**AT45 Flash Memory Module**

Anahit Sarao

*Computer Engineering Department, College of Engineering*

*San Jose State University, San Jose, CA 94303*

*E-mail:* indianvip60@gmail.com

**Abstract**

*The following report will explain step by step the designing and building process to connect an AT45 module to the LPC1769 microcontroller. The AT45 is a flash memory module which uses serial peripheral interface also knowns as SPI to allow data transfer from the host. This project report thoroughly describes the hardware and software connectivity.*

**1. Introduction**

Being able to store and manipulate data is a fundamental function of today’s world. Since the beginning of man the ability to store and preserve data has improved significantly. Devices in the modern day store data in large memory modules. One way to store data is to flash is onto a unit, hence the name flash memory. Another method of storage is RAM which stands for random access memory. However, RAM is erased in the absence of power, this can be a major problem if the data is important. Flash memory solves this problem by writing data onto the memory array which is retained even if power is not present. This solves many different problems to prove this concept the microcontroller will read and write to the AT45 module.

**2. Methodology**

To provide an elaborate but concise and effective report; steps were taken to achieve high quality work. This document is an official representation of all the designs, material, documentation produced by the author stated above. Each section will provide up to date specifications.

**2.1. Objectives and Technical Challenges**

To be able to test memory module many ports were used such as MOSI1, SCK1, SSEL1, MISO1. The ports allow the memory module to act as a slave to the microcontroller. Once data is send to the first buffer the memory is put into write mode, once the writing is complete the memory is put into read mode and the data can be read. The code was written inside LPCX using the header files provided, this allows us to verify the memory module through code by reading and writing.

**2.2. Problem Formulation and Design**

The interface between the microcontroller and the flash module has a connection using the SPI interface. To learn SPI protocol, the user can learn from the AT45 datasheet and project material slides. Lastly, a message is constructed which is written to the memory, the board is shut down and the message is read to the console. This shows full verification of a working flash module. The first module was soldered onto a SOIC but however it was already shorted from factory. A second memory module was acquired and fitted into a SOIC adapter which was functional. Below table two shows the bill of materials used for the project.

Table 1. Bill of Material

|  |  |
| --- | --- |
| Name | Description |
| LPC1769 rev D | ARM Cortex CPU Module |
| Switch x 2 | SPDT Switch |
| Resistor | 1K Ohms |
| 9 Volt Adapter |  |
| LM7805 | Voltage regulator |
| Red LED |  |
| Resistor x 3 | 390 Ohms |
| Green Led |  |

**3. Implementation**

The following sections provide technical and analytical information for the hardware and software design.

**3.1. Hardware Design**

Figure one provides an abstraction level block diagram of the major components of the circuit. The memory module is connected to the microcontroller using various pins. This project is not dependent on the power

supply and GPIO testing circuit build in the previous project.

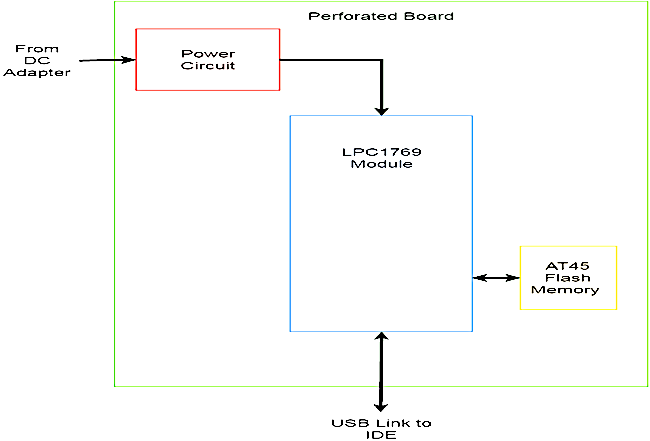
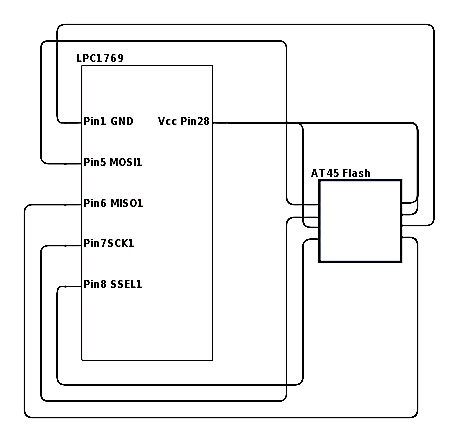
Figure 1. Project Block Diagram

