ECEN 5863: Programmable Logic Embedded System Design

FA23 Project 3 Technical Report

Matt Harnett, Eric Percin & Isha Sharma

Table of Contents

Executive Summary

Project 3 provides an introduction to various essential functions of the DE1-SoC. It explores hardware and software features throughout three dedicated submodules. Our team completed each module. In the first, four separate designs were successfully created to understand the interfacing of inputs and outputs with the FPGA. In the second, designs were developed using CODEC and Monitor Program testing the audio filtering and C coding functionalities. In the third, HPS was interfaced with the FPGA using Qsys and EDS. Lastly, a snake game prototype was developed for the FPGA using a VGA Monitor and on-board keys as the hardware interface and Nios-II console as the software interface. In conclusion, Altera's Cyclone V SoCs are very versatile providing numerous functionalities. They combine the flexibility of FPGA fabric with the processing power of ARM Cortex-A9 cores, providing a versatile platform for a wide range of applications. We had a lot of fun playing this board.

Objectives

Our team successfully accomplished the following objectives:

Module I

Developed four designs integrating Switches, LEDs, and 7-Segment displays with the DE1-SoC board. These designs consisted of becoming familiar with interfacing various SoC inputs and outputs using Quartus and Verilog.
Also developed Verilog designs for clocks and timed circuits. The designs fundamentally included developing a modulo-k counter with rollover. This was used to build a BCD counter and a real-time clock.

Module II

Became familiar with audio sampling, processing, and filtering. Understood the CODEC on DE1-SoC by developing an FIR filter for noise reduction.
Familiarised ourselves with the Monitor Program and ARM Cortex-A9 processor. Developed C designs including finding the largest number, real-time clock, and scrolling display.

Module III

Learned about using HPS/ARM to communicate with an FPGA. Developed a design to control LEDs using the HPS program via Qsys and Altera SoC EDS using Linux.

Final Module: FPGA-Based Snake Game Prototype

Developed an FPGA system for the arcade game *Snake* with the DE1-SoC and a VGA monitor. Used Qsys and Altera University Program IP to implement a VGA controller that interfaces with a Nios II soft processor. Used Nios II Software Build tools to program the game logic in C.

Procedure

Module I

Follow the ‘lab1_verilog’ guide, building four simple projects for the DE1-SoC that use implement simple input/output devices on the SoC. Prepare the relevant pin assignments, compile and implement the design, and then finally program the chip and test the hardware.
Follow the ‘lab5_verilog’ guide, building three different timing circuit projects. Simulate the design using the necessary pin assignments and test the circuit after programming the board.

Module II

Follow the ‘lab12_verilog’ guide creating a design using the start kit circuit and programming it. Complete the provided audio input to output schematic by assigning internal wires to the correct outputs. Create an FIR filter to lower noise levels in the output signal.
Follow the provided steps, and using the Monitor program to develop C codes, identify the largest number in a list of 7 32-bit integers stored in memory, create a real-time clock, and make text scroll across the 7-segment displays. Program the board and test functionality. This helps in understanding the working of timed circuits using counters.=

Module III

Follow the ‘My First HPS-Fpga’ guide to develop an HPS-enabled Quartus project. Download the source code with the included pin declarations. Following the guide and adding the PIO component using Qsys, generate the source code. Compile and program the DE1-SoC. Follow the guide to design the Arm HPS C program using the Arm EDS. Finally, bootload the board using an SD card with a Linux image, launch the program, and observe the LED output.

Final Module: FPGA-Based Snake Game Prototype

Create a playable prototype of the classic arcade game *Snake* on the DE1-SoC using pushbuttons to accept player input and VGA output to display the state of the game on a monitor. Design a Qsys system that allows a Nios II soft processor to interface with a VGA controller by reading and writing in SRAM. Write a C program that implements game logic, VGA control, and pushbutton input to create a cohesive game.

Module Test Results

Module I

“Follow the instructions in lab1 Parts I – IV (skip V and VI) and record your observations.”
The DE1-SoC FPGA was successfully programmed to implement the on-board LEDs and Switches in Lab1 parts 1,2 and 3 as described; a direct switch to LED circuit, a 4-bit wide 2-to-1 multiplexer, a

two-bit wide 3-to-1 multiplexer, and a 7-segment decoder were designed, programmed and tested to work exactly as expected.

"Follow the instructions in lab5 Parts I – III (skip V and VI) and record your observations."
Verilog timed circuits were successfully designed. A modulo-k counter with rollover count for k-1 was developed in part 1. It was instantiated for n=8 and k=20. This was then used in part 2 to develop a BCD counter that counts from 0-999. Further, this was used to develop a real-time clock of 50MHz. The board behaved as expected. The counter in part was reset using KEY1 and the LEDs depicted the increment in count. In part 2, the HEX2-0 displayed the BCD counter values and KEY0 was used to reset it. The real-time clock in part 3 displayed on the HEX5-0 counted 0-99 hundredth seconds and 0-59 minutes and seconds. The states of SW7-0 depicted values for resetting the minutes when KEY1 was pressed. KEY0 was used to pause the clock. The timers were roughly accurate as the counter was used to develop a clock divider for the required clock frequency and then the following instances were made. This clock divider ensured the correct working of the timers using the n and k values.

The labeled screenshots for each part are present in the appendix.

"Record the fMAX and estimate the % utilization of the FPGA logic."

Circuit	Maximum Frequency	Percent Utilization
Lab1 Part1	No clock	<1%
Lab1 Part2	No clock	<1%
Lab1 Part3	No clock	<1%
Lab1 Part4	No clock	<1%
Lab5 Part1	435.92 MHz	<1%
Lab5 Part2	265.04 MHz(Clock_50), 295.86 MHz (rollover clock enable)	<1%
Lab5 Part3	300.3 MHz (Clock_50), 367.92 MHz (rollover clock enable)	<1%

Module II

"Follow the instructions in lab12 Parts I and II (skip III) and record your observations"

Circuit	Maximum Frequency	Percent Utilization
Lab12 Part1	226.6 MHz	1%
Lab12 Part2	164.04 MHz	<1%

In part 3, the largest number in an array of 32-digit integers was displayed on the semihosting terminal of the Monitor Program. Using the tutorial in the Monitor Program and updating the values to match the 10msec, hex table to display on the 7-segment display and rolling count, a real-time clock was implemented in part 4. A pushbutton key was used to reset the clock. The board behaved as expected. Similarly, using the example program a scrolling message display for part 5 was successfully implemented. The board scrolled the ‘de1-SoC from right to left at 0.2 seconds per character stopping/running by the press of any KEY.

