AVR - 2

임베디드스쿨1기

Lv1과정

2020. 09. 11

강경수

AVR INTERRUPT

```c
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define sbi(PORTX,DitX) (PORTX |= (1<<DitX)) //bit를 체크하기위하여 정의  A

SIGNAL(INT1_vect) //벡터테이블주소
{
  PORTB = 0x20;
  _delay_ms(200);
}

int main(void)
{
    sbi(SREG,7);

    sbi(EIMSK,INT1);

    EICRA = 0x08;
    DDRB = 0x20;
    DDRD = 0x00;
    PORTD = 0xff;

    /* Replace with your application code */
    while (1)
    {
      PORTB = 0x00;
    }
}
```

추가 공부 할 것
ISR 과 SERVICE 함수 차이
Volatile 변수 선언에 대해서

```
00000000 <__vectors>:
   0:   0c 94 34 00     jmp 0x68    ; 0x68 <__ctors_end>
   4:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
   8:   0c 94 40 00     jmp 0x80    ; 0x80 <__vector_2>
   c:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
  10:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
  14:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
  18:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
  1c:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
  20:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
  24:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
  28:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
  2c:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
  30:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
  34:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
  38:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
  3c:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
  40:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
  44:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
  48:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
  4c:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
  50:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
  54:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
  58:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
  5c:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
  60:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
  64:   0c 94 3e 00     jmp 0x7c    ; 0x7c <__bad_interrupt>
```

벡터별로 할당된 주소로
Jmp 하는것 볼 수 있음.

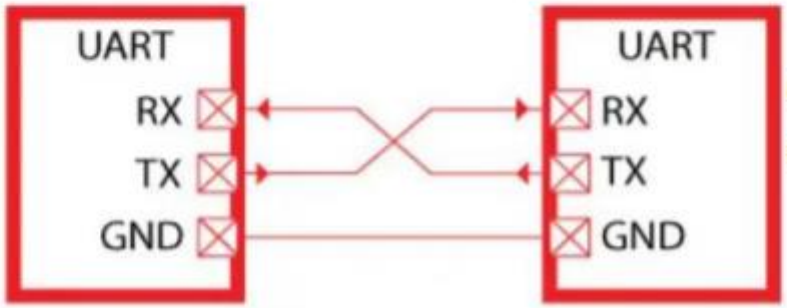
AVR UART

▣ Atmega328p UART

2020.09.25 KKS

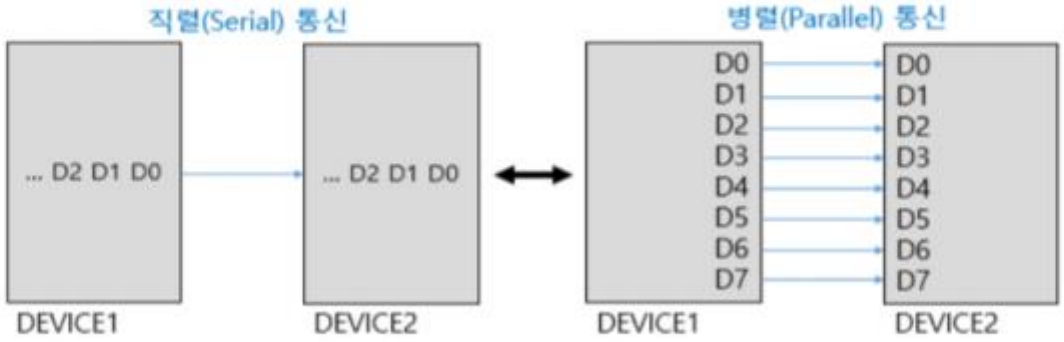
1. UART는 전이중방식?

- Simple Duplex(단방향 통신) : RX 혹은 TX한 기능만 가능함.
- Half Duplex(반이중 통신) : RX, TX 다 가능하지만 RX시 TX는 불가능하며 TX시 RX는 불가능함.
- Full Duplex(전이중 방식) : RX하며 동시에 TX도 가능함.



UART의 경우 전이중방식
RX TX 동시에 사용 가능

2. UART통신은 직렬통신?



- 한 통신선을 이용하여 단위 시간에 1개의 데이터만 전송 및 수신하므로 직렬통신

AVR UART

- RS232? RS423? RS485?

