고급 디지털회로설계 보고서

Design Project

PC based Function Generator

전자공학과

11조

20111526 원 재 성 (Won Jae Seung)

20111569 최 재 성 (Choi Jae Seung)

1. 제목

PC 및 FPGA 기반 Function generator (PCFG)

2. 목적

- 설계에서 주어진 기능의 구현을 통해 디지털 회로 설계 능력을 배양한다.
- Block diagram으로 회로를 도식화 하고 Timing diagram을 통해 회로의 동작을 설계하는 능력을 배양한다.
- Xilinx ISE와 Simulator(ISim 또는 ModelSim)를 사용하여 회로를 설계하고 검증하는 능력을 배양한다.

3. 목표 및 기준 설정 - 설계 과정1

가. 전반적인 설계:

요번 설계의 목표는 VHDL 언어로 FPGA를 제작하여 PC로 제어하는 기본적인 파형발생기와 디지털 오실로스코프를 설계하는 것이다. 이때 PC의 역할은 FPGA에 명령을 넣어주는 제어의 역할 이고, USB를 통해서 명령을 입력받으면 FPGA는 이에 따라 7가지 기능으로 작동하게 된다. 이때 입력으로는 USB를 통해서 PC로부터 파형에 대한 Data를 입력받거나, ADC로부터 sin 파형을 입력 받을 수 있고, 이를 가공하여 DAC로 출력 하거나 USB를 통하여 PC로 Data를 보낼 수 있다.

이때 FPGA의 동작을 위하여 Main Frequency 가 입력되며, 8254를 통해 Sys Frequency 가 만들어지고 그다음 모든 파트들이 작동하게 된다.

나. 파트별 설계 목표 및 배경지식:

0) Data Specification

USB-Logic		**Controller-RAM**		**ADC/DAC**	
Data Bus	8bit	Data Bus	8bit	Data Bus	8bit
Address Bus	9bit	Address Bus	10bit	Data Type	Unsigned Integer
Clock Frequency					
Clock Buffer를 통해 FPGA로 들어오는 signal의 frequency				40MHz	
8254를 통해 분주하여 system clock으로 사용하는 signal의 frequency				5MHz	
ADC conversion rate				40MHz	
DAC conversion rate				DAC 수행마다 가변	

0) USB를 통한 PC와의 통신:

USB를 이용한 통신에는 여러 가지 Mode 가 있는데, 요번 설계에서 우리는 GPIF master mode 중 Full Handshake I/O Model을 이용하였다. 이 방식은 Device가 매우 느린 rate로 동작할 때, 가변한 transfer time 이 필요할 때 쓰는 방식으로 PC에서 데이터를 보낼 주소를 알려주면 Device에서 Ready=1을 보내서 준비가 되있음을 알려주고, PC에서 wen (ren) 신호와 데이터를 보내주면 Device가 다 받는대로 Ready=0을 보내어 모든 과정이 종료가 되었음을 알려주는 방식의 통신방식이다. 이러한 방식 덕에 VHDL 코딩에 있어 시간적 여유를 두고 각 state를 작성 할 수 있고, 데이터 송수신에 있어 안정성을 확보 할 수 있다.

0) **FPGA 입, 출력 핀**

Pin Name	Bit Width	Description
Input		
Address	9	회로의 mode를 지정할 때 또는 8254에 control word를 쓸 때 사용
CMD_data	1	Address signal의 enable 역할
wen	1	회로 전반적인 동작에서 write enable 역할
ren	1	회러 전반적인 동작에서 read enable 역할
clk0~2 (8254)	1	8254 내부의 각 counter에 입력되는 clock
gate0~2 (8254)	1	8254 내부의 각 counter의 enable signal
ADC data	8	ADC로 입력된 신호가 unsigned integer로 sampling 된 것
Output		
Data (DAC)	8	Analog signal을 생성하기 위해 DAC로 보내는 data (unsigned integer)
Clock (DAC)	1	DAC의 clock (ADC와는 달리 FPGA에서 clock 인가)
ready	1	Full-handshake 방식의 USB 통신을 하기 위해 필요한 signal
Input/Output		
Data (PC)	8	PC와 FPGA간의 data 전송을 위한 signal

1) Mode 별 설계 내용

가) System Clock Setting Mode (8254 Setting):

기본적으로 8254에는 입력받은 Clock을 분주하거나 rate pulse를 발생 하는 등 여러 가지 동작 모드가 있는데, 우리는 여기서 clock을 분주하는 용도로 사용하려고 한다. PC에서 USB통신을 이용하여 8254를 셋팅해주는 Address와 Data가 넘어오게 되고 이를 이용하여 8254는 작동하게 된다.

분주하는 용도로 사용하기 위해 Address 에 11(A1,A0)을 주어 Control word Register를 선택하고 이 때 data '36H'를 주어 counter0을 선택, LSB->MSB 순으로 데이터가 입력되며 mode3로 출력 및 16-bit binary counter 가 되도록 설정한다. 이어서 Address에 '140H'를 주어 counter0이 선택되고 각각 분주에 필요한 LSB, MSB 데이터를 입력한다.

이번 설계에서는 40Mhz의 Main Clock을 5Mhz로 낮춰야 하므로 140H의 데이터로는 8을 입력하였다.
8254 Setting 이 끝나면 기본적으로 CSG (=Control Signal Generator) 는 Idle State 로 있게 된다.

나) PC Mode

PC Mode 에는 총 4가지 모드가 있다. RAM0와 RAM1 에 각각 읽기와 쓰기 이다.
USB 통신을 통하여 컴퓨터로부터 데이터가 넘어오면 차곡차곡 기록하거나 읽으면 된다. 이때 RAM의 Address 는 PC로부터 안 오고, RAM counter를 이용하여 순차적으로 기록하거나 읽어야 한다. 설계에서는 RAM0와 RAM1에 11bit Address 카운터를 두고 각가 이를 이용하였다. 이때 RAM 은 Dual Port Memory 로 8bit * 2048 = 2KB 의 용량을 가졌다.

USB 통신에 의하여 한번에 8bit의 데이터만 넘어오고 끝난다. 만약 8bit*30 = 30Byte 의 데이터를 기록하고 싶다면 USB통신을 총 30번 해야하는 것이다. 이때 매번 데이터를 받고 Ready=0을 한다음 다른 명령이 올 수 있으므로 일단 Idle State 로 빠지게 설계를 하였다.

또한 RAM 에 기록을 하다가 갑자기 읽기모드로 바꿀 수 있다. 이때 Counter는 같은 Counter 가 이용됨으로 이를 감지하여 Reset 하도록 설정 하였다. 또한 RAM0 -> RAM1 혹은 역으로 mode를 바꿀 경우도 카운터를 reset 하도록 설정 하였다.

Write 모드에서는 매번 counter를 올릴때마다 내부에 Ram0_Data_CNT 라는 값에 기록을 하게 된다. 추후 Data Transfer 시 이 값을 이용하여 RAM0 에 기록된 데이터 양을 알고 그만큼 Transfer 하게 된다.

PC Mode 에서는 read, write 가 개별 하나의 명령임으로 이 작동이 끝날때마다 Idle State로 빠지게 되어있다.

다) Data Transfer Mode

Data Transfer Mode 에서는 RAM0에 기록된 데이터를 RAM1으로 넘겨준다. 이때 데이터의 개수가 몇개인지는 Control Signal Generator에서 알고 있어야 한다. 우리는 RAM0 Write 또는 RAM1 Write에서 다른 Mode 로 변경될 때 알아서 Ram counter 의 값을 CSG 에 별도의 변수 (Ram0_Data_CNT) 에 기록할 수 있도록 설계하였다. 그래서 Data Transfer 시 RAM1의 카운터는 Ram0_Data_CNT 까지만 증가하면된다. 이때 유의할점은 RAM에서 Read 할 때 한 Clock 딜레이 되어 값이 나옴으로 RAM0 카운터가 RAM1의 카운터보다 먼저 켜지게 되고, REN 신호 또한 WEN 신호 보다 먼저 켜지게 된다.

주어진 개수만큼의 데이터 (Ram0_data_cnt 만큼) 전송이 다 되고나면 Ready=0을 주며 Idle State 로 빠진다.

라) DA Mode

DA mode 에서는 RAM1의 값을 DAC로 출력하게 된다. 이때 미리 RAM1 에 데이터가 어디까지 기록되어있는지를 알고 있어야 하며, 그 값만큼 즉 RAM1_Data_Cnt 만큼 데이터를 출력하고 마지막 번지까지 출력한 다음 다시 0번지부터 출력하여 연속적으로 작동하게 하였으며, 이를 통해 우리의 설계 목표인 Function Generator 가 가능하도록 하였다.

PC와는 USB 통신을 하다보니 일단 DA Mode Address가 오면 Ready=1을 주었다가 곧 이어 Ready=0을 주게 된다. 이때 DAC 출력은 계속 이루어 지고 있으며 후에 PC에서 DA Stop Address 가 넘어오면 그때야 비로서 멈추게 된다.

마) AD Mode

앞에서는 PC와의 통신을 통하여 RAM 에 데이터를 입력하였다. AD Mode 에서는 Analog Signal 즉 함수발생기의 신호를 직접 RAM 에 기록하는 Mode 이다. 이때 한 파장(주기)가 몇인지는 CSG가 직접 알수

가 없으므로 PC에서 Data Bus를 통해 알려준다. 즉 AD Mode Address 와 함께 ADC Signal에서 catch 해야 할 데이터 개수를 BUS를 통해 제공한다. CSG 에서는 이 값을 잘 기록하여 RAM0의 카운터를 통제 해야한다.