Module III

The HPS program was prepared and run as defined by the guide. The board behaved as expected; one of the LEDs was off and constantly bounced across the LED array from side to side. The Quartus summary reports an fMAX of 63.42 MHz for altera_reserved_tck, 89.16 MHz for clock_50, and 1184.83 MHz for soc_system:uo. The highest speed clock used in this design is 50 MHz, as shown in the sdc file. The % utilization was estimated to be approximately 7%.

Final Module: FPGA-Based Snake Game Prototype

The final *Snake* prototype worked as specified in the project proposal. The player character can be controlled using the four pushbuttons on the DE1-SoC. Collecting food will cause the player character to grow in length. Colliding with the game zone boundaries or with the snake body will cause the player to lose and the game to be reset. The Nios II and VGA controller system was successfully implemented, although largely with Altera University Program IP due to difficulties the team could not overcome in time when attempting to design their own Qsys system. The game logic was successfully programmed in C on the Nios II soft processor via the Nios II Software Build Tools for Eclipse.

Lessons Learned

- In the DE1-SoC, the ARM-9 processor is integrated into the system-on-chip (SoC) architecture, providing a powerful processing unit alongside programmable logic resources.
- The Intel FPGA Monitor Program provides the ease of coding in C and loading the FPGA very efficiently.
- Leveraging SoC EDS for a comprehensive development environment helps integrate both hardware and software components seamlessly.
- A successful FPGA project requires a seamless integration of both hardware and software components
- Proper understanding of interfacing protocols and thorough testing is essential to ensure reliable communication between peripherals and the DE1-SoC board.
- Attention to detail is crucial in the coding and mapping of hexadecimal values to 7-segment displays.
- Careful consideration of clock domains and proper use of counters are vital to achieving accurate and reliable timed functionalities within the FPGA project.

- Rigorous testing and verification are essential steps in FPGA design. Identifying and resolving issues related to hardware interfacing, software development, and communication between different components is key.
- Time management is very important to have a better understanding of the project.
- Dealing with the complexity of FPGA systems demands careful planning, resource optimization, and a deep understanding of the application's needs. Successfully managing complexity is essential to deliver effective FPGA designs.
- Some common subsystems–such as VGA output–are more complex in their inner workings than they might seem from the surface.
- Existing IP blocks can be an incredibly valuable resource; while we initially planned to complete our project independently, we ultimately would not have been able to finish in time without leveraging IP cores

Conclusions

We were able to successfully navigate the complexities of the DE1-SoC FPGA, in both hardware and software. This project helped us understand the FPGA systems and also improved our skills in overcoming challenges associated with diverse development tools. We achieved significant milestones in understanding and working with the DE1-SoC FPGA platform. Interfacing switches and LEDs, designing timed circuits with counters, utilizing the 7-segment hex display, incorporating audio filtering using the codec on the DE1-SoC, interfacing HPS with the FPGA, and developing software in C using the Monitor Program on the DE1-SoC. Module 1 and 2 helped us understand the value of and need for seamless integration between hardware and software. Module 3 helped us grasp the basics of connecting HPS with an FPGA. Module 4 tested the FPGA design and programming skills that we have been developing throughout this semester. Though we encountered many challenges and relied upon existing IP more than we wanted to, we were able to successfully deliver upon the final product described in our original proposal.

Appendix

Figures associated with each module are presented below.

Module I

Lab I Part I

```
// Simple module for DE1-SoC and DE0-CV that connects the SW switches to the LEDR lights

module part1 (SW, LEDR);
   input [9:0] SW; // slide switches
   output [9:0] LEDR; // red LEDs

   assign LEDR = SW;
endmodule
```

