|  |  |
| --- | --- |
| Ex.no: 1  Date: | **A User space program to toggle any one of the LED in Rugged Board A5d2x** |

**AIM:**

To Write a user space program to toggle any one of the LED in Rugged Board A5d2x

**ALGORITHM:**

**STEP 1:** Start the program.

**STEP 2:** Include necessary header files for system calls (sys/types.h, sys/stat.h, fcntl.h, unistd.h) and standard I/O operations (stdio.h, string.h, stdlib.h).

**STEP 3:** Define macros for the GPIO paths (GPIO\_EN, GPIO\_DIR, GPIO\_Val) based on the file structure of sysfs.

**STEP 4:**  Define variables: gpio\_fd for file descriptors, gpio\_buf for storing GPIO number as a string, gpio\_num for GPIO pin number, and count for loop iteration.

**STEP 5:** Open the GPIO export file (GPIO\_EN) to enable GPIO pins.

**STEP 6:** Write the GPIO pin number to the export file to enable the specified GPIO pin.

**STEP 7:** Close the export file.

**STEP 8:** Open the direction file (GPIO\_DIR) of the GPIO pin to set it as an output.

**STEP 9:** Write "out" to the direction file to configure the GPIO pin as an output pin.

**STEP 10:** Close the direction file.

**STEP 11:** Open the value file (GPIO\_Val) of the GPIO pin to write values to it.

**STEP 12:** Print a message indicating the GPIO pin toggling will start.

**STEP 13:** Toggle the GPIO pin by writing "0" and "1" alternately to the value file with a delay of 2 seconds using a loop.

**STEP 14:** Close the value file.

**STEP 15:** End the program.

**PROGRAM:**

**blink.c**

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#define GPIO\_EN "/sys/class/gpio/export"

#define GPIO\_DIR "/sys/class/gpio/PC13/direction"

#define GPIO\_Val "/sys/class/gpio/PC13/value"

int blink()

{

int gpio\_fd;

char gpio\_buf[30];

int gpio\_num = 77;

int count =10;

gpio\_fd = open(GPIO\_EN,O\_WRONLY);

if(gpio\_fd < 0)

{

printf("Unable to open the file %s\n",GPIO\_EN);

exit(0);

}

sprintf(gpio\_buf,"%d",gpio\_num);

write(gpio\_fd,gpio\_buf,strlen(gpio\_buf));

close(gpio\_fd);

gpio\_fd = open(GPIO\_DIR,O\_WRONLY);

if(gpio\_fd < 0)

{

printf("Unable to open the file %s",GPIO\_DIR);

exit(0);

}

write(gpio\_fd,"out",3);

close(gpio\_fd);

gpio\_fd = open( GPIO\_Val,O\_WRONLY);

if(gpio\_fd < 0)

{

printf("Unable to open the file %s",GPIO\_Val);

exit(0);

}

printf("Toggling the GPIO\_PIN\n");

while(count--)

{

write(gpio\_fd,"0" ,1);

sleep(2);

write(gpio\_fd,"1" ,1);

sleep(2);

}

close(gpio\_fd);

}

**blink.h**

int blink();

**main.c**

#include<stdio.h>

#include "blink.h"

int main()

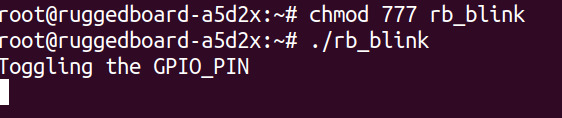
{

blink();

return 0;

}

**OUTPUT :**

****

**RESULT:**

|  |  |
| --- | --- |
| Ex.no: 2  Date: | **A User space program to blink an LED using a user switch using Rugged Board A5D2X** |

**AIM:**

To write a user space program to blink an LED using a user switch using Rugged Board A5d2x.

**ALGORITHM:**

**STEP 1:** Start the program.

**STEP 2:** Include necessary header files for system calls and standard I/O operations.

**STEP 3:** Define macros for the GPIO paths (GPIO\_EN, GPIO\_DIR, GPIO\_Val) based on the file structure of sysfs.

**STEP 4:** Define variables: gpio\_fd, fd for file descriptors, gpio\_buf for storing GPIO number as a string, gpio\_num for GPIO pin number, and value for reading the input GPIO pin value.

**STEP 5:** Open the GPIO export file (GPIO\_EN) to enable GPIO pins.

**STEP 6:** Write the GPIO pin number to the export file to enable the specified GPIO pin (PC13).

**STEP 7:** Close the export file.

**STEP 8:** Open the direction file (GPIO\_DIR) of the GPIO pin to set it as an output.

**STEP 9:** Write "out" to the direction file to configure the GPIO pin as an output pin.

**STEP 10:** Close the direction file.

**STEP 11:** Open the value file (GPIO\_Val) of the GPIO pin to write values to it.

**STEP 12:** Export the second GPIO pin (PC12) and set its direction as input.

**STEP 13:** Read the value of the input GPIO pin (PC12) in a loop and write its value to the output GPIO pin (PC13).

**STEP 14:** Close all file descriptors.

**STEP 15:** End the program

.

**PROGRAM:**

**switch.c**

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include"userswitch.h"

#define GPIO\_EN "/sys/class/gpio/export"

#define GPIO\_DIR "/sys/class/gpio/PC13/direction"

#define GPIO\_Val "/sys/class/gpio/PC13/value"

int switches()

{

int gpio\_fd,fd;

char gpio\_buf[30];

int gpio\_num = 77;

char value;

gpio\_fd = open(GPIO\_EN,O\_WRONLY);

if(gpio\_fd < 0)

{

printf("Unable to open the file %s\n",GPIO\_EN);

exit(0);

}

sprintf(gpio\_buf,"%d",gpio\_num);

write(gpio\_fd,gpio\_buf,strlen(gpio\_buf));

close(gpio\_fd);

gpio\_fd = open(GPIO\_DIR,O\_WRONLY);

if(gpio\_fd < 0)

{

printf("Unable to open the file %s",GPIO\_DIR);

exit(0);

}

write(gpio\_fd,"out",3);

close(gpio\_fd);

gpio\_fd = open( GPIO\_Val,O\_WRONLY);

if(gpio\_fd < 0)

{

printf("Unable to open the file %s",GPIO\_Val);

exit(0);

