【 기술 노트 19 】

AVR 마이크로콘트롤러의 퓨즈비트 설정에 대하여

이 기술 노트에서는 AVR 마이크로콘트롤러를 사용할 때 퓨즈비트와 관련하여 발생할 수 있는 문제점과 그 해결방법을 정리한다.

1. 퓨즈비트란 무엇인가?

퓨즈비트(fuse bit)란 AVR 마이크로콘드롤러 내부에 있는 몇 바이드의 불휘발성 메모리로서 AVR 마이크로콘트롤러의 기능 설성용 제어 비트라고 말할 수 있다. 이것은 다운로드 명령을 사용하여 바이트 단위로 읽거나 쓸 수 있는 메모리지만, 각 비트가 독립적으로 고유한 제어 기능을 수행하므로 흔히 퓨즈 "바이트"보다는 퓨즈 "비트"라고 부른다.

퓨즈비트는 AVR의 모델에 따라 그 메모리의 길이나 비트 구성이 다르다. 그러나, 이것으로 설정할 수 있는 내용은 거의 비슷하여 대체로 다음과 같은 기능들을 갖는다.

- JTAG의 사용 가능 여부를 선택 (JTAGEN, OCDEN)
- 다운로드 케이블에 의한 직렬 프로그래밍 사용 가능 여부를 선택 (SPIEN)
- 시스템 클럭의 소스나 오실레이터 옵션을 선택 (CKSEL4~CKSEL0, CKOPT)
- 기동 시간을 선택 (SUT1~SUT0)
- 워치독 타이머 사용 가능 여부를 선택 (WDTON)
- BOD 기능의 사용 여부 및 BOD 전압 레벨을 선택 (BODEN, BODLEVEL)
- 부트 메모리 사이즈나 리셋 벡터를 선택 (BOOTSZ1~BOOTSZ0, BOOTRST)
- 칩삭제 명령에서 내부 EEPROM을 삭제할 것인지 여부를 선택 (EESAVE)
- 그 이전 모델과의 호환 모드로 동작할지 여부를 선택 (M103C)

위의 내용은 ATmega128을 중심으로 설명한 것인데, 이중에서 ( )안은 ATmega128에서의 퓨즈비트 이름을 나타낸다.

이러한 퓨즈비트는 해당 기능을 사용하려면 반드시 0으로 설정(즉 모든 퓨즈비트는 low active 또는 low enable 비트이다.)해야 하며, 공장에서 출고시에에는 항상 일정하게 디폴트 값으로 설정되어 있다. 예를 들어서 모든 AVR 마이크로콘트롤러는 공장 출하시에 클럭 소스로서 내부 RC 오실레이터를 사용하도록 설성되어 있기 때문에 사용자가 여기에 어떤 하드웨어를 추가하지 않아도 AVR 마이크로콘트롤러의 시스템 클럭이 기본적으로 동작한다.

(1) 퓨즈비트 하나하나의 의미를 이해하는 것이 가장 중요하다

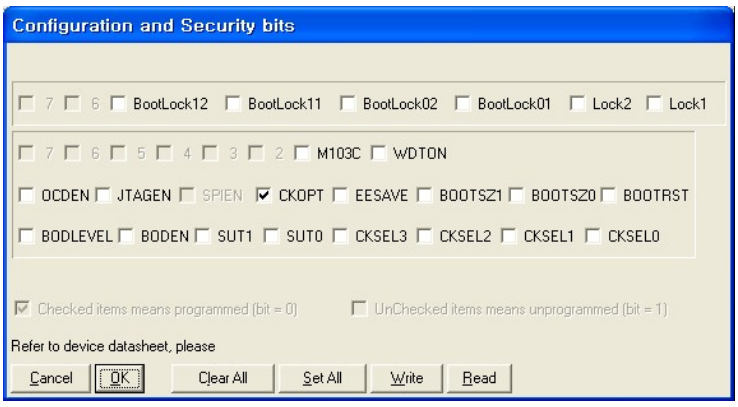
먼저 자신이 사용하는 AVR 모델의 퓨즈비트 하나하나에 대하여 그 의미를 정확히 이해해야 한다. 이를 0으로 설정하면 어떤 기능이 수행되고, 반대로 이를 1로 하면 어떻게 되는지 알아야 한다는 것이다. 어떤 비트는 쉽게 그 의미를 알 수 있지만, AVR의 구조와 기능을 좀더 깊이 공부해야만 이해되는 퓨즈비트도 있다. 예를 들어 직렬 프로그래밍 사용가능 여부를 선택하는 SPIEN 비트는 그냥 쉽게 이해가 되지만, BODEN 및 BODLEVEL 비트를 이해하려면 먼저 BOD가 뭔지 그리고 왜 BOD 레벨을 다르게 선택할 필요가 있는지를 먼저 알아야 할 것이다.

(2) 다운로드 프로그램에서 퓨즈비트 설정 방법을 정확하게 알아야 한다

AVR의 다운로드 프로그램에는 사용자 프로그램을 다운로드하는 것을 물론이고 퓨즈비트를 읽거나 라이트하는 기능을 포함하고 있다. 이들 프로그램에서는 AVR 모델을 지정하면 그 모델에서 사용하는 퓨즈비트가 화면에 나타나므로 원하는 항목들을 선택하고 라이트 아이콘을 누르면 된다. 그러나, 다운로드 프로그램에 따라 퓨즈비트를 설정하는 방법이 약간 다르므로 자칫하면 잘못 설정할 수 있으니 주의해야 한다.

