피지컬 컴퓨팅

Lec. 8. Analog to Digital Conversion (ADC)

Heenam Yoon

Department of
Human-Centered Artificial Intelligence

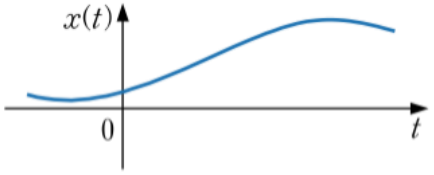
E-mail) h-yoon@smu.ac.kr
Room) 0112

Analog to Digital Conversion (ADC)

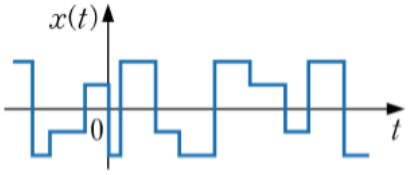
신호의 분류

- 연속신호 (continuous time signal): 시간에 대해 끊어지지 않고 지속적으로 나타나는 신호
- 이산신호 (discrete time signal): 특정한 시각에서만 정의되는 신호

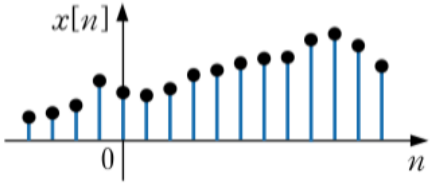
- 아날로그 신호 (analog signal): 시간과 크기가 모두 연속적인 값을 가지는 신호
- 디지털 신호 (digital signal): 시간과 크기가 모두 이산적인 값을 가지는 신호



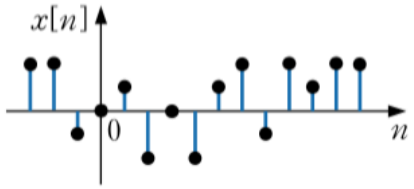
(a) 아날로그 신호



(b) 연속 시간 이산 크기 신호



(c) 이산 시간 연속 크기 신호

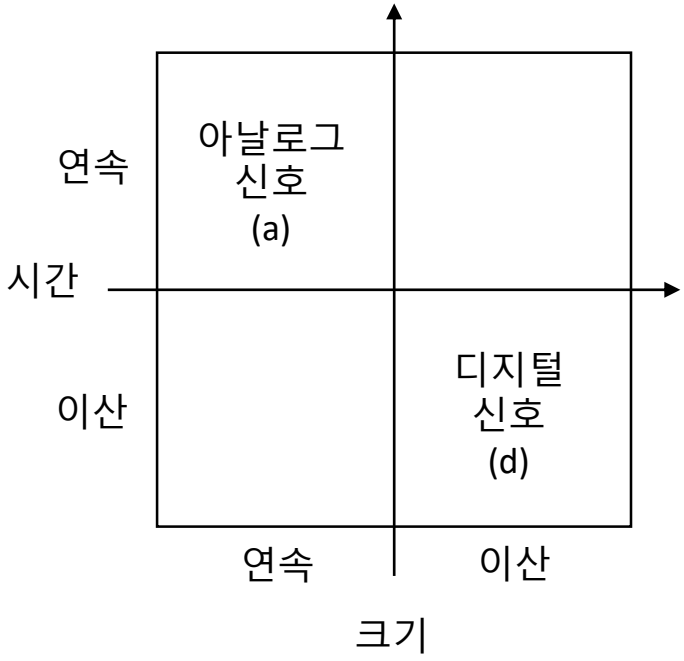


(d) 디지털 신호

Analog to Digital Conversion (ADC)

신호의 분류

- 아날로그 신호: 연속시간, 연속크기 신호
- 디지털 신호: 이산시간, 이산크기 신호
  - 자연적으로 존재하는 신호가 아니라 아날로그 신호로부터 인위적인 작업을 통해 만들어짐



Analog to Digital Conversion (ADC)

디지털 신호의 생성방법

- 아날로그-디지털 변환기(analog-to-digital converter: ADC)

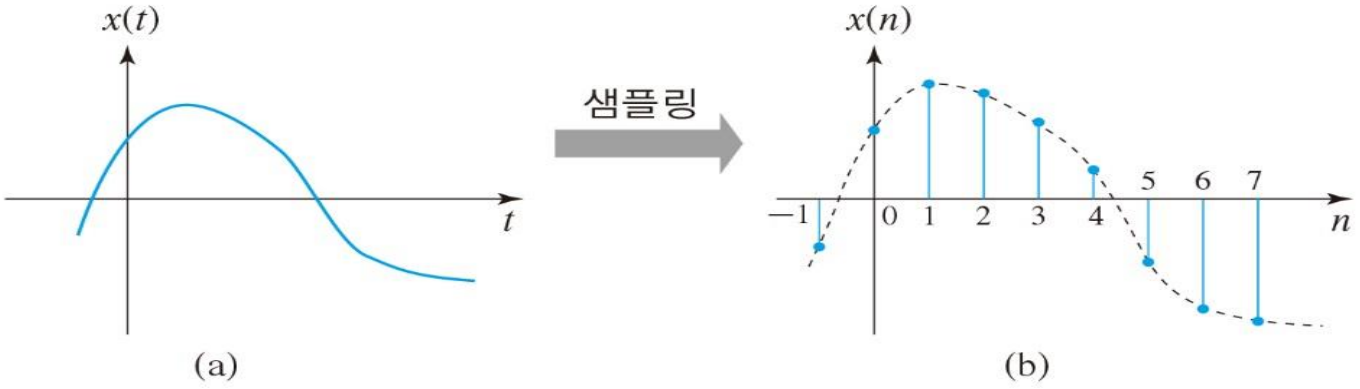
- 샘플링 (sampling) : 시간에 대해 이산화하는 과정
- 양자화 (quantization): 크기에 대해 이산화하는 과정
- 부호화 (coding): 각 양자화 구간에 하나의 이진수를 대응시키는 과정

Analog to Digital Converter (ADC)

Analog to Digital Conversion (ADC)

