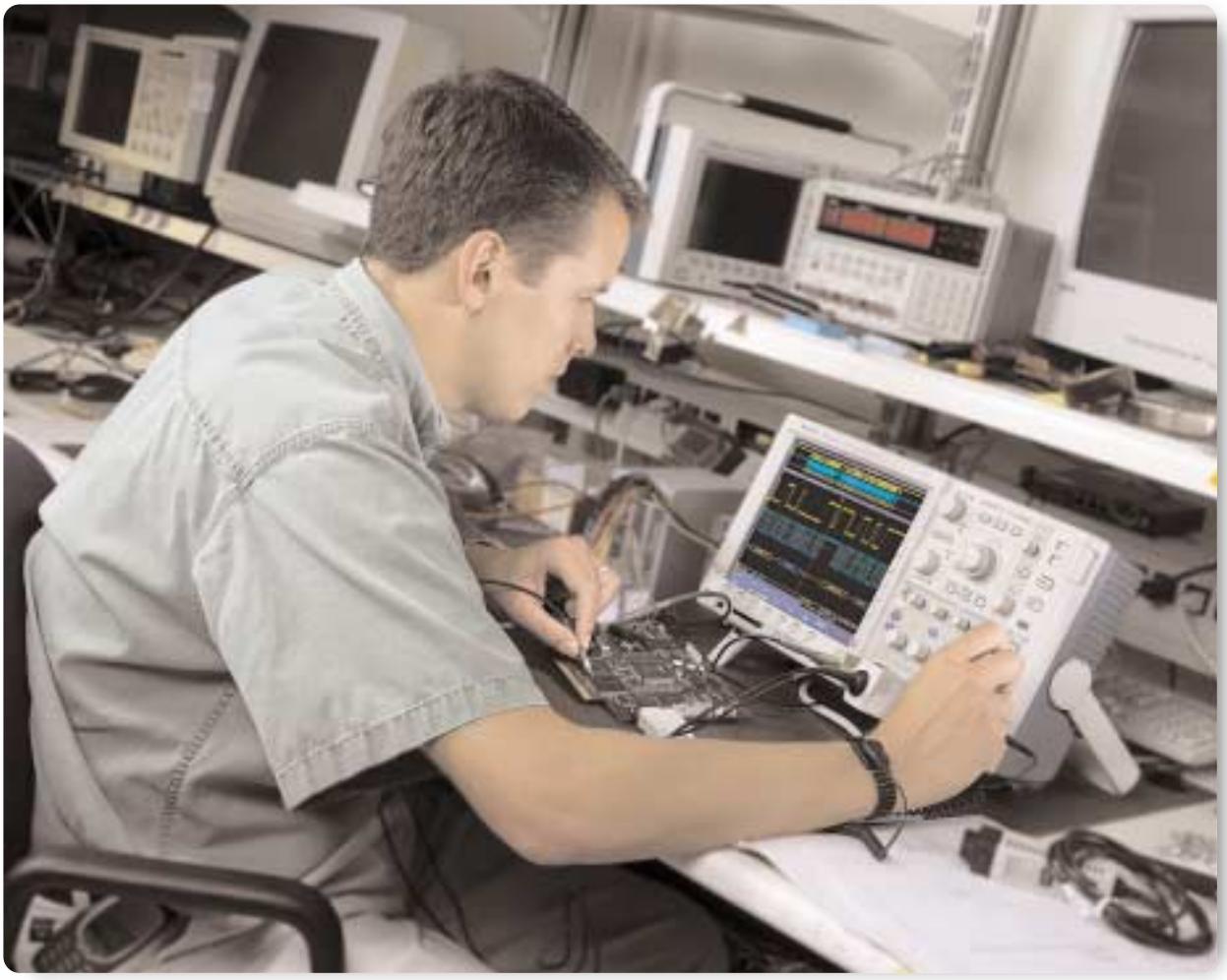
임베디드 시스템 설계의 저속 직렬 버스 디버깅



소개

임베디드 시스템은 사실상 오늘날 우리가 살아가는 사회의 모든 곳에 있다고 할 수 있습니다. 임베디드 시스템의 간단한 정의는 시스템에 모니터링 및 제어 서비스의 제공 목적을 가진 더 큰 시스템의 일부로서 특수 목적의 컴퓨터 시스템입니다.

일반적인 임베디드 시스템은 켜지자마자 특수 목적의 애플리케이션 실행을 시작하고 꺼질 때까지는 작동이 멈추지 않습니다. 사실상, 오늘날 설계되고 생산되는 모든 전자 장치는 임베디드 시스템입니다. 임베디드 시스템의 예로는 다음과 같은 것이 있습니다.

Tektronix
Enabling Innovation

– 알람 클럭
– ATM 시스템
– 휴대폰
– 컴퓨터 프린터
– 앤티록 브레이크 컨트롤러
– 전자레인지
– 미사일 관성 유도 시스템
– DVD 플레이어
– PDA
– 산업 자동화 및 모니터링용 PLC
– 휴대용 음악 플레이어
– 심지어 토스터에도 임베디드 시스템이 있을 수 있습니다.

임베디드 시스템에는 마이크로프로세서, 마이크로컨트롤러, DSP, RAM, EPROM, FPGA, A/D, D/A 및 I/O를 비롯한 다양한 유형의 소자가 포함될 수 있습니다. 이 다양한 소자는 전통적으로 넓은 병렬 버스를 사용하여 서로간 그리고 외부 세계와 통신해왔습니다. 하지만 오늘날에는 임베디드 시스템 설계에 사용되는 점점 더 많은 구성 요소에 이런 넓은 병렬 버스가 직렬 버스로 대체되어가고 있으며 그 이유는 다음과 같습니다.

– 라우팅할 신호가 줄어듬에 따라 필요한 보드 공간 절감
– 비용 저렴
– 저전력 요구
– 패키지 상의 핀 수 적음
– 임베디드 클럭
– 노이즈 내성을 높이기 위한 차동 신호처리
– 표준 직렬 인터페이스를 사용하는 구성 요소를 폭 넓게 이용할 수 있음

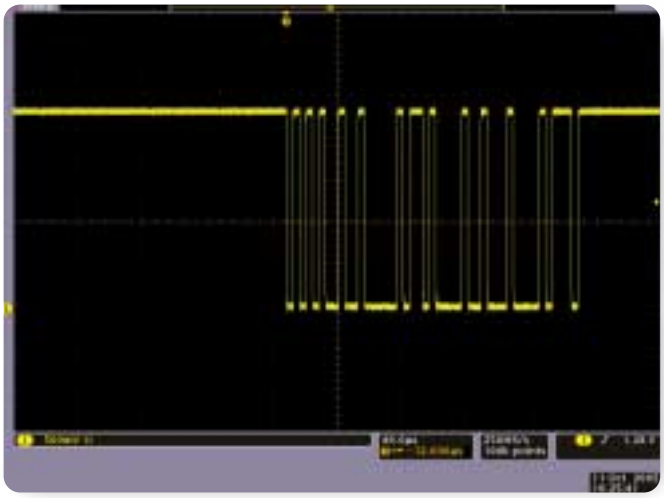
직렬 버스는 수많은 이점을 제공하지만 단지 정보가 병렬이 아닌 직렬 방식으로 전송된다는 사실로 인해 임베디드 시스템 설계자에게 몇 가지 중요한 과제도 안겨 줍니다. 이 응용 자료에서는 임베디드 시스템 설계자에게 남겨진 공통적인 해결 과제와 새로 나온 DPO4000 시리즈 오실로스코프의 각종 기능을 사용하여 이 문제를 극복하는 방법을 설명합니다.

