer Engineering Department, College of Engineering
San Jose State University, San Jose, CA 94303
E-mail: nadim.sarras@gmail.com

Abstract

This project focuses on utilizing the AT45 abilities of storing data. Combined with the serial connections of the LPC 1769 board a flash data logger was created. The console was used to write the data to a buffer, then into the AT45. The prototype board was then turned off and turned back on. The prototype board should then be able to read back from the flash to one of the buffers to the LPC1769, allowing the data to be displayed on the console.

1. Introduction

The experiment focused on the utilization of the AT45 flash memory module to create a Data Logger that would retain information, even while the power is turned off. The AT45 module was used in conjunction with the LPC1769 CPU board. The AT45 uses SPI protocol methods with regards to MOSI and MISO, to transfer data back and forth. [1] The functionality of the AT45 pins also needed to be connected to the corresponding pins on the LPC 1769 CPU. The pin functionality of both the AT45 and the LPC 1769 CPU are shown in their corresponding data sheets. [2] (AT45 Data Sheet [3] (LPC 1769 Data sheet)

2. Methodology

In order to achieve the goal of this project and to achieve full functionality, a great amount of planning occurred before any construction occurred. This section will describe the accumulation of materials, technical challenges, and design of the project.

2.1. Objectives and Technical Challenges

The objective of the lab consisted of writing the following message, "SJSU CMPE127 Nadim Sarras 008382608" to a buffer. The Buffer then writes the flash. The entire prototype configuration is then turned off and turned back on. Without writing to the buffer, buffer two reads the data back from the flash and into the LPC1769 CPU board. The received message is then displayed to the output of the console using the printf function. The main technical challenges faced during this experiment involved mounting the AT45 Flash Memory module on the prototype board. This was not accounted for and delayed the start of constructing the hardware.

2.2. Problem Formulation and Design

The following sections will provide the design and implementation methods for this lab. The Hardware Design section will include system block diagrams, schematics, and a bill of material. The schematics will represent the design created on the prototype board while the block diagrams will visualize the functional relationships between the different modules. The Software Design incorporates flow charts, algorithms, and pseudo code. The flow charts visualize the software process of the program. Algorithms and pseudo code act as a description of what the source code is implementing.

3. Implementation

This section will go over the steps taken as well as the thought process behind the hardware and software design of the project.

3.1. Hardware Design

The hardware design of building off of the prototype board from the last lab. The AT45module is added to the prototype board to fulfill the task of creating a data logger. Figure 1 displays the functional interactions between the LPC 1769 CPU board and the AT45 flash memory module. MOSI (master out slave in) drives the input of the AT45. While MISO (master in slave out) drives the input of the CPU from the AT45. The serial clock was used to synchronize the data transfer. SSEL 1 served as the chip select where the CPU would select the AT45 module based on the low value coming from the CPU. Figure 2 displays the power schematics to power the board, an alternative method from using the USB. While Figure 3 are the actual schematics constructed in this lab.

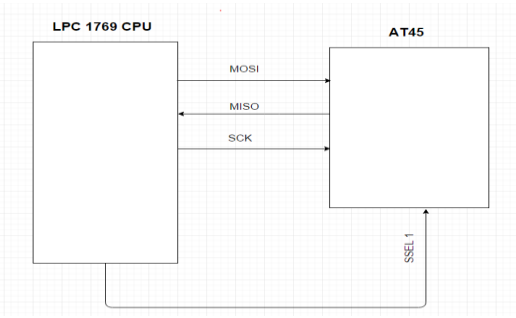

Figure 1. AT45 – CPU Functional connections

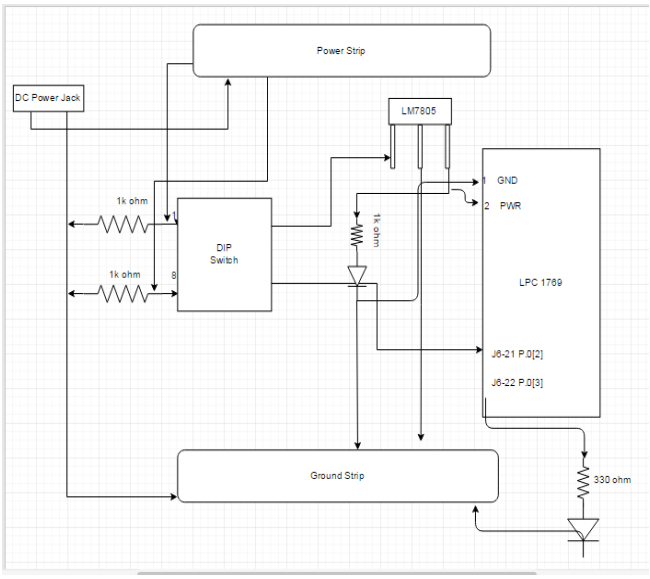

Figure 2. Power Circuit Schematic (Used to power board)

