More Fun with Zynq

14:332:439:02 - Embedded Systems Hardware/Software

Lab EC

Raphaelle Marcial

December 20, 2017

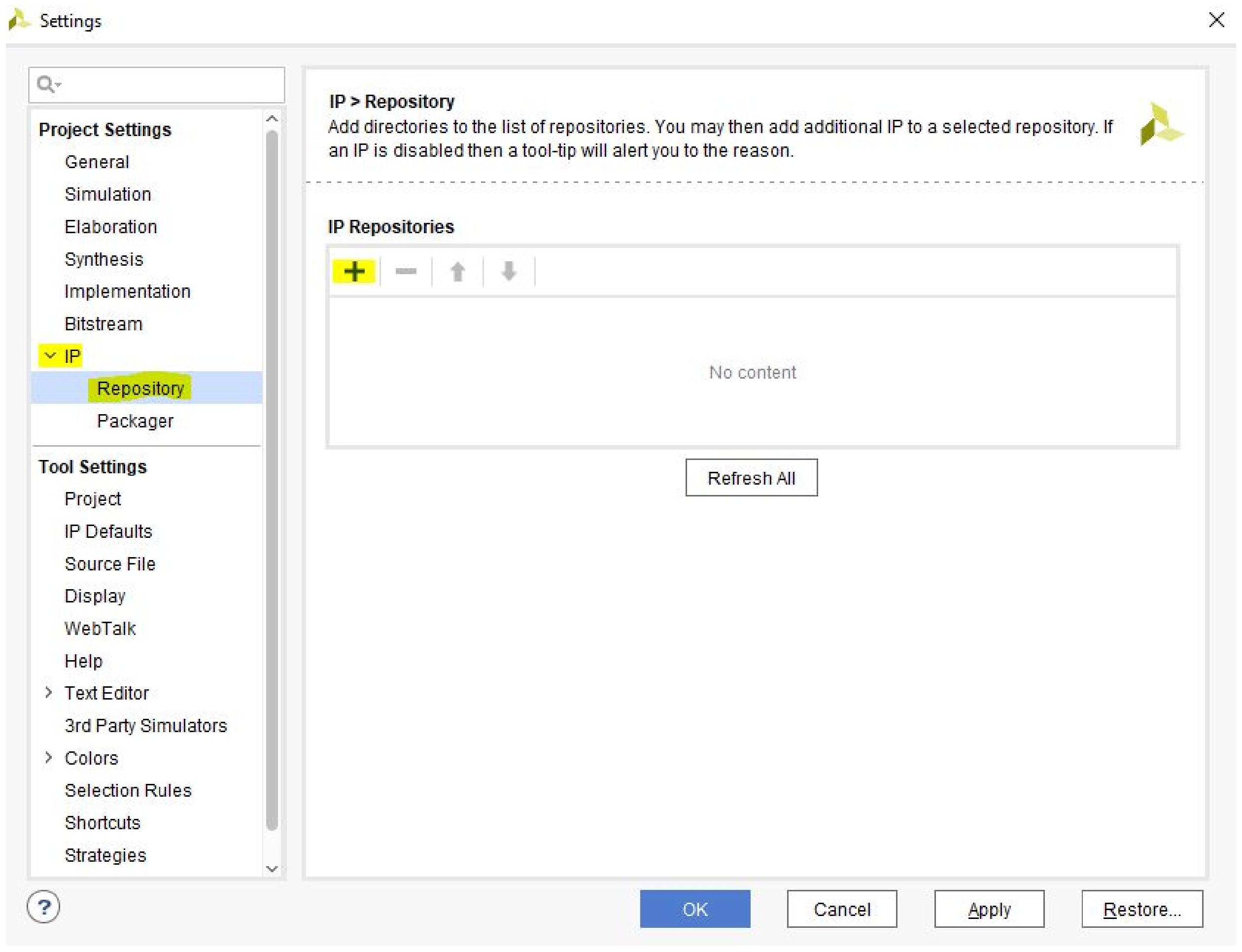
Purpose

During this lab, a circuit was designed via Vivado's IP Integrator tool that used the Zybo board ARM processor to display the current date and time from a RTCC (Real-time clock/calendar) PMOD (peripheral module) on an OLED PMOD.