병렬 vs. 직렬

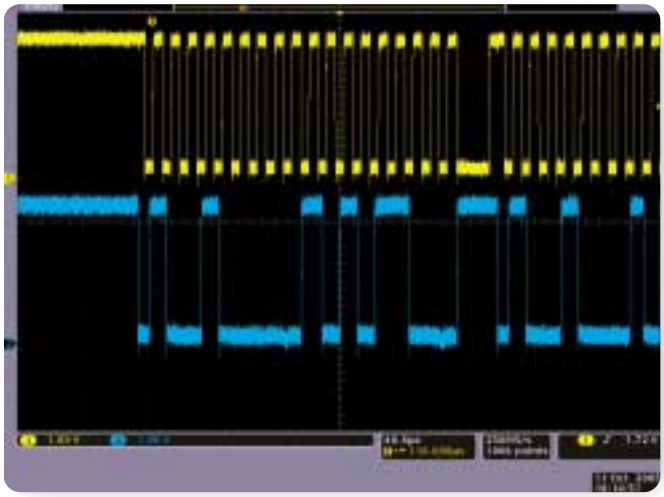
병렬 아키텍처로 되어 있는 버스의 각 소자에는 자체적인 신호 경로가 있습니다. 주소 라인 16개, 데이터 라인 16개, 클럭 라인 1개 그리고 다른 다양한 제어 신호가 있을 수 있습니다. 버스를 통해 전송된 주소 또는 데이터 값은 모든 병렬 라인을 통해서도 동시에 전달됩니다. 이런 특성으로 인해 대부분의 오실로스코프와 로직 분석기에 있는 State(상태) 또는 Pattern(패턴) 트리거링을 이용하여 관심 있는 이벤트에 대해 비교적 쉽게 트리거할 수 있습니다. 또한 오실로스코프나 로직 분석기 디스플레이 중 하나에서 캡처하는 데이터를 한눈에 파악하기도 쉽습니다. 예를 들어 그림 1에서는 로직 분석기를 사용하여 마이크로컨트롤러로부터 클럭, 주소, 데이터 및 제어 라인을 획득했습니다. 상태 트리거를 사용함으로써 우리가 찾고 있는 버스 전송을 따로 분리했습니다. 버스에서 무슨 일이 일어나고 있는지 "디코드"하기 위해 해야 할 일이라고는 오로지 주소, 데이터 및 제어 라인 각각의 논리적 상태를 살펴보는 것입니다. 직렬 버스를 사용하는 경우 이 모든 정보를 동일한 소수의 전도체(때로는 하나의 전도체)에서 직렬로 전송해야 합니다. 이는 단 하나의 신호에 주소, 제어 정보, 데이터 및 클럭 정보가 포함될 수 있음을 의미합니다.

한 예로서 그림 2에 표시된 CAN(Controller Area Network) 직렬 신호를 살펴 봅시다.

▶그림 1. 마이크로컨트롤러의 클럭, 주소 버스, 데이터 버스 및 제어 라인을 로직 분석기가 획득한 모습



▶그림 2. CANbus에서 획득된 한 메시지



▶그림 3. I²C 버스에서 획득된 한 메시지

이 메시지에는 프레임 시작, 식별자(주소), 데이터 길이 코드, 데이터, CRC, 프레임 끝 뿐만 아니라 몇 가지 다른 제어 비트도 들어 있습니다. 문제를 더욱 복잡하게 만드는 것은, 데이터에 클럭이 임베디드되어 있고 클럭에 맞춰 잠기는 소자를 받기 위해 적당한 수의 에지가 발생하도록 비트 삽입을 이용한다는 점입니다. 이런 메시지를 매우 익숙하게 파악할 줄 아는 사람의 눈에도 이 메시지의 내용을 신속히 해석해내기가 매우 어려울 것입니다. 이것이 하루에 한 번만 발생하는 잘못된 메시지이고 이 메시지가 발생할 때 트리거해야 한다고 생각해 봅시다. 전통적인 오실로스코프와 로직 분석기에는 이런 종류의 신호를 다룰 수 있는 기능이 잘 구비되어 있지 않습니다.

I²C와 같이 더 간단한 직렬 표준을 이용하더라도 병렬 프로토콜을 이용할 때에 비해 버스를 통해 무엇이 전송되고 있는지 관찰하기가 여전히 어렵습니다.

I²C는 별개의 클럭과 데이터 라인을 사용하므로 최소한 이 경우 클럭을 기준점으로 사용할 수 있습니다. 하지만 여전히 메시지 시작 부분(클럭이 높을 때는 데이터가 Low 상태가 됨)을 찾고 클럭의 모든 상승 에지에서 데이터 값을 수동으로 검사하여 기록한 다음 비트를 메시지 구조로 구성해야 합니다. I²C는 긴 획득 데이터에서 단 하나의 메시지를 디코드만 하는 데만 몇 분이 걸릴 수 있고 그 메시지가 실제로 찾고 있는 메시지인지 알 수 없습니다. 만약 그렇지 않다면 이 지루하고 오류가 쉽게 발생하곤 하는 프로세스를 다음 프로세스에서 시작해야 합니다. 찾고 있는 메시지 내용에 대해 트리거만 하는 것이 좋겠지만 스코프와 로직 분석기에서 다년간 사용해온 상태 및 패턴 트리거가 여기서는 별 소용이 없을 것입니다.

이런 계측기들은 여러 채널에서 동시에 발생하는 패턴을 찾도록 설계되어 있기 때문입니다. 직렬 버스에서 작업하려면 트리거 엔진 깊이가 수천 개 이상의 상태(비트 당 1개의 상태)가 되어야 합니다. 이런 트리거 기능이 있다 하더라도 이런 모든 비트에 대해 상태별로 프로그램하는 작업이 유쾌할 리는 없을 것입니다. 따라서 더 나은 방법을 찾아야 합니다!

