HACETTEPE UNIVERSITY
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

INTERNSHIP REPORT

Name Surname: Fatih KAYA
Student Number: 21329281
Grade: 4th
Name of Company: ASELSAN A.Ş.
Internship Start and End Date: 07.08.2017-25.09.2017
Internship Duration: 30 Workdays

Table of Contents

1. INTRODUCTION TO COMPANY

ASELSAN is the largest defense electronics company of Turkey whose capability/product portfolio comprises communication and information technologies, radar and electronic warfare, electro-optics, avionics, unmanned systems, land, naval and weapon systems, air defence and missile systems, command and control systems, transportation, security, traffic, automation and medical systems. Today ASELSAN has become an indigenous products exporting company, investing in international markets through various cooperation models with local partners and listed as one of the top 100 defence companies of the world

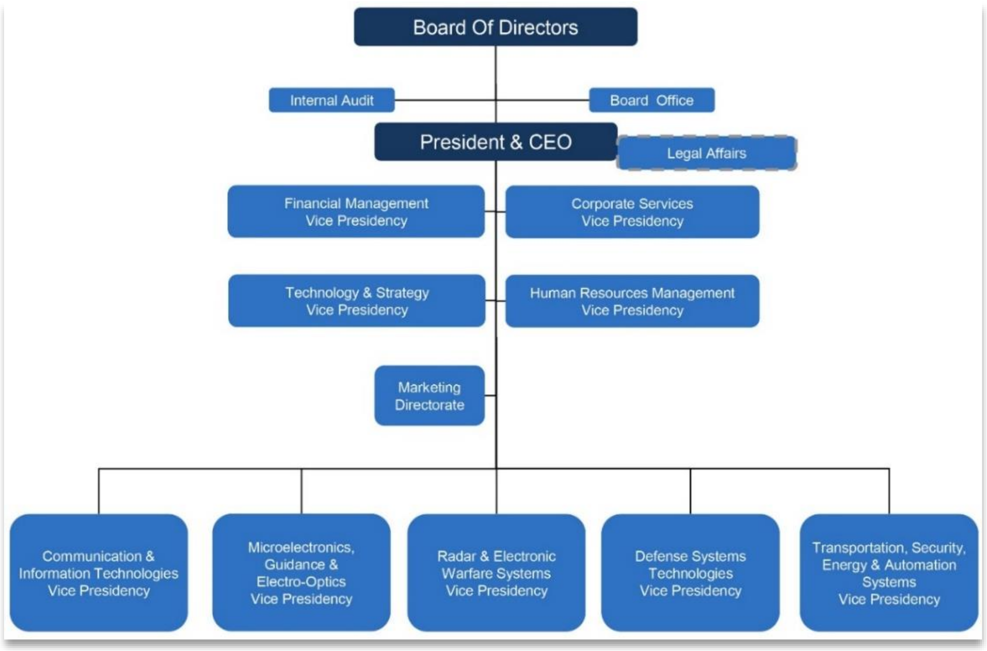
ASELSAN, together with the technology emphasis in its vision, has targeted to be a company that maintains its sustainable growth by creating value in the global market; preferred due to its competitiveness, trusted as a strategic partner, and caring for the environment and people.

Together with the highly qualified engineering staff within more than 5000 employees, being the main driving factor of the company's success, ASELSAN allocates 6% of its annual income for self-financed research and development activities

1.1. Address of Company:

ASELSAN Gölbaşı Facilities .
Konya Yolu 8. Km, Oğulbey Mah. 3051. Sok. No:3, 06830 Ankara, Türkiye

1.2. Organizational structure of ASELSAN


Board Of Directors
Internal Audit
Board Office
President & CEO
Legal Affairs
Financial Management Vice Presidency
Corporate Services Vice Presidency
Technology & Strategy Vice Presidency
Human Resources Management Vice Presidency
Marketing Directorate
Communication & Information Technologies Vice Presidency
Microelectronics, Guidance & Electro-Optics Vice Presidency
Radar & Electronic Warfare Systems Vice Presidency
Defense Systems Technologies Vice Presidency
Transportation, Security, Energy & Automation Systems Vice Presidency

2. WEEK 1

In the first week of the internship in Aselsan there would not be any job to do. The week was a orientation week for Aselsan and internship. In the half of the week, experts gave conferences about their expertise. In the first day, the subject was the Aselsan. They gave information about Aselsan, the History of the Aselsan, vision and the mission of the Aselsan. They also touched on the information security in Aselsan. Information is very important for Aselsan so that any device that can be to take information from Aselsan is forbidden like flash disk, CD even telephone is forbidden for interns (just in REHIS).

Occupational health safety training experts gave information about basics of occupational health safety training. In the morning, they presented about information on working legislation, legal rights and responsibilities of employees, workplace cleaning and tidying and legal consequences of work accidents and occupational diseases. Sides of occupational health safety training are state, employer and employee. They gave information about the labor law numbered 4857 and 6331. Arrangement(classification), layout(systematization), cleanliness(sweep), perseverance(standardize) and discipline support each other. In the afternoon, doctor who works in ASELSAN presented about the reasons of occupational diseases, prevention of diseases and application of protection techniques, biological and psychosocial risk factors and first aid. Group A diseases are diseases with chemical substances. Group B diseases are diseases with skin. Group C diseases are diseases with respiratory. Group D diseases are infectious disease. Group E diseases are diseases with physical factors. Our ears are damaged after 85 dB so we need to have a headphone after 85 dB. Technical planning is done to prevent risk factors. The solution is produced respectively 1) on the source, 2)in the environment, 3)in person. Thermal comfort terms are known as air temperature, air humidity, air flow rate, and air radiant heat. It was said that the technical issues will be presented tomorrow.

Presentation of occupational health safety training continued. In the morning, they presented about information of technical topics such as chemical, physical and ergonomic risk factors, glare, explosion, fire and fire protection, working with screened tools, safety and health signs, use of personal protective equipment, evacuation and rescue. Fires can be several types such as fires of solid, liquid, gas, metal and electric. Fire signs progress in the order of smell, smoke and flame respectively. Safety and health signs can be various colors and specific meaning. Red signs are forbidden, yellow signs are warning, blue signs are obligatory, green signs are safe situation.

In the afternoon of the third day of the week, interns are distributed to facilities by Aselsan. I went to REHIS test systems and processes design directorate. My departments responsible is the to design test software and hardware for digital, analog circuit parts of the radars. In that department C# is used for GUI softwares, and Xilinx FPGAs to design digital test hardwares.

3. WEEK 2

In this week, I made researches to have knowledge about the board I use, and the software environment that I worked with.

3.1. Introduction to VC707 Evaluation Board

During the internship, I have been using VC707 evaluation board which provides a hardware environment or developing and evaluating designs targeting Virtex-7 XC7Vx485T-2FFG1761C FPGA.

Some features of VC707 evaluation board are listed below.

- Clock Generation:
    - Fixed 200 MHz LVDS oscillator (Differential)
    - I2C Programmable LVDS oscillator (Differential)
    - SMA connector (Differential)
    - SMA connector for GTX transceiver (Differential)
    - Also 25 MHz clock is available for Ethernet PHY

- User I/O
  - 8 User LEDs
  - 5 User Push Buttons
  - 8-pole DIP Switches
  - One pair of User SMA GPIO Connectors
  - 16x2 Character LCD
- PMBus voltage and current monitoring through TI power controller
- HDMI CODEC
- 10/100/1000 tri-speed Ethernet PHY
- SFP+ Connector
- PCI Express endpoint connectivity
- VITA 57.1 FMC1/2 HPC Connector
- 1 GB DDR3 memory SODIMM
- 128 MB BPI Flash memory

Block Diagram of the Board is shown below.

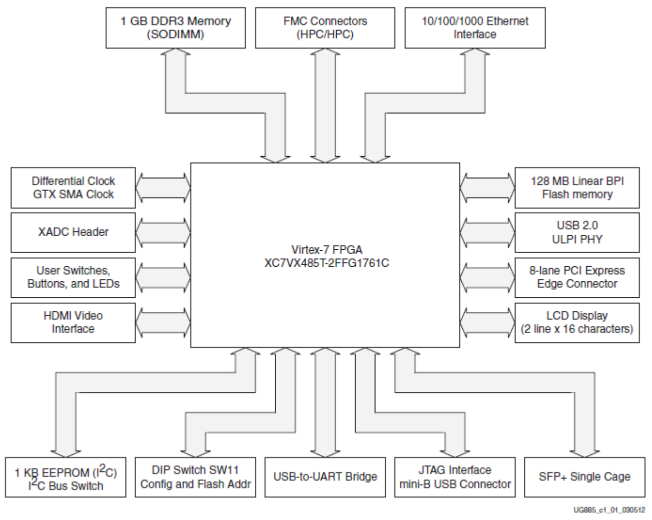

1 GB DDR3 Memory (SODIMM)
FMC Connectors (HPC/HPC)
10/100/1000 Ethernet Interface
Differential Clock GTX SMA Clock
XADC Header
User Switches, Buttons, and LEDs
HDMI Video Interface
Virtex-7 FPGA XC7VX485T-2FFG1761C
128 MB Linear BPI Flash memory
USB 2.0 ULPI PHY
8-lane PCI Express Edge Connector
LCD Display (2 line x 16 characters)
1 KB EEPROM (I2C) I2C Bus Switch
DIP Switch SW11 Config and Flash Addr
USB-to-UART Bridge
JTAG Interface mini-B USB Connector
SFP+ Single Cage
UG885_c1_01_030512

Figure 1-1: **VC707 Board Block Diagram**

FPGA Configuration:

Before using the FPGA board, configuration switches must be set. There are two configuration modes of the VC707 which are Master BPI and JTAG. Configuration switches and modes are shown below.