Procedure

Part 1: Setup

Before starting the design process, make sure the "Zybo" board files are added to the "board_files" folder in the Vivado path. When creating a new project, the default part is "Zybo" and **not** the usual "xc7z020clg400-1" part. Furthermore, the "vivado-library-master" folder must be downloaded in order to use Digilent PMOD IPs in the IP Integrator. Add this folder to the IP Repository by going to "Settings" in the Project Manager toolbar, clicking the arrow next to IP, and clicking repository. Then, the option to add a directory to the repositories is shown with the "+" sign.



Settings

IP > Repository

Add directories to the list of repositories. You may then add additional IP to a selected repository. If an IP is disabled then a tool-tip will alert you to the reason.

IP Repositories

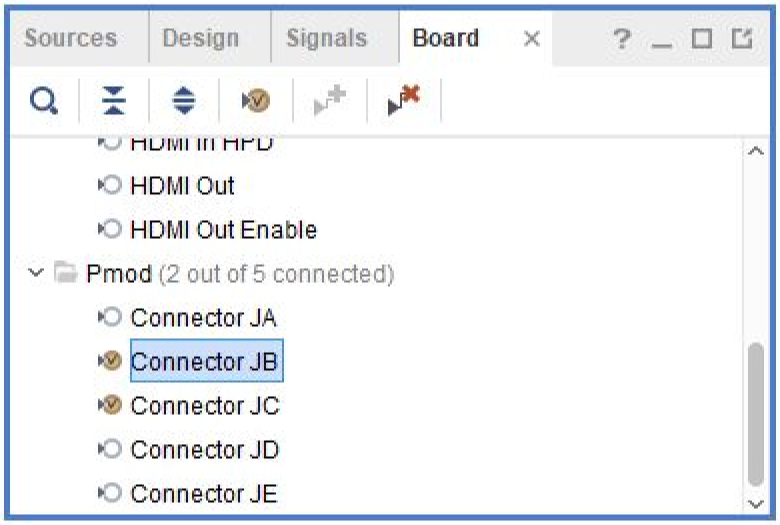
No content

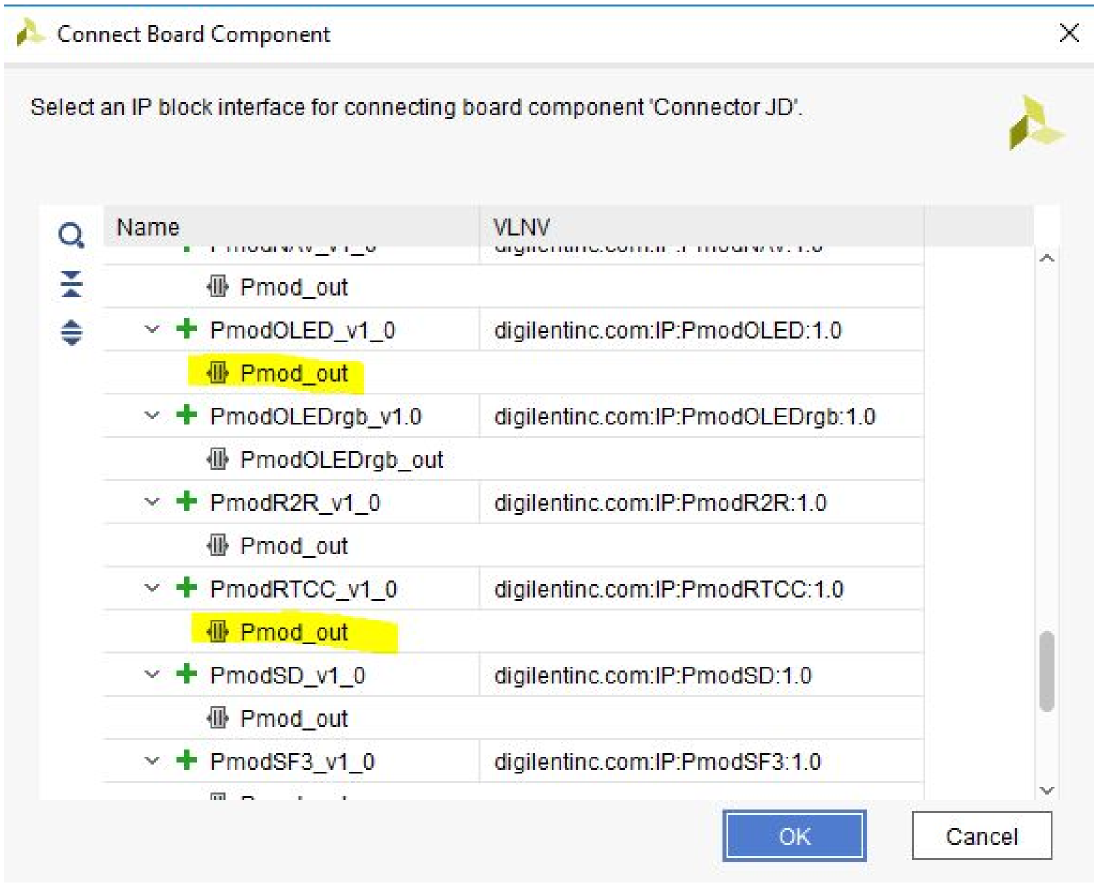
Figure 1: Steps for adding the PMOD IPs

After adding this folder, proceed to creating the block design.

Part 2: IP Integrator

Click on the Add IP option and add the ZYNQ7 Processing System to the design. Once added, the Designer Assistance bar will appear. Click on Run Block Automation and press OK. Next, add the RTCC and OLED PMODs to the design. Navigate to Board tab and scroll down to Pmod. Double click the PMOD ports and select the corresponding RTCC and OLED IPs.





