```
/* main.c
* Author: Nachiket Kelkar and Puneet Bansal
* Reference: The project is based on the freeRTOS example code on
* https://github.com/akobyl/TM4C129_FreeRTOS_Demo/blob/master/main.c
*/

#include
#include
#include
#include "main.h"
#include "drivers/pinout.h"
#include "utils/uartstdio.h"

// TivaWare includes
#include "driverlib/sysctl.h"
#include "driverlib/debug.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "inc/hw_memmap.h"

// FreeRTOS includes
#include "FreeRTOSConfig.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "timers.h"
#include "pwm.h"

#include "src/spi.h"
#include "src/LCDdriver.h"
#include "driverlib/gpio.h"
#include "driverlib/adc.h"
#include "src/Logger.h"
#include "src/sensor.h"

/*Additions*/
#include "src/actuator.h"

#include "driverlib/pwm.h"
```

```
#include "driverlib/pin_map.h"

// Demo Task declarations
void demoLEDTask(void *pvParameters);
void demoSerialTask(void *pvParameters);

uint32_t g_ui32SysClock;

void motor_control_init();
void motor_control_config(uint32_t period_in_khz, uint8_t duty_cycle);
void TXFF_interrupt();
void TestCallback();

TaskHandle_t TempTaskHandle;
TaskHandle_t SMTaskHandle;
TaskHandle_t IBTaskHandle;
TaskHandle_t LCDTaskHandle;
TaskHandle_t MotorTaskHandle;
TaskHandle_t FanTaskHandle;


// Main function
int main(void)
{
// Initialize system clock to 120 MHz
g_ui32SysClock = ROM_SysCtlClockFreqSet(
(SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN |
SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480),
SYSTEM_CLOCK);

Logger_Init();
UARTStdioConfig(0, 115200, g_ui32SysClock);

UARTprintf("Creating tasks\n");

// Creating all the required task
xTaskCreate(TemperatureTask, "Temperature", 256, NULL, 1,
&TempTaskHandle);
xTaskCreate(SoilMoistureTask, "Moisture", 256, NULL, 1, &SMTaskHandle);
xTaskCreate(InterBoardSPI, "InterBoardCom", 256, NULL, 1, &IBTaskHandle);
xTaskCreate(LCDTask, "LCDTask", 256, NULL, 1, &LCDTaskHandle);
xTaskCreate(MotorTask, "MotorTask", 256, NULL, 1, &MotorTaskHandle);
xTaskCreate(FanTask, "FanTask", 256, NULL, 1, &FanTaskHandle);


vTaskStartScheduler();
UARTprintf("I should not have come here\n");
}

/* ASSERT() Error function
*
* failed ASSERTS() from driverlib/debug.h are executed in this function
*/
void __error__(char *pcFilename, uint32_t ui32Line)
{
// Place a breakpoint here to capture errors until logging routine is finished
while (1)
{
}
}


/*
* main.h
*
* Created on: April 21, 2019
* Author: Nachiket Kelkar and Puneet Bansal
*/

#ifndef MAIN_H_
#define MAIN_H_

// System clock rate, 120 MHz
#define SYSTEM_CLOCK 120000000U


#endif /* MAIN_H_ */
```

```
/*
* actuator.c
*
* Created on: Apr 22, 2019
* Author: puneet bansal and Nachiket Kelkar
*/

#include
#include
#include "driverlib/sysctl.h"
#include "driverlib/debug.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "inc/hw_memmap.h"
#include "utils/uartstdio.h"
#include "../FreeRTOS/include/projdefs.h"

#include "spi.h"
#include "driverlib/gpio.h"
#include "driverlib/adc.h"
#include "driverlib/pin_map.h"
#include "driverlib/ssi.h"

// FreeRTOS includes
#include "FreeRTOSConfig.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "timers.h"
#include "actuator.h"
#include "LCDdriver.h"

QueueHandle_t LCDQueue;

void LCDTask(void *pvParameters)
{
UARTprintf("Entered LCD Task");
LCDStruct dataReceived;
```

```
lcd_init();
lcd_on();
lcd_pos(0, 0);
lcd_write_string("Temp");
lcd_pos(0, 8);
lcd_write_string("Moist");
lcd_pos(2, 0);
lcd_write_string("Fan");
lcd_pos(2, 8);
lcd_write_string("Motor");
LCDQueue=xQueueCreate(10, sizeof(LCDStruct));
float val=0;

while(1)
{
if(xQueueReceive(LCDQueue, &dataReceived, portMAX_DELAY))
{
switch(dataReceived.source)
{
case 0x55:
if(dataReceived.task == 1)
{
lcd_pos(1, 0);
lcd_write_string(" ");
lcd_pos(1, 0);
val=dataReceived.sensing_data*0.25;
if(val != 0)
{
lcd_print_float(val);
}
else
{
lcd_write_string("SEN NC");
}
}
else
{
lcd_pos(3, 0);
lcd_write_string(" ");
lcd_pos(3, 0);
lcd_print_digit((long)dataReceived.actuation_data);
}
break;
case 0xaa:
if(dataReceived.task == 1)
{
lcd_pos(1, 8);
lcd_write_string(" ");
val=dataReceived.sensing_data;
lcd_pos(1, 8);
if(val != 0)
{
lcd_print_float(val);
}
else
{
lcd_write_string("SEN NC");
}
}
else
{
lcd_pos(3, 8);
lcd_write_string(" ");
lcd_pos(3, 8);
lcd_print_digit(dataReceived.actuation_data);
break;
}
}
}
}

}

void FanTask(void *pvParameters)
{
UARTprintf("Entered Fan Task");
uint32_t notificationVal;
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
```

```
GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_0);

while(1)
{
xTaskNotifyWait(0x00, 0xffffffff , ¬ificationVal , portMAX_DELAY);
switch(notificationVal)
{
case 0: UARTprintf("Turn off the fan");
GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_0, 0);
break;

case 1: UARTprintf("Turn on the fan");
GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_0, GPIO_PIN_0);
break;
}
}

}

int duty_cycle = 0;

void MotorTask(void *pvParameters)
{
// UARTprintf("Entered Motor Task");
uint32_t notificationVal;
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);

GPIOPadConfigSet(GPIO_PORTG_BASE, GPIO_PIN_0,
GPIO_STRENGTH_4MA, GPIO_PIN_TYPE_STD);
GPIODirModeSet(GPIO_PORTG_BASE, GPIO_PIN_0,
GPIO_DIR_MODE_OUT);

/* Initialize the timer for periodic measurements */
TimerHandle_t MotorTimer = xTimerCreate("Motor", pdMS_TO_TICKS(1),
pdTRUE, (void*)0, MotorCallback);

/* Start the timer after 100ms */
BaseType_t return_val = xTimerStart(MotorTimer, pdMS_TO_TICKS(0));
```

```
while(1)
{
xTaskNotifyWait(0x00, 0xffffffff , ¬ificationVal , portMAX_DELAY);
UARTprintf("Print - %d",notificationVal);
duty_cycle = notificationVal;
}
}

void MotorCallback(TimerHandle_t xtimer)
{
static int x = 0;
if(x < duty_cycle)
{
GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_0, GPIO_PIN_0);
x ++;
}
else
{
GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_0, 0);
x ++;
}
if(x == 10)
{
x = 0;
}
}
```

```
/*
* actuator.h
*
* Created on: Apr 22, 2019
* Author: puneet bansal and Nachiket Kelkar
*/

#ifndef SRC_ACTUATOR_H_
#define SRC_ACTUATOR_H_
```

```
typedef struct
{
uint8_t source;
uint16_t sensing_data;
uint8_t actuation_data;
uint8_t task;
}LCDStruct;


/*
* Function name: LCDTask()
* Description: This function is the task to be exectued for LCD. It waits for the
* data from the sensor and then displays it on the LCD.
* @param: void
* @return: void
*/
void LCDTask(void *pvParameters);


/*
* Function name: FanTask()
* Description: This tasks waits for the control messages and switches ON the
* temperature controller fan or switches it ON based on the control
* message from controller node.
* @param: void
* @return: void
*/
void FanTask(void *pvParameters);


/*
* Function name: MotorTask()
* Description: This function waits for the message from control node. It controls
the
* PWM duty cycle based on the control message from controller node.
* @param: void
* @return: void
*/
void MotorTask(void *pvParameters);
```

```
/*
* Function name: MotorCallback()
* Description: This function is the timer call back to generate the PWM signal
* for controlling the motor.
* @param: void
* @return: void
*/
void MotorCallback();

#endif /* SRC_ACTUATOR_H_ */
```

```
/*
* LCDdriver.c
*
* Created on: Apr 15, 2019
* Author: nachiket kelkar & puneet bansal
*/

#include
#include
#include
#include
#include "driverlib/gpio.h"
#include "inc/hw_memmap.h"
#include "LCDdriver.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"

void lcd_init()
{
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOK);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOP);

while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOP))
{
}

/* Configure pins as output */
