Assignment-2 Stage-2 Submission

Indian Institute of Technology, Delhi

COL216: Computer Architecture

1 Introduction

The overall assignment is designing hardware for implementing a processor that can execute a subset of ARM instructions. This is the stage 2 of assignment. In this stage, I have built a simple processor in VHDL, that can execute some basic ARM instructions, that are - add, sub, cmp, mov, ldr, str, b, beq and bne.
(PS: I have submitted the asignment 1 day late. I used the late-day "credit" of previous assignment, as posted on Piazza.)

2 Stage 1

2.1 Objective

The objective of this stage is to design a single-cycle processor, that can execute a small subset of ARM instructions. The modules designed in 1st stage - ALU, Register File, Program Memory, Data Memory, were put together, with some combinational logic to form the simple processor.

During this, some new components were designed, which are as follows:

Program Counter with next address logic: This normally updates the Program Counter on every clock by adding 4. However, if the current instruction is a branch instruction, and the specified condition is true, it adds appropriate offset to the Program Counter.

Flags and associated circuit: On every clock it conditionally updates the Flags. We look at the s-bit of the instructions, and decide accordingly if we need to update the flags. It takes as input the inputs and outputs of the ALU to take decision.

Condition Checker: This is a combinational circuit that looks at the Flags and condition field of the instruction to decide whether the specified condition is true or not. Although only NE and EQ have been used at this stage, I have implemented all the conditions as given in lecture slides.

Instruction Decoder: This is a combinational circuit that looks at an instruction and identifies its classes and various subclasses. This design have been taken from Moodle, which was provided by professor himself.

2.2 Assumptions

I have used Edaplayground to write and test my code. EdaPlayGround has been used to Simulate the code using Aldec Riviera Pro tool. Quartus was downloaded and used to synthesize the design. Results of both of these have been attached.

For this design, I have assumed that the Program Counter is a separate register and has nothing to do with R15 of register file.

I have assume that Program Memory and Data Memory have independent address spaces. That is, both have byte addresses 0 to 255 (or word addresses 0 to 63).

For the purposes of this stage, I have assumed that the instructions are hardcoded in the program memory, and as soon as a X"00000000" instruction is encountered, the program halts.

All the executions are done on the rising edge of the clock, except the flags, that are set on the falling edge of the clock. This helps them to be available for the next instruction.

2.3 Logistics

The project directory has the following category of files:

1. **VHDL files:** There are a total of 13 VHDL files in this stage. These are:

a) TestBench.vhd : This file has the top level entity RAM_TB. The logic of this file is just a clock with a period of 1 ns, which is input to the processor,

b) Processor.vhd : This file is the crux of this assignment. It has the overall logic of a processor, It has 2 component instantiations i.e. a datapath and a controlpath. The data path has control signals as input and status signals as output, while the controlpath has control signals as output and status signals as input. The processor only takes a clock as an input and gives out no output.

c) Datapath.vhd : This file has the overall logic of datapath. It has many component instantiations, all of which receive control signals by the controller.

d) flags.vhd : This file contains the logic of resetting flags on each clock edge depending on the s-bit of the instruction.

e) program_counter.vhd : This file contains the PC register. It takes as input the "next" value of the program counter register, which is written in it, but only available in next clock edge. The initial value of data_in is set as 0 and that of data_out is -4, so that it is synced.

f) condition_checker.vhd : Contains the logic of updating the predicate value, p. This value along with the present state of flags is used to decide whether to branch or not.

g) Decoder.vhd : Given an instruction, it decodes the instruction based on its different fields.

h) MyTypes.vhd : This file contains the package of types that have been used in this project to easily identify different types and make the code more readable.

i) Controller.vhd : The overall control logic is in this file. On input of instruction and flag status, it returns control signals as an output to the datapath.

The other 4 files are the same as submmited in stage 1. No to minor changes were made in ALU, data_memory and register_file. Changes were made in program_memory.vhd file as the program had to be hardcoded.

2. **ARM files:** The directory also contains 3 .s files, that were used to test the processor (apart from the examples given in the problem statement). One file is fibonnaci.s, which contains fibonnaci logic. The file NewFibonnaci contains a modified Fibonnaci approach. This was implemented to extensively test mov and other DP instructions, The third file load_store_test contains a simple program that extensively tests the load and store instructions.