시작	주소	R/W	Ack	Data0	Ack0	Data1	Ack1	.....	DataN	AckN	정지
	7 또는 10비트	1비트	1비트	8비트	1비트	8비트	1비트		8비트	1비트	

▶그림 4. I²C 메시지 구조

DPO4000 시리즈를 사용하면 더 나은 방법을 찾을 수 있습니다. 다음 단원에서는 임베디드 시스템 설계에 가장 일반적으로 사용되는 저속 직렬 표준 중 몇 가지와 함께 DPO4000 시리즈를 사용할 수 있는 방법을 집중 조명하겠습니다.

I²C

배 경

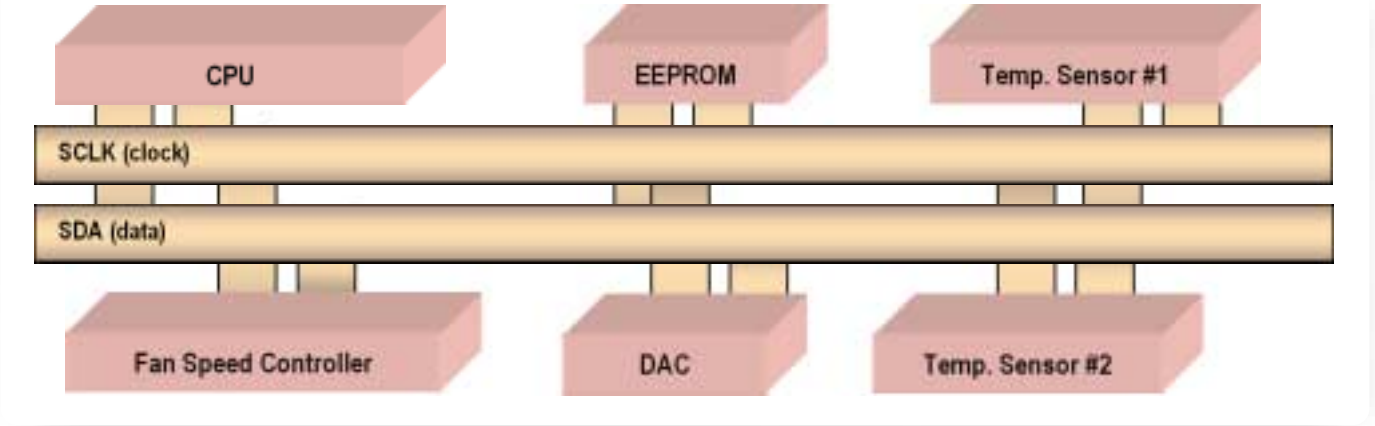
I²C는 Inter Integrated Circuit의 약자입니다. 이 회로는 원래 1980년대 초에 Philips가 TV 세트의 주변장치 칩에 저렴한 비용으로 컨트롤러를 연결할 수 있는 방법을 제공하기 위해 개발한 것이지만 그 이후로 임베디드 시스템 내 소자 간 통신을 위한 세계적 표준으로 발전했습니다. 단순한 2선식 설계를 바탕으로 Analog Devices, Atmel, Infineon, Cyprus, Intel, Maxim, Philips, Silicon Laboratories, ST Microelectronics, Texas Instruments, Xicor 등의 많은 선도적 칩 메이커들이 I/O, A/D, D/A, 온도 센서, 마이크로컨트롤러 및 마이크로프로세서와 같은 다양한 칩을 개발해왔습니다.

작동 원리

I²C의 물리적 2선식 인터페이스는 양방향 직렬 클럭(SCL)과 데이터(SDA) 라인으로 구성됩니다. I²C는 버스 상의 여러 마스터와 슬레이브를 지원하지만 한 번에 한 개의 마스터만 작동할 수 있습니다. 어떤 I²C 장치든 버스에 부착할 수 있으므로 임의의 마스터 장치를 이용하여 슬레이브 장치와 정보를 교환할 수 있습니다. 각 장치는 고유의 주소로 식별되며 장치의 기능에 따라 송신기나 수신기 중 하나로 작동할 수 있습니다. 처음에는 I²C가 7비트 주소만 사용했지만 시간이 지나면서 10비트 주소 지정도 허용하는 방향으로 발전했습니다. 100kbps(기본 모드), 400kbps(패스트 모드) 및 3.4Mbps(고속 모드)의 3가지 비트율이 지원됩니다. 최대 장치 수는 400pf 또는 대략 20–30개의 장치를 포괄하는 최대 커패시턴스로 결정됩니다. I²C 표준에서는 그림 4에서 다음 형식을 지정합니다.

- 시작장치가 – 버스를 제어하고 메시지가 뒤따른다는 것을 나타냅니다.
- 주소 – 어떤 데이터를 읽거나 거기에 쓰는 장치의 주소를 나타내는 7 또는 10비트의 숫자입니다.
- R/W 비트 – 데이터를 해당 장치에서 읽어오거나 그 장치에 쓸 것인지 여부를 나타내는 1비트의 정보입니다.
- 승인(Ack) – 마스터의 동작을 승인하는 슬레이브 장치에서 받는 1비트의 정보입니다. 일반적으로 각 주소와 데이터 바이트에는 승인(Ack)이 하나 있지만 항상 그런 것은 아닙니다.
- 데이터(Data) – 해당 장치에서 읽거나 그 장치에 쓴 정수 바이트입니다.
- 정지 – 메시지가 완료되었고 마스터가 버스를 릴리즈했음을 나타냅니다.

▶그림 5. I²C 버스 설정 메뉴



▶그림 6. I²C 버스 예

I²C를 이용한 작업

DPO4EMBD 직렬 트리거링 및 분석 애플리케이션 모듈을 갖춘 DPO4000 시리즈는 임베디드 시스템 설계자가 I²C 버스를 이용하여 설계하는 데 있어 강력한 도구가 됩니다. 전면 패널에는 사용자가 스코프에 대한 입력을 버스로 정의할 수 있는 2개의 Bus(버스) 버튼(B1과 B2)이 있습니다. I²C 버스 설정 메뉴는 그림 5에 나와 있습니다.

로직 1과 0을 결정하는 데 사용되는 임계값과 함께 단순히 어떤 채널 클럭과 데이터가 실행 중인지 정의함으로써 오실로스코프가 버스를 통해 전송되는 프로토콜을 이해할 수 있도록 해왔습니다. 이 정보를 알고 있는 오실로스코프는 지정된 메시지 레벨 정보에 대해 트리거한 다음 결과 획득 정보를 의미 있고 해석하기 쉬운 결과로 디코드할 수 있습니다.

관심 있는 이벤트를 분석하기 위하여 이벤트를 획득한 다음 메시지를 분석하여 문제를 찾은 후에 메시지를 수동으로 디코드하던 에지 트리거링을 사용하던 시절은 갔습니다.

한 예로서 그림 6에 나온 임베디드 시스템을 생각해 봅시다. I²C 버스가 CPU, EEPROM, 팬 속도 컨트롤러, DAC 그리고 한 쌍의 온도 센서를 포함한 여러 소자에 연결되어 있습니다.

