Oakland University
School of Electrical & Computer Engineering
Fall 2022
ECE 6740
Advance Embedded System Design

Lab #1

Submitted by:
Pradeep Potturi

Date: 25/01/2024

Lab 1 ECE 6740 01/25/2024

PART A:

Create a new Vivado project and create a block design with the ZYNQ-7 core and the UART enabled to use for *stdio*. Disable the timers and other items that we will not be using in this lab. Synthesize and Implement this design. Export the design to the SDK. Create a board support package (BSP) in the SDK and sample 'Hello World' application. Modify the 'Hello World' application to perform the following calculation and print the *answer* to the screen (using the UART as *stdio*, of course).

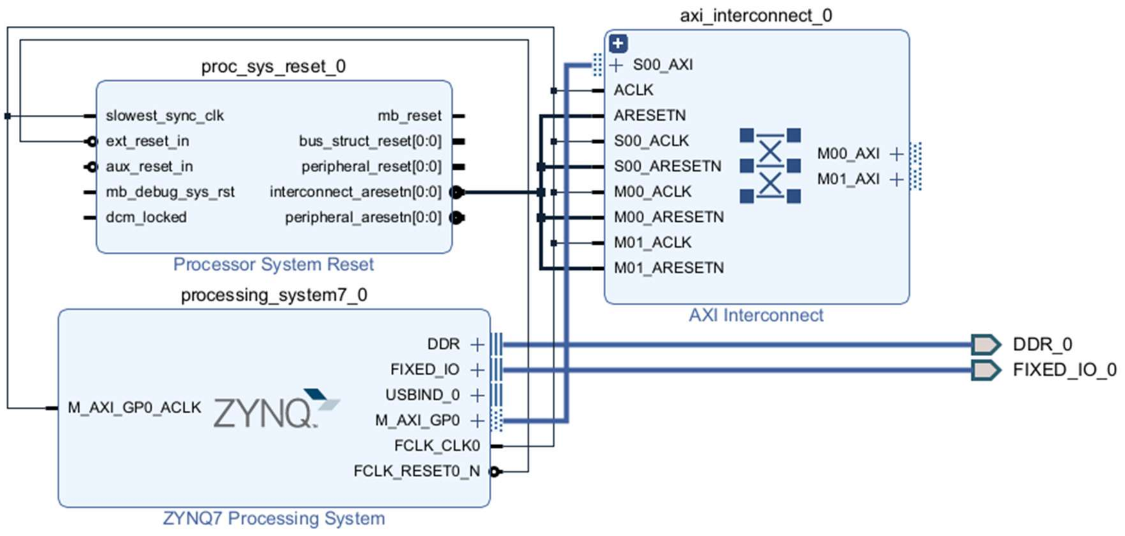
```
int a, b, answer;
a = 32; b = 41;
answer = a + b;
```

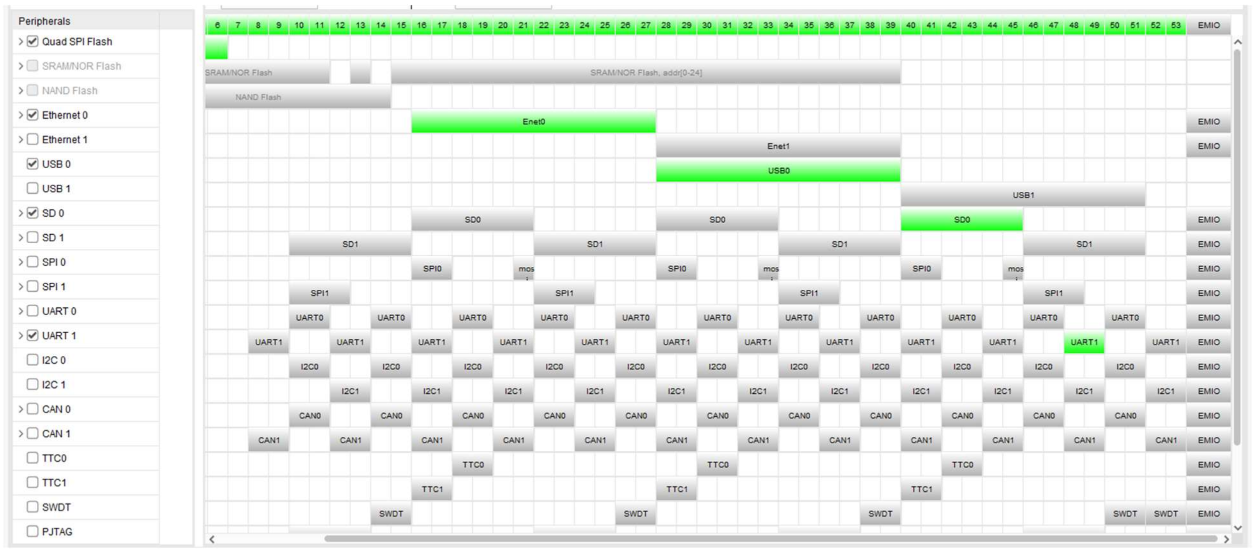
Download the system to the board and test it using the built-in terminal program.

Response:

Vivado:

1. Created New Vivado Project using Sample project from Youtube Link
2. Added following IP
   a. Zynq7 Processing System
   b. Processor System Reset
   c. AXI Interconnect
3. Connected Clk and Reset lines
4. Run Block connections
5. Verify UART is connected (double-click on ZYNQ7 block)

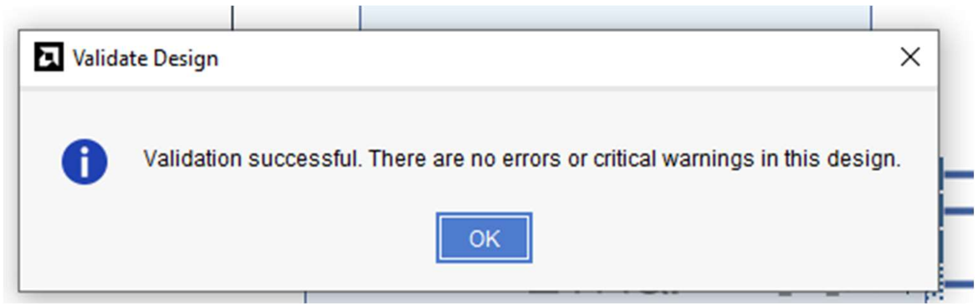




Peripherals

- [✓] Quad SPI Flash
- [ ] SRAM/NOR Flash
- [ ] NAND Flash
- [✓] Ethernet 0
- [ ] Ethernet 1
- [✓] USB 0
- [ ] USB 1
- [✓] SD 0
- [ ] SD 1
- [ ] SPI 0
- [ ] SPI 1
- [ ] UART 0
- [✓] UART 1
- [ ] I2C 0
- [ ] I2C 1
- [ ] CAN 0
- [ ] CAN 1
- [ ] TTC0
- [ ] TTC1
- [ ] SWDT
- [ ] PJTAG

