

MAXIM

MAX1258 평가 키트/평가 시스템

평가 대상: MAX1057/MAX1058/MAX1257/MAX1258

제품설명

MAX1258 평가 시스템 (EV 시스템)은 MAX1258 평가 키트 (EV 키트)과 Maxim 68HC16MODULE-DIP 마이크로컨트롤러 (μC) 모듈로 구성된다. MAX1258은 다중채널, 12비트 아날로그-디지털 컨버터 (ADC), 온도 센서, 8채널 12비트 디지털-아날로그 컨버터 (DAC), 그리고 구성 가능 범용 I/O 포트 (GPIO)를 제공한다. 이 평가 소프트웨어는 Windows®95/98/2000/XP에서 실행되며, MAX1258의 기능들을 실험할 수 있는 간편한 사용자 인터페이스를 제공한다.

PC를 사용하여 MAX1258을 종합적으로 평가하려면 완벽한 EV 시스템 (MAX1258EVC16)을 주문한다. 이미 이전의 Maxim EV 시스템과 함께 68HC16MODULE 모듈을 구입했거나 또는 다른 μC 기반 시스템에서 맞춤 사용할 경우에는 EV 키트 (MAX1258EVKIT)을 주문한다.

이 시스템은 또한 MAX1057/MAX1058/MAX1257을 평가할 수 있다. 자세한 내용은 하드웨어 세부설명 부분을 참조한다. 이 제품들의 무료 샘플에 대해서는 공장에 문의한다.

MAX1258 독립형 EV 키트

MAX1258 EV 키트는 MAX1258 평가를 쉽게 해주는 입증된 PC 보드 레이아웃을 제공한다. 이 EV 키트는 적합한 타이밍 신호에 인터페이싱되어야 올바르게 동작한다. 사용자가 제공한 6VDC~28VDC 전원과 접지를 단자 블록 TB1에 되연결함으로써 온 보드 MAX1615 LDO에 전력을 공급한다. 타이밍 요구사항에 대해서는 그림 7과 MAX1258 데이터 시트를 참조한다.

MAX1258 EV 시스템

MAX1258 EV 시스템은 사용자가 제공한 7VDC~20VDC 전원으로 동작한다. 이 평가 소프트웨어는 IBM PC의 Windows 95/98/2000/XP에서 실행되며, 컴퓨터의 직렬 통신 포트를 통해 EV 시스템 보드에 인터페이싱된다. 설정 및 동작 지침에 대해서는 퀵 스타트를 참조한다.

SUPPLIER	PHONE	FAX	WEBSITE
Johanson Dielectric	818-364-9800	818-364-6100	www.johanson-caps.com
Murata	770-436-1300	770-436-3030	www.murata.com
Panasonic	714-373-7366	714-737-7323	www.panasonic.com
Taiyo Yuden	800-348-2496	847-925-0899	www.t-yuden.com
TDK	847-803-6100	847-390-4405	www.component.tdk.com

참고: 위 부품 공급업체에 문의할 경우 MAX1258 사용자임을 알린다.

Windows는 Microsoft Corporation의 등록상표이다.



제품특징

- ◆ 입증된 PC 보드 레이아웃
- ◆ 완벽한 평가 시스템
- ◆ 편리한 온 보드 테스트 포인트
- ◆ 데이터 로깅 소프트웨어
- ◆ 완전 조립 및 테스트 통과

주문정보

PART	TEMP RANGE	INTERFACE TYPE
MAX1258EVKIT	0°C to +70°C	User supplied
MAX1258EVC16	0°C to +70°C	Windows software

참고: MAX1258 평가 소프트웨어는 완벽한 평가 시스템 MAX1258EVC16 (68HC16MODULE-DIP 모듈과 MAX1258EVKIT을 포함)과 함께 사용하도록 설계되었다. MAX1258 평가 소프트웨어를 사용하지 않을 경우, MAX1258EVKIT 보드는 μC 없이 그 자체만으로 구입 가능하다.

MAX1258EVC16 시스템 부품목록

PART	QTY	DESCRIPTION
MAX1258EVKIT	1	MAX1258 evaluation kit
68HC16MODULE-DIP	1	68HC16 μC module

부품 공급업체

MAX1258 평가 키트/평가 시스템

MAX1258EVKIT 부품목록

REFERENCE	QTY	DESCRIPTION
C1, C14, C19, C21	4	0.1 μ F \pm 10%, 16V X7R ceramic capacitors (0805) Murata GRM219R71C104K Johanson 250R15W104KV4Z TDK C2012X7R1C104K-0.85
C2-C13, C15, C16, C23	15	0.01 μ F \pm 10% ceramic capacitors (0603) Taiyo Yuden UMK107B103KZ TDK C1608X7R1H103K Murata GRM188R71H103K
C17	1	470pF \pm 10%, 50V X7R ceramic capacitor (0603) Taiyo Yuden UMK107B471KZ TDK C1608X7R1H471K Murata GRM188R71H471K
C18	1	100pF \pm 5%, 50V C0G ceramic capacitor (0603) Murata GRM1885C1H101J Taiyo Yuden UMK107CH101JZ TDK C1608C0G1H101J
C20, C22	2	1 μ F \pm 10%, 10V X7R ceramic capacitors (0805) Murata GRM21BR71A105K Taiyo Yuden LMK212BJ105KG TDK C2012X7R1A105K
C24, C25, C26	3	10 μ F \pm 20%, 6.3V X5R ceramic capacitors (0805) TDK C2012X5R0J106M Taiyo Yuden JMK212BJ106MG Panasonic ECJ2FB0J106M

REFERENCE	QTY	DESCRIPTION
H1-H4	4	12 pins
H5, H6, H7	3	2 x 4 dual row header pins
H8	1	2 x 5 dual row header pin
J1	1	2 x 20 right angle socket
JU1, JU2	2	3 pins
JU4, JU5	2	2 pins
R18	1	100k Ω 5% resistor (1206)
R19	1	10k Ω 5% resistor (1206)
R1-R17, R22, R23	19	10 Ω 5% resistors (1206)
R20	1	510 Ω 5% resistor (1206)
R21	1	2k Ω 5% resistor (1206)
TB1	1	Two-circuit terminal block
U1	1	MAX1258BETM (48-pin, TQFN, 7mm x 7mm)
U2	1	MAX1615EUK-T, ABZD, SOT23-5
U3, U4	2	MAX1840EUB (μ MAX-10) or MAX1841EUB
—	6	Shunts (JU1: 1-2, JU2: 2-3, JU4: 1-2, JU5: 1-2, H8: 1-2, H8: 9-10)
—	1	PC board, MAX1258 EV kit

퀵 스타트

필요 장비

시작하기 전에 다음 장비가 필요하다.

- Maxim MAX1258EVC16 (MAX1258EVKIT 보드와 68HC16MODULE-DIP을 포함)
- DC 전원, +7V~+20VDC, 0.25A
- Windows 95/98/2000/XP PC와 사용 가능한 직렬 (COM) 포트
- 9핀 I/O 연장 케이블

절차

모든 연결이 완료될 때까지 전원을 켜지 않도록 한다.

- JU1이 5V 위치에 있고, JU2가 1-2 위치에 있고, JU5가 폐쇄되어 있는지 확인한다. JU4는 개방되어야 한다.

점퍼 위치 JU3은 이전에 잘라내지 않은 한 폐쇄되어 있다. 표 2-6 참조.

- MAX1258 EV 키트의 40핀 헤더와 68HC16MODULE-DIP 모듈의 40핀 커넥터를 맞추어 보드를 조심스럽게 연결한다. 보드를 동시에 살짝 누른다. 두 보드는 서로 맞대어 있어야 한다.
- +7V~+20V DC 전원 소스를 온/오프 스위치 옆에 위치한 단자 블록의 μ C 모듈에 연결한다. 이때 μ C 모듈의 최상단 에지를 따라 연결하고, 보드에 표시된 극성에 주의한다.
- 컴퓨터의 직렬 포트에서 나온 케이블을 μ C 모듈에 연결한다. 9핀 직렬 포트를 사용하는 경우, 직통 9핀 암-수 케이블을 사용한다. 유일하게 사용 가능한 직렬 포트가 25핀 커넥터를 사용할 경우, 표준 25핀 대 9핀 어댑터가 필요하다. EV 키트 소프트웨어는 모뎀 상태 회전 (CTS, DSR, DCD)을 점검하여 정확한 포트가 선택되었는지 확인한다.

MAX1258 평가 키트/평가 시스템

평가 대상: MAX1057/MAX1058/MAX1257/MAX1258

- 5) 플로피 디스크에 있는 INSTALL.EXE 프로그램을 실행하여 컴퓨터에 평가 소프트웨어를 설치한다. 프로그램 파일들이 복사되고 Windows 시작 메뉴에 프로그램 아이콘이 만들어진다.
- 6) 전원장치를 켜다.
- 7) 시작 메뉴에서 MAX1258 프로그램 아이콘을 열어 프로그램을 시작한다.
- 8) 프로그램의 지시에 따라 μC 모듈을 연결하고 모듈 전원을 켜다. SW1을 ON 위치로 민다. 정확한 직렬 포트를 선택하고 OK를 클릭한다. 이제 프로그램은 자동으로 소프트웨어를 모듈로 다운로드한다.
- 9) 입력 신호를 AIN0에 적용하고 **Perform Action**을 클릭한다. 화면 내용을 주의 깊게 읽는다. 조치를 반복해서 수행하려면, **every 200ms** 상자를 선택한다.
- 10) 측정 그래프를 보려면, **View** 메뉴를 풀다운하여 **Graph**를 클릭한다.
- 11) **DAC Outputs** 탭을 열고, **1111 cccc cccc 001x Power On Selected Channels** 조치를 선택한 다음, **Perform Action**을 클릭한다.
- 12) **1100 Write and Load OUT0-OUT7**을 선택하고, OUT1 코드를 2048로 설정한 다음, **Perform Action**을 클릭한다. OUT0의 전압이 이제 중간값 ($V_{REF} = 4.096V$ 라고 가정할 때, 2.048V)임을 확인한다.
- 13) **GPIO Pins** 탭을 연다. GPIOA0을 **High Output**으로 설정하고, GPIOB0을 **Low Output**으로 설정하고, GPIOC3을 Input으로 설정한 다음, **Write Output Pins**를 클릭한다. 이제 A0 핀은 로직 하이이고 B0 핀은 로직 로우임을 확인한다.
- 14) GPIO 핀 C3을 로직 하이 (A0)에 연결하고 **Read Input Pins**를 클릭한다. 소프트웨어가 C3을 하이로 표시하고 있음을 확인한다.
- 15) GPIO 핀 C3을 로직 로우 (B0)에 연결하고 **Read Input Pins**를 클릭한다. 소프트웨어가 C3을 로우로 표시하고 있음을 확인한다.

소프트웨어 세부설명

평가 소프트웨어의 메인 창은 데이터 컨버터를 구성하고 아날로그 입력을 측정한다. **Action**에서, 스캐닝 순서, 반복 변환 또는 단일 변환을 선택한다. 선택된 각각의 채널의 측정 결과는 해당 **Measurements Results** 필드에 표시된다.

Action 설정 **read single channel repeatedly**는 선택된 채널이 측정되어야 할 횟수를 결정하기 위해 **Repetition**에서 선택이 이루어져야 한다.

각각의 선택된 채널에 대한 일련의 측정 결과를 산술적 중간값으로 요약하려면 **Averaging**을 사용한다. **Repetition**은 **Averaging**과 함께 사용되어 다수의 측정 결과를 소수의 샘플 평균값으로 요약한다.

Low-Level Interface Details 패널은 가장 최근의 로우 레벨 레지스터 쓰기 작업을 보여준다. 각 레지스터에 기록된 내용의 요약은 **Low-level registers** 탭에서 제공된다.

AIN14와 AIN15 채널은 설정 레지스터에서 대체 기능이 선택될 경우 자동으로 생략된다. 평가 소프트웨어는 대체 기능이 활성화 또는 비활성화될 때마다 디스플레이를 업데이트하여 이러한 채널을 표시하거나 숨긴다.

Setup 탭은 AIN14와 AIN15 핀을 위한 대체 기능들을 구성하고, 인접 채널을 차동 입력 쌍으로 구성한다.

Low-level registers 탭에는 액티브 구성을 생성하는 명령이 요약되어 있다. **Reset All Registers** 버튼은 이러한 소프트웨어에 의해 숨겨진 레지스터 값을 재설정하고 MAX1258로 재설정된 명령을 전송한다. 저속 모드와 밴드갭 모드는 옵션으로 구성된다.

DAC 출력

메인 창의 **DAC Outputs** 탭은 아날로그 출력 핀을 제어한다. 사용하기 전에 DAC 채널에 전력을 공급하려면, **1111 cccc cccc 001x Power On Selected Channels** 조치를 선택한다. 확인란이 설정되어 있는지 확인하고 **Perform Action**을 클릭한다. 점퍼 JU2는 DAC 출력 핀의 초기 상태를 정의한다.

모든 DAC 출력을 0으로 재설정하려면, **0001 0... Reset all DACs to 000 (zero scale)** 조치를 선택하고 **Perform Action**을 클릭한다.

모든 DAC 출력을 최대값으로 재설정하려면, **0001 1... Reset all DACs to FFF (full scale)** 조치를 선택하고 **Perform Action**을 클릭한다.

참고: 파워 온 시 DAC는 REF1에서 외부 기준 전압이 있을 것으로 기대한다. MAX1258 데이터 시트를 참조한다.

여러 개의 DAC 채널을 기록하고 부하하려면, **1100 Write and Load OUT0-OUT7** 조치를 선택하고, **OUT0** 편집 필드에 원하는 출력 코드 값 (0~4095)를 입력한 다음, **Perform Action**을 클릭한다. OUT0-OUT7 핀의 전압은 즉시 새로운 값으로 변경된다.

MAX1258 평가 키트/평가 시스템

단일 DAC 출력 (예를 들어, OUT5)을 기록하려면, **0110 Write OUT5** 조치를 선택한다. **OUT5** 편집 필드에 원하는 출력 코드 값 (0~4095)을 입력하고 **Perform Action**을 클릭한다. 점퍼 JU5가 폐쇄되어 있지 않는 한 (LDAC 핀 로우를 주장), 입력 레지스터로부터 핀 전압을 업데이트하려면 별도 부하 명령이 필요하다. **1110 cccc cccc xxxx Load Selected Channels** 조치를 선택하고, 채널 5의 Load 확인란에 표시한 다음, **Perform Action**을 클릭하여 OUT5를 부하한다.

GPIO 핀

메인 창의 **GPIO Pins** 탭은 범용 디지털 입/출력 핀을 구성, 기록 및 판독한다.

각각의 GPIO 핀은 해당 핀의 **High Output, Low Output, Input** 또는 **Open-Drain Pull-Down** 모드를 구성하는 드롭다운 콤보 상자를 가진다. 출력 모드를 선택하고, **Write Output Pins**를 클릭한다. **Read Input Pins**를 클릭하여 핀을 읽는다.

샘플링

측정 데이터는 외부 클럭 모드에서 샘플링될 수 있다. **Setup** 탭에서, **Clock Mode**를 **0111xxxx ext clock**로 설정한 다음, **Measurement** 탭으로 돌아가 **Get Samples**를 클릭한다.

그래프 창

최근에 측정된 데이터를 보려면, **View** 메뉴를 드롭다운하여 **graph**를 선택한다. 데이터는 시간 순서 플롯, 히스토그램 플롯 또는 원시 숫자 표로 조회될 수 있다 (그림 6 참조). 사용 가능한 그래프 명령은 표 1을 참조한다.

진단 창

진단 창은 평가 키트를 선적하기 전 공장 테스트를 위해 사용된다. 이 창은 고객 사용을 위한 창이 아니다.

하드웨어 세부설명

테스트 중인 MAX1258 소자 (U1)는 다중채널 12비트 ADC, 온도 센서, 8채널 12비트 DAC 및 구성 가능한 GPIO를 제공한다. 저항 R1-R16과 커패시터 C1-C16은 각 입력에 대해 단극 저역 통과 앤티 앨리어싱 필터를 형성한다. 커패시터 C17은 U1을 위한 전원 바이패싱을 제공한다. 그림 7과 MAX1258 데이터 시트를 참조한다.

이 EV 키트는 MAX1257과 5V μ C를 함께 사용할 수 있도록 지원하기 위해 MAX1615 3V/5V 선형 레귤레이터 (U2)와 일련의 MAX1840/MAX1841 레벨 슈프터 (U3과 U4)를 포함하고 있다.

MAX1257 평가

MAX1257은 MAX1258의 3V 버전이다. MAX1257BETM의 무료 샘플을 요청한다. U1을 MAX1257로 교체하고 JU1 선트를 3V 위치로 이동시킨다. 소프트웨어의 **options** 메뉴에서 **reference = 2.500V**를 선택한다.

MAX1057 평가

MAX1057은 MAX1257의 3V, 10비트 버전이다. MAX1057BETM의 무료 샘플을 요청한다. U1을 MAX1057로 교체하고 JU1 선트를 3V 위치로 이동시킨다. 소프트웨어의 **Options** 메뉴에서 **reference = 2.500V**를 선택한다.