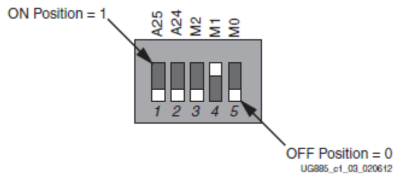


Figure 1-3: **SW11 Default Settings**

Table 1-2: **VC707 Board FPGA Configuration Modes**

Configuration Mode	SW13 DIP switch Settings (M[2:0])	Bus Width	CCLK Direction
Master BPI	010	x8, x16	Output
JTAG	101	x1	Not Applicable

3.2. Introduction to Microblaze Soft Processor IP

Microblaze soft processor ip is reduced instruction set computer(RISC) optimized for implementation in Xilinx FPGAs. Block Diagram of MicroBlaze is shown below.

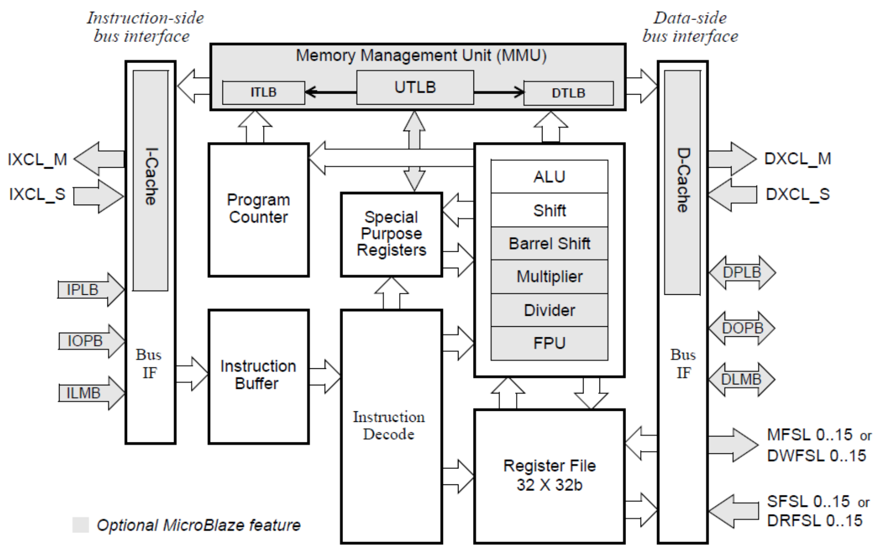


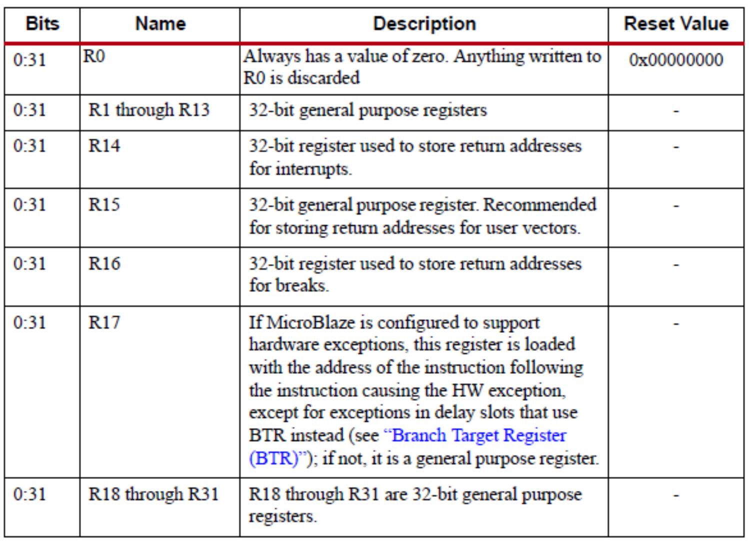
Figure 1-1: **MicroBlaze Core Block Diagram**

Gray parts on the diagram are optional parts and they can be added or removed by configuring the IP in the Vivado project. Others are fixed and cannot be configured.

Microblaze uses Big-Endian format to represent data and the supported data types are word, half word, and byte.

Microblaze has 32 32-bit general purpose registers and up to 18 32-bit special purpose registers, depending on configurations.

Table 1-7: **General Purpose Registers (R0-R31)**



Bits	Name	Description	Reset Value
0:31	R0	Always has a value of zero. Anything written to R0 is discarded	0x00000000
0:31	R1 through R13	32-bit general purpose registers	-
0:31	R14	32-bit register used to store return addresses for interrupts.	-
0:31	R15	32-bit general purpose register. Recommended for storing return addresses for user vectors.	-
0:31	R16	32-bit register used to store return addresses for breaks.	-
0:31	R17	If MicroBlaze is configured to support hardware exceptions, this register is loaded with the address of the instruction following the instruction causing the HW exception, except for exceptions in delay slots that use BTR instead (see "Branch Target Register (BTR)"); if not, it is a general purpose register.	-
0:31	R18 through R31	R18 through R31 are 32-bit general purpose registers.	-

Special Purpose Registers	
PC	Program Counter
MSR	Machine Status Register
EAR	Exception Address Register
ESR	Exception Status Register

BTR	Branch Target Register
FSR	Floating Point Status Register
EDR	Exception Data Register
PID	Process Identifier Register
ZPR	Zone Protection Register
TLBLO	Translation Look-Aside Buffer Low Register
TLBHI	Translation Look-Aside Buffer High Register
TLBX	Translation Look-Aside Buffer Index Register
TLBSX	Translation Look-Aside Buffer Search Index Register
PVR	Process Version Register

MicroBlaze Instructions are pipelined. For most instructions, each stage takes one clock cycle to complete. Number of pipeline stages can be configured by setting C_AREA_OPTIMIZED. To minimize the hardware cost three stage pipeline can be used by setting C_AREA_OPTIMIZED 1. And to maximize the performance five stage pipeline can be used by setting C_AREA_OPTIMIZED 0.

The MicroBlaze cores are organized as a Harvard architecture with separate bus interface units for data and instruction accesses. Supported bus interfaces are listed below.

- Local Memory Bus (LMB) : Provides single-cycle access to dual port block RAM.
- Processor Local Bus (PLB) or On-Chip Peripheral Bus (OPB) : Provide a connection to both on-chip and off-chip peripherals and memory.
- Xilinx CacheLink (XCL) : provides interface between caches and external memory controllers.

There is some IP block not mandatory to add to the project to work with MicroBlaze. But they are required to set up a basic MicroBlaze system. They are LMB Block RAM, MicroBlaze Debug Module (MDM), Processor System Reset, and Clocking Wizard.

- LMB Block RAM: Provides a low area, high frequency and low latency connection for the MicroBlaze DLMB (for Data) and ILMB (for instructions) ports to device block RAM. The supported block RAM sizes are determined by the Block Memory Generator (4 – 256 KB).
- MDM: Enables JTAG-based debugging of one or more (up to 32) MicroBlaze processors. Provides a configurable AXI4 Master port for direct access to memory from JTAG.
- Clocking Wizard: Accepts up to two input clocks and up to seven output clocks per clock network. Provides feature to monitor the clocks. Optionally buffers clock signals.
- System Processor Reset: Reset inputs are selectable as active-High and active-Low.

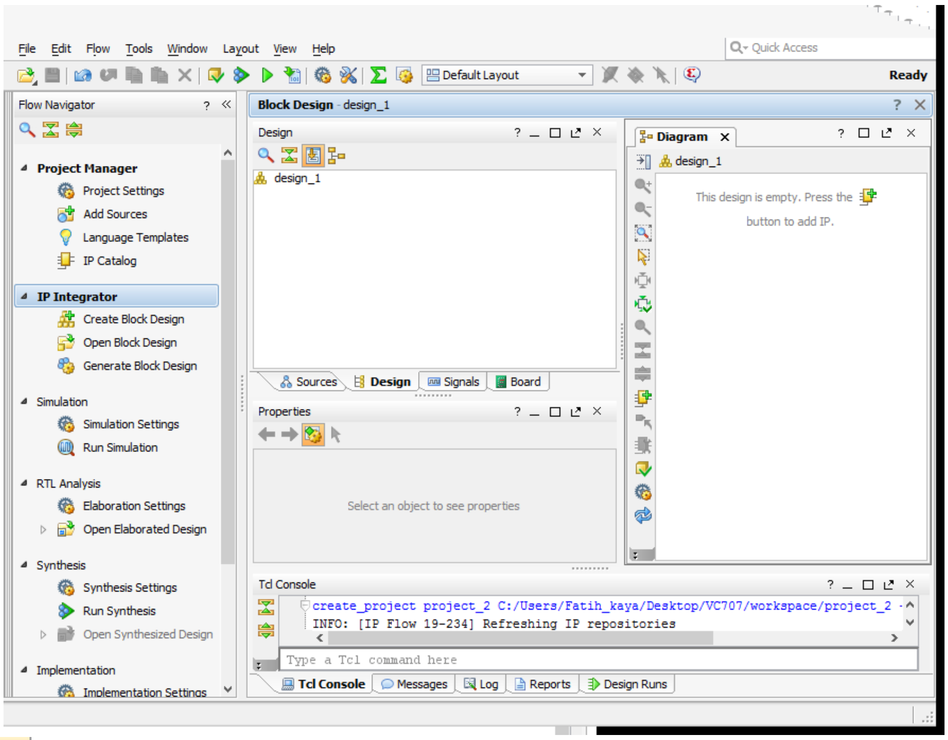
3.2.1. How to Open a MicroBlaze Project in Vivado

Xilinx has two software to design, synthesize and implement hardware for their FPGAs. Xilinx ISE which is the older software does not support newer FPGA boards. VC707 evaluation board is one of them so Vivado which is newer software, is used to design MicroBlaze project on VC707.To open a new Microblaze following steps are done.

