C언어 – HW9

임베디드스쿨2기

Lv1과정

2021. 05. 31

차현호

UART

아래의 코드는 Atmega328p mcu의 UART를 사용하기 위해 Init 하는 함수이다.

Init함수에서 가장 먼저해주는 것은 UBRR0H와 UBRR0L레지스터 셋팅이다.
그럼 각각 어떤 레지스터인지 확인해 보자

```
void USART_Init()
{
    // ubrr = ((F_CPU) / (16 * BAUDRATE) - 1)
    UBRR0H = ubrr >> 8;
    UBRR0L = ubrr;

    UCSR0B = (1 << RXEN0) | (1<<TXEN0);
    UCSR0C = (1 << USBBS0) | (3 << UCSZ00);

}
```

19.10.5 UBRRnL and UBRRnH – USART Baud Rate Registers

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	

- **Bit 15:12 – Reserved Bits**

These bits are reserved for future use. For compatibility with future devices, these bit must be written to zero when UBRRnH is written.

- **Bit 11:0 – UBRR11:0: USART Baud Rate Register**

This is a 12-bit register which contains the USART baud rate. The UBRRnH contains the four most significant bits, and the UBRRnL contains the eight least significant bits of the USART baud rate. Ongoing transmissions by the transmitter and receiver will be corrupted if the baud rate is changed. Writing UBRRnL will trigger an immediate update of the baud rate prescaler.

ATmega328p의 데이터시트를 보면 UBRR레지스터는 USART의 Baud Rate를 셋팅하는 레지스터임을 확인 할 수 있다.

따라선 16bit UBRR 값을 구한 후 상위 8bit는 UBRR0H에 하위 8bit는 UBRR0L에 넣어 주면 된다.

UART

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRRn Value
Asynchronous normal mode (U2Xn = 0)	$BAUD = \frac{f_{OSC}}{16(UBRRn + 1)}$	$UBRRn = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous double speed mode (U2Xn = 1)	$BAUD = \frac{f_{OSC}}{8(UBRRn + 1)}$	$UBRRn = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous master mode	$BAUD = \frac{f_{OSC}}{8(UBRRn + 1)}$	$UBRRn = \frac{f_{OSC}}{2BAUD} - 1$

Note: 1. The baud rate is defined to be the transfer rate in bit per second (bps)

이전슬라이드에서 언급한 UBRR을 구하는 식은 위의 그림에서도 나와있듯이
UBRRn = Fosc / (16 x Baud Rate) - 1로 구할 수있다.

UBRR 레지스터의 세팅이 끝난 후 UCSR0B, UCSR0C 레지스터를 살펴보자.

19.10.3 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	

19.10.4 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	

UCSR 레지스터들은 USART의 Control 셋팅과 상태 레지스터이다.

이전에 작성된 코드를 보면 UCSR0B 레지스터를 통해 UART의 RX, TX를 Enable 하고 UCSR0C레지스터를 통해 Stop bit, Parity bit 등의 셋팅을 한다.

UART

```
void USART_Transmit(unsigned char data)
{
    while(!(UCSRnA & (1 << UDREn)));

    UDRn = data;
}
```

UART의 Initialization 후에 데이터를 보내는 함수를 살펴보자.

먼저 보내고자하는 데이터를 매개변수로 넘겨주고

while문안의 로직이 거짓이 될때까지 기다린다.
while문안의 로직을 살펴보면 UDREn 비트가 0일때 거짓 1일때는 참이다.

UDREn 비트는 버퍼에 데이터가 존재하는지 안하는지 확인하는 비트이다.
즉, 위의 while문 로직은 UART가 현재 통신중인지 아닌지 판별하는 로직이다.

UART

UART(Universal asynchronous receiver/transmitter)는 범용 비동기화 송수신기라고 부르며 데이터를 비동기 직렬로 전송하는 방식이며 전이중 통신이 가능하다.

UART 통신시 주고 받는 데이터의 형태는 아래와 같다.