Characteristics of RS232, RS422, RS423 and RS485

	RS232	RS423	RS422	RS485
Differential	no	no	yes	yes
Max number of drivers Max number of receivers	1 1	1 10	1 10	32 32
Modes of operation	half duplex full duplex	half duplex	half duplex	half duplex
Network topology	point-to-point	multidrop	multidrop	multipoint
Max distance (acc. standard)	15 m	1200 m	1200 m	1200 m
Max speed at 12 m Max speed at 1200 m	20 kbs (1 kbs)	100 kbs 1 kbs	10 Mbs 100 kbs	35 Mbs 100 kbs
Max slew rate	30 V/µs	adjustable	n/a	n/a
Receiver input resistance	3..7 kΩ	≧ 4 kΩ	≧ 4 kΩ	≧ 12 kΩ
Driver load impedance	3..7 kΩ	≧ 450 Ω	100 Ω	54 Ω
Receiver input sensitivity	±3 V	±200 mV	±200 mV	±200 mV
Receiver input range	±15 V	±12 V	±10 V	-7..12 V
Max driver output voltage	±25 V	±6 V	±6 V	-7..12 V
Min driver output voltage (with load)	±5 V	±3.6 V	±2.0 V	±1.5 V

- 기본 DATA전송방식은 UART
- PC와 통신시 혹은 더 장거리 통신을 위하여 TTL레벨에서 전압레벨을 더 크게 변환시킴
- 잡음내성이 강해지고 장거리 송수신가능
- 별도의 추가공부 필요.

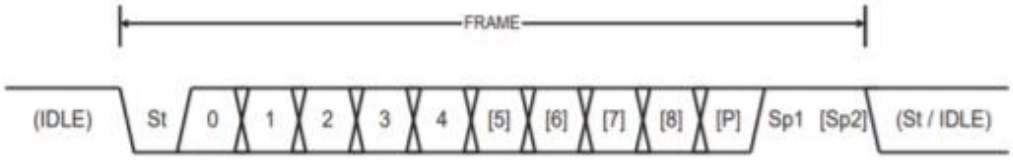
3. 일반적인 UART통신 특징
- Baud Rate 가 있다.
- CLOCK라인 따로 없이 데이터신호만을 이용한다.
- 별도의 STOP비트 (1~2) 사용이 가능하다.
- 별도의 패리티비트(짝수 또는 홀수) 사용이 가능하다.
- 송신,수신 관련 인터럽트가 발생한다.

3. Baud rate vs BPS
- 일반적으로 bps(bit per second) 와 Baud Rate가 같은 말인줄 알고있다.
- Baud Rate : 신호한번 변할때 전송되는 Bit 수
- BPS : 초당 변하는 Bit 수
- UART에서는 1CLOKC 에 1개의 변화만 존재하므로 Bps = Baud Rate

4. Synchronous VS Asynchronous
- 동기화 출력 VS 동기화 입력
  Baud Rate Clock을 외부 Slave에서 사용 할 수 있게 출력하는 것.
  Baud Rate Clock을 외부 Master에서 받아 오는 것.
- 비 동기화의 경우 내부 Clock Generator가 존재함.

5. 전송데이터 포맷

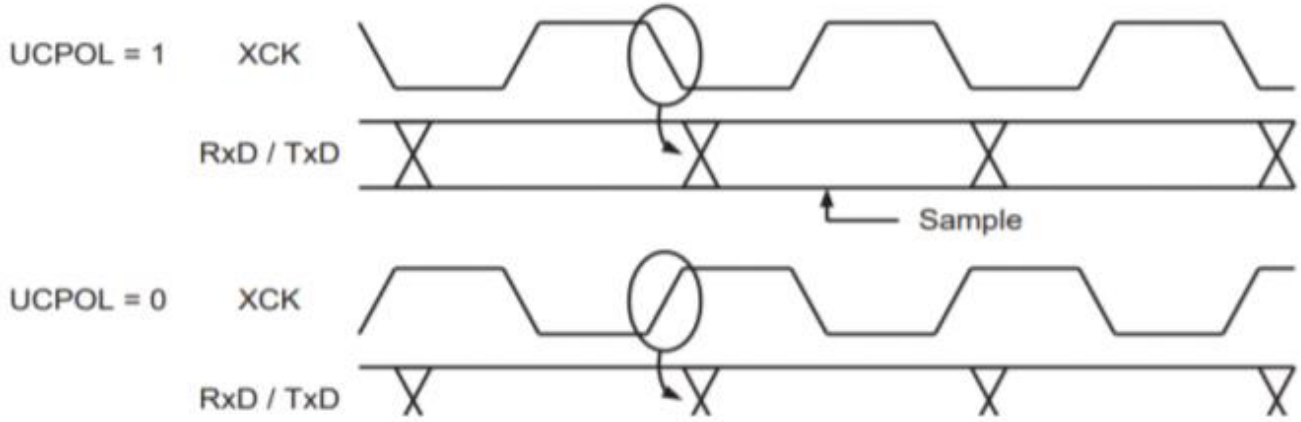
Figure 17-4. Frame Formats



St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxDn or TxDn). An IDLE line

6. Synchronous 모드에서 sample(입력) data change(출력) 시점
- 아래 그림처럼 UCPOL이 1이면 하강EDGE에, 0이면 상승에지 샘플 및 변화.
- 이걸 왜 별도로 설정하는지는 잘 모르겠음..

