

**과 목 : 디지털 시스템**

**과 제 명 : HW2 DigitalClock**

**담당교수 : 민 형 복**

**학 과 : 소프트웨어학과**

**학 년 : 3**

**학 번 : 2016312761**

**이 름 : 여혁수**

**제 출 일 : 2020-11-01**

1. **Purpose**

이번 실습에서 설계한 것은 디지털 시계에서 시간 모드, 날짜 모드, 타이머 모드, 알람 모드를 모드 선택 스위치를 누름에 따라 모드를 결정하고 또한 시간이나 날짜 등의 값을 1씩 증가시키는 스위치 기능도 하는 회로이다.

1. **Problem statement**

**1) Describe what is the problem.**

이번 실습에서 구현할 것은 크게 둘로 나눌 수 있다. 하나는 시계의 모드를 결정하는 mode generator이다. 구현할 시계에서 모드를 결정하는 스위치는 총 2개인데, 하나는 시간 모드에서 날짜 모드로 가듯이 모드를 아예 변경해주는 스위치, 다른 하나는 어떤 한 모드 안에서 시간을 수정하는 것과 같은 세부기능이 있을 때 그것을 변경해주는 스위치이다. 세부기능이 3개가 있다면 스위치를 누를 때마다 변경이 되고 마지막 3번째 기능에서 스위치를 누르면 첫번째 기능으로 다시 돌아오는 방식으로 기능이 변경된다. 모드도 마찬가지로 변경이 된다. 구현할 다른 하나는 시간 값을 증가시키며 조작하는 스위치인 increase generator이다. 스위치를 누르면 그때 결정된 모드가 시간 값을 증가시키는 모드라면 증가를 시켜주고, 아니라면 아무것도 하지 않는 스위치이다.

이 회로는 모드 스위치 2개를 동시에 누른 경우 모드는 현재 모드를 유지해야하는 조건이 있고, 또한 시간 값 증가 스위치를 눌렀을 때 바로 증가 값을 주는 것이 아니라 모드 변경이 있다면 변경을 한 후에 증가가 이루어져야 한다.

**2) Describe how do you solve the problem.**

이 회로를 구현할 소스 코드는 Verilog 언어로 작성하고, modelsim 프로그램[1]으로 회로를 시뮬레이션한다. 정상적으로 시뮬레이션이 된다면 quartus prime 프로그램으로 synthesize를 마친다. 우선 수업을 통해 배운 synthesizable Verilog code를 참고[2]하여 Verilog 코드를 짜고, comment 및 indentation도 알맞게 작성한다. 자세한 조건사항들은 과제 설명 pdf[3]에서 참고하였다. 특히 모드를 결정하는 기능에서 쓰이는 핵심 문법인 case 구문에서 모든 경우에 값을 설정해줘야 정상적인 synthesize가 된다는 것을 생각하며 구현한다. 코드 구현 후 시뮬레이션을 하는데, 처음엔 시간 값을 증가시키는 스위치도 모드 결정 스위치와 마찬가지로 rising edge에서 업데이트를 해주었는데 에러가 나서 모드 변경 후 증가 값은 falling edge에서 따로 업데이트를 해주어야 함을 알아차렸다. 시뮬레이션을 위한 테스트벤치 파일[4]은 과제 자료에서 받아 활용하였다. 이렇게 시뮬레이션을 마치고 synthesize를 하며 synthesize가 되지 않는 원인을 에러 프린트를 통해 분석했고, multi-driver error를 인터넷 자료[5]를 참고하여 해결하였다.

1. **Sources & Results**

**1) Analysis of HDL Source Codes of Design**

우선 clk의 rising edge에서 동작하는 always block을 만들었다. Rising edge에서 모드의 변경이 이루어지기 때문에 rising edge일 때의 모드 변경 스위치 2개의 값을 확인한다.

if (sw2 === 1'b1) begin

// both sw1, sw2 is 1, mode should be held.

if (sw1 === 1'b0) begin

if (mode1 === M1\_TIME) begin

case (mode2)

M2\_TIME\_G : mode2 <= M2\_TIME\_HOUR;

M2\_TIME\_HOUR : mode2 <= M2\_TIME\_MIN;

M2\_TIME\_MIN : mode2 <= M2\_TIME\_SEC;

M2\_TIME\_SEC : mode2 <= M2\_TIME\_G;

Endcase

이 부분은 mode2를 변경하는 부분인데 mode2 스위치를 의미하는 인풋인 sw2가 1이 들어오면 mode2를 변경한다. 이 때 mode1 스위치도 동시에 눌렸다면 그냥 현재 모드를 유지해야 하므로 mode1 스위치가 눌리지 않았을 때만 고려한다. Case 구문을 통해 현재 모드가 무엇이냐에 따라 다음 순서의 모드로 변경해준다. Mode1이 무엇이냐에 따라 mode2가 다르므로 mode1도 조건문으로 확인을 해야한다.

