색인

1 첫 프로젝트 만들기: 입력으로 출력 켜기 6
1-1 이 장에서 만들 것 6
1-2 화면 둘러보기 7
1-3 화면 알림 읽기 9
1-4 새 프로젝트 만들기와 보드 선택 9
1-4-1 쓸 핀을 먼저 켜기 11
1-5 래더 편집: P0 접점에서 P32 코일까지 12
1-6 래더는 어떻게 아두이노 C 코드가 되나 14
1-7 빌드하기 15
1-8 포트 선택과 업로드 16
1-9 동작 확인: 시리얼 모니터와 실시간 모니터링 18
1-10 이 튜토리얼에서 사용한 단축키 20
2 필독사항 22
2-1 데이터 메모리 22
2-1-1 표현 범위 22
2-1-2 영역 크기와 메모리 설정 23
2-1-3 접두어와 비트 접근 24
2-1-4 사용한 메모리 확인 25
2-2 특수 메모리 26
2-3 단축키 27
2-4 핀 편집 30
2-4-1 보드 핀맵 편집 30
2-4-2 래더 셀 편집 32
2-4-3 핀 설정 따라 하기 32
3 메뉴 툴바 36
3-1 파일 36
3-1-1 환경설정 37
3-2 편집 45
3-3 보기 47
3-4 실행 47

3-4-1 통신 54
3-4-2 메모리 쓰기 목록 55
3-5 설정 57
3-6 도움말 59
4 래더 그리드 편집 61
4-1 그리드 하단 상태표시줄 61
4-2 커서 이동과 선택 61
4-2-1 박스함수 위에서의 커서 62
4-2-2 범위 선택 62
4-3 래더 셀 우클릭 메뉴 63
4-4 아두이노에 값 쓰기 65
4-5 셀 삽입과 삭제 66
4-6 줄 추가와 삭제 68
4-7 클립보드와 실행취소 70
5 접점 72
5-1 접점 종류 72
5-2 NO 접점과 NC 접점 75
5-2-1 NO 접점 (A접점) 75
5-2-2 NC 접점 (B접점) 76
5-2-3 상수 접점 (@ON / @OFF) 77
5-3 코일 출력 78
5-3-1 출력 코일 78
5-3-2 SET 코일 78
5-3-3 RESET 코일 79
5-3-4 입력핀을 코일 대상으로 쓰기 80
5-4 신호 반전 80
5-5 상승/하강 검출 접점 82
5-5-1 상승 검출 접점 (상승엣지) 82
5-5-2 하강 검출 접점 (하강엣지) 83
5-6 비교 접점 84
5-7 자기유지 회로 85
5-8 워드 메모리의 비트 접점 86
5-9 아두이노 변수 사용 87

5-9-1 쓸 수 있는 변수 88
5-9-2 코일로 쓰기 89
5-9-3 입력하기 89
5-9-4 모니터링 중에는 값을 알 수 없습니다 90
5-9-5 오류와 해결 90
5-9-6 이름이 겹칠 때 91

6 박스함수 93

6-1 타이머 95
6-1-1 TON — 온 딜레이 타이머 96
6-1-2 TOFF — 오프 딜레이 타이머 97
6-1-3 TPL — 펄스 타이머 98
6-1-4 TMON — 모노스테이블 타이머 99
6-1-5 TMR — 적산 타이머 100
6-1-6 타이머 단위 설정 101
6-1-7 메모리로 프리셋 지정 (HMI에서 시간 바꾸기) 103
6-2 카운터 104
6-2-1 CTU — 가산 카운터 105
6-2-2 CTD — 감산 카운터 106
6-2-3 CTUD — 가감산 카운터 107
6-2-4 메모리로 프리셋 지정 (HMI에서 카운트 바꾸기) 108
6-3 산술 111
6-3-1 사칙연산 (ADD·SUB·MUL·DIV·MOD) 111
6-3-2 증감 (INC·DEC) 112
6-3-3 SCALE — 스케일 변환 113
6-3-4 실수 연산 (FLOOR·SQRT) 115
6-4 전송 115
6-4-1 MOV — 값 전송 116
6-4-2 블록 전송 (GMOV·FMOV) 116
6-5 비트연산 118
6-5-1 비트 논리 (WAND·WOR·WXOR·WNOT) 118
6-5-2 시프트 (SHL·SHR) 119
6-5-3 회전 (ROL·ROR) 120
6-6 변환 121
6-6-1 반전 (INV·NEG) 121

6-6-2 BCD 변환 (H2D·D2H) 122
6-7 비트/래치 123
6-7-1 SET·RST 124
6-8 로직 제어 125
6-8-1 조건부 점프 (JMP·JMPN) 125
6-8-2 마스터 컨트롤 (MCS·MCSCLR) 126
6-9 자유입력형 박스함수 128
7 코드 편집과 자동완성 130
7-1 코드 에디터 130
7-2 코드 탭 운용 131
7-3 자동완성 132
7-4 정의로 이동 133
7-5 찾기와 바꾸기 135
7-6 하단 패널 읽기 136
8 시리얼 모니터, 래더 모니터링 139
8-1 시리얼 모니터 개요 139
8-2 연결하기 139
8-3 데이터 수신 141
8-4 데이터 송신 142
8-5 래더 모니터링 시작하기 144
8-6 도통과 라이브 값 읽기 146
8-7 표시 형식과 특수 표시 147
8-8 시리얼 모니터와 래더 모니터링 149
9 활동바와 사이드 패널 150
9-1 활동바 개요 150
9-2 탐색기 패널 152
9-3 래더설정 패널 154
9-4 라이브러리 패널 156
9-5 심볼 160
9-5-1 심볼 창 열기 160
9-5-2 화면 구성 161
9-5-3 등록·편집·삭제 161
9-5-4 이름 규칙 161

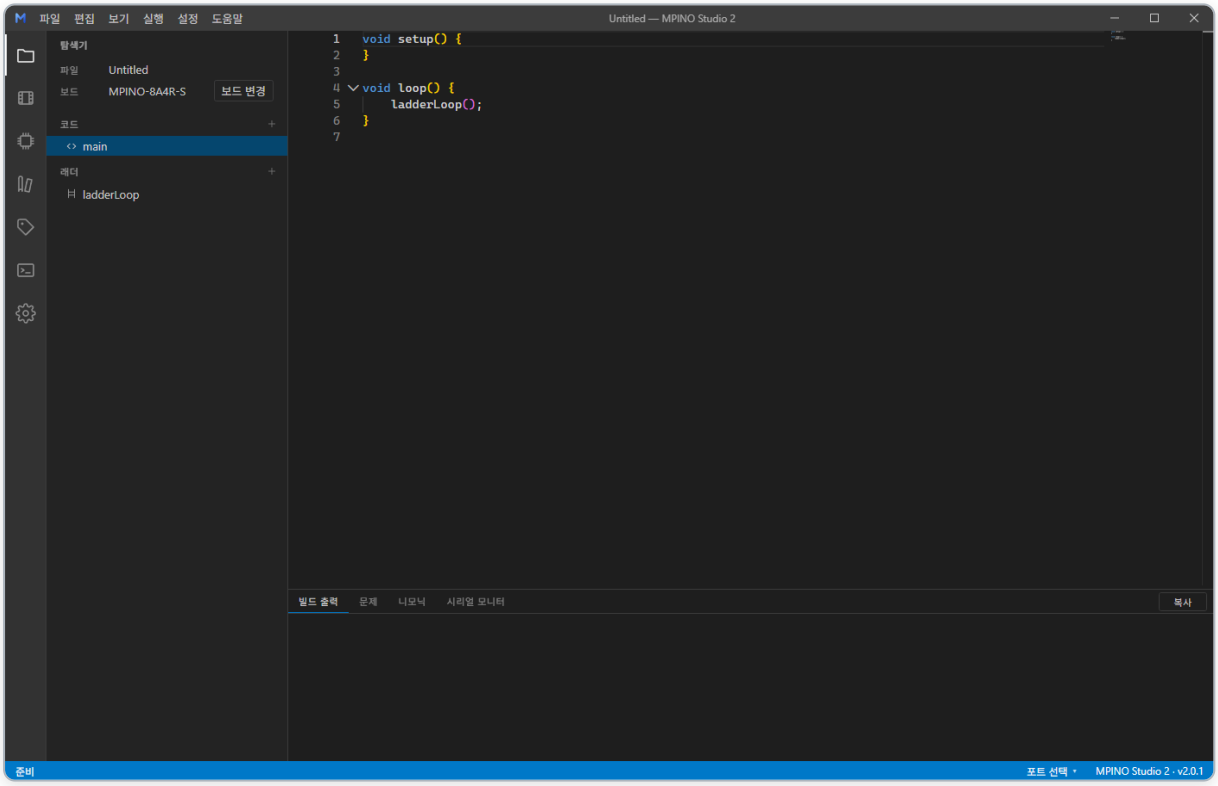
9-5-5 주소·이름을 바꾸면 162
9-5-6 가져오기·내보내기 163
9-5-7 C 코드에서 쓰기 163
9-6 그 밖의 활동바 항목 163

1 첫 프로젝트 만들기: 입력으로 출력 켜기

1-1 이 장에서 만들 것

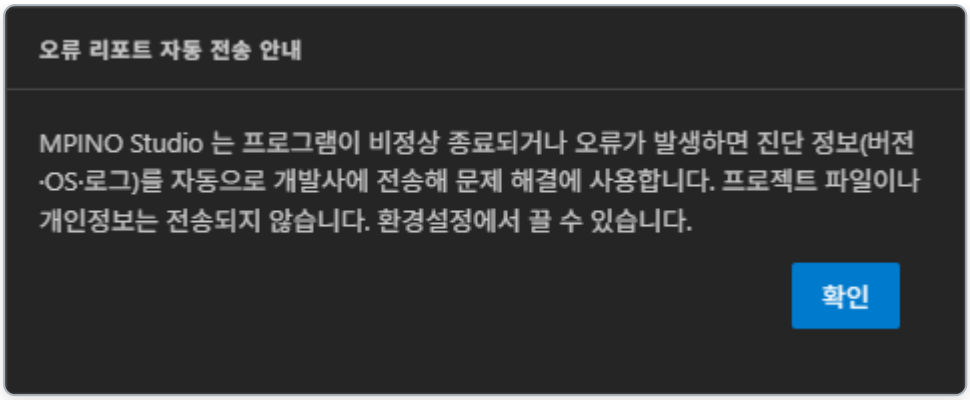
MPINO Studio 2는 ILOGICS MPINO 산업용 PLC 보드를 위한 데스크톱 IDE입니다. 아두이노 C 코드와 래더 로직을 한 화면에서 동시에 편집하고, 빌드하면 래더 로직이 자동으로 C 함수로 변환되어 `ladderLoop()` 안에서 실행됩니다. 이 장에서는 앱을 처음 실행하는 순간부터 새 프로젝트를 만들고, 입력 접점 하나가 출력 릴레이 하나를 켜는 가장 단순한 회로를 그려서 빌드하고, 실제 보드에 업로드해 동작을 확인하기까지 전체 흐름을 한 번에 따라갑니다.

앱을 처음 실행하면 "환영 화면"이나 "프로젝트가 없습니다" 같은 빈 안내 화면은 나타나지 않습니다. 대신 이미 완전히 동작하는 기본(Untitled) 프로젝트가 곧바로 열려 있습니다. 보드는 MPINO-8A4R(T)-S가 기본값으로 잡혀 있고, 코드 탭 **main**과 빈 래더 탭 **ladderLoop**가 준비돼 있어 바로 편집을 시작할 수 있습니다. 앱 창은 실행할 때마다 화면을 가득 채운 최대화 상태로 열립니다.



< 그림 1-1 앱을 처음 실행하면 나타나는 기본 화면 — 빈 화면이 아니라 이미 동작하는 기본 프로젝트다 >

MPINO Studio 2를 맨 처음 실행하면, 위 화면과 함께 "오류 리포트 자동 전송 안내"라는 제목의 안내 창이 한 번 뜹니다. 프로그램이 비정상 종료되거나 오류가 발생하면 진단 정보(버전·OS·로그)를 자동으로 개발사에 전송해 문제 해결에 쓴다는 내용이며, 프로젝트 파일이나 개인정보는 전송되지 않는다고 함께 안내합니다. **확인** 버튼을 눌러 닫으면 다음 실행부터는 다시 나타나지 않습니다. 자동 전송을 원치 않으면 나중에 환경설정 › 일반 탭의 **오류 리포트 자동 전송**(3-1-1) 체크박스에서 끌 수 있습니다.



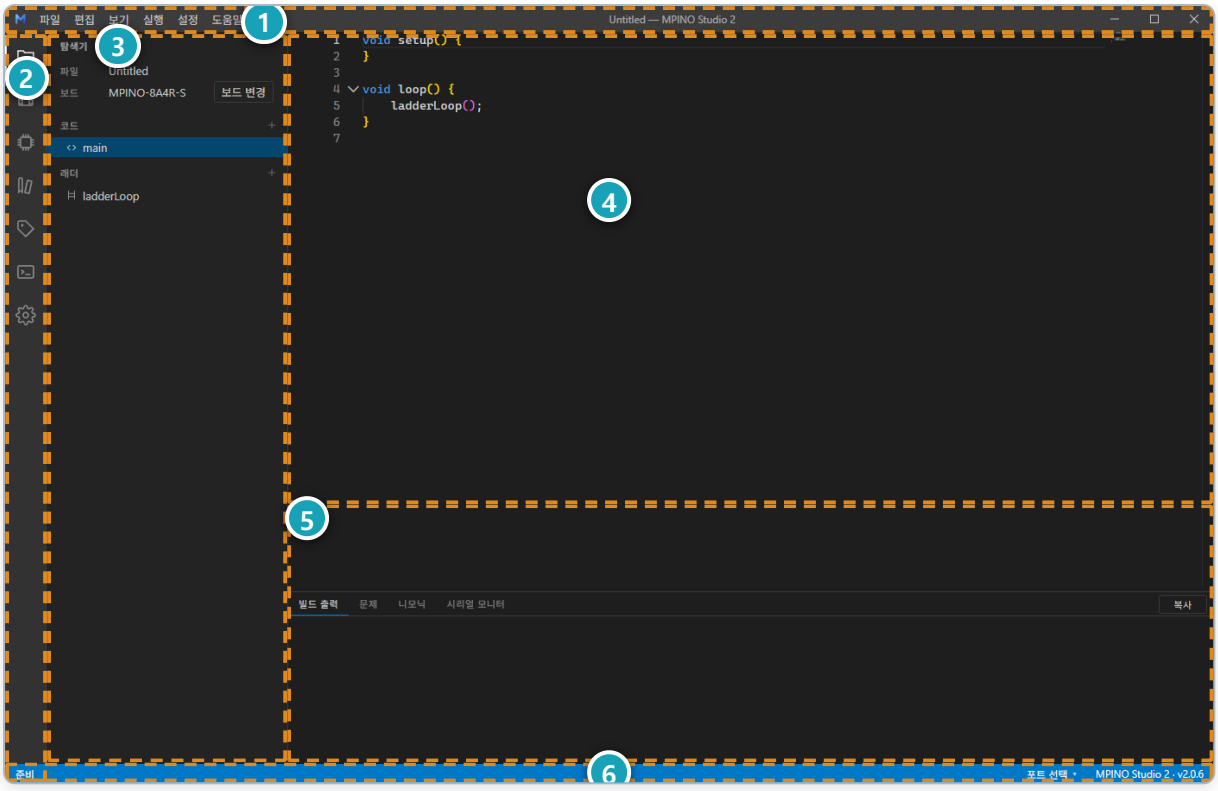
< 그림 1-2 최초 실행 고지 — 오류 리포트 자동 전송 안내 창(확인을 누르면 다시 뜨지 않음) >

앱은 실행할 때마다 한 번 **업데이트가 있는지 자동으로 확인**합니다. 새 버전이 있으면 버전 정보와 함께 [지금 업데이트]·[나중에] 버튼이 있는 "업데이트 확인" 창이 뜨고, 최신 버전이거나 서버에 연결하지 못하면 아무 안내 없이 조용히 넘어갑니다. 원할 때 직접 확인하려면 도움말 메뉴의 **업데이트 확인**을 쓰며, 이때는 결과가 항상 토스트로 표시됩니다(**도움말**(3-6) 참고).

> 프로젝트 파일(`.mp2`)은 일반 텍스트(TOML) 형식이라 버전 관리 도구로 변경 이력을 추적하기 좋습니다.

1-2 화면 둘러보기

본격적으로 손을 대기 전에, 화면을 이루는 여섯 개 영역부터 눈에 익혀 두겠습니다. 아래 화면은 좌측 액티비티바에서 **탐색기**를 선택했을 때의 기본 레이아웃이며, 각 영역의 이름과 주요 동작은 그림 아래 범례에 정리되어 있습니다.



< 그림 1-3 MPINO Studio 2 기본 레이아웃 — 메뉴바·액티비티바·사이드바·에디터·하단 패널·상태바 >

① **메뉴바** — 파일·편집·보기·실행·설정·도움말 여섯 개 최상위 메뉴. 새 프로젝트·저장·빌드·업로드 등 거의 모든 명령이 여기서 출발

② **액티비티바** — 탐색기·래더설정·래더 핀맵 설정·라이브러리·심볼·시리얼 모니터·설정 일곱 개 아이콘. 클릭하면 사이드바 패널이 전환되거나 별도 모달이 열림(시리얼 모니터는 하단 패널로 이동)

③ **사이드바(탐색기)** — 현재 프로젝트의 파일/보드 메타 정보와 코드·래더 탭 트리. 행을 클릭하면 그 코드/래더가 에디터 영역에 표시됨

④ **에디터 영역** — 코드 에디터와 래더 에디터가 겹쳐 있는 공간. 사이드바 탭 트리를 클릭해 전환(상단 탭 스트립 없음)

⑤ **하단 패널** — 빌드 출력·문제·니모닉·시리얼 모니터 네 개 탭. 빌드/업로드 실행 시 자동으로 빌드 출력 탭으로 전환

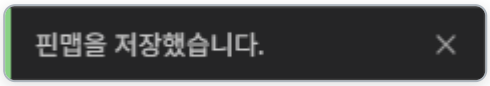
⑥ **상태바** — 좌측 준비 텍스트, 우측 포트 선택 드롭다운과 버전 표시. 업로드·시리얼 통신 전반의 유일한 포트 선택 경로

몇 가지는 이름만 보고 짐작하면 어긋납니다. 액티비티바의 **래더설정** 아이콘은 보드를 갈아끼우는 기능이 아니라 현재 보드의 핀 매핑과 메모리 영역을 읽기 전용으로 보여주는 패널을 엽니다 — 이 패널 자체에는 보드를 바꾸는 버튼이 없습니다. 다만 프로젝트를 새로 만들지 않고 대상 보드만 바꾸는 기능은 따로 있습니다 — 탐색기 패널의 "보드" 줄 옆 **[보드 변경]** 버튼이나 설정 메뉴의 **보드 변경**을 쓰면, 코드와 래더는 그대로 둔 채 대상 보드만 교체됩니다(자세한 내용은 **탐색기 패널**(9-2)과 설정 메뉴의 **보드 변경**(3-5) 참고). 또 RAM 칩 모양의 아이콘은 생김새와 달리 메모리 에디터가 아니라 **래더 핀맵 설정** 모달을 엽니다(툴팁도 "래더 핀맵 설정"입니다).

화면 맨 아래 **상태바** 왼쪽에는 "준비"가 항상 표시됩니다. 빌드나 모니터링이 진행 중이어도 이 문구는 바뀌지 않는 정적 표시입니다. 오른쪽에는 시리얼 포트를 고르는 **포트 버튼**(자세한 사용법은 아래 포트 선택과 업로드 절의 포트 드롭다운(그림 1-9) 참고)과 앱 버전 "MPINO Studio 2 · v2.2.0"가 나란히 표시됩니다.

1-3 화면 알림 읽기

저장 성공이나 오류, 간단한 안내처럼 여러 동작의 결과는 화면 오른쪽 아래에 잠깐 나타났다 사라지는 알림(토스트)으로 표시됩니다.



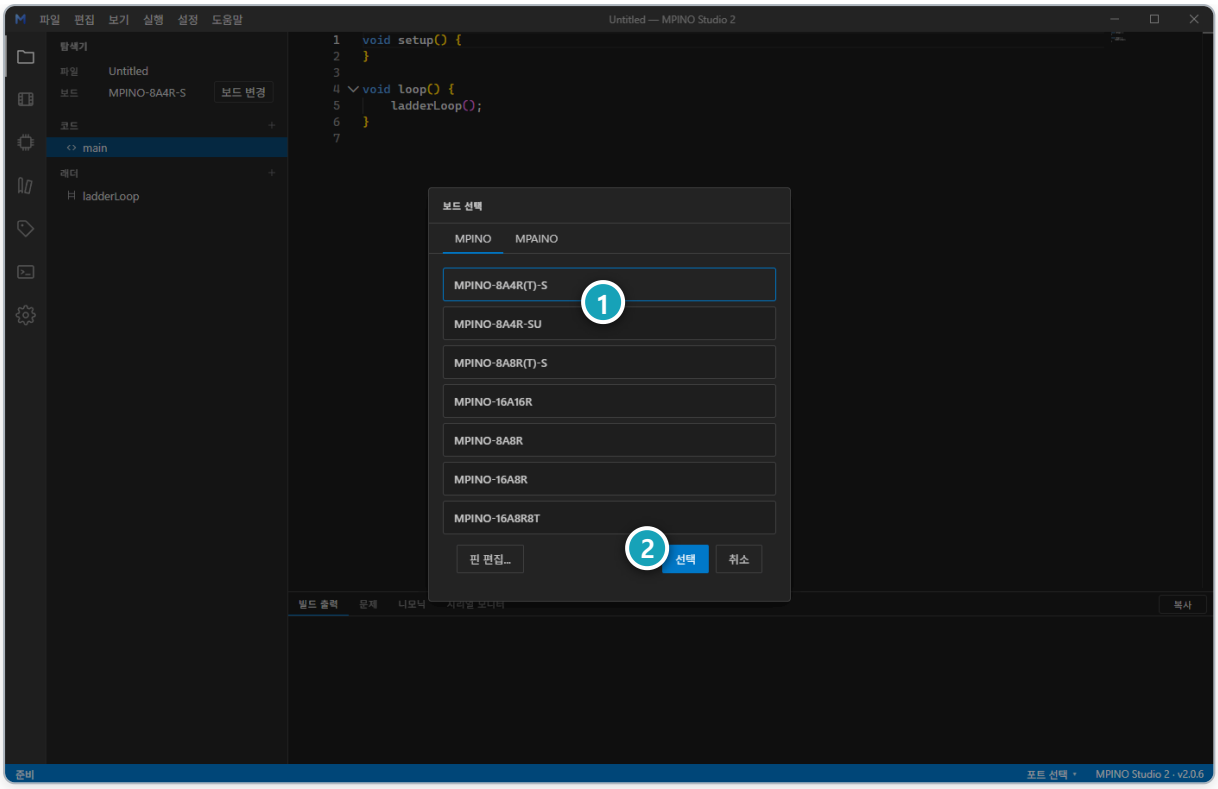
< 그림 1-4 화면 오른쪽 아래에 잠깐 나타났다 사라지는 알림(토스트) 예시 — 핀맵 저장 성공 시 뜨는 초록색 토스트 >

토스트는 약 **4초** 후 자동으로 사라집니다. 급하게 닫고 싶으면 알림의 ✕ 버튼을 클릭하거나 **Esc** 키를 눌러 바로 닫을 수 있습니다. 단, **저장 실패** 알림만은 자동으로 사라지지 않고 이렇게 직접 닫을 때까지 남습니다(저장이 안 됐다는 사실을 놓치지 않게 하기 위해서입니다). 종류는 **오류·정보·성공** 세 가지이며, 왼쪽 세로선 색으로 구분됩니다(오류는 빨강, 정보는 강조색, 성공은 초록). 한 번에 **최대 3개까지** 세로로 쌓여 보이고, 새 알림은 아래쪽(화면 모서리에 가까운 쪽)에 추가되며 이전 알림은 그 위로 밀려 올라갑니다. 토스트는 뒤 화면을 가리지 않고 별도 조작을 요구하지 않는다는 점에서, 확인을 눌러야 닫히는 모달이나 다이얼로그와는 다릅니다.

1-4 새 프로젝트 만들기와 보드 선택

이제 진짜 프로젝트를 하나 만들어 보겠습니다. 메뉴 경로는 **파일 > 새로 만들기**, 단축키는 **Ctrl+N** 입니다.

1. **파일** 메뉴에서 **새로 만들기**를 클릭하거나 `Ctrl+N`을 누릅니다.
2. 편집 중이던 프로젝트에 저장하지 않은 변경 사항이 있으면 **변경 내용 버리기** 확인 창이 먼저 뜹니다. 계속하려면 **버리기**를, 취소하려면 **취소**를 누릅니다.
3. 곧바로 **보드 선택** 모달이 열립니다. 프로젝트 이름을 먼저 묻지 않는다는 점이 특이한데, 이름과 저장 위치는 나중에 저장할 때(Ctrl+S) 한 번에 정해집니다.
4. 모달 위쪽의 **MPINO** / **MPAINO** 두 탭 중 **MPINO** 탭이 열려 있는지 확인합니다 (탭은 클릭으로 바꾸며, 모달에 포커스가 있는 상태에서 ←/→ 방향키로도 전환됩니다).
5. 목록에서 **MPINO-8A4R(T)-S** 카드를 클릭해 선택합니다. 이 보드는 디지털 입력 P0~P7과 디지털 출력 P32~P35를 갖고 있어, 이번 튜토리얼의 P0 입력·P32 출력 회로에 딱 맞습니다.
6. **선택** 버튼을 클릭하거나 카드를 더블클릭해 확정합니다(Enter 키로도 확정, Esc나 바깥 클릭으로는 취소됩니다).



< 그림 1-5 보드 선택 모달 — MPINO-8A4R(T)-S가 선택된 상태 >

① **보드 카드** — 클릭해 사용할 보드를 선택

② **선택 버튼** — 선택을 확정(더블클릭·Enter도 가능)

선택을 확정하면 디스크에는 아무것도 쓰지 않고, 메모리 위에서만 코드 탭 **main**과 빈 래더 탭 **ladderLoop**를 가진 새 프로젝트가 만들어집니다.

보드 목록은 **MPINO**와 **MPAINO** 두 카테고리 탭으로 나뉘며, 두 탭을 합쳐 총 열다섯 종이 준비되어 있습니다. MPINO 탭에는 이 보드 말고도 MPINO-8A4R-SU, MPINO-8A8R(T)-S, MPINO-8A8R, MPINO-16A16R, MPINO-16A8R, MPINO-16A8R8T가 있고, MPAINO 탭에는 MPAINO-8A8R부터 MPAINO-64A64R까지 여덟 종이 있습니다. 실제 현장에서는 손에 든 보드에 맞춰 고르면 됩니다.

MPAINO 탭의 보드를 선택하면 목록 아래에 **옵션 모듈** 구획이 함께 나타납니다. **아날로그 입력 : X**, **아날로그 출력 : Y**, **PT100Ω 입력 : F**, **펄스 출력 : K** 네 가지를 각각 [−]/[+] 버튼으로 몇 개 붙일지 지정하는 자리입니다. 붙이지 않은 모듈은 "없음"으로, 붙인 모듈은 "X1"·"Y2"처럼 접두 문자와 개수로 표시되며 처음에는 넷 다 "없음"입니다. 개수에는 상한이 있어(X 5, Y 4, F 5, K 2) 더 늘릴 수 없는 상태가 되면 [+] 버튼이 비활성화되고, 그 위에 마우스를 올리면 이유가 툴팁으로 표시됩니다. 모듈 사이에도 합계 제한이 있습니다 — "아날로그 입력(X)과 PT100Ω 입력(F)은 합계 5모듈까지", "아날로그 출력(Y)과 펄스 출력(K)는 3×Y + 6×K ≤ 12 채널까지"입니다. 펄스 출력(K)을 아날로그 출력(Y)과 함께 붙이거나 펄스 출력(K)을 2모듈 붙이면 "디지털 입력 단자가 D2~D9에서 D22~D29 위치로 이동합니다."라는 안내도 함께 표시됩니다. 그 구성에서는 K가 D2~D8 실핀을 쓰기 때문입니다.

> **아날로그 출력(Y)·펄스 출력(K) 채널에 넣는 값.** 모듈의 각 채널은 C 워드 주소에 연결되며 (주소는 **핀 편집**(2-4) 표의 LD 열에서 확인 — 예: Y 첫 모듈 1채널 = C20), 래더에서 그 주소에 값을 넣으면 스캔 끝에 출력됩니다. 핀 편집에서 그 채널의 **사용**을 켜야 출력됩니다.

- **Y(아날로그 출력)**: 0~65535. 값에 비례해 0 = 0 V, 65535 = 약 10 V가 나갑니다(예: 16384 ≈ 2.5 V, 32768 ≈ 5 V). 사용을 켠 Y·K 채널로 가는 **MOV**(`MOV 65535 C20`, `C20 = 65535`)는 0~65535를 그대로 받습니다 — 경고 없이 출력에 65535가 나갑니다(MPINO-16A8R8T의 Aout·PWMout 같은 보드 내장 아날로그·PWM 출력도 사용을 켜면 같습니다). 65535보다 큰 값은 하위 16비트만 들어가고 경고가 뜹니다. MOV가 아닌 계산(`ADD` 등)이나 비교에서는 이 예외가 없습니다(**표현 범위**(2-1-1) 참고). 래더 모니터링도 사용을 켠 Y·K 채널 주소의 값을 0~65535로 표시합니다.
- **K(펄스 출력)**: 0~255. 약 490 Hz PWM의 듀티로, 0 = 항상 LOW, 255 = 항상 켜짐입니다.

목록 아래에는 **Serial RX 버퍼** 구획도 있습니다. 보드가 가진 UART 채널마다 수신 버퍼 크기를 64·128·256·512바이트 중에서 고르는 곳으로, 기본은 접힌 상태이고 제목 줄을 클릭하면 펼쳐집니다. 보드 기본값에서 바꾼 채널이 있으면 접어 두어도 제목 옆에 "Serial1 256바이트"처럼 바뀐 채널만 요약되어 보이고, 전부 기본값이면 요약이 표시되지 않습니다. 이 구획은 버퍼 선택을 지원하는 보드에서만 나타나므로, 이번 튜토리얼에서 고른 MPINO-8A4R(T)-S에서는 보이지 않습니다.

1-4-1 쓸 핀을 먼저 켜기

래더를 그리기 전에 반드시 거쳐야 하는 단계가 하나 있습니다. **새 프로젝트의 핀은 모두 "사용 안 함" 상태로 시작합니다.** 쓸 핀을 먼저 켜 두지 않으면 회로를 아무리 정확하게 그려도 빌드가 거절됩니다.

> **핀의 기본값은 "사용 안 함"입니다.** 래더가 꺼진 핀을 참조한 채로 빌드하면 빌드 출력에 "핀 P0 (Din_1)은(는) 보드 정의에서 비활성화되어 있습니다(usepin = false). 보드 핀 편집기에서 활성화하거나 다른 핀을 사용하세요."라는 오류가 표시되고 컴파일이 중단됩니다. 이 기본값은 래더에서 쓰지도 않는 출력 핀을 펌웨어가 매 스캔 0으로 덮어써, 아두이노 코드에서 직접 제어하던 출력이 눌리는 것을 막기 위한 것입니다.

이번 튜토리얼에서 쓸 핀은 입력 **P0**와 출력 **P32** 두 개입니다. 다음 순서로 켭니다.

1. `Ctrl+Shift+P`를 누르거나 설정 메뉴의 **래더 핀맵 설정**을 클릭합니다. 보드의 핀이 한 줄씩 나열된 핀 편집 표가 열립니다.
2. **P0** 행과 **P32** 행을 찾아 **사용** 열의 체크박스를 켭니다. 표에서 쓸 핀이 많다면 **사용** 열 **머리글의 체크박스**를 한 번 눌러 전체를 한꺼번에 켜고 끌 수도 있습니다(일부만 켜져 있으면 머리글 체크박스는 중간 상태로 표시됩니다).
3. **[확인]** 버튼을 눌러 창을 닫습니다.
4. `Ctrl+S`로 프로젝트를 저장합니다 — 핀 설정은 현재 프로젝트(`.mp2`)에 저장되므로 저장해야 디스크에 기록됩니다.

핀을 켜 두면 부수 효과가 하나 더 있습니다. 뒤에서 래더 셀에 주소를 입력할 때 뜨는 **자동완성 후보에는 켜져 있는 핀만 나타납니다.** 새 프로젝트에서 자동완성 목록이 비어 보인다면 핀이 아직 하나도 켜지지 않은 것입니다.

> **이미 만들어 둔 프로젝트를 열 때도 마찬가지입니다.** 이전에 저장한 `.mp2`에 켜진 핀이 적혀 있지 않으면 모든 핀이 "사용 안 함"으로 열립니다. 예전에 잘 빌드되던 프로젝트가 갑자기 위 오류로 막힌다면, 핀 편집 표에서 쓰던 핀을 다시 켠 뒤 저장하세요.

핀 편집 표의 열 구성과 각 열의 의미는 **핀 편집**(2-4)에서 자세히 다룹니다. 이 회로로 핀을 켜는 과정과 켜지 않았을 때 나오는 오류는 **핀 설정 따라 하기**(2-4-3)에서 그림과 함께 따라갈 수 있습니다.

1-5 래더 편집: P0 접점에서 P32 코일까지

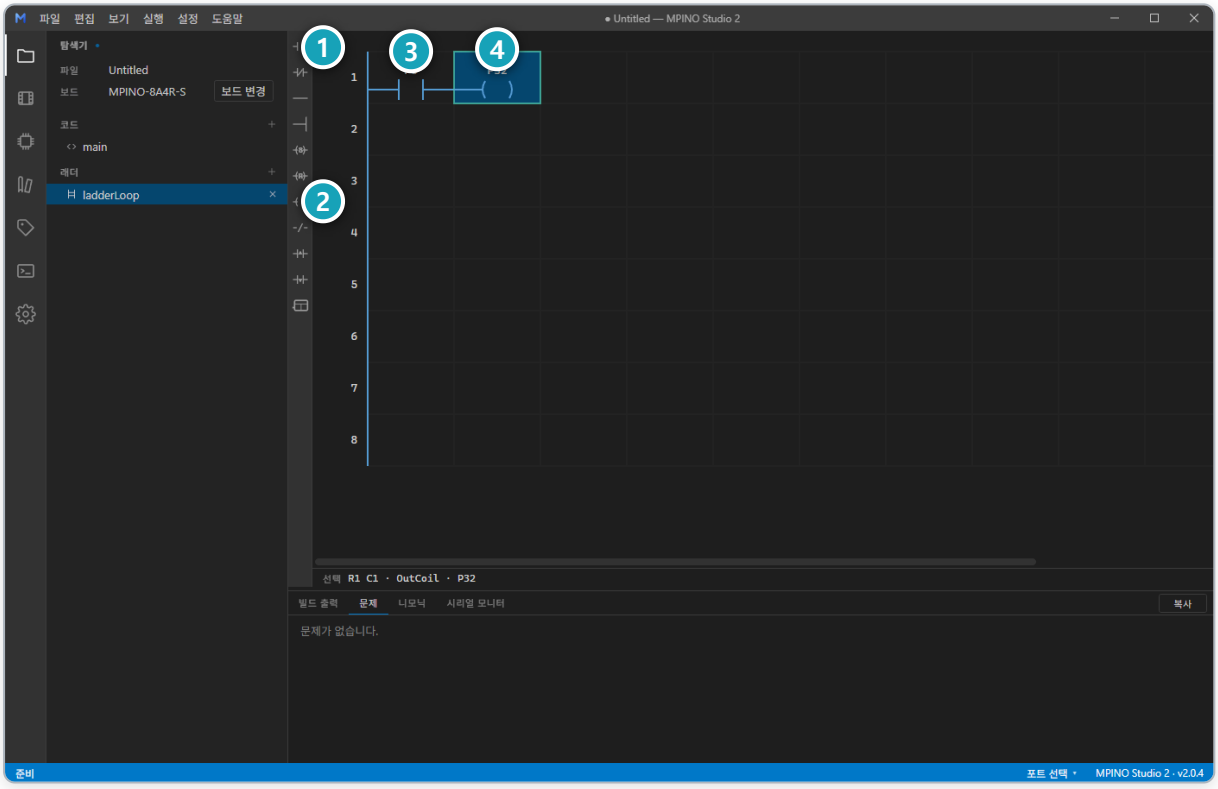
이제 핵심입니다. "P0가 켜지면 P32가 켜진다"는 최소 회로를 래더 그리드에 직접 그려보겠습니다. 먼저 사이드바 탭 트리에서 **래더** 그룹의 **ladderLoop** 행을 클릭해 래더 에디터로 이동합니다.

래더 에디터 왼쪽에 세로로 배치된 툴바에는 열한 가지 도구가 위에서 아래로 나열되어 있습니다. 위쪽 열 개는 기능 키(F2~F12)에 순서대로 대응하고, 맨 아래 박스함수만 `\`(백슬래시 — 한국어 키보드에서는 `₩` 키)입니다. 셀을 하나 선택한 상태에서 이 중 하나를 클릭하거나 단축키를 누르면, 별도의 "도구 모드" 없이 그 셀에 즉시 적용됩니다.

단축키	도구	기호	설명
F2	NO 접점 (A접점)	`┤ ├`	평상시 열림 접점
F3	NC 접점 (B접점)	`┤/├`	평상시 닫힘 접점
F5	직선	`──`	신호를 그대로 통과시키는 가로선
F6	수직	`|`	셀 우측에 세로 연결선을 토글
F7	출력 코일 SET	`-(S)-`	조건이 켜지면 래치(유지)
F8	출력 코일 RESET	`-(R)-`	조건이 켜지면 래치 해제
F9	출력 코일	`-( )-`	매 스캔 조건 그대로 출력
F10	신호 반전	`-/-`	앞 조건의 결과를 반전(`!`)
F11	상승엣지	`┤↑├`	상승 엣지에서 한 스캔만 켜짐
F12	하강엣지	`┤↓├`	하강 엣지에서 한 스캔만 켜짐
`\`	박스함수	`[TON|T0|100]`	타이머(TON)·카운터(CTU) 등 — 파라미터마다 칸 하나

1. 그리드의 1행 1열 빈 셀을 클릭해 선택합니다.

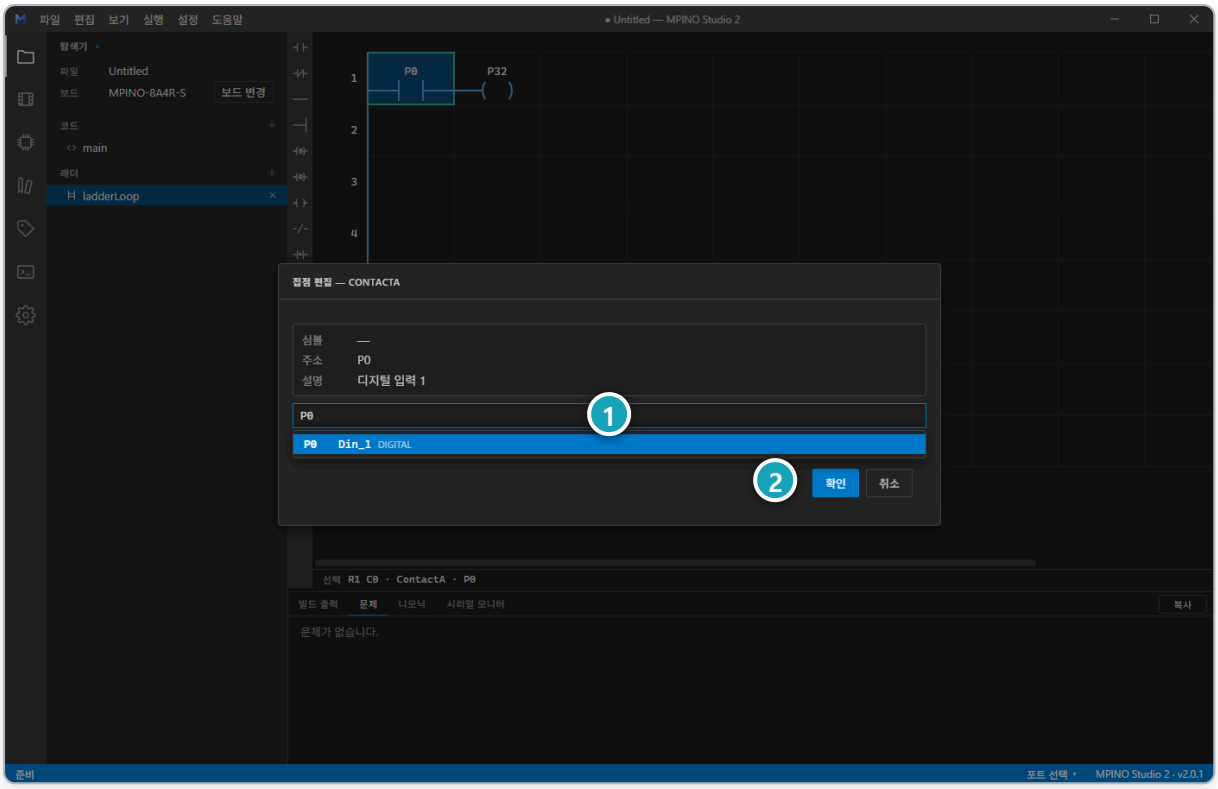
2. 툴바에서 **NO 접점** 도구(F2)를 클릭합니다 — 선택한 셀이 곧바로 NO 접점 기호로 바뀝니다.
3. 같은 1행의 다음 칸(1행 2열)을 클릭해 선택합니다.
4. 툴바에서 **출력 코일** 도구(F9)를 클릭합니다 — 이 셀이 출력 코일 기호로 바뀝니다.
5. 방금 놓은 **접점 셀**(1행 1열)을 더블클릭합니다. **접점 편집** 모달이 열립니다.
6. 입력란(안내 문구: "주소(P0), 심볼명, 또는 비교식(D0 > 100)")에 "P0"를 입력하고 **확인**을 클릭합니다.
7. **코일 셀**(1행 2열)을 더블클릭합니다. 코일도 핀을 참조하는 셀이라 접점과 똑같은 모달이 열립니다 — 모달 제목은 여전히 "접점 편집"으로 표시되지만 코일 값을 편집하는 화면이 맞습니다.
8. 입력란에 "P32"를 입력하고 **확인**을 클릭합니다.



< 그림 1-6 래더 툴바와 완성된 회로 — 1행에 접점(P0)과 출력 코일(P32)이 놓인 상태 >

① **NO 접점 도구** — 단축키 F2
② **출력 코일 도구** — 단축키 F9
③ **P0 접점 셀** — 회로의 입력 조건을 검사하는 자리
④ **P32 출력 코일 셀** — 조건 결과를 릴레이 출력으로 내보내는 자리

셀 편집 모달은 Enter 키나 더블클릭 어느 쪽으로 열어도 동일하게 동작합니다. 모달 제목은 "접점 편집" 뒤에 셀 종류(내부 자료형 이름, 예: ContactA)가 붙어 표시됩니다. 입력란에 새 주소나 심볼을 입력한 뒤 **확인** 버튼을 누르면 값이 커밋되고 모달이 닫힙니다. 아직 등록되지 않은 주소를 입력해도 저장은 되며, 이후 심볼이나 핀 정의가 채워지면 자동으로 해결됩니다.



< 그림 1-7 접점 편집 모달 — 주소 입력란에 P0를 입력하는 화면 >

① **주소 입력란** — 셀이 참조할 대상을 입력하는 칸. 세 가지 형식을 모두 받음

② **확인 버튼** — 입력값을 셀에 확정

이렇게 하면 1행에 `ContactA(P0)`에서 `OutCoil(P32)`로 이어지는 회로가 완성됩니다. "P0가 ON이면 P32가 ON"이 되는, 래더에서 가장 단순한 형태의 예제입니다. F6(수직)이나 박스함수 같은 나머지 도구는 여러 조건을 분기하거나 타이머·카운터를 쓰는 복잡한 회로에서 필요해지므로, 이번 최소 예제에서는 다루지 않습니다.

1-6 래더는 어떻게 아두이노 C 코드가 되나

방금 그린 회로가 실제로 보드에서 어떻게 실행되는지 짚고 넘어가겠습니다. MPINO Studio 2는 `mpinoc`이라는 별도 변환 엔진을 통해 래더 그리드를 rung(가로줄) 단위로 분해해 표현식 트리로 만들고, 이를 `ladder_generated.h`라는 헤더 파일 안의 함수 쌍으로 코드 생성합니다. 래더 탭 이름이 "ladderLoop"이므로 여기서 생성되는 함수는 `ladderLoopSetup()`과 `ladderLoop()`입니다.

트랜스파일러는 사용자가 편집하는 `.ino` 코드를 대신 고쳐 쓰지 않습니다. 기본 프로젝트의 **main** 탭을 열어 보면, 아래처럼 `loop()` 안에서 `ladderLoop()`를 호출하는 최소 스케치가 이미 준비되어 있어 별도 설정 없이 첫 빌드부터 동작합니다(파일 맨 위의 자동 생성 헤더 `#include` 줄은 편의상 생략했습니다).

```
void setup() {
}
```

```
void loop() {
    ladderLoop();
}
```

여기서 `setup()`이 비어 있는 점에 주목하세요. 래더의 초기화 함수 `ladderLoopSetup()`은 사용자가 직접 적지 않습니다. 대신 빌드할 때 트랜스파일러가 컴파일 대상 스케치의 `setup()` 안으로 자동으로 넣어 줍니다(이 주입은 빌드 산출물에만 일어나고, 화면에 보이는 원본 코드나 `.mp2` 파일은 손대지 않습니다). 반대로 `loop()` 안의 `ladderLoop()` 호출은 호출 순서가 사용자의 의도이므로 자동 주입 대상이 아니라 사용자가 직접 책임집니다.

> `setup()` 안에 `ladderLoopSetup()`을 손으로 또 적을 필요는 없습니다. 트랜스파일러가 중복을 걸러 주지 않기 때문에, 직접 적으면 초기화 함수가 두 번 호출됩니다.

전체 흐름은 다음과 같이 이어집니다.

```
.mp2 프로젝트
  → mpinoc 트랜스파일
      → main.ino
      + ladder_generated.h
      + mpino_runtime.h/.cpp
      + board_pins.h
  → arduino-cli compile → .hex
  → arduino-cli upload → 보드에서 실행
```

`mpinoc`을 GUI와 분리된 독립 실행 모듈 겸 CLI로 만든 이유는, 원본 IDE가 트랜스파일러를 GUI 안에 가둬 놓아 GUI 없이는 자동화나 CI 검증이 불가능했던 함정을 피하기 위해서입니다. 실제로 `mpinoc compile project.mp2 -o build/` 형태로 앱을 켜지 않고도 커맨드라인에서 곧장 빌드할 수 있습니다.

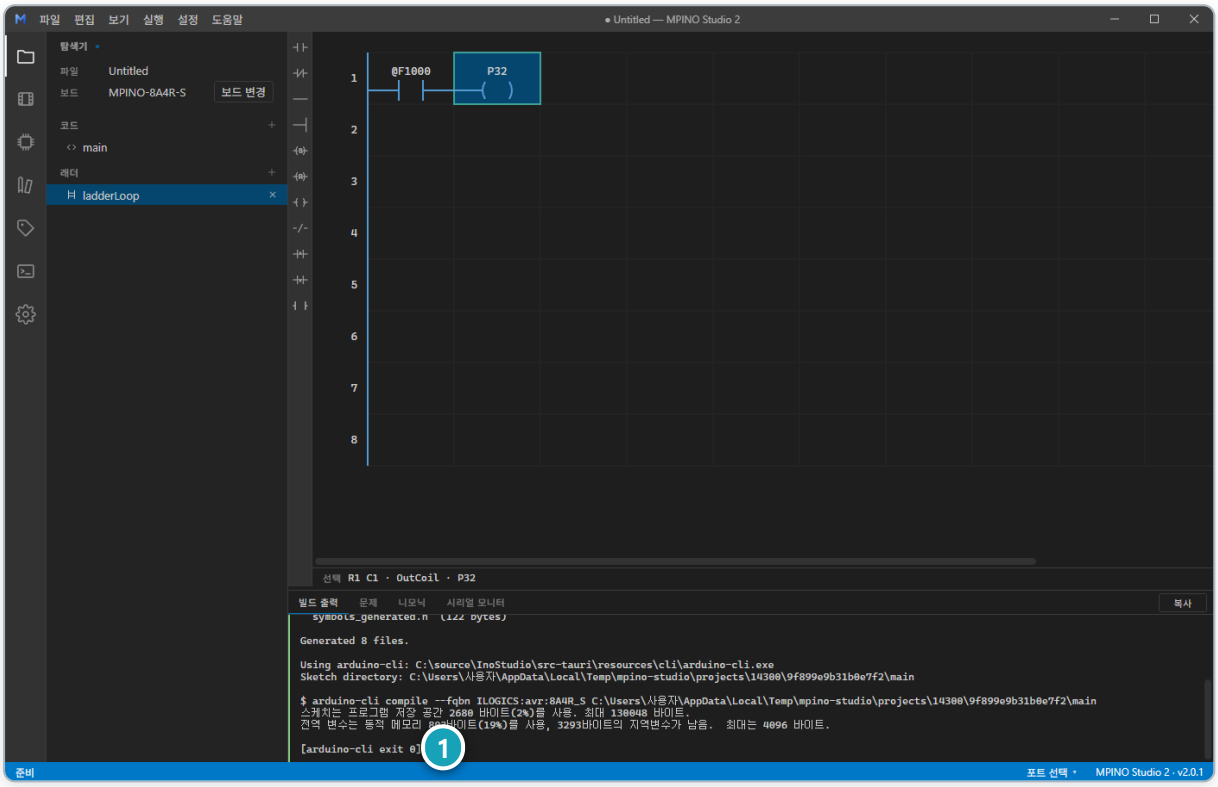
1-7 빌드하기

회로를 다 그렸다면 컴파일해 봅니다. **실행 > 빌드** 메뉴 항목 또는 단축키 **Ctrl+R**을 사용합니다.

> Ctrl+R은 웹 브라우저의 "새로고침" 단축키와 같지만, MPINO Studio 2 안에서는 이 조합을 가로채 **빌드**(컴파일) 전용으로만 사용합니다. 화면이 새로고침되지 않을까 걱정할 필요는 없습니다.

1. **실행** 메뉴에서 **빌드**를 클릭하거나 `Ctrl+R`을 누릅니다.

2. 하단 패널이 자동으로 **빌드 출력** 탭으로 전환되고, "빌드 중..." 로그가 표시됩니다.
3. arduino-cli가 트랜스파일된 스케치를 실제로 컴파일하는 과정이 그대로 출력됩니다.
4. 마지막 줄에 `[arduino-cli exit 0]`이 보이면 빌드 성공입니다.



< 그림 1-8 빌드 출력 패널 — 컴파일 성공 시 exit 0으로 종료 >

① **빌드 성공(exit 0)**

빌드 패널은 진행 상태를 대기·빌드 중·성공·실패 네 가지로 구분해 보여줍니다. 실패하면 로그 아래쪽에 한 줄짜리 요약 오류 메시지가 함께 표시되므로, 어디서 막혔는지 로그 전체를 훑을 필요 없이 바로 확인할 수 있습니다. 이때 앱은 "보드 코어가 설치되지 않았다"와 "보드 정의의 FQBN·메뉴 설정이 잘못됐다"를 구분해 안내합니다 — 후자는 "hardware 패키지 설치 문제가 아닙니다."라는 문장과 함께 어느 항목을 봐야 하는지 짚어 줍니다.

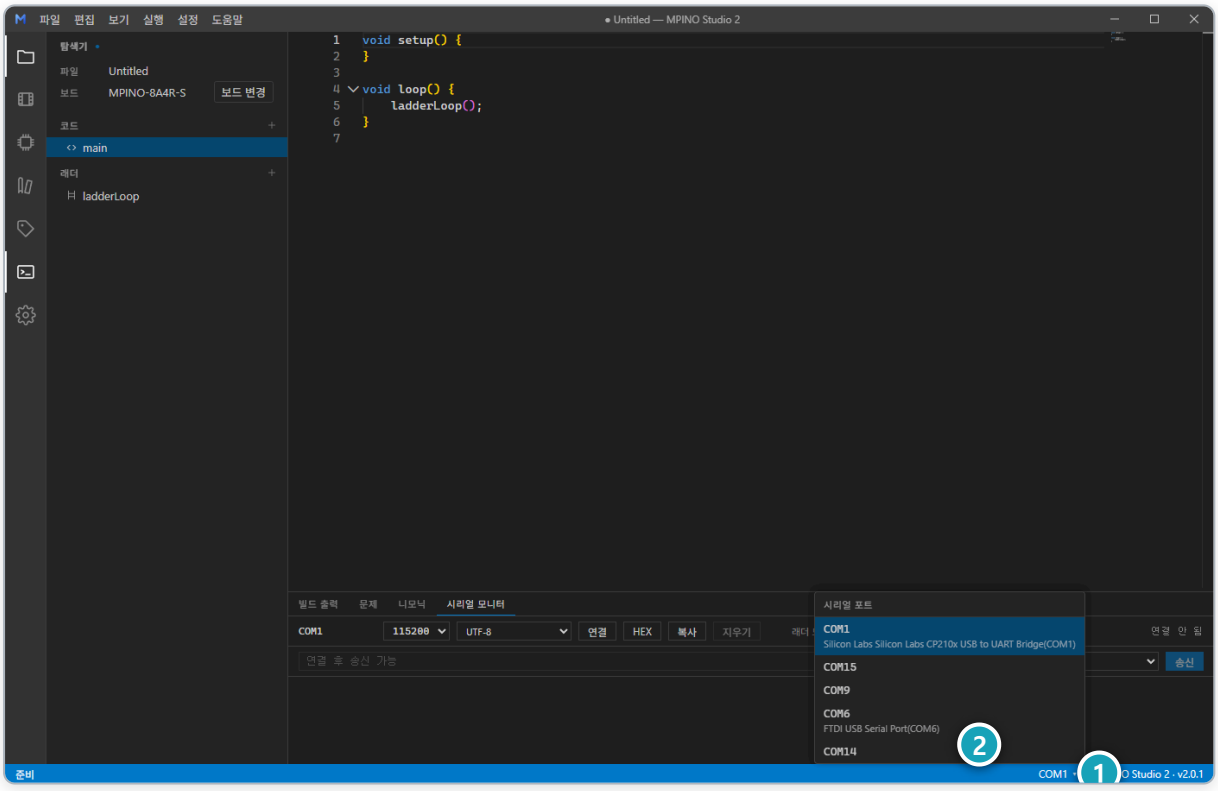
> **첫 빌드에 인터넷이 필요하지 않습니다.** 컴파일에 쓰는 arduino-cli와 ILOGICS 보드 코어·툴체인이 모두 설치본에 함께 들어 있어, 인터넷이 없는 현장 PC에서도 설치 직후 바로 빌드할 수 있습니다.

1-8 포트 선택과 업로드

빌드가 끝났다면 이제 실제 보드에 올릴 차례입니다. 포트를 먼저 고르고, 그다음 업로드를 실행하는 순서를 지켜야 합니다.

> 업로드(Ctrl+U)는 포트가 선택되어 있지 않으면 안내와 함께 즉시 중단됩니다. 반드시 상태바에서 포트를 먼저 선택하세요.

1. 상태바 우측의 포트 버튼(미선택 시 "포트 선택", 선택 후에는 포트 이름 그대로 표시)을 클릭합니다.
2. 드롭다운이 열리는 순간 포트 목록이 자동으로 새로고침됩니다(별도 새로고침 버튼은 없습니다).
3. 목록에서 사용할 포트(예: COM3)를 클릭해 선택합니다.
4. **실행** 메뉴에서 **업로드**를 클릭하거나 `Ctrl+U`를 누릅니다.
5. 하단 패널의 **빌드 출력** 탭에 빌드 로그 아래로 `--- 업로드 ---` 구분선과 함께 업로드 로그가 이어서 표시됩니다.



< 그림 1-9 상태바 포트 드롭다운 — 시리얼 포트를 선택하는 유일한 경로 >

① **상태바 포트 버튼** — 포트 선택 드롭다운을 여는 버튼. 버튼 라벨 자체가 현재 선택 상태 표시를 겸함

② **포트 목록 항목** — 여기서 고른 포트는 업로드와 시리얼 모니터 연결에 함께 사용됨

포트 선택은 상태바 드롭다운 하나가 유일한 경로입니다. 업로드가 진행되는 동안에는 패널 우측에 **업로드 취소** 버튼이 나타나고, 취소를 요청하면 잠시 "취소 중..."으로 바뀝니다.

업로드는 한 번에 보드에 쓰는 것이 아니라 **변환 → 컴파일 → 보드에 쓰기** 순서로 진행됩니다. 하단 패널 탭 줄 오른쪽에 지금 단계가 "컴파일 중 (2/3)"처럼 표시되고, 탭 줄 아래의 얇은 막대가 진행을 보여 줍니다. 변환·컴파일 동안에는 막대가 좌우로 움직이기만 합니다. 컴파일이 끝나면 먼저 "보드에 연결 중"이 표시되고(케이블이 빠져 있으면 여기서 약 10초 머문 뒤 실패합니다), 실제 쓰기가 시작되면 "보드에 쓰는 중"으로 바뀌며 막대가 쓰기 진행률(%)만큼 채워집니다. 끝나면 **완료·실패·취소됨** 중 하나가 남습니다.

업로드를 시작할 때, 래더 모니터링이 쓰기로 한 포트를 다른 곳에서도 쓰고 있으면 곧바로 업로드하지 않고 **"Serial1 중복 사용"** 처럼 포트 이름이 들어간 확인 창이 먼저 뜹니다. 충돌하는 상대에 따라 본문이 세 가지로 갈립니다.

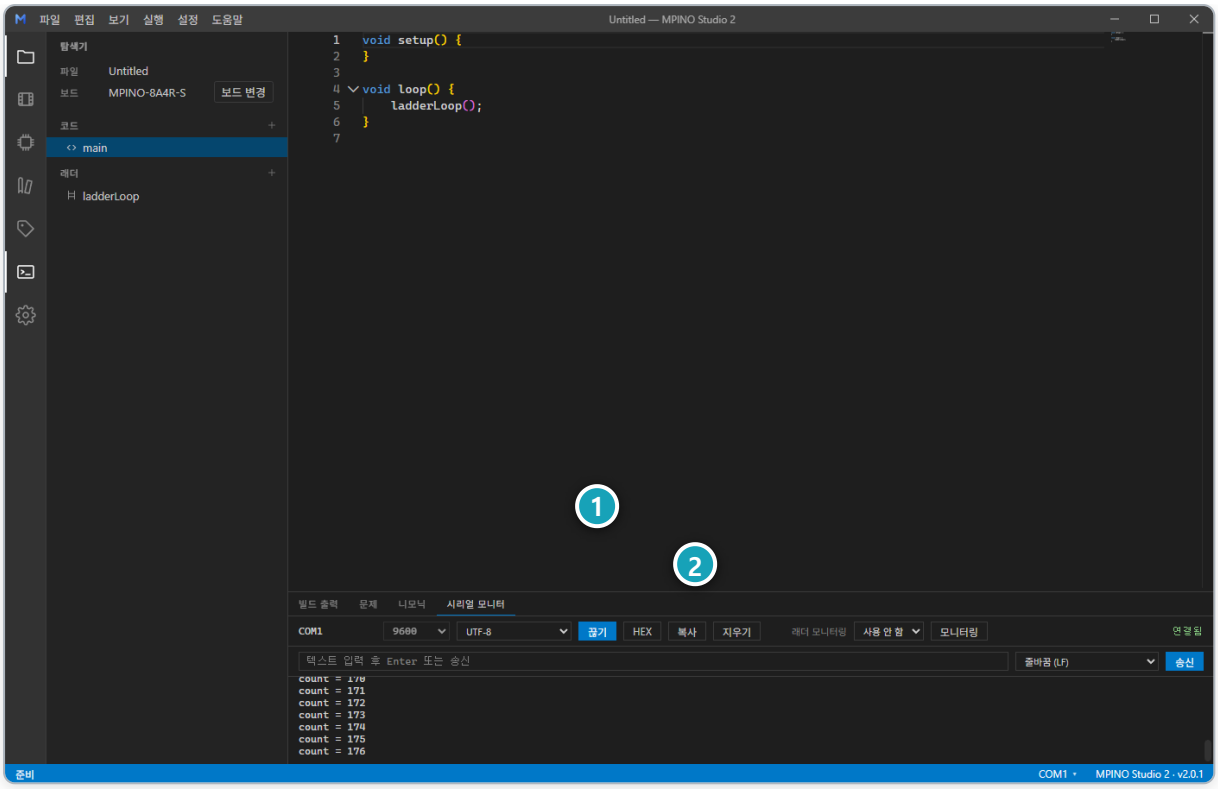
충돌 상대	안내 내용
통신 설정(실행 › 통신)의 채널	그 채널의 프로토콜을 **없음**으로 바꾸거나 모니터링 포트를 다른 것으로 바꾸라고 안내
아두이노 코드의 통신 초기화 호출	코드의 그 함수 호출에 다른 포트를 넘기라고 안내(함수 이름을 그대로 보여 줍니다)
아두이노 코드가 그 포트를 직접 사용	코드에서 다른 포트를 쓰거나 모니터링 포트를 바꾸라고 안내

세 경우 모두 **막는 것이 아니라 경고**입니다. [업로드]를 누르면 그대로 진행되고, [취소]를 누르면 업로드가 실행되지 않습니다.

1-9 동작 확인: 시리얼 모니터와 실시간 모니터링

마지막으로 보드가 실제로 어떻게 동작하는지 두 가지 방법으로 확인합니다. 하나는 시리얼 포트로 오가는 원시(raw) 문자열을 보는 **시리얼 모니터**이고, 다른 하나는 래더 회로의 통전 상태를 화면에서 직접 보는 **실시간 모니터링**입니다. 둘은 서로 다른 기능이라 독립적으로 켜고 끌 수 있습니다.

