



아날로그 엔지니어의 회 로 안내서: MSPM0 MCU

편집자 메시지

Arm® Cortex®-M0+ 기반 MCU에 대한 이 아날로그 엔지니어 회로 안내서는 설계자가 특정 시스템 요구 사항을 충족하도록 빠르게 조정할 수 있는 서브시스템 예제를 제공합니다. TI MSPM0 MCU는 크기가 작고 비용 경쟁력이 있으며, 역사적으로 고정 기능 아날로그 장치에 의해 수행되었던 시스템을 대체하도록 설계되었습니다. 이 안내서는 이러한 서브시스템에 대한 자세한 개요를 제공하며, 각 서브시스템은 단계별 지침, 설계 인사이트, 소프트웨어 및 기능 향상을 위한 제안이 모두 포함된 예제로 제시됩니다. 이러한 서브시스템을 독립형 시스템으로 사용하거나 함께 결합하여 더 복잡한 애플리케이션을 만들 수 있습니다.

MSPM0 포트폴리오는 특정 시스템 요구 사항의 크기와 단순성 요구 사항에 맞출 수 있는 옵션으로 확장이 가능하며, 모든 서브시스템 예제를 **SysConfig**를 사용하여 MSPM0 포트폴리오의 모든 장치로 손쉽게 이식할 수 있습니다. www.ti.com/mspm0에서 전체 MSPM0 포트폴리오를 확인하세요. MCU 설계가 처음이라면 **TIPL(TI Precision Labs) 마이크로컨트롤러** 교육 시리즈와 **Zero Code Studio** 교육을 완료하는 것이 좋습니다. **E2E 포럼**에서 질문 및 지원을 확인하세요.

목차

편집자 메시지.....	2
아날로그 및 감지.....	3
ADC-PWM.....	4
ADC를 이용한 DMA 핑퐁.....	9
디지털 FIR 필터.....	13
ADC-I2C.....	17
디지털 IIR 필터.....	21
ADC-SPI.....	24
ADC-UART.....	26
데이터 센서 애그리게이터 서브시스템 설계.....	29
M0 장치를 지원하는 2개의 OPA 계측 증폭기.....	37
동적 프로그래머블 게인 증폭기.....	40
스캐닝 비교기.....	48
트랜스임피던스 증폭기.....	54
서미스터 온도 감지.....	59
통신 브리지.....	64
CAN-I2C 브리지.....	65
I2C-UART 서브시스템 설계.....	75
CAN-SPI 브리지.....	82
CAN-UART 브리지.....	90
병렬 IO-UART 브리지.....	98
UART 브리지를 통한 I2C 확장기.....	103
UART-I2C 브리지.....	109
UART-SPI 브리지.....	114
기타 MCU 기능.....	119
디지털 MUX 에뮬레이션.....	120
5V 인터페이스.....	124
작업 스케줄러.....	126
타이밍 및 제어.....	132
연결된 다이오드 매트릭스.....	133
주파수 카운터: 톤 감지.....	138
PWM을 사용하는 LED 드라이버.....	143
전원 시퀀서.....	147
PWM DAC.....	151

아날로그 및 감지

- ADC-PWM •
- ADC를 이용한 DMA 핑퐁 •
- 디지털 FIR 필터 •
- ADC-I2C •
- 디지털 IIR 필터 •
- ADC-SPI •
- ADC-UART •
- 데이터 센서 애그리게이터 서브시스템 설계 •
- M0 장치를 지원하는 2개의 OPA 계측 증폭기 •
- 동적 프로그래머블 게인 증폭기 •
- 스캐닝 비교기 •
- 트랜스임피던스 증폭기 •
- 서미스터 온도 감지 •

ADC-PWM

설명

이 예제는 아날로그 신호를 4kHz PWM 출력으로 변환하는 방법을 보여줍니다. 아날로그 입력 신호는 MSPM0 통합 ADC를 사용하여 샘플링됩니다. PWM 출력의 듀티 사이클은 ADC 판독값을 기준으로 업데이트됩니다. 이 예제에는 두 개의 타이머가 필요하며, 하나는 ADC 판독을 트리거하고 다른 하나는 PWM 출력을 생성하는 데 사용됩니다. 이 예제의 코드를 다운로드하세요.

그림 1에서는 이 예제에서 사용된 주변 기기의 기능 블록 다이어그램을 보여줍니다.

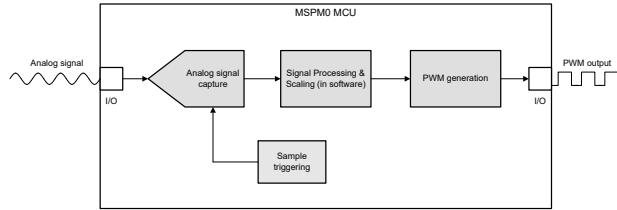


그림 1. 서브시스템 기능 블록 다이어그램

필요한 주변 기기

이 애플리케이션에는 타이머 2개, 통합 ADC 1개, 장치 핀 2개가 필요합니다.

표 1. 주변 기기 요구 사항

하위 블록 기능	주변 장치 사용	참고
샘플 트리거링	(1x) 타이머 G	코드에서 <code>TIMER_0_INST</code> 라고 부름
PWM 생성	(1x) 타이머 G	코드에서 <code>PWM_0_INST</code> 라고 부름
아날로그 신호 캡처	ADC 채널 1개	코드에서 <code>ADC12_0_INST</code> 라고 부름
IO	2핀	(1x) ADC 입력 (1x) PWM 출력

호환 가능 장치

표 1의 요구 사항에 따라 이 예제는 표 2의 장치와 호환됩니다. 해당 EVM은 프로토타입 제작에 사용할 수 있습니다.

표 2.

MSPM0Lxxx	EVM
MSPM0Lxxx	LP-MSPM0L1306

표 2. (계속)

MSPM0Lxxx

LP-MSPM0G3507

설계 단계

1. 필요한 PWM 출력 주파수 및 해상도를 결정합니다. 이 두 매개 변수는 다른 설계 매개 변수를 계산할 때의 시작점입니다. 이 예제에서는 4kHz의 PWM 출력 주파수와 10비트의 PWM 해상도를 선택했습니다.
2. 타이머 클럭 주파수를 계산합니다. $F_{\text{clock}} = F_{\text{pwm}} \times \text{해상도}$ 방정식을 사용하여 타이머 클럭 주파수를 계산할 수 있습니다.
3. ADC 샘플링 속도를 결정합니다. 샘플링 속도는 출력 PWM 주파수와 관련이 있습니다. 이 예제에서는 단일 ADC 샘플을 통해 듀티 사이클이 결정됩니다. $F_{\text{adc}} = F_{\text{pwm}}$. 그러나 필터링 및 평균화를 위해서는 애플리케이션에서 다른 샘플링 속도를 선택해야 할 수 있습니다.
4. **SysConfig**에서 주변 기기를 구성합니다. 사용할 타이머 인스턴스를 선택합니다. ADC 입력 및 PWM 출력에 사용할 장치 핀을 구성합니다. 이 예제에서는 PWM 출력(타이머 G4에 연결됨)에 PA17을 사용하고 아날로그 입력에 A0.4를 사용합니다.
5. 애플리케이션 코드를 작성합니다. 이 애플리케이션의 나머지 부분은 ADC 샘플을 PWM 타이머로 전송하는 것입니다. 이는 소프트웨어에서 수행됩니다. 소프트웨어 흐름도에서 애플리케이션의 개요를 확인하거나 코드를 직접 살펴봅니다.

설계 고려 사항

1. 최대 출력 주파수: 기본적으로 최대 PWM 출력 주파수는 IO 속도에 의해 제한됩니다. 그러나 듀티 사이클 해상도는 최대 출력 주파수에도 영향을 미칩니다. 해상도를 높이려면 타이머 수가 더 많아야 하므로 출력 기간이 늘어납니다.
2. 클로킹: 사용할 클럭과 사용할 클럭 분할 비율을 결정하는 것은 이 애플리케이션에서 중요한 설계 고려 사항입니다.
 - a. 확장 작업에서 b를 곱하고 나누는 대신 시프트를 사용할 수 있도록 2의 거듭제곱 해상도를 선택합니다.
 - b. 일반적으로 느린 클럭을 더 낮은 주파수로 분할하면 안 됩니다. 대신 소비 전력을 줄이기 위해 느린 클럭을 선택합니다.
3. gCheckADC의 경쟁 상태: 이 애플리케이션은 가능한 한 빨리 gCheckADC를 지우도록 합니다. 애플리케이션이 gCheckADC를 너무 늦게 지우면 새로운 데이터를 놓칠 수 있습니다.
4. 파이프라인: 이 애플리케이션에서 선택한 PWM 타이머는 타이머 비교 값 파이프라인을 지원합니다. 파이프라인을 사용하면 애플리케이션에서 출력에서 글리치를 유발하지 않고 타이머 비교 값에 대한 업데이트를 예약할 수 있습니다. 파이프라인에 대한 지원 없이 타이머의 글리치를 완화하는 기술이 존재합니다. 그러나 이는 이 문서의 범위를 벗어납니다.

소프트웨어 흐름도

그림 2에서는 ADC 판독값을 PWM 출력으로 변환하기 위해 애플리케이션에서 수행하는 작업을 보여줍니다.

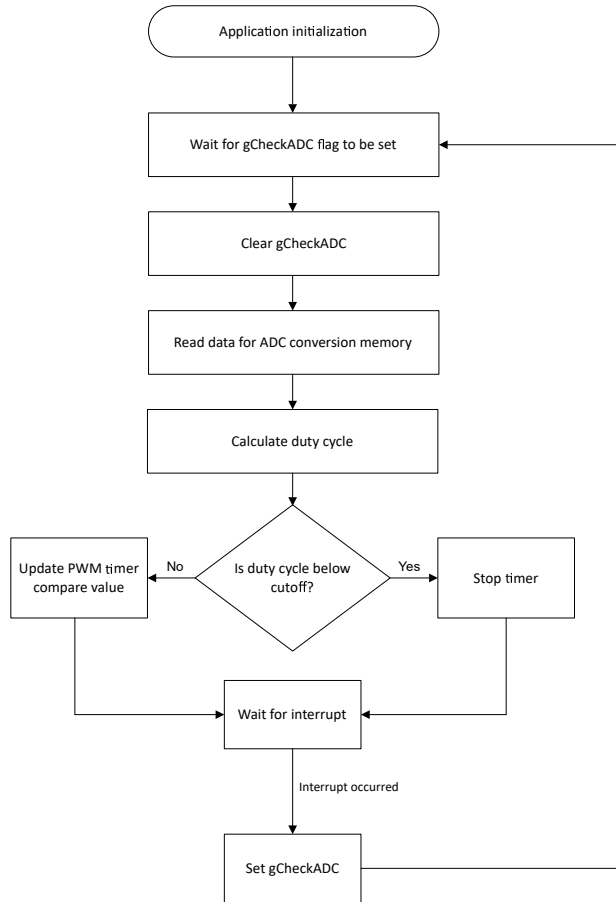


그림 2. 애플리케이션 소프트웨어 흐름도

애플리케이션 코드

이 애플리케이션의 PWM 출력은 10비트의 해상도를 갖습니다. 그러나 ADC 샘플은 12비트이므로 12비트 ADC 판독값을 PWM 타이머의 비교 값을 설정하는 데 사용할 수 있는 10비트 값으로 변환해야 합니다. 애플리케이션 요구 사항에 따라 다른 확장이 필요할 수 있습니다.

또한 수신 데이터의 보다 발전된 신호 처리가 필요할 수 있습니다. 예를 들어 제한, 평균화 또는 기타 필터링은 서로 다른 시나리오에서 중요할 수 있습니다. 이러한 유형의 작업은 아래 함수에서 수행할 수 있습니다.

```

void updatePWMfromADCvalue(uint16_t adcValue) {
    // Check to see if the adc value is above our minimum threshold
    if (adcValue > PWM_DEADBAND)
    {
        // Convert 12bit adcValue into 10bit value by right
        // shifting by 2 because the PWM resolution is 10bit
        uint16_t adcValue_10bit = adcValue >> 2;
        // PWM timer is configured as a down counter (i.e it
        // starts counting down from PWM_LOAD_VAL) and its
        // initial state is high therefore we must perform
        // the following operation so that small values of
        // adcValue_10bit result in small duty cycles
        uint16_t ccv = PWM_LOAD_VAL - adcValue_10bit;
        // Write the new ccv value into the corresponding timer
        // register
        DL_TimerG_setCaptureCompareValue(PWM_0_INST,
                                         CCV,
                                         DL_TIMER_CC_0_INDEX);

        // Start the timer if it is not already running
        if ( !DL_TimerG_isRunning(PWM_0_INST) ) {
            DL_TimerG_startCounter(PWM_0_INST);
        }
    }
    else {
        // If adcResult is not above deadband value then disable timer
        DL_TimerG_stopCounter(PWM_0_INST);
    }
}

```

결과

입력 전압이 프리셋 데드밴드 값보다 낮으면 그림 1-3과 같이 출력이 비활성화됩니다.

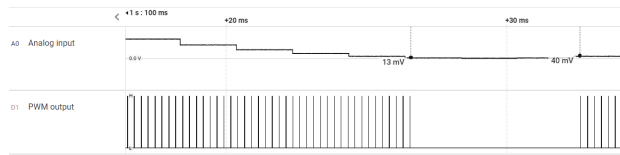


그림 3. ADC 입력이 데드밴드 이하인 경우 PWM 출력이 비활성화됨

그림 1-4에서 입력 전압은 2.264V입니다. 측정된 듀티 사이클은 67.93%입니다. 빠른 계산을 통해 예상 듀티 사이클이 68.4%임을 확인합니다.

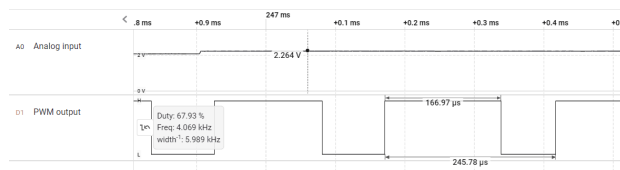


그림 4. PWM 출력 듀티 사이클은 입력 전압에 해당함

추가 리소스

- [MSPM0 SDK 다운로드](#)
- [SysConfig에 대해 자세히 알아보기](#)
- [MSPM0L LaunchPad](#)
- [MSPM0G LaunchPad](#)
- [MSPM0 타이머 아카데미](#)
- [MSPM0 ADC 아카데미](#)

ADC를 이용한 DMA 핑퐁

설명

ADC를 이용한 DMA 핑퐁 예제에서는 서로 다른 두 버퍼 간에 ADC 데이터를 전송하기 위해 DMA를 사용하는 방법을 보여 주며, 이는 DMA 핑퐁이라고도 합니다. CPU가 다른 버퍼와 함께 작동하는 동안 DMA 핑퐁은 일반적으로 하나의 버퍼로 데이터를 전송하는 데 사용됩니다. **그림 5**의 파란색 경로는 DMA가 버퍼 1로 데이터를 전송하고 CPU가 버퍼 2에서 데이터를 가져오는 것을 보여줍니다. 경로가 전환되면 DMA가 버퍼 2로 데이터를 전송하고 CPU가 버퍼 1에서 데이터를 가져옵니다. 이 기술의 이점은 CPU가 항상 데이터 섹션에서 자유롭게 작동할 수 있기 때문에 전체 애플리케이션 런타임이 더 빨라진다는 것입니다. 이 예제에서 ADC는 단일 변환 모드로 구성되며 각 변환 후 버퍼 간에 DMA와 CPU가 전환됩니다.

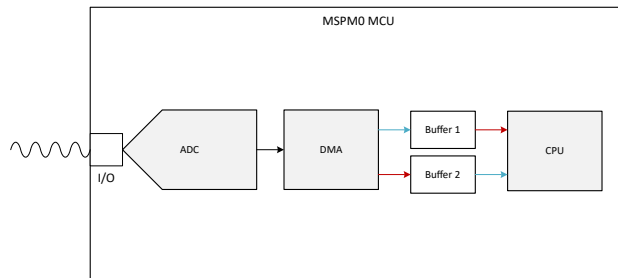


그림 5. 서브시스템 기능 블록 다이어그램

필요한 주변 기기

이 애플리케이션에는 통합 ADC 및 DMA가 필요합니다. 내부 VREF는 다른 레퍼런스 값이 필요할 경우 ADC 레퍼런스에 대한 추가 옵션입니다.

표 3. 필요한 주변 기기

하위 블록 기능	사용되는 주변 기기	참고
아날로그 신호 캡처	ADC	코드에서 ADC12_0_INST라고 부름
메모리 이동	DMA	PREIRQ 기능을 활용하려면 모든 기능을 갖춘 DMA 채널이 필요합니다. 이 예제는 PREIRQ 없이 작동하도록 변경할 수 있습니다.

호환 가능 장치

표 3의 요구 사항을 바탕으로 일부 호환 가능 장치가 표 4에 열거되어 있습니다. 해당 EVM은 빠른 평가를 위해 사용할 수 있습니다. 다른 MSPM0 장치는 필요한 주변 기기가 충족되면 이 서브시스템과 함께 작동합니다. 빠른 포팅을 위해서는 SysConfig에서 장치/전환 옵션을 사용합니다.

표 4. 호환 가능 장치

호환 가능 장치	EVM
MSPM0Cx	LP-MSPM0C1104
MSPM0Lx	LP-MSPM0L1306
MSPM0Gx	LP-MSPM0G3507

설계 단계

1. 지정된 아날로그 입력 및 설계 요구 사항을 기반으로 레퍼런스 소스, 레퍼런스 값, 해상도 및 샘플링 속도를 포함한 ADC에 대한 구성을 결정합니다.
2. ADC 데이터를 저장하는 두 개의 배열 버퍼를 생성하고 버퍼 크기와 DMA 전송 크기를 동일하게 설정하여 DMA가 전체 버퍼를 채우도록 합니다.
3. 단계 1에서 확인한 프로젝트 요구 사항을 바탕으로 SysConfig에서 ADC를 구성합니다.
4. SysConfig에서 ADC 섹션의 DMA를 구성합니다.
5. DMA의 대상 주소를 동적으로 변경하여 버퍼 간에 전환할 수 있도록 애플리케이션 코드를 작성합니다. 그림 6에서 개요를 확인하거나 코드를 직접 살펴봅니다.

설계 고려 사항

1. **최대 샘플링 속도:** ADC의 샘플링 속도는 입력 신호 주파수, 아날로그 프론트 엔드, 필터 또는 샘플링에 영향을 미치는 기타 설계 매개 변수를 기반으로 합니다.
2. **ADC 레퍼런스:** ADC의 최대 눈금 범위를 활용하기 위해 예상 최대 입력과 정렬할 레퍼런스를 선택합니다.
3. **클럭 설정:** 클럭 소스에서 변환의 총 시간을 결정합니다. SCOMP 설정과 함께 사용되는 클럭 분할기에 따라 총 샘플링 시간이 결정됩니다. SysConfig는 샘플링 시간 설정에 따라 적절한 SCOMP를 설정합니다.

소프트웨어 흐름도

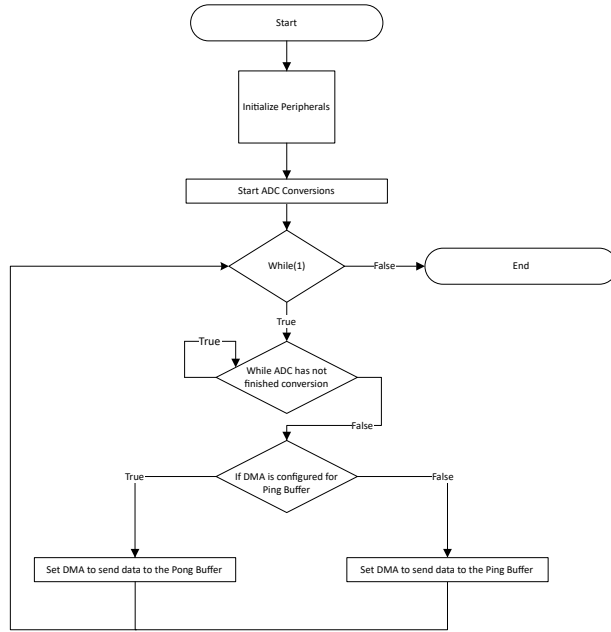


그림 6. 소프트웨어 흐름도

설계 결과

다음 내용에서는 코드 실행 결과를 보여줍니다. **그림 7**에서는 메인 루프의 첫 번째 실행 후 버퍼의 결과를 보여줍니다. 버퍼가 채워지면 코드가 DMA 대상을 두 번째 버퍼로 스왑하며, 이제 CPU가 첫 번째 버퍼의 데이터를 자유롭게 활용할 수 있습니다.

Expression	Type	Value	Address
gADCSamplePing	unsigned short[64]	0x20200000	0x20200000
gADCSamplePing	unsigned short[64]	0x20200000	0x20200000

그림 7. 첫 번째 통과 후 버퍼

그림 8에서는 메인 루프의 두 번째 실행 후 두 번째 버퍼의 결과를 보여줍니다. 버퍼가 채워지면 코드가 DMA 대상을 다시 첫 번째 버퍼로 스왑하고, 이제 CPU가 두 번째 버퍼의 데이터를 사용할 수 있습니다.