Figure 17-3. Synchronous Mode XCKn Timing.



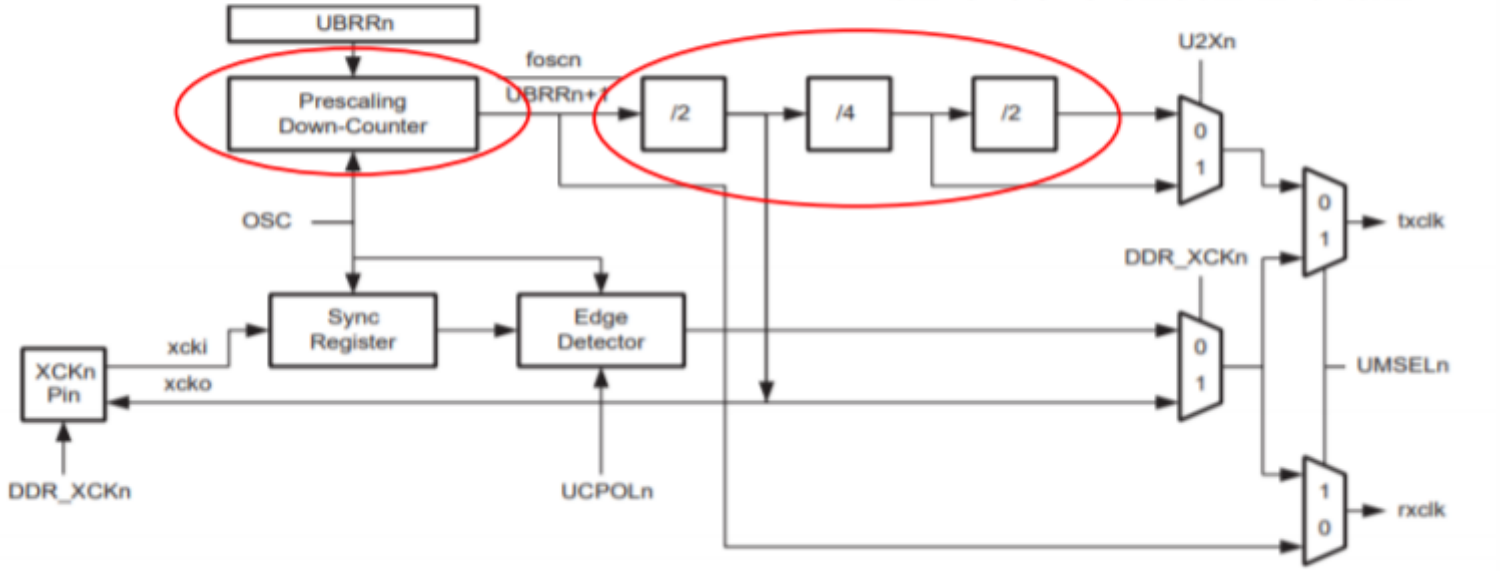
7. Double speed Operation

- U2Xn을 set해서 Baud Rate를 2배로 할 수 있다. 비동기시에만 사용 가능.

  데이터 수신 중간에 Bps를 2배로 한다고 기술 돼 있는데 왜 필요한기능인지 모르겟음..

7. Clock 블록

내부생성, 혹은 외부입력 받은 clock에서
배율이 나누어지는것을 볼 수 있음.



7. 주요 레지스터

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDRn (Read)
	TXB[7:0]								UDRn (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- UDRn에 수신된 RX, TX레지스터가 존재한다.
- RX, TX DATA는 모두 이 레지스터에 쓰인다.
- 값을 쓰면 RX값이 쓰이고, 값을 읽으면 TX값이 읽힌다.