1. 액티비티바의 **시리얼 모니터** 아이콘을 클릭합니다 — 사이드바가 아니라 하단 패널의 **시리얼 모니터** 탭으로 바로 이동합니다.
2. 헤더에서 baud(기본 115200, 총 16종 중 선택)와 인코딩(UTF-8 등 5종)을 확인합니다.
3. **연결** 버튼을 클릭합니다. 연결되면 버튼 라벨이 "끊기"로 바뀌고 강조 색으로 표시됩니다.
4. 수신된 문자열이 로그 영역에 그대로 쌓입니다. ASCII/HEX 버튼으로 표시 형식을 바꿀 수 있습니다.



< 그림 1-10 시리얼 모니터 — 연결 후 baud/인코딩과 함께 원시 문자열을 확인 >

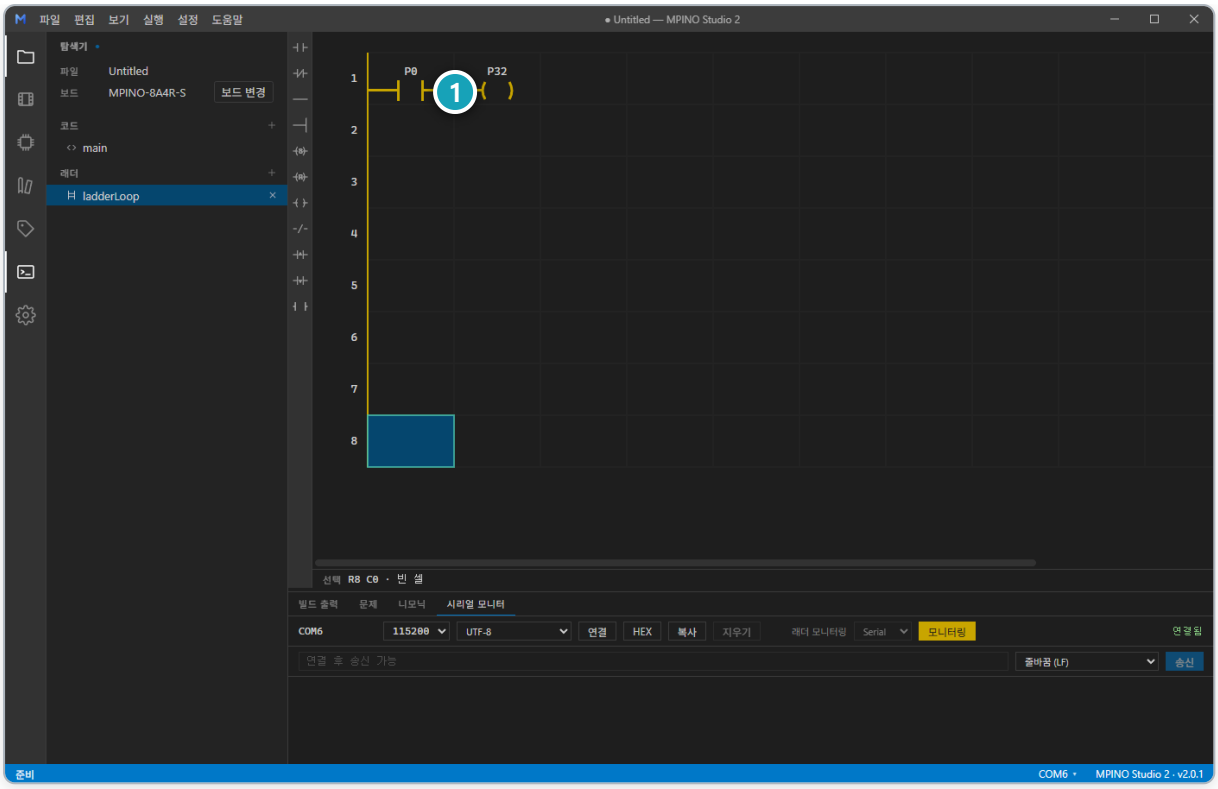
① **연결 버튼** — 시리얼 연결을 켜고 끄는 토글

② **모니터링 버튼** — 실시간 래더 통전 표시를 켜고 끄는 토글. 옆의 연결 버튼과는 독립 동작

> **모니터링** 버튼은 실시간 래더 모니터링(통전 표시)을 켜는 별도 스위치입니다. 처음 누르면 안내 모달이 뜨는데, 정확한 문구는 다음과 같습니다: "모니터링은 환경설정 → 래더 설정 → 모니터링 사용 포트에서 통신포트를 설정해야 합니다." 여기서 말하는 곳은 '화면 둘러보기'(1-2절)에서 본 액티비티바의 **래더설정** 패널(핀맵·메모리 읽기 전용 열람)이 아니라, **설정** (환경설정) 모달 안의 **래더 설정** 탭입니다 — 이름이 비슷하지만 완전히 다른 화면이니 헷갈리지 않도록 주의하세요. 해당 탭의 라디오 옵션은 **사용 안 함 / Serial / Serial1 / Serial2** 중에서 **현재 보드가 실제로 가진 UART만** 남겨 표시됩니다 — 이번 튜토리얼의 MPINO-8A4R(T)-S는 UART가 둘뿐이라 "사용 안 함 / Serial / Serial1" 세 가지만 보입니다. 이 중 하나를 지정하면(사용 안 함이면 꺼진 상태) 다음 빌드/업로드부터 펌웨어가 모니터링 데이터를 실제로 송신합니다.

같은 선택은 시리얼 모니터 헤더에서도 할 수 있습니다. 헤더 오른쪽 **래더 모니터링** 묶음의 **모니터링 채널** 드롭다운이 환경설정의 라디오와 같은 값을 다루므로, 어느 쪽에서 바꾸든 결과는 같습니다(드롭다운에 마우스를 올리면 "펌웨어가 모니터링 데이터를 내보낼 채널. 바꾼 뒤 업로드해야 보드에 반영됩니다."라는 설명이 표시됩니다).

모니터링 포트를 설정했다면, 회로가 반영된 펌웨어를 업로드한 뒤 **모니터링** 버튼을 다시 누릅니다. `Ctrl+M` 이나 래더 그리드 우클릭 메뉴의 **모니터링 ON**으로도 같은 동작을 켤 수 있습니다. 통전 중인 접점과 코일은 강조 색으로 바뀌고 선이 굵어져 눈에 띄게 표시됩니다 — 단, 이 강조 표시가 실제로 나타나려면 사람이 보드의 P0 입력에 물리적으로 신호를 인가해야 합니다.



< 그림 1-11 실시간 모니터링 — P0에 신호가 들어오면 P0 접점과 P32 코일이 통전 색으로 강조된 모습 >

① **통전 표시** — 통전 중인 접점·코일이 강조색과 굵은 선으로 표시됨(P0 입력에 실제 신호가 인가돼야 나타남)

모니터링을 끄고 싶으면 래더 그리드에 포커스가 있는 상태에서 **Escape** 키를 누르면 바로 꺼집니다.

1-10 이 튜토리얼에서 사용한 단축키

단축키	명령
Ctrl+N	새로 만들기
Ctrl+Shift+P	래더 핀맵 설정(핀 켜기)
Ctrl+S	저장
Ctrl+R	빌드
Ctrl+U	업로드
Ctrl+M	모니터링 토글
F2	NO 접점 배치
F9	출력 코일 배치

여기까지가 MPINO Studio 2의 첫 프로젝트, 처음부터 끝까지입니다. 새 프로젝트를 만들고, 보드를 고르고, 접점 하나와 코일 하나로 최소 회로를 그리고, 빌드해서 실제 보드에 올리고, 동작까지 눈으로 확인했습니다. 이후 장에서는 여러 접점을 조합하는 방법, 타이머·카운터 같은 박스함수, 그리고 메모리 영역과 통신 프로토콜 설정을 다룹니다.

2 필독사항

2-1 데이터 메모리

메모리는 래더 회로와 아두이노 C 코드가 값을 읽고 쓰는 저장 공간입니다. MPINO Studio 2는 이 저장 공간을 성격이 다른 6개 영역으로 나눠 관리하며, 주소는 영역을 나타내는 문자 뒤에 10진 인덱스를 붙이는 형태(`P0`, `M10`, `D0`, …)로 표기합니다.

영역	타입	크기	주소 예	용도
P	포트 (비트)	1비트	P0, P32	디지털 입력·출력
M	비트	1비트	M0, M10	내부 비트 릴레이(보조 접점)
D	워드	16비트	D0, D10	범용 데이터(정수)
C	카운터	16비트	C0, C1	카운터 박스함수용(일반 워드로도 사용)
T	타이머	16비트	T0, T1	타이머 박스함수용(일반 워드로도 사용)
R	실수	32비트	R0, R1	부동소수점(IEEE754 단정밀도)

P는 보드의 실제 핀에 연결된 디지털 입력·출력 영역입니다. **M**은 보드 핀과 무관하게 회로 내부에서만 쓰는 보조 비트 릴레이로, 접점 값을 잠시 저장하거나 여러 rung을 이어줄 때 씁니다. **D**는 정수 값을 담는 범용 워드 영역입니다. **C**와 **T**는 각각 카운터·타이머 박스함수 전용으로 예약돼 있지만, 박스함수에 쓰지 않는 나머지 주소는 D처럼 일반 워드로도 쓸 수 있습니다. **R**은 소수점이 있는 값을 담는 32비트 실수(IEEE754 단정밀도) 영역입니다.

> **P의 입력/출력 구분은 고정이 아니라 보드 핀 방향(IN/OUT)이 결정합니다.** 예를 들어 챕터 1에서 쓴 MPINO-8A4R(T)-S는 P0~P7이 입력, P32~P35가 출력이지만, 이는 그 보드의 정의(`boards/*.toml`)가 그렇게 배치했을 뿐 "P0~P31은 항상 입력"처럼 앱이 강제하는 규칙이 아닙니다. 다른 보드를 고르면 같은 P0라도 입력이 아닌 출력일 수 있습니다.

> **R(실수)는 부동소수점 오차가 있습니다.** 예를 들어 7.1을 저장해도 실제로는 7.10001처럼 미세하게 어긋날 수 있으므로, R 값을 정확히 비교해야 하는 조건식에서는 이 오차 여지를 감안하세요.

2-1-1 표현 범위

메모리는 영역별 native 폭 외에도, 담을 수 있는 값의 범위에 따라 다음 네 단위로 구분됩니다.

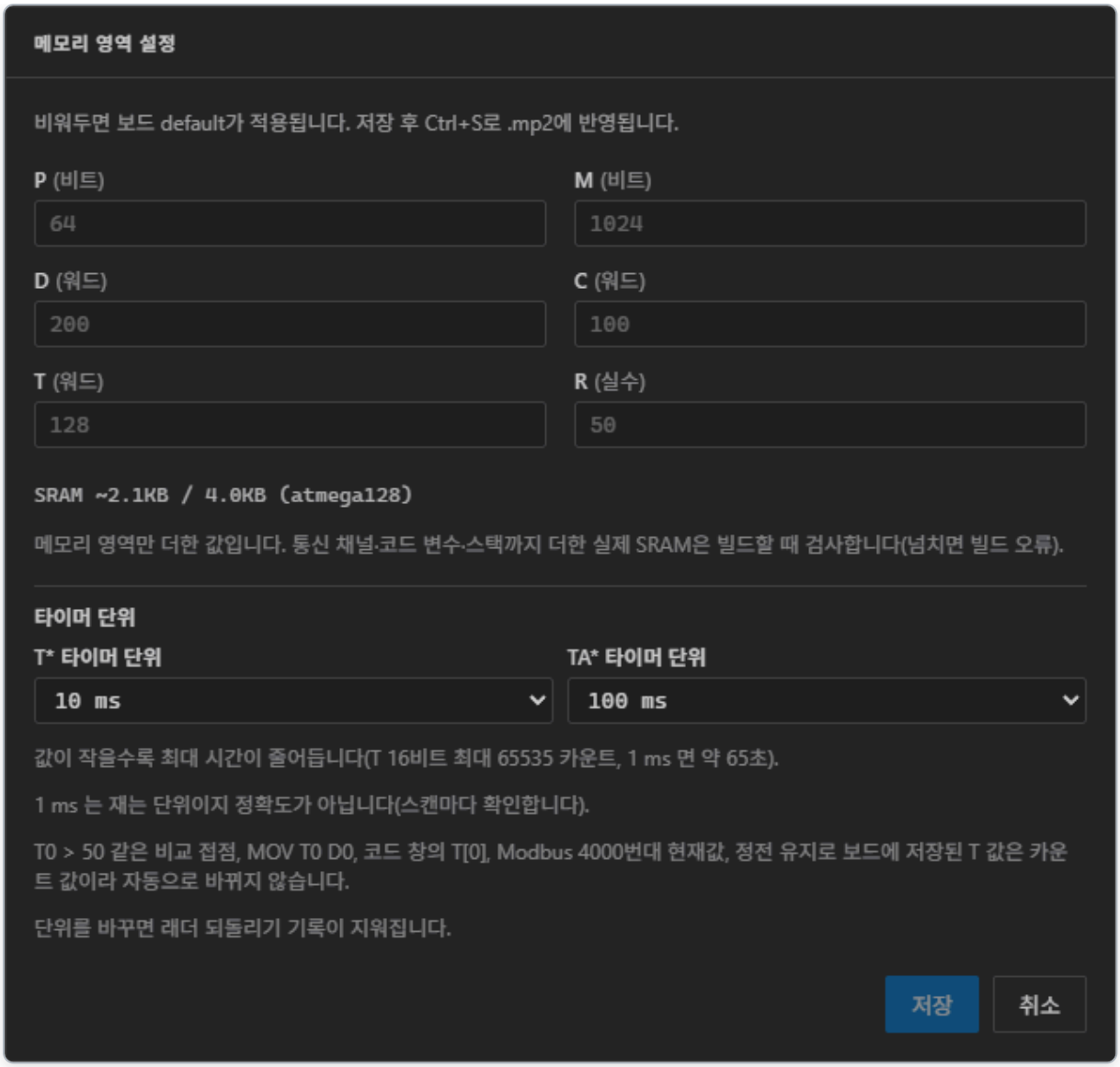
단위	표현 범위	구성
BIT	0 또는 1	메모리의 최소 단위
BYTE	−128 ~ +127	8비트
WORD	−32,768 ~ +32,767	2바이트
DWORD	−2,147,483,648 ~ +2,147,483,647	2워드

32767보다 큰 상수는 DWORD 상수입니다. −32,768 ~ +32,767 밖의 숫자는 WORD가 아니라 DWORD 상수로 봅니다. DWORD 상수를 WORD(D·C·T)에 넣으면 **하위 16비트만** 들어가고 「문제」 탭과 빌드 출력에 경고가 뜹니다 — 빌드는 그대로 됩니다.

- `MOV 65535 C20` → C20 = −1(하위 16비트 1111 1111 1111 1111), `MOV 70000 C20` → C20 = 4464. 의도한 값이면 `MOV -1 C20`처럼 WORD 범위로 쓰면 경고가 사라집니다.
- **예외 1 — 명령 자체가 부호 없이 동작하는 박스함수**(`WAND` `WOR` `WXOR` `WNOT` `INV` `SHL` `SHR` `ROL` `ROR` `H2D` `D2H`, 타이머·카운터 설정값)는 0~65535를 WORD로 받습니다. 자유입력형 식도 `&` `|` `^` `~`만으로 되어 있으면(`C20 = C21 & 65535`) 같습니다.
- **예외 2 — 사용을 켠 아날로그·펄스(PWM) 출력 채널로 가는 MOV**는 0~65535를 그대로 받습니다. MPAINO의 아날로그 출력(Y)·펄스 출력(K) 모듈 채널과, MPINO-16A8R8T의 Aout·PWMout 같은 보드 내장 출력이 모두 해당합니다(`MOV 65535 C20`이면 Y 출력에 65535). 계산·비교에는 이 예외가 없습니다.
- WORD 주소를 DWORD 상수와 직접 비교하거나(`C20 > 40000`) 나누면(`C20 / 40000`) 결과가 늘 같아 경고가 뜹니다. 0~65535 값으로 다루려면 `UW(C20) > 40000`처럼 **UW**(6-9)로 감싸세요.

2-1-2 영역 크기와 메모리 설정

6개 영역의 개수(P가 몇 개, M이 몇 개, …)는 보드마다 기본값이 다릅니다. 예를 들어 MPINO-8A4R(T)-S의 기본값은 P 64, M 1024, D 200, C 100, T 128, R 50입니다. 이 기본값을 프로젝트 단위로 조정하려면 **메모리 영역 설정** 모달(**Ctrl+Shift+M**)을 엽니다. 6개 입력란을 빈칸으로 두면 보드 기본값이 그대로 적용되고, 값을 채우면 그 값으로 덮어씁니다. 모달 하단에는 현재 입력값 기준 SRAM 사용량 합계가 실시간으로 표시되며, **합계가 칩의 SRAM 용량을 넘으면 빨간색으로 강조되고 그 상태로는 저장할 수 없습니다.** 저장을 누르면 값이 즉시 반영되고, 이후 **Ctrl+S**로 프로젝트(`.mp2`)에 기록됩니다.



< 그림 2-1 메모리 영역 설정 모달 — 6개 영역의 개수를 조정하고 하단에서 SRAM 사용량을 확인한다 >

같은 모달의 아래쪽 **타이머 단위**에서는 타이머 한 카운트의 시간을 바꿉니다. `T*` 5종(TON·TOFF·TPL·TMON·TMR)은 1ms 또는 10ms(기본 10ms), `TA*` 5종(TAON·TAOFF·TAPL·TAMON·TAMR)은 10ms·100ms·1000ms 중 하나(기본 100ms)이며, `TA*` 단위는 `T*` 단위보다 커야 합니다. 프로젝트마다 저장되고, 자세한 내용은 **타이머 단위 설정**(6-1-6)을 참고하세요.

2-1-3 접두어와 비트 접근

기본(native) 단위 외에, 접두어를 붙여 같은 주소를 다른 폭으로 읽거나 `.n`을 붙여 특정 비트 하나만 지정할 수도 있습니다.

접근	표기 예	사용 가능 영역
기본(native)	P0, M0, D0, C0, T0, R0	전체
B — 바이트 뷰	BP0, BM1	P·M

접근	표기 예	사용 가능 영역
W — 워드 뷰	WP0, WM1	P·M
D — 더블워드 뷰	DM0, DD2, DC31	P·M·D·C·T
.n — 비트 뷰	D0.0, D0.15, WM1.3	P·M 폭뷰, D·C

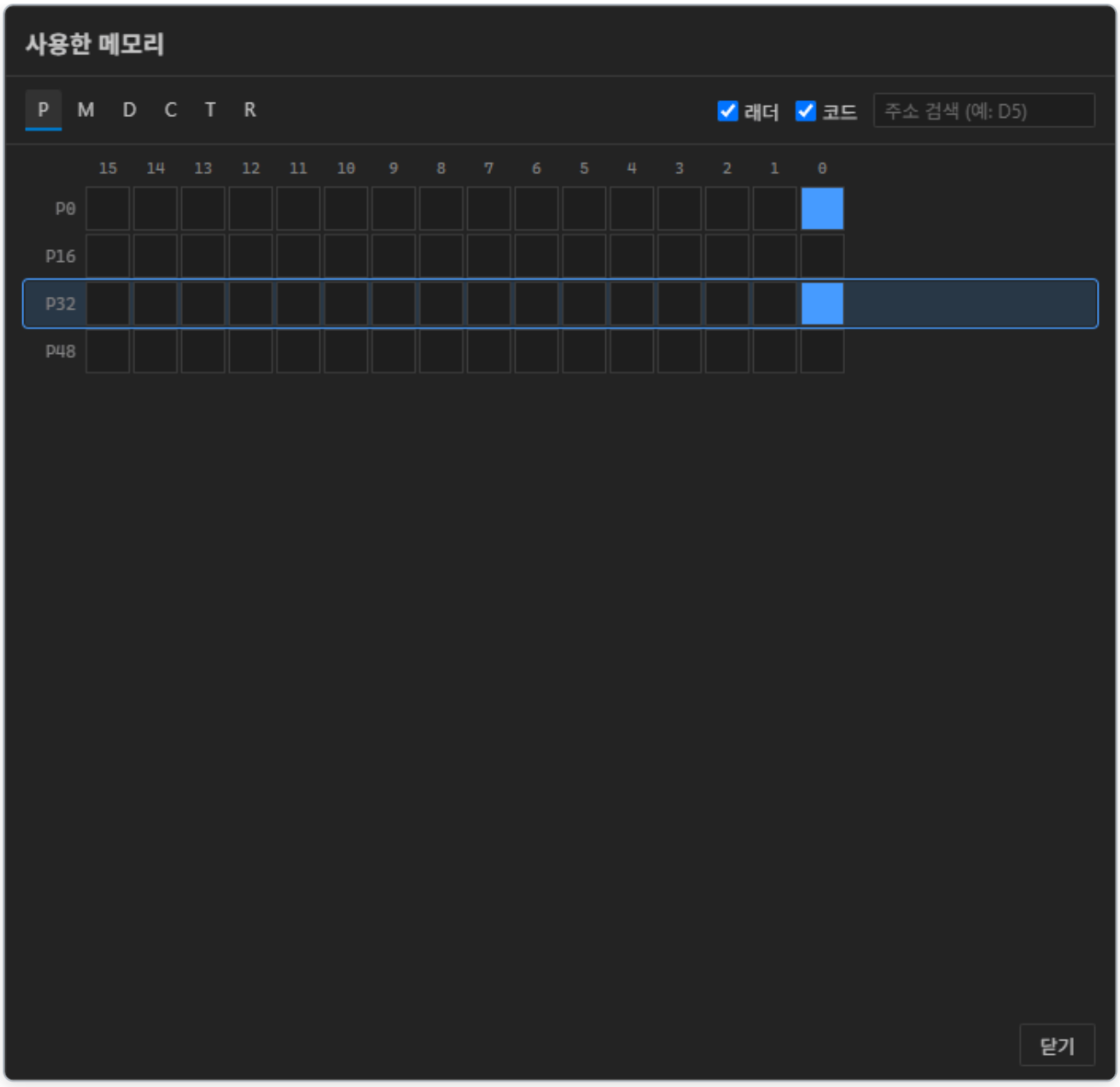
> **바이트(B)·워드(W) 뷰는 P·M 영역에만 쓸 수 있습니다.** 원본 매뉴얼 예시에 나오는 `BD0`(D 영역을 바이트로 보려는 표기)처럼 D·C·T 영역에 B/W 접두어를 붙이면 이 앱에서는 사용할 수 없습니다(D·C·T에서 접두어를 붙이는 경우는 더블워드 `D` 접두어 하나뿐입니다). R(실수)은 폭 접두어와 `.n` 비트 접근 모두 지원하지 않습니다.

2-1-4 사용한 메모리 확인

지금 프로젝트가 어떤 주소를 실제로 쓰고 있는지는 **사용한 메모리** 모달(**Ctrl+Shift+L**)에서 확인합니다. 상단의 P/M/D/C/T/R 탭으로 영역을 바꿔 가며 볼 수 있고, 사용 중인 셀은 격자 위에 색으로 표시됩니다. 사용 중인 셀을 클릭하면 그 주소를 참조하는 래더 셀·코드 줄 목록이 뜨고, 각 항목 옆의 이동 버튼으로 해당 위치까지 바로 이동할 수 있습니다.

이 모달은 다음 세 가지도 함께 지원합니다.

- **주소 검색창**: 모달 우상단 입력란(placeholder "주소 검색 (예: D5)")에 주소를 입력하고 **Enter**를 누르면 그 주소가 속한 영역 탭으로 전환되며 해당 칸이 강조됩니다. 목록을 걸러내는 것이 아니라 찾아가 강조하는 방식이라 격자의 다른 칸도 계속 보입니다. 대소문자는 가리지 않고, 영역 글자(P/M/D/C/T/R) 뒤에 숫자가 오는 형식이어야 하며 형식이 맞지 않으면 아무 동작도 하지 않습니다. 검색어를 지우면 강조가 풀립니다.
- **래더/코드 체크박스**: 격자 위쪽의 **래더·코드** 체크박스로 사용 중 셀을 소스(래더 회로에서 쓰는지·C 코드에서 쓰는지)별로 표시하거나 숨길 수 있습니다. 기본값은 둘 다 켜짐입니다.
- **탭·스크롤 위치 보존**: 선택한 탭과 격자 스크롤 위치는 모달을 닫았다 다시 열어도 그대로 유지됩니다(단 앱을 종료했다 다시 실행하면 초기화됩니다).



< 그림 2-2 사용한 메모리 모달 — 프로젝트에서 실제로 사용 중인 주소를 영역별 격자에서 확인한다 >

2-2 특수 메모리

특수 접점은 사람이 값을 써 넣는 일반 메모리와 달리, 시간이 흐르면 앱이 자동으로 ON/OFF를 갱신해 주는 특수 비트입니다. 주소 대신 `@`로 시작하는 표기를 써서 **접점(입력) 셀이나 엣지 셀 자리에만** 놓을 수 있고, **출력 코일 자리에는 놓을 수 없습니다** — 특수 접점은 어디까지나 조건을 만드는 입력 쪽 표현이지, 값을 내보내는 출력 쪽 표현이 아닙니다.

특수 접점은 다음 2종만 지원합니다.

표기	의미	예
`@<ms>`	one-shot — 지정한 밀리초(ms) 경계를 넘는 그 한 스캔만 ON	`@100` (100ms마다 1회), `@1000` (1초마다 1회)
`@F<ms>`	주기 토글 — ms 동안 ON, ms 동안 OFF를 반복	`@F1000` (1초 ON / 1초 OFF 점멸)

`@100`처럼 `@` 뒤에 밀리초 숫자를 바로 이어 씁니다. one-shot(`@<ms>`)은 경계를 넘은 뒤 다음 스캔에 한 번 켜집니다. 스캔이 지정한 주기보다 길면 그사이 지나간 여러 경계가 한 번으로 합쳐지므로, 스캔당 최대 한 번만 켜진다고 보면 됩니다. 또 `millis()`가 약 1.024ms마다 갱신되므로 `@1`은 초당 최대 약 977회까지 켜집니다.

예제 1 — one-shot으로 값 세기

`@100` 접점을 박스함수 `INC D0`(또는 `D0 = D0 + 1`)와 연결하면, 스캔이 100ms보다 짧으면 100ms마다 정확히 한 스캔만 조건이 켜져 D0가 1씩 증가합니다.

예제 2 — 점멸

`@F1000` 접점을 출력 코일 P32와 연결하면, P32가 1초 동안 켜졌다가 1초 동안 꺼지기를 반복합니다.



< 그림 2-3 특수 접점 예제 — @F1000 접점으로 출력 코일을 1초 간격으로 점멸시키는 회로 >

> **표기 방식이 구 MP STUDIO와 다릅니다.** 이 앱은 괄호 없이 `@100`·`@F100`처럼 쓰고 단위는 ms입니다(구형의 `@(n)` 괄호 표기·10ms 단위·최대 8종 제한이 아닙니다).

"항상 ON"이나 "항상 OFF"인 조건은 **상수 접점**으로 적습니다. NO/NC 접점 셀에 `@ON`·`@OFF` (또는 `TRUE`·`FALSE`)를 입력하면 그 접점이 각각 상시 통전·상시 차단으로 동작합니다. 입력 접점 없이 곧바로 코일로 이어지는 rung도 매 스캔 항상 실행되므로 같은 뜻이 되지만, 상수 접점을 쓰면 "여기는 일부러 항상 켠 자리"라는 의도가 회로에 드러납니다. 자세한 사용법과 제약은 **상수 접점 (@ON / @OFF)**(5-2-3)을 참고하세요.

2-3 단축키

MPINO Studio 2의 단축키 전체를 기능별로 정리합니다.

파일

단축키	명령
Ctrl+O	프로젝트 열기
Ctrl+S	저장
Ctrl+Shift+S	다른 이름으로 저장

단축키	명령
Ctrl+N	새 프로젝트(보드 선택)
Ctrl+P	인쇄(활성 탭이 코드면 코드, 래더면 래더 도면)
Ctrl+Shift+M	메모리 영역 설정
Ctrl+Shift+D	래더 디자인 설정
Ctrl+Shift+P	래더 핀맵 설정
Ctrl+Shift+L	사용한 메모리(메모리 맵)
Ctrl+Shift+E	EEPROM 정전유지 설정

보기

단축키	명령
Ctrl+B	사이드바 토글
Ctrl+J	하단 패널 토글
Ctrl+M	실시간 모니터링 토글
Ctrl+Enter	심볼 창 열기/닫기(텍스트 입력 중에는 동작하지 않음)

빌드·업로드

단축키	명령
Ctrl+R	빌드(컴파일)
Ctrl+U	업로드

래더 이동·선택

단축키	명령
↑ ↓ ← →	셀 선택 이동
Shift+ ↑ ↓ ← →	범위(직사각형) 선택 확장
Delete	선택한 셀 비우기
Ctrl+ ↑ / Ctrl+ ↓	사이드바 활성 항목 위/아래로 이동

래더 도구

단축키	도구
F2	NO 접점
F3	NC 접점
F5	직선
F6	수직
F7	출력 코일 SET
F8	출력 코일 RESET
F9	출력 코일
F10	신호 반전
F11	상승엣지
F12	하강엣지
`\` (한국어 키보드 `₩`)	박스함수(TON·CTU 등)

래더 클립보드·편집

단축키	명령
Ctrl+C / Ctrl+X / Ctrl+V	셀 복사 / 잘라내기 / 붙여넣기
Ctrl+Z	래더 실행 취소
Ctrl+Y	래더 재실행
Insert	셀 삽입
Ctrl+Delete	셀 삭제(뒤 당김)
Alt+Insert / Alt+Delete	줄 추가 / 줄 삭제
Ctrl+F	래더 찾기/바꾸기
Ctrl+Shift+W / Alt+Enter	아두이노에 값 쓰기
Ctrl+1 / Ctrl+0	모니터링 중 선택 셀의 비트(P, M)에 1 / 0 쓰기

> **일부 단축키는 위치에 따라 다르게 동작합니다.** 방향키(↑ ↓ ← →)는 래더 편집 화면에서 선택한 셀을 옮기고, 코드 에디터에서는 텍스트 커서를 옮깁니다. 한편 래더 도구(F2~F12)·Shift+방향키, 그리고 Ctrl+C/X/V/Z/Y/F는 **래더 그리드에 포커스가 있을 때만** 위 표의 동작을 하고, 코드 에디터에 포커스가 있으면 일반 텍스트 편집 단축키로 동작합니다. 또 **Enter 또는 더블클릭**은

> 선택한 래더 셀의 편집 창을 열고(글로벌 단축키가 아니라 셀 조작), **Esc**(래더 그리드 포커스 시)는 실시간 모니터링을 끕니다.

2-4 핀 편집

"핀 편집"이라는 말은 이 앱에서 서로 다른 두 화면을 가리킵니다. 하나는 **보드 핀맵 편집**으로, 보드 하드웨어의 핀 정의 자체를 고치는 화면입니다. 다른 하나는 **래더 셀 편집**으로, 개별 래더 셀이 참조할 값(주소·심볼 등)을 입력하는 화면입니다. 이름은 비슷하지만 고치는 대상이 완전히 다르므로 구분해서 씁니다.

2-4-1 보드 핀맵 편집

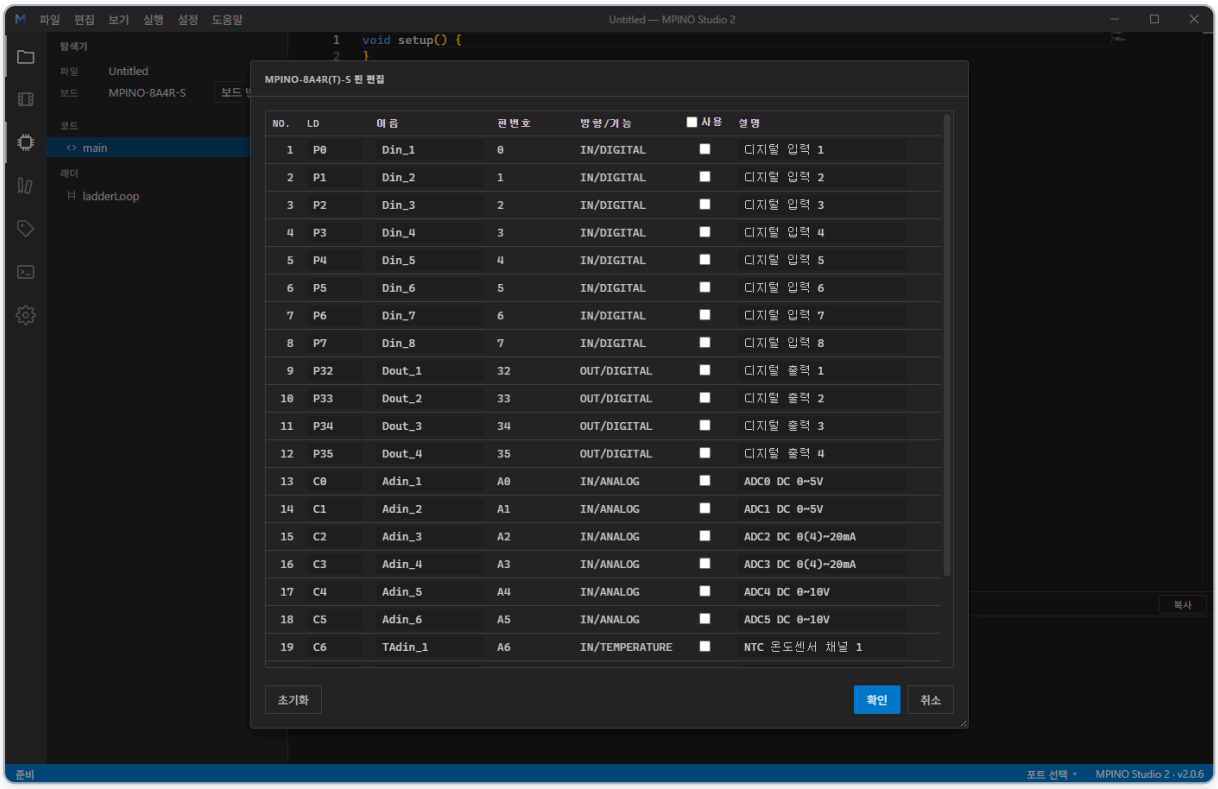
Ctrl+Shift+P(래더 핀맵 설정)를 누르거나, 보드 선택 창에서 **[핀 편집...]** 버튼을 눌러 엽니다(새로 만들기의 보드 선택 창에서 열면 공장 기본값에서 시작해 새 프로젝트에 저장됩니다). 표에서 보드에 정의된 각 핀을 편집합니다. 표의 열은 다음과 같습니다.

열	뜻	편집
No.	표에서의 행 번호	읽기 전용
LD	이 핀의 LD 주소(예 P0). 영역+번호를 자유롭게 편집 가능	편집 가능
이름	핀의 표시 이름	편집 가능
핀번호	실제 아두이노 핀 번호(정수 또는 `A0` 같은 별칭)	읽기 전용
방향/기능	IN/OUT(방향), DIGITAL/ANALOG/TEMPERATURE/PULSE(기능)	읽기 전용
사용	이 핀을 래더에서 사용할지 여부(체크박스)	편집 가능
설명	자유 텍스트	편집 가능

이 앱은 ILOGICS 산업용 보드 전용이라 핀번호와 방향/기능은 하드웨어에 고정된 값이라 **읽기 전용**으로만 표시되며, 행을 새로 추가하거나 기존 행을 삭제하는 기능도 없습니다 — 편집할 수 있는 것은 LD·이름·사용·설명 네 필드뿐입니다.

사용 열은 머리글 자체에도 체크박스가 하나 있습니다. 이 체크박스를 누르면 표의 모든 핀을 한꺼번에 켜거나 끌 수 있고, 일부만 켜져 있는 동안에는 중간 상태로 표시됩니다.

> **핀의 기본값은 "사용 안 함"입니다.** 새로 만든 프로젝트는 모든 핀이 꺼져 있고, 켜진 핀이 적혀 있지 않은 기존 프로젝트를 열어도 마찬가지로 전부 꺼진 상태로 열립니다. 꺼진 핀을 래더에서 참조하면 빌드가 "핀 P0 (Din_1)은(는) 보드 정의에서 비활성화되어 있습니다 (usepin = false). 보드 핀 편집기에서 활성화하거나 다른 핀을 사용하세요."라는 오류로 중단됩니다. 또 래더 셀 편집 창의 **자동완성 후보에도 켜진 핀만** 나타나므로, 후보가 하나도 뜨지 않으면 먼저 이 표에서 쓸 핀을 켜세요.



< 그림 2-4 보드 핀맵 편집 — 핀번호·방향/기능은 읽기 전용, LD·이름·사용·설명만 편집 가능 >

표 하단에는 [초기화]·[확인]·[취소] 세 버튼이 있습니다. [초기화]는 공장 기본 핀 구성으로 되돌리고(LD·이름·사용·설명 등 편집 가능한 값만 되돌리며, 핀 자체는 애초에 삭제할 수 없으므로 되살릴 것도 없습니다), [확인]은 바뀐 값을 실제로 반영하며, [취소]는 변경 없이 창을 닫습니다. [초기화]는 되돌리기 전에 "핀 매핑을 공장 기본값으로 되돌립니다. … 계속하시겠습니까?"라고 한 번 더 확인합니다.

> **이 편집은 보드 정의 파일이 아니라 현재 프로젝트(`.mp2`)에 저장됩니다.** [확인]을 누르면 변경이 열려 있는 프로젝트에 반영되고, 디스크에 기록되는 시점은 **Ctrl+S**로 저장할 때입니다. 보드 정의 파일(`boards/<보드>.toml`)은 읽기 전용 공장 기본값이라 바뀌지 않으므로, **다른 프로젝트에는 아무 영향이 없습니다.** 같은 핀 설정을 다른 프로젝트에서도 쓰려면 그 프로젝트를 열어 똑같이 설정해야 합니다.

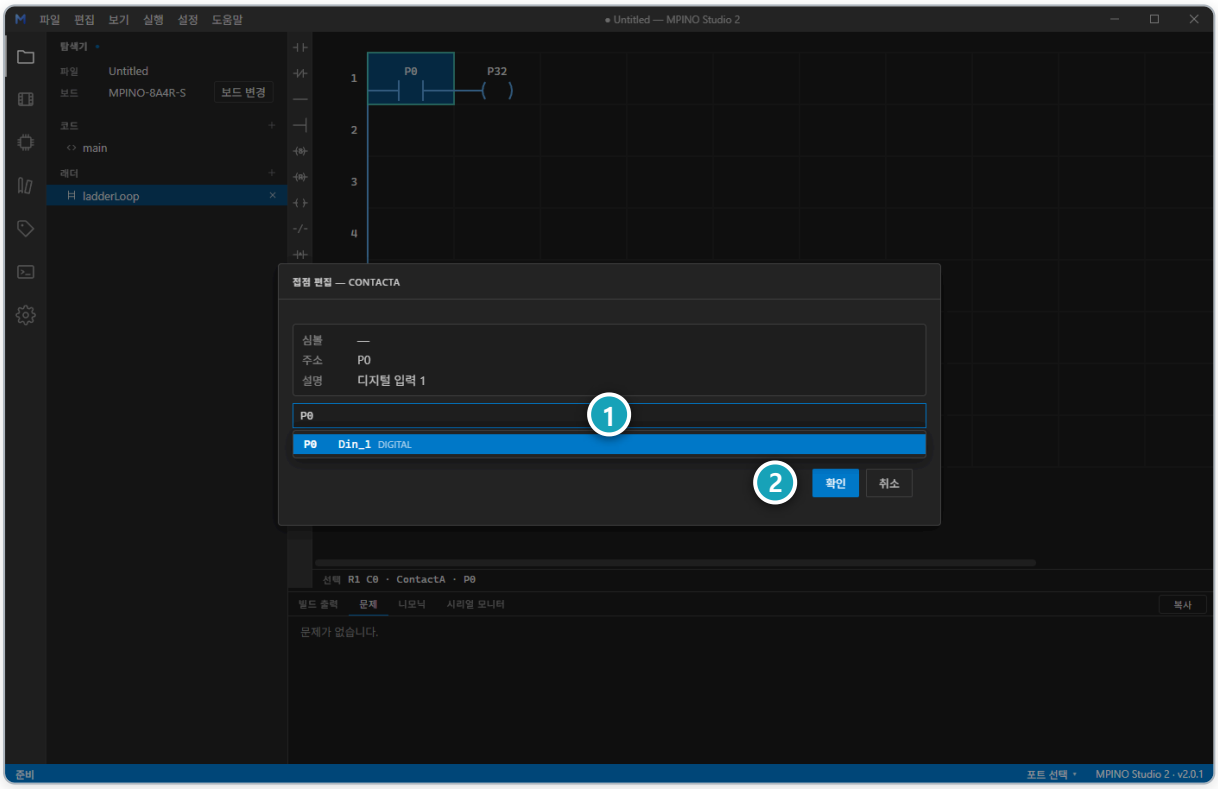
> **MPINO-16A8R8T의 아날로그 출력(C6·C7 = 단자 AO(6)·AO(7))은 16비트입니다.** 값 0~65535가 그대로 PWM 듀티가 되어(스케일 변환 없음) 0 = 0 V, 32768 ≈ 2.5 V, 65535 = 5 V가 나갑니다(출력 범위 0~5 V). C6·C7 중 하나라도 **사용**을 켜면 래더가 시작할 때(`ladderLoopSetup()`) 보드의 TIMER4를 16비트 PWM(최대값 65535, 약 244 Hz)으로 설정합니다. 출력 회로의 필터가 PWM을 직류로 바꾸므로 값을 바꾼 뒤 출력이 자리 잡는 데 약 0.3초가 걸립니다. 출력 회로는 외부 24 V 전원이 있어야 동작합니다(USB 전원만으로는 나오지 않습니다). 펄스 출력(C8~C11)은 그대로 0~255입니다.

> **C 코드에서 TIMER4를 직접 쓰려면 C6·C7의 사용을 끄세요.** C6·C7을 켠 채로 C 코드에 `TCCR4A` · `TCCR4B` · `TCCR4C` · `ICR4` · `TCNT4` · `OCR4A` · `OCR4B` · `OCR4C`, `analogWrite(6, …)` · `analogWrite(7, …)` · `analogWrite(8, …)`, `analogWriteInit` · `IanalogWrite…`가 있으면 래더와 C 코드가 같은 타이머를 서로 덮어쓸 수 있어 「문제」 탭에 경고가 뜹니다(빌드는 됩니다). 고속카운터 TCNT4도 TIMER4를 쓰므로 함께 쓸 수 없습니다. 둘 다 끄면 래더는 TIMER4를 건드리지 않습니다.

2-4-2 래더 셀 편집

래더 그리드에서 접점·엣지·코일 셀을 **더블클릭**하거나 선택 후 **Enter**를 누르면 열립니다. 이 창에서는 그 셀 하나가 참조할 **주소·심볼·특수 접점 값**을 입력합니다(예 `P0`, 심볼명, `@F1000`). 입력하는 동안 일치하는 핀·심볼·타이머/카운터 완료비트 후보가 자동완성 목록으로 뜨며, 목록에서 골라 바로 채울 수 있습니다. 핀 후보에는 **보드 핀맵 편집에서 "사용"으로 켜 둔 핀만** 나타납니다.

보드 정의는 건드리지 않고 **그 셀의 값만** 바꿉니다. 아직 등록되지 않은 주소나 심볼을 입력해도 값은 저장되며, 이후 그 핀·심볼이 정의되면 자동으로 해결(연결)됩니다.



< 그림 2-5 래더 셀 편집 — 셀이 참조할 주소나 심볼을 입력한다(챕터 1에서 본 접점 편집 창과 같은 창) >

① **주소 입력란** — 셀이 참조할 대상을 입력하는 칸. 세 가지 형식을 모두 받음

② **확인 버튼** — 입력값을 셀에 확정

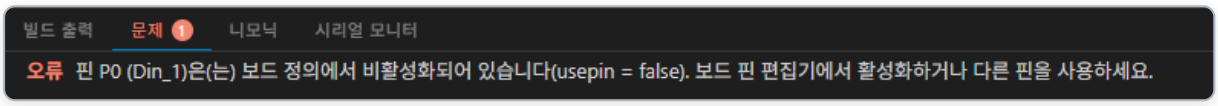
2-4-3 핀 설정 따라 하기

새 프로젝트는 입력·출력 핀이 **모두 "사용 안 함"으로 시작합니다.** 래더에서 P 주소(`P0`, `P32` 등)로 쓰는 보드 핀은 하나도 빠짐없이 **보드 핀맵 편집**(2-4-1)에서 "사용"으로 켜야 빌드됩니다. 쓰지 않는

핀까지 켜 두지 않는 이유는, 켜진 출력 핀은 래더가 매 스캔 값을 써서 아두이노 코드에서 직접 제어하던 출력을 덮어쓰기 때문입니다. 그래서 **쓸 핀만 골라 켜는** 것이 기본 사용법입니다. M·D·T·C·R 같은 내부 메모리는 핀이 아니므로 켤 필요가 없습니다.

여기서는 1장의 `P0 NO 접점 → P32 코일` 회로(MPINO-8A4R(T)-S 보드)를 예로 듭니다. 5장 이후의 예제를 따라 할 때도 그 예제가 쓰는 P 주소를 같은 방법으로 켜면 됩니다.

1단계 — 켜지 않은 핀을 쓰면 나오는 오류. 회로만 그려 두면 핀이 꺼져 있어도 셀은 평소와 똑같이 보입니다. 대신 빌드(Ctrl+R)하면 하단 패널의 **문제** 탭 옆에 빨간 숫자가 붙고, 탭을 열면 다음과 같은 오류가 있습니다.

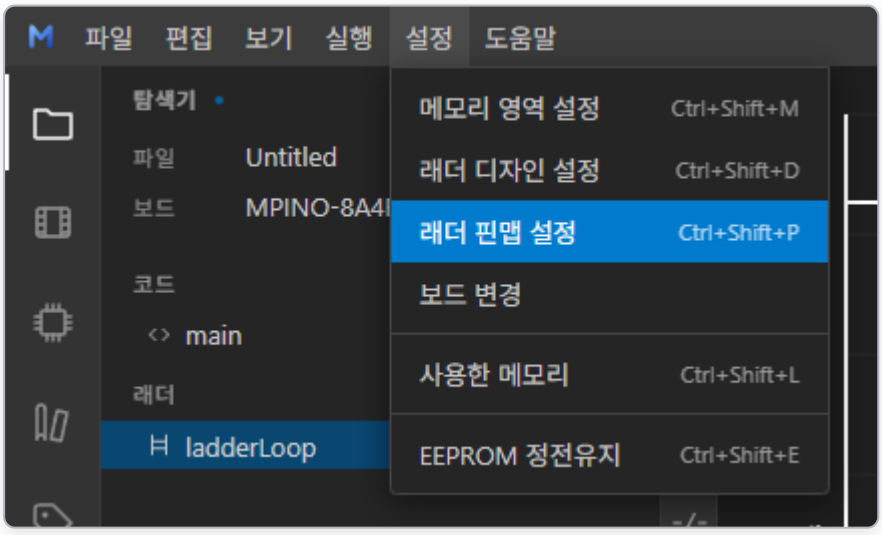


< 그림 2-6 P0를 켜지 않고 회로를 그렸을 때 문제 탭에 나타나는 오류 >

이때 **빌드 출력**에도 같은 문장이 `error:` 줄로 찍히고 컴파일이 중단됩니다. 오류 문장의 "보드 핀 편집기"는 이 절에서 다루는 **래더 핀맵 설정** 창이고, "비활성화되어 있습니다(usepin = false)"는 그 창에서 해당 핀의 **사용** 칸이 체크되지 않았다는 뜻입니다.

> **꺼진 핀이 여러 개면 오류는 한 번에 하나씩 나옵니다.** 위 회로는 P0와 P32가 모두 꺼져 있지만 문제 탭에는 P0 하나만 보이고, P0를 켜고 다시 빌드하면 그제야 P32 오류가 나타납니다. 오류에 나온 핀만 켜지 말고, 회로에서 쓰는 P 주소를 한 번에 모두 켜세요.

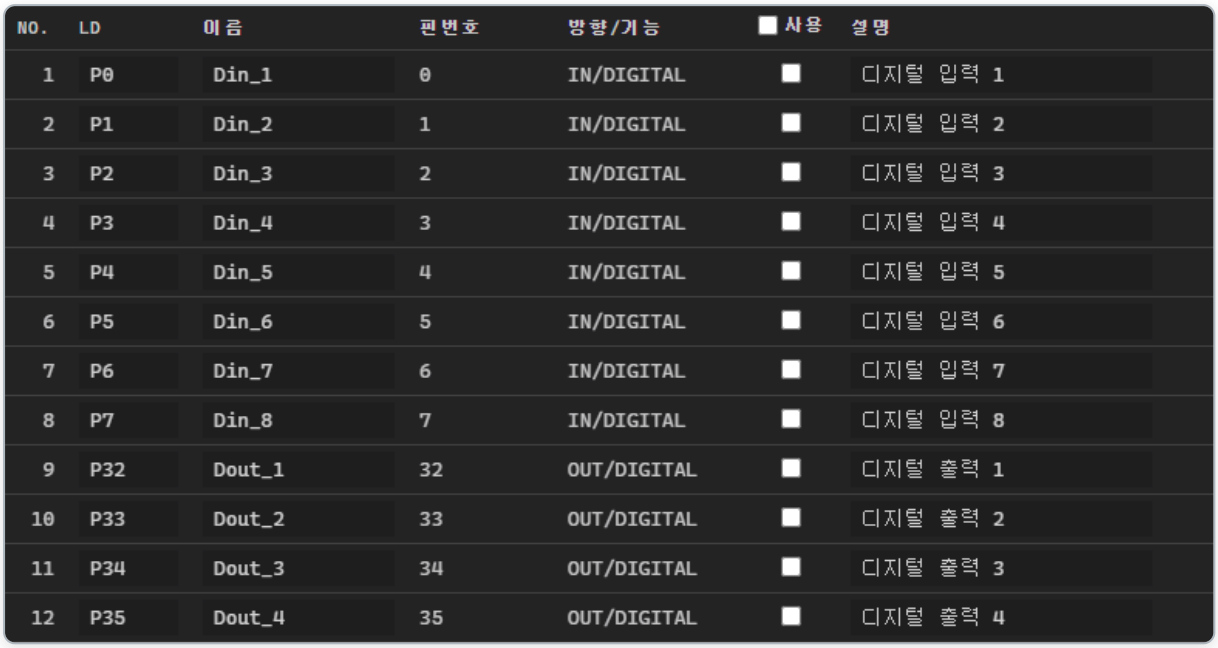
2단계 — 핀맵 편집 창 열기. 메뉴 **설정 › 래더 핀맵 설정**을 클릭합니다. 단축키 **Ctrl+Shift+P**나 왼쪽 활동바의 **래더 핀맵 설정** 아이콘(칩 모양)으로도 같은 창이 열립니다.



< 그림 2-7 설정 메뉴의 래더 핀맵 설정 — 단축키 Ctrl+Shift+P >

창에는 보드의 핀이 한 줄씩 나열되고(전체 모습은 그림 2-4), 처음에는 **사용** 열이 모두 비어 있습니다. 위쪽은 디지털 입력(방향/기능 `IN/DIGITAL`, 이 보드는 P0~P7), 그 아래는 디지털 출력

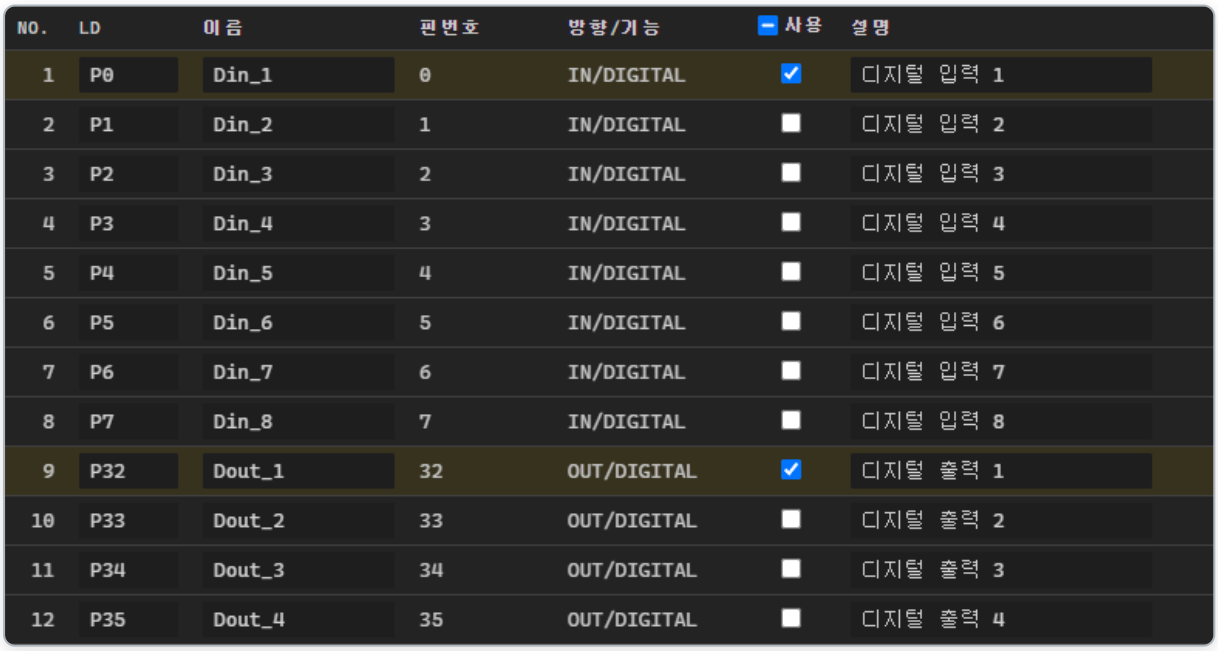
(`OUT/DIGITAL`, P32~P35)이고, 아날로그 입력 등이 뒤따릅니다. MPAINO 보드에 옵션 모듈을 붙였다면 모듈 채널도 같은 표의 아래쪽에 이어서 나오고, 켜는 방법도 같습니다.



NO.	LD	이름	핀번호	방향/기능	사용	설명
1	P0	Din_1	0	IN/DIGITAL	☐	디지털 입력 1
2	P1	Din_2	1	IN/DIGITAL	☐	디지털 입력 2
3	P2	Din_3	2	IN/DIGITAL	☐	디지털 입력 3
4	P3	Din_4	3	IN/DIGITAL	☐	디지털 입력 4
5	P4	Din_5	4	IN/DIGITAL	☐	디지털 입력 5
6	P5	Din_6	5	IN/DIGITAL	☐	디지털 입력 6
7	P6	Din_7	6	IN/DIGITAL	☐	디지털 입력 7
8	P7	Din_8	7	IN/DIGITAL	☐	디지털 입력 8
9	P32	Dout_1	32	OUT/DIGITAL	☐	디지털 출력 1
10	P33	Dout_2	33	OUT/DIGITAL	☐	디지털 출력 2
11	P34	Dout_3	34	OUT/DIGITAL	☐	디지털 출력 3
12	P35	Dout_4	35	OUT/DIGITAL	☐	디지털 출력 4

< 그림 2-8 켜기 전 — 입력 P0와 출력 P32의 사용 칸이 비어 있다 >

3단계 — 쓸 핀의 사용 칸 체크. **LD** 열에서 `P0` 행과 `P32` 행을 찾아 **사용** 열의 체크박스를 클릭합니다. 바꾼 행은 배경색이 달라지고, 일부만 켜진 동안 머리글의 체크박스는 중간 상태(–)로 표시됩니다.



NO.	LD	이름	핀번호	방향/기능	사용	설명
1	P0	Din_1	0	IN/DIGITAL	☑	디지털 입력 1
2	P1	Din_2	1	IN/DIGITAL	☐	디지털 입력 2
3	P2	Din_3	2	IN/DIGITAL	☐	디지털 입력 3
4	P3	Din_4	3	IN/DIGITAL	☐	디지털 입력 4
5	P4	Din_5	4	IN/DIGITAL	☐	디지털 입력 5
6	P5	Din_6	5	IN/DIGITAL	☐	디지털 입력 6
7	P6	Din_7	6	IN/DIGITAL	☐	디지털 입력 7
8	P7	Din_8	7	IN/DIGITAL	☐	디지털 입력 8
9	P32	Dout_1	32	OUT/DIGITAL	☑	디지털 출력 1
10	P33	Dout_2	33	OUT/DIGITAL	☐	디지털 출력 2
11	P34	Dout_3	34	OUT/DIGITAL	☐	디지털 출력 3
12	P35	Dout_4	35	OUT/DIGITAL	☐	디지털 출력 4

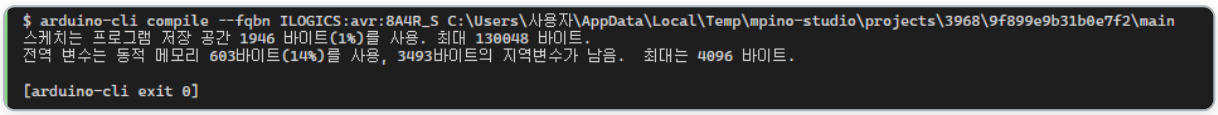
< 그림 2-9 켠 뒤 — P0와 P32의 사용 칸이 체크되고 바뀐 행이 강조된다 >

다 체크했으면 **[확인]** 버튼을 누릅니다. 오른쪽 아래에 "핀맵을 저장했습니다." 알림이 잠깐 뜨고 창이 닫힙니다.

4단계 — 오류가 사라졌는지 확인. [확인]을 누른 뒤 다시 빌드(Ctrl+R)하면 문제 탭의 오류가 사라지고 컴파일이 끝까지 진행되며, 빌드 출력 끝에 프로그램 크기와 `[arduino-cli exit 0]`이 표시됩니다. 문제 탭은 빌드할 때만 갱신되므로, [확인]만 누르고 빌드하지 않으면 이전 오류가 목록에 그대로 남아 있습니다.



< 그림 2-10 P0와 P32를 켜고 다시 빌드한 뒤의 문제 탭 — 오류가 사라졌다 >



< 그림 2-11 핀을 켠 뒤 빌드한 결과 — 컴파일이 끝까지 진행되었다(경로의 사용자 이름은 가림) >

> **[확인]만 누르고 저장하지 않으면 핀 설정은 파일에 남지 않습니다.** "핀맵을 저장했습니다." 알림은 열려 있는 프로젝트에 반영했다는 뜻이고, 디스크에 쓰는 것은 **Ctrl+S**입니다(제목 표시줄의 ● 표시가 저장 전이라는 뜻입니다). 핀 설정은 보드 정의가 아니라 그 프로젝트의 `.mp2` 파일에 저장되므로 프로젝트마다 따로 켜야 합니다. 핀을 끄려면 같은 창에서 체크를 해제하고 [확인]을 누릅니다.

보드에 없는 P 주소. 핀 번호와 개수는 보드마다 다릅니다. 예를 들어 MPINO-8A4R(T)-S의 디지털 출력은 P32~P35 넷뿐이라 `P36`은 이 표에 없고, 켤 수도 없습니다. 다른 보드로 만든 예제를 따라 할 때는 표의 **LD** 열에서 내 보드에 있는 주소를 확인해 바꿔 쓰세요.

3 메뉴 툴바

MPINO Studio 2의 메인 메뉴는 상단 메뉴바의 6개 메뉴(파일, 편집, 보기, 실행, 설정, 도움말)와 각 메뉴 내의 항목들로 구성돼 있습니다. 이 장에서는 각 메뉴의 기능과 사용 방법을 설명합니다.

3-1 파일

프로젝트를 새로 만들거나 열고 저장하며, 코드·래더를 인쇄하고, 앱 환경설정에 접근하거나 프로그램을 종료하는 7개 항목으로 구성됩니다.

항목	단축키	설명
새로 만들기	Ctrl+N	보드를 새로 선택해 프로젝트를 새로 만듭니다
열기	Ctrl+O	기존 `.mp2` 프로젝트 파일을 불러옵니다
저장	Ctrl+S	현재 프로젝트를 저장합니다
다른 이름으로 저장	Ctrl+Shift+S	저장 위치·파일명을 다시 지정해 저장합니다
인쇄	Ctrl+P	활성 탭(코드 또는 래더)의 내용을 인쇄합니다
환경설정	—	환경설정 모달을 "일반" 탭으로 엽니다(자세한 내용은 아래 환경설정 절 참고)
끝내기	—	프로그램 창을 닫아 앱을 종료합니다

새로 만들기(Ctrl+N) 는 편집 중이던 프로젝트에 저장하지 않은 변경 사항이 있으면 먼저 버릴지 확인하는 창이 뜨고, 이어서 보드 선택 창이 열립니다. 전체 절차는 **새 프로젝트 만들기와 보드 선택**(1-4)를 참고하세요.

열기(Ctrl+O) 도 저장하지 않은 변경 사항이 있으면 새로 만들기와 똑같이 먼저 확인 창이 뜹니다. 이어서 OS 파일 열기 대화상자가 열리고, `.mp2` 확장자로 필터링된 목록에서 프로젝트 파일을 선택합니다. 메뉴를 거치지 않고 **탐색기에서 `.mp2` 파일을 더블클릭**해도 됩니다 — 앱이 설치될 때 `.mp2` 확장자가 연결되므로, 더블클릭하면 앱이 실행되면서 그 프로젝트가 곧바로 열립니다.

저장(Ctrl+S) 은 프로젝트가 이미 한 번 저장된 경로를 가지고 있으면 그 경로에 곧바로 덮어씁니다. 새로 만들기 직후처럼 아직 한 번도 저장하지 않은 프로젝트라면 자동으로 "다른 이름으로 저장" 절차로 넘어갑니다.