GPIOPinTypeGPIOOutput(GPIO_PORTK_BASE, GPIO_PIN_0 | GPIO_PIN_1
| GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 |
GPIO_PIN_7);
GPIOPinTypeGPIOOutput(GPIO_PORTP_BASE, GPIO_PIN_0 | GPIO_PIN_1
| GPIO_PIN_4);
}


void lcd_write_data(char data)
{
/* Write data on pins for LCD */
GPIOPinWrite(GPIO_PORTP_BASE, GPIO_PIN_0, GPIO_PIN_0);
GPIOPinWrite(GPIO_PORTP_BASE, GPIO_PIN_1, 0);

GPIOPinWrite(GPIO_PORTK_BASE, GPIO_PIN_7 | GPIO_PIN_6 |
GPIO_PIN_5 | GPIO_PIN_4 | GPIO_PIN_3 | GPIO_PIN_2 | GPIO_PIN_1 |
GPIO_PIN_0, (int)data);
latch_data();
}


void lcd_write_string(char* data)
{
int len,i = 0;
len = strlen(data);
while(i != len)
{
lcd_write_data(data[i++]);
}
}


void lcd_write_command(uint8_t command)
{
/* Write data on pins for LCD */
GPIOPinWrite(GPIO_PORTP_BASE, GPIO_PIN_0, 0);
GPIOPinWrite(GPIO_PORTP_BASE, GPIO_PIN_1, 0);

GPIOPinWrite(GPIO_PORTK_BASE, GPIO_PIN_7 | GPIO_PIN_6 |
GPIO_PIN_5 | GPIO_PIN_4 | GPIO_PIN_3 | GPIO_PIN_2 | GPIO_PIN_1 |
GPIO_PIN_0, command);
latch_data();
}


void latch_data()
{
/* Write data on pins for LCD */
GPIOPinWrite(GPIO_PORTP_BASE, GPIO_PIN_4, GPIO_PIN_4);
delay(1000);
GPIOPinWrite(GPIO_PORTP_BASE, GPIO_PIN_4, 0);
}


void lcd_pos(uint8_t row, uint8_t position)
{
uint8_t command;
switch(row)
{
case 0:
command = 0x80 | position;
break;
case 1:
command = 0x80 | (64+position);
break;
case 2:
command = 0x80 | (16+position);
break;
case 3:
command = 0x80 | (80+position);
}
lcd_write_command(command);
}
```

```
void delay(uint16_t x)
{
int i = 0;
for(i=0; i< x; i++);
for(i=0; i< x; i++);
for(i=0; i< x; i++);
for(i=0; i< x; i++);
}


void lcd_on()
{
lcd_write_command(0x30);
lcd_write_command(0x30);
lcd_write_command(0x30);

SYSTEM_SET;
DISPLAY_OFF;
DISPLAY_ON;
ENTRY_MODE;
CLEAR_DISPLAY;
delay(4000);
CURSOR_HOME;
delay(4000);
}

void lcd_print_digit(long no)
{
char buffer[10];
ltoa(no, buffer);
lcd_write_string(buffer);
}

void lcd_print_float(float no)
{
lcd_print_digit(no);
no = no - (long)no;
no = no * 1000;
lcd_write_data('.');
lcd_print_digit(no);
}


/*
* LCDdriver.h
*
* Created on: Apr 15, 2019
* Author: nachiket kelkar & puneet bansal
*/

#ifndef SRC_LCDDRIVER_H_
#define SRC_LCDDRIVER_H_

/*
* Function name: lcd_init()
* Description: This function initializes all the GPIO pins required by the LCD.
* @param: void
* @return: void
*/
void lcd_init();


/*
* Function name: lcd_write_data()
* Description: This function takes the data to write writes at the cursor position
* on LCD
* @param: char(character to be written)
* @return: void
*/
void lcd_write_data(char data);


/*
* Function name: lcd_write_command()
* Description: This function takes the command for LCD and executes the
command of LCD
* @param: uint8_t(command to be executed)
* @return: void
```

```
*/
void lcd_write_command(uint8_t command);


/*
* Function name: latch_data()
* Description: This function toggles the Enable pin of LCD so that data is
latched
* @param: void
* @return: void
*/
void latch_data();


/*
* Function name: lcd_pos()
* Description: This function takes the row and column to postion the cursor
* @param: uint8_t(row of LCD), uint8_t(column of LCD)
* @comment: The valid row values are: 0,1,2,3 and valid position values are
* from 0 to 15.
* @return: void
*/
void lcd_pos(uint8_t row, uint8_t position);


/*
* Function name: delay()
* Description: This function generates the required delay
* @param: uint16_t(delay value)
* @return: void
*/
void delay(uint16_t x);


/*
* Function name: lcd_on()
* Description: This function configures the LCD and places the cursor at the
home position
* @param: void
* @return: void
*/
void lcd_on();


/*
* Function name: lcd_write_string()
* Description: This function accepts the string and displays it on LCD
* @param: char* (string to be written to LCD)
* @return: void
*/
void lcd_write_string(char* data);


/*
* Function name: lcd_print_digit()
* Description: This function prints the numbers on thee display
* @param: long (number to be printed)
* @return: void
*/
void lcd_print_digit(long x);


/*
* Function name: lcd_print_float()
* Description: This function displays the float values on the screen
* @param: float (float value to be printed)
* @return: void
*/
void lcd_print_float(float no);

#define CLEAR_DISPLAY lcd_write_command(0x01)
#define CURSOR_HOME lcd_write_command(0x02)
#define ENTRY_MODE lcd_write_command(0x06)
#define DISPLAY_ON lcd_write_command(0x0C)
#define DISPLAY_OFF lcd_write_command(0x08)
#define DISPLAY_SHIFT lcd_write_command(0x10)
#define SYSTEM_SET lcd_write_command(0x38)
```

```
#endif /* SRC_LCDDRIVER_H_ */



/*
* Logger.c
*
* Created on: Apr 17, 2019
* Author: nachiket kelkar & puneet bansal
*/
#include
#include
#include
#include
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "inc/hw_memmap.h"
#include "driverlib/gpio.h"
#include "Logger.h"
#include "utils/uartstdio.h"
#include "projdefs.h"

void Logger_Init(void)
{
/* Enable UART pins */
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

/* Configure GPIO pins as UART */
GPIOPinConfigure(GPIO_PA0_U0RX);
GPIOPinConfigure(GPIO_PA1_U0TX);
ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 |
GPIO_PIN_1);
}
```

```
/*
* Logger.h
*
* Created on: Apr 17, 2019
* Author: nachiket kelkar & puneet bansal
*/

#ifndef SRC_LOGGER_H_
#define SRC_LOGGER_H_

/*
* Name: Logger_Init()
* Description: This function initializes the logger i.e the UART
* @param: void
* @return: void
*/
void Logger_Init(void);

#endif /* SRC_LOGGER_H_ */



/*
* sensor.c
*
* Created on: Apr 21, 2019
* Author: nachiket kelkar & puneet bansal
*/

#include
#include
#include "driverlib/sysctl.h"
#include "driverlib/debug.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "inc/hw_memmap.h"
#include "utils/uartstdio.h"
#include "../FreeRTOS/include/projdefs.h"
```

```
#include "sensor.h"
#include "spi.h"
#include "driverlib/gpio.h"
#include "driverlib/adc.h"
#include "driverlib/pin_map.h"
#include "driverlib/ssi.h"

// FreeRTOS includes
#include "FreeRTOSConfig.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "timers.h"
#include "actuator.h"

extern uint32_t g_ui32SysClock;
extern TaskHandle_t TempTaskHandle;
extern TaskHandle_t SMTaskHandle;
extern QueueHandle_t IBQueue;
extern QueueHandle_t LCDQueue;

uint16_t temp_data;

void moisture_sensor_init()
{
/* Enable ADC and gpio port for using moisture sensor */
SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);

/* Configure gpio pin as ADC */
GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_3);
ADCSequenceConfigure(ADC0_BASE, SEQUENCE_NO,
ADC_TRIGGER_PROCESSOR, 0);
ADCSequenceStepConfigure(ADC0_BASE, SEQUENCE_NO, 0,
ADC_CTL_CH0 | ADC_CTL_IE | ADC_CTL_END);
ADCSequenceEnable(ADC0_BASE, SEQUENCE_NO);
ADCIntClear(ADC0_BASE, SEQUENCE_NO);
}
```

```
uint32_t moisture_data()
{
uint32_t data;
int i;
ADCProcessorTrigger(ADC0_BASE, SEQUENCE_NO);
for(i=0; i<10000; i++);
ADCIntClear(ADC0_BASE, SEQUENCE_NO);
ADCSequenceDataGet(ADC0_BASE, SEQUENCE_NO, &data);
return data;
}


void temp_sens_init(uint32_t mode, uint32_t clk_speed)
{
SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI1);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOE));

/* Configure the GPIO pins for using it as SPI */
GPIOPinConfigure(GPIO_PE5_SSI1XDAT1);
GPIOPinConfigure(GPIO_PE4_SSI1XDAT0);
GPIOPinConfigure(GPIO_PB4_SSI1FSS);
GPIOPinConfigure(GPIO_PB5_SSI1CLK);


GPIOPinTypeSSI(GPIO_PORTB_BASE, GPIO_PIN_4 | GPIO_PIN_5);
GPIOPinTypeSSI(GPIO_PORTE_BASE, GPIO_PIN_4 | GPIO_PIN_5);

SSIConfigSetExpClk(SSI1_BASE, g_ui32SysClock,
SSI_FRF_MOTO_MODE_0, mode, clk_speed, 16);
SSIEnable(SSI1_BASE);
}


uint16_t temp_data_get()
{
uint32_t buffer;
/* junk value to start the SPI transaction */
```

```
uint16_t junk_val = 0x1234;
