C basic language

임베디드스쿨 2기

Lv1과정

2021. 06. 04

김효창

Cache

CPU Cache 는 데이터가 메모리에 접근할 때 대기 시간을 줄인다.
자주 사용되는 메인 메모리 위치의 데이터 복사본을 저장합니다..
성능 향상시키기 위해 CPU Cache 를 활용해야 한다.

Cache 는 L1 , L2 , L3 가 있다.

CPU Cache 에서 필요할 수 있는 것을 찾기 위해 코드를 최적화 하려고 한다.

코드는 데이터 명령어와 데이터가 있는 위치를 지정할 수 없습니다.
컴퓨터 하드웨어가 그렇게 하므로 특정 요소를 CPU 캐시에 강제로 넣을 수 없다.

CPU는 메모리에 액세스 할 때마다 새로운 캐시 라인이 필요합니다. 이는 성능을 저하시킵니다.

프로세서가 L1 캐시에서 찾고 있는 것을 찾을 수 없다면, 그것은 L1 캐시의 누락률이다. 그런 다음 프로세서가 L2 캐시로 이동하여 데이터를 찾습니다. 사용 가능한 데이터가 있으면 프로세서가 데이터를 사용하거나 프로세서가 L3으로 이동하여 데이터를 조회합니다.

시스템이 다음에 필요한 데이터를 정확하게 예측할 수 있는 경우 해당 데이터를 캐시에 미리 로드하여 필요한 경우 더 빠른 블록이 대기할 필요가 없도록 할 수 있습니다.

Cache

성능 향상을 위해 CPU 캐시를 가장 잘 활용하는 코드를 작성하는 방법은 무엇입니까?

효과적/캐시에 적합한 코드(캐시 적중률, 캐시 누락 횟수)를 만드는 방법

두 가지 관점 모두에서 데이터 캐시와 프로그램 캐시(명령 캐시) 즉, 데이터 구조와 코드 구성과 관련된 자신의 코드에서 어떤 것들이 캐시를 효과적으로 만들기 위해 다루어져야 한다.

사용/사용하지 않아야 하는 특정 데이터 구조가 있습니까?
아니면 코드 캐시를 효과적으로 만들기 위해 해당 구조의 멤버 등에 액세스하는 특정 방법이 있습니까?

이 문제에서 따라야 할 프로그램 구조 (if, for, switch, break, goto, ...), 코드 흐름 (if 내부, for 내부 경우 등)이 있습니까?

메모리 가져 오기 지연으로 인한 고통을 피하는 기술은 일반적으로 가장 먼저 고려해야 할 사항이며 때로는 큰 도움이됩니다.

Cache

일반적인 기술은 다음과 같습니다.

더 작은 데이터 유형 사용

정렬 구멍을 피하기 위해 데이터를 구성합니다
(크기를 줄여 구조체 멤버를 정렬하는 것은 한 가지 방법입니다).
Organize your data to avoid alignment holes (sorting your struct members by decreasing size is one way)

표준 동적 메모리 할당자에 주의하십시오.
이 할당자는 예열 될 때 메모리에 구멍이 생기고 데이터가 분산 될 수 있습니다.
Beware of the standard dynamic memory allocator, which may introduce holes and spread your data around in memory as it warms up.

모든 인접 데이터가 실제로 핫 루프에서 사용되는지 확인하십시오. 그렇지 않으면 핫 루프가 핫 데이터를 사용하도록 데이터 구조를 핫 및 콜드 구성 요소로 분할하는 것이 좋습니다.

불규칙한 액세스 패턴을 나타내는 알고리즘 및 데이터 구조를 피하고 선형 데이터 구조를 선호합니다.

최적화

모두 최적 optimal 인 시스템은 없다.

수행 시간, 메모리 등 어떤 자원에 우선 순위를 두는가에 따라 최적화 방향이 달라질 수 있다.

“섣부른 최적화는 모든 악의 근원이다.” – 도널드 커누스, 컴퓨터 과학자

나눗셈은 매우 느린 연산. (시프트 연산 활용하면 시간 절약)

분기문 : 다음에 실행할 명령어가 무엇인지 모른다.

예측이 맞다면 정상 진행할 수 있지만, 틀리면 작업한 것을 모두 버리고 기존 수행해야 될 명령어를 다시 실행

비트 연산 (OR, AND, XOR 등등) 은 컴퓨터에서 가장 빠르게 실행되는 연산.

최적화된 소스 코드는 같은 성능의 하드웨어에서도 수행 속도를 높일 수 있으며 메모리 사용량도 최소화 할 수 있다.

메모리 사용량의 감소는 비용 절감 효과.

