ASIC Design Methology

<Front – end design>

1) Design Concept

-내가 무엇을 디자인 할 것인지 ?

2) Design Specification

-디자인 하겠다고 결정된 문제에 대해서 정확한 입력과 출력, 기능과 같은 것들을 알고리즘 수준에서 결정한다. C 모델링 까지 포함.

3) RTL Development

-Verilog와 같은 RTL 코딩을 한다.

4) Functional Verification

-Simulation과 같은 테스트 코드로 파형 검증을 하는 것과 같이 기능을 검증한다.

5) Synthesis

-검증이 완료된 회로를 **합성**해서 **게이트 수준의 회로도로 바꿔준다**. (전반부에서 최종적인 회로도)

-Translation 기능 : Verilog 언어 형태를 회로의 형태로 바꾼다

-Mapping 기능 : Library cell의 형태로 존재하는 gate들을 게이트 수준의 회로도로 되게 매핑한다.

-Optimization : Power 소모, 필요한 동작 주파수에 맞게 최적화 시킨다.

6) Timing Verification

-만들어진 게이트 수준의 회로도를 대상으로 전문적인 timing 검증 툴을 가지고 평가한다.

-원하는 동작 주파수에 따른 전체 회로의 delay를 계산하고 수정한다.

SOC Design Challenges

-Frequency of the design : 동작 주파수

-Number of clock domains : clock 신호의 팬-아웃은 굉장히 큰데 clock은 1개가 아니라 여러개다.

-Number of gates : gate 수가 많아질수록 복잡하다.

-Density : 최대한 chip을 작게 만들어야 한다. =>발열과 파워소모 문제가 발생

-Number of blocks : 관련 gate 들끼리 하나의 block을 이룬다. 따라서, block 각각 마다 배치 문제

-System integration : 위의 것들을 고려하여 하나의 통합 시스템으로 구축해야 함

<back - end esign>

1) Floor Planning

2) Place & Route

-합성이 된 게이트 수준의 회로도가 적절하게 배치가 되었는지, Power 소모, 동작 주파수 등등

3) Post-route Verification

4) Sign-Off

<Verilog HDL ch 6.>

-게이트 수준 모델링은 작은 회로에서는 적합하지만 게이트가 많아지면 부적합 함.

데이터 플로우 모델링 기법

1) **연속 할당문 assign 키워드**

-연속 할당문의 왼쪽은 항상 스칼라나 벡터의 넷, 스칼라 넷과 벡터 넷으로 합쳐진 것이 와야함

-왼쪽에는 스칼라나 벡터의 레지스터형은 올 수 없다

-오른쪽은 레지스터, 넷, 함수 호출문, 스칼ㄹ, 벡터

ex) assign out = i1 & i2; // 연속 할당문 out, i1, i2는 넷이다

assign addr[15 : 0] = addr1_bits[15 : 0] ^ addr2_bits[15 : 0]; // ^는 xor임

//일반적인 연속 할당문

wire out;

assign out = in1 & in2;

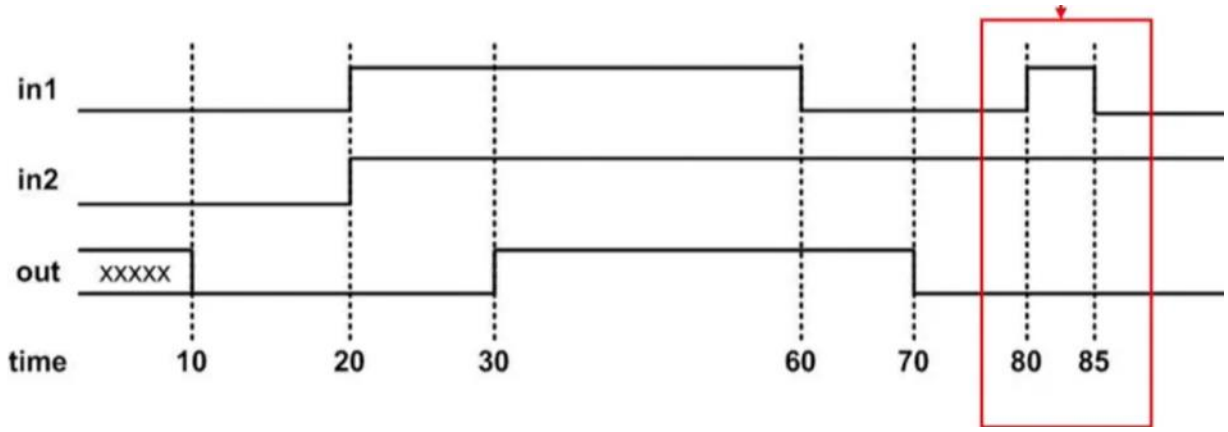
정규 할당 지연 (연속 할당문에 지연값을 직접 할당하는 방법)

ex)

assign #10 out = in1 & in2; // 10 cycle의 delay 이후에 실행된다.

-in1 또는 in2의 값이 변하더라도 10만큼의 지연 시간이 지난 후에 in1 & in2를 계산하여 결과값을 out에 출력한다.

-그러나, **지연 시간 이내에 입력값이 변하면 출력에 영향을 못미침. 90s에도 80s의 입력과 같아야**



함축적 연속 할당 지연

ex)

wire #10 out = in1 & in2;

과

wire out;

assign #10 out = in1 & in2; 는 같은 표현이다.