Figure 2. Hardware Connection Diagram



The data transfer pins are MOSI, MISO, SCK, and SSEL. These pins allow the transfer of data between the AT45 and the microcontroller. The reset the Vcc are connected to the Vcc output of the microcontroller which provides a 3.3V. The gnd is connected to the common ground rail. Table two shows the AT45 pin connections which directly connect to the microcontroller. While figure two shows a hardware schematic of the pins used.

Table 2. Pin Connections.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| AT45 Flash Memory | | | LPC1769 | | |
| Pin No. | Pin Label | Description | Pin No. | Pin Label | Description |
| 1 | SI | Slave Input | 5 | MOSI1 | Master Output Slave Input |
| 2 | SCK | Serial Clock | 7 | SCK1 | Serial Clock |
| 3 | nReset | Active Low Reset | 28 | Vcc | 3.3V Power Supply |
| 4 | nCS | Active Low Chip Select | 8 | SSEL1 | Slave Select |
| 5 | nWP | Active Low Write Protect | 28 | Vcc | 3.3V Power Supply |
| 6 | Vcc | 3.3V Power Supply | 28 | Vcc | 3.3V Power Supply |
| 7 | GND | Common Ground | 1 | GND | Common Ground |
| 8 | SO | Slave Output | 6 | MISO1 | Master Input Slave Output |

**3.2. Software Design**

To be able to have a valid software design is to be able to properly utilize the microcontroller. The microcontroller uses a JTAG connection to be able to transfer user written program onto the flash memory. This also allows the user to use the debugging features and many other enhancements such as SWO profiling, memory watching, power management toolkit. The code is also dependent on the core project which is provided by NXP. Figure three shows the software level block diagram.

Figure 3. Software Connection Diagram

C:\Users\india\Downloads\Chrome-DL\Copy of Untitled Diagram.png

The software design was very simple once the hardware was constructed. Using the provided CMSIS\_CORE\_LPC17xx project which contains the needed header and booting configurations to flash and connect to the microcontroller through USB. A new project was started and a library link to the core package was initialized. The software uses the initSSP1() to initialize the SSP1 connection. After the initialization, the software uses the proper opcode to utilize certain functions. Table three shows the OpCodes and their corresponding functionalities.

Table 3. OpCode Functions

|  |  |
| --- | --- |
| OpCode | Function |
| 0x9F | Get Device & Manufacturer ID |
| 0x84 | Buffer Write |
| 0xD4 | Buffer Read (High Frequency) |
| 0x83 | Buffer to Main Memory Page Program with Built-In Erase |
| 0x53 | Main Memory Page to Buffer Transfer |

The basic flow of the software was to initialize the SPI interface. Then write to the buffer and read from the same buffer. After it is verified that the first buffer was able to read and write the second flash buffer is initialized. This buffer will actually write the data array of the flash memory. This will allow the data to be retained after power has been turned off to the memory module. Once the board has lost power it is given power and the memory module is initialized again and it begins to read from the data array. The flow of this software process is shown in figure four below.

Program flowchart

Figure 4. Software Flowchart

The algorithm for writing to the memory flash module.

Opcode 0x84 is sent to the device, to transfer data from the array to buffer.

