SliceKit GPIO Example Applications

REV A

Publication Date: 2013/3/12

XMOS®

Table of Contents

1 Overview

This document covers example applications that demonstrate various features of the XA-SK-GPIO Slice Card including the ADC, LEDs, UART connector and buttons, the I2C master xSOFTip component, as well as various basic features of the xCORE processor.

Low level details of how GPIO example application is implemented are covered here, with information about how to modify and experiment with them.

For getting started instructions for these applications, please refer to their respective Quick Start guides.

2 Evaluation Platforms

IN THIS CHAPTER

2.1 Recommended Hardware

This application may be evaluated using the Slicekit Modular Development Platform, available from digikey. Required board SKUs are:

- XP-SKC-L2 (Slicekit L2 Core Board) plus XA-SK-GPIO plus XA-SK-XTAG2 (Slicekit XTAG adaptor) plus XTAG2 (debug adaptor)

2.2 Example Applications

2.2.1 app_slicekit_simple_demo

This application has the following features:

- module_i2c_master from the xSOFt ip library is used to access the external ADC, which is equipped with an external linearised thermistor circuit for temperature sensing.
- simple code to print the recorded temperature to the XDE debug console on the press of one of the Slice Card buttons
- simple code to cycle through the 4 LEDs each time the other button is pressed.
- demonstrates use of XC select statements for handling multiple concurrent inputs
- demonstrates basic usage of XCore ports

2.2.2 app_slicekit_com_demo

This application extends the simple demo to provide the following functionality (all options are dynamically reconfigurable via the APIs for app_slicekit_com_demo application):

-
  - Baud Rate: 150 to 115200 bps
  - Parity: None, Mark, Space, Odd, Even
  - Stop Bits: 1,2
  - Data Length: 1 to 30 bits (Max 30 bits assumes 1 stop bit and no parity)

- Cycles Through LEDs on button Press
- Displays temperature value and button press events on the terminal console of a host PC via the UART

2.2.3 app_sk_gpio_eth_combo_demo

- XA-SK-E100 Ethernet Slice Card is additionally required for this demo

This application extends the GPIO com port demo to utilize Ethernet and XTCP components in order to host a web page to

- Turn GPIO Slice Card LEDs on and off
- Display GPIO Slice Card button press status
- Read the room temperature via the onboard ADC and display on the web page

2.2.4 app_sk_gpio_wifi_tiwisl_combo_demo

- XA-SK-WIFI Slice Card is additionally required for this demo

This application extends the GPIO com port demo to utilize Wi-Fi component in order to host a web page to

- Turn GPIO Slice Card LEDs on and off
- Display GPIO Slice Card button press status
- Read the room temperature via the on-board ADC and display on the web page

3 Programming Guide

IN THIS CHAPTER

3.1 Simple Demo

3.1.1 Structure

All of the files required for operation are located in the `app_sk_gpio_simple_demo/src` directory. The files that are need to be included for use of this component in an application are:

File	Description
`common.h`	**Header file for API interfaces and Look up tables for thermistor.**
`main.xc`	Main file which implements the demo functionality

3.1.2 API

```
void app_manager(chanend c_uartTX,
                 chanend c_chanRX,
                 chanend c_process,
                 chanend c_end)
```

Polling uart RX and push button switches and send received commands to process_data thread.

This function has the following parameters:

`c_uartTX` Channel to Uart TX Thread

`c_chanRX` Channel to Uart RX Thread

`c_process` Channel to process data Thread

`c_end` Channel to read data from process thread

```
int linear_interpolation(int adc_value)
```

Calculates temperatue based on linear interpolation.

This function has the following parameters:

adc_value int value read from ADC

This function returns:

Returns linear interpolated Temperature value

int read_adc_value()

Read ADC value using I2C.

This function returns:

Returns ADC value

3.1.3 Usage and Implementation

The port declaration for the LEDs, Buttons and I2C are declared as below. LEDs and Buttons use 4 bit ports and I2C uses 1 bit port for SCL(I2c Clock) and SDA (I2C data).

```
on stdcore[CORE_NUM]: out port p_led=XS1_PORT_4A;
on stdcore[CORE_NUM]: port p_PORT_BUT_1=XS1_PORT_4C;
on stdcore[CORE_NUM]: struct r_i2c i2cOne = {
                XS1_PORT_1F,
                XS1_PORT_1B,
                1000
 };
```

The app_manager API writes the configuration settings information to the ADC as shows below.

```
i2c_master_write_reg(0x28, 0x00, data, 1, i2cOne); //Write configuration information to
  ↪ ADC
```

