Practical Homework 3

ECEN 5863

Matt Hartnett

Section 1:

Q1)
I was able to successfully install EDS and DS-5, as shown in figure 1.

Q2)
As shown in figure 2, I was able to power up the board and observe its initial power-on configuration that's loaded from the on-board flash.

Q3)
Using the Quartus programmer, I configured the FPGA with the my_first_fpga.sof file found in the system CD file. A screenshot showing the successful programming is shown in figure 3.

Q4)
I created a linux boot system on the SD card and I was able to create the Putty terminal settings as shown in figure 4.

Q5)
I was able to get the linux desktop environment to run on the DE1-SoC, however it did crash rather frequently and had a lot of trouble playing the video. I was able to capture an image of the video playing, as shown in figure 5.

Q6)

a. As shown in figure 2, I was able to get the DE1-SoC configured and set up, and the LEDs lit up in an alternating flashing pattern.
b. In order to see the percent utilization of the FPGA, I opened the project and recompiled to update the compilation report. The percent utilization was <1%, as shown in figure 6.
c. Once the board was programmed, the LEDs behaved as expected. If and only if KEY1 was pressed down, the LEDs count up based on the outlook of the PLL. This is shown in figure 7.
d. As shown in figure 8, there wasn't any subdirectories in the
e. I was able to find and play the video, as shown in figure 5, but I wasn't able to play it for long. It was very prone to crashing and it required a few tries to get as far as playing the video.

Section 2:

Q1)
As shown in figure 9, I was able to successfully get the monitor program installed and running.

Q2)
After getting the monitor program installed, I created the project as instructed and successfully got the board programmed, as shown in figures 10 through 12. The board's LEDs cycled on and off to create a wave pattern.

Upon completion of the sample project, I moved on to part 2. Shown below in figures 13 through 15, I compiled and loaded the program specified in the instructions onto the DE1-SoC. Although the instructions indicated that the result should be stored in the memory location 0x50, when I ran it, the memory address for RESULT was 0x38. This is reflected in both the value of r4, shown in figure 14, and the memory, shown in figure 15.

When I ran until the breakpoint at 0x2C, I found the value of R0 was 0x5. When I changed the program counter to skip the first instruction, I observed that the change was that R0 now contained the value 0x78, instead of the 0x8 value that it should have (shown in Fig X). In addition to that, since I manually loaded R4, instead of writing to memory address 0x38, it wrote to memory address 0x54, as shown in figure 18.

For part 3, in order to change the program to include a subroutine, I added the LARGE label, the branch instruction, and a "return" instruction (mov PC, LR). This modified code is shown in figure 19. I verified the operation by stepping through to ensure that the branch worked as expected, then I placed a breakpoint at the end of the subroutine and ran. Once the breakpoint was hit, I stepped through to confirm that the program returned to the correct place and terminated as expected. I then checked memory to verify that the largest value (0x8) was stored in the correct memory location (0x40), as shown in figure 20.

Finally, for part 4, I began by modifying the code to allow for any divisor to be passed into the DIVIDE subroutine. This modification was simple, as all it required was changing the #10s to R1s in the assembly. This functionally switched the divisor from decimal 10 to whatever was stored in R1. I then verified that these changes created the correct output with 76 as the test number.

The second modification was to the main body of the code, where instead of calling the DIVIDE subroutine once with a divisor of 10, I called the divide subroutine three different times, each with a decrementing divisor. This allowed me to first divide by 1000 to find the thousands place, then 100 for the hundreds, and lastly 10 for the tens and ones (remainder register). After each call to the DIVIDE subroutine, I wrote the appropriate value into the corresponding address in memory. This resulted in the code shown in figure 21, which results in the correct memory output, as shown in figure 22.

Q3)
This process is different from the standard ARM microprocessor software development flow partially because of the ability to run it on hardware. The ability to execute the debugger while running the board allows for a lot of new opportunities for debugging programs. When combined with the hardware capabilities of the board, such as the ability to control the lights, you can debug a lot easier than other systems with less feedback. However, this does require the board to be functional, and you do have to have the board programmed to execute this code. That could cause problems if the board is currently in use, since it would require the board to stop what it was doing to test your code.

Q4)
The value in the program counter is the memory address of the final instruction in the program. This depends on the program, but can sometimes be calculated by adding the _start memory address (usually 0x0) and the number of instructions * 0x4. This isn't always the end value of the program counter, as some programs, such as in part 4, have subroutines that are in memory after the END instruction.

Figures:

Section1:

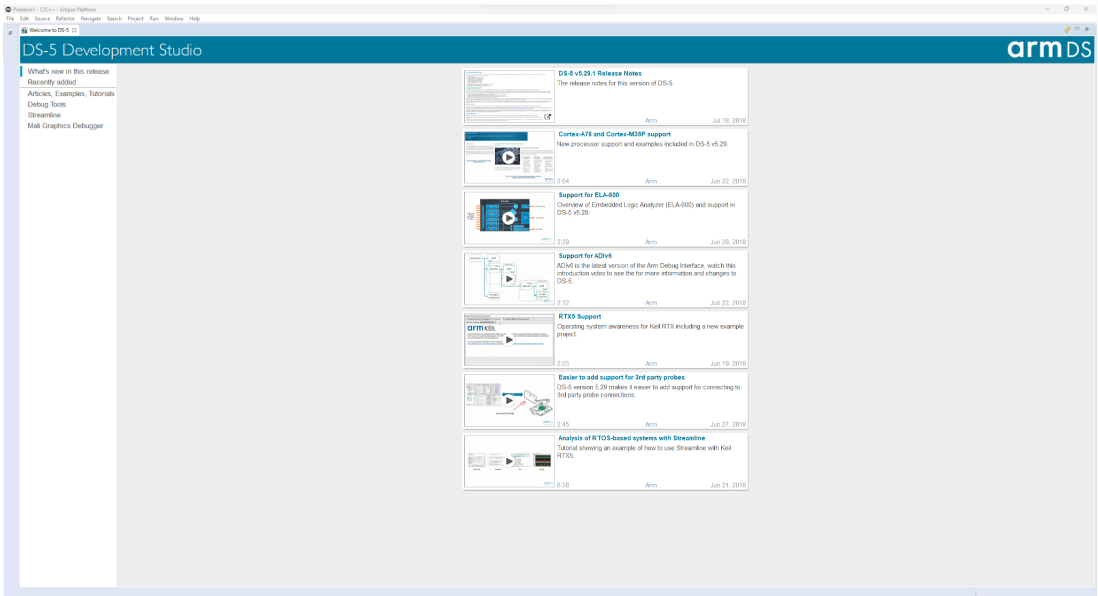


Fig 1. A screenshot of the successful installation of DS-5.

