

임베디드 시스템을 위한 SW 구조 설계

(file #6 / 10) ver0.2

Yongseok Chi

1. Develop an understanding of technologies

about the micro controller & processor systems using evaluation kit

2. Skill up a **design ability the micro controller application systems**

(1) technology of **the hardware and software** components

(2) **debugging** technology about the micro controller

(3) understanding of a **circuit design** skill

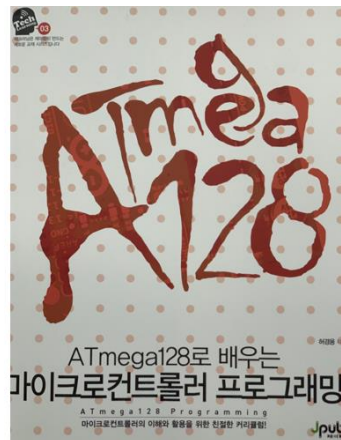
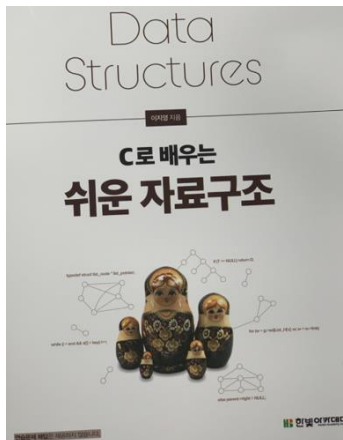
3. Develop an ability of design about the micro controller

1. Reference information

(1) Atmel datasheet, <http://atmel.com> (→ microchip.com)

(2) ARM Architecture Reference Manual, <http://arm.com>

2. Books



3. 실험KIT (Evaluation board)

한백전자 IOT 실험 KIT



KUT-128_Com 실험 키트

-
1. Micro Processor 원리
 2. Atmel사의 8bit Micro-controller
 3. KUT0128 Evaluation Board 기능과 특징
 4. IO Port 제어
 5. External Interrupt 제어
 6. Timer / Counter 제어
 7. UART 제어
 8. AD Converter 제어
 9. Comparator 제어
 10. EEPROM 제어 (IIC, Parallel method)
 11. SPI 제어

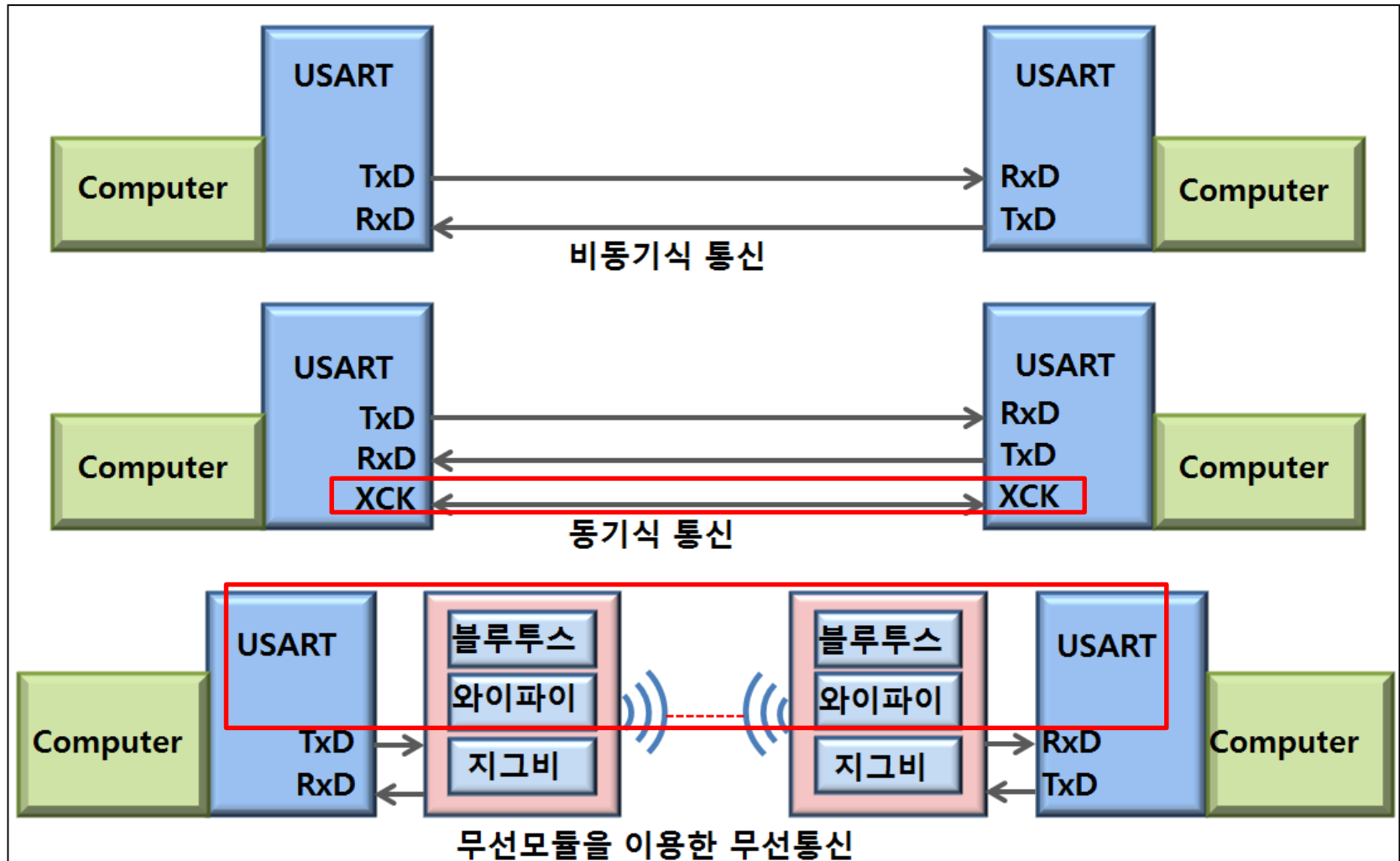
7. USART (Universal Synchronous and Asynchronous Receiver and Transmitter)

컴퓨터 통신

- 병렬 통신(Parallel communication)
 - : 여러 개의 라인을 통해 동시에 데이터를 전송
 - : 고속 데이터 통신이 가능하지만 통신 거리에 제한이 있음
- 직렬 통신(Serial communication)
 - : 하나(또는 2-3개)의 라인을 통해 데이터를 전송
 - : 통신 속도는 병렬 통신에 비해 느리지만 먼 거리까지 통신이 가능함
 - : 동기식, 비동기식 직렬 통신
- 동기식 시리얼 통신(Synchronous communication)
 - : 데이터 라인 외에 기준 clock인 동기 clock 라인이 있음
 - : 데이터를 기준 clock에 동기 되어 순차적으로 송수신 함
 - : 비동기 방식에 비해 비용이 많이 들지만 통신의 안정성이 좋음
- 비동기식 시리얼 통신(Asynchronous communication)
 - : 동기 clock 없이 데이터만을 송수신 함
 - : 데이터의 전송 속도 및 기타 사항을 송수신부간에 동기 시킴

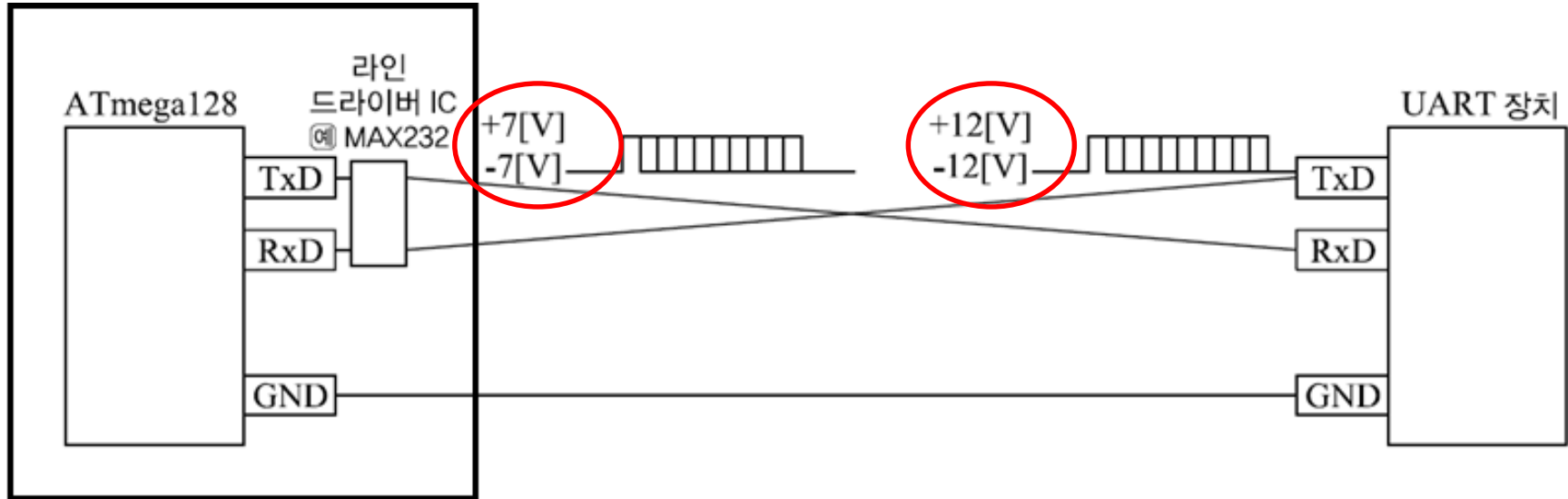
7. USART

Universal Synchronous and Asynchronous Receiver and Transmitter



7. USART

Universal Synchronous and Asynchronous Receiver and Transmitter

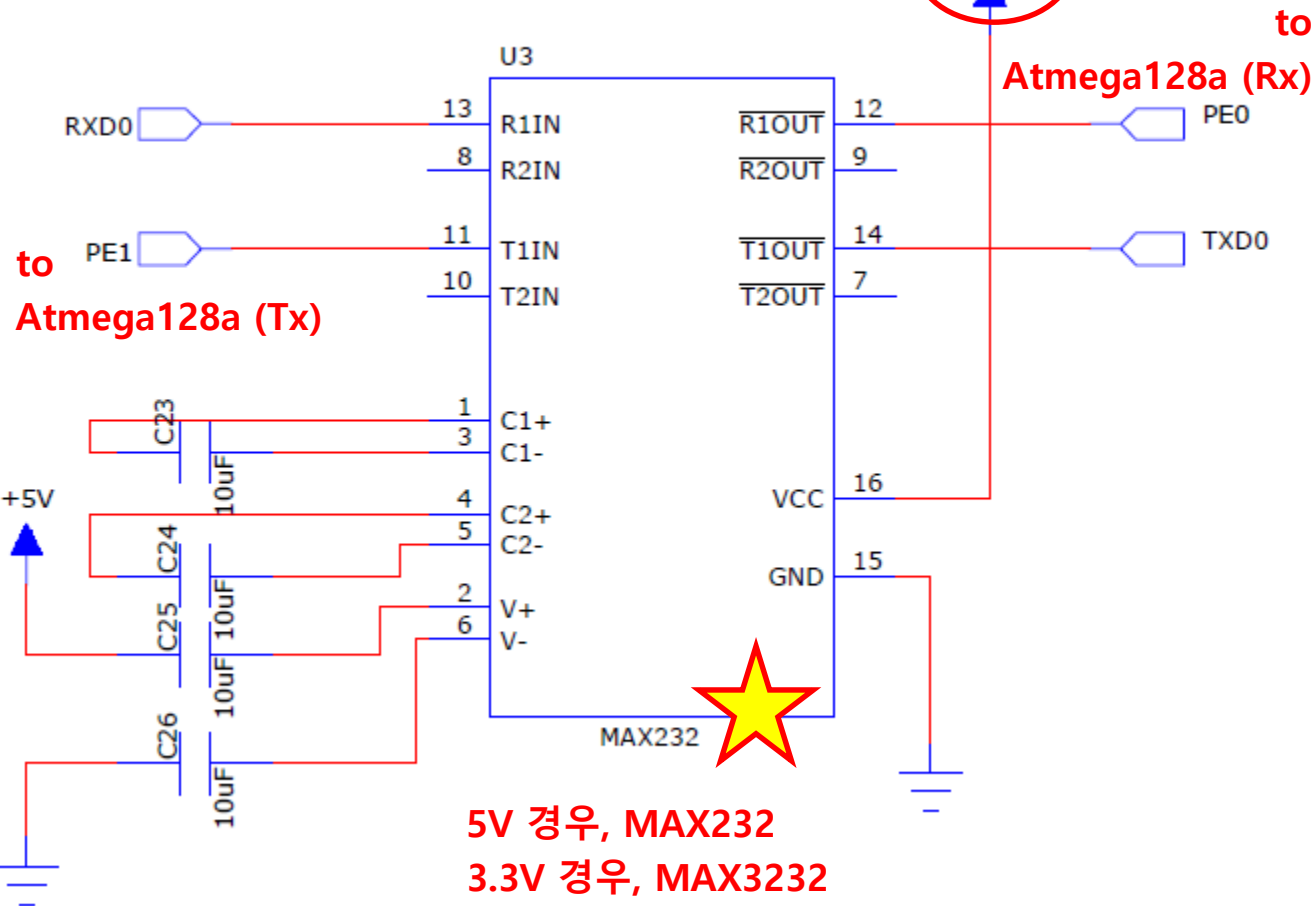
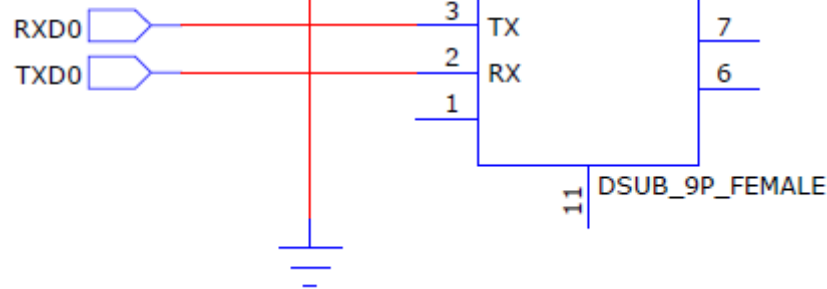


- ATmega128 : $V_{cc}(5V)$, GND를 1과 0 으로 사용
- 외부 UART RS-232 장치(PC)
 - 출력 : 출력 Tx 신호는 0,1을 $+12[V]$ 와 $-12[V]$ 로 사용
 - 입력 : 입력 Rx 신호는 $+3[V]$, $-3[V]$ 이상 일 때 0과1로 인식
- IC MAX232
 - Tx 신호는 ATmega128 신호를 입력 받아 $+7[V]$ ~ $-7[V]$ 로 변환

- Schematic

Strictly Private and Confidential

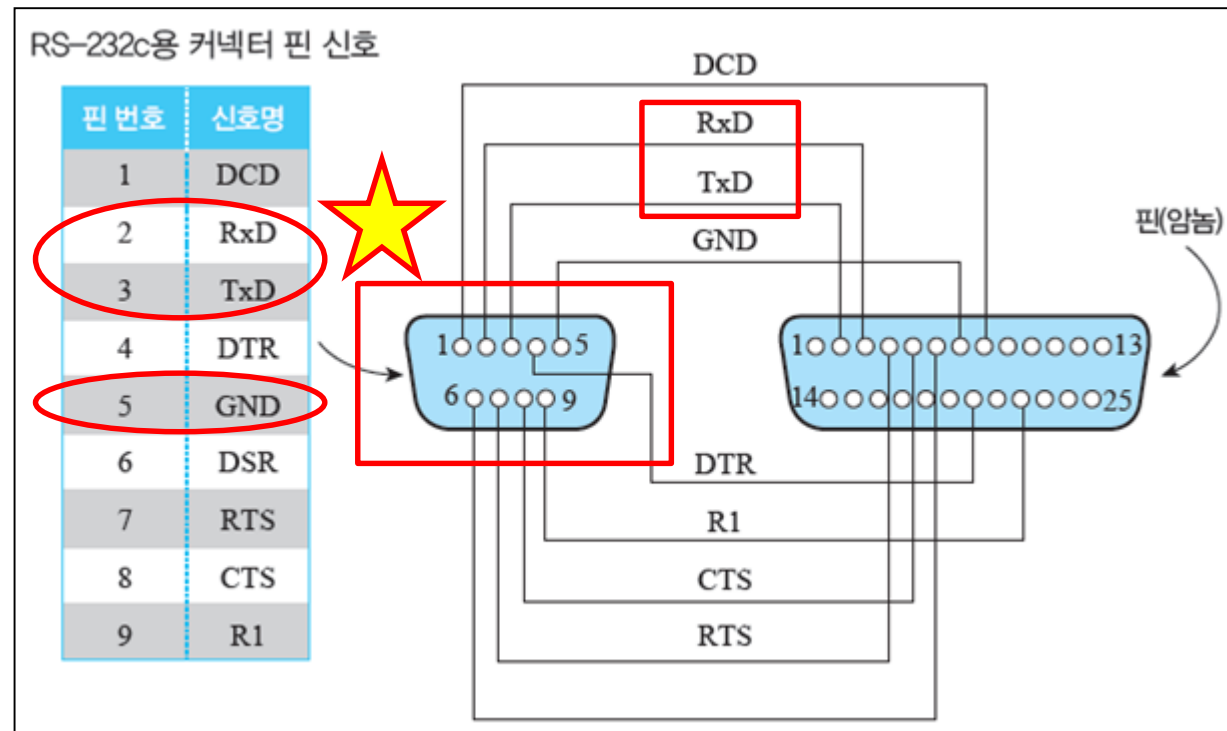
to PC



7. USART

UART(Universal Asynchronous Receiver and Transmitter)

- 컴퓨터에 내장된 USART의 송수신 신호는 TTL logic 레벨이기 때문에
원거리 통신을 위해 신호 레벨을 변경하고, RS-232C, RS-422, RS-485 규격을 사용.
- RS-232C는 전화선을 이용한 데이터 통신을 하기 위해
1962년에 미국의 EIA(Electronic Industries Association)에서 DTE(Data Terminal
Equipment)와 모뎀 등과 같은 DCE(Data Communication Equipment) 사이에
데이터 전송용으로
제정한 통신 규격임



7. USART

UART(Universal Asynchronous Receiver and Transmitter)