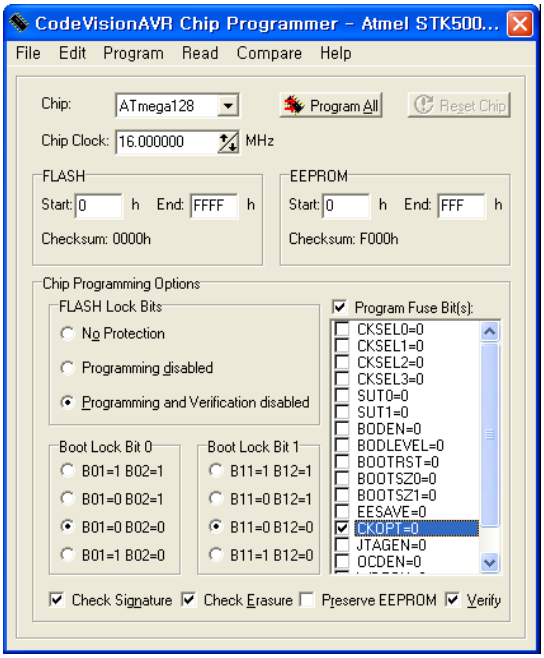
위에서도 설명하였듯이 AVR에서 모든 퓨즈비트는 low active 또는 low enable 비트이므로 해당 기능을 사용하려면 반드시 0으로 설정해야 한다. AVR에서는 이와 같이 퓨즈비트를 0으로 설정하는 것을 프로그램한다(programmed)라고 부르고, 퓨즈비트를 1로 설정하는 것을 프로그램하지 않는다(unprogrammed)라고 부른다.

① PonyProg2000에서 퓨즈비트를 설정하는 화면을 ATmega128의 경우에 대하여 보이면 <그림 19.1>과 같은데, 여기서는 0으로 설정하고 싶은 비트들을 ✓표로 체크하고 나서 Write 아이콘을 누르면 퓨즈비트가 라이트된다.

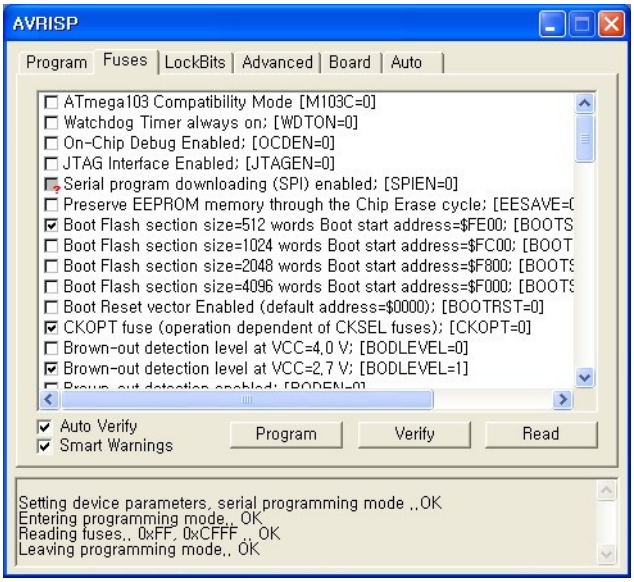


<그림 19.1> PonyProg200에서 퓨즈비트를 설정하는 화면

② CodeVision C컴파일러에는 시렬포트를 사용한 STK500 프로토콜의 시렬포트형 다운로드 케이블을 지원하는데, 이를 사용한 퓨즈비트 설정 화면을 ATmega128의 경우에 대하여 보이면 <그림 19.2>와 같다. 여기서도 0으로 설정하고 싶은 퓨즈비트들을 ✓표로 체크하고 나서 Program → Fuse Bit(s) 메뉴를 선택하면 퓨즈비트가 라이트된다.



<그림 19.2> CodeVision C컴파일러에서 퓨즈비트를 설정하는 화면



<그림 19.3> AVR Studio에서 퓨즈비트를 설정하는 화면

③ AVR Studio에는 직렬포트를 사용한 STK500 프로토콜의 직렬포트형 다운로드 케이블을 지원하는데, 이를 사용한 퓨즈비트 설정 화면을 ATmega128의 경우에 대하여 보이면 <그림 19.3>과 같다. 여기서는 앞의 2가지 경우와 다르게 하나하나의 비트를 설정하기 보다는 기능별로 원하는 항목을 선택하는 방식을 사용하는데, 오히려 이게 사용하기가 좀더 불편할 수도 있고 자칫 퓨즈비트 설정에 오류를 범하기 쉽다. 리스트된 기능을 세심하게 읽고 원하는 사항을 정확하게 선택하여 ✓표로 체크하고 나서 Program 아이콘을 누르면 퓨즈비트가 라이트된다.

