

SoC ADC

Agenda

1. SoC 칩에서 사용되는 ADC이해와 실습
 - ADC 개요
 - ADC 모듈의 기본 구조
 - ADC 작동 원리
 - ADC 모듈의 주요 기능
 - ADC 트리거 메커니즘
 - ADC 하드웨어 구성
 - SAR ADC에 대한 이해
 - 실습 시뮬레이션
2. SoC 칩에서 사용되는 DAC이해
 - DAC 개요
 - DAC 모듈의 기본 구조
 - DAC 작동 원리

ADC 개요

ADC의 정의와 역할

- ADC(Analog-to-Digital Converter)는 아날로그 신호를 디지털 신호로 변환하는 시스템.
 - 예를 들어, 마이크가 포착한 소리나 디지털 카메라로 들어오는 빛을 디지털 데이터로 변환.
 - 디지털 출력은 일반적으로 입력 신호의 크기를 나타내는 2진수 형태로 제공

SoC에서 ADC의 중요성

- SoC(System-on-Chip)에서 ADC는 다양한 임베디드 애플리케이션에서 필수적인 구성 요소
- SoC에 통합된 ADC는 별도의 외부 ADC를 대체할 수 있으며, 시스템 크기를 줄이고 비용을 절감하는 데 도움.
- ADC는 센서 데이터 처리, 신호 분석, 통신 등 다양한 작업에서 중요한 역할

ADC 종류

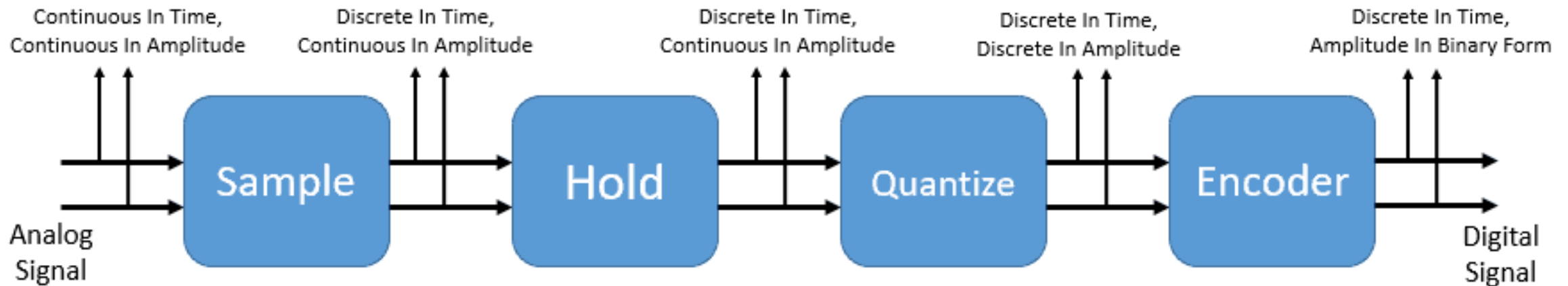
ADC 종류	장점	단점	주요 용도
Successive Approximation (SAR) ADC	<ul style="list-style-type: none">- 속도와 해상도의 균형이 우수함- 저전력 소모- 소형화 가능	<ul style="list-style-type: none">- 반(aliasing) 방지 기능 없음	데이터 수집, 센서 인터페이스
Delta-Sigma ($\Delta\Sigma$) ADC	<ul style="list-style-type: none">- 높은 동적 성능- 반(aliasing) 방지 기능 내장- 고해상도 가능	<ul style="list-style-type: none">- 비정상 신호에서 히스테리시스 발생 가능	오디오 처리, 소음 분석
Dual Slope ADC	<ul style="list-style-type: none">- 정확하고 저렴함- 노이즈에 강함	<ul style="list-style-type: none">- 속도가 느림	전압계, 정밀 측정
Pipelined ADC	<ul style="list-style-type: none">- 매우 빠른 변환 속도- 고속 신호 처리에 적합	<ul style="list-style-type: none">- 해상도가 제한적임	오실로스코프, 영상 처리
Flash ADC	<ul style="list-style-type: none">- 가장 빠른 변환 속도- 실시간 신호 처리 가능	<ul style="list-style-type: none">- 낮은 비트 해상도- 높은 전력 소모	고속 신호 처리

ADC 모듈의 기본 구조

- ADC 동작원리

- 아날로그 신호 → 샘플링 → 홀드 → 양자화 → 인코딩 → 디지털 출력

www.cselectricalandelectronics.com

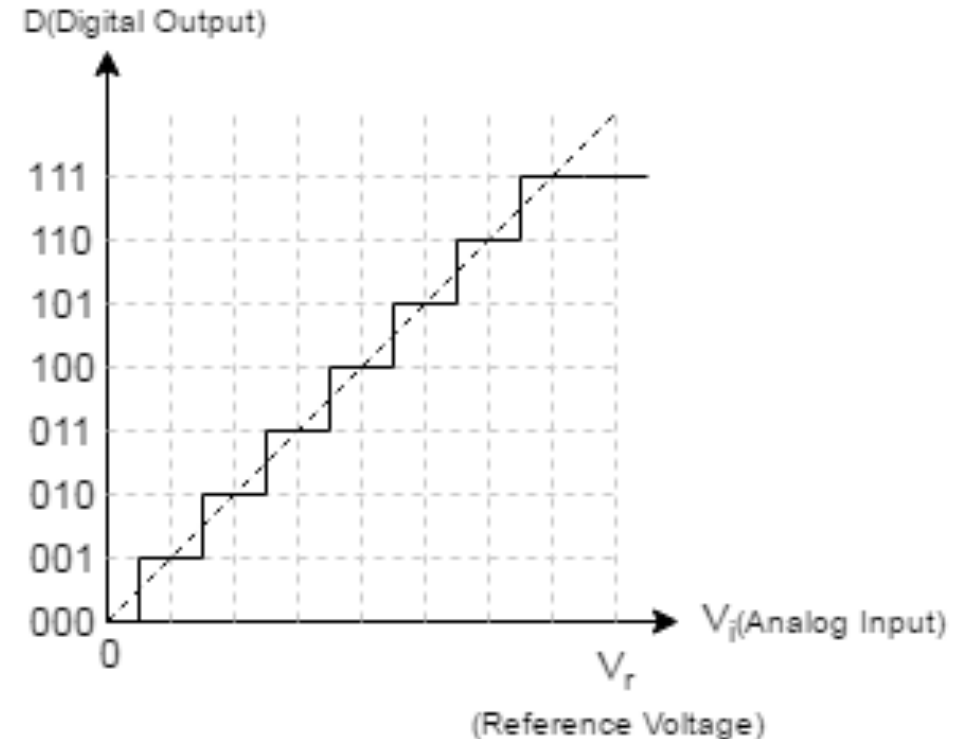


양자화, 범위, 분해능

- 양자화(Quantization)