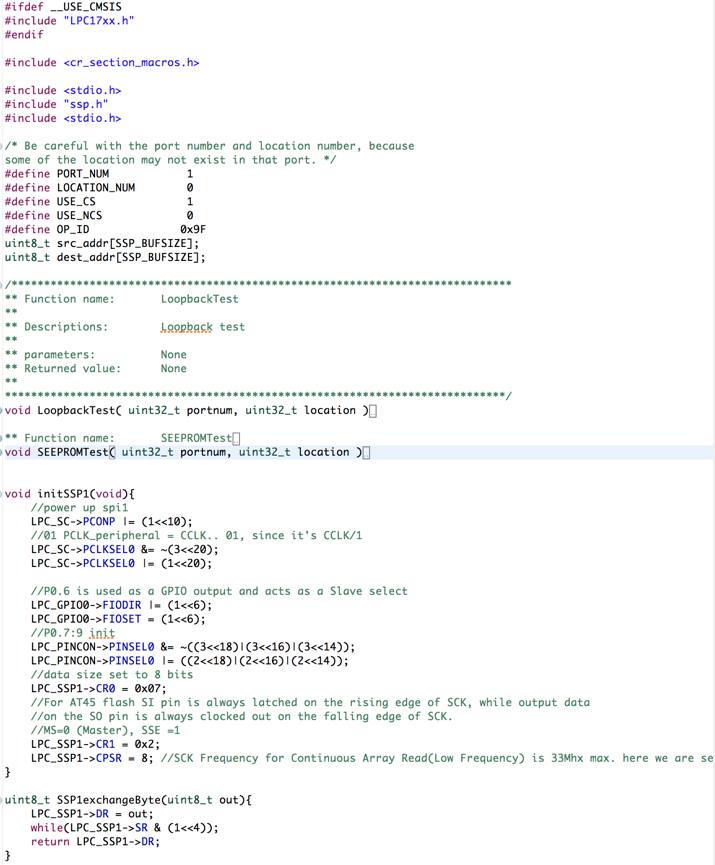
Opcode 0x83 is again sent to the device, to read from the buffer. The read message is displayed on the console.

The algorithm for reading the memory is a similar to the steps which are implemented for the writing and getting the device and manufacturer id.

Opcode 0x53 is sent to the device, to transfer data from the array to buffer.

Opcode 0xD4 is again sent to the device, to read from the buffer. The read message is displayed on the console.

Figure 4. Code for Initializing and Transferring



**4. Testing and Verification**

The steps taken for verifying the hardware and software cohesion are list as:

1. Load code onto LPC board
2. Write to buffer
3. Turn of board
4. Read from buffer

By following these steps, it was verified that the board was programmed and the hardware was working correctly. Figure five and six show the output of the console when the memory module is used.