디지털 신호의 생성방법

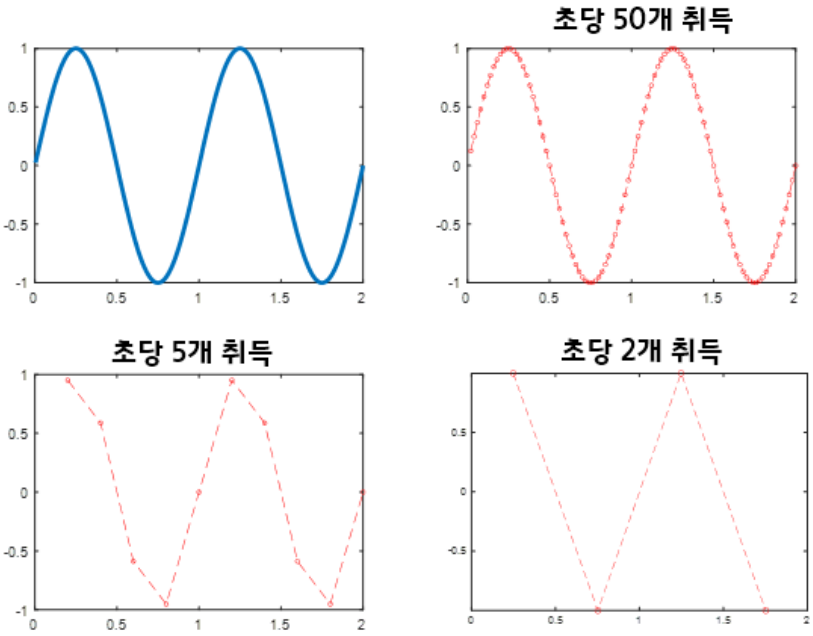
- 샘플링 (Sampling)
  - 일정한 간격을 가지고 규칙적으로 연속 신호의 샘플을 취함으로써 연속신호를 이산 신호로 변환



Analog to Digital Conversion (ADC)

디지털 신호의 생성방법

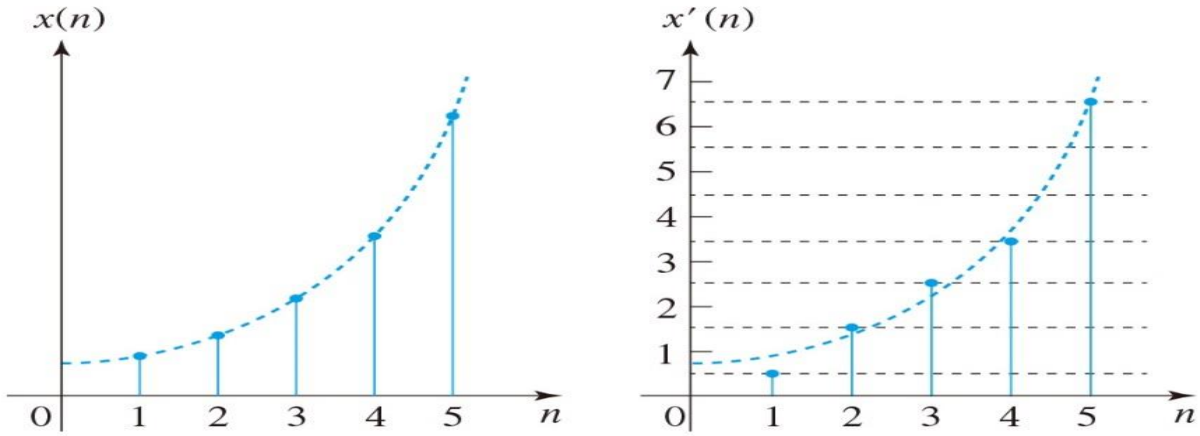
- Nyquist Sampling Theorem
  - 원신호의 최대 주파수의 2배 이상으로 취득해야 함



Analog to Digital Conversion (ADC)

디지털 신호의 생성방법

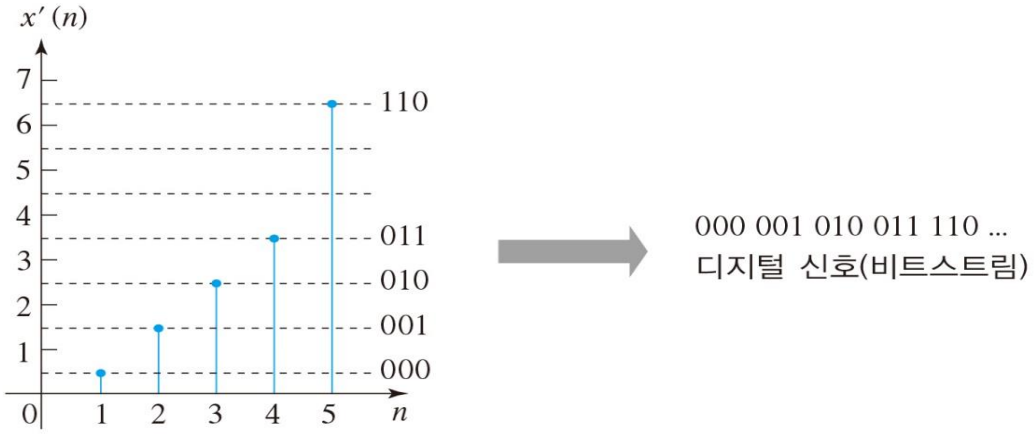
- 양자화 (Quantization)
  - 이산 신호를 이산 값을 갖도록 변환하는 과정
  - 양자화를 거치면 신호가 가질 수 있는 값의 개수는 유한하게 됨
  - 신호 값의 전체 범위는 몇 개의 구간(레벨)으로 구분하는가에 따라 결정됨



Analog to Digital Conversion (ADC)

디지털 신호의 생성방법

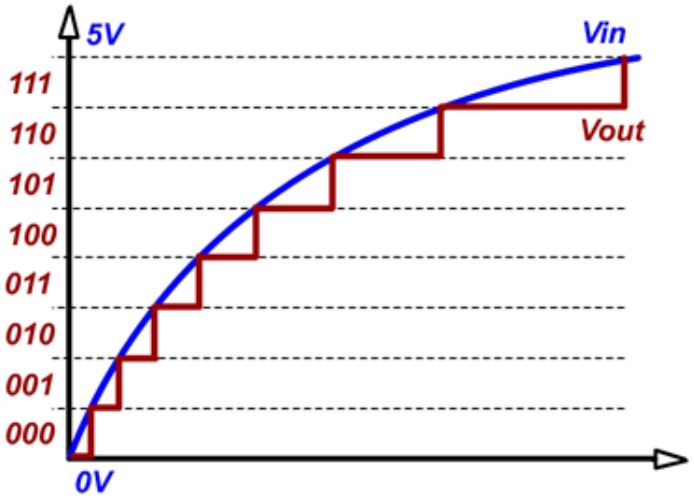
- 부호화 (Coding)
  - '0'과 '1'로 즉 이진수로 표현되는 디지털 신호
  - 부호화 과정은 각 양자화 구간에 하나의 이진수를 대응시킴
  - 신호의 값을 전체 범위 L개의 구간으로 나눈다면 L개의 서로 다른 이진수가 필요



