# ECE 5722 Homework 5

**By Bruce Huynh**

**11/13/2021**

**Introduction:**

In this homework assignment, we learn to actually program the hardware and software of the DE0-CV board using Intel Quartus and its dependent applications. We use everything to be able to reverse and display an array stored in the memory.

**A. C/C++ code**

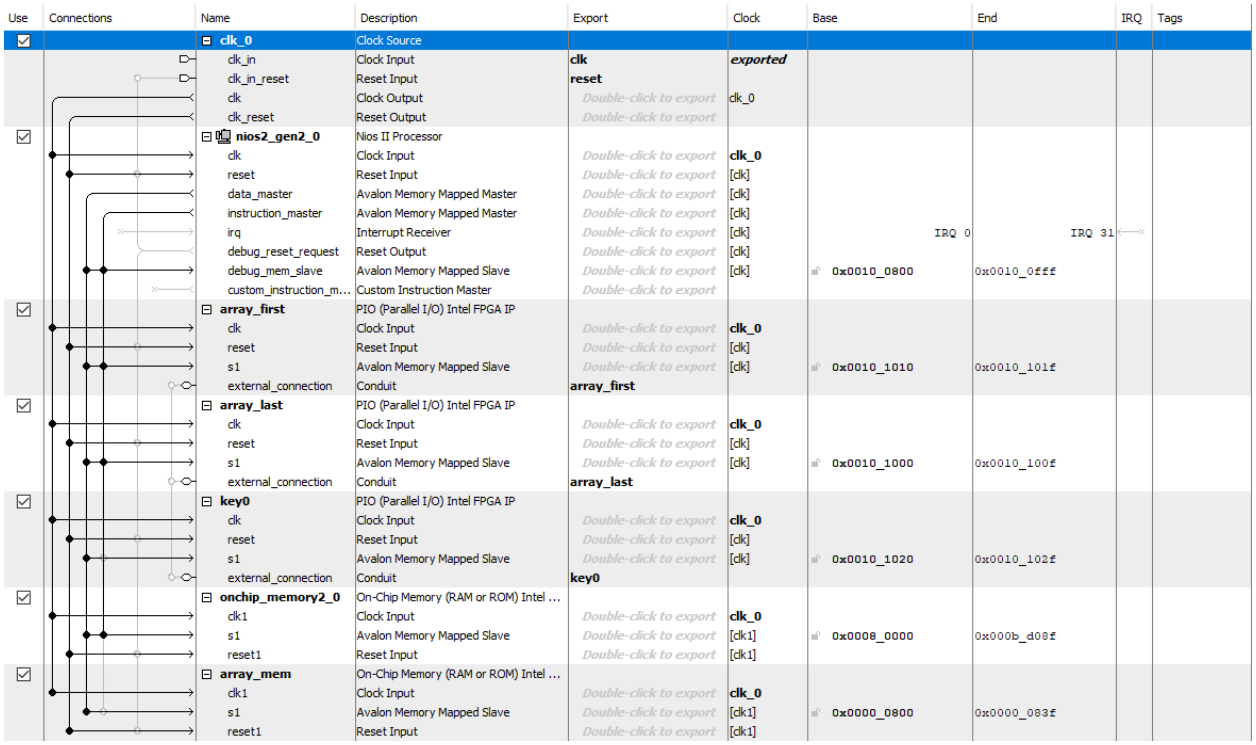
```
#include <stdio.h>
#include <string.h>
#include "altera_avalon_pio_regs.h"
#include "system.h"
#include "unistd.h"
#include "io.h"



int main(void)
{     int keyStatus;
    int myArrayOG[] = { 5, 2, 3, 4, 5, 6, 7, 8, 6 ,6 };
    int n = sizeof(myArrayOG)/sizeof(myArrayOG[0]);
    int myArrayR[n];
    int myArray1stOG = myArrayOG[0];  // first of array
     int myArrayLastOG = myArrayOG[n-1]; // last of array


    for (int x = 0; x < n; x++){
      myArrayR[x] = myArrayOG[x]; // making the copy equal to
myArrayOG
    }

    for (int y = 0; y < n; y++){ // putting the values of the array
in the memory
     IOWR_8DIRECT(0x800, 4, myArrayOG[y]); // using 0x800 since
ARRAY_MEM_BASE doesn't work

     if (y == 63) { // breaks if array is larger than 0x800 - 0x840
```

```
            break;
      }

    }

    while(1){
    keyStatus = IORD_ALTERA_AVALON_PIO_DATA(KEY0_BASE);



    if (keyStatus == 0){
        for (int low = 0, high = n - 1; low < high; low++, high--)
// reverse array
        {
            int temp = myArrayR[low];
            myArrayR[low] = myArrayR[high];
            myArrayR[high] = temp;
        }
        int myArray1st = myArrayR[0];
        int myArrayLast = myArrayR[n-1];



        IOWR_ALTERA_AVALON_PIO_DATA(ARRAY_FIRST_BASE, myArray1st);
        IOWR_ALTERA_AVALON_PIO_DATA(ARRAY_LAST_BASE, myArrayLast);
        usleep(500000);
    }
    else if(keyStatus == 1){
        IOWR_ALTERA_AVALON_PIO_DATA(ARRAY_FIRST_BASE, myArray1stOG);
        IOWR_ALTERA_AVALON_PIO_DATA(ARRAY_LAST_BASE, myArrayLastOG);
    }


    printf(" |Current size of array: %i| ",n);

    }
    return 0;
}
```