- ADC의 아날로그 입력(V_i)과 디지털 출력(D)과의 상관관계 오른쪽 식과 같음
- V_r 은 레퍼런스 전압을 의미하고, n 은 디지털 출력의 비트수임
- 디지털 출력은 이진수로 표현할 수 있는 정수이고, 만약 n 비트 정수라면 그 범위는 0부터 (2^n-1) 까지임
- ADC는 아날로그 위의 식에 따라 아날로그 입력값을 반올림, 혹은 버림 등의 과정을 거쳐서 디지털 값으로 양자화 변환

$$D = \frac{V_i}{V_r} \times 2^n$$



양자화, 범위, 분해능

- 범위(Range)

- ADC에서 변환할 수 있는 아날로그 입력값에는 허용되는 범위
- 허용 범위내의 아날로그 입력만이 유효하며, 만약 이 값의 범위를 벗어나는 입력이 인가되는 경우에는 전기적으로 허용되는 범위내에서는 포화된 값(0 혹은 (2^n-1))을 갖게 됨

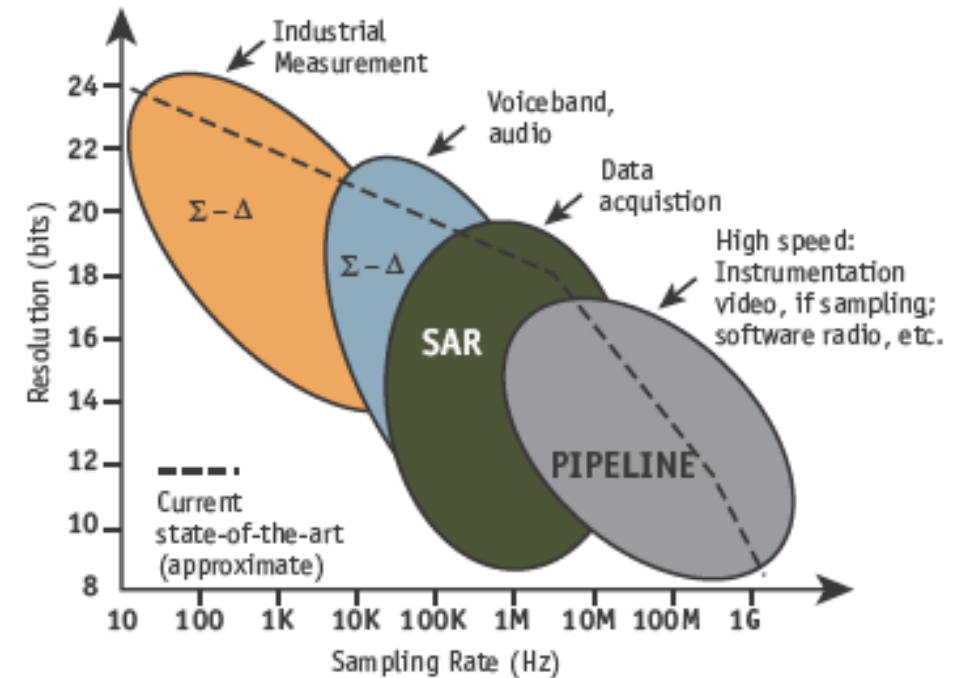
- 분해능(Resolution)

$$resolution = \frac{V_r}{2^n}$$

- ADC는 아날로그 값을 가장 가까운 디지털 값으로 근사화 시키는 과정
- 근사화 시키는 과정이므로 얼마나 정밀하게 아날로그 값을 근사하는 것인지가 중요한 지표로
- 이것을 분해능이라는 지표를 다음과 같이 정의하고 있다.
- 이 분해능이란 디지털 1비트의 정보가 아날로그 전압으로 얼마만큼을 나타내는지를 의미하는 것
- 고정밀도의 분해능을 얻고자 한다면 입력 범위를 나타내는 V_r 값이 작던지, 디지털로 변환하는 비트수 n 이 크면 된다.
- 대부분의 경우 V_r 은 3.3V 혹은 5V 의 값으로 고정되어 있으므로 높은 정밀도를 얻기 위해서는 비트수를 높여야 한다.
- 예를 들어 5V 범위의 신호를 가정하였을 경우 8-bit 의 경우 약 0.02V(0.4%) 의 분해능을, 10-bit 의 경우 0.005V(0.1%)의 분해능을 갖게 된다.

SAR(Successive Approximation Register) ADC란

- SAR ADC의 주요 특징
 - 고해상도와 정확성 제공(10~14비트 이상)
 - 이진 탐색 알고리즘 사용
 - 저전력 소비로 에너지 효율적
 - 단일 비교기 사용으로 간단한 구조
 - 센서 및 저속 응용 분야에 적합



ADC architectures, applications, resolution, and sampling rates.

