ASSIGNMENT 2 STAGE 3 REPORT

Sreemanti Dey

January 2022

1 Objective

The modules that have been designed include ALU, Register File, Memory, Decoder, Condition-Checker, Flag-Update and finally the processor.

2 Assumptions

VHDL
edaplayground
Aldec Riviera Pro 2020.04 used for simulation
Mentor Precision 2021.1 used for synthesis

3 Implementation details

I have made a testbench for the final processor file that instantiates all the components that have been made in the 2 stages and the EPWave shows all the signals present.
Apart from this, I have also made testbenches for all the components - ALU, register, flag updater, condition-checker, memory and included their EPwaves and synthesis results. I have also included synthesis result of the final processor and have added 3 programs that check all the instructions that we have been asked to implement in our processor. prog1.s tests mov, sub, add, ldr, str instructions. prog2.s tests bne, cmp instructions. prog3.s tests b instruction. thus covering the instruction set we have been asked to implement.

4 Processor

This is the main file, that integrates all the entities that have been made in this stage. I have instantiated all the components in the file and then I have defined signals that will take care of mapping of the components. The signals include the various flags, the predicate, the input and output for memory and register file and also for the ALU operands. I have made a process that has clock in its sensitivity list and it has a case statement that is basically my FSM and this FSM updates its states with every cycle. The FSM has 9 states, that have been named by a signal named curr. I initialise my curr to 0 then update it to 1 to go to the next state. After that based on the instruction class, I update my state to 30,31,32, 30 being the DP state 3, 31 being the state common to load-store and 32 for the last stage of the branch instruction. Then I have states 40,41,42 and finally the last stage for DT-load is named 5. These are the 9 states that do the work as mentioned in the slides. I have used pc as a local signal to make the implementation simpler. All control signals are assigned combinationally but the registers IR, A, B, DR, RES get their value inside the states. My flag-updater, memory, register file are clocked and rest all are concurrent assignment statements. I conditionally make write enable as on or off for various instructions to function properly. SBit is always 0 for mov, add, sub while it is always 1 for cmp. ALU operand decides plus offset or minus offset for ldr, str instructions, mw signal i,e, memory read/write conditionally writes (only in case of str), register write is enabled when it is a dp instruction and not cmp and also when it is a ldr instruction. Also, write data of the register conditionally gets input from alu output or memory output. The pc also calculates its address with the help of the ALU. Thus overall my glue logic takes care of the fact that my data path and control path are connected together and they function properly.

4.1 Testbench

I have hardcoded a value 80 in the loop that updates the clock every cycle. This value can be changed depending on number of instructions in the instruction memory also later on when we learn swi instructions, then we will be able to halt our code.
Note: When testing out various programs, kindly update the memory.vhd file and ensure that the addresses specified start from 64 since my program memory partition of the memory array starts from address 64.

4.2 EPWave for prog1

Here is a picture of the simulation results I have achieved by EPWave for prog1.s