평가 소프트웨어는 12비트 데이터를 기대하지만, MAX1057은 10비트의 의미있는 데이터만을 제공한다. 가장 중요한 비트 (MSB)는 정렬되어 있기 때문에, 소프트웨어에 의해 보고된 측정 코드 숫자는 실제 측정 코드 숫자의 4배이다. 재구성된 전압 값은 영향을 받지 않는다. **Options** 메뉴를 선택하고 **Sub-LSBs**를 2로 설정하여 그래프 창을 조정한다.

MAX1058 평가

MAX1058은 MAX1258의 10비트 버전이다. MAX1058BETM의 무료 샘플을 요청한다. U1을 MAX1058로 교체하고 JU1 선트를 5V 위치로 이동시킨다.

평가 소프트웨어는 12비트 데이터를 기대하지만, MAX1058은 10비트의 의미있는 데이터만을 제공한다. MSB가 정렬되어 있기 때문에, 소프트웨어에 의해 보고되는 측정 코드 숫자는 실제 측정 코드 숫자의 4배이다. 재구성된 전압 값은 영향을 받지 않는다. **Options** 메뉴를 선택하고 **Sub-LSBs**를 2로 설정하여 그래프 창을 조정한다.

외부 기준전압 사용

모든 ADC 또는 DAC는 변환을 수행할 기준 전압이 필요하다. ADC의 내부 기준 전압을 사용하는 동안 MAX1258 DAC를 위해 단일 종단 외부 기준 전압이 적용될 수 있다. 이것이 기본 모드이다. 전력이 꺼진 상태에서 DAC 외부 기준 전압을 REF1에 연결한다. 그 다음, 시스템의 전원을 켜고 평가 소프트웨어를 실행한다. **Setup** 탭에서, **Reference Input**을 **01xx10xx Pin 48=AIN14, ADCREF=Internal, DACREF=REF1**로 설정한다. JU4에 선트를 설치하여 온 보드 REF1 바이패스 커패시터 C14를 연결한다. 소프트웨어가 DAC 코드 값에 해당하는 전압을 계산할 때마다 그 계산은 사용자가 제공한 **REF1 pin voltage** 값에 따라 달라진다.

ADC와 DAC 모두에 대해 내부 기준전압을 사용하려면, **Setup** 탭에서 **Reference Input**을 **01xx00xx Pin 48=AIN14, ADCREF=Internal, DACREF=Internal**로 설정한다. 내부 기준전압을 사용할 때에는 JU4를 개방해 둔다.

독립된 단 일 종단 기준전압은 DAC와 ADC 모두에 대해 적용될 수 있다. 전원이 꺼진 상태에서, DAC 외부 기준 전압을 REF1에 연결하고 ADC 외부 기준전압을 REF2에

MAX1258 평가 키트/평가 시스템

평가 대상: MAX1057/MAX1058/MAX1257/MAX1258

연결한다. 그 다음, 시스템의 전원을 켜고 평가 소프트웨어를 실행한다. **Setup** 탭에서, **Reference Input**을 **01xx01xx Pin 48=REF2, ADCREF=REF2, DACREF=REF1**으로 설정한다. JU4에 선트를 설치하여 온 보드 REF1 바이패스 커패시터 C14를 연결한다. 소프트웨어가 DAC 코드 값에 해당하는 전압을 계산할 때마다 이 계산은 사용자가 제공한 **REF1 pin voltage** 값에 따라 달라진다. 소프트웨어가 ADC 코드 값에 해당하는 전압을 계산할 때마다 이 계산은 사용자가 제공한 **REF2 pin voltage** 값에 따라 달라진다.

차동 외부 기준전압이 ADC에 적용되는 동안 단일 종단 외부 기준전압이 DAC를 위해 적용될 수 있다. 전원이 꺼진 상태에서 DAC 외부 기준전압을 REF1에 연결하고, REF1 > REF2가 되도록 REF1과 REF2 사이에 ADC 외부 기준전압을 연결한다. 그 다음 시스템의 전원을 켜고 평가 소프트웨어를 실행한다. **Setup** 탭에서, **Reference Input**을 **01xx11xx Pin 48=REF2, ADCREF=REF1-REF2, DACREF=REF1**로 설정한다. JU4에 선트를 설치하여 온 보드 REF1 바이패스 커패시터 C14를 연결한다.

표 1. 그래프 도구 버튼

도구	기능
	사용 가능한 전체 입력 범위를 표시.
	그래프 데이터를 확장하여 창을 채움.
	뷰를 왼쪽 또는 오른쪽으로 이동.
	뷰를 위 또는 아래로 이동.
	x축을 확대 또는 축소.
	y축을 확대 또는 축소.
	파일에서 데이터를 불러옴.
	데이터를 파일에 저장.
	데이터를 저장할 때 헤더를 작성하기 위한 옵션.
	데이터를 저장할 때 라인 수를 기록하기 위한 옵션.
	코드 대 시간 플롯으로 보기.
	히스토그램 플롯 (각 코드의 누적 주파수)으로 보기.
	테이블로 보기.
Min	표 보기에 최소값을 표시.
Max	표 보기에 최대값을 표시.
Span	표 보기에 범위를 표시. <DD> 범위 = 최대값 - 최소값.
N	표 보기에 샘플 수를 표시.
Sum(x)	표 보기에 샘플 합계를 표시.
Sum(x*x)	표 보기에 샘플 제곱의 합을 표시.
Mean	표 보기에 산술 평균을 표시. $\text{Mean} = \frac{\sum(x)}{n}$

도구	기능
StdDev	표 보기에 표준 편차를 표시. $\text{표준 편차} = \sqrt{\frac{n\sum(x^2) - \left(\sum x\right)^2}{(n-1)n}}$
Rms	표 보기에 제곱의 평균의 루트 값 (RMS)을 표시. $\text{RMS} = \sqrt{\frac{\sum(x^2)}{n}}$
0	채널 0 인에이블.
1	채널 1 인에이블.
2	채널 2 인에이블.
3	채널 3 인에이블.
4	채널 4 인에이블.
5	채널 5 인에이블.
6	채널 6 인에이블.
7	채널 7 인에이블.
8	채널 8 인에이블.
9	채널 9 인에이블.
10	채널 10 인에이블.
11	채널 11 인에이블.
12	채널 12 인에이블.
13	채널 13 인에이블.
14	채널 14 인에이블.
15	채널 15 인에이블.
16	채널 16 인에이블 (온도).

MAX1258 평가 키트/평가 시스템

표 2. 접퍼 JU1 (V_{DD} 전압 선택)

선택 위치	V _{DD} 전압	기능
1-2*	5V	U1 = MAX1058 또는 MAX1258인 정상 동작.
2-3	3V	U1 = MAX1057 또는 MAX1257인 정상 동작.
개방	비지정	JU1이 개방된 상태에서 EV 키트를 동작하지 않도록 한다.

*기본 구성

표 3. 접퍼 JU2 (RES_SEL)

선택 위치	RES_SEL 상태	기능
1-2	하이	전원이 처음 적용될 때, DAC 출력은 내부 100kΩ 저항을 통해 REF1로 풀린다. 모든 DAC 입력 레지스터는 0xFFFF로 설정된다.
2-3*	로우	전원이 처음 적용될 때, DAC 출력은 내부 100kΩ 저항을 통해 AGND로 풀린다. 모든 DAC 입력 레지스터는 0x000으로 설정된다.
개방	비구동	JU2가 개방된 상태에서 EV 키트의 전원을 켜지 않도록 한다.

*기본 구성

표 4. 옵션 접퍼 JU3 (AIN15 대체 기능)

JU3 상태	U1 PIN1 연결	기능
폐쇄*	μC 모듈 J1 핀 29에 연결됨 (레벨 쉬프터를 통해)	U1 핀 1 = CNVST 변환 시작 명령. AIN15 패드는 연결시 키지 않는다.
개방	AIN15 패드에 연결	U1 핀 1 = AIN15 아날로그 입력. 신호 소스를 AIN15 패드에 연결한다.

*기본 구성

표 5. 접퍼 JU4 (REF1 바이패스)

선택 위치	REF1 바이패스 커패시터	기능
개방	사용자 제공	내부 기준 전압을 사용할 경우 JU4는 개방 상태로 둔다.
폐쇄*	C14는 REF를 바이패스함.	외부 기준 전압을 사용할 경우 JU4는 폐쇄한다.

*기본 구성

표 6. 접퍼 JU5 (LDAC)

선택 위치	LDAC 상태	기능
개방	하이	SPI를 통해 명령을 전송하여 DAC 출력을 업데이트한다.
폐쇄*	로우	DAC 입력 레지스터는 DAC 출력 레지스터로 전송된다.

*기본 구성

소프트웨어가 DAC 코드 값에 해당하는 전압을 계산할 때마다 이 계산은 사용자가 제공한 **REF1 pin voltage** 값 및 **REF2 pin voltage** 값에 따라 달라진다.

문제 해결

문제: 출력 측정 없음. 시스템이 0 전압을 보고하는 것 같거나 측정을 수행하지 못한다.

해결책: V_{DD} 전원 전압을 확인한다. 디지털 전압계를 사용하여 기준전압을 확인한다. 오실로스코프를 사용하여 변환 시작 신호가 스트로브되고 있는지 확인한다.

문제: 측정이 불규칙, 불안정 또는 부정확하다.

해결책: 디지털 전압계를 사용하여 기준전압을 확인한다. 오실로스코프를 사용하여 잡음을 확인한다. 잡음을 검출할 때, 오실로스코프 접지 반사 리드를 가능한 한 짧게 유지한다. 0.5인치 (10mm) 미만이 좋다.

문제: 트랜스듀서를 측정할 때 수용할 수 없는 오류가 발생한다.

해결책: 대부분의 신호 소스는 MAX1258의 아날로그 입력에 직접 연결될 수 있지만, 일부 하이 임피던스 신호 소스 (300Ω 이상)는 입력 버퍼가 필요할 수 있다. 수집 시간 (내부 클럭 모드 01)을 늘려 안정화 오류를 확인한다. 필요 시, MAX4430을 사용하여 하이 임피던스 신호 소스를 버퍼한다. MAX1258 데이터 시트를 참조한다.