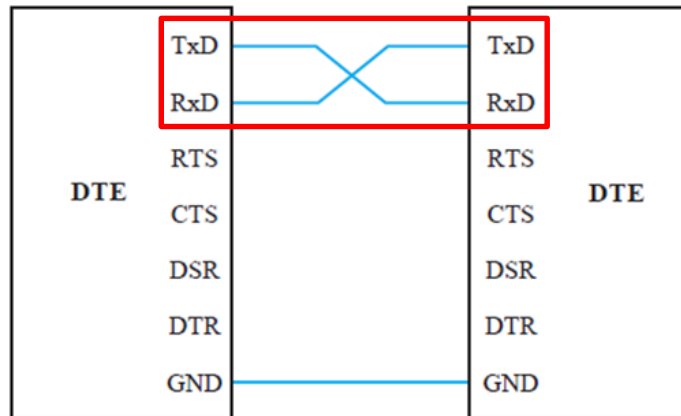
신호	설 명	입출력	기 능
TxD	Transmitter Data	출력	DTE에서 전송하는 시리얼 데이터
RxD	Receiver Data	입력	DTE에서 수신 받는 시리얼 데이터
$\overline{\text{DTR}}$	Data Terminal Ready	출력	DTE가 송수신 가능한 ready 상태임을 DCE에게 알리는 신호
$\overline{\text{DSR}}$	Data Set Ready	입력	DCE가 동작 가능한 상태임을 DTE에게 알리는 신호
$\overline{\text{RTS}}$	Request To Send	출력	DTE가 DCE에게 전송할 데이터 있음을 알리는 신호
$\overline{\text{CTS}}$	Clear To Send	입력	DCE가 신호를 받고 회선에 캐리어를 출력하여 데이터를 전송할 상태가 됐음을 DTE에게 알리는 신호
$\overline{\text{CD}}$	Data Carrier Detect	입력	상대방에서 전송되어 오는 캐리어를 검출하였음을 나타내는 신호
RI	Ring Indicator	입력	DCE가 회선에서 호출 신호를 받으면 active되는 신호

7. USART

UART(Universal Asynchronous Receiver and Transmitter)

- Data Terminal Equipment (DTE간 연결)**

: 가장 단순한 직렬 통신 연결, **15m 이하(RS-232)의** 근거리에서 연결



RS-422, RS-485 : 1.2Km

- 단 방향 전송(Simplex transmission)**

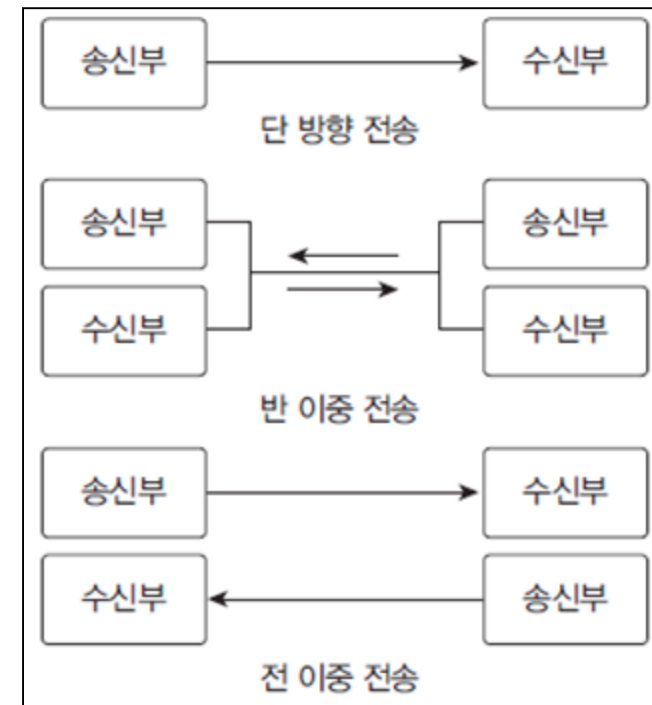
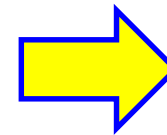
: 한 방향으로만 전송 가능

- 반 이중 전송(Half-duplex transmission) :**

: 1개 회선으로 양 방향 전송

- 전 이중 전송(Full-duplex transmission) :**

: **2개 회선으로 양 방향 전송**, 동시에 양 방향 전송이 가능함



7. USART

UART(Universal Asynchronous Receiver and Transmitter)

- ATmega128은 동기 및 비동기 전송모드에서
전이중 통신(송수신 동시가능)이 가능한
USART 2개를 내장

특징

: 전이중 Full-duplex transmission

: 높은 정밀도의

Baud Rate Generator 내장

: 5~9비트의 데이터 비트와

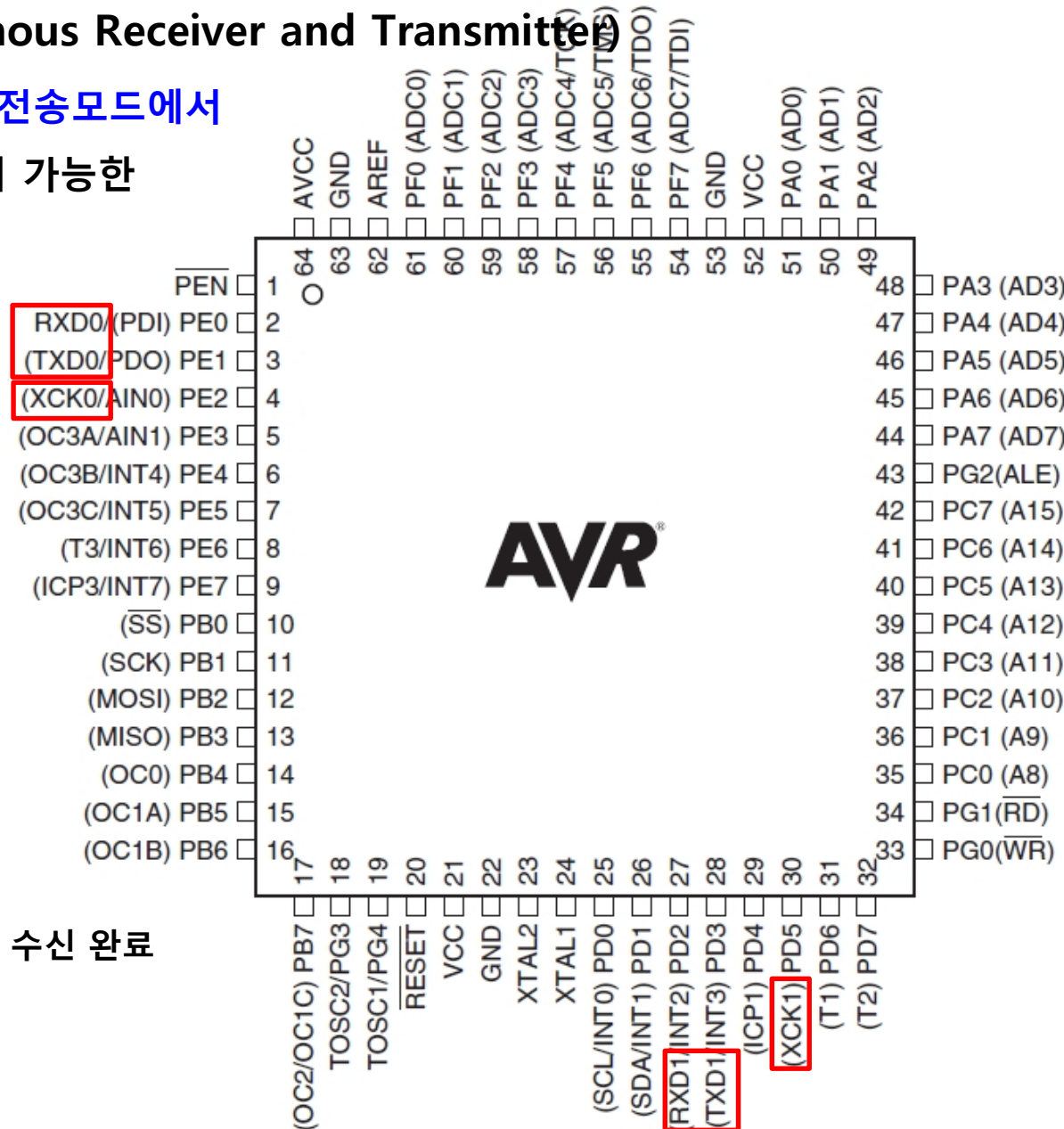
1~2비트의 스톱 비트 설정

: Noise filtering 내장

: 짝수 또는 홀수 Parity bit 설정

: 3개의 인터럽트 소스

➔ 송신 완료, 송신데이터 Empty, 수신 완료



7. USART

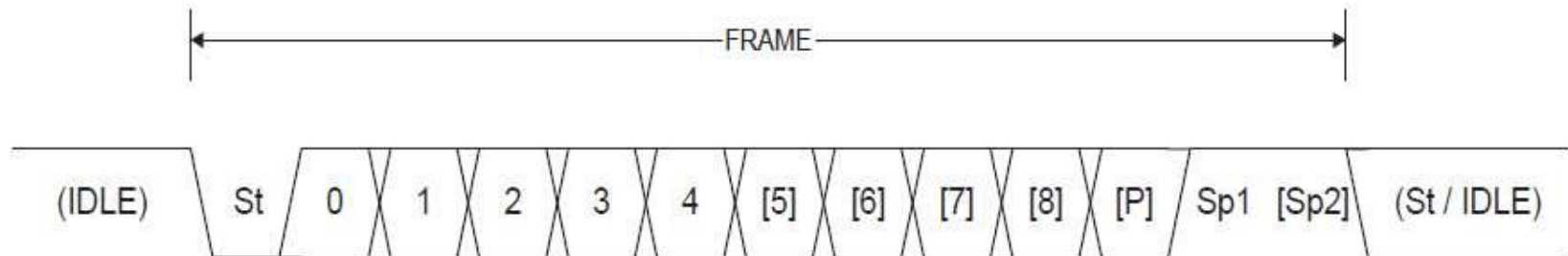
UART(Universal Asynchronous Receiver and Transmitter)

• 용어

: Baud, bps

Baud는 직렬 통신에서 전송 속도의 단위, **bps(bit per second)** 개념

: **Parity bit** - 통신 정상 전송 검출 bit (Data 전송 오류를 탐지하기 위한 bit)



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 (RxD or TxD). An IDLE line must be high.

7. USART

Parity bit

- 데이터 이전 과정에서 발생하는 오류(error)를 탐지하기 위해. 1개의 비트, 단어 또는 이보다 큰 단위를 기준으로 하여, 그 안에 포함된 2진 숫자인 0이나 1의 수를 동일하게 만들어 주는 것. 이러한 목적을 위해서 데이터에다 불필요한 0 또는 1을 첨가하는데 이를 패리티 비트라고 한다.
- 패리티에는 이븐 패리티(even parity)와 오드 패리티(odd parity)가 있다. 이븐 패리티에서는 한 단어 안에 있는 데이터를 구성하는 1의 수가 홀수이면, 패리티 비트가 1, 짝수이면 0이 된다. 오드 패리티에서는 반대로 한 단어 안의 1의 수가 짝수이면 패리티 비트가 1이 되고, 홀수이면 0이 된다. 즉, 이븐 패리티에서는 1의 수가 짝수가 되고, 오드 패리티에서는 1의 수가 홀수가 되도록 한다. 데이터 이전 과정에서 이 패리티의 상태를 점검하는 것을 패리티검사 (parity check)라고 한다. 이때, 패리티의 상태가 올바르지 않으면 패리티 오류 (parity error)가 발생한다.
- 패리티 비트 값은 다음 식에 의해 구해짐

$$P_{\text{even}} = d_{(n-1)} \oplus \dots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0 \text{ (1의 수가 홀수면 +1, 짝수면 +0)}$$

$$P_{\text{odd}} = d_{(n-1)} \oplus \dots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1 \text{ (1의 수가 짝수면 +1, 홀수면 +0)}$$

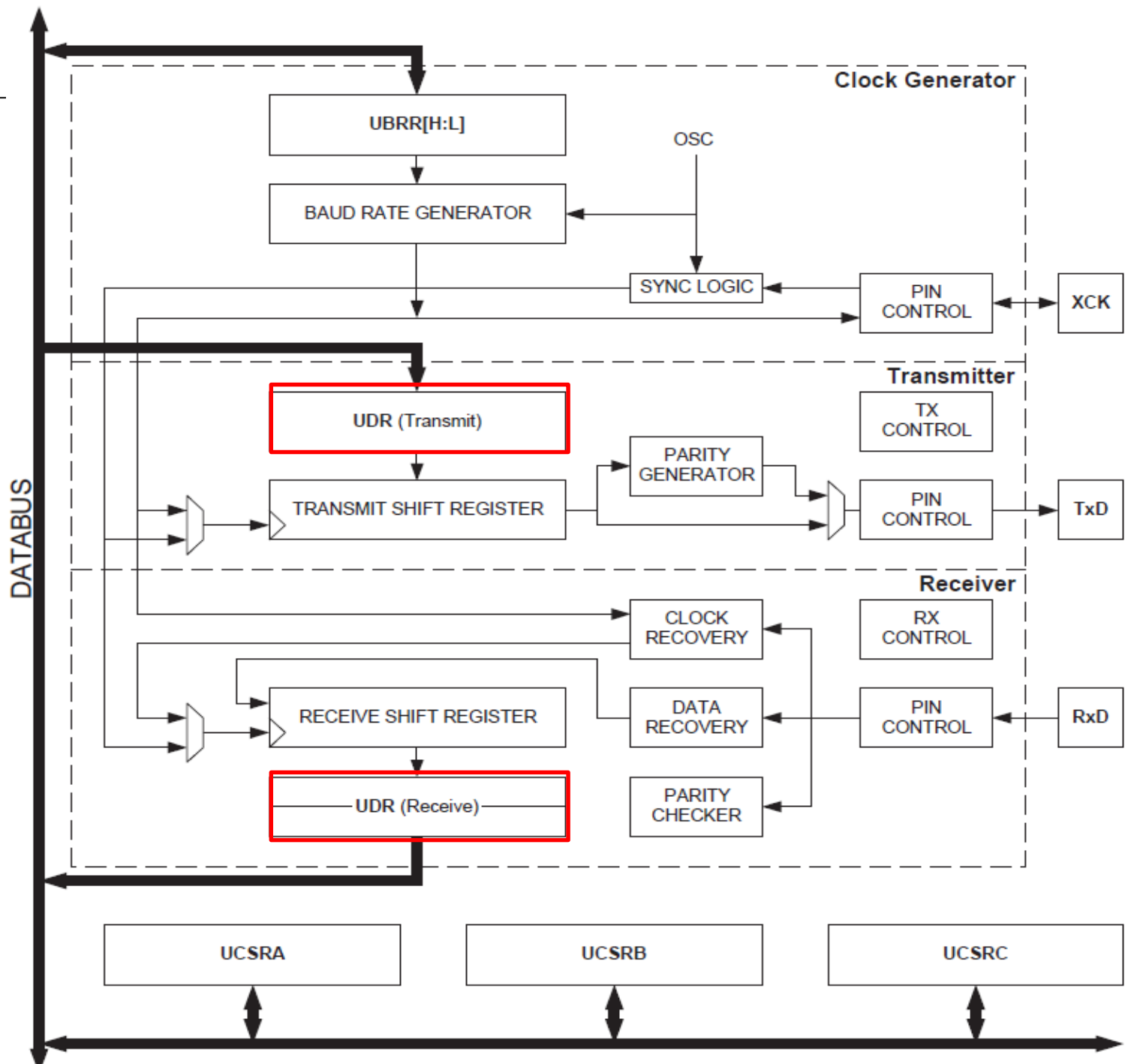
7. USART

UART(Universal Asynchronous Receiver and Transmitter)

- Register

설정 대상	선택 항목	초기 설정	설정	
			레지스터	비트명
통신 모드	동기 통신 모드, 비동기 통신 모드	비동기 통신 모드	UCSRnC	UMSELn
데이터 비트 수	5, 6, 7, 8, 9 비트	8비트	UCSRnB	UCSZn2
패리티 모드	홀수, 짝수, 없음	없음	UCSRnC	UCSZn1~UCSZn0
정지 비트 수	1, 2 비트	1비트	UCSRnC	UPMn1~UPMn0
클록 샘플링 위치 (동기 통신 모드인 경우)	상승에지, 하강에지	상승에지	UCSRnC	UCPOLn
클록 마스터/슬레이브 (동기 통신 모드인 경우)	마스터/슬레이브	슬레이브	DDRE DDR0	XCK0 XCK1
전송 속도	계산식에 의해 선택	알 수 없음	UBRRnH UBRRnL	UBRRn[11:8] UBRRn[7:0]
2배속 클록 (비동기 통신 모드인 경우)	1배속, 2배속	1배속	UCSRnA	U2Xn

7. USART



7. USART

UART(Universal Asynchronous Receiver and Transmitter)

- Register

: UDR₀, UCSR_{0A}, UCSR_{0B}, UCSR_{0C}, UBRR_{0H}, UBRR_{0L}

: UDR₁, UCSR_{1A}, UCSR_{1B}, UCSR_{1C}, UBRR_{1H}, UBRR_{1L}

Register	설 명
UDR _n	USART _n Data Register USART _n 전송/수신 데이터 레지스터
UCSR _{nA} , UCSR _{nB} , UCSR _{nC}	USART _n Control and Status Register A, B, C USART _n 전송/수신 동작 제어, 상태 기록
UBRR _{nH} , UBRR _{nL}	USART _n Baud Rate Register H, L

n = 0, 1

7. USART

- UDRn (USARTn Data Register)

n = 0, 1

: 전송/수신 데이터 버퍼 레지스터

: 전송/수신 데이터 버퍼가 동일하지만 내부적으로 별개의 레지스터

➔ 전송은 UDR에 **WRITE**, 수신은 UDR에서 **READ**

Bit	7	6	5	4	3	2	1	0	
	RXBn[7:0]								UDRn (Read)
	TXBn[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	

7. USART

- UCSRnA (USARTn Control and Status Register A)

n = 0, 1

bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREN	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
초기값	0	0	0	0	0	0	0	0	

[7] RXCn (USARTn Receive Complete)

: 수신 완료 상태를 나타내는 플래그 비트

: UDRn (수신 버퍼)에 새로운 데이터가 수신될 때 마다 RXCn→1이 됨

이 때 수신 완료 인터럽트(USARTn Receive Complete Interrupt)를 발생

UDRn을 READ하면 RXCn→0으로 CLEAR 됨

[6] TXCn (USARTn Transmit Complete)

: 전송 완료 상태를 나타내는 플래그 비트

: 전송 쉬프트 레지스터에서 데이터가 전송되고

UDRn(전송 버퍼)에 새로운 전송 데이터가 write 되지 않으면 TXCn→1이 됨

이 때, 전송 완료 인터럽트(USARTn Transmit Complete Interrupt) 발생

USARTn 전송 완료 인터럽트가 처리되면 자동적으로 TXCn→0

7. USART

• UCSRnA (USARTn Control and Status Register A)

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

n = 0, 1

[5] UDREn (USARTn Data Empty)