Hardware Design Part

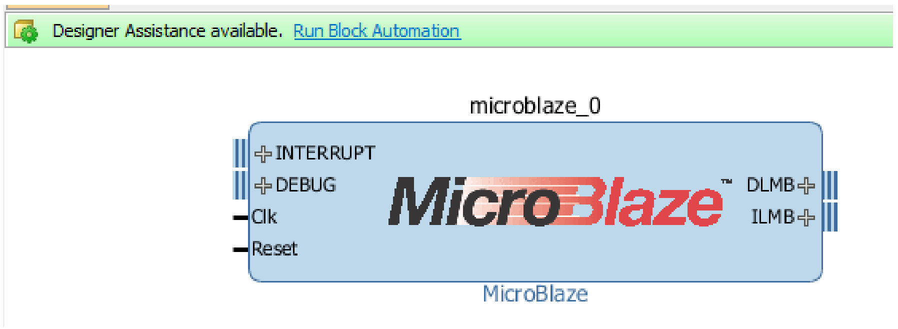
- Select “Create New Project” from quick start or File menu.
- Set “Project Name” and “Project Directory” then press “Next”.
- Select “RTL Project” on the opening page then press “Next”.
- Select the chip or board on the opening page. Board selection is recommended for easy pin assignment.
- After all, select finish. Then project will open.

- Select “Create Block Design” below “Project Manager/IP Integrator” menu. And name the design.
  - This will create a new block design project. After all, window must look as following.

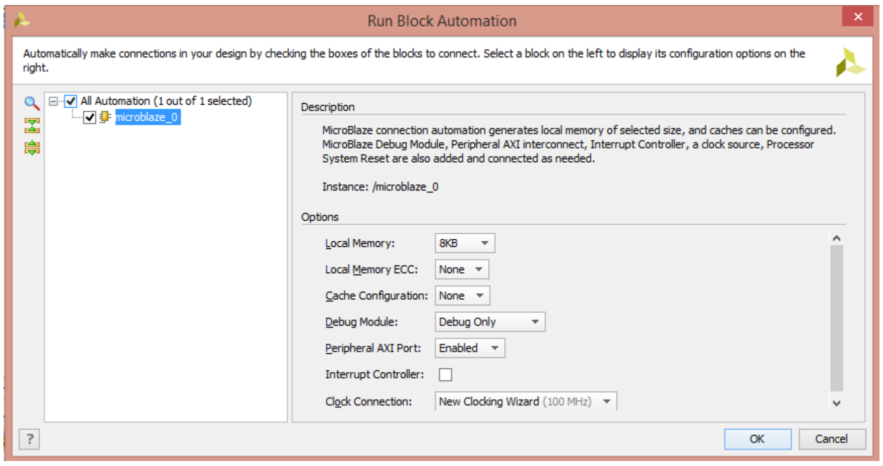


```
create_project project_2 C:/Users/Fatih_kaya/Desktop/VC707/workspace/project_2
INFO: [IP Flow 19-234] Refreshing IP repositories
Type a Tcl command here
```

- Click symbol to add a new IP. Then search and select Microblaze on the opening window.



- Click on the “Run Block Automation”. This wizard will add necessary blocks to your design and makes their connection depending on your selection



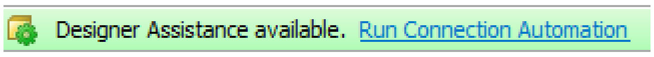
Automatically make connections in your design by checking the boxes of the blocks to connect. Select a block on the left to display its configuration options on the right.

MicroBlaze connection automation generates local memory of selected size, and caches can be configured. MicroBlaze Debug Module, Peripheral AXI interconnect, Interrupt Controller, a clock source, Processor System Reset are also added and connected as needed.

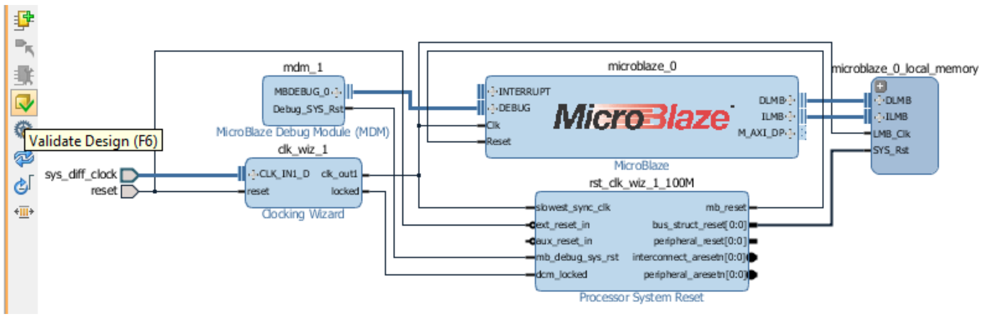
Instance: /microblaze_0

Local Memory:	8KB
Local Memory ECC:	None
Cache Configuration:	None
Debug Module:	Debug Only
Peripheral AXI Port:	Enabled
Interrupt Controller:	☐
Clock Connection:	New Clocking Wizard (100 MHz)

- Local Memory can be increased up to 128 KB if the program written in c needs more memory. Memory requirement error can be faced in large software. Also, cache can be added up to 64 KB to increase performance. Clock connection can be selected any clock pin.
- After configuration done. Click “OK”. And click “Run Connection Automation”.

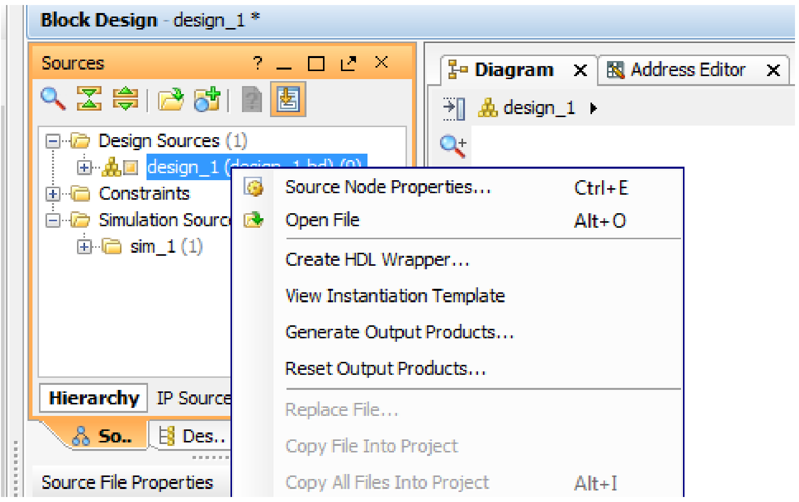


Select all connection configurations. That will make the necessary connection.



-

- Your design must seem as above. Click Validate Design to check whether there is any error in the connections.



