Final Project - Digi Clock

(Topics in Adv Comp Eng: Embedded Systems Hardware EE 493)

NAME: Prince K. Bose
NET ID: pkb44
RUID: 186008149

DATE: 5/7/19

Purpose

The main purpose behind making the Digi Clock, was to get temperature, time, date and day information onto a single display at the click of a button. This was implemented using the techniques and knowledge of Vivado, VHDL and C taught in the class.

The main components of the circuit are:

1. Zybo Board AXI interface (GPIO connections)
2. OLED Pmod
3. RTCC Pmod
4. TMP3 Pmod

There are three modes in the Digi Clock.

1. **Digi Clock Mode:** shows the current time in hh:mm:ss format
2. **Temperature Mode:** shows the current temperature in Fahrenheit and Celsius
3. **Calendar Mode:** shows the current day of the week and the date in mm/dd/yyyy format

RTCC PMOD

RTCC stands for Real Time Clock and Calendar. This module keeps track of updated current time of the system. This information can be read by a microprocessor, usually over a serial interface to facilitate the software performing functions that are time dependent. RTCs are designed for ultra-low power consumption as they usually continue running when the main system is powered down. This enables them to maintain current time against an absolute time reference, usually set by the microprocessor directly.

They usually interface to a microprocessor circuit by an SPI or I^2C serial bus, and may contain a number of other functions like backup memory, a watchdog timer for supervising the microprocessor and countdown timers to generate real time event. Some RTCs include second or minute interrupt outputs and are even clever enough to account for leap years.

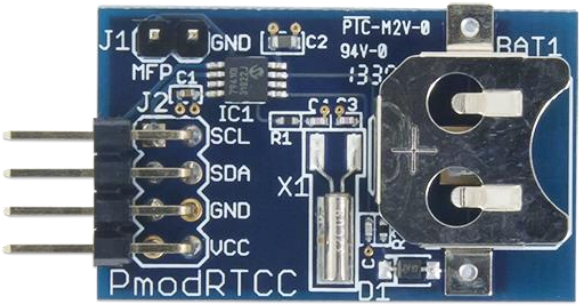

Figure: RTCC Peripheral Module

We connect the RTCC over the I^2C interface. The I^2C interface is a serial protocol for two-wire interface to connect low-speed devices like microcontrollers, EEPROMs, A/D and D/A converters, I/O interfaces and other similar peripherals in embedded systems. It was invented by Philips and now it is used by almost all major IC manufacturers. Each I2C slave device needs an address. This protocol's simplicity of usage makes it common in the industry.

RTCC is connected to the Connection Port JC on the Zybo.

TMP3 PMOD

TMP3 is a temperature sensor module. This is built around the Microchip TCN75A. The resolution is of 12 bits. It is also capable of setting a trigger upon crossing a user defined threshold.

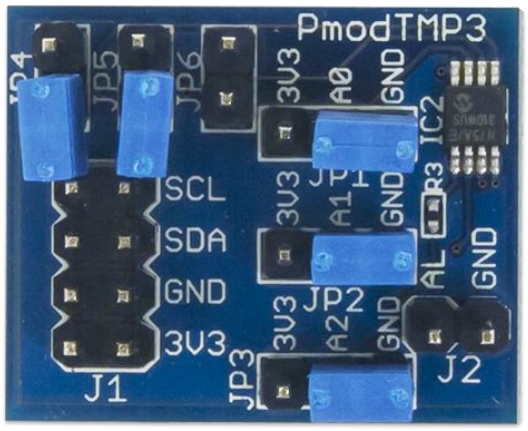

Figure: TMP3 Peripheral Module

We connect the TMP3 over the I^2C interface. The I^2C interface is a serial protocol for two-wire interface to connect low-speed devices like microcontrollers, EEPROMs, A/D and D/A converters, I/O interfaces and other similar peripherals in embedded systems. It was invented by Philips and now it is used by almost all major IC manufacturers. Each I2C slave device needs an address. This protocol's simplicity of usage makes it common in the industry.

TMP3 is connected to the Connection Port JA on the Zybo, since it is connected to an ADC.

OLED PMOD

The Pmod OLED is 128x32 pixel monochrome organic LED (OLED) panel powered by the Solomon Systech SSD1306. Users can display any sort of graphical design by programming the device through SPI as well as sending bitmap images.
The resolution of the OLED display is 128×32 pixels. It has an internal display buffer. There is a 12-pin PMOD connector with SPI interface to connect to the Zybo board.

Figure: OLED Peripheral Module

SPI devices communicate in full duplex mode using a master-slave architecture with a single master. The master device originates the frame for reading and writing. Multiple slave-devices are supported through selection with individual slave select (SS) (sometimes called chip select (CS)) lines.

