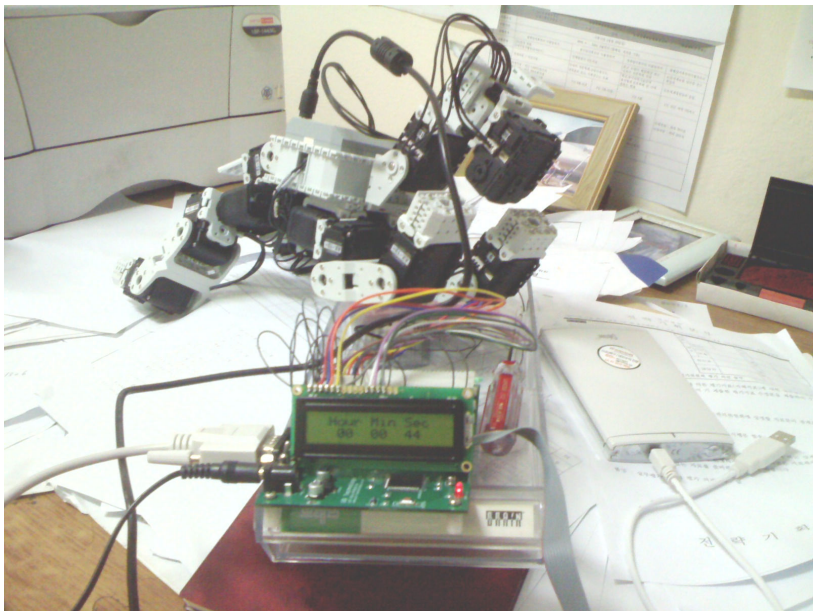
마이콤 기초와 응용 강의노트
(ATmega128 시스템)



최한호
서울시 중구 필동 3가 26번지 동국대학교 전기공학과
TEL: 2260-3777, FAX:2275-6013
e-mail : hhchoi@dongguk.edu, WWW: //home.dongguk.edu/user/hhchoi

제목 차례

1. AVR 개요 ····· 1
1.1. AVR의 일반적 특징 ····· 1
1.2. AVR의 종류 ····· 1
1.3. ATmega128의 특징 ····· 3
1.4. 외형과 핀 기능 ····· 4
1.5. ATmega128의 메모리 ····· 7
1.5.1. 프로그램 메모리 ····· 7
1.5.2. 데이터메모리 : 범용레지스터 ····· 8
1.5.2.1. X,Y,Z 레지스터 ····· 8
1.5.3. 데이터 메모리 : I/O 레지스터 ····· 8
1.5.3.1. 상태레지스터(SREG) ····· 9
1.5.3.2. 스택 포인터(SP) ····· 10
1.5.3.3. RAMPZ(RAM Page Z select) 레지스터 ····· 10
1.5.4. 데이터 메모리 : 확장 I/O 레지스터 ····· 10
1.5.5. 데이터 메모리 : 내부 SRAM ····· 10
1.5.6. 데이터 메모리 : EEPROM ····· 11
1.5.6.1. EEAR 레지스터 ····· 12
1.5.6.2. EEDR 레지스터 ····· 12
1.5.6.3. EECR 레지스터 ····· 12
1.5.6.4. EEPROM 쓰기과정 ····· 12
1.5.6.5. EEPROM 읽기과정 ····· 12
1.5.7. 외부 데이터 메모리 ····· 13
1.5.7.1. 외부 메모리 인터페이스 ····· 13
1.5.7.2. MCUCR레지스터 ····· 14
1.5.7.3. XMCRA레지스터 ····· 14
1.5.7.4. XMCRB레지스터 ····· 14
1.5.8. 메모리 lock 비트 ····· 15
1.5.9. 퓨즈 비트 ····· 16
1.5.9.1. Extended Fuse Byte ····· 16
1.5.9.2. Fuse High Byte ····· 16
1.5.9.3. Fuse Low Byte ····· 16
1.6. 시스템 클록과 슬립 모드 ····· 17
1.6.1. 클록 분배 ····· 17
1.6.2. 클록 발생과 선택 ····· 17
1.6.2.1. 내부 RC(디폴트 클록) 발진기 ····· 17
1.6.2.2. 외부 RC 발진기 ····· 18
1.6.2.3. 외부 수정발진기 ····· 19
1.6.2.4. 저주파 수정 발진기 ····· 19
1.6.2.5. 외부 클록 ····· 20
1.6.2.6. XDIV레지스터를 이용한 클럭 주파수 조정 ····· 20
1.6.3. 슬립모드 ····· 20
1.6.3.1. MCUCR레지스터를 이용한 모드 설정 ····· 21

1.6.3.2. 아이들 모드 ····· 21
1.6.3.3. ADC noise reduction 모드 ····· 21
1.6.3.4. Power-down 모드 ····· 21
1.6.3.5. Power-save 모드 ····· 21
1.6.3.6. Standby 모드 ····· 21
1.6.3.7. Extended Standby 모드 ····· 21
1.6.3.8. 슬립모드의 동작 요약 ····· 22
1.7. Reset과 워치독 타이머 ····· 22
1.7.1. 리셋 ····· 22
1.7.1.1. 파워온 리셋(POR) ····· 23
1.7.1.2. 외부 리셋 ····· 23
1.7.1.3. Brown-out 리셋 ····· 24
1.7.1.4. 워치독 리셋 ····· 24
1.7.1.5. MCUCSR 레지스터 ····· 25
1.7.2. 워치독 타이머 ····· 25
1.7.2.1. 워치독 타이머의 구성 ····· 25
1.7.2.2. WDTCR 레지스터 ····· 25
1.7.2.3. 워치독의 설정 ····· 26
2. ATmega128 시스템 개발 기초 ····· 27
2.1. 명령어 분류와 주소지정 방식 ····· 27
2.2. 명령어 요약 ····· 28
2.2.1. 어셈블리어의 일반형태 ····· 30
2.2.2. 지시어(의사 명령) ····· 30
2.2.3. 숫자, 문자, 연산자 ····· 32
2.2.4. 함수 ····· 32
2.3. AVR Studio4 ····· 33
2.3.1. 새 프로젝트 등록과 실행 파일 만들기 ····· 33
2.3.2. 디버깅과 시뮬레이션 ····· 37
2.4. AVR의 ISP ····· 40
2.4.1. 마이컴 시스템 개발 방법 ····· 40
2.4.1.1. ICE(In Cricuit Emulator)를 이용하는 법 ····· 40
2.4.1.2. ROM Emulator를 이용하는 법 ····· 40
2.4.1.3. ROM Writer를 이용하는 법 ····· 41
2.4.1.4. 외부 RAM을 이용하는 법 ····· 41
2.4.1.5. ISP를 이용하는 법 ····· 41
2.4.2. AVR ISP용 다운로더 ····· 42
2.4.2.1. AVR Studio와 ATAVRISP ····· 42
2.4.2.2. Atmel AVR ISP 프로그래머 ····· 42
2.4.2.3. PonyProg2000 ····· 42
2.4.2.4. Codevisin AVR ····· 43
2.5. Codevision AVR C 컴파일러 ····· 43
2.5.1. 새 프로젝트 등록하기와 옵션 설정하기 ····· 43
2.5.2. 새 소스 파일 만들기 ····· 46
2.5.3. 프로젝트에 소스파일 등록하고 실행파일 만들기 ····· 47
2.5.4. 디버깅 ····· 51

2.5.5. 플래쉬롬 ISP 프로그래밍하기 ····· 54
2.5.6. EEPROM ISP 프로그래밍하기 ····· 56
2.6. C 프로그래밍에서 유의할 점 ····· 56
2.6.1. C 프로그램의 구성 요소 ····· 56
2.6.2. C 프로그램의 일반적 형식 ····· 56
2.6.2.1. 전처리 ····· 57
2.6.2.2. 선언문 ····· 57
2.6.2.3. 주석 ····· 57
2.6.2.4. 함수와 main 함수 부분 ····· 57
2.6.3. 기본 데이터형 ····· 57
2.6.3.1. 문자형 ····· 58
2.6.3.2. 정수형 ····· 58
2.6.3.3. 실수형 ····· 58
2.6.3.4. 변수나 상수 이름 짓는 법 ····· 58
2.6.3.5. 상수 정의 ····· 59
2.6.3.6. 변수 선언과 초기화 ····· 59
2.6.4. 확장 데이터형 ····· 59
2.6.4.1. bit 데이터형 ····· 59
2.6.4.2. eeprom 데이터형 ····· 59
2.6.4.3. flash 데이터형 ····· 59
2.6.4.4. sfrb, sfrw 데이터형 ····· 59
2.6.5. 연산자 ····· 59
2.6.5.1. 조건 연산자 ····· 59
2.6.5.2. 데이터 형변환 연산자 ····· 59
2.6.5.3. 콤마 연산자 ····· 60
2.6.5.4. 산술 연산자 ····· 60
2.6.5.5. 논리와 비교 연산자 ····· 60
2.6.5.6. 비트 연산자 ····· 60
2.6.5.7. 대입 연산자 ····· 61
2.6.6. 제어문 ····· 61
2.6.6.1. if 문 ····· 61
2.6.6.2. switch-case 문 ····· 61
2.6.6.3. for 문 ····· 62
2.6.6.4. while 문 ····· 62
2.6.6.5. do-while 문 ····· 62
2.6.6.6. break 문 ····· 62
2.6.6.7. continue 문 ····· 62
2.6.6.8. goto 문 ····· 62
2.6.7. 함수와 변수 ····· 62
2.6.7.1. 프로토타입 선언 ····· 62
2.6.7.2. 일반 함수의 정의 ····· 62
2.6.7.3. 라이브러리 함수 ····· 63
2.6.7.4. 인터럽트 함수의 정의 ····· 63
2.6.7.5. 변수 ····· 63
2.6.7.6. 어셈블리어와 결합 ····· 64

2.6.8. 포인터와 배열 ····· 64
2.6.8.1. 배열 선언 방법 ····· 64
2.6.8.2. 배열 초기화 ····· 64
2.6.8.3. 포인터 선언 방법 ····· 65
2.6.8.4. 포인터 초기화 및 참조 방법 ····· 65
2.6.8.5. 포인터의 연산 ····· 65
2.6.8.6. 함수 포인터 ····· 65
2.6.8.7. 배열과 포인터의 차이점 ····· 66
2.6.9. typedef, structure, union, enum ····· 66
2.6.9.1. typedef를 사용한 새로운 데이터형 정의법 ····· 66
2.6.9.2. 구조체의 개념과 필요성 ····· 66
2.6.9.3. 구조체 선언법과 초기화 ····· 66
2.6.9.4. 구조체 사용법 ····· 67
2.6.9.5. 공용체(union) ····· 67
2.6.9.6. 열거형 상수 ····· 67
3. 기본 프로그래밍과 입출력 실험 ····· 68
3.1. 기본 프로그래밍 연습 ····· 68
3.1.1. 실험1 : 내부램에 데이터 쓰기1 ····· 68
3.1.2. 실험2 : 내부램에 데이터 쓰기2 ····· 68
3.1.3. 실험3 : EEPROM에 데이터 쓰기와 읽기 ····· 68
3.1.4. 실험4 : 롬 데이터 읽기 ····· 69
3.1.5. 실험5 : 롬데이터 검색1 ····· 70
3.1.6. 실험6 : 롬데이터 검색2 ····· 70
3.1.7. 실험7 : 8비트 2진수의 BCD 변환 ····· 70
3.1.8. 실험8 : 16비트 2진수의 BCD 변환 ····· 71
3.1.9. 실험9 : 아스키 문자의 16진수 변환 ····· 72
3.1.10. 실험10 : 숫자의 아스키코드 변환 ····· 72
3.1.11. 실험11 : 지연 루틴 ····· 73
3.2. I/O Port ····· 74
3.2.1. 관련 레지스터 ····· 74
3.2.1.1. DDRx 레지스터 ····· 74
3.2.1.2. PORTx 레지스터 ····· 74
3.2.1.3. PINx 레지스터 ····· 74
3.2.1.4. SFIOR 레지스터 ····· 74
3.2.2. 구조 및 동작 ····· 74
3.3. 단순 출력 연습 ····· 75
3.3.1. 실험12 : LED 점멸1 ····· 75
3.3.2. 실험13 : LED 점멸2 ····· 77
3.3.3. 실험14 : LED 점멸3 ····· 77
3.3.4. 실험15 : LED 점멸4 ····· 78
3.4. 단순 입출력 연습 ····· 78
3.4.1. 실험16 : 키 누름수 세기 ····· 78
3.4.2. 실험17 : 파일럿 램프1 ····· 80
3.4.3. 실험18 : 파일럿 램프2 ····· 80
3.4.4. 실험19 : 입출력함수 구현 ····· 81

3.4.5. 실험20 : 입력값에 따른 LED 점멸속도 조절 ····· 81
4. 인터럽트와 타이머 실험 ····· 82
4.1. 외부 인터럽트 실험 ····· 82
4.1.1. 인터럽트의 개념과 종류 ····· 82
4.1.2. ATmega128의 인터럽트 처리과정 ····· 82
4.1.3. 인터럽트 처리 시간 ····· 82
4.1.4. 인터럽트 제어 ····· 83
4.1.4.1. 벡터 배치 ····· 83
4.1.4.2. MCUCR 레지스터 ····· 83
4.1.4.3. 허용과 우선순위 ····· 83
4.1.4.4. 외부 인터럽트 트리거 ····· 83
4.1.4.5. EICRA 레지스터 ····· 85
4.1.4.6. EICRB 레지스터 ····· 85
4.1.4.7. EIMSK 레지스터 ····· 85
4.1.4.8. EIFR 레지스터 ····· 85
4.1.5. 실험21 : LED 점멸1 ····· 85
4.1.6. 실험22 : LED 점멸2 ····· 86
4.1.7. 실험23 : 카운터 ····· 87
4.2. 타이머 실험 ····· 88
4.2.1. 타이머 카운터0, 2 ····· 88
4.2.1.1. 구성 ····· 89
4.2.1.2. TCCRn 레지스터 ····· 89
4.2.1.3. TIMSK 레지스터 ····· 90
4.2.1.4. TIFR 레지스터 ····· 90
4.2.1.5. ASSR 레지스터 ····· 90
4.2.1.6. SFIOR 레지스터 ····· 91
4.2.1.7. 타이머0,2의 일반 모드 ····· 91
4.2.1.8. 타이머0,2의 CTC모드 ····· 91
4.2.1.9. 타이머 0, 2의 고속 PWM 모드 ····· 92
4.2.1.10. 타이머 0, 2의 Phase Correct PWM 모드 ····· 93
4.2.2. 타이머 카운터1, 3 ····· 93
4.2.2.1. 구성 ····· 94
4.2.2.2. TCCRxA 레지스터 ····· 95
4.2.2.3. TCCRxB 레지스터 ····· 95
4.2.2.4. TCCRxC 레지스터 ····· 96
4.2.2.5. TIMSK 레지스터 ····· 96
4.2.2.6. ETIMSK 레지스터 ····· 96
4.2.2.7. TIFR 레지스터 ····· 97
4.2.2.8. ETIFR 레지스터 ····· 97
4.2.2.9. 타이머1, 3의 일반 모드 ····· 97
4.2.2.10. 타이머1, 3의 CTC모드 ····· 98
4.2.2.11. 타이머 1, 3의 고속 PWM 모드 ····· 98
4.2.2.12. 타이머 1, 3의 Phase Correct PWM 모드 ····· 98
4.2.2.13. 타이머 1, 3의 Phase and Frequency Correct PWM 모드 ····· 99
4.2.2.14. 출력비교 변조 ····· 99

4.2.3. 실험24 : 타이머를 이용한 LED 점멸1 ····· 100
4.2.4. 실험25 : 타이머를 이용한 LED 점멸2 ····· 100
4.2.5. 실험26 : 타이머를 이용한 LED 점멸3 ····· 101
4.2.6. 실험27 : 타이머를 이용한 LED 점멸4 ····· 102
4.2.7. 실험28 : 타이머를 이용한 LED 점멸5 ····· 102
4.2.8. 실험29 : 워치독 타이머 ····· 103
4.2.9. 실험30 : 타이머를 이용한 카운터 ····· 104
4.2.10. 실험31 : 펄스 폭 측정 ····· 104
4.3. 인터럽트와 타이머 복합 실험 ····· 105
4.3.1. 실험32 : LED 점멸 속도 조절 ····· 105
4.3.2. 실험33 : 스톱 워치 ····· 105
4.3.3. 실험34 : PWM 신호 발생1 ····· 106
4.3.4. 실험35 : PWM 신호 발생2 ····· 107
5. 응용 실험 ····· 109
5.1. FND(Flexible Numeric Display) ····· 109
5.1.1. FND의 개요 ····· 109
5.1.2. 실험36 : 디코더 없는 FND 구동 ····· 109
5.1.3. 실험37 : 디코더를 사용한 FND구동 ····· 110
5.1.4. 실험38 : FND 동적 구동 ····· 110
5.1.5. 실험39 : 시계 ····· 111
5.1.6. 실험40 : 스톱워치 ····· 112
5.1.7. 실험41 : 시간 조정이 가능한 시계 ····· 113
5.2. 키보드 스캐닝 ····· 115
5.2.1. 실험42 : 단순 키보드 스캔 ····· 115
5.2.2. 실험43 : 키 매트릭스 스캔 ····· 116
5.3. 음향 발생 ····· 117
5.3.1. 음향 발생 개요 ····· 117
5.3.2. 실험44 : 사이렌 소리내기 ····· 118
5.3.3. 실험45 : 전화벨 소리내기 ····· 119
5.3.4. 실험46 : 동요 재생하기 ····· 120
5.3.5. 실험47 : 전자 오르간 ····· 121
5.4. 16x2 문자 LCD ····· 122
5.4.1. 16x2 문자 LCD 모듈 개요 ····· 122
5.4.2. LCD 제어기의 내부 구성 ····· 123
5.4.3. 명령어 ····· 123
5.4.4. 초기화 방법(4비트 인터페이스 경우) ····· 124
5.4.5. 실험48 : LCD 초기화 ····· 124
5.4.6. 실험49 : 점멸하며 문자표시하기 ····· 127
5.4.7. 실험50 : 사용자 폰트 이용하기 ····· 127
5.4.8. 실험51 : 시간 조정이 가능한 시계 ····· 128
5.5. 모터 제어 ····· 129
5.5.1. 스텝모터 개요 ····· 129
5.5.1.1. 특징 ····· 129
5.5.1.2. 종류 ····· 130
5.5.1.3. 특성과 사양 ····· 130

5.5.1.4. 4상 스텝 모터의 결선 확인법 ····· 130
5.5.2. 4상 스텝 모터의 구동원리 ····· 131
5.5.2.1. 1상 여자 방법 ····· 131
5.5.2.2. 2상 여자 방법 ····· 131
5.5.2.3. 1-2상 여자 방법 ····· 131
5.5.3. 스텝 모터의 위치 제어법 ····· 132
5.5.4. 스텝 모터의 속도 제어법 ····· 132
5.5.5. 스텝 모터의 가감속 제어시 펄스레이트 결정법 ····· 133
5.5.6. 모터 구동용 H브리지 회로 ····· 133
5.5.7. 모터 구동용 IC L298의 개요 ····· 134
5.5.8. 실험52 : 스텝 모터 정속 제어 ····· 135
5.5.9. 실험53 : 스텝 모터 속도 제어 ····· 136
5.5.10. 실험54 : 스텝 모터 가감속 제어 ····· 137
5.5.11. 직류모터 개요 ····· 137
5.5.11.1. 특징 ····· 137
5.5.11.2. 직류모터 구동원리 ····· 138
5.5.11.3. 직류모터 구조 ····· 138
5.5.12. 실험55 : 직류모터 속도 제어1 ····· 138
5.5.13. 실험56 : 직류모터 속도 제어2 ····· 139
5.6. D/A 변환 ····· 140
5.6.1. D/A변환기의 종류 ····· 140
5.6.2. DAC 0800소개 ····· 141
5.6.3. 실험57 : 톱니파 발생 ····· 141
5.6.4. 실험58 : 사인파 발생 ····· 141
5.7. A/D 변환 ····· 142
5.7.1. A/D변환과정 ····· 142
5.7.2. A/D변환기의 종류 ····· 143
5.7.3. ATmega128의 ADC ····· 143
5.7.3.1. 특징 ····· 143
5.7.3.2. ADCH, ADCL 레지스터 ····· 144
5.7.3.3. ADMUX 레지스터 ····· 144
5.7.3.4. ADCSRA 레지스터 ····· 144
5.7.3.5. 동작 ····· 144
5.7.3.6. 잡음 제거 방법 ····· 145
5.7.4. 실험59 : 조도 변화 측정 ····· 146
5.8. 아날로그 비교기 ····· 147
5.8.1. 구성 ····· 147
5.8.2. 제어 ····· 148
5.8.2.1. ACSR 레지스터 ····· 148
5.8.2.2. SFIOR 레지스터 ····· 148
5.8.3. 실험60 : 아날로그 비교기 실험 ····· 149
5.9. 직렬 통신 ····· 150
5.9.1. SPI 포트 제어 ····· 150
5.9.1.1. SPI의 동작 ····· 150
5.9.1.2. SPCR 레지스터 ····· 151

5.9.1.3. SPSR 레지스터 ····· 151
5.9.2. USART 포트 제어 ····· 152
5.9.2.1. 데이터 프레임 포맷 ····· 152
5.9.2.2. UDRn 레지스터 ····· 152
5.9.2.3. UCSRnA 레지스터 ····· 152
5.9.2.4. UCSRnB 레지스터 ····· 153
5.9.2.5. UCSRnC 레지스터 ····· 153
5.9.2.6. UBRRnH, UBRRnL 레지스터 ····· 154
5.9.3. RS232C 개요 ····· 154
5.9.4. 실험61 : PC와 통신1 ····· 155
5.9.5. 실험62 : PC와 통신2 ····· 156
5.9.6. 실험63 : PC와 통신3 ····· 156
5.9.7. 실험64 : PC와 통신4 ····· 157
5.9.8. 실험65 : PC와 통신5 ····· 157
6. 참고문헌 ····· 161
7. 색인 ····· 162

표 차례

표 1. AVR의 종류와 사양 ····· 2
표 2. 포트B의 다른 기능 설명 ····· 5
표 3. 포트D의 다른 기능 설명 ····· 6
표 4. 포트E의 다른 기능 설명 ····· 6
표 5. 포트F의 다른 기능 설명 ····· 6
표 6. 포트G의 다른 기능 설명 ····· 7
표 7. 어셈블리 명령에 쓰인 약자의 의미 ····· 28
표 8. 기본 데이터형 ····· 58
표 9. 산술연산자 ····· 60
표 10. 논리, 비교 연산자 ····· 60
표 11. 비트 연산자 ····· 60
표 12. 대입 연산자 ····· 61
표 13. 음계에 따른 tone_data 값 ····· 118
표 14. 박자에 따른 지속시간값 ····· 118
표 15. $f_1 = 500, f_{10} = 1000$일 때 k에 따른 속도 ····· 133

그림 차례

그림 1. ATmega128의 블록선도(출처:ATMEL) ····· 4
그림 2. ATmega128의 핀배치(출처:ATMEL) ····· 5
그림 3. ATmega128의 메모리 맵(출처: ATMEL) ····· 7
그림 4. 범용레지스터(출처:ATMEL) ····· 8
그림 5. XYZ레지스터(출처:ATMEL) ····· 8
그림 6. ATmega128의 I/O레지스터(출처:ATMEL) ····· 9
그림 7. ATmega128의 확장 I/O레지스터(출처:ATMEL) ····· 11
그림 8. 외부램의 연결(출처:ATMEL) ····· 13
그림 9. SRL2-SRL0를 이용한 메모리 분할(출처:ATMEL) ····· 14
그림 10. Lock bit의 구성(출처:ATMEL) ····· 15
그림 11. Lock bit설정에 따른 기능 (출처:ATMEL) ····· 15
그림 12. 부트 사이즈의 선택(출처:ATMEL) ····· 16
그림 13. ATmega128의 클록 분배(출처:ATMEL) ····· 17
그림 14. 외부 RC 발진기의 연결(출처:ATMEL사) ····· 18
그림 15. 외부 RC 발진기 동작 모드 (출처:ATMEL) ····· 18
그림 16. 외부 RC발진기의 기동시간 선택(출처:ATMEL) ····· 18
그림 17. 외부 수정 발진기의 연결(출처:ATMEL) ····· 19
그림 18. 외부 수정 발진기 동작 모드 (출처:ATMEL) ····· 19
그림 19. 수정발진기의 기동시간 선택(출처:ATMEL) ····· 19
그림 20. 저주파수정발진기의 기동시간 선택(출처:ATMEL) ····· 20
그림 21. 슬립모드의 설정(출처:ATMEL) ····· 20
그림 22. 슬립모드의 동작 요약(출처:ATMEL) ····· 21
그림 23. 리셋 관련 내부 회로 블록선도(출처:ATMEL) ····· 22
그림 24. 리셋 관련 변수(출처:ATMEL) ····· 23
그림 25. 파워온 리셋 동작 타이밍도(출처:ATMEL) ····· 23
그림 26. 외부리셋 동작 타이밍도(출처:ATMEL) ····· 24
그림 27. 저전압 검출 리셋 동작 타이밍도(출처:ATMEL) ····· 24
그림 28. 워치독 리셋 동작 타이밍도(출처:ATMEL) ····· 24
그림 29. 워치독 타이머의 구성(출처:ATMEL) ····· 25
그림 30. 워치독 타이머 프리스케일러의 설정(출처:ATMEL) ····· 26
그림 31. 워치독 타이머의 설정(출처:ATMEL) ····· 26
그림 32. 분기 명령 모음 (출처:ATMEL) ····· 28
그림 33. MCU 제어 명령 모음 (출처:ATMEL) ····· 28
그림 34. 데이터 명령 모음(출처:ATMEL) ····· 29
그림 35. 산술과 논리 연산 명령 모음(출처:ATMEL) ····· 29
그림 36. 비트 조작 명령 모음 (출처:ATMEL) ····· 30
그림 37. AVR Studio의 초기화면 ····· 33
그림 38. New Project의 실행 ····· 34
그림 39. 프로젝트의 설정 ····· 34
그림 40. 디버그 플랫폼과 디바이스의 설정 ····· 35
그림 41. 프로젝트 등록의 최종 결과로 생긴 에디터 창 ····· 35
그림 42. Build 실행 ····· 36

그림 43. 성공적으로 실행파일이 만들어진 결과 ····· 36
그림 44. 에러가 있을 때 결과 ····· 37
그림 45. [Debug->Start Debugging] 메뉴 ····· 37
그림 46. Debug 세션의 시작 ····· 38
그림 47. 디버그 세션에서 메인 메뉴 [View->Register]의 실행 ····· 38
그림 48. 디버그 세션에서 메인 메뉴 [View->Memory]의 실행 ····· 39
그림 49. 디버그 세션에서 메인 메뉴 [View->Watch]의 실행 ····· 39
그림 50. [Debug]메뉴의 하위메뉴들 ····· 40
그림 51. Atmel AVR ISP, PonyProg2000, CodevisonAVR용의 ISP 인터페이스 회로 ····· 42
그림 52. CodevisionAVR Compiler의 초기화면 ····· 43
그림 53. [File->New] 메뉴의 실행과 새로운 프로젝트 생성 ····· 44
그림 54. CodeWizardAVR 사용 여부를 설정하기 위한 창 ····· 44
그림 55. CodevisionAVR에서 새로운 프로젝트 이름의 설정 ····· 45
그림 56. CodevisionAVR에서 프로젝트 옵션 설정하기 ····· 45
그림 57. [File->New] 메뉴의 실행과 새로운 파일생성 ····· 46
그림 58. CodevisionAVR에서 새로운 파일의 작성 ····· 46
그림 59. [File->Save As] 메뉴를 통한 파일 저장 ····· 47
그림 60. [Project->Configure]메뉴의 실행 ····· 47
그림 61. Add메뉴를 통한 소스파일 등록 ····· 48
그림 62. Add메뉴를 통한 소스파일 등록 결과 ····· 48
그림 63. CodeVisionAVR의 Make 실행 ····· 49
그림 64. CodevisionAVR에서 Make의 성공적인 결과 ····· 49
그림 65. CodevisionAVR에서 파일에 에러가 있을 때 Make 결과 ····· 50
그림 66. CodeVisionAVR를 이용한 문법에러의 수정 ····· 50
그림 67. Debugger의 선택 ····· 51
그림 68. [Tools->Debugger] 메뉴 ····· 51
그림 69. AVR Studio4의 프로젝트 선택창 ····· 52
그림 70. cof파일의 선택 대화창 ····· 52
그림 71. 디버깅을 위한 프로젝트 생성 ····· 53
그림 72. CodeVisionAVR에서 AVR Studio4를 이용한 디버깅화면 ····· 53
그림 73. AVR Chip 프로그래머 설정 화면 ····· 54
그림 74. CodeVisionAVR의 칩 프로그래머 대화창 ····· 54
그림 75. 다운로드할 rom이나 hex 혹은 bin 파일의 선택 ····· 55
그림 76. 플래시롬을 지우는 과정을 보여주는 창 ····· 55
그림 77. 프로그램의 다운로드 과정을 보여주는 창 ····· 56
그림 78. I/O 포트의 기본 구조(출처:ATMEL) ····· 75
그림 79. 기본 입출력 실험을 위한 회로도 ····· 76
그림 80. 단순 키보드 스캔을 위한 회로도 ····· 79
그림 81. 딥스위치를 이용한 간단한 입력회로 ····· 80
그림 82. 인터럽트 벡터의 배치(출처:ATMEL) ····· 83
그림 83. ATmega128의 인터럽트 발생원과 벡터값(출처:ATMEL) ····· 84
그림 84. 간단한 채터링 방지 키 입력 회로 ····· 86
그림 85. 타이머0의 블록선도(출처:ATMEL) ····· 88
그림 86. 타이머0,2의 CTC모드 동작(출처:ATMEL) ····· 92
그림 87. 타이머의 고속 PWM 모드 동작(출처:ATMEL) ····· 92

그림 88. 타이머0, 2의 위상교정 PWM 모드 동작 (출처:ATMEL) ····· 93
그림 89. 타이머1,3의 불록선도(출처:ATMEL) ····· 94
그림 90. 타이머1,3의 동작 모드 설정(출처:ATMEL) ····· 96
그림 91. 타이머1,2를 이용한 출력비교 변조의 예 ····· 99
그림 92. LED배열 회로. ····· 107
그림 93. FND의 내부구성과 핀배치 ····· 109
그림 94. FND 기초 실험도(왼쪽:디코더 비사용, 오른쪽:디코더 사용) ····· 110
그림 95. FND의 동적 구동을 위한 회로도 ····· 111
그림 96. 키보드 스캔을 위한 순서도 ····· 115
그림 97. 5X5 키 매트릭스 ····· 117
그림 98. 음향 발생을 위한 회로도 ····· 119
그림 99. 16x2 문자 LCD 실험 회로도 ····· 125
그림 100. LCD에 문자 등록을 위해 패턴값 구하는 예 ····· 128
그림 101. 스텝모터의 토크 속도 특성 ····· 130
그림 102. 4상 스텝모터의 결선도 ····· 131
그림 103. 1상 여자 방법(왼쪽: sw1만 닫힌 경우, 오른쪽: sw2만 닫힌 경우) ····· 131
그림 104. 2상 여자 방법(왼쪽: sw1, sw2만 닫힌 경우, 오른쪽: sw2, sw3만 닫힌 경우) ····· 132
그림 105. 스텝모터의 가속 패턴 ····· 133
그림 106. 다목적 H브리지 회로의 예 ····· 134
그림 107. 모터 실험 회로도 ····· 135
그림 108. 직류모터의 구동원리 ····· 138
그림 109. D/A변환기의 종류. 가산형(왼쪽)과 사다리형(오른쪽) ····· 141
그림 110. D/A 변환 실험 회로도 ····· 142
그림 111. ADCH, ADCL 레지스터(출처:ATMEL) ····· 144
그림 112. ADC 채널과 Gain의 설정(출처:ATMEL) ····· 145
그림 113. 아날로그 전원 단자의 처리(출처:ATMEL) ····· 146
그림 114. 조도 측정 회로 ····· 147
그림 115. 아날로그 비교기의 블록 선도(출처:ATMEL) ····· 148
그림 116. SPI 포트의 블록선도(출처:ATMEL) ····· 150
그림 117. SPI의 동작(출처:ATMEL) ····· 151
그림 118. USART 통신의 데이터 프레임(출처:ATMEL) ····· 152
그림 119. 보레이트와 UBRR의 관계(출처:ATMEL) ····· 154
그림 120. RS232C 통신을 위한 인터페이스 ····· 155
그림 121. 직렬통신에 의한 문자열 수신 순서도 ····· 158

머리말

마이컴이란 프로그램 메모리, 데이터 메모리, 입출력 포트 등으로 구성된 작은 규모의 컴퓨터인 마이크로컴퓨터의 기능을 단일칩에 집적해 구현한 단일칩 마이크로컴퓨터를 지칭하는 말로 마이크로콘트롤러라고도 불리운다. 마이컴은 아래와 같은 특성을 갖기 때문에 모든 산업 전반에 걸쳐 다양하게 응용되고 있다.

- 소형화와 경량화 : 다양한 기능을 VLSI기술에 의해 단일 칩에 집적해 구현했기 때문에 제품 제작시 소형화와 경량화를 실현할 수 있다.
- 쉽고 편리한 개발 : 입출력, 인터럽트 처리, 비트 조작 명령어가 많아 프로그램 작성이 수월하여 개발을 빨리 할 수 있다.
- 저가격 : 개발비, 부품비와 제작비를 감소시켜 제품의 가격을 저렴하게 할 수 있다.
- 융통성 : 프로그램 변경만으로 기능을 변경하거나 확장시키는 것을 짧은 시간내에 수월하게 할 수 있다.
- 신뢰성 : 마이크로컴퓨터의 기능이 VLSI기술에 의해 단일 칩에 집적되어 있기 때문에 제품의 부품수를 줄일 수 있어 제품의 구성을 간단하게 하고 고장률을 줄일 수 있으며 고장시 유지 보수가 쉽다.

마이컴의 시초는 인텔사가 1971년 2300여개의 트랜지스터를 집적하여, 46개의 명령어, 12개의 주소선, 16개의 내부 4비트 범용레지스터를 사양으로 108kHz의 발진주파수에서 동작하도록 만든 4비트 마이컴 4004이다. 그 후 인텔은 1972년에는 4004를 개선한 8008 8비트 마이크로 프로세서를, 1973년에는 8008을 개선한 8080 8비트 프로세서를 발표하였다. 1975년에는 TI에서 중앙처리장치와, 메모리, 입출력 장치를 단일칩에 내장한 TMS1000 시리즈를 출시하여 마이컴 시대를 열었고 이후 마이크로프로세서의 개발은 연산용 프로세서와 제어용 마이컴으로 분화되었다. 1976년에는 8비트 마이컴 8048, 8748, 8035가 포함된 MCS48시리즈를 인텔사에서 출시하고 그 상위버전인 MCS51시리즈의 8051을 1981년에 출시한 이 후 여러 회사에서 계속적으로 향상된 기능의 마이컴을 출시하고 있다.

인텔의 80960, 모토로라의 68332, ST마이크로의 ST40이나 STR7, 삼성전자의 KS32 등 32비트의 고성능 마이컴도 있지만 아직 8비트 마이컴이 가전제품(TV/VCR, CD, 카메라, 리모콘, 전자레인지, 공기청정기 등) , 사무기기(모니터, 마우스, 키보드, 스캐너, 프린터, 복사기, 팩스, PC LAN시스템, 바코드리더, 탁상계산기, 하드디스크 등), 통신기기(전화기, 자동응답기, 모뎀, 발신자 확인기 등), 자동차(ABS, 속도측정기, Power seat, 운항제어, 자동경보기, 온도제어, 연료제어, Air bag 센서, Sun roof 제어 등), 계측기, CCTV, 자동차 주행 시험장의 점수 계산장치, PLC, 신용카드리더, 공정제어 등 산업 전반에 걸쳐 다양하게 응용되고 있는 형편이다. 그리고 인텔의 8051, 모토로라의 68HC11, ST마이크로의 ST7시리즈, Microchip의 PIC시리즈, Zilog의 Z88C00, Atmel의 AVR 등의 8비트 마이컴에서 Atmel의 AVR은 교육로봇 (robotkim.com), US Technology(www.us-technology.co.kr), 로보블록 (roboblock.co.kr), 마이컴월드 (www.micomworld.co.kr) 등 국내 여러 회사에서 값싼 키트를 공급하고 있으며 AVR이 플래쉬롬을 내장하여 값싸고 쉽게 프로그래밍이 가능한 이점을 갖고 있어 초보자들이 쉽게 구현해 볼 수 있으므로 여기에서는 AVR 그 가운데 ATmega128 시스템 개발의 기초와 응용을 다룬다.

1장에서는 ATmega128의 구조와 기능에 대하여 설명하고 2장에서는 AVR 시스템 개발을 위한 환경을 꾸미는 방법과 어셈블리어와 C언어에 대한 설명을 한다. 3장에서는 기초적인 프로그램과 단순 입출력을 위한 프로그램 작성과 실습을 다루고 4장에서는 타이머와 인터럽트를 사용한 프로그램 작성과 실습을 다룬다. 5장에서는 응용실험으로 마이컴 시스템의 고기능화를 위해 사용되는 FND, 키보드, D/A와 A/D 변환, 모터, 스피커, LCD 등 다양한 주변장치에 대한 설명과 이들을 이용한 응용 시스템을 위한 프로그램 작성과 실습을 다룬다.

아무쪼록 본 책이 독자들의 마이컴 시스템 개발 능력을 향상시키는데 일조하길 바랍니다.

2007년 여름 최한호

1. AVR 개요

1.1. AVR의 일반적 특징

AVR은 Atmel사의 마이컴들로 저가의 고기능 마이컴으로 다음과 같은 특성을 갖고 있다.

- 유사 RISC (Reduced Instruction Set Code)구조 : 명령어가 많고 복잡하며 주소지정방식도 다양한 CISC (Complex Instruction Set Code) 방식이 사용된 8051에 비해 명령어와 주소지정방식이 작지 않지만 유사 RISC 구조로 연산 및 전송이 레지스터간에 직접 이루어지고 명령어의 길이가 16비트 1워드로 고정적이어 디코딩이 쉬운 특징을 갖고 있으며 발진주파수를 분주해서 내부클록으로 사용하는 8051과 달리 발진주파수와 같은 내부클록을 사용하며 대부분 1기계사이클에 명령어가 실행되어 보통 1MHz당 1MIPS의 처리속도를 보인다. 결국 통상적인 RISC처럼 명령어와 주소지정방식이 작지 않아 어셈블리어 프로그램 작성에서 불리하지도 않으면서도 통상적인 RISC처럼 속도가 매우 빠른 유사 RISC의 특성을 갖고 있다.
- 하바드 구조의 메모리 구성 : Z80처럼 프로그램 메모리와 데이터 메모리가 구분되지 않은 폰노이만 방식이 아니고 TMS320과 같이 프로그램 메모리와 데이터 메모리가 분리되어 있는 하바드 구조를 채택하였다.
- 저전력 소모 : CMOS 기술을 채택하고 있어 소비전력이 매우 적고 동작전압이 1.8-5.5V로 큰 장점이 있다. 다양한 동작 모드를 제공해 저전력 동작을 지원한다.
- 큰 잡음여유와 입력 임피던스 : CMOS 기술을 채택하고 있어 잡음여유와 입력 임피던스가 큰 장점이 있으나 정전기 및 과전압에 의한 파손 위험이 큰 단점도 있다.
- 고집적 : CMOS기술에 의해 1K-256Kbyte 플래시롬 메모리와 변수 저장을 위한 EEPROM 및 SRAM이 작은 칩 하나에 내장되어 있다.
- 고속 : 유사 RISC 구조와 32개의 레지스터 사용과 고집적으로 대부분 1기계사이클에 명령어가 실행되어 보통 1MHz당 1MIPS의 처리속도를 보인다.
- 효율적인 프로그래밍 : C언어를 고려하여 설계하였으며 32개의 레지스터와 C와 유사한 주소지정 방식를 사용하고 있으며 16비트와 32비트의 산술 연산을 지원하여 효율적으로 최적의 프로그램을 가능하게 한다.
- 다양하고 값싼 개발 도구 : 공짜의 개발 도구인 AVR Studio와 값싼 평가보드나 Starter 키트가 이용가능하다.
- 다양한 사양 지원 : 8핀에서 100핀의 외형과 이에 상응하는 메모리와 기능을 갖는 다양한 시리즈의 제품이 존재하여 적절한 응용에 대응하여 제어기를 개발할 수 있다.
- ISP (In System Program) 지원 : 내부에 롬라이터와 같은 기능을 하는 부분이 내장되어 있어 내부 플래시롬을 롬라이터없이 읽고 쓰기가 가능하여 따로 값 비싼 롬라이터를 이용하지 않고도 PCB 기판 상에 마이컴을 실장하고 전원과 클럭이 공급되는 상태에서 프로그래밍이 가능한 ISP 기능이 제공된다.

1.2. AVR의 종류

Atmel사의 8비트 마이컴 AVR은 크게 Tiny, AT90, Mega시리즈로 나뉠 수 있다. 표 1은 AVR의 사양을 보여준다.

- Tiny시리즈 : 핀수가 8-24핀 정도의 작은 외형으로 대부분 외부 시스템버스가 없고 내부에 1K-2K byte 정도의 플래시 메모리를 가지고 있어 용량도 작은 편이다. UART를 지원하지 않고 RTC 타이머가 없으며 16비트 타이머가 없고 8비트 타이머만 1-2개 있는 등 기능이나 성능이 비교적 낮지만 가격이 저렴하여 소형제어기에 적당하다.
- Mega시리즈 : 28-100핀 정도의 외형을 갖고 내부에 8K-256Kbyte 정도의 플래시 메모리와 256-4K byte 정도의 EEPROM과 512-4K byte의 SRAM을 내장하고 있다. 20MHz의 클럭에서 20MIPS의 속도를 갖는 등 성능과 기능이 높으나 가격도 높다.

● AT90시리즈 : 중간정도 사양을 가진 시리즈로 AT90S의 경우에는 사양화 되었다.

표 1. AVR의 종류와 사양

Device	Flash (K)	EEPROM (K)	SRAM (byte)	M a x I/O	F.max (MHz)	Vcc (V)	16-bit Timers	8-bit Timer	PWM (채널)	UART	TWI	A/D (채널)	인터럽트
AT90PWM1	8	0.5	512	19	16	2.7-5.5	1	1	7	No	--	11	26
AT90PWM2	8	0.5	512	19	16	2.7-5.5	1	1	7	Yes	--	11	29
AT90PWM216	16	0.5	1024	19	16	2.7-5.5	1	1	10	Yes	--	11	29
AT90PWM3	8	0.5	512	27	16	2.7-5.5	1	1	10	Yes	--	11	29
AT90PWM316	16	0.5	1024	27	16	2.7-5.5	1	1	10	Yes	--	11	29
ATmega128	128	4	4096	53	16	2.7-5.5	2	2	8	2	Yes	8	34
ATmega1280	128	4	8192	86	16	1.8-5.5	4	2	16	4	Yes	--	57
ATmega1281	128	4	8192	54	16	1.8-5.5	4	2	9	2	Yes	--	48
ATmega128RZAV	128	4	8192	54	16	1.8-5.5	4	2	9	2	Yes	--	48
ATmega128RZBV	128	4	8192	86	16	1.8-5.5	4	2	16	4	Yes	--	57
ATmega16	16	0.5	1024	32	16	2.7-5.5	1	2	4	1	Yes	--	20
ATmega162	16	0.5	1024	35	16	1.8-5.5	2	2	6	2	--	--	28
ATmega164P	16	0.5	1024	32	20	1.8-5.5	1	2	6	2	Yes	8	31
ATmega165	16	0.5	1024	54	16	1.8-5.5	1	2	4	1	USI	--	23
ATmega165P	16	0.5	1024	54	16	1.8-5.5	1	2	4	1	USI	--	23
ATmega168	16	0.5	1024	23	20	1.8-5.5	1	2	6	1	Yes	6/8	26
ATmega168P	16	0.5	1024	23	20	1.8-5.5	1	2	6	1	Yes	8	26
ATmega169	16	0.5	1024	54	16	1.8-5.5	1	2	4	1	USI	--	23
ATmega169P	16	0.5	1024	54	16	1.8-5.5	1	2	4	1	USI	--	23
ATmega2560	256	4	8192	86	16	1.8-5.5	4	2	16	4	Yes	--	57
ATmega2561	256	4	8192	54	16	1.8-5.5	4	2	9	2	Yes	--	48
ATmega256RZAV	256	4	8192	54	16	1.8-5.5	4	2	9	2	Yes	--	48
ATmega256RZBV	256	4	8192	86	16	1.8-5.5	4	2	16	4	Yes	--	57
ATmega32	32	1	2048	32	16	2.7-5.5	1	2	4	1	Yes	--	19
ATmega324P	32	1	2048	32	20	1.8-5.5	1	2	6	2	Yes	--	31
ATmega325	32	1	2048	54	16	1.8-5.5	1	2	4	1	USI	--	23
ATmega3250	32	1	2048	69	16	1.8-5.5	1	2	4	1	USI	--	32
ATmega3250P	32	1	2048	69	20	1.8-5.5	1	2	4	1	USI	--	32
ATmega325P	32	1	2048	54	20	1.8-5.5	1	2	4	1	USI	--	23
ATmega328P	32	1	2048	23	20	1.8-5.5	1	2	6	1	Yes	8	26
ATmega329	32	1	2048	54	16	1.8-5.5	1	2	4	1	USI	--	25
ATmega3290	32	1	2048	69	16	1.8-5.5	1	2	4	1	USI	--	25
ATmega3290P	32	1	2048	69	20	1.8-5.5	1	2	4	1	USI	--	25
ATmega329P	32	1	2048	54	20	1.8-5.5	1	2	4	1	USI	--	25
ATmega406	40	0.5	2048	18	1	4.0-25	1	1	1	--	Yes	--	23
ATmega48	4	0.25	512	23	20	1.8-5.5	1	2	6	1	Yes	6/8	26
ATmega48P	4	0.25	512	23	20	1.8-5.5	1	2	6	1	Yes	6/8	26
ATmega64	64	2	4096	54	16	2.7-5.5	2	2	8	2	Yes	--	34
ATmega640	64	4	8192	86	16	1.8-5.5	4	2	16	4	Yes	--	57
ATmega644	64	2	4096	32	20	1.8-5.5	1	2	6	1	Yes	--	31
ATmega644P	64	2	4096	32	20	1.8-5.5	1	2	6	2	Yes	--	31
ATmega645	64	2	4096	54	16	1.8-5.5	1	2	4	1	USI	--	23
ATmega6450	64	2	4096	69	16	1.8-5.5	1	2	4	1	USI	--	32
ATmega649	64	2	4096	54	16	1.8-5.5	1	2	4	1	USI	--	25
ATmega6490	64	2	4096	69	16	1.8-5.5	1	2	4	1	USI	--	25
ATmega64RZAPV	64	2	4096	32	20	1.8-5.5	1	2	6	2	Yes	--	31
ATmega64RZAV	64	2	4096	32	20	1.8-5.5	1	2	6	1	Yes	--	31
ATmega8	8	0.5	1024	23	16	2.7-5.5	1	2	3	1	Yes	6/8	18
ATmega8515	8	0.5	512	35	16	2.7-5.5	1	1	3	1	--	--	16
ATmega8535	8	0.5	512	32	16	2.7-5.5	1	2	4	1	Yes	--	20
ATmega88	8	0.5	1024	23	20	1.8-5.5	1	2	6	1	Yes	6/8	26
ATmega88P	8	0.5	1024	23	20	1.8-5.5	1	2	6	1	Yes	8	26
ATtiny11	1	--	--	6	6	2.7-5.5	--	1	--	--	--	--	4
ATtiny12	1	0.0625	--	6	8	1.8-5.5	--	1	--	--	--	--	5
ATtiny15L	1	0.0625	--	6	1.6	2.7-5.5	--	2	1	--	--	4	8
ATtiny2313	2	0.125	128	18	20	1.8-5.5	1	1	4	1	USI	--	8
ATtiny24	2	0.125	128	12	20	1.8-5.5	1	1	4	--	USI	8	17
ATtiny25	2	0.125	128	6	20	1.8-5.5	--	2	4	--	USI	4	15
ATtiny26	2	0.125	128	16	16	2.7-5.5	--	2	2	--	USI	11	11
ATtiny261	2	0.125	128	16	20	1.8-5.5	1	2	2	--	USI	11	19
ATtiny28L	2	--	32	11	4	1.8-5.5	--	1	--	--	--	--	5
ATtiny44	4	0.25	256	12	20	1.8-5.5	1	1	4	--	USI	8	17
ATtiny45	4	0.25	256	6	20	1.8-5.5	--	2	4	--	USI	4	15
ATtiny461	4	0.25	256	16	20	1.8-5.5	1	2	2	--	USI	11	19
ATtiny84	8	0.5	512	12	20	1.8-5.5	1	1	4	--	USI	8	17
ATtiny85	8	0.5	512	6	20	1.8-5.5	--	2	4	--	USI	4	15
ATtiny861	8	0.5	512	16	20	1.8-5.5	1	2	2	--	USI	11	19

1.3. ATmega128의 특징

그림 1은 ATmega128의 블록선도로 다음과 같은 특징을 갖는다.

- 유사 RISC 구조 : 대부분 한 클럭에 동작하는 133개의 명령어, 32개의 8비트 범용 레지스터, 16MHz에서 16MIPS 의 성능, 2사이클에서 실행되는 내장 곱셈장치
- 6개의 Sleep 모드 : idle, ADC noise reduction, power save, power down, standb, extended standby 모드 지원
- 메모리 : 128K byte의 10000번까지 ISP를 이용해 쓰고 지우기 가능한 플래시 롬 내장하고 있으며 부트 코드 영역으로 사용 가능, 십만번까지 쓰고 지우기 가능한 4K바이트의 EEPROM 내장, 4K바이트의 SRAM을 내장, 최대 64K바이트의 외부 데이터 메모리 추가 가능, 소프트웨어의 보안을 위한 프로그램 잠금 기능 제공
- ALU : 32개의 범용레지스터와 직접 연결되어 수학연산, 논리연산, 비트 연산을 보통 한 개의 시스템 클럭안에서 수행한다.
- Watchdog Timer : 프로세서가 안정적으로 동작하는지 감시하는 기능을 수행한다. 프로세서에 이상이 생겨 일정 시간마다 Watchdog Timer가 리셋되지 않게 되는 경우 시스템에 인터럽트를 발생시켜 프로세서를 초기화 시키는 동작이 이루어지게 하여 안정적인 동작을 보장할 수 있다.
- 입출력 I/O : 6개의 8비트 병렬 I/O포트와 1개의 5비트 병렬 I/O포트로 총 53개의 프로그램 가능한 입출력 선을 갖고 있다.
- 타이머/카운터 : 2개의 8비트 타이머(타이머0와 2)와 2개의 16비트 타이머(타이머1와 2)를 가지고 있다. 이들은 2개의 8비트 PWM 출력, 2-16비트 PWM 출력, 출력비교 단자 등과 관련되어 동작한다.
- AD 변환기 : 8채널의 10비트 AD 변환기를 제공한다. 이들은 8개의 단일입력 혹은 7개의 차동입력 채널로 사용 가능하며, 8개중 프로그램이 가능한 입력 게인(1x, 10x, 200x)을 갖는 2개의 채널이 존재한다.
- 비교기 : 아날로그 비교기를 1개 내장하고 있다.
- 발진기 : RC 발진기를 내장하고 있으며 외부에서 크리스탈을 접속해 정확한 클럭소스를 사용할 수도 있다.
- 직렬통신포트(Serial Port) : 동시 양방향 전송이 가능한 범용 동기/비동기 직렬 통신(USART) 기능을 제공하는 직렬통신포트를 2개 존재하여 쉽게 RS232C 통신을 구현할 수 있다.
- 인터럽트 : 리셋과 8개의 외부인터럽트를 포함하여 총 35개의 인터럽트 벡터를 가지고 있다.
- JTAG 인터페이스 : JTAG 인터페이스로 플래시, EEPROM, 퓨즈, Lock 비트를 프로그램이 가능하며, 내장된 상태에서 디버그를 가능하게 한다.
- SPI (Serial Peripheral Interface) 인터페이스 : SPI는 오직 3라인을 이용한 통신 방법으로 MOSI (Master Out Slave In), MISO (Master In Slave Out), SCK (Serial ClocK) 신호를 이용한다. Motolora에서 개발되었으며 Master와 Slave가 SCK에 동기하여 데이터를 교환하는 방식으로 마이컴의 RST(리셋) 핀을 0으로 한 상태에서 앞의 세 시그널을 이용하여 메모리의 데이터를 읽고 쓰기가 가능하다. 즉 내부에 롬라이터와 같은 기능을 하는 부분이 내장되어 있어 내부 플래시롬을 롬라이터없이 읽고 쓰기가 가능하므로 SPI가 지원되면 따로 롬라이터를 이용하지 않고도 PCB 기판 상에 마이컴을 실장하고 전원과 클럭이 공급되는 상태에서 프로그래밍이 가능한 ISP(In System Programming) 기능이 제공된다.
- 동작 범위 : 정상적인 동작의 경우 5.5mA, idle모드의 경우 1.6mA, 파워 다운 모드의 경우 1마이크로A이하를 소비한다. -40도부터 85도사이에서 동작가능하고 ATmega128L의 경우 2.7-5.5V에서 최대 8MHz에서 동작 가능하고, ATmega128의 경우 4.5-5.5V에서 최대 12MHz에서 동작 가능하다.

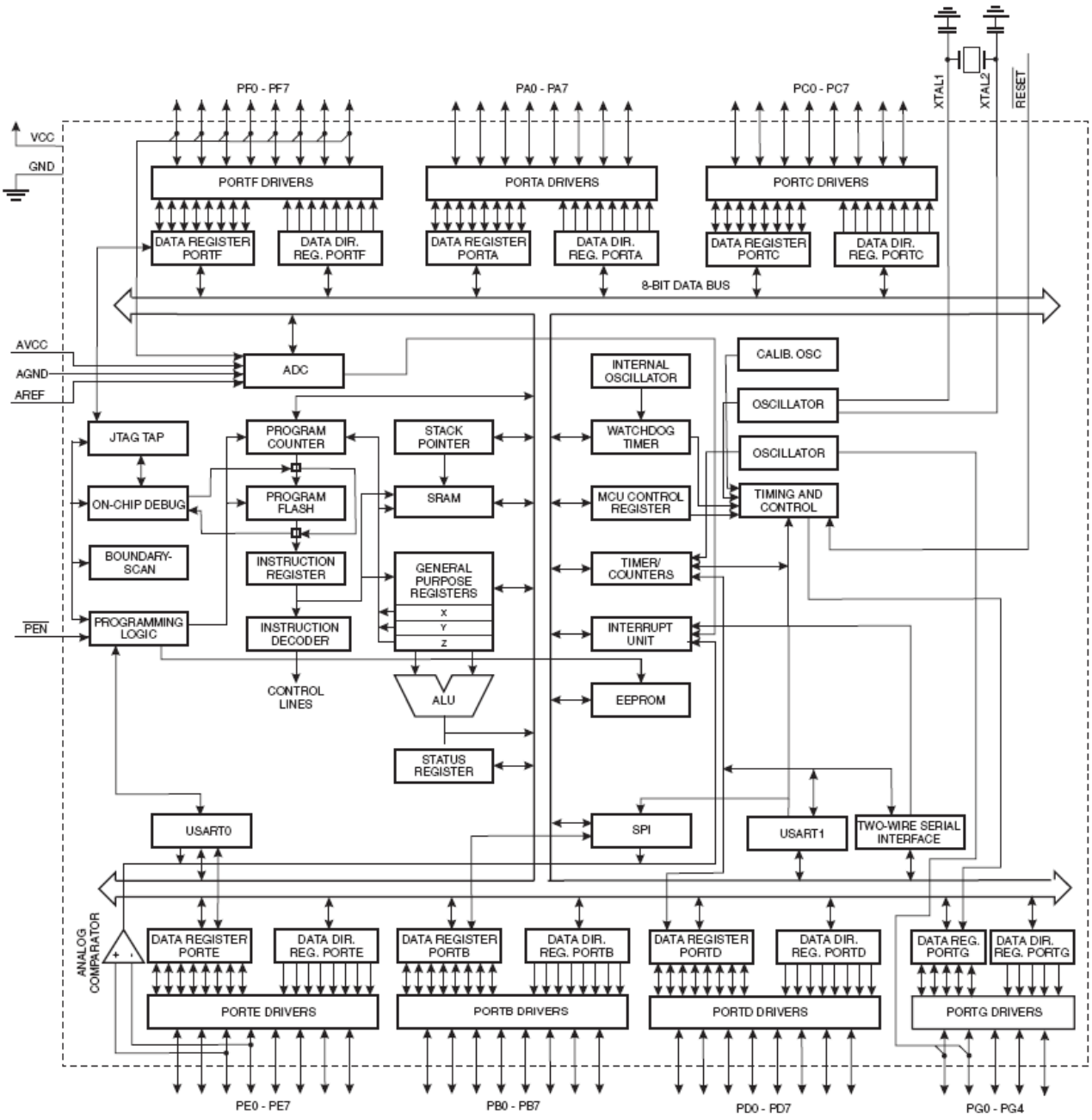


그림 1. ATmega128의 블록선도(출처:ATMEL)

1.4. 외형과 핀 기능

ATmega128은 그림 2와 같이 64핀을 갖는 TQFP(Thin Quad Flat Pack)형 또는 MLF(Micro Lead Frame)형이 있다.

- $\overline{\text{RESET}}$(핀20) : 입력단자로 1레벨이 입력되면 리셋되어 PC(Program Counter)는 일반적으로 0번지를 가르키고 0번지부터 프로그램이 시작됨(1.5.9절 참조). 리셋시 대부분의 레지스터는 0으로 된다.
- XTAL1, XTAL2(핀24,23) : 발진용 증폭기 입력 및 출력 단자.
- Vcc(핀21,51) : 전원 입력 단자.
- GND (핀22,53,63) : 그라운드 입력 단자.
- AVCC(핀64) : AD변환기 및 포트 F에 대한 공급 전압

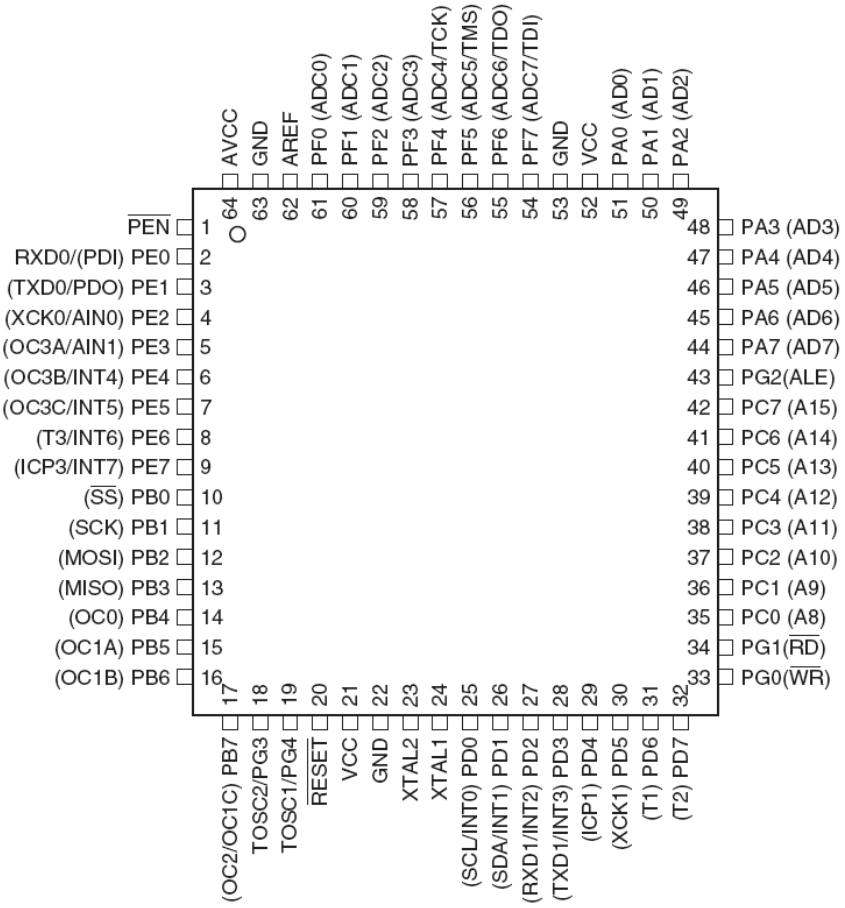


그림 2. ATmega128의 핀배치(출처:ATMEL)

- AREF(핀62) : ADC 참조 전압
- $\overline{\text{PEN}}$(핀1) : SPI를 활성화시키는 단자로 일반적인 동작모드에서는 사용하지 않고 파워 온 리셋시 0상태로 유지해 SPI를 허용하게 한다
- 포트A (PA7~PA0:핀44-51) : 내부 풀업 저항이 있는 8비트 양방향 입출력 단자. 외부메모리를 둘 경우에는 주소버스(A7-A0)와 데이터버스(D7-D0)로 사용.
- 포트B (PB7~PB0:핀10-17) : 내부 풀업 저항이 있는 8비트 양방향 입출력 단자. SPI용 단자 혹은 PWM 단자로도 사용된다(표2 참조).

표 2. 포트B의 다른 기능 설명

포트 핀	다른 특수 기능
PB7(핀17)	OC2,OC1C : 타이머2용의 비교신호 출력 혹은 타이머 1용의 비교신호C 출력
PB6(핀16)	OC1B : 타이머 1용의 비교신호B 출력
PB5(핀15)	OC1A : 타이머 1용의 비교신호A 출력
PB4(핀14)	OC0 : 타이머0 용의 비교신호 출력
PB3(핀13)	MISO : SPI 채널의 마스터 데이터 입력 혹은 슬레이브 데이터 출력 신호 단자
PB2(핀12)	MOSI : SPI 채널의 마스터 데이터 출력 혹은 슬레이브 데이터 입력 신호 단자
PB1(핀11)	SCK : SPI 채널의 마스터 클럭 출력 혹은 슬레이브 클럭 입력 신호 단자
PB0(핀10)	$\overline{\text{SS}}$: SPI 태널의 슬레이브 선택 입력 신호 단자

- 포트C (PC7~PC0:핀35-42) : 내부 풀업 저항이 있는 8비트 양방향 입출력 단자. 외부메모리를 둘 경우에는 주소버스(A15-A8)로 사용된다.

● 포트D (PD7~PD0:핀25-32) : 내부 풀업 저항이 있는 8비트 양방향 입출력 단자. 타이머용 단자 혹은 외부인터럽트용 단자로도 사용된다(표3 참조).

표 3. 포트D의 다른 기능 설명

포트 핀	다른 특수 기능
PD7(핀25)	T2 : 타이머2 클럭 입력
PD6(핀26)	T1 : 타이머1 클럭 입력
PD5(핀27)	XCK : USART1 외부클럭 입출력
PD4(핀28)	ICP1 : 타이머1 입력 캡춰
PD3(핀29)	INT3/TXD1 : 외부인터럽트 3 혹은 USART1 송신
PD2(핀30)	INT2/RXD1 : 외부인터럽트 2 혹은 USART1 수신
PD1(핀31)	INT1/SDA : 외부인터럽트 1 혹은 2선 방식의 직렬 인터페이스용 데이터 단자
PD0(핀32)	INT0/SCL : 외부인터럽트 0 혹은 2선 방식의 직렬 인터페이스용 클록 단자

● 포트E (PE7~PE0:핀2-9) : 내부 풀업 저항이 있는 8비트 양방향 입출력 단자. 타이머용 단자, 외부인터럽트, 아날로그 비교기, USART용 단자로도 사용된다(표4 참조).

표 4. 포트E의 다른 기능 설명

포트 핀	다른 특수 기능
PE7(핀2)	INT7/ICP3 : 외부인터럽트 7 혹은 타이머3 입력 캡춰
PE6(핀3)	INT6/T3 : 외부인터럽트 6 혹은 타이머3 클럭 입력
PE5(핀4)	INT5/OC3C : 외부인터럽트 5 혹은 타이머 3용의 비교신호C 출력
PE4(핀5)	INT4/OC3B : 외부인터럽트 4 혹은 타이머 3용의 비교신호B 출력
PE3(핀6)	AIN1/OC3A : 비교기 -입력 혹은 타이머 3용의 비교신호A 출력
PE2(핀7)	AIN0/XCK0 : 비교기 +입력 혹은 USART0 외부 클럭 입출력
PE1(핀8)	PDO/TXD0 : 프로그램 데이터 출력(ISP 케이블의 MISO와 연결) 혹은 USART0 송신
PE0(핀9)	PDI/RXD0 : 프로그램 데이터 입력(ISP 케이블의 MOSI와 연결) 혹은 USART0 수신

● 포트F (PF7~PF0:핀54-61) : 내부 풀업 저항이 있는 5비트 양방향 입출력 단자. AD변환기 혹은 JTAG 인터페이스용 단자로도 사용된다(표5참조).

표 5. 포트F의 다른 기능 설명

포트 핀	다른 특수 기능
PE7(핀54)	ADC7/TDI : ADC 입력채널 7 혹은 JTAG 테스트용 데이터 입력 단자
PE6(핀54)	ADC6/TDO : ADC 입력채널 6 혹은 JTAG 테스트용 데이터 출력 단자
PE5(핀54)	ADC5/TMS : ADC 입력채널 5 혹은 JTAG 테스트용 모드 선택 단자
PE4(핀54)	ADC4/TCK : ADC 입력채널 4 혹은 JTAG 테스트용 클럭 단자
PE3(핀54)	ADC3 : ADC 입력채널 3
PE2(핀54)	ADC2 : ADC 입력채널 2
PE1(핀54)	ADC1 : ADC 입력채널 1
PE0(핀54)	ADC0 : ADC 입력채널 0

● 포트G (PG4~PE0:핀19, 18, 43, 34, 33) : 내부 풀업 저항이 있는 8비트 양방향 입출력 단자. 외부 메모리 접속을 위한 스트로브 신호용, RTC(Real Time Counter) 타이머용 발진기 단자로도 사용된다(표5참조).

표 6. 포트G의 다른 기능 설명

포트 핀	다른 특수 기능
PG4(핀19)	TOSC1 : 타이머 0의 RTC 기능 사용시 클럭 발생을 위한 수정발진자 접속단자
PG3(핀18)	TOSC2 : 타이머 0의 RTC 기능 사용시 클럭 발생을 위한 수정발진자 접속단자
PG2(핀43)	ALE : 외부 메모리에 접근할 때 하위주소값을 래치하도록 신호를 출력
PG1(핀34)	$\overline{RD}$: 외부 데이터 메모리를 읽을 때 사용되는 스트로브 신호 출력 단자로 사용
PG0(핀33)	$\overline{WR}$: 외부 데이터 메모리에 쓸 때 사용되는 스트로브 신호 출력 단자로 사용

1.5. ATmega128의 메모리

ATmega128의 메모리는 프로그램 메모리와 데이터 메모리가 분리된 하바드구조로 프로그램 메모리는 128K바이트의 크기로 ISP가 가능한 플래쉬롬이 내장되어 있다(메모리맵은 그림 3의 좌측 참조). 데이터메모리는 램과 EEPROM으로 나뉠 수 있는데 그림3의 우측은 데이터 메모리의 램의 2가지 가능한 메모리 맵 구조를 보여준다.

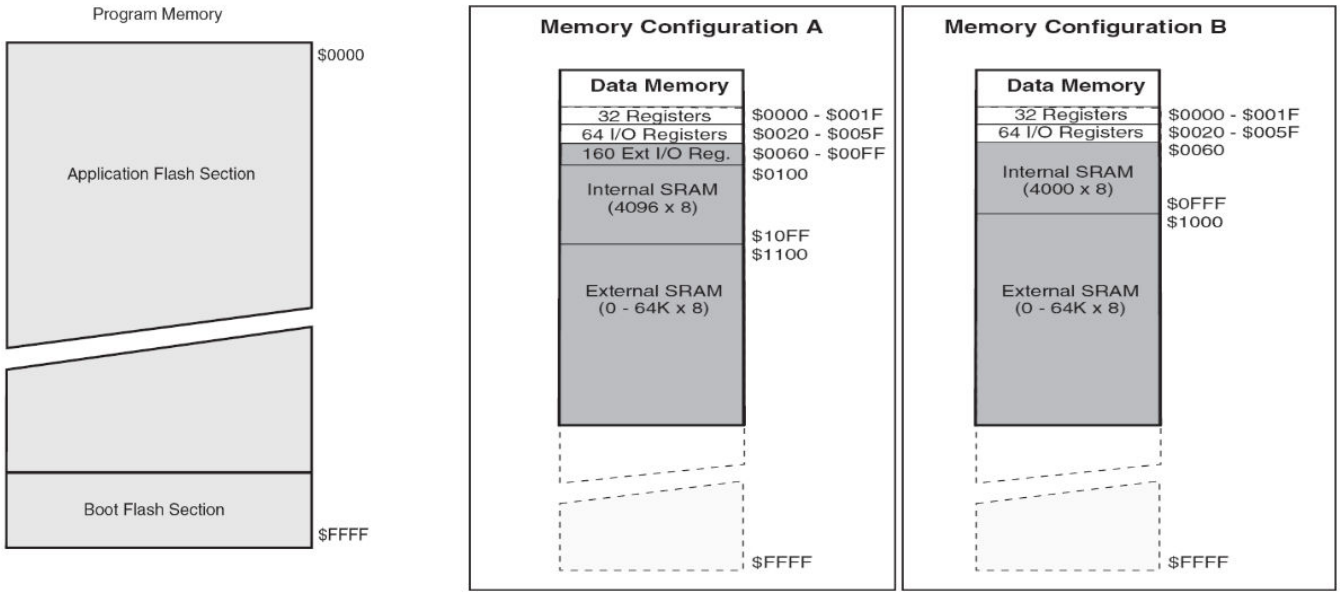


그림 3. ATmega128의 메모리 맵(출처: ATMEL)

- 플래시 프로그램 메모리 : 그림3의 좌측에 보여진 것처럼 16비트 단위로 접근 가능한 64K바이트의 주소공간(0000번지에서 FFFF번지)을 갖는 플래시롬이 프로그램 메모리로 내장되어 있다.
- 데이터 메모리 : 데이터 메모리는 크게 램과 롬으로 나뉠 수 있고 램은 그림3의 우측에 보여진 것처럼 32바이트의 범용레지스터(0000번지에서 001F번지), 64바이트의 I/O레지스터, 그리고 4K의 SRAM이 내장되어 있고 롬은 EEPROM으로 10만번 쓰기가 가능한 4K바이트가 내장되어 있다. 그림3의 우측에 보여진 것처럼 이전의 구형 ATmega103 모델과의 호환성을 유지하기 위하여 (4K - 100) 바이트의 맵을 사용하는 configuration B와 내장된 4K 바이트 전부를 사용하는 configuration A모드(노말 모드)를 지원한다. 그림3의 우측에 보여진 것처럼 60K바이트까지 외부 램의 연결이 가능하다.

1.5.1. 프로그램 메모리

- 구성 : 128K바이트의 10000번까지 쓰기가 가능한 플래시롬으로 그림3의 좌측에 보여진 것처럼 16비트 단위로 접근 가능한 64K바이트의 주소공간(0000번지에서 FFFF번지)을 갖는다. 부트로더 섹션과 응용프로그램 섹션으로 나뉠 수 있다.
- 역할 : 16비트 또는 32비트 구조로 되어 있는 각 명령어가 프로그램 메모리에 1~2개의 번지를 차지하며 저장

된다.

- 프로그래밍 : SPI 통신방식을 이용한 ISP에 의하여 프로그램을 쓰거나, JTAG 에뮬레이터를 사용하거나 병렬 프로그래밍 모드를 이용하여 프로그램을 쓸 수 있다.

1.5.2. 데이터메모리 : 범용 레지스터

- 구성 : 그림 3과 4와 같이 32바이트로 내부메모리의 00번지에서 1F번지를 차지한다.
- 역할 : 8051의 A레지스터와 같은 ALU가 따로 있지 않고 32개가 모두 ALU역할을 하여 기본적인 사칙연산을 수행하고 일부의 상수 데이터를 사용하는 연산명령은 R16-R31에서 수행한다.

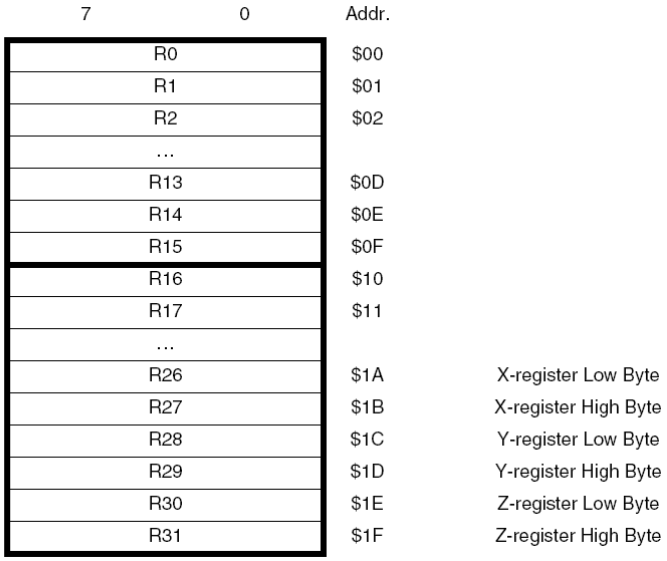


그림 4. 범용레지스터(출처:ATMEL)

1.5.2.1. X,Y,Z 레지스터

그림 4,5와 같이 R26-R31은 각각 2개씩 합쳐져서 3개의 16비트 레지스터인 X,Y,Z 레지스터로 사용될 수 있다. 이들은 데이터 메모리의 16비트 주소를 간접 지정하는 포인터로 사용되며 Z레지스터(R30, R31의 쌍)는 LPM, ELPM, SPM 명령에서 메모리값에 접근할 때 사용한다.

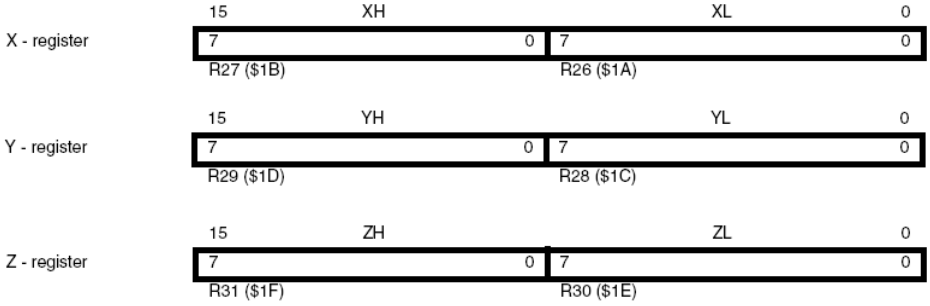


그림 5. XYZ레지스터(출처:ATMEL)

1.5.3. 데이터 메모리 : I/O 레지스터

그림 3과 6에서처럼 64바이트로 구성되며 내장된 각종 I/O 장치를 제어하기 위한 레지스터로 IN, OUT 명령을 사용하여 입출력 장치에 접근한다. 이들은 그림 6의 괄호에 표시한 것처럼 0x20번지에서 0x5F번지까지 존재하지만 IN, OUT 명령을 사용할 때는 그림 6에 표시한 것처럼 0x00에서 0x3F로 지칭해야 한다. 0x00(0x20)번지에서 0x1F(0x3F)번지에 위치하는 32개의 레지스터는 CBI, SBI, SBIC, SBIS 명령을 사용하여 비트 어드레싱이 가능하다.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C
$3E ($5E)	SPH	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
$3C ($5C)	XDIV	XDIVEN	XDIV6	XDIV5	XDIV4	XDIV3	XDIV2	XDIV1	XDIV0
$3B ($5B)	RAMPZ	–	–	–	–	–	–	–	RAMPZ0
$3A ($5A)	EICRB	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40
$39 ($59)	EIMSK	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0
$38 ($58)	EIFR	INTF7	INTF6	INTF5	INTF4	INTF3	INTF	INTF1	INTF0
$37 ($57)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
$36 ($56)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
$35 ($55)	MCUCR	SRE	SRW10	SE	SM1	SM0	SM2	IVSEL	IVCE
$34 ($54)	MCUCSR	JTD	–	–	JTRF	WDRF	BORF	EXTRF	PORF
$33 ($53)	TCCR0	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
$32 ($52)	TCNT0	Timer/Counter0 (8 Bit)							
$31 ($51)	OCR0	Timer/Counter0 Output Compare Register							
$30 ($50)	ASSR	–	–	–	–	AS0	TCN0UB	OCR0UB	TCR0UB
$2F ($4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10
$2E ($4E)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10
$2D ($4D)	TCNT1H	Timer/Counter1 – Counter Register High Byte							
$2C ($4C)	TCNT1L	Timer/Counter1 – Counter Register Low Byte							
$2B ($4B)	OCR1AH	Timer/Counter1 – Output Compare Register A High Byte							
$2A ($4A)	OCR1AL	Timer/Counter1 – Output Compare Register A Low Byte							
$29 ($49)	OCR1BH	Timer/Counter1 – Output Compare Register B High Byte							
$28 ($48)	OCR1BL	Timer/Counter1 – Output Compare Register B Low Byte							
$27 ($47)	ICR1H	Timer/Counter1 – Input Capture Register High Byte							
$26 ($46)	ICR1L	Timer/Counter1 – Input Capture Register Low Byte							
$25 ($45)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20
$24 ($44)	TCNT2	Timer/Counter2 (8 Bit)							
$23 ($43)	OCR2	Timer/Counter2 Output Compare Register							
$22 ($42)	OCDR	IDRD/OCDR7	OCDR6	OCDR5	OCDR4	OCDR3	OCDR2	OCDR1	OCDR0
$21 ($41)	WDTCR	–	–	–	WDCE	WDE	WDP2	WDP1	WDP0
$20 ($40)	SFIOR	TSM	–	–	–	ACME	PUD	PSR0	PSR321
$1F ($3F)	EEARH	–	–	–	–	EEPROM Address Register High			
$1E ($3E)	EEARL	EEPROM Address Register Low Byte							
$1D ($3D)	EEDR	EEPROM Data Register							
$1C ($3C)	EECR	–	–	–	–	EERIE	EEMWE	EEWE	EERE
$1B ($3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
$1A ($3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
$19 ($39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
$15 ($35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
$14 ($34)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
$13 ($33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
$12 ($32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
$11 ($31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
$10 ($30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
$0F ($2F)	SPDR	SPI Data Register							
$0E ($2E)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X
$0D ($2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
$0C ($2C)	UDR0	USART0 I/O Data Register							
$0B ($2B)	UCSR0A	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0
$0A ($2A)	UCSR0B	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80
$09 ($29)	UBRR0L	USART0 Baud Rate Register Low							
$08 ($28)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
$07 ($27)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
$06 ($26)	ADCSRA	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0
$05 ($25)	ADCH	ADC Data Register High Byte							
$04 ($24)	ADCL	ADC Data Register Low byte							
$03 ($23)	PORTE	PORTE7	PORTE6	PORTE5	PORTE4	PORTE3	PORTE2	PORTE1	PORTE0
$02 ($22)	DDRE	DDE7	DDE6	DDE5	DDE4	DDE3	DDE2	DDE1	DDE0
$01 ($21)	PINE	PINE7	PINE6	PINE5	PINE4	PINE3	PINE2	PINE1	PINE0
$00 ($20)	PINF	PINF7	PINF6	PINF5	PINF4	PINF3	PINF2	PINF1	PINF0

그림 6. ATmega128의 I/O레지스터(출처:ATMEL)

1.5.3.1. 상태레지스터(SREG)

상태레지스터(SREG:Status REGister)는 ALU의 연산 후 상태와 결과를 표시하는 레지스터로 8051의 PSW에 해당한다.

- 비트7(I : Global Interrupt Enable) : 전체 인터럽트를 허용하도록 설정하는 비트로 SEI 및 CLI 명령으로 이 비트를 제어할 수 있다.인터럽트 처리가 시작되면 자동적으로 클리어되고 RETI명령을 만나면 원래 상태로 복구된다. 개별 인터럽트의 허용은 인터럽트 마스크 레지스터로 설정한다.
- 비트6(T : Bit Copy Storage) : BLD, BST 명령을 사용하여 어느 레지스터의 한 비트값의 복사가 가능하다.
- 비트5(H : Half Carry Flag) : 산술연산의 가감산에서 비트3에서 올림수가 발생하면 1로 세트. BCD 연산에 사용하며 8051 PSW레지스터의 AC에 해당한다.
- 비트4(S : Sign Bit) : 플랙 N과 V의 XOR(eXclusive OR)값으로 정수들의 크기를 판단할 때 사용한다.
- 비트3(V : 2's Complement Overflow Flag) : 2의 보수 연산에서 오버플로를 표시한다.
- 비트2(N : Negative Flag) : 연산 결과값의 최상위 비트가 1로되어 2의 보수표현을 사용하는 경우 연산 결과가 음수임을 표시한다.
- 비트1(Z : Zero Flag) : 연산 결과값이 0이 되었음을 표시한다.
- 비트0(C : Carry Flag) : 연산으로 자리올림이나 자리내림이 발생하면 1로 세트된다. 8051 PSW레지스터의 CY에 해당한다.

1.5.3.2. 스택 포인터 (SP)

- 서브루틴이나 인터럽트 호출시 이들을 처리하고 다시 본래 위치로 되돌아오기 위한 복귀주소인 프로그램카운터(PC)값의 SRAM내의 스택 위치, push 또는 pop할 SRAM내의 스택 위치를 표시한다. C프로그램에서 지역변수를 저장하거나 어셈블리 프로그램에서 임시 데이터를 저장하는 용도로도 사용한다.
- 초기값은 0x0000인데 사용자 프로그램에서 SP의 초기값은 적어도 0x0100번지 이상의 값으로 설정한다.
- PC값이 저장될 때는 SP값이 1만큼 증가하고 증가된 SP값이 가리키는 내부램에 하위바이트(PCL)가 먼저 저장되고 다시 SP값이 1만큼 증가하고 증가된 SP값이 가리키는 내부램에 상위바이트(PCH)가 저장된다.
- push와 pop은 1바이트 데이터만이 대상이 되는데 push 명령이 내려지면 1바이트 데이터를 저장하기 전에 먼저 SP값이 1만큼 증가하여 증가된 SP값이 가리키는 SRAM내에 데이터가 저장되고 pop 명령이 내려지면 현재 SP값이 가리키는 내부램에서 1바이트 데이터를 꺼낸 후에 SP값이 1만큼 감소한다.

1.5.3.3. RAMPZ(RAM Page Z select) 레지스터

- Z레지스터(R30, R31의 쌍)로 LPM, ELPM, SPM 명령을 사용해 메모리값에 접근할 때 RAMPZ0=1로 하면 0x8000 에서 0xffff에 접근하고 RAMPZ0=0로 하면 0x0000 에서 0x7fff에 접근할 수 있다.

1.5.4. 데이터 메모리 : 확장 I/O 레지스터

ATmega128에 추가된 각종 I/O를 제어하기 위한 레지스터로 160바이트(0x60에서 0xff번지)로 I/O레지스터처럼 IN, OUT 명령을 사용할 수 없고 LD, LDS, LDD, ST, STS, STD 명령어로 접근해야 한다(그림7 참조).

1.5.5. 데이터 메모리 : 내부 SRAM

- 구성 : ATmega128은 4K의 SRAM이 내장되어 있다. 그림3의 우측에 보여진 것처럼 이전의 구형 ATmega103 모델과의 호환성을 유지하기 위하여 (4K - 100) 바이트의 맵을 사용하는 configuration B와 내장된 4K 바이트 전부를 사용하는 configuration A모드(노말모드)를 지원한다. 노말모드의 경우 0x0100에서 0x10ff번지를 점유한다.
- 접근법 : LS, LDS, LDD 또는 ST, STS, STD의 명령을 사용하여 16비트 직접 데이터에 의한 번지값을 지정하여 접근하거나 X,Y,Z 레지스터를 사용하여 간접지정으로 접근한다.
- 역할 : 프로그램에서 사용되는 사용자 변수의 저장영역이나 스택영역으로 사용된다.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
($FF)	Reserved	–	–	–	–	–	–	–	–
..	Reserved	–	–	–	–	–	–	–	–
($9E)	Reserved	–	–	–	–	–	–	–	–
($9D)	UCSR1C	–	UMSEL1	UPM11	UPM10	USBS1	UCSZ11	UCSZ10	UCPOL1
($9C)	UDR1	USART1 I/O Data Register							
($9B)	UCSR1A	RXC1	TXC1	UDRE1	FE1	DOR1	UPE1	U2X1	MPCM1
($9A)	UCSR1B	RXCIE1	TXCIE1	UDRIE1	RXEN1	TXEN1	UCSZ12	RXB81	TXB81
($99)	UBRR1L	USART1 Baud Rate Register Low							
($98)	UBRR1H	–	–	–	–	USART1 Baud Rate Register High			
($97)	Reserved	–	–	–	–	–	–	–	–
($96)	Reserved	–	–	–	–	–	–	–	–
($95)	UCSR0C	–	UMSEL0	UPM01	UPM00	USBS0	UCSZ01	UCSZ00	UCPOL0
($94)	Reserved	–	–	–	–	–	–	–	–
($93)	Reserved	–	–	–	–	–	–	–	–
($92)	Reserved	–	–	–	–	–	–	–	–
($91)	Reserved	–	–	–	–	–	–	–	–
($90)	UBRR0H	–	–	–	–	USART0 Baud Rate Register High			
($8F)	Reserved	–	–	–	–	–	–	–	–
($8E)	Reserved	–	–	–	–	–	–	–	–
($8D)	Reserved	–	–	–	–	–	–	–	–
($8C)	TCCR3C	FOC3A	FOC3B	FOC3C	–	–	–	–	–
($8B)	TCCR3A	COM3A1	COM3A0	COM3B1	COM3B0	COM3C1	COM3C0	WGM31	WGM30
($8A)	TCCR3B	ICNC3	ICES3	–	WGM33	WGM32	CS32	CS31	CS30
($89)	TCNT3H	Timer/Counter3 – Counter Register High Byte							
($88)	TCNT3L	Timer/Counter3 – Counter Register Low Byte							
($87)	OCR3AH	Timer/Counter3 – Output Compare Register A High Byte							
($86)	OCR3AL	Timer/Counter3 – Output Compare Register A Low Byte							
($85)	OCR3BH	Timer/Counter3 – Output Compare Register B High Byte							
($84)	OCR3BL	Timer/Counter3 – Output Compare Register B Low Byte							
($83)	OCR3CH	Timer/Counter3 – Output Compare Register C High Byte							
($82)	OCR3CL	Timer/Counter3 – Output Compare Register C Low Byte							
($81)	ICR3H	Timer/Counter3 – Input Capture Register High Byte							
($80)	ICR3L	Timer/Counter3 – Input Capture Register Low Byte							
($7F)	Reserved	–	–	–	–	–	–	–	–
($7E)	Reserved	–	–	–	–	–	–	–	–
($7D)	ETIMSK	–	–	TICIE3	OCIE3A	OCIE3B	TOIE3	OCIE3C	OCIE1C
($7C)	ETIFR	–	–	ICF3	OCF3A	OCF3B	TOV3	OCF3C	OCF1C
($7B)	Reserved	–	–	–	–	–	–	–	–
($7A)	TCCR1C	FOC1A	FOC1B	FOC1C	–	–	–	–	–
($79)	OCR1CH	Timer/Counter1 – Output Compare Register C High Byte							
($78)	OCR1CL	Timer/Counter1 – Output Compare Register C Low Byte							
($77)	Reserved	–	–	–	–	–	–	–	–
($76)	Reserved	–	–	–	–	–	–	–	–
($75)	Reserved	–	–	–	–	–	–	–	–
($74)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
($73)	TWDR	Two-wire Serial Interface Data Register							
($72)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
($71)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0
($70)	TWBR	Two-wire Serial Interface Bit Rate Register							
($6F)	OSCCAL	Oscillator Calibration Register							
($6E)	Reserved	–	–	–	–	–	–	–	–
($6D)	XMCRA	–	SRL2	SRL1	SRL0	SRW01	SRW00	SRW11	
($6C)	XMCRB	XMBK	–	–	–	–	XMM2	XMM1	XMM0
($6B)	Reserved	–	–	–	–	–	–	–	–
($6A)	EICRA	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00
($69)	Reserved	–	–	–	–	–	–	–	–
($68)	SPMCSR	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN
($67)	Reserved	–	–	–	–	–	–	–	–
($66)	Reserved	–	–	–	–	–	–	–	–
($65)	PORTG	–	–	–	PORTG4	PORTG3	PORTG2	PORTG1	PORTG0
($64)	DDRG	–	–	–	DDG4	DDG3	DDG2	DDG1	DDG0
($63)	PING	–	–	–	PING4	PING3	PING2	PING1	PING0
($62)	PORTF	PORTF7	PORTF6	PORTF5	PORTF4	PORTF3	PORTF2	PORTF1	PORTF0
($61)	DDRF	DDF7	DDF6	DDF5	DDF4	DDF3	DDF2	DDF1	DDF0
($60)	Reserved	–	–	–	–	–	–	–	–

그림 7. ATmega128의 확장 I/O레지스터(출처:ATMEL)

1.5.6. 데이터 메모리 : EEPROM

- ATmega128은 EEPROM으로 10만번 쓰기가 가능한 4K바이트가 내장되어 있는데 내부 램, I/O, 확장 I/O 레지스터와는 별개의 주소가 할당되어 있어 EEPROM용 주소 레지스터(EEARH, EEARL), 데이터레지스터(EEDR),

제어 레지스터(EECR)를 이용해 접근한다.