In general the code was very straightforward to create. I just had to do a couple of for loops to transfer array contents and to reverse the array contents. There were issues with accessing the hex file from Eclipse so I had to directly call the memory location in order to write the memory.

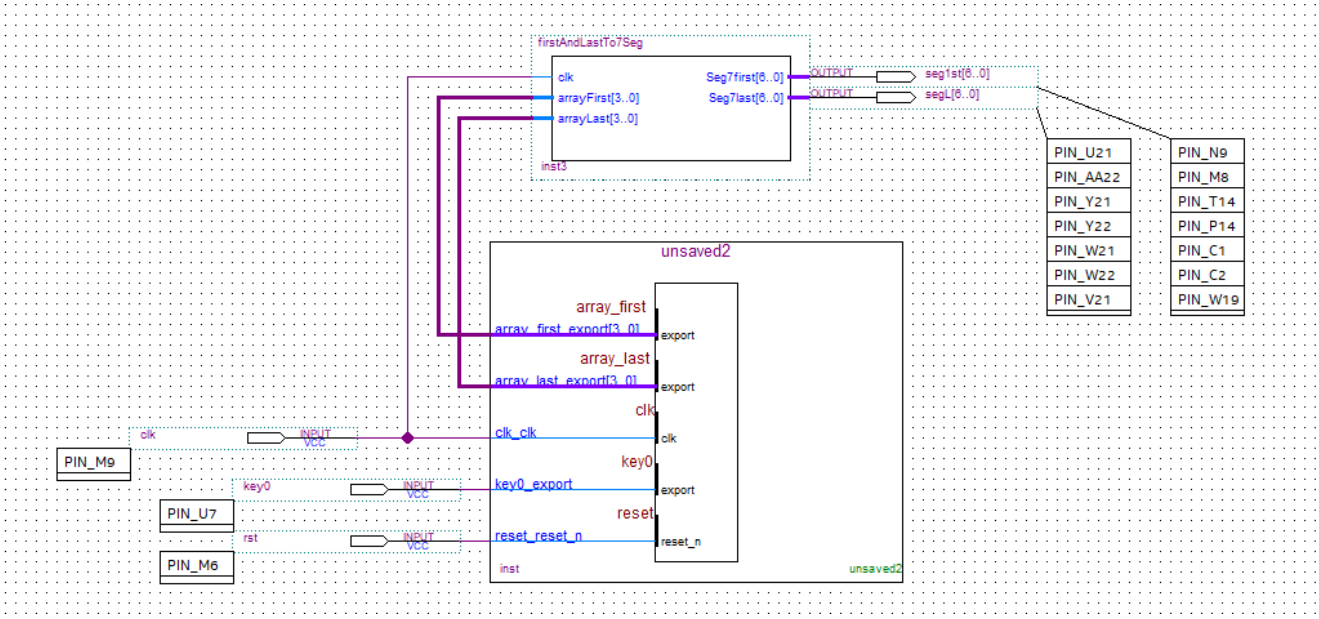
## B. Qsys design

### Qsys component:



| Use | Connections | Name | Description | Export | Clock | Base | End | IRQ | Tags |
|---|---|---|---|---|---|---|---|---|---|
| ☑ | | **clk_0** | Clock Source | | | | | | |
| | | clk_in | Clock Input | **clk** | ***exported*** | | | | |
| | | clk_in_reset | Reset Input | **reset** | | | | | |
| | | clk | Clock Output | *Double-click to export* | clk_0 | | | | |
| | | clk_reset | Reset Output | *Double-click to export* | | | | | |
| ☑ | | **nios2_gen2_0** | Nios II Processor | | | | | | |
| | | clk | Clock Input | *Double-click to export* | **clk_0** | | | | |
| | | reset | Reset Input | *Double-click to export* | [clk] | | | | |
| | | data_master | Avalon Memory Mapped Master | *Double-click to export* | [clk] | | | | |
| | | instruction_master | Avalon Memory Mapped Master | *Double-click to export* | [clk] | | | | |
| | | irq | Interrupt Receiver | *Double-click to export* | [clk] | IRQ 0 | IRQ 31 | | |
| | | debug_reset_request | Reset Output | *Double-click to export* | [clk] | | | | |
| | | debug_mem_slave | Avalon Memory Mapped Slave | *Double-click to export* | [clk] | 0x0010_0800 | 0x0010_0fff | | |
| | | custom_instruction_m... | Custom Instruction Master | *Double-click to export* | | | | | |
| ☑ | | **array_first** | PIO (Parallel I/O) Intel FPGA IP | | | | | | |
| | | clk | Clock Input | *Double-click to export* | **clk_0** | | | | |