}

fd = open("/sys/class/gpio/export",O\_WRONLY);

if(fd < 0)

{

printf("Unable to export the file PC12 \n");

exit(0);

}

write(fd,"76",3);

close(fd);

fd =open("/sys/class/gpio/PC12/direction",O\_WRONLY);

if(fd < 0)

{

printf("Unable to open the file direction \n");

exit(0);

}

write(fd,"in",3);

close(fd);

while(10)

{

fd = open("/sys/class/gpio/PC12/value",O\_RDONLY);

if(fd < 0)

{

printf("Unable to open the file \n");

exit(0);

}

read(fd,&value,1);

if(value=='0')

write(gpio\_fd,"0" ,1);

else

write(gpio\_fd,"1" ,1);

close(fd);

}

close(gpio\_fd);

}

**switch.h**

int switches();

**main.c**

#include "userswitch.h"

#include <stdio.h>

int main()

{

switches();

return 0;

}

**OUTPUT:**

|  |  |
| --- | --- |
| Ex.no 3  Date | **A user space program to communicate**  **with uart (transmit) in Rugged Board** |

**AIM:**

To Write a user space program to communicate with uart (transmit).

**ALGORITHM:**

**STEP 1: Prepare the Development Environment:**

- Open terminal and navigate to the directory containing your UART source files (uart\_tx.c ).

- Set up the cross-compilation environment for your target platform by executing:

**. /opt/poky-tiny/2.5.2/environment-setup-cortexa5hf-neon-poky-linux-musleabi**

**STEP 2: Compile Source Files:**

- Compile the transmitter (uart\_tx.c) source files using the cross-compiler by executing:

${CC} uart\_tx.c -o uart\_tx

**STEP 3: Copy Executables to Target Board:**

- Copy the compiled executables (uart\_tx ) to the TFTP server directory on the target board by executing:

sudo cp uart\_tx /var/lib/tftpboot/

**STEP 4: Setup Hardware Connection:**

- Connect the jumper wire from the UART TX pin of the rugged board to the RX pin on the micro bus, effectively creating a loopback connection.

**STEP 5: Configure Rugged Board:**

- Access the terminal of the rugged board and login as root.

- Configure the Ethernet interface with an IP address, for example:

ifconfig eth0 192.168.1.35

- Verify the Ethernet link status.

- Use TFTP to retrieve the transmiter executable (uart\_tx) from your development machine to the target board:

tftp -r uart\_tx -g 192.168.1.30

**STEP 6: Run Receiver on Rugged Board:**

- Set appropriate permissions for the received executable:

chmod 777 uart\_tx

- Execute the receiver program:

./uart\_tx

**STEP 7: Transmit Data from Development Machine:**

- On your development machine, execute the transmitter program (uart\_tx).

- Enter the string you want to transmit when prompted. For example:

Enter the string to transmit: Hello

**STEP 8: Verify Reception on Rugged Board:**

- Check the rugged board's terminal where the receiver program is running. It should display the received data. For example:

Received: Hello

**PROGRAM:**

**Transmitter\_uart.h:**

#ifndef UART\_H

#define UART\_H

#include <stdio.h>

#include <termios.h>

int set\_interface\_attribs(int fd, int speed);

void uart\_transmit(int fd, const char\* data, int size);

#endif /\* UART\_H \*/

**Transmitter\_uart.c:**

#include "transmitter\_uart.h"

#include <errno.h>

#include <fcntl.h>

#include <string.h>

#include <unistd.h>

int set\_interface\_attribs(int fd, int speed) {

struct termios tty;

if (tcgetattr(fd, &tty) < 0) {

printf("Error from tcgetattr: %s\n", strerror(errno));

return -1;

}

cfsetispeed(&tty, (speed\_t)speed);

cfsetospeed(&tty, (speed\_t)speed);

tty.c\_cflag |= (CLOCAL | CREAD);

tty.c\_cflag &= ~CSIZE;

tty.c\_cflag |= CS8;

tty.c\_cflag &= ~PARENB;

tty.c\_cflag &= ~CSTOPB;

tty.c\_cflag &= ~CRTSCTS;

tty.c\_iflag = IGNPAR;

tty.c\_oflag = 0;

tty.c\_lflag = 0;

tty.c\_cc[VMIN] = 0

tty.c\_cc[VTIME] = 5

if (tcsetattr(fd, TCSANOW, &tty) != 0) {

printf("Error from tcsetattr: %s\n", strerror(errno));

return -1;

}

return 0;

}

void uart\_transmit(int fd, const char\* data, int size) {

int wlen = write(fd, data, size);

if (wlen != size) {

printf("Error writing to serial port\n");

}

}

**Transmitter\_main.c:**

#include "transmitter\_uart.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include <fcntl.h>

#include <unistd.h>

int main() {

char \*portname = "/dev/ttyS3";

int fd;

char input[100];

fd = open(portname, O\_RDWR | O\_NOCTTY | O\_SYNC);

if (fd < 0) {

printf("Error opening %s: %s\n", portname, strerror(errno));

return -1;

}

if (set\_interface\_attribs(fd, B9600) < 0) {

printf("Error setting serial port attributes\n");

close(fd);

return -1;

}

while (1) {

fgets(input, sizeof(input), stdin);

input[strcspn(input, "\n")] = '\0'; // Remove newline character

uart\_transmit(fd, input, strlen(input));

usleep(100000); // 100 ms

}

close(fd);

return 0;

4 RECEIVER

**AIM:**

To Write a user space program to communicate with uart (receiver).

**ALGORITHM:**

**STEP 1: Prepare the Development Environment:**

- Open terminal and navigate to the directory containing your UART source files (uart\_rx.c ).

- Set up the cross-compilation environment for your target platform by executing:

**. /opt/poky-tiny/2.5.2/environment-setup-cortexa5hf-neon-poky-linux-musleabi**

**STEP 2: Compile Source Files:**

- Compile the receiver (uart\_rx.c) source files using the cross-compiler by executing:

${CC} uart\_rx.c -o uart\_rx

**STEP 3: Copy Executables to Target Board:**

- Copy the compiled executables (uart\_rx ) to the TFTP server directory on the target board by executing:

sudo cp uart\_rx /var/lib/tftpboot/

**STEP 4: Ping the Rugged Board:**

- Open a terminal on your development machine.