1.5.6.1. EEAR 레지스터

- EEAR의 비트12에서 비트 15 : 항상 0으로 읽히며 향후 버젼을 위해 유보되어 있는 비트들이다.
- EEAR의 비트11(EEAR11)에서 비트0(EEAR0) : 읽고 쓸 EEPROM의 4K바이트의 주소 지정에 사용된다. 초기값은 부여되지 않고 임의의 값이 될 수 있다. 항상 EEPROM에 접근하기 전에 주소값을 지정하도록 한다.

1.5.6.2. EEDR 레지스터

- 쓰기의 경우에는 EEAR의 주소에 쓰여질 값을 임시로 저장하거나 읽기의 경우에는 읽힌 값을 저장한다.
- 초기값은 0x00이다.

1.5.6.3. EECR 레지스터

- 비트7-비트4 : 유보된 비트들이다.
- 비트3(EERIE : EEPROM Ready Interrupt Enable) : 1로 설정되면 EEWE 비트가 0이 될때 EEPROM Ready 인터럽트를 발생하는 기능을 가진다.
- 비트2(EEMWE : EEPROM Master Write Enable) : EEWE를 1로 설정할때 EEPROM에 쓰기가 가능하도록 허용할지 여부를 결정한다.
- 비트1(EEWE : EEPROM Write Enable) : EEMWE가 1로 설정되면 EEWE가 1로 되고 나서 4클록 사이클 이내에 EEDR의 데이터를 EEAR이 가리키는 EEPROM의 주소에 쓴다. 쓰기가 완료되면 EEMWE는 자동적으로 0이 된다.
- 비트0(EERE : EEPROM Read Enable) : EEPROM의 읽기 스트로브로서 EEAR 레지스터에 적절한 주소값이 부여된 경우 EERE를 1로 해야 EEPROM 읽기가 시작될 수 있다.

1.5.6.4. EEPROM 쓰기과정

- 다음의 과정을 거쳐 쓰기가 이루어지는데 쓰기가 수행되고 나면 CPU는 다음 명령을 실행하기 전에 2클럭 동안 멈추고 EEPROM 쓰기는 CPU가 플래시메모리에 쓰기를 할 때는 불가능하다.

  ①EEWE가 0일 될 때까지 기다린다.

  ②SPMCSR 레지스터의 SPMEN비트가 0이 될 때까지 기다린다.

  ③쓰기할 EEPROM주소를 EEAR레지스터에 저장한다.

  ④쓰기할 EEPROM 데이터를 EEDR레지스터에 저장한다.

  ⑤EECR레지스터의 EEMWE는 1로 EEWE는 0으로 설정한다.

  ⑥4클럭 사이클 이내에 EEWE를 1로 설정한다. 그리고 쓰기가 수행된다.

- 위의 쓰기 과정에서 ③,④는 순서가 바뀌어도 상관없다. ⑤, ⑥ 과정중에 인터럽트가 발생되면 EEMWE의 동작 제한시간이 초과되어 쓰기 동작이 실패하게 되므로 이를 방지하기 위해 ③-⑥ 과정동안에는 상태레지스터 SREG의 I비트를 0으로 설정하여 인터럽트 발생을 금지시켜야 한다.
- EEPROM 쓰기 과정 중에 파워다운 슬립모드에 진입하면 파원다운 슬립모드에 제대로 진입하지 못하게 되므로 파워다운 모드에 진입하려면 EEPROM 쓰기가 완료되었는지를 확인해야 한다.
- 전원 전압이 낮은 동안에는 EEPROM 데이터가 손상될 수 있다. 이를 방지하기 위해 내부 Brown-out 감지기를 작동하게 하고 전원전압이 낮아지면 리셋되도록 하드웨어를 구성하는 것도 괜찮다.

1.5.6.5. EEPROM 읽기과정

①EEWE가 0일 될 때까지 기다린다.

②읽기할 EEPROM주소를 EEAR레지스터에 저장한다.

③EECR레지스터의 EERE를 1로 설정한다.

④EEPROM 데이터를 EEDR레지스터로부터 읽는다.

1.5.7. 외부 데이터 메모리

ATmega128은 노말모드에서 0x1100-0xffff번지에 외부 데이터 메모리로 사용가능하며 이들은 외부 램, 외부 플래쉬 롬 또는 LCD나 AD변환기와 같은 주변장치의 인터페이스용으로 사용할 수 있다. 이를 위해 다음과 같은 기능이 제공된다.

① 주변장치와 적절한 인터페이스를 위한 0-3의 대기 사이클을 지정할 수 있다.

② 2개의 섹터로 외부 데이터 메모리를 분할하고 이들에 독립적인 대기 사이클을 지정할 수 있다.

③ 16비트 주소의 상위바이트에 중 필요한 갯수의 비트만을 주소 버스로 동작하게 할 수 있다.

④ 데이터 버스의 신호들이 동작할 때 전류 소비량이 감소되도록 Bus-keeper 기능을 설정할 수 있다.

1.5.7.1. 외부 메모리 인터페이스

- 외부 램 연결을 위한 핀 : MCUSR레지스터를 이용해서 설정한다.

① PA7~PA0(핀44~핀51) : 8051의 포트0처럼 하위주소와 데이터버스로 사용된다.

② PC7~PC0(핀35~핀42) : 8051의 포트2처럼 상위주소버스로 사용된다.

③ ALE(Address Latch Enable, PG2, 핀43) : 외부메모리에 접근할 때 PA에서 출력되는 하위주소값을 래치할 수 있도록 CPU에서 1레벨값을 출력해준다.

④ $\overline{RD}$(PG1, 핀34) : 외부 데이터 메모리를 읽을 때 사용되는 스트로브 신호 출력 단자로 사용

⑤ $\overline{WR}$ (PG0, 핀33) : 외부 데이터 메모리에 쓸 때 사용되는 스트로브 신호 출력 단자로 사용

- 외부램 연결 블록선도 : 그림 8은 외부램과의 연결을 보여준다.

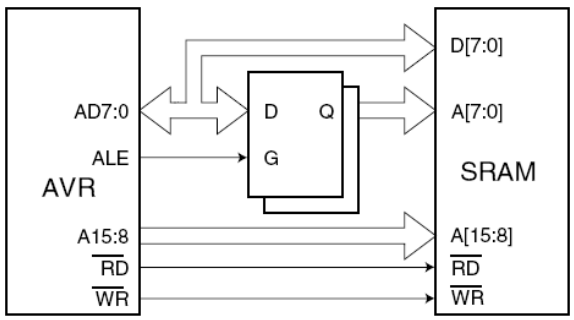


그림 8. 외부램의 연결(출처:ATMEL)

- 외부램 읽기(쓰기) 과정 :

①외부 램 읽기 명령이 내려지면 ALE신호가 1레벨이 되고 Port A(AD7:0)는 하위주소 버스로 하위주소 값을 내보내고 Port C(A15:8(는 상위주소 버스로 페이지 값을 출력시킨다.

②ALE 가 1레벨인 동안 하위주소값은 래치되어 ALE가 0레벨이 된 후에도 래치에 값이 보존된다.

③읽기신호 $\overline{RD}$ (쓰기신호 $\overline{WR}$)가 0레벨로 변하여 활성상태로 되면

④외부 램의 Output Enable(혹은 Write Enable)이 활성화되고 외부 램의 데이터 값이 Port A를 통해 CPU로 읽혀지도록(램에 쓰여지도록) 한다. (외부램과의 인터페이스는 기본적으로 3클럭 사이클에 수행되나 소프트웨어적으로 1-3개의 여분의 대기 사이클을 줄 수도 있다.)

1.5.7.2. MCUCR레지스터

MCUCR(MCU Control Register)레지스터는 CPU의 전체적인 기능을 설정하는데 비트7과 6을 사용해서 외부 데이터 메모리 영역을 설정할 수 있다.

- 비트7(SRE:external SRAM/XMEM Enable) : 1로 세트하면 PA7-PA0(핀44-핀51), PC7-PC0(핀35-핀42), ALE(PG2, 핀43), $\overline{RD}$(PG1, 핀34), $\overline{WR}$(PG0, 핀33) 핀들을 외부메모리와 인터페이스를 위한 핀들로 설정하게 되고 0으로 하면 병렬 포트로 동작하도록 한다.
- 비트6(SRW10:Wait-state Select Bit for Upper Sector) : 외부 메모리와 인터페이스 할 때 외부 데이터의 Upper 섹터에 부여하는 여분의 대기 사이클의 수를 XMCRA레지스터의 SRW11비트와 조합되어 설정한다. [SRW11:SRW10]=00일때 대기 사이클은 0개, 01일때 1개, 10일때 2개, 11일때 3개를 준다.

1.5.7.3. XMCRA레지스터

XMCRA(eXternal Memory Control Register A)레지스터는 외부 데이터 메모리 영역을 분할하거나 대기 사이클의 수를 설정할 수 있다.

- 비트6(SRL2)~비트4(SRL0) : 외부 데이터 메모리 영역을 Lower와 Upper 2개의 섹터로 분할하여 각 섹터에 별도로 대기 사이클의 수를 설정할 수 있도록 한다.

SRL2	SRL1	SRL0	Sector Limits
0	0	0	Lower sector = N/A Upper sector = 0x1100 - 0xFFFF
0	0	1	Lower sector = 0x1100 - 0x1FFF Upper sector = 0x2000 - 0xFFFF
0	1	0	Lower sector = 0x1100 - 0x3FFF Upper sector = 0x4000 - 0xFFFF
0	1	1	Lower sector = 0x1100 - 0x5FFF Upper sector = 0x6000 - 0xFFFF
1	0	0	Lower sector = 0x1100 - 0x7FFF Upper sector = 0x8000 - 0xFFFF
1	0	1	Lower sector = 0x1100 - 0x9FFF Upper sector = 0xA000 - 0xFFFF
1	1	0	Lower sector = 0x1100 - 0xBFFF Upper sector = 0xC000 - 0xFFFF
1	1	1	Lower sector = 0x1100 - 0xDFFF Upper sector = 0xE000 - 0xFFFF

그림 9. SRL2-SRL0를 이용한 메모리 분할(출처:ATMEL)

- 비트3~비트2(SRW01~SRW00:Wait-state Select Bit for Lower Sector) : 외부 메모리와 인터페이스 할 때 외부 데이터의 Lower 섹터에 부여하는 여분의 대기 사이클의 수를 설정한다. [SRW01:SRW00]=00일때 대기 사이클은 0개, 01일때 1개, 10일때 2개, 11일때 3개를 준다.
- 비트1(SRW11:Wait-state Select Bit for Upper Sector) : 외부 메모리와 인터페이스 할 때 외부 데이터의 Upper 섹터에 부여하는 여분의 대기 사이클의 수를 XMCRA레지스터의 SRW11비트와 조합되어 설정한다. [SRW11:SRW10]=00일때 대기 사이클은 0개, 01일때 1개, 10일때 2개, 11일때 3개를 준다.

1.5.7.4. XMCRB레지스터

XMCRB(External Memory Control Register B)레지스터는 버스 키퍼 기능을 설정하고 외부 메모리 주소의 상위 바이트에서 어느 부분까지가 주소지정에 사용되는지를 설정한다.

- 비트7(XMBK:eXternal Memory Bus-Keeper enable) : 1로 세팅되면 PA7-PA0(핀44-핀51)의 값들이 보존되는 버스 키퍼 기능을 활성화시킨다.
- 비트6~비트4 : 유보된 비트들이다.
- 비트2~비트0(XMM2~0 : eXternal Memory high Mask) : 60K이하의 외부 메모리를 사용하는 경우 상위주소 버스로 사용되는 port C의 일부분만이 주소버스로 사용되도록 하고 나머지는 입출력 포트로 설정하는데 사용된

다. 만약 [XMM2:XMM1:XMM0]=111로 하면 포트 C 모두를 입출력 포트로 설정하고, 110이면 PC0와 PC1만을 주소버스로 설정하고, 101이면 PC0~PC2, 100이면 PC0~PC3, 011이면 PC0~PC4, 010이면 PC0~PC5, 001이면 PC0~PC6을 주소버스로 설정하고, 000이면 포트 C 모두를 주소버스로 설정한다.

1.5.8. 메모리 lock 비트

메모리의 보호를 위해 설정하는데 사용하는 1바이트 구조의 비트로 디폴트로 비트값이 1로 되어 있고 프로그램하면 0으로 설정되고 chip erase명령에 의해 1로 환원된다. 그림 10은 lock비트의 구성을 보여준다. 그리고 그림 11은 lock비트 값에 따른 모드 설정값과 각 모드에 따른 메모리 보호기능을 보여준다.

Lock Bit Byte	Bit No.	Description	Default Value
	7	–	1 (unprogrammed)
	6	–	1 (unprogrammed)
BLB12	5	Boot lock bit	1 (unprogrammed)
BLB11	4	Boot lock bit	1 (unprogrammed)
BLB02	3	Boot lock bit	1 (unprogrammed)
BLB01	2	Boot lock bit	1 (unprogrammed)
LB2	1	Lock bit	1 (unprogrammed)
LB1	0	Lock bit	1 (unprogrammed)

그림 10. Lock bit의 구성(출처:ATMEL)

Memory Lock Bits			Protection Type
LB mode	LB2	LB1	
1	1	1	No memory lock features enabled.
2	1	0	Further programming of the Flash and EEPROM is disabled in Parallel and SPI/JTAG Serial Programming mode. The Fuse bits are locked in both Serial and Parallel Programming mode.(1)
3	0	0	Further programming and verification of the Flash and EEPROM is disabled in Parallel and SPI/JTAG Serial Programming mode. The Fuse bits are locked in both Serial and Parallel Programming mode.(1)
BLB0 mode	BLB02	BLB01	
1	1	1	No restrictions for SPM or (E)LPM accessing the Application section.
2	1	0	SPM is not allowed to write to the Application section.
3	0	0	SPM is not allowed to write to the Application section, and (E)LPM executing from the Boot Loader section is not allowed to read from the Application section. If interrupt vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section.
4	0	1	(E)LPM executing from the Boot Loader section is not allowed to read from the Application section. If interrupt vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section.
BLB1 mode	BLB12	BLB11	
1	1	1	No restrictions for SPM or (E)LPM accessing the Boot Loader section.
2	1	0	SPM is not allowed to write to the Boot Loader section.
3	0	0	SPM is not allowed to write to the Boot Loader section, and (E)LPM executing from the Application section is not allowed to read from the Boot Loader section. If interrupt vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section.
4	0	1	(E)LPM executing from the Application section is not allowed to read from the Boot Loader section. If interrupt vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section.

그림 11. Lock bit설정에 따른 기능 (출처:ATMEL)

1.5.9. 퓨즈 비트

ATmega128의 기본적인 설정용으로 사용되며 Extended Fuse, Fuse High, Fuse Low바이트로 모두 3개의 바이트로 구성되어 있다. 디폴트로 비트값이 1로 되어 있고 프로그램하면 0으로 설정되고 chip erase명령에 의해 영향을 받지 않으므로 메모리 Lock비트의 LB1을 0으로 하여 퓨즈 비트를 변경할 수 없도록 할 수 있다. 퓨즈 비트를 먼저 설정하고 메모리 Lock비트를 설정해야 한다.

1.5.9.1. Extended Fuse Byte

비트1~0만 사용하여 ATnega103과 호환모드의 설정, Watchdog 타이머 동작의 설정에 사용한다.

- 비트1(M103C) : 디폴트 상태로는 0으로 설정되어 ATmega13과 호환모드로 되어 있다. 1로 프로그램하면 노말모드로 ATmega128의 고유기능을 전부 활용할 수 있다.
- 비트0(WDTON: WatchDog Timer ON) : 디폴트 상태로는 1로 설정되어 워치독 타이머 기능이 동작하도록 한다.

1.5.9.2. Fuse High Byte

- 비트7(OCDEN:On Chip Debug eNable) : 디폴트 상태로는 1로 설정되어 On chip debug를 허용한다.
- 비트6(JTAGEN : JTAG ENable) : 디폴트 상태로는 1로 설정되어 JTAG를 허용한다.
- 비트5(SPIEN : SPI Enable) : 디폴트 상태로는 1로 설정되어 SPI를 통한 직렬 프로그래밍을 허용한다.
- 비트4(CKOPT) : 클럭 옵션을 설정한다(1.6절 참조).
- 비트3(EESAVE) : 디폴트 상태로는 1로 설정되어 Chip Erase할 때 EEPROM의 내용을 보호한다.
- 비트2~1(BOOTSZ1~0) : 부트 사이즈를 선택하는 데 사용한다(그림12 참조).
- 비트0(BOOTRST) : Reset vector의 선택에 사용한다. 디폴트 상태로는 1로 설정되어 0x0000번지를 리셋 주소로 하고 0이면 그림 12에 주어진 값을 리셋 주소로 사용한다.

BOOTSZ1	BOOTSZ0	Boot Size	Pages	Application Flash Section	Boot Loader Flash Section	End Application section	Boot Reset Address (start Boot Loader Section)
1	1	512 words	4	$0000 - $FDFF	$FE00 - $FFFF	$FDFF	$FE00
1	0	1024 words	8	$0000 - $FBFF	$FC00 - $FFFF	$FBFF	$FC00
0	1	2048 words	16	$0000 - $F7FF	$F800 - $FFFF	$F7FF	$F800
0	0	4096 words	32	$0000 - $EFFF	$F000 - $FFFF	$EFFF	$F000

그림 12. 부트 사이즈의 선택(출처:ATMEL)

1.5.9.3. Fuse Low Byte

- 비트7(BODLEVEL) : 디폴트 상태로는 1로 설정되어 Brown Out Detector의 기능을 레벨값에 따라 트리거되도록 설정한다.
- 비트6(BODEN: Brown Out Detector ENable) : 디폴트 상태로는 1로 설정되어 Brown Out Detector의 기능을 허용한다.
- 비트5~4(SUT1~0) : Start up 시간을 설정한다(1.6~7절 참조). 디폴트 상태로는 10의 값을 갖는다.
- 비트3~0(CKSEL3~0) : 클럭 소스를 설정한다(1.6절 참조). 디폴트 상태로는 0001의 값을 갖고 있어 디폴트 상태로는 1MHz의 내부 RC발진기를 통한 클록을 사용하도록 설정되어 있다.

1.6. 시스템 클록과 슬립 모드

1.6.1. 클록 분배

그림13에 보여진 것처럼 ATmega128은 다양한 소스에 의해 클록을 발생시키고 분배가 가능하다. 또한 내부의 각 부분은 모두 동시에 클록이 필요치 않아 소비 전력을 절약시키기 위해 개별적으로 공급을 차단할 수도 있다.

- CPU 클록(clk_{CPU}) : 범용레지스터, 상태레지스터, 데이터 메모리와 같은 AVR 핵심적인 동작과 관련된 클록으로 공급을 차단하면 이들의 동작을 멈추게 한다.
- I/O 클록($clk_{I/O}$) : 타이머, SPI, USART 등 I/O모듈 대부분에서 사용되는 클록이다. 외부인터럽트 모듈에서도 사용되나 일부 외부인터럽트는 I/O 클록의 공급이 차단되어도 인터럽트 처리가 된다.
- 플래쉬 클록(clk_{FLASH}) : 플래쉬롬과의 인터페이스를 제어하며 보통 CPU 클록과 동시에 사용된다.
- 비동기 타이머 클록(clk_{ASY}) : 외부 32kHz 수정 발진기를 소스로 하는 비동기 타이머용 클록이다.
- AD변환기 클록(clk_{ADC}) : AD변환기용의 클록으로 CPU와 I/O클록을 정지시키고 AD변환을 이루어지도록 하여 노이즈를 줄일 수 있도록 한다.

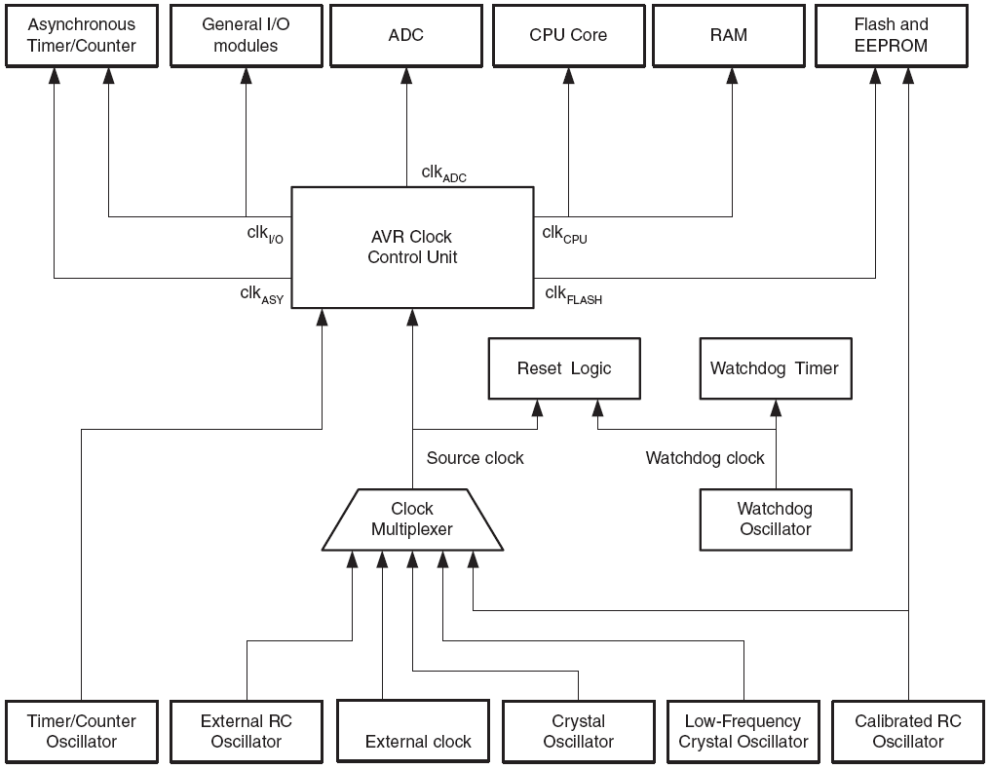


그림 13. ATmega128의 클록 분배(출처:ATMEL)

1.6.2. 클록 발생과 선택

5가지의 클록발생원이 존재하며 CKSEL3~0와 SUT1~0, XDIV레지스터를 이용하여 클록발생원과 주파수를 설정한다.

1.6.2.1. 내부 RC(디폴트 클록) 발진기

- XTAL1, XTAL2 단자의 연결 : 내부 발진기를 사용하므로 XTAL1, XTAL2에는 아무 것도 연결하지 않는다.
- CKOPT의 설정 : 퓨즈 하이 바이트의 비트4 CKOPT를 1로 설정해야 한다.

- CKSEL3~0의 설정 : 퓨즈 로우 바이트의 비트3~0(CKSEL3~0)의 값들을 0001로 하면 1MHz, 0010으로 하면 2MHz, 0011로 하면 4MHz, 0100으로 하면 8MHz의 클록을 사용한다. 디폴트 상태로는 0001의 값을 갖고 있어 디폴트 상태로는 1MHz의 내부 RC발진기를 통한 클록을 사용하도록 설정되어 있다.
- SUT1~0의 설정 : 퓨즈 로우 바이트의 비트5~4(SUT1~0)의 값들을 11로 설정하는 것은 유보되어 있고 나머지 값들(00,01,10)으로 하면 기동시간이 6클럭으로 설정된다. 또한 00로의 설정은 BOD(Brown Out Detector) 사용에 권장되고, 01로 하면 리셋의 경우 추가적으로 4.1 ms의 지연이 발생하고 Fast rising power 사용에 권장되고, 10로 하면 리셋의 경우 추가적으로 65ms의 지연이 발생하고 slowly rising power 사용에 권장된다.
- 내부 발진기를 통해 발생되는 클럭은 주파수가 부정확하므로 확장된 I/O레지스터인 OSCCAL(OSCillator CALibration)레지스터를 조정해서 사용한다. OSCCAL의 값이 0x00이면 CKSEL3~0으로 설정한 값의 50%~100%의 주파수값이 클록으로 사용되며, 0x7F이면 75%~150%, 0xFF이면 100%~200%의 주파수값이 클록으로 사용된다.

1.6.2.2. 외부 RC 발진기

- XTAL1, XTAL2 단자의 연결 : 그림 14와 같이 연결하고 커패시터의 값은 적어도 22pF 이상으로 한다.

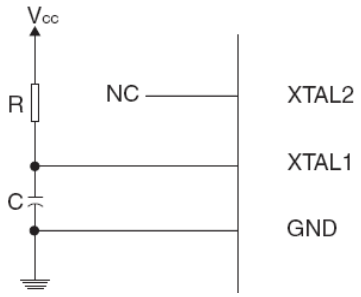


그림 14. 외부 RC 발진기의 연결(출처:ATMEL사)

- 발진주파수 : 클럭의 주파수 값은 $f = 1/3RC$로 결정된다.
- CKOPT의 설정 : 0으로 설정하면 내부의 36pF 커패시터가 사용되므로 그림 18의 커패시터 C를 연결하면 안 된다.
- CKSEL3~0의 설정 : 그림 15참조

CKSEL3..0	Frequency Range (MHz)
0101	0.1 - 0.9
0110	0.9 - 3.0
0111	3.0 - 8.0
1000	8.0 - 12.0

그림 15. 외부 RC 발진기 동작 모드 (출처:ATMEL)

- SUT1~0의 설정 : 그림 16참조.

SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	18 CK	–	BOD enabled
01	18 CK	4.1 ms	Fast rising power
10	18 CK	65 ms	Slowly rising power
11	6 CK(1)	4.1 ms	Fast rising power or BOD enabled

Note: 1. This option should not be used when operating close to the maximum frequency of the device.

그림 16. 외부 RC발진기의 기동시간 선택(출처:ATMEL)

1.6.2.3. 외부 수정발진기

XTAL1, XTAL2에 수정이나 세라믹 resonator를 연결하여 사용하는 경우이다.

- XTAL1, XTAL2 단자의 연결 : 그림 17과 같이 연결하고 커패시터의 값은 그림 18을 참조한다.

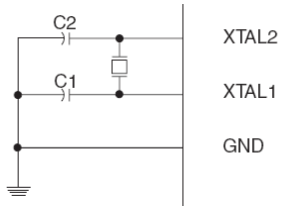


그림 17. 외부 수정 발진기의 연결(출처:ATMEL)

- CKOPT의 설정 : 그림 18참조.

CKOPT	CKSEL3..1	Frequency Range (MHz)	Recommended Range for Capacitors C1 and C2 for Use with Crystals
1	101(1)	0.4 - 0.9	–
1	110	0.9 - 3.0	12 pF - 22 pF
1	111	3.0 - 8.0	12 pF - 22 pF
0	101, 110, 111	1.0 -	12 pF - 22 pF

Note: 1. This option should not be used with crystals, only with ceramic resonators.

그림 18. 외부 수정 발진기 동작 모드 (출처:ATMEL)

- CKSEL3~0의 설정 : 그림 18과 19 참조

CKSEL0	SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
0	00	258 CK(1)	4.1 ms	Ceramic resonator, fast rising power
0	01	258 CK(1)	65 ms	Ceramic resonator, slowly rising power
0	10	1K CK(2)	–	Ceramic resonator, BOD enabled
0	11	1K CK(2)	4.1 ms	Ceramic resonator, fast rising power
1	00	1K CK(2)	65 ms	Ceramic resonator, slowly rising power
1	01	16K CK	–	Crystal Oscillator, BOD enabled
1	10	16K CK	4.1 ms	Crystal Oscillator, fast rising power
1	11	16K CK	65 ms	Crystal Oscillator, slowly rising power

Notes: 1. These options should only be used when not operating close to the maximum frequency of the device, and only if frequency stability at start-up is not important for the application. These options are not suitable for crystals.
2. These options are intended for use with ceramic resonators and will ensure frequency stability at start-up. They can also be used with crystals when not operating close to the maximum frequency of the device, and if frequency stability at start-up is not important for the application.

그림 19. 수정발진기의 기동시간 선택(출처:ATMEL)

- SUT1~0의 설정 : 그림 19참조.

1.6.2.4. 저주파 수정 발진기

XTAL1, XTAL2에 저주파수 수정을 연결하여 사용하는 경우이다.

- XTAL1, XTAL2 단자의 연결 : 그림 17과 같이 연결하고 수정은 32.768kHz를 사용한다.
- CKOPT의 설정 : 0으로 설정하면 내부의 36pF 커패시터가 사용되므로 그림 14의 커패시터들을 연결하면 안

된다.

- CKSEL3~0의 설정 : 1001로 설정한다.
- SUT1~0의 설정 : 그림 20참조.

SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	1K CK(1)	4.1 ms	Fast rising power or BOD enabled
01	1K CK(1)	65 ms	Slowly rising power
10	32K CK	65 ms	Stable frequency at start-up
11	Reserved		

Note: 1. These options should only be used if frequency stability at start-up is not important for the application.

그림 20. 저주파수정발진기의 기동시간 선택(출처:ATMEL)

1.6.2.5. 외부 클록

- XTAL1, XTAL2 단자의 연결 : 외부의 클럭 신호를 XTAL1 단자에 연결하고 XTAL2단자는 비워둔다.
- CKOPT의 설정 : 0으로 설정하면 내부의 36pF 커패시터가 사용된다.
- CKSEL3~0의 설정 : 0000으로 설정한다.
- SUT1~0의 설정 : 퓨즈 로우 바이트의 비트5~4(SUT1~0)의 값들을 11로 설정하는 것은 유보되어 있고 나머지 값들(00,01,10)으로 하면 기동시간이 6클럭으로 설정된다. 또한 00로의 설정은 BOD(Brown Out Detector) 사용에 권장되고, 01로 하면 리셋의 경우 추가적으로 4.1 ms의 지연이 발생하고 Fast rising power 사용에 권장되고, 10로 하면 리셋의 경우 추가적으로 65ms의 지연이 발생하고 slowly rising power 사용에 권장된다.
- 유의점 : 클럭 주파수가 2%이상 변동하면 시스템의 동작이 불안정해지므로 변동이 없도록 해야 한다.

1.6.2.6. XDIV레지스터를 이용한 클럭 주파수 조정

XDIV(Xtal DIVide)레지스터를 이용하여 클럭 발생원을 2~129로 분주한 주파수값을 시스템 클럭으로 사용할 수 있도록 설정할 수 있다.

- 비트7(XDIVEN:Xtal DIVide ENable) : 1로 설정되면 클럭발생원의 주파수 값(clk)을 XDIV6~0의 값으로 설정한 d값을 이용하여 다음의 공식에 따라 모든 클럭의 주파수값(f)으로 설정한다.

$$f = clk / (129 - d)$$

- 비트6~0(XDIV6~0) : 클럭 주파수의 분주비를 설정한다.

SM2	SM1	SM0	Sleep Mode
0	0	0	Idle
0	0	1	ADC Noise Reduction
0	1	0	Power-down
0	1	1	Power-save
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Standby(1)
1	1	1	Extended Standby(1)

Note: 1. Standby mode and Extended Standby mode are only available with external crystals or resonators.

그림 21. 슬립모드의 설정(출처:ATMEL)

1.6.3. Sleep 모드

전원을 절약할 수 있는 6가지의 다양한 슬립모드가 제공된다. MCUCR 레지스터를 설정하여 모드를 선택하고 SLEEP명령을 실행하여 슬립모드에 돌입하도록 한다.

1.6.3.1. MCUCR레지스터를 이용한 모드 설정

MCUCR(MCU Control Register)의 비트5(SE), 비트4,3,2(SM1,SM0,SM2)를 이용해 모드를 설정한다.

- 비트5(SE:Sleep Enable) : 디폴트는 0으로 1로 세트되면 SLEEP 명령시 슬립모드의 돌입이 허용한다.
- 비트4,3,2(SM1,SM0,SM2) : 그림 21과 같은 형태로 값을 조절하여 모드를 설정할 수 있다.

1.6.3.2. Idle모드

CPU의 동작은 멈추지만 SPI, USART, 아날로그 비교기, ADC, TWI, 타이머, 워치독, 인터럽트의 동작은 유지된다. clk_{CPU}, clk_{FLASH} 클럭은 차단되지만 나머지 클럭은 공급된다.

1.6.3.3. ADC noise reduction 모드

CPU의 동작은 멈추지만 ADC, TWI, 타이머0, 워치독, 인터럽트의 동작은 유지된다. $clk_{CPU}, clk_{FLASH}, clk_{I/O}$ 클럭은 차단되지만 나머지 클럭은 공급된다. ADC가 잡음에 영향을 덜받아서 정밀하게 동작할 수 있도록 하는 모드로 ADC의 동작이 활성화된 경우 ADC noise reduction모드로의 돌입은 ADC 변환이 자동으로 시작된다.

1.6.3.4. Power-down 모드

외부 수정 발진기 또는 세라믹 resonator의 동작이 정지되며 TWI, 워치독, 인터럽트의 동작은 유지된다. 모든 클럭이 차단되어 비동기 모듈의 동작만이 가능하다. 이 모드가 해제될 때는 클럭이 안정적으로 되기 위해 충분한 여유 기동시간이 필요한데 이를 사용자가 지정할 수 있다.

1.6.3.5. Power-save 모드

Power-down모드와 유사하나 ASSR 레지스터의 AS0비트를 1로 설정하여 타이머0을 TOSC1, TOSC2 단자로 입력되는 외부 클럭에 의하여 비동기로 동작시킬 때 사용하는 것이 다른 점이다. clk_{ASY} 클럭을 제외한 모든 클럭이 차단된다.

1.6.3.6. Standby 모드

외부 수정 발진기 또는 세라믹 resonator을 클럭 발생원으로 한 경우에만 가능한 슬립모드로 Power-down모드와 유사하나 발진기가 동작하는 것이 다르다. 해제되어 정상동작이 개시되는데 6개의 클럭 사이클만이 소요된다.

1.6.3.7. Extended Standby 모드

외부 수정 발진기 또는 세라믹 resonator을 클럭 발생원으로 한 경우에만 가능한 슬립모드로 Power-save모드와 유사하나 발진기가 동작하는 것이 다르다. 해제되어 정상동작이 개시되는데 6개의 클럭 사이클만이 소요된다.

	Active Clock Domains					Oscillators		Wake Up Sources					
Sleep Mode	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{ASY}	Main Clock Source Enabled	Timer Osc Enabled	INT7:0	TWI Address Match	Timer 0	SPM/ EEPROM Ready	ADC	Other I/O
Idle			X	X	X	X	X(2)	X	X	X	X	X	X
ADC Noise Reduction				X	X	X	X(2)	X(3)	X	X	X	X	
Power-down								X(3)	X				
Power-save					X(2)		X(2)	X(3)	X	X(2)			
Standby(1)						X		X(3)	X				
Extended Standby(1)					X(2)	X	X(2)	X(3)	X	X(2)			

Notes: 1. External Crystal or resonator selected as clock source
2. If AS0 bit in ASSR is set
3. Only INT3:0 or level interrupt INT7:4

그림 22. 슬립모드의 동작 요약(출처:ATMEL)

1.6.3.8. 슬립모드의 동작 요약

- 동작 : MCUCR의 SE를 1로 세트하고 SM2~0을 그림 21에 따라 모드를 설정하고 SLEEP명령어를 실행 시킨다. 6개의 동작모드 Idle, ADC noise reduction, Power-down, Power-save, Standby, Extended Standby 동작 요약은 그림 22에 주어졌다.
- 인터럽트에 의한 해제 : 슬립모드안에서 인터럽트를 만나면 MCU가 동작을 시작하고 슬립모드가 해제되는데 기동시간 이외에 4사이클이 더 경과한 후에 동작이 개시되고 인터럽트 서비스를 다 실행하고 슬립모드에 돌입하도록 한 이전의 SLEEP 명령의 바로 뒤에 이어지는 명령으로 복귀된다. 해제된 모든 레지스터와 메모리에 있는 내용은 슬립모드 이전의 값이 보존된다.
- 리셋에 의한 해제 : 슬립모드에서 리셋이 발생하면 슬립모드가 해제되고 리셋동작이 수행된다.

1.7. Reset과 Watchdog 타이머

1.7.1. 리셋

- 정상적으로 동작하고 있는 마이컴이 리셋되면 모든 I/O레지스터값이 디폴트값으로 초기화되고 프로그램은 리셋벡터에서 시작된다. 리셋은 5가지의 발생원이 존재한다.

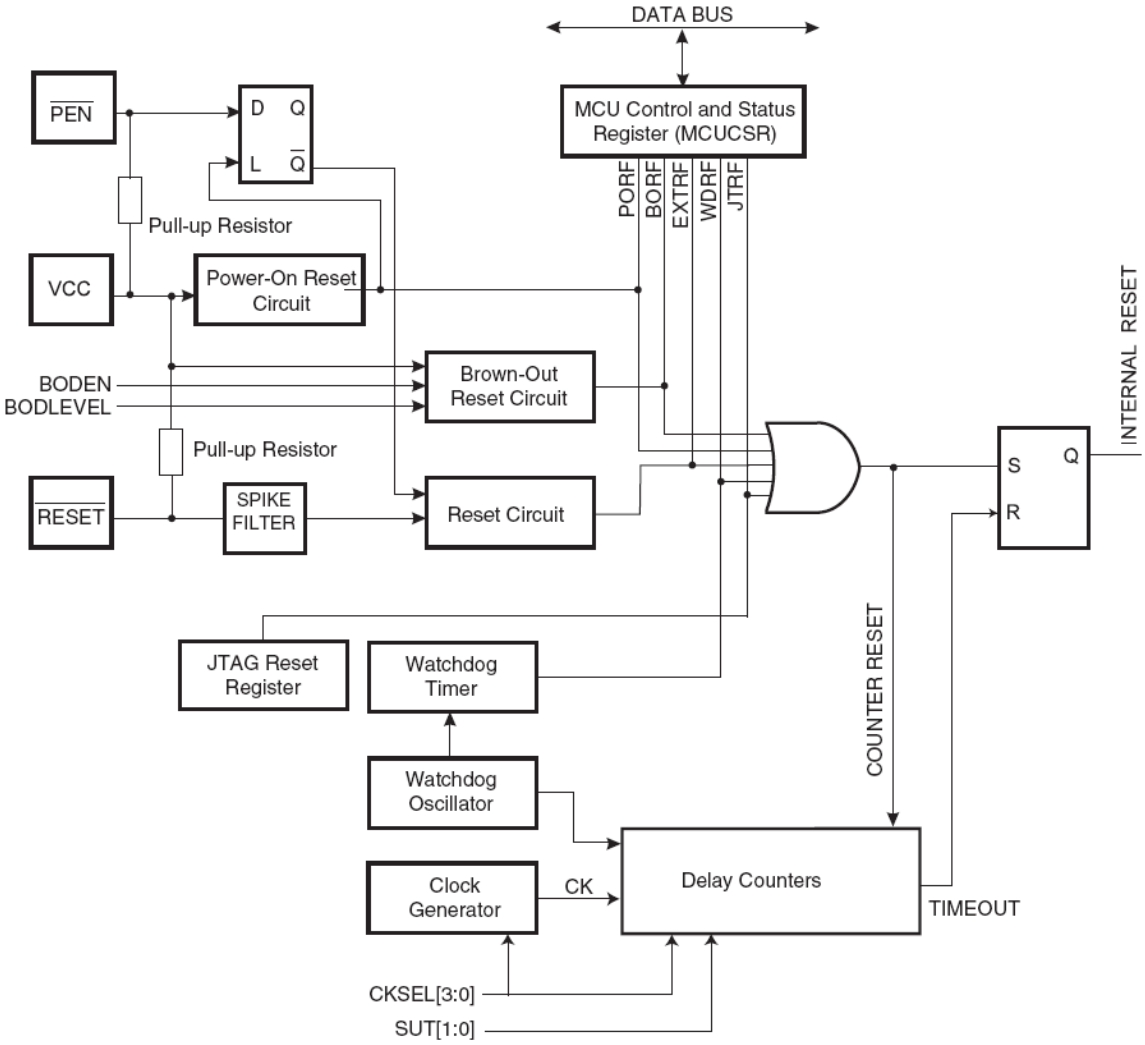


그림 23. 리셋 관련 내부 회로 블록선도(출처:ATMEL)

① 파워온 리셋 : 공급전원이 그림 24에 정의된 V_{POT}이하일 때 리셋되는 것.

② 외부 리셋 : $\overline{\text{RESET}}$핀에 50ns 이상동안 그림 24에 정의된 V_{RST}이하의 신호가 입력되면 리셋되는 것

③ Brown-out 리셋 : 공급전원이 2마이크로초 이상동안 그림 24에 정의된 V_{BOT}이하로 떨어져 Brown-out 감지기가 작동해 리셋되는 것.

④ 워치독 리셋 : 워치독 타이머가 지정한 주기 이상이 경과되어 워치독 기능이 동작할 때 리셋되는 것

⑤ JTAG AVR 리셋 : JTAG이 연결되어 있고 JTAG명령 AVR_RESET 명령에 따라 선택된 JTAG 리셋 레지스터에 논리 1이 저장되어 있는 경우 리셋되는 것

Symbol	Parameter	Condition	Min	Typ	Max	Units
V_{POT}	Power-on Reset Threshold Voltage (rising)			1.4	2.3	V
	Power-on Reset Threshold Voltage (falling)			1.3	2.3	V
V_{RST}	$\overline{\text{RESET}}$ Pin Threshold Voltage		0.2 V_{CC}		0.85 V_{CC}	V
t_{RST}	Pulse width on $\overline{\text{RESET}}$ Pin		1.5			µs
V_{BOT}	Brown-out Reset Threshold Voltage	BODLEVEL = 1	2.4	2.6	2.9	V
		BODLEVEL = 0	3.7	4.0	4.5	
t_{BOD}	Minimum low voltage period for Brown-out Detection	BODLEVEL = 1		2		µs
		BODLEVEL = 0		2		µs
V_{HYST}	Brown-out Detector hysteresis			100		mV

그림 24. 리셋 관련 변수(출처:ATMEL)

1.7.1.1. 파워온 리셋(POR)

- 전원 투입후 제대로 리셋이 되지 않으면 PC와 레지스터들이 제대로 초기화되지 못하여 마이컴은 임의의 위치에서 명령을 시작하게 되는데 이를 막기 위해 전원 투입후 자동으로 리셋이 되도록 하는 것을 파워온 리셋이라 한다.
- POR회로는 V_{POT}보다 공급전원이 작으면 동작이 시작되는데 공급전원이 투입되고 나서 곧바로 정상적인 레벨로 올라가지 못하고 그림 25처럼 V_{POT}보다 작으므로 POR회로가 작동되어 파워온 리셋 동작이 시작되고 공급전원이 V_{POT}에 도달하면 시간지연카운터를 동작시켜 일정한 시간 t_{OUT}동안 리셋을 유지시킨다.

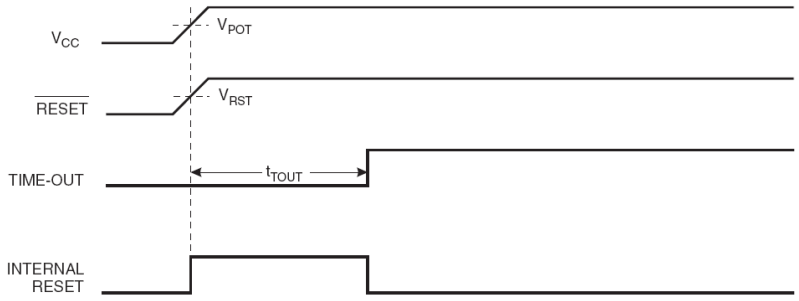


그림 25. 파워온 리셋 동작 타이밍도(출처:ATMEL)

1.7.1.2. 외부 리셋

- $\overline{\text{RESET}}$핀에 50ns 이상 동안 그림 24에 정의된 V_{RST}이하의 신호가 입력되면 리셋이 시작되는데 그림 26에서처럼 리셋신호가 저전압을 유지하다가 V_{RST}이상으로 올라가면 시간지연카운터를 동작시켜 일정한 시간 t_{OUT}

이 부가적으로 경과될 때까지 리셋을 유지시킨다.

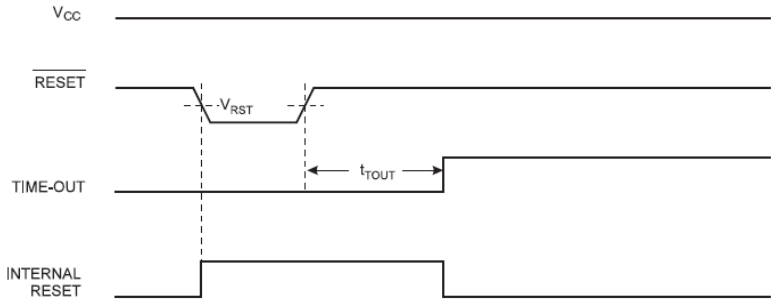


그림 26. 외부리셋 동작 타이밍도(출처:ATMEL)

1.7.1.3. Brown-out 리셋

- 마이컴 내부에 있는 저전압 검출회로(BOD:Brown-Out Detector)가 공급전원이 2마이크로초 이상동안 그림 24에 정의된 V_{BOT}이하로 떨어지면 발생된다.
- 그림 27의 동작 타이밍도처럼 발생하는데 전압 스파이크에 의한 오동작을 줄이기 위해 히스테리시스 특성을 갖도록 하여 전압이 감소하여 $V_{\mathrm{BOT-}} = V_{\mathrm{BOT}} - V_{HYST}/2$이하로 될 때 리셋동작이 시작되고 전압이 증가하여 $V_{\mathrm{BOT+}} = V_{\mathrm{BOT}} - V_{HYST}/2$이상으로 된 후 시간지연카운터를 동작시켜 일정한 시간 t_{OUT}이 부가적으로 경과될 때까지 리셋을 유지시킨다.

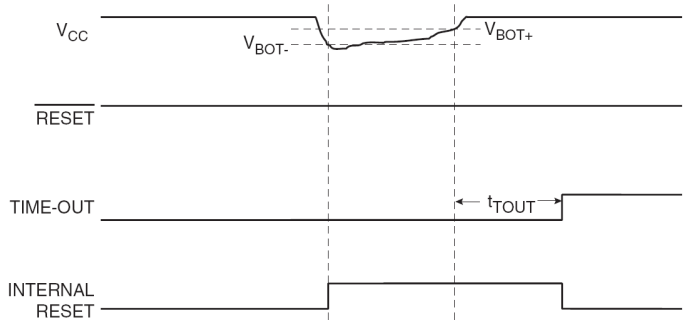


그림 27. 저전압 검출 리셋 동작 타이밍도(출처:ATMEL)

- BOD 기능은 1.5.9절에 설명된 Fuse Low byte의 BODEN 비트를 1로 하여 활성화시킬 수 있고 BODKEVEL 비트에 따라 그림 24처럼 설정할 수 있다.

1.7.1.4. 워치독 리셋

- 워치독 타이머가 지정한 주기 이상이 경과되면 그림 28처럼 1클럭 사이클의 리셋펄스가 발생하여 리셋동작이 시작되고 리셋펄스가 끝나면 시간지연카운터를 동작시켜 일정한 시간 t_{OUT}이 부가적으로 경과될 때까지 리셋을 유지시킨다.

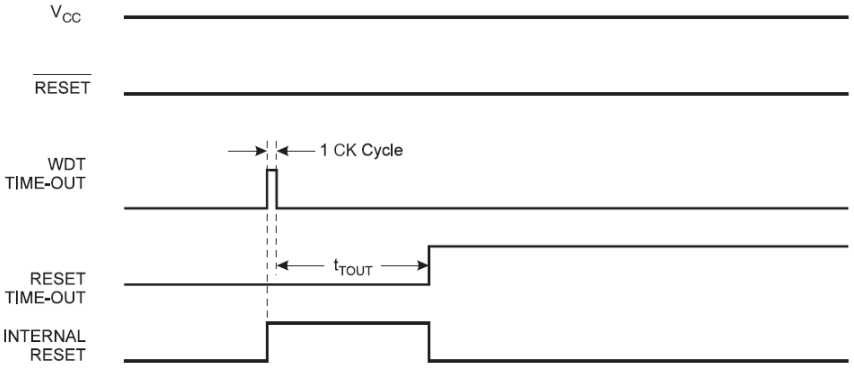


그림 28. 워치독 리셋 동작 타이밍도(출처:ATMEL)

1.7.1.5. MCUCSR 레지스터

MCUCSR(MCU Control and Status Register)레지스터는 마이컴의 리셋 발생원에 대한 정보를 제공한다.

- 비트4(JTRF:JTAG Reset Flag) : JTAG 리셋이 발생한 경우 1로 세트된다. 파워온 리셋이나 0을 써넣음으로써 클리어된다.
- 비트3(WDRF:Watch Reset Flag) : 워치독 리셋이 발생한 경우 1로 세트된다. 파워온 리셋이나 0을 써넣음으로써 클리어된다.
- 비트2(BORF:Brown-Out Reset Flag) : Brown-out 리셋이 발생한 경우 1로 세트된다. 파워온 리셋이나 0을 써넣음으로써 클리어된다.
- 비트1(EXTRF:EXTernal Reset Flag) : 외부 리셋이 발생한 경우 1로 세트된다. 파워온 리셋이나 0을 써넣음으로써 클리어된다.
- 비트0(PORF:Power-On Reset Flag) : 파워온 리셋이 발생한 경우 1로 세트된다. 파워온 리셋이나 0을 써넣음으로써 클리어된다.

1.7.2. 워치독 타이머

1.7.2.1. 워치독 타이머의 구성

- 역할 : 프로세서가 안정적으로 동작하는지 감시하는 기능을 수행한다. 프로세서에 이상이 생겨 일정 시간마다 워치독 타이머가 리셋되지 않게 되는 경우 시스템을 리셋시켜 마이컴의 안정적인 동작을 보장하도록 한다. 워치독 타이머의 리셋은 WDR 명령을 실행하거나 워치독의 동작을 금지시키거나 마이컴이 리셋되는 경우 리셋된다.
- 구성 : 그림 29처럼 구성되어 마이컴 내부의 RC발진기와는 별개로 내장되어 있는 워치독발진기와 그 곳에서 만들어지는 1MHz 클럭을 받아 이를 8가지로 분주하는 워치독 프리스케일러, WDTCR레지스터의 설정값에 따라 마이컴 리셋을 발생시키는 논리회로로 구성된다.

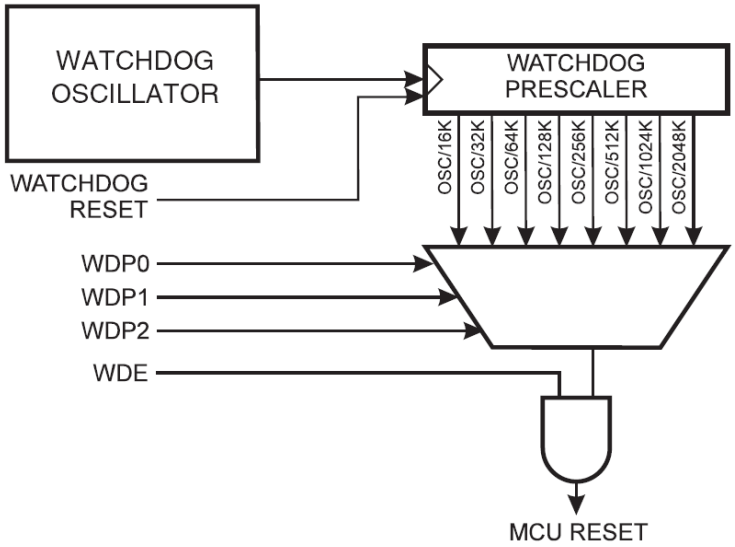


그림 29. 워치독 타이머의 구성(출처:ATMEL)

1.7.2.2. WDTCR 레지스터

- 비트4(WDTOE:WatchDog Turn-Off Enable) : 디폴트 값은 0이다. WDE비트가 0일때 1로 세트되어야 한다. 그렇지 않으면 워치독은 비활성화된다. 1로 세트되면 4 클록이 지난 후에 하드웨어적으로 클리어된다. 안전 레벨 1과 2에서 프리스케일러 비트를 바꾸는 경우 1로 세트되어야 한다.
- 비트3(WDE:WatchDog Enable) : 디폴트 값은 0이다. 1로 세트되면 워치독 타이머가 활성화되고 0으로 클리어되면 비활성화된다. WDE는 WDCE비트가 1로 세트된 경우에만 클리어될 수 있다.

● 비트2~0(WDP2~0: Watch-Dog timer Prescaler2~0) : 디폴트 값은 0이다. 워치독 타이머가 활성화된 경우 프리스케일러를 그림 30처럼 설정한다.

WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V_{CC} = 3.0V	Typical Time-out at V_{CC} = 5.0V
0	0	0	16K (16,384)	14.8 ms	14.0 ms
0	0	1	32K (32,768)	29.6 ms	28.1 ms
0	1	0	64K (65,536)	59.1 ms	56.2 ms
0	1	1	128K (131,072)	0.12 s	0.11 s
1	0	0	256K (262,144)	0.24 s	0.22 s
1	0	1	512K (524,288)	0.47 s	0.45 s
1	1	0	1,024K (1,048,576)	0.95 s	0.9 s
1	1	1	2,048K (2,097,152)	1.9 s	1.8 s

그림 30. 워치독 타이머 프리스케일러의 설정(출처:ATMEL)

1.7.2.3. 워치독의 설정

● 워치독 타이머의 안전레벨 설정 : 1.5.9.1절에 소개된 Etended Fuse Byte의 비트들 값으로 안전레벨0, 1, 2로 그림 31에 보여진 것처럼 설정할 수 있다(퓨즈의 비트값들은 프로그램되면 0이되고 프로그램되지 않으면 1).

M103C	WDTON	Safety Level	WDT Initial State	How to Disable the WDT	How to Change Time-out
Unprogrammed	Unprogrammed	1	Disabled	Timed sequence	Timed sequence
Unprogrammed	Programmed	2	Enabled	Always enabled	Timed sequence
Programmed	Unprogrammed	0	Disabled	Timed sequence	No restriction
Programmed	Programmed	2	Enabled	Always enabled	Timed sequence

그림 31. 워치독 타이머의 설정(출처:ATMEL)

● 안전레벨0 : ATmega103과 호환모드로 그림 31에 보여진 것처럼 초기에는 워치독이 금지되어 있다. WDE를 1로 세트하면 워치독이 허용된다. WDP2~0를 바꿔 즉 프리스케일러의 값을 바꿔 타임아웃 주기를 변경시키는 것에는 제한이 없으나 워치독 타이머를 비활성화시키려면 다음의 과정을 거쳐야 한다.

①WDTCR레지스터에 동시에 WDCE=1, WDE=1로 세트시킨다.

② 4클록 사이클 이내에 WDE=0으로 클리어시킨다.

● 안전레벨1 : 디폴트 상태이다. 그림 31에 보여진 것처럼 초기에는 워치독이 금지되어 있다. WDE를 1로 세트하면 워치독이 허용된다. WDP2~0를 바꿔 즉 프리스케일러의 값을 바꿔 타임아웃 주기를 변경시키거나 워치독 타이머를 비활성화시키려면 다음의 과정을 거쳐야 한다.

①WDTCR레지스터에 동시에 WDCE=1, WDE=1로 세트시킨다.

② 4클록 사이클 이내에 WDE비트와 WDP2~0 비트에 원하는 값을 쓴다.

● 안전레벨2 : 그림 31에 보여진 것처럼 초기에 워치독이 허용되어 있고 WDE가 1로 세트되어 있고 워치독을 비활성화시킬 수 없다. WDP2~0를 바꿔 즉 프리스케일러의 값을 바꿔 타임아웃 주기를 변경시키려면 다음의 과정을 거쳐야 한다.

①WDTCR레지스터에 동시에 WDCE=1, WDE=1로 세트시킨다.

② 4클록 사이클 이내에 WDCE=0으로 클리어시키고 WDP2~0 비트에 원하는 값을 쓴다.

2. ATmega128 시스템 개발 기초

2.1. 명령어 분류와 주소지정 방식

- 명령어 기본 형식 : 명령어는 명령코드(OP 코드:OPeration code)와 오퍼랜드(operand)로 구성되는데 명령코드는 명령 그 자체이고 오퍼랜드는 명령의 대상으로 레지스터, 메모리, 상수값이 사용될 수 있으며 명령에 따라 오퍼랜드 없이 명령코드에 포함되는 경우가 있다. 명령코드는 대부분 16비트의 길이이지만 LDS, STS, JMP, CALL 4개의 명령어는 32비트이다.
- 실행시간에 따른 분류 : 실행하는데 1~4 기계사이클를 요구하여 ① 1기계사이클 명령어(예: ADD Rd, Rr)와 ② 2기계사이클 명령어(예:ADIW Rdl, K) ③3기계사이클 명령어(예:ELPM 혹은 LPM) ④4기계사이클 명령어(예:RETI 혹은 RET)로 나뉠 수 있다.
- 기능에 따른 분류 : 데이터전송, 산술과 논리 연산, 분기, 비트 조작, MCU 제어 명령으로 나눌 수 있다.

① 데이터전송 명령 : 내부램이나 외부램의 데이터 전송을 위한 명령으로 MOV, MOVW, LDI, LD, LDD, LDS, ST, STD, STS, LPM, ELPM, SPM, IN, OUT, PUSH, POP 등 16가지가 관계된다.

② 산술과 논리연산 명령 : 가감산, 증감, 곱하기 등 산술연산과 논리연산 명령어로 ADD, ADC, ADIW, SUB, SUBI, SBC, SBCI, SBIW, AND, ANDI, OR, ORI, EOR, COM, NEG, SBR, CBR, INC, DEC, TST, CLR, SER, MUL, MULS, MULSU, FMUL, FMULS, FMULSU 등 28가지가 관계된다.

③ 분기 명령 : 조건을 따져 분기하거나 프로그램의 실행순서를 바꾸는 명령으로 RJMP, IJMP, JMP, RCALL, ICALL, CALL, RET, RETI, CPSE, CP, CPC, CPI, SBRC, SBRS, SBIC, SBIS, BRBS, BRBC, BREQ, BRNE, BRCS, BRCC, BRSH, BRLO, BRMI, BRPL, BRGE, BRLT, BRHS, BRHC, BRTS, BRTC, BRVS, BRVC, BRIE, BRID 등이 관계된다.

④ 비트 조작 명령 : 1비트값에 대한 비트조작 불리안 연산 명령으로 SBI, CBI, LSL, LSR, ROL, ROR, ASR, SWAP, BSET, BCLR, BST, BLD, SEC, CLC, SEN, CLN, SEZ, CLZ, SEI, CLI, SES, CLS, SEV, CLV, SET, CLT, SEH, CLH등이 관계된다.

⑤ MCU 제어 명령 : NOP, SLEEP, WDR, BREAK 등 4개의 MCU 제어 관련 명령

- 오퍼랜드의 주소지정방식

① 직접상수지정 방식 : 오퍼랜드로서 상수값을 사용하는 방식이다.

② I/O 직접주소지정 방식 : 오퍼랜드로서 64개의 I/O 레지스터의 주소값을 직접 지정한다.

③ 레지스터직접주소지정 방식 : R0~R31이 오퍼랜드로서 지정된다.

④ 데이터 직접주소지정 방식 : 오퍼랜드로서 데이터 메모리 영역의 16비트 주소값을 직접 지정한다.

⑤ 데이터 간접지정 방식 : 오퍼랜드의 주소를 저장하고 있는 X,Y,Z레지스터를 이용하여 간접적으로 지정한다.

⑥ 선증가 데이터 간접지정방식 : 데이터 간접지정방식과 비슷하나 X,Y,Z레지스터값을 먼저 1만큼 감소시킨다.

⑦ 후증가 데이터 간접지정방식 : 데이터 간접지정방식과 비슷하나 X,Y,Z레지스터값을 명령 실행 후에 1만큼 증가시킨다.

⑧ 변위 데이터 간접지정방식 : 데이터 간접지정과 비슷하나 Y,Z레지스터값에 변위값을 합하여 사용한다.

⑨ 프로그램 메모리 상수 지정방식 : Z레지스터를 사용하여 프로그램 메모리 영역의 상수값을 지정한다.

⑩ 후증가 프로그램 메모리 상수 지정방식 : 프로그램 메모리 상수 지정방식과 비슷하나 Z레지스터 값을 명령 실행 후에 1만큼 증가시킨다.

2.2. 명령어 요약

다음은 ATmega128의 명령을 요약한 것으로 항목에서 #clocks은 수행에 걸리는 기계사이클, Flags은 명령 수행으로 영향을 받는 CSEG의 비트를 의미한다. 그리고 다음의 정의들이 사용된다.

표 7. 어셈블리 명령에 쓰인 약자의 의미

Rd	Destination register R0~R31	Rr	Source register R0~R31
b, s	바이트 데이터의 비트번호값 0~7	P	I/O레지스터 주소값 (0-31/63)
K	8비트 상수값	k,q	상수
Rdl	R24, R26, R28, R30.	X	Indirect address register R27:R26
Y	Indirect address register R29:R28	Z	Indirect address register R31:R30

- **분기 명령**

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1 / 2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1 / 2
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
JMP	k	Direct Jump	PC ← k	None	3
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
CALL	k	Direct Subroutine Call	PC ← k	None	4
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1 / 2 / 3
CP	Rd,Rr	Compare	Rd – Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd – K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1 / 2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1 / 2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1 / 2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1 / 2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1 / 2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1 / 2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1 / 2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1 / 2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1 / 2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1 / 2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1 / 2

그림 32. 분기 명령 모음 (출처:ATMEL)

- **MCU 제어 명령**

Mnemonics	Operands	Description	Operation	Flags	#Clocks
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For On-chip Debug Only	None	N/A

그림 33. MCU 제어 명령 모음 (출처:ATMEL)

● 데이터전송 명령

Mnemonics	Operands	Description	Operation	Flags	#Clocks
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
ELPM		Extended Load Program Memory	R0 ← (RAMPZ:Z)	None	3
ELPM	Rd, Z	Extended Load Program Memory	Rd ← (RAMPZ:Z)	None	3
ELPM	Rd, Z+	Extended Load Program Memory and Post-Inc	Rd ← (RAMPZ:Z), RAMPZ:Z ← RAMPZ:Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2

그림 34. 데이터 명령 모음(출처:ATMEL)

● 산술과 논리 연산 명령

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← $FF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← $00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • ($FF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← $FF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2

그림 35. 산술과 논리 연산 명령 모음(출처:ATMEL)

● 비트 조작 명령

Mnemonics	Operands	Description	Operation	Flags	#Clocks
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1

그림 36. 비트 조작 명령 모음 (출처:ATMEL)

2.2.1. 어셈블리어의 일반형태

● 어셈블리어의 일반형태는 아래와 같다.

[라벨 + : + 1개 이상의 공백] + 명령어(혹은 지시어) + [1개 이상의 공백 + ; + 주석]

(예: START: LDI AL, 0b10000000 ;AL에 이진수 10000000 값을 넣어라.)

● 라벨 (Label)은 기호주소로 분기명령이나 참조에 사용되는 심볼(Symbol)로 ① 심볼과 바로 뒤에 공백없이 : (콜론)을 두고 콜론과 명령어는 1개 이상의 공백을 두어야 한다. ② 어셈블러(어셈블리어를 기계어로 바꾸어주는 프로그램)에 따라 다르지만 길이는 보통 1에서 수십개까지의 기호가 허용되고 AVR Studio4에서 심볼은 영문자와 숫자와 특수문자(_)의 조합을 사용하고 처음은 반드시 숫자가 아닌 기호(영문자나 _)로 시작해야한다. ③ R0-R31과 같은 어셈블러 특수 심볼 그리고 명령어와 지시어 등 예약어는 심볼로 사용될 수 없고 ④ 심볼은 보통 명령어, 지시어와 함께 대문자와 소문자의 구분이 없다(AVR Studio4에서 구분을 하게 옵션 설정 가능).

● 주석은 반드시 앞에 ;(세미콜론)을 붙여야 하는데 어셈블러는 세미콜론과 같은 줄에 존재하며 세미콜론 뒤쪽에 존재하는 글은 주석으로 여기고 모두 무시한다.

2.2.2. 지시어(의사 명령)

● 지시어란 기계어로 번역되지 않고 어셈블러에게 어떤 정보만을 제공하는 것으로 의사명령이라고 불린다.

● **CSEG** : 프로그램 메모리에 해당하는 코드 세그먼트의 시작을 정의한다. 파라메터를 갖고 있지 않다.

● **DSEG** : SRAM 데이터 메모리 세그먼트의 시작을 정의한다. 파라메터를 갖고 있지 않다.

● **ESEG** : EEPROM 데이터 메모리 세그먼트의 시작을 정의한다. 파라메터를 갖고 있지 않다.

● **EQU** : .EQU + 1개 이상의 공백 + 심볼 + 1개 이상의 공백 + 상수값의 형태로 어떤 심볼의 값을 정의하는데 사용된다. EQU를 사용하면 숫자로 된 값대신에 기억하기 쉬운 심볼을 사용하여 프로그램을 쉽게 할 수 있다. 어셈블러에서는 기계어로 번역할 때 라벨을 상수값으로 대체하여 해석하게 된다. 한 번 할당된 값은 수정되거나 다시 정의될 수 없다.

예 : .EQU BUF= 0x20 ;기계어로 번역시 BUF는0x20으로 대체된다.

- **SET** : EQU와 유사하나 나중에 다시 정의되어 수정 가능하다.
- **DEF** : .DEF +1개 이상의 공백 + 심볼 +1개 이상의 공백 +레지스터이름의 형태로 기억하기 쉬운 심볼에 레지스터를 할당하여 프로그램을 쉽게 할 수 있다. 나중에 다시 정의되어 수정 가능하고 같은 레지스터에 여러 가지 심볼을 할당할 수 있다.

  예 : .DEF BUF= R0 ; BUF는 R0
- **UNDEF** : .UNDEF +1개 이상의 공백 + 심볼의 형태로 심볼에 레지스터를 할당한 것을 해제한다.

  예 : .UNDEF BUF ; BUF를 해제
- **ORG** : .ORG +1개 이상의 공백 +주소의 형태로 바로 뒤에 오는 프로그램 명령어의 시작번지나 데이터의 시작번지 지정에 사용된다. 주소로는 숫자 뿐만 아니라 라벨도 사용될 수 있다. 다음과 같은 프로그램의 경우 명령어 LDI AL, 0b10000000와 그 이후의 명령은 프로그램 메모리의 20번지를 시작으로 차례로 저장된다.

```
예: .CSEG
        .ORG    0x0020
START:  LDI     AL, 0b10000000    ;AL에 이진수 10000000 값을 넣어라.
        ..............
```

- **BYTE** : 심볼 +: +1개 이상의 공백 +.BYTE +1개 이상의 공백 +숫자의 형태로 SRAM 데이터 세그먼트에 숫자 만큼의 메모리를 바이트 단위로 예약해둔다.

  예 : CNT: .BYTE 1 ;CNT을 위해 1바이트를 유보해 두라.
- **DB** (Define Byte) : 라벨 +: + 1개 이상의 공백 +.DB +1개 이상의 공백 +데이터의 형태로 1바이트(8비트) 단위로 숫자나 문자 데이터들을 프로그램 메모리나 EEPROM 세그먼트에 저장할 때 사용한다.

```
예 : .ESEG
DATA:   DB      1, 2, 'S'  ; 라벨 DATA번지에 1을 DATA+1에 2를 DATA+2에 'S'를 저장.
```

- **DW** (Define Word) : 라벨 +: + 1개 이상의 공백 +.DW +1개 이상의 공백 +데이터의 형태로 1워드(2바이트,16비트) 단위로 숫자나 문자 데이터들을 프로그램 메모리나 EEPROM 세그먼트에 저장할 때 사용한다.
- **DD** (Define Doubleword) : 라벨 +: + 1개 이상의 공백 +.DD +1개 이상의 공백 +데이터의 형태로 2워드(4바이트, 32비트)단위로 숫자나 문자 데이터들을 프로그램 메모리나 EEPROM 세그먼트에 저장할 때 사용한다.
- **DQ** (Define Quadword) : 라벨 +: + 1개 이상의 공백 +.DQ +1개 이상의 공백 +데이터의 형태로 4워드(8바이트, 64비트)단위로 숫자나 문자 데이터들을 프로그램 메모리나 EEPROM 세그먼트에 저장할 때 사용한다.
- **INCLUDE** : .INCLUDE + 1개 이상의 공백 +"+ 파일이름 +"의 형태로 외부의 파일을 include하여 어셈블할 때 사용한다.
- **IF** : .IF + 1개 이상의 공백 +조건식의 형태로 조건식을 만족하면 아래 부분를 어셈블한다.
- **ELIF** : .ELIF + 1개 이상의 공백 +조건식의 형태로 앞선 IF의 조건을 만족시키지 않는 다른 조건을 판단할 때 사용하며 주어진 조건식을 만족하면 아래 부분를 어셈블한다.
- **IFDEF** : .IFDEF + 1개 이상의 공백 +심볼의 형태로 심볼이 정의되어 있으면 아래 부분을 어셈블한다.
- **IFNDEF** : .IFNDEF + 1개 이상의 공백 +심볼의 형태로 심볼이 정의되지 않았으면 아래 부분을 어셈블한다.
- **ELSE** : IF, IFDEF, IFNDEF와 같은 조건문에서 이전 조건이 만족되지 않으면 아래 부분을 어셈블한다.
- **ENDIF** : .ENDEF의 형태로 조건 블록의 마지막을 표시한다.
- **ENDM 혹은 ENDMACRO** : .ENDM 혹은 .ENDMACRO의 형태로 마크로를 정의를 종료할 때 사용한다.
- **MACRO** : .MACRO + 1개 이상의 공백 +마크로이름의 형태로 마크로를 정의할 때 사용한다. 마크로는 @0~@9로 지정된 10개까지의 파라메터가 사용될 수 있고 마크로를 호출할 때는 아래의 예에서처럼 마크로 이름 다음에 공백을 두고 해당 파라메터를 차례로 ,를 사용해 나열하면 된다.

```
예: .MACRO SUBL16               ; Start macro definition
        subi  @1,low(@0)    ; Subtract low byte
```

```
    sbci@2,high(@0)    ; Subtract high byte
.ENDM
.CSEG
SUBL16 0x3456, R0,R1
```

- **ERROR** : .WARNING + 1개 이상의 공백 +"주의문장"의 형태로 어셈블시 주의문장을 출력하고 어셈블을 중지한다.

  예: .IFDEF EXPTEST
  .ERROR "This is not proper" ;만약 EXPTEST가 정의되었으면 This is ...을 출력하고 어셈블을 멈춘다.
  .ENDIF

- **WARNING** : .WARNING + 1개 이상의 공백 +"주의문장"의 형태로 어셈블시 주의문장을 출력하나 ERROR처럼 어셈블을 중지하지는 않는다.

  예: .IFDEF EXPTEST
  .WARNING "This is not properly tested, use at own risk." ;만약 EXPTEST가 정의되었으면 This is ...을 출력
  .ENDIF

2.2.3. 숫자, 문자, 연산자

- 오퍼랜드로 상수값을 사용할 때는 반드시 진법 표현을 명확히 해야 한다. 2진수의 경우에는 0b를 앞에 붙여주고(예: 0b00000000), 16진수의 경우에는 0x 혹은 $를 붙여주고, 8진수의 경우 O를 붙여주고, 10진수의 경우에는 아무런 표시도 하지 않는다.
- 문자상수값은 반드시 앞뒤에 반드시 따옴표(')를 붙여준다. 문자열의 경우에는 큰 따옴표를 사용한다.
- 오퍼랜드에 사용되는 수식의 연산자에는 산술연산자, 논리 연산자, 비교 연산자, 조건연산자가 있다.
- 산술연산자로 +(더하기), -(빼기), /(나누기), *(곱하기), %(나머지가 정수가 되도록 나눈 나누기 연산의 나머지 값을 결과로 내놓는 mode연산), <<(오른쪽으로 이동), >>(왼쪽으로 이동)이 있다.
- 논리 연산자로 !(NOT), &&(논리곱), ||(논리합), ~(비트단위 NOT), &(비트단위 AND), |(비트단위 OR), ^(비트단위 exclusive OR)이 쓰인다.
- 비교 연산자로 == (동치), != (Not Equal), < (Less Than), <= (Less than or Equal to), > (Greater Than), >= (Greater than or Equal to)가 있다.
- 조건 연산자로 아래와 같은 형식으로 사용하며 삼항 연산자로 보통 불린다. 조건식이 참이거나 0이 아니면 식1을 처리하고 거짓이거나 0이면 식2를 처리하라는 의미이다.

  변수 = (조건) ? 식1 : 식2;

2.2.4. 함수

- LOW(x) : x의 하위 바이트값을 돌려준다.
- HIGH(x) : x의 2번째 바이트값을 돌려준다.
- BYTE2(x) : x의 2번째 바이트값을 돌려준다.
- BYTE3(x) : x의 3번째 바이트값을 돌려준다.
- BYTE4(x) : x의 4번째 바이트값을 돌려준다.
- LWRD(x) : x의 비트 15~0 값을 돌려준다.
- HWRD(x) : x의 4번째 바이트값을 돌려준다.
- PAGE(x) : x의 비트 16~21값을 돌려준다.
- EXP2(x) : 2^x 값을 돌려준다.

- LOG2(x) : $\log_2 x$ 값의 정수부를 돌려준다.
- INT(x) : x의 정수부분을 돌려준다.
- FRAC(x) : x의 소수부분을 돌려준다.
- Q7(x) : x를 FMUL/FMULS.FMULSU 명령에 적절한 형태의 부호있는 1바이트 수로 변환한다.
- Q15(x) : x를 FMUL/FMULS.FMULSU 명령에 적절한 형태의 부호있는 1바이트 수로 변환한다.
- ABS(x) : x의 절대값을 돌려준다.
- DEFINED(x) : x라는 심볼이 정의된 경우 1을 돌려준다.

2.3. AVR Studio4

AVR Studio는 ATMEL사(http://www.atmel.com)에서 개발한 AVR용의 소프트웨어 개발툴로 현재 Ver 4.13.528이 나와있어 윈도우즈에 기반한 통합 개발환경을 제공한다. 에뮬레이터와 시뮬레이터를 동시에 사용할 수 있고 어셈블러를 제공한다. 만약에 AVRISP, JTAG ICE, STK500, ICE40, ICE50 등의 에뮬레이터가 PC의 직렬포트에 접속되어 있으면 에뮬레이터로 동작하고 그렇지 않으면 시뮬레이터로 동작한다.

2.3.1. 새 프로젝트 등록과 실행 파일 만들기

① 초기화면 : AVR Studio Ver 4.13.528을 http://www.atmel.com에서 내려받아 설치하고 실행하면 그림 37과 같은 초기화면이 뜬다. 상단에 [File], [Project], [Build], [View], [Tools], [Debug], [Help] 메뉴가 있고 [New Project], [Open] 등의 메뉴를 갖는 Welcome to AVR Studio 4라는 이름의 작은 창이 있고 그 이외에 I/O View, Message라는 이름의 작은 창이 있다.

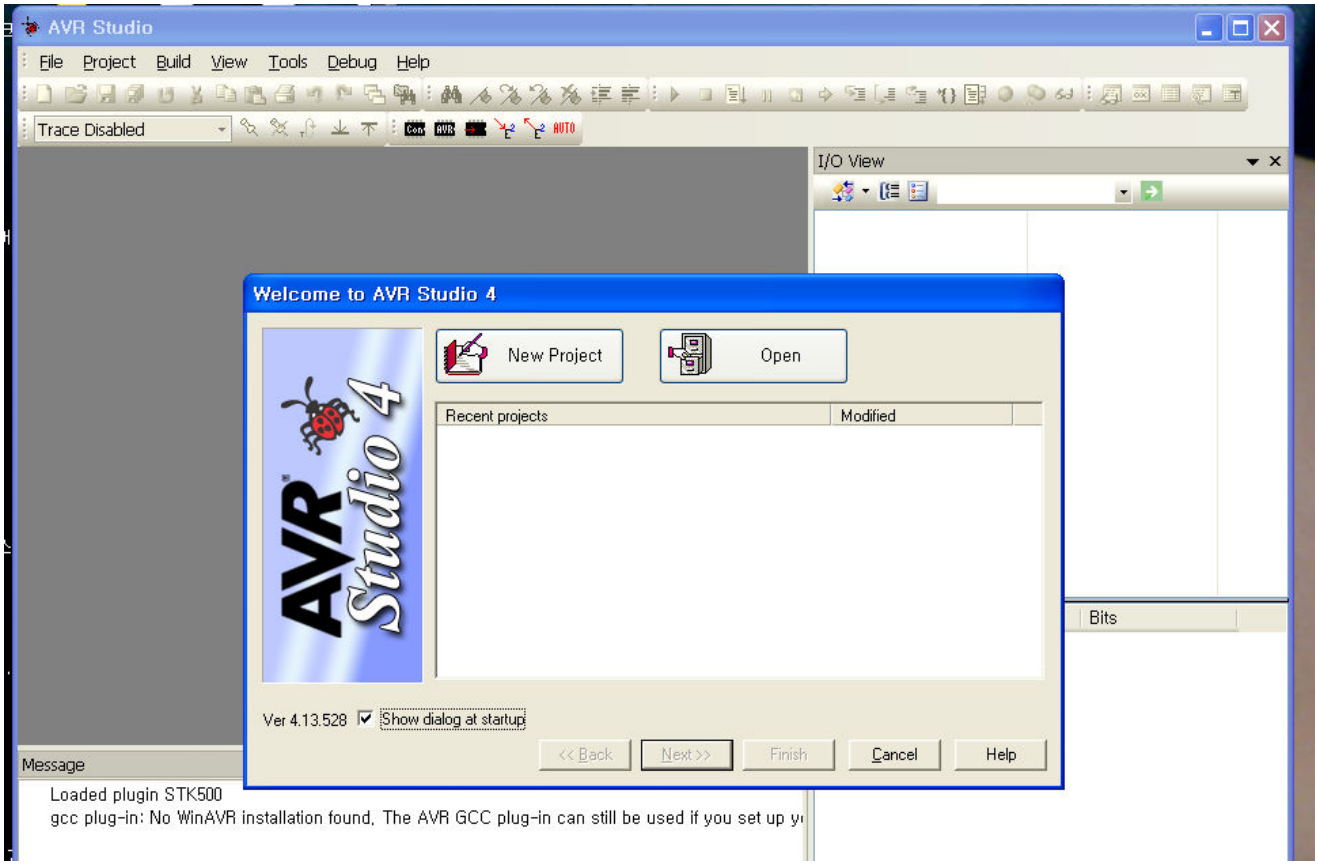


그림 37. AVR Studio의 초기화면

② **New Project 실행 :** Welcome to AVR Studio 4라는 이름의 작은 창의 메뉴 [New Projct]를 클릭하면 그림 38과 같은 창이 뜨게 된다. 이 창에서 프로젝트 형태(어떤 어셈블러를 사용할 것인가를 결정), 프로젝트 이름, 어셈블리어 파일 이름, 파일들이 위치할 디렉토리를 결정할 수 있다.

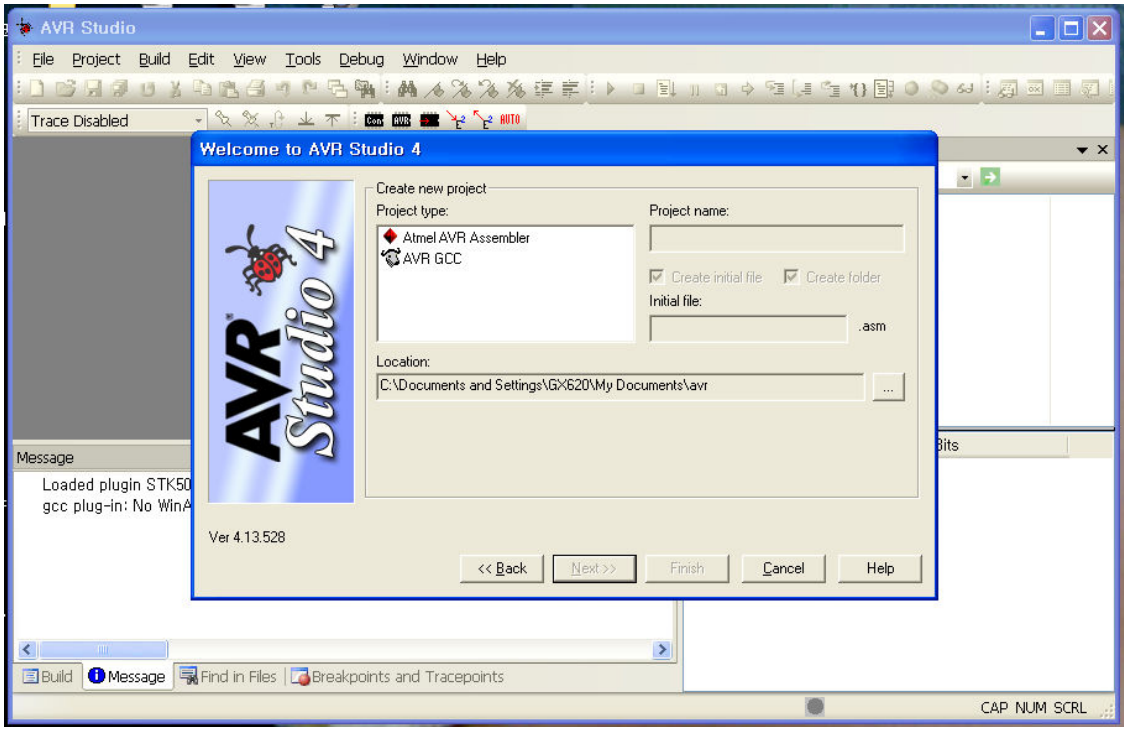


그림 38. New Project의 실행

③ **프로젝트의 설정** : 그림 39와 같이 메뉴에서 Project type로 Atmel AVR Assember을 선택하고 Project name와 Initial File, Location은 적절한 값으로 설정한다(여기에서는 각각 hhchoi, hhchoi, C:₩Documents and Settings₩GX620₩My Documents₩avr₩at128로 설정했다). 그러면 그림 39처럼 [Next] 메뉴가 활성화된다.

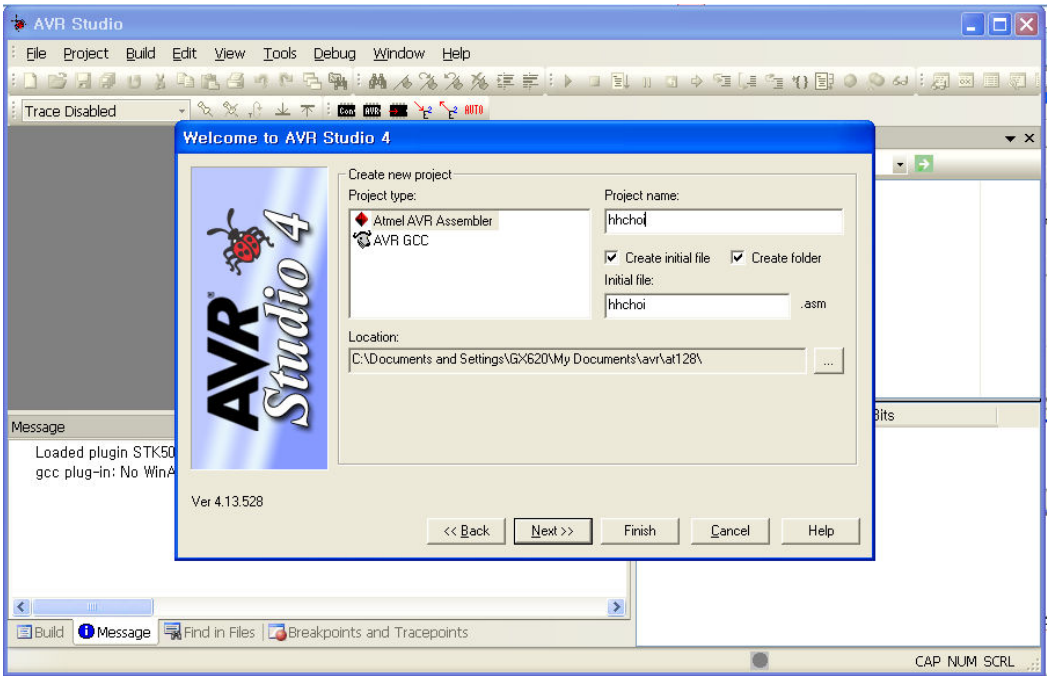


그림 39. 프로젝트의 설정

④ Debug 플랫폼과 디바이스 설정 : Project name와 Initial File, Location은 적절한 값으로 설정하고 [Next] 메뉴를 누르면 그림 40과 같이 디버그 플랫폼과 디바이스를 설정하는 창이 뜬다. Debug platform은 AVR Simulator를 선택하고 Device는 ATmega128을 선택하고 마지막으로 [Finish] 메뉴를 누른다.

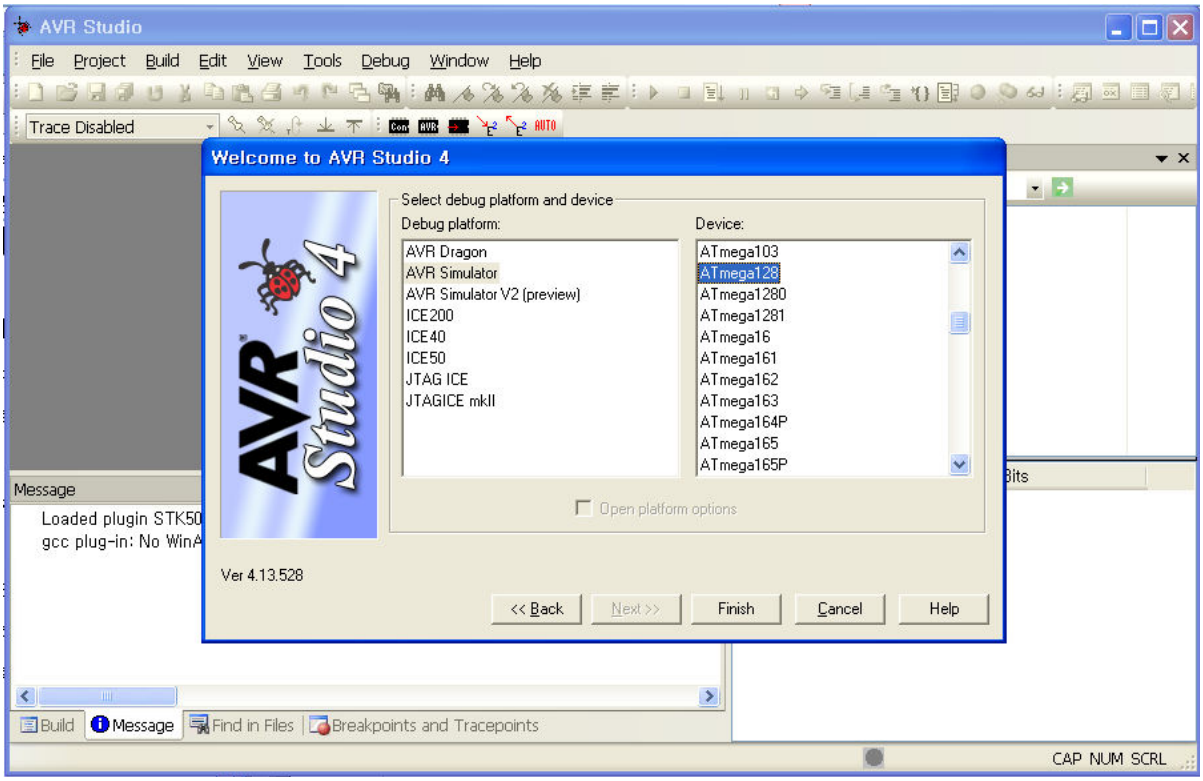


그림 40. 디버그 플랫폼과 디바이스의 설정

⑤ 에디터 창 생성 : 디버그 플랫폼과 디바이스를 설정하고 [Finish]메뉴를 누르면 그림41과 같이 프로젝트 설정단계에서 지정한 어셈블리 파일의 코드를 작성할 수 있는 파일 이름을 갖는 에디터 창과 Project라는 이름의 조그만 창이 뜬다.