Select an IP block interface for connecting board component 'Connector JD'.

Name	VLNV
Pmod_out	
PmodOLED_v1_0	digilentinc.com:IP:PmodOLED:1.0
Pmod_out	
PmodOLEDrgb_v1.0	digilentinc.com:IP:PmodOLEDrgb:1.0
PmodOLEDrgb_out	
PmodR2R_v1_0	digilentinc.com:IP:PmodR2R:1.0
Pmod_out	
PmodRTCC_v1_0	digilentinc.com:IP:PmodRTCC:1.0
Pmod_out	
PmodSD_v1_0	digilentinc.com:IP:PmodSD:1.0
Pmod_out	
PmodSF3_v1_0	digilentinc.com:IP:PmodSF3:1.0

Figure 2: Adding OLED and RTCC to Design

After this is finished, the Designer Assistance bar will appear once again. Click on Run Connection Automation. The tool will automatically port map and add necessary components. Verify the design and create the HDL wrapper for this block. Then, generate the bitstream. Once the bitstream is exported to hardware, launch the SDK. Now comes the software portion.

Part 3: SDK

The SDK will automatically open with the design created from Vivado attached to it. Once the environment is done setting up, navigate to the example code provided with each PMOD. Get familiar with each program. These files will be merged together and ran on the board.

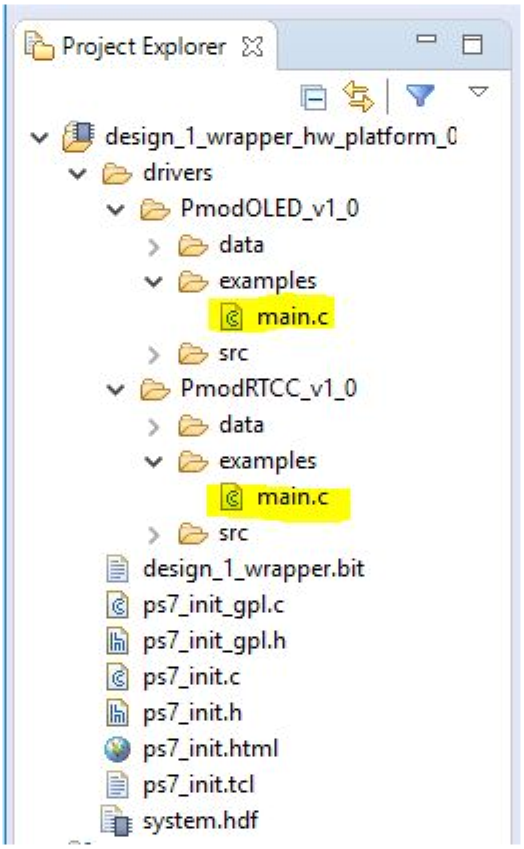


Figure 3: Finding example code for PMODs

Create a new C blank application project. Then, copy one of the main.c programs from the OLED PMOD into the src folder of the new project.