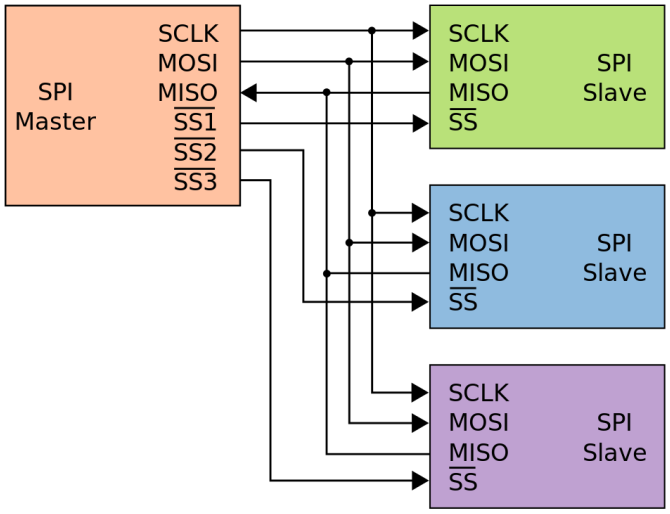

Figure: An example SPI peripheral interface

Connection Overview

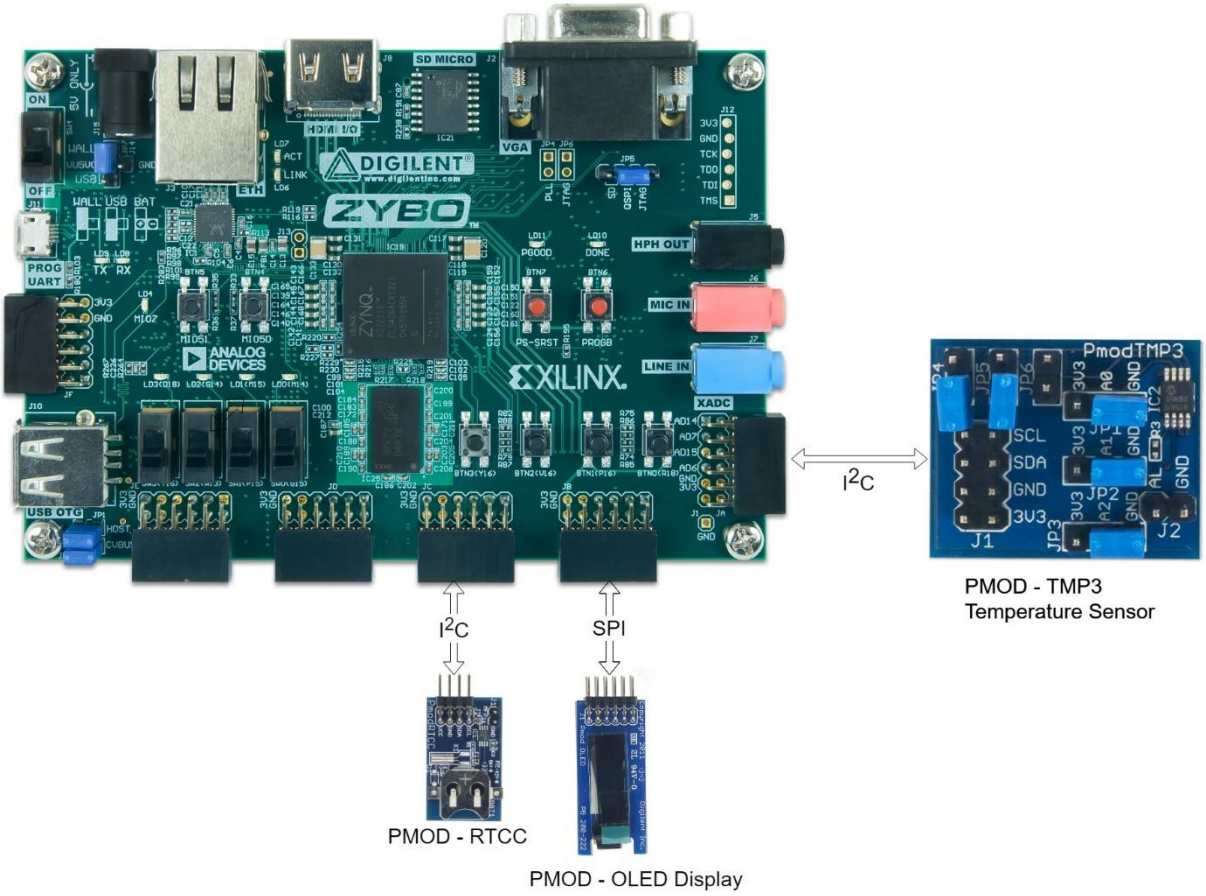

Figure: Connection Interface

To connect all the PMODs to the Zybo, we import the IP blocks in the block Block Design Integrator. Then we run the Connection Automation and the Block Automation.
Since, TMP3 and RTCC are interfaced via the I^2C interface, these PMODs generate an Interrupt for the Zynq Processing Block. Since the Zynq has only one IRQ pin, we concat both the interrupts into the Concat block and provide the IRQ pin with a 2 bit signal.
Now we generate the bitstream and build an application project on top of the Board Support Package in the Xilinx SDK.