비트수	1	2	3	4	5	6	7	8	9	10	11
	시작 비트 (Start bit)	5–8 데이터 비트								패리티 비트 (parity bit)	종료 비트 (Stop bit(s))
	Start	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7	Parity	Stop

출처 : https://ko.wikipedia.org/wiki/UART

1. Start bit : 데이터 통신의 시작을 알리는 비트
2. Data bit : 실제 전송할 데이터 비트자리이며 7, 8 bit 설정이 가능하다.
3. Parity bit : 오류 검증시 사용하는 비트이며 짝수, 홀수 패리티 설정이 가능하다. 수신부에서 데이터를 받은 후 1의 갯수가 짝수인지 홀수 인지 확인한다.
4. Stop bit : 데이터 통신의 종료를 알리는 비트이며 1, 1.5, 2비트등 으로 설정이 가능하다.

UART

UART통신은 비동기 통신이기 때문에 BAUD RATE라는 통신 속도를 수신부와 송신부에서 일치를 해줘야 데이터를 제대로 읽을 수 있다.

Atmega328p의 Baud Rate 속도 테이블은 아래와 같으며 단위는 bps이다.

Table 19-12. Examples of UBRRn Settings for Commonly Used Oscillator Frequencies

	f_{osc} = 16.0000MHz			
	U2Xn = 0		U2Xn = 1	
Baud Rate (bps)	UBRRn	Error	UBRRn	Error
2400	416	–0.1%	832	0.0%
4800	207	0.2%	416	–0.1%
9600	103	0.2%	207	0.2%
14.4k	68	0.6%	138	–0.1%
19.2k	51	0.2%	103	0.2%
28.8k	34	–0.8%	68	0.6%
38.4k	25	0.2%	51	0.2%
57.6k	16	2.1%	34	–0.8%
76.8k	12	0.2%	25	0.2%
115.2k	8	–3.5%	16	2.1%
230.4k	3	8.5%	8	–3.5%
250k	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%
Max.(1)	1Mbps		2Mbps	

Note: 1. UBRRn = 0, error = 0.0%

UART

이전 슬라이드에서 보여준 테이블은 Atmega328p의 시스템 주파수를 16Mhz로 사용할때의 Baud rate와 오차율이 나와있는 테이블이다.

여기서 오차율이라는 단어가 나왔는데 Uart의 큰단점은 바로 비동기식 통신 방식이기 때문에 서로 다른 주파수에서 동작하는 시스템 사이의 오차가 발생 할 수 있다는 점이다.

위의 테이블에서 예를 들어보면

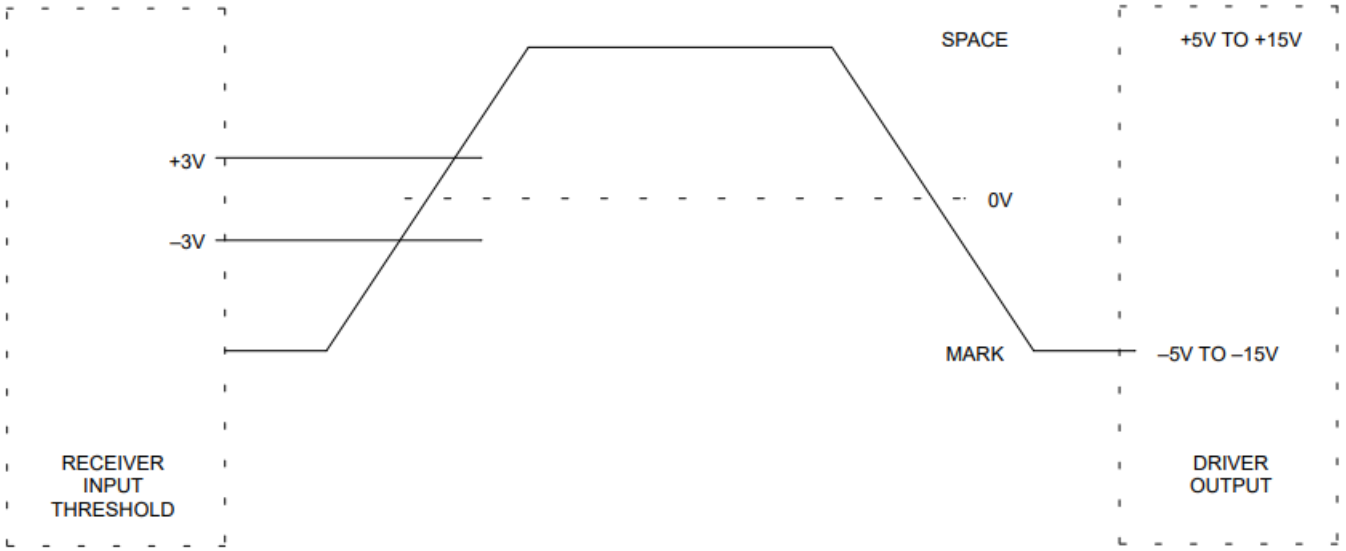
Baud rate 9600에서 오차율은 0.2%이다.