실행 속도가 빠른 코드는 애플리케이션의 응답성을 높여주므로 기능 구현에 도움 된다.
#pragma 를 활용하면 최적화와 함수 단위로 지정 가능.

Type-based alias analysis: 데이터 타입이 다르면 별도의 선언이 없는 경우, 서로 다른 위치에 데이터가 존재하게 된다.

동기 , 비동기

동기 방식 : 클럭의 업다운에 의해 시작과 끝을 결정하게 된다.
또한 데이터의 처리 역시 동기 방식을 취할 수 있음.
최근 트렌드에서는 속도 측면 때문에 시작과 끝을 동기 처리하고 중간 데이터는
비동기로 처리하는 편 (하이브리드 방식)이다. (I2C , SPI)

비동기 방식 : 클럭의 업다운에 영향을 받지 않으며, 송/수신측이 통신속도를 서로 약속하고,
최대한 빨리 여러 데이터를 처리하는 방식이다. (RS232, RS485, 무선통신)

Polling (TX)

부모님이 오늘 집에 오기로 했는데 도착하기 전까지 수시로 집 밖으로 나가서 확인하는 것을 반복.

Interrupt (RX)

부모님이 오늘 집에 오기로 했는데 집 안에서 게임하다가 부모님이 도착해서 초인종을 눌렀을 때
밖으로 나가서 확인.

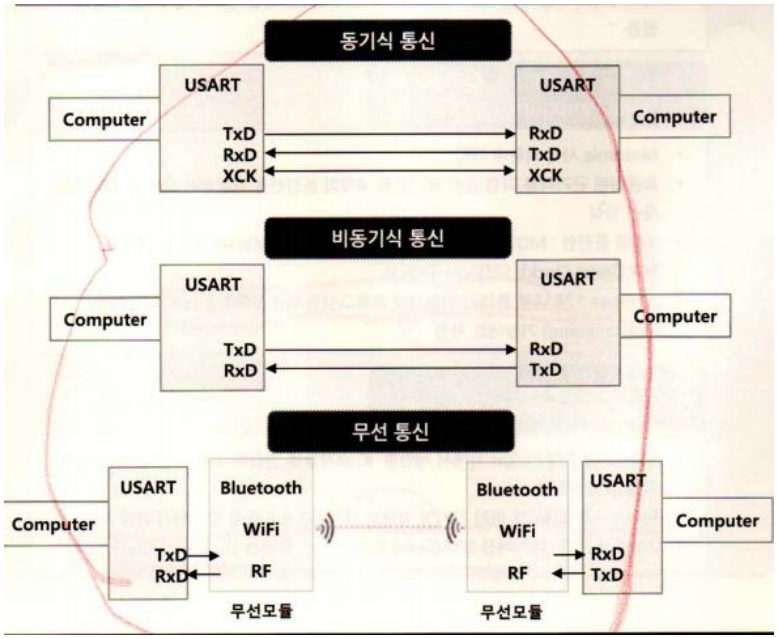
MCU 입장에선 USART 만 다룬다. RS232, RS485 는 X

시간적인 간격을 가지고 프레임을 구분하는 것 보다, 데이터의 내용물을 가지고 프레임을 구분하도록 코딩

USART

송수신이 동시에 가능한 USART를 2개 내장.
높은 정밀도의 Baud Rate Generator 내장
5 ~ 9 개의 Data Bit , 1 ~ 2 개의 Stop Bit 설정
짝수, 홀수 패리티 설정.
데이터 OverRun , 프레임 오류 검출.
3 개의 인터럽트 소스 : 송신 완료, 송신 데이터 레지스터 빔 (empty), 수신 완료

USART 는 임베디드용 printf
USART 단점 : 오차율



USART Data Register
- UDR 에 write 하면 TxBn 에 값이 쓰여지며,
  전송 종료되면 자동으로 데이터가 없는 상태가 된다.
- UDR 에 Read 하면 RxBn 에 값이 읽혀지며,
  읽은 후에는 데이터가 없는 상태가 된다
- 송신 버퍼에는 UCSRA 의 UDRE 플래그 비트가 1 일 때만
  값을 write 할 수 있으며, 0 일 때 write 하면 무시.
- shift 가 비어 있으면 송신 버퍼에 쓰여진 데이터는
  shift 에 이동하여 TxD pin 을 통해 1 Bit 씩 전송.

USART Control Status Register
송신과 수신 완료 인터럽트 발생과 송수신 기능 활성화.
RXCIEn 를 1 로 설정 후, 전역 인터럽트를 활성화 하면, UDR 에 수신 문자가 있을 때.
인터럽트가 발생된다.