이런 식으로 모드 변경 스위치 2개의 기능을 모두 구현할 수 있다. Asynchronous reset도 같은 always block에 구현되어 있는데 negedge일 때 mode1,2를 초기 모드로 변경해 주었다.

또한 시간 관련 값을 증가시키는 경우는 falling edge에 업데이트를 해야하기 때문에 다른 always block을 만들었다. 증가시킬지에 대한 여부는 rising edge에서 판단을 한다.

if (set === 1'b1) begin

update\_inc <= 1'b1;

end

else begin

update\_inc <= 1'b0;

end

이 코드는 rising edge에서 동작하는 always block에 있는 부분인데 증가시켜야한다, 아니다를 판단해 하나의 변수에 넣어줌으로서 저장만 해주는 것이다.

if (update\_inc === 1'b1) begin

if (mode1 === M1\_TIME) begin

case (mode2)

M2\_TIME\_G : increase <= 1'b0;

M2\_TIME\_HOUR : increase <= 1'b1;

M2\_TIME\_MIN : increase <= 1'b1;

M2\_TIME\_SEC : increase <= 1'b1;

Endcase

이 부분이 falling edge에서 동작하는데 아까 rising edge에서 설정해준 변수 값을 보고 증가시켜야 한다면 현재 모드를 확인하고 가능한 모드라면 증가를 의미하는 아웃풋 ‘increase’에 1을 주는 것이다.

만약 ‘increase’에 1을 줬다면 이 값이 다음 rising edge에서도 유효하면 안될 것이다. 따라서 0으로 reset을 해주는 것이 필요하다.

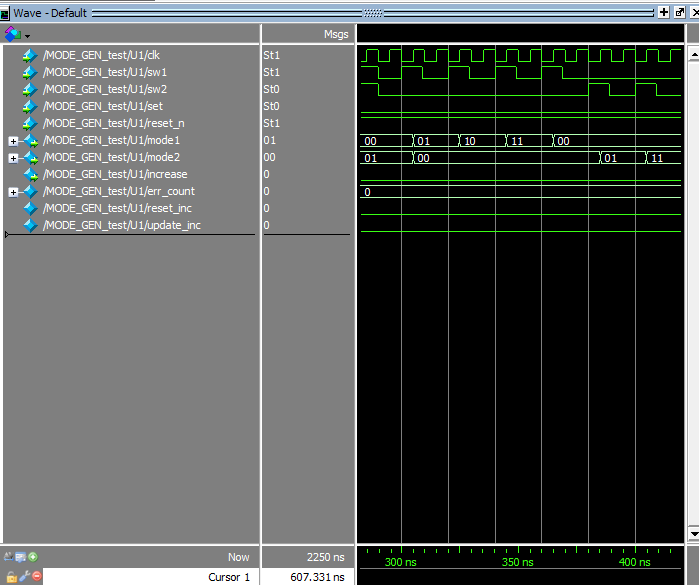
if (increase === 1'b1) begin

reset\_inc <= 1'b1;

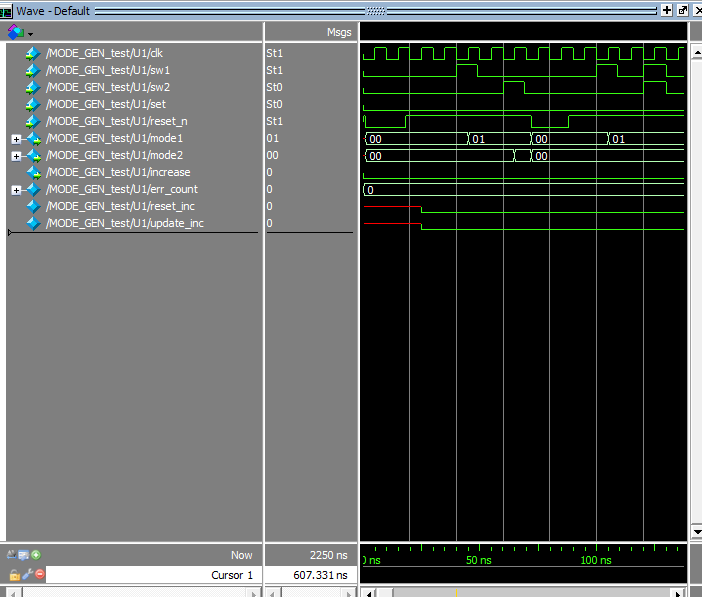
end

rising edge에서 ‘increase’의 값을 봤는데 이미 1이다? 그러면 이렇게 리셋을 해줘야 한다고 새로운 변수를 지정하여 판단만 해주는 것이다. 그리고 falling edge에서 리셋을 의미하는 변수에 1이 들어왔다면 리셋을 해주면 된다.