3. **Result files:** There are multiple PNG and PDF files that contain the results of simulation and synthesis.

2.4 Contributions

The MyTypes and Decoder file was taken from Moodle. No other help of any form was taken from anybody/anywhere.

2.5 Test Cases

The program was tested on many different test cases.

The test cases were designed to exhaustively cover all the cases. As explained above in the ARM files, these were the EPWaves corresponding to each instruction set.

Though these EPWaves were really difficult to analyse, I went through each instructions, and ensured that the value at each port was correct. As a final check, the value at register_file ports can be checked, which has been highlighted in some of the screenshots. The synthesis design are shown below:

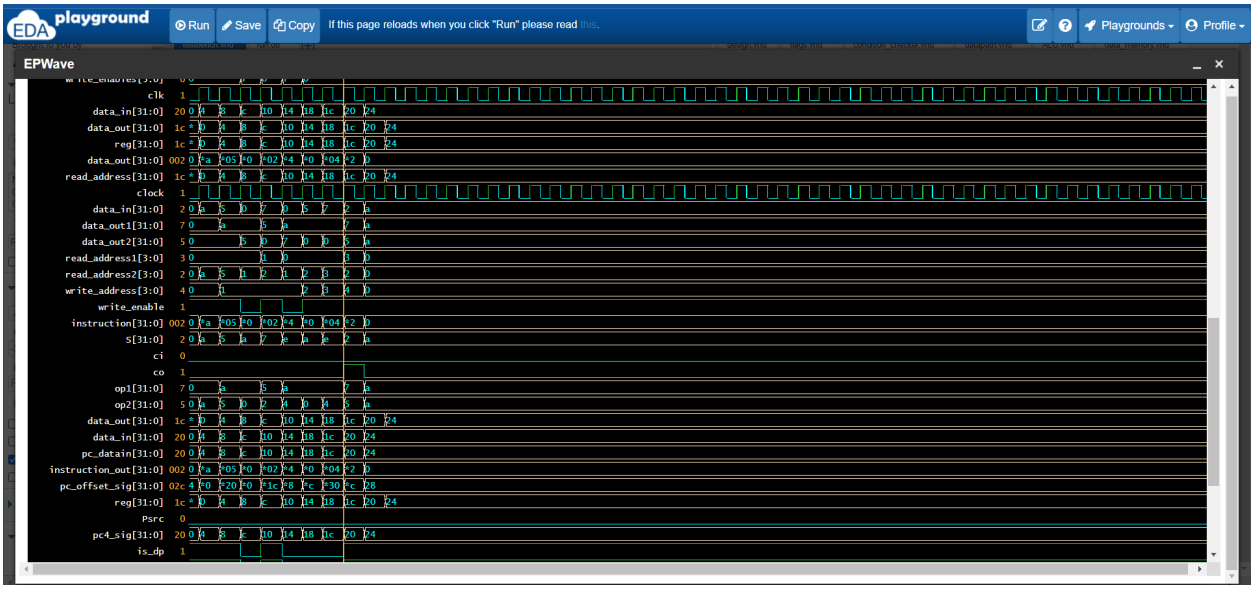


Figure 1: Problem Statement Test Case - 1

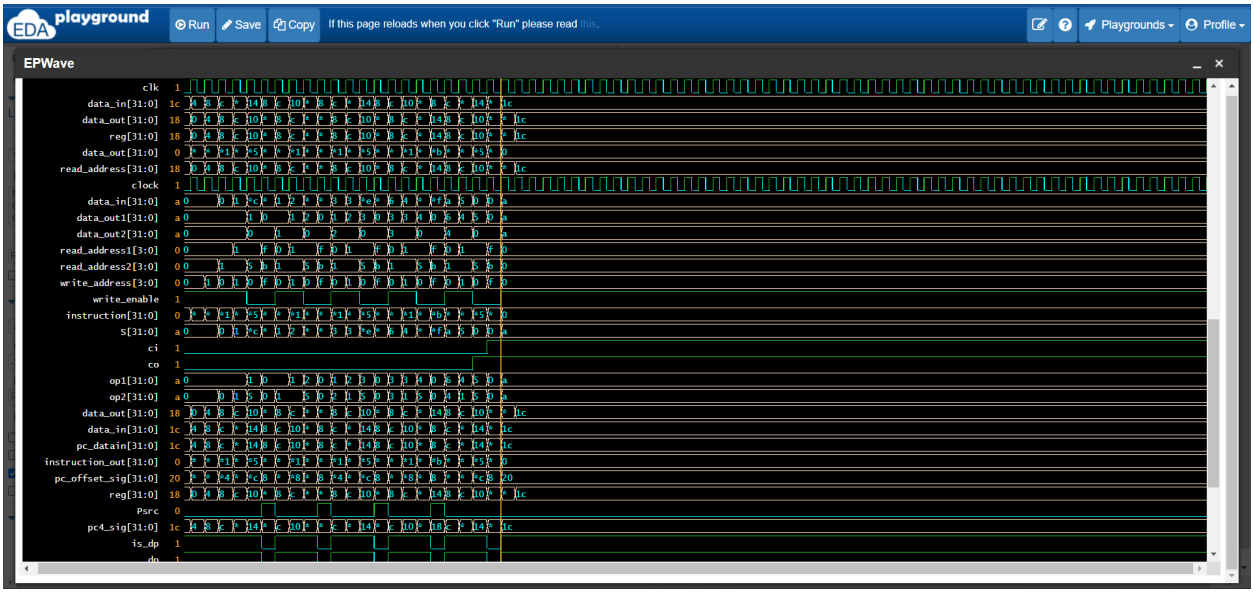


Figure 2: Problem Statement Test Case - 2