SAR ADC의 주요 구성 요소

1. CDAC (Charge Redistribution DAC)

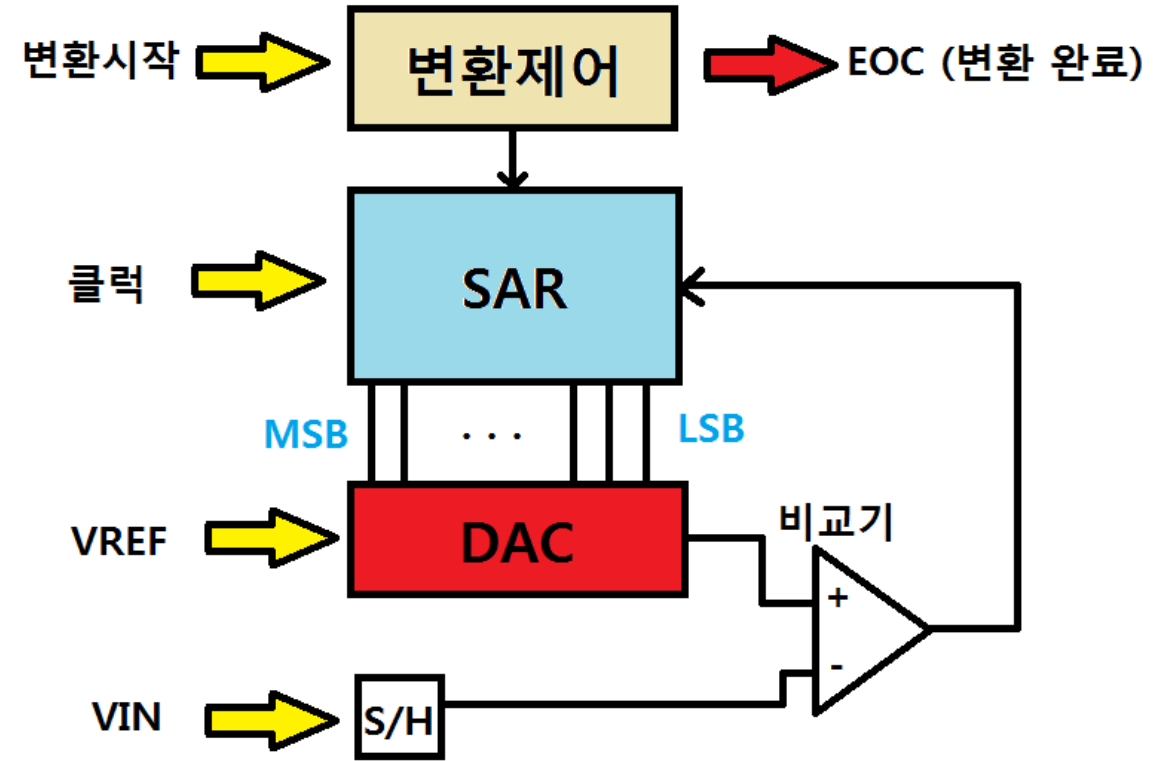
- 입력 전압을 샘플링하고 비교기 입력으로 전달

2. Comparator

- 입력 전압(V_{in})과 DAC 출력 전압(V_{dac})을 비교

3. SAR Logic

- 비교 결과를 바탕으로 디지털 값을 결정



Basic successive-approximation (SAR) ADC

SAR ADC의 주요 구성 요소

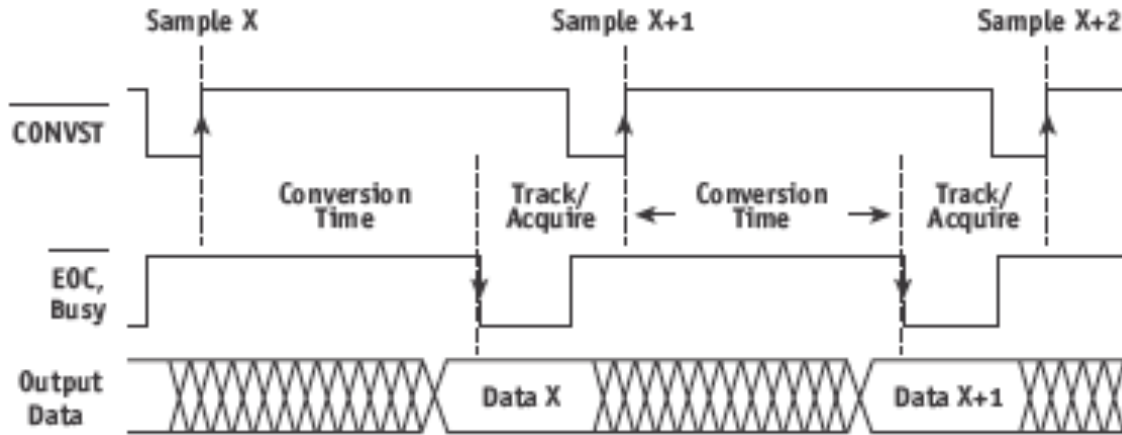


Fig1 Simplified timing diagram of a SAR A/D converter.

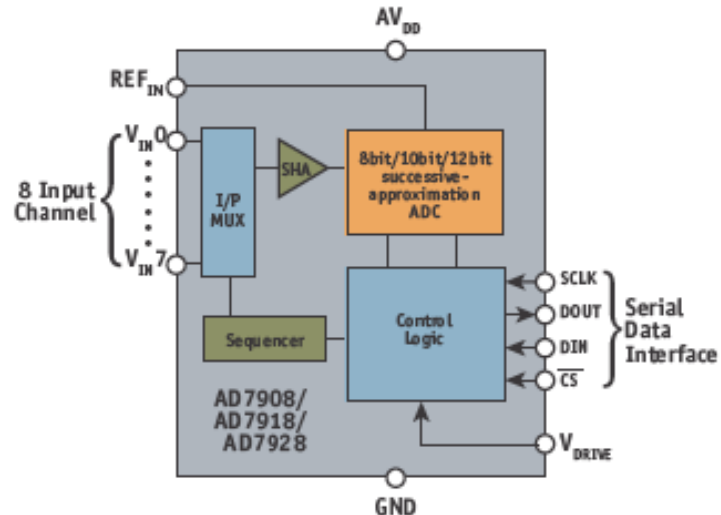


Fig3 Functional block diagram of a modern 1-MSPS SAR ADC with 8-channel input multiplexer