다른 이름으로 저장(Ctrl+Shift+S) 은 실행할 때마다 항상 저장 위치·파일명을 다시 묻습니다. 이때 새로 정해진 파일명이 **main 코드 탭의 이름과 자동으로 동기화**됩니다 — main 탭 이름은 항상 프로젝트 파일명과 같아야 하기 때문입니다 (다른 코드 탭에 이미 같은 이름이 있으면 저장이 거부됩니다). 파일명을 확장자 없이 적거나 파일 형식을 "All files"로 바꿔 저장해도 이름 끝에 `.mp2`가 붙어 저장됩니다 (`foo` → `foo.mp2`, `foo.txt` → `foo.txt.mp2`).

저장에 실패하면 오류 알림이 뜨고, 이 알림은 ✕ 로 닫을 때까지 남습니다. 실패 원인이 쓰기 권한 거부(`os error 5`)라면 알림 아래에 안내가 한 줄 더 붙습니다 — 흔한 원인은 Windows 보안의 '제어된 폴더 액세스'가 문서 폴더 등에 쓰기를 막는 경우이므로, 다른 폴더에 저장하거나 **Windows 보안 > 랜섬웨어 방지**에서 MPINO Studio 2를 허용된 앱에 추가하세요.

인쇄(Ctrl+P) 는 화면에 보이는 모습이 아니라 **현재 활성 탭의 종류**를 기준으로 인쇄 대상을 정합니다. 코드 탭이 활성이면 프로젝트의 모든 코드(.ino) 탭을 main부터 탭 순서대로 모아, 탭마다 새 페이지에서 시작해 인쇄합니다(코드 탭이 하나뿐이면 그 하나만 인쇄됩니다). 래더 탭이 활성이면 프로젝트의 모든 래더 회로를 정적 도면(SVG)으로 인쇄합니다. 인쇄할 코드나 래더가 없으면 (예: 래더가 하나도 없는 프로젝트) 아무 동작도 하지 않습니다.



< 그림 3-1 파일 메뉴 — 새로 만들기부터 끝내기까지 7개 항목이 표시된 드롭다운 >

> **끝내기는 저장 여부를 묻지 않고 곧바로 창을 닫습니다.** 저장하지 않은 변경 사항이 있다면 끝내기 전에 먼저 Ctrl+S(또는 Ctrl+Shift+S)로 저장하세요.

3-1-1 환경설정

환경설정은 파일 메뉴의 **환경설정** 항목(단축키 없음)으로 열며, 이 경로로 열면 항상 **일반** 탭으로 진입합니다. 액티비티바의 ⚙ 아이콘으로도 열 수 있습니다(액티비티바 상세는 이 챕터의 범위 밖입니다). 모달 상단에는 4개 탭(**일반** / **단축키** / **래더 설정** / **래더 디자인**)이 있고, 탭을 클릭하거나 모달에 포커스가 있는 상태에서 ←/→ 방향키로 전환합니다.

일반

항목	내용
사용자 라이브러리 폴더	라이브러리 설치·가져오기·폴더 열기가 사용하는 위치. 현재 경로가 코드 서식으로 표시됩니다
플랫폼 폴더	ILOGICS 보드 코어·툴체인 폴더. 빌드와 자동완성이 이 폴더를 참조합니다

항목	내용
언어	한국어 / English / 日本語 라디오 3종
오류 리포트	자동 전송 여부 체크박스(기본 켜짐)
업로드	업로드 후 검증 체크박스(기본 꺼짐)

사용자 라이브러리 폴더는 [찾아보기]로 새 폴더를 지정합니다. OS 폴더 선택창에서 고르면 그 경로로 재지정되며, **기존 라이브러리 파일은 옮겨지지 않습니다**(Arduino IDE의 스케치북 위치 변경과 같은 방식 — 재지정일 뿐 이동이 아닙니다). [기본값 복원]은 폴더를 기본 위치로 되돌리며, 이미 기본 위치라면 이 버튼은 비활성화됩니다. 두 동작 모두 성공하면 라이브러리 새로고침이 자동으로 한 번 실행되어(벤더 채널 재동기화 + 설치 목록 재조회 + 자동완성 엔진 재시작) 다음과 같은 안내가 뜹니다.

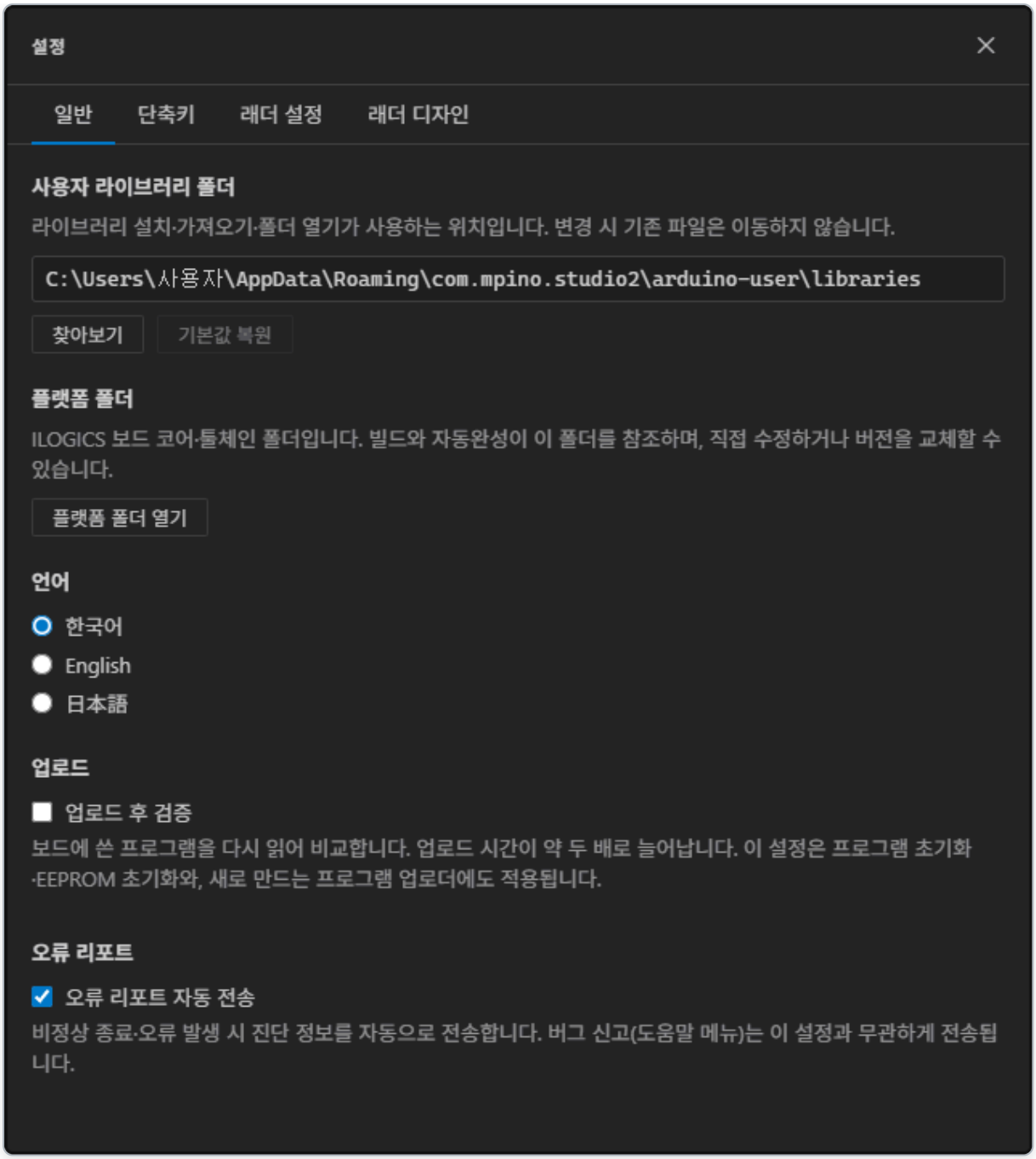
> **"라이브러리 새로고침 — 자동완성 갱신 중(웜 ~3초, 콜드 최대 ~1분 걸릴 수 있습니다)"** — 새로고침 직후 자동완성이 잠시 느려지거나 비어 보일 수 있습니다. 정상 동작이며 잠시 기다리면 회복됩니다.

플랫폼 폴더는 [플랫폼 폴더 열기]를 누르면 OS 파일 탐색기에서 바로 열립니다. 빌드와 자동완성이 참조하는 폴더이므로, 필요하면 탐색기에서 직접 내용을 수정하거나 버전을 교체할 수 있습니다.

언어는 라디오 버튼 선택 즉시 앱 전체 UI가 그 언어로 전환됩니다(별도 저장 버튼 없음). 처음 실행할 때의 기본 언어는 **설치한 판**이 정합니다 — 설치 파일이 한글판·영문판·일어판 세 종류로 나뉘어 있고, 앱 자체는 하나라 설치한 판의 언어로 처음 뜹니다. 여기서 언어를 한 번 고르면 그 선택이 우선하므로, 이후 다른 판으로 업데이트해도 고른 언어가 유지됩니다.

오류 리포트 자동 전송 체크박스는 기본으로 켜져 있으며, 켜두면 비정상 종료·오류 발생 시 진단 정보가 자동으로 전송됩니다. 도움말 메뉴의 버그 신고는 이 설정과 무관하게 항상 전송됩니다.

업로드 후 검증 체크박스는 기본으로 꺼져 있습니다. 켜면 보드에 프로그램을 쓴 뒤 보드에서 다시 읽어 프로그램과 같은지 비교합니다. 하단 패널에는 쓰기가 끝난 뒤 "검증 중" 단계가 더 나오고, 업로드 시간은 **약 두 배**로 늘어납니다. 이 설정은 업로드(Ctrl+U)뿐 아니라 실행 메뉴의 프로그램 초기화·EEPROM 초기화에도 적용되며, **프로그램 업로더 생성...** 으로 새로 만드는 업로더에도 만드는 시점의 값이 그대로 들어갑니다(이미 만든 업로더는 바뀌지 않습니다). 검증에 실패하면 "업로드 후 검증에 실패했습니다 — 보드에 쓴 내용이 프로그램과 다릅니다. 다시 업로드하세요." 가 표시됩니다.



< 그림 3-2 환경설정 모달 — 일반 탭. 사용자 라이브러리 폴더·플랫폼 폴더·언어·업로드 후 검증·오류 리포트 자동 전송 >

단축키

이 탭에서 재정의할 수 있는 단축키는 래더 도구·셀/줄 편집·모니터링 토글·심볼 토글, 총 17개뿐입니다 (그 밖의 전체 단축키 목록은 **단축키**(2-3)를 참고하세요).

명령	기본 단축키
모니터링 토글	Ctrl+M
심볼 토글	Ctrl+Enter

명령	기본 단축키
NO 접점	F2
NC 접점	F3
직선	F5
수직	F6
출력 코일 SET	F7
출력 코일 RESET	F8
출력 코일	F9
신호 반전	F10
상승엣지	F11
하강엣지	F12
박스함수	`\`
셀 삽입	Insert
셀 삭제	Ctrl+Delete
줄 추가	Alt+Insert
줄 삭제	Alt+Delete

심볼 토글(Ctrl+Enter)은 텍스트 입력 중(코드 편집기·셀 입력칸 등)에는 동작하지 않습니다.

각 행에는 [변경]과 [초기화] 버튼이 있고, 표 위쪽 헤더에는 전체를 되돌리는 [모두 default로] 버튼이 있습니다. [변경]을 누르면 "단축키 변경" 다이얼로그가 뜨며, 안내 문구("새 단축키를 누르세요. Enter로 확정, Esc로 취소.")대로 원하는 키(조합)를 누르면 그 자리에서 조합이 표시됩니다. **Enter**로 확정하고 **Esc**로 취소하며, 확정 전에 다른 키를 다시 누르면 방금 잡은 조합을 새 조합으로 덮어씁니다(오타 복구용). 잡은 조합이 이미 다른 명령에 쓰이고 있으면 "이 키는 이미 '저장'에 할당됨 — 그래도 적용 시 충돌됨"처럼 기존 명령 이름을 넣은 경고가 뜨지만, 그래도 [확정]을 누르면 적용됩니다. [초기화]는 그 행 하나만 기본값으로 되돌리며(재정의된 적이 없으면 비활성화), [모두 default로]는 재정의된 항목이 하나도 없으면 비활성화됩니다. 이 탭은 모두 **즉시 적용**되며 별도 저장 버튼은 없습니다.

> **충돌 검사는 이 표의 17개뿐 아니라 전체 단축키 범위에서 이뤄집니다.** 예를 들어 NO 접점을 Ctrl+S로 바꾸면 표에는 없는 "저장" 명령과 충돌합니다. 충돌이 있으면 탭 상단에 "충돌하는 단축키" 경고 목록이 뜨고, 먼저 등록된 쪽이 우선 발화하며 나머지는 눌러도 반응하지 않습니다. 그래도 저장(적용) 자체를 막지는 않습니다 — 의도적으로 특정 기능을 비활성화하려는 경우도 있기 때문입니다.



< 그림 3-3 환경설정 모달 — 단축키 탭. 재정의 가능한 17개 명령과 [변경]/[초기화]/[모두 default로] >

래더 설정

탭 맨 위에는 "래더 셀에 심볼(주소 별명)을 어떻게 표시할지 — 사용자 전역 설정 (이 PC). 프로젝트별 래더 디자인과는 별개입니다"라는 안내가 있습니다. 이 탭의 항목 중 **모니터링 사용 포트**만 프로젝트별(`.mp2`)로 저장되고, 나머지 항목은 모두 이 PC의 **사용자 전역 설정**입니다.

항목	저장 위치	내용
심볼 표시 모드	전역	래더 셀에 주소·심볼을 표시하는 방식 2종
래더 사용	전역	체크 해제 시 아두이노 코드만으로 빌드

항목	저장 위치	내용
이중코일 검사	전역	같은 메모리를 2개 이상의 출력 코일이 쓰면 빌드 에러
접점 생성시 자동 입력박스	전역	접점 생성 도구 사용 시 입력창 자동 오픈
모니터링 사용 포트	**프로젝트별(`.mp2`)**	래더 모니터링 데이터를 보낼 시리얼 포트

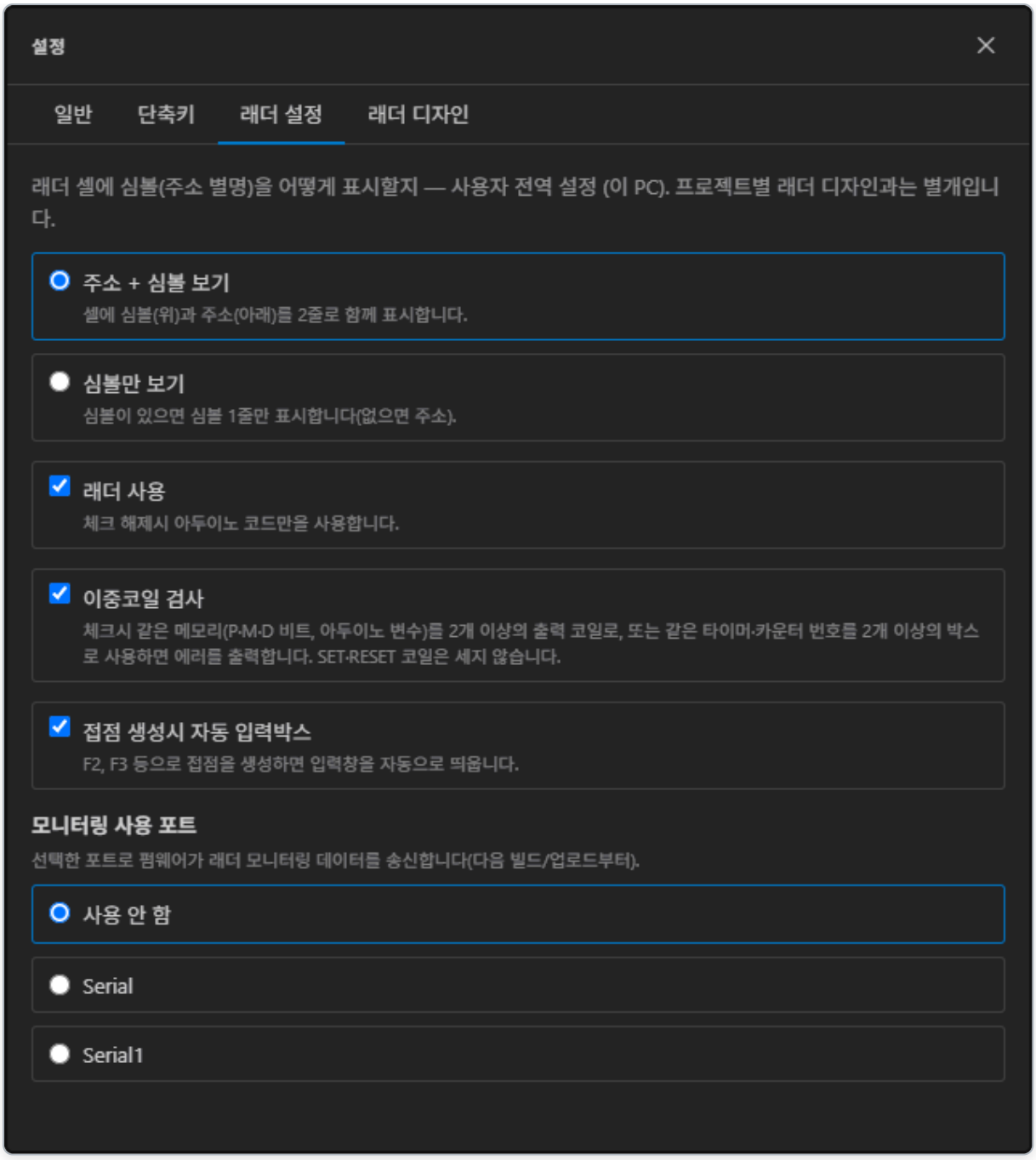
심볼 표시 모드는 라디오 2종입니다. **주소 + 심볼 보기**는 셀에 심볼(위)과 주소(아래)를 2줄로 함께 표시하고, **심볼만 보기**는 심볼이 있으면 심볼 1줄만 표시합니다(없으면 주소). 다음에 다룰 "래더 디자인" 탭(셀 색상·크기 등)과는 별개의, 이 PC에서만 쓰는 표시 방식입니다.

래더 사용 체크박스를 해제하면 빌드 시 래더 생성 코드가 제외되고 순수 아두이노 코드만으로 빌드합니다(래더 탭 자체는 그대로 남아 편집할 수 있습니다).

이중코일 검사를 체크하면 같은 메모리가 2개 이상의 회로에서 출력 코일로 쓰였을 때 빌드가 에러로 막힙니다. P 출력뿐 아니라 `M1`·`D0.5` 같은 내부 메모리와 아두이노 변수도 검사합니다(미쓰비시 PLC와 같은 방식). 같은 비트를 다른 표기로 쓴 것(`P3`과 `WP0.3`, `M0`과 `WM0.0`, `D1.4`와 `DD0.20`)은 같은 코일로 셉니다. SET·RESET 코일은 같은 메모리를 출력 코일과 함께 써도 이중 코일이 아닙니다. 타이머·카운터는 같은 번호(`T0`, `C3`)를 박스 2개 이상이 쓰면 이중 코일입니다(명령이 달라도 같음). 리셋 코일과 `RST` 박스는 세지 않습니다.

접점 생성시 자동 입력박스를 체크하면 F2/F3 등 접점 생성 도구를 쓸 때마다 셀 편집 입력창이 자동으로 열립니다.

모니터링 사용 포트는 라디오로 고르며, 전체 선택지는 **사용 안 함** / **Serial** / **Serial1** / **Serial2** / **Serial3** 다섯 가지이지만 화면에는 **현재 보드가 실제로 가진 UART만** 남아 표시됩니다(예를 들어 UART가 둘뿐인 MPINO-8A4R(T)-S에서는 "사용 안 함 / Serial / Serial1" 세 가지만 보입니다). 고를 수 있게 해 놓고 빌드에서 막는 일이 없도록 하기 위한 것이며, UART가 더 많은 보드로 저장한 프로젝트를 적은 보드로 바꾼 경우에는 그 값이 목록에 그대로 남아 보이므로 다른 포트로 고쳐 주면 됩니다(그대로 빌드하면 "이 보드에 없는 포트"로 빌드가 거부됩니다). **Serial3**은 ATmega2560 보드에만 있는 TTL 레벨 포트(D14 TXD3 / D15 RXD3)라 PC와 연결하려면 **USB-TTL 변환기**가 필요합니다. 선택한 포트로 다음 빌드/업로드부터 래더 모니터링 데이터를 송신합니다 — 포트를 고르는 것 자체가 모니터링 사용 여부를 겸합니다(사용 안 함=꺼짐). 이미 아두이노 코드에서 그 시리얼 포트를 직접 쓰고 있으면(예: `Serial.begin(...)`을 쓰는 코드에서 **Serial**을 선택) "Serial 중복 사용" 경고("아두이노 코드에서 Serial을 사용하고 있습니다. 중복으로 사용할 경우, 래더 모니터링이 정상 동작하지 않을 수 있습니다.")가 뜨지만 확인만 하면 선택은 그대로 적용됩니다.



< 그림 3-4 환경설정 모달 — 래더 설정 탭. 심볼 표시 모드·래더 사용·이중코일 검사·접점 자동 입력박스·모니터링 사용 포트 >

래더 디자인

탭 상단에 "현재 테마(다크)에만 적용됩니다. 비워두면 기본값을 사용합니다"라는 안내가 표시됩니다(괄호 안에는 현재 테마 이름이 들어가며, 이 앱은 현재 다크 테마만 제공합니다). 이 탭은 다른 3개 탭과 달리 값을 바꿔도 곧바로 반영되지 않고 **[저장]을 눌러야** 적용되며, 저장하지 않은 채 다른 탭으로 전환하거나 모달을 닫으려 하면 변경 사항을 버릴지 확인하는 창이 뜹니다.

색상은 6종이 있으며, 이 중 4종만 실제로 적용되고 나머지 2종은 화면에 "(향후 기능)"이라고 표시됩니다(입력은 가능하고 값도 저장되지만 아직 어디에도 적용되지 않습니다).

색상	적용 여부
셀 색상	적용됨
셀 선택 색상	적용됨
라인 색상	적용됨
텍스트 색상	적용됨
심볼 색상	(향후 기능)
모니터링 색상	(향후 기능)

각 색상은 `#RRGGBB` 형식의 hex 텍스트 입력칸과 색상 선택기(스와치)가 나란히 있으며 서로 값이 동기화됩니다. 빈칸으로 두면 기본값을 쓰고, `#RRGGBB` 6자리가 아닌 값을 넣으면 경고가 뜨고 [저장]이 비활성화됩니다.

숫자 항목은 5종이며, 각 입력칸의 범위 안내에 실제 기본값이 함께 표시됩니다.

항목	기본값	범위
가로 (px)	100	100~240
세로 (px)	60	40~120
가로 셀 수	12	4~32
텍스트 크기 (px)	13	9~24
라인 두께	2	1~4(0.5 단위)

빈칸으로 두면 기본값을 쓰고, 범위를 벗어나거나 정수(라인 두께는 0.5 배수)가 아니면 경고가 뜨고 [저장]이 비활성화됩니다.

하단의 [공장 초기화]는 모든 색상·숫자 입력을 빈칸으로 되돌립니다(누르는 즉시 화면에는 반영되지만, [저장]을 눌러야 실제로 적용됩니다). [저장]을 누르면 변경 사항이 기록되고 환경설정 모달이 닫힙니다.



< 그림 3-5 환경설정 모달 — 래더 디자인 탭. 색상 6종(4종 적용 + 2종 향후 기능)과 숫자 5종, 공장 초기화·저장 >

3-2 편집

편집 메뉴는 실행 취소·재실행·잘라내기·복사·붙여넣기·삭제, 6개 항목으로 구성되며 래더 회로를 편집하는 명령입니다(실행 취소·재실행은 코드 탭에서는 코드 에디터에 적용됩니다).

항목	단축키	설명
실행 취소	Ctrl+Z	가장 최근의 래더 편집을 한 단계 되돌립니다
재실행	Ctrl+Y	실행 취소로 되돌린 편집을 한 단계 다시 적용합니다

항목	단축키	설명
잘라내기	Ctrl+X	선택한 래더 셀을 클립보드로 옮기고 원본을 비웁니다
복사	Ctrl+C	선택한 래더 셀을 클립보드로 복사합니다
붙여넣기	Ctrl+V	클립보드의 셀을 현재 선택 위치에 붙여넣습니다
삭제	Delete	선택한 래더 셀을 비웁니다

실행 취소(Ctrl+Z) 는 셀 도구 적용·잘라내기·붙여넣기·삭제 등 되돌릴 수 있는 가장 최근 래더 편집 한 단계를 취소합니다. 되돌릴 편집이 없으면 아무 동작도 하지 않습니다.

재실행(Ctrl+Y) 은 실행 취소로 되돌린 편집을 다시 적용합니다. 실행 취소를 여러 번 했다면 재실행도 그만큼 차례로 되살릴 수 있습니다. 실행 취소 뒤에 새 편집을 하면 그때까지 되돌려 둔 편집은 재실행 대상에서 사라집니다. 다시 적용할 편집이 없으면 아무 동작도 하지 않습니다.

잘라내기(Ctrl+X) 는 선택한 셀을 클립보드에 담고 그 자리를 비웁니다. **복사(Ctrl+C)** 는 원본은 그대로 두고 클립보드에만 담습니다. **붙여넣기(Ctrl+V)** 는 클립보드에 담긴 셀들을 현재 선택된 셀을 좌상단 기준점 삼아 붙여넣으며(선택한 셀이 없으면 그리드 첫 행 첫 칸부터), 클립보드가 비어 있으면 아무 동작도 하지 않습니다.

복사·잘라내기한 셀은 시스템 클립보드에도 함께 담깁니다. 그래서 MPINO Studio 2를 두 개 띄워 놓고, 한 창에서 복사한 회로를 다른 창의 래더에 그대로 붙여넣을 수 있습니다.

삭제(Delete) 는 선택한 셀을 비웁니다. 단일 셀을 선택했을 때, 세로 연결선(F6)이 함께 있는 셀은 첫 번째 삭제로 연결선만 제거되고 다시 한 번 삭제해야 셀 전체가 비워집니다. 여러 셀을 범위 선택한 상태에서는 한 번의 삭제로 연결선과 내용이 함께 지워집니다.



< 그림 3-6 편집 메뉴 — 실행 취소부터 삭제까지 6개 항목이 표시된 드롭다운 >

편집 메뉴 항목은 실행 위치에 따라 다르게 동작합니다.

- **잘라내기·복사·붙여넣기·삭제는 래더 탭이 활성일 때만 동작합니다.** 코드 탭이 활성인 상태에서 이 네 항목을 클릭하면 아무 일도 일어나지 않습니다.

- **실행 취소·재실행은 활성 탭을 따라갑니다.** 래더 탭이 활성이면 래더 편집을, 코드 탭이 활성이면 그 코드 에디터의 텍스트 편집을 되돌리거나 다시 적용합니다. 코드 탭에서 클릭해도 보이지 않는 래더가 바뀌지 않습니다.
- 코드 에디터 안에서 직접 Ctrl+Z·Ctrl+Y·Ctrl+X·Ctrl+C·Ctrl+V를 누르면 Monaco 에디터의 일반 텍스트 실행 취소·재실행·잘라내기·복사·붙여넣기가 동작합니다(Ctrl+Shift+Z 도 코드 에디터에서는 재실행입니다).

셀 삽입/삭제·줄 추가/삭제 등 그 외 편집 단축키는 **단축키**(2-3)를 참고하세요.

3-3 보기

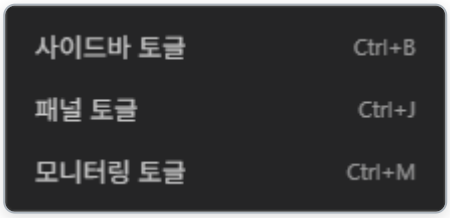
보기 메뉴는 사이드바 토글·패널 토글·모니터링 토글, 3개 항목으로 구성되며 각 항목은 화면의 한 영역을 열고 닫습니다.

항목	단축키	설명
사이드바 토글	Ctrl+B	왼쪽 사이드바를 열고 닫습니다
패널 토글	Ctrl+J	에디터 아래쪽 하단 패널을 열고 닫습니다
모니터링 토글	Ctrl+M	래더 실시간 모니터링 통신을 켜고 끕니다

사이드바 토글(Ctrl+B) 은 화면 왼쪽의 사이드바를 열고 닫습니다. 사이드바에는 액티비티바에서 선택한 패널이 표시되며, 어떤 패널이 선택돼 있었는지는 사이드바를 닫았다 다시 열어도 그대로 유지됩니다(액티비티바 상세는 이 챕터의 범위 밖입니다).

패널 토글(Ctrl+J) 은 에디터 아래쪽의 하단 패널(빌드 출력·문제·니모닉·시리얼 모니터 탭)을 열고 닫습니다. 패널 높이와 마지막에 선택했던 탭은 닫았다 다시 열어도 그대로 유지됩니다.

모니터링 토글(Ctrl+M) 은 래더 실시간 모니터링 통신을 켜고 끕니다. 환경설정에서 모니터링 사용 포트가 지정돼 있지 않으면 통신을 켜는 대신 포트를 설정하라는 안내 창이 뜹니다. 모니터링 화면 구성과 데이터 표시 방식은 이 챕터의 범위 밖입니다.



< 그림 3-7 보기 메뉴 — 사이드바 토글부터 모니터링 토글까지 3개 항목이 표시된 드롭다운 >

3-4 실행

실행 메뉴는 빌드·업로드·프로그램 초기화·EEPROM 초기화·업로드 취소·프로그램 업로더 생성, 그리고 통신·아두이노에 값 쓰기·메모리 쓰기 목록 창을 여는 9개 항목으로 구성됩니다.

항목	단축키	설명
빌드	Ctrl+R	현재 프로젝트를 트랜스파일·컴파일해 오류가 없는지 검증합니다
업로드	Ctrl+U	현재 프로젝트를 빌드한 뒤 선택한 포트로 보드에 플래시합니다
프로그램 초기화...	—	보드의 프로그램을 빈 프로그램으로 바꿔 모든 출력을 끕니다(정전유지 값은 유지)
EEPROM 초기화...	—	프로그램 초기화와 같이 빈 프로그램을 올리고, EEPROM 전체(정전유지 값 포함)를 지웁니다
업로드 취소	—	진행 중인 업로드를 중단합니다
프로그램 업로더 생성...	—	현재 프로젝트를 컴파일해, MPINO Studio 2 없이 보드에 올릴 수 있는 업로더 실행 파일(.exe)을 만듭니다
통신	—	통신 채널(프로토콜) 설정 창을 엽니다
아두이노에 값 쓰기	—	실행 중인 보드의 P/M/D/C/T/R 메모리 주소 하나에 값을 쓰는 팝업을 엽니다(주소 칸은 비어 있음) — 상세는 **아두이노에 값 쓰기**(4-4)
메모리 쓰기 목록	—	메모리 쓰기 목록 창을 엽니다



< 그림 3-8 실행 메뉴 — 빌드부터 메모리 쓰기 목록까지 9개 항목이 표시된 드롭다운 >

빌드(Ctrl+R) 는 현재 프로젝트를 트랜스파일·컴파일해 문법·빌드 오류가 없는지 검증합니다(보드에는 아무 것도 쓰지 않습니다). 결과는 하단 패널의 빌드 출력 탭에 상태(대기/빌드 중/성공/실패)와

로그로 표시됩니다.

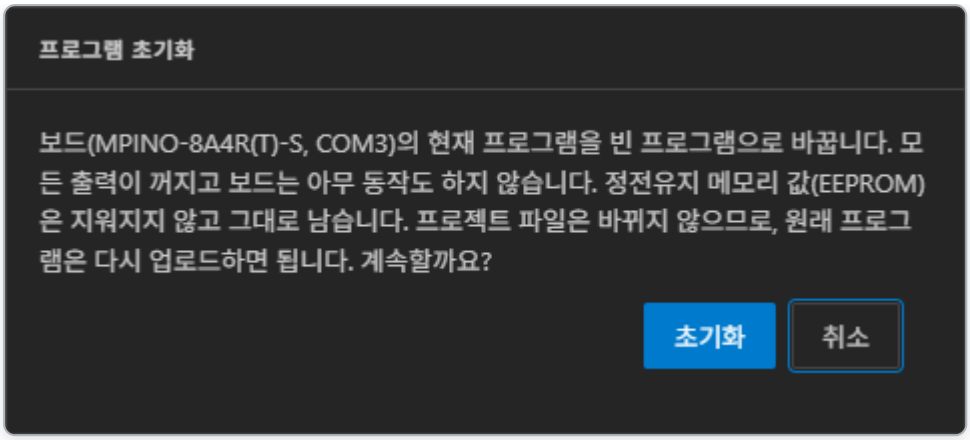
업로드(Ctrl+U) 는 빌드와 마찬가지로 현재 프로젝트를 트랜스파일·컴파일한 뒤 이어서 선택된 포트로 보드에 곧바로 플래시까지 한 번에 실행합니다(먼저 따로 빌드할 필요 없음). 포트가 선택돼 있지 않으면 포트를 먼저 선택하라는 오류 안내(상태바 우측 ▾ 클릭으로 선택)가 뜨고 업로드는 실행되지 않습니다. 업로드 진행 상황은 하단 패널에 실시간으로 표시됩니다.

래더 모니터링이 쓰기로 한 포트를 다른 곳에서도 쓰고 있으면, 업로드를 시작하기 전에 **"Serial1 중복 사용"** 처럼 포트 이름이 들어간 확인 창이 먼저 뜹니다. 상대가 통신 설정의 채널인지, 아두이노 코드의 통신 초기화 호출인지, 코드가 그 포트를 직접 쓰는 것인지에 따라 본문과 해결 방법 안내가 갈립니다. 막는 것이 아니라 경고이므로 [업로드]를 누르면 그대로 진행되고, [취소]를 누르면 업로드가 실행되지 않습니다. 자세한 세 갈래는 **포트 선택과 업로드**(1-8)를 참고하세요.

업로드 취소 는 업로드가 실제로 진행 중일 때만 의미가 있으며(그 외 상태에서 클릭해도 아무 동작도 하지 않습니다), 클릭 즉시 진행 중이던 업로드를 중단시킵니다. 취소된 업로드는 실패가 아니라 별도의 "취소됨" 상태로 표시됩니다.

프로그램 초기화... 는 보드에 들어 있는 프로그램을 **빈 프로그램**으로 바꿉니다. 초기화가 끝나면 모든 출력(릴레이·트랜지스터·아날로그 출력)이 꺼지고 보드는 아무 동작도 하지 않습니다. 보드를 잠시 멈춰 두거나 배선을 점검할 때, 또는 다른 장비에 옮기기 전에 이전 프로그램이 출력을 움직이지 않게 할 때 씁니다.

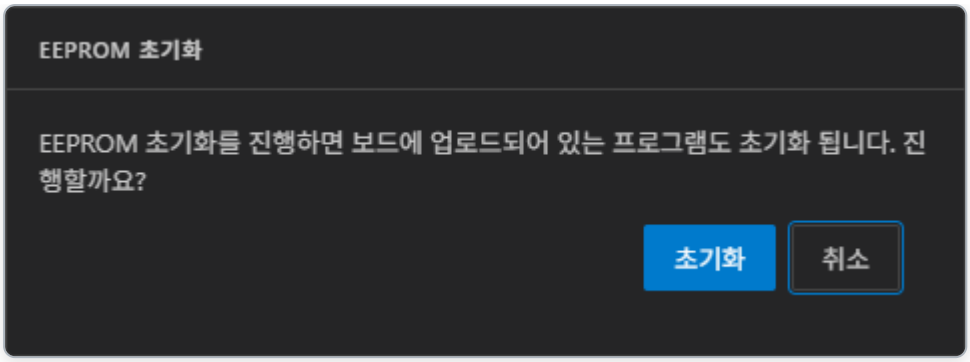
- **대상 보드와 포트**: 지금 열려 있는 프로젝트의 보드(모듈 구성 포함)와 상태바에서 고른 포트를 그대로 씁니다 — 업로드와 같습니다. 포트가 선택돼 있지 않으면 업로드와 같은 안내가 뜨고 실행되지 않습니다.
- **확인 창**: 실행 전에 반드시 확인 창이 뜨고, 처음 선택된 버튼은 [취소]입니다. [초기화]를 눌러야 진행됩니다.
- **남는 것**: 정전유지 메모리 값(EEPROM)은 지워지지 않습니다. 프로젝트 파일과 에디터 내용도 바뀌지 않으므로, 원래 프로그램은 **업로드(Ctrl+U)** 로 다시 올리면 됩니다.
- **진행과 취소**: 진행 상황은 업로드처럼 하단 패널에 "프로그램 초기화" 구분선과 함께 표시되고, **업로드 취소**로 중단할 수 있습니다. 업로드나 초기화가 진행 중일 때는 다시 실행되지 않습니다.
- **모니터링**: 래더 모니터링이 켜져 있었다면 초기화하면서 꺼집니다. 빈 프로그램에는 모니터링 기능이 없기 때문입니다. 초기화 뒤 모니터링을 켜면 먼저 프로그램을 업로드할지 묻고, [업로드]가 성공해야 모니터링이 켜집니다.



< 그림 3-9 프로그램 초기화 확인 창 — 대상 보드·포트와 남는 것을 알리고, 처음 선택된 버튼은 [취소] >

EEPROM 초기화... 는 보드의 EEPROM **전체**(4KB)를 공장 출하 상태(모든 바이트 `0xFF`)로 지웁니다. 정전유지 값뿐 아니라 아두이노 코드가 `EEPROM.put` 등으로 저장한 데이터도 모두 지워집니다. 지우는 일은 보드에 올린 빈 프로그램이 하므로, **보드의 프로그램도 프로그램 초기화와 똑같이 빈 프로그램으로 바뀝니다**(모든 출력 OFF).

- **확인 창**: "EEPROM 초기화를 진행하면 보드에 업로드되어 있는 프로그램도 초기화 됩니다. 진행할까요?" 라고 묻고, 처음 선택된 버튼은 [취소]입니다.
- **완료 확인**: 업로드가 끝난 뒤 보드가 EEPROM 을 지우는 데 걸리는 시간은 칩마다 다릅니다. 처음 지울 때 ATmega128 보드(MPINO-8A4R-S·8A4R-SU·8A8R-S·16A16R)는 **최대 약 40초**, ATmega2560 보드(그 밖의 MPINO·MPAINO)는 **최대 약 20초**입니다. 업로드가 끝나면 하단 패널에 바로 "보드가 EEPROM 을 지우는 중입니다 … 보드 전원을 끄지 마세요" 가 표시되고, 앱은 보드가 보내는 완료 신호를 ATmega128 은 최대 63초, ATmega2560 은 최대 31초 기다렸다가 **"EEPROM 초기화 완료: COMx (4096 바이트를 0xFF 로)"** 를 표시합니다. 이 줄이 나오기 전에는 **보드 전원을 끄지 마세요.** 그 시간 안에 신호가 오지 않으면 성공이 아니라 경고로 표시되며, 그 때는 잠시 기다렸다가 EEPROM 초기화를 다시 실행하면 됩니다(이미 지워진 바이트는 다시 쓰지 않으므로 반복해도 EEPROM 수명에 부담이 없습니다).
- **지우는 순서**: EEPROM 의 끝(정전유지 확인 표식이 있는 곳)부터 지웁니다. 그래서 지우는 도중 전원이 끊겨도 반쯤 지워진 값이 정전유지 값으로 되살아나지 않습니다.
- **완료 확인이 안 될 때**: 업로드 직후 시리얼 모니터 같은 다른 프로그램이 먼저 포트를 잡으면 앱이 완료 신호를 받으려고 포트를 열지 못해 경고가 표시됩니다(보드는 계속 지웁니다 — 안내된 시간 동안 전원을 끄지 마세요).
- **정전유지 값**: 정전유지 기능은 EEPROM 끝에 적어 둔 확인 표식이 맞을 때만 값을 되살립니다. 초기화로 이 표식이 지워지므로, 정전유지를 쓰는 프로그램을 다시 업로드하면 새 보드에 처음 올렸을 때처럼 **초기값에서 시작**합니다.
- 대상 보드·포트, 진행 표시와 **업로드 취소**, 모니터링이 꺼지고 다시 켤 때 업로드를 먼저 묻는 것은 프로그램 초기화와 같습니다. 완료 신호를 기다리는 중에 취소하면 기다림만 멈추고, 보드는 지우기를 계속할 수 있습니다.



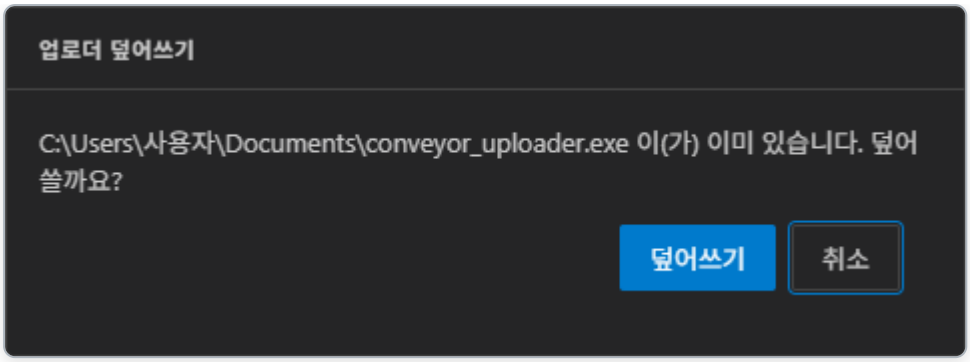
< 그림 3-10 EEPROM 초기화 확인 창 — 보드의 프로그램도 초기화된다고 묻고, 처음 선택된 버튼은 [취소] >

프로그램 업로더 생성... 은 현재 프로젝트를 컴파일한 프로그램을 품은 **업로더 실행 파일**(.exe)을 만듭니다. 이 파일만 있으면 MPINO Studio 2·아두이노 프로그램·라이브러리를 설치하지 않은 PC에서도 MPINO·MPAINO 보드에 프로그램을 올릴 수 있으므로, 현장 담당자나 고객에게 프로그램을 전달할 때 씁니다.

- **만드는 방법**: 메뉴를 누르면 "업로더를 저장할 폴더 선택" 창이 뜹니다. 폴더를 고르면 빌드(Ctrl+R)와 똑같이 컴파일한 뒤 그 폴더에 `<프로젝트명>_uploader.exe` 를 만듭니다. 진행 상황은 하단 패널 빌드 출력 탭에 "--- 프로그램 업로더 생성 ---" 구분선과 함께 표시되고, 끝나면 "업로더를 만들었습니다: ..." 줄(파일 크기·프로그램 크기·보드·쓰기 후 검증 여부)과 함께 알림이 뜹니다. 알림의 [폴더 열기]를 누르면 탐색기에서 그 폴더가 열립니다.
- **대상 보드**: 지금 열려 있는 프로젝트의 보드(모듈 구성 포함)가 그대로 들어갑니다. 업로드와 달리 포트는 고르지 않아도 됩니다 — 포트는 업로더를 실행할 때 고릅니다.
- **같은 이름이 있을 때**: "업로더 덮어쓰기" 확인 창이 뜨고, 처음 선택된 버튼은 [취소]입니다. [덮어쓰기]를 눌러야 바뀝니다. 그 업로더가 실행 중이면 덮어쓸 수 없다는 안내가 뜨고 기존 파일은 그대로 남습니다 — 업로더를 닫고 다시 시도하세요.
- **만들어지지 않는 경우**: 컴파일 오류가 있으면 빌드 실패와 같이 표시되고 파일은 만들어지지 않습니다. 업로드·초기화 같은 다른 작업이 진행 중일 때는 "다른 작업이 진행 중입니다" 안내만 뜹니다.
- **언어와 검증**: 업로더 화면은 만들 때의 MPINO Studio 2 언어(한국어/English/日本語)로 표시됩니다. 환경설정 일반 탭의 **업로드 후 검증** 값도 만들 때 그대로 들어갑니다.



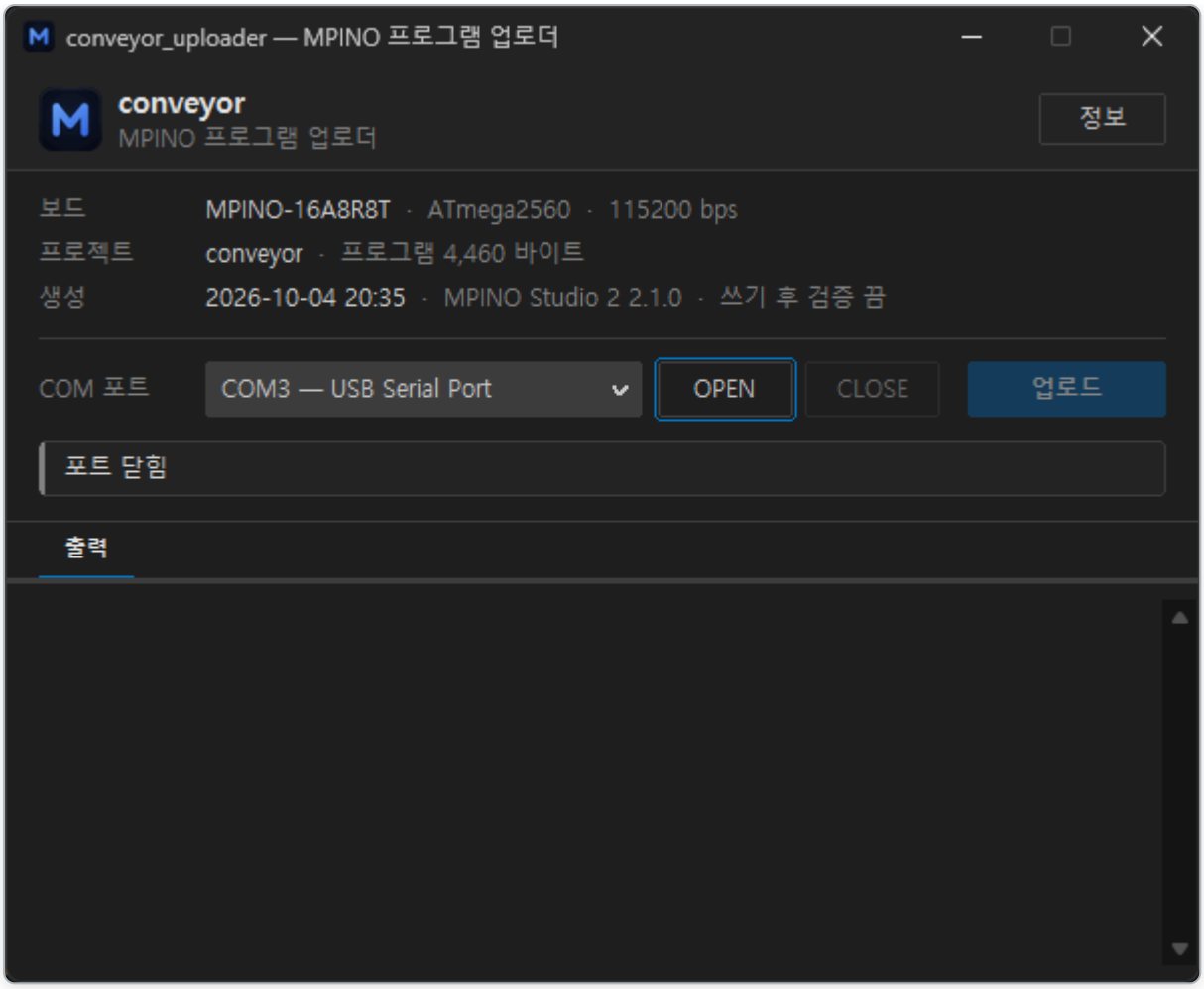
< 그림 3-11 업로더 생성 완료 알림 — 만든 파일 이름과 [폴더 열기] 버튼 >



< 그림 3-12 업로더 덮어쓰기 확인 창 — 같은 이름의 업로더가 이미 있을 때 뜨며, 처음 선택된 버튼은 [취소] >

업로더 사용법은 다음과 같습니다(Windows 10/11 64비트).

1. 받은 `<프로젝트명>_uploader.exe` 를 실행합니다. 창 위쪽에 보드·프로젝트·프로그램 크기· 생성 시각이 표시됩니다.
2. 보드를 USB 케이블로 PC에 연결하고 **COM 포트** 목록에서 보드의 포트를 고릅니다. 케이블을 꽂거나 뽑으면 목록이 저절로 갱신됩니다.
3. **OPEN** 을 누릅니다. 포트를 쓸 수 있는지만 확인하며, 이때 보드는 리셋되지 않습니다(돌던 프로그램이 멈추지 않습니다). 상태가 "포트 열림 (COMx)" 로 바뀌고 **업로드** 버튼이 활성화됩니다. 다른 프로그램(MPINO Studio 2 의 시리얼 모니터 등)이 그 포트를 쓰고 있으면 사용 중이라는 안내가 뜨므로, 그 프로그램을 닫고 다시 OPEN 합니다.
4. **업로드** 를 누릅니다. 진행 막대가 "보드에 연결 중..." → "보드에 쓰는 중 NN%" 로 바뀌고, 끝나면 로그에 "업로드 완료 — N 바이트, N초" 가 표시됩니다. 업로드 후 검증을 켜고 만든 업로더는 쓰기 뒤 "검증 중 NN%" 를 거쳐 "업로드·검증 완료" 로 끝납니다. 포트는 열린 채로 남으므로 OPEN 을 다시 누르지 않고 이어서 업로드할 수 있습니다.
5. 다 끝나면 **CLOSE** 로 포트를 놓습니다.



< 그림 3-13 업로더 창 — 보드·프로젝트·생성 정보, COM 포트 목록과 OPEN·CLOSE·업로드 버튼, 상태 줄과 출력 로그(포트 닫힘 상태) >

- **[정보]**: 프로젝트·보드·FQBN·MCU와 업로드 방식·프로그램 크기·검증 여부·생성 시각·만든 MPINO Studio 2 버전과, 업로더에 포함된 오픈소스 라이선스 고지를 보여 줍니다. 업로드 중에도 열 수 있습니다.
- **업로드가 안 될 때**: "보드가 응답하지 않습니다" 는 포트가 맞는지, 보드 전원이 켜져 있는지 확인하세요. "연결된 보드의 MCU … 가 이 업로더의 대상 … 과 다릅니다" 는 다른 종류의 보드에 연결한 경우로, 이때는 보드에 아무것도 쓰지 않습니다. 업로드 중에 케이블이 빠지거나 창을 닫아 중간에 멈췄다면 보드 프로그램이 불완전할 수 있으니 다시 업로드하면 복구됩니다.
- **지워지지 않는 것**: 업로드(Ctrl+U)와 같이 보드의 EEPROM(정전유지 값)은 건드리지 않습니다.



< 그림 3-14 업로더 [정보] 창 — 프로젝트·보드·FQBN·MCU와 업로드 방식·프로그램 크기·검증 여부·생성 정보, 오픈소스 라이선스 고지 >

> **"Windows의 PC 보호" 경고가 뜰 때** — 업로더는 코드 서명이 없어서, 메일이나 웹으로 받은 파일을 처음 실행하면 Microsoft Defender SmartScreen 이 "Windows의 PC 보호" 창을 띄울 수 있습니다. 신뢰하는 곳에서 받은 업로더라면 창의 **[추가 정보]** 를 누른 뒤 나타나는 **[실행]** 을 누르면 실행됩니다.

> **빌드와 업로드는 모두 저장 여부와 관계없이 에디터의 현재 내용 그대로 실행됩니다.** 저장하지 않은 변경 사항이 있어도 그대로 반영되며, 저장을 요구하는 확인창은 뜨지 않습니다.

3-4-1 통신

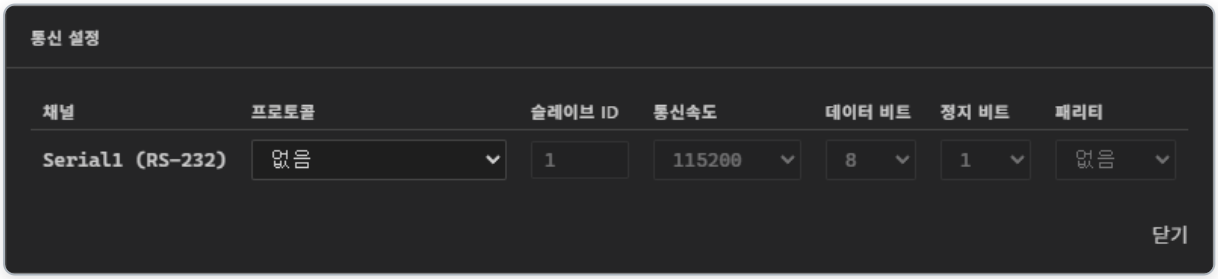
통신(단축키 없음)을 클릭하면 "통신 설정" 창이 뜹니다. 창을 열 때마다 현재 보드의 통신 채널 정보를 새로 불러오며, 불러오는 동안에는 "채널 정보 로딩 중..."이, 실패하면 "보드 정보를 불러오지 못했습니다."가 표시됩니다. 통신 채널이 없는 보드를 선택 중이면 "이 보드는 통신 채널이 없습니다."라는 안내만 표시됩니다.

통신 채널이 있는 보드는 채널마다 한 행씩 표가 표시되며, 각 행은 다음 7개 열로 구성됩니다.

열	내용
채널	채널 이름(보드마다 다름, 예: Serial1 (RS-232))
프로토콜	없음 / Modbus RTU Slave / LS산전 Cnet Slave / Modbus RTU Master(추후 지원) / Modbus TCP/IP(추후 지원)
슬레이브 ID	1~247
통신속도	300~500000bps 중 선택(기본 115200)
데이터 비트	5/6/7/8(기본 8)
정지 비트	1/2(기본 1)
패리티	없음/짝수/홀수(기본 없음)

프로토콜을 "없음"으로 두면 그 채널은 사용하지 않으며, 나머지 5개 열은 비활성 상태로 표시됩니다. **Modbus RTU Slave** 또는 **LS산전 Cnet Slave**를 선택하면 나머지 5개 열이 활성화됩니다. **Modbus RTU Master**와 **Modbus TCP/IP**는 "(추후 지원)"으로 표시되며 아직 선택할 수 없습니다. 두 동작 프로토콜(Modbus RTU Slave ↔ LS산전 Cnet Slave) 사이를 전환하면 이미 입력한 5개 값이 그대로 유지되고, 없음(또는 추후 지원 옵션)에서 동작 프로토콜로 새로 전환하면 슬레이브 ID 1·통신속도 115200·데이터 비트 8·정지 비트 1·패리티 없음의 기본값으로 초기화됩니다.

변경 사항은 즉시 프로젝트에 반영되며(별도 저장 버튼 없음), 다음 빌드나 업로드부터 곧바로 적용됩니다 — 프로젝트 파일을 따로 저장하지 않아도 됩니다. **닫기** 버튼으로 창을 닫습니다.



< 그림 3-15 통신 설정 창 — 채널별 프로토콜·슬레이브 ID·통신속도·데이터 비트·정지 비트·패리티 >

3-4-2 메모리 쓰기 목록

메모리 쓰기 목록(단축키 없음)을 클릭하면 실행 중인 보드의 P/M/D/C/T/R 메모리 주소에 값을 쓰고 현재 값을 실시간으로 관찰하는 창이 뜹니다. 래더 셀을 우클릭해 여는 개별 값 쓰기 팝업과는 별개의 기능입니다(래더 편집 화면 관련 내용은 이 챕터의 범위 밖입니다). 다른 설정 모달과 달리 배경을 덮지 않는 **모달리스** 창이라 열려 있는 동안에도 래더 편집을 계속할 수 있으며, 헤더를 드래그해 위치를 옮길 수 있습니다. 헤더의 ✕ 또는 Esc로 닫습니다.

모니터링 사용 포트만 설정돼 있으면 **모니터링을 켜지 않아도** 값을 쓸 수 있습니다. 모니터링이 꺼져 있으면 쓰는 순간에만 포트를 열어 쓰고 확인한 뒤 닫습니다(보드는 리셋되지 않습니다 — 자세한 동작과 결과 안내는 **아두이노에 값 쓰기**(4-4) 참고). 모니터링 사용 포트가 **사용 안 함**이면 쓰기 대신 설정 경로를 안내하는 창이 뜹니다. 헤더의 **모니터링** 버튼은 현재값 열을 실시간으로 보고 싶을 때 켭니다. P 주소는 출력 핀과 핀에 연결되지 않은 P만 쓸 수 있고, 입력 핀 P는 "입력 핀은 쓰기할 수 없음"으로 거부됩니다.

창 상단의 입력 행(주소 + 쓸 값)에 값을 입력하고 **Write**를 누르거나 Enter를 치면, 그 자리에서 보드에 값을 쓰고 성공 시 아래 목록에 그 주소가 새 행으로 추가됩니다(이미 목록에 있는 주소면 그 행의 값만 갱신). 쓰기에 실패하면 목록은 바뀌지 않고 오류 토스트가 뜹니다.

목록 위 툴바에는 4개 버튼이 있습니다.

버튼	동작
가져오기	파일에서 목록을 불러와 현재 목록을 통째로 교체합니다(선택 취소 시 기존 목록 유지)
내보내기	현재 목록을 파일로 저장합니다
지우기	목록을 전부 삭제합니다(확인 창을 거치며, 목록이 비어 있으면 비활성화)
전체 쓰기	목록의 모든 행을 순서대로 보드에 씁니다

전체 쓰기는 처리 중이거나, 목록에 문법·범위 오류가 있는 행이 하나라도 있으면 비활성화됩니다. 실행되면 쓸 값이 비어 있는 행은 건너뛰고 나머지 행을 차례로 쓰되, 도중에 한 행이라도 쓰기에 실패하면 그 자리에서 즉시 중단하고 오류를 표시합니다(이후 행은 시도하지 않음).

목록 표는 등록한 순서 그대로 표시되며(정렬 없음), 주소·현재값·쓸 값 3개 열에 이어 열 제목이 없는 쓰기·삭제 버튼 열이 붙습니다.

- **주소·쓸 값** 셀은 클릭하면 바로 그 자리에서 입력 상태로 바뀝니다 (Enter로 확정, Esc로 취소, 다른 곳 클릭도 확정). 주소를 `P` / `M` / `D` / `C` / `T` / `R` 뒤에 숫자가 오는 형식이 아니거나 이미 목록에 있는 주소로 바꾸려 하면 변경이 거부되고 표 아래에 사유가 표시됩니다.
- **현재값**은 모니터링으로 수신한 실시간 값을 표시하며, 값이 없으면 "—"으로 표시됩니다.
- 주소 형식은 맞지만 현재 보드의 메모리 범위를 벗어나는 주소는 그 행이 오류로 표시되고 표 아래에 범위를 알려주는 안내가 함께 뜹니다. 이 경우 편집 자체는 거부되지 않지만 그 행의 **Write** 버튼은 비활성화됩니다.
- 각 행의 **Write** 버튼(툴팁: 쓰기)은 그 행 하나만 보드에 씁니다. 같은 행의 ✕ 버튼(툴팁: 삭제)은 보드에는 아무 것도 쓰지 않고 목록에서만 그 행을 지웁니다.

목록이 비어 있으면 "등록된 메모리 없음 — 위 입력 행에서 주소(M/D/C/T/R)를 추가하세요."라는 안내만 표시됩니다.



< 그림 3-16 메모리 쓰기 목록 — 상단 입력 행·가져오기/내보내기/지우기/전체 쓰기 툴바·주소별 현재값 관찰 표 >

3-5 설정

설정 메뉴는 6개 항목으로 구성됩니다. 이 중 5개는 챕터 2의 필독사항이나 이 챕터 앞부분의 환경설정 절, 또는 아래 **보드 변경**처럼 다른 화면으로 곧바로 이동하는 진입점이고, EEPROM 정전유지만 이 절에서 처음 소개합니다.

항목	단축키	설명
메모리 영역 설정	Ctrl+Shift+M	프로젝트별 메모리 영역 크기와 타이머 단위를 조정하는 모달을 엽니다
래더 디자인 설정	Ctrl+Shift+D	래더 셀 색상·크기 등 디자인을 조정하는 모달을 엽니다
래더 핀맵 설정	Ctrl+Shift+P	현재 보드의 핀 정의를 편집하는 모달을 엽니다
보드 변경	—	새로 열지 않고 현재 프로젝트의 대상 보드만 다른 보드로 교체합니다
사용한 메모리	Ctrl+Shift+L	프로젝트가 실제 사용 중인 주소를 확인하는 모달을 엽니다
EEPROM 정전유지	Ctrl+Shift+E	정전 후에도 유지할 메모리 영역과 저장 방식을 설정하는 모달을 엽니다

메모리 영역 설정(Ctrl+Shift+M) 은 보드 기본값을 넘어 6개 메모리 영역의 개수를 프로젝트별로 조정하는 모달을 엽니다. 이 모달의 아래쪽에서는 타이머 단위(`T*` 1ms/10ms, `TA*` 10ms/100ms/1000ms)도 고를 수 있습니다. 자세한 내용은 **데이터 메모리**(2-1)의 "영역 크기와 메모리 설정"과 **타이머 단위 설정**(6-1-6)을 참고하세요.

래더 디자인 설정(Ctrl+Shift+D) 은 환경설정 모달을 "래더 디자인" 탭으로 곧바로 엽니다 — 이 챕터 앞부분의 **환경설정**(3-1-1)에서 다룬 것과 같은 화면입니다.

래더 핀맵 설정(Ctrl+Shift+P) 은 현재 프로젝트가 사용 중인 보드의 핀 정의를 표에서 편집하는 모달을 엽니다. 자세한 내용은 **핀 편집**(2-4)의 "보드 핀맵 편집"을 참고하세요.