다시 말하면 여기서 퓨즈비트 설정에 오류를 범하지 않기 위해는 각 항목을 세심하게 읽고 항목의 끝에 [ ]안에 해당하는 퓨즈비트를 1로 설정한다는 것인지 0으로 설정한다는 것인지를 철저하게 확인하고 선택해야 한다. 예를 들어 ATmega128에서 클럭 소스를 선택하는 항목은

☐ Ext. Clock; Start-up time: 6 CK + 0ms; [CKSEL=0000 SUT=00]
☐ Ext. Clock; Start-up time: 6 CK + 4ms; [CKSEL=0000 SUT=01]
☐ Ext. Clock; Start-up time: 6 CK + 64ms; [CKSEL=0000 SUT=10]
☐ Int. RC Osc. 1MHz; Start-up time: 6 CK + 0ms; [CKSEL=0001 SUT=00]
☐ Int. RC Osc. 1MHz; Start-up time: 6 CK + 4ms; [CKSEL=0001 SUT=01]
☐ Int. RC Osc. 1MHz; Start-up time: 6 CK + 64ms; [CKSEL=0001 SUT=10]: default value
☐ Int. RC Osc. 2MHz; Start-up time: 6 CK + 0ms; [CKSEL=0010 SUT=00]
☐ Int. RC Osc. 2MHz; Start-up time: 6 CK + 4ms; [CKSEL=0010 SUT=01]
☐ Int. RC Osc. 2MHz; Start-up time: 6 CK + 64ms; [CKSEL=0010 SUT=10]
☐ Int. RC Osc. 4MHz; Start-up time: 6 CK + 0ms; [CKSEL=0011 SUT=00]
☐ Int. RC Osc. 4MHz; Start-up time: 6 CK + 4ms; [CKSEL=0011 SUT=01]
☐ Int. RC Osc. 4MHz; Start-up time: 6 CK + 64ms; [CKSEL=0011 SUT=10]
☐ Int. RC Osc. 8MHz; Start-up time: 6 CK + 0ms; [CKSEL=0100 SUT=00]
☐ Int. RC Osc. 8MHz; Start-up time: 6 CK + 4ms; [CKSEL=0100 SUT=01]
☐ Int. RC Osc. 8MHz; Start-up time: 6 CK + 64ms; [CKSEL=0100 SUT=10]
☐ Ext. RC Osc. – 0.9MHz; Start-up time: 18 CK + 0ms; [CKSEL=0101 SUT=00]
☐ Ext. RC Osc. – 0.9MHz; Start-up time: 18 CK + 4ms; [CKSEL=0101 SUT=01]
☐ Ext. RC Osc. – 0.9MHz; Start-up time: 18 CK + 64ms; [CKSEL=0101 SUT=10]
☐ Ext. RC Osc. – 0.9MHz; Start-up time: 6 CK + 4ms; [CKSEL=0101 SUT=11]
☐ Ext. RC Osc. 0.9MHz – 3.0MHz; Start-up time: 18 CK + 0ms; [CKSEL=0110 SUT=00]
☐ Ext. RC Osc. 0.9MHz – 3.0MHz; Start-up time: 18 CK + 4ms; [CKSEL=0110 SUT=01]
☐ Ext. RC Osc. 0.9MHz – 3.0MHz; Start-up time: 18 CK + 64ms; [CKSEL=0110 SUT=10]
☐ Ext. RC Osc. 0.9MHz – 3.0MHz; Start-up time: 6 CK + 4ms; [CKSEL=0110 SUT=11]
☐ Ext. RC Osc. 3.0MHz – 8.0MHz; Start-up time: 18 CK + 0ms; [CKSEL=0111 SUT=00]
☐ Ext. RC Osc. 3.0MHz – 8.0MHz; Start-up time: 18 CK + 4ms; [CKSEL=0111 SUT=01]
☐ Ext. RC Osc. 3.0MHz – 8.0MHz; Start-up time: 18 CK + 64ms; [CKSEL=0111 SUT=10]
☐ Ext. RC Osc. 3.0MHz – 8.0MHz; Start-up time: 6 CK + 4ms; [CKSEL=0111 SUT=11]
☐ Ext. RC Osc. 8.0MHz – 12.0MHz; Start-up time: 18 CK + 0ms; [CKSEL=1000 SUT=00]
☐ Ext. RC Osc. 8.0MHz – 12.0MHz; Start-up time: 18 CK + 4ms; [CKSEL=1000 SUT=01]
☐ Ext. RC Osc. 8.0MHz – 12.0MHz; Start-up time: 18 CK + 64ms; [CKSEL=1000 SUT=10]
☐ Ext. RC Osc. 8.0MHz – 12.0MHz; Start-up time: 6 CK + 4ms; [CKSEL=1000 SUT=11]
☐ Ext. Low-Freq. Crystal; Start-up time: 1 CK + 4ms; [CKSEL=1001 SUT=00]
☐ Ext. Low-Freq. Crystal; Start-up time: 1 CK + 64ms; [CKSEL=1001 SUT=01]
☐ Ext. Low-Freq. Crystal; Start-up time: 32 CK + 64ms; [CKSEL=1001 SUT=10]
☐ Ext. Crystal/Resonator Low Freq.; Start-up time: 258 CK + 4ms; [CKSEL=1010 SUT=00]
☐ Ext. Crystal/Resonator Low Freq.; Start-up time: 258 CK + 64ms; [CKSEL=1010 SUT=01]
☐ Ext. Crystal/Resonator Low Freq.; Start-up time: 1 CK + 0ms; [CKSEL=1010 SUT=10]
☐ Ext. Crystal/Resonator Low Freq.; Start-up time: 1 CK + 4ms; [CKSEL=1010 SUT=11]
☐ Ext. Crystal/Resonator Low Freq.; Start-up time: 1 CK + 64ms; [CKSEL=1011 SUT=00]
☐ Ext. Crystal/Resonator Low Freq.; Start-up time: 16 CK + 0ms; [CKSEL=1011 SUT=01]
☐ Ext. Crystal/Resonator Low Freq.; Start-up time: 16 CK + 4ms; [CKSEL=1011 SUT=10]
☐ Ext. Crystal/Resonator Low Freq.; Start-up time: 16 CK + 64ms; [CKSEL=1011 SUT=11]
☐ Ext. Crystal/Resonator Medium Freq.; Start-up time: 258 CK + 4ms; [CKSEL=1100 SUT=00]
☐ Ext. Crystal/Resonator Medium Freq.; Start-up time: 258 CK + 64ms; [CKSEL=1100 SUT=01]
☐ Ext. Crystal/Resonator Medium Freq.; Start-up time: 1 CK + 0ms; [CKSEL=1100 SUT=10]
☐ Ext. Crystal/Resonator Medium Freq.; Start-up time: 1 CK + 4ms; [CKSEL=1100 SUT=11]
☐ Ext. Crystal/Resonator Medium Freq.; Start-up time: 1 CK + 64ms; [CKSEL=1101 SUT=00]
☐ Ext. Crystal/Resonator Medium Freq.; Start-up time: 16 CK + 0ms; [CKSEL=1101 SUT=01]

