**2018학년도 1학기**

**Embedded Hardware 과제#1**

**- LED Mover 보고서 -**

|  |  |
| --- | --- |
| 과목 | 임베디드 하드웨어 |
| 담당교수 | 박능수 교수님 |
| 학과 | 컴퓨터공학과 |
| 학번 | 201411317 |
| 이름 | 조민규 |
| 제출일 | 20180510 |

목차

[1. Project Overview 3](#_Toc513628085)

[2. Project Description 8](#_Toc513628086)

[3. Demostration 12](#_Toc513628087)

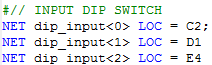
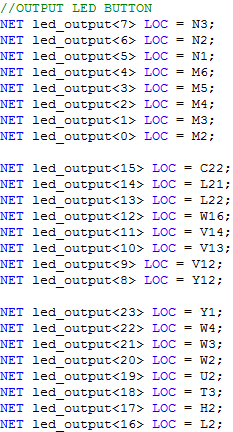
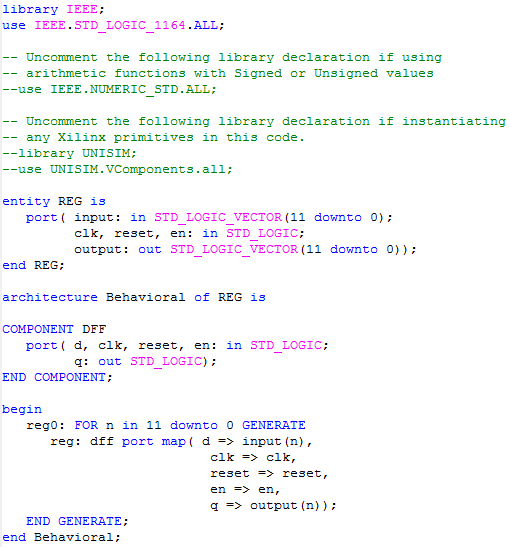
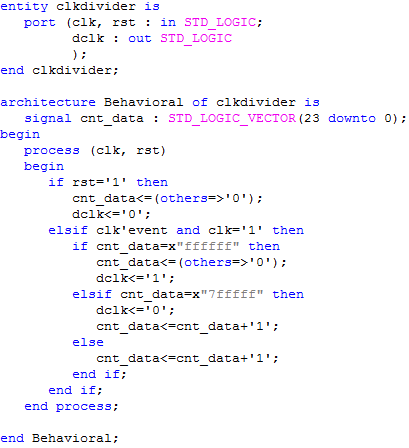
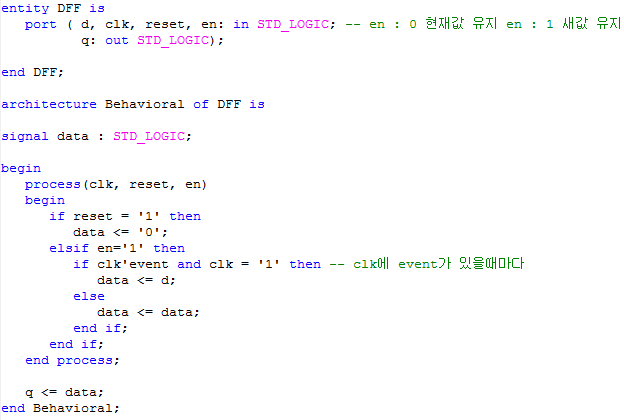
[4. ETC 13](#_Toc513628088)

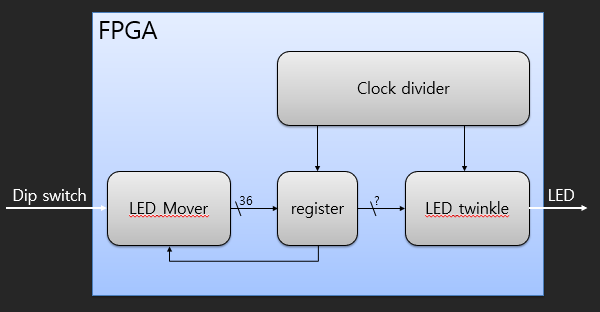
|  |  |  |
| --- | --- | --- |
|  |  | **Project Introduction** |