Figure 1: Verilog code

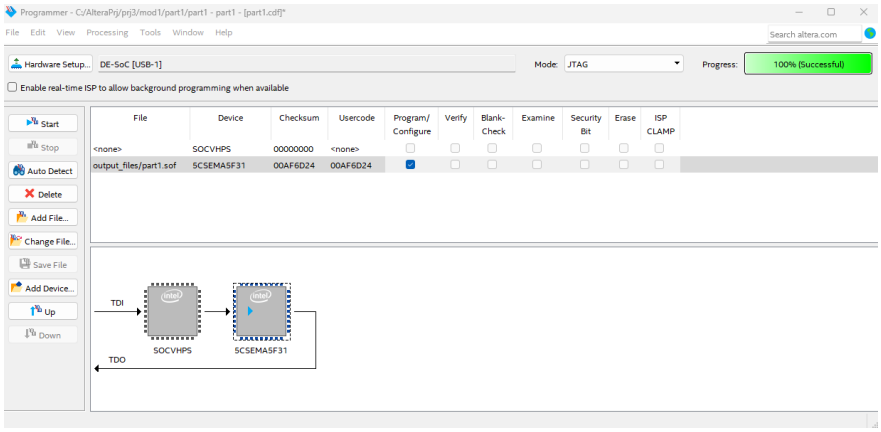


Figure 2: Sucessful programming

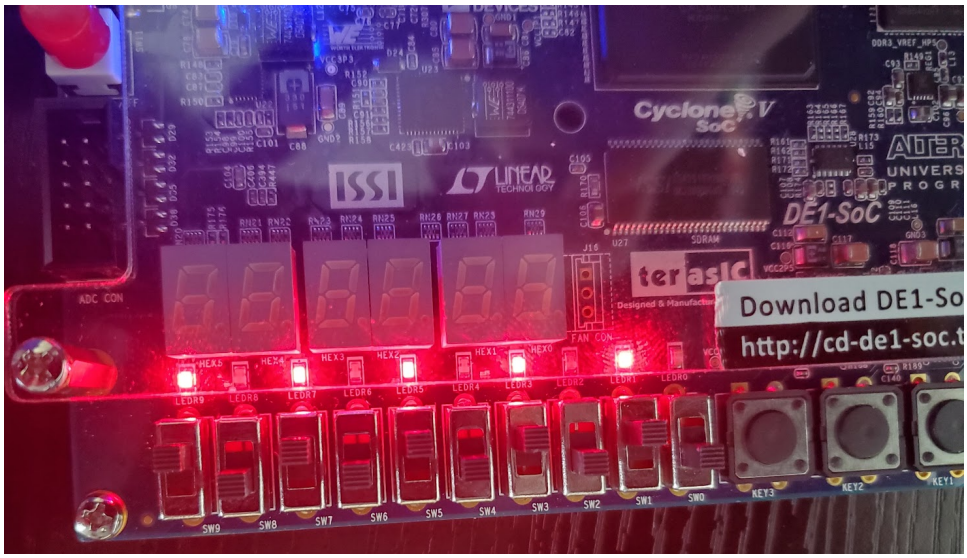


Figure 3: Successful working of the board where each switch directly corresponding to its adjacent LED

Logic utilization (in ALMs)	1 / 32,070 (< 1 %)
Total registers	0
Total pins	20 / 457 (4 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (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 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 4: %utlization

Lab I Part II

```
// Simple module for DE1-SoC and DE0-CV that connects the SW switches to the LEDR lights

module part2 (SW, LEDR);
   input [9:0] SW; // slide switches
   output [9:0] LEDR; // red LEDs

   // SW[9]    = S
   // SW[3:0]  = X
   // SW[7:4]  = Y
   // LED[3:0] = M

   assign LEDR[0] =  (!SW[9] & SW[0]) | (SW[9] & SW[4]);
   assign LEDR[1] =  (!SW[9] & SW[1]) | (SW[9] & SW[5]);
   assign LEDR[2] =  (!SW[9] & SW[2]) | (SW[9] & SW[6]);
   assign LEDR[3] =  (!SW[9] & SW[3]) | (SW[9] & SW[7]);

   assign LEDR[9] = SW[9];

endmodule
```

Figure 5: Verilog code for four-bit wide 2-to-1 multiplexer

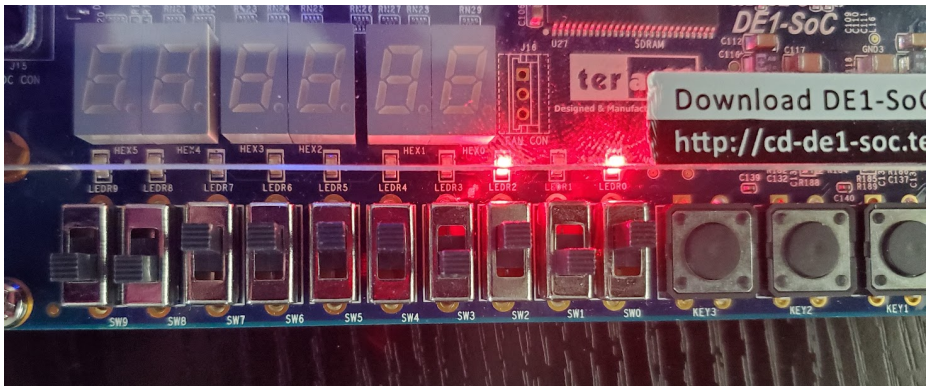


Figure 6: X = 1010, Y = 1111, SW = 0 -> M = X Board output

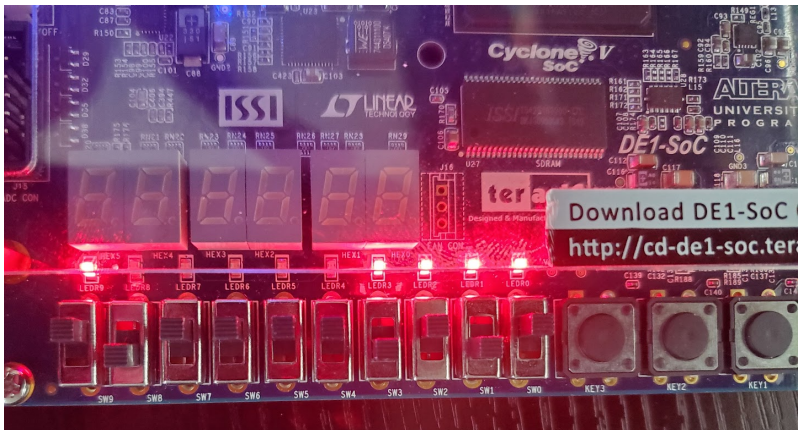


Figure 7:X = 1010, Y = 1111, SW = 1 -> M = Y Board output

Logic utilization (in ALMs)	3 / 32,070 (< 1 %)
Total registers	0
Total pins	20 / 457 (4 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (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 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 8: %utlization

Lab I Part III

```
module part3 (SW, LEDR);
   input [9:0] SW; // slide switches
   output [9:0] LEDR; // red LEDs

   // SW[9:8]   = S
   // SW[1:0]   = U
   // SW[3:2]   = V
   // SW[5:4]   = W

   assign LEDR[1:0] = SW[9] ? (SW[8] ? 0 : SW[5:4]) : (SW[8] ? SW[3:2] : SW[1:0]);

endmodule
```