Expression	Type	Value	Address
gADCSamplePing	unsigned short[64]	0x20200000	0x20200000
gADCSamplePing	unsigned short[64]	0x20200000	0x20200000

그림 8. 두 번째 통과 후 버퍼

추가 리소스

- 텍사스 인스트루먼트, [MSPM0 SDK 다운로드](#)
- 텍사스 인스트루먼트, [SysConfig에 대해 자세히 알아보기](#)
- 텍사스 인스트루먼트, [MSPM0L LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0G LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0 ADC 아카데미](#)
- 텍사스 인스트루먼트, [MSPM0 DMA 아카데미](#)

E2E

TI의 **E2E™** 지원 포럼 토론을 보고 새로운 스레드를 게시하여 설계에서 MSPM0 장치를 활용하는 데 필요한 기술 지원을 받으세요.

디지털 FIR 필터

설명

이 서브시스템은 MSPM0G 장치 제품군 내에서 내부 ADC, 수학 가속기(MATHACL) 모듈을 사용하여 아날로그 신호의 간단한 스트리밍 FIR 필터를 구현하는 방법을 보여줍니다. 이 구성에서는 소프트웨어 부동 소수점 계산을 기다리지 않고 원하는 필터 차수와 계수를 기반으로 아날로그 신호의 잡음을 필터링할 수 있습니다.

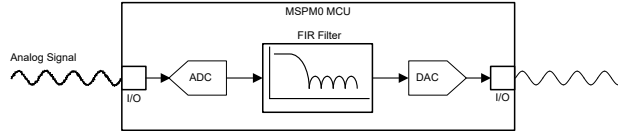


그림 9. FIR 필터 기능 블록 다이어그램

필요한 주변 기기

필요한 주변 기기

이 애플리케이션에는 통합 ADC 1개, MathACL 및 DAC12 모듈이 필요합니다.

표 5. 필요한 주변 기기

하위 블록 기능	사용되는 주변 기기	참고
아날로그 신호 캡처	(1x) ADC	코드에서 <code>ADC12_0_INST</code> 로 표시됨
FIR 필터	(1x) MathACL	코드에서 <code>MATHACL</code> 로 표시됨
아날로그 신호 출력 (옵션)	(1x) DAC12	코드에서 <code>DAC12_0_INST</code> 로 표시됨

호환 가능 장치

표 5에 열거된 요구 사항에 따라 이 예제는 표 6에 열거된 장치와 호환됩니다. 해당 EVM은 프로토타입 제작에 사용할 수 있습니다.

표 6. 호환 가능 장치

호환 가능 장치	EVM
MSPM0G35xx, MSPM0G15xx	LP-MSPM0G3507

설계 단계

- 원하는 코너 주파수와 필터 응답을 결정합니다.
- ADC 샘플링 주파수를 설정합니다. 신호의 예상 대역폭보다 최소한 2배 이상 높아야 합니다.
- 원하는 계수와 필터 차수를 계산합니다. 필터 계수는 샘플링 주파수와 결합하여 필터의 통과 및 제거 대역을 결정하는 유리수입니다.
 - FIR 필터 계수 계산을 위한 다양한 방법 및 툴이 있으며, 이 문서에서는 해당 계산을 다루지 않습니다.
- 필터 계수를 고정점 값으로 변환합니다.
 - 예제 코드에서는 Q16(16개의 분수 비트) 표현이 사용됩니다. **IQMath 라이브러리**를 사용하거나 계수에 2^n 을 곱하여 이 변환을 수행합니다. 여기서 n 은 원하는 분수 비트 수입니다. 선택한 데이터 형식에서 오버플로 없이 이러한 값을 가질 수 있는지 확인합니다.

b. 필터 계수는 상수 값이며 이로 인해 필요할 경우 플래시에 포함시켜 SRAM의 공간을 절약할 수 있습니다.

설계 고려 사항

1. **입력 신호 대역폭:** 분석해야 하는 신호의 대역폭에 따라 ADC 샘플링 주파수 및 코드가 처리해야 하는 데이터의 양이 결정됩니다.
2. **ADC 레퍼런스 전압:** 신호 진폭을 우수한 해상도로 완전히 캡처할 수 있도록 ADC 레퍼런스 전압을 선택해야 합니다.
3. **필터 차수:** 필터 차수가 증가할 때마다 사용자가 샘플별로 수행해야 하는 작업의 모음이 늘어납니다. 이를 통해 샘플 간의 전체 처리 시간이 증가하고 사용자가 수행할 수 있는 다른 프로세스의 양이 제한됩니다. 그 결과 필터 제거가 증가하고 원하는 신호의 해상도가 높아집니다.

소프트웨어 흐름도

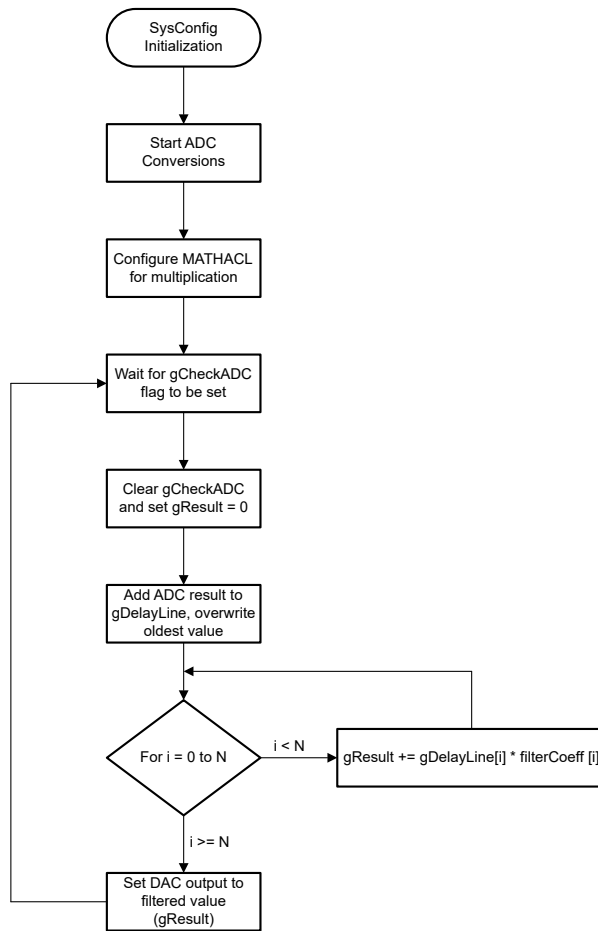


그림 10. 소프트웨어 시퀀스 예제

애플리케이션 코드

```

#define FILTER_ORDER 24
#define FIXED_POINT_PRECISION 16
volatile bool gCheckADC;
uint32_t gDelayLine[FILTER_ORDER];
uint32_t gResult = 0;
/* Filter coefficients are input as 16-bit Precision fixed point values */

```

```

static int32_t filterCoeff[FILTER_ORDER] = {
    -62, -153, -56, 434, 969, 571,
    -1291, -3237, -2173, 3989, 13381, 20518,
    20518, 13381, 3989, -2173, -3237, -1291,
    571, 969, 434, -56, -153, -62
};

const DL_MathACL_operationConfig gMpyConfig = {
    .opType      = DL_MATHACL_OP_TYPE_MAC,
    .opSign     = DL_MATHACL_OPSIGN_SIGNED,
    .iterations = 0,
    .scaleFactor = 0,
    .qType      = DL_MATHACL_Q_TYPE_Q16};

int main(void)
{
    SYSCFG_DL_init();
    NVIC_EnableIRQ(ADC12_0_INST_INT_IRQN);
    gCheckADC = false;
    DL_ADC12_startConversion(ADC12_0_INST);

    /* Configure MathACL for Multiply */
    DL_MathACL_configOperation(MATHACL, &gMpyConfig, 0, 0);

    while (1) {
        while (false == gCheckADC) {
            __WFE();
        }

        gCheckADC = false;
        gResult = 0;
        /* Append the most recent ADC result to the delay line */
        memmove(&gDelayLine[1], gDelayLine, sizeof(gDelayLine) - sizeof(gDelayLine[0]));
        gDelayLine[0] = DL_ADC12_getMemResult(ADC12_0_INST, DL_ADC12_MEM_IDX_0);

        /* Calculate FIR Filter Output */
        for (int i = 0; i < FILTER_ORDER; i++){
            /* Set Operand One last */
            DL_MathACL_setOperandTwo(MATHACL, filterCoeff[i]);
            DL_MathACL_setOperandOne(MATHACL, gDelayLine[i]);
            DL_MathACL_waitForOperation(MATHACL);
        }
        /* Our result should not exceed the bounds of RES1 register, in other applications you may use both
        RES1 and RES2 registers */
        gResult = DL_MathACL_getResultOne(MATHACL);
        DL_DAC12_output12(DAC0, (uint32_t)(gResult));

        /* Clear Results Registers */
        DL_MathACL_clearResults(MATHACL);
    }

    /* Set the ADC Result flag to trigger our main loop to process the new data */
    void ADC12_0_INST_IRQHandler(void)
    {
        switch (DL_ADC12_getPendingInterrupt(ADC12_0_INST)) {
            case DL_ADC12_IIDX_MEM0_RESULT_LOADED:
                gCheckADC = true;
                break;
            default:
                break;
        }
    }
}

```

추가 리소스

- 텍사스 인스트루먼트, [MSPM0 G 시리즈 80MHz 마이크로컨트롤러 기술 레퍼런스 매뉴얼](#), 기술 레퍼런스 매뉴얼.
- 텍사스 인스트루먼트, [CAN-FD 인터페이스를 지원하는 MSPM0G350x 혼합 신호 마이크로컨트롤러](#), 데이터 시트.
- 텍사스 인스트루먼트, [MSPM0G150x 혼합 신호 마이크로컨트롤러](#), 데이터 시트.

E2E

TI의 E2E 지원 포럼에서 토론을 보고 새로운 스레드를 게시하여 설계에서 MSPM0 장치를 활용하는 데 필요한 기술 지원을 받으세요.

ADC-I2C

설명

ADC-I2C 서브시스템 예제에서는 내부 ADC를 사용하여 아날로그 신호를 디지털 표현으로 변환하고 I2C를 통해 결과를 전송하는 방법을 보여줍니다. 이 예제에서는 MCU가 외부 ADC 역할을 하고, I2C 컨트롤러에서 I2C 명령을 수신하고, 수신된 명령을 적절하게 실행하도록 구성합니다. 제공된 간단한 예제 명령으로 사용자는 이를 기반으로 자체 명령을 구현할 수 있습니다. 선택적으로 MCU는 I2C를 통해 데이터를 전송하기 전에 ADC 데이터를 처리할 수 있으며, 이는 원시 데이터를 의미 있는 값으로 처리해야 하는 애플리케이션에서 특히 유용합니다. [여기에서 ADC-I2C 서브시스템의 코드를 다운로드하세요.](#)

아래 그림은 시스템의 블록 다이어그램을 보여줍니다.

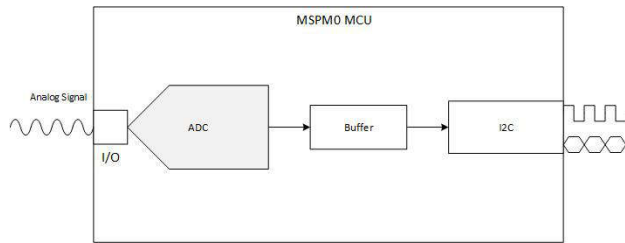


그림 11. 서브시스템 기능 블록 다이어그램

필요한 주변 기기

애플리케이션에는 내부 ADC와 1개의 I2C 인스턴스가 필요합니다.

하위 블록 기능	사용되는 주변 기기	참고
아날로그 신호 캡처	ADC	코드에서 ADC12_0_INST라고 부름
ADC 데이터 전송	I2C	이 장치는 이 예제의 대상임

호환 가능 장치

필요한 주변 기기 테이블의 요구 사항을 바탕으로 일부 호환 가능 장치 및 해당하는 EVM이 아래에 열거되어 있습니다. 다른 MSPM0 장치는 필요한 주변 기기가 있으면 이 서브시스템과 함께 사용할 수 있습니다.

호환 가능 장치	EVM
MSPM0Lxxx	LP-MSPM0L1306
MSPM0Gxxx	LP-MSPM0G3507

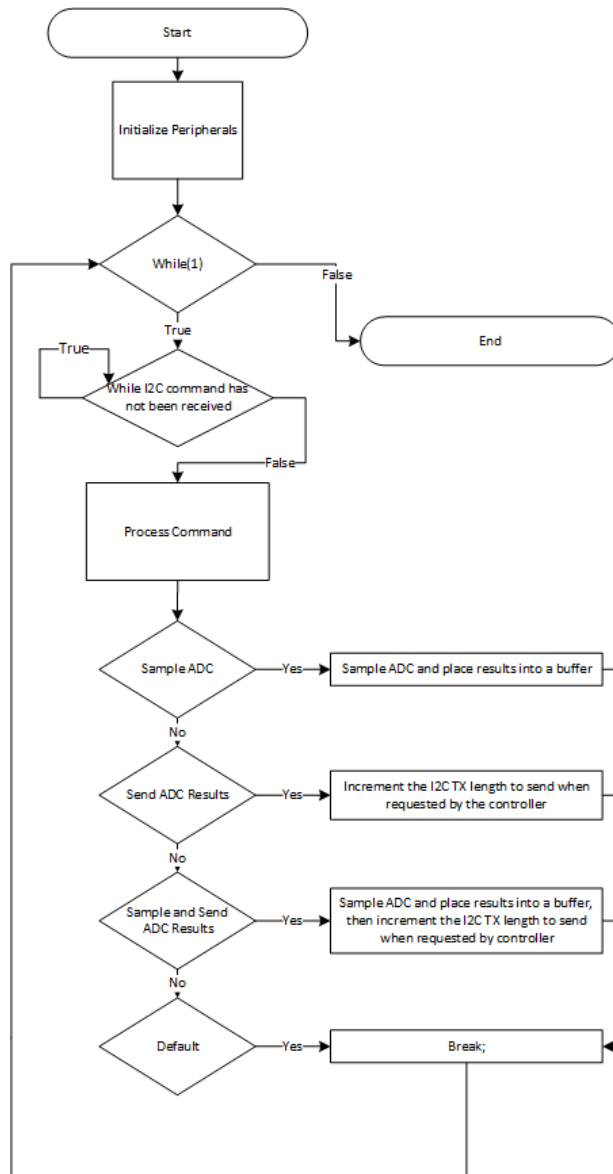
설계 단계

1. 예상되는 아날로그 입력 및 설계 요구 사항을 기반으로 레퍼런스 소스, 레퍼런스 값 및 샘플링 속도를 포함한 ADC에 대한 구성을 결정합니다.
2. 이전 단계의 요구 사항에 따라 SysConfig에서 ADC를 구성합니다.
3. SysConfig에서 I2C 주변 기기를 구성하고 대상 모드에서 I2C를 설정합니다.
4. 메모리 레지스터에서 I2C TX FIFO로 ADC 데이터를 전송하는 애플리케이션 코드를 작성합니다. 소프트웨어 흐름도에 서 개요를 확인하거나 코드를 직접 살펴봅니다.

설계 고려 사항

1. 최대 샘플링 속도: ADC의 샘플링 속도는 입력 신호 주파수, 아날로그 프런트 엔드, 필터 또는 샘플링에 영향을 미치는 기타 설계 매개 변수를 기반으로 합니다.
2. ADC 레퍼런스: ADC의 최대 눈금 범위를 활용하기 위해 예상 최대 입력과 정렬할 레퍼런스를 선택합니다.
3. 클럭 설정: 클럭 소스에서 샘플 및 변환 시간의 총 시간을 결정했습니다. SCOMP 설정과 함께 사용되는 클럭 분할기에 따라 총 샘플링 시간이 결정됩니다. SysConfig는 샘플링 시간 설정에 따라 적절한 SCOMP를 설정합니다.
4. I2C 주소, 주소 지정 모드, 글리치 필터, 클럭 스트레칭 등과 같은 컨트롤러 요구 사항에 따라 I2C 구성을 조정할 수 있습니다.

소프트웨어 흐름도



추가 리소스

- [MSPM0 SDK 다운로드](#)
- [SysConfig에 대해 자세히 알아보기](#)
- [MSPM0L1306](#)
- [MSPM0G3507](#)
- [MSPM0 ADC 아카데미](#)
- [MSPM0 I2C 아카데미](#)

디지털 IIR 필터

설명

이 서브시스템은 MSPM0G 장치 제품군 내에서 내부 ADC, 수학 가속기(MATHACL) 모듈을 사용하여 아날로그 신호의 간단한 스트리밍 IIR 필터를 구현하는 방법을 보여줍니다. 이 구성에서는 단극 IIR 필터를 사용하여 아날로그 신호의 잡음이 필터링됩니다. 정의된 베타 값은 주파수에 대한 IIR 필터 감쇠를 제어하기 위해 조정할 수 있습니다.

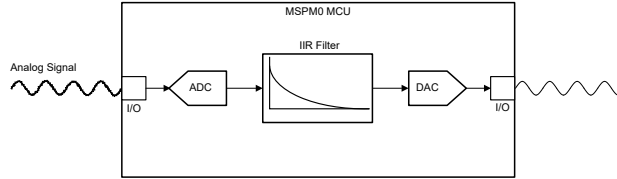


그림 12. IIR 필터 기능 블록 다이어그램

필요한 주변 기기

필요한 주변 기기

이 애플리케이션에는 통합 ADC 1개, MathACL 및 DAC12 모듈이 필요합니다.

표 7. 필요한 주변 기기

하위 블록 기능	사용되는 주변 기기	참고
아날로그 신호 캡처	(1x) ADC	코드에서 <code>ADC12_0_INST</code> 로 표시됨
IIR 필터	(1x) MathACL	코드에서 <code>MATHACL</code> 로 표시됨
아날로그 신호 출력(옵션)	(1x) DAC12	코드에서 <code>DAC12_0_INST</code> 로 표시됨

호환 가능 장치

표 7의 요구 사항에 따라 이 예제는 표 8에 열거된 장치와 호환됩니다. 해당 EVM은 프로토타입 제작에 사용할 수 있습니다.

표 8. 호환 가능 장치

호환 가능 장치	EVM
MSPM0G35xx, MSPM0G15xx	LP-MSPM0G3507

설계 단계

- 필요한 최소 ADC 샘플링 주파수를 결정합니다. 입력 신호의 대역폭보다 최소한 2배 이상 높아야 합니다.
- 원하는 제거 계수를 결정합니다. 단극 IIR 필터의 제거 계수는 주파수에 대한 필터 감쇠 속도를 제어합니다. 제거 계수는 베타(β) 값 또는 감쇠 값이라고도 합니다.
 - IIR 필터 계수 계산을 위한 다양한 툴이 있으며, 이 문서에서는 해당 계산을 다루지 않습니다.
- 필터 계수를 고정점 값으로 변환합니다.
 - 예제 코드에서는 Q8(8개의 분수 비트) 표현이 사용됩니다. **IQMath 라이브러리**를 사용하거나 계수에 2^n 을 곱하여 이 변환을 수행합니다. 여기서 n 은 원하는 분수 비트 수입니다. 선택한 데이터 형식에서 오버플로 없이 이러한 값을 가질 수 있는지 확인합니다.
 - 필터 계수는 상수 값이며 필요할 경우 플래시에 포함시켜 SRAM의 공간을 절약할 수 있습니다.

설계 고려 사항

1. 입력 신호 대역폭:

분석해야 하는 신호의 대역폭에 따라 ADC 샘플링 주파수 및 코드가 처리해야 하는 데이터의 양이 결정됩니다.

2. ADC 레퍼런스 전압:

신호 진폭을 우수한 해상도로 완전히 캡처할 수 있도록 ADC 레퍼런스 전압을 선택해야 합니다.

3. 감쇠 계수:

단극 IIR 필터에서 감쇠 값은 현재 결과에 대한 새로운 샘플의 기여도를 가중시키는 단일 계수입니다. 감쇠 계수의 크기는 0과 1 사이입니다. 감쇠 값이 높을수록 차단 주파수가 더 낮습니다.