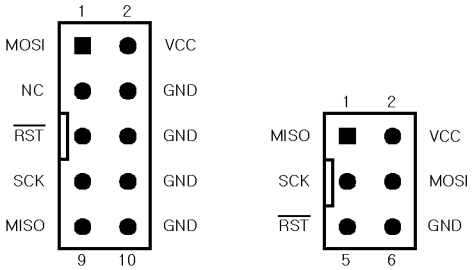
☐ Ext. Crystal/Resonator Medium Freq.; Start-up time: 16 CK + 4ms; [CKSEL=1101 SUT=10]
☐ Ext. Crystal/Resonator Medium Freq.; Start-up time: 16 CK + 64ms; [CKSEL=1101 SUT=11]
☐ Ext. Crystal/Resonator High Freq.; Start-up time: 258 CK + 4ms; [CKSEL=1110 SUT=00]
☐ Ext. Crystal/Resonator High Freq.; Start-up time: 258 CK + 64ms; [CKSEL=1110 SUT=01]
☐ Ext. Crystal/Resonator High Freq.; Start-up time: 1 CK + 0ms; [CKSEL=1110 SUT=10]
☐ Ext. Crystal/Resonator High Freq.; Start-up time: 1 CK + 4ms; [CKSEL=1110 SUT=11]
☐ Ext. Crystal/Resonator High Freq.; Start-up time: 1 CK + 64ms; [CKSEL=1111 SUT=00]
☐ Ext. Crystal/Resonator High Freq.; Start-up time: 16 CK + 0ms; [CKSEL=1111 SUT=01]
☐ Ext. Crystal/Resonator High Freq.; Start-up time: 16 CK + 4ms; [CKSEL=1111 SUT=10]
☑ Ext. Crystal/Resonator High Freq.; Start-up time: 16 CK + 64ms; [CKSEL=1111 SUT=11]

처럼 많으므로 이중에서 자신이 설정하기를 원하는 항목을 1개만 선택해야 한다. *OK-128* 키트에서처럼 외부에 16MHz의 크리스탈을 사용하는 경우에는 위의 예에서 보였듯이 가장 마지막의 항목을 선택하는 것이 좋으며, 이렇게 하면 다른 항목의 체크표 ✓는 자동적으로 해제된다.

(3) 다운로드 케이블을 올바르게 접속해야 한다

사용자 프로그램의 다운로드나 퓨즈비트 설정이 올바르게 수행되려면 우선 다운로드 케이블이 올바르게 접속되어야 한다. 하드웨어적으로 결함이 있으면 절대로 소프트웨어만으로는 올바른 다운로드 동작을 기대할 수 없다.

상업용으로 판매되는 트레이닝 키트와 다운로드 케이블을 사용하고 있다면 납땜 불량 이외에는 거의 다운로드 케이블 접속에 문제가 발생할 가능성이 적지만, 자작의 트레이닝 키트나 다운로드 케이블을 사용하고 있다면 신호의 사용이 올바르지 않을 수도 있고, 납땜의 불량 가능성도 있으며, 그밖에도 여러 가지 오류의 가능성을 염두에 두어야 한다.



(a) ISP 10핀 DIP형 (b) ISP 6핀 DIP형

<그림 19.4> Atmel 표준의 AVR ISP 다운로드 케이블 콘넥터

Atmel 표준의 10핀 및 6핀 다운로드 케이블의 콘넥터는 <그림 19.4>와 같다. 여기서 VCC와 GND는 각각 타겟 보드의 VCC와 GND에 접속한다. $\overline{\text{RST}}$는 타겟 보드에서 AVR로 들어가는 리셋신호 $\overline{\text{RESET}}$과 연결하는데, 이때 타겟 보드의 리셋신호와 다운로드 케이블에서 출력하는 리셋신호가 서로 충돌하지 않도록 하드웨어를 설계해야 한다. 그리고, MOSI,

MISO, SCK 신호는 AVR의 SPI 모듈에서 사용하는 MOSI, MISO, SCK 단자에 각각 연결한다. 그러나, ATmega128, ATmega1281, ATmega1280의 3가지 모델에서만은 AVR에 다운로드 전용신호를 따로 가지고 있으므로 다운로드 케이블의 MOSI 신호를 AVR의 PDI 신호에 연결하고, 다운로드 케이블의 MISO 신호를 AVR의 PDO 신호에 연결해야 한다.

