실험 9. Data EEPROM 메모리와 watchdog 사용하기

1. EEPROM 메모리

일반적인 RAM 메모리는 쓰고 읽기를 마음대로 할 수 있는 기억소자이다. 우리가 register file로 명명한 기억소자가 바로 RAM이며, 이 영역을 이용하여 계산된 결과를 저장하거나 저장된 내용을 불러와서 사용하였다. 그러나 RAM은 전원이 꺼지면 기억된 내용이 전부 지워지므로(지워진다고 하여 꼭 '00'이 된다는 의미는 아니며, 메모리의 내부회로에 따라서 값이 결정된다.) 시스템의 설정조건이나 사용자 계수 등은 전원이 꺼져도 보관될 수 있는 별도의 기억장치가 필요하다. 일반적으로 메모리에 기억된 내용이 지워지지 않도록 된 기억소자를 ROM(read only memory)라고 하며, 결과적으로 ROM에 저장되어야 한다는 의미이다. 그러나 순수한 의미의 ROM은 제조 시부터 내용을 고정시킨 masked ROM과, ROM을 만들고 나서 내용을 별도로 기억시키는 PROM(programmable ROM)으로 나누어진다. 다시 PROM은 OTP(one time program)과 여러 번 내용을 저장할 수 있는 EPROM(erasable PROM)으로 나누어진다. EPROM은 지우는 방식에 따라서 일반적인 EPROM과 EEPROM(Electrical Programmable ROM)으로 나누어진다. 일반적인 ERPOM은 IC 위에 창을 가지고 있고 창을 통하여 적외선을 비추어 줌으로써 기억된 내용이 지워지게되는 소자이며, 상품명은 27XXXX 등의 이름으로 불린다. 예로 27256은 256Kbit 용량을 가지는 EPROM이다. 그리고 전기적으로 기억된 내용을 지울 수 있는 소자를 EEPROM이라고 한다. 즉 기억된 내용을 전기적으로 지우고 바로 쓸 수 있다는 것이며, 따라서 EPROM보다는 사용하기가 훨씬 편리하다. 반면 flash 메모리 (때로 'flash RAM'으로도 불린다)는 기억된 내용이 지워지지 않은 비휘발성 메모리로서 블록단위로 내용을 지울 수도 있고, 다시 프로그램 할 수도 있다. 플래시메모리는 EEPROM의 변형 중 하나인데, byte 단위로 지울 수도 있고 수정할 수도 있는 EEPROM과는 달리 블록단위로 수정되기 때문에 속도가 빠르다. 플래시메모리는 종종 PC의 BIOS와 같은 제어코드를 저장하는데 사용된다. BIOS를 수정해야할 필요가 있을 때, 플래시메모리는 바이트 단위가 아닌 블록 단위로 기록됨으로써 수정이 쉽다. 한편, 플래시메모리는 일반 RAM처럼 유용하지는 못한 이유는, RAM은 블록이

아닌 바이트 단위의 주소지정이 가능해야하기 때문이다.

플래시메모리라는 이름은 메모리 cell이 섬광(flash)처럼 단 한번의 동작으로 지워질 수 있도록 구성되었기 때문이며, Fowler-Nordheim 터널효과에 의해서 전자들이 얇은 유전체(誘電體) 물질을 관통하여 각 메모리 cell과 결합되어 있는 floating gate로부터 전하를 제거한다. Flash RAM은 program을 저장하는 메모리로 사용되고 있으며, 별도의 writer가 필요하다.

실제로 여러 응용에서 사용자가 설정한 값이나, 계수 등은 데이터로 RAM에 저장되며, 따라서 전원이 차단되면 지워지게된다. 이를 위하여 16F84에는 데이터를 영구적으로 보관할 수 있는 별도의 기억장치를 가지고 있으며, 이것이 데이터 EEPROM이다. EEPROM 데이터 메모리는 byte 단위로 read/write할 수 있으며, write하면 자동으로 내용을 지운 후에 새로운 내용이 기억된다. 그리고 write하는데 소요되는 시간은 약 10msec이므로, 한번 write한 후에는 이 시간 동안 기다려야 한다. 데이터 EEPROM 영역은 독립된 메모리 공간으로 존재하며, 따라서 접근하는 방법도 특수기능 register를 통하여 이루어지며. 4개의 register가 사용된다.