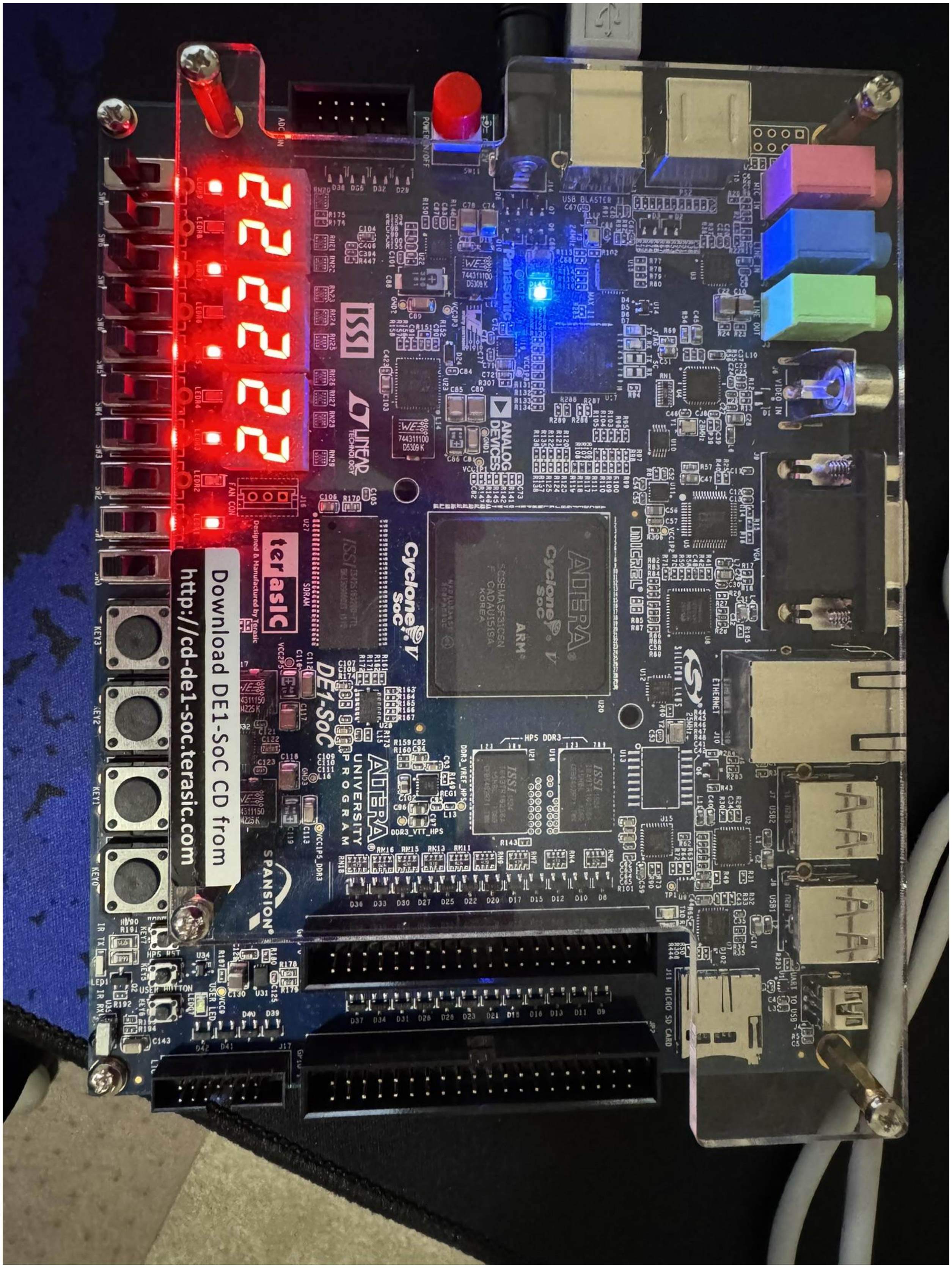


Fig 2. An image showing the DE1-SoC in operating condition with the default program running.

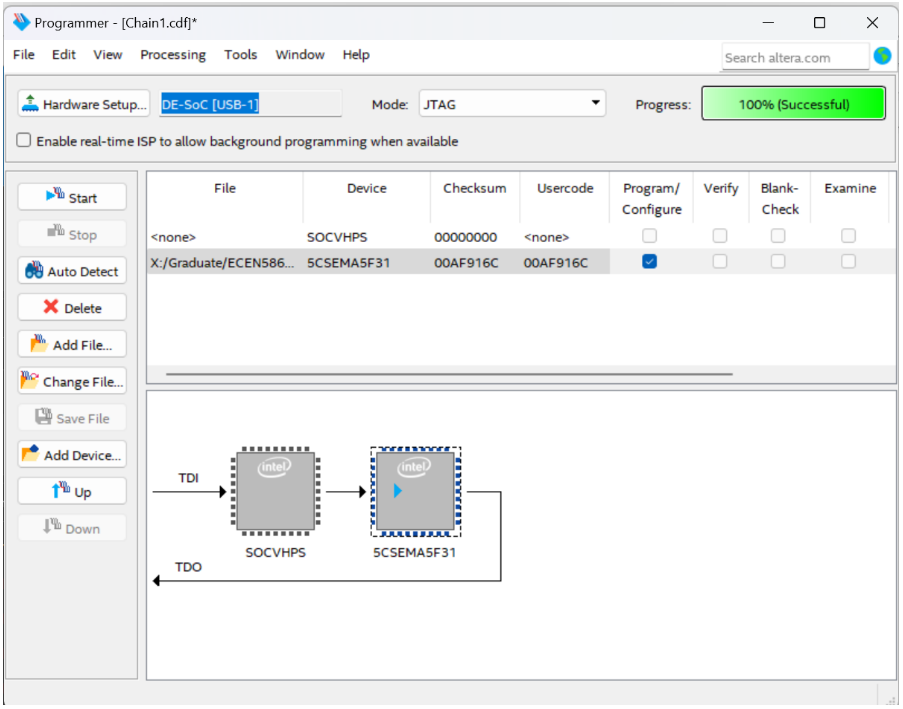


Fig 3. A successful programming of the my_first_fpag.sof configuration file onto the FPGA.