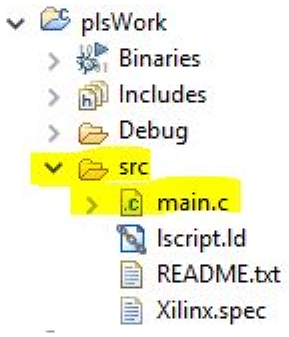


Figure 4: Adding example code to project

Click on the Program FPGA button on the top toolbar and click Program. Then click on Run -> Run As -> Launch on Hardware(System Debugger). Verify that the OLED has an output and proceed to combining the RTCC code.

<u>Note</u>: It would be wise for the student to test the RTCC module individually. Simply connect the UART PMOD and repeat this process.

Part 4: Finish

Manipulate both main.c files from the RTCC and OLED PMODs so that the OLED prints the time and date. When printing, remember that the OLED output passes strings as arguments and time from the RTCC is not of type string. Therefore, sprintf is useful in this program.

Design

combined.c

```
#include "PmodRTCC.h"
#include "xparameters.h"
#include "xil_cache.h"
#include "sleep.h"
#include <stdio.h>
#include "xil_printf.h"
#include "PmodOLED.h"
#include <stdlib.h>

const u8 orientation = 0b0;  //set up for Normal PmodOLED(false) vs normal Onboard OLED(true)
const u8 invert = 0b0;      //true = whitebackground/black letters      false = black background /white letters

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

// struct containing each possible 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"};

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

PmodRTCC myDevice;
PmodOLED myDevice2;

RTCC_Time time;
                        RTCC_Target src;
/************************** Function Declarations ***************************/

// core demo functions
void DemoRun();
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() {
    DemoInitialize(SET_RTCC);
            DemoRun();
            DemoCleanup();
            return 0;
}


/** void DemoInitialize()
**
**          Description:
**              This function initializes the demo, initializes the RTCC clock if power has failed or if
**              told to, sets up the two alarms on the device to trigger 30 seconds and 60 seconds from
**              the current time, and prints the status of each of the RTCC's time fields.
*/
void DemoInitialize(u8 mode)
{
            //OLED lines
            EnableCaches();
    OLED_Begin(&myDevice2, XPAR_PMODOLED_0_AXI_LITE_GPIO_BASEADDR,
XPAR_PMODOLED_0_AXI_LITE_SPI_BASEADDR, orientation, invert);

            // RTCC lines
            RTCC_Time time;

            EnableCaches();

            RTCC_begin(&myDevice, 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(&myDevice) || mode == SET_RTCC)
            {
                        //set the real time clock to Friday 8/7/17 1:00:00 PM
                        RTCC_stopClock(&myDevice);

                        time.second = 0x00;
                        time.minute = 0x20;
                        time.hour   = 0x03;
                        time.ampm   = RTCC_PM;
                        time.day    = 0x06;
                        time.date   = 0x11;
                        time.month  = 0x0C;
                        time.year   = 0x17;

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

                        RTCC_startClock(&myDevice);
                        xil_printf("The time has been set \r\n");
                        //set vbat high
                        RTCC_enableVbat(&myDevice);
            }
```

```
		else
		{
				time = GetTime(&myDevice, RTCC_TARGET_RTCC);
		}


		//Enable back up battery
		RTCC_enableVbat(&myDevice);

		RTCC_clearPWRFAIL(&myDevice);
}

/** void DemoRun()
**
**		Description:
**		    This function prints the current time over UART once per second, and tells the user if one of the alarms has been
triggered
*/

/** u8 bcd2int(u8 data)
**
**		Description:
**		    This function converts 8 bit binary coded decimal numbers to 8 bit unsigned integers.
*/
u8 bcd2int(u8 data) {
		return ((data >> 4) * 10) + (data & 0xF);
}

/** u8 bcd2int(u8 data)
**
**		Description:
**		    This function converts 8 bit unsigned integers to 8 bit binary coded decimal numbers.
**
**		Notes:
**				This function will lose data if requested to convert numbers larger than 99.
**				However, numbers in this range are not needed for operating the RTCC.
*/
u8 int2bcd(u8 data) {
		return (((data / 10) & 0xF) << 4) + ((data % 10) & 0xF);
}



/** RTCC_Time GetTime(PmodRTCC *InstancePtr, RTCC_Target src)
**
**		Parameters:
**				InstancePtr		- the target device to retrieve data from
**				src			- RTCC_TARGET_RTCC - real-time clock
**		      RTCC_TARGET_ALM0 - Alarm 0
**		      RTCC_TARGET_ALM1 - Alarm 1
**		      RTCC_TARGET_PWRD - power-down time-stamp
**		      RTCC_TARGET_PWRU - power-up time-stamp
**
**		Return Value:
**				val		- the contents of all time registers in the target area
**
**		Description:
**		    This function retrieves the contents of one of the Pmod RTCC's time areas
*/
RTCC_Time GetTime(PmodRTCC *InstancePtr, RTCC_Target src) {
		RTCC_Time val;
		if(src != RTCC_TARGET_PWRD && src != RTCC_TARGET_PWRU)
		val.second = RTCC_getSec(&myDevice, src);

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

		if (src == RTCC_TARGET_RTCC)
				val.year = RTCC_getYear(&myDevice);
		else
				val.year = 0;

		return val;
}