2. 퓨즈비트를 올바르게 설정하는 방법에 관한 조언

퓨즈비트를 잘못 설정하면 AVR 마이크로콘트롤러가 엉뚱하게 동작하거나 심한 경우는 아예 동작하지 않는 경우도 있다. 이를 방지하려면 아래와 같은 사항을 먼저 이해하는 것이 좋다.

(1) 퓨즈비트를 설정하기 전에 먼저 사용자 프로그램을 다운로드하라

AVR 마이크로콘트롤러를 사용하여 처음으로 시스템을 제작하고 나서 이를 동작시키고자 한다면 먼저 퓨즈비트를 설정하려 하지 말고 우선 사용자 프로그램을 다운로드하라. 시스템을 처음 제작하였을 때는 설계에 오류가 있을 수도 있고, 납땜이 불량할 수도 있으며, 부품이 불량한 것이 포함되어 있을 수도 있다. 그런데, 만약 이런 상황에서 퓨즈비트까지 잘못 설정하여 시스템이 정상적으로 동작하지 않는 상황이 되어버린다면 그야말로 어디가 문제인지를 찾고 해결하는데 큰 어려움에 빠지게 된다.

흔히 초보자는 AVR을 처음 사용하려면 무조건 퓨즈비트를 먼저 설정해야만 된다고 생각하는 경향이 있지만 전혀 그렇지 않다. 퓨즈비트는 공장에서 출고될 때 디폴트값으로 설정되어 있으므로 기본적으로 디폴트 상태에서 동작한다. 예를 들어 시스템 클럭은 내부 RC 오실레이터에 의하여 1MHz로 동작하고 있다. 그러면 자신이 만약 16MHz의 외부 크리스탈에 의하여 동작하는 것을 전제로 프로그램을 작성했다고 하더라도 이것이 좀 느리게 동작하는 것일뿐 대부분의 기능이 별 문제없이 수행된다.(☞ 그러나, 호환 모드를 가진 AVR의 경우에는 주의가 필요하다. 예를 들어 ATmega128의 경우는 디폴트로 ATmega103 호환 모드로 설정되어 있기 때문에 만약 사용자 프로그램이 ATmega103에는 없는 기능만을 사용하도록 작성되어 있다면 이는 올바르게 동작하지 않을 수도 있다.)

일단 사용자 프로그램을 다운로드하여 느린 속도에서라도 동작을 하게 되면 이제 기본적으로 시스템의 하드웨어에는 문제가 없으며 다운로드 케이블도 올바르게 접속되어 있다는 것을 의미한다. 그러면 이제는 퓨즈비트만을 올바르게 설정해 주면 되는 것이다. 그러나, 만약 사용자 프로그램이 전혀 실행되지 않는다면 시스템의 하드웨어에 문제가 있는지를 하나씩 점검해 나가야 하며, 다운로드 에러가 발생한다면 다운로드 케이블에 이상이 없는지 또는 다운로드 케이블을 AVR 마이크로콘트롤러의 해당 신호에 올바르게 접속하였는지를 점검해 보아야 한다.

(2) 이것저것 아무렇게나 설정해보는 어리석음을 범하지 마라

엔지니어의 가장 큰 덕목 중에 하나는 과감한 탐구정신이다. 그러나, AVR의 퓨즈비트에 대하여 무모하게 탐구정신을 발휘하여 이것저것 임의로 설정해 보는 것은 절대로 금물이다. 머리를 사용하는 것이 귀찮아서 몸으로 이것저것을 먼저 저질러보는 방법으로 잘못된 탐구정신을 발휘하다보면 시스템이 정지하여 이제부터 많은 시간과 돈을 낭비하고 고생하게 된다. 그러나, 그 과정에서 중요한 기술적인 사항을 경험하고 교훈을 얻으면 괜찮은데 AVR은 사용하기 골아픈 것이라거나 퓨즈비트는 무조건 위험한 것이라는 잘못된 사고방식만 가지게 되는 것이 문제다. 퓨즈비트는 참으로 편리한 기능이며 이 때문에 AVR은 그만큼 다양한 기능을 가지게 된 것이다. 기능이 다양하다는 것은 정상적인 사용자에게는 편리한 것이지만 이를 잘모르는 초보자들에게는 골아프고 헷갈리는 것일 수도 있다.

간혹 다운로드 프로그램의 기능을 잘 몰라서 실수하는 경우도 있다. 다운로드 프로그램의 기능을 잘 모르면서 이것저것 함부로 체크하거나 아무 아이콘이나 마구잡이로 눌러보는 것도 금물이다. 그러다가 엉뚱한 값으로 퓨즈비트가 라이트되는 수가 있다.

퓨즈비트에 관한한 모험을 하기 전에 먼저 퓨즈비트와 다운로드 프로그램을 제대로 알아야 한다. 내가 정확히 그 기능을 알고 있는 부분이 아니면 함부로 설정하거나 변경하지 마라. 때로는 퓨즈비트를 이해하기 위하여 AVR을 좀더 깊이 알아야 하는 경우도 있다. 그렇다면 당연히 이 마이크로콘트롤러에 대하여 공부해야 한다. 행동하기 전에 먼저 이해하라. 그것이 퓨즈비트 설정의 기본 원칙이다.