UCSRnA – USART Control and Status Register n A

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- UART수신완료, UART 송신완료, 데이터 EMPTY 확인, 에러, MULTI SLAVE 레지스터

UCSRnB – USART Control and Status Register n B

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- RX TX 완료 인터럽트 허용, RX,TX, 허용 , 9번째 BIT 저장 레지스터

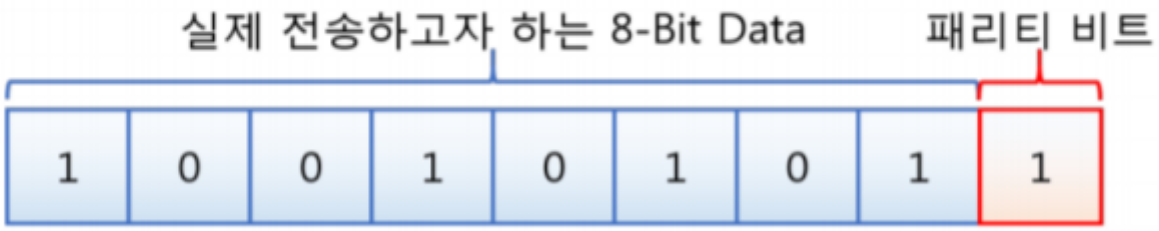
UCSRnC – USART Control and Status Register n C

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

-동기,비동기, STOP BIT, PARITY BIT, DATA SIZE 설정 레지스터

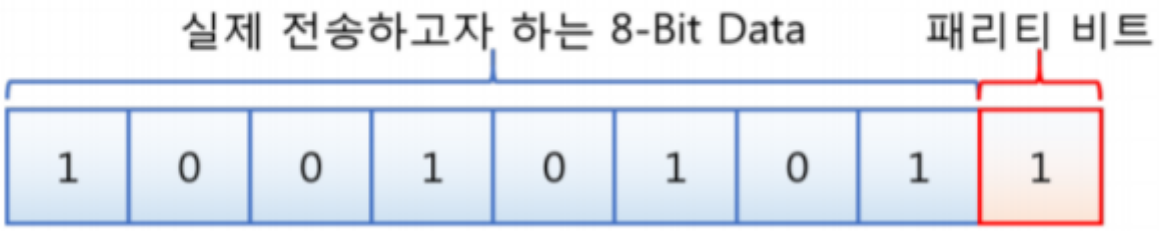
8. MPCMn에 관한 자세한 설명
- 1개의 MASTER에 n개의 slave가 사용 될 수 있음.
- 이때 9비트방식을 사용하여 제어할 slave를 선택한이후
  8bit data를 전송하는 방식으로 사용함.(선택받지 못한 slave들은 계속 9bit확인)

9. Paritiy bit란 무엇인가?
- 정보의 전달과정에서 noise등으로 인한 오류가 생겼는지 검출하기 위한 bit
- EVEN, ODD로 나뉨
- Even 은 1의 개수를 짝수로 맞춰주는 bit
- odd 는 1의 개수를 홀수로 맞춰주는 bit
- 10110111 에서 even parity : 1 odd paritu: 0
- 수신하는 측에서 이 parity bit를 보고 실제 데이터와 맞지 않을경우 master에게 재전송 요청



8. MPCMn에 관한 자세한 설명
- 1개의 MASTER에 n개의 slave가 사용 될 수 있음.
- 이때 9비트방식을 사용하여 제어할 slave를 선택한이후
  8bit data를 전송하는 방식으로 사용함.(선택받지 못한 slave들은 계속 9bit확인)

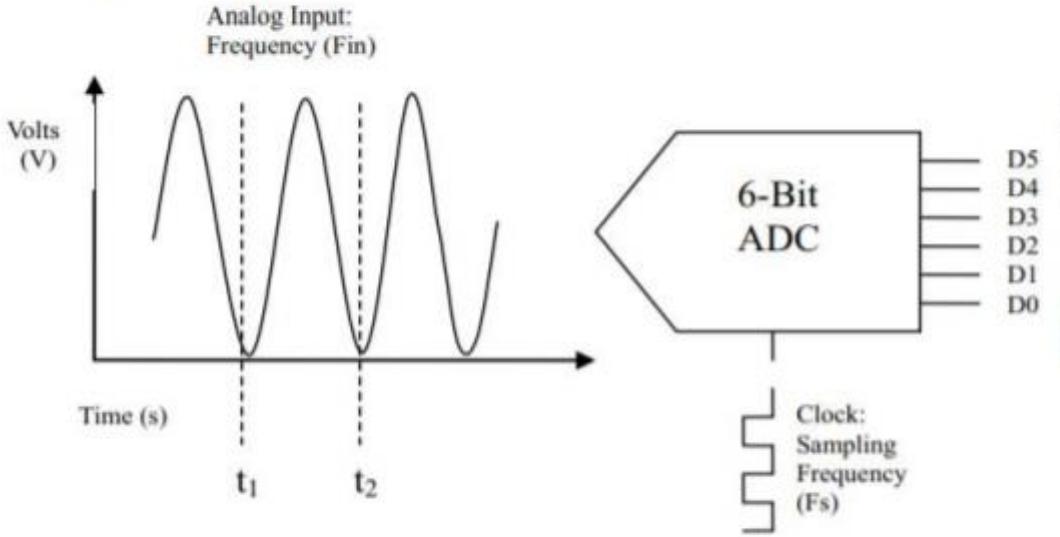
9. Paritiy bit란 무엇인가?
- 정보의 전달과정에서 noise등으로 인한 오류가 생겼는지 검출하기 위한 bit
- EVEN, ODD로 나뉨
- Even 은 1의 개수를 짝수로 맞춰주는 bit
- odd 는 1의 개수를 홀수로 맞춰주는 bit
- 10110111 에서 even parity : 1 odd paritu: 0
- 수신하는 측에서 이 parity bit를 보고 실제 데이터와 맞지 않을경우 master에게 재전송 요청