/** void SetTime(PmodRTCC *InstancePtr, RTCC_Target src, RTCC_Time val)
**
**		Parameters:
**				InstancePtr		- the target device to retrieve data from
**				src		- RTCC_TARGET_RTCC - real-time clock
**			RTCC_TARGET_ALM0 - Alarm 0
**			RTCC_TARGET_ALM1 - Alarm 1
**			RTCC_TARGET_PWRD - power-down time-stamp
**			RTCC_TARGET_PWRU - power-up time-stamp
**	val	- container for the time data to be written into the target area's registers
**
**		Return Value:
**				None
**
**		Description:
**		    This function writes data into each of the registers of one of the Pmod RTCC's time fields
*/
void SetTime(PmodRTCC *InstancePtr, RTCC_Target dest, RTCC_Time val) {
    if(dest != RTCC_TARGET_PWRD && dest != RTCC_TARGET_PWRU)
		RTCC_setSec(&myDevice, dest, val.second);

    RTCC_setMin(&myDevice, dest, val.minute);
		RTCC_setHour12(&myDevice, dest, val.hour, val.ampm);
		RTCC_setDay(&myDevice, dest, val.day);
		RTCC_setDate(&myDevice, dest, val.date);
		RTCC_setMonth(&myDevice, dest, val.month);

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

/** void PrintTime(RTCC_Target src)
**
**		Description:
**		    This function prints the current time over UART, formatted to the equivalent of "Monday 1/1/00 01:00:00 AM"
*/
void PrintTime(RTCC_Target src)
{
		RTCC_Time time;

		//fetch the time from the device
		time = GetTime(&myDevice, 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)
**
**		Parameters:
**	time			- container for time data
**	delta_seconds			- how many seconds to increment time forward by
**
**		Return:
**				result		- time data incremented forward
**
**		Description:
**		    This function steps the time parameter forward by delta_seconds, returning the result
**		    after ensuring that all modified fields are in the proper range.
**
**		Errors:
**		    This function will return the time parameter if requested to cross the midnight/noon boundary.
*/
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;
}
void DemoCleanup() {
		DisableCaches();
		OLED_End(&myDevice2);
}

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
}

void DemoRun()
{
		//OLED lines
    int irow, ib, i;
    u8 *pat;
    char c;
    char buffer[50];
    xil_printf("UART and SPI opened for PmodOLED Demo\n\r");

		while(1)
		{


				//fetch the time from the device
								time = GetTime(&myDevice, src);

								time = IncrementTime(time, 1);//TEST
				EnableCaches();

				RTCC_begin(&myDevice, XPAR_PMODRTCC_0_AXI_LITE_IIC_BASEADDR, 0x6F);


				//OLED lines
        xil_printf("entering loop\r\n");
        //Choosing Fill pattern 0
        pat = OLED_GetStdPattern(0);
        OLED_SetFillPattern(&myDevice2, pat);
        //Turn automatic updating off
        OLED_SetCharUpdate(&myDevice2, 0);

        //Draw a rectangle over writing then slide the rectangle
        //down slowly displaying all writing

        for (irow = 0; irow < OledRowMax; irow++)
        {
        	OLED_SetCursor(&myDevice2, 0, 1);
        	sprintf(buffer, "%d", time.day);
            OLED_PutString(&myDevice2, weekdays[time.day]);
            OLED_SetCursor(&myDevice2, 0, 0);
            sprintf(buffer, "%d", time.month);
            OLED_PutString(&myDevice2, (buffer));

            OLED_PutString(&myDevice2,"-");
            sprintf(buffer, "%d", (time.date));

            OLED_PutString(&myDevice2, buffer);
            OLED_PutString(&myDevice2,"-");

            sprintf(buffer, "%d",bcd2int(time.year));
            OLED_PutString(&myDevice2, buffer);


           OLED_MoveTo(&myDevice2, 0, irow);
           OLED_FillRect(&myDevice2, 127, 31);
           OLED_MoveTo(&myDevice2, 0, irow);
           OLED_LineTo(&myDevice2, 127, irow);
           OLED_Update(&myDevice2);
// Time
           OLED_SetCursor(&myDevice2, 0, 3);
           sprintf(buffer, "%d", time.hour);
           OLED_PutString(&myDevice2, (buffer));

           OLED_PutString(&myDevice2,":");
           sprintf(buffer, "%d", bcd2int(time.minute));

           OLED_PutString(&myDevice2, buffer);
           OLED_PutString(&myDevice2,":");

           sprintf(buffer, "%d",bcd2int(time.second));
            OLED_PutString(&myDevice2, buffer);

            //OLED_PutString(&myDevice2, "PM");

           OLED_MoveTo(&myDevice2, 0, irow);
           OLED_FillRect(&myDevice2, 127, 31);
           OLED_MoveTo(&myDevice2, 0, irow);
           OLED_LineTo(&myDevice2, 127, irow);
           OLED_Update(&myDevice2);
        }

        for (irow = OledRowMax-1; irow >= 0; irow--)
                {
                        OLED_SetDrawColor(&myDevice2, 3);
                        OLED_SetDrawMode(&myDevice2, OledModeSet);
                        OLED_MoveTo(&myDevice2, 0, irow);
                        OLED_LineTo(&myDevice2, 127, irow);
                        OLED_Update(&myDevice2);
                                                usleep(250);
                        OLED_SetDrawMode(&myDevice2, OledModeXor);
                        OLED_MoveTo(&myDevice2, 0, irow);
                        OLED_LineTo(&myDevice2, 127, irow);
                        OLED_Update(&myDevice2);
                      }



            }

}
```