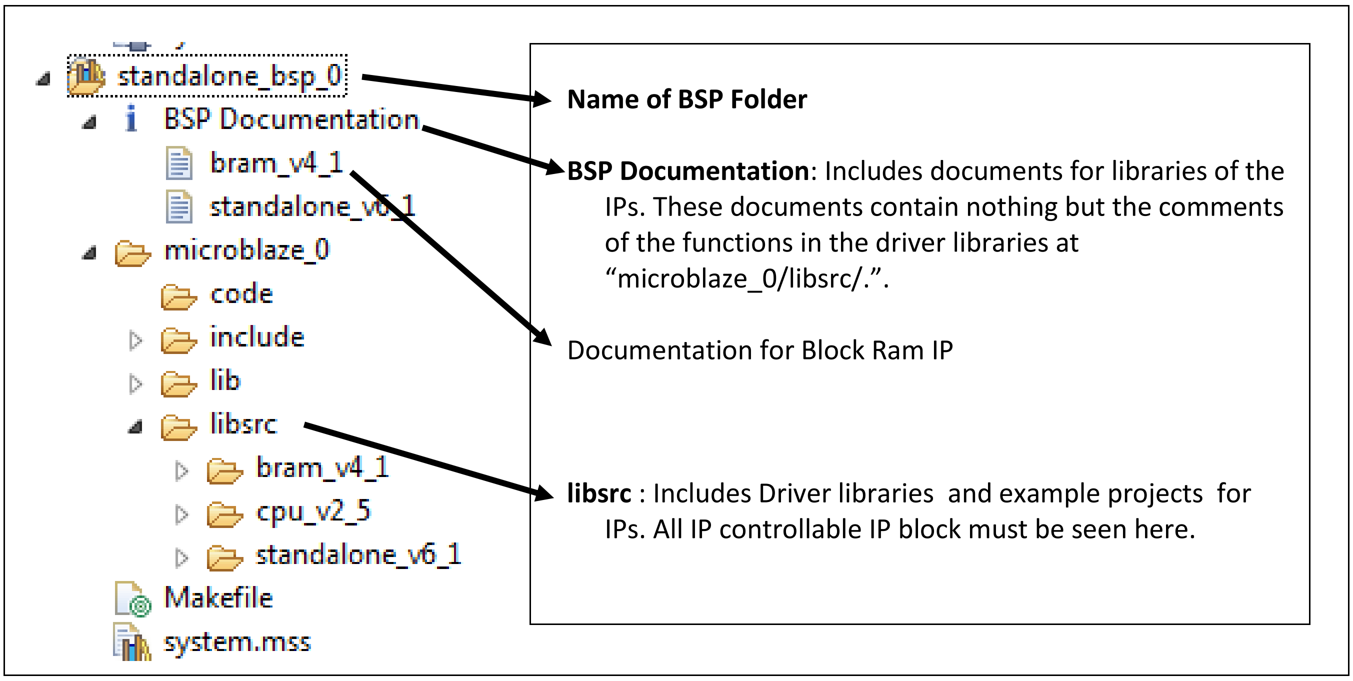
- On the Sources/Hierarchy tab right click on the design and select “Create HDL Wrapper”.
- Now it is ready to generate Bitstream. Click on “Generate Bitstream”. Select “Yes” and “Ok” to all opening windows.
- After Bitstream is generated successfully select “File/Export/Export Hardware”
- Tick the “Include Bitstream” selection. Then click Ok.
- Select “File/Launch SDK” to open SDK where C code to load to designed microprocessor is written.

Coding Part

After SDK is opened the hardware platform folder is generated which is named as <Your Design Wrapper Name>_hw_platform_0 and includes three files with .bit, .mmi and .hdf extensions. First thing we need to the is generating Board Support Package (BSP).

- Select “File/New/ Board Support Package” to generate a new BSD file. Name it and click finish.
- Select “File/New/ Application Project”.
    - Name the Project
    - OS platform = standalone

- Hardware Platform = <Your Design Wrapper Name>_hw_platform_0
- Processor = microblaze_x
- Language = Choose which you want to write
- Board Support Package = Can be created one more or can be used existing one. Select the use existing one. (It depends on you.)



Name of BSP Folder

BSP Documentation: Includes documents for libraries of the IPs. These documents contain nothing but the comments of the functions in the driver libraries at "microblaze_0/libsrc/.".

Documentation for Block Ram IP

libsrc : Includes Driver libraries and example projects for IPs. All IP controllable IP block must be seen here.

- Click finish or click next and select an example project.
-

Up to now only simple MicroBlaze system is designed. From now new peripherals will be added and controlled on software.

4. WEEK 3

In this week, I started to add some peripherals and control them with SDK. First, I tried to run the example designs and by reading and changing the code I learned how the examples work, and I started to make my designs.

4.1. Using Uart with Microblaze

4.1.1. Overview of Uartlite IP

Uartlite IP uses AXI4-Lite interface for register access and data transfer. It has 16-character transmit and receive FIFOs. The Number of data bits (5-8), parity bit (odd, even or none) and baud rate are configurable on hardware. In software part, there is no access to these configurations.

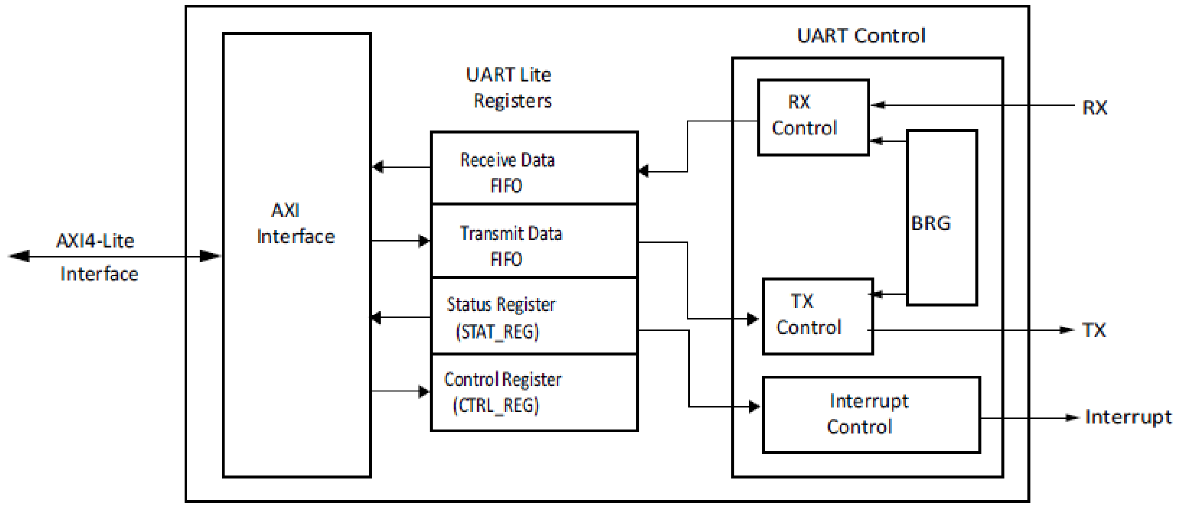


Figure 1-1: **Block Diagram of AXI UART Lite**

Receive Data FIFO holds the received data and it is a read only register.

Transmit Data FIFO holds the data to transmit and it is a write only register.

Status register contains the status of the receive and transmit data FIFOs when interrupts are enabled and errors are present. Some of these error and status are Tx FIFO Empty, Rx FIFO Full, Parity Error, Interrupt Enabled etc.

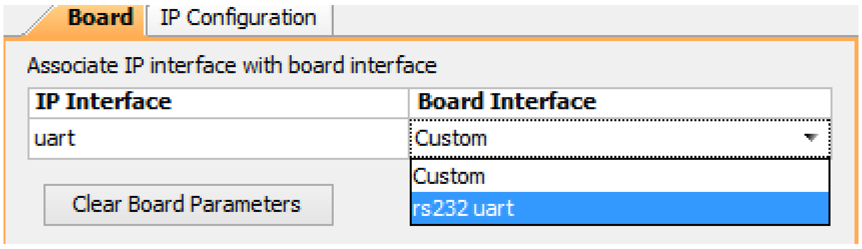
Control register contains the enable interrupt bit and reset pin for the receive and transmit data FIFO.

If the interrupts are enables, a rising edge- sensitive interrupt is generated when the receive FIFO becomes non-empty or when the transmit FIFO becomes empty.

4.1.2. Hardware Part

In this part adding a Uartlite IP to a MicroBlaze project will be discussed.

- First click on “Add New IP” button and select “AXI Uartlite”.
- Double Click on the IP to configure.

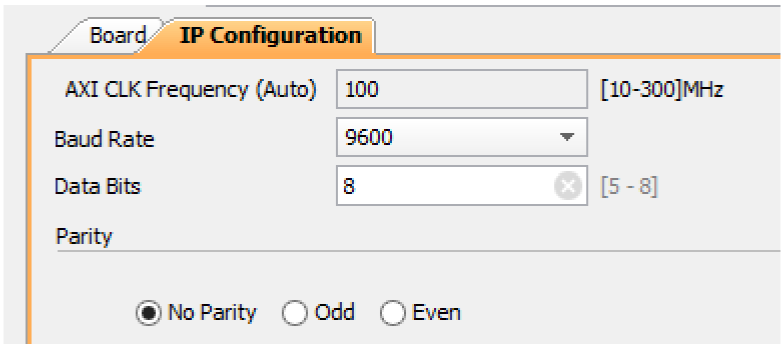


Associate IP interface with board interface

IP Interface	Board Interface
uart	Custom

- Select rs232 Uart on board interface under Board tab.

- Set the configuration under the IP configuration tab



AXI CLK Frequency (Auto) 100 [10-300]MHz
Baud Rate 9600
Data Bits 8 [5 - 8]
Parity
[✓ No Parity] Odd Even

- Click Ok.
- Click "Run Connection Automation"
- Generate bitstream, export hardware and overwrite to existing hardware.
- Open SDK.

4.1.3. Software Part

There are lots of ways to send data from FPGA to computer via Uart. Two of them will be discussed in this section.

First and Effortless Way:

```
#include "stdio.h"

int main()
{
	xil_printf("Hello World");
	xil_printf("% is an integer.",5);
	xil_printf("%c is a character.",'A');
	return 0;
}
```

Xil_printf() is the same as the standard printf() function in C. "stdio.h"library must be included to use this function.