바) Interpolation Mode

interpolation이란 샘플과 샘플 사이에 데이터를 채우는 것을 말한다. 즉 interpolation mode에서는 원래 데이터 사이에 추가데이터를 넣어 채우는데 이때 이 데이터는 연속성을 띄게 된다. 이를 이용하면 클럭이 느려지는 효과를 얻을 수 있다.

우선 interpolation mode 가 되면 RAM0의 현재의 값들 사이에 000을 집어 넣어 간격을 넓힌다. 그다음 LPF 필터와 convolution 하여 결과를 얻어 내면 000입력하였던 값들이 Data와 Data 사이를 알아서 매워주어 우리가 원하는 결과가 나온다. 이때 convolution 특성상 Transient 구간이 생기는데 앞뒤로 8bit를 잘라주어야 깔끔한 파형을 얻을 수 있다.

여기서는 Data000 Data000 식으로 Zero fill 한다음 convolution 했으므로 클럭은 1/4로 줄어들게된다. 이때 000이 Data와 Data 사이를 매우게 되지만, Sin 그래프 특성상 기울기가 계속 변함으로 interpolation 결과의 sin그래프는 깔끔한 파형보다는 약간 울퉁불퉁한 모양을 띄게 된다.

사) Reset Mode

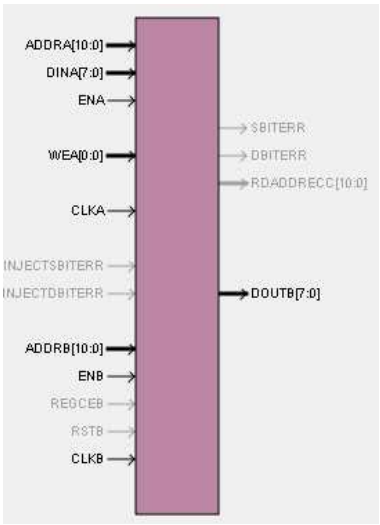
본 설계에는 Reset 이 Hardware Reset과 Software Reset 이 존재한다. Software Reset은 PC로부터 Reset Address를 받아 수행하는 것으로 8254 빼고 모두다 Reset 하게 된다. Hardware Reset 은 8254 도 Reset 시키는 것으로 완전히 기기를 처음 켰을 때처럼 다시 8254 셋팅부터 해주어야 한다.

아) Idle Mode

항상 Idle Mode 에 있다가 PC로부터 명령이 넘어오면 동작을 수행하게 된다.

2) 파트별 설계 내용

가) Simple Dual Port RAM (ipcore 이용)



요번 설계에서는 Simple Dual Port RAM을 이용하였다. Dual port 라 하면 데이터의 입 출력 포트가 개별로 있다는 뜻으로 Data BUS를 덜 복잡하게 이용할 수 있다는 장점과 Data In/Out을 동시에 할 수 있다는 장점이 있다. (Single port 경우 하나의 Data BUS를 공유하게 되므로 중간중간 Latch 가 필요하여 복잡하여짐)

A사이드가 입력이고, B 사이드가 출력이다.

ENA는 Enable Aside (= WEN) , ENB는 Enable Bside (= REN) 이다. 특이점은 WEA[0..0] 포트가 있어 Write Enable 시 여기에도 signal을 주어야 한다. 이때 버스형태로 [0..0] 으로 연결 되어야 한다.

주소도 개별적으로 주어야 하는데, 읽기와 쓰기를 동시에 할 일이 없으므로 요번 설계에서는 하나의 Address 신호로 묶어서 Control Signal Generator 에 연결하였다. CLK 또한 같은 Sys_Clock을 이용하였다.

나) Counter

본 설계에서는 PC에서 Address 는 Mode를 Select 하는데만 사용되고, 램에 기록할 때 램의 Address 는 카운터를 통해서 공급받게 된다. 즉 원하는 Random 의 데이터를 선택하지 못 하는 대신 Address 대역폭에 있어 이득을 취할 수 있다. 이로인하여 본 설계의 RAM은 순차적 읽기 쓰기만 가능하다.

RAM 이 8bit*2048 의 램이므로 11bit 의 Address를 요구하므로 Address counter 는 11bit를 갖는다.

다) Latch

외부 소자와 회로의 온전한 데이터 입/출력을 위해서는 Latch를 사용하는 것이 권장된다. Latch 는 CLK이 들어올 때 데이터를 읽어와서 다음번 데이터가 들어올때까지 그 값을 출력하고 있으므로, 보다 안정적이게 외부의 데이터를 공급받을 수 있다. 본 설계에서는 Data in/out Bus, ADC, DAC, Data bus에서 사용된다.

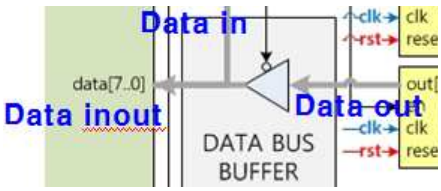
라) MUX

MUX는 데이터가 흘러갈 길을 선택해주는 역할을 한다. SEL 신호를 입력 받아 길을 선택하게 되며, MUX를 통해 원하는 값을 좀더 효율적으로 공급받을 수 있다. (데이터는 BUS를 통해서만 오가는데 MUX를 둠으로써 불필요한 구간을 끊어 좀더 효율적이게 혹은 동시에 BUS를 사용할 수 있게 해준다.)

여기서는 총 3개의 MUX가 사용되었으며 RAM0 와 RAM1 의 입력 그리고 그사이 bus 데이터 전송간에 쓰인다.

마) Tri- State Buffer

Buffer는 enable 신호가 들어올 경우 데이터를 통과시키고, disable 될 경우 High-Impedance 상태로 만들어 다른 소자간의 Data Transfer가 가능하도록 한다. 만약 Buffer 가 없을 경우 FPGA 내부 Data bus가 PC 의 USB Data BUS 까지 이어지는 모양이 되어 BUS 사용에 있어 효율이 떨어지게 된다.



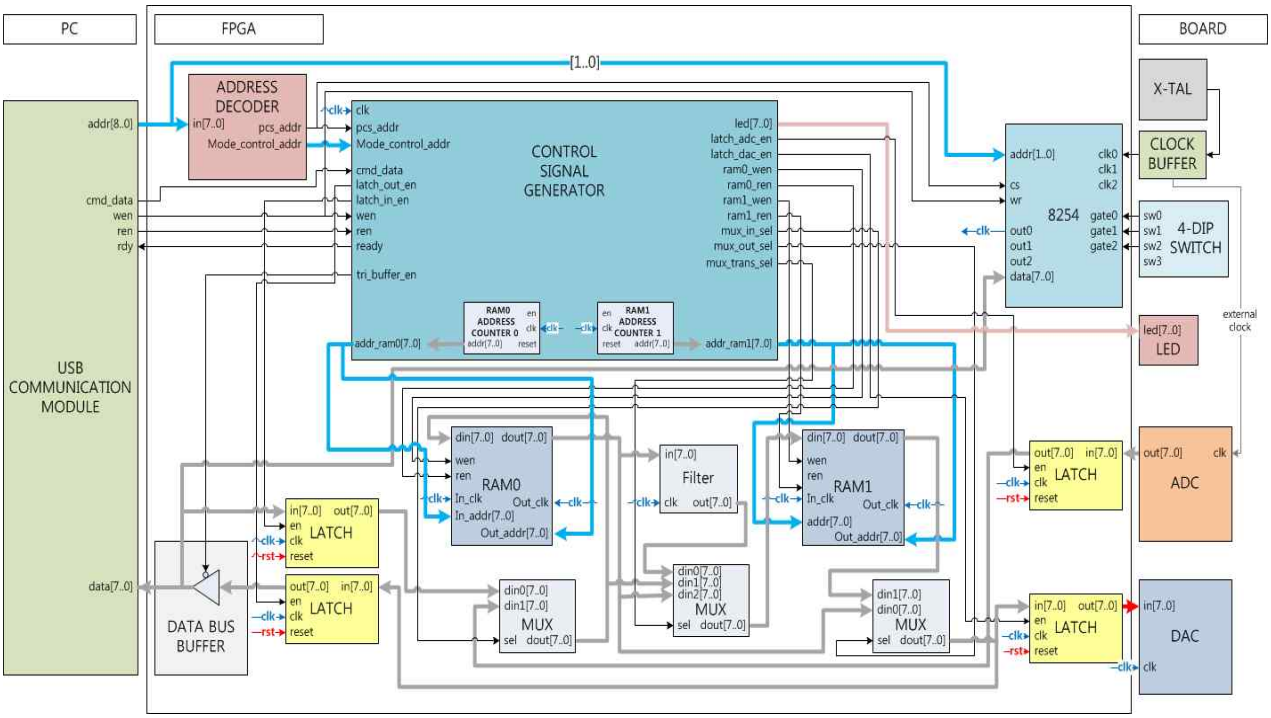
바) Multiplexer (ipcore)

interpolation Mode에서 zero-fill을 한뒤 LPF 필터와 convolution을 하게 되는데 이 convolution을 수동으로 설계할 경우 각각의 값들으 곱한다음에 더하는 꼴이 된다. 곱셈에 있어 IEEE Library로 처리하는데 번거로움이 있으므로 요번 설계에서는 ipcore에서 Multiplexor (곱셈기)를 만들었다. 이를 이용하면 곱셈기에서 A와 B에 값을 넣어주고 클럭을 넣어주면 결과가 출력된다.

```
x1 : mul
port map( A => s_in(0),
          B =>"00000100011",
          clk => m_clk,
          P =>s_out(0));
```

3) 전제 Block Diagram 및 Control Signal Generator

가) Block Diagram



나) Control Signal Generator

Control Signal Generator 에서는 모든 신호들을 관장한다. 처음에 8254 분주가 시작된 다음부터 나오는 SYS_Clock을 이용하여 작동하며, 내부에는 RAM0 와 RAM1 에 Address를 공급할 Counter를 갖고 있다. 데이터 하나가 이동하기 위해서는 해당되는 길(Bus) 에 Latch와 MUX를 연결해주고, 상황에 맞게 WEN 신호나 REN 신호를 주어 데이터를 이동시켜야 하며, 박자에 맞게 Counter에 Enable 신호를 주어야 한다. 또한 USB 통신 규격에 맞게 PC 에 Ready 신호를 보내 데이터를 주고 받아야 하고, 또한 각 Mode 에 맞는 상황들을 제어해야한다. 즉 본 설계의 제일 핵심 부분이라 할 수 있다.