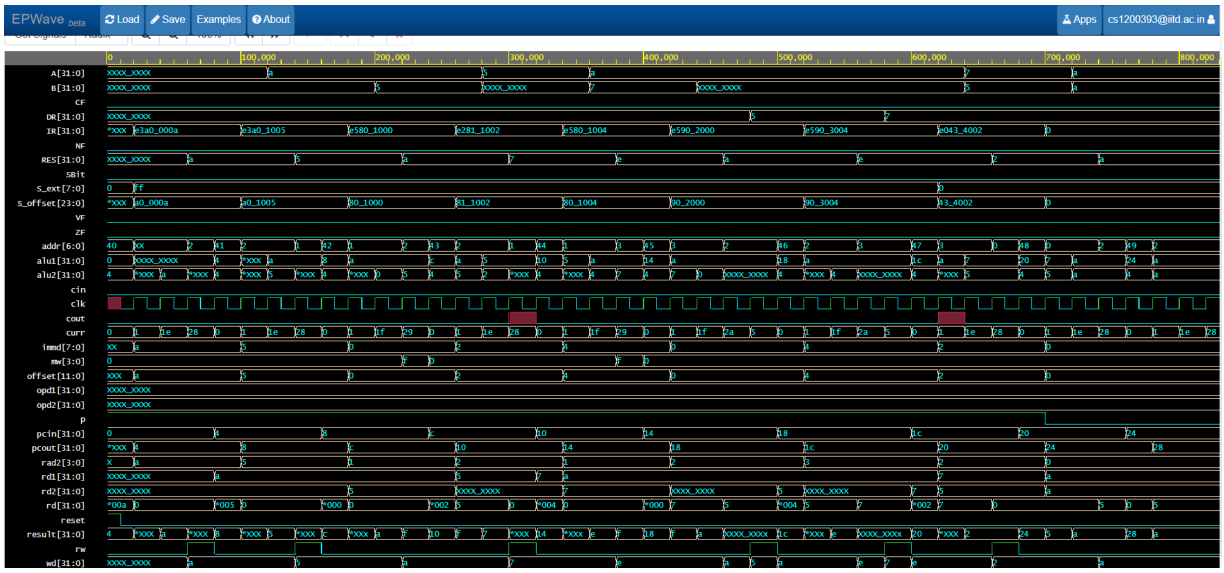

Figure 1: Program 1

The code corresponding to this program is in prog1.s file. This file tests ldr, str, add, mov, sub instructions. The code is supposed to give the result 2 at the end of the program, and thus the correctness of the processor is verified by wd i.e. my write data signal having the value 2 when rw is 1 for the last time. Also, the rw is '1' everytime register file needs to be updated, thus by traversing the assembly code step by step I have checked the correctness of EPWave, which indicates correctness of my code.

4.3 EPWave for prog2

Here is a picture of the simulation results I have achieved by EPWave for prog2.s

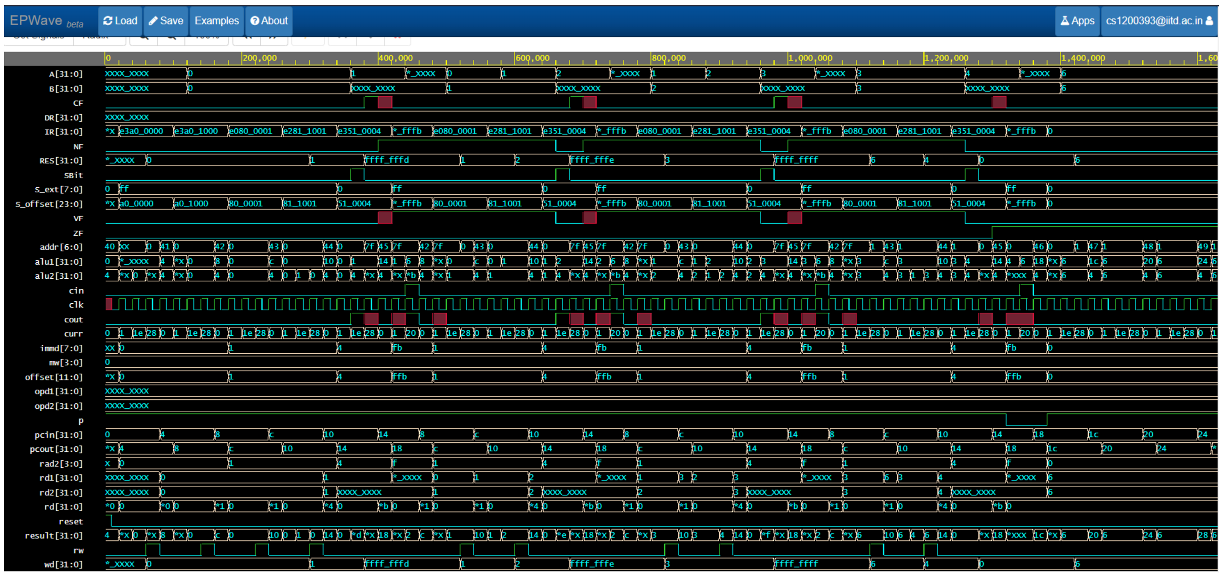

Figure 2: Program 2

The code corresponding to this program is in prog2.s file. This file tests bne, mov, cmp, add instructions. The code is supposed to give the result 6 at the end of the program, and thus the correctness of the processor is verified by wd i.e. my write data signal having the value 6 (that updates r0) and value 4(that updates r1) when rw is 1 for the last time. Also, the rw is '1' everytime register file needs to be updated, thus by traversing the assembly code step by step I have checked the correctness of EPWave, which indicates correctness of my code.