(1) EEDATA register는 read/write 할 8bit 데이터를 저장하는 register이고,
(2) EEADR register는 access할 EEPROM의 주소를 저장하는 register이며, 64개의 용량을 가지므로 00H~3FH가 된다.
(3) EECON1 register는 EEPROM 제어용으로 사용되며, 기능은 아래와 같다.

EECON1 REGISTER (ADDRESS 88h)

U-0	U-0	U-0	R/W-0	R/W-x	R/W-0	R/S-0	R/S-0
—	—	—	EEIF	WRERR	WREN	WR	RD
bit 7							bit 0

bit 7-5 **Unimplemented:** Read as '0'

bit 4 **EEIF:** EEPROM Write Operation Interrupt Flag bit
1 = The write operation completed (must be cleared in software)
0 = The write operation is not complete or has not been started

bit 3 **WRERR:** EEPROM Error Flag bit
1 = A write operation is prematurely terminated
(any $\overline{\text{MCLR}}$ Reset or any WDT Reset during normal operation)
0 = The write operation completed

bit 2 **WREN:** EEPROM Write Enable bit
1 = Allows write cycles
0 = Inhibits write to the EEPROM

bit 1 **WR:** Write Control bit
1 = Initiates a write cycle. The bit is cleared by hardware once write is complete. The WR bit can only be set (not cleared) in software.
0 = Write cycle to the EEPROM is complete

bit 0 **RD:** Read Control bit
1 = Initiates an EEPROM read RD is cleared in hardware. The RD bit can only be set (not cleared) in software.
0 = Does not initiate an EEPROM read

(4) EECON2 register는 데이터 write시 제어 코드를 넣어 주는 용도로 사용된다. 즉 write bit를 set하기 전에 이 register를 사용하여 미리 55H와 AAH를 write해야 한다.

이런 방법을 사용하는 것은 단순하게 생각하면 필요없이 복잡하게 만드는 것으로 보이나, EEPROM의 데이터가 전원 ON/OFF시나 다른 여러 가지 잡음신호로부터 변경되는 것을 막아주는 용도로 잡음으로 만들어질 수 없는 상반된 두 코드 55H와 AAH를 확인하여 동작되도록 한 것입니다.

☞ EEPROM 데이터 read 순서
① EEADR에 주소 저장
② EECON1의 bit0(read bit)=1 한 후
③ EEDATA를 읽으면 된다.

예1) 00번지 EEPROM 내용 읽어서 W에 넣기

```
BCF     STATUS,RP0      ; bank 0
MOVLW   00H
MOVWF   EEADR           ; 주소 설정
BSF     STATUS,RP0      ; BANK 1
BSF     EECON1,RD
BCF     STATUS,RP0      ; bank 0
MOVF    EEDATA,W        ; W=EEDATA
```

☞ EEPROM에 데이터를 write 하는 순서
① EEADR에 주소 저장
② EEDATA에 데이터 기록
③ EECON1의 bit2(write enable bit)=1 하고
④ EECON2에 55H,0AAH를 순서대로 기록한다. 그리고
⑤ EECON1의 bit1(write bit)=1 하면 write 가 시작된다. 그리고
⑥ Write가 완전히 되면, EECON1의 bit4(EEIF)가 1로 set된다.
⑦ 따라서 사용자는 EEIF bit가 1로 set되었는가를 확인하여 write를 종료한다.
⑧ Write가 끝나면, 반듯이 EEIF, WREN bit를 0으로 만들어 주어야 한다.

예2) 00번지 EEPROM에 41H 기억시키기

```
BCF     STATUS,RP0      ; bank 0
MOVLW   00H
MOVWF   EEADR           ; 주소 설정
MOVLW   41H
MOVWF   EEDATA          ; 기억할 데이터 이동
BSF     STATUS,RP0      ; BANK 1
BSF     EECON1,WREN
MOVLW   055H
MOVWF   EECON2          ; 제어코드 출력
MOVLW   0AAH
MOVWF   EECON2          ; 제어코드 출력
```

```
        BSF     EECON1,WR       ; write 시작
LP7     BTFSS   EECON1,EEIF     ; 써넣기 완료 확인
        GOTO    LP7
        BCF     EECON1,EEIF
        BCF     EECON1,WREN
        BCF     STATUS,RP0      ; bank 0
```