: UDRn(전송 버퍼)에 새로운 데이터를 받을 준비가 되어 있으면

UDREn→1이 되는 플래그 비트, **UDRn이 비어 있으면 UDREn→1**이 됨

: UDREn→1일 때 전송 데이터 **USARTn Data Register Empty Interrupt** 발생

: **UDRn에 새로운 데이터를 write** 하면 UDREn→0으로 clear 됨

[4] Fen (USARTn Frame Error)

: UDRn에 데이터를 수신하는 동안 프레임 에러가 발생하였음을 나타내는 플래그

: 프레임 에러는 수신 데이터의 첫 번째 정지 비트가 0일 때 발생

: FEn은 UDRn을 read 할 때까지 유효하며

UCSRnA 레지스터를 write 하면 FEn→0으로 clear

7. USART

• UCSRnA (USARTn Control and Status Register A)

bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREN	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
초기값	0	0	0	0	0	0	0	0	

n = 0, 1

[3] DORn (USARTn Data Over Run)

- : USARTn이 수신 동작할 때 **오버런** 에러를 나타내는 **플래그** 비트
- : **오버런** 에러란 **UDRn에 read 하지 않은 수신 데이터가 있는 상태에서**
수신 쉬프트 레지스터(Receive Shift Register)에서 새로운 데이터 수신이 완료되고
다음 수신 데이터의 시작 비트가 검출될 때 발생
- : UDRn을 read 때까지 유효하며 UCSRnA 레지스터를 write하면 DORn→0 clear

[2] UPEn (USARTn Parity Error)

- : USARTn 패리티 에러 비트
- : 수신하는 데이터에서 **패리티 에러가 발생하면 UPEn→1**
- : UDRn을 read 때까지 유효하며 UCSRnA 레지스터를 write 하면 UPEn→0 clear

7. USART

• UCSRnA (USARTn Control and Status Register A)

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

n = 0, 1

[1] U2Xn (USARTn Double Transmission Speed)

- : U2Xn은 **비동기 모드에서만 사용 가능한 비트**
- : 비동기 통신에서 **U2Xn=1이면 USARTn의 통신 속도가 2배로 빨라짐**
- : 실제로는 USARTn의 clock 분주비를 16에서 8로 하여 통신 속도를 2배로 높임

[0] MPCMn (Multi-Processor Communication Mode)

- : USARTn을 **멀티-프로세서 통신** 모드로 설정하는 비트
- : 멀티-프로세서 통신 모드에서 어드레스 정보를 포함하지 않는 수신 데이터는 수신부에서 무시됨

7. USART

• UCSRnB (USARTn Control and Status Register B)

n = 0, 1

bit	7	6	5	4	3	2	1	0	
	RXCIE _n	TXCIE _n	UDRIE _n	RXEN _n	TXEN _n	UCSZ _{n2}	RXB8 _n	TXB8 _n	UCSR _n B
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
초기값	0	0	0	0	0	0	0	0	

[7] RXCIE_n (RX Complete Interrupt Enable)

: 수신 완료 인터럽트 enable 비트

수신 완료 상태를 나타냄

: **RXCIE_n=1**, SREG 레지스터의 **I=1**이면 UCSR_nA 레지스터의 [7] **RXC_n→1** 일 때
수신 완료 인터럽트가 발생하고 처리됨

[6] TXCIE_n (TX Complete Interrupt Enable)

: 전송 완료 인터럽트 enable 비트

전송 완료 상태를 나타냄

: **TXCIE_n=1**, SREG 레지스터의 **I=1**이면 UCSR_nA 레지스터의 [6] **TXC_n→1**일 때
전송 완료 인터럽트가 발생하고 처리됨

7. USART

bit	7	6	5	4	3	2	1	0	
	RXCIE _n	TXCIE _n	UDRIE _n	RXEN _n	TXEN _n	UCSZ _{n2}	RXB8 _n	TXB8 _n	UCSR _n B
Read/Write 초기값	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	

- UCSR_nB (USART_n Control and Status Register B)

[5] UDRIE_n (Data Register Empty Interrupt Enable)

- : 전송 데이터 레지스터 Empty 인터럽트 Enable 비트
- : UDRIE_n=1, SREG 레지스터의 I=1이면 UCSR_nA 레지스터의 [5] UDRE_n→1일 때
새로운 data를 받을 준비가 됨
- USART_n 전송 데이터 레지스터 Empty 인터럽트가 발생하고 처리됨

[4] RXEN_n (Receiver Enable)

- : 수신부가 동작하도록 Enable 하는 비트
- : RXEN_n=1이면 RxD_n 핀이 I/O 포트가 아닌 수신 단자로 사용 가능

[3] TXEN_n (Transmitter Enable)

- : 전송부가 동작하도록 Enable 하는 비트
- : TXEN_n=1이면 TXD_n 핀이 I/O 포트가 아닌 전송 단자로 사용 가능

[2] UCSZ_{n2} (Character Size)

- : 통신 데이터 크기를 설정하는 비트
- : UCSR_nC 레지스터의 UCSZ_{n1}, UCSZ_{n0}와 같이 통신 데이터 크기를 설정하는 비트

7. USART

- **UCSRnB (USARTn Control and Status Register B)**

[1] RXB8n (Receive Data Bit 8)

: 통신 데이터가 9비트일 때 9번째 비트(MSB)를 수신 받는 비트

: UDRn (수신 버퍼)을 read 하기 전에 반드시 RXB8n을 먼저 read해야 함

[0] TXB8n (Transmit Data Bit 8)

: 통신 데이터가 9비트일 때 9번째 비트(MSB)를 전송하는 비트

: UDRn(수신 버퍼)에 write 하기 전에 반드시 TXB8n을 먼저 write해야 함

7. USART

- UCSRnC (USARTn Control and Status Register C)

n = 0, 1

bit	7	6	5	4	3	2	1	0	
	-	UMSELn	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	
초기값	0	0	0	0	0	0	0	0	

[6] UMSELn (USART Mode Select) : USART 모드 설정 비트

- 통신 모드 설정 비트
- 0이면 비동기 모드, 1이면 동기 모드가 설정

[5][4] UPMn (USART Parity Mode) : 패리티 모드 설정 비트

- 표와 같이 패리티 모드를 설정

UPMn1	UPMn0	USARTn Parity 모드
0	0	패리티 사용 안함
0	1	-
1	0	짝수(even) 패리티로 설정
1	1	홀수(odd) 패리티로 설정

7. USART

- UCSRnC (USARTn Control and Status Register C)

n = 0, 1

bit	7	6	5	4	3	2	1	0	
	-	UMSELn	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
Read/Write 초기값	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	

[3] USBSn (Stop Bit Select) : 정지 비트 선택

- 0이면 1비트, 1이면 2비트가 설정

UCSRnB [2], UCSRnC [2][1] UCSZn1, UCSZn0(Character Size)

- 전송 데이터 비트 수 설정

UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5비트
0	0	1	6비트
0	1	0	7비트
0	1	1	8비트
1	0	0	-
1	0	1	-
1	1	0	-
1	1	1	9비트

7. USART

- UCSRnC (USARTn Control and Status Register C)

n = 0, 1

bit	7	6	5	4	3	2	1	0	
	-	UMSELn	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	
초기값	0	0	0	0	0	0	0	0	

[0] UCPOLn (Clock Polarity)

: UCPOLn은 동기 모드 통신에서 전송 데이터의 출력 시점과

수신 데이터의 입력 시점을 설정하는 비트

: UCPOLn=0일 때 전송 데이터는

XCKn clock의 rising edge에서 TXDn 핀으로부터 출력되고

수신 데이터는 XCKn clock의 falling edge에서 RXDn 핀으로 입력됨

: UCPOLn=1일 때는 반대로 됨

7. USART

- UBRRnH, UBRRnL (USARTn Baud Rate Register)

$n = 0, 1$

[11]~[0] USART Baud Rate Register

: 12개의 비트에 의해서 전송속도가 설정

: 상위 바이트 값 (UBRRnH)를 먼저 쓰고 하위 바이트 값 (UBRRnL)을 나중에.

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRRn[11:8]				UBRRnH
	UBRRn[7:0]								UBRRnL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	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	
	0	0	0	0	0	0	0	0	

7. USART

- UBRRnH, UBRRnL (USARTn Baud Rate Register)

n = 0, 1

: 16MHz 에서 전송속도에 따른 UBRRn 레지스터의 설정 값

전송속도	비동기 일반 모드		비동기 2배속 모드	
(Baud Rate : bps)	UBRR	에러	UBRR	에러
2,400	416	-0.1%	832	+0.0%
4,800	207	+0.2%	416	-0.1%
9,600	103	+0.2%	207	+0.2%
14.4K	68	+0.6%	103	-0.1%
19.2K	51	+0.2%	68	+0.2%
28.8K	34	-0.8%	51	+0.6%
38.4K	25	+0.2%	34	+0.2%
57.6K	16	+2.1%	25	-0.8%
76.8K	12	+0.2%	16	+0.2%
115.2K	8	-3.5%	12	+2.1%
230.4K	3	+8.5%	8	-3.5%
250K	3	+0.0%	3	+0.0%
0.5K	1	+0.0%	3	+0.0%
1M	0	+0.0%	1	+0.0%

7. USART

- UBRRnH, UBRRnL (USARTn Baud Rate Register)

n = 0, 1

: 전송속도 및 설정해야 할 UBRRn 레지스터 값 계산식

동작 모드	전송속도 계산식(bps)	UBRR 계산식
비동기 정상 모드(U2Xn=0)	$BAUD = \frac{f_{osc}}{16(UBRRn+1)}$	$UBRRn = \frac{f_{osc}}{16 \cdot BAUD} - 1$
비동기 2배속 모드(U2Xn=1)	$BAUD = \frac{f_{osc}}{8(UBRRn+1)}$	$UBRRn = \frac{f_{osc}}{8 \cdot BAUD} - 1$
동기 마스터 모드	$BAUD = \frac{f_{osc}}{2(UBRRn+1)}$	$UBRRn = \frac{f_{osc}}{2 \cdot BAUD} - 1$

7. USART

UART(Universal Asynchronous Receiver and Transmitter)

- 16MHz 시스템 clock 을 갖는 경우의 전송속도에 따른 UBRRn 레지스터의 설정
- 비동기 통신 모드에서 전송속도에 대한 오차범위는 8비트 일반 모드의 경우 $\pm 2.0\%$ 이며, 2배속 모드의 경우 $\pm 1.5\%$
- ATmega128의 USART clock 발생부는 데이터의 송신과 수신을 위해 기본 clock을 발생하며, 비동기 일반모드, 비동기 2배속 모드, 동기 마스터 모드, 동기 slave 모드 지원
- USART 제어 및 상태 레지스터 UCSRnC의 UMSEL 비트는 동기모드와 비동기 모드 선택
- 2배속 모드는 UCSRnA 레지스터의 U2X비트에 의해 제어
- 동기모드에서(UMSEL = 1) XCKn 핀에 대한 데이터 방향 레지스터의 값은 클럭 소스가 내부(마스터 모드)에 있는지? 외부에 있는지(slave모드)에 따라 제어
- XCKn핀은 동기모드인 경우에만 동작

USART 사용법(초기화)

1. 통신 모드 설정

- ➔ 동기/ 비동기, 송신 Enable, 수신 Enable,
패리티 비트, 스톱 비트, 데이터 비트 수, 인터럽트 Enable 비트
- ➔ 사용 레지스터 : UCSRnA, UCSRnA, UCSRnC [n=0, 1]

2. 보레이트 (전송 속도) 설정

- ➔ 사용 레지스터 : UBRRnL, UBRRnH [n=0, 1]

3. 전역 인터럽트 Enable 비트 I set (인터럽트 사용시에만 설정)

- ➔ 사용 레지스터 : SREG(최상위 비트 I)

USART 사용법(송.수신)

[6] 전송 완료를 나타냄 =0

<송신 동작>

[5] 새로운data를 받을 준비가 됨 =1

- UCSRnA 레지스터의 **TXCn** 또는 **UDREN** 비트 대기
- 전송할 데이터를 UDRn 레지스터에 설정(write) 송신

※ 송신은 인터럽트 방식보다 **polling** 방식을 이용하는 것이 편리



<수신 동작>

[7] 수신 완료를 나타냄 =1

- UCSRnA 레지스터의 **RXCn** 비트가 set 될 때까지 대기
- 전송된 데이터를 UDRn 레지스터로부터 읽음(수신 동작)

※ 수신은 언제 데이터가 전송되어 올지 모르기 때문에

polling 방식 보다는 **인터럽트** 방식을 이용하는 것이 편리



Pointer 변수

일반 변수

프로그램에서 사용하는 데이터를 저장

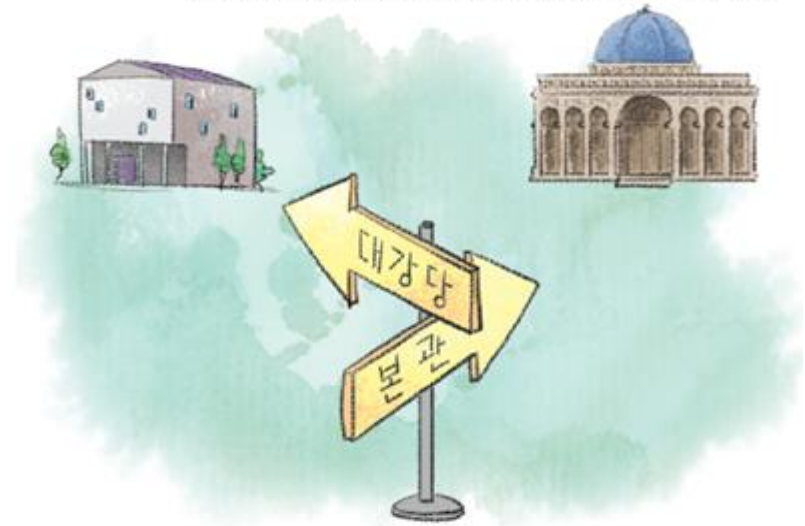
포인터 변수

데이터가 저장된
주기억장치의 **주소만 저장**
(간단히 포인터(pointer)라고도 함)

포인터의 장점

- 직접 참조할 수 없는 변수를 포인터를
이용 간접적으로 참조 가능
- 기억 공간의 효율적 사용
- 프로그램 성능 개선

데이터가 저장된 곳을 알려주는 포인터 변수



Pointer 변수

- 포인터 변수도 일반 변수와 같이 선언 하고 사용
- 포인터 변수는 이름 앞에 간접 참조 연산자^{indirection operator}인 * 를 붙여 선언

asterisk

```

01 #include <stdio.h>
02
03 int main()
04 {
05     int max, *pmax;
06     max = 100;
07     pmax = &max;
08     printf("변수 max의 값은 %d\n", max);
09     printf("변수 max의 주소는 %p\n", &max);
10     printf("포인터 변수 pmax의 값(주소)은 %p\n", pmax);
11 }

```

--- 일반 변수와 함께 포인터 변수 선언

--- 포인터 변수에 일반 변수의 주소 배정

--- 포인터 변수의 값(주소) 출력

변수 max의 값은 100

변수 max의 주소는 00120000

포인터 변수 pmax의 값(주소)은 00120000

문자열(string)

- 문자열^{string}은 연속된 문자들의 집합으로 프로그램에서 가장 많이 사용되는 요소
 - 문자 : 1개의 글자로 작은따옴표(예 : 'a')로 표시
 - 문자열 : 하나 이상의 연속된 문자로 구성되며 큰따옴표(예 : "a", "university")로 표시하고 문자열은 끝에 널 null 문자(\0)가 반드시 있다



10	HEX	문자	10	HEX	문자	10	HEX	문자	10	HEX	문자	10	HEX	문자	10	HEX	문자
0	0x00	NULL	22	0x16	STN	44	0x2C	,	66	0x42	B	88	0x58	X	110	0x6E	n
1	0x01	SOH	23	0x17	ETB	45	0x2D	-	67	0x43	C	89	0x59	Y	111	0x6F	o
2	0x02	STX	24	0x18	CAN	46	0x2E	.	68	0x44	D	90	0x5A	Z	112	0x70	p
3	0x03	ETX	25	0x19	EM	47	0x2F	/	69	0x45	E	91	0x5B	[113	0x71	q
4	0x04	EOT	26	0x1A	SUB	48	0x30	0	70	0x46	F	92	0x5C	₩	114	0x72	r
5	0x05	ENQ	27	0x1B	ESC	49	0x31	1	71	0x47	G	93	0x5D]	115	0x73	s
6	0x06	ACK	28	0x1C	FS	50	0x32	2	72	0x48	H	94	0x5E	^	116	0x74	t
7	0x07	BEL	29	0x1D	GS	51	0x33	3	73	0x49	I	95	0x5F	_	117	0x75	u
8	0x08	BS	30	0x1E	RS	52	0x34	4	74	0x4A	J	96	0x60	`	118	0x76	v
9	0x09	HT	31	0x1F	US	53	0x35	5	75	0x4B	K	97	0x61	a	119	0x77	w
10	0x0A	₩n	32	0x20	SP	54	0x36	6	76	0x4C	L	98	0x62	b	120	0x78	x
11	0x0B	VT	33	0x21	!	55	0x37	7	77	0x4D	M	99	0x63	c	121	0x79	y
12	0x0C	FF	34	0x22	"	56	0x38	8	78	0x4E	N	100	0x64	d	122	0x7A	z
13	0x0D	₩r	35	0x23	#	57	0x39	9	79	0x4F	O	101	0x65	e	123	0x7B	{
14	0x0E	SO	36	0x24	\$	58	0x3A	:	80	0x50	P	102	0x66	f	124	0x7C	
15	0x0F	SI	37	0x25	%	59	0x3B	;	81	0x51	Q	103	0x67	g	125	0x7D	}
16	0x10	DLE	38	0x26	&	60	0x3C	<	82	0x52	R	104	0x68	h	126	0x7E	~
17	0x11	DC1	39	0x27	'	61	0x3D	=	83	0x53	S	105	0x69	i	127	0x7F	DEL
18	0x12	DC2	40	0x28	(62	0x3E	>	84	0x54	T	106	0x6A	j			
19	0x13	DC3	41	0x29)	63	0x3F	?	85	0x55	U	107	0x6B	k			
20	0x14	DC4	42	0x2A	*	64	0x40	@	86	0x56	V	108	0x6C	l			
21	0x15	NAK	43	0x2B	+	65	0x41	A	87	0x57	W	109	0x6D	m			

문자열(string)

```
char s1[4];
s1[0] = 'k';
s1[1] = 'i';
s1[2] = 'm';
s1[3] = '\0';
```

