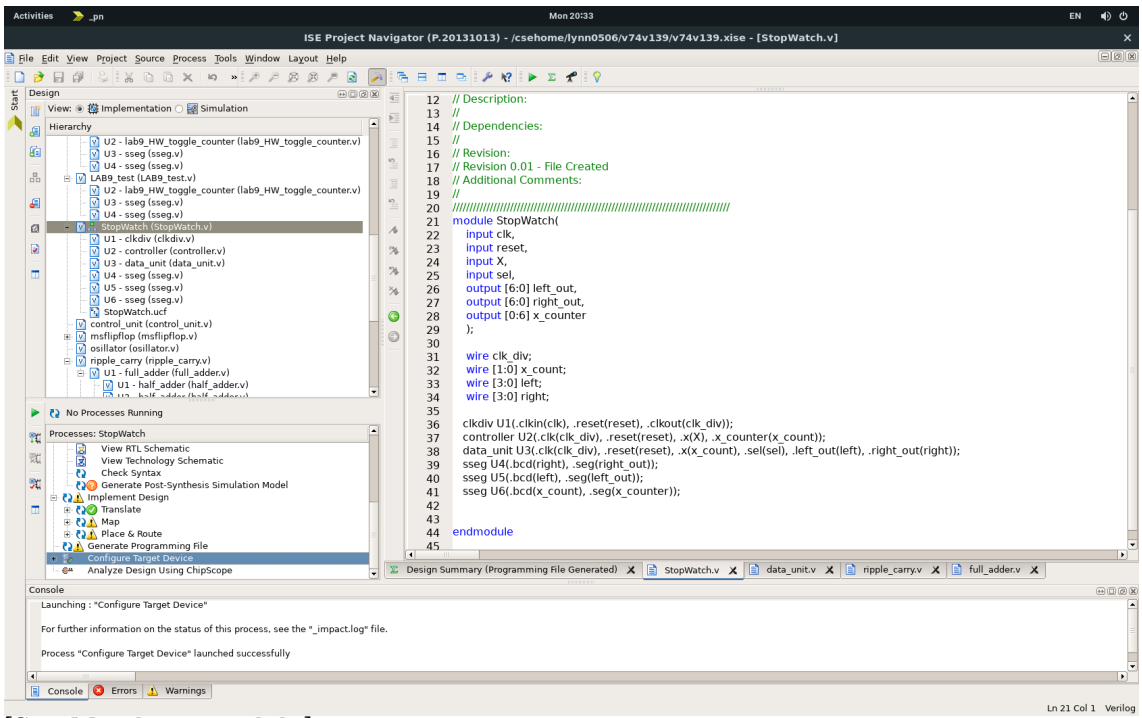
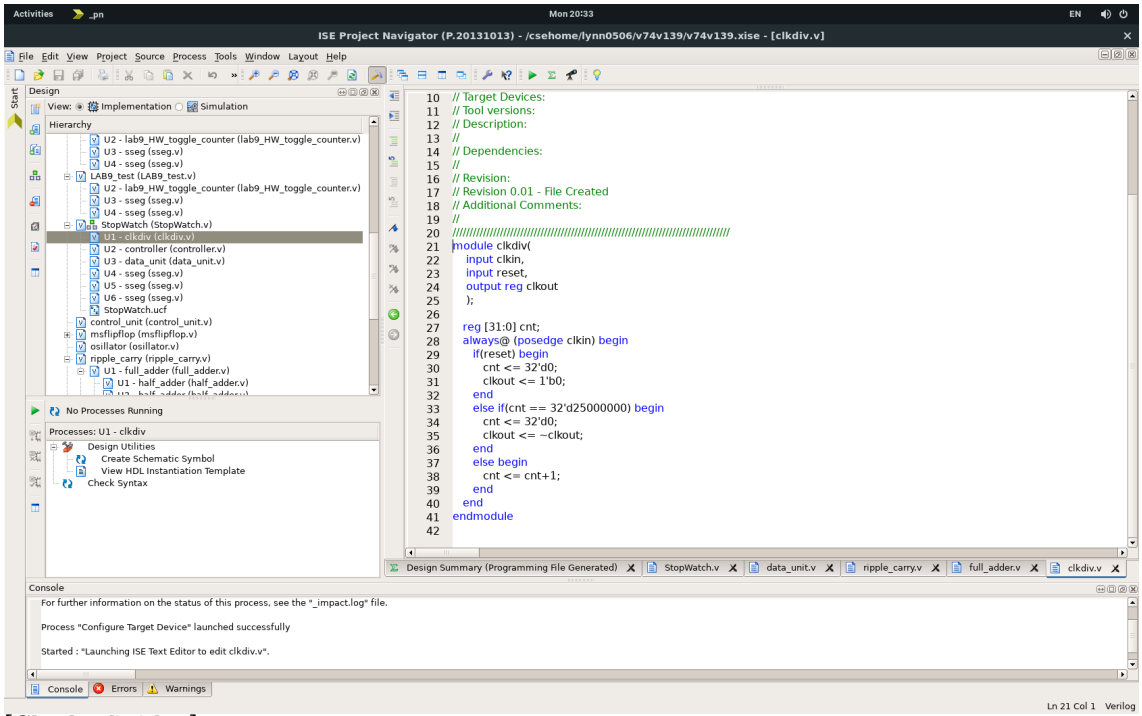
2014-19498 정은주 독어교육과