▣ ATmega328p ADC

2020.09.25 KKS

1. ADC란 무엇인가?

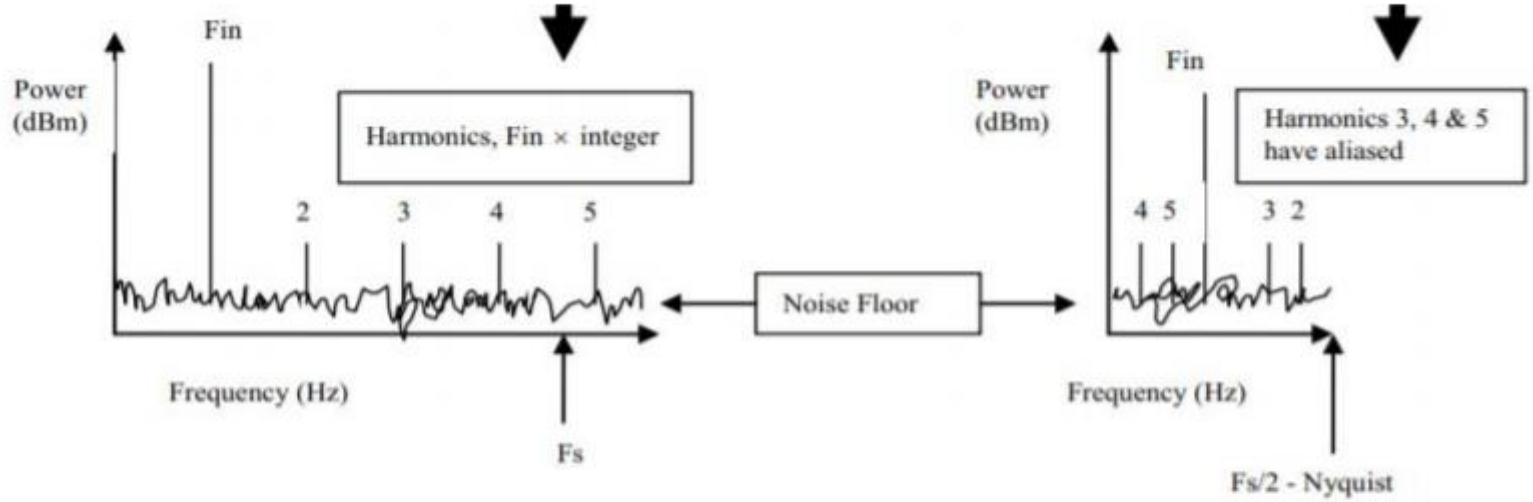


Digital Output

Time	D5	D4	D3	D2	D1	D0
1/Fs	0	1	1	1	1	1
2/Fs	1	0	1	0	1	0
3/Fs	1	1	1	0	0	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
N-2/Fs	0	1	0	1	0	1
N-1/Fs	0	0	0	1	1	1
N/Fs	1	1	0	1	0	1