이 시스템은 사용 중에 계속 너무 뜨거워져 자체적으로 작동이 중단되는 문제가 있어 고장 분석을 위해 엔지니어링 팀에 반품되었습니다. 첫 번째로 점검할 것은 팬 컨트롤러와 팬 자체지만 둘 다 정상 작동하는 것으로 보입니다. 다음으로 점검할 사항은 온도 센서 고장 여부입니다. 팬 속도 컨트롤러는 2개의 온도 센서(각각 계측기의 다른 부분에 위치해 있음)를 주기적으로 검사하고 팬 속도를 조정하여 내부 온도를 조절합니다. 이 두 온도 센서 중 하나 또는 둘 다 온도를 올바르게 읽지 못한다는 의심이 듭니다. 센서와 팬 속도 컨트롤러 사이의 상호 작용을 보려면 I^2C 클럭과 데이터 라인에 연결하여 DPO4000 상에 버스를 하나 설정해주기만 하면 됩니다. 두 센서가 I^2C 버스에서 주소가 각각 18번과 19번이라는 점을 알고 있으므로 트리거 이벤트를 설정하여 18번 주소에 대해 쓰기 작업이 이루어지는지 살펴보기로 결정합니다(현재 온도를 센서가 감지하는지 팬 속도 컨트롤러가 검사). 그림 7은 트리거된 획득을 보여주고 있습니다.

이 경우 채널 1(노란색)은 SCLK에, 채널 2(청록색)는 SDA에 연결됩니다. 자주색 파형은 오실로스코프에 몇 가지 간단한 파라미터를 입력하여 정의한 I^2C 버스입니다. 디스플레이 윗부분은 전체 획득을 나타낸 것입니다. 이 경우 화면을 확대한 중간 부분에서 폭발적인 활동이 일어나는 많은 버스 Idle Time을 캡처했습니다. 디스플레이 아래쪽의 더 큰 부분은 확대/축소 윈도우입니다. 보시다시피, 오실로스코프가 버스를 거쳐 지나가는 각 메시지의 내용을 디코드했습니다. DPO4000 시리즈 상의 버스들은 표 1에 나온 색/표시를 사용하여 메시지의 중요한 부분을 나타냅니다.

▶그림 7. I^2C 주소 및 데이터 버스 파형 디코딩

획득된 파형을 살펴보면 오실로스코프가 실은 18번 주소에 대한 Write(쓰기) 작업에서 트리거했다는 것을 알 수 있습니다(디스플레이의 왼쪽 아래에 표시되어 있음). 사실, 팬 속도 컨트롤러는 18번 주소에 두 차례 쓰기를 시도했지만 두 경우 모두 온도 센서에 쓰기를 시도한 후 승인을 받지 못했습니다. 그런 다음 팬 속도 컨트롤러는 19번 주소로 지정된 온도 센서를 확인하여 원하는 정보를 다시 받았습니다. 그렇다면 왜 첫 번째 온도 센서는 팬 컨트롤러에 응답하지 않는 것일까요? 보드 상의 실제 부품을 살펴보면 주소 라인 중 하나의 납땜이 올바로 되어 있지 않은 것을 알게 됩니다. 첫 번째 온도 센서가 버스에서 통신할 수 없는 상태였기 때문에 그 결과로 장치가 과열된 것이었습니다. I^2C 트리거와 DPO4000 시리즈의 버스 디코딩 기능 덕분에 2, 3분 만에 이렇듯 포착하지 못했을 뻔한 문제를 찾아낼 수 있었습니다.

버스 상태	표시
시작(Start)은 녹색 세로 막대로 표시됩니다. Stop(정지)이 앞서 나오지 않고 또 다른 시작이 표시되면 반복 시작이 일어납니다.	
Address(주소)는 쓰기의 경우 [W] 또는 읽기의 경우 [R]과 함께 노란색 상자에 표시됩니다. 주소 값은 16진수나 2진수로 표시될 수 있습니다.	
Data(데이터)는 청록색 상자에 표시됩니다. 데이터 값은 16진수나 2진수로 표시될 수 있습니다.	
Missing Ack(승인 누락)는 빨간색 상자 내에 느낌표로 표시됩니다.	
Stop(정지)은 빨간색 세로 막대로 표시됩니다.	

▶표 1. 버스 상태

그림 7의 예에서 쓰기에 대해 트리거했지만 DPO4000의 강력한 I2C 트리거링에는 다른 수많은 기능이 있습니다.

- Start(시작) – SCL이 High인 상태에서 SDA가 Low가 될 때 트리거합니다.
- Repeated Start(반복 시작) – 정지 상태가 앞서 나오지 않고 시작 상태가 발생하면 트리거합니다. 이것은 보통 마스터가 버스를 릴리즈하지 않고 여러 개의 메시지를 보낼 때입니다.
- Stop(정지) – SCL이 High인 상태에서 SDA가 High가 될 때 트리거합니다.
- Missing Ack(승인 누락) – 슬레이브는 종종 주소 및 데이터의 각 바이트 뒤에 승인 메시지를 전송하도록 구성됩니다. 오실로스코프는 슬레이브가 승인 비트를 생성하지 않는 경우에 트리거할 수 있습니다.
- Address(주소) – 사용자가 지정한 주소나 General Call(일반 호출), Start Byte(시작 바이트), HS 모드, EEPROM 또는 CBUS를 비롯하여 미리 프로그램된 특수 주소에서 트리거합니다. 주소는 7비트나 10비트로 지정될 수 있으며 2진수나 16진수로 입력됩니다.
- Data(데이터) – 2진수나 16진수 중 하나로 입력된 최대 12바이트의 사용자 지정 데이터 값에 대해 트리거합니다.
- Address and Data(주소 및 데이터) – 이것을 이용해 관심 있는 이벤트를 정확하게 캡처하기 위한 읽기와 쓰기 뿐만 아니라 주소와 데이터 값 모두 입력할 수 있습니다.

이들 트리거를 이용하여 관심 있는 특정 버스 트래픽을 따로 분리할 수 있는 한편, 디코딩 기능을 이용해 획득된 데이터에서 버스를 통해 전송된 모든 메시지의 내용을 즉시 볼 수 있습니다.