SSIDataPut(SSI1_BASE, junk_val);
SSIDataGet(SSI1_BASE, &buffer);
return (uint16_t)buffer;
}


void TemperatureTask(void *pvParameters)
{
IBStruct dataToSendToIB;
LCDStruct dataToSendToLCD;

dataToSendToIB.source = TEMP_SOURCE_ID;
dataToSendToLCD.source = TEMP_SOURCE_ID;
dataToSendToLCD.task = SENS_TASK_ID;

/* Initialize the temperature sensor */
temp_sens_init(MASTER, TEMP_SPI_CLK);

/* BIST */
temp_data = temp_data_get()>>3;
if(temp_data == 0)
{
UARTprintf("Temperature sensor BIST failed\n");
}

/* Initialize the timer for periodic measurements */
TimerHandle_t TakeTempReadings = xTimerCreate("TakeTemperature",
pdMS_TO_TICKS(2000), pdTRUE, (void*)0, TemperatureCallback);
/* Start the timer after 100ms */
BaseType_t return_val = xTimerStart(TakeTempReadings,
pdMS_TO_TICKS(0));
while(1)
{
/* Wait for notification from the timer to take reading from sensors */
xTaskNotifyWait(0x00, 0xffffffff, NULL, portMAX_DELAY);
// UARTprintf("Temp task notify reading\n");

/* Take the reading from the sensor */
// data_to_send.data = temp_data_get()>>3;

dataToSendToIB.data = temp_data;
dataToSendToLCD.sensing_data = temp_data;

/* Send it to the queue of the SPI task */
xQueueSend(IBQueue, &dataToSendToIB, pdMS_TO_TICKS(0));
xQueueSend(LCDQueue, &dataToSendToLCD, pdMS_TO_TICKS(0));
}
}


void TemperatureCallback(TimerHandle_t xtimer)
{
/* Notify the task to take the readings */
static int x = 0;
LCDStruct dataToSendToLCD;
dataToSendToLCD.source = TEMP_SOURCE_ID;
dataToSendToLCD.task = SENS_TASK_ID;

if(TempTaskHandle != NULL)
{
temp_data = temp_data_get()>>3;
if(temp_data != 0)
{
xTaskNotify(TempTaskHandle, 1, eSetBits);
x = 0;
}
else
{
if(x == 0)
{
dataToSendToLCD.sensing_data = 0;
UARTprintf("Temperature sensor disconnected\n");
x = 1;
xQueueSend(LCDQueue, &dataToSendToLCD, pdMS_TO_TICKS(0));
}
}
}
```

```
}


void SoilMoistureTask(void *pvParameters)
{
static int x = 0;
UARTprintf("Moist task\n");

IBStruct data_to_send;
LCDStruct dataTOSendTOLCD;

data_to_send.source = SM_SOURCE_ID;
dataTOSendTOLCD.source = SM_SOURCE_ID;
dataTOSendTOLCD.task = SENS_TASK_ID;

/* Initialize the soil moisture sensor ADC. */
moisture_sensor_init();
// Initialize the timer for periodic measurements */
TimerHandle_t TakeSoilReadings = xTimerCreate("TakeSoilMoisture",
pdMS_TO_TICKS(2000), pdTRUE, (void*)0, MoistureCallback);
/* Start the timer after 100ms */
BaseType_t return_val = xTimerStart(TakeSoilReadings,
pdMS_TO_TICKS(100));
if(return_val != pdPASS)
{
UARTprintf("Moisture timer failed\n");
}

/* BIST */
data_to_send.data = moisture_data();
if(data_to_send.data > 5)
{
UARTprintf("Soil moisture BIST failed");
}

while(1)
{
/* Wait for notification from the timer to take reading from sensors */
xTaskNotifyWait(0x00, 0xffffffff, NULL, portMAX_DELAY);
/* Take the reading from the sensor */
data_to_send.data = moisture_data();
if(data_to_send.data > 5)
{
dataTOSendTOLCD.sensing_data = data_to_send.data;

/* Send it to the queue of the SPI task */
xQueueSend(IBQueue, &data_to_send, pdMS_TO_TICKS(0));
xQueueSend(LCDQueue, &dataTOSendTOLCD, pdMS_TO_TICKS(0));
x = 0;
}
else
{
if(x == 0)
{
dataTOSendTOLCD.sensing_data = 0;
UARTprintf("Soil moisture sensor disconnected\n");
xQueueSend(LCDQueue, &dataTOSendTOLCD, pdMS_TO_TICKS(0));
x = 1;
}
}
}
}


void MoistureCallback(TimerHandle_t xtimer)
{
/* Notify the task to take the readings */
xTaskNotify(SMTaskHandle, 1, eSetBits);
}


float temperature_in_c(uint16_t hex_val)
{
hex_val = hex_val >> 3;
return (hex_val * TEMP_CONV_FACTOR);
}
```

```
/*
* sensor.h
*
* Created on: Apr 21, 2019
* Author: nachiket kelkar & puneet bansal
*/

#ifndef SRC_SENSOR_H_
#define SRC_SENSOR_H_

#define TEMP_SPI_CLK 500000
#define TEMP_SOURCE_ID 0x55
#define TEMP_CONV_FACTOR 0.25
#define SM_SOURCE_ID 0xAA
#define SEQUENCE_NO 3
#define SENS_TASK_ID 1


/*
* Function name: moisture_sensor_init()
* Description : This function initializes ADC to get the analog voltage from soil
moisture
* sensor and convert it to the digital value.
* @param : void
* @return : void
*/
void moisture_sensor_init();


/*
* Function name: moisture_data()
* Description : This function gets the digital from ADC module.
* @param : void
* @return : uint32_t(Digital converted data from ADC)
*/
uint32_t moisture_data();
```

```
/*
* Function name: temp_sens_init()
* Description : This function initializes the SSI3 module of the TIVA board
which is used for
* SPI communication with the temperature sensor.
* @param : uint32_t(mode for SPI), uint32_t(clock speed)
* @comments : The mode can be Master or Slave
* @return : void
*/
void temp_sens_init(uint32_t, uint32_t);


/*
* Function name: temp_data_get()
* Description: This function gets the data from temperature sensor data and
return it.
* @param: void
* @return: uint16_t (data received from temperature sensor register to Tiva)
*/
uint16_t temp_data_get();


/*
* Function name: TemperatureTask()
* Description : This function executed which contains the logic for temperature
sensor task.
* @param : void
* @return : void *
*/
void TemperatureTask(void *pvParameters);


/*
* Function name: TemperatureCallback()
* Description : The callback function executed when the timer elapses. The
temperature task
* is notified when this timer is expired.
* @param : void
* @return : void
*/
void TemperatureCallback();


/*
* Function name: SoilMoistureTask()
* Description : This function executed which contains the logic for soil moisture
sensor task.
* @param : void
* @return : void *
*/
void SoilMoistureTask(void *pvParameters);


/*
* Function name: MoistureCallback()
* Description : The callback function executed when the timer elapses. The
soilMoisture task
* is notified when this timer is expired.
* @param : void
* @return : void
*/
void MoistureCallback();


/*
* Function name: temperature_in_c()
* Description : This function takes the ADC values and converts it to the Celcius
values and
* returns the valid value.
* @param : void
* @return : void
*/
float temperature_in_c(uint16_t);

#endif /* SRC_SENSOR_H_ */
```

```
/*
* spi.c
*
* Created on: Apr 15, 2019
* Author: nachiket kelkar & puneet bansal
*/

#include
#include
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "inc/hw_memmap.h"
#include "driverlib/pin_map.h"
#include "utils/uartstdio.h"
#include "spi.h"

// FreeRTOS includes
#include "FreeRTOSConfig.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "timers.h"
#include "actuator.h"

extern uint32_t g_ui32SysClock;
QueueHandle_t IBQueue;
extern QueueHandle_t LCDQueue;
uint8_t trid = 0x00;
uint16_t packet;
extern TaskHandle_t IBTaskHandle;
extern TaskHandle_t FanTaskHandle;
extern TaskHandle_t MotorTaskHandle;


int prev_state = 0;

void spi_init(uint32_t mode, uint32_t clk_speed)
{
SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI2);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOD));

GPIOPinConfigure(GPIO_PD0_SSI2XDAT1);
GPIOPinConfigure(GPIO_PD1_SSI2XDAT0);
GPIOPinConfigure(GPIO_PD2_SSI2FSS);
GPIOPinConfigure(GPIO_PD3_SSI2CLK);

GPIOPinTypeSSI(GPIO_PORTD_BASE, GPIO_PIN_0 | GPIO_PIN_1 |
GPIO_PIN_2 | GPIO_PIN_3);

SSIConfigSetExpClk(SSI2_BASE, g_ui32SysClock,
SSI_FRF_MOTO_MODE_0, mode, clk_speed, 16);
SSIEnable(SSI2_BASE);
}

void spi_data_write(uint64_t data_to_write, uint8_t no_of_bytes)
{
SSIDataPutNonBlocking(SSI2_BASE, (uint16_t)data_to_write);
}

uint16_t spi_data_read()
{
uint32_t buffer;
SSIDataGet(SSI2_BASE, &buffer);
return (uint16_t)buffer;
}

void InterBoardSPI(void *pvParameters)
{
// UARTprintf("SPI task\n");
// SPI testing
// uint16_t received_data;
// uint16_t control_message;
// IBStruct rec_msg;
prev_state = 0;

//Initialize the queue
IBQueue = xQueueCreate( 15, sizeof(IBStruct));
```