넷 선언 지연

-넷을 선언할 때 지연값을 연속 할당문에 사용 하지 않고도 할당할 수 있다.

ex) wire #10 out;

assign out = in1 & in2;

결합 연산자

-결합연산자 ({ , }) 는 여러 피 연산자를 한데 묶는 매커니즘을 제공

ex)

A = 1'b1 , B = 2'b00, C= 2'b10, D = 3'b110

Y = {B, C} // 4'b0010

Y = {A, B[0], C[1]} // 3'b101

reg A;

reg [1:0] B, C;

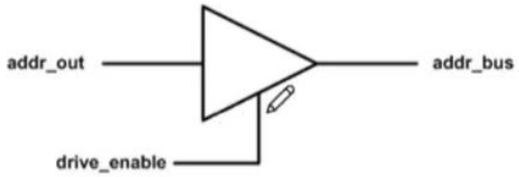
reg [2:0] D;

A = 1'b1, B= 2'b00; C=2'b10; D=3'b110

Y = { 4{A} } // 결과값은 4'b1_1_1_1

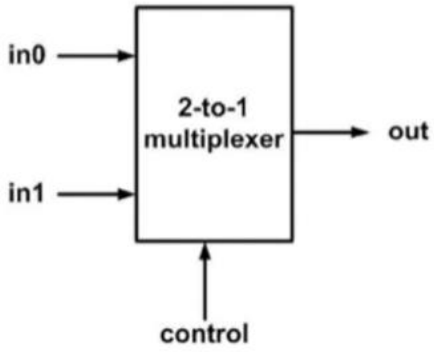
Y = { 4{A}, 2{B} } // 결과값은 8'b1111_0000

연산자의 형태




```
// tristate 버퍼의 기능적 모델
assign addr_bus =
        drive_enable ? addr_out : 36'bz;
```

=>drive_enable 이 1이면 addr_out 출력, 0이면 36'bz 출력




```
// 2-to-1 mux의 기능적 모델
assign out = control ? in1 : in0;
```

=>control이 1이면 in1 출력, 0이면 in0 출력

중첩된 조건 연산자를 이용하여 4 – 1 mux 를 만든 예시

=>assign out = (A == 3) ? (control ? x : y) : (control ? m : n);

데이터 플로우를 이용한 4 : 1 멀티 플렉서

```
module mux4_to_1(out, i0, i1, i2, i3, s1, s0);
output out;
input i0, i1, i2, i3;
input s1, s0;

assign out = ( ~s1 & ~s0 & i0 ) |
             ( ~s1 & s0 & i1 ) |
             ( s1 & ~s0 & i2 ) |
             ( s1 & s0 & i3 ) ;
endmoudle;
```

혹은 assign out = s1 ? (s0 ? i3 : i2) : (s0 ? i1 : i0) 으로 할 수 있다.

```
module stimulus;
reg IN0, IN1, IN2, IN3;
```

```
reg S0, S1;
wire OUTPUT;

mux4_to_1 mymux(OUTPUT, IN0, IN1, IN2, IN3, S1, S0);

initial
begin
```

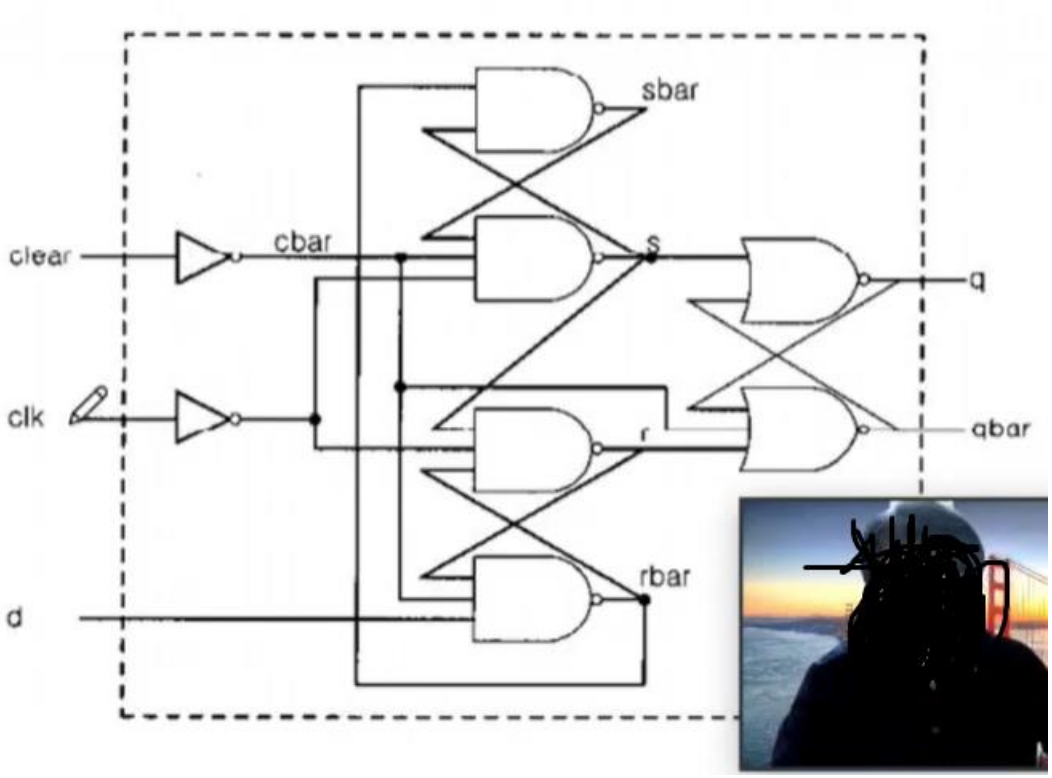
~ 이렇게 stimulus 코드는 동일하게 작성하면 됨.