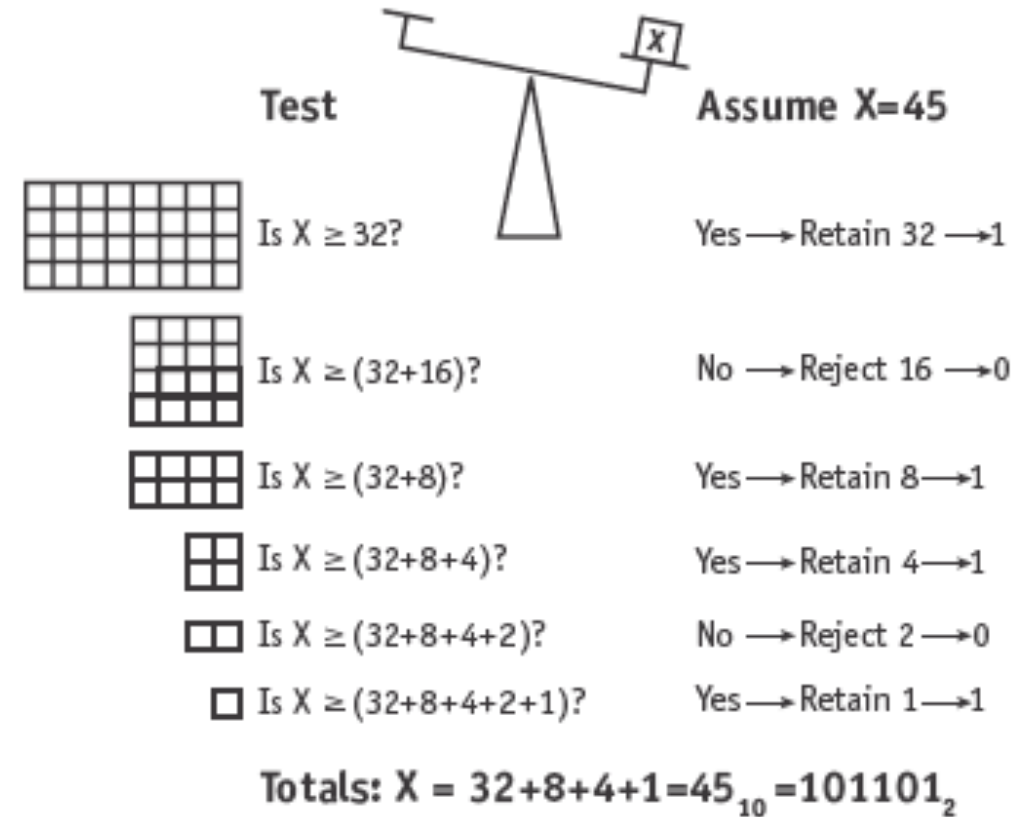
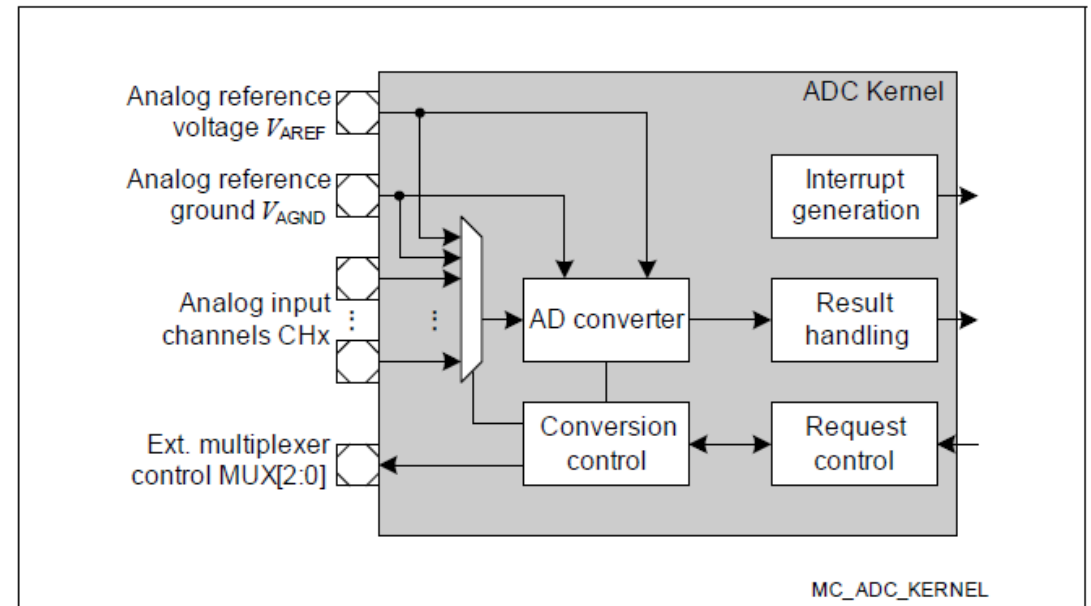
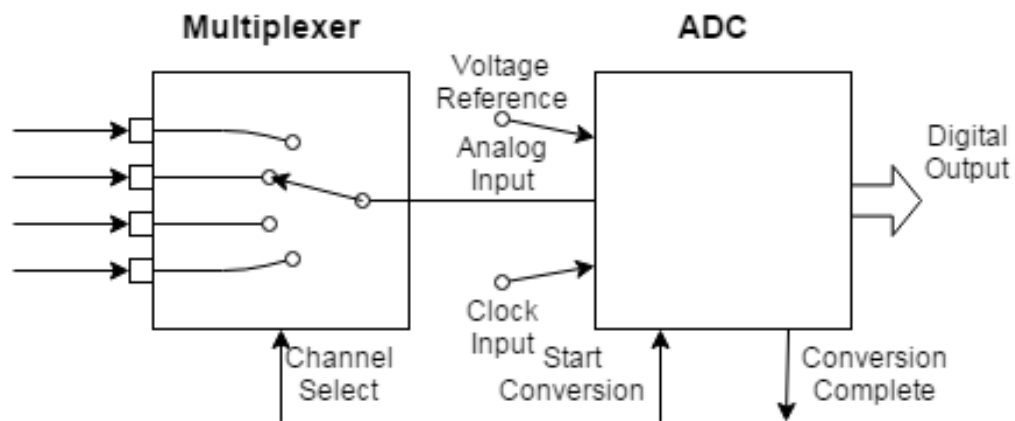


Fig2 Successive-approximation ADC algorithm using balance scale and binary weights.

ADC 하드웨어 구성

일반적인 ADC 모듈 구조

- 한 개의 ADC와 여러 아날로그 입력 채널을 변환하기 위해 **아날로그 멀티플렉서(Analog MUX)**를 사용한다.
- 복수개의 ADC를 사용하는 것보다 한 개의 ADC와 MUX를 활용하는 것이 경제적이다.
- ADC는 고정밀 아날로그 회로로 구성되며 CPU와 인터페이스하기 위해 복잡한 구조를 갖는다.