소프트웨어 흐름도

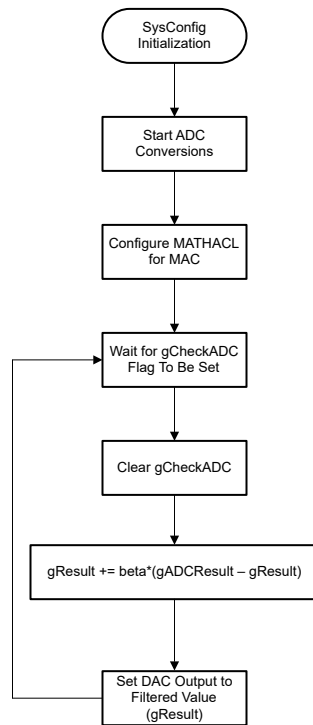


그림 13. 소프트웨어 시퀀스 예제

애플리케이션 코드

```

volatile bool gCheckADC;
/* Filtered Result */
uint32_t gResult = 0;
/* ADC Value Output */
uint32_t gADCResult = 0;

/* Scaling Factor, Q8 value (0-255) */
uint32_t gBeta = 16;
const DL_MathACL_operationConfig gMpyConfig = {
    .opType      = DL_MATHACL_OP_TYPE_MAC,
    .opSign     = DL_MATHACL_OP_SIGN_SIGNED,
    .iterations  = 0,
    .scaleFactor = 0,
    .qType      = DL_MATHACL_Q_TYPE_Q8};
int main(void)
{
    SYSCFG_DL_init();
  
```

```

NVIC_EnableIRQ(ADC12_0_INST_INT_IRQN);
gCheckADC = false;
DL_ADC12_startConversion(ADC12_0_INST);

/* Configure MathACL for Multiply and Accumulate */
DL_MathACL_configOperation(MATHACL, &gMpyConfig, 0, 0);
DL_MathACL_enableSaturation(MATHACL);

while (1) {
    while (false == gcheckADC) {
        __WFE();
    }
    gCheckADC = false;

    /* Calculate IIR Filter Output */
    gADCResult = DL_ADC12_getMemResult(ADC12_0_INST, DL_ADC12_MEM_IDX_0);
    /* Set Operand One last */
    DL_MathACL_setOperandTwo(MATHACL, gADCResult - gResult);
    DL_MathACL_setOperandOne(MATHACL, gBeta);
    DL_MathACL_waitForOperation(MATHACL);
    gResult = DL_MathACL_getResultOne(MATHACL);
    DL_DAC12_output12(DAC0, gResult);

}
}
/* Set the ADC Result flag to trigger our main loop to process the new data */
void ADC12_0_INST_IRQHandler(void)
{
    switch (DL_ADC12_getPendingInterrupt(ADC12_0_INST)) {
        case DL_ADC12_IIDX_MEM0_RESULT_LOADED:
            gCheckADC = true;
            break;
        default:
            break;
    }
}
}

```

추가 리소스

- 텍사스 인스트루먼트, [MSPM0 G 시리즈 80MHz 마이크로컨트롤러](#), 기술 레퍼런스 매뉴얼.
- 텍사스 인스트루먼트, [MSPM0 L 시리즈 32MHz 마이크로컨트롤러](#), 기술 레퍼런스 매뉴얼.
- 텍사스 인스트루먼트, [CAN-FD 인터페이스를 지원하는 MSPM0G350x 혼합 신호 마이크로컨트롤러](#), 데이터 시트.
- 텍사스 인스트루먼트, [MSPM0G150x 혼합 신호 마이크로컨트롤러](#), 데이터 시트.
- 텍사스 인스트루먼트, [MSPM0L130x 혼합 신호 마이크로컨트롤러](#), 데이터 시트.

E2E

TI의 E2E 지원 포럼에서 토론을 보고 새로운 스레드를 게시하여 설계에서 MSPM0 장치를 활용하는 데 필요한 기술 지원을 받으세요.

ADC-SPI

설명

ADC-SPI 서브시스템 예제에서는 내부 ADC를 사용하여 아날로그 신호를 디지털 표현으로 변환하고 SPI를 통해 결과를 전송하는 방법을 보여줍니다. 이 예제에서는 MCU가 외부 ADC 역할을 하고, SPI 컨트롤러에서 SPI 명령을 수신하고, 수신된 명령을 적절하게 실행하도록 구성합니다. 제공된 간단한 예제 명령으로 사용자는 이를 기반으로 자체 명령을 구현할 수 있습니다. 선택적으로 MCU는 SPI를 통해 데이터를 전송하기 전에 ADC 데이터를 처리할 수 있으며, 이는 원시 데이터를 유의미한 값으로 변환해야 하는 애플리케이션에서 특히 유용합니다. [ADC-SPI 예제의 코드를 다운로드하세요.](#)

아래 그림은 시스템의 블록 다이어그램을 보여줍니다.

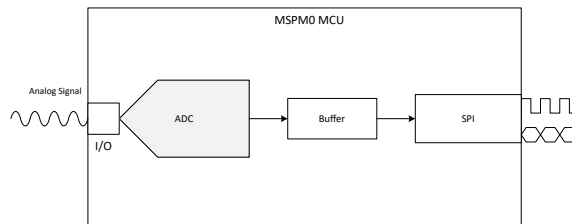


그림 14. 서브시스템 기능 블록 다이어그램

필요한 주변 기기

애플리케이션에는 내부 ADC와 1개의 SPI 인스턴스가 필요합니다.

하위 블록 기능	사용되는 주변 기기	참고
아날로그 신호 캡처	ADC	코드에서 ADC12_0_INST라고 부름
ADC 데이터 전송	SPI	이 장치는 이 예제의 주변 기기임

호환 가능 장치

[필요한 주변 기기 테이블](#)의 요구 사항을 바탕으로 일부 호환 가능 장치 및 해당하는 EVM이 아래에 열거되어 있습니다. 다른 MSPM0 장치는 필요한 주변 기기가 있으면 이 서브시스템과 함께 사용할 수 있습니다.

호환 가능 장치	EVM
MSPM0Lxxx	LP-MSPM0L1306
MSPM0Gxxx	LP-MSPM0G3507

설계 단계

1. 예상되는 아날로그 입력 및 설계 요구 사항을 기반으로 레퍼런스 소스, 레퍼런스 값 및 샘플링 속도를 포함한 ADC에 대한 구성을 결정합니다.
2. 이전 단계의 요구 사항에 따라 SysConfig에서 ADC를 구성합니다.
3. SysConfig에서 SPI 주변 기기를 구성하고 주변 기기 모드에서 SPI를 설정합니다.
4. SPI를 통해 전송하기 위해 메모리 레지스터에서 ADC 데이터를 전송하는 애플리케이션 코드를 작성합니다. 필요에 따라 다른 작업을 수행하기 위한 명령을 추가합니다. 소프트웨어 흐름도에서 개요를 확인하거나 코드를 직접 살펴봅니다.

소프트웨어 흐름도

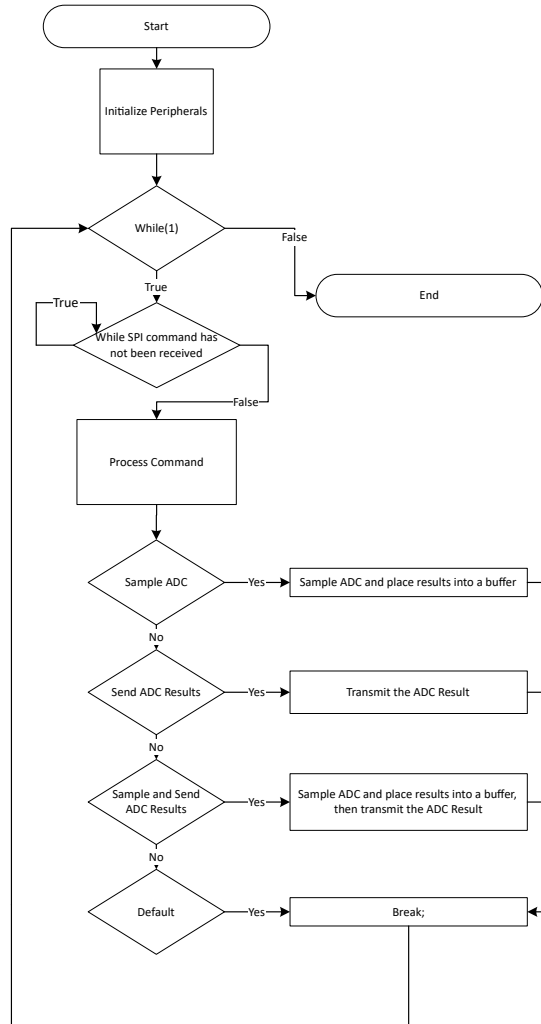


그림 15. 애플리케이션 소프트웨어 흐름도

추가 리소스

- [MSPM0 SDK 다운로드](#)
- [SysConfig에 대해 자세히 알아보기](#)
- [MSPM0L1306](#)
- [MSPM0G3507](#)
- [MSPM0 ADC 아카데미](#)
- [MSPM0 SPI 아카데미](#)

ADC-UART

설명

ADC-UART 서브시스템 예제에서는 내부 ADC를 사용하여 아날로그 신호를 디지털 표현으로 변환하고 UART를 통해 결과를 전송하는 방법을 보여줍니다. 이 예제에서는 MCU가 외부 ADC 역할을 하고 UART를 통해 원시 ADC 데이터를 전송하도록 구성합니다. 선택적으로 MCU가 데이터를 사전 처리한 후 I2C를 통해 전송할 수도 있습니다. [ADC-UART 예제의 코드를 다운로드하세요.](#)

아래 그림은 시스템의 블록 다이어그램을 보여줍니다.

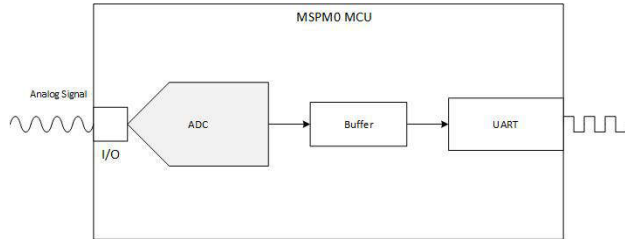


그림 16. 서브시스템 기능 블록 다이어그램

필요한 주변 기기

애플리케이션에는 내부 ADC와 1개의 UART 인스턴스가 필요합니다.

하위 블록 기능	사용되는 주변 기기	참고
아날로그 신호 캡처	ADC	코드에서 ADC12_0_INST라고 부름
ADC 데이터 전송	UART	2개의 UART 트랜잭션이 수행되어 전체 ADC 데이터를 전송합니다.

호환 가능 장치

상기 표의 요구 사항을 바탕으로 호환 가능 장치가 아래에 열거되어 있습니다. 해당 EVM은 빠른 평가를 위해 사용할 수 있습니다.

호환 가능 장치	EVM
MSPM0Lxxx	LP-MSPM0L1306
MSPM0Gxxx	LP-MSPM0G3507

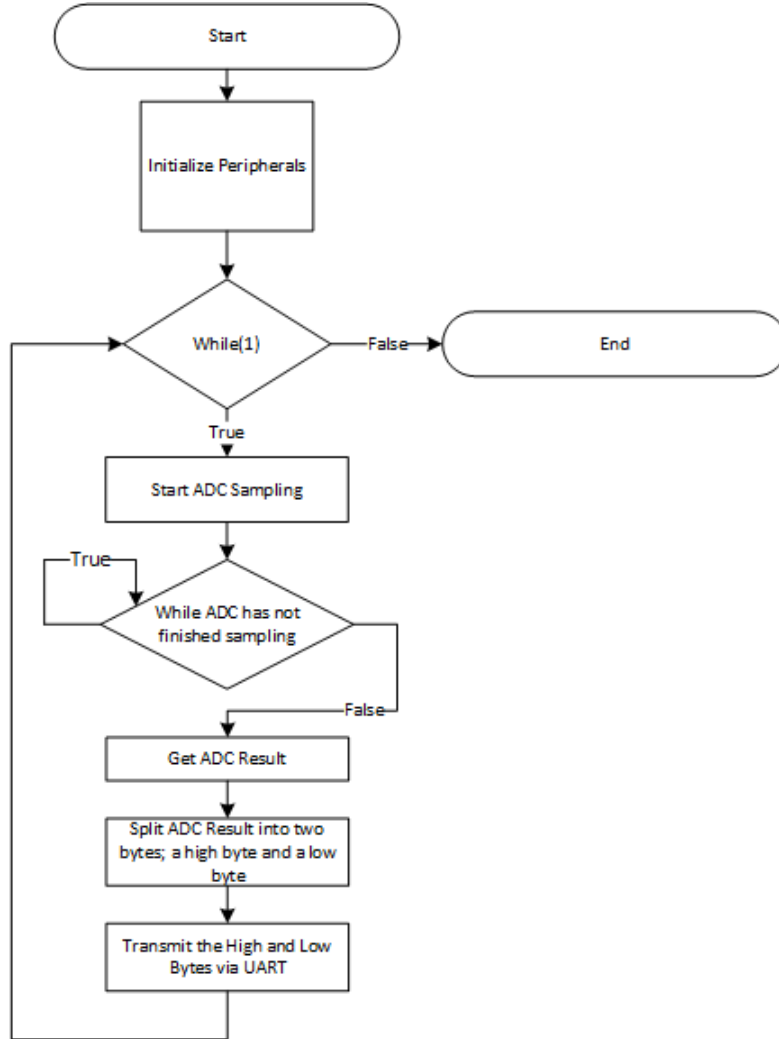
설계 단계

1. 예상되는 아날로그 입력 및 설계 요구 사항을 기반으로 레퍼런스 소스, 레퍼런스 값 및 샘플링 속도를 포함한 ADC에 대한 구성을 결정합니다.
2. 이전 단계의 요구 사항에 따라 SysConfig에서 ADC를 구성합니다.
3. SysConfig에서 UART 주변 기기를 구성하고, 의도한 통신을 위해 UART를 의도한 보 레이트 및 기타 UART 옵션으로 설정합니다.
4. 메모리 레지스터에서 UART로 ADC 데이터를 전송하는 애플리케이션 코드를 작성합니다. [소프트웨어 흐름도](#)에서 개요를 확인하거나 코드를 직접 살펴봅니다.

설계 고려 사항

1. 최대 샘플링 속도: ADC의 샘플링 속도는 입력 신호 주파수, 아날로그 프론트 엔드, 필터 또는 샘플링에 영향을 미치는 기타 설계 매개 변수를 기반으로 합니다.
2. ADC 레퍼런스: ADC의 최대 눈금 범위를 활용하기 위해 예상 최대 입력과 정렬할 레퍼런스를 선택합니다.
3. 클럭 설정: 클럭 소스에서 샘플 및 변환 시간의 총 시간을 결정합니다. SCOMP 설정과 함께 사용되는 클럭 분할기에 따라 총 샘플링 시간이 결정됩니다. SysConfig는 샘플링 시간 설정에 따라 적절한 SCOMP를 설정합니다.
4. 패리티, 보 레이트 등과 같은 UART 시스템에 따라 UART 구성을 조정할 수 있습니다.

소프트웨어 흐름도



애플리케이션 코드

UART 주변 기기는 한 번에 8비트 패킷으로 데이터를 전송합니다. ADC 모듈은 데이터를 16비트 레지스터에 저장합니다. UART 주변 기기를 통해 데이터를 전송하려면 ADC 데이터를 높은 바이트와 낮은 바이트로 분할해야 합니다. 높은 바이트

에는 상위 8비트가 포함되고 낮은 바이트에는 하위 8비트가 포함됩니다. 아래는 ADC 결과를 분할하고 UART를 통해 데이터를 전송하는 코드입니다.

```
gADCResult = DL_ADC12_getMemResult(ADC12_0_INST, DL_ADC12_MEM_IDX_0);
uint8_t lowbyte = (uint8_t)(gADCResult & 0xFF);
uint8_t highbyte = (uint8_t)((gADCResult >> 8) & 0xFF);
DL_UART_Main_transmitData(UART_0_INST, highbyte);
DL_UART_Main_transmitData(UART_0_INST, lowbyte);
```

추가 리소스

- [MSPM0 SDK 다운로드](#)
- [SysConfig에 대해 자세히 알아보기](#)
- [MSPM0L1306](#)
- [MSPM0G3507](#)
- [MSPM0 ADC 아카데미](#)
- [MSPM0 UART 아카데미](#)

데이터 센서 애그리게이터 서브시스템 설계

설계 설명

이 서브시스템은 BP-BASSENSORSMKII BoosterPack™ 플러그 인 모듈을 위한 인터페이스 역할을 합니다. 이 모듈은 온도 및 습도 센서, 홀 효과 센서, 주변광 센서, 관성 측정 유닛 및 자력계를 갖추고 있습니다. 이 모듈은 TI LaunchPad™ 개발 키트와 상호 작용하도록 설계되었습니다. 이 서브시스템은 I2C 인터페이스를 사용하여 이러한 센서에서 데이터를 수집하고 UART 인터페이스를 사용하여 데이터를 전송합니다. 이를 통해 사용자가 신속하게 프로토타입을 제작하고 MSPM0 및 BASSENSORSMKII BoosterPack 모듈로 실험할 수 있습니다.

MSPM0은 I2C 인터페이스를 사용하여 BP-BASSENSORSMKII에 연결됩니다. MSPM0은 UART 인터페이스를 사용하여 처리된 데이터를 전달합니다.

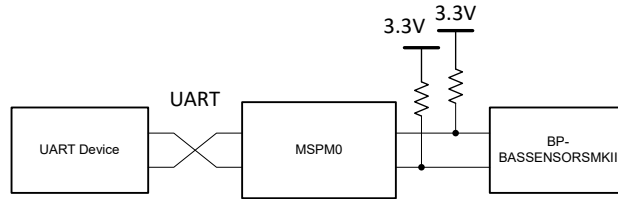


그림 17. 시스템 기능 블록 다이어그램

필요한 주변 기기

사용되는 주변 기기	참고
I2C	코드에서 I2C_INST라고 부름
UART	코드에서 UART_0_INST라고 부름
DMA	UART TX에 사용됨
GPIO	5개의 GPIO는 다음과 같이 지칭됩니다. HDC_V, DRV_V, OPT_V, INT1, INT2
ADC	코드에서 ADC12_0_INST라고 부름
이벤트	UART TX FIFO로 데이터를 전송하는 데 사용됨

호환 가능 장치

필요한 주변 기기의 요구 사항에 따라 이 예제는 다음 표에 표시된 장치와 호환됩니다. 해당 EVM은 프로토타입 제작에 사용할 수 있습니다.

호환 가능 장치	EVM
MSPM0Lxxxx	LP-MSPM0L1306
MSPM0Gxxxx	LP-MSPM0G3507

설계 단계

1. SysConfig에서 GPIO 모듈을 설정합니다. 이름이 HDC_V인 GPIO를 PB24의 출력으로 추가합니다. 이름이 DRV_V인 두 번째 GPIO를 PA22의 출력으로 추가합니다. 이름이 OPT_V인 세 번째 GPIO를 PA24의 출력으로 추가합니다. 이름이 INT1인 네 번째 GPIO를 PA26의 출력으로 추가합니다. 이름이 INT2인 다섯 번째인 마지막 GPIO를 PB6의 출력으로 추가합니다.
2. SysConfig에서 ADC12 모듈을 설정합니다. 자동 샘플링 모드에서 주소 0부터 시작하는 단일 변환 모드를 사용하여 인스턴스를 추가합니다. 트리거 소스를 소프트웨어로 설정합니다. ADC 변환 메모리 구성 탭을 열고 메모리 0의 이름이 0

이고 PA25의 채널 2를 사용하며 VDDA가 레퍼런스 전압으로, 샘플링 타이머 0이 샘플 기간 소스로 지정되어 있는지 확인합니다. 인터럽트 구성 탭에서 로드된 MEM0 결과에 대한 인터럽트를 활성화합니다.

3. SysConfig에서 I2C 모듈을 설정합니다. 컨트롤러 모드를 활성화하고 버스 속도를 100kHz로 설정합니다. 인터럽트 구성 탭에서 RX 완료, TX 완료, RX FIFO 트리거 및 주소/데이터 NACK 인터럽트를 활성화합니다. PinMux 섹션에서 I2C1이 선택한 주변 기기인지, PB3에 SDA가, PB2에 SCL이 지정되어 있는지 확인합니다.
4. SysConfig에서 UART 모듈을 설정합니다. UART 인스턴스를 추가하고 9600Hz 보 레이트를 사용합니다. 인터럽트 구성 탭에서 전송 중 DMA 완료 및 전송 종료 인터럽트를 활성화합니다. DMA 구성 탭에서 UART TX 인터럽트로 DMA TX 트리거를 선택하고 활성화합니다. DMA 채널 TX 설정이 고정 주소 모드에 대한 블록을 사용하고 소스 및 대상 길이가 바이트로 설정되어 있는지 확인합니다. 소스 주소 방향이 증분되도록 설정하고 전송 모드를 단일로 설정합니다. 소스 및 대상 주소 증분은 모두 "각 전송 후 주소를 변경하지 않음"으로 설정해야 합니다. PinMux 섹션에서 RX에 대해 UART0 및 PA11을, TX에 대해 PA10을 선택합니다.

설계 고려 사항

1. 서브시스템 용도에 맞게 코드 시작 부분의 최대 패킷 크기 정의를 살펴보고 올바른지 확인합니다.
2. 사용 중인 I2C 모듈에 적합한 풀업 저항 값을 선택합니다. 일반적으로 10kΩ은 100kHz에 적합합니다. I2C 버스 속도 높을수록 더 낮은 값의 풀업 저항이 필요합니다. 400kHz 통신의 경우 4.7kΩ에 더 가까운 저항을 사용합니다.
3. UART의 보 레이트를 높이려면 SysConfig에서 UART 모듈을 열고 대상 보 레이트 값을 편집합니다. 계산된 실제 보 레이트와 계산된 오류가 표시됩니다.
4. 여기에 오류 감지 및 처리를 추가하여 보다 강력한 애플리케이션을 만들 수 있도록 지원하기 위해 많은 모듈에 오류 인터럽트가 있어 오류 사례를 쉽게 모니터링할 수 있습니다.
5. UART를 통해 데이터가 전송되는 형식을 편집하려면 "전송" 기능을 참조하세요.