MAX1258 평가 킷/평가 시스템

평가 대상: MAX1057/MAX1058/MAX1257/MAX1258

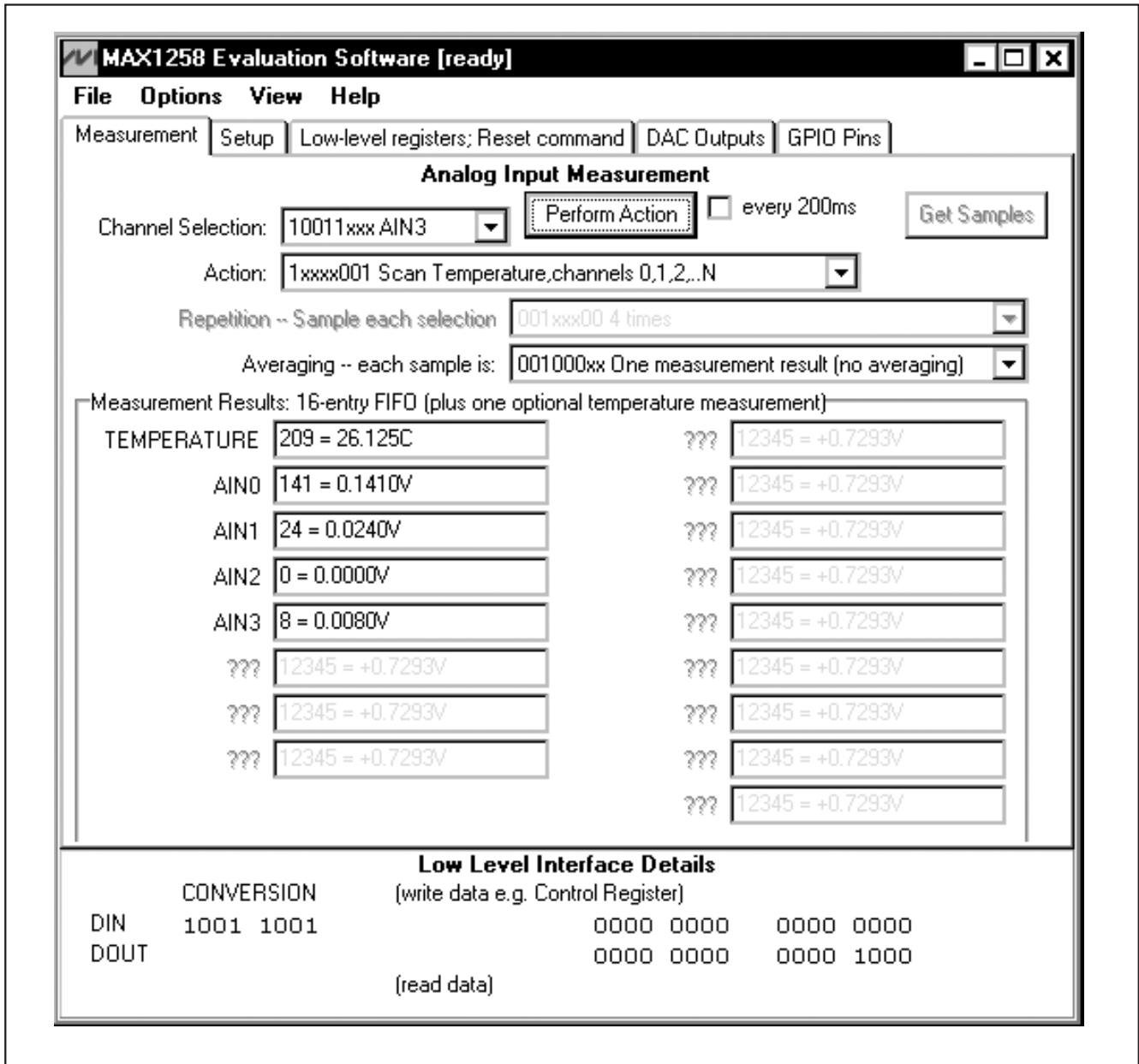


그림 1. MAX 1258 평가 소프트웨어의 메인 창 — 데이터 컨버터를 구성하고 아날로그 입력을 측정

MAX1258 평가 키트/평가 시스템

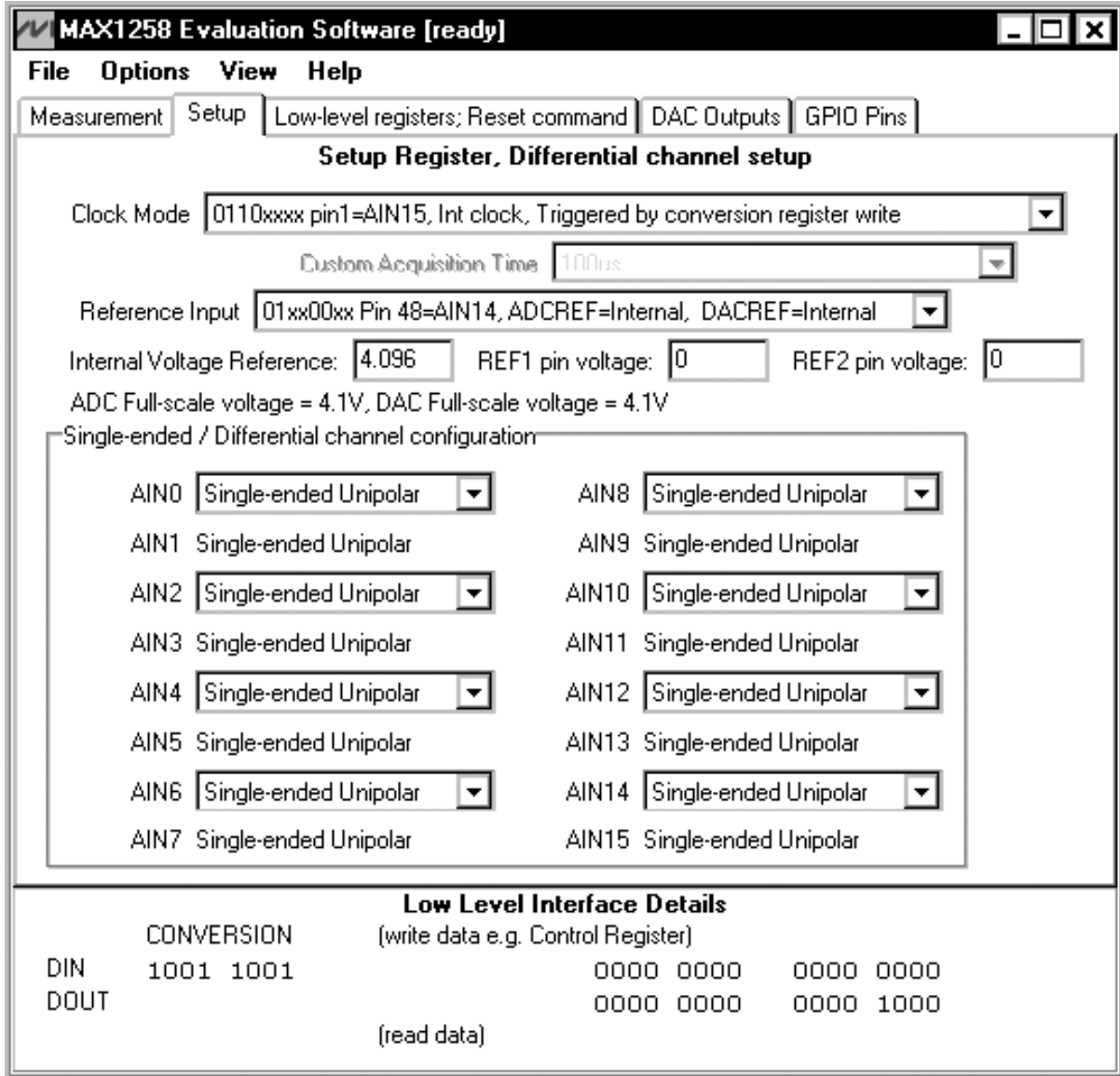


그림 2. 메인 창의 설정 탭 — AIN14 및 AIN15를 위한 대체 기능을 구성하고, 인접 채널을 차동 입력 쌍으로 구성

MAX1258 평가 키트/평가 시스템

평가 대상: MAX1057/MAX1058/MAX1257/MAX1258

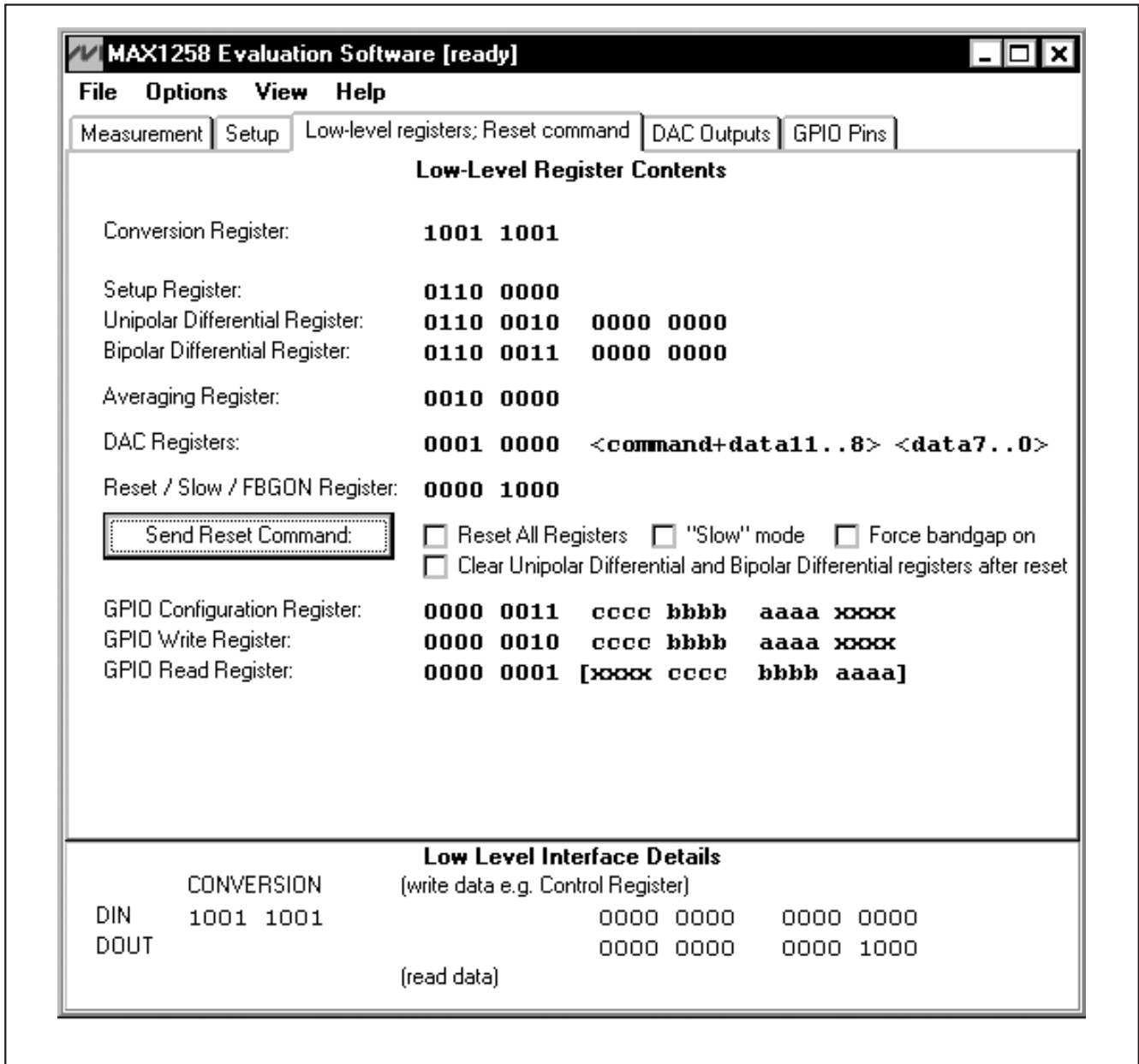


그림 3. 메인 창의 로우 레벨 레지스터 — 액티브 구성을 생성하는 명령을 요약

MAX1258 평가 킷/평가 시스템

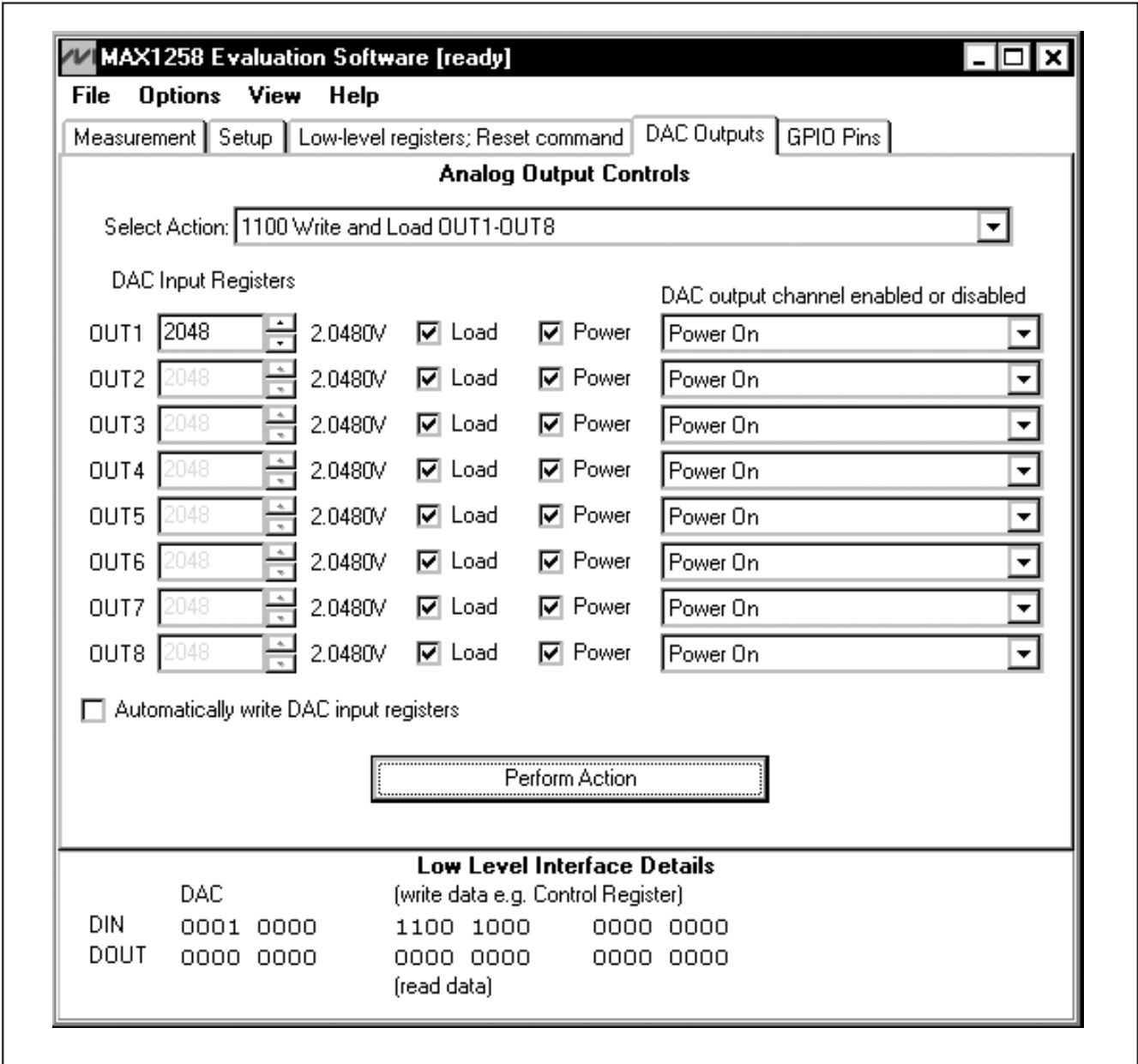


그림 4. 메인 창의 DAC 출력 탭 — 아날로그 출력 핀 제어

MAX1258 평가 키트/평가 시스템

평가 대상: MAX1057/MAX1058/MAX1257/MAX1258

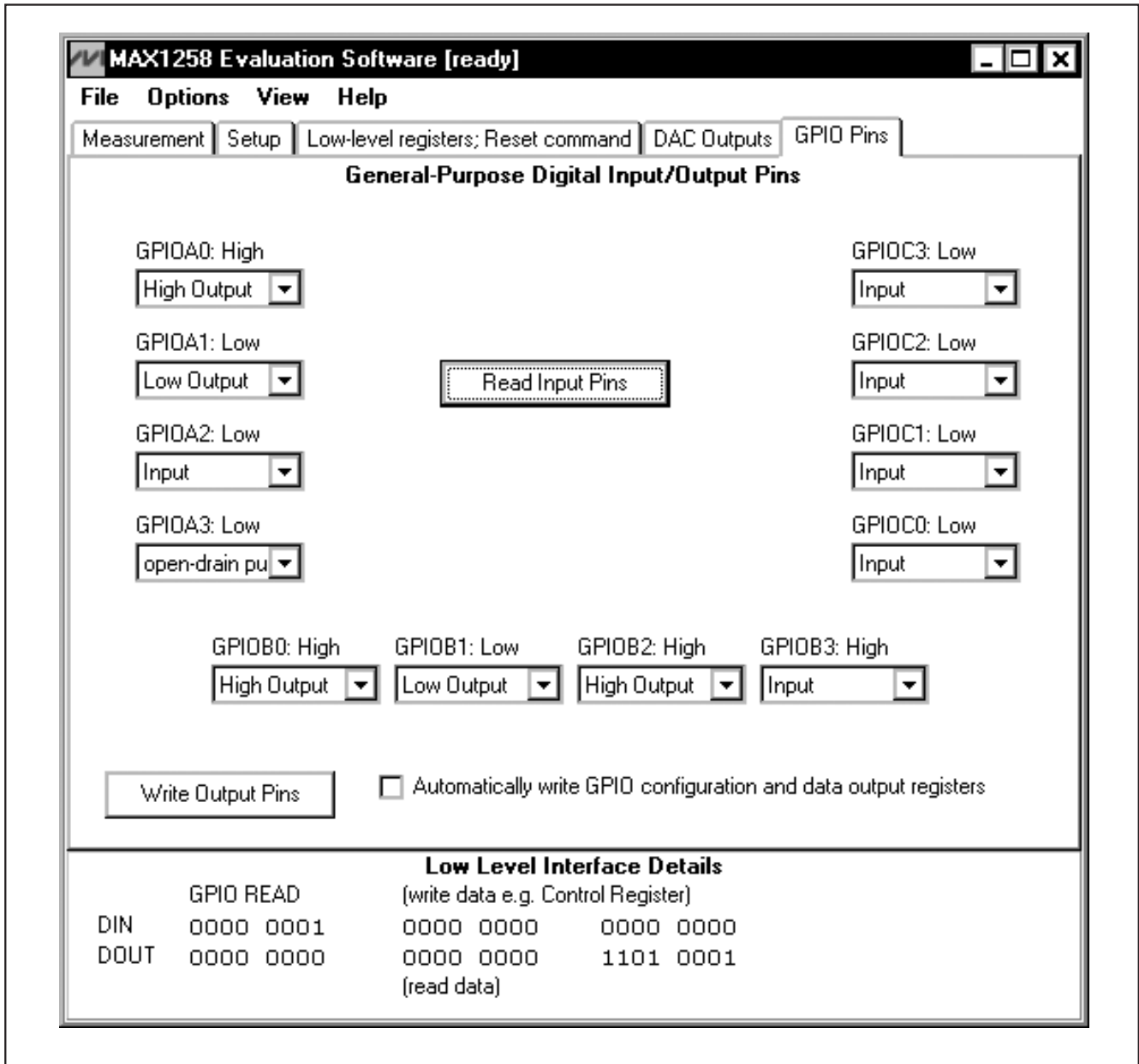


그림 5. GPIO 핀 탭 — 디지털 GPIO 핀을 구성, 기록 및 판독

MAX1258 평가 키트/평가 시스템

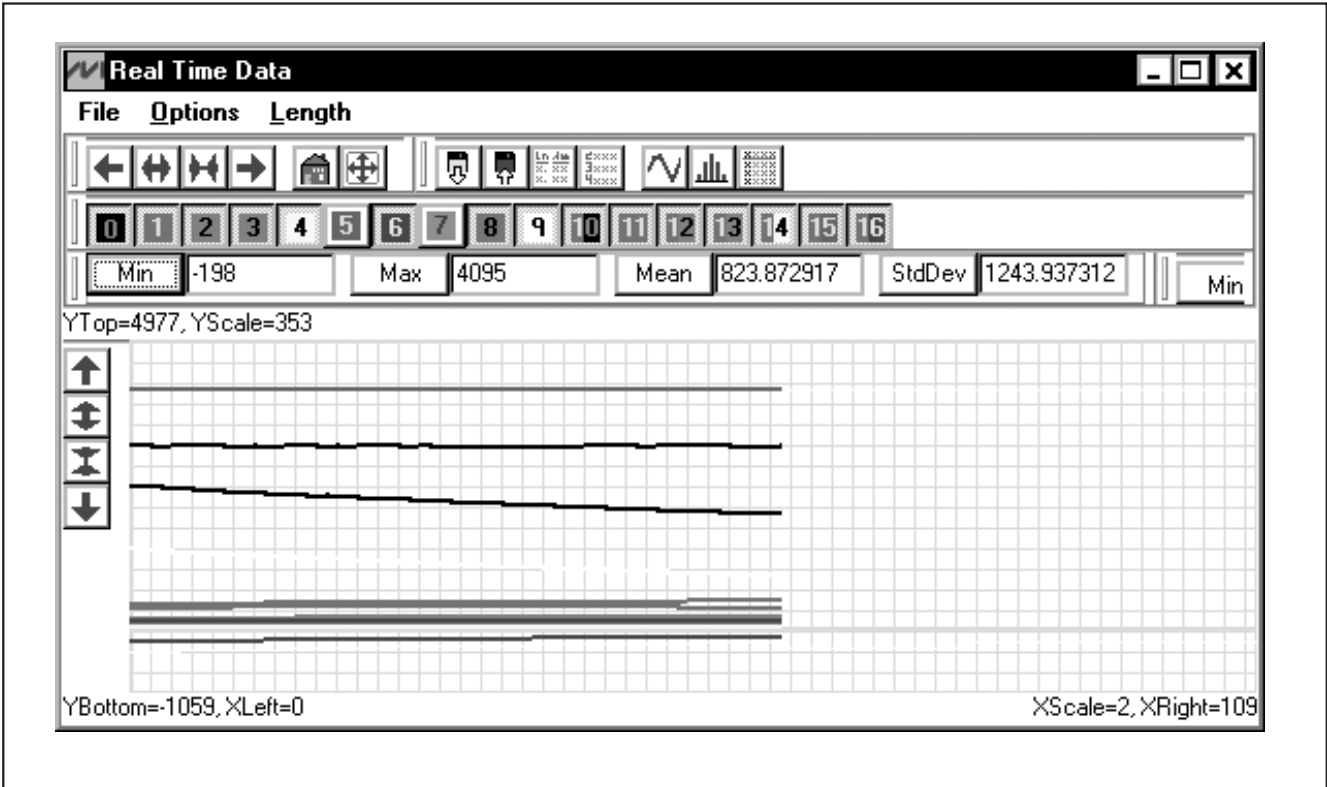


그림 6. 실시간 데이터 및 샘플링된 데이터 그래프 — 데이터를 시간 순서, 히스토그램 또는 표로 표시

MAX1258 평가 키트/평가 시스템

평가 대상: MAX1057/MAX1058/MAX1257/MAX1258

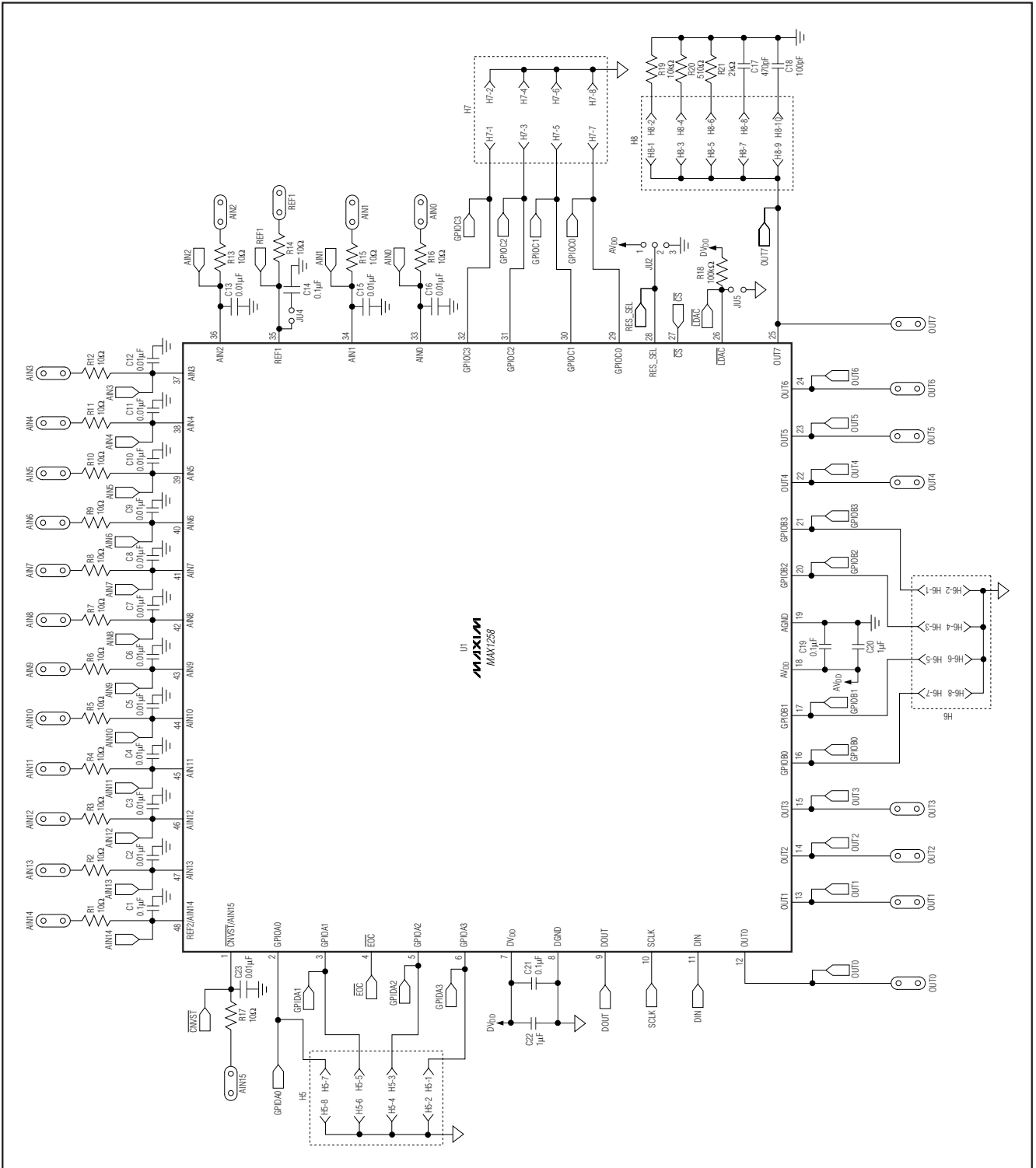


그림 7a. MAX1258 EV 키트 회로도 (1/2)

MAX1258 평가 키트/평가 시스템

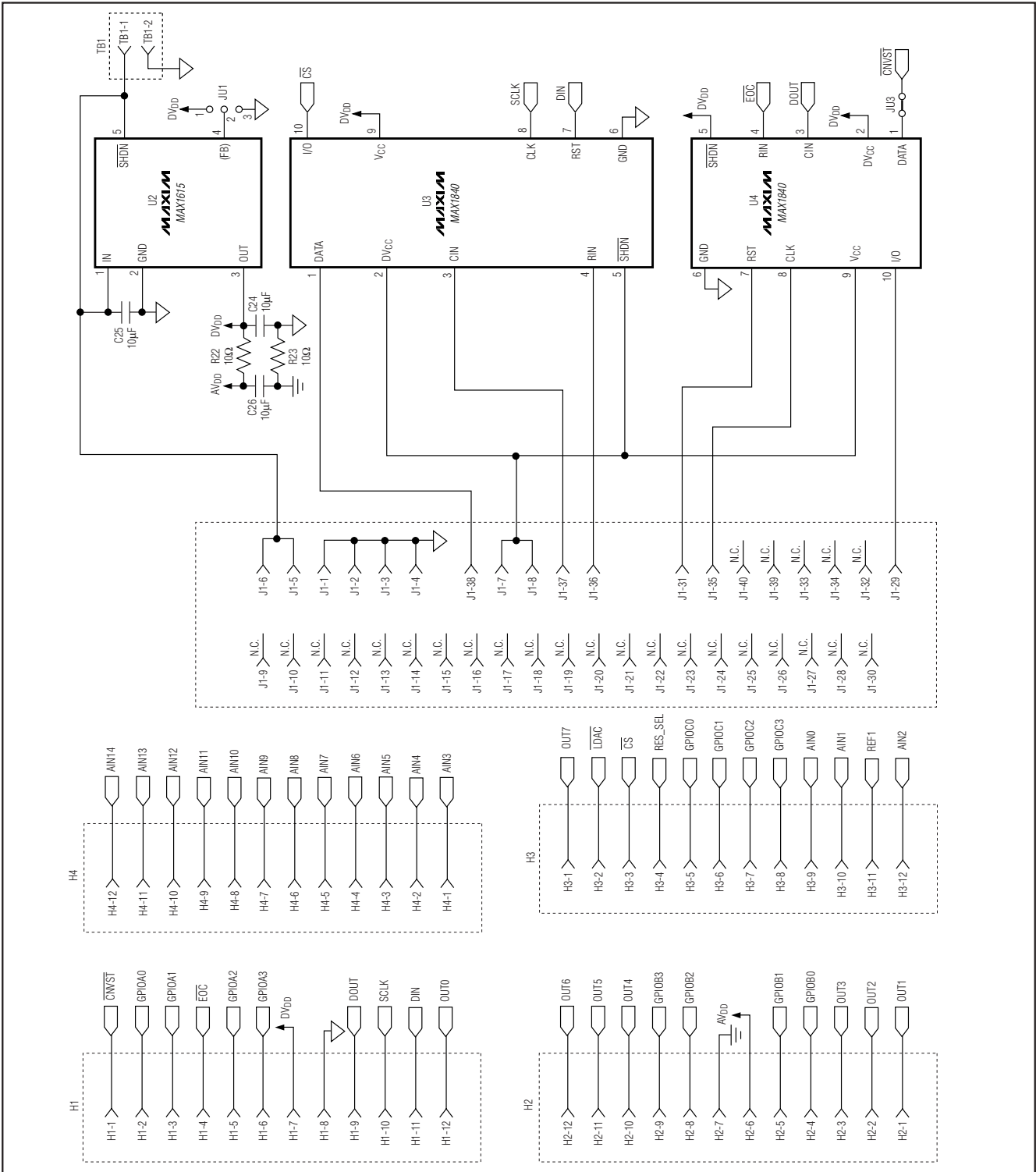


그림 7b. MAX1258 EV 키트 회로도 (2/2)

MAX1258 평가 키트/평가 시스템

평가 대상: MAX1057/MAX1058/MAX1257/MAX1258

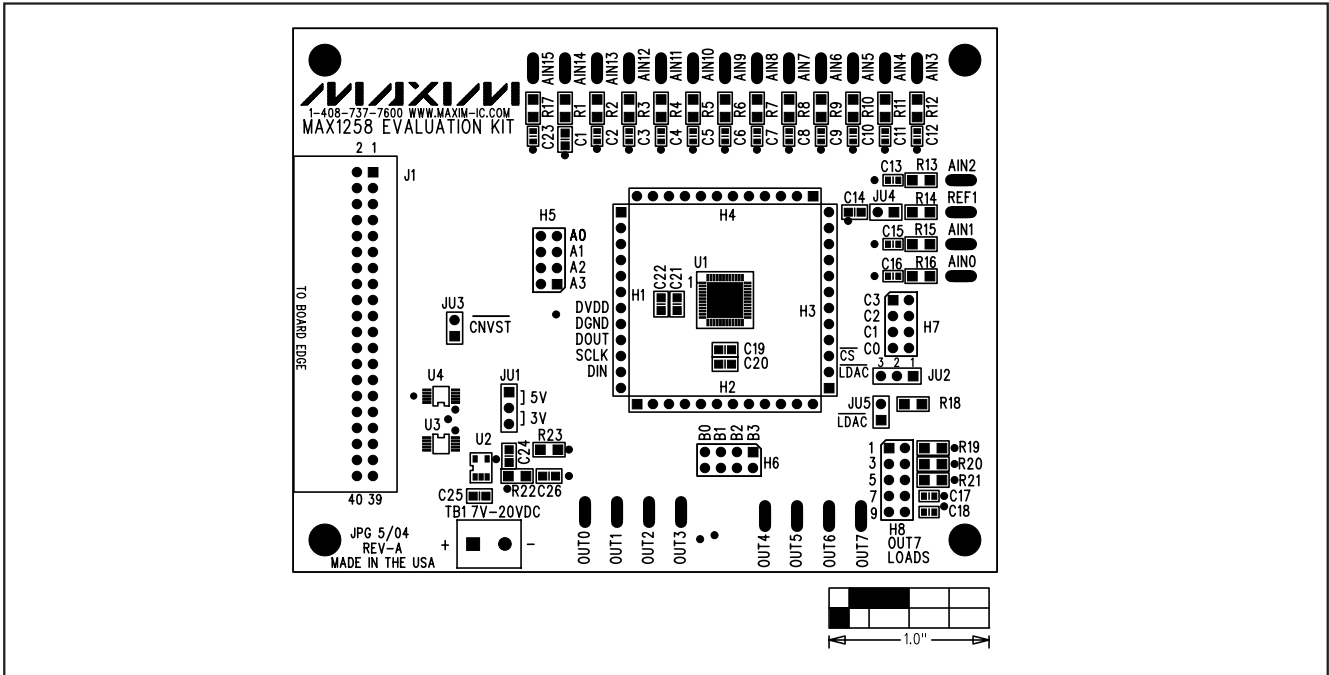


그림 8. MAX1258 EV 키트 부품 배치 가이드 — 부품 측

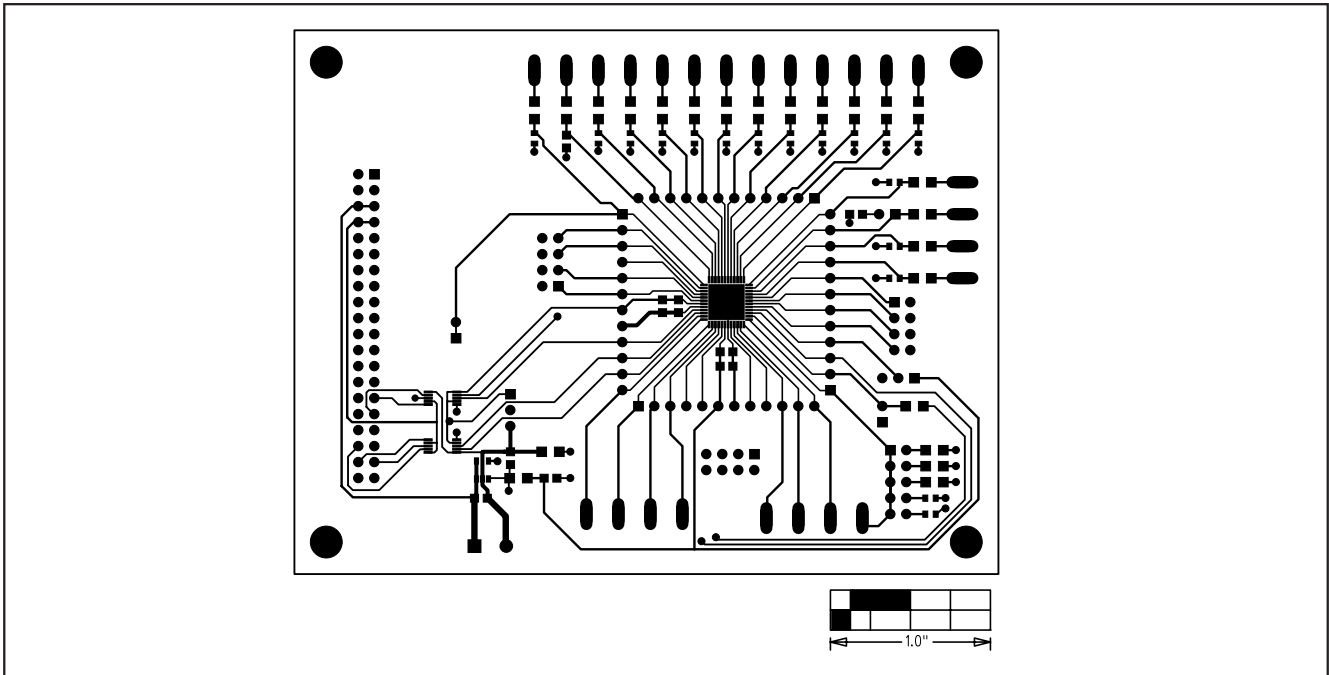


그림 9. MAX1258 EV 키트 PC 보드 레이아웃 — 부품 측

MAX1258 평가 키트/평가 시스템

평가 대상: MAX1057/MAX1058/MAX1257/MAX1258

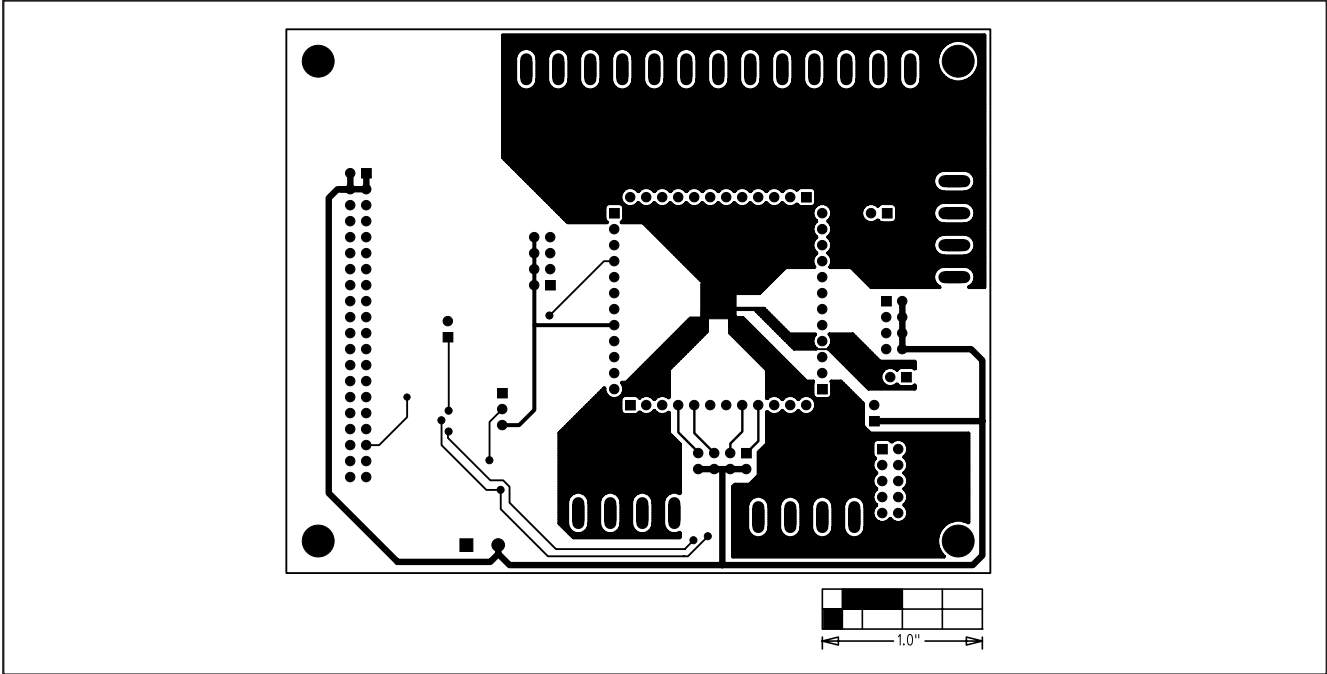


그림 10. MAX1258 EV 키트 PC 보드 레이아웃 — 납땜 측

MAX1258 평가 키트/평가 시스템

평가 대상: MAX1057/MAX1058/MAX1257/MAX1258

```
MAX1258 EV kit Listing 1          06/01/04          1
MAX1258EV listing1

// Drv1258.h
// MAX1258-specific driver.
// mku 04/07/2004