4비트 전 가산기

```
module fulladd4_df(sum, c_out, a, b, c_in);
output [3: 0] sum;
output c_out;
input [3:0 ] a, b;
input c_in;

assign { c_out, sum } = a + b + c_in;

endmodule;
```



module edge_dff(q, qbar, d, clk, clear);

output q, qbar;

input d, clk, clear;

wire s, sbar, r, rbar, cbar;

assign sbar = ~ (rbar & s),

s = ~ (sbar & cbar & ~clk),

r = ~ (rbar & ~clk & s),

rbar = ~ (r & cbar & d);

assign q = ~ (s & qbar),

qbar = ~(q & r & cbar);

endmodule;

ASIC Verification

1) Functional Verification (기능 검증)

-Simulation : 베릴로그를 이용한 모니터, 파형 분석 (RTL 등)

-Testbenches : simulation를 하기 위한 코드

=>Vector-based : 다양한 입력값들을 기반으로 한 검증 방식

=>BFM-based : 시스템 버스 상에서 각 모듈간의 종합적인 연계 테스트를 위한 것이다. 예를 들어, 버스에 총 5개 모듈이 연결되는데, 2개만 설계가 완료되었다면 설계가 안된 나머지 모듈들을 단지 테스트를 위한 간단한 모듈로만 설계하고 테스트한다. (가짜 인터페이스)

2) Formal Verification

-**수학적인 방식으로 검증**하는 방법이다.

-전반부, 후반부 과정을 통해 RTL이 Gate 수준의 회로도와 같이 논리적으로는 동일한 동작을 하는 회로지만 위와 같은 과정을 진행하면서 여러 문제를 해결하기 위해 변형이 가해진다.

-Formal Verification은 하나의 소프트웨어인데 합성 전의 회로도, 합성 후의 회로도를 입력으로 받아서 논리적으로 동일한 등가회로인지 수학적으로 검증한다.

-위와 같은 Formal Verification 기법을 쓰지 않으면 수정될 때마다 Simulation을 해야 해서 불편하다.

3) Code Coverage

-"내가 작성한 코드가 몇 %까지 테스트가 되었는가?" 를 위한 과정이다.

-테스트를 할 때마다 테스트가 된 부분과 되지 않은 부분을 보여주고 완료 퍼센트를 보여준다.

4) Analog Mixed-Signal Simulation

-아날로그 회로와 디지털 회로가 함께 존재하는 경우인데, 동시에 검증하는 방법이 존재한다.

5) Emulation

-Simulation보다 빠른 방법이다.

-FPGA 칩을 활용해서 새로만든 회로도를 다운로드 받아서 테스트 해보는 방식이다.

Design for Test(DFT)

-Chip을 만든 다음에도 Chip 내부의 중요한 곳에서의 값의 이동을 보기 위해(리버싱과 비슷) 사후 테스트에 관한 회로도

1) JTAG

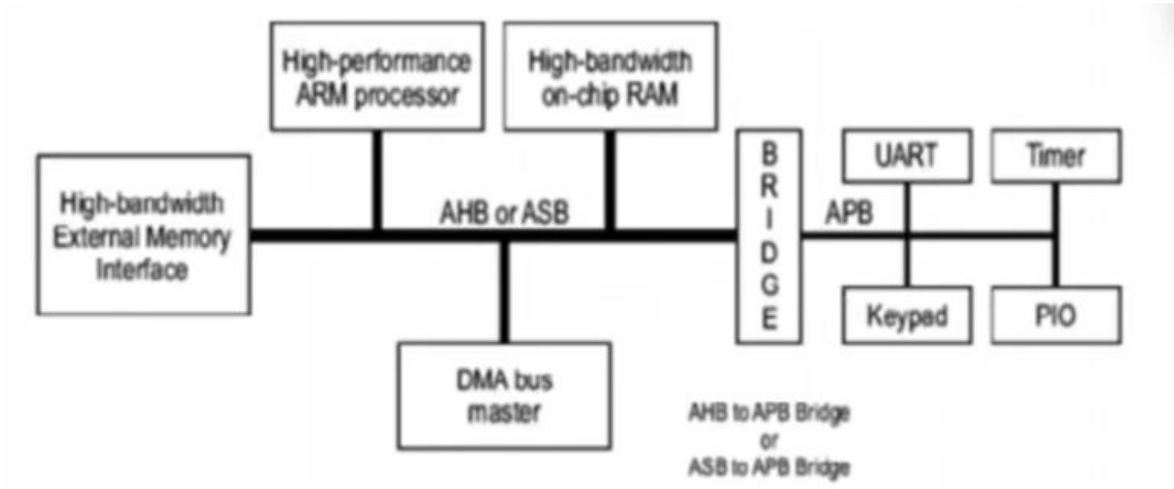
-멀티플렉서와 flip flop으로 구성된 Scan cell 이라는 부분이 있는데, 일반적인 Date가 들어가는 부분과 테스트 데이터를 위한 Scan Data로 입력이 구성되어 있고, Scan Mode Select가 위를 결정한다.