Second Way:

```
#include "xparameters.h"
#include "xuartlite.h"

XUartLite Uart;
#define Uart_Device_ID XPAR_AXI_UARTLITE_0_DEVICE_ID

#define BUFFER_SIZE 20
int main()
{
	u8 data[BUFFER_SIZE] = "Hello World";
	XUartLite_Initialize(&Uart,Uart_Device_ID);

	XUartLite_Send(&Uart,data,BUFFER_SIZE);
	return 0;
}
```

"xparameters.h " library contains all parameters and configurations of the IPs including base addresses, device IDs, register addresses and so on. "xuartlite.h" contains APIs, types and definitions for Uartlite IP.

"XUartlite" is a struct type definition for Uart module, and it must be declared before using any function because all functions get it as a parameter. `XUartLite_Initialize()` function initializes the Uart module and sets the does the configuration according to the device ID which is `XPAR_AXI_UARTLITE_0_DEVICE_ID` for Uartlite. `XUartLite_Send()` function send the specified data via Uart. `XUartLite_Recv()` Function is available to receive data to FPGA but it will be discussed after interrupt block.

4.2. Using GPIO with MicroBlaze

4.2.1. Overview

The AXI GPIO design provides a general-purpose input/output interface to an AXI4-Lite interface. The AXI GPIO can be configured as either a single or a dual-channel device.

Ports direction can be changed dynamically on software unless ALL_INPUTS and ALL_OUTPUTS configurations set to 0, and 3-state buffer is enabled for inputs and outputs. The width of each channel is independently configurable. The channels can be configured to generate an edge triggered interrupt.

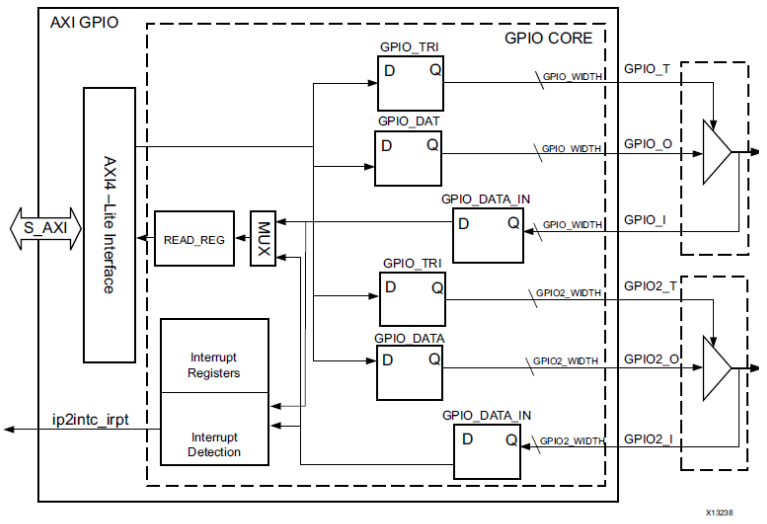
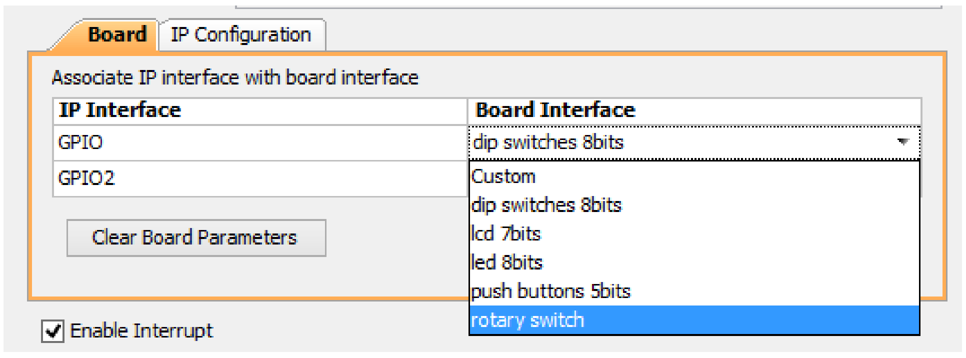


Figure 1-1: **AXI GPIO Block Diagram**

4.2.2. Hardware Part

In this part adding a GPIO IP to a MicroBlaze project will be discussed.

- First click on "Add New IP" button and select "AXI GPIO"
- Double Click on the IP block.



Associate IP interface with board interface

IP Interface	Board Interface
GPIO	dip switches 8bits
GPIO2	

☑ Enable Interrupt

-

- Select the peripherals you want to use. If you want to use interrupt do not remove the “Enable Interrupts” tick. There are two channel you can use. But it also can be used as a single channel.
- If you are not using the custom interface it not possible to set inputs as outputs and outputs as input in Vivado GUI.
- Click Ok.
- Click “Run Connection Automation”
- Generate bitstream, export hardware and overwrite to existing hardware.
- Open SDK.

4.2.3. Software Part

The code below shows how to configure GPIO ports as output or input and how to read data from I/O and write data to I/O. After following code is executed all LEDs on VC707 board will be the same as the DIP states.

```
#include "xstatus.h"
#include "xparameters.h"
#include "xgpio.h"

#define GPIO_LED_DEVICE_ID XPAR_GPIO_0_DEVICE_ID
#define LED_CHANNEL 1
#define DIP_CHANNEL 2

XGpio Gpio; /* The Instance of the GPIO Driver */

int main(void){
	int Status;
	int DIP_State;

	Status = XGpio_Initialize(&Gpio, GPIO_LED_DEVICE_ID);
	if (Status != XST_SUCCESS) {
		return XST_FAILURE;
	}
	XGpio_SetDataDirection(&Gpio, DIP_CHANNEL, 0xff);
	XGpio_SetDataDirection(&Gpio, LED_CHANNEL, 0x00);

	DIP_State = XGpio_DiscreteRead(&Gpio,DIP_CHANNEL);
	XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, DIP_State);

	return XST_SUCCESS;
}
```

- “xstatus.h” library contains `XST_SUCCESS` and `XST_FAILURE` macros for status of the functions.
- “xgpio.h” library contains all APIs, types and definitions to use GPIO IP block.
- LED_CHANNEL can be 1 or 2 according to the connection.
- `DIP` _CHANNEL can be 1 or 2 according to the connection.
- “XGpio” is struct type definition for GPIO module, and it must be declared before calling any GPIO function, because all functions get this type as a parameter.
- `XGpio_Initialize()` function initializes a GPIO instance according to G`PIO_LED_DEVICE_ID`. All initialization functions return failure or success. This can be used when program does not work correctly, to understand where is the problem
- `XGpio_SetDataDirection()` sets the direction of the data. Third parameter of the function is the mask. And it is specifying which bits are input and which are outputs. Bits set

to 0 are output and bits set to 1 are inputs. In this case all LEDs are output and the DIP switches are input. If all bits are output or input there is no need to use this function.

- `XGpio_DiscreteRead()` function return the state of the DIP switches.
- `XGpio_DiscreteWrite()` function set the LEDs state as DIP states.

4.3. GPIO Interrupts with MicroBlaze

4.3.1. Overview of Interrupt Controller IP

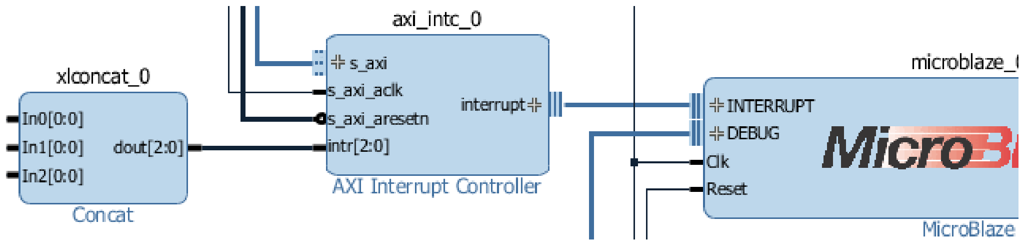
Some feature of Interrupt Controller is listed below:

- Supports up to 32 interrupts. It is cascadable to provide additional interrupt input.
- Priority between interrupt requests is determined by vector position. The LSB has the highest priority.
- It has Interrupt Enable Register for selectively enabling individual interrupts and Master Enable Register for enabling interrupt request output.
- Software interrupts and nested interrupts are supported.
- Each device connected to the AXI INTC core can use either normal or fast interrupt mode, based on the latency requirement. Fast interrupt mode can be chosen for designs requiring lower latency like Ethernet interrupts.
- Each input is configurable for edge or level sensitivity.

4.3.2. Hardware Part

In this part adding a Interrupt Controller IP to a MicroBlaze project will be discussed.

- First click on “Add New IP” button and select “AXI Interrupt Controller”
- If you have more than one interrupt source(most probably you have), then Concat IP is needed. Select Concat IP and configure the number of inputs according to the number of interrupt sources.
- Connect dout[] port of the Concat to intr[] port of Interrupt Controller.
- Connect the interrupt port of the Interrupt Controller to INTERRUPT port of the MicroBlaze.
- Connect the interrupt sources to the inputs of Concat IP.



- Then “Run Connection Automaton”.
- Generate bitstream, export hardware and overwrite to existing hardware.
- Open SDK.