The select statement in the app_manager API selects one of the two cases in it, checks if there is IO event or timer event. This statement monitors both the events and executes which ever event is occurred first. The select statement in the application is listed below. The statement checks if there is button press or not. If there is button press then it looks if the button state is same even after 20msec. If the buton state is same then it recognises as an valid push.

```
select
{
        case button => p_PORT_BUT_1 when pinsneq(button_press_1):> button_press_1: //
          ↪ checks if any button is pressed
                button=0;
                t:>time;
                break;

        case !button => t when timerafter(time+debounce_time):>void: //waits for 20ms and
          ↪  checks if the same button is pressed or not
                p_PORT_BUT_1:> button_press_2;
                if(button_press_1==button_press_2)
                if(button_press_1 == BUTTON_PRESS_VALUE) //Button 1 is pressed
                {
                        printstrln("Button 1 Pressed");
                        p_led<:(led_value);
                        led_value=led_value<<1;
```

```
                        led_value|=0x01;
                        led_value=led_value & 0x0F;
                        if(led_value == 15)
                        {
                                led_value=0x0E;
                        }
                }
                if(button_press_1 == BUTTON_PRESS_VALUE-1) //Button 2 is pressed
                {
                        data1[0]=0;data1[1]=0;
                        i2c_master_rx(0x28, data1, 2, i2cOne); //Read ADC value using I2C
                          ↪  read
                        printstrln("Reading Temperature value....");
                        data1[0]=data1[0]&0x0F;
                        adc_value=(data1[0]<<6)|(data1[1]>>2);
                        printstr("Temperature is :");
                        printintln(linear_interpolation(adc_value));
                }

                button=1;
                break;
}
```

After recognising the valid push then it checks if Button 1 is pressed or Button 2 is pressed. IF Button 1 is pressed then, the application reads the status of LEDs and shift the position of the LEDs to left by 1. If Button 2 is pressed, then the applciation reads the contents of ADC register using I2C read instruction and input the ADC value to linear interpolation function as shown below.

```
int linear_interpolation(int adc_value)
{
        int i=0,x1,y1,x2,y2,temper;
        while(adc_value<TEMPERATURE_LUT[i][1])
        {
                i++;
        }
//::Formula start
        //Calculating Linear interpolation using the formula y=y1+(x-x1)*(y2-y1)/(x2-x1)
//::Formula
        x1=TEMPERATURE_LUT[i-1][1];
        y1=TEMPERATURE_LUT[i-1][0];
        x2=TEMPERATURE_LUT[i][1];
        y2=TEMPERATURE_LUT[i][0];
        temper=y1+(((adc_value-x1)*(y2-y1))/(x2-x1));
        return temper;//Return Temperature value
}
```

The linear intepolation function calculates the linear interpolation value using the following formula and returns the temperature value from temperature look up table.

```
//Calculating Linear interpolation using the formula y=y1+(x-x1)*(y2-y1)/(x2-x1)
```

```
int TEMPERATURE_LUT[][2]= //Temperature Look up table
{
                {-10,850},{-5,800},{0,750},{5,700},{10,650},{15,600},{20,550},{25,500},{30,450},{35,40
                  ↪
```

```
                {40,350},{45,300},{50,250},{55,230},{60,210}
};
```

3.2 COM Port Demo

3.2.1 Structure

All of the files required for operation are located in the app_slicekit_simple_demo/src directory. The files that are need to be included for use of this component in an application are:

File	Description
common.h	**Header file for API interfaces and Look up tables for thermistor. FIXME - what about the uart**
main.xc	Main file which implements the demo functionality

3.2.2 API

```
void app_manager(chanend c_uartTX,
                 chanend c_chanRX,
                 chanend c_process,
                 chanend c_end)
```

Polling uart RX and push button switches and send received commands to process_data thread.

This function has the following parameters:

c_uartTX Channel to Uart TX Thread

c_chanRX Channel to Uart RX Thread

c_process Channel to process data Thread

c_end Channel to read data from process thread

```
void process_data(chanend c_process, chanend c_end)
```

process received data to see if received data is valid command or not Polling switches to see for button press

This function has the following parameters:

c_process Channel to receive data from app manager Thread

c_end Channel to communicate to app manager thread

```
void uart_tx_string(chanend c_uartTX, unsigned char message[100])
```

Transmits byte by byte to the UART TX thread for an input string.

This function has the following parameters:

`c_uartTX` Channel to receive data from app Uart TX Thread

`message` Buffer to store array of characters

`int linear_interpolation(int adc_value)`

Calculates temperatue based on linear interpolation.

This function has the following parameters:

`adc_value` int value read from ADC

This function returns:

Returns linear interpolated Temperature value

`int read_adc_value()`

Read ADC value using I2C.