Figure 9: Verilog code for Two-bit wide 3-to-1 multiplexer

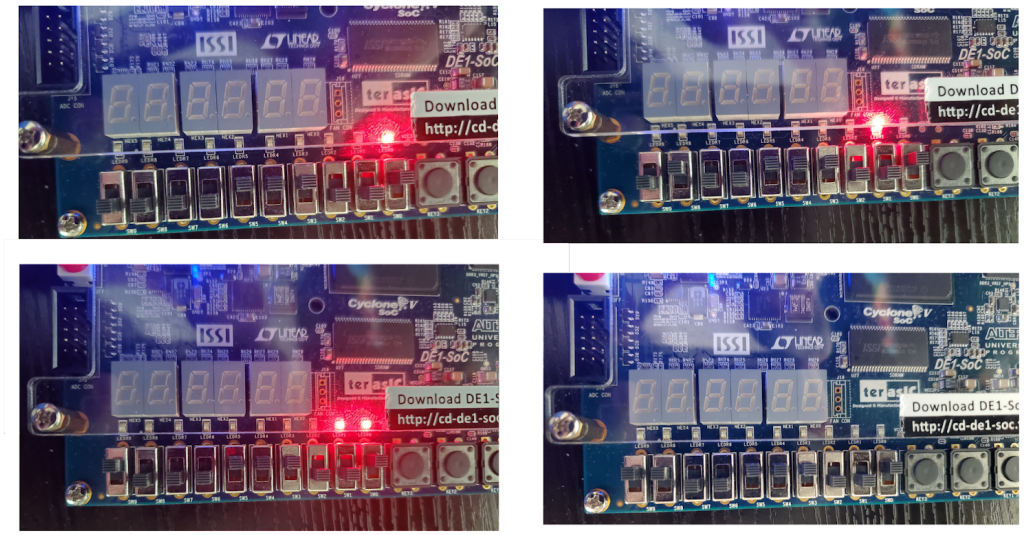


Figure 10: Board outputs for U = 01, V = 10, W = 11, Top left: S = 00 -> M = U, Top right: S = 01 -> M = V, Bottom left: S = 10 -> M = W, Bottom right: S = 11 -> M = 0

Logic utilization (in ALMs)	2 / 32,070 (< 1 %)
Total registers	0
Total pins	20 / 457 (4 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (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 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 11: %utlization

Lab I Part IV

```
module part4 (SW, HEX0);
   input [9:0] SW; // slide switches
   output [0:6] HEX0; // 7-seg 0

   // SW[1:0]  = c
   // 00 -> d
   // 01 -> E
   // 10 -> 1

   assign HEX0[0] = SW[1] | !SW[0];
   assign HEX0[1] = SW[0];
   assign HEX0[2] = SW[0];
   assign HEX0[3] = SW[1];
   assign HEX0[4] = SW[1];
   assign HEX0[5] = SW[1] | !SW[0];
   assign HEX0[6] = SW[1];

endmodule
```

Figure 12: Verilog code for 7-seg “DE1”

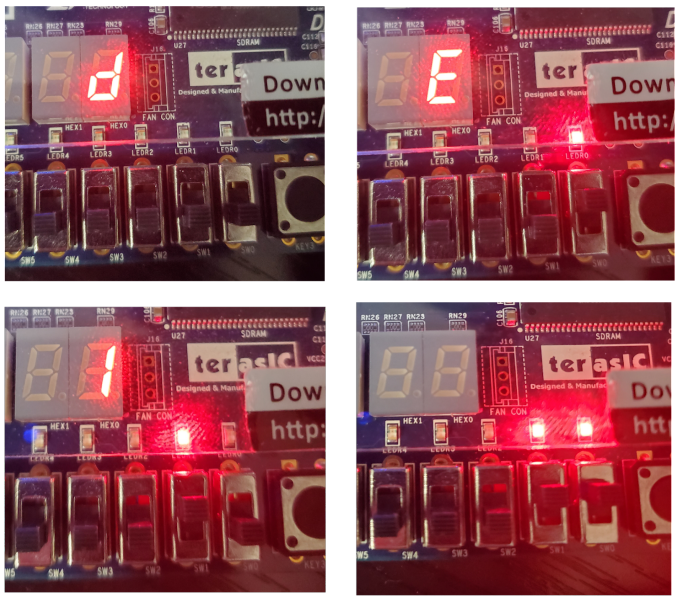


Figure 13: Board outputs for 7-seg display

Logic utilization (in ALMs)	1 / 32,070 (< 1 %)
Total registers	0
Total pins	17 / 457 (4 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (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 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 14: %utlization