(3) 퓨즈비트를 읽어 필요한 비트만 수정한 후에 다시 라이트하라

그래도 퓨즈비트에 대하여 모두 이해가 되지 않았는데 퓨즈비트의 일부 기능을 수정하여 설정하고 싶다면 그렇게 할 수 있다. 즉, 퓨즈비트의 다른 부분은 건드리지 말고 원하는 부분을 건드리면 된다는 것이다. 그러나, 이때 자신이 원하는 부분만을 0이 되도록 체크하고 라이트하면 될 것이라고 생각하면 큰 오산이다. 퓨즈비트는 몇 개의 바이트로 구성되어 있어서 어느 한 비트만을 라이트하지 못하고 바이트 단위로만 라이트하기 때문에 원하지 않는 인접한 다른 비트들도 함께 변경되기 때문이다.

이런 경우에는 먼저 퓨즈비트를 읽어들여라. 어느 다운로드 프로그램에나 퓨즈비트를 라이트(write)하는 기능 및 리드(read)하는 기능을 가지고 있다. 퓨즈비트를 읽어들이면 현재 AVR 모델에 설정되어 있는 값들이 읽히게 되므로 이 상태에서 자신이 원하는 부분만을 수정하고 이를 다시 라이트하면 된다.

그리고, 퓨즈비트는 한번 설정하였으면 특별한 상황이 아니면 다시 건드릴 필요가 없다. 따라서, 이제부터 사용자 프로그램을 다운로드할 때는 쓸데없이 퓨즈비트가 함께 라이트되지 않도록 설정해 두는 것이 좋다. 그 방법은 소프트웨어마다 다르다.

(4) 가장 많이 하는 실수

초보 AVR 사용자가 퓨즈비트를 설정할 때 가장 많이 범하는 실수는 시스템 클럭을 잘못 선택하는 것이다. 시스템 클럭을 잘못 선택하면 아예 마이크로콘트롤러가 동작하지 않기 때문에 엉뚱하게 이를 교체하는 제2차적인 실수를 범하게 된다. DIP형 소자인 경우에는 이를 교체하는 것이 간단한 일이지만 SMD형 팩키지인 경우는 작업이 만만하지 않은데다가 벌쩡한 소자를 하나 버리게 되고 자칫하면 이를 제거하다가 PCB까지 망가뜨리는 사고로 이어질 수 있기 때문에 이로 인하여 벌어지는 결과는 참담할 수 있다. 머리가 나쁘면 몸이 고생한다는 말은 이런 때 딱 어울린다.

이렇게 퓨즈비트에서 시스템 클럭을 잘못 선택하는 사고가 발생하는 것은 대부분의 설정 오류가 모든 비트를 0으로 잘못 설정하거나 모든 비트를 1로 잘못 설정하는 것인데, 이중에서 모든 퓨즈비트를 0으로 설정하게 되면 이는 시스템 클럭을 선택하는 기능에 해당하는 CKSEL3~0 비트가 0000이 되고 이는 외부 클럭으로 작용하기 때문이다. 즉, 시스템 클럭으로 외부 클럭을 선택한다는 것은 AVR 마이크로콘트롤러가 외부에서 XTAL1 단자로 구형파 클럭신호가 들어오기를 기다리는 것인데, 사용자는 내부의 RC 발진을 사용하려 하거나 아니면 외부에 크리스탈을 접속해 놓고 이것의 발진을 기대하고 있으니 시스템이 제대로 동작할 턱이 없는 것이다. 따라서, 실계에 문제가 없고 납땜에도 문제가 없다고 판단되는데 시스템이 동작하지 않으며 다운로드 케이블도 동작하지 않으면 이 상황을 의심해 보아야 한다. 이 문제의 해결 방법은 무조건 다음 절에서 설명하는 인공호흡 방법이다.