| | | reset | Reset Input | *Double-click to export* | [clk] | | | | |
| | | s1 | Avalon Memory Mapped Slave | *Double-click to export* | [clk] | 0x0010_1010 | 0x0010_101f | | |
| | | external_connection | Conduit | **array_first** | | | | | |
| ☑ | | **array_last** | PIO (Parallel I/O) Intel FPGA IP | | | | | | |
| | | clk | Clock Input | *Double-click to export* | **clk_0** | | | | |
| | | reset | Reset Input | *Double-click to export* | [clk] | | | | |
| | | s1 | Avalon Memory Mapped Slave | *Double-click to export* | [clk] | 0x0010_1000 | 0x0010_100f | | |
| | | external_connection | Conduit | **array_last** | | | | | |
| ☑ | | **key0** | PIO (Parallel I/O) Intel FPGA IP | | | | | | |
| | | clk | Clock Input | *Double-click to export* | **clk_0** | | | | |
| | | reset | Reset Input | *Double-click to export* | [clk] | | | | |
| | | s1 | Avalon Memory Mapped Slave | *Double-click to export* | [clk] | 0x0010_1020 | 0x0010_102f | | |
| | | external_connection | Conduit | **key0** | | | | | |
| ☑ | | **onchip_memory2_0** | On-Chip Memory (RAM or ROM) Intel ... | | | | | | |
| | | clk1 | Clock Input | *Double-click to export* | **clk_0** | | | | |
| | | s1 | Avalon Memory Mapped Slave | *Double-click to export* | [clk1] | 0x0008_0000 | 0x000b_d08f | | |
| | | reset1 | Reset Input | *Double-click to export* | [clk1] | | | | |
| ☑ | | **array_mem** | On-Chip Memory (RAM or ROM) Intel ... | | | | | | |
| | | clk1 | Clock Input | *Double-click to export* | **clk_0** | | | | |
| | | s1 | Avalon Memory Mapped Slave | *Double-click to export* | [clk1] | 0x0000_0800 | 0x0000_083f | | |
| | | reset1 | Reset Input | *Double-click to export* | [clk1] | | | | |