소프트웨어 흐름도

다음 흐름도에서는 센서 BoosterPack 플러그 인 모듈에서 데이터를 판독, 수집, 처리, 전송하는 데 수행되는 소프트웨어 단계에 대한 개략적인 개요를 보여줍니다.

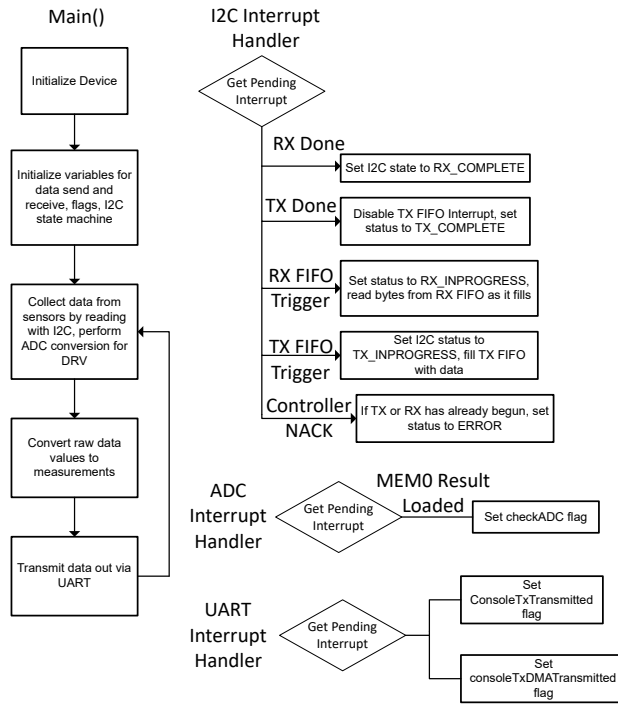


그림 18. 애플리케이션 소프트웨어 흐름도

장치 구성

이 애플리케이션은 TI 시스템 구성 툴(**SysConfig**) 그래픽 인터페이스를 사용하여 장치 주변 기기의 구성 코드를 생성합니다. 그래픽 인터페이스를 사용하여 장치 주변 기기를 구성하면 애플리케이션 프로토타입 제작 프로세스를 간소화할 수 있습니다.

소프트웨어 흐름도에 설명된 코드는 *data_sensor_aggregator.c* 파일의 main() 시작 부분에서 확인할 수 있습니다.

애플리케이션 코드

이 애플리케이션은 UART 및 I2C 전송 크기를 설정한 후 전송할 값을 저장할 메모리를 할당하여 시작합니다. 그런 다음, UART를 통해 전송할 때 최종 사후 처리 측정을 저장할 메모리를 할당합니다. 또한 I2C 컨트롤러 상태를 기록하기 위한 열거형을 정의합니다. 자체 구현에서 일부 패킷 크기를 조정하고 일부 데이터 스토리지를 변경할 수 있습니다. 또한 일부 애플리케이션에서는 오류 처리를 추가하는 것이 좋습니다.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "ti_msp_dl_config.h"

/* Initializing functions */

void DataCollection(void);
void TxFunction(void);
void RxFunction(void);
void Transmit(void);
void UART_Console_write(const uint8_t *data, uint16_t size);

/* Earth's gravity in m/s^2 */
#define GRAVITY_EARTH (9.80665f)

/* Maximum size of TX packet */
#define I2C_TX_MAX_PACKET_SIZE (16)
    
```

```

/* Number of bytes to send to target device */
#define I2C_TX_PACKET_SIZE (3)

/* Maximum size of RX packet */
#define I2C_RX_MAX_PACKET_SIZE (16)

/* Number of bytes to received from target */
#define I2C_RX_PACKET_SIZE (16)

/*
 * Number of bytes for UART packet size
 * The packet will be transmitted by the UART.
 * This example uses FIFOs with polling, and the maximum FIFO size is 4.
 * Refer to interrupt examples to handle larger packets.
 */
#define UART_PACKET_SIZE (8)

uint8_t gSpace[] = "\r\n";
volatile bool gConsoleTxTransmitted;
volatile bool gConsoleTxDMATransmitted;
/* Data for UART to transmit */
uint8_t gTxData[UART_PACKET_SIZE];

/* Booleans for interrupts */
bool gCheckADC;
bool gDataReceived;

/* Variable to change the target address */
uint8_t gTargetAdd;

/* I2C variables for data collection */
float gHumidity, gTempHDC, gAmbient;
uint16_t gAmbientE, gAmbientR, gDRV;
uint16_t gMagX, gMagY, gMagZ, gGyrX, gGyrY, gGyrZ, gAccX, gAccY, gAccZ;

/* Data sent to the Target */
uint8_t gTxPacket[I2C_TX_MAX_PACKET_SIZE];

/* Counters for TX length and bytes sent */
uint32_t gTxLen, gTxCount;

/* Data received from Target */
uint8_t gRxPacket[I2C_RX_MAX_PACKET_SIZE];

/* Counters for TX length and bytes sent */
uint32_t gRxLen, gRxCount;

/* Indicates status of I2C */
enum I2cControllerStatus {
    I2C_STATUS_IDLE = 0,
    I2C_STATUS_TX_STARTED,
    I2C_STATUS_TX_INPROGRESS,
    I2C_STATUS_TX_COMPLETE,
    I2C_STATUS_RX_STARTED,
    I2C_STATUS_RX_INPROGRESS,
    I2C_STATUS_RX_COMPLETE,
    I2C_STATUS_ERROR,
} gI2cControllerStatus;

```

이 애플리케이션의 main()에서 모든 주변 기기 모듈을 초기화한 후 메인 루프에서 장치가 센서의 모든 데이터를 수집하고 처리 후 전송합니다.

```

int main(void)
{
    SYSCFG_DL_init();

    NVIC_EnableIRQ(I2C_INST_INT_IRQN);
    NVIC_EnableIRQ(ADC12_0_INST_INT_IRQN);
    NVIC_EnableIRQ(UART_0_INST_INT_IRQN);
    DL_SYSCFG_disableSleepOnExit();

    while(1) {

```

```

    DataCollection();
    Transmit();
    /* This delay is to the data is transmitted every few seconds */
    delay_cycles(100000000);
}
}

```

다음 코드 블록에는 모든 인터럽트 서비스 루틴이 포함되어 있습니다. 첫 번째는 I2C 루틴이고, 다음은 ADC 루틴이고, 마지막으로 UART 루틴입니다. I2C 루틴은 주로 일부 플래그를 업데이트하고 컨트롤러 상태 변수를 업데이트하는 역할을 합니다. 또한 TX 및 RX FIFO를 관리합니다. ADC 인터럽트 서비스 루틴은 메인 루프가 ADC 값이 유효한지 확인할 수 있도록 플래그를 설정합니다. UART 인터럽트 서비스 루틴은 UART 데이터의 유효성을 확인하는 플래그도 설정합니다.

```

void I2C_INST_IRQHandler(void)
{
    switch (DL_I2C_getPendingInterrupt(I2C_INST)) {
        case DL_I2C_IIDX_CONTROLLER_RX_DONE:
            gI2CControllerStatus = I2C_STATUS_RX_COMPLETE;
            break;
        case DL_I2C_IIDX_CONTROLLER_TX_DONE:
            DL_I2C_disableInterrupt(
                I2C_INST, DL_I2C_INTERRUPT_CONTROLLER_TXFIFO_TRIGGER);
            gI2CControllerStatus = I2C_STATUS_TX_COMPLETE;
            break;
        case DL_I2C_IIDX_CONTROLLER_RXFIFO_TRIGGER:
            gI2CControllerStatus = I2C_STATUS_RX_INPROGRESS;
            /* Receive all bytes from target */
            while (DL_I2C_isControllerRXFIFOEmpty(I2C_INST) != true) {
                if (gRxCount < gRxLen) {
                    gRxPacket[gRxCount++] =
                        DL_I2C_receiveControllerData(I2C_INST);
                } else {
                    /* Ignore and remove from FIFO if the buffer is full */
                    DL_I2C_receiveControllerData(I2C_INST);
                }
            }
            break;
        case DL_I2C_IIDX_CONTROLLER_TXFIFO_TRIGGER:
            gI2CControllerStatus = I2C_STATUS_TX_INPROGRESS;
            /* Fill TX FIFO with next bytes to send */
            if (gTxCount < gTxLen) {
                gTxCount += DL_I2C_fillControllerTXFIFO(
                    I2C_INST, &gTxPacket[gTxCount], gTxLen - gTxCount);
            }
            break;
        /* Not used for this example */
        case DL_I2C_IIDX_CONTROLLER_ARBITRATION_LOST:
        case DL_I2C_IIDX_CONTROLLER_NACK:
            if ((gI2CControllerStatus == I2C_STATUS_RX_STARTED) ||
                (gI2CControllerStatus == I2C_STATUS_TX_STARTED)) {
                /* NACK interrupt if I2C Target is disconnected */
                gI2CControllerStatus = I2C_STATUS_ERROR;
            }
        case DL_I2C_IIDX_CONTROLLER_RXFIFO_FULL:
        case DL_I2C_IIDX_CONTROLLER_TXFIFO_EMPTY:
        case DL_I2C_IIDX_CONTROLLER_START:
        case DL_I2C_IIDX_CONTROLLER_STOP:
        case DL_I2C_IIDX_CONTROLLER_EVENT1_DMA_DONE:
        case DL_I2C_IIDX_CONTROLLER_EVENT2_DMA_DONE:
        default:
            break;
    }
}

void ADC12_0_INST_IRQHandler(void)
{
    switch (DL_ADC12_getPendingInterrupt(ADC12_0_INST)) {
        case DL_ADC12_IIDX_MEM0_RESULT_LOADED:
            gCheckADC = true;
            break;
        default:
            break;
    }
}

```

```

}
void UART_0_INST_IRQHandler(void)
{
    switch (DL_UART_Main_getPendingInterrupt(UART_0_INST)) {
        case DL_UART_MAIN_IIDX_EOT_DONE:
            gConsoleTxTransmitted = true;
            break;
        case DL_UART_MAIN_IIDX_DMA_DONE_TX:
            gConsoleTxDMATransmitted = true;
            break;
        default:
            break;
    }
}
}

```

이 블록은 UART 인터페이스를 사용하여 전송하기 위한 데이터를 포맷합니다. 데이터를 UART 터미널과 같은 장치에서 보도록 쉽게 읽을 수 있는 형식으로 전달합니다. 자체 구현에서는 전송되는 데이터의 형식을 변경할 수 있습니다.

```

/* This function formats and transmits all of the collected data over UART */
void Transmit(void)
{
    int count = 1;
    char buffer[20];
    while (count < 14)
    {
        /* Formatting the name and converting int to string for transfer */
        switch(count){
            case 1:
                gTxData[0] = 84;
                gTxData[1] = 67;
                gTxData[2] = 58;
                gTxData[3] = 32;
                sprintf(buffer, "%f", gTempHDC);
                break;
            case 2:
                gTxData[0] = 72;
                gTxData[1] = 37;
                gTxData[2] = 58;
                gTxData[3] = 32;
                sprintf(buffer, "%f", gHumidity);
                break;
            case 3:
                gTxData[0] = 65;
                gTxData[1] = 109;
                gTxData[2] = 58;
                gTxData[3] = 32;
                sprintf(buffer, "%f", gAmbient);
                break;
            case 4:
                gTxData[0] = 77;
                gTxData[1] = 120;
                gTxData[2] = 58;
                gTxData[3] = 32;
                sprintf(buffer, "%i", gMagX);
                break;
            case 5:
                gTxData[0] = 77;
                gTxData[1] = 121;
                gTxData[2] = 58;
                gTxData[3] = 32;
                sprintf(buffer, "%i", gMagY);
                break;
            case 6:
                gTxData[0] = 77;
                gTxData[1] = 122;
                gTxData[2] = 58;
                gTxData[3] = 32;
                sprintf(buffer, "%i", gMagZ);
                break;
            case 7:
                gTxData[0] = 71;
                gTxData[1] = 120;
                gTxData[2] = 58;

```

```

        gTxData[3] = 32;
        sprintf(buffer, "%i", gGyrX);
        break;
    case 8:
        gTxData[0] = 71;
        gTxData[1] = 121;
        gTxData[2] = 58;
        gTxData[3] = 32;
        sprintf(buffer, "%i", gGyrY);
        break;
    case 9:
        gTxData[0] = 71;
        gTxData[1] = 122;
        gTxData[2] = 58;
        gTxData[3] = 32;
        sprintf(buffer, "%i", gGyrZ);
        break;
    case 10:
        gTxData[0] = 65;
        gTxData[1] = 120;
        gTxData[2] = 58;
        gTxData[3] = 32;
        sprintf(buffer, "%i", gAccX);
        break;
    case 11:
        gTxData[0] = 65;
        gTxData[1] = 121;
        gTxData[2] = 58;
        gTxData[3] = 32;
        sprintf(buffer, "%i", gAccY);
        break;
    case 12:
        gTxData[0] = 65;
        gTxData[1] = 122;
        gTxData[2] = 58;
        gTxData[3] = 32;
        sprintf(buffer, "%i", gAccZ);
        break;
    case 13:
        gTxData[0] = 68;
        gTxData[1] = 82;
        gTxData[2] = 86;
        gTxData[3] = 32;
        sprintf(buffer, "%i", gDRV);
        break;
    }
    count++;
    /* Filling the UART transfer variable */
    gTxData[4] = buffer[0];
    gTxData[5] = buffer[1];
    gTxData[6] = buffer[2];
    gTxData[7] = buffer[3];

    /* Optional delay to ensure UART TX is idle before starting transmission */
    delay_cycles(160000);