// (C) 2004 Maxim Integrated Products
//-----
// Revision history (latest at top):
// =====
// __/__/2004: Initial Release as MAX1258 Version 1.0
// =====
//-----
#ifndef DRV1258H
#define DRV1258H
//-----

//-----
// The following interface protocols must be provided by
// the appropriate low-level interface code.
//

/* SPI interface:
**  byte_count = transfer length
**  mosi[] = array of master-out, slave-in data bytes
**  miso_buf[] = receive buffer for master-in, slave-out data bytes
**
** 04/07/2004: master-in slave-out data from MAX1258 is delayed one clock cycle.
** When instructed to transfer n bytes, the hardware must generate
** n * 8 clock pulses. However, the first bit of miso_buf[] must be sampled
** concurrent with the SECOND bit of mosi[].
** The final bit of miso_buf[] does not get a clock pulse, instead the
** final state of the MAX1258 DOUT pin is sampled prior to negating CS.
*/
extern bool SPI_Transfer_MISO_Delayed(int byte_count,
const unsigned __int8 mosi[], unsigned __int8 miso_buf[]);

// Read the state of the EOC pin until the pin is low
// or until a platform-specific timeout expires.
// On Exit:
// Return value = true if EOC pin is low
// Return value = false if timeout expires and EOC is still high
//
extern bool Wait_MAX1258_EOC_Low(void);

// Pulse the CONV pin low and then high.
//
extern void Pulse_MAX1258_CONV(void);

// Set acquisition time (when in clock mode 01)
// DelayString is of the form:
// 200us
// 500us
// 1ms
// 2ms
// 5ms
// 10ms
// 20ms
// 50ms
// 100ms
// 200ms
// 500ms
// 1s
extern bool Set_Acquisition_Time(const char* DelayString);
```