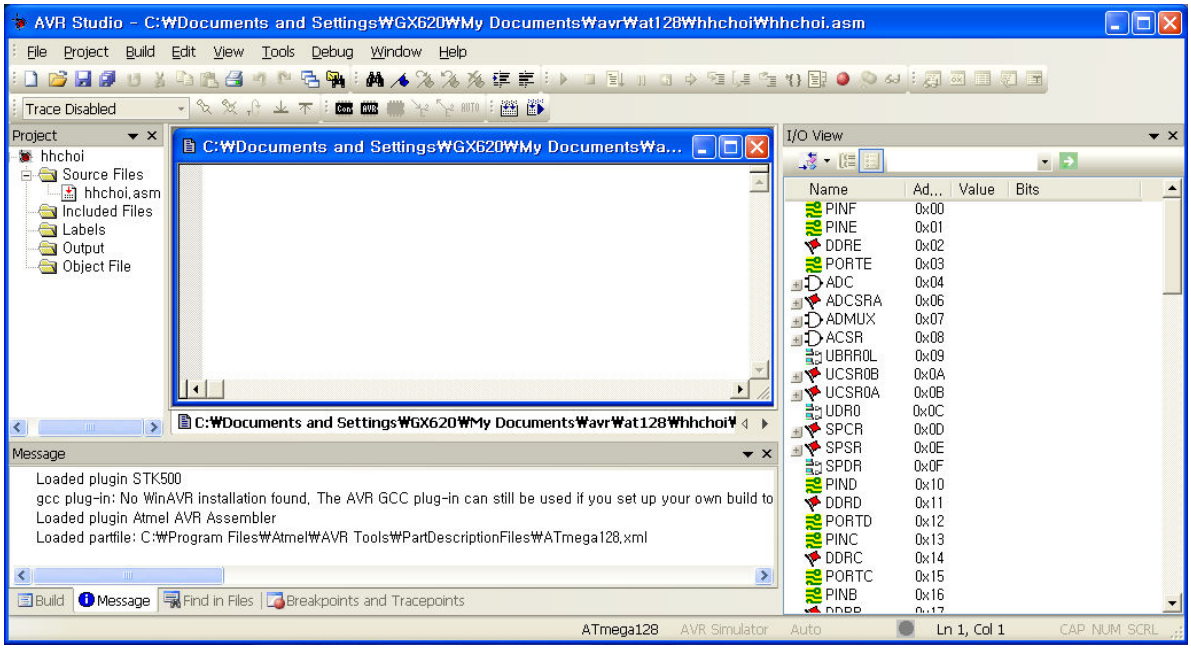


그림 41. 프로젝트 등록의 최종 결과로 생긴 에디터 창

⑥ **Build 실행** : 어셈블리 파일의 코드를 작성한 이후에 메인창의 [Build]메뉴를 그림 42처럼 선택하여 실행시킨다.

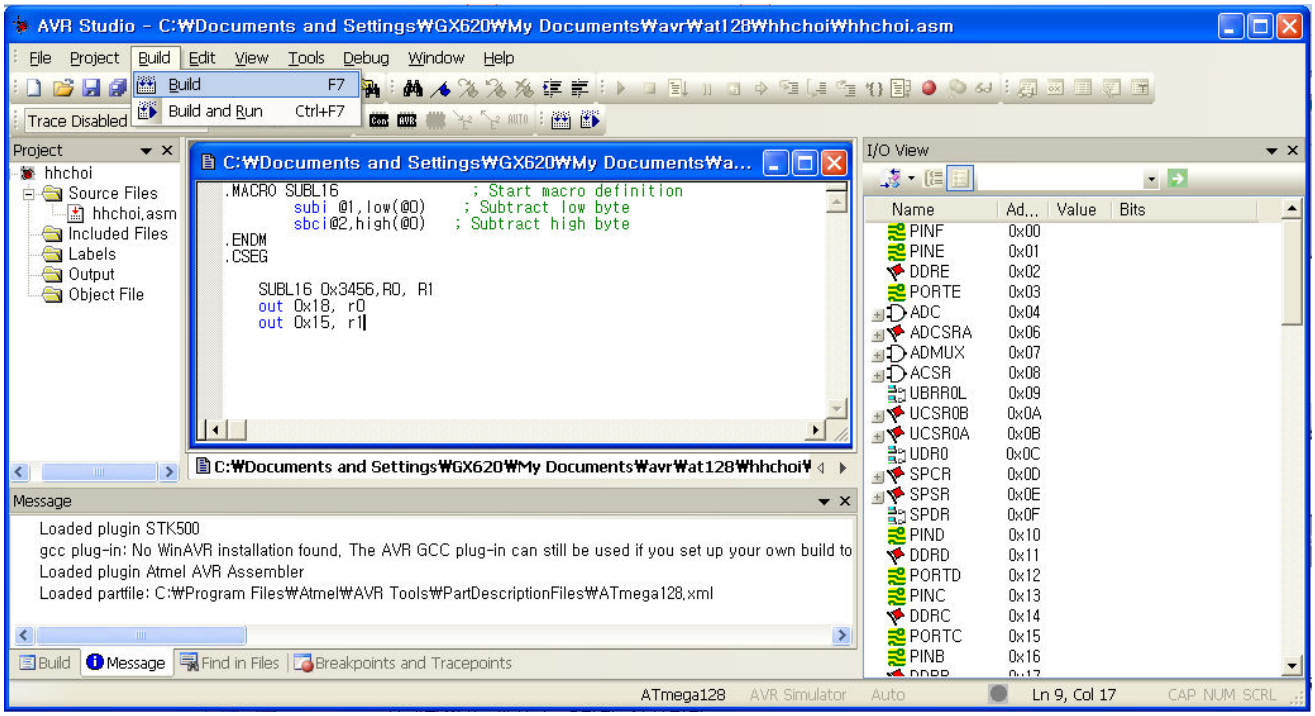


그림 42. Build 실행

⑦ **에러없이 종료된 결과 :** 그림 43은 에러없이 실행파일이 만들어진 경우를 보여준다. Build라는 이름의 창에 실행파일을 생성한 결과가 보여진다.

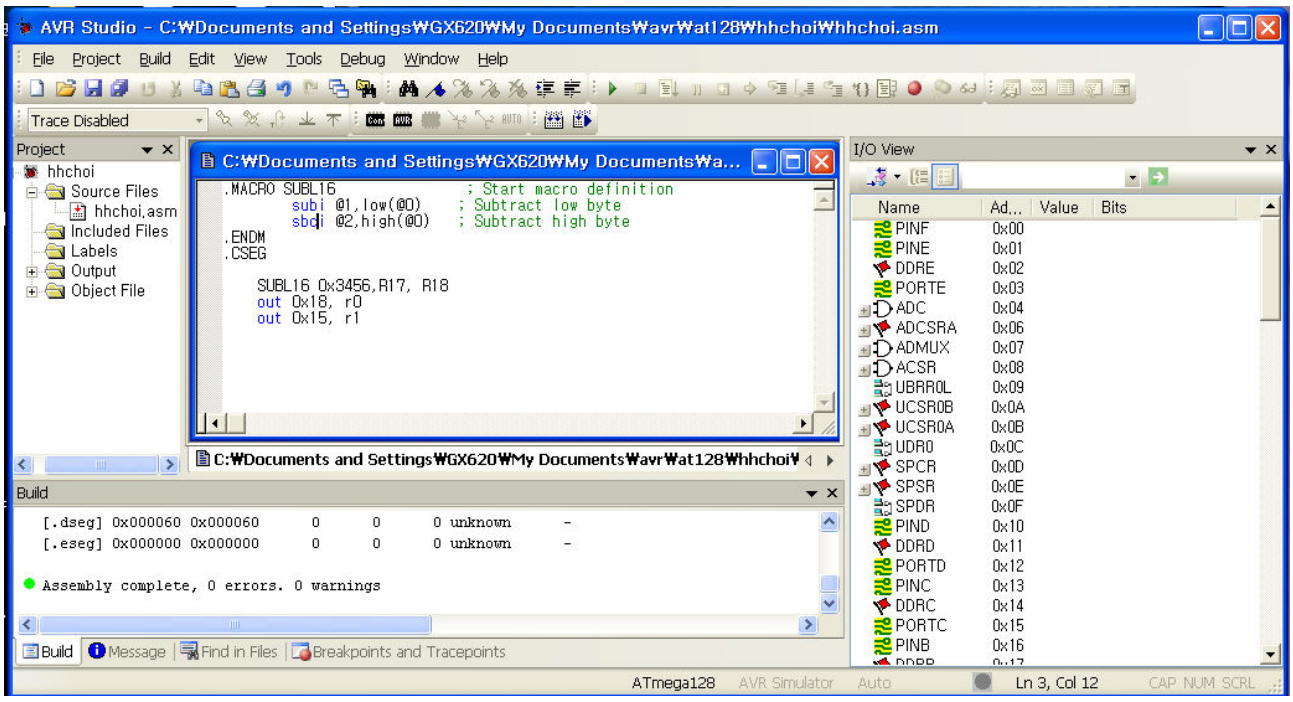


그림 43. 성공적으로 실행파일이 만들어진 결과

⑧ 에러가 있을 때 결과 : 그림 44는 실행파일을 만드는 과정에서 문법적인 에러가 발생한 경우를 보여준다. Build창에 에러가 발생한 행번호와 에러메시지가 나타나 있음을 알 수 있다. 에러메시지 부분을 더블클릭하면 에러가 발생한 위치로 커서가 자동적으로 이동하여 손쉽게 에러를 수정할 수 있게 한다.

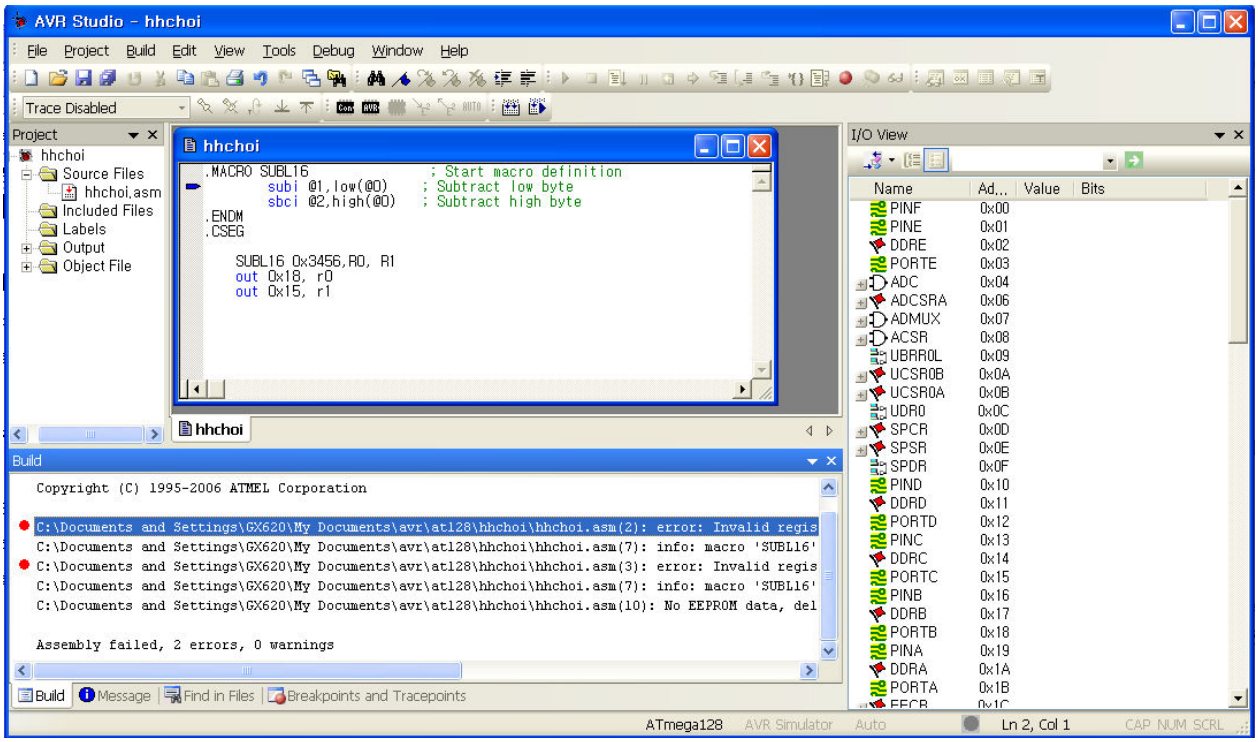


그림 44. 에러가 있을 때 결과

2.3.2. 디버깅과 시뮬레이션

① Debug 선택 : 문법적인 에러가 없어 실행파일이 만들어진 경우 그림 45과 같이 [Debug->Start Debugging] 메뉴를 선택하면 디버깅과 간단한 시뮬레이션을 할 수 있다.

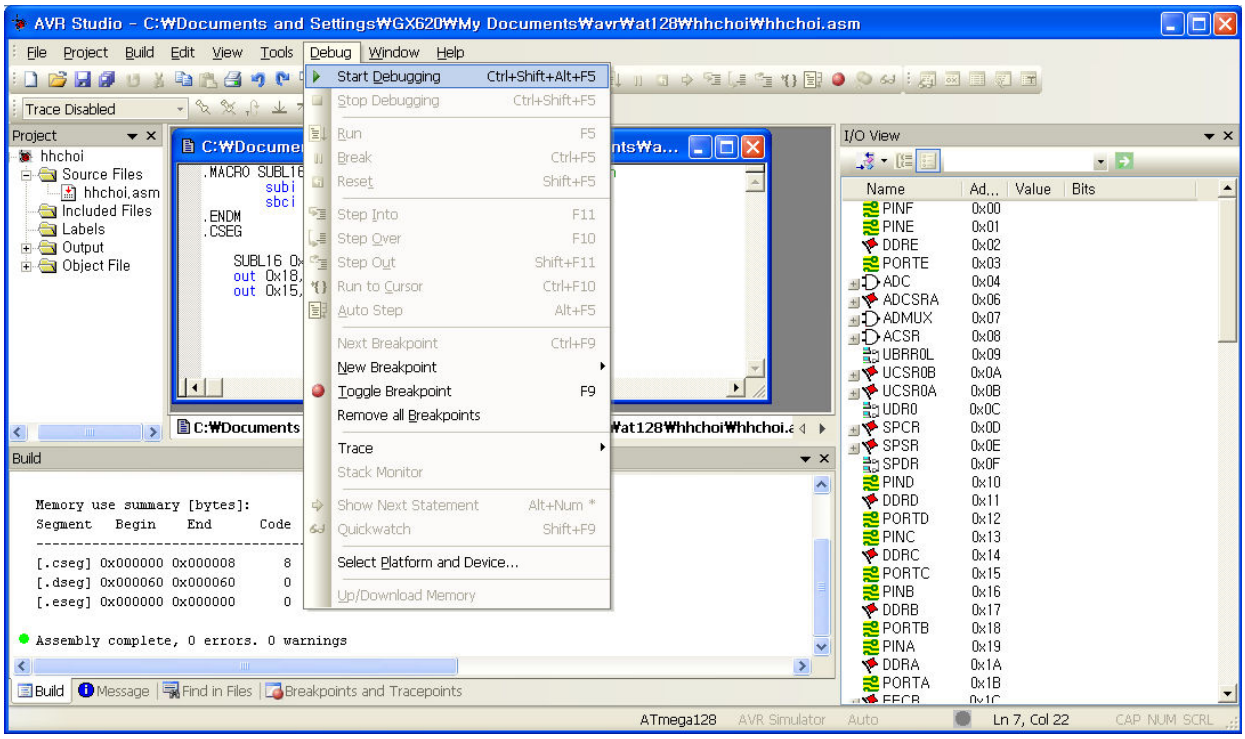


그림 45. [Debug->Start Debugging] 메뉴

② **Debug 시작 :** 디버그세션을 시작하면 46과 같이 I/O View창이 I/O레지스터의 값들을 비트별로 그래픽하게 확인할 수 있도록 변하고 에디터 창에 노란 화살표가 나타난다.

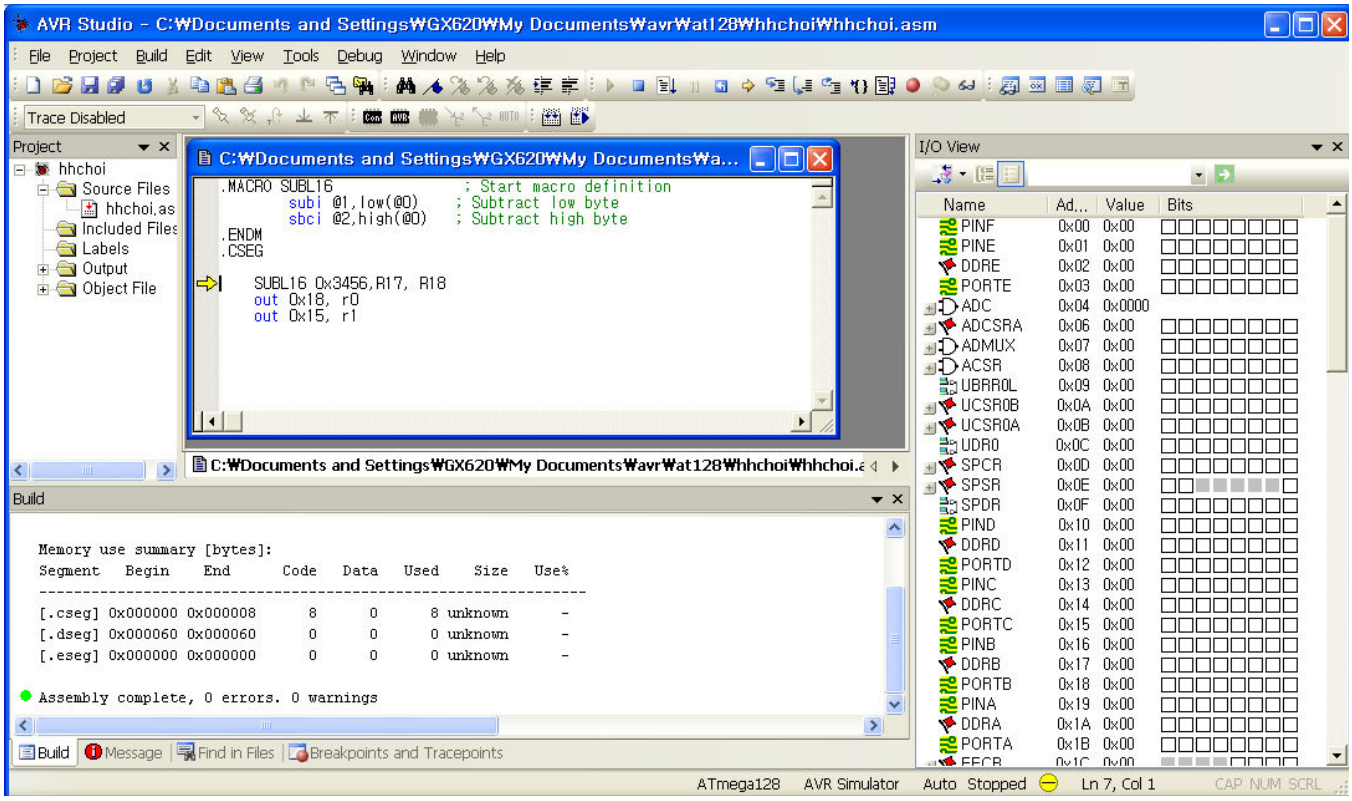


그림 46. Debug 세션의 시작

③ **View->Register 실행 :** 디버그 세션에서 메인메뉴의 [View->Register]를 실행하면 그림 47처럼 Register라는 이름의 창이 뜬다. 디버깅하며 값이 바뀌는 R0~R31 레지스터의 값이 Register창에 나타난다.

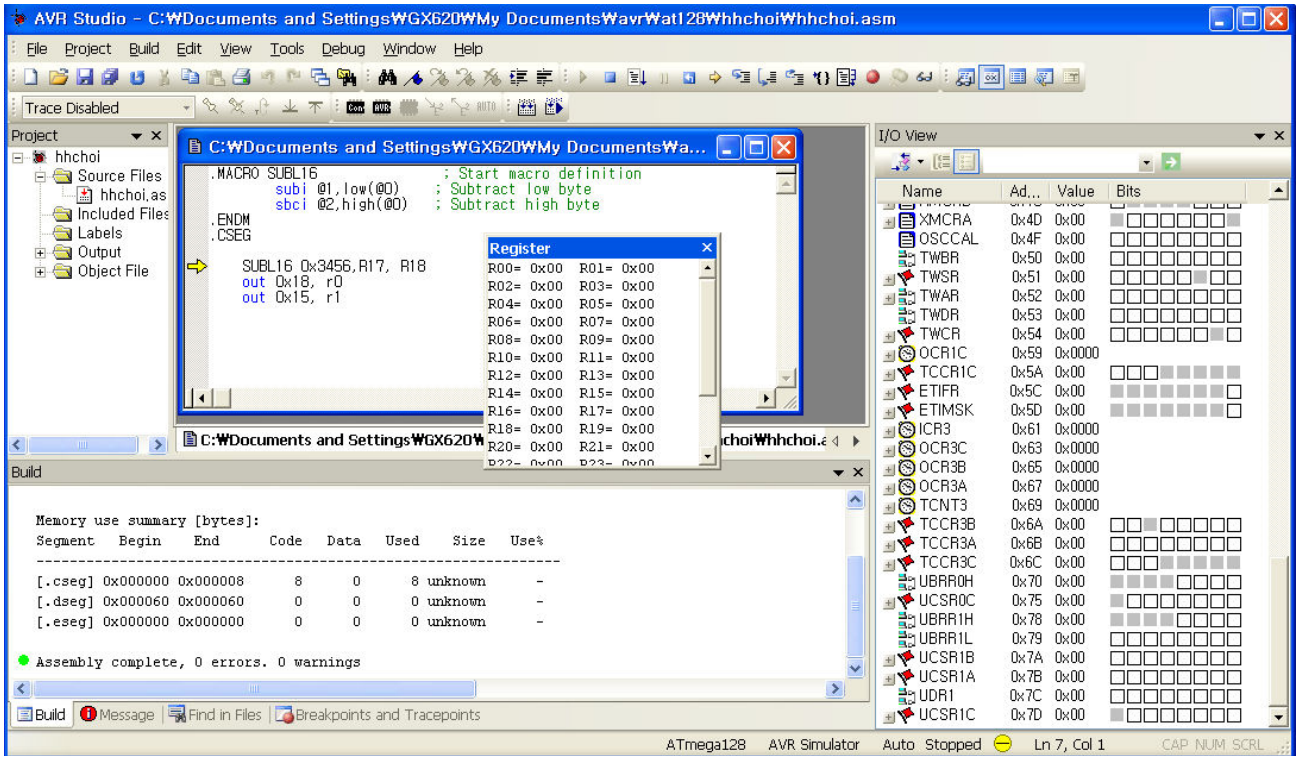


그림 47. 디버그 세션에서 메인 메뉴 [View->Register]의 실행

④ **View-> Memory 실행** : 디버그 세션에서 메인메뉴의 [View->Memory]를 실행하면 그림 48처럼 Memory라는 이름의 창이 뜬다. 그 창에는 그림 48에 보여진 것처럼 [Data], [EEPROM], [Extended I/O], [I/O], [Program], [Register]의 서브메뉴가 있는데 이를 이용하여 SRAM 데이터 메모리, EEPROM, I/O레지스터, 프로그램 메모리, R0~R31 레지스터의 값 중 특정한 어느 하나를 선택하여 변화를 모니터할 수 있다. [View->Memory]와 같은 역할을 하는 [View->Memory2], [View->Memory3] 메뉴가 더 존재한다.

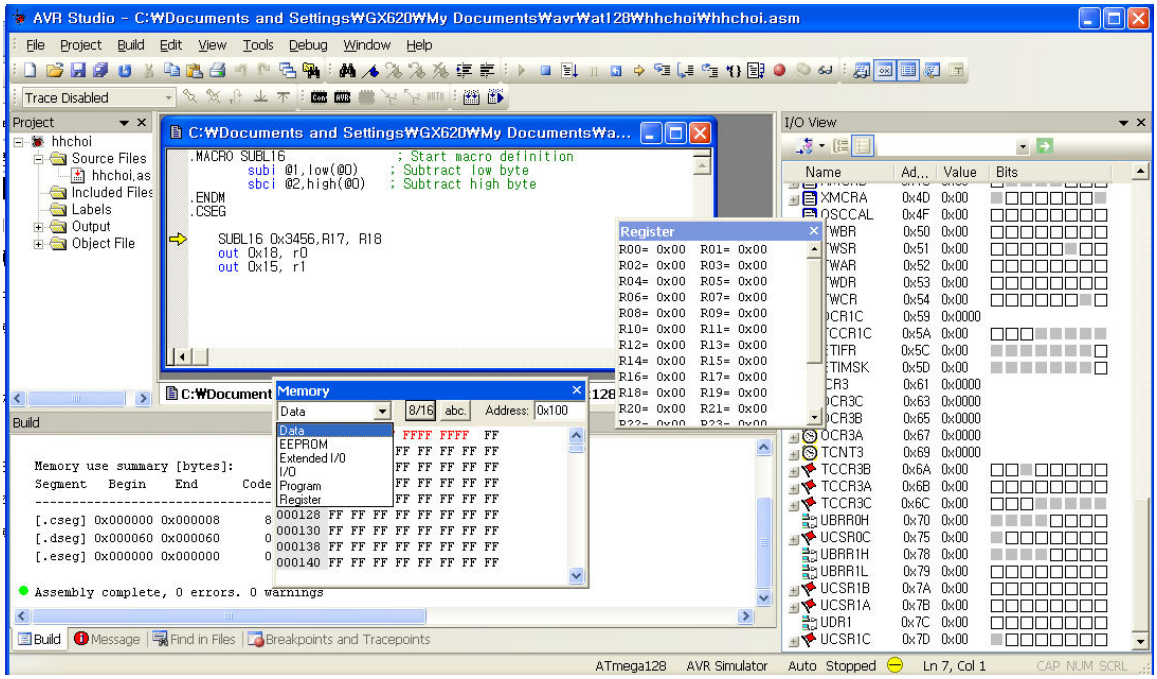


그림 48. 디버그 세션에서 메인 메뉴 [View->Memory]의 실행

⑤ **View-> Watch 실행** : 디버그세션에서 메인메뉴의 [View->Watch]를 선택하면 Watch라는 자그만 창이 뜨는데 그림 49처럼 Name 에 레지스터나 심볼값을 입력하고 이들 값의 변화를 모니터할 수 있다.

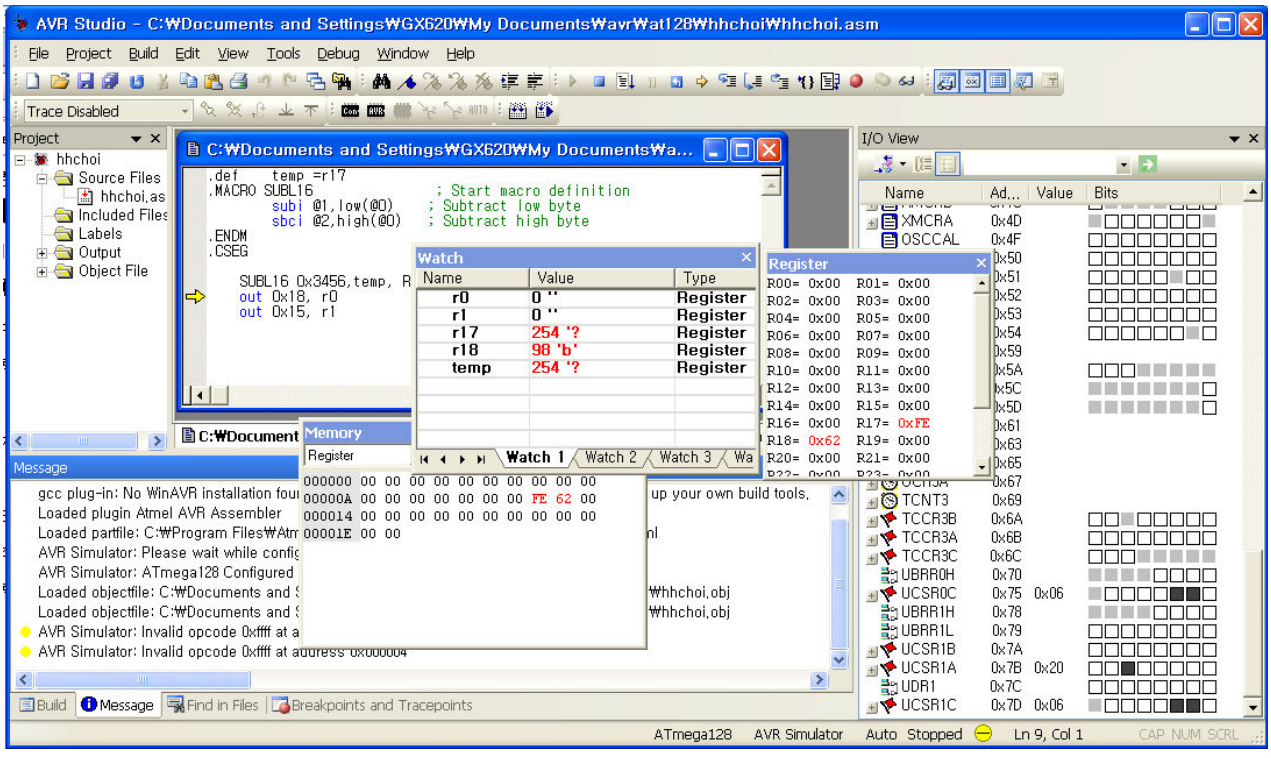


그림 49. 디버그 세션에서 메인 메뉴 [View->Watch]의 실행

⑥ **디버그세션에서 여러 가지 메뉴 :** 디버그세션에서 메인메뉴 [Debug]의 하위 메뉴에는 그림 50처럼 여러 가지가 있는데 이중 [Run to Cursor] 메뉴는 브레이크포인트 또는 커서가 놓인 곳까지 실행하도록 한다. [Step Into]는 한 명령씩 실행하도록 하며 [Step Over]는 서브루틴 콜도 한 명령 수행처럼 수행하도록 하고 [Auto Step]는 Studio4에서 알아서 적당한 지점까지 실행한다. 디버그세션에서 메인메뉴의 [Goggle Breakpoint]를 선택하면 커서가 놓인 곳을 브레이크포인트로 설정하거나 해제한다. 그 이외에 다양한 메뉴를 활용해 손쉽게 디버깅할 수 있다.

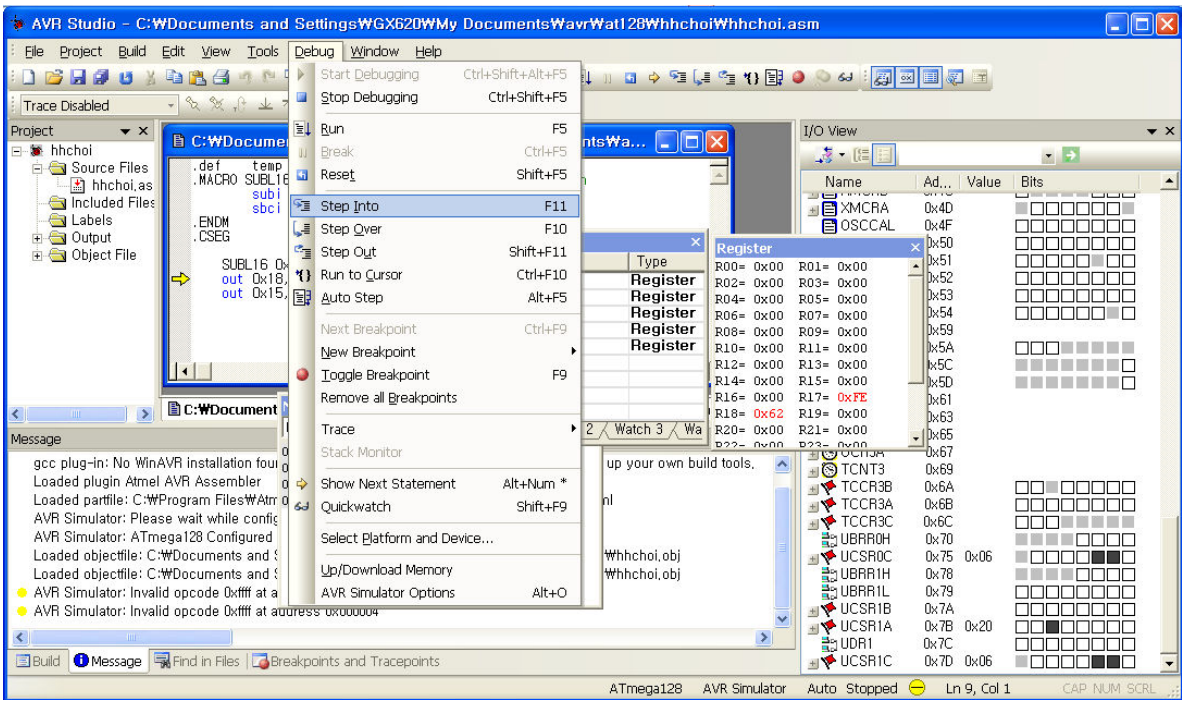


그림 50. [Debug]메뉴의 하위메뉴들

2.4. AVR의 ISP

2.4.1. 마이컴 시스템 개발 방법

2.4.1.1. ICE (In Cricuit Emulator)를 이용하는 법

- **ICE의 역할 :** 개발중인 Target System의 마이컴 역할을 임시적으로 대신해주도록 타겟 시스템과 동일한 마이컴을 내장한 시스템
- **연결 :** 타겟 시스템의 마이컴을 빼고 그 자리에 ICE의 출력핀을 꼽고 한편 ICE는 호스트 컴퓨터와도 연결된다.
- **동작 :** 소스코드 프로그램을 호스트 컴퓨터에서 PK51과 같은 소프트웨어 개발툴을 사용해서 디버깅하고 실행파일을 만들어 ICE에 다운로드 시키면 ICE는 자신의 램에 저장하고 그 프로그램을 자신의 마이컴으로 실행시키는데 그 핀들이 출력핀으로 타겟 시스템 마이컴 핀자리에 대신 들어가 있어 결국 타겟 시스템의 마이컴 역할을 수행한다.
- **특징 :** ICE는 싱글 스텝 동작, 역어셈블링, 브레이크 기능 등 고기능으로 시스템 개발용에 가장 편리한 장비이나 고가로 교육용에는 부적합하다.

2.4.1.2. ROM Emulator를 이용하는 법

- **롬에뮬레이터의 역할 :** 개발중인 타겟 시스템의 프로그램 메모리 역할을 임시적으로 대신해주는 마이컴 시스

템

- **연결 :** 타겟 시스템의 프로그램 메모리를 빼고 그 자리에 롬에뮬레이터 출력핀을 꼽고 한편 호스트 컴퓨터와도 연결된다.
- **동작 :** 소스코드 프로그램을 호스트 컴퓨터에서 PK51과 같은 소프트웨어 개발툴을 사용해서 디버깅하고 실행파일을 만들어 롬에뮬레이터에 다운로드 시키면 롬에뮬레이터는 자신의 램에 저장하고 롬에뮬레이터 출력핀들이 타겟 시스템 프로그램 메모리 핀자리에 대신 들어가 있으므로 결국 타겟 시스템의 롬 역할을 수행한다.
- **특징 :** Z80, 8031과 같이 프로그램메모리가 외장형인 경우에는 적절한 개발방법이 될 수 있으나 롬이 내장된 8051과 같은 형태는 일부러 롬을 장착해야하므로 하드웨어적으로 복잡해지는 단점이 있다. 롬에뮬레이터는 롬을 대신하는 것이라 ICE에 비해 저가의 시스템이다.

2.4.1.3. ROM Writer를 이용하는 법

- **롬라이터의 역할 :** 실행파일를 PROM에 기록해주는 마이컴 시스템
- **연결 :** 타겟 시스템의 롬을 빼서(롬 내장형 마이컴의 경우에는 마이컴을 뺀다) 롬라이터에 장착하고 호스트 컴퓨터와 연결한다.
- **동작 :** 소스코드를 호스트 컴퓨터에서 PK51과 같은 소프트웨어 개발툴을 사용해서 디버깅하고 실행파일을 만들어 롬라이터에 다운로드 시키면 롬라이터는 장착된 롬에 프로그램을 기록하고 기록이 다 끝나면 롬을 빼서 타겟시스템에 장착하고 프로그램을 실행한다.
- **특징 :** 프로그램을 교체할 때 마다 타겟 시스템의 롬을 빼어, 롬 소거, 롬 기록, 다시 롬의 장착과 실행을 되풀이해야하므로 엄청난 시간과 노력이 소모되며 실행하며 타겟 시스템에서의 상황을 모니터하기 어려워 디버깅도 힘이드나 ICE나 롬에뮬레이터에 비해 롬라이터는 저가이다.

2.4.1.4. 외부 RAM을 이용하는 법

- **외부 램의 역할 :** 타겟 시스템에 여분의 외부램을 장착하고 이를 개발할 프로그램을 위한 메모리로 사용
- **연결 :** 롬에 호스트 컴퓨터와 통신을 위한 모니터프로그램이 기록된 타겟 시스템과 호스트 컴퓨터를 연결한다. 호스트 컴퓨터에서 타겟시스템과 통신을 위한 터미널프로그램으로는 윈도우 기본 프로그램중 하나인 하이퍼터미널이 널리 이용된다.
- **동작 :** 소스코드를 호스트 컴퓨터에서 소프트웨어 개발툴을 사용해서 디버깅하고 실행파일을 만들어 하이퍼터미널과 같은 터미널프로그램을 사용하여 모니터프로그램이 기록된 타겟 시스템에 다운로드 시키면 타겟 시스템은 이를 외부램에 저장하고 이를 실행한다.
- **특징 :** 호스트 컴퓨터인 PC의 통신용 소프트웨어(예: 하이퍼터미널)를 통해 타겟 시스템과 통신할 수 있고 실행 상황을 모니터할 수 있어 값싸고 실용적인 방법이나 일부러 램을 장착하고 미리 모니터프로그램을 기록해야하는 등이 불편하다. 교육용으로 많이 사용된다.

2.4.1.5. ISP를 이용하는 법

- **ISP의 역할 :** AVR은 내장 플래시롬의 SPI (Serial Peripheral Interface) 인터페이스를 제공한다. SPI는 오직 3라인을 이용한 통신 방법으로 MOSI(Master Out Slave In), MISO(Master In Slave Out), SCLK(Serial CLocK) 신호를 이용한다. Motolora에서 개발되었으며 Master와 Slave가 SCLK에 동기하여 데이터를 교환하는 방식으로 마이컴의 RST(리셋) 핀을 0으로 한 상태에서 앞의 세 시그널을 이용하여 플레시롬의 데이터를 읽고 쓰기가 가능하다. 즉 내부에 롬라이터와 같은 기능을 하는 부분이 내장되어 있어 내부 플래시롬을 롬라이터없이 읽고 쓰기가 가능하므로 SPI가 지원되면 따로 롬라이터를 이용하지 않고도 PCB 기판 상에 마이컴을 실장하고 전원과 클럭이 공급되는 상태에서 프로그래밍이 가능한 ISP (In System Programming) 기능이 제공된다.
- **연결 :** 타겟 시스템과 호스트 컴퓨터를 값싼 ISP용 인터페이스를 사용하여 연결한다.
- **동작 :** 소스코드를 호스트 컴퓨터에서 PK51과 같은 소프트웨어 개발툴을 사용해서 디버깅하고 실행파일을 만

들어 호스트 컴퓨터에 설치한 ISP용 소프트웨어를 통해 PCB 기판 상에 마이컴을 실장하고 전원과 클럭이 공급되는 상태에서 프로그래밍한다.

- **특징 :** 약간의 회로 추가로 시스템의 디버깅이 용이하고 업그레이드가 쉬운 시스템을 만들 수 있다는 장점이 있다. AT89S8252, AT89Sxx나 AVR처럼 내부에 플래시롬이 내장되어 있고 SPI를 지원하는 마이컴을 사용해야 하며 부피가 커지고 제품 단가가 올라간다는 단점이 있다.

2.4.2. AVR ISP용 다운로더

2.4.2.1. AVR Studio와 ATAVRISP

Atmel사의 AVR Studio4는 AVR 마이컴의 프로그램을 개발하고 디버깅할수 있는 윈도우기반의 소프트웨어 개발툴로 통합 개발환경을 제공하는데 AVR뿐만 아니라 AT89S51과 AT89S52의 플래시롬을 프로그래밍하는 ISP기능도 지원한다. AVR Studio를 이용한 ISP를 위해서는 Atmel사의 ATAVRISP라는 직렬통신 케이블을 사용해야 하는데 ATAVRISP는 호스트컴퓨터인 PC의 시리얼포트와 실장된 마이컴의 SPI신호핀간의 RS-232직렬인터페이스를 제공한다. AVR Studio는 Atmel사의 홈페이지인 http://www.atmel.com에서 공짜로 다운로드 받을 수 있는 반면에 ATAVRISP는 구입해야 한다.

2.4.2.2. Atmel AVR ISP 프로그래머

Atmel사의 AVR ISP 프로그래머는 AVR 마이컴의 내장 플래시롬을 ISP하기위한 윈도우즈 기반의 소프트웨어로 MCS-51호환기종인 AT89S8252와 AT89S53의 ISP도 가능하다. AVR ISP 프로그래머를 이용한 ISP는 호스트컴퓨터인 PC의 프린터포트와 타겟시스템 마이컴의 SPI신호핀간 인터페이스회로가 필요한데 그림 51과 같이 74HC244가 들어가 있어 신호를 제어하는 매우 간단한 회로면 충분하다. 여기에서 ISP CON은 PCB 기판 상에 실장되어 전원과 클럭이 공급되는 상태에서 AVR 과 연결한다. AVR의 ISP를 할 때는 딥스위치를 on으로 하여 VCC에서 공급되는 전원을 74HC244에 넣어 작동시키고 ISP를 다 끝낸 경우에는 딥스위츠를 off로 하여 74HC244를 꺼버려 PC와 마이컴의 연결을 차단한다. 그림에서 LED는 ISP가 진행중일 때 불이 들어와 알리는 기능을 하는 것으로 프린터 포트 8 번핀(D6), 74HC244의 4번과 16번핀, LED와 저항은 굳이 연결 안해도 된다. AVR ISP는 Atmel사의 홈페이지인 http://www.atmel.com에서 공짜로 다운로드 받을 수 있다.

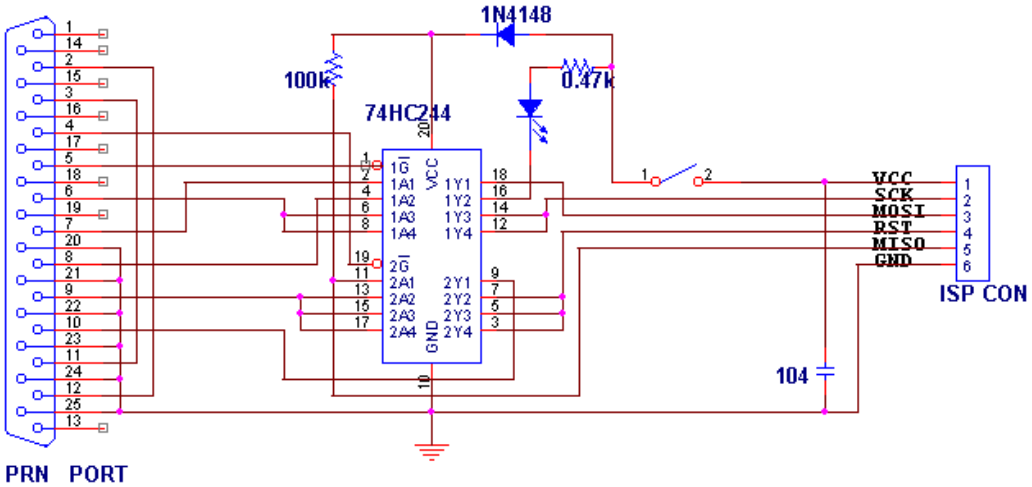


그림 51. Atmel AVR ISP, PonyProg2000, CodevisonAVR용의 ISP 인터페이스 회로

2.4.2.3. PonyProg2000

PonyProg2000은 윈도우즈기반의 직렬통신을 통한 여러 다양한 마이컴과 PROM을 읽고 쓰기위한 공짜 소프트웨어로 http://www.lancos.com에서 무료로 다운로드 가능하다. PonyProg2000는 원래 직렬통신을 위한 것이지만 그림 51에 주어진 Atmel AVR ISP용 인터페이스회로를 사용해서 PC의 프린터포트와 타겟시스템 마이컴의 SPI신호핀간에 연결해도 되고 http://www.lancos.com에 주어진 직렬포트용 인터페이스회로를 사용해도 된다.

2.4.2.4. Codevisin AVR

비교적 저렴한 수십만원대의 컴파일러로 2K바이트까지 용량 제한이 있는 평가판을 인터넷에서 자유롭게 다운로드 받아서 사용가능한 Codevision AVR C 컴파일러와 그림 51의 ISP용 인터페이스회로를 사용해서 ISP가 가능하다. 연결할 때 ISP CON의 VCC는 AVR의 VCC와, ISP CON의 SCK는 AVR 마이컴의 SCK PB1(SCK, 핀11), ISP CON의 MOSI는 AVR의 PE0(PDI/RXD0, 핀9), ISP CON의 MISO는 PE1(PDO/TXD0, 핀8), ISP CON의 RST는 AVR의 $\overline{\text{RESET}}$(핀20)과 연결한다. 프로그램 다운로드 하는 과정은 다음 절에서 설명할 것이다.

2.5. Codevision AVR C 컴파일러

- AVR용 C 컴파일러들 : 다음과 같은 3가지 정도가 많이 사용된다.

① IAR사의 EWAVR C 컴파일러 : 수백만원대의 고가의 컴파일러이지만 상대적으로 뛰어난 압축률과 안정된 기능을 내장하고 있어 산업 현장에서 많이 사용되고 있다. 2K바이트까지 용량 제한이 있는 평가판을 인터넷에서 자유롭게 다운로드 받아서 사용가능하다.

② AVRGCC 컴파일러 : 기본적으로 text환경에서 동작하도록 만들어진 꽁짜의 컴파일러이나 ANSI C언어와 다른 점이 많은 편이라 ANSI C언어에 익숙한 사람은 불편함을 느낄 수 있다.

③ HPinfotech사의 Codevision AVR C 컴파일러 : 비교적 저렴한 수십만원대의 컴파일러로 ANSI C언어의 모든 구성요소와 거의 가깝게 구현되어 있다. 2K바이트까지 용량 제한이 있는 평가판을 HPinfotech사의 홈페이지 http://www.hpinfotech.ro에서 자유롭게 다운로드 받아서 사용가능하다.

- Codevision AVR C 컴파일러의 특징은 다음과 같다. ① AVR Studio4와 연계하여 디버깅이 가능하다. ② ISP 다운로더 프로그램을 내장하였다. ③ 프로그램을 자동으로 생성할 수 있는 CodeWizardAVR를 내장하였다. ④ ANSI C언어의 표준 라이브러리 이외에 캐릭터 LCD, 필립스 I2C 버스, RTC DS1302, EEPROM DS2430, SPI용, 지연함수 등을 위한 AVR에 특화된 다양한 라이브러리를 제공한다.

2.5.1. 새 프로젝트 등록하기와 옵션 설정하기

① 초기화면 : 평가판을 http://www.hpinfotech.ro에서 내려 받은 후 실행파일 setup_eval.exe을 실행하면 그림 52와 같이 [File], [Edit], [View], [Project], [Tools], [Settings], [Windows], [Help]의 메뉴를 갖고 CodeVisonAVR이라는 이름을 가진 초기화면이 나타난다.

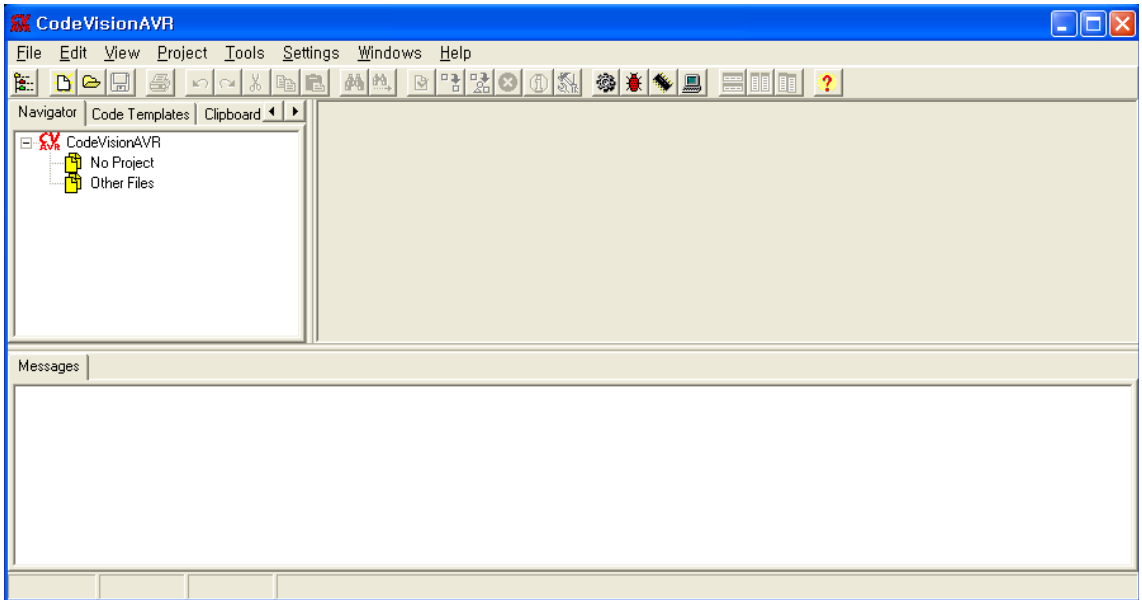


그림 52. CodevisionAVR Compiler의 초기화면

② **New Project 실행 :** [File->New] 메뉴를 선택하면 Create New File이라는 작은 창이 생성된다. 그 때 그림 53과 같이 project를 선택하고 OK버튼을 클릭한다.

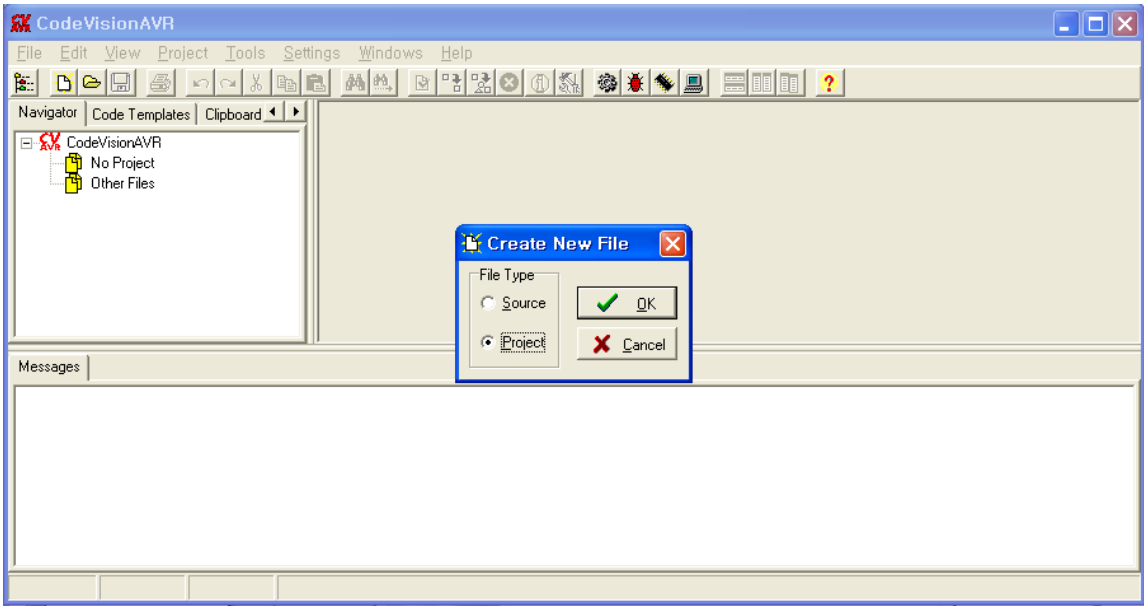


그림 53. [File->New] 메뉴의 실행과 새로운 프로젝트 생성

③ **CodeWizardAVR 사용 여부의 설정** : Create New File창에서 project를 선택하고 OK버튼을 클릭하면 그림 54와 같이 CodeWizardAVR의 사용여부를 묻는 Confirm창이 생성되는데 No를 선택한다. CodeWizardAVR은 입출력포트의 방향, AVR 칩의 종류와 주파수, 외부 인터럽트, 타이머, AD변환기, 동기 및 비동기 통신, LCD 등을 대화창을 통해 쉽게 설정할 수 있는 위저드 기능을 제공하여 쉽게 프로그램을 할 수 있도록 하는데 스스로 공부하면 쉽게 알 수 있으므로 여기에서는 설명을 따로 하지 않고 일단 CodeWizardAVR를 사용하지 않는 것으로 하여 설명한다.

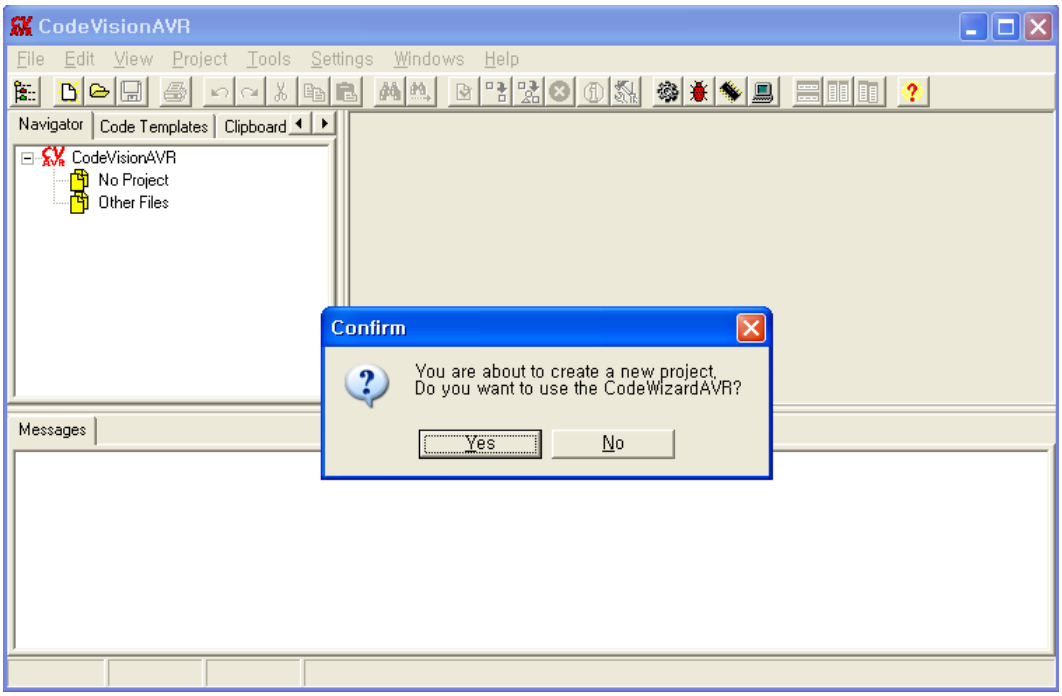


그림 54. CodeWizardAVR 사용 여부를 설정하기 위한 창

④ 프로젝트 이름 설정 : CodeWizardAVR의 사용여부를 묻는 Confirm창이 생성되어 No를 선택하면 그림 55처럼 Create New Project라는 이름의 대화상자가 생성되는데 대화상자내의 [저장위치(I)]메뉴에서 적당한 위치를 선택하고 자신이 원하는 프로젝트 이름(예:test)을 [파일이름(N)]메뉴의 입력창에 기입한 후에 [저장(S)] 버튼을 클릭한다.

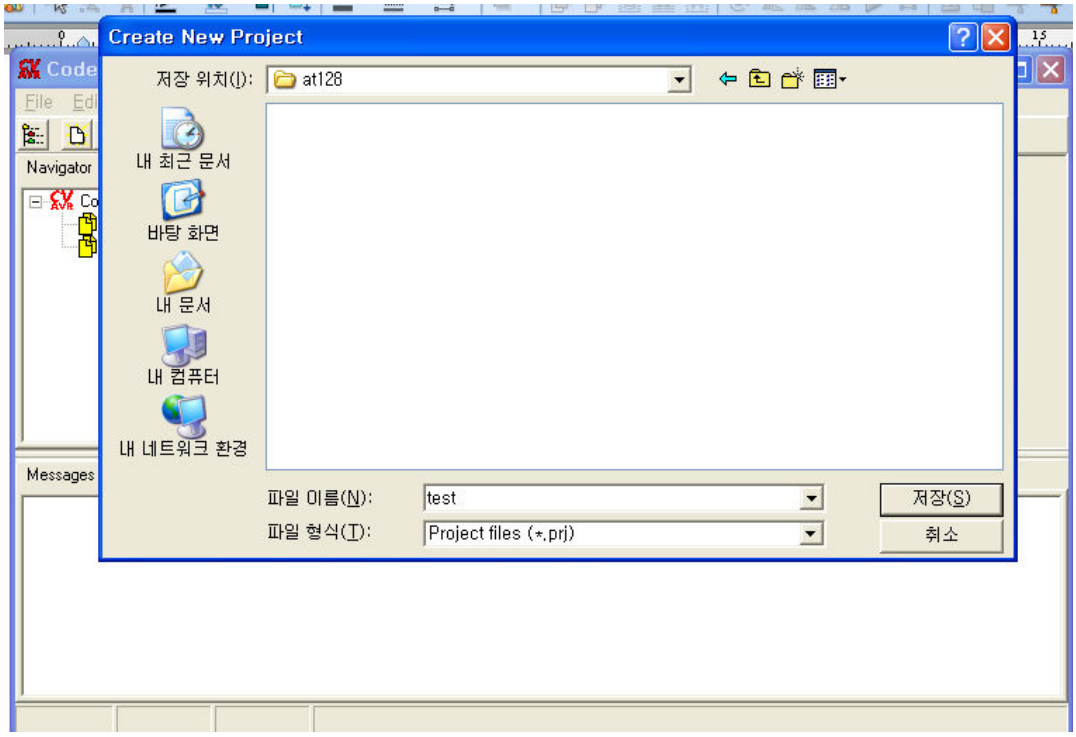


그림 55. CodevisionAVR에서 새로운 프로젝트 이름의 설정

⑤ 프로젝트 옵션 설정하기 : 프로젝트 이름을 설정(예:test)하고 나면 [Files], [C Complier], [Before Make], [After Make] 메뉴를 갖고 있는 Configure Project test.prj라는 이름의 대화창이 생성된다. 16MHz의 주파수를 사용하는 ATmega128을 대상으로 하는 경우 그림56 처럼 설정하면 된다.

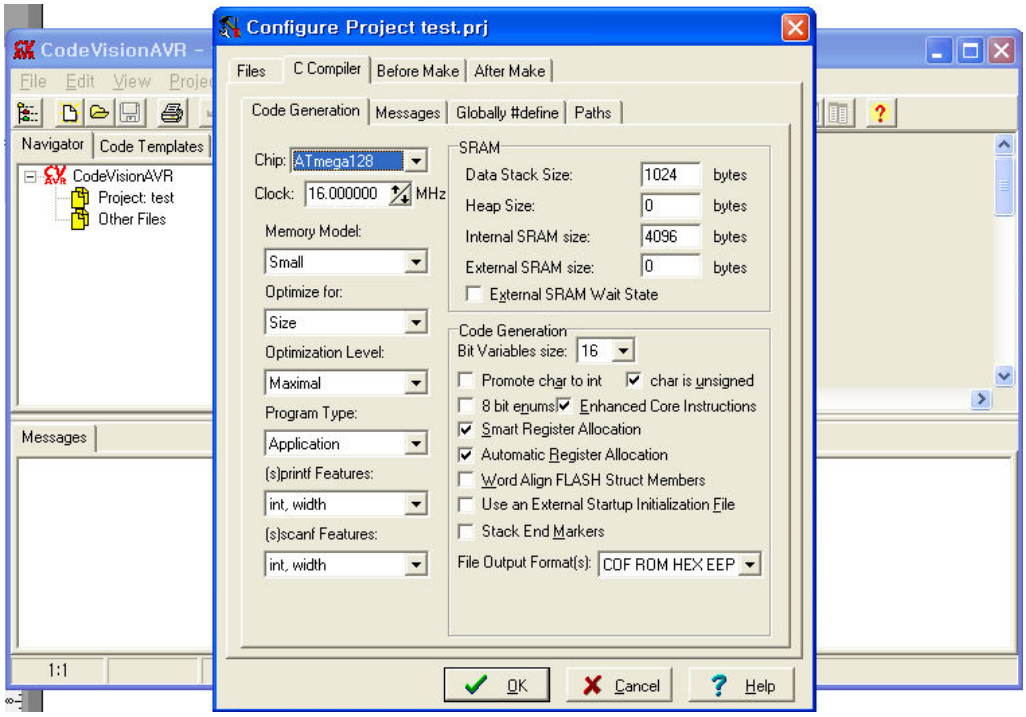


그림 56. CodevisionAVR에서 프로젝트 옵션 설정하기

2.5.2. 새 소스 파일 만들기

① **New Project 실행** : 메인창의 [File->New] 메뉴를 선택하면 Create New File이라는 작은 창이 생성된다. 그 때 그림 57과 같이 source를 선택하고 OK버튼을 클릭한다.

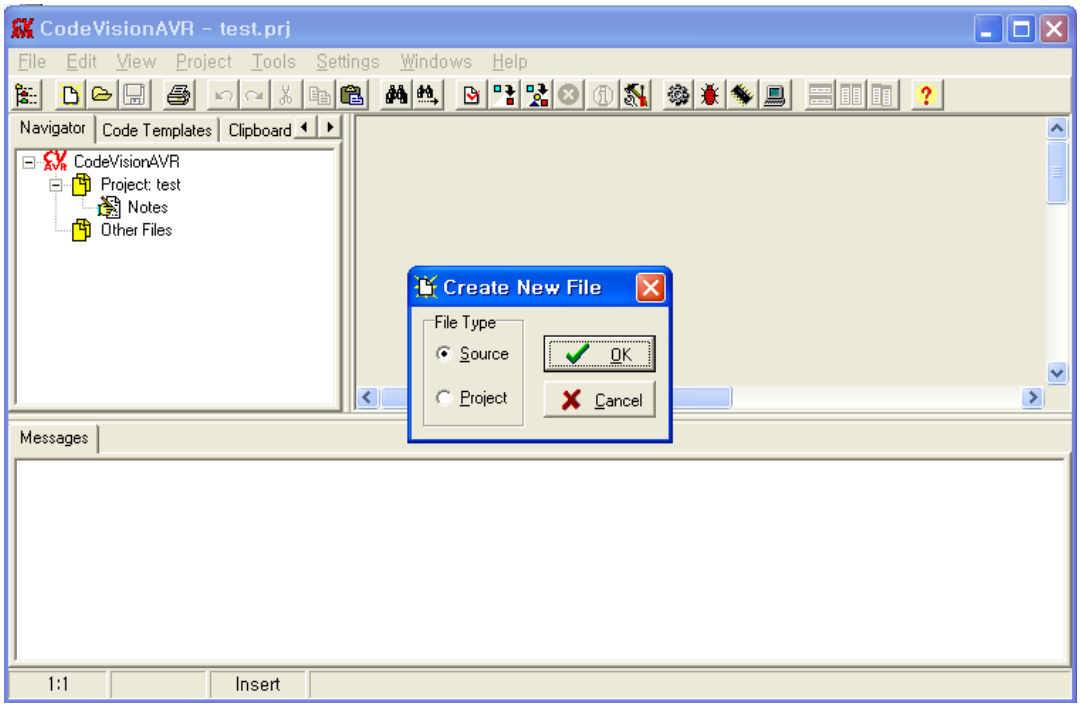


그림 57. [File->New] 메뉴의 실행과 새로운 파일생성

② **에디터 창 생성** : Create New File에서 source를 선택하고 OK버튼을 클릭하면 그림 58처럼 특정 디렉토리명₩untitled,c라는 이름을 갖는 C코드를 작성할 수 있는 에디터 창이 뜬다. 그 창에 파일을 작성한다. 작성시 [Navigator], [Code Templates], [Clipboard]메뉴를 활용하면 작성을 쉽게 할 수 있다. 자세한 사항은 [Help]메뉴를 통해 얻을 수 있다.

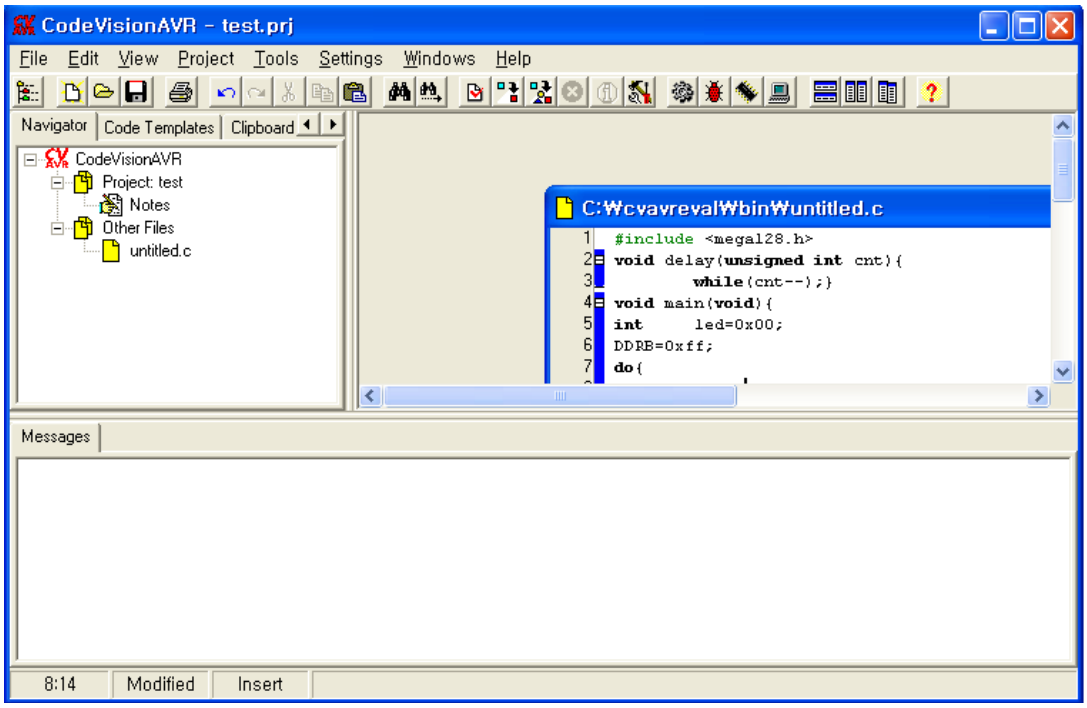


그림 58. CodevisionAVR에서 새로운 파일의 작성

③ **Save As 실행** : 소스코드의 작성을 완료하고 메인창의 [File->Save as...] 메뉴를 선택하면 특정 디렉토리명₩untitled,c As라는 이름의 대화상자가 생성되는데 대화상자내의 [저장위치(I)]메뉴에서 적당한 위치를 선택하고 자신이 원하는 파일 이름(예: fnd)을 [파일이름(N)]메뉴의 입력창에 기입한 후에 [저장(S)] 버튼을 클릭한다. 그러면 본 예에서는 at128에 fnd.c가 저장된다.

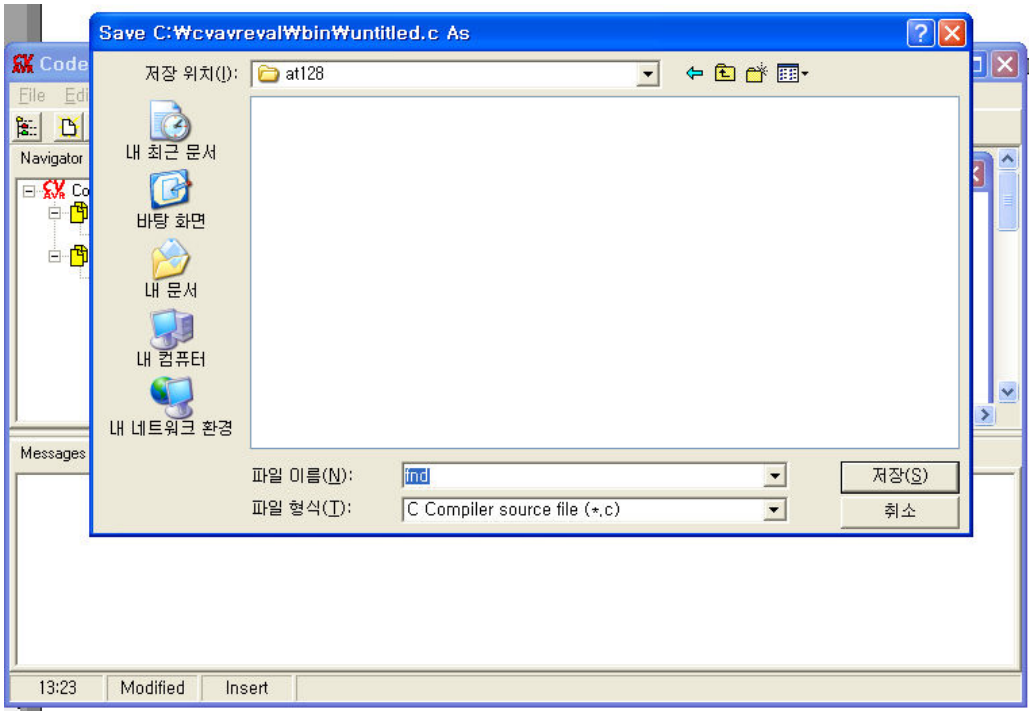


그림 59. [File->Save As] 메뉴를 통한 파일 저장

2.5.3. 프로젝트에 소스파일 등록하고 실행파일 만들기

① **Configure 실행** : 소스파일은 프로젝트에 등록이 되어야 컴파일하고 디버깅이 가능하다. 그림처럼 CodevisionAVR을 실행해 새 프로젝트를 생성하거나 기존의 프로젝트를 열어 놓은 상황에서 메인창의 [Project->Configure]를 실행하면 그림60과 같은 Configure Project 대화창이 새롭게 생성된다.

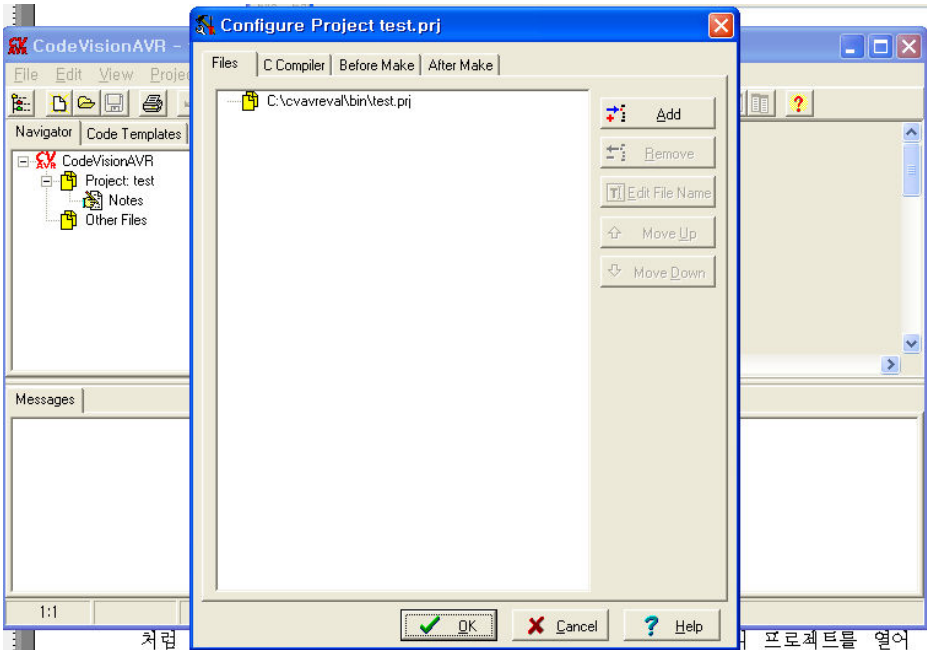


그림 60. [Project->Configure]메뉴의 실행

② **Add 실행 :** 그림60과 같은 Configure Project 대화창에서 [Add]메뉴를 선택하면 그림 61과 같은 Add File To Project라는 이름의 대화창이 생성되는 데 대화상자내의 [찾는 위치(I)]메뉴에서 적당한 위치(예:at128)를 선택하고 파일 이름(예: fnd)을 [파일이름(N)]메뉴의 입력창에 기입한 후에 [열기(O)] 버튼을 클릭한다.

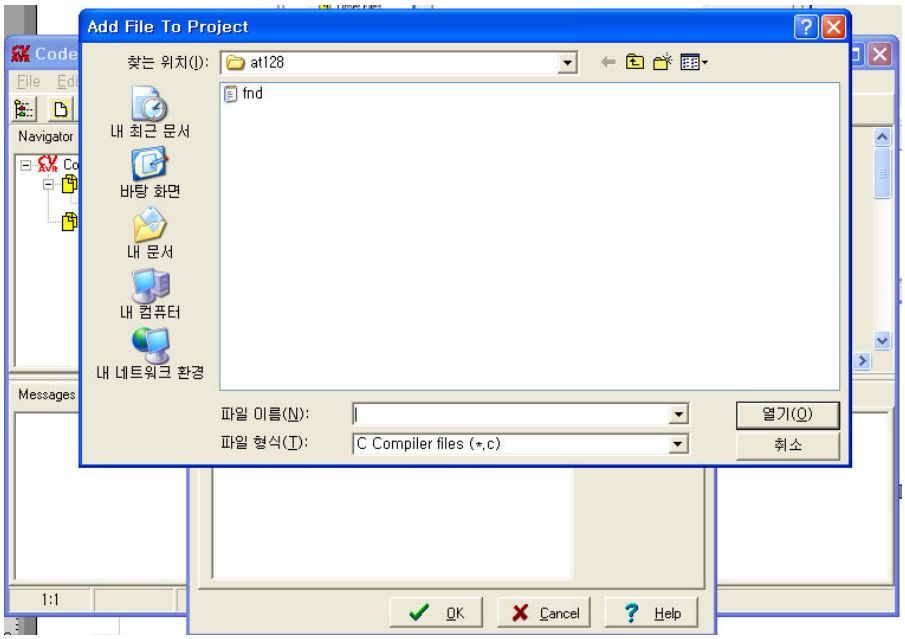


그림 61. Add메뉴를 통한 소스파일 등록

③ **Add 실행 결과 :** Add File To Project라는 이름의 대화창에서 파일을 등록하면 그림 62와 같이 프로젝트에 등록된 결과를 Configure Project 대화창을 통해 확인할 수 있다. 다른 파일을 더 등록하려면 마찬가지 과정을 통해 실행한다. 특정 파일을 프로젝트에서 제거하려면 Configure Project 대화창에서 파일을 선택하고 [Remove]메뉴를 통해 제거하면 된다. 본 예에서는 at128디렉토리의 fnd,c가 test.prj프로젝트에 등록된 것을 그림 62에서 확인할 수 있다. 등록을 다했으면 대화창에서 [OK]메뉴를 누르고 나간다.

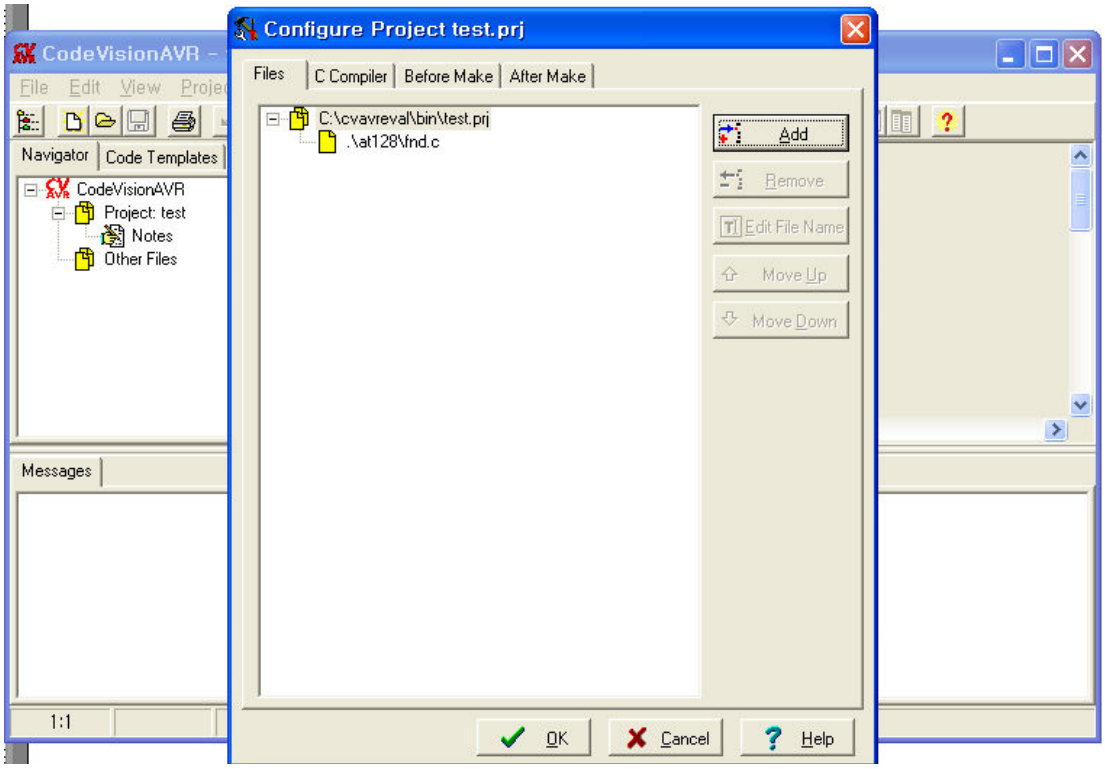


그림 62. Add메뉴를 통한 소스파일 등록 결과

④ Make 실행 : 소스파일을 프로젝트에 등록을 완료한 후에 그림 63처럼 메인창의 [Project->Make]메뉴를 통해 실행파일을 생성한다.

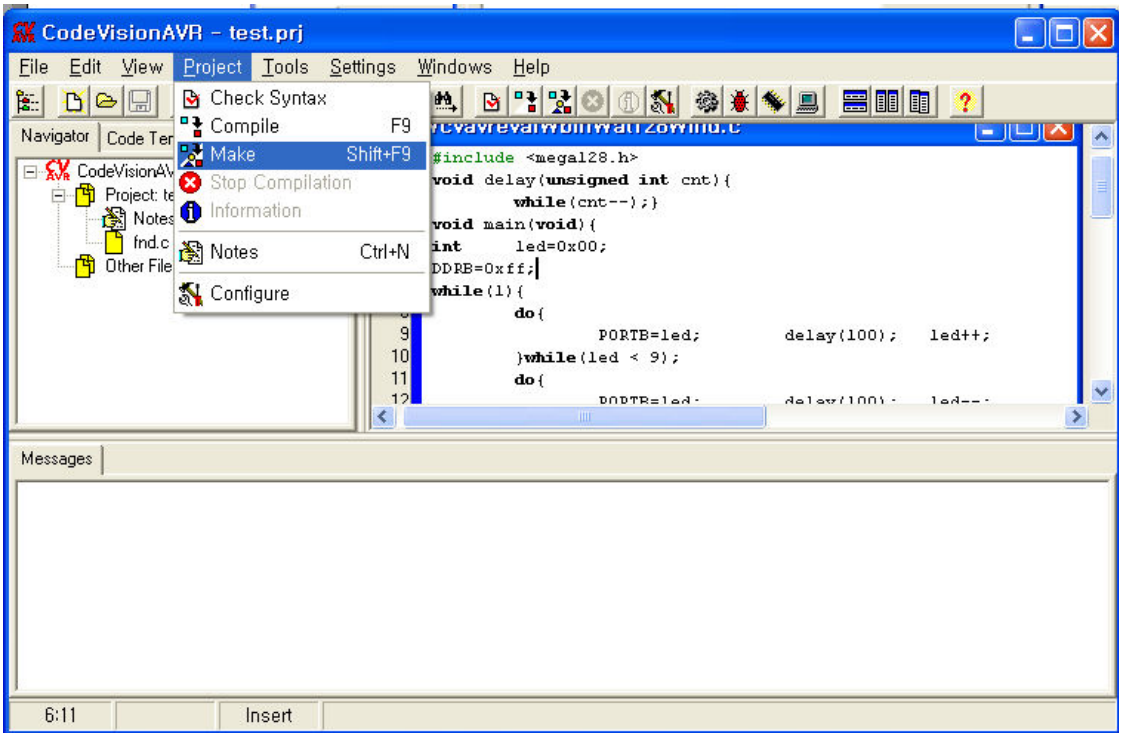


그림 63. CodeVisionAVR의 Make 실행

⑤ 에러없이 종료된 결과 : 그림 64는 에러없이 실행파일이 만들어진 경우를 보여준다. Information이라는 이름의 창이 생겨 Make 결과를 설명한다.

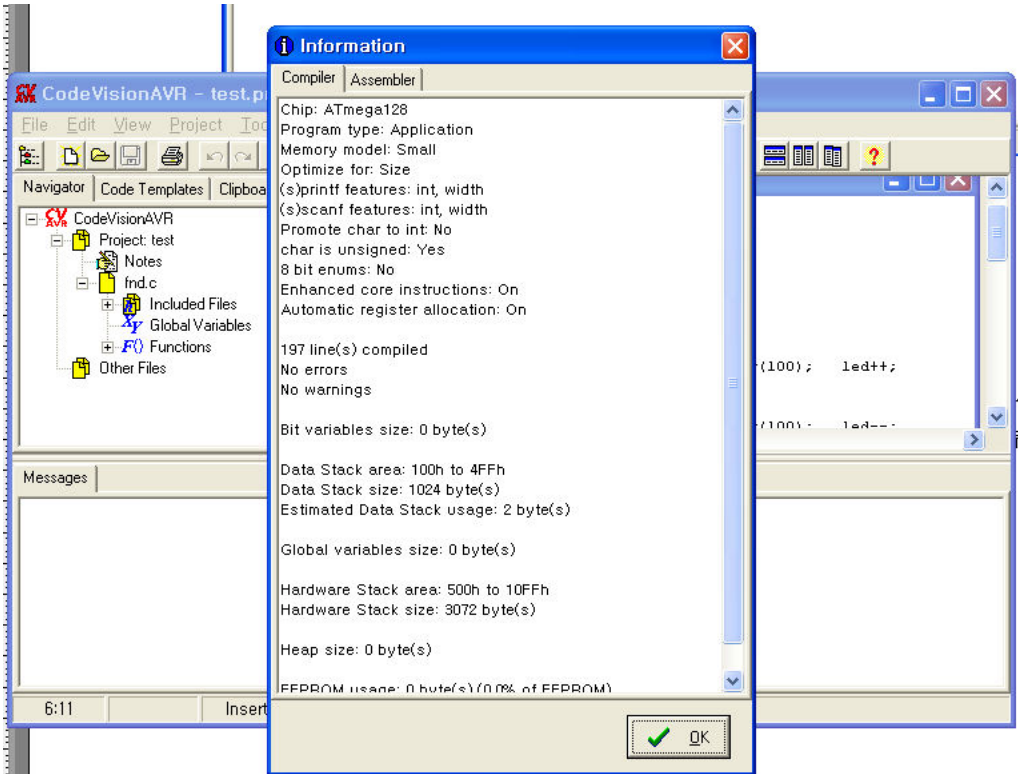


그림 64. CodevisionAVR에서 Make의 성공적인 결과

⑥ 에러가 있을 때 결과 : 그림 65는 실행파일을 만드는 과정에서 문법적인 에러가 발생한 경우를 보여준다. Information창에 결과가 나타나 있고 messages창에 에러가 발생한 행번호와 에러메시지가 나타나 있음을 알 수 있다. Information창의 [OK]메뉴를 클릭하면 창이 없어지고 그림 66과 같이 된다. 에러메시지 부분을 더블클릭하면 에러가 발생한 위치로 커서가 자동적으로 이동하여 손쉽게 에러를 수정할 수 있게 한다.

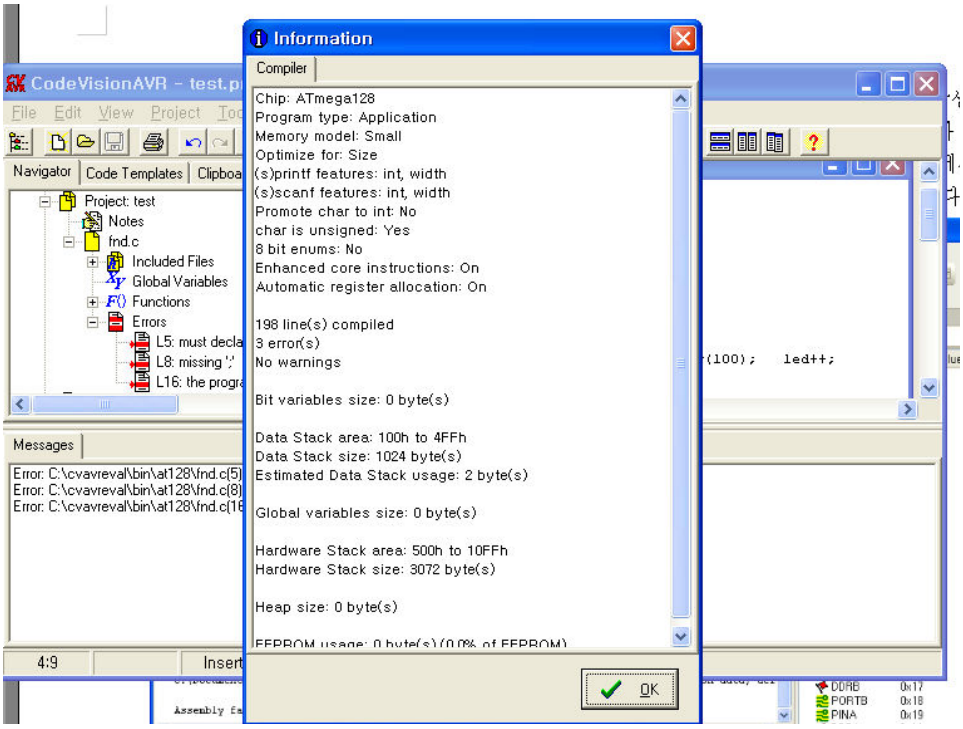


그림 65. CodevisionAVR에서 파일에 에러가 있을 때 Make 결과

⑦ 에러 수정 : 그림 65에서 Information창의 [OK]메뉴를 클릭하면 창이 없어지고 그림 66과 같이 된다. 에러메시지 부분을 더블클릭하면 에러가 발생한 위치로 커서가 자동적으로 이동하여 손쉽게 에러를 수정할 수 있게 한다.

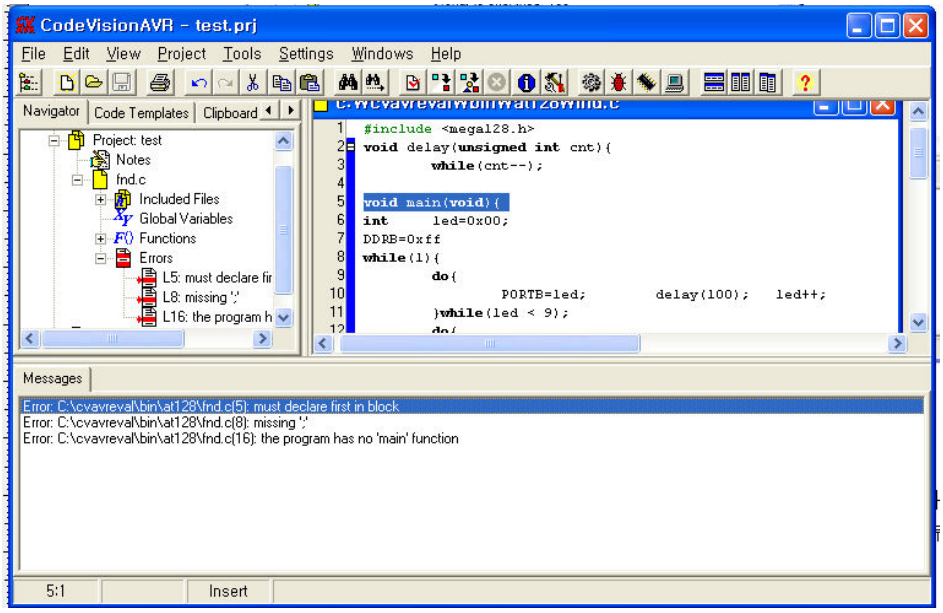


그림 66. CodeVisionAVR를 이용한 문법에러의 수정

2.5.4. 디버깅

① Debugger 설정 : 문법적인 에러가 없이 성공적으로 Make가 되어 실행파일이 만들어진 경우 AVR Studio4가 설치되어 있으면 이를 이용해 디버깅과 간단한 시뮬레이션을 할 수 있다. 그림 68과 같이 CodeVisionAVR 메인메뉴 [Settings->Debuger]를 선택하면 Debugger Settings라는 이름의 대화창이 생성되는데 AVR Studio4가 설치된 디렉토리와 디버거의 이름을 입력하고 [OK]메뉴를 클릭한다.