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_1);

spi_init(SLAVE, 500000);

spi_data_write(0x0011, 1);
while(1)
{
spi_state_machine();
}
}


void decode_message(uint16_t ctrl_msg)
{
LCDStruct dataToSend;
uint8_t actual_msg= (ctrl_msg >> 8);
dataToSend.task = 2;
if((ctrl_msg & 0xff) == 0x55)
{
dataToSend.source=0x55;
dataToSend.actuation_data=actual_msg;
xQueueSend(LCDQueue, &dataToSend, pdMS_TO_TICKS(0));
xTaskNotify(FanTaskHandle,actual_msg, eSetValueWithoutOverwrite);
//Send the data to the fan actuator queue
}
else if((ctrl_msg & 0xff) == 0xaa)
{
dataToSend.source=0xaa;
dataToSend.actuation_data=actual_msg;
xQueueSend(LCDQueue, &dataToSend, pdMS_TO_TICKS(0));
xTaskNotify(MotorTaskHandle,actual_msg, eSetValueWithoutOverwrite);
//send the data to the motor actuator queue
}
}


void spi_state_machine()
```

```
{
uint32_t buffer;
IBStruct rec_msg;
int bytes_rec;

static uint8_t state = STATE_SEND_TRID;

switch(state)
{
case STATE_SEND_TRID:
UARTprintf("In state 1\n");
xQueueReceive(IBQueue, &rec_msg, portMAX_DELAY);
packet = ((uint16_t)++trid<<8) | rec_msg.source;
UARTprintf("source - packet - data is %x - %x -
%x\n",rec_msg.source,packet,rec_msg.data);
if(prev_state != state)
{
spi_data_write((uint16_t)packet, 1);
prev_state = 1;
}
bytes_rec = SSIDataGetNonBlocking(SSI2_BASE, &buffer);
UARTprintf("RX 1 - %x\n\r",buffer);
if(bytes_rec != 0 && buffer == 0x01)
{
state = STATE_SEND_DATA;
}
else if(buffer == 0x02)
{
state = STATE_SEND_DATA;
}
else
{
GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1, GPIO_PIN_1);
UARTprintf("Control node disconnected\n");
self_control(rec_msg);
}
break;
case STATE_SEND_DATA:
UARTprintf("In state 2\n");
```

```
if(prev_state != state)
{
spi_data_write(rec_msg.data, 1);
prev_state = 2;
}
bytes_rec = SSIDataGetNonBlocking(SSI2_BASE, &buffer);
// SSIDataGet(SSI2_BASE, &buffer);
UARTprintf("RX 2 - %x\n\r",buffer);
if(buffer == 0x02 && bytes_rec != 0)
{
state = STATE_GET_CTRL;
spi_data_write(packet, 1);
}
else if(buffer == 0x01)
{
state = STATE_SEND_TRID;
}
else if(buffer == 0x02)
{
state = STATE_GET_CTRL;
}
else if(buffer != 0x01 || buffer != 0x02)
{
state = STATE_GET_CTRL;
}
break;
case STATE_GET_CTRL:
UARTprintf("In state 3\n");
bytes_rec = SSIDataGetNonBlocking(SSI2_BASE, &buffer);
// SSIDataGet(SSI2_BASE, &buffer);
state = STATE_SEND_TRID;
prev_state = 3;
if(buffer == 0x01 || buffer == 0x02)
{
break;
}
UARTprintf("Control Message - %x\n\r",buffer);
decode_message(buffer);
GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1, 0);
```

```
//Send control message
break;
}
}


void self_control(IBStruct rec_msg)
{
uint16_t control_msg_to_send;
uint8_t source;
uint16_t data;

source = rec_msg.source;
data = rec_msg.data;
switch(source)
{
case 0x55:
if((data*0.25) > 30)
{
control_msg_to_send = 0x0100 | source;
}
else
{
control_msg_to_send = 0x0000 | source;
}
break;
case 0xAA:
if(data > 0x1aa)
{
control_msg_to_send = 0x0000 | source;
}
else
{
control_msg_to_send = 0x0a00 | source;
}
break;
}
decode_message(control_msg_to_send);
}
```

```
/*
* spi.h
*
* Created on: Apr 15, 2019
* Author: nachiket kelkar & puneet bansal
*/

#ifndef SRC_SPI_H_
#define SRC_SPI_H_
#include "driverlib/ssi.h"

#define MASTER SSI_MODE_MASTER
#define SLAVE SSI_MODE_SLAVE

#define STATE_SEND_TRID 0x01
#define STATE_SEND_DATA 0x02
#define STATE_GET_CTRL 0x03

typedef struct
{
uint8_t source;
uint16_t data;
}IBStruct;

/*
* Function name: spi_init()
* Description : This function initializes the SPI module in Master or Slave mode
and
* at the clock frequency input by the user.
* @param : uint32_t(mode to initialize the SPI Master or slave), uint32_t(clock
rate)
* @Comments : Mode can only be MASTER or SLAVE
* @return : void
*/
void spi_init(uint32_t, uint32_t);
```

```
/*
* Function name: spi_data_write()
* Description : This function takes the data to write and number of bytes to write
from
* user and write the data using SPI protocol.
* @param : uint64_t(data to be written), uint8_t(no of bytes to write)
* @Comments : Maximum of 8 bytes of data can be written at a time.
* @return : void
*/
void spi_data_write(uint64_t, uint8_t);


/*
* Function name: spi_data_read()
* Description : This function reads the byte value and return it to the user
* @param : void
* @return : uint8_t (data in the SPI data register is returned)
*/
uint16_t spi_data_read();


/*
* Function name: InterBoardSPI()
* Description : This function contains the logic for the inter-board
communication task
* @param : void *
* @return : void
*/
void InterBoardSPI(void *pvParameters);


/*
* Function name: decode_message()
* Description : This function gets the source from the control message and
parameter and
* passes the actuation message to appropriate queue of actuator.
* @param : uint16_t (control message received from the beagle bone)
* @return : void
```

```
*/
void decode_message(uint16_t);


/*
* Function name: spi_state_machine()
* Description : This function waits for the message from the sensor tasks, then it
send the data
* to the control node when the SPI transfer is initiated by the control node.
* It gets the control message from the control node and sends it for actuation.
* @param : void
* @return : void
*/
void spi_state_machine();


/*
* Function name: self_control()
* Description : This function is called when controller node is not present so that
the
* actuators can be controlled in reduced state.
* @param : IBStruct (received message)
* @return : void
*/
void self_control(IBStruct);
#endif /* SRC_SPI_H_ */