다) Address Decoder

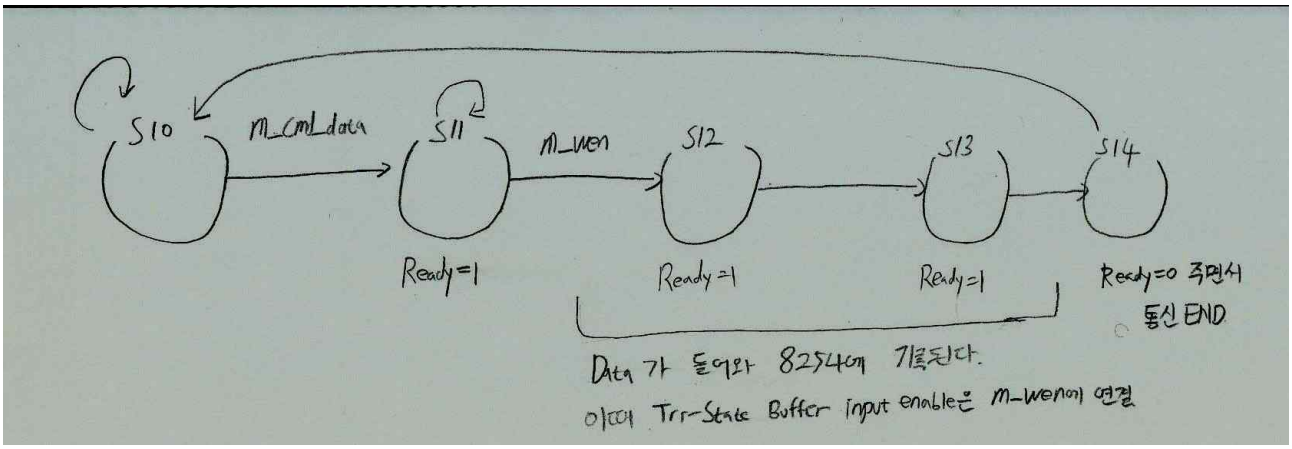
Address Decoder 는 9비트의 어드레스를 입력받아 CSG에 좀더 간략한 형태로 데이터를 넘기는 역할을 한다. 이때 9비트중 하위 2비트는 8254에도 연결이 된다. 140~180H의 주소를 입력받으면 mode 에 맞게 해독하여 4bit의 Mode_Control_Addr 와 PCs_addr를 이용하여 CSG에 데이터를 넘겨준다.

4. 합성 및 분석 - 설계 과정2

설계를 시작할 때 Timing Diagram 은 별도로 작성하지 않았다. Timing Diagram 경우 그리는데 시간도 많이 걸리므로 State Diagram 에 어느 시기에 어떤 신호가 필요하고 출력되는지 적음으로써 Timing Diagram을 대체하였고, 이를 토대로 CSG을 설계하였다.

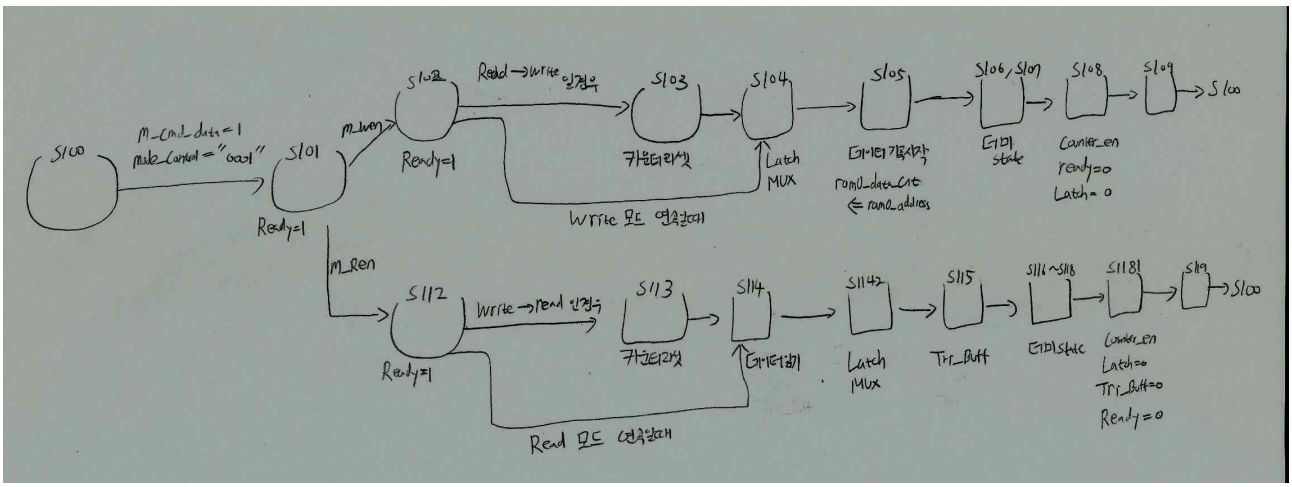
이하 각 Mode 별 State Diagram 이다.

1) 8254 State Diagram



8254 의 State Diagram 이다. 현재 설계에서 Tri-State Buffer 의 input enable 신호는 아예 m_wen 에 연결을 해두어서 PC에서 Data 가 넘어올 때 알아서 열린다. (Tri-buffer output enable 신호는 CSG에서 제어하게 되어있다.)

2) PC Mode State Diagram



PC 모드의 State Diagram 이다. RAM0 기준으로 만들어 졌으며, RAM1도 동일한 방식이다.

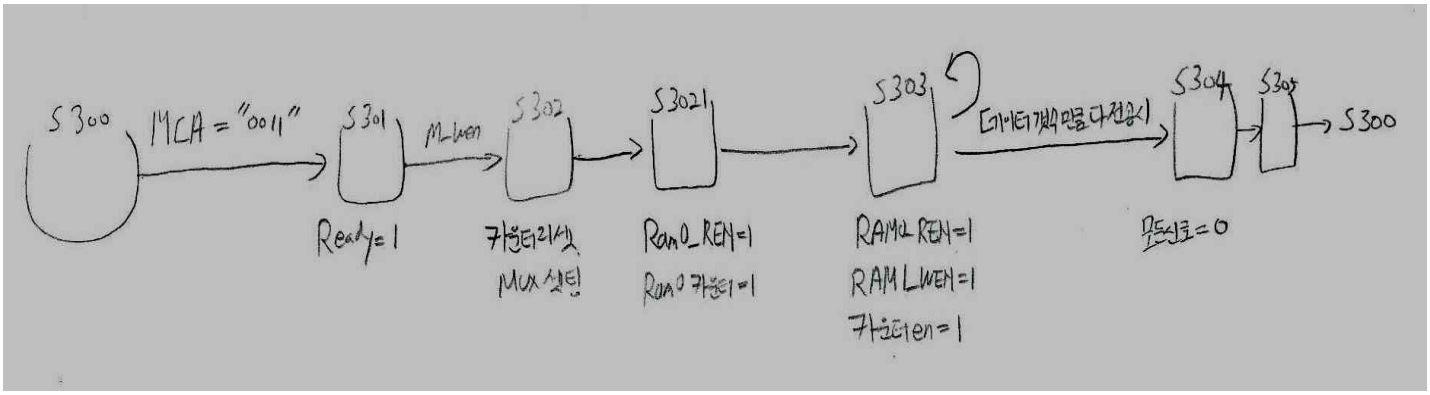
S100 은 PC mode 의 Idel State 이다. 이상태에서 PC mode 의 어드레스 신호와 층_data 신호가 들어오면 S101 로 넘어가고, 여기서 REN 이냐 WEN 이냐에 따라 S102 (WEN), S112(REN) 으로 나누어진다. 그다음 s103 104 경우 Read mode에서 write mode 로 넘어온 경우 카운터 reset 이 필요하므로 s103 에서 해주고, 그게 아닌 경우 바로 s104로 넘어간다. 이것은 if문 으로 처리가 가능하며 read 모드에 있다가 write mode 로 넘어갈 경우 s_ram0_read_count 값이 0이 아니게 됨으로 if문에서 이를 감지하여

s103으로 보내 counter를 리셋하게 된다. 이때 s_ram0_read_count 는 다른게아니라 read mode에서 카운터에서 출력하는 address 값을 변수로 받아 노은 것이다. 그리하여 이 변수값이 0 이 아닌 경우 read mode 에 있었다 온 것으로 간주 할 수 있으므로 s103 스테이트로 가고 카운터 리셋 및 s_ram0_read_count 신호도 리셋하게 된다.