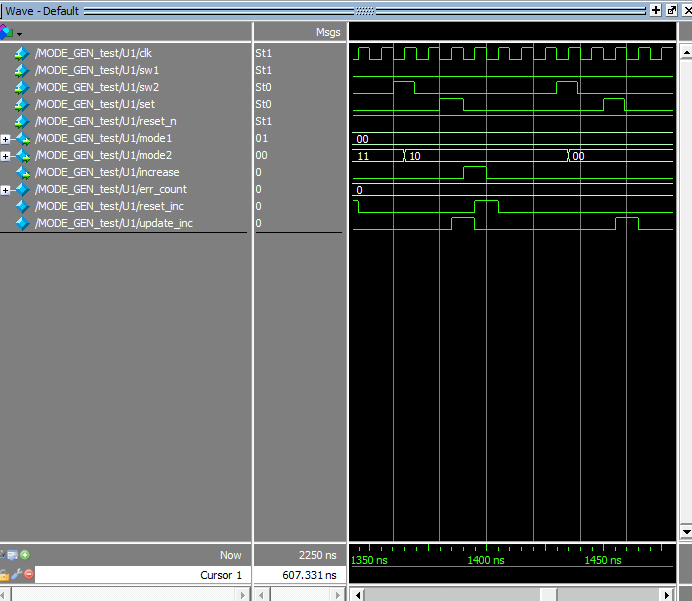
**2) Simulation Waveforms**



Mode1과 mode2가 변경되는 부분이다. Sw1, sw2는 스위치 비트로 setup time과 hold time을 가져 clk의 rising edge에 1이 돼있음을 보장하고 있는 것을 볼 수 있고, clk의 rising edge와 같은 타이밍에 mode1, 2의 값이 바뀌는 것을 볼 수 있다.