--- 문자 배열을 선언하고 배열의 각 요소 단위로 초기화

```
char s2[4] = "kim";
```

--- 선언과 초기화를 한 문장으로 지정

```
char s3[4] = "park";
```

--- 오류 발생. 배열의 크기가 문자보다 1이 커야 함(널 문자)

```
char s4[10] = "korea";
```

--- 문자 다섯 자리 뒤는 모두 널 문자

```
char s5[] = "korea";
```

--- 크기가 6인 문자 배열 생성

```
char s6[][6] = {"korea","usa","china","japan"};
```

--- 크기가 4*6인 문자 배열 생성

- C 언어는 문자(ctype.h)와 문자열(string.h) 처리를 위한 함수를 제공

포인터를 통한 문자열의 사용

- 문자열을 사용하기 위해 문자 배열을 사용하는 방법과 문자열 포인터를 사용하는 방법

```
01 char city1[] = "Seoul"; --- 문자 배열로 문자열 선언
02 char *city2 = "Seoul"; --- 문자열 포인터로 문자열 선언
```

- 두 방법의 차이

```
01 char city1[] = "Seoul"; --- 문자 배열로 문자열 선언
02 char *city2 = "Seoul"; --- 문자열 포인터로 문자열 선언
03 city1 = "Busan"; --- 오류 발생. 새로운 문자열로 문자 배열 변경 불가
04 city2 = "Busan"; --- 문자열 포인터 값 변경 가능
```

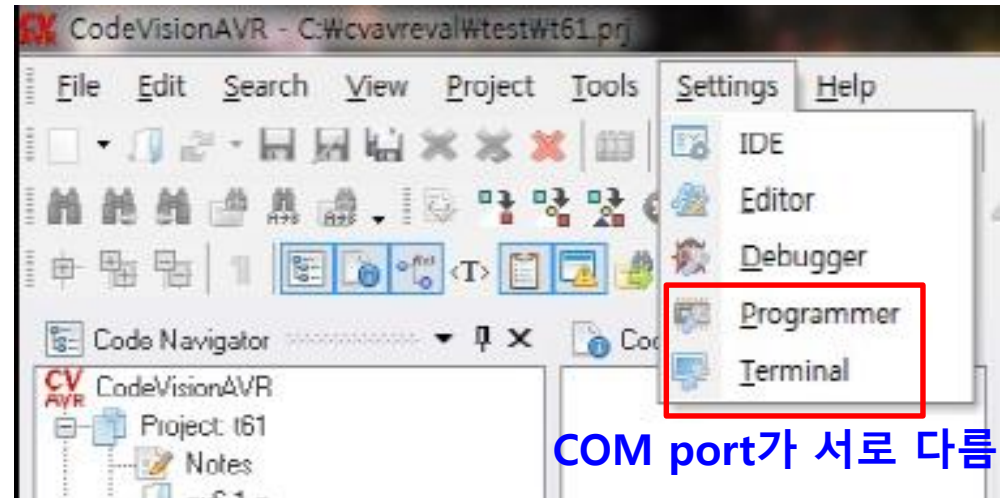
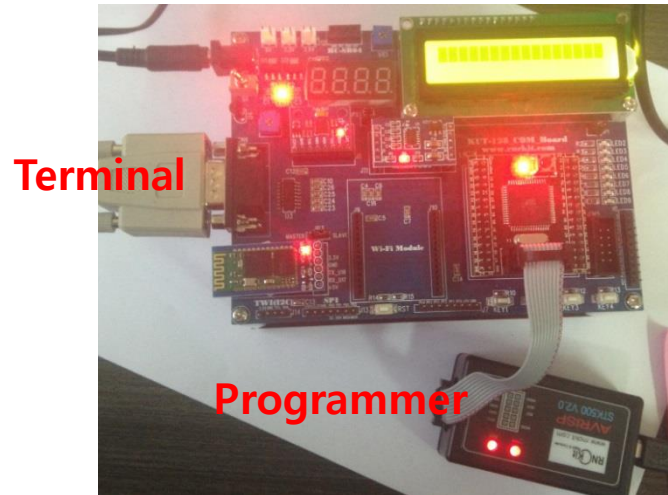

포인터를 통한 문자열의 사용

■ 문자열을 사용하기 위해 문자 배열을 사용하는 방법과 문자열 포인터를 사용하는 방법

01 char city1[] = "Seoul";	--- 문자 배열 생성
02 char *pcity1 = city1;	--- 배열 주소로 문자 포인터 pcity1 생성
03 char *pcity2 = "Seoul";	--- 문자열 포인터 pcity2 생성
04 pcity2 = "Busan";	--- pcity2를 새로운 문자열로 변경
05 printf("%s\n", pcity1);	--- Seoul 출력
06 printf("%s\n", pcity2);	--- Busan 출력
07 printf("%c\n", *pcity1);	--- 첫 번째 문자 S 출력
08 printf("%c\n", *pcity2);	--- 첫 번째 문자 B 출력
09 printf("%c\n", *(pcity1+1));	--- 두 번째 문자 e 출력
10 printf("%c\n", *(pcity2+1));	--- 두 번째 문자 u 출력

Code Vision AVR 터미널설정

- PC와 KUT-128_COM 보드를 RS-232 케이블로 연결

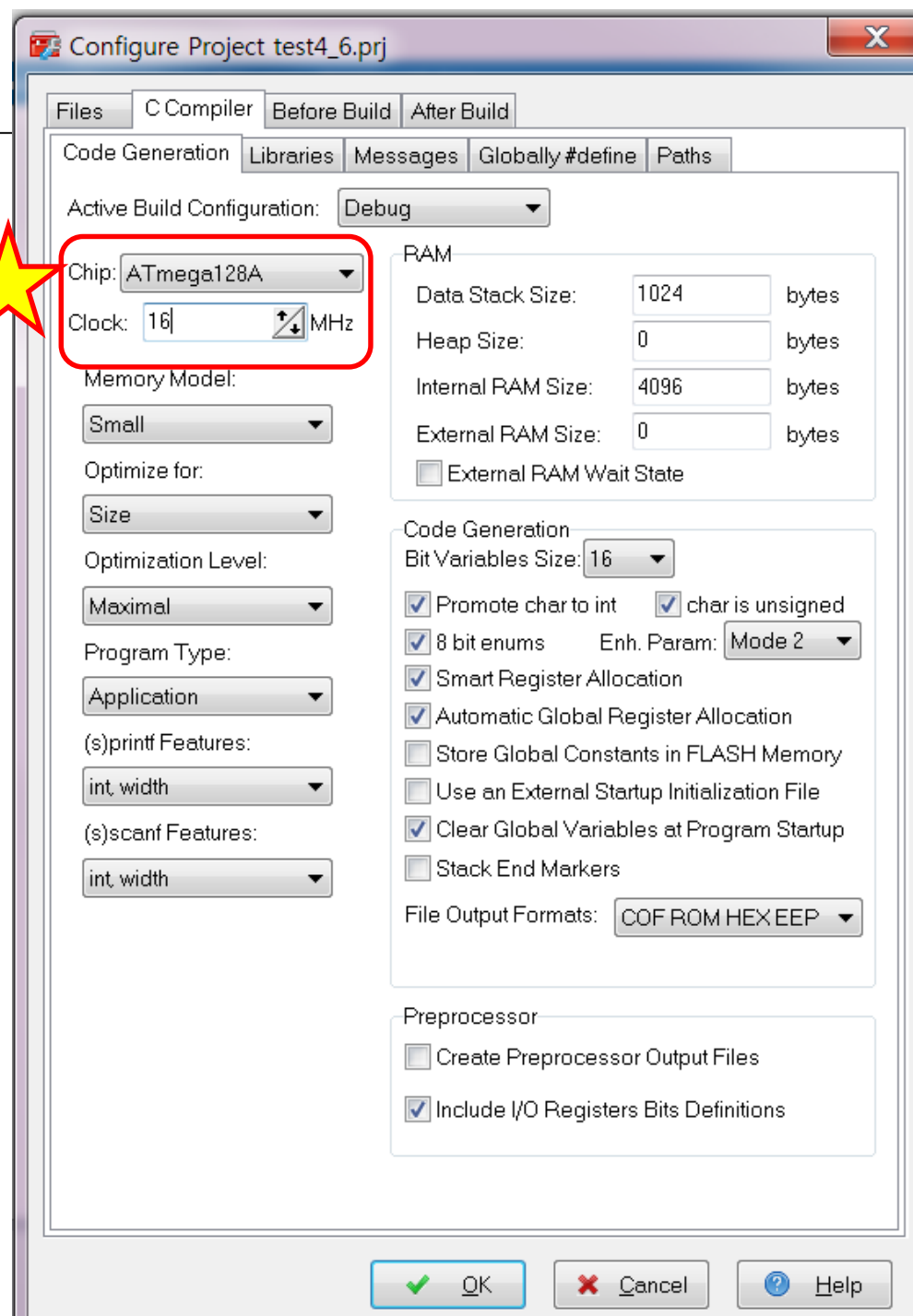


- 장치 관리자
 - : 포트에 할당 된 COM번호와 Baud rate를 확인
 - : 통신포트 COM1 - Terminal
 - : USB Serial Port(COM3) – Programmer
- Baud rate는 기본 (9600) 그대로 사용

노트북 :
USB to serial
Converter

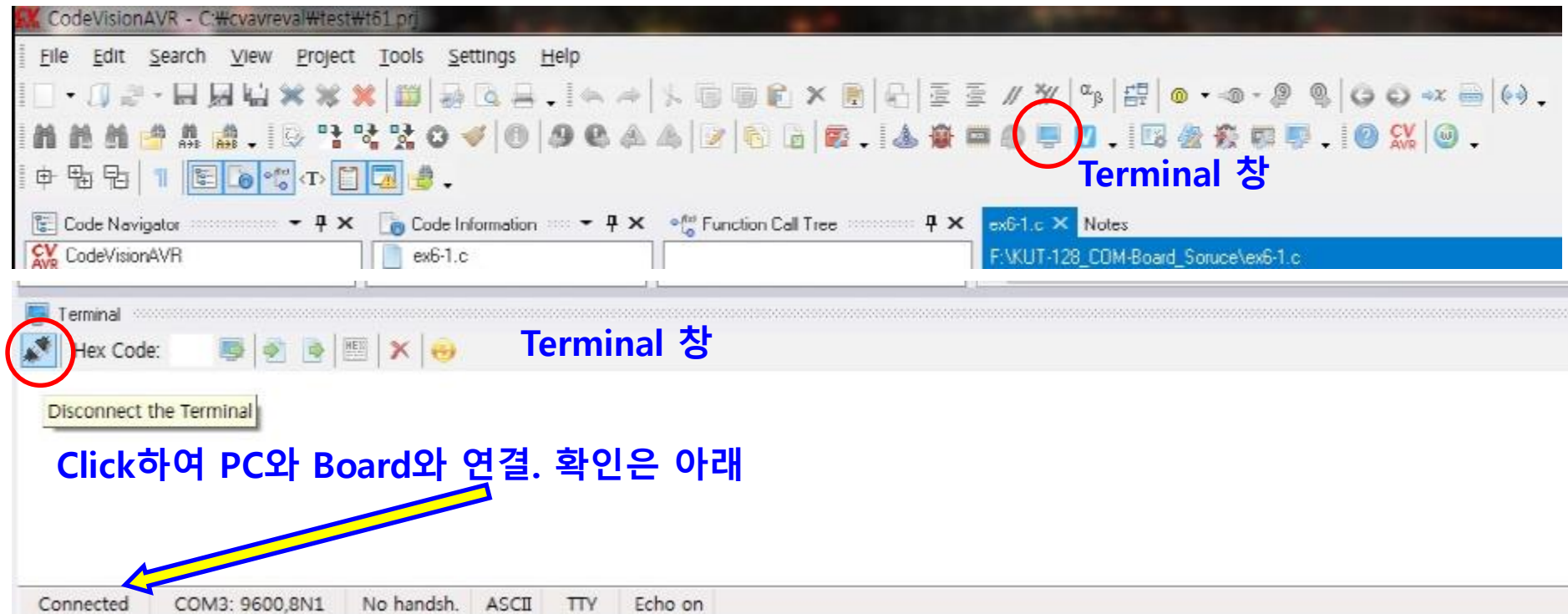


- Project 생성시,
C-Compiler에서
clock 8MHz를 16MHz로



Code Vision AVR 터미널설정

- Code Vision AVR을 실행하고 <Settings → Teminal> 통신환경 설정
 - PC 장치관리자에 할당된 포트와 Baud rate 를 동일하게 설정




통신 Port : COM3, Baud rate : 9600, 8bits data, Parity 없음, Stop bit 1



[예제 7-1] USART0 송신 실험

KUT-128_COM보드의 USART0을 통해서 문자열

“This is USART control program”을 컴퓨터에 전송하는 프로그램을 작성하라
전송속도는 9600bps로 하고, 정지 비트는 1비트로 한다.

전송된 문자열은 CodeVisionAVR에 내장된 터미널 창을 통해서 확인할 수 있으며, 툴바의  아이콘을 클릭하면 터미널 창이 열린다.

한 번만 실행되므로, MCU보드의 리셋 스위치를 눌러가며 동작을 확인한다.

BAUD = 9600, DataBit = 8 bit, StopBit = 1, Parity = None;



1. RS232 연결, 장치관리자에서 COM Port 확인
2. Code vision에서 Setting에서 Programmer와 Terminal 연결 (COM Port)
3. 프로그램
4. Terminal 창 열고 메시지 확인, 못 볼 경우에는 Board의 power switch를 off-on

```
#include <mega128.h>
```

```
#include <delay.h>
```

```
void Putch(char); // 송신
```

```
void main(void)
```

```
{
```

```
    char string[] = "This is USART0 control program. ^_____^"; // 전송 문자열
```

```
    char *pStr; // 문자열 포인터
```

```
    delay_ms(7000); // PC에서 메시지 확인 위해
```

```
    UCSR0A = 0x0; // USART 초기화 (251p)
```

```
    UCSR0B = 0b00001000; // 송신 인에이블 TXEN = 1 (253p)
```

```
    UCSR0C = 0b00000110; // 비동기[7], 데이터 8비트 모드 (253,4p)
```

```
    UBRR0H = 0; // X-TAL = 16MHz 일때, BAUD = 9600
```

```
    UBRR0L = 103; // (257p)
```

```
    pStr = string;
```

```
    while(*pStr != 0) Putch(*pStr++); // 문자열 전송
```

```
    while(1);
```

```
}
```

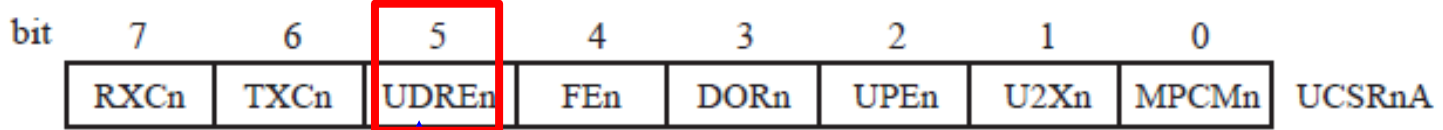
```
void Putch(char data) // 한 바이트 송신
```

```
{
```

```
    while((UCSR0A & 0x20) == 0x0); //UDRE0[5] = 1 송신준비완료 될 때까지 대기
```

```
    UDR0 = data; // 데이터 전송
```

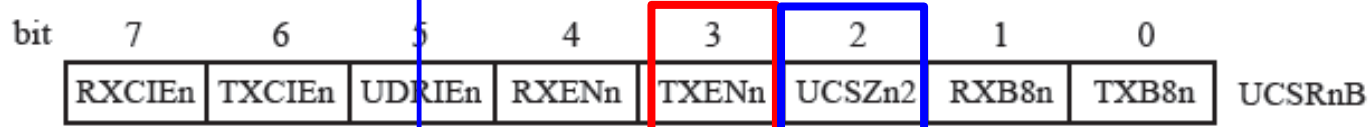
```
}
```



송신준비완료 될 때까지 대기

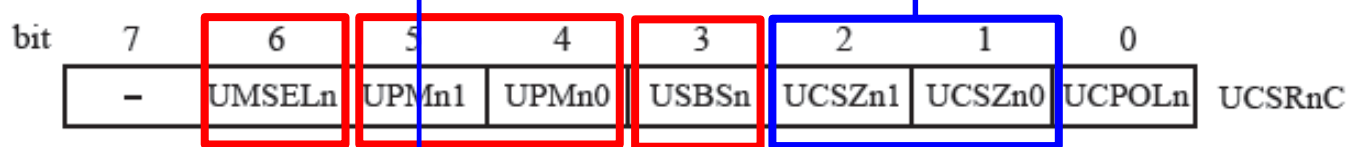
UCSR0A = 0x0;

// USART 초기화



UCSR0B = 0b00001000;

// 송신부 enable TXEN = 1



UCSR0C = 0b00000110;

// 비동기:0, No parity: 00, 정지bit 1이면:0, 데이터 8비트 모드

UBRR0H = 0;

// X-TAL = 16MHz 일때, BAUD = 9600

UBRR0L = 103;

pStr = string;

while(*pStr != 0) Putch(*pStr++);

// 문자열 전송, 마지막 null문자

while(1);

}

void Putch(char data)

// 한 바이트 송신

{

while((UCSR0A & 0x20) == 0x0);

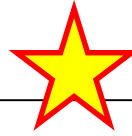
// 송신 버퍼가 비어지면 UDRE0 = 1 이 될 때까지 대기
전송 준비가 되었는지 확인하는 것

UDR0 = data;

// 데이터 전송

}

[예제 7-2] USART0 수신 실험



Code Vision AVR에서 터미널 창을 오픈 후

컴퓨터에서 key 입력된 16진 문자 ('0'~'9','A'~'F' 또는 'a'~'f')를

KUT-128_COM보드에서 수신하여

맨 우측의 7-세그먼트에 출력 표시하는 프로그램을 작성하라.

실험할 때는 <Terminal Settings> 창에서 “Echo Transmitted Character”의 선택을 해제하고 실행한다.