USART

	레지스터 n = 0 ,1	Bit	레지스터	
USART0 USART1	UDRn	8	USARTn I/O Data	송신, 수신 데이터 저장
	UCSRnA	8	USARTn Control Status	USARTn 동작 모드 설정
	UCSRnB	8		
	UCSRnC	8		
	UBRR0H	8	USARTn Baud Rate	전송 속도 설정
	UBRR0L	8		

CODE

delay 함수 사용시

#define F_CPU 16000000UL 를 먼저 선언. (클럭 값)
UL = unsigned long : 0 ~ 18,446,744,073,709,551,615

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

USART 통신 속도 매크로 정의
매크로 명 : USART_BAUDRATE
상수 값 : 9600

```
#define USART_BAUDRATE 9600
```

Baud rate 값 얻기 위한 UBRR 값 계산식

```
#define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16UL)))-1)
```

USART

UDR : 송신 , 수신 데이터 임시 저장.
UCSZ02 , UCSZ01 , UCSZ00 : 데이터 SIZE 결정
UBRR0H , UBRR0L : 전송 속도 12 Bit 설정 (1초에 몇 Bit 를 전송할 것인가)
RXEN1 , TXEN1 : 직렬 통신 송신/수신 Port pin 으로 설정
UDREn : UDR 에 내용 있음 = 0 , 내용 없음 = 1
U2X0 : 비동기 일반 모드 , 비동기 2배속 모드 , 동기 마스터 모드 설정
UMSEL : 비동기 , 동기 모드 설정.
USBS : Stop bit 설정
RXC0 : 읽지 않은 유효한 데이터가 있을 때 = 1
TXC0 : Shift register 의 데이터가 모두 송신되고, UDR 에 data 가 write 되지 않은 상태이면 = 1

UDRE 가 TxC 보다 전송 속도 면에서는 빠르다.

(0xC6)	UDR0	USART I/O data register								159
(0xC5)	UBRR0H					USART baud rate register high				162
(0xC4)	UBRR0L	USART baud rate register low								162
(0xC3)	Reserved	–	–	–	–	–	–	–	–	
(0xC2)	UCSR0C	UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01 /UDORD0	UCSZ00 / UCPHA0	UCPOL0	161/172
(0xC1)	UCSR0B	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80	160
(0xC0)	UCSR0A	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0	159

USART

U2X = 0 일 때 16 MHz 크리스탈 사용

$$UBRR = \left(\frac{16\times10^6}{16\times9600}\right) - 1 = 103.1667$$

$$Baud = \left(\frac{16\times10^6}{16\times(103+1)}\right) = 9615.385$$

$$Error = \left(\frac{9615.385}{9600-1}\right) \times 100 = 0.16\ \%$$

Table 19-1. Equations for Calculating Baud Rate Register Setting

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)

BAUD	Baud rate (in bits per second, bps)
f_{osc}	System oscillator clock frequency
UBRRn	Contents of the UBRRnH and UBRRnL registers, (0-4095)

Some examples of UBRRn values for some system clock frequencies are found in Table 19-9 on page 163.

타이밍 블록 다이어그램

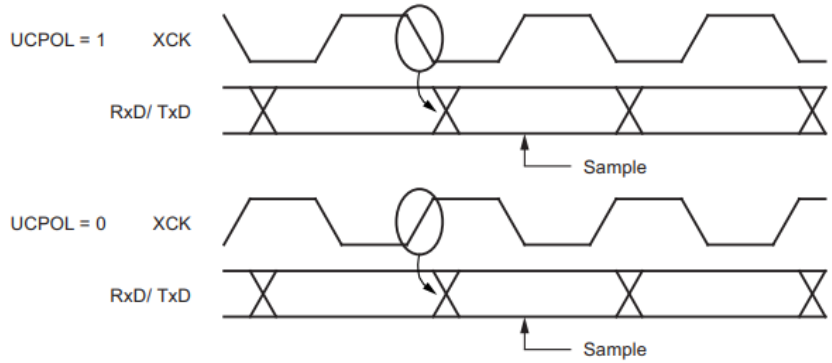
UCPOLn: Clock Polarity

선택 여부에 따라 falling , rising 으로 구분
동기 모드에만 사용 (비동기일 경우 0 으로 설정)

Table 19-8. UCPOLn Bit Settings

UCPOLn	Transmitted Data Changed (Output of TxDn Pin)	Received Data Sampled (Input on RxDn Pin)
0	Rising XCKn edge	Falling XCKn edge
1	Falling XCKn edge	Rising XCKn edge