RS-232

앞에서 보았던 UART는 비동기 직렬통신의 방식을 서술한것이고 이러한 통신이 원활하게 되기 위해서 하드웨어 스펙을 정해놓은것이 RS-232라고 한다.

먼저 RS-232의 신호 level먼저 살펴보면 RS-232 규격은 전압 level을 +-5V ~ +-15V로 제한하였다. 또한 High or Low로 인식되는 Threshold +3V, -3V인것을 아래 그림에서 확인 할 수 있다.

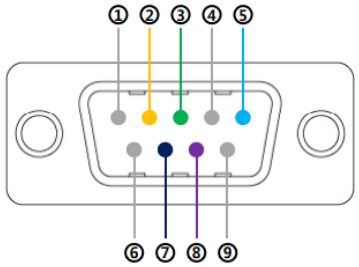
RS–232 LOGIC LEVEL SPECIFICATIONS Figure 1



RS-232

또한 RS-232 규격에서 주로 사용되는 커넥터는 D-subminiature라는 9핀 짜리 커넥터를 많이 사용한다.

D-sub connector



RS-232 핀 맵

핀 번호	핀 이름	설 명
①	NC	Do not used
②	RX	Receive DATA
③	TX	Transmit DATA
④	NC	Do not used
⑤	GND	Ground
⑥	NC	Do not used
⑦	RTS	Request to send
⑧	CTS	Clear to send
⑨	NC	Do not used

NS-RS232_datasheet 참고

RS-232 Pin map

Request To Send	DTE requests the DCE prepare to transmit data.	RTS	out	in	4
Ready To Receive	DTE is ready to receive data from DCE. If in use, RTS is assumed to be always asserted.	RTR	out	in	4
Clear To Send	DCE is ready to accept data from the DTE.	CTS	in	out	5

RS-232

전 슬라이드의 빨간색 박스 안을 살펴보면 RTS/CTS 시그널을 확인 할 수 있다.

이 2개의 시그널은 Hardware Flow control을 하기위한 시그널이며 기능은 아래와 같다.

RTS : 송신자가 수신자에게 데이터를 보낸다는것을 알려준다.
CTS : 수신자가 송신자에게 데이터를 받을 준비가 됐다는것을 알려준다.

요약하자면 데이터를 주고받는 과정을 기존의 UART 시그널 외에 다른 시그널로 조정하겠다는 의도이다.

RS-232
싱글엔드(Single-ended Signal)
10m전후
20kb/s

왼쪽의 표는 RS-232표준을 만든 EIA라는 단체에서 권고하는 RS-232 규격 사용시 권장사항이다.

거리는 10m전후 속도는 최대 20kb/s 까지 권장하고 있다.

RS-422

RS-232 규격 말고도 RS-422이라는 규격도 존재한다. RS-422의 특징은 다음과 같다.

Signal Voltage level : -10V to +10V
Signal : TX+, TX-, RX+, RX- (Differential Signal)사용
Maximum speed : 10M bps
Maximum Distance : 1.2km
Multi-Drop

RS-232와 비교하였을때 가장 큰 특징은 속도가 10Mbps까지 전송 가능하다는 것과 신호라인이 Differential Signal을 사용하는점을 알 수 있다.

그리고 Multi-Drop이가능한데 기존의 RS-232는 Point to Point 통신만 가능했다면 RS-422은 1 : n 통신이 가능하다.