This function returns:

Returns ADC value

3.2.3 Usage and Implementation

The port declaration for the LEDs, Buttons, I2C and UART are declared as below. LEDs and Buttons uses 4 bit ports, UART uses 1 bit ports both for Transmit and Receive and I2C uses 1 bit port for SCL(I2c Clock) and SDA (I2C data).

```
#define CORE_NUM 1
#define BUTTON_PRESS_VALUE 2
on stdcore[CORE_NUM] : buffered in port:1 p_rx =  XS1_PORT_1G;
on stdcore[CORE_NUM] : out port p_tx = XS1_PORT_1C;
on stdcore[CORE_NUM]: port p_led=XS1_PORT_4A;
on stdcore[CORE_NUM]: port p_button1=XS1_PORT_4C;
on stdcore[CORE_NUM]: struct r_i2c i2cOne = {
                XS1_PORT_1F,
                XS1_PORT_1B,
                1000
 };
```

The app_manager API writes the configuration settings information to the ADC as shows below.

```
i2c_master_write_reg(0x28, 0x00, i2c_register, 1, i2cOne); //Configure ADC by writing the
  ↪  settings to register
```

The select statement in the app_manager API selects one of the three cases in it, checks if there is IO event or timer event or any event on the Uart Receive pin. This statement monitors all the events and executes which ever event is occurred first. The select statement in the applciation is listed below. The statement checks if there is button press or availability of data on the Uart Receive pin. If there is button press then it looks if the button state is same as even after 200msec. If the buton state is same then it recognises as a valid push. If there is data on the Uart Receive