/****************************************************************
* File name : decisionTask.c *
* Authors : Puneet Bansal and Nachiket Kelkar *
* Description : Responsible for starting decision task which sends command
signal based on received sensor data *
* Tools used : GNU make, gcc, arm-linux-gnueabihf-gcc *
****************************************************************

#include "genericIncludes.h"
#include "decisionTask.h"
#include "mq.h"
```

```
extern void signal_handler(int , siginfo_t * , void* );

void *decisionTaskRoutine(void *dataObj)
{
spiStruct dataToSend;
decisionStruct dataReceived;
logStruct dataToSendToLog;

printf("Entered Decision Routine\n");
mqd_t decisionQueue = mqueue_init(DECISIONQUEUENAME,
DECISION_QUEUE_SIZE, sizeof(decisionStruct));
mqd_t spiQueue = mqueue_init(SPIQUEUENAME, SPI_QUEUE_SIZE,
sizeof(spiStruct));
mqd_t logQueue = mqueue_init(LOGQUEUENAME, LOG_QUEUE_SIZE,
sizeof(logStruct));

dataToSendToLog.logLevel = info;
dataToSendToLog.type = actuation;
dataToSendToLog.remoteStatus=none_state;

if(decisionQueue < 0)
{
perror("Failed to create decision queue");
}
if(spiQueue < 0)
{
perror("Failed to create spi queue");
}
if(logQueue < 0)
{
perror("Failed to create log queue");
}

while(1)
{

int ret = mq_receive(decisionQueue,
(char*)&dataReceived,sizeof(decisionStruct),0);
if(ret<0)
```

```
{
//printf("mq receive failed in decision task\n");
}
else
{
//printf("Data received from spi task is %x from source
%x\n",dataReceived.data,dataReceived.source);
if(dataReceived.source == 0x55)
{
uint16_t temp;
temp = dataReceived.data;
temp =temp >> 2 ;
dataToSendToLog.source = 0x55;
printf("source is temperature");
if(temp > TEMP_THRESHOLD)
{
dataToSend.sourceAndCommand = 0x01;
dataToSend.sourceAndCommand <<= 8;
dataToSend.sourceAndCommand |= dataReceived.source;

dataToSendToLog.data =0x01;
dataToSendToLog.remoteStatus=none_state;
}
else
{
dataToSend.sourceAndCommand = 0x00;
dataToSend.sourceAndCommand <<= 8;
dataToSend.sourceAndCommand |= dataReceived.source;
dataToSendToLog.data =0x00;
dataToSendToLog.remoteStatus=none_state;
}
int ret=mq_send(spiQueue, (char*)&dataToSend,sizeof(spiStruct),0);
if(ret<0)
{
printf("Sending to spi queue failed\n");
}
ret=mq_send(logQueue, (char*)&dataToSendToLog,sizeof(logStruct),0);
if(ret<0)
{
```

```
printf("Sending to log queue failed\n");
}
}
if(dataReceived.source == 0xaa)
{
printf("Source is soil moisture");
uint8_t command;
dataToSendToLog.source = 0xaa;
command = getCommand(dataReceived.data);
dataToSend.sourceAndCommand = command;
dataToSend.sourceAndCommand <<= 8;
dataToSend.sourceAndCommand |= dataReceived.source;
dataToSendToLog.data =command;
dataToSendToLog.remoteStatus=none_state;

printf("Command message is %x\n",dataToSend.sourceAndCommand);
int ret=mq_send(spiQueue, (char*)&dataToSend,sizeof(spiStruct),0);
if(ret<0)
{
printf("Sending to spi queue failed\n");
}
ret=mq_send(logQueue, (char*)&dataToSendToLog,sizeof(logStruct),0);
if(ret<0)
{
printf("Sending to log queue failed\n");
}

}


}
if(exitThread)
{
break;
}

}
mq_close(decisionQueue);
mq_close(spiQueue);
mq_close(logQueue);
mq_unlink(DECISIONQUEUENAME);
mq_unlink(SPIQUEUENAME);
mq_unlink(LOGQUEUENAME);
pthread_exit(NULL);
}

uint8_t getCommand(uint16_t data)
{
uint8_t commandToSend;
if(data>0 && data <500)
{
commandToSend = 10;
}
else if(data>=500 && data <1500)
{
commandToSend = 6;
}
else if(data>=1500 && data <3000)
{
commandToSend = 3;
}
else if(data>=3000 && data <4095)
{
commandToSend = 0;
}
return commandToSend;
}



/*****************************************************************
* File name : decisionTask.h *
* Authors : Puneet Bansal and Nachiket Kelkar *
* Description : Contains header files and function definitions for decisionTask.c
*
* Tools used : GNU make, gcc, arm-linux-gnueabihf-gcc *
******************************************************************
```

```
#include
#define TEMP_THRESHOLD 30
#define SOILMOISTURE_THRESHOLD 100

/**
* @brief : function that returns a command message based on the defined
thresholds
* @param1 : 16 bit data for which the commmand is to be generated.
* @returns : 8 bit command
* */
uint8_t getCommand(uint16_t);

/**
* @brief : Decision Task callback function. Receives sensor data from SPI task
over message queue, determines the control message
* and sends it to the SPI taks and the logger task.
* */

void *decisionTaskRoutine(void *);

/*****************************************************************
* File name : genericincludes.h *
* Authors : Puneet Bansal and Nachiket Kelkar *
* Description : Contains header files common to multiple files *
* Tools used : GNU make, gcc, arm-linux-gnueabihf-gcc *
******************************************************************

#include
#include
#include
#include
#include
#include
#include
#include
#include
#include
#include
#include
```

```
#include "gpio.h"
#include

/**************************GLOBAL
VARIBALES**************************************************/
uint8_t spi_handler;
uint8_t connection_handler;
uint8_t recoveryIndiacation;
uint8_t printOnlyOnce;
bool exitThread;
uint8_t notDegraded;
uint8_t revived;

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


/*************************************************************
* File name : gpio.c *
* Authors : Nachiket Kelkar and Puneet Bansal *
* Description : The functions used for gpio operations. Setting the direction of
pin and *
* the value. This functions are restricted for use of only USER LED pins. *
* Tools used : GNU make, gcc, arm-linux-gcc. *
*************************************************************
#define _GNU_SOURCE

/* Including standard libraries */
#include
#include
#include
#include
#include
#include
#include
#include

/* Including user libraries */
#include "gpio.h"
```

```
void gpio_init(int gpio_pin,int gpio_direction)
{
//FILE *fp;
int fp;
char *file = (char*)malloc(40);
char pintoWrite[10];

//if(is_pin_valid(gpio_pin))
{
fp = open("/sys/class/gpio/export", O_RDWR);
//fprintf(fp,"%d",gpio_pin);

sprintf(pintoWrite,"%d",gpio_pin);
write(fp,pintoWrite,sizeof(pintoWrite));
close(fp);

sprintf(file,"/sys/class/gpio/gpio%d/direction",gpio_pin);
fp = open(file,O_RDWR);
if(gpio_direction == out)
{
write(fp,(void *)"out",sizeof("out"));
//fprintf(fp,"out");
}
else if(gpio_direction == in)
{
write(fp,(void *)"in",sizeof("in"));
//fprintf(fp,"in");
}
else
{
printf("Enter direction only as in or out");
}
close(fp);
}
//else
{
//printf("Enter valid pin number");
}
free(file);
}


void gpio_write_value(int gpio_pin, int gpio_value)
{
//FILE *fp;
int fp;
char *file = (char*)malloc(40);

//if(is_pin_valid(gpio_pin))
{
sprintf(file,"/sys/class/gpio/gpio%d/value",gpio_pin);
//printf("file name is %s\n",file);

fp = open(file,O_RDWR);

if(gpio_value == low)
{
write(fp,(void *)"0",sizeof("0"));
//fprintf(fp,"%d",low);
}
else if(gpio_value == high)
{
write(fp,(void *)"1",sizeof("1"));
//fprintf(fp,"%d",high);
}
else
{
printf("Enter value only as low or high");
}
close(fp);
}
//else
{
//printf("Enter valid pin number");
}
free(file);
}
```

```
void pwm_generate(uint8_t duty_cycle)
{
int i;
// while(1)
// {
for(i=0;i {
gpio_write_value(56,1);
}
for(i=duty_cycle;i<10;i++)
{
gpio_write_value(56,0);
}
// }
}

void toggle_led()
{
uint32_t i;
i=0;
gpio_write_value(55,1);
for(i=0;i<5000000;i++);
gpio_write_value(55,0);
for(i=0;i<5000000;i++);
}

/*
int gpio_read_value(int gpio_pin)
{
FILE *fp;
char *file = (char*)malloc(40);
int value;

if(is_pin_valid(gpio_pin))
{
sprintf(file,"/sys/class/gpio/gpio%d/value",gpio_pin);
fp = fopen(file,"r");
fscanf(fp,"%d",&value);
fclose(fp);
```

