20190612 3일차 수업

타이머 매뉴얼 업데이트는 tcnt에 저장된 값이
내려오는 것을 말함.

타이머 주파수 및 카운터 계산 공식

rTCNTx(카운터) = Hz * 원하는 시간 (sec)
= 1000 * 10
= 10000

Ex)
65535(최대치) = 1,562,500(진동수)(변동가능한수) * 40ms
40msec 범위안에서만 측정가능

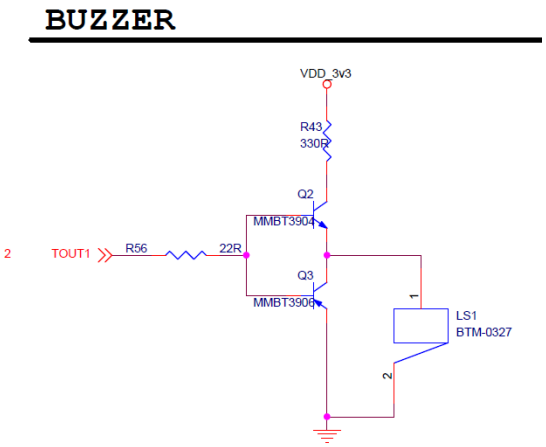
4-bit Divider Settings	Minimum Resolution (prescaler = 0)	Maximum Resolution (prescaler = 255)	Min. Interval (TCNTBn = 1)	Max. Interval (TCNTBn = 65535)
1/2 (PCLK = 50 MHz)	0.0400 us (25.000 MHz)	10.2400 us (97.6562 kHz)	0.0800 us	0.6710 sec
1/4 (PCLK = 50 MHz)	0.0800 us (12.500 MHz)	20.4800 us (48.8281 kHz)	0.1600 us	1.3421 sec
1/8 (PCLK = 50 MHz)	0.1600 us (6.250 MHz)	40.9601 us (24.4140 kHz)	0.3200 us	2.6843 sec
1/16 (PCLK = 50 MHz)	0.3200 us (3.125 MHz)	81.9188 us (12.2070 kHz)	0.6400 us	5.3686 sec

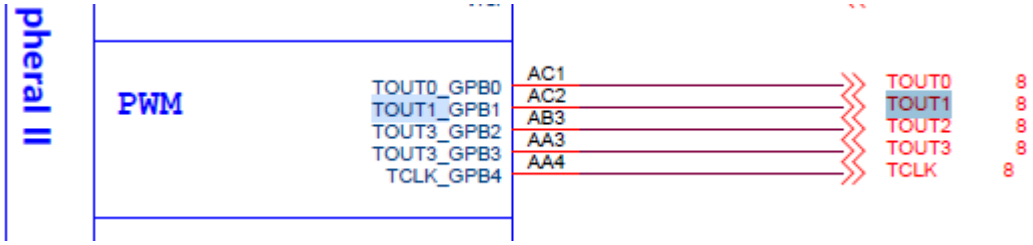
Us는 주기
5.3686초가 시간지연 최대값.

return (0xffff - rTCNTO0); //65535 최대값에 빼면 시간 됨 hz로로나누면 시간이됨 15625 15.625로나누면 msec₩

==========1

부저 buzzur
10hz ~20khz 소리가 다름





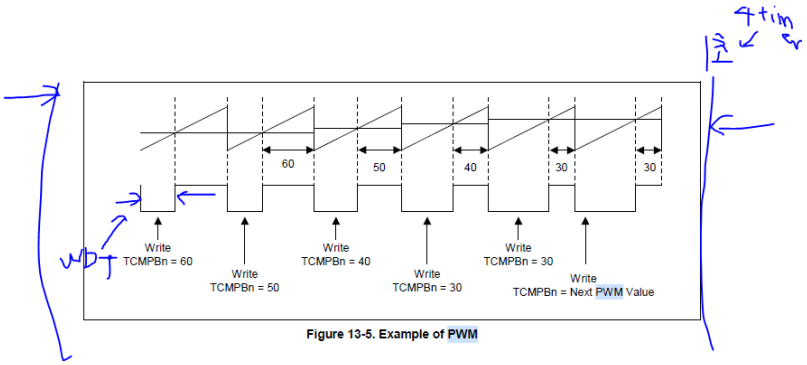


Figure 13-5. Example of PWM

버져는 PWM으로 소리의 주파수를 바꾼게아니고
시간지연으로 주파수를 바꾼것이다.