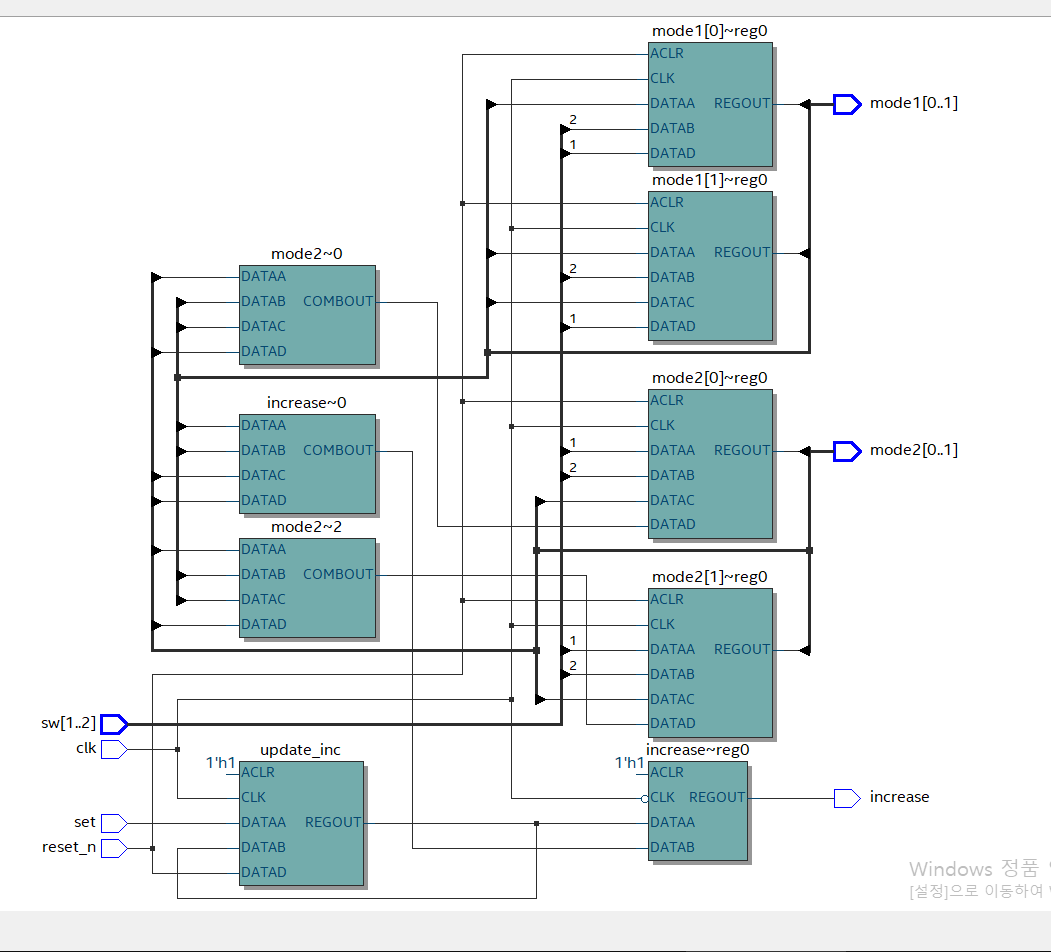


Reset\_n이 asynchronous reset을 담당하는데 falling edge일 때 mode1,2이 2’b00 으로 초기화되는 것을 볼 수 있다.

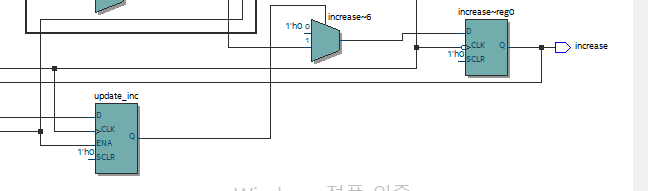
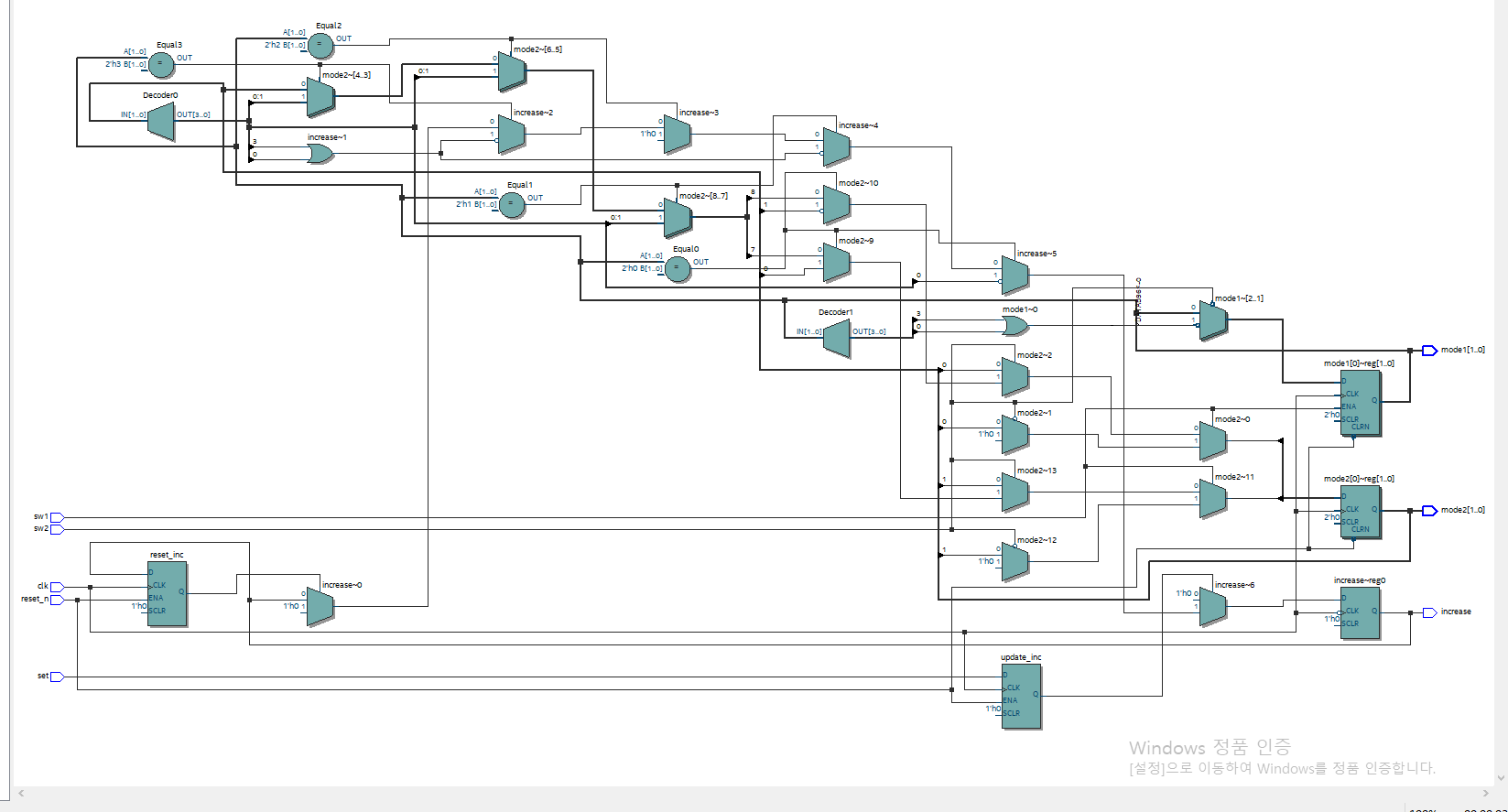