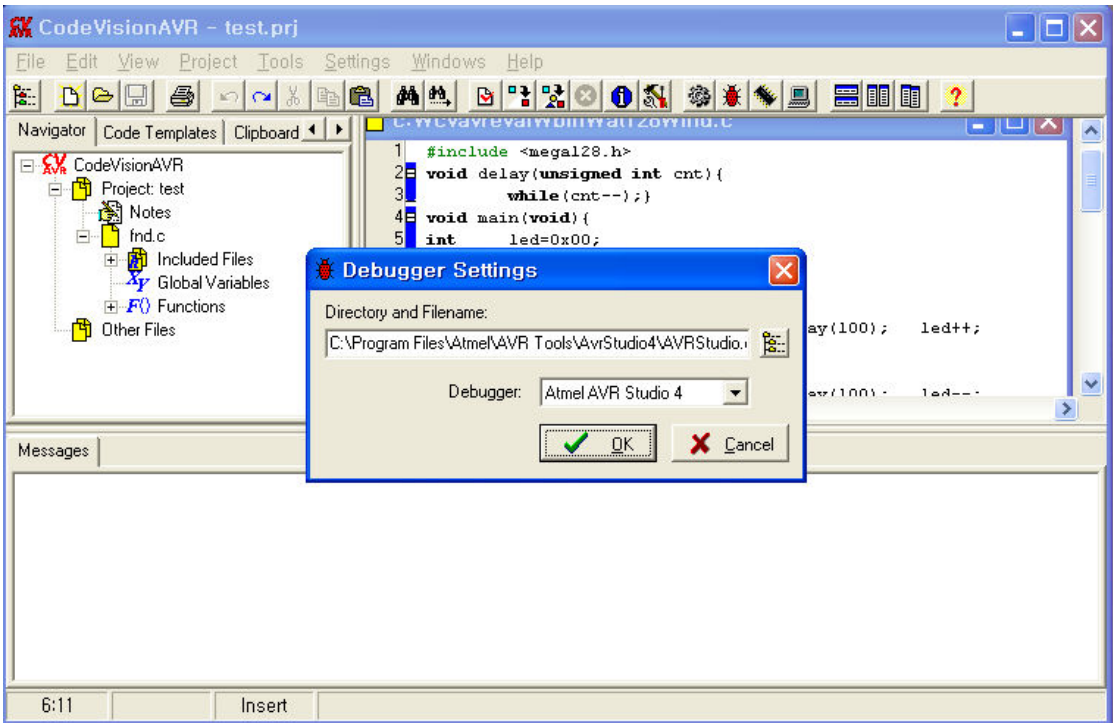


그림 67. Debugger의 선택

② Debug 선택 : 문법적인 에러가 없어 실행파일이 만들어진 경우 AVR Studio4가 설치되어 있으면 그림 67과 같이 [Tools->Debugger] 메뉴를 선택하면 디버깅과 간단한 시뮬레이션을 할 수 있다.

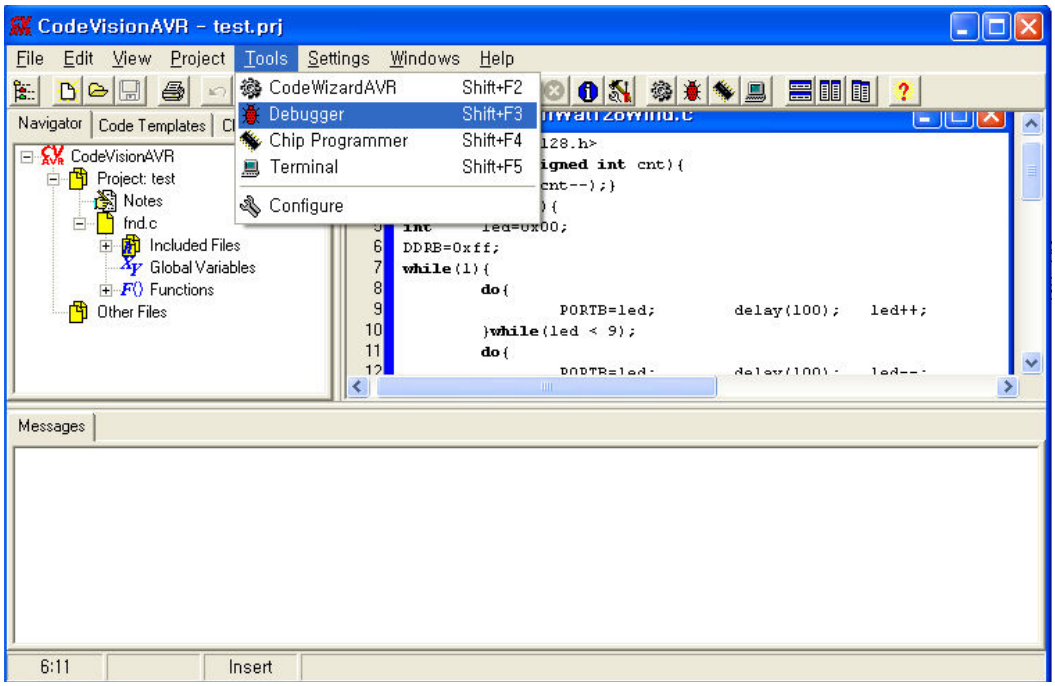


그림 68. [Tools->Debugger] 메뉴

② 프로젝트 선택 : [Tools->Debugger] 메뉴를 선택하면 그림 69처럼 AVR Studio4가 실행되며 프로젝트를 선택하기 위한 Welcome to AVR Studio4라는 이름의 대화창이 생성된다.

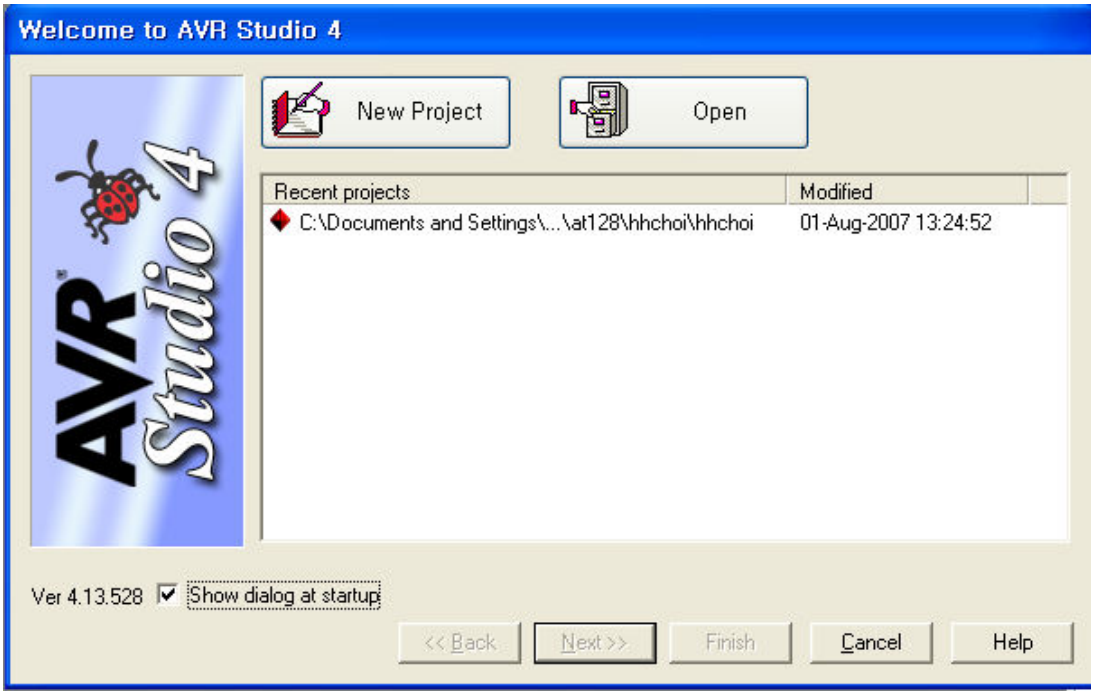


그림 69. AVR Studio4의 프로젝트 선택창

③ cof 파일 선택 : Welcome to AVR Studio4라는 이름의 창의 메뉴 [Open]을 클릭하면 그림 70과 같은 Open Project File or Object File이라는 이름의 대화창이 생성된다. [파일 이름(N)] 메뉴에 디버깅하려하는 파일을 Make하여 생성된 파일들 중에서 extension이 cof로 끝나는 파일을 선택한다. 본 예에서는 fnd.c를 make 하였으므로 fnd.cof를 선택한다. 그리고 [열기(O)] 메뉴를 더블클릭한다.

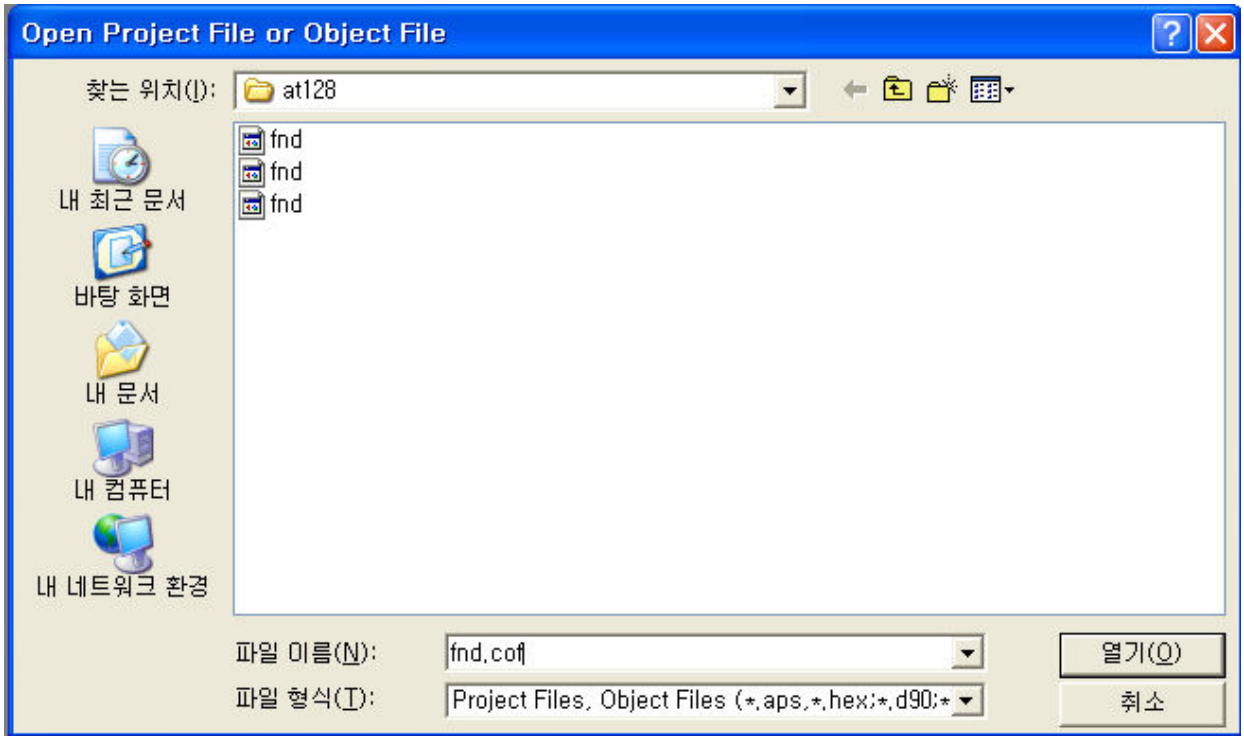


그림 70. cof파일의 선택 대화창

④ 프로젝트 생성 : cof 파일을 선택하면 그림 71과 같은 Save AVR Studio Project File이라는 이름의 대화창이 생성된다. 적당한 이름을 [파일이름(N)]메뉴의 입력창에 쓰고 [저장(S)]메뉴를 클릭한다.

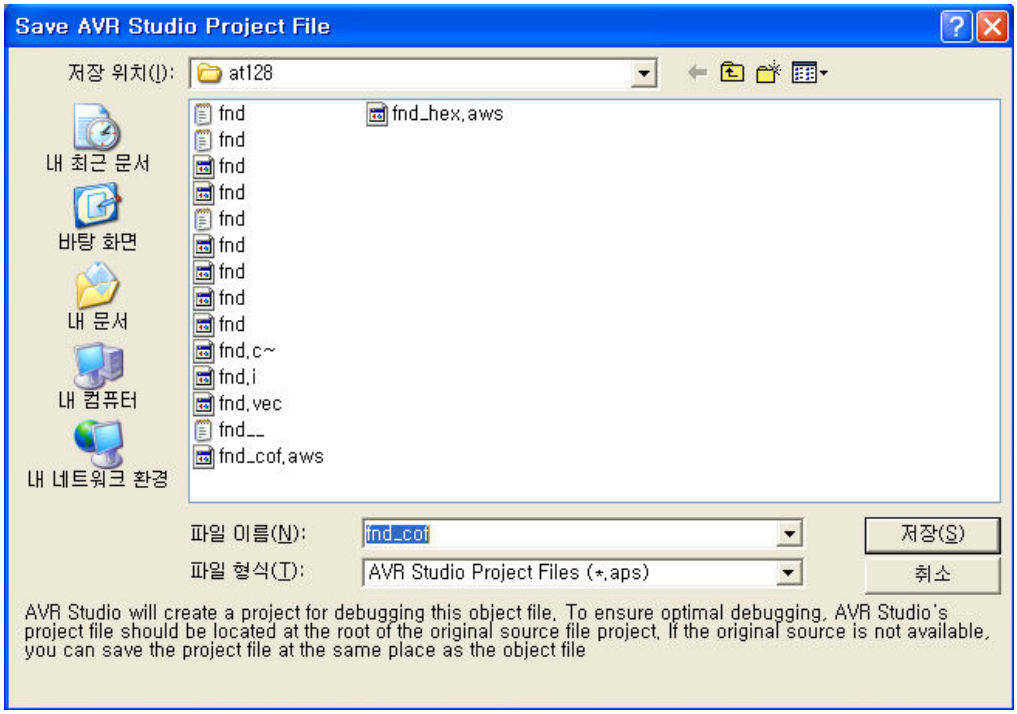


그림 71. 디버깅을 위한 프로젝트 생성

⑤ Debug 플랫폼과 디바이스 설정과 디버깅하기 : 적절한 프로젝트 이름을 설정하고 나면 그림 40과 같이 디버그 플랫폼과 디바이스를 설정하는 창이 뜬다. Debug platform은 AVR Simulator를 선택하고 Device는 ATmega128을 선택하고 마지막으로 [Finish] 메뉴를 누른다. 그러면 그림72처럼 디버그세션이 시작되고 /O View창이 I/O레지스터의 값들을 비트별로 그래픽하게 확인할 수 있도록 변하고 에디터 창에 노란 화살표가 나타난다. 이 이후의 과정은 2.3.2절을 참조하면 된다.

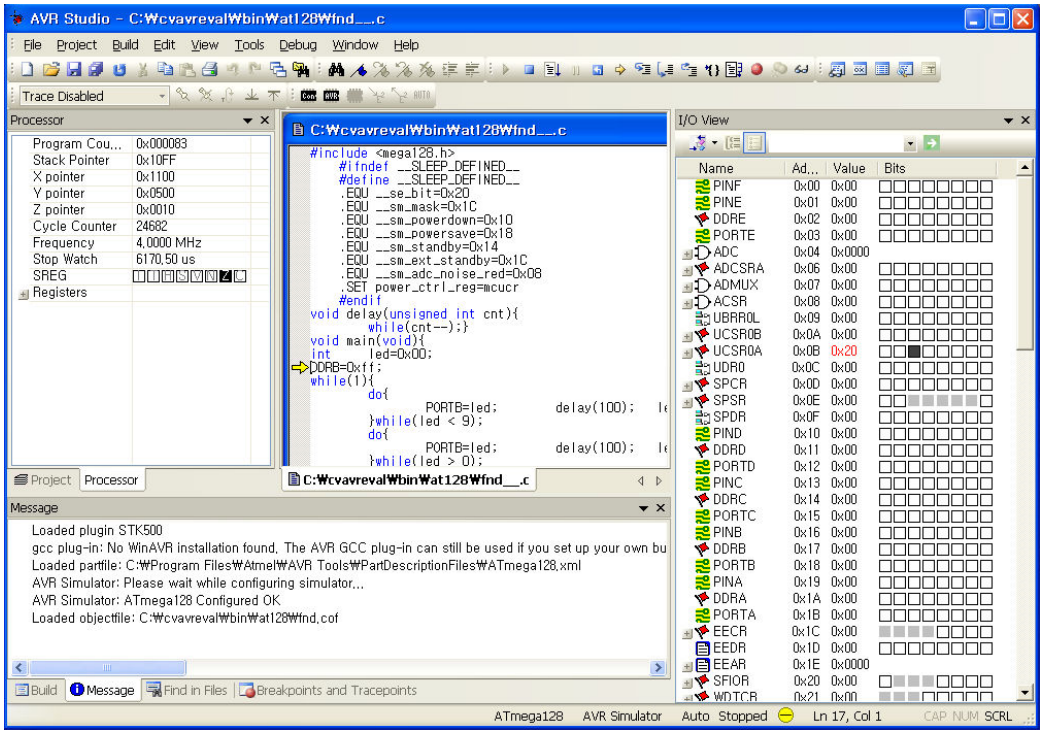


그림 72. CodeVisionAVR에서 AVR Studio4를 이용한 디버깅화면

2.5.5. 플래쉬롬 ISP 프로그래밍하기

① **Programmer 설정** : 프록그램의 시뮬레이션까지 끝내어 완벽해진 경우 ATmega128의 프로그램메모리에 ISP를 통해 프로그램을 다운로드 해야한다. CodeVisonAVR의 메인메뉴 [Settings->Programmer]를 선택하면 그림 73과 같은 대화창이 생성된다. 그림 51의 병렬 ISP 인터페이스 회로를 사용하는 경우 [AVR Chip Programmer Type]는 그림 73처럼 STK200+/300으로 선택하고 [Print Port]는 ATmega128과 연결된 병렬포트를 선택한다.

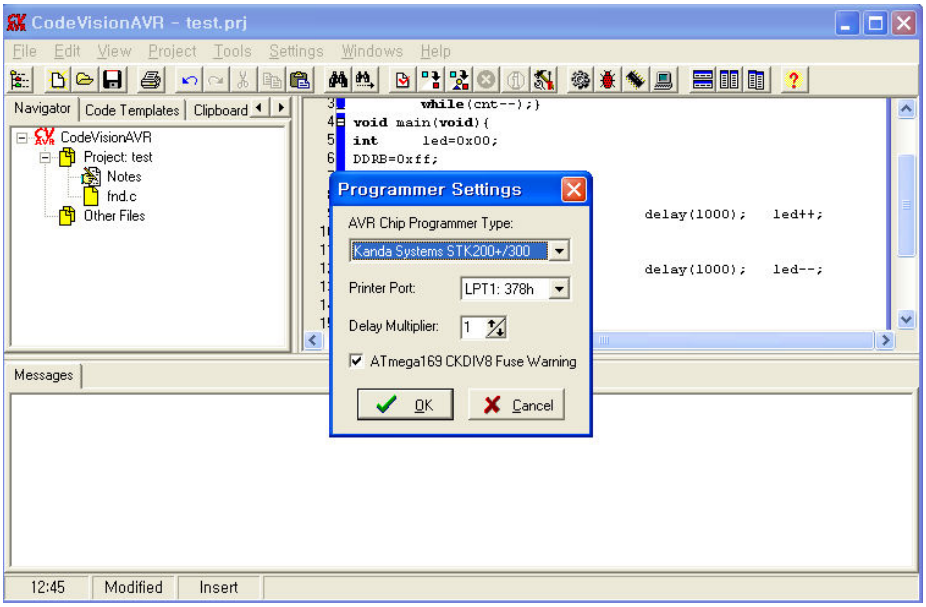


그림 73. AVR Chip 프로그래머 설정 화면

② **Programmer 실행** : 프로그래머의 설정이 이루어 졌으면 CodeVisonAVR의 메인메뉴 [Tools-> Chip Programmer]를 선택한다. 그러면 그림 74와 같은 CodeVisionAVR Chip Programmer 이라는 이름을 갖는 대화창이 생성된다.

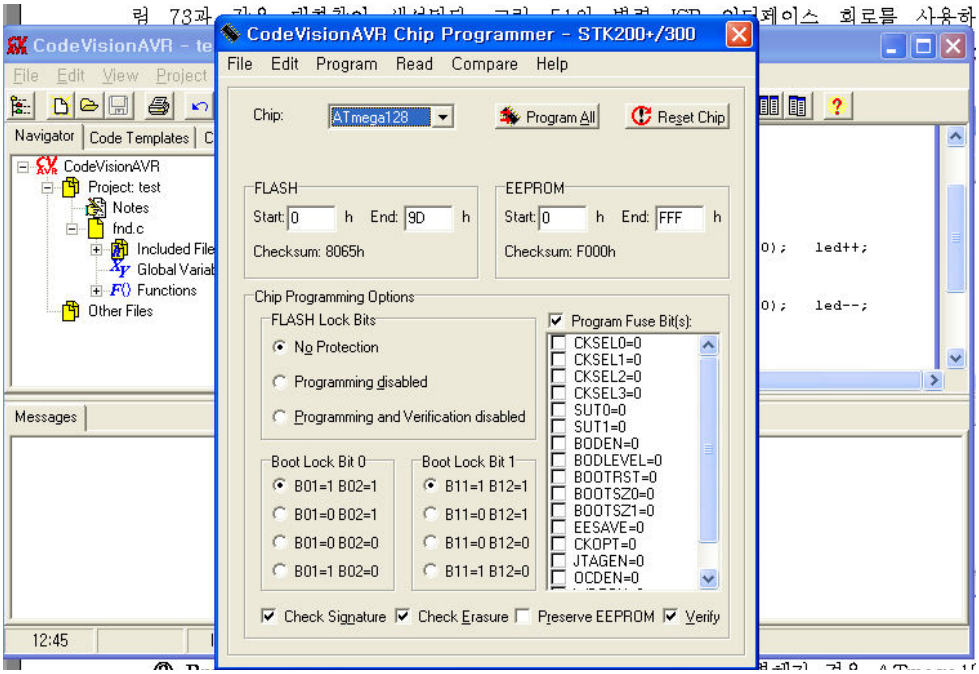


그림 74. CodeVisionAVR의 칩 프로그래머 대화창

③ rom 파일 선택 : CodeVisionAVR Chip Programmer 이라는 이름을 갖는 대화창이 생성된다. 대화창의 메뉴 [File->Load FLASH]를 선택하면 그림 75와 같은 Load File to FLASH Buffer이라는 이름의 대화창이 생성된다. [찾는 위치(I)]메뉴에서 파일이 존재하는 위치(예:at128)를 선택하고 make를 통해 만들어진 rom이나 hex, bin 파일 이름(예: fnd.rom)을 [파일이름(N)]메뉴의 입력창에 기입하거나 선택한 후에 [열기(O)] 버튼을 클릭한다.

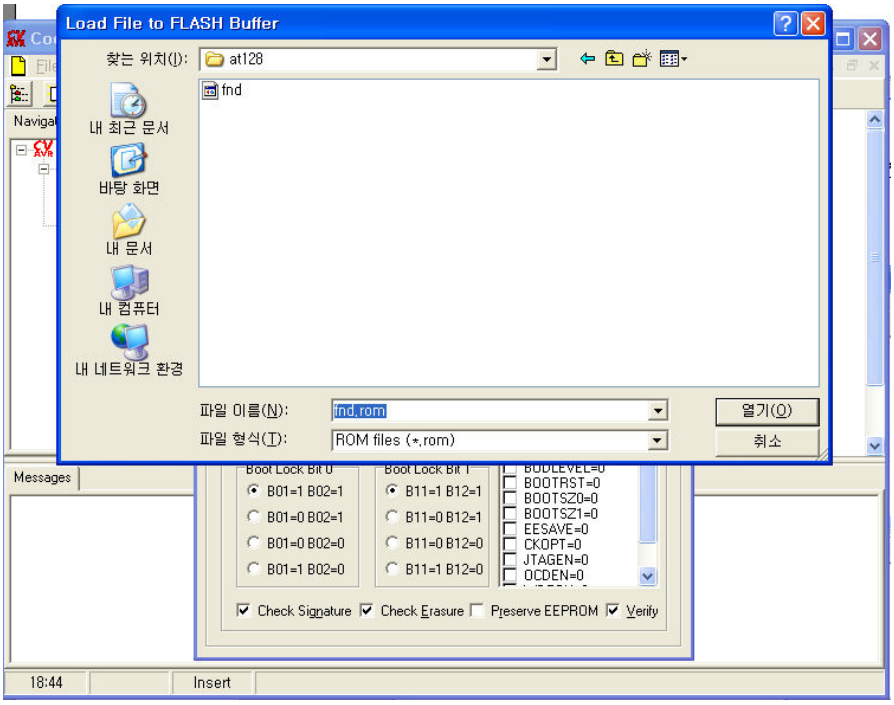


그림 75. 다운로드할 rom이나 hex 혹은 bin 파일의 선택

④ erase chip 실행 : 그림 51의 병렬 ISP 인터페이스 회로와 ATmega128시스템이 연결되어 있고 전원이 들어간 상태에서 다운로드할 hex파일을 선택한 다음 CodeVisionAVR Chip Programmer 대화창의 메뉴 [Program->Erase Chip]를 선택하면 그림 76과 같은 FLASH Erasure checking창이 생성되고 새로 써 넣기 전에 Flash 롬의 내용을 지운다.

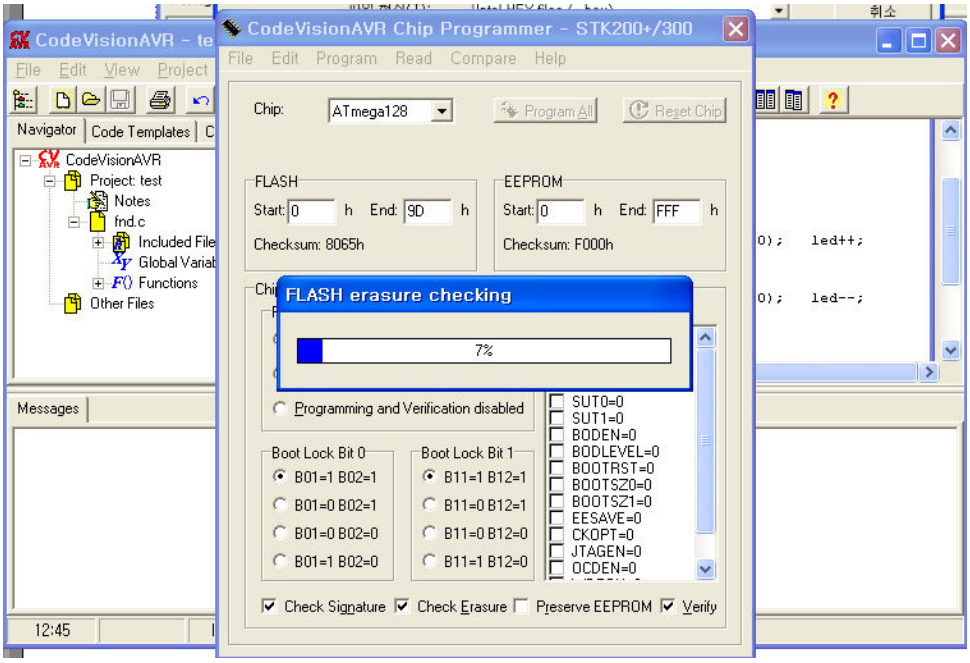


그림 76. 플래시롬을 지우는 과정을 보여주는 창

⑤ **프로그램 다운로드 :** 위의 과정을 거친 다음 CodeVisionAVR Chip Programmer 대화창의 메뉴 [Program->Frash]를 선택하면 그림 77과 같은 Verifying the FLASH programming창이 생성되고 플래쉬롬에 프로그램이 다운로드 된다.

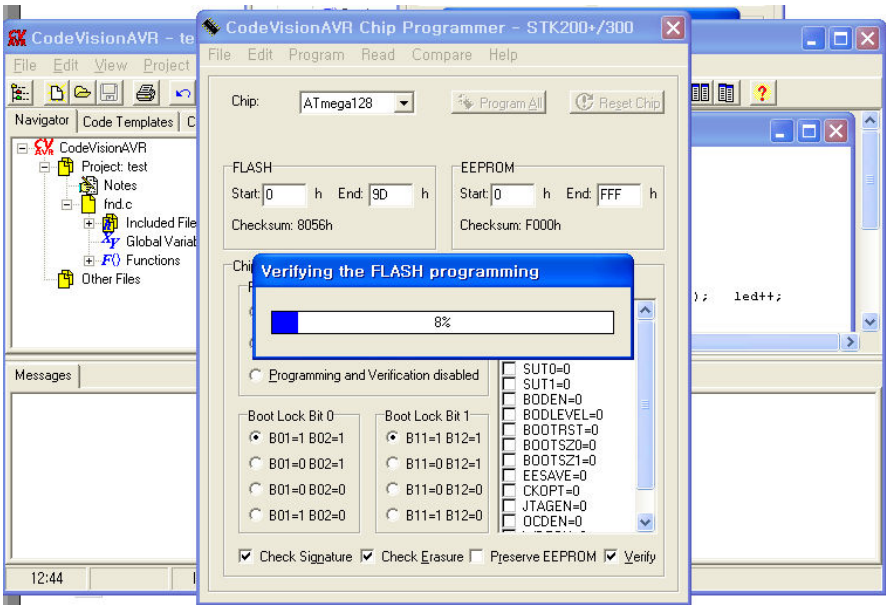


그림 77. 프로그램의 다운로드 과정을 보여주는 창

2.5.6. EEPROM ISP 프로그래밍하기

① **Programmer 실행 :** CodeVisonAVR의 메인메뉴 [Tools-> Chip Programmer]를 선택한다. 그러면 그림 74와 같은 CodeVisionAVR Chip Programmer 이라는 이름을 갖는 대화창이 생성된다.

② **EEPROM 파일 로드 :** CodeVisionAVR Chip Programmer 대화창의 메뉴 [File->Load EEPROM]으로 화일을 선택해서 로드하거나 [Edit->EEPROM]을 선택해 Edit EEPROM Buffer이라는 이름을 갖는 대화창을 생성시키고 직접 편집을 한다.

③ **다운로드 :** 위의 과정을 거친 다음 Chip Programmer 대화창의 메뉴 [Program->EEPROM]를 선택하면 그림 76, 77과 비슷한 EEPROM programming창과 Verifying the EEPROM programming 창이 차례대로 생성되고 EEPROM이 프로그램된다.

2.6. C 프로그래밍에서 유의할 점

2.6.1. C 프로그램의 구성 요소

C 프로그램은 지시어, 심볼, 문장, 블록의 4가지 구성요소가 있다.

- 지시어 또는 키워드: C 프로그램 명령어들과 같이 C 컴파일러가 특별한 의미로 인식하는 예약된 문자들
- 심볼 : 지시어와 달리 사용자가 특별히 의미를 부여한 문자이다. 알파벳 대문자와 소문자과 숫자와 _가 가능하며 첫문자는 반드시 숫자가 아니어야 하며 대 소문자가 구별된다.
- 문장 : 프로그램 실행 단위로 반드시 ;(세미콜론)으로 끝낸다.
- 블록 : {로 시작해 }로 끝 맺으며 여러개의 문장을 묶은 것이다.

2.6.2. C 프로그램의 일반적 형식

C는 매우 자유로운 형식의 언어인데 C 프로그램 파일은 소스파일과 헤더파일로 구분된다.

- 헤더파일 : 여러 소스파일에 공통으로 들어가는 함수의 프로토타입의 선언문, 새로운 데이터형의 정의 또는 다른 헤더파일을 include시킨다. 전역(global) 변수의 선언문이나 함수의 정의는 헤더파일에 두지 않는 것이 좋다.
- 소스파일 : 일반적으로 전처리문, 선언문, 주석, main 함수 및 사용자 정의 함수 등으로 구성된다.

2.6.2.1. 전처리

#문자로 시작하며 다음과 같은 지시어들이 존재한다.

- #include : 외부 파일로부터 소스를 읽어 들인다. include 디렉토리에 존재하는 외부파일은 < >로 묶어주어 include시키고 현재의 디렉토리에 존재하는 외부파일은 " "로 묶어 준다.
- #define : 매크로나 상수를 정의한다. 매크로의 정의가 길어 한줄에 쓰지 못할 때는 ₩를 사용해 줄 바꾼다.
- #undef : 매크로나 상수 정의를 취소한다.
- #ifdef : 매크로나 상수가 정의되어 있다면 해당부분만 컴파일한다.
- #ifndef : 매크로나 상수가 정의되어 있지 않다면 해당부분만 컴파일한다.
- #if : 어떤 조건을 만족하면 해당부분만 컴파일한다.
- #elif : ifdef, ifndef, if와 같은 조건문에서 또 다른 조건에 의한 분기조건을 정의하여 그 조건이 만족하면 해당부분만 컴파일한다.
- #else : ifdef, ifndef, if와 같은 조건문에서 이전 조건이 만족되지 않으면 해당부분만 컴파일한다.
- #endif : ifdef, ifndef, if, elif, else 조건 블록의 마지막을 표시한다.

2.6.2.2. 선언문

함수와 변수 등 심볼을 선언하는 부분이다. 변수는 문자형인지, 정수형인지, 실수형인지 데이터형을 선언하고 함수의 경우에는 함수의 이름, 함수를 처리하고 난 후 리턴할 데이터형, 전달받을 인수의 데이터형 등 함수의 프로토타입을 선언한다. 결과를 리턴하지 않는 경우 void를 쓴다.

2.6.2.3. 주석

- /* 와 */ : /*로 시작해서 */로 끝 맺으면 그 사이의 글자는 컴파일러가 무시한다. /*과 */ 사이에 줄을 바꿔도 그 사이의 글자는 주석으로 간주된다.
- // : //를 사용하면 같은 줄에서 //뒤의 글자는 컴파일러가 무시한다.

2.6.2.4. 함수와 main 함수 부분

반드시 하나의 블럭으로 되어 있어 어떤 내용을 처리하고 그 결과를 리턴하는 등 실제적인 프로그램을 기술하는 부분이다. 다음의 형태를 띤다.

```
리턴데이터형   함수명([데이터형 인수1,데이터형 인수2, ...])
{ 프로그램 내용 }
```

리턴데이터형은 내용을 처리하고 난 후 리턴할 결과 데이터 형태가 정수형인지, 문자형인지, 실수형인지 등 무엇인지를 기술하는 부분이다. 결과를 리턴하지 않는 경우 void를 쓴다. 함수명은 함수의 이름을 나타내는 심볼이며 프로그램을 처리할 때 전달받을 값은 인수들을 통해 하는 데 함수를 정의할 때 인수들의 데이터형과 함께 ( ) 안에 써 넣는다. 전달받을 값이 없으면 인수를 안쓰거나 void라고 쓴다. main 함수는 함수명이 main으로 반드시 C 프로그램에서 만들어야 하는 함수이다.

2.6.3. 기본 데이터형

C언어에서 데이터는 문자와 숫자로 분류할 수 있고 또한 상수와 변수로 구분할 수 있다. 변수는 사용범위에 따라 광역(또는 전역, global)과 지역(local) 변수가 있다. 그리고 이들 데이터는 기본적으로 문자형, 정수형, 실수형

으로 구분할 수 있다. 표5에 기본 데이터형 지시어의 설명이 주어졌다.

- 문자 : 알파벳과 특수기호들로 컴퓨터에 표현하려면 특정 코드를 사용해야 하는데 C언어에서는 아스키코드를 사용한다. 문자 한개를 표현할 때는 'a', 'b', '@' 식으로 ' '기호를 사용하고 2개 이상의 문자열의 경우에는 " "기호를 사용한다.
- 숫자 : 정수와 소수로 나뉠 수 있다. 정수는 10진수, 16진수, 8진수 등을 사용하는데 16진수를 가장 많이 쓰며 16진수는 숫자 앞에 0x를 붙여 주고 8진수는 0만 붙여주고 10진수는 0이나 0x를 붙이지 않고 표기한다. 소수의 경우에는 0.0123 과 같은 식으로 소수를 쓰는 방법과 1.23E-2 식으로 지수를 사용하는 방법이 있다. 음수의 경우에는 -를 숫자 앞에 붙이고 양수의 경우 +를 생략해도 된다.
- 상수 : 프로그램내에서 값의 변화가 없는 데이터를 지칭한다.
- 변수 : 값의 변화가 있는 데이터를 지칭하며 값을 마음대로 바꿀 수 있는 기억장소를 의미한다. 변수는 전역(global), 지역(local), 정적(static), 외부(extern)변수, 레지스터(register)로 나뉠 수 있다(함수 부분 설명 참조).

표 8. 기본 데이터형

데이터형	지시어	크기	범위
문자형	char	1 바이트	-128 ~ 127
	unsigned char	1 바이트	0 ~ 255
정수형	int	2 바이트	-32768 ~ 32767
	unsigned int	2 바이트	0 ~ 65535
	long	4 바이트	-2147483648 ~ 214783647
	unsigned long	4 바이트	0 ~ 42967295
실수형	float	8 바이트	± 1.2E-38 ~ ± 3.4E38
	double	8 바이트	± 1.2E-38 ~ ± 3.4E38

2.6.3.1. 문자형

문자형 데이터는 1바이트 크기로 하나의 문자를 의미하며 작은 숫자나 아스키 문자를 표현하기 위해 사용되는데 지시어로 부호가 없는 형태에 char와 부호가 있는 형태에 unsigned char를 사용한다. char형은 -128 ~127 사이의 값을 갖고 unsigned char형은 0 ~ 255 사이의 값을 갖는다.

2.6.3.2. 정수형

정수를 표현하기 위해 사용되는 지시어로 int, unsigned int, long, unsigned long가 있다. int는 2바이트 크기로 -32768 ~ 32767 사이의 값을 갖고 unsigned int는 2바이트 크기로 0 ~ 65535 사이의 값을 갖는다. long은 바이트 크기로 -2147483648 ~ 214783647 사이의 값을 갖고 unsigned long는 4바이트 크기로 0 ~ 4294967295 사이의 값을 갖는다.

2.6.3.3. 실수형

실수를 표현하기 위해 사용되는 지시어로 float, double이 있는데, float는 8바이트 크기로 ± 1.2E-38 ~ ± 3.4E38사이의 값을 갖고, double은 8바이트 크기로 ± 1.2E-38 ~ ± 3.4E38사이의 값을 갖는다.

2.6.3.4. 변수나 상수 이름 짓는 법

- 알파벳 대문자와 소문자과 숫자와 _가 가능하며 대 소문자가 구별된다.
- 변수이름의 첫 글자는 숫자가 아니어야 한다.
- _가 아닌 특수문자는 사용이 안된다.
- 지시어는 사용할 수 없다.
- 되도록이면 헝가리언 표기법에 따라 특별한 접두어를 사용해서 변수이름을 짓는다. 포인터를 정의할 때는 접두어 p, 문자열은 str, 플래그를 정의할 때는 f, 카운터나 int형을 정의할 때는 접두어 c, long형은 l, unsigned

int형은 u를 접두어로 사용하면 알아보기 쉽다.

2.6.3.5. 상수 정의

- #define을 사용하여 다음과 같은 식으로 정의한다. 여기에서 문자의 끝에 ;가 들어가지 않음에 유의해야 한다.

  #define　　상수이름　상수값

- const 지시어를 사용하여 다음과 같은 식으로 정의한다.

  const　　데이터형　　상수이름 = 상수값 ;

2.6.3.6. 변수 선언과 초기화

변수는 함수내의 첫부분에서 선언하거나 함수 밖에서 프로그램 첫부분에서 선언할 수 있다. 함수내의 첫부분에서 선언하면 지역변수가 되어 함수내에서만 사용할 수 있으며 함수 밖에서 선언하면 전역변수가 되어 정의된 소스 프로그램내의 모든 함수에서 사용 가능하다. 다음과 같은 식으로 선언하고 초기화 한다. 변수명1만 선언하는 경우에는 [ ]안에 주어진 값과 [ ]기호는 생략 가능하다.

데이터형　　변수명1[=초기값1, 변수명2=초기값2, ...];

2.6.4. 확장 데이터형

CodeVisionAVR 컴파일러는 2.6.3절에 주어진 ANSI C의 데이터형에 더해 bit, eeprom, flash, sfrb, sfrw등의 데이터형을 지원해주어 AVR 하드웨어 특성을 활용할 수 있도록 하였다.

2.6.4.1. bit 데이터형

범용 I/O레지스터와 R2~R14레지스터에 1비트 단위의 데이터를 위치시킨다. bit 데이터형은 일반적으로 광역변수로 선언된다. R15레지스터에 1비트 단위의 데이터를 위치시키는 지역 변수의 선언도 8개까지 허용된다.

2.6.4.2. eeprom 데이터형

EEPROM 메모리에 위치한 상수 데이터를 접근하는데 사용한다.

2.6.4.3. flash 데이터형

플래쉬 메모리에 위치한 상수 데이터를 접근하는데 사용한다.

2.6.4.4. sfrb, sfrw 데이터형

I/O레지스터를 바이트 혹은 워드 단위로 접근하는데 사용한다. Keil C컴파일러의 sfr 데이터형과 유사하다. 신텍스는 다음과 같다.

sfrb 심볼1 = IO레지스터주소값;　　sfrw 심볼2 = IO레지스터주소값;

I/O레지스터들은 WINC 디렉토리 밑에 디바이스명칭.h(예:mega128.h)의 형태로 헤더파일에 정의되어 있으므로 사용자는 별도로 선언할 필요없이 헤더파일을 프로그램 선두에 include시켜서 사용하면 된다. 즉 예로 ATmega128용으로 프로그램을 작성하는 경우 #include <mega128.h>를 프로그램 선두에 놓으면 DDRA, PORTA, TCNT1, ICR1 등 그림7과 8에 주어진 I/O레지스터의 이름을 사용해서 아래의 예처럼 값을 직접 할당할 수 있다.

예 : DDRA=0xff;　PORTA = 0x01　// PORT A를 출력으로 설정하고 0x01값을 출력하라.

2.6.5. 연산자

2.6.5.1. 조건 연산자

아래와 같은 형식으로 사용하며 삼항 연산자로 보통 불린다. 조건식이 참이거나 0이 아니면 식1을 처리하고 거짓이거나 0이면 식2를 처리하라는 의미이다.

변수 = (조건) ? 식1 : 식2;

2.6.5.2. 데이터 형변환 연산자

변수가 선언될 때 설정한 데이터형과 다른 데이터형을 사용할 때 사용하는 연산자로 다음과 같이 (와)사이에 형변환하려는 데이터형의 이름을 넣어 주면 변수1은 데이터형으로 형이 변환되어 변수2에 대입된다.

변수2 = (데이터형) 변수1;

2.6.5.3. 콤마 연산자

두 개 이상의 여러 행에 걸쳐 사용하는 문장을 하나의 행에서 실행할 때 사용한다. 두 개 이상의 문장을 ,를 사용하여 구분하여 나열하면 왼쪽부터 차례대로 실행된다. 아래의 형태는 문장1을 실행하고 문장2를 실행하고 문장n을 실행한다.

문장1, 문장2, 문장n;

2.6.5.4. 산술 연산자

표 9. 산술연산자

연산자	사용예	설명
+	a+b	a와 b를 더하라
-	a-b	a에서 b를 빼라
-	-a	a의 부호를 반대로 바꿔라
*	a*b	a와 b를 곱하라
/	a/b	a를 b로 나누어라
%	a%b	a/b의 나머지를 구하라
++	c++	c를 포함한 연산을 수행하고 c= c+1
	++c	c=c+1을 하고 c를 포함하여 연산하라
--	c--	c를 포함한 연산을 수행하고 c= c-1
	--c	c=c-1을 하고 c를 포함하여 연산하라

2.6.5.5. 논리와 비교 연산자

조건문에 사용되는 연산자들로 비교 연산자는 크기를 비교할 때 사용되고 논리 연산자는 둘 이상의 비교 연산자를 묶을 때 사용한다.

표 10. 논리, 비교 연산자

연산자	사용예	설명
<	a<b	a가 b보다 작다
>	a>b	a가 b보다 크다
<=	a<=b	a가 b보다 작거나 같다.
>=	a>=b	a가 b보다 크거나 같다.
==	a==b	a와 b는 동치이다.
!=	a!=b	a와 b는 동치가 아니다.
!	!(a<b)	(a<b)의 논리부정
&&	a>b && b>c	a>b 와 b>c의 논리곱
||	a>b || b>c	a>b 와 b>c의 논리합

2.6.5.6. 비트 연산자

비트를 조작하는 데 사용하는 연산자로 데이터를 비트별로 처리한다.

표 11. 비트 연산자

연산자	사용예	설명
<<	a<<2	a값을 우측으로 2비트 이동
>>	a>>2	a값을 좌측으로 2비트 이동
~	~a	a의 비트반전값
&	a&b	a와 b의 대응 비트별로 계산한 논리곱
|	a|b	a와 b의 대응 비트별로 계산한 논리합
^	a^b	a와 b의 대응 비트별로 계산한 XOR

2.6.5.7. 대입 연산자

표 12. 대입 연산자

연산자	사용예	설명
=	c = a+b	a+b의 결과를 c에 대입하라
+=	c += 2	c = c+2와 등가
-=	c -= 3	c = c -3과 등가
*=	c *= 3	c = c*3 과 등가
/=	c /= 2	c = b/2와 등가
%=	c %= 3	c = c % 3과 등가
<<=	c <<= 3	c = c << 3 과 등가
>>=	c >>= 2	c = c >> 2와 등가
&=	c &= b	c = c & b과 등가
^=	c ^= b	c = c^b 과 등가
|=	c |= b	c = c|b 와 등가

2.6.6. 제어문

2.6.6.1. if 문

- if 문 : 아래와 같은 형식으로 괄호안의 조건식이 충족되거나 0이 아니면 그 뒤의 문장이나 블록을 수행한다.

```
if ( 조건 ) 문장 ;
```

- if ~ else 문 : 아래와 같은 형식으로 괄호안의 조건이 충족되거나 0이 아니면 문장1을 수행하고 괄호안의 조건이 충족되지 않으면 문장2를 수행한다. 문장1이나 문장2는 블록이 될 수 있다.

```
if ( 조건 ) 문장1;
else 문장2;
```

- if ~ else if ~ else 문 : 아래와 같은 형식으로 여러 가지의 조건식으로 제어가능하다. 우선 if 문의 조건1이 충족되는지 점검하고 충족되거나 0이 아니면 문장1을 수행하고 if 문의 조건1이 성립하지 않으면 else if 문의 조건2를 점검하고 충족되거나 0이 아니면 문장2를 수행하고 else if 문의 조건2가 성립하지 않고 그 다음에 else if문이 존재하면 앞서의 과정을 반복하고 else문이 나오면 else문 뒤의 문장n을 수행한다. 문장1이나 문장2, 문장 n은 블록이 될 수 있다.

```
if ( 조건1 ) 문장1;
else if ( 조건2 ) 문장2;
else 문장n;
```

2.6.6.2. switch-case 문

if문과 함께 switch-case문은 조건문으로 if문에서는 조건이 비교연산자의 형태가 되어 변수값이 어떤 범위에 존재하는지 여부를 점검하지만 switch-case문에서는 switch 문의 변수 값이 case문의 값과 정확히 일치하는지를 비교해 일치하면 case문의 뒤에 오는 문장을 실행한다. 아래의 예에서는 변수값을 상수1과 비교해서 같으면 문장1을 실행하고 break에 따라 switch-case문의 블록을 빠져 나온다(만약 break가 없으면 블록을 빠져 나오지 않고 그 다음 case문을 점검한다). 변수값이 상수1과 다르면 다음 case문으로 가서 상수2와 변수를 비교해서 같으면 문장2를 실행하고 break에 따라 switchc-case문의 블록을 빠져 나온다. 그 다음에 case문이 존재하면 앞서의 과정을 반복하고 default문이 나오면 default문 뒤의 문장n을 수행하고 break에 따라 switch-case문의 블록을 빠져 나온다. 문장1이나 문장2, 문장 n은 블록이 될 수 있다.

```
switch ( 변수 ) {
    case 상수1 :    문장1;
                    break;
    case 상수2 :    문장2;
                    break;
    default :       문장n;
                    break; }
```

2.6.6.3. for 문

반복문의 하나로 제어할 변수의 초기화, 조건, 연산이 하나의 문장으로 해결할 수 있다. 아래와 같은 형태로 초기화식, 제어 조건식, 연산식은 ;를 써서 분리되어 있고 초기화식과 연산식은 콤마(,)를 사용하여 여러 개의 수식이 사용가능하고 또한 생략도 가능하다. 제어 조건식은 반드시 하나만 설정하거나 생략 가능하다. 처음에는 초기화식에 따라 제어변수를 초기화하고, 제어 조건식을 점검하고, 제어 조건식이 성립하면 문장을 실행하고, 제어변수를 연산식에 따라 연산하고, 제어 조건식을 다시 점검하고, 제어 조건식이 성립하면 문장을 실행하고, 제어변수를 연산식에 따라 연산한다. 이러한 과정을 제어 조건식이 충족되는 동안 반복적으로 수행한다. 문장이 제어변수 연산식에 포함된 형태인 경우 문장은 생략가능하다. 문장은 블록이 될 수 있다.

for(제어변수 초기화식 ; 제어변수 제어 조건식 ; 제어변수 연산식) 문장 ;

2.6.6.4. while 문

반복문의 하나로 초기화식과 연산식이 생략된 for문이라 생각하면 된다. 아래와 같은 형태로 조건이 충족되거나 조건식의 결과가 0이 아닌 동안 반복적으로 문장을 수행한다. 문장은 블록이 될 수 있다.

while(조건) 문장 ;

2.6.6.5. do-while 문

아래와 같은 형태로 while문과 비슷하나 while문이 반복문을 수행하기 전에 조건을 점검하나 do-while문은 반복문을 실행한 다음에 조건을 점검한다. 즉 do-while문은 적어도 한 번은 반복문이 수행이 되나 while문은 한 번도 수행되지 않을 수 있다. 문장은 블록이 될 수 있다.

```
do { 문장 ; } while( 조건 ) ;
```

2.6.6.6. break 문

반복문이나 조건문 등 제어문의 실행에서 현재의 실행을 중지하고 제어문을 빠져나가게 한다.

2.6.6.7. continue 문

반복문이나 조건문 등 제어문의 실행에서 제어문을 완전히 빠져나가지 말고 현재 차례의 실행만을 중지하고 다음 횟수의 반복을 진행하게 한다.

2.6.6.8. goto 문

어셈블리어의 JMP명령과 유사하게 미리 정해져 있는 label의 위치로 점프하게 한다. 아래의 예와 같은 경우 goto문을 만나면 문장2는 실행이 되지않고 goto문이 가리키는 label1으로 점프하여 문장1이 수행된다. label 이름 뒤에는 어셈블리어 프로그래밍에서 처럼 :(콜론)을 붙인다.

```
                goto label1;
                문장2;
label1:         문장1;
```

2.6.7. 함수와 변수

2.6.7.1. 프로토타입 선언

- 프로토타입 : 프로그램에서 사용할 함수이름과 리턴값, 인수(파라미터)의 데이터형을 main() 함수 위에서 미리 정의하는 것을 프로토타입 선언이라 한다. 다음의 형태로 프로토 타입을 정의한다. 함수의 내용을 main() 함수 위에서 정의한 경우는 불필요하다.

  리턴데이터형 함수명([인수1의 데이터형, 인수2의 데이터형, ...]);

  다음의 형태로 인수들을 명기 해도 된다.

  리턴데이터형 함수명([데이터형 인수1,데이터형 인수2, ...]);

2.6.7.2. 일반 함수의 정의

함수는 반드시 하나의 블럭으로 되어 있어 어떤 내용을 처리하고 그 결과를 리턴하는 등 실제적인 프로그램을 기술하는 부분이다. 다음의 형태를 띤다.

```
리턴데이터형  함수명([데이터형 인수1,데이터형 인수2, ...])
{ 프로그램 내용 }
```

- 리턴데이터형 : 함수는 내용을 수행하고 리턴데이터가 있는 경우와 없는 경우가 있는데 함수를 call한 쪽으로 결과를 돌려보내는 경우 프로그램 내용 끝에 다음과 같은 명령을 표시한다.

```
return 리턴데이터;
```

리턴데이터형이 정수형인지, 문자형인지, 실수형인지 등 무엇인지를 기술하는 부분이다. 결과를 리턴하지 않는 경우 void를 쓴다.

- 함수명 : 함수의 이름을 나타내는 심볼로 알파벳 대문자와 소문자과 숫자와 _가 가능하며 대 소문자가 구별된다. 2.4.3.4절의 변수나 상수 이름 짓는 법에 따라 이름을 만들면 된다. main 함수는 함수명이 main으로 반드시 C 프로그램에서 만들어야 하는 함수이다.

- 인수 : 프로그램을 처리할 때 전달받을 값은 인수들을 통해 하는 데 함수를 정의할 때 인수들의 데이터형과 함께 ( )안에 써 넣는다. 전달받을 값이 없으면 인수를 안쓰거나 void라고 쓴다.

- 프로그램 내용 : { }안에 정의하여 블록을 이루며 실제 함수의 실행코드가 위치한다.

2.6.7.3. 라이브러리 함수

CodeVisionAVR은 ANSI C언어의 표준 라이브러리의 함수들을 제공해준다. 자세한 것은 CodeVisionAVR manual(http://www.hpinfotech.ro)를 참조하거나 CodeVisoinAVR C컴파일러의 help를 참조하길 바랍니다.

- stdio.h : 이 헤더파일을 include시키면 UART에서 사용할 수 있는 getchar(), putchar(), puts(), putsf(), printf(), vprintf(), psprintf(), scanf() 등의 기본 입출력 함수를 사용할 수 있다.

- stdlib.h : 이 헤더파일을 include시키면 atoi(), itoa(), ftoa(), atof(), rand(), *malloc(), free() 등의 기본 라이브러리 함수를 사용할 수 있다.

- math.h : 이 헤더파일을 include시키면 abs(), max(), fmax(), min(), fmin(), sign(), sqrt(), ceil(), floor(), fmod(), exp(), log(), log10(), pow(), sin(), cos(), tan(), sinh(), cosh(), tanh(), asin(), acos(), atan(), atan2() 등의 수학 함수를 사용할 수 있다.

- string.h : 이 헤더파일을 include시키면 *strcat(), *strcatf(), strcmp(), *strcpy(), strnlen() 등의 문자열 조작용 함수를 사용할 수 있다.

2.6.7.4. 인터럽트 함수의 정의

AVR은 다수의 인터럽트 소스를 가지며 이를 위한 특별한 정의가 요구된다. 다음의 형태를 띤다.

```
interrupt [인터럽트벡터번호] void 인터럽트함수명( void )
{ 인터럽트 서비스 프로그램 내용 }
```

- 인터럽트 함수명 : 인터럽트 함수의 이름을 나타내는 심볼로 알파벳 대문자와 소문자과 숫자와 _가 가능하며 대 소문자가 구별된다. 앞서 설명된 변수나 상수 이름 짓는 법에 따라 이름을 만들면 된다.

- 인터럽트 벡터번호 : 각 AVR 디바이스가 제공할 수 있는 인터럽트 소스 가운데 선언할 인터럽트를 지정하는 번호로 다른 일반 함수와 구별하기 위해 반드시 써줘야 한다. 헤더파일에 인터럽트 소스명이 정의되어 있다. ATmega128의 경우 mega128.h에 리셋 인터럽트를 제외한 34개의 인터럽트 소스명이 정의되어 있는데 외부인터럽트 0~7(인터럽트 번호 2~9)은 EXT_INT0~EXT_INT7로 정의되어 있다.

2.6.7.5. 변수

변수는 값의 변화가 있는 데이터를 지칭하며 값을 마음대로 바꿀 수 있는 기억장소를 의미한다. 변수는 전역(global), 지역(local), 정적(static), 외부(extern)변수, 레지스터로 나뉠 수 있다.

- 전역변수 : 소스 파일 내부에 있는 모든 함수나 프로그램 블록에서 사용할 수 있다. 소스파일 전체에서 서로 공유하는 데이터가 있는 경우에 사용하면 좋다.

- 지역변수 : 하나의 프로그램 블록 안에서 선언되고 사용되는 변수로 선언된 블록 밖에서는 변수로서의 기능을

상실한다. 프로그램 블록이 실행되는 순간에 지역변수를 위한 기억장소가 확보되고 실행이 종료되면 자동적으로 해제되어 다른 프로그램 블록이 실행되는 동안에는 메모리에 전혀 영향을 주지 않는다.

- 레지스터변수 : CPU의 레지스터를 사용하여 변수를 저장한다. 그러므로 연산속도가 향상될 수 있다. 일반적으로 지역변수에만 지정할 수 있고 레지스터의 갯수가 한정이 되어 있으므로 일정 한도가 넘어서면 나머지들은 레지스터변수로 선언되었어도 무시당하고 지역변수에 할당된다.

- 정적변수 : 아래와 같이 자료형 앞에 static이라는 지시어를 데이터 형 앞에 써 정의하며 정적변수가 선언된 함수가 종료되더라도 그 값을 계속 유지하며 프로그램이 종료될 때까지 유효한 변수이다.

```
static          데이터형          변수명1[=초기값, 변수명2=초기값2, .... ];
```

- 외부변수 : 다른 소스 파일에 정의된 변수를 지칭한다. 프로그램 선두에 아래와 같이 extern이라는 지시어를 데이터형 앞에 써 정의하여 사용하며 프로그램이 종료될 때까지 유효하다. 여러 사람이 공동으로 하나의 프로그램을 개발할 때 다른 사람이 만든 변수를 쉽게 사용할 수 있도록 해준다. 함수의 경우도 마찬가지로 프로그램 선두에 프로토타입선언문 앞에 extern 지시어를 붙여서 선언하여 다른 소스 파일에 정의된 함수를 사용하며 프로그램이 종료될 때까지 유효하다.

```
extern      데이터형          변수명1[=초기값, 변수명2=초기값2, .... ];
```

2.6.7.6. 어셈블리어와 결합

함수내의 C 소스 중간에 어셈블리어를 사용하기 시작하려면 지시어 #asm을 사용하고 어셈블리어의 사용을 끝내려면 지시어 #endasm을 사용한다. #asm("어셈블리어 명령어")도 가능하다.

2.6.8. 포인터와 배열

2.6.8.1. 배열 선언 방법

배열의 선언은 아래와 같이 데이터형, 배열명을 써주고 배열차원수를 [ ]기호로 묶어주면 된다.

```
데이터형          배열명[배열항목수1][배열항목수2] ... [배열항목수n];
```

- 데이터형 : 앞에 나온 데이터형을 사용한다.

- 배열명 : 배열의 이름을 나타내는 심볼로 알파벳 대문자와 소문자과 숫자와 _가 가능하며 대 소문자가 구별된다. 2.4.3.4절의 변수나 상수 이름 짓는 법에 따라 이름을 만들면 된다

- 배열항목수 : 배열의 크기를 설정한다. 주의할 점은 배열의 마지막 인덱스는 (배열차원수-1)이고 첫 인덱스는 0임에 유의해야 한다. 예로 char buf[3];와 같이 선언된 배열 buf의 첫 번째 항목은 buf[1]이 아니라 buf[0]이며 두 번째 항목은 buf[2]가 아니라 buf[1]이고 세 번째 항목이자 마지막 항목은 buf[3]이 아니라 buf[2]이다.

2.6.8.2. 배열 초기화

- 1차원 배열의 초기화 : 배열은 선언과 동시에 값을 초기화 할 수 있는데 { } 기호를 사용하여한다. 1차원 배열의 경우 아래와 같이 차례대로 값을 나열하고 ,(콤마)로 구분하고 { }기호로 묶어주면 된다. 인덱스 0에서부터 차례대로 n-1까지 순서대로 주어진 항목값에 따라 초기화된다. 배열항목수는 n보다 작거나 같을 수 있는 데 작은 경우 초기화 되지 않은 항목의 값은 자동적으로 0으로 초기화된다.

```
데이터형      배열명[배열항목수] = { 1번째항목값, .... , n번째항목값 } ;
```

1차원 배열의 선언과 초기화를 동시에 하는 경우 아래와 같이 배열항목수를 빼고 해도 된다. 그러나 이 경우 선언된 배열명의 항목수는 n으로 됨에 유의해야 한다. 그리고 2차원 이상의 경우에는 허용되지 않는다.

```
데이터형      배열명[] = { 1번째항목값, .... , n번째항목값 } ;
```

- 문자형 배열의 초기화 : 문자열의 경우에는 초기화할 때 { }기호를 사용하지 않고 큰 따옴표를 써서 해도 된다. 그래서 아래의 두 선언문은 같다. 정수형이나 다른 데이터형의 경우에는 허용되지 않는다.

```
char        buf[11] = "I am HHCHOI";
char        buf[[11] = {'I', ' ', 'a', 'm', ' ', 'H', 'H', 'C', 'H', 'O', 'I'};
```

- 다차원 배열의 초기화 : 1차원배열의 초기화와 비슷하며 다만 2중으로 { }기호를 사용한다. 1번째값1이 배열

의 배열명[0][0] ... [0]의 항목값으로 초기화되고 차례대로 m번째항목값1이 배열명[0][0] .... [m-1]의 항목값으로 초기화된다. 1번째값2은 배열명[1][0] ... [0]의 항목값으로 초기화되고 차례대로 p번째항목값2이 배열명[1][0] ... [p-1]의 항목값으로 초기화된다. 마찬가지의 과정을 거쳐 차례대로 초기화되며 초기화되지 않은 항목의 값은 자동적으로 0으로 초기화된다. 여기에서 1차원 배열과 달리 항목수는 반드시 기입해야 한다.

데이터형 배열명[항목수1] ... [항목수n] = { {1번째값1, ... , m번째항목값1}, {1번째값2, ... , p번째항목값2} ... } ;

2.6.8.3. 포인터 선언 방법

아래와 같이 일반 변수와 같은 형식으로 선언하되 변수의 이름 앞에 *(애스터리스크) 기호를 붙인다. *는 포인터 변수의 내용을 의미하는 연산자이다.

데이터형 *포인터이름;

2.6.8.4. 포인터 초기화 및 참조 방법

- 일반 변수 주소를 포인터 변수에 대입하는 방법 : 아래와 같은 형태로 주소연산자 &를 사용하여 포인터에 데이터를 할당하고 초기화할 수 있다.

```
데이터형 일반변수이름=초기값;
데이터형 *포인터이름;
포인터이름 = &일반변수이름;
```

위의 형태는 아래와 같은 형태로 대체될 수 있다.

```
데이터형 일반변수이름=초기값;
데이터형 *포인터이름 = &일반변수이름;
```

- 문자열을 포인터로 선언하는 법 : 포인터를 선언하면서 동시에 문자열 데이터로 초기화할 때는 아래와 같이 큰따옴표를 사용하면 된다.

```
char *포인터이름 = “문자열”;
```

- 포인터로 배열을 가리키게 하기 위해 배열이름을 포인터 변수에 대입하는 방법 : 미리 선언된 배열을 미리 선언된 포인터 변수로 가리키게 하기 위해 다음과 같이 배열이름을 포인터이름에 대입할 수 있다.

```
데이터형 배열명[배열항목수] = { 1번째항목값, .... , n번째항목값 }, *포인터이름;
포인터이름 = 배열이름;
```

위의 형태는 아래와 같은 형태로 대체될 수 있다.

```
데이터형 배열명[배열항목수] = { 1번째항목값, .... , n번째항목값 };
데이터형 *포인터이름 = 배열이름;
```

- 포인터로 배열을 가리키게 하기 위해 배열의 첫번째 항목의 주소를 포인터 변수에 대입하는 방법 : 미리 선언된 1차원배열을 미리 선언된 포인터 변수로 가리키게 하기 위해 다음과 같이 주소연산자 &를 사용하여 배열의 첫번째 항목의 주소를 사용할 수 있다.

```
데이터형 배열명[배열항목수] = { 1번째항목값, .... , n번째항목값 }, *포인터이름;
포인터이름 = &배열이름[0];
```

- 포인터 변수에서 값 읽기 방법 : 미리 선언되고 사용중인 포인터변수의 내용을 그 내용과 같은 자료형의 일반변수에 값을 써 넣을 때는 연산자 *를 사용해 아래와 같이 한다.

```
일반변수이름 = *포인터이름;
```

- 포인터 변수에 값 쓰기 방법 : 미리 선언되고 사용중인 포인터 변수의 내용과 같은 자료형의 일반변수 값을 포인터 변수에 값을 써 넣을 때도 연산자 *를 사용해 아래와 같이 한다.

```
*포인터이름 = 일반변수이름;
```

2.6.8.5. 포인터의 연산

포인터변수는 ++, --, +, -, +=, -= 등의 연산자를 사용하여 연산이 가능하다. 만약 다음과 같이 포인터와 배열이 선언된 경우 *ptr++ 가 가리키는 값은 buf[1] 즉 b와 같다. 여기에서 ++나 --가 *연산자보다 우선하기 때문에 *ptr++ 와 *(ptr++)는 동일하고 , *ptr-- 과 *(ptr--)은 동일함에 유의해야 한다.

```
char buf[6] = "abcdef";  char *ptr = buf;
```

2.6.8.6. 함수 포인터

● 함수포인터의 선언 : 함수프로토타입 선언과 같이 아래처럼 하면 함수 포인터를 선언할 수 있다. 함수프로토타입 선언에서처럼 인수들은 생략하고 인수들의 데이터형만 ( )안에 나열해도 된다. 인수가 없는 경우에는 void를 사용한다.

리턴데이터형 (* 함수명) ([데이터형 인수1, 데이터형 인수2, ...]);

● 함수포인터의 무조건 점프명령에의 활용 : 아래와 같이 function_ptr이라는 함수포인터가 선언된 경우

void (* function_ptr) (void);

function_ptr에 다음과 같은 식으로 절대번지값(예: 0x4000)을 대입을 하고 콜을 하면 프로그램카운터(PC)값은 절대번지값 0x4000으로 바뀌어 0x4000번지부터 명령을 수행하게 된다.

function_ptr = (void *) 0x4000; function_ptr();

2.6.8.7. 배열과 포인터의 차이점

● 선언 : 배열은 선언시 차원을 미리 지정해줘야 하나 포인터는 그럴 필요없다.

● 초기화 : 배열은 초기화가 쉬우나 포인터는 미리 메모리를 할당받고 해야 한다.

● 데이터 접근 : 배열은 [ ]기호사이에 넣어진 값 즉 인덱스를 사용하여 바로 필요한 위치의 데이터에 접근 할 수 있으나 포인터는 포인터 변수의 연산을 통해 데이터를 읽고 써야 한다.

● 메모리의 사용 : 배열이 선언될 때 필요한 메모리가 할당되어 중간에 메모리 크기를 변경할 수 없으나 포인터는 그렇지 않다.

2.6.9. typedef, structure, union, enum

2.6.9.1. typedef를 사용한 새로운 데이터형 정의법

typedef는 자주 사용하는 데이터형을 기억하기 쉽거나 편리한 이름을 갖는 새로운 데이터형을 정의하여 쓸 수 있도록 한다. 사용 형식은 아래와 같다.

typedef 기존데이터형 새데이터형;

일반적으로 다음과 같은 데이터형의 변경이 많이 쓰인다.

```
typedef   unsigned char   byte;
typedef   unsigned int    word;
```

2.6.9.2. 구조체의 개념과 필요성

서로 다른 유형의 변수들을 새로운 하나의 데이터형으로 묶어서 정의한 것으로 구조체(structure)는 다양한 유형의 변수를 구성원으로 갖는 변수의 묶음이다. 인사기록카드를 만드는 경우 사원의 이름, 사번, 주민번호, 나이, 최종학력 등이 필요한데 이러한 것은 사원별로 나누어지는 데이터이므로 각 사원에 대하여 하나로 묶어 처리하면 취급, 프로그램 개발과 이해가 매우 쉽겠다.

2.6.9.3. 구조체 선언법과 초기화

● 방법1 : 우선 아래와 같은 형태로 구조체항목들을 나열하여 { }로 묶어 구조체이름으로 새로운 데이터형을 정의한다.

```
struct      구조체이름      {
    데이터형 구조체항목1;          데이터형 구조체항목2;
    데이터형 구조체항목n; };
```

이렇게 만든 구조체로 지시어 struct와 구조체이름을 사용하여 변수를 아래와 같이 선언한다.

struct 구조체이름 변수이름;

선언과 동시에 초기화할 경우에는 아래와 같이 한다.

struct 구조체이름 변수이름 = {구조체항목1의 초기값, 구조체항목2의 초기값, 구조체항목n의 초기값};

● 방법2 : 아래와 같은 형태로 구조체항목들을 나열하여 { }로 묶어 구조체이름으로 새로운 데이터형을 정의하며 동시에 정의된 구조체형태의 변수를 선언하고 초기화할 수 있다.

```
struct          구조체이름          {
    데이터형 구조체항목1;                  데이터형 구조체항목2;
    데이터형 구조체항목n;
} 변수이름 = {구조체항목1의 초기값, 구조체항목2의 초기값, 구조체항목n의 초기값};
```

- 방법3 : typedef를 사용하여 우선 아래와 같은 형태로 구조체항목들을 나열하여 { }로 묶어 구조체이름으로 새로운 데이터형을 정의한다.

```
typedef     struct    _구조체이름          {
    데이터형 구조체항목이름1;              데이터형 구조체항목이름2;
    데이터형 구조체항목이름n; } 구조체이름;
```

이렇게 만든 경우 지시어 struct를 쓸 필요없이 구조체이름만 사용하여 변수를 아래와 같이 선언한다.

```
구조체이름    변수이름;
```

구조체 변수 선언과 동시에 초기화할 경우에는 아래와 같이 한다.

```
구조체이름    변수이름 = {구조체항목이름1의 초기값, 구조체항목이름2의 초기값, 구조체항목이름n의 초기값};
```

- 방법4 : typedef를 사용하여 아래와 같은 형태로 구조체를 정의하고 변수의 선언과 초기화는 방법3과 똑같이 할 수도 있다.

```
typedef     struct    {
    데이터형 구조체항목이름1;     데이터형 구조체항목이름2;
    데이터형 구조체항목이름n;              } 구조체이름;
```

- 구조체 포인터 변수나 배열 선언 : 앞서의 방법1~4 가운데 하나를 사용해서 구조체 이름을 선언하고 변수를 포인터나 배열로 선언할 수 있다. 2.4.9절에 주어진 일반 데이터형의 포인터나 배열 선언 방법과 동일하다. 예로 구조체 이름을 방법3이나 4를 이용하여 선언한 경우 포인터나 배열변수는 아래와 같이 하면 된다.

```
구조체이름    *구조체포인터변수이름, 구조체배열이름[배열항목수];
```

2.6.9.4. 구조체 사용법

- . 연산자를 통한 항목 접근 : 구조체 변수의 항목은 아래와 같이 .(도트)연산자를 통해 접근한다.

```
구조체변수이름.구조체항목이름
```

- -> 연산자를 통한 항목 접근 : 구조체 포인터 변수의 항목은 아래와 같이 -> 연산자를 통해 접근한다.

```
구조체보인터변수이름->구조체항목이름
```

2.6.9.5. 공용체(union)

- 개념 : 항목들이 같은 메모리 공간을 점유하는 것을 제외하고 구조체와 동일하다. 하나의 union 변수가 점유하는 메모리의 크고 변수의 항목에서 가장 큰 것과 동일하다. AVR에서는 SRAM에 저장된다. 아래의 예를 보면 쉽게 필요성과 사용법을 이해 할 수 있을 것이다. 같은 값의 다른 표현에 활용할 수 있다.

```
union          tag_length          {
    unsigned int        all;      unsigned char      byte[2];          } data;
data.all = 0x1230;        //data공용체에 16진수 0x1234할당
low = data.byte[0] ;      // low에 data의 하위바이트를 할당 즉 low=0x30와 동치
high = data.byte[1] ;     // hig에 data의 상위바이트를 할당 즉 high=0x12와 동치
```

- 선언법과 초기화 : 구조체와 같다.
- 사용법 : 구조체와 같다.

2.6.9.6. 열거형 상수

- 개념 : 정수형 상수의 일종으로 상수값에 일종의 별명을 부여하여 열거해 둔 것이다.
- 선언법과 초기화 : 선언법과 초기화는 구조체와 비슷한데 각 항목에 할당되는 값은 만약 n-1번째 항목의 값이 정수 N-1이라면 n번째 항목 값은 N이다. 초기화는 첫번째 항목값만 정의해주면 된다. 첫번째 항목을 정의하지 않으면 첫번째 항목에 0이 할당된다. 아래의 예는 상수 d_of_weeks의 sunday항목은 1에, monday는 2에,..... saturday에는 7을 할당하는 것이다.

```
enum          days      {sunday=1, monday, tuesday, wednesday, thursday, friday, saturday} d_of_week;
```

3. 기본 프로그래밍과 입출력 실험

3.1. 기본 프로그래밍 연습

3.1.1. 실험1 : 내부램에 데이터 쓰기1

- 예제 : 내부램에 데이터를 쓰는 다양한 방법을 익힌다. 내부램 0x130-0x136 번지에 차례대로 숫자 1, 2, 0xff와 문자 'A', 'a', 'F', 'f' 를 써 넣어라. 즉 다음을 하라.

  (0x130) <-1, (0x131) <-2, (0x132) <-0xff, (0x133) <-'A', (0x132) <-'a', (0x135) <-'F', (0x136) <-'f'

- 프로그램 예 :

```
#include <mega128.h>
#define DBYTE ((unsigned char volatile *) 0)
#define lm136 *((unsigned char *) 0x0136)  /* define에는 ;로 종결하지 않음에 주의 */
unsigned char *plm130 = 0x0130;
void main(void){
    *plm130++ = 1;                  /* (0x130) <- 1, plm130h = plm130 + 1 */
    *(plm130++) = 2;                /* *plm130++ = *(plm130++) */
    *plm130 = 0xff ;                /* (0x132) <- 0xff */
    *++plm130 = 'A' ;               /* plm130 = plm130 +1 , *plm130 <- 'A' */
    *(++plm130) = 'a';              /* *(++plm130) = *++plm130 */
    DBYTE[0x0135] = 'F';            /* (0x135) <- 'F' */
    lm136 = 'f';                    /* (0x136) <- 'f' */
}
```

- 실험방법 : ①CodeVisionAVR의 에디터를 통해 실행파일을 만든 후 AVR Studion4디버그세션에 들어간 후 메인메뉴의 [View->Memory]를 실행하여 Memory라는 이름의 창을 생성하고 창에서 [Data]을 선택한다. 또한 [View->Register]를 실행하여 Register라는 이름의 창을 생성한다. ②디버그세션에서 메인메뉴 [Debug]의 하위 메뉴인 [Step Into], [Auto Step], [Run to Cusor] 등을 적절히 활용하면서 Memory, Register, I/O View 창을 통해 내부램과 레지스터의 값의 변화를 확인하라. 성공적으로 수행된 경우 내부램 0x130-0x136에는 차례대로 1,2, 0xff, 0x41, 0x61, 0x46, 0x66값이 저장된다.

- 과제 : 내부램 0x140-0x147 번지에 차례대로 문자 'A', 'b', 'C', 'D', 'E', 'F', 'G', 'h'를 다양한 방식으로 써 넣어라.

3.1.2. 실험2 : 내부램에 데이터 쓰기2

- 예제 : 내부램 0x130-0x17F 번지에 차례대로 0xff을 써 넣어라.

- 프로그램 예 :

```
#include <mega128.h>
unsigned char *plm130 = 0x0130;
void main(void){
    for(;plm130<=0x17f;){
    *plm130++ = 0xff;  /* 포인터 plm130가 가리키는 곳에 ff를 쓰고 다음 위치를  가르키도록 1증가 시킨다. */
    }
}
```

- 실험방법 : 실험1과 같은 방식으로 수행하라. 성공적으로 수행된 경우 내부램 0x130H-0x17F 번지에 차례대로 0xff 값이 저장된다.

- 과제 : #define DBYTE ((unsigned char volatile *) 0) 를 활용하여 같은 역할을 하도록 하라.

3.1.3. 실험3 : EEPROM에 데이터 쓰기와 읽기

● 예제 : EEPROM의 일정영역에 데이터를 저장시켜 놓고 읽는 방법을 익힌다. EEPROM에 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x67 등 10개의 데이터를 쓰고 저장된 값을 읽어 들여 SRAM 0x130 번지를 시작으로 하여 차례대로 저장한다.

● 프로그램 작성시 유의점 : 1.5.6.4절과 1.5.6.5절을 참조하여 작성한다.

● 프로그램 예 :

```
#include <mega128.h>
#include <mem.h> // void pokeb(unsigned int addr, unsigned char data) 명령어를 사용해 SRAM에 쓸 수 있다.
#include <delay.h>
unsigned char read_eeprom(unsigned int addr)          {          // EEPROM 읽기
      while((EECR & 0x02) == 0x02);          // EEWE = 0 인가를 확인한다.
      EEAR = addr;
      EECR |= 0x01;                          // EERE =1
      return EEDR;
}
void write_eeprom(unsigned int addr,unsigned char data)          {          // EEPROM에 쓰기
            while((EECR & 0x02) == 0x02);          // EEWE = 0 인가를 확인한다.
            EEAR = addr;
            EEDR = data;
            EECR |= 0x04;                          // EEMWE = 1
            EECR |= 0x02;                          // EEWE = 1
}
void main(void){
      unsigned char  arr[ ]={0x3f, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x67 };
      int i, j;
      j = 0x00;
      while(arr[j] != 0) write_eeprom(j,arr[j++]);
      delay_ms(100);
      for(i=0x0130, j=0;i<=0x013a; i++, j++) {
            pokeb(i, read_eeprom(j));                          // 읽은 데이터를 램에 쓴다.
            delay_ms(10);}
}
```

● 실험방법 : 실험1과 비슷한 방식으로 수행하라. EEPROM창을 띠우고 이를 통해 성공적으로 써졌는지 확인하라. 성공적으로 수행된 경우 내부램 0x130-0x13a에는 차례대로 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x67 값이 저장된다.

● 과제 : 플래쉬롬에 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x67 등 10개의 데이터를 저장해두고 저장된 값을 읽어 들여 실험1을 참조하여 다른 방식으로 EEPROM에 0x00번지를 시작으로 하여 차례대로 저장하는 프로그램을 작성하라.

3.1.4. 실험4 : 롬 데이터 읽기

● 예제 : 플래쉬 롬의 일정영역에 데이터를 저장시켜 놓고 읽는 방법을 익힌다. 롬에 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x67 등 10개의 데이터를 저장해두고 저장된 값을 읽어 들여 SRAM 0x130 번지를 시작으로 하여 차례대로 저장한다.

● 프로그램 예 :

```
#include <mega128.h>
#include <mem.h> // void pokeb(unsigned int addr, unsigned char data) 명령어를 사용해 SRAM에 쓸 수 있다.
flash unsigned char arr[ ]={0x3f, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x67 };
void main(void){
      int i, j;
      for(i=0x0130, j=0;i<=0x013a; i++, j++)    pokeb(i, arr[j]);
}
```

● 실험방법 : 실험1과 같은 방식으로 수행하라. 성공적으로 수행된 경우 내부램 0x130-0x13a에는 차례대로 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x67 값이 저장된다.

● 과제 : 플래쉬롬에 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x67 등 10개의 데이터를 저장해두고 저장된 값을 읽어 들여 실험1을 참조하여 다른 방식으로 SRAM에 0x130번지를 시작으로 하여 차례대로 저장하는 프로그램을 작성하라.

3.1.5. 실험5 : 롬데이터 검색1

● 예제 : 플래쉬 롬의 일정영역에 저장된 16개의 데이터에서 5의 배수를 뽑아 SRAM 0x131부터 저장하고 5의 배수의 개수를 0x130에 저장한다.

● 프로그램 예 :

```
#include <mega128.h>
#define DBYTE ((unsigned char volatile *) 0)
flash unsigned char  arr[ ]={12, 5, 2 , 25, 3, 9, 15, 19, 25, 99, 45, 56, 77, 99, 34, 26 };
void main(void){
      int  i = 0x0131, j = 0, k =0 ;
      for(;j<=15; j++)                  /* j=0으로 초기화하는 것은 선언문에서 했으므로 생략 */
            if ( arr[j]%5 == 0 ) { DBYTE[i++] = arr[j]; k++ ; }
      DBYTE[0x0130] = k;

}
```

● 실험방법 : 실험1과 같은 방식으로 수행하라. Eeprom창과 Data, IO view 창을 띠우고 변화하는 모양을 성공적으로 수행된 경우 내부 램 0x130H~0x135H에 차례로 0x05, 0x05, 0x19, 0x0F, 0x19, 0x2D가 저장된다.

● 과제 : 프로그램 메모리 일정영역에 저장된 16개의 데이터에서 5의 배수가 아닌 것을 뽑아 SRAM 0x131부터 저장하고 그 개수를 0x130에 저장하는 프로그램을 작성하라.

3.1.6. 실험6 : 롬데이터 검색2

● 예제 : 롬의 일정영역에 저장된 16개의 데이터에서 가장 작은 수를 뽑아 SRAM 0x130번지에 저장한다.

● 프로그램 예 :

```
#include <mega128.h>
#define DBYTE ((unsigned char volatile *) 0)
flash unsigned char arr[ ]={12, 5, 2 , 25, 3, 9, 15, 19, 25, 99, 45, 56, 77, 99, 34, 26 };
void main(void){
      unsigned char j = 0;
      DBYTE[0x00130] = arr[0];                        /* 가장 작은 수가 될 후보로 arr[0]로 */
      for(j=1;j<=15; j++)
            if ( DBYTE[0x0130] > arr[j] ) DBYTE[0x0130] = arr[j];
}
```

● 실험방법 : 실험1과 같은 방식으로 수행하라. 성공적으로 수행된 경우 내부 램 0x0130에 2가 저장된다.

● 과제 : 프로그램 메모리 일정영역에 저장된 16개의 데이터에서 가장 큰 값을 찾아 SRAM 0x130에 저장하라.

3.1.7. 실험7 : 8비트 2진수의 BCD 변환

● 예제 : 8비트 2진수를 3자리수 BCD 값으로 변환하는 프로그램을 작성한다. 주어진 2진수 헥사값(예: 0xFF)을 BCD로 변환하여 100의 자리수는 SRAM 0x0130번지에 10의 자리수는 0x0131번지의 상위 4비트에 1의 자리수는 0x0131번지의 하위4비트에 저장하는 프로그램을 작성하라.

● 프로그램 작성시 유의점 : bcd.h를 include시켜서 bin2bcd()를 사용해서 해도 되나 이 경우 0~99까지의 값만 사용할 수 있다. 8비트의 2진수는 0~255의 값을 갖는다. 그러므로 결과값의 저장은 2바이트를 요구하고 100의

자리수가 저장되는 0x0130는 0~2의 값을 갖고 0x0131에는 00~99의 값을 갖는다.

● 프로그램 예 :

```
#include <mega128.h>
#define DBYTE ((unsigned char volatile *) 0)
void h2bcd(unsigned char bin){
    unsigned char tmp;
    DBYTE[0x0130] = bin/100;              /* 100의 자리수는 0x130 번지에 저장 */
    DBYTE[0x0131] = ( bin % 100 ) / 10;   /* 10의 자리수가 0x131번지의 하위 4비트에 저장 */
    tmp = ( bin % 100 ) % 10 ;            /* 1의 자리수를 tmp에 임시 저장 */
    DBYTE[0x0131] <<=4 ;                  /* 10의 자리수를 0x131번지의 상위 4비트로 저장하기 위해 4번 쉬프트 */
    DBYTE[0x0131] |= tmp ;                /* 1의 자리수를 0x131번지의 하위 4비트로 저장 */
}
void main(void){
    unsigned char bin = 0xff;             /* 주어진 2진수를 0xff로 초기화 */
    h2bcd(bin);
}
```

● 실험방법 : 실험1과 같은 방식으로 수행하라. 성공적으로 수행된 경우 내부램 0x130, 0x131에는 차례대로 0x02, 0x55 값이 저장된다.

● 과제 : 주어진 2진수 0xE0를 BCD로 변환하여 100의 자리수는 램 0x1000번지에 10의 자리수는 0x1001번지의 상위 4비트에 1의 자리수는 0x1001번지의 하위4비트에 저장하는 프로그램을 실험1을 참조하여 다른 방식으로 작성하라.

3.1.8. 실험8 : 16비트 2진수의 BCD 변환

● 예제 : 16비트 2진수를 5자리수 BCD 값으로 변환하는 프로그램을 작성한다. 주어진 2바이트 2진수의 헥사값(예: 0xF0FF)를 BCD로 변환하여 10000의 자리수는 0x130번지에 1000의 자리수는 0x131번지의 상위 4비트에 100의 자리수는 0x131번지의 하위4비트에 10의 자리수는 0x132번지의 상위 4비트에 1의 자리수는 0x132번지의 하위4비트에 저장하는 프로그램을 작성하라.

● 프로그램 작성시 유의점 : 16비트의 2진수는 0~65535의 값을 갖는다. 그러므로 결과값의 저장은 5바이트를 요구하고 10000의 자리수가 저장되는 0x130는 0~6의 값을 갖고 0x131~0x132에는 00~99의 값을 갖는다.

● 프로그램 예 :

```
#include <mega128.h>
#define DBYTE ((unsigned char volatile *) 0)
void h2bcd2(unsigned int bin){
    unsigned char tmp;
    DBYTE[0x0130] = (char)(bin/10000);                          /* 10000의 자리수 저장 */
    DBYTE[0x0131] = (char) (( bin % 10000 ) / 1000) ;           /* 1000의 자리수가 0x131번지의 하위 4비트에 저장 */
    tmp = (char) ((( bin % 10000 ) % 1000) / 100) ; /* 100의 자리수를 tmp에 임시 저장 */
    DBYTE[0x0131] <<=4 ;        /* 1000의 자리수를 0x131번지의 상위 4비트로 저장하기 위해 4번 쉬프트 */
    DBYTE[0x0131] |= tmp ;                  /* 100의 자리수를 0x131번지의 하위 4비트로 저장 */
    DBYTE[0x0132] = (char)((( (bin % 10000) % 1000) %100)/10);  /* 10의 자리수가 132번지의 하위 4비트에 저장 */
    tmp = (char)((( ( bin % 10000 ) % 1000) % 100) % 10);    /* 1의 자리수를 tmp에 임시 저장 */
    DBYTE[0x0132] <<=4 ;                    /* 10의 자리수를 0x132번지의 상위 4비트로 저장하기 위해 4번 쉬프트 */
    DBYTE[0x0132] |= tmp ;                  /* 1의 자리수를 0x132번지의 하위 4비트로 저장 */
}
void main(void){
    unsigned int bin = 0xf0ff;              /* 주어진 2진수를 0xff로 초기화 */
    h2bcd2(bin);
}
```

● 실험방법 : 실험1과 같은 방식으로 수행하라. 성공적으로 수행된 경우 내부램 0x130~132에는 차례대로 0x06, 0x16, 0x95 값이 저장된다.

● 과제 : 주어진 2진수 0xffe0를 BCD로 변환하여 10000의 자리수는 램 0x1000번지에 1000의 자리수는 0x1001번지의 상위 4비트에 100의 자리수는 0x1001번지의 하위4비트에 10의 자리수는 0x1002번지의 상위 4비트에 1의 자리수는 0x1002번지의 하위4비트에 저장하는 프로그램을 실험1을 참조하여 다른 방식으로 작성하라.

3.1.9. 실험9 : 아스키 문자의 16진수 변환

● 예제 : 아스키코드로 주어진 숫자 배열 {‘0’, ‘1’, ... , ‘9’, 'A', 'B', ..., 'F', 'a', ... , 'f'}을 16진수 헥사코드로 변환하여 내부램 0x130를 시작번지로 차례대로 저장한다.

● 프로그램 작성시 유의점 : 숫자 0~9의 값은 stdlib.h를 include시켜 itoa()함수를 사용해서 할 수도 있다. 숫자 0~9의 값은 아스키코드는 0x30~0x39이고, A~F의 값은 0x41~46, a~f의 값은 0x61~66이라는 점을 염두에 두어야 한다.

● 프로그램 예 :

```
#include <mega128.h>
#define DBYTE ((unsigned char volatile *) 0)
flash unsigned char arr[ ]= "0123456789ABCDEFabcdef";
unsigned char asc2hex(unsigned char asc){
      if ((asc >= '0') && (asc <='9'))          return (asc - '0');
      else if ((asc >= 'A') && (asc <='F'))     return ((asc - 'A')+10);
      else if ((asc >= 'a') && (asc <='f'))     return ((asc - 'a')+10);
      else       return(0xff);
}
void main(void){
      int j = 0, i=0x0130;
      for(;j<=21; j++) DBYTE[i++] = asc2hex(arr[j]);
}
```

● 실험방법 : 실험1과 같은 방식으로 수행하라. 성공적으로 수행된 경우 내부램 0x130~13f에는 차례대로 0x00~0x0F값이 0x140~145에는 0x0A~0x0F값이 저장된다.

3.1.10. 실험10 : 숫자의 아스키코드 변환

● 예제 : 한자리를 갖는 16진 숫자 배열 {0,1,2,3,4,5,6,7,8,9, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f}를 아스키코드로 변환하여 내부램 0x130를 시작번지로 차례대로 저장한다.

● 프로그램 작성시 유의점 : 숫자 0~9의 값은 stdlib.h를 include시켜 itoa()함수를 사용해서 할 수도 있다. 숫자 0~9의 값은 아스키코드는 0x30~0x39이고, A~F의 값은 0x41~46, a~f의 값은 0x61~66이라는 점을 염두에 두어야 한다.

● 프로그램 예 :

```
#include <mega128.h>
#define DBYTE ((unsigned char volatile *) 0)
flash unsigned char arr[ ]= {0,1,2,3,4,5,6,7,8,9, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};
unsigned char hex2asc(unsigned char num){
      if(num>=10) return(num+'A'-10);
      else return(num+'0');
}
void main(void){
      int j = 0, i=0x0130;
      for(;j<=15; j++) DBYTE[i++] = hex2asc(arr[j]);
}
```

● 실험방법 : 실험1과 같은 방식으로 수행하라. 성공적으로 수행된 경우 내부램 0x130~139에는 차례대로 0x30~0x39값이 0x13a~13f에는 0x41~0x46값이 저장된다.