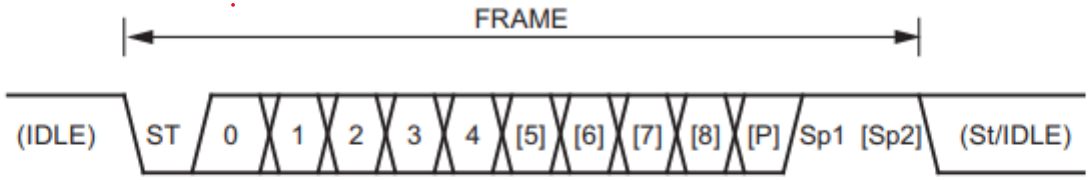
Figure 19-3. Synchronous Mode XCKn Timing



- 1 start bit
- 5, 6, 7, 8, or 9 data bits
- no, even or odd parity bit
- 1 or 2 stop bits

A frame starts with the start bit followed by the least significant data bit. Then the next data bits, up to a total of nine, are succeeding, ending with the most significant bit. If enabled, the parity bit is inserted after the data bits, before the stop bits. When a complete frame is transmitted, it can be directly followed by a new frame, or the communication line can be set to an idle (high) state. Figure 19-4 illustrates the possible combinations of the frame formats. Bits inside brackets are optional.

Figure 19-4. 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 must be high.

USART

U2X 는 clock 의 2배속 사용 여부를 결정한다.
1 일 경우 2배속.

16 MHz 에서 9600 bps 로 통신할 경우 103 으로 설정.

Baud Rate (bps)	f_{osc} = 16.0000MHz			
	U2Xn = 0		U2Xn = 1	
	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%

CODE

UCSRnC : Bit 6 을 0 으로 설정. (0 = 비동기 , 1 = 동기)
PARITY_MODE <= DISABLED <= (0 << UPM00)

```
#define ASYNCHRONOUS (0 << UMSEL00)
```

10진수	UPMn1	UPMn0	패리티 모드
0	0	0	사용하지 않음
	0	1	-
2	1	0	짝수 패리티
3	1	1	홀수 패리티

```
#define DISABLED (0 << UPM00)
#define EVEN_PARITY (2 << UPM00)
#define ODD_PARITY (3 << UPM00)
#define PARITY_MODE DISABLED
```

CODE

USBSn : Stop Bit Select 정지 비트 선택
0 = 1 개 (1 Bit) 로 설정
1 = 2 개 (2 Bit) 로 설정

```
#define ONE_BIT (0 << USBS0)
#define TWO_BIT (1 << USBS0)
#define STOP_BIT ONE_BIT
```

UCSZ00 : Character Size 전송 데이터 Bit 수 결정.

UCSZn2	UCSZn1	UCSZn0	데이터 비트 수
0	0	0	5 비트
0	0	1	6 비트
0	1	0	7 비트
0	1	1	8 비트
1	1	1	9 비트

```
#define FIVE_BIT (0 << UCSZ00)
#define SIX_BIT (1 << UCSZ00)
#define SEVEN_BIT (2 << UCSZ00)
#define EIGHT_BIT (3 << UCSZ00)
#define DATA_BIT EIGHT_BIT
```

CODE

UBRRnH , UBRRnL : 전송속도 설정

2 Byte 공간 중 1 Byte 는 Read , 1 Byte 는 write
16 Bit 중에서 0 ~ 11 Bit 제어

BAUD_PRESCLAE = 103 (UBRR 값)
103 >> 8 : 오른쪽으로 8번 이동하여 상위 8 Bit UBRR0H 의 공간을 0 으로 채운다.
103 → 0110 0111 을 하위 8 Bit 에 대입.

UCSR0C = ASYNCHRONOUS | PARITY_MODE | STOP_BIT | DATA_BIT;

UMSEL01 / UMSEL00 = 0 / UPM01 / UPM00 = 0
/ USBS0 = 0 / UCSZ01 = 1 / UCSZ00 = 1 / UCPOL0

UCSR0B = (1 << RXEN0) | (1 << TXEN0);

USART 송신 수신 기능이 허용되면서 TxD , RxD pin 에 대한 일반 port 작동은 무시한다.

```
void USART_init()
{
    UBRR0H = BAUD_PRESCALE >> 8;
    UBRR0L = BAUD_PRESCALE;

    UCSR0C = ASYNCHRONOUS | PARITY_MODE | STOP_BIT | DATA_BIT;

    UCSR0B = (1 << RXEN0) | (1 << TXEN0);
}
```

Primitive System Data Type

각 자료형이 사용하는 Bit 수를 고정.

Bit 가 명확한 typedef 으로 정의된 자료형을 사용
크기를 정확하게 표현해야 하는 파일 압축 및 암호화, 일반적인 프로그래밍, 메모리 관리를 하기 위해 사용.