소리는 duty에 영향이 없어서 그냥 5:5비율만하면됨

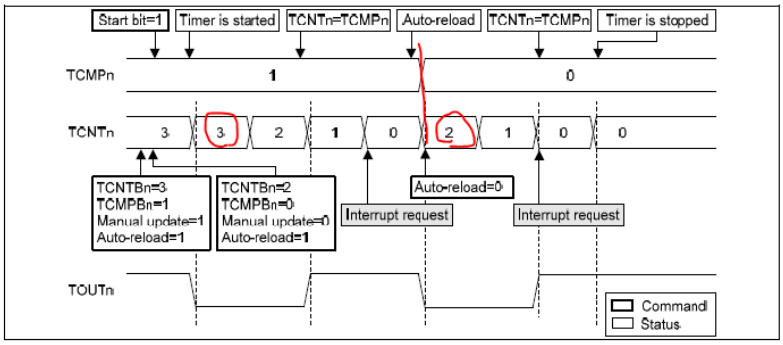
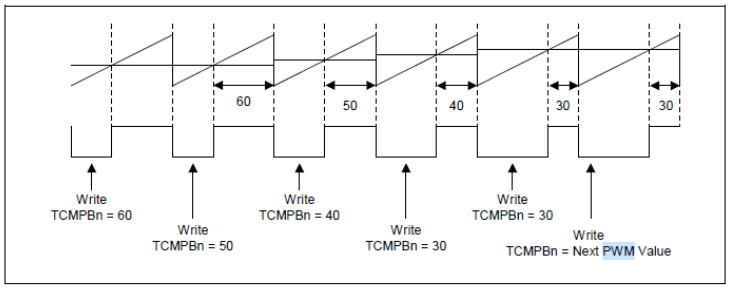


Figure 13-2. Timer Operations

왜 3이아니고 2일까 Tout 0이되고 이후에

타이머 스타트는 카운팅이 독립적으로 하고있음
멈추지않는이상 혼자 계속 카운팅하고잇음.
그래서 manual update 할때 시간이 걸리니까 이후에 2로
그래서 타이머 스타트 이후 중간에 TCNTB0의 값을 바꿀수도있다. (더블버퍼링)



위의 사진을보면 타이머 도중에 값이 바뀌는걸볼수 있다.

부져에 딜레이를 주는코드

```
        while(rTCNTO4 !=0)
        {
                rGPBDAT &= ~(0x1<<1);
                DelayForPlay2(BASE10/tone);
                rGPBDAT |= (0x1<<1);
                DelayForPlay2(BASE10/tone);
        }
        rTCON &= ~(1<<20);
}
```

위의 것을 pwm_timer1_start();

Stop();으로 간단하게 pwm을 활용하여 코드를 바꿀수있다.

데드존은 반전된 신호를 받는 일을 하는데

두개의 신호의 위상차가 발생할수 있는데

그 부분을 데드존이라함

그걸 타이머가 보정해줌.

==========

RTC 시계 + 달력 (타이머는 스탑워치)

레지스터가 많다. 알람기능까지 있어서 또 많다.

RTC 는 BCD 를 사용한다.

16 진수 F 로 저장하는 것 보단 BCD 로 가독성이 뛰어남

하지만 표현하기 위한 비트 수는 더 많이 필요함.

BCDHOUR 데이터시트에 있다.

비트가 BCD 로 표기된다.

ALMHOUR 알람

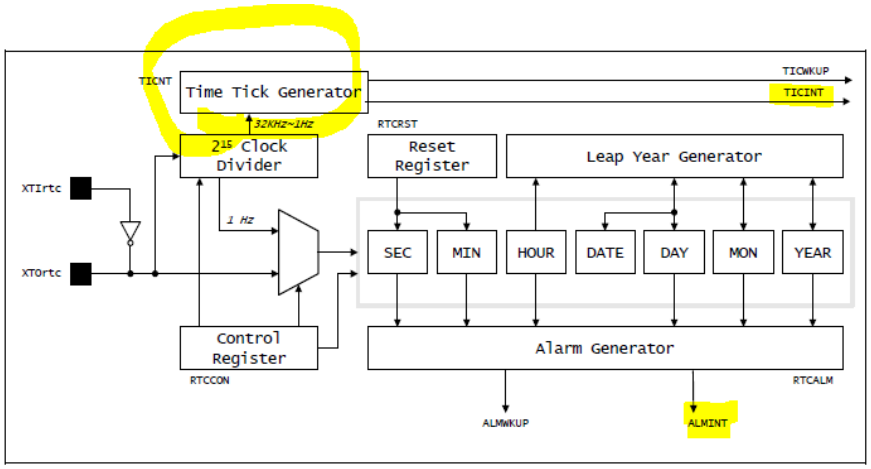


Figure 14-1. Real Time Clock Block Diagram

RTC 는 구현이 너무 간단하고 쉬우니까

혼자 실습해라.