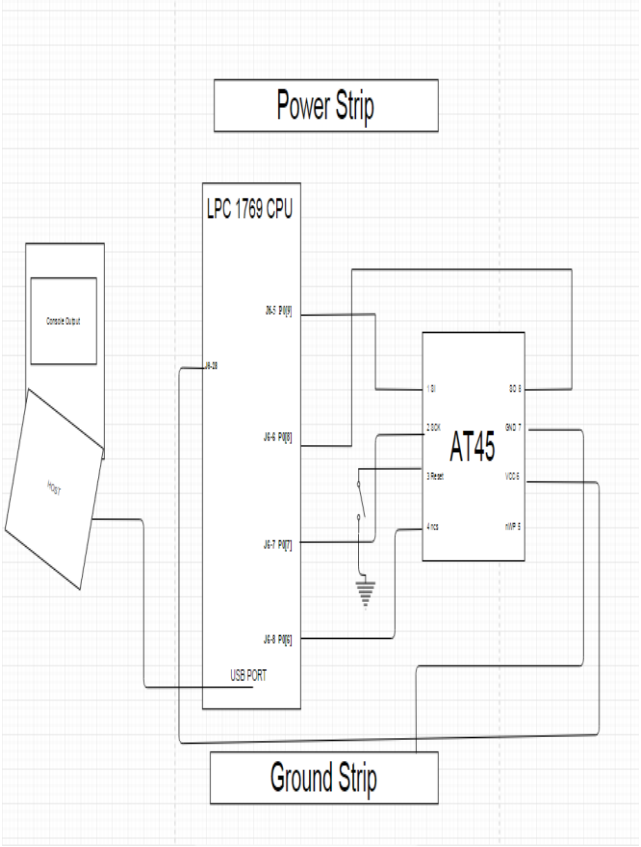

Figure 3. AT45 connection schematic to CPU

Table 1. Pin Usage

Functional Name	Pin Number	Purpose	CPU Model LPC 1769 Port Num / Pin
GND	J6-1	Ground	GNDX
VIN	J6-2	Power Supply	EXT_POWX
MOSI	J6-5	Send Data	Port 0/ Pin 9
SCK	J6-7	Serial Clock	Port 0/Pin 7
MISO	J6-6	Receive Data	Port 0/ Pin 8
SSEL1	J6-8	Chip Select	Port 0/Pin 6
VOUT	J6-28	Output Power	VIO_3V3X

Table 2. Bill of Materials

Part	Function
LPC1769CD	CPU Module
AT45 Flash Memory Module	Data Logger/ Retain Data
Power Supply, 9v/2A	Power Circuit
USB Cable	Loading Program/ Debugging

3.2. Software Design

The bulk of the software was given, however minor adjustments needed to be made to read and write the correct output message. Figure 4 portrays the flowchart used to design the software. When the program starts, depending on the user intention it will decide to write to buffer 1 or to begin reading from buffer 2. This input is decided if the data requested is already loaded into the flash module. If it is not the program will continue to write the desired message to buffer 1. The message is then read from the buffer and printed on the console for verification. The program then continues to have the buffer write the message to the flash memory. Once the board is reastarted or this step is complete, the program will then write from the flash and onto buffer 2. The data on buffer 2 will then be read and displayed to the console. If the output on the console is the same as the output from when the message was written, then the Data Logger is successful and reading and writing information.

The only additional algorithm required the addition of SSP1exchangeByte to completely print the desired message. In addition a for loop was created to the size of the message in order to allow all characters of the messages to be printed onto the console. The pseudocode below figure 5 explains how the program interacts with the CPU, buffers, and the flash memory module.