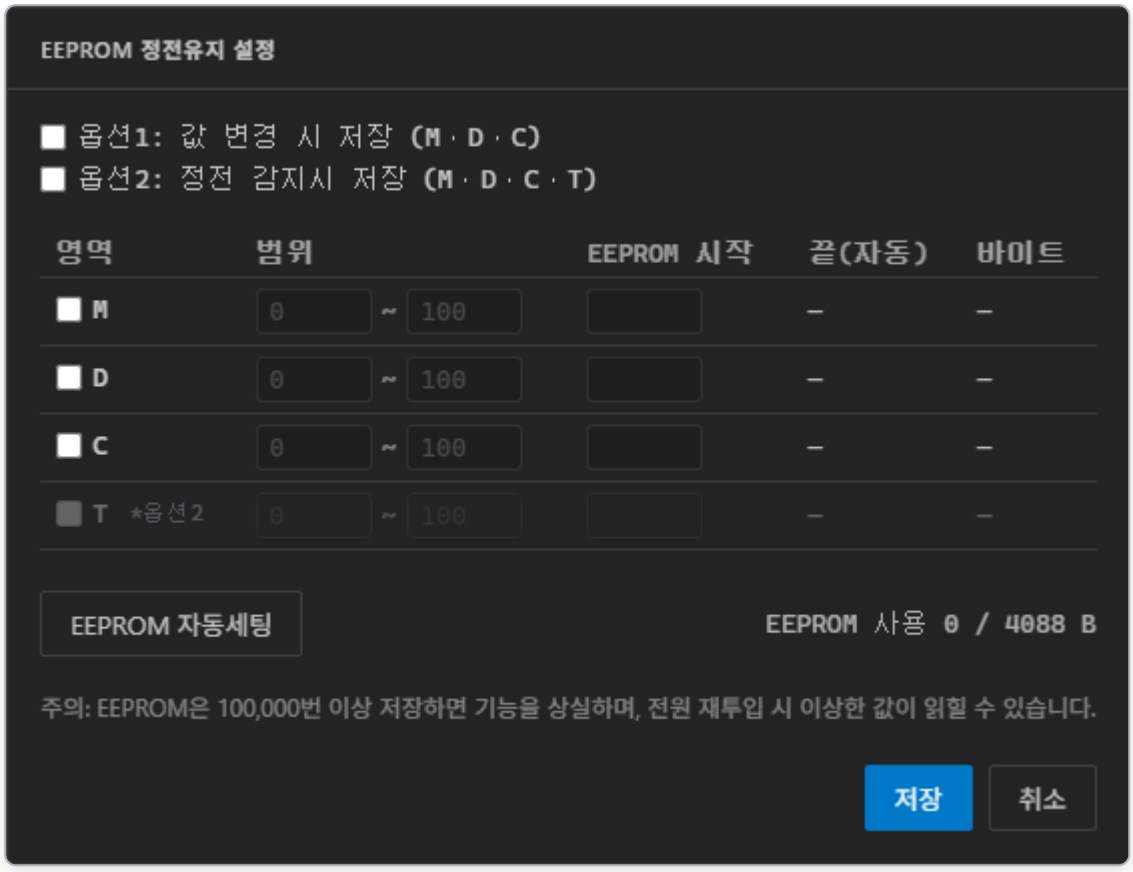
보드 변경(단축키 없음)은 새 프로젝트를 만들지 않고 현재 프로젝트가 대상으로 하는 보드만 다른 보드로 바꾸는 보드 선택 모달을 엽니다. 지금 선택된 보드가 초기값으로 미리 선택된 채 열리며, 다른 보드를 골라 확정하면 코드·래더는 그대로 둔 채 대상 보드 정보만 교체됩니다(취소하면 보드는 바뀌지 않습니다). 이 모달은 새로 만들기에서 열리는 것과 같은 화면이라 **MPINO/MPAINO** 두 카테고리 탭, MPAINO 보드의 **옵션 모듈** 개수 지정, **Serial RX 버퍼** 구획도 그대로 쓸 수 있습니다 — 자세한 내용은 **새 프로젝트 만들기와 보드 선택**(1-4)을 참고하세요. 같은 모달은 탐색기 패널의 "보드" 줄 옆 **[보드 변경]** 버튼으로도 열 수 있습니다 — 자세한 내용은 **탐색기 패널**(9-2)을 참고하세요.

사용한 메모리(Ctrl+Shift+L) 는 프로젝트가 실제로 사용 중인 주소를 영역별 격자에서 확인하는 모달을 엽니다. 자세한 내용은 **데이터 메모리**(2-1)의 "사용한 메모리 확인"을 참고하세요.



< 그림 3-17 설정 메뉴 — 메모리 영역 설정부터 EEPROM 정전유지까지 6개 항목이 표시된 드롭다운 >

EEPROM 정전유지(Ctrl+Shift+E) 는 "EEPROM 정전유지 설정" 창을 열어, 전원이 꺼져도 유지할 M·D·C·T 메모리 영역과 저장 시점을 지정합니다. 저장 시점은 옵션1(값이 바뀔 때마다 저장, M·D·C만)과 옵션2(정전 감지 시 저장, M·D·C·T 모두), 두 개의 독립된 체크박스로 정하며 둘 다 동시에 켤 수도 있습니다. 옵션2를 쓰려면 전원감지 핀을 지정해야 합니다(감지시간(ms)은 비워두면 0으로 처리됩니다). 영역별로 지정한 EEPROM 주소는 [EEPROM 자동세팅]으로 겹치지 않게 한 번에 배치할 수 있고, 전체 사용량이 4088바이트(전체 4096바이트 중 8바이트는 내부용으로 예약)를 넘거나 서로 겹치면 저장이 막힙니다.



< 그림 3-18 EEPROM 정전유지 설정 — 옵션1/2, 영역별 범위·EEPROM 주소, 자동세팅과 사용량 >

> **EEPROM은 100,000번 이상 저장하면 기능을 상실하며, 전원 재투입 시 이상한 값이 읽힐 수 있습니다.** 특히 옵션1(값 변경 시 저장)은 값이 바뀔 때마다 저장하므로, 자주 바뀌는 주소를 유지 영역에 포함하면 저장 횟수가 빠르게 누적됩니다.

3-6 도움말

도움말 메뉴는 버그 신고·개선 요청·업데이트 확인·정보, 4개 항목으로 구성되며 모두 단축키가 없습니다.

항목	단축키	설명
버그 신고	—	문제 상황을 설명해 신고합니다
개선 요청	—	원하는 기능이나 개선 사항을 요청합니다
업데이트 확인	—	새 버전이 있는지 서버에서 확인합니다
정보	—	앱 이름과 버전을 토스트로 표시합니다

버그 신고를 클릭하면 "버그 신고" 창이 뜨며, 문제 상황을 설명하고 [보내기]로 전송합니다. 기본 진단 정보(버전·OS·로그)는 항상 포함되며, 현재 프로젝트(`.mp2`)· 화면 스크린샷·연락처는 선택적으로

첨부할 수 있습니다. 기본 진단 정보 체크박스는 **회색으로 잠겨 있어 끌 수 없고 항상 켜진 상태로** 표시되며, 나머지 세 항목 (프로젝트·스크린샷·연락처)만 켜고 끌 수 있습니다.

개선 요청을 클릭하면 "개선 요청사항 보내기" 창이 뜨며, 원하는 기능이나 개선 사항을 입력하고 화면 스크린샷·연락처를 선택적으로 첨부해 [보내기]로 전송합니다. 버그 신고 창과 달리 개선 요청 창에는 **진단 정보 항목이 없고**, 화면 스크린샷·연락처만 선택적으로 첨부합니다.

업데이트 확인을 클릭하면 곧바로 업데이트 서버에 확인 요청을 보냅니다. 이미 최신 버전이면 "최신 버전을 사용 중입니다." 안내 토스트가, 서버 연결에 실패하면 "업데이트 서버에 연결할 수 없습니다." 오류 토스트가 뜨며, 새 버전이 있으면 버전 정보와 [지금 업데이트]/[나중에] 버튼이 있는 업데이트 창이 뜹니다.

앱은 이 메뉴와 별개로 **실행할 때마다 한 번 자동으로 업데이트를 확인**합니다. 자동 확인은 새 버전이 있을 때만 업데이트 창을 띄우고, 최신 버전이거나 서버에 연결하지 못하면 아무 안내 없이 조용히 넘어갑니다 — 토스트가 항상 뜨는 쪽은 이 메뉴로 직접 확인했을 때뿐입니다.

정보를 클릭하면 "MPINO Studio 2 · v2.2.0" 안내 토스트가 뜹니다.



< 그림 3-19 도움말 메뉴 — 버그 신고부터 정보까지 4개 항목이 표시된 드롭다운 >

4 래더 그리드 편집

래더 그리드를 실제로 편집하는 방법은 크게 두 갈래로 나뉩니다. 하나는 셀을 우클릭해 여는 **컨텍스트 메뉴**로, 모니터링 켜기·표시 형식 전환·아두이노에 값 쓰기 같은 명령과 셀·줄 편집 명령이 한곳에 모여 있는 진입점입니다. 다른 하나는 셀 삽입·삭제, 줄 추가·삭제처럼 그리드의 칸 배치 자체를 바꾸는 조작입니다. 이 장은 이 두 축을 차례로 설명하되, 그 앞에 모든 편집 명령이 공통으로 딛고 서는 **커서 이동과 선택**을 먼저 다룹니다. **접점**(5장)이나 **박스함수**(6장)를 배치하기 전에 먼저 읽어두면, 셀을 다루는 기본 조작을 미리 익힐 수 있습니다.

4-1 그리드 하단 상태표시줄

래더 그리드 캔버스 바로 아래에 있는 상태표시줄은 지금 선택한 셀의 좌표와 종류를 실시간으로 보여줍니다. 화면 맨 아래 포트 선택용 상태바와는 별개로, 래더 편집기 안에서만 나타나는 표시줄입니다. 우클릭 메뉴나 단축키를 실행하기 전에 어떤 셀이 선택돼 있는지 이 줄로 항상 확인할 수 있습니다.

좌표는 `R{행} C{열}` 형식으로 표시되며, 행 번호는 1부터, 열 번호는 0부터 셉니다. 그리드 맨 위 왼쪽 첫 셀을 선택하면 상태표시줄에는 `R1 C0`가 나타납니다 — 1장 튜토리얼이 같은 자리를 "1행 1열"(행·열 모두 1부터)로 부르는 것과 달리, 상태표시줄의 열 번호는 0부터 시작한다는 점에 유의하세요.

선택한 셀에 요소가 있으면 좌표 뒤에 셀 종류가 이어 붙습니다. 종류는 앱 내부에서 쓰는 이름 그대로 표시되며, 접점은 `ContactA`(A접점)와 `ContactB`(B접점), 코일은 `OutCoil`·`SetCoil`·`ResetCoil`, 박스함수는 `Func`으로 나타납니다. 그 외에 엣지 검출 접점(`RisingEdge`·`FallingEdge`), 신호 반전(`Reverse`), 가로 연결선(`LineH`)도 각각 자기 이름으로 표시됩니다. F6(수직)으로 놓는 세로 연결선은 별도의 셀 종류가 아니라 기존 셀에 딸린 속성(link_to_below)이라, F6을 적용해도 상태표시줄의 종류 표시는 그 셀 원래의 종류(빈 셀이면 `None`, 접점이면 `ContactA` 등) 그대로 유지됩니다 — `LineV`라는 종류가 따로 뜨는 일은 없습니다. 메모리 주소나 함수 코드가 입력된 셀은 종류 뒤에 그 코드가 한 번 더 붙어 `R1 C0 · ContactA · P0`처럼 보입니다.

빈 셀을 선택하면, 즉 그 좌표에 해당하는 셀이 배열에 아예 없으면 종류 자리에 "빈 셀"이 붙어 `R{행} C{열} · 빈 셀`로 표시됩니다. 다만 Delete로 비운 셀처럼 셀은 배열에 그대로 남아 있으나 어떤 도구도 적용되지 않은 경우는 내부 타입 이름 그대로 `R{행} C{열} · None`으로 표시됩니다 — 그리드에는 두 경우 모두 똑같이 빈 칸으로 보이지만, 상태표시줄에서는 이렇게 구분됩니다. 아무 셀도 선택하지 않은 상태에서는 종류 대신 "셀을 클릭하거나 방향키로 이동, Enter로 편집, F-key로 도구, Delete로 비우기." 조작 힌트가 표시됩니다.

4-2 커서 이동과 선택

래더 그리드의 거의 모든 명령은 "지금 커서가 놓인 셀"을 대상으로 합니다. 그래서 커서를 어떻게 옮기고 어디까지 한꺼번에 고르는지가 편집의 출발점입니다. 그리드에 포커스가 있을 때 방향키(↑·↓·←·→)로 한 칸씩 커서를 옮기고, **Enter**로 그 셀의 편집 창을 열며, **F키**로 도구를 적용하고, **Delete**로 셀을 비웁니다 — 상태표시줄이 아무것도 선택되지 않았을 때 보여주는 힌트와 같은 조작입니다.

Shift+Enter는 같은 편집 창을 **심볼로** 엽니다. 입력칸에 주소 대신 그 셀의 현재 심볼 이름이 들어 있고(심볼이 없으면 빈 칸), 후보 목록에 심볼 이름과 주소가 함께 나옵니다. 다른 심볼을 고르고 확정하면 셀이 그 심볼의 주소를 가리키게 됩니다(심볼 표는 바뀌지 않습니다). 박스함수는 커서를 둔 인자 칸을 이렇게 바꿉니다. 비교식처럼 심볼로 바꿀 수 없는 셀은 Enter와 같은 편집 창이 열립니다.

4-2-1 박스함수 위에서의 커서

박스함수(6장)는 화면에서 **여러 칸을 차지**합니다. 명령어 띠 아래로 인자 하나가 한 칸씩 자리를 잡고, 오른쪽 빈 칸이나 왼쪽 가로선 위로 몸통이 넓어집니다. 그러나 래더 데이터상으로 박스함수는 여전히 **셀 하나**입니다. 그래서 박스가 차지한 어느 칸을 골라도 F키·Delete·Enter(편집)·우클릭·툴바 도구는 모두 **박스 전체**를 대상으로 동작합니다. 몸통에 덮여 보이지 않는 가로선만 따로 지워지는 일은 없습니다.

다만 방향키 ←/→는 예외로, 박스 **안에서는 인자 칸을 하나씩** 옮깁니다.

- 박스 밖에서 →로 들어오면 **첫 인자** 칸에, ←로 들어오면 **마지막 인자** 칸에 커서가 놓입니다(누르던 방향을 그대로 이어갑니다).
- 박스 안에서 ←/→를 누르면 인자 칸이 하나씩 옮겨가고, **끝 인자에서 한 번 더** 누르면 그때 박스 밖 다음 칸으로 나갑니다.
- ↑/↓는 커서가 **화면에서 서 있는 열**을 지킨 채 위아래로 한 칸 움직입니다. 도착한 자리가 다른 박스에 덮여 있으면 그 박스의 같은 열이 가리키는 인자 칸으로 들어갑니다.
- 그리드 끝에서는 제자리에 머물고, 인자 칸 커서도 그대로 유지됩니다.

커서가 놓인 인자 칸은 셀 커서와 같은 색의 **테두리와 옅은 채움**으로 표시됩니다(쓰기 대상 칸의 음영은 채움만 있어 서로 구분됩니다). 그 상태에서 Enter를 누르거나 더블클릭하면 박스 전체가 아니라 **그 인자 글자만 선택된 채** 편집 창이 열려, 인자 하나만 바로 고쳐 쓸 수 있습니다.

> **인자 칸 커서가 생기지 않는 박스도 있습니다.** `D0 = D0 + 1` 같은 **대입식**(6-9) 박스는 칸을 나누지 않고 원문을 한 줄로 그리고(같은 자유입력형이라도 `setMotor(100)` 같은 함수 호출은 인자마다 칸이 있어 칸 커서가 생깁니다), 몸통을 넓힐 자리가 모자라 인자 수보다 좁게 **압축**된 박스는 열과 인자가 1:1로 맞지 않습니다. 이 두 경우에는 ←/→가 예전처럼 박스를 **통째로 한 칸처럼 건너뛰며**, Enter로 열면 한 줄 전체가 선택됩니다.

4-2-2 범위 선택

여러 셀을 한꺼번에 고르는 것을 **범위 선택**이라고 합니다. 만드는 방법은 네 가지입니다.

방법	조작	결과
Shift+방향키	셀을 하나 고른 뒤 Shift를 누른 채 방향키	처음 셀을 기준점으로 직사각형이 한 칸씩 넓어집니다(래더 그리드에 포커스가 있을 때만)
Shift+클릭	셀을 하나 고른 뒤 Shift를 누른 채 다른 셀 클릭	기준점과 클릭한 셀을 양 끝으로 하는 직사각형
Ctrl+클릭	Ctrl을 누른 채 셀 클릭	그 셀 하나만 선택에 넣거나 뺍니다(토글) — 떨어져 있는 셀들을 모을 때
드래그	셀에서 마우스 버튼을 누른 채 끌기	끄는 동안 사각형 테두리가 따라오고, 그 안의 셀이 선택됩니다

범위 선택의 단위는 **셀**이지 박스함수의 인자 칸이 아닙니다. 그래서 Shift+방향키는 박스를 한 칸처럼 건너뛰고, 사각형이 박스에 걸리면 **박스함수 셀과 그 왼쪽의 덮인 가로선이 한 덩어리로** 선택에 들어갑니다. Ctrl+클릭으로 뺄 때도 마찬가지로 한 덩어리가 함께 빠집니다 — 몸통에 가려 보이지 않는 가로선만 선택에 남아 전원 경로가 조용히 끊기는 일을 막기 위해서입니다.

범위를 고른 뒤 쓸 수 있는 명령은 다음과 같습니다.

- **Delete** — 선택한 셀을 모두 한 번에 비웁니다(Ctrl+Z로 한 번에 되돌아갑니다).
- **Ctrl+C** / **Ctrl+X** / **Ctrl+V** — 아래 **클립보드와 실행취소** 참고.
- **F키·`\` 도구** — 여러 셀에 한꺼번에 적용되지 않습니다. 선택이 **커서가 마지막으로 놓였던 셀 하나**로 좁혀진 뒤 그 셀에만 적용됩니다.

4-3 래더 셀 우클릭 메뉴

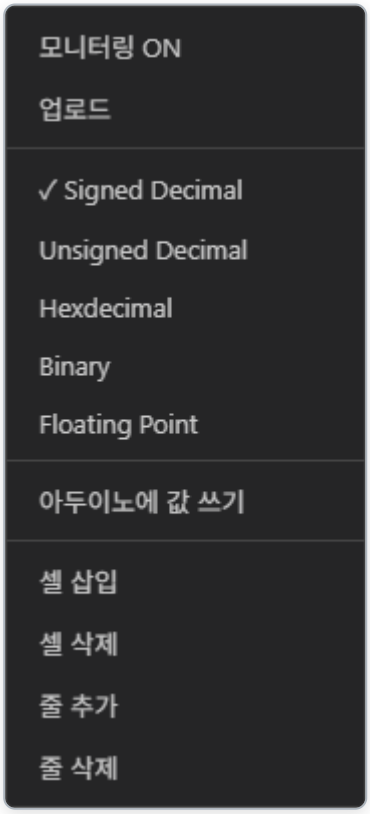
래더 셀을 우클릭하면 그 셀이 먼저 선택된 뒤(선택 표시가 그 셀로 옮겨간 뒤) 컨텍스트 메뉴가 열립니다. 메뉴의 삽입·삭제·표시 형식 명령은 모두 이렇게 새로 선택된 셀을 기준으로 동작하므로, 우클릭한 셀이 어떤 셀인지가 이어지는 명령의 결과를 좌우합니다.

메뉴는 접점·코일·박스함수·빈 셀 등 셀 종류와 무관하게 항상 같은 12개 항목을 보여 줍니다. 실제 메뉴에는 구분선으로 네 그룹 — ① 모니터링·업로드, ② 표시 형식 5종, ③ 아두이노에 값 쓰기, ④ 그리드 편집 4종 — 이 나뉘어 있으며, 아래 표도 이 그룹 순서 그대로 나열합니다. 셀 타입에 따라 항목이 추가되거나 빠지는 경우는 없고, 동적으로 바뀌는 부분은 1번 항목의 ON/OFF 라벨과 3~7번 항목의 라디오 체크 위치뿐입니다.

항목	설명
모니터링 ON / 모니터링 OFF	래더 실시간 모니터링 통신을 켜고 끕니다. 현재 상태에 따라 라벨이 바뀝니다(꺼져 있으면 "모니터링 ON", 켜져 있으면 "모니터링 OFF") — 상세는 **래더 모니터링 시작하기**(8-5) 참고
업로드	현재 프로젝트를 빌드한 뒤 선택한 포트로 보드에 플래시합니다 — 상세는 **실행**(3-4) 참고

항목	설명
Signed Decimal	모니터링 값을 부호 있는 10진수로 표시합니다(기본값)
Unsigned Decimal	모니터링 값을 부호 없는 10진수로 표시합니다
Hexdecimal	모니터링 값을 16진수로 표시합니다(앱의 실제 표기이며 오타가 아닙니다)
Binary	모니터링 값을 2진수로 표시합니다
Floating Point	모니터링 값을 부동소수점으로 표시합니다
아두이노에 값 쓰기	선택한 셀의 메모리 주소가 채워진 값 쓰기 팝업을 엽니다. 실행 중인 보드의 그 주소 값을 입력한 값으로 곧바로 덮어씁니다 — 상세는 아래 **아두이노에 값 쓰기**
셀 삽입	그리드에 셀 한 칸을 삽입합니다 — 상세는 아래 **셀 삽입과 삭제**
셀 삭제	그리드에서 셀 한 칸을 삭제합니다 — 상세는 아래 **셀 삽입과 삭제**
줄 추가	그리드에 줄 하나를 추가합니다 — 상세는 아래 **줄 추가와 삭제**
줄 삭제	그리드에서 줄 하나를 삭제합니다 — 상세는 아래 **줄 추가와 삭제**

3~7번 표시 형식은 라디오 항목으로, 현재 선택된 형식에 체크 표시가 붙습니다. 다섯 라벨은 **영문 그대로** 메뉴에 표시되며, 각 형식이 실제 값을 어떻게 나타내는지와 시간 단위 표시가 어떻게 결합되는지 같은 자세한 내용은 **표시 형식과 특수 표시** (8-7)를 참고하세요.



< 그림 4-1 래더 셀 우클릭 메뉴 — 12개 항목이 항상 동일하게 나온다 >

4-4 아두이노에 값 쓰기

우클릭 메뉴의 "아두이노에 값 쓰기"(또는 단축키)를 선택하면 값 쓰기 팝업이 열립니다. 커서(선택)가 놓인 셀의 메모리 주소에, 입력한 값을 실행 중인 보드에서 곧바로 덮어씁니다. 같은 팝업은 실행 메뉴의 **아두이노에 값 쓰기**(3-4)로도 열 수 있습니다(이때는 주소 칸이 비어 있어 직접 입력합니다).

쓸 수 있는 조건 — 환경설정 → 래더 설정 → **모니터링 사용 포트**가 **Serial** / **Serial1** / **Serial2** / **Serial3** 중 하나로 설정돼 있고, 그 설정으로 업로드한 프로그램이 보드에서 실행 중이어야 합니다. 포트가 **사용 안 함**이면 메뉴를 눌러도 팝업 대신 설정 경로를 안내하는 창이 뜹니다. 상태바에서 PC 포트(COM)도 골라 두어야 합니다.

여는 법 — 래더 셀 우클릭 후 "아두이노에 값 쓰기"를 선택하거나, 단축키 **Ctrl+Shift+W** 또는 **Alt+Enter**를 누릅니다(단축키는 래더 그리드에 포커스가 있을 때만 동작하며, 셀 편집창이나 입력칸 안에서는 동작하지 않습니다). 모든 경로의 결과는 같으며, 팝업은 항상 커서(선택) 셀 기준으로 열리고 커서는 바로 **값 칸**에 놓입니다 — 값을 입력하고 Enter만 치면 됩니다.

팝업 상단 주소 칸(placeholder `"M0"`)에는 쓸 수 있는 주소가 자동으로 채워집니다. 쓸 수 있는 영역은 **P·M·D·C·T·R**이며, 선택한 셀의 내용에서 자동으로 뽑아냅니다 — 박스함수처럼 주소 후보가 여러 개면 드롭다운에서 고를 수 있고, 접점·코일처럼 주소가 하나면 그 주소가, 빈 셀이면 빈 칸이 채워져 직접 입력합니다. 잘못된 주소를 넣으면 "…는 쓸 수 없는 주소입니다 (P/M/D/C/T/R 만)" 안내가 인라인으로 뜹니다.

P(입출력) 쓰기 — 입력 핀에 연결된 P는 쓸 수 없습니다("입력 핀은 쓰기할 수 없음"). 써도 보드가 다음 순간 실제 입력 값으로 다시 덮어쓰기 때문입니다. 출력 핀에 연결된 P와 어느 핀에도 연결되지 않은 P는 쓸 수 있습니다 — 출력 P에 1을 쓰면 실제 출력이 켜집니다. 다만 래더의 코일이 그 출력을 구동하고 있으면 다음 스캔에 코일 값으로 돌아갑니다. 이 기능이 생기기 전에 업로드한 프로그램에는 P 쓰기가 없으므로 "보드의 프로그램이 P 쓰기를 지원하지 않습니다. 다시 업로드한 뒤 시도하세요."가 뜹니다(M/D/C/T/R 쓰기는 그대로 됩니다).

값 칸(placeholder `"쓸 값"`)에 값을 입력하고 **쓰기** 버튼을 누르거나 Enter를 치면 그 자리에서 보드에 값이 써집니다. 값의 형식은 영역의 자료형에 맞게 해석됩니다. 쓰기에 성공하면 팝업은 열린 채로 값 칸만 비워져 같은 주소나 다른 주소를 연달아 쓸 수 있습니다.

모니터링이 켜져 있을 때와 꺼져 있을 때 — 쓰기는 래더 모니터링을 켜 둔 상태와 꺼 둔 상태 모두에서 됩니다.

- **켜져 있을 때**: 열려 있는 모니터링 연결로 바로 보내고, 팝업에 현재값이 실시간으로 표시됩니다.
- **꺼져 있을 때**: 그 순간에만 포트를 열어 보드의 모니터링 응답을 먼저 확인하고, 값을 보내고, 바뀐 값을 확인한 뒤 포트를 닫습니다. 포트를 열 때 보드는 리셋되지 않으므로 실행 중인 프로그램은 멈추지 않습니다. 다른 프로그램(시리얼 모니터·업로드 등)이 그 포트를 쓰고 있으면 "… 를 열 수 없습니다"가 뜨고 쓰지 않습니다.

결과는 보드가 보내는 다음 모니터링 값으로 확인합니다(Enter 후 보통 0.3초 안팎). 값이 그대로 보이면 성공 안내가, 다른 값이 보이면 "보냈지만 현재값은 X 입니다. 래더가 같은 주소를 덮어쓰고 있을 수 있습니다."가, 1초 안에 응답이 없으면 "보냈지만 보드 응답으로 확인하지 못했습니다."가 뜹니

다. 보드에서 모니터링 응답이 전혀 없으면 (모니터링 없이 업로드했거나 케이블·포트가 다른 경우) 아무것도 보내지 않고 안내만 표시합니다. 업로드 중에는 쓸 수 없습니다.

> **Serial3**은 ATmega2560 보드에만 있는 TTL 레벨 포트(D14 TXD3 / D15 RXD3)입니다. 모니터링 포트를 Serial3으로 쓰려면 PC와 보드 사이에 **USB-TTL 변환기**가 필요합니다.

단축키로 비트 바로 쓰기 — 모니터링 중에 래더 셀을 선택하고 **Ctrl+1**을 누르면 그 셀의 비트(P, M)에 1을, **Ctrl+0**을 누르면 0을 씁니다. 팝업을 거치지 않고 곧바로 쓰며, 결과 안내는 팝업과 같습니다. 입력 핀 P는 쓸 수 없습니다. 접점·코일·엣지처럼 주소가 P 또는 M 하나인 셀에서만 되고, 그 밖의 셀(T/C/D/R·박스함수·비교식·빈 셀)에서는 쓰지 않고 "비트 주소(P, M)가 없음"이라고 안내합니다. 모니터링이 꺼져 있으면 쓰지 않고 안내만 합니다. 셀 편집 창·모달·코드 에디터에 입력 중일 때는 동작하지 않습니다.

닫는 법 — 팝업은 **Esc**, 팝업 바깥 클릭, 헤더의 ✕ 버튼으로 닫습니다. 다른 설정 창과 달리 배경을 덮지 않는 **모달리스** 창이라 열린 채로도 래더의 다른 부분을 볼 수 있고, 헤더를 드래그해 위치를 옮길 수 있습니다.

이름이 비슷한 실행 메뉴의 **메모리 쓰기 목록**(3-4-2)과 혼동하기 쉬운데, 둘은 다음과 같이 다릅니다.

항목	아두이노에 값 쓰기(즉석 팝업)	메모리 쓰기 목록(등록형 창)
여는 법	래더 셀 우클릭 · Ctrl+Shift+W · Alt+Enter · 실행 메뉴	실행 메뉴의 "메모리 쓰기 목록"
성격	커서 셀에 붙는 단발 팝업	배경을 덮지 않는 관찰 목록 창
주소	선택 셀에서 자동 추출(직접 수정 가능)	사용자가 행으로 등록
대상 영역	P/M/D/C/T/R	P/M/D/C/T/R
부가 기능	없음(한 번에 하나)	가져오기·내보내기·지우기·전체 쓰기

둘 다 실행 중인 보드에 곧바로 값을 쓰며(모니터링을 켜지 않아도 됨), 같은 주소 규칙(P/M/D/C/T/R, 입력 핀 P 제외)을 씁니다. 목록 창 쪽은 **메모리 쓰기 목록**(3-4-2)을 참고하세요.



< 그림 4-2 아두이노에 값 쓰기 팝업 — 선택 셀의 주소가 자동으로 채워지고 값 칸에 커서가 놓인 창 >

4-5 셀 삽입과 삭제

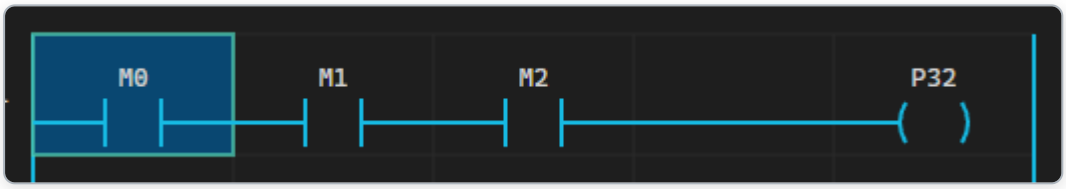
셀 하나를 삽입하거나 삭제해 그리드의 칸 배치를 조정합니다. 두 조작 모두 커서가 놓인 셀(또는 우클릭으로 새로 선택된 셀)을 기준으로 동작하며, 단축키와 우클릭 메뉴 어느 쪽으로 실행해도 결과는

같습니다.

셀 삽입 — 단축키 **Insert** 또는 우클릭 메뉴의 "셀 삽입"

커서가 있는 칸부터 오른쪽으로 한 칸씩 밀어냅니다. 다만 무조건 행 전체를 밀어내는 것이 아니라, 각 행에서 **가장 가까운 빈자리**(빈칸 또는 빈 가로선)까지만 밀어 그 자리를 흡수합니다. 행 끝에 코일이 있으면 코일은 밀리지 않고 제자리를 지키고, 접점과 코일 사이의 세로 연결선은 정렬이 흐트러지지 않도록 위·아래 칸이 함께 이동합니다. 코일 앞에는 흡수할 빈자리가 먼저 잡히기 때문인데, **박스함수도 마찬가지로** 밀리지 않고 제자리를 지킵니다. 그리드의 마지막 열에서 셀 삽입을 실행하면 더 밀어낼 자리가 없으므로 그리드는 바뀌지 않고, 아래와 같은 안내 토스트가 뜹니다.

> 그리드의 마지막 열에서 실행하거나, 영향을 받는 행 중 단 한 행이라도 오른쪽에 흡수할 빈자리가 없으면(오른쪽 끝까지 빈틈없이 차 있으면) 그리드 전체가 바뀌지 않고 "더 이상 밀 수 없습니다 — 오른쪽 끝이 찼습니다." 안내 토스트가 뜹니다. 일부 행만 밀리고 나머지는 그대로 남는 일은 없습니다.



< 그림 4-3 셀 삽입 전 — 커서가 놓인 셀 >



< 그림 4-4 셀 삽입 후 — 오른쪽 셀들이 흡수 가능한 만큼 밀려남 >

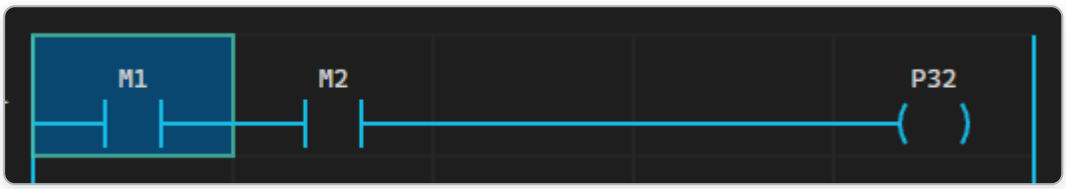
셀 삭제 — 단축키 **Ctrl+Delete** 또는 우클릭 메뉴의 "셀 삭제"

커서가 있는 셀을 지우고, 그 오른쪽 셀들을 한 칸씩 왼쪽으로 당겨 빈자리를 메웁니다. 셀 삽입과 마찬가지로 행 끝 코일은 당겨지지 않고 제자리에 남으며, 코일 앞에 생기는 빈자리는 가로선으로 이어집니다. 코일이 있는 칸 자체를 커서에 놓고 셀 삭제를 실행하면 코일은 삭제되지 않고 아무 일도 일어나지 않습니다. 이때 행의 종착점(sink)으로 취급되는 것은 출력 코일·SET 코일·RESET 코일 **그리고 박스함수** 네 가지라, 행 끝이 박스함수여도 똑같이 지켜집니다.

> **세로 연결선의 양 끝이 모두 접점이면 당기지 않습니다.** 두 줄짜리 세로 연결선으로 묶인 병렬 가지의 위·아래 끝이 **둘 다 접점**일 때, 그중 하나를 셀 삭제하면 오른쪽을 당기는 대신 그 접점만 **가로선**으로 제자리에서 바뀝니다 — 세로 연결선은 그대로 남아 병렬 가지가 끊기지 않습니다. 끝점 한쪽이 빈칸이거나 가로선이면 의미 있는 병렬 가지로 보지 않아 평소대로 당깁니다.



< 그림 4-5 셀 삭제 전 — 커서가 놓인 셀 >



< 그림 4-6 셀 삭제 후 — 오른쪽 셀들이 왼쪽으로 당겨짐 >

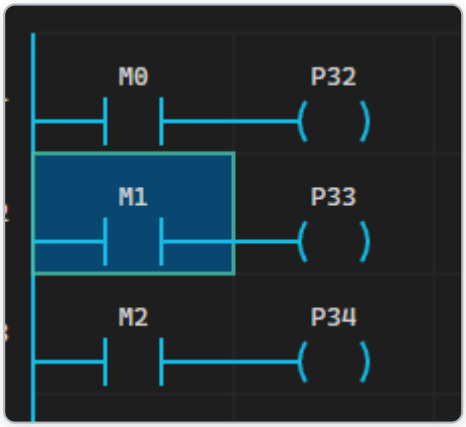
셀 삽입·삭제 모두 실수했다면 **Ctrl+Z**로 되돌릴 수 있습니다 — 자세한 내용은 아래 **클립보드와 실행 취소**를 참고하세요.

4-6 줄 추가와 삭제

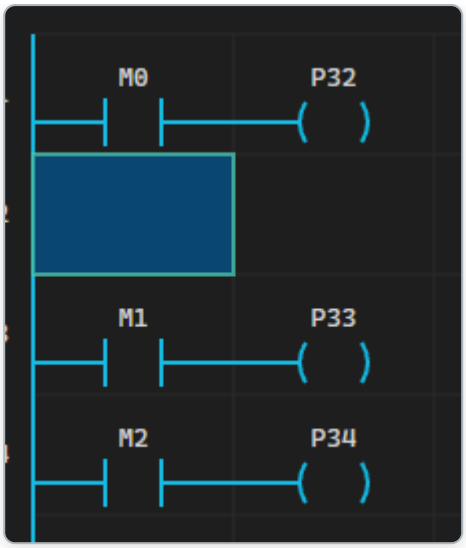
줄 단위로 그리드에 빈 줄을 넣거나 기존 줄을 통째로 지웁니다.

줄 추가 — 단축키 **Alt+Insert** 또는 우클릭 메뉴의 "줄 추가"

커서가 있는 줄 **위**에 빈 줄 하나를 삽입합니다. 이때 그 위치를 관통하던 세로 연결선은 끊기지 않고 자동으로 새 빈 줄까지 이어집니다.



< 그림 4-7 줄 추가 전 — 커서가 놓인 줄 >

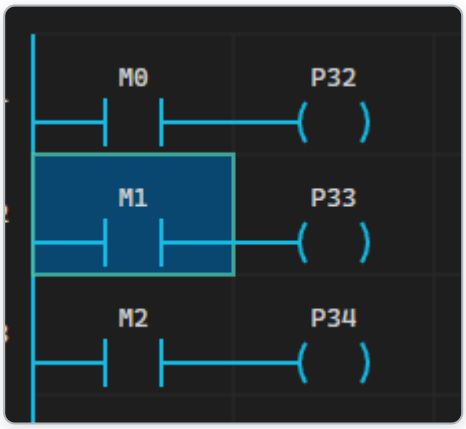


< 그림 4-8 줄 추가 후 — 커서 줄 위에 빈 줄이 삽입됨 >

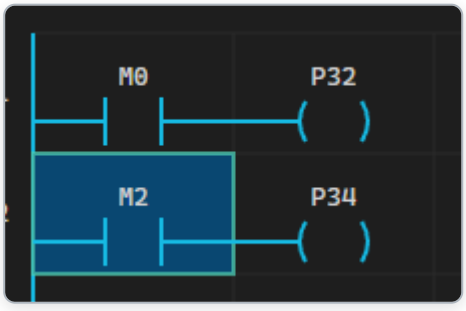
줄 삭제 — 단축키 **Alt+Delete** 또는 우클릭 메뉴의 "줄 삭제"

커서가 있는 줄을 통째로 지우고, 그 아래 줄들을 한 칸씩 위로 당깁니다. 삭제된 줄을 관통하던 세로 연결선은 자동으로 다시 이어집니다.

> 줄이 하나만 남았을 때 줄 삭제를 실행하면 래더가 비워집니다. 실수했다면 Ctrl+Z로 되돌리세요.



< 그림 4-9 줄 삭제 전 — 커서가 놓인 줄 >



< 그림 4-10 줄 삭제 후 — 아래 줄들이 위로 당겨짐 >

줄 추가·삭제 모두 실수했다면 **Ctrl+Z**로 되돌릴 수 있습니다 — 자세한 내용은 아래 **클립보드와 실행취소**를 참고하세요.

4-7 클립보드와 실행취소

셀 복사·잘라내기·붙여넣기 — 단축키 **Ctrl+C / Ctrl+X / Ctrl+V**

선택한 셀(또는 **범위 선택**(4-2-2)한 셀들)을 복사·잘라내기·붙여넣기 합니다. 이 단축키는 래더 그리드에 포커스가 있을 때만 위 동작을 하며, 코드 에디터에 포커스가 있으면 일반 텍스트 편집 단축키로 동작합니다.

복사·잘라내기한 셀은 시스템 클립보드에도 함께 담기므로, 같은 프로젝트의 다른 래더 탭뿐 아니라 **MPINO Studio 2를 두 개 띄운 창 사이에서도** 붙여넣을 수 있습니다. 다른 창에서 복사한 내용이 있으면 그것이 먼저 붙습니다.

박스함수 인자 값만 복사·붙여넣기

박스함수 셀 하나만 선택하고 커서를 **값이 들어 있는 인자 칸**에 둔 채 Ctrl+C를 누르면, 셀과 **함께 그 인자 글자**도 담깁니다. 단축키는 그대로 Ctrl+C / Ctrl+V 하나이며, 어느 쪽으로 붙을지는 **붙여넣는 자리**가 정합니다. 다만 인자 글자는 **이 창 안에만** 보관되므로, 다른 창에서 복사해 온 것을 붙여넣을 때는 언제나 박스 전체가 붙습니다.

- 붙여넣는 자리가 **다른 박스함수의 인자 칸**이면 → 그 칸의 값만 바뀝니다. 셀 종류도 나머지 인자도 그대로이고, 인자 칸 커서도 그 자리에 남아 연달아 붙여넣을 수 있습니다.
- 붙여넣는 자리가 **빈 칸이나 보통 셀**이면 → 예전처럼 박스함수 셀 전체가 붙습니다.

값만 붙여넣기를 받는 것은 `MOV D0 D1` 같은 **정형 명령어**와 `setMotor(100)` 같은 **함수 호출** 박스뿐입니다. 대입식(`D0 = D0 + 1`)이나 라벨처럼 인자 칸을 가려낼 수 없는 박스는 대상이 아니며, 이때는 토스트 없이 박스 전체가 붙는 셀 붙여넣기로 넘어갑니다.

메모장이나 코드 편집기처럼 **앱 바깥에서 복사한 일반 텍스트**도 인자 칸에 붙여넣을 수 있습니다. 이때는 **내용이 있는 첫 줄**만 쓰고 앞뒤 공백을 떼어 넣습니다 — 내용이 있는 줄이 둘 이상이면 버린 줄이 있다는 안내가 함께 뜹니다(줄 끝 개행 하나만 딸려 온 흔한 경우에는 뜨지 않습니다).

> **박스함수 셀을 붙여넣으면 붙여넣은 칸이 언제나 첫 인자 자리가 됩니다.** 빈 칸에 붙이든 가로선이 이어진 줄기 한가운데에 붙이든 결과가 같도록, 앱이 박스 몸통을 커서 칸에서 오른쪽으로 펼치거나(오른쪽이 비어 있을 때) 커서 칸이 첫 인자가 되도록 자리를 맞춰 놓습니다.

붙여넣기가 어긋난 자리에 조용히 놓이는 일은 없습니다. 아래 경우에는 안내 토스트가 뜹니다.

상황	안내 문구
박스 글자와 인자 분해가 어긋나 칸의 글자 범위를 찾지 못함	"이 박스함수는 인자 자리를 가려낼 수 없어 값만 붙여넣을 수 없습니다. Enter로 박스를

상황	안내 문구
	열어 직접 수정하세요."
여러 줄 중 첫 줄만 넣었음(붙여넣기는 성공)	"여러 줄이 복사되어 있어 첫 줄만 인자에 붙여넣었습니다."
박스 몸통이 들어갈 자리가 모자람(붙지 않음)	"박스함수를 놓을 자리가 모자랍니다 — 커서 칸부터 오른쪽으로 N칸이 필요합니다. 왼쪽 칸이나 빈 줄에 붙여넣으세요."(N = 필요한 칸 수)

실행취소 — 단축키 **Ctrl+Z**

셀 삽입·삭제, 줄 추가·삭제, 붙여넣기 등 그리드를 바꾸는 조작을 모두 되돌립니다 (단, 아무것도 바뀌지 않은 무동작 경로 — 예: 오른쪽 끝이 찬 셀 삽입 — 는 되돌릴 대상 자체가 없습니다). 편집 중 실수했다면 주저 없이 Ctrl+Z를 누르세요.

재실행 — 단축키 **Ctrl+Y**

실행취소로 되돌린 편집을 다시 적용합니다. 너무 많이 되돌렸다면 Ctrl+Y로 한 단계씩 되살리면 됩니다. 실행취소 뒤에 새 편집을 하면 되돌려 둔 편집은 재실행 대상에서 사라집니다.

이 단축키들은 2장 단축키표(래더 클립보드·편집)에도 정리돼 있습니다 — 상세는 **단축키**(2-3) 참고하세요.

5 접점

5-1 접점 종류

래더 그리드에 배치할 수 있는 접점·라인·코일 계열 셀 도구는 총 10종입니다. 왼쪽 툴바에 F-key 순서대로 세로로 배치돼 있으며, 아래 표의 "이름" 열은 앱이 실제로 표시하는 라벨입니다.

기호	이름	설명	단축키
┤ ├	NO 접점	평상시(조건 비트 OFF)에는 열려 있어 통전되지 않다가, 조건 비트가 ON이 되면 닫혀 통전되는 접점입니다	F2
┤/├	NC 접점	평상시(조건 비트 OFF)에는 닫혀 있어 통전되다가, 조건 비트가 ON이 되면 열려 통전이 끊기는 접점입니다	F3
──	직선	셀과 셀을 가로로 잇는 배선입니다	F5
|	수직	셀 타입이 아니라, 선택한 셀 오른쪽에 위–아래 방향 연결선을 켜고 끄는 토글입니다. 이미 놓인 접점·코일은 그대로 유지되며, 박스함수 셀에는 걸 수 없습니다	F6
−(S)−	출력 코일 SET	조건이 ON되는 순간 출력 비트를 켜고, 이후 조건이 OFF돼도 값을 그대로 유지합니다(래치)	F7
−(R)−	출력 코일 RESET	조건이 ON되는 순간 출력 비트를 끄고, 이후 조건이 OFF돼도 값을 그대로 유지합니다(래치)	F8
−( )−	출력 코일	조건 상태를 매 스캔 그대로 출력 비트에 반영합니다(조건 ON=출력 ON, 조건 OFF=출력 OFF)	F9
−/−	신호 반전	그 rung까지 누적된 조건을 반전(NOT)시켜 코일로 넘깁니다	F10
┤↑├	상승엣지	조건 비트가 OFF에서 ON으로 바뀌는 한 스캔만 통전되는 접점입니다	F11
┤↓├	하강엣지	조건 비트가 ON에서 OFF로 바뀌는 한 스캔만 통전되는 접점입니다	F12

셀에 도구를 적용하는 방법은 세 가지입니다.

- 셀을 선택한 뒤 위 표의 단축키(F키)를 누릅니다.
- 셀을 선택한 뒤 왼쪽 툴바에서 원하는 도구 버튼을 클릭합니다.
- 툴바 버튼을 원하는 셀 위로 드래그&드롭합니다(별도 선택 없이도 적용됩니다).

수직(F6)은 세 방법의 결과가 같습니다. 나머지 도구는 **단축키**로 놓을 때 앱이 전원 경로를 이어 주는 일을 더 합니다 — 접점·직선·신호 반전·엣지는 왼쪽 빈 칸을 채우고, **코일 3종과 박스함수**는 놓이는 자리까지 갈립니다.

접점·직선·신호 반전·엣지(F2·F3·F5·F10·F11·F12)를 단축키로 놓으면 커서 칸에 놓이고, 그 왼쪽의 빈 칸이 직선(가로선)으로 채워집니다. 아래 코일 단축키가 코일 앞 빈 칸을 채우는 것과 같은 규칙입니다 — 왼쪽 끝까지 이어 가되 윗 행에서 내려오는 세로 연결선을 만나면 거기서 멈추고, 이미 놓인 접점이나 주소는 그대로 둔 채 그 너머의 빈 칸까지 잇습니다. 코일과 달리 행의 맨 오른쪽으로 옮기지는 않습니다. 실행 취소(Ctrl+Z) 한 번이면 놓은 셀과 채운 직선이 함께 되돌아갑니다. 툴바 버튼 클릭이나 드래그&드롭으로 놓으면 선택한 칸에만 놓이고 왼쪽은 채우지 않습니다.

> **코일 3종(F7·F8·F9)과 박스함수(`\`)는 단축키로 놓으면 커서 칸이 아니라 그 행의 맨 오른쪽 칸에 놓입니다.** 코일은 행의 종착점(sink)이므로, 앱이 커서가 있는 **행**만 쓰고 열은 맨 오른쪽으로 옮긴 뒤 그 앞의 빈 칸을 직선(가로선)으로 채워 전원 경로를 이어 줍니다(윗 행에서 내려오는 세로 연결선을 만나면 거기서 멈춥니다 — 전원이 이미 그 지점으로 들어오기 때문입니다). 이미 놓인 접점이나 주소는 그대로 보존됩니다. 이어서 선택이 그 칸으로 옮겨가고 **편집 창이 자동으로 열립니다.**
>
> 툴바 버튼을 **클릭**하거나 셀 위로 **드래그&드롭**한 경우에는 이 자리 옮김이 일어나지 않습니다 — 선택한(또는 떨어뜨린) 셀 바로 그 자리에 놓이고, 편집 창도 자동으로 열리지 않습니다. 원하는 칸에 코일을 직접 놓고 싶을 때는 툴바 쪽을 쓰세요.

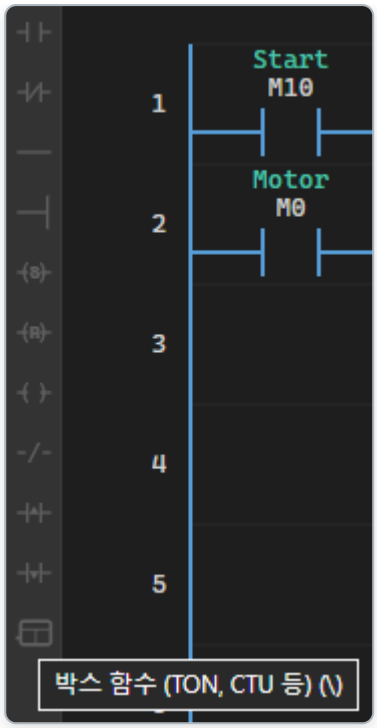
단축키 재정의를 포함한 전체 단축키 목록은 **단축키**(2-3), 셀에 놓은 뒤 주소·심볼을 입력하는 방법은 **핀 편집**(2-4)의 "래더 셀 편집"을 참고하세요.

같은 툴바에는 박스함수(`\`) 도구도 있습니다. 접점·코일과 달리 TON·CTU 같은 기능 블록을 배치하는 용도이며, 자세한 내용은 별도 장에서 다룹니다.



< 그림 5-1 래더 도구 모음 — 접점·라인·코일·검출 접점 10종이 F-key 순서로 놓이고, 맨 아래에 박스함수 버튼이 배치된 툴바 >

툴바 버튼에 마우스를 올려 두면 그 버튼의 이름과 단축키가 툴팁으로 나타나므로, 아이콘만 보고 헷갈릴 때 확인할 수 있습니다.



< 그림 5-2 툴바 버튼에 마우스를 올리면 이름과 단축키가 툴팁으로 표시된다(박스함수 버튼의 예) >

> **예제를 따라 하기 전에 그 예제가 쓰는 P 주소의 핀을 먼저 켜세요.** 이 장의 예제는 출력 `P32` · `P33` · `P35` · `P36`, 입력 `P0` 같은 보드 핀을 씁니다. 새 프로젝트는 모든 핀이 "사용 안 함"으로 시작하므로, 회로만 그리고 빌드하면 문제 탭과 빌드 출력에 "핀 P32 (Dout_1)은(는) 보드 정의에서 비활성화되어 있습니다(usepin = false)..." 같은 오류가 나고 빌드가 멈춥니다. **설정 › 래더 핀맵 설정**(Ctrl+Shift+P)에서 그 주소의 **사용**을 체크하고 [확인]을 누르세요 — 순서와 그림은 **핀 설정 따라 하기**(2-4-3). `M0` · `D0` 같은 내부 메모리는 켤 필요가 없습니다. 보드에 없는 P 주소(예: 출력이 P32~P35뿐인 MPINO-8A4R(T)-S의 `P36`)는 내 보드에 있는 출력으로 바꿔 쓰세요.

5-2 NO 접점과 NC 접점

NO(Normally Open, A접점)와 NC(Normally Closed, B접점)는 접점 중에서도 가장 기본적인 두 종류로, 조건 비트가 꺼져 있을 때(평상시)의 상태가 서로 반대입니다.

5-2-1 NO 접점 (A접점)

NO 접점은 평상시(조건 비트가 OFF일 때)에는 열려 있어 통전되지 않다가, 조건 비트가 ON이 되면 닫혀 통전되는 접점입니다. F2 또는 툴바의 NO 접점 버튼으로 배치합니다.

가장 단순한 형태는 `M0 NO → P32 코일` 입니다. M0가 OFF면 P32도 OFF, M0가 ON이면 P32도 ON이 됩니다. 물리 입력 P0~P15 주소도 동일하게 동작합니다. 예제의 P32와, M0 대신 쓰는 입력 P 주소는 먼저 핀 설정에서 켜 둡니다(2-4-3).

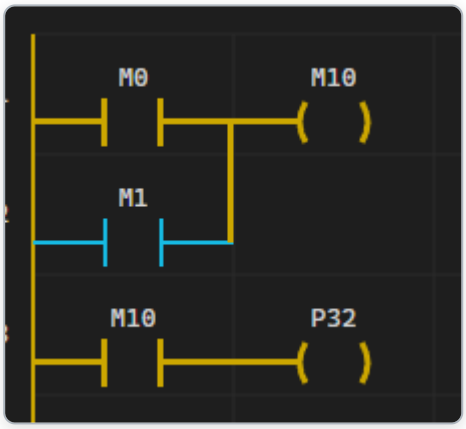


< 그림 5-3 NO 접점 기본 예제 — M0 OFF, P32 OFF(통전 없음) >



< 그림 5-4 NO 접점 기본 예제 — M0 ON, P32 ON(통전 강조) >

여러 접점을 병렬로 배치하면 OR로 동작합니다. `M0 NO`와 `M1 NO`를 병렬로 놓고 결과를 `M10 코일`로 받으면(`M0 ‖ M1 → M10 코일`), 둘 중 하나만 ON이어도 M10이 ON됩니다. 코일이 켜지면 같은 주소를 참조하는 접점(보조접점)도 함께 닫히므로, 다음 rung에 `M10 NO → P32 코일`을 이어두면 M10이 ON일 때 P32도 켜집니다.



< 그림 5-5 NO 접점 병렬 OR 예제 — M0 또는 M1이 ON이면 M10을 거쳐 P32가 켜진다 >

5-2-2 NC 접점 (B접점)

NC 접점은 평상시(조건 비트가 OFF일 때)에는 닫혀 있어 통전되다가, 조건 비트가 ON이 되면 열려 통전이 끊기는 접점입니다 — NO와 정반대로 동작합니다. F3 또는 툴바의 NC 접점 버튼으로 배치합니다.

`M0 NC → P32 코일`로 놓으면, M0가 OFF일 때 P32는 ON(통전)이고, M0가 ON이 되면 P32는 OFF로 바뀝니다. 물리 입력 P0~P15 주소도 동일하게 동작합니다.



< 그림 5-6 NC 접점 예제 — M0 OFF, P32 ON(통전) >



< 그림 5-7 NC 접점 예제 — M0 ON, P32 OFF(통전 끊김) >

모니터링 화면에서는 통전 중인 접점·코일이 강조 색으로 표시됩니다(모니터링 화면 구성 자체는 이 챕터의 범위 밖입니다).

5-2-3 상수 접점 (@ON / @OFF)

NO·NC 접점의 주소 자리에는 실제 메모리 주소 대신 **항상 참** 또는 **항상 거짓**인 상수를 넣을 수 있습니다. 조건 없이 매 스캔 실행하고 싶은 rung의 머리에 두거나, 회로의 한 갈래를 손대지 않고 임시로 끊어 둘 때 씁니다.

입력	의미
`@ON` 또는 `TRUE`	항상 참 — 상시 통전 소스
`@OFF` 또는 `FALSE`	항상 거짓 — 상시 차단 소스

`@ON`과 `TRUE`는 완전히 같은 뜻이고, `@OFF`와 `FALSE`도 같습니다. **대소문자를 가리지 않으므로** `@on` · `true` · `False`처럼 써도 그대로 인식되며, 셀에는 입력한 글자가 바뀌지 않고 그대로 남습니다.

NC 접점에 넣으면 NC의 성질대로 **반전**됩니다. 따라서 네 조합의 결과는 다음과 같습니다.

배치	결과
NO 접점 + `@ON`	상시 통전
NO 접점 + `@OFF`	상시 차단
NC 접점 + `@ON`	상시 차단
NC 접점 + `@OFF`	상시 통전

모니터링 중에도 상수 접점은 보드에서 값을 받아 오지 않고 위 표대로 늘 같은 상태로 표시됩니다.

상수 접점(`@ON`·`TRUE`·`@OFF`·`FALSE`)은 NO 접점과 NC 접점에서만 쓸 수 있습니다. 상승/하강 검출 접점(5-5)의 입력 자리나 코일의 대상 주소 자리에 상수를 넣으면 빌드 시 거부됩니다. 값이 절대 바뀌지 않는 상수에는 검출할 엣지가 없고, 코일 자리는 값을 **기록할** 주소를 요구하기 때문입니다.

이름이 비슷한 **펄스 특수 접점** `@<ms>`·`@F<ms>`(**특수 메모리**(2-2))와는 별개의 기능입니다. `@100`은 100밀리초마다 한 스캔만 ON되는 one-shot 펄스, `@F100`은 100밀리초 간격으로 ON/OFF를 되풀이하는 펄스이고, `@ON`은 값이 바뀌지 않는 상수입니다.

5-3 코일 출력

코일은 rung의 맨 오른쪽에 놓이는 종착점(sink)으로, 그 rung까지 누적된 조건의 결과를 출력 비트에 기록합니다. 기록 방식이 다른 코일 3종(출력 코일 / SET 코일 / RESET 코일)을 제공하며, 마지막 절에서는 세 코일에 공통으로 해당하는 입력핀 대상 사용법을 다룹니다.

5-3-1 출력 코일

출력 코일은 조건이 ON이면 출력 비트를 ON으로, OFF면 다시 OFF로 — 매 스캔마다 조건 상태를 그대로 출력 비트에 반영합니다. 조건이 꺼지면 값도 함께 꺼진다는 점에서 아래 SET·RESET 코일과 다릅니다. F9 또는 툴바의 출력 코일 버튼으로 배치합니다.

예제 회로는 NO 접점(5-2-1)에서 쓴 것과 같은 `M0 NO → P32 코일`입니다. M0가 ON인 동안만 P32가 ON이고, M0가 OFF로 돌아가면 P32도 즉시 OFF로 돌아갑니다. 이 절의 예제도 코일 대상 P32를 먼저 핀 설정에서 켜 둡니다(2-4-3).



< 그림 5-8 출력 코일 예제 — M0 OFF, P32 OFF >



< 그림 5-9 출력 코일 예제 — M0 ON, P32 ON >

5-3-2 SET 코일

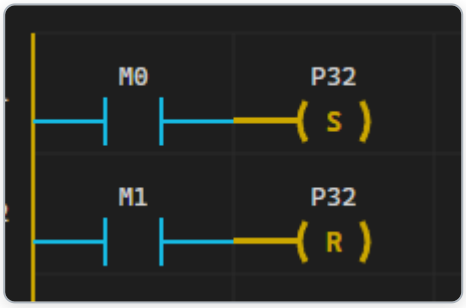
SET 코일은 조건이 ON이 되는 순간 출력 비트를 ON으로 기록하고, 그 뒤 조건이 다시 OFF로 돌아가도 값을 그대로 유지합니다(래치). F7 또는 툴바의 SET 코일 버튼으로 배치합니다.

> **SET 코일의 조건이 OFF로 돌아가도 출력 비트는 OFF되지 않습니다.** 값을 끄려면 같은 주소를 대상으로 하는 RESET 코일이 별도로 필요합니다(아래 RESET 코일 절 참고).

`M0 NO → P32 SET`로 놓으면, M0가 ON되는 순간 P32가 ON으로 기록됩니다. 이후 M0가 OFF로 돌아가도 P32는 계속 ON을 유지합니다.



< 그림 5-10 SET 코일 예제 — M0 ON, P32 ON으로 기록 >

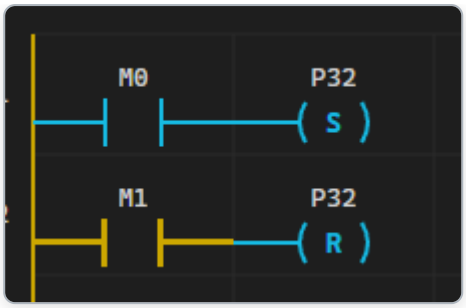


< 그림 5-11 SET 코일 예제 — M0 OFF로 복귀해도 P32는 ON 유지 >

5-3-3 RESET 코일

RESET 코일은 조건이 ON이 되는 순간 출력 비트를 OFF로 기록하고(해제), 이후 조건이 OFF로 돌아가도 값을 그대로 유지합니다(래치) — SET 코일과 동형이되 방향만 반대입니다. F8 또는 툴바의 RESET 코일 버튼으로 배치합니다.

SET과 RESET은 같은 주소를 켜고 끄는 표준 짝으로 함께 씁니다. `M0 NO → P32 SET` / `M1 NO → P32 RESET` 두 rung을 이어 놓으면, M0로 P32를 켜고 M1로 끄는 회로가 됩니다.



< 그림 5-12 SET/RESET 짝 예제 — M1 ON 순간 P32가 OFF로 전환 >

이 예제의 P32는 트랜지스터 출력입니다. 보드의 릴레이 출력 주소(예: MPINO-16A8R8T의 P40~P47)도 코일이 다루는 방식은 동일합니다 — 어떤 물리 신호로 나가는지는 보드가 결정할 뿐, 코일 자체의 동작(ON/OFF 기록)은 출력 종류와 무관합니다.

타이머(T)·카운터(C) 주소를 RESET 코일 대상으로 지정하면 일반 비트 해제가 아니라 누적값과 완료 비트를 함께 리셋하는 특수 의미로 동작합니다. 자세한 내용은 타이머/카운터를 다루는 장에서 설명합니다.

5-3-4 입력핀을 코일 대상으로 쓰기

코일의 대상 주소로는 출력핀뿐 아니라 **입력핀**(P0~P15 등)도 쓸 수 있습니다. PLC에서는 디지털 입력을 소프트웨어로 덮어쓰는 관용 기법이 있어 이를 허용합니다 — 대표적인 쓰임이 접점 **채터링 방지**와, 수위 센서처럼 신호가 요동치는(헌팅) 입력에 **일부러 지연을 거는** 용도입니다. 출력 코일·SET 코일·RESET 코일 세 가지 모두 입력핀을 대상으로 지정할 수 있습니다.