RTC 의 3 가지 용도

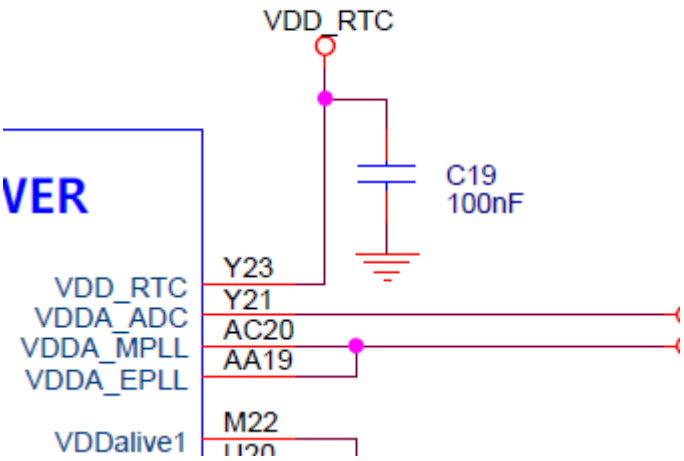
1. 시계
2. 달력(with Y2K)
3. 알람 (with low power)

   RTC 가 자고있는 CPU 를 깨워준다. (ALMINT)

전력 손실을 줄일수 있다.

RTC에 붙는 크리스탈의 정확도가 중요함 시간의 정확성에 관련됨

정확히 1hz를 보내줘야됨



RTC는 독립적인 전원을 사용함

==========

WDT 시스템 감시 타이머

오동작감지

시스템을 껏다 킴

사용자에게 알림

WTDAT= = TCNTx

WTCNT(=TCNTOx) => 타이머 카운터 H/W

Wtcnt와 tcnto는 비슷하게도 읽어오는것인데 wtcnt는 쓰기도 가능함

WDT_TEST

```
WDT_Init(3); // limitation (1sec ~ 32sec)
WDT_Start(1);
for(;;); // endless loop
```

3초 후 리셋 함

WDT_Init();

WDT_Start(1);

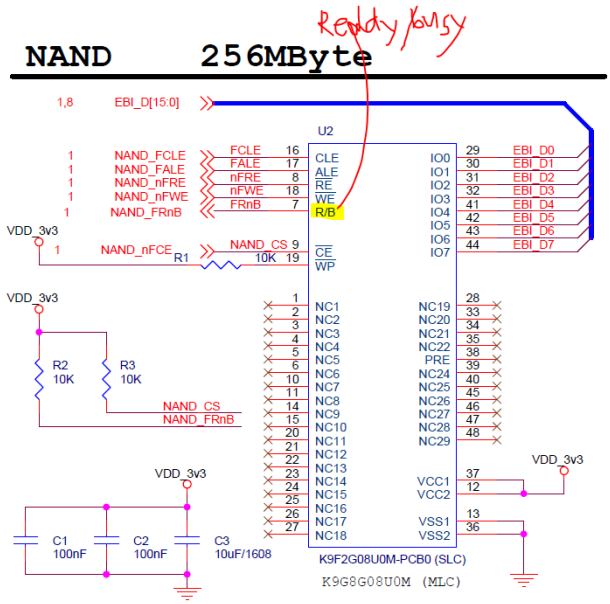
For(;;){};

어떤 반복문의 실행 시간을 구하고

시간측정후 그 시간이 초과되고 안되면

리셋하게끔 활용하면된다.

낸드플레쉬



비지 레디

최소 2k가 들어옴

그걸 번함 태움

태울때는 비지 R/B

다태우면 레디

Cpu가 마이크로 단위의 초단위로 번함

레디비지가 최소 바뀌는 시간이 넘어가면 다운되는데

그걸 WDT로 방지 하게끔할수있다.

디버깅에도 좋고 활용하면 참으로 좋다.

==========

2

P44 startup

Main 함수보다 먼저 실행되는게

startup코드임

부트로더와 startup과 전혀 다른 맥락임

부트코드는 다른 소스들과 얽혀 만들어짐

==========

메모리 시스템

Soc에는 여러 장치들이 있다.

메모리 제어 장치(메모리 컨트롤러)도 있다. 매우 어려움

외부에 여러종류의 메모리를 단다.

플레쉬 메모리

1 노어 플래시 (프로그램 저장용) intel flash

2 낸드 플래시 (프로그램과 일반데이타 저장용) -SSD

Rom은 리드온리

Otprom / eprom / eeprom /

IROM(내장메모리)은 점프스위치를 통해서 낸드플래시를 대신해 sd카드로 부팅할수있다.