### BDF:




### C. Qsys design explanation

For our Qsys model, since the array will already be on the memory, we do not have to place any inputs for it. I had three inputs for the Qsys design which would be the key to flip the array, the reset key, and the clock which was internally connected. Since the Qsys design does not know how to work with the 7-segment display, I created a verilog file that turned the binary into 7-segment display outputs.

```
module firstAndLastTo7Seg(input clk, input [3:0] arrayFirst,
arrayLast, output [6:0] Seg7first, Seg7last);
	reg [6:0] seg1, segL;
	always @(posedge clk)
		begin
			case (arrayFirst) //case statement
				4'b0000 : seg1 = 7'b1000000;
				4'b0001 : seg1 = 7'b1111001;
				4'b0010 : seg1 = 7'b0100100;
				4'b0011 : seg1 = 7'b0110000;
				4'b0100 : seg1 = 7'b0011001;
				4'b0101 : seg1 = 7'b0010010;
				4'b0110 : seg1 = 7'b0000010;
```

```
                        4'b0111 : seg1 = 7'b1111000;
                        4'b1000 : seg1 = 7'b0000000;
                        4'b1001 : seg1 = 7'b0010000;
                        default : seg1 = 7'b1111111;
                endcase

                    case (arrayLast) //case statement
                        4'b0000 : segL = 7'b1000000;
                        4'b0001 : segL = 7'b1111001;
                        4'b0010 : segL = 7'b0100100;
                        4'b0011 : segL = 7'b0110000;
                        4'b0100 : segL = 7'b0011001;
                        4'b0101 : segL = 7'b0010010;
                        4'b0110 : segL = 7'b0000010;
                        4'b0111 : segL = 7'b1111000;
                        4'b1000 : segL = 7'b0000000;
                        4'b1001 : segL = 7'b0010000;
                        default : segL = 7'b1111111;
                endcase
        end

        assign Seg7first = seg1;
        assign Seg7last = segL;



endmodule
```

After creating the Qsys design and the verilog module, we generated symbols for them and connected them with the DE0-CV pins. For the clock, we used a 50 MHz clock signal and pushbuttons are used for the swapping and reset.