입력핀 코일은 **메모리에만 쓰는 코일**로 동작합니다. 입력핀에는 물리적으로 값을 쓸 수 없으므로, 앱은 하드웨어 출력 명령을 만들지 않고 그 핀의 **메모리 이미지만** 덮어씁니다. 뒤따르는 rung의 접점은 이 덮어쓴 값을 봅니다.

> **입력핀 코일이 쓴 값은 그 스캔 안에서만 유지됩니다.** 매 스캔이 시작될 때 입력 동기화가 물리 핀의 값을 다시 읽어 메모리 이미지를 채우므로, 앞선 스캔에서 코일이 써 둔 값은 그때 물리값으로 덮입니다. 값을 계속 유지하려면 매 스캔 코일이 다시 쓰도록 회로를 구성해야 합니다.

예를 들어 입력 `P0`의 NO 접점으로 타이머(TON)를 돌리고 그 완료비트를 다시 `P0` 코일로 받으면, P0가 일정 시간 이상 계속 ON으로 들어올 때만 뒤 rung들이 P0를 ON으로 보게 됩니다 — 짧게 튀는 채터링 신호가 걸러집니다. 순수한 내부 비트로 같은 일을 하고 싶다면 P가 아니라 M 주소를 코일 대상으로 쓰면 됩니다.

5-4 신호 반전

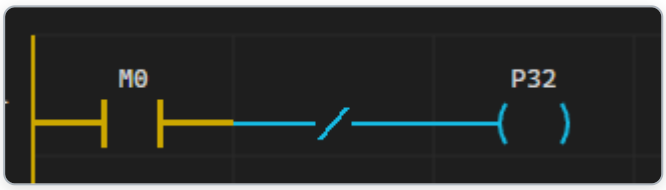
신호 반전(NOT)은 별도 접점이 아니라, 그 자리까지 누적된 rung 조건을 반전시켜 다음 칸으로 넘기는 셀입니다. 다른 접점과 달리 셀에 주소를 입력하지 않습니다. F10 또는 툴바의 신호 반전 버튼으로 배치합니다.

> **한 rung(가지)에는 신호 반전을 1개만 놓을 수 있습니다.** 2개 이상 놓으면 빌드 시 컴파일 에러가 발생합니다.

가장 단순한 형태는 `M0 NO → 신호 반전 → P32 코일`입니다. M0가 OFF면 반전을 거쳐 조건이 참이 되어 P32가 ON, M0가 ON이면 반전을 거쳐 조건이 거짓이 되어 P32는 OFF가 됩니다 — NO 접점(5-2-1)과 정반대로 동작합니다.



< 그림 5-13 신호 반전 기본 예제 — M0 OFF, 반전을 거쳐 P32 ON >

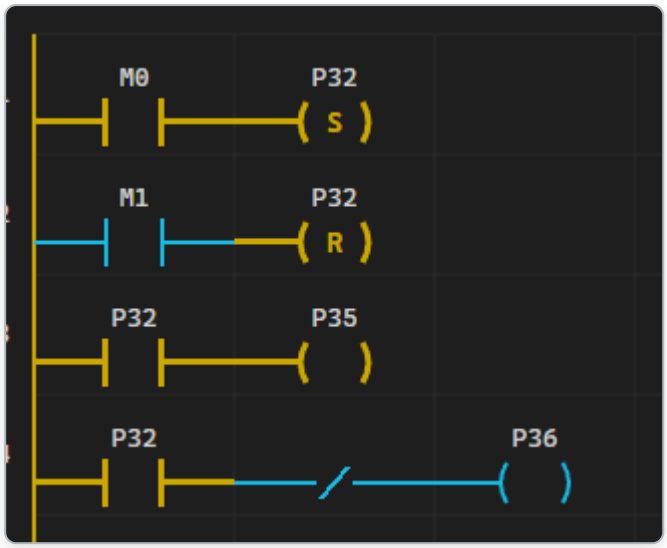


< 그림 5-14 신호 반전 기본 예제 — M0 ON, 반전을 거쳐 P32 OFF >

램프 응용에서는 하나의 상태 비트를 신호 반전으로 나눠 운전등과 정지등을 동시에 켜고 끕니다.

- `M0 NO → P32 SET` — 시작 신호로 P32(모터 운전 상태)를 ON
- `M1 NO → P32 RESET` — 정지 신호로 P32를 OFF
- `P32 NO` 접점 하나에서 두 코일로 갈라집니다 — 그대로 이어지는 갈래는 `P35 코일` (운전등), 신호 반전을 거치는 갈래는 `P36 코일`(정지등)입니다

P32가 ON이면 운전등(P35)이 켜지고 정지등(P36)은 꺼집니다. P32가 OFF면 반대로 정지등만 켜집니다. 따라 할 때는 P32·P35·P36 세 핀을 모두 켜 둡니다(2-4-3).



< 그림 5-15 신호 반전 응용 — P32 ON 상태, 운전등(P35) 점등·정지등(P36) 소등 >

5-5 상승/하강 검출 접점

래더 그리드의 일반 접점은 조건이 켜져 있는 동안 계속 반응하지만, 상승 검출 접점과 하강 검출 접점은 조건이 바뀌는 그 순간만 한 스캔 동안 통전되는 접점 2종입니다. 카운트업이나 트리거처럼 딱 한 번만 실행하고 싶은 로직에 씁니다.

5-5-1 상승 검출 접점 (상승엣지)

상승 검출 접점은 조건 비트가 OFF에서 ON으로 바뀌는 그 한 스캔만 통전되는 접점입니다. F11 또는 툴바의 **상승엣지** 버튼으로 배치합니다.

이전 스캔에 조건이 어떤 상태였는지 기억하기 위해 내부적으로 비트 1개를 씁니다. **이 비트는 컴파일러가 자동으로 관리하는 내부 메모리입니다.** 사용자가 상태 저장용 주소를 따로 지정하거나 신경 쓸 필요가 없고, P·M 같은 사용자 메모리를 소비하지도 않습니다 — 셀에는 조건이 될 입력 주소만 입력하면 됩니다. 같은 주소를 여러 상승/하강 검출 셀에 함께 써도 셀마다 상태가 따로 관리되므로 서로 영향을 주지 않습니다.

일반 접점과의 차이는 예제로 비교하면 분명합니다. 엣지 없이 `M0 NO → 박스함수 D0 = D0 + 1`로 놓으면 M0가 ON인 동안 매 스캔마다 D0가 계속 증가합니다. 반면 `M0 상승 검출 접점 → 박스함수 D0 = D0 + 1`로 놓으면, M0가 ON되는 그 순간 딱 한 번만 D0가 증가합니다. M0를 OFF했다가 다시 ON하면 그때 또 한 번 증가합니다.



< 그림 5-16 상승 검출 접점 예제 — M0 OFF, D0 값 유지(통전 없음) >



< 그림 5-17 상승 검출 접점 예제 — M0 ON 직후, 그 한 스캔만 통전돼 D0가 1 증가 >

상승/하강 검출 접점의 입력 자리에는 다음을 쓸 수 있습니다.

- P·M·D·C 메모리의 비트 참조(`P0` , `M10` 등)
- 타이머·카운터의 완료비트(예: `T0` , `C0`)
- 특수 접점 `@<ms>` · `@F<ms>` (**특수 메모리**(2-2) 참조)
- 워드.비트 형태의 비트 참조(`D0.5` 등 — 워드 메모리의 비트 접점(5-8) 참조)

D0처럼 비트가 아닌 워드 값 자체는 입력으로 쓸 수 없고, 상수 접점 (`@ON` · `TRUE` · `@OFF` · `FALSE` — 5-2-3)도 검출할 엣지가 없어 쓸 수 없습니다.

5-5-2 하강 검출 접점 (하강엣지)

하강 검출 접점은 조건 비트가 ON에서 OFF로 바뀌는 그 한 스캔만 통전되는 접점입니다. F12 또는 툴바의 **하강엣지** 버튼으로 배치합니다. 상태를 관리하는 방식과 입력으로 쓸 수 있는 것은 상승 검출 접점(5-5-1)과 동일합니다.

`M0 하강 검출 접점 → 박스함수 D1 = D1 + 1` 로 놓으면, M0가 ON에서 OFF로 바뀌는 그 순간 딱 한 번만 D1이 증가합니다.



< 그림 5-18 하강 검출 접점 예제 — M0가 ON에서 OFF로 바뀐 순간, 그 한 스캔만 통전돼 D1이 1 증가 >

5-6 비교 접점

비교 접점은 별도의 셀 타입이 아니라, NO·NC 접점의 주소 입력칸에 주소 대신 비교식을 넣어 만드는 접점입니다. 그 식이 참일 때 접점이 통전됩니다. 입력하는 곳은 다른 접점과 같은 셀 편집 모달의 입력칸입니다 — **핀 편집**(2-4)의 "래더 셀 편집" 참고.

지원하는 비교 연산자는 다음 6종입니다.

표기	의미
`>`	좌변이 우변보다 크면 통전
`<`	좌변이 우변보다 작으면 통전
`>=`	좌변이 우변 이상이면 통전
`<=`	좌변이 우변 이하이면 통전
`==`	좌변과 우변이 같으면 통전
`!=`	좌변과 우변이 다르면 통전

예를 들어 `D0 > D1`은 D0가 D1보다 클 때 통전됩니다.

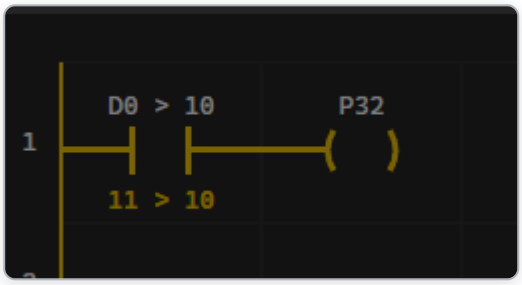
> **구버전 MP STUDIO의 `=>` · `=<` · `=` · `<>` 표기는 지원하지 않습니다.** 반드시 위 표의 정규 표기(`>=` · `<=` · `==` · `!=`)를 써야 합니다.

비교식은 NO·NC 접점에만 쓸 수 있습니다(검출 접점·코일에는 쓸 수 없습니다). 또한 한 칸에는 비교식 1개만 들어갑니다 — `&&` · `||`로 여러 비교식을 묶을 수 없고, 여러 조건을 함께 쓰려면 접점을 직렬(AND)이나 병렬(OR)로 배치해서 표현합니다. NC 접점에 비교식을 넣으면 동작이 반전돼, 식이 거짓일 때 통전됩니다.

식의 좌변·우변에는 메모리 주소(D·C·T 등 워드 값, R 등 실수 값)와 숫자 리터럴뿐 아니라 심볼 이름도 그대로 넣을 수 있어, `Temp > 100`처럼 주소 대신 등록한 이름을 식에 바로 쓸 수 있습니다. 심볼(주소 별명)을 등록·관리하는 방법은 **심볼**(9-5)을 참고하세요. 아두이노 코드에서 만든 변수(`speed > 100`)도 같은 방식으로 쓸 수 있습니다 — **아두이노 변수 사용**(5-9)을 참고하세요.

> **D·C·T 워드는 부호 있는 16비트(−32,768 ~ +32,767)입니다.** 이 범위 밖의 숫자는 DWORD 상수라 워드 주소와 직접 비교하는 식(`C20 > 40000`, `T0 == 65535`)은 결과가 항상 같습니다. 빌드는 되지만「문제」탭에 "항상 거짓(또는 참)" 경고가 뜹니다. 아날로그 출력 채널·타이머 현재값처럼 0~65535로 보는 값을 비교하려면 `UW(C20) > 40000`처럼 **UW**(6-9)로 감싸세요. 같은 비트 패턴인지를 보려면 음수로 써도 됩니다(`C20 == -1`은 65535와 같은 비트).

예제로 `D0 > 10 (NO 비교 접점) → P32 코일`을 놓으면, D0가 10을 초과할 때 P32가 켜집니다. 모니터링 중에는 비교식 아래에 양변의 현재값(예: `6 > 10`)이 함께 표시되고, 값이 추가되는 만큼 그 셀의 행 높이가 자동으로 커집니다. 따라 할 때는 P32를 먼저 켜 둡니다(2-4-3).



< 그림 5-19 비교 접점 모니터링 예제 — D0=11로 식이 참이 되어 11 > 10 값과 함께 통전, P32 ON >

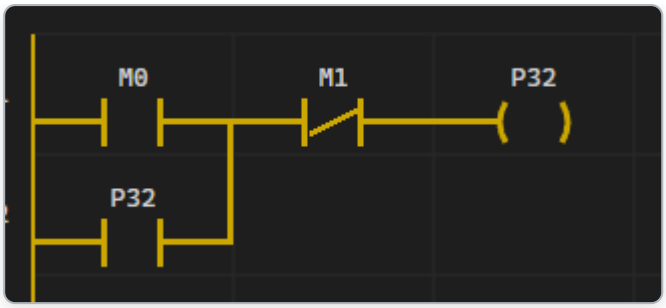
5-7 자기유지 회로

시작 신호(M0)가 ON되면 출력(P32, 예: 모터)이 ON되고, 그 출력 자신의 보조접점(P32 NO)이 함께 닫혀 M0를 다시 OFF해도 회로가 계속 유지됩니다. 정지 신호(M1, NC 접점)가 ON되면 회로가 끊겨 P32가 OFF로 돌아갑니다. 이렇게 출력의 보조접점으로 자기 자신의 ON 상태를 유지하는 회로를 자기유지 회로라고 합니다.

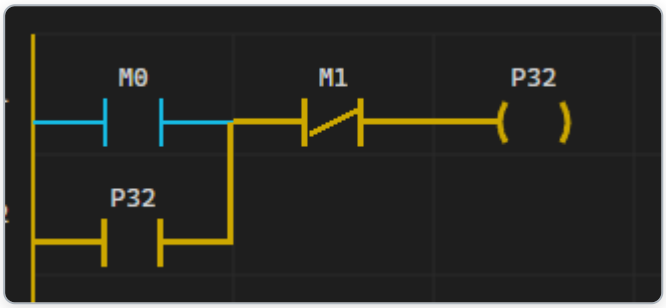
회로는 `[M0 NO ‖ P32 NO] → M1 NC → P32 코일`입니다 — 시작 신호 M0와 P32의 보조접점을 병렬로 놓고, 정지 신호 M1(NC)을 그 뒤에 직렬로 이어, 마지막을 P32 코일로 받습니다(P32는 먼저 핀 설정에서 켜 둡니다 — 2-4-3). 물리 입력 P0~P15 주소도 동일하게 동작합니다.

동작 순서는 다음과 같습니다.

1. 아무 신호도 없으면 P32는 OFF입니다.
2. M0가 ON되면(M1은 눌리지 않아 NC 접점이 닫혀 있으므로) P32가 ON되고, 동시에 P32의 보조접점도 닫힙니다.
3. M0를 다시 OFF로 되돌려도, 방금 닫힌 P32 보조접점이 병렬 경로를 대신 유지하므로 P32는 계속 ON을 유지합니다.
4. M1이 ON되면 NC 접점이 열려 회로가 끊기고, P32가 OFF로 돌아가면서 보조접점도 함께 열립니다.



< 그림 5-20 자기유지 회로 예제 — M0 ON 순간, P32가 ON되며 보조접점도 함께 통전 >



< 그림 5-21 자기유지 회로 예제 — M0가 OFF로 복귀해도 보조접점 경로로 P32가 계속 ON 유지 >

SET 코일과 RESET 코일(5-3-2, 5-3-3)의 짝도 조건이 사라져도 값을 유지한다는 같은 목적을 이룹니다. 자기유지 회로는 SET/RESET 없이 NO·NC 접점 배치만으로 같은 효과를 내는 고전적인 방법입니다.

5-8 워드 메모리의 비트 접점

워드 영역 주소 뒤에 `.비트번호`를 붙이면 그 워드 안의 비트 하나만 접점이나 코일로 쓸 수 있습니다(예: `D0.5`는 D0의 5번째 비트). 주소 표기 전체(native·바이트(B)·워드(W)· 더블워드(D) 뷰 + `.n` 비트 접근)의 사용 가능 영역은 **데이터 메모리**(2-1)의 "접두어와 비트 접근"에서 다룹니다 — 여기서는 그 표기를 접점·코일 자리에 실제로 쓸 때의 대응 관계와 제약을 다룹니다.

워드(W) 뷰로 P·M 영역을 16비트 단위로 묶어 참조할 때는, (워드 인덱스 × 16) + 비트번호가 실제로 가리키는 물리 비트입니다. 바이트(B) 뷰는 같은 방식으로 ×8, 더블워드(D) 뷰는 ×32로 계산합니다. 대표적인 대응 관계는 다음과 같습니다.

표기	대응	계산
`WP0.1`	P1	워드 인덱스 0 × 16 + 비트 1
`WM1.15`	M31	워드 인덱스 1 × 16 + 비트 15
`D10.0` ~ `D10.15`	D10의 0~15번째 비트	native 단위가 이미 워드라 뷰 접두어 없이 바로 비트 지정

폭 뷰를 쓸 수 있는 범위는 영역마다 다릅니다 — P·M은 바이트(B)·워드(W)·더블워드(D) 세 가지 모두, D·C·T는 더블워드(D)만, R은 폭 뷰 자체가 없습니다. 비트 범위는 폭에 따라 B 0~7, W 0~15, D

0~31입니다.

다만 접점·코일 자리에 `.비트`를 붙여 실제로 쓸 수 있는 것은 P·M(폭 뷰를 거쳐)과 D·C(native 워드 또는 더블워드(D) 뷰)뿐입니다. T(타이머)는 더블워드(D) 뷰로 주소를 쓸 수는 있지만, `.비트`를 붙여 접점으로 쓸 수는 없습니다.

워드 안의 비트는 코일로도 쓸 수 있습니다. `D0.5`를 출력·SET·RESET 코일의 대상으로 놓으면 D0의 5번째 비트만 바뀌고 나머지 비트는 그대로 남습니다. 더블워드(D) 뷰(`DD2.20` 등)도 같습니다.

> **접점·코일 자리에는 비트값 참조만 쓸 수 있습니다.** `D0`나 `WM0`처럼 `.비트`가 없는 워드 값 그대로는 접점·코일로 쓸 수 없습니다(빌드 시 거부됩니다). 워드 값을 그대로 비교하려면 접점이 아니라 비교식(5-6)을 씁니다.

예를 들어 `D0.0 NO → P32 코일` / `D0.1 NO → P33 코일` 두 rung을 놓고 D0에 1을 넣으면, 1은 이진수로 0번째 비트만 1이므로 D0.0을 보는 P32만 켜지고, D0.1을 보는 P33은 꺼진 채로 남습니다 — 워드 값 자체와 그 안의 개별 비트가 서로 다르다는 것을 보여주는 예제입니다. 따라 할 때는 P32·P33을 먼저 켜 둡니다(2-4-3).



< 그림 5-22 워드 비트 접점 예제 — D0=1일 때 D0.0 접점(1단)만 통전, D0.1 접점(2단)은 비통전 >

5-9 아두이노 변수 사용

아두이노 코드(`.ino`)에서 만든 변수를 래더에서 접점·코일·박스함수의 자리에 **이름 그대로** 쓸 수 있습니다. 예를 들어 코드 탭에 다음 전역 변수가 있다고 합시다.

```
bool motorOn;
int flags;
int speed;
```

그러면 래더에서 이렇게 쓸 수 있습니다.

래더에 쓰는 것	의미
`motorOn` NO 접점 → M10 코일	C 코드가 `motorOn`을 true로 만들면 M10이 켜집니다

래더에 쓰는 것	의미
`flags.3` NO 접점	`flags` 변수의 3번째 비트(`.0` 이 맨 아래 비트)가 1이면 통전
`speed > 100` 비교 접점	`speed` 가 100보다 크면 통전(비교 접점, 5-6)
`M0` NO 접점 → `motorOn` 코일	래더가 C 코드의 변수에 값을 씁니다
`M0` NO 접점 → `flags.3` SET 코일	조건이 ON이 되면 `flags` 의 3번째 비트를 1로 만듭니다
박스함수 `MOV D0 speed`	D0의 값을 `speed` 에 넣습니다

5-9-1 쓸 수 있는 변수

변수는 `.ino` 코드 탭(어느 탭이든)의 **함수 밖에 선언한 단순 전역 변수**여야 합니다. 한 문장에 변수 하나씩 적고, 형 앞에 `const` 나 `volatile` 이 붙어도 됩니다.

분류	C 형
비트	`bool` , `boolean`
바이트	`char` , `signed char` , `unsigned char` , `int8_t` , `uint8_t` , `byte`
워드(16비트)	`int` , `unsigned` , `short` 계열, `int16_t` , `uint16_t` , `word`
더블워드(32비트)	`long` , `unsigned long` , `int32_t` , `uint32_t`
실수	`float` , `double`

다음은 쓸 수 없습니다 — 배열, 구조체, 포인터, `String` , 함수 안의 지역 변수, `static` 변수, `.cpp` · `.h` 탭에 선언한 변수, `#if` 블록 안에 선언한 변수.

변수를 쓰는 자리마다 허용되는 형이 다릅니다.

자리	쓸 수 있는 것
NO·NC 접점, 상승·하강 검출 접점	`bool` 변수(`motorOn`), 정수 변수의 비트(`flags.3`)
비교 접점	지원하는 모든 형(`speed > 100`)
박스함수의 식 양쪽, 값 명령	지원하는 모든 형 (쓰는 쪽은 `const` 가 아닌 변수). 값 명령은 `MOV` , `ADD` , `SUB` , `MUL` , `DIV` , `MOD` , `INC` , `DEC` , `WAND` , `WOR` , `WXOR` , `NEG`
출력·SET·RESET 코일	`bool` 변수, 정수 변수의 비트(`flags.3`)

정수 변수의 비트는 `변수이름.비트번호` 로 씁니다. 번호 범위는 변수 폭을 따릅니다 — 바이트 0~7, 워드 0~15, 더블워드 0~31. `bool` 이나 실수 변수에는 `.비트번호` 를 붙일 수 없습니다.

> **정수·실수 변수를 접점이나 코일 자리에 이름만 써서 놓을 수 없습니다.** `D0`를 접점으로 쓸 수 없는 것(5-8)과 같은 규칙입니다. `flags.3`처럼 비트를 고르거나, 값을 보려면 `flags != 0` 같은 비교 접점을 쓰세요.

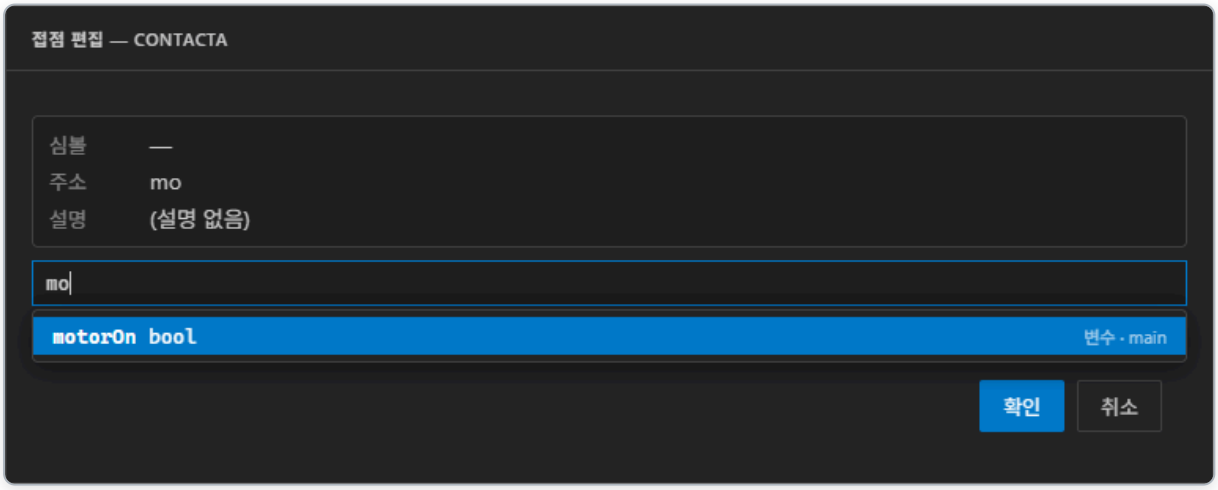
5-9-2 코일로 쓰기

변수를 코일 대상으로 쓰는 동작은 `M` 같은 래더 비트와 같습니다.

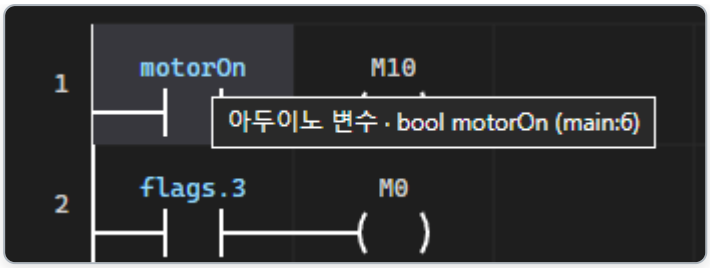
- **출력 코일**: 조건 상태를 매 스캔 변수에 씁니다. 그래서 C 코드가 같은 변수에 쓴 값은 다음 스캔에 래더의 값으로 덮입니다. 값을 유지하려면 SET·RESET 코일을 쓰세요.
- **SET·RESET 코일**: 조건이 ON이 되는 순간 1 또는 0을 쓰고 이후 유지합니다.
- **정수 변수의 비트**(`flags.3`): 그 비트 하나만 바꾸고 나머지 비트는 건드리지 않습니다. `volatile` 변수는 쓰는 순간에만 인터럽트를 잠시 막아, 인터럽트 함수가 바꾼 다른 비트가 지워지지 않게 합니다.
- 같은 변수(또는 같은 비트)를 출력 코일로 두 번 쓰면 이중 코일 오류입니다. `flags.3`과 `flags.4`는 서로 다른 코일이고, SET·RESET 코일은 세지 않습니다.

5-9-3 입력하기

접점·코일·박스함수 입력칸에 변수 이름을 직접 쓰면 됩니다. 이름은 대소문자를 구분합니다. 입력하는 동안 코드 탭에서 찾은 변수가 자동완성 후보로 뜹니다. 정수 변수는 `flags.`처럼 점이 붙은 후보로 나오므로, 고른 뒤 비트 번호를 이어서 입력합니다. 셀에는 변수 이름이 전용 색으로 표시되고, 마우스를 올리면 형과 선언 위치(탭과 줄)가 툴팁으로 나옵니다.



< 그림 5-23 접점 편집창에 mo 를 입력한 모습 — 코드의 bool 변수 motorOn 이 자동완성 후보로 뜬다 >



< 그림 5-24 변수 셀 — 변수 이름이 전용 색으로 표시되고, 마우스를 올리면 형과 선언 위치(main 탭 6번째 줄) 툴팁이 뜬다 >

5-9-4 모니터링 중에는 값을 알 수 없습니다

래더 모니터링은 보드의 래더 메모리 값만 읽어 옵니다. 아두이노 변수는 거기에 들어 있지 않아 모니터링 중에 값을 알 수 없습니다. 그래서 변수 셀은 켜짐·꺼짐 어느 쪽으로도 그려지지 않고 **점선**으로 표시되며(툴팁 "아두이노 변수 값은 모니터링되지 않습니다"), 그 셀 뒤로 이어지는 선도 켜짐·꺼짐 대신 점선으로 표시됩니다 — 변수 때문에 그 뒤 통전 여부를 알 수 없다는 뜻입니다. 래더는 실제로 변수 값대로 동작하니, 값은 시리얼 출력 등으로 확인하세요.

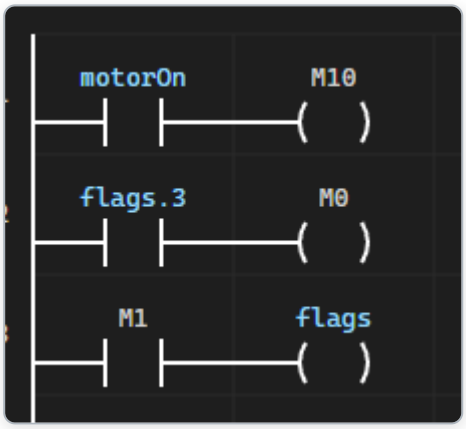
변수 셀에서는 **Ctrl+1**·**Ctrl+0**(4-4)과 **아두이노에 값 쓰기**도 쓸 수 없고, 쓰려고 하면 "`이름` 는 아두이노 변수라 여기서 값을 쓸 수 없습니다."라고 안내합니다.

5-9-5 오류와 해결

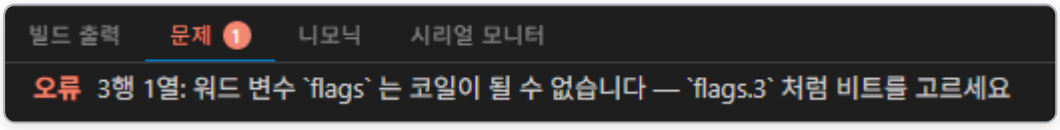
오류는 빌드 출력과 「문제」 탭에 같은 문구로 나오며, 「문제」 탭은 빌드할 때 갱신됩니다.

상황	안내	해결
변수 이름이 코드에 없다(이름을 바꾸거나 지움)	"`이름` 를 찾을 수 없습니다 — 래더 주소, 심볼, 아두이노 전역 변수 어느 것도 아닙니다"	셀의 이름을 새 변수 이름으로 고칩니다
`const` 변수에 쓰기	"읽기 전용(`const`) 변수라 쓸 수 없습니다"	코일·대상 칸에는 `const` 가 아닌 변수를 씁니다
정수 변수를 비트 없이 접점·코일로 사용	"워드 변수 `flags` 는 코일이 될 수 없습니다 — `flags.3` 처럼 비트를 고르세요"	비트를 고르거나 비교 접점을 씁니다
`bool`·실수 변수에 `.비트번호` , 또는 범위 밖 비트	비트 지정 오류	`bool` 은 이름만, 비트 번호는 변수 폭 안으로

상황	안내	해결
폭에 따라 결과가 달라지는 정형 명령에 변수를 사용	"`SHR` 명령은 래더 메모리만 받습니다 — 아두이노 변수 `flags` 는 박스함수 식 (`flags = flags >> 1` 처럼) 으로 쓰세요"	자유입력형 박스함수 식으로 씁니다. 해당 명령은 `SHL`, `SHR`, `ROL`, `ROR`, `WNOT`, `INV`, `H2D`, `D2H`, `GMOV`, `FMOV`, `SCALE`, `FLOOR`, `SQRT`
타이머·카운터의 설정값 칸에 아두이노 변수	타이머는 정수·시간·워드/더블워드 래더 메모리만, 카운터는 정수·워드/더블워드 래더 메모리만 받는다는 오류	숫자를 직접 쓰거나 `D10` 같은 래더 메모리를 씁니다 (6-1-7, 6-2-4)



< 그림 5-25 변수를 쓴 회로 — 1·2행은 motorOn·flags.3 접점, 3행은 정수 변수 flags 를 비트 없이 코일로 써서 오류 >



< 그림 5-26 문제 탭 — 3행 코일의 flags 에 비트를 고르라는 오류 >

> **래더에서 쓰는 변수의 이름을 코드에서 바꾸면, 그 이름을 쓴 셀이 오류가 됩니다.** 래더 셀에는 변수 이름이 글자 그대로 저장되며 자동으로 따라 바뀌지 않습니다. 이름을 바꿨다면 빌드한 뒤 「문제」 탭에서 해당 셀을 찾아 고치세요.

5-9-6 이름이 겹칠 때

- **심볼과 변수의 철자가 같으면 오류입니다.** 심볼(9-5)은 주소의 별명이라, 같은 이름의 변수는 래더에 입력할 수 없습니다. 심볼이나 변수 중 하나의 이름을 바꾸세요.

- **대소문자만 다르면 경고**입니다. 주소 하나만 넣는 셀과 명령의 주소 칸에서는 심볼이 우선하고, 비교 접점과 박스함수 식에서는 철자가 정확히 같은 변수를 씁니다. 그 칸에서도 변수를 쓰려면 이름을 바꾸세요.
- `WM0`**처럼 래더 주소 모양인 변수 이름**은 래더에서 늘 주소로 처리됩니다. 그 변수를 쓰려면 이름을 바꾸세요.

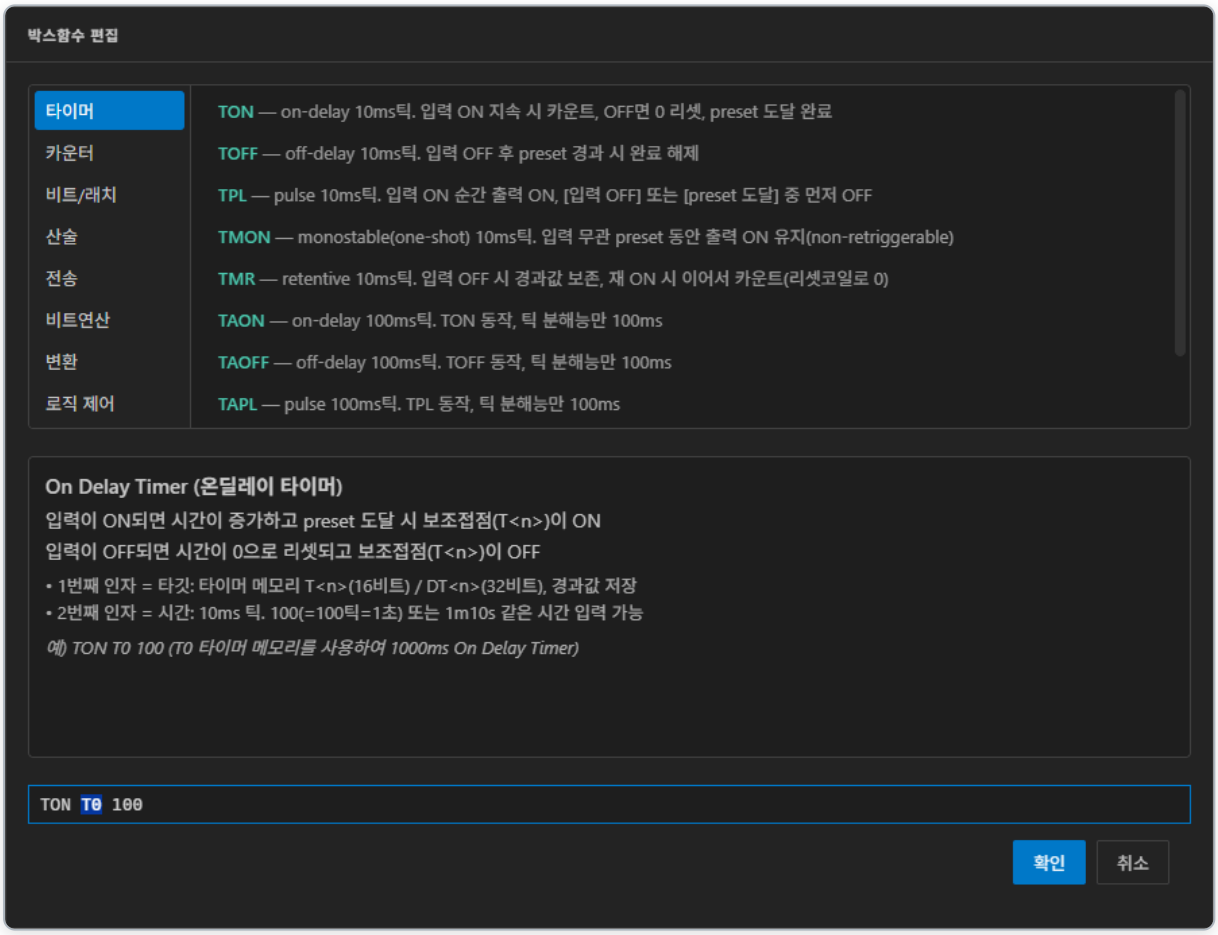


< 그림 5-27 심볼 창 — 심볼 motorOn(M5)이 아두이노 변수 motorOn 과 철자가 같아 오류로 표시된 행 >

6 박스함수

박스함수는 접점·코일과 마찬가지로 래더 그리드에 놓는 셀이지만, 자리는 코일과 같은 rung의 맨 오른쪽(종착점)입니다. 그 rung까지 이어진 조건이 참이면 셀에 적은 명령을 실행합니다 — 출력 코일이 조건 결과를 비트 하나에 그대로 기록하는 것과 같은 원리로, 조건이 참일 때 "명령을 실행"한다는 점만 다릅니다. 타이머·카운터처럼 여러 스캔에 걸쳐 상태를 쌓아 가는 동작부터 사칙연산·비트연산 같은 단발성 계산, 임의의 아두이노 함수 호출까지 모두 이 자리에서 처리합니다.

박스함수 셀은 `\`(백슬래시 — 한국어 키보드에서는 `₩` 키, 또는 왼쪽 툴바의 박스함수 버튼)로 빈 셀에 배치합니다. 배치한 셀을 더블클릭하거나 선택한 뒤 **Enter**를 누르면 박스함수 편집 창이 열립니다. 창 위쪽에는 카테고리별 명령어 팔레트가 있어, 원하는 명령을 클릭하면 입력칸에 템플릿이 채워지고(예: `TON T0 100`), 그 아래 설명 패널에 선택한 명령의 정의·인자·예시가 표시됩니다. 팔레트를 거치지 않고 입력칸에 직접 코드를 입력해도 똑같이 저장됩니다.



< 그림 6-1 박스함수 편집 창 — 카테고리별 명령어 팔레트와 선택한 명령의 설명 패널 >

박스함수는 앱이 팔레트로 제공하는 **정형 명령어**와, 입력칸에 직접 써 넣는 **자유입력형** 두 갈래로 나뉩니다. 정형 명령어는 타이머·카운터·산술·전송·비트연산· 변환·비트/래치·로직 제어 8개 카테고리로 묶여 있으며, 이 장에서는 카테고리별로 6-1부터 차례로 설명합니다. 자유입력형은 `D0 = D0 + 1` 같은 대입식이나 `setMotor(100)` 같은 임의 함수 호출로, 6-9에서 따로 다룹니다.

박스함수 셀의 왼쪽에 접점을 하나도 놓지 않으면 조건이 항상 참인 것으로 취급됩니다. 접점·코일 자체의 동작은 **접점 종류**(5장)에서, 이 장에서 계속 나오는 전용 특수 메모리 C(카운터)·T(타이머)는 **데이터 메모리**(2-1)에서 이미 다뤘습니다. 예제에 `P0`·`P32` 같은 보드 핀(P 주소)이 나오면, 따라 하기 전에 그 핀을 **래더 핀맵 설정**에서 "사용"으로 켜야 빌드됩니다 — **핀 설정 따라 하기**(2-4-3) 참고.

박스함수는 이렇게 생겼다

박스함수 셀은 가로로 긴 표 모양으로 그려집니다. 맨 위는 명령어 이름을 적은 **띠**이고, 그 아래는 **파라미터 1개당 칸 1개**로 나뉩니다 — `ADD D0 D1 D2`라면 띠에 `ADD`, 그 아래에 `D0`·`D1`·`D2` 세 칸입니다. 칸 수는 명령어마다 정해져 있어서, 인자를 덜 적어도 빈 자리가 점선 칸으로 남아 몇 개가 모자란지 그대로 보입니다.

- **쓰기 대상 칸에는 음영**이 깔립니다. `ADD D0 D1 D2`의 `D2`, `MOV D0 D1`의 `D1`처럼 결과가 저장되는 칸을 표시하는 것으로, 대상이 마지막 토큰이 아닌 SCALE(6-3-3)에서 특히 도움이 됩니다. 쓰기 대상이 없는 JMP·JMPN·MCS·MCSCLR(6-8)에는 음영이 없습니다.
- 박스는 칸 수만큼 **같은 행의 이웃 칸까지 차지**합니다. 오른쪽 빈 칸을 먼저 쓰고, 모자라면 왼쪽으로 이어진 가로선 위를 덮습니다. 덮인 칸은 외곽선이 사라져 박스가 한 덩어리로 보이며, 도중에 세로선이 있으면 그 경계를 넘지 않습니다.
- **칸마다 값과 심볼이 따로 표시**됩니다. 그 주소에 등록한 심볼(9-5)이 있으면 심볼 이름이 주 글자로 나오고, 래더 모니터링(8-5) 중이면 칸마다 그 자리의 현재값이 함께 붙습니다. 프리셋을 `7s`·`1m10s`처럼 시간 문법으로 쓴 타이머는 대상 칸도 틱 수 대신 시간으로 환산해 보여 주고(8-7), CTUD(6-2-3)의 up·down 칸에 T·C 주소를 쓰면 워드값이 아니라 접점과 같은 완료비트를 보여 줍니다.
- 인자 자리에는 주소 대신 **등록한 심볼 이름**을 그대로 써도 됩니다 — `MOV Temp D0` 처럼 쓰면 되고, 심볼 등록·관리는 **심볼**(9-5)을 참고하세요. 확정하면 심볼 이름은 그 심볼의 **주소로 바뀌어 저장**되고(칸에는 계속 심볼 이름이 보입니다), 박스함수를 붙여넣을 때도 같습니다. 그래서 나중에 심볼 이름을 바꾸거나 지워도 인자는 같은 주소를 가리킵니다. 프로젝트에 없는 이름은 입력한 그대로 남습니다.

자유입력형(6-9) 가운데 대입식(`D0 = D0 + 1`)은 칸을 나누지 않습니다 — 띠에 `code`가 찍히고 그 아래에 적어 넣은 식이 한 줄로 그대로 나오며, 모니터링 중에는 그 밑에 좌변 주소의 현재값이 붙습니다. 함수 호출(`setMotor(100)`)은 정형 명령어처럼 띠에 함수 이름이, 그 아래에 인자가 하나씩 칸으로 나옵니다.

인자 칸 단위 편집

박스함수는 칸 단위로 고칩니다. 칸을 더블클릭하거나 커서를 올려놓고 **Enter**를 누르면 편집 창이 열리면서 **그 칸의 글자만 선택**된 상태가 되므로, 곧바로 타이핑해도 줄 전체가 날아가지 않습니다. 박스 안에서 `←`/`→`를 누르면 인자 칸을 하나씩 옮겨 다니고, 커서가 있는 칸에는 테두리가 생깁니다(쓰기 대상 칸 음영은 채움이라 같은 칸에서 겹쳐도 구분해 볼 수 있습니다). 끝 인자에서 한 번 더 누르면 박스 밖으로 나갑니다. 명령어 이름 띠는 커서 대상이 아니므로, 명령을 바꾸려면 편집 창에서 고칩니다.

복사·붙여넣기도 칸 단위로 동작합니다. 키는 **Ctrl+C/Ctrl+V** 그대로이고, 갈리는 기준은 **붙여넣는 자리**입니다 — 다른 박스함수의 인자 칸에 붙이면 **그 칸의 값만** 바뀌고, 빈 칸이나 보통 셀에 붙이면 박스 전체가 들어갑니다. 박스 전체를 붙여넣을 때는 **붙여넣은 자리가 언제나 첫 인자**가 되도록 놓이며, 자리가 모자라거나 세로선이 가로막으면 붙이지 않고 그 이유를 알려 줍니다.

칸 커서가 없는 박스에서는 `←`/`→`가 박스를 통째로 건너뜁니다 — 칸을 나누지 않는 대입식(6-9), 그리고 폭이 칸 수보다 좁아 압축돼 그려진 박스가 여기에 해당합니다. 범위 선택과 Ctrl 토글의 단위는 인자가 아니라 여전히 셀이며, 박스에 걸리면 왼쪽으로 덮인 가로선까지 한 덩어리로 다룹니다. 셀 커서를 옮기고 범위를 만드는 방법 자체는 **래더 그리드 편집**(4장)의 "커서 이동과 선택"을 참고하세요.

> **분기 회로에서도 박스함수는 단일 rung과 같이 쓸 수 있습니다. 예외는 MCS·MCSCLR뿐입니다.** 접점 하나가 세로선으로 갈라져 박스함수 여러 개를 동시에 먹이는 회로에서도, 각 박스함수는 단일 rung에 놓았을 때와 똑같이 그 가지의 조건으로 동작합니다. MCS·MCSCLR만은 자기 rung이 아니라 뒤따르는 rung들에 걸리는 구간 게이트라, 분기에 놓이면 옆 가지의 코일이 구간 안인지 밖인지가 정해지지 않으므로 **컴파일이 거부**됩니다(6-8). MCS·MCSCLR은 rung 하나를 통째로 차지하게 두세요.

6-1 타이머

타이머 박스함수는 총 10종으로, 동작 5종(TON·TOFF·TPL·TMON·TMR)에 틱 분해능 2종을 곱한 구성입니다 — 무접두(예: `TON`)는 기본 10ms/카운트, `TA`를 접두한 판(예: `TAON`)은 기본 100ms/카운트입니다. 이 단위는 프로젝트마다 설정에서 바꿀 수 있습니다(6-1-6). 이 절의 10ms·100ms는 모두 기본값 기준이고, 다섯 동작 모두 문법은 같습니다. 이 절의 예제는 기본 단위(`T*` 10ms, `TA*` 100ms) 기준입니다.

```
<함수> <타깃> <프리셋>
```

- **타깃**: 경과값을 저장할 타이머 메모리 주소입니다. `T<n>`(16비트, 경과값 하나) 또는 `DT<n>`(32비트, `T<n>`과 `T<n+1>` 두 워드를 이어 씀)을 씁니다.
- **프리셋**: 완료까지 필요한 틱 수입니다. `100`처럼 정수를 그대로 쓰면 그 개수만큼 틱(기본값에서 무접두는 10ms, TA 접두는 100ms)을 기다립니다. `1m10s`·`500ms`·`2s`처럼 시간 단위(h/m/s/ms 조합 가능)로도 쓸 수 있고, 저장할 때 내부적으로 틱 수로 환산됩니다. 프리셋에는 **타깃 폭이 정하는 상한**이 있어, 환산한 틱 수가 그 폭을 넘으면 컴파일이 거부됩니다 — `T<n>`(16비트)은 최대 **65535틱**이라 무접두 `TON T0 1h`(360000틱)는 에러가 되지만, 같은 1시간이라도 100ms 틱인 `TAON T0 1h`(36000틱)는 통과합니다(둘 다 기본 단위 기준). 더 긴 시간이 필요하면 32비트 타깃 `DT<n>`(최대 4294967295틱)을 쓰세요. `D10`처럼 **메모리 주소**를 쓰면 스캔마다 그 메모리의 값을 프리셋으로 읽어, 실행 중에도 HMI 등으로 시간을 바꿀 수 있습니다(6-1-7).

같은 이름이라도 자리에 따라 뜻이 다릅니다. 박스함수 자리(타깃)에 쓴 `T0`는 경과값을 가리키지만, **같은 `T0`를 접점 자리에 놓으면 경과값이 아니라 완료비트**를 가리킵니다. 경과값이 프리셋에 도달하면 완료비트가 ON되고, 그 상태를 `T0` 접점으로 읽어 다음 코일로 넘길 수 있습니다.

타이머 타깃(`T<n>`·`DT<n>`)을 **RESET 코일**(5-3-3)의 대상으로 지정하면, 일반 비트 해제가 아니라 경과값과 완료비트를 함께 0으로 되돌리고, 그 시점부터 프리셋 시간을 새로 재는 특수 의미로 동작합니다. TON·TOFF·TPL·TMON은 입력 상태만으로도 자연히 리셋되지만, 적산 동작인 TMR은 입력이 꺼져도 값을 그대로 보존하므로 중간에 리셋하려면 RESET 코일이 필요합니다(아래 TMR 절 참고).

6-1-1 TON — 온 딜레이 타이머

정의

TON(On Delay Timer, 온 딜레이 타이머)은 입력이 ON인 동안 경과 시간을 누적하다가, 그 값이 프리셋에 도달하면 완료비트를 ON으로 켜는 타이머입니다. 입력이 OFF로 돌아가면 경과값과 완료비트가 즉시 0으로 리셋됩니다. "입력이 켜진 채로 프리셋 시간만큼 유지돼야 완료"되는, 가장 기본적인 지연 동작입니다.

사용법

```
TON <타깃> <프리셋>
```

`TON T0 100`은 T0를 경과값 타깃으로 써서 100틱(기본값에서 100×10ms=1초) 뒤에 완료비트를 켭니다. 더 큰 단위(기본 100ms)가 필요하면 접두 `TA`를 붙인 `TAON`을 씁니다.

니모닉	틱	의미
`TON`	10ms/카운트(기본)	무접두판. `TON T0 100` = 100×10ms = 1초
`TAON`	100ms/카운트(기본)	`TA` 접두판. 동작은 TON과 동일, `TAON T0 30` = 30×100ms = 3초

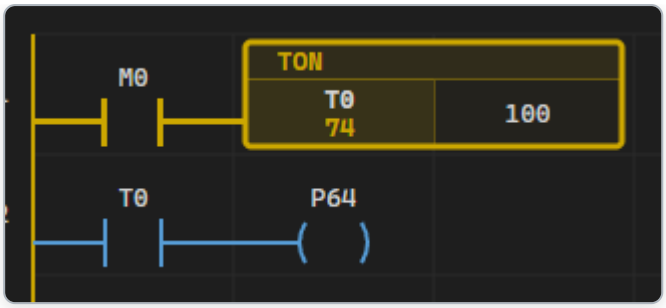
예제

`M0 NO 접점 → TON T0 100`으로 놓으면, M0가 ON인 동안 경과 시간이 쌓이다가 1초가 지나면 완료비트 T0가 ON됩니다.

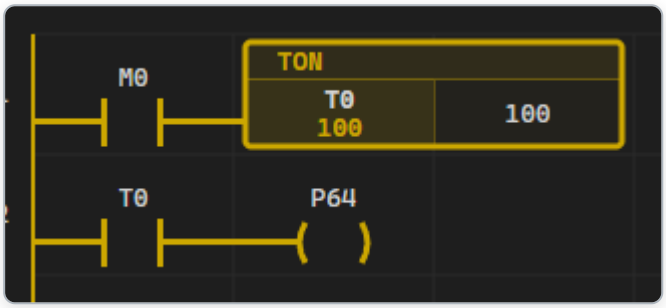


< 그림 6-2 TON 예제 회로 — M0 NO 접점으로 TON T0 100 구동 >

완료비트를 다른 rung에서 받으려면 `T0 NO 접점 → P32 코일`처럼 이어두면, T0가 완료비트로 통전될 때 P32도 함께 켜집니다.



< 그림 6-3 TON 모니터링 — M0 ON 지속 중 경과값이 증가하는 모습 >



< 그림 6-4 TON 모니터링 — 경과값이 프리셋(100)에 도달해 완료비트 T0가 ON으로 통전 >

주의사항

> **프리셋의 시간 문법은 그 틱의 배수여야 합니다.** 예를 들어 무접두(기본 10ms 틱)에서 `TON T0 15ms`처럼 10의 배수가 아닌 시간을 쓰면 컴파일 시 Syntax 에러가 됩니다. 정수 틱(`100`)이나 배수가 맞는 시간(`150ms` · `1s`)을 쓰세요.

- 타깃 인덱스가 프로젝트에 설정된 타이머 메모리 개수를 넘으면 컴파일이 거부됩니다. 개수 조정은 **데이터 메모리**(2-1)의 "영역 크기와 메모리 설정"을 참고하세요.
- 입력이 한 번이라도 OFF로 돌아가면 경과값이 0으로 리셋됩니다. 중간에 끊기지 않고 이어서 세고 싶다면 TMR(적산 타이머, 6-1-5)을 씁니다.

6-1-2 TOFF — 오프 딜레이 타이머

정의

TOFF(Off Delay Timer, 오프 딜레이 타이머)는 입력이 ON이면 완료비트를 지연 없이 즉시 ON으로 켜고, 입력이 OFF로 바뀐 뒤부터 경과 시간을 세어 프리셋에 도달해야 비로소 완료비트를 OFF로 돌리는 타이머입니다. 켜지는 쪽은 즉시, 꺼지는 쪽만 지연된다는 뜻에서 "오프 딜레이"입니다.

사용법

```
TOFF <타깃> <프리셋>
```

`TOFF T1 100`은 입력이 OFF로 바뀐 뒤 100틱(기본값에서 1초)이 지나야 완료비트가 꺼집니다. 더 큰 단위(기본 100ms)는 `TAOFF`를 씁니다.

니모닉	틱	의미
`TOFF`	10ms/카운트(기본)	무접두판. `TOFF T1 100` = OFF 전환 후 1초 뒤 완료비트 OFF
`TAOFF`	100ms/카운트(기본)	`TA` 접두판. 동작은 TOFF와 동일, `TAOFF T1 100` = OFF 전환 후 10초 뒤 OFF

예제

`M1 NO 접점 → TOFF T1 100`으로 놓으면, M1이 ON인 동안 완료비트 T1이 즉시 ON을 유지합니다. M1이 OFF로 바뀐 순간부터 경과 시간이 쌓이기 시작해, 1초가 지나야 완료비트가 OFF로 풀립니다.



< 그림 6-5 TOFF 예제 회로 — M1 NO 접점으로 TOFF T1 100 구동 >

주의사항

- 입력이 ON과 OFF를 빠르게 반복하면, OFF로 바뀔 때마다 경과 시간이 처음부터 다시 쌓입니다 — 완료비트가 계속 ON을 유지한 채 꺼지지 않을 수 있습니다.
- 타깃 인덱스 범위·프리셋 시간 문법 규칙은 TON(6-1-1)과 동일합니다.

6-1-3 TPL — 펄스 타이머

정의

TPL(Pulse Timer, 펄스 타이머)은 입력이 OFF에서 ON으로 바뀌는 상승 엣지 순간 완료비트를 켜고 경과 시간을 세기 시작하는 타이머입니다. 이후 **입력이 OFF로 돌아가는 것**과 **경과값이 프리셋에 도달하는 것** 중 먼저 오는 쪽에서 완료비트가 자동으로 꺼집니다 — 입력이 길게 유지돼도 펄스 길이는 프리셋을 넘지 않습니다.

사용법

```
TPL <타깃> <프리셋>
```

`TPL T2 50`은 입력 상승 엣지 순간부터 최대 50틱(기본값에서 0.5초) 동안만 완료비트를 ON으로 유지합니다. 더 큰 단위(기본 100ms)는 `TAPL`을 씁니다.

니모닉	틱	의미
`TPL`	10ms/카운트(기본)	무접두판. `TPL T2 50` = 상승 엣지부터 최대 0.5초 펄스
`TAPL`	100ms/카운트(기본)	`TA` 접두판. 동작은 TPL과 동일, `TAPL T2 50` = 최대 5초 펄스

예제

`M2 NO 접점 → TPL T2 50`으로 놓으면, M2가 ON되는 순간(상승 엣지) 완료비트 T2가 즉시 켜지고 0.5초 카운트를 시작합니다. 그사이 M2가 먼저 OFF로 돌아가거나 0.5초가 다 차면(둘 중 먼저 오는 쪽) 완료비트가 자동으로 꺼집니다.



< 그림 6-6 TPL 예제 회로 — M2 NO 접점으로 TPL T2 50 구동 >

주의사항

- 입력이 계속 ON을 유지해도 펄스는 프리셋 시간만큼만 지속되고 자동으로 꺼집니다 — 일반 접점처럼 "ON인 동안 계속 통전"되는 동작이 아닙니다.
- 타깃 인덱스 범위·프리셋 시간 문법 규칙은 TON(6-1-1)과 동일합니다.

6-1-4 TMON — 모노스테이블 타이머

정의

TMON(Monostable Timer, 모노스테이블/원샷 타이머)은 입력 상승 엣지 한 번으로 완료비트를 켠 뒤, 그 뒤로는 **입력 상태와 무관하게** 프리셋 시간이 다 찰 때까지 완료비트를 ON으로 유지하는 타이머입니다. 프리셋이 진행되는 동안 입력이 다시 OFF→ON으로 바뀌어도(재트리거) 무시되고, 처음 시작한 카운트가 그대로 이어집니다.

사용법

```
TMON <타깃> <프리셋>
```

`TMON T3 200`은 입력 상승 엣지 순간부터 200틱(기본값에서 2초) 동안 완료비트를 ON으로 유지합니다. 더 큰 단위(기본 100ms)는 `TAMON`을 씁니다.

니모닉	틱	의미
`TMON`	10ms/카운트(기본)	무접두판. `TMON T3 200` = 상승 엣지부터 2초간 ON 유지
`TAMON`	100ms/카운트(기본)	`TA` 접두판. 동작은 TMON과 동일, `TAMON T3 200` = 20초간 ON 유지

예제

`M3 NO 접점 → TMON T3 200`으로 놓으면, M3가 ON되는 순간 완료비트 T3이 켜지고, 그 뒤로는 M3의 상태와 무관하게 2초가 다 찰 때까지 켜진 채로 유지됩니다.



< 그림 6-7 TMON 예제 회로 — M3 NO 접점으로 TMON T3 200 구동 >

주의사항

> **TPL과 혼동하기 쉽습니다.** TPL은 입력이 OFF로 돌아가면 프리셋 전이라도 즉시 꺼지지만, TMON은 한 번 켜지면 입력 상태와 무관하게 프리셋이 끝나야만 꺼집니다.

- 완료비트가 ON을 유지하는 동안 들어오는 새 상승 엣지는 재트리거하지 않고 무시됩니다(non-retriggerable).
- 타깃 인덱스 범위·프리셋 시간 문법 규칙은 TON(6-1-1)과 동일합니다.

6-1-5 TMR — 적산 타이머

정의

TMR(Retentive Timer, 적산/누적 타이머)은 입력이 ON인 동안만 경과 시간을 쌓고, 입력이 OFF가 되면 그 값을 그대로 둔 채 멈추는 타이머입니다. TON과 달리 입력이 꺼져도 경과값이 리셋되지 않으며, 입력이 다시 ON되면 멈췄던 값부터 이어서 누적됩니다 — 여러 번에 걸쳐 ON된 시간을 모두 더해 프리셋에 도달시키는 용도입니다.

사용법

```
TMR <타깃> <프리셋>
```

`TMR T4 500`은 입력이 ON인 시간을 모두 합쳐 500틱(기본값에서 5초)이 되면 완료비트가 켜집니다. 더 큰 단위(기본 100ms)는 `TAMR`을 씁니다.

니모닉	틱	의미
`TMR`	10ms/카운트(기본)	무접두판. `TMR T4 500` = 누적 ON 시간 5초에 완료
`TAMR`	100ms/카운트(기본)	`TA` 접두판. 동작은 TMR과 동일, `TAMR T4 500` = 누적 ON 시간 50초에 완료

예제

`M4 NO 접점 → TMR T4 500`으로 놓으면, M4가 ON인 동안만 경과 시간이 쌓이고 OFF가 되면 멈춥니다. M4가 다시 ON되면 멈췄던 값부터 이어서 세어, ON된 시간을 모두 합해 5초가 되면 완료비트 T4가 켜집니다.



< 그림 6-8 TMR 예제 회로 — M4 NO 접점으로 TMR T4 500 구동 >

주의사항

> **TMR은 입력만으로는 경과값이 리셋되지 않습니다.** 도중에 누적값을 0으로 되돌리려면 T4를 대상으로 하는 **RESET 코일**(5-3-3)을 별도로 둬야 합니다 — `M5 NO 접점 → T4 RESET`처럼 놓으면 M5가 ON되는 순간 T4의 경과값과 완료비트가 함께 0이 되고, 그 시점부터 프리셋 시간을 새로 잽니다. M4가 ON인 채로 리셋해도 프리셋 전체를 처음부터 다시 재며, M4가 OFF일 때 리셋하면 쌓아 둔 값이 지워지고 다음에 M4가 ON될 때부터 프리셋 전체를 잽니다.

- 타깃 인덱스 범위·프리셋 시간 문법 규칙은 TON(6-1-1)과 동일합니다.

6-1-6 타이머 단위 설정

카운트 하나가 몇 ms인지는 프로젝트마다 정합니다. **설정 › 메모리 영역 설정**(**Ctrl+Shift+M**, 2-1)을 열고 아래쪽 **타이머 단위**에서 고르면, 선택한 값이 프로젝트(`.mp2`)에 함께 저장됩니다.

항목	적용되는 박스함수	고를 수 있는 값	기본값
`T*` 타이머 단위	TON·TOFF·TPL·TMON·TMR	1ms / 10ms	10ms
`TA*` 타이머 단위	TAON·TAOFF·TAPL·TAMON·TAMR	10ms / 100ms / 1000ms	100ms

`TA*` 단위는 `T*` 단위보다 커야 합니다. `T*`보다 작거나 같은 `TA*` 값은 목록에서 고를 수 없고, 두 값이 겹치게 되면 경고가 뜨며 저장할 수 없습니다. 예를 들어 `T*`를 10ms로 두면 `TA*`는 100ms 이상을 골라야 합니다.

단위를 바꾸면 이 장의 설명이 다음과 같이 달라집니다.

- **정수 프리셋은 카운트 수입니다.** `TON T0 100`의 100은 100카운트이므로 기본 설정에서는 1초, `T*`를 1ms로 바꾸면 0.1초입니다.
- **시간 문법 프리셋은 어느 설정에서나 같은 시간입니다.** `TON T0 1s`는 1초입니다. 다만 새 단위의 배수가 아니거나(`TON T0 15ms`를 10ms 단위에서 쓰는 경우) 환산한 카운트가 최대 카운트를 넘으면 컴파일 에러가 됩니다.
- **최대 시간이 단위에 비례합니다.** `T<n>`(16비트)은 최대 65535카운트이므로 `T*`를 1ms로 두면 약 65초까지만 쓸 수 있습니다. 더 긴 시간은 `TA*` 판이나 32비트 타깃 `DT<n>`을 쓰세요.
- **기본값이 아닌 프로젝트에서는 숫자 프리셋 칸 아래에 실제 시간이 표시됩니다.** `T*`를 1ms로 바꾼 프로젝트에서 `TON T0 100`을 쓰면 `100ms`로 보입니다. 기본값 프로젝트의 화면은 그대로입니다.