4.3.3. Software Part

Following code shows how to use GPIO input interrupt with MicroBlaze.

```
#include "xparameters.h"
#include "xgpio.h"
#include "xil_exception.h"
#include "xintc.h"

// Definitions for GPIO
#define GPIO_DEVICE_ID            XPAR_GPIO_0_DEVICE_ID
#define BUTTON_CHANNEL      1
```

```
//Definitions for GPIO interrupt
#define INTC_GPIO_INTERRUPT_ID    XPAR_INTC_0_GPIO_0_VEC_ID
#define BUTTON_INTERRUPT XGPIO_IR_CH1_MASK
// Definitions for Interrupt controller
#define INTC_DEVICE_ID     XPAR_INTC_0_DEVICE_ID
//Function Prototypes
void GpioHandler(void *CallbackRef);

int main(void)
{
       XIntc IntcInstance; // Interrupt Controller Instance
       XGpio GPIOInstance;// GPIO Instance

       XGpio_Initialize(&GPIOInstance, GPIO_DEVICE_ID);

       XIntc_Initialize(&IntcInstance, INTC_DEVICE_ID);
       XIntc_Connect(&IntcInstance,
INTC_GPIO_INTERRUPT_ID,(Xil_ExceptionHandler)GpioHandler, &GPIOInstance);
       XIntc_Enable(&IntcInstance, INTC_GPIO_INTERRUPT_ID);
       XIntc_Start(&IntcInstance, XIN_REAL_MODE);

       XGpio_InterruptEnable(&GPIOInstance, BUTTON_INTERRUPT);
       XGpio_InterruptGlobalEnable(&GPIOInstance);

       Xil_ExceptionInit();
       Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,(void *)XIntc_InterruptHandler,
&GPIOInstance);
       Xil_ExceptionEnable();

       while(1){
       }
}
// Interrupt Function
void GpioHandler(void *CallbackRef)
{
       XGpio *GpioPtr = (XGpio *)CallbackRef;
       /*
        * USER CODES...
        */
       /* Clear the Interrupt */
       XGpio_InterruptClear(GpioPtr, BUTTON_INTERRUPT);      }
```

- “” and “” libraries contain APIs, types and definitions for interrupt and exceptions.
- `INTC_DEVICE_ID` is the device ID of the Interrupt Controller IP.
- `INTC_GPIO_INTERRUPT_ID` is the device ID of the GPIO interrupt.
- `BUTTON_INTERRUPT` the interrupt mask for GPIO channel. Interrupt request can come one of the two channels. It determines which channels interrupt is used.
- “XIntc” is a struct type for interrupt controller instances.
- `XIntc_Initialize()` function initializes the interrupt controller according to the `INTC_DEVICE_ID`.
- `XIntc_Connect()` function makes the connection between the ID of the interrupt source and the associated handler that is to run when the interrupt is recognized. Callbackref is used as the argument for the handler when it is called.
- `XIntc_Enable` () enables the interrupt source provided as the argument Id.
- `XIntc_Start()` starts the interrupt controller by enabling the output from the controller to the processor.
- `Xil_ExceptionInit()` initializes exception handling for the processor.
- `XGpio_InterruptEnable()` enables interrupts.
- `XGpio_InterruptGlobalEnable()` enables the interrupt output signal.
- `Xil_ExceptionRegisterHandler()` makes the connection between the Id of the exception source and the associated handler that is to run when the exception is recognized.

- `Xil_ExceptionEnable()` enables exceptions.

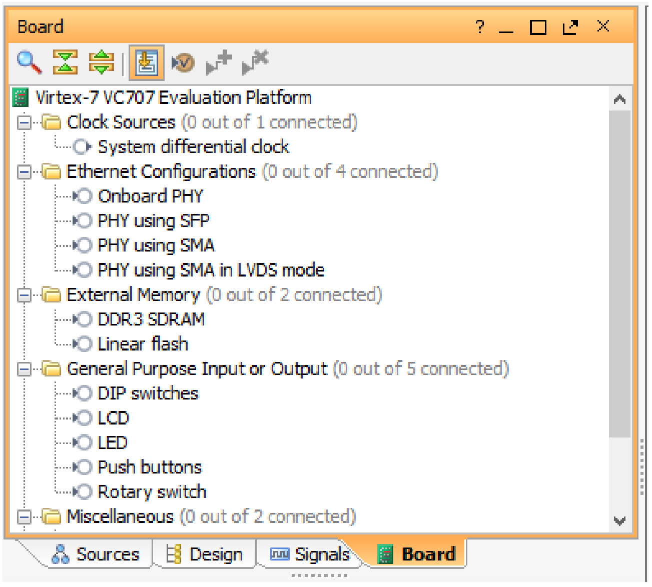
5. WEEK 4

In this week, I tried to control the DDR3 DRAM. I had some troubles about clock connections in the design so this took some time for me to finish.

5.1. Using DRAM and MIG with MicroBlaze

5.1.1. Hardware Part

In "Board" tab showing in the figure below, important peripherals on the VC707 evaluation board are showing. It is possible to select the peripheral and add it to the design by just double clicking on it.



Board

Virtex-7 VC707 Evaluation Platform
- Clock Sources (0 out of 1 connected)
  - System differential clock
- Ethernet Configurations (0 out of 4 connected)
  - Onboard PHY
  - PHY using SFP
  - PHY using SMA
  - PHY using SMA in LVDS mode
- External Memory (0 out of 2 connected)
  - DDR3 SDRAM
  - Linear flash
- General Purpose Input or Output (0 out of 5 connected)
  - DIP switches
  - LCD
  - LED
  - Push buttons
  - Rotary switch
- Miscellaneous (0 out of 2 connected)

Sources | Design | Signals | **Board**

- To add DDR3 SDRAM double click on it. Then click OK on the opening window. That wizard will create a Memory Interface Generator (MIG) to control DRAM. If instead of board, the FPGA chip is selected when opening the project then MIG configurations must be done manually.
- Do not click "Run Connection Automation" in this step.
- Add a MicroBlaze and click on "Run Block Automation".
- Set the desired configurations except "Clock Connection". Use "/mig_7series/ui_addn_clk_0" or "/mig_7series/ui_clk" as a clock source. MIG can be configured to generate more than one clock signal with desired frequency. "ui_addn_clk_x" pins will pop up when a new clock output is added. These clocks can be selected.
- Click OK. Add desired peripherals, and click on "Run Connection Automation".
- Generate Bitstream, export hardware, and open SDK to write C codes.

5.1.2. Software Part

Following code show how to write one-byte value to memory and read it.

```
int* DRAM_MEMORY = (int *)0x80000000;

	char dataVar;
	DRAM_MEMORY[1] = 'A'; // write data to memory
	dataVar = DRAM_MEMORY[1];//read data from memory
```

0x80000000 is the base address of the DRAM and DRAM_MEMORY is the pointer which point to this address. Any data can be written to the DRAM using this pointer. This method is not used often.

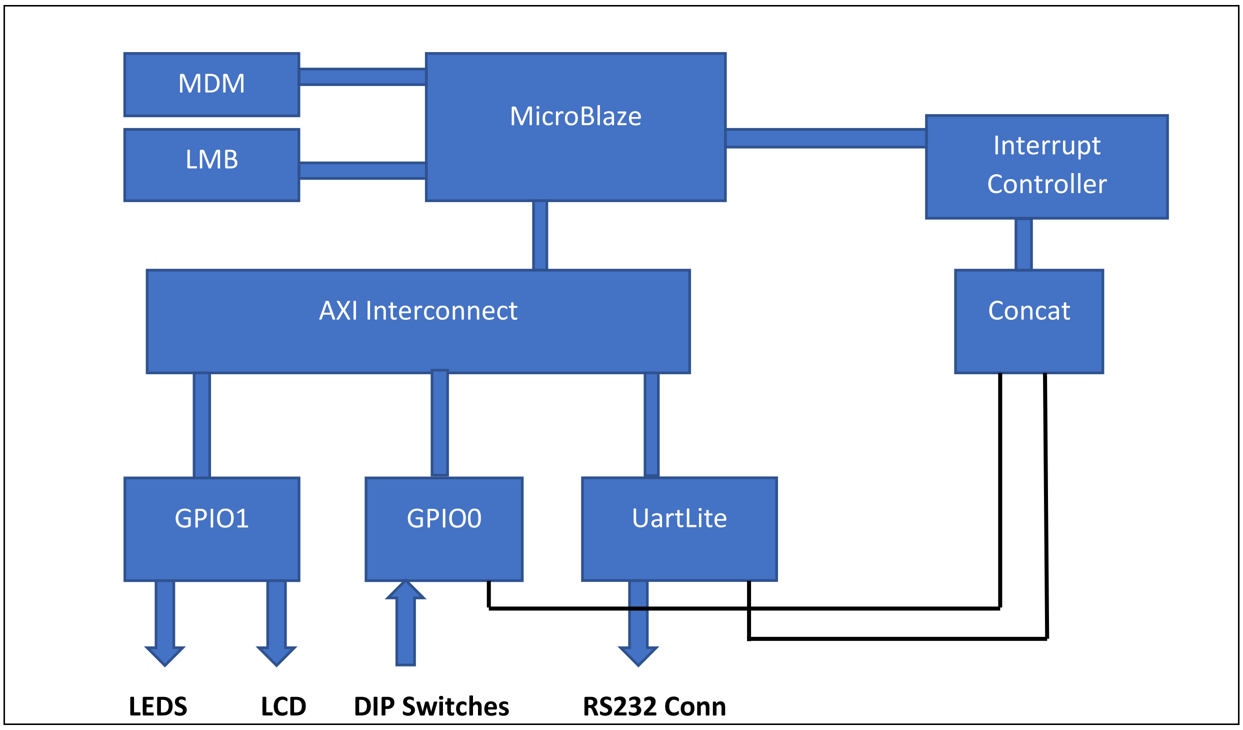
6. WEEK 5

In this week, I tried to make a connection between PC and FPGA via Uart to send commands to FPGA from PC and vice versa. I designed a MicroBlaze based hardware, embedded software, and a GUI.