Source Code in C for the above generated BSP:

```
/*
 * Prince Bose
 * pkb44
 * ECE 544 Embedded Systems - Final Project
 * Digital Clock
 * This project needs one Zybo board, 3 PMODs - OLED, RTCC, TMP3
 * This clock has 3 modes, that can be controlled at the click of a button.
 * BTN0 switches between the 3 modes - Digi Clock Mode, Temperature Mode, Calender
Mode
 *
 * Port JA - TMP3
 * Port JB - OLED
 * Port JC - RTCC
 *
 * Digi Clock Mode:
 * Shows the current time in hh:mm:ss AM/PM format
 *
 * Temperature Mode:
 * Shows temperatures in both Farenheit and Celcius
 *
 * Calender Mode:
 * Shows the current day, and the date in mm/dd/yy format
 *
 * */


/*************************** Include Files ******************************/

#include "PmodRTCC.h"
#include "PmodOLED.h"
#include "PmodTMP3.h"
#include <xgpio.h>

#include "sleep.h"
#include "xil_cache.h"
#include "xparameters.h"
#include <stdio.h>

#include "xil_printf.h"
#include "xparameters.h"

/************************** Type Declarations ***************************/

// Struct containing each field of the RTCC's time registers represented in
// 8-bit binary coded decimal - 0x30 in the minute field represents 30 minutes.
typedef struct RTCC_Time {
   u8 second;
   u8 minute;
   u8 hour;
   u8 ampm;
   u8 day;
   u8 date;
```

```
   u8 month;
   u8 year;
} RTCC_Time;

/************************** Global Declarations ****************************/

// Which weekday starts this array is arbitrary, as long as it stays the same
// when you set and read the day
const char *weekdays[7] = {
   "Monday",
   "Tuesday",
   "Wednesday",
   "Thursday",
   "Friday",
   "Saturday",
   "Sunday"
};

XGpio input, output;
int button_data = 0;
int ModeSelect = 1;

// If RTCC is already set, change 1 to 0
#define SET_RTCC 1

PmodRTCC RTCCDEVICE;
PmodOLED OLEDDEVICE;
PmodTMP3 TEMPDEVICE;

// To change between PmodOLED and OnBoardOLED is to change Orientation
const u8 orientation = 0x0; // Set up for Normal PmodOLED(false) vs normal
                            // Onboard OLED(true)
const u8 invert = 0x0; // true = whitebackground/black letters
                       // false = black background /white letters
/************************** Function Declarations **************************/

// Core demo functions
void DigiClockMode();
void DemoInitialize(u8 mode);
void DemoCleanup();
void EnableCaches();
void DisableCaches();

// Additional demo functions to manage the driver
RTCC_Time GetTime(PmodRTCC *InstancePtr, RTCC_Target src);
RTCC_Time IncrementTime(RTCC_Time time, int delta_seconds);
void SetTime(PmodRTCC *InstancePtr, RTCC_Target dest, RTCC_Time val);
void PrintTime(RTCC_Target src);
u8 bcd2int(u8 data);
u8 int2bcd(u8 data);

/************************** Function Definitions ***************************/

int main() {
```

```
	XGpio_Initialize(&input, XPAR_AXI_GPIO_0_DEVICE_ID); //initialize input XGpio
variable
	XGpio_Initialize(&output, XPAR_AXI_GPIO_1_DEVICE_ID);	//initialize output
XGpio variable

	XGpio_SetDataDirection(&input, 1, 0xF);		//set first channel
tristate buffer to input
	XGpio_SetDataDirection(&input, 2, 0xF);		//set second channel
tristate buffer to input

	XGpio_SetDataDirection(&output, 1, 0x0);		//set first channel
tristate buffer to output
	init_platform();
	EnableCaches();
	xil_printf("\x1B[H");  // Move terminal cursor to top left
	xil_printf("\x1B[1K"); // Clear terminal
	xil_printf("Connected to PmodTMP3 Demo over UART\n\r");
	TMP3_begin(&TEMPDEVICE, XPAR_PMODTMP3_0_AXI_LITE_IIC_BASEADDR, TMP3_ADDR);
	xil_printf("Connected to PmodTMP3 over IIC on JA\n\r\n\r");
	DemoInitialize(SET_RTCC);
	xil_printf("Connected to RTCC\n\r\n\r");
	DigiClockMode();

	DemoCleanup();
	cleanup_platform();
	return 0;
}


void DemoInitialize(u8 mode) {
   RTCC_Time time;
   OLED_Begin(&OLEDDEVICE,
XPAR_PMODOLED_0_AXI_LITE_GPIO_BASEADDR,XPAR_PMODOLED_0_AXI_LITE_SPI_BASEADDR,
orientation, invert);
   EnableCaches();

   RTCC_begin(&RTCCDEVICE, XPAR_PMODRTCC_0_AXI_LITE_IIC_BASEADDR, 0x6F);

   // Print the power-fail time-stamp
   xil_printf("Lost Power at: ");
   PrintTime(RTCC_TARGET_PWRD);
   xil_printf("\r\n");

   xil_printf("Power was back at: ");
   PrintTime(RTCC_TARGET_PWRU);
   xil_printf("\r\n");

   if (!RTCC_checkVbat(&RTCCDEVICE) || mode) {
      // Set the real time clock to Tuesday 2/6/18 12:24:36 PM
      RTCC_stopClock(&RTCCDEVICE);

      time.second = 0x00;
      time.minute = 0x30;
      time.hour   = 0x06;
      time.ampm   = RTCC_PM;
```

```
    time.day    = 0x03;
    time.date   = 0x09;
    time.month  = 0x05;
    time.year   = 0x19;

    time = IncrementTime(time, 0); // TEST
    SetTime(&RTCCDEVICE, RTCC_TARGET_RTCC, time);

    RTCC_startClock(&RTCCDEVICE);
    xil_printf("The time has been set \r\n");
    // Set vbat high
    RTCC_enableVbat(&RTCCDEVICE);
} else {
    time = GetTime(&RTCCDEVICE, RTCC_TARGET_RTCC);
}

// Sset alarm 0 for 30 seconds from now
time = IncrementTime(time, 30);
SetTime(&RTCCDEVICE, RTCC_TARGET_ALM0, time);

// Sset alarm 1 for 1 minute from now
time = IncrementTime(time, 30);
SetTime(&RTCCDEVICE, RTCC_TARGET_ALM1, time);

// Pprint current time
xil_printf("Current time is: ");
PrintTime(RTCC_TARGET_RTCC);
xil_printf("\r\n");

// Print alarm 0
xil_printf("Alarm 0 is set to : ");
PrintTime(RTCC_TARGET_ALM0);
xil_printf("\r\n");

// Print alarm 1
xil_printf("Alarm 1 is set to : ");
PrintTime(RTCC_TARGET_ALM1);
xil_printf("\r\n");

// Enables alarm 0
// Set configuration bits to:
//    RTCC_ALM_POL | RTCC_ALMC2 | RTCC_ALMC1 | RTCC_ALMC0
// This will drive the MPF pin high when the alarm triggered
// It also sets the alarm to be triggered when the alarm matches
// Seconds, Minutes, Hour, Day, Date, Month of the RTCC
RTCC_enableAlarm(&RTCCDEVICE, RTCC_TARGET_ALM0,
      RTCC_ALM_POL | RTCC_ALMC2 | RTCC_ALMC1 | RTCC_ALMC0);

// Enable alarm 1
// Set configuration bits to RTCC_ALM_POL
// This will drive the MPF pin high when the alarm triggered
// It also sets the alarm to be triggered when the alarm matches