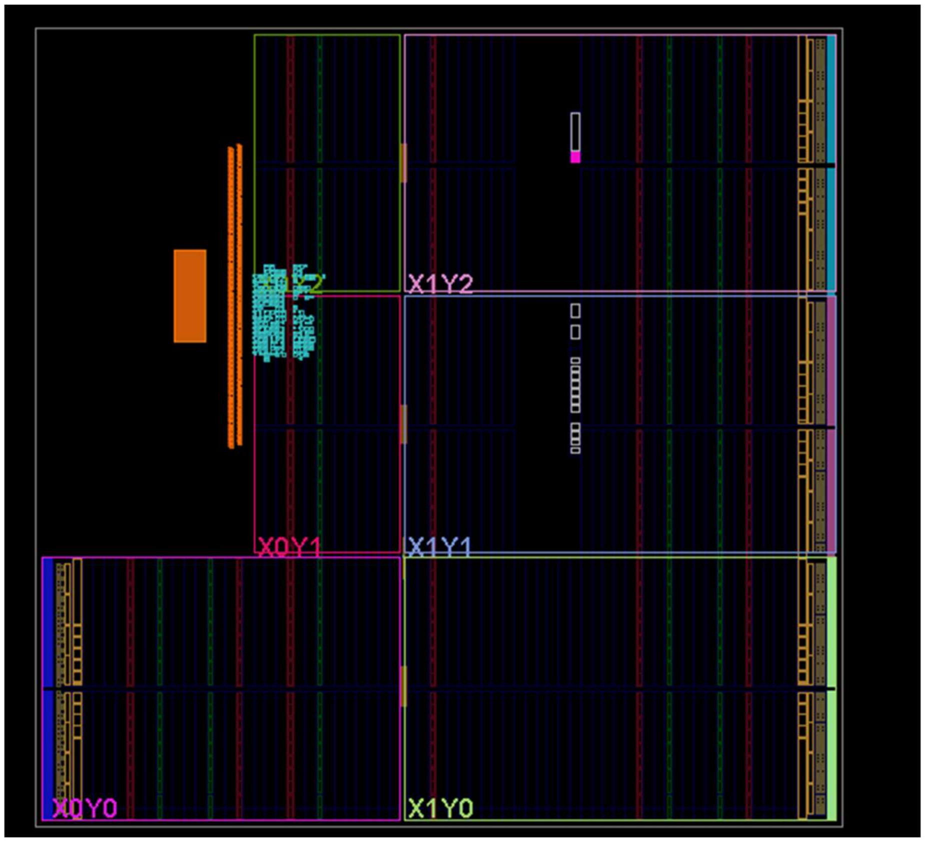
6. Validate design



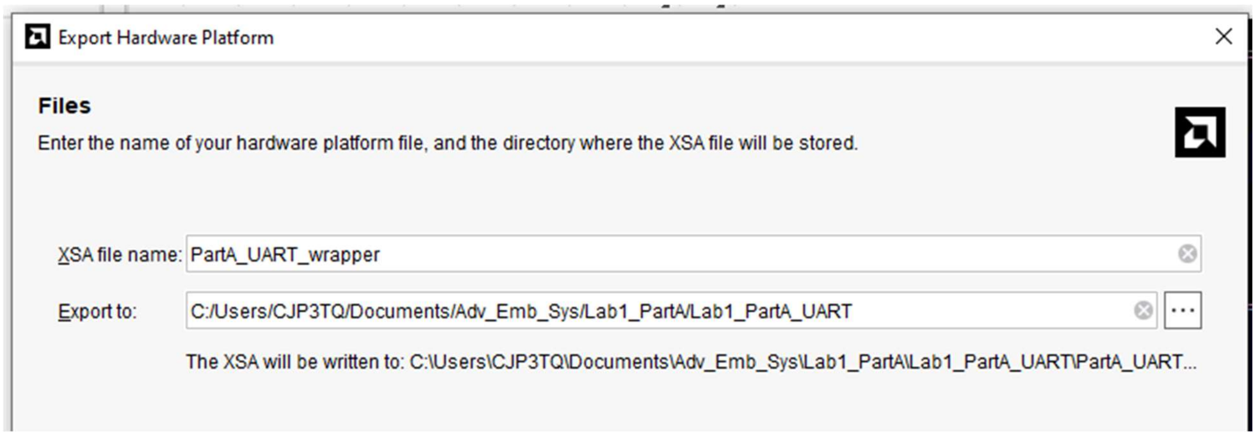
Validate Design

Validation successful. There are no errors or critical warnings in this design.

7. Generate bitstream to be used for SDK
8. Fabric Generated



9. Export Hardware to be used in Vitis



Files

Enter the name of your hardware platform file, and the directory where the XSA file will be stored.