Lab V Part I

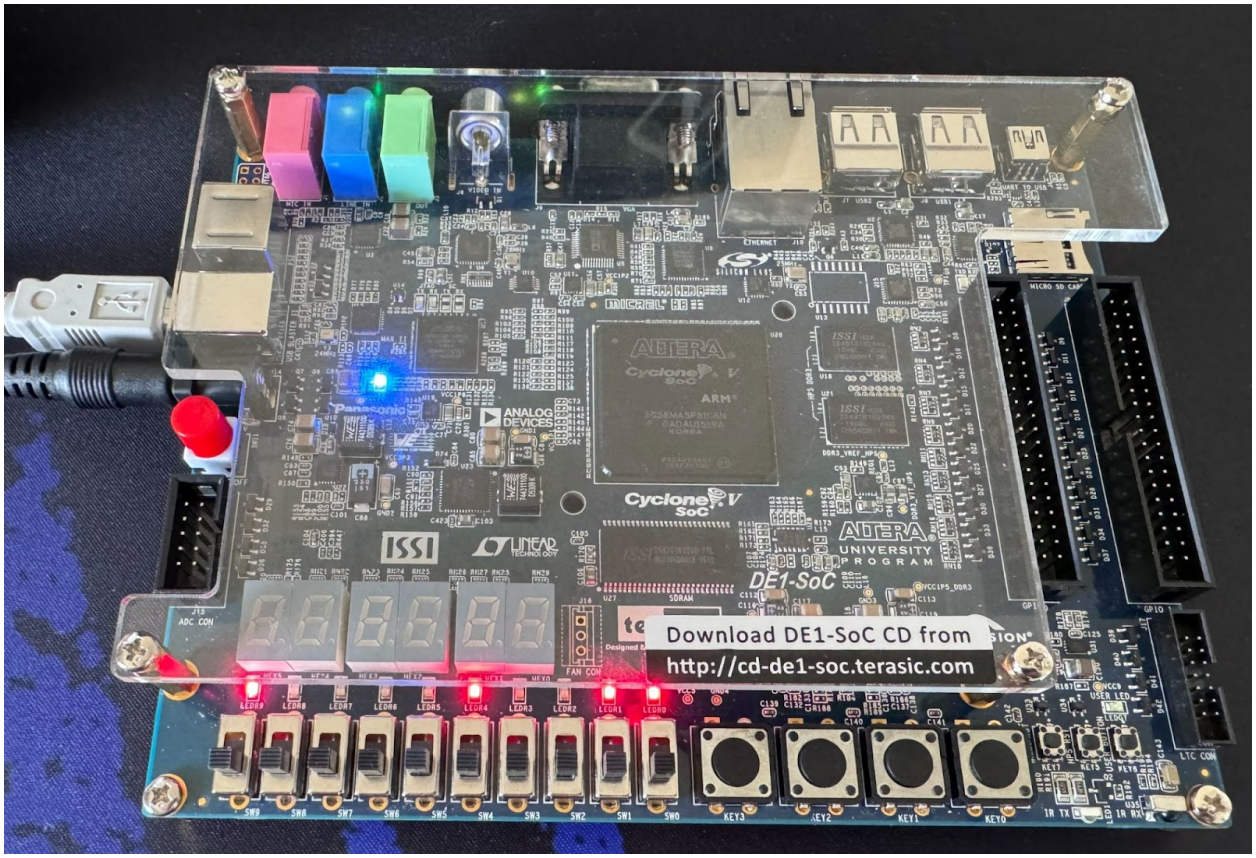


Figure 15: Board Output

	Fmax	Restricted Fmax	Clock Name	Note
1	435.92 MHz	435.92 MHz	KEY[1]	

Figure 16: Reported Fmax

Flow Status	Successful - Sat Nov 18 11:52:57 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	part1
Top-level Entity Name	part1
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	7 / 32,070 (< 1 %)
Total registers	14
Total pins	14 / 457 (3 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (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 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 17: Estimated % utilization

Lab V Part II

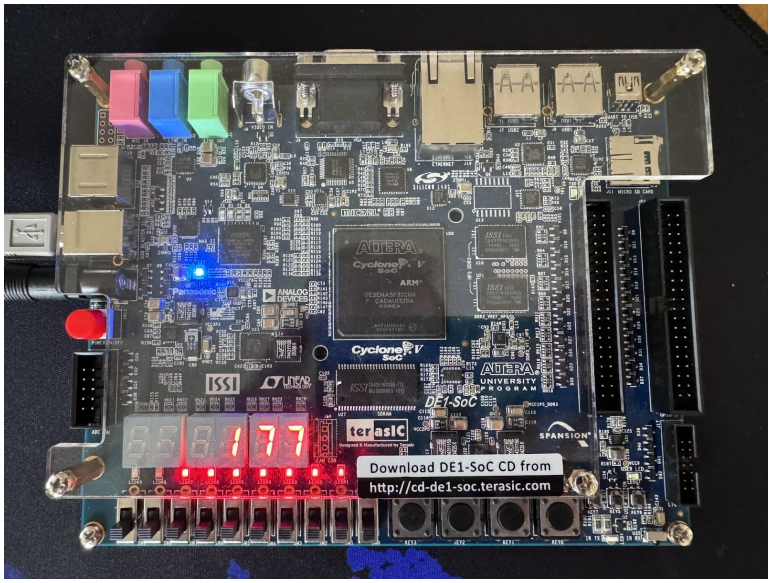


Figure 18: Board output

	Fmax	Restricted Fmax	Clock Name	Note
1	265.04 MHz	265.04 MHz	CLOCK_50	
2	295.86 MHz	295.86 MHz	counte...llover	

Figure 19: Fmax reported

Flow Status	Successful - Sat Nov 18 14:35:39 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	part2
Top-level Entity Name	part2
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	178 / 32,070 (< 1 %)
Total registers	53
Total pins	36 / 457 (8 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (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 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 20: Estimated % utilization

Lab V Part III

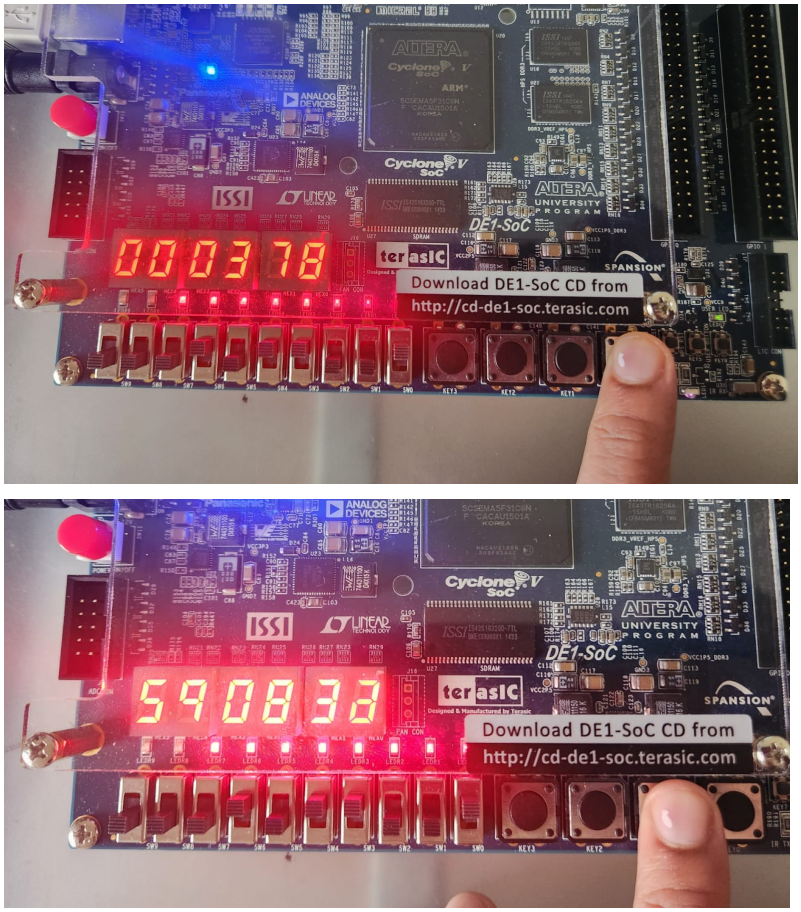


Figure 21 a,b : Board Outputs

	Fmax	Restricted Fmax	Clock Name	Note
1	300.3 MHz	300.3 MHz	CLOCK_50	
2	367.92 MHz	367.92 MHz	clk_enable	

Figure 22: Fmax reported

Logic utilization (in ALMs)	297 / 32,070 (< 1 %)
Total registers	65
Total pins	64 / 457 (14 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (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 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 23: Estimated % utilization