3.1.11. 실험11 : 지연 루틴

- 예제 : 타이머를 쓰지 않고 소프트웨어에 의해 적절한 시간지연을 얻는 방법을 익힌다. 100usec시간의 지연을 주면서 내부램(0x130)에 저장된 값을 증가시키는 프로그램을 작성한다.(발진주파수는 16MHz라고 가정한다).
- 프로그램 작성시 유의점 : 다음과 같이 while 문을 반복적으로 사용하여 시간지연을 얻는 프로그램을 작성할 수 있겠다.

```
void delay(unsigned char i){
    while(i--); }
void main(void){
    delay(0x0100);
}
```

위의 프로그램을 컴파일하고 AVR Studio 4를 통해 디버그세션에 들어가 메인메뉴의 [View->Disassembler]를 클릭하여 Disassembler 창을 띠우면 while(i--) 문에 해당하는 부분에서 다음과 비슷하게 어셈블리어로 번역되어 나타난 모습을 확인 할 수 있다.

```
LDD     R30,Y+0         ; Load indirect with displacement,    2기계사이클 소요 (반복)
SUBI    R30,0x01        ;  Subtract immediate,                      1기계사이클 소요 (반복)
STD     Y+0,R30         ;  Store indirect with displacement,  2기계사이클 소요 (반복)
SUBI    R30,0xFF        ; Subtract immediate,                       1기계사이클 소요 (반복)
BRNE    PC-0x04         ; Branch if not equal,                      1/2기계사이클 소요(반복할때 1 벗어날때 2)
ADIW    R28,0x01        ; Add immediate to word,              2기계사이클 소요
RET                     ; Subroutine return,                        4기계사이클 소요
```

위의 루틴에서 i값에 대하여 반복 수행하는 데 걸리는 시간을 계산해 (i+1)*(2+1+2+1+1)+7 기계사이클과 같다. i값은 0~255를 갖게 되니 최소지연값은 14기계사이클(발진주파수16MHz일때 14/16=0.875usec) 최대지연값은 1799기계사이클(발진주파수 16MHz일때 1799/16=112.4375usec)가 된다. 결국 원하는 delaytime_us 마이크로초에 대한 i값은 (i+1)*(2+1+2+1+1)+7=16*delaytime_us에 의하여 다음처럼 주어진다.

```
i = delaytime_us*16/7 -2
```

위 식에 따라 100usec를 얻으려면 i=226으로 하면 될 것이다. 만약 112usec보다 긴 시간의 지연을 얻으려면 delay()를 여러분 호출하면 될 것이다. 위의 루틴으로 정확한 시간지연을 얻기에는 i값이 원하는 지연시간에 대하여 정수로 딱 떨어지지도 않아 오차가 생기며 마이컴의 발진주파수에도 좌우되는 등 상황에 따라 시간지연값이 들쭉날쭉해 적절하지 않을 수도 있다. CodeVisionAVR에는 이를 위해 함수가 준비되어 있다. delay.h 헤더파일을 include시켜 delay_ms(unsigned int i), delay_us(unsigned int i)를 사용하여 밀리초나 마이크로초 단위로 비교적 정확하게 시간 지연을 얻을 수 있다. 그러나 인터럽트 등에 의해 지연이 발생할 수 있어 아주 정확한 것은 아니다. 아주 정확한 시간 지연은 타이머를 사용하거나 DS1302와 같은 RTC IC를 사용한다.

- 프로그램 예1 :

```
#include <mega128.h>
#include <delay.h>
#define DBYTE ((unsigned char volatile *) 0)
void main(void){
    for(;;){    DBYTE[0x0130] += 1;          delay_us(100); }
}
```

- 프로그램 예2 :

```
#include <mega128.h>
#define DBYTE ((unsigned char volatile *) 0)
void delay(unsigned char i){
    while(i--); }
void main(void){
    for(;;){    DBYTE[0x0130] += 1;          delay(226); }
}
```

- 실험방법 : 실험1과 같은 방식으로 수행하라. 시뮬레이션에서 시간 지연은 호스트 PC의 성능에 좌우된다. 시뮬레이션에서의 지연시간이 실제 하드웨어를 구현했을 때의 시간지연과는 다르다는 것을 명심해야 한다. 실제 하드웨어에 구현하였을 때는 프로그램 예2의 delay()에 의한 시간지연은 마이컴의 발진주파수에 좌우된다.

3.2. I/O Port

8051은 CPU와 외부장치를 연결해주기 위해 양방향성을 갖는 6개(PORTA~PORTF)의 8비트 입출력(I/O) 포트와 1개(PORTG)의 5비트 입출력 포트를 갖고 있다. 8051과 달리 LED 1개 정도는 구동하기에 충분한 전류 40mA를 최대구동전류로 출력하므로 LED를 포트에 연결할 때 전류흡입형으로 구현할 필요는 없다.

3.2.1. 관련 레지스터

3.2.1.1. DDRx 레지스터

입출력의 방향설정을 하여 DDRA~DDRG레지스터에 입출력포트에 대응하는 해당 비트에 1을 쓰면 출력으로 0을 쓰면 입력으로 설정 된다. 초기값은 0으로 설정되어 있다. 비트별로 만약 포트 A의 비트3를 출력으로 설정하려면 DDRA.3 = 1로 하면 되 포트 A를 전부 출력으로 설정하려면 DDRA = 0xff로 하면 된다.

3.2.1.2. PORTx 레지스터

DDRx의 값을 조절하여 출력으로 설정된 경우 PORTx 레지스터에 해당하는 값을 쓰면 된다. 비트별로 설정하려면 PORTx.n = 1의 형태로 하면 된다(x는 A~G, n은 0~7). 만약 포트 B의 비트3에 1을 출력하려면 PORTB.3 = 1로 하면 된다.

3.2.1.3. PINx 레지스터

입력으로 설정된 경우 PINx 레지스터에 해당하는 값을 읽으면 된다. 해당 핀의 값을 읽어 들인다. 쓰기가 금지되어 있다. 비트별로 읽어 들이려면 PINx.n를 사용 하면 된다 비트별로 만약 포트 C의 비트3값을 읽어 들여 led3라는 비트값에 할당하려면 led3 = PINC.3로 하면 된다.

3.2.1.4. SFIOR 레지스터

SFIOR(Special Function IO Register)의 비트2(PUD: Pull-Up Disable)를 1로 세트하면 풀업 저항을 비활성화 시킨다.

3.2.2. 구조 및 동작

- 기본 구조 : 그림 78에 보여진 것과 같이 양방향성으로 20K~100K옴의 내부 풀업을 갖는 구조를 갖고 있다.
- 내부풀업저항의 설정 : SFIOR의 비트 2를 이용하여 모든 입출력핀의 풀업저항을 비활성화할 수 있다. DDRx.n의 값을 조절하여 입력으로 설정된 경우에는 해당하는 PORTx.n에 1을 쓰면 내부 풀업 저항을 사용할 수 있다. 내부 풀업 저항을 사용하지 않기 위해서는 PORTx.n에 0을 쓰거나 해당 핀을 출력으로 설정하면 된다.
- 읽어 들이는 동작 : PINx.n레지스터 비트를 읽을 때 안정을 위하여 최대 1.5클럭에서 0.5클럭의 지연이 존재하므로 이를 감안해서 읽어야 한다. 결국 PORTx.n 비트에 출력한 값을 다시 읽어 들일 때는 NOP(1클럭이 소요됨)와 같은 명령을 삽입하는 것이 좋다.
- 슬립 모드시 핀의 동작 : 슬립모드가 동작하면 포트의 핀 입력은 접지로 클램프되어 동작이 멈춘다. 이때 사용하지 않는 핀은 외부에서 풀업 또는 풀다운으로 확실한 논리 상태를 입력하는 것이 전력소비를 감소시킨다. 손쉬운 방법으로 내부의 풀업을 이용하는 것이다.
- 포트A (PA7~PA0:핀44-51) : 내부 풀업 저항이 있는 8비트 양방향 입출력 단자. 외부메모리를 둘 경우에는 주소버스(A7-A0)와 데이터버스(D7-D0)로 사용.
- 포트B (PB7~PB0:핀10-17) : 내부 풀업 저항이 있는 8비트 양방향 입출력 단자. SPI용 단자 혹은 PWM 단자

로도 사용된다(표2 참조).

- 포트C (PC7~PC0:핀35-42) : 내부 풀업 저항이 있는 8비트 양방향 입출력 단자. 외부메모리를 둘 경우에는 주소버스(A15-A8)로 사용된다.
- 포트D (PD7~PD0:핀25-32) : 내부 풀업 저항이 있는 8비트 양방향 입출력 단자. 타이머용 단자 혹은 외부인터럽트용 단자로도 사용된다(표3 참조).
- 포트E (PE7~PE0:핀2-9) : 내부 풀업 저항이 있는 8비트 양방향 입출력 단자. 타이머용 단자, 외부인터럽트, 아날로그 비교기, USART용 단자로도 사용된다(표4 참조).
- 포트F (PF7~PF0:핀54-61) : 내부 풀업 저항이 있는 5비트 양방향 입출력 단자. AD변환기 혹은 JTAG 인터페이스용 단자로도 사용된다(표5참조).
- 포트G (PG4~PE0:핀19, 18, 43, 34, 33) : 내부 풀업 저항이 있는 8비트 양방향 입출력 단자. 외부 메모리 접속을 위한 스트로브 신호용, RTC(Real Time Counter) 타이머용 발진기 단자로도 사용된다(표5참조).

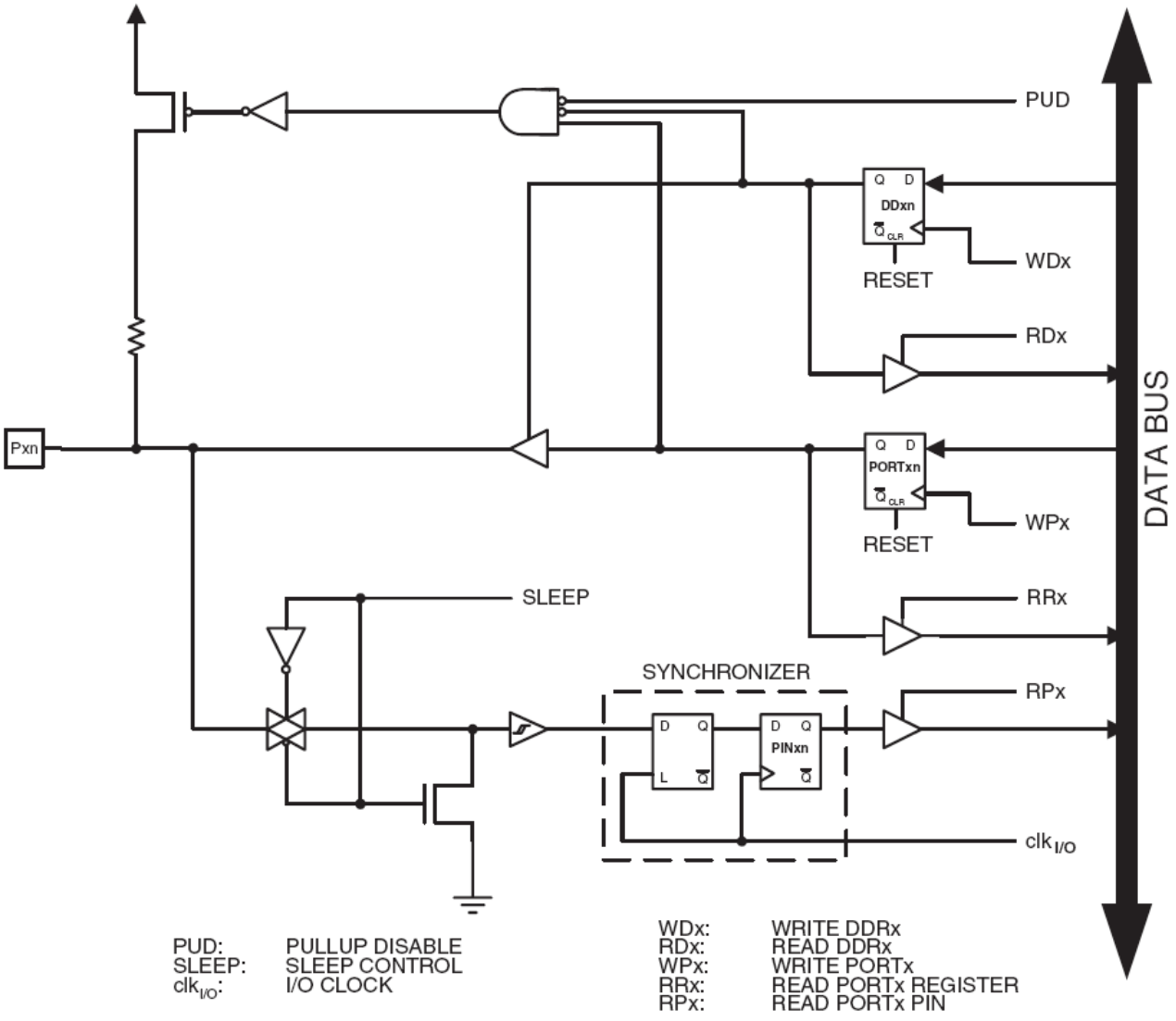


그림 78. I/O 포트의 기본 구조(출처:ATMEL)

3.3. 단순 출력 연습

3.3.1. 실험12 : LED 점멸1

- 예제 : 포트C에 연결된 8개의 LED를 0.4초 간격으로 좌로 이동하며 한 개씩 켠다. 즉 처음 PORTC.0에 연결된 LED만 켜고 0.4초 후에 PORTC.1에 연결된 LED만 켜고 이런 식으로 반복적으로 8개의 LED를 순차적으로 켠다.

- 프로그램 작성시 유의점 : 0.4초의 시간지연은 실험10의 delay루틴을 사용한다. 그림 79의 회로에서 0을 출력해야 LED가 켜지고 1을 출력하면 꺼진다. mega128.h를 include시켜야 PORTC.n을 사용할 수 있다. delay_ms()를 사용하려면 delay.h를 include시켜야 한다.

- 회로도 : 그림 79의 시스템을 구성한다. LED는 화합물 반도체의 PN접합에 빛이 투과하도록 P형층을 매우 얇게 만든 소자로 P형(애노드:양극)에 (+)전위 N형(캐소드:음극)에 (-)전위를 가하면 즉 순방향을 가하면 전류가 흐르면서 전계발광현상에 의해 발광한다. 대략 2 V와 20 mA 정도의 직류전압 전류에서 동작한다. 기본적으로 전압구동소자가 아니라 전류구동소자이며 발광하는 광의 세기는 전류에 비례하며 각종 표시장치에 응용되는 소자이다. LED의 단자중 긴 것이 애노드(양극)이다. 실험할 때는 LEDSW를 닫고 한다.

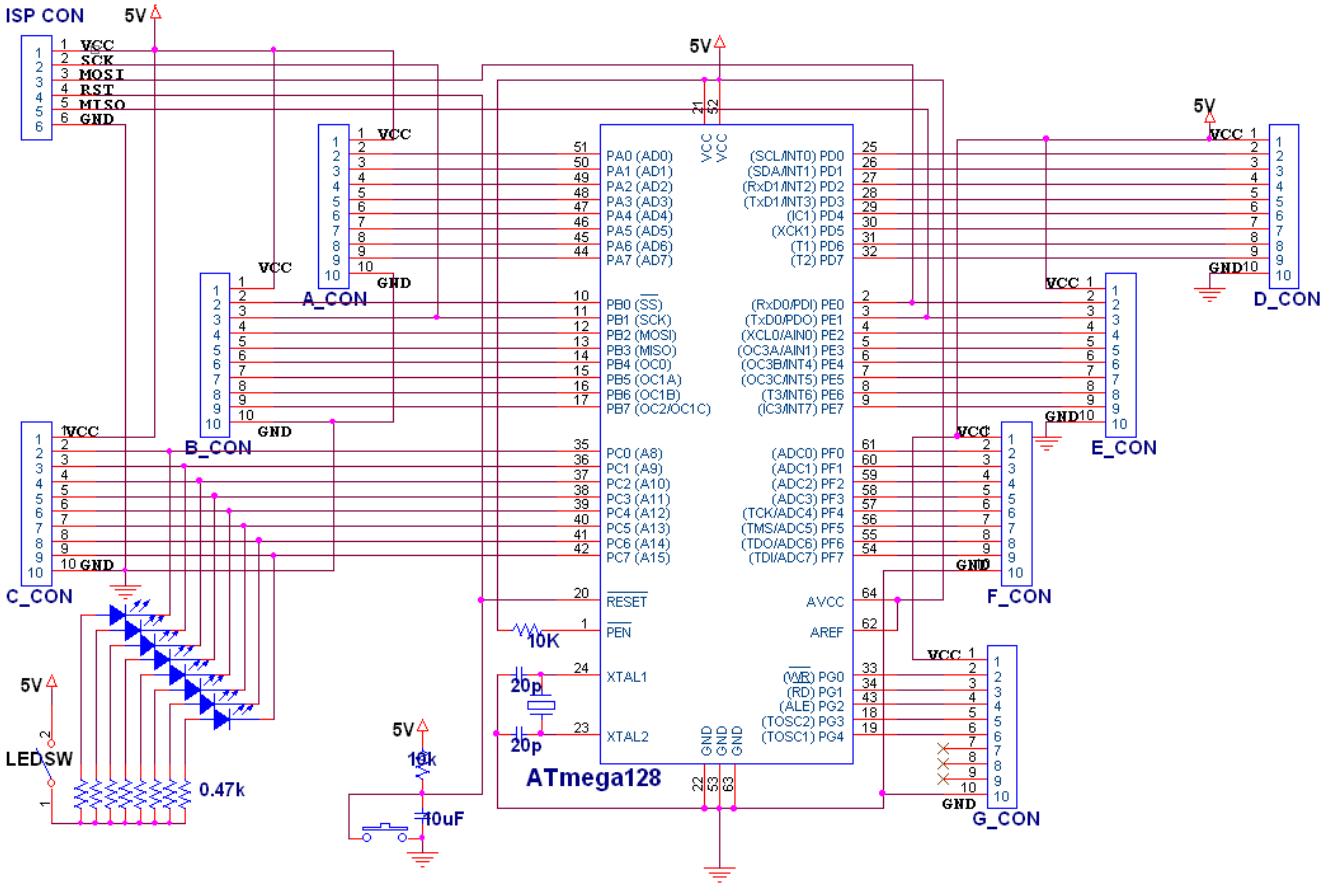


그림 79. 기본 입출력 실험을 위한 회로도

- 프로그램 예 :

```
#include <mega128.h>
void delay(unsigned char i){        // 시간 지연 루틴 약 0.875~112usec 시간지연 얻을 수 있다
      while(i--); }                 // delay(226)은 대략 0.1msec의 시간지연을 얻을 수 있다.
void delayms(unsigned int i){       // i msec 시간 지연을 얻을 수 있다.
      unsigned int j,k;
      for(j=0;j<i;j++) for(k=0;k<10;k++)   delay(226); }
void main(void){
unsigned char   led=0xfe;
DDRC = 0xff;
      for(;;){
      if (led != 0x7f) led = (led << 1 ) | 0x01; /* 비트7를 켜지 않았으면 1비트씩 좌로 시프트하고 빈자리에 1을 */
      else led = 0xfe;                          /* 비트7을 켰으면 다시 처음으로 와서 비트0를 켠다. */
```

```
        delayms(400);       /* 0.4초 시간 지연 , 시뮬레이션 할 때는 제거하고 해도 됨*/
        PORTC = led; }
    }
```

- 실험방법 : ①CodeVisionAVR의 에디터를 통해 실행파일을 만든 후 AVR Studion4디버그세션에 들어간 후 메인메뉴의 [View->Memory]를 실행하여 Memory라는 이름의 창을 생성하고 창에서 [Data]을 선택한다. 메인메뉴의 [View->Memory1]를 실행하여 Memory1라는 이름의 창을 생성하고 창에서 [Program]을 선택한다. 또한 [View->Register]를 실행하여 Register라는 이름의 창을 생성한다. ②디버그세션에서 메인메뉴 [Debug]의 하위 메뉴인 [Step Into], [Auto Step], [Run to Cusor] 등을 적절히 활용하면서 Memory, Register, I/O View 창을 통해 내부램과 레지스터의 값의 변화를 확인하라. 시뮬레이션할 때 delay(), delayms()는 시간을 많이 걸리게 하므로 제거하고 해도 된다. ③시뮬레이션이 성공적이었으면 실행파일을 만들고 그림 51의 인터페이스회로와 그림 79의 ISP CON을 연결하고 79의 전원을 넣은 상태에서 2.5.5절에 나온 방법을 참조하여 다운로드하라. ④ 시스템이 성공적으로 동작하는지 확인하라.
- 과제 : ① 0.3초마다 우로 이동하며 점멸동작하도록 해보라. ② LED가 2진 가산형태로 0.4초마다 점멸동작하도록 해보라.

3.3.2. 실험13 : LED 점멸2

- 예제 : 포트C에 연결된 8개의 LED를 0.04초 간격으로 좌로 이동하며 한 개씩 켜고 PORTC.7 해당하는 LED까지 켰으면 우로 이동하며 한 개씩 켠다. 이것을 반복적으로 수행하여 점멸이 좌우로 흔들리도록 한다.
- 프로그램 작성시 유의점 : 그림 79의 회로에서 0을 출력해야 LED가 켜지고 1을 출력하면 꺼진다.
- 회로도 : 그림 79의 시스템을 사용한다. 실험할 때는 LEDSW를 닫고 한다.
- 프로그램 예 :

```
#include <mega128.h>
#include <delay.h>      // delay_ms(), delay_us()를 이용할 것이다.
void main(void){
unsigned char  led=0xfe, i;
DDRC = 0xff;
    for(;;){
    for(i=0;i<=6;i++) { led = (led << 1 ) | 0x01; /* 비트7를 켜지 않았으면 1비트씩 좌로 시프트하고 빈자리에 1을 */
    delay_ms(40);       // 시뮬레이션 할 때는 제거하고 해도 됨
    PORTC = led; }
    for(i=0;i<=6;i++) { led = (led >> 1 ) | 0x80; /* 비트7을 켰으면 1비트씩 우로 시프트하고 빈자리에 1을 */
    delay_ms(40);       // 시뮬레이션 할 때는 제거하고 해도 됨
    PORTC = led; } }
}
```

- 실험방법 : 실험11을 참조해서 하라. 시뮬레이션할 때 delay_ms()는 시간을 많이 걸리게 하므로 제거하고 해도 된다.
- 과제 : 0.04초 간격 이외의 0.01초나 0.1초등 다양한 시간간격으로 똑같은 동작을 구현해보라.

3.3.3. 실험14 : LED 점멸3

- 예제 : PORTC포트에 연결된 8개의 LED를 롬 데이터에 따라 점등패턴과 지속시간을 결정하여 점등한다. 롬 데이터를 끝까지 다 읽어 점등했으면 다시 처음의 데이터에 따라 점등한다. 이를 무한히 반복한다.
- 프로그램 작성시 유의점 : 지속시간은 롬데이터에 주어진 값에 100을 곱한 값으로 하며 롬데이터의 끝까지 수행한 경우 다시 처음의 데이터를 읽어 반복 수행한다.
- 회로도 : 그림 79의 시스템을 사용한다. 실험할 때는 LEDSW를 닫고 한다.
- 프로그램 예 :

```
#include <mega128.h>
void delay(unsigned char i){        // 시간 지연 루틴 약 0.875~112usec 시간지연 얻을 수 있다
    while(i--); }                   // delay(226)은 0.1msec의 시간지연을 얻을 수 있다.
void delayms(unsigned int i){       // i msec 시간 지연을 얻을 수 있다.
    unsigned int j,k;
    for(j=0;j<i;j++) for(k=0;k<10;k++)  delay(226); }
flash unsigned char  ledp[ ] = { 0xfe, 0xef, 0x44, 0x11, 0x23, 0x34, 0x35, 0x67};
flash unsigned char  ledt[ ] = { 0x22, 0x33, 0x44, 0x45, 0x56, 0x66, 0x78, 0x66};
void main(void){
unsigned char i;
    DDRC = 0xff;
    for(;;){
        for(i=0;i<=8;i++){
            PORTC = ledp[i];
            delayms((unsigned int)(ledt[i]*100));
        }
    }
}
```

- 과제 : 다양한 시간간격과 패턴으로 점멸하는 동작을 구현해보라.

3.3.4. 실험15 : LED 점멸4

- 예제 : C포트에 연결된 8개의 LED를 stdlib.h에 있는 rand()함수를 이용하여 패턴과 지속시간을 결정하여 점등한다.
- 프로그램 작성시 유의점 : rand()를 사용하여려면 #include <stdlib.h>를 프로그램에 포함시켜야 한다. 그리고 rand()는 0~32767까지의 숫자를 만들어낸다.
- 회로도 : 그림 79의 시스템을 사용한다. 실험할 때는 LEDSW를 닫고 한다.
- 프로그램 예 :

```
#include <mega128.h>
#include <stdlib.h>       // rand()를 이용할 것이다.
void delay(unsigned char i){        // 시간 지연 루틴 약 0.875~112usec 시간지연 얻을 수 있다
    while(i--); }                   // delay(226)은 0.1msec의 시간지연을 얻을 수 있다.
void delayms(unsigned int i){       // i msec 시간 지연을 얻을 수 있다.
    unsigned int j,k;
    for(j=0;j<i;j++) for(k=0;k<10;k++)  delay(226); }
void main(void){
    DDRC = 0xff;
    for(;;){ PORTC = (char)(rand() % 0x80);
        delayms((unsigned int)(2*rand())); }
}
```

- 과제 : delay.h를 include하여 delay_ms(), delay_us()로 시간지연하는 프로그램으로 수정해보라.

3.4. 단순 입출력 연습

3.4.1. 실험16 : 키 누름수 세기

- 예제 : PD0에 연결된 누름키가 눌린 수를 세어 C포트에 연결된 8개의 LED로 표시한다.
- 회로도 : 그림 80의 SW_CON을 그림 79의 D_CON에 연결하고 실험할 때는 LEDSW를 닫고 한다. 또한 그림 80의 SW1을 누르면서 실험한다.
- 프로그램 작성시 유의점 : 기계식 키를 누르면 탄성에 의해(키를 누르면 0이고 안 누른 상태를 1이라 가정했을 때) 0상태에 안정적으로 도달하기 전에 0과1을 여러번 반복하는 바운스(Bounce) 또는 채터링(Chattering)이

일어난다. 바운스현상은 수십 msec 동안 일어나 사람은 단지 1번 누른 것처럼 느끼지만 마이컴은 여러번 눌린 것처럼 오인하게된다. 키를 눌렀다 뗄 때도 마찬가지로 바운스 현상이 일어난다. 키에 2개의 NAND 게이트를 사용하여 하드웨어적으로 바운스현상을 줄일 수도 있지만 소프트웨어에 의해 할 수 있다. 여러 가지 소프트웨어에 의한 바운스 처리법이 있지만 본 실험에서는 키가 눌리지 않은 상태(1레벨)에서 시작하여 키가 눌려짐(0레벨)을 검출하고 적절한 시간지연 후에(보통 20-50 msec) 또 다시 눌려짐(0레벨)을 검출하는 즉 두 번 확인하는 방식을 택한다. *그림 80의 회로에서 풀업 저항 10K옴 대신 무한대의 저항 즉 풀업 저항 10K옴을 없애고 개방하여 입력장치로 사용해도 된다. 이것은 I/O포트들이 입력으로 사용될 때 내부 풀업 저항을 사용할 수 있기 때문이다. 이런 경우 해당 포트 핀을 입력으로 설정하기 위해 DDRx.n=0 명령에 부가적으로 PORTx.n=1 명령으로 해당 포트 핀에 1을 써 넣어야 내부 풀업 저항을 사용할 수 있다.*

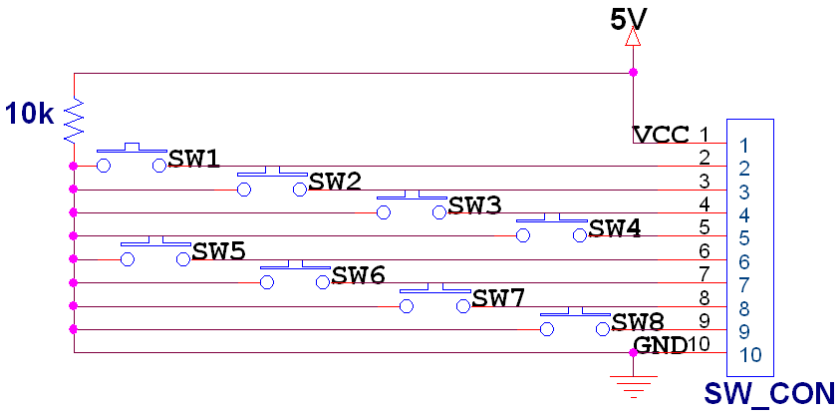


그림 80. 단순 키보드 스캔을 위한 회로도

● 프로그램 예 :

```
#include <mega128.h>
#include <delay.h>        // delay_ms(), delay_us()를 이용할 것이다.
void main(void){
unsigned char key_num = 0;
      DDRC = 0xff; // 포트C를 출력으로
      DDRD = 0x00; //포트D를 입력으로 디폴트 입력으로 설정되어 있으므로 필요하지는 않음
      for(;;) {

                  while( PIND.0 == 1 );          /* 눌릴 때(PIND.0 ==0)까지 대기 눌리면 아래로 */
                  delay_ms(40);                  /* 약 40msec 시간지연하고 다시 읽기*/
                  if( PIND.0 == 0){              /* 40msec 후에도 눌려있으면 유효한 키입력이므로 아래로 */
                  key_num += 1;
                  PORTC = ~key_num; }            /* 1일 때 꺼지고 0일때 켜지므로 반전시켜 출력시킨다. */
                 while(PIND.0 == 0); }           /* 키 눌림이 해제될 때까지 대기 */
}
```

● 실험방법 : ①CodeVisionAVR의 에디터를 통해 실행파일을 만든 후 AVR Studion4디버그세션에 들어간 후 메인메뉴의 [View->Memory]를 실행하여 Memory라는 이름의 창을 생성하고 창에서 [Data]을 선택한다. 메인메뉴의 [View->Memory1]를 실행하여 Memory1라는 이름의 창을 생성하고 창에서 [Program]을 선택한다. 또한 [View->Register]를 실행하여 Register라는 이름의 창을 생성한다. ②디버그세션에서 메인메뉴 [Debug]의 하위 메뉴인 [Step Into], [Auto Step], [Run to Cusor] 등을 적절히 활용하면서 Memory, Register, I/O View 창을 통해 내부램과 레지스터의 값의 변화를 확인하라. 시뮬레이션할 때 delay_ms()는 시간을 많이 걸리게 하므로 제거하고 해도 된다. [Auto Step]를 누르고 IO View창의 PIND레지스터 비트0의 값을 변화시켜가며 PORTC의 변화를 관찰하라. PIND.0을 변화시킬 때마다 PORTC의 값이 증가해야 한다. ③시뮬레이션이 성공적이었으면 실행파일을 ISP소프트웨어와 인터페이스회로를 사용하여 그림 79의 시스템에 다운로드하라. ④시스템이 성공적으로 동작하는지 확인하라.

● 과제 : 시간 지연을 다르게 하여 수행해보라(수 msec 단위로 작게 수백 msec 단위로 크게).

3.4.2. 실험17 : 파일럿 램프1

● 예제 : 딥스위치가 연결된 PORTD.0~3로부터 입력을 받아 그에 대응하는 PORTD.4~7에 연결된 LED를 켠다.

● 프로그램 작성시 유의점 : LED가 켜지는 것이 0상태이고 꺼지는 것이 1상태이다. 그러므로 그림 81의 회로에서 딥스위치를 조작했을 때 그에 연결된 LED가 켜졌다면 입력으로 0상태가 들어간 것이다.

● 회로도 : 그림 81에 주어진 회로의 L1을 그림 79의 D_CON을 연결하고, 그림 81의 딥스위치의 5~8번을 개방한 상태로 한다. 그러면 PORTD.0~3을 통해 LED를 점멸할 수 있는 상태가 된다. 그림 81의 딥스위치의 1~4번을 열고 닫아 실험하면 된다.

● 프로그램 예 :

```
#include <mega128.h>
void main(void){
unsigned        char        ledp;
        DDRD = 0xf0;                        // PORTD.4~7을 출력으로 나머지를 입력으로 설정
        for(;;) {
                ledp = PIND;
                ledp = (ledp << 4) & 0xf0 ; // 비트4~7에 연결된 LED에 출력하기 위해 4번 시프트한다.
                PORTD = ~ledp;              // 반전시켜 출력하라.
        }
}
```

● 실험방법 : 실험 16을 참조해서 수행하라.

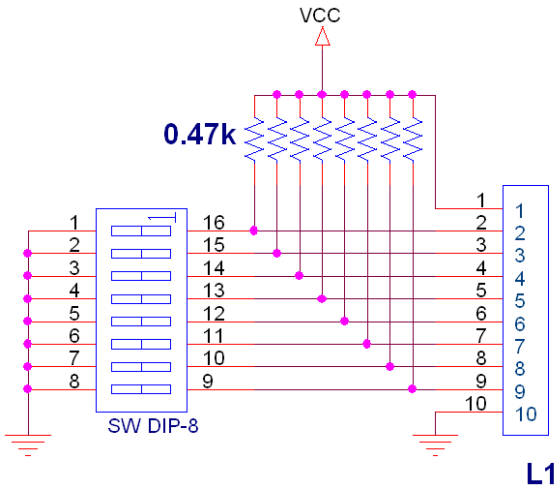


그림 81. 딥스위치를 이용한 간단한 입력회로

3.4.3. 실험18 : 파일럿 램프2

● 예제 : 딥스위치가 연결된 포트D로부터 입력을 받아 그에 대응하는 포트A에 연결된 LED를 켠다.

● 회로도 : 그림 81에 주어진 회로의 L1을 그림 79의 D_CON을 연결한다. 그림 79의 LEDSW를 닫는다. 그림 81의 딥스위치의 1~8번을 열고 닫아 실험하면 된다.

● 프로그램 예 :

```
#include <mega128.h>
void main(void){
        DDRD = 0x00;                        // PORTD를  입력으로 설정
        DDRA = 0xff;                        //포트A를 출력으로 설정
        for(;;) {
                PORTA = ~PIND;              // 반전시켜 출력하라.
```

```
    }
}
```

3.4.4. 실험19 : 입출력함수 구현

● 예제 : 딥스위치가 연결된 포트D 비트0~2로부터 입력을 받아 미리 롬에 기록된 데이타 값에 따라 포트D 3~7에 연결된 LED를 켜고 끈다. 본 실험에서 구현된 입출력 관계는 다음과 같다.

입출력 관계: 000->01101, 001->10010, 010->00010, 011->10001,

100->11100, 101->10101, 110->00100, 111->10100

● 회로도 : 그림 81에 주어진 회로의 L1을 그림 79의 D_CON을 연결하고, 그림 81의 딥스위치의 4~8번을 개방한 상태로 한다. 그러면 PORTD.3~7를 통해 LED를 점멸할 수 있는 상태가 된다. 그림 81의 딥스위치의 1~3번을 열고 닫아 실험하면 된다.

● 프로그램 작성시 유의점 : LED가 켜지는 것이 0상태이고 꺼지는 것이 1상태이다. 입력이 000일 때 01101을 출력하는 것으로 했는데 롬에 이 패턴을 기록해 둘 때는 01101xxx(여기에서 x는 0 또는 1)이 되어야 한다. 본 실험에서는 x=0으로 하였다. 즉 01101의 패턴은 0x68로 기록되고 10010의 패턴은 0x90 식으로 한다.

● 프로그램 예 :

```
#include <mega128.h>
unsigned char arr[ ]={ 0x68, 0x90, 0x10, 0x88, 0xE0, 0xA8, 0x20, 0xA0};
void main( void ) {
unsigned        char        ledp;
      DDRD = 0b11111000; //비트0~2는 입력, 나머지는 출력
      for(;;) {
      ledp = PIND & 0b00000111 ;              /* 하위 3비트만 마스킹 */
      PORTD = arr[ledp] ;            // 반전시켜 출력한다.
      }
}
```

● 예제 : 다른 형태의 입출력 함수를 구현해 보아라.

3.4.5. 실험20 : 입력값에 따른 LED 점멸속도 조절

● 예제 : 딥스위치가 연결된 포트D로부터 입력을 받아 포트C에 연결된 LED의 점멸속도를 조절한다.

● 회로도 : 그림 81에 주어진 회로의 L1에 그림79의 D_CON을 연결한다. 그림 81의 딥스위치를 열고 닫아 그 값을 입력으로 삼아 실험하면 된다. 실험할 때는 그림79의 LEDSW를 닫고 한다.

● 프로그램 예 :

```
#include <mega128.h>
#include <delay.h>
void main(void){
unsigned        char        led=0xfe;
unsigned        int         dt;
      DDRC = 0xff;        //Port C를 출력으로 설정.
      DDRD = 0x00;        //포트 D를 입력으로 설정
      for(;;){
      for(dt=0; dt < (int) 256*PIND ;dt++) delay_ms(1);  // 시간지연 값을 입력받아 시간 지연한다.
      if (led != 0x7f) led = (led << 1 ) | 0x01;    // 비트7를 켜지 않았으면 1비트씩 좌로 시프트하고 빈자리에 1을
      else led = 0xfe;                              /* 비트7을 켰으면 다시 처음으로 와서 비트0를 켠다. */
      PORTC = led; }
}
```

4. 인터럽트와 타이머 실험

4.1. 외부 인터럽트 실험

4.1.1. 인터럽트의 개념과 종류

- 인터럽트는 어떤 조건이나 사건의 발생으로 정상적인 프로그램을 일시적으로 중지시키고 보다 시급한 작업을 먼저 수행하고 다시 원래로 복귀하여 실행순서를 변경하도록 만드는 것을 말한다. 인터럽트 요청에 따라 수행하는 프로그램을 인터럽트 서비스루틴 또는 인터럽트 핸들러라 한다. 인터럽트는 마이컴시스템을 구현하는데 커다란 역할을 차지하며 발생시기를 예측하기 힘든 사건에 마이컴이 가장 빠르게 대응할 수 있는 방법이며 비교적 저속으로 동작하는 주변장치의 요청에 고속으로 동작하는 마이컴 사이에 효율적으로 일을 수행하는 중요한 수단이 된다.
- 인터럽트의 3요소 : ① 발생원 :누가 요청했는가? ② 우선순위 : 2개 이상의 요청시 누구를 먼저 서비스 할까? ③ 인터럽트벡터 : 서비스루틴의 시작번지는 어디인가?
- 인터럽트의 종류 : 발생원인이 존재하는 곳에 따라 내부와 외부로 나뉘고, 차단가능성에 따라 차단가능 인터럽트와 차단 불가능 인터럽트로 나뉘고, 인터럽트발생 원인을 확인하는 방법에 따라 조사형과 벡터형으로 나뉜다.
- 내부 인터럽트는 영으로 나누는 등 금지된 연산이나 금지된 메모리로의 접근 시도 등에 따라 마이컴 내부에서 발생하는 인터럽트며 외부 인터럽트는 입출력장치의 작업종료, 타이머의 오버플로, 외부장치의 통신요청 등 외부장치에 의해 발생되는 인터럽트로 일반적으로 인터럽트하면 외부인터럽트를 지칭한다. ATmega128은 8051처럼 내부 인터럽트에 해당하는 것이 없다.
- 프로그램에 의해 인터럽트서비스 요청을 차단하지 못하는 것을 차단불가능 인터럽트라하며 전원이상 등의 매우 중요한 돌발 사태에 대비하기 위해 대부분의 마이크로프로세서에서 사용되지만 ATmega128는 8051처럼 차단불가능 인터럽트를 사용하지 않는다. 차단가능한 인터럽트를 차단하는 방법은 특정레지스터값의 설정에 따라 한다.
- 인터럽트를 요청한 장치가 특정한 상태비트를 세트해 표시해 놓으면 이를 마이컴이 소프트웨어적으로 조사하고 찾는 순서에 따라 우선순위가 정해지는 방식을 조사형 인터럽트라 하고 인터럽트 발생원인에 따라 미리 지정된 서비스루틴의 시작번지(인터럽트 벡터)에서 처리하는 방식을 벡터형이라 하는데 벡터형이 인터럽트 발생원인을 찾고 서비스루틴을 처리하는 시간이 빠른 장점이 있다. ATmega128은 벡터형을 사용한다.
- ATmega128의 인터럽트는 차단가능한 외부인터럽트에 해당하며 리셋을 포함하여 총 35개의 인터럽트 벡터를 가지고 있다. 리셋을 제외한 34개의 인터럽트는 외부 핀을 통하여 요청되는 외부인터럽트 8개, 타이머0 관련 2개, 타이머1 관련 5개, 타이머2 관련 2개, 타이머3 관련 5개, USART0 관련 3개, USART1 관련 3개, 기타 6개로 분류될 수 있다.

4.1.2. ATmega128의 인터럽트 처리과정

- **1단계** : 매 사이클의 인터럽트 요청이 있는지 확인하고 해당 인터럽트 플랙에 기록한다.
- **2단계** : 어느 인터럽트 요청이 있는지 인터럽트 플랙을 조사하고 우선순위와 허용여부를 결정한다.
- **3단계** : 인터럽트 벡터주소를 찾아가기 위한 CALL 명령이 수행되며 다른 인터럽트 발생을 방지하기 위해 SREG의 I비트를 클리어시키고 복귀주소(PC)값을 스택에 저장한다.
- **4단계** : 인터럽트벡터에 따라 인터럽트 서비스 루틴으로 점프하여 실행한다.
- **5단계** : RETI를 만나면 스택에 저장된 PC를 복구하여 원래 프로그램으로 복귀한다.

4.1.3. 인터럽트 처리 시간

● 응답시간 : 인터럽트를 허용하는 SEI 명령이 실행되었더라도 최소한 그 다음에 있는 1개의 명령이 실행되고 인터럽트가 수행된다. 인터럽트 서비스를 끝내고 원래의 프로그램으로 복귀한 경우에도 이미 대기 상태에 있던 다른 인터럽트를 수행하기 전에 최소한 1개의 명령이 실행된다. 또한 인터럽트 벡터는 3클럭이 요구되는 JMP 명령으로 이루어져있어 1개의 명령이 최소 1클럭이므로 인터럽트 요청 후 서비스 시작하기까지 최소 4클럭이 걸린다. 이 동안에 다른 인터럽트 발생을 방지하기 위해 SREG의 I비트를 클리어시키고 PC를 스택에 저장하고 인터럽트 벡터주소를 찾아가서 거기에 있는 값에 따라 인터럽트 서비스 루틴으로 점프한다. 만약 2클럭 이상의 명령을 수행하고 있었으면 응답시간은 증가한다. 슬립모드에 있었을 때는 기동시간이 포함되어 응답시간이 증가된다.

● 복귀시간 : RETI는 4클럭이 소요된다. 즉 인터럽트 서비스를 마치고 원래 프로그램으로 복귀하는 데에는 4클럭이 소요된다. 이 시간동안 PC의 값이 스택에서 꺼내지고 스택 포인터가 2만큼 증가되며 SREG의 I비트가 1로 세트된다.

4.1.4. 인터럽트 제어

4.1.4.1. 벡터 배치

Fuse High Byte의 BOOTRST 비트값과 MCUCR 레지스터의 IVSEL 비트값으로 벡터의 배치를 그림 82처럼 바꿀 수 있다. 그림 82에서 Boot Reset Address는 그림 12에 주어졌다.

BOOTRST	IVSEL	Reset Address	Interrupt Vectors Start Address
1	0	$0000	$0002
1	1	$0000	Boot Reset Address + $0002
0	0	Boot Reset Address	$0002
0	1	Boot Reset Address	Boot Reset Address + $0002

그림 82. 인터럽트 벡터의 배치(출처:ATMEL)

그림 83은 Fuse High Byte의 BOOTRST 비트값이 1로 MCUCR 레지스터의 IVSEL은 0으로 즉 디폴트로 설정된 경우의 인터럽트 벡터를 보여준다.

4.1.4.2. MCUCR 레지스터

● 비트1(IVSEL:Interrupt Vector SELect) : 그림 82처럼 인터럽트 벡터의 배치를 변경할 수 있다. IVSEL이 변경되 인터럽트 벡터의 예기치 않게 이동되는 것을 방지하기 위해 IVSEL의 변경은 다음의 절차에 따른다. ① IVCE비트에 1을 쓴다. ② 4사이클 이내에 IVCE =0 으로 설정하고 IVSEL에 원하는 비트값으로 설정한다.

● 비트0(IVCE:Interrupt Vector Change Enable) : 1로 세트하면 IVSEL을 변경할 수 있다. IVCE비트는 IVSEL 비트를 변경하고 난 이후에 4클럭이 지나면 하드웨어의 의해서 0으로 자동적으로 클리어된다.

4.1.4.3. 허용과 우선순위

● 전체적인 허용은 상태레지스터(SREG)의 비트7(I)를 1로 하여 설정한다.

● 개별적인 허용은 여러가지 인터럽트 마스크 레지스터를 통하여 개별적으로 허용여부를 제어한다.

● ATmega128은 8051처럼 인터럽트의 우선순위를 줄 수 없고 먼저 들어온 것을 먼저 처리한다. 동시에 들어오는 경우에는 하드웨어에 의해 순위가 정해져 있는 데 그림 83에 벡터표 순서대로 순위가 고정되어 있다.

4.1.4.4. 외부 인터럽트 트리거

● 입력 : 외부 인터럽트 핀 INT0~3(핀25~28, PD0~3), INT4~7(핀6~9, PE4~7)을 통해 입력되는 신호로 발생하며 입출력포트 핀이 출력으로 설정되더라도 발생시킬 수 있다. 소프트웨어적으로 해당 핀에 값을 출력시켜 인

터럽트를 발생 시킬 수 있다.

Vector No.	Program Address	Source	Interrupt Definition
1	$0000	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	INT3	External Interrupt Request 3
6	$000A	INT4	External Interrupt Request 4
7	$000C	INT5	External Interrupt Request 5
8	$000E	INT6	External Interrupt Request 6
9	$0010	INT7	External Interrupt Request 7
10	$0012	TIMER2 COMP	Timer/Counter2 Compare Match
11	$0014	TIMER2 OVF	Timer/Counter2 Overflow
12	$0016	TIMER1 CAPT	Timer/Counter1 Capture Event
13	$0018	TIMER1 COMPA	Timer/Counter1 Compare Match A
14	$001A	TIMER1 COMPB	Timer/Counter1 Compare Match B
15	$001C	TIMER1 OVF	Timer/Counter1 Overflow
16	$001E	TIMER0 COMP	Timer/Counter0 Compare Match
17	$0020	TIMER0 OVF	Timer/Counter0 Overflow
18	$0022	SPI, STC	SPI Serial Transfer Complete
19	$0024	USART0, RX	USART0, Rx Complete
20	$0026	USART0, UDRE	USART0 Data Register Empty
21	$0028	USART0, TX	USART0, Tx Complete
22	$002A	ADC	ADC Conversion Complete
23	$002C	EE READY	EEPROM Ready
24	$002E	ANALOG COMP	Analog Comparator
25	$0030	TIMER1 COMPC	Timer/Countre1 Compare Match C
26	$0032	TIMER3 CAPT	Timer/Counter3 Capture Event
27	$0034	TIMER3 COMPA	Timer/Counter3 Compare Match A
28	$0036	TIMER3 COMPB	Timer/Counter3 Compare Match B
29	$0038	TIMER3 COMPC	Timer/Counter3 Compare Match C
30	$003A	TIMER3 OVF	Timer/Counter3 Overflow
31	$003C	USART1, RX	USART1, Rx Complete
32	$003E	USART1, UDRE	USART1 Data Register Empty
33	$0040	USART1, TX	USART1, Tx Complete
34	$0042	TWI	Two-wire Serial Interface
35	$0044	SPM READY	Store Program Memory Ready

그림 83. ATmega128의 인터럽트 발생원과 벡터값(출처:ATMEL)

● 트리거링 : 논리 0의 레벨, 하강에지, 상승 에지값을 해당 핀에 주어 인터럽트를 발생시킬 수 있다. 이들은 EICRA(INT0~3용), EICRB(INT4~7용)으로 설정한다. 에지 트리거는 50ns 이상의 펄스폭을 가져야 한다. 레벨

트리거 인터럽트 신호는 워치독 오실레이터에 의해 2번 샘플링되며 이 기간이상의 펄스폭을 주어야한다.

- 허용 제어 : SREG의 I비트를 사용하여 전체적인 허용을 제어하고 EICRA, EICRB를 이용하여 트리거링을 설정하고 EIMSK를 이용하여 개별적으로 허용하여 발생을 제어한다. 개별적으로 허용되어 있더라도 SREG의 I비트가 0으로 되어 있으면 인터럽트는 발생되지 않는다. 개별적인 허용과 전체적인 허용이 있고 해당핀에 적절한 신호가 들어와야 인터럽트는 발생될 수 있다.
- 슬립모드의 해제 : 레벨 트리거로 설정한 경우의 INT0~7과 하강 또는 상승 에지 트리거 방식으로 설정된 INT0~3은 인터럽트가 클럭에 관계없이 비동기로 검출되어 슬립모드를 해제하는 데 사용할 수 있다. 하강 또는 상승 에지 트리거 방식으로 설정된 INT4~7의 경우는 I/O클럭을 필요로 하여 이 클럭이 차단되는 슬립모드들(아이들 모드를 제외한 모든 슬립모드)에서는 슬립모드를 해제하는 데 사용할 수 없다.

4.1.4.5. EICRA 레지스터

EICRA(External Interrupt Control Register A) 레지스터는 초기값은 00으로 외부 인터럽트 0~3의 트리거 설정에 사용된다.

- 비트7~0(ISCn1, ISCn0) : 여기에서 n은 3~0의 값을 갖는다. 값이 ISCn1:ISCn0 = 11일 때 INTn을 상승에지에서, 10일때는 하강에지에서, 00일때는 0 레벨에서 트리거되도록 설정한다. 01은 보류되어 있다.

4.1.4.6. EICRB 레지스터

- 비트7~0(ISCn1, ISCn0) : 여기에서 n은 7~4의 값을 갖는다. 값이 ISCn1:ISCn0 = 11일 때 INTn을 상승에지에서, 10일때는 하강에지에서, 01일때는 상승에지 또는 하강에지에서, 00일때는 0 레벨에서 트리거되도록 설정한다.

4.1.4.7. EIMSK 레지스터

EIMSK(External Interrupt MaSK register)레지스터는 INT7~0의 개별적인 허용 제어에 사용된다.

- 비트7~0(INT7~0) : 초기값은 00으로 1로 설정되어 있으면 해당 인터럽트가 허용된다.

4.1.4.8. EIFR 레지스터

EIFR(External Interrupt Frag Register)레지스터는 외부 인터럽트가 에지 트리거에 의해 요청된 경우 허용여부에 상관없이 1로 세트된다. 만약 인터럽트가 허용된 경우 인터럽트 서비스와 함께 자동적으로 0으로 클리어 된다. 플랙에 1을 써 넣음으로 클리어 시킬 수 있다(0을 써 넣는 것이 아님에 주의하라). 레벨 트리거로 설정한 경우에는 항상 0으로 클리어 된다.

4.1.5. 실험21 : LED 점멸1

- 예제 : INT0(PORTD.0)에 연결된 스위치를 누르면 상승 에지 트리거에 의한 인터럽트가 발생하여 PORTC에 연결된 LED가 1칸 옮겨가면서 켜진다.
- 회로도 : 그림 80의 SW_CON을 그림 79의 D_CON에 연결하고 실험할 때는 LEDSW를 닫고 한다. 또한 그림 80의 SW1을 누르면서 실험한다.
- 프로그램 작성시 유의점 : 디버깅시 [Auto Step]를 누르고 IO View창의 PIND레지스터 비트0의 값을 변화시켜가며 PORTC의 변화를 관찰하라. PIND.0을 변화시킬 때마다 PORTC의 값이 변화해야 한다. CodeVisoinAVR에서 인터럽트 번호에 따른 이름을 예로 INT0~7은 EXT_INT0~EXT_INT7로 ATmega128의 경우에는 mega128.h에 정의해 놓았다. 이를 이용해서 프로그램을 짜도 된다. *그림 80의 회로에서 풀업 저항 10K옴 대신 무한대의 저항 즉 풀업 저항 10K옴을 없애고 개방하여 입력장치로 사용해도 된다. 이것은 I/O포트들이 입력으로 사용될 때 내부 풀업 저항을 사용할 수 있기 때문이다. 이런 경우 해당 포트 핀을 입력으로 설정하기 위해 DDRx.n=0 명령에 부가적으로 PORTx.n=1 명령으로 해당 포트 핀에 1을 써 넣어야 내부 풀업 저항을 사용할 수 있다.*

● 프로그램 예 :

```
#include <mega128.h>
#include <delay.h>        // delay_ms(), delay_us()를 이용할 것이다.
void main(void){
unsigned char led = 0xfe;
interrupt [2] void exint0(void){   // interrupt [EXT_INT0] void exint0(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
      if(led!=0x7f) led = (led << 1) | 0x01; // 비트7을 켜지 않았으면 1비트씩 좌로 시프트하고 빈자리에는 1을 채운다.
      else led = 0xfe;                  // 비트7을 켰으면 다시 처음으로 와서 비트 0을 켠다.
      PORTC = led;
      delay_ms(5);                                 // 5msec delay
}
void main(void){
      DDRC = 0xff; // 포트C를 출력으로
      PORTC = led;
      DDRD = 0x00; //포트D를 입력으로 디폴트 입력으로 설정되어 있으므로 필요하지는 않음
      SREG.7 = 1;         //인터럽트 전체 허용
      EIMSK |= 0x01;      // INT0 개별 허용
      EICRA = 3; // 상승에지 트리거
      while(1);
}
```

● 과제 : ① 실제 구현 했을 때 한 번 누를 때마다 하나씩 이동하지 않고 여러 개가 이동하는 현상이 발생할 것이다. 이는 실험 16에서도 이야기한 채터링에 의한 현상이다. delay_ms(5)의 값을 변화시켜 가면서 실험해 보고 적정한 시간 지연을 찾아 보아라. ② 그림 80의 회로 대신 그림84의 회로를 이용하되 시간지연을 작게 해서 실험해보고 비교해보라. ③외부인터럽트1~3으로 같은 역할을 하는 프로그램을 구현해보라.

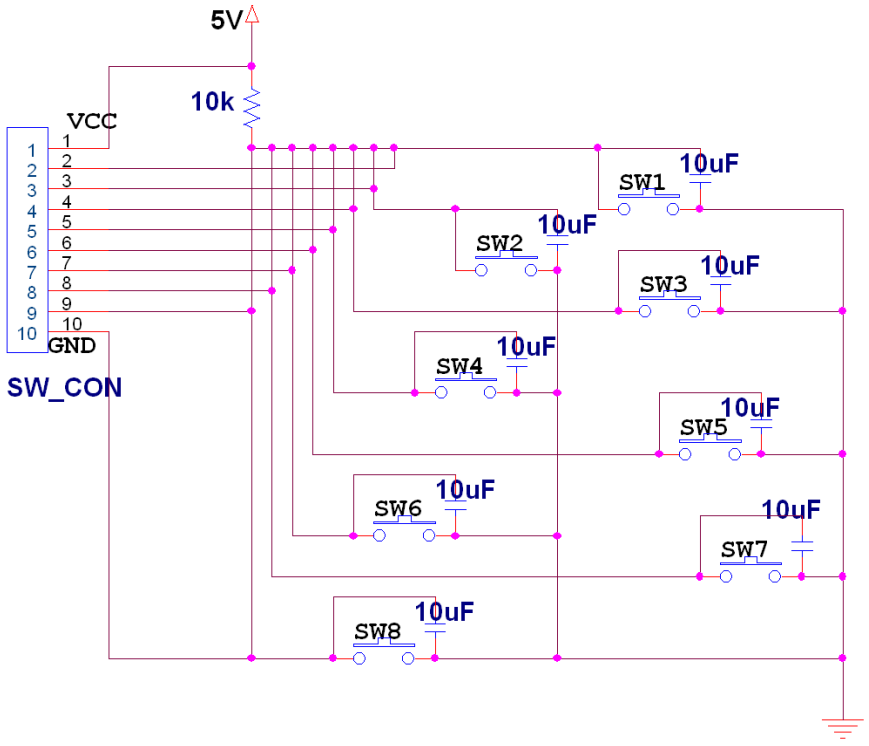


그림 84. 간단한 채터링 방지 키 입력 회로

4.1.6. 실험22 : LED 점멸2

● 예제 : INT0(PORTD.0)에 연결된 스위치를 누르면 상승 에지 트리거에 의한 인터럽트가 발생하여 PORTC에 연결된 LED가 1칸 좌로 옮겨가면서 켜진다. INT1(PORTD.1)에 연결된 스위치를 누르면 레벨 트리거에 의한 인터럽트가 발생하여 PORTC에 연결된 LED가 1칸 우로 옮겨가면서 켜진다.

● 회로도 : 그림 80의 SW_CON을 그림 79의 D_CON에 연결하고 실험할 때는 LEDSW를 닫고 한다. 또한 그림 80의 SW1과 SW2를 누르면서 실험한다.

● 프로그램 예 :

```
#include <mega128.h>
#include <delay.h>        // delay_ms(), delay_us()를 이용할 것이다.
unsigned char led = 0xfe;
interrupt [2] void exint0(void){  // interrupt [EXT_INT0] void exint0(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
      if(led!=0x7f) led = (led << 1) | 0x01; // 비트7을 켜지 않았으면 1비트씩 좌로 시프트하고 빈자리에는 1을 채운다.
      else led = 0xfe;                // 비트7을 켰으면 다시 처음으로 와서 비트 0을 켠다.
      PORTC = led;
      delay_ms(40);
}
interrupt [3] void exint1(void){  // interrupt [EXT_INT1] void exint1(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
      if (led != 0xfe) led = (led >> 1 ) | 0x80; /* 비트0를 켜지 않았으면 1비트씩 우로 시프트하고 빈자리에 1을 */
      else led = 0x7f;                          /* 비트0을 켰으면 다시 처음으로 와서 비트7를 켠다. */
      PORTC = led;
      delay_ms(40);
}
void main(void){
      DDRC = 0xff; // 포트C를 출력으로
      PORTC = led;
      DDRD = 0x00; //포트D를 입력으로 디폴트 입력으로 설정되어 있으므로 필요하지는 않음
      SREG.7 = 1;         //인터럽트 전체 허용
      EIMSK |= 0b00000011;         // INT0, 1 개별 허용
      EICRA = 3;                   // INT0 상승에지 트리거        , INT1 레벨 트리거
      while(1);
}
```

● 과제 : ① INT0(PORTD.0)와 INT1(PORTD.1)의 역할을 바꾼 프로그램을 구현해보라. ② INT2(PORTD.2)와 INT3(PORTD.3)을 이용하여 같은 역할을 하는 것을 구현해보라.

4.1.7. 실험23 : 카운터

● 예제 : 에지트리거되는 외부 인터럽트를 이용하여 카운트하는 프로그램으로 INT0(PORTD.0)에 연결된 SW1를 누르면 상승 에지 트리거에 의한 인터럽트가 발생하여 눌린 수를 세어 PORTC에 연결된 LED에 표시한다.

● 회로도 : 그림 80의 SW_CON을 그림 79의 D_CON에 연결하고 실험할 때는 LEDSW를 닫고 한다. 또한 그림 80의 SW1을 누르면서 실험한다.

● 프로그램 예 :

```
#include <mega128.h>
#include <delay.h>        // delay_ms(), delay_us()를 이용할 것이다.
unsigned char led;
interrupt [2] void exint0(void){  // interrupt [EXT_INT0] void exint0(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
      PORTC = ~(++led);            /* led를 1 증가시키고 반전한 것을 출력시킨다 */
      delay_ms(5);
}
void main(void){
      DDRC = 0xff; // 포트C를 출력으로
      DDRD = 0x00; //포트D를 입력으로 디폴트 입력으로 설정되어 있으므로 필요하지는 않음
      SREG.7 = 1;         //인터럽트 전체 허용
      EIMSK = 0b00000001;          // INT0 개별 허용
      EICRA = 3;                   // INT0 상승에지 트리거
      PORTC = ~led;      /* 0일때 LED가 켜지고 1일때 꺼지므로 반전시켜서 출력 */
      for(;;);                     /* 인터럽트 발생할 때까지 대기 */
}
```

● 과제 : ① 실제 구현 했을 때 한 번 누를 때마다 하나씩 이동하지 않고 여러 개가 이동하는 현상이 발생할 것이다. 이는 실험 16에서도 이야기한 채터링에 의한 현상이다. delay_ms(5)의 값을 변화시켜 가면서 실험해 보고 적정한 시간 지연을 찾아 보아라. ② 그림 80의 회로 대신 그림84의 회로를 이용하되 시간지연을 작게 해서 실험해보고 비교해보라. ③외부인터럽트1~3으로 같은 역할을 하는 프로그램을 구현해보라. ④ INT0(PORTD.0)에 연결된 SW1를 누르면 증가하고 INT1(PORTD.1)에 연결된 SW2를 누르면 감소하는 프로그램을 구현해보라.

4.2. 타이머 실험

● ATmega128에는 타이머0~3 모두 4개의 타이머/카운터가 존재한다. 이들은 인터럽트와 PWM 출력 기능을 갖고 있다. 그 중에서 타이머0, 2는 8비트로 서로 기능이 유사하고 타이머1, 3는 16비트로 서로 기능이 유사하다. 다만 타이머0은 다른 타이머와 달리 32.768KHz의 수정 진동자에 접속하는 TOSC1,2 단자를 가지고 있어서 Real Time Clock의 기능을 갖고 있다.

● 인터럽트는 오버플로에 의해 발생하는 오버플로 인터럽트, 카운터 값이 출력비교 레지스터의 값과 같게 되는 경우 발생하는 출력비교 인터럽트, 입력 캡쳐 인터럽트가 있다.

● PWM 기능은 출력비교 기능을 이용하여 출력비교 신호에 주기와 duty ratio를 가변할 수 있는 출력신호를 생성하여 OC0, OC2, OC1A~C, OC3A~C 단자로 출력하는 것이다.

● 타이머1, 3은 외부 트리거 신호에 의해 현재의 카운터 값을 캡쳐할 수 있는 기능을 갖고 있다.

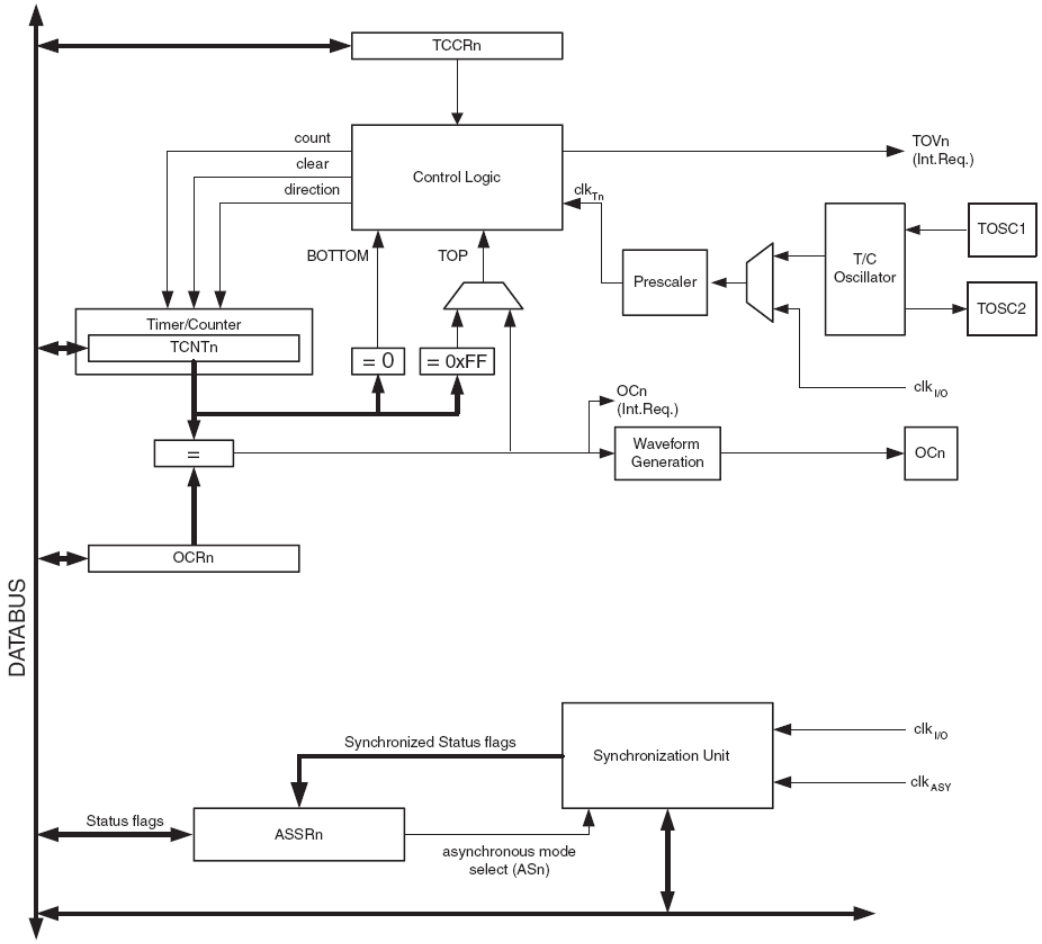


그림 85. 타이머0의 블록선도(출처:ATMEL)

4.2.1. 타이머 카운터0, 2

PWM 및 비동기 동작 모드를 갖는 8비트 업/다운 카운터로 10비트의 프리스케일러를 갖고 있어 이를 통하여

내부 클럭을 소스로 받아 타이머나 카운터로 동작한다. 오버플로와 출력비교에 의한 인터럽트 기능을 갖고 있으며 주파수를 발생할 수 있는 기능을 갖고 있고 출력비교와 재장전 기능을 갖고 있다. 타이머0의 경우 부가적으로 32.768KHz의 수정 진동자를 TOSC1,2 단자에 연결해 Real Time Clock으로 사용할 수 있는 기능이 있다.

4.2.1.1. 구성

그림 85는 8비트 타이머의 블록선도를 보여준다.

- 프리스케일러(prescaler) : 타이머0의 경우에는 TOSC1에 입력된 시스템클럭 주파수의 1/4미만의 외부클럭, TOSC1과 TOSC2단자에 연결된 수정 진동자에 의해 발생되는 클럭이나 내부 클럭 $clk_{I/O}$을 클럭 소스로 사용하여 1, 8, 32, 64, 128, 256, 1024의 분주비로 프리스케일러에서 나누어 clk_{T0}클럭을 생성한다. 타이머0의 클럭 소스는 ASSR레지스터로 선택한다. 타이머2의 경우에는 TOSCT1,2핀에 해당하는 것이 없고 대신 T2핀으로 입력되는 외부 클럭을 프리스케일러를 거치지 않고 clk_{T2}클럭을 생성하거나 내부 클럭 $clk_{I/O}$을 클럭 소스로 사용하여 1, 8, 64, 256, 1024의 분주비로 프리스케일러에서 나누어 clk_{T2}클럭을 생성한다. 분주비의 선택은 TCCRn 레지스터로 설정한다.
- TCCRn : 타이머n의 동작모드를 설정하는 레지스터(n은 0 또는 2)
- TCNTn (Timer/CouNTer n) 레지스터 : 프리스케일러에서 생성된 clk_{Tn}클럭을 제어로직에 따라 업/다운 카운팅을 수행하거나 리셋되는 타이머n의 8비트 카운터 값을 저장하는 레지스터.
- TOVn : TCNTn가 카운팅에 의해 0xFF에서 0x00로 오버플로우가 일어나면 TOVn플랙이 세트되고 오버플로 인터럽트를 발생시킬 수 있게 된다.
- OCRn(Output Compare Resister n) 레지스터 : TCNTn의 값과 출력 비교되기 위한 8비트 데이터 값을 저장하고 있는 레지스터로 TCNTn의 값과 같으면 OCFn플랙을 세트시키고 출력비교 인터럽트를 요청하고 출력핀 OCn단자로 적절한 데이터가 출력되도록 한다.
- 제어로직(Control Logic) : clk_{Tn}와 TCNTn가 제로가 되었을 때 발생하는 BOTTOM 신호, TCNTn 카운터값이 0xFF에 도달하거나 OCRn 레지스터에 설정된 값에 도달할 때 발생하는 TOP신호를 입력으로 받아들여 TCNTn 카운터를 업/다운 카운팅을 수행하게 하거나 리셋시킨다.

4.2.1.2. TCCRn 레지스터

TCCRn(Timer/Counter Control Register n)는 타이머n의 동작모드를 설정하고 프리스케일러의 분주비를 설정한다.

- 비트7(FOCn:Force Output Compare n) : PWM 모드가 아닌 경우에만 유효하며 1로 설정하면 강제로 OC0(PB4, 핀14) 단자(타이머2의 경우 OC2(PB7, 핀17)에 출력비교가 일치할때 출력하게 되는 값과 동일한 출력을 내보낸다. 이 경우 인터럽트가 발생하지도 않고 CTC모드에서 TCNTn를 클리어시키지도 않으므로 보통 0으로 설정한다.
- 비트6, 3(WGMn0, n1: Waveform Generation Mode n0, n1) : 타이머n는 4가지의 동작모드를 갖는데 각각 Normal(일반), CTC, PC(Phase Correct) PWM, 고속 PWM이다. WGMn1:n0을 00으로 하면 일반모드, 01로 하면 PC PWM, 10으로 하면 CTC, 11로 하면 고속 PWM모드로 설정한다.
- 비트5,4(COMn1, n0 : Compare Match output mode n1, n0) : OCn핀의 동작을 다음처럼 조절한다.

  ① 00 : 정상적인 범용 입출력 포트로 동작(OCn 출력을 차단)

  ② 01 : PWM모드가 아닌 경우 출력비교에 의해 OCn 출력을 토글시키며, PWM 모드에서는 보류되어 있다.

  ③ 10 : PWM모드가 아닌 경우 출력비교에 의해 OCn 출력을 클리어, FAST PWM 모드의 경우 OCn를 클리어하고 TOP신호를 받아 OCn를 1로 세트, Phase Correct PWM 모드 상향카운팅의 경우 OCn를 클리어하고 PWM 모드 하향카운팅의 경우 OCn를 1로 세트 시킨다.

④ 11 : PWM모드가 아닌 경우 출력비교에 의해 OCn 출력을 세트, FAST PWM 모드의 경우 OCn를 세트하고 TOP신호를 받아 OCn를 클리어, Phase Correct PWM 모드 상향카운팅의 경우 OCn를 세트하고 PWM 모드 하향카운팅의 경우 OCn를 클리어 시킨다.

- 비트2~0(CSn2~n0: Clock Selectn2~n0) : n=0 즉 타이머0일 때 클럭의 분주비를 선택하여 000일 때 클럭입력을 차단하고, 001일때 1분주, 010일때 8분주, 011일때 32분주, 100일때 64분주, 101일때 128분주, 110일때 256분주, 111일때 1024분주비가 되도록 한다. n=2 즉 타이머2의 경우에는 분주비와 클럭소스를 선택하여 000일 때 클럭입력을 차단하고, 001일때 1분주, 010일때 8분주, 011일때 64분주, 100일때 256분주, 101일때 1024분주비가 되도록 하고 110일때 T2(PD7, 핀32)핀에서 입력되는 외부 클럭의 하강에지에서 카운터2가 동작하고, 111일때는 T2(PD7, 핀32)핀에서 입력되는 외부 클럭의 상승에지에서 카운터2가 동작하도록 한다.

4.2.1.3. TIMSK 레지스터

TIMSK(Timer Interrupt MaSK) 레지스터는 타이머0,1,2가 발생하는 인터럽트를 개별적으로 허용 제어하는 레지스터이다.

- 비트7(OCIE2:Output Compare match Interrupt Enable for timer2) : 1로 설정되면 타이머2의 출력비교 인터럽트를 개별적으로 허용한다.
- 비트6(TOIE2:Timer Overflow Interrupt Enable for timer2) : 1로 설정되면 타이머2의 오버플로 인터럽트를 개별적으로 허용한다.
- 비트1(OCIE0:Output Compare match Interrupt Enable for timer0) : 1로 설정되면 타이머0의 출력비교 인터럽트를 개별적으로 허용한다.
- 비트0(TOIE0:Timer Overflow Interrupt Enable for timer0) : 1로 설정되면 타이머0의 오버플로 인터럽트를 개별적으로 허용한다.

4.2.1.4. TIFR 레지스터

TIFR(Timer Interrupt Frag Register) 레지스터는 타이머0~2가 발생하는 인터럽트 플랙을 저장한다. *소프트웨어적으로 강제로 클리어 하려면 1을 기록해야 한다.*

- 비트7(OCF2 : Output Compare match Flag 2) : 타이머2의 TCNT2레지스터와 출력비교 레지스터 OCR2의 값을 비교하여 같으면 OCF2는 1로 세트되고 출력비교 인터럽트가 요청된다. 인터럽트가 서비스되기 시작하면 자동적으로 클리어된다.
- 비트6(TOV2 : Timer Overflow Flag 2) : 타이머2에서 오버플로가 발생하면(0xFF까지 세고 0x00으로 넘어가게 되면) 이 TOV2는 1로 세트되면서 오버플로 인터럽트가 요청된다. 인터럽트가 서비스되기 시작하면 자동적으로 클리어된다. Phase correct PWM 모드에서는 타이머2이 0x00에서 계수방향을 바꿀 때 TOV2가 세트된다.
- 비트1(OCF0 : Output Compare match Flag 0) : 타이머0의 TCNT0레지스터와 출력비교 레지스터 OCR0의 값을 비교하여 같으면 OCF0는 1로 세트되고 출력비교 인터럽트가 요청된다. 인터럽트가 서비스되기 시작하면 자동적으로 클리어된다.
- 비트0(TOV0 : Timer Overflow Flag 0) : 타이머0에서 오버플로가 발생하면(0xFF까지 세고 0x00으로 넘어가게 되면) 이 TOV0는 1로 세트되면서 오버플로 인터럽트가 요청된다. 인터럽트가 서비스되기 시작하면 자동적으로 클리어된다. Phase correct PWM 모드에서는 타이머0이 0x00에서 계수방향을 바꿀 때 TOV0가 세트된다.

4.2.1.5. ASSR 레지스터

ASSR(ASynchronous Status Register) 레지스터는 타이머0이 클럭소스를 선택하거나 외부 클럭에 의해 비동기 모드로 동작하는 경우 제어를 위한 레지스터이다.

- 비트3(AS0 : ASynchronous timer 0) : 0이면 내부클럭 $clk_{I/O}$를 클럭소스로 설정하여 동기모드로 타이머0가 동작하게 하고 1이면 TOSC1을 통해 들어오는 외부클럭을 소스로 설정하여 비동기모드로 동작하게 한다.

AS0의 수정은 TCNT0, OCR0, TCCR0의 내용의 변동을 초래하고 예상하지 못한 인터럽트가 발생할 수 있으므로 주의해야 한다.

- 비트2(TCN0UB : Timer/CouNter0 Uptate Busy) : 타이머0이 외부에서 TOSC1 단자로 입력되는 클럭을 소스로 하여 비동기모드로 동작하고 있을 때 TCNT0레지스터에 새로운 값을 기록하면 1로 세트된다. 기록이 완료되면 0으로 자동적으로 클리어되어 TCNT0에 새로운 값을 기록할 수 있음을 표시한다.
- 비트1(OCR0UB : Output Compare Register 0 Update Busy) : 타이머0이 외부에서 TOSC1 단자로 입력되는 클럭을 소스로 하여 비동기모드로 동작하고 있을 때 OCR0레지스터에 새로운 값을 기록하면 1로 세트된다. 기록이 완료되면 0으로 자동적으로 클리어되어 OCR0에 새로운 값을 기록할 수 있음을 표시한다.
- 비트0(TCR0UB : Timer Conter Register 0 Update Busy) : 타이머0이 외부에서 TOSC1 단자로 입력되는 클럭을 소스로 하여 비동기모드로 동작하고 있을 때 TCCR0 레지스터에 새로운 값을 기록하면 1로 세트된다. 기록이 완료되면 0으로 자동적으로 클리어되어 TCCR0 에 새로운 값을 기록할 수 있음을 표시한다.

4.2.1.6. SFIOR 레지스터

SFIOR(Special Function I/O Register)는 타이머들을 동기화시키는데 사용되는 레지스터이다.

- 비트7(TSM : Timer Synchronization Mode) : 0으로 클리어하면 PSR0, PSR321비트가 하드웨어적으로 클리어되며 타이머들이 동시에 카운팅 동작을 시작한다. 1로 하면 PSR0, PSR321비트에 기록한 값을 유지하여 대응하는 프리스케일러 리셋신호를 발생하고 해당 타이머의 동작을 정지시켜 같은 값으로 설정할 수 있도록 한다.
- 비트1(PSR0 : Prescaler Reset Timer 0) : 1로하면 타이머0의 프리스케일러를 리셋시키며 완료후에 자동적으로 클리어된다. TSM = 1이면 자동적으로 클리어되지 않는다.
- 비트0(PSR123 : Prescaler Reset Timer 123) : 1로하면 타이머0의 프리스케일러를 리셋시키며 완료후에 자동적으로 클리어된다. TSM = 1이면 자동적으로 클리어되지 않는다.

4.2.1.7. 타이머0,2의 일반 모드

- 설정 : TCCRn레지스터의 WGMn1:n0 = 00으로 설정한다.
- 동작 : 타이머n는 항상 상향 카운터로만 동작한다. TCNTn의 값이 255에서 0으로 바뀌는 순간 TIFR레지스터의 TOVn비트가 세트되며 오버플로 인터럽트 발생이 요청된다. 출력비교 인터럽트를 일반모드에서 사용할 수도 있으나 FAST PWM모드를 사용하는 것이 좋다.
- 시간간격 설정 : 오버플로 인터럽트가 요청될 때 마다 인터럽트를 발생시켜 TCNTn의 초기값 T_0을 적당하게 설정하면 원하는 시간 간격을 얻을 수 있다. TCCRn의 비트2~0(CSn2~0)값에 따라 1, 8, 32, 64, 128, 256, 1024 분주가 되므로(타이머2의 경우 1,8, 64, 256, 1024) 클럭을 fMHz로 사용하는 경우 분주비를 N이라 하면 다음에 의해 시간 간격 T가 결정된다. 즉 오버플로 인터럽트가 요청되는 시간간격은 T이다.

$$T[\mu\sec] = N \times (256 - T_0)/f$$

4.2.1.8. 타이머0,2의 CTC모드

- 설정 : TCCRn레지스터의 WGMn1:n0 = 10으로 설정한다.
- 동작 : CTC(Clear Timer on Compare match)모드에서는 TCNTn의 값이 OCRn레지스터에 설정한 값과 일치되면 그 다음 클럭 사이클에서 TCNTn의 값이 0으로 클리어된다. 이때 TIFR레지스터의 OCFn비트가 세트되며 출력비교 인터럽트 발생이 요청된다. 이때 인터럽트를 발생시켜 OCRn의 값을 바꾸면 주기를 원하는 대로 변경할 수 있다. OCn단자로 하여금 출력파형을 발생하도록 할 수 있는데 TCCRn의 COMn1~n0을 01로 설정하고 OCRn레지스터 값을 바꿔가면서 출력비교에 의해 OCn의 신호를 토글하는 경우를 그림 86이 보여준다.
- 주기 설정 : TCCRn의 COMn1~n0을 01로 설정하고 OCRn레지스터 값을 어떤 일정한 값 OCR0로 놓고 출력비교에 의해 OCn의 신호를 토글하는 경우 TCCRn의 비트2~1(CSn2~n0)값에 따라 1, 8, 32, 64, 128, 256, 1024 분주가 되므로(타이머2의 경우 1,8, 64, 256, 1024) 클럭을 fMHz로 사용하고 분주비를 N이라 하면 다음

에 의해 주기 T가 결정된다. 출력비교 인터럽트가 요청되는 시간간격은 $T/2$이다.

$$T[\mu\text{sec}] = 2N(1 + OCR0)/f$$

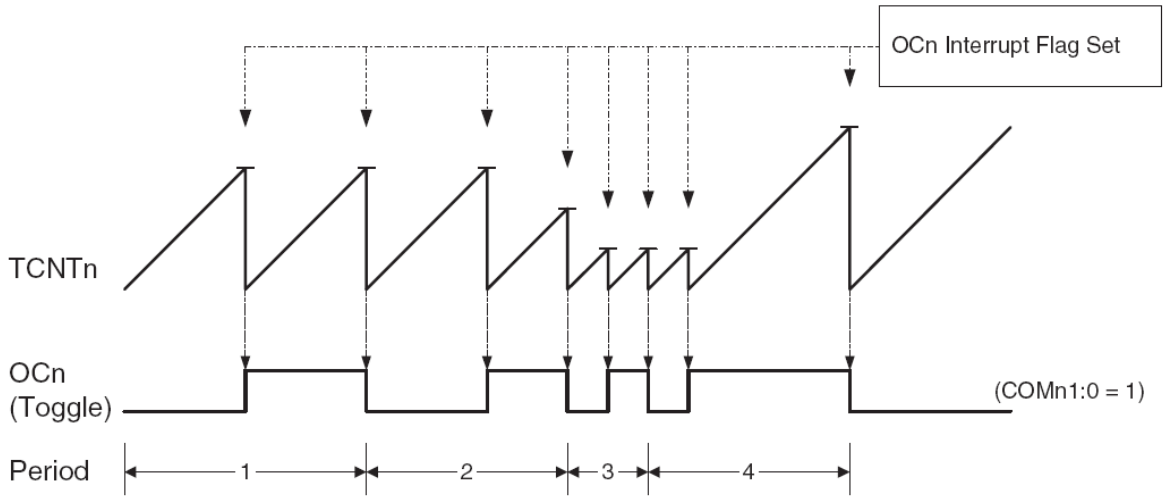


그림 86. 타이머0,2의 CTC모드 동작(출처:ATMEL)

4.2.1.9. 타이머 0, 2의 고속 PWM 모드

- 설정 : TCCRn레지스터의 WGMn1:n0 = 11으로 설정한다.
- 동작 : TCNTn는 반복적으로 업카운팅하며 항상 0x00~0xFF의 값을 갖는다. TCNTn의 값이 OCRn레지스터에 설정한 값과 일치되면 TIFR레지스터의 OCFn비트가 세트되며 출력비교 인터럽트가 요청되며 OCn의 값은 0으로 클리어 된다. 그리고 TCNTn는 업카운팅을 계속하여 TCNTn는 255까지 증가했다가 0으로 바뀌는 순간 TIFR레지스터의 TOVn비트가 세트되어 오버플로 인터럽트가 요청되며 OCn는 1로 세트된다. (그림 87참조). TOVn비트가 세트되어 오버플로 인터럽트가 요청될 때 인터럽트를 발생시켜 OCRn의 값을 바꾸어 듀티비를 제어할 수 있다. OCRn레지스터 값과 듀티비가 비례함을 그림 88에서 알 수 있다. OCRn = 0x00일 때 듀티비는 1/256이며 OCRn=0xFF일 때 100%가 됨을 알 수 있다. 레귤레이터, 정류 및 DA 변환에 적합하다. 특히 고주파를 사용하기 때문에 코일, 콘덴서 등 외부소자가 필요 없어 비용을 절감시킬 수 있다.

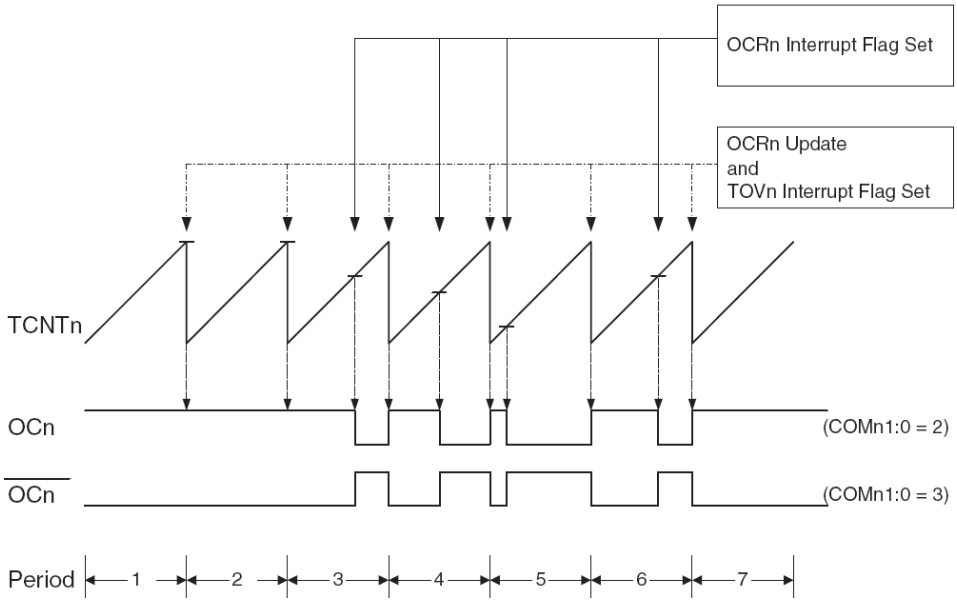


그림 87. 타이머의 고속 PWM 모드 동작(출처:ATMEL)

- 주기 설정 : TCCRn의 COMn1~n0을 10 혹은 11로 설정하고 OCRn레지스터 값을 변화시켜가며 출력비교에 의해 OCn의 신호를 PWM하는 경우 TCCRn의 비트2~1(CSn2~n0)값에 따라 1, 8, 32, 64, 128, 256, 1024 분

주가 되므로(타이머2의 경우 1,8, 64, 256, 1024) 클럭을 fMHz로 사용하고 분주비를 N이라 하면 다음에 의해 주기 T가 결정된다. 즉 오버플로 인터럽트가 요청되는 시간간격은 T이다.

$$T[\mu\sec] = N \times 256/f$$

4.2.1.10. 타이머 0, 2의 Phase Correct PWM 모드

- 설정 : TCCRn레지스터의 WGMn1:n0 = 01으로 설정한다.
- 동작 : TCNTn는 업카운팅하여 0xFF으로 증가하다가 다운카운팅으로 0x00으로 감소를 반복한다. TCCRn의 COMn1~n0을 10 혹은 11로 설정하고 OCRn레지스터 값을 변화시켜가며 출력비교에 의해 OCn의 신호를 PWM하는 경우TCNTn의 값이 OCRn레지스터에 설정한 값과 일치되면 TIFR레지스터의 OCFn비트가 세트되며 출력비교 인터럽트 발생이 요청되며 OCn의 값은 상향 카운팅의 경우에는 0으로 하향의 경우에는 1로 세트된다. 그리고 TCNTn는 업카운팅을 계속하여 TCNTn는 255에 도달하면 OCR2의 값이 경신되고 하향카운팅이 시작된다. 하향카운팅을 하다가 0에 도달하면 TIFR레지스터의 TOVn비트가 세트되며 오버플로 인터럽트가 요청된다(그림 88참조). PC PWM모드에서는 OCRn 레지스터에 이중 버퍼링 기능이 있어 PWM 주기를 변경하기 위해 OCRn 레지스터에 새로운 값을 기록하더라도 즉기 변경되지 않고 TCNTn이 255에 도달하면 갱신된다(그림 88 참조). OCRn레지스터 값과 듀티비가 비례함을 그림 88에서 알 수 있다. OCRn = 0x00일 때 듀티비는 0이며 OCRn=0xFF일 때 100%가 됨을 알 수 있다.