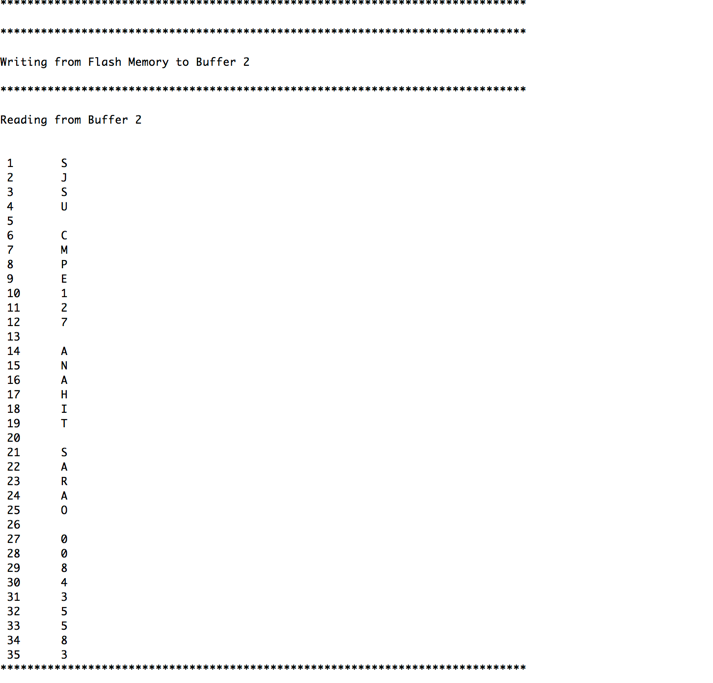
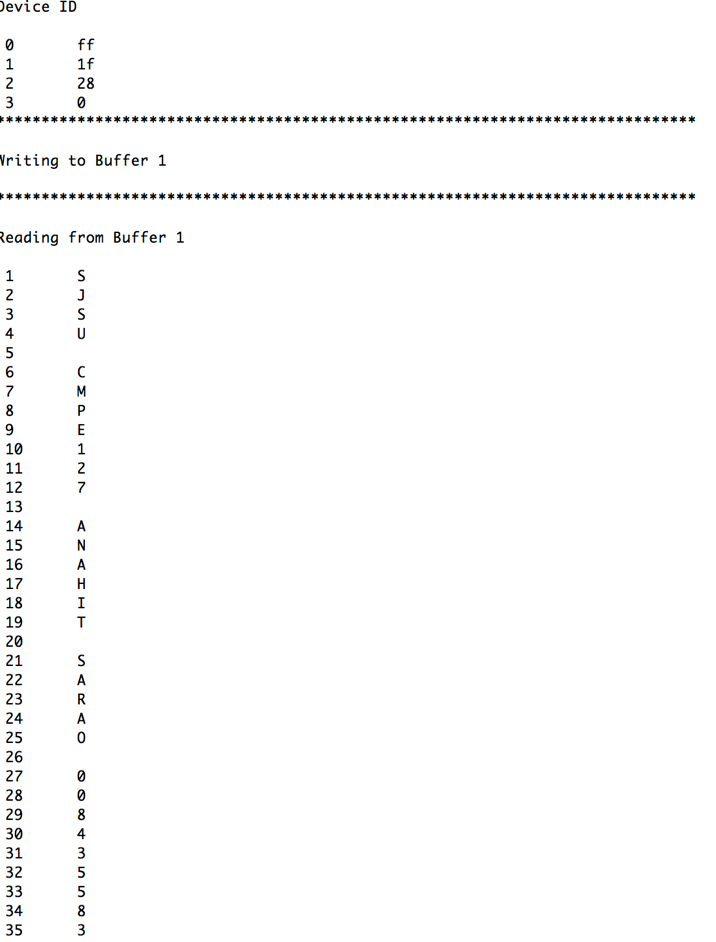


Figure 6. Read Console Output

****

The verification for this lab is successful as the console displays the manufacturing ID plus the reading and writing process.

**5. Conclusion**

In conclusion, this project was very successful in helping understand and get familiar with LPCX and the SPI connection of microcontrollers. The board now has an external memory module which can store data and retain it after power loss. This is a key function which is built into almost all devices in modern day embedded electronics. A more advance design would be to add parallel reading and writing to the process is quicker. The AT45 opens up so much more options and shows how useful microprocessors are in the modern day. This experience was very positive and educating even though there were many problems such as wires detaching, connections touching other junctions and the board not being fully utilized by the IDE. Additional figures are given within the appendix to show the functionality and verification of this lab.

**6. Acknowledgement**

Provide acknowledgement if needed, such as support, help, or assistance from someone. These support, help, assistance are crucial.

**7. References**

[1] H. Li, “Author Guidelines for CMPE 146/242 Project Report”, *Lecture Notes of CMPE 146/242*, Computer Engineering Department, College of Engineering, San Jose State University, March 6, 2006, pp. 1.