Analog to Digital Conversion (ADC)

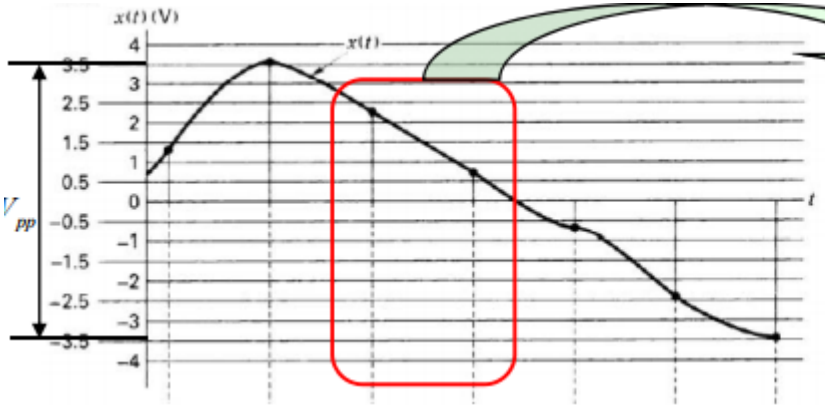
디지털 신호의 생성방법

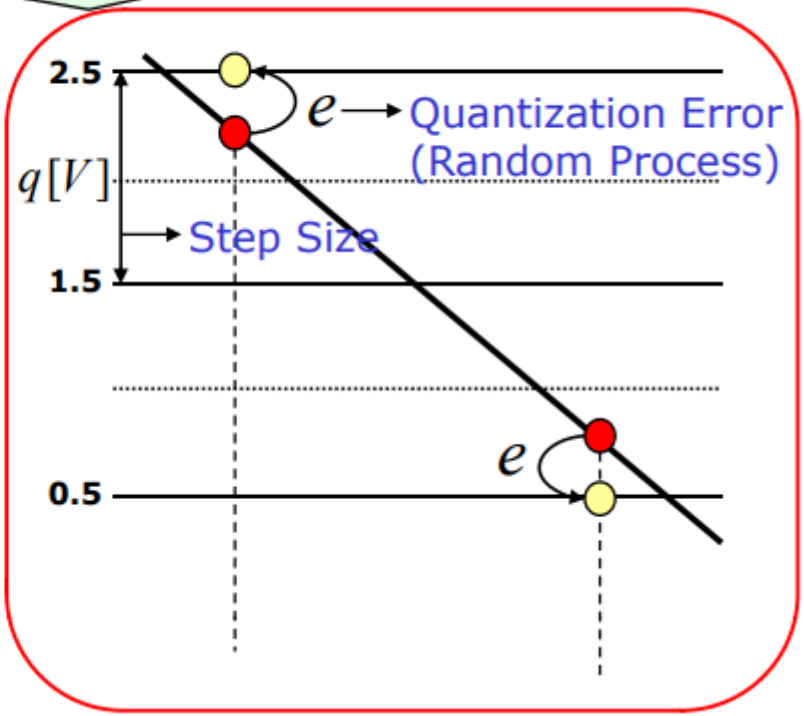
- 양자화 오차
  - 양자화 레벨 값과 실제 신호 값의 차이
  - 양자화시 근사에 의해 발생하는 오차
  - 유한개의 비트로 연속 값을 표현하여 오차 발생
  - 디지털 신호처리 시스템의 성능 저하 및 작동 불능 원인이 될 수 있음

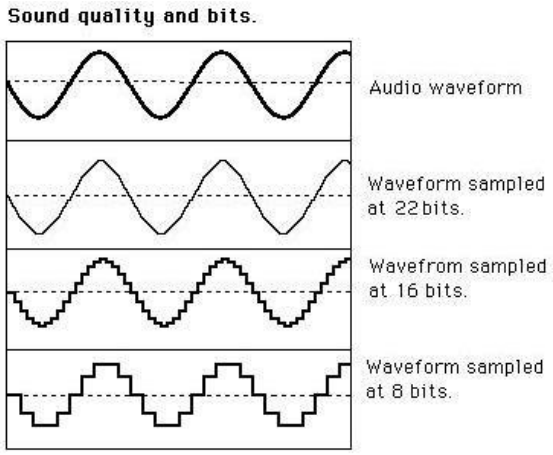


Analog to Digital Conversion (ADC)