2. Watchdog timer 동작시키기

프로그램이 복잡해지고 여러 가지 기능을 구현 하다가 보면, 미처 고려되지 않은 상황이 발생할 수도 있고 잡음에 의해서 오동작 할 수도 있다. 따라서 이러한 상태가 나타나면 자동적으로 멈추거나 스스로 초기화가 되어 다시 시작하여야 할 것이다. 아직도 프로세서가 멈추어 아무 일도 하지 않은 상태로 있을 수 있다고 생각하는가? 아주 특별한 명령어(SLEEP 명령어: 주 참조)가 들어오지 않는 한 cpu는 계속적으로 동작을 한다. 따라서 원하지 않는 엉뚱한 일을 할 수도 있다. 그러므로 cpu가 원하는 프로그램을 충실히 수행하고 있는가를 지속적으로 감시하고 있다가 문제가 발생하면 cpu를 초기화시켜 지속적으로 엉뚱한 일을 하지 못하도록 막아야 한다. 이러한 기능을 watchdog 이라고 한다. watchdog를 실현하는 방법은 프로그램이 정상적으로 돌아가고 있을 때 반복적으로 수행되어야하는 부분에 특정펄스를 출력하도록 하고, 우리는 이 펄스의 주기가 변하거나 신호 자체가 나오지 않으면 프로그램이 정상적으로 동작하고 있지 않다고 판단 할 수가 있다. 이러한 기능을 일반화시키려면 펄스의 개수를 계수할 수 있는 timer가 필요하다. 즉 특정시간 안에 펄스가 나오지 않으면 cpu를 reset시키면 된다. 실제의 회로에서는 시간 조절을 프로그램으로 할 수 있도록 펄스를 출력하는 것이 아니고, 프로그램에서는 timer를 clear시키는 신호를 내보낸다. 이렇게 함으로서 정해진 시간이 지나면 time-out에 의한 cpu의 reset를 억제한다. 이 기능에 해당하는 명령어가 CLRWDT이다. 즉 cpu clock과는 별도로 RC 발진회로를 구성하고 이것을 watchdog timer에 의해서 분주되어 18msec의 시간간격으로 출력을 발생시킨다. 이것을 기본 출력으로 하여 원하는 time-out 시간을 얻기 위해 prescaler를 사용한다. 이때 사용되는 prescaler는 timer0에서 사용되는 prescaler와 같은 것으로 설정은 앞에서 설명한 OPTIONS register에 의해서 이루어진다. 최

대 time-out 시간은 128*18msec로 약 2.3초가 된다. (Timer0에서 prescaler를 사용하면, watchdog timer에서는 prescaler를 사용할 수 없다. 반대도 마찬가지이다. 그리고 PIC16F84 내부에 이 회로가 구현되어 있으며, RC회로를 사용하여 기본 시간을 만들므로 정확한 시간은 아니다.) 그리고 watchdog 기능을 사용하려면, chip에 프로그램을 기록시 watchdog bit를 enable로 만들어 주어야 한다. Emulator 상에서는 setup 기능에서 enable로 선택해야한다.(평상시는 WDT는 disable로 되어 있음)