리스트 1 (1/10)

MAX1258 평가 키트/평가 시스템

MAX1258 EV kit Listing 1
MAX1258EV listing1

06/01/04

2

```

//-----
// MAX1258 Conversion register
// 1xxx xxxx
#define MAX1258_CONV 0x80
//
// Power-on state: 1000 0000
#define MAX1258_CONV_POR 0x80
//
// Channel Selection
#define MAX1258_CONV_AIN00 0x80 /* 1000xxxx AIN0 */
#define MAX1258_CONV_AIN01 0x88 /* 10001xxxx AIN1 */
#define MAX1258_CONV_AIN02 0x90 /* 10010xxxx AIN2 */
#define MAX1258_CONV_AIN03 0x98 /* 10011xxxx AIN3 */
#define MAX1258_CONV_AIN04 0xA0 /* 10100xxxx AIN4 */
#define MAX1258_CONV_AIN05 0xA8 /* 10101xxxx AIN5 */
#define MAX1258_CONV_AIN06 0xB0 /* 10110xxxx AIN6 */
#define MAX1258_CONV_AIN07 0xB8 /* 10111xxxx AIN7 */
#define MAX1258_CONV_AIN08 0xC0 /* 11000xxxx AIN8 */
#define MAX1258_CONV_AIN09 0xC8 /* 11001xxxx AIN9 */
#define MAX1258_CONV_AIN10 0xD0 /* 11010xxxx AIN10 */
#define MAX1258_CONV_AIN11 0xD8 /* 11011xxxx AIN11 */
#define MAX1258_CONV_AIN12 0xE0 /* 11100xxxx AIN12 */
#define MAX1258_CONV_AIN13 0xE8 /* 11101xxxx AIN13 */
#define MAX1258_CONV_AIN14 0xF0 /* 11110xxxx AIN14 */
#define MAX1258_CONV_AIN15 0xF8 /* 11111xxxx AIN15 */
//
// Actions
#define MAX1258_CONV_SCAN_00_N 0x80 /* 1xxxx000 Scan 0,1,2,...N */
#define MAX1258_CONV_SCAN_T_00_N 0x81 /* 1xxxx001 Scan T,0,1,2,...N */
#define MAX1258_CONV_SCAN_N_15 0x82 /* 1xxxx010 Scan N,N+1,...,15 */
#define MAX1258_CONV_SCAN_T_N_15 0x83 /* 1xxxx011 Scan T,N,N+1,...,15 */
#define MAX1258_CONV_SINGLE_REPEAT 0x84 /* 1xxxx10x Read repeatedly */
#define MAX1258_CONV_SINGLE_READ 0x86 /* 1xxxx11x Read once */
//
#define MAX1258_ACTION_MASK 0x87 /* 1xxxx111 bits to test*/

//-----
// MAX1258 Setup register
// 01xx xx00
//
// Setup register may optionally be followed by
// one of the the differential configuration registers.
// 01xxxx10 followed by a second byte, selecting Unipolar-Differential inputs
// 01xxxx11 followed by a second byte, selecting Bipolar-Differential inputs
#define MAX1258_SETUP 0x40 /* 01xxxx00 no additional bytes */
#define MAX1258_SETUP_UNIDIFF 0x42 /* 01xxxx10 followed by another byte */
#define MAX1258_SETUP_BIPDIFF 0x43 /* 01xxxx11 followed by another byte */
//
// Power-on state: 0110 0000
#define MAX1258_SETUP_POR 0x60
//
// Clock Mode
// 0100xxxx pin16=CNVST, Int clock, Triggered by CNVST pulse
// 0101xxxx pin16=CNVST, Int clock, Triggered by CNVST pulses, custom Tacq
// 0110xxxx pin16=AIN15, Int clock, Triggered by conversion register write
// 0111xxxx pin16=AIN15, Ext clock, Triggered by conversion register write
#define MAX1258_SETUP_INTCLK_CNVST 0x40 /* 0100xxxx CNVST */
#define MAX1258_SETUP_INTCLK_CNVST_TACQ 0x50 /* 0101xxxx CNVST */
#define MAX1258_SETUP_INTCLK 0x60 /* 0110xxxx AIN15 */
#define MAX1258_SETUP_EXTCLK 0x70 /* 0111xxxx AIN15 */
//
// Reference Voltage
// MAX1258: 01xx00xx Pin 48=AIN14, ADCREF=Internal, DACREF=Internal
// MAX1258: 01xx01xx Pin 48=REF2, ADCREF=REF2, DACREF=REF1

```

리스트 1 (2/10)

MAX1258 평가 키트/평가 시스템

평가 대상: MAX1057/MAX1058/MAX1257/MAX1258

MAX1258 EV kit Listing 1
MAX1258EV listing1

06/01/04

3

```
// MAX1258: 01xx10xx Pin 48=AIN14, ADCREF=Internal, DACREF=REF1
// MAX1258: 01xx11xx Pin 48=REF2, ADCREF=REF1-REF2, DACREF=REF1
#define MAX1258_SETUP_REF00 0x40 /* 01xx00xx */
#define MAX1258_SETUP_INTREF_SLEEP 0x40 /* 01xx00xx Pin 48=AIN14 */
#define MAX1258_SETUP_REF01 0x44 /* 01xx01xx */
#define MAX1258_SETUP_EXTREF 0x44 /* 01xx01xx Pin 48=REF2 */
#define MAX1258_SETUP_REF10 0x48 /* 01xx10xx */
#define MAX1258_SETUP_INTREF_ACTIVE 0x48 /* 01xx10xx Pin 48=AIN14 */
#define MAX1258_SETUP_REF11 0x4C /* 01xx11xx */
#define MAX1258_SETUP_EXTREF_DIFF 0x4C /* 01xx11xx Pin 48=REF2 */
//
//
// MAX1258 Unipolar-Differential input pairs
// Byte Following MAX1258_SETUP_UNIDIFF
// 01xx xx10 unidiff
//
// Power-on state: 0110 0010 0000 0000
#define MAX1258_SETUP_UNIDIF_POR 0x00
//
#define MAX1258_SETUP_UNIDIF0001 0x80
#define MAX1258_SETUP_UNIDIF0203 0x40
#define MAX1258_SETUP_UNIDIF0405 0x20
#define MAX1258_SETUP_UNIDIF0607 0x10
#define MAX1258_SETUP_UNIDIF0809 0x08
#define MAX1258_SETUP_UNIDIF1011 0x04
#define MAX1258_SETUP_UNIDIF1213 0x02
#define MAX1258_SETUP_UNIDIF1415 0x01
//
// MAX1258 Bipolar-Differential input pairs
// Byte Following MAX1258_SETUP_BIPDIFF
// 01xx xx11 bipdiff
//
// Power-on state: 0110 0011 0000 0000
#define MAX1258_SETUP_BIPDIF_POR 0x00
//
#define MAX1258_SETUP_BIPDIF0001 0x80
#define MAX1258_SETUP_BIPDIF0203 0x40
#define MAX1258_SETUP_BIPDIF0405 0x20
#define MAX1258_SETUP_BIPDIF0607 0x10
#define MAX1258_SETUP_BIPDIF0809 0x08
#define MAX1258_SETUP_BIPDIF1011 0x04
#define MAX1258_SETUP_BIPDIF1213 0x02
#define MAX1258_SETUP_BIPDIF1415 0x01
//
//-----
// MAX1258 Averaging register
// 001x xxxx
//
// Power-on state: 0010 0000
#define MAX1258_AVERAGE_POR 0x20
//
// Averaging
// 001000xx One measurement result (no averaging)
// 001100xx Mean of 4 measurement results
// 001101xx Mean of 8 measurement results
// 001110xx Mean of 16 measurement results
// 001111xx Mean of 32 measurement results
#define MAX1258_AVERAGE_1 0x20 /* 001000xx No averaging */
#define MAX1258_AVERAGE_4 0x30 /* 001100xx Mean of 4 measurements */
#define MAX1258_AVERAGE_8 0x34 /* 001101xx Mean of 8 measurements */
#define MAX1258_AVERAGE_16 0x38 /* 001110xx Mean of 16 measurements */
#define MAX1258_AVERAGE_32 0x3C /* 001111xx Mean of 32 measurements */
//
// Repeat Count
```

리스트 1 (3/10)

MAX1258 평가 키트/평가 시스템

```

MAX1258 EV kit Listing 1          06/01/04          4
MAX1258EV listing1

// Enabled by MAX1258_CONV_SINGLE_REPEAT 1xxxx10x
// Internal clock modes only
#define MAX1258_REPEAT_4      0x20    /* 001xxx00 4 times */
#define MAX1258_REPEAT_8      0x21    /* 001xxx01 8 times */
#define MAX1258_REPEAT_12     0x22    /* 001xxx10 12 times */
#define MAX1258_REPEAT_16     0x23    /* 001xxx11 16 times */

//-----
// MAX1258 Reset register (reset command)
// 0 0 0 1 <RESET> <SLOW> <FBGON>
//
// Reset all registers to their power-on default states
#define MAX1258_RESET_ALL      0x0C    /* 000011xx Reset All Registers */
//
// "Slow" mode
#define MAX1258_RESET_SLOW     0x0A    /* 0000101x "Slow" mode */
//
// Force bandgap and bias block to be turned on (and clear FIFO)
#define MAX1258_RESET_FBGON    0x09    /* 000010x1 Force bandgap on */

//-----
// MAX1258 GPIO (General-purpose Input/Output)
//
// MAX1220 has four GPIO pins
// MAX1221 has four GPIO pins
// MAX1257/MAX1258 have twelve GPIO pins
//
// To configure a GPIO pin for OUTPUT,
// set the corresponding bit 1 in the configuration register.
// Pin state is controlled by a bit in the write-data register.
//
// To configure a GPIO pin for INPUT,
// set the corresponding bit 0 in the configuration register
// and set the corresponding bit 1 in the write-data register.
// Pin state is returned in a bit in the read-data register.
//
//-----
// MAX1258 GPIO Configuration register
// 0 0 0 0 0 1 1 <GPIO pin masks>
//
#define MAX1258_GPIO_CONFIG    0x03    /* 00000011 next 16 bits = GPIO configuration
data */
#define MAX1220_GPIO_CONFIG    0x03    /* 00000011 next 8 bits = GPIO configuration
data */
//-----
// MAX1258 GPIO Write register
// 0 0 0 0 0 1 0 <GPIO pin masks>
//
#define MAX1258_GPIO_WRITE     0x02    /* 00000010 next 16 bits = GPIO write data */
#define MAX1220_GPIO_WRITE     0x02    /* 00000010 next 8 bits = GPIO write data */
//-----
// MAX1258 GPIO Read register
// 0 0 0 0 0 0 1 <GPIO pin masks>
//
#define MAX1258_GPIO_READ      0x01    /* 00000001 next 16 bits = GPIO read data */
#define MAX1220_GPIO_READ      0x01    /* 00000001 next 8 bits = GPIO read data */
//-----
// MAX1257/MAX1258 GPIO pin WRITE/CONGIFURE mask bits
// <GPIOC3> <GPIOC2> <GPIOC1> <GPIOC0> <GPIOB3> <GPIOB2> <GPIOB1> <GPIOB0>
// <GPIOA3> <GPIOA2> <GPIOA1> <GPIOA0> X X X X
//
// MAX1257/MAX1258 GPIO pin READ mask bits

```

리스트 1 (4/10)

MAX1258 평가 키트/평가 시스템

평가 대상: MAX1057/MAX1058/MAX1257/MAX1258

```

MAX1258 EV kit Listing 1                                06/01/04                                5
MAX1258EV listing1

//  X X X X <GPIOC3> <GPIOC2> <GPIOC1> <GPIOC0>
//  <GPIOB3> <GPIOB2> <GPIOB1> <GPIOB0> <GPIOA3> <GPIOA2> <GPIOA1> <GPIOA0>
//
// For READ, the GPIOB3..GPIOB0 pins migrate to the third byte.
//
#define MAX1258_GPIO_WRC3      0x8000 /* 1xxxxxxx xxxxxxxx */
#define MAX1258_GPIO_WRC2      0x4000 /* x1xxxxxx xxxxxxxx */
#define MAX1258_GPIO_WRC1      0x2000 /* xx1xxxxx xxxxxxxx */
#define MAX1258_GPIO_WRC0      0x1000 /* xxx1xxxx xxxxxxxx */
#define MAX1258_GPIO_WRB3      0x0800 /* xxxxlxxx xxxxxxxx */
#define MAX1258_GPIO_WRB2      0x0400 /* xxxxlxxx xxxxxxxx */
#define MAX1258_GPIO_WRB1      0x0200 /* xxxxlxxx xxxxxxxx */
#define MAX1258_GPIO_WRB0      0x0100 /* xxxxxxxx1 xxxxxxxx */
#define MAX1258_GPIO_WRA3      0x0080 /* xxxxxxxx1 xxxxxxxx */
#define MAX1258_GPIO_WRA2      0x0040 /* xxxxxxxx1 xxxxxxxx */
#define MAX1258_GPIO_WRA1      0x0020 /* xxxxxxxx1 xxxxxxxx */
#define MAX1258_GPIO_WRA0      0x0010 /* xxxxxxxx1 xxxxxxxx */
//
//-----
#define MAX1258_GPIO_RDC3      0x0800 /* xxxxlxxx xxxxxxxx */
#define MAX1258_GPIO_RDC2      0x0400 /* xxxxlxxx xxxxxxxx */
#define MAX1258_GPIO_RDC1      0x0200 /* xxxxlxxx xxxxxxxx */
#define MAX1258_GPIO_RDC0      0x0100 /* xxxxxxxx1 xxxxxxxx */
#define MAX1258_GPIO_RDB3      0x0080 /* xxxxxxxx1 xxxxxxxx */
#define MAX1258_GPIO_RDB2      0x0040 /* xxxxxxxx1 xxxxxxxx */
#define MAX1258_GPIO_RDB1      0x0020 /* xxxxxxxx1 xxxxxxxx */
#define MAX1258_GPIO_RDB0      0x0010 /* xxxxxxxx1 xxxxxxxx */
#define MAX1258_GPIO_RDA3      0x0008 /* xxxxxxxx1 xxxxxxxx */
#define MAX1258_GPIO_RDA2      0x0004 /* xxxxxxxx1 xxxxxxxx */
#define MAX1258_GPIO_RDA1      0x0002 /* xxxxxxxx1 xxxxxxxx */
#define MAX1258_GPIO_RDA0      0x0001 /* xxxxxxxx1 xxxxxxxx */
//
//-----
// GPIO pin masks for all GPIO commands
// MAX1220/MAX1221 GPIO pin WRITE/CONGIFURE mask bits
//  <GPIOC3> <GPIOC2> <GPIOA1> <GPIOA0> X X X X
//
// MAX1220/MAX1221 GPIO pin READ mask bits
//  X X X X <GPIOC3> <GPIOC2> <GPIOA1> <GPIOA0>
//
#define MAX1220_GPIO_WRC1      0x80 /* 1xxxxxxx */
#define MAX1220_GPIO_WRC0      0x40 /* x1xxxxxx */
#define MAX1220_GPIO_WRA1      0x20 /* xx1xxxxx */
#define MAX1220_GPIO_WRA0      0x10 /* xxx1xxxx */
//
#define MAX1220_GPIO_RDC1      0x08 /* xxxxlxxx */
#define MAX1220_GPIO_RDC0      0x04 /* xxxxlxxx */
#define MAX1220_GPIO_RDA1      0x02 /* xxxxxxxx1 */
#define MAX1220_GPIO_RDA0      0x01 /* xxxxxxxx1 */
//
//-----

//-----
// MAX1258 DAC Select register
// 0001 x x x x
//  <C3> <C2> <C1> <C0> <D11> <D10> <D09> <D08>
//  <D07> <D06> <D05> <D04> <D03> <D02> <D01> <D00>
//
// This 8-bit preamble is followed by 4-bit command and 12-bit data,
// described in the next table.
//
#define MAX1258_DAC      0x10 /* 0001xxxx next 16 bits = DAC command and data */
//
// Because the DAC requires sending 3 bytes, the following constants
// use the prefix MAX1258_DACc_ for the second byte (command byte)

```

리스트 1 (5/10)