6.1. Project 1: PC and FPGA Connection With UART

6.1.1. Hardware Part

In addition to basic MicroBlaze system two AXI GPIO block, one Uartlite block, and one interrupt controller are required. One of the GPIO block is for DIP input so it has to be dual channel disabled and interrupt enabled. The other block is for LCD and LEDs so it has to be dual channel enable, and there is no need for interrupt for outputs. I configure the Uartlite block as 115200 of baud rate, 8 data bits, and no parity. General connections are as follows.



6.1.2. Software Part

There are two interrupts in the software.

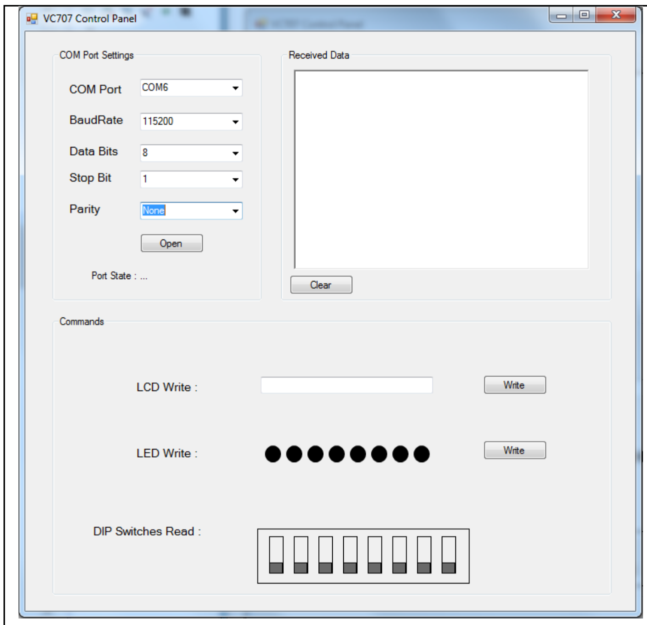
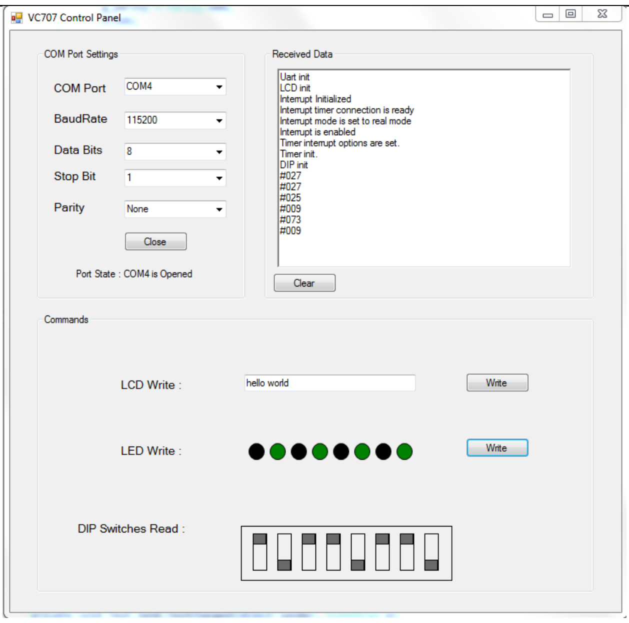
One of them is Uartlite interrupt occurring according to state of the RX and TX buffers. Interrupt is generated when receive FIFO becomes non-empty or when the transmit FIFO becomes empty. I use only the receive interrupt to get the command from the computer. There was a minor problem about the receive interrupt. That is receive FIFO must be empty for the interrupt to be finished, but the received data size cannot be estimated. To solve this problem, I decided to send the size of data first and then the data. There has to be communication protocol to understand the received command. I assign some unused characters to define a command. I used '#' character for LCD write command, '*' character for LED write command. After all, the whole packet received to FPGA consists 3 sections, first the length of the data and the command, second the command, and third is the data.

Other interrupt is GPIO interrupt occurring with the change in the level of the any switches. I used this interrupt to send the switches state to computer. I assign ‘#’ character as DIP switches state is changed, update the DIP switches command. When interrupt occurred FPGA first sends the character ‘#’ and then the 8-bit DIP switches state.

6.1.3. GUI

I designed a graphical user interface with C# to send commands to hardware on FPGA and to get the commands from FPGA. GUI consists of three group boxes.

- COM Port Settings: This part is used to make connection settings. Form is initialized with a default connection setting but they can be changed if they are not valid. After settings are valid then by clicking “Open” button, Com port can be opened. Port state will change as “<Port_name> is opened”.
- Received Data: This part is used to observe received data from the FPGA and some error messages. “Clear” button cleans the screen.
- Command: This part is used to send and receive commands. To write a string to the LCD display on the evaluation board, it is enough to write desired string to text box on the right of the LCD Write label and click on the Write button. Black circles represent the LEDs on the board. When clicked on the one of them, it will be green meaning that it is on. By just clicking on the circles LED status is selected and with the click on Write button selected status will appear on the LEDs on the board. The figure on the right side of the “DIP Switches Read” label show the status of the DIP switches on the board. When an interrupt asserted by the DIP switches, status of the switches is send with the DIP read command and the DIP status will appear on this part.

 VC707 Control Panel COM Port Settings COM Port: COM6 BaudRate: 115200 Data Bits: 8 Stop Bit: 1 Parity: None Port State : ... Received Data Commands LCD Write : LED Write : ●●●●●●●● DIP Switches Read :	 VC707 Control Panel COM Port Settings COM Port: COM4 BaudRate: 115200 Data Bits: 8 Stop Bit: 1 Parity: None Port State : COM4 is Opened Received Data Uart init LCD init Interrupt Initialized Interrupt timer connection is ready Interrupt mode is set to real mode Interrupt is enabled Timer interrupt options are set. Timer init. DIP init #027 #027 #025 #009 #073 #009 Commands LCD Write : hello world LED Write : ●🟢●🟢●🟢●🟢 DIP Switches Read :
When the Program is opened.	A working example.

7. WEEK 6

In this week, I tried to make a connection between FPGA and PC via Ethernet to send commands from PC to FPGA. I designed a hardware to use lwIP library, then set a server to FPGA, and with a client GUI I send commands to FPGA.

7.1. Project 2: PC and FPGA Connection with Ethernet

Before diving into the design part for Ethernet usage with Microblaze, some information must be given. This information will be useful for software part. Hardware design has some constraints to run the software and these constraints will be discussed under these headers.

7.1.1. What is lwIP (light weight IP) ?

The lwIP is an open source TCP/IP protocol suite available under the BSD license. The lwIP is a standalone stack; there are no operating systems dependencies, although it can be used along with operating systems. LwIP support IP, ICMP, UDP, TCP, ARP, DHCP, IGMP protocols.

LwIP libraries provides adapters for the AXI_Ethernetlite, the TEMAC (which will be used in this design), and the Gigabit Ethernet Controller. The library can run on MicroBlaze and ARM Cortex - A9 processors. To work with lwIP some hardware and software are required. LwIP functions are used with real time operating systems. Old resources say available library for MicroBlaze is xil_kernel for Microblaze, but now FreeRTOS can be used and is recommended. Hardware requirements are as follows:

- **Processor**: Either a MicroBlaze or a Cortex-A9 processor. The Cortex-A9 processor applies to Zynq systems.
- **MAC**: LwIP supports axi_ethernetlite, axi_ethernet, and Gigabit Ethernet controller and MAC (GigE) cores.
- **Timer**: to maintain TCP timers, lwIP raw API based applications require that certain functions are called at periodic intervals by the application. An application can do this by registering an interrupt handler with a timer.
- **DMA**: For axi_ethernet based systems, the axi_ethernet cores can be configured with a soft DMA engine or a fifo interface. For GigE-based Zynq systems, there is a built-in DMA and so no extra configuration is needed. Same applies to axi_ethernetlite based systems, which have their built-in buffer management provisions.

7.1.2. FreeRTOS

Before using lwIP functions concept of RTOS must be understood. In this section basics of FreeRTOS is mentioned.