SPI

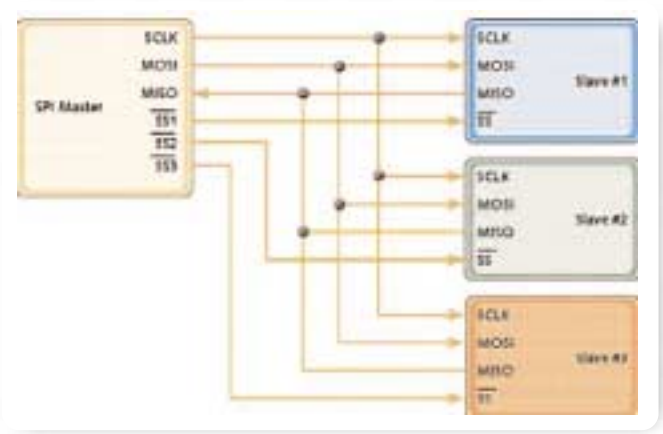
배경

SPI(Serial Peripheral Interface) 버스는 원래 1980년대 말에 Motorola가 68000 시리즈 마이크로컨트롤러용으로 개발했습니다. 이 버스는 구조가 간단하고 인기가 높아 다른 많은 제조업체가 다년간에 걸쳐 표준으로 채택했습니다. 이 버스는 현재 임베디드 시스템 설계에서 일반적으로 사용되는 다양한 소자에서 찾아볼 수 있습니다. SPI는 주로 마이크로컨트롤러와 이에 직접 연결된 주변 장치 사이에 사용됩니다. 휴대폰, PDA 및 다른 모바일 장치에서 SPI가 CPU, 키보드, 디스플레이 그리고 메모리 칩 사이에서 데이터 통신을 담당하는 것을 흔히 볼 수 있습니다.

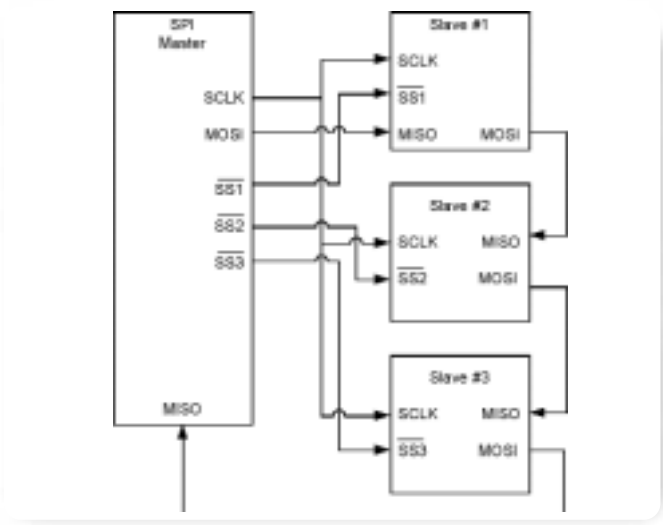
작동 원리

SPI(Serial Peripheral Interface) 버스는 마스터/슬레이브 방식의 4선식 직렬 통신 버스입니다. 4개의 신호는 클럭(SCLK), 마스터 출력/슬레이브 입력(MOSI), 마스터 입력/슬레이브 출력(MISO), 슬레이브 선택(SS)입니다. 두 장치가 통신할 때마다 하나를 "마스터", 다른 하나를 "슬레이브"라고 합니다. 마스터가 직렬 클럭을 구동합니다. 데이터는 동시에 송수신되어 전이중 프로토콜을 구성합니다. SPI는 버스 상의 각 장치에 고유 주소를 지정하지 않고 SS 라인을 사용하여 어떤 장치 데이터가 송수신될지 지정합니다. 이와 같이, 버스 상에 있는 각각의 고유 장치는 저마다 마스터로부터 SS 신호를 수신해야 합니다. 3개의 슬레이브 장치가 있는 경우 마스터로부터 3개의 SS 리드 선이 있는데, 그림 8에 나타낸 것과 같이 서로에게 각각 슬레이브가 됩니다.

그림 8에서 각 슬레이브는 마스터와만 통신합니다. 하지만 SPI를 데이지 체인 방식의 슬레이브 장치와 유선으로 연결하여 각각 차례대로 작동을 수행한 다음 그 결과를 그림 9에 표시한 것과 같이 마스터로 되돌려보낼 수 있습니다.



▶그림 8. 일반적인 SPI 구성



▶그림 9. 데이지 체인 방식의 SPI 구성

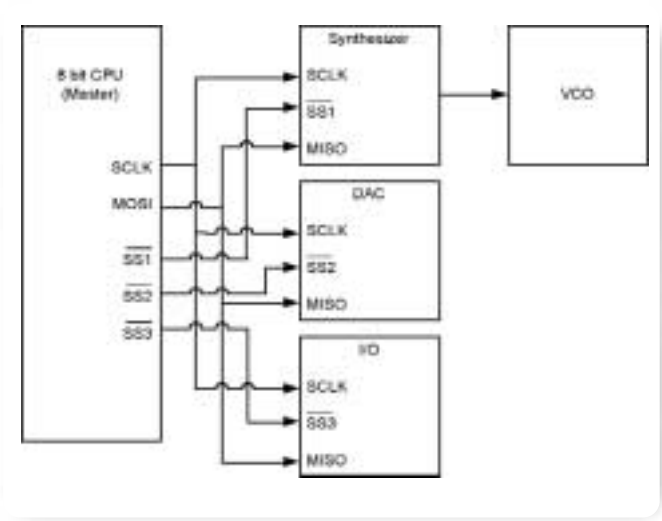
따라서 아시다시피 SPI 구현을 위한 "표준"은 따로 없습니다. 슬레이브에서 마스터로 반대 방향의 통신이 필요 없는 일부 경우에는 MISO 신호가 모두 무시될 수 있습니다.

▶그림 10. SPI 버스 설정 메뉴

SPI 데이터 전송이 발생하면 8비트의 데이터 워드가 MOSI로 시프트되는 동시에 다른 8비트 데이터 워드가 MISO로 시프트됩니다. 이것은 16비트의 순환 시프트 레지스터로 볼 수 있습니다. 전송이 발생하면 이 16비트의 시프트 레지스터가 8개의 위치만큼 이동되고 따라서 마스터와 슬레이브 장치 간에 8비트 데이터를 교환합니다. 한 쌍의 레지스터, 클럭 극성(CPOL) 및 클럭 위상(CPHA)이 데이터가 구동되는 클럭의 에지를 결정합니다. 각 레지스터에는 서로가 모두 호환 가능한 4가지의 조합을 구현할 수 있는 2가지의 상태가 있을 수 있습니다. 따라서 마스터/슬레이브 쌍이 서로 통신하려면 동일한 파라미터 값을 사용해야 합니다. 서로 다른 구성으로 고정된 여러 개의 슬레이브를 사용하는 경우 마스터는 다른 슬레이브와 통신할 필요가 있을 때마다 스스로 재구성해야 합니다.

SPI를 이용한 작업

DPO4EMBD 직렬 트리거링 및 분석 애플리케이션 모듈도 SPI 버스에 대해 유사한 기능을 지원합니다. 여기서 다시 전면 패널의 B1 또는 B2 버튼을 사용하여 SCLK, SS, MOSI 및 MISO 중 어떤 채널이 켜져 있는지, 임계값과 극성은 어떻게 되는지를 비롯하여 버스의 기본적인 파라미터만 입력하면 SPI 버스를 정의할 수 있습니다(그림 10 참조).
한 예로서 그림 11의 임베디드 시스템을 생각해 봅시다.