4.4 EPWave for prog3

Here is a picture of the simulation results I have achieved by EPWave

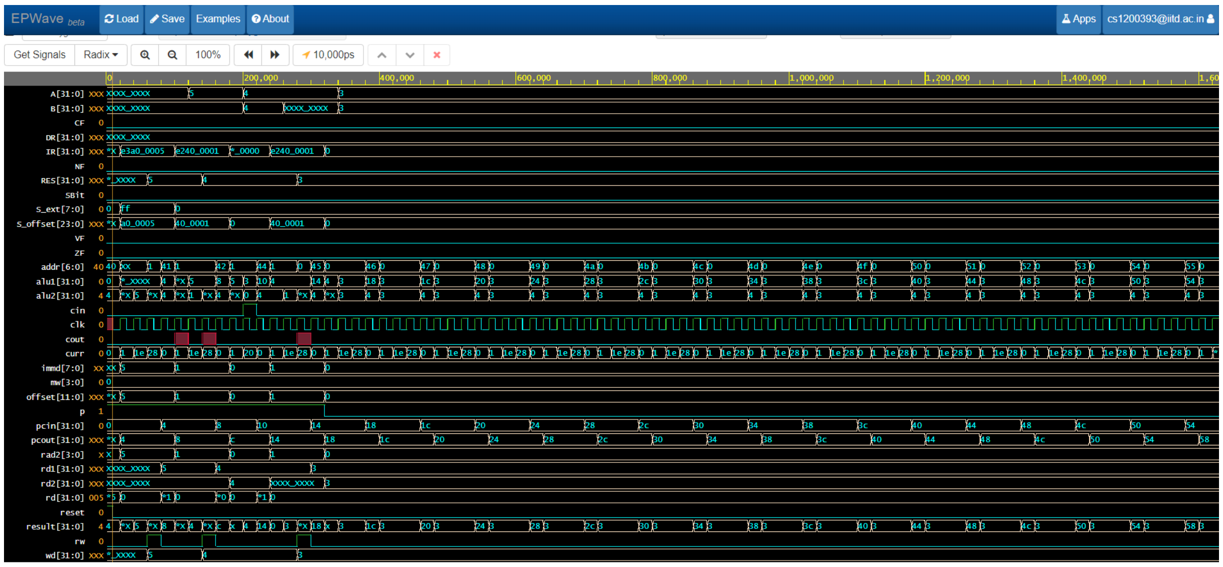

Figure 3: Program 3

The code corresponding to this program is in prog3.s file. This file tests b, mov, sub instructions. The code is supposed to give the result 3 at the end of the program, and thus the correctness of the processor is verified by wd i.e. my write data signal having the value 3 when rw is 1 for the last time. Also, the 3rd instruction is skipped since we have used b instruction here, thus verifying program correctness.

4.5 Synthesis results

```
# Info: Device Utilization for 7A100TCSG324
# Info: *************************************************************
# Info: Resource                          Used    Avail    Utilization
# Info: -------------------------------------------------------------
# Info: IOs                               2       210        0.95%
# Info: Global Buffers                    0       32         0.00%
# Info: LUTs                              0       63400      0.00%
# Info: CLB Slices                        0       15850      0.00%
# Info: Dffs or Latches                   0       126800     0.00%
# Info: Block RAMs                        0       135        0.00%
# Info: DSP48E1s                          0       240        0.00%
# Info: -------------------------------------------------------------
# Info: ***************************************************************
# Info: Library: work    Cell: Processor    View: beh_Processor
# Info: ***************************************************************
# Info:  Number of ports :                          2
# Info:  Number of nets :                            0
# Info:  Number of instances :                       0
# Info:  Number of references to this view :         0
# Info: Total accumulated area :
# Info:  Number of gates :                           0
# Info:  Number of accumulated instances :           0
# Info: *****************************
# Info:  IO Register Mapping Report
# Info: *****************************
# Info: Design: work.Processor.beh_Processor
# Info: +----------+-----------+----------+----------+----------+
# Info: | Port     | Direction |   INFF   |  OUTFF   |  TRIFF   |
# Info: +----------+-----------+----------+----------+----------+
# Info: | clk      | Input     |          |          |          |
# Info: +----------+-----------+----------+----------+----------+
# Info: | reset    | Input     |          |          |          |
# Info: +----------+-----------+----------+----------+----------+
# Info: Total registers mapped: 0
```

4.6 ALU

4.6.1 Details

I have written case statements inside a process that takes in a,b,cin,opcode in its sensitivity list. I have declared a variable temp that contains the result of the operation performed on zero-extended a and zero-extended b, from where we give MSB to carry out and the rest to the result. In case of operations that don't change carry, carry is don't care value, so I have just set value of carry out to carry in.