같은 원리로 write 모드에 있다가 read 모드로 넘어갈때도 카운터를 리셋하게 된다.

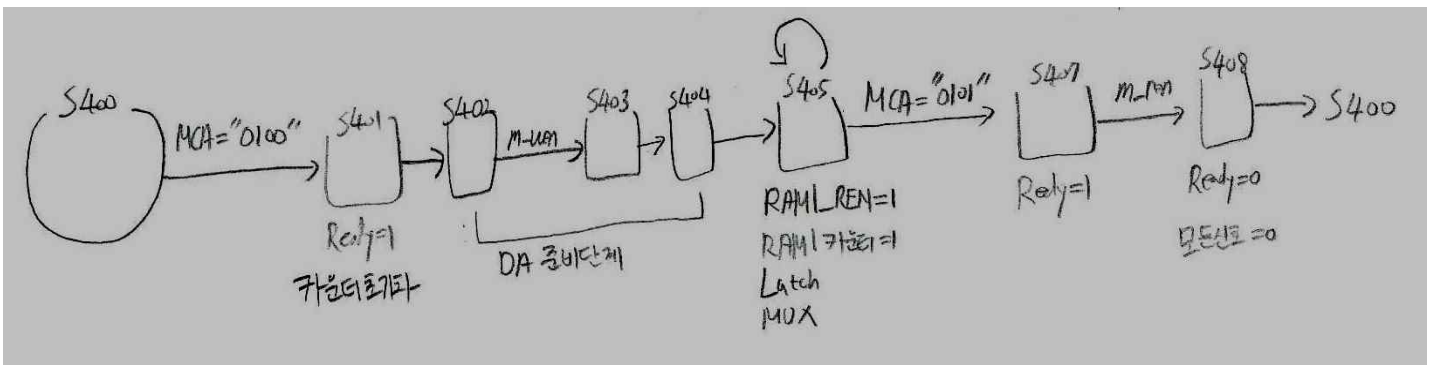
이때 다른점이 있다면 write 모드에서는 개수를 알아야 나중에 transfer 나 DA를 할 수 있으므로 이를 s_ram0_data_count 라는 값으로도 받아서 백업해놓는다. 후에 이를 이용하여 if문 처리하게된다.

3) Transfer mode State Diagram



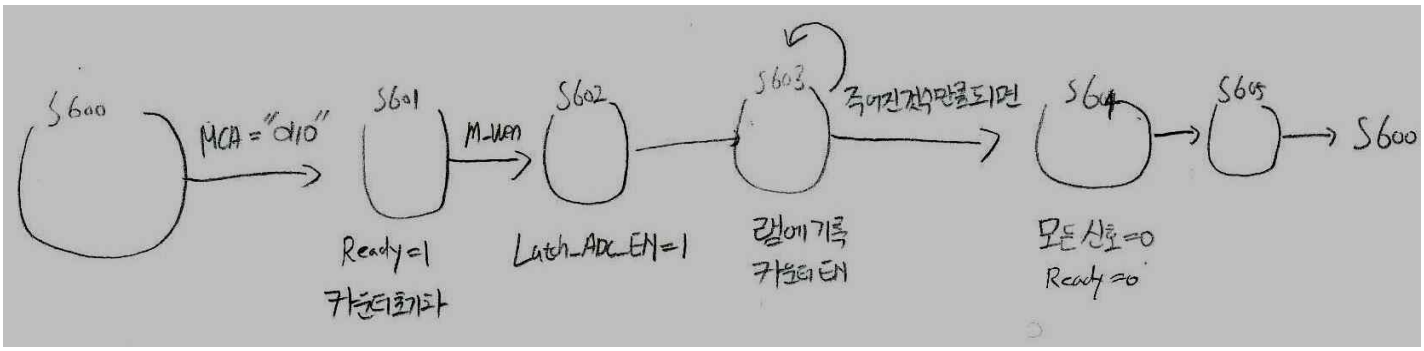
MCA (Mode Control Address) 값을 받으면서 state 가 시작한다. 별 다른 특이점은 없고, 다만 PC 모드에서 받아놓은 s_ram0_data_count 와 현재 ram0_address 가 같아지면 그것까지 복사하고 s304로 빠져나간다. s304에서 모든 signal을 0을 주고, 이러면서 transfer mode 가 종료된다.

4) DA Mode State Diagram



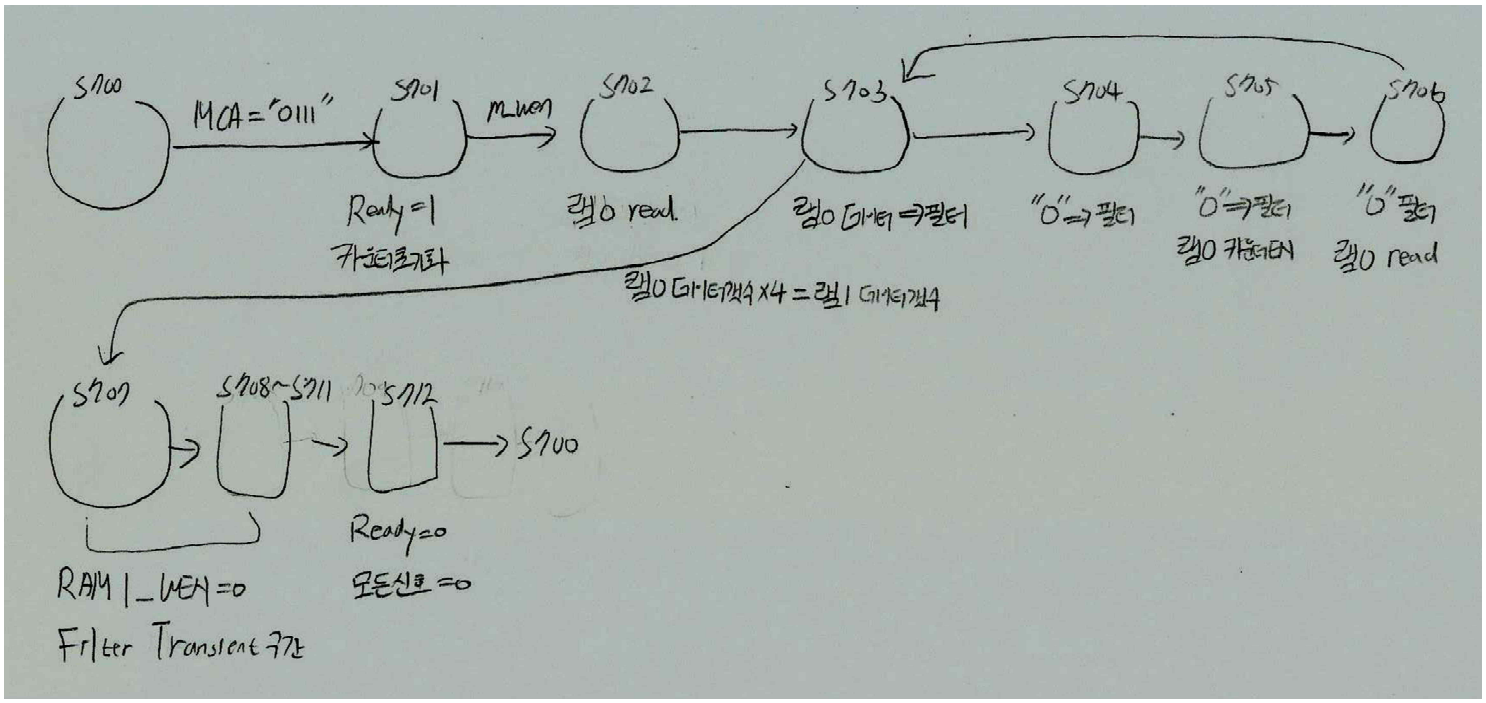
MCA (Mode Control Address) 값을 받으면서 state 가 시작한다. 별 다른 특이점은 없고, 다만 PC 모드에서 받아놓은 s_ram0_data_count 와 현재 ram0_address 가 같아지면 그것까지 출력한다음 counter를 리셋한다. 이렇게 루프를 돌다가 MCA에서 DA stop address가 들어오면 s407로 빠져나오면서 DA출력을 멈춘다. 그뒤는 USB 통신에 맞게 신호를 주고 받는다.

5) AD Mode State Diagram



MCA (Mode Control Address) 값을 받으면서 state 가 시작한다. 별 다른 특이점은 없고, 다만 PC 모드에서 AD모드에서 받을 데이터 개수를 Data bus 로 넘겨 주는데, CSG에서 이를 받아 s_ad_data 로 받아놓고 후에 ram0_address 와 같아지면 그것까지 입력받고 루프를 빠져 나온다.
그 뒤는 USB 통신에 맞게 신호를 주고 받는다.