입력된 Key가 terminal창에서 2번씩 표시됨

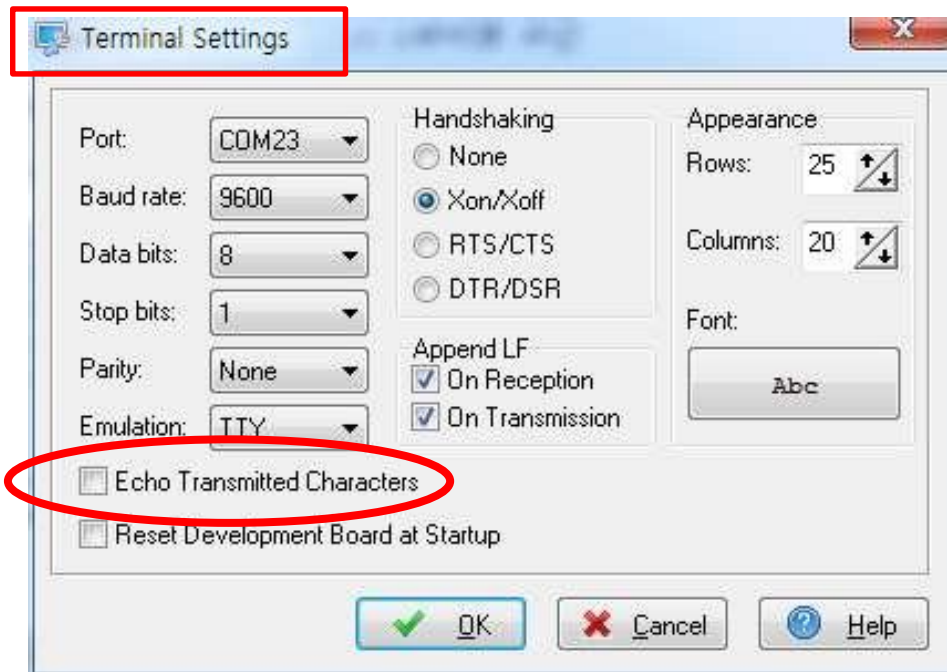
BAUD = 9600,

Data Bit = 8 bits,

Stop Bit = 1,

Parity = None

입력 ASCII를 배열 값으로 어떻게 변환할까?



[예제 7-2] USART0 수신 실험

입력 ASCII를 Hex **배열**로 어떻게 변환할까?

배열 : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, b, c, d, E, F

0, ~ ,9 : $0x30 \leq \boxed{rd} \leq 0x39$ 경우, $dd = rd - 0x30$

$\text{if}((rd \geq '0') \ \&\& \ (rd \leq '9')) \ dd = rd - '0'$

예, $rd = '7'$ 경우, $dd = 0x37 - 0x30 = 7$

A, ~ ,F : $0x41 \leq \boxed{rd} \leq 0x46$ 경우, $dd = rd - 0x41 + 10$

$\text{else if}(rd \geq 'A' \ \&\& \ rd \leq 'F') \ dd = rd - 'A' + 10$

예, $rd = 'D'$ 경우, $dd = 0x44 - 0x41 + 10 = 13$

a, ~ ,f : $0x61 \leq \boxed{rd} \leq 0x66$ 경우, $dd = rd - 0x61 + 10$

$\text{else if}(rd \geq 'a' \ \&\& \ rd \leq 'f') \ dd = rd - 'a' + 10$

예, $rd = 'c'$ 경우 $dd = 0x63 - 0x61 + 10 = 12$

```
#include <mega128.h>
```

```
flash unsigned char seg_pat[16]= {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d,  
                                0x07,0x7f, 0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79, 0x71};
```

```
void main(void)
```

```
{
```

```
    unsigned char  rd, dd = 0;           //rd 입력 data,
```

```
    DDRB = 0xF0;                        // 포트 B 상위 4비트 출력 설정
```

```
    DDRD = 0xF0;                        // 포트 D 상위 4비트 출력 설정
```

```
    DDRG = 0x0F;                        // 포트 G 하위 4비트 출력 설정
```

```
    PORTG = 0b00001000;                // 맨 우측 세그먼트 on
```

```
    UCSR0A = 0x0;                      // USART 초기화
```

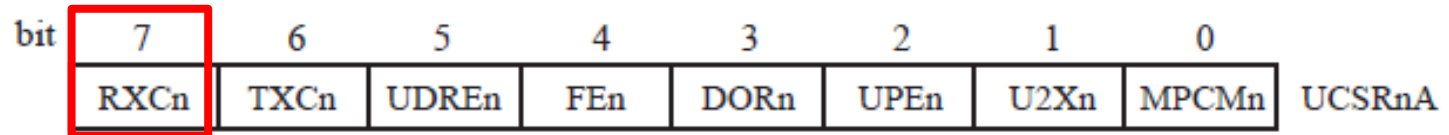
```
    UCSR0B = 0b00010000;               // 수신 enable RXEN0 [4]bit=1
```

```
    UCSR0C = 0b00000110; // 비동기:0, No parity: 00, 정지bit 1이면:0, 데이터 8비트 모드
```

```
    UBRR0H = 0;                        // 16MHz 일때, BAUD = 9600
```

```
    UBRR0L = 103;
```

```
while(1) {
    while((UCSR0A & 0x80) == 0x0)           // RXC0 [7]bit = 1이 될 때까지 대기
    {                                         // 수신 버퍼에 새로운 데이터가 수신되면 RXC0 = 1
```



```
    PORTD = ((seg_pat[dd] & 0x0F) << 4) | (PORTD & 0x0F);
```

```
    PORTB = (seg_pat[dd] & 0x70) | (PORTB & 0x0F);
```

```
}
```

```
rd = UDR0;                                     // 수신되면 while문을 빠져 나와, 값을 읽음
                                              // 값을 읽으면 바로 RXC0 [7]bit = 0 됨
```

```
    if(rd >= '0' && rd <= '9') dd = rd - '0';
```

```
    else if(rd >= 'a' && rd <= 'f') dd= rd - 'a' + 10;
```

```
    else if(rd >= 'A' && rd <= 'F') dd = rd - 'A' + 10;
```

```
}
```

```
}
```

[예제 7-3] USART0 송수신 실험 [polling 방식]

CodeVisionAVR에서 터미널 창을 열고, 컴퓨터에서 key 입력된 문자를 수신하고,
수신한 문자가 알파벳 대문자이면 소문자로

소문자이면 대문자로 변환하여 컴퓨터로 전송하는 프로그램을 작성하라.

즉, 터미널 창에 a를 누르면 A가 표현되고, B를 누르면 b가 표현

: USART0 송수신 실험,

: BAUD = 9600, DataBit = 8 bit, StopBit = 1, Parity = None;

<대문자 ➡ 소문자로 변환>

?

<소문자 ➡ 대문자로 변환>

?

[예제 7-3] USART0 송수신 실험 [polling 방식]

<대문자 → 소문자로 변환>

character = input character - 'A' + 'a'

input character = 'A' 이면, character = 0x41 - 0x41 + 0x61 = 0x61 → a

input character = 'W' 이면, character = 0x57 - 0x41 + 0x61 = 0x77 → w

<소문자 → 대문자로 변환>

character = input character - 'a' + 'A'

input character = 'a' 이면 character = 0x61 - 0x61 + 0x41 = 0x41 → A

input character = 'g' 이면 character = 0x67 - 0x61 + 0x41 = 0x47 → G

```
#include <mega128.h>
```

```
void main(void)
```

```
{
```

```
    unsigned char character ;
```

```
    UCSR0A = 0x0;
```

```
    UCSR0B = 0b00011000;    // 송수신부 enable TXEN0 [3] = 1, RXEN0 [4] = 1
```

```
    UCSR0C = 0b00000110; // 비동기:0, No parity: 00, 정지bit 1이면:0, 데이터 8비트 모드
```

```
    UBRR0H = 0;    // 16MHz 일때, BAUD = 9600
```

```
    UBRR0L = 103;
```

```
    while(1) {
```

```
        while((UCSR0A & 0x80) == 0x0);    // RXC0=1 될 때까지 대기
                                           // 수신 버퍼에 새로운 데이터가 수신되면 RXC0 = 1
        character = UDR0;    // 수신, 값을 읽으면 바로 RXC0 [7]bit = 0 됨
```

```
        if((character >= 'a') && (character <= 'z'))
            character = character - 'a' + 'A';    // 소문자 -> 대문자
        else if((character >= 'A') && (character <= 'Z'))
            character = character - 'A' + 'a';    // 대문자 -> 소문자
```

```
        while((UCSR0A & 0x20) == 0x0);    // 송신 버퍼가 비어지면 UDRE0 = 1 이 될 때까지 대기
                                           // 전송 준비가 되었는지 확인하는 것
```

```
        UDR0 = character ;    // 송신
```

```
    }
```

```
}
```

[예제 7-4] USART0 송수신 실험 [수신 인터럽트 방식]

[예제 6-3]에 대해 송신은 그대로 polling 방식을 이용하고,
수신은 수신완료 인터럽트를 이용하여 프로그램을 작성하라.

USART0 송수신 실험 (수신 인터럽트 방식)

BAUD = 9600, DataBit = 8 bit, StopBit = 1, Parity = None;

```
#include <mega128.h>
```

```
void main(void)
```

```
{
```

```
    UCSR0A = 0x0;
```

```
    UCSR0B = 0b10011000;
```

```
// RXCIE0 [7] = 1 : 수신 완료 인터럽트 enable
```

```
// 송수신 enable TXEN0 [3] = 1, RXEN0 [4] = 1
```

```
    UCSR0C = 0b00000110;
```

```
// 비동기 데이터 8비트 모드
```

```
    UBRR0H = 0;
```

```
// X-TAL = 16MHz 일때, BAUD = 9600
```

```
    UBRR0L = 103;
```

```
    SREG |= 0x80;
```

```
// 전역 인터럽트 인에이블 1 비트 셋
```

```
    while(1);
```

```
// 문자 수신할 때까지 대기
```

```
}
```

```
interrupt [USART0_RXC] void usart0_rx(void)
```

```
// 수신 완료 인터럽트
```

```
{
```

```
    unsigned char ch;
```

```
    ch = UDR0;
```

```
// 수신
```

```
    if(ch >= 'a' && ch <= 'z') ch = ch - 'a' + 'A';
```

```
// 소문자 -> 대문자
```

```
    else if(ch >= 'A' && ch <= 'Z') ch = ch - 'A' + 'a';
```

```
// 대문자 -> 소문자
```

```
    while((UCSR0A & 0x20) == 0x0);
```

```
// UERE0=1 될 때까지 대기
```

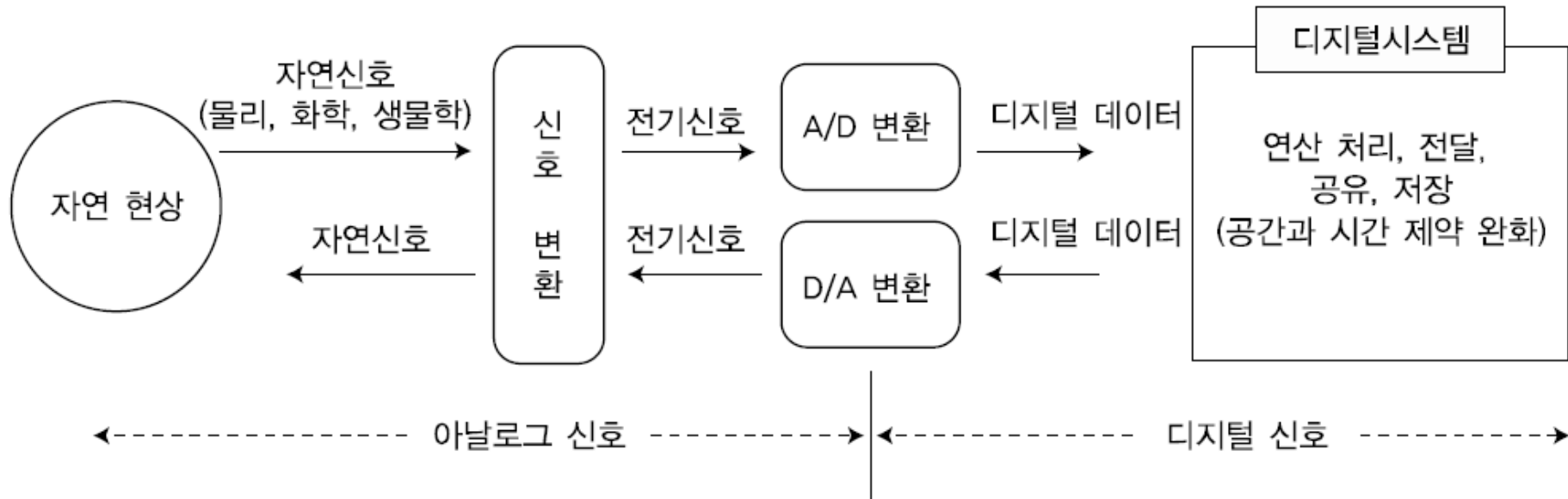
```
    UDR0 = ch;
```

```
// 송신
```

```
}
```


-
1. Micro Processor 원리
 2. Atmel사의 8bit Micro-controller
 3. KUT0128 Evaluation Board 기능과 특징
 4. IO Port 제어
 5. External Interrupt 제어
 6. Timer / Counter 제어
 7. UART 제어
 8. AD Converter 제어
 9. Comparator 제어
 10. EEPROM 제어 (IIC, Parallel method)
 11. SPI 제어

8. ADC (Analog to Digital Converter)



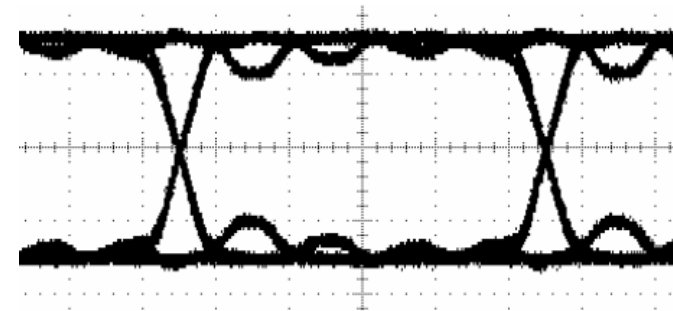
A/D 변환기란 ? 아날로그 신호를 디지털 데이터로 변환

ATmega128의 A/D 변환기 특징

- : **10-bit resolution**(분해 능)
- : **13~260 μ s** 변환시간
- : 8개의 단극성 입력 채널

(**ADC0~ADC7**)을 포트 F로 입력 받아 multi-plexer에 의해 A/D 변환기로 연결

- : 10배와 20배 증폭을 갖는 2개 **차동(differential) 입력** (ADC1,ADC0 그리고 ADC3, ADC2)