Module II

Lab 12 Part I

	Fmax	Restricted Fmax	Clock Name	Note
1	226.6 MHz	226.6 MHz	CLOCK_50	

Figure 24: Fmax Reported

Lab 12 Part II

	Fmax	Restricted Fmax	Clock Name	Note
1	164.04 MHz	164.04 MHz	CLOCK_50	

Figure 25: Fmax Reported

Module 2 Part III

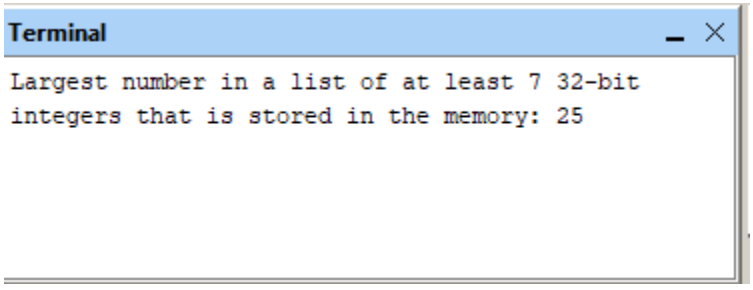


Figure 26: Terminal Output

Module 2 Part IV

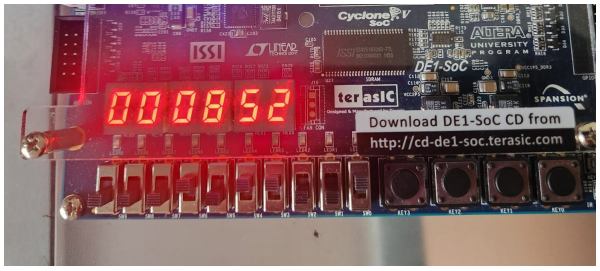


Figure 27: 7-segment display clock output

Module 2 Part V

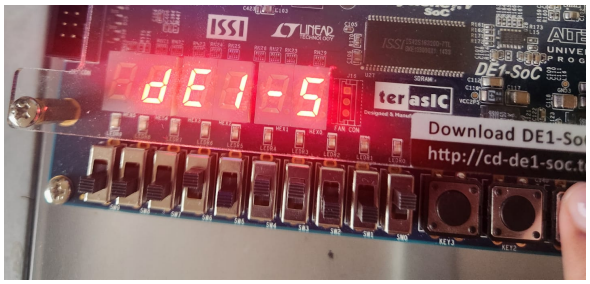


Figure 28: 7-segment display scrolling display output

Module III

Slow 1100mV 85C Model Fmax Summary

<<Filter>>

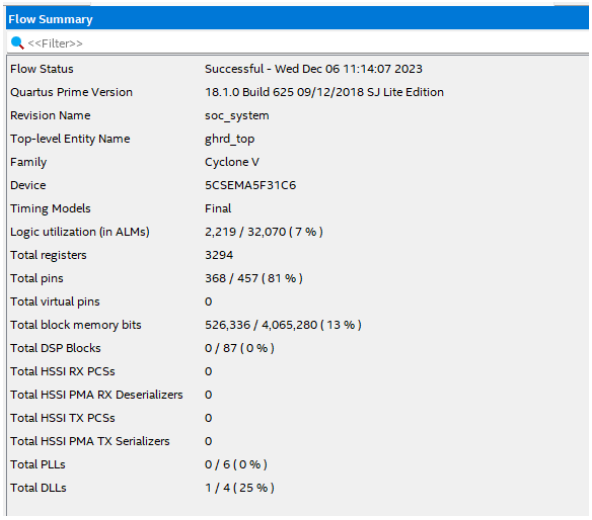
	Fmax	Restricted Fmax	Clock Name	Note
1	63.42 MHz	63.42 MHz	altera_reserved_tck	
2	89.16 MHz	89.16 MHz	clock_50_1	
3	1184.83 MHz	717.36 MHz	soc_system:u0|soc_syste...l:pll|afi_clk_write_clk	lim...in)

Figure 29: Reported Fmax

```
# Clock constraints

create_clock -name "clock_50_1" -period 20.000ns [get_ports {CLOCK_50}]
create_clock -name "clock_50_2" -period 20.000ns [get_ports {CLOCK2_50}]
create_clock -name "clock_50_3" -period 20.000ns [get_ports {CLOCK3_50}]
create_clock -name "clock_50_4" -period 20.000ns [get_ports {CLOCK4_50}]
create_clock -name "clock_27_1" -period 37.000ns [get_ports {TD_CLK27}]
```

Figure 30: clock Constraints



Flow Summary

<<Filter>>

Flow Status	Successful - Wed Dec 06 11:14:07 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	soc_system
Top-level Entity Name	ghrd_top
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	2,219 / 32,070 (7 %)
Total registers	3294
Total pins	368 / 457 (81 %)
Total virtual pins	0
Total block memory bits	526,336 / 4,065,280 (13 %)
Total DSP Blocks	0 / 87 (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 / 6 (0 %)
Total DLLs	1 / 4 (25 %)

Figure 31: Estimated % utilization

```
#define PIO_LED_COMPONENT_TYPE altera_avalon_pio
#define PIO_LED_COMPONENT_NAME pio_led
#define PIO_LED_BASE 0x0
#define PIO_LED_SPAN 32
#define PIO_LED_END 0x1f
#define PIO_LED_BIT_CLEARING_EDGE_REGISTER 0
#define PIO_LED_BIT_MODIFYING_OUTPUT_REGISTER 0
#define PIO_LED_CAPTURE 0
#define PIO_LED_DATA_WIDTH 10
#define PIO_LED_DO_TEST_BENCH_WIRING 0
#define PIO_LED_DRIVEN_SIM_VALUE 0
#define PIO_LED_EDGE_TYPE NONE
#define PIO_LED_FREQ 50000000
#define PIO_LED_HAS_IN 0
#define PIO_LED_HAS_OUT 1
#define PIO_LED_HAS_TRI 0
#define PIO_LED_IRQ_TYPE NONE
#define PIO_LED_RESET_VALUE 1023
```