// Seconds of the RTCC
RTCC_enableAlarm(&RTCCDEVICE, RTCC_TARGET_ALM1,
      RTCC_ALM_POL | RTCC_ALMC2 | RTCC_ALMC1 | RTCC_ALMC0);
```

```
   // Enable back up battery
   RTCC_enableVbat(&RTCCDEVICE);

   RTCC_clearPWRFAIL(&RTCCDEVICE);
}

void DigiClockMode() {

       int x = 1;
       RTCC_Time time1;
       time1 = GetTime(&RTCCDEVICE, RTCC_TARGET_RTCC);
       OLED_ClearBuffer(&OLEDDEVICE);
       OLED_SetCursor(&OLEDDEVICE, 0, 0);
       OLED_PutString(&OLEDDEVICE, " Digital Clock ");
       OLED_SetCursor(&OLEDDEVICE, 0, 2);
       OLED_PutString(&OLEDDEVICE, " -By Prince Bose ");
       OLED_SetCursor(&OLEDDEVICE, 0, 3);
       OLED_PutString(&OLEDDEVICE, "          (pkb44) ");
       sleep(2);
       while (1)
       {
          OLED_ClearBuffer(&OLEDDEVICE);
          button_data = XGpio_DiscreteRead(&input, 1);
          if(button_data == 0b0001){ ModeSelect = (ModeSelect + 1)%3;
xil_printf("\n\rBUTTON PRESSED MODE CHANGE Mode=%d",ModeSelect);}
          if (ModeSelect == 1)
          {
          OLED_SetCursor(&OLEDDEVICE, 0, 0);
          OLED_PutString(&OLEDDEVICE, " Digi Clock Mode ");
         xil_printf("\r\nCurrent time is : ");
         PrintTime(RTCC_TARGET_RTCC);
         OLED_SetCursor(&OLEDDEVICE, 0, 2);

         int my_hour_LSB = bcd2int(time1.hour);
         int my_hour_MSB = 0;
         int my_min_LSB = bcd2int (time1.minute);
         int my_min_MSB = 0;
         int my_sec_LSB = bcd2int(time1.second);
         int my_sec_MSB = 0;

         if(my_hour_LSB > 9)
         {
                my_hour_MSB = floor(my_hour_LSB/10);
                my_hour_LSB = my_hour_LSB % 10;
                time1.ampm = RTCC_PM;
         }

         if(my_min_LSB > 9)
         {

              my_min_MSB = floor(my_min_LSB/10);
              my_min_LSB = my_min_LSB % 10;
         }
```

```
        if(my_sec_LSB > 9)
        {
             my_sec_MSB = floor(my_sec_LSB/10);
             my_sec_LSB = my_sec_LSB % 10;
        }

        my_hour_LSB = my_hour_LSB + 48;
        my_hour_MSB = my_hour_MSB + 48;

        my_min_LSB = my_min_LSB + 48;
        my_min_MSB = my_min_MSB + 48;

        my_sec_LSB = my_sec_LSB + 48;
        my_sec_MSB = my_sec_MSB + 48;


        OLED_PutChar(&OLEDDEVICE, my_hour_MSB);
        OLED_PutChar(&OLEDDEVICE, my_hour_LSB);

        x++;
        if (x%2==0) OLED_PutChar(&OLEDDEVICE, ':');
        else OLED_PutChar(&OLEDDEVICE, ' ');



        OLED_PutChar(&OLEDDEVICE, my_min_MSB);
        OLED_PutChar(&OLEDDEVICE, my_min_LSB);
        if (x%2==0) OLED_PutChar(&OLEDDEVICE, ':');
                    else OLED_PutChar(&OLEDDEVICE, ' ');
        OLED_PutChar(&OLEDDEVICE, my_sec_MSB);
        OLED_PutChar(&OLEDDEVICE, my_sec_LSB);
        OLED_PutChar(&OLEDDEVICE, ' ');

        if(time1.ampm)
        {
              OLED_PutString(&OLEDDEVICE, " PM");
        }
        else
        {
              OLED_PutString(&OLEDDEVICE, " AM");
        }
        OLED_Update(&OLEDDEVICE);
        time1 = IncrementTime(time1,1);
        usleep(1000000);
         }
         else if (ModeSelect == 2)//TEMPERATURE MODE
         {

                    OLED_ClearBuffer(&OLEDDEVICE);
                    OLED_SetCursor(&OLEDDEVICE,0,0);
                    OLED_PutString(&OLEDDEVICE,"Temperature Mode");
                    time1 = IncrementTime(time1,1); //invariantly keep increasing
time
                    double temp, temp2, temp3 = 0.0;
```

```
temp  = TMP3_getTemp(&TEMPDEVICE);
temp2 = TMP3_CtoF(temp);
temp3 = TMP3_FtoC(temp2);

int temp2_round = 0;
int temp2_int   = 0;
int temp2_frac  = 0;
// Round to nearest hundredth, multiply by 100
if (temp2 < 0) {
temp2_round = (int) (temp2 * 1000 - 5) / 10;
temp2_frac  = -temp2_round % 100;
} else {
temp2_round = (int) (temp2 * 1000 + 5) / 10;
temp2_frac  = temp2_round % 100;
}
temp2_int = temp2_round / 100;

int temp3_round = 0;
int temp3_int   = 0;
int temp3_frac  = 0;
if (temp3 < 0) {
temp3_round = (int) (temp3 * 1000 - 5) / 10;
temp3_frac  = -temp3_round % 100;
} else {
temp3_round = (int) (temp3 * 1000 + 5) / 10;
temp3_frac  = temp3_round % 100;
}
temp3_int = temp3_round / 100;

//TEMP 2 into ascii
int temp_2_int_LSB = bcd2int(temp2_int);
int temp_2_int_MSB = 0;
if(temp_2_int_LSB > 9)
{
      temp_2_int_MSB = floor(temp_2_int_LSB/10);
      temp_2_int_LSB = temp_2_int_LSB % 10;
}
temp_2_int_MSB = temp_2_int_MSB + 48;
temp_2_int_LSB = temp_2_int_LSB + 48;

int temp_2_frac_LSB = bcd2int(temp2_frac);
int temp_2_frac_MSB = 0;
if(temp_2_frac_LSB > 9)
{
      temp_2_frac_MSB = floor(temp_2_frac_LSB/10);
      temp_2_frac_LSB = temp_2_frac_LSB % 10;
}
temp_2_frac_MSB = temp_2_frac_MSB + 48;
temp_2_frac_LSB = temp_2_frac_LSB + 48;



//TEMP 3 into ascii
int temp_3_int_LSB = bcd2int(temp3_int);
int temp_3_int_MSB = 0;
```

```
                        if(temp_3_int_LSB > 9)
                        {
                               temp_3_int_MSB = floor(temp_3_int_LSB/10);
                               temp_3_int_LSB = temp_3_int_LSB % 10;
                        }
                        temp_3_int_MSB = temp_3_int_MSB + 48;
                        temp_3_int_LSB = temp_3_int_LSB + 48;

                        int temp_3_frac_LSB = bcd2int(temp3_frac);
                        int temp_3_frac_MSB = 0;
                        if(temp_3_frac_LSB > 9)
                        {
                               temp_3_frac_MSB = floor(temp_3_frac_LSB/10);
                               temp_3_frac_LSB = temp_3_frac_LSB % 10;
                        }
                        temp_3_frac_MSB = temp_3_frac_MSB + 48;
                        temp_3_frac_LSB = temp_3_frac_LSB + 48;




                        xil_printf("Temperature: %d.%d in Fahrenheit\n\r", temp2_int,
temp2_frac);
                        xil_printf("Temperature: %d.%d in Celsius\n\r", temp3_int,
temp3_frac);


                        OLED_SetCursor(&OLEDDEVICE,0,2);

                        OLED_PutChar(&OLEDDEVICE,temp_2_int_MSB);
                        OLED_PutChar(&OLEDDEVICE,temp_2_int_LSB);