MAX1258 평가 키트/평가 시스템

```

MAX1258 EV kit Listing 1          06/01/04          6
MAX1258EV listing1

// and the prefix MAX1258_DACd_ for the third byte (data byte).
//
// A complete DAC command requires a 3-byte SPI transfer.
//
//-----
// MAX1258 DAC commands
// <C3> <C2> <C1> <C0> <D11..D0 = 0>
//
#define MAX1258_DACc_NOP          0x00 /* 0000 no operation */
//
// DAC reset commands
// 0001 0xxx xxxx xxxx // reset all input and DAC registers to 0x000
// 0001 1xxx xxxx xxxx // reset all input and DAC registers to 0xFFF
#define MAX1258_DACc_RESET_000  0x10 /* 00010xxx reset to 0x000 */
#define MAX1258_DACc_RESET_FFF  0x18 /* 00011xxx reset to 0xFFF */
//
// DAC input register write commands (not updating DAC outputs)
#define MAX1258_DACc_WRITE1     0x20 /* 0010 write input register 1 */
#define MAX1258_DACc_WRITE2     0x30 /* 0011 write input register 2 */
#define MAX1258_DACc_WRITE3     0x40 /* 0100 write input register 3 */
#define MAX1258_DACc_WRITE4     0x50 /* 0101 write input register 4 */
#define MAX1258_DACc_WRITE5     0x60 /* 0110 write input register 5 */
#define MAX1258_DACc_WRITE6     0x70 /* 0111 write input register 6 */
#define MAX1258_DACc_WRITE7     0x80 /* 1000 write input register 7 */
#define MAX1258_DACc_WRITE8     0x90 /* 1001 write input register 8 */
//
// DAC input register and DAC write-through commands (updating DAC outputs)
#define MAX1258_DACc_WRITE14LOAD 0xA0 /* 1010 write input registers 1-4 and DAC
registers 1-4 */
#define MAX1258_DACc_WRITE58LOAD 0xB0 /* 1011 write input registers 5-8 and DAC
registers 5-8 */
#define MAX1258_DACc_WRITE18LOAD 0xC0 /* 1100 write input registers 1-8 and DAC
registers 1-8 */
//
// DAC multiple input register write commands (not updating DAC outputs)
#define MAX1258_DACc_WRITE18     0xD0 /* 1101 write input registers 1-8 */
//
// DAC load commands
// 1110 xxxx xxx1 xxxx // load DAC 1 from input register 1
// 1110 xxxx xx1x xxxx // load DAC 2 from input register 2
// 1110 xxxx x1xx xxxx // load DAC 3 from input register 3
// 1110 xxxx 1xxx xxxx // load DAC 4 from input register 4
// 1110 xxx1 xxxx xxxx // load DAC 5 from input register 5
// 1110 xx1x xxxx xxxx // load DAC 6 from input register 6
// 1110 x1xx xxxx xxxx // load DAC 7 from input register 7
// 1110 1xxx xxxx xxxx // load DAC 8 from input register 8
#define MAX1258_DACc_LOAD        0xE0 /* 1110 load DAC registers from input registers,
masked... */
#define MAX1258_DACc_CH8         0x08 /* xxxx10000000xxxx DAC 8 */
#define MAX1258_DACc_CH7         0x04 /* xxxx01000000xxxx DAC 7 */
#define MAX1258_DACc_CH6         0x02 /* xxxx00100000xxxx DAC 6 */
#define MAX1258_DACc_CH5         0x01 /* xxxx00010000xxxx DAC 5 */
#define MAX1258_DACd_CH4         0x80 /* xxxx00001000xxxx DAC 4 */
#define MAX1258_DACd_CH3         0x40 /* xxxx00000100xxxx DAC 3 */
#define MAX1258_DACd_CH2         0x20 /* xxxx00000010xxxx DAC 2 */
#define MAX1258_DACd_CH1         0x10 /* xxxx00000001xxxx DAC 1 */
#define MAX1258_DACd_CH1234      0xF0 /* xxxx00001111xxxx DAC 1..4 */
#define MAX1258_DACc_CH5678      0x0F /* xxxx11110000xxxx DAC 5..8 */
//
//-----
// DAC Power-up and Power-down commands
// All of these power-up/power-down commands operate on individual DAC buffers.
//
#define MAX1258_DACc_PWR          0xF0 /* 1111 power-up or power-down DAC channels*/

```

리스트 1 (6/10)

MAX1258 평가 키트/평가 시스템

평가 대상: MAX1057/MAX1058/MAX1257/MAX1258

```
MAX1258 EV kit Listing 1          06/01/04          7
MAX1258EV listing1

//
// DAC power-up/power-down channel select mask bits:
// 1111 <dac8> <dac7> <dac6> <dac5> <dac4> <dac3> <dac2> <dac1> x x x x
// (note: 0 = no change in DAC power-on state)
//
// DAC power-up/power-down command bits:
// 1111 d d d d d d d 0 0 1 x // power up selected DAC buffers
// 1111 d d d d d d d 0 1 0 x // power off selected DAC buffers, high impedance output
// 1111 d d d d d d d 1 0 0 x // power off selected DAC buffers, 1kohm to AGND
// 1111 d d d d d d d 0 0 0 x // power off selected DAC buffers, 100kohm to AGND
// 1111 d d d d d d d 1 1 1 x // power off selected DAC buffers, 100kohm to V(REF1)
#define MAX1258_DACd_PWR_ON          0x02 /* d4d3d2d1 001x power ON selected
channels */
#define MAX1258_DACd_PWR_OFF        0x04 /* d4d3d2d1 010x power OFF, high
impedance */
#define MAX1258_DACd_PWR_OFF_1K_AGND 0x08 /* d4d3d2d1 100x power OFF, 1kohm to
AGND */
#define MAX1258_DACd_PWR_OFF_100K_AGND 0x00 /* d4d3d2d1 000x power OFF, 100kohm to
AGND */
#define MAX1258_DACd_PWR_OFF_100K_VREF 0x0F /* d4d3d2d1 111x power OFF, 100kohm to
V(REF1) */

//-----
// Enumerated type defining the meaning of each of the
// MAX1258's FIFO data slots.
typedef enum {
//
// Unused FIFO slot; meaningless data.
UNDEFINED = 0,
//
// Temperature measurement.
// The scan modes always place temperature data
// at the head of the FIFO.
TEMPERATURE,
//
// Single-ended unipolar analog inputs.
// Code 0x0000 = minimum voltage
// Code 0x0FFF = maximum voltage
// Note that AIN14 and AIN15 pins have optional alternate functions.
UNIAIN00, UNIAIN01, UNIAIN02, UNIAIN03,
UNIAIN04, UNIAIN05, UNIAIN06, UNIAIN07,
UNIAIN08, UNIAIN09, UNIAIN10, UNIAIN11,
UNIAIN12, UNIAIN13, UNIAIN14, UNIAIN15,
//
// Unipolar differential input pairs.
// Code 0x0000 = minimum voltage
// Code 0x0FFF = maximum voltage
UNIDIF0001, UNIDIF0203, UNIDIF0405, UNIDIF0607,
UNIDIF0809, UNIDIF1011, UNIDIF1213, UNIDIF1415,
//
// Bipolar differential input pairs.
// Code 0x07FF = maximum voltage
// Code 0x0000 = zero volts
// Code 0x0800 = minimum voltage
BIPDIF0001, BIPDIF0203, BIPDIF0405, BIPDIF0607,
BIPDIF0809, BIPDIF1011, BIPDIF1213, BIPDIF1415,
//
NUM_FIFO_ENTRY_TYPES
} MAX1258_fifo_entry_t;

//-----
// Enumerated type defining each pair of MAX1258 inputs
```

리스트 1 (7/10)

MAX1258 평가 키트/평가 시스템

MAX1258 EV kit Listing 1
MAX1258EV listing1

06/01/04

8

```
// as single-ended or differential
typedef enum {
    SINGLE_ENDED = 0,
    UNIPOLAR_DIFFERENTIAL,
    BIPOLAR_DIFFERENTIAL
} MAX1258_channel_config_t;

//-----
// C++ class representing the state of a MAX1258
class MAX1258
{
public:
    // MAX1258 registers cannot be read,
    // so keep track of the register values here.
    int conversion_register;
    int new_conversion_register;
    int setup_register;
    int setup_unidiff_register;
    int setup_bipdiff_register;
    int averaging_register;
    //
    int reset_register;

    int gpio_config;
    int gpio_data;
    //
    int dac_input[8];

    // The reference voltage is used to calculate the input voltage
    // represented by each measurement.
    double Vintref; // internal reference voltage
    double VpinREF1; // REF1 pin voltage
    double VpinREF2; // REF2/AIN14 pin voltage
    double VrefDAC(void); // DAC full-scale voltage depends on setup register
    double VrefADC(void); // ADC full-scale voltage depends on setup register

    int adc_code[17];

    double DAC_Voltage(int code) {
        return VrefDAC() * code / 4096;
    };

    // Constructor for class MAX1258.
    MAX1258(void);

    // Write a value to one of the part's registers.
    bool Write_Conversion(int value);
    bool Write_Setup(int value);
    bool Write_Setup_UniDiff(int value, int unidiff);
    bool Write_Setup_BipDiff(int value, int bipdiff);
    bool Write_Averaging(int value);
    bool Write_Reset(int value);
    int Read_Data(int index);
    int Read_Data_Trigger_Next_Conversion(int index);
    bool Read_Multiple_Data_Channels(int count);

    // Input configuration
    // Array InputPairConfig[] determines whether each pair
    // of input channels is configured as two single-ended inputs,
    // a unipolar differential pair, or a bipolar differential pair.
    MAX1258_channel_config_t InputPairConfig[8];
    //
    // Member function to figure out the values of InputPairConfig
    // based on the MAX1258 register values.
    // Call this function after setting setup_unidiff_register and setup_bipdiff_register
    // before using the values of InputPairConfig[].
```

리스트 1 (8/10)

MAX1258 평가 키트/평가 시스템

평가 대상: MAX1057/MAX1058/MAX1257/MAX1258

MAX1258 EV kit Listing 1
MAX1258EV listing1

06/01/04

9

```
void Update_InputPairConfig(void);

// The MAX1258 has a FIFO data buffer with
// 16 entries (plus an optional temperature measurement).
// Array FIFO_meaning[] determines what to do with
// each successive word read from the MAX1258.
MAX1258_fifo_entry_t FIFO_meaning[17];
//
// Member function to figure out the values of FIFO_meaning
// based on the MAX1258 register values.
// Call this function after setting conversion_register, setup_register, and averaging_register
// before using the values of FIFO_meaning[].
void Update_FIFO_meaning(void);

int channel (MAX1258_fifo_entry_t meaning) {
    if ((UNIAIN00 <= meaning) && (meaning <= UNIAIN15)) {
        return (meaning - UNIAIN00);
    } else
    if ((UNIDIF0001 <= meaning) && (meaning <= UNIDIF1415)) {
        return (meaning - UNIDIF0001) * 2;
    } else
    if ((BIPDIF0001 <= meaning) && (meaning <= BIPDIF1415)) {
        return (meaning - BIPDIF0001) * 2;
    } else
    if (meaning == TEMPERATURE) {
        return 16;
    } else {
        return 0;
    }
};

// General-Purpose Input/Output pin functions
//
// Configure the specified GPIO pins as outputs without changing the other pins
bool GPIO_Outputs(int pins_mask);
//
// Configure the specified GPIO pins as inputs without changing the other pins
bool GPIO_Inputs(int pins_mask);
//
// Read the specified GPIO pins
int GPIO_Read(int pins_mask);
//
// Write the specified GPIO output pins high, all other output pins low.
bool GPIO_Write(int pins_mask);
//
// Write the specified GPIO pins high without changing the other pins
bool GPIO_Set(int pins_mask);
//
// Write the specified GPIO pins low without changing the other pins
bool GPIO_Clear(int pins_mask);

// DAC Analog Outputs
//
#define MAX1258_MASK_CH1 0x10
#define MAX1258_MASK_CH2 0x20
#define MAX1258_MASK_CH3 0x40
#define MAX1258_MASK_CH4 0x80
#define MAX1258_MASK_CH5 0x01
#define MAX1258_MASK_CH6 0x02
#define MAX1258_MASK_CH7 0x04
#define MAX1258_MASK_CH8 0x08
//
// Send the DAC prefix with the no-operation command
// MAX1258_DACc_NOP
bool DAC_No_Operation(void);
```

리스트 1 (9/10)

MAX1258 평가 키트/평가 시스템

평가 대상: MAX1057/MAX1058/MAX1257/MAX1258

MAX1258 EV kit Listing 1
MAX1258EV listing1

06/01/04

10

```
//
// Reset all DAC channels to all minimum output (all 0's)
// MAX1258_DACc_RESET_000
bool DAC_Reset_All_000(void);
//
// Reset all DAC channels to all maximum output (all 1's)
// MAX1258_DACc_RESET_FFF
bool DAC_Reset_All_FFF(void);
//
// Write the specified DAC channel(s) input register, without changing the output value
// MAX1258_DACc_WRITE1 .. MAX1258_DACc_WRITE8
bool DAC_Write(int channel_number_12345678, int value);
//
// Write the specified value to DAC input registers channel 1-4 and update the outputs
// MAX1258_DACc_WRITE14LOAD
bool DAC_Write_Load_CH1234(int value);
//
// Write the specified value to DAC input registers channel 5-8 and update the outputs
// MAX1258_DACc_WRITE58LOAD
bool DAC_Write_Load_CH5678(int value);
//
// Write the specified value to DAC input registers channel 1-8 and update the outputs
// MAX1258_DACc_WRITE18LOAD
bool DAC_Write_Load_All(int value);
//
// Write the specified value to DAC input registers channel 1-8
// MAX1258_DACc_WRITE18
bool DAC_Write_All(int value);
//
// Update the specified DAC channel(s) output value from the corresponding input register, changing the output
value
// MAX1258_DACc_LOAD
bool DAC_Load_channel(int channel_number_12345678);
bool DAC_Load_channels(int channel_mask);
//
// Power-on the specified DAC channel(s) without changing the others
// MAX1258_DACd_PWR_ON
bool DAC_PowerOn_channels(int channel_mask);
//
// Power-off the specified DAC channel(s) high-impedance without changing the others
// MAX1258_DACd_PWR_OFF
bool DAC_PowerOff_HiZ_channels(int channel_mask);
//
// Power-off the specified DAC channel(s) VREF-100kohm without changing the others
// MAX1258_DACd_PWR_OFF_100K_VREF
bool DAC_PowerOff_Vref100k_channels(int channel_mask);
//
// Power-off the specified DAC channel(s) GND-100kohm without changing the others
// MAX1258_DACd_PWR_OFF_100K_AGND
bool DAC_PowerOff_Gnd100k_channels(int channel_mask);
//
// Power-off the specified DAC channel(s) GND-1kohm without changing the others
// MAX1258_DACd_PWR_OFF_1K_AGND
bool DAC_PowerOff_Gnd1k_channels(int channel_mask);

};

//-----
#endif // DRV1258H
```

리스트 1 (10/10)

MAX1258 평가 키트/평가 시스템

평가 대상: MAX1057/MAX1058/MAX1257/MAX1258

```
MAX1258 EV kit Listing 2          06/01/04          1
MAX1258EV listing2

// Drv1258.cpp
// MAX1258-specific driver.
// mku 04/07/2004
// (C) 2004 Maxim Integrated Products
// For use with Borland C++ Builder 3.0
//-----
// Revision history:
//=====
// __/__/2004: Initial Release as MAX1258 Version 1.0
//=====
//-----

#include "Drv1258.h"

//-----
// To indicate failure using return value instead of C++ exception,
// define the preprocessor symbol THROW_EXCEPTIONS=0.
//
#ifdef THROW_EXCEPTIONS
#define THROW_EXCEPTIONS 1
#endif
//
// Functions that return a data value must indicate failure by either
// throwing a C++ exception, or by returning a value that could never happen.
#ifdef THROW_EXCEPTIONS
#else
#define MAX1258_IMPOSSIBLE_DATA_VALUE 32000
#endif
//-----
MAX1258::MAX1258(void)
{
    conversion_register = MAX1258_CONV_POR;
    setup_register = MAX1258_SETUP_POR;
    setup_unidiff_register = MAX1258_SETUP_UNIDIF_POR;
    setup_bipdiff_register = MAX1258_SETUP_BIPDIF_POR;
    averaging_register = MAX1258_AVERAGE_POR;
    //~ Vref = 2.500;
    Vintref = 4.096;
    VpinREF1 = 0;
    VpinREF2 = 0;
    for (int index = 0; index < 8; index++) {
        InputPairConfig[index] = SINGLE_ENDED;
    }
    for (int index = 0; index < 17; index++) {
        FIFO_meaning[index] = UNDEFINED;
    }
}
//-----
bool MAX1258::Write_Conversion(int value)
{
    // NOTE: the act of writing to the conversion register
    // has the effect of triggering one or more conversions
    // if the clock mode is 10 or 11.

    value = value &~ 0x00; //xxxx xxxx
    value = value | 0x80; //1xxx xxxx
                        //1xxx xxxx

    const unsigned __int8 mosi[] = {
        (unsigned __int8) (value)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        conversion_register = value;
    }
}
```

리스트 2 (1/14)