**8. Appendix**

Figure 7. SOIC connected to Microcontroller

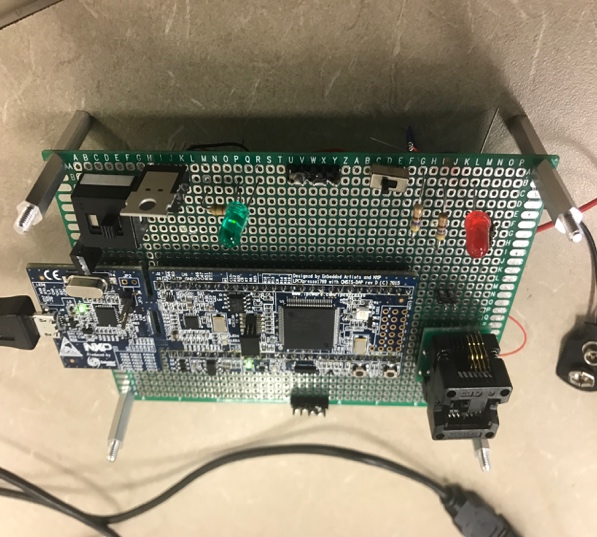
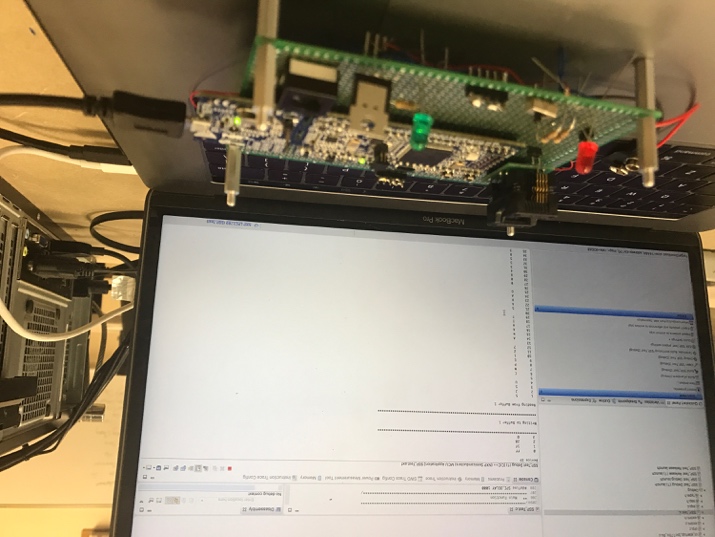


Figure 8. Writing to Flash Memory

****

Figure 9 Writing to Flash Memory



Source Code:

/\*

===============================================================================

Name : SSP\_Test.c

Author : $(author)

Version :

Copyright : $(copyright)

Description : main definition

===============================================================================

\*/

#ifdef \_\_USE\_CMSIS

#include "LPC17xx.h"

#endif

#include <cr\_section\_macros.h>

#include <stdio.h>

#include "ssp.h"

#include <stdio.h>

/\* Be careful with the port number and location number, because

some of the location may not exist in that port. \*/

#define PORT\_NUM 1

#define LOCATION\_NUM 0

#define USE\_CS 1

#define USE\_NCS 0

#define OP\_ID 0x9F

uint8\_t src\_addr[SSP\_BUFSIZE];

uint8\_t dest\_addr[SSP\_BUFSIZE];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name: LoopbackTest

\*\*

\*\* Descriptions: Loopback test

\*\*

\*\* parameters: None

\*\* Returned value: None

\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void LoopbackTest( uint32\_t portnum, uint32\_t location )

{

uint32\_t i;

#if !USE\_CS

/\* Set SSEL pin to output low. \*/

SSP\_SSELToggle( portnum, 0 );

#endif

i = 0;

while ( i <= SSP\_BUFSIZE )

{

/\* to check the RXIM and TXIM interrupt, I send a block data at one time

based on the FIFOSIZE(8). \*/

SSPSend( portnum, (uint8\_t \*)&src\_addr[i], FIFOSIZE );