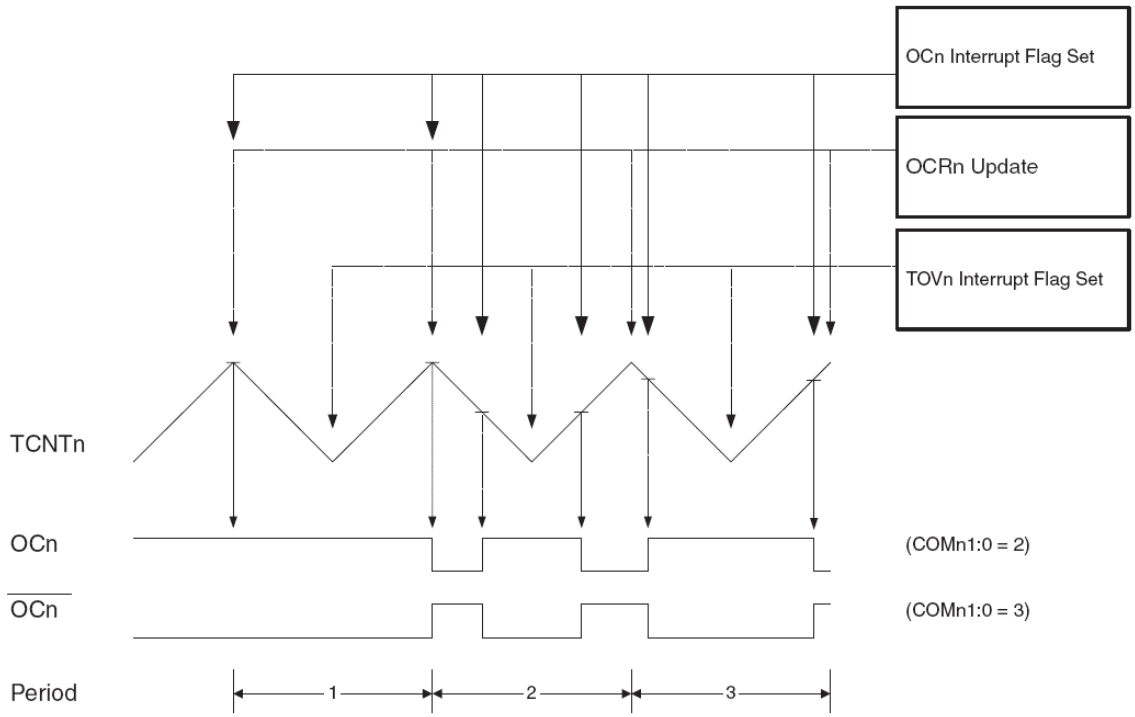


그림 88. 타이머0, 2의 위상교정 PWM 모드 동작 (출처:ATMEL)

- 주기 설정 : TCCRn의 COMn1~n0을 10 혹은 11로 설정하고 OCRn레지스터 값을 변화시켜가며 출력비교에 의해 OCn의 신호를 PWM하는 경우 TCCRn의 비트2~1(CSn2~n0)값에 따라 1, 8, 32, 64, 128, 256, 1024 분주가 되므로(타이머2의 경우 1,8, 64, 256, 1024) 클럭을 fMHz로 사용하고 분주비를 N이라 하면 다음에 의해 주기 T가 결정된다. 즉 오버플로 인터럽트가 요청되는 시간간격은 T이다.

$$T[\mu\sec] = 2N \times 255/f$$

4.2.2. 타이머 카운터1, 3

3개의 PWM 출력 및 캡춰 기능을 갖는 16 비트 업/다운 카운터로 10비트의 프리스케일러를 갖고 있어 이를 통하여 내부 클럭을 소스로 받아 타이머나 카운터로 동작한다. 오버플로와 출력비교에 의한 인터럽트 이외에 입력

캡춰 인터럽트 기능을 갖고 있고 주파수를 발생할 수 있는 기능과 출력비교와 재장전 기능을 갖고 있다.

4.2.2.1. 구성

그림 89는 타이머1과 3의 블록선도를 보여준다.

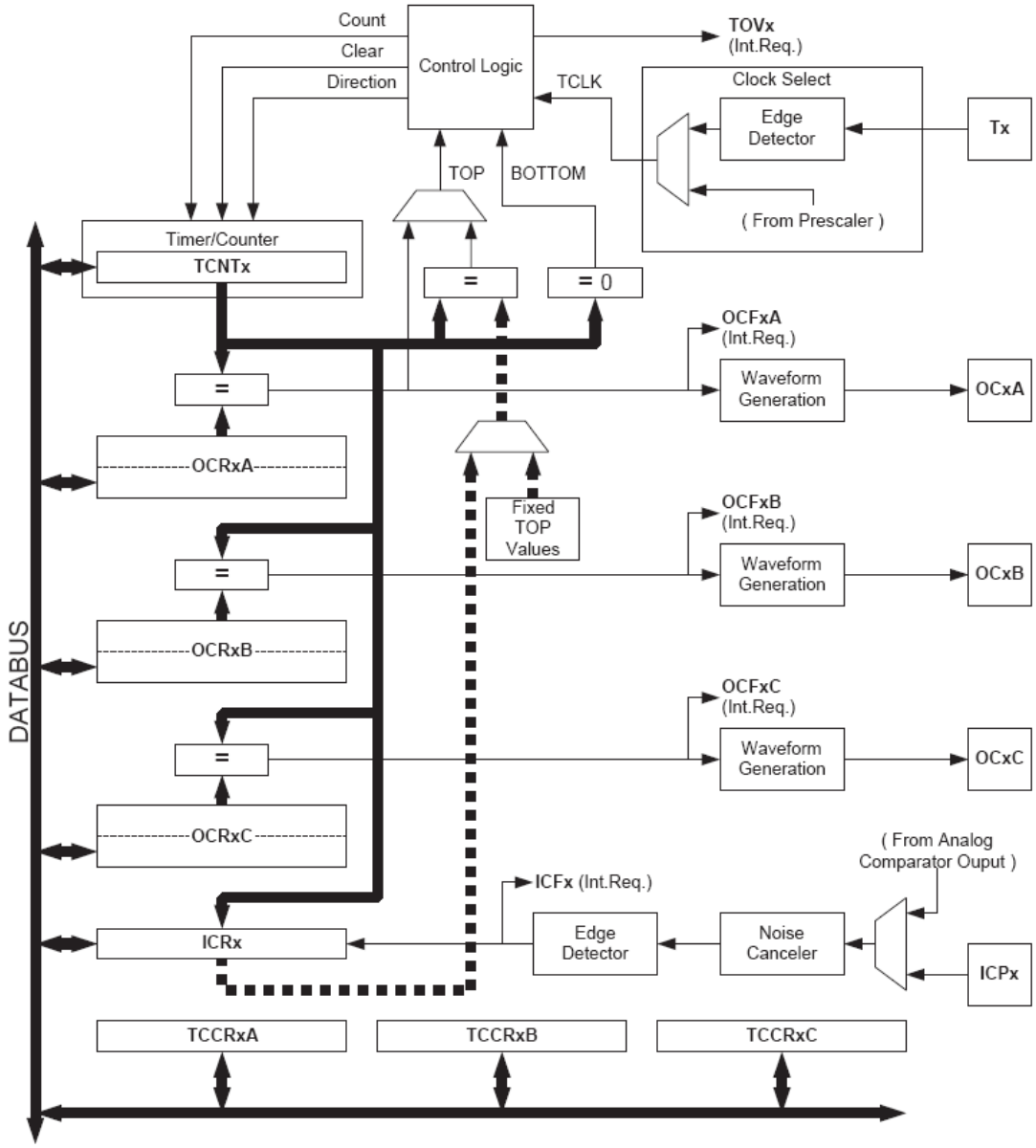


그림 89. 타이머1,3의 블록선도(출처:ATMEL)

- 프리스케일러(prescaler) : 내부 클럭 $clk_{I/O}$을 클럭 소스로 사용하여 1, 8, 64, 256, 1024의 분주비로 나누어 TCLK에 사용될 클럭을 생성한다. 분주비의 선택은 TCCRxB레지스터로 설정한다(x는 1 또는 3).
- TCLK : Tx핀으로 입력되는 외부 클럭을 프리스케일러를 거치지 않고 TCLK클럭이 생성되거나 내부 클럭 $clk_{I/O}$을 프리스케일러에서 나누어 TCLK클럭이 생성된다.
- TCCRxA~C : 타이머x의 설정을 위한 레지스터.
- TCNTx (Timer/CouNTer x) 레지스터 : TCLK클럭을 제어로직에 따라 업/다운 카운팅을 수행하거나 리셋되는 타이머x의 16비트 카운터 값을 저장하는 레지스터.
- TOVx : TCNTx가 카운팅에 의해 0xFFFF에서 0x0000로 오버플로우가 일어나면 TOVx플랙이 세트되고 오버플로 인터럽트를 발생시킬 수 있게 된다.
- OCRxA~C(Output Compare Resister x A~C) 레지스터 : TCNTx의 값과 출력 비교되기 위한 16비트 데이

터 값을 저장하고 있는 레지스터로 TCNTx의 값과 같으면 OCFxA~C플랙이 세트되고 출력비교 인터럽트를 요청하여 출력핀 OCxA~C단자로 적절한 데이터가 출력되도록 한다.

- ICRx(Input Capture Register x) : 입력캡쳐핀 ICx(타이머1의 경우에는 아날로그 비교기의 출력신호도 가능)으로 들어오는 신호변화를 검출하여 일어나는 입력캡쳐시 TCNTx의 카운터 값을 저장하는 16비트 레지스터로 이때 ICFx 플랙이 세트되고 입력캡쳐인터럽트가 요청된다. 어떤 신호의 주기 측정에 응용될 수 있다.

- 제어로직(Control Logic) : TCNTx가 제로가 되었을 때 발생하는 BOTTOM 신호, TCNTx 카운터값이 0xFFFF에 도달하거나 카운터가 실제로 도달할 수 있는 최대값 혹은 OCRxA~C, ICRx에 설정된 값에 도달하면 발생하는 TOP신호를 입력으로 받아들여 TCNTx 카운터를 업/다운 카운팅을 수행하게 하거나 리셋시킨다.

4.2.2.2. TCCRxA 레지스터

TCCRxA(Timer/Counter Control Register x A)는 타이머x의 동작모드와 출력비교 단자의 파형 발생법 등을 설정한다.

- 비트7~6(COMxA1~0:Compare match Output Mode for channel A) : OCxA핀의 동작을 아래처럼 설정.

- 비트5~4(COMxB1~0:Compare match Output Mode for channel B) : OCxB핀의 동작을 아래처럼 설정.

- 비트3~2(COMxC1~0:Compare match Output Mode for channel C) : OCxC핀의 동작을 아래처럼 설정.

① 00 : 정상적인 범용 입출력 포트로 동작(OCxn 출력을 차단, n=A,B,C)

② 01 : PWM모드가 아닌 경우 출력비교에 의해 OCxn 출력을 토글시키며, FAST PWM 모드에서는 모드 15의 경우 출력비교에 의해 OCxA 출력을 토글하고 OCxB, OCxC는 출력을 차단하고 기타의 모드에서 OCxn 출력을 차단하며, Phase Correct PWM 모드에서 모드9,11의 경우 출력비교에 의해 OCxA 출력을 토글하고 OCxB, OCxC는 출력을 차단하고 기타의 모드에서 OCxn 출력을 차단한다.

③ 10 : PWM모드가 아닌 경우 출력비교에 의해 OCxn 출력을 클리어, FAST PWM 모드에서 출력 비교에 의해 OCxn출력을 클리어하고 TOP신호를 받아 OCxn를 1로 세트, Phase Correct PWM 모드 상향카운팅의 경우 OCxn를 클리어하고 PWM 모드 하향카운팅의 경우 OCxn를 1로 세트 시킨다.

④ 11 : PWM모드가 아닌 경우 출력비교에 의해 OCxn 출력을 세트, FAST PWM 모드의 경우 OCxn를 세트하고 TOP신호를 받아 OCxn를 클리어, Phase Correct PWM 모드 상향카운팅의 경우 OCxn를 세트하고 PWM 모드 하향카운팅의 경우 OCxn를 클리어 시킨다.

- 비트1~0(WGMx0~1: Waveform Generation Mode x 1~0) : 타이머x는 16가지의 동작모드를 갖는데 TCCRxB의 비트4~3(WGMx3~2)와 결합하여 동작모드를 설정한다. 그림 90은 모드의 번호, WGMx4~0 비트값, 모드의 동작, TOP 값, OCRx 레지스터의 업데이트 시점, TOVx 플랙의 세트 시점을 보여준다.

4.2.2.3. TCCRxB 레지스터

TCCRxB(Timer/Counter Control Register x B)는 타이머x의 동작모드와 분주비 등을 설정한다.

- 비트7(ICNCx:Input Capture Noise Canceler x) : 1로 세트하면 입력캡쳐단자 ICx로 입력되는 캡쳐신호를 위한 노이즈 회로를 작동시켜 필터링을 한다. 필터링에 의해 캡쳐신호가 4클럭 지연된다.

- 비트6(ICESx:Input Capture Edge Selectr x) : 1로 세트하면 ICx로 입력되는 캡쳐신호가 상승에지일 때 0이면 하강에지일때 캡쳐가 수행된다.

- 비트5 : 보류된 비트

- 비트3~2(WGMx3~2: Waveform Generation Mode x 3~2) : 타이머x는 16가지의 동작모드를 갖는데 TCCRxA의 비트1~0(WGMx1~0)와 결합하여 동작모드를 설정한다. 그림 90은 모드의 번호, WGMx4~0 비트값, 모드의 동작, TOP 값, OCRx 레지스터의 업데이트 시점, TOVx 플랙의 세트 시점을 보여준다.

- 비트2~0(CSx2~0: Clock Select x2~x0) : 분주비와 클럭소스를 선택하여 000일 때 클럭입력을 차단하고,

001일때 1분주, 010일때 8분주, 011일때 64분주, 100일때 256분주, 101일때 1024분주비가 되도록 하고 110일때 Tx핀에서 입력되는 외부 클럭의 하강에지에서 카운터x가 동작하고, 111일때는 Tx핀에서 입력되는 외부 클럭의 상승에지에서 카운터x가 동작하도록 한다.

Mode	WGMx3	WGMx2	WGMx1	WGMx0	Timer/Counter Mode of Operation	TOP	Update of OCRxn at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRxA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRx	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCRxA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRx	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRxA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRx	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICRx	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRxA	BOTTOM	TOP

그림 90. 타이머1,3의 동작 모드 설정(출처:ATMEL)

4.2.2.4. TCCRxC 레지스터

- 비트7~5(FOCxA~C:Force Output Compare x A~C) : PWM 모드가 아닌 경우에만 유효하며 1로 설정하면 강제로 OCxA~C에 출력비교가 일치할 때 출력하게 되는 값과 동일한 출력을 내보낸다. 이 경우 인터럽트가 발생하지도 않고 CTC모드에서 TCNTx를 클리어시키지도 않으므로 보통 0으로 설정한다.

4.2.2.5. TIMSK 레지스터

TIMSK(Timer Interrupt MaSK) 레지스터는 타이머0,1,2가 발생하는 인터럽트를 개별적으로 허용 제어하는 레지스터이다.

- 비트5(TICIE1:Timer Input Capture Interrupt Enable 1) : 1로 설정되면 타이머1의 입력캡쳐 인터럽트를 개별적으로 허용한다.
- 비트4~3(OCIE1A~B: Output Compare match Interrupt Enable timer 1 A~B) : 1로 설정되면 타이머1의 출력비교 인터럽트 A, B를 개별적으로 허용한다.
- 비트2(TOIE1:Timer Overflow Interrupt Enable for timer 1) : 1로 설정되면 타이머1의 오버플로 인터럽트를 개별적으로 허용한다.

4.2.2.6. ETIMSK 레지스터

ETIMSK(Extended Timer Interrupt MaSK) 레지스터는 타이머1, 3이 발생하는 인터럽트를 개별적으로 허용 제어하는 레지스터이다.

- 비트5(TICIE3:Timer Input Capture Interrupt Enable 3) : 1로 설정되면 타이머3의 입력캡쳐 인터럽트를 개별적으로 허용한다.
- 비트4~3(OCIE3A~B: Output Compare match Interrupt Enable timer 3 A~B) : 1로 설정되면 타이머3의

출력비교 인터럽트 A, B를 개별적으로 허용한다.

- 비트2(TOIE3:Timer Overflow Interrupt Enable for timer 1) : 1로 설정되면 타이머3의 오버플로 인터럽트를 개별적으로 허용한다.
- 비트1~0(OCIExC: Output Compare match Interrupt Enable timer x C) : 1로 설정되면 타이머x의 출력비교 인터럽트 C를 개별적으로 허용한다.

4.2.2.7. TIFR 레지스터

TIFR(Timer Interrupt Frag Register) 레지스터는 타이머0~2가 발생하는 인터럽트 플랙을 저장한다. *소프트웨어적으로 강제로 클리어 하려면 1을 기록해야 한다.*

- 비트5(ICF1 : Input Capture Flag 1) : 타이머1의 입력캡쳐신호 또는 아날로그 비교기로부터의 신호에 의해 캡쳐 동작이 수행될 때 1로 세트되고 입력캡쳐 인터럽트가 요청된다. ICR1 레지스터가 TOP 값으로 사용되는 동작 모드에서 TCNT1의 값이 TOP과 같아질 때 1로 세트되고 인터럽트가 요청된다. 인터럽트가 서비스되기 시작하면 자동적으로 클리어된다.
- 비트4~3(OCF1A~B : Output Compare match Flag 1 A~B) : 타이머1의 TCNT1레지스터와 출력비교 레지스터 OCR1A~B의 값을 비교하여 같으면 OCF1A~B는 1로 세트되고 출력비교 인터럽트가 요청된다. 인터럽트가 서비스되기 시작하면 자동적으로 클리어된다.
- 비트2(TOV1 : Timer Overflow Flag 1) : 타이머1에서 오버플로가 발생하면(0xFFFF까지 세고 0x0000으로 넘어가게 되면) 이 TOV1는 1로 세트되면서 오버플로 인터럽트가 요청된다. 인터럽트가 서비스되기 시작하면 자동적으로 클리어된다. Phase correct PWM 모드에서는 타이머1이 0x00에서 계수방향을 바꿀 때 TOV1가 세트된다.

4.2.2.8. ETIFR 레지스터

ETIFR(Extenede Timer Interrupt Frag Register) 레지스터는 타이머1,3가 발생하는 인터럽트 플랙을 저장한다. *소프트웨어적으로 강제로 클리어 하려면 1을 기록해야 한다.*

- 비트5(ICF3 : Input Capture Flag 3) : 타이머3의 입력캡쳐신호 또는 아날로그 비교기로부터의 신호에 의해 캡쳐 동작이 수행될 때 1로 세트되고 입력캡쳐 인터럽트가 요청된다. ICR3 레지스터가 TOP 값으로 사용되는 동작 모드에서 TCNT3의 값이 TOP과 같아질 때 1로 세트되고 인터럽트가 요청된다. 인터럽트가 서비스되기 시작하면 자동적으로 클리어된다.
- 비트4,3,1(OCF3A,B,C: Output Compare match Flag 3 A,B,C) : 타이머3의 TCNT3레지스터와 출력비교 레지스터 OCR3A~C의 값을 비교하여 같으면 OCF3A~C는 1로 세트되고 출력비교 인터럽트가 요청된다. 인터럽트가 서비스되기 시작하면 자동적으로 클리어된다.
- 비트2(TOV3 : Timer Overflow Flag 3) : 타이머3에서 오버플로가 발생하면(0xFFFF까지 세고 0x0000으로 넘어가게 되면) 이 TOV3는 1로 세트되면서 오버플로 인터럽트가 요청된다. 인터럽트가 서비스되기 시작하면 자동적으로 클리어된다. Phase correct PWM 모드에서는 타이머1이 0x00에서 계수방향을 바꿀 때 TOV3가 세트된다.
- 비트0(OCF1C: Output Compare match Flag 1 C) : 타이머1의 TCNT1레지스터와 출력비교 레지스터 OCR1C의 값을 비교하여 같으면 OCF1C는 1로 세트되고 출력비교 인터럽트가 요청된다. 인터럽트가 서비스되기 시작하면 자동적으로 클리어된다.

4.2.2.9. 타이머1, 3의 일반 모드

- 설정 : TCCRxA~B의 WGMx3~0 = 00으로 설정한다.
- 동작 : 타이머x는 항상 상향 카운터로만 동작한다. TCNTx의 값이 0xFFFF에서 0으로 바뀌는 순간 TOVx비트가 세트되며 오버플로 인터럽트가 발생이 요청된다. 출력비교 인터럽트를 일반모드에서 사용할 수도 있으나

FAST PWM모드를 사용하는 것이 좋다.

- 시간간격 설정 : 오버플로 인터럽트가 요청될 때 마다 인터럽트를 발생시켜 TCNTx의 초기값 T_0을 적당하게 설정하면 원하는 시간 간격을 얻을 수 있다. TCCRxB의 비트2~0(CSx2~0)값에 따라 1,8, 64, 256, 1024가 될 수 있으므로 클럭을 fMHz로 사용하는 경우 분주비를 N이라 하면 다음에 의해 주기 T가 결정된다. 즉 오버플로 인터럽트가 요청되는 시간간격은 T이다.

$$T[\mu\sec] = N \times (65536 - T_0)/f$$

4.2.2.10. 타이머1, 3의 CTC모드

- 설정 : TCCRxB레지스터의 WGMx3~0 을 4 또는 12로 설정한다.
- 동작 : CTC(Clear Timer on Compare match)모드의 4번 모드에서는 TCNTx의 값이 OCRxA레지스터에 설정한 값과 일치되면 그 다음 클럭 사이클에서 TCNTx의 값이 0으로 클리어된다. 이때 OCFxA비트가 세트되며 출력비교 인터럽트 발생이 요청된다. 이때 인터럽트를 발생시켜 OCRxA의 값을 바꾸면 주기를 원하는 대로 변경할 수 있다. OCx단자로 하여금 출력파형을 발생하도록 할 수 있는데 COMxn1~0을 01로 설정하고 OCRxA레지스터 값을 바꿔가면서 출력비교에 의해 OCxn의 신호를 토글하는 경우는 그림 86과 비슷하다(여기에서 x=1,3, n=A,B,C). 12번 모드에서는 TCNTx의 값이 입력캡춰 레지스터 ICRx에 설정한 값과 일치되면 그 다음 클럭 사이클에서 TCNTx의 값이 0으로 클리어된다. 이때 ICFx비트가 세트되며 입력캡춰 인터럽트 발생이 요청된다. 이때 인터럽트를 발생시켜 ICRx의 값을 바꾸면 주기를 원하는 대로 변경할 수 있다. OCxn단자로 하여금 출력파형을 발생하도록 할 수 있는데 COMxn1~0을 01로 설정하고 OCRxn레지스터 값을 바꿔가면서 출력비교에 의해 OCxn의 신호를 토글하는 경우는 그림 86과 비슷하다.
- 주기 설정 : COMxn1~0을 01로 설정하고 OCRxA나 ICRx레지스터 값을 어떤 일정한 값 CR0로 놓고 출력비교에 의해 OCxn의 신호를 토글하는 경우 TCCRxB의 비트2~0(CSx2~0)값에 따라 1,8, 64, 256, 1024의 분주비로 분주되므로 클럭을 fMHz로 사용하고 분주비를 N이라 하면 다음에 의해 주기 T가 결정된다. 즉 오버플로 인터럽트가 요청되는 시간간격은 $T/2$이다.

$$T[\mu\sec] = 2N(1 + CR0)/f$$

4.2.2.11. 타이머 1, 3의 고속 PWM 모드

- 설정 : TCCRxB레지스터의 WGMx3~0 을 5,6,7,14,15로 설정한다.
- 동작 : TCNTx는 반복적으로 업카운팅하며 항상 0x0000~TOP의 값을 갖는다. TOP의 값은 모드 번호에 따라 0x00FF, 0x1FF, 0x03FF, ICRx, OCRxA을 갖는다(그림 90참조). TCNTx의 값이 TOP 값과 일치되면 그 다음 사이클에서 0으로 클리어되며 OCFxA비트가 세트되어 출력비교 인터럽트가 요청되거나 ICFx비트가 세트되어 입력캡춰 인터럽트가 요청되거나 TOVx비트가 세트되어 오버플로 인터럽트가 요청될 수 있으며 OCRxA와 TOP 값을 업데이트 할 수 있다. COMxn1~0을 10으로 세팅한 경우 TCNTx의 값은 ICRx나 OCRxn값과 같아지면 OCxn의 값은 0으로 클리어 된다. 그리고 TCNTx는 업카운팅을 계속하여 TOP까지 증가했다가 0으로 바뀌는 순간 OCxn는 1로 세트된다. 인터럽트가 요청될 때 인터럽트를 발생시켜 TOP 값이나 OCRxn의 값을 바꾸어 듀티비를 제어할 수 있다. 레귤레이터, 정류 및 DA 변환에 적합하다. 특히 고주파를 사용하기 때문에 코일, 콘덴서 등 외부소자가 필요 없어 비용을 절감시킬 수 있다.
- 주기설정 : 0~TOP값을 업카운팅하므로 TCCRxB의 비트2~0(CSx2~x0)값에 따라 1,8, 64, 256, 1024의 분주비로 분주되므로 클럭을 fMHz로 사용하고 분주비를 N이라 하면 다음에 의해 파형의 주기 T가 결정된다. 즉 오버플로 인터럽트가 요청되는 시간간격은 T이다.

$$T[\mu\sec] = N \times (1 + \mathrm{TOP})/f$$

4.2.2.12. 타이머 1, 3의 Phase Correct PWM 모드

- 설정 : TCCRxB레지스터의 WGMx3~0 을 1, 2, 3, 10, 11로 설정한다.
- 동작 : TCNTx는 업카운팅하여 TOP으로 증가하다가 다운카운팅으로 0x0000으로 감소를 반복한다. TOP의

값은 모드 번호에 따라 0x00FF, 0x1FF, 0x03FF, ICRx, OCRxA을 갖는다(그림 90참조). TCNTx가 TOP값에 도달하면 OCRxn, TOP의 값이 경신되고 OCFxA비트가 세트되어 출력비교 인터럽트가 요청되거나 ICFx비트가 세트되어 입력캡춰 인터럽트가 요청될 수 있다. TCNTx가 0x0000에 도달하면 TOVx비트가 세트되어 오버플로 인터럽트가 요청될 수 있다. COMxn1~0을10으로 세팅한 경우 TCNTx의 값이 OCRxn레지스터에 설정한 값과 일치되면 OCxn의 값은 상향 카운팅의 경우에는 0으로 하향의 경우에는 1로 세트된다. 그리고 TCNTx는 업카운팅을 계속하여 TOP에 도달하면 하향카운팅이 시작된다(그림 88참조). PC PWM모드에서는 OCRxn 레지스터에 이중 버퍼링 기능이 있어 PWM 주기를 변경하기 위해 OCRxn 레지스터에 새로운 값을 기록하더라도 즉시 변경되지 않고 TCNTxn이 TOP에 도달하면 갱신된다(그림 88참조). OCRxn레지스터 값과 듀티비가 비례함을 그림 88에서 알 수 있다.

- 주기 설정 : 0~TOP값을 업/다운 카운팅하므로 TCCRxB의 비트2~0(CSx2~x0)값에 따라 1,8, 64, 256, 1024의 분주비로 분주되므로 클럭을 fMHz로 사용하고 분주비를 N이라 하면 다음에 의해 주기 T가 결정된다. 즉 오버플로 인터럽트가 요청되는 시간간격은 T이다.

$$T[\mu\mathrm{sec}] = 2N \times \mathrm{TOP}/f$$

4.2.2.13. 타이머 1, 3의 Phase and Frequency Correct PWM 모드

- 설정 : TCCRxB레지스터의 WGMx3~0 을 8,9로 설정한다.
- 동작 : PC PWM모드와 거의 똑같고 다른 점은 TOP의 값이 모드 번호에 따라 ICRx, OCRxA 만을 갖는다는 것이고 새로운 값으로의 경신이 TCNTx가 0일 때 이루어진다는 것이다(그림 90참조).TCNTx는 업카운팅하여 TOP으로 증가하다가 다운카운팅으로 0x0000으로 감소를 반복한다. TCNTx가 TOP값에 도달하면 OCFxA비트가 세트되어 출력비교 인터럽트가 요청되거나 ICFx비트가 세트되어 입력캡춰 인터럽트가 요청될 수 있다. TCNTx가 0x0000에 도달하면 OCRxn, TOP의 값이 경신되고 TOVx비트가 세트되어 오버플로 인터럽트가 요청될 수 있다. COMxn1~0을10으로 세팅한 경우 TCNTx의 값이 OCRxn레지스터에 설정한 값과 일치되면 OCxn의 값은 상향 카운팅의 경우에는 0으로 하향의 경우에는 1로 세트된다. 그리고 TCNTx는 업카운팅을 계속하여 TOP에 도달하면 하향카운팅이 시작된다(그림 88참조).
- 주기설정 : Phase Correct PWM과 동일.

4.2.2.14. 출력비교 변조

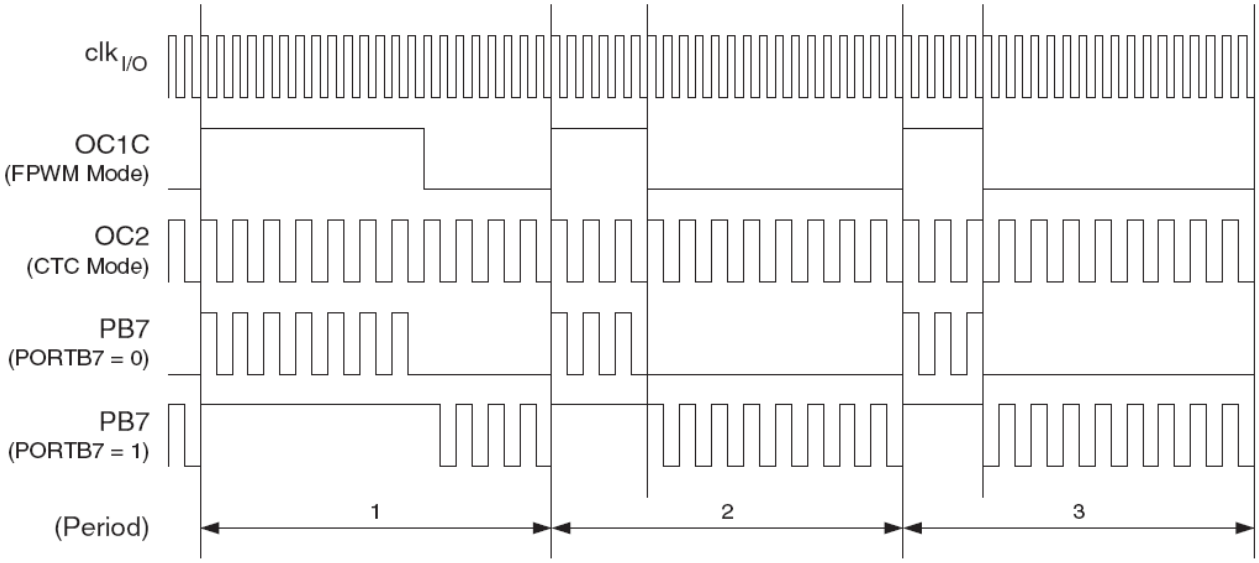


그림 91. 타이머1,2를 이용한 출력비교 변조의 예

- 설정 : 타이머1용 출력비교 설정을 위한 TCCR1A 레지스터의 비트 3~2(COM1C1~0)를 10 이나 11로 설정하여 출력비교가 이루어지도록 하는 동시에 타이머2용 출력비교 설정을 위한 TCCR2 레지스터의 비트 5~4(COM21~0)를 01~11로 설정하여 출력비교가 이루어지도록하고 DDRB.7=1로 한다.

● 동작 : PORTB.7의 값을 1로 세트시키면 OC1C와 OC2의 신호를 OR시킨 것이 17번핀(PB7, OC1C, OC2)에서 출력되고 0으로 하면 0C1C와 OC2를 AND 시킨 신호가 17번핀에서 출력된다.

4.2.3. 실험24 : 타이머를 이용한 LED 점멸1

● 예제 : 타이머0의 일반모드를 이용하여 10밀리초 간격으로 오버플로 인터럽트를 발생시켜 PORTC에 연결된 LED가 1칸씩 왼쪽으로 회전시키며 점멸시킨다.

● 회로도 : 그림 80의 SW_CON을 그림 79의 D_CON에 연결하고 실험할 때는 LEDSW를 닫고 한다.

● 프로그램 작성시 유의점 : 디버깅시 [Auto Step]를 누르고 IO View창의 TCNT0와 PORTC의 변화를 관찰하라. 그림 83과 같이 타이머0~3의 오버플로 인터럽트는 17, 15, 11, 30번이다. CodeVisoinAVR에서 인터럽트 번호에 따른 이름을 TIM0_OVF~TIM3_OVF로 mega128.h에 정의해 놓았다. 이를 이용해도 된다. 시뮬레이션에서는 10밀리초의 간격으로 회전시키면 시간이 너무 많이 걸리므로 줄여서 구현하라.

● 프로그램 예 :

```
#include <mega128.h>
unsigned char led = 0xfe;
unsigned int    cnt;
interrupt [17] void timerint0(void){   // interrupt [TIM0_OVF] void timerint0(void)로 해도 됨
      if(cnt++ >= 9999){              // 10000*1 = 10000 usec
      cnt = 0;
      if(led!=0x7f) led = (led << 1) | 0x01;    // 비트7을 안켰으면 1비트씩 좌로 시프트하고 빈자리에는 1을 채운다.
      else led = 0xfe;     }                    // 비트7을 켰으면 다시 처음으로 와서 비트 0을 켠다.
      TCNT0 = 240;        // 재정의 1usec = 1*(256-240)/16          16MHz와 분주비 1사용시
      PORTC = led;
}
void main(void){
      DDRC = 0xff; // 포트C를 출력으로
      PORTC = led;
      SREG |= 0x80;       //인터럽트 전체 허용
      TIMSK |= 1;         // 타이머0 오버플로 인터럽트 개별허용
      TCCR0 |= 1;         // 분주비를 1로 설정
      TCNT0 = 240;
      while(1);
}
```

● 과제 : ①동일한 일을 타이머1~3을 이용하여 수행하라. ② 1초 간격으로 LED가 증가하는 형태로 켜지도록 프로그램하라.

4.2.4. 실험25 : 타이머를 이용한 LED 점멸2

● 예제 : 타이머1의 일반모드를 이용하여 오버플로 인터럽트를 발생시키지 말고 100밀리초 간격으로 PORTC에 연결된 LED가 1칸씩 왼쪽으로 회전시키며 점멸시킨다.

● 회로도 : 그림 80의 SW_CON을 그림 79의 D_CON에 연결하고 실험할 때는 LEDSW를 닫고 한다.

● 프로그램 작성시 유의점 : 타이머1는 초기값에서 시작하여 0xFFFF까지 다 세면(오버플로가 발생하면) TOV1가 1로 세트된다. while문을 이용하여 TOV1가 세트되는 것을 감시하여 인터럽트 발생없이 PORTC에 연결된 LED를 왼쪽으로 회전시키며 점멸시킨다. 인터럽트를 발생시키지 않았기 때문에 TOV1는 자동으로 클리어되지 않기 때문에 오버플로가 발생할 때마다 클리어 시켜야 됨을 유의해야 한다. *인터럽트 플랙의 클리어는 1을 기록해야 된다.* 시뮬레이션에서는 100밀리초의 간격으로 회전시키면 시간이 너무 많이 걸리므로 줄여서 구현하라.

● 프로그램 예 :

```
#include <mega128.h>
unsigned char led = 0xfe;
```

```
void delay_i_ms(unsigned int i){
    do{
    while(!(TIFR & 0x04));          // TOV1 가 세트되었는지 확인
    TIFR |= 0x04;       /* 인터럽트 발생없이 타이머 사용했으므로 자동적으로 클리어되지 않으므로 강제로 클리어 */
    TCNT1H = (65536-16000)/256;  TCNT1L = (65536-16000) % 256;  /* 재정의 , 1msec 초를 얻기 위한 시간정수*/
    i--;
    } while(i);
}
void main(void){
    DDRC = 0xff; // 포트C를 출력으로
    TCCR1B |= 1;       // 분주비를 1로 설정
    TCNT1H = (65536-16000)/256;  TCNT1L = (65536-16000) % 2;
    for(;;){
    PORTC = led;
    if (led != 0x7f) led = (led << 1 ) | 0x01; else led = 0xfe;
    delay_i_ms(100); }              /*  100*1 msec 시간 지연 */
}
```

- 과제 : ①동일한 일을 타이머0,2,3을 이용하여 수행하라. ② 0.1초 간격으로 LED가 증가하는 형태로 켜지도록 프로그램하라.

4.2.5. 실험26 : 타이머를 이용한 LED 점멸3

- 예제 : 타이머2의 CTC 모드를 이용하여 500밀리초 간격으로 출력비교 인터럽트를 발생시켜 PORTC에 연결된 LED가 1칸씩 왼쪽으로 회전시키며 점멸시킨다.
- 회로도 : 그림 80의 SW_CON을 그림 79의 D_CON에 연결하고 실험할 때는 LEDSW를 닫고 한다.
- 프로그램 작성시 유의점 : 그림 83과 같이 출력비교 인터럽트는 16(타이머0), 13(타이머1A), 14(타이머1B) 25(타이머1C), 10(타이머2), 27(타이머3A), 28(타이머3B), 29(타이머3C)번이다. CodeVisoinAVR에서 인터럽트 번호에 따른 이름을 TIMx_COMPn(여기에서 x는 0~3, n은 A,B,C)로 mega128.h에 정의해 놓았다. 이를 이용해도 된다. 시뮬레이션에서는 500밀리초의 간격으로 회전시키면 시간이 너무 많이 걸리므로 줄여서 구현하라.
- 프로그램 예 :

```
#include <mega128.h>
unsigned char led = 0xfe;
unsigned int   cnt;
interrupt [10] void timerint(void){  // interrupt [TIM2_COMP] void timerint(void)로 해도 됨
    if(cnt++ >= 49999){             // 50000*10 = 500 msec
    cnt = 0;
    if(led!=0x7f) led = (led << 1) | 0x01;   // 비트7을 안켰으면 1비트씩 좌로 시프트하고 빈자리에는 1을 채운다.
    else led = 0xfe;     }                   // 비트7을 켰으면 다시 처음으로 와서 비트 0을 켠다.
    PORTC = led;
}
void main(void){
    DDRC = 0xff; // 포트C를 출력으로
    PORTC = led;
    SREG |= 0x80;      //인터럽트 전체 허용
    TIMSK |= 0x80;     // 타이머2 출력비교 인터럽트 개별허용
    TCCR2 |= 1;        // 분주비를 1로 설정
    TCCR2 |= 0x20;     // CTC mode
    OCR2 = 159;        // 10usec = 1*(1+159)/16          16MHz와 분주비 1사용시
    for(;;);
}
```

- 과제 : ①동일한 일을 타이머0,1,3을 이용해서 수행하라. ② 동일한 일을 실험 25와 비슷한 요령으로 인터럽트 발생없이 OCF2를 감시하면서 LED를 회전시키면 점멸시키는 프로그램을 구현해보라.

4.2.6. 실험27 : 타이머를 이용한 LED 점멸4

- 예제 : 타이머3의 Fast PWM 모드를 이용하여 500밀리초 간격으로 오버플로 인터럽트 플랙을 감시하여 인터럽트 발생없이 PORTC에 연결된 LED가 1칸씩 왼쪽으로 회전시키며 점멸시킨다.
- 회로도 : 그림 80의 SW_CON을 그림 79의 D_CON에 연결하고 실험할 때는 LEDSW를 닫고 한다.
- 프로그램 작성시 유의점 : 타이머3는 초기값에서 시작하여 TOP(모드번호에 따라 다른 값을 갖는다) 값까지 다 세면(오버플로가 발생하면) TOV3가 1로 세트된다. while문을 이용하여 TOV3가 세트되는 것을 감시하여 인터럽트 발생없이 PORTC에 연결된 LED를 왼쪽으로 회전시키며 점멸시킨다. 인터럽트를 발생시키지 않았기 때문에 TOV3는 자동으로 클리어되지 않기 때문에 오버플로가 발생할 때마다 클리어 시켜야 됨을 유의해야 한다. *인터럽트 플랙의 클리어는 1을 써주어야 한다.* 시뮬레이션에서는 500밀리초의 간격으로 회전시키면 시간이 너무 많이 걸리므로 줄여서 구현하라.
- 프로그램 예 :

```
#include <mega128.h>
unsigned char led = 0xfe;
void delay_i_ms(unsigned int i){
      do{
      while(!(ETIFR & 0x04));         // TOV3 가 세트되었는지 확인
      ETIFR |=0x04;       /* 인터럽트 발생없이 타이머 사용했으므로 강제로 클리어 */
      i--;
      } while(i);
}
void main(void){
      DDRC = 0xff; // 포트C를 출력으로
      TCCR3B |= 1;                  // 분주비를 1로 설정
      TCCR3A |= 0b00000011;         // 고속 PWM모드의 15번으로 설정 하여 TOP를 OCR3A로 설정
      TCCR3B |= 0b00011000;         // 고속 PWM모드의 15번으로 설정 하여 TOP를 OCR3A로 설정
      ETIMSK |= 0x04;               // 타이머3 오버플로 인터럽트 개별허용
      OCR3AH = (unsigned char) 15999/256;
      OCR3AL = (unsigned char) 15999%256; // 1msec 시간지연
      for(;;){
      PORTC = led;
      if (led != 0x7f) led = (led << 1 ) | 0x01; else led = 0xfe;
      delay_i_ms(500); }            /*  500*1 msec 시간 지연 */
}
```

- 과제 : ①동일한 일을 타이머1을 이용해서 수행하라. ② 1초 간격으로 LED가 증가하는 형태로 켜지도록 프로그램하라. ③ 인터럽트를 사용해서 동일한 일을 하도록 구현해보라.

4.2.7. 실험28 : 타이머를 이용한 LED 점멸5

- 예제 : 타이머1의 Phase Correct PWM 모드를 이용하여 500밀리초 간격으로 오버플로 인터럽트을 사용하여 PORTC에 연결된 LED가 1칸씩 왼쪽으로 회전시키며 점멸시킨다.
- 회로도 : 그림 80의 SW_CON을 그림 79의 D_CON에 연결하고 실험할 때는 LEDSW를 닫고 한다.
- 프로그램 작성시 유의점 : 타이머1는 초기값에서 시작하여 모드번호에 따라 다른 값을 갖는 TOP 값까지 상향 카운팅을 하다가 다시 하향카운팅을 하여 0에 도달하면 TOV1가 1로 세트된다. 시뮬레이션에서는 500밀리초의 간격으로 회전시키면 시간이 너무 많이 걸리므로 줄여서 구현하라.
- 프로그램 예 :

```
#include <mega128.h>
unsigned char led = 0xfe;
unsigned int   cnt;
```

```
interrupt [15] void timerint(void){  // interrupt [TIM1_OVF] void timerint(void)로 해도 됨
    if(cnt++ == 499){              // 500*1 = 500 msec
    cnt = 0;
    if(led!=0x7f) led = (led << 1) | 0x01;   // 비트7을 안켰으면 1비트씩 좌로 시프트하고 빈자리에는 1을 채운다.
    else led = 0xfe;     }                   // 비트7을 켰으면 다시 처음으로 와서 비트 0을 켠다.
    PORTC = led;
}
void main(void){
    DDRC = 0xff; // 포트C를 출력으로
    PORTC = led;
    SREG |= 0x80;      //인터럽트 전체 허용
    TCCR1B |= 1;                // 분주비를 1로 설정
    TCCR1A |= 0b00000011;       // PC PWM모드의 11번으로 설정 하여 TOP를 OCR1A로 설정
    TCCR1B |= 0b00010000;       // PC PWM모드의 11번으로 설정 하여 TOP를 OCR1A로 설정
    TIMSK |= 0x04;      // 타이머1 오버플로 인터럽트 개별허용과 타이머 스타트
    OCR1A = 8000;       // 1msec 시간지연
    for(;;);
}
```

- 과제 : ①동일한 일을 타이머3를 이용해서 수행하라. ② 1초 간격으로 LED가 증가하는 형태로 켜지도록 프로그램하라. ③ 인터럽트를 사용하지 말고 인터럽트 플랙을 감시하여 동일한 일을 하도록 구현해보라.

4.2.8. 실험29 : 워치독 타이머

- 예제 : 타이머0의 일반모드를 이용하여 1초 간격으로 오버플로 인터럽트를 발생시켜 PORTC에 연결된 LED로 0~9를 반복적으로 쓴다. 워치독 타이머의 주기기 완료되기 전에 워치독 타이머를 리셋하지 않으면 워치독 리셋이 발생하는 것을 배워본다.
- 회로도 : 그림 80의 SW_CON을 그림 79의 D_CON에 연결하고 실험할 때는 LEDSW를 닫고 한다.
- 프로그램 작성시 유의점 : 시뮬레이션에서는 1초의 간격으로 점멸시키면 시간이 너무 많이 걸리므로 줄여서 구현하라.
- 프로그램 예 :

```
#include <mega128.h>
unsigned char led = 0x00;
unsigned int   cnt;
interrupt [17] void timerint0(void){  // interrupt [TIM0_OVF] void timerint0(void)로 해도 됨
    if(cnt++ >= 999){              // 1000*1 = 1 sec
    cnt = 0;
    if(led++==9) led=0; }                  // 9이면 다시 처음으로 와서 비트 0을 켠다.
    TCNT0 = 6;          // 재정의 1msec = 64*(256-6)/16          16MHz와 분주비 64사용시
    PORTC = ~led;
    #asm("wdr");
}
void main(void){
    DDRC = 0xff; // 포트C를 출력으로
    PORTC = ~led;
    SREG |= 0x80;      //인터럽트 전체 허용
    TIMSK |= 1;         // 타이머0 오버플로 인터럽트 개별허용
    TCCR0 |= 4;         // 분주비를 64로 설정
    TCNT0 = 6;
    WDTCR = 0x18;
    WDTCR = 0x0f;      //워치독 타이머 타임아웃 주기 약 2초로 설정.
    while(1);
}
```

- 과제 : ① #asm("wdr")를 없애고 수행해보라. ② 동일한 일을 타이머1~3을 이용하여 수행하라.

4.2.9. 실험30 : 타이머를 이용한 카운터

- 예제 : 타이머2를 카운터로 사용해 T2(PD7, 32번핀)에 연결된 스위치가 눌릴 때마다 PORTC에 연결된 LED가 증가형태로 점멸한다.
- 회로도 : 그림 80의 SW_CON을 그림 79의 D_CON에 연결하고 실험할 때는 LEDSW를 닫고 한다. 또한 그림 80의 SW8을 누르면서 실험한다.
- 프로그램 작성시 유의점 : 스위치가 눌리면 하강에지가 발생한다. 이때 입력캡쳐 인터럽트 플랙 ICF1을 클리어하고 스위치가 해제될 때 상승에지가 발생하므로 시뮬레이션에서는 1초의 간격으로 점멸시키면 시간이 너무 많이 걸리므로 줄여서 구현하라.
- 프로그램 예 :

```
#include <mega128.h>
#include <delay.h>
void main(void){
      DDRC = 0xff;        // 포트C를 출력으로
      TCCR0 |= 6;         // 하강에지에서 카운트
      TCNT2 = 0;          // TCNT2 초기화
      while(1){  PORTC = ~TCNT2;  delay_ms(5);}
}
```

- 과제 : ① 실제 구현 했을 때 한 번 누를 때마다 하나씩 이동하지 않고 여러 개가 이동하는 현상이 발생할 수도 있을 것이다. 이는 실험 16에서도 이야기한 채터링에 의한 현상이다. delay_ms(5)의 값을 변화시켜 가면서 실험해 보고 적정한 시간 지연을 찾아 보아라. ② 그림 80의 회로 대신 그림84의 회로를 이용하되 시간지연을 작게 해서 실험해보고 비교해보라. ③ 동일한 일을 타이머1,3을 사용해서 구현해 보라.

4.2.10. 실험31 : 펄스 폭 측정

- 예제 : IC1(PD4, 29번핀)에 연결된 스위치가 눌린 시간을 즉 펄스 폭을 측정하여 PORTC에 연결된 LED에 msec 단위로 표시한다.
- 회로도 : 그림 80의 SW_CON을 그림 79의 D_CON에 연결하고 실험할 때는 LEDSW를 닫고 한다. 또한 그림 80의 SW5을 누르면서 실험한다. 분주비를 64로 한다. 결국 4 usec의 해상도로 최대 64*65536/16 usec = 256msec까지 펄스폭을 잴 수 있다. 대충 ICR1H값만큼의 msec 로 펄스폭이 구해짐을 알 수있다.
- 프로그램 예 :

```
#include <mega128.h>
#include <delay.h>
void main(void){
unsigned char pwidth;
      DDRC = 0xff;        // 포트C를 출력으로
      while(1){
      TCCR1B |= 0x03;             // 64분주 타이머 시작
      TIFR |= 0x20;       // ICF1 클리어
      while(!(TIFR & 0x20));      // 하강에지가 검출될 때까지 기다린다.
      TIFR  |= 0x20;              // 하강에지가 검출되었으므로 ICF1을 클리어
      TCNT1 = 0;
      TCCR1B |=0x40;              // 상승에지에서 캡춰하도록 설정
      TCCR1B |= 0x03;             // 64분주
      while(!(TIFR & 0x20));      // 상승에지가 검출될 때까지 기다린다.
      pwidth = (unsigned char) (( 256*ICR1H + ICR1L )*4 /1000);
      PORTC = ~pwidth;  delay_ms(50);}
}
```

- 과제 : 타이머 3를 사용해서 구현해 보라.

4.3. 인터럽트와 타이머 복합 실험

4.3.1. 실험32 : LED 점멸 속도 조절

- 예제 : PORTC에 연결된 LED가 0.1초 간격으로 왼쪽으로 회전하며 점멸하다가 INT0(PORTD.0, 핀25)에 연결된 스위치를 누르면 상승 에지 트리거에 의한 인터럽트가 발생하여 속도가 느려지고 INT1(PORTD.1, 핀26)에 연결된 푸시버튼을 누르면 회전 속도가 빨라진다.
- 프로그램 작성시 유의점 : 일정한 시간 간격을 얻기 위해 타이머0의 일반모드 인터럽트를 사용한다. 최대로 느린 회전속도 간격은 0.15초이고 빠른 간격은 0.05초로 한다. 시뮬레이션에서는 너무 느리므로 타이머 인터럽트 서비스 루틴의 시간지연을 조정 한다. 실제 하드웨어 구현할 때는 제거한다.
- 회로도 : 그림 80의 SW_CON을 그림 79의 D_CON에 연결하고 실험할 때는 LEDSW를 닫고 한다. 또한 그림 80의 SW1, SW2를 누르면서 실험한다.
- 프로그램 예 :

```
#include <mega128.h>
unsigned char led=0xfe;
unsigned int cnt, delaytime, step, delaymax, delaymin;
interrupt [2] void exint0(void){  // interrupt [EXT_INT0] void exint0(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
    if( delaytime + step <= delaymax ) delaytime += step;
}
interrupt [3] void exint1(void){  // interrupt [EXT_INT1] void exint1(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
    if( delaytime - step >= delaymin ) delaytime -= step;
}
interrupt [17] void timerint0(void){  // interrupt [TIM0_OVF] void timerint0(void)로 해도 됨
    if(cnt-- == 1){                 //
    cnt = delaytime;
    if(led!=0x7f) led = (led << 1) | 0x01;   else led = 0xfe;    }
    TCNT0 = 6; PORTC = led;
}
void main(void){
    DDRC = 0xff; // 포트C를 출력으로
    PORTC = led;
    SREG |= 0x80;      //인터럽트 전체 허용
    EIMSK |= 0b00000011;        // INT0, 1 개별 허용
    EICRA |= 0x0f;              // INT0, 1 상승에지 트리거
    TIMSK |= 1;         // 타이머0 오버플로 인터럽트 개별허용
    TCCR0 |= 4;         // 분주비를 64로 설정
    TCNT0 = 6;          // 64*(256-6)/16=1000usec = 1msec
    delaytime = 100;    cnt = delaytime;                    /* 점멸회전주기를 100*0.001 = 0.1초로 초기화 */
    delaymax = 1000;    delaymin = 10;                /*점멸회전주기의 최대값과 최소값을 1초와 0.01초로 초기화 */
    step = 10;                                              /* 주기 변경 폭을 10으로 초기화 */
    while(1);
}
```

- 과제 : ①동일한 일을 수행하되 타이머0 대신에 타이머1~3을 사용하는 프로그램을 작성하라. ② INT3(PORTD.2, 핀27))에 연결된 푸시버튼를 누르면 속도가 변하지 않고 LED 점멸 회전 방향이 바뀌는 프로그램을 작성하라.

4.3.2. 실험33 : 스톱 워치

- 예제 : INT0(PORTD.0, 핀25)에 연결된 스위치를 누르면 상승 에지 트리거에 의한 인터럽트가 발생하여 타이머 1의 카운팅이 정지되고 다시 한번 누르면 카운팅을 시작한다.

● 회로도 : 그림 80의 SW_CON을 그림 79의 D_CON에 연결하고 실험할 때는 LEDSW를 닫고 한다. 또한 그림 80의 SW1를 누르면서 실험한다.

● 프로그램 예 :

```
#include <mega128.h>
unsigned char led, onoff=1;
interrupt [2] void exint0(void){  // interrupt [EXT_INT0] void exint0(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
    if(onoff == 0) onoff = 1; else onff=0;
}
void delay_i_ms(unsigned int i){
    do{
    while(!(TIFR & 0x04));          // TOV1 가 세트되었는지 확인
    TIFR |= 0x04;       /* 인터럽트 발생없이 타이머 사용했으므로 자동적으로 클리어되지 않으므로 강제로 클리어 */
    TCNT1H = (65536-16000)/256;  TCNT1L = (65536-16000) % 256;  /* 재정의 , 1msec 초를 얻기 위한 시간정수*/
    i--;
    } while(i);
}
void main(void){
    DDRC = 0xff; // 포트C를 출력으로
    DDRD = 0x00; //포트D를 입력으로 디폴트 입력으로 설정되어 있으므로 필요하지는 않음
    SREG.7 = 1;         //인터럽트 전체 허용
    EIMSK |= 0x01;      // INT0 개별 허용
    EICRA |= 3;         // 상승에지 트리거
    TCCR1B |= 1;                // 타이머에 클록을 1분주로 클록 공급
    TCNT1H = (65536-16000)/256;  TCNT1L = (65536-16000) % 256;
    for(;;){
    PORTC = ~led ;
    delay_i_ms(1000); /* 1 sec 시간 지연 */
    if(onoff==0) led++;}
}
```

● 과제 : ① 타이머 0,2,3를 사용하는 프로그램을 작성하라. ② 타이머0~3의 인터럽트를 사용해서 구현해보라.

4.3.3. 실험34 : PWM 신호 발생1

● 예제 : PORTC에 연결된 LED가 INT0(PORTD.0, 핀25)에 연결된 푸쉬버튼을 누르면 점점 밝아지고 INT1(PORTD.1, 핀26)에 연결된 푸시버튼을 누르면 어두워진다.

● 프로그램 작성시 유의점 : 일정한 시간간격에서 1이 지속되는 시간을 조절하여 밝기를 조절한다. 일정한 시간간격을 얻기 위해 타이머2를 사용한다.

● 회로도 : 그림 80의 SW_CON을 그림 79의 D_CON에 연결하고 실험할 때는 LEDSW를 닫고 한다. 또한 그림 80의 SW1, SW2를 누르면서 실험한다.

● 프로그램 예 :

```
#include <mega128.h>
unsigned char led;
unsigned int    cnt, step=10, duration=500, MAX=999;
interrupt [2] void exint0(void){  // interrupt [EXT_INT0] void exint0(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
    if(MAX > duration+step ) duration += step;
}
interrupt [3] void exint1(void){  // interrupt [EXT_INT1] void exint1(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
    if( duration-step >0 ) duration -=step;
}
interrupt [10] void timerint(void){  // interrupt [TIM2_COMP] void timerint(void)로 해도 됨
    if(cnt < duration) led = 0xff;
    else led = 0x00;
    if(cnt++ >= MAX) cnt = 0;     // 1000*10 = 10 msec
```

```
}
void main(void){
      DDRC = 0xff; // 포트C를 출력으로
      SREG |= 0x80;      //인터럽트 전체 허용
      EIMSK |= 0b00000011;         // INT0, 1 개별 허용
      EICRA |= 0x0f;               // INT0, 1 상승에지 트리거
      TIMSK |= 0x80;     // 타이머2 출력비교 인터럽트 개별허용
      TCCR2 |= 1;        // 분주비를 1로 설정
      TCCR2 |= 0x20;     // CTC mode
      OCR2 = 159;        // 1usec = 1*(1+159)/16          16MHz와 분주비 1사용시
      for(;;) PORTC = led;
}
```

- 과제 : ①MAX와 duration을 10배 이상으로 하고 수행해보라. 어떤 현상이 일어나나 살피고 그 원인에 대하여 생각해보라. ② 타이머 0,1,3으로 구현해 보아라.

4.3.4. 실험35 : PWM 신호 발생2

- 예제 : 타이머의 PWM모드를 이용해서 LED의 밝기를 조절한다. INT0(PORTD.0, 핀25)에 연결된 푸쉬버튼을 누르면 점점 밝아지고 INT1(PORTD.1, 핀26)에 연결된 푸시버튼을 누르면 어두워진다.
- 프로그램 작성시 유의점 : 타이머0의 PWM신호는 OC0(14번핀, PB4)에서 출력된다. 시뮬레이션을 할 때는 타이머의 카운트가 너무 오래 걸릴 수 있으므로 분주비를 적절하게 조절해야 한다.
- 회로도 : 그림 80의 SW_CON과 그림 92의 L2를 각각 그림 79의 D_CON과 B_CON에 연결한다. 또한 그림 80의 SW1, SW2를 누르면서 실험한다.

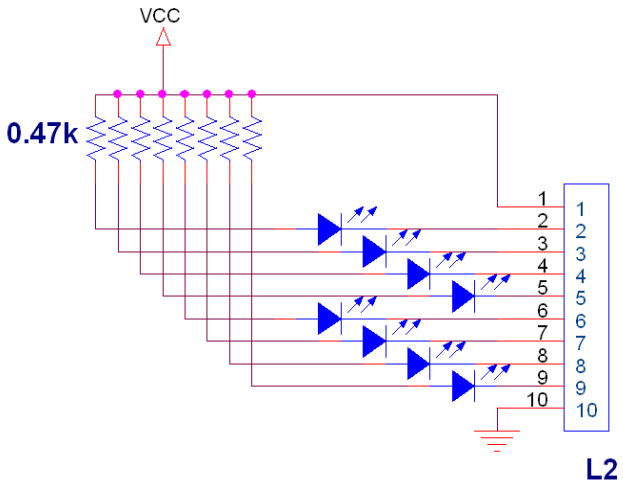


그림 92. LED배열 회로.

```
#include <mega128.h>
unsigned int step=5;
interrupt [2] void exint0(void){  // interrupt [EXT_INT0] void exint0(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
      if( OCR0 < 0xff - step ) OCR0 += step;
}
interrupt [3] void exint1(void){  // interrupt [EXT_INT1] void exint1(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
      if( OCR0  > step ) OCR0 -=step;
}
void main(void){
      DDRB.4 = 1;        // PORTB bit 4를 출력으로
      SREG |= 0x80;      //인터럽트 전체 허용
      EIMSK |= 0b00000011;         // INT0, 1 개별 허용
      EICRA |= 0x0f;               // INT0, 1 상승에지 트리거
      TCCR0 |= 6;        // 분주비를 256로 설정
```

```
    TCCR0 |= 0b01001000;        // 타이머0 FAST PWM 모드로 설정
    TCCR0 |= 0x20;              // 출력비교모드 설정
    OCR0 = 0x7F;
    for(;;);
}
```

- 과제 : ① 타이머 1~3으로 구현해 보아라.

5. 응용 실험

5.1. FND(Flexible Numeric Display)

5.1.1. FND의 개요

- 구성 : FND는 7세그먼트 표시기로도 불리며 그림 93처럼 보통 8개의 LED 중 7개의 LED를 8자형태로 배열하고 0-9까지의 숫자와 ABCDEF(HEX 값)을 표시하는 데 사용한다. 나머지 LED는 소숫점의 표시에 사용한다.
- 종류 : 모든 LED의 양극을 한데 묶은 양극공통형(CA), 음극을 묶은 음극공통형(CC)형이 있다.
- 핀배치 : 정면에서 보았을 때 왼쪽아래에서부터 핀번호가 1번으로 시작하여 반시계방향으로 배열되어 있다(그림 49 참조). 만약 CA형의 경우 1을 표시하려면 3번과 8번핀에 Vcc를 연결하고 6번과 4번핀에 0을 연결한다. 만약 CA형의 경우 3을 표시하려면 3번과 8번핀에 Vcc를 연결하고 2,4,6,7,10번핀에 0을 연결한다.
- 특성 : LED는 소형으로 발열이 거의 없고, 수명이 반영구적이며 응답속도가 빠르고 발광효율이 높고 가격이 싸며 보통 10mA 이상의 구동전류가 필요한데 직류의 저전압에서 동작이 가능하다는 등의 장점이 있어 FND도 마찬가지의 장점이 있다. 또한 16x2 캐릭터 LCD가 만원정도인데 비해 FND는 개당 수백원 꼴이라 값싼 시스템을 위한 표시장치로 쓰인다. CA형을 위한 디코딩용 구동 TTL은 7447, CC형을 위한 디코딩용 구동 TTL은 7446이 있다.

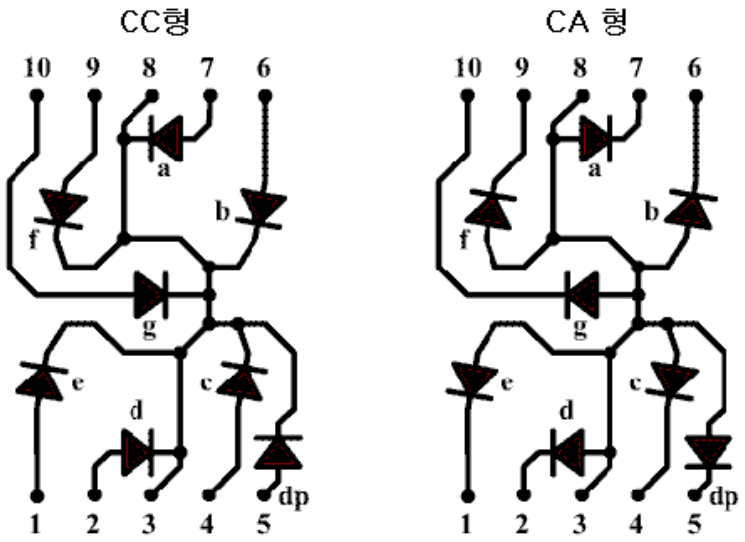


그림 93. FND의 내부구성과 핀배치

5.1.2. 실험36 : 디코더 없는 FND 구동

- 예제 : CA형의 FND를 C_CON에 직접 연결시키고 00~FFH값을 넣었을 때 패턴을 살핀다.
- 프로그램 작성시 유의점 : 너무 빨리 패턴이 지나가지 않도록 시간 지연을 준다.
- 회로도 : 그림 94의 왼쪽 디코더 없는 회로를 구현하고 FND_CON을 그림 79에 보여진 시스템의 C_CON에 연결한다.
- 프로그램 예 :

```
#include <mega128.h>
#include <delay.h>
unsigned char int;
void main(void){
```

```
    DDRC = 0xff;        // 포트C를 출력으로
    for(;;) {delay_ms(500); PORTC = led++;}
}
```

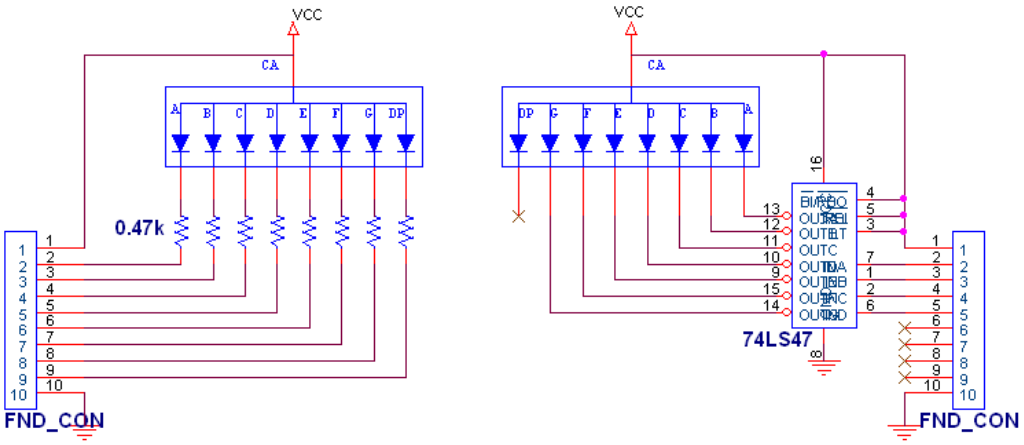


그림 94. FND 기초 실험도(왼쪽:디코더 비사용, 오른쪽:디코더 사용)

- 과제 : ① 0~9, A,B,C,D,E,F값을 표시하기 위한 입력패턴 값을 찾아라. ② 찾은 패턴값을 이용하여 0~9, A,B,C,D,E,F값을 1초 간격으로 표시하는 프로그램을 작성하라.

5.1.3. 실험37 : 디코더를 사용한 FND구동

- 예제 : CA형의 FND를 7447을 이용하여 구동하고 패턴을 살핀다.
- 프로그램 작성시 유의점 : 너무 빨리 패턴이 지나가지 않도록 시간 지연을 준다.
- 회로도 : 그림 94의 오른쪽 회로를 구현하고 FND_CON을 그림 79에 보여진 시스템의 C_CON에 연결한다.
- 프로그램 예 :

```
#include <mega128.h>
#include <delay.h>
unsigned char int;
void main(void){
    DDRC = 0xff;        // 포트C를 출력으로
    for(;;) {delay_ms(500); PORTC = led++;}
}
```

- 과제 : ① 위의 프로그램은 0x00~0xff의 오름차순으로 표시하는 것이다. 내림차순으로 표시하는 프로그램을 작성하라. ② 0~9값을 반복적으로 표시하는 프로그램을 작성하라. ③ delay_ms()를 사용하지 말고 타이머를0~3을 사용하여 시간지연을 주어 표시하는 프로그램을 작성하라.

5.1.4. 실험38 : FND 동적 구동

- 예제 : CA형의 FND0~5의 6개를 C_CON 하나만을 할애하여 동적으로 구동한다. 2msec 마다 각 FND들을 번갈아 구동시켜 잔상에 의해 6개 모두가 켜져 동작하는 것처럼 보이게 한다.
- 프로그램 작성시 유의점 : 2msec의 주기를 얻기 위해 타이머0의 일반모드 인터럽트를 이용한다. 각 FND에 표시할 데이터값은 1,2,3,4,5,6이라고 가정한다. 시뮬레이션을 수행할 때는 너무 오래 걸릴 수 있으므로 시간지연을 적절히 조절해서 한다. PORTC로 출력되는 값의 상위4비트가 데이터값이고 하위4비트로 활성화될 FND를 지정한다.
- 회로도 : 그림 95의 회로를 구현하고 FND_CON을 그림 79에 보여진 시스템의 C_CON에 연결한다.
- 프로그램 예 :

```
#include <mega128.h>
```

```
unsigned char dbuf[6], dbuf_index;
unsigned int cnt;
interrupt [17] void timerint0(void){  // interrupt [TIM0_OVF] void timerint0(void)로 해도 됨
      unsigned char tmpbuf;
      if( ++cnt == 2 ){
      cnt = 0; if( dbuf_index++ == 5 ) dbuf_index= 0; }
      tmpbuf = dbuf[dbuf_index];
      PORTC = (( tmpbuf << 4 )&0xf0) | dbuf_index;
                        /* 표시할 데이터와 FND의 위치는 tmpbuf의 상위비트와 하위비트로 */
      TCNT0 = 6;          // 재정의 1msec = 64*(256-6)/16       시뮬레이션할 때는 조절할 필요가 있음
}
void main(void){
      DDRC = 0xff; // 포트C를 출력으로
      SREG |= 0x80;       //인터럽트 전체 허용
      TIMSK |= 1;         // 타이머0 오버플로 인터럽트 개별허용
      TCCR0 |= 4;         // 분주비를 64로 설정                  시뮬레이션할 때는 조절할 필요가 있음
      TCNT0 = 6;          // 1msec = 64*(256-6)/16               시뮬레이션할 때는 조절할 필요가 있음
      dbuf[0]= 1; dbuf[1] = 2; dbuf[2]=3; dbuf[3] = 4; dbuf[4] = 5; dbuf[5] = 6;    /* 표시할 데이터 초기화 */
      for(;;);                                  /* 인터럽트 발생할 때까지 대기 */
}
```

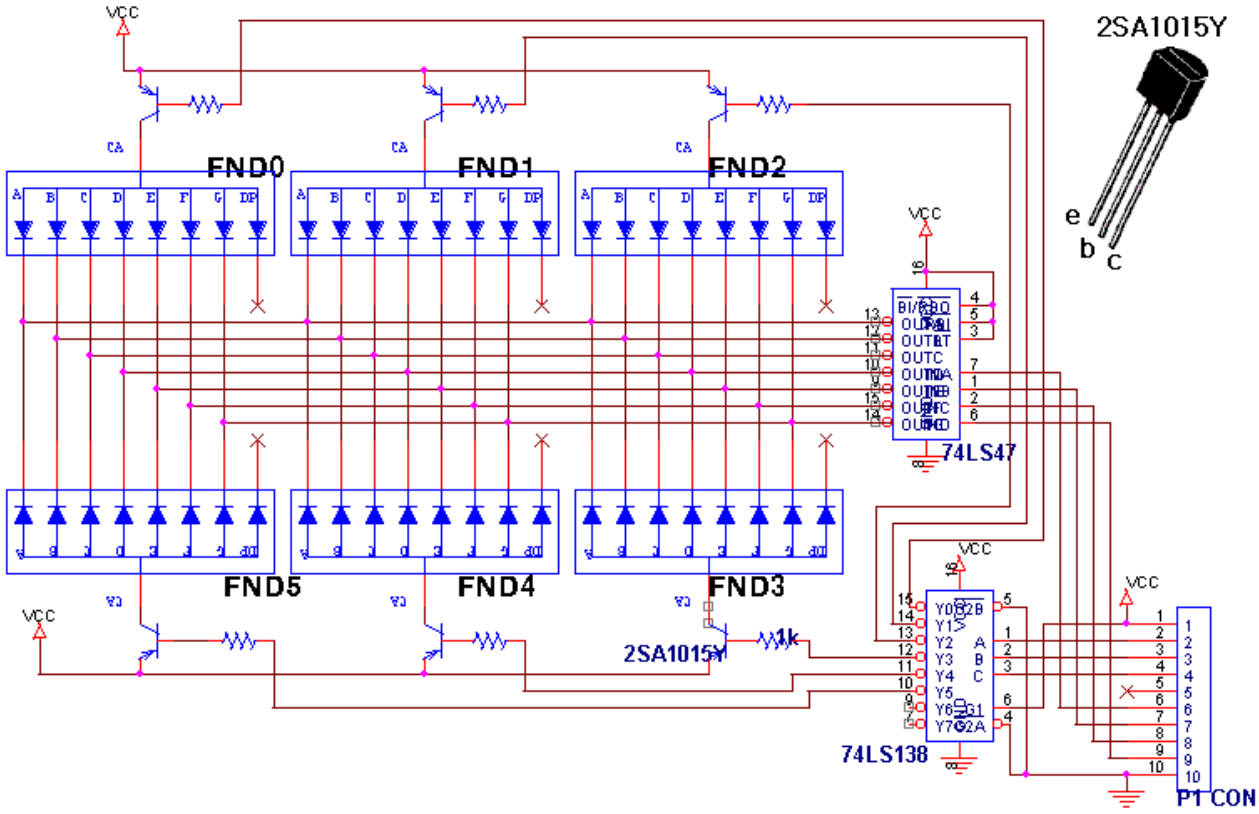


그림 95. FND의 동적 구동을 위한 회로도

- 과제 : ①타이머0 대신 타이머1~3을 사용하고 주기를 1 msec로 하여라. ② 주기를 12 msec 이상 예로 40 msec로 하여 하드웨어를 구현해보고 어떤 현상이 나타나나 보아라.

5.1.5. 실험39 : 시계

- 예제 : CA형의 FND5에 시간의 10의 자리값, FND4에 시간의 1의 자리값, FND3에 분의 10의 자리값, FND2에 분의 1의 자리값, FND1에 초의 10의 자리값, FND0에 초의 1의 자리값을 표시하는 시계를 만든다.
- 프로그램 작성시 유의점 : 타이머0의 일반모드를 사용하였고 1 msec를 주기로 FND를 동적으로 구동한다.

● 회로도 : 그림 95의 회로를 구현하고 FND_CON을 그림 79에 보여진 시스템의 C_CON에 연결한다.

● 프로그램 예 :

```
#include <mega128.h>
unsigned char dbuf[6], dbuf_index;
unsigned char hou, min, sec;
unsigned int cnt;
void hex2bcd(unsigned char hex, unsigned char *ptr) {          /* 0x63 이하의 헥사값을 bcd로 바꾼다 */
    *ptr = hex % 10;
    *(ptr+1) = hex / 10;
}
void watch(void){
    if( ++sec == 60) { sec = 0;
            if( ++min == 60 ) { min = 0;
                    if( ++hou == 24 ) hou = 0;
            }
    }
    hex2bcd(sec, &dbuf[0]);
    hex2bcd(min, &dbuf[2]);
    hex2bcd(hou, &dbuf[4]);
}
interrupt [17] void timerint0(void){  // interrupt [TIM0_OVF] void timerint0(void)로 해도 됨
    unsigned char tmpbuf;
    if( ++cnt == 1000 ){
    cnt = 0; watch(); }
    if( dbuf_index++ == 5 ) dbuf_index= 0;
    tmpbuf = dbuf[dbuf_index];
    PORTC = (( tmpbuf << 4 )&0xf0) | dbuf_index;
                    /* 표시할 데이터와 FND의 위치는 tmpbuf의 상위비트와 하위비트로 */
    TCNT0 = 6;          // 재정의 1msec = 64*(256-6)/16       시뮬레이션할 때는 조절할 필요가 있음
}
void main(void){
    DDRC = 0xff; // 포트C를 출력으로
    SREG |= 0x80;       //인터럽트 전체 허용
    TIMSK |= 1;         // 타이머0 오버플로 인터럽트 개별허용
    TCCR0 |= 4;         // 분주비를 64로 설정                    시뮬레이션할 때는 조절할 필요가 있음
    TCNT0 = 6;          // 1msec = 64*(256-6)/16                 시뮬레이션할 때는 조절할 필요가 있음
    for(;;);                                /* 인터럽트 발생할 때까지 대기 */
}
```

● 과제 : 타이머0 대신 타이머1~3을 사용하여라.

5.1.6. 실험40 : 스톱 워치

● 예제 : CA형의 FND5에 시간의 10의 자리값, FND4에 시간의 1의 자리값, FND3에 분의 10의 자리값, FND2에 분의 1의 자리값, FND1에 초의 10의 자리값, FND0에 초의 1의 자리값을 표시하는 시계를 만든다. INT0(PORTD.0, 핀25)에 연결된 스위치를 누르면 상승 에지 트리거에 의한 인터럽트가 발생하여 타이머 1의 카운팅이 정지되고 다시 한번 누르면 카운팅을 시작한다.

● 회로도 : 그림 95의 회로를 구현하고 FND_CON을 그림 79에 보여진 시스템의 C_CON에 연결한다. 그림 80의 SW_CON을 그림 79의 D_CON에 연결한다. 또한 그림 80의 SW1를 누르면서 실험한다.

● 프로그램 작성시 유의점 : 타이머0의 일반모드를 사용하여 10 msec를 주기로 FND를 동적으로 구동한다. 그리고 타이머2의 일반모드를 사용하여 시간을 잰다.

● 프로그램 예 :

```
#include <mega128.h>
unsigned char dbuf[6], dbuf_index;
```

```
unsigned char hou, min, sec, onoff;
unsigned int cnt0, cnt2;
void hex2bcd(unsigned char hex, unsigned char *ptr) {          /* 0x63 이하의 헥사값을 bcd로 바꾼다 */
    *ptr = hex % 10;
    *(ptr+1) = hex / 10;
}
void watch(void){
    if( ++sec == 60) { sec = 0;
            if( ++min == 60 ) { min = 0;
                    if( ++hou == 24 ) hou = 0;
            }
    }
    hex2bcd(sec, &dbuf[0]);
    hex2bcd(min, &dbuf[2]);
    hex2bcd(hou, &dbuf[4]);
}
interrupt [2] void exint0(void){  // interrupt [EXT_INT0] void exint0(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
    if(onoff == 0) onoff = 1; else onoff = 0;
}
interrupt [17] void timerint0(void){  // interrupt [TIM0_OVF] void timerint0(void)로 해도 됨
    if(onoff == 0) if( ++cnt0 == 1000 ){
    cnt0 = 0; watch(); }
    TCNT0 = 6;          // 재정의 1msec = 64*(256-6)/16       시뮬레이션할 때는 조절할 필요가 있음
}
interrupt [11] void timerint2(void){  // interrupt [TIM2_OVF] void timerint0(void)로 해도 됨
    unsigned char tmpbuf;
    if( ++cnt2 == 10 ){
    if( dbuf_index++ == 5 ) dbuf_index= 0;
    tmpbuf = dbuf[dbuf_index];
    PORTC = (( tmpbuf << 4 )&0xf0) | dbuf_index; }
                    /* 표시할 데이터와 FND의 위치는 tmpbuf의 상위비트와 하위비트로 */
    TCNT2 = 6;          // 재정의 1msec = 64*(256-6)/16       시뮬레이션할 때는 조절할 필요가 있음
}
void main(void){
    DDRC = 0xff;         // 포트C를 출력으로
    DDRD = 0x00;        //포트D를 입력으로 디폴트 입력으로 설정되어 있으므로 필요하지는 않음
    SREG.7 = 1;         //인터럽트 전체 허용
    EIMSK |= 0x01;      // INT0 개별 허용
    EICRA |= 3;         // 상승에지 트리거
    TIMSK |= 1;         // 타이머0 오버플로 인터럽트 개별허용
    TCCR0 |= 4;         // 분주비를 64로 설정                    시뮬레이션할 때는 조절할 필요가 있음
    TCNT0 = 6;          // 1msec = 64*(256-6)/16                 시뮬레이션할 때는 조절할 필요가 있음
    TIMSK |= 0x40;      // 타이머2 오버플로 인터럽트 개별허용
    TCCR2 |= 3;         // 분주비를 64로 설정                    시뮬레이션할 때는 조절할 필요가 있음
    TCNT2 = 6;          // 1msec = 64*(256-6)/16                 시뮬레이션할 때는 조절할 필요가 있음
    for(;;);            /* 인터럽트 발생할 때까지 대기 */
}
```

- 과제 : ① 타이머0과 2의 역할을 바꿔 프로그램 하여라. ② 타이머 1,3을 이용하여 구현해보라.

5.1.7. 실험41 : 시간 조정이 가능한 시계

- 예제 : CA형의 FND5에 시간의 10의 자리값, FND4에 시간의 1의 자리값, FND3에 분의 10의 자리값, FND2에 분의 1의 자리값, FND1에 초의 10의 자리값, FND0에 초의 1의 자리값을 표시하는 시계를 만든다. INT0(PORTD.0, 핀25)에 연결된 스위치를 누르면 상승 에지 트리거에 의한 인터럽트가 발생하여 타이머 1의 카운팅이 정지되고 다시 한번 누르면 카운팅을 시작한다. INT1(PORTD.1, 핀26)에 연결된 스위치를 누르면 상승 에지 트리거에 의한 인터럽트가 발생하여 분값이 증가하고 INT2(PORTD.1, 핀27)에 연결된 스위치를 누르면 상승 에지 트리거에 의한 인터럽트가 발생하여 시간값이 증가한다.

● 회로도 : 그림 95의 회로를 구현하고 FND_CON을 그림 79에 보여진 시스템의 C_CON에 연결한다. 그림 80의 SW_CON을 그림 79의 D_CON에 연결한다. 또한 그림 80의 SW1~3를 누르면서 실험한다.

● 프로그램 예 :

```
#include <mega128.h>
unsigned char dbuf[6], dbuf_index;
unsigned char hou, min, sec, onoff=1;
unsigned int cnt0, cnt2;
void hex2bcd(unsigned char hex, unsigned char *ptr) {          /* 0x63 이하의 헥사값을 bcd로 바꾼다 */
    *ptr = hex % 10;
    *(ptr+1) = hex / 10;
}
void watch(void){
    if( ++sec == 60) { sec = 0;
            if( ++min == 60 ) { min = 0;
                    if( ++hou == 24 ) hou = 0;
            }
    }
    hex2bcd(sec, &dbuf[0]);
    hex2bcd(min, &dbuf[2]);
    hex2bcd(hou, &dbuf[4]);
}
interrupt [2] void exint0(void){  // interrupt [EXT_INT0] void exint0(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
    if(onoff == 0) onoff = 1; else onoff = 0;
}
interrupt [3] void exint1(void){  // interrupt [EXT_INT1] void exint1(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
    if( ++min == 60 ) { min = 0; if( ++hou == 24 ) hou = 0; }
}
interrupt [4] void exint2(void){  // interrupt [EXT_INT2] void exint2(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
    if( ++hou == 24 ) hou = 0;
}
interrupt [17] void timerint0(void){  // interrupt [TIM0_OVF] void timerint0(void)로 해도 됨
    if(onoff == 0) if( ++cnt0 == 1000 ){
    cnt0 = 0; watch(); }
    TCNT0 = 6;          // 재정의 1msec = 64*(256-6)/16       시뮬레이션할 때는 조절할 필요가 있음
}
interrupt [11] void timerint2(void){  // interrupt [TIM2_OVF] void timerint0(void)로 해도 됨
    unsigned char tmpbuf;
    if( ++cnt2 == 10 ){
    if( dbuf_index++ == 5 ) dbuf_index= 0;
    tmpbuf = dbuf[dbuf_index];
    PORTC = (( tmpbuf << 4 )&0xf0) | dbuf_index; }
                    /* 표시할 데이터와 FND의 위치는 tmpbuf의 상위비트와 하위비트로 */
    TCNT2 = 6;          // 재정의 1msec = 64*(256-6)/16       시뮬레이션할 때는 조절할 필요가 있음
}
void main(void){
    DDRC = 0xff;         // 포트C를 출력으로
    DDRD = 0x00;        //포트D를 입력으로 디폴트 입력으로 설정되어 있으므로 필요하지는 않음
    SREG.7 = 1;         //인터럽트 전체 허용
    EIMSK |= 0x01;      // INT0 개별 허용
    EIMSK |= 0x02;      // INT0 개별 허용
    EIMSK |= 0x04;      // INT0 개별 허용
    EICRA |= 3;         // 상승에지 트리거
    TIMSK |= 1;         // 타이머0 오버플로 인터럽트 개별허용
    TCCR0 |= 4;         // 분주비를 64로 설정                    시뮬레이션할 때는 조절할 필요가 있음
    TCNT0 = 6;          // 1msec = 64*(256-6)/16                 시뮬레이션할 때는 조절할 필요가 있음
    TIMSK |= 0x40;      // 타이머2 오버플로 인터럽트 개별허용
    TCCR2 |= 3;         // 분주비를 64로 설정                    시뮬레이션할 때는 조절할 필요가 있음
```

```
    TCNT2 = 6;          // 1msec = 64*(256-6)/16          시뮬레이션할 때는 조절할 필요가 있음
    for(;;);            /* 인터럽트 발생할 때까지 대기 */
}
```

- 과제 : ①타이머1,3를 사용하여 프로그램 하여라. ② 초값은 표시하지 않아 시간과 분만을 표시하여 FND 4개만을 사용하는 시간조정이 가능한 시계용 프로그램을 작성해보아라.

5.2. 키보드 스캐닝

5.2.1. 실험42 : 단순 키보드 스캔

- 예제 : D_CON에 연결된 8개의 누름키에서 PORTD.0에 연결된 것부터 차례대로 SW1, SW2, ...SW8의 일련번호를 붙이고 눌린 번호키의 2진수값을 PORTC에 연결된 LED에 표시하는 프로그램을 작성한다.
- 프로그램 작성시 유의점 : 채터링현상을 소프트웨어적으로 처리하기 위해 유효한 값을 읽기 위해 그림 96의 순서도에 주어진 순서를 따라한다. 먼저 연속해서 두번 읽어 같은 값이 감지되었는지 확인한다. 그 후 감지된 값이 키가 눌린 경우(읽힌 값은 0xff가 아니다)인지 여부를 확인한다. 읽은 값이 (20-50 msec) 이전에 읽어 oldkey에 보관된 값과 같으면 유효한 키값으로 keybuf에 저장하고 그렇지 않으면 20-50 msec시간 정도의 지연 후에 같은 과정을 반복한다. 즉 두 번 확인하는 방식을 택한다.
- 회로도 : 그림 80의 SW_CON을 그림 79의 D_CON에 연결하고 실험할 때는 LEDSW를 닫고 한다. 또한 그림 80의 SW1을 누르면서 실험한다.

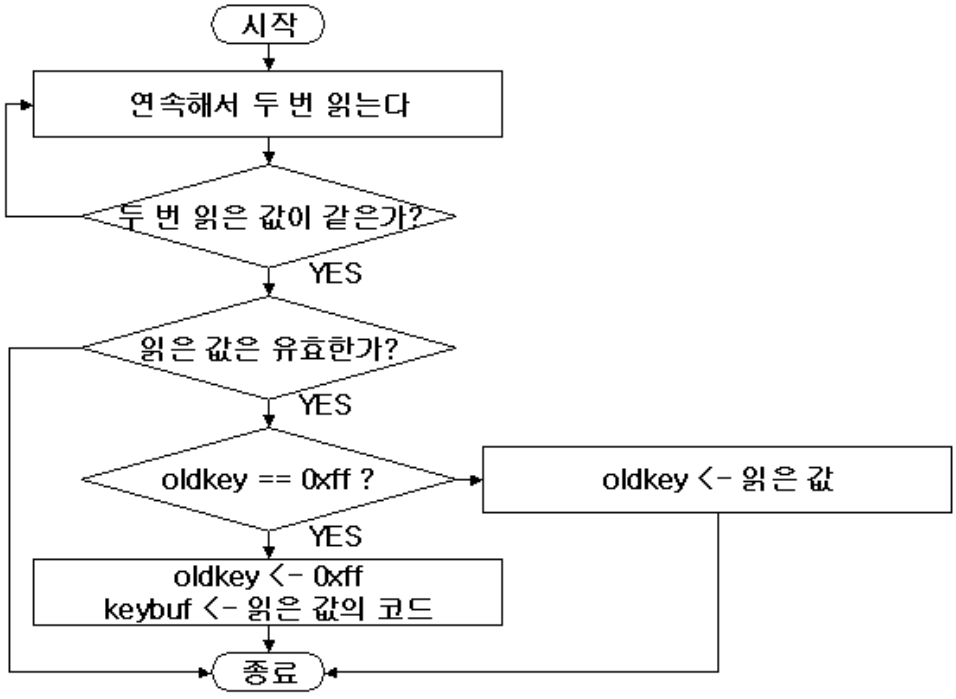


그림 96. 키보드 스캔을 위한 순서도

- 프로그램 예 :

```
#include <mega128.h>
#include <delay.h>        // delay_ms(), delay_us()를 이용할 것이다.
unsigned        char      keybuf=0xff, oldkey=0xff;
void scankey(void){
    unsigned char tmp;
    do{
    tmp = PIND;                         /* 첫번째 읽은 것을 tmp에 저장 */
    } while( tmp != PIND );       /* 첫번째 읽은 값과 두번째 읽은 값이 일치하지 않으면 다시 읽는다 */
    if(tmp != 0xff) {             /* 두번 읽은 것이 키가 눌린 경우이면 (0xff이면 아무키도 안눌린 상태임) 아래로 */
```

```
            if(tmp == oldkey) {          /* 현재 읽어들인 값(tmp)와 oldkey가 같으면 아래로 */
                  if(tmp == 0xfe) keybuf = 1;
                  else if( tmp == 0xfd ) keybuf = 2;
                  else if( tmp == 0xfb ) keybuf = 3;
                  else if( tmp == 0xf7 ) keybuf = 4;
                  else if( tmp == 0xef ) keybuf = 5;
                  else if( tmp == 0xdf ) keybuf = 6;
                  else if( tmp == 0xbf ) keybuf = 7;
                  else if( tmp == 0x7f ) keybuf = 8;
                  else oldkey = 0xff;
                  oldkey = 0xff;       }
            else      oldkey = tmp;      /* 현재 읽어들인 값(tmp)와 oldkey가 다르면 oldkey를 tmp로 바꾼다. */
      }
}
void main(void){
      DDRC = 0xff; // 포트C를 출력으로
      DDRD = 0x00; //포트D를 입력으로 디폴트 입력으로 설정되어 있으므로 필요하지는 않음
      for(;;){
      scankey();
      PORTC = keybuf;
      delay_ms(50);       /* 50 msec 시간 지연 */
      }
}
```

- 과제 : 타이머인터럽트를 사용하여 30msec마다 키보드 스캔을 하는 프로그램를 작성 하여라.