From 2주차 수업

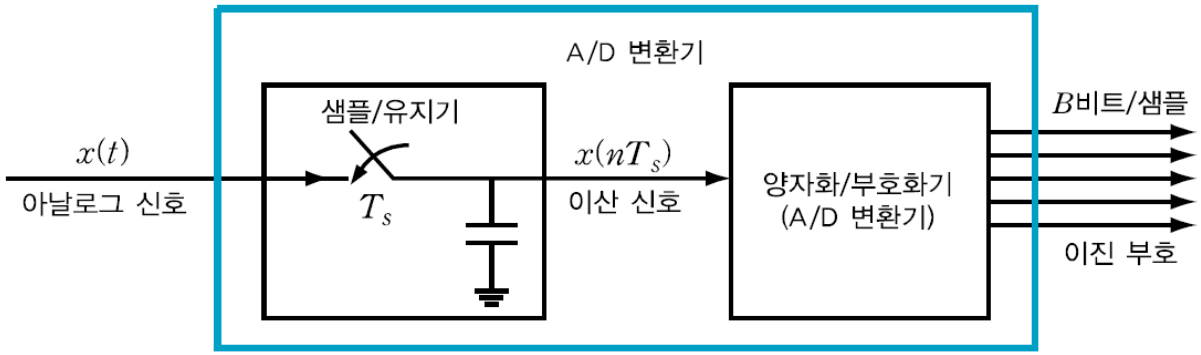


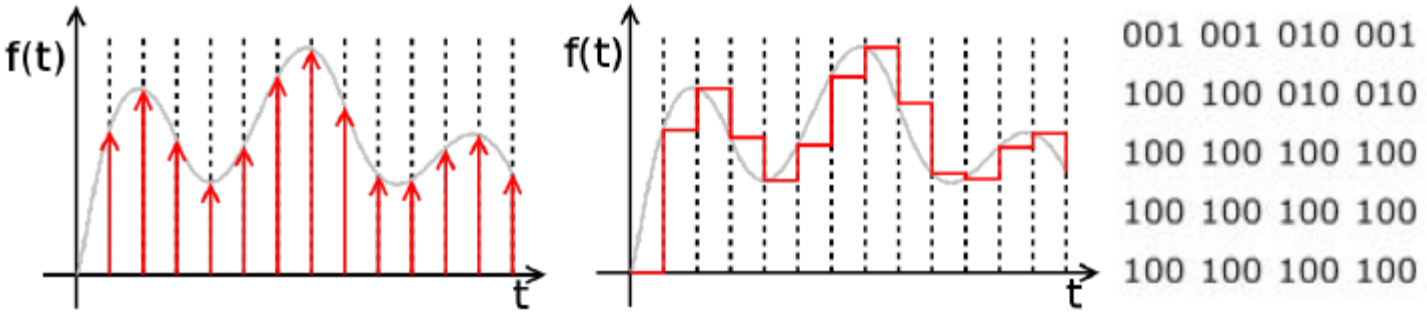




Analog to Digital Conversion (ADC)

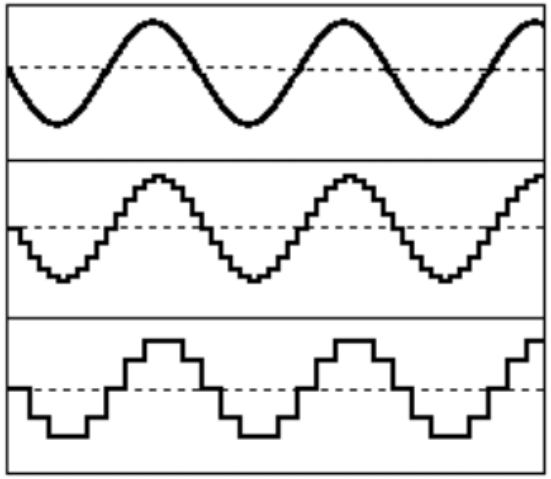
ADC Diagram

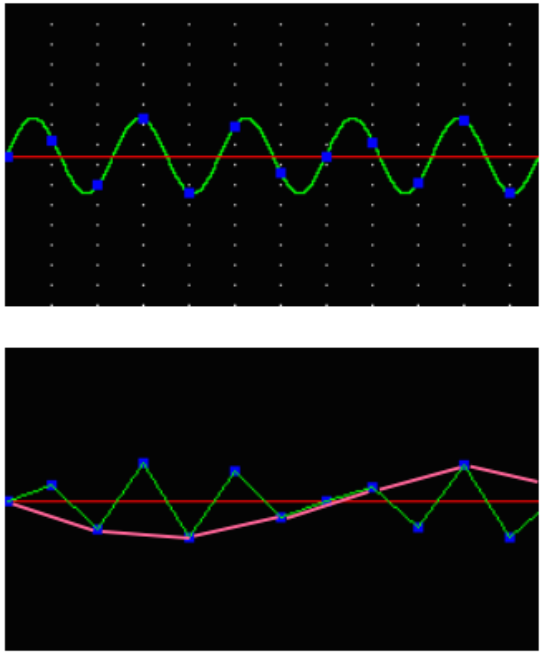




Analog to Digital Conversion (ADC)

참고





크기 축에서 성기게 Quantization하면 신호가 엉성?해진다

시간 축에서 느리게 Sampling하면 신호가 왜곡된다

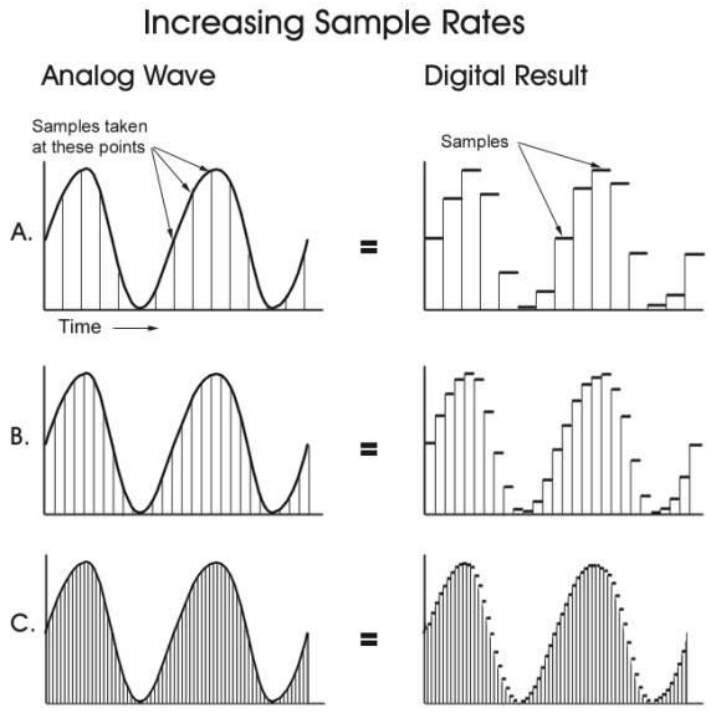
Analog to Digital Conversion (ADC)

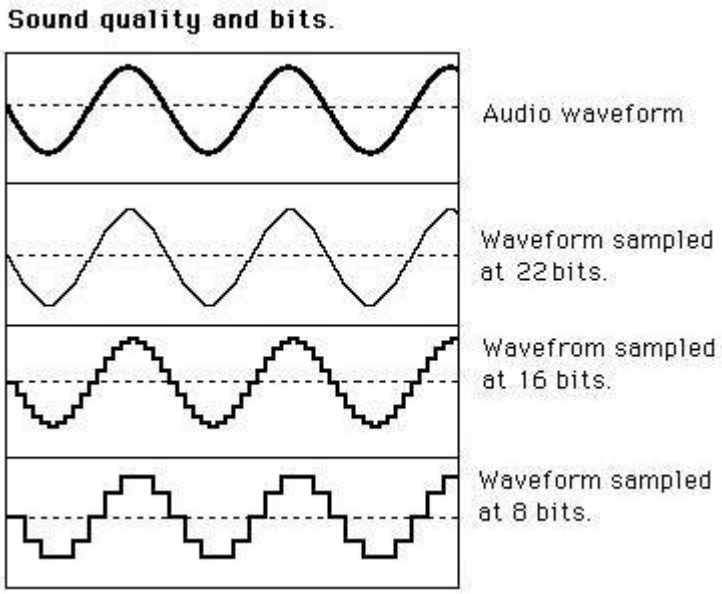
참고

- **음악파일: 44.1kHz, 16bits (2^{16}, 2bytes), 2 ch (스테레오), 240 sec (시간)**

  가청주파수*2 + 10% 0~65,535

  Sampling과 관련 Quantization과 관련





Analog to Digital Conversion (ADC)