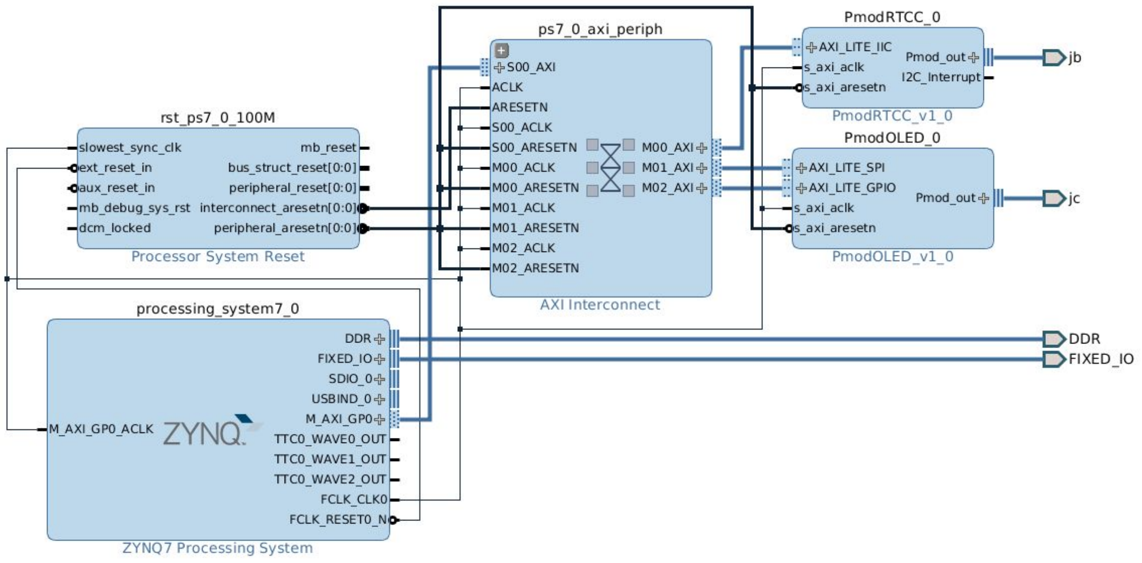


Figure 5: Block Design