RS-422

RS-422 사용시 이전슬라이드에서 최대 속도가 10Mbps라고 설명 하였는데 거리에따라 전송 가능한 속도에 차이가 있다. 아래 그래프는 거리에따른 전송속도를 나타낸다.
대략 150m까지는 10Mbps로 전송이 가능한것을 확인 할 수 있다.

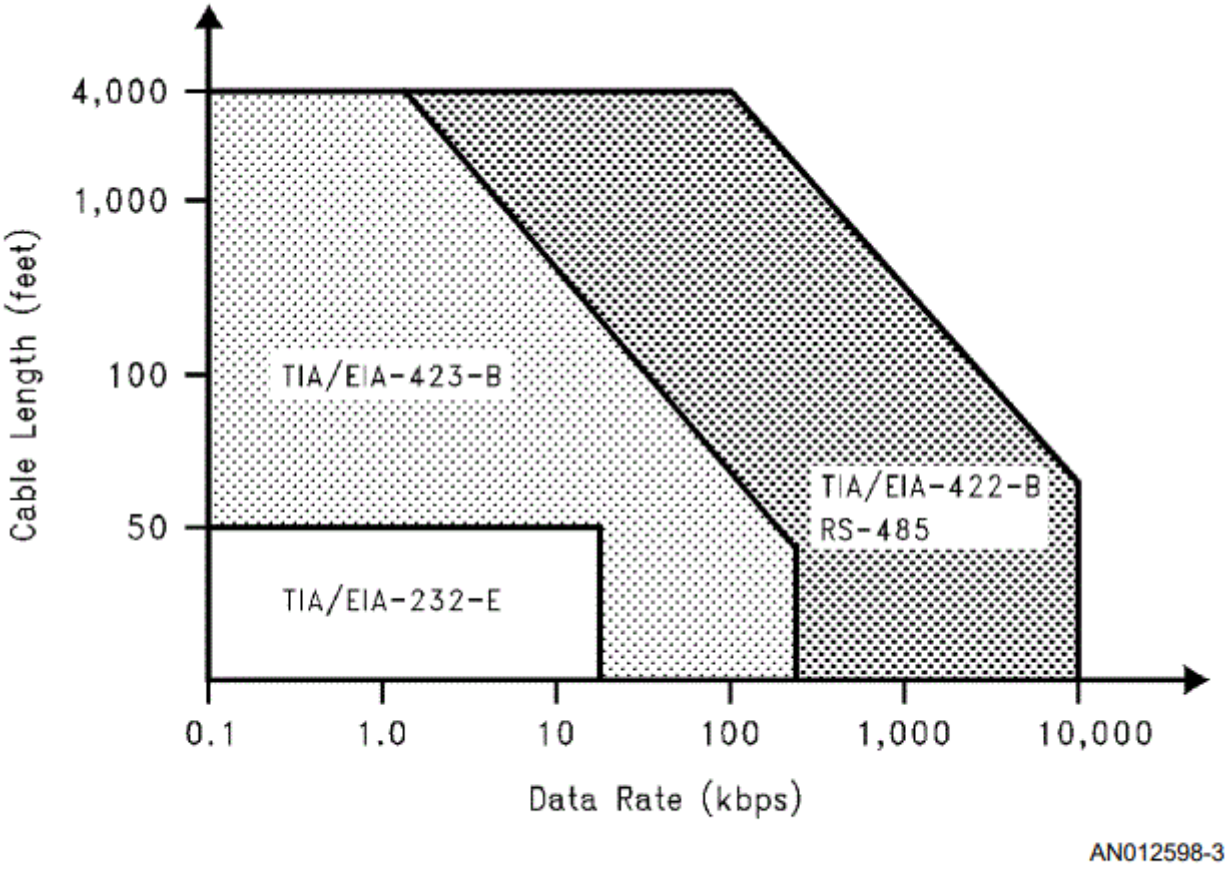


FIGURE 3. Cable Length vs. Data Rate

RS-485

마지막으로는 RS-485 규격을 살펴보자.