8. ADC (Analog to Digital Converter)

AVcc

: 아날로그 전원 핀으로 디지털 전원 핀인 Vcc와 다름

: $V_{cc} \pm 0.3V$ 이내이어야 함

VREF

: A/D 변환기의 기준 전압으로

A/D 변환기의 최대 변환 범위 값

: AVcc, 내부 기준 전압인

2.56V, AREF 핀에 입력되는

전압 중 하나를 선택

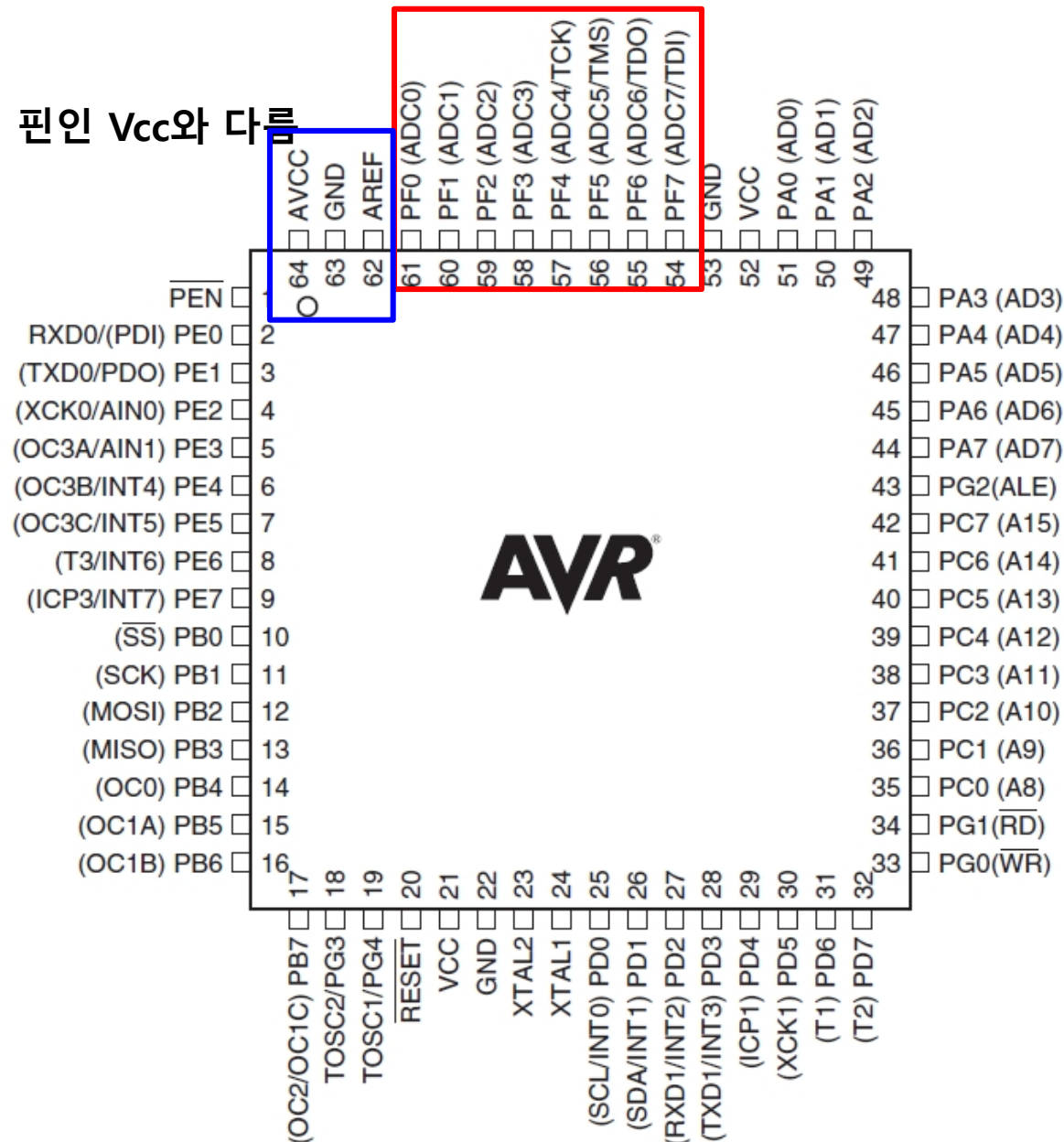
단극성 입력, 차동 입력

: 단극성 입력은 ADC0~ADC7 중

하나만을 A/D 변환

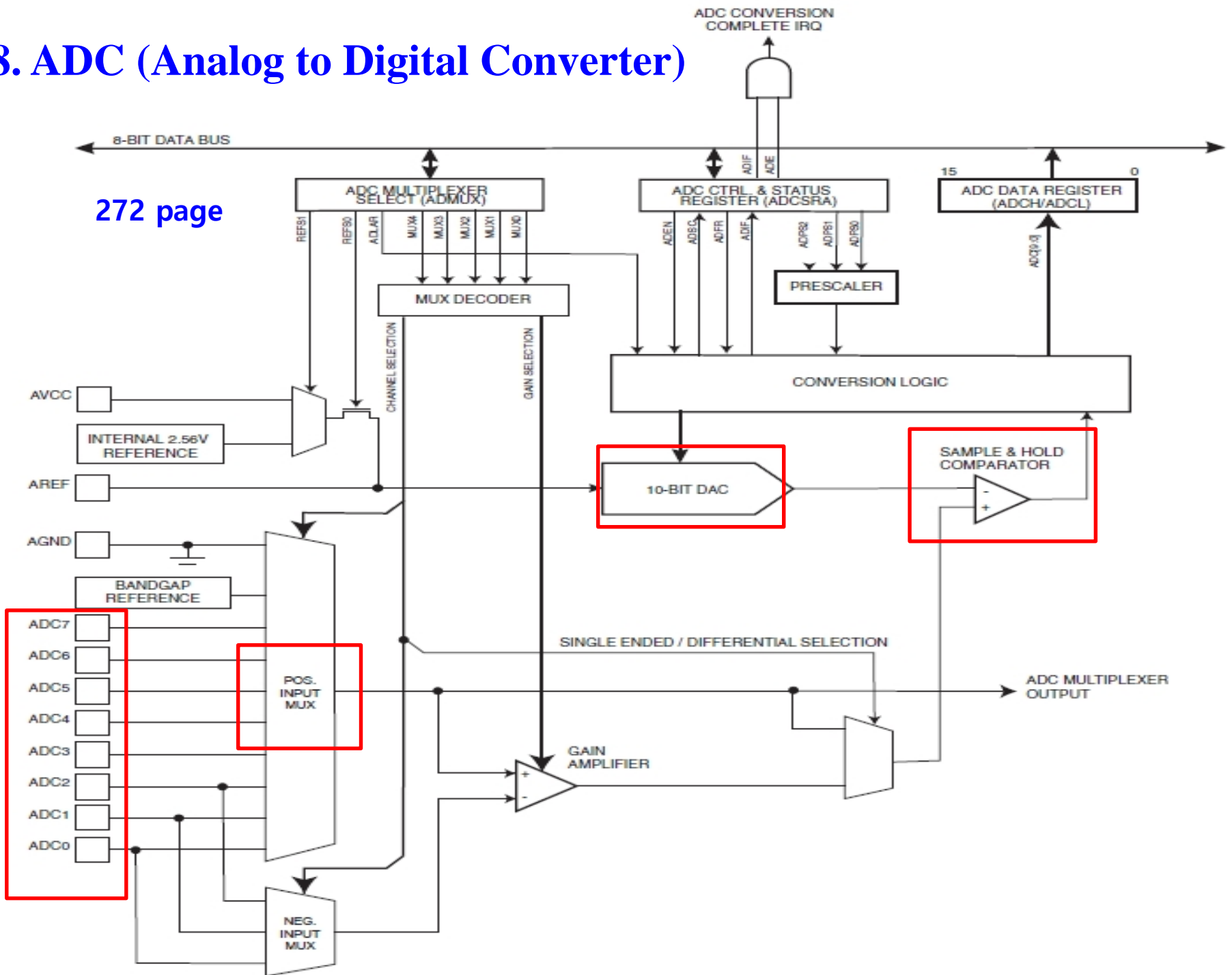
: 차동 입력은 (+)전압과 (-)전압의

차이를 A/D 변환



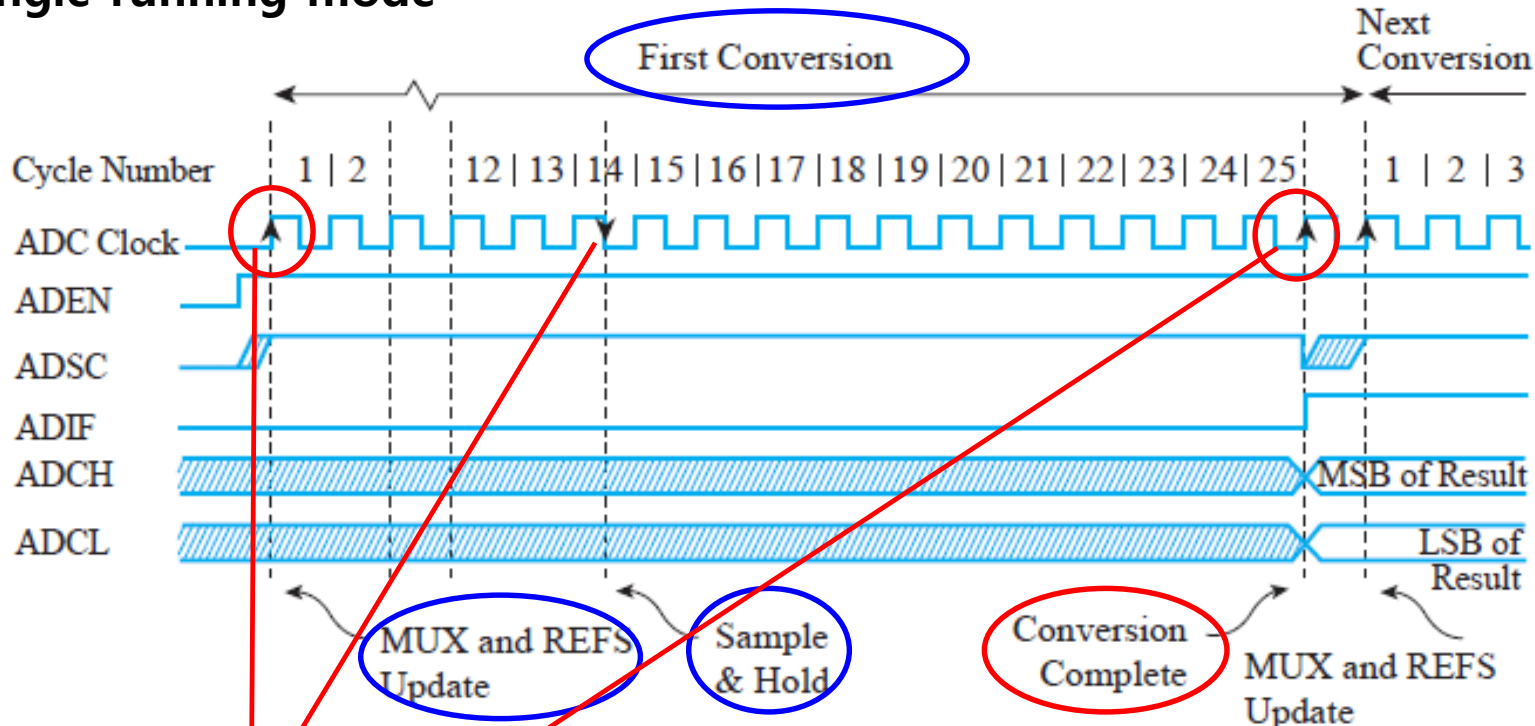
8. ADC (Analog to Digital Converter)

272 page



8. ADC (Analog to Digital Converter)

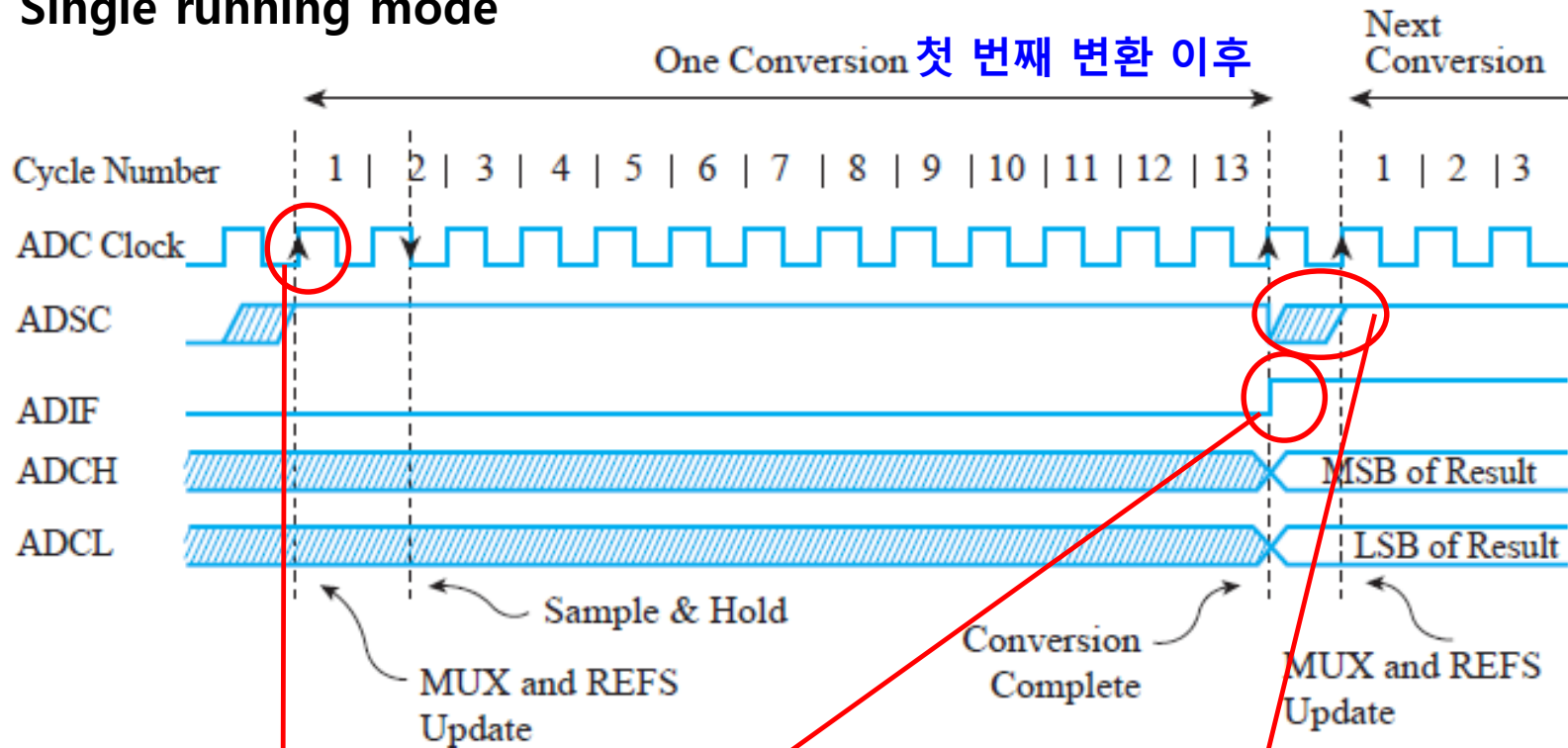
Single running mode



- : 단일변환 모드에서 **ADCSRA 레지스터의 ADSC**(ADC Start Conversion) **bit**를 1로 하면, 1로 한 이후 첫 번째 A/D변환기 clock의 rising edge에서 변환이 시작
- : **ADEN** (ADC Enable) =1로 설정 후, 첫 번째 변환에는 **25개의** A/D변환기 clock 필요
그 이후의 변환에서는 **13개의 clock** 필요
- : 첫 번째 변환은 샘플/홀드는 변환이 시작되고 나서 13.5번째 clock에서 이루어짐

8. ADC (Analog to Digital Converter)

Single running mode



: 이후의 변환에서는 1.5 번째 clock에서 S&H가 이루어짐

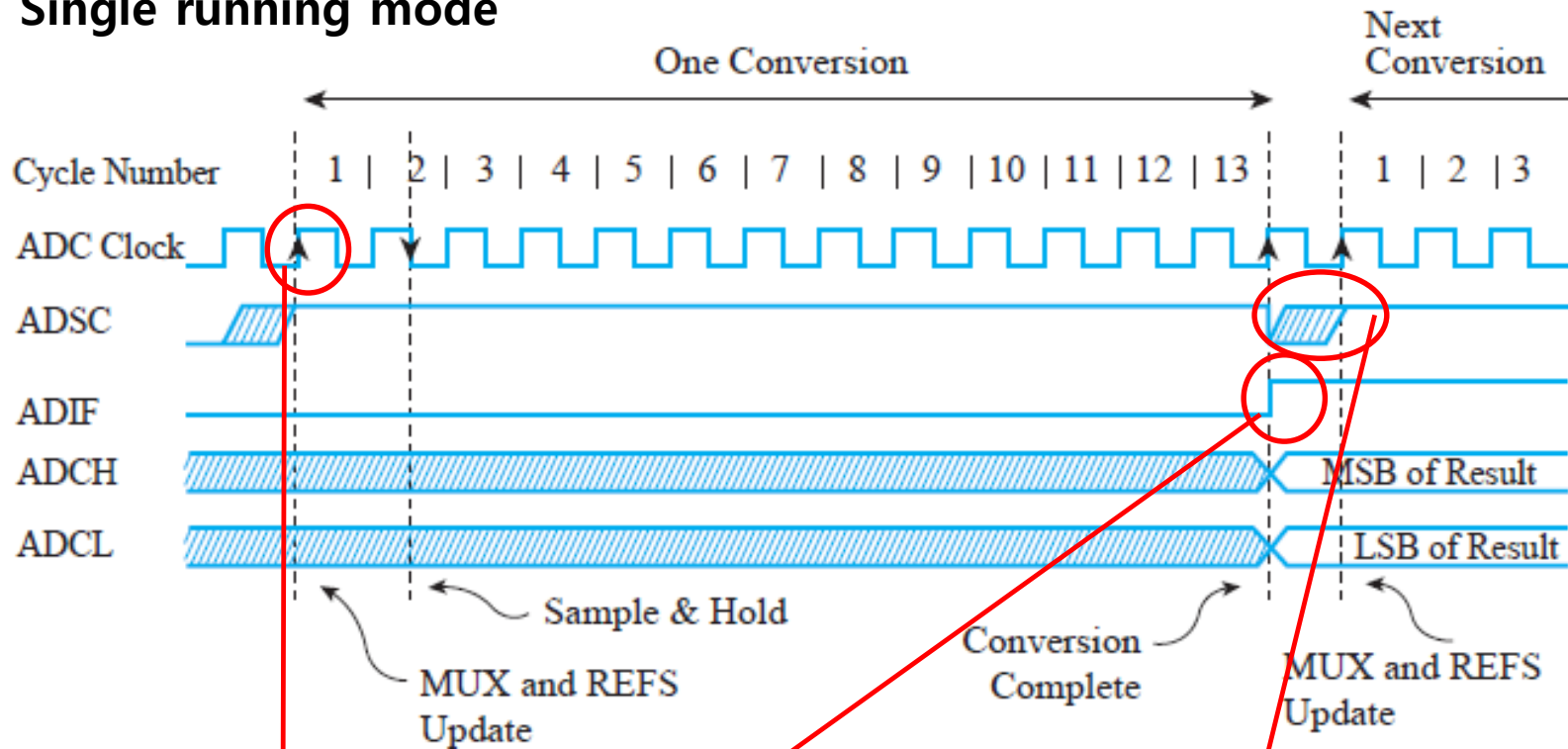
: A/D 변환이 완료되면 그 결과가 A/D 변환기 데이터 레지스터에 쓰여지게 되며,
ADIF(interrupt flag) 비트가 set

: 단일변환 모드에서는 **ADSC(변환시작) 비트가 자동 clear 되어,**

다시 소프트웨어적으로 셋 할 수 있게 됨

8. ADC (Analog to Digital Converter)

Single running mode



: 두 번째 이후의 변환에서는 1.5 번째 clock에서 S&H가 이루어짐

: A/D 변환이 완료되면 그 결과가 A/D 변환기 데이터 레지스터에 쓰여지게 되며,
ADIF(interrupt flag) 비트가 set

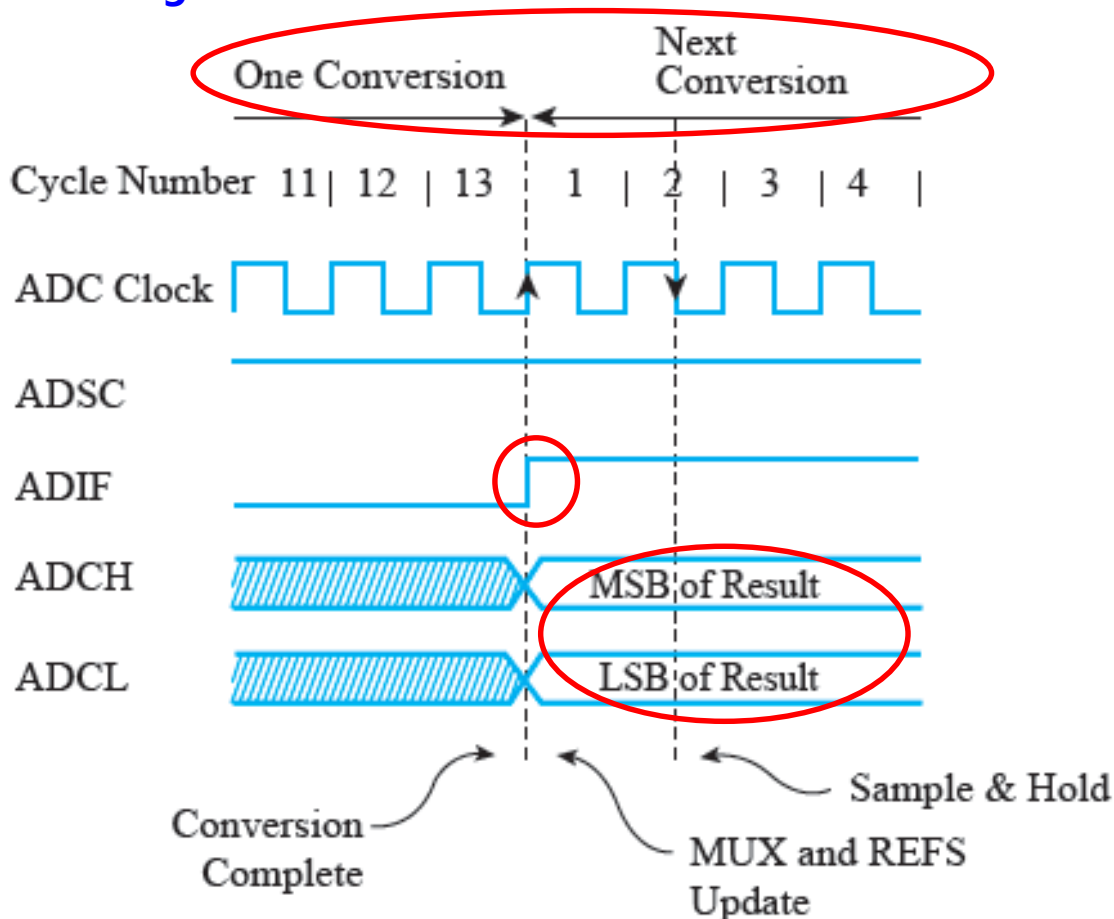
: 단일변환 모드에서는 ADSC(변환시작) 비트가 자동 clear 되어,