32 Bit 와 64 Bit 데이터 길이의 변화에 따른 버그, 에러를 방지하기 위해 define 해서 사용.

Specifier	Common Equivalent	Signing	Bits	Bytes	Minimum Value	Maximum Value
int8_t	signed char	signed	8	1	-128	127
uint8_t	unsigned char	unsigned	8	1	0	255
int16_t	short	signed	16	2	-32,768	32,767
uint16_t	unsigned short	unsigned	16	2	0	65,535
int32_t	long	signed	32	4	-2,147,483,648	2,147,483,647
uint32_t	unsigned long	unsigned	32	4	0	4,294,967,295
int64_t	long long	signed	64	8	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
uint64_t	unsigned long long	unsigned	64	8	0	18,446,744,073,709,554,615

CODE

void USART_TransmitPolling(uint8_t DataByte)

void : 값을 반환하지 않으므로 return 필요없다. UDR0 를 출력해라.

(uint8_t DataByte) : 함수에 입력되는 값을 의미

while((UCSR0A & (1 << UDRE0))==0) {};

송신 끝날 때까지 대기 (UDR 비워지도록 대기)

UDR0 = DataByte;

파라미터 uint8_t DataByte 를 제공 받아서 읽고 , 값을 UDR0 에 쓰기
보낼 데이터가 있으면 전송

```
void USART_TransmitPolling(uint8_t DataByte)
{
    while((UCSR0A & (1 << UDRE0))==0) {}
    UDR0 = DataByte;
}
```

CODE

USART_init(); USART TX 초기 설정 함수 정의

USART_TransmitPolling(‘ ‘); 하나의 문자 송신 함수 정의

17 개를 1 Byte 씩 전송

＼ r : 현재 줄(행)에서 맨 앞으로 이동
＼ n : 다음 줄로 갑니다.
＼ 0 : 자동으로 초기화된 문자열 다음에 null 문자의 끝 알림.
포인터 변수를 초기화

문자	ASCII	16진수
K	75	0x4B
I	73	0x49
M	77	0x4D
	32	0x20
H	72	0x48
Y	89	0x59
O	79	0x4F
	32	0x20
C	67	0x43
H	72	0x48
A	65	0x41
N	78	0x4E
G	71	0x47
!	33	0x21
\ r	13	0xD
\ n	10	0xA
₩0	0	0x0

```c
int main(void)
{
    USART_init();
    while(1)
    {
        USART_TransmitPolling('K');
        USART_TransmitPolling('I');
        USART_TransmitPolling('M');
        USART_TransmitPolling(' ');
        USART_TransmitPolling('H');
        USART_TransmitPolling('Y');
        USART_TransmitPolling('O');
        USART_TransmitPolling(' ');
        USART_TransmitPolling('C');
        USART_TransmitPolling('H');
        USART_TransmitPolling('A');
        USART_TransmitPolling('N');
        USART_TransmitPolling('G');
        USART_TransmitPolling('!');
        USART_TransmitPolling('\r');
        USART_TransmitPolling('\n');
        USART_TransmitPolling('\0');

        _delay_ms(1000);
    }

    return 0;
}
```

Memory Map

0x20 은 default
DDRB = 0x04
SFIR OFFSET 0x20
8bit = 1 Byte
DDRA .... 무조건 1 Byte 순서대로 붙어 있는 것은 아니다

송신 완료 인터럽트 : 데이터 → UDR → shift register → TxD 로 모두 나가면 완료
송신 빈 인터럽트 : UDR → shift register 로 모두 들어오면 UDR 이 비어 있다.
다시 데이터를 UDR 에 채울 수 있다.

stop 비트가 왜 2개 ? 이어서 전송하게 된다면 , 보낼 데이터가 추가로 있으면 다음 단에 low
없으면 idle (high 유지)

오실로스코프로 측정하려면 1 개의 frame 이 몇 sec 소요하는지 파악.
bps 를 주파수로 보고 역산 하면 1 Bit 에 걸리는 주기가 나온다.
sec 로 계산된 주기를 us 로 전환하면 (× 1,000,000)
1 Bit = 103us 가 계산된다.
stop Bit 는 계산하지 않기 때문에 9 개의 비트만 계산 (Start Bit 1 개 , Data Bit 8 개)
9 개 Bit 의 주기 ?

ubuntu

header file (#include 를 이용하여 불러오기) 은 컴파일러에 의해 다른 소스 파일에 자동으로 포함된 파일.