- Ping the IP address of the rugged board (192.168.1.35) to ensure connectivity:

ping 192.168.1.35

**STEP 5: SSH into the Rugged Board:**

- SSH into the rugged board using the root account:

ssh [root@192.168.1.35](mailto:root@192.168.1.35)

**STEP 6:TFTP to retrieve the receiver executable:**

- Use TFTP to retrieve the recevier executable (uart\_rx) from your development machine to the target board:

tftp -r uart\_rx -g 192.168.1.30

**STEP 7: Run Receiver on Rugged Board:**

- In the rugged board's terminal, navigate to the directory where the receiver executable (uart\_rx) is located.

- Execute the receiver program:

./uart\_rx

Received: hello

- The receiver program will start listening for incoming data.

**PROGRAM:**

**receive\_uart.h:**

#ifndef RECEIVE\_UART\_H

#define RECEIVE\_UART\_H

#include <stdio.h>

#include <fcntl.h>

#include <string.h>

#include <termios.h>

#include <unistd.h>

#include <errno.h>

int set\_interface\_attribs(int fd, int speed);

void receive\_data(int fd);

#endif /\* RECEIVE\_UART\_H \*/

**receive\_uart.c:**

#include "receive\_uart.h"

int set\_interface\_attribs(int fd, int speed) {

struct termios tty;

if (tcgetattr(fd, &tty) < 0) {

printf("Error from tcgetattr: %s\n", strerror(errno));

return -1;

}

cfsetispeed(&tty, (speed\_t)speed);

tty.c\_cflag |= (CLOCAL | CREAD);

tty.c\_cflag &= ~CSIZE;

tty.c\_cflag |= CS8;

tty.c\_cflag &= ~PARENB;

tty.c\_cflag &= ~CSTOPB;

tty.c\_cflag &= ~CRTSCTS;

tty.c\_iflag = IGNPAR;

tty.c\_lflag = 0;

tty.c\_cc[VMIN] = 1;

tty.c\_cc[VTIME] = 1;

if (tcsetattr(fd, TCSANOW, &tty) != 0) {

printf("Error from tcsetattr: %s\n", strerror(errno));

return -1;

}

return 0;

}

void receive\_data(int fd) {

unsigned char buf[256];

while (1) {

int rdlen = read(fd, buf, sizeof(buf));

if (rdlen > 0) {

printf("Received: %.\*s\n", rdlen, buf);

} else if (rdlen < 0) {

printf("Error reading from serial port\n");

}

}

}

**receive\_main.c:**

#include "receive\_uart.h"

int main() {

char \*portname = "/dev/ttyS3";

int fd;

fd = open(portname, O\_RDWR | O\_NOCTTY | O\_SYNC);

if (fd < 0) {

printf("Error opening %s: %s\n", portname, strerror(errno));

return -1;

}

set\_interface\_attribs(fd, B9600);

receive\_data(fd);

close(fd);

return 0;

}

**OUTPUT:**

|  |  |
| --- | --- |
| Ex.no: 5  Date: | A **Userspace program to write and read the content of the A5D2x using EEPROM** |

**AIM:**

To write a userspace program to write and read the content of the a5d2x using EEPROM

**ALGORITHM:**

STEP 1: Import necessary libraries for I2C communication, file operations, and error handlingand set the path to the EEPROM file.

STEP 2: Create a structure to represent EEPROM data.

STEP 3: Define a function to read data from the EEPROM file.

STEP 4: Attempt to open the EEPROM file for reading,If opened successfully, read data from the file into the provided structure.

STEP 5: Handle errors during reading.

STEP 6: Define a function to write data to the EEPROM file

STEP 7: Attempt to open the EEPROM file for writing and write data to it.

STEP 8: Start program execution.

STEP 9: Initialize EEPROM data, write it to the EEPROM, and clear the data structure.

STEP 10: Read data from the EEPROM and print it.

**PROGRAM:**

**Board\_EEPROM.h**

#ifndef BOARD\_EEPROM\_H

#define BOARD\_EEPROM\_H

typedef struct \_cBoardEEPROM {

char ModuleID[16];

char SerialNo[16];

char data[512];

} cBoardEEPROM;

int ReadBoardEEPROM(cBoardEEPROM\* boardEEPROM);

void WriteBoardEEPROM(cBoardEEPROM\* boardEEPROM);

#endif /\* BOARD\_EEPROM\_H \*/

#include "board\_eeprom.h"

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <string.h>

#include <errno.h>

static const char EEPROM\_FILE\_BOARD[] = "/sys/class/i2c-adapter/i2c-0/0-0050/eeprom";

int ReadBoardEEPROM(cBoardEEPROM\* boardEEPROM) {

int Fd = 0;

int ReadBytes = 0;

memset(boardEEPROM, 0, sizeof(cBoardEEPROM));

if ((Fd = open(EEPROM\_FILE\_BOARD, O\_RDONLY)) < 0) {

printf("EEPROM\_FILE\_BOARD open error: %s\n", strerror(errno));

return -1;

}

ReadBytes = read(Fd, (void\*)boardEEPROM, sizeof(cBoardEEPROM));

if (ReadBytes < 0) {

printf("EEPROM\_FILE\_SOM read error: %s\n", strerror(errno));

close(Fd);

return -1;

}

close(Fd);

return 0;

}

**Board EEPROM.c**

void WriteBoardEEPROM(cBoardEEPROM\* boardEEPROM) {

int Fd = 0;

int WriteBytes = 0;

if ((Fd = open(EEPROM\_FILE\_BOARD, O\_WRONLY)) < 0) {

printf("EEPROM\_FILE\_BOARD open error: %s\n", strerror(errno));

return;

}

WriteBytes = write(Fd, (void\*)boardEEPROM, sizeof(cBoardEEPROM));

if (WriteBytes < sizeof(cBoardEEPROM)) {

if (WriteBytes < 0)

printf("EEPROM\_FILE\_BOARD write error : %s\n", strerror(errno));

else

printf("EEPROM\_FILE\_BOARD write error : Partial write to EEPROM (%d)\n", WriteBytes);

}

close(Fd);

}

**main.c**

#include <stdio.h>

#include <string.h>

#include "board\_eeprom.h"

int main(int argc, char \*argv[]) {

int Result = 0;

cBoardEEPROM boardEEPROM, board;

strcpy(boardEEPROM.ModuleID, "RBEEPROM1234");

strcpy(boardEEPROM.SerialNo, "112233445566");

strcpy(boardEEPROM.data, "Phytec Embedded Pvt Ltd");

WriteBoardEEPROM(&boardEEPROM);

memset(&board, 0x00, sizeof(board));

Result = ReadBoardEEPROM(&board);

printf("[EEPROM content]=========================\n");

printf("Module ID : %s\n", board.ModuleID);

printf("Serial NO : %s\n", board.SerialNo);

printf("Data : %s\n", board.data);

return 0;

}

**OUTPUT:**

**RESULT:**

|  |  |
| --- | --- |
| Ex.no: 6  Date: | **Interface an Lcd to rugged board using**  **12c interface to display the Name & Regno** |

**AIM:**

To interface an lcd to rugged board using 12c interface to display the name & regnum.