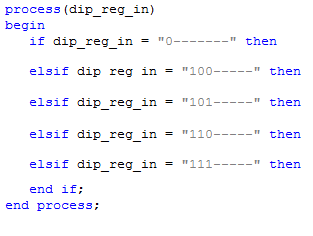
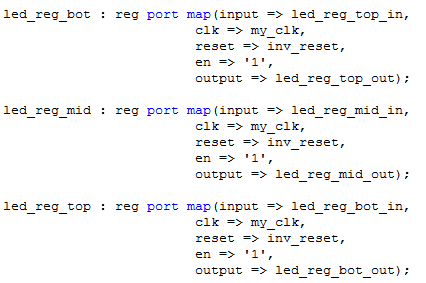
|  |
| --- |
| 1. Project Overview |

◦ 프로젝트 과제: LED Mover

◦ 프로젝트 구상:

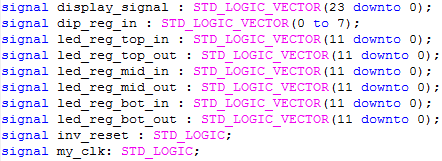
* Input: DipSwitch
  + STD\_LOGIC\_VECTOR (0 to 7)로 구현한다.
  + UFC 파일에서 Input으로 설정해준다.
  + 0 to 2만 사용하므로 3 to 7은 설정하지 않는다.
* Output: LED
  + STD\_LOGIC\_VECTOR (23 downto 0)로 구현한다.
  + UFC 파일에서 Output으로 설정해준다.
* Component: Register \* 3
  + STD\_LOGIC\_VECTOR (11 downto 0)로 구현한다.
  + DFF를 이용하여 구현한다.
  + 3칸의 LED를 각각 Top, Mid, Bot으로 나누어 관리한다.
* Component: Clock Divider
  + Clock Divider => STD\_LOGIC
  + 특정 시간마다 1로 바뀌는 dclk를 보내도록 구현한다.
* Component: DFF (Dip Flip Flop)
  + 1 Bit를 저장하는 Component
  + N Bit를 저장하는 Register를 구현하기 위해 사용한다.