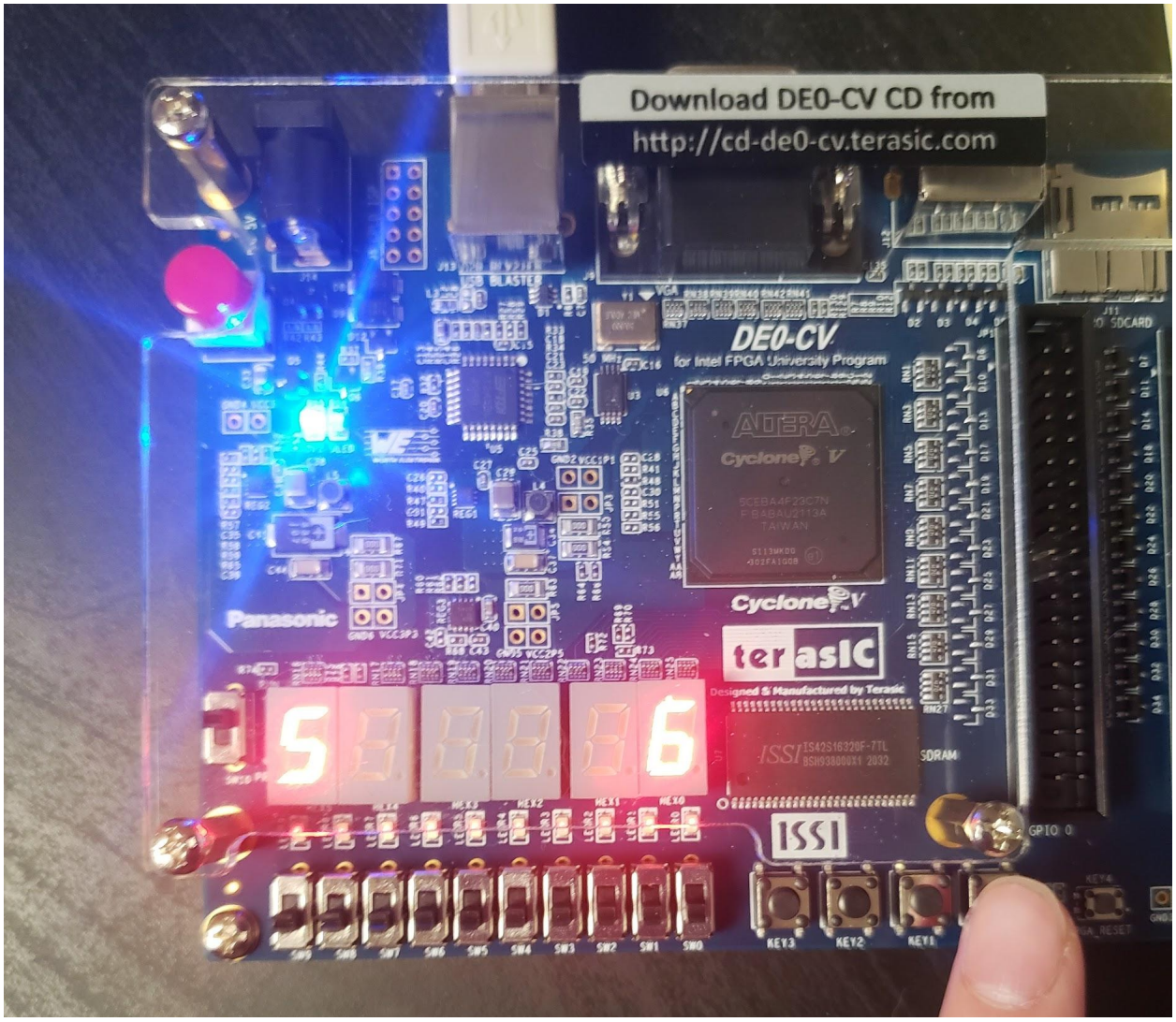
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| clk | Input | PIN_M9 | 3B | B3B_N0 | PIN_M9 | 2.5 V | | 12mA (default) | |
| key0 | Input | PIN_U7 | 3A | B3A_N0 | PIN_U7 | 2.5 V | | 12mA (default) | |
| rst | Input | PIN_M6 | 3A | B3A_N0 | PIN_M6 | 2.5 V | | 12mA (default) | |
| seg1st[6] | Output | PIN_W19 | 4A | B4A_N0 | PIN_W19 | 2.5 V | | 12mA (default) | 1 (default) |
| seg1st[5] | Output | PIN_C2 | 2A | B2A_N0 | PIN_C2 | 2.5 V | | 12mA (default) | 1 (default) |
| seg1st[4] | Output | PIN_C1 | 2A | B2A_N0 | PIN_C1 | 2.5 V | | 12mA (default) | 1 (default) |
| seg1st[3] | Output | PIN_P14 | 4A | B4A_N0 | PIN_P14 | 2.5 V | | 12mA (default) | 1 (default) |
| seg1st[2] | Output | PIN_T14 | 4A | B4A_N0 | PIN_T14 | 2.5 V | | 12mA (default) | 1 (default) |
| seg1st[1] | Output | PIN_M8 | 3B | B3B_N0 | PIN_M8 | 2.5 V | | 12mA (default) | 1 (default) |
| seg1st[0] | Output | PIN_N9 | 3B | B3B_N0 | PIN_N9 | 2.5 V | | 12mA (default) | 1 (default) |
| segL[6] | Output | PIN_AA22 | 4A | B4A_N0 | PIN_AA22 | 2.5 V | | 12mA (default) | 1 (default) |
| segL[5] | Output | PIN_Y21 | 4A | B4A_N0 | PIN_Y21 | 2.5 V | | 12mA (default) | 1 (default) |
| segL[4] | Output | PIN_Y22 | 4A | B4A_N0 | PIN_Y22 | 2.5 V | | 12mA (default) | 1 (default) |
| segL[3] | Output | PIN_W21 | 4A | B4A_N0 | PIN_W21 | 2.5 V | | 12mA (default) | 1 (default) |
| segL[2] | Output | PIN_W22 | 4A | B4A_N0 | PIN_W22 | 2.5 V | | 12mA (default) | 1 (default) |
| segL[1] | Output | PIN_V21 | 4A | B4A_N0 | PIN_V21 | 2.5 V | | 12mA (default) | 1 (default) |
| segL[0] | Output | PIN_U21 | 4A | B4A_N0 | PIN_U21 | 2.5 V | | 12mA (default) | 1 (default) |