논리설계 lab10(Report and Homework)

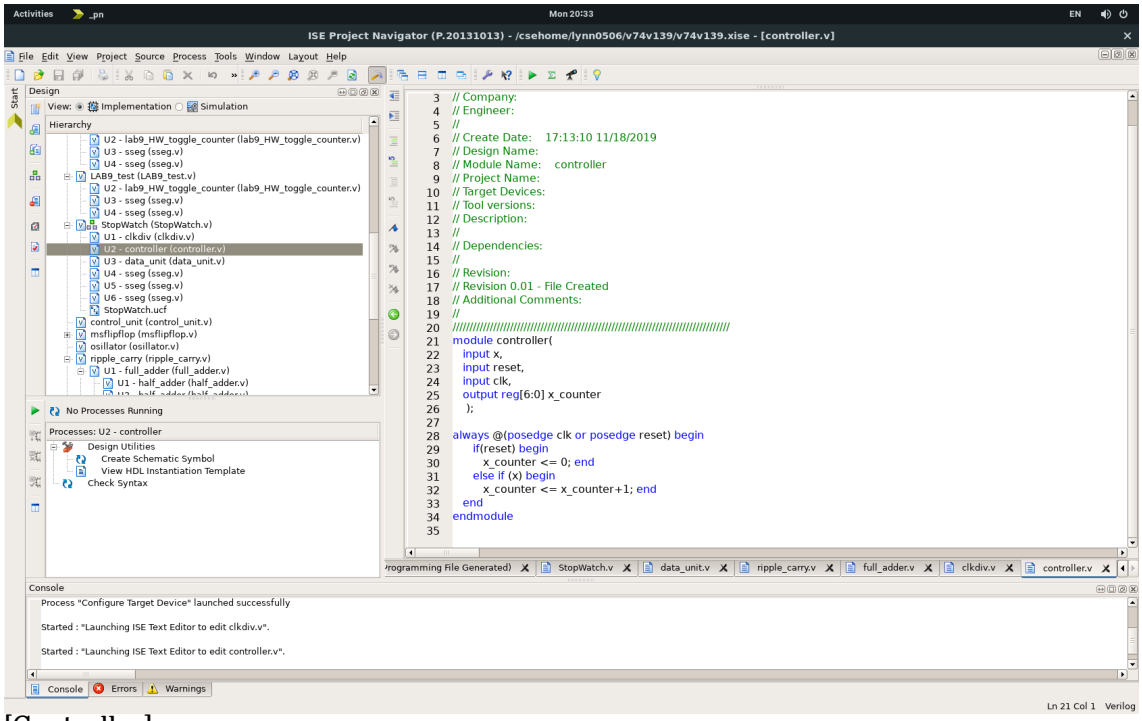


[StopWatch_topmodule]

Controller, clock divider, data_unit과 7-segment로 각각 module을 나누어서 작성하였고, 본 StopWatch module은 이를 아우르는 top module로 동작한다. 편의상 x_count를 하는 횟수를 보드 위에서 볼 수 있도록 x_counter 역시 따로 output으로 구현하였다.