4.6.2 ALU_tb

This is the testbench I have made for testing the various opcodes in ALU. It tests all the opcodes with different values of a, b and cin and throws an assertion error if the result does not match with the intended result.

4.6.3 Simulation result via EPWave

Here is a picture of the simulation results I have achieved by EPWave

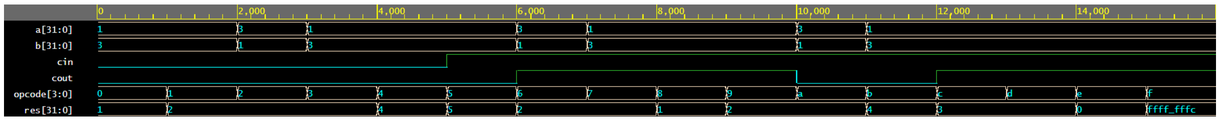

Figure 4: ALU

4.6.4 Synthesis result

Here is a picture of the synthesis results - that includes number of LUTs used, number of IOs, buffers and so on.

```
# Info: Device Utilization for 7A100TCSG324
# Info: ***************************************************************
# Info: Resource                          Used    Avail   Utilization
# Info: ---------------------------------------------------------------
# Info: IOs                               102     210      48.57%
# Info: Global Buffers                    0       32        0.00%
# Info: LUTs                              167     63400     0.26%
# Info: CLB Slices                        41      15850     0.26%
# Info: Dffs or Latches                   0       126800    0.00%
# Info: Block RAMs                        0       135       0.00%
# Info: DSP48E1s                          0       240       0.00%
# Info: ---------------------------------------------------------------
# Info: *****************************************************
# Info: Library: work    Cell: ALU    View: beh_ALU
# Info: *****************************************************
# Info:  Number of ports :                          102
# Info:  Number of nets :                           405
# Info:  Number of instances :                      304
# Info:  Number of references to this view :          0
# Info: Total accumulated area :
# Info:  Number of LUTs :                           167
# Info:  Number of Primitive LUTs :                 168
# Info:  Number of LUTs with LUTNM/HLUTNM :           2
# Info:  Number of MUX CARRYs :                      64
# Info:  Number of accumulated instances :          401
```

4.7 Register

4.7.1 Details

Register File contains an array of 16 std logic vectors of 32-bits each. Since there are 16 vectors, I have made address of 4 bits. Its inputs include two read addresses, one write address, one data input, one write enable and a clock. There are two data outputs on which contents of the array elements selected by read addresses are continuously available. If write enable is active, at clock edge the input data gets written in the array element selected by write address.

4.7.2 Register_tb

This is the testbench I have made for testing the read and write operations of the register file. I have written some data into the register first at different addresses and after that I have checked using read operation whether the data that is present is the correct one or not. It throws an assertion error if the result does not match with the correct data.