시간 관련 값을 1 증가시키라는 의미의 set비트에 1이 들어오면 rising edge에서 update\_inc에 1을 줌으로서 증가시켜야 한다는 판단만 해주고, 다음 falling edge에서 increase값이 업데이트가 되는 것을 볼 수 있다. 또한 increase가 1이 된 후에 다음 rising edge에서는 increase가 이미 1이기 때문에 reset\_inc에 1을 줌으로서 리셋을 해야함을 표시해준다. 이로 인해 다음 falling edge에선 increase 값이 초기화되는 것을 볼 수 있다.

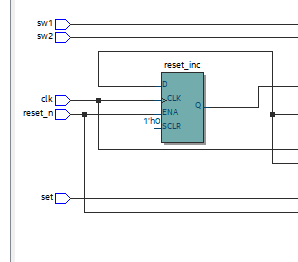
**4) Synthesis Results**



Quartus Prime[6]에서 synthesis한 후 Technology map viewer로 본 netlist의 결과이다. Increase를 다루는 레지스터에 clk 인풋에는 not gate가 달려 있고, update\_inc의 아웃풋을 인풋 데이터로 사용하고 있는 것으로 보아 잘 구현된 것 같다. Mode1, 2를 결정하는 부분도 mode1과 mode2간에 서로를 확인하면서 결정이 되는 것 같다.



Mode generator에서 set을 통해 update\_inc를 판단한 것과 Increase generator를 다룬 코드가 이 레지스터들에 해당된다.

이 레지스터가 asynchronous reset 코드에 해당한다.

그리고 나머지 파란 원에 해당하는 레지스터들이 mode generator에서 mode를 결정하는 코드에 해당한다. 수많은 멀티플렉서가 모드 2개를 결정하는 데 필요하다는 것이 의아하고 신기했다.

1. **Conclusion**

Synthesizable code를 만드는 것은 많이 해보면 익숙해져서 빠르게 잘 할지도 모르겠지만 코딩 능력보다 노력이 중요한 것 같다. Synthesizable code가 되려면 지켜야할 규칙에 대해서 많이 배우는 계기가 되었다. Mode의 변경과 increase의 변경이 다른 타이밍에 이루어지는 것을 구현하는 것이 좀 어려웠는데, 나는 임시 저장 변수를 만들어서 타이밍을 다르게 만들었지만 이것보다 깔끔한 방법이 있을 것 같아서 완벽히 스스로 만족스러운 과제 결과물은 아니었다.

1. **참고 문헌**

## [1] Modelsim. (FPGA), Intel. [Accessed Oct. 9, 2020]. Available: <https://fpgasoftware.intel.com/20.1/?edition=lite>

## [2] 민형복, “SynthesizableVerilog.pdf,” [Online]. Available: <http://class.icc.skku.ac.kr/~min/di/> . [Accessed Oct. 27, 2020].

## [3] 민형복, “DigitalClock2.pdf,” [Online]. Available: <http://class.icc.skku.ac.kr/~min/di/> . [Accessed Oct. 27, 2020].

## [4] mode\_gen\_tb.v, 민형복. Accessed: Oct. 27, 2020]. Avaliable: <http://class.icc.skku.ac.kr/~min/di/>.

## [5] bogus919, “multi-driver error”, [Online]. Available: <https://bogus919.tistory.com/entry/multidriver-error> . [Accessed Oct. 30, 2020].

## [6] Intel Quartus Prime. (20.1), Intel. [Accessed Oct. 30, 2020]. Available: <https://fpgasoftware.intel.com/20.1/?edition=lite>