참고

Analog to Digital Conversion (ADC)

STM32 ADC

- Successive approximation analog-to-digital converter
- 12-bit resolution
- Interrupt generation at the end of conversion, end of injected conversion, and in case of analog watchdog or overrun events
- Single and continuous conversion modes
- Scan mode for automatic conversion of channel 0 to channel 'n'
- Data alignment with in-built data coherency
- ADC supply requirements: 2.4 V to 3.6 V at full speed and down to 1.8 V at slower speed
- ADC input range: VREF- ≤ VIN ≤ VREF+

오늘의 실습

출력 전압을 ADC하여, FND에 출력한다

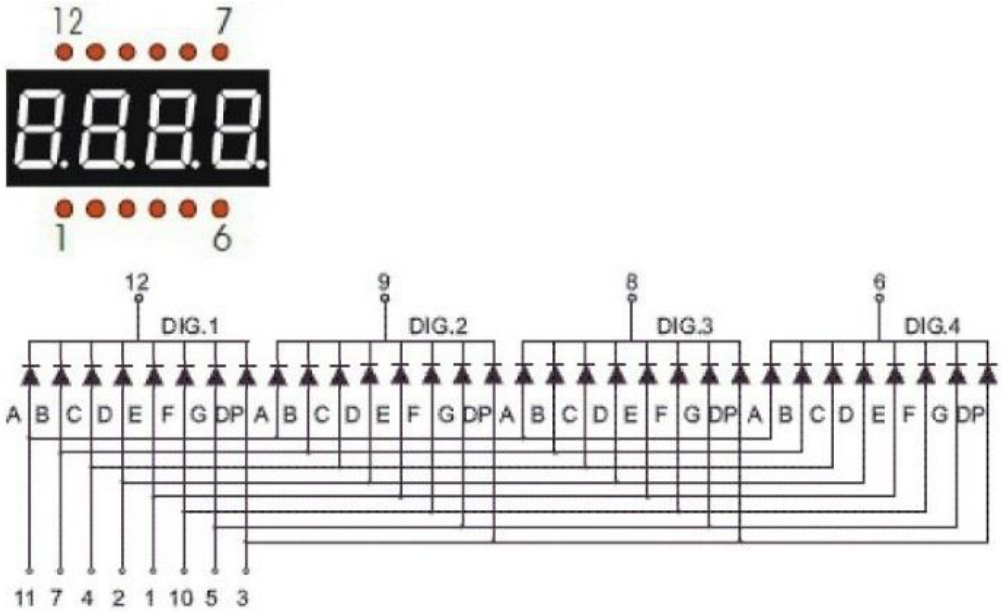
실습 1: polling 방식의 ADC

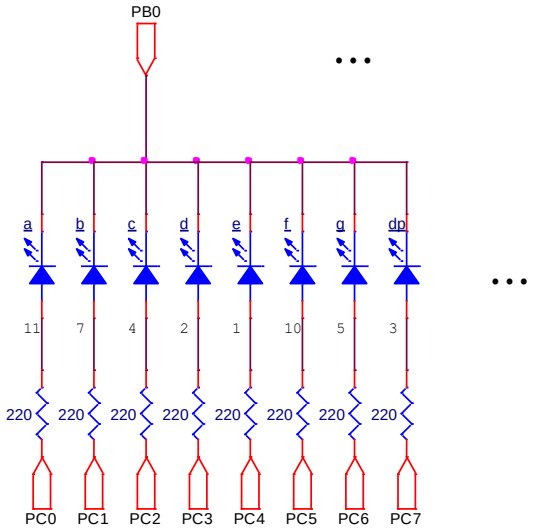
실습 2: interrupt 방식의 ADC

실습

HW 구성 (FND)

- FND의 A,B,C,D,E,F,G,DP를 저항(220Ω)을 연결한 후 PC1~PC8과 연결
- FND의 common인 12,9,8,6핀을 PB0~PB3과 연결
- 최대한 한쪽에 몰아서 구성할 것