MAX1258 평가 키트/평가 시스템

MAX1258 EV kit Listing 2
MAX1258EV listing2

06/01/04

2

```

    return result;
}
//-----
bool MAX1258::Write_Setup(int value)
{
    value = value &~ 0x83; //0xxx xx00
    value = value | 0x40; //x1xx xxxx
                    //01xx xx00

    const unsigned __int8 mosi[] = {
        (unsigned __int8) (value)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        setup_register = value;
        Update_FIFO_meaning();
    }
    return result;
}
//-----
double MAX1258::VrefDAC(void)
{
    switch (setup_register & MAX1258_SETUP_REF1) {
    case MAX1258_SETUP_REF0:
        return Vintref;
    case MAX1258_SETUP_REF01:
        return VpinREF1;
    case MAX1258_SETUP_REF10:
        return VpinREF1;
    case MAX1258_SETUP_REF11:
        return VpinREF1;
    }
    return Vintref;
}
//-----
double MAX1258::VrefADC(void)
{
    switch (setup_register & MAX1258_SETUP_REF1) {
    case MAX1258_SETUP_REF0:
        return Vintref;
    case MAX1258_SETUP_REF01:
        return VpinREF2;
    case MAX1258_SETUP_REF10:
        return Vintref;
    case MAX1258_SETUP_REF11:
        return VpinREF1-VpinREF2;
    }
    return Vintref;
}
//-----
bool MAX1258::Write_Setup_UniDiff(int value, int unidiff)
{
    value = value &~ 0x81; //0xxx xxx0
    value = value | 0x42; //x1xx xx1x
                    //01xx xx10 unidiff

    //01xx xx10
    const unsigned __int8 mosi[] = {
        (unsigned __int8) (value),
        (unsigned __int8) (unidiff)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        setup_register = value;
        setup_unidiff_register = unidiff;
        Update_InputPairConfig();
        Update_FIFO_meaning();
    }
}

```

리스트 2 (2/14)

MAX1258 평가 킷/평가 시스템

평가 대상: MAX1057/MAX1058/MAX1257/MAX1258

MAX1258 EV kit Listing 2
MAX1258EV listing2

06/01/04

3

```
    return result;
}
//-----
bool MAX1258::Write_Setup_BipDiff(int value, int bipdiff)
{
    value = value &~ 0x80; //0xxx xxxx
    value = value | 0x43; //x1xx xx11
                        //01xx xx11 bipdiff

    //01xx xx10
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(value),
        (unsigned __int8)(bipdiff)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        setup_register = value;
        setup_bipdiff_register = bipdiff;
        Update_InputPairConfig();
        Update_FIFO_meaning();
    }
    return result;
}
//-----
bool MAX1258::Write_Averaging(int value)
{
    value = value &~ 0xC0; //00xx xxxx
    value = value | 0x20; //xx1x xxxx
                        //001x xxxx

    const unsigned __int8 mosi[] = {
        (unsigned __int8)(value)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        averaging_register = value;
        Update_FIFO_meaning();
    }
    return result;
}
//-----
bool MAX1258::Write_Reset(int value)
{
    value = value &~ 0xF0; //0000 xxxx
    value = value | 0x08; //xxxx 1xxx
                        //0000 1xxx

    const unsigned __int8 mosi[] = {
        (unsigned __int8)(value)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
    }
    return result;
}
//-----
int MAX1258::Read_Data(int index)
{
    if ((index < 0) || (index >= 17)) {
        #if THROW_EXCEPTIONS
            throw "illegal channel index in MAX1258::Read_Data";
        #else
            return MAX1254_IMPOSSIBLE_DATA_VALUE;
        #endif
    }
    const unsigned __int8 mosi[] = {
        (unsigned __int8)(0),

```

리스트 2 (3/14)

MAX1258 평가 키트/평가 시스템

MAX1258 EV kit Listing 2
MAX1258EV listing2

06/01/04

4

```

        (unsigned __int8) (0)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        int data16 = (miso_buf[0] * 0x100) + miso_buf[1];
        adc_code[index] = data16;
        return data16;
    }
#ifdef THROW_EXCEPTIONS
    throw "SPI_Transfer failed in MAX1258::Read_Data";
#else
    return MAX1258_IMPOSSIBLE_DATA_VALUE;
#endif
}
//-----
int MAX1258::Read_Data_Trigger_Next_Conversion(int index)
{
    if ((index < 0) || (index >= 17)) {
#ifdef THROW_EXCEPTIONS
        throw "illegal channel index in MAX1258::Read_Data_Trigger_Next_Conversion";
#else
        return MAX1254_IMPOSSIBLE_DATA_VALUE;
#endif
    }
    const unsigned __int8 mosi[] = {
        (unsigned __int8) (0),
        (unsigned __int8) (conversion_register)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        int data16 = (miso_buf[0] * 0x100) + miso_buf[1];
        adc_code[index] = data16;
        return data16;
    }
#ifdef THROW_EXCEPTIONS
    throw "SPI_Transfer failed in MAX1258::Read_Data_Trigger_Next_Conversion";
#else
    return MAX1258_IMPOSSIBLE_DATA_VALUE;
#endif
}
//-----
bool MAX1258::Read_Multiple_Data_Channels(int count)
{
    switch(setup_register & 0x30)
    {
    case 0x00: // 0100xxxx pin16=CNVST, Int clock, Triggered by CNVST pulse
        // Clock mode 00:
        // Pulse CNVST pin
        // Wait for EOC low
        // issue command "R" to read each 16-bit value from the FIFO
        //
        // Update configuration register value only if necessary.
        if (new_conversion_register != conversion_register) {
            Write_Conversion(new_conversion_register);
        }
        // Trigger conversion by pulsing the CNVST pin.
        Pulse_MAX1258_CONV();
        if (Wait_MAX1258_EOC_Low()) {
            for (int index = 0; index < count; index++) {
                Read_Data(index);
            }
        }
        return true;
    case 0x10: // 0101xxxx pin16=CNVST, Int clock, Triggered by CNVST pulses, custom Tacq
        // Clock mode 01:
        // For each requested channel,

```

리스트 2 (4/14)

MAX1258 평가 키트/평가 시스템

평가 대상: MAX1057/MAX1058/MAX1257/MAX1258

MAX1258 EV kit Listing 2
MAX1258EV listing2

06/01/04

5

```
// Drive CNVST pin low
// Delay for the acquisition time (run this delay on 68HC16?)
// Drive CNVST pin high
// Wait for EOC low
// issue command "R" to read each 16-bit value from the FIFO
//
// Update configuration register value only if necessary.
if (new_conversion_register != conversion_register) {
    Write_Conversion(new_conversion_register);
}
// Trigger conversion by pulsing the CNVST pin for each channel.
for (int index = 0; index < 17; index++) {
    if (FIFO_meaning[index] != UNDEFINED)
    {
        Pulse_MAX1258_CONV(); // TODO: pulse width sets acquisition time for each channel
        if (Wait_MAX1258_EOC_Low()) {
            Read_Data(index);
        }
    }
}
return true;
case 0x20: // 0110xxxx pin16=AIN15, Int clock, Triggered by conversion register write
{
    // Clock mode 10:
    // Write to the conversion register 1xxx xxxx using command "Wxx"
    // Wait for EOC low
    // issue command "R" to read each 16-bit value from the FIFO
    //
    // Trigger conversion by writing the conversion register.
    Write_Conversion(new_conversion_register);
    if (Wait_MAX1258_EOC_Low()) {
        for (int index = 0; index < count; index++) {
            Read_Data(index);
        }
    }
}
return true;
case 0x30: // 0111xxxx pin16=AIN15, Ext clock, Triggered by conversion register write
// Clock mode 11:
// Write to the conversion register 1xxx xxxx using command "Wxx"
// issue command "R" to read 16-bit value
//
// Trigger conversion by writing the conversion register.
Write_Conversion(new_conversion_register);
// Clock mode 11 does not generate an EOC pulse.
Read_Data_Trigger_Next_Conversion(0);
return true;
}
return false; // TODO: this code was optimized into machine language file KIT1258.ASM
// to increase data throughput. Need to convert back to portable C code,
// and probably #ifdef it back to the optimized 68HC16-specific program.
}
//-----
void MAX1258::Update_InputPairConfig(void)
{
    for (int index = 0; index < 8; index++) {
        InputPairConfig[index] = SINGLE_ENDED;
    }

    if (setup_bipdiff_register & MAX1258_SETUP_BIPDIF0001)
        InputPairConfig[0] = BIPOLAR_DIFFERENTIAL;
    if (setup_bipdiff_register & MAX1258_SETUP_BIPDIF0203)
        InputPairConfig[1] = BIPOLAR_DIFFERENTIAL;
    if (setup_bipdiff_register & MAX1258_SETUP_BIPDIF0405)
        InputPairConfig[2] = BIPOLAR_DIFFERENTIAL;
    if (setup_bipdiff_register & MAX1258_SETUP_BIPDIF0607)
```

리스트 2 (5/14)

MAX1258 평가 키트/평가 시스템

MAX1258 EV kit Listing 2
MAX1258EV listing2

06/01/04

6

```

    InputPairConfig[3] = BIPOLAR_DIFFERENTIAL;
    if (setup_bipdiff_register & MAX1258_SETUP_BIPDIF0809)
        InputPairConfig[4] = BIPOLAR_DIFFERENTIAL;
    if (setup_bipdiff_register & MAX1258_SETUP_BIPDIF1011)
        InputPairConfig[5] = BIPOLAR_DIFFERENTIAL;
    if (setup_bipdiff_register & MAX1258_SETUP_BIPDIF1213)
        InputPairConfig[6] = BIPOLAR_DIFFERENTIAL;
    if (setup_bipdiff_register & MAX1258_SETUP_BIPDIF1415)
        InputPairConfig[7] = BIPOLAR_DIFFERENTIAL;

    if (setup_unidiff_register & MAX1258_SETUP_UNIDIF0001)
        InputPairConfig[0] = UNIPOLAR_DIFFERENTIAL;
    if (setup_unidiff_register & MAX1258_SETUP_UNIDIF0203)
        InputPairConfig[1] = UNIPOLAR_DIFFERENTIAL;
    if (setup_unidiff_register & MAX1258_SETUP_UNIDIF0405)
        InputPairConfig[2] = UNIPOLAR_DIFFERENTIAL;
    if (setup_unidiff_register & MAX1258_SETUP_UNIDIF0607)
        InputPairConfig[3] = UNIPOLAR_DIFFERENTIAL;
    if (setup_unidiff_register & MAX1258_SETUP_UNIDIF0809)
        InputPairConfig[4] = UNIPOLAR_DIFFERENTIAL;
    if (setup_unidiff_register & MAX1258_SETUP_UNIDIF1011)
        InputPairConfig[5] = UNIPOLAR_DIFFERENTIAL;
    if (setup_unidiff_register & MAX1258_SETUP_UNIDIF1213)
        InputPairConfig[6] = UNIPOLAR_DIFFERENTIAL;
    if (setup_unidiff_register & MAX1258_SETUP_UNIDIF1415)
        InputPairConfig[7] = UNIPOLAR_DIFFERENTIAL;
}
//-----
void MAX1258::Update_FIFO_meaning(void)
{
    int conversion_register = new_conversion_register;

    for (int index = 0; index < 17; index++) {
        FIFO_meaning[index] = UNDEFINED;
    }

    int channel_field = (conversion_register >> 3) & 0x0F;
    int channel_index = channel_field;
    int repeat_count = 4;
    switch(averaging_register & 0xE3) {
    case MAX1258_REPEAT_4: // 001xxx00 4 times
        repeat_count = 4;
        break;
    case MAX1258_REPEAT_8: // 001xxx01 8 times
        repeat_count = 8;
        break;
    case MAX1258_REPEAT_12: // 001xxx10 12 times
        repeat_count = 12;
        break;
    case MAX1258_REPEAT_16: // 001xxx11 16 times
        repeat_count = 16;
        break;
    }

    MAX1258_fifo_entry_t meaning =
        (MAX1258_fifo_entry_t)(UNIAIN00 + channel_field);
    if (InputPairConfig[channel_field / 2] == BIPOLAR_DIFFERENTIAL) {
        meaning = (MAX1258_fifo_entry_t)(BIPDIF0001 + channel_field/2);
    } else if (InputPairConfig[channel_field / 2] == UNIPOLAR_DIFFERENTIAL) {
        meaning = (MAX1258_fifo_entry_t)(UNIDIF0001 + channel_field/2);
    }
    int index = 0;
    switch(conversion_register & MAX1258_ACTION_MASK) {
    case MAX1258_CONV_SCAN_00_N:
        meaning = UNIAIN00;
        channel_index = 0;
        while(channel_index <= channel_field) {
            int pair_index = channel_index / 2;
            if (InputPairConfig[pair_index] == BIPOLAR_DIFFERENTIAL) {

```

리스트 2 (6/14)

MAX1258 EV kit Listing 2
MAX1258EV listing2

06/01/04

7

```

        meaning = (MAX1258_fifo_entry_t)(BIPDIF0001 + pair_index);
        FIFO_meaning[index++] = meaning;
        channel_index = channel_index + 2;
    } else if (InputPairConfig[pair_index] == UNIPOLAR_DIFFERENTIAL) {
        meaning = (MAX1258_fifo_entry_t)(UNIDIF0001 + pair_index);
        FIFO_meaning[index++] = meaning;
        channel_index = channel_index + 2;
    } else {
        meaning = (MAX1258_fifo_entry_t)(UNIAIN00 + channel_index);
        FIFO_meaning[index++] = meaning;
        channel_index = channel_index + 1;
    }
}
break;
case MAX1258_CONV_SCAN_T_00_N:
    FIFO_meaning[index++] = TEMPERATURE;
    meaning = UNIAIN00;
    channel_index = 0;
    while(channel_index <= channel_field) {
        int pair_index = channel_index / 2;
        if (InputPairConfig[pair_index] == BIPOLAR_DIFFERENTIAL) {
            meaning = (MAX1258_fifo_entry_t)(BIPDIF0001 + pair_index);
            FIFO_meaning[index++] = meaning;
            channel_index = channel_index + 2;
        } else if (InputPairConfig[pair_index] == UNIPOLAR_DIFFERENTIAL) {
            meaning = (MAX1258_fifo_entry_t)(UNIDIF0001 + pair_index);
            FIFO_meaning[index++] = meaning;
            channel_index = channel_index + 2;
        } else {
            meaning = (MAX1258_fifo_entry_t)(UNIAIN00 + channel_index);
            FIFO_meaning[index++] = meaning;
            channel_index = channel_index + 1;
        }
    }
}
break;
case MAX1258_CONV_SCAN_N_15:
    while (index < 17) {
        int pair_index = channel_index / 2;
        if (InputPairConfig[pair_index] == BIPOLAR_DIFFERENTIAL) {
            meaning = (MAX1258_fifo_entry_t)(BIPDIF0001 + pair_index);
            FIFO_meaning[index++] = meaning;
            channel_index = channel_index + 2;
        } else if (InputPairConfig[pair_index] == UNIPOLAR_DIFFERENTIAL) {
            meaning = (MAX1258_fifo_entry_t)(UNIDIF0001 + pair_index);
            FIFO_meaning[index++] = meaning;
            channel_index = channel_index + 2;
        } else {
            meaning = (MAX1258_fifo_entry_t)(UNIAIN00 + channel_index);
            FIFO_meaning[index++] = meaning;
            channel_index = channel_index + 1;
        }
    }
    if (meaning == UNIAIN15) break;
    if (meaning == UNIDIF1415) break;
    if (meaning == BIPDIF1415) break;
}
break;
case MAX1258_CONV_SCAN_T_N_15:
    FIFO_meaning[index++] = TEMPERATURE;
    while (index < 17) {
        int pair_index = channel_index / 2;
        if (InputPairConfig[pair_index] == BIPOLAR_DIFFERENTIAL) {
            meaning = (MAX1258_fifo_entry_t)(BIPDIF0001 + pair_index);
            FIFO_meaning[index++] = meaning;
            channel_index = channel_index + 2;
        } else if (InputPairConfig[pair_index] == UNIPOLAR_DIFFERENTIAL) {
            meaning = (MAX1258_fifo_entry_t)(UNIDIF0001 + pair_index);
            FIFO_meaning[index++] = meaning;
            channel_index = channel_index + 2;
        } else {
            meaning = (MAX1258_fifo_entry_t)(UNIAIN00 + channel_index);
            FIFO_meaning[index++] = meaning;
            channel_index = channel_index + 1;
        }
    }
}

```

리스트 2 (7/14)

MAX1258 평가 키트/평가 시스템

MAX1258 EV kit Listing 2
MAX1258EV listing2

06/01/04

8

```

    }
    if (meaning == UNIAIN15) break;
    if (meaning == UNIDIF1415) break;
    if (meaning == BIPDIF1415) break;
  }
  break;
case MAX1258_CONV_SINGLE_REPEAT:
  while (index < repeat_count) {
    FIFO_meaning[index++] = meaning;
  }
  break;
case MAX1258_CONV_SINGLE_READ:
  FIFO_meaning[index++] = meaning;
  break;
}

/* Check for setups where AIN14-AIN15 are used for an alternate function */
if ((setup_register & 0xE0) == MAX1258_SETUP_INTCLK_CNVT) {
  /* 0100xxxx AIN15 alternate function as CNVST input */
  /* 0101xxxx AIN15 alternate function as CNVST input */
  for (int index = 0; index < 17; index++) {
    meaning = FIFO_meaning[index];
    if (meaning == UNIAIN15)
      FIFO_meaning[index] = UNDEFINED;
    if (meaning == UNIDIF1415)
      FIFO_meaning[index] = UNDEFINED;
    if (meaning == BIPDIF1415)
      FIFO_meaning[index] = UNDEFINED;
  }
}

// MAX1258 AIN14 alternate pin function in mode 01xx01xx */
#if 1
switch (setup_register & 0xCC) {
case MAX1258_SETUP_EXTREF:
case MAX1258_SETUP_EXTREF_DIFF:
  /* AIN14 alternate function as REF2 input */
  for (int index = 0; index < 17; index++) {
    meaning = FIFO_meaning[index];
    if (meaning == UNIAIN14)
      FIFO_meaning[index] = UNDEFINED;
    if (meaning == UNIDIF1415)
      FIFO_meaning[index] = UNDEFINED;
    if (meaning == BIPDIF1415)
      FIFO_meaning[index] = UNDEFINED;
  }
  break;
default:
  break;
}
#else
if ((setup_register & 0xCC) == MAX1258_SETUP_EXTREF_DIFF) {
  /* MAX1231: 01xx11xx AIN14 alternate function as REF- input */
  for (int index = 0; index < 17; index++) {
    meaning = FIFO_meaning[index];
    if (meaning == UNIAIN14)
      FIFO_meaning[index] = UNDEFINED;
    if (meaning == UNIDIF1415)
      FIFO_meaning[index] = UNDEFINED;
    if (meaning == BIPDIF1415)
      FIFO_meaning[index] = UNDEFINED;
  }
}
#endif
}
//-----
bool MAX1258::GPIO_Outputs(int pins_mask)
{
  gpio_config = gpio_config | pins_mask;
}