4.7.3 Simulation result via EPWave

Here is a picture of the simulation results I have achieved by EPWave

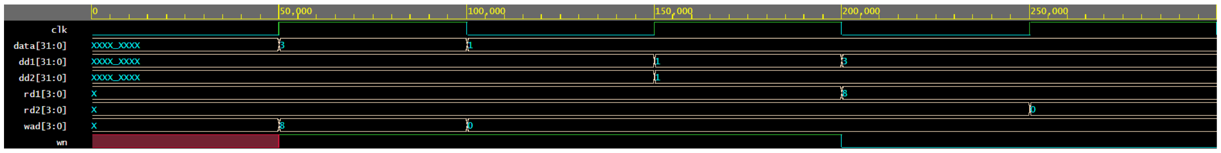

Figure 5: Register

4.7.4 Synthesis result

Here are my synthesis results - that includes number of LUTs used, number of IOs, buffers and so on.

```
# Info: Device Utilization for 7A100TCSG324
# Info: ***************************************************************
# Info: Resource                          Used    Avail   Utilization
# Info: ---------------------------------------------------------------
# Info: IOs                               110     210      52.38%
# Info: Global Buffers                    1       32        3.12%
# Info: LUTs                              48      63400     0.08%
# Info: CLB Slices                        12      15850     0.08%
# Info: Dffs or Latches                   0       126800    0.00%
# Info: Block RAMs                        0       135       0.00%
# Info: Distributed RAMs
# Info:    RAM32M                         10
# Info:    RAM64M                         2
# Info: DSP48E1s                          0       240       0.00%
# Info: ---------------------------------------------------------------
# Info: *************************************************************
# Info: Library: work    Cell: regtr    View: register_arch
# Info: *************************************************************
# Info:  Number of ports :                          110
# Info:  Number of nets :                           220
# Info:  Number of instances :                      111
# Info:  Number of references to this view :          0
# Info: Total accumulated area :
# Info:  Number of LUTs :                            48
# Info:  Number of Primitive LUTs :                  48
# Info:    Number of LUTs as Distributed RAM :       48
# Info:  Number of accumulated instances :          123
```

4.8 Memory

4.8.1 Details

My memory has twice the size as present in the last stage - the first half I use for data memory and second half I use for program memory. The two parts have been described below:

1. Program memory contains an array of 64 std logic vectors of 32-bits. Since there are 64 std logic vectors hence I have provided a 6-bit input for my address. Program Memory has one read port. Read operations are modeled such that read is unclocked (like a combinational circuit). And program memory starts from the 64th location in the memory array.

2. Data memory contains an array of 64 std logic vectors of 32-bits. Data Memory has one read port and one write port. Since there are 64 std logic vectors hence I have provided a 6-bit input for my address. Read/write operations are modeled such that write is clocked whereas read is unclocked (like a combinational circuit). I have provided 4 write enable signals to provide byte level write operation, i.e. write enable is a std logic vector of 4 bits where a set bit implies that part of the input data will be written.

4.8.2 Memory_tb

This is the testbench I have made for testing the read and write operations of the memory. I have written some data into the memory first at addresses starting from 64 onwards and after that I have checked using read operation whether the data that is present is the correct one or not. It throws an assertion error if the result does not match with the intended result.

4.8.3 Simulation result via EPWave

Here is a picture of the simulation results I have achieved by EPWave

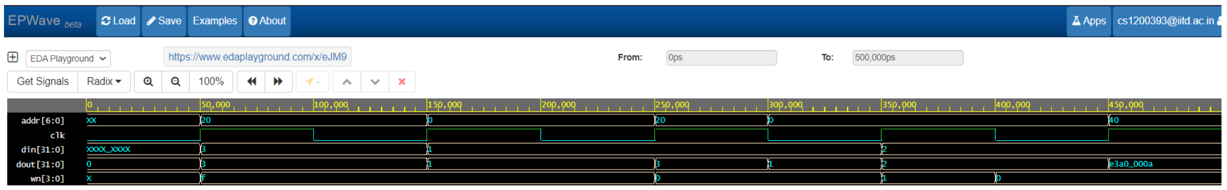

Figure 6: Memory

4.8.4 Synthesis result

Here is my synthesis results - that includes number of LUTs used, number of IOs, buffers and so on.

```
# Info: ***************************************************************
# Info: Device Utilization for 7A100TCSG324
# Info: ****************************************************************
# Info: Resource                         Used    Avail   Utilization
# Info: ----------------------------------------------------------------
# Info: IOs                              76      210      36.19%
# Info: Global Buffers                   1       32        3.12%
# Info: LUTs                             64      63400     0.10%
# Info: CLB Slices                       16      15850     0.10%
# Info: Dffs or Latches                  0       126800    0.00%
# Info: Block RAMs                       0       135       0.00%
# Info: Distributed RAMs
# Info:    RAM128X1S                     32
# Info: DSP48E1s                         0       240       0.00%
# Info: ----------------------------------------------------------------
# Info: ******************************************************
# Info: Library: work    Cell: mem    View: mem_arch
# Info: ******************************************************
# Info:  Number of ports :                          76
# Info:  Number of nets :                          152
# Info:  Number of instances :                      77
# Info:  Number of references to this view :         0
# Info: Total accumulated area :
# Info:  Number of LUTs :                           64
# Info:  Number of Primitive LUTs :                 64
# Info:    Number of LUTs as Distributed RAM :      64
# Info:  Number of accumulated instances :         108
```

4.9 Condition-checker

4.9.1 Details

Condition-checking is done by a file called cond.vhd that checks various conditions and evaluates the value of predicate whether it is true or false.