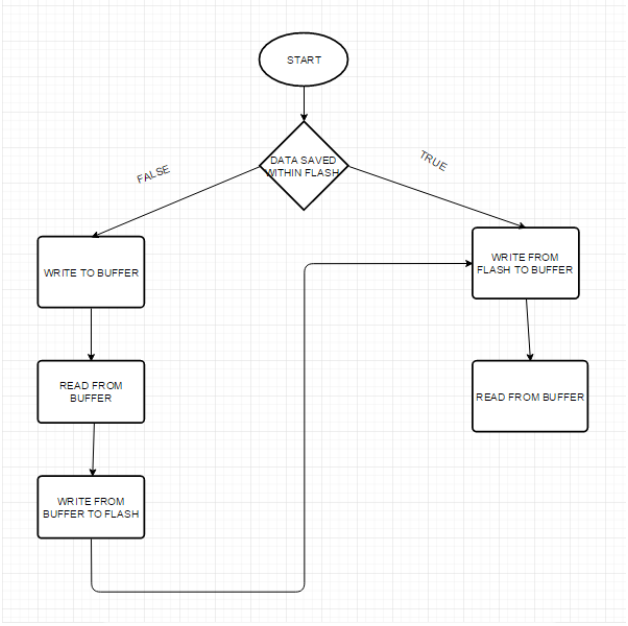

Figure 4. Software Buffer Read/Write Flow Chart

```
SSP1exchangeByte('8');
SSP1exchangeByte('3');
SSP1exchangeByte('8');
SSP1exchangeByte('2');
SSP1exchangeByte('6');
SSP1exchangeByte('0');
SSP1exchangeByte('8');

LPC_GPIO2->FIOSET = (1<<6); // port 2 pin 6
for ( i = 0; i < 10000; i++ );
printf("\n**********************************

initSSP1();
printf("\nReading from Buffer 1\n");
LPC_GPIO0->FIOCLR = (1<<6);   // port 0 pin 6
//************************Read from buff 1
//Opcode for Read buff 1
SSP1exchangeByte(0xd4);
//Send 3 Bytes = 24 bits= 16 don't care bits +
SSP1exchangeByte(0x00);
SSP1exchangeByte(0x00);
SSP1exchangeByte(0x00);
SSP1exchangeByte(0x00); //trial for buff1
//Send 1 dummy byte
int count = 0;
for(count=0; count<35;count++){
printf("\n %i\t %c",i,SSP1exchangeByte(0x00));
}
printf("\n 2\t %c",SSP1exchangeByte(0x00));
printf("\n 3\t %c",SSP1exchangeByte(0x00));
printf("\n 4\t %c",SSP1exchangeByte(0x00));
```

Figure 5. Algorithm

Psuedocode

if(nothing stored in flash){
write message to buffer;
buffer write to flash
}
else{
do nothing
}

for(i=0;i<size_message;i++){
write from flash memory to buffer;
read from buffer;
printf(character_message);
}

4. Testing and Verification

The testing procedure involved the following steps;

1. Make Sure code is debugged and loaded into board
2. All connections are tight and there are no loose wires
3. Run the debugger and load the program in the board
4. Program will write to buffer and then to flash
5. Console will output what is being written
6. Turn off Board and all modules attached on the prototype board
7. Reload the program into the board excluding the write section (only reading)
8. If initial program stored successfully into flash then program should write from flash to buffer and read from the buffer
9. The read message will display on the console

```
Device ID

 0       ff
 1       1f
 2       28
 3       0
**********

Writing to Buffer 1

**********

Reading from Buffer 1

 10000   S
 10000   J
 10000   S
 10000   U
 10000
 10000   C
 10000   M
 10000   P
 10000   E
 10000   1
 10000   2
 10000   7
 10000
 10000   N
 10000   A
 10000   D
 10000   I
 10000   M
```

Figure 6. Writing to buffer and reading output before shutting off board

```
Reading from Buffer 2

 10000   S
 10000   J
 10000   S
 10000   U
 10000
 10000   C
 10000   M
 10000   P
 10000   E
 10000   1
 10000   2
 10000   7
 10000
 10000   N
 10000   A
 10000   D
 10000   I
 10000   M
 10000
 10000   S
 10000   A
 10000   R
 10000   R
 10000   A
 10000   S
 10000
 10000   0
 10000   0
 10000   8
```

Figure 7. Writing to buffer and reading output before shutting off board

```
Device ID

 0       ff
 1       1f
 2       28
 3       0
**********

**********

Writing from Flash Memory to Buffer 2

**********
```

Figure 8. Board reloaded without write program. Flash wrting to buffer. Console read from buffer

```
Device ID

 0       ff
 1       1f
 2       28
 3       0
**********

**********

Writing from Flash Memory to Buffer 2

**********

Reading from Buffer 2


 10000   S
 10000   J
 10000   S
 10000   U
 10000
 10000   C
 10000   M
 10000   P
 10000   E
```