## D, E, F. Synthesis and Board Programming

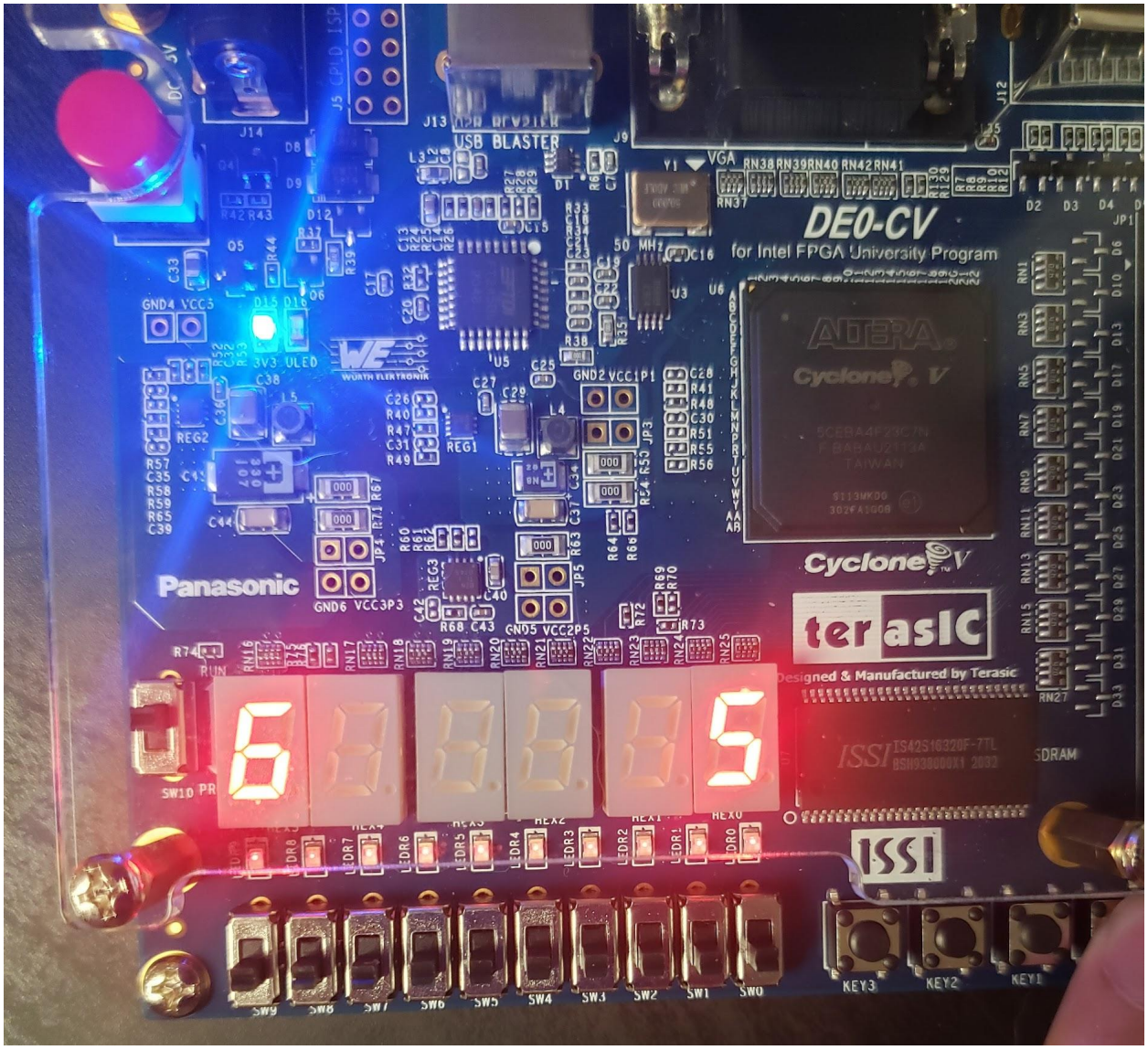
The following is the synthesis report of our full design:

| | |
|---|---|
| Flow Status | Successful - Mon Nov 15 16:34:38 2021 |
| Quartus Prime Version | 20.1.1 Build 720 11/11/2020 SJ Lite Edition |
| Revision Name | ece5722hw5 |
| Top-level Entity Name | Block2 |
| Family | Cyclone V |
| Device | 5CEBA4F23C7 |
| Timing Models | Final |
| Logic utilization (in ALMs) | 762 / 18,480 ( 4 % ) |
| Total registers | 861 |
| Total pins | 17 / 224 ( 8 % ) |
| Total virtual pins | 0 |
| Total block memory bits | 2,010,752 / 3,153,920 ( 64 % ) |
| Total DSP Blocks | 0 / 66 ( 0 % ) |
| Total HSSI RX PCSs | 0 |
| Total HSSI PMA RX Deserializers | 0 |
| Total HSSI TX PCSs | 0 |
| Total HSSI PMA TX Serializers | 0 |
| Total PLLs | 0 / 4 ( 0 % ) |
| Total DLLs | 0 / 4 ( 0 % ) |

Here are the results of programming the board with the array (5, 2, 3, 4, 5, 6, 7, 8, 6 ,6):




These are the first and last values of the array when the button is not pressed.