                        OLED_PutString(&OLEDDEVICE,".");
                        OLED_PutChar(&OLEDDEVICE,temp_2_frac_MSB);
                        OLED_PutChar(&OLEDDEVICE,temp_2_frac_LSB);
                        OLED_PutString(&OLEDDEVICE," F");

                        OLED_SetCursor(&OLEDDEVICE,0,3);


                        OLED_PutChar(&OLEDDEVICE,temp_3_int_MSB);
                        OLED_PutChar(&OLEDDEVICE,temp_3_int_LSB);
                        OLED_PutString(&OLEDDEVICE,".");
                        OLED_PutChar(&OLEDDEVICE,temp_3_frac_MSB);
                        OLED_PutChar(&OLEDDEVICE,temp_3_frac_LSB);
                        OLED_PutString(&OLEDDEVICE," C");

                        OLED_Update(&OLEDDEVICE);
                        print("\n\r");


                usleep(1000000);

        }
        else
```

```
		{

		time1 = IncrementTime(time1,1); //invariantly keep increasing time
		OLED_ClearBuffer(&OLEDDEVICE);
		OLED_SetCursor(&OLEDDEVICE,0,0);
		OLED_PutString(&OLEDDEVICE,"Calender Mode");

		OLED_SetCursor(&OLEDDEVICE,0,2);
		OLED_PutString(&OLEDDEVICE,weekdays[time1.day]);
		OLED_SetCursor(&OLEDDEVICE,0,3);
		int month_int_LSB = bcd2int (time1.month);
		int month_int_MSB = 0;

		if (month_int_LSB>9)
		{
			month_int_MSB = floor(month_int_LSB/10);
			month_int_LSB = month_int_LSB % 10;
		}
		month_int_MSB = month_int_MSB + 48;
		month_int_LSB = month_int_LSB + 48;

		OLED_PutChar(&OLEDDEVICE, month_int_MSB);
		OLED_PutChar(&OLEDDEVICE, month_int_LSB);

		OLED_PutChar(&OLEDDEVICE, '/');
		int date_int_LSB = bcd2int (time1.date);
		int date_int_MSB = 0;

		if (date_int_LSB>9)
		{
			date_int_MSB = floor(date_int_LSB/10);
			date_int_LSB = date_int_LSB % 10;
		}
		date_int_MSB = date_int_MSB + 48;
		date_int_LSB = date_int_LSB + 48;
		OLED_PutChar(&OLEDDEVICE, date_int_MSB);
		OLED_PutChar(&OLEDDEVICE, date_int_LSB);


		OLED_PutChar(&OLEDDEVICE, '/');
		int year_int_LSB = bcd2int (time1.year);
		int year_int_MSB = 0;

		if (year_int_LSB>9)
		{
			year_int_MSB = floor(year_int_LSB/10);
			year_int_LSB = year_int_LSB % 10;
		}
		year_int_MSB = year_int_MSB + 48;
		year_int_LSB = year_int_LSB + 48;
		OLED_PutChar(&OLEDDEVICE, year_int_MSB);
		OLED_PutChar(&OLEDDEVICE, year_int_LSB);

		xil_printf("%s %x/%x/%02x", weekdays[time1.day], time1.month,
time1.date, time1.year);
```

```
            print("\n\r");

            OLED_Update(&OLEDDEVICE);
            usleep(1000000);
         }
            // Check if alarm 0 is triggered
        if (RTCC_checkFlag(&RTCCDEVICE, RTCC_TARGET_ALM0)) {
             // Alarm 0 has been triggered
             xil_printf("ALARM 0!!!");
             // Disable alarm 0
             RTCC_disableAlarm(&RTCCDEVICE, RTCC_TARGET_ALM0);
             xil_printf("\r\n");
        }

        // Check if alarm 1 is triggered
        if (RTCC_checkFlag(&RTCCDEVICE, RTCC_TARGET_ALM1)) {
             // Alarm 1 has been triggered
             xil_printf("ALARM 1!!!");
             // Disable alarm
             RTCC_disableAlarm(&RTCCDEVICE, RTCC_TARGET_ALM1);
             xil_printf("\r\n");
        }
      }
}

RTCC_Time GetTime(PmodRTCC *InstancePtr, RTCC_Target src) {
   RTCC_Time val;

   if (src != RTCC_TARGET_PWRD && src != RTCC_TARGET_PWRU) {
      val.second = RTCC_getSec(&RTCCDEVICE, src);
   }

   val.minute = RTCC_getMin(&RTCCDEVICE, src);
   val.hour   = RTCC_getHour(&RTCCDEVICE, src);
   val.ampm   = RTCC_getAmPm(&RTCCDEVICE, src);
   val.day    = RTCC_getDay(&RTCCDEVICE, src);
   val.date   = RTCC_getDate(&RTCCDEVICE, src);
   val.month  = RTCC_getMonth(&RTCCDEVICE, src);

   if (src == RTCC_TARGET_RTCC) {
      val.year = RTCC_getYear(&RTCCDEVICE);
   } else {
      val.year = 0;
   }

   return val;
}

void SetTime(PmodRTCC *InstancePtr, RTCC_Target dest, RTCC_Time val) {
   if (dest != RTCC_TARGET_PWRD && dest != RTCC_TARGET_PWRU) {
      RTCC_setSec(&RTCCDEVICE, dest, val.second);
   }

   RTCC_setMin(&RTCCDEVICE, dest, val.minute);
   RTCC_setHour12(&RTCCDEVICE, dest, val.hour, val.ampm);
```

```
   RTCC_setDay(&RTCCDEVICE, dest, val.day);
   RTCC_setDate(&RTCCDEVICE, dest, val.date);
   RTCC_setMonth(&RTCCDEVICE, dest, val.month);

   if (dest == RTCC_TARGET_RTCC) {
      RTCC_setYear(&RTCCDEVICE, val.year);
   }
}

void PrintTime(RTCC_Target src) {
   RTCC_Time time;

   // Fetch the time from the device
   time = GetTime(&RTCCDEVICE, src);

   xil_printf("%s %x/%x", weekdays[time.day], time.month, time.date);

   // Year is only available for the RTCC
   if (src == RTCC_TARGET_RTCC) {
      xil_printf("/%02x", time.year);
   }

   xil_printf(" %x:%02x", time.hour, time.minute);

   // Second is not supported by the power fail registers
   if (src != RTCC_TARGET_PWRD && src != RTCC_TARGET_PWRU) {
      xil_printf(":%02x", time.second);
   }

   if (time.ampm) {
      xil_printf(" PM");
   } else {
      xil_printf(" AM");
   }
}

RTCC_Time IncrementTime(RTCC_Time time, int delta_seconds) {
   RTCC_Time result;
   int temp;
   result = time;
   temp = bcd2int(result.second) + delta_seconds;
   result.second = int2bcd(temp % 60);          // Convert seconds
   temp = bcd2int(result.minute) + temp / 60;   // Carry seconds -> minutes
   result.minute = int2bcd(temp % 60);          // Convert minutes
   temp = bcd2int(result.hour) + temp / 60 - 1; // Carry minutes -> hours
   result.hour = int2bcd((temp % 12) + 1);      // Convert hours
   return result;
}

u8 bcd2int(u8 data) {
   return ((data >> 4) * 10) + (data & 0xF);
}

u8 int2bcd(u8 data) {
   return (((data / 10) & 0xF) << 4) + ((data % 10) & 0xF);
```

```
}

void DemoCleanup() {
   DisableCaches();
}

void EnableCaches() {
#ifdef __MICROBLAZE__
#ifdef XPAR_MICROBLAZE_USE_ICACHE
   Xil_ICacheEnable();
#endif
#ifdef XPAR_MICROBLAZE_USE_DCACHE
   Xil_DCacheEnable();
#endif
#endif
}

void DisableCaches() {
#ifdef __MICROBLAZE__
#ifdef XPAR_MICROBLAZE_USE_DCACHE
   Xil_DCacheDisable();
#endif
#ifdef XPAR_MICROBLAZE_USE_ICACHE
   Xil_ICacheDisable();
#endif
#endif
}
```