Figure 9. Board reloaded without write program. Flash wrting to buffer. Console read from buffer

5. Conclusion

In conclusion, the lab was successful and the console outputs were correct as shown in figures 6-9. The Data logger was able to read and write from buffers. The AT45 module successfully retained the desired message after the board had been shut off, and was able to write

that data back to a buffer where it was eventually read and printed onto the console display. This lab was useful in understanding the functions of MISO and MOSI, and their importance in being able to send data back and forth between the CPU, buffer, and the flash memory module.

6. Acknowledgement

An acknowledgement of thanks is given to the professor and teacher assistants who had cleared up questions about the implementation of the lab.

7. References

[1] H. Li, "Lab 2 Flash Memory Interface/ AT45DB081 FLASH.pdf", *Lecture Notes of CMPE 146/242*, Computer Engineering Department, College of Engineering, San Jose State University, March 6, 2006, pp. 1.

[2] H. Li, "Lab 2 Flash Memory Interface/ Reference Program Flash Pseudo Code 2009-10-5.pdf", *Lecture Notes of CMPE 146/242*, Computer Engineering Department, College of Engineering, San Jose State University, March 6, 2006, pp. 1.

[3] H. Li, "LPCXpressoLPC1769revB", *CPU Datasheets of CMPE 127*, Computer Engineering Department, College of Engineering, San Jose State University, March 1, 2016, pp. 7.