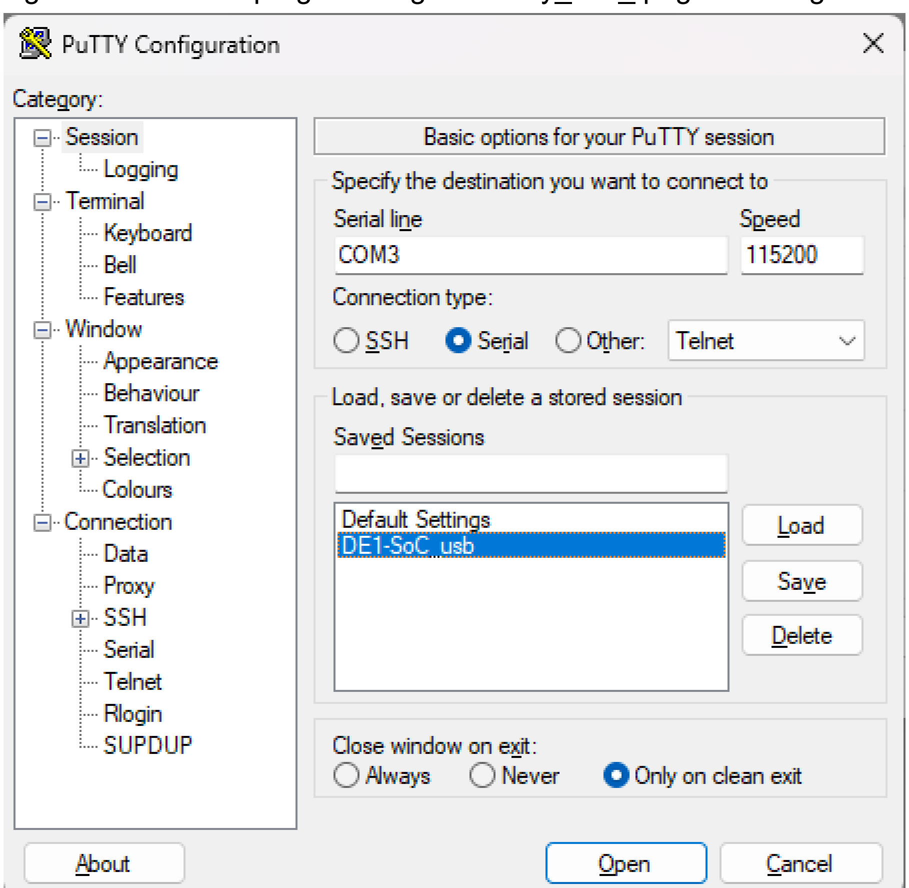
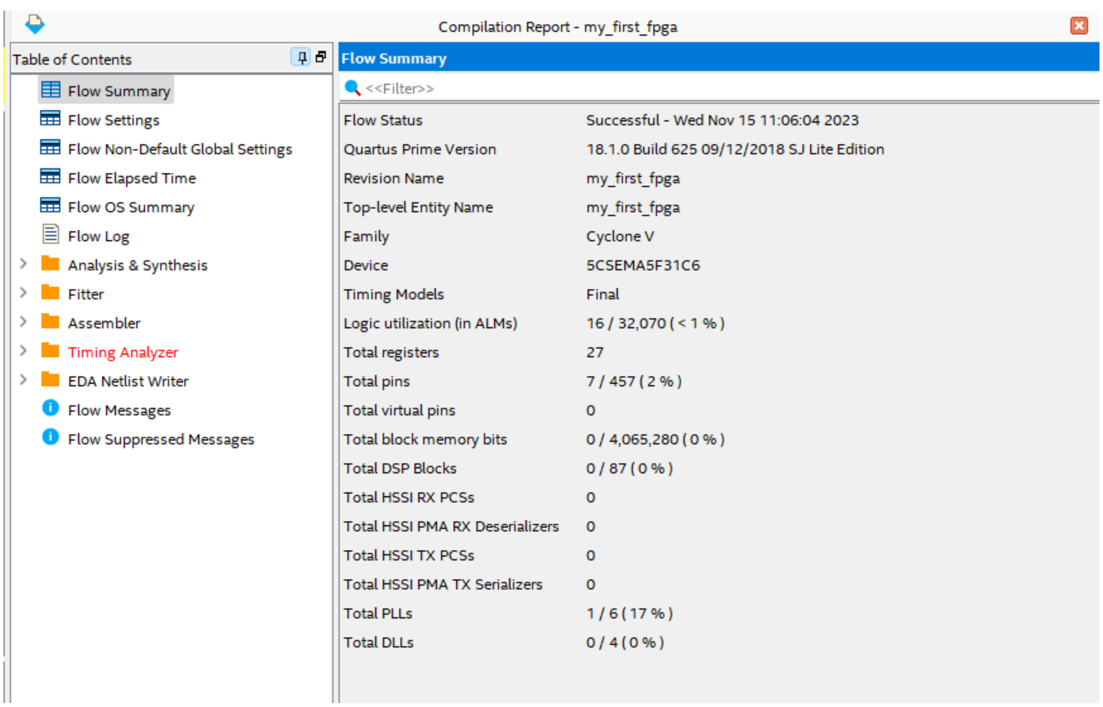


Fig 4. Correctly set up Putty terminal.

Fig 5. A fully operational desktop environment playing the santa video.



Flow Status	Successful - Wed Nov 15 11:06:04 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	my_first_fpga
Top-level Entity Name	my_first_fpga
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	16 / 32,070 (< 1 %)
Total registers	27
Total pins	7 / 457 (2 %)
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	1 / 6 (17 %)
Total DLLs	0 / 4 (0 %)

Fig 6. The compilation report for the my_first_fpga project.