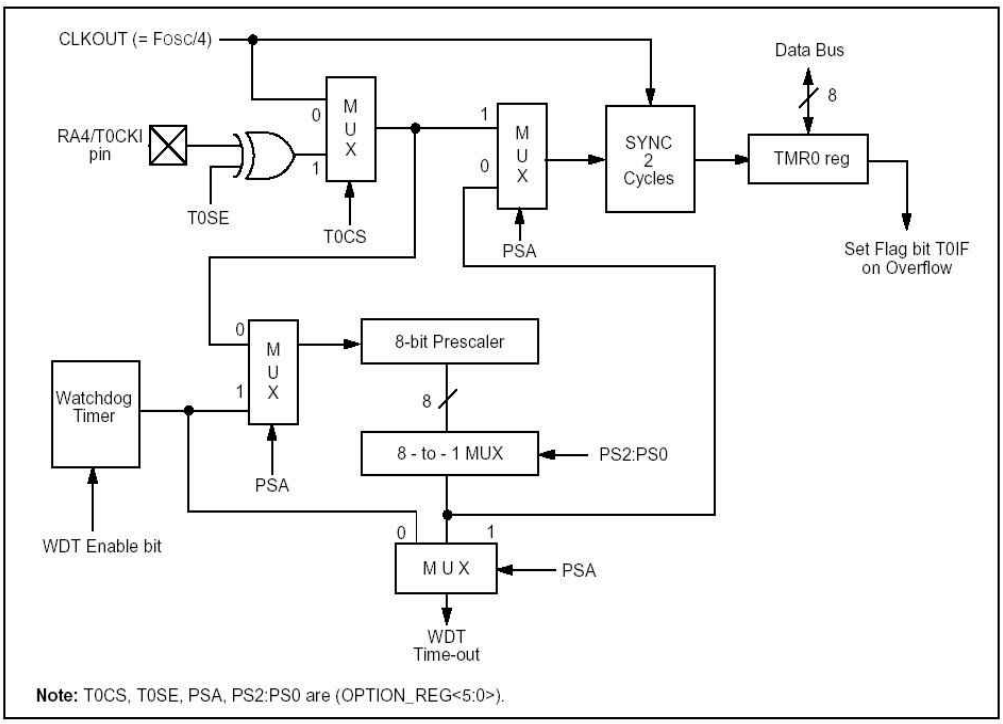

그림 E9-1. Watchdog timer와 timer 0 block diagram

1) SLEEP 명령어

마이크로프로세서가 들어 있는 시스템에서 전원(통상적으로 건전지 사용하는 곳)을 절약하기 위하여 마이크로프로세서가 사용되고 있지 않는 동안에는 cpu의 동작을 멈추고 I/O 단자를 끊어둔 상태(high output impedance 상태)

로 만드는 기능이다. 이러한 기능을 사용하는 예로 대표적인 것이 TV 비디오 등을 조작하는 리모컨일 것이다. 리모컨에 넣는 작은 건전지 하나로 1년 이상의 동작을 한다. 즉 리모컨 안에 있는 cpu는 평상시는 동작되지 않고 있다가 여러분이 key를 누른 순간만 살아나서 동작하고 다시 잠자고 있는 상태가 된다. 전지의 수명을 늘이기 위한 가장 단순한 방법은 사용자가 전원을 on하고 key를 누르면 되나, 사용상 불편하고 실수로 사용 후 전원을 끄지 않으면 계속 동작하여 전지의 수명이 몇 일 가지 못하게 될 것이다. 따라서 이러한 기능을 가능하게 만든 것이 마이크로프로세서의 SLEEP 명령어이다. 마이크로프로세서가 SLEEP 상태에서 깨어나도록 하는 방법은 여러 가지가 있으며, 기본 기능이 interrupt를 이용하는 것이다. 또 watchdog timer에 의해서도 wake-up이 가능하다.

☞ 모든 CMOS 논리회로는 이상적으로는 스위칭 시에만 전류를 소모하므로 발진 주파수를 낮추면 전류 소모가 급격히 줄어든다. 그리고 SLEEP 상태에서는 clock이 정지하므로 소모되는 전류는 2V 전원에서 1uA보다 적게된다. 자세한 것은 data sheet를 참조하기 바란다.

2) CLRWDT 명령어

Watchdog timer와 prescaler(watchdog 용으로 사용시에만)를 clear시키는 명령어로, 하드웨어적으로 특정시간이 지나면 time-out 신호가 발생되도록 만들어진watchdog 기능에서 counter를 clear시켜 cpu를 reset시키는 time-out 신호의 발생을 억제한다. 따라서 CLRWDT 명령어는 설정된 watchdog time 보다 빠른 주기로 반복 수행되어야 cpu가 정상적으로 동작한다.

예 3) 앞 시계 프로그램 예에서 WDT를 사용하기

시계 프로그램에서 timer0과 prescaler를 사용하므로 WDT를 사용하려면 기본시간 18msec 만 가능하다. 그리고 ISR에 약 2msec 주기로 들어오므로 ISR 안에 CLRWDT를 삽입하고 시스템 설정을 watchdog enable로 하면 된다. 즉 WDT가 18msec로 되어있으므로 18msec 안에 WDT를 clear하여야 한다. 그런데 여기서는 약 2msec 단위로 반복 clear 하므로 time-out는 일어나지 않는다. 따라서 외형적으로는 watchdog의 기능이 느껴지지 않는 것이 원칙이다. 이것을 테스트 해보려면 ISR에 있는 CLRWDT 명령어를 죽여보면 cpu는 계속 reset될 것이다.