8. Appendix

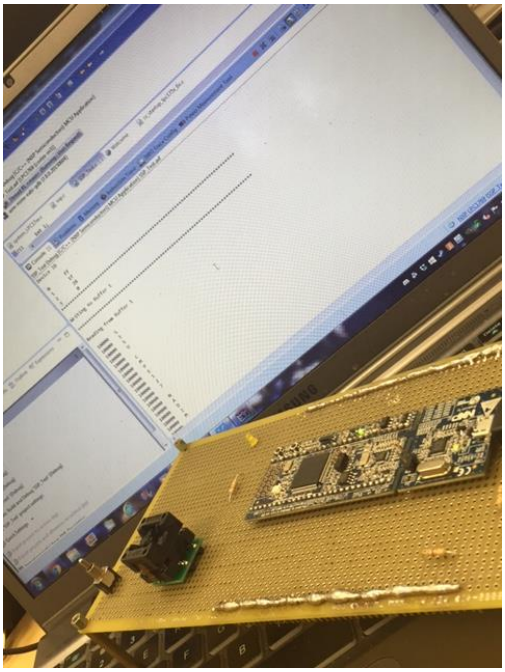

Figure 10. Writing to Flash Memory With Console Output

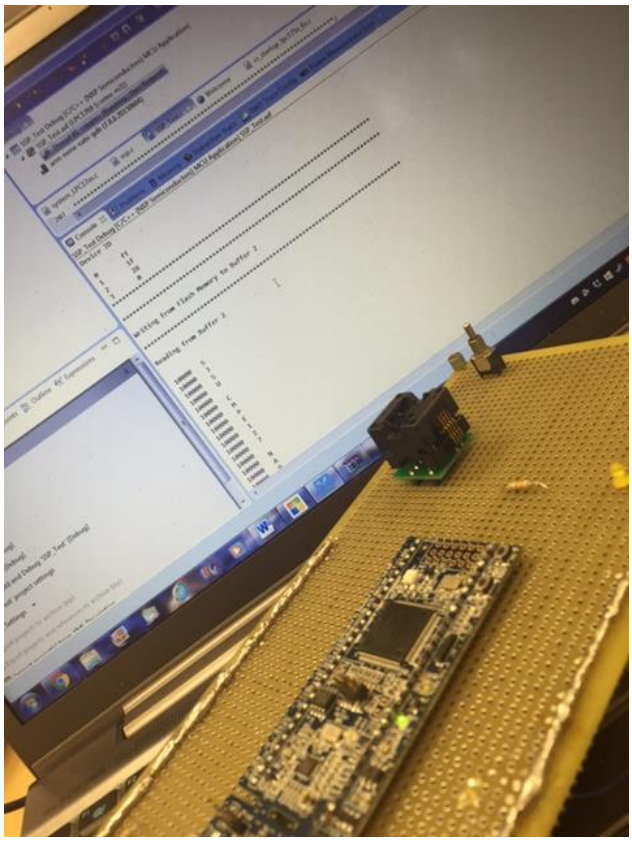

Figure 11. Reading Data back from Flash Memory Module with console output

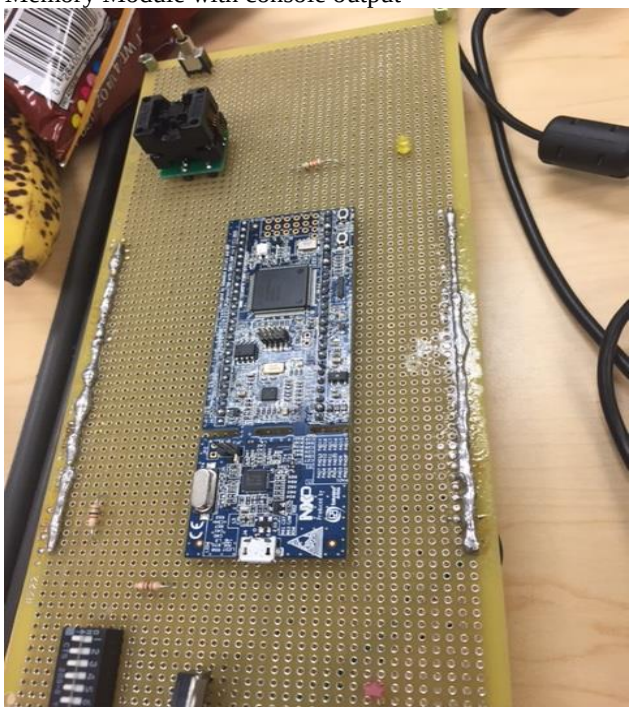

Figure 12. Layout of the prototype board with all modules.