BLOCK DESIGN:

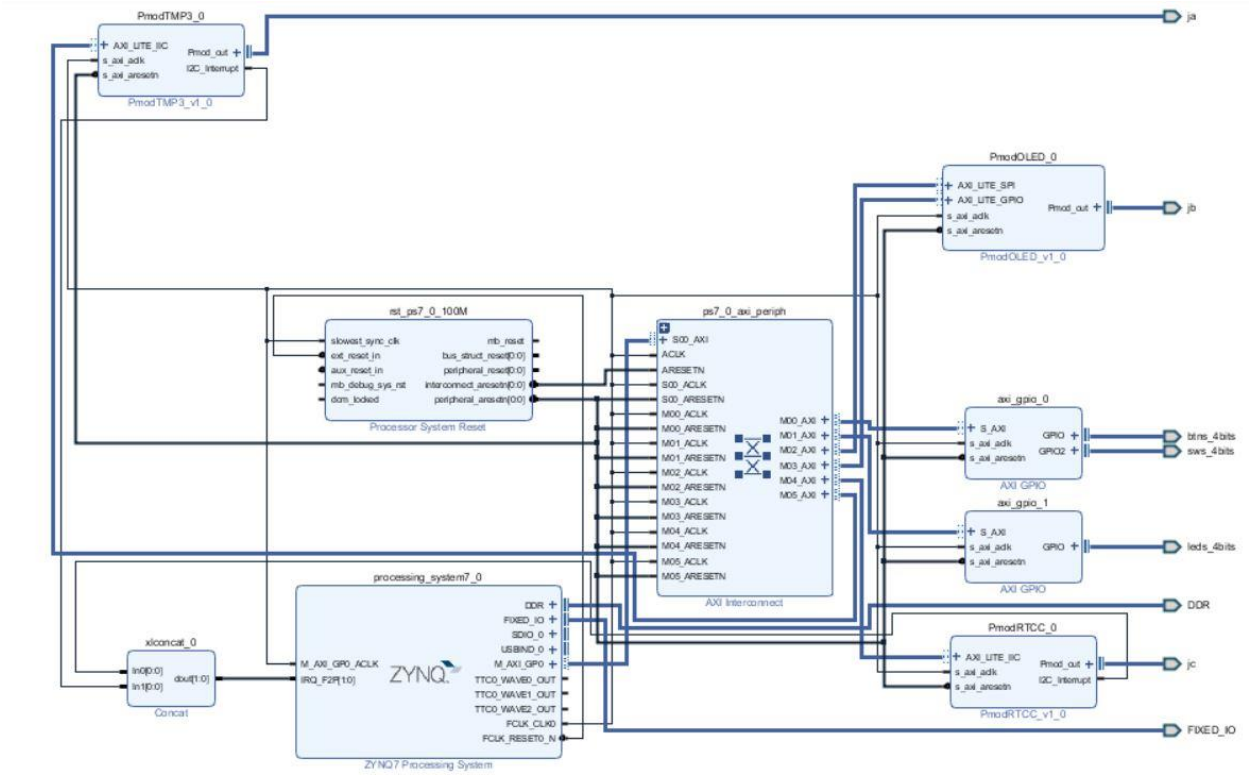

Figure: Block Design

SYNTHESIZED DESIGN:

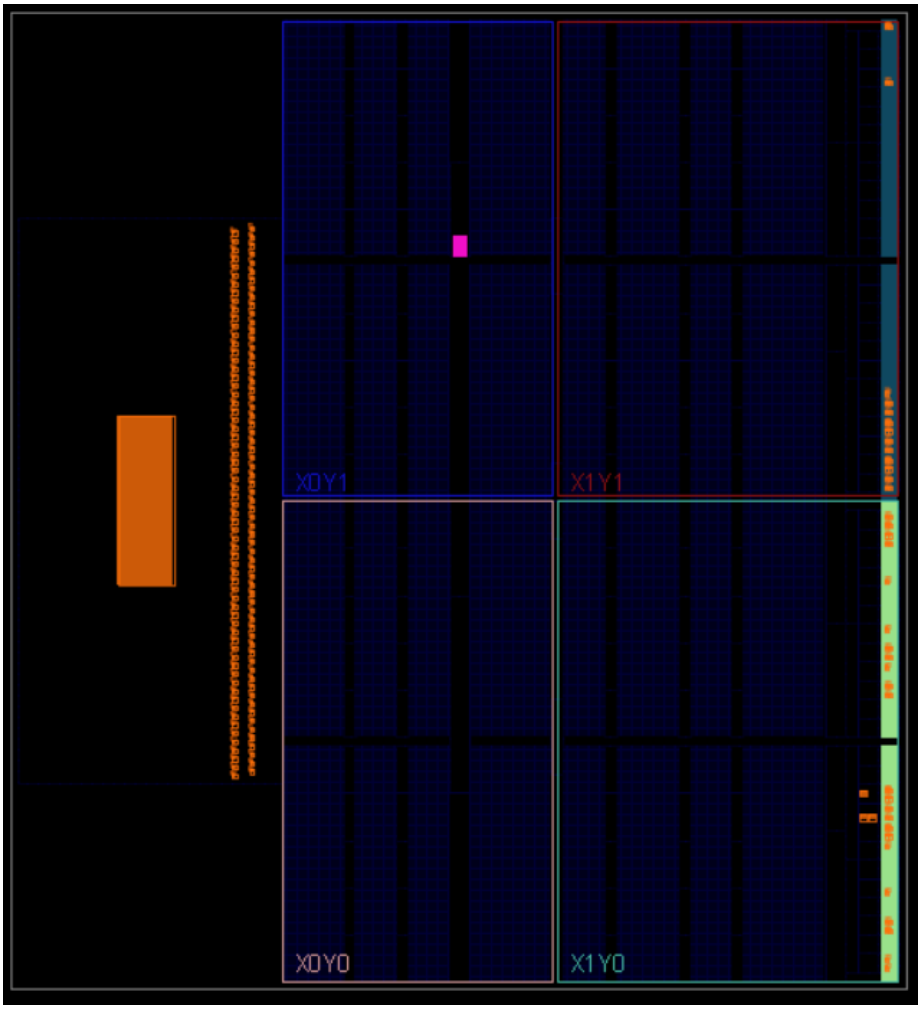

Figure: Synthesized Netlist

IMPLEMENTED DESIGN:

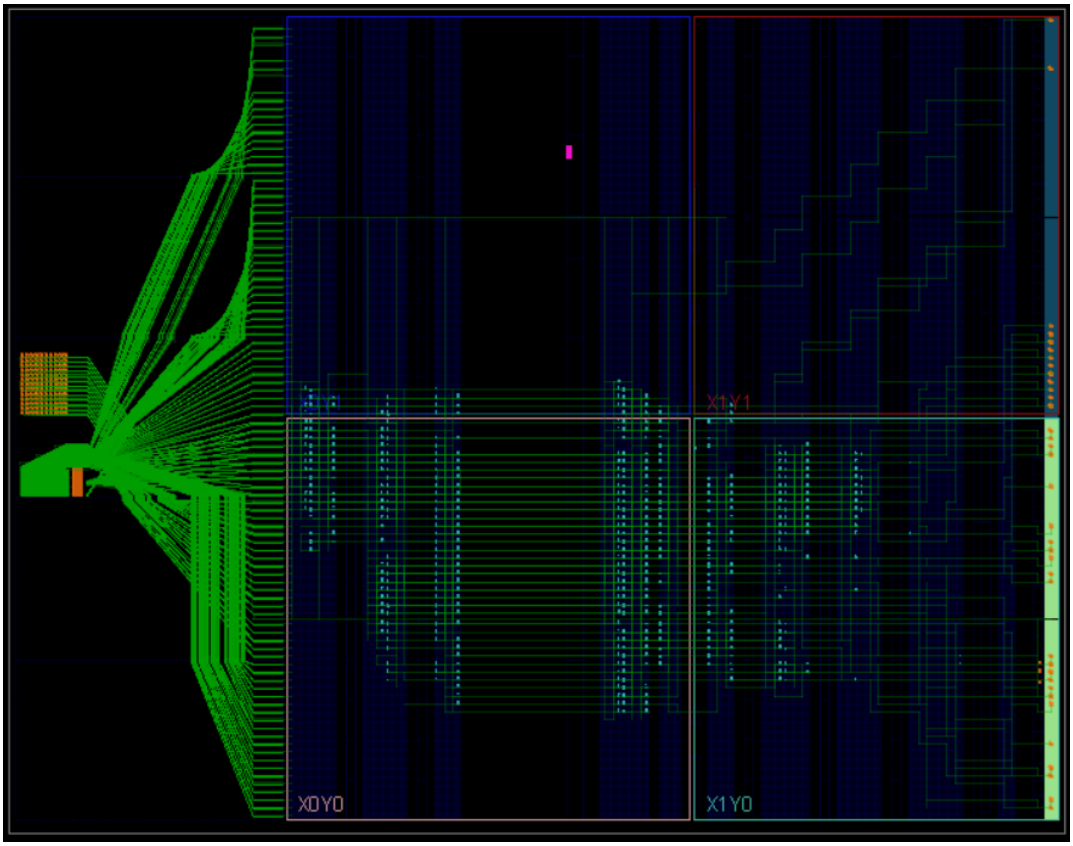

Figure: Implemented Netlist with Routing

Reports:

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**1.702 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**44.6°C**
Thermal Margin:	40.4°C (3.4 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