Figure 32: pio_led definitions in hps_0.h

```
socfpga login: root
root@socfpga:~# ls
root@socfpga:~# cd ..
root@socfpga:/home# ls
root
root@socfpga:/home# cd ..
root@socfpga:/# ls
bin                 init                mnt                 tmp
boot                lib                 my_first_hps-fpga   usr
dev                 linuxrc             proc                var
etc                 lost+found          sbin                www
home                media               sys
root@socfpga:/# chmod +x my_first_hps-fpga
root@socfpga:/# ./my_first_hps-fpga
root@socfpga:/#
```

Figure 33: Execution and completion of my_first_hps-fpga executable

Final Module: FPGA-Based Snake Game Prototype

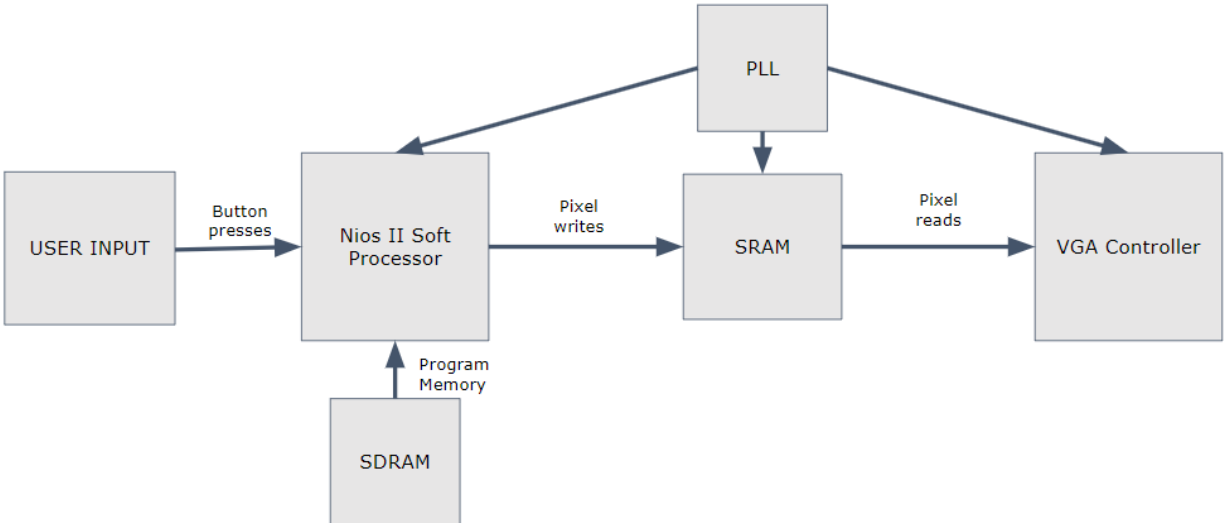


Figure 34: Initial simplified block diagram

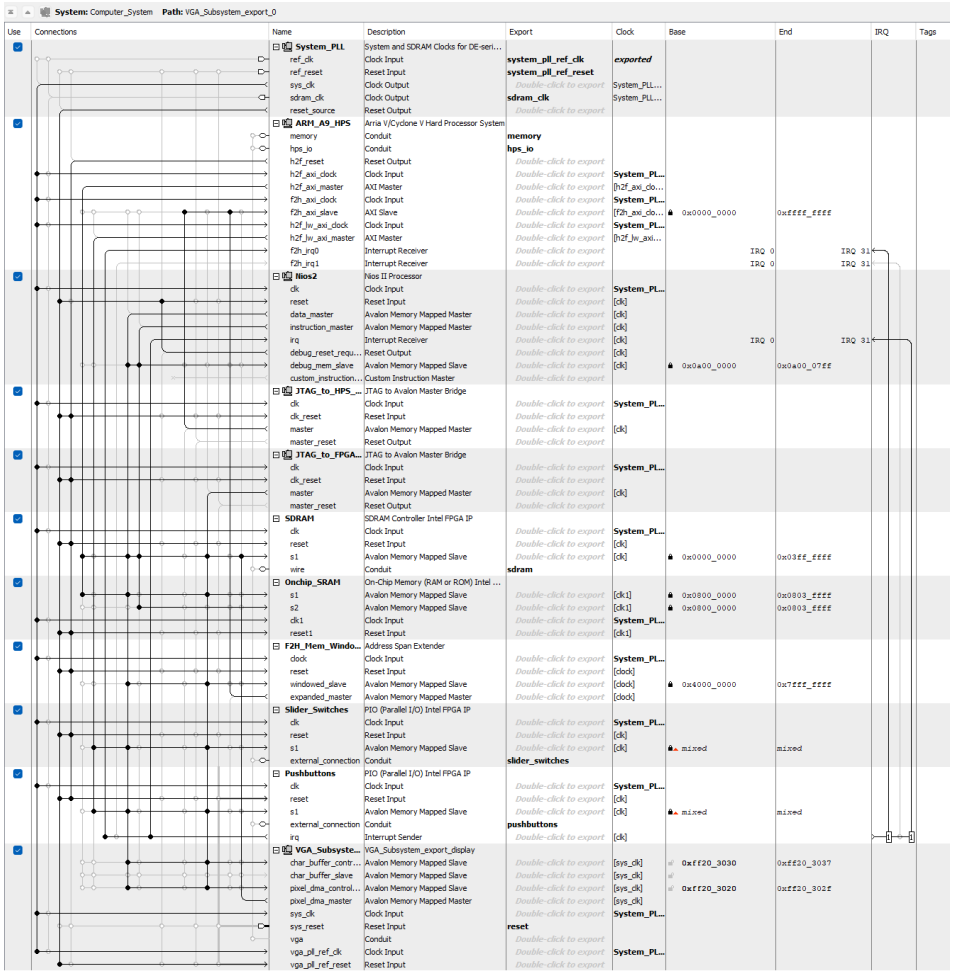


Figure 35: Top-level Qsys view

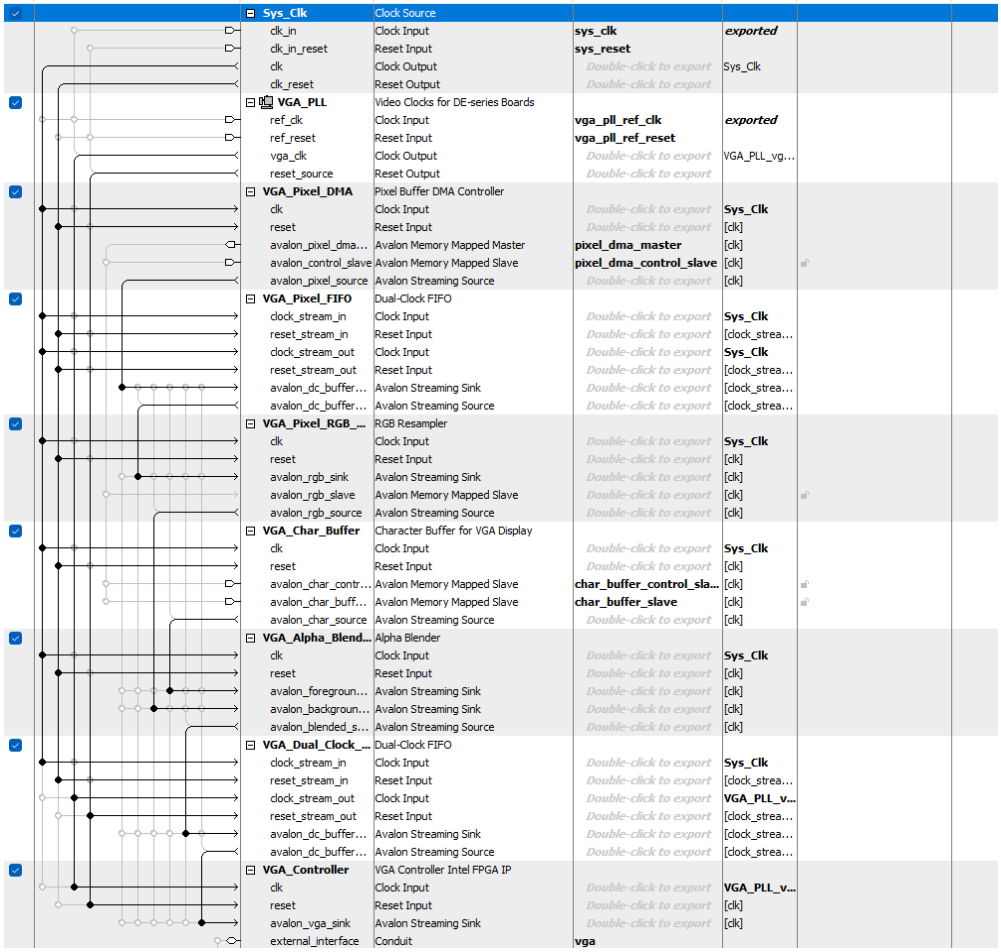