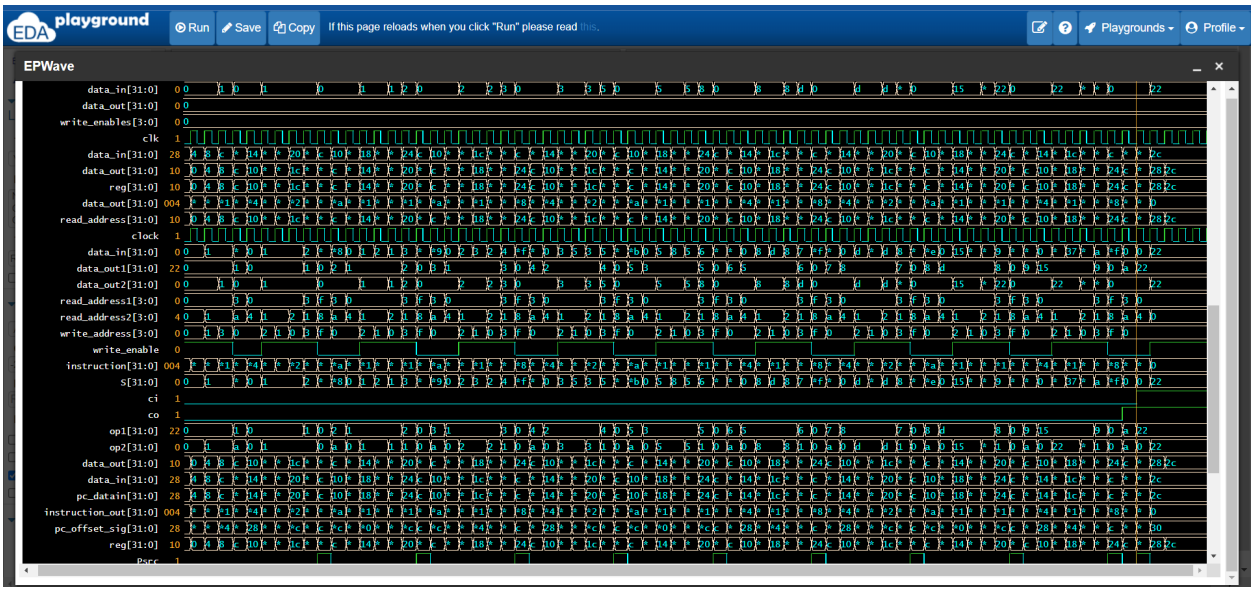


Figure 3: Simulation of Fibonacci sequence

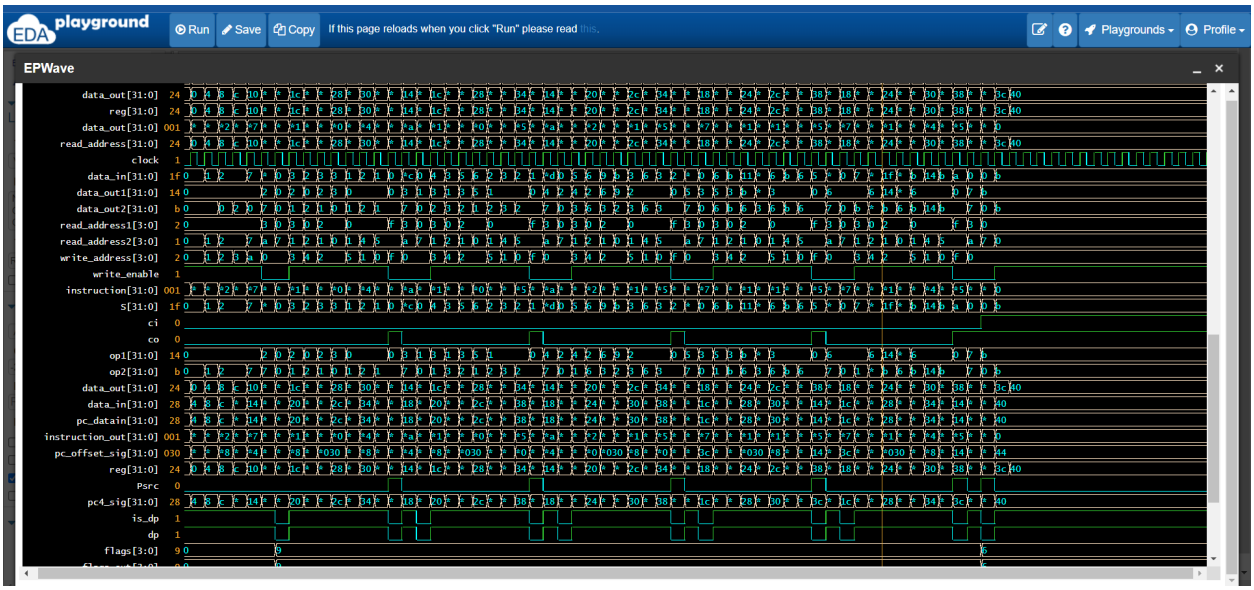
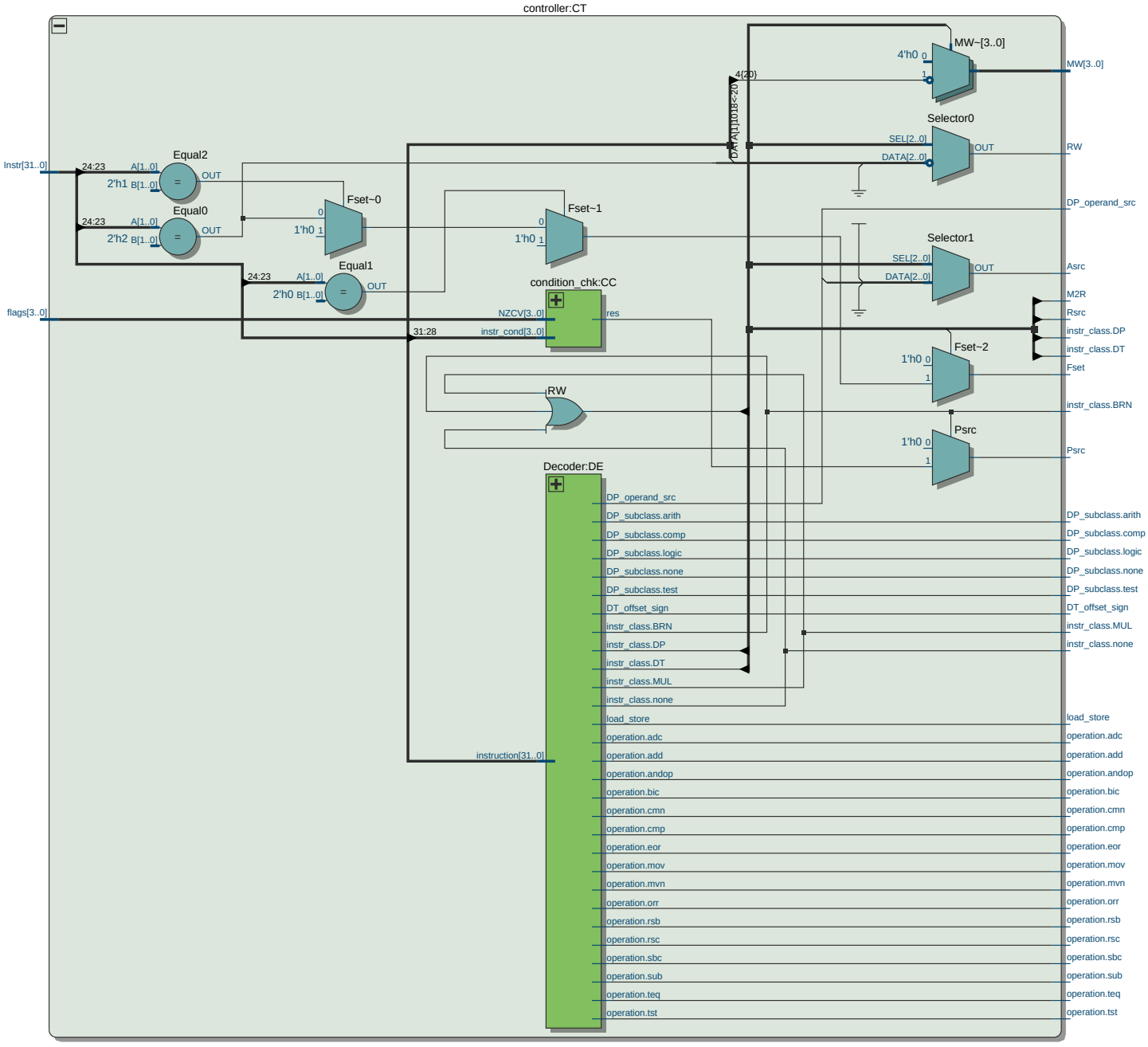
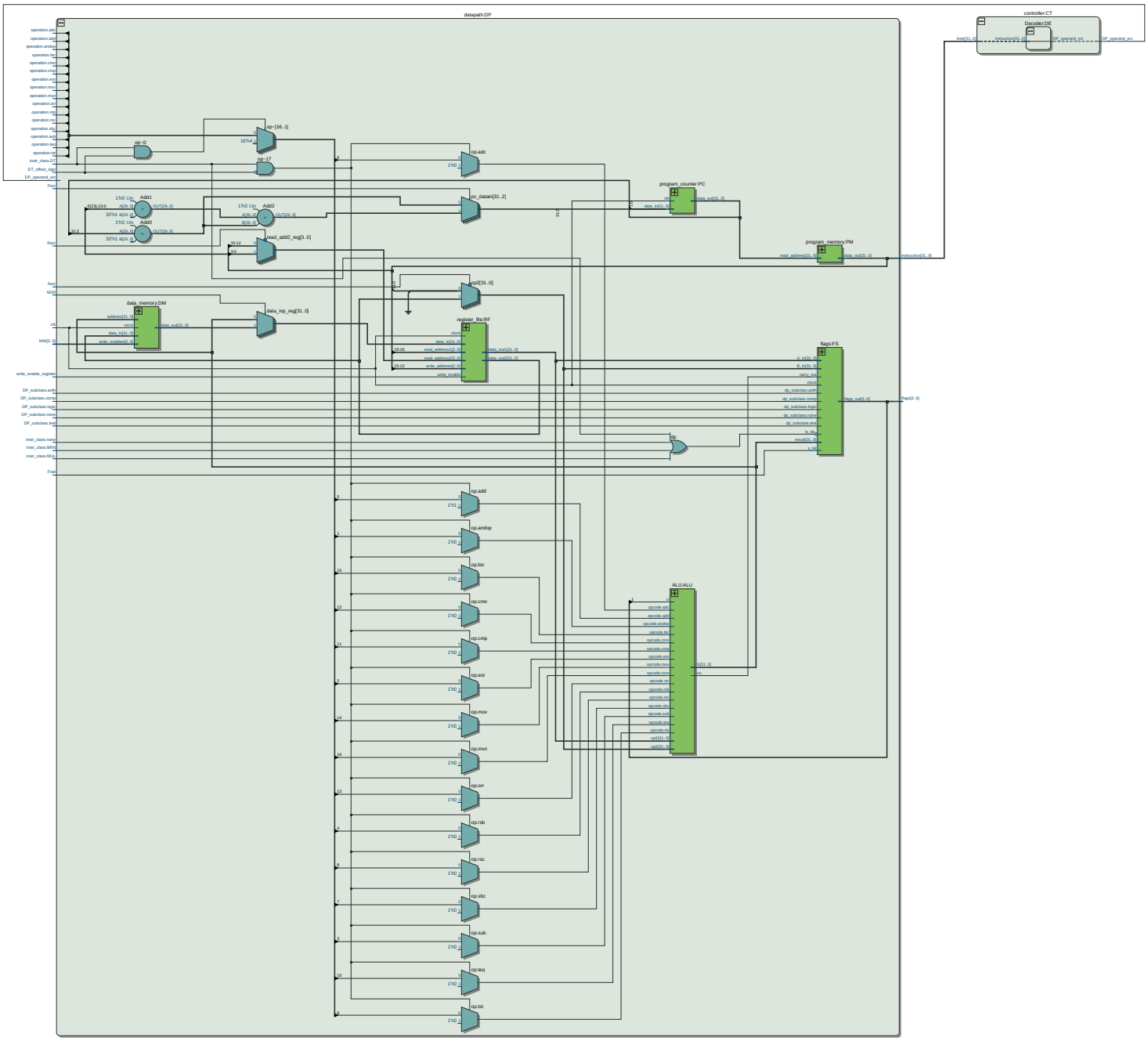
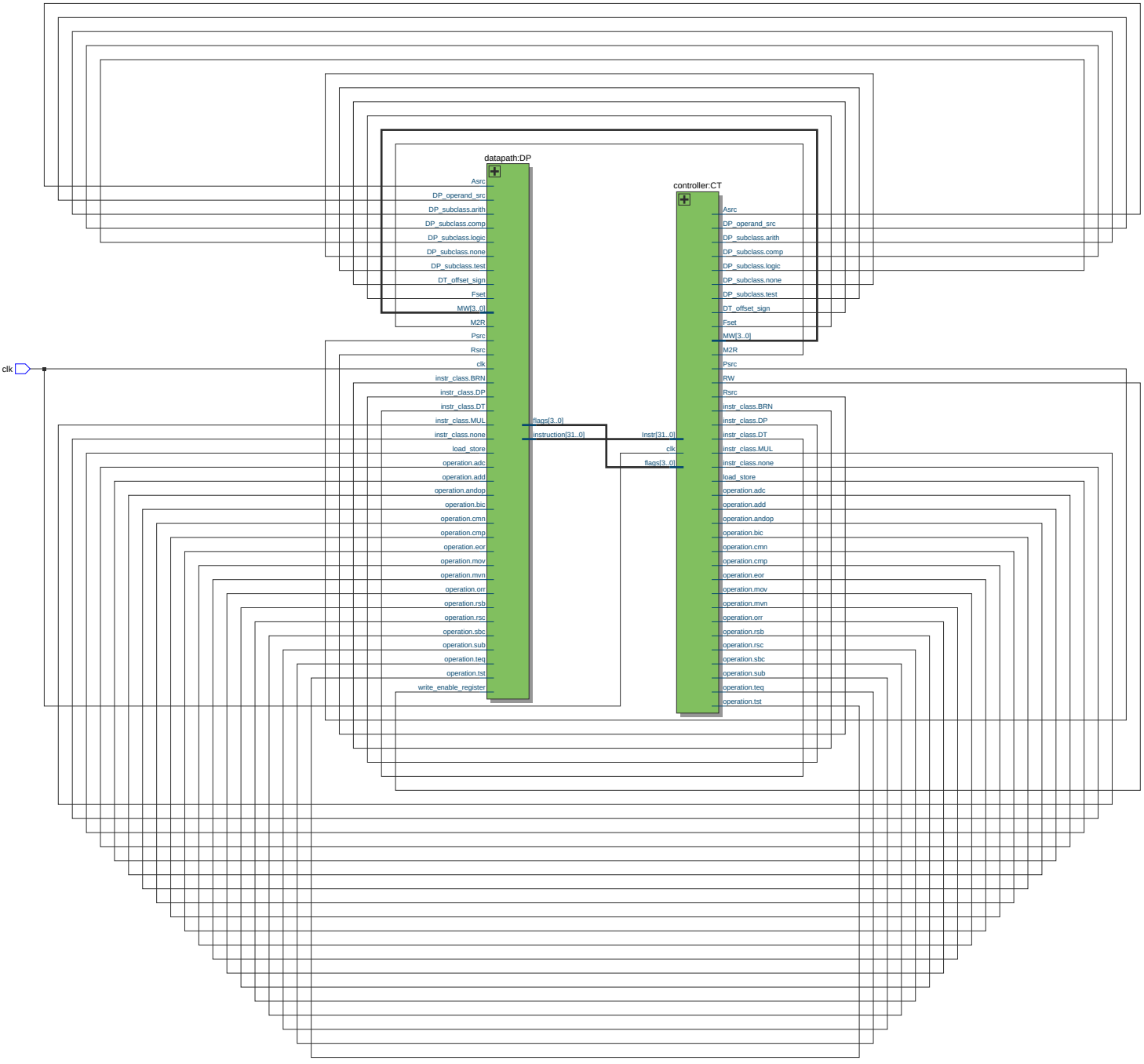