5.2.2. 실험43 : 키 매트릭스 스캔

- 예제 : D_CON에 연결된 5X5 키 매트릭스에 차례대로 SW1, SW2, ...SW25의 일련번호를 붙이고 눌린 번호키의 2진수값을 PORTC에 연결된 LED에 표시하는 프로그램을 작성한다.
- 프로그램 작성시 유의점 : 채터링현상을 소프트웨어적으로 처리하기 위해 유효한 값을 읽기 위해 그림 96의 순서도에 주어진 순서를 따라한다.
- 회로도 : 그림 80의 SW_CON을 그림 79의 D_CON에 연결하고 실험할 때는 LEDSW를 닫고 한다. 또한 그림 80의 SW1을 누르면서 실험한다.
- 프로그램 예 :

```
#include <mega128.h>
unsigned int cnt;
unsigned char keybuf=0xff, oldkey=0xff;
void scankey(void){
      unsigned char tmp, row, cul;
      for(row=0; row<5 ;row++){
            do{
            PORTC = 0xf8 | row;                  /* row에 1을 쓴다 */
            tmp = PIND & 0xf8 ;                  /* 첫번째 읽은 포트3의 비트7~비트3값을 tmp에 저장 */
            PORTC = 0xf8 | row;                  /* 두번째 읽기 위해 1을 쓴다 */
            } while( (PIND & 0xf8) != tmp );     /* 첫번째와 두번째 읽은 값이 일치하지 않으면 다시 읽는다 */
            if(tmp != 0xf8) {    /* 두번 읽은 값이 일치하고 키 눌림이 있는 경우라면 아래로 */
                  tmp += row;
                  if(tmp == oldkey) {            /* 현재의 키값(tmp)이 oldkey값과 같으면 아래로 */
                        tmp = (tmp >>3) &0x1f;
                        if(tmp == 0x1e) cul = 1;
                        else if( tmp == 0x1d ) cul = 2;
                        else if( tmp == 0x1b ) cul = 3;
                        else if( tmp == 0x17 ) cul = 4;
                        else if( tmp == 0x0f ) cul = 5;
                        else goto next;
```

```
                        keybuf = row*6 + cul;
                next:   oldkey = 0xff;      }
                else    oldkey = tmp;       /* 현재의 키값과 oldkey값이 다르면 oldkey <- tmp */
            }
        }
    }
    interrupt [17] void timerint0(void){  // interrupt [TIM0_OVF] void timerint0(void)로 해도 됨
        if( ++cnt == 50 ){                    // 50값을 시뮬레이션할 때는 조절할 필요가 있음
        cnt = 0; scankey(); }
        TCNT0 = 6;          // 재정의 1msec = 64*(256-6)/16      시뮬레이션할 때는 조절할 필요가 있음
    }
    void main(void){
        DDRC = 0xff;  // 포트C를 출력으로
        DDRD = 0x07; // 포트D의 비트0~2를 출력으로 설정
        SREG.7 = 1;         //인터럽트 전체 허용
        TCCR0 |= 4;         // 분주비를 64로 설정                  시뮬레이션할 때는 조절할 필요가 있음
        TCNT0 = 6;          // 1msec = 64*(256-6)/16               시뮬레이션할 때는 조절할 필요가 있음
        for(;;) PORTC = keybuf;
    }
```

- 과제 : 타이머인터럽트1~3를 사용하는 프로그램를 작성 하여라.

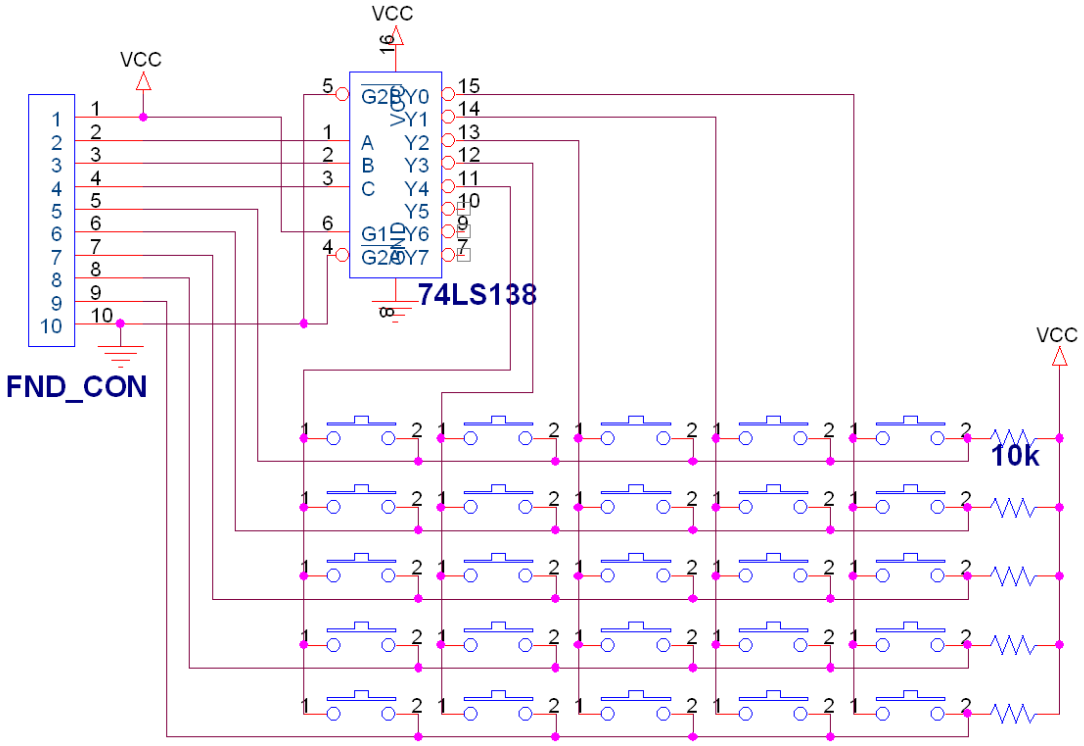


그림 97. 5X5 키 매트릭스

5.3. 음향 발생

5.3.1. 음향 발생 개요

- 음향 발생 원리: 스피커에 1레벨과 0레벨 신호를 반복적으로 주어 온오프시키는데 주기를 바꾸어 주면 음의 고저가 다른(주파수가 다른 신호)를 발생시킬 수 있고 지속시간을 바꾸어 주어 장단이 다른 신호를 발생시킨다.
- 음계 발생 프로그램 기법 : 내고자 하는 음계의 주파수(주기의 역수)가 주어지고 내고자 하는 음계의 지속시간이 주어지면 반주기(주파수 역수의 반) 동안 스피커를 온시켜주고 반주기 동안은 스피커를 오프시키는 것을 반복하되 지속시간(1/32박자를 1로 정규화한 time_data)이 지나면 끝내도록 하면 된다. 표13과 표14에 음계와 박자

에 따른 시간 정보를 보였다.

표 13. 음계에 따른 tone_data 값

음계	주파수x2(2/T)	반주기(msec)	음계	주파수x2(2/T)	반주기(msec)
도	1046.5	0.95	도#	1100.7	0.90
레	1174.7	0.84	레#	1244.5	0.80
미	1318.5	0.75	파	1396.9	0.71
파#	1479.9	0.67	솔	1567.9	0.63
라	1760.0	0.56	라#	1864.6	0.53
시	1975.6	0.50	도	2093.0	0.47
도#	2201.4	0.45	레	2349.4	0.42
레#	2489.0	0.39	미	2637.0	0.37
파	2793.8	0.35	파#	2959.8	0.33
솔	3135.8	0.31	솔#	3322.4	0.29
라	3520.0	0.28	라#	3729.2	0.26
시	3951.2	0.25	도	4186.0	0.23

표 14. 박자에 따른 지속시간값

박자	시간(msec)	time_data	박자	시간(msec)	time_data
1/32	21.306	1	1/16	42.613	2
1/8	85.227	4	1/4	170.454	8
1/2	340.909	16	1	681.818	32
1과 1/2	1,022.727	48	2	1,363.636	64
2와 1/2	1,704.545	80	3	2,045.454	96

5.3.2. 실험44 : 사이렌 소리내기

- 예제 : 저주파의 소리에서 점점 고주파의 소리를 변하게 하는 것을 반복적으로 한다. PORTD.0에 연결된 스위치를 누르면 PORTC.0에 연결된 스피커에서 소리를 내기 시작한다. 3박자마다 주파수를 변하게 한다.
- 프로그램 작성시 유의점 : 이 경우에는 음계에 따라 반주기값을 바꿔 줄 필요가 없다.
- 회로도 : 그림 98의 회로를 구현하고 BUZZ_CON을 그림 79의 C_CON에 연결한다. 그림 80의 SW_CON을 그림 79의 D_CON에 연결하고 그림 80의 SW1을 누르면서 실험한다. 그림 98에서 2SC1815Y와 PORTC.0 사이에 들어가는 콘덴서와 저항과 다이오드는 필터역할을 하는 것으로 생략가능하다. 즉 다이오드를 없애 2SC1815Y의 베이스와 에미터를 개방시키고 콘덴서와 저항은 없애고 PORTC.0과 2SC1815Y의 베이스를 연결해서 사용해도 된다.
- 프로그램 예 :

```
#include <mega128.h>
#include <delay.h>
unsigned char              time_data;
interrupt [17] void timerint0(void){   // interrupt [TIM0_OVF] void timerint0(void)로 해도 됨
      time_data--;
      TCNT0 = 77;          // 21.306msec = 1024*(256-77)/16          시뮬레이션할 때는 조절할 필요가 있음
}
void playtone(unsigned int tone, unsigned char time){ // tone는 반주기값 1usec, time는 박자값
unsigned int buf;
      TCCR0 |= 7;          // 분주비를 1024로 설정                  시뮬레이션할 때는 조절할 필요가 있음
      TCNT0 = 77;          // 21.306msec = 1024*(256-77)/16          시뮬레이션할 때는 조절할 필요가 있음
      time_data = time;
      buf = tone;
      do{
```

```
                PORTC.0 = 1;        do{ delay_us(1);} while(buf--);
                PORTC.0 = 0;        do{ delay_us(1);} while(tone--);
        } while(time_data);
        time_data = time;
        TCNT0 = 77;         // 21.306msec = 1024*(256-77)/16          시뮬레이션할 때는 조절할 필요가 있음
        while(time_data);
        TCCR0 =0x00;
}
void main(void){
        unsigned int tone_data=0xffff;
        DDRC = 0xff;          // 포트C를 출력으로
        DDRD = 0x00;        //포트D를 입력으로 디폴트 입력으로 설정되어 있으므로 필요하지는 않음
        SREG.7 = 1;         //인터럽트 전체 허용
        TIMSK |= 1;         // 타이머0 오버플로 인터럽트 개별허용
        for(;;){
loop:           while(PIND.0 == 1);             /* PORTD.0가 눌릴때까지 대기 */
                delay_ms(50);
                if(PIND.0 == 1) goto            loop;       /*눌림이 50msec동안 지속 안되면 키입력은 무시된다*/
                while(PIND.0 !=1);                          /* PORTD.0가 해제될 때까지 대기 */
                if(tone_data > 100){tone_data -=200;                        playtone(tone_data,250); }
                else tone_data=0xffff;
        }
}
```

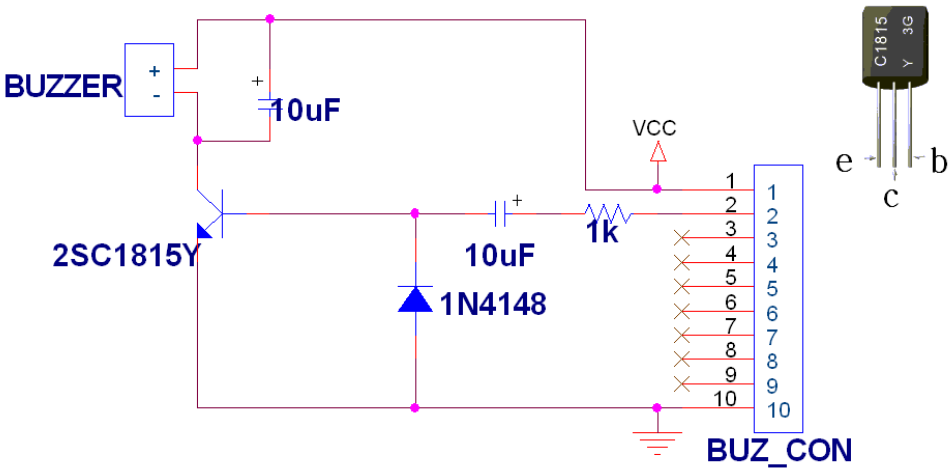


그림 98. 음향 발생을 위한 회로도

- 과제 : ①타이머1~3를 사용해 구현해 보라. ②PORTD.0에 연결된 스위치를 누르면 소리가 나고 5초후에 멈추는 프로그램을 작성하라.

5.3.3. 실험45 : 전화벨 소리내기

- 예제 : PORTD.0에 연결된 스위치가 눌리면 소리를 내기 시작하여 480Hz 1/32박자, 320Hz 1/32박자를 20번 반복하여 소리를 내고 멈춘다.
- 프로그램 작성시 유의점 : 480Hz와 320Hz의 주기는 각각 2.083msec와 3.125msec이다. 1/32박자만큼 지속하므로 time_data를 1로 주면 된다.
- 회로도 : 그림 98의 회로를 구현하고 BUZZ_CON을 그림 79의 C_CON에 연결한다. 그림 80의 SW_CON을 그림 79의 D_CON에 연결하고 그림 80의 SW1을 누르면서 실험한다.
- 프로그램 예 :

```
#include <mega128.h>
```

```
#include <delay.h>
unsigned char           time_data;
interrupt [17] void timerint0(void){  // interrupt [TIM0_OVF] void timerint0(void)로 해도 됨
    time_data--;
    TCNT0 = 77;       // 21.306msec = 1024*(256-77)/16          시뮬레이션할 때는 조절할 필요가 있음
}
void playtone(unsigned char tone, unsigned char time){ // tone는 반주기값 10usec, time는 박자값
unsigned char buf;
    TCCR0 |= 7;       // 분주비를 1024로 설정                  시뮬레이션할 때는 조절할 필요가 있음
    TCNT0 = 77;       // 21.306msec = 1024*(256-77)/16          시뮬레이션할 때는 조절할 필요가 있음
    time_data = time;
    buf = tone;
    do{
            PORTC.0 = 1;      do{ delay_us(10);} while(buf--);
            PORTC.0 = 0;      do{ delay_us(10);} while(tone--);
    } while(time_data);
    time_data = time;
    TCNT0 = 77;       // 21.306msec = 1024*(256-77)/16          시뮬레이션할 때는 조절할 필요가 있음
    while(time_data);
    TCCR0 =0x00;
}
void main(void){
    unsigned char i;
    DDRC = 0xff;       // 포트C를 출력으로
    DDRD = 0x00;      //포트D를 입력으로 디폴트 입력으로 설정되어 있으므로 필요하지는 않음
    SREG.7 = 1;       //인터럽트 전체 허용
    TIMSK |= 1;       // 타이머0 오버플로 인터럽트 개별허용
    for(;;){
loop:       while(PIND.0 == 1);          /* PORTD.0가 눌릴때까지 대기 */
            delay_ms(50);
            if(PIND.0 == 1) goto          loop;     /*눌림이 50msec동안 지속 안되면 키입력은 무시된다*/
            while(PIND.0 !=1);                      /* PORTD.0가 해제될 때까지 대기 */
            for(i=0;i<20;i++){ playtone(208/2,1);
            playtone(313/2,1);}
    }
}
```

● 과제 :①타이머1~3를 사용해 구현해 보라. ② PORTC.0에 연결된 스위치를 누르면 전화벨 소리가 나고 PORTC.1에 연결된 스위치를 누르면 사이렌소리가 나는 프로그램을 작성하라.

5.3.4. 실험46 : 동요 재생하기

● 예제 : PORTC.0에 연결된 스위치를 누르면 동요 곰 세마리가 연주한다.

● 회로도 : 그림 98의 회로를 구현하고 BUZZ_CON을 그림 79의 C_CON에 연결한다. 그림 80의 SW_CON을 그림 79의 D_CON에 연결하고 그림 80의 SW1을 누르면서 실험한다.

● 프로그램 작성시 유의점 : 메모리에 동요 곰 세마리의 음계와 박자를 저장하되 마지막에 박자에 00가 들어가도록 한다. 순서대로 읽어 들이되 박자가 0이면 연주가 끝났으므로 처음으로 복귀하여 스위치가 눌리는지를 감시하도록 한다.

● 프로그램 예 :

```
#include <mega128.h>
#include <delay.h>
unsigned char           time_data;
unsigned char  timetone[ ]={32, 95, 16, 95, 32, 95, 32, 95, 32, 75, 16, 63, 16, 63, 32, 75, 32, 95,
    16, 63, 16, 63, 32, 75, 16, 63, 16, 63, 32, 75, 32, 95, 32, 95, 64, 95,
    32, 63, 32, 63, 32, 75, 32, 95, 32, 63, 32, 63, 64, 63,
    32, 63, 32, 63, 32, 75, 32, 95, 32, 63, 32, 63, 64, 63,
```

```
    32, 63, 32, 63, 32, 75, 32, 95, 24, 63, 8, 63, 16, 63, 16, 63, 64, 63,
    16, 31, 16, 0, 16, 63, 16, 0, 16, 31, 16, 0, 16, 63, 16, 0, 32, 75, 32, 84, 64, 95, 0};
interrupt [17] void timerint0(void){  // interrupt [TIM0_OVF] void timerint0(void)로 해도 됨
    time_data--;
    TCNT0 = 77;        // 21.306msec = 1024*(256-77)/16          시뮬레이션할 때는 조절할 필요가 있음
}
void playtone(unsigned char tone, unsigned char time){ // tone는 반주기값 10usec, time는 박자값
unsigned char buf;
    TCCR0 |= 7;        // 분주비를 1024로 설정                  시뮬레이션할 때는 조절할 필요가 있음
    TCNT0 = 77;        // 21.306msec = 1024*(256-77)/16          시뮬레이션할 때는 조절할 필요가 있음
    time_data = time;
    buf = tone;
    do{
            PORTC.0 = 1;      do{ delay_us(10);} while(buf--);
            PORTC.0 = 0;      do{ delay_us(10);} while(tone--);
    } while(time_data);
    time_data = time;
    TCNT0 = 77;        // 21.306msec = 1024*(256-77)/16          시뮬레이션할 때는 조절할 필요가 있음
    while(time_data);
    TCCR0 =0x00;
}
void main(void){
unsigned char dataidx, tone_data, time;
    DDRC = 0xff;        // 포트C를 출력으로
    DDRD = 0x00;       //포트D를 입력으로 디폴트 입력으로 설정되어 있으므로 필요하지는 않음
    SREG.7 = 1;        //인터럽트 전체 허용
    TIMSK |= 1;        // 타이머0 오버플로 인터럽트 개별허용
    for(;;){
            dataidx = 0;
loop:       while(PIND.0 == 1);          /* PORTD.0가 눌릴때까지 대기 */
            delay_ms(50);
            if(PIND.0 == 1) goto         loop;     /*눌림이 50msec동안 지속 안되면 키입력은 무시된다*/
            while(PIND.0 !=1);                     /* PORTD.0가 해제될 때까지 대기 */
            while ( (time =timetone[dataidx++]) != 0) {
                    tone_data=timetone[dataidx++];
                    playtone(tone_data, time);}
    }
}
```

- 과제 : ①타이머1~3를 사용해 구현해 보라. ②다른 동요를 저장하고 연주하는 프로그램을 작성해보라.

5.3.5. 실험47 : 전자 오르간

- 예제 : PORTD에 연결된 스위치가 눌리면 맞는 음계의 소리를 발생시킨다.
- 회로도 : 그림 98의 회로를 구현하고 BUZZ_CON을 그림 79의 C_CON에 연결한다. 그림 80의 SW_CON을 그림 79의 D_CON에 연결하고 그림 80의 SW1~8을 누르면서 실험한다.
- 프로그램 작성시 유의점 : 박자를 위해 타이머를 쓸 필요없이 눌림을 감지해 감지되면 한 주기 동안만 해당 음계의 소리가 나도록 한다. SW1이 눌리면 도 음계가 발생하고 SW2가 눌리면 레 음계가 발생하는 식으로 짠다.
- 프로그램 예 :

```
#include <mega128.h>
#include <delay.h>
unsigned char          time_data;
interrupt [17] void timerint0(void){  // interrupt [TIM0_OVF] void timerint0(void)로 해도 됨
    time_data--;
    TCNT0 = 77;        // 21.306msec = 1024*(256-77)/16          시뮬레이션할 때는 조절할 필요가 있음
```

```
}
void playtone(unsigned char tone, unsigned char time){ // tone는 반주기값 10usec, time는 박자값
unsigned char buf;
    TCCR0 |= 7;         // 분주비를 1024로 설정                    시뮬레이션할 때는 조절할 필요가 있음
    TCNT0 = 77;         // 21.306msec = 1024*(256-77)/16           시뮬레이션할 때는 조절할 필요가 있음
    time_data = time;
    buf = tone;
    do{
            PORTC.0 = 1;        do{ delay_us(10);} while(buf--);
            PORTC.0 = 0;        do{ delay_us(10);} while(tone--);
    } while(time_data);
    time_data = time;
    TCNT0 = 77;         // 21.306msec = 1024*(256-77)/16           시뮬레이션할 때는 조절할 필요가 있음
    while(time_data);
    TCCR0 =0x00;
}
void main(void){
unsigned char           tone_data, keybuf;
    DDRC = 0xff;         // 포트C를 출력으로
    DDRD = 0x00;        //포트D를 입력으로 디폴트 입력으로 설정되어 있으므로 필요하지는 않음
    SREG.7 = 1;         //인터럽트 전체 허용
    TIMSK |= 1;         // 타이머0 오버플로 인터럽트 개별허용
    for(;;){
    keybuf = PIND;              /* 읽은 값을 keybuf에 저장 */
    if((keybuf == PIND) && (keybuf !=0xff) ) {          /*두번 읽어 같으며 0xff가 아니면 유효한 키입력이다*/
            if(keybuf == 0xfe) tone_data= 95;
            else if (keybuf == 0xfd) tone_data = 84;
            else if (keybuf == 0xfb) tone_data = 75;
            else if (keybuf == 0xf7) tone_data = 71;
            else if (keybuf == 0xef) tone_data = 63;
            else if (keybuf == 0xef) tone_data = 56;
            else if (keybuf == 0xbf) tone_data = 50;
            else if (keybuf == 0x7f) tone_data = 47;
            else tone_data = 0;
            if(tone_data !=0)playtone(tone_data,1);
    }}
}
```

- 과제 : 타이머1~3를 사용해 구현해 보라.

5.4. 16x2 문자 LCD

5.4.1. 16x2 문자 LCD 모듈 개요

- LCD 개요: 동적 LCD와 전계효과 LCD가 있는데 전계효과형이 가장 흔히 사용되어 전자계산기, 시계와 컴퓨터 등에 사용된다. 일반적으로 약 300 μA 의 작은 구동전류에서 작동하나 교류전원을 요구한다. LCD 표시부와 LCD 제어부를 하나로 하여 LCD 모듈로 시판된다. 문자형과 그래픽형 2가지가 있다. 영문자와 숫자표시에는 문자형이 적당하며 한글이나 그래픽 표시를 자유롭게 하려면 그래픽형을 사용해야 한다. 문자형에는 8x1(8문자를 1라인에 표시), 8x2, 16x1, ..., 16x4, 20x1, ..., 20x4 등 다양한 종류가 있다.
- HD44780 16x2 LCD 제어기 특징 : 16x2 LCD제어기에 가장 보편적으로 사용되는 히타치사의 제어기, 5V 단일 전원 사용, 5x7 나 5x10 도트의 내장 폰트 존재, 8비트 나 4비트 데이터 버스 방식, 최대 80문자까지 표시할 문자 저장하는 80바이트 DDRAM(Display Data RAM) 내장, 5x7 폰트는 160개 5x10 폰트는 32개를 수용할 수 있는 CGROM(Character Generator ROM) 내장.
- 핀배치 : 1번(Vss:GND), 2번(Vdd: +5V), 3번(Vo:휘도조절, 보통 GND에 연결), 4번(RS: Register Select핀

으로 1레벨이면 데이터레지스터가 선택되고 0레벨이면 명령레지스터가 선택됨), 5번(R/W: 읽기(H), 쓰기(L) 제어), 6번(E: Enable신호로 1레벨이면 LCD에 명령이나 데이터를 보낼 수 있다), 15번(A: 백라이트양극), 16번(K: 백라이트 음극), 7-14번(DB0-DB7:데이터버스, 4비트모드의 경우 D4-D7를 사용하여 상위4비트 하위4비트 순으로 읽고 씀)

5.4.2. LCD 제어기의 내부 구성

명령(Instruction)과 데이터(Data)를 위한 2개의 레지스터, BF(Busy Flag), AC(Address Counter), 문자발생램(CGRAM), 문자발생롬(CGROM), 데이터표시램(DDRAM)이 있다.

- 레지스터

  명령레지스터(IR): DDRAM과 CGRAM에 대한 어드레스와 클리어, 커서시프트 등 제어명령을 보유

  데이터레지스터(DR): DDRAM과 CGRAM에 쓴 데이터나 읽은 데이터를 일지적으로 저장

  레지스터 선택방법: RS(4번핀)과 R/W(5번핀)을 사용하여 선택

  RS=0, R/W=0 : IR쓰기(각종 제어 명령 쓰기)

  RS=0, R/W=1 : BF읽기, AC읽기

  RS=1, R/W=0 : DR쓰기

  RS=1, R/W=1 : DR읽기

- BF : 1이면 LCD의 콘트롤러가 동작 중으로 명령 수행 불능, 0이면 다음 명령 수행 가능 표시

- AC : DDRAM과 CGRAM의 주소를 지정할 때 사용하는데 IR에 주소 정보를 쓰면 주소 정보가 AC로 전송되고 DDRAM이나 DDROM에 데이터를 쓰면 AC는 자동으로 +1혹은 -1이 된다.

- DDRAM(Data Display RAM) : 80x8비트 용량으로 80개의 8비트 아스키코드를 저장할 수있다. 0x00-0F 주소가 LCD 1행의 1-16번째, 0x40-4F 주소가 LCD 2행의 1-16번째 문자로 표시된다.(1행 2행의 번호 불연속임에 주의해야 한다.)

- CGRAM(Character Generator RAM) : 사용자가 자유로이 문자를 만들 때 사용하는 램으로 5x7은 8개, 5x10은 4개 만들어 저장 가능하다.

- CGROM(Character Generator ROM) : 5x7, 5x10 도트의 문자를 내장하고 있다. 특수기호, 숫자, 영문자의 문자코드는 아스키코드와 일치한다.

5.4.3. 명령어

4번(RS), 5번(R/W), 6번(E: ENABLE신호), 15번(A: 백라이트양극), 16번(K: 백라이트 음극), 7-14번(DB0-DB7) 값을 이용하여 제어한다. E=1이어야 명령이 유효하다.

- 표시클리어: RS=R/W=0, DB=0b00000001, 실행시간 1.64msec, 표시내용을 소거하고 커서는 홈(1행1열, 0번지)로 돌아감, DDRAM에 스페이스코드(0x20)가 쓰여지고 AC=0으로 됨

- 커서홈 : RS=R/W=0, DB=0b0000001x, 실행시간 1.64msec, DDRAM의 AC=0x00로 세트되어 커서가 홈으로 가며 DDRAM의 내용은 변동 없음

- 엔트리모드 설정: 커서의 이동방향과 표시 내용의 시프트 여부 설정, 실행시간 0.04msec

  RS=R/W=0, DB=0b00000101 : DDRAM에 데이터를 써넣은 후 표시 전체를 좌로 이동, 커서는 이동 안함

  RS=R/W=0, DB=0b00000111 : DDRAM에 데이터를 써넣은 후 표시 전체를 우로 이동, 커서는 이동 안함

  RS=R/W=0, DB=0b00000100 : 표시를 이동하지 않고 AC를 감소시키고 커서가 좌측으로 이동하게 함

RS=R/W=0, DB=0b00000110 : 표시를 이동하지 않고 AC를 증가시키고 커서가 우측으로 이동하게 함

- 표시 온오프제어: 커서나 전체표시의 온오프제어 및 커서위치 문자의 깜박임 제어, 실행시간 0.04msec

  RS=R/W=0, DB=0b00001dcb : d=1일때 전체 온, c=1 커서 온, b=1 커서 위치 문자 점멸

- 커서/표시 시프트: DDRAM의 내용을 변경시키지 않은 상태에서 커서를 움직이거나 전체문자를 이동, 실행시간 0.04msec

  RS=R/W=0, DB=0b000100xx : AC를 1감소시키고 커서를 좌로 이동

  RS=R/W=0, DB=0b000101xx : AC를 1증가시키고 커서를 우로 이동

  RS=R/W=0, DB=0b000110xx : 모든 표시와 커서를 좌로 이동

  RS=R/W=0, DB=0b000111xx : 모든 표시와 커서를 우로 이동

- 기능 설정: 표시행수와 문자 폰트, 인터페이스 길이를 설정, 실행시간 0.04msec

  RS=R/W=0, DB=0b0011NFxx : 8비트 인터페이스

  RS=R/W=0, DB=0b0010NFxx : 4비트 인터페이스

  여기에서 NF=00(1행 5x7 폰트), NF=01(1행 5x10 폰트), NF=1x(2행 5x7 폰트)

- CGRAM 주소설정: CGRAM의 6비트 주소를 설정, 설정 후 전송 데이터는 CGRAM의 데이터로 취급됨, 실행시간 0.04msec

  RS=R/W=0, DB=0b01xxxxxx : xxxxxx 부분의 값이 AC로 설정됨

- DDRAM 주소설정: DDRAM의 7비트 주소를 설정, 설정 후 전송 데이터는 DDRAM의 데이터로 취급됨, 실행시간 0.04msec

  RS=R/W=0, DB=0b1xxxxxxx : xxxxxxx 부분의 값이 AC로 설정됨

- BF와 AC 읽기: BF의 값이나 AC의 값을 읽어 들인다. 실행시간 0.04msec

  RS=0, R/W=1 : DB7이 BF의 값이고 DB6-0가 AC값이다.

- CGRAM, DDRAM에 데이터 쓰기: 이전에 주소가 설정된 램에 쓰고 엔트리 모드에 따라 주소값이 +1 혹은 -1

  RS=1, R/W=0 : 기능설정의 인터페이스 길이에 따라 DB를 통해 쓴다.

- CGRAM, DDRAM에 데이터 읽기: 읽기 전에 주소설정 명령을 해야 함, 안 그러면 두 번째 데이터부터 유효

  RS=1, R/W=1 : 기능설정의 인터페이스 길이에 따라 DB를 통해 읽는다.

5.4.4. 초기화 방법(4비트 인터페이스 경우)

전원 온 -> Vdd가 4.5V 이상된 후 15msec 지연 -> RS=R/W=0, DB=0b0010xxxx 출력

-> 5msec 지연 -> RS=R/W=0, DB=0b0010xxxx -> 0.1msec 지연 ->

-> 4비트 인터페이스 기능설정 RS=R/W=0, DB=0b0010NFxx(NF는 적절한 값)

-> 표시 온오프 제어 RS=R/W=0, DB=0b00001100

-> 표시클리어 RS=R/W=0, DB=0b00000001

-> 엔트리모드 설정 RS=R/W=0, DB=0b000001**(**는 적절한 값) -> 초기화 완료

5.4.5. 실험48 : LCD 초기화

● 예제 : PORTC에 연결된 LCD를 4비트 인터페이스 모드로 초기화하고 Enjoy yourselves라는 문자열을 1행에 mega128 u-COM라는 문자열을 2행에 출력한다.

● 프로그램 작성시 유의점 : 다른 응용에서도 include시켜 써먹을 수 있도록 작성하였다. 그러므로 작성된 프로그램으로 실험할 때에는 main()의 앞뒤에 있는 /*********과 *******/ 주석표시를 제거한다. lcd.h 헤더파일을 다음처럼 include시키면 된다.

```
#asm
.equ __lcd_port= 0x15  ;포트C에 LCD 위치
#endasm
#include <lcd.h>
```

그러면 다음 함수를 사용해서 lcd를 제어할 수 있다.

lcd_init(unsigned char lcd_columns) : lcd_columns개의 열을 갖는 LCD를 사용할 수 있도록 초기화한다.

lcd_clear() : LCD에 써진 것을 지운다.

lcd_gotoxy(unsighed char x, unsigned char y) : 커서를 x+1열, y+1행 위치로 보낸다.

lcd_putchar(char c) : LCD의 현재 커서에 문자 c를 표시한다.

lcd_putsf(char flash *str) : flash에 위치한 스트링을 LCD의 현재 커서를 시작점으로 하여 표시한다.

● 회로도 : 그림 99의 회로를 구현하고 LCD_CON을 그림 79의 C_CON에 연결한다.

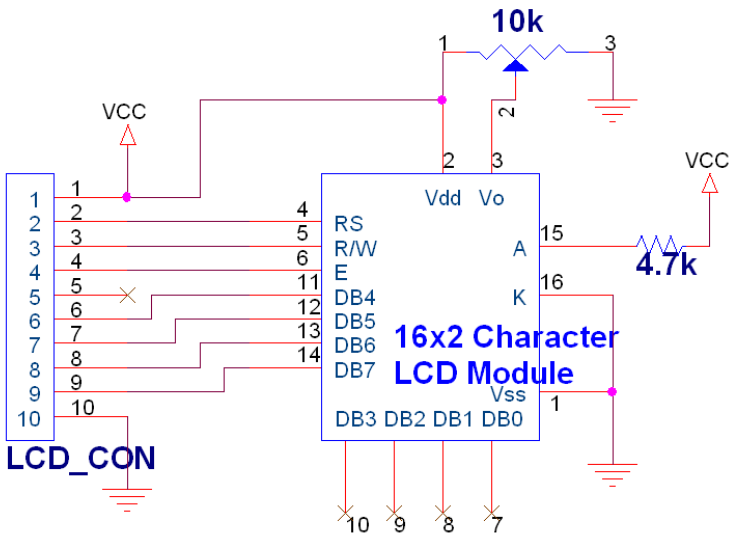


그림 99. 16x2 문자 LCD 실험 회로도

● 프로그램 예1 :

```
#include <mega128.h>
#include <delay.h>
#define     CUR11       0x02    /* 커서 홈에 위치시키는 명령어 값 */
#define     CUR21       0xC0    /* 커서 2라인 1열에 위치시키는 명령어 값 */
#define     LCDON       0x0C    /* LCD ON 명령어 값 */
#define     LCDOFF      0x08    /* LCD OFF 명령어 값 */
#define     MODE4       0x20    /* 4비트 인터페이스 설정 */
#define     FSET        0x28    /* 기능설정, 4비트 인터페이스, NF=10 : 2행 5x7 폰트 */
#define     LCDCLR      0x01    /* LCD 클리어 */
#define     MODENT      0x06    /* 엔트리모드 설정, 표시는 이동 않고, AC 증가, 커서 우측 이동 */
void lcd_cmd(unsigned char ch){         /* LCD에 명령어값 1바이트 쓰는 서브루틴, ch는 명령어 값 */
    unsigned char temp0, temp1;
    delay_ms(15);               /* 약 15msec delay */
    DDRC = 0xff;                                    // 포트C를 출력으로
```

```
    temp0 = ch & 0xf0;                          /* 상위4비트 마스크 */
    temp1 = temp0 | 0x04;                       /* 하위 4비트에 0100b(RS=R/W=0, E=1) 써넣기 */
    PORTC = temp1;                              /* LCD에 명령어값의 상위 4비트 써넣기 */
    temp1 = temp0 & 0xf0;                       /* 하위 4비트에 0000b(RS=R/W=0, E=0) 써넣기    */
    PORTC = temp1;                              /* LCD에 입력 불허 */
    delay_us(10);                               /* 약 0.01 msec 시간지연 */
    temp0 = (ch << 4) & 0xf0 ;                  /* 하위4비트를 상위로 옮기고 마스크 */
    temp1 = temp0 | 0x04 ;                      /* 하위 4비트에 0100b(RS=R/W=0, E=1) 써넣기 */
    PORTC = temp1;                              /* LCD에 명령어값의 하위 4비트 써넣기 */
    temp1 = temp0 & 0xf0;                       /* 하위 4비트에 0000b(RS=R/W=0, E=0) 써넣기    */
    PORTC = temp1;          /* LCD에 입력 불허 */
    delay_ms(1);            /* 약 1 msec 시간지연 */
}
void lcd_ln11(void){    /* LCD의 1행1열에 커서를 위치시키는 서브루틴 */
    lcd_cmd(CUR11);
}
void lcd_ln21(void){    /* LCD의 2행1열에 커서를 위치시키는 서브루틴 */
    lcd_cmd(CUR21);
}
void lcd_dat(unsigned char ch){    /* LCD에 1글자 쓰는 서브루틴, ch는 글자의 아스키 코드 값 */
    unsigned char temp0, temp1;
    delay_ms(15);           /* 약 15msec delay */
    DDRC = 0xff;                                // 포트C를 출력으로
    temp0 = ch & 0xf0;      /* 상위4비트 마스크 */
    temp1 = temp0 | 0x05;   /* 하위 4비트에 0101b(RS=1, R/W=0, E=1) 써넣기 */
    PORTC = temp1;          /* LCD에 DATA값의 상위 4비트 써넣기 */
    temp1 = temp0 | 0x01; /* 하위 4비트에 0001b(RS=1, R/W=0, E=0) 써넣기       */
    PORTC = temp1;          /* LCD에 입력 불허 */
    delay_us(10);           /* 약 0.01 msec 시간지연 */
    temp0 = (ch << 4) & 0xf0 ;          /* 하위4비트를 상위로 옮기고 마스크 */
    temp1 = temp0 | 0x05;   /* 하위 4비트에 0101b(RS=1, R/W=0, E=1) 써넣기 */
    PORTC = temp1;          /* LCD에 DATA값의 하위 4비트 써넣기 */
    temp1 = temp0 | 0x01; /* 하위 4비트에 0001b(RS=1, R/W=0, E=0) 써넣기       */
    PORTC = temp1;          /* LCD에 입력 불허 */
    delay_ms(1);            /* 약 1 msec 시간지연 */
}
void lcd_init(void){    /* LCD 초기화 서브루틴 */
    delay_ms(15);           /* 약 15msec delay */
    lcd_cmd(MODE4);
    delay_ms(5);            /* 약 5msec delay */
    lcd_cmd(MODE4);   delay_us(100);
    lcd_cmd(MODE4);
    lcd_cmd(FSET);          /* 기능설정, 4비트 인터페이스, NF=10 : 2행 5x7 폰트 */
    lcd_cmd(LCDON);         /* LCD ON 명령어 값 */
    lcd_cmd(LCDCLR);        /* LCD Clear 명령어 값 */
    lcd_cmd(MODENT);        /* 엔트리모드 설정, 표시는 이동 않고, AC 증가, 커서 우측 이동 */
}
void lcd_str(char flash *str){    /* LCD에 flash에 저장된 문자열을 출력하는 함수 */
    unsigned int        i=0;
    for(;str[i] != 0; i++) lcd_dat(str[i]);
}
/************************************
void main(void){
    lcd_init();
    lcd_ln11();
    lcd_str("Enjoy yourselves");
    lcd_ln21();
    lcd_str("mega128 u-COM");
}
```

```
*********************************/
```

● 프로그램 예2 : lcd.h를 사용하면 다음처럼 쉽게 짤 수 있다.

```
#include <mega128.h>
#include <delay.h>
#asm
.equ __lcd_port= 0x15  ;포트C에 LCD 위치
#endasm
#include <lcd.h>
void main(void){
      lcd_init(16);                  lcd_gotoxy(0,0);          lcd_putsf("Enjoy yourselves");
                                     lcd_gotoxy(0,1);          lcd_putsf("mega128 u-COM");
}
```

5.4.6. 실험49 : 점멸하며 문자표시하기

● 예제 : PORTC에 연결된 LCD를 4비트 인터페이스 모드로 초기화하고 Enjoy yourselves라는 문자열을 1행에 mega128 u-COM라는 문자열을 2행에 출력하되 1초 정도 간격으로 깜박이도록 한다.

● 회로도 : 그림 99의 회로를 구현하고 LCD_CON을 그림 79의 C_CON에 연결한다.

● 프로그램 작성시 유의점 : 실험45의 프로그램(main()함수를 제거해버린)을 lcdchoi.h라고 저장하고 이를 include시켜 초기화하기 위한 서브루틴 lcd_init, 커서위치를 위한 서브루틴 lcd_ln11과 lcd_ln21, 명령 출력을 위한 서브루틴 lcd_cmd, 데이터 출력을 위한 서브루틴 lcd_dat()을 재정의 없이 사용한다.

● 프로그램 예 :

```
#include "lcdchoi.h"
void main(void){
      lcd_init();                          lcd_ln11();                    lcd_str("Enjoy yourselves");
                                           lcd_ln21();                    lcd_str("mega128 u-COM");
      for(;;){
            lcd_cmd(LCDOFF);
            delay_ms(1000);     /* 약 1초 시간지연 */
            lcd_cmd(LCDON);     delay_ms(1000);          }
}
```

5.4.7. 실험50 : 사용자 폰트 이용하기

● 예제 : 최한호라는 폰트를 만들고 등록하여 LCD의 1행에는 'Choi Han Ho is'라는 문자열을 출력하고 2행에는 '최한호'라는 문자열을 출력한다.

● 프로그램 작성시 유의점 : CGRAM은 5x8 폰트 8개, 5x10 폰트 4개를 사용자가 정의해서 자유롭게 사용할 수 있다. 폰트의 패턴제작은 그림 100을 참조하면 알 수있다. 5x8의 행렬에 원하는 문자를 형상화한다고 생각하고 채워야 하는 곳은 1을 써 넣고 공백으로 남겨야 하는 곳에 0을 써서 만든다(그림 100에서 x는 무정의 조건으로 1이되도 상관없다. 그러므로 패턴값 0x09나 0x49나 0x89나 같다). 문자폰트를 등록하기 위해서는 CGRAM 번지 설정 명령을 LCD에 보내고 패턴값 데이터를 보내주면 된다. 한 개의 문자폰트를 등록하기 위해서는 5x8 폰트의 경우 번지 설정 명령과 패턴값 데이터 보내는 것을 8번 수행해야 한다. 문자폰트의 등록은 한 문자당 8개 번지가 소요되면 0~7, 8~15, 16~23번지 등등 처럼 등록이 된다. 등록된 문자폰트의 코드는 차례로 0부터 일련번호가 붙으므로 두번째로 등록된 문자폰트(8~15번지에 등록이 되어 있는 문자폰트)를 불러 쓸려면 lcd_dat(0x01) (lcdchoi.h에 있는 서브루틴)명령을 사용하면 된다.

● 회로도 : 그림 99의 회로를 구현하고 LCD_CON을 그림 79의 C_CON에 연결한다.

● 프로그램 예 :

```
#include "lcdchoi.h"
void main(void){
```

```
unsigned char i, chhcode[24] = { 0x09, 0x1D, 0x09, 0x15, 0x15, 0x01, 0x09, 0x1D,
                                 0x0A, 0x1E, 0x0A, 0x17, 0x0A, 0x02, 0x10, 0x1E,
                                 0x04, 0x1F, 0x0E, 0x11, 0x11, 0x0E, 0x04, 0x1F};
    lcd_init();
    for(i=0;i<24;i++){                      /* CGRAM에 패턴 저장하기 */
            lcd_cmd(0x40+i);                /* CGRAM 주소는 0x40부터 시작 */
            lcd_dat(chhcode[i]);            /* data write */
    }
    lcd_ln11();
    lcd_str("Choi Han Ho is  ");
    lcd_ln21();
    lcd_dat(0x00);      lcd_dat(0x01);      lcd_dat(0x02);
}
```

데이터								패턴값
x	x	x	0	1	0	0	1	09H
x	x	x	1	1	1	0	1	1DH
x	x	x	0	1	0	0	1	09H
x	x	x	1	0	1	0	1	15H
x	x	x	1	0	1	0	1	15H
x	x	x	0	0	0	0	1	01H
x	x	x	0	1	0	0	1	09H
x	x	x	1	1	1	0	1	1DH

데이터								패턴값
x	x	x	0	1	0	1	0	0AH
x	x	x	1	1	1	1	0	1FH
x	x	x	0	1	0	1	0	0AH
x	x	x	1	0	1	1	1	17H
x	x	x	0	1	0	1	0	0AH
x	x	x	0	0	0	1	0	02H
x	x	x	1	0	0	0	0	10H
x	x	x	1	1	1	1	0	1EH

데이터								패턴값
x	x	x	0	0	1	0	0	04H
x	x	x	1	1	1	1	1	1FH
x	x	x	0	1	1	1	0	0EH
x	x	x	1	0	0	0	1	11H
x	x	x	1	0	0	0	1	11H
x	x	x	0	1	1	1	0	0EH
x	x	x	0	0	1	0	0	04H
x	x	x	1	1	1	1	1	1FH

그림 100. LCD에 문자 등록을 위해 패턴값 구하는 예

- 과제 : 다른 글자를 등록하고 이를 표시하는 프로그램를 작성 하여라.

5.4.8. 실험51 : 시간 조정이 가능한 시계

- 예제 : INT0(PORTD.0, 핀25)에 연결된 스위치를 누르면 상승 에지 트리거에 의한 인터럽트가 발생하여 타이머 1의 카운팅이 정지되고 다시 한번 누르면 카운팅을 시작한다. INT1(PORTD.1, 핀26)에 연결된 스위치를 누르면 상승 에지 트리거에 의한 인터럽트가 발생하여 분값이 증가하고 INT2(PORTD.1, 핀27)에 연결된 스위치를 누르면 상승 에지 트리거에 의한 인터럽트가 발생하여 시간값이 증가한다. LCD의 1행에는 ‘ Hour Min Sec ' 라는 문자열을 출력하고 2행에는 ’ hh mm ss '형태로 시간, 분, 초가 나타난다 .

- 회로도 : 그림 99의 회로를 구현하고 LCD_CON을 그림 79의 C_CON에 연결한다.그림 80의 SW_CON을 그림 79의 D_CON에 연결한다. 또한 그림 80의 SW1~3를 누르면서 실험한다.

- 프로그램 작성시 유의점 : 타이머0의 모드1을 사용하였고 메인프로그램에서는 키가 눌렸는지를 감시한다. lcdchoi.h라고 저장하고 이를 include시켜 초기화하기 위한 서브루틴 lcd_init, 커서위치를 위한 서브루틴 lcd_ln11()과 lcd_ln21(), 명령 출력을 위한 서브루틴 lcd_cmd(), 데이터 출력을 위한 서브루틴 lcd_dat(), 그리고 시간지연 서브루틴 delay()를 재정의 없이 사용한다. 타이머로 계산한 시간 데이터 숫자값을 출력하기 위해서는 숫자에 해당하는 코드를 LCD에 써줘야 하는데 ‘0’의 코드는 0x30, '1'의 코드는 0x31, ‘2’의 코드는 0x32식이다. 그러므로 이를 감안해서 lcd_dat() 서브루틴을 콜해야 LCD에 표시된다.

- 프로그램 예 :

```
#include "lcdchoi.h"
unsigned char hou, min, sec, onoff=1;
unsigned char dbufh[2], dbufm[2], dbufs[2];
unsigned int cnt0;
void h2b2asc(unsigned char hex, unsigned char *ptr) {          /* lcd에 출력하기 위해 아스키코드로 변환 */
      *ptr = (hex / 10) + '0';                  *(ptr+1) = (hex % 10) + '0';
}
void watch(void){
    if( ++sec == 60) { sec = 0;
            if( ++min == 60 ) { min = 0;
                    if( ++hou == 24 ) hou = 0;
```

```
                }
        }
        h2b2asc(sec, dbufs);        h2b2asc(min, dbufm);        h2b2asc(hou, dbufh);
        lcd_ln21();                                             // lcd 2행 1렬에 커서를 위치시킨다.
        lcd_str("    ");
        lcd_dat(dbufh[0]); lcd_dat(dbufh[1]);           /* 시간값 출력 */
        lcd_str("  ");
        lcd_dat(dbufm[0]); lcd_dat(dbufm[1]);           /* 분값 출력 */
        lcd_str("  ");
        lcd_dat(dbufs[0]); lcd_dat(dbufs[1]);           /* 초값 출력 */
}
interrupt [2] void exint0(void){  // interrupt [EXT_INT0] void exint0(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
        if(onoff == 0) onoff = 1;       else onoff = 0;
}
interrupt [3] void exint1(void){  // interrupt [EXT_INT1] void exint1(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
        if( ++min == 60 ) { min = 0; if( ++hou == 24 ) hou = 0; }
}
interrupt [4] void exint2(void){  // interrupt [EXT_INT2] void exint2(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
        if( ++hou == 24 ) hou = 0;
}
interrupt [17] void timerint0(void){  // interrupt [TIM0_OVF] void timerint0(void)로 해도 됨
        if(onoff == 0) if( ++cnt0 == 1000 ){     cnt0 = 0;          watch(); }
        TCNT0 = 6;          // 재정의 1msec = 64*(256-6)/16       시뮬레이션할 때는 조절할 필요가 있음
}
void main(void){
        DDRC = 0xff;         // 포트C를 출력으로
        DDRD = 0x00;        //포트D를 입력으로 디폴트 입력으로 설정되어 있으므로 필요하지는 않음
        SREG.7 = 1;         //인터럽트 전체 허용
        EIMSK |= 0x07;      // INT0,1,2 개별 허용
        EICRA |= 0x3f;      // 상승에지 트리거
        TIMSK |= 1;         // 타이머0 오버플로 인터럽트 개별허용
        TCCR0 |= 4;         // 분주비를 64로 설정                  시뮬레이션할 때는 조절할 필요가 있음
        TCNT0 = 6;          // 1msec = 64*(256-6)/16               시뮬레이션할 때는 조절할 필요가 있음
        lcd_init();
        lcd_str("  Hour Min Sec  ");
        for(;;);
}
```

- 과제 : ① lcd.h를 사용하여 같은 기능의 프로그램을 작성해보라. ②타이머1~3를 사용하여 프로그램 하여라. ③초값은 표시하지 않아 시간과 분만을 표시하는 시간조정이 가능한 시계용 프로그램을 작성해보아라.

5.5. 모터 제어

5.5.1. 스텝모터 개요

5.5.1.1. 특징

- 모터의 총회전각은 입력펄스수의 총수에 비례한다.
- 모터의 속도는 1초당 입력펄스수(펄스레이트)에 비례(보통수십kpps 정도가 사용됨)한다.
- 기동, 정지, 회전방향 변경, 변속이 쉽고 응답특성도 양호하다.
- 높은 유지토크를 내어 특정 위치에 정지할 수 있고 기동 정지 응답특성이 양호하여 서보모터로 사용가능하다.
- 1 스텝당 각도오차가 5% 이내이며 회전각의 오차가 누적되지 않는다.
- 회전각 검출이 필요 없어 제어기가 간단하고 가격이 상대적으로 저렴하다.
- 직류모터에서와 같이 브러쉬 교환 등 보수를 필요로 하지 않아 유지보수가 쉽고 부품수가 적어 신뢰성이 높다.

- 진동, 공진이 발생하기 쉽고 관성이 있는 부하에 약하다.
- 모터가 정지할 때 흔들이와 같이 진동하면서 정지. 진동음이 큰 경우도 종종 발생한다.
- 펄스레이트가 모터의 고유진동수의 정수분의 1일때 큰 진동이 발생한다.
- 통상 100pps 이하에서 공진이 발생하는 경우가 많고 탈조등의 현상이 발생한다.
- 고속운전시 모터 인덕턴스의 영향으로 권선에 충분한 전류가 흐르지 않아 토크가 저하하고 탈조하기 쉽다.

5.5.1.2. 종류

- 고정자 권선에 따른 종류 : 고정자에 있는 독립된 권선의 개수에 따라 2상,3상,4상등의 모터가 있고 2상은 저출력, 3상은 가변 릴럭턴스형 모터 4상은 고출력장치에 적용된다.
- 구조에 따른 종류 :

①영구자석형 : 회전자가 영구자석으로 고정자에 감긴 코일에 전류를 흘리면 회전자가 전류가 흘러진 고정자와 정렬하면서 회전하게 됨. 보통 스텝각이 1.8, 7.5, 15, 30, 34, 90도이다.

②가변릴럭턴스형 : 회전자가 연강재질로 톱니바퀴 형태. 고정자 권선에 전류를 가하면 회전자는 최소 자기저항을 갖도록 정렬하면서 회전하게됨. 보통 스텝각이 7.5, 15도이다. 회전자의 관성이 작고 고속응답특성이 우수하나 권선에 전류를 흘리지 않을 때 유지토크가 0임

③하이브리드형 : 영구자석형의 높은 효율과 가변릴럭턴스형의 작은 스텝각의 장점을 조합. 회전자가 원통형의 영구자석위에 톱니형의 규소강판이 적층되어 있음. 스텝각은 0.8, 1.8도임. 고정밀, 고출력토크용

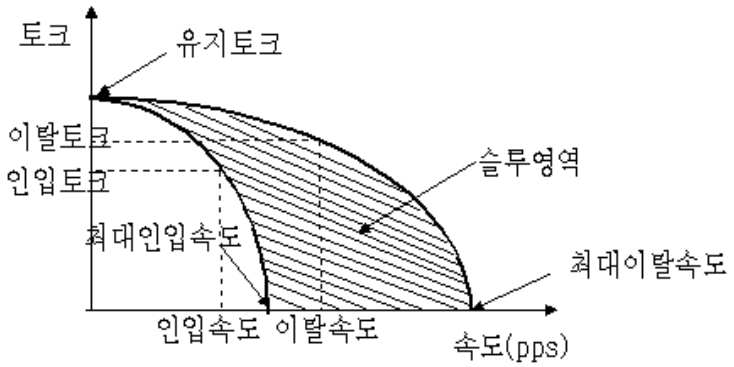


그림 101. 스텝모터의 토크 속도 특성

5.5.1.3. 특성과 사양

그림 101에 스텝모터의 속도와 토크의 관계가 보여졌다.

- 스텝각: 한 입력 펄스당 회전각
- 유지토크(holding torque): 정지상태에서 모터에 작용 가능한 최대 부하 토크
- 슬루영역(slew range) : 가감속 회전, 역회전, 정지, 기동이 펄스입력에 동기가능한 범위
- 최대인입속도: 무부하시 펄스입력에 동기되어 기동 가능한 최대 입력펄스 수
- 최대이탈속도: 무부하시 입력펄스에 동기되어 회전 가능한 최대 입력펄스 수

5.5.1.4. 4상 스텝 모터의 결선 확인법

4상 스텝 모터의 경우 선이 6개 혹은 5개 나와 있다. 전원을 A-B-C-D 순서로 가하면 시계 방향으로 회전한다고 가정하고 선이 6개나와 있다고 가정하자. 이럴 경우 결선들의 상은 아래에 따라 결정하면 된다.

①먼저 Ground 공통선 G(G1,G2)를 찾기 위해 저항계를 사용하여 저항이 무한대가 아닌 짝들을 찾아낼 수 있

을 것이다. 6개의 선이 나와있는 경우는 3개의 선들로 이루어진 짝들을 찾아 X짝 Y짝이라 하자. 그림 60의 경우에는 (A,G1,C)와 (B,G1,D)짝을 찾을 수 있다.

②먼저 X짝에서 저항계를 다시 이용하여 공통선을 찾고 다음 Y짝에서 공통선을 찾는다. 그림 102에서 (A,G1,C) 짝의 경우 (A,C) 간의 저항은 (A,G1) 그리고 (G1,C)간의 저항의 2배 정도가 될 것이다.

③Ground 공통선 G(G1,G2)를 연결하고 단계 1)에서 찾은 X와 Y짝 중 X짝에서 공통선이 아닌 두 선을 Xa와 Xb라 하고 Y짝에서도 마찬가지로 Ya와 Yb라 하자. 먼저 Xa와 공통선간에 정격전원을 가해본다. 전원을 끊고 Ya와 공통선간에 정격전원을 가하면 회전을 하게 되는데 시계 방향으로 회전하면 Xa는 A상 Xb는 C상 Ya는 B상 Yb는 D상으로 하면 된다. 반시계 방향으로 회전하면 Xa는 A상 Xb는 C상 Ya는 D상 Yb는 B상이다.

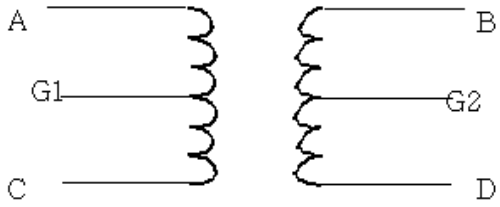


그림 102. 4상 스텝모터의 결선도

5.5.2. 4상 스텝 모터의 구동원리

5.5.2.1. 1상 여자 방법

각 스위치를 닫아 각 상에 전류가 흐르면 S극이 회전축 방향으로 자화가 된다고 하자. 먼저 sw1을 닫으면 그림 103의 왼쪽과 같이 되고 다음 sw1을 열고 sw2만 닫으면 회전자가 90도 회전하여 그림 103의 오른쪽과 같이된다. 마찬가지로 sw2를 열고 sw3만 닫으면 회전자가 90도 회전한다. 결국 A-B-C-D상 순서로 전류를 흘리면 모터가 회전함을 알 수 있다. 마찬가지로 D-C-B-A순서로 전류를 흘리면 모터가 반시계방향으로 회전한다. 1개의 상만 전류를 흘려서(여자시켰는데) 회전시켜 1상여자 방법이라 한다.

5.5.2.2. 2상 여자 방법

sw1과 sw2만 닫혔다고 하자. 그러면 그림 104의 왼쪽과 같이 될 것이다. 다음 sw1이 열리고 sw2와 sw3가 닫혔다고 하자. 그러면 회전자는 90도 회전하여 그림 104의 오른쪽과 같이 된다. 마찬가지로 sw2가 열고 sw3와 sw4가 닫으면 회전자는 90도 회전하게 된다. 결국 AB-BC-CD-DA상 순서로 전류를 흘리면 모터가 회전함을 알 수 있다. 마찬가지로 AB-AD-DC-CB-BA순서로 전류를 흘리면 모터가 반시계방향으로 회전한다. 이처럼 2개의 상을 여자시켜 스텝모터를 구동하는 방법을 2상여자 방법이라 한다. 1상여자에 비해 두배의 전력을 요하나 그 만큼 힘이 좋은 구동 방법이다

5.5.2.3. 1-2상 여자 방법

처음에 sw1만 닫고 그 다음 sw1과 sw2만 닫고 그 다음은 sw1을 열고 sw2만 닫고 이런 식으로 A-AB-B-BC-C-CD-D-DA-... 식으로 여자하면 45도씩 회전하게 된다. 이처럼 1개의 상 그 다음은 2개의 상을 번갈아 여자시켜 스텝모터를 구동하는 방법을 1-2상여자 방법이라 한다. 여자 순서가 복잡해지나 회전 각도의 resolution을 증가시킬 수 있는 방법이다

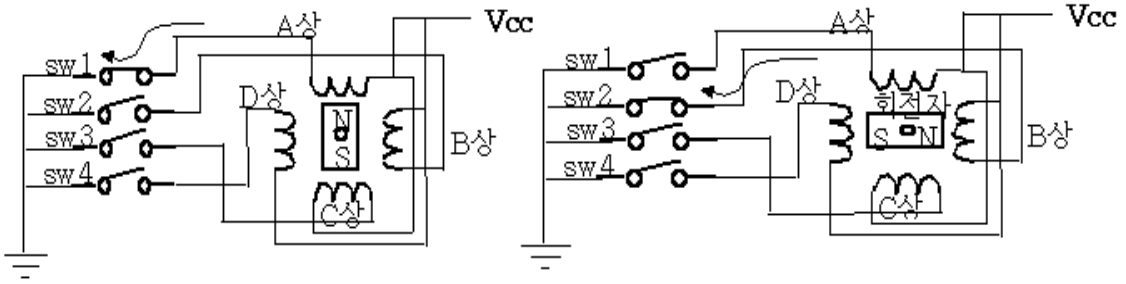


그림 103. 1상 여자 방법(왼쪽: sw1만 닫힌 경우, 오른쪽: sw2만 닫힌 경우)

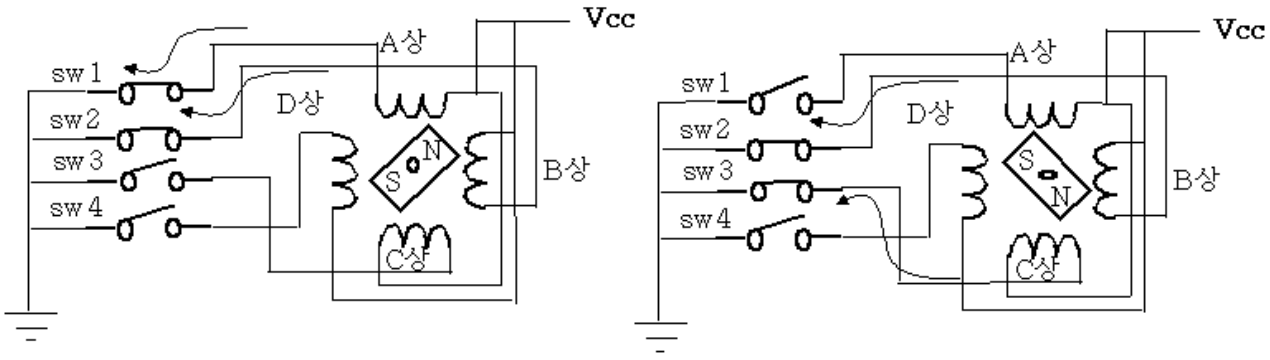


그림 104. 2상 여자 방법(왼쪽: sw1, sw2만 닫힌 경우, 오른쪽: sw2, sw3만 닫힌 경우)

5.5.3. 스텝 모터의 위치 제어법

스텝 모터는 여자를 A상에서 B상, B상에서 C상 이렇게 순서대로 함에 따라 스텝각이라 불리는 각도만큼씩 회전하게 되기 때문에 각도의 정밀도는 스텝각에 따라 의존하게 된다. 1상여자나 2상여자를 사용하여 구동하는 경우에는 스텝각도가 그 정밀도 한계가 되고 1-2상여자의 경우에는 스텝각의 1/2가 위치 제어의 정밀도 한계가 된다. 원하는 각도만큼 회전시키기 위해서는 그 각도를 정밀도 한계로 나누어 얻어진 값과 가장 근사한 정수만큼 여자순서를 바꾸어 주면 원하는 각도만큼 회전시켜 회전자를 위치시킬 수 있다.

- 4상의 스텝모터 1상여자 방법 : 에 의해 시계 방향으로 스텝각의 n배만큼 회전시키는 경우를 생각해보자. 이를 위해 마이콤을 사용하여 sw1, sw2, sw3, sw4를 열고 닫는 여자 패턴을 생성시켜 제어하는 경우 다음과 같이 여자패턴을 n번 만들어 출력포트로 내보내 주면 될 것이다

  10001000 -> 01000100(1번) -> 00100010(2번) -> 00010001(3번) -> .....(n번)

sw1, sw2, sw3, sw4의 작동은 출력포트 값중 하위4bit나 상위4bit중 아무 신호를 사용하여 작동시키면 된다. 그리고 위의 패턴으로부터 시계방향으로 n번 회전시킬 경우 시작값 10001000을 어떤 레지스터에 저장해서 출력하고 레지스터 값을 왼쪽으로 회전시켜서 어느 시간 간격후에 출력시키고 또 레지스터 값을 왼쪽으로 회전시켜서 어느 시간 간격후에 출력시키는 과정을 n번 반복 수행하는 식으로 마이콤 프로그램을 짜면 됨을 알 수 있다. 반시계방향의 경우는 오른쪽으로 레지스터 값을 회전시켜서 출력하면 된다.

- 2상여자의 경우 : 다음과 같은 여자패턴을 사용한다.

  11001100 -> 01100110(1번) -> 00110011(2번) -> 10011001(3번) -> .....(n번)

- 1-2상 여자의 경우 : 다음과 같이 여자 패턴을 발생시킨다. sw1, sw2, sw3, sw4의 작동은 출력포트 값중 1,3,5,7번째 bit의 신호를 사용하여 작동시키면 된다. 위의 1-2상 여자 패턴으로부터 처음에는 sw1만 닫히고 그 다음은 sw1과 sw2가 닫히고 그 다음은 sw2만 닫히고 그 다음은 sw2와 sw3가 닫히도록 하는 식으로 신호가 발생됨을 확인할 수 있다.

  11100000 -> 01110000(1번) -> 00111000(2번) -> 00011100(3번) -> .....(n번)

5.5.4. 스텝 모터의 속도 제어법

스텝 모터의 속도는 여자시키기 위해 각 상에 전원을 순서대로 끊고 연결시키는 간격에 따라 좌우된다. 즉 한 여자패턴에서 다음 스텝각으로 회전시키기 위한 여자패턴으로의 이동시간 간격에 따라 좌우된다. 예로 1상 여자방식으로 스텝모터를 마이콤을 사용하여 스텝각의 n배만큼 회전시키는 경우 다음과 같은 신호들이 출력되고

10001000 -> 01000100(1번) -> 00100010(2번) -> 00010001(3번) -> .....(n번)

각 신호의 값들이 생성되어 출력되는 간격이 좁으면 회전속도는 빠르고 간격이 넓으면 늦을 것이다. 만약 일정하다면 정속도 운전이 될 것이고 간격이 넓었다고 좁아지면 가속 운전이 되고 좁았다고 넓어지면 감속 운전이 될

것이다. 스텝모터는 기계적인 한계로 인해 최저 속도와 최고 속도가 사양으로 주어지기 때문에 원하는 응용에 맞는 규격을 갖는 스텝모터를 선정하고 이 한도 내에서 여자 간격을 줌으로써 안정적인 속도를 갖도록 제어할 수 있다.

5.5.5. 스텝 모터의 가감속 제어시 펄스레이트 결정법

가감속제어를 위해서는 여자패턴의 이동시간 간격을 제어해야 하는 데 간격을 결정하는 법을 설명하겠다. 초기속도 f_1 , 최종속도 f_N , 최종속도에 도달에 필요한 스텝수 N 이 주어졌고 그림 105와 같은 가속패턴이 요구된다고 가정하자. 감속의 경우는 f_N 과 f_1 을 바꿔놓으면 된다.

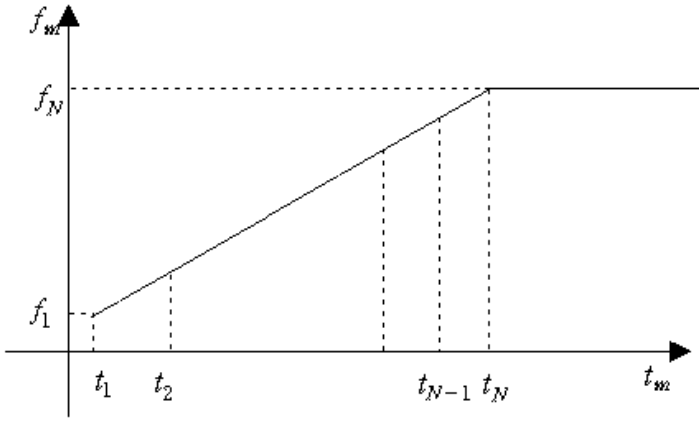


그림 105. 스텝모터의 가속 패턴

$t_1 = 1/f_1$, $t_k - t_{k-1} = 1/f_k$, $f_k - f_1 = \frac{(f_N - f_1)}{(t_N - t_1)}(t_k - t_1)$, $k = 2,...,N$, 이 성립하고 각 시간 간격마다 한 스텝씩 움직이는 것이므로 각각의 마름모의 면적은 1이 되어야 한다. 그러므로 다음의 수식이 성립한다.

$2(k-1) = (f_k + f_1)(t_k - t_1)$, $2(N-1) = (f_1 + f_N)(t_N - t_1)$

결국 다음의 수식을 얻을 수 있다.

$$f_k^2 - f_1^2 = \frac{(f_N^2 - f_1^2)(k-1)}{(N-1)} \quad , \quad t_k - t_{k-1} = \frac{1}{\sqrt{f_1^2 + \frac{(f_N^2 - f_1^2)(k-1)}{(N-1)}}}$$

예로 최종속도에 도달에 필요한 스텝수 $N = 10$ 으로 초기속도와 최종속도가 pps로 각각 $f_1 = 500$, $f_{10} = 1000$ 일 때 스텝인덱스 k와 이에 따른 각 스텝에서의 속도와 시간간격은 표15과 같다.

표 15. $f_1 = 500$, $f_{10} = 1000$ 일 때 k에 따른 속도

k	f_k(pps)	$t_k - t_{k-1}$(msec)	t_k(msec)
1	500.0	2.0000	2.0000
2	577.4	1.7321	3.7321
3	645.5	1.5491	5.2812
4	707.1	1.4142	6.6955
5	763.8	1.3093	8.0048
6	816.5	1.2247	9.2295
7	866.0	1.1547	10.3842
8	912.9	1.0954	11.4797
9	957.4	1.0445	12.5241
10	1000	1.0000	13.5241

5.5.6. 모터 구동용 H브리지 회로

그림 106에 보여진 회로는 H브리지 회로로 여러가지 모터를 구동하기 위해 쓰일 수 있다.

- C3205Y와 A1273Y 쌍의 역할 : 출력으로 유출입되는 전류의 흐름을 통제하는 on-off 스위치의 역할을 한다. IN1 입력이 1(5V level)일 때 C3205Y은 off 상태 A1273Y는 on 상태로 되어 OUT1 출력이 GND에 연결되고, IN1 입력이 0(0V level)일 때 C3205Y은 켜지고 A1273Y는 꺼져 OUT1 출력은 VCC에 연결된다. 그래서 부하를 OUT1과 OUT2사이에 연결하고 IN1에 1레벨을 입력시키고 IN2에 0레벨을 입력시키면 부하에는 전류가 OUT2 측에서 흘러들어가 OUT1 측으로 흘러 나가게 된다. 2SD560 (파워TR:npn)과 2SB601 쌍으로 혹은 BDX53C(npn)과 BDX54C 쌍으로 대체가능하다.

- (0.5옴[1W], 2SC1815Y)과 (0.5옴[1W], 2SA1015Y)의 역할 : 출력으로 유출(유입)되는 전류가 어느 한도를 넘으면 2SC1815Y(혹은 2SA1015Y)가 도통이 되고 C3205Y(혹은 A1273Y)이 꺼지게 되어 출력으로 유출(유입)되는 전류를 제한한다. 주어진 회로도에서는 1.2 A(=0.6V/0.5옴)가 한도가 된다. 저항값을 조정함으로써 출력쪽으로의 유출입 전류 상한값을 설정할 수 있다.

- FR105의 역할 : 스위치 역할을 하는 C3205Y(혹은 A1273Y)가 손상되는 것을 방지하는 환류다이오드의 역할은 한다. 이들 트랜지스터가 켜져있다가 갑자기 꺼지게 되면 출력의 부하가 인덕턴스 성분을 갖는 모타의 경우 부하 전류가 급격히 변화할 수 없는 상황에서 전류가 흐를 곳이 없어 트랜지스터를 파괴하고 흐를 소지가 있다. 이 때 다이오드 FR105가 전류의 통로를 마련해준다.

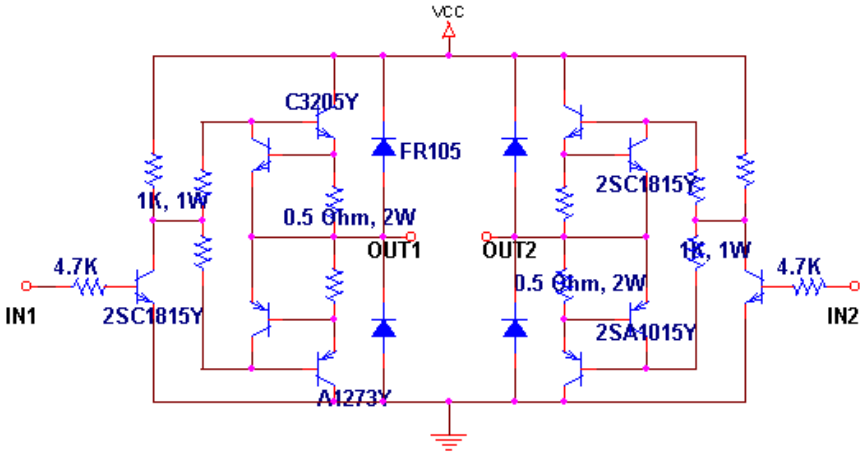


그림 106. 다목적 H브리지 회로의 예

5.5.7. 모터 구동용 IC L298의 개요

- 특징: A와 B 두개의 H브리지 회로 내장, 회로가 간단하고 부피가 적음, 최대 전류 2.5A, 평균 전류 0.2-0.25A, 2000~4000 pps 구동이 가능. 환류다이오드를 외부에서 달아 주어야 함.

- 핀배치 : 8번(GND), 6번(ENA:A 브리지회로의 인에블 단자로 1레벨이면 동작허용), 5번(IN1:A 브리지회로의 입력1), 2번(OUT1:A 브리지회로의 출력1), 7번(IN2: A 브리지회로의 입력2), 3번(OUT1:A 브리지회로의 출력2), 10번(IN3:B 브리지회로의 입력1), 11번(ENB:B 브리지회로의 인에블 단자로 1레벨이면 동작허용),13번(OUT3:B 브리지회로의 출력1), 12번(IN4: B 브리지회로의 입력2), 14번(OUT4:B 브리지회로의 출력2), 4번(VS: 모터측 전원으로 마이컴측 전원보다 2.5V이상 커야 함, GND와 사이에 100nF 콘덴서를 필히 연결해야 함), 9번(VSS: 마이컴측 전원단자. GND와 사이에 100nF 콘덴서를 필히 연결해야 함), 1번(ISENA:A 브리지회로에 흐르는 부하 전류를 측정하기 위해 GND와 사이에 저항을 삽입하기 위한 단자, 과부하 전류 차단용 제어회로나 PWM 제어에 활용 가능), 15번(ISENB)

- 동작방법 : IN1=1레벨, IN2=0레벨, ENA=1레벨시 A브리지회로가 동작하고 OUT1에서 전류가 흘러나와 OUT2쪽으로 전류가 흘러들어간다. IN1=0레벨, IN2=1레벨, ENA=1레벨시 A브리지회로가 동작하고 OUT2에서 전류가 흘러나와 OUT1쪽으로 전류가 흘러들어간다. IN1=IN2이며 ENA=1레벨시 급속하게 정지되고 ENA=0레벨시 입력에 관계없이 천천히 정지한다. B브리지회로도 마찬가지이다. 결국 ENx를 1레벨로 고정시키고 INx입력

을 조정하여 모터의 정지, 운전, 회전방향을 제어할 수도 있고, INx를 어느 한방향으로 전류가 흘러들어가게 고정시키고 ENx를 조정하여 회전방향은 일정하게 놓고 모터의 정지와 운전만을 제어할 수도 있고, ENx와 INx를 모두 조정하여 모터의 정지, 운전, 회전방향을 제어할 수도 있다.

5.5.8. 실험52 : 스텝 모터 정속 제어

- 예제 : PORTD.0에 연결된 스위치가 홀수번 눌리면 C포트에 연결된 스텝모터를 500pps로 스텝모터를 회전시킨다. 짝수번 눌리면 정지한다.
- 프로그램 작성시 유의점 : 500pps로 회전시키기 위해서는 2msec마다 여자 패턴을 바꾸어 주어야 한다. 타이머 0의 노말모드를 이용하여 2msec 마다 인터럽트를 발생시키고 여자 패턴을 바꾸어준다. 1상여자로 한다고 가정한다. 그림 107의 회로에서는 ENx를 1레벨에 고정시켰음에 유의해서 작성해야 한다.

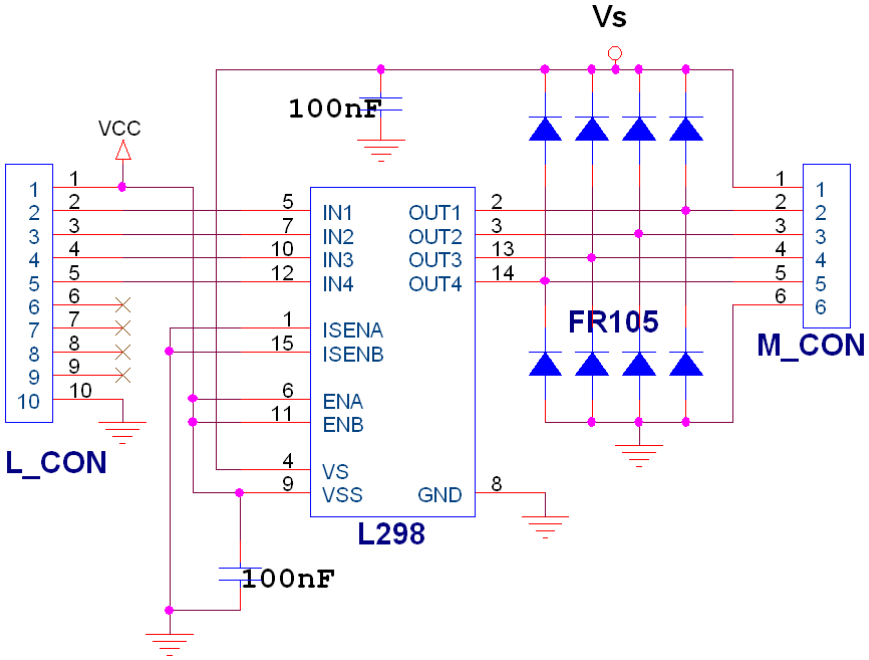


그림 107. 모터 실험 회로도

- 회로도 : 그림 107의 회로를 구현하고 L_CON을 그림 79의 C_CON에 연결한다. 그림 107의 MOTOR_CON을 모터측 전원과 A,B,C,D 상과 GND에 연결한다. ISENx 단자를 GND에 직접 연결하여 부하전류의 궤환을 하지 않는다. 그림 80의 SW_CON을 그림 79의 D_CON에 연결한다. 또한 그림 80의 SW1를 누르면서 실험한다.
- 프로그램 예 :

```
#include <mega128.h>
unsigned int cnt0;
unsigned char led=0x88, onoff=1;
interrupt [2] void exint0(void){  // interrupt [EXT_INT0] void exint0(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
      if(onoff == 0) onoff = 1; else onoff = 0;
}
interrupt [17] void timerint0(void){  // interrupt [TIM0_OVF] void timerint0(void)로 해도 됨
      if(onoff == 0) if( ++cnt0 == 2 ){
      cnt0 = 0;  led >>= 1; if(led == 0x08) led=0x88; }
      PORTC = led;
      TCNT0 = 6;          // 재정의 1msec = 64*(256-6)/16       시뮬레이션할 때는 조절할 필요가 있음
}
void main(void){
      DDRC = 0xff;         // 포트C를 출력으로
      DDRD = 0x00;        //포트D를 입력으로 디폴트 입력으로 설정되어 있으므로 필요하지는 않음
      SREG.7 = 1;         //인터럽트 전체 허용
```

```
    EIMSK |= 0x01;      // INT0 개별 허용
    EICRA |= 3;         // 상승에지 트리거
    TIMSK |= 1;         // 타이머0 오버플로 인터럽트 개별허용
    TCCR0 |= 4;         // 분주비를 64로 설정                    시뮬레이션할 때는 조절할 필요가 있음
    TCNT0 = 6;          // 1msec = 64*(256-6)/16                 시뮬레이션할 때는 조절할 필요가 있음
    for(;;);
}
```

- 과제 : 타이머1~3을 이용하여 작성해보라.

5.5.9. 실험53 : 스텝 모터 속도 제어

- 예제 : PORTD.0에 연결된 스위치가 홀수번 눌리면 C포트에 연결된 스텝모터를 500pps로 스텝모터를 회전시킨다. 짝수번 눌리면 정지한다. PORTD.1에 연결된 스위치가 눌리면 속도가 증가하고 PORTD.2에 연결된 스위치가 눌리면 감속된다.
- 프로그램 작성시 유의점 : 500pps로 회전시키기 위해서는 2msec마다 여자 패턴을 바꾸어 주어야 한다. 타이머 0의 노말모드를 이용하여 인터럽트를 발생시키고 여자 패턴을 바꾸어준다. 1상여자로 한다고 가정한다. 외부 인터럽트를 사용하여 가감속을 제어한다.
- 회로도 : 그림 107의 회로를 구현하고 L_CON을 그림 79의 C_CON에 연결한다. 그림 107의 MOTOR_CON을 모터측 전원과 A,B,C,D 상과 GND에 연결한다. ISENx 단자를 GND에 직접 연결하여 부하전류의 궤환을 하지 않는다. 그림 80의 SW_CON을 그림 79의 D_CON에 연결한다. 또한 그림 80의 SW1~3를 누르면서 실험한다.
- 프로그램 예 :

```
#include <mega128.h>
unsigned int cnt, delaytime, step, delaymax, delaymin;
unsigned char led=0x88, onoff=1;
interrupt [2] void exint0(void){  // interrupt [EXT_INT0] void exint0(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
    if(onoff == 0) onoff = 1; else onoff = 0;
}
interrupt [3] void exint1(void){  // interrupt [EXT_INT1] void exint1(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
    if( delaytime + step <= delaymax ) delaytime += step;
}
interrupt [4] void exint2(void){  // interrupt [EXT_INT2] void exint2(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
    if( delaytime - step >= delaymin ) delaytime -= step;
}
interrupt [17] void timerint0(void){  // interrupt [TIM0_OVF] void timerint0(void)로 해도 됨
    if(onoff == 0) if( cnt-- == 1 ){
    cnt = delaytime;
    led >>= 1; if(led == 0x08) led=0x88; }
    TCNT0 = 231; // 재정의 1msec = 64*(256-6)/16      시뮬레이션할 때는 조절할 필요가 있음
    PORTC = led;
}
void main(void){
    DDRC = 0xff; // 포트C를 출력으로
    PORTC = led;
    SREG |= 0x80;      //인터럽트 전체 허용
    EIMSK |= 0b00000111;        // INT0, 1,2 개별 허용
    EICRA |= 0x3f;              // INT0, 1,2 상승에지 트리거
    TIMSK |= 1;         // 타이머0 오버플로 인터럽트 개별허용
    TCCR0 |= 4;         // 분주비를 64로 설정
    TCNT0 = 231;        // 64*(256-231)/16=100usec = 0.1msec
    delaytime = 20;     cnt = delaytime;                    /* 점멸주기를 20*0.0001 = 2msec로 초기화 */
    delaymax = 40;      delaymin = 1;                 /*점멸회전주기의 최대, 최소값을 4m, 0.1m초로 초기화 */
    step = 2;                                        /* 주기 변경 폭을 10으로 초기화 */
    while(1);
}
```

- 과제 : 최소 200pps 최대 2000pps 사이에서 작동하도록 프로그램을 변경하여 보아라.

5.5.10. 실험54 : 스텝 모터 가감속 제어

- 예제 : PORTD.0에 연결된 스위치가 한번 눌리면 포트C에 연결된 스텝모터를 500pps에서 1000pps로 10단계에 걸쳐 가속시켜 그 이후에 1000pps로 운전하고 또 눌리면 반대로 감속시켜 정지시킨다.
- 프로그램 작성시 유의점 : 표15에 주어진 $t_k - t_{k-1}$를 프로그램메모리에 저장시키고 이를 불러다가 사용한다.
- 회로도 : 회로도 : 그림 107의 회로를 구현하고 L_CON을 그림 79의 C_CON에 연결한다. 그림 107의 MOTOR_CON을 모터측 전원과 A,B,C,D 상과 GND에 연결한다. ISENx 단자를 GND에 직접 연결하여 부하전류의 궤환을 하지 않는다. 그림 80의 SW_CON을 그림 79의 D_CON에 연결한다. 또한 그림 80의 SW1를 누르면서 실험한다.
- 프로그램 예:

```
#include <mega128.h>
#include <delay.h>
unsigned char led=0x88, status, vtblidx;
unsigned int vtbl[ ] = {2000,1732,1549,1414,1309,1225,1155,1095,1045,1000 };
unsigned int    cnt, cvt;
interrupt [17] void timerint0(void){  // interrupt [TIM0_OVF] void timerint0(void)로 해도 됨
      if(cnt++ >= cvt){
                cnt = 0;
                if(status !=0){
                          if(vtblidx<=8) vtblidx++;
                          led >>= 1; if(led == 0x08) led=0x88;
                }
                else {
                          if(vtblidx>0) {
                          vtblidx--;
                          led >>= 1; if(led == 0x08) led=0x88;}
      }         }
      TCNT0 = 240;                          // 재정의 1usec = 1*(256-240)/16          16MHz와 분주비 1사용시
      PORTC = led;
}
void main(void){
      DDRC = 0xff;                                    // 포트C를 출력으로
      PORTC = led;
      SREG |= 0x80;                                   //인터럽트 전체 허용
      TIMSK |= 1;                                     // 타이머0 오버플로 인터럽트 개별허용
      TCCR0 |= 1;                                     // 분주비를 1로 설정
      TCNT0 = 240;
      for(;;){
loop:           while(PIND.0 == 1);                   /* PORTD.0가 눌릴때까지 대기 */
      delay_ms(50);                                   /* 50msec 시간지연 */
      if(PIND.0 == 1)  goto          loop;            /*눌림이 50msec동안 지속 안되면 무효 다시 키입력을 받는다*/
      while(PIND.0 !=1);                              /* PORTD.0가 해제될 때까지 대기 */
      cvt  = vtbl[vtblidx];
      status = ~status;
      }
}
```

5.5.11. 직류모터 개요

5.5.11.1. 특징

- 기동토크가 크고 입력 전압에 비례하는 회전속도를 얻을 수 있다.

- 토크에 대하여 회전속도는 반비례하는 특성을 갖고 있다.
- 입력전류에 대하여 출력토크가 직선적이며 출력 효율도 양호하며 가격이 저렴하다.
- 양호한 토크-속도, 토크-전류 특성으로 광범위한 속도 제어와 여러 다양한 제어방법의 적용이 가능하다.
- 직류 모터는 높은 회전력, 광범위한 속도 제어 능력, 휴대 용이, 속도 회전력 특성 양호, 여러 다양한 제어 방법 적용가능 등의 특성 때문에 로봇, 카셋트 테이프, 하드디스크, 기계공구 등 광범위하게 사용되고 있다.
- 브러시와 정류자의 기계식 접점때문에 소음과 노이즈, 먼지가 많이 발생하며 브러시등을 교환해주어야 하기 때문에 유지보수가 스텝모터에 비해 쉽지않다(브리시리스 직류모터는 제외).

5.5.11.2. 직류모터 구동원리

그림 108의 왼쪽그림에서 만약 AB단자에 +Vcc를 걸면 플레밍의 왼손 법칙에 의해 A단자측 도선은 아래쪽으로 힘이 작용하고 B단자측 도선은 위쪽으로 힘이 작용하여 aa'축에 대하여 반시계 방향으로 회전력이 발생한다. 만약 도선을 여러 개로 하여 aa'회전축으로 회전 가능한 원통에 방사상으로 배치하여 고정시킨 그림108의 오른쪽과 같은 경우 bb'의 좌측 도선은 지면으로 흘러 들어가는 방향으로 전류를 흘리고 bb'의 우측 도선은 지면에서 흘러 나오는 방향으로 전류를 흘리면 aa'축을 중심으로 하여 반시계 방향으로 회전력이 발생하여 도선이 고정된 원통은 반시계 방향으로 회전할 수 있다. 어떤 장치를 사용하여 bb'의 좌측에 위치하는 도선은 항상 지면으로 흘러 들어가게 전류를 주고 bb'의 우측에 위치하는 도선은 항상 지면에서 흘러 나오게 전류를 주면 계속적으로 원통을 회전하게 할 수 있다.

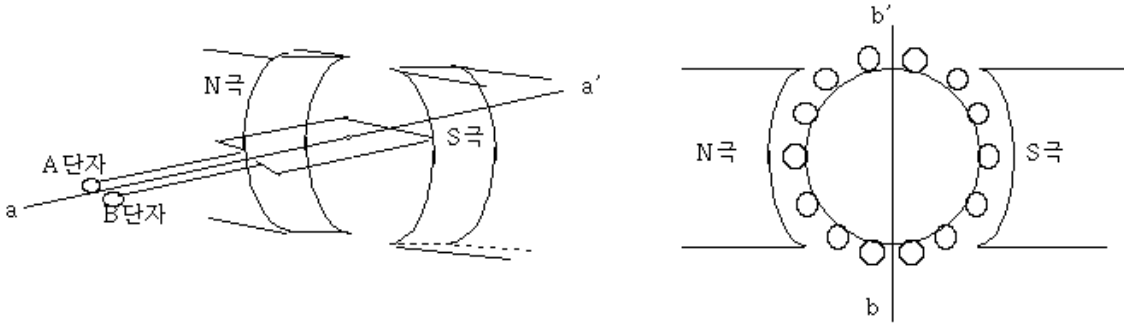


그림 108. 직류모터의 구동원리

5.5.11.3. 직류모터 구조

그림 108에 보여진 것과 구조가 유사하여 영구 자석으로 이루어진 고정자가 있고 도선이 감긴 원통형의 회전자가 있으며 회전자의 도선에 흐르는 전류의 방향을 bb'축의 좌측 또는 우측에 위치하느냐에 따라 일정한 방향으로 흘러 들어가게 하는 정류자와 브러시라는 장치로 구성되어 있다. 회전자가 영구자석으로 이루어지고 고정자에 도선을 감은 형태의 직류모터도 있다.

5.5.12. 실험55 : 직류모터 속도 제어1

- 예제 : PORTD.0에 연결된 스위치가 홀수번 눌리면 C포트에 연결된 직류모터를 회전시킨다. 짝수번 눌리면 정지한다. PORTD.1에 연결된 스위치가 눌리면 속도가 증가하고 PORTD.2에 연결된 스위치가 눌리면 감속된다.
- 프로그램 작성시 유의점 : 타이머 0의 노말모드를 이용하여 인터럽트를 발생시켜 출력을 온오프시킨다. 외부 인터럽트를 사용하여 온오프의 듀티비를 조절하여 가감속을 제어한다.
- 회로도 : 그림 107의 회로를 구현하고 L_CON을 그림 79의 C_CON에 연결한다. 그림 107의 MOTOR_CON의 2번을 모터측 A단자에 GND를 모터측 B단자에 연결한다. ISENx 단자를 GND에 직접 연결하여 부하전류의 궤환을 하지 않는다. 그림 80의 SW_CON을 그림 79의 D_CON에 연결한다. 또한 그림 80의 SW1~3를 누르면서 실험한다.
- 프로그램 예:

```
#include <mega128.h>
unsigned int cnt, step=1, delaymax=10, delaymin=1, speed=5;
unsigned char onoff=1;
interrupt [2] void exint0(void){  // interrupt [EXT_INT0] void exint0(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
     if(onoff == 0) onoff = 1; else onoff = 0;
}
interrupt [3] void exint1(void){  // interrupt [EXT_INT1] void exint1(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
     if( speed + step < delaymax ) speed += step;
}
interrupt [4] void exint2(void){  // interrupt [EXT_INT2] void exint2(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
     if( speed - step >= delaymin ) speed -= step;
}
interrupt [17] void timerint0(void){  // interrupt [TIM0_OVF] void timerint0(void)로 해도 됨
     if(onoff == 0) { if( cnt-- == 0 ) cnt = delaymax;
               if(speed <= cnt ) PORTC.0 = 1;
               else PORTC.0 = 0; }
     else PORTC.0 = 0;
     TCNT0 = 6;                    // 재정의 1msec = 64*(256-6)/16      시뮬레이션할 때는 조절할 필요가 있음
}
void main(void){
     DDRC = 0xff;                  // 포트C를 출력으로
     PORTC = 0x00;
     SREG |= 0x80;                 //인터럽트 전체 허용
     EIMSK |= 0b00000111;          // INT0, 1,2 개별 허용
     EICRA |= 0x3f;                // INT0, 1,2 상승에지 트리거
     TIMSK |= 1;                   // 타이머0 오버플로 인터럽트 개별허용
     TCCR0 |= 4;                   // 분주비를 64로 설정
     TCNT0 = 6;                    // 64*(256-6)/16=1000usec = 1msec
     while(1);
}
```

- 과제 : 타이머0 대신 타이머 1~3을 사용해서 작성해 보아라.