자동으로 바뀌지 않는 것이 있습니다. `T0 > 50` 같은 비교 접점, `MOV T0 D0`, 코드 창의 `T[0]`, Modbus 4000번대 현재값, `TON T0 D10`의 D10에 든 프리셋 값, 정전 유지로 보드에 저장된 T 값은 **카운트 값**이라 단위를 바꿔도 따라 바뀌지 않습니다. T 영역을 옵션2(정전 감지 저장)로 유지하는 경우, 단위를 바꾼 뒤 다음 정전 저장 전까지는 부팅할 때마다 옛 카운트가 복원되어 그 타이머가 일찍 또는 늦게 완료될 수 있습니다. 단위를 바꾼 뒤에는 이런 자리의 숫자를 직접 확인하세요. 또 단위를 바꾸면 래더 되돌리기 기록이 지워집니다.

1ms는 재는 단위이지 정확도가 아닙니다. `millis()`가 약 1.024ms마다 갱신되고 가끔 2씩 뛰며, 완료는 스캔마다 확인하므로 실제 완료가 스캔 시간에 약 1ms를 더한 만큼 늦을 수 있습니다.

Modbus 프레임을 받는 동안에는 스캔이 약 12ms 멈춥니다. 통신으로 입력을 켜면 그 시간이 경과로 잡혀 타이머가 프리셋보다 약 12ms 일찍 끝날 수 있습니다. 1ms 단위로 5처럼 이보다 짧은 값이면 입력이 켜지자마자 완료할 수 있습니다. 짧은 시간을 통신으로 켜는 회로는 이 오차를 고려하세요.

단위를 바꿔 저장할 때의 대화상자

영향받는 타이머가 있는 상태에서 단위를 바꿔 저장하면 **타이머 단위 변경** 대화상자가 열려, 어느 박스함수가 바뀌는지 개수로 알려 줍니다. 개수는 다음 세 가지입니다.

- **변환 가능 n개** — 숫자를 변환하면 시간이 유지됩니다(예: `TON T0 100`을 10ms에서 1ms로 바꾸면 `TON T0 1000`).
- **변환 불가 n개** — 같은 시간을 숫자로 쓸 수 없어 글자를 바꾸지 않습니다(시간이 달라집니다). 새 단위로 나누어떨어지지 않거나 최대 카운트를 넘는 경우입니다.
- **시간 표기 문제 n개** — 고치지 않으면 컴파일 에러가 됩니다. `1s`·`15ms` 같은 시간 프리셋이 새 단위의 배수가 아니거나 최대 카운트를 넘는 경우입니다.

각 항목은 래더 이름과 행·열 위치와 함께 나열됩니다. 변환 불가와 시간 표기 문제 항목에는 사유가 붙고, 변환 가능 항목은 `옛 글자 → 새 글자`(예: `TON T0 100 → TON T0 1000`)로 보여 주며 처음에는 접혀 있습니다(제목을 누르면 펼쳐집니다). 변환할 항목이 없으면 **숫자를 변환해 시간 유지** 버튼은 비활성입니다. 버튼은 세 개입니다.

버튼	동작
숫자를 변환해 시간 유지	변환 가능한 정수 프리셋의 숫자를 새 단위에 맞게 바꿔 시간을 그대로 둡니다
숫자 그대로(시간이 바뀜)	글자는 건드리지 않고 단위만 바꿉니다. 정수 프리셋의 시간이 달라집니다
취소	단위를 바꾸지 않습니다

이 설정이 든 프로젝트는 이 기능이 없는 옛 버전에서 열리지 않습니다. 기본값(`T*` 10ms,

> `TA*` 100ms) 그대로인 프로젝트는 옛 버전에서도 열립니다.

6-1-7 메모리로 프리셋 지정 (HMI에서 시간 바꾸기)

정의

타이머의 프리셋 자리에 숫자나 시간 대신 **메모리 주소**를 쓰면, 타이머가 스캔마다 그 메모리의 현재 값을 프리셋으로 읽습니다. 프로그램을 다시 올리지 않고도 HMI(터치 패널)나 통신으로 그 메모리에 값을 써서 타이머 시간을 바꿀 수 있습니다. 타이머 10종(TON·TOFF·TPL·TMON·TMR, TAON·TAOFF·TAPL·TAMON·TAMR) 모두, 타깃이 `T<n>`이든 `DT<n>`이든 같은 규칙입니다.

사용법

```
TON <타깃> <프리셋 메모리>
```

종류	쓸 수 있는 주소	예
워드 메모리	`D` `C` `T` `WP` `WM`	`TON T0 D10`, `TON T0 WM1`
더블워드 메모리	`DD` `DC` `DT` `DP` `DM`	`TON DT0 DD5`
심볼	위 주소를 가리키는 심볼	`TON T0 Speed` (`Speed`가 `D10`을 가리킬 때)

쓸 수 없는 것은 컴파일 에러가 됩니다 — 실수(`R`), 비트(`M0`, `D0.5`, `WM0.3`), 8비트(`BM0`), 아두이노 변수, 식(`D10+1`), 메모리 범위 밖 주소. 프리셋 자리의 `T<n>`은 완료비트가 아니라 경과값 워드를 읽습니다. 카운터(CTU·CTD·CTUD)의 메모리 프리셋은 6-2-4를 보세요.

예제

`M0 NO 접점 → TON T0 D10`, `T0 NO 접점 → P32 코일`로 놓으면, M0가 ON인 동안 경과값이 쌓이다가 D10에 든 카운트만큼 지나면 T0가 ON되고 P32가 켜집니다. HMI가 통신으로 D10에 `300`을 쓰면 기본 단위(10ms)에서 3초, `500`을 쓰면 5초가 됩니다. 래더를 고치거나 다시 올릴 필요가 없습니다.

값이 뜻하는 것

> **메모리 값은 시간이 아니라 카운트입니다.** 기본 단위에서 `TON T0 D10`의 D10이 500이면 5초, `TAON T0 D10`의 D10이 50이면 5초입니다(`TAON`은 100ms 단위). 타이머 단위 설정(6-1-6)을 바꾸면 같은 값의 시간이 달라지고, 메모리에 든 값은 저절로 바뀌지 않습니다. 타이머 단위 변경 대화상자의 숫자 변환 대상도 아니므로, 단위를 바꾼 뒤에는 HMI에서 쓰는 값을 직접 확인하세요.

- **값은 부호 없이 읽습니다.** 모니터링에서 −1로 보이는 D10은 65535카운트입니다.

- `T<n>` **(16비트) 타깃에 더블워드를 넣으면 65535에서 멈춥니다.** 더 큰 값을 넣어도 65535카운트로 취급합니다. 32비트 `DT<n>` 타깃은 32비트 값을 그대로 씁니다.
- **실행 중에 값을 바꾸면 바로 반영됩니다.** 진행 중에 값을 올리면 새 값까지 이어서 세고, 경과값보다 아래로 내리면 다음 스캔에 타이머가 끝납니다.
- **TON·TMR이 완료된 뒤에 값을 올려도 완료비트는 켜진 채입니다.** TON은 입력이 OFF되거나 리셋될 때까지, TMR은 리셋될 때까지 꺼지지 않습니다. 경과값은 완료된 뒤 지나간 시간이 반영되어 새 값까지 한꺼번에 올라갈 수 있습니다. 시간을 처음부터 다시 재려면 입력을 껐다 켜거나 리셋하세요. TMR은 입력을 꺼도 값이 보존되므로 리셋해야 합니다.
- **값이 0이면** TON·TMR은 입력이 ON되자마자 완료되고, TOFF는 입력이 OFF되자마자 완료비트가 풀립니다. TPL·TMON은 같은 스캔에 끝나므로 출력 ON이 보이지 않습니다.

주의사항

> **전원을 켠 직후에는 값이 0일 수 있습니다.** HMI가 값을 쓰기 전의 D10은 0이라, 그 사이 TON은 바로 완료되고 TPL·TMON의 출력은 나타나지 않습니다. 전원을 껐다 켜도 값이 남아 있어야 한다면 그 메모리를 **EEPROM 정전유지**(설정 메뉴, Ctrl+Shift+E)에 넣으세요.

- **더블워드 번호는 타깃과 프리셋이 다릅니다.** 타이머 타깃의 `DT2`는 `T[2]`, `T[3]`이지만, 프리셋 자리의 `DT1`이 `T[2]`, `T[3]`입니다. 프리셋에서는 더블워드가 식·MOV와 같은 규칙을 따라 `DD<n>` = `D[2n]`, `D[2n+1]`이므로, `DD5`는 `D[10]`, `D[11]`입니다.
- 메모리 프리셋 칸은 숫자가 아니라서 타이머 단위 변경 대화상자가 글자를 바꾸지 않습니다.

6-2 카운터

카운터 박스함수는 총 3종(CTU·CTD·CTUD)입니다. 타이머와 달리 틱 접두판(TA)이 없고, 경과 시간이 아니라 입력의 상승 엣지 횟수를 셉니다. CTU와 CTD는 문법이 같지만 (`<함수> <타깃> <프리셋>`), CTUD는 up·down 두 신호를 따로 받는 4-인자 문법 (`CTUD <타깃> <up 비트> <down 비트> <프리셋>`)을 씁니다.

- **타깃**: 카운트값을 저장할 카운터 메모리 주소입니다. `C<n>`(16비트, 카운트값 하나) 또는 `DC<n>`(32비트, `C<n>`과 `C<n+1>` 두 워드를 이어 씀)을 씁니다.
- **프리셋**: 완료까지 필요한 카운트 수입니다. `5`처럼 정수를 쓰거나, `D10`처럼 메모리 주소를 써서 실행 중에 HMI 등으로 바꿀 수 있습니다(6-2-4). 타이머와 달리 시간 문법(`1s`·`500ms` 등)은 지원하지 않습니다.

타이머와 마찬가지로 같은 이름이라도 자리에 따라 뜻이 다릅니다. 박스함수 자리(타깃)에 쓴 `C8`은 카운트값을 가리키지만, 같은 `C8`을 접점 자리에 놓으면 카운트값이 아니라 완료비트를 가리킵니다. 카운트값이 프리셋에 도달하면 완료비트가 ON되고, 그 상태를 `C8` 접점으로 읽어 다음 코일로 넘길 수 있습니다.

카운터 타깃(`C<n>`·`DC<n>`)을 RESET 코일(5-3-3)의 대상으로 지정하면, 일반 비트 해제가 아니라 카운트값과 완료비트를 함께 되돌리는 특수 의미로 동작합니다. 되돌아가는 값은 카운터 종류에 따라

다릅니다 — CTU·CTUD는 0으로 되돌아가지만, CTD는 설정된 프리셋값으로 되돌아갑니다(감산 카운터는 다시 셀 수 있도록 프리셋에서 출발해야 하므로, 아래 CTD 절 참고).

6-2-1 CTU — 가산 카운터

정의

CTU(Count Up, 가산 카운터)는 입력이 OFF에서 ON으로 바뀌는 상승 엣지가 발생할 때마다 카운트값을 1씩 늘리는 카운터입니다. 카운트값이 프리셋에 도달하면 완료비트가 ON되고, 그 뒤로 입력 상승 엣지가 더 들어와도 완료비트는 계속 켜진 채 카운트값만 계속 늘어납니다. 카운트값이 상한(16비트 65535, 32비트 4294967295)에 이르면 더 늘지 않고 그대로 유지됩니다(saturate).

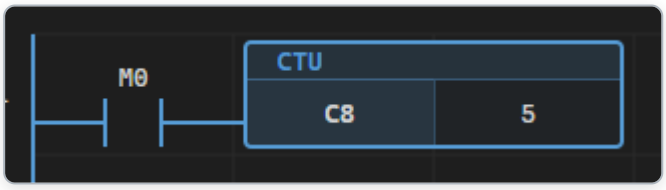
사용법

```
CTU <타깃> <프리셋>
```

`CTU C8 5`는 입력이 상승 엣지를 5번 만들면 완료비트가 켜집니다. CTU 헬퍼는 내부에 상승 엣지 검출을 내장하고 있어, 일반 NO 접점을 그대로 입력으로 두어도 접점이 OFF에서 ON으로 바뀔 때마다 한 번씩만 세어집니다(상승 검출 접점을 따로 앞에 둘 필요가 없습니다).

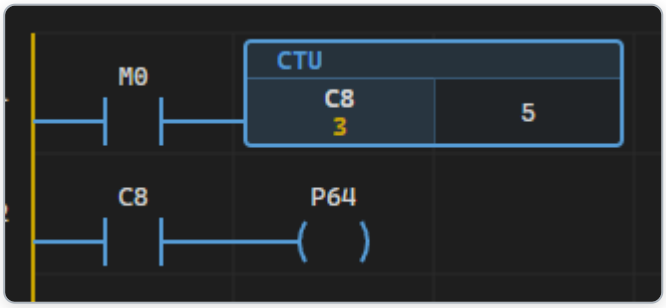
예제

`M0 NO 접점 → CTU C8 5`로 놓으면, M0가 OFF에서 ON으로 바뀔 때마다(상승 엣지) 카운트값이 1씩 늘고, 다섯 번째 상승 엣지에서 완료비트 C8이 ON됩니다. 상승 엣지 검출이 헬퍼 안에 들어 있어, 상승 검출 접점을 따로 앞에 두지 않고 일반 NO 접점을 그대로 입력으로 씁니다.

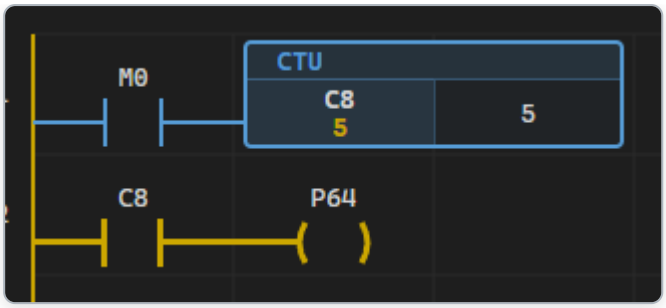


< 그림 6-9 CTU 예제 회로 — M0 NO 접점으로 CTU C8 5 구동 >

완료비트를 다른 rung에서 받으려면 `C8 NO 접점 → P32 코일`처럼 이어두면, 카운트값이 프리셋에 도달해 C8이 완료비트로 통전될 때 P32도 함께 켜집니다.



< 그림 6-10 CTU 모니터링 — M0 상승 엣지가 반복될 때마다 카운트값이 증가하는 모습 >



< 그림 6-11 CTU 모니터링 — 카운트값이 프리셋(5)에 도달해 완료비트 C8이 ON으로 통전 >

주의사항

> 프리셋은 정수나 메모리 주소만 쓸 수 있습니다(6-2-4). 타이머 프리셋과 달리 `1s`·`500ms` 같은 시간 문법은 지원하지 않으며, 그런 표기를 쓰면 컴파일 시 에러가 됩니다.

- 타깃 인덱스가 프로젝트에 설정된 카운터 메모리 개수를 넘으면 컴파일이 거부됩니다. 개수 조정은 **데이터 메모리**(2-1)의 "영역 크기와 메모리 설정"을 참고하세요.
- 완료비트가 켜진 뒤에도 입력 상승 엣지가 계속 들어오면 카운트값은 상한까지 계속 늘어납니다. 카운트값·완료비트를 되돌리려면 RESET 코일(위 도입 참고)을 씁니다.

6-2-2 CTD — 감산 카운터

정의

CTD(Count Down, 감산 카운터)는 **처음 실행될 때 카운트값이 프리셋으로 자동으로 채워지고**, 입력 상승 엣지가 발생할 때마다 카운트값을 1씩 줄이는 카운터입니다. 카운트값이 0에 도달하면 완료비트가 ON되고, 더 줄어들지 않고 0에서 그대로 유지됩니다 (saturate, 음수 없음). 운전 중 다시 프리셋으로 되돌리려면 RESET 코일로 해당 카운터를 리셋합니다(리셋할 때도 0이 아니라 프리셋으로 채워집니다). CTU와 반대 방향으로 세는 카운터입니다.

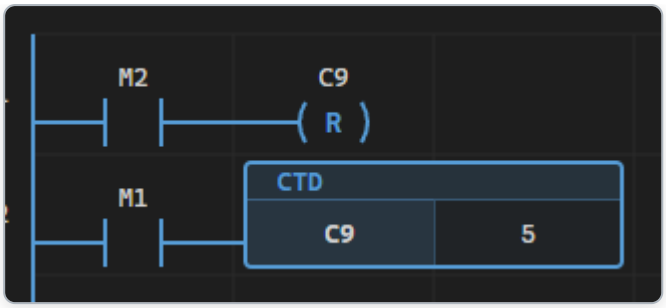
사용법

```
CTD <타깃> <프리셋>
```

`CTD C9 5`는 처음 실행 시 카운트값이 5로 채워지고, 입력 상승 엣지가 5번 발생해 카운트값이 0에 도달하면 완료비트가 켜집니다.

예제

`M1 NO 접점 → CTD C9 5`를 놓으면, 카운트값은 처음 실행 시 프리셋 5에서 시작하고 M1이 상승 엣지를 만들 때마다 1씩 줄어듭니다. 다섯 번째 엣지에서 카운트값이 0이 되어 완료비트 C9가 켜집니다. CTD도 CTU와 마찬가지로 상승 엣지 검출이 헬퍼에 내장돼 있어 NO 접점을 그대로 입력으로 씁니다. 그림처럼 `M2 NO 접점 → C9 RESET`을 함께 두면 운전 중 언제든 카운트값을 프리셋으로 되돌릴 수 있습니다(첫 실행 프리셋은 자동이므로 리셋을 두지 않아도 됩니다).



< 그림 6-12 CTD 예제 회로 — M1 NO 접점으로 CTD C9 5 구동(첫 실행 시 프리셋 5 자동 로드), M2 RESET으로 프리셋 재장전 >

주의사항

> CTD는 **처음 실행(부팅) 시 카운트값이 자동으로 프리셋으로 채워지므로**, 사용 전에 따로 리셋을 넣지 않아도 프리셋에서 감산이 시작됩니다. 리셋(RESET 코일)은 운전 중 카운트값을 다시 프리셋으로 되돌리고 싶을 때 씁니다.

- 프리셋은 정수나 메모리 주소(6-2-4)이고, 타깃 인덱스 범위 규칙은 CTU(6-2-1)와 동일합니다. 프리셋이 메모리일 때 CTD는 그 값을 부팅과 리셋 때만 읽습니다(6-2-4).
- **C 영역을 정전유지로 설정했다면 부팅 값은 프리셋이 아니라 저장된 값입니다.** 정수 프리셋은 부팅 시퀀스가 CTD 자동 프리셋 로드를 먼저 하고 그 다음에 EEPROM 정전유지 값을 복구하므로, 나중에 실행되는 정전유지 복구가 자동 프리셋을 덮어씁니다 — 정전 직전의 카운트값이 그대로 이어진다는 뜻이고, 이는 정전유지를 켠 목적 그대로입니다. 부팅 때마다 프리셋에서 시작하게 하려면 해당 카운터를 정전유지 대상에서 빼거나, 부팅 직후 한 번만 켜지는 신호로 RESET 코일(5-3-3)을 걸어 주세요.

6-2-3 CTUD — 가감산 카운터

정의

CTUD(Count Up-Down, 가감산 카운터)는 up 신호와 down 신호를 각각 별도의 비트 접점으로 받아, up 신호가 상승 엣지를 만들 때마다 카운트값을 1 늘리고 down 신호가 상승 엣지를 만들 때마다 1 줄이는 카운터입니다. CTU·CTD와 달리 rung 왼쪽에 놓는 접점은 카운트 입력이 아니라 인에이블(게이트) 역할만 하며, 이 접점이 꺼져 있으면 up·down 신호가 바뀌어도 카운트값이 움직이지 않습니다. 카운트값이 프리셋에 도달하면(up 방향) 완료비트가 켜지고, 상한·하한에서 각각 saturate합니다(위 도입 참고).

사용법

```
CTUD <타깃> <up 비트> <down 비트> <프리셋>
```

CTU·CTD와 달리 인자가 4개입니다. `CTUD C8 M1 M2 100`으로 놓으면 rung 왼쪽 접점이 인에이블, M1이 up 신호, M2가 down 신호, 100이 프리셋입니다. up·down 자리에는 일반 접점과 같은 입력

(P·M·D·C 등의 비트 주소, 타이머·카운터 완료비트 `T0` · `C0` 포함)을 그대로 쓸 수 있습니다.

예제

`M0 NO 접점 → CTUD C8 M1 M2 100`으로 놓으면, M0가 ON인 동안에만 M1의 상승 엣지에서 카운트 값이 1 늘고 M2의 상승 엣지에서 1 줍니다. rung 왼쪽 M0 접점은 카운트 입력이 아니라 인에이블(게이트) 역할이라는 점에 유의하세요.



< 그림 6-13 CTUD 예제 회로 — M0 인에이블 접점으로 CTUD C8 M1 M2 100 구동(up=M1, down=M2) >

주의사항

> up과 down이 같은 스캔에 동시에 상승 엣지를 만들면 둘 다 무시되어 카운트값이 그대로입니다(IEC 61131-3 표준 카운터와 같은 동작).

- 인에이블 접점이 꺼져 있는 동안 up·down 신호가 바뀌어도 그 변화는 이미 반영된 것으로 기록됩니다 — 인에이블이 다시 켜져도 꺼져 있던 동안의 변화는 다시 카운트되지 않습니다.
- 프리셋은 정수나 메모리 주소입니다(CTU·CTD와 동일, 타이머의 시간 문법 없음). 메모리 프리셋은 6-2-4를 보세요.
- 타깃 인덱스 범위 규칙은 CTU(6-2-1)와 동일합니다.

6-2-4 메모리로 프리셋 지정 (HMI에서 카운트 바꾸기)

정의

카운터의 프리셋 자리에 숫자 대신 **메모리 주소**를 쓰면, 카운터가 그 메모리의 값을 프리셋으로 읽습니다. 프로그램을 다시 올리지 않고도 HMI(터치 패널)나 통신으로 그 메모리에 값을 써서 목표 카운트를 바꿀 수 있습니다. 카운터 3종(CTU·CTD·CTUD) 모두, 타깃이 `C<n>`이든 `DC<n>`이든 같은 규칙입니다. 다만 값을 읽는 때가 CTU·CTUD와 CTD가 다릅니다(아래).

사용법

```
CTU <타깃> <프리셋 메모리>
CTD <타깃> <프리셋 메모리>
CTUD <타깃> <up 비트> <down 비트> <프리셋 메모리>
```

종류	쓸 수 있는 주소	예
워드 메모리	`D` `C` `T` `WP` `WM`	`CTU C8 D10`, `CTD C9 WM1`

종류	쓸 수 있는 주소	예
더블워드 메모리	`DD` `DC` `DT` `DP` `DM`	`CTU DC10 DD5`
심볼	위 주소를 가리키는 심볼	`CTU C8 Limit` (`Limit`이 `D10`을 가리킬 때)

쓸 수 없는 것은 컴파일 에러가 됩니다 — 시간 표기(`1s`), 실수(`R`), 비트(`M0`, `D0.5`, `WM0.3`), 8비트(`BM0`), 아두이노 변수, 식(`D10+1`), 메모리 범위 밖 주소. 프리셋 자리의 `C<n>`·`T<n>`은 완료비트가 아니라 카운트(경과값) 워드를 읽습니다.

예제

`M0 NO 접점 → CTU C8 D10`, `C8 NO 접점 → P32 코일`로 놓으면, M0가 OFF에서 ON으로 바뀔 때마다 카운트값이 1씩 늘고 카운트값이 D10에 든 값에 이르면 C8이 ON되어 P32가 켜집니다. HMI가 통신으로 D10에 `20`을 쓰면 카운트가 20이 될 때, `50`을 쓰면 50이 될 때 완료됩니다. 래더를 고치거나 다시 올릴 필요가 없습니다. D10에 값을 쓰기 전에 들어온 입력 엣지도 카운트에 쌓이므로, 값을 쓴 뒤 0부터 다시 세려면 RESET 코일(5-3-3)을 한 번 켜세요. D10이 0인 동안(전원을 넣은 직후 HMI가 아직 값을 쓰지 않았을 때)에는 완료하지 않아 P32가 켜지지 않습니다.

값이 뜻하는 것

- **값은 부호 없는 카운트입니다.** 모니터링에서 −1로 보이는 D10은 65535입니다.
- `C<n>`**(16비트) 타깃에 더블워드를 넣으면 65535에서 멈춥니다.** 더 큰 값을 넣어도 65535로 취급합니다. 32비트 `DC<n>` 타깃은 32비트 값을 그대로 씁니다.

값이 0일 때

메모리 프리셋의 값이 0이면 CTU·CTUD·CTD 모두 완료하지 않습니다. 전원을 넣었을 때 HMI가 아직 값을 쓰지 않았어도 완료비트를 받는 출력이 켜지지 않습니다. 상수 프리셋 0(`CTU C8 0`)은 메모리 프리셋과 달리 첫 판정에 바로 완료합니다.

CTU·CTUD

- HMI가 값을 쓰면 리셋 없이 그 값까지 셉니다. 값을 올리면 새 값까지 이어서 셉니다.
- 값이 0인 동안 들어온 입력 엣지도 카운트에 쌓입니다. 시작할 때 0부터 세려면 값을 쓴 뒤 RESET 코일(5-3-3)을 한 번 켜세요.
- 이미 카운트가 값 이상이면 입력 엣지가 없어도 다음 판정에서 완료됩니다(CTUD는 동작 조건인 인에이블 접점이 ON일 때 판정하므로, 꺼져 있었다면 켜지는 스캔에 완료됩니다).
- 완료 뒤 값을 올리거나 0으로 내려도 완료비트는 켜진 채입니다. RESET 코일로 리셋할 때까지 꺼지지 않고, 카운트값은 계속 셉니다.

CTD

- **메모리 값은 부팅 때와 리셋 때 카운트로 읽습니다.** 실행 중에 값을 바꿔도 다음 리셋까지 현재 카운트에는 영향이 없습니다. RESET 코일은 켜지는 순간의 메모리 값으로 카운트를 채웁니다.
- 완료는 입력으로 카운트가 1에서 0으로 내려갈 때 일어납니다.

- 값이 0일 때 리셋하면 카운트는 0이고 완료하지 않습니다.
- 카운트 번지에 `MOV`나 통신으로 직접 0을 써도 완료하지 않습니다.

> **CTD는 부팅 뒤에 HMI·통신·Arduino 코드(`setup()`)가 값을 쓰면 RESET 코일을 한 번 켜야 값이 로드됩니다.** 리셋하지 않으면 카운트는 부팅 때 읽은 값 그대로입니다.

CTD의 부팅

메모리 프리셋을 쓰는 CTD는 래더를 초기화할 때 정전유지 복구가 끝난 뒤에 한 번 리셋되어, 그 시점의 프리셋 메모리 값으로 카운트가 채워집니다. 그래서 프리셋 메모리를 정전유지로 두었다면 정전 전의 값이 그대로 들어갑니다.

경우	부팅 직후 카운트
정전유지를 쓰지 않음	그 시점의 프리셋 메모리 값. 보통 0이라 카운트 0으로 시작하고 **완료하지 않습니다**
프리셋 메모리만 정전유지	복구된 프리셋 값
카운터 `C`도 정전유지이고 복구에 성공	**저장된 카운트를 그대로** 유지합니다. 복구된 카운트가 0이고 프리셋 값이 0이 아니면 완료 상태로 복구하고, 복구된 카운트가 0이 아니면 이어서 셉니다

`DC<n>` 타깃은 두 워드가 **모두** 정전유지일 때만 저장된 카운트를 유지합니다. 한 워드만 정전유지이면 복구된 값을 믿지 않고 프리셋으로 채웁니다. 정전유지 복구를 건너뛰는 부팅(첫 부팅 등)도 프리셋으로 채웁니다.

> **부팅 뒤에 HMI·통신·Arduino 코드(`setup()`)가 써 넣는 값은 이 시점에 아직 없습니다.** 그 값이 들어오기 전의 프리셋 메모리는 0이라, CTD는 카운트 0으로 시작하고 완료하지 않습니다. 값을 쓴 뒤에 RESET 코일을 한 번 켜서 새 값으로 채우세요. 프리셋 메모리를 정전유지로 두면 이 문제가 없습니다.

> **카운터 `C`를 정전유지로 둘 때는 프리셋 메모리도 함께 정전유지로 두세요.** 프리셋 메모리가 정전유지가 아니면 부팅 때 값이 0이라 완료 상태가 복구되지 않습니다.
>
> **둘 다 정전유지일 때는 값을 처음 쓴 뒤 전원을 끄기 전에 RESET 코일을 한 번 켜세요.** 리셋하지 않고 전원을 다시 넣으면 카운트 0, 프리셋 값은 0이 아닌 상태이므로 완료 상태로 복구되어, 완료 코일의 출력이 전원을 넣자마자 켜집니다.

주의사항

- **핀에 연결된 번지는 부팅 때 0입니다.** `WP0`나 아날로그 핀과 겹치는 `C` 주소는 부팅 때 입력을 아직 읽지 않았으므로, CTD가 부팅에서 읽는 값은 0이고 카운트 0으로 시작해 완료하지 않습니

다. RESET 코일을 켜야 그때 읽은 입력값이 로드됩니다. 스캔이 시작된 뒤에는 입력을 읽은 값이 쓰이므로 CTU·CTUD는 이 제한이 없습니다.

- **더블워드 번호는 타깃과 프리셋이 다릅니다.** 카운터 타깃의 `DC2`는 `C[2]`, `C[3]`이지만, 프리셋 자리의 `DC1`이 `C[2]`, `C[3]`입니다. 프리셋에서는 더블워드가 식·MOV와 같은 규칙을 따라 `DC<n>` = `C[2n]`, `C[2n+1]`입니다. `DD<n>`도 같아서 `DD5`는 `D[10]`, `D[11]`입니다.
- 시간 단위가 없으므로 타이머 단위 변경 대화상자(6-1-6)와는 관계가 없습니다.

6-3 산술

산술 박스함수는 사칙연산 5종(ADD·SUB·MUL·DIV·MOD)과 증감 2종(INC·DEC), 스케일 변환(SCALE), 실수 연산 2종(FLOOR·SQRT)까지 총 10종입니다. 모두 지정한 대상 주소에 계산 결과를 저장하는 명령으로, 셀 왼쪽에 접점을 놓으면 그 조건이 참일 때만 계산하고, 접점을 놓지 않으면 코일과 마찬가지로 매 스캔 그대로 계산합니다.

계산에 쓰는 워드 메모리 **D·C·T는 부호 있는 16비트**라 표현 범위가 **-32768 ~ 32767**입니다(2-1-1). 이 범위를 넘는 계산은 넘친 자리가 잘려 부호가 뒤집힌 값이 되므로, 큰 수를 다룰 때는 32비트 더블워드 뷰(`DD0`·`DC0`·`DT0`)나 실수 영역 R을 쓰거나, 중간 계산 단계를 나눠 범위 안에 들어오게 하세요.

6-3-1 사칙연산 (ADD·SUB·MUL·DIV·MOD)

정의

ADD·SUB·MUL·DIV·MOD는 두 값을 사칙연산(덧셈·뺄셈·곱셈·나눗셈·나머지)한 결과를 지정한 주소에 저장하는 박스함수 5종입니다. 다섯 명령 모두 인자 개수와 자리가 완전히 같고 연산자만 다릅니다.

니모닉	연산자	의미
`ADD`	`+`	덧셈
`SUB`	`-`	뺄셈
`MUL`	`*`	곱셈
`DIV`	`/`	나눗셈
`MOD`	`%`	나머지(모듈로)

사용법

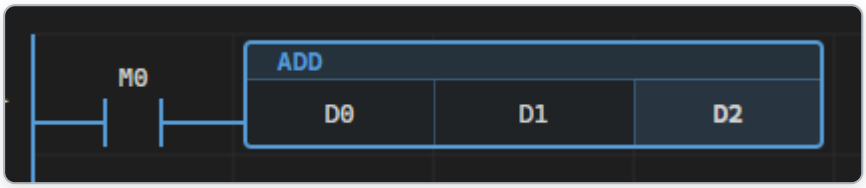
```
<함수> <S1> <S2> <D>
```

인자는 항상 4토큰이며 대상(D)은 마지막 토큰입니다. `ADD D0 D1 D2`는 `D2 = D0 + D1`로 변환되고, `MUL D0 2 D0`처럼 자기 자신을 대상으로 다시 쓸 수도 있습니다. S1·S2 자리에는 워드 주소

(`D0` · `C10` 등)와 정수 리터럴을 섞어 쓸 수 있고, 대상 D는 결과를 저장할 워드 주소여야 합니다.

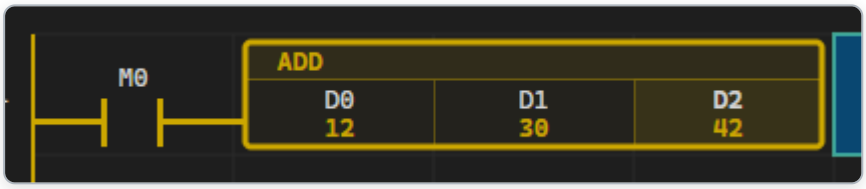
예제

`M0 NO 접점 → ADD D0 D1 D2`로 놓으면, M0가 ON인 스캔마다 D0와 D1을 더한 값이 D2에 저장됩니다.



< 그림 6-14 사칙연산 ADD 예제 회로 — M0 NO 접점 뒤에 ADD D0 D1 D2 박스함수를 놓은 모습 >

SUB·MUL·DIV·MOD도 셀에 적는 명령어만 바꾸면 되고, 배선과 인자 자리(S1 S2 D)는 완전히 같습니다. 같은 자리에 `SUB D0 D1 D2`로 놓으면 D2에 D0−D1이, `MUL D0 D1 D2`로 놓으면 D0×D1이 저장됩니다.



< 그림 6-15 ADD 모니터링 — M0 ON 시 D0+D1 결과가 D2에 반영되는 모습 >

주의사항

> **DIV·MOD에서 두 번째 값(S2)이 0이면 컴파일 단계에서 걸러지지 않습니다.** 실행 중에 0으로 나누는 상황이 생기지 않도록, 래더를 설계할 때 값의 범위를 직접 확인하세요.

- 대상(D)은 항상 마지막 토큰입니다. 뒤에서 설명할 SCALE(6-3-3)은 대상 위치가 다르므로 혼동하지 마세요.
- D 영역은 16비트 정수이므로(2-1) DIV는 소수점 이하를 버리는 정수 나눗셈입니다 — `DIV D0 D1 D2`에서 D0=7, D1=2면 D2는 3(3.5가 아닌)이 됩니다.

6-3-2 증감 (INC·DEC)

정의

INC·DEC는 대상 하나의 값을 1만큼 늘리거나 줄이는 단항 박스함수입니다. INC는 `D = D + 1`, DEC는 `D = D - 1`로 동작합니다.

사용법

```
INC <D>
DEC <D>
```

인자는 대상 하나뿐입니다. `INC D0`는 스캔마다 D0를 1 늘리고, `DEC D5`는 스캔마다 D5를 1 줄입니다.

예제

`M0 상승 검출 접점 → INC D0`로 놓으면, M0가 OFF에서 ON으로 바뀌는 상승 엣지에서만 D0가 1 늘어납니다. 상승 검출 접점(5-5-1) 뒤에 INC 박스함수를 놓았으므로, 접점이 켜져 있는 내내가 아니라 켜지는 순간에만 D0가 한 번씩 증가합니다.



< 그림 6-16 증감 INC 예제 회로 — M0 상승 검출 접점 뒤에 INC D0 박스함수를 놓은 모습 >

주의사항

> **INC·DEC 자체에는 엣지 검출 기능이 없습니다.** 접점 없이 `INC D0`만 놓으면 매 스캔(초당 수백~수천 회)마다 D0가 계속 늘어나 사실상 순식간에 값이 치솟습니다. 카운터 CTU(6-2-1)는 헬퍼 함수 안에 상승 엣지 검출이 내장돼 있어 일반 접점을 그대로 연결해도 되지만, INC·DEC는 그런 내장 검출이 없으므로 "조건이 켜질 때마다 1번만 증가"시키려면 상승 검출 접점(5-5-1)을 반드시 앞에 둬야 합니다. 자유입력형 대입식(`D0 = D0 + 1`, 상승 검출 접점(5-5-1) 예제)도 같은 이유로 상승 검출 접점과 함께 씁니다.

- 대상 인덱스가 프로젝트에 설정된 메모리 개수를 넘으면 컴파일이 거부됩니다.

6-3-3 SCALE — 스케일 변환

정의

SCALE은 입력값을 한 범위에서 다른 범위로 선형 변환(스케일링)하는 박스함수입니다. 예를 들어 아날로그 입력값을 0~100 범위의 백분율 값으로 바꾸는 데 씁니다. ILOGICS 코어에 번들된 스케일 함수를 그대로 호출하는 형태로 변환되며, 정수판 `Iscale`과 실수판 `Iscalef` 중 어느 쪽을 부를지는 앱이 자동으로 고릅니다.

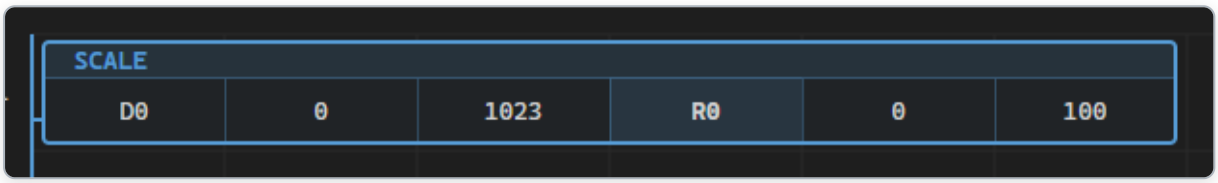
사용법

```
SCALE <IN> <IN_MIN> <IN_MAX> <OUT> <OUT_MIN> <OUT_MAX>
```

인자는 항상 6개(토큰 7개)이며, `IN`을 `IN_MIN`~`IN_MAX` 범위의 값으로 보고 `OUT_MIN`~`OUT_MAX` 범위로 비례 환산한 결과를 `OUT`에 저장합니다.

예제

`SCALE D0 0 1023 R0 0 100`으로 놓으면, D0(0~1023 범위로 가정)를 0~100 범위로 환산한 값이 R0에 저장됩니다. 앞에 접점을 두지 않았으므로 매 스캔 그대로 환산이 이뤄집니다. 여기서 쓴 입력 범위 `0~1023`은 보드에 내장된 온보드 아날로그 입력을 기준으로 한 예시입니다 — MPAINO 아날로그 입력 모듈(X)의 채널은 풀스케일이 **32767**이라 범위가 다릅니다. 읽어 오는 채널이 실제로 어떤 범위의 값을 내는지 확인한 뒤 `IN_MIN`·`IN_MAX`를 거기에 맞춰 적으세요.



< 그림 6-17 SCALE 예제 회로 — 접점 없이 SCALE D0 0 1023 R0 0 100 박스함수를 놓은 모습 >

주의사항

> **대상(OUT)이 마지막 토큰이 아니라 5번째 토큰입니다.** 앞의 사칙연산(6-3-1)이나 뒤의 MOV(6-4-1)는 이 장의 다른 박스함수와 마찬가지로 대상이 항상 마지막 토큰이지만, SCALE만 예외로 `OUT`이 중간(`IN_MIN`·`IN_MAX` 다음, `OUT_MIN`·`OUT_MAX` 앞)에 옵니다. 인자를 그대로 밀어 쓰면 엉뚱한 주소에 결과가 저장되니 순서를 확인하세요.

> **정수 연산과 실수 연산 중 어느 쪽으로 계산할지는 6개 인자 전체가 정합니다.** 인자 여섯 개(`IN`·`IN_MIN`·`IN_MAX`·`OUT`·`OUT_MIN`·`OUT_MAX`) 가운데 **하나라도 실수면** 실수판 `Iscalef`로, **전부 정수면** 정수판 `Iscale`로 계산합니다. 여기서 실수란 R 영역 주소(또는 R 영역을 가리키는 심볼)와 `2.5`·`1e3`처럼 소수점·지수 표기가 있는 리터럴을 말하고, `100` 같은 정수 리터럴은 실수로 보지 않습니다. 즉 `SCALE D0 0 1023 D1 0 100`은 처음부터 끝까지 정수 연산이라 소수점이 생기지 않고, 소수점을 살리고 싶다면 대상을 R로 두거나(`SCALE D0 0 1023 R0 0 100`) 범위 인자를 `0.0`·`100.0`처럼 실수로 적습니다.

- 결과는 **출력 범위를 벗어나지 않습니다(clamp).** `IN`이 `IN_MIN`~`IN_MAX` 밖의 값이어도 가장 가까운 경계에 해당하는 출력이 저장되므로, 센서값이 예상 범위를 넘쳐도 `OUT_MIN`~`OUT_MAX` 밖으로 튀지 않습니다.
- `IN_MIN`과 `IN_MAX`가 같으면 0으로 나누는 것을 피해 **`OUT_MIN`을 그대로** 돌려줍니다.
- 범위는 **역방향으로도 쓸 수 있습니다.** `OUT_MIN`이 `OUT_MAX`보다 크면(예: `SCALE D0 0 1023 R0 100 0`) 입력이 커질수록 출력이 작아지는 반비례 환산이 되고, 입력 범위를 `IN_MIN` > `IN_MAX`로 뒤집어 써도 같은 방식으로 처리됩니다.
- 대상이 D(정수) 영역이면 실수판으로 계산하더라도 **저장하는 순간 소수점이 잘립니다.** 소수점이 필요하면 대상을 R(실수, 2-1) 영역으로 두세요.

6-3-4 실수 연산 (FLOOR·SQRT)

정의

FLOOR·SQRT는 실수값에 수학 함수를 적용해 그 결과를 저장하는 단항 박스함수입니다. FLOOR는 소수점 이하를 버리는 내림(floor), SQRT는 제곱근(square root)입니다. 둘 다 C 표준 라이브러리(`math.h`)의 `floor()`·`sqrt()`를 그대로 호출합니다.

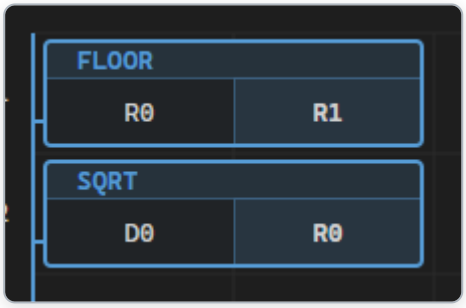
사용법

```
FLOOR <S> <D>
SQRT <S> <D>
```

`FLOOR R0 R1`은 R0의 내림값을 R1에, `SQRT D0 R0`은 D0의 제곱근을 R0에 저장합니다.

예제

접점 없이 `FLOOR R0 R1`을 놓으면 매 스캔 R0의 내림값이 R1에, `SQRT D0 R0`을 놓으면 D0의 제곱근이 R0에 저장됩니다. 사칙연산과 마찬가지로 대상은 마지막 토큰이며, 필요하면 앞에 접점을 두어 특정 조건에서만 계산하게 할 수도 있습니다.



< 그림 6-18 실수 연산 예제 회로 — 접점 없이 FLOOR R0 R1·SQRT D0 R0 박스함수를 놓은 모습 >

주의사항

- 결과가 실수이므로 대상은 R(실수, 2-1) 영역을 씁니다. D(정수) 영역을 대상으로 쓰면 소수점 이하가 잘립니다.
- SQRT에 음수를 입력하면 `math.h`의 `sqrt()` 동작을 그대로 따릅니다(정의되지 않은 결과) — 입력값이 음수가 되지 않도록 앞단에서 걸러 주세요.

6-4 전송

전송 박스함수는 값을 그대로 옮기는 MOV와, 여러 워드를 한 번에 옮기거나 채우는 GMOV·FMOV(블록 전송) 두 갈래로 나뉩니다. 두 갈래 모두 셀 왼쪽에 접점이 있으면 그 조건이 참일 때만 옮기고, 접점이 없으면 매 스캔 그대로 옮깁니다.

6-4-1 MOV — 값 전송

정의

MOV는 한 워드 값을 다른 주소로 그대로 옮기는 박스함수입니다. `D = S`로 동작합니다.

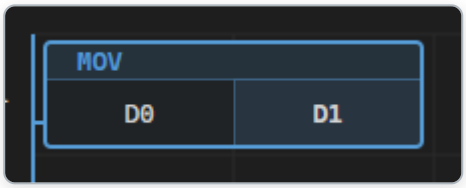
사용법

```
MOV <S> <D>
```

`MOV D0 D1`은 D0 값을 D1로 복사합니다. 소스(S) 자리에는 주소뿐 아니라 정수 리터럴도 쓸 수 있어(`MOV 100 D5` → `D5 = 100`), 상수를 워드 메모리에 로드하는 용도로도 씁니다.

예제

접점 없이 `MOV D0 D1`만 놓으면, 매 스캔마다 D0 값이 그대로 D1에 복사됩니다.



< 그림 6-19 MOV 예제 회로 — 접점 없이 MOV D0 D1 박스함수를 놓은 모습 >

주의사항

- 대상(D)은 마지막 토큰입니다(사칙연산 6-3-1과 동일, SCALE 6-3-3만 예외).
- 소스가 리터럴이면 매 스캔 같은 값이 계속 덮어써집니다 — 초기화를 한 번만 하고 싶다면 접점(예: 부팅 직후 한 스캔만 켜지는 신호)으로 게이팅하는 것을 고려하세요.
- 대상이 D·C·T 워드면 리터럴은 −32768~32767입니다. 그보다 큰(또는 작은) 리터럴은 DWORD 상수라 하위 16비트만 들어가고 경고가 뜹니다(`MOV 65535 C20` → C20 = −1, `MOV 70000 C20` → C20 = 4464). 단, 사용을 켠 아날로그·펄스(PWM) 출력 채널(Y·K 모듈 채널과 보드 내장 Aout·PWMout)로 가는 MOV는 0~65535를 경고 없이 그대로 받습니다 — `MOV 65535 C20`이면 Y 출력에 65535가 나갑니다(**표현 범위**(2-1-1)).

6-4-2 블록 전송 (GMOV·FMOV)

정의

GMOV·FMOV는 워드 여러 개를 한 번에 옮기는 박스함수입니다. GMOV(Group Move)는 소스에서 대상으로 N개 워드를 순서대로 그대로 복사하고, FMOV(Fill Move)는 소스 값 하나로 대상 N개 워드를 모두 같은 값으로 채웁니다.

사용법

```
GMOV <S> <D> <N>
FMOV <S> <D> <N>
```

인자 순서는 소스 → 대상 → 개수입니다. `N`(개수)은 정수 리터럴 또는 워드 주소 둘 다 쓸 수 있습니다.

예제

`GMOV D10 D20 5`는 D10~D14 다섯 워드를 D20~D24로 그대로 복사합니다. `FMOV D0 D100 10`은 D0 값 하나로 D100~D109 열 워드를 모두 채우고, `FMOV 0 D100 10`처럼 채울 값을 숫자로 직접 적어도 됩니다. 두 명령 모두 접점 없이 놓으면 매 스캔 실행되며, 인자 순서는 소스 → 대상 → 개수입니다.



< 그림 6-20 블록 전송 예제 회로 — 접점 없이 GMOV D10 D20 5·FMOV D0 D100 10 박스함수를 놓은 모습 >

주의사항

> **GMOV의 소스·대상과 FMOV의 대상은 D·C·T 워드 주소만 허용됩니다.** P·M 같은 비트 영역이나 R(실수) 영역, `.bit`·바이트/더블워드 뷰는 쓸 수 없습니다 — 그런 주소를 쓰면 컴파일이 거부됩니다.
>
> **FMOV의 채울 값(S)은** 정수 상수, 또는 `D` `C` `T` `WM` `WP`를 받습니다. 더블워드, 실수 `R`, 비트, 8비트 뷰는 컴파일 에러입니다. `WM`·`WP`는 채우기 전에 한 번만 읽어 그 값으로 채웁니다. 16비트 워드에 넣을 수 없는 상수(−32768~32767 밖)는 MOV(6-4-1)와 같이 하위 16비트만 들어가고 경고가 뜹니다.

- **개수(N)를 정수 리터럴로 적으면 블록의 끝 주소까지 검사합니다.** `GMOV D10 D20 5` 라면 소스 `D10`~`D14`와 대상 `D20`~`D24`가 모두 설정된 메모리 개수(2-1) 안에 있는지 확인하고, 하나라도 넘으면 컴파일이 거부됩니다. 직접 계산해 둘 필요가 없습니다.
- **개수(N)를 주소로 쓰면**(`GMOV D10 D20 D7`) 값이 실행 중에 정해지므로 빌드 때는 끝 주소를 검사하지 못합니다. 대신 실행 중에 **영역 끝까지만 전송합니다.** 소스(GMOV)나 대상의 끝을 넘는 만큼은 줄어들며, 오류 비트나 경고는 없습니다. N이 0이거나 음수면 전송하지 않습니다. 예를 들어 D 영역이 200개일 때 `GMOV D10 D20 D7`에서 D7이 500이면 D20~D199의 180워드만 전송합니다. N은 스캔마다 전송을 시작하기 전에 한 번만 읽습니다.

- **개수(N)가 `R`(실수) 값이고 소수이면 올림해서 전송합니다.** 2.5이면 3칸을 전송합니다.
- 개수가 줄어들어도 알려 주지 않으므로, **필요한 만큼 다 전송됐는지는 값을 직접 확인하세요.** N으로 들어갈 수 있는 최댓값에 시작 주소를 더한 값이 메모리 개수를 넘지 않는지 미리 따져 보는 것이 좋습니다.

6-5 비트연산

비트연산 박스함수는 논리 연산 4종(WAND·WOR·WXOR·WNOT), 시프트 2종(SHL·SHR), 회전 2종(ROL·ROR)까지 총 8종입니다. 비트 논리와 시프트는 산술(6-3)과 마찬가지로 셀에 적은 명령을 `대상 = 식` 형태의 대입식으로 바꿔 기존 검증된 식 변환 경로를 재사용하지만, 회전은 C에 순환 시프트 연산자가 없어 앱이 전용 인라인 수식으로 직접 처리합니다. 세 갈래 모두 셀 왼쪽에 접점을 놓으면 `if (조건) { ... }`로 감싸이고, 없으면 코일과 마찬가지로 조건 없이 매 스캔 그대로 실행됩니다.

6-5-1 비트 논리 (WAND·WOR·WXOR·WNOT)

정의

WAND·WOR·WXOR는 두 값을 비트 단위로 논리 연산(AND·OR·XOR)한 결과를 지정한 주소에 저장하는 박스함수입니다. WNOT은 값 하나를 비트 반전(NOT)한 결과를 저장하는 단항 박스함수입니다. 접점의 AND/OR 결합과 이름이 비슷하지만 서로 다른 개념이라, 워드 단위 연산이라는 뜻의 `W` 접두어로 구분합니다.

니모닉	연산자	인자	의미
`WAND`	`&`	이항	비트 AND
`WOR`	`|`	이항	비트 OR
`WXOR`	`^`	이항	비트 XOR
`WNOT`	`~`	단항	비트 반전(NOT)

사용법

```
<함수> <S1> <S2> <D>
```

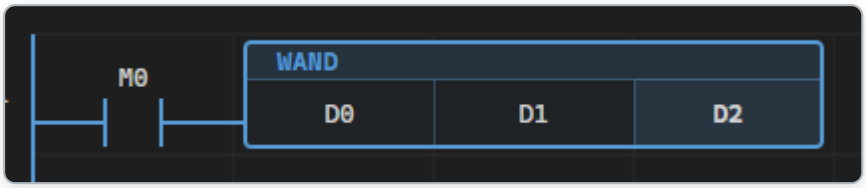
WAND·WOR·WXOR는 사칙연산(6-3-1)과 인자 자리가 같습니다 — 대상(D)이 항상 마지막 토큰입니다. WNOT은 소스 하나만 받으므로 형식이 다릅니다.

```
WNOT <S> <D>
```

`WAND D0 D1 D2`는 `D2 = D0 & D1`로, `WNOT D0 D1`은 `D1 = ~D0`로 변환됩니다.

예제

`M0 NO 접점 → WAND D0 D1 D2`로 놓으면, M0가 ON인 스캔마다 D0와 D1을 비트 AND한 값이 D2에 저장됩니다.



< 그림 6-21 비트 논리 WAND 예제 회로 — M0 NO 접점 뒤에 WAND D0 D1 D2 박스함수를 놓은 모습 >

WOR·WXOR는 셀에 적는 명령어만 바꾸면 되고, 배선과 인자 자리(S1 S2 D)는 완전히 같습니다 — 같은 자리에 `WOR D0 D1 D2`로 놓으면 D0와 D1의 비트 OR가, `WXOR D0 D1 D2`로 놓으면 비트 XOR가 D2에 저장됩니다. WNOT은 소스 하나만 받으므로 `WNOT D0 D1`처럼 놓으면 D0를 비트 반전한 값이 D1에 저장됩니다.

주의사항

> **WNOT과 뒤에서 다룰 INV(6-6-1)는 동작·결과가 완전히 같습니다.** 둘 다 소스 값을 비트 반전(NOT)해 대상에 저장하며, 팔레트 카테고리(비트연산/변환)만 다를 뿐이므로 어느 쪽을 골라도 상관없습니다.

- 대상(D)은 WAND·WOR·WXOR에서 항상 마지막 토큰이고, WNOT은 소스(S) 다음의 두 번째(마지막) 토큰입니다.

6-5-2 시프트 (SHL·SHR)

정의

SHL·SHR은 값을 왼쪽 또는 오른쪽으로 지정한 비트 수만큼 시프트한 결과를 저장하는 박스함수입니다. SHL은 왼쪽으로 미는 만큼 하위 비트를 0으로 채우고, SHR은 오른쪽으로 미는 만큼 상위 비트를 0으로 채웁니다.

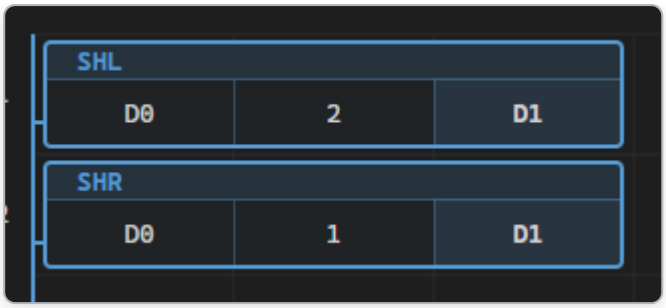
사용법

```
SHL <S> <n> <D>
SHR <S> <n> <D>
```

인자는 소스(S) → 시프트 비트수(n) → 대상(D) 순입니다. `n`은 정수 리터럴이나 워드 주소 모두 쓸 수 있습니다.

예제

접점 없이 `SHL D0 2 D1`과 `SHR D0 1 D1`을 두 rung으로 놓으면, 매 스캔마다 SHL은 D0를 왼쪽으로 2비트 시프트한 값을, SHR은 D0를 오른쪽으로 1비트 시프트한 값을 각각 D1에 저장합니다.



< 그림 6-22 시프트 예제 회로 — 접점 없이 SHL D0 2 D1·SHR D0 1 D1 박스함수를 놓은 모습 >

주의사항

> **SHR이 논리 시프트(zero-fill)가 되는 것은 소스가 D 영역 워드일 때뿐입니다.** D·C·T는 모두 부호 있는 16비트(`int16_t`)라 그대로 오른쪽 시프트하면 부호 비트가 퍼지는 산술 시프트가 됩니다. 앱은 **소스가 D 영역 워드 주소(또는 D 영역을 가리키는 심볼)인 경우에만** 부호 없는 값으로 캐스트해 상위 빈 자리를 0으로 채웁니다. 그래서 `SHR D0 1 D1`은 논리 시프트지만, `SHR C0 1 D1`·`SHR T0 1 D1`**은 산술 시프트**라 소스가 음수일 때 상위 자리가 1로 채워집니다. C·T 워드를 논리 시프트하려면 자유입력형(6-9)에서 `D1 = UW(C0) >> 1`처럼 캐스트를 직접 씌우세요.

> SHL은 소스 영역과 무관하게 하위 빈 자리를 0으로 채우므로 이 구분이 없습니다. 소스가 더블 워드 뷰(`DD0`)나 `.bit` 참조(`D0.3`)이면 그 값 자체가 음수가 될 수 없어, SHR을 써도 상위 자리가 0으로 채워집니다.

- 시프트 비트수가 16 이상이면 워드 폭(16비트)을 모두 밀어내 값이 0이 될 수 있습니다 — 16비트 워드 순환이 필요하면 다음 절의 회전(ROL·ROR)을 쓰세요.

6-5-3 회전 (ROL·ROR)

정의

ROL·ROR은 값을 16비트 워드 안에서 왼쪽 또는 오른쪽으로 순환시킨 결과를 저장하는 박스함수입니다. 시프트와 달리 밀려난 비트가 사라지지 않고 반대쪽 끝으로 다시 들어옵니다 — ROL은 왼쪽으로 밀려난 상위 비트가 최하위 비트로, ROR은 오른쪽으로 밀려난 하위 비트가 최상위 비트로 재진입합니다.

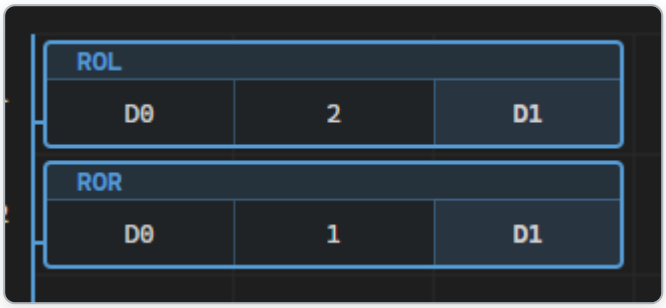
사용법

```
ROL <S> <n> <D>
ROR <S> <n> <D>
```

인자 순서는 SHL·SHR과 같습니다(소스 → 회전 비트수 → 대상). C에는 순환 시프트 연산자가 없어 시프트처럼 대입식으로 바꾸는 대신, 앱이 전용 인라인 마스킹 수식을 직접 생성합니다.

예제

접점 없이 `ROL D0 2 D1`과 `ROR D0 1 D1`을 두 rung으로 놓으면, 매 스캔마다 ROL은 D0를 왼쪽으로 2비트 순환시킨 값을, ROR은 D0를 오른쪽으로 1비트 순환시킨 값을 각각 D1에 저장합니다. 시프트와 달리 밀려난 비트는 사라지지 않고 반대쪽 끝으로 다시 들어옵니다.



< 그림 6-23 회전 예제 회로 — 접점 없이 ROL D0 2 D1·ROR D0 1 D1 박스함수를 놓은 모습 >

주의사항

> **회전은 항상 16비트 워드 단위로 순환합니다.** 회전 비트수(n)는 내부적으로 `n & 15`로 마스킹되어 0~15 범위로 고정됩니다(16 이상을 넣어도 나머지만 적용되어, 정의되지 않은 시프트 동작을 차단합니다) — `ROL D0 16 D1`과 `ROL D0 0 D1`은 결과가 같습니다(회전 없음).

- 대상(D)은 항상 마지막 토큰입니다(시프트·사칙연산과 동일).
- **쓸 수 있는 주소**: 소스(S)는 정수 상수, `D` `C` `T` `WM` `WP`이고, 대상(D)은 `D` `C` `T` `WM` `WP`입니다. 16비트 회전이라 더블워드(`DD` `DC` `DT` `DP` `DM`), 실수 `R`, 비트, 8비트 뷰(`BM` `BP`)는 컴파일 에러입니다. 16비트에 들어가지 않는 상수는 하위 16비트만 쓰고 MOV(6-4-1)와 같이 경고합니다.

6-6 변환

변환 박스함수는 반전 2종(INV·NEG), BCD 변환 2종(H2D·D2H)까지 총 4종입니다. INV는 비트연산(6-5-1)의 WNOT과 완전히 같은 대입식으로, NEG는 단항 마이너스 대입식으로 변환됩니다. H2D·D2H는 회전(6-5-3)과 마찬가지로 C에 대응하는 연산자가 없어 전용 인라인 수식으로 처리합니다. 네 명령 모두 셀 왼쪽에 접점이 있으면 조건으로 감싸이고, 없으면 조건 없이 매 스캔 실행됩니다.

6-6-1 반전 (INV·NEG)

정의

INV는 값을 비트 반전(NOT)한 결과를, NEG는 값의 2의 보수(부호 반전)를 구한 결과를 저장하는 단항 박스함수입니다.

니모닉	식	의미
`INV`	`~S`	비트 반전(NOT)
`NEG`	`-S`	2의 보수(부호 반전)

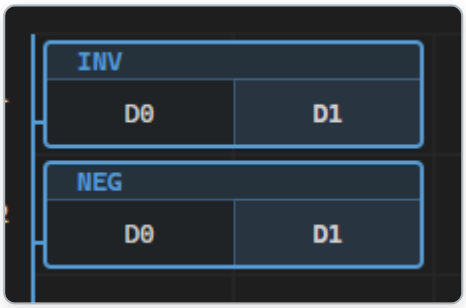
사용법

```
INV <S> <D>
NEG <S> <D>
```

인자는 소스(S) → 대상(D) 2개뿐입니다. `INV D0 D1`은 `D1 = ~D0`로, `NEG D0 D1`은 `D1 = -D0`로 변환됩니다.

예제

접점 없이 `INV D0 D1`과 `NEG D0 D1`을 두 rung으로 놓으면, 매 스캔마다 INV는 D0를 비트 반전한 값을, NEG는 D0의 부호를 반전(2의 보수)한 값을 각각 D1에 저장합니다.



< 그림 6-24 반전 예제 회로 — 접점 없이 INV D0 D1·NEG D0 D1 박스함수를 놓은 모습 >

주의사항

> **INV는 비트연산(6-5-1)의 WNOT과 동작·결과가 완전히 같습니다.** 둘 다 소스 값을 비트 반전(NOT)해 대상에 저장하며, 팔레트 카테고리(비트연산/변환)만 다를 뿐이므로 어느 쪽 명령어를 골라도 상관없습니다.