```
}
else
{
printf("Enter valid pin number");
}
free(file);
return value;
}


bool is_pin_valid(int gpio_pin)
{
int gpio_allowed[total_gpio] = access_pin_allowed;
bool is_valid = false;

for(int i=0; i {
if(gpio_pin == gpio_allowed[i])
is_valid = is_valid | true;
else
is_valid = is_valid | false;
}
return is_valid;
}


void gpio_interrupt_state(int gpio_pin, gpio_interrupt interrupt)
{
FILE *fp;
char *file = (char*)malloc(40);

if(is_pin_valid(gpio_pin))
{
gpio_init(gpio_pin,in);
sprintf(file,"/sys/class/gpio/gpio%d/edge",gpio_pin);
fp = fopen(file,"w");
switch(interrupt)
{
case rising:
fprintf(fp,"rising");
break;
case falling:
fprintf(fp,"falling");
break;
case both:
fprintf(fp,"both");
break;
case none:
fprintf(fp,"none");
break;
}
fclose(fp);
}
else
{
printf("Enter valid pin number");
}
free(file);
}


int gpio_open_value(int gpio_pin)
{
char *file = (char*)malloc(40);
int fd;
if(is_pin_valid(gpio_pin))
{
sprintf(file,"/sys/class/gpio/gpio%d/value",gpio_pin);
fd = open(file, O_RDONLY);
}
else
{
printf("Enter valid pin number");
fd = -1;
}
return fd;
}
```

```
int gpio_read_val_with_fd(int fd)
{
int value;
read(fd, &value, sizeof(value));
lseek(fd, 0, SEEK_SET);
return value & 0x1;
}
*/

/*****************************************************************
* File name : gpio.h *
* Authors : Nachiket Kelkar and Puneet Bansal *
* Description : The functions used for gpio operations. Setting the direction of
pin and *
* the value. *
* Tools used : GNU make, gcc, gcc-linux-gcc. *
******************************************************************
#include

#define total_gpio 5
#define access_pin_allowed {53,54,55,56,22}

/**************** Enumerations used for gpio direction and gpio value
****************/
enum gpio_direction{
in = 0,
out,
};

enum gpio_value{
low = 0,
high,
};

typedef enum{
falling,
rising,
both,
none,
}gpio_interrupt;

/********************** Functions for the gpio operations
*********************/
/*
* Function name:- gpio_init
* Description:- The function takes the gpio pin number and assignes it as input
pin or
* output pin.
* @param:- int (gpio pin number), int (gpio pin direction)
* @return:- void
* gpio pin direction - 0 for in and 1 for out.
*/
void gpio_init(int,int);


/*
* Function name:- gpio_write_value
* Description:- The function takes the gpio pin number and outputs the pin high
or low.
* @param:- int (gpio pin number), int (gpio pin value)
* @return:- void
* gpio pin direction - 0 for in and 1 for out.
*/
void gpio_write_value(int,int);


/*
* Function name:- gpio_read_value
* Description:- The function takes the gpio pin number and returns the value on
the pin.
* @param:- int (gpio pin number), int (gpio pin value)
* @return:- int (value high or low)
*/
int gpio_read_value(int);


/*
* Function name:- is_pin_valid
* Description:- The function takes the gpio pin number and returns if valid pin
no is entered.
* @param:- int (gpio pin number)
* @return:- bool (true if pin number is valid and false if not)
* gpio pin direction - 0 for in and 1 for out.
* Need to maintain pin values and no of valid pins in above define.
*/
bool is_pin_valid(int);


/*
* Function name:- gpio_interrupt_state
* Description:- The function takes the gpio pin number and sets the gpio
interrupt as rising
* falling, both or none based on second parameter.
* @param:- int (gpio pin number), gpio_interrupt (which interrupt);
* @return:- void
* Comments:- gpio_interrupt: can be falling, rising, both or none to disable the
interrupts.
* Need to maintain pin values and no of valid pins in above define.
*/
void gpio_interrupt_state(int, gpio_interrupt);


/*
* Function name:- gpio_open_value
* Description:- The function takes the gpio pin number and opens the file and
returns the
* file descriptor.
* @param:- int (gpio pin number)
* @return:- int (file descriptor)
* Comments:- Need to maintain pin values and no of valid pins in above define.
*/
int gpio_open_value(int);


/*
* Function name:- gpio_read_val_with_fd
* Description:- The function takes the file descriptior of gpio pin and reurnts the
```

```
state of the
* pin wether high or low.
* @param:- int (file descriptor)
* @return:- int (pin state)
* Comments:- pin state: Pin state can be high or low.
*/
int gpio_read_val_with_fd(int);

void pwm_generate(uint8_t);
void toggle_led();


/**************************************************************
* File name : loggerTask.c *
* Authors : Puneet Bansal and Nachiket Kelkar *
* Description : Functions to configure logging to a file *
* Tools used : GNU make, gcc, arm-linux-gnueabihf-gcc *
***************************************************************

#include "genericIncludes.h"
#include "loggerTask.h"

extern uint8_t count;
extern void signal_handler(int , siginfo_t * , void* );

void *loggerTaskRoutine(void *fileName)
{
logStruct dataReceived;

printf("Entered Logger Routine\n");
mqd_t decisionQueue = mqueue_init(DECISIONQUEUENAME,
DECISION_QUEUE_SIZE, sizeof(decisionStruct));
mqd_t spiQueue = mqueue_init(SPIQUEUENAME, SPI_QUEUE_SIZE,
sizeof(spiStruct));
mqd_t logQueue = mqueue_init(LOGQUEUENAME, LOG_QUEUE_SIZE,
sizeof(logStruct));

while(1)
{
```

```
int ret = mq_receive(logQueue,(char*)&dataReceived,sizeof(logStruct),0);
if(ret<0)
{
//printf("mq receive failed in decision task\n");
}
else
{
logToFile(fileName,dataReceived);

}
if(exitThread)
{
break;
}
}
mq_close(decisionQueue);
mq_close(spiQueue);
mq_close(logQueue);
mq_unlink(DECISIONQUEUENAME);
mq_unlink(SPIQUEUENAME);
mq_unlink(LOGQUEUENAME);
pthread_exit(NULL);
}

char* printTimeStamp()
{
char* time_stamp=malloc(40);
struct timespec thTimeSpec;
clock_gettime(CLOCK_REALTIME, &thTimeSpec);
sprintf(time_stamp,"[s: %ld, ns: %ld]",thTimeSpec.tv_sec,thTimeSpec.tv_nsec);
return time_stamp;
}


void logToFile(char *fileName, logStruct dataToReceive)
{

FILE *logging;
char level[20];
char source[20];
char type[20];
char remoteNodeStatus[30];

if(dataToReceive.logLevel==alert)
{
strcpy(level,"[ALERT]");
}
else if(dataToReceive.logLevel==info)
{
strcpy(level,"[INFO]");
}
else
{
strcpy(level,"[DEBUG]");
}


if(dataToReceive.source == 0x55)
{
strcpy(source,"TEMPERATURE");
}
else if(dataToReceive.source == 0xaa)
{
strcpy(source,"SOIL MOISTURE");
}
else
{
strcpy(source,"");
}


printf("printing the value of source %s",source);

if(dataToReceive.type == actuation)
{
strcpy(type,"[ACTUATION DATA]");
}
else if(dataToReceive.type == sensing)
```

```
{
strcpy(type,"[SENSING DATA]");
}

if(dataToReceive.remoteStatus == degraded)
{
strcpy(remoteNodeStatus,"[DEGRADED STATE]");
}
else if(dataToReceive.remoteStatus == notActive)
{
strcpy(remoteNodeStatus,"[REMOTE NODE INACTIVE]");
}
else if(dataToReceive.remoteStatus == active)
{
strcpy(remoteNodeStatus,"[ACTIVE STATE]");
}


if(dataToReceive.remoteStatus==none_state)
{
if(count ==1)
{
logging = fopen(fileName,"w");
count=0;
}
else
{
logging = fopen(fileName,"a");
}

fprintf(logging,"%s %s [%s] %s %d
\n",printTimeStamp(),level,source,type,dataToReceive.data);
fclose(logging);
}
else /*if(dataToReceive.remoteStatus== degraded || dataToReceive.remoteStatus
==notActive || dataToReceive.remoteStatus ==active) */
{

if(count ==1)
{
logging = fopen(fileName,"w");
count=0;
}
else
{
logging = fopen(fileName,"a");
}
fprintf(logging,"%s %s %s\n",printTimeStamp(),level,remoteNodeStatus);
fclose(logging);