◦ 프로젝트 구조 및 설계

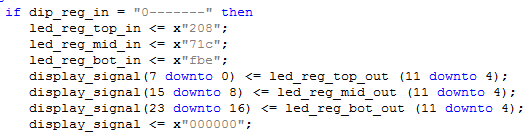
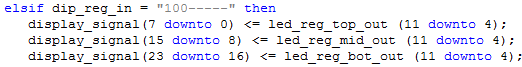
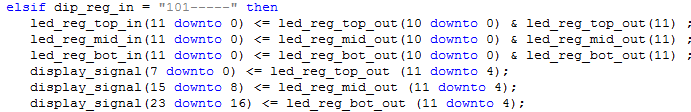
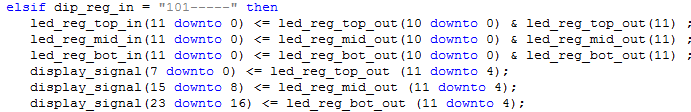
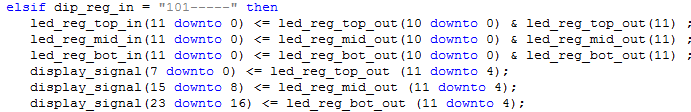
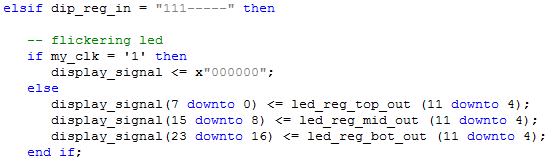
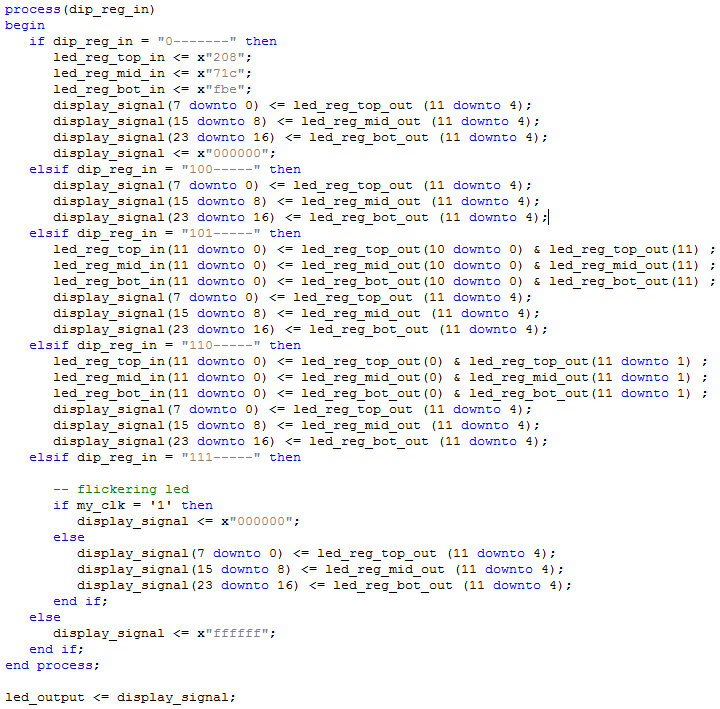
* LED\_Mover에서 LED\_twinkle을 결정하는 것은 DipSwitch이다.  
  그러므로 DipSwitch의 상태에 따라 작동을 다르게 하기 위해서 아래와 같이 if문으로 분기를 해주어야 한다.
* DipSwitch 3개를 111로 바꾸었을 때에는 현재 상태를 저장한 후에 Flickering을 해주어야 한다. 그래서 12 Bit를 저장하는 Register를 3개 선언해준다. 12 Bit를 저장하는 이유는 2개의 삼각형을 만들기 위해서 가로 12Bit 세로 3Bit가 필요하기 때문이다.
* HardWare의 clk를 사용하면 속도가 매우 빠르기 때문에 사용자가 임의로 지정한 시간마다 1로 Output을 보내는 Clock Divider를 사용한다. 그리고 Clock Divider의 Output인 dclk를 STD\_LOGIC my\_clk에 저장한다.

|  |
| --- |
| 2. Project Description |

◦ Signal 선언:

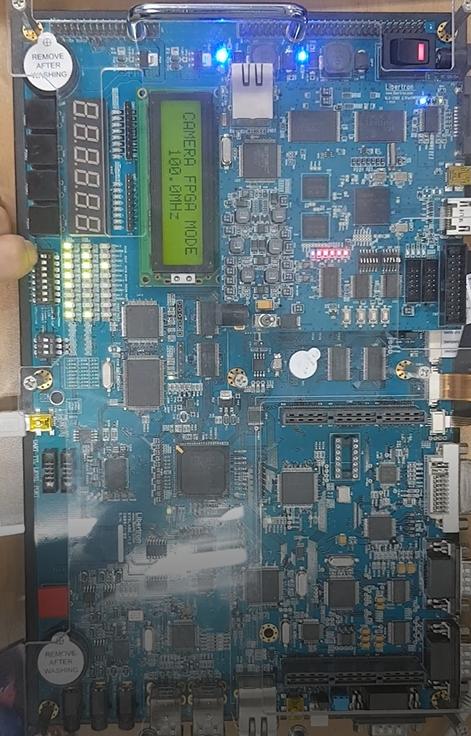
* display\_signal: led에 출력하기 위한 정보를 저장하기 위한 Signal로 LED로 출력을 하기 위해서 display\_signal을 이용한다.
* dip\_reg\_in: Dip Switch의 입력을 저장하기 위한 Signal로 좌측부터 오름차순으로 선언하기 위해서 downto가 아닌 to를 사용한다.
* led\_reg\_top\_in: 상단 LED의 출력을 위한 정보를 저장하기 위한 Signal
* led\_reg\_top\_out: 상단 LED의 출력을 위한 정보를 참조하기 위한 Signal
* led\_reg\_mid\_in: 중간 LED의 출력을 위한 정보를 저장하기 위한 Signal
* led\_reg\_mid\_out: 중간 LED의 출력을 위한 정보를 참조하기 위한 Signal
* led\_reg\_bot\_in: 하단 LED의 출력을 위한 정보를 저장하기 위한 Signal
* led\_reg\_bot\_out: 하단 LED의 출력을 위한 정보를 참조하기 위한 Signal
* my\_clk: Clock Devider의 출력을 위한 정보를 저장하기 위한 Signal
* 출력을 하기 위해서는 24Bit를 가지는 Display Signal는 LED의 출력정보를 저장하기 위해 32Bit를 가지는 led\_reg와 차이가 있는데 led\_reg의 일정부분만을 출력에 사용한다.