    UART_Console_write(&gTxData[0], 8);
    UART_Console_write(&gSpace[0], sizeof(gSpace));
}
UART_Console_write(&gSpace[0], sizeof(gSpace));
}

```

추가 리소스

1. [MSPM0 SDK 다운로드](#)
2. [SysConfig에 대해 자세히 알아보기](#)
3. [MSPM0L LaunchPad 개발 키트](#)
4. [MSPM0G LaunchPad 개발 키트](#)
5. [MSPM0 I2C 아카데미](#)
6. [MSPM0 UART 아카데미](#)
7. [MSPM0 ADC 아카데미](#)
8. [MSPM0 DMA 아카데미](#)
9. [MSPM0 이벤트 관리자 아카데미](#)

M0 장치를 지원하는 2개의 OPA 계측 증폭기

설명

이 **서브시스템 소프트웨어 예제**는 MSPM0 및 외부 저항을 사용하여 두 개의 OPA INA(계측 증폭기)를 만듭니다. 이 구성에서는 V_{i1} 및 V_{i2} 사이의 차이가 증폭되고 높은 공통 모드 제거로 단일 종단 신호를 출력합니다. 통합 INA의 출력은 장치의 내부 ADC 채널을 사용하여 샘플링할 수 있습니다.

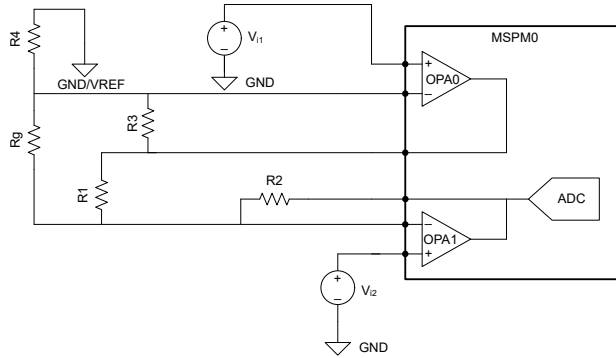


그림 19. 서브시스템 기능 블록 다이어그램

필요한 주변 기기

이 애플리케이션에는 MSPM0에 통합된 OPA 2개와 결과를 샘플링하기 위한 ADC 1개가 필요합니다.

표 9. 필요한 주변 기기

하위 블록 기능	사용되는 주변 기기	참고
OPA	OPA0	핀은 선택한 입력 소스를 기준으로 SysConfig에서 구성됨
OPA	OPA1	
ADC	ADC0	INA의 출력 전압을 측정하는 데 사용됨

호환 가능 장치

표 9의 요구 사항에 따라 이 예제는 표 10의 장치와 호환됩니다. 해당 EVM은 프로토타입 제작에 사용할 수 있습니다.

표 10. 호환 가능 장치

호환 가능 장치	EVM
MSPM0Lx	LP-MSPM0L1306
MSPM0Gx	LP-MSPM0G3507

설계 노트

MSPM0의 통합 증폭기를 사용하는 두 연산 증폭기 계측 증폭기의 설계는 개별 연산 증폭기를 사용하는 설계와 다르지 않습니다. **두 개의 연산 증폭기 계측 증폭기 회로** 애플리케이션 노트에서는 이 회로의 설계 노트를 다룹니다. 이는 편의를 위해 다음 목록에 설명되어 있습니다.

1. R_g 는 회로의 게인을 설정합니다.
2. 높은 값의 저항은 회로의 위상 여유를 저하시키고 회로에서 추가 잡음을 일으킬 수 있습니다.
3. R_4 와 R_3 의 비율은 R_g 가 제거된 경우의 최소 게인을 설정합니다.
4. R_2/R_1 과 R_4/R_3 의 비율은 계측 증폭기의 DC CMRR 열화를 방지하고 V_{ref} 게인이 1V/V가 되도록 일치해야 합니다.
5. 선형 작동은 사용된 개별 연산 증폭기의 입력 공통 모드 및 출력 스윙 범위에 따라 달라집니다. 선형 출력 스윙 범위는 장치 데이터 시트의 A_{OL} 테스트 조건에서 명시되어 있습니다.

설계 단계

설계 노트와 마찬가지로 두 개의 연산 증폭기 INA의 외부 회로를 설계하는 단계는 개별 방법과 다르지 않습니다. 다음 목록에서는 개별 설계에 대해 다루는 문서의 단계를 설명합니다.

1. 회로의 전송 함수를 계산합니다.

$$V_o = V_{iDiff} \times G + V_{ref} = (V_{i2} - V_{i1}) \times G + V_{ref} \quad (1)$$

when $V_{ref} = 0$, the transfer function simplifies to the following equation:

$$V_o = (V_{i2} - V_{i1}) \times G$$

where G is the gain of the instrumentation amplifier and $G = 1 + \frac{R_4}{R_3} + \frac{2R_2}{R_g}$

2. R_4 및 R_3 을 선택하여 최소 게인을 설정합니다.

$$G_{min} = 1 + \frac{R_4}{R_3} = 5 \frac{V}{V} \quad (2)$$

Choose $R_4 = 20k\Omega$

$$G_{min} = 1 + \frac{20k\Omega}{R_3} = 5 \frac{V}{V}$$

$$R_3 = \frac{R_4}{5-1} = \frac{20k\Omega}{4} = 5k\Omega \rightarrow R_3 = 5.1k\Omega \text{ (Standard Value)}$$

3. R_1 및 R_2 를 선택합니다. R_1/R_2 및 R_3/R_4 비율이 일치되어 1V/V에서 레퍼런스 전압에 게인이 인가되도록 설정합니다.

$$\frac{V_{o_ref}}{V_{ref}} = \left(-\frac{R_3}{R_4}\right) \times \left(-\frac{R_2}{R_1}\right) = \frac{R_3 \times R_2}{R_4 \times R_1} = 1 \frac{V}{V} \quad (3)$$

$$\frac{R_2}{R_1} = \frac{R_4}{R_3} \rightarrow R_1 = R_3 = 5.1k\Omega \text{ and } R_2 = R_4 = 20k\Omega \text{ (Standad Value)}$$

4. 원하는 최대 게인 $G = 10V/V$ 를 충족하도록 R_g 를 선택합니다.

$$G = 1 + \frac{R_4}{R_3} + \frac{2R_2}{R_g} = 1 + \frac{20k\Omega}{5.1k\Omega} + \frac{2 \times 20k\Omega}{R_g} = 10 \frac{V}{V} \quad (4)$$

$$R_g = 8k\Omega \rightarrow R_g = 7.87k\Omega \text{ (Standard Value)}$$

장치 구성

1. SysConfig 구성:
 - a. OPA의 반전 및 비반전 입력을 선택합니다.
 - b. 두 OPA에 대한 출력을 활성화합니다.
2. SysConfig의 상관 핀에 대한 연결로 외부 회로를 구축합니다.
3. 두 입력 전압 및 게인을 결정합니다. 자세한 내용은 [설계 고려 사항](#)에서 확인할 수 있습니다.

설계 고려 사항

1. 전압 레퍼런스:
 - 이 예제에서 V_{ref} 는 GND로 설정되지만, 전압을 R_4 로 연결하여 DC 레벨을 변경할 수 있습니다.
2. 출력 제한:
 - MSPM0 제품군의 경우 출력 신호는 VDD보다 클 수 없습니다.
3. OPA 모듈에 내장된 PGA도 사용할 수 있지만, 외부 저항 값을 조정해야 합니다. 비율이 반드시 같을 필요는 없으므로, 정합이 완전하지 않을 수 있습니다.
4. ADC는 설명된 것처럼 다양한 샘플링 속도 및 변환 해상도에 대해 설정할 수 있습니다. 이러한 구성은 SysConfig에서 수행할 수 있으며 ADC 및 OPA 기능에 대한 자세한 내용은 장치 TRM 및 데이터 시트에서 확인할 수 있습니다.
5. LaunchPad 구성: LaunchPad에서 OPA 입력 및 출력을 온보드 포토다이오드 또는 서미스터 회로와 같은 다양한 회로에 연결할 수 있습니다. 관련된 LaunchPad 사용 설명서를 확인하여 제거할 접퍼를 결정합니다.

참조

- 텍사스 인스트루먼트, [MSPM0 SDK 다운로드](#)
- 텍사스 인스트루먼트, [SysConfig에 대해 자세히 알아보기](#)
- 텍사스 인스트루먼트, [MSPM0L LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0G LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0 아카데미](#)
- 텍사스 인스트루먼트, 이 회로의 개별 구현을 위한 [두 개의 연산 증폭기 계속 증폭기 회로](#)

E2E

TI의 [E2E™](#) 지원 포럼 토론을 보고 새로운 스레드를 게시하여 설계에서 MSPM0 장치를 활용하는 데 필요한 기술 지원을 받으세요.

동적 프로그래머블 게인 증폭기

설계 설명

이 서브시스템은 PGA(프로그래머블 게인 증폭기) 구성에서 MSPM0 내부 연산 증폭기를 설정하고, 게인을 동적으로 변경하고, 증폭된 신호를 출력하고, ADC로 결과를 읽는 방법을 보여줍니다. 이 구성을 사용하면 사용자가 높은 게인 및 작은 입력 전압 신호로 해상도를 최대화할 수 있지만, 더 낮은 게인으로 변경하여 더 큰 신호를 샘플링할 수 있습니다. [이 예제의 코드를 다운로드하세요.](#)

그림 20에서는 이 서브시스템의 기능 다이어그램을 보여줍니다.

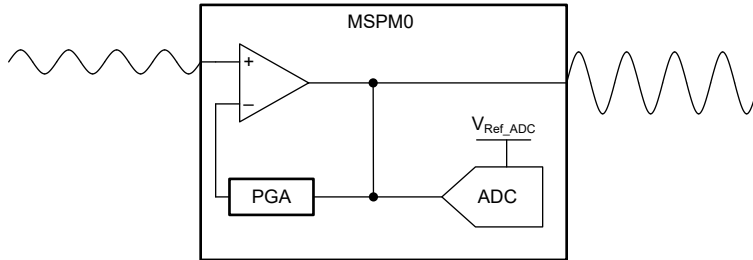


그림 20. 서브시스템 기능 블록 다이어그램

필요한 주변 기기

이 애플리케이션에는 통합 OPA 1개 및 ADC가 필요합니다.

표 11. 필요한 주변 기기

하위 블록 기능	사용되는 주변 기기	참고
게인 증폭기	(1x) OPA	코드에서 "OPA_0_INST"라고 부름
아날로그 신호 캡처	(1x) ADC12	코드에서 "ADC12_0_INST"라고 부름

호환 가능 장치

표 11의 요구 사항에 따라 이 예제는 표 12의 장치와 호환됩니다. 해당 EVM은 프로토타입 제작에 사용할 수 있습니다.

표 12. 호환 가능 장치

호환 가능 장치	EVM
MSPM0L13xx	LP-MSPM0L1306
MSPM0G35xx, MSPM0G15xx	LP-MSPM0G3507

설계 단계

1. 원하는 신호에 적용하려는 최고 및 최저 게인 설정을 결정합니다. OPA 모듈이 제공할 수 있는 최저 게인은 2이고, 최대 게인은 32입니다. ADC를 사용한 샘플링 시 설계 고려 사항도 참조하세요.

- a. 최대 입력 전압에 따른 시스템의 최소 게인을 계산합니다.

$$G_{min} = \frac{V_{ADC_Ref}}{V_{in_max}} \tag{5}$$

- b. 원하는 최소 입력 전압에 따른 시스템의 최대 게인을 계산합니다.

$$G_{max} = \frac{V_{ADC_Ref}}{V_{in_min}} \quad (6)$$

여기서

- G_{max} 는 OPA에 대해 선택한 최대 시스템 게인 설정입니다.
- G_{min} 은 OPA에 대해 선택한 최소 시스템 게인 설정입니다.
- V_{in_max} 는 최대 입력 전압입니다.
- V_{in_min} 은 원하는 최소 입력 전압입니다.
- V_{ADC_Ref} 는 ADC 레퍼런스 전압입니다.

2. 주어진 입력 전압과 게인에 대해 ADC로 인가되는 전압을 계산합니다.

$$V_{ADCin} = V_{OPAin} \times G_{OPA} \quad (7)$$

여기서

- V_{ADCin} 은 ADC 입력에서 샘플링되는 전압입니다.
- V_{OPAin} 은 OPA에 대한 전압 입력입니다.
- G_{OPA} 은 OPA에 대해 설정된 현재 게인입니다.

3. 주어진 ADC 입력 전압에 대한 ADC 코드를 계산합니다.

$$N_{ADC} = 2^{12} \times \frac{V_{ADCin} + \left(0.5 \times \frac{V_{ADC_Ref}}{2^{12}}\right)}{V_{ADC_Ref}} \quad (8)$$

여기서

- N_{ADC} 는 ADC 변환의 숫자 코드입니다.

4. 지정된 ADC 코드의 OPA 입력 전압을 계산할 때 다음 방정식을 사용합니다. 이 방정식과 설계 단계 3의 방정식은 OPA 게인 전환을 위한 ADC 윈도우 비교기 값을 결정할 때 다음 단계에서 유용합니다.

$$V_{OPAin} = \frac{V_{ADC_Ref}(N_{ADC} - 0.5)}{G_{OPA} \times 2^{12}} \quad (9)$$

5. 고압측 전환 레벨을 계산합니다. ADC 판독값이 이 값을 초과하면 가능한 경우 예제에서는 OPA 게인을 줄입니다. 이 예제에서 고압측 전환 레벨은 최대 ADC 레벨의 상위 5%로 설정됩니다.

$$V_{OPA_in} > H_T \times \frac{V_{ADC_Ref}}{G_{OPA}} \quad (10)$$

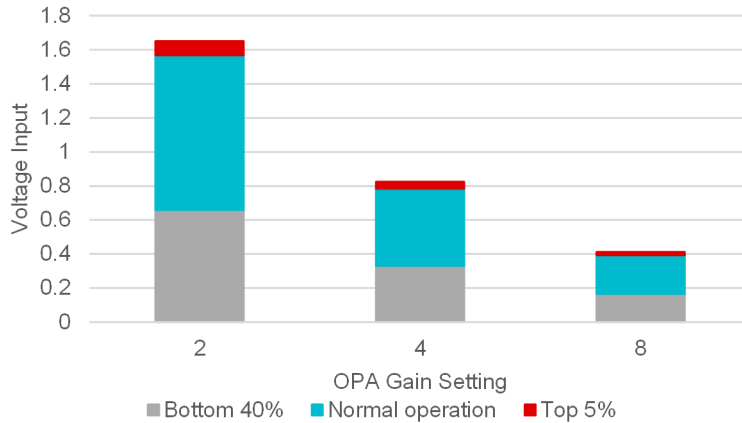
여기서 H_T 는 상한 백분율입니다.

6. 저압측 전환 레벨을 계산합니다. ADC 판독값이 이 값에 미달하면 가능한 경우 예제에서는 OPA 게인을 증가시킵니다. 이 예제에서 저압측 전환 레벨은 최대 ADC 레벨의 하위 40%로 설정됩니다.

$$V_{OPA_in} < L_T \times \frac{V_{ADC_Ref}}{G_{OPA}} \tag{11}$$

여기서 L_T 는 하한 백분율입니다.

- 설계 단계 5 및 6에서 설명된 레벨(H_T 는 상위 5%, L_T 는 하위 40%)은 아래 차트에서 여러 OPA 게인 설정별로 확인할 수 있습니다. 이러한 값은 전환을 위한 일부 버퍼와 함께 낮은 전압 레벨 입력에서 해상도를 최대화하는 데 도움을 주기 위해 선택되었습니다. 아래 차트에서 빨간색은 설계 단계 5의 전환 레벨에 해당하고, 회색은 설계 단계 6의 전환 수준을 나타내며, 마지막으로 파란색 영역은 게인이 변경되지 않은 전압 범위를 나타냅니다. 전환 레벨 선택에 대한 자세한 내용은 설계 고려 사항 6에서 설명합니다.



- SysConfig에서 외부 입력 및 외부 출력으로 PGA 구성을 위한 OPA를 설정합니다.
- SysConfig에서 VCC를 레퍼런스로 사용하여 윈도우 비교기 모드를 위한 ADC를 설정하고(V_{Ref_ADC}) OPA 출력 샘플링 합니다.
- 설계 단계 3의 방정식을 사용하여 설계 단계 5 및 6에서 결정된 전환 레벨을 ADC 코드로 변환하고 이를 SysConfig에서 ADC 윈도우 비교기 한도에 배치합니다.
- (옵션) 선택한 ADCMEMx로 OPA 출력도 샘플링하도록 ADC를 설정합니다.
- SysConfig에서 ADC 샘플 시간을 장치의 데이터 시트에 지정된 대로 t_{Sample_PGA} 의 최소 시간으로 설정합니다.

설계 고려 사항

- OPA 공급 전원은 MSPM0의 VCC입니다.
- OPA GBW 설정: OPA의 GBW 설정이 낮으면 소비 전류는 적지만, 더 느리게 응답합니다. 반대로 GBW가 높을수록 전류를 더 소비하지만, 회전을 더 크고 활성화와 정착 시간이 더 빨라집니다. 모드 간의 정확한 사양 차이는 장치별 데이터 시트를 확인하세요.
- OPA 게인 전환: OPA 게인 레벨을 건너뛰려면 레벨을 늘리거나 줄이는 대신 ADC 윈도우 비교기 ISR(인터럽트 서비스 루틴)에 코드를 추가하여 OPA 게인 설정을 명시적으로 설정해야 합니다. 설계 단계 5 및 6에서 계산된 전환 레벨도 이러한 유형의 전환을 반영하도록 합니다.
- 최소 OPA 게인: MSPM0 MCU는 OPA를 비활성화하지 않고 OPA 게인 설정을 동적으로 변경할 수 있습니다. PGA 구성에서 OPA의 최소 게인은 2입니다. 게인 값 2에서 OPA 버퍼 구성(OPA 게인 = 1)으로 변경하려면 이 문서의 범위를 벗어나는 추가 절차를 수행하여 OPA를 이 모드로 재구성해야 합니다.

5. ADC 레퍼런스 선택: MSPM0 장치는 내부 레퍼런스 생성기(VREF), 외부 소스 또는 MCU VCC에서 ADC에 레퍼런스 전압을 공급할 수 있습니다. 선택한 장치에 사용할 수 있는 옵션은 MSPM0 장치 데이터 시트에서 확인하세요. 선택한 레퍼런스 전압은 ADC가 샘플링할 수 있는 최대 눈금 범위를 설정하며, 최대 OPA 출력 전압을 수용해야 합니다.
6. ADC 윈도우 비교기 레벨:
 - a. 더 낮은 게인 값에서 더 높은 게인 값으로 전환(예: $G = 2 \rightarrow 4$)하여 입력 신호의 증폭을 증가시킬 때, 설계 단계 2의 방정식을 사용하여 전환에 대해 선택한 전압 레벨이 새로운 게인 설정에서 신호를 레일에 걸리지 않는지 확인하세요.
 - b. 더 높은 게인 값에서 더 낮은 게인 값으로 전환(예: $G = 4 \rightarrow 2$)하여 입력 신호의 증폭을 줄일 때 선택한 전압 레벨이 설계 고려 사항 6.a에서 선택한 전환 레벨보다 커야 합니다. 이는 시스템 불안정성을 일으킬 수 있는 게인 변경 루프를 방지하기 위한 것입니다.
7. ADC 샘플링: 이 예제에서는 윈도우 비교기 모드에서 OPA 출력을 지속적으로 샘플링합니다. OPA 출력의 지속적인 모니터링을 원하지 않는 경우 타이머를 사용하여 고정된 샘플링 간격을 설정할 수 있습니다.
8. ADC 결과: OPA 출력의 선택적 ADC 샘플링이 포함된 코드 예제에서는 전역 변수 *gADCResult*에 캡처된 최신 결과만 저장합니다. 전체 애플리케이션은 데이터에 대한 작업을 수행하기 전에 배열에 여러 개의 판독값을 저장할 수 있습니다.
9. ADC 결과: ADC 결과 캡처 옵션을 사용하는 경우 현재 OPA 게인 설정에 따라 처리되는 데이터를 처리하도록 코드를 추가해야 합니다. 이는 OPA 게인 설정에 따라 ADC 최대 눈금 범위가 변경되기 때문에 OPA의 여러 입력 전압 레벨에서 동일한 ADC 코드를 확인할 수 있습니다.
10. gCheckADC의 경쟁 상태: 이 애플리케이션은 가능한 한 빨리 gCheckADC를 지웁니다. 애플리케이션이 gCheckADC를 너무 늦게 지우면 새로운 데이터를 놓칠 수 있습니다.

소프트웨어 흐름도

그림 21에서는 ADC가 OPA 출력을 샘플링하고 OPA 게인을 변경하는 방법을 설명하는 *Dynamic_PGA_Example2*에 대한 코드 흐름 다이어그램을 보여줍니다. *Dynamic_PGA1_Example*의 소프트웨어 흐름도는 ADC가 시작된 후 메인 루프가 절전 모드로 전환되고 ADC ISR(인터럽트 서비스 루틴)의 중앙 스위치 케이스가 존재하지 않으므로 아래 흐름에서 약간 간소화됩니다.

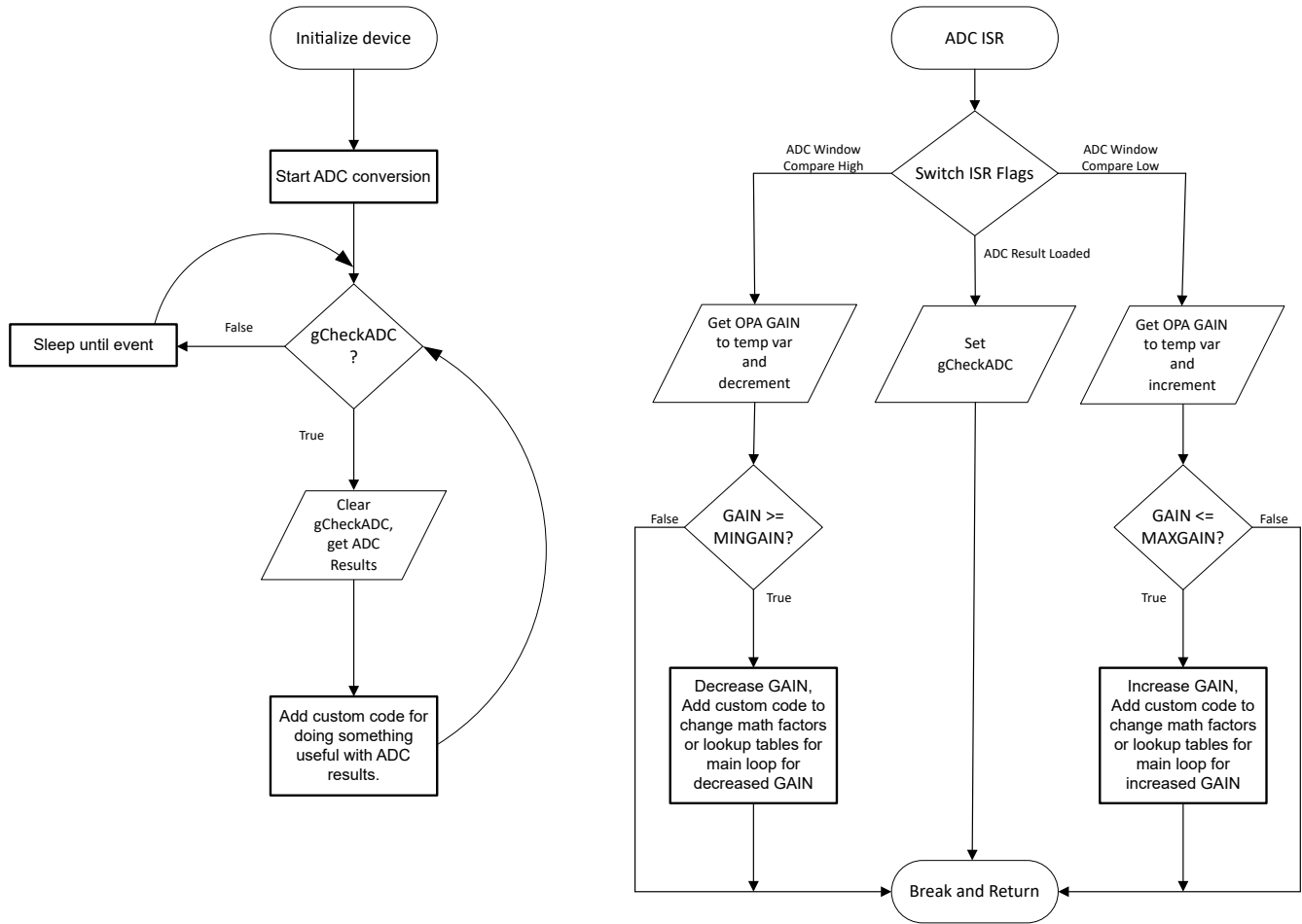


그림 21. 애플리케이션 소프트웨어 흐름도

장치 구성

이 애플리케이션은 TI 시스템 구성 툴(SysConfig) 그래픽 인터페이스를 사용하여 OPA 및 ADC의 구성 코드를 생성합니다. 그래픽 인터페이스를 사용하여 장치 주변 기기를 구성하면 애플리케이션 프로토타입 제작 프로세스를 간소화할 수 있습니다.

그림 2에 설명된 코드는 *Dynamic_PGA1_Example.c* 또는 *Dynamic_PGA_Example2.c* 파일의 *main()* 시작 부분에서 확인할 수 있습니다.

애플리케이션 코드

다음 코드 스니펫은 설계 단계 2에서 설명한 대로 최대 ADC 코드의 백분율에 따라 OPA 게인 레벨 및 전환 지점을 조정할 수 있는 위치를 보여줍니다. 사용 가능한 OPA 게인 정의에 대한 MSPM0 SDK 및 DriverLib 문서를 참조하세요.

```
#include "ti_msp_dl_config.h"

#define HIGHMARGIN 3890 // 4095*0.75 = 75% of max ADC value
#define LOWMARGIN 1638 // 4095*0.25 = 25% of max ADC value
#define MAXGAIN DL_OPA_GAIN_N7_P8 // Maximum GAIN level of OPA wanted
#define MINGAIN DL_OPA_GAIN_N1_P2 // Minimum GAIN level of OPA wanted.
//For non-inverting PGA mode this is an OPA GAIN of 2x. See advisory in TRM for MIN GAIN.
```

다음 코드 스니펫에서는 ADC 결과를 가져온 후 유용한 작업을 수행하기 위해 사용자 지정 코드를 추가하는 위치를 보여줍니다. 일반적으로 이는 배열, 필터링 또는 조회 테이블 액세스에 여러 결과를 배치하는 일종의 수학입니다.

```
while (1) {
    //This while loop waits until the next ADC result is loaded
    while (false == gcheckADC) {
        __WFE();
    }
    gcheckADC = false;
    //Grab latest ADC Result
    gADCResult = DL_ADC12_getMemResult(ADC12_0_INST, DL_ADC12_MEM_IDX_0);

    //Add in code to do math on ADC results.
    //Scaling factors for the math will be dependent on the current OPA Gain levels.
}
```

다음 코드 스니펫은 OPA 게인 설정에 따라 ADC 결과 해석을 조정할 수 있는 위치를 보여줍니다. 수행할 작업을 결정하고 ADC 결과와 OPA 게인 설정 및 입력 전압과 어떻게 연관 지을지는 사용자에게 달려 있습니다.