▶그림 11. SPI를 통해 제어되는 합성기

SPI 버스는 합성기, DAC 및 일부 I/O에 연결되어 있습니다. 합성기는 시스템 나머지 부분에 2.5GHz의 클럭을 제공하는 VCO에 연결되어 있습니다. 이 합성기는 시작 시 CPU에 의해 프로그램되는 것으로 가정합니다. 하지만 VCO가 3GHz를 생성하는 레일에 고정되어 있으므로 어떤 것은 올바르게 작동하지 않습니다. 이 문제를 디버깅하는 첫 번째 단계는 신호가 존재하는지 여부와 물리적 연결 문제는 없다는 점을 확인하기 위해 CPU와 합성기 사이의 신호를 검사하는 것이지만 잘못된 점을 찾지는 못합니다. 다음으로, 합성기를 프로그램하기 위해 SPI 버스를 통해 전달되는 실제 정보를 살펴보기로 합니다. 이 정보를 캡처하기 위해 합성기에서 활성 상태가 되는 Slave Select(슬레이브 선택) 신호가 발생할 때 트리거하도록 오실로스코프를 설정하고 DUT 전원을 켜서 시작 프로그래밍 명령을 캡처합니다. 그림 12에 획득된 결과가 표시되어 있습니다.

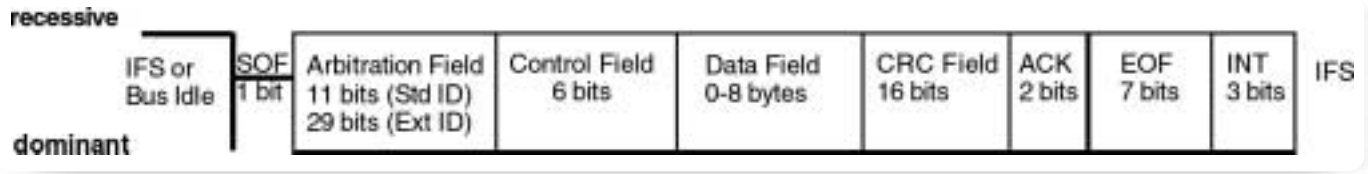
채널 1(노란색)은 SCLK, 채널 2(청록색)는 MOSI, 채널 3(자홍색)은 SS입니다. 장치를 올바르게 프로그래밍하고 있는지 판단하기 위해 합성기에 대한 데이터 시트를 살펴봅니다. 버스에 대한 첫 3개의 메시지는 합성기를 초기화하고 분주기 비율을 로드하고 데이터를 래치하는 것이 됩니다. 사양에 따르면 첫 3개의 전달 메시지에 있는 마지막 니블(단일 16진수 문자)은 각각 3, 0, 1이 되어야 하지만 0, 0, 0인 것으로 나타납니다. 메시지 끝에서 모든 0을 보면서 소프트웨어에서 역순으로 각 24비트 워드에서 비트를 프로그래밍함으로써 SPI를 사용할 때 가장 흔히 저지르는 실수 중 하나를 저질렀다는 것을 알게 됩니다. 소프트웨어에서의 빠른 변화는 그림 13에 나타낸 것과 같이 다음 획득이 발생하고 VCO가 2.5GHz에서 올바르게 잠기는 결과를 초래합니다. 위 예에서 우리는 간단한 SS Active 트리거를 사용했습니다. DPO4000 시리즈의 전체 SPI 트리거링 기능에는 다음 유형이 포함됩니다.

- SS Active – 슬레이브 선택 라인이 슬레이브 장치에 대해 true가 될 때 트리거합니다.
- MOSI – 마스터에서 슬레이브까지 최대 16바이트의 사용자 지정 데이터를 트리거합니다.
- MISO – 슬레이브에서 마스터까지 최대 16바이트의 사용자 지정 데이터를 트리거합니다.
- MOSI/MISO – 마스터에서 슬레이브 및 슬레이브에서 마스터 두 경우 모두에 대해 최대 16바이트의 사용자 지정 데이터를 트리거합니다.

다시 한 번 이들 트리거를 이용하여 관심 있는 특정 버스 트래픽을 따로 분리할 수 있는 한편, 디코딩 기능을 이용해 획득된 데이터에서 버스를 통해 전송된 모든 메시지의 내용을 즉시 볼 수 있습니다.

▶그림 12. SPI 버스의 합성기 구성 메시지 획득

▶그림 13. 올바른 합성기 구성 메시지



▶그림 14. CAN 데이터/원격 프레임

CAN

배경

CAN(Controller Area Network)은 원래 1980년대에 Robert Bosch GmbH가 전기 노이즈가 많은 환경에서 작동하는 장치 간에 사용할 저가형 통신 버스로 개발되었습니다. 1992년 Mercedes-Benz는 자동차 시스템에 CAN을 채택한 최초의 자동차 제조사가 되었습니다. 현재는 거의 모든 자동차 제조사가 CAN 컨트롤러와 네트워크를 사용하여 윈드실드 와이퍼 모터 컨트롤러, 우적 감지 센서, 에어백, 도어 잠금 장치, 엔진 타이밍 컨트롤, 앤티 록 브레이킹 시스템, 동력 전달 장치 컨트롤 및 전동식 윈도우 등의 장치를 제어하며 그 예는 너무나 많습니다. 전기 노이즈 허용 오차, 최소한의 배선, 우수한 오류 탐지 능력과 고속 데이터 전송 능력 덕분에 CAN은 산업 공정 제어, 해운, 의료, 우주항공 등과 같은 다른 응용 분야로 급속히 확대되어가고 있습니다.

작동 원리

CAN 버스는 STP(Shielded Twisted Pair), UTP(Un-shielded Twisted Pair) 또는 리본 케이블 중 하나에서 작동하는 평형(차동) 2선식 인터페이스입니다. 각 노드에서는 9핀 D 수 커넥터를 사용합니다. NRZ(Non Return to Zero) 비트 인코딩이 비트 삽입과 함께 사용되어 최소한의 전이와 높은 노이즈 내성을 가진 컴팩트한 메시지를 만들 수 있습니다. CAN 버스 인터페이스는 버스를 사용할 수 있을 때마다 어떤 노드든 전송을 시작할 수 있는 비동기 전송 방식을 사용합니다. 메시지는 네트워크 상의 모든 노드로 브로드캐스트됩니다. 여러 개의 노드가 동시에 메시지를 보내기 시작하는 경우에는 비트 단위 중재 기능을 이용해 어떤 메시지의 우선 순위가 더 높은지 결정합니다. 메시지는 Data Frame(데이터 프레임), Remote Transmission Request(RTR) Frame(원격 전송 요청 프레임), Error Frame(오류 프레임) 또는 Overload Frame(과부하 프레임)의 4가지 유형 중 하나가 될 수 있습니다. 버스 상의 어떤 노드든 오류를 탐지하면 오류 프레임을 전송하여 버스 상의 모든 노드가 현재 메시지를 불완전한 것으로 보고 전송 노드가 그 메시지를 재전송하게끔 합니다. 과부하 프레임은 장치가 아직 데이터를 받을 준비가 되어 있지 않음을 표시하기 위해 수신 장치가 시작합니다. 데이터 프레임은 데이터를 전송하는 데 사용되는 반면 원격 프레임은 데이터를 요청하는 데 사용됩니다. 데이터 및 원격 프레임은 각 프레임의 시작과 끝에서 시작 및 정지 비트에 의해 제어되고 그림 14에 표시한 것처럼 Arbitration(중재) 필드, Control(제어) 필드, Data(데이터) 필드, CRC 필드 및 ACK 필드와 같은 필드를 포함합니다.