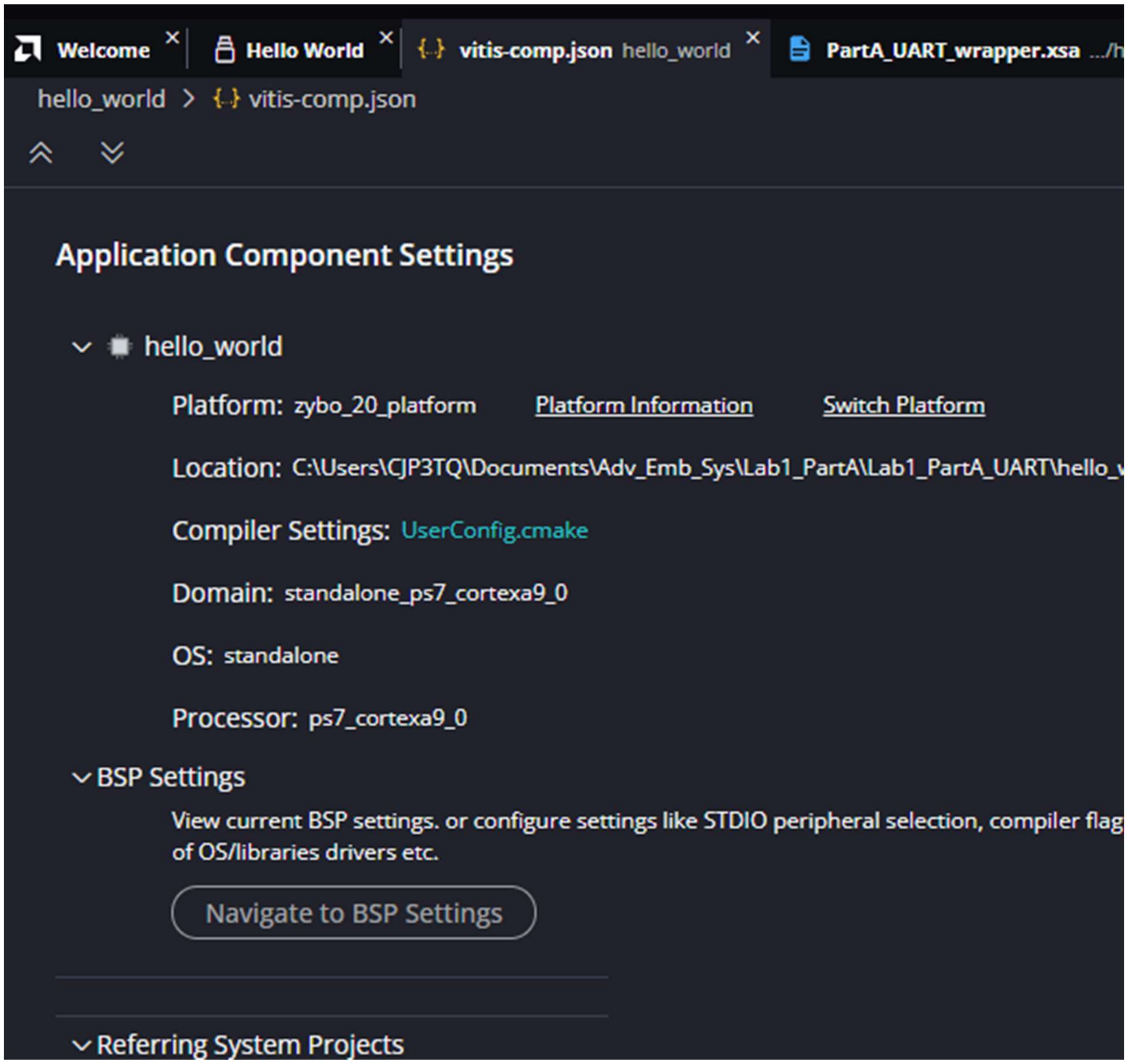
XSA file name: PartA_UART_wrapper

Export to: C:/Users/CJP3TQ/Documents/Adv_Emb_Sys/Lab1_PartA/Lab1_PartA_UART

The XSA will be written to: C:\Users\CJP3TQ\Documents\Adv_Emb_Sys\Lab1_PartA\Lab1_PartA_UART\PartA_UART...

Vitis:

1. Create New Platform
    a. Use .xsa file generated in Vivado
2. Create Application using Hello World example
3. Modify helloworld.c to add two number
4. Build and fix any warnings
5. Navigate BSP


Welcome
Hello World
vitis-comp.json hello_world
PartA_UART_wrapper.xsa .../h
hello_world > vitis-comp.json

Application Component Settings

hello_world

Platform: zybo_20_platform   Platform Information   Switch Platform

Location: C:\Users\CJP3TQ\Documents\Adv_Emb_Sys\Lab1_PartA\Lab1_PartA_UART\hello_w

Compiler Settings: UserConfig.cmake

Domain: standalone_ps7_cortexa9_0

OS: standalone

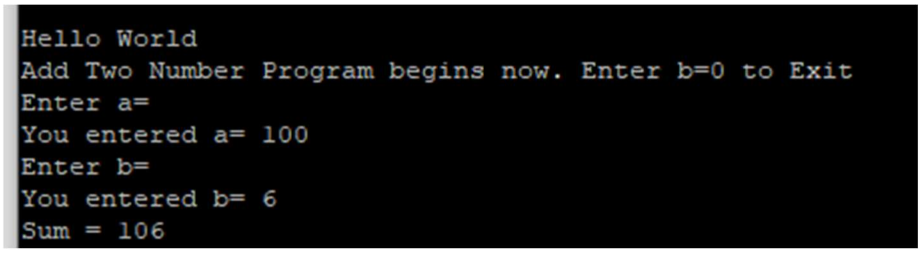
Processor: ps7_cortexa9_0

BSP Settings

View current BSP settings. or configure settings like STDIO peripheral selection, compiler flag of OS/libraries drivers etc.