pin the it echoes the data back to the uart Transmit pin until > character is received in the input data.

```
                select
                {
                        case c_end:>data:
                                c_end:>data;
                                if(data == BUTTON_1) //Cycle LEDs on button 1 press
                                {
                                        printstrln("Button 1 Pressed");
                                        p_led<:(led_value);
                                        led_value=led_value<<1;
                                        if(led_value == 16) //If LED value is 16 then
                                          ↪ assigns LED value to 1 which cycles LEDs by
                                          ↪ button press
                                        {
                                                led_value=0x01;
                                        }
                                }
                                if(data == BUTTON_2) //Displays Temperature on console if
                                  ↪  Button 2 is pressed
                                {
                                        adc_value=read_adc_value();
                                        data_arr[0]=(linear_interpolation(adc_value));
                                        printstr("Temperature is :");
                                        printint(linear_interpolation(adc_value));
                                        printstrln(" C");
                                }
                                break;
                        case uart_rx_get_byte_byref(c_uartRX, rxState, buffer):
//::Command start
                                if(buffer == '>') //IUF received data is '>' character
                                  ↪ then expects cmd to endter into command mode
                                {
                                        j=0;
                                        uart_rx_get_byte_byref(c_uartRX, rxState, buffer)
                                          ↪ ;
                                        cmd_rcvbuffer[j]=buffer;
                                        if((cmd_rcvbuffer[j] == 'C' )|| (cmd_rcvbuffer[j]
                                          ↪  =='c')) //Checks if received data is 'C' or
                                          ↪  'c'
                                        {
                                                j++;
                                                uart_rx_get_byte_byref(c_uartRX, rxState,
                                                  ↪  buffer);
                                                cmd_rcvbuffer[j]=buffer;

                                                if((cmd_rcvbuffer[j] == 'm' )|| (
                                                  ↪ cmd_rcvbuffer[j] =='M')) //Checks if
                                                  ↪  received data is 'M' or 'm'
                                                {
                                                        j++;
                                                        uart_rx_get_byte_byref(c_uartRX,
                                                          ↪ rxState, buffer);
                                                        cmd_rcvbuffer[j]=buffer;
                                                        if((cmd_rcvbuffer[j] == 'D' )|| (
                                                          ↪ cmd_rcvbuffer[j] =='d'))//
                                                          ↪ Checks if received data is '
                                                          ↪ D' or 'd'
                                                        {
```

```
                                                uart_tx_send_byte(
                                                  ↪ c_uartTX, '\r');
                                                uart_tx_send_byte(
                                                  ↪ c_uartTX, '\n');
                                                uart_tx_string(c_uartTX,
                                                  ↪ CONSOLE_MESSAGES[0])
                                                  ↪ ;
                                                COMMAND_MODE=1; //
                                                  ↪ activates command
                                                  ↪ mode as received
                                                  ↪ data is '>cmd'
                                                uart_tx_send_byte(
                                                  ↪ c_uartTX, '\r');
                                                uart_tx_send_byte(
                                                  ↪ c_uartTX, '\n');
                                                uart_tx_send_byte(
                                                  ↪ c_uartTX, '>'); //
                                                  ↪ displays '>' if
                                                  ↪ command mode is
                                                  ↪ activated
                                        }
                                        else
                                        {
                                                uart_tx_send_byte(
                                                  ↪ c_uartTX, '>');
                                                for(int i=0;i<3;i++)
                                                        uart_tx_send_byte
                                                          ↪ (c_uartTX,
                                                          ↪ cmd_rcvbuffer
                                                          ↪ [i]); // if
                                                          ↪ received dta
                                                          ↪  is not 'c'
                                                          ↪ displays
                                                          ↪ back the
                                                          ↪ received
                                                          ↪ data
                                        }
                                }
                                else
                                {
                                        uart_tx_send_byte(c_uartTX, '>');
                                          ↪  //if received data is not '
                                          ↪ m' displays the received
                                          ↪ data
                                        for(int i=0;i<2;i++)
                                                uart_tx_send_byte(
                                                  ↪ c_uartTX,
                                                  ↪ cmd_rcvbuffer[i]);
                                }
                        }
                        else
                        {
                                uart_tx_send_byte(c_uartTX, '>');
                                uart_tx_send_byte(c_uartTX, cmd_rcvbuffer
                                  ↪ [j]);
                                j=0;
                        }
                }
                else
                {
```

```
                                        uart_tx_send_byte(c_uartTX,buffer); //Echoes back
                                          ↪  the input characters if not in command mode
                                }
//::Command
                                while(COMMAND_MODE) //Command mode activated
                                {
                                        j=0;
                                        skip=1;
                                        while(skip == 1)
                                        {
//::Send to process start
                                                select
                                                {
                                                        case uart_rx_get_byte_byref(
                                                          ↪ c_uartRX, rxState, buffer):
                                                              cmd_rcvbuffer[j]=buffer;
                                                              if(cmd_rcvbuffer[j++] ==
                                                                ↪ '\r')
                                                              {
                                                                      skip=0;
                                                                      j=0;
                                                                      while(
                                                                        ↪ cmd_rcvbuffer
                                                                        ↪ [j] != '\r')
                                                                      {
                                                                              c_process
                                                                                ↪ <:
                                                                                ↪ cmd_rcvbuffer
                                                                                ↪ [j];
                                                                                ↪  //
                                                                                ↪ received
                                                                                ↪
                                                                                ↪ valid
                                                                                ↪
                                                                                ↪ command
                                                                                ↪  and
                                                                                ↪
                                                                                ↪ send
                                                                                ↪  the
                                                                                ↪
                                                                                ↪ command
                                                                                ↪  to
                                                                                ↪ the
                                                                                ↪ process_data
                                                                                ↪
                                                                                ↪ theread
                                                                                ↪
                                                                              uart_tx_send_byte
                                                                                ↪ (
                                                                                ↪ c_uartTX
                                                                                ↪ ,
                                                                                ↪ cmd_rcvbuffer
                                                                                ↪ [j])
                                                                                ↪ ;
                                                                              j++;
                                                                      }
                                                                      cmd_rcvbuffer[j
                                                                        ↪ ]='\0';
                                                                      c_process<:
                                                                        ↪ cmd_rcvbuffer
                                                                        ↪ [j];
```

```
                                                        for(int inc=0;inc
                                                          ↪ <20;inc++)
                                                          ↪ //Clears the
                                                          ↪  command
                                                          ↪ buffer
                                                                cmd_rcvbuffer
                                                                  ↪ [inc
                                                                  ↪ ]='0';
                                                                  ↪
                                                        j=0;
                                                }
                                                break;
//::Send
                                        case c_end:>data:
                                                if(data!=EXIT && data!=
                                                  ↪ INVALID )
                                                {
                                                        uart_tx_string(
                                                          ↪ c_uartTX,
                                                          ↪ CONSOLE_MESSAGES
                                                          ↪ [3]); //
                                                          ↪ Displays
                                                          ↪ COmmand
                                                          ↪ Executed
                                                          ↪ Message on
                                                          ↪ Uart
                                                }
//::State machine
                                                switch(data)
                                                {
                                                        case EXIT: //Exit
                                                          ↪  from
                                                          ↪ command mode
                                                                COMMAND_MODE
                                                                  ↪ =0;
                                                                skip=0;
                                                                uart_tx_string
                                                                  ↪ (
                                                                  ↪ c_uartTX
                                                                  ↪ ,
                                                                  ↪ CONSOLE_MESSAGES
                                                                  ↪ [1])
                                                                  ↪ ;
                                                                uart_tx_string
                                                                  ↪ (
                                                                  ↪ c_uartTX
                                                                  ↪ ,
                                                                  ↪ CONSOLE_MESSAGES
                                                                  ↪ [13])
                                                                  ↪ ;
                                                                break;

                                                        case SET_LED_1:
                                                          ↪ //Read port
                                                          ↪ Value and
                                                          ↪ Set LED 1 ON
                                                                p_led:>
                                                                  ↪ data
                                                                  ↪ ;
```

```
                data=data
                  ↪  | 0
                  ↪ xE;
                p_led<:
                  ↪ data
                  ↪ ;
                break;

        case CLEAR_LED_1
          ↪ :://Read port
          ↪  Value and
          ↪ Set LED 1
          ↪ OFF
                p_led:>
                  ↪ data
                  ↪ ;
                p_led<:
                  ↪ data
                  ↪ &0x1
                  ↪ ;
                break;

        case SET_LED_2:
          ↪ //Read port
          ↪ Value and
          ↪ Set LED 2 ON
                p_led:>
                  ↪ data
                  ↪ ;
                p_led<:
                  ↪ data
                  ↪  | 0
                  ↪ xD;
                break;

        case CLEAR_LED_2:
          ↪  //Read port
          ↪  Value and
          ↪ Set LED 2
          ↪ OFF
                p_led:>
                  ↪ data
                  ↪ ;
                p_led<:
                  ↪ data
                  ↪ &0x2
                  ↪ ;

                break;

        case SET_LED_3:
          ↪ //Read port
          ↪ Value and
          ↪ Set LED 3 ON
                p_led:>
                  ↪ data
                  ↪ ;
                p_led<:
                  ↪ data
                  ↪  | 0
                  ↪ xB;
```

```
            break;

        case CLEAR_LED_3:
          ↪  //Read port
          ↪  Value and
          ↪ Set LED 3
          ↪ OFF
                p_led:>
                  ↪ data
                  ↪ ;
                p_led<:
                  ↪ data
                  ↪ &0x4
                  ↪ ;
                break;

        case SET_LED_4:
          ↪ //Read port
          ↪ Value and
          ↪ Set LED 4 ON
                p_led:>
                  ↪ data
                  ↪ ;
                p_led<:
                  ↪ data
                  ↪  | 0
                  ↪ x7;
                break;

        case CLEAR_LED_4:
          ↪  //Read port
          ↪  Value and
          ↪ Set LED 4
          ↪ OFF
                p_led:>
                  ↪ data
                  ↪ ;
                p_led<:
                  ↪ data
                  ↪ &0x8
                  ↪ ;

                break;

        case CLEAR_ALL:
          ↪ //sets all
          ↪ four LEDs
          ↪ OFF
                p_led<:0
                  ↪ xF;
                break;

        case SET_ALL: //
          ↪ sets all
          ↪ four LEDs ON
                p_led<:0
                  ↪ x0;
                break;

        case
          ↪ BUTTON_PRESSED
```

```
                                                                    ↪ : //Checks
                                                                    ↪ if button is
                                                                    ↪  pressed
                                                                        c_end:>
                                                                          ↪ button
                                                                          ↪ ;
                                                                        if(button
                                                                          ↪  ==
                                                                          ↪ BUTTON_1
                                                                          ↪ ) //
                                                                          ↪ Prints
                                                                          ↪