```
switch (DL_ADC12_getPendingInterrupt(ADC12_0_INST)) {
    case DL_ADC12_IIDX_WINDOW_COMP_HIGH:
        // Entered high side margin window. Decrease OPA GAIN if possible.
        tempGain = DL_OPA_getGain(OPA_0_INST);
        if(tempGain > MINGAIN){
            //Update OPA gain.
            DL_OPA_decreaseGain(OPA_0_INST);
            //For full applications, at this point you would want to adjust any math factors or
            //look up tables to the new voltage ranges being captured by the ADC, or set a flag to do so
in main while loop.
        }
        break;
    case DL_ADC12_IIDX_WINDOW_COMP_LOW:
        // Entered low side margin window. Increase OPA GAIN if possible.
        tempGain = DL_OPA_getGain(OPA_0_INST);
        if(tempGain < MAXGAIN){
            //Update OPA gain.
            DL_OPA_increaseGain(OPA_0_INST);
            //For full applications, at this point you would want to adjust any math factors or
            //look up tables to the new voltage ranges being captured by the ADC, or set a flag to do so
in main while loop.
        }
        break;
    default:
        break;
}
```

결과

다음 그래프는 OPA 입력 변경 및 해당 게인 적용 출력의 캡처를 보여줍니다. OPA 게인 레벨은 다음과 같습니다. 2x, 4x, 8x.

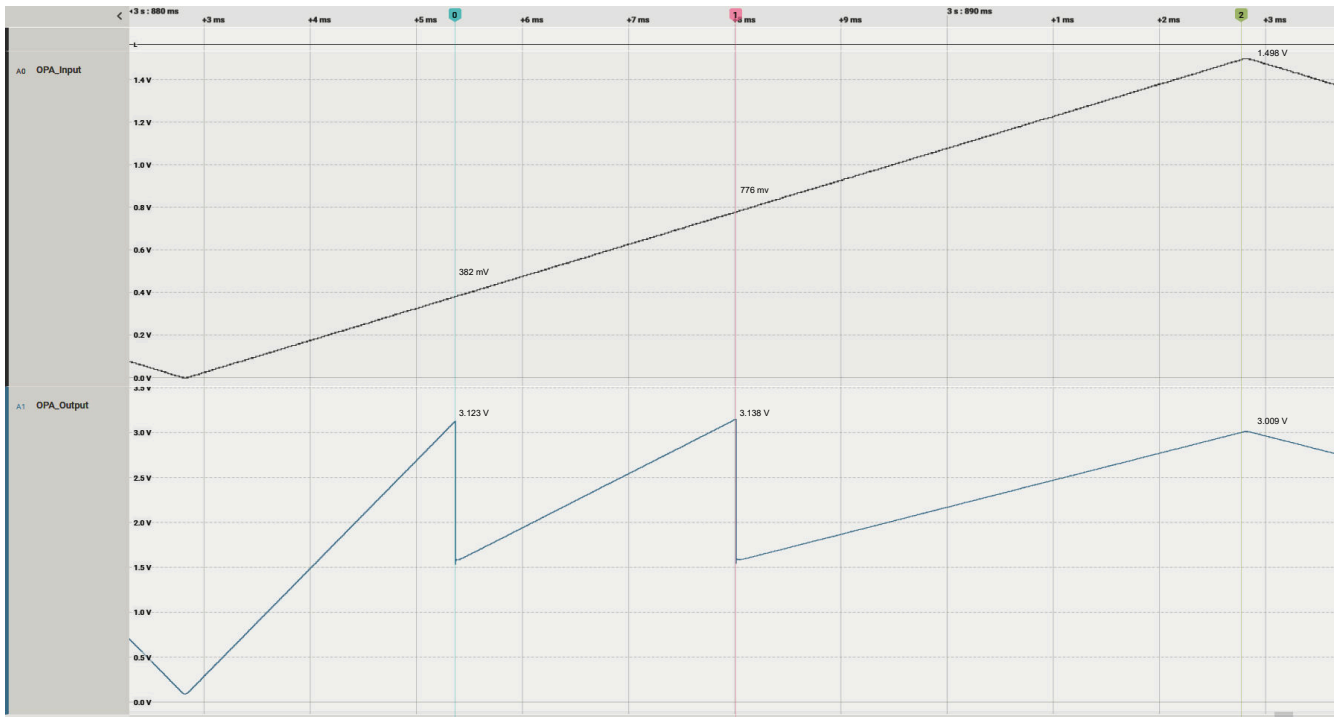


그림 22. OPA PGA 게인 늘리기

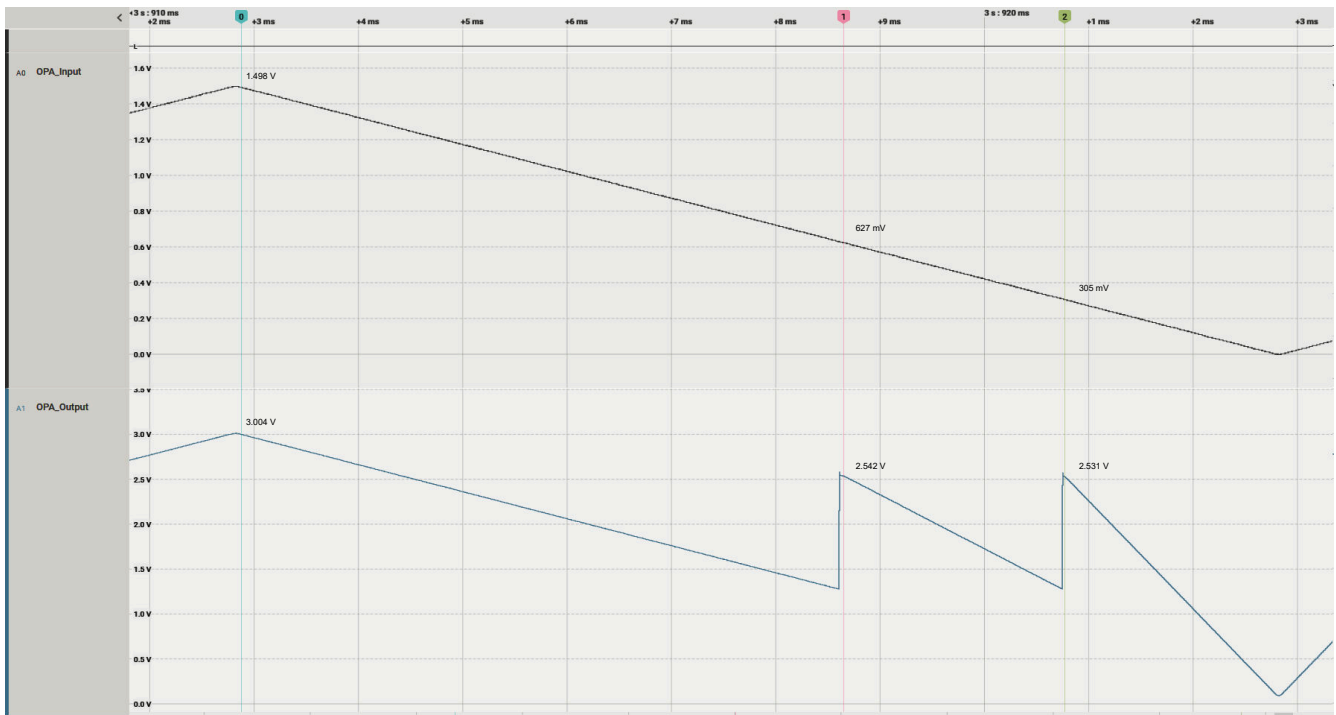


그림 23. OPA PGA 게인 줄이기

추가 리소스

- [MSPM0 SDK 다운로드](#)
- [SysConfig에 대해 자세히 알아보기](#)
- [MSPM0L TRM\(기술 레퍼런스 매뉴얼\)](#)

- [MSPM0G TRM\(기술 레퍼런스 매뉴얼\)](#)
- [MSPM0L LaunchPad 개발 키트](#)
- [MSPM0G LaunchPad 개발 키트](#)
- [MSPM0 타이머 아카데미](#)
- [MSPM0 ADC 아카데미](#)
- [MSPM0 OPA 아카데미](#)

스캐닝 비교기

설명

이 서브시스템은 하나의 MSPM0 마이크로컨트롤러에 단일 통합 비교기 및 소프트웨어로 여러 비교기를 나타내는 방법을 보여줍니다. 이 프로세스를 통해 설계자는 비교기 기능을 극대화하고 장치의 실제 비교기보다 더 많은 이론적 비교기를 활용할 수 있습니다. 이 예제에서는 **그림 24**에서와 같이 세 가지 비교기 구성과 입력 핀을 순환하고 결과값이 있는 3개의 출력 핀을 설정합니다.

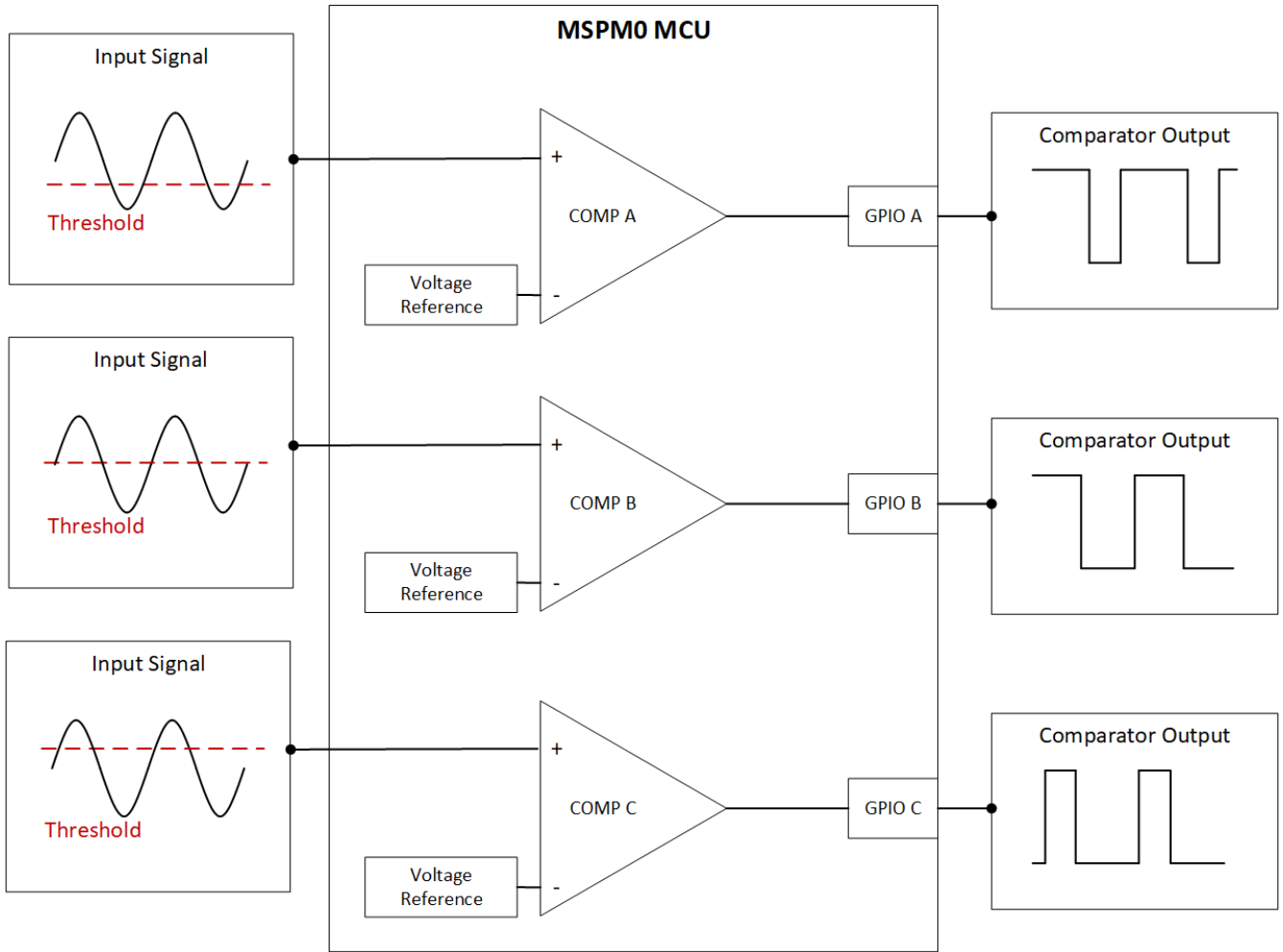


그림 24. 스캐닝 비교기 서브시스템의 이론적 기능

이 예제에서는 MSPM0 비교기에 대한 사용자 지정 가능한 IO 멀티플렉싱을 활용하여 동일한 비교기에 대한 여러 신호 입력을 지원합니다. 이 예제의 세 가지 신호 입력은 **그림 25**에 표시된 것처럼 COMP_IN0+, COMP_IN0- 및 COMP_IN1- 핀에 위치합니다.

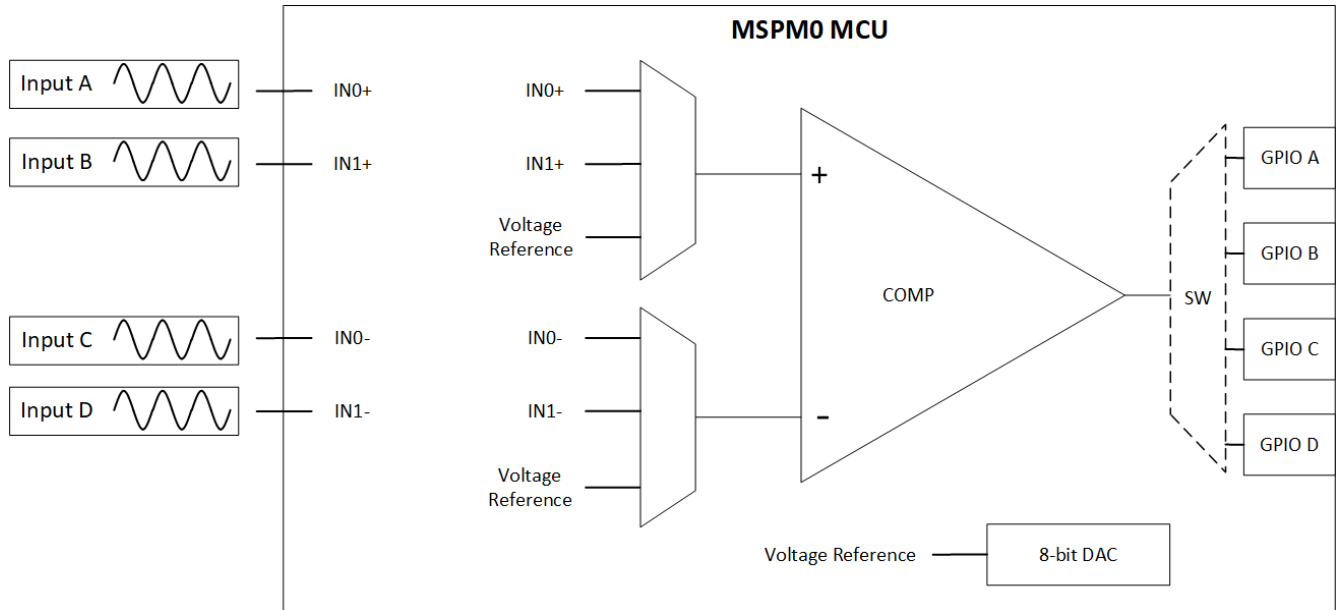


그림 25. 비교기 입력 및 출력 멀티플렉서

필요한 주변 기기

표 13에서는 필요한 통합 비교기 및 GPIO를 설명합니다.

표 13. 필요한 주변 기기

사용되는 주변 기기	참고
비교기	코드에서 COMP_INST라고 부름(8비트 레퍼런스 DAC 포함)
GPIO	세 GPIO를 핀 A, B 및 C라고 지칭합니다.

호환 가능 장치

표 13의 요구 사항에 따라 이 예제는 표 14에 표시된 장치와 호환됩니다. 해당 EVM은 프로토타입 제작에 사용할 수 있습니다.

표 14. 호환 가능 장치

호환 가능 장치	EVM	하드웨어 COMP	최대 비교기 입력
MSPM0L13xx	LP-MSPM0L1306	1	4
MSPM0Lx22x	LP-MSPM0L2228	1	4
MSPM0Gx5xx	LP-MSPM0G3507	3	17

설계 단계

1. 설계 요구 사항에 따라 작동 모드, 채널 입력 및 전압 레퍼런스를 비롯한 비교기에 대한 여러 구성을 결정합니다.
2. SysConfig를 사용하여 비교기 구성 코드를 생성합니다.
3. SysConfig에서 필요한 GPIO를 구성합니다.
4. 단계 1~2에서 각 비교기 구성에 대한 별도의 기능을 생성합니다.
5. 각 구성 설정을 호출하고 정착 시간을 위해 지연시키며, 결과를 해당 IO 핀에 할당하는 애플리케이션 코드를 작성합니다. **그림 26**에서 소프트웨어 개요를 확인하세요.

설계 고려 사항

1. 정착 시간: 비교기 구성을 업데이트한 후에는 결과를 읽기 전에 활성화 시간, DAC 정착 시간 및 전파 지연을 허용하는 지연이 애플리케이션 코드에 필요합니다. 애플리케이션 코드의 지연을 설정할 때는 해당 MSPM0 데이터 시트의 비교기 사양 섹션을 참조하세요.
2. 작동 모드: 비교기에는 고속 및 저전력 모드가 모두 있습니다. 고속 모드는 더 많은 전류를 소비하지만 비교기 판독 간 시간이 줄어듭니다. 저전력 모드는 판독 간에 더 긴 지연이 필요하지만 장치의 전류 소비가 감소합니다. 참고로, 이 예제에서는 고속 모드를 사용합니다.
3. 응답 시간: 서브시스템이 여러 비교기 구성을 통해 순환됨에 따라 이 프로세스는 최대 비교기 응답 시간을 증가시킵니다. 최대 응답 시간은 정착 시간 지연에 에물레이트된 비교기 구성 수를 곱한 값입니다. 표준 응답 시간 = x , 에물레이트된 응답 시간 = 지연 \times 에물레이트된 비교기(45 μ s)

소프트웨어 흐름도

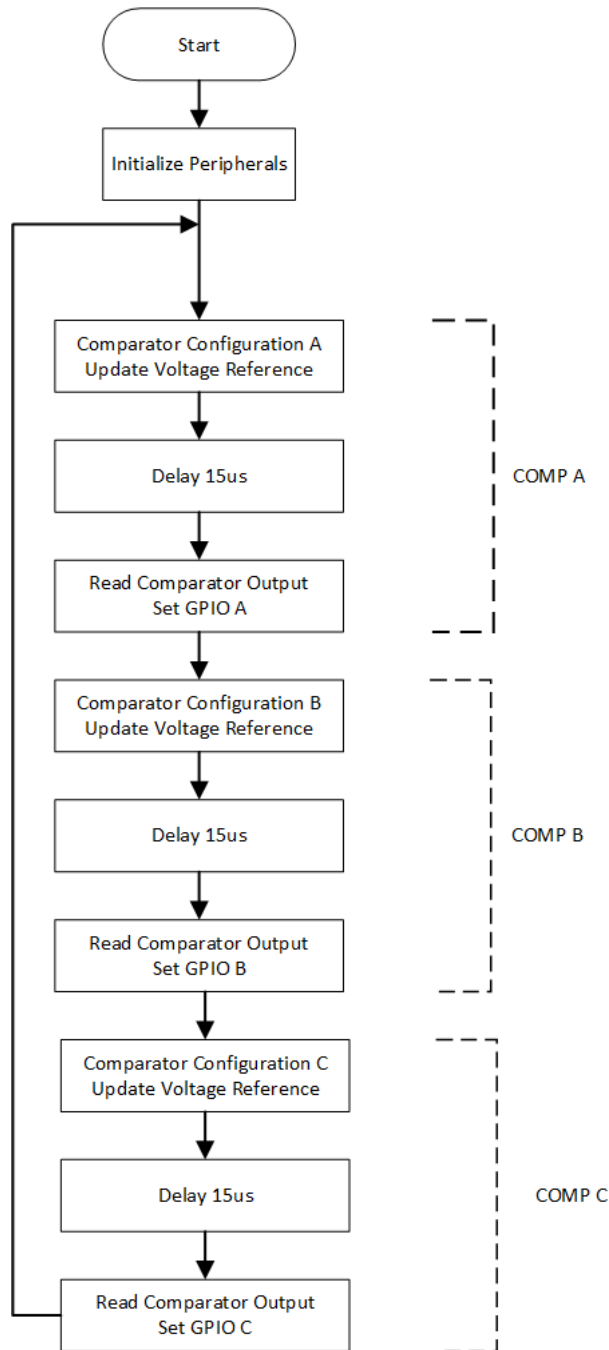


그림 26. 애플리케이션 소프트웨어 흐름도

애플리케이션 코드

애플리케이션 코드는 다음의 세 가지 함수를 호출하여 세 가지 비교기 구성을 순환합니다. update_comp_configA(), update_comp_configB() 및 update_comp_configC(). 비교기를 다시 구성할 때마다 애플리케이션 코드가 전파 및 정착 지연을 위해 15 μ s 동안 지연된 후 비교기 출력을 읽고 각 GPIO를 설정합니다.