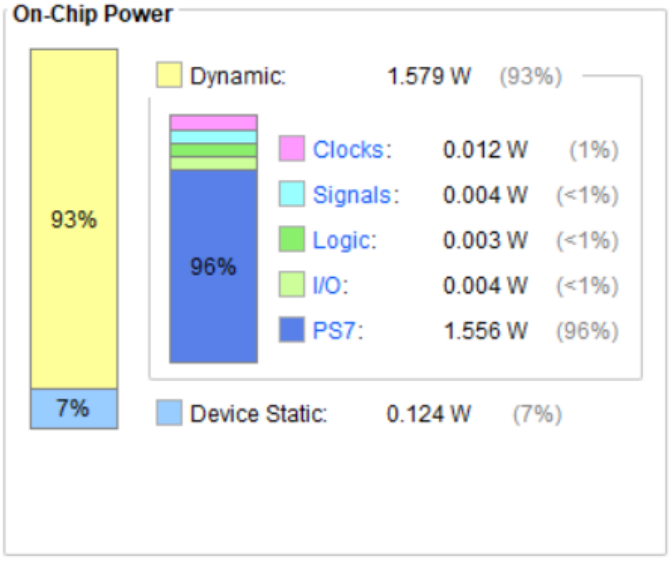

Figure: Power Report

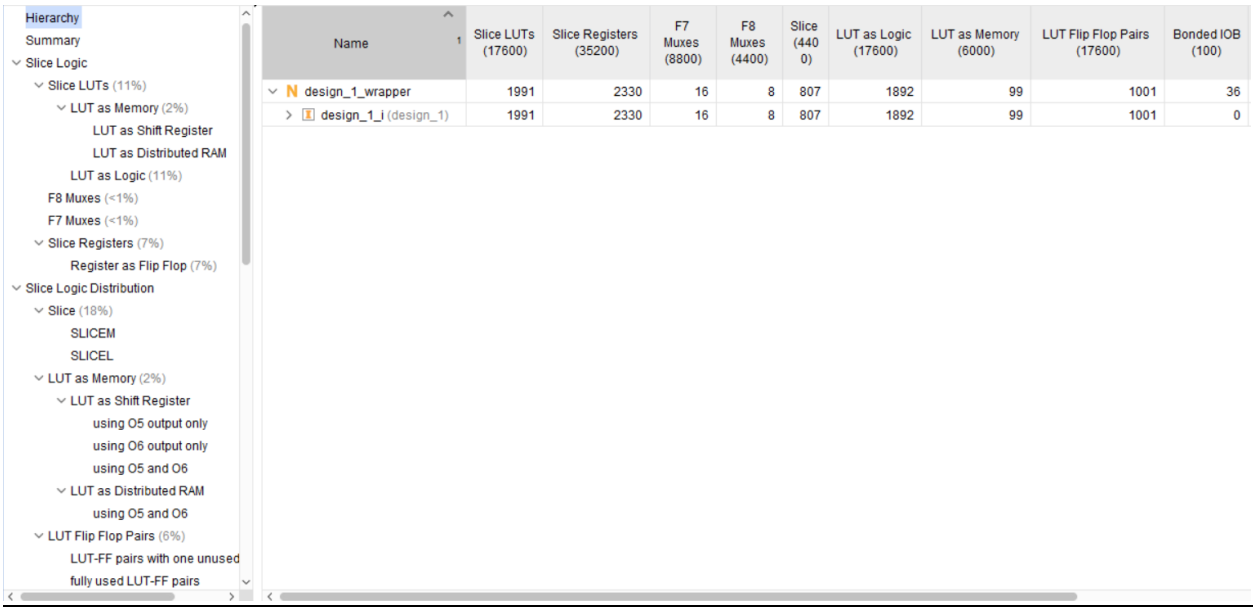

Figure: Utilization Report

Final Implementation:

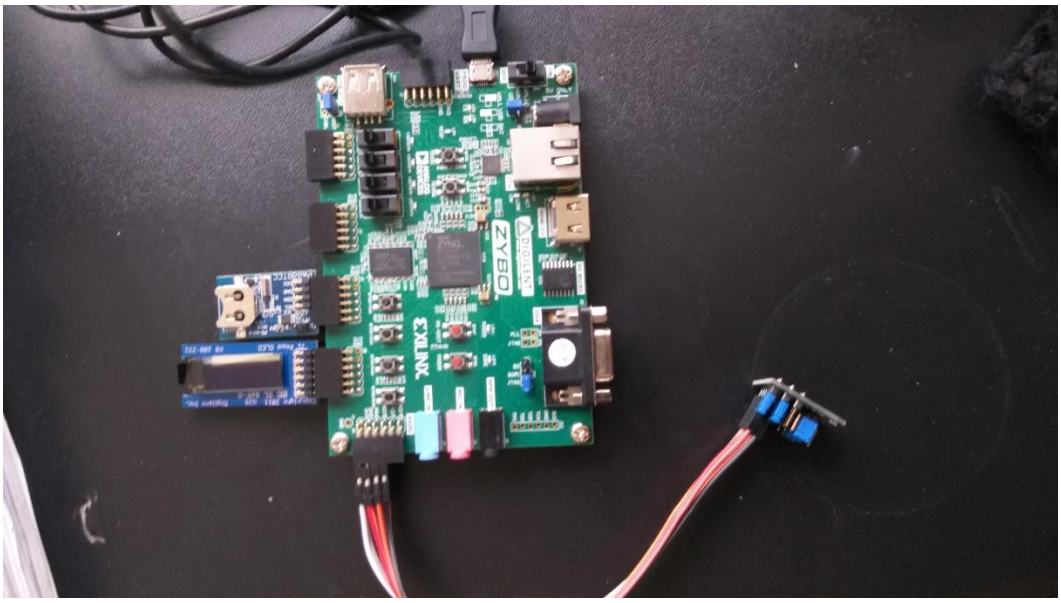

Figure: Experimental Setup

Conclusion:

- The embedded system was successfully implemented. If the program is bootstrapped onto the system, the board will only require a constant 5V DC supply to run consistently.
- This is only a prototype, upon fabrication, the entire zybo will be replaced by a single chip and all other PMODs will be SoC devices (System on Chip).