                                                                          ↪ Button
                                                                          ↪  1
                                                                          ↪ is
                                                                          ↪ pressed
                                                                          ↪  on
                                                                          ↪ the
                                                                          ↪ Uart
                                                                        {
                                                                                CONSOLE_MESSAG
                                                                                  ↪ [4][9]='1
                                                                                  ↪
                                                                                uart_tx_string
                                                                                  ↪ (
                                                                                  ↪ c_uartTX
                                                                                  ↪ ,
                                                                                  ↪ CONSOLE_M
                                                                                  ↪ [4])
                                                                                  ↪ ;
                                                                                  ↪
                                                                                button1_press
                                                                                  ↪ =1;
                                                                                  ↪
                                                                        }
                                                                        if(button
                                                                          ↪  ==
                                                                          ↪ BUTTON_2
                                                                          ↪ ) //
                                                                          ↪ Prints
                                                                          ↪
                                                                          ↪ Button
                                                                          ↪  2
                                                                          ↪ is
                                                                          ↪ pressed
                                                                          ↪  on
                                                                          ↪ Uart
                                                                        {
                                                                                CONSOLE_MESSAG
                                                                                  ↪ [4][9]='2
                                                                                  ↪
                                                                                uart_tx_string
                                                                                  ↪ (
                                                                                  ↪ c_uartTX
                                                                                  ↪ ,
                                                                                  ↪ CONSOLE_M
                                                                                  ↪ [4])
                                                                                  ↪ ;
                                                                                  ↪
```

```
                                                button2_press
                                                  ↪ =1;
                                                  ↪
                        }
                        break;
                case HELP: //
                  ↪ Displays
                  ↪ help
                  ↪ messages on
                  ↪ Uart
                        uart_tx_string
                          ↪ (
                          ↪ c_uartTX
                          ↪ ,
                          ↪ CONSOLE_MESSAGES
                          ↪ [14])
                          ↪ ;
                        uart_tx_string
                          ↪ (
                          ↪ c_uartTX
                          ↪ ,
                          ↪ CONSOLE_MESSAGES
                          ↪ [7])
                          ↪ ;
                        uart_tx_string
                          ↪ (
                          ↪ c_uartTX
                          ↪ ,
                          ↪ CONSOLE_MESSAGES
                          ↪ [8])
                          ↪ ;
                        uart_tx_string
                          ↪ (
                          ↪ c_uartTX
                          ↪ ,
                          ↪ CONSOLE_MESSAGES
                          ↪ [9])
                          ↪ ;
                        uart_tx_string
                          ↪ (
                          ↪ c_uartTX
                          ↪ ,
                          ↪ CONSOLE_MESSAGES
                          ↪ [10])
                          ↪ ;
                        uart_tx_string
                          ↪ (
                          ↪ c_uartTX
                          ↪ ,
                          ↪ CONSOLE_MESSAGES
                          ↪ [15])
                          ↪ ;
                        uart_tx_string
                          ↪ (
                          ↪ c_uartTX
                          ↪ ,
                          ↪ CONSOLE_MESSAGES
                          ↪ [11])
                          ↪ ;
                        uart_tx_send_byte
                          ↪ (
```

```
                                                    ↪ c_uartTX
                                                    ↪ , '\
                                                    ↪ r');
                                                  uart_tx_send_byte
                                                    ↪ (
                                                    ↪ c_uartTX
                                                    ↪ , '\
                                                    ↪ n');
                                                  break;
                                          case READ_ADC: //
                                            ↪ Displays
                                            ↪ temperature
                                            ↪ value on the
                                            ↪  Uart
                                                  adc_value
                                                    ↪ =
                                                    ↪ read_adc_value
                                                    ↪ ();
                                                  data_arr
                                                    ↪ [0]=(
                                                    ↪ linear_interpolat
                                                    ↪ (
                                                    ↪ adc_value
                                                    ↪ ));
                                                  uart_tx_string
                                                    ↪ (
                                                    ↪ c_uartTX
                                                    ↪ ,
                                                    ↪ CONSOLE_MESSAGES
                                                    ↪ [12])
                                                    ↪ ;
                                                  uart_tx_send_byte
                                                    ↪ (
                                                    ↪ c_uartTX
                                                    ↪ , (
                                                    ↪ data_arr
                                                    ↪ [0]/10)
                                                    ↪ +'0')
                                                    ↪ ;
                                                  uart_tx_send_byte
                                                    ↪ (
                                                    ↪ c_uartTX
                                                    ↪ , (
                                                    ↪ data_arr
                                                    ↪ [0]%10)
                                                    ↪ +'0')
                                                    ↪ ;
                                                  uart_tx_send_byte
                                                    ↪ (
                                                    ↪ c_uartTX
                                                    ↪ ,
                                                    ↪ 32);
                                                  uart_tx_send_byte
                                                    ↪ (
                                                    ↪ c_uartTX
                                                    ↪ , 'C
                                                    ↪ ');
                                                  break;
                                          case INVALID: //
                                            ↪ Displays
                                            ↪ command
```

```
                                                                    ↪ input is
                                                                    ↪ invalid
                                                                    ↪ command on
                                                                    ↪ the Uart
                                                                        uart_tx_string
                                                                          ↪ (
                                                                          ↪ c_uartTX
                                                                          ↪ ,
                                                                          ↪ CONSOLE_MESSAGES
                                                                          ↪ [2])
                                                                          ↪ ;
                                                                        break;
                                                                case CHK_BUTTONS:
                                                                  ↪  //Checks if
                                                                  ↪  button are
                                                                  ↪ pressed and
                                                                  ↪ displays on
                                                                  ↪ the Uart
                                                                        if(
                                                                          ↪ button1_press
                                                                          ↪ )
                                                                        {
                                                                                CONSOLE_MESSAG
                                                                                  ↪ [4][9]='1
                                                                                  ↪
                                                                                uart_tx_string
                                                                                  ↪ (
                                                                                  ↪ c_uartTX
                                                                                  ↪ ,
                                                                                  ↪ CONSOLE_M
                                                                                  ↪ [4])
                                                                                  ↪ ;
                                                                                  ↪
                                                                                  ↪ //
                                                                                  ↪ Displays
                                                                                  ↪
                                                                                  ↪ Button
                                                                                  ↪
                                                                                  ↪ 1
                                                                                  ↪
                                                                                  ↪ is
                                                                                  ↪
                                                                                  ↪ pressed
                                                                                  ↪
                                                                        }
                                                                        if(
                                                                          ↪ button2_press
                                                                          ↪ )
                                                                        {
                                                                                CONSOLE_MESSAG
                                                                                  ↪ [4][9]='2
                                                                                  ↪
                                                                                uart_tx_string
                                                                                  ↪ (
                                                                                  ↪ c_uartTX
                                                                                  ↪ ,
                                                                                  ↪ CONSOLE_M
                                                                                  ↪ [4])
                                                                                  ↪ ;
                                                                                  ↪
                                                                                  ↪ //
```

```
                                                                    ↪ Dipslays
                                                                    ↪
                                                                    ↪ Button
                                                                    ↪
                                                                    ↪ 2
                                                                    ↪
                                                                    ↪ is
                                                                    ↪
                                                                    ↪ pressed
                                                                    ↪
                                                            }
                                                            if( !
                                                              ↪ button1_press
                                                              ↪  &&
                                                              ↪ !
                                                              ↪ button2_press
                                                              ↪ )
                                                            {
                                                                    uart_tx_string
                                                                    ↪ (
                                                                    ↪ c_uartTX
                                                                    ↪ ,
                                                                    ↪ CONSOLE_M
                                                                    ↪ [5])
                                                                    ↪ ;
                                                                    ↪
                                                                    ↪ //
                                                                    ↪ Displays
                                                                    ↪
                                                                    ↪ No
                                                                    ↪
                                                                    ↪ Buttons
                                                                    ↪
                                                                    ↪ are
                                                                    ↪
                                                                    ↪ pressed
                                                                    ↪
                                                            }
                                                            button1_press
                                                              ↪ =0;
                                                            button2_press
                                                              ↪ =0;
                                                            break;
                                                    }
                                                    if(data != EXIT) //Exits
                                                      ↪ from command mode
                                                    {
                                                            uart_tx_send_byte
                                                              ↪ (c_uartTX,
                                                              ↪ '\r');
                                                            uart_tx_send_byte
                                                              ↪ (c_uartTX,
                                                              ↪ '\n');
                                                            uart_tx_send_byte
                                                              ↪ (c_uartTX,
                                                              ↪ '>');
                                                    }
                                                    break;
                                            }//select
//::State
                                    }//skip
```

```
                                        j=0;
                                }//command mode
                                break;
                }//main select
```