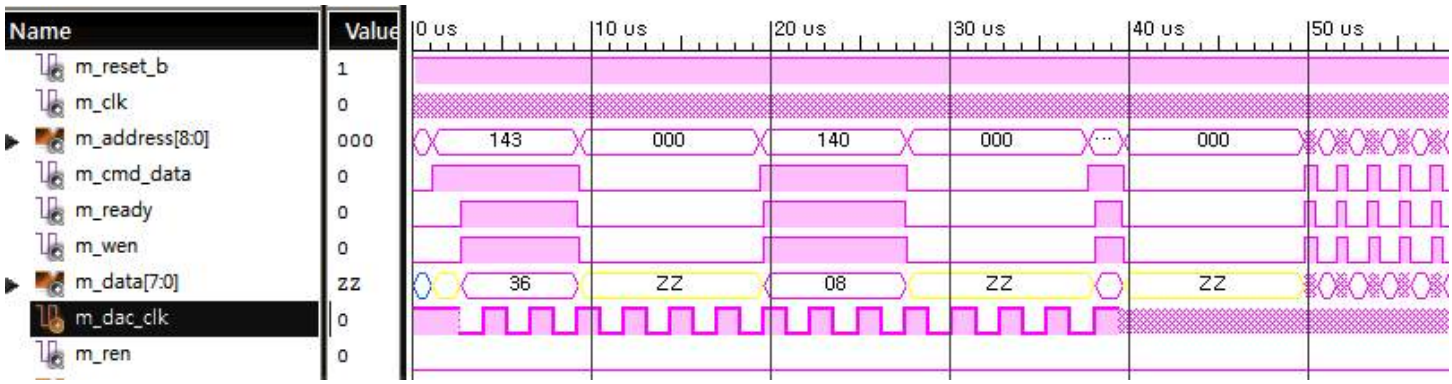
6) Interpolation Mode State Diagram



MCA (Mode Control Address) 값을 받으면서 state 가 시작한다. 우선 여기서 중요한건 데이터가 Linear 하게 들어가는 것이다. 즉 중간에 끊어짐이 있으면 안된다. 필터에는 Ram0_Data 0 0 0 Ram0_Data 형태로 꾸준히 들어가야되고 이 결과가 처음 8개 그 다음것부터 Ram1 에 순차적으로 기록되어야 한다. RAM 경우 REN을 주면 그다음 클럭부터 데이터가 나오므로 루프 돌기 전에 미리 S702 state에서 read 신호를 주었다. 그다음 S703~s706 은 루프를 돌다가 램0 데이터 개수 *4 한 값과 램1의 데이터 개수가 같아지는 순간 스테이트가 s707로 넘어간다. 이하 필터에서 필요한 값까지 받기위해 여유 스테이트를 두고 S712에서 모든 신호를 0을 줌으로써 끝이 난다.

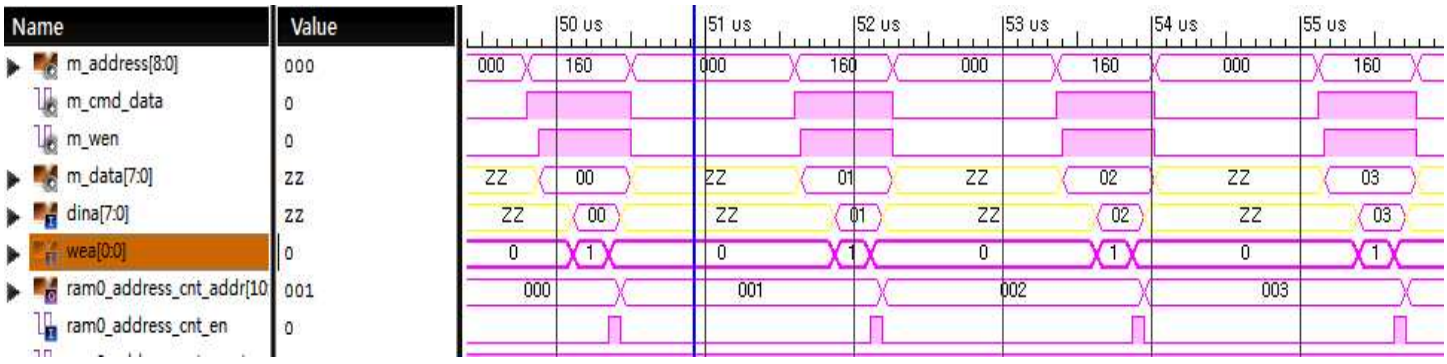
5. 시험 및 평가 - 설계 과정3

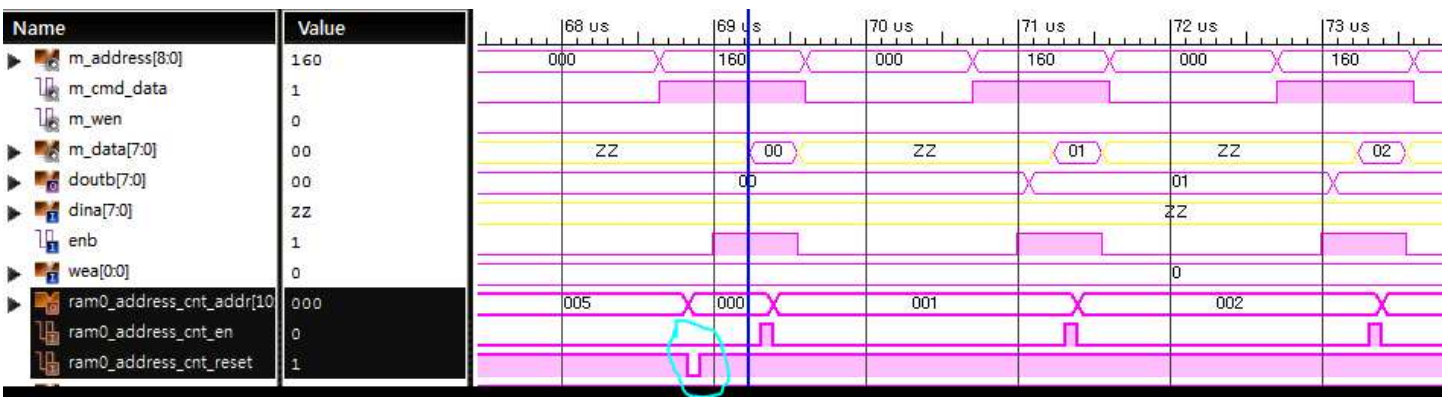
1) 8254 sys_clk setting



command_data가 1이 된다음 m_ready가 1이 되고 m_wen이 1이되면서 데이터가 들어오는 것을 확인 할 수 있다. 총 3번의 데이터가 들어온 다음부터 8254에서 sys_clk 이 나오는 것을 확인 할 수 있다.
dac_clk 에 sys_clk 이 연결됨으로 이 신호를 보고 알 수 있다. 1번째 데이터가 들어온 직후 긴 주기를 그리며 파형이 나오기 시작하였고, 3번째 데이터가 들어온 다음부턴 정상적으로 출력되기 시작했다.

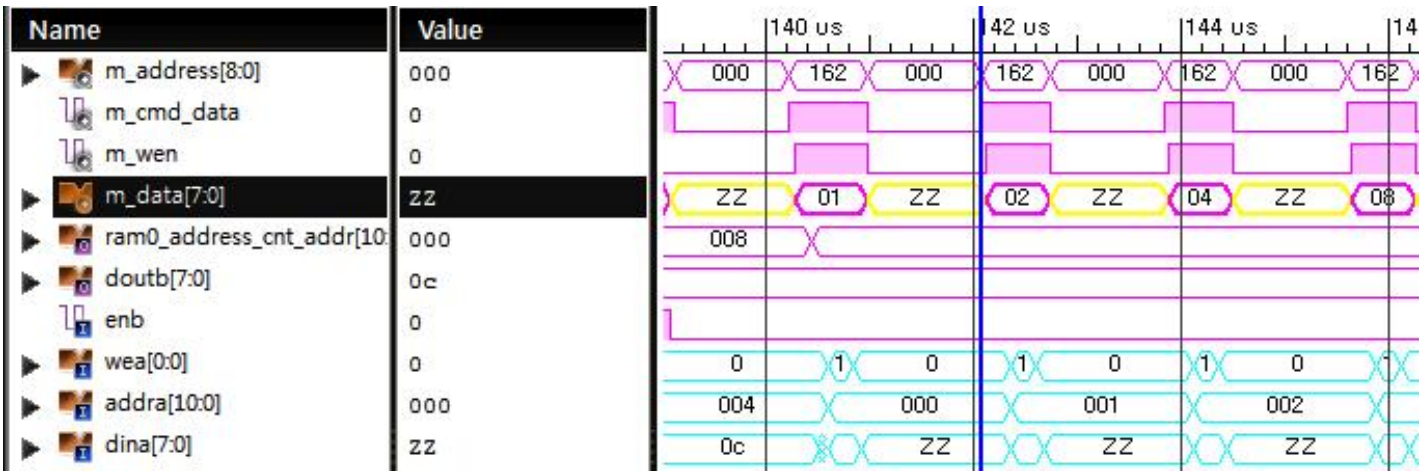
2) PC Mode

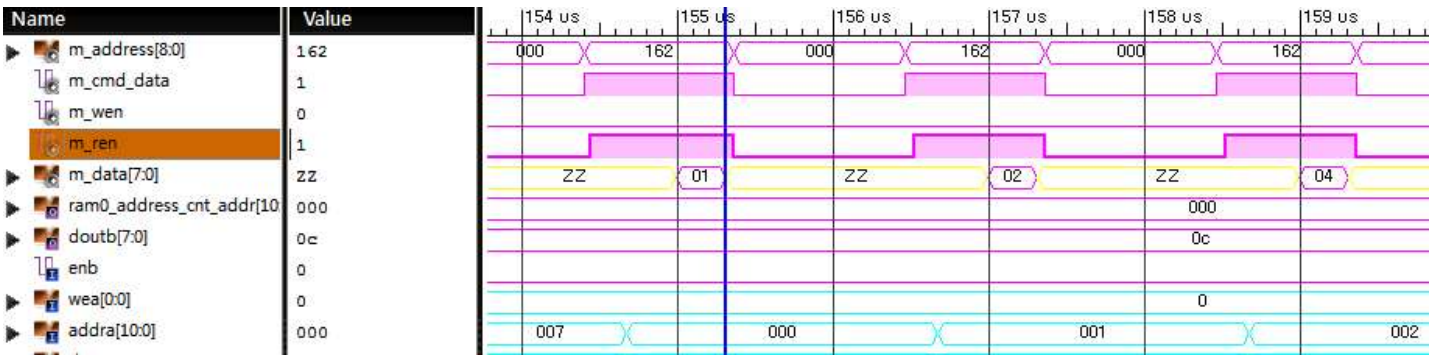




첫 번째 그림이 ram0에 데이터를 기록하는 결과이고, 두 번째 그림이 ram0로부터 데이터를 읽어오는 결과이다. din_a 까지 (RAM0 의 데이터인풋까지) 데이터가 잘 넘어옴을 알 수 있고, 또한 ram counter가 적절한 시기에 리셋 및 count enable 됨을 알 수 있다. 또한 적절한 더미 state를 집어넣어서 시간적 여유를 두어 실제 FPGA상에서도 문제 없이 돌아가도록 설계 되었다.