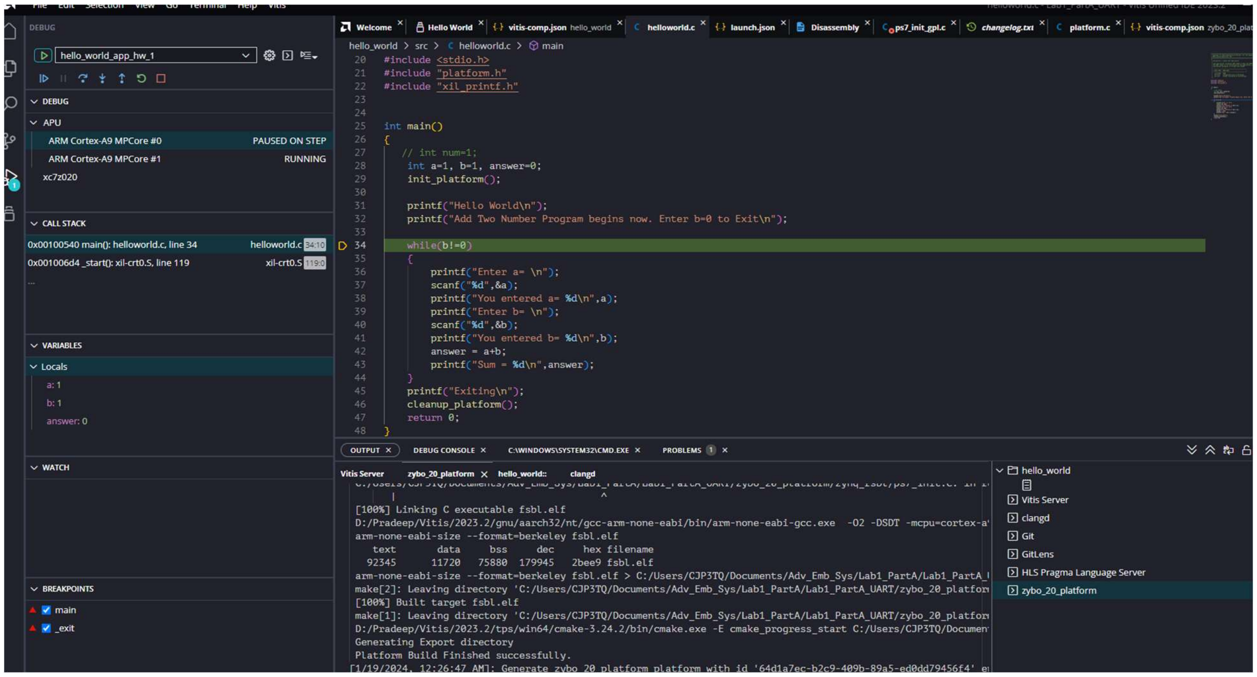
Run:

1. Open Putty for Serial communication
2. Click on ‘Run’ to Flash and execute code
3. Verify your code



```
Hello World
Add Two Number Program begins now. Enter b=0 to Exit
Enter a=
You entered a= 100
Enter b=
You entered b= 6
Sum = 106
```

Debug:



DEBUG: hello_world_app_hw_1

APU
- ARM Cortex-A9 MPCore #0 — PAUSED ON STEP
- ARM Cortex-A9 MPCore #1 — RUNNING
- xc7z020

CALL STACK
- 0x00100540 main(): helloworld.c, line 34 — helloworld.c 34:10
- 0x001006d4 _start(): xil-crt0.S, line 119 — xil-crt0.S 119:0

VARIABLES — Locals
- a: 1
- b: 1
- answer: 0

BREAKPOINTS
- main
- _exit

```c
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"


int main()
{
   // int num=1;
    int a=1, b=1, answer=0;
    init_platform();

    printf("Hello World\n");
    printf("Add Two Number Program begins now. Enter b=0 to Exit\n");

    while(b!=0)
    {
        printf("Enter a= \n");
        scanf("%d",&a);
        printf("You entered a= %d\n",a);
        printf("Enter b= \n");
        scanf("%d",&b);
        printf("You entered b= %d\n",b);
        answer = a+b;
        printf("Sum = %d\n",answer);
    }
    printf("Exiting\n");
    cleanup_platform();
    return 0;
}
```

```
[100%] Linking C executable fsbl.elf
D:/Pradeep/Vitis/2023.2/gnu/aarch32/nt/gcc-arm-none-eabi/bin/arm-none-eabi-gcc.exe  -O2 -DSDT -mcpu=cortex-a'
arm-none-eabi-size --format=berkeley fsbl.elf
   text     data      bss      dec      hex filename
  92345    11720    75880   179945    2bee9 fsbl.elf
arm-none-eabi-size --format=berkeley fsbl.elf > C:/Users/CJP3TQ/Documents/Adv_Emb_Sys/Lab1_PartA/Lab1_PartA_
make[2]: Leaving directory 'C:/Users/CJP3TQ/Documents/Adv_Emb_Sys/Lab1_PartA/Lab1_PartA_UART/zybo_20_platfor
[100%] Built target fsbl.elf
make[1]: Leaving directory 'C:/Users/CJP3TQ/Documents/Adv_Emb_Sys/Lab1_PartA/Lab1_PartA_UART/zybo_20_platfor
D:/Pradeep/Vitis/2023.2/tps/win64/cmake-3.24.2/bin/cmake.exe -E cmake_progress_start C:/Users/CJP3TQ/Documen
Generating Export directory
Platform Build Finished successfully.
[1/19/2024, 12:26:47 AM]: Generate zybo_20_platform platform with id '64d1a7ec-b2c9-409b-89a5-ed0dd79456f4' e
```

<u>PART B:</u>

Create a new Vivado project and create an entity and architecture for a 2-input 32-bit adder. Don't worry about an input nor output carry. Create an AXI interface IP package with 4 interface registers (we will only use 3 registers) and an IP package for your adder component. Modify the AXI interface package to:

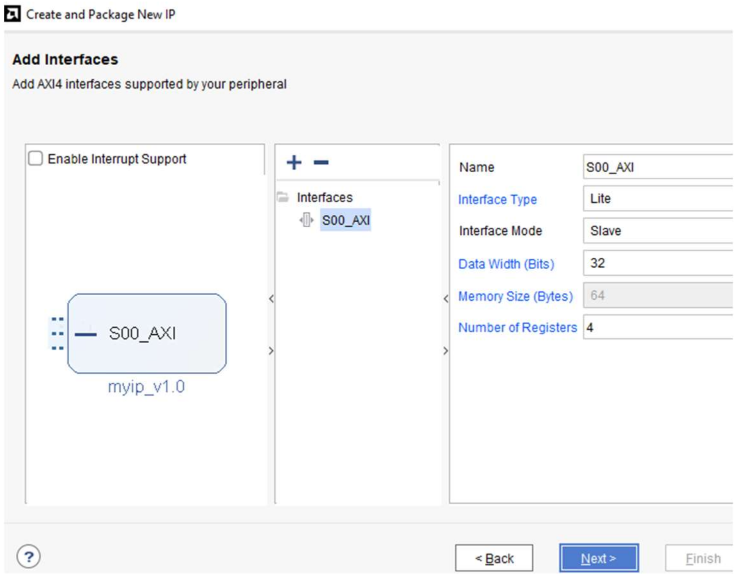
1. incorporate your adder where two of the interface registers will be input to the adder and the output from the adder will be input to the third interface register.
2. connect the adder output separately out of the AXI package so that it can be output to the LEDs on the general IO pins.