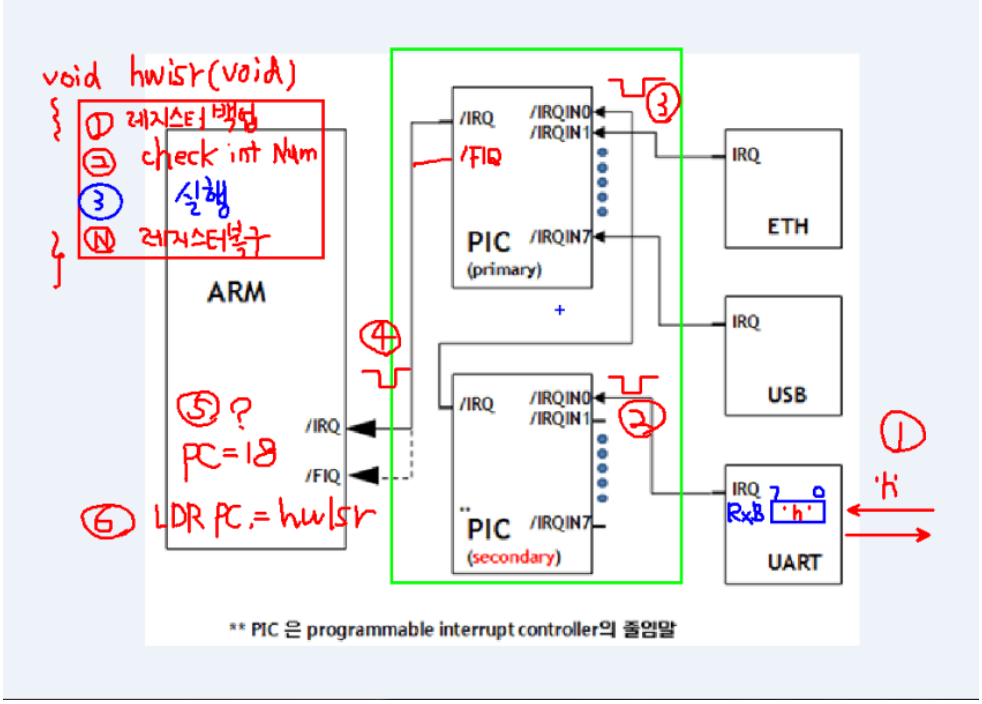
==========

4장 인터럽트

폴링 – 입력값이 들어오는지 안오는지 체크하기위해 대기하는 것.

전력소모가 많아진다.

마이크로프로세서에서 **인터럽트(interrupt**, 문화어: 중단, 새치기)란 마이크로프로세서(CPU)가 프로그램을 실행하고 있을 때, 입출력 하드웨어 등의 장치나 또는 예외상황이 발생하여 처리가 필요할 경우에 마이크로프로세서에게 알려 처리할 수 있도록 하는 것을 말한다.



인터럽트 제어장치

인터럽트 컨트롤러

인터럽트 멀티플렉서

ETH USB UART 신호들을 PIC에 묶어서 정리해서 ARM에 연결한다.

바로 다이렉트로 ARM에 연결할순 없다.