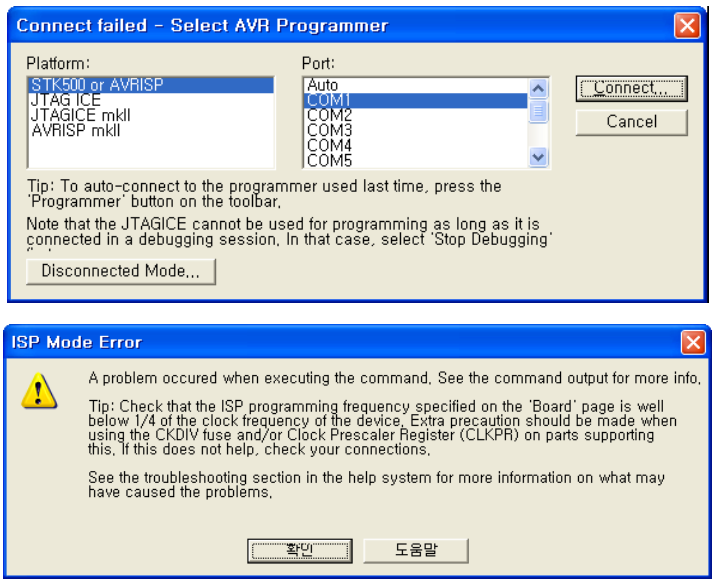
또 하나 가끔 보고되는 사고는 JTAG이나 다운로드 케이블의 접속을 해제하는 설정을 하는 것이다. 즉, JTAG 에뮬레이터를 사용하면서 퓨즈비트에서 JTAGEN 비트를 1로 해제해 버리거나 다운로드 케이블을 사용하면서 퓨즈비트에서 SPIEN 비트를 1로 해제해 버리는 것이다. 이렇게 되면 이제부터는 더 이상 해당 통신이 수행되지 않기 때문에 AVR 마이크로콘트롤러에 소프트웨어적으로 접근할 수가 없다. 이런 경우는 반대편 방법으로 해결을 시도해 볼 수 있다. 즉, 만약 JTAG 에뮬레이터를 사용하다가 JTAGEN 비트를 1로 해제하였다면 아직 SPIEN 비트를 0으로 되어 있어서 직렬 프로그램이 가능할 수도 있다는 가정하에 다운로드 케이블을 사용해 보는 것이며, 반대로 다운로드 케이블을 SPIEN 비트를 1로 해제하였다면 아직 JTAGEN 비트를 0으로 되어 있어서 JTAG 인터페이스가 가능할 수도 있다는 가정하에 에뮬레이터를 사용해 보는 것이다. 그러나, 이들 비트가 모두 1로 해제되어 있다면 이제는 해결이 사실상 불가능하므로 이 AVR 칩은 버리는 수밖에 없다. 병렬 프로그램 방법이 남아있기는 하지만 이는 해당 기능을 가지는 ROM 라이터가 필요한데 AVR에서는 이를 잘 사용하지 않으므로 현실적으로 어려운 방법이다. 이런 사고의 위험성 때문에 대부분의 다운로드 프로그램에서는 SPIEN 비트를 사용자가 변경할 수 없도록 하고 있다.

3. 퓨즈비트 설정 오류로 클럭이 동작하지 않을 때의 인공호흡 방법

(1) 시스템이 전혀 동작하지 않을 때는 퓨즈비트 설정을 의심하라

AVR을 사용한 시스템에서 하드웨어 설계에 문제가 없고 납땜에도 문제가 없다고 판단되는데 시스템이 동작하지 않으며 다운로드 케이블도 연결되지 않는다면 퓨즈비트를 잘못 설정하여 AVR 마이크로콘트롤러가 동작하지 않을 수 있다는 것을 의심해 보아야 한다. 특히 시스템이 제작 후에 처음부터 전혀 동작하지 않았다면 설계나 조립을 의심해 보는 것이 좋지만, 잘 동작하던 시스템이 어느 순간부터 갑자기 동작하지 않는다면 거의 틀림없이 이 문제일 가능성이 높다. 필자는 이제까지 "잘 동작하던 키트가 갑자기 동작하지 않는다. 퓨즈비트는 건드리지도 않았다."고 말하는 사람을 많이 보았는데, 이런 경우는 100% 퓨즈비트 설정 오류였으며 퓨즈비트는 건드리지도 않았다는 그의 말은 결과적으로 명백한 거짓말이었으므로 전혀 믿을 바가 되지 못했다.

<그림 19.5> PonyProg2000에서 다운로드 케이블이 연결되지 않는 에러 메시지 화면



<그림 19.6> AVR Studio에서 다운로드 케이블이 연결되지 않는 에러 메시지 화면

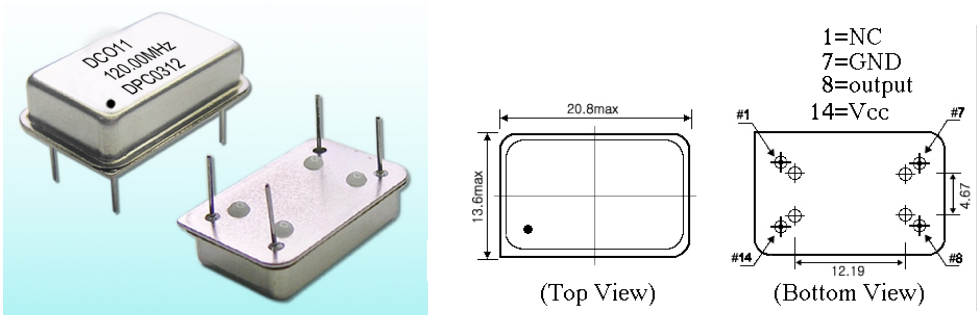
다운로드 케이블이 연결되지 않을 때 PonyProg2000을 사용하는 경우라면 <그림 19.5>와 같은 에러 메시지 화면이 나타난다. 그러나, AVR Studio를 사용하는 경우라면 <그림

19.6>처럼 아예 직렬통신 포트에 접속이 안되거나 또는 접속이 된 이후에도 에러 메시지 화면이 나타난다.