If the received data is > character the it waits to see if the next received successive bytes are `c`, `m` and `d`. If the successive received data is `>cmd` then the application activates comman mode otherwise the data is echoed back to the Uart Transmit pin. The part of code which explains about the command mode is as blow.

```
if(buffer == '>') //IUF received data is '>' character then expects cmd to endter into
  ↪ command mode
{
        j=0;
        uart_rx_get_byte_byref(c_uartRX, rxState, buffer);
        cmd_rcvbuffer[j]=buffer;
        if((cmd_rcvbuffer[j] == 'C' )|| (cmd_rcvbuffer[j] =='c')) //Checks if received
          ↪ data is 'C' or 'c'
        {
                j++;
                uart_rx_get_byte_byref(c_uartRX, rxState, buffer);
                cmd_rcvbuffer[j]=buffer;

                if((cmd_rcvbuffer[j] == 'm' )|| (cmd_rcvbuffer[j] =='M')) //Checks if
                  ↪ received data is 'M' or 'm'
                {
                        j++;
                        uart_rx_get_byte_byref(c_uartRX, rxState, buffer);
                        cmd_rcvbuffer[j]=buffer;
                        if((cmd_rcvbuffer[j] == 'D' )|| (cmd_rcvbuffer[j] =='d'))//Checks
                          ↪  if received data is 'D' or 'd'
                        {
                                uart_tx_send_byte(c_uartTX, '\r');
                                uart_tx_send_byte(c_uartTX, '\n');
                                uart_tx_string(c_uartTX,CONSOLE_MESSAGES[0]);
                                COMMAND_MODE=1; //activates command mode as received data
                                  ↪  is '>cmd'
                                uart_tx_send_byte(c_uartTX, '\r');
                                uart_tx_send_byte(c_uartTX, '\n');
                                uart_tx_send_byte(c_uartTX, '>'); //displays '>' if
                                  ↪ command mode is activated
                        }
                        else
                        {
                                uart_tx_send_byte(c_uartTX, '>');
                                for(int i=0;i<3;i++)
                                        uart_tx_send_byte(c_uartTX, cmd_rcvbuffer[i]); //
                                          ↪  if received dta is not 'c' displays back
                                          ↪ the received data
                        }
                }
                else
                {
                        uart_tx_send_byte(c_uartTX, '>'); //if received data is not 'm'
                          ↪ displays the received data
                        for(int i=0;i<2;i++)
                                uart_tx_send_byte(c_uartTX, cmd_rcvbuffer[i]);
                }
        }
```

```
        else
        {
                uart_tx_send_byte(c_uartTX, '>');
                uart_tx_send_byte(c_uartTX, cmd_rcvbuffer[j]);
                j=0;
        }
}
else
{
        uart_tx_send_byte(c_uartTX,buffer); //Echoes back the input characters if not in
          ↪ command mode
}
```