-실제로 Soc에서 수많은 Scan cell이 존재하는데, 하나의 Scan cell의 출력은 다른 Scan cell의 입력으로 들어가고 중요한 데이터를 다루는 레지스터와 같은 것들에 Scan cell을 연결한다.
-임베디드 시스템과 호스트 컴퓨터를 **JTAG interface 케이블**을 사용해서 연결하고 JTAG cable을 이용해 **Scan cell 체인으로 연결된 것들에 대해 정지시키고 읽고 쓰는것이 가능**해진다.
-따라서, **사후검증이 쉽게 가능**해진다.
-JTAG을 구축하기 위해 turn around time이 길어지고 추가적인 회로가 필요하고 칩 사이즈가 증가한다는 단점이 존재한다.

2) ATPG (Automatic Pattern Generation)
-개발자가 테스트할 때 입력값을 일일이 넣지않고, 필요한 테스트 값을 자동적으로 발생시킨다.

3) BIST (Built-In-Self-Test)
-메모리 소자를 테스트하는 방식인데, 메모리 소자마다 자체적으로 테스트 할 수 있는 BIST로직이 있는데 모든 셀들이 이상이 있는지 없는지 확인한다.

AMBA BUS

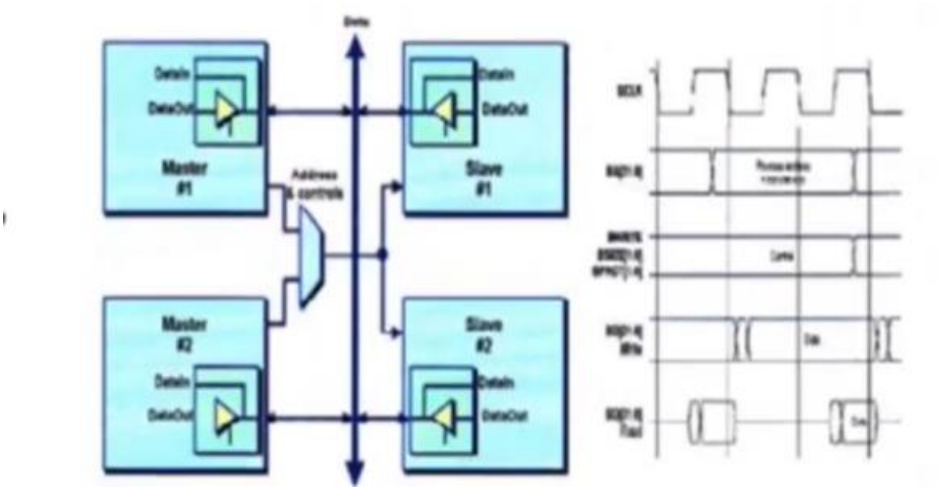


AHB(메인버스) : 고성능 모듈들을 위한 고성능 버스
-파이프라인 구조, Burst transfer로 동작한다.

APB : 데이터가 적은 주변기기들을 위한 버스
-bridge를 통하여 latched된 데이터가 온다.

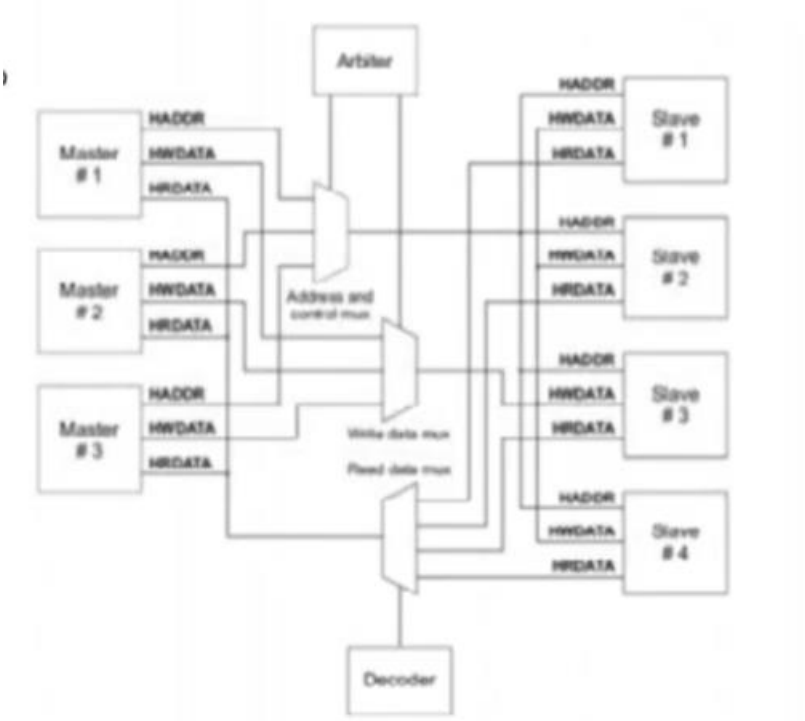
=>AHB와 APB는 서로 특성이 다르기 때문에 Bridge를 통하여 연결한다.

