EGR680 High Level Implementation on FPGA

Laboratory 05

Custom IP Design using PYNQ

Professor: Dr. C. Parikh

Student: Dimitri Häring

October 03, 2018

Contents

1 Introduction

The goal of the lab 3 is to familiarize the student with a finite state machine implementation in verilog.

2 Design

In this section the design and decisions that where made to achieve the laboratory are discussed.

2.1 SDK Lab specification

In this part, you will create a simple hardcore ARM Cortex-A9 based embedded system on the PYNQ board. The embedded system design is broken up into three parts: Hardware design of the ARM Cortex-A9 hardcore processor, application software design using SDK, and finally hardware implementation of the software running on the hardcore processor. The application you will design in this part is a UART application that prints "HELLO WORLD" to a terminal emulator like Tera Term. Use the figure below as a flow guide for this lab. The diagram for the completed design is shown in Figure 1.

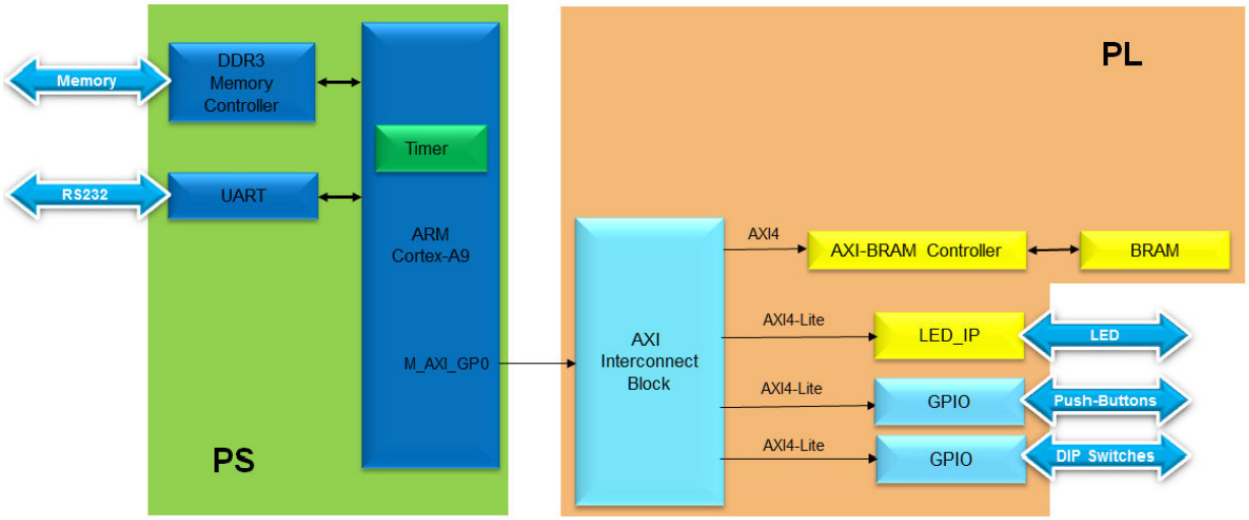

Figure 1: Completed Design.

2.2 HDL

Figure 2 shows the HDL top level based of the ZYNQ 7 Processing system which is connected with an AXI bus S00_AXI to a intellectual property (IP) block that manages peripherals. From there an AXI bus is used to connect two general purpose input output (GPIO) IP blocks, one for buttons and another one for switches. Furthermore, a Processor System Reset IP block is used that interconnects all resets.