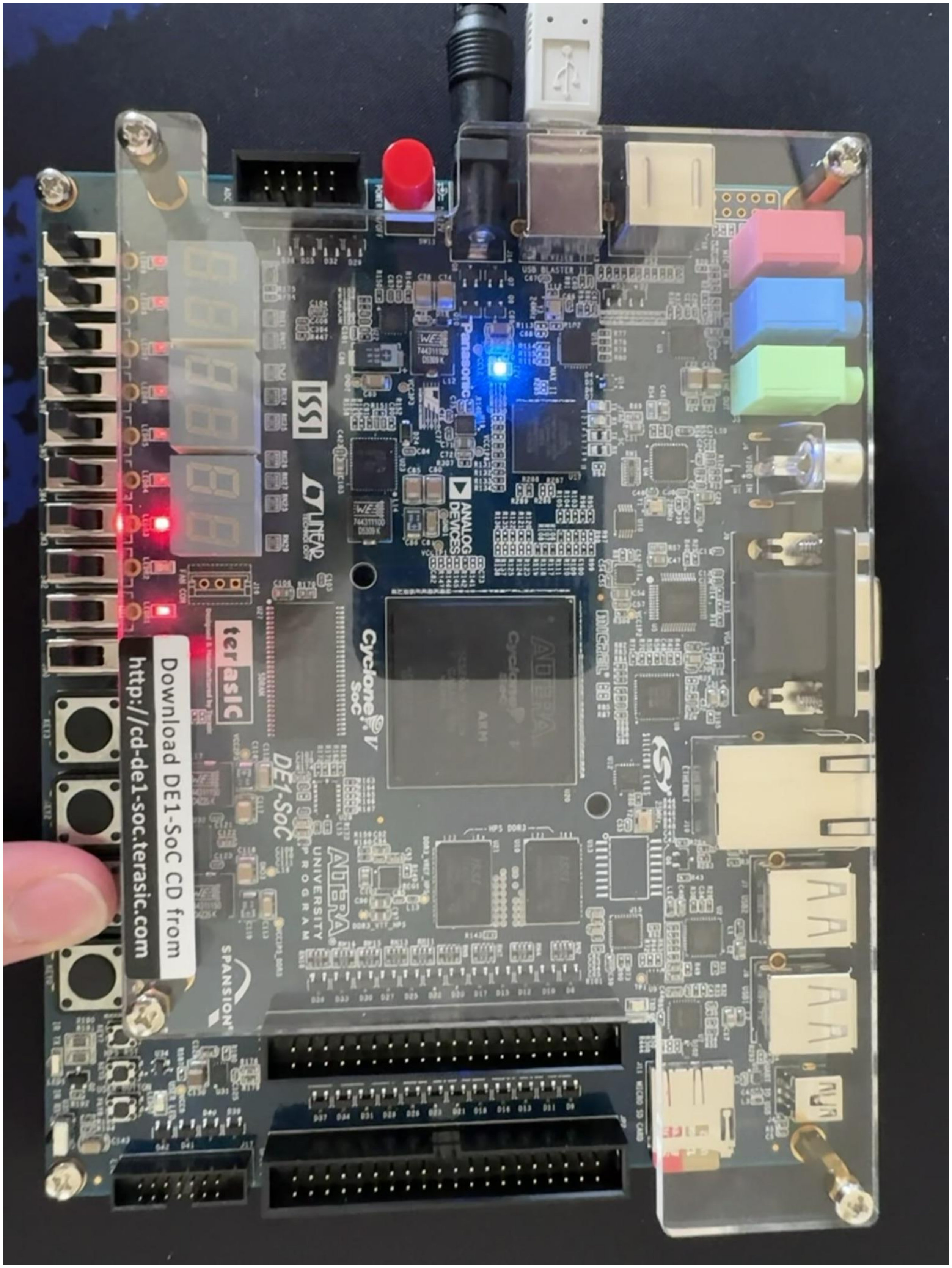


Fig 7. The DE1-SoC board with the my_first_fpga project running on it.

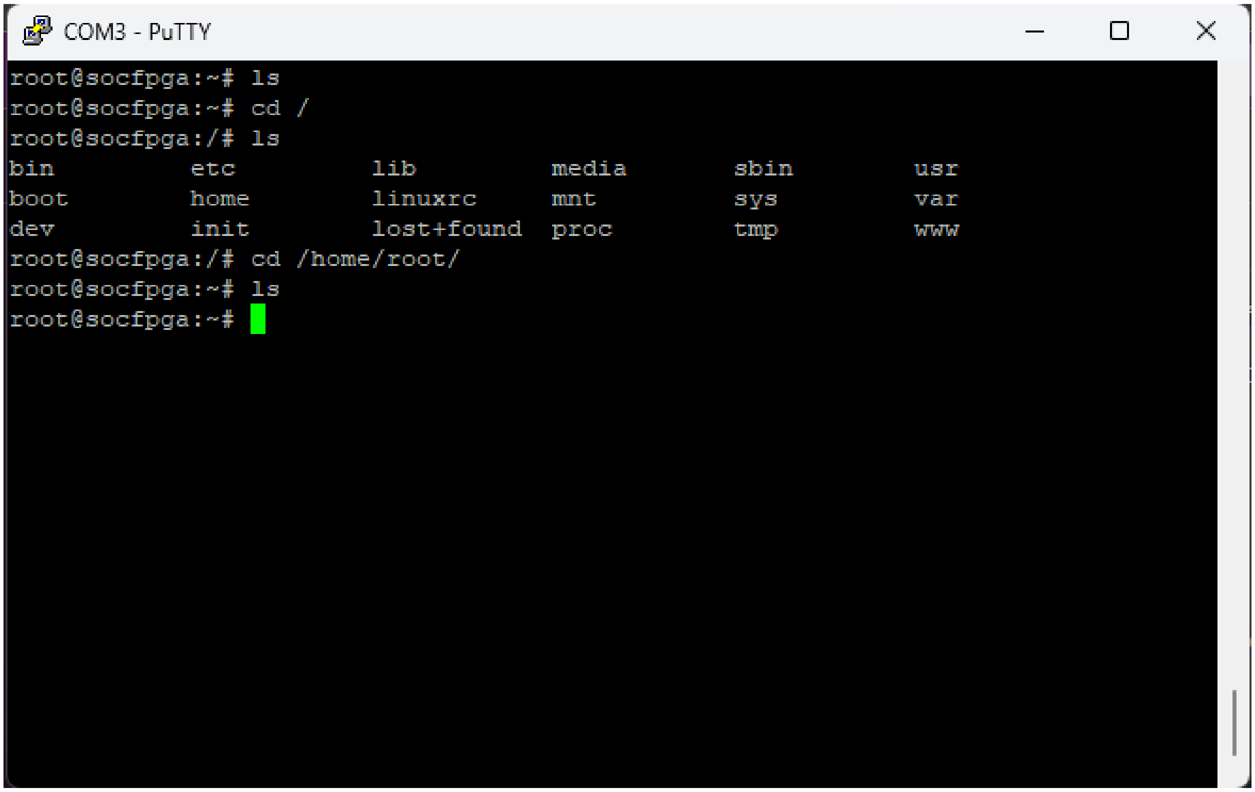


Fig 8. The putty linux terminal showing the directory layout.