4.9.2 Simulation result via EPWave

Here is a picture of the simulation results I have achieved by EPWave

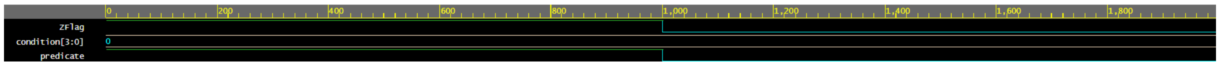

Figure 7: Condition Checker

4.9.3 Synthesis result

Here is my synthesis results - that includes number of LUTs used, number of IOs, buffers and so on.

```
# Info: Device Utilization for 7A100TCSG324
# Info: ***************************************************************
# Info: Resource                          Used    Avail   Utilization
# Info: ---------------------------------------------------------------
# Info: IOs                               9       210       4.29%
# Info: Global Buffers                    0       32        0.00%
# Info: LUTs                              4       63400     0.01%
# Info: CLB Slices                        1       15850     0.01%
# Info: Dffs or Latches                   0       126800    0.00%
# Info: Block RAMs                        0       135       0.00%
# Info: DSP48E1s                          0       240       0.00%
# Info: ---------------------------------------------------------------
# Info: ********************************************************
# Info: Library: work    Cell: cond    View: cond_arch
# Info: ********************************************************
# Info:  Number of ports :                           9
# Info:  Number of nets :                           21
# Info:  Number of instances :                      13
# Info:  Number of references to this view :         0
# Info: Total accumulated area :
# Info:  Number of LUTs :                            4
# Info:  Number of Primitive LUTs :                  4
# Info:  Number of accumulated instances :          13
```

4.10 Flag-Updater

4.10.1 Details

This file updates all flags based on the input instruction type, mainly this is based on the pdf file uploaded by sir that explains the circuit for maintaining various flags.

4.10.2 Flag-updater_tb

This tests the signals.

4.10.3 Simulation result via EPWave

Here is a picture of the simulation results I have achieved by EPWave

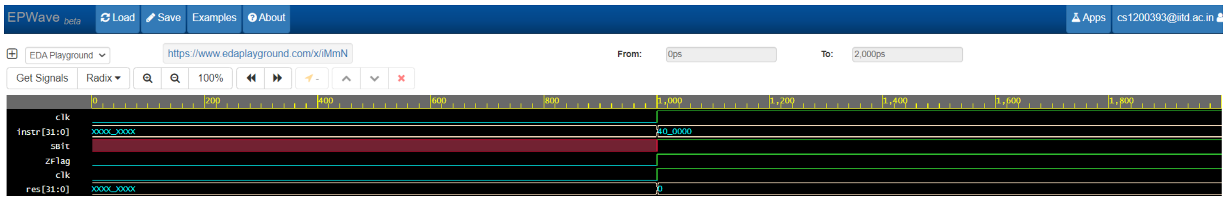

Figure 8: Flag Updater

4.10.4 Synthesis result

Here is my synthesis results - that includes number of LUTs used, number of IOs, buffers and so on.

```
# Info: Resource                          Used    Avail   Utilization
# Info: -----------------------------------------------------------------
# Info: IOs                               74      210      35.24%
# Info: Global Buffers                    1       32        3.12%
# Info: LUTs                              13      63400     0.02%
# Info: CLB Slices                        3       15850     0.02%
# Info: Dffs or Latches                   4       126800    0.00%
# Info: Block RAMs                        0       135       0.00%
# Info: DSP48E1s                          0       240       0.00%
# Info: -----------------------------------------------------------------
# Info: ***********************************************************
# Info: Library: work    Cell: flagupd    View: flag_arch
# Info: ***********************************************************
# Info:  Number of ports :                          74
# Info:  Number of nets :                          127
# Info:  Number of instances :                      75
# Info:  Number of references to this view :         0
# Info: Total accumulated area :
# Info:  Number of Dffs or Latches :                 4
# Info:  Number of LUTs :                           13
# Info:  Number of Primitive LUTs :                 14
# Info:  Number of LUTs with LUTNM/HLUTNM :          2
# Info:  Number of accumulated instances :          75
# Info: *****************************
```

4.11 My Types

This is the file uploaded on moodle and I have used it to enumerate the DP opcodes and the types of instructions.

4.12 Decoder

This is the file uploaded on moodle and I have used it for my instruction decoder.