Signal Voltage level : -7V to +12V
Signal : TX+, TX-, RX+, RX- (Differential Signal)사용
Maximum speed : 10M bps
Maximum Distance : 1.2km
최대 32개의 Driver 와 Receiver 사용 가능

RS-422와 비교 했을때 큰 차이점은 Signal level 정도 이다.

또한 RS-485 규격을 만족 했을시 RS-422도 사용이 가능하다는 점이 특징이다.

이전 슬라이드에서 사용한 케이블 길이별 전송 가능 속도이다. RS-485와 RS-422이 동일하다.

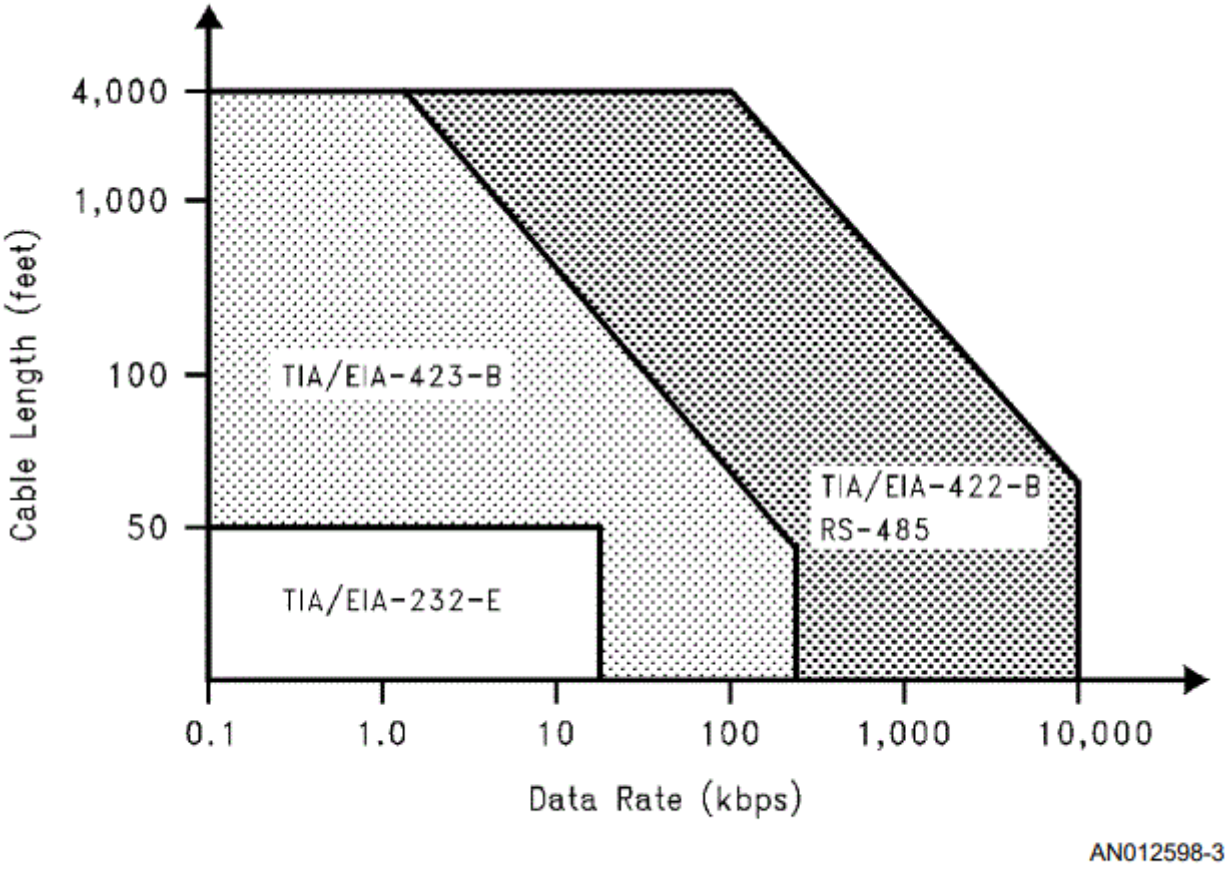


FIGURE 3. Cable Length vs. Data Rate

RS-422

RS-422 사용시 이전슬라이드에서 최대 속도가 10Mbps라고 설명 하였는데 거리에따라 전송 가능한 속도에 차이가 있다. 아래 그래프는 거리에따른 전송속도를 나타낸다.
대략 150m까지는 10Mbps로 전송이 가능한것을 확인 할 수 있다.