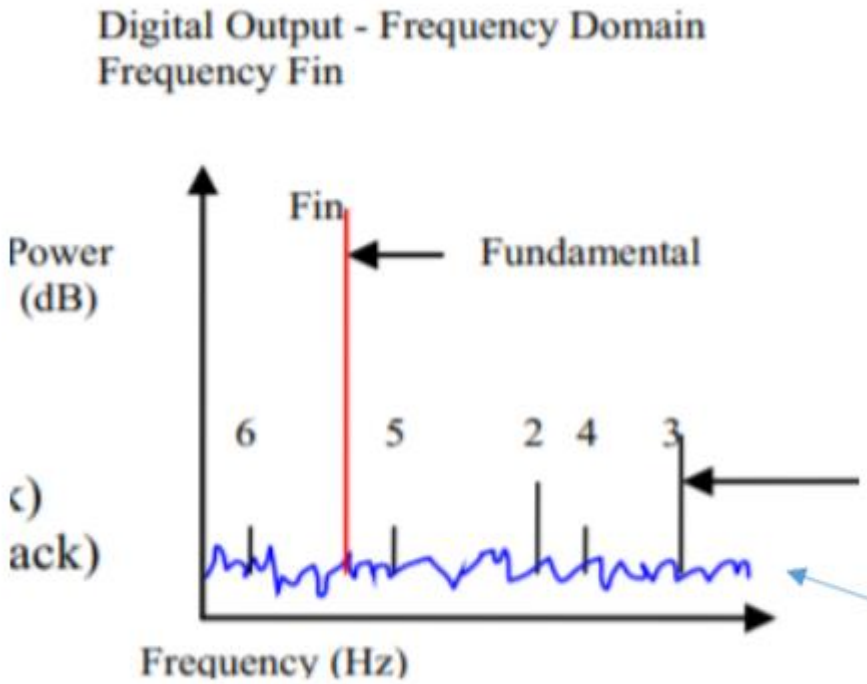
- 연속적인 아날로그 신호를 디지털단위로 바꾸는 장치를 이야기함.
- 아날로그 신호가 일정 구간마다 2^6의 최대값을 갖는 디지털값으로 변환 된 것을 볼 수 있다.

2. 주파수 관점에서 ADC



- Harmonic : 샘플링하는 주파수의 정수배로 생기는 잡음(고조파)
- SNR : Signal to Noise Ratio 즉 본 신혼 Fundamental 대비Noise Floor 비
- Nyquist 이론에 따르면 10kHz 간격으로 Sampling할 시 5kHz미만의 대역만 사용 가능

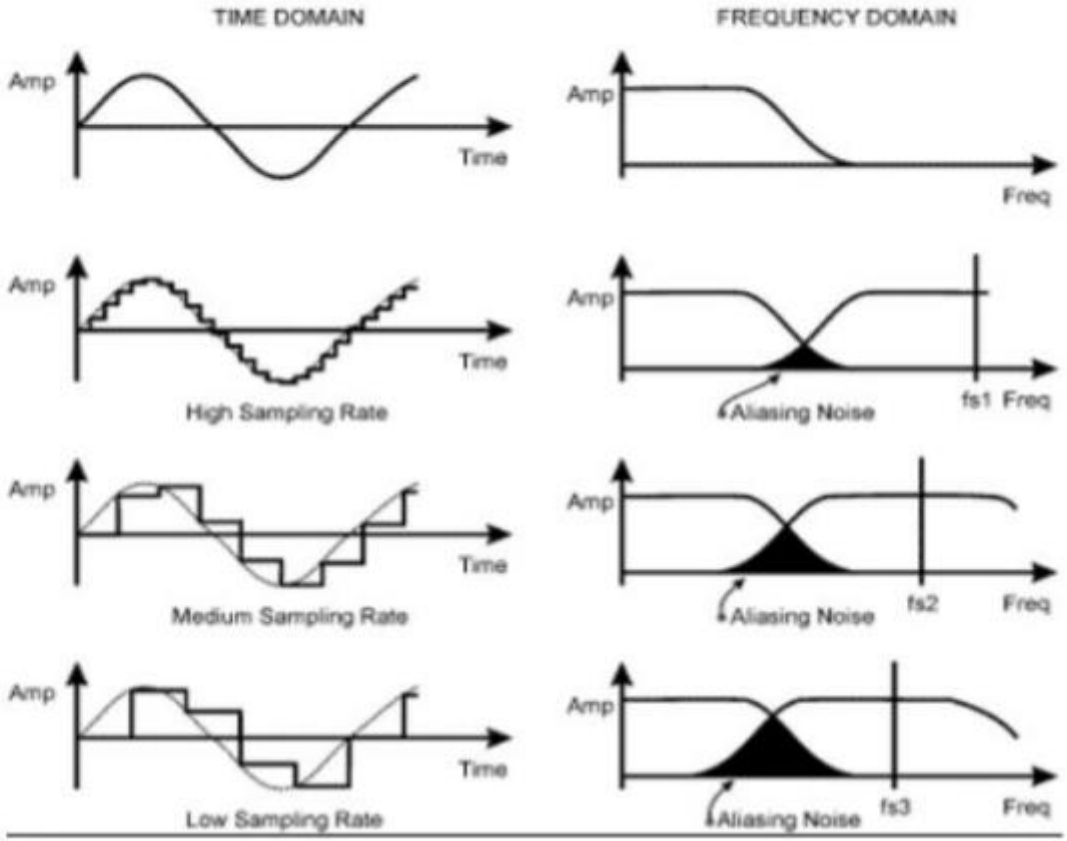
AVR ADC



- Nyquist - shannon sampling frequency
  어떤 파형을 표현하기 위해선 아날로그 파형 주기의 2배의를 샘플링 해야함.
→ 실제 자연계 아날로그 신호에선 사용 불가능 하다. 자연계 아날로그신호 주파수 대역이 거의 무한대인데.
무한대 * 2 = 무한대 이기 때문이다.