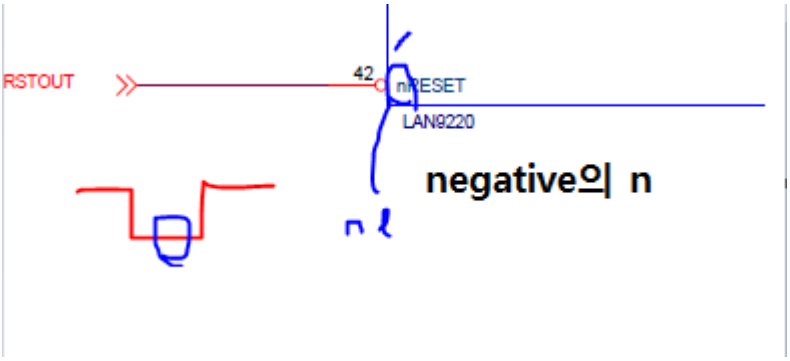
1대 N관계로 해주는게 인터럽트 컨트롤러

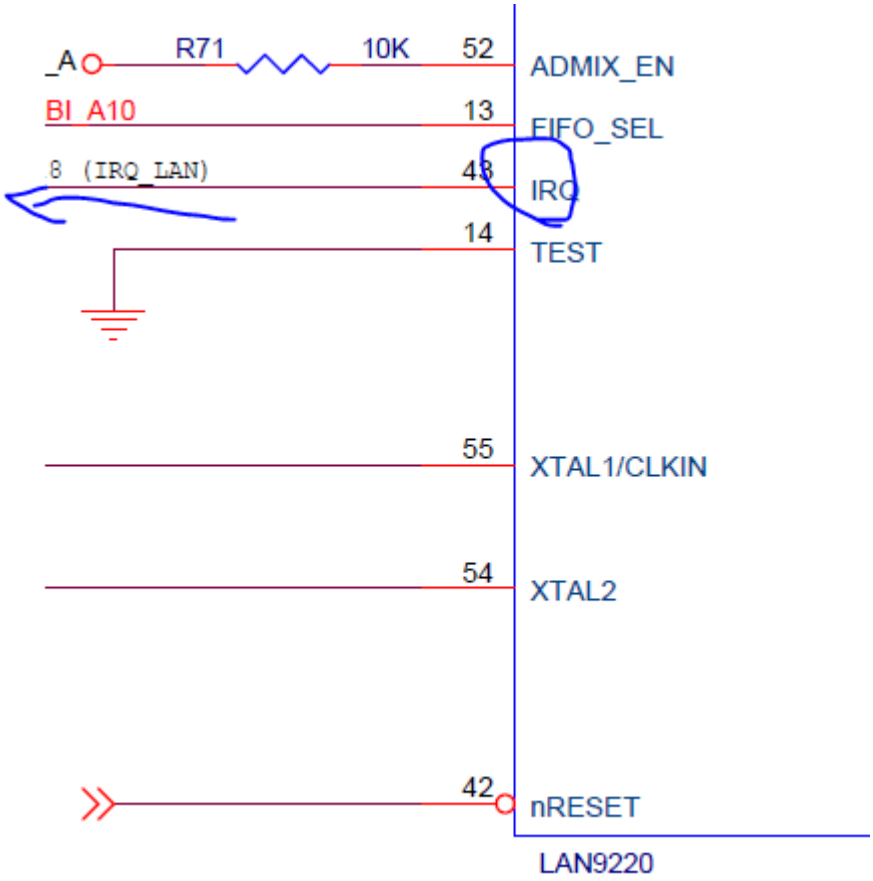
인터럽트는 확장이 가능함

인터럽트들 간에 주종관계가 된다.

'h'라는 글자가 UART RX 버퍼에 들어오면 인터럽트가 발생함.

이후의 1번부터 순차적인 순서들은 물리적인 전달이다.





IRQ 네트워크칩에서 출력이 나감 positive 신호 (high)가 인터럽트를 건다.

만약 신호받는곳이 negative를받는다면 낫을 써서 반대로 인터럽트 넣어줘야됨

Interrupt Request	IRQ	VO8/ VOD8	1	Programmable Interrupt request. Programmable polarity, source and buffer types.

네트워크 칩에서 IRQ부분 설정인데 인터럽트를 positive로할지 negative로할지 설정할수있다.

2	**PME Polarity (PME_POL).** This bit controls the polarity of the PME signal. When set, the PME output is an active high signal. When reset, it is active low. When PME is configured as an open-drain output this field is ignored, and the output is always active low.	R/W NASR	0b

개념도로 돌아와서

5번 프로그램카운터 를 통해서 함수로 이동

6번 을 통해서 프로그램이 작동됨

인터럽트가 어디서 들어온지 알수 있을까?

인터럽트 상태 레지스터(primary) 비트를 통해서 인터럽트를 인지한다. (primary;비서가)ARM에게 알려줌

인터럽트 상태 2레지스터 (secondary;사무처장)

Cpu가 다 실행하고나서 레지스터를 복구하는데

그사이에 인터럽트 상태레지스터 클리어를 해줘야한다.

왜나면 인터럽트 컨트롤러들에게 비트를 지워야 인터럽트가 소멸되게함
클리어하지않으면 계속 같은 인터럽트가 발생함.
그 소멸될때도 순서가있다.
먼저 IRQ UART 에서 먼저 인터럽트를 정리해줘야함.
(인터럽트 발생 시작부분부터 정리해줘야 인터럽트가 시작되지않는다.)
그리고 인터럽트 컨트롤러 정리한다.

인터럽트 마스크(킹)
usb인터럽트가 와도 무시하고싶을 때
마스크 레지스터에 지우는 것을 마스킹이라함.

인터럽트 우선순위(정적/동적)
인터럽트가 동시에 발생햇을 때의 우선순위를 말한다.
정적은 이미 번호로 우선순위가 정해져있는것임
동적은 마음대로 변경가능한 것.

긴급은 FIQ
IRQ보다 우선순위가 높다.