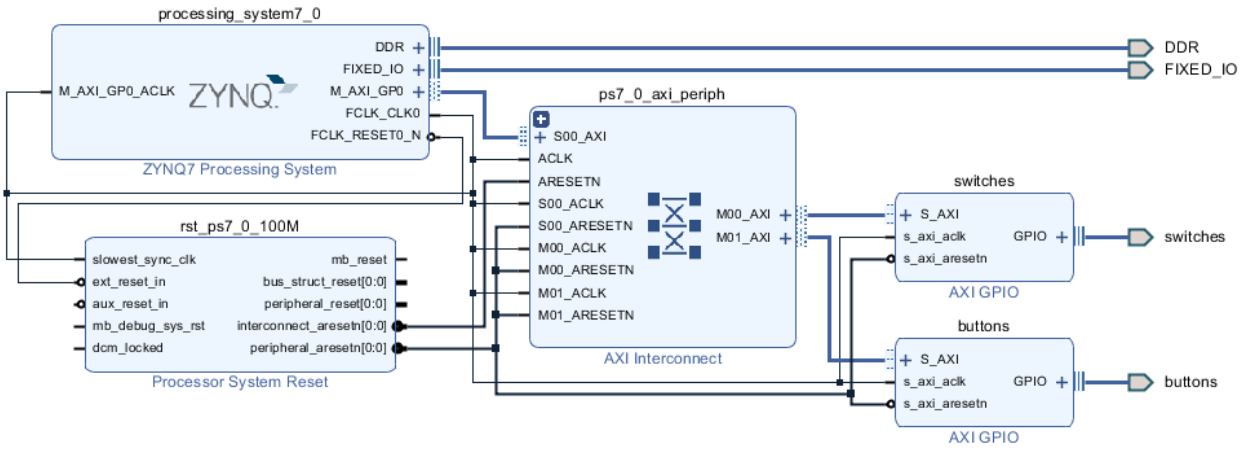

Figure 2: HDL Top Level Design.

2.3 SDK

After the HDL top level is defined the SDK can be launched with **File** → **Launch SDK**. The *.hdf file should be shown or can be opened that shows the base registers for the switches and buttons, as highlighted in Figure 3.

ps7_ocmc_0	0xf800c000	0xf800cfff		REGISTER
ps7_pl310_0	0xf8f02000	0xf8f02fff		REGISTER
ps7_coresight_comp_0	0xf8800000	0xf88fffff		REGISTER
ps7_uart_0	0xe0000000	0xe0000fff		REGISTER
ps7_scugic_0	0xf8f00100	0xf8f001ff		REGISTER
switches	0x41200000	0x4120ffff	S_AXI	REGISTER
ps7_dev_cfg_0	0xf8007000	0xf80070ff		REGISTER
ps7_dma_ns	0xf8004000	0xf8004fff		REGISTER
ps7_gpv_0	0xf8900000	0xf89fffff		REGISTER
ps7_ram_1	0xffff0000	0xfffffdff		MEMORY
ps7_ram_0	0x00000000	0x0002ffff		MEMORY
buttons	0x41210000	0x4121ffff	S_AXI	REGISTER

Figure 3: *.hdf file that shows the base register for switches and buttons.

After writing the C code given in part two and loading the the bit stream file and the build elf file onto the board the serial console Terra Term shows the status of the buttons and switches, as shown in Figure 4

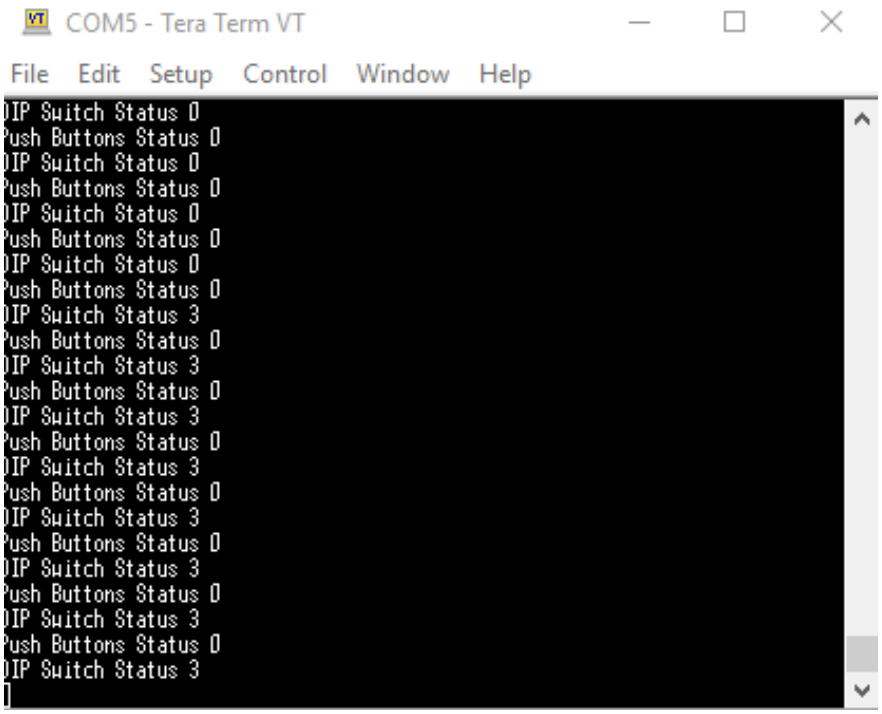

Figure 4: Terra Term print out of the switches and buttons status.