- 대상(D)은 항상 마지막 토큰입니다.

6-6-2 BCD 변환 (H2D·D2H)

정의

H2D·D2H는 16진 표현(hex)과 BCD(Binary-Coded Decimal, 4비트 니블 하나에 10진 숫자 하나를 담는 표현) 사이를 서로 변환하는 박스함수입니다. H2D는 BCD 디코드(예: `0x1234` → 1234), D2H는 그 반대인 BCD 인코드(1234 → `0x1234`)입니다.

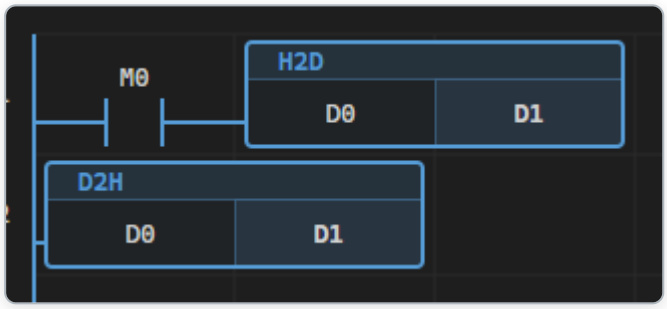
사용법

```
H2D <S> <D>
D2H <S> <D>
```

인자는 소스(S) → 대상(D) 2개뿐입니다. D 영역은 16비트(4개 니블)이므로 한 번에 4자리 BCD까지 다룹니다.

예제

`M0 NO 접점 → H2D D0 D1`로 놓으면, M0가 ON인 스캔마다 D0의 각 니블을 10진 자리값으로 가중합한 결과가 D1에 저장됩니다. 접점 없이 `D2H D0 D1`을 놓으면 반대 방향(10진 → BCD)으로 변환됩니다.



< 그림 6-25 BCD 변환 예제 회로 — M0 NO 접점 뒤 H2D D0 D1과 접점 없이 놓은 D2H D0 D1 >

주의사항

> **4자리 BCD 범위를 벗어난 값은 의도한 대로 변환되지 않습니다.** H2D는 소스의 각 니블(0~F)을 10진 자리로 그대로 가중합하므로, 니블 값이 9를 넘으면(예: `0x1A00`의 `A`) BCD가 아닌 순수 16진값이라 결과가 어긋납니다. D2H는 소스가 0~9999 범위를 벗어나면 `% 10` 연산으로 자리 올림이 잘려 나갑니다 — 두 명령 모두 입력값이 정상 BCD(H2D) 또는 0~9999(D2H) 범위인지 미리 확인하세요.

- 대상(D)은 항상 마지막 토큰입니다.
- **쓸 수 있는 주소**: 소스(S)는 정수 상수, `D` `C` `T` `WM` `WP`이고, 대상(D)은 `D` `C` `T` `WM` `WP`입니다. 4자리 BCD 명령이라 더블워드(`DD` `DC` `DT` `DP` `DM`), 실수 `R`, 비트, 8비트 뷰(`BM` `BP`)는 컴파일 에러입니다. 16비트에 들어가지 않는 상수는 하위 16비트만 쓰고 MOV(6-4-1)와 같이 경고합니다.
- D2H는 0~9999 값에 쓰세요. 그 밖의 값은 결과를 보장하지 않습니다.

6-7 비트/래치

비트/래치 박스함수는 SET·RST 2종으로, 대상 비트 하나를 켜거나 끄는 것이 전부인 가장 단순한 박스함수입니다. 다른 카테고리처럼 값을 계산하거나 변환하는 것이 아니라, 접점 종류 장의 SET 코일·RESET 코일(5-3-2, 5-3-3)과 **완전히 같은 동작을 합니다.** 그래서 이 카테고리는 다른 어떤 박스함수보다도 셀 코일과 가까운, 사실상 "코일을 박스함수 자리에 놓은 것"과 같은 명령입니다.

6-7-1 SET·RST

정의

SET·RST는 각각 대상 비트 하나를 ON 또는 OFF로 기록하는 박스함수입니다. SET은 조건이 ON되는 순간 대상을 ON으로 기록하고, 그 뒤 조건이 다시 OFF로 돌아가도 값을 그대로 유지합니다(래치). RST는 조건이 ON되는 순간 대상을 OFF로 기록(해제)하며, 마찬가지로 조건이 꺼져도 값을 유지합니다. 접점 종류 장의 SET 코일·RESET 코일(5-3-2, 5-3-3)과 동작이 완전히 같습니다 — 박스함수 SET/RST는 셀 SET 코일·RESET 코일을 박스함수 자리에 옮겨 놓은 것과 같아, 대상을 켜고 끄고 유지(래치)하는 동작이 서로 다르지 않습니다.

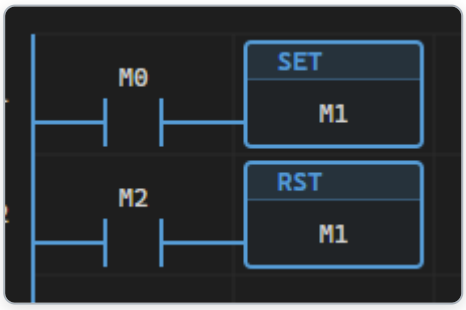
사용법

```
SET <타깃>
RST <타깃>
```

인자는 대상 하나뿐입니다 — 프리셋이 없어 타이머·카운터의 3~4토큰 문법과 다릅니다. 대상 자리에는 코일과 마찬가지로 P·M 등 비트 주소를 씁니다.

예제

`M0 NO 접점 → SET M1`, `M2 NO 접점 → RST M1` 두 rung을 놓으면, M0가 ON되는 순간 M1이 ON으로 기록되고 M0가 OFF로 돌아가도 유지되다가, M2가 ON되는 순간 M1이 OFF로 풀립니다.



< 그림 6-26 SET·RST 예제 회로 — M0 NO 접점 뒤 SET M1과 M2 NO 접점 뒤 RST M1 >

이 동작은 접점 종류 장(5-3-2/5-3-3)에서 본 셀 SET 코일·RESET 코일과 대상 주소만 다를 뿐 완전히 같습니다. 이번 예제는 내부 릴레이 M1이 대상이라 내부 비트만 기록하지만, 등록된 출력 핀(P32 등)을 대상으로 하면 SET 코일 예제(5-3-2, 그림 5-10)처럼 실제 물리 핀 출력까지 함께 갱신됩니다 — 대상이 핀이냐 내부 릴레이냐에 따른 차이일 뿐, SET/RST 자체의 동작 차이는 아닙니다.

주의사항

> **SET은 조건이 OFF로 돌아가도 대상이 OFF되지 않습니다.** 값을 끄려면 같은 주소를 대상으로 하는 RST를 별도로 둬야 합니다 — SET 코일(5-3-2)과 정확히 같은 주의사항입니다.

- 타이머(T)·카운터(C) 주소를 RST의 대상으로 지정하면 일반 비트 해제가 아니라 경과값(또는 카운트값)과 완료비트를 함께 리셋하는 특수 의미로 동작합니다 — 타이머 (6-1)·카운터(6-2)에서 다룬 RESET 코일의 특수 의미와 같습니다(RST도 코일과 같은 emit 경로를 타므로 동일하게 적용됩니다).
- SET·RST는 인자가 하나뿐이므로, 타이머·카운터처럼 프리셋을 덧붙여 쓰면(`SET M1 5` 등) 인자 개수 불일치로 컴파일이 거부됩니다.

6-8 로직 제어

로직 제어 박스함수는 조건부 점프 JMP·JMPN, 마스터 컨트롤 MCS·MCSCLR 총 4종입니다. 지금까지 본 명령이 모두 자기 rung 안에서 값 하나를 갱신하고 끝났다면, 이 카테고리는 ladder 전체의 실행 흐름(어느 rung을 이번 스캔에 건너뛸지, 어느 구간을 통째로 게이트할지)을 바꾸는 유일한 박스함수 그룹입니다.

JMP·JMPN은 분기 회로에서도 쓸 수 있습니다 — 한 접점이 박스함수 여러 개를 먹이는 자리에 놓아도 단일 rung과 똑같이 그 가지의 조건으로 점프합니다. 반면 MCS·MCSCLR은 **분기 회로에서 쓸 수 없습니다** — 자기 rung이 아니라 뒤따르는 rung들에 걸리는 구간 게이트라, 분기에 놓이면 옆 가지의 코일이 구간 안인지 밖인지가 정해지지 않기 때문입니다. 분기 자리에 놓으면 컴파일이 거부되므로 MCS·MCSCLR은 rung 하나를 통째로 차지하게 두세요.

6-8-1 조건부 점프 (JMP·JMPN)

정의

JMP·JMPN은 rung 왼쪽 조건에 따라 지정한 라벨 위치로 건너뛰는(점프) 박스함수입니다. JMP는 조건이 ON이면 점프하고, JMPN은 조건을 반전해 조건이 OFF일 때 점프합니다. 점프가 일어나면 점프 rung과 라벨 rung 사이에 있는 rung들이 이번 스캔에서 통째로 건너뛰어져 실행되지 않습니다 — 그 rung의 코일·박스함수가 갱신하던 값은 갱신되지 않고 직전 스캔 값을 그대로 유지합니다(hold).

라벨은 점프 도착 지점을 표시하는 전용 셀입니다. 왼쪽에 접점 없이 Func 셀 하나만 놓고, 코드 칸에 점프 대상 이름(예: `SKIP`)만 적으면 그 rung이 라벨로 인식됩니다 — 단, 같은 이름을 참조하는 JMP 또는 JMPN이 그 ladder 안에 하나라도 있어야 라벨로 인식됩니다. 참조가 없는 bare 식별자 셀은 라벨이 아닌 일반 식으로 해석되어 빌드 때 효과 없는 식이라는 오류가 납니다. 화면에서 라벨 박스는 띠에 `LABEL`이, 그 아래 칸에 라벨 이름이 표시됩니다.

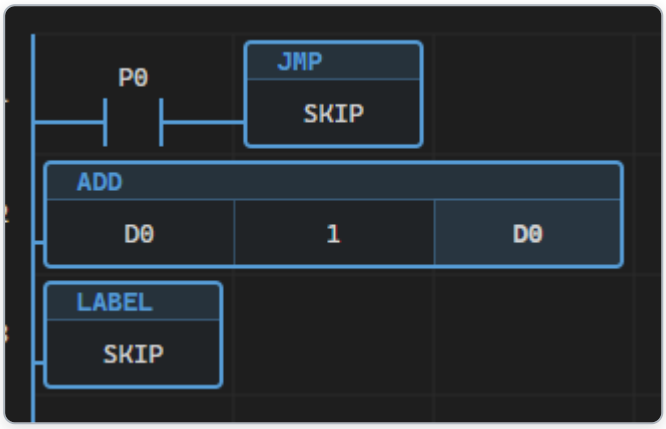
사용법

```
JMP <라벨>
JMPN <라벨>
```

라벨명은 영문자 또는 밑줄로 시작하고 그 뒤로 영숫자·밑줄만 오는 식별자입니다 (`^[A-Za-z_][A-Za-z0-9_]*$`) — 그대로 C 라벨로 쓰이므로 C 식별자 규칙과 같습니다.

예제

`P0 NO 접점 → JMP SKIP`, 그다음 rung `ADD D0 1 D0`(건너뛸 대상), 마지막 rung에 라벨 `SKIP`을 놓으면, P0가 ON인 스캔에서는 ADD rung이 건너뛰어져 D0가 갱신되지 않고 직전 값을 유지합니다.



< 그림 6-27 조건부 점프 예제 회로 — P0 NO 접점 뒤 JMP SKIP, 건너뛸 ADD D0 1 D0, 도착 라벨 SKIP >

JMPN은 조건이 반전되어, 반대로 P0가 OFF인 스캔에서 SKIP으로 점프합니다.

주의사항

> **점프 대상 라벨이 정의돼 있지 않거나, 같은 라벨명이 두 번 이상 정의되면 컴파일이 거부됩니다.** 추측으로 점프 대상을 짐작해 넘어가지 않고, 명확한 라벨 짝만 허용합니다.

- 마스터 컨트롤(MCS~MCSCLR, 6-8-2) 구간 안에도 JMP·JMPN을 놓을 수 있습니다. 구간이 닫혀 있는(게이트 OFF) 동안은 JMP·JMPN 모두 **점프하지 않습니다** — 점프 명령 자체가 실행되지 않으므로 구간 안 출력은 MCS 규칙대로 처리됩니다.
- 단, 점프가 MCS 구간 경계를 가로지르면 컴파일이 거부됩니다. JMP/JMPN과 그 라벨은 같은 구간 안(또는 둘 다 구간 밖)에 두고, 그 사이에 MCS·MCSCLR rung이 끼지 않게 배치하세요.

6-8-2 마스터 컨트롤 (MCS·MCSCLR)

정의

MCS·MCSCLR은 그 사이 구간에 있는 모든 코일·박스함수를 한 조건으로 일괄 게이트하는 박스함수 짝입니다. MCS는 구간 시작, MCSCLR은 구간 종료를 표시하며, MCS를 놓은 rung의 조건이 OFF이면 구간 안의 출력 코일은 원래 조건과 무관하게 모두 OFF로 강제됩니다 (SET·RESET 코일과 SET·RST 박스함수는 예외 — 아래 주의사항 참고). 여러 MCS를 겹쳐 놓아(중첩) 더 좁은 구간을 다시 게이트할 수 있고, 이 중첩 우선순위를 레벨 번호(0~15)로 관리합니다.

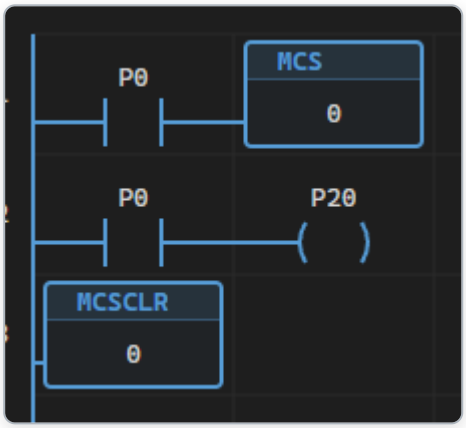
사용법

```
MCS <레벨>
MCSCLR <레벨>
```

레벨은 0~15 범위의 정수입니다. `MCS <k>`로 연 구간은 레벨 k 이하(같거나 더 낮은 번호)의 `MCSCLR`로 닫습니다 — `MCSCLR <n>`은 레벨 n 이상(더 안쪽에 중첩된 구간 포함)인 열린 구간을 한꺼번에 모두 닫습니다. 중첩할 때는 안쪽 MCS의 레벨이 바깥보다 커야 합니다 — 이미 열린 구간과 같거나 낮은 레벨로 MCS를 다시 열면 컴파일이 거부됩니다.

예제

`P0 NO 접점 → MCS 0`, 그다음 rung `P0 NO 접점 → P20 코일`, 마지막 rung `MCSCLR 0`으로 놓으면, P0가 ON일 때만 구간 안 P20 코일이 원래 조건대로 동작하고, P0가 OFF면 P20이 강제로 OFF됩니다.



< 그림 6-28 마스터 컨트롤 예제 회로 — P0 NO 접점 뒤 MCS 0, 게이트 대상 P20 코일, 구간 종료 MCSCLR 0 >

MCS를 중첩하면 안쪽 구간은 바깥 게이트가 이미 반영된 상태에서 자기 조건을 다시 결합해 게이트됩니다 — 안쪽 구간의 코일은 바깥 조건까지 따로 신경 쓸 필요 없이, 바깥이 닫히면 안쪽도 함께 닫힙니다.

주의사항

> **MCS와 MCSCLR의 레벨은 짝이 맞아야 합니다.** 대응하는 MCS 없이 MCSCLR만 두거나 레벨 번호를 어긋나게 겹치면 컴파일이 거부됩니다. 여는 순서의 역순으로 닫는(가장 안쪽부터 닫는) 중첩 구조를 지키세요.

- 레벨은 0~15 범위를 벗어나면(음수이거나 16 이상) 컴파일이 거부됩니다.
- SET 코일·RESET 코일(5-3-2, 5-3-3)과 박스함수 SET·RST(6-7)는 구간이 닫혀 있는 (게이트 OFF) 동안 실행되지 않고 **결과를 그대로 유지**합니다. 구간이 닫히기 전에 SET으로 켜 둔 비트는 구간이 닫혀도 꺼지지 않으며, 끄려면 구간이 열린 상태에서 RST가 실행되거나 구간 밖의 RST로 해제해야 합니다.

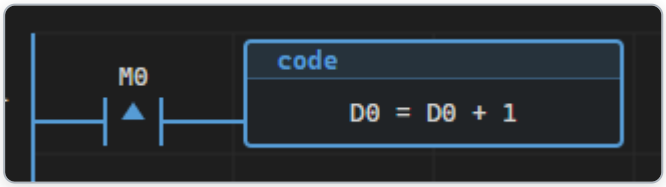
- 타이머·카운터가 MCS 구간 안에 있으면, 구간이 닫혀 있는(게이트 OFF) 동안은 그 rung의 입력이 꺼진 것과 똑같이 취급됩니다 — 결과는 종류마다 평소의 "입력 OFF" 동작을 그대로 따릅니다. TON·TOFF·TPL·TMON처럼 입력 OFF만으로 자연히 리셋되거나 상태가 바뀌는 종류는 게이트가 닫혀도 마찬가지로 그 동작을 합니다. 반면 적산 동작인 TMR은 입력 OFF에서 경과값을 그대로 보존하므로(**TMR — 적산 타이머**(6-1-5) 참고), 게이트가 닫혀도 경과값이 리셋되지 않고 그 자리에서 멈추기만 합니다. 카운터(CTU·CTD·CTUD)도 입력 OFF 취급이라 새 상승 엣지가 검출되지 않아 카운트가 멈추며, 카운트값 자체는 리셋되지 않습니다.

6-9 자유입력형 박스함수

지금까지 다룬 명령은 모두 팔레트에서 골라 쓰는 정형 명령어였습니다. 자유입력형은 팔레트를 거치지 않고 편집창 입력칸에 직접 C에 가까운 식을 써 넣는 방식으로, 크게 두 갈래로 나뉩니다. 화면에 그려지는 모양은 갈래마다 다릅니다 — 대입식은 칸으로 나뉘지 않아 명령어 띠에 `code`가 찍히고 그 아래에 적어 넣은 식이 한 줄로 그대로 나오며, 함수 호출은 띠에 함수 이름이, 그 아래에 인자가 칸마다 하나씩 나옵니다.

대입식은 `대상 = 식` 형태로, 접점 종류 장의 상승 검출 접점(5-5-1) 예제에서 이미 쓴 `D0 = D0 + 1`이 가장 단순한 형태입니다. 좌변(대상)에는 워드 주소(`D0`·`R0` 등)뿐 아니라 워드 안의 비트 하나(`D0.5`)나 P·M 영역의 폭 뷰 비트(워드 메모리의 비트 접점(5-8) 참고)도 그대로 쓸 수 있고, 우변에는 사칙연산·비교 등 임의로 조합한 식을 쓸 수 있습니다. `R0 = REAL(D0) / 10.0`처럼 아래에서 다룰 캐스트와 산술을 섞어 쓰는 것도 대입식의 한 형태입니다. 주소 대신 등록한 심볼 이름(9-5)을 써도 됩니다.

> **좌변에 P·M 영역의 폭 뷰 비트(`WP0.3` 등)를 쓸 때는 그 핀을 먼저 켜 두세요. 핀 편집**(2-4)의 "보드 핀맵 편집"에서 해당 핀의 `사용`을 체크하지 않으면, 폭 뷰로 쓴 비트는 에러도 경고도 없이 **조용히 0으로 처리**됩니다. 같은 핀을 `P0`처럼 단일 주소로 쓰면 컴파일 단계에서 "사용 안 함" 에러로 걸리지만, 폭 뷰는 그 검사를 타지 않아 회로가 조용히 동작하지 않는 상태가 됩니다. 폭 뷰 비트를 쓰기 전에 **그 핀이 켜져 있는지 반드시 확인**하세요.



< 그림 6-29 자유입력형 대입식 예제 회로 — M0 상승 검출 접점 뒤에 D0 = D0 + 1 대입식을 놓은 모습 >

임의 함수 호출은 대입 없이 함수 호출 하나만 써 넣는 형태입니다(`setMotor(100)`, `doStuff()`). 반환값을 저장할 필요 없이 부수효과(모터 구동, 상태 갱신 등)만 필요할 때 씁니다. 물론 `D0 = readSensor()`처럼 대입식의 우변에 함수 호출을 써서 반환값을 저장할 수도 있습니다. 호출하는 함수는 아두이노 내장 함수든 같은 프로젝트의 다른 .ino 탭에 정의한 사용자 함수든 상관없습니다.



< 그림 6-30 자유입력형 함수 호출 예제 회로 — M0 NO 접점 뒤에 setMotor(100) 호출을 놓은 모습 >

> **앱은 함수 이름이 실제로 존재하는지, 인자 개수가 맞는지 검사하지 않습니다.** 이 검증은 그대로 arduino-cli 컴파일 단계에 위임됩니다 — 존재하지 않는 함수를 쓰면 앱은 코드를 그대로 생성하고, 빌드할 때 컴파일러가 에러를 냅니다.

마지막으로, 값의 폭·부호를 그 자리에서 바꾸는 캐스트 유사함수 5종이 있습니다. 모두 인자 하나만 받는 단항 형태이고, 대입 좌변(lvalue)으로는 쓸 수 없습니다.

함수	대응 C 캐스트	의미
`REAL`	`(float)`	정수 → 실수
`DW`	`(int32_t)`	부호 있는 32비트 정수
`WD`	`(int16_t)`	부호 있는 16비트 정수
`UD`	`(uint32_t)`	부호 없는 32비트 정수
`UW`	`(uint16_t)`	부호 없는 16비트 정수

예를 들어 `R0 = REAL(D0) / 10.0`은 D0(정수)를 실수로 캐스트한 뒤 10.0으로 나눠, 소수점 이하가 보존된 결과를 R0에 저장합니다. 캐스트는 `REAL(WD(D0))`처럼 겹쳐 쓸 수도 있어, 부호나 폭을 먼저 바로잡은 뒤 실수로 바꾸는 것도 가능합니다.

> **대입도 함수 호출도 아닌 "맨 식"은 거부됩니다.** `D0 + 1`처럼 계산 결과를 어디에도 쓰지 않고 그대로 셀에 적으면, 아무 효과가 없는 코드로 간주해 컴파일 단계에서 에러가 됩니다. 값을 계산했으면 반드시 대상에 대입하거나(`D1 = D0 + 1`), 부수효과가 있는 함수 호출로 써야 합니다.

- 캐스트 이름은 대문자만 인식합니다 — 소문자로 쓰면(`real(D0)`) 캐스트가 아니라 그 이름의 일반 함수 호출로 취급되어, 실제 그런 함수가 없으면 빌드 단계에서 에러가 됩니다.
- 대입식에 상승 검출 접점 없이 자기 자신을 갱신하는 식(`D0 = D0 + 1`)을 놓으면, 조건이 켜져 있는 매 스캔마다 계속 실행됩니다 — 한 번만 증가시키려면 증감(6-3-2)과 마찬가지로 상승 검출 접점(5-5-1)을 앞에 둬야 합니다(위 상승 검출 접점(5-5-1) 예제와 같은 이유).

7 코드 편집과 자동완성

MPINO Studio 2는 래더 로직과 함께 **Arduino C 코드**를 직접 편집합니다. 이 장은 코드 에디터의 기본, 코드 탭(`.ino`) 운용, 자동완성과 정의로 이동(F12), 편집 대상에 따라 달라지는 찾기/바꾸기, 그리고 하단 패널 읽는 법을 다룹니다. 하나의 작은 예제 — `setMotor()` 헬퍼 함수를 코드 탭에 작성해 자동완성·F12로 확인하고 래더 박스함수 `setMotor(100)`에서 부르는 흐름 — 을 여러 절에 걸쳐 이어 가며 코드와 래더가 어떻게 맞물리는지 보여 줍니다.

7-1 코드 에디터

MPINO Studio 2의 코드 에디터는 Monaco 기반 C/C++ 편집기입니다. 새 프로젝트를 만들면 `main` 이라는 코드 탭 하나가 기본으로 열리며, 이 탭의 이름은 프로젝트 파일명과 항상 같이 바뀝니다. 글꼴은 Cascadia Code 14px 고정폭이고, 왼쪽에 줄 번호, 오른쪽에 미니맵이 표시됩니다. 줄 번호 옆 여백(gutter)에는 코드 폴딩 화살표가 항상 보여, 함수나 블록을 접었다 펼 수 있습니다. Tab 키는 스페이스 4칸으로 들어가고, 커서가 놓인 현재 줄은 옅게 강조됩니다.

래더와의 연결

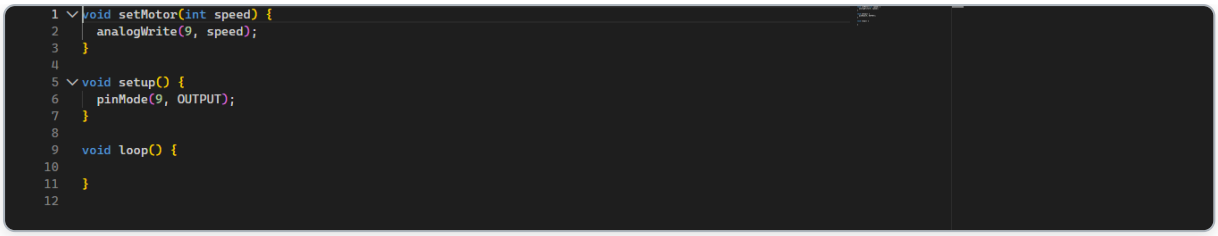
래더를 쓰는 프로젝트에서는 코드 맨 위에 자동으로 생성된 `#include` 줄들(`mpino_runtime.h` · `ladder_generated.h` · `ino_shared.h`, 필요하면 `symbols_generated.h`)이 들어갑니다. 에디터에서는 이 블록이 감춰지고 줄 번호가 1번부터 다시 매겨집니다 — 보기 편의일 뿐 실제 파일에는 그대로 있으며, 저장·빌드 에도 변함없이 반영됩니다. 래더를 쓰지 않는(순수 아두이노) 프로젝트에는 이 블록이 애초에 없어 감출 것도 없습니다.

`setup()` · `loop()` 뼈대도 함께 생성되는데, `loop()` 안의 `ladderLoop()` 호출은 래더 로직을 매 스캔 실행시키는 진입점입니다. 이 호출을 지우면 래더가 더 이상 실행되지 않습니다 — 트랜스파일러는 사용자가 쓴 코드를 고쳐 쓰지 않으므로, 이 호출을 유지하는 것은 사용자 책임입니다(래더가 어떻게 코드가 되는지는 **래더는 어떻게 아두이노 C 코드가 되나**(1-6) 참고). `ladderLoop()`를 넣고 빼는 메뉴는 코드 탭이 아니라 래더 탭에 있으며, **코드 탭 운용**(7-2)에서 다룹니다.

우클릭 메뉴

코드 에디터에서 마우스 우클릭을 하면 컨텍스트 메뉴 맨 위에 **업로드**가 있어, 곧바로 빌드·업로드를 한 번에 실행할 수 있습니다. 빌드·업로드 자체의 자세한 절차는 **포트 선택과 업로드**(1-8)와 **실행**(3-4)를 참고하세요.

> 코드 폴딩, 미니맵, 생성 `#include` 숨김은 모두 **보기 편의** 기능일 뿐 저장·빌드 결과에는 영향을 주지 않습니다. 실제 `.ino` 파일에는 감춰진 `#include` 줄이 그대로 들어 있습니다.



< 그림 7-1 코드 에디터 — setMotor 함수 작성 화면. 왼쪽 줄번호와 폴딩 화살표, 오른쪽 미니맵 >

7-2 코드 탭 운용

코드 그룹

프로젝트의 `.ino` 코드 파일들은 왼쪽 사이드바 **탐색기**(Explorer)의 **코드** 그룹에 탭으로 나열됩니다. 래더 회로는 그 아래 별도의 **래더** 그룹에 모입니다. 각 행을 클릭하면 그 탭으로 전환되고, 그룹 헤더 오른쪽의 **+** 버튼으로 새 탭을 추가하며, 행 위에 마우스를 올리면 오른쪽에 나타나는 ✕ 버튼으로 그 탭을 지웁니다.

> 코드·래더 탭은 에디터 영역 위쪽이 아니라 왼쪽 사이드바 **탐색기** 트리에서 관리합니다. 에디터 자체에는 별도의 탭 바가 없습니다 — 추가·전환·이름 변경·삭제· 순서 변경·우클릭 메뉴가 모두 위에서 설명한 탐색기 트리에서 이뤄지며, 이는 코드와 래더 양쪽에 똑같이 적용됩니다.

추가·이름 변경·삭제

동작	방법
추가	그룹 헤더의 **+** 버튼, 또는 행 우클릭 › **추가**. 이름을 입력하면 새 탭이 만들어지고 곧바로 활성화됩니다
이름 변경	행 더블클릭, 행을 선택한 뒤 F2 또는 Enter, 또는 행 우클릭 › **이름 변경**
삭제	행 hover 시 나타나는 ✕ 버튼, 행을 선택한 뒤 Delete, 또는 행 우클릭 › **삭제**. 셋 다 삭제 전에 확인 창을 거칩니다

순서 변경

같은 그룹 안에서 탭 순서를 바꾸려면 행을 **드래그&드롭**으로 끌어 옮기거나, 행을 선택한 뒤 Ctrl+ ↑ / Ctrl+ ↓로 한 칸씩 위아래로 움직입니다. 두 방법 모두 쓸 수 있습니다.

main 탭 잠금

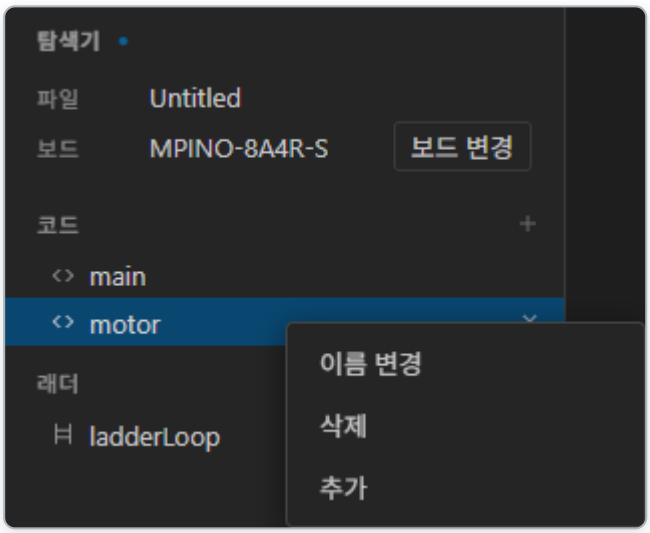
코드 그룹의 첫 탭인 `main`은 특별히 잠겨 있어 **이름 변경·삭제·이동이 되지 않습니다.** `main` 탭의 이름은 프로젝트 파일명과 항상 자동으로 같이 바뀌기 때문입니다(다른 이름으로 저장하면 파일명을 따라갑니다). 그래서 `main` 행을 우클릭하면 메뉴에 **추가**만 있습니다.

loop() 추가·삭제

`loop()` 안의 `ladderLoop()` 호출을 코드에 넣고 빼는 메뉴는 **래더** 그룹 행의 우클릭 메뉴에 있습니다 — **loop() 추가**와 **loop() 삭제**입니다. 코드 그룹 행에는 이 메뉴가 없습니다(**코드 에디터**(7-1)에서 설명한, `ladderLoop()` 호출 유지가 사용자 책임인 것과 이어집니다). 래더 탭의 이름을 바꾸면 `loop()` 안의 호출 이름도 자동으로 함께 갱신됩니다.

여러 .ino는 하나의 스케치

코드 탭이 여러 개여도 빌드할 때는 **하나의 스케치로 합쳐집니다.** 한 탭에서 정의한 함수를 다른 코드 탭에서도, 래더 박스함수에서도 부를 수 있습니다(래더에서 코드 함수를 부르는 방법은 **자유입력형 박스함수**(6-9), 여러 탭이 합쳐져 빌드되는 과정은 **빌드하기**(1-7) 참고). `main` 외에 새로 추가한 `.ino` 탭에는 맨 위에 `#include "ino_shared.h"` 한 줄이 미리 채워집니다 — 탭들이 서로의 함수를 공유하도록 잇는 줄입니다.



< 그림 7-2 탐색기 코드 그룹 — main과 추가한 motor 탭, 코드 행 우클릭 메뉴 >

7-3 자동완성

동작

코드를 입력하는 동안 완성 목록이 자동으로 떠오릅니다. 목록이 닫혀 있을 때는 Ctrl+Space로 직접 열 수 있습니다. 위/아래 방향키로 후보를 고른 뒤 Enter나 Tab을 누르면 선택한 후보가 삽입됩니다.

완성 후보는 어디서 오나

완성 목록에는 네 갈래의 후보가 섞여 나옵니다.

갈래	예
방금 문서에서 쓴 단어	조금 전에 정의한 함수·변수 이름
Arduino 내장 함수·상수	`pinMode` · `digitalWrite` · `analogWrite` · `Serial.*` · `HIGH` · `LOW` 등
열려 있는 `.ino` 탭들의 로컬 함수	코드 탭에서 직접 정의한 함수

갈래	예
인식된 라이브러리의 심볼	사용자 라이브러리·보드 코어가 제공하는 함수·상수

후보에 붙는 한 줄 설명

Arduino 내장 함수·상수(둘째 갈래) 후보에는 **한국어 한 줄 설명**이 함께 표시됩니다 — 예를 들어 `analogWrite`를 고르면 "PWM 출력(0~255).", `pinMode`에는 "핀 모드를 INPUT/OUTPUT/INPUT_PULLUP 으로 설정.", `INPUT_PULLUP`에는 "내부 풀업 입력 모드."가 붙습니다. 이 설명은 앱의 표시 언어를 따라가므로, 언어를 바꾸면 설명도 그 언어로 바뀝니다(언어 전환은 **환경설정**(3-1-1) 참고). 직접 정의한 로컬 함수나 라이브러리 심볼에는 한 줄 설명 대신 파라미터 목록 같은 시그니처 정보가 표시됩니다.

라이브러리 인식 지연

라이브러리를 새로 추가하거나 새로고침한 직후에는 그 라이브러리의 심볼이 완성 목록에 뜨기까지 잠깐 걸릴 수 있습니다(코드 분석을 준비하는 시간입니다). 잠시 후 다시 입력하면 반영됩니다. 라이브러리 폴더 지정과 새로고침은 **환경설정**(3-1-1)을 참고하세요.

관통 예제

코드 탭에 아래와 같이 함수를 하나 작성해 보겠습니다.

```
void setMotor(int speed) {
  analogWrite(9, speed);
}
```

이 줄을 쓰는 동안 `analogWrite`는 Arduino 내장 함수(둘째 갈래)로 완성됩니다. 그리고 이렇게 `setMotor`를 한 번 정의해 두면, 이후 다른 위치에서 `setM`까지만 쳐도 방금 정의한 `setMotor`가 로컬 함수(셋째 갈래)로 완성 목록에 떠오릅니다.



< 그림 7-3 자동완성 — setM 입력 시 방금 정의한 setMotor 함수가 후보로 표시 >

7-4 정의로 이동

F12로 정의 보기

코드 에디터에서 함수나 변수 이름 위에 커서를 두고 F12를 누르면 그 정의가 있는 곳으로 곧바로 이동합니다. 앞서 만든 `setMotor`처럼 직접 정의한 함수든, `analogWrite` 같은 Arduino 내장 함수든 똑같이 동작합니다.

정의를 여는 세 가지 길

F12 말고 마우스로도 같은 곳에 닿습니다.

조작	결과
F12	커서가 놓인 이름의 정의로 이동합니다
Ctrl+클릭	클릭한 이름의 정의로 이동합니다. 결과는 F12와 같습니다
Ctrl을 누른 채 이름 위에 마우스 올리기	이동하지 않고, 정의 부분을 **미리보기 팝업**으로 그 자리에서 보여 줍니다. Ctrl을 떼거나 마우스를 치우면 사라집니다

Ctrl을 누르고 있는 동안 이름에는 밑줄이 그어져, 지금 어느 이름이 이동 대상인지 눈으로 확인할 수 있습니다.

세 가지 이동

정의가 어디에 있느냐에 따라 이동 방식이 세 갈래로 갈립니다.

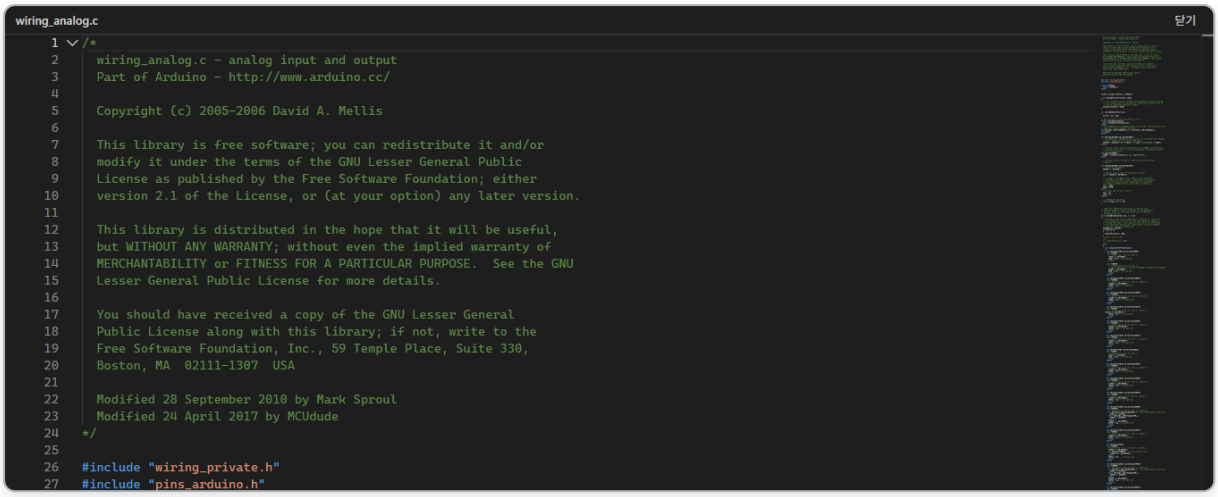
정의 위치	이동 방식
같은 `.ino` 탭 안	그 줄로 커서가 이동합니다
다른 `.ino` 탭	해당 탭으로 전환하면서 그 줄로 이동합니다
내장·라이브러리 정의	**읽기 전용 뷰어**가 열려 원본을 보여 줍니다. Esc 또는 닫기 버튼으로 닫으며, 이 뷰어에서는 내용을 고칠 수 없습니다

> 정의로 이동(F12) 외에 **선언으로 이동**·**타입 정의로 이동**·**구현으로 이동**· **모든 참조 찾기** 네 명령은 아직 동작하지 않습니다. 버그가 아니라 의도적으로 꺼 둔 것입니다 — 코드 분석 서버가 지금 편집 중인 코드 범위 밖의 위치를 정의로 돌려받으면 불안정해지는 경우가 있어, 그 불안정을 피하려고 이 네 명령만 비활성화해 두었습니다. 정의로 이동(F12)은 이 문제에서 벗어나 있어 정상 동작합니다.

> F12는 **커서가 어디에 있느냐**에 따라 하는 일이 달라집니다. 코드 에디터에 포커스가 있으면 F12는 방금 설명한 정의로 이동이고, 래더 그리드에 포커스가 있으면 F12는 하강 검출 접점을 놓는 도구입니다(**단축키**(2-3) 표 참조). 두 기능은 서로 충돌하는 것이 아니라, 지금 편집 중인 대상에 맞춰 골라 동작하는 것입니다.

관통 예제

앞서 작성한 코드에서 `analogWrite` 위에 커서를 두고 F12를 눌러 보면, Arduino 내장 정의가 읽기 전용 뷰어로 열립니다. 이 뷰어는 원본을 확인하는 용도이므로 내용은 바꿀 수 없고, Esc나 닫기 버튼으로 빠져나옵니다.



< 그림 7-4 정의로 이동 — analogWrite에서 F12를 누르면 내장 정의(Arduino 코어 소스)가 읽기 전용 뷰어로 열린다 >

7-5 찾기와 바꾸기

편집 대상에 따라 달라지는 Ctrl+F

Ctrl+F로 여는 찾기/바꾸기는 지금 편집하는 대상이 코드냐 래더냐에 따라 서로 다른 위젯이 뜹니다. 코드 탭에서 누르면 코드 텍스트를 찾는 표준 위젯이, 래더 탭에서 누르면 회로를 찾는 래더 전용 위젯이 열립니다(찾기 단축키 자체는 **단축키**(2-3) 참조).

코드 탭 — 텍스트 찾기

코드 탭에서는 익숙한 표준 찾기 위젯이 뜹니다. 찾을 말과 바꿀 말을 입력하고, 대소문자 구분·단어 단위·정규식 옵션을 켤 수 있으며, 이전/다음 화살표로 일치 지점 사이를 오갑니다.



< 그림 7-5 코드 탭의 찾기/바꾸기 — 텍스트 대상 표준 위젯 >

래더 탭 — 회로 찾기

래더 탭에서는 래더 전용 위젯이 뜹니다. 접점의 주소나 박스함수 안의 코드를 검색하며, **[검색]** 버튼으로 검색을 실행하고 이전(◀)·다음(▶) 화살표로 일치하는 셀 사이를 이동합니다. 위젯에는 현재 위치와 전체 일치 개수가 함께 표시되고, **바꾸기**·**모두 바꾸기**로 찾은 내용을 교체할 수 있습니다. 옵션은 다음과 같습니다.

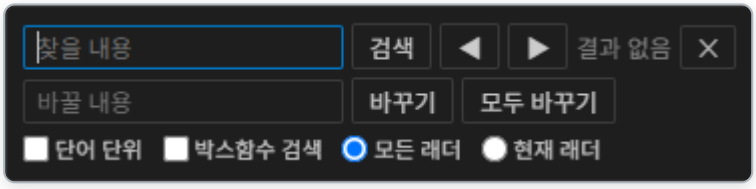
옵션	설명
단어 단위	검색어가 하나의 온전한 단어와 일치할 때만 찾습니다
박스함수 검색	박스함수 안의 코드를 대상으로 검색합니다. 이 옵션을 켜면 찾기 전용이 되어 **바꾸기**와 **단어 단위**는 함께 비활성화됩니다

옵션	설명
모든 래더 / 현재 래더	검색 범위를 전체 래더로 넓힐지, 지금 보고 있는 래더 하나로 한정할지 고릅니다

위젯을 열 때 커서가 놓여 있던 셀의 내용이 검색어 칸에 미리 채워집니다. 다만 코드 탭의 찾기와 달리 **글자를 입력하는 동안에는 아직 검색이 돌지 않습니다** — **Enter**를 누르거나 **[검색]** 버튼을 눌러야 그때 회로를 훑어 일치 개수가 채워집니다. 검색어를 바꾸지 않은 채 Enter를 다시 누르면 다음 일치로 넘어가고, **Shift+Enter**는 이전 일치로 돌아갑니다(▶·◀ 화살표와 같은 동작입니다). 일치하는 것이 하나도 없으면 개수 자리에 **결과 없음**이 표시됩니다.

옵션(단어 단위·박스함수 검색·검색 범위)을 바꾸거나 래더를 편집하면 그때는 [검색]을 다시 누르지 않아도 결과가 자동으로 다시 계산됩니다. **Esc**를 누르면 위젯이 닫힙니다.

래더 셀은 심볼 이름이 아니라 **주소로 저장**되므로, 검색어가 등록된 심볼 이름이면 그 심볼의 주소를 쓰는 셀도 함께 찾습니다. 바꾸기로 심볼 이름을 넣으면 셀 편집 창에서 확정할 때와 똑같이 그 심볼의 주소로 저장됩니다.



< 그림 7-6 래더 탭의 찾기/바꾸기 위젯 — 단어 단위·박스함수 검색·범위 옵션 >

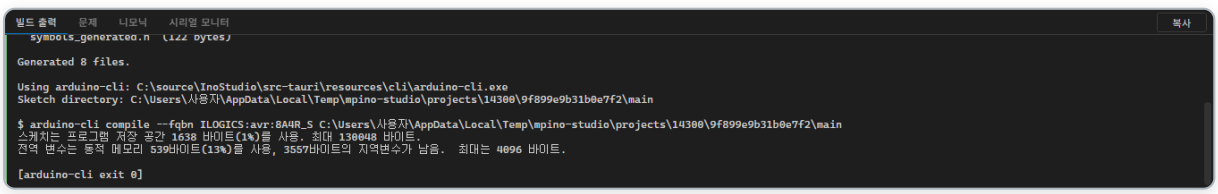
7-6 하단 패널 읽기

패널 열기와 닫기

작업 영역 아래쪽의 하단 패널은 Ctrl+J로 열고 닫습니다. 패널에는 **빌드 출력**, **문제**, **니모닉**, **시리얼 모니터** 네 개의 탭이 있으며, 이 장에서는 앞의 세 탭을 자세히 살펴봅니다.

빌드 출력

빌드 출력 탭에는 컴파일과 업로드가 진행되는 동안의 로그가 흐릅니다. 성공했는지 실패했는지, 실패했다면 어디서 멈췄는지가 이 로그에 나타납니다. 빌드·업로드를 실행하는 방법 자체는 **빌드하기**(1-7)와 **실행**(3-4)에서 다루므로, 여기서는 그 결과 로그를 읽는 탭이라는 점만 알아 두면 됩니다.



< 그림 7-7 빌드 출력 탭 — 컴파일 성공 로그 >

문제

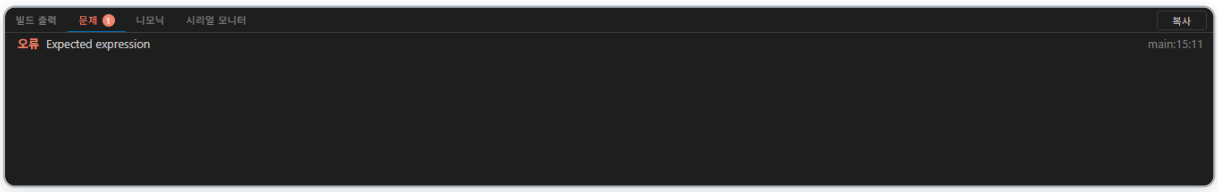
문제 탭에는 오류와 경고 같은 진단이 목록으로 모입니다. 진단이 있으면 탭 이름 옆에 개수 배지가 뜨는데, 오류는 붉은색, 경고는 노란색으로 구분됩니다. 목록의 항목을 클릭하면 그 진단이 가리키는 위치로 곧바로 점프합니다 — 코드 진단이면 해당 코드 탭의 그 줄로, 래더 진단이면 해당 래더의 그 셀로 이동합니다.

이 목록과 탭의 개수 배지, 그리고 진단이 있을 때 패널이 자동으로 열리는 동작은 **빌드·업로드· 프로그램 업로더 생성이 끝날 때만** 갱신됩니다. 편집하는 동안에는 직전 빌드의 목록이 그대로 남아 있고, 다른 프로젝트를 열면 목록이 비워집니다. 아직 빌드하지 않았다면 "빌드하면 여기에 문제가 표시됩니다."라는 안내가 보입니다. 반면 래더 셀에 잘못 입력해 글자가 적색이 되는 표시는 편집하는 즉시 바뀌며, 박스함수의 인자 칸에 잘못된 메모리 이름을 적었을 때도 그 글자가 적색이 됩니다. 코드 에디터의 오류 밑줄도 입력하는 대로 갱신됩니다.

빌드와 업로드는 성공하지만 보드에서 **아무 일도 일어나지 않을** 입력은 빌드를 막지 않고 경고로만 알려 줍니다. 경고가 떠도 빌드는 되니, 의도한 것이 아니면 문구대로 고치세요.

- **꺼진 핀·채널을 가리킬 때** — 박스함수 인자나 `WP0`처럼 여러 핀을 덮는 주소가 **사용 안 함**인 핀·모듈 채널을 가리킵니다. 핀 편집기(2-4)에서 그 핀의 **사용**을 켜세요. 문구 예: `C20은(는) 래더에서 사용안함으로 설정되어 있습니다`, `WP0 안의 P0~P15 중 래더에서 사용안함으로 설정되어 있습니다`
- **래더 함수를 부르지 않을 때** — 래더에 회로가 있는데 코드 탭 어디에서도 그 래더 함수를 부르지 않아 래더가 실행되지 않습니다. `loop()` 안에 `ladderLoop();`를 넣으세요. 문구 예: `ladderLoop()가 code에 없습니다`

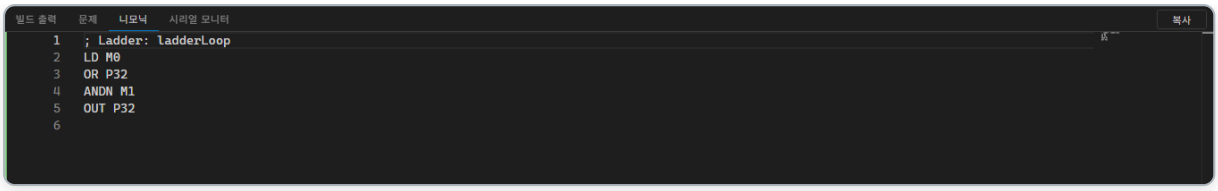
32767보다 큰 상수(DWORD 상수)가 16비트 워드에 쓰일 때도 빌드는 그대로 하고 경고로만 알려 줍니다 — 하위 16비트만 들어가거나, 비교·나눗셈 결과가 늘 같을 때입니다(**표현 범위**(2-1-1)). 이 경고는 **빌드 출력** 탭에도 `warning: ladderLoop row=1,col=0: …`처럼 래더 이름과 셀 위치를 붙여 함께 적힙니다. 문구 예: `상수 65535은(는) 16비트 범위를 넘는 더블워드 상수라 워드 C20에는 하위 16비트만 들어갑니다(C20 = -1).`



< 그림 7-8 문제 탭 — 진단 목록과 탭의 개수 배지 >

니모닉

니모닉 탭은 래더가 변환된 IL(중간 언어)을 읽기 전용으로 보여 줍니다. 래더 회로가 실제로 어떤 명령 순서로 바뀌는지 확인하는 보조 창으로, 래더가 어떻게 아두이노 코드가 되는지 이해하는 데 도움이 됩니다(**래더는 어떻게 아두이노 C 코드가 되나**(1-6) 참고). IL은 빌드가 **성공**했을 때만 채워집니다 — 아직 빌드하지 않았거나 직전 빌드가 실패했을 때는 안내 문구가 대신 표시되며, 실패 시점에는 이전에 남아 있던 IL도 함께 지워집니다.



< 그림 7-9 니모닉 탭 — 래더가 변환된 IL을 읽기 전용으로 표시 >

시리얼 모니터

네 번째 **시리얼 모니터** 탭은 보드가 보내오는 값을 읽는 창입니다. 보드에 연결해 수신 값을 확인하는 방법은 첫 실행 튜토리얼의 **동작 확인: 시리얼 모니터와 실시간 모니터링**(1-9)에서 다룹니다. 텍스트나 hex 값을 보드로 보내는 송신 등 더 자세한 사용법은 이후 장에서 이어집니다.

8 시리얼 모니터, 래더 모니터링

MPINO Studio 2는 보드와 두 가지 방식으로 대화한다. **시리얼 모니터**는 아두이노 `Serial` 코드가 주고받는 텍스트를 읽고 보내고, **래더 모니터링**은 보드에서 돌아가는 래더 프로그램의 내부 상태 — 접점 ON/OFF, 메모리 값, 전원이 흐르는 도통 흐름 — 을 래더 화면 위에 실시간으로 겹쳐 보여 준다. 이 장은 시리얼 모니터의 연결·수신·송신과 래더 모니터링의 계측·화면 읽기·표시 형식을 차례로 다루고, 두 기능이 같은 시리얼 포트를 나눠 쓰기 때문에 한 번에 하나만 켤 수 있다는 점을 마지막에 정리한다.

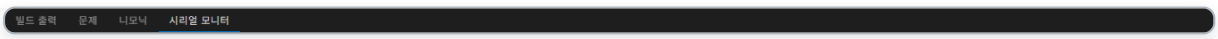
8-1 시리얼 모니터 개요

시리얼 모니터란

시리얼 모니터는 아두이노 보드의 `Serial`(USART) 포트로 PC와 텍스트를 주고받는 창입니다. 보드가 `Serial.print`로 보낸 값을 읽고(수신), 키보드로 입력한 텍스트를 보드로 보냅니다(송신). 펌웨어 디버깅이나 간단한 명령 전송에 씁니다.

패널 위치

시리얼 모니터는 화면 하단 패널의 탭 4종 — **빌드 출력 / 문제 / 니모닉 / 시리얼 모니터** 중 맨 오른쪽에 있습니다. 하단 패널 열기/닫기와 빌드 출력· 니모닉·문제 탭의 자세한 내용은 **하단 패널 읽기** (7-6) 참조. 좌측 활동바의 시리얼 아이콘을 눌러도 이 탭이 열립니다.



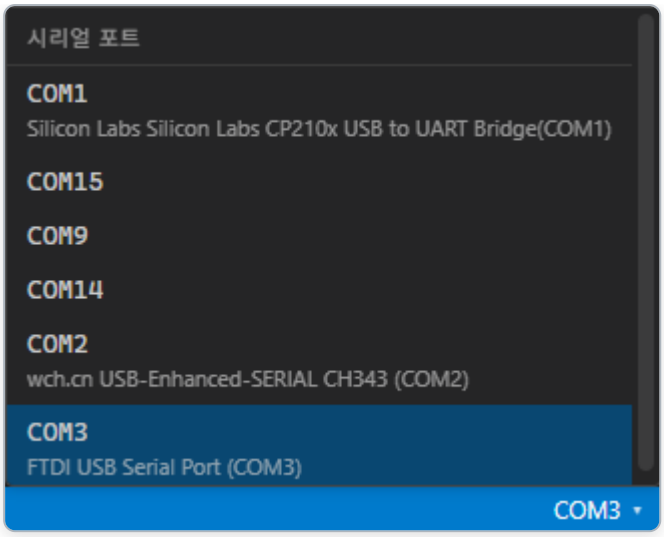
< 그림 8-1 하단 패널의 시리얼 모니터 탭 — 빌드 출력·문제·니모닉과 나란히 >

> **빌드나 업로드가 시작되면 시리얼 모니터를 보고 있어도 화면이 자동으로 빌드 출력 탭으로 전환됩니다.** 업로드가 끝난 뒤 시리얼 모니터 탭을 다시 누르면 됩니다.

8-2 연결하기

포트 선택은 상태바에서

연결할 COM 포트는 화면 하단 상태바 오른쪽의 포트 드롭다운(▾)에서 고릅니다 — 모니터 패널 안에는 포트 선택 기능이 없습니다. 드롭다운을 열 때마다 목록이 자동으로 새로고침되며, 각 포트는 이름과 USB 설명이 함께 표시됩니다. 포트가 없으면 "사용 가능한 포트가 없습니다"가 뜹니다. 이 포트 선택은 업로드와 같은 값을 공유합니다 — 자세한 내용은 **포트 선택과 업로드**(1-8) 참조.



< 그림 8-2 상태바 오른쪽 포트 드롭다운 — 연결할 COM 포트를 여기서 고른다 >

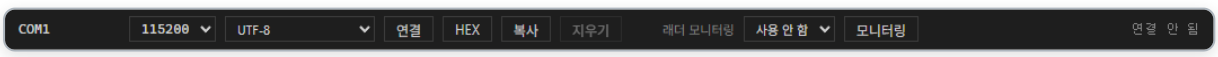
baud(통신 속도)

baud는 모니터 헤더의 baud 셀렉터에서 고릅니다(기본 115200, 16종 중 하나를 선택). baud는 **펌웨어의 `Serial.begin()` 속도와 반드시 같아야** 합니다 — 다르면 경고 없이 글자가 깨져 보일 뿐입니다.

연결/끊기

헤더의 연결 버튼을 누르면 연결됩니다. 버튼 라벨은 상태에 따라 **연결** → **끊기**(연결 중) → **중단**(재연결 시도 중)으로 바뀝니다. 상태 표시는 **연결 안 됨** / **연결됨** / **재연결 중** / **오류** 네 가지로 구분됩니다. 포트를 고르지 않고 연결 버튼을 누르면 "포트를 먼저 선택하세요 (상태바 우측 ▾ 클릭). 포트 없이 모니터를 열 수 없습니다."라는 안내가 뜹니다.

헤더의 컨트롤은 왼쪽부터 **포트 표시 → baud → 인코딩 → 연결 버튼 → HEX/ASCII 토글 → 복사 → 지우기** 순으로 놓입니다. 그 오른쪽에는 한 칸 띄어 **래더 모니터링**이라는 라벨이 붙은 묶음(**채널 선택 + [모니터링] 버튼**)이 따로 서고, 맨 오른쪽이 연결 **상태** 표시입니다. 이 묶음만 띄어 놓은 것은 앞의 것들이 시리얼 모니터 자신의 컨트롤인 반면 뒤의 둘은 래더 모니터링에 속하기 때문입니다(**래더 모니터링 시작하기**(8-5) 참고).



< 그림 8-3 시리얼 모니터 헤더 — 왼쪽부터 포트, baud, 인코딩, 연결 버튼, HEX 토글, 복사, 지우기, 그리고 한 칸 띄어 래더 모니터링 묶음(채널 선택 + 모니터링 버튼), 맨 오른쪽 상태 >

> 연결 중에는 baud를 바꿔도 즉시 반영되지 않습니다 — 한 번 끊고 다시 연결해야 새 속도가 적용됩니다. 보드를 뽑았다 꽂으면 자동으로 재연결을 시도하고(최대 5회), 성공하면 "-- 자동 재연결됨 --" 줄이 찍힙니다.

> **연결한 직후 0.6초쯤은 화면이 비어 있다가 그동안의 출력이 한꺼번에 나타납니다.** 포트를 연 프로그램은 처음 0.5초 동안 들어오는 바이트를 보면서 이것이 래더 모니터링 데이터인지 그냥 텍스트인지를 가릅니다. 모니터링 데이터가 아니라고 판정되면 그때 **보류해 둔 내용을 순서 그대로 되돌려** 화면에 풀어 놓습니다. 한 글자도 버리지 않으므로 `setup()`에서 한 줄만 찍고 조용해지는 스케치의 첫 줄도 그대로 보입니다. 잠깐 멈춘 것처럼 보이는 것은 정상입니다.

8-3 데이터 수신

수신 화면

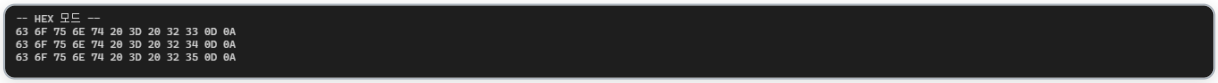
보드가 `Serial.print` / `Serial.println`으로 보낸 텍스트는 모니터 본문에 실시간으로 쌓이고, 새 줄이 도착할 때마다 **자동으로 맨 아래로 스크롤됩니다**. 관통 예제인 카운터 스케치(전체 코드는 데이터 송신에서 다룹니다)를 올리면 `count = 1`, `count = 2` … 가 1초 간격으로 흐르는 것을 볼 수 있습니다.



< 그림 8-4 데이터 수신 — 카운터 스케치가 보낸 count 값이 1초 간격으로 흐른다(상태 연결됨) >

ASCII / HEX 뷰

헤더의 뷰 토글 버튼으로 표시 방식을 바꿉니다. 버튼에는 지금 보고 있는 모드가 아니라 **바뀔 대상 모드**가 적혀 있습니다 — ASCII를 보는 중이면 "HEX", HEX를 보는 중이면 "ASCII"가 표시됩니다. HEX 뷰는 수신 바이트를 `AB CD 01 0A`처럼 2자리 대문자로, 바이트마다 공백으로 구분해 보여줍니다. 모드를 바꾸면 화면이 비워지고 `-- HEX 모드 --` 또는 `-- ASCII 모드 --` 줄이 찍힙니다(이전 로그는 사라집니다).



< 그림 8-5 HEX 뷰 — 같은 수신 데이터를 바이트 단위로 표시(-- HEX 모드 -- 줄) >

수신 인코딩

헤더의 인코딩 셀렉터로 수신 텍스트를 해석할 문자 인코딩을 고릅니다. 지원하는 인코딩은 **UTF-8 / EUC-KR (한국어) / Shift-JIS (일본어) / Latin-1 (ISO-8859-1) / Windows-1252** 다섯 가지이며, 기본값은 UTF-8입니다. 한글 펌웨어(EUC-KR)가 보낸 한글이 깨져 보이면 인코딩을 EUC-KR로 바꾸면 됩니다. 인코딩을 바꾸면 화면이 비워지고 `-- 인코딩: … --` 줄이 찍힙니다(단, HEX 뷰에서는 화면 변화 없이 설정값만 바뀝니다).

> 인코딩 선택은 **수신 해석에만** 적용됩니다(HEX 뷰에서는 의미가 없습니다). 송신은 항상 UTF-8입니다 — 송신 인코딩은 **데이터 송신**(8-4)에서 다룹니다.

지나간 내용 거슬러 읽기

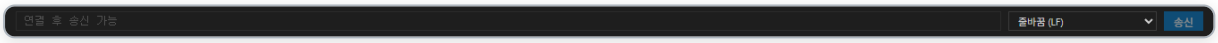
값이 빠르게 흐르는 중에 앞부분을 읽고 싶으면 본문을 **위로 스크롤하면 됩니다** — 맨 아래 고정이 그 순간 풀려, 새 줄이 계속 도착해도 화면이 아래로 끌려가지 않습니다. 다시 맨 아래까지 스크롤해 내리면 고정이 되살아나 자동 스크롤이 이어집니다. **지우기**를 누르거나 ASCII/HEX 뷰를 전환할 때도 화면이 다시 맨 아래로 고정됩니다.