실습1 - ADC 시뮬레이션

- 코드를 완성 하시오



```
1 timescale 1ns / 1ps
2
3 module adc_emulation (
4     input clk,
5     input reset,
6     input start_of_conversion,
7     input [11:0] analog_input, // 가상의 아날로그 입력
8     output reg [11:0] digital_output,
9     output reg end_of_conversion
10 );
11     reg [3:0] state;
12     localparam IDLE = 0, SAMPLING = 1, CONVERSION = 2, OUTPUT = 3;
13
14     always @(posedge clk or posedge reset) begin
15         if (reset) begin
16             state <= IDLE;
17             end_of_conversion <= 0;
18         end else begin
19             case (state)
20                 IDLE: if (start_of_conversion) state <= SAMPLING;
21                 SAMPLING: state <= CONVERSION; // 샘플링 완료 후 변환 단계로 이동
22                 CONVERSION: begin
23                     digital_output <= analog_input; // 단순히 입력을 복사 (에뮬레이션)
24                     state <= OUTPUT;
25                 end
26                 OUTPUT: begin
27                     end_of_conversion <= 1;
28                     state <= IDLE;
29                 end
30             endcase
31         end
32     end
33 endmodule
```

실습1 - ADC 시뮬레이션

- 시뮬레이션 테스트 벤치 코드를 완성 하시오
- 다음장 이어서

```
1  `timescale 1ns / 1ps
2  module tb_adc_emulation;
3      // Testbench 내부 신호 선언
4      reg clk;
5      reg reset;
6      reg start_of_conversion;
7      reg [11:0] analog_input; // 가상의 아날로그 입력
8      wire [11:0] digital_output;
9      wire end_of_conversion;
10
11     // 파라미터 설정
12     parameter NUM_POINTS = 100; // 사인파 데이터 포인트 개수
13     parameter AMPLITUDE = 4095; // 최대 진폭 (12비트 ADC 최대값)
14     parameter PI = 3.141592653589793;
15
16     // 사인파 데이터를 저장할 배열
17     real sine_wave_real[0:NUM_POINTS-1];
18     integer sine_wave_int[0:NUM_POINTS-1];
19
20     // DUT 인스턴스화
21     adc_emulation dut (
22         .clk(clk),
23         .reset(reset),
24         .start_of_conversion(start_of_conversion),
25         .analog_input(analog_input),
26         .digital_output(digital_output),
27         .end_of_conversion(end_of_conversion)
28     );
29     // 클럭 생성 (50MHz)
30     initial begin
31         clk = 0;
32         forever #10 clk = ~clk; // 20ns 주기 (50MHz)
33     end
```

실습1 – ADC 시뮬레이션

```
35 // 사인파 데이터 생성
36 integer i;
37 real angle;
38 initial begin
39     for ( i = 0; i < NUM_POINTS; i=i+1) begin
40         angle = (2 * PI * i) / NUM_POINTS; // 각도 계산 (라디안 단위)
41         sine_wave_real[i] = $sin(angle); // 실수형 사인파 값 계산
42         sine_wave_int[i] = $rtoi((sine_wave_real[i] + 1) * (AMPLITUDE / 2));
43         // 정수형으로 변환 (0~4095 범위로 스케일링)
44     end
45 end
46 // 초기 조건 설정 및 시뮬레이션 진행
47 initial begin
48     // 초기화
49     reset = 1;
50     start_of_conversion = 0;
51     analog_input = 12'b0;
52     // 리셋 해제
53     #50 reset = 0;
54     // 사인파 데이터를 순차적으로 입력
55     for ( i = 0; i < NUM_POINTS; i = i+1) begin
56         #20 start_of_conversion = 1;
57         analog_input = sine_wave_int[i]; // 사인파 데이터 입력
58         #20 start_of_conversion = 0;
59         #100; // 변환 대기 시간
60     end
61     // 시뮬레이션 종료
62     #200 $stop;
63 end
64 // 출력 모니터링
65 initial begin
66     $monitor("Time=%0t | Analog Input=%d | Digital Output=%d | End of Conversion=%b",
67             $time, analog_input, digital_output, end_of_conversion);
68 end
69 endmodule
```


실습2 – 일반적인 ADC Controller

- 코드를 완성 하시오



```
1  `timescale 1ns / 1ps
2  module adc_controller (
3      input wire clk,          // Clock signal
4      input wire go,          // Start conversion signal
5      input wire cmp,          // Comparator output
6      output reg valid,        // Conversion complete signal
7      output reg [7:0] result, // 8-bit digital output
8      output wire sample,      // Sample and hold control signal
9      output wire [7:0] value  // Current DAC value
10 );
11
12 // State encoding for the state machine
13 parameter sWait = 2'b00, sSample = 2'b01, sConv = 2'b10, sDone = 2'b11;
14
15 reg [1:0] state;          // Current state
16 reg [7:0] mask;          // Mask for binary search
17
18 // State machine logic (synchronous)
19 always @(posedge clk) begin
20     if (!go) begin
21         state <= sWait;      // Reset to waiting state if go=0
22         valid <= 0;         // Clear valid flag
23     end else begin
24         case (state)
25             sWait: begin      // Waiting for start signal
26                 state <= sSample;
27             end
28
29             sSample: begin    // Sampling the input signal
30                 state <= sConv;
31                 mask <= 8'b10000000; // Initialize mask to MSB
32                 result <= 8'b0;      // Clear result register
33             end
```