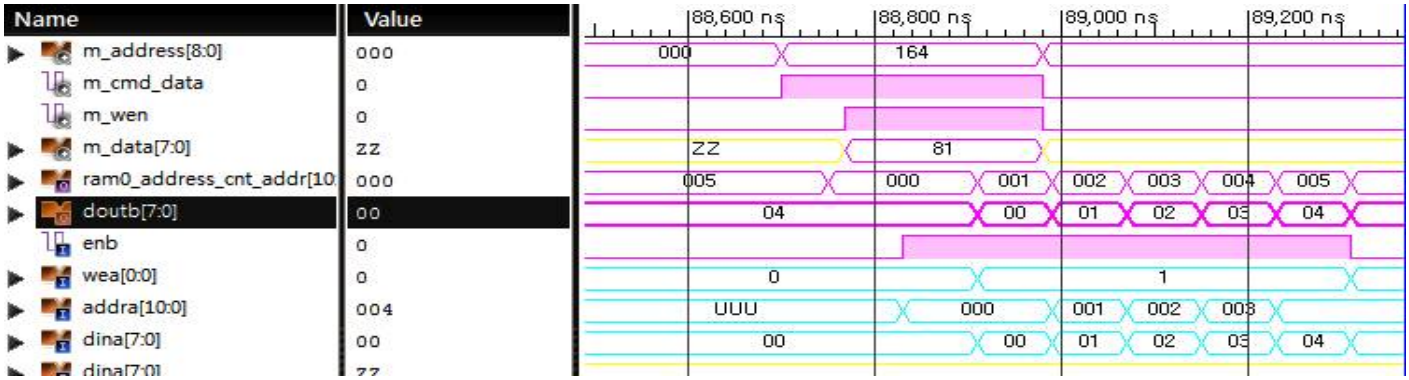
Read mode에서 보면 시작하자마자 Reset 신호가 들어가 counter를 초기화 시켜줌을 알 수 있다. 이는 위의 설계파트에서 설명하였듯이 해당 조건을 감지하여 초기화한 것이다. 또한 Read mode 도 더미 state를 두어 m_data 까지 데이터가 잘 전송됨을 확인할 수 있다.





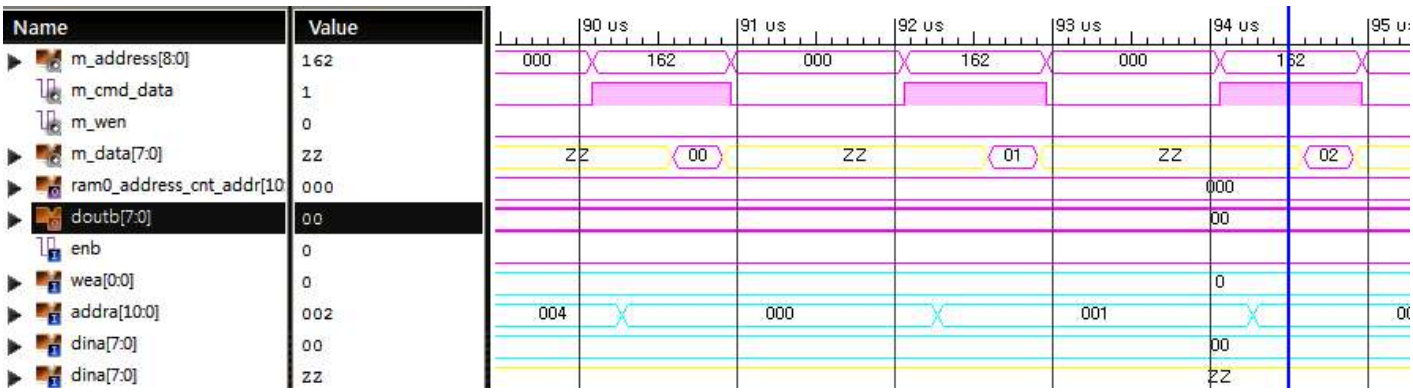
RAM1에서 읽고 쓰기 또한 잘 됨을 확인 할 수 있다.

3) Transfer Mode



Transfer Mode 에서도 RAM0에서 출력된 값이 RAM1에 잘 입력되고 있음을 확인 할 수 있다. 그림에서 보면 RAM0 Read 경우 데이터가 한 클럭 딜레이 되어 나옴을 확인 할 수 있고 이 데이터가 RAM1에 올바른 번지수에 기록됨을 확인 할 수 있다. 이때 딜레이 때문에 RAM0 는 READ와 Counter Enable을 먼저 시작하여 같은 시간에 메모리 번지수는 다르게 가르키고 있음을 확인 할 수 있다.

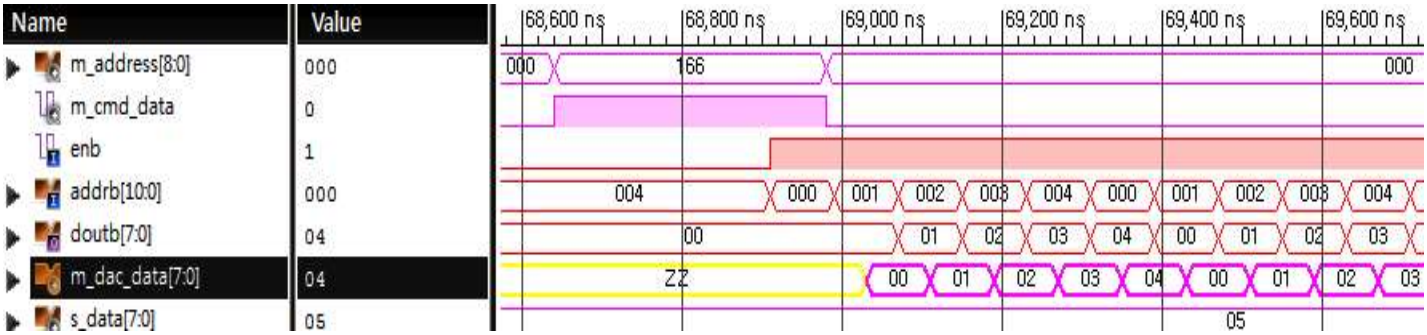
데이터는 시뮬레이션의 편리를 위해 0~4까지만 주었다.



또한 Transfer 된 값들이 RAM1에서 잘 읽혀져 나옴을 확인 할 수 있다.

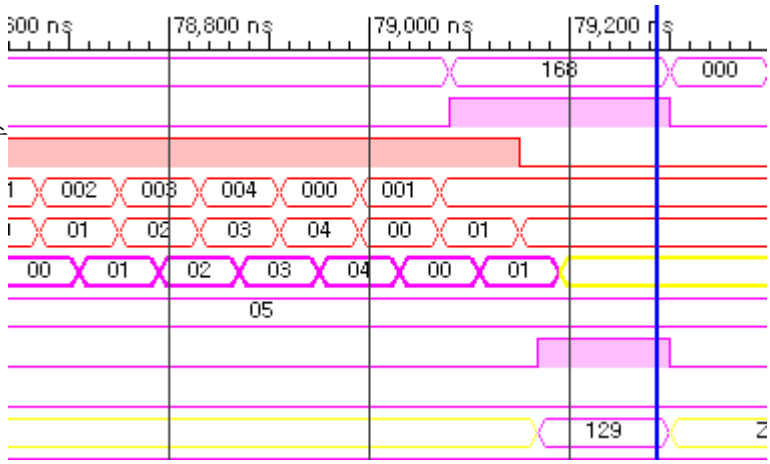
이로써 Transfer 도 잘 되고 RAM1 Read 도 잘 됨을 확인 할 수 있다.