2.4 Let's Make a Deal

Part III of laboratory 4 is to program a game show named "Let's Make a Deal!". Therefor, the previous HDL code is reused because the push buttons (psb) and universal asynchronous receiver transmitter (UART) is used for the game show while the switches are just not initials in the program code and not used.

The game works as followed a computer, in this case ZYNQ embedded system, uses based on the users input time. The *startTime* stamp is made as the game starts while the *stopTime* stamp is made as the user presses the input button, the *deltaTime* is calculated according Equation 1. A random number is generated out of the *deltaTime* stamp that is determined according to Equation 2. The user input is to chose a door which hides a good deal but to get it the users answer has to match the computers generated door number.

The program consist of two switch case statements one for the program flow as shown in Figure 5. The second state machine contains the input logic to map the binary button input to a integer number. In addition, simple delays where used to de-bounce the buttons.

The four buttons are configured as the following btn0 is Door 1, btn1 is Door 2, btn2 is Door 3, btn3 is Door 4.

The complete code listening can be found under Section 5.1.

IMPORTANT: For the game show the programmable logic has to be flashed first with the generated bit stream file than the compiled C code can be flashed with the *.elf file.

$$deltaTime = startTime - stopTime \tag{1}$$

$$rnd = (deltaTime \ \% \ 4) + 1 \tag{2}$$

2.5 Game Show State Machine

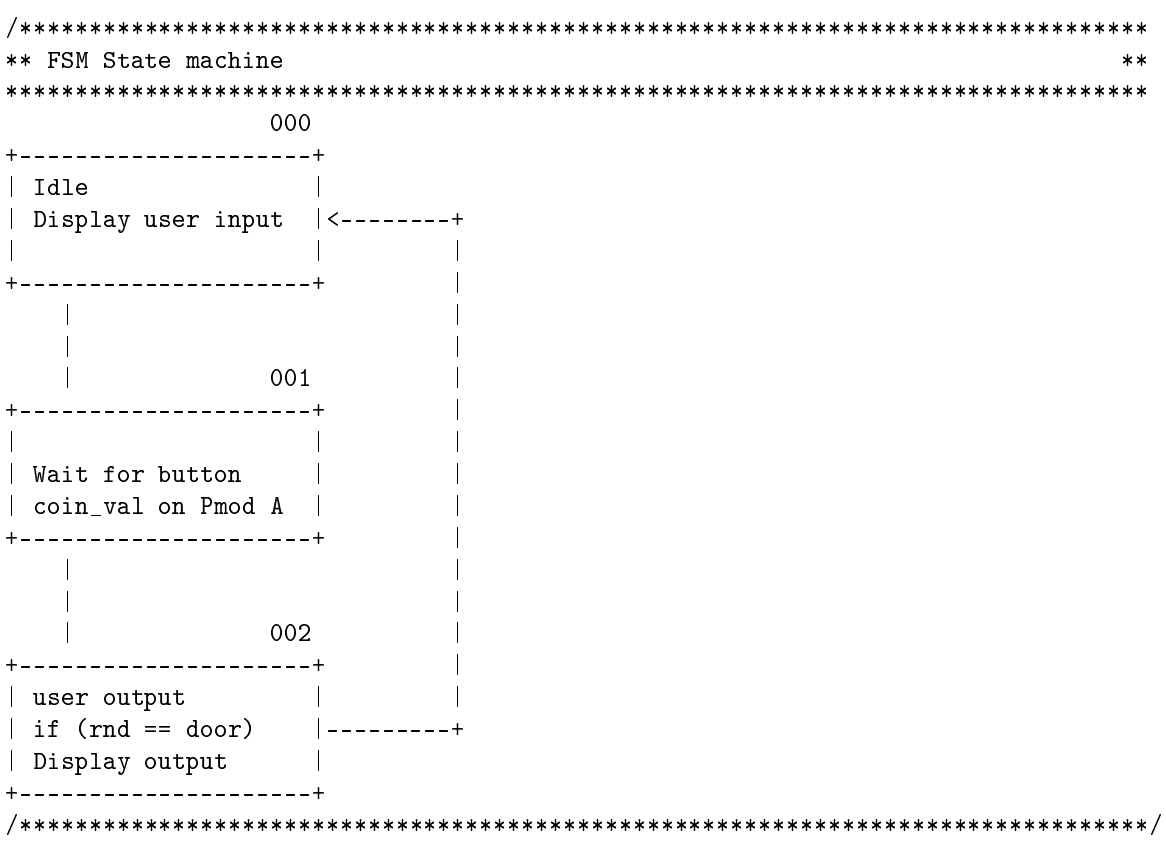

Figure 5: Game Show state machine.