다시 소프트웨어적으로 셋 할 수 있게 됨

8. ADC (Analog to Digital Converter)

Free running mode

: 변환이 완료되면 자동적으로 즉시 새로운 A/D변환이 시작,
ADSC 비트는 High상태 유지



8. ADC (Analog to Digital Converter)

ADC 레지스터

레지스터	기능
ADMUX	A/D 변환기 멀티플렉서 선택 레지스터 (ADC Multiplexer Selection Register)
ADCSRA	A/D 변환기 제어 및 상태 레지스터 A (ADC Control and Status Register A)
ADCH, ADCL	A/D 변환기 데이터 레지스터 (ADC Data Register)

8. ADC (Analog to Digital Converter)

ADMUX (ADC Multiplexer Selection Register)

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
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 기준 전압 선택
 ADC 결과값 저장 형식 선택
 ADC 입력 채널 선택

REFS1	REFS0	[7],[6] bit → 기준 전압 선택
0	0	AREF 단자 전압 이용
0	1	AREF 단자 전압 이용하며, AREF 단자와 GND 사이를 콘덴서로 접속
1	0	—
1	1	내부 2.56V를 이용하며, AREF단자와 GND 사이를 콘덴서로 접속

8. ADC (Analog to Digital Converter)

ADMUX (ADC Multiplexer Selection Register)

[5]bit – ADLAR (ADC **Left Adjust** Result)

: A/D 변환 결과값을 ADCH와 ADCL에 저장 형식을 지정하는 비트

: **ADLAR=0** → 하위부터 10비트에 A/D 변환 결과 저장

: **ADLAR=1** → 상위부터 10비트에 A/D 변환 결과 저장

ADLAR = 0 :

bit	15	14	13	12	11	10	9	8	
	-	-	-	-	-	-	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	

ADLAR = 1 :

bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	-	-	-	-	-	-	ADCL
	7	6	5	4	3	2	1	0	

8. ADC (Analog to Digital Converter)

ADMUX (ADC Multiplexer Selection Register)

[4]~[0]bit - MUX4~ MUX0(Analog **Channel and Gain Selection** Bits)

MUX4~MUX0	단극성 입력	(+) 차동 입력	(-) 차동 입력	이득(gain)
00000	ADC0			
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6	Offset 조정용으로 사용됨		
00111	ADC7			
01000		ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010		ADC0	ADC0	200x
01011		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x

8. ADC (Analog to Digital Converter)

ADMUX (ADC Multiplexer Selection Register)

MUX4~MUX0	단극성 입력	(+) 차동 입력	(-) 차동 입력	이득
01110		ADC2	ADC2	200x
01111		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x
11101		ADC5	ADC2	1x
11110	1.23V			
11111	GND			

8. ADC (Analog to Digital Converter)

ADCSRA (ADC Control and Status Register A)

bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
초기값	0	0	0	0	0	0	0	0	

[7] ADEN (ADC Enable) : **A/D 변환기 enable 비트**

: **ADEN=1** → A/D 변환기 동작 가능

: ADEN=0 → A/D 변환기 동작 정지, 소비 전력이 감소시킬 수 있음

[6] ADSC(ADC Start Conversion) : **ADC 변환 시작 비트**

: single running 에서 **ADSC=1** → A/D 변환이 시작

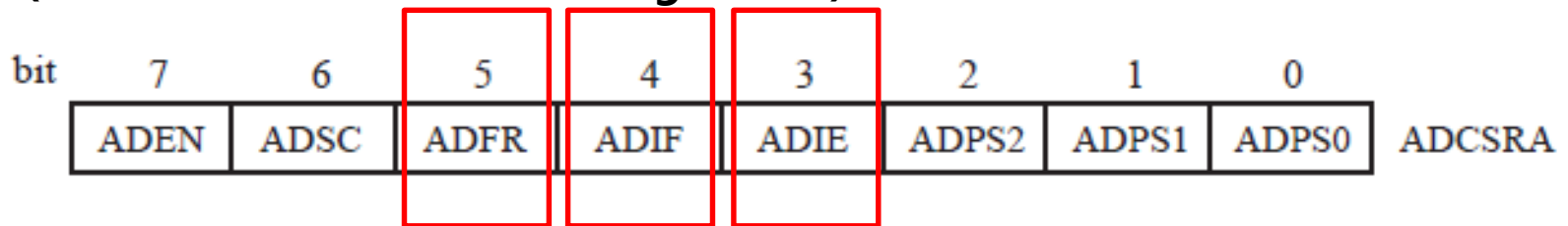
: free running 모드, **ADSC=1** → 첫 번째 A/D 변환 후 자동적으로 A/D 변환이 계속

ADEN=1이 된 후, 첫 번째 A/D 변환에서 clkADC가 **25clock** 필요하고

다음 변환부터는 **13clock** 이 소요됨

8. ADC (Analog to Digital Converter)

ADCSRA (ADC Control and Status Register A)



[5] ADFR (ADC Free Running Select) : A/D 변환을 연속적으로 수행하는 모드

: ADFR=1 → A/D 변환기는 Free Running 모드로 설정

: ADFR=0 → A/D 변환기는 single running 모드로 설정

[4] ADIF(ADC Interrupt Flag) : A/D 변환 완료 인터럽트 flag 비트

: A/D 변환이 완료되면 ADCH, ADCL에 A/D 변환 결과값이 write되고

ADIF=1이 되면서 A/D 변환 완료 인터럽트 발생

: ADIE=1이고 SREG의 I=1이면,

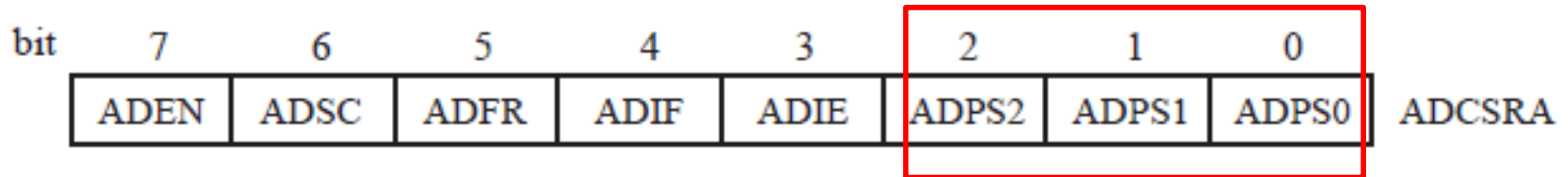
A/D 변환 완료 인터럽트가 처리되고 이 때 ADIF=0으로 clear됨

: ADIF를 강제로 clear 시키려면 ADIF=1로 write 시켜야 함

[3] ADIE(ADC Interrupt Enable) : A/D 변환 완료 인터럽트 enable 비트

8. ADC (Analog to Digital Converter)

ADCSRA (ADC Control and Status Register A)



[2]~[0] ADPS2~ADPS0 (ADC Pre-scaler Select Bits)

: A/D 변환기에서 사용되는 **시스템 clock의 분주비** 선택

ADPS1	ADPS1	ADPS0	분주 비
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

8. ADC (Analog to Digital Converter)

ADCH, ADCL (ADC Data Register)

: 10비트의 A/D 변환 결과값을 저장

: A/D 변환 결과값을 리드할 때 ADCH를 먼저 read해야 함

ADLAR = 0 :

bit	15	14	13	12	11	10	9	8	
	-	-	-	-	-	-	ADC9	ADC8	ADCH
	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	
초기값	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	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	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	
초기값	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

8. ADC (Analog to Digital Converter)

A/D 변환 결과는 ADCH와 ADCL에 저장

단극성 입력의 A/D 변환 결과

: 변환식

$$ADC = \frac{V_{IN} \times 1024}{V_{REF}}$$

VIN : 단극성 입력 채널의 전압

VREF : 기준 전압

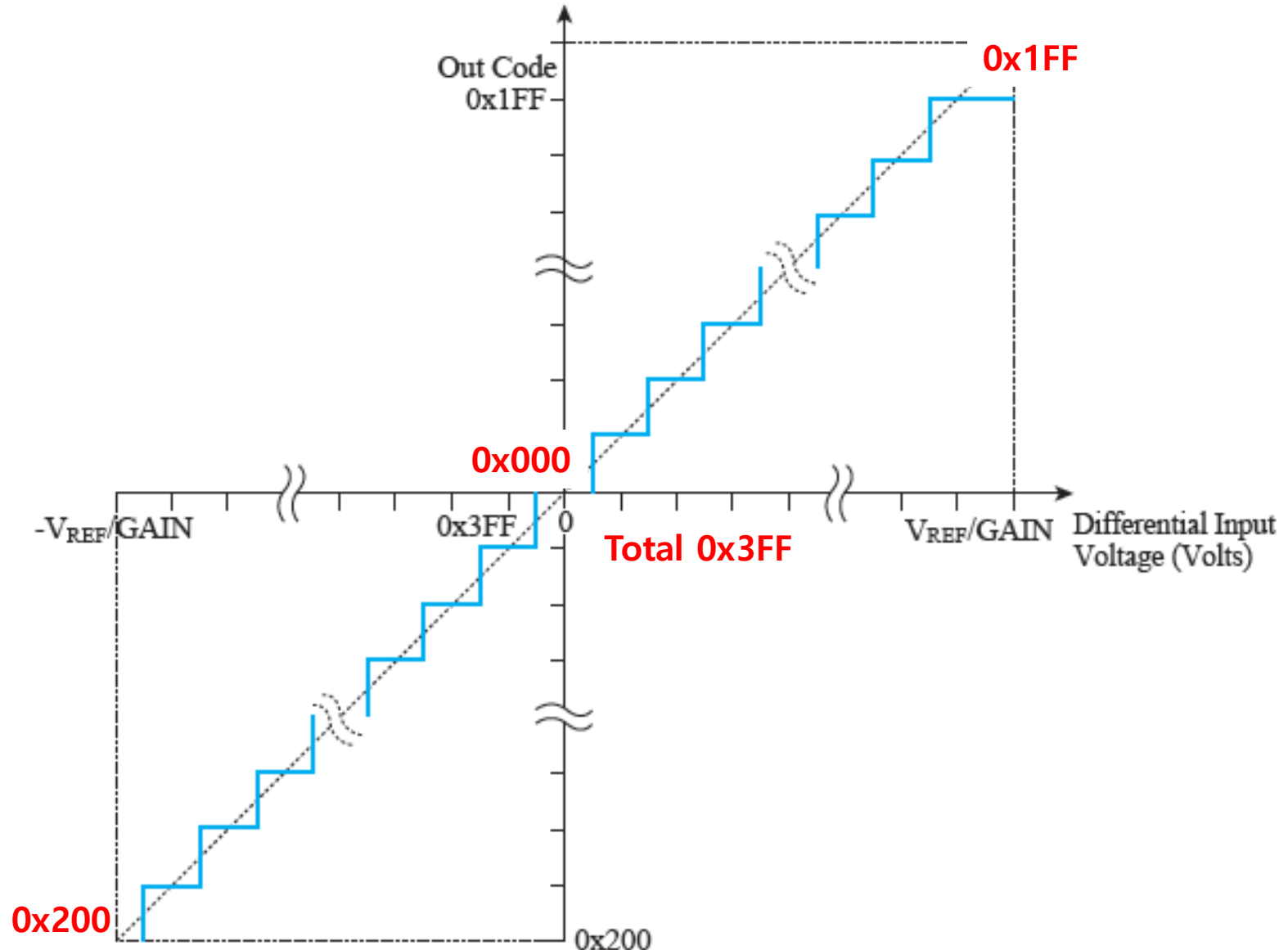
: 10비트 양의 정수인 0~1023 (000H~3FFH) 범위내의 값으로 변환

예, 0(000H) = 아날로그 입력 전압 접지(0V)

1023(0x3FF) = (VREF - 1 LSB) 전압 값

8. ADC (Analog to Digital Converter)

Differential (차동) 입력의 A/D 변환 결과 값의 범위



Summary

ADC 초기화

- ① AREF 전압 선택, A/D 변환 결과 저장 형식 선택,
A/D 변환 입력채널 설정 (단극성 입력, 차동입력)
➔ 사용 레지스터 : **ADMUX**

- ② 사용 모드 설정
A/D 변환 enable, 분주비 설정,
변환모드 설정(단일변환/ 프리러닝), A/D변환 인터럽트 enable,
A/D변환기 입력 클럭 분주비 설정
➔ 사용 레지스터 : **ADCSRA**

- ③ 전역 인터럽트 enable 레지스터 : **SREG(최상위 비트)**

Summary

<단일변환 모드 A/D 변환 동작>

- ADCSRA 레지스터의 ADSC(ADC Start Conversion) 를 1로 하여 A/D 변환 시작
- ADIF = 1(또는 ADSC=0)이 되어 A/D변환이 완료될 때까지 대기
- A/D 변환 결과 읽기(ADCW 또는 ADCH:ADCL)

<프리러닝 모드 A/D 변환 동작>

- 사용모드 설정하면서 ADCSRA 레지스터의 ADSC 비트를 1로 하여 A/D 변환 시작
- A/D 변환 결과 (ADCW 또는 ADCH : ADCL)를 읽고 싶을 때 읽기 가능

- Project 생성시,
C-Compiler에서
clock 8MHz를 16MHz로



Configure Project test4_6.prj

Files C Compiler Before Build After Build

Code Generation Libraries Messages Globally #define Paths

Active Build Configuration: Debug

Chip: ATmega128A

Clock: 16 MHz

Memory Model: Small

Optimize for: Size

Optimization Level: Maximal

Program Type: Application

(s)printf Features: int_width

(s)scanf Features: int_width

RAM

Data Stack Size: 1024 bytes

Heap Size: 0 bytes

Internal RAM Size: 4096 bytes

External RAM Size: 0 bytes

☐ External RAM Wait State

Code Generation

Bit Variables Size: 16

☒ Promote char to int ☒ char is unsigned

☒ 8 bit enums Enh. Param: Mode 2

☒ Smart Register Allocation

☒ Automatic Global Register Allocation

☐ Store Global Constants in FLASH Memory

☐ Use an External Startup Initialization File

☒ Clear Global Variables at Program Startup

☐ Stack End Markers

File Output Formats: COF ROM HEXEEP

Preprocessor

☐ Create Preprocessor Output Files

☒ Include I/O Registers Bits Definitions

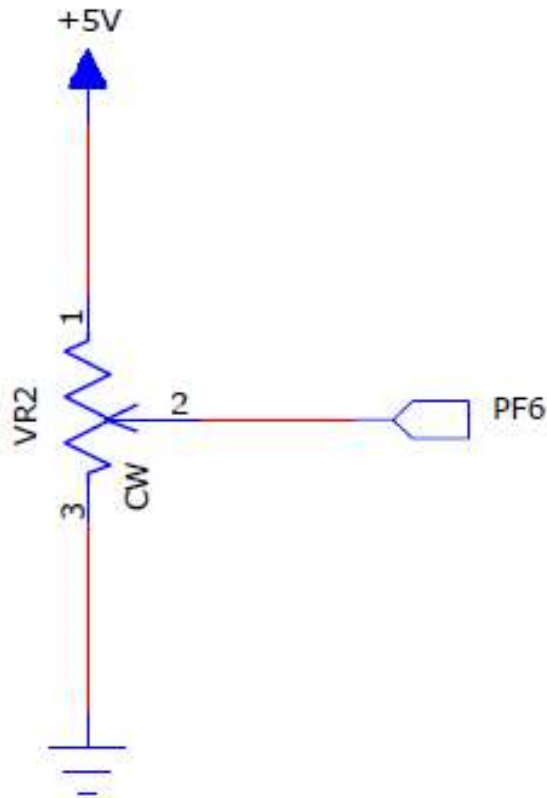
OK Cancel Help

[예제 8-1] ADC6에 입력되는 전압 7-Segment에 표시

ADC6 (PF6)에는 가변저항(VR2) 10K Ω 이 연결되어 있으며

이 단자를 통해 입력되는 전압(단극성 입력)을

3개의 7-Segment에 소수 두 자리까지 전압 값으로 표시하는 프로그램을 작성



소수 자리값 추출

① 10^2 을 곱한 후 정수화 : $(\text{int})(2.3456 \times 100) \Rightarrow 234$

② 234를 100으로 나눈 몫과 나머지를 구한다.

=> 몫 : 2(정수 1자리), 나머지 34

③ 나머지 34를 10으로 나눈 몫과 나머지를 구한다.