BUS Structure



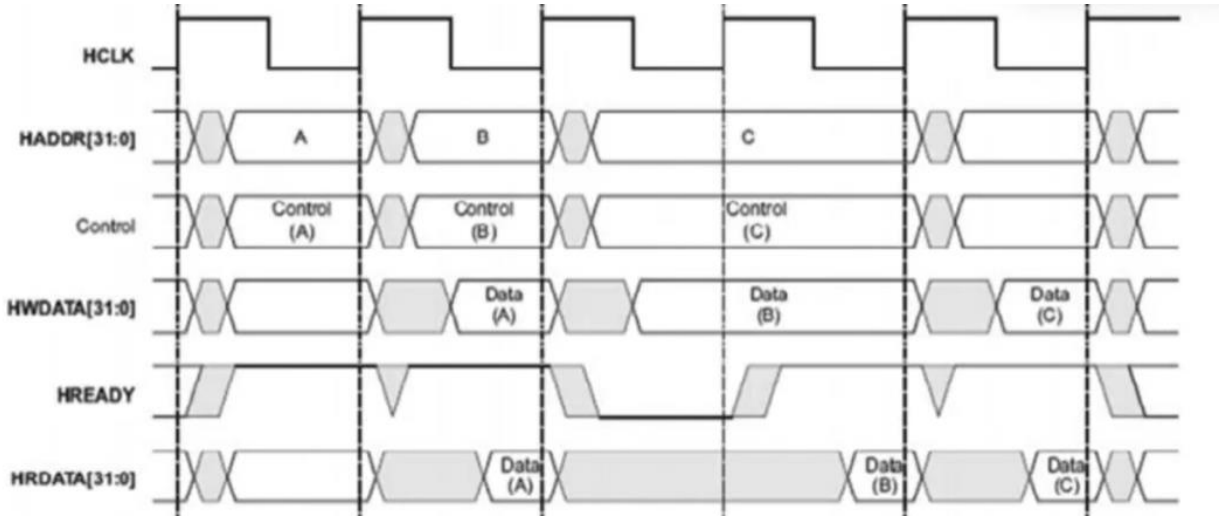
-여러 모듈들은 BUS를 사용하기 위해 tri-state-buffer를 가지고 있다.
-tri-state-buffer의 control을 통해 BUS에 연결여부를 결정할 수 있다 (LOW->High impedence)
-이러한 버스 구조는 속도가 느리고, 고성능 동작을 하는데는 적합하지 않다.

AMBA BUS Architecture



-AMBA BUS의 구조는 tri-state-buffer가 아닌 멀티플렉서로 이루어져 있다.
-slave->master, master->slave 쪽으로 데이터가 멀티플렉서로 전송이 된다.
-해당 멀티플렉서를 컨트롤하는 신호는 **Arbiter**로 부터 나오고, slave에서 master로 가는 멀티플렉서를 컨트롤 하는 신호는 **decorder**로 부터 나온다.

AMBA Waveform



-2단계 파이프라인 구조로 되어있는데, 1번쨔 파이프라인에서 HADDR, Control 신호를 내보내고 2번째 파이프라인에서 데이터를 보내거나 받는다. 데이터를 보내거나 받는 2번째 stage에서 다음번 명령에 해당되는 주소와 control 신호를 내보낸다.

Master transfer type

HTRANS[1:0]	Type	Description
00	IDLE	Master가 데이터 전송이 필요 없음을 나타냄. Slave는 언제나 IDEL에 대해 OK response
01	BUSY	Burst 전송 중간에 master기 다음 사이클을 위한 준비가 안되어 IDLE 상태를 만들고 싶을 때 사용. Slave는 언제나 BUSY에 대해 OK response.
10	NONSEQ	단일 데이터 전송이나 burst 전송을 시작할 때 사용. (전에 전송한 것과 관련 없음)
11	SEQ	Burst 전송 시 NONSEQ의 다음부터는 SEQ로 동작 (전에 전송한 것과 관련 있음)

-IDLE : 다음 데이터를 전송하기 위해 준비하는 상태
-BUSY : 데이터를 보내는 중간에 준비가 안되어 있을 때 대기
-NONSEQ : burst가 아닌 일반 데이터를 한번 보내는 상태, burst 모드의 첫 시작 단계
-SEQ : 연속되게 burst 모드로 전송될 때

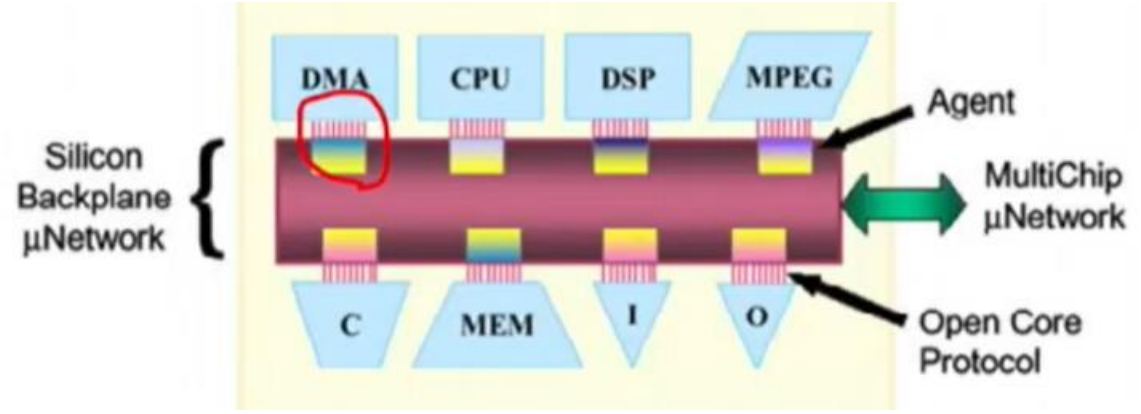
Burst type

1) INCR4, 8, 16 : 주소가 순차적으로 바뀌는 구조

2) WRAP4, 8, 16 : 주소가 주소의 윈도우내에서 바뀌는 구조

ARM CPU Addressing => WRAP 강의 보자 이건...