Import your adder/AXI IP into the project from part A. Connect it to the ZYNQ-7 core using an AXI interface. Connect the least significant bits of the adder's output to the LEDs. Synthesize and implement your design and export it to the SDK. Create a BSP and sample application. Modify the application to write the 32 and 41 values to the appropriate registers and read the answer back from the third memory-mapped register. Download the system to the board and display the result on the screen through the UART *stdio*. Verify that the answer is correct and that the least significant bits are correctly output to the LEDs.

Response:

1. Created New Vivado Project

2. Create AXI IP
    a. Menu->tools->Create and Package new IP
    b. Create AXI4 Peripheral
    c. Click next until you see Interface option



Create and Package New IP

Add Interfaces

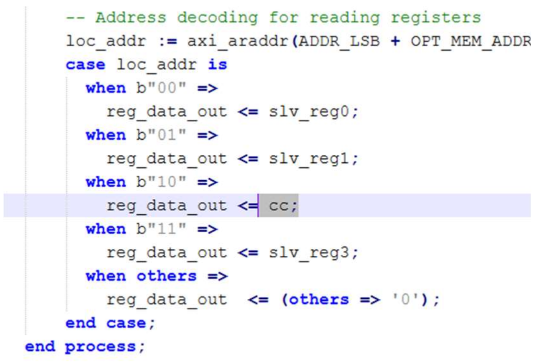
Add AXI4 interfaces supported by your peripheral

☐ Enable Interrupt Support

Interfaces: S00_AXI

Name	S00_AXI
Interface Type	Lite
Interface Mode	Slave
Data Width (Bits)	32
Memory Size (Bytes)	64
Number of Registers	4

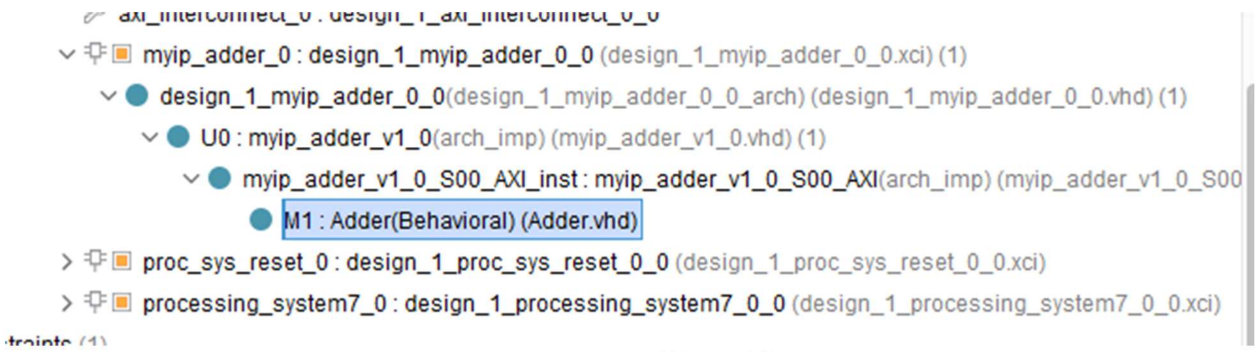
3. Modify the following in the myip_adder AXI IP
    a. Myip_adder_v1_0_S00_AXI.vhd
        i. Add user port LED
        ii. Add user signal to save "sum" result
        iii. Add Component "Adder"
        iv. At the user logic, call Adder component by passing values received from Salve registers 0 &1 and modify Slave read case statement to return "sum"
        v. Create a signal 'cc' to save Sum and pass it to LED port to display LSB 4bits



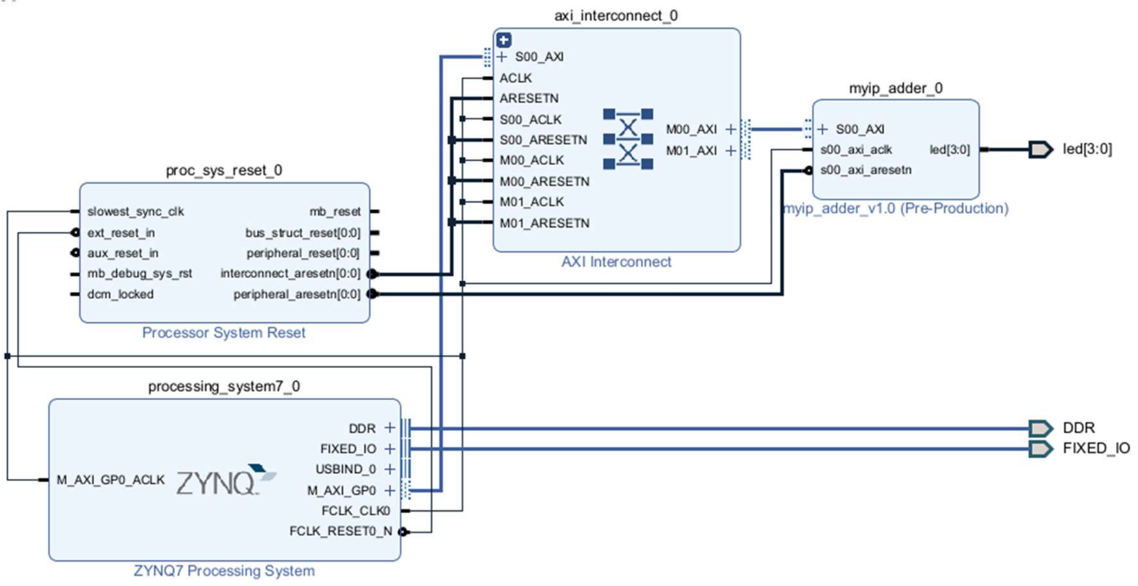
```
    -- Address decoding for reading registers
    loc_addr := axi_araddr(ADDR_LSB + OPT_MEM_ADDR
    case loc_addr is
      when b"00" =>
        reg_data_out <= slv_reg0;
      when b"01" =>
        reg_data_out <= slv_reg1;
      when b"10" =>
        reg_data_out <= cc;
      when b"11" =>
        reg_data_out <= slv_reg3;
      when others =>
        reg_data_out  <= (others => '0');
    end case;
end process;
```

    b. Adder.vhd
        i. Create Adder logic to add two 32 bit numbers and return Sum result