이와 같이 퓨즈비트 설정의 오류로 AVR 마이크로콘트롤러가 동작하지 않는 것은 실제로 하드웨어는 XTAL1과 XTAL2 단자에 크리스탈 소자를 붙여 놓았는데 사용자가 퓨즈비트를 설정한 것은 외부 클럭으로 잘못 설정되어 AVR 마이크로콘트롤러의 외부에서 XTAL1 단자로 구형파 클럭신호가 들어오기를 기다리고 있기 때문이다. 따라서, 이를 해결하려면 우선 외부에서 AVR 마이크로콘트롤러의 XTAL1 단자에 클럭신호를 인가하여 최소한 AVR이 동작하도록 해주어야 한다. 그래야만 다시 퓨즈비트를 올바르게 수정할 수 있기 때문이다. 어느 마이크로콘트롤러나 그렇듯이 AVR 마이크로콘트롤러가 동작하기 위한 최소한의 조건은 전원전압, 리셋신호, 시스템 클럭이다. 이와 같이 전원전압과 리셋신호는 정상인 상태에서 외부로부터 클럭신호를 입력하는 방법을 AVR의 엔지니어들 사이에서는 “인공호흡”이라고 한다. 제법 멋진 이름이다. “인공호흡”이라는 말을 듣는 순간 이 문제를 해결하기 위하여 어떻게 해주어야 하는지 머릿속에 팍팍 떠오르지 않는가!

(2) 외부 오실레이터를 사용한 인공호흡 방법

퓨즈비트 설정 오류로 인하여 동작하지 않는 AVR 마이크로콘트롤러에게 인공호흡을 실시하려면 외부의 구형파 클럭신호가 필요하다. 당연히 현재 AVR을 5V로 사용하고 있으면 클럭신호로 5V 레벨의 구형파가 필요하고, AVR을 3.3V로 사용하고 있으면 클럭신호로 3.3V 레벨의 구형파가 필요하다. 이러한 외부 클럭은 실험실의 평션 제너레이터를 사용할 수도 있고, 좀더 간편하게는 크리스탈 오실레이터 소자를 사용할 수도 있다.



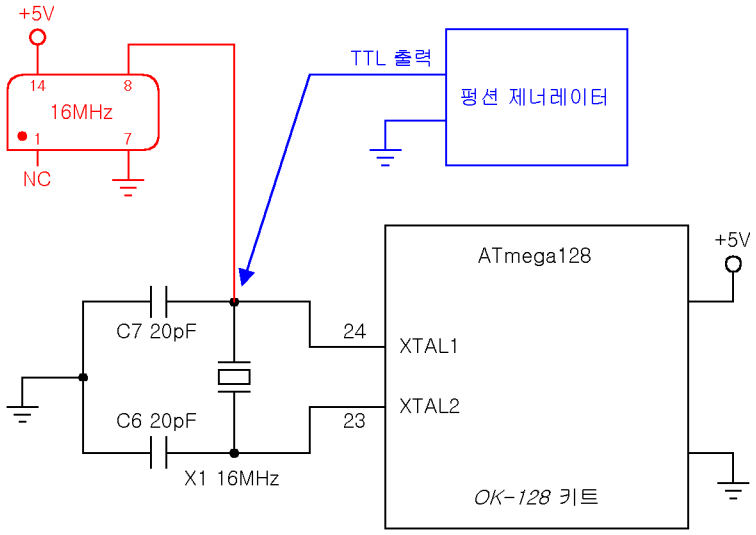
<그림 19.7> 크리스탈 오실레이터

<그림 19.7>에 크리스탈 오실레이터를 보였다. 크리스탈은 발진회로를 구성하기 위한 하나의 부품에 불과하지만, 크리스탈 오실레이터는 크리스탈 소자와 기타의 부품으로 구성된 발진회로를 1개의 소자로 몰딩시켜 만든 것이다. 이것은 5V 전원과 3.3V 전원에 공통으로

사용할 수 있다. 이 그림에 보였듯이 크리스탈 오실레이터는 4핀으로 구성되는데, 이는 TTL처럼 14핀 DIP형 소자에 대응하는 핀을 따라 1, 7, 8, 14번 핀으로 부른다. 간혹 이를 그냥 1, 2, 3, 4번 핀으로 부르는 경우도 있다.

예를 들어 ATmega128을 사용한 *OK-128* 키트 V3.0에서 인공호흡을 실시하기 위한 회로의 접속방법은 보이면 <그림 19.8>과 같다. 인공호흡은 적색으로 표시한 것처럼 크리스털 오실레이터를 사용할 수도 있고, 청색으로 표시한 것처럼 펑션 제너레이터를 사용할 수도 있다. 이때 타겟 보드에서 XTAL1 및 XTAL2 단자에 크리스탈이 붙어 있다면 이는 상당히 임피던스가 크므로 그냥 두어도 상관없다.

외부에서 크리스탈 오실레이터나 펑션 제너레이터로 인가하는 클럭신호의 주파수는 타겟 AVR의 사양 범위 이내라면 어떤 것을 사용해도 좋다. 예를 들어 타겟 AVR이 16MHz 버전인 ATmega128이라면 16MHz 이하를 사용하면 되고, 8MHz 버전인 ATmega128L이라면 8MHz 이하를 사용하면 된다.



<그림 19.8> 크리스탈 오실레이터 또는 펑션 제너레이터를 사용한 인공호흡 방법

이와 같이 회로를 구성하고 나서 퓨즈비트를 수정하여 시스템 클럭을 원하는 대로 외부 크리스탈 또는 내부 RC 오실레이터로 선택한다. 그리고 나서 인공호흡을 위하여 임시로 접속하였던 외부 오실레이터나 펑션 제너레이터를 제거하면 된다.

【 참고문헌 】

1. 윤덕용, AVR ATmega128 마스터, Ohm사, 2004
2. 윤덕용, AVR ATmega128 정복, Ohm사, 2006