- Following libraries must be included to use FreeRTOS in projects.

```
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "timers.h"
```

- To create a new task xTaskCreate function is used as follows

```
xTaskCreate( nameOfTheTask, textNameForTheTask, StackSizeToBeAllocatedForTask,
                TaskParameter, TastPriority, TaskHandlePointer);
```

- Tasks must be in infinite loop. The task below prints “Hello” to screen in every second. `pdMS_TO_TICKS` converts the time value in millisecond to tick for timer. `TickType_t` is a type definition for timer ticks and it is 32 bits or 16 bits unsigned integer according to the processor.

```
static void prvTxTask( void *pvParameters )
{
const TickType_t x1second = pdMS_TO_TICKS( 1000 );
      for( ;; )
      {
            /* Delay for 1 second. */
            vTaskDelay( x1second );
            xil_printf("Hello\n");
      }
}
```

- Queues are used to provide communication between tasks. They are created by using xQueueCreate() function. This function takes two parameters. First parameter is the space in the queue and the second is the size of data to hold. Following example created a queue called xQueue, size of integer with 2 spaces.

```
static QueueHandle_t xQueue = NULL;
xQueue = xQueueCreate(  2,sizeof(int) );
```

- To read data in the queue `xQueueReceive()` function is used with 3 parameters which are accordingly the queue being read, pointer to hold read data, time to wait the queue. Following example waits for the data to be read from xQueue to ReceivedData 1000 ticks.

```
      xQueueReceive(xQueue,Recdstring, 1000);
```

- To write data to the queue `xQueueReceive()` function is used with 3 parameters which are accordingly the queue being written, pointer of the data being written, and block time.

```
Char SendData = ‘A’;
      xQueueSend(xQueue, SendData, 0UL );
```

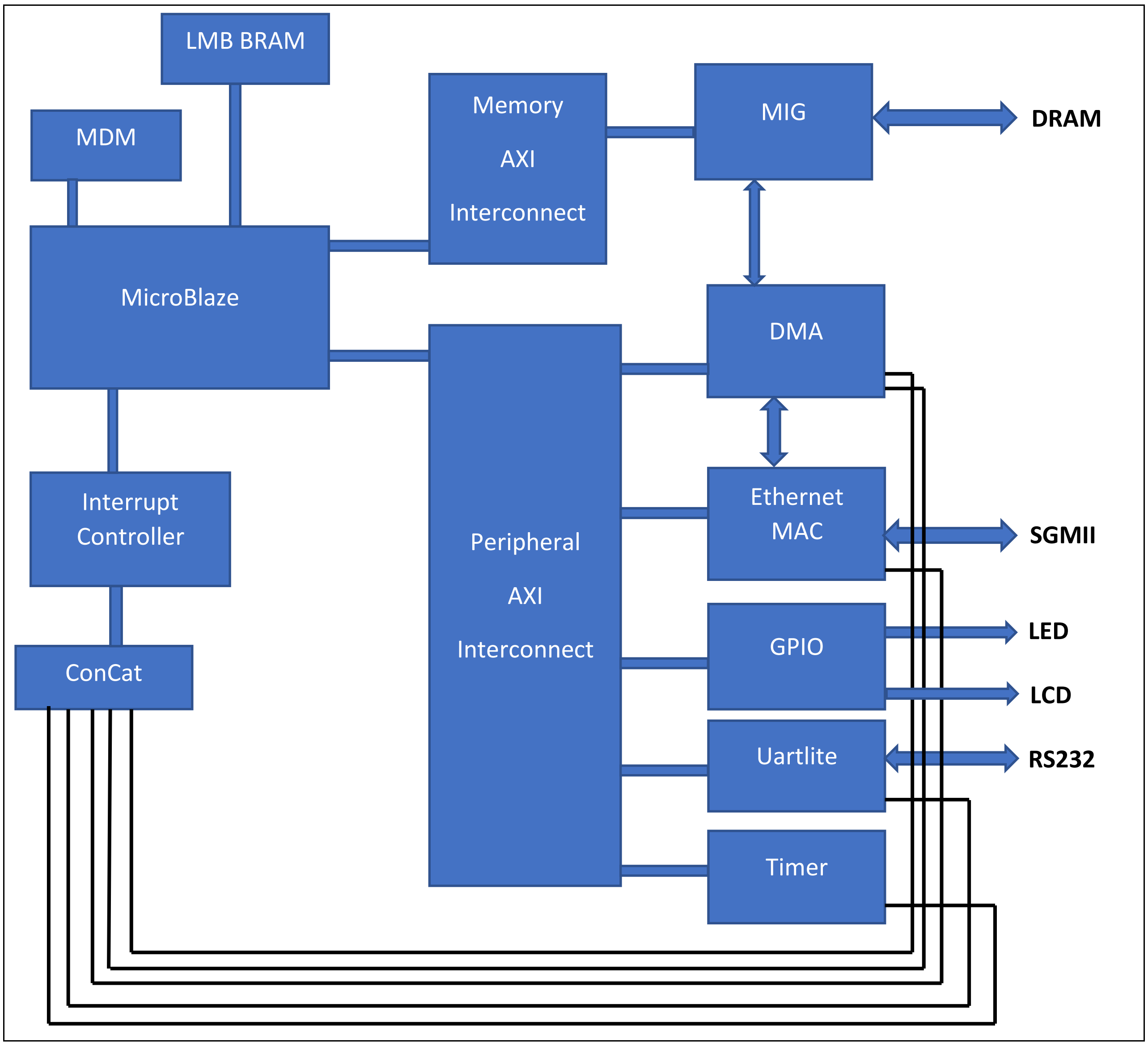
- Timer is created using `xTimerCreate`() function 5 parameter which are accordingly timer name to use to assist in debugging, period in ticks, auto reload option true or false, timer ID and finally the name of callback function.

```
static TimerHandle_t xTimer = NULL;
xTimer = xTimerCreate( (const char *) "Timer",x10seconds,
                        pdFALSE, (void *) TIMER_ID, vTimerCallback);
```

- Timer is initialized with `xTimerStart( xTimerHandler, Blocktime );` function.
- Task are initialized with `vTaskStartScheduler();` function.
- Tasks are closed with `vTaskDelete( xTaskHandle );` function.

7.1.3. Hardware Part

As it is mentioned in 7.1.1. part processor system, ethernet MAC, DMA, and timer are basic requirements to use lwIP library. In addition to these hardwares I used GPIO block, and AXI_Uartlite to perform some tasks. Hardware is as follow in general.



7.1.4. Software Part

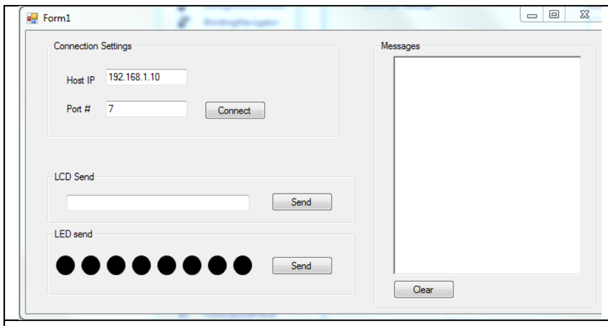
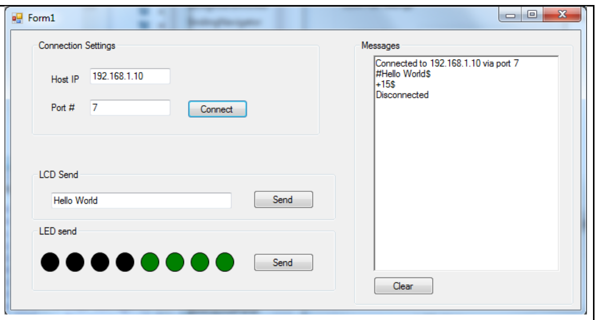
Software on the FPGA is a basic echo server written with lwIP libraries. Software sends all received data back, but when some special characters are received it performs writing a string to LCD task and changing LED status task. (‘#’ character is for LCD and ‘*’ character for LED). Beyond command characters I used ‘$’ character to specify the end of the send data.

7.1.5. GUI

I designed a graphical user interface with C# to send commands to hardware on FPGA and to get the commands from FPGA. GUI consists of four parts as it in the previous project.

- Connection Settings: This part is used to make connection settings. Form is initialized with a default Host IP, and port. To use different port and IP software on the FPGA must be changed.

- Messages: This part is used to observe received data from the FPGA and some messages. “Clear” button cleans the screen.
- LCD Send: This part is used to send LCD write command to the evaluation board. To write a string to the LCD display, it is enough to write desired string to text box and click on the Send button.
- LED Send: Black circles represent the LEDs on the board. When clicked on the one of them, it will be green meaning that it is on. By just clicking on the circles LED status is selected and with the click on Write button, selected status will appear on the LEDs on the board.

	
When the program is opened.	A working example.

```
Connected to 192.168.1.10 via port 7
#Hello World$
+15$
Disconnected
```

8. REFERENCES

- Getting Started with the Virtex-7 FPGA VC707 Evaluation Kit
- VC707 Evaluation Board for the Virtex-7 FPGA User Guide
- MicroBlaze Processor Reference Guide
- Tri-Mode Ethernet MAC v5.2 User Guide
- Microblaze Debug Module v3.2 Product Guide
- Clocking Wizard v5.3 Product Guide
- AXI Interrupt Controller v4.1 Product Guide
- AXI UART Lite v2.0 Product Guide
- AXI GPIO v2.0 Product Guide
- LMB BRAM Interface Controller v4.0 Product Guide
- AXI Timer v2.0 Product Guide
- Processor System Reset Module v5.0 Product Guide
- https://www.xilinx.com/video/hardware/microblaze-overview.html
- https://www.xilinx.com/video/software/embedded-design-with-the-microblaze-soft-processor-core.html
- https://www.xilinx.com/video/hardware/the-microblaze-microcontroller-system.html