```

리스트 2 (8/14)

MAX1258 평가 키트/평가 시스템

평가 대상: MAX1057/MAX1058/MAX1257/MAX1258

MAX1258 EV kit Listing 2
MAX1258EV listing2

06/01/04

9

```
const unsigned __int8 mosi[] = {
    (unsigned __int8) (MAX1258_GPIO_CONFIG),
    (unsigned __int8) (gpio_config / 0x100),
    (unsigned __int8) (gpio_config & 0xFF)
};
unsigned __int8 miso_buf[sizeof(mosi)];
bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
if (result) {
    return true; // success
} // else operation failed
return false;
}
//-----
bool MAX1258::GPIO_Inputs(int pins_mask)
{
    //~ To configure a pin for input requires writing BOTH
    //~ a MAX1258_GPIO_CONFIG with bit = 0 and also
    //~ a MAX1258_GPIO_WRITE with bit = 1.
    //~ Writing CONFIG = 0 and WRITE = 0 will configure the pin for "open-drain pull-down" mode. Not input.

    gpio_config = gpio_config &~ pins_mask;
    const unsigned __int8 mosi[] = {
        (unsigned __int8) (MAX1258_GPIO_CONFIG),
        (unsigned __int8) (gpio_config / 0x100),
        (unsigned __int8) (gpio_config & 0xFF)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        return GPIO_Write(gpio_data | pins_mask);
    } // else operation failed
    return false;
}
//-----
int MAX1258::GPIO_Read(int pins_mask)
{
    const unsigned __int8 mosi[] = {
        (unsigned __int8) (MAX1258_GPIO_READ),
        (unsigned __int8) (0),
        (unsigned __int8) (0)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        unsigned __int16 pins = miso_buf[1] * 0x100 + miso_buf[2];
        return (pins & pins_mask);
    } // else operation failed
    return 0;
}
//-----
bool MAX1258::GPIO_Write(int pins_mask)
{
    gpio_data = pins_mask;
    const unsigned __int8 mosi[] = {
        (unsigned __int8) (MAX1258_GPIO_WRITE),
        (unsigned __int8) (gpio_data / 0x100),
        (unsigned __int8) (gpio_data & 0xFF)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        return true; // success
    } // else operation failed
    return false;
}
//-----
bool MAX1258::GPIO_Set(int pins_mask)
```

리스트 2 (9/14)

MAX1258 평가 키트/평가 시스템

MAX1258 EV kit Listing 2
MAX1258EV listing2

06/01/04

10

```

{
    return GPIO_Write(gpio_data | pins_mask);
}
//-----
bool MAX1258::GPIO_Clear(int pins_mask)
{
    return GPIO_Write(gpio_data &~ pins_mask);
}
//-----
bool MAX1258::DAC_No_Operation(void)
{
    // Send the DAC prefix with the no-operation command
    unsigned __int8 mosi[] = {
        (unsigned __int8) (MAX1258_DAC),
        (unsigned __int8) (MAX1258_DACc_NOP),
        (unsigned __int8) (0)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//-----
bool MAX1258::DAC_Reset_All_000(void)
{
    // Reset all DAC channels to all minimum output (all 0's)
    unsigned __int8 mosi[] = {
        (unsigned __int8) (MAX1258_DAC),
        (unsigned __int8) (MAX1258_DACc_RESET_000),
        (unsigned __int8) (0)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//-----
bool MAX1258::DAC_Reset_All_FFF(void)
{
    // Reset all DAC channels to all maximum output (all 1's)
    unsigned __int8 mosi[] = {
        (unsigned __int8) (MAX1258_DAC),
        (unsigned __int8) (MAX1258_DACc_RESET_FFF),
        (unsigned __int8) (0)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//-----
bool MAX1258::DAC_Write(int channel_number_12345678, int DAC_value)
{
    // Write the specified DAC channel(s) input register, without changing the output value
    unsigned __int8 mosi[] = {
        (unsigned __int8) (MAX1258_DAC),
        (unsigned __int8) (DAC_value >> 8 & 0x0F),
        (unsigned __int8) (DAC_value & 0xFF)
    };
};

```

리스트 2 (10/14)

MAX1258 평가 키트/평가 시스템

평가 대상: MAX1057/MAX1058/MAX1257/MAX1258

MAX1258 EV kit Listing 2
MAX1258EV listing2

06/01/04

11

```
switch (channel_number_12345678) {
case 1: mosi[1] |= MAX1258_DACc_WRITE1; break;
case 2: mosi[1] |= MAX1258_DACc_WRITE2; break;
case 3: mosi[1] |= MAX1258_DACc_WRITE3; break;
case 4: mosi[1] |= MAX1258_DACc_WRITE4; break;
case 5: mosi[1] |= MAX1258_DACc_WRITE5; break;
case 6: mosi[1] |= MAX1258_DACc_WRITE6; break;
case 7: mosi[1] |= MAX1258_DACc_WRITE7; break;
case 8: mosi[1] |= MAX1258_DACc_WRITE8; break;
default:
return false; // invalid channel mask
}
unsigned __int8 miso_buf[sizeof(mosi)];
bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
if (result) {
// success
return true;
} // else operation failed
return false;
}
//-----
bool MAX1258::DAC_Load_channel(int channel_number_12345678)
{
// Update the specified DAC channel(s) output value from the corresponding input register, changing the output
value
unsigned __int8 mosi[] = {
(unsigned __int8)(MAX1258_DAC),
(unsigned __int8)(MAX1258_DACc_LOAD),
(unsigned __int8)(0)
};
switch (channel_number_12345678) {
case 1: mosi[2] |= MAX1258_DACd_CH1; break;
case 2: mosi[2] |= MAX1258_DACd_CH2; break;
case 3: mosi[2] |= MAX1258_DACd_CH3; break;
case 4: mosi[2] |= MAX1258_DACd_CH4; break;
case 5: mosi[1] |= MAX1258_DACc_CH5; break;
case 6: mosi[1] |= MAX1258_DACc_CH6; break;
case 7: mosi[1] |= MAX1258_DACc_CH7; break;
case 8: mosi[1] |= MAX1258_DACc_CH8; break;
default:
return false; // invalid channel mask
}
unsigned __int8 miso_buf[sizeof(mosi)];
bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
if (result) {
// success
return true;
} // else operation failed
return false;
}
//-----
bool MAX1258::DAC_Load_channels(int channel_mask)
{
// Update the specified DAC channel(s) output value from the corresponding input register, changing the output
value
unsigned __int8 mosi[] = {
(unsigned __int8)(MAX1258_DAC),
(unsigned __int8)(MAX1258_DACc_LOAD),
(unsigned __int8)(0)
};
if (channel_mask & MAX1258_MASK_CH1) mosi[2] |= MAX1258_DACd_CH1;
if (channel_mask & MAX1258_MASK_CH2) mosi[2] |= MAX1258_DACd_CH2;
if (channel_mask & MAX1258_MASK_CH3) mosi[2] |= MAX1258_DACd_CH3;
if (channel_mask & MAX1258_MASK_CH4) mosi[2] |= MAX1258_DACd_CH4;
if (channel_mask & MAX1258_MASK_CH5) mosi[1] |= MAX1258_DACc_CH5;
if (channel_mask & MAX1258_MASK_CH6) mosi[1] |= MAX1258_DACc_CH6;
if (channel_mask & MAX1258_MASK_CH7) mosi[1] |= MAX1258_DACc_CH7;
if (channel_mask & MAX1258_MASK_CH8) mosi[1] |= MAX1258_DACc_CH8;
unsigned __int8 miso_buf[sizeof(mosi)];
}
```

리스트 2 (11/14)

MAX1258 평가 키트/평가 시스템

MAX1258 EV kit Listing 2
MAX1258EV listing2

06/01/04

12

```

    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//-----
bool MAX1258::DAC_Write_Load_CH1234(int value)
{
    unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_DAC),
        (unsigned __int8)((value >> 8 & 0x0F) | MAX1258_DACc_WRITE14LOAD),
        (unsigned __int8)(value & 0xFF)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//-----
bool MAX1258::DAC_Write_Load_CH5678(int value)
{
    unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_DAC),
        (unsigned __int8)((value >> 8 & 0x0F) | MAX1258_DACc_WRITE58LOAD),
        (unsigned __int8)(value & 0xFF)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//-----
bool MAX1258::DAC_Write_Load_All(int value)
{
    unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_DAC),
        (unsigned __int8)((value >> 8 & 0x0F) | MAX1258_DACc_WRITE18LOAD),
        (unsigned __int8)(value & 0xFF)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//-----
bool MAX1258::DAC_Write_All(int value)
{
    unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_DAC),
        (unsigned __int8)((value >> 8 & 0x0F) | MAX1258_DACc_WRITE18),
        (unsigned __int8)(value & 0xFF)
    };
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    }
}

```

리스트 2 (12/14)

MAX1258 평가 키트/평가 시스템

평가 대상: MAX1057/MAX1058/MAX1257/MAX1258

MAX1258 EV kit Listing 2
MAX1258EV listing2

06/01/04

13

```
    } // else operation failed
    return false;
}
//-----
bool MAX1258::DAC_PowerOn_channels(int channel_mask)
{
    // Power-on the specified DAC channel(s) without changing the others
    unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_DAC),
        (unsigned __int8)(MAX1258_DACc_PWR),
        (unsigned __int8)(MAX1258_DACd_PWR_ON)
    };
    if (channel_mask & MAX1258_MASK_CH1) mosi[2] |= MAX1258_DACd_CH1;
    if (channel_mask & MAX1258_MASK_CH2) mosi[2] |= MAX1258_DACd_CH2;
    if (channel_mask & MAX1258_MASK_CH3) mosi[2] |= MAX1258_DACd_CH3;
    if (channel_mask & MAX1258_MASK_CH4) mosi[2] |= MAX1258_DACd_CH4;
    if (channel_mask & MAX1258_MASK_CH5) mosi[1] |= MAX1258_DACc_CH5;
    if (channel_mask & MAX1258_MASK_CH6) mosi[1] |= MAX1258_DACc_CH6;
    if (channel_mask & MAX1258_MASK_CH7) mosi[1] |= MAX1258_DACc_CH7;
    if (channel_mask & MAX1258_MASK_CH8) mosi[1] |= MAX1258_DACc_CH8;
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//-----
bool MAX1258::DAC_PowerOff_HiZ_channels(int channel_mask)
{
    // Power-off the specified DAC channel(s) high-impedance without changing the others
    unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_DAC),
        (unsigned __int8)(MAX1258_DACc_PWR),
        (unsigned __int8)(MAX1258_DACd_PWR_OFF)
    };
    if (channel_mask & MAX1258_MASK_CH1) mosi[2] |= MAX1258_DACd_CH1;
    if (channel_mask & MAX1258_MASK_CH2) mosi[2] |= MAX1258_DACd_CH2;
    if (channel_mask & MAX1258_MASK_CH3) mosi[2] |= MAX1258_DACd_CH3;
    if (channel_mask & MAX1258_MASK_CH4) mosi[2] |= MAX1258_DACd_CH4;
    if (channel_mask & MAX1258_MASK_CH5) mosi[1] |= MAX1258_DACc_CH5;
    if (channel_mask & MAX1258_MASK_CH6) mosi[1] |= MAX1258_DACc_CH6;
    if (channel_mask & MAX1258_MASK_CH7) mosi[1] |= MAX1258_DACc_CH7;
    if (channel_mask & MAX1258_MASK_CH8) mosi[1] |= MAX1258_DACc_CH8;
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//-----
bool MAX1258::DAC_PowerOff_Vref100k_channels(int channel_mask)
{
    // Power-off the specified DAC channel(s) VREF-100kohm without changing the others
    unsigned __int8 mosi[] = {
        (unsigned __int8)(MAX1258_DAC),
        (unsigned __int8)(MAX1258_DACc_PWR),
        (unsigned __int8)(MAX1258_DACd_PWR_OFF_100K_VREF)
    };
    if (channel_mask & MAX1258_MASK_CH1) mosi[2] |= MAX1258_DACd_CH1;
    if (channel_mask & MAX1258_MASK_CH2) mosi[2] |= MAX1258_DACd_CH2;
    if (channel_mask & MAX1258_MASK_CH3) mosi[2] |= MAX1258_DACd_CH3;
    if (channel_mask & MAX1258_MASK_CH4) mosi[2] |= MAX1258_DACd_CH4;
    if (channel_mask & MAX1258_MASK_CH5) mosi[1] |= MAX1258_DACc_CH5;
    if (channel_mask & MAX1258_MASK_CH6) mosi[1] |= MAX1258_DACc_CH6;
    if (channel_mask & MAX1258_MASK_CH7) mosi[1] |= MAX1258_DACc_CH7;
```

리스트 2 (13/14)

MAX1258 평가 킷/평가 시스템

MAX1258 EV kit Listing 2
MAX1258EV listing2

06/01/04

14

```

if (channel_mask & MAX1258_MASK_CH8) mosi[1] |= MAX1258_DACc_CH8;
unsigned __int8 miso_buf[sizeof(mosi)];
bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
if (result) {
    // success
    return true;
} // else operation failed
return false;
}
//-----
bool MAX1258::DAC_PowerOff_Gnd100k_channels(int channel_mask)
{
    // Power-off the specified DAC channel(s) GND-100kohm without changing the others
    unsigned __int8 mosi[] = {
        (unsigned __int8) (MAX1258_DAC),
        (unsigned __int8) (MAX1258_DACc_PWR),
        (unsigned __int8) (MAX1258_DACd_PWR_OFF_100K_AGND)
    };
    if (channel_mask & MAX1258_MASK_CH1) mosi[2] |= MAX1258_DACd_CH1;
    if (channel_mask & MAX1258_MASK_CH2) mosi[2] |= MAX1258_DACd_CH2;
    if (channel_mask & MAX1258_MASK_CH3) mosi[2] |= MAX1258_DACd_CH3;
    if (channel_mask & MAX1258_MASK_CH4) mosi[2] |= MAX1258_DACd_CH4;
    if (channel_mask & MAX1258_MASK_CH5) mosi[1] |= MAX1258_DACc_CH5;
    if (channel_mask & MAX1258_MASK_CH6) mosi[1] |= MAX1258_DACc_CH6;
    if (channel_mask & MAX1258_MASK_CH7) mosi[1] |= MAX1258_DACc_CH7;
    if (channel_mask & MAX1258_MASK_CH8) mosi[1] |= MAX1258_DACc_CH8;
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//-----
bool MAX1258::DAC_PowerOff_Gnd1k_channels(int channel_mask)
{
    // Power-off the specified DAC channel(s) GND-1kohm without changing the others
    unsigned __int8 mosi[] = {
        (unsigned __int8) (MAX1258_DAC),
        (unsigned __int8) (MAX1258_DACc_PWR),
        (unsigned __int8) (MAX1258_DACd_PWR_OFF_1K_AGND)
    };
    if (channel_mask & MAX1258_MASK_CH1) mosi[2] |= MAX1258_DACd_CH1;
    if (channel_mask & MAX1258_MASK_CH2) mosi[2] |= MAX1258_DACd_CH2;
    if (channel_mask & MAX1258_MASK_CH3) mosi[2] |= MAX1258_DACd_CH3;
    if (channel_mask & MAX1258_MASK_CH4) mosi[2] |= MAX1258_DACd_CH4;
    if (channel_mask & MAX1258_MASK_CH5) mosi[1] |= MAX1258_DACc_CH5;
    if (channel_mask & MAX1258_MASK_CH6) mosi[1] |= MAX1258_DACc_CH6;
    if (channel_mask & MAX1258_MASK_CH7) mosi[1] |= MAX1258_DACc_CH7;
    if (channel_mask & MAX1258_MASK_CH8) mosi[1] |= MAX1258_DACc_CH8;
    unsigned __int8 miso_buf[sizeof(mosi)];
    bool result = SPI_Transfer_MISO_Delayed(sizeof(mosi), mosi, miso_buf);
    if (result) {
        // success
        return true;
    } // else operation failed
    return false;
}
//-----

```

리스트 2 (14/14)

Maxim cannot assume responsibility for use of any circuitry other than circuitry entirely embodied in a Maxim product. No circuit patent licenses are implied. Maxim reserves the right to change the circuitry and specifications without notice at any time.

40 **Maxim Integrated Products, 120 San Gabriel Drive, Sunnyvale, CA 94086 408-737-7600**