**ALGORITHM:**

Step -1: Start the process.

Step -2: Power on the A5D2x board and ensure all necessary peripherals are connected, including the LCD module.

Step -3: Create three C files (.c) with the name main.c , LiqudCrystal\_PCF8574.c , i2c-dev.c.

Step-4: Cross compile all the three .c files nd name the file.

Step-5: Open new terminal and open minicom and login as the root user.

Step-6: Enter ifconfig eth0 and your board IP Address.

Step-7: Enter tftp -r filename -g board IP Address.

Step-8: Enable the file by usng chmod 777 filename.

Step-9: Execute the file and the entered name and register number is displayed in the LCD.

**PROGRAM:**

**i2c-dev.c :**

#include "i2c-dev.h"

**LiqudCrystal\_PCF8574.c :**

#include "LiquidCrystal\_PCF8574.h"

#include "stdint.h"

#include <math.h>

#include <signal.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <linux/types.h>

#include <sys/ioctl.h>

#include<string.h>

#include<fcntl.h>

#define SLAVE\_ADDRESS\_LCD (0x27) // change this according to ur setup

int i2c\_dev\_node;

int i2c\_init(void)

{

char i2c\_dev\_node\_path[] = "/dev/i2c-0";

int ret\_val = 0;

i2c\_dev\_node = open(i2c\_dev\_node\_path, O\_RDWR);

if (i2c\_dev\_node < 0)

{

perror("Unable to open device node.");

exit(1);

}

ret\_val = ioctl(i2c\_dev\_node,I2C\_SLAVE,SLAVE\_ADDRESS\_LCD);

if (ret\_val < 0)

{

perror("Could not set I2C\_SLAVE.");

exit(2);

}

}

void lcd\_write\_i2c(int i2c,\_\_u8 \*buffer, \_\_u8 length)

{

int ret\_val = 0;

ret\_val=i2c\_smbus\_write\_block\_data(i2c, 0x00,length, buffer);

if (ret\_val < 0)

{

perror("Could not write data");

exit(2);

}

}

void lcd\_send\_cmd (char cmd)

{

char data\_u, data\_l;

uint8\_t data\_t[4];

data\_u = (cmd&0xf0);

data\_l = ((cmd<<4)&0xf0);

data\_t[0] = data\_u|0x0C;

data\_t[1] = data\_u|0x08;

data\_t[2] = data\_l|0x0C;

data\_t[3] = data\_l|0x08;

lcd\_write\_i2c(i2c\_dev\_node,(\_\_u8 \*)data\_t,4);

}

void lcd\_send\_data (char data)

{

char data\_u, data\_l;

uint8\_t data\_t[4];

data\_u = (data&0xf0);

data\_l = ((data<<4)&0xf0);

data\_t[0] = data\_u|0x0D;

data\_t[1] = data\_u|0x09;

data\_t[2] = data\_l|0x0D;

data\_t[3] = data\_l|0x09;

lcd\_write\_i2c(i2c\_dev\_node,(\_\_u8 \*)data\_t,4);

}

void setCursor(int a, int b)

{

int i=0;

switch(b){

case 0:lcd\_send\_cmd(0x80);break;

case 1:lcd\_send\_cmd(0xC0);break;

case 2:lcd\_send\_cmd(0x94);break;

case 3:lcd\_send\_cmd(0xd4);break;}

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

lcd\_send\_cmd(0x14);

}

void lcd\_send\_string (char \*str)

{

while (\*str) lcd\_send\_data (\*str++);

}

void lcd\_init (void)

{

i2c\_init();

usleep(50000);

lcd\_send\_cmd (0x3);

usleep(5000);

lcd\_send\_cmd (0x3);

usleep(110);

lcd\_send\_cmd (0x3);

usleep(10000);

lcd\_send\_cmd (0x2);

usleep(10000);

lcd\_send\_cmd (0x28);

usleep(1000);

lcd\_send\_cmd (0x08);

usleep(1000);

lcd\_send\_cmd (0x01);

usleep(1000);

usleep(1000);

lcd\_send\_cmd (0x06);

usleep(1000);

lcd\_send\_cmd (0x0C);

}

void lcd\_clear (void)

{

#define LCD\_CLEARDISPLAY 0x01

lcd\_send\_cmd(LCD\_CLEARDISPLAY);

usleep(100000);

}

**main.c :**

#include "LiquidCrystal\_PCF8574.h"

#include "stdio.h"

#include <unistd.h>

int main(void)

{

lcd\_init();

setCursor(0,0);

lcd\_send\_string("Prog for I2C on");

setCursor(0,1);

lcd\_send\_string("STM32F411RE");

usleep(2000000);

lcd\_clear();

while(1)

{

setCursor(0,0);

lcd\_send\_string("DEVA");

setCursor(0,1);

lcd\_send\_string("SHIVA");

usleep(1000000);

}

}

**OUTPUT :**

|  |  |
| --- | --- |
| Ex.no: 7  Date: | **A User space program to test the SPI present in the Microbus connector of RBA5D2X** |

**AIM:**

To write a user space program to test the SPI present in the micro bus connector.

**ALGORITHM:**

STEP 1: Start the program.

STEP 2: Include all necessary for SPI communication and other standard functionalities.

STEP 3: Macro “ARRAY\_SIZE” is defined to calculate the number of element array.

STEP 4: “pabort” function is defined for error handling. It prints an error message and aborts the program.