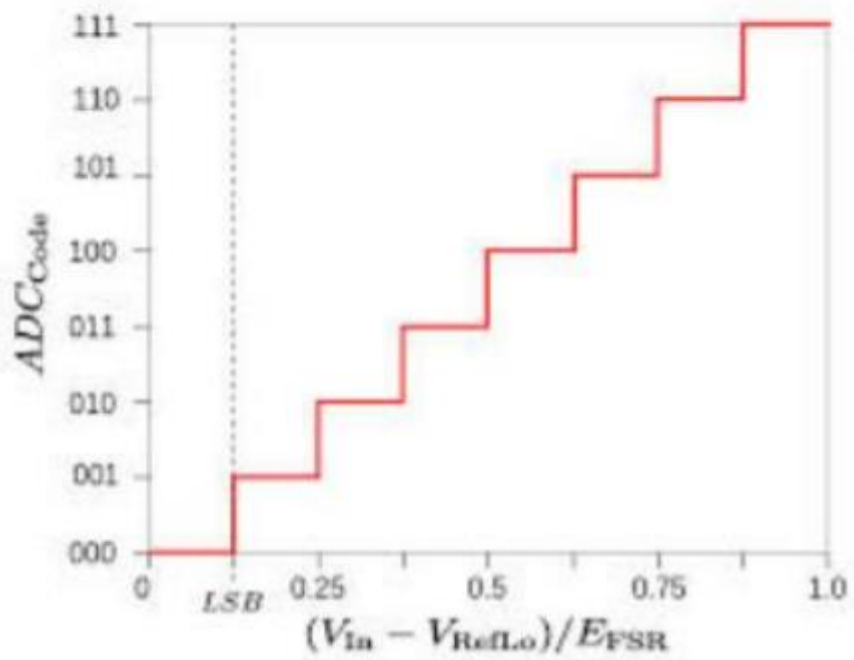
옆 그래프처럼 주파수 도메인으로 나타내주기 위해서는 신호에대해 퓨리에 변환이 필요하다.

- 아래와같은 에일리어싱 노이즈가 발생한다. 이를 주로 H/W필터(LOW PASS)로 처리한다.



AVR ADC

3. 분해능



분해능 : 전압 LEVEL을 얼마나 세세하게 나눌 수 있는가.

← 옆 그래프의 분해능은 3bit 즉 2^3 = 8

1V라는 크기를 8개의 수로 표현한다.

ATMEGA328p 10bit 기준 1024의 분해능

3. ATmega328p ADC

1) Single Conversion VS Free Running

- 1번만 adc를 실행함 마친이후 인터럽트 발생
- 계속해서 adc를 실행함

2) Single ended VS differential (328p에서는 singled ended만 사용 가능)

- single ended : AVCC AREF 내부 2.54V 중 선택
- differential : 두 채널의 전압 차를 ADC EX) ADC0 : 1.5V ADC1 : 1V ADC결과 : 0.5V

4) 주요 레지스터

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- 레퍼런스 전압 선택 비트

ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- ADC Control & Status 레지스터

ADCSRB – ADC Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
(0x7B)	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

AVR ADC

- 오토 트리거 레지스터

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
(0x79)	–	–	–	–	–	–	ADC9	ADC8	ADCH
(0x78)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
(0x79)	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
(0x78)	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