Sonics Bus



-AMba bus 보다 고성능 버스이다.

-Higher efficiency, flexible configuration

-Guaranteed bandwidth and latency, integrated arbitration

-Pipelining

<Behavioral Modeling>

구조적 프로시저

-Verilog 에는 두 가지 구조적 프로시저 always와 initial이 있다.

-Verilog는 순차적으로 실행되는 C 프로그램과 달리 **병렬적으로** 수행되는 프로그래밍 언어이다.

-Verilog 에는 각 always와 initial 문은 분리되어 수행된다.

-각 수행은 시뮬레이션 시간 0에서 시작한다.

1) initial

-여러 개의 initial 블록은 시간 0에서 동시에 시작하고, 시뮬레이션동안 한 번만 수행된다.

-여러 개의 행위 수준 문장은 반드시 키워드 begin 과 end 문을 이용해서 묶어야 한다.

ex)

```
module stimulus;
reg x, y, a, b, m;
```

```
initial
        m = 1'b0;
initial
begin
        #5 a = 1'b1;
        #25 b = 1'b0; => 두개의 문장이라서 begin, end로 묶어주었다.
end

initial
begin
        #10 x = 1'b0;
        #25 y = 1'b1;
end
initial
        #50 $finish;
endmodule
```

시간	수행된 문장
0	m = 1'b0;
5	a = 1'b1;
10	x = 1'b0;
30	b = 1'b0;
35	y = 1'b1;
50	$finish;

2) always

-시간 0에서 시작되고 블록 내의 문장을 루프 형식으로 연속되게 수행한다.

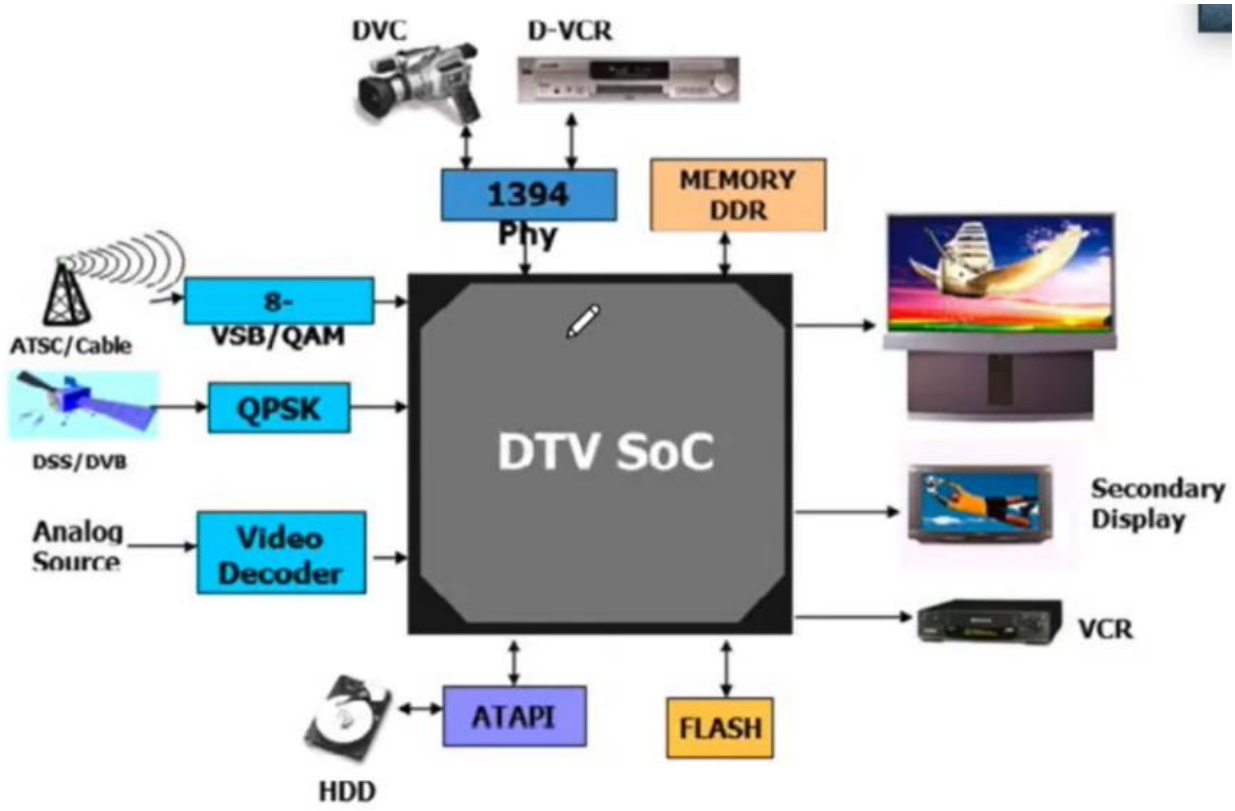
ex)

```
module clock_gen(output reg clock);
initial
        clock = 1'b0;   // 단위 시간 0에 클럭을 초기화 한다.
always
        #10 clock = ~clock; // 매 1/2 주기마다 클럭이 바뀐다(시간 주기 = 20)
```

```
initial
        #1000 $finish;
endmodule
```