STEP 5: Several global variables including the SPI device path, SPI mode, bits per word, speed, delay and file paths for input and output.

STEP 6: Default transmit(“default\_tx”) and receive(“default\_rx”)buffers are defined.

STEP 7: “hex\_dump”,”unescape” and “transfer” helper functions are defined.

STEP 8: It parses command-line options, opens the SPI device, sets SPI mode, bits per word, and max speed. It then performs SPI transfer based on the selected options (input\_tx, input\_file, or default transmit buffer). Finally, it closes the SPI device and returns.

STEP 9: Stop the program.

**PROGRAM:**

**SPI.C**

#include "spi.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <getopt.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/ioctl.h>

#include <linux/ioctl.h>

#include <sys/stat.h>

#define ARRAY\_SIZE(a) (sizeof(a) / sizeof((a)[0]))

void pabort(const char \*s) {

perror(s);

abort();

}

void hex\_dump(const void \*src, size\_t length, size\_t line\_size, char \*prefix) {

int i = 0;

const unsigned char \*address = src;

const unsigned char \*line = address;

unsigned char c;

printf("%s | ", prefix);

while (length-- > 0) {

printf("%02X ", \*address++);

if (!(++i % line\_size) || (length == 0 && i % line\_size)) {

if (length == 0) {

while (i++ % line\_size)

printf("\_\_ ");

}

printf(" | "); /\* right close \*/

while (line < address) {

c = \*line++;

printf("%c", (c < 33 || c == 255) ? 0x2E : c);

}

printf("\n");

if (length > 0)

printf("%s | ", prefix);

}

}

}

int unescape(char \*\_dst, char \*\_src, size\_t len) {

int ret = 0;

int match;

char \*src = \_src;

char \*dst = \_dst;

unsigned int ch;

while (\*src) {

if (\*src == '\\' && \*(src+1) == 'x') {

match = sscanf(src + 2, "%2x", &ch);

if (!match)

pabort("malformed input string");

src += 4;

\*dst++ = (unsigned char)ch;

} else {

\*dst++ = \*src++;

}

ret++;

}

return ret;

}

void transfer(int fd, uint8\_t const \*tx, uint8\_t const \*rx, size\_t len) {

int ret;

int out\_fd;

struct spi\_ioc\_transfer tr = {

.tx\_buf = (unsigned long)tx,

.rx\_buf = (unsigned long)rx,

.len = len,

.delay\_usecs = delay,

.speed\_hz = speed,

.bits\_per\_word = bits,

};

if (mode & SPI\_TX\_QUAD)

tr.tx\_nbits = 4;

else if (mode & SPI\_TX\_DUAL)

tr.tx\_nbits = 2;

if (mode & SPI\_RX\_QUAD)

tr.rx\_nbits = 4;

else if (mode & SPI\_RX\_DUAL)

tr.rx\_nbits = 2;

if (!(mode & SPI\_LOOP)) {

if (mode & (SPI\_TX\_QUAD | SPI\_TX\_DUAL))

tr.rx\_buf = 0;

else if (mode & (SPI\_RX\_QUAD | SPI\_RX\_DUAL))

tr.tx\_buf = 0;

}

ret = ioctl(fd, SPI\_IOC\_MESSAGE(1), &tr);

if (ret < 1)

pabort("can't send spi message");

if (verbose)

hex\_dump(tx, len, 32, "TX");

if (output\_file) {

out\_fd = open(output\_file, O\_WRONLY | O\_CREAT | O\_TRUNC, 0666);

if (out\_fd < 0)

pabort("could not open output file");

ret = write(out\_fd, rx, len);

if (ret != len)

pabort("not all bytes written to output file");

close(out\_fd);

}

if (verbose || !output\_file)

hex\_dump(rx, len, 32, "RX");

}

void parse\_opts(int argc, char \*argv[]) {

while (1) {

static const struct option lopts[] = {

{ "device", 1, 0, 'D' },

{ "speed", 1, 0, 's' },

{ "delay", 1, 0, 'd' },

{ "bpw", 1, 0, 'b' },

{ "input", 1, 0, 'i' },

{ "output", 1, 0, 'o' },

{ "loop", 0, 0, 'l' },

{ "cpha", 0, 0, 'H' },

{ "cpol", 0, 0, 'O' },

{ "lsb", 0, 0, 'L' },

{ "cs-high", 0, 0, 'C' },

{ "3wire", 0, 0, '3' },

{ "no-cs", 0, 0, 'N' },

{ "ready", 0, 0, 'R' },

{ "dual", 0, 0, '2' },

{ "verbose", 0, 0, 'v' },

{ "quad", 0, 0, '4' },

{ NULL, 0, 0, 0 },

};

int c;

c = getopt\_long(argc, argv, "D:s:d:b:i:o:lHOLC3NR24p:v",

lopts, NULL);

if (c == -1)

break;

switch (c) {

case 'D':

device = optarg;

break;

case 's':

speed = atoi(optarg);

break;

case 'd':

delay = atoi(optarg);

break;

case 'b':

bits = atoi(optarg);

break;

case 'i':

input\_file = optarg;

break;

case 'o':

output\_file = optarg;

break;

case 'l':

mode |= SPI\_LOOP;

break;

case 'H':

mode |= SPI\_CPHA;

break;

case 'O':

mode |= SPI\_CPOL;

break;

case 'L':

mode |= SPI\_LSB\_FIRST;

break;

case 'C':

mode |= SPI\_CS\_HIGH;

break;

case '3':

mode |= SPI\_3WIRE;

break;

case 'N':

mode |= SPI\_NO\_CS;

break;

case 'v':

verbose = 1;

break;

case 'R':

mode |= SPI\_READY;

break;

case 'p':

input\_tx = optarg;

break;

case '2':

mode |= SPI\_TX\_DUAL;

break;

case '4':

mode |= SPI\_TX\_QUAD;

break;

default:

print\_usage(argv[0]);

break;

}

}

if (mode & SPI\_LOOP) {

if (mode & SPI\_TX\_DUAL)

mode |= SPI\_RX\_DUAL;

if (mode & SPI\_TX\_QUAD)

mode |= SPI\_RX\_QUAD;

}

}

void print\_usage(const char \*prog) {

printf("Usage: %s [-DsbdlHOLC3]\n", prog);

puts(" -D --device device to use (default /dev/spidev1.1)\n"