어셈블리어 분석 gdb -q filename.out 또는 간단한 공학 수학 계산용

```
5월  26  2020 alloca.h
5월  26  2020 assert.h
6월   4 16:07 avr
5월  26  2020 compat
5월  26  2020 ctype.h
5월  26  2020 errno.h
5월  26  2020 fcntl.h
5월  26  2020 inttypes.h
5월  26  2020 locale.h
5월  26  2020 math.h
5월  26  2020 setjmp.h
5월  26  2020 signal.h
5월  26  2020 stdfix-avrlibc.h
5월  26  2020 stdint.h
5월  26  2020 stdio.h
5월  26  2020 stdlib.h
5월  26  2020 string.h
5월  26  2020 sys
5월  26  2020 time.h
5월  26  2020 unistd.h
5월  26  2020 util
```

```
hyochangkim@hyochangkim-ThinkPad-X390-Yoga:~$ gdb
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) p/d (16000000 / (16*9600) - 1) >>8
$1 = 0
(gdb) p/d (16000000 / (16*9600) - 1)
$2 = 103
(gdb) p/f 9600.0/9615.0
$3 = 0.99843993759750393
(gdb) p 9600/9615
$4 = 0
```

ubuntu

grep : 찾는 문자열이 포함되었는지를 조사
-r : 서브 폴더까지 검색하도록 지정
-n : 해당 내용이 포함된 파일 내부에서의 줄 위치 (:828:)
-i : 알파벳 대문자 소문자 차이를 무시하고 검색.
“UPM00” : 찾고 싶은 내용
./ : 찾을 파일이 있는 폴더 경로 (검색 시작점)
| : 파이프라인으로 연결해서 출력 범위를 좁힐 수 있다. (여러 가지 한 번에 검색.)

```
r/include/avr$ grep -rn "UPM00" ./ | grep 328p
./iom328pb.h:828:#define UPM00   4
./iom328p.h:764:#define UPM00 4
```

```
r/include/avr$ grep -rn "DDRB" ./ | grep 328p
./iom328pb.h:46:#define DDRB    _SFR_IO8(0x04)
./iom328pb.h:47:#define DDRB7   7
./iom328pb.h:48:// Inserted "DDB7" from "DDRB7" due to compatibility
./iom328pb.h:50:#define DDRB6   6
./iom328pb.h:51:// Inserted "DDB6" from "DDRB6" due to compatibility
./iom328pb.h:53:#define DDRB5   5
./iom328pb.h:54:// Inserted "DDB5" from "DDRB5" due to compatibility
./iom328pb.h:56:#define DDRB4   4
./iom328pb.h:57:// Inserted "DDB4" from "DDRB4" due to compatibility
./iom328pb.h:59:#define DDRB3   3
./iom328pb.h:60:// Inserted "DDB3" from "DDRB3" due to compatibility
./iom328pb.h:62:#define DDRB2   2
./iom328pb.h:63:// Inserted "DDB2" from "DDRB2" due to compatibility
./iom328pb.h:65:#define DDRB1   1
./iom328pb.h:66:// Inserted "DDB1" from "DDRB1" due to compatibility
./iom328pb.h:68:#define DDRB0   0
./iom328pb.h:69:// Inserted "DDB0" from "DDRB0" due to compatibility
./iom328p.h:64:#define DDRB _SFR_IO8(0x04)
```

어셈블리에서 값을 할당 할 수 있도록 해당 레지스터의 번호 또는 주소가 무엇인지 이해

레지스터 주소 값 0x04 ?

_SFR_IO8 ()은 무엇을 의미?

AVR IO 주소 공간 레지스터에 액세스하는 매크로이며 해당 주소 공간에서 레지스터0x04는 DDRB로 보인다.

0x04는 IO 주소이고 0x24는 메모리 매핑 주소

```
#define _SFR_IO8(io_addr) _MMIO_BYTE((io_addr) + __SFR_OFFSET)
#define _SFR_IO16(io_addr) _MMIO_WORD((io_addr) + __SFR_OFFSET)
```

```
#define _MMIO_BYTE(mem_addr) (*(volatile uint8_t *)(mem_addr))
#define _MMIO_WORD(mem_addr) (*(volatile uint16_t *)(mem_addr))
#define _MMIO_DWORD(mem_addr) (*(volatile uint32_t *)(mem_addr))
#endif
```

```
#if (__SFR_OFFSET == 0x20)
/* No need to use ?: operator, so works in assembler too.  */
#define _SFR_ADDR(sfr) _SFR_MEM_ADDR(sfr)
#elif !defined(__ASSEMBLER__)
#define _SFR_ADDR(sfr) (_SFR_IO_REG_P(sfr) ? (_SFR_IO_ADDR(sfr) + 0x20) : _SFR_MEM_ADDR(sfr))
#endif

#else  /* !_SFR_ASM_COMPAT */

#ifndef __SFR_OFFSET
#  if __AVR_ARCH__ >= 100
#    define __SFR_OFFSET 0x00
#  else
#    define __SFR_OFFSET 0x20
#  endif
#endif
```