Figure 36: Qsys Altera University Program VGA subsystem

```
/*  0x0800_0000 is VGA base address. X cord is 9 bits 1-10
 *  and y cord is 8 bits 11-18. Color is a 16-bit RGB value */
void write_pixel(int x, int y, short color) {
  volatile short *vga_addr = (volatile short *)(0x08000000 + (y << 10) + (x << 1));
  *vga_addr = color;
}
```

Figure 37: C code to write a single pixel to VGA

```
/* 0x09000_0000 is base character buffer address. X coord is
 * 7 bits 0-6. Y coord is 6 bits 7-13. Then use one byte character
 * ASCII code to print it
 */
void write_char(int x, int y, char c) {
  // VGA character buffer
  volatile char *character_buffer = (char *)(0x09000000 + (y << 7) + x);
  *character_buffer = c;
}
```

Figure 38: C code to write a single ASCII character to VGA

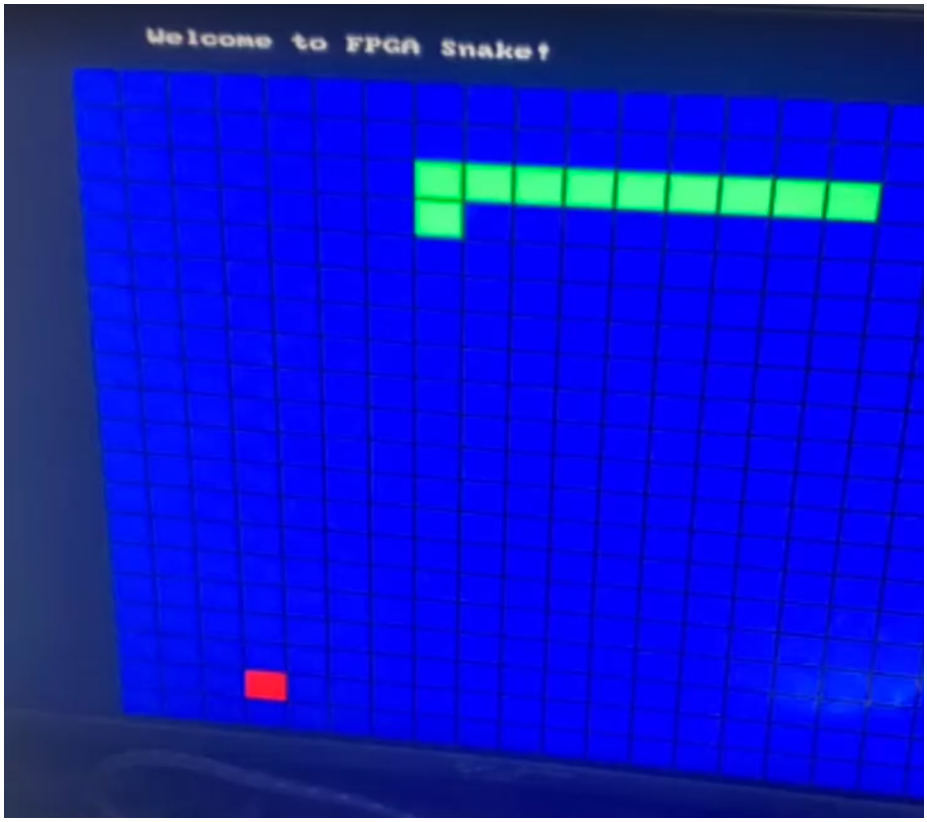


Figure 39: Completed game running on VGA monitor