}

}


/*********************************************************************
* File name : decisionTask.h *
* Authors : Puneet Bansal and Nachiket Kelkar *
* Description : Contains header files and function definitions for decisionTask.c
*
* Tools used : GNU make, gcc, arm-linux-gnueabihf-gcc *
**********************************************************************


#include "mq.h"
#include

/**
* @brief: Logger task call back function. Logs the following to a file.
* 1)Timestamp
* 2)Source of the sensor/actuation value
* 3)Sensor/Actuation data values
* 4)Log level : ALERT, DEBUG, INFO
* 5)Reporting if remote node is in active, degraded or not active state.
*
* Receives data from spitask and decision task and logs it to the file.
*
* @param1: name of the log file I
* * */
void *loggerTaskRoutine(void *);

/**
* @brief : returns the time stamp value
* @preturns: timestamp
* */
char* printTimeStamp();

/**
* @brief: opens the file in appropriate mode, and logs messages depending upon several conditions.
* * */
void logToFile(char *, logStruct);

/*************************************************************
* File name : maintask.c *
* Authors : Puneet Bansal and Nachiket Kelkar *
* Description : The main logic of the code *
* Tools used : GNU make, gcc, arm-linux-gnueabihf-gcc *
**************************************************************

#include "genericIncludes.h"
#include "mainTask.h"

pthread_t spiTask,decisionTask,loggerTask;
extern void signal_handler(int , siginfo_t * , void* );
char *logFileName;
uint8_t count;

int main(int argc, char *argv[])
{
spi_handler=0;
count=1;
temp=0;
soilMoisture=0;
recoveryIndiacation=0;
printOnlyOnce=0;
connection_handler =0;
```

```
exitThread=false;
notDegraded=0;
revived=0;
gpio_init(55,1);
gpio_init(56,1);
gpio_write_value(55,0);
gpio_write_value(56,0);
degradedState=0;
logFileName = malloc(20);
if(argc ==2)
{
strcpy(logFileName, argv[1]);
}
else
{
strcpy(logFileName, "logFile.txt");
}

if(pthread_create(&spiTask,NULL,&spiTaskRoutine,NULL)!=0)
{
perror("SPI Task create failed");
}
if(pthread_create(&decisionTask,NULL,&decisionTaskRoutine,NULL)!=0)
{
perror("Decision Task create failed");
}
if(pthread_create(&loggerTask,NULL,&loggerTaskRoutine,logFileName)!=0)
{
perror("Logger Task create failed");
}
while(1)
{
if(exitThread)
{
break;
}
}

pthread_join(spiTask,NULL);
pthread_join(decisionTask,NULL);
pthread_join(loggerTask,NULL);
return 0;
}

/**************************************************************
* File name : maintask.c *
* Authors : Puneet Bansal and Nachiket Kelkar *
* Description : The main logic of the code *
* Tools used : GNU make, gcc, arm-linux-gnueabihf-gcc *
***************************************************************

#include
#include "spiTask.h"
#include "decisionTask.h"
#include "loggerTask.h"
#include
#include "mq.h"
#include

mqd_t mqueue_init(const char* queue_name, int queue_size, int message_size)
{
mqd_t msg_q_des;
struct mq_attr queue_attr;
//printf("queue name in %s is %s\n",__func__,queue_name);
//printf("queue size in %s is %d\n",__func__,queue_size);
//printf("queue name in %s is %s",__func__,queue_name);
queue_attr.mq_maxmsg = queue_size;
queue_attr.mq_msgsize = message_size;
queue_attr.mq_flags = O_NONBLOCK;
msg_q_des = mq_open(queue_name, O_CREAT | O_RDWR | O_NONBLOCK,
0666, &queue_attr);

return msg_q_des;
}
```

```
/****************************************************************
* File name : mq.h *
* Authors : Nachiket Kelkar and Puneet Bansal *
* Description : Function definitions of wrapper made to initialise queue *
* Tools used : GNU make, gcc, arm-linux-gnueabihf-gcc *
****************************************************************

#include
#include
#include

/* Message queues for all the tasks */
#define SPIQUEUENAME "/spiqueue1"
#define DECISIONQUEUENAME "/decisionqueue1"
#define LOGQUEUENAME "/logqueuequeue11"

/* Message queue size for all the tasks */
#define SPI_QUEUE_SIZE 10
#define DECISION_QUEUE_SIZE 10
#define LOG_QUEUE_SIZE 10


typedef enum
{
info,
alert,
debug
}loglevel_enum;

typedef enum{
sensing,
actuation,
}type_enum;

typedef enum{
none_state,
active,
degraded,
notActive,
}remoteNodeStatus_typedef;

/* Structure to communicate to the maintask*/
typedef struct
{ uint16_t sourceAndCommand;
}spiStruct;

/* Structure to communicate to the temperature task*/
typedef struct{
uint8_t source;
uint16_t data;
}decisionStruct;

typedef struct{
uint8_t source;
uint16_t data;
loglevel_enum logLevel;
type_enum type;
remoteNodeStatus_typedef remoteStatus;
}logStruct;


/*user defined functions*/

/**
* @name: mqueue_init
*
* @param1: message queue name
* @param2: max message queue size
* @param3: size of the data to send
*
* @description: wrapper around mq_open function. Sets the attributes of the
queue and opens the queue with the specified parameters.
*
* return: message queue file descriptor.
* */
mqd_t mqueue_init(const char*, int, int);
```

```
/****************************************************************
* File name : spitask.c *
* Authors : Puneet Bansal and Nachiket Kelkar *
* Description : Functions to configure spi, send and receive values to the remote
node via SPI *
* Tools used : GNU make, gcc, arm-linux-gnueabihf-gcc *
* References :
https://raw.githubusercontent.com/torvalds/linux/master/tools/spi/spidev_test.c *
****************************************************************

#include "genericIncludes.h"
#include "spiTask.h"
#include "mq.h"
#include "myTimer.h"

static const char *device = "/dev/spidev1.0";
static uint32_t mode;
static uint8_t bits = 16;
static uint32_t speed = 250000;
static uint16_t delay;
uint16_t tx1;
uint16_t rx1;
uint16_t tx;
uint16_t rx;

void signal_handler(int , siginfo_t * , void* );

void *spiTaskRoutine(void *dataObj)
{
printf("Entered SPI Task\n");

int spi_fd;
uint8_t present_trid,present_source;

spiStruct dataReceived;
decisionStruct dataToSend;
logStruct dataToSendToLog;

spi_fd=spi_init();
```

```
/***************Configuring the timer to poll remote node every 2
second****************/
struct sigevent spiEvent;
struct sigaction spiAction;
struct itimerspec spiSpec;
timer_t spiTimer;


spiAction.sa_flags = SA_SIGINFO;
spiAction.sa_sigaction = signal_handler;

if((sigaction(SIGRTMIN, &spiAction, NULL))<0)
{
perror("Failed setting timer handler for SPI");
}

//Assigning signal to timer
spiEvent.sigev_notify = SIGEV_SIGNAL;
spiEvent.sigev_signo = SIGRTMIN;
spiEvent.sigev_value.sival_ptr = &spiTimer;

if((timer_create(CLOCK_REALTIME, &spiEvent, &spiTimer)) < 0)
{
perror("Timer creation failed for spi task");
}

//Setting the time and starting the timer
spiSpec.it_interval.tv_nsec = 0;
spiSpec.it_interval.tv_sec = 2;
spiSpec.it_value.tv_nsec = 0;
spiSpec.it_value.tv_sec = 2;

if((timer_settime(spiTimer, 0, &spiSpec, NULL)) < 0)
{
perror("Starting timer in SPI task failed");
}
```

```
/********************Configuring a 5 second timer to check remote node
connection********************/

struct sigevent connectionEvent;
struct sigaction connectionAction;
struct itimerspec connectionSpec;
timer_t connectionTimer;


connectionAction.sa_flags = SA_SIGINFO;
connectionAction.sa_sigaction = signal_handler;

if((sigaction(SIGRTMIN+1, &connectionAction, NULL))<0)
{
perror("Failed setting timer handler for SPI");
}

//Assigning signal to timer
connectionEvent.sigev_notify = SIGEV_SIGNAL;
connectionEvent.sigev_signo = SIGRTMIN + 1;
connectionEvent.sigev_value.sival_ptr = &connectionTimer;

if((timer_create(CLOCK_REALTIME, &connectionEvent, &connectionTimer))
< 0)
{
perror("Timer creation failed for checking connection");
}

//Setting the time and starting the timer
connectionSpec.it_interval.tv_nsec = 0;
connectionSpec.it_interval.tv_sec = 1;
connectionSpec.it_value.tv_nsec = 0;
connectionSpec.it_value.tv_sec = 8;

if((timer_settime(connectionTimer, 0, &connectionSpec, NULL)) < 0)
{
perror("Starting timer for checking connection failed");
}
```