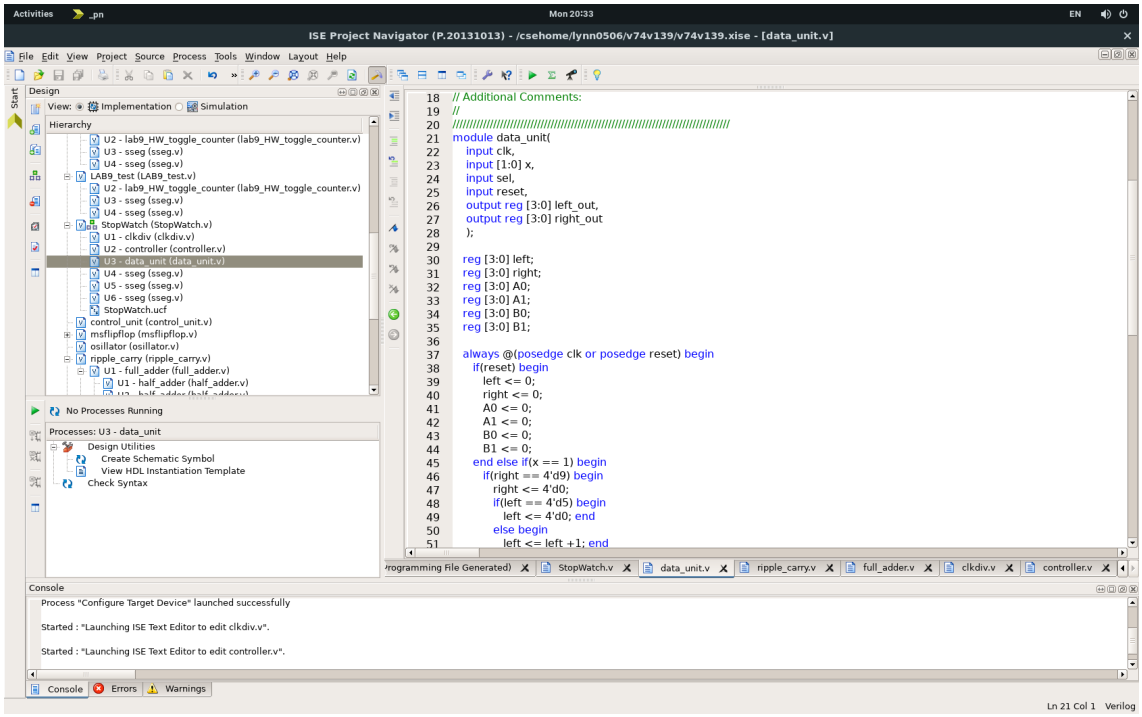


[Clock_divider]

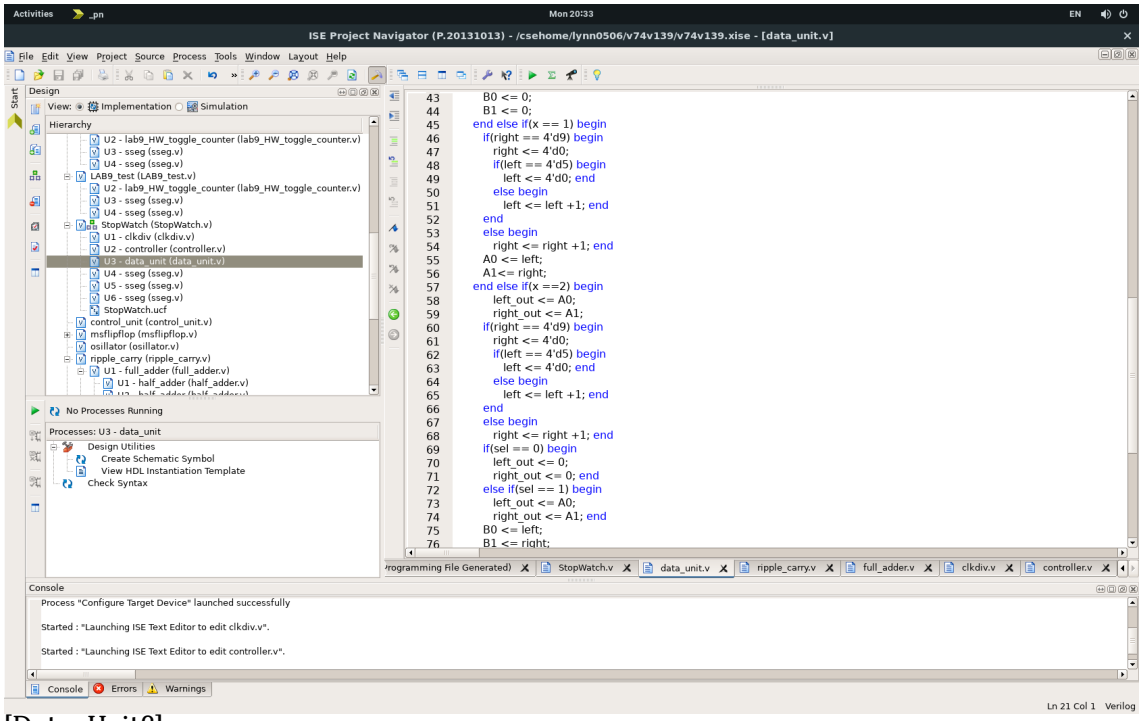


[Controller]

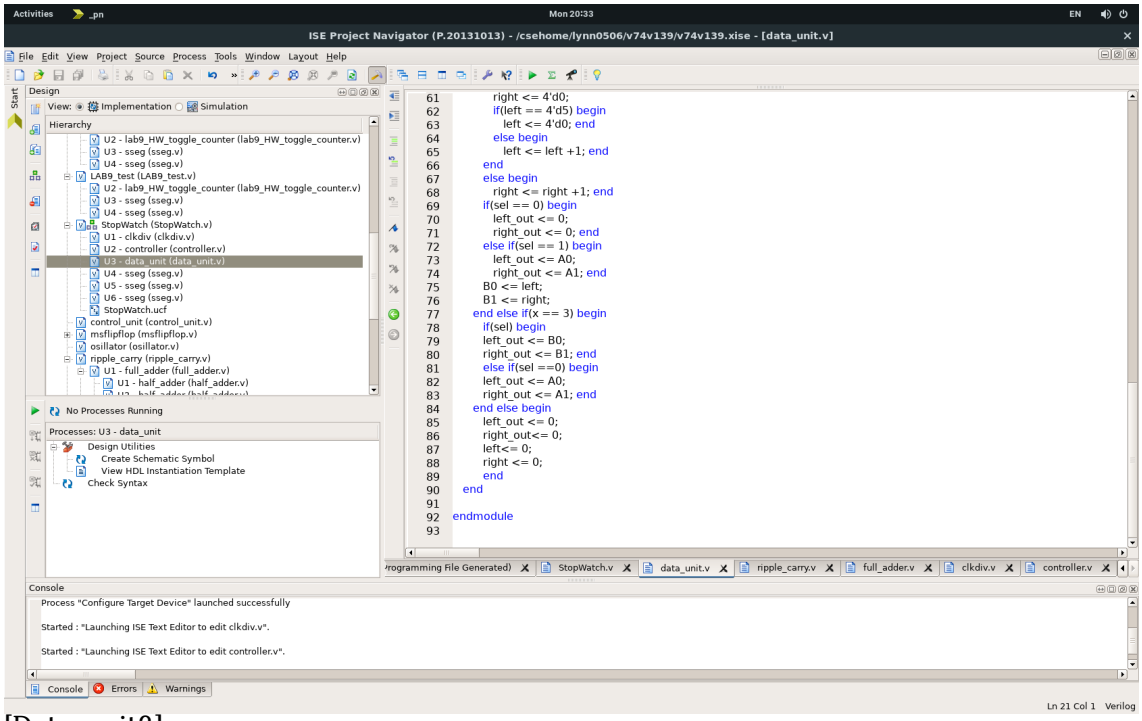
본 코드는 X_counter로 작동하며, X input이 눌릴 때마다, count 된다. 혹은 reset 버튼이 눌리게 된다면, X_counter 역시 0으로 reset된다.