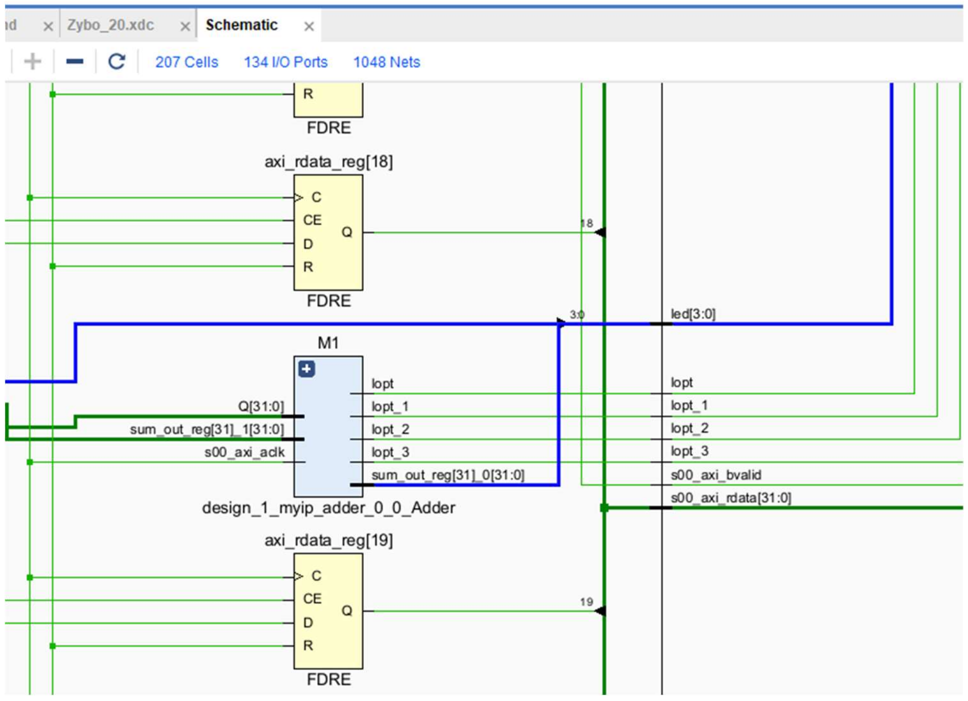
4. Add custom AXI (with Adder) as component to main project

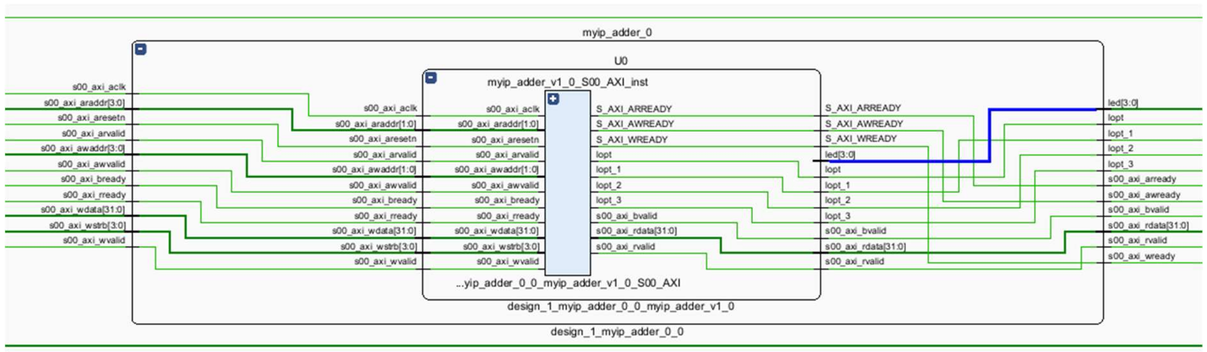


```
axi_interconnect_0 : design_1_axi_interconnect_0_0
myip_adder_0 : design_1_myip_adder_0_0 (design_1_myip_adder_0_0.xci) (1)
  design_1_myip_adder_0_0(design_1_myip_adder_0_0_arch) (design_1_myip_adder_0_0.vhd) (1)
    U0 : myip_adder_v1_0(arch_imp) (myip_adder_v1_0.vhd) (1)
      myip_adder_v1_0_S00_AXI_inst : myip_adder_v1_0_S00_AXI(arch_imp) (myip_adder_v1_0_S00
        M1 : Adder(Behavioral) (Adder.vhd)
proc_sys_reset_0 : design_1_proc_sys_reset_0_0 (design_1_proc_sys_reset_0_0.xci)
processing_system7_0 : design_1_processing_system7_0_0 (design_1_processing_system7_0_0.xci)
traints (1)
```



5. Generate bitstream
6. Observe the schematics to ensure LEDs are properly connected

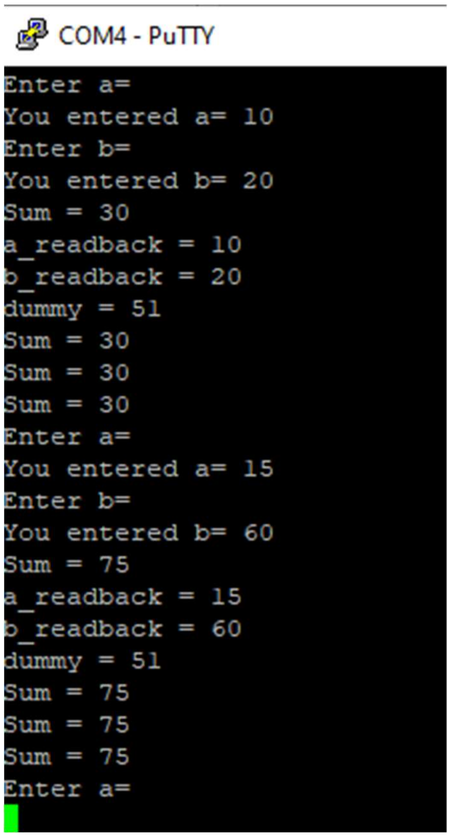




7. Export Hardware and Save as .xsa file

Vitis:

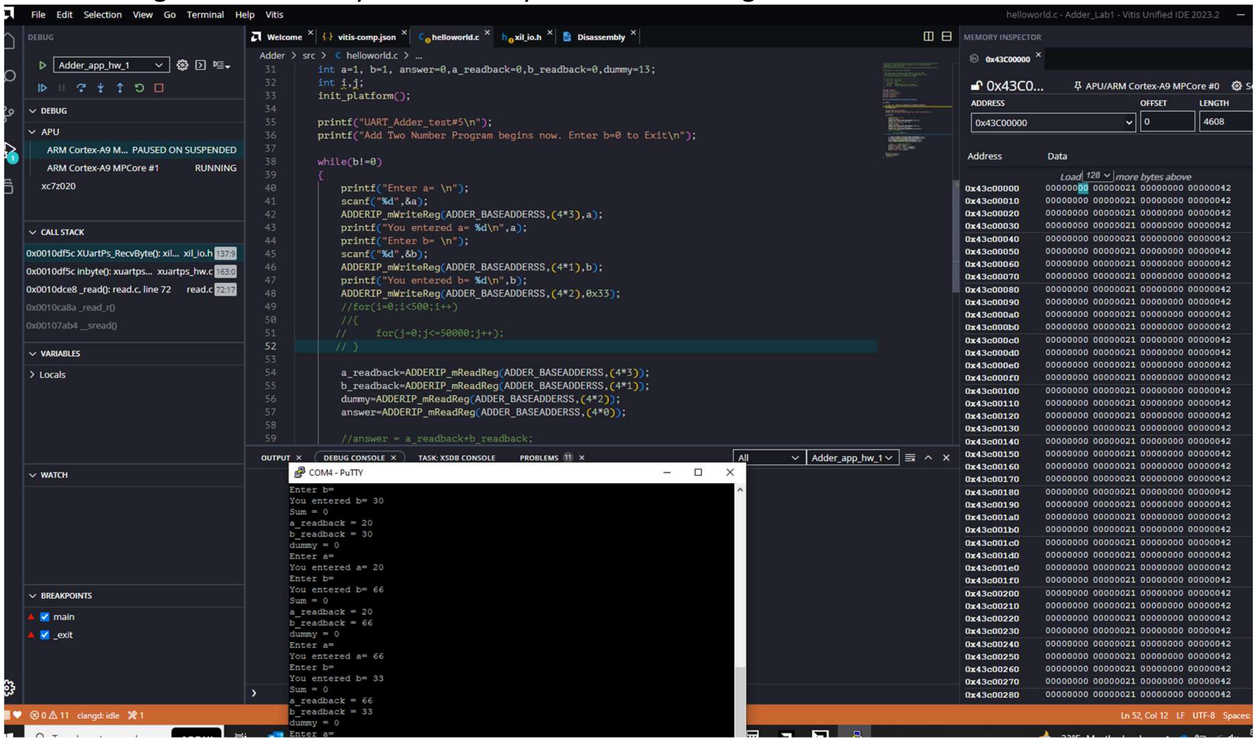
1. Create new Platform with .xsa file from previous step
2. Add Hello world application.
3. Modify the code to adder calculator (Actual .c code at the end of the document)
4. Ask user to input two values and push it to AXI port address
5. Print Sum result by reading from AXI port


COM4 - PuTTY

```
Enter a=
You entered a= 10
Enter b=
You entered b= 20
Sum = 30
a_readback = 10
b_readback = 20
dummy = 51
Sum = 30
Sum = 30
Sum = 30
Enter a=
You entered a= 15
Enter b=
You entered b= 60
Sum = 75
a_readback = 15
b_readback = 60
dummy = 51
Sum = 75
Sum = 75
Sum = 75
Enter a=
```

Issues I faced:

1. Only two AXI Slave registers are working:
   For some reason I see only two slave registers are working. When I tried to switch to other two then the other two works. Not sure what is the wrong in the AXI IP that I created.
   Below figure shows Putty and Memory window in debug mode.



```
0x0010df5c XUartPs_RecvByte(): xil... xil_io.h 137:9
0x0010df5c inbyte(): xuartps... xuartps_hw.c 163:0
0x0010dce8 _read(): read.c, line 72    read.c 72:17
0x0010ca8a _read_r()
0x00107ab4 __sread()
```

```c
    int a=1, b=1, answer=0,a_readback=0,b_readback=0,dummy=13;
    int i,j;
    init_platform();

    printf("UART_Adder_test#5\n");
    printf("Add Two Number Program begins now. Enter b=0 to Exit\n");

    while(b!=0)
    {
        printf("Enter a= \n");
        scanf("%d",&a);
        ADDERIP_mWriteReg(ADDER_BASEADDERSS,(4*3),a);
        printf("You entered a= %d\n",a);
        printf("Enter b= \n");
        scanf("%d",&b);
        ADDERIP_mWriteReg(ADDER_BASEADDERSS,(4*1),b);
        printf("You entered b= %d\n",b);
        ADDERIP_mWriteReg(ADDER_BASEADDERSS,(4*2),0x33);
        //for(i=0;i<500;i++)
        //{
       //     for(j=0;j<=50000;j++);
       // }

        a_readback=ADDERIP_mReadReg(ADDER_BASEADDERSS,(4*3));
        b_readback=ADDERIP_mReadReg(ADDER_BASEADDERSS,(4*1));
        dummy=ADDERIP_mReadReg(ADDER_BASEADDERSS,(4*2));
        answer=ADDERIP_mReadReg(ADDER_BASEADDERSS,(4*0));

        //answer = a_readback+b_readback;
```

```
Enter b=
You entered b= 30
Sum = 0
a_readback = 20
b_readback = 30
dummy = 0
Enter a=
You entered a= 20
Enter b=
You entered b= 66
Sum = 0
a_readback = 20
b_readback = 66
dummy = 0
Enter a=
You entered a= 66
Enter b=
You entered b= 33
Sum = 0
a_readback = 66
b_readback = 33
dummy = 0
Enter a=
```