Section 2:

Part 1:

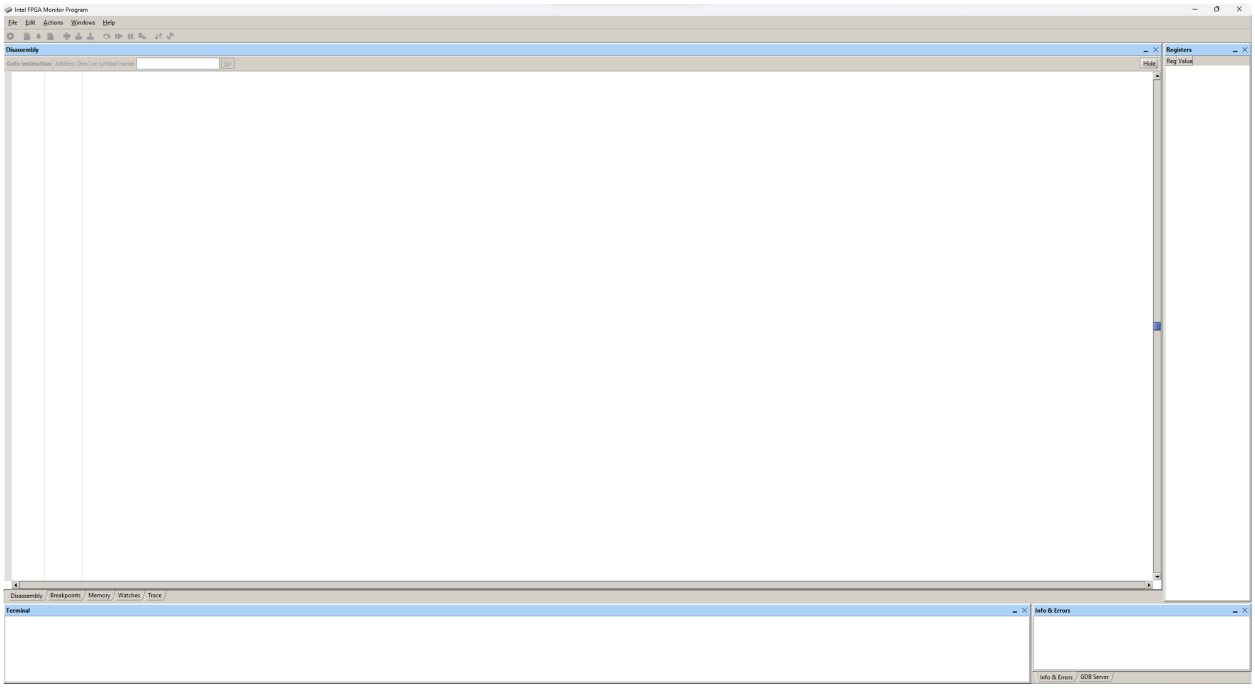


Fig 9. A screenshot showing the successful installation of the Intel FPGA Monitor Program.

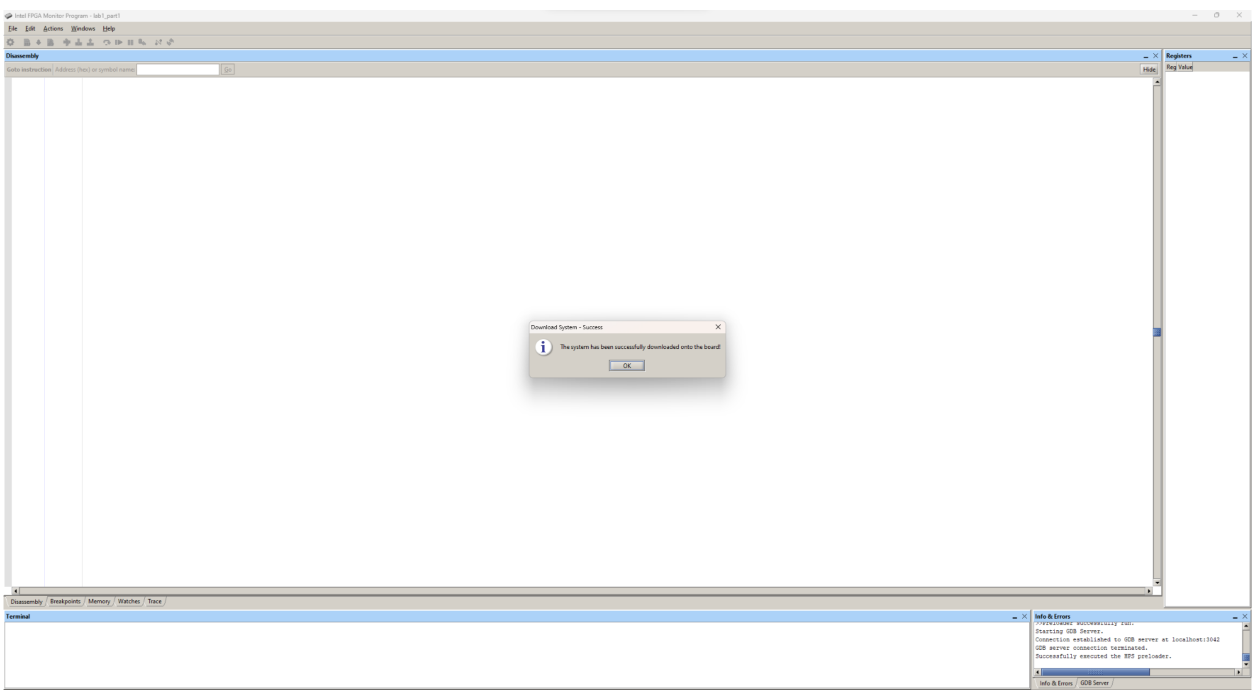


Fig 10. A screenshot showing the getting started program successfully downloaded onto the DE1-SoC.