[Data_Unit]

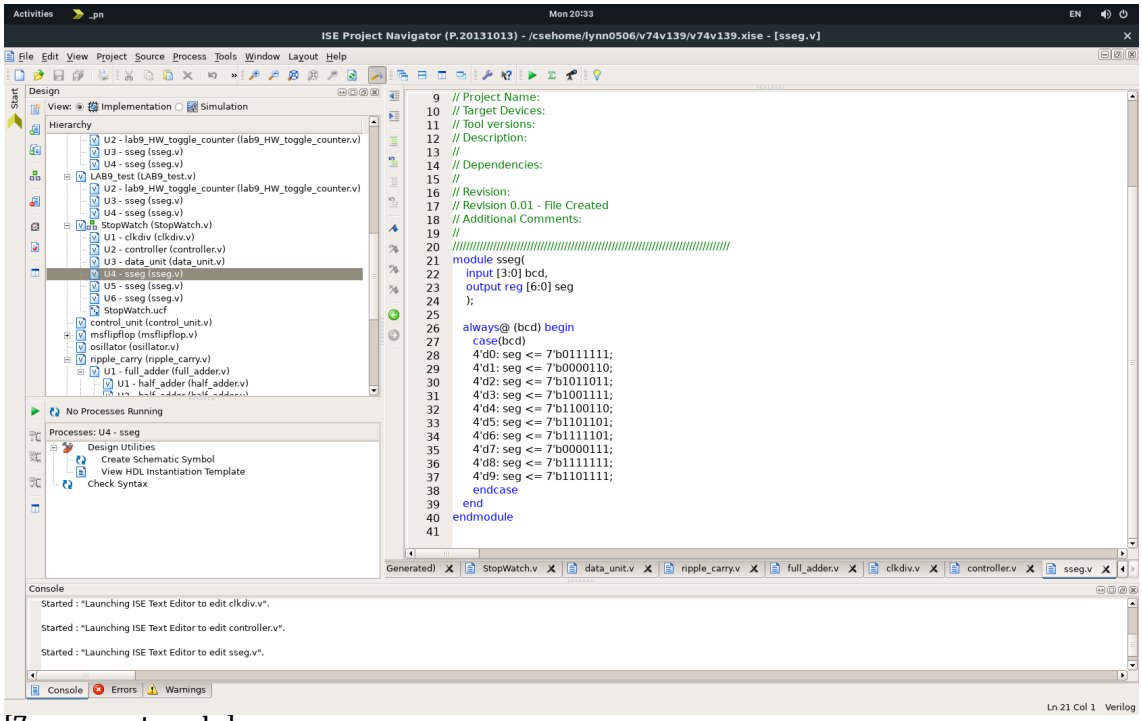


[Data_Unit2]