3 Simulation

4 Conclusion

The lab demonstrates the use of the combined concepts of HDL logic implementation and sequential processor flow. The HDL defines the input peripherals and eventual logic, which is in this case none, around the embedded microprocessor. The state machine and program flow can then be easy initiated in C code and flashed on the microprocessor. This combination allows a broad spectrum of system design instead of predefined peripherals as used in a common micro-controller. A draw back is it adds complexity and a further step of development which might add complexity in maintenance and life cycle control.

5 Appendix

The appendix contains code listening and other large information parts that contain partial or complete relevance to the reports topic.

5.1 C code Part III

```
/*
 * GPIO_game.c
 * Implements a game show with four buttons and the uart console named
 * Let's Make a Deal!
 *
 *  Created on: 01.10.2018
 *      Author: D. Häring
 *
 *
 */

 /*    ************************************************************
 *    ** FSM State machine                                       **
 *    ************************************************************
 *                        000
 *    +-------------------------+
 *    | Idle                    |
 *    | Display user input      |<---------+
 *    |                         |          |
 *    +-------------------------+          |
 *         |                               |
 *         |                               |
 *         |                 001           |
 *    +-------------------------+          |
 *    |                         |          |
 *    | Wait for button         |          |
 *    | coin_val on Pmod A      |          |
 *    +-------------------------+          |
 *         |                               |
 *         |                               |
 *         |                 002           |
 *    +-------------------------+          |
 *    | user output             |          |
 *    | if (rnd == door)        |----------+
 *    | Display output          |
 *    +-------------------------+
 * ************************************************************/


#include "xparameters.h"
#include "xgpio.h"
#include "xtime_l.h"
#include "sleep.h"

int main (void)
{
  XGpio dip, push;
  int psb_check, state, door, rnd; //dip_check,
  XTime startTime, stopTime, deltaTime;

  xil_printf("\r\n\r\n -- Start of the Program Let's Make a Deal -- \r\n");
  XGpio_Initialize(&dip, XPAR_SWITCHES_DEVICE_ID);
  XGpio_SetDataDirection(&dip, 1, 0xffffffff);

  XGpio_Initialize(&push, XPAR_BUTTONS_DEVICE_ID);
  XGpio_SetDataDirection(&push, 1, 0xffffffff);

  state = 0;  // set default state
  door = 0; // set default door value
```

```
  rnd = 0;  // set default rnd value
  startTime = 0;    // set default startTime value
  stopTime = 0;   // set default stopTime value
  deltaTime = 0;    // set default deltaTime value

  while(1)
  {

    psb_check = XGpio_DiscreteRead(&push, 1);

    switch (state) {
    case 0:// Idle state
      xil_printf("\r\n\r\n**********************************\r\n");
      xil_printf("---Welcome to Let's Make a Deal!---\r\n");
      xil_printf("**********************************\r\n");
      xil_printf("Select between 1 and 4 to seed the Random Number Generator: ");
      door = 0; // set default door value
      rnd = 0;  // set default rnd value
      startTime = 0;    // set default startTime value
      stopTime = 0;   // set default stopTime value
      deltaTime = 0;    // set default deltaTime value
      XTime_GetTime(&startTime); // get start time
      state = 1;
      break;
    case 1: // wait for door to be chosen
      if (door){
        XTime_GetTime(&stopTime);
        if (startTime < stopTime) // handle eventual overflow
          deltaTime = stopTime - startTime;
        else
          deltaTime = startTime - stopTime;
        rnd = deltaTime % 4;
        rnd = rnd + 1;
        xil_printf("  %d\r\n", door);
//        xil_printf("  --> Debug: deltaTime %08x \r\n", deltaTime); // debug only
        usleep(400000); // from GVSU EE EGR326 2015 Fall lab 02
        state=2;
      }
      else
        state=1;

      break;
    case 2: // print result after player chose door
      xil_printf("\r\nComputer chose %d\r\n", rnd);
      xil_printf("----------------------\r\n");
      if (rnd == door)
        xil_printf("You Win!\r\n");
      else
        xil_printf("You suck !!!\r\n");
      xil_printf("----------------------\r\n");


      door = 0; // set default door value
      rnd = 0;  // set default rnd value
      startTime = 0;    // set default startTime value
      stopTime = 0;   // set default stopTime value
      deltaTime = 0;    // set default deltaTime value
      state = 0;      // set default state
      break;
    default:
      xil_printf("  --> Debug: %d %s\r\n", __FILE__, __LINE__);
      break;
    }

    switch (psb_check) {
    usleep(800000); // from GVSU EE EGR326 2015 Fall lab 02
    case 1:
      door = 1;
```

```
128         break;
129       case 2:
130         door = 2;
131         break;
132       case 4:
133         door = 3;
134         break;
135       case 8:
136         door = 4;
137         break;
138       default:
139         break;
140       }
141
142   } // end while(1)
143 } // end main
```