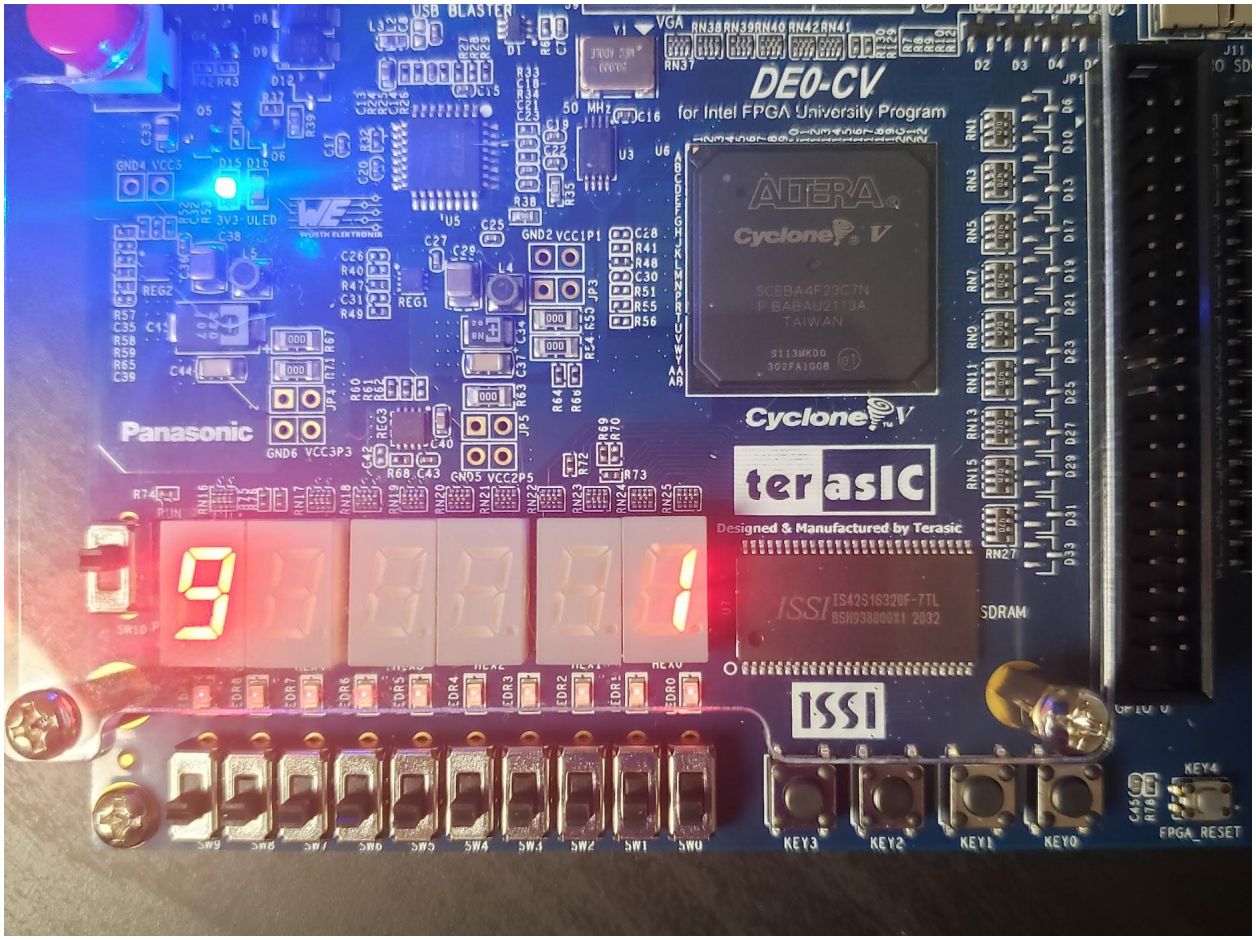

When the buttons are pressed, 6 becomes the first value of the array and 5 becomes the last.