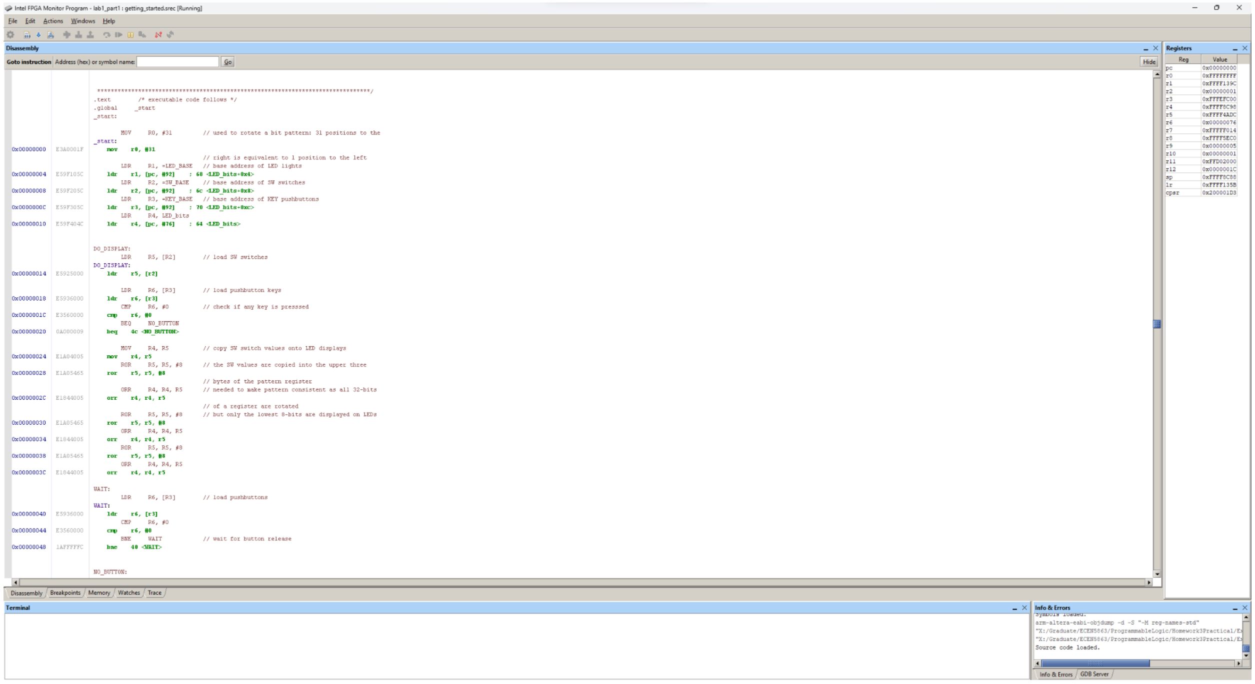


Fig 11. A screenshot showing the program running on the DE1-SoC.

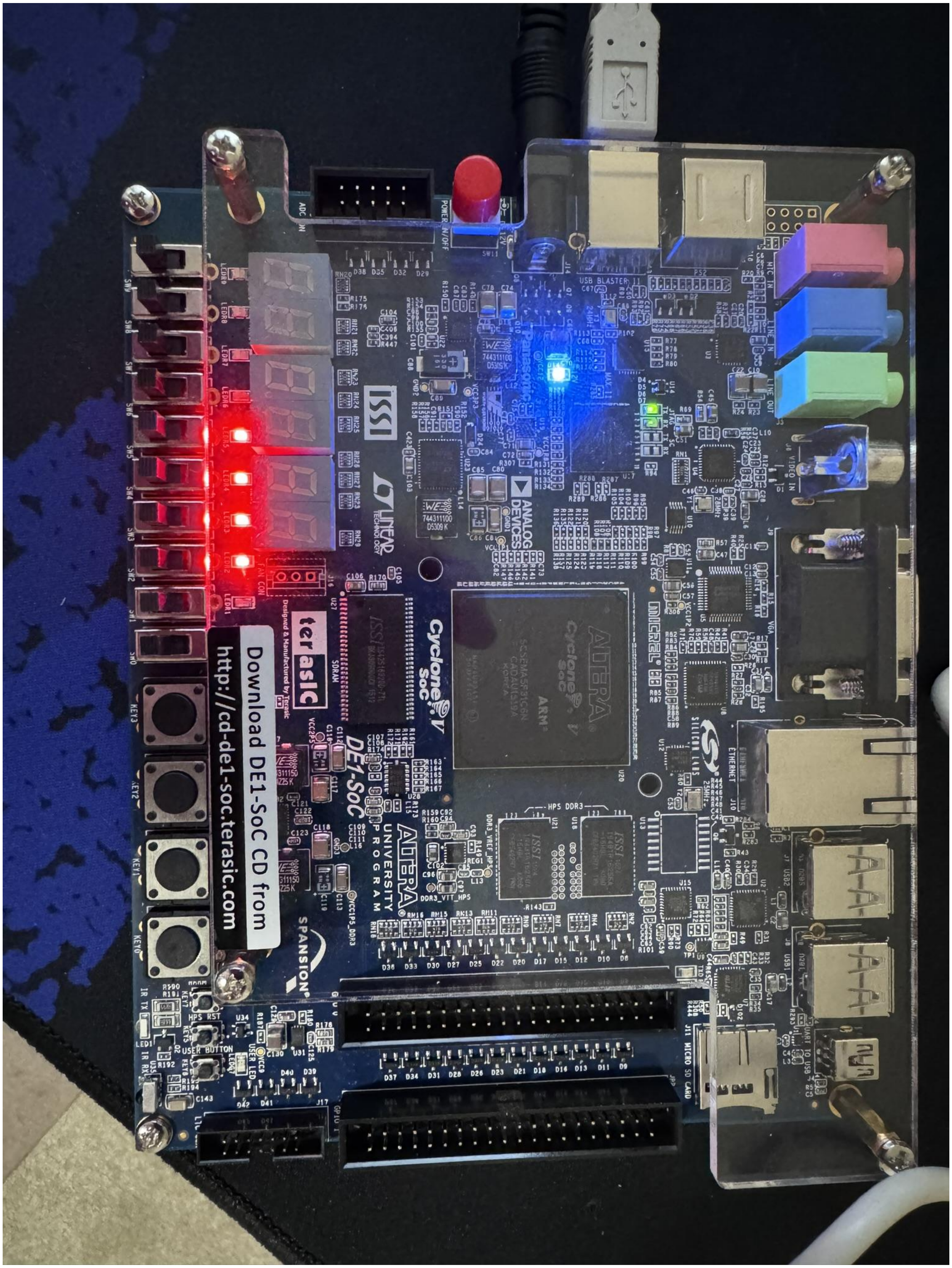


Fig 12. An image of the DE1-SoC with the getting started program running.