5.5.13. 실험56 : 직류모터 속도 제어2

- 예제 : 타이머3의 PWM모드를 이용해서 직류모터의 속도를 제어한다. PORTD.0에 연결된 스위치가 눌리면 속도가 감소하고 PORTD.2에 연결된 스위치가 눌리면 증가된다.
- 프로그램 작성시 유의점 : 타이머 3의 출력비교 모드를 이용하여 PWM 신호를 발생시켜 출력을 온오프시킨다. 외부 인터럽트를 사용하여 온오프의 듀티비를 조절하여 가감속을 제어한다.
- 회로도 : 그림 107의 회로를 구현하고 L_CON을 그림 79의 E_CON에 연결한다. 그림 107의 MOTOR_CON의 2번을 모터측 A단자에 GND를 모터측 B단자에 연결한다. ISENx 단자를 GND에 직접 연결하여 부하전류의 궤환을 하지 않는다. 그림 80의 SW_CON을 그림 79의 D_CON에 연결한다. 또한 그림 80의 SW1~2를 누르면서 실험한다.
- 프로그램 예:

```
#include <mega128.h>
unsigned int step=50;
interrupt [2] void exint0(void){  // interrupt [EXT_INT0] void exint0(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
unsigned int ocr3a;
        ocr3a = 256*OCR3AH+OCR3AL;
     if( ocr3a > step ) {
               ocr3a -= step;
               OCR3AH = ocr3a/256;
               OCR3AL = ocr3a%256;}
}
interrupt [3] void exint1(void){  // interrupt [EXT_INT1] void exint1(void) 처럼 mega128.h에 저장된 정의를 사용해도 됨
```

```
unsigned int ocr3a;
        ocr3a = 256*OCR3AH+OCR3AL;
    if( ocr3a < 0xffff - step ) {
            ocr3a += step;
            OCR3AH = ocr3a/256;
            OCR3AL = ocr3a%256;}
}
void main(void){
    DDRE.3 = 1;                 // PORTE bit 4를 출력으로
    SREG |= 0x80;               //인터럽트 전체 허용
    EIMSK |= 0b00000111;        // INT0, 1,2 개별 허용
    EICRA |= 0x3f;              // INT0, 1,2 상승에지 트리거
    TCCR3B |= 2;                // 분주비를 8로 설정
    TCCR3A |=2;                 // 타이머3 PC PWM 모드 14로 설정
    TCCR3B |= 0x10;             // 타이머3 PC PWM 모드 14로 설정
    TCCR3A |= 0x80;             // 출력비교모드 설정
    OCR3AH = 0x7f;
    OCR3AL = 0xff;
    ICR3H = 0xff;
    ICR3L = 0xff;
    for(;;);
}
```

5.6. DA 변환

5.6.1. DA변환기의 종류

디지털-아날로그 변환기는 2진부호로 표현된 디지털량을 그에 대응하는 직류전압으로 변환하는 회로로 그림 109에 보여진 것처럼 가산형과 사다리형이 있다.

①가산형(2진 하중 저항형): n 비트의 디지털값을 아날로그량으로 변환하기 위해 $R, 2R, 4R, \cdots, 2^{n-1}R$ 값의 저항들과 연산증폭기를 사용하여 가산증폭기 형태로 구성한 D/A변환기로 정밀한 변환을 위해서는 최대저항과 최소저항 사이의 저항비가 정밀해야하고 비트수가 많아지면 최대저항과 최소저항 사이의 저항비가 너무 커지며 각 디지털입력은 서로 다른 부하를 갖는 단점을 지녀 4비트 이상의 D/A변환에 거의 사용되지 않는다. 그림 109(왼쪽)에 4비트의 디지털값을 아날로그량으로 변환하기 위한 가산형 D/A변환기가 보여졌다. 1레벨 입력전압이 V_r 일 때 출력값 V_o 는 디지털값 b_i 에 의존하여 다음과 같이 주어진다

$$V_o = -\frac{R_F}{R}\left(\frac{b_1}{1} + \frac{b_2}{2} + \frac{b_3}{4} + \frac{b_4}{8}\right)V_r$$

②사다리형(R-2R형): $R, 2R$ 두 종류의 저항과 연산증폭기를 사용하여 구성한 D/A 변환기로 고정도로 만들 수 있다. 저가로 많이 사용되는 내셔날세미컨덕터사의 DAC08시리즈 D/A변환기가 사다리형의 대표적인 예이다. 그림 62(오른쪽)에 4비트의 디지털값을 아날로그량으로 변환하기 위한 사다리형 D/A변환기가 보여졌다. 1레벨 입력전압이 V_r 일 때 출력값 V_o 는 디지털값 b_i 에 의존하여 다음과 같이 주어진다.

$$V_o = \left(\frac{b_1}{2} + \frac{b_2}{4} + \frac{b_3}{8} + \frac{b_4}{16}\right)V_r$$

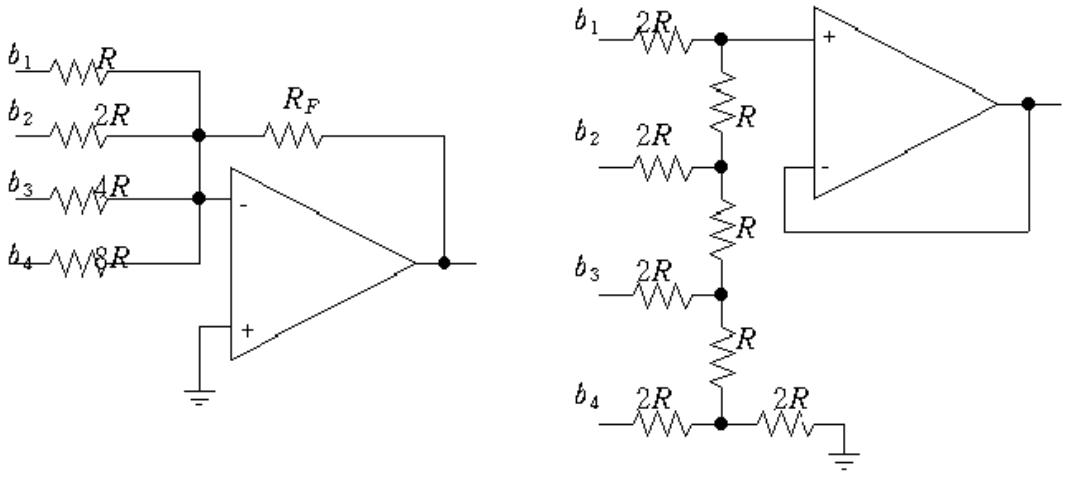


그림 109. D/A변환기의 종류. 가산형(왼쪽)과 사다리형(오른쪽)

5.6.2. DAC 0800소개

- 특징: 8비트 분해능, 넓은 범위의 전원(4.5~18V), TTL이나 CMOS와 직접 연결가능, 100 nsec의 변환시간, 저전력소비(5V에서 33mW)
- 핀배치 : 1번(VLC:논리임계값 제어핀으로 보통 GND에 연결한다.), 2번(/IOUT:전류출력핀), 3번(V-:-전원단자), 4번(IOUT:전류출력핀), 5-12번(D7-D0입력단자) 13번(V+:+ 전원단자), 14번(VREF+:최대출력전압), 15번(VREF-:최저출력전압), 16번(COMP:보정용 단자로 콘덴서를 통해 V-에 연결)

5.6.3. 실험57 : 톱니파 발생

- 예제 : PORTD.0에 연결된 누름 스위치가 눌리면 톱니파가 발생한다.
- 회로도 : 그림 110의 회로를 구현하고 DAC_CON을 그림 79의 C_CON에 연결한다. 또한 그림 80의 회로를 그림 79의 D_CON에 연결한다. SW1을 누르면서 실험한다.
- 프로그램 예 :

```
#include <mega128.h>
#include <delay.h>
void main(void){
      unsigned char i=0xff;
      DDRC = 0xff;        // PORTC 를 출력으로
loop:          while(PIND.0 == 1);          /* PORTD.0가 눌릴때까지 대기 */
               delay_ms(50);                                  /* 50msec 시간지연 */
      if(PIND.0 == 1)  goto       loop;     /*눌림이 50msec동안 지속 안되면 무효 다시 키입력을 받는다*/
      while(PIND.0 !=1);                          /* PORTD.0가 해제될 때까지 대기 */
      for(;;i--){
      PORTC = i;
      delay_ms(20);                               /* 20msec 시간지연 */
      }
}
```

5.6.4. 실험58 : 사인파 발생

- 예제 :PORTD.0에 연결된 누름 스위치가 눌리면 사인파가 발생한다.
- 프로그램 작성시 유의점 : 삼각함수와 관련된 명령을 사용하지 않고 사인값을 100등분하여 구하고 표로 만들어 롬에 저장시켜 놓고 사용한다.
- 그림 110의 회로를 구현하고 DAC_CON을 그림 79의 C_CON에 연결한다. 또한 그림 80의 회로를 그림 79의

D_CON에 연결한다. SW1을 누르면서 실험한다.

● 프로그램 예 :

```
#include <mega128.h>
#include <delay.h>
flash unsigned char sin_tbl[ ] = {128, 136, 144, 151, 159, 167, 175, 182, 189, 196, 203, 209, 215, 221, 226,
      249, 247, 243, 240, 236, 231, 226, 221, 215, 209, 203, 196, 189, 182, 175,
      167, 159, 151, 144, 136, 127,          119, 111, 104, 96, 88, 80, 73, 66, 59,
      52, 46, 40, 34, 29, 24, 19, 15, 12, 8, 6, 4, 2, 1, 0, 0, 0, 1, 2, 4,
      6, 8, 12, 15, 19, 24, 29, 34, 40, 46, 52, 59, 66, 73, 80, 88, 96, 104, 111, 119 };
void main(void){
      unsigned char i=0;
      DDRC = 0xff;       // PORTC 를 출력으로
loop:          while(PIND.0 == 1);          /* PORTD.0가 눌릴때까지 대기 */
      delay_ms(50);                                  /* 50msec 시간지연 */
      if(PIND.0 == 1)  goto          loop;      /*눌림이 50msec동안 지속 안되면 무효 다시 키입력을 받는다*/
      while(PIND.0 !=1);                             /* PORTD.0가 해제될 때까지 대기 */
      for(;;){
      PORTC = sin_tbl[i];
      delay_ms(20);                                  /* 20msec 시간지연 */
      if(i<=98) i++;
      else i=0;}
}
```

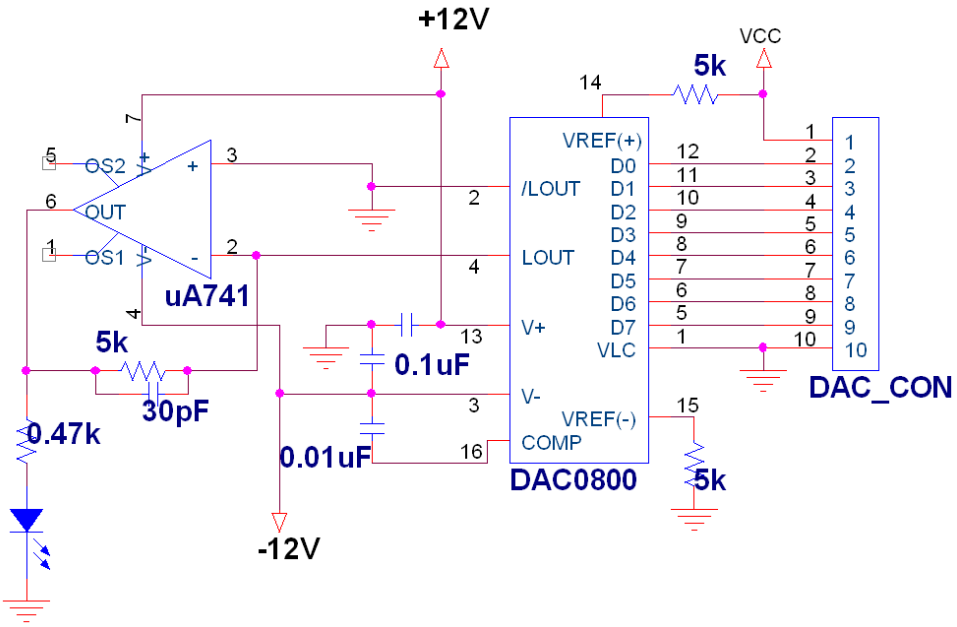


그림 110. D/A 변환 실험 회로도

5.7. AD 변환

5.7.1. AD변환과정

AD의 변환은 다음 과정을 통해 이루어진다.

①전처리: 아날로그 신호에 포함된 잡음을 제거하고 신호의 대역폭을 제한하여 엘리어싱(aliasing)을 줄인다.

②표본화: 신호 대역폭 두배이상의 일정한 샘플링(sampling) 주파수에 따라 신호값을 취하여, 즉 신호에 포함된 최고 주파수 성분 주기의 1/2보다 작은 주기로 신호값을 취하여 저장한다.

③양자화: 표본화된 아날로그 신호는 연속적인 양으로 이를 2진화하면 무한한 자리수를 요구할 수도 있다. 그러므로 표본화된 값을 소구간으로 분할하여 유한한 단계의 유한한 자리수를 갖는 불연속적인 대표값에 할당할 필요가 있는 데 이 것을 양자화라 한다. 양자화 단계의 갯수가 많으면 실제값과 대표값의 차인 양자화오차를 줄일 수 있으나 디지털 출력의 비트수가 증가하여 이를 처리하기 위한 시간과 장치가 많이 요구된다.

④부호화: 양자화된 값에 2진 디지털 코드를 부여하는 것을 부호화라 한다.

5.7.2. AD변환기의 종류

AD 변환기는 적분방식과 비교방식으로 크게 나뉠 수 있고 적분방식은 저속으로 변환이 이루어지는 데 전압시간변환, 이중적분, 전압주파수변환형이 있고 비교방식은 적분방식에 비해 고속으로 변환이 이루어지는 데 축차비교, 추종비교, 병렬비교형이 있다.

①전압시간변환형: 기준신호를 적분한 것과 샘플링된 입력전압을 비교하여 입력전압의 크기에 비례하는 계수시간을 결정하고 그 시간 동안 기준펄스의 수를 계수하여 변환하는 방식으로 비교적 간단하나 빠른 속도와 높은 정도를 기대하기 힘들다.

②이중적분형: 샘플링된 +의 입력전압을 미리 설정된 일정한 시간동안 적분한 값을 초기치로 하여 -의 기준신호를 적분한 값이 0에 도달할 때까지 기준펄스의 수를 계수하여 변환하는 방식으로 전압시간변환형보다는 안정되어 디지털 전압계 등에 많이 활용된다.

③전압주파수변환형: 샘플링된 입력신호에 비례하는 반복 주파수 펄스를 발생시키고 카운터로 계수하여 A/D 변환한다.

④계수비교형: 샘플링회로, D/A변환기, 업카운터, 비교기로 구성되어 있다. 0부터 차례로 증가시킨 카운터의 값을 D/A 변환기로 아날로그값으로 변환하여 샘플링된 입력신호와 비교기에서 비교하여 같아지면 업카운터의 카운팅을 중지시켜 A/D변환한다. 최악의 경우 n 비트형의 경우 2^n-1 번 비교를 실행해야 하므로 변환속도가 느리나 회로구성이 비교적 간단하고 가격이 저렴하다.

⑤축차비교형(연속근사형): 샘플링회로, D/A변환기, 연속근사 레지스터, 비교기로 구성되어 있다. 계수비교형과 달리 카운터의 최상위비트값만 1로 세트해 놓고 입력신호와 비교하여 최상위비트값을 결정하기 시작하여 다음 상위비트값들의 값을 결정해나가는 방식의 A/D변환기로 n 비트형의 경우 n 번의 비교로 변환이 완료되어 변환시간은 백 μs 이하로 계수비교형에 비해 작고 일반적인 응용범위에 널리 사용된다. 저가로 많이 사용되는 내셔날 세미컨덕터사의 ADC08시리즈는 8비트 축차비교형 A/D 변환기의 대표적 예이다.

⑥병렬비교형: 샘플링회로, 사다리형 저항기, 다수개의 비교기, 디코더로 구성되어 있다. n 비트 변환기의 경우 2^n-1 개의 비교기를 사용하여 사다리형 저항기에 따라 적절히 분압된 입력전압을 동시에 병렬처리하여 디코더를 통해 디지털 출력값을 만들어 내기 때문에 현재 변환기 중 변환속도가 수십 ns이하로 가장 빠르나 가장 고가이다. 아날로그디바이스사의 AD5010이나 AD6020은 6비트 병렬비교형 A/D 변환기의 대표적 예이다.

5.7.3. ATmega128의 ADC

5.7.3.1. 특징

- 10비트 8채널(PF0~7, 핀번호 61~54)의 축차비교형 단극성 변환기 제공
- 7채널의 차동입력 변환기로 사용 가능
- 10배 또는 200배의 증폭률을 가진 2채널의 차동입력 변환기능 제공
- 0~Vref범위의 전압을 입력으로 갖을 수 있다. 차동의 경우 -Vref ~ Vref까지 가능하다. 기준 전압 Vref는 VCC를 초과할 수 없다.
- 변환완료 인터럽트가 제공된다. 완료시 ADCSRA 레지스터의 ADIF 플랙이 세트된다.

- 13~260 usec의 변환시간을 사용자가 설정할 수 있다.

5.7.3.2. ADCH, ADCL 레지스터

ADC 결과를 단극성의 경우 $1024 \times V_{IN} / V_{ref}$로 0~1023 범위값으로,차동입력의 경우 2의 보수를 사용하여 $512 \times \mathrm{Gain} \times (V_+ - V_-) / V_{ref}$로 -512~ 511범위의 값으로 저장한다. 초기값은 0x0000이다. ADMMUX 레지스터의 ADLAR값에 따라 그림 111에 보여진 방식으로 정렬하여 저장한다.

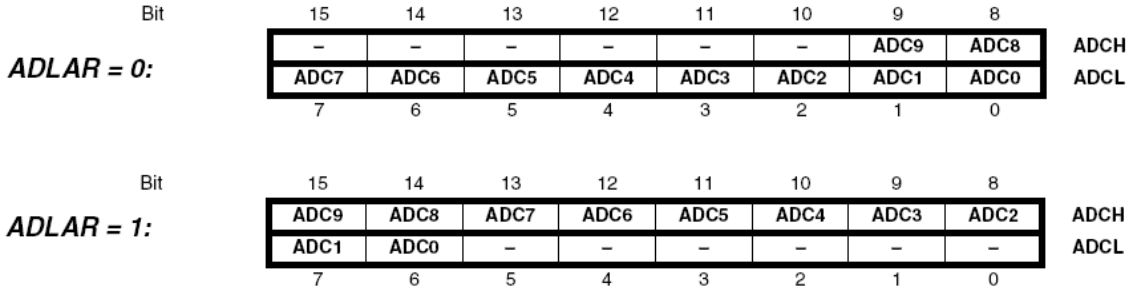


그림 111. ADCH, ADCL 레지스터(출처:ATMEL)

5.7.3.3. ADMUX 레지스터

ADSC, ADEN이 1로 세트되어 AD변환이 시작될 때마다 ADMUX의 설정값에 따라 AD변환이 진행된다. 설정값의 변환은 차동입력을 사용하는 경우에는 AD변환을 시작한 후 적어도 125usec가 경과한 후에 한다.

- 비트7~6(REFS1~0 : REFerence Selection 1~0) : 입력전압의 범위의 기준전압 Vref를 선택한다.

  ① 00 : 외부의 AREF 단자(62번핀)로 입력된 전압을 기준전압으로 사용.

  ② 01 : 외부의 AVCC 단자(64번핀)로 입력된 전압을 기준전압으로 사용.

  ③ 10 : 보류

  ④ 11 : 내부의 2.56V를 기준전압으로 사용.

- 비트5(ADLAR : ADC Left Adjust Result) : 변환결과가 ADC 데이터 레지스터에 저장될 때 모양을 그림 111의 경우 처럼 결정한다.

- 비트4~0(MUX4~0) : 그림 112처럼 ADC의 입력단자를 선택하고 차동입력의 경우 이득을 선택 하는데 사용한다.

5.7.3.4. ADCSRA 레지스터

- 비트7(ADEN : ADc ENable) : 1로 설정하여 동작을 허용하고 0으로 하여 중지시킨다.

- 비트6(ADSC : ADc Start Conversion) : 1로 설정하면 변환이 시작된다. 변환이 완료되면 자동적으로 클리어된다.

- 비트5(ADFR : ADc Free Running select) : 1로하면 Free running 모드로 설정한다.

- 비트4(ADIF : ADc Interrupt Flag) : ADCH, ADCL 레지스터가 갱신되면 ADIF가 1로 세트되며 인터럽트를 요청한다.

- 비트3(ADIE : ADc Interrupt Enable) : 1로 세트된 경우 변환완료 인터럽트를 개별적으로 허용한다.

- 비트2~0(ADPS2~0 : ADc Prescaleer Select 2~0) : 시스템 클럭을 분주하여 clk_{ADC}를 만들 때 분주비를 결정한다. 000, 001이면 분주비를 2로, 010이면 분주비를 4로, 011이면 분주비를 8로, 100이면 분주비를 16으로, 101이면 32로, 110이면 64로, 111이면 분주비를 128로 한다.

5.7.3.5. 동작

- 단일변환 모드 : ADCSRA 레지스터에서 ADEN, ADSC 플랙을 1로 설정하여 변환을 시작시킬 수 있다. 최초

변환에 25개의 clk_{ADC}클록이 필요하다. 변환이 완료되면 변환 결과가 ADCH, ADCL 레지스터에 결과가 저장되면 ADIF가 1로 세트되며 AD 변환완료 인터럽트가 요청되고 ADSC비트는 자동적으로 클리어된다. 다시 변환을 시작시키려면 ADSC플랙을 1로 하면 된다. 그리고 이때부터는 13개의 clk_{ADC}클록이 필요하다.

- Free Running 모드 : ADCSRA 레지스터에서 ADEN, ADSC, ADFR 플랙을 1로 설정하여 변환을 시작시킬 수 있다. 변환완료에 13개의 clk_{ADC}클록이 필요하다. 변환이 완료되면 변환 결과가 ADCH, ADCL 레지스터에 결과가 저장되면서 즉시 그 다음 차례의 AD변환이 자동적으로 시작된다.

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000		ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010	N/A	ADC0	ADC0	200x
01011		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110		ADC2	ADC2	200x
01111		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x
11101		ADC5	ADC2	1x
11110	1.23V (V_{BG})	N/A		
11111	0V (GND)			

그림 112. ADC 채널과 Gain의 설정(출처:ATMEL)

5.7.3.6. 잡음 제거 방법

- 아날로그 입력 신호선을 최소한으로 짧게 한다.
- AVCC단자에는 디지털 전원 VCC를 LC 필터로 안정화시켜 인가한다(그림 113참조).

- AD변환을 하는 중에는 나머지 PF7~0중에서 디지탈 입출력 포트로 사용하는 포트의 값을 스위칭하지 않는다.
- ADC noise reduction 모드를 이용하면 매우 안정적인 변환을 이룰 수 있다.

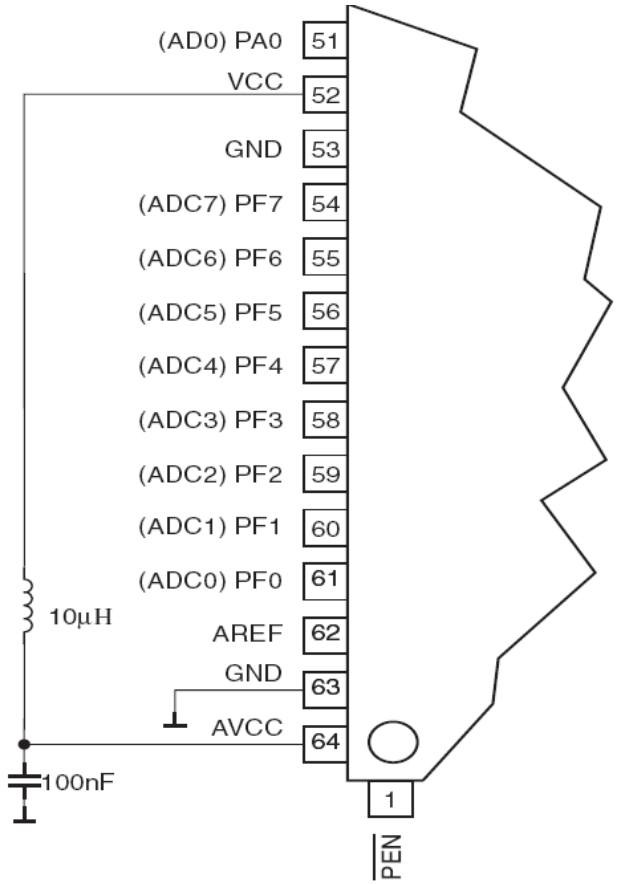


그림 113. 아날로그 전원 단자의 처리(출처:ATMEL)

5.7.4. 실험59 : 조도 변화 측정

- 예제 : 조도에 따라 저항값이 변화하는 CdS의 저항변화를 측정하여 표시한다.
- 회로도 : 그림 114의 회로를 구현하고 CdS_CON을 그림 79의 F_CON에 연결한다. 그림 95의 회로를 구현하고 FND_CON을 그림 79의 C_CON에 연결한다.그림 95에서 FND0~3만 이용할 것이므로 그 4개만 구현하면 된다.
- 프로그램 예 :

```
#include <mega128.h>
#include <delay.h>
unsigned char dbuf[4], dbuf_index;
unsigned int cnt;
void hex2bcd(unsigned int hex, unsigned char *ptr) {            /* 10240이하의 헥사값을 3자리수 bcd로 변환한다. */
      *ptr = hex / 1000;
      *(ptr+1) = (hex % 1000) / 100;
      *(ptr+2) = ((hex % 1000) % 100 ) / 10;
      *(ptr+3) = ((hex % 1000) % 100 ) %10;
}
interrupt [17] void timerint0(void){  // interrupt [TIM0_OVF] void timerint0(void)로 해도 됨
      unsigned char tmpbuf;
      if( ++cnt == 2 ){
      cnt = 0; if( dbuf_index++ == 3 ) dbuf_index= 0; }
      tmpbuf = dbuf[dbuf_index];
      PORTC = (( tmpbuf << 4 )&0xf0) | dbuf_index;
```

```
                        /* 표시할 데이터와 FND의 위치는 tmpbuf의 상위비트와 하위비트로 */
    TCNT0 = 6;          // 재정의 1msec = 64*(256-6)/16       시뮬레이션할 때는 조절할 필요가 있음
}
void main(void){
unsigned        int         sum;
unsigned        char        i;
    DDRC = 0xff; // 포트C를 출력으로
    SREG |= 0x80;       //인터럽트 전체 허용
    TIMSK |= 1;         // 타이머0 오버플로 인터럽트 개별허용
    TCCR0 |= 4;         // 분주비를 64로 설정                    시뮬레이션할 때는 조절할 필요가 있음
    TCNT0 = 6;          // 1msec = 64*(256-6)/16                 시뮬레이션할 때는 조절할 필요가 있음
    ADMUX = 0x00;       // ADC0를 사용하고 Vref = Vcc사용
    ADCSRA = 0x83;      // ADEN = 1 , 분주비는 8
    delay_ms(10);
    for(;;){    sum=0;
                for(i=0;i<16;i++){
                ADCSRA |= 0x40;                     // ADSC = 1 , 변환시작
                while((ADCSRA & 0x10) != 0x10);     // 변환완료때까지 기다린다.
                sum += ADCH*256 + ADCL;}
                sum = sum >> 4;                     // 16으로 나누어 평균값을 구한다.
    hex2bcd(sum, dbuf);             /* 표시할 데이터 초기화 */
    }
}
```

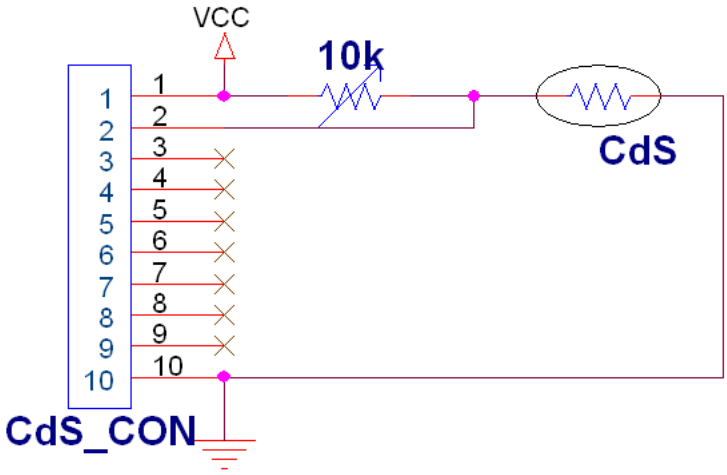


그림 114. 조도 측정 회로

5.8. 아날로그 비교기

5.8.1. 구성

그림 115는 ATmega128에 내장된 아날로그 비교기의 구성을 보여준다.

- AIN0~1(PE2~3:핀4~5) : 비교기 양극성 입력, 비교기 음극성 입력
- ACO(Analog Comparator Output) : ACSR 레지스터의 비트5 플랙으로 양극성 입력이 음극성 입력보다 크면 1로 세트된다.
- ACIS1~0 : 아날로그 비교기의 인터럽트 모드를 설정한다.
- 밴드갭 레퍼런스 : 양극성입력으로 사용될 수 있는 1.23V 내부 기준 전압
- ADC MUX OUTPUT : 음극성 입력으로 사용될 수 있는 ADC의 입력신호들 ADC0~7

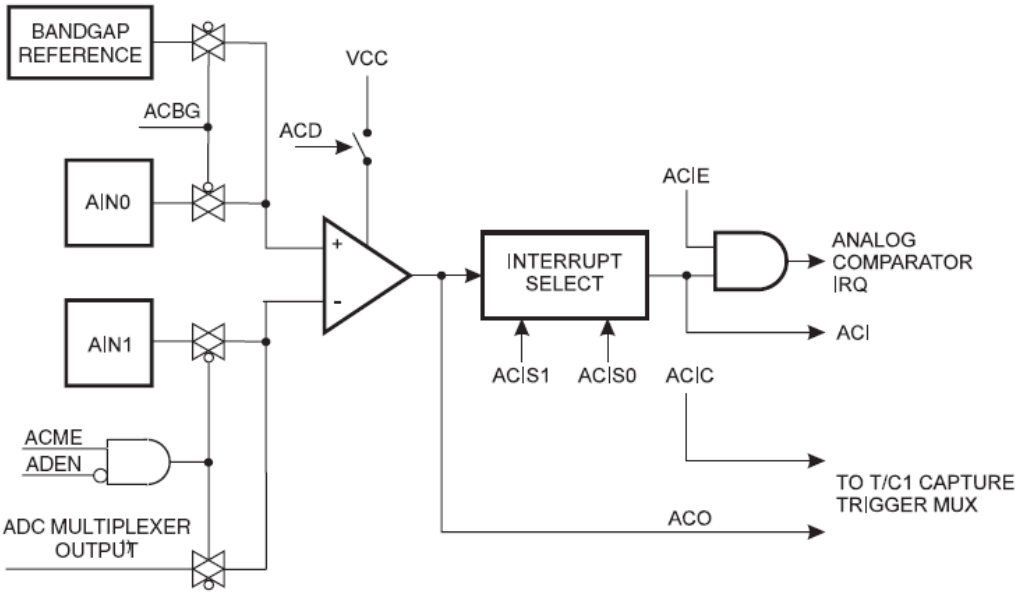


그림 115. 아날로그 비교기의 블록 선도(출처:ATMEL)

5.8.2. 제어

ACSR(Analog Comparator control and Status Register)와 SFIOR(Special Function IO Register) 레지스터를 사용해서 아날로그 비교기의 기능과 입력신호를 선택할 수 있고 동작 상태를 알아 낼 수 있다.

5.8.2.1. ACSR 레지스터

- 비트7(ACD:Analog Comparator Disable) : 1이면 아날로그 비교기의 동작을 금지시켜 소비전력을 감소시킨다. 디폴트는 0이므로 비교기를 사용하지 않는 경우 1로 하면 전력소비를 줄일 수 있다.
- 비트6(ACBG:Analog Comparator Bandgap Select) : 아날로그 비교기의 양극성 입력을 선택하여 1이면 내부 기준 전압 1.23V가 선택되고 0이면 AIN0단자(PE2, 핀4)로 선택된다.
- 비트5(ACO:Analog Comparator Output) : 비교기의 출력값으로 클럭과 동기되어 발생된다. 양극성입력이 음극성 입력보다 크면 1로 세트된다.
- 비트4(ACI:Analog Comparator Interrupt Flag) : 아날로그 비교기의 인터럽트 발생 플랙으로 ACIS1~0의 설정에 따라 결정된다. 비교기 인터럽트 개별허용 비트가 ACIE가 1로 세트되어 있고 SREG.7일 1일때 인터럽트는 서비스되고 그와 함께 자동적으로 ACI비트는 클리어된다.
- 비트3(ACIE:Analog Comparator Interrupt Enable) : 비교기 인터럽트 개별허용 비트
- 비트2(ACIC:Analog Comparator Input Capture Enable) : 아날로그 비교기의 출력 ACO값을 타이머1의 입력 캡춰 트리거 신호로 사용될 수 있도록 설정한다.
- 비트1~0(ACIS1~0:Analog Comparator Interrupt mode Select 1~0) : 아날로그 비교기 인터럽트 모드를 설정한다. 설정을 변경하는 경우 원하지 않는 인터럽트가 발생할 수 있으므로 ACIE는 클리어하고 변경한다.

  ① 00 : ACO의 하강 및 상승에지에서 아날로그 비교기 인터럽트 발생

  ② 01 : 보류

  ③ 10 : ACO의 하강에지에서 아날로그 비교기의 인터럽트 발생

  ④ 11 : ACO의 상승에지에서 아날로그 비교기의 인터럽트 발생

5.8.2.2. SFIOR 레지스터

- 비트3(ACME : Analog Comparator Multiplexer Enable) : 아날로그 비교기의 음극성 입력에 ADC 입력을 사

용할 수 있도록 허용한다. 0이며 음극성 입력으로 AIN1(핀5, PE3)가 사용되며 1로 세트시키고 ADCSRA레지스터의 비트7(ADEN비트)를 0으로 클리어하여 ADC의 동작을 정지시키면 ADC0~7중 하나를 음극성 입력으로 사용할 수 있다. 이때 ADC0~7중 하나를 선택하는 것은 ADMUX 레지스터의 비트2~0(MUX2~0비트)로 설정한다.

5.8.3. 실험60 : 아날로그 비교기 실험

- 예제 : 조도에 따라 저항값이 변화하는 CdS의 저항변화가 어느 이상이면 소리를 낸다.
- 회로도 : 그림 114의 회로를 구현하고 CdS_CON을 그림 79의 F_CON에 연결한다. 그림 98의 회로를 구현하고 BUZZ_CON을 그림 79의 C_CON에 연결한다.
- 프로그램 예 :

```
#include <mega128.h>
#include <delay.h>
unsigned char           time_data;
interrupt [17] void timerint0(void){  // interrupt [TIM0_OVF] void timerint0(void)로 해도 됨
    time_data--;
    TCNT0 = 77;       // 21.306msec = 1024*(256-77)/16          시뮬레이션할 때는 조절할 필요가 있음
}
void playtone(unsigned int tone, unsigned char time){ // tone는 반주기값 1usec, time는 박자값
unsigned int buf;
    DDRC.0 = 1;       // PORTC.0을 출력으로 설정
    TCCR0 |= 7;       // 분주비를 1024로 설정                  시뮬레이션할 때는 조절할 필요가 있음
    TCNT0 = 77;       // 21.306msec = 1024*(256-77)/16          시뮬레이션할 때는 조절할 필요가 있음
    time_data = time;
    buf = tone;
    do{
            PORTC.0 = 1;      do{ delay_us(1);} while(buf--);
            PORTC.0 = 0;      do{ delay_us(1);} while(tone--);
    } while(time_data);
    time_data = time;
    TCNT0 = 77;       // 21.306msec = 1024*(256-77)/16          시뮬레이션할 때는 조절할 필요가 있음
    while(time_data);
    TCCR0 =0x00;
}
void main(void){
unsigned      int       sum;
unsigned      char      i;
    DDRC = 0xff; // 포트C를 출력으로
    SREG |= 0x80;     //인터럽트 전체 허용
    TIMSK |= 1;       // 타이머0 오버플로 인터럽트 개별허용
    TCCR0 |= 4;       // 분주비를 64로 설정                    시뮬레이션할 때는 조절할 필요가 있음
    TCNT0 = 6;        // 1msec = 64*(256-6)/16                 시뮬레이션할 때는 조절할 필요가 있음
    ADMUX = 0x00;     // ADC0를 사용하고 Vref = Vcc사용
    ADCSRA = 0x83;    // ADEN = 1 , 분주비는 8
    delay_ms(10);
    for(;;){    sum=0;
            for(i=0;i<16;i++){
            ADCSRA |= 0x40;                 // ADSC = 1 , 변환시작
            while((ADCSRA & 0x10) != 0x10);     // 변환완료때까지 기다린다.
            sum += ADCH*256 + ADCL;}
            sum = sum >> 4;                 // 16으로 나누어 평균값을 구한다.
    hex2bcd(sum, dbuf);                 }               /* 표시할 데이터 초기화 */
}
void main(void){
    unsigned int tone_data=0xffff;
    DDRC = 0xff;        // 포트C를 출력으로
    SREG.7 = 1;         //인터럽트 전체 허용
```

```
    TIMSK |= 1;          // 타이머0 오버플로 인터럽트 개별허용
    ACSR |= 0x40;        // 양극성 입력을 1.23으로
    SFIOR |= 0x08;      // ADC입력을 -입력으로
    ADCSR &=0x7f;        // ADEN = 0
    ADMUX &=0xf8;        // ADC0을 -입력으로
    delay_ms(100);
    for(;;){
    if((ACSR & 0x20) != 0x20){              // if 1.23 > ADC0 and (ACSR & 0x20) == 0x20
            if(tone_data > 100){tone_data -=200;                    playtone(tone_data,250); }
            else tone_data=0xffff;}
    delay_ms(100);       }
}
```

5.9. 직렬 통신

5.9.1. SPI 포트 제어

ATmega128은 주변 장치 또는 동일한 AVR 계열간의 고속 직렬 동기 통신용 모듈인 SPI(Serial Peripheral Interface)를 제공하는데 $\overline{SS}$(PB0, pin 10), SCK(PB1, pin 11), MOSI(PB2, pin 12), MISO(PB3, pin 13) 4선을 이용하여 전이중으로 통신이 가능하며, 마스터 및 슬레이브 동작을 가능하게 하며, 7개의 전송속도를 제공하여 최대 수십 MHz까지의 통신이 가능하며 전송완료 인터럽트를 제공한다. 또한 아이들 모드를 해제하는 기능도 제공한다. 그림 116은 ATmega 1298의 SPI포트의 블록선도이다. SPI는 ATmega128의 플래쉬롬에 프로그램을 다운로드하는 ISP 기능도 제공한다.

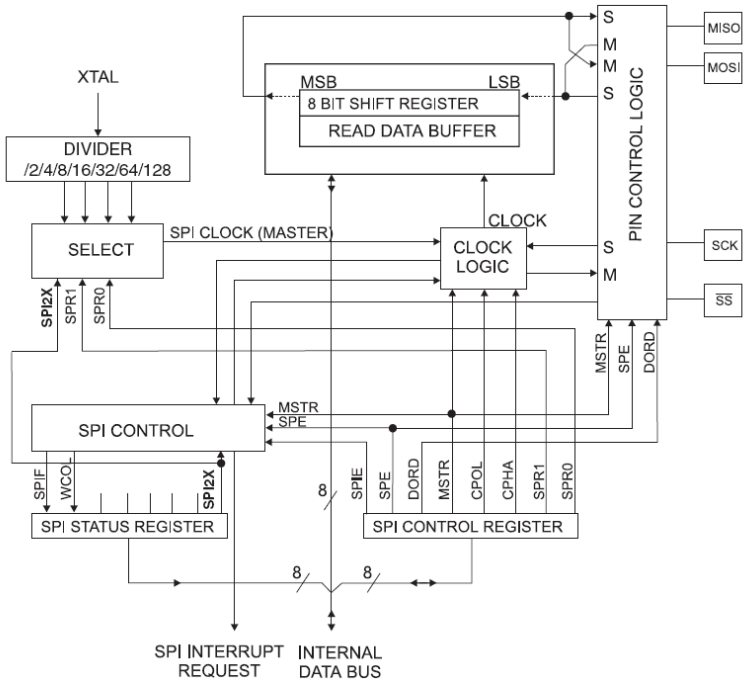


그림 116. SPI 포트의 블록선도(출처:ATMEL)

5.9.1.1. SPI의 동작

- 반드시 1개의 마스터와 1개의 슬레이브 사이에서만 이루어져 마스터가 슬레이브에게 데이터를 보내려면 여러 슬레이브에서 원하는 슬레이브에게 $\overline{SS}$ (Slave Select) 신호를 0레벨로 출력하여 선택하고 마스터는 클록 신호를 발생하여 SCK(Serial Clock)을 통해 출력하고 송신할 데이터를 시프트 레지스터에 데이터를 준비하여 MOSI (Master Output Slave Input) 단자로 출력한다. 이때 동시에 MISO (Master Input Slave Output) 단자를 통해서는 더미 데이터가 입력된다(그림 117참조).
- 슬레이브 모드 : $\overline{SS}$ 가 입력핀으로 동작하며 0레벨상태로 입력되는 경우에만 SPI가 활성화되고 MISO가 출

력단자로 된다. 이때 외부에서 입력된 SCK 신호에 의하여 데이터 레지스터의 1바이트가 전송되고 나면 SPSR (SPi Status Register)의 비트7(SPIF)이 1로 되면서 SPI 전송완료 인터럽트가 요청된다. 만약 슬레이브 모드에서 $\overline{SS}$ 단자에 1레벨상태가 입력되면 송신 및 수신 로직을 즉시 리셋시키고 시프트 레지스터에 일부 수신된 데이터도 버리고 SPI는 입력되는 데이터를 받을 수 없는 비활성상태로 된다.

- 마스터 모드 : SPCR(SPi Control Register)의 비트4(MSTR)가 1로 설정되면 마스터가 되며 사용자는 $\overline{SS}$ 핀의 방향을 DDRB0 비트를 사용하여 결정할 수 있다. 마스터 모드에서 $\overline{SS}$ 핀이 입력으로 설정되어 있고 0레벨이면 외부에서 다른 마스터가 슬레이브로 지정하고 데이터를 전송하기 시작하는 것으로 해석될 수 있으므로 $\overline{SS}$ 핀을 입력으로 설정하면 반드시 풀업저항과 같은 외부회로에 의해 1레벨로 유지되어야 한다. 마스터가 되어 $\overline{SS}$ 핀을 출력으로 설정하고 0레벨을 출력시켜 슬레이브를 지정한 경우 SPDR(SPi Data Register)에 데이터를 기록하여 송신할 수 있다. 이 때 클럭이 발생되고 하드웨어적으로 SPDR이 쉬프트되며 슬레이브로 전송된다. 전송이 끝나면 클럭이 정지되고 SPSR의 비트7(SPIF)이 세트되면서 SPI 전송완료 인터럽트가 요청된다. 이후 SPDR에 데이터를 다시 기록하면 다시 전송이 반복된다. 데이터 패킷의 전송을 마치려면 $\overline{SS}$로 1레벨을 출력시키면 된다.

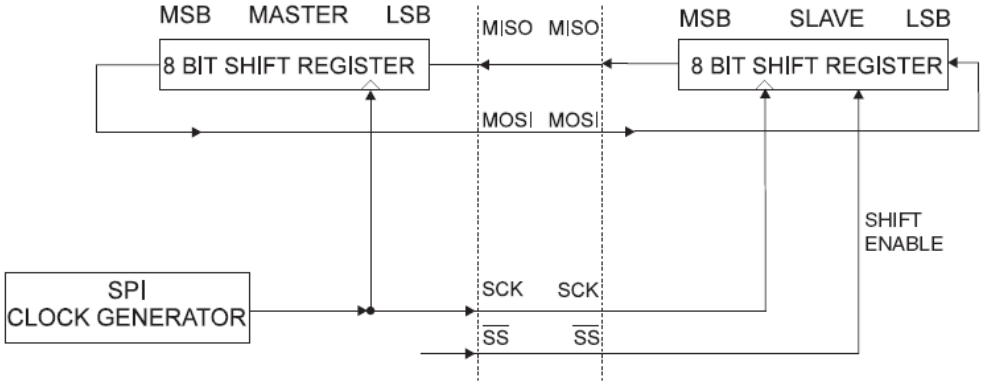


그림 117. SPI의 동작(출처:ATMEL)

5.9.1.2. SPCR 레지스터

- 비트7(SPIE:SPi Interrupt Enable) : 1로 세트하면 SPI 전송완료 인터럽트 개별 허용
- 비트6(SPE: SPi Enable) : 1로 세트하면 SPI 직렬통신을 허용
- 비트5(DORD:Data ORDer) : 1로 설정하면 LSB부터 전송하고 0으로 하면 MSB부터 전송한다.
- 비트4(MSTR:Master/Slaver Select) : 1로 설정하면 마스터로 동작하고 0으로 하면 슬레이브로 동작한다. 마스터로 설정하고 $\overline{SS}$ 핀이 입력으로 설정되어 0레벨이 입력되면 MSTR비트는 자동으로 클리어되며 슬레이브 모드로 되고 SPSR의 비트7(SPIF)이 세트되면서 인터럽트가 요청된다.
- 비트3(CPOL:Clock POLarity) : 데이터 샘플링 동작이 수행되는 SCK 클록의 극성을 설정한다. 디폴트값인 0이면 Leading Edge의 경우는 상승 에지로, Trailing Edge의 경우에는 하강에지로 선정된다. CPOL=1이면 Leading Edge의 경우는 하강 에지로, Trailing Edge의 경우에는 상승에지로 선정된다.
- 비트2(CPHA:Clock PHAse) : 데이터 샘플링 동작이 수행되는 SCK 클록의 위상을 설정한다. 디폴트값인 0이면 Leading Edge의 경우는 샘플링이 되고, Trailing Edge의 경우에는 셋업된다. CPHA=1이면 반대로 된다.
- 비트1~0(SPR1~0:SPi clock Rate select 1~0): SPSR의 비트0(SPI2X비트)과 함께 SCK 클럭신호의 주파수 분주비를 설정한다. SPSR의 비트0이 0인 상태에서 00이면 시스템 클록의 4분주, 01이면 16분주, 10이면 64분주, 11이면 128분주로 설정된다. SPSR의 비트0이 1인 상태에서는 주파수를 두배로 하여 분주비를 반감시키는데 00이면 시스템 클록의 2분주, 01이면 8분주, 10이면 32분주, 11이면 64분주로 설정된다.

5.9.1.3. SPSR 레지스터

- 비트7(SPIF:SPi Interrupt Flag) : 전송이 완료되면 1로 세트되면서 인터럽트가 요청된다. 마스터로 설정하고 $\overline{SS}$ 핀이 입력으로 설정되어 0레벨이 입력되면 SPCR의 비트4(MSTR)는 자동으로 클리어되며 슬레이브 모드로

되고 SPIF가 세트되면서 인터럽트가 요청된다. 인터럽트가 서비스되면 자동으로 클리어된다. SPSR을 읽고 SPDR에 접근하는 경우에 WCOL과 함께 클리어된다.

- 비트6(WCOL:Write COLision flag) : SPI를 통해 데이터를 전송하고 있는 동안에 SPDR레지스터를 기록하려고 하면 세트된다. SPSR을 읽고 SPDR에 접근하는 경우에 SPIF와 함께 클리어된다.
- 비트0(SPI2X:SPI Double speed) : 마스터로 동작할 때 SCK클록 신호의 주파수를 2배로 설정한다.

5.9.2. USART 포트 제어

ATmega128은 8비트 병렬 데이터를 직렬형태로 바꾸어 송신하고 수신된 직렬 데이터를 병렬 데이터로 바꿔 CPU에 입력하는 기능의 직렬통신포트를 2개 내장하고 있다. 직렬통신포트는 송신과 수신을 동시에 할 수 있는 완전 이중방식이며 동기 및 비동기 전송이 가능하다. 또한 멀티 프로세서 통신 모드로 동작하는 것도 가능하며 높은 정밀도의 보레이트 발생기를 내장하고 있다. 송신완료, 송신 데이터 레지스터 준비완료, 수신완료 등 3가지의 인터럽트를 제공한다.

5.9.2.1. 데이터 프레임 포맷

그림 118에 보여진 것처럼 (1 비트의 스타트 비트) +(5,6,7,8,9 비트의 데이터 비트) + (0, 1 비트의 패리티 비트) + (1,2 비트의 스탑비트) 프레임으로 이루어져 최소 7비트 최대 13비트로 구성가능하다.

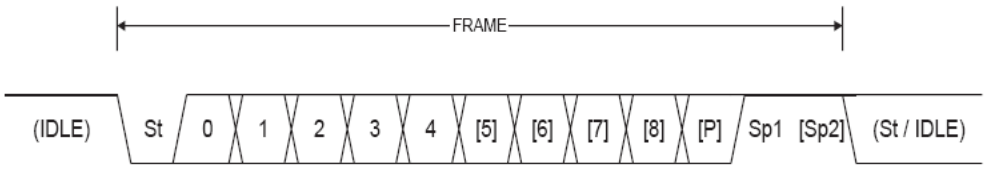


그림 118. USART 통신의 데이터 프레임(출처:ATMEL)

- 스타트 비트 : 1비트로 이루어 졌으며 항상 0레벨이다. 송신시에 자동적으로 생성된다.
- 데이터 비트 : 5,6,7,8,9비트가 가능하다.
- 패리티 비트 : 패리티를 사용하지 않을 수도 있고 사용하는 경우 홀수 혹은 짝수 패리티 1비트를 사용한다.
- 스탑 비트 : 1,2개의 비트가 가능하며 항상 1레벨이다. 송신시에 자동적으로 생성된다.

5.9.2.2. UDRn 레지스터

- UDRn(Usart i/o Data Register n)레지스터는 USARTn 포트의 송수신 데이터의 버퍼의 기능을 수행하는 8비트 레지스터이다(여기에서 n은 0 또는 1). 송신 및 수신 버퍼는 동일한 번지에 위치하지만 내부적으로는 서로 다른 별개의 레지스터로 송신할 데이터를 UDRn에 쓰면 송신 데이터 버퍼 TXBn에 저장되고 수신된 데이터를 UDRn으로 읽으면 수신 데이터 버퍼 RXBn에 수신되어 있는 값이 읽힌다.
- 송신 버퍼는 UCSRnA 레지스터의 UDREn 비트가 세트된 경우에만 쓰기가 가능하다.

5.9.2.3. UCSRnA 레지스터

UCSRnA(Usart Control and Status Register n A) 레지스터는 USART 포트의 송수신 동작을 제어하거나 상태를 저장하는 8비트 레지스터이다.

- 비트7(RXCn: Receive Complete n) : UDRn의 수신 버퍼에 읽히지 않은 수신 문자가 있으면 세트되고 읽은 경우 0으로 클리어 된다. 세트되는 경우 수신완료 인터럽트가 요청된다. 1을 쓰면 소프트웨어적으로 클리어된다.
- 비트6(TXCn: Transmit Complete n) : UDRn의 송신 버퍼에 아직 새로운 송신 문자가 기록되지 않은 경우 세트된다. 이 경우 송신완료 인터럽트를 요청하고 인터럽트가 서비스 시작되면 자동으로 0으로 클리어된다.
- 비트5(UDREn: Usart Data Register Empty n) : UDRn의 송신 버퍼가 비어 있어서 송신 문자를 받을 준비가 되어 있으면 세트된다. 세트되는 경우 송신 데이터 레지스터 준비완료 인터럽트가 요청된다. 1을 쓰면 소프트웨

어적으로 강제로 클리어된다.

- 비트4(FEn: Frame Error n) : UDRn의 수신 버퍼에 저장되어 있는 문자를 수신하는 동안에 수신 문자의 첫 번째 스톱 비트가 0으로 검출되는 프레임 에러가 발생하였음을 나타내며 세트되는 플랙비트로 이 때 세트된 것은 UCSRnA 레지스터를 기록하면 클리어된다.
- 비트3(DORn: Data Overrun Error n) : 수신동작에서 UDRn의 수신 버퍼에 읽지 않은 수신 문자가 들어 있는 상태에서 수신 시프트 레지스터에 새로운 문자가 수신 완료되고 다시 그 다음 수신 문자의 스타트 비트가 검출되는 오버런 에러가 발생하였음을 나타내며 세트되는 플랙비트로 UCSRnA 레지스터를 기록하면 클리어된다.
- 비트2(UPEn:Usart Parity Error n) : UDRn의 수신 버퍼에 저장된 문자에 패리티 에러가 발생하였음을 나타내는 플랙비트로 UCSRnC레지스터의 UPMn1이 세트되어 패리티 비트를 사용하도록 설정된 경우에만 세트될 수 있다. UCSRnA 레지스터를 기록하면 클리어된다.
- 비트1(U2Xn:Usart 2 transmission speed) : 비동기 모드에서만 유효한 것으로 1로 세트하면 클럭의 분주비를 16에서 8로 낮추어 전송속도를 2배 증가시킬 수 있다.
- 비트0(MPCMn:Multi-Processor Communication Mode n) : 1로 세트하면 멀티 프로레서 통신모드로 설정한다.

5.9.2.4. UCSRnB 레지스터

- 비트7(RXCIEn: RX Complete Interrupt Enable n) : 수신완료 인터럽트를 개별 허용 비트
- 비트6(TXCIEn: TX Complete Interrupt Enable n) : 송신완료 인터럽트를 개별 허용 비트
- 비트5(UDRIE : UDR empty Interrupt Enable n) : 송신 데이터 레지스터 준비완료 인터럽트 개별 허용 비트
- 비트4(RXENn: RX ENable n) : 1로 세트하면 RxD0,1(PE0, PD2)가 USART용 수신단자로 동작하는 것을 허용하고 에러 플랙 비트 DORn, UPEn, U2Xn의 동작을 허용한다.
- 비트3(TXENn: TX ENable n) : 1로 세트하면 TxD0,1(PE1, PD3)가 USART용 송신단자로 동작하는 것을 허용한다. 클리어하더라도 송신이 완료될 때까지는 송신단자로 동작한다.
- 비트2(UCSZn2:Usart Character SiZe n2) : UCSRnC레지스터의 비트2~1(UCSZn1~0)와 함께 전송 문자의 데이터 비트수를 설정하는 데 사용한다. ①000 : 5비트, ②001 : 6비트, ③010 : 7비트, ④011 : 8비트, ⑤111 : 9비트, ⑥100~110: 보류.
- 비트1(RXB8n:RX data Bit 8 n) : 전송문자가 9비트로 설정된 경우 수신된 문자의 9번째 비트를 저장한다. 반드시 UDRn 레지스터보다 먼저 읽혀져야 한다.
- 비트0(TXB8n:TX data Bit 8 n) : 전송문자가 9비트로 설정된 경우 송신할 문자의 9번째 비트를 저장한다. 반드시 UDRn 레지스터보다 먼저 기록되야 한다.

5.9.2.5. UCSRnC 레지스터

- 비트6(UMSELn:Usart Mode SELect n) : 1이면 USARTn 포트를 동기 모드로 0이면 비동기 모드로 설정
- 비트5~4(UPMn1~0:Usart Parity Mode 1~0) : 패리티 모드를 설정한다. ① 00 : 패리티 체크 기능을 사용하지 않는다. ② 01 : 보류 ③ 10 : 짝수 패리티 사용 ④ 11 : 홀수 패리티 사용
- 비트3(USBSn:Usart Stop Bit Select n) : 1로 세트하면 스톱비트를 2개로 설정하고 0이면 스톱비트를 1개로 설정한다.
- 비트2~1(UCSZn1~0:Usart Character SiZe n1~0) : UCSRnB레지스터의 비트2(UCSZn2)와 함께 전송문자의 비트수를 설정한다. ①000 : 5비트, ②001 : 6비트, ③010 : 7비트, ④011 : 8비트, ⑤111 : 9비트, ⑥100~110: 보류.
- 비트0(UCPOLn:Usart Clock POLarity n) : 동기 전송 모드의 슬레이브 동작에서만 유효한 것으로 1로 설정

하면 송신 데이터는 XCKn 클럭의 하강에지에서 새로운 값이 출력되고 수신 문자는 XCKn의 상승에지에서 얻어진다. 클리어된 경우는 반대로 된다.

5.9.2.6. UBRRnH, UBRRnL 레지스터

UBRRn(Usart Baud Rate Register n)은 16비트 레지스터로 그 중 하위 12비트가 유효한 것으로 송수신 속도를 다음의 공식에 따라 설정한다. 그림 119는 클럭이 16MHz인 경우의 보레이트와 UBRRn레지스터의 설정값, UCSRnA 레지스터의 비트1(U2Xn)의 설정값, 그리고 에러를 보여준다. 230,400bps를 제외하고 모든 전송속도에서 해당 보레이트와 UBRRn, U2Xn을 설정하여 에러를 무시하고 사용될 수 있다.

$$\text{Baud 레이트} = 2^{U2Xn} \times f_{OSC}/16(UBRRn+1), \quad U2Xn = 0,1$$

Baud Rate (bps)	f_{osc} = 16.0000 MHz			
	U2X = 0		U2X = 1	
	UBRR	Error	UBRR	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%

그림 119. 보레이트와 UBRR의 관계(출처:ATMEL)

5.9.3. RS232C 개요

- 용도 : 저가로 동기식 혹은 비동기식으로 직렬통신하기 위한 규격으로 프린터나 모뎀 등 각종 주변장치를 컴퓨터에 연결할 때 사용된다.
- 특징 : 통신속도는 110bps~수십kbps 이상까지 다양하게 사용하며, 신호를 노이즈에 강하고 멀리 보낼수 있도록 하기 위해 TTL레벨과 달리 10V이면 0레벨, 0V이면 1레벨로 한다. PC에 장착된 직렬포트의 경우 9개의 핀으로 구성되어 있지만 컴퓨터간의 통신에서는 송신선과 수신선, 그라운드선 등 3개를 연결하여 통신이 가능하다.
- 연결길이 : 규격에서는 선의 길이는 약 15m로 되어 있으나 실제의 경우에 고품질의 케이블을 사용하면 최장 3km까지 사용가능하고 일반적으로 보레이트가 1kbps 이하로 작은 경우 고품질 케이블을 사용하면 1km까지 사용가능하고 보통의 케이블을 사용하면 300m까지 사용가능하고 보레이트가 증가하면 사용가능 거리도 줄어 9600에서 고품질의 케이블을 사용하면 75m, 보통 케이블을 사용하면 30m 까지 사용가능하다.
- 인터페이스 칩 : PC의 직렬포트에서 나오는 신호는 RS232C 레벨로 10V와 0V를 사용하고 반전된 상태이다. TTL레벨을 사용하는 마이컴으로 PC와 직렬통신하기 위해서는 RS232C 레벨과 TTL레벨로 바꿔주고 반전시켜주기위한 인터페이스가 필요하다. 이러한 기능을 하는 가장 흔한 칩에 MAX232가 있다.
- 터미널프로그램 설치: 마이컴과 PC의 직렬통신시 PC측에서 표시를 담당할 터미널프로그램으로 윈도우 기본 프로그램중 하나인 하이퍼터미널을 사용하면 된다. [시작메뉴->보조프로그램->통신->하이퍼터미널]을 선택하여 실행한다. 설치가 되어 있지 않은 경우에는 [제어판->프로그램 추가/제거->윈도우즈 구성요소 추가/제거]를 선

택하여 설치하면 된다. 하이퍼터미널을 실행하면 “연결 설명” 창이 나타난다. 여기에서 [이름(N)]에 적절한 이름을 입력하고 [아이콘(I)]에 적절한 아이콘을 선택한다. “연결 대상” 창이 나타나면 [연결에 사용할 모뎀(N)]에 COM1이나 COM2 등에서 사용자에 맞는 포트를 선택한다. 포트설정을 위한 창이 나타나면 [흐름제어(F)]메뉴에서 없음을 선택하고 [비트/초(B)]에서 9600으로 보레이트를 설정하고, [데이터 비트(D)]에서 8, 패리티(P)에서 없음, [정지비트(S)]에서 1을 선택한다(여기에서는 보레이트로 9600bps를 사용한다고 가정한 것인데 포트설정 값들은 마이컴에 탑재할 응용프로그램의 직렬통신 모드 설정에 따라 달라짐).

5.9.4. 실험61 : PC와 통신1

- 예제 : 직렬통신을 통해 문자열을 전송해 PC의 하이퍼터미널에 표시한다.
- 프로그램 작성시 유의점 : USART0 비동기 모드로 보레이트 9600을 사용한다고 가정한다.
- 회로도 : 그림 120의 회로를 구현하고 SC_CON을 그림 79의 E_CON에 연결한다. 그림 120의 9핀 코넥터(CONDB9)를 PC의 시리얼포트에 연결한다. ISP와 USART가 서로 간섭을 일으킬 수 있으므로 될도록 이면 프로그램을 다운로드할 때에는 9핀 코넥터를 연결하지 않도록 하고 USART통신을 할 때는 ISP 케이블을 연결하지 않도록 한다.

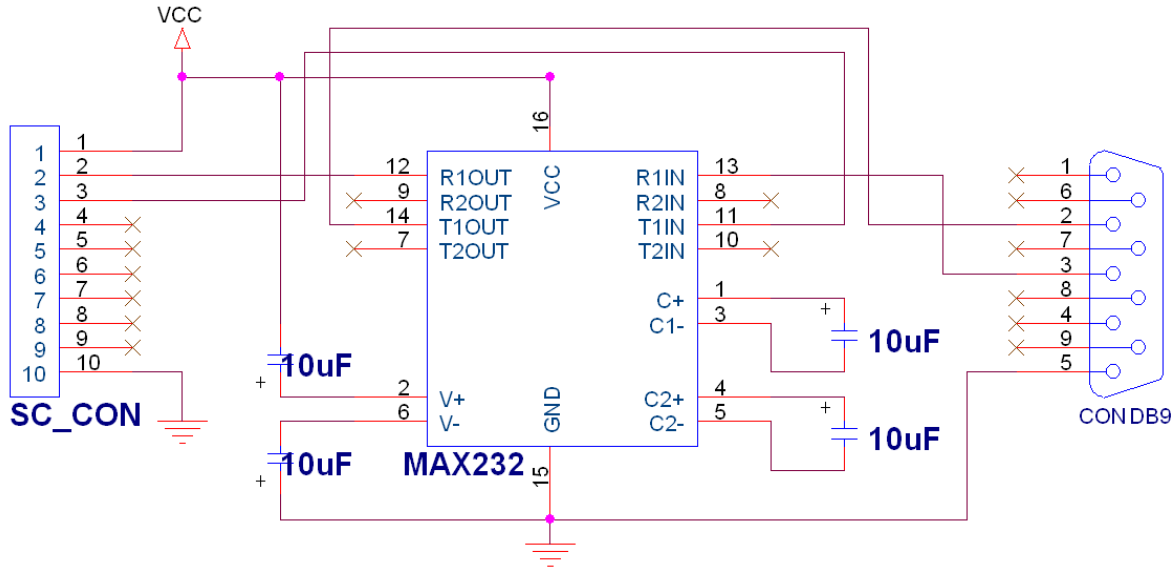


그림 120. RS232C 통신을 위한 인터페이스

- 프로그램 예 :

```
#include <mega128.h>
#include <delay.h>
void serial_init(void){      delay_ms(10);
      DDRE = 0xfe;
      UBRR0H = 0;       UBRR0L = 103;              // baud rate 9600
      UCSR0A = 0;                                  // 비동기 모드
      UCSR0B = 0x18;                               // usart 통신 enable,
      UCSR1C = 0x06;                               // no parity, 1 stop bit, 8 data bit
}
void putch(unsigned char ch){                 /* 한개의 문자를 pc로 송신한다. */
      while((UCSR0A & 0x20) == 0); UDR0 = ch;
      UCSR0A |= 0x20;   // 문자 송신준비완료
}
void putstrf(flash unsigned char *str){                  /* 문자열을 pc로 송신한다. */
      unsigned int          i=0;
      for(;str[i] != 0; i++) putch(str[i]);
```

```
}
void main(void){
    serial_init();
    putch(0x0d); putch(0x0a);                    /* 0x0d는 리턴의, 0x0a는 라인피드의 아스키코드 */
    putstrf("MY NAME IS HAN HO CHOI₩r₩n");       /* ₩r은 리턴, ₩n은 라인피드 */
}
```

● 과제 : ①다른 글자를 보내는 프로그램를 작성 하여라. ②그림 80의 SW_CON을 그림 79의 D_CON에 연결하고 그림 80의 SW1을 누를 때마다 글자를 보내게 하는 프로그램을 작성하여라.

5.9.5. 실험62 : PC와 통신2

● 예제 : PC의 키보드를 치면 아스키코드값이 직렬포트를 통해 전송되어 PORTC에 연결된 LED에 표시된다.

● 회로도 : 그림 120의 회로를 구현하고 SC_CON을 그림 79의 E_CON에 연결한다. 그림 120의 9핀 코넥터(CONDB9)를 PC의 시리얼포트에 연결한다. 실험을 할 때는 LED_SW를 닫고 한다.

● 프로그램 예 :

```
#include <mega128.h>
#include <delay.h>
void serial_init(void){     delay_ms(10);       DDRE = 0xfe;
    UBRR0H = 0;         UBRR0L = 103;               // baud rate 9600
    UCSR0A = 0;         UCSR0B = 0x18;              UCSR1C = 0x06;
}
unsigned        char        getch(void){                /* 한개의 문자를 pc에서 수신한다. */
    while((UCSR0A & 0x80) == 0); UCSR0A |= 0x80; return(UDR0);
}
void main(void){
    serial_init();
    DDRC = 0xff;        //포트C를 출력으로
    for(;;) PORTC = getch();
}
```

● 과제 : 타이머0 모드2를 사용하는 프로그램를 작성 하여라.

5.9.6. 실험63 : PC와 통신3

● 예제 : USART 인터럽트를 이용해 PC에서 보낸 아스키 코드값을 다시 송신한다.

● 회로도 : 그림 120의 회로를 구현하고 SC_CON을 그림 79의 E_CON에 연결한다. 그림 120의 9핀 코넥터(CONDB9)를 PC의 시리얼포트에 연결한다

● 프로그램 예 :

```
#include <mega128.h>
#include <delay.h>
void serial_init(void){     delay_ms(10);       DDRE = 0xfe;
    UBRR0H = 0;         UBRR0L = 103;               // baud rate 9600
    UCSR0A = 0;         UCSR0B = 0x18;              UCSR1C = 0x06;
}
void putch(unsigned char ch){               /* 한개의 문자를 pc로 송신한다. */
    while((UCSR0A & 0x20) == 0); UDR0 = ch;
    UCSR0A |= 0x20;   // 문자 송신준비완료
}
void putstrf(flash unsigned char *str){             /* 문자열을 pc로 송신한다. */
    unsigned int        i=0;
    for(;str[i] != 0; i++) putch(str[i]);
}
interrupt [19] void u0rx(void){
    putch(0x0d); putch(0x0a);
```

```
        putstrf("Pressed key is    ");
        putch(UDR0);
        putstrf("₩r₩n");      /* ₩r은 리턴, ₩n은 라인피드 */
        UCSR1A |= 0x80;
        putch(0x0d); putch(0x0a);                    /* 0x0d는 리턴의, 0x0a는 라인피드의 아스키코드 */
        putstrf("PRESS ANY KEY!!!!₩r₩n");                    /* ₩r은 리턴, ₩n은 라인피드 */
    }
    void main(void){
        serial_init();
        SREG.7 = 1;
        UCSR0B |=0x80;    // rx interrupt 개별 허용
        putch(0x0d); putch(0x0a);                    /* 0x0d는 리턴의, 0x0a는 라인피드의 아스키코드 */
        putstrf("PRESS ANY KEY!!!!₩r₩n");                    /* ₩r은 리턴, ₩n은 라인피드 */
        for(;;);
    }
```

- 과제 : 인터럽트를 사용하지 말고 프로그램를 작성해보아라.

5.9.7. 실험64 : PC와 통신4

- 예제 : PORTD에 연결된 누름 스위치가 눌리면 그 값이 눌렸음을 직렬포트를 통해 PC에 알리고 표시한다.
- 회로도 : 그림 120의 회로를 구현하고 SC_CON을 그림 79의 E_CON에 연결한다. 그림 120의 9핀 코넥터(CONDB9)를 PC의 시리얼포트에 연결한다. 그림 80의 SW_CON을 그림 79의 D_CON에 연결한다. 또한 그림 80의 SW1~3을 누르면서 실험한다.
- 프로그램 예 :

```
    #include <mega128.h>
    #include <delay.h>
    void serial_init(void){      delay_ms(10);       DDRE = 0xfe;
        UBRR0H = 0;       UBRR0L = 103;              // baud rate 9600
        UCSR0A = 0;       UCSR0B = 0x18;             UCSR1C = 0x06;
    }
    void putch(unsigned char ch){                /* 한개의 문자를 pc로 송신한다. */
        while((UCSR0A & 0x20) == 0); UDR0 = ch;
        UCSR0A |= 0x20;   // 문자 송신준비완료
    }
    void putstrf(flash unsigned char *str){                /* 문자열을 pc로 송신한다. */
        unsigned int        i=0;
        for(;str[i] != 0; i++) putch(str[i]);
    }
    void main(void){
        serial_init();
        for(;;){
        putch(0x0d); putch(0x0a);                                /* 0x0d는 리턴의, 0x0a는 라인피드의 아스키코드 */
        putstrf("PRESS ANY KEY!!!!₩r₩n");                        /* ₩r은 리턴, ₩n은 라인피드 */
        if(PIND.0 == 0) {putstrf("PRESSED KEY IS D.0₩r₩n");      /* D.0가 눌렸으므로 D.0.가 눌렸음을 알린다. */
        delay_ms(50);                                            /* 50msec 시간 지연 */
        while(PIND.0 == 0);}                                /* D.0가 눌렸다가 안눌린 상태로 복귀할 때가지 기다린다. */
        if(PIND.1 == 0) {putstrf("PRESSED KEY IS D.1₩r₩n"); delay_ms(50); while(PIND.1 == 0);}
        if(PIND.2 == 0) {putstrf("PRESSED KEY IS D.2₩r₩n"); delay_ms(50); while(PIND.2 == 0);}}
    }
```

- 과제 : 타이머0 모드2를 사용하는 프로그램를 작성 하여라.

5.9.8. 실험65 : PC와 통신5

- 예제 : 프로그램을 실행하면 맨 처음에 터미널프로그램인 하이퍼터미널 화면에 뒤에 설명할 'P', 'A', 'R', 'I', 'H' 명령에 대한 설명이 나타나고 그 다음 'Com>> ' 이라는 프롬프트가 나타난다. P라고 PC의 키보드를 치면

'Enter PortC value!' 라는 문자가 나타나고 이때 16진수값을 1바이트 입력시키면 그 값이 직렬포트를 통해 전송되어 PORTC에 연결된 LED에 표시된다. A라고 PC의 키보드를 치면 내부메모리 0x00~0xFF의 값들이 PC로 전송되어 하이퍼터미널 화면에 나타난다. R과 I는 내부메모리 범용레지스터의 값과 I/O레지스터의 값들을 하이퍼터미널 화면에 나타낸다. H와 ?는 메뉴에 대한 설명을 화면에 나타낸다.

- 프로그램 작성시 유의점 : 10개까지의 문자열 입력을 받기 위한 서브루틴 gets를 위한 순서도는 그림 121과 같다. P라는 명령에 의해 입력된 값을 LED에 표시할 때 입력된 값은 아스키코드이므로 이를 16진수값으로 변환할 때 아스키코드로 0~9의 값은 0x30~39이고, A~F의 값은 0x41~46, a~f의 값은 0x61~66이라는 점을 염두에 두어야 한다.

- 회로도 : 그림 120의 회로를 구현하고 SC_CON을 그림 79의 E_CON에 연결한다. 그림 120의 9핀 코넥터(CONDB9)를 PC의 시리얼포트에 연결한다. 그림 79의 LED_SW는 닫고 실험한다.

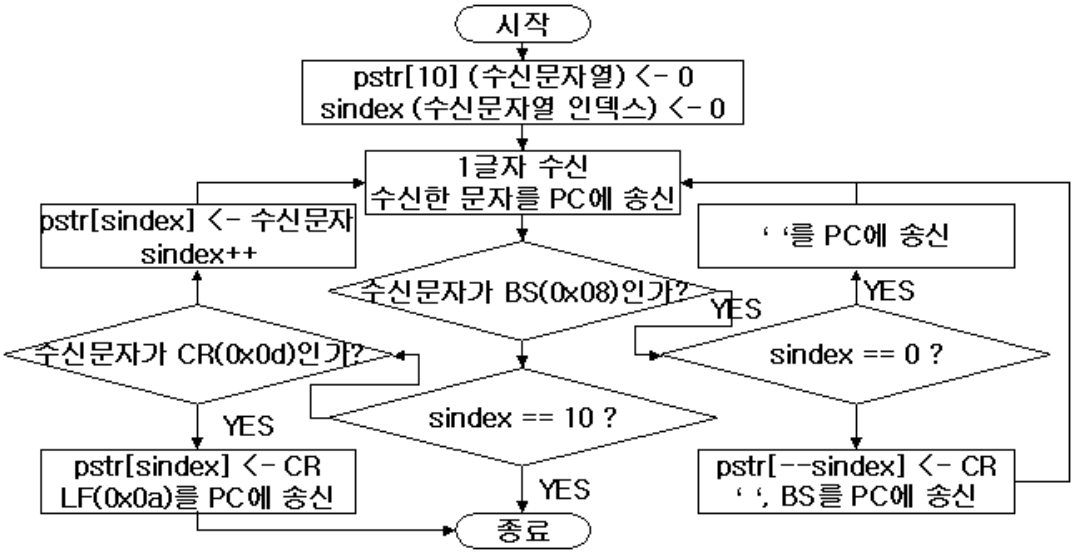


그림 121. 직렬통신에 의한 문자열 수신 순서도

- 프로그램 예 :

```
#include <mega128.h>
#include <delay.h>
#define     DBYTE ((unsigned char *) 0)
#define     BS      0x08      /* 백스페이스 정의 */
unsigned char pstr[10];
void serial_init(void){     delay_ms(10);       DDRE = 0xfe;
    UBRR0H = 0;     UBRR0L = 103;           // baud rate 9600
    UCSR0A = 0;     UCSR0B = 0x18;          UCSR1C = 0x06;
}
void putch(unsigned char ch){               /* 한개의 문자를 pc로 송신한다. */
    while((UCSR0A & 0x20) == 0); UDR0 = ch;     UCSR0A |= 0x20;
}
void putstrf(flash unsigned char *str){             /* 문자열을 pc로 송신한다. */
    unsigned int        i=0;
    for(;str[i] != 0; i++) putch(str[i]);
}
unsigned char asc2hex(unsigned char asc){  /* 아스키코드를 헥사값으로 바꾼다. */
    if ((asc >= '0') && (asc <='9'))        return (asc - '0');
    else if ((asc >= 'A') && (asc <='F'))   return ((asc - 'A')+10);
    else if ((asc >= 'a') && (asc <='f'))   return ((asc - 'a')+10);
    else        return(0xff);
}
unsigned char hex2asc(unsigned char num){ /* 헥사값을 아스키코드로 바꾼다. */
    if(num>=10) return(num+'A'-10);         else return(num+'0');
```

```
}
void putstr(unsigned char *str){     /* 문자열을 pc로 송신한다. */
     unsigned int          i=0;
     for(;str[i] != 0; i++) putch(str[i]);
}
void puthex(unsigned char num){   /* 1바이트 헥사값을 2바이트의 아스키코드로 변환해서 pc로 송신한다. */
     putch(hex2asc(num>>4));       putch(hex2asc(num&0x0f));
}
unsigned       char       getch(void){                    /* 한개의 문자를 pc에서 수신한다. */
     while((UCSR0A & 0x80) == 0); UCSR0A |= 0x80; return(UDR0);
}
void gets(void){                                  /* pc로부터 10개까지의 문자를 수신한다. 10개 이상의 문자는 무시된다. */
     unsigned char ch, sindex=0;
gets1:          ch = getch();
     putch(ch);
     if(ch == BS ){       if(sindex == 0) putch(' ');
                          else { pstr[--sindex] = 0x0d; putch(' '); putch(BS); }
                          goto gets1;
     }
     else {               if(sindex != 10){
                               if(ch != 0x0d) {pstr[sindex++] = ch; goto gets1;}
                               else {pstr[sindex] = ch; putch(0x0a);}
                          }
     }
}
void psubrt(void){          /* P메뉴의 서비스 루틴 */
     DDRC = 0xff;           // PORTC를 출력으로 설정
psubrt1:         putstrf("₩r₩nEnter PortC value ! : ");
     gets();
     if(pstr[0] != 0x0d){
     PORTC = ~((asc2hex(pstr[0]) << 4) | asc2hex(pstr[1]));
     goto psubrt1;}
}
void asubrt(void){          /* a 메뉴의 서비스 루틴 */
     unsigned char i,j;
     putstrf("₩r₩n₩r₩n₩r₩n");
     putstrf("Address 00--01--02--03--04--05--06--07--08--09--0A--0B--0C--0D--0E--0F₩r₩n");
     putstrf("------+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---₩r₩n");
     do{
     putstrf(" 0x");         puthex(j);              putstrf(" : ");                    /* 내부 램의 번지를 pc로 송신 */
               for(i=0;i<=15;i++){                                                      /* 내부 램의 값 16개를 pc로 송신 */
                          puthex(DBYTE[j++]);                          putstrf("   "); }
     putstrf("₩r₩n"); } while(j != 0);
}
void rsubrt(void){          /* r 메뉴의 서비스 루틴 */
     unsigned char i,j;
     putstrf("₩r₩n₩r₩n₩r₩n");
     putstrf("Address 00--01--02--03--04--05--06--07--08--09--0A--0B--0C--0D--0E--0F₩r₩n");
     putstrf("------+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---₩r₩n");
     do{  putstrf(" 0x");  puthex(j);              putstrf(" : ");
               for(i=0;i<=15;i++){   puthex(DBYTE[j++]);                          putstrf("   "); }
     putstrf("₩r₩n"); } while(j != 0x20);
}
void rsubrt(void){          /* i 메뉴의 서비스 루틴 */
     unsigned char i,j=0x20;
     putstrf("₩r₩n₩r₩n₩r₩n");
     putstrf("Address 00--01--02--03--04--05--06--07--08--09--0A--0B--0C--0D--0E--0F₩r₩n");
     putstrf("------+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---₩r₩n");
     do{       putstrf(" 0x");         puthex(j);              putstrf(" : ");
```

```
            for(i=0;i<=15;i++){   puthex(DBYTE[j++]);                    putstrf("   "); }
        putstrf("₩r₩n"); } while(j != 0x60);
}
void hsubrt(void){          /* help 메뉴 */
        putstrf("₩r₩n------------------------------");
        putstrf("₩r₩n H : Help                       ");
        putstrf("₩r₩n ? : Help                      ");
        putstrf("₩r₩n P : Output 0xXX to PortC       ");
        putstrf("₩r₩n A : Dump memory 0x00~FF        ");
        putstrf("₩r₩n R : Dump R0~R31                ");
        putstrf("₩r₩n I  : Dump I/O memory           ");
        putstrf("₩r₩n------------------------------₩r₩n");
}
void main(void){
        serial_init();
        hsubrt();
        for(;;){    putstrf("₩r₩n₩r₩nCom >> ");
                    gets();                          /* 문자열을 입력받는다. */
                    switch(pstr[0]){
                            case 'P': psubrt(); break;      case 'p': psubrt(); break;
                            case 'A': isubrt(); break;      case 'a': isubrt(); break;
                            case 'R': isubrt(); break;      case 'r': isubrt(); break;
                            case 'I': isubrt(); break;      case 'i': isubrt(); break;
                            case 'H': isubrt(); break;      case 'h': isubrt(); break;
                            case '?': isubrt(); break;      case '?': isubrt(); break;
                            case '₩r': break;
                            default: putstrf("₩r₩n₩r₩nINVALID COMMAND!!₩r₩n");
                    }
        }
}
```

- 과제 : ① 문자열 입력을 인터럽트를 사용해서 받는 것으로 구현해보라. ② EEPROM 데이터 값을 읽거나 쓰는 메뉴 아니면 포트 A, B, D, F에 쓰거나 값을 읽어 들이는 등의 다른 메뉴도 구현해 보라.

6. 참고문헌

1. *ATmega128 User Manual,* http://www.atmel.com
2. *CodeVisoinAVR User Manual*, http://www.hpinfotech.ro
3. 최한호, *마이크로프로세서 기초와 응용 강의노트*, http://home.dongguk.edu/user/hhchoi
4. 신동욱, 조영준, *쉬운 예제와 Kit로 배우는 ATmega8515 AVR 기초와 응용*, Ohm사, 2005년
5. 진달복, *ATMEGA8515와 그 응용*, 양서각, 2005년
6. 윤덕용, *AVR ATmega128 마스터*, Ohm사, 2004년
7. 송봉길, *IAR EWAVR 컴파일러를 이용한 AVR ATmega128 마이크로컨트롤러*, 성안당, 2005년
8. 김종부 외, *AVR Embedded System 설계를 위한 ATmega128 이론 및 실험*, 복두출판사,2007년
9. 황해권, 배성준, *AvrEdit3.6과 함께 배우는 AVR: I LOVE ATMEGA 128*, 복두출판사, 2004년

7. 색인

(특수문자와 숫자)

! 60
32, 57
$ 31
% 60
& 60, 65
* 60, 65, 66
*/ 57
*= 61
+ 58, 60, 65
++ 60
+= 61
, 60, 64
- 58, 60, 65
-- 60
-= 61
-> 67
. 67
/ 60
/* 57
// 57
/= 61
: 30, 62
; 30, 56
< 60
<< 60
= 61
> 60
>> 60
@ 31
[64, 66
₩ 57
] 64, 66
^ 60
_ 30, 58
{ 56, 64
| 60
} 64
~ 60
10진수 58
16진수 58
2SA1015Y 134
2SB601 134
2SC1815Y 118, 134
2SD560 134
74LS46 109
74LS47 109, 110
8진수 58

(A)

A/D변환기 143
A1273Y 134
ACI 148
ACIE 148
ACME 148
ACSR 148
ADC 21, 143
ADC0804 143
ADCH 144
ADCSRA 144
ADD 27
ADEN 144, 149
ADIF 144
ADMUX 144
AEC 42
ALE/PROG 4
AND 27
asm 64
ASSR 90
AVR ISP 42
AVR studio 42

(B)

BCD 70, 71
BDX53C 134
BDX54C 134
bit 59
BLD 27
BOOTRST 16
BOOTSZ 16
BRBS 27
BRCS 27
BREAK 27
break 61, 62

BRTS 27
BSET 27
BYTE 31

(C)
C3205Y 134
CALL 27
case 61
CBI 27
CBR 27
char 58
CISC 1, 3
CLI 27
CLR 27
Codevision 43
COM 27
const 59
continue 62
CP 27
CSEG 30
CTC 91, 98

(D)
D/A변환기 140
DAC 141
DAC0800 141
DATA 31
DB 31
DD 31
DDRx 74
DEC 27
DEF 31
default 61
define 31, 57, 59
delay.h 73
do 62
double 58
DQ 31
DSEG 30
DW 31

(E)
EEAR 12
EECR 12
EEDR 12
EEPROM 12, 68
eeprom 59
EICRA 85
EICRB 85
EIFR 85
EIMSK 85
ELIF 31
elif 57
ELPM 27
ELSE 31
else 31, 57, 61
endasm 64
ENDIF 31
endif 31, 57
ENDM 31
ENDMACRO 31
enum 67
EOR 27
EQU 30, 31
ERROR 32
ESEG 30
ETIFR 97
ETIMSK 96
extern 64

(F)
flash 59
float 58
FMUL 27
FND 109
for 62
FR105 134
Fuse 16

(G)
goto 62

(H)
HD44780 122
H브리지 133

(I)
I/O 8, 10
ICALL 27
ICE 40

ICR 95
Idle 모드 20
IF 31
if 57, 61
IFDEF 31
ifdef 57
IFNDEF 31
ifndef 57
IJMP 27
IN 27
INC 27
INCLUDE 31
include 31, 57
int 58
interrupt 63
ISP 1, 3, 41, 54

(J)
JMP 27

(L)
L298 134
Label 30
LABEL 30
label 62
LCD 109, 122
lcd.h 125
LD 27
LDI 27
led 76
long 58
LPM 27
LSL 27

(M)
MACRO 31
main 57, 62
math.h 63
MAX232 154
MCUCR 14, 20, 83
MCUCSR 25
mega128.h 76
MISO 3
MOSI 3
MOV 27
MOVW 27
MUL 27

(N)
NEG 27
NOP 27

(O)
OCR 89, 94
OP 27
Operand 27
OR 27
ORG 31
OUT 27

(P)
PINx 74
POP 27
PonyProg2000 42
PORTx 74
pragma 64
Prescaler 89
PUSH 27
PWM 92, 93, 98, 99, 106, 107, 139

(R)
RAMPZ 10
RCAL 27
RET 27
return 63
RISC 1, 3
RJMP 27
ROL 27
RS232C 150, 154
RST(ReSeT) 5, 74

(S)
SBC 27
SBI 27
sbit 59
SBR 27
SCLK 3
SEC 27
SEI 27
SEN 27

SER 27
SET 27, 31
SFIOR 74, 91, 148
sfrb 59
SLEEP 27
Sleep 20
SPCR 151
SPI 3, 41, 150
SPI2X 152
SPIE 151
SPIF 151
SPM 27
SPSR 151
SRAM 10
SREG 9
ST 27
static 64
stdio.h 63
stdlib.h 63
string.h 63
struct 66
SUB 27
SWAP 27
switch 61
Symbol 30

(T)
TCCRn 89
TCCRxA 95
TCCRxB 95
TCCRxC 96
TCLK 94
TCNTn 89
TCNTx 94
TIFR 90, 97
TIMSK 90, 96
TOVn 89
TOVx 94
TST 27
typedef 66, 67

(U)
UBRRn 154
UCSRnA 152
UCSRnB 153
UCSRnC 153
UDRn 152
UNDEF 31
undef 31, 57
union 67
unsigned 58
USART 152

(V)
void 57, 63, 66
volatile 68

(W)
WARNING 32
Watchdog 3
WDTCR 25
while 62

(X)
X,Y,Z 레지스터 8
XDIV 20
XMCRA 14
XMCRB 14

(ㄱ)
간접지정 27

(ㄴ)
내부램 7

(ㄷ)
도트 67
동적구동 110
디버깅 37, 51

(ㄹ)
라벨 30
레지스터지정 27
롬라이터 41
롬에뮬레이터 40
리셋 22
리턴데이터형 63

(ㅁ)
메모리 LOCK 15

명령코드 27
모터 129
문자 58
문장 56

(ㅂ)
발진회로 17
배열 64
범용 레지스터 8
변수 58
블록 56
비교기 147

(ㅅ)
상수 58
상태레지스터 9
소스파일 57
숫자 58
스택 10
스텝모터 129
스톱워치 105, 112
시간지연 73
시계 111, 113, 128
심볼 56

(ㅇ)
오퍼랜드 27
외부롬 68, 69, 70
외부변수 64
음계 117
음향 117
인수 57, 62, 63
인터럽트 82, 105

(ㅈ)
전역변수 63
정적변수 64
주소지정방식 27
지시어 56
지역변수 63
직렬통신 117
직류모터 137
직접상수지정 27
직접지정 27

(ㅊ)
채터링 78

(ㅋ)
카운터 104
키보드 115
키워드 56

(ㅌ)
타이머 88, 105
터미널프로그램 154

(ㅍ)
포인터 64, 65
프로그램 56
프로토타입 57, 62

(ㅎ)
하이퍼터미널 154
함수 63
헤더파일 57