To show that the whole array is getting reversed instead of the last two values:

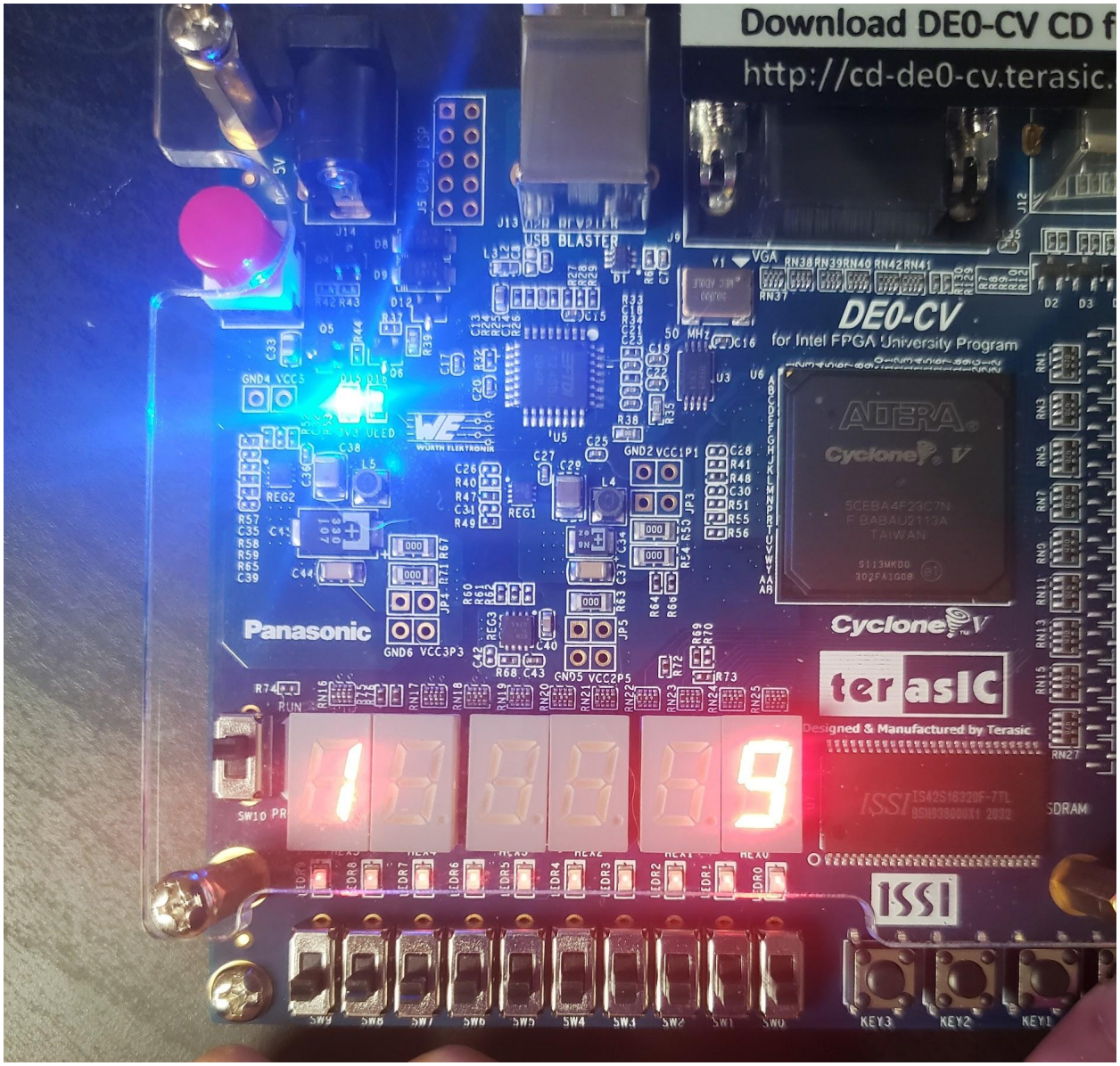
Original array: 6 6 8 7 6 5 4 3 2 5

New array: 5 2 3 4 5 6 7 8 6 6

Here are the results of programming the board with the array (9, 2, 1, 0, 5, 4, 1, 9, 9, 3, 3, 2, 1, 5, 6, 6, 3, 7, 7, 8, 1):




This is an image of the first and last value of the array when key0 is not pressed.




This image shows the array being reversed when the button is being held down.

Again, to show that the whole array is being reversed and not just the two values:

Original array: 9 2 1 0 5 4 1 9 9 3 3 2 1 5 6 6 3 7 7 8 1

New array: 1 8 7 7 3 6 6 5 1 2 3 3 9 9 1 4 5 0 1 2 9

This code can possibly work for numbers larger than 9 as well but that would require more 7-segment displays to be programmed.

**Conclusion:**

In this assignment, we learned how to use most of the Quartus tools to create a synthesizable FPGA design. Although this first assignment is easy, we can expect the next one to be much more challenging and will test our knowledge of hardware design even further.