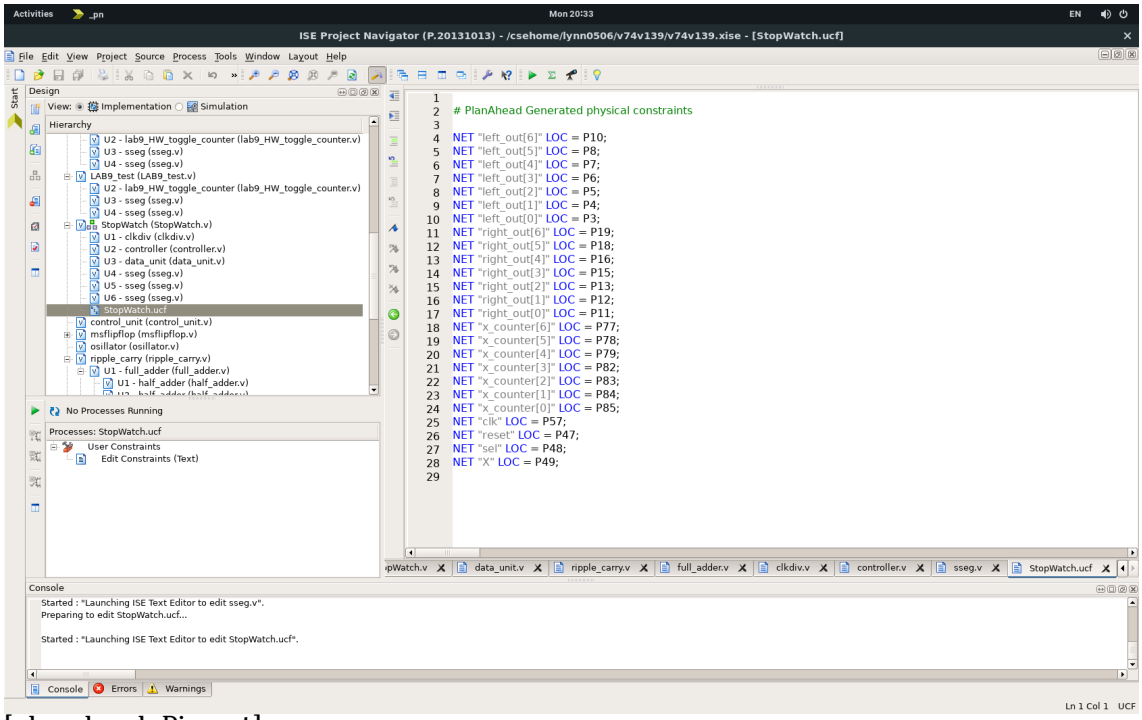


[Data_unit3]

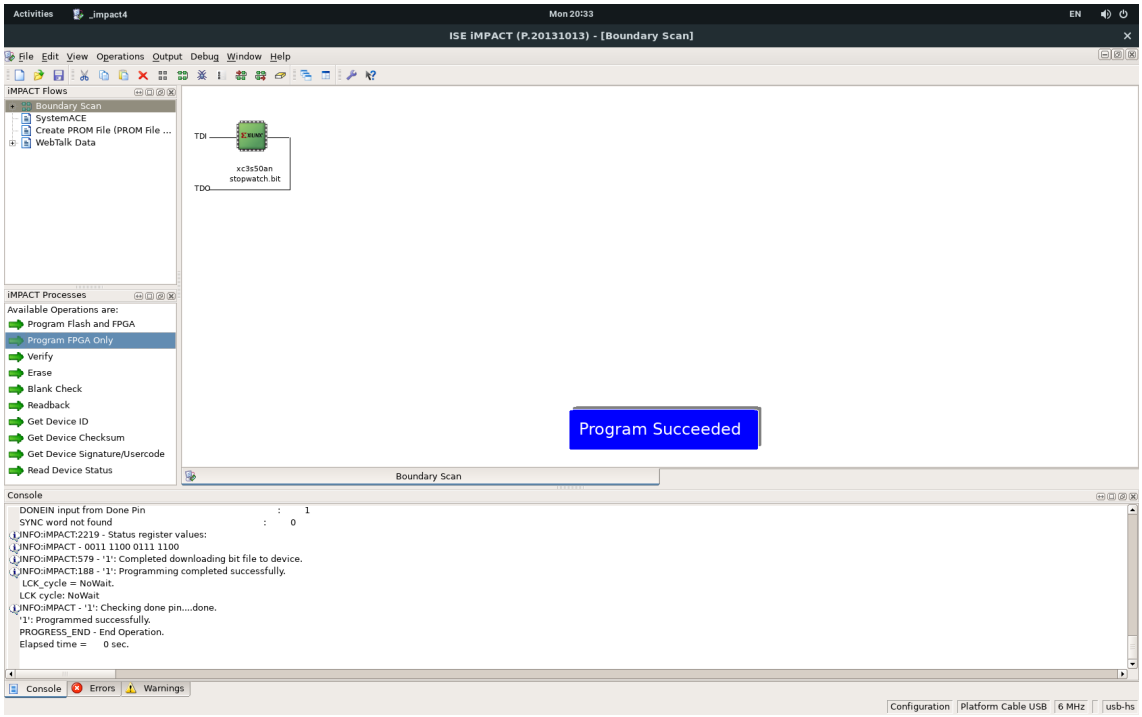
A0, A1,(X:1) B0, B1(X:2), counter인, left와 right를 설정하였고, select signal에 의해서 input이 0이면, previous value (A0, A1 - X 가 1일 때, B0, B1 - X가 2일 때)를 출력할 수 있도록 하였다. left와 right는 계속해서 count하도록 하였고, 만약 X가 2일 때 select input이 0이면, 이전 Value인 0이, select input이 1이면, X가 2가 되는 순간이었던 값이 출력된다. X가 3일 때, select input이 눌리기 전에는 X가 2가 되는 순간이었던 값이 출력되고 있고, 이 때, select input이 1이면, X가 3이 되는 순간이었던 값이 출력되게 된다.