D-TV Soc

-비디오, 오디오 및 데이터를 디지털 처리한 후 디지털 전송방식으로 전송하는 시스템
-아날로그 신호에 비해 잡음이 강하고 송신전력이 적다.
-영상 및 음성신호의 대폭적인 대역압축이 가능하다.
-고도의 암호화가 용이



입력부분 : ATSC(지상파)/Cable
-ATSC에서 방송 송출 신호를 뽑아내는 방식이 8-VSB/QAM Modulation 방식이 있다.
-아날로그 source로 된 데이터가 들어올 때, Video decorder가 있다.

출력부분 : 동시에 두 개의 display로 출력할 수 있다.

ATSC

-미국과 한국의 **지상파 전송 기술 표준**

-전송속도 : 19.4Mbps

-6MHz RF 채널

-과거 아날로그에서는 interace 방식으로 보냈는데, 현재는 progressive 방식으로 보낸다.

(interace : 한번 보낼때 홀수 라인, 짝수 라인, 홀수 라인.... 이렇게 번갈아서 보내는 방식)

-Multiple picture format 지원 => HDTV 1920 * 1080 지원

UHD 4k * 2k 지원

-디지털 오디오/비디오 압축

=>현재 **HEVC** 기술 사용

=>처음 나왔을 때는 MP2 기술을 사용하였다.

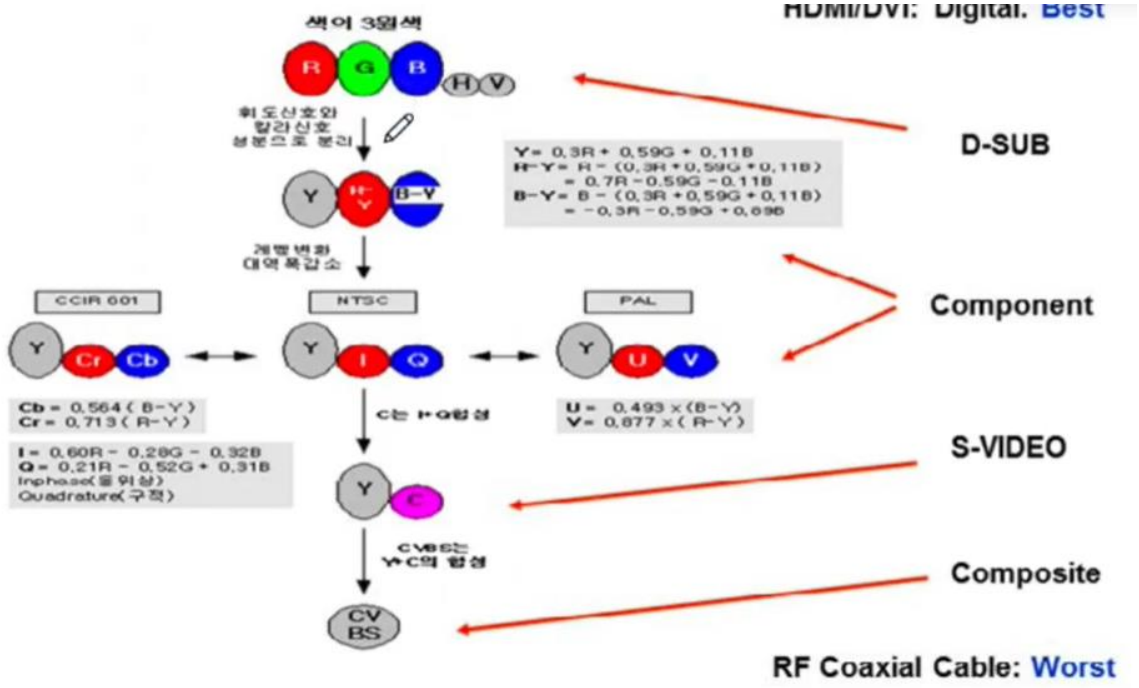
-Packetizatin

=>ATSC도 기존의 네트워크인 ATM (패킷조립화) 방식을 사용한다

-RF 신호 변조 기술 8VSB 사용

=>기본데이터는 디지털이지만 이것을 전송하기 위해서 아날로그 신호로 변조해줘야 하는데 8VSB와 같은 Modulation 방식을 쓴다.