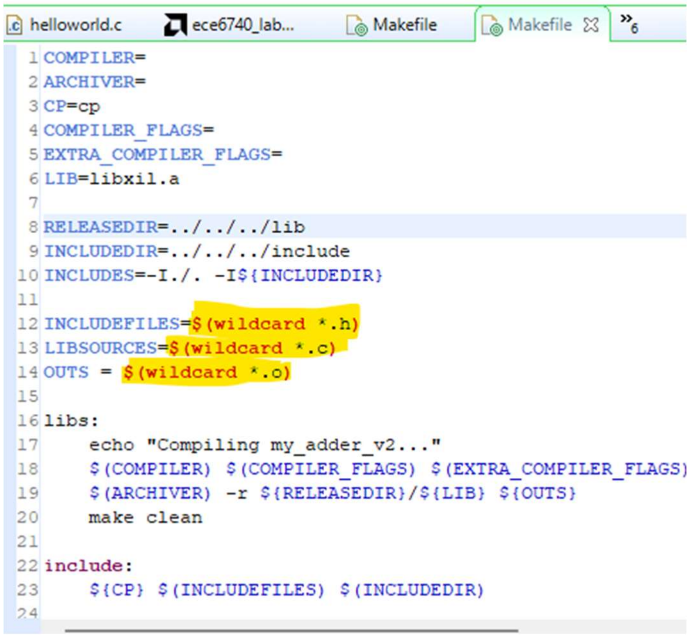
```
ADDRESS 0x43C00000   OFFSET 0   LENGTH 4608

Address       Data
0x43c00000    00000000 00000021 00000000 00000042
0x43c00010    00000000 00000021 00000000 00000042
0x43c00020    00000000 00000021 00000000 00000042
0x43c00030    00000000 00000021 00000000 00000042
0x43c00040    00000000 00000021 00000000 00000042
0x43c00050    00000000 00000021 00000000 00000042
0x43c00060    00000000 00000021 00000000 00000042
0x43c00070    00000000 00000021 00000000 00000042
0x43c00080    00000000 00000021 00000000 00000042
0x43c00090    00000000 00000021 00000000 00000042
0x43c000a0    00000000 00000021 00000000 00000042
0x43c000b0    00000000 00000021 00000000 00000042
0x43c000c0    00000000 00000021 00000000 00000042
0x43c000d0    00000000 00000021 00000000 00000042
0x43c000e0    00000000 00000021 00000000 00000042
0x43c000f0    00000000 00000021 00000000 00000042
0x43c00100    00000000 00000021 00000000 00000042
0x43c00110    00000000 00000021 00000000 00000042
0x43c00120    00000000 00000021 00000000 00000042
0x43c00130    00000000 00000021 00000000 00000042
0x43c00140    00000000 00000021 00000000 00000042
0x43c00150    00000000 00000021 00000000 00000042
0x43c00160    00000000 00000021 00000000 00000042
0x43c00170    00000000 00000021 00000000 00000042
0x43c00180    00000000 00000021 00000000 00000042
0x43c00190    00000000 00000021 00000000 00000042
0x43c001a0    00000000 00000021 00000000 00000042
0x43c001b0    00000000 00000021 00000000 00000042
0x43c001c0    00000000 00000021 00000000 00000042
0x43c001d0    00000000 00000021 00000000 00000042
0x43c001e0    00000000 00000021 00000000 00000042
0x43c001f0    00000000 00000021 00000000 00000042
0x43c00200    00000000 00000021 00000000 00000042
0x43c00210    00000000 00000021 00000000 00000042
0x43c00220    00000000 00000021 00000000 00000042
0x43c00230    00000000 00000021 00000000 00000042
0x43c00240    00000000 00000021 00000000 00000042
0x43c00250    00000000 00000021 00000000 00000042
0x43c00260    00000000 00000021 00000000 00000042
0x43c00270    00000000 00000021 00000000 00000042
0x43c00280    00000000 00000021 00000000 00000042
```

Down graded to Vivado 2022.1 and Vitis 2022.1
I followed youtube video
Hello world video using Xilinx Zynq, Vivado 2020, and Vitis:
https://www.youtube.com/watch?v=Mb-cStd4Tqs

In Vitis: need to take care of makefile issue. For the IP that user created. After importing Bitstream to Platform. Make sure all the user instances of makefile (2-4, better to search and fix all the files) This must be performed everytime you update the Hardware.

Then perform Clean and Build. If you still see issue



```
COMPILER=
ARCHIVER=
CP=cp
COMPILER_FLAGS=
EXTRA_COMPILER_FLAGS=
LIB=libxil.a

RELEASEDIR=../../../lib
INCLUDEDIR=../../../include
INCLUDES=-I./. -I${INCLUDEDIR}

INCLUDEFILES=$(wildcard *.h)
LIBSOURCES=$(wildcard *.c)
OUTS = $(wildcard *.o)

libs:
	echo "Compiling my_adder_v2..."
	$(COMPILER) $(COMPILER_FLAGS) $(EXTRA_COMPILER_FLAGS)
	$(ARCHIVER) -r ${RELEASEDIR}/${LIB} ${OUTS}
	make clean

include:
	${CP} $(INCLUDEFILES) $(INCLUDEDIR)

```

References:
https://www.fpgadeveloper.com/2014/08/creating-a-custom-ip-block-in-vivado.html/

hello_world.c

```
/*
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * ------------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ------------------------------------------------
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xil_io.h"
#include "myip_adder.h"
```

```
#define ADDER_BASEADDRESS 0x43C00000

int main()
{
   int a, b=1, answer=0,a_readback=0,b_readback=0,dummy=13,i,j;

   init_platform();

   printf("UART_Adder_test#5\n");
   printf("Add Two Number Program begins now. Enter b=0 to Exit\n");

   while(b!=0)
   {
      printf("Enter a= \n");
      scanf("%d",&a);
      MYIP_ADDER_mWriteReg(ADDER_BASEADDRESS,(4*0),a);
      printf("You entered a= %d\n",a);
      printf("Enter b= \n");
      scanf("%d",&b);
      MYIP_ADDER_mWriteReg(ADDER_BASEADDRESS,(4*1),b);
      printf("You entered b= %d\n",b);
      MYIP_ADDER_mWriteReg(ADDER_BASEADDRESS,(4*3),0x33);

      a_readback=MYIP_ADDER_mReadReg(ADDER_BASEADDRESS,(4*0));
      b_readback=MYIP_ADDER_mReadReg(ADDER_BASEADDRESS,(4*1));
      dummy=MYIP_ADDER_mReadReg(ADDER_BASEADDRESS,(4*3));
      answer=MYIP_ADDER_mReadReg(ADDER_BASEADDRESS,(4*2));


      printf("Sum = %d\n",answer);
      printf("a_readback = %d\n",a_readback);
      printf("b_readback = %d\n",b_readback);
      printf("dummy = %d\n",dummy);
      for(i=0;i<500;i++)
      {
         for(j=0;j<=5000;j++);
      }
      answer=MYIP_ADDER_mReadReg(ADDER_BASEADDRESS,(4*2));
      printf("Sum = %d\n",answer);
   }
   printf("Exiting\n");
   cleanup_platform();
   //return 0;
}
```