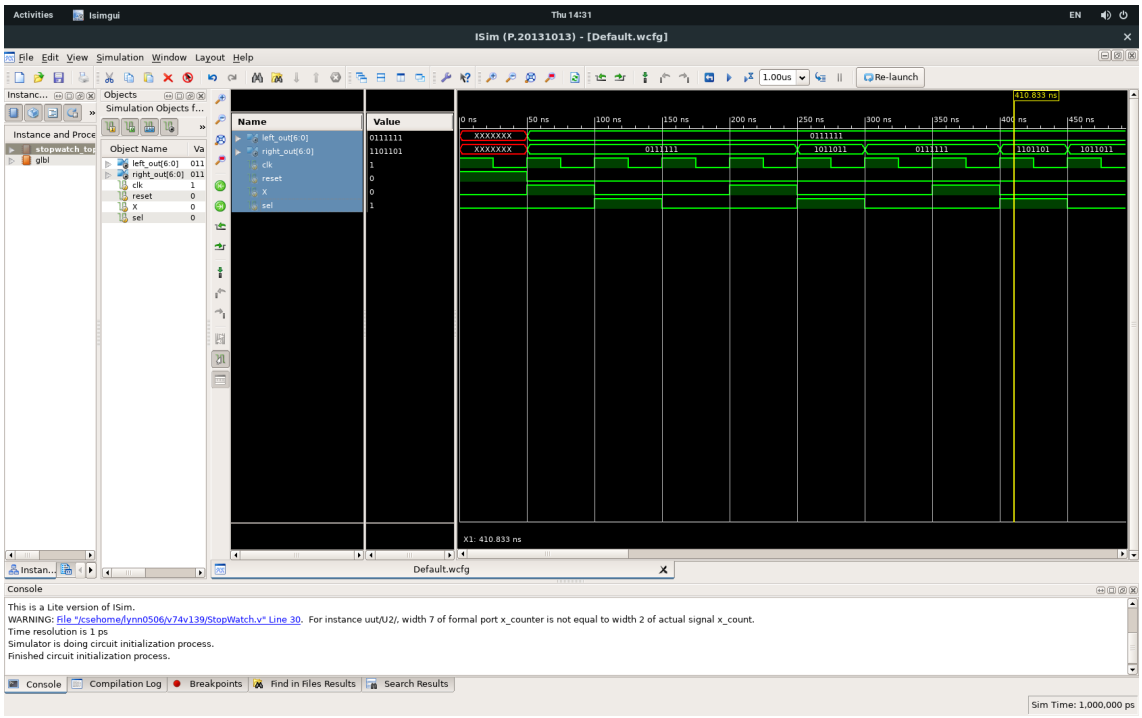


[7_segment_code]

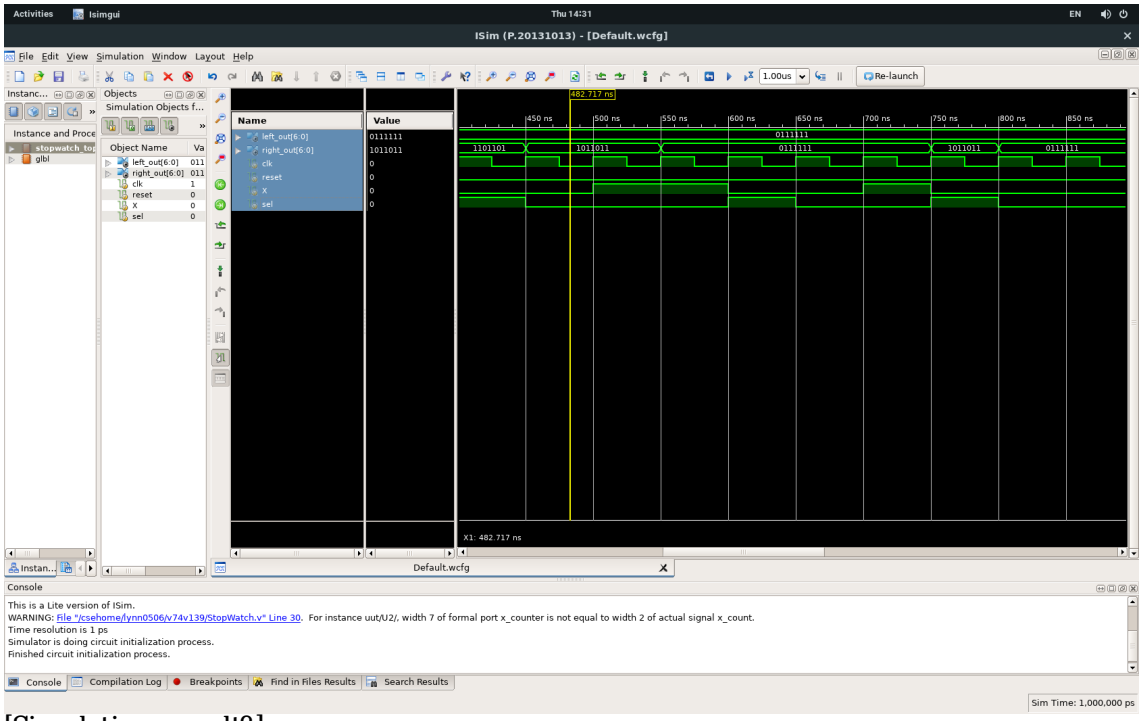


[planahead_Pin_set]





[Simulation Result]



[Simulation_result2]

2.Explain the architecture of a microprecessor

1) Describe the 4 type of operation and how they work

operation의 종류는 많다. 기본 R-type 연산에 해당하는 add sub 뿐만이 아니라, arithmetic and logical shift right left 비교 연산, 그리고 beq, bne, beg blt 등등 비교 연산과 jump and link, unconditional jump 등의 branch 연산이 있다. 이중에 microprocessor 구현을 위해 사용되는 네 개의 연산인 add, store, load, jump만을 설명한다.