– SOF – 프레임이 SOF(start of frame) 비트로 시작됩니다.

– Arbitration(중재) – Arbitration(중재) 필드에는 데이터 프레임과 데이터 요청 프레임(원격 프레임이라고도 함)을 구분하는 데 사용되는 식별자(주소)와 RTR(Remote Transmission Request) 비트가 포함됩니다. 식별자는 표준 형식(11비트 – 버전 2.0A) 또는 확장 형식(29비트 – 버전 2.0B) 중 하나가 될 수 있습니다.

– Control(제어) – Control(제어) 필드는 CAN 2.0A(11비트 식별자) 표준 프레임과 CAN 2.0B(29비트 식별자) 확장 프레임을 구분하는 IDE(Identifier Extension) 비트를 포함한 6개의 비트로 구성됩니다. Control(제어) 필드에는 DLC(Data Length Code)도 포함됩니다. DLC는 데이터 프레임의 데이터 필드에 있는 바이트 수나 원격 프레임에서 요청하는 바이트 수를 나타내는 4비트의 코드입니다.

– Data(데이터) – 데이터 필드는 0~8바이트의 데이터로 구성됩니다.

– CRC – 15비트의 순환 중복 검사(CRC) 코드와 역행 구분 문자 비트입니다.

–ACK – Acknowledge(승인) 필드는 길이가 2비트입니다. 첫 번째 비트는 역행으로 전송되는 슬롯 비트지만 전송된 메시지를 성공적으로 수신하는 노드에서 전송되는 우세한 비트로 덮어쓰기가 됩니다. 두 번째 비트는 역행 구분 문자 비트입니다.

– EOF – 7개의 역행 비트로서 EOF(end of frame)를 나타냅니다.

3개의 역행 비트로 구성된 INT(중단) 필드는 버스를 사용할 수 있는 상태임을 나타냅니다. 버스 Idle Time은 0을 포함한 임의의 길이가 될 수 있습니다.

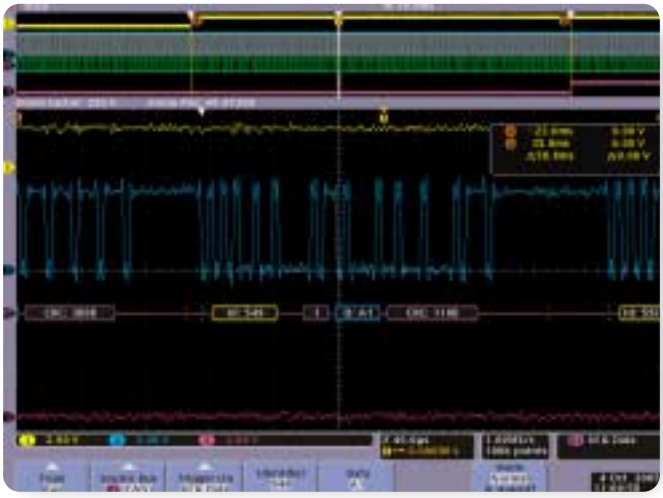
1Mb/s를 가장 빠른 속도로, 5kb/s를 가장 느린 속도로 하여 다양한 데이터 전송률이 정해집니다. 모든 모듈은 최소한 20kb/s를 지원해야 합니다. 케이블 길이는 사용되는 데이터 전송률에 따라 결정됩니다. 일반적으로 시스템 내의 모든 장치는 균일하고 고정된 비트율로 정보를 전달합니다. 최대 라인 길이는 저속에서는 수천 미터가 될 수 있고 1Mbps의 속도에서는 보통 40미터입니다. 케이블의 각 단부에는 종단 저항이 사용됩니다.

▶그림 15. CAN 버스 설정 메뉴

CAN을 이용한 작업

DPO4AUTO 직렬 트리거링 및 분석 애플리케이션 모듈은 CAN 버스에 대해 비슷한 트리거링 및 분석 기능을 지원합니다. 다시 전면 패널의 B1 또는 B2 버튼을 사용하여 프로브 대상 CAN 신호의 종류와 전달되는 채널, 비트율, 임계값 및 샘플 포인트(비트 시간 중 %로 표시)를 포함한 버스의 기본적인 파라미터만 입력하면 CAN 버스를 정의할 수 있습니다(그림 15 참조).

운전자가 승객석 차창 내림 스위치를 누를 때부터 운전석 도어에 있는 CAN 모듈이 명령을 내린 다음 승객석 차창이 실제로 움직이기 시작할 때까지의 지연 시간과 관련된 타이밍 계측을 수행해야 한다고 생각해 봅시다. "차창을 아래로 내려라"는 명령과 관련된 데이터 뿐만 아니라 운전석 도어에 있는 CAN 모듈의 ID를 지정함으로써 찾고 있는 정확한 데이터 프레임에서 트리거할 수 있습니다. 운전석 도어에 있는 차창 내림 스위치와 승객석 도어에 있는 모터 구동을 동시에 프로빙함으로써 이 타이밍 계측은 그림 16에 나타낸 것처럼 매우 쉬워집니다.



▶그림 16. CAN 버스 상의 특정 식별자 및 데이터에 대한 트리거링 및 획득 데이터의 모든 메시지 디코딩

그림에 있는 흰색 삼각형은 파형에 배치하여 기준점으로 삼는 표시입니다. 오실로스코프의 전면 패널에 있는 Set/Clear Mark(표시 설정/지우기) 버튼만 누르면 이런 표시를 디스플레이에 추가하거나 디스플레이에서 제거할 수 있습니다. 전면 패널에 있는 Previous(이전) 및 Next(다음) 버튼을 누르면 확대/축소 윈도우가 한 표시에서 다음 표시로 이동하므로 획득한 데이터에서 관심 있는 이벤트 사이를 간단히 이동할 수 있습니다.