/\* If RX interrupt is enabled, below receive routine can be

also handled inside the ISR. \*/

SSPReceive( portnum, (uint8\_t \*)&dest\_addr[i], FIFOSIZE );

i += FIFOSIZE;

}

#if !USE\_CS

/\* Set SSEL pin to output high. \*/

SSP\_SSELToggle( portnum, 1 );

#endif

/\* verifying write and read data buffer. \*/

for ( i = 0; i < SSP\_BUFSIZE; i++ )

{

if ( src\_addr[i] != dest\_addr[i] )

{

while( 1 ); /\* Verification failed \*/

}

}

return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name: SEEPROMTest

\*\*

\*\* Descriptions: Serial EEPROM(Atmel 25xxx) test

\*\*

\*\* parameters: None

\*\* Returned value: None

\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void SEEPROMTest( uint32\_t portnum, uint32\_t location )

{

uint32\_t i, timeout;

#if SSP\_DEBUG

uint8\_t temp[2];

#endif

if ( portnum == 1 )

{

LPC\_GPIO0->FIODIR |= (0x1<<16); /\* SSP1, P0.16 defined as Outputs \*/

}

else

{

LPC\_GPIO0->FIODIR |= (0x1<<6); /\* SSP0 P0.6 defined as Outputs \*/

}

SSP\_SSELToggle( portnum, 0 );

/\* Test Atmel 25016 SPI SEEPROM. \*/

src\_addr[0] = WREN; /\* set write enable latch \*/

SSPSend( portnum, (uint8\_t \*)src\_addr, 1 );

SSP\_SSELToggle( portnum, 1 );

for ( i = 0; i < DELAY\_COUNT; i++ ); /\* delay minimum 250ns \*/

SSP\_SSELToggle( portnum, 0 );

src\_addr[0] = RDSR; /\* check status to see if write enabled is latched \*/

SSPSend( portnum, (uint8\_t \*)src\_addr, 1 );

SSPReceive( portnum, (uint8\_t \*)dest\_addr, 1 );

SSP\_SSELToggle( portnum, 1 );

if ( dest\_addr[0] & (RDSR\_WEN|RDSR\_RDY) != RDSR\_WEN )

/\* bit 0 to 0 is ready, bit 1 to 1 is write enable \*/

{

while ( 1 );

}

for ( i = 0; i < SSP\_BUFSIZE; i++ ) /\* Init RD and WR buffer \*/

{

src\_addr[i+3] = i; /\* leave three bytes for cmd and offset(16 bits) \*/

dest\_addr[i] = 0;

}

/\* please note the first two bytes of WR and RD buffer is used for

commands and offset, so only 2 through SSP\_BUFSIZE is used for data read,

write, and comparison. \*/

SSP\_SSELToggle( portnum, 0 );

src\_addr[0] = WRITE; /\* Write command is 0x02, low 256 bytes only \*/

src\_addr[1] = 0x00; /\* write address offset MSB is 0x00 \*/

src\_addr[2] = 0x00; /\* write address offset LSB is 0x00 \*/

SSPSend( portnum, (uint8\_t \*)src\_addr, SSP\_BUFSIZE );

SSP\_SSELToggle( portnum, 1 );

for ( i = 0; i < 0x30000; i++ ); /\* delay, minimum 3ms \*/

timeout = 0;

while ( timeout < MAX\_TIMEOUT )

{

SSP\_SSELToggle( portnum, 0 );

src\_addr[0] = RDSR; /\* check status to see if write cycle is done or not \*/

SSPSend( portnum, (uint8\_t \*)src\_addr, 1);

SSPReceive( portnum, (uint8\_t \*)dest\_addr, 1 );

SSP\_SSELToggle( portnum, 1 );

if ( (dest\_addr[0] & RDSR\_RDY) == 0x00 ) /\* bit 0 to 0 is ready \*/

{

break;