Figure 4: Simulation of New Fibonacci sequence

Figure 5: Simulation of Load-Store Instructions


controller:CT
MW~[3..0]
4'h0
MW[3..0]
4{20}
DATA[31..12,10..8,5..0]
Selector0
SEL[2..0]
DATA[2..0]
OUT
RW
DP_operand_src
Selector1
Asrc
M2R
Rsrc
instr_class.DP
instr_class.DT
Fset
Fset~2
1'h0
instr_class.BRN
Psrc
Instr[31..0]
Equal2
24:23
A[1..0]
2'h1
B[1..0]
Equal0
2'h2
Equal1
2'h0
Fset~0
Fset~1
condition_chk:CC
flags[3..0]
NZCV[3..0]
instr_cond[3..0]
res
31:28
RW
Decoder:DE
instruction[31..0]
DP_operand_src
DP_subclass.arith
DP_subclass.comp
DP_subclass.logic
DP_subclass.none
DP_subclass.test
DT_offset_sign
instr_class.BRN
instr_class.DP
instr_class.DT
instr_class.MUL
instr_class.none
load_store
operation.adc
operation.add
operation.andop
operation.bic
operation.cmn
operation.cmp
operation.eor
operation.mov
operation.mvn
operation.orr
operation.rsb
operation.rsc
operation.sbc
operation.sub
operation.teq
operation.tst


datapath:DP
controller:CT
Decoder:DE
program_counter:PC
program_memory:PM
data_memory:DM
register_file:RF
flags:FS
ALU:ALU
Add1
Add2
Add0
op:adc
op:add
op:andop
op:bic
op:cmn
op:cmp
op:eor
op:mov
op:mvn
op:orr
op:rsb
op:rsc
op:sbc
op:sub
op:teq
op:tst


datapath:DP
Asrc
DP_operand_src
DP_subclass.arith
DP_subclass.comp
DP_subclass.logic
DP_subclass.none
DP_subclass.test
DT_offset_sign
Fset
MW[3..0]
M2R
Psrc
Rsrc
clk
instr_class.BRN
instr_class.DP
instr_class.DT
instr_class.MUL
instr_class.none
load_store
operation.adc
operation.add
operation.andop
operation.bic
operation.cmn
operation.cmp
operation.eor
operation.mov
operation.mvn
operation.orr
operation.rsb
operation.rsc
operation.sbc
operation.sub
operation.teq
operation.tst
write_enable_register
flags[3..0]
instruction[31..0]
controller:CT
Instr[31..0]
clk
flags[3..0]
Asrc
DP_operand_src
DP_subclass.arith
DP_subclass.comp
DP_subclass.logic
DP_subclass.none
DP_subclass.test
DT_offset_sign
Fset
MW[3..0]
M2R
Psrc
RW
Rsrc
instr_class.BRN
instr_class.DP
instr_class.DT
instr_class.MUL
instr_class.none
load_store
operation.adc
operation.add
operation.andop
operation.bic
operation.cmn
operation.cmp
operation.eor
operation.mov
operation.mvn
operation.orr
operation.rsb
operation.rsc
operation.sbc
operation.sub
operation.teq
operation.tst
clk