Part 2:

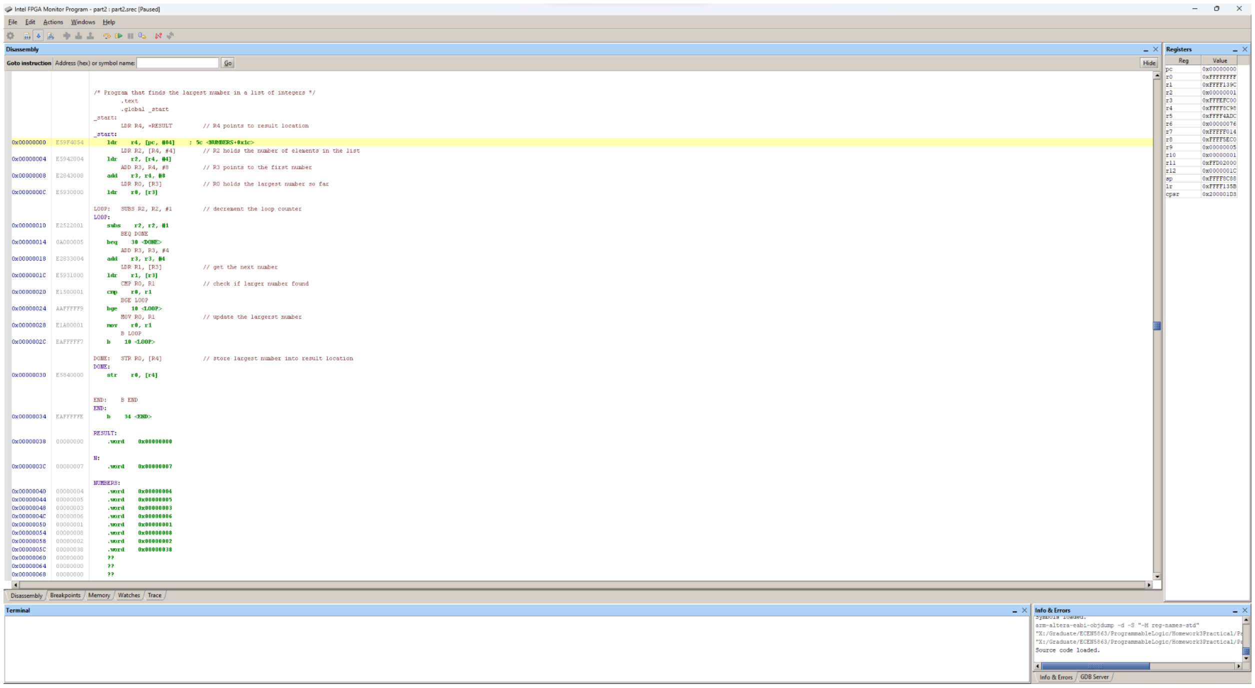


Fig 13. The program from part2, successfully compiled and loaded onto the FPGA.

Registers

Reg	Value
pc	0x00000034
r0	0x00000008
r1	0x00000002
r2	0x00000000
r3	0x00000058
r4	0x00000038
r5	0xFFFF4ADC
r6	0x00000076
r7	0xFFFFF014
r8	0xFFFF5EC0
r9	0x00000005
r10	0x00000001
r11	0xFFD02000
r12	0x0000001C
sp	0xFFFF8C88
lr	0xFFFF135B
cpsr	0x600001D3

Fig 14. The register list after execution of the program, showing 0x8 loaded in R0.

	+0x0	+0x4	+0x8	+0xc
0x00000000	E59F4054	E5942004	E2843008	E5930000
0x00000010	E2522001	0A000005	E2833004	E5931000
0x00000020	E1500001	AAFFFFF9	E1A00001	EAFFFFF7
0x00000030	E5840000	EAFFFFFE	00000008	00000007
0x00000040	00000004	00000005	00000003	00000006
0x00000050	00000001	00000008	00000002	00000038
0x00000060	00000000	00000000	00000000	00000078

Fig 15. A screenshot of the memory after the program execution, showing the largest value (0x8) being stored in 0x38.

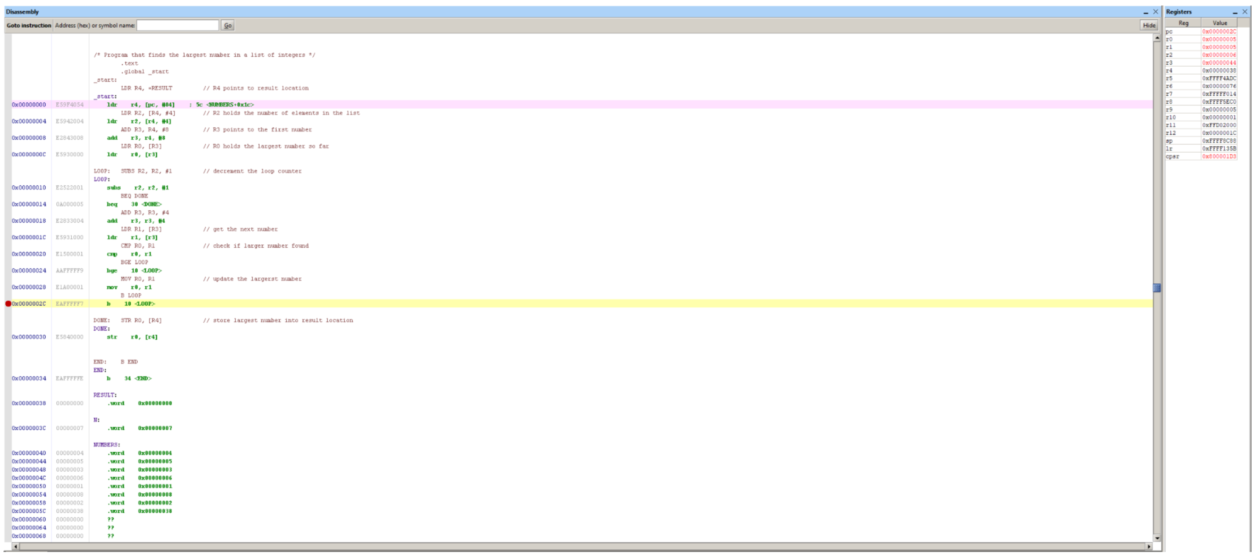


Fig 16. The program execution was halted due to the breakpoint at line 0x2C.

Registers

Reg	Value
pc	0x00000034
r0	0x00000078
r1	0x00000000
r2	0x00000000
r3	0x00000070
r4	0x00000054
r5	0xFFFF4ADC
r6	0x00000076
r7	0xFFFFF014
r8	0xFFFF5EC0
r9	0x00000005
r10	0x00000001
r11	0xFFD02000
r12	0x0000001C
sp	0xFFFF8C88
lr	0xFFFF135B
cpsr	0x600001D3

Fig 17. The values of the registers after modification and execution.