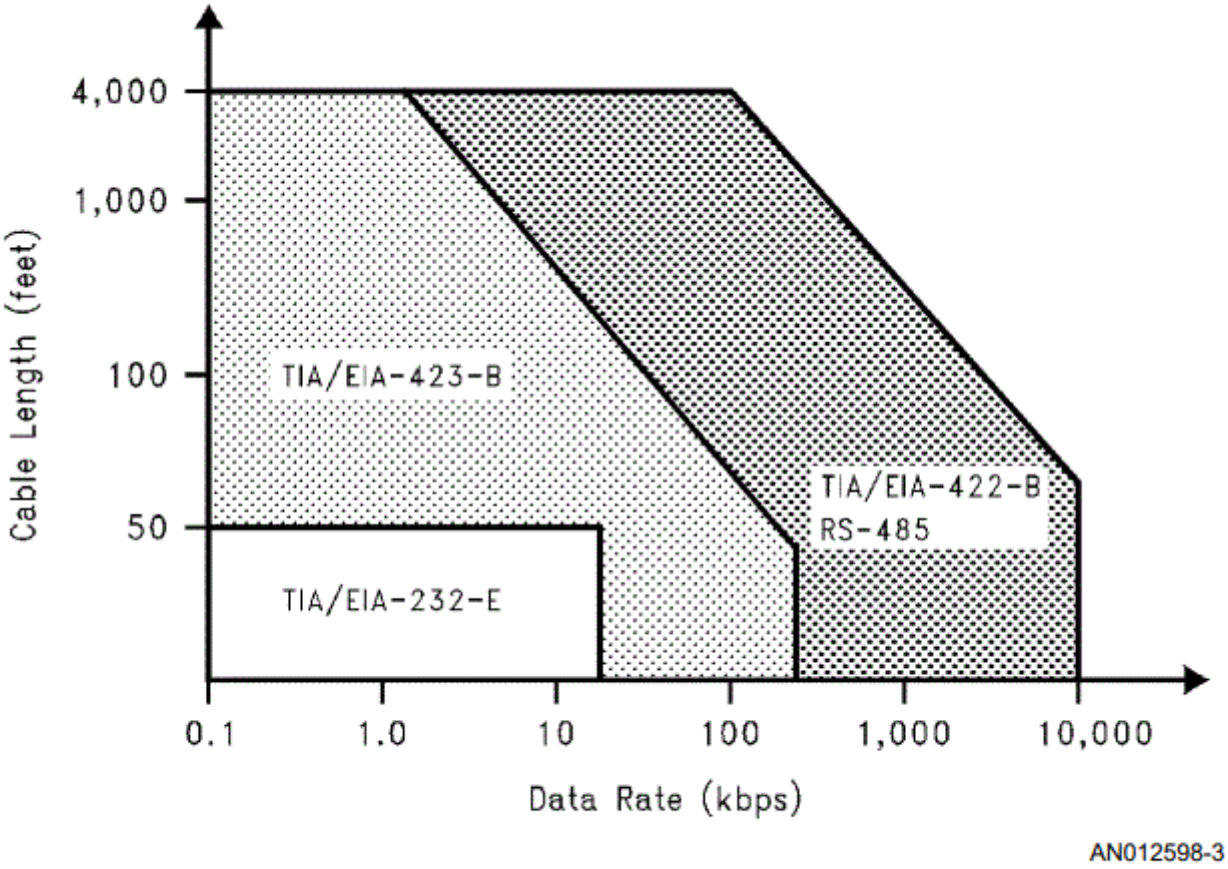


FIGURE 3. Cable Length vs. Data Rate

Bottom Half란?

- 임베디드 기기들에는 수많은 인터럽트 요청이 들어올 수 있는데 이때 인터럽트 발생시 실행되는 ISR에서 CPU 자원을 많이 사용하는 로직이 들어있다면 ISR을 처리하느라 전체 시스템이 멈추는 현상이 생길 수 있다.