=> 몫 : 3(소수 첫번째 자리), 나머지 4(소수 둘째 자리)

※ 정수화 할 때 0.5를 더하면서 수행하면

$(\text{int})(2.3456 \times 100 + 0.5) = 235$

가 되어 소스 3번째 자리에서 반올림한 값이 얻어진다.

```
#include <mega128.h>
```

```
#include <delay.h>
```

```
typedef unsigned char u_char;
```

```
flash u_char seg_pat[10]= {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f};
```

```
void AD_disp(int);
```

```
// A/D 변환값 표시
```

```
void main(void)
```

```
{
```

```
int ad_val;
```

```
DDRB = 0xF0;
```

```
DDRD = 0xF0;
```

```
DDRG = 0x0F;
```

```
ADMUX = 0x06;
```

```
ADCSRA = 0x87;
```

```
delay_ms(5);
```

```
while(1){
```

```
ADCSRA = 0xC7;
```

```
while((ADCSRA & 0x10) == 0);
```

```
ad_val = (int)ADCL + ((int)ADCH << 8);
```

```
AD_disp(ad_val);
```

```
}
```

7	6	5	4	3	2	1	0	
REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	
7	6	5	4	3	2	1	0	
ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA

```
// ADC6 단극성 입력 선택
```

```
// ADEN=1, 16MHz 256분주 -> 125kHz
```

```
// ADEN=1, ADSC = 1 변환 시작
```

```
// ADIF=1이 될 때까지
```

```
// A/D 변환값 읽기
```

```
// A/D 변환값 표시
```



```

void AD_disp(int val) {
    float fval;
    int ival, buf;
    u_char N100, N10, N1;

    fval = (float)val * 5.0 / 1024.0;           // 전압 값으로 변환
    ival = (int)(fval * 100.0 + 0.5);          // 반올림 후 정수화, (소수 둘째자리까지)

    N100 = ival / 100;                         // 정수부 추출
    buf = ival % 100;

    N10 = buf / 10;                           // 소수 첫째 자리 추출
    N1 = buf % 10;                            // 소수 둘째 자리 추출

    PORTG = 0b00001000;                       // PG3=1, 소수 둘째 자리
    PORTD = ((seg_pat[N1] & 0x0F) << 4) | (PORTD & 0x0F);
    PORTB = (seg_pat[N1] & 0x70 ) | (PORTB & 0x0F);
    delay_ms(1);

    PORTG = 0b00000100;                       // PG2=1, 소수 첫째 자리
    PORTD = ((seg_pat[N10] & 0x0F) << 4) | (PORTD & 0x0F);
    PORTB = (seg_pat[N10] & 0x70 ) | (PORTB & 0x0F);
    delay_ms(1);

    PORTG = 0b00000010;                       // PG1=1, 정수부
    PORTD = ((seg_pat[N100] & 0x0F) << 4) | (PORTD & 0x0F);
    PORTB = (seg_pat[N100] & 0x70 ) | (PORTB & 0x0F);
    PORTB = PORTB | 0x80;                     // DP on(소수점)
    delay_ms(1);
}

```

[예제 8-2] Free running, ADC6 입력 전압 7-Segment 표시

ADC6 (PF6)에는 가변저항(VR2) 10K Ω 이 연결되어 있으며
이 단자를 통해 입력되는 전압(단극성 입력)을
3개의 7-Segment에 소수 두 자리까지 전압 값으로 표시하는 프로그램을 작성
Free running mode

```
#include <mega128.h>
#include <delay.h>
```

```
typedef unsigned char u_char;
```

```
const u_char seg_pat[10]= {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f};
```

```
void AD_disp(int);          // A/D 변환값 표시
```

```
void main(void)
```

```
{
```

```
    int  ad_val;
```

```
    DDRB = 0xF0;
```

```
    DDRD = 0xF0;
```

```
    DDRG = 0x0F;
```

```
    ADMUX = 0x06;
```

```
    ADCSRA = 0xE7;
```

```
    delay_ms(5);
```

```
    while(1)
```

```
    {
```

```
        ad_val = (int)ADCL + ((int)ADCH << 8);
```

```
        AD_disp(ad_val);
```

```
        delay_ms(2);
```

```
    }
```

```
}
```

7	6	5	4	3	2	1	0	
REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	
7	6	5	4	3	2	1	0	
ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA

```
    // 포트 B 상위 4비트 출력 설정
```

```
    // 포트 D 상위 4비트 출력 설정
```

```
    // 포트 G 하위 4비트 출력 설정
```

```
    // ADC6 단극성 입력 선택
```

```
    // ADEN=1, ADSC=1, ADFR=1, 16MHz, 256분주 -> 125kHz
```

```
    // A/D값 읽기
```

```
    // A/D값 표시
```

```

void AD_disp(int val) {
    float fval;
    int ival, buf;
    u_char N100, N10, N1;

    fval = (float)val * 5.0 / 1023.0;    // 전압값으로 변환
    ival = (int)(fval * 100.0 + 0.5);    // 반올림 후 정수화

    N100 = ival / 100;                  // 정수부 추출
    buf = ival % 100;

    N10 = buf / 10;                     // 소수 첫째 자리 추출
    N1 = buf % 10;                      // 소수 둘째 자리 추출

    PORTG = 0b00001000;                // PG3=1, 소수 둘째 자리
    PORTD = ((seg_pat[N1] & 0x0F) << 4) | (PORTD & 0x0F);
    PORTB = (seg_pat[N1] & 0x70) | (PORTB & 0x0F);
    delay_ms(1);

    PORTG = 0b00000100;                // PG2=1, 소수 첫째 자리
    PORTD = ((seg_pat[N10] & 0x0F) << 4) | (PORTD & 0x0F);
    PORTB = (seg_pat[N10] & 0x70) | (PORTB & 0x0F);
    delay_ms(1);

    PORTG = 0b00000010;                // PG1=1, 정수부
    PORTD = ((seg_pat[N100] & 0x0F) << 4) | (PORTD & 0x0F);
    PORTB = (seg_pat[N100] & 0x70) | (PORTB & 0x0F);
    PORTB = PORTB | 0x80;              // DP on(소수점)
    delay_ms(1);
}

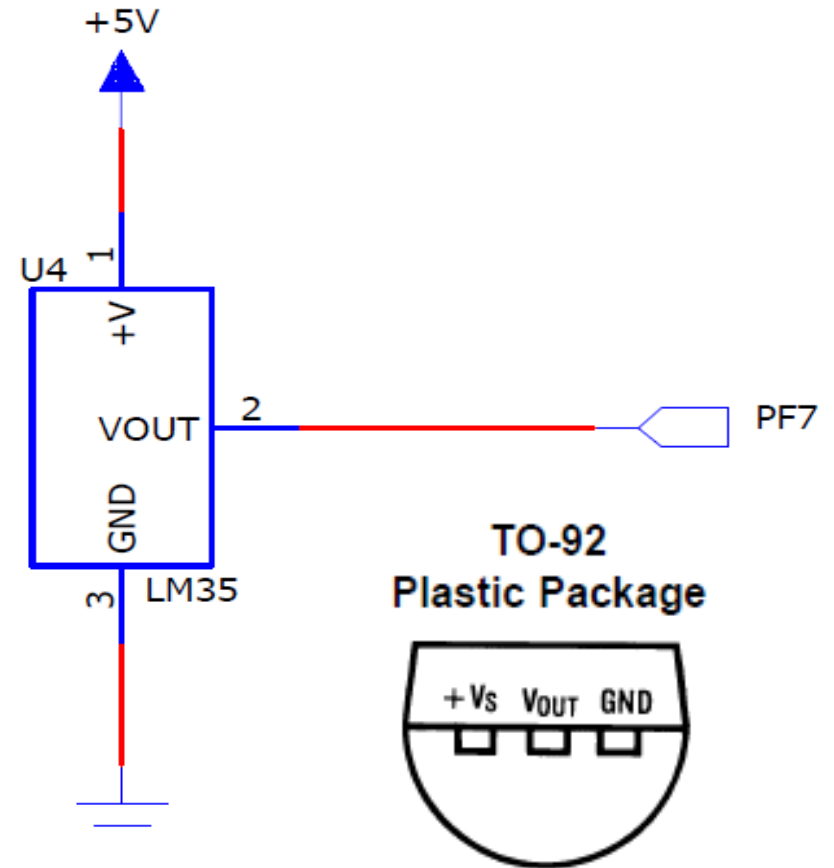
```

National semiconductor

LM35DZ : ADC7

: $0\text{mV} + 10.0\text{mV}/^{\circ}\text{C}$

: 예 $25^{\circ}\text{C} \Rightarrow 250\text{mV}$



Order Number LM35CZ,
LM35CAZ or LM35DZ

• Stabilization of SW - MISRA (Motor Industry Software Reliability Association)

: MISRA

“Guidelines for the use of the C language in vehicle based software” – 1998, 영국산업신뢰성협회

S.No	MISRA Rule#	Description
1	MISRA rule 5	Only those characters and escape sequences which are defined in the ISO C standard shall be used.
2	MISRA rule 7	Trigraphs shall not be used.
3	MISRA rule 8	Multibyte characters and wide string literals shall not be used.
4	MISRA rule 9	Comments shall not be nested.
5	MISRA rule 10	Sections of code should not be 'commented out'
6	MISRA rule 13	The basic types of char, int, long, float and double should not be used, but specific length equivalents should be typedefed for the specific compiler (and microprocessor).
7	MISRA rule 14	The type char shall always be declared as unsigned char or signed char.
8	MISRA rule 16	The underlying bit representations of floating point numbers shall not be used in any way by the programmer.
9	MISRA rule 17	typedef names shall not be reused.
10	MISRA rule 19	Octal constants (other than zero) shall not be used.
11	MISRA rule 20	All object and function identifiers shall be declared before use.
12	MISRA rule 21	Identifiers in an inner scope shall not use the same name as an identifier in an outer scope and therefore hide that identifier.
13	MISRA rule 22	Declarations of objects should be at function scope unless a wider scope is necessary.
14	MISRA rule 23	All declarations at file scope should be static where possible.
15	MISRA rule 25	An identifier with external linkage shall have exactly one external definition.
16	MISRA rule 26	If objects or functions are declared more than once they shall have compatible declarations
17	MISRA rule 27	External objects should not be declared in more than one file.
18	MISRA rule 29	The use of a tag shall agree with ist declaration.
19	MISRA rule 30	All automatic variables shall have been assigned a value before being used.
20	MISRA rule 31	Braces shall be used to indicate and match the structure in the non-zero initialisation of arrays and structures.
21	MISRA rule 32	In an enumerator list, the "=" construct shall not be used to explicitly initialise members other than the first, unless all items are explicitly initialised.
22	MISRA rule 33	The right hand operand of a && or operator shall not contain side effects.
23	MISRA rule 34	The operands of a logical && or shall be primary expressions
24	MISRA rule 35	Assignment operators shall not be used in expressions which return Boolean values.
25	MISRA rule 37	Bitwise operations shall not be performed on signed integer types.
26	MISRA rule 38	The right hand operand of a shift operator shall lie between zero and one less than the width in bits of the lefthand operand (inclusive).
27	MISRA rule 39	The unary minus operator shall not be applied to an unsigned expression.
28	MISRA rule 40	The sizeof operator should not be used on expressions that contain side effects.
29	MISRA rule 43	Implicit conversions which may result in a loss of information shall not be used.
30	MISRA rule 45	Type casting from any type to or from pointers shall not be used.
31	MISRA rule 46	The value of an expression shall be the same under any order of evaluation that the standard permits.
32	MISRA rule 48	Mixed precision arithmetic should use explicit casting to generate the desired result.
33	MISRA rule 50	Floating point variables shall not be tested for exact equality or inequality.
34	MISRA rule 52	There shall be no unreachable code.
35	MISRA rule 56	The goto statement shall not be used.
36	MISRA rule 59	The statements forming the body of an if, else if, else, while, do... While or for statement shall always be enclosed in braces.
37	MISRA rule 61	Every non-empty case clause in a switch statement shall be terminated with a break statement.
38	MISRA rule 62	All switch statements should contain a final default clause.
39	MISRA rule 64	Every switch statement shall have at least one case.
40	MISRA rule 65	Floating point variables shall not be used as loop counters.
41	MISRA rule 68	Functions shall always be declared at file scope.
42	MISRA rule 69	Functions with variable numbers of arguments shall not be used.
43	MISRA rule 70	Functions shall not call themselves, either directly or indirectly.
44	MISRA rule 71	Functions shall always have prototype declarations and the prototype shall be visible at both the function definition and call.
45	MISRA rule 75	Every function shall have an explicit return type.
46	MISRA rule 76	Functions with no parameters shall be declared with parameter type void.
47	MISRA rule 78	The number of parameters passed to a function shall match the function prototype.
48	MISRA rule 79	The values returned by void functions shall not be used.
49	MISRA rule 80	Void expressions shall not be passed as function parameters.
50	MISRA rule 81	const qualification should be used on function parameters which are passed by reference, where it is intended that the function will not modify the parameter.
51	MISRA rule 83	For functions with non-void return type
52		i, there shall be one return statement for every exit branch (including the end of program)
53		ii, each return shall have an expression
54		iii, the return expression shall match the declared return type.
55	MISRA rule 84	For functions with void return type, return statements shall not have an expression.
56	MISRA rule 85	Functions called with no parameters should have empty parentheses.
57	MISRA rule 87	#include statements in a file shall only be preceded by other preprocessor directives or comments.
58	MISRA rule 88	Non-standard characters shall not occur in header file names in #include directives.
59	MISRA rule 89	The #include directive shall be followed by either a <filename> or "filename" sequence.
60	MISRA rule 91	Macros shall not be #defined and #undefed within a block.
61	MISRA rule 94	A function-like macro shall not be 'called' without all of its arguments.
62	MISRA rule 95	Arguments to a function-like macro shall not contain tokens that look like preprocessing directives.
63	MISRA rule 96	In the definition of a function-like macro the whole definition, and each instance of a parameter, shall be enclosed in parentheses.
64	MISRA rule 98	There shall be at most one occurrence of the # or ## pre-processor operators in a single macro definition.
65	MISRA rule 99	All uses of the #pragma directive shall be documented and explained.
66	MISRA rule 100	The defined pre-processor operator shall only be used in one of the two standard forms.
67	MISRA rule 102	No more than 2 levels of pointer indirection should be used.
68	MISRA rule 103	Relational operators shall not be applied to pointer types except where both operands are of the same type and point to the same array, structure or union.
69	MISRA rule 104	Non-constant pointers to functions shall not be used.
70	MISRA rule 105	All the functions pointed to by a single pointer to function shall be identical in the number and type of parameters and the return type.
71	MISRA rule 106	The address of an object with automatic storage shall not be assigned to and object which may persist after the object has ceased to exist.
72	MISRA rule 107	The null pointer shall not be de-referenced.

Abbreviation	description	Criteria
N1	Total Number of operator	
N2	Total number of operands	
STMT	Number of statement	< =50
VG	Cyclomatic Number (VG)	< =20
AVGS	Average size of statement	< =9
GOTO	Number of GOTO statement	0
RETU	Number of Return Statement	1
NBCALLING	Number of caller	< =10
PATH	Number of path	< =1000
PARA	Number of function parameter	< =5
ap_cg_cycle	call Graph recursions	0
	dead code	
DRCT_CALLS	Number of calls direct	
LEVL	Number of Level	



1. while(1)문 시작 전에 7segments에 자신의 학번(8자리)을 쓰고 눈으로 확인할 수 있도록 디스플레이 하기. 이때 학번 8자리 반드시 표시하고(표시방법은 자유롭게 프로그램) 그리고 while(1)문에서 계속 학번 뒤 4자리를 계속 디스플레이 하기
2. while(1)이전에서 External INT 4, 5, 6, 7를 활성화(Enable 시킬 것)시키고, nesting은 허용하지 않도록 함. While(1)에서는 아래의 이벤트를 수행한다
 - (1) External INT 4(rising edge) 이벤트가 발생하면 8bit timer2 overflow mode로 학번 뒤 (2자리 x msec) 주기를 만들고, (예를 들어 20211234라면 34msec) 8bit timer2 overflow Interrupt를 발생시키고, 이를 이용하여 4500msec 될 때마다 ADC를 통해 온도(free running mode) 측정하여 7-segment에 디스플레이 할 것. 이때 학번 표시는 멈추고, 온도를 디스플레이 한다. 온도는 소수점 1의 자리까지, 소수점 표시 => 예 31.2 : 7-segments 4개 모두 사용) 그리고 ADC를 통해 온도검출후 온도표시는 10회 동안만 동작시킨다. 그이후 다시 학번을 정상적으로 표시한다.
 - (2) External INT 5(falling edge) 발생하면 발생하면 16bit timer1 compare match(CTC)로 학번 뒤 (2자리 x 100msec) 주기를 만들고, (예를 들어 20211234라면 3400msec) timer1 CTC interrupt를 발생시키고 Interrupt 발생하면 ADC를 통해 전압(single mode)을 측정하고 측정된 값을 UART를 동작시켜 PC화면에 전압 값을 디스플레이 시킬 것
 - (3) External INT 6(rising edge) 발생하면 timer3의 90bit PWM mode6를 동작시키고, OC3A(PE3) pin으로 DUTY비-학번_뒤자리_2개 (예를 들어 20211234라면 34% duty)를 PWM를 출력시킨다
 - (4) External INT 7(falling edge) 발생하면 문제3의 timer3의 9bit PWM mode 출력을 중지시키고 timer3의 9bit Phase correct PWM mode2를 동작시키고, OC3A(PE3) pin으로 DUTY비-학번_뒤자리_2개 (예를 들어 20211234라면 34% duty)를 PWM를 출력시킨다
3. interrupt service routine은 짧게 프로그래밍할 것 / 변수는 typedef 문을 사용하여 선언할 것 / 프로그램 맨 위에는 주석 작성

Project 2-2

: UART

: ADC

: Project2 debugging 완성하기

Project 2에서

[Project 3]

1. while(1)문 시작 전에 7segments에 자신의 학번(8자리)을 쓰고 눈으로 확인할 수 있도록 디스플레이 하기. 이때 학번 8자리 반드시 표시하고(표시방법은 자유롭게 프로그램)
2. while(1)에서 UART 통신을 사용하여 External INT 4, 5, 6, 7를 활성화(Enable 시킬 것)시키고, nesting은 허용하지 않도록 함.

UART 메인 메뉴 :

The UART program mode

External Interrupt4 : No 4

External Interrupt4 : No 5

External Interrupt4 : No 6

External Interrupt4 : No 7

Please press the number :

Project 2와 동일

1. BH2220FVM 8bit 3ch D/A converter

DAC BH2220FVM (MSB FIRST)

- D7 ~ D0 : 0000 0000 => GND
- : 0000 0001 => (VCC-GND)/256 X 1
- : 1111 1111 => (VCC-GND)/256 X 255

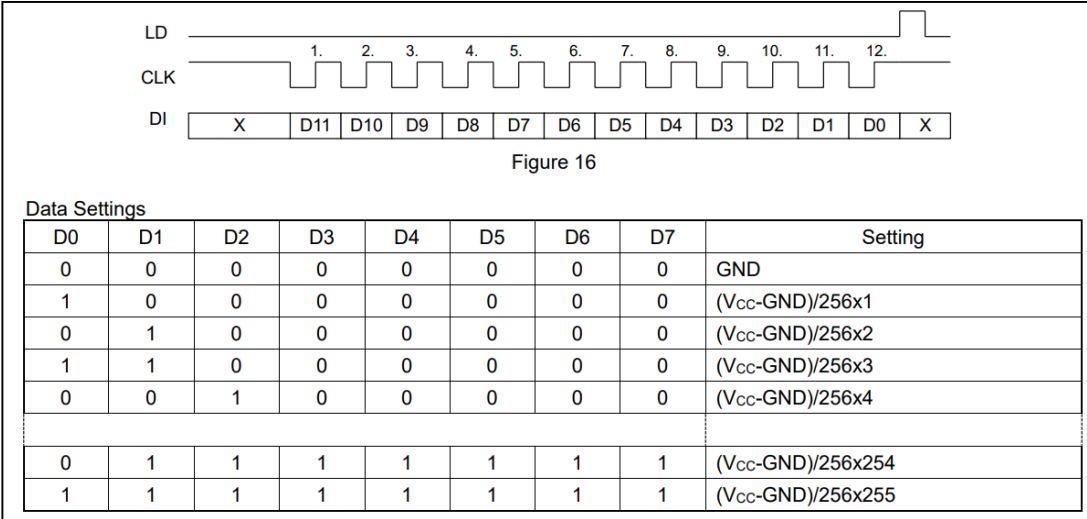
- D9,D8 : 00=>OUTPUT DA01(1pin)
- : 01=>OUTPUT DA02(2pin)
- : 10=>OUTPUT DA03(3pin)

Maximum Data Transfer Frequency: 10MHz(Max)

3-line serial interface

Channel setting (BH2220FVM)

D8	D9	D10	D11	Setting
0	0	X	X	AO1
1	0	X	X	AO2
0	1	X	X	AO3
1	1	X	X	Not used



감사합니다