" -s --speed max speed (Hz)\n"

" -d --delay delay (usec)\n"

" -b --bpw bits per word\n"

" -i --input input data from a file (e.g. \"test.bin\")\n"

" -o --output output data to a file (e.g. \"results.bin\")\n"

" -l --loop loopback\n"

" -H --cpha clock phase\n"

" -O --cpol clock polarity\n"

" -L --lsb least significant bit first\n"

" -C --cs-high chip select active high\n"

" -3 --3wire SI/SO signals shared\n"

" -v --verbose Verbose (show tx buffer)\n"

" -p Send data (e.g. \"1234\\xde\\xad\")\n"

" -N --no-cs no chip select\n"

" -R --ready slave pulls low to pause\n"

" -2 --dual dual transfer\n"

" -4 --quad quad transfer\n");

exit(1);

}

void transfer\_escaped\_string(int fd, char \*str) {

size\_t size = strlen(str);

uint8\_t \*tx;

uint8\_t \*rx;

tx = malloc(size);

if (!tx)

pabort("can't allocate tx buffer");

rx = malloc(size);

if (!rx)

pabort("can't allocate rx buffer");

size = unescape((char \*)tx, str, size);

transfer(fd, tx, rx, size);

free(rx);

free(tx);

}

void transfer\_file(int fd, char \*filename) {

ssize\_t bytes;

struct stat sb;

int tx\_fd;

uint8\_t \*tx;

uint8\_t \*rx;

if (stat(filename, &sb) == -1)

pabort("can't stat input file");

tx\_fd = open(filename, O\_RDONLY);

if (fd < 0)

pabort("can't open input file");

tx = malloc(sb.st\_size);

if (!tx)

pabort("can't allocate tx buffer");

rx = malloc(sb.st\_size);

if (!rx)

pabort("can't allocate rx buffer");

bytes = read(tx\_fd, tx, sb.st\_size);

if (bytes != sb.st\_size)

pabort("failed to read input file");

transfer(fd, tx, rx, sb.st\_size);

free(rx);

free(tx);

 close(tx\_fd);

}

**SPI.h :**

#ifndef SPI\_H

#define SPI\_H

#include <stdint.h>

#define ARRAY\_SIZE(a) (sizeof(a) / sizeof((a)[0]))

void pabort(const char \*s);

void hex\_dump(const void \*src, size\_t length, size\_t line\_size, char \*prefix);

int unescape(char \*\_dst, char \*\_src, size\_t len);

void transfer(int fd, uint8\_t const \*tx, uint8\_t const \*rx, size\_t len);

void parse\_opts(int argc, char \*argv[]);

void transfer\_escaped\_string(int fd, char \*str);

void transfer\_file(int fd, char \*filename);

#endif /\* SPI\_H \*/

**SPI\_main.c :**

#include "spi.h"

#include <stdio.h>

static const char \*device = "/dev/spidev3.0";

static uint32\_t mode;

static uint8\_t bits = 8;

static char \*input\_file;

static char \*output\_file;

static uint32\_t speed = 500000;

static uint16\_t delay;

static int verbose;

uint8\_t default\_tx[] = {

0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,

0x40, 0x00, 0x00, 0x00, 0x00, 0x95,

0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,

0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,

0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,

0xF0, 0x0D,

};

uint8\_t default\_rx[ARRAY\_SIZE(default\_tx)] = {0, };

char \*input\_tx;

int main(int argc, char \*argv[]) {

int ret = 0;

int fd;

parse\_opts(argc, argv);

fd = open(device, O\_RDWR);

if (fd < 0)

pabort("can't open device");

ret = ioctl(fd, SPI\_IOC\_WR\_MODE32, &mode);

if (ret == -1)

pabort("can't set spi mode");

ret = ioctl(fd, SPI\_IOC\_RD\_MODE32, &mode);

if (ret == -1)

pabort("can't get spi mode");

ret = ioctl(fd, SPI\_IOC\_WR\_BITS\_PER\_WORD, &bits);

if (ret == -1)

pabort("can't set bits per word");

ret = ioctl(fd, SPI\_IOC\_RD\_BITS\_PER\_WORD, &bits);

if (ret == -1)

pabort("can't get bits per word");

/\*

\* max speed hz

\*/

ret = ioctl(fd, SPI\_IOC\_WR\_MAX\_SPEED\_HZ, &speed);

if (ret == -1)

pabort("can't set max speed hz");

ret = ioctl(fd, SPI\_IOC\_RD\_MAX\_SPEED\_HZ, &speed);

if (ret == -1)

pabort("can't get max speed hz");

printf("spi mode: 0x%x\n", mode);

printf("bits per word: %d\n", bits);

printf("max speed: %d Hz (%d KHz)\n", speed, speed/1000);

if (input\_tx && input\_file)

pabort("only one of -p and --input may be selected");

if (input\_tx)

transfer\_escaped\_string(fd, input\_tx);

else if (input\_file)

transfer\_file(fd, input\_file);

else

transfer(fd, default\_tx, default\_rx, sizeof(default\_tx));

close(fd);

   return ret;

}

|  |  |
| --- | --- |
| Ex.no: 8  Date: | **ANALOG TO DIGITAL CONVERTOR** |

**AIM:**

This application is to read the Analog data using ADC interface via /sys/fs entries.

**ALGORITHM:**

STEP 1: Start the process.

STEP 2: Open a terminal or command prompt. Navigate to the directory where you saved

STEP 3: Compile the program using a C compiler such as GCC.

STEP 4: Convert into tftp file in copy their file and using ethernet cable to use target terminal to given the commands to run it .

STEP 5: After successful compilation, run the program by executing the following command

./adc\_test <ADC\_CHANNEL\_NUMBER>

STEP 6: For example, to read from ADC channel 0, you would run

./adc\_test 0

STEP 7: Stop the program.