이러한 인터럽트의 과정들을 간략하게 보자면
마스터 핸들러가 서브핸들러들을 불러들이는 것

INTOFFSET1 primary

INTOFFSET2 secondary이다.
인터럽트 클리어는 주의하자!
주의!! 클리어라고 0으로 비트를 만드는것이아니라 (&~) 이아닌 셋임 set을 해당 비트에 해주면 해당되는 것을 클리어해줌 즉, 클리어 비트셋이라보면됨
비트셋으로 클리어해야됨
그렇기 때문에 |= 연산을 쓰면 셋이아닌 클리어기 때문에 덮어서 지워짐.

```
/* TODO : Pending Register Clear */
        rSRCPND1 &= ~(1<<10);
        rINTPND1 &= ~(1<<10);
```

```
/* TODO : Pending Register Clear */
        rSRCPND1 = (1<<10);
        rINTPND1 = (1<<10);
```

인터럽트 상태를 클리어해야 끝남

지우면 계속 인터럽트 걸림

이클리어는 셋이아님 그부분을 클리어한다는것임.특수한 예외적인 부분임 주의!

```
rINTPND1 |= (1<<10);
```

이렇게 코딩하면 다른곳에서 인터럽트들어온곳이 이 연산으로 덮어써져서 다른 비트에도 영향을 준다.

인터럽트 신호 전달체계

ARM < --- rINTPND1 ←---rINTMSK1 ←- rSRCPND1 <==-주변장치

(마스킹후) (마스킹) (마스킹하기전)

1번과 7번이 둘다 마스킹이 없으면

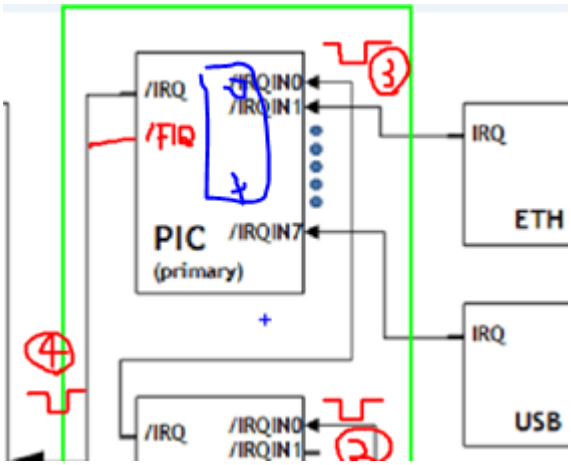
동시에 들어오면 우선순위가높은 1번만 들어온다.

그래서 마스킹후면 1번만 체킹되어있음

그리고 1번처리되면 바로 7번비트가 마스킹후 들어옴.

**회장님이 원하는 우선순위를 정해서 실행할수있음.

그때는 마스킹전에 들어가서 실행하는데 마스킹되어있는지 체크하고 실행함.



0번이랑 7번이 인터럽트 들어옴

rINTMSK1 = rINTMSK1 &~(1<<1); // ETHER (IRQ Active)

rSRCPND1 ? 0b10000010

rINTPND1 ? 0b00000010

//rINTPND1 ? 0b00100000 (rINTOFFSET=5)

// rINTOFFSET 을통해서 몇변의 입터럽트인지 번호로 확인하기 편하다 바로 읽어낼수잇음

rSRCPND1 = (1<<3); // (0b10000010 -> 0b1000X000)

rSRCPND1 = (1<<1); //먼저 밖에서부터 멈춰야됨. (0b10000010 -> 0b10000000)

rINTPND1 = (1<<1); //인터럽트 클리어 (0b00000010 -> 0b00000000)

==========

초보 엔지니어용 보드 혹은 ARM 프로세서 =>STmicro & ATMEL

상급자용 => TI (Texas Instrument)

Broadcom ;;; 비추 (라즈베리파이)