After the command mode is active the applicaion receives all the input commands and send to the process_data API using a channel.The part of the code is shown below.

```
select
{
        case uart_rx_get_byte_byref(c_uartRX, rxState, buffer):
                cmd_rcvbuffer[j]=buffer;
                if(cmd_rcvbuffer[j++] == '\r')
                {
                        skip=0;
                        j=0;
                        while(cmd_rcvbuffer[j] != '\r')
                        {
                                c_process<:cmd_rcvbuffer[j]; //received valid command and
                                  ↪  send the command to the process_data theread
                                uart_tx_send_byte(c_uartTX, cmd_rcvbuffer[j]);
                                j++;
                        }
                        cmd_rcvbuffer[j]='\0';
                        c_process<:cmd_rcvbuffer[j];
                        for(int inc=0;inc<20;inc++) //Clears the command buffer
                                cmd_rcvbuffer[inc]='0';
                        j=0;
                }
                break;
```

The process_data thread checks if any button is pressed or checks if there is any command from app_manager thread. If there is button press then the thread sends instructions to the app_manager thread about the button or if command is received, then it send instructions about teh command received. The details in the process_data thread is as shown below.

Process_data thread send instructions to the app_manager thread about the command received. The app_manager thread then implementys the state machine according to the instructions received from the process_data thread. The state machine of app_manager thread is as below.