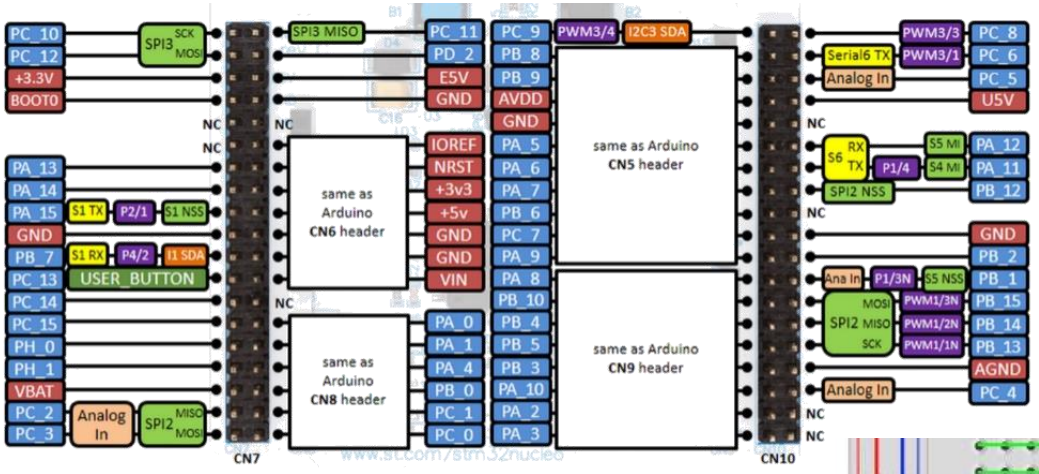


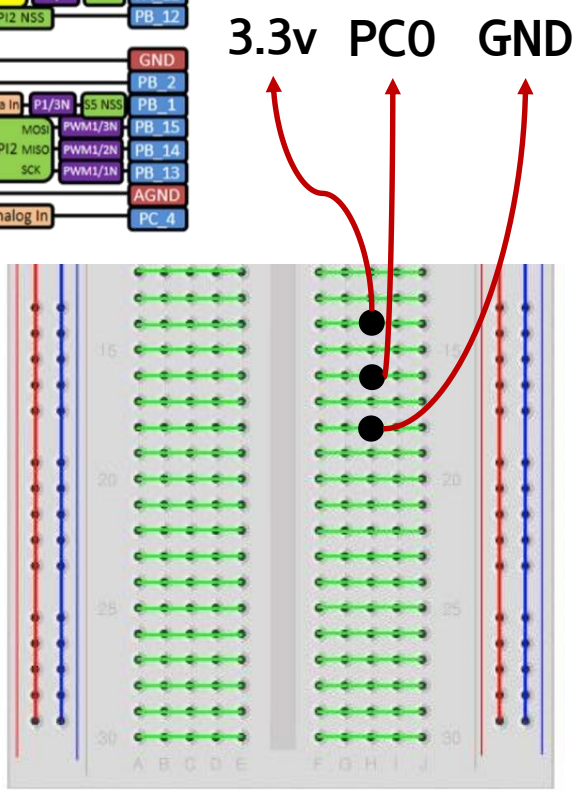


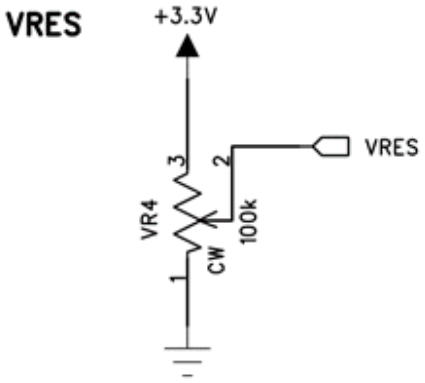
실습

HW 구성 (가변저항)

- 가변 저항을 브레드 보드에 연결 후, 3.3V, GND, PC0에 연결

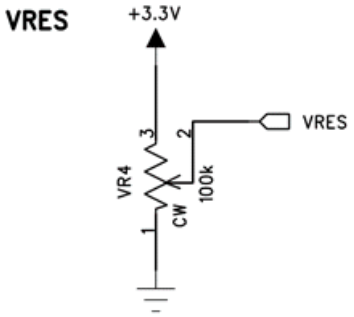


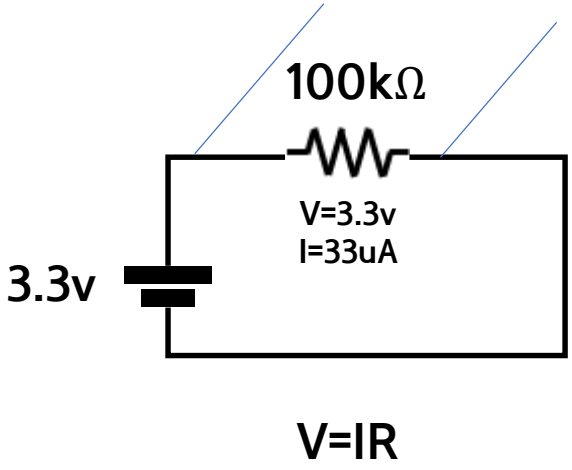


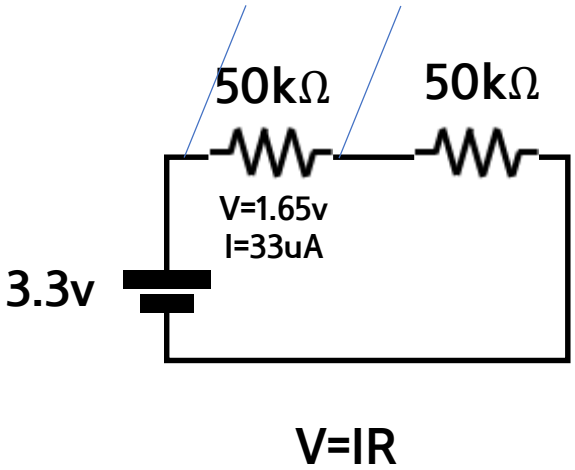


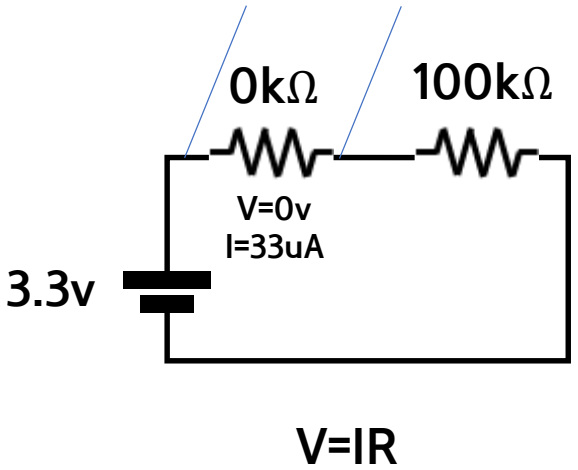
실습

HW 구성 (가변저항)









실습

파일 추가

- 폴더 : STM32F4xx_DSP_StdPeriph_Lib_V1.8.0₩Libraries₩ STM32F4xx_StdPeriph_Driver₩src
- 파일 : stm32f4xx_adc.c misc.c, stm32f4xx_tim.c, stm32f4xx_syscfg.c

실습 1: ADC polling 기반

```
#include "stm32f4xx.h"

unsigned char Font[10] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xD8, 0x80, 0x90};

uint8_t ADC1_flag = 0;
uint32_t ADC1_data = 0;
uint32_t ADC1_voltage = 0;

void Delay(__IO uint32_t nCount)
{
  for(; nCount != 0; nCount--);
}
```

```
void LED_init(void)
{
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
  GPIO_InitTypeDef GPIO_InitStructure;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```

실습 1: ADC polling 기반

```
void FND_Init(void)
{
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

  GPIO_InitTypeDef GPIO_InitStructure;

  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5 | GPIO_Pin_6
| GPIO_Pin_7 | GPIO_Pin_8;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;

  GPIO_Init(GPIOC, &GPIO_InitStructure);

  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;

  GPIO_Init(GPIOB, &GPIO_InitStructure);
}
```

실습 1: ADC polling 기반

```
void ADC_Configuration(void)
{
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

  GPIO_InitTypeDef GPIO_InitStructure;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOC, &GPIO_InitStructure);

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

  ADC_CommonInitTypeDef ADC_CommonInitStructure;

  ADC_CommonInitStructure.ADC_DMAAccessMode = ADC_DMAAccessMode_Disabled;
  ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div2;

  ADC_CommonInit(&ADC_CommonInitStructure);
```

ADC가 하는 일
아날로그 신호를 받아 디지털로 변환

ADC1을 사용하기 위한 클럭공급

뒤에서 설명드리겠음
(ADC를 수행하는 빈도를 결정하는 값)

실습 1: ADC polling 기반