NVDIA tx1, tx2

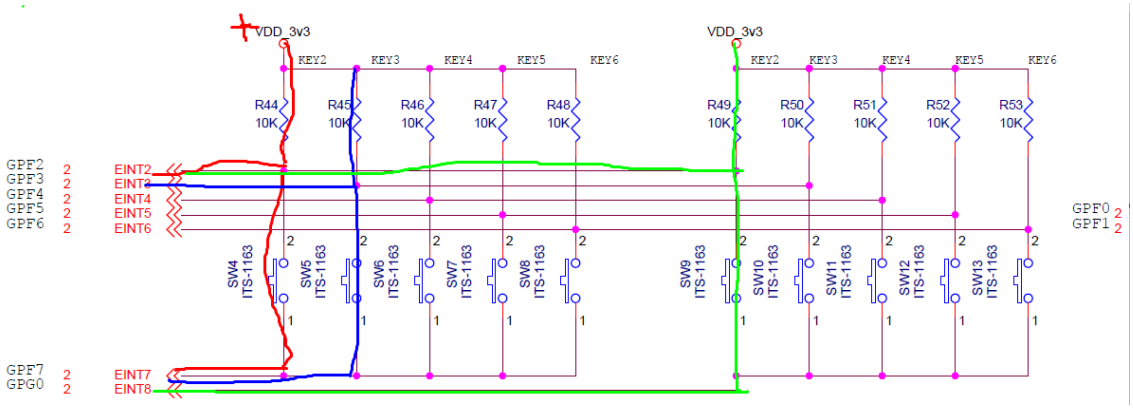
==========

KEY 인터럽트로 구현하기

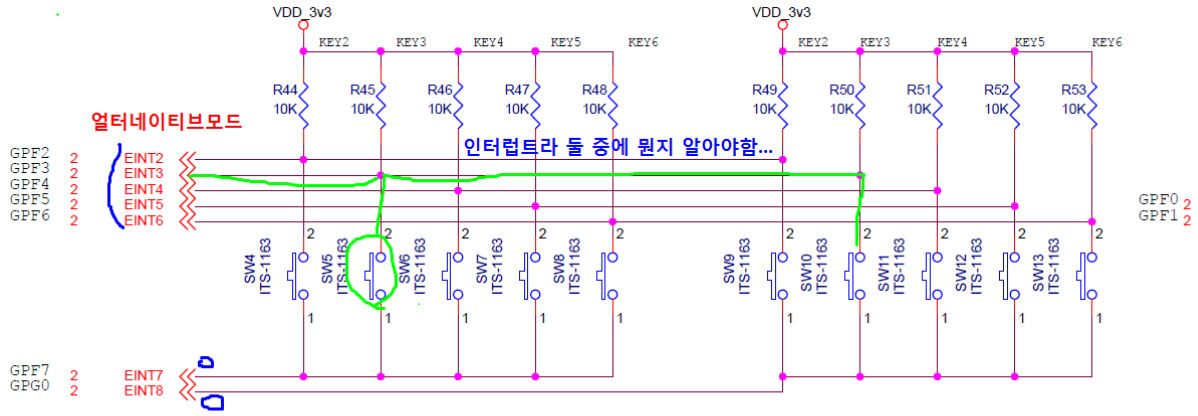
```
void Key_INT_Init(void)
{
  /* Set External Interrupt Vector */
  HandleEINT2 = (unsigned int)Key_ISR;
  HandleEINT3 = (unsigned int)Key_ISR;
  HandleEINT4_7 = (unsigned int)Key_ISR;
```

인터럽트를 3개를 쓴다...

폴링 방식은 이렇게했엇찌.



인터럽트방식은 이렇게.



인터럽트를 순간 무슨키인지 확인하기위해서는

순간적으로 키가 눌릴 때 인터럽트 걸릴 때 다시 폴링으로 바꾸고

무슨 키인지 확인하고 잡은후 처리하고 다시 나올 때 폴링을 인터럽트로 바꿔놓고 나온다.

INT_TIMER0	[10]	0 = Not requested, 1 = Requested	0
INT_WDT/AC97	[9]	0 = Not requested, 1 = Requested	0
INT_TICK	[8]	0 = Not requested, 1 = Requested	0
nBATT_FLT	[7]	0 = Not requested, 1 = Requested	0
INT_CAM	[6]	0 = Not requested, 1 = Requested	0
EINT8_23	[5]	0 = Not requested, 1 = Requested	0
EINT4_7	[4]	0 = Not requested, 1 = Requested	0
EINT3	[3]	0 = Not requested, 1 = Requested	0
EINT2	[2]	0 = Not requested, 1 = Requested	0
EINT1	[1]	0 = Not requested, 1 = Requested	0
EINT0	[0]	0 = Not requested, 1 = Requested	0

필기 참고.

EINT4-7은 구분하기위해따로 만들어진곳이있다 그곳이

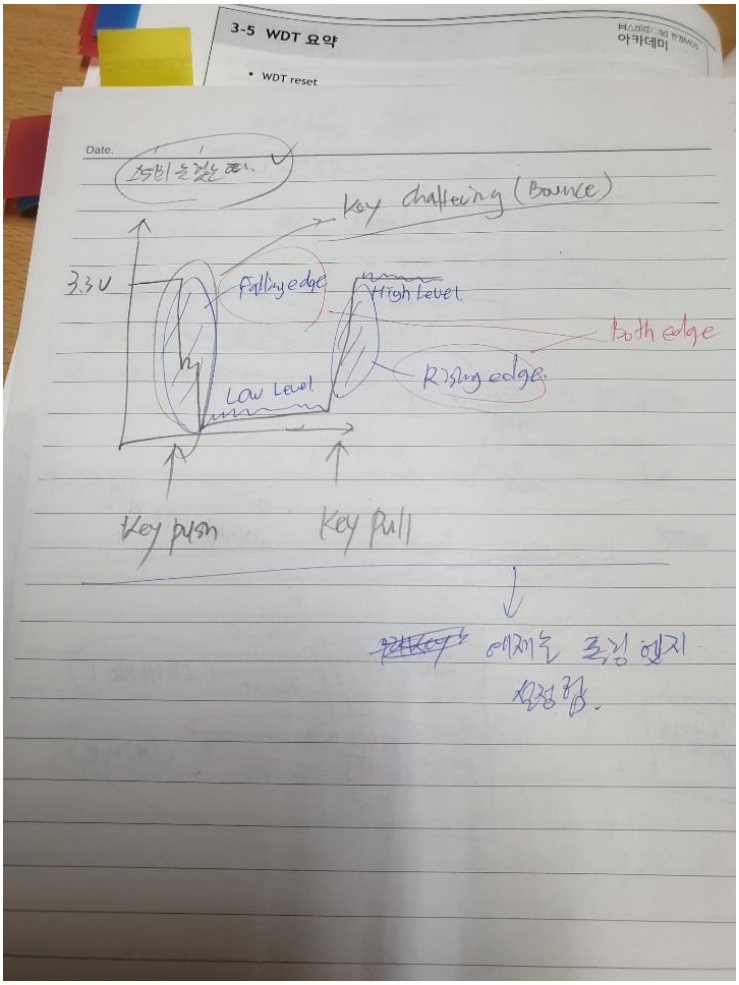
rEXTINT0 이다.