- add: add는 기본적으로 r-type Instruction이며, rs1 과 rs2의 값을 더해서 rd 즉 destination register에 기록한다.
- store: store는 data memory에 접근하는 Instruction이며, rs1의 값과 immediate를 sign extend 한 후에 그것에 해당하는 주소 값에 rs2 register의 data 값을 저장한다. 즉 memory에 access한 후 값을 write한다.
- load: load 역시 data memory에 접근하는 Instruction이며, rs1의 값과 sign extend 된 값을 더한 주소 값에 위치한 값을 다시 rs2 값에 load 즉, register에 값을 쓰는 여산으로 memory를 read한다.
- branch(jump): branch는 conditional jump와 unconditional jump가 있는데, beq, bne, blt, 등등 연산의 결과에 의해 해당 pc로 jump를 할 수도 있고, 한 번에 해당 register의 data 값으로 pc를 옮기는 연산도 가능하다.

2. Draw a block diagram of each sub-module and explain the operation of each sub-module (block diagram 사진 마지막 첨부)

Instruction memory: Instruction memory는 instruction을 Fetch하는 단계에서 pc를 이용해 Instruction memory에 접근해 관련 Instruction을 읽어내기 위해 사용된다. instruction memory의 경우는 읽기 기능만 가능하며, 쓰기 기능은 제한되어 있다.

Data memory: Data memory는 256 8-bit data로, 일정 정보를 담고 있다. load, store 등의 s-type 연산의 경우만 접근 가능하며, 이를 위해 mem_write, mem_read등의 시그널을 사용해 메모리에 접근해 읽고 쓰는 기능을 진행한다. 그리고 load instruction의 경우에는 읽어낸 값을 register에 다시 쓰는 과정이 필요하므로, reg write signal을 이용하여 pipeline register를 따라 저장되어 있는 써야하는 register 정보와 함께 Register file에 접근해 값을 쓰게 된다. store의 경우에는 단순히 메모리에 접근해 관련 정보를 쓴다.

Register file: Register file은 register들을 담고 있는데, Instruction decode 단계에서 사용되는 부분으로, 이 곳에서는 register를 rs1과 rs2의 data를 읽고, output으로 내보내게 된다. 더불어 write back 단계에서 온 write data와 write reg 정보를 이용해 관련 register에 정보를 적는다.

ALU: arithmetic logic unit으로 add 연산이 진행되는 곳이다. Instruction의 rs1, rs2의 data 등으로 계산이 진행된다. overflow를 고려하지 않는다.

Control unit: 제어하는 control signal을 생성해 내는 부분을 의미한다. 보통 Instruction Decode 단계에 위치하고 있으며, register momory, data momory, ALU 등을 위한 control signal(write mem, read mem, register write, opcode, alu_src, branch, 등등)을 만들어낸다.

sign extension: sign bit의 값이 extend되는 것으로, 만약 two's complement의 수로 표현되어 있다면 음수인 경우는 sign bit가 1이기 때문에, 1로 남아있는 비트들을 채워서 111....10..이런 패턴을 만들 것이고, 양수라면 0의 값이 남은 비트를 채우게 된다.

PC: program counter로 현재 주소 값의 다음 주소 값의 정보를 나타내준다.

3.Describe the data paths for the 4 instructions and explain why the control signals are needed

Data path에는 Register, ALU, Mux, Instruction memory, Register file, 그리고 Data memory 등이 있다. 각각 Instruction은 32bit로 각각 rs1, rs2, rd, opcode, function code, 등등 여러 정보를 담고 있다. Fetch, Decode, Execution, Memory, Write back 등의 5단계를 거치게 되는데, 이때, 각각 Instruction 별로 작동하는 동작이 다르고, 각각 부분 요소 정보들이 다르다.