}

timeout++;

}

if ( timeout == MAX\_TIMEOUT )

{

while ( 1 );

}

for ( i = 0; i < DELAY\_COUNT; i++ ); /\* delay, minimum 250ns \*/

SSP\_SSELToggle( portnum, 0 );

src\_addr[0] = READ; /\* Read command is 0x03, low 256 bytes only \*/

src\_addr[1] = 0x00; /\* Read address offset MSB is 0x00 \*/

src\_addr[2] = 0x00; /\* Read address offset LSB is 0x00 \*/

SSPSend( portnum, (uint8\_t \*)src\_addr, 3 );

SSPReceive( portnum, (uint8\_t \*)&dest\_addr[3], SSP\_BUFSIZE-3 );

SSP\_SSELToggle( portnum, 1 );

/\* verifying, ignore the difference in the first two bytes \*/

for ( i = 3; i < SSP\_BUFSIZE; i++ )

{

if ( src\_addr[i] != dest\_addr[i] )

{

while( 1 ); /\* Verification failed \*/

}

}

return;

}

void initSSP1(void){

//power up spi1

LPC\_SC->PCONP |= (1<<10);

//01 PCLK\_peripheral = CCLK.. 01, since it's CCLK/1

LPC\_SC->PCLKSEL0 &= ~(3<<20);

LPC\_SC->PCLKSEL0 |= (1<<20);

//P0.6 is used as a GPIO output and acts as a Slave select

LPC\_GPIO0->FIODIR |= (1<<6);

LPC\_GPIO0->FIOSET = (1<<6);

//P0.7:9 init

LPC\_PINCON->PINSEL0 &= ~((3<<18)|(3<<16)|(3<<14));

LPC\_PINCON->PINSEL0 |= ((2<<18)|(2<<16)|(2<<14));

//data size set to 8 bits

LPC\_SSP1->CR0 = 0x07;

//For AT45 flash SI pin is always latched on the rising edge of SCK, while output data

//on the SO pin is always clocked out on the falling edge of SCK.

//MS=0 (Master), SSE =1

LPC\_SSP1->CR1 = 0x2;

LPC\_SSP1->CPSR = 8; //SCK Frequency for Continuous Array Read(Low Frequency) is 33Mhx max. here we are setting it below it.

}

uint8\_t SSP1exchangeByte(uint8\_t out){

LPC\_SSP1->DR = out;

while(LPC\_SSP1->SR & (1<<4));

return LPC\_SSP1->DR;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Main Function

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#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);

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);

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('A');

SSP1exchangeByte('N');

SSP1exchangeByte('A');

SSP1exchangeByte('H');

SSP1exchangeByte('I');

SSP1exchangeByte('T');

SSP1exchangeByte(' ');

SSP1exchangeByte('S');

SSP1exchangeByte('A');

SSP1exchangeByte('R');

SSP1exchangeByte('A');

SSP1exchangeByte('O');

SSP1exchangeByte(' ');

SSP1exchangeByte('0');

SSP1exchangeByte('0');

SSP1exchangeByte('8');

SSP1exchangeByte('4');

SSP1exchangeByte('3');

SSP1exchangeByte('5');

SSP1exchangeByte('5');

SSP1exchangeByte('8');

SSP1exchangeByte('3');

LPC\_GPIO2->FIOSET = (1<<6);

for ( i = 0; i < 10000; i++ );

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

initSSP1();

printf("\nReading from Buffer 1\n");