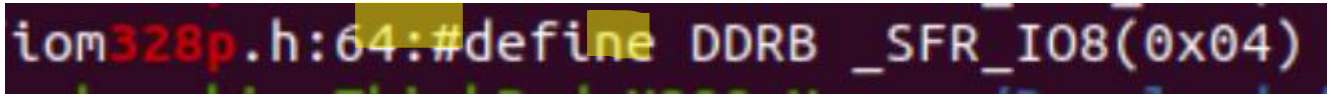
매크로 함수 통해 DDRx 레지스터로 접근 (최적화와 관련되어 있는 듯)

```
#define _SFR_IO8(io_addr) _MMIO_BYTE((io_addr) + _SFR_OFFSET)
→ SFR_OFFSET = 0x20

sfr_defs.h 에서 _MMIO_BYTE 검색
#define _MMIO_BYTE(mem_addr) (*(volatile uint8_t *)(mem_addr))

sfr_defs.h 에서 _SFR_OFFSET 검색
#ifndef __SFR_OFFSET
/* Define as 0 before including this file for compatibility with old asm
   sources that don't subtract _SFR_OFFSET from symbolic I/O addresses.  */
#  if __AVR_ARCH__ >= 100
#    define _SFR_OFFSET 0x00 ← 확장과 관련???
#  else
#    define _SFR_OFFSET 0x20 ← default
#  endif
#endif
```

```
PORTA = 0xFF;
_SFR_IO8(0x04) = 0xFF;
_MMIO_BYTE((0x04) + _SFR_OFFSET) = 0xFF;
(*(volatile uint8_t *))((0x04) + _SFR_OFFSET)) = 0xFF;
(*(volatile uint8_t *)(0x04) + 0x20)) = 0xFF;
(*(volatile uint8_t *)(0x24)) = 0xFF;
```

DDRB = 0x24 주소 값의 1 Byte 공간에 0xFF 쓰기

DDRx , PORTx , PINx 등등 매크로 함수로 되어 있다.

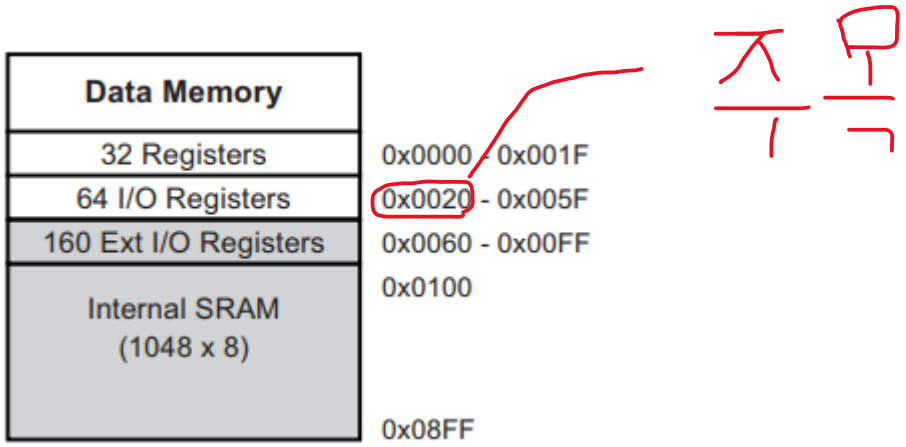


Memory Map

LD 및 ST 명령어를 사용하여 I / O 레지스터를 데이터 공간으로 주소 값 지정하는 경우 이러한 주소에 0x20을 추가 한다.

0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
0x08 (0x28)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
0x07 (0x27)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
0x06 (0x26)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
0x02 (0x22)	Reserved	–	–	–	–	–	–	–	–
0x01 (0x21)	Reserved	–	–	–	–	–	–	–	–
0x0 (0x20)	Reserved	–	–	–	–	–	–	–	–

Figure 7-2. Data Memory Map



1. 프로젝트 소개

product name

LED 제어

Function

전원부는 5V , 12V , 24V 中 택1

약 5 ~ 10 개 정도의 LED 를 순차적으로 ON/OFF 하며 반복하는 것.

LED 의 ON 시간과 OFF 시간을 버튼을 통해 사용자가 설정.

ON / OFF 시간 및 카운터는 LCD 또는 7-segment 를 통해 숫자를 표기.

카운터 설정을 통해 반복되는 구간을 설정.