실습2 – 일반적인 ADC Controller

- 코드를 완성 하시오



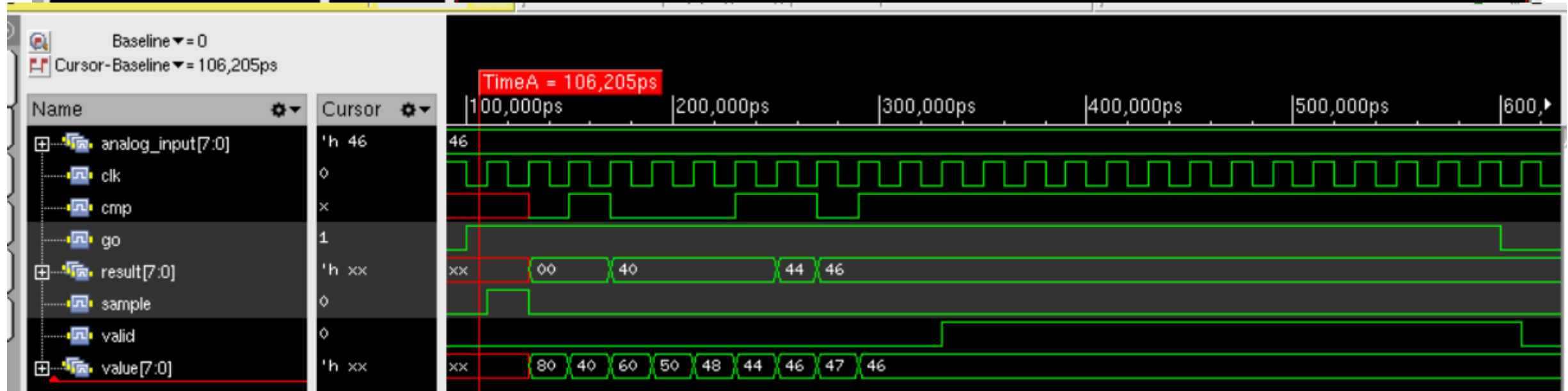
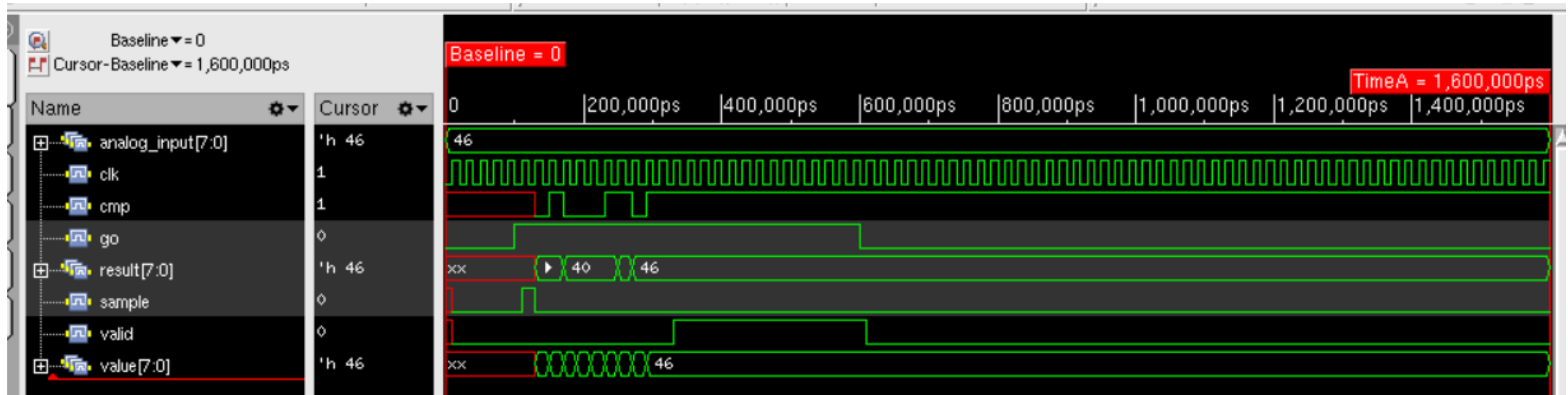
```
35  ∨      sConv: begin      // Conversion process (binary search)
36  ∨          if (cmp)
37          |      result <= result | mask; // Set bit if comparator indicates larger input
38
39          |      mask <= mask >> 1;      // Shift mask to next bit
40
41  ∨          if (mask == 0)
42          |      state <= sDone;          // Move to done state if all bits are processed
43          end
44
45  ∨      sDone: begin      // Conversion complete
46          |      valid <= 1;          // Set valid flag to indicate completion
47  ∨          if (!go)
48          |      state <= sWait;        // Return to wait state if go=0
49          end
50
51          default: state <= sWait;      // Default to wait state for safety
52      endcase
53  end
54  end
55
56  // Output assignments for sample and value signals
57  assign sample = (state == sSample);    // Sample signal is high during sampling state
58  assign value = result | mask;          // Current DAC value during conversion
59
60  endmodule
```

실습2 – 일반적인 ADC Controller

- 시뮬레이션 테스트 벤치 코드를 완성 하시오

```
1 timescale 1ns / 1ps
2 module testbench();
3     reg clk, go;
4     wire valid, sample, cmp;
5     wire [7:0] result;
6     wire [7:0] value;
7
8     reg [7:0] analog_input = 8'b01000110; // Example analog input
9
10    adc_controller uut (
11        .clk(clk),
12        .go(go),
13        .valid(valid),
14        .result(result),
15        .sample(sample),
16        .value(value),
17        .cmp(cmp)
18    );
19
20    always #10 clk = ~clk; // Generate clock with period of 20 time units
21
22    initial begin
23        clk = 0;
24        go = 0;
25
26        #100 go = 1; // Start conversion after 100 time units
27
28        #500 go = 0; // Stop conversion after some time
29
30        #1000 $stop; // End simulation after sufficient time
31    end
32
33    assign cmp = (analog_input >= value); // Simulate comparator behavior
34
35 endmodule
```

실습2 - 일반적인ADC Controller 시뮬레이션 결과1

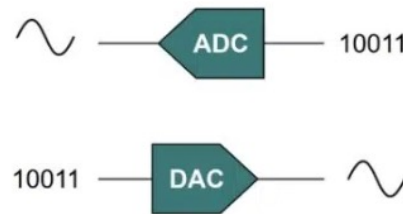
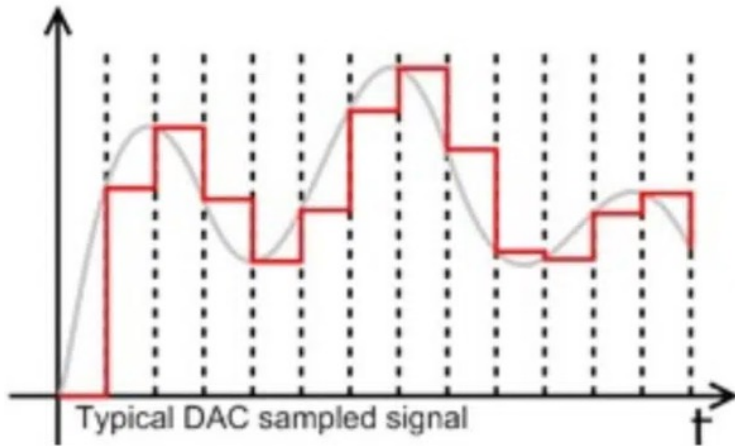


SoC DAC

DAC 개요

DAC란 무엇인가?