이제 이런 기능을 이용하지 않고 이 작업을 수행한다고 생각해 봅시다. CAN 트리거링 기능이 없으면 스위치 자체에 대해 트리거하고 충분히 긴 활동 시간 범위를 가진 신호를 캡처한 다음 최종적으로 알맞은 프레임을 찾을 때까지 CAN 버스 상에서 프레임을 하나하나 수동으로 디코딩하기 시작해야 할 것입니다. 과거에는 수십 분 내지는 몇 시간이나 걸렸을 작업이 이제는 거의 순간적으로 수행될 수 있습니다.

DPO4000의 강력한 CAN 트리거링 기능으로는 다음과 같은 것이 있습니다.

- Start of Frame(프레임 시작) – SOF 필드에서 트리거합니다.
- Frame Type(프레임 종류) – Data Frame(데이터 프레임), Remote Frame(원격 프레임), Error Frame(오류 프레임) 및 Overload Frame(과부하 프레임)
- Identifier(식별자) – Read/Write(읽기/쓰기) 인증 기능으로 특정한 11비트 또는 29비트의 식별자 값에서 트리거합니다.
- Data(데이터) – 1–8바이트의 사용자 지정 데이터에서 트리거합니다.
- Missing Ack(승인 누락) – 수신 장치가 승인 메시지를 제공하지 않을 때마다 트리거합니다.
- End of Frame(프레임 끝) – EOF 필드에서 트리거합니다.

이런 종류의 트리거를 이용하면 CAN 버스에서 찾고 있는 사실상 어떤 것이든 쉽사리 분리해낼 수 있습니다. 하지만 트리거링은 시작에 불과합니다.

▶그림 17. CAN 이벤트 테이블

문제해결을 위해서는 종종 트리거 이벤트 전후 모두에 메시지 내용을 검사해야 합니다. 획득 데이터에서 여러 메시지의 내용을 볼 수 있는 간단한 방법은 그림 17에 나타낸 DPO4000 시리즈의 Event Table(이벤트 테이블)을 이용하는 것입니다.

이벤트 테이블은 타임스탬프와 함께 표 형식으로 획득 데이터의 모든 메시지에 대해 디코드된 메시지 내용을 표시합니다. 이 테이블을 이용하면 버스 상의 모든 트래픽을 쉽게 볼 수 있을 뿐만 아니라 메시지 간에 타이밍 계측을 쉽게 수행할 수도 있습니다. 이벤트 테이블은 I^2C와 SPI 버스에 대해 모두 이용 가능합니다.

트리거링과 검색

본 응용 자료에서 내내 설명한 바와 같이, 직렬 버스에서 관심 있는 이벤트를 따로 분리하려면 기능이 뛰어난 트리거링 시스템이 필요합니다. 하지만 일단 데이터를 획득한 후(스코프는 정지됨) 이를 분석하려면 트리거링이 더 이상 적용되지 않습니다. 정지된 파형 데이터를 분석하기 위해 트리거와 같은 리소스가 스코프에 있다면 좋지 않을까요? DPO4000 시리즈의 Wave Inspector는 강력한 검색 기능과 함께 이런 기능을 제공합니다. 이 문서에서 논의된 모든 버스 트리거 기능은 이미 획득한 데이터에 대한 검색 기준으로도 사용할 수 있습니다. 예를 들어 그림 18에서 오실로스코프는 특정 주소와 데이터 내용을 가진 모든 CAN 메시지에 대해 긴 획득을 통해 검색하여 디스플레이 상단에 획득된 메시지를 속이 빈 흰색 삼각형으로 표시했습니다. 전면 패널의 Previous(이전)와 Next(다음) 버튼만 누르면 간단히 발생된 이벤트 간에 이동할 수 있습니다.

물론, 보다 전통적인 트리거 종류에 대해서도 검색 기능을 사용할 수 있습니다. 검색 종류에는 에지, 펄스 폭, 런트, 설정 및 보류 시간, 로직 그리고 상승/하강 시간이 포함됩니다.

결론

임베디드 시스템 설계에 있어 병렬 버스에서 직렬 버스로 옮겨가게 되면 많은 이점이 있지만 설계 엔지니어에게도 수많은 해결 과제가 남겨지게 됩니다.

▶그림 18. CAN 버스 획득에서 지정된 식별자와 데이터 검색

전통적인 테스트 및 계측 도구를 사용하면 찾고 있는 이벤트가 발생할 때 이를 트리거하기가 훨씬 더 어려우므로 아날로그 신호를 관찰하는 것만으로는 어떤 정보가 있는지 알아내기가 거의 불가능에 가까우며 문제를 진단하기 위해 주기가 긴 버스 활동을 수동으로 디코드해야 하기 때문에 시간이 엄청나게 많이 걸리고 오류도 발생할 가능성이 높습니다. DPO4000 시리즈는 모든 것을 바꾸어 놓습니다. 오늘날의 설계 엔지니어들은 DPO4000 시리즈의 강력한 트리거, 디코드 및 검색 기능을 이용하여 매우 효율적으로 임베디드 시스템 설계 문제를 해결할 수 있습니다.

텍트로닉스 연락처:

동남아시아/대양주/파키스탄 (65) 6356 3900
오스트리아 +41 52 675 3777
발칸, 이스라엘, 남아프리카 및 다른 ISE 국가들 +41 52 675 3777
벨기에 07 81 60166
브라질 및 남미 55 (11) 3741-8360
캐나다 1 (800) 661-5625
중앙동유럽, 우크라이나 및 발트국 +41 52 675 3777
중앙 유럽 및 그리스 +41 52 675 3777
덴마크 +45 80 88 1401
핀란드 +41 52 675 3777
프랑스 및 북아프리카 +33 (0) 1 69 86 81 81
독일 +49 (221) 94 77 400
홍콩 (852) 2585-6688
인도 (91) 80-22275577
이태리 +39 (02) 25086 1
일본 81 (3) 6714-3010
룩셈부르크 +44(0) 1344 392400
멕시코, 중앙아메리카 및 카리브해 52 (55) 56666-333
중동, 아시아 및 북아프리카 +41 52 675 3777
네덜란드 090 02 021797
노르웨이 800 16098
중국 86 (10) 6235 1230
폴란드 +41 52 675 3777
포르투갈 80 08 12370
대한민국 82 (2) 528-5299
러시아 및 CIS 7 095 775 1064
남아프리카 +27 11 254 8360
스페인 (+34) 901 988 054
스웨덴 020 08 80371
스위스 +41 52 675 3777
대만 886 (2) 2722-9622
영국 및 아일랜드 +44 (0) 1344 392400
미국 1 (800) 426-2200
기타 지역: 1 (503) 627-7111

최종 갱신일 2005년 6월 15일

추가 정보

Tektronix는 최첨단 기술을 다루는 엔지니어를 지원하기 위해 응용 자료, 기술 문서 및 기타 리소스 등을 총 망라한 방대한 자료를 보유 관리하고 있으며 이를 계속 확장하고 있습니다.
www.tektronix.com을 참조하십시오.

10/05 EA/WOW 48K-19040-0

Tektronix
Enabling Innovation