**PROGRAM:**

**Test.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include <unistd.h>

#include <fcntl.h>

#include "adc.h"

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

MAIN

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

int main(int argc, char \*\*argv, char \*\*envp)

{

int digital\_val;

struct adc\_config\_t adc\_desc;

float analog\_val;

adc\_desc.adc\_mode = ONESHOT;

adc\_desc.capture = 0;

adc\_desc.buff\_length = 100;

adc\_desc.adc\_channel = atoi(argv[1]);

printf("\n \*\*\* ADC TESTING UTILITY \*\*\* \n");

adc\_init(&adc\_desc);

adc\_set\_channel(&adc\_desc, adc\_desc.adc\_channel);

while (1)

{

digital\_val = adc\_output(&adc\_desc, adc\_desc.adc\_channel);

analog\_val = (digital\_val \* 0.805664062)/4095;

printf("Digital Value on Channel %d = %d\n", adc\_desc.adc\_channel, digital\_val);

printf("Analog Value on Channel %d = %f\n", adc\_desc.adc\_channel, analog\_val);

sleep(1);

}

return 0;

}

**ADC.h**

#ifndef \_ADC\_H

#define \_ADC\_H

#define ONESHOT "oneshot"

#define CONTINUOUS "continuous"

struct adc\_config\_t

char \*adc\_mode;

int capture;

unsigned int buff\_length;

int adc\_channel;

unsigned char buf[100];

};

int adc\_init(struct adc\_config\_t \*);

int adc\_set\_channel(struct adc\_config\_t \*, int);

int adc\_output(struct adc\_config\_t \*, int);

#endif

**ADC.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include <unistd.h>

#include <fcntl.h>

#include "adc.h"

int adc\_init(struct adc\_config\_t \*adc\_desc)

{

int mode\_fd;

int capture\_fd;

capture\_fd = open("/sys/bus/iio/devices/iio:device0/buffer/enable", O\_WRONLY);

if(capture\_fd < 0 )

{

perror("open buffer enable error");

}

if(adc\_desc->capture)

write(capture\_fd, "1", 2);

else

write(capture\_fd, "0", 2);

close(capture\_fd);

int length\_fd;

if(length\_fd < 0 )

{

perror("open buffer length error");

}

snprintf(adc\_desc->buf, sizeof(adc\_desc->buff\_length), "%d", adc\_desc->buff\_length);

write(length\_fd, adc\_desc->buf, strlen(adc\_desc->buf));

close(length\_fd);

return 0;

}

int adc\_set\_channel(struct adc\_config\_t \*adc\_desc, int channel)

{

int channel\_fd;

sprintf(adc\_desc->buf, "/sys/bus/iio/devices/iio:device0/scan\_elements/in\_voltage%d\_en", channel);

channel\_fd = open(adc\_desc->buf, O\_WRONLY);

if(channel\_fd < 0 )

{

perror("open scan\_elements/in\_voltage6\_en error");

}

write(channel\_fd, "1", 2);

close(channel\_fd);

return 0;

}

int adc\_output(struct adc\_config\_t \*adc\_desc, int channel)

{

int output\_fd, adc\_data;

char result[10], ret=0;

sprintf(adc\_desc->buf, "/sys/bus/iio/devices/iio:device0/in\_voltage%d\_raw", channel);

output\_fd = open(adc\_desc->buf, O\_RDONLY);

if(output\_fd < 0 )

{

perror("open in\_voltage6\_raw error");

}

ret = read(output\_fd, result, sizeof(result));

adc\_data = atoi(result);

close(output\_fd);

return adc\_data;

}

**OUTPUT:**

|  |  |
| --- | --- |
| Ex.no: 9  Date: | **Control the intensity of an LED using PWM.** |

**AIM:**

To control the intensity of an LED using PWM.

**ALGORITHM:**

STEP 1 : Start the process.

STEP 2: Include standard C libraries like stdio.h, stdlib.h, string.h, fcntl.h, unistd.h, and time.h.

STEP 3: Declare Global Variables like export\_path, pwm\_periods, pwm\_enable, duty\_cycle: paths to the PWM sysfs files.

STEP 4: Takes command-line arguments for the period and duty cycle.

STEP 5: Open and write to the export\_path to export the PWM pin.

STEP 6: Open and write the period value (from command-line argument) to pwm\_periods.

STEP 7: Set Duty Cycle and Generate PWM Signal

STEP 8: Close all file descriptors after use.

STEP 9: Stop the process.

RTC\_CR\_BYPSHAD.

STEP 9: Clear Initialization Flag

- Clear initialization flag (RTC\_ISR\_INIT) to indicate end of RTC initialization.

STEP 10: Disable Access to RTC Registers

- Disable access to RTC registers to prevent unintended writes by clearing PWR\_CR\_DBP.

**PROGRAM:**

**PWM\_controller.h:**

#ifndef PWM\_CONTROLLER\_H

#define PWM\_CONTROLLER\_H

void export\_pwm\_pin();

void set\_pwm\_period(int period);

void enable\_pwm();

void generate\_pwm\_signal(int period, int initial\_duty\_cycle);

#endif

**PWM\_controller.c:**

#include "pwm\_controller.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <fcntl.h>

#include <unistd.h>

char export\_path[] = "/sys/class/pwm/pwmchip0/export";

char pwm\_periods[] = "/sys/class/pwm/pwmchip0/pwm1/period";

char pwm\_enable[] = "/sys/class/pwm/pwmchip0/pwm1/enable";

char duty\_cycle[] = "/sys/class/pwm/pwmchip0/pwm1/duty\_cycle";

void export\_pwm\_pin() {

int fd = open(export\_path, O\_WRONLY);

if (fd < 0) {

perror("export path error \n");

exit(1);

}

write(fd, "1", 2);

close(fd);

}

void set\_pwm\_period(int period) {

int fd = open(pwm\_periods, O\_WRONLY);

if (fd < 0) {

perror("period error \n");

exit(1);

}

char period\_str[5];

snprintf(period\_str, sizeof(period\_str), "%d", period);

write(fd, period\_str, strlen(period\_str));

close(fd);

}

void enable\_pwm() {

int fd = open(pwm\_enable, O\_WRONLY);

if (fd < 0) {

perror("enable error \n");

exit(1);

}

write(fd, "1", 2);

close(fd);

}

void generate\_pwm\_signal(int period, int initial\_duty\_cycle) {

int fd = open(duty\_cycle, O\_WRONLY);

if (fd < 0) {

perror("duty cycle error \n");

exit(1);

}

int value = initial\_duty\_cycle;

char value\_str[5];

while (1) {

snprintf(value\_str, sizeof(value\_str), "%d", value);

write(fd, value\_str, strlen(value\_str));

printf("pwm value : %d \n", value);

value++;

usleep(100000);

if (value >= period) {

value = 0;

}

}

close(fd);

}

**Main.c:**

#include "pwm\_controller.h"

int main(int argc, char \*\*argv) {

if (argc != 3) {

printf("Usage: %s <period> <initial\_duty\_cycle>\n", argv[0]);

return 1;

}

int period = atoi(argv[1]);

int initial\_duty\_cycle = atoi(argv[2]);

export\_pwm\_pin();

set\_pwm\_period(period);

enable\_pwm();

generate\_pwm\_signal(period, initial\_duty\_cycle);

return 0;

}

**OUTPUT:**

|  |  |
| --- | --- |
| Ex.no: 10  Date: | **A User space program for echo server**  **and test it in Rugged Board** |

**AIM :**

To write a user space program for echo server and test in RBA5T2X.