> 시각(타임스탬프)과 일시정지 기능은 없습니다. 화면을 비우려면 **지우기**, 내용을 복사하려면 복사 버튼을 씁니다. 화면은 약 20만 자를 넘으면 앞부분부터 자동으로 지워집니다.

8-4 데이터 송신

입력과 전송

모니터 패널 아래쪽 입력창에 텍스트를 입력한 뒤 **Enter** 키 또는 **송신** 버튼으로 보냅니다. 연결되어 있지 않으면 입력창이 비활성화되고 "연결 후 송신 가능"이 표시됩니다. 보낸 텍스트는 화면에 `>` 접두어를 달고 그대로 남아(echo), 보드에서 오는 수신 내용과 시간 순서대로 섞여 보입니다.



< 그림 8-6 입력창 상태 — 미연결이면 비활성(연결 후 송신 가능), 연결되면 입력 가능 >

줄 끝 문자(line ending)

입력창 옆 셀렉터로 전송 시 뒤에 붙일 줄바꿈을 고릅니다 — **None (LE 없음) / 줄바꿈 (LF) / 캐리지 리턴 (CR) / 줄바꿈 + 캐리지 리턴 (CRLF)** 중에서 고르며, 기본값은 **줄바꿈 (LF)** 방식입니다. 펌웨어가 `Serial.read()`로 줄 단위 입력을 처리한다면 이 설정이 펌웨어가 기대하는 개행과 일치해야 합니다.

HEX 송신

HEX 뷰에서는 입력창에 `AB CD 01`처럼 공백으로 구분하거나 `ABCD01`처럼 붙여서 16진수를 직접 입력해 raw 바이트를 그대로 보냅니다. 형식이 틀리면(16진수가 아닌 문자가 섞였거나 자릿수가 홀수면) "송신 실패: 잘못된 hex 입력 (예: 'AB CD 01' 또는 'ABCD01')" 안내가 뜹니다.

> **송신은 항상 UTF-8로 인코딩됩니다**(수신 인코딩 설정과 무관합니다). 그래서 EUC-KR만 이해하는 펌웨어에 **한글을 보내려면** HEX 뷰에서 그 바이트를 직접 입력해야 합니다. 또한 보낸 내용을 다시 불러오는 **입력 이력(↑/↓)은 없습니다**.

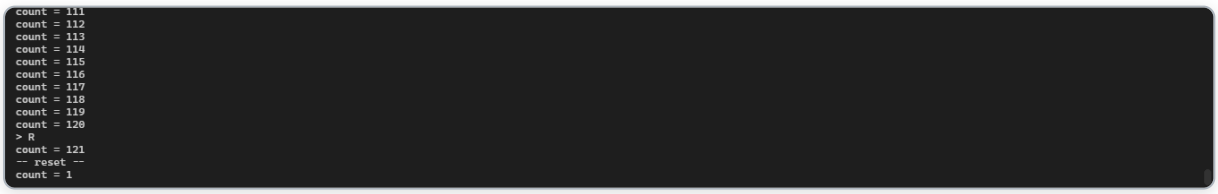
관통 예제(카운터 스케치)

아래 스케치를 코드 탭에 작성하고 빌드·업로드합니다 — 코드 작성 요령은 **코드 에디터**(7-1)를 참고하세요. 시리얼 모니터를 baud 9600으로 연결하면 1초마다 `count` 값이 수신되고, 입력창에 `R`을 보내면(송신) `-- reset --`과 함께 `count`가 0으로 돌아옵니다.

```
unsigned long lastTick = 0;
int count = 0;

void setup() {
  Serial.begin(9600);
  Serial.println("Counter ready. Send R to reset.");
}

void loop() {
  if (millis() - lastTick >= 1000) {   // 1초마다 카운터 송신
    lastTick = millis();
    count++;
    Serial.print("count = ");
    Serial.println(count);
  }
  if (Serial.available() > 0) {        // 사용자 입력 수신
    char c = Serial.read();
    if (c == 'R' || c == 'r') {
      count = 0;
      Serial.println("-- reset --");
    }
  }
}
```



< 그림 8-7 데이터 송신 — R을 보내면 echo(> R)와 보드 응답(-- reset --)이 수신과 섞여 보인다 >

> 이 예제는 **래더 모니터링이 꺼진 상태**에서 동작합니다. 래더 모니터링을 켜면 프로그램이 시리얼 포트를 가져가 사용자 `Serial` 코드가 밀립니다 — 자세한 이유는 **시리얼 모니터와 래더 모니터링**(8-8)에서 다룹니다.

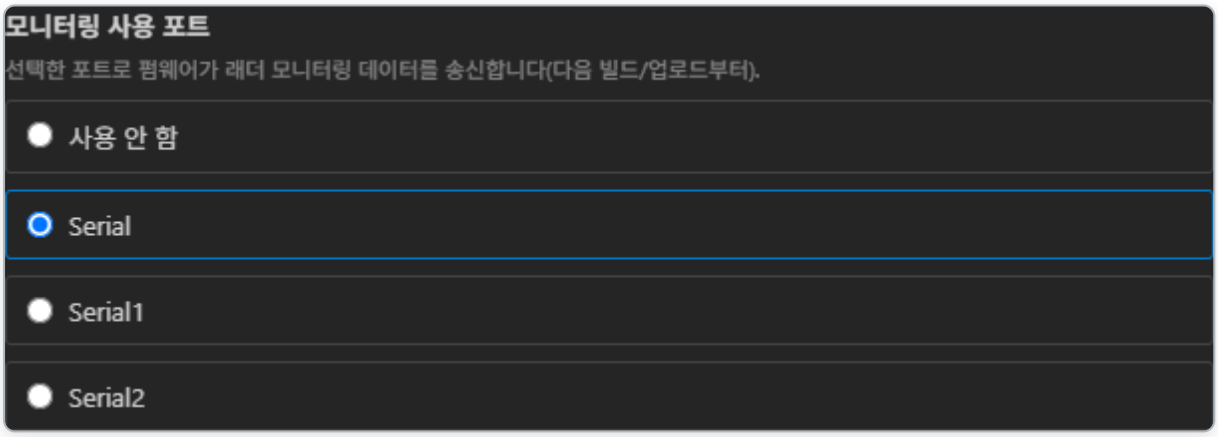
8-5 래더 모니터링 시작하기

래더 모니터링이란

래더 모니터링은 보드에서 실행 중인 래더 프로그램의 내부 상태 — 접점의 ON/OFF, 메모리 값, 전원이 흐르는 도통 흐름 — 을 래더 화면 위에 실시간으로 겹쳐 보여주는 기능입니다. 시리얼 모니터가 텍스트를 주고받는 것과 달리, 래더 모니터링은 PLC 내부 논리 상태를 직접 들여다봅니다.

세 단계 준비

래더 모니터링을 보려면 순서대로 세 단계를 거쳐야 합니다. ① **환경설정 → 래더 설정 → 모니터링 사용 포트**에서 통신 포트를 지정합니다(전체 항목 설명은 **환경설정**(3-1-1) 참고) — 이 선택이 펌웨어에 모니터링 데이터를 내보내는 코드를 심습니다(계측). ② 그 상태로 **다시 빌드·업로드**합니다 — 설정 화면의 힌트가 이 순서를 그대로 알려줍니다: "선택한 포트로 펌웨어가 래더 모니터링 데이터를 송신합니다(다음 빌드/업로드부터)." ③ **모니터링** 버튼(또는 Ctrl+M, 래더 화면 우클릭 "모니터링 ON")으로 통신을 켭니다 — 켜는 자리는 이 밖에도 있으며 아래 **켜는 다섯 가지 방법**에 모아 두었습니다.



< 그림 8-8 환경설정 → 래더 설정 → 모니터링 사용 포트 — 여기서 지정해야 펌웨어가 계측된다 >

포트를 고르는 두 자리

①의 포트는 환경설정 말고 **시리얼 모니터 헤더에서도 곧바로 고를 수 있습니다.** 헤더 오른쪽 **래더 모니터링** 묶음의 왼쪽 셀렉터가 그것으로, 마우스를 올리면 "펌웨어가 모니터링 데이터를 내보낼 채널. 바꾼 뒤 업로드해야 보드에 반영됩니다."라는 설명이 뜹니다. 두 자리는 **완전히 같은 값**을 가리키므로 어느 쪽에서 바꾸든 다른 쪽에도 즉시 반영되며, 뒤따르는 확인 절차도 똑같습니다. 매번 환경설정을 열지 않아도 되는 지름길이라고 보면 됩니다.

진입점	자리
환경설정 라디오	**환경설정 → 래더 설정 → 모니터링 사용 포트**
시리얼 모니터 헤더 셀렉터	헤더 오른쪽 **래더 모니터링** 묶음의 왼쪽

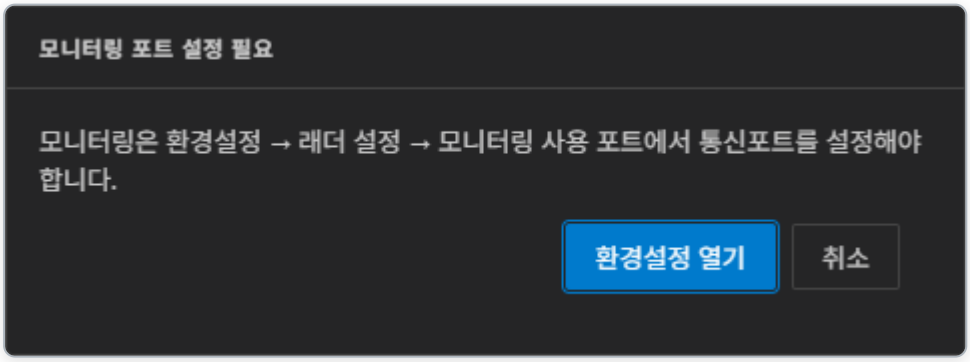
고를 수 있는 값은 **사용 안 함 / Serial / Serial1 / Serial2 / Serial3**이며, 실제로 나열되는 것은 **지금 보드가 가진 통신 채널만**입니다 — 예를 들어 UART가 둘뿐인 보드에서는 Serial2가 목록에 나오지 않습니다. **사용 안 함**을 고르면 계측 자체가 빠지므로 래더 모니터링을 쓸 수 없습니다. **Serial3**은 ATmega2560 보드에만 있는 TTL 레벨 포트(D14 TXD3 / D15 RXD3)라 PC와 연결하려면 **USB-TTL 변환기**가 필요합니다.

헤더의 셀렉터는 **모니터링 통신이 켜져 있는 동안과 업로드가 진행되는 동안에는 잠깁니다.** 통신을 끄거나 업로드가 끝나면 다시 고를 수 있습니다.

포트를 바꿀 때 확인 창이 뜰 수 있습니다 — [취소]를 누르면 바꾸지 않습니다.

- 아두이노 코드가 이미 그 포트를 쓰고 있으면(예: 코드에 `Serial1.begin()`이 있는데 모니터링 포트로 Serial1을 고른 경우) **"Serial1 중복 사용"** 창이 떠 "아두이노 코드에서 Serial1을 사용하고 있습니다. 중복으로 사용할 경우, 래더 모니터링이 정상 동작하지 않을 수 있습니다."라고 알려 줍니다. **[확인]을** 눌러야 포트가 바뀌고, **[취소]를** 누르면 포트도 화면 표시도 바뀌기 전 그대로 남습니다.
- 포트를 바꾼 뒤 **업로드하지 않은 채** 모니터링을 켜려고 하면 **"모니터링 채널 변경"** 창이 떠 "모니터링 채널을 변경하고 프로그램 업로드를 해야만 채널 변경이 반영됩니다. 업로드를 하시겠습니까?"라고 묻습니다. **[업로드]를** 고르면 곧바로 업로드하고, 성공하면 이어서 모니터링이 켜집니다. **[취소]를** 고르면 업로드도 하지 않고 모니터링도 켜지지 않습니다.

"모니터링" 버튼은 통신만 켭니다. 펌웨어가 모니터링 데이터를 실제로 내보내게 만드는 것은 ①의 포트 설정과 ②의 재업로드입니다. 포트를 설정하지 않은 채 모니터링 버튼(또는 Ctrl+M, 우클릭 메뉴)을 켜려고 하면 **"모니터링 포트 설정 필요"** 안내가 뜨며 설정 방법을 알려 줍니다: "모니터링은 환경설정 → 래더 설정 → 모니터링 사용 포트에서 통신포트를 설정해야 합니다." 예전에 저장한 프로젝트는 계측 코드가 없을 수 있으니, 포트를 설정한 뒤 다시 빌드·업로드해야 모니터링 데이터가 나옵니다.



< 그림 8-9 포트를 설정하지 않고 모니터링을 켜면 나오는 안내 >

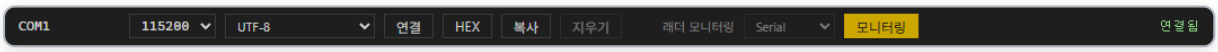
켜는 다섯 가지 방법

포트 설정과 재업로드를 마쳤다면, 통신은 다음 어느 자리에서 켜도 같습니다. 다섯 자리 모두 **같은 하나의 스위치**를 누르는 것이라, 한 곳에서 켜면 나머지 네 곳의 표시도 함께 켜진 상태로 바뀝니다.

자리	조작
시리얼 모니터 헤더	**래더 모니터링** 묶음 오른쪽의 **[모니터링]** 버튼
래더 화면 우클릭 메뉴	첫 항목 — 꺼져 있으면 **모니터링 ON**, 켜져 있으면 **모니터링 OFF**
단축키	**Ctrl+M**
보기 메뉴	**모니터링 토글**(**보기**(3-3) 참고)
메모리 쓰기 창	**메모리 쓰기 목록**(3-4-2) 창과 **아두이노에 값 쓰기**(4-4) 팝업의 헤더에 있는 **[모니터링]** 버튼

버튼이 켜지면 노란색으로 강조되고, 마우스를 올리면 지금 통신이 켜져 있는지 꺼져 있는지가 툴팁으로 표시됩니다.

> 메모리 쓰기 창의 [모니터링] 버튼이 같이 있는 이유는, 메모리에 값을 쓰고 **그 결과를 현재값 열에서 바로 확인**하려면 모니터링 통신이 켜져 있어야 하기 때문입니다. 창을 닫지 않고 그 자리에서 켤 수 있습니다.

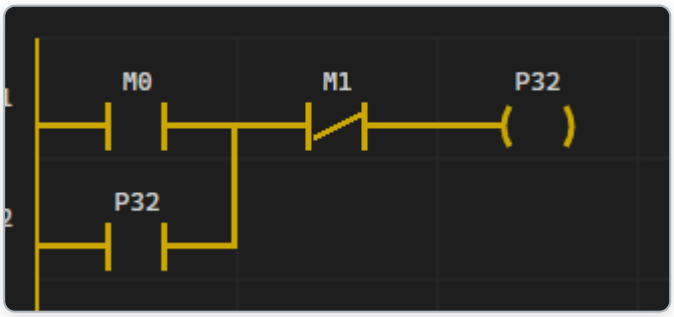


< 그림 8-10 모니터링 버튼이 켜진 상태(노란색 강조) >

8-6 도통과 라이브 값 읽기

도통(전원 흐름) 강조

모니터링을 켜면 전원이 흐르는 접점·코일·배선이 **노란색으로 굵게** 표시되고, 왼쪽 전원 레일도 함께 활성화됩니다. NC 접점(B접점)은 비트가 OFF일 때 도통합니다 — 자세한 동작은 **NC 접점**(5-2-2) 참조.

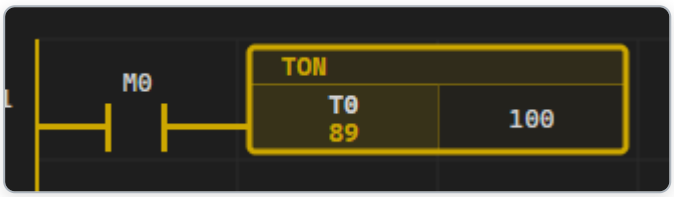


< 그림 8-11 도통 강조 — 전원이 흐르는 접점과 배선이 노란색으로 굵게 표시된다 >

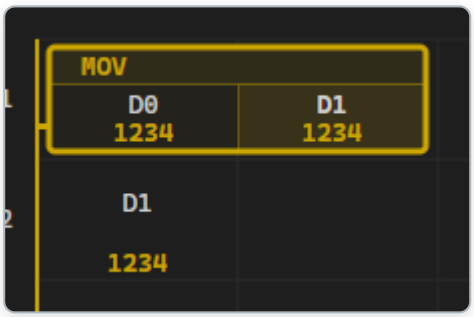
셀별 라이브 값

라이브 값이 표시되는 자리와 형태는 셀 종류에 따라 다릅니다.

- 비트 접점(P/M): 값 글자 없이 도통 색으로만 ON/OFF를 보여줍니다. 접점에 T·C 주소를 넣으면 현재값이 아니라 완료비트를 가리키므로, 이때도 숫자 없이 도통 색으로만 표시됩니다.
- 박스함수(Func): 주소가 든 칸마다 그 칸 아래 줄에 현재 값이 표시됩니다(예: `ADD D0 D1 D2` → `5` · `3` · `8`). 식·상수 칸에는 값이 없습니다.
- 빈 셀에 주소만 넣은 경우(watch): 비트 주소(P/M)는 셀 아래에 ON/OFF로, 숫자 주소(D/R 워드, 타이머·카운터 현재값)는 그 값이 셀 아래에 숫자로 표시됩니다.



< 그림 8-12 박스함수 라이브 값 — 주소가 든 칸마다 현재 값이 표시된다 >



< 그림 8-13 watch 셀 — 빈 셀에 넣은 주소의 현재 값이 셀 아래에 숫자로 표시된다 >

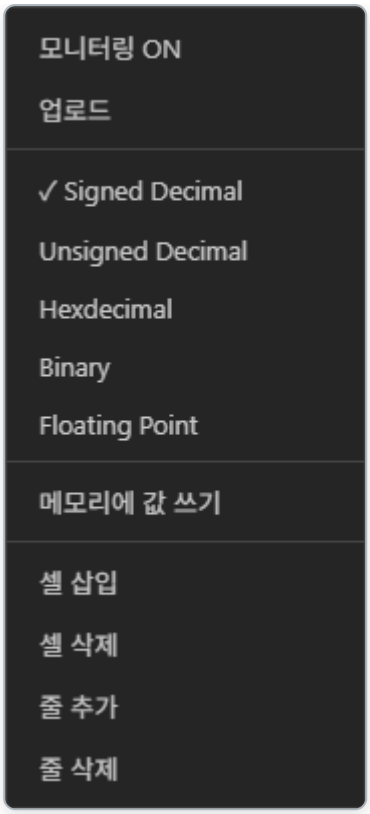
> 카운터·타이머를 접점 자리(완료 접점)에 놓으면 숫자 대신 완료 여부(도통)로만 표시됩니다. 예전에 저장·업로드한 보드는 이 완료 신호를 보내지 않아 완료 접점이 실제 상태와 다르게 표시될 수 있습니다 — 모니터링 설정 후 다시 빌드·업로드하면 정확해집니다.

> 보드가 멈추거나 통신이 끊기면 마지막으로 받은 값과 도통 색이 그대로 남습니다 — 거짓 실시간처럼 보이지 않도록 시간에 따라 바뀌는 표시만 멈춥니다.

8-7 표시 형식과 특수 표시

숫자 표시 형식 5종

라이브 숫자 값을 어떤 진법으로 볼지는 래더 셀 우클릭 메뉴에서 고릅니다. 메뉴에는 **Signed Decimal / Unsigned Decimal / Hexdecimal / Binary / Floating Point** 다섯 가지가 영어 라벨 그대로 나열되고, 현재 선택한 항목에는 체크(✓)가 붙습니다. 이 설정은 셀마다 따로 적용되는 것이 아니라 **모든 프로젝트에 공통으로 적용되는 전역 설정**이며, 기본값은 Signed Decimal입니다.

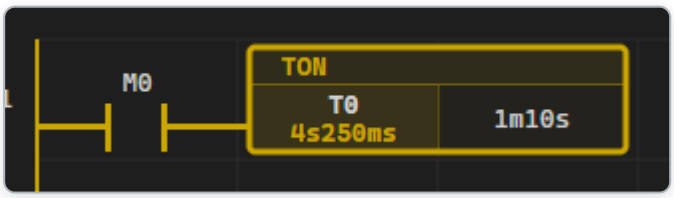


< 그림 8-14 표시 형식 5종 — 래더 셀 우클릭 메뉴에서 진법을 고른다 >

타이머 시간 단위

타이머 박스함수의 프리셋을 시간 문법으로 쓰면(예: `TON T0 1m10s`), 모니터링 중 현재값도 시간 단위로 표시됩니다 — 큰 단위부터 필요한 만큼만 이어 붙여 `1m10s`, `43s210ms`, `500ms`처럼 나타납니다. 프리셋을 순수 숫자(카운트)나 `D10` 같은 메모리 주소로 쓰면 현재값도 숫자 그대로 표시됩니다. 한 카운트가 몇 ms인지는 프로젝트의 타이머 단위 설정을 따릅니다(6-1-6).

이 시간 단위 표시는 표시 형식이 기본값인 Signed Decimal일 때만 적용됩니다. 표시 형식을 다른 진법으로 바꾸면 시간 단위 대신 그 진법으로 나타낸 숫자가 보이고, 카운터는 애초에 시간 단위 표시를 쓰지 않습니다.

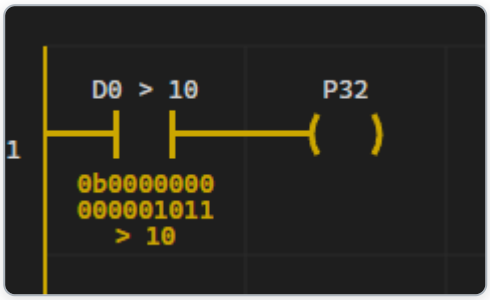


< 그림 8-15 타이머 시간 단위 — 시간 문법 프리셋이면 현재값도 1m10s처럼 표시된다 >

비교 접점 값 표시

`D0 > D1`처럼 비교식을 넣은 접점(자세한 내용은 **비교 접점**(5-6) 참조)은 모니터링 중에 양변을 실제 값으로 바꾼 `5 > 3` 형태를 접점 아래에 함께 표시합니다. 이 값이 표시되는 동안 그 셀의 세로 크기가 잠시 커지고, 모니터링을 끄면 원래 크기로 돌아갑니다.

> 표시 형식이 Binary일 때만 값이 좌변과 연산자+우변으로 나뉘어 여러 줄로 표시됩니다. 그 밖의 형식(Signed Decimal·Unsigned Decimal·Hexdecimal· Floating Point)은 한 줄로 표시됩니다. 두 경우 모두 셀이 커지지만, 여러 줄로 나뉘는 Binary 쪽이 더 많이 커집니다.



< 그림 8-16 비교 접점 값 — 모니터링 중 실제 값이 접점 아래에 표시되며, Binary 형식에서는 여러 줄로 나뉘어 셀이 더 커진다 >

8-8 시리얼 모니터와 래더 모니터링

하나의 포트, 두 용도

시리얼 모니터의 **연결**과 래더 **모니터링**은 같은 시리얼 포트를 나눠 씁니다. 그래서 **동시에 켤 수 없습니다** — 하나를 켜면 다른 하나는 자동으로 꺼집니다. 래더 모니터링을 켜면 그 세션이 포트를 115200으로 가져가고, 시리얼 송신 입력창은 비활성화됩니다(모니터링 통신에는 래더 상태 데이터만 흘러야 하기 때문입니다).

언제 무엇을 쓰나

하고 싶은 것	쓰는 기능
아두이노 코드(`Serial.print`)의 디버그 출력을 보거나 텍스트 명령을 보낼 때	시리얼 모니터(8-1~8-4)
래더 프로그램의 접점·메모리·도통 상태를 실시간으로 볼 때	래더 모니터링(8-5~8-7)

관통 예제와의 관계

8-4의 카운터 스케치처럼 사용자가 직접 `Serial`을 쓰는 코드는 **래더 모니터링을 끈 상태**에서만 시리얼 모니터로 관찰됩니다. 래더 모니터링을 켜면 프로그램이 시리얼 포트를 계측용으로 가져가기 때문입니다.

> **시리얼 모니터 = 아두이노 코드와 텍스트로 통신하는 창, 래더 모니터링 = PLC 내부 상태를 들여다보는 화면**입니다. 둘은 같은 포트를 나눠 쓰므로 한 번에 하나만 켤 수 있습니다.

9 활동바와 사이드 패널

MPINO Studio 2의 왼쪽 가장자리에는 **활동바**(Activity Bar)라는 세로 아이콘 열이 있습니다. 이 아이콘들을 누르면 프로젝트 파일, 보드 정보, 라이브러리 같은 정보는 화면 왼쪽 **사이드 패널**(Side Panel)로, 심볼처럼 별도 창이 필요한 정보는 모달로 열려 탐색하고 관리할 수 있습니다. 이 장은 활동바의 일곱 가지 아이콘과 그 각각이 여는 화면의 쓰임새를 설명합니다.

9-1 활동바 개요

활동바란

화면 왼쪽 끝에 세로로 늘어선 아이콘 열입니다. 아이콘을 누르면 그 아이콘이 맡은 화면이 열립니다 — 어떤 아이콘은 왼쪽에 사이드 패널을 펼치고, 어떤 아이콘은 별도의 모달(팝업) 창을 띄우며, 어떤 아이콘은 에디터 아래쪽 하단 패널로 이동합니다. 활동바 자체는 **항상 화면에 표시**되며 숨기는 기능은 없습니다.

7개 아이콘이 여는 화면

아이콘	여는 화면	이 매뉴얼에서
탐색기	사이드 패널	아래 **탐색기 패널**(9-2)
래더설정	사이드 패널	아래 **래더설정 패널**(9-3)
래더 핀맵 설정	모달	**핀 편집**(2-4)의 "보드 핀맵 편집"
라이브러리	사이드 패널	아래 **라이브러리 패널**(9-4)
심볼	모달(모달리스)	아래 **심볼**(9-5)
시리얼 모니터	하단 패널	**시리얼 모니터 개요**(8-1), **래더 모니터링 시작하기**(8-5)
설정	모달	**환경설정**(3-1-1)

아이콘 라벨은 순서대로 "탐색기" / "래더설정" / "래더 핀맵 설정" / "라이브러리" / "심볼" / "시리얼 모니터" / "설정" 입니다. 이 가운데 탐색기·래더설정·라이브러리 세 아이콘만 사이드 패널을 열고, 나머지 네 아이콘은 모달이나 하단 패널처럼 다른 화면으로 이동합니다. 현재 열려 있는 화면에 대응하는 아이콘은 배경이 강조되어 어떤 화면이 열려 있는지 한눈에 알 수 있습니다.



< 그림 9-1 화면 왼쪽 활동바 — 아이콘마다 사이드 패널·모달·하단 패널이 열린다 >

사이드바 열고 닫기

사이드 패널은 `Ctrl+B`로 열고 닫을 수 있습니다(보기 메뉴의 "사이드바 토글"과 같은 동작입니다 — **보기**(3-3) 참고). 사이드 패널을 여는 세 아이콘(탐색기·래더설정· 라이브러리) 가운데 하나를 누르면 다음과 같이 동작합니다.

- 사이드바가 닫혀 있었다면 → 그 아이콘의 패널이 열립니다.
- 사이드바가 열려 있고 다른 패널이 선택돼 있었다면 → 그 아이콘의 패널로 전환되며 사이드바는 열린 상태를 유지합니다.
- 사이드바가 열려 있고 이미 그 패널이 선택돼 있었다면 → 사이드바가 닫힙니다(같은 아이콘을 한 번 더 누른 것으로 취급).

> 사이드 패널의 폭은 **280px 고정**이며, 경계를 드래그해 넓히거나 좁힐 수 없습니다. 에디터 아래쪽 하단 패널(빌드 출력·시리얼 모니터 등)은 경계를 드래그하면 높이를 조절할 수 있지만, 이는 사이드 패널의 폭 조절과는 무관한 별개 기능입니다. 하단 패널 경계를 클릭하거나 Tab 키로 포커스를 옮기면 키보드로도 높이를 조절할 수 있습니다 — **↑ / ↓ 키**는 8px씩, **PageUp/PageDown 키**는 32px씩 높이를 늘리거나 줄이고, **Home 키**는 최대 높이로, **End 키**는 최소 높이로 바로 이동합니다. 마우스 드래그와 키보드 조작은 함께 사용할 수 있습니다.

9-2 탐색기 패널

프로젝트 이름표

탐색기 패널 위쪽에는 지금 열려 있는 프로젝트의 정보가 두 줄로 표시됩니다. **파일** 줄은 현재 프로젝트 `.mp2`의 파일 이름이고, **보드** 줄은 그 프로젝트가 대상으로 하는 보드 이름입니다. 지금 빌드 입력이 어떤 파일이고 어떤 보드를 겨냥하는지 여기서 바로 확인할 수 있습니다. 저장하지 않은 변경이 있으면 패널 헤더 오른쪽에 점(●)이 나타나 알려 줍니다.

보드 줄 오른쪽에는 **[보드 변경]** 버튼이 있습니다. 새 프로젝트를 만들지 않고도 지금 프로젝트가 대상으로 하는 보드만 다른 보드로 바꿀 수 있는 진입점으로, 누르면 현재 보드가 미리 선택된 채로 **보드 선택** 모달이 열립니다(모달 자체는 **새 프로젝트 만들기와 보드 선택**(1-4)에서 본 것과 같은 화면입니다). 다른 보드를 골라 확정하면 코드·래더는 그대로 두고 대상 보드 정보만 교체되고, 취소하면 보드는 바뀌지 않습니다. 같은 기능은 설정 메뉴의 **보드 변경**(3-5)으로도 열 수 있습니다.

옵션 모듈을 붙인 프로젝트라면 이름표가 세 줄이 됩니다. **보드** 줄 아래에 **옵션 모듈** 줄이 하나 더 붙어, 지금 프로젝트가 어떤 모듈을 몇 개 쓰는지를 `X2 · Y1 · K1`처럼 요약해 보여 줍니다. 글자는 모듈 종류(X = 아날로그 입력, Y = 아날로그 출력, F = PT100Ω 입력, K = 펄스 출력)이고 뒤의 숫자가 개수이며, **개수가 0인 종류는 아예 나열되지 않습니다.** 네 종류가 모두 0이거나 옵션 모듈을 지원하지 않는 보드에서는 이 줄 자체가 나타나지 않아, 앞서 말한 두 줄만 보입니다. 모듈 개수를 정하는 곳은 보드 선택 모달이며 **새 프로젝트 만들기와 보드 선택**(1-4)에서 다룹니다.

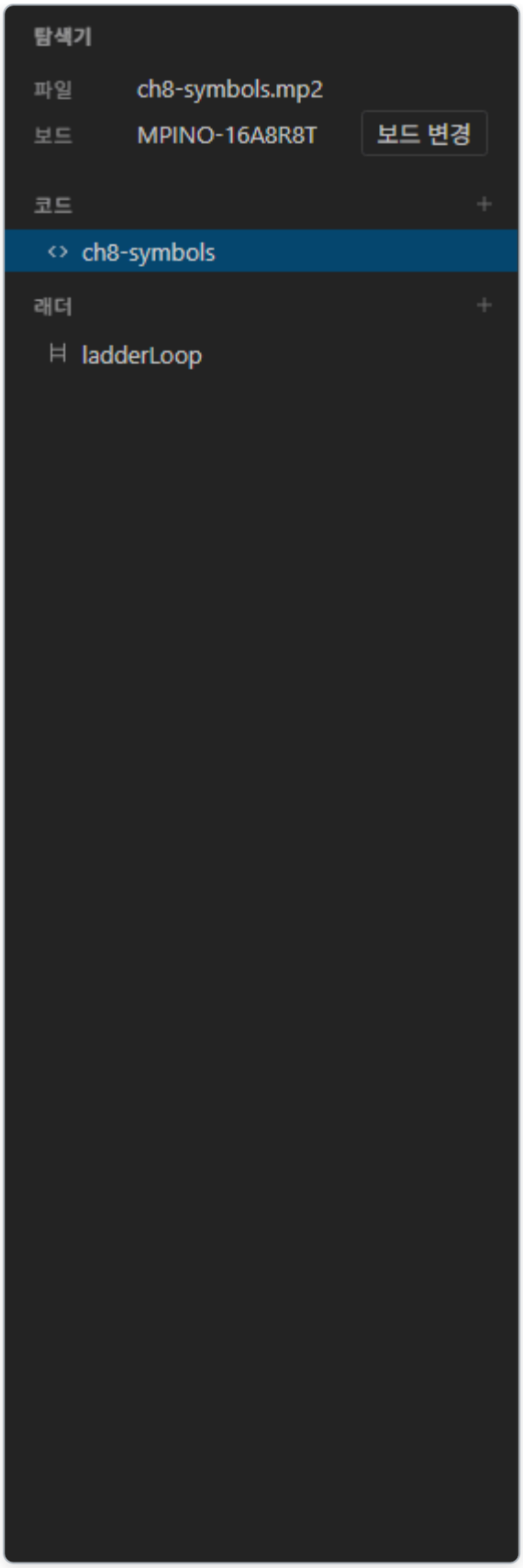
코드와 래더 트리

이름표 아래로는 `.ino` 파일이 모인 **코드** 그룹과 래더 회로가 모인 **래더** 그룹, 두 그룹으로 나뉜 트리가 이어집니다. 편집기 영역에는 별도의 탭 바가 없어 이 트리가 곧 탭 역할을 합니다 — 트리에서

항목을 클릭하면 그 파일이 편집기 영역에 열리고, 이미 열려 있으면 그 파일로 전환됩니다.

파일/래더 추가, 이름 변경(F2), 삭제, 드래그로 순서 바꾸기 같은 자세한 조작 방법은 **코드 탭 운용**(7-2)에서 다룹니다 — 탐색기 패널의 코드/래더 그룹이 곧 그 절에서 설명하는 트리이므로, 여기서는 되풀이하지 않습니다.

> 코드 파일과 래더 회로는 모두 같은 하나의 `.mp2` 프로젝트에 담기므로, 탐색기 패널은 그 프로젝트가 품은 편집 대상 전체를 **한 화면에서** 훑어볼 수 있는 자리입니다.



< 그림 9-2 탐색기 패널 — 코드/래더 파일 트리와 프로젝트·보드 이름(자세한 조작은 7장) >

9-3 래더설정 패널

무엇이 보이나

활동바의 "래더설정" 아이콘을 누르면 사이드 패널에 **현재 프로젝트가 쓰는 보드의 정보**가 뜹니다. 패널 맨 위에는 보드 이름이 한 줄로 표시되고, 그 아래로 두 섹션이 이어지며 각 섹션 제목을 누르면 접거나 펼 수 있습니다. 이 화면은 어디까지나 **읽기 전용**입니다 — 이 패널 자체에는 보드를 바꾸는 버튼이 없고, 지금 프로젝트가 쓰는 보드의 정보를 확인하는 용도입니다. 보드를 바꾸려면 **탐색기 패널**(9-2)의 **[보드 변경]** 버튼이나 설정 메뉴의 **보드 변경**(3-5)을 씁니다.

핀 매핑

래더에서 쓰는 핀 번호(LD)와 실제 아두이노 핀 번호를 나란히 보여 주는 표입니다. 보드가 정의한 핀 전체가 **LD·아두이노** 두 열로 나열되며, **"사용 안 함" 상태인 핀은 흐리게** 표시됩니다(그 행에 마우스를 올리면 "사용 안 함" 툴팁이 뜹니다).

> **새로 만든 프로젝트에서는 이 표가 처음부터 끝까지 전부 흐리게 보입니다.** 핀의 "사용" 여부는 기본값이 **해제**라, 아직 아무 핀도 켜지 않은 상태이기 때문입니다. 표시가 잘못된 것이 아니며, 쓸 핀을 켜는 순간 그 행부터 또렷하게 바뀝니다. 켜는 방법은 **핀 편집**(2-4)의 "보드 핀맵 편집" — 활동바의 **래더 핀맵 설정** 아이콘이 여는 핀맵 편집기에서 **사용** 열을 체크합니다.

메모리 영역

P·M·D·C·T·R 여섯 영역의 개수가 각각의 단위(비트/워드/실수)와 함께 나열되고, 목록 아래에는 여섯 영역을 합산한 전체 SRAM 사용량이 칩 이름과 함께 표시됩니다. 보드에 따라 그 옆에 보드의 총 SRAM 용량도 함께 표시되기도 합니다.

> 이 패널은 현재 보드의 정보를 **보여 주기만** 합니다. 핀 대응을 **편집**하려면 활동바의 **래더 핀맵 설정**(RAM칩 아이콘)이 여는 핀맵 편집기를 써야 합니다 — 자세한 내용은 **핀 편집**(2-4)의 "보드 핀맵 편집"을 참고하세요. 메모리 영역의 개수를 바꾸려면 설정 메뉴의 **메모리 영역 설정**(3-5)을 씁니다(**데이터 메모리**(2-1) 참고).

> 이 "래더설정" **사이드 패널**과 환경설정 창의 **"래더 설정" 탭**(**환경설정** (3-1-1)의 네 탭 중 하나 — 심볼 표시 방식·래더 사용 여부 등을 정합니다)은 이름만 비슷할 뿐 서로 다른 화면입니다. 이 패널은 현재 보드의 핀·메모리 정보를 읽기 전용으로 보여 주고, 환경설정의 "래더 설정" 탭은 래더를 어떻게 표시하고 다룰지를 정하는 옵션입니다.



< 그림 9-3 래더설정 패널 — 메모리 영역과 SRAM 합산(읽기 전용). 각 섹션은 제목을 눌러 접고 펼 수 있다 >

9-4 라이브러리 패널

라이브러리 패널이란

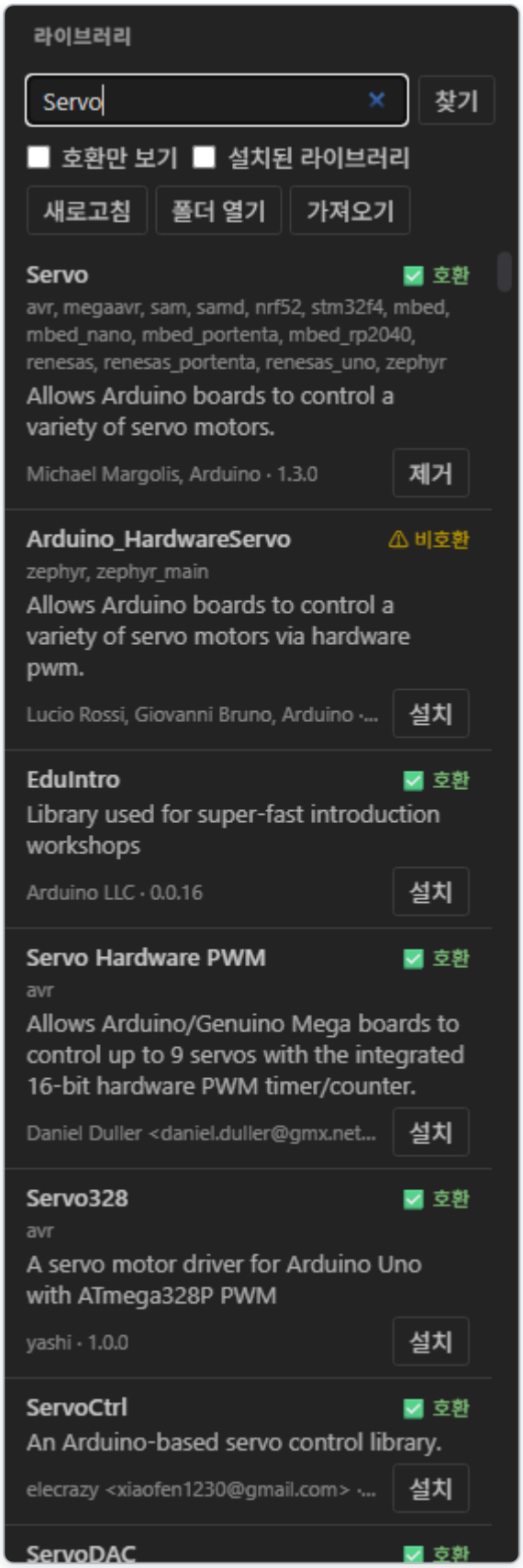
활동바의 "라이브러리" 아이콘을 누르면 열리는 사이드 패널입니다. 아두이노 라이브러리를 검색·설치·가져오기·관리하는 화면으로, arduino-cli의 라이브러리 기능을 그대로 감싸고 있습니다. 여기서 설치한 라이브러리의 함수와 상수는 코드 편집기 자동완성 목록에 나타납니다 — 자세한 내용은 **자동완성**(7-3)을 참고하세요.

검색과 설치

패널 위쪽 검색창(placeholder "라이브러리 검색...")에 이름을 입력하면 입력을 멈춘 뒤 잠시 있다가 자동으로 검색됩니다. 검색창 옆 [찾기] 버튼은 온라인 라이브러리 목록(레지스트리) 자체를 새로 내려받아 갱신합니다.

검색 결과 각 항목에는 라이브러리 이름과 호환 배지("✅ 호환" 또는 "⚠ 비호환"), 지원 아키텍처(전체 아키텍처를 지원하면 표시하지 않고, 특정 아키텍처로 제한된 경우에만 나열), 한 줄 설명, 만든 이·버전이 표시되고, 오른쪽에는 설치 여부에 따라 [설치] 또는 [제거] 버튼이 있습니다. 검색창 아래 두 체크박스 **호환만 보기**와 **설치된 라이브러리**로 목록을 걸러 볼 수 있습니다(**설치된 라이브러리**를 체크하면 검색 결과 대신 지금 설치된 라이브러리 목록으로 화면이 바뀝니다 — 아래 "설치된 라이브러리 관리와 배포" 참고).

> 호환 배지는 참고용 힌트일 뿐입니다. 많은 라이브러리가 특정 아키텍처를 지정하지 않고 "모든 보드 지원"으로 등록돼 있어서, 배지가 실제 동작까지 100% 보장하지는 않습니다.

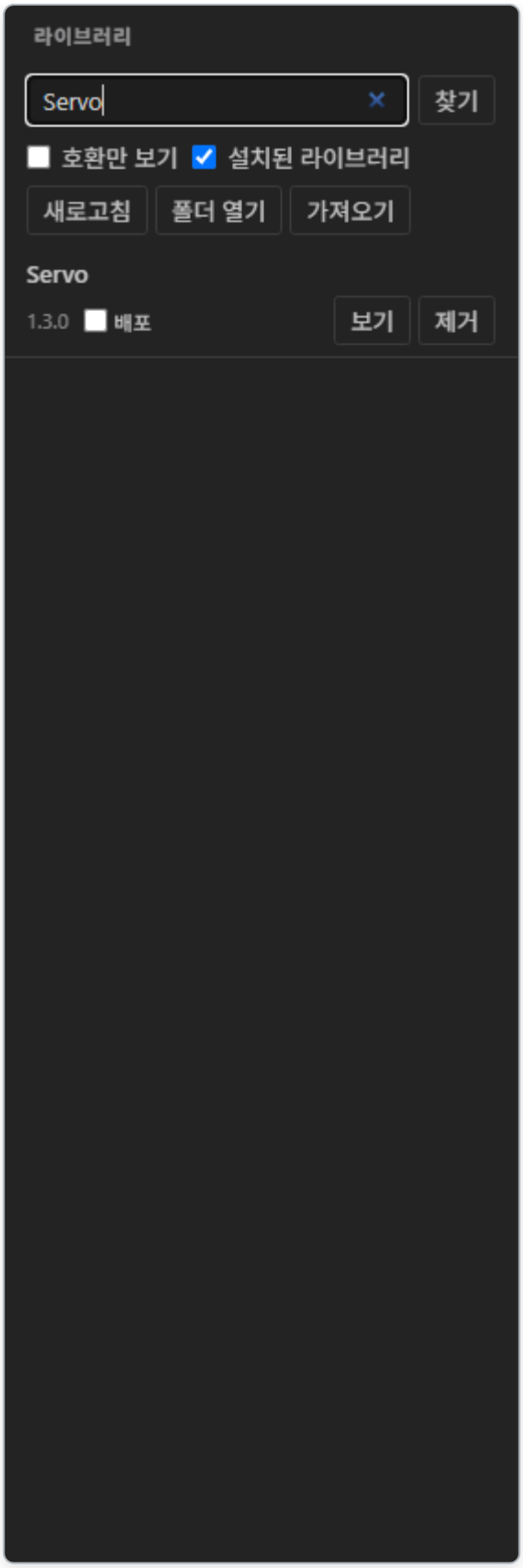


< 그림 9-4 라이브러리 검색 — 이름·호환 배지·설치 여부에 따른 설치 또는 제거 버튼과 필터 체크박스 >

설치된 라이브러리 관리와 배포

체크박스 **설치된 라이브러리**를 켜면 검색어와 무관하게 지금 이 PC에 설치된 라이브러리 전체가 목록으로 뜹니다(레지스트리에서 설치한 것, 아래 "가져오기"로 넣은 것, 그리고 라이브러리 폴더에 직접 넣고 [새로고침]한 것 모두 포함). 각 항목에는 이름과 버전이 보이고, [보기]를 누르면 그 라이브러리가 설치된 폴더가 열리며, [제거]를 누르면 삭제됩니다.

항목마다 **배포** 체크박스가 있습니다. 이 체크박스를 켜면 그 라이브러리가 `.mp2` 프로젝트를 저장할 때 프로젝트 파일 안에 함께 담깁니다 — 다른 PC에서 그 `.mp2`를 열어도 라이브러리를 따로 설치할 필요가 없습니다. 단, 앱이 기본으로 제공하는 라이브러리(벤더 채널로 받은 사본)는 배포 체크박스 자체가 없습니다. 프로그램을 설치하면 같은 사본이 이미 함께 제공되므로 프로젝트 안에 중복으로 담을 필요가 없기 때문입니다.



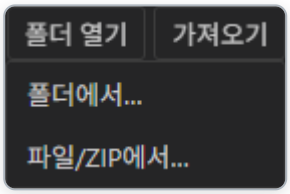
< 그림 9-5 설치된 라이브러리 — 보기·제거와 .mp2에 함께 담는 배포 체크박스 >

가져오기(오프라인 라이브러리)

레지스트리에 없는 라이브러리는 [가져오기] 버튼을 눌러 직접 넣을 수 있습니다. 버튼을 누르면 "폴더에서..."와 "파일/ZIP에서..." 두 메뉴가 뜹니다.

- "폴더에서..."는 라이브러리 폴더를 통째로 골라 넣습니다.
- "파일/ZIP에서..."는 ZIP 파일이나 낱개 소스 파일을 고릅니다. ZIP을 고르면 바로 설치되지만, 낱개 소스 파일을 고르면 라이브러리 이름을 물어보는 창이 뜹니다(이름은 비워둘 수 없고 `\ / : * ? " < > |` 같은 문자는 쓸 수 없습니다). ZIP과 낱개 소스 파일을 한 번에 섞어 고르면 "ZIP과 소스 파일을 섞어 선택할 수 없습니다 — 따로 가져오세요"라는 안내와 함께 거부되니, 둘은 따로 나눠 가져와야 합니다.

라이브러리 폴더에 직접 파일을 끌어다 놓았다면(가져오기를 거치지 않은 경우) [새로고침] 버튼으로 그 변경을 반영합니다 — 반영에는 짧으면 약 3초, 길면 최대 1분 정도 걸릴 수 있고 그동안 자동완성이 잠시 느려지거나 비어 보일 수 있습니다. 라이브러리 폴더 구조를 갖추지 못한 채 놓인 파일이 있으면 패널에 "인식 안 된 파일 N개" 안내가 뜨는데, 이때는 [가져오기] → "파일/ZIP에서..."로 정식으로 설치해야 빌드와 자동완성에 반영됩니다. 라이브러리가 실제로 놓인 폴더를 탐색기에서 바로 보려면 [폴더 열기]를 누릅니다.



< 그림 9-6 가져오기 — 폴더에서 또는 파일/ZIP에서 오프라인 라이브러리를 넣는다 >

> **환경설정**(3-1-1)의 "일반" 탭에 있는 **사용자 라이브러리 폴더**는 라이브러리를 어느 폴더에서 읽고 쓸지 **위치를 지정**하는 설정이고, 이 라이브러리 패널은 그 폴더에 든 라이브러리를 **실제로 검색·설치·관리**하는 화면입니다. 폴더 위치를 바꾸고 싶다면 환경설정에서, 라이브러리 자체를 다루고 싶다면 이 패널에서 작업하세요.

9-5 심볼

심볼은 `D0`, `M10`처럼 메모리 주소에 사람이 읽기 쉬운 별명을 붙이는 기능입니다(예: `D0` → `Temp`). 별명을 붙여 두면 래더의 접점·박스함수·비교식과 C 코드에서 주소 대신 그 이름을 쓸 수 있어 회로가 한결 읽기 쉬워집니다.

> 여기서 말하는 "심볼"은 **주소의 별명**입니다. **자동완성**(7-3)에 나오는 "라이브러리의 심볼"(라이브러리·보드 코어가 제공하는 함수·상수)과는 다른 개념이니 혼동하지 마세요.

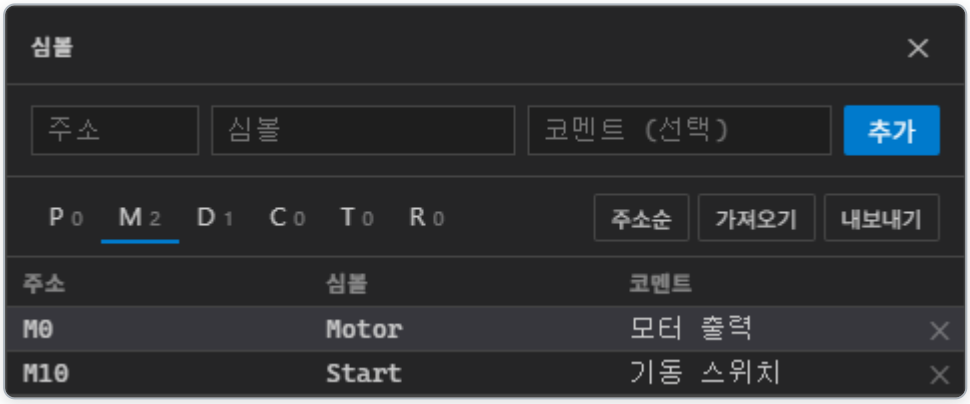
9-5-1 심볼 창 열기

활동바의 "심볼" 아이콘(태그 모양)을 누르거나 단축키 `Ctrl+Enter`를 누르면 열립니다. 이 창은 다른 활동바 모달과 달리 **모달리스**입니다 — 창을 띄운 채로도 래더 편집기를 그대로 조작할 수 있고, 헤더를 잡고 끌어서 화면 안 원하는 위치로 옮길 수 있습니다. 같은 아이콘이나 `Ctrl+Enter`를 다시 누르거나 Esc를 눌러 닫습니다. 단, 래더 셀 편집처럼 다른 곳에 입력 포커스가 가 있는 상태에서 누른 Esc는 그 입력을 취소하는 데 먼저 쓰이고 심볼 창은 닫지 않습니다. `Ctrl+Enter`는 텍스트 입력 중(코드 편집기·셀 입력칸 등)에는 동작하지 않습니다 — 입력칸에서는 평소처럼 입력을 확정하는 데 쓰입니다. 메뉴 항목은 없습니다.

9-5-2 화면 구성

메모리 영역별로 **P·M·D·C·T·R 6개 탭**이 있고, 각 탭은 주소 첫 글자로 심볼을 나누어 보여 줍니다(예: `D` 탭에는 `D`로 시작하는 주소만 나열). 탭에는 그 영역에 등록된 심볼 개수가 함께 표시됩니다. 탭 아래 표는 **주소** / **심볼** / **코멘트** 세 열로 구성됩니다.

표 위 [주소순] / [심볼순] 버튼은 지금 목록이 정렬된 기준을 보여 주며, 누르면 반대 기준으로 전환됩니다. 이 정렬은 **화면에 보이는 순서만** 바꿀 뿐, 파일에 저장되는 심볼 순서나 데이터 자체에는 영향을 주지 않습니다.



< 그림 9-7 심볼 창 — P·M·D·C·T·R 탭과 주소·심볼·코멘트 열(모달리스, 옮길 수 있음) >

9-5-3 등록·편집·삭제

맨 위 입력줄(순서대로 placeholder "주소" / "심볼" / "코멘트 (선택)")에 값을 채우고 [추가] 버튼을 누르거나 Enter를 치면 새 심볼이 등록됩니다. 등록에 성공하면 그 주소가 속한 탭으로 화면이 자동 전환되고, 입력줄은 비워진 채 다음 항목을 이어서 입력할 수 있게 포커스가 유지됩니다.

이미 등록된 행은 원하는 칸을 클릭하면 바로 고칠 수 있습니다(인라인 편집). Enter를 치거나 칸 밖을 클릭하면(포커스 이탈) 값이 저장되고, Esc를 누르면 편집을 취소하고 원래 값으로 되돌아갑니다. 행 끝의 × 버튼은 확인창 없이 그 자리에서 바로 삭제합니다.

> 심볼 등록·편집·삭제(주소를 바꿀 때 함께 재작성되는 래더 셀 포함)는 래더 편집의 실행취소(Ctrl+Z)에 포함되지 않습니다. 되돌리려면 값을 다시 손으로 고쳐야 합니다.

9-5-4 이름 규칙

주소와 이름 모두에 문법 규칙이 있고, 어기면 그 행 자체가 등록되지 않거나 (이미 등록된 행이라면) 빨갛게 강조되며 표 아래에 이유가 함께 표시됩니다.

- **주소**는 영역 글자(P/M/D/C/T/R) 하나 뒤에 숫자가 오는 형태(`P0`, `D10` 처럼)여야 합니다. 예: `심볼주소 'D0X' 문법 위반`.
- **이름**은 비어 있으면 안 되고, 첫 글자는 문자(한글 포함)나 `_`여야 하며, 주소와 똑같은 형태이거나 `@`로 시작해서는 안 됩니다. 예: `심볼명이 비어 있음`.
- **주소·이름 모두 중복을 허용하지 않습니다.** 이름 중복은 대소문자를 구분하지 않고 판정합니다 (`Motor`와 `motor`는 같은 이름으로 취급). 예: `심볼 주소 중복 `D0``, `심볼명 중복 `Motor`` (주소·이름 중복 메시지는 실제로 값 앞뒤를 백틱으로 감싸 표시합니다).



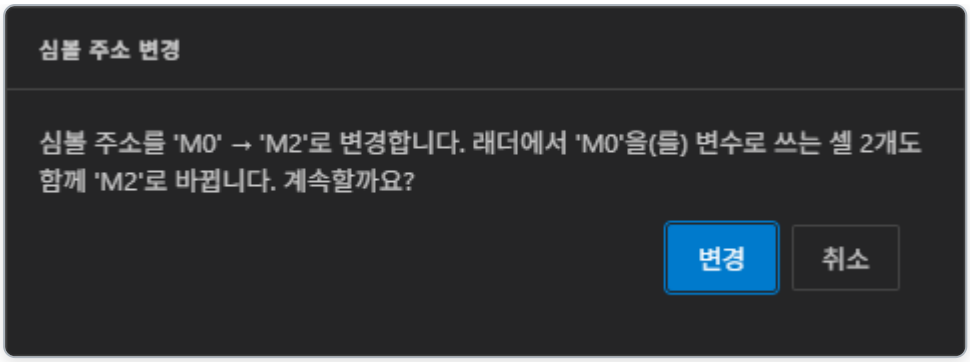
< 그림 9-8 이름 규칙 위반 — 규칙을 어긴 입력은 등록되지 않고 표 아래에 이유가 표시된다 >

9-5-5 주소·이름을 바꾸면

주소를 바꾸면 그 주소를 쓰던 래더 셀의 값도 함께 새 주소로 바뀝니다 — 되돌리기 어려운 변경이므로 바꾸기 전에 다음과 같은 확인창이 뜹니다: "심볼 주소를 '...' → '...'로 변경합니다. 래더에서 '...'을(를) 변수로 쓰는 셀 N개도 함께 '...'로 바뀝니다. 계속할까요?" [변경]을 눌러야 실제로 적용되고, 취소하면 심볼과 셀 모두 그대로 남습니다(부분 적용 없음). 영향받는 셀이 하나도 없으면 확인창 없이 바로 바뀝니다.

이름만 바꾸면 그 심볼을 가리키던 셀이 자동으로 새 이름을 따라갑니다 — 확인창은 뜨지 않습니다.

> 래더 셀에는 항상 **실제 주소**가 저장되고, 심볼은 그 주소를 표시하고 입력할 때 편의를 위한 별명일 뿐입니다. 그래서 이름만 바꾸는 것은 회로 동작에 아무 영향이 없습니다.



< 그림 9-9 주소 변경 확인 — 함께 바뀔 래더 셀 개수를 알려 준다 >

9-5-6 가져오기·내보내기

[가져오기] / [내보내기] 버튼으로 심볼 목록을 파일로 주고받을 수 있습니다. 지원 형식은 **Excel(.xlsx)** 과 **CSV(.csv)** 두 가지이며, CSV의 열 순서는 주소 / 심볼 / 코멘트입니다.

> 가져오기는 **지금 목록을 통째로 새 목록으로 바꿉니다**(기존 목록과 합치지 않습니다). 그리고 파일 안 한 줄이라도 이름 규칙을 어기면 **가져오기 전체가 취소**되고 기존 목록이 그대로 유지됩니다 — 일부만 반영되는 일은 없습니다.

9-5-7 C 코드에서 쓰기

빌드하면 등록된 심볼마다 `#define 이름 영역[번호]` 형태의 매크로가 자동 생성되어(예: `M0`에 `Motor`라는 이름을 붙였다면 `#define Motor M[0]`), C 코드에서 그 이름으로 해당 메모리를 그대로 읽고 쓸 수 있습니다.

단, 다음에 해당하는 심볼은 이 매크로에서 **제외**됩니다 — 헤더 파일에 "(제외)" 주석으로 이유가 함께 남습니다.

- 이름이 C 식별자 문법에 맞지 않는 경우(한글 등)
- 이름이 C/C++ 예약어인 경우
- 이름이 메모리 영역 이름(P/M/D/C/T/R)이나 다른 생성 코드의 이름과 겹치는 경우

즉 한글로 지은 심볼은 래더 회로와 비교식에는 그대로 쓸 수 있지만, C 코드에서 이름으로 접근하는 매핑에는 나타나지 않습니다 — C 코드에서 그 메모리를 다루려면 주소를 직접 씁니다.

> 아직 등록하지 않은 주소나 이름을 래더 셀에 먼저 입력해도 값은 그대로 저장되며, 나중에 그 주소나 이름으로 심볼을 등록하면 자동으로 연결됩니다. 자세한 동작은 **래더 편집: P0 접점에서 P32 코일까지**(1-5)와 **래더 셀 편집**(2-4-2)에서 설명한 지연 해결과 같습니다.

9-6 그 밖의 활동바 항목

앞서 9-2부터 9-5까지 다룬 탐색기·래더설정·라이브러리·심볼 네 아이콘은 사이드 패널이나 모달리스 창을 엽니다. 나머지 세 아이콘 — 래더 핀맵 설정·시리얼 모니터·설정 — 은 사이드 패널이 아니라 모달이나 하단 패널을 열며, 이미 다른 장에서 자세히 다루므로 여기서는 무엇을 여는지와 자세한 설명을 어디서 볼 수 있는지만 짧게 안내합니다.

래더 핀맵 설정

RAM칩 모양 아이콘(또는 단축키 `Ctrl+Shift+P`)을 누르면 현재 프로젝트가 쓰는 보드의 핀 대응을 편집하는 모달이 열립니다. 앞서 본 **래더설정 패널**(9-3)이 핀 매핑을 읽기 전용으로 보여 주기만 하는 것과 달리, 이 아이콘은 그 대응을 실제로 고칠 수 있는 편집 화면으로 이어집니다. 각 핀의 **사용** 여부를 켜고 끄는 곳도 이 화면입니다.

옵션 모듈을 붙인 프로젝트에서는 보드 자체의 핀뿐 아니라 그 모듈이 만들어 낸 핀(MPAINO 파생 핀)도 같은 표에서 함께 편집합니다. 그래서 보드 선택 모달의 **[핀 편집...]** 버튼으로 여는 화면과 이 아이콘으로 여는 화면은 같은 프로젝트에서 항상 같은 행을 보여 줍니다. 두 경로 모두 편집 결과는 **지금 열려 있는 프로젝트(`.mp2`)에 저장**되며, 디스크에 남기려면 `Ctrl+S`로 저장해야 합니다 — 보드 정의 파일 자체는 바뀌지 않으므로 다른 프로젝트에는 영향이 없습니다. 단, **새로 만들기**(Ctrl+N)의 보드 선택 창에서 연 핀 편집은 공장 기본값(모든 핀 사용 안 함)에서 시작하고, 그 결과는 새로 만들 프로젝트에 들어갑니다 — 그때까지 열려 있던 프로젝트는 바뀌지 않습니다.

자세한 내용은 **핀 편집**(2-4)의 "보드 핀맵 편집"을 참고하세요.

시리얼 모니터

이 아이콘은 사이드 패널이 아니라 화면 아래쪽 **하단 패널**의 시리얼 모니터 탭으로 이동합니다. 하단 패널이 닫혀 있었다면 열리며 시리얼 모니터 탭이 선택되고, 다른 탭이 열려 있었다면 시리얼 모니터 탭으로 전환되며, 이미 시리얼 모니터 탭이 선택돼 있었다면 하단 패널이 닫힙니다 — 위 "사이드바 열고 닫기"에서 설명한 세 갈래 동작과 같은 방식입니다. 자세한 내용은 **시리얼 모니터 개요**(8-1)와 **래더 모니터링 시작하기**(8-5)를 참고하세요.

설정

톱니 아이콘을 누르면 환경설정 모달이 열립니다. 사이드 패널과는 무관하게 별도 창으로 뜨는 화면입니다. 자세한 내용은 **환경설정**(3-1-1)을 참고하세요.