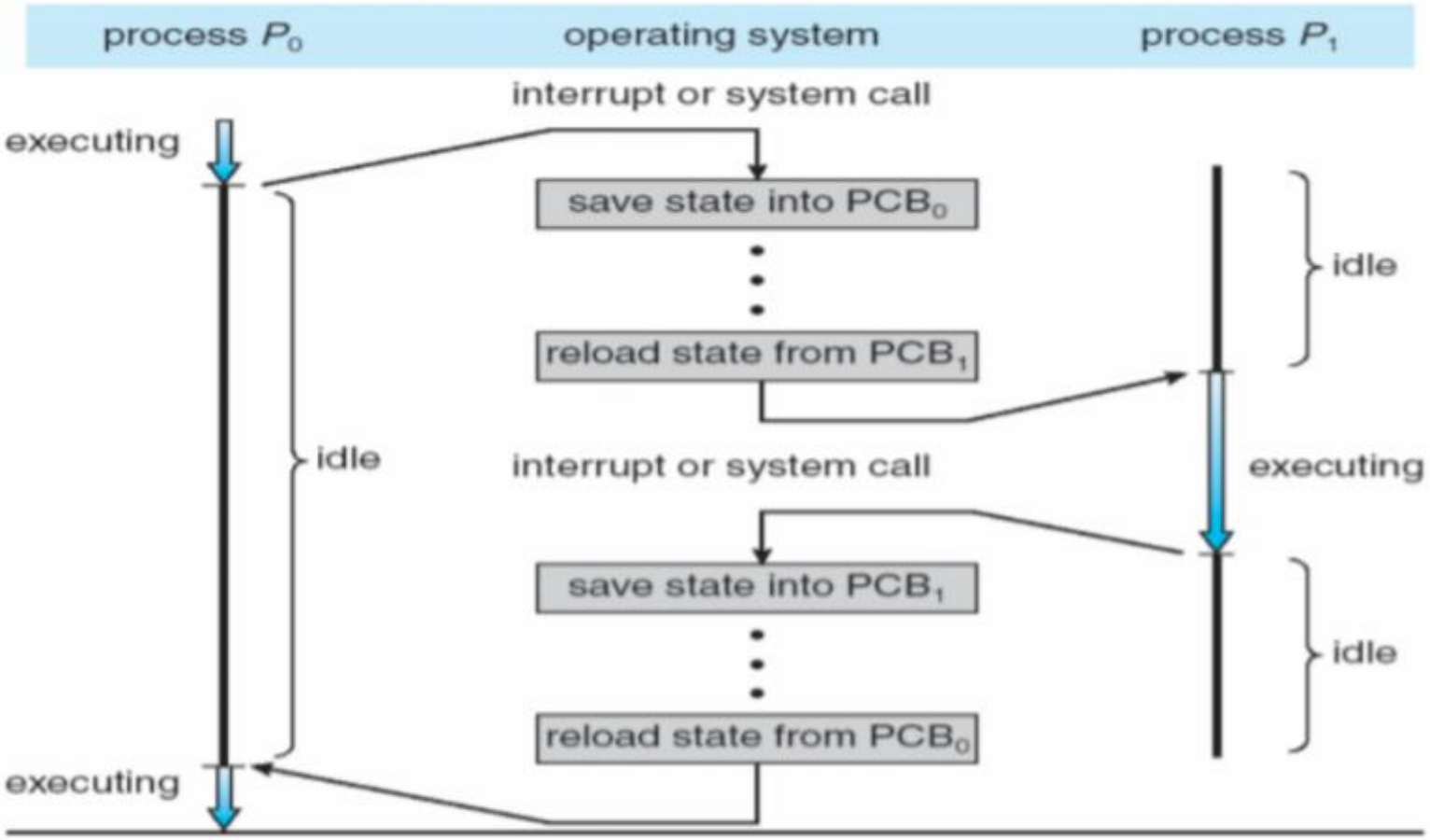
- ADC저장되는 레지스터
- 일단 ADCL을 읽으면 이후에 ADCH를 읽지 않을경우 ADC값 갱신안된다.
- ADCH만 읽는건 상관없음.

▣ Context switching

2020.09.25 KKS

1. Context swtiching 이란?
 - Data section, LINUX에서는 SHARD MEMORY 변수에 값을 쓴 이후
 시스템콜 혹은 인터럽트에 의하여 다른 Task 혹은 프로세서에서 해당 변수, 레지스터에
 접근시 데이터 무결성이 손실 될 수 있음. 이를 방지하기 위해
 PCB(Process Control Block에 레지스터 값을 저장해두고 인터럽트,시스템콜 전으로 돌아와 실행

2. 전역변수 a = 10; 이라고 선언되었다 해보자. 이 전역변수에는 다른 모든 task들이 접근이 가능함.
 이때 어떤 함수에서 a변수를 mov 했다고 해보자. 그다음 인터럽트 발생 인터럽트 핸들러에게
 우선권을 빼앗겼다 가정하자. 인터럽트 서비스 루틴에서 a = 9라는 값을 새로 썼다.
 이때 인터럽트에서 복귀할 시 아까 호출된 함수는 a = 10으로 알고 있다.
 하지만 실제 a = 9 이며 데이터 무결성이 깨어진다.
 이런 일이 일어나는 영역을 'critcial section' 이라 하며 이를 방지하기 위한 것에
 context swtiching , mutex , Semephore등이 있다.



3. 펌웨어(주로 Polling으로 도는 non os)에서는 고려해 주지 않아도 무방하나
RTOS와 같이 task단위로 또한 scheduling에 의해 돌아가는 시스템에서는 고려해주어야 함.

AVR ADC