Source Code

```
#define SPI_DELAY 1000
int main (void)
{
	int i;
	initSSP1();
	//enable_timer(0);
	for ( i = 0; i < 10000; i++ );
	printf("Device ID\n");
	LPC_GPIO0->FIOCLR = (1<<6);	//
0.6 port 0 pin 6
	printf("\n 0\t
%x",SSP1exchangeByte(0x9f));
	printf("\n 1\t
%x",SSP1exchangeByte(0x9f));
	printf("\n 2\t
%x",SSP1exchangeByte(0x00));
	printf("\n 3\t
%x",SSP1exchangeByte(0x00));
	LPC_GPIO2->FIOSET = (1<<6);
	for ( i = 0; i < 10000; i++ );
	printf("\n************************
****************************************
***********\n");

	//************************Write
to buffer1
	//Opcode for write to buff1
	initSSP1();
	printf("\nWriting to Buffer 1\n");
	LPC_GPIO0->FIOCLR = (1<<6);
// port 0 pin 6
	SSP1exchangeByte(0x84); //Send 3
Bytes = 24 bits= 16 don't care bits + 8
(1st Byte of buffer) bits
	SSP1exchangeByte(0x00);
	SSP1exchangeByte(0x00);
	SSP1exchangeByte(0x00);
	//SSP1exchangeByte('0');
	SSP1exchangeByte('S');
	SSP1exchangeByte('J');
	SSP1exchangeByte('S');
	SSP1exchangeByte('U');
	SSP1exchangeByte(' ');
	SSP1exchangeByte('C');
	SSP1exchangeByte('M');
	SSP1exchangeByte('P');
	SSP1exchangeByte('E');
	SSP1exchangeByte('1');
	SSP1exchangeByte('2');
	SSP1exchangeByte('7');
	SSP1exchangeByte(' ');
	SSP1exchangeByte('N');
	SSP1exchangeByte('A');
	SSP1exchangeByte('D');
	SSP1exchangeByte('I');
	SSP1exchangeByte('M');
	SSP1exchangeByte(' ');
	SSP1exchangeByte('S');
	SSP1exchangeByte('A');
	SSP1exchangeByte('R');
	SSP1exchangeByte('R');
	SSP1exchangeByte('A');
	SSP1exchangeByte('S');
	SSP1exchangeByte(' ');
	SSP1exchangeByte('0');
	SSP1exchangeByte('0');
	SSP1exchangeByte('8');
	SSP1exchangeByte('3');
	SSP1exchangeByte('8');
	SSP1exchangeByte('2');
	SSP1exchangeByte('6');
	SSP1exchangeByte('0');
	SSP1exchangeByte('8');

	LPC_GPIO2->FIOSET = (1<<6); // port
2 pin 6
	for ( i = 0; i < 10000; i++ );
	printf("\n************************
****************************************
***********\n");

	initSSP1();
	printf("\nReading from Buffer
1\n");
	LPC_GPIO0->FIOCLR = (1<<6);	//
port 0 pin 6
	//************************Read
from buff 1
	//Opcode for Read buff 1
	SSP1exchangeByte(0xd4);
	//Send 3 Bytes = 24 bits= 16 don't
care bits + 8 buffer addr bits ????
	SSP1exchangeByte(0x00);
	SSP1exchangeByte(0x00);
	SSP1exchangeByte(0x00);
	SSP1exchangeByte(0x00);	//trial
for buff1
	//Send 1 dummy byte
	int count = 0;
	for(count=0; count<35;count++){
	printf("\n %i\t
%c",i,SSP1exchangeByte(0x00));
	}
//	printf("\n 2\t
%c",SSP1exchangeByte(0x00));
```

```
//     printf("\n 3\t
%c",SSP1exchangeByte(0x00));
//     printf("\n 4\t
%c",SSP1exchangeByte(0x00));

       printf("\n************************
****************************************
***********\n");
       LPC_GPIO2->FIOSET = (1<<6);
       for ( i = 0; i < 10000; i++ );

       initSSP1();
       /*********************************
**/
       printf("\nWriting from Buffer 1 to
Flash Memory\n");
       LPC_GPIO0->FIOCLR = (1<<6);
       //*************************Write
from buffer1 to main memory
       //Opcode for write from buffer1 to
main memory with built-in erase
       SSP1exchangeByte(0x83);
       //Send 3 Bytes = 24 bits= 4 don't
care bits + 12 addr bits (A19-A8) + 8
don't care bits
       SSP1exchangeByte(0x00);
       SSP1exchangeByte(0x00);
       SSP1exchangeByte(0x00);
       // SSP1exchangeByte(0xff);
       LPC_GPIO2->FIOSET = (1<<6);
       for ( i = 0; i < 10000; i++ );

       //*********************************
****************************************
************************************

       //PART B Verification
       initSSP1();
       printf("\n************************
****************************************
***********\n");
       printf("\nWriting from Flash Memory
to Buffer 2\n");
       LPC_GPIO0->FIOCLR = (1<<6);
       //*************************Read
from main memory to buff 1
       //Opcode for Read from main memory
to buff 1
       SSP1exchangeByte(0x55);
       //Send 3 Bytes = 24 bits= 4 don't
care bits + 12 addr bits (A19-A8) + 8
don't care bits
       SSP1exchangeByte(0x00);
       SSP1exchangeByte(0x00);
       SSP1exchangeByte(0x00);
       LPC_GPIO2->FIOSET = (1<<6);
       for ( i = 0; i < 10000; i++ );
       printf("\n************************
****************************************
***********\n");

       initSSP1();
       printf("\nReading from Buffer
2\n");
       LPC_GPIO0->FIOCLR = (1<<6);
       //*************************Read
from buff 1
       //Opcode for Read buff 1
       SSP1exchangeByte(0xd6);
       //Send 3 Bytes = 24 bits= 16 don't
care bits + 8 buffer addr bits ????
       SSP1exchangeByte(0x00);
       SSP1exchangeByte(0x00);
       SSP1exchangeByte(0x00);   //trial
for buff1
       //Send 1 dummy byte
       printf("\n",SSP1exchangeByte(0x00))
;

       int counter = 0;
       for(counter=0;
counter<35;counter++){
       printf("\n %i\t
%c",i,SSP1exchangeByte(0x00));
       }
//     printf("\n 1\t
%c",SSP1exchangeByte(0x00));
//     printf("\n 2\t
%c",SSP1exchangeByte(0x00));
//     printf("\n 3\t
%c",SSP1exchangeByte(0x00));
//     printf("\n 4\t
%c",SSP1exchangeByte(0x00));
       printf("\n************************
****************************************
***********\n");
       LPC_GPIO2->FIOSET = (1<<6);
       for ( i = 0; i < 10000; i++ );
       //LPC_GPIO0->FIOCLR = (1<<6);


       while(1);
       //delayMs(0,100);
       // printf("\n1\t%x", in);
       //extern void SSPReceive( 1,
uint8_t *buf, uint32_t Length );

       return 0;
       }
```