LED 구간을 정해 반복 작동하기 위한 값으로 카운터가 있다.
- 5 로 설정하면 1 ~ 5 라인만 반복.
- 10 으로 설정하면 1 ~ 10 라인만 반복

2. 프로젝트 일정 및 계획

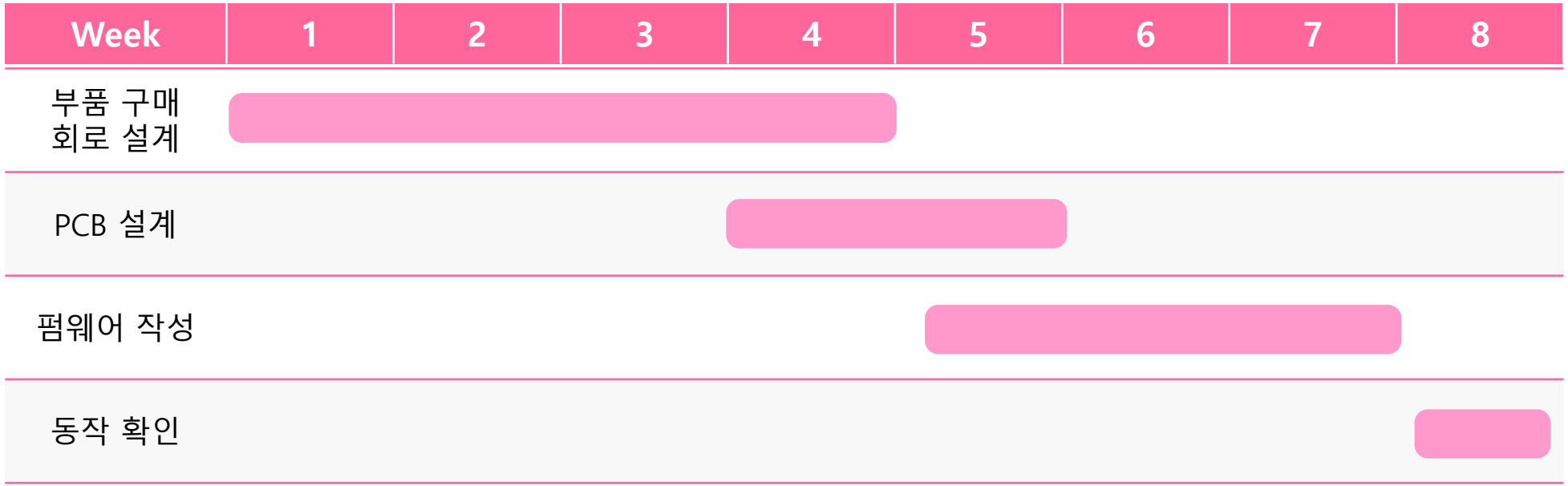
부품 구매 : 판다파츠 또는 디바이스 마트에서 부품 구매.

회로 설계 : 오실로스코프 , 함수 발생기 , 디지털 멀티미터 2개 사용.

PCB 설계 : 만능기판으로 납땜.

펌웨어 작성 : Function 을 C 코드로 구현.

동작 확인 : 장시간 동안 동작해도 문제 없는지 확인. (디버깅)



3. BOM 리스트

일련번호	품목	규격	소요량	비고
1	KEYPAD	Adafruit Industries 3845	2	
2	MCU	ATMEGA328P-PU	2	
3	MCU	ATMEGA4809-PB	3	
4	PROGRAMMER	PG164140	1	
5	IC - CMOS	CD4532	1	
6	IC - CMOS	CD4042B	1	
7	IC - CMOS	74HC85	1	
8	IC - Regulator	LM7805	1	
9	IC - CMOS	74HC02	1	
10	IC - CMOS	CD4518B	1	
11	IC - CMOS	74HC42	1	
12	IC - CMOS	CD4516B	1	
13	IC - TTL	74LS47	1	
14	Timekeeping	DS1302	1	
15	timer	NE555P	2	
16	timer	NA556N	1	
17	RS232	MAX232CPE+	1	
18	MOSFET - P	IRF9Z24PBF	5	
19	MOSFET- N	IRF630PBF	5	
20	TR - NPN	2N3904BU	3	
21	DIODE	1N4148	4	소신호
22	DIODE	1N5231B	3	제너
23	DIODE	1N4007	4	정류
24	콘덴서	세라믹, 전해 등등	?	
25	저항		?	
26	Crystal	Crystals WE-XTAL 16.0MHz 50ppm	1	
27	부저	KPX1205A	1	
28	LED	10 Segment Light Bar Graph LED Display -Red	1	
29	7 segment	FND507	6	
30	pin header	molex	-	
31	Switches	1825360-3	1	
32	만능기판	-	1	

End of Document