```
 ADC_InitTypeDef ADC_InitStructure;

 ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
 ADC_InitStructure.ADC_ScanConvMode = DISABLE;
 ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
 ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
 ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
 ADC_InitStructure.ADC_NbrOfConversion = 1;
 ADC_Init(ADC1, &ADC_InitStructure);

 ADC_RegularChannelConfig(ADC1, ADC_Channel_10, 1, ADC_SampleTime_3Cycles);
 ADC_Cmd(ADC1, ENABLE);
}
```

ADC의 양자화를 12bit로 하겠음

다채널 ADC를 할 때 필요 (현재는 1채널만)

주기적 ADC 변환 여부 (이 실습에서는 원할 때만 수행)

외부 트리거에 의한 AD변환 시작

데이터 좌우 정렬 설정

ADC 변환수 (사용 채널 수)

실습 1: ADC polling 기반

```
void TIM2_Configuration(int intervalms)
{
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

  TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
  NVIC_InitTypeDef NVIC_InitStructure;

  TIM_TimeBaseStructure.TIM_Prescaler = 26880 - 1;
  TIM_TimeBaseStructure.TIM_Period = intervalms - 1;
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
  TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
  TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
  TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
  TIM_Cmd(TIM2, ENABLE);

  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
  NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}
```

실습 1: ADC polling 기반

```
void TIM2_IRQHandler(void)
{
 if(TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET)
 {
  GPIO_ToggleBits(GPIOA, GPIO_Pin_5);

  ADC1_data = (uint32_t)ADC1->DR;
  ADC_SoftwareStartConv(ADC1);
  ADC1_flag = 1;

  // clear interrupt flag
  TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
 }
}
```

Timer 2에 인터럽트가 걸리면,

User LED를 on/off함

ADC된 값을 가져옴
다시 ADC를 수행 함

Timer 2에 인터럽트 flag 초기화

Timer2의 인터럽트가 걸릴 때마다 ADC된 결과 값을 가져오고,
ADC 작업을 수행시킴
(원하는 시점에 ADC를 수행하고 있음)

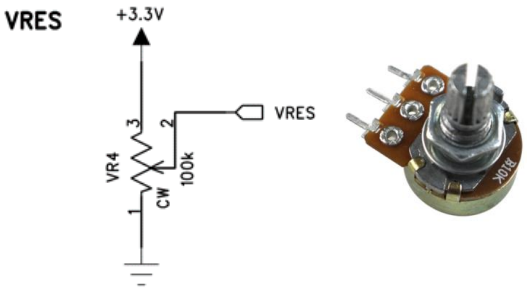
실습 1: ADC polling 기반

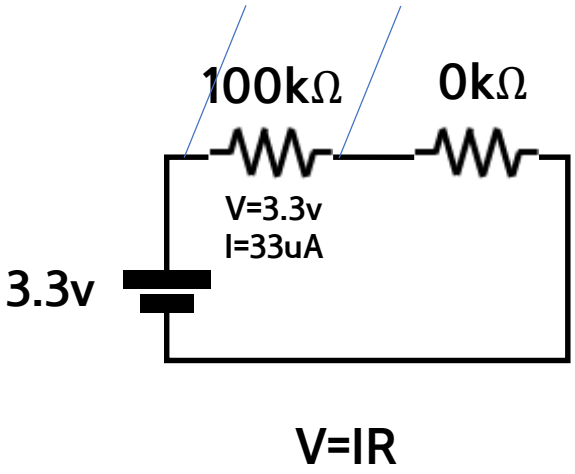
```
uint32_t ADC_to_Voltage(uint16_t data)
{
  ADC1_voltage = data *3300/4095;
  return ADC1_voltage;
}
```

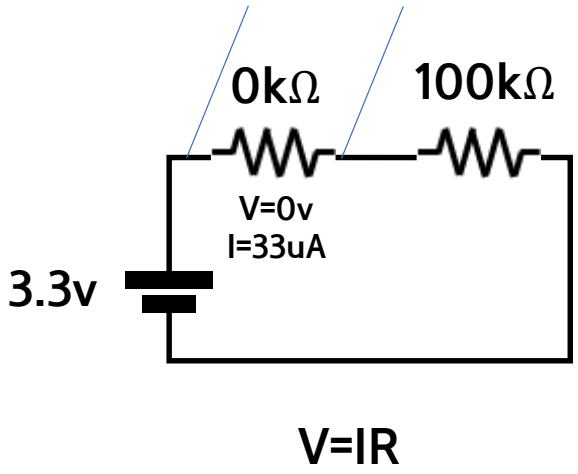
인가한 전압이 3.3V임 (=3300mV)
따라서, 저항에 따라 결과로 나올 수 있는 전압은
0 ~ 3300mV임

우리는 12비트로 ADC를 했음
0 ~ 3300mV가 2^{12} = 4096 등분되서,
0 ~ 4095 중 하나의 값으로 출력 됨

이 0 ~4095를 우리가 원하는 단위 mV로 변환이 필요







실습 1: ADC polling 기반

unsigned char Font[10] = {**0xC0**, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xD8, 0x80, 0x90};

```
void Count_Progress(int d_3, int d_2, int d_1, int d_0)
{
  GPIO_Write(GPIOB, 0x0008);
  GPIO_Write(GPIOC, Font[d_0]<<1);
  Delay(10000);

  GPIO_Write(GPIOB, 0x0004);
  GPIO_Write(GPIOC, Font[d_1]<<1);
  Delay(10000);

  GPIO_Write(GPIOB, 0x0002);
  GPIO_Write(GPIOC, Font[d_2]<<1);
  Delay(10000);

  GPIO_Write(GPIOB, 0x0001);
  GPIO_Write(GPIOC, Font[d_3]<<1);
  Delay(10000);
}
```

기존에는 FND를 PC0 ~ PC7에 연결하였음
이번에는 PC1 ~ PC8에 연결하였음

예. 0xC0

PC15	PC14	PC13	PC12	PC11	PC10	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
0	0	0	0	0	0	0	0	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**

PC15	PC14	PC13	PC12	PC11	PC10	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
0	0	0	0	0	0	0	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	0

실습 1: ADC polling 기반

```
int main()
{
  ADC1_flag = 0;
  ADC1_data = 0;