◦ 코드 설명:

* if dip\_reg\_in = “0-------” then: Dip Switch 의 좌측 버튼이 0인 경우에는 Register에 2개의 삼각형 모양을 가지는 신호를 저장하고 아무 것도 출력하지 않는다.
* if dip\_reg\_in = “100-----” then: Dip Switch의 버튼이 100인 경우에는 Register에 저장된 신호를 출력한다.
* if dip\_reg\_in = “101-----” then: Dip Switch의 버튼이 101인 경우에는 LED를 좌측으로 움직여야 한다. 여기서 이어 붙이는 연산인 &을 사용하는데, 가장 우측의 레지스터를 좌측으로 이어 붙여서 Rotate를 구현한 후에 8Bit\*3를 display\_signal에 넣어준다.
* if dip\_reg\_in = “110-----” then: Dip Switch의 버튼이 110인 경우에는 LED를 우측으로 움직여야 한다. 여기서 이어 붙이는 연산인 &을 사용하는데, 가장 좌측의 레지스터를 우측으로 이어 붙여서 Rotate를 구현한 후에 8Bit\*3를 display\_signal에 넣어준다.
* if dip\_reg\_in = “110-----” then: Dip Switch의 버튼이 110인 경우에는 LED를 우측으로 움직여야 한다. 여기서 이어 붙이는 연산인 &을 사용하는데, 가장 좌측의 레지스터를 우측으로 이어 붙여서 Rotate를 구현한 후에 8Bit\*3를 display\_signal에 넣어준다.
* if dip\_reg\_in = “111-----” then: Dip Switch의 버튼이 111인 경우에는 LED를 Flickering 시켜야 한다. 이 부분에서 앞서 구현했던 my\_clk를 사용했는데, my\_clk가 1인 경우에는 아무것도 출력하지 않으며 LED를 끄고 그렇지 않은 경우에는, led\_reg에 저장해 두었던 변수들을 LED에 출력함으로 깜빡이는 모습을 구현하였다.
* else: 이후에 혹시나 오류가 발생한 경우에는 24Bit의 모든 LED를 출력하여 오류를 탐지하도록 구현하였다.
* led\_output <= display\_signal: 출력에 대한 정보를 저장해둔 display\_signal을 led\_output으로 넣어줌으로 실제 LED로의 출력을 구현하였다.
* 전체 코드

|  |  |  |
| --- | --- | --- |
|  |  | **Project Output** |

|  |
| --- |
| 3. Demostration |



◦ 실행화면

◦ 실행 동영상: 용량 등의 문제로 아래의 Youtube에 올려 두었습니다.  
LED\_Mover: https://youtu.be/7GTNSzBsXBg

|  |
| --- |
| 4. ETC |

◦ 질문

- Process문은 Parameter(?)로 온 변수가 변경되었을 때에만 실행된다고 이론시간에   
배웠는데 Xillinx에서는 While문처럼 실행되는 이유가 궁금합니다!

- 아래는 제가 작성한 코드인데, 빨간 네모가 상당히 많은 부분에서 사용되고 있는데   
이 부분을 어떻게 독립적으로 구현하여 코드의 중첩을 해결할 수 있는지 궁금합니다!