```
int main(void)
{
    //initialization
    SYSCFG_DL_init();
    DL_COMP_enable(COMP_INST);
    DL_SYSCCTL_enableSleepOnExit();

    while (1) {

        //0.5V reference
        update_comp_configA();
        delay_cycles(480); //15us delay for comp stabilization
        if (DL_COMP_getComparatorOutput(COMP_INST) == 1){
            DL_GPIO_setPins(COMP_OUTPUT_PORT,COMP_OUTPUT_A_PIN); //set GPIO high
        }else{
            DL_GPIO_clearPins(COMP_OUTPUT_PORT,COMP_OUTPUT_A_PIN); //set GPIO low
        }

        //1.0V reference
        update_comp_configB();
        delay_cycles(480); //15us delay for comp stabilization
        if (DL_COMP_getComparatorOutput(COMP_INST) == 1){
            DL_GPIO_setPins(COMP_OUTPUT_PORT,COMP_OUTPUT_B_PIN); //set GPIO high
        }else{
            DL_GPIO_clearPins(COMP_OUTPUT_PORT,COMP_OUTPUT_B_PIN); //set GPIO low
        }

        //1.5V reference
        update_comp_configC();
        delay_cycles(480); //15us delay for comp stabilization
        if (DL_COMP_getComparatorOutput(COMP_INST) == 1){
            DL_GPIO_setPins(COMP_OUTPUT_PORT,COMP_OUTPUT_C_PIN); //set GPIO high
        }else{
            DL_GPIO_clearPins(COMP_OUTPUT_PORT,COMP_OUTPUT_C_PIN); //set GPIO low
        }
    }
}
```

그림 27. 스캐닝 비교기 Main.C

결과

그림 28에서는 스캐닝 비교기 서브시스템 예제의 결과를 보여줍니다. 에뮬레이트된 비교기 A, B 및 C는 각각 레퍼런스 전압이 0.5V, 1.0V 및 1.5V로 설정되었습니다. 세 개의 에뮬레이트된 비교기에서 각각 동일한 입력 신호가 측정되었습니다.

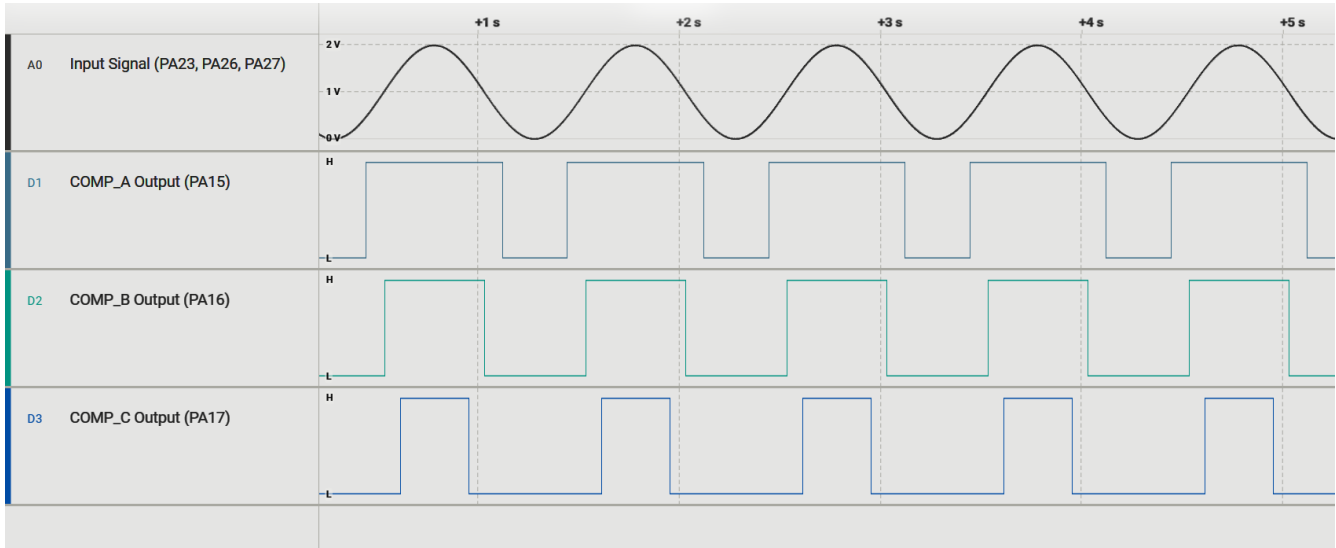


그림 28. 결과

범위 판독값은 하나의 물리적 비교기가 동시에 실행되는 세 개의 비교기를 모방할 수 있음을 보여줍니다. 이 예제 코드는 comp_hal.c 내에서 함수를 변경하여 다양한 비교기 수 및 구성에 맞게 편집할 수 있습니다.

추가 리소스

- 텍사스 인스트루먼트, [MSPM0 SDK 다운로드](#)
- 텍사스 인스트루먼트, [SysConfig에 대해 자세히 알아보기](#)
- 텍사스 인스트루먼트, [MSPM0L LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0G LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0 아카데미](#)

E2E

TI의 E2E™ 지원 포럼 토론을 보고 새로운 스레드를 게시하여 설계에서 MSPM0 장치를 활용하는 데 필요한 기술 지원을 받으세요.

트랜스임피던스 증폭기

설계 설명

이 서브시스템은 MSPM0 내부 연산 증폭기를 TIA(트랜스임피던스 증폭기) 구성으로 설정하고 내부 ADC로 출력을 읽는 방법을 보여줍니다. 트랜스임피던스 연산 증폭기 회로 구성은 입력 전류 소스를 출력 전압으로 변환합니다. 전류-전압 게인은 피드백 저항을 기반으로 합니다. **이 예제의 코드를 다운로드하세요.**

그림 29에서는 이 서브시스템의 기능 다이어그램을 보여줍니다.

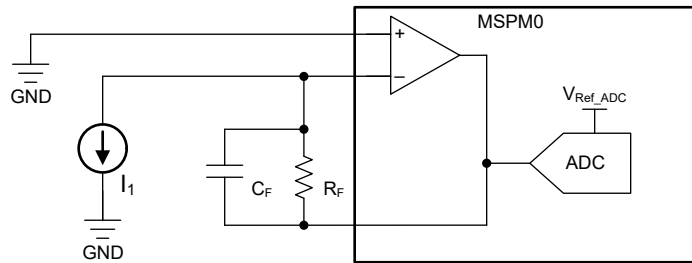


그림 29. 서브시스템 기능 블록 다이어그램

필요한 주변 기기

이 애플리케이션에는 통합 OPA 1개 및 ADC가 필요합니다.

표 15. 필요한 주변 기기

하위 블록 기능	사용되는 주변 기기	참고
TIA(전류-전압 변환)	(1x) OPA	코드에서 "TIA_INST"라고 부름
아날로그 신호 캡처	(1x) ADC12	코드에서 "ADC12_0_INST"라고 부름

호환 가능 장치

표 15의 요구 사항에 따라 이 예제는 표 16의 장치와 호환됩니다. 해당 EVM은 프로토타입 제작에 사용할 수 있습니다.

표 16. 호환 가능 장치

호환 가능 장치	EVM
MSPM0L13xx	LP-MSPM0L1306
MSPM0G35xx, MSPM0G15xx	LP-MSPM0G3507

설계 단계

1. 게인 저항 R_F 를 계산

$$R_F = \frac{V_{Ref_ADC} - V_{Min}}{I_{1Max}} \quad (12)$$

여기서

- V_{Ref_ADC} 는 ADC 주변 기기에 대해 선택한 레퍼런스입니다.
- V_{Min} 는 최소 연산 증폭기 출력 전압입니다.
- I_{1Max} 는 입력 전류 소스의 최대 전류입니다.

2. 회로 대역폭을 충족하도록 피드백 커패시터를 계산합니다.

$$C_F \leq \frac{1}{2 \times \pi \times R_F \times f_p} \quad (13)$$

여기서 f_p 는 입력 전류 소스의 최대 주파수입니다.

3. 회로가 안정적이기 위해 필요한 연산 증폭기 게인 대역폭(GBW)을 계산합니다.

$$GBW > \frac{C_i + C_F}{2 \times \pi \times R_F \times C_F^2} \quad (14)$$

여기서 $C_i = C_s + C_d + C_{cm}$ 주어진 값:

- C_s : 입력 소스 커패시턴스
- C_d : 증폭기의 차동 입력 커패시턴스. 일반적으로 MSPM0 장치의 경우 3pF로 추정할 수 있습니다.
- C_{cm} : 반전 입력의 공통 모드 입력 커패시턴스

4. 단계 3에서 계산된 값과 하한을 비교하여 활용할 수 있는 OPA GBW 설정을 결정합니다.
5. 회로에 대한 외부 연결을 위해 SysConfig에서 OPA를 설정합니다.
6. 선택한 OPA 출력에 대한 내부 연결을 위해 SysConfig에서 ADC를 설정합니다.
7. SysConfig에서 ADC 샘플 시간을 장치 데이터 시트에 지정된 대로 최소 t_{Sample_PGA} 로 설정합니다.

설계 고려 사항

1. OPA 공급 전원은 MSPM0의 VCC입니다.
2. OPA GBW 설정: OPA의 GBW 설정이 낮으면 소비 전류는 적지만, 더 느리게 응답합니다. 반대로 GBW가 높을수록 전류를 더 소비하지만, 회전율이 더 크고 활성화와 정착 시간이 더 빨라집니다. 모드 간의 사양 차이는 장치별 데이터 시트를 참조하세요.
3. OPA 비반전 입력: GND 전위 대신 작은 바이어스 전압이 OPA 비반전 입력에 인가되어 전류 소스가 활성 상태가 아닌 경우(예: 포토다이오드가 조명이 없는 상태인 경우) 출력이 GND로 포화되지 않도록 유지할 수 있습니다. 이는 외부 전압 입력 또는 비교기 주변 기기 내의 DAC12나 DAC8과 같은 내부 주변 기기를 통해 달성할 수 있습니다. 후자의 경우 OPA 비반전 입력과 연결된 핀을 다른 목적으로 사용할 수 있습니다.
4. ADC 샘플링: 이 예제에서는 OPA 출력을 지속적으로 샘플링합니다. 이를 원하지 않는 경우 타이머를 사용하여 고정된 샘플링 간격을 설정할 수 있습니다.

5. ADC 결과: 이 예제에서는 전역 변수 *gADCResult*에 캡처된 최신 결과만 저장합니다. 전체 애플리케이션은 데이터에 대한 작업을 수행하기 전에 배열에 여러 개의 판독값을 저장할 수 있습니다.
6. ADC 레퍼런스 선택: MSPM0 장치는 내부 레퍼런스 생성기(VREF), 외부 소스 또는 MCU VCC에서 ADC에 레퍼런스 전압을 공급할 수 있습니다. 선택한 장치에 사용할 수 있는 옵션은 장치별 데이터 시트를 참조하세요. 선택한 레퍼런스 전압은 ADC가 샘플링할 수 있는 최대 눈금 범위를 설정하며, 최대 OPA 출력 전압을 수용해야 합니다.
7. gCheckADC의 경쟁 상태: 이 애플리케이션은 가능한 한 빨리 gCheckADC를 지웁니다. 애플리케이션이 gCheckADC를 너무 늦게 지우면 새로운 데이터를 놓칠 수 있습니다.

소프트웨어 흐름도

그림 30에서는 이 예제에 대한 코드 흐름 다이어그램을 보여주고 ADC에서 OPA 출력을 샘플링하는 방법을 설명합니다.

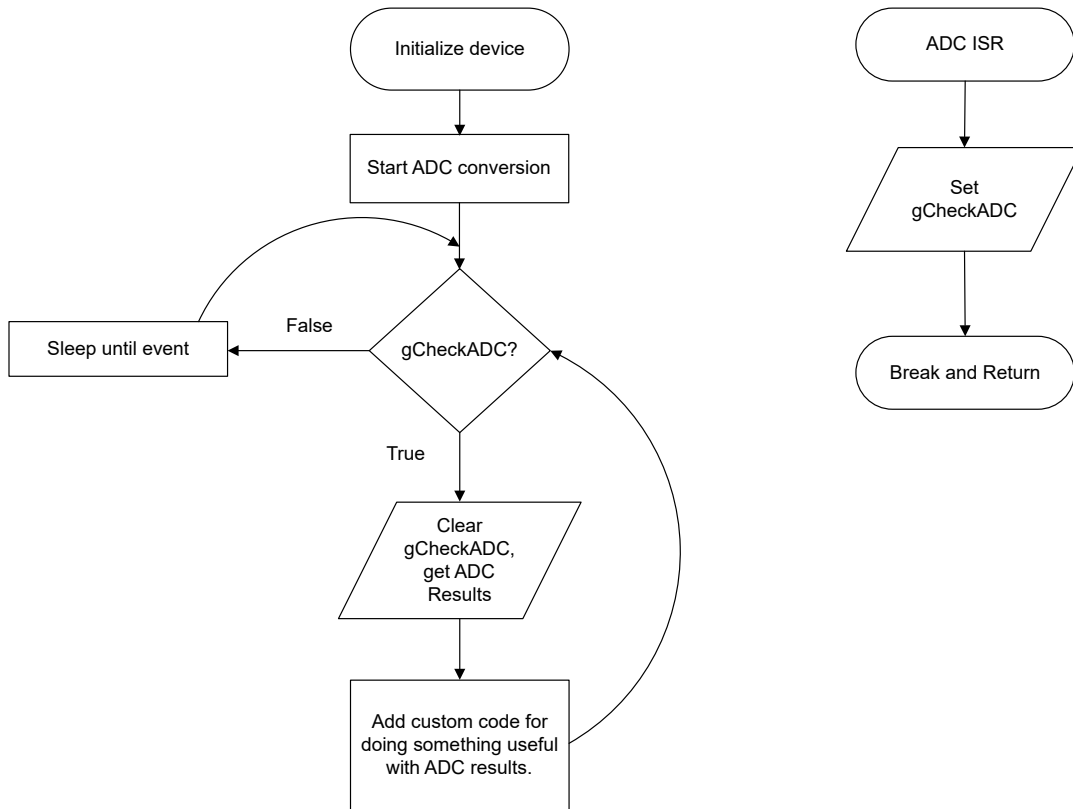


그림 30. 애플리케이션 소프트웨어 흐름도

장치 구성

이 애플리케이션은 TI 시스템 구성 툴(SysConfig) 그래픽 인터페이스를 사용하여 OPA 및 ADC의 구성 코드를 생성합니다. 그래픽 인터페이스를 사용하여 장치 주변 기기를 구성하면 애플리케이션 프로토타입 제작 프로세스를 간소화할 수 있습니다.

애플리케이션 코드

그림 30에 설명된 코드는 TIA_Example.c 파일의 main() 시작 부분에서 확인할 수 있습니다. 다음 코드 스니펫에서는 측정된 전류 소스의 ADC 결과를 가져온 후 유용한 작업을 수행하기 위해 사용자 지정 코드를 추가하는 위치를 보여줍니다. 수행할 작업을 결정하고 ADC 결과와 현재 소스 활동의 상관 관계를 파악하는 것은 사용자가 담당합니다. 예를 들어, 포토다이오드에 연결된 경우 설계에서 빛의 작은 변화는 무시하고 큰 변화를 감지하기 위해 델타 계산을 수행하도록 ADC 결과의 평균을 계산할 수 있습니다.

```

while (1) {
  DL_ADC12_startConversion(ADC12_0_INST);
  while (false == gCheckADC) {
    __WFE();
  }
  /* * This is where the ADC result is grabbed from ADC memory.
  * A user may want to modify this to place multiple results into an array,
  * or add code to perform additional calculations or filters to data obtained.
  */
  gADCResult = DL_ADC12_getMemResult(ADC12_0_INST, DL_ADC12_MEM_IDX_0);
}

```

```
gCheckADC = false;  
DL_ADC12_enableConversions(ADC12_0_INST);  
}
```

추가 리소스

- [MSPM0 SDK 다운로드](#)
- [SysConfig에 대해 자세히 알아보기](#)
- [MSPM0L LaunchPad 개발 키트](#)
- [MSPM0G LaunchPad 개발 키트](#)
- [MSPM0 타이머 아카데미](#)
- [MSPM0 ADC 아카데미](#)
- [MSPM0 OPA 아카데미](#)

서미스터 온도 감지

설계 설명

이 서브시스템은 PTC(양의 온도 계수) 서미스터(**TMP61**)와 저항을 직렬로 연결하여 전압 분할기를 형성함으로써 온도에 따라 선형적으로 변화하는 출력 전압을 생성하는 효과를 냅니다. 이 외부 회로는 버퍼 구성에서 MSPM0 내부 연산 증폭기를 설정하고 ADC를 사용하여 샘플링하여 판독됩니다. 온도 상승이 측정되면 RGB LED가 빨간색으로 바뀌고 온도가 감소하면 LED가 파란색으로 바뀝니다. 온도에 큰 변화가 없으면 LED는 녹색으로 유지됩니다. 이러한 계산은 선택한 서미스터에 따라 달라지기 때문에 ADC 판독값에서 온도 값을 계산하는 방법에 대해서는 이 문서에서 자세히 다루지 않습니다. [여기에서 코드 예제를 다운로드하세요.](#)

그림 31에서는 이 서브시스템의 기능 다이어그램을 보여줍니다.

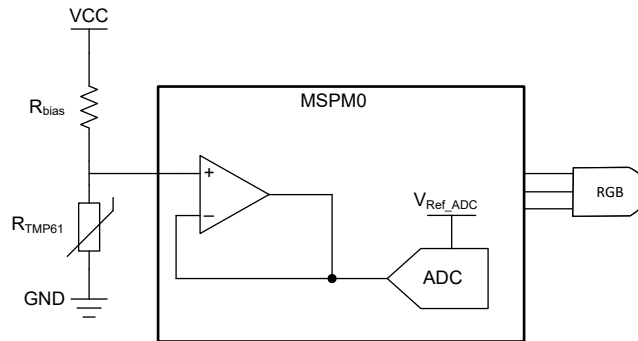


그림 31. 서브시스템 기능 블록 다이어그램

필요한 주변 기기

이 애플리케이션에는 통합 OPA 1개, ADC, 타이머 및 I/O 핀이 필요합니다.

표 17.

하위 블록 기능	사용되는 주변 기기	참고
버퍼 증폭기	(1x) OPA	코드에서 Thermistor_OPA_INST라고 부름
아날로그 신호 캡처	(1x) ADC12	코드에서 ADC_INST라고 부름
ADC 샘플링을 위한 타이머	(1x) TIMERx	코드에서 Thermistor_TIMER_ADC라고 부름
RGB LED 제어	(3x) I/O 핀	코드에서 RGB_RED_PIN, RGB_BLUE_PIN 및 RGB_GREEN_PIN이라고 부름

호환 가능 장치

표 17의 요구 사항에 따라 이 예제는 표 18의 장치와 호환됩니다. 해당 EVM은 프로토타입 제작에 사용할 수 있습니다.

표 18.

호환 가능 장치	EVM
MSPM0L13xx	LP-MSPM0L1306
MSPM0G35xx, MSPM0G15xx	LP-MSPM0G3507

설계 단계

1. R_{bias} 를 결정합니다. 이 설계에 사용되는 TMP61 서미스터의 경우 R_{bias} 을 10kΩ으로 설정하는 것을 권장합니다. 다른 구성도 사용할 수 있습니다. 자세한 내용은 TMP61 데이터 시트를 참조하세요.
 - a. 다른 서미스터에는 R_{bias} 를 계산하는 데 사용할 수 있는 다른 R_{bias} 권장 사항 또는 방정식이 적용될 수 있습니다. 자세한 내용은 선택한 서미스터의 문서를 참조하세요.
2. 외부 입력을 사용하여 버퍼 구성하려면 SysConfig에서 OPA를 설정합니다.
3. 선택한 ADCMEMx를 사용하여 SysConfig 샘플 OPA 출력에서 ADC를 설정합니다.
4. SysConfig에서 ADC 샘플 시간을 장치 데이터 시트에 지정된 대로 최소 t_{Sample_PGA} 로 설정합니다.
5. ADC 판독값을 온도 판독값으로 변환하는 데 사용할 온도 알고리즘을 결정합니다. 이 예제에서는 원시 ADC 판독값을 사용하여 온도 변화를 계산합니다.

설계 고려 사항

1. 온도 계산: 서미스터마다 ADC 판독 및 외부 회로에서 온도를 계산하기 위해 다양한 방정식 또는 조회 테이블이 있습니다. 이 설계에 통합할 수 있는 리소스에 대해서는 서미스터 참고 자료를 확인하세요.
 - a. 조회 테이블은 계산 시간이 짧지만, 일부 상황에서 유효하지 않을 수 있으며 많은 메모리를 사용할 수 있습니다.
 - b. 방정식은 계산 시간이 더 걸리지만, 외부 변수에 더 유연합니다. 방정식의 복잡성은 정확도 또는 온도 범위 요구 사항에 따라 달라집니다.
2. OPA 공급 전원은 MSPM0의 VCC가 됩니다.
3. OPA GBW 설정: OPA의 GBW 설정이 낮으면 소비 전류는 적지만, 더 느리게 응답합니다. 반대로 GBW가 높을수록 전류를 더 소비하지만, 회전율이 더 크고 활성화와 정착 시간이 더 빨라집니다. 모드 간의 사양 차이는 장치별 데이터 시트를 참조하세요.
4. ADC 레퍼런스 선택: MSPM0 장치는 내부 레퍼런스 생성기(VREF), 외부 소스 또는 MCU VCC에서 ADC에 레퍼런스 전압을 공급할 수 있습니다. 선택한 장치에 사용할 수 있는 옵션은 MSPM0 장치 데이터 시트에서 확인하세요. 이 설계의 구성을 위해서는 ADC 레퍼런스가 외부 서미스터 회로의 바이어스 전압(VCC)과 같도록 하는 것이 좋습니다.
5. ADC 샘플링: 이 예제에서는 타이머 트리거를 사용하여 외부 회로를 주기적으로 샘플링합니다. 회로가 샘플링되는 빈도를 조정하려면 타이머 매개 변수를 조정합니다.
6. ADC 결과: 이 코드 예제에서는 전역 변수 `gThermistorADCResult`에 캡처된 최신 결과만 저장합니다. 전체 애플리케이션에서는 데이터에 대한 작업을 수행하기 전 배열에 여러 개의 판독값을 저장해야 할 수 있습니다.
7. `gCheckThermistor`의 경쟁 상태: 이 애플리케이션은 가능한 한 빨리 `gCheckThermistor`를 지웁니다. 애플리케이션이 `gCheckThermistor`를 지울 때까지 너무 오래 대기하면 애플리케이션이 실수로 새 데이터를 놓칠 수 있습니다.

소프트웨어 흐름도

그림 32에서는 이 예제에 대한 코드 흐름 다이어그램을 보여주고 ADC에서 OPA 출력을 샘플링하는 방법과 LED 조명의 의사 결정 트리를 설명합니다.

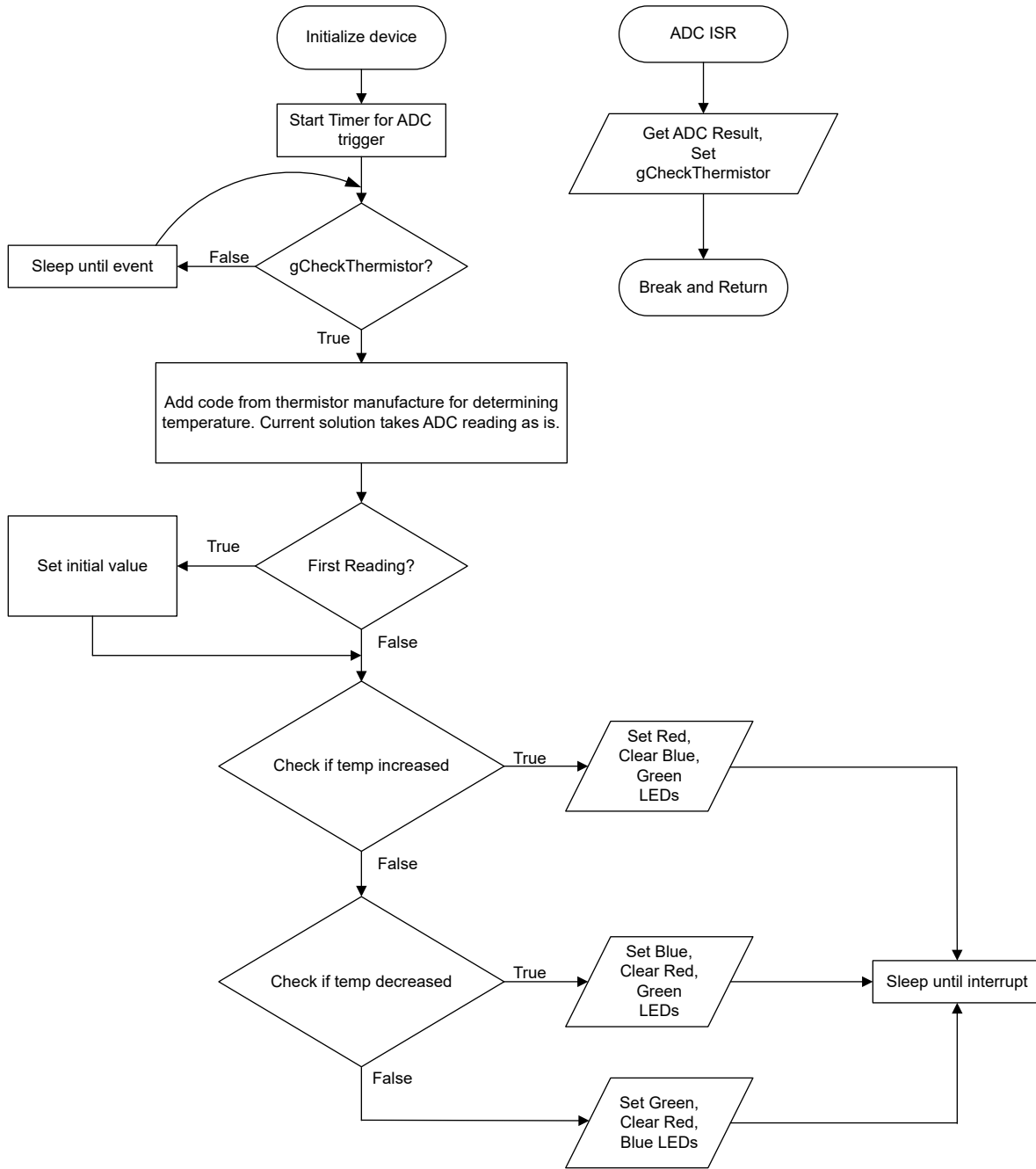


그림 32. 애플리케이션 소프트웨어 흐름도

장치 구성

이 애플리케이션은 TI 시스템 구성 툴(SysConfig) 그래픽 인터페이스를 사용하여 장치 주변 기기의 구성 코드를 생성합니다. 그래픽 인터페이스를 사용하여 장치 주변 기기를 구성하면 애플리케이션 프로토타입 제작 프로세스를 간소화할 수 있습니다.

그림 32에 설명된 코드는 *Thermistor_Example.c* 파일의 *main()* 시작 부분에서 확인할 수 있습니다.

애플리케이션 코드

이 애플리케이션은 온도를 직접 계산하지 않고 온도 변화를 찾습니다. 다음 코드 스니펫에는 온도 변화를 감지하기 전에 최소한의 ADC 값 변경을 확인하는 데 사용되는 CHANGEFACTOR 값이 포함되어 있습니다.