- 디지털 데이터를 아날로그 신호로 변환하는 장치
- 디지털 신호: 이산적이고 단계적인 데이터 (0과 1)
- 아날로그 신호: 연속적인 전압 또는 전류



항목	ADC (Analog-to-Digital Converter)	DAC (Digital-to-Analog Converter)
기능	아날로그 신호를 디지털 신호로 변환	디지털 신호를 아날로그 신호로 변환
입력	연속적인 아날로그 신호 (전압, 전류 등)	이산적인 디지털 데이터 (이진 코드)
출력	이산적인 디지털 데이터 (이진 코드)	연속적인 아날로그 신호 (전압 또는 전류)
사용 예시	센서 데이터 처리, 오디오 녹음	오디오 재생, 영상 출력
변환 과정	샘플링 → 양자화 → 부호화	디지털 값에 따라 가중치를 부여한 후 아날로그 신호 생성
주요 기술	Successive Approximation, Sigma-Delta 등	Weighted Resistor 방식, R-2R Ladder 방식

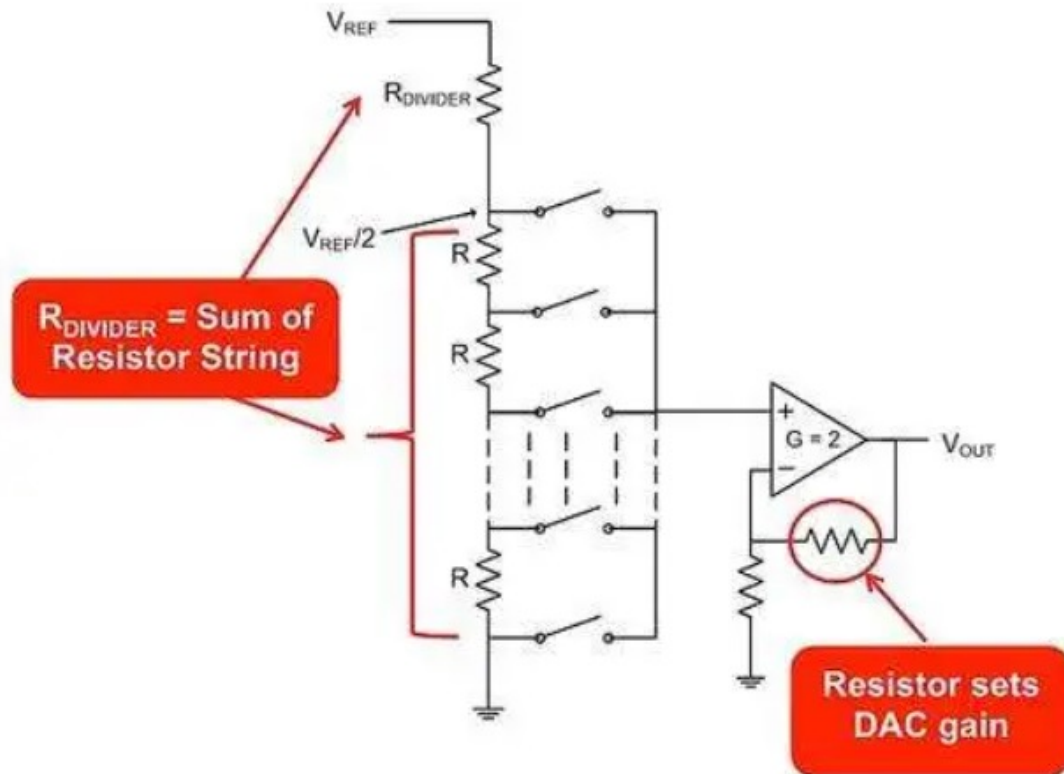
DAC 종류

DAC 종류	원리	장점	단점	주요 응용 분야
이진 가중치 방식	디지털 입력 비트에 가중치를 부여하여 출력 전압/전류 생성	설계가 간단하고 빠름	고해상도에서 정확도 유지 어려움	일반적인 신호 변환
R-2R 래더 방식	R, 2R 저항 네트워크를 사용하여 디지털 입력을 아날로그로 변환	경제적이고 안정적	상대적으로 느림	오디오 및 영상 처리
델타-시그마 방식	오차를 누적하여 보정하며 고해상도 아날로그 출력 생성	높은 정밀도와 낮은 왜곡	낮은 샘플링 속도	오디오, 정밀 측정
세그먼트 방식	MSB는 Thermometer 코드, LSB는 Binary Weighted 방식을 결합	속도와 정확성 간 균형 제공	설계가 복잡하고 비용이 높음	고속 및 고정밀 응용
PWM 기반 방식	디지털 신호의 펄스 폭을 조절하여 아날로그 출력 생성	간단하고 경제적	낮은 정확도와 느린 응답 속도	LED 제어, 모터 속도 제어
온도계 코드 방식	각 출력 상태에 대해 고유의 스위치 네트워크를 사용하여 아날로그 값 생성	매우 빠르고 정확	많은 구성 요소 필요	고속 ADC와 함께 사용
하이브리드 방식	여러 DAC 방식을 조합하여 설계	다양한 요구 충족 가능	설계 복잡	상용 칩, 다목적 응용