Video signal format



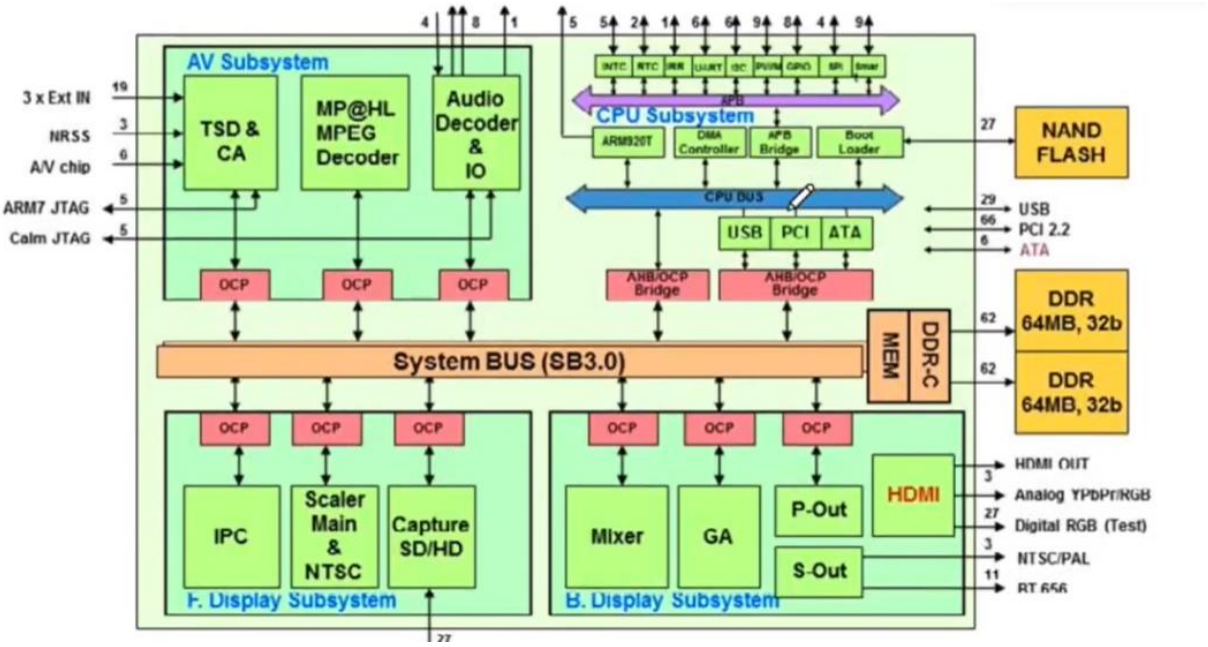
-사람 눈이 영상의 밝기 정도(휘도)에는 민감하고 색상 정도에는 민감하지 못하다.
-아날로그 방식 중 가장 좋은 방식은 R G B를 사용하는 D-SUB 방식이고 VGA Cable을 사용한다
(하지만 여기까지 아날로그 방식이다.)
-따라서, 가장 좋은 것이 R G B 이지만 아날로그 시절에는 이것을 처리하지 못해서 R B로 추출.
-이 두개의 칼라 정보 R B의 데이터를 또 줄였다 (사이즈가 줄어든거임)
=>이것이 Y cb cr, Y I Q, Y U V
(**R G B에서 2개를 뽑아내고 , 해당 색상 데이터를 줄이는 방식을 Component Video 라고 한다**)

위 2개의 색상 성분을 다시 1개의 성분으로 줄인 방식을 S-VIDEO 라고 한다.)
(DVD 에서 사용된 Video format 이고 데이터 포맷은 Y C , 동축케이블에 S-VIDEO Cable이 추가됨)

-S-VIDEO 보다 한단계 이전의 방식은 Y C 를 하나로 합친 CVBS 데이터 포맷이다
(휘도와 색상 정보가 하나고 동축케이블을 통해 아날로그 TV를 사용하던 시절이고, Composite 방식이라고 한다.)

아날로그 방식 : Composite, S-VIDEO, Component, D-SUB
디지털 방식 : HDMI/DVI
-HDMI/DVI 방식은 R G B 로 처리가 되고 아날로그가 아닌 디지털로 데이터가 처리가 되서 가장 좋은 방식이다.

D-TV SoC Block Diagram



-**AV Subsystem**
=>지상파 케이블 , 위성신호들이 들어온 것에 영상, 음성신호를 처리해 주는 부분
=>압축된 영상, 음성 신호들의 압축을 풀어준다
-**Cpu subsystem** : AV Subsystem 옆에 존재한다

-**Front-end Display subsystem**
=>압축해제된 영상, 음성신호들을 좀 더 개선해주는 역할을 한다.
=>**IPC** (Interace progressive convert) : Interaced으로 들어온 영상을 progressive 형식으로 뿌려준다
-즉, interaced 형태를 progressive로 변환시키는 알고리즘이 존재한다. (화질 개선)

=>**Scaler Main & NTSC** : 방송국에서 온 영상이 우리의 영상 화질에 꼭 맞는것은 아니라서 이것을 적절히 변환해준다 (**Display의 해상도에 맞게 적절히 변환**해 준다)
-스케일링 작업이라고 부른다.

=>**Capture** : AV Subsystem과 같이 영상의 전처리 과정을 TV의 셋업박스속에서 처리를 해줘서 **처리된 데이터를 그대로 넘겨받으면 되는 경우**도 존재한다.
-이런 것들을 처리해주는 곳이다.

-**Back-end Display subsystem**
=>**GA** : 리모컨으로 TV를 조작할 시 메뉴 GUI를 그래픽으로 화면에 입히는 역할을 한다.
=>**Mixer** : 기존의 방송국에서 뿌려주는 방송영상과 GA 기능을 조화롭게 결합시켜주는 역할을 한다.