- 이러한 현상을 막기위해 실제 자원이 많이 사용되는 로직은 Task에서 처리해주는 방식이다.

Bottom Half란?

- 임베디드 기기들에는 수많은 인터럽트 요청이 들어올 수 있는데 이때 인터럽트 발생시 실행되는 ISR에서 CPU 자원을 많이 사용하는 로직이 들어있다면 ISR을 처리하느라 전체 시스템이 멈추는 현상이 생길 수 있다.

- 이러한 현상을 막기위해 실제 자원이 많이 사용되는 로직은 Task에서 처리해주는 방식이다.

Bottom Half란?

- 임베디드 기기들에는 수많은 인터럽트 요청이 들어올 수 있는데 이때 인터럽트 발생시 실행되는 ISR에서 CPU 자원을 많이 사용하는 로직이 들어있다면 ISR을 처리하느라 전체 시스템이 멈추는 현상이 생길 수 있다.

- 이러한 현상을 막기위해 실제 자원이 많이 사용되는 로직은 Task에서 처리해주는 방식이다.

인터럽트 disable
Flag on
인터럽트 enable
인터럽트발생 한 로직 처리

Bottom Half

```c
unsigned char g_isr_flag;

void ISR(void)
{
    g_isr_flag = 1;
}

void job1(void const *argument)
{
    int i;

    while(1)
    {
        /* logic start */
        if (g_isr_flag)
        {
            for (i = 0; i < 100000; i++)
            {

            }

            g_isr_flag = 0;
        }

        /* logic end */

        osDelay(100) // 100ms
    }
}

int main(void)
{


    osKernelInitialize ();   // initialize RTOS kernel

    osThreadCreate (osThread(job1)); // create Task

    osKernelStart (); // Start Task

    return 1;
}
```

Bottom Half 코드 예문을 보면

ISR발생시 g_isr_flag를 1로 바꾼 후 job1 Task 실행시 flag 체크 후 loop문을 처리하는것을 확인 할 수 있다.

grep이란?

리눅스 명령어 중 하나이며 문자열을 검색할때 사용한다.

※ 옵션

-c : 검색할 문자열이 속한 행의 개수를 출력한다.
-H : 파일 이름과 함께 출력을 한다.
-i : 대소문자를 구분하지 않고 출력한다.
-n : 찾으려는 문자가 속해있는 행의 번호와 같이 출력한다.
-r : 현재 경로부터 하위경로까지 검색해서 출력을 한다.
-v : 찾을려고 하는 문자가 없는 행을 출력한다.
-w : 패턴 표현식을 하나의 단어로 취급하여 검색한다.

grep을 이용한 소스코드 분석 예시

```
eyl@eyl-VirtualBox:~/test$ grep hello test.c
        char str[20] = "hello!";
eyl@eyl-VirtualBox:~/test$ grep -n hello test.c
6:      char str[20] = "hello!";
eyl@eyl-VirtualBox:~/test$ cd ..
eyl@eyl-VirtualBox:~$ grep -r hello test
test/test.c:    char str[20] = "hello!";
eyl@eyl-VirtualBox:~$ grep -nr hello test
test/test.c:6:  char str[20] = "hello!";
eyl@eyl-VirtualBox:~$
```

grep hello test.c : test.c 파일 내부에서 hello가 들어가있는 행을 출력해준다.

grep –n hello test.c : test.c 파일 내부에서 hello가 들어가있는 행의 숫자까지 출력해준다.

grep –r hello test : test디렉토리안에서 hello가 들어가있는 파일과 행을 출력해준다.

grep –nr hello test : test디렉토리안에서 hello가 들어가있는 파일과 행넘버를 같이 출력해준다.

Project BOM

MultiPlayer (2021-06-03)				
For Assembly				
Item	Quantity	Part	Package	Maker
1	1	ATSAMD21J16B	TQFP-68	
2	1	AZ1117-3.3	SOT23-3	
3	2	3.5mm 스테레오 커넥터		
4	1	TFT Display module		
5	1	SD Card Slot		
6	1	Battery Holder		
7	1	2.1pi Power Jack		
8	1	FM Receiver module		

Project Block Diagram