예 4) Watchdog를 확인하기 위한 간단한 프로그램 구현

프로그램이 초기화되면 초기화된 횟수를 세는 프로그램을 만들면 watchdog의 기능을 가장 쉽게 확인 가능하다. 횟수를 숫자로 나타내려면 복잡해지므로 LED에 출력하도록 하자. 이때도 18msec로 초기화가 되면 너무 빠른 속도로 LED가 on/off하면 확인이 어려우므로 이를 64로 나누어 약 1sec 주기로 불이 오도록 하자.

```
; 프로그램 초기 부분
      .
; 계수 변수 WDT_CNT 선언
START_UP
            BSF       STATUS,RP0  ; RAM BANK 1 선택
            MOVLW     B'????????'   ; PORT I/O 선택
            MOVWF     TRISA
            MOVLW     B'????????'   ; PORT I/O 선택
            MOVWF     TRISB
; WATCHDOG 시간 설정     --- 18mSEC 주기
            MOVLW     B'00000010'   ;
            MOVWF     OPTIONR
            BCF       STATUS,RP0  ; RAM BANK 0 선택
; MAIN PROGRAM
            INCF      WDT_CNT,F
            MOVF      WDT_CNT,W
            MOVWF     PORTB

      LLLP  NOP
            GOTO      LLLP
            END
```

이 프로그램은 수행되면 LLLP LOOP에서 무한 반복하므로 WDT_CNT의 내용이 변하지 않을 것이고, 따라서 PORTB의 내용도 일정할 것이다. 그러나 watchdog 기능을 동작시키면 18msec 안에 watchdog timer를 clear시키는 기능이 동작되지 못하므로 time-out에 의해서 cpu는 reset되게되어 0000번지부

터 다시 프로그램을 수행하게되므로 WDT_CNT가 증가될 것이다. 우리는 증가되는 상황을 PORTB를 통하여 보고자하는 것이며, PORTB의 BIT6에 LED를 연결하면 64로 나눈 결과가 나오게된다.

■ 예비문제

1) SLEEP 기능이 필요한 이유와 사용되고 있는 구체적인 예를 주변 장치에서 찾아보시오.
2) 디지털 데이터의 저장이나 전송 시 발생되는 신호의 오류를 확인하기 위하여 사용하는 방법들이 무엇인가 찾아봅시다. (예: parity bit, crc check 등)
3) 실험장치의 emulation 환경에서 watchdog 기능을 설정하는 방법을 찾아봅시다.

■ 실 험

1) 예 1,2)를 이용하여 EEPROM에 자신의 주소 값보다 1이 큰 값을 기억시킨 후 읽어서 확인하는 프로그램을 작성해 봅시다.

☞ EEPROM에 기록하고 바로 읽어보는 것이 아니고, 전부 기록하고 나서 전원을 끄고 RAM에 저장된 내용이 지워지도록 한 후 전원을 다시 on하여 EEPROM을 읽어 과거에 기록한 내용이 지워졌는가를 확인하라는 말이다. 00H부터 3FH 번지까지를 기록하기 위해서는 loop을 이용하면 되고, 읽어서 기록된 내용과 비교하려면 EEPROM 내용을 읽어서 RAM에 순서대로 넣어야 한다. 이런 용도 사용된 명령어가 간접 주소 지정 방법이고 pointer라고 앞에서 설명한 것이므로 적절히 활용하길 바란다.

2) 예 4)를 작성하고 동작시켜 봅시다. 그리고 RB6에 LED를 연결하여 on/off 상태를 확인합시다. LED는 on이나 off 상태로 고정되어 있게될 것입니다. 이 것을 실험장치의 emulation 환경에서 watchdog 기능을 설정하는 방법을 찾아 watchdog 기능을 enable로 하여 프로그램을 동작시켜 봅시다. 정상적으로 작성된 프로그램은 LED가 on/off를 약 1초 간격으로 반복할 것입니다. 이것을 다시 CLRWDT 명령어를 삽입하여 처음 상태가 되도록 만들고 의미를 이해하시오.