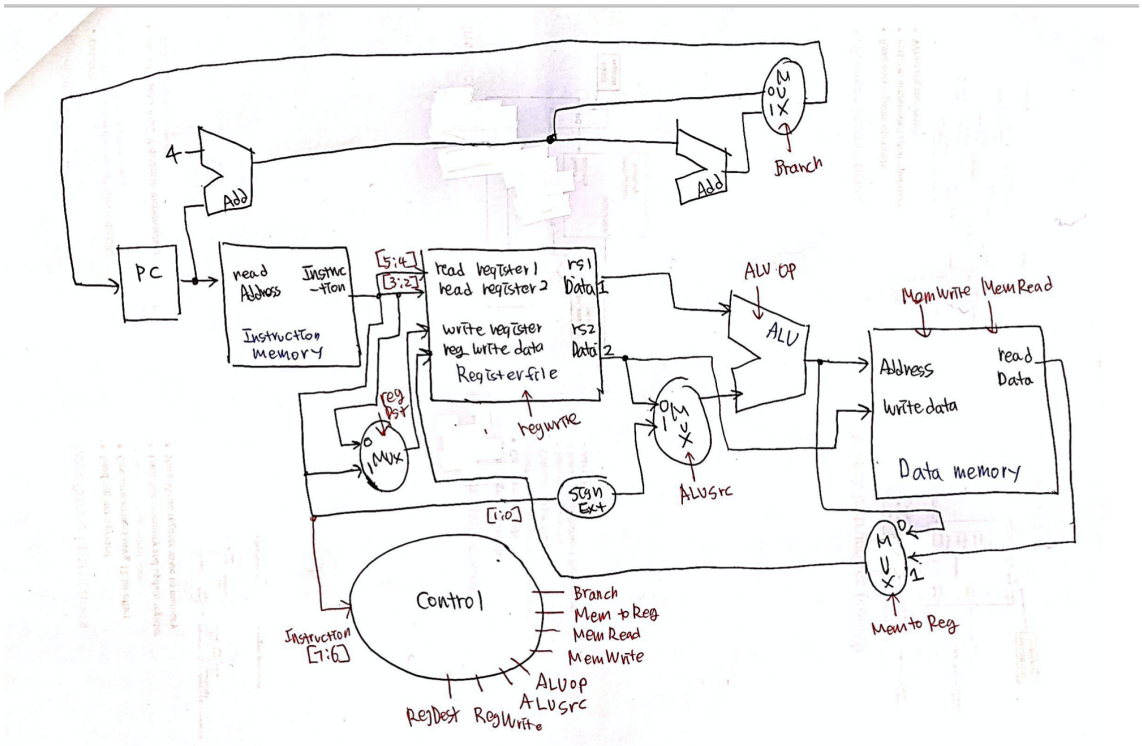
add: add instruction은 Instruction memory -> Register file -> (Mux) -> ALU -> Register file 등의 과정이 요구된다. instruction memory단계에서 fetch 되어 register file에서 해당 rs1, rs2, 그리고 rd 정보를 decode하고, ALU에서 연산이 진행된 후에 write back 단계에서 다시 register에 값이 쓰이게 된다.

store: store instruction은 Instruction memory -> Register file -> ALU -> data Memory 의 단계를 거치게 되는데, ALU에서 rs1 + immediate 의 sum 결과 값이 address 값이 되고, 그 결과값을 통해 data memory에 접근한다. 그리고 rs2의 값을 write 하며 연산이 끝난다.

load: load instruction은 Instruction memory -> Register file -> ALU -> data Memory -> Register file 단계를 거치게 되는데, 위의 store instruction과 비슷한 data path를 거치나 store 연산과 다르게 ALU 결과값에 해당하는 값으로 data memory 주소에 접근 후에 그 값을 읽어내서 rs2에 해당하는 register에 그 값을 쓰게 되는 과정이 추가된다.

why control signals needed?

위의 언급된 모든 과정들은 각각의 data path에 어떤 연산 혹은 어떤 행동을 취할지를 알려주는 controller가 필요하다. 예를 들어 add 연산의 경우는 data memory에 접근할 수 없도록 cotrol signal인 write mem 혹은 read mem을 0으로 설정해주어야 하고, 반대로 s-type 연산인 store 혹은 load의 경우에는 이 시그널들을 1로 set 해주어야 한다. 그리고 이러한 signal들을 만드는 Control Unit이 이 역할을 하게 된다. 이외의 ALU data path를 위한 ALU op, mux를 위한 signal들 역시 어떤 연산을 할 것인지에 대한 정보를 담는다.



[Microprocessor_block_diagram]

4. Suggest the additional implementation to upgrade your microprecessor

1) insturction을 추가할 수 있다. 먼저 제시된 add, load, store, jump 이외에도, beq, bne, blt, bgt 혹은 sub, mul, div 와 같은 연산도 가능할 수 있고, 더불어 jump의 경우에도 unconditional jump 혹은 conditional jump 등으로 나누어 pc controller를 추가할 수 있다.

2) pipelining 사실 각 fetch, decode, execution, memory, writeback 단계 사이 사이에는 pipeline register가 존재하기 때문에 이를 통해서 forwarding이나 stall, 그리고 nop(bubble)기능을 하도록 설계할 수 있을 것이다. 이를 통해 clock cycle 수를 줄일 수 있게 된다.

3) tactile swtich를 통해 여러 조절을 가능하도록 하는 방법이 있다. reset, execution, input 등 program의 mode를 select한다던가 혹은 data memory에 접근하는 혹은 register에 write back 하는 값들을 seven segment를 통해서 보여주는 방법도 있다. 혹은 LED의 색을 통해서 각각 어떤 단계를 거치고 있는지를 보여주는 방법도 있다.