	+0x0	+0x4	+0x8	+0xc
0x00000000	E59F4054	E5942004	E2843008	E5930000
0x00000010	E2522001	0A000005	E2833004	E5931000
0x00000020	E1500001	AAFFFFF9	E1A00001	EAFFFFF7
0x00000030	E5840000	EAFFFFFE	00000008	00000007
0x00000040	00000004	00000005	00000003	00000006
0x00000050	00000001	00000078	00000002	00000038
0x00000060	00000000	00000000	00000000	00000078

Fig 18. Address 0x54 in memory, which contains the value of R0 (0x78)

Part 3:

```
                        _start:
                                LDR R4, =RESULT         // R4 points to result location
                        _start:
0x00000000   E59F405C       ldr    r4, [pc, #92]    ; 64 <NUMBERS+0x1c>
                                LDR R2, [R4, #4]        // R2 holds the number of elements in the list
0x00000004   E5942004       ldr    r2, [r4, #4]
                                ADD R3, R4, #8          // R3 points to the first number
0x00000008   E2843008       add    r3, r4, #8
                                BL LARGE                // Call to the subroutine LARGE
0x0000000C   EB000000       bl     14 <LARGE>


                        END:    B END
                        END:
0x00000010   EAFFFFFE       b      10 <END>


                        LARGE:                          // Subroutine to find the largest number
                                LDR R0, [R3]            // R0 holds the largest number so far
                        LARGE:
0x00000014   E5930000       ldr    r0, [r3]

                        LOOP:   SUBS R2, R2, #1         // decrement the loop counter
                        LOOP:
0x00000018   E2522001       subs   r2, r2, #1
                                BEQ DONE
0x0000001C   0A000005       beq    38 <DONE>
                                ADD R3, R3, #4
0x00000020   E2833004       add    r3, r3, #4
                                LDR R1, [R3]            // get the next number
0x00000024   E5931000       ldr    r1, [r3]
                                CMP R0, R1              // check if larger number found
0x00000028   E1500001       cmp    r0, r1
                                BGE LOOP
0x0000002C   AAFFFFF9       bge    18 <LOOP>
                                MOV R0, R1              // update the largerst number
0x00000030   E1A00001       mov    r0, r1
                                B LOOP
0x00000034   EAFFFFF7       b      18 <LOOP>

                        DONE:   STR R0, [R4]            // store largest number into result location
                        DONE:
0x00000038   E5840000       str    r0, [r4]
                                MOV PC, LR                      // Return to calling program
0x0000003C   E1A0F00E       mov    pc, lr
```

Fig 19. The modified and compiled code for the program with the subroutine added. This is after the code was executed.

	+0x0	+0x4	+0x8	+0xc
0x00000000	E59F405C	E5942004	E2843008	EB000000
0x00000010	EAFFFFFE	E5930000	E2522001	0A000005
0x00000020	E2833004	E5931000	E1500001	AAFFFFF9
0x00000030	E1A00001	EAFFFFF7	E1200070	E1A0F00E
0x00000040	00000008	00000007	00000004	00000005

Fig 20. This is program memory after execution, showing that the largest value is in the correct place in memory (0x40).

Part 4:

```
                                    .global _start
                            _start:
                                    LDR R4, =N
                            _start:
0x00000000    E59F405C          ldr    r4, [pc, #92]    ; 64 <Digits+0x4>
                                    ADD R5, R4, #4      // R5 points to the decimal digits storage location
0x00000004    E2845004          add    r5, r4, #4
                                    LDR R4, [R4]        // R4 holds N
0x00000008    E5944000          ldr    r4, [r4]
                                    MOV R0, R4          // parameter for DIVIDE goes in R0
0x0000000C    E1A00004          mov    r0, r4
                                    MOV R1, #1000       // divisor parameter for DIVIDE goes in R1
0x00000010    E3A01FFA          mov    r1, #1000    ; 0x3e8
                                    BL DIVIDE
0x00000014    EB000008          bl     3c <DIVIDE>
                                    STRB R1, [R5, #3]   // Thousands digit is in R1
0x00000018    E5C51003          strb    r1, [r5, #3]

                                    MOV R1, #100        // divisor parameter for DIVIDE goes in R1
0x0000001C    E3A01064          mov    r1, #100    ; 0x64
                                    BL DIVIDE
0x00000020    EB000005          bl     3c <DIVIDE>
                                    STRB R1, [R5, #2]   // Hundreds digit is in R1
0x00000024    E5C51002          strb    r1, [r5, #2]

                                    MOV R1, #10         // divisor parameter for DIVIDE goes in R1
0x00000028    E3A0100A          mov    r1, #10
                                    BL DIVIDE
0x0000002C    EB000002          bl     3c <DIVIDE>
                                    STRB R1, [R5, #1]   // Tens digit is in R1
0x00000030    E5C51001          strb    r1, [r5, #1]
                                    STRB R0, [R5]       // Ones digit is in R0
0x00000034    E5C50000          strb    r0, [r5]

                            END:    B END
                            END:
0x00000038    EAFFFFFE          b     38 <END>

                            /* Subroutine to perform the integer division R0 / 10.
                            * Returns: quotient in R1, and remainder in R0
                            */
                            DIVIDE: MOV R2, #0
                            DIVIDE:
0x0000003C    E3A02000          mov    r2, #0

                            CONT:   CMP R0, R1
                            CONT:
0x00000040    E1500001          cmp    r0, r1
                                    BLT DIV_END
0x00000044    BA000002          blt    54 <DIV_END>
                                    SUB R0, R1
0x00000048    E0400001          sub    r0, r0, r1
                                    ADD R2, #1
0x0000004C    E2822001          add    r2, r2, #1
                                    B CONT
0x00000050    EAFFFFFA          b     40 <CONT>

                            DIV_END: MOV R1, R2         // return quotient in R1 (remainder is in R0)
                            DIV_END:
0x00000054    E1A01002          mov    r1, r2
                                    BX LR
```

Fig 21. A screenshot showing the modified code for part 4. This screenshot was taken after execution of the code.

	+0x0	+0x4	+0x8	+0xc
0x00000000	E59F405C	E2845004	E5944000	E1A00004
0x00000010	E3A01FFA	EB000008	E5C51003	E3A01064
0x00000020	EB000005	E5C51002	E3A0100A	EB000002
0x00000030	E5C51001	E5C50000	EAFFFFFE	E3A02000
0x00000040	E1500001	BA000002	E0400001	E2822001
0x00000050	EAFFFFFA	E1A01002	E12FFF1E	00002694
0x00000060	09080706	0000005C	00000000	00000000
0x00000070	00000000	00000080	00000000	00000000

Fig 22. A screenshot showing the memory location 0x60, which contains the decimal breakdown of N=9876.