  LED_init();
  FND_Init();
  TIM2_Configuration(10);
  ADC_Configuration();

  while(1)
  {
    if(ADC1_flag == 1)
    {
      ADC1_voltage = ADC_to_Voltage(ADC1_data);
      Count_Progress(ADC1_voltage/1000, ((ADC1_voltage/100)%10), ((ADC1_voltage/10)%10), (ADC1_voltage%10));
      //Count_Progress(ADC1_voltage/1000, ((ADC1_voltage/100)%10), 0, 0);
      ADC1_flag = 0;
    }

  }
  return 0;
}
```

결과적으로 10ms에 한번씩 ADC를 할 것임

실습 2: ADC Interrupt 기반

```
#include "stm32f4xx.h"

unsigned char Font[10] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xD8, 0x80, 0x90};

uint8_t ADC1_flag = 0;
uint32_t ADC1_data = 0;
uint32_t ADC1_voltage = 0;

uint32_t sum_adc_data = 0;
uint32_t mean_adc_data = 0;
uint32_t adc_cnt = 0;

#define WIN 1000

void Delay(__IO uint32_t nCount)
{
  for(; nCount != 0; nCount--);
}
```

```
void LED_init(void)
{
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
  GPIO_InitTypeDef GPIO_InitStructure;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```

실습 2: ADC Interrupt 기반

```
void FND_Init(void)
{
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

  GPIO_InitTypeDef GPIO_InitStructure;

  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5 |
GPIO_Pin_6 | GPIO_Pin_7 | GPIO_Pin_8;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;

  GPIO_Init(GPIOC, &GPIO_InitStructure);

  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;

  GPIO_Init(GPIOB, &GPIO_InitStructure);
}
```

실습 2: ADC Interrupt 기반

```
void Count_Progress(int d_3, int d_2, int d_1, int d_0)
{
  GPIO_Write(GPIOB, 0x0008);
  GPIO_Write(GPIOC, Font[d_0]<<1);
  Delay(10000);

  GPIO_Write(GPIOB, 0x0004);
  GPIO_Write(GPIOC, Font[d_1]<<1);
  Delay(10000);

  GPIO_Write(GPIOB, 0x0002);
  GPIO_Write(GPIOC, Font[d_2]<<1);
  Delay(10000);

  GPIO_Write(GPIOB, 0x0001);
  GPIO_Write(GPIOC, Font[d_3]<<1);
  Delay(10000);
}
```

```
uint32_t ADC_to_Voltage(uint16_t data)
{
  ADC1_voltage = data *3300/4095;
  return ADC1_voltage;
}
```

실습 2: ADC Interrupt 기반

```
void ADC_Configuration(void)
{
 RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

 GPIO_InitTypeDef GPIO_InitStructure;
 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
 GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
 GPIO_Init(GPIOC, &GPIO_InitStructure);

 NVIC_InitTypeDef NVIC_InitStructure;
 NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
 NVIC_InitStructure.NVIC_IRQChannel = ADC_IRQn;
 NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x01;
 NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x02;
 NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
 NVIC_Init(&NVIC_InitStructure);

 RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

 ADC_CommonInitTypeDef ADC_CommonInitStructure;
 ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;
 ADC_CommonInitStructure.ADC_DMAAccessMode = ADC_DMAAccessMode_Disabled;
 ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div2;
 ADC_CommonInit(&ADC_CommonInitStructure);
```

ADC가 완료되면, interrupt를 request!

ADC를 독립적으로 수행해라

ADC를 얼마의 빈도로 수행할지를 결정 (_Div8로 해볼 것!)

실습 2: ADC Interrupt 기반

```
  ADC_InitTypeDef ADC_InitStructure;

  ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
  ADC_InitStructure.ADC_ScanConvMode = DISABLE;
  ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
  ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T1_CC1;
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfConversion = 1;
  ADC_Init(ADC1, &ADC_InitStructure);

  ADC_RegularChannelConfig(ADC1, ADC_Channel_10, 1, ADC_SampleTime_3Cycles);
  ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE);
  ADC_Cmd(ADC1, ENABLE);
}
```

ADC를 계속 수행하라

실습 2: ADC Interrupt 기반

```
void ADC_IRQHandler(void)
{                                                    EOC: end of conversion
 if (ADC_GetITStatus(ADC1, ADC_IT_EOC) != RESET)     ADC가 완료되어 interrupt가 걸리면,
 {
  ADC_ClearITPendingBit(ADC1, ADC_IT_EOC);
  ADC1_data = ADC_GetConversionValue(ADC1);          변환된 값을 가져와라

  sum_adc_data += ADC1_data;                         누적해서 더해라
  adc_cnt++;

  if (adc_cnt >= WIN)                                WIN만큼 더하면,
  {
   GPIO_ToggleBits(GPIOA, GPIO_Pin_5);

   ADC1_flag = 1;
   adc_cnt = 0;
   mean_adc_data = sum_adc_data / WIN;               평균을 계산하라
   sum_adc_data = 0;

   // ADC1_voltage = ADC_to_Voltage(mean_adc_data);
   // Count_Progress(ADC1_voltage/1000, ((ADC1_voltage/100)%10), ((ADC1_voltage/10)%10), (ADC1_voltage%10));
  }
 }
}
```

실습 2: ADC Interrupt 기반

```
int main()
{
  ADC1_flag = 0;
  ADC1_data = 0;
  adc_cnt = 0;
  mean_adc_data = 0;
  sum_adc_data = 0;

  LED_init();
  FND_Init();
  ADC_Configuration();

  ADC_SoftwareStartConv(ADC1);

  while(1)
  {
    if(ADC1_flag == 1)
    {
      ADC1_voltage = ADC_to_Voltage(mean_adc_data);
      Count_Progress(ADC1_voltage/1000, ((ADC1_voltage/100)%10), ((ADC1_voltage/10)%10), (ADC1_voltage%10));

      ADC1_flag = 0;
    }

  }
  return 0;
}
```

ADC를 시작해라.
Continuous 모드이므로 한번 시작하면, 특정 빈도에 따라 계속 ADC를 수행

남은 내용

- USART: Serial 통신
  - RX: 데이터 수신
  - TX: 데이터 송신
  - 실습 1: (아마도) PC에서 데이터를 받아 MCU의 무언가를 제어
  - 실습 2: (아마도) PC에 데이터를 송신하여 PC에서 확인
- PC UI 구현
  - MCU에서 ADC한 결과를 Serial 통신으로 PC에 보내고, 그래프로 그리고, 처리하고 등등
  - 실습 1: UI 구현
  - 실습 2: 심전도 데이터 수집 & 그래프 그리기 & 데이터 처리
- 프로젝트

과제

- ADC 따라해보기

Thank you.