4) DA Mode

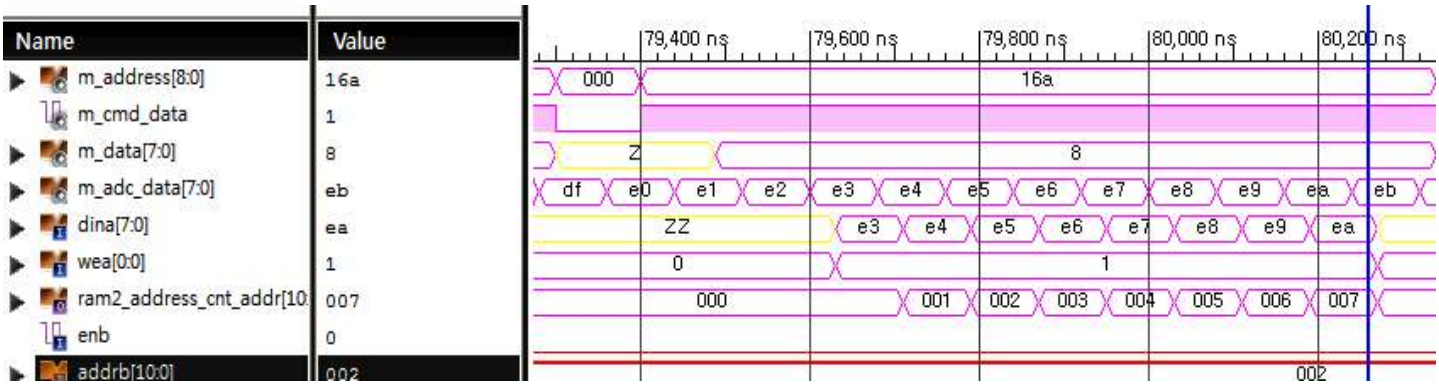


DA mode에서 166을 주고 루프가 시작한다음 ready 신호가 0이 되어 m_address 는 000 으로 빠짐을 확인 할 수 있었다. 또한 RAM1에 있는 Data 들이 mux 와 latch를 통과하여 m_dac_data 로 잘 나오고 있음을 확인할 수 있다. 데이터를 00~04까지 주었으므로 m_dac_data를 보면 00~04를 반복하여 출력하는 것을 확인 할 수 있다.

DA_Stop 또한 잘 되는 것을 확인 할 수 있다.

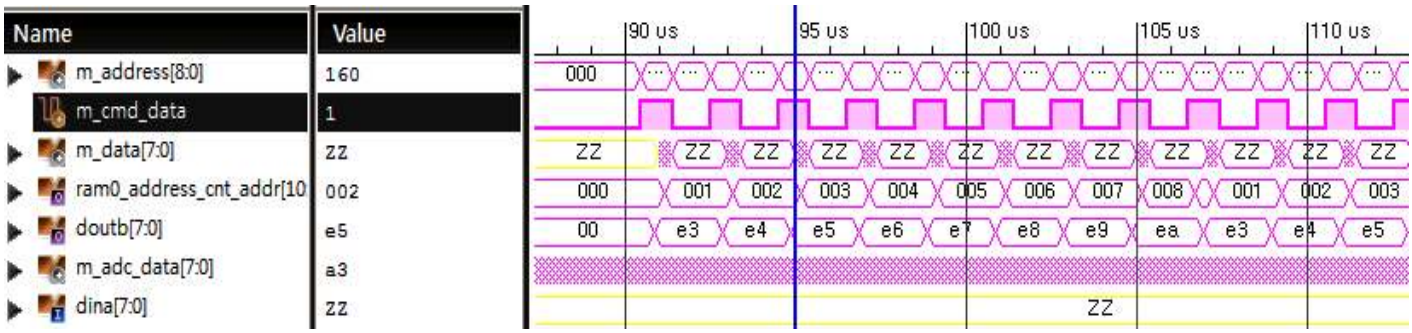


5) AD Mode



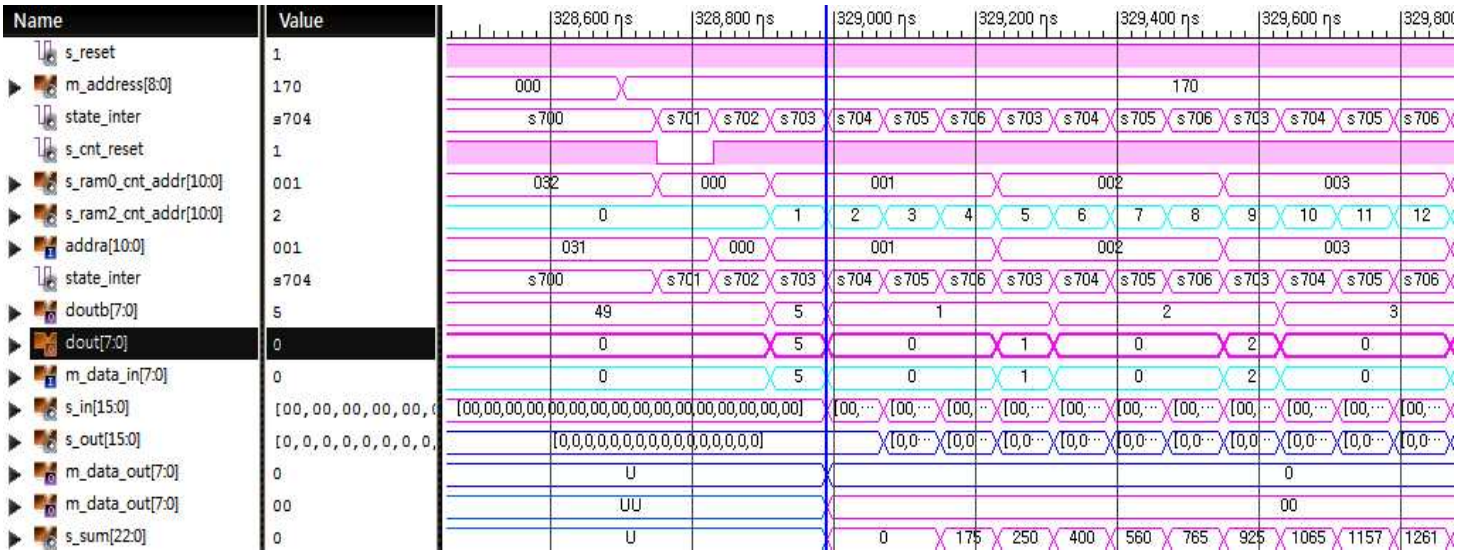
AD_Mode 에서는 PC에서 샘플링할 데이터 수를 Data bus 로 보내준다. 8개의 데이터를 샘플하라고 Data bus를 통해서 들어오고 있고, CSG에서 이를 인지하여 지나가는 m_adc_data 중 e3~ea 까지 정확히 8개를 램에 기록하는 것을 확인 할 수 있다. 이때 address counter 또한 정확한 타이밍에 작동을 하고 있음을 확인 할 수 있다.

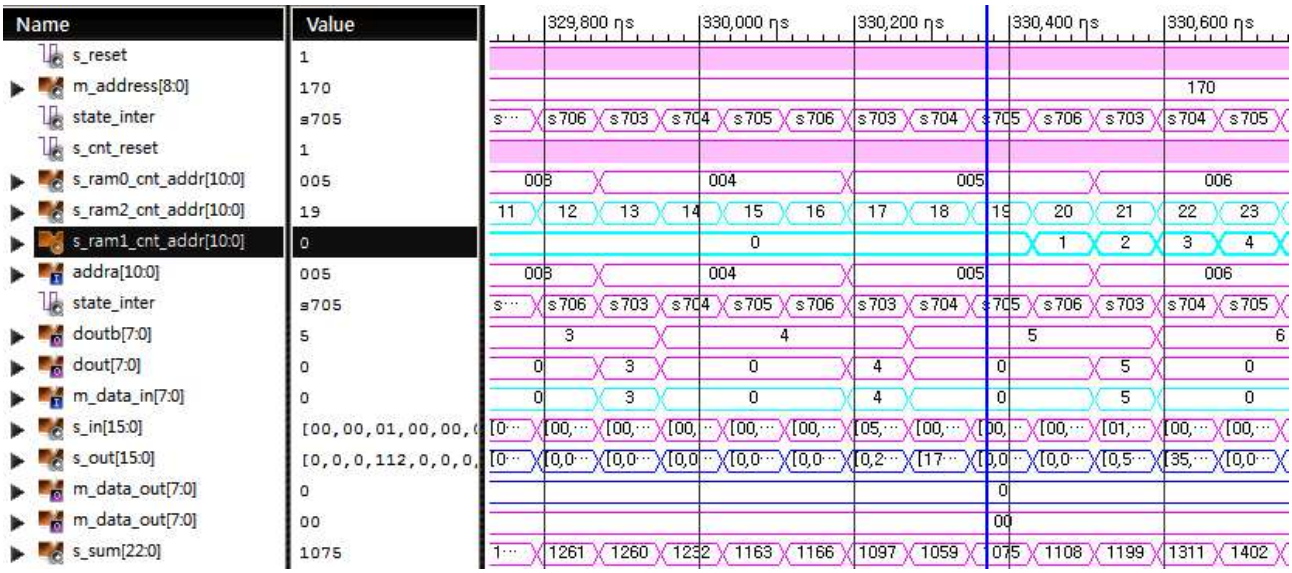
이때



RAM0 출력을 보면 ADC에서 인풋한 값을 정확히 출력함을 확인 할 수 있다.
또한 데이터가 8개여서 8개씩 반복출력 하는 것을 확인 할 수 있다.
이를 통해 우리는 ADC 데이터가 8개라는 정보가 ram0_data_cnt 에 잘 넘어갔고, PC mode에서 이값을 토대로 연속적인 데이터 즉 sin 파를 만들어 낸다고 가정할 수 있겠다. 실제로 오실로스코프를 통해서 출력하였을 때 연속적인 파형이 나왔다.