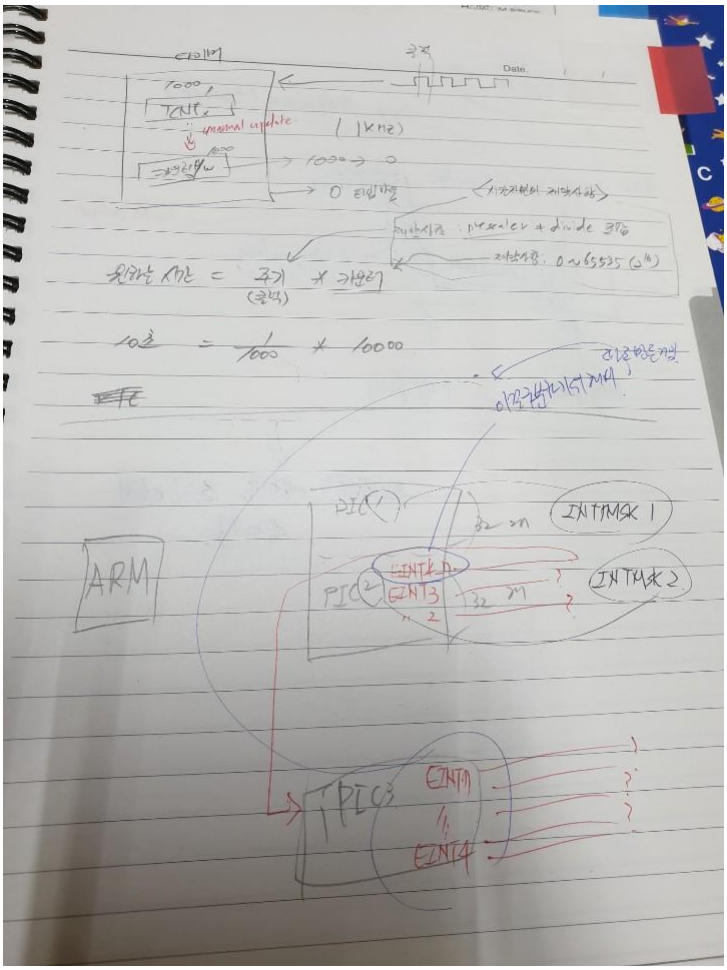
그리고 셋팅이 폴링엣지로 설정되어있다. 필기참고

```
void Key_ISR(void)
{
  int temp_GPFCON, input_value;

  /* GPFCON을 input mode로 변경시키기 위해 interrupt mode의 값을 저장 */
  temp_GPFCON = rGPFCON;
  /* GPFCON을 input mode로 변경 */
  rGPFCON &= ~(0x3ff<<4);
  rGPFUDP &= ~(0x1f<<2);
  /* input 모드 상황에서 눌려진 키값을 확인 */
  while (! (input_value= Key_Get_Pressed()));
  Uart_Printf("Key Value =%d\n",  input_value);
  /* 다시 interrupt 모드로 변경 */
  rGPFCON = temp_GPFCON;
  /* Keyout 값을 0으로 설정해놓고 ISR을 빠져 나옴 */
  rGPGDAT &= ~0x1;
  rGPFDAT &= ~(0x1<<7);

  /* clear global interrupt pending */
  rEINTPEND = (0x7<<4);
  rSRCPND1 = (0x7<<2);
  rINTPND1 = (0x7<<2);
}
```

==========

필기


3-5 WDT 요약
• WDT reset
Date.
Key chattering (Bounce)
3.3V
Falling edge
High Level
Both edge
Rising edge
Low Level
Key push
Key pull


Date.
ARM
PIC
INTMSK 1
INTMSK 2