**ALGORITHM:**

**Echoserver:**

Step 1: Start the Program.

Step 2: Create a socket for the server (server\_sockfd).

Step 3: Associate the server socket with a specific IP address and port (bind()).

Step 4: Mark the server socket as passive, indicating it will be used to accept incoming connections (listen()).

Step 5: Block until a client connection is established, then create a new socket (client\_sockfd) for communication with the client (accept()).

Step 6: Continuously read data from the client and write it back to the client.

Step 7: Close the socket connection with the client.

Step 8: Go back to step 4 to accept and handle new client connections indefinitely.

Step 9: Stop the Program.

**Echoclient:**

Step 1: Start the program

Step 2: Check if the correct number of command-line arguments is provided. If not, print an error message and exit.

Step 3: Create a socket

Step 4: Set up the server’s address and port in a ‘sockaddr\_in’ structure.

Step 5: Connect the socket to the server.

Step 6: Enter a loop:

\* Read input from stdin.

\*Send the input to the server.

\*Read the response from the server.

\*Print the response to stdout.

Step 7: Close the socket and exit the program.

Step 8: Stop the program.

**PROGRAM :**

**EchoServer:**

**server.h**

#ifndef SERVER\_H

#define SERVER\_H

#include <sys/types.h>

#include <sys/socket.h>

#include <netdb.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

int create\_server\_socket();

void bind\_server\_socket(int server\_sockfd, int port);

void listen\_for\_connections(int server\_sockfd);

int accept\_client\_connection(int server\_sockfd, struct sockaddr\_in \*client\_addr);

void handle\_client\_connection(int client\_sockfd);

#endif // SERVER\_H

**server.c**

#include "server.h"

int create\_server\_socket() {

return socket(AF\_INET, SOCK\_STREAM, 0);

}

void bind\_server\_socket(int server\_sockfd, int port) {

struct sockaddr\_in server\_addr;

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(port);

inet\_aton("130.0.12.164", &server\_addr.sin\_addr);

memset(&(server\_addr.sin\_zero), '\0', sizeof server\_addr.sin\_zero);

bind(server\_sockfd, (struct sockaddr\*)&server\_addr, sizeof server\_addr);

}

void listen\_for\_connections(int server\_sockfd) {

listen(server\_sockfd, 10);

}

int accept\_client\_connection(int server\_sockfd, struct sockaddr\_in \*client\_addr) {

int client\_sockfd;

int client\_len = sizeof \*client\_addr;

client\_sockfd = accept(server\_sockfd, (struct sockaddr\*)client\_addr, &client\_len);

return client\_sockfd;

}

void handle\_client\_connection(int client\_sockfd) {

char buf[100];

int rv;

while ((rv = read(client\_sockfd, buf, sizeof buf)) > 0)

write(client\_sockfd, buf, rv);

close(client\_sockfd);

}

**main.c**

#include "server.h"

int main() {

int server\_sockfd;

int client\_sockfd;

struct sockaddr\_in client\_addr;

server\_sockfd = create\_server\_socket();

bind\_server\_socket(server\_sockfd, 54154);

listen\_for\_connections(server\_sockfd);

while (1) {

fprintf(stderr, "\nServer waiting for client connection...");

client\_sockfd = accept\_client\_connection(server\_sockfd, &client\_addr);

fprintf(stderr, "\n\*\*\*\*\*\*\*\*\* CLIENT CONNECTION ESTABLISHED \*\*\*\*\*\*\*\*");

handle\_client\_connection(client\_sockfd);

fprintf(stderr, "\n\*\*\*\*\*\*\*\*\* CLIENT CONNECTION TERMINATED \*\*\*\*\*\*\*\*");

}

return 0;

}

**Echoclient:**

**Main.c**

#include “client.h”

Int main(int argc, char\*\* argv) {

If (argc != 3) {

Printf(“Must enter IP and port\n”);

Exit(1);

}

Int sockfd = create\_socket(argv[1], argv[2]);

Client\_loop(sockfd);

Close(sockfd);

Exit(0);

}

**Client.h**

#ifndef CLIENT\_H

#define CLIENT\_H

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netdb.h>

#include <netinet/in.h>

#include <arpa/inet.h>

Int create\_socket(const char \*ip, const char \*port);

Void client\_loop(int sockfd);

#endif

**Client.c**

#include “client.h”

Int create\_socket(const char \*ip, const char \*port) {

Int sockfd = socket(PF\_INET, SOCK\_STREAM, 0);

If (sockfd == -1) {

Perror(“socket”);

Exit(1);

}

Struct sockaddr\_in dest\_addr;

Dest\_addr.sin\_family = AF\_INET;

Dest\_addr.sin\_port = htons(atoi(port));

Inet\_aton(ip, &dest\_addr.sin\_addr);

Memset(&(dest\_addr.sin\_zero), ‘\0’, sizeof dest\_addr.sin\_zero);

If (connect(sockfd, (struct sockaddr\*)&dest\_addr, sizeof dest\_addr) == -1) {

Perror(“connect”);

Exit(1);

}

Return sockfd;

}

Void client\_loop(int sockfd) {

Char buff1[128], buff2[128];

While (1) {

Int n = read(0, buff1, sizeof buff1);

Buff1[n] = ‘\0’;

Write(sockfd, buff1, strlen(buff1));

N = read(sockfd, buff2, sizeof buff2);

Buff2[n] = ‘\0’;

Write(1, buff2, n);

}

}

**OUTPUT:**