LPC\_GPIO0->FIOCLR = (1<<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

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 5\t %c",SSP1exchangeByte(0x00));

printf("\n 6\t %c",SSP1exchangeByte(0x00));

printf("\n 7\t %c",SSP1exchangeByte(0x00));

printf("\n 8\t %c",SSP1exchangeByte(0x00));

printf("\n 9\t %c",SSP1exchangeByte(0x00));

printf("\n 10\t %c",SSP1exchangeByte(0x00));

printf("\n 11\t %c",SSP1exchangeByte(0x00));

printf("\n 12\t %c",SSP1exchangeByte(0x00));

printf("\n 13\t %c",SSP1exchangeByte(0x00));

printf("\n 14\t %c",SSP1exchangeByte(0x00));

printf("\n 15\t %c",SSP1exchangeByte(0x00));

printf("\n 16\t %c",SSP1exchangeByte(0x00));

printf("\n 17\t %c",SSP1exchangeByte(0x00));

printf("\n 18\t %c",SSP1exchangeByte(0x00));

printf("\n 19\t %c",SSP1exchangeByte(0x00));

printf("\n 20\t %c",SSP1exchangeByte(0x00));

printf("\n 21\t %c",SSP1exchangeByte(0x00));

printf("\n 22\t %c",SSP1exchangeByte(0x00));

printf("\n 23\t %c",SSP1exchangeByte(0x00));

printf("\n 24\t %c",SSP1exchangeByte(0x00));

printf("\n 25\t %c",SSP1exchangeByte(0x00));

printf("\n 26\t %c",SSP1exchangeByte(0x00));

printf("\n 27\t %c",SSP1exchangeByte(0x00));

printf("\n 28\t %c",SSP1exchangeByte(0x00));

printf("\n 29\t %c",SSP1exchangeByte(0x00));

printf("\n 30\t %c",SSP1exchangeByte(0x00));

printf("\n 31\t %c",SSP1exchangeByte(0x00));

printf("\n 32\t %c",SSP1exchangeByte(0x00));

printf("\n 33\t %c",SSP1exchangeByte(0x00));

printf("\n 34\t %c",SSP1exchangeByte(0x00));

printf("\n 35\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));

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 5\t %c",SSP1exchangeByte(0x00));

printf("\n 6\t %c",SSP1exchangeByte(0x00));

printf("\n 7\t %c",SSP1exchangeByte(0x00));

printf("\n 8\t %c",SSP1exchangeByte(0x00));

printf("\n 9\t %c",SSP1exchangeByte(0x00));

printf("\n 10\t %c",SSP1exchangeByte(0x00));

printf("\n 11\t %c",SSP1exchangeByte(0x00));

printf("\n 12\t %c",SSP1exchangeByte(0x00));

printf("\n 13\t %c",SSP1exchangeByte(0x00));

printf("\n 14\t %c",SSP1exchangeByte(0x00));

printf("\n 15\t %c",SSP1exchangeByte(0x00));

printf("\n 16\t %c",SSP1exchangeByte(0x00));

printf("\n 17\t %c",SSP1exchangeByte(0x00));

printf("\n 18\t %c",SSP1exchangeByte(0x00));

printf("\n 19\t %c",SSP1exchangeByte(0x00));

printf("\n 20\t %c",SSP1exchangeByte(0x00));

printf("\n 21\t %c",SSP1exchangeByte(0x00));

printf("\n 22\t %c",SSP1exchangeByte(0x00));

printf("\n 23\t %c",SSP1exchangeByte(0x00));

printf("\n 24\t %c",SSP1exchangeByte(0x00));

printf("\n 25\t %c",SSP1exchangeByte(0x00));

printf("\n 26\t %c",SSP1exchangeByte(0x00));

printf("\n 27\t %c",SSP1exchangeByte(0x00));

printf("\n 28\t %c",SSP1exchangeByte(0x00));

printf("\n 29\t %c",SSP1exchangeByte(0x00));

printf("\n 30\t %c",SSP1exchangeByte(0x00));

printf("\n 31\t %c",SSP1exchangeByte(0x00));

printf("\n 32\t %c",SSP1exchangeByte(0x00));

printf("\n 33\t %c",SSP1exchangeByte(0x00));

printf("\n 34\t %c",SSP1exchangeByte(0x00));

printf("\n 35\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;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* End Of File

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/