Listing 1: GPIO_game.c fill contained C code.

5.2 Errors

5.2.1 Implementation Error [Place 30-574] Poor placement for routing between an I/O pin and BUFG

Poor placement for routing between an I/O pin and BUFG is a state where a clk signal is routed to anon clock net or a non clock signal is routed to clock net like a posedge or negedge clk signal. Now it seems this clock placement error that occurs by implementing the project is somewhat related to the decoder. As the decoder module is commented out of the top level block the error is not there any more. Still the source or what pin shall causing the issue could not be located at first. The issue was that the in the decoder initialization the clk and rst pin where switched.

5.2.2 Board file

The following error was generated due to the use of the chip set while generating the Vivado project instead of configure the project with the board file.

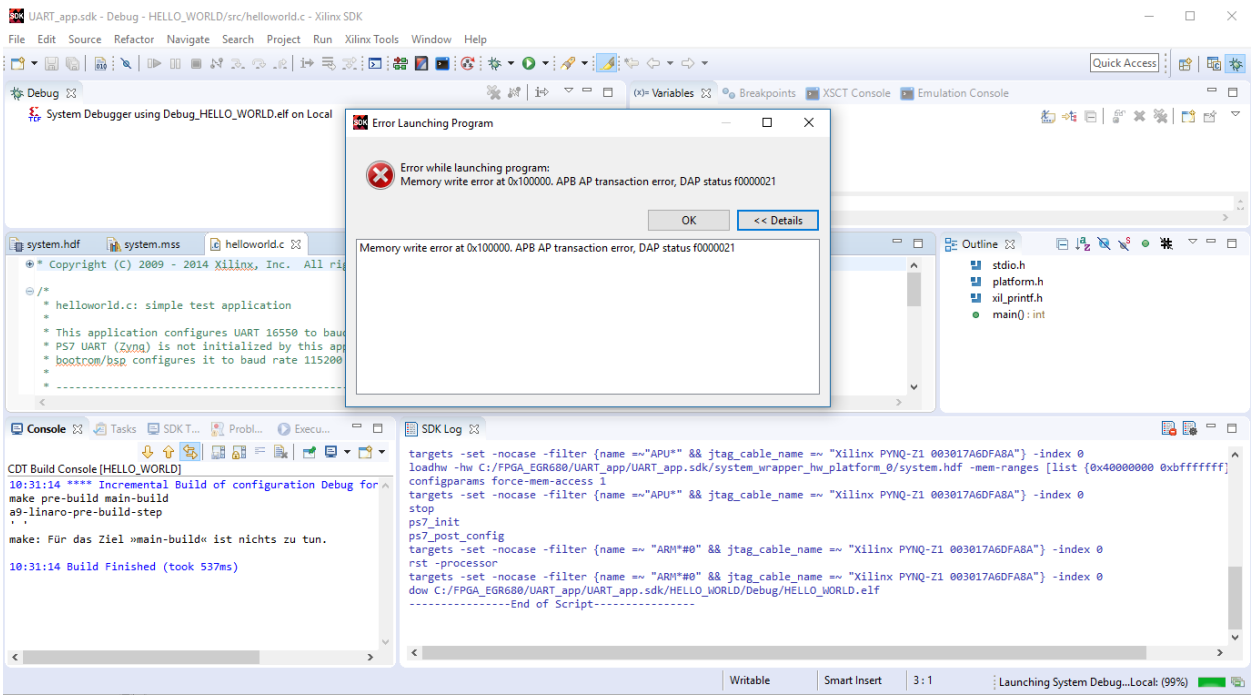

Figure 6: Error SDK Part I.