3.3 GPIO Ethernet Combo Demo

3.3.1 Structure

All the files required for operation are located in the app_sk_gpio_eth_combo_demo/src directory. The files to be included for use of the dependent components are:

File	Description
ethernet_board_support.h	Defines OTP and ethernet pins required for Ethernet component
xtcp.h	Header file for xtcp API interface
web_server.h	Header file for web server API interface in order to use web pages
i2c.h	Defines i2c pins and API interface to use i2c master component for GPIO adc interfacing
app_handler.h	Application specific defines and API interface to implement demo functionality

3.3.2 API


doxygenfunction: Cannot find function “app_handler” in doxygen xml output


doxygenfunction: Cannot find function “process_web_page_data” in doxygen xml output


doxygenfunction: Cannot find function “get_web_user_selection” in doxygen xml output

3.3.3 Usage and Implementation

- Ethernet uses ethernet_board_support.h configuration
- I2C uses 1 bit port for SCL(I2C Clock) and SDA (I2C data)
- LEDs and Buttons uses 4 bit ports

The app_manager API writes the configuration settings to the ADC as shows below

The select statement in the app_handler API selects either I/O events to check for any valid button presses or uses channel events to detect any commands from web page requests. Valid web page commands are either to set or reset LEDs and check button press state since last check request.

Whenever there is a change in values of the button ports, a flag is used to kick start a timer for debounce interval, and port value is sampled to identify which button is pressed. The code is as shown below

Every time a web page request is received, app_handler records current ADC value and button press status and sends it to web page. Button check statuses are reset immediately after the web page request.

3.3.4 Using sc_website to build web page for this demo application

Makefile in app_sk_gpio_eth_combo_demo folder should include the following line:

WEBFS_TYPE = internal

This value indicates sc_website component to use program memory instead of FLASH memory to store the web pages.

As a next step, create `web` folder in app_sk_gpio_eth_combo_demo

In order to include any images to be displayed on the web page, create images folder as follows app_sk_gpio_eth_combo_demo/web/images

For this application, we have created index.html web page using html script. This page uses XMOS logo from images folder. We have defined desired GPIO controls for LEDs in this web page.

APIs that are required to be executed by the demo application should be enclosed between the tags {% and %}. These are to be defined in `web_page_functions.c` file. For example, index.html contains {% read_temperature(buf, app_state, connection_state) %}

This indicates `read_temperature` is a function executed by the program and result is returned to the web page. After execution, sc_website component replaces this function as "Temperature recorded from onboard ADC: NA ^oC"

3.4 GPIO Wi-Fi Combo Demo

3.4.1 Structure

All the files required for operation are located in the app_sk_gpio_wifi_tiwisl_combo_demo/src directory. The files to be included for use of the dependent components are:

File	Description
`wifi_tiwisl_server.h`	Header file for Wi-Fi API interface
`web_server.h`	Header file for web server API interface in order to use web pages
`i2c.h`	Defines i2c pins and API interface to use i2c master component for GPIO adc interfacing
`app_handler.h`	Application specific defines and API interface to implement demo functionality

3.4.2 API


doxygenfunction: Cannot find function "app_handler" in doxygen xml output


doxygenfunction: Cannot find function "process_web_page_data" in doxygen xml output


doxygenfunction: Cannot find function "get_web_user_selection" in doxygen xml output

3.4.3 Usage and Implementation

- 1 four-bit port for nCS(Chip Select) and Power enable
- 1 one-bit port for nIRQ(interrupt)
- SPI uses 3 one-bit ports for MOSI, CLK and MISO

- I2C uses 1 bit port for SCL(I2C Clock) and SDA (I2C data)
- LEDs and Buttons use 4 bit ports

The app_manager API writes the configuration settings to the ADC as shows below

The select statement in the app_handler API selects either I/O events to check for any valid button presses or uses channel events to detect any commands from web page requests. Valid web page commands are either to set or reset LEDs and check button press state since last check request.

Whenever there is a change in values of the button ports, a flag is used to kick start a timer for debounce interval, and port value is sampled to identify which button is pressed. The code is as shown below

Every time a web page request is received, app_handler records current ADC value and button press status and sends it to web page. Button check statuses are reset immediately after the web page request.

3.4.4 Using sc_website to build web page for this demo application

Makefile in app_sk_gpio_wifi_tiwisl_combo_demo folder should include the following line:

WEBFS_TYPE = internal

This value indicates sc_website component to use program memory instead of FLASH memory to store the web pages.

As a next step, create `web` folder in app_sk_gpio_wifi_tiwisl_combo_demo

In order to include any images to be displayed on the web page, create images folder as follows app_sk_gpio_wifi_tiwisl_combo_demo/web/images

For this application, we have created index.html web page using html script. We have defined desired GPIO controls for LEDs in this web page.

APIs that are required to be executed by the demo application should be enclosed between the tags {% and %}. These are to be defined in `web_page_functions.c` file. For example, index.html contains {% read_temperature(buf, app_state, connection_state) %}

This indicates `read_temperature` is a function executed by the program and result is returned to the web page. After execution, sc_website component replaces this function as "Temperature recorded from on-board ADC: NA ^oC"