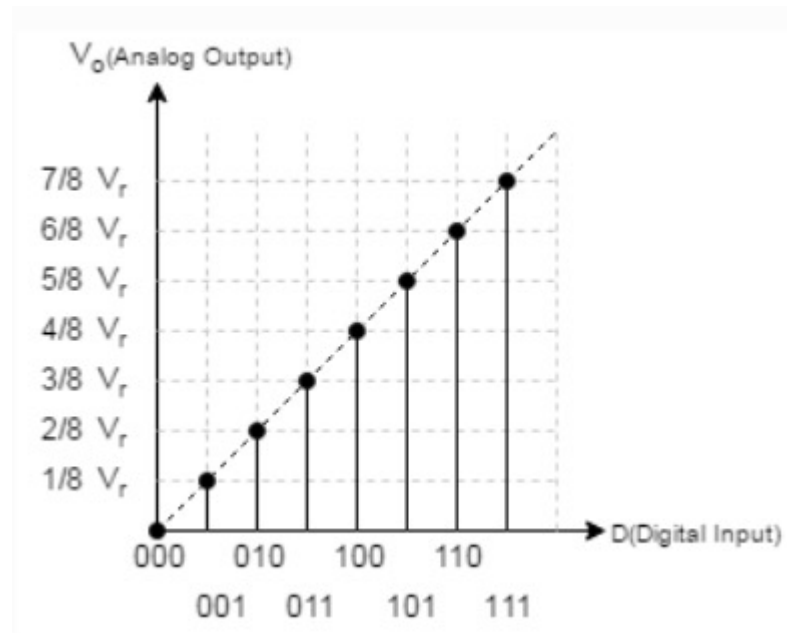
DAC 모듈의 기본 구조

- DAC 동작원리

- 저항 네트워크 → 스위치 → 출력 증폭기 → 샘플링 및 유지 회로 → 전류 소스 또는 전압 소스




$$V_o = \frac{D}{2^n} \times V_r$$



실습2 – R-2R 래더 DAC

- 코드를 완성 하시오



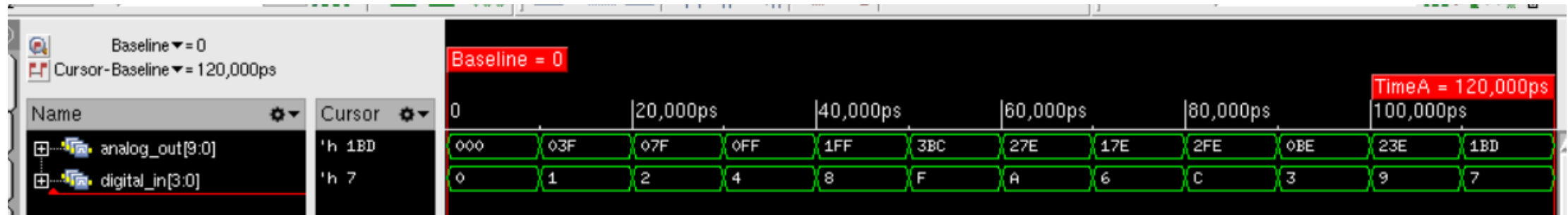
```
1  `timescale 1ns / 1ps
2  module r2r_dac (
3      input [3:0] digital_in, // 4비트 디지털 입력
4      output reg [9:0] analog_out // 아날로그 출력 (10비트 정밀도)
5  );
6
7      // 참조 전압 설정 (예: 5V)
8      parameter VREF = 1023; // 10비트 정밀도에서 최대값 (5V를 1023으로 표현)
9
10     always @(*) begin
11         // 아날로그 출력 계산
12         analog_out = (digital_in[3] * VREF / 2) +
13                       (digital_in[2] * VREF / 4) +
14                       (digital_in[1] * VREF / 8) +
15                       (digital_in[0] * VREF / 16);
16     end
17
18 endmodule
```


실습2 – R-2R 래더 DAC

- 시뮬레이션 테스트 벤치 코드를 완성 하시오

```
1  `timescale 1ns / 1ps
2  module tb_r2r_dac;
3      reg [3:0] digital_in;      // 4비트 디지털 입력
4      wire [9:0] analog_out;    // 10비트 아날로그 출력
5
6      // DUT (Device Under Test) 인스턴스화
7      r2r_dac uut (
8          .digital_in(digital_in),
9          .analog_out(analog_out)
10     );
11
12     initial begin
13         // 테스트 시작
14         $display("Time\tDigital Input\tAnalog Output");
15         $monitor("%0t\t%b\t\t%d", $time, digital_in, analog_out);
16
17         // 테스트 벡터
18         digital_in = 4'b0000; #10; // 최소값 (0V 예상)
19         digital_in = 4'b0001; #10; // LSB 활성화 (~VREF/16 예상)
20         digital_in = 4'b0010; #10; // 두 번째 비트 활성화 (~VREF/8 예상)
21         digital_in = 4'b0100; #10; // 세 번째 비트 활성화 (~VREF/4 예상)
22         digital_in = 4'b1000; #10; // MSB 활성화 (~VREF/2 예상)
23
24         digital_in = 4'b1111; #10; // 최대값 (VREF 예상)
25         digital_in = 4'b1010; #10; // 특정 조합 (~VREF/2 + VREF/8 예상)
26         digital_in = 4'b0110; #10; // 특정 조합 (~VREF/4 + VREF/8 예상)
27
28         digital_in = 4'b1100; #10; // 상위 비트 활성화 (~VREF/2 + VREF/4 예상)
29         digital_in = 4'b0011; #10; // 하위 비트 활성화 (~VREF/8 + VREF/16 예상)
30
31         digital_in = 4'b1001; #10; // MSB와 LSB 활성화 (~VREF/2 + VREF/16 예상)
32         digital_in = 4'b0111; #10; // 하위 세 비트 활성화 (~VREF/4 + VREF/8 + VREF/16 예상)
33
34         $stop;
35     end
36 endmodule
```

실습2 - R-2R 래더 DAC



디지털 입력 (digital_in)	아날로그 출력 (analog_out)	설명
4'b0000	0	모든 비트가 비활성화되어 최소값(0V)이 출력됩니다.
4'b0001	511.5	LSB 활성화로 참조 전압의 VREF/16에 해당하는 값이 출력됩니다.
4'b0010	255.75	두 번째 비트 활성화로 VREF/8V에 해당하는 값이 출력됩니다.
4'b0100	127.875	세 번째 비트 활성화로 VREF/4V에 해당하는 값이 출력됩니다.
4'b1000	63.9375	MSB 활성화로 VREF/2V에 해당하는 값이 출력됩니다.
4'b1111	959.0625	모든 비트가 활성화되어 최대값(VREF)이 출력됩니다.
4'b1010	319.6875	MSB와 두 번째 비트 활성화로 VREF/2+VREF/8V에 해당하는 값입니다.
4'b0110	383.625	세 번째와 두 번째 비트 활성화로 VREF/4+VREF/8V에 해당하는 값입니다.
4'b1100	191.8125	MSB와 세 번째 비트 활성화로 VREF/2+VREF/4V에 해당하는 값입니다.
4'b0011	767.25	두 번째와 LSB 활성화로 VREF/8+VREF/16V에 해당하는 값입니다.
4'b1001	575.4375	MSB와 LSB 활성화로 VREF/2+VREF/16V에 해당하는 값입니다.
4'b0111	895.125	하위 세 비트 활성화로 VREF/4+VREF/8+VREF/16에 해당하는 값입니다.

Thanks