```
/*********************************************************************
mqd_t decisionQueue = mqueue_init(DECISIONQUEUENAME,
DECISION_QUEUE_SIZE, sizeof(decisionStruct));
mqd_t spiQueue = mqueue_init(SPIQUEUENAME, SPI_QUEUE_SIZE,
sizeof(spiStruct));
mqd_t logQueue = mqueue_init(LOGQUEUENAME, LOG_QUEUE_SIZE,
sizeof(logStruct));

if(decisionQueue < 0)
{
perror("Failed to create decision queue");
}
if(spiQueue < 0)
{
perror("Failed to create spi queue");
}
if(logQueue < 0)
{
perror("Failed to create spi queue");
}

dataToSendToLog.type=sensing;
dataToSendToLog.logLevel=info;

int c=0;
prev_trid=DEFAULT_TRID;
while(1)
{
dataToSendToLog.logLevel=none_state;
int k;
if(connection_handler)
{
connection_handler=0;
printf("Remote Node not active\n");

gpio_write_value(55,1); //turning on the LED if remote node is not active.

dataToSendToLog.logLevel=alert;
dataToSendToLog.remoteStatus=notActive;
```

```
if(mq_send(logQueue,(char*)&dataToSendToLog,sizeof(logStruct),0)!=0)
{
printf("Data sending from spitask to logTask failed 1");
}
}
else if(degradedState==1)
{

degradedState=0;
dataToSendToLog.logLevel=alert;
dataToSendToLog.remoteStatus=degraded;

gpio_write_value(56,1); //turning on LED on pin 56 if remote node working in degraded state
//led blink
if(mq_send(logQueue,(char*)&dataToSendToLog,sizeof(logStruct),0)!=0)
{
printf("Data sending from spitask to logTask failed 2");
}
}
else if(revived==1)
{
revived=0;
dataToSendToLog.logLevel=alert;
dataToSendToLog.remoteStatus=active;
if(mq_send(logQueue,(char*)&dataToSendToLog,sizeof(logStruct),0)!=0)
{
printf("Data sending from spitask to logTask failed 2");
}
}
// else
// {
// dataToSendToLog.logLevel=none_state;
// }
dataToSendToLog.logLevel=none_state;

/*Polling the remote node*/
if(spi_handler==1)
{
```

```
spi_handler=0;
tx=POLL_REQ;
rx=0;
spi_transfer(spi_fd,&tx,&rx,2);
printf("rx is %x\n",rx);
present_source= (rx & SOURCE_BITMASK);
present_trid= (rx & TRID_BITMASK) >> 8;
printf("source is %x and trid is %x\n",present_source,present_trid);

if(present_trid != prev_trid )
{
gpio_write_value(55,0); //turning off the led if remote node is active
checkDegradedState(present_source);
if((timer_settime(connectionTimer, 0, &connectionSpec, NULL)) < 0)
{
perror("Starting timer for checking connection failed");
}

for(k=0;k<100000;k++); //Giving inline waits to synchronise the communication
for(k=0;k<100000;k++);
for(k=0;k<100000;k++);
for(k=0;k<100000;k++);

tx1=DATA_REQ;
rx1=0;

spi_transfer(spi_fd, &tx1, &rx1, 2);

dataToSend.source=present_source;
dataToSend.data=rx1;
dataToSendToLog.source = present_source;
dataToSendToLog.data = rx1;

printf("Data is %x\n",dataToSendToLog.data);
printf("Source is %x\n",dataToSendToLog.source);

if(mq_send(decisionQueue,(char*)&dataToSend,sizeof(decisionStruct),0)!=0)
{
printf("Data sending from spitask to decisionTak failed");
}
else
{
//printf("Before mq receive in spitask\n");
if(mq_receive(spiQueue,(char*)&dataReceived,sizeof(spiStruct),0)>-1)
{
//printf("data received from decision queue in spitask
%x\n",dataReceived.sourceAndCommand);
spi_transfer(spi_fd, &dataReceived.sourceAndCommand, &rx1, 2);
}

}
dataToSendToLog.remoteStatus=none_state;
if(mq_send(logQueue,(char*)&dataToSendToLog,sizeof(logStruct),0)!=0)
{
printf("Data sending from spitask to logTask failed");
}

//printf("Came out of mq_receive in spiTask\n");
prev_trid= present_trid;

}
for(k=0;k<100000;k++);
for(k=0;k<100000;k++);
for(k=0;k<100000;k++);
for(k=0;k<100000;k++);

}
if(exitThread)
{
break;
}
}

mq_close(decisionQueue);
mq_close(spiQueue);
mq_close(logQueue);

mq_unlink(DECISIONQUEUENAME);
mq_unlink(SPIQUEUENAME);
mq_unlink(LOGQUEUENAME);

timer_delete(spiTimer);
timer_delete(connectionTimer);

pthread_exit(NULL);
}

int spi_init()
{
int fd;
int ret=0;

fd = open(device, O_RDWR);
if (fd < 0)
{
perror("can't open device");
abort();
}


ret = ioctl(fd, SPI_IOC_WR_MODE32, &mode);
if (ret == -1)
{
perror("can't set spi mode");
abort();
}

ret = ioctl(fd, SPI_IOC_RD_MODE32, &mode);
if (ret == -1)
{
perror("can't get spi mode");
abort();
}

ret = ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &bits);
if (ret == -1)
{
```

```
perror("can't set bits per word");
abort();
}

ret = ioctl(fd, SPI_IOC_RD_BITS_PER_WORD, &bits);
if (ret == -1)
{
perror("can't get bits per word");
abort();
}

ret = ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed);
if (ret == -1)
{
perror("can't set max speed hz");
abort();
}

ret = ioctl(fd, SPI_IOC_RD_MAX_SPEED_HZ, &speed);
if (ret == -1)
{
perror("can't get max speed hz");
abort();
}

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);

return fd;
}

void spi_transfer(int fd, uint16_t const *tx, uint16_t const *rx, size_t len)
{
int ret;

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)
{
perror("can't send spi message");
abort();
}

}

void checkDegradedState(uint8_t source)
{
if(source==0x55)
{
temp++;
}
else if(source ==0xaa)
{
soilMoisture++;
```

```
}
if(abs(temp-soilMoisture)>10)
{
printf("------------------------------------>DEGRADED STATE<-----------------------
--------------- \n");
degradedState=1;
temp=0;
soilMoisture=0;
notDegraded=0;
}
else if(abs(temp-soilMoisture)<20)
{
notDegraded++;
if(notDegraded>10)
{
printf("------------------------------------>ACTIVE<--------------------------------------
\n");
revived=1;
gpio_write_value(56,0);
notDegraded=0;
}

}



}

void signal_handler(int sig, siginfo_t * var1, void* var2)
{
switch(sig)
{
case 2:
printf("SIGINT signal is received\n ----------> Exiting thread <---------\n");
exitThread = true;
break;
case 34:
spi_handler=1;
break;
```

```
case 35:
connection_handler=1;
break;
}
}




/**************************************************************
* File name : spiTask.h *
* Authors : Nachiket Kelkar and Puneet Bansal *
* Description : Contains header files and function definitions for spiTask.c *
* Tools used : GNU make, gcc, arm-linux-gnueabihf-gcc *
***************************************************************

#include "myTimer.h"

#define POLL_REQ 0x0001
#define DATA_REQ 0X0002
#define DEFAULT_TRID 0x00
#define SOURCE_BITMASK 0x00ff
#define TRID_BITMASK 0xff00
uint8_t prev_trid;
uint8_t degradedState;

/**
* @brief: Spi task call back function. Creates timers to take poll slave after
every 2 second and to check whether slave is active or not.
* Takes the message from slave via SPI, sends it to the decision task to get
command for the data and to logger task. It also receives the command
* messages from decisionTask and sends it via SPI to the slave.
*
* */
void *spiTaskRoutine(void *);

/**
* @brief: Configures SPI
* @returns : file descriptor for SPI
* */
int spi_init();

/**
* @brief: Uses IOCTL command to send and receive specific bytes of data from the slave.
* @param1 : file descriptor for SPI
* @param2 : buffer pointing to the data to send
* @param3 : buffer pointing to the data received.
* @param4 : number of bytes of data to transmit/receive.
*
* */
void spi_transfer(int , uint16_t const *, uint16_t const *, size_t);

/**
* @brief : function to check if the remote node is operating in degraded state or not.
* @param 1: sensor source to which the received message pertains.
* * */
void checkDegradedState(uint8_t);

/*************GLOBAL VARIABLES**************************/
uint8_t temp,soilMoisture;
```