6) Interpolation Mode




Name
Value
s_reset
1
m_address[8:0]
170
state_inter
s705
s_cnt_reset
1
s_ram0_cnt_addr[10:0]
005
s_ram2_cnt_addr[10:0]
19
s_ram1_cnt_addr[10:0]
0
addra[10:0]
005
state_inter
s705
doutb[7:0]
5
dout[7:0]
0
m_data_in[7:0]
0
s_in[15:0]
[00,00,01,00,00,(
s_out[15:0]
[0,0,0,112,0,0,0,
m_data_out[7:0]
0
m_data_out[7:0]
00
s_sum[22:0]
1075
329,800 ns
330,000 ns
330,200 ns
330,400 ns
330,600 ns


Name
Value
s_reset
1
m_address[8:0]
170
state_inter
s709
s_cnt_reset
0
s_ram0_cnt_addr[10:0]
000
s_ram2_cnt_addr[10:0]
0
s_ram1_cnt_addr[10:0]
0
addra[10:0]
033
state_inter
s709
doutb[7:0]
0
dout[7:0]
0
m_data_in[7:0]
0
s_in[15:0]
[00,2f,00,00,00,
s_out[15:0]
[0,0,3760,0,0,0,
m_data_out[7:0]
10
m_data_out[7:0]
0a
s_sum[22:0]
22450
344,600 ns
344,800 ns
345,000 ns
345,200 ns

6. 결과 도출 - 설계 과정4

구현한 회로를 FPGA에서 동작시켜 안정된 결과를 출력하는지 확인한다. 만약 시뮬레이션에서 성공하였는데 FPGA 상에서 실패했다면 그 이유를 분석하고 다시 회로를 구현한다.

설계 보고서를 작성하기 까지 많은 시행 착오들이 있었다. 이 보고서에는 최종 완성본, 통과한 결과에 대한 보고서이기 때문에 기록이 안 되어 있다.

우선 시뮬레이션에 성공 하였는데 FPGA 상에서 실패했을 때 어떻게 고쳐서 해결하였는지 서술하겠다.

1. 1차, 2차 Fail.

이때 1차 2차 테스트에서 8254마저 분주가 안 되었었다. 분명 시뮬레이션에서는 잘 돌아갔는데, FPGA에서는 꿈쩍도 안 했다. 그래서 조교님들께 물어 본 결과 state 안에서 너무 많은 if문과 신호처리를 하게 될 경우 FPGA에서 잘 안될 수도 있다는 답변을 받았다. 그래서 state 안에서 신호처리는 최소화하고, state문 밖에서 signal 들 처리를 하였더니 정상적으로 작동 되었다.

이를 통해 FPGA 설계시 state 에 대한 파트와 Signal 들에 대한 파트를 최대한 분리 시키는게 더 안정적임을 배울 수 있었다.

2. 3차 Fail (AD_Mode)

3차에서는 AD mode 가 안 되었었는데, 다름이 아니라 AD_CLK을 연결을 안 했었다. TOP_Level 에 output 포트가 존재하기는 하였으나, Block Diagram 상에 별다른 포트가 없어 그냥 두었는데, 나중에 알고보니 ad_clk 가 연결되어야 ADC 가 자극을 받아 값을 넘겨주는것이었다.

3. 3차 fail (Interpolation Mode)

7. 토의

1) 제시된 주요 설계 요소와 제한요소들을 만족하는 설계를 하였는지 논의

제시된 주요 설계 요소와 제한 요소들을 만족하는 설계를 하기 위하여 State Diagram을 작성하고 이에 맞게 signal 들을 나열 한 다음 이를 토대로 VHDL 소스를 작성 하였고, FPGA 상에서의 테스트와 시뮬레이션을 돌려 본 결과 원래 목적 대로 잘 작동하였고, 제시된 주요 설계 요소들을 만족하며 제한 요소들도 만족하여 정상 작동 하였다.

2) Top-down 방식으로 설계하였는지 논의

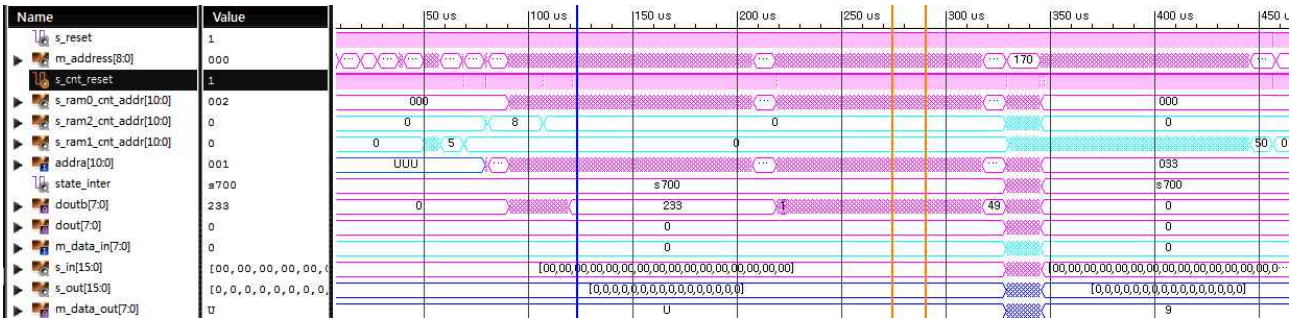
우선 Top-Level에서 접근을 하였다. 가장먼저 해야 할 부분이 Adress Decoder 와 8254 였고 이 둘을 만들고 Top-Level에서 연결하여 정상작동함을 확인하였다. 그다음 각종 소자들을 만들고 CSG를 만들어 PC 모드까지 작동함을 확인 하였다. 그다음 순서대로 Mode를 만들었다. 이덕에 중간에 어떤 Mode에서 에러가 나더라도 선행단계까지는 정상 작동 하였으므로 디버깅 하는데 매우 큰 도움이 되었다. 즉 top-down 방식으로 설계를 하였다고 생각한다.

3) 임의의 입력에 대해서도 안정적으로 동작하는 논의

ram0 write read // ram1 write read // ram0 write read // transfer // ram1 read

를 수행하여 정상적으로 값들이 출력됨을 확인하였다. 위 경우를 통하여 몇 번을 ram에 기록하여도 카운

터 초기화가 정상적으로 작동하여 문제를 안 일으킴을 확인 할 수 있었다.
나머지 DA mode 나 AD mode 경우 애초에 state 2번째에서 다 카운터 리셋을 시킴으로 몇 번을 수행하여도 문제가 없다. 요번 설계에서 임의의 입력에서 안정적인지를 좌우하는곳은 PC mode 라고 생각하고, 테스트 결과 이상이 없었다.



위의 경우들을 처음부터 끝까지 수행했을때의 시뮬레이션 결과이다. 개별 사진은 캡쳐하기 너무 많은것같아 전체 사진만 첨부하였다.

4) logic량은 최적화 되었는지 논의

Logic 량이 최적화 되었다고는 생각하지 않는다. 우선 RAM 카운터를 총 2개만 써야지 최적화가 되어있는 상태 일텐데, DA 모드에서 연속데이터 출력시 reset 신호를 이용하면 카운터가 0 값을 2클럭을 내보내는 바람에 이를 카운터 내부에서 처리하게 만들었고, 그결과 DA모드에서는 별도의 카운터를 사용하게 되었다. AD 모드에서도 별도의 카운터를 쓰게 되었는데, DA 모드 한 다음에 만들어서 별도의 카운터를 사용하게 되었던거 같다. 최적화를 하려면 이 2개의 카운터를 줄이고, state 에 있어서도 더미 state를 조금 줄여야 한다고 생각한다. 더미 state 경우 안정적인 동작을 고려하여 만든것인데, 더미 state를 조금씩 줄여가면서 FPGA상 테스트를 거치면 최적화를 할 수 있다고 생각한다.

5) one hot state로 회로를 구현하였을 때의 장점에 대해서 논의

one hot state 방식은 state를 관리하는 방법 중 하나로 각 state 마다 Flip-Flop을 둬서 관리하는 방식이다. 설계된 회로를 구현하다보면 fan-in 현상을 고려해야 하는데, 한 state 마다 하나의 레지스터를 할당하는 one hot state 방식을 사용하게 되면 register 의 수가 늘어난다는 단점이 있긴하지만, fan-in 문제를 해결 할 수 있다. 이번 설계의 경우 FPGA를 이용하여 회로를 구현하였는데, 보통 레지스터의 수가 충분하므로 one hot state 방식으로 구현하는게 안정성에 있어 좋다고 본다.

6) 해당 설계 과제가 실제 사회에서 응용되는 예와 사회에 어떠한 영향을 미칠 수 있는지를 파악

이번 설계 과제는 PC를 기반으로해서 Funcion을 Generate 하는 FPGA 였다. 여기서 입력장치를 pc 대신 다른 기판을 통해서 할 수만 있다면 이는 우리가 실제 생활에서 쓰는 Function Generator 로써 역할을 할 수 있을 것이다. function generator 라면 원하는 클락대로 출력이 나오는건데, 8254의 분주와 interpolation 의 분주를 조합하면 모든 주파수를 다 만들 수 있을 것이라 생각한다. 또한 ADC, DAC 포트도 있고, 컴퓨터로 Data 도 보낼 수 있으므로 일반적인 function generator 보다 좀더 유용하게 쓸 수 있으리라 생각한다.

7) 설계결과를 실제 제품개발에 사용될 수 있는지를 검토한다.

Function Generator 는 함수를 발생시키는 제품이다. 그런데 요번 설계에서 만든 제품은 별다른

입력이 없으면 자기 혼자서 출력은 못하는 Function Generator 이다. 이 사실만 놓고 보면 실제 제품으로 내놓을 이유가 없는 제품이다. 다만 이 제품은 USB를 통하여 컴퓨터와 통신을 할 수도 있고, 또한 ADC에서 입력받은 함수를 가지고 변형을 취할 수 있으므로 현존하는 Function Generator 에다가 요번 설계에서 개발된 기능들을 첨가하면 좀더 고성능의 다재다능한 Function Generator를 만들 수 있지 않을까 생각이 든다.