```
#include"ti_msp_dl_config.h"
#include<math.h>
#define CHANGEFACTOR 10
volatileuint16_tgThermistorADCResult = 0;
volatileboolgCheckThermistor = false;
```

다음 코드 스니펫에서는 실제 온도 값을 계산하기 위해 서미스터의 온도 계산 방법을 추가할 위치를 보여줍니다. 현재 코드는 시작 시 초기 판독값(*gInitial_reading*)을 취하고 현재 판독값(*gCelcius_reading*)을 CHANGEFACTOR 조정과 비교하여 온도가 충분히 증가/감소했는지 또는 변하지 않았는지 확인합니다. 그런 다음, 비교 결과에 따라 RGB LED가 빨간색(증가), 파란색(감소) 또는 녹색(변화 없음)으로 바뀝니다.

```
while (1) {
    while (gCheckThermistor == false) {
        __WFE();
    }
    //Insert Thermistor Algorithm
    gCelcius_reading = gThermistorADCResult;
    if (first_reading) {
        gInitial_reading = gCelcius_reading;
        first_reading = false;
    }
    /*
     * Change in LEDs is based on current sample compared to previous sample
     *
     * If the new sample is warmer than CHANGEFACTOR from initial temp, turn LED red
     * If the new sample is colder than CHANGEFACTOR from initial temp, turn LED blue
     * Else, keep LED green
     * Variable gAlivecheck is utilized for debug window to confirm code is executing.
     * It is not needed in final applications.
     */
    gAlivecheck++;
    if(gAlivecheck >= 0xFFFF){gAlivecheck =0;}
    if (gCelcius_reading - CHANGEFACTOR > gInitial_reading) {
        DL_GPIO_clearPins(
            RGB_PORT, (RGB_GREEN_PIN | RGB_BLUE_PIN));
        DL_GPIO_setPins(RGB_PORT, RGB_RED_PIN);
    } else if (gCelcius_reading < gInitial_reading - CHANGEFACTOR) {
        DL_GPIO_clearPins(
            RGB_PORT, (RGB_RED_PIN | RGB_BLUE_PIN));
        DL_GPIO_setPins(RGB_PORT, RGB_BLUE_PIN);
    } else {
        DL_GPIO_clearPins(
            RGB_PORT, (RGB_RED_PIN | RGB_BLUE_PIN));
        DL_GPIO_setPins(RGB_PORT, RGB_GREEN_PIN);
    }
    gCheckThermistor = false;
    __WFI();
}
```

추가 리소스

1. [MSPM0 SDK 다운로드](#)
2. [SysConfig에 대해 자세히 알아보기](#)
3. [MSPM0L LaunchPad](#)
4. [MSPM0G LaunchPad](#)

5. **MSPM0 타이머 아카데미**
6. **MSPM0 ADC 아카데미**
7. **MSPM0 OPA 아카데미**

통신 브리지

- CAN-I2C 브리지 •
- I2C-UART 서브시스템 설계 •
- CAN-SPI 브리지 •
- CAN-UART 브리지 •
- 병렬 IO-UART 브리지 •
- UART 브리지를 통한 I2C 확장기 •
- UART-I2C 브리지 •
- UART-SPI 브리지 •

CAN-I2C 브리지

설계 설명

이 서브시스템은 CAN-I2C 브리지를 구축하는 방법을 보여줍니다. CAN-I2C 브리지를 사용하면 장치가 한 인터페이스에서 정보를 송신 또는 수신하고 다른 인터페이스에서 정보를 수신 또는 전송할 수 있습니다. **이 예제의 코드를 다운로드하세요.** 각각 컨트롤러 모드 또는 대상 모드에서 작동하도록 I2C를 지원하는 두 개의 예제 코드가 제공됩니다.

그림 33에서는 이 서브시스템의 기능 다이어그램을 보여줍니다. I2C 대상에서 I2C 컨트롤러로의 메시지 전송을 구현하기 위해 IO 인터럽트에 하나의 라인이 추가되었습니다.

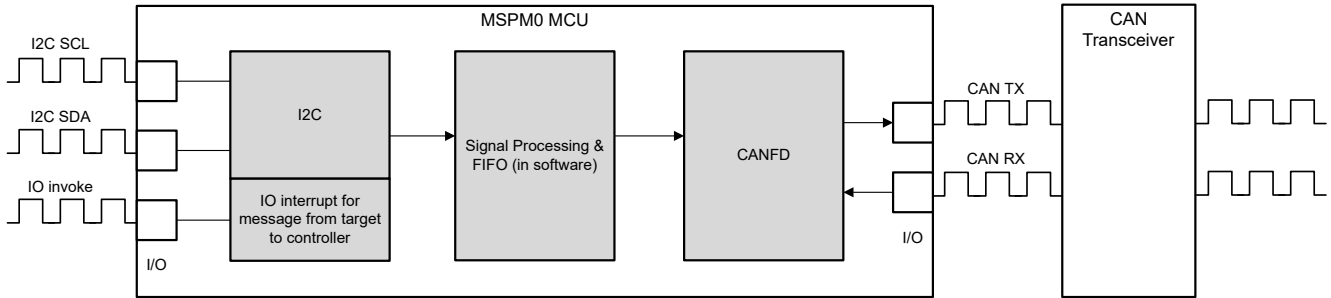


그림 33. 서브시스템 기능 블록 다이어그램

필요한 주변 기기

이 애플리케이션에는 CANFD 및 I2C가 필요합니다.

표 19. 필요한 주변 기기

하위 블록 기능	사용되는 주변 기기	참고
CAN 인터페이스	(1x) CANFD	코드에서 <i>MCAN0_INST</i> 라고 부름
I2C 인터페이스	(1x) I2C	코드에서 <i>I2C_INST</i> 라고 부름

호환 가능 장치

표 19의 요구 사항에 따라 이 예제는 표 20의 장치와 호환됩니다. 해당 EVM은 프로토타입 제작에 사용할 수 있습니다.

표 20. 호환 가능 장치

호환 가능 장치	EVM
MSPM0G35xx	LP-MSPM0G3507

설계 단계

1. CAN 모드, 비트 타이밍, 메시지 RAM 구성 등을 포함한 CAN 인터페이스의 기본 설정을 결정합니다. 고정된 설정과 애플리케이션에서 변경되는 설정을 고려합니다. 예제 코드에서 CANFD는 250kbit/s 중재 비율 및 2Mbit/s 데이터 전송률로 사용됩니다.
 - a. CAN-FD 주변 기기의 주요 특징은 다음과 같습니다.
 - i. ECC가 있는 전용 1KB 메시지 SRAM
 - ii. 구성 가능한 전송 FIFO, 전송 대기열 및 이벤트 FIFO(최대 32개 요소)

- iii. 최대 32개의 전용 전송 버퍼 및 64개의 전용 수신 버퍼가 있습니다. 2개의 구성 가능한 수신 FIFO(각각 최대 64개 요소)
 - iv. 최대 128개의 필터 요소
- b. CANFD 모드가 활성화된 경우:
- i. 64바이트 CAN-FD 프레임을 완벽하게 지원
 - ii. 최대 8Mbit/s의 비트 레이트
- c. CANFD 모드가 비활성화된 경우:
- i. 8바이트 클래식 CAN 프레임을 완벽하게 지원
 - ii. 최대 1Mbit/s의 비트 레이트
2. 데이터 길이, 비트 레이트 전환, 식별자, 데이터 등을 포함한 CAN 프레임을 결정합니다. 고정된 부품과 애플리케이션에서 변경해야 하는 부분을 고려합니다. 예제 코드에서 식별자, 데이터 길이 및 데이터는 서로 다른 프레임에서 변경되는 반면 다른 항목은 고정되어 있습니다. 프로토콜 통신이 필요한 경우 사용자는 코드를 수정해야 합니다.

```

/**
 * @brief Structure for MCAN Rx Buffer element.
 */
typedef struct {
    /* Identifier */
    uint32_t id;
    /* Remote Transmission Request
     * 0 = Received frame is a data frame
     * 1 = Received frame is a remote frame
     */
    uint32_t rtr;
    /* Extended Identifier
     * 0 = 11-bit standard identifier
     * 1 = 29-bit extended identifier
     */
    uint32_t xtd;
    /* Error State Indicator
     * 0 = Transmitting node is error active
     * 1 = Transmitting node is error passive
     */
    uint32_t esi;
    /* Rx Timestamp */
    uint32_t rxts;
    /* Data Length Code
     * 0-8 = CAN + CAN FD: received frame has 0-8 data bytes
     * 9-15 = CAN: received frame has 8 data bytes
     * 9-15 = CAN FD: received frame has 12/16/20/24/32/48/64 data bytes
     */
    uint32_t dlc;
    /* Bit Rat Switching
     * 0 = Frame received without bit rate switching
     * 1 = Frame received with bit rate switching
     */
    uint32_t brs;
    /* FD Format
     * 0 = Standard frame format
     * 1 = CAN FD frame format (new DLC-coding and CRC)
     */
    uint32_t fdf;
    /* Filter Index */
    uint32_t fidx;
    /* Accepted Non-matching Frame
     * 0 = Received frame matching filter index FIDX
     * 1 = Received frame did not match any Rx filter element
     */
    uint32_t anmf;
    /* Data bytes.
     * Only first dlc number of bytes are valid.

```

```

*/
uint16_t data[DL_MCAN_MAX_PAYLOAD_BYTES];
} DL_MCAN_RxBufElement;
    
```

3. I2C 모드, 버스 속도, 대상 주소, FIFO 등을 포함한 I2C 인터페이스의 기본 설정을 결정합니다. 고정된 설정과 애플리케이션에서 변경되는 설정을 고려합니다. 한 예제 코드는 400kHz 버스 속도를 가진 I2C 컨트롤러에 사용되고, 다른 하나는 주소 0x48의 I2C 대상에 사용됩니다.
 - a. I2C 주변 기기의 주요 특징은 다음과 같습니다.
 - i. 최대 1Mbps의 비트 레이트로 컨트롤러 또는 대상으로 구성 가능
 - ii. 수신 및 전송을 위한 독립적인 8바이트 FIFO
 - iii. 이중 대상 주소 기능, 글리치 억제
 - iv. 독립 컨트롤러 및 대상 인터럽트 생성, DMA에 대한 하드웨어 지원
 - v. 중재, 클록 동기화, 다중 컨트롤러 지원을 통한 컨트롤러 작동
4. I2C 메시지 형식을 결정합니다. 일반적으로 I2C는 바이트 단위로 전송됩니다. 고급 통신을 달성하기 위해 사용자는 소프트웨어를 통해 프레임 통신을 구현할 수 있습니다. 필요한 경우 사용자는 특정 통신 프로토콜을 도입할 수도 있습니다. 예제 코드에서 메시지 형식은 < 55 AA ID1 ID2 ID3 ID4 Length Data1 Data2...>입니다. 사용자는 I2C를 통해 동일한 형식으로 데이터를 전송할 수 있습니다. 55 AA는 헤더입니다. ID 영역은 4바이트입니다. 길이 영역은 데이터 길이를 나타내는 1바이트입니다. 사용자가 I2C 패킷 양식을 수정해야 하는 경우 프레임 수집 및 구문 분석을 위한 코드도 수정해야 합니다.

표 21. I2C 패킷 양식

헤더	주소	데이터 길이	데이터
0x55 0xAA	4바이트	1바이트	(데이터 길이) 바이트

5. 변환해야 할 메시지, 메시지 변환 방법 등을 포함하여 브리지 구조를 결정합니다.
 - a. 브리지가 단방향인지 또는 양방향인지를 고려합니다. 일반적으로 각 인터페이스에는 수신 및 전송이라는 두 가지 기능이 있습니다. 일부 기능만 포함해야 하는지 고려합니다(예: I2C 수신 및 CAN 전송). 예제 코드에서 CAN-I2C 브리지는 양방향 구조입니다. I2C 대상의 수신 및 전송은 I2C 컨트롤러에 의해 제어되므로 I2C 대상은 I2C 컨트롤러로의 전송을 시작할 수 없습니다. 대상에서 컨트롤러로의 통신을 구현하기 위해 이 설계에 라인이 추가됩니다. 대상의 IO 풀다운은 전송할 정보가 있음을 컨트롤러에 알립니다.
 - b. 변환할 정보 및 해당 캐리어(변수, FIFO)를 고려합니다. 예제 코드에서 식별자, 데이터 및 데이터 길이는 한 인터페이스에서 다른 인터페이스로 변환됩니다. **그림 34**에 나와 있는 것처럼 코드에 정의된 두 개의 FIFO가 있습니다.

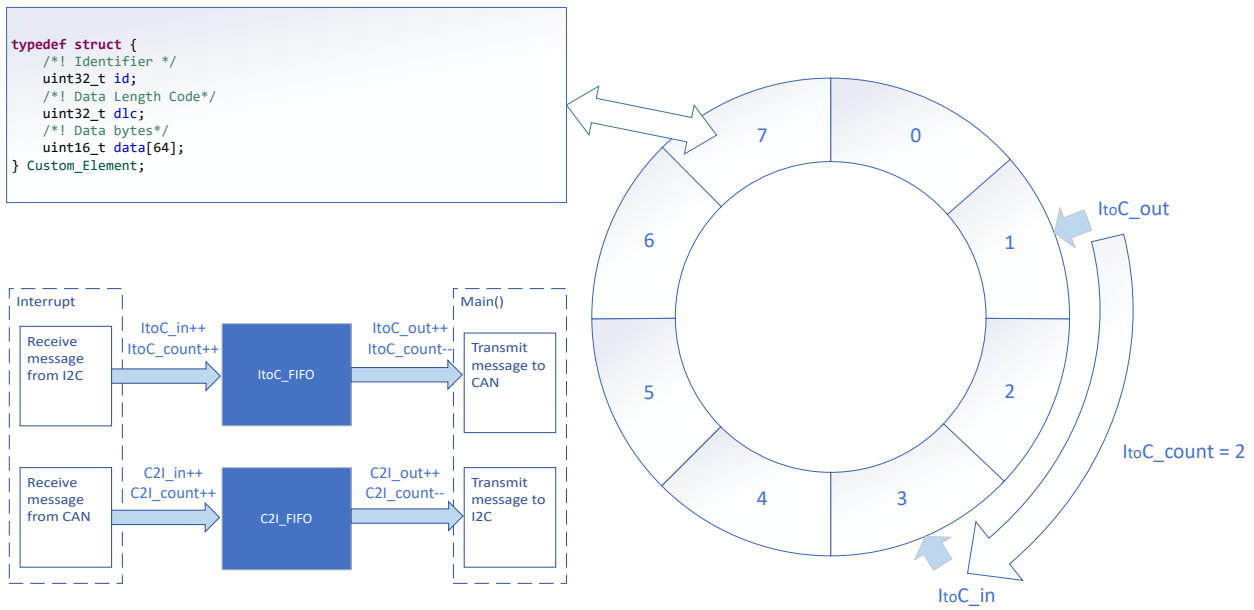


그림 34. 브리지 구조

6. (옵션) 우선 순위 설계, 정체 상황, 오류 처리 등을 고려합니다.

설계 고려 사항

1. 애플리케이션의 정보 흐름을 고려하여 각 인터페이스에서 수신 또는 전송할 정보, 따라야 할 프로토콜을 결정하고, 서로 다른 인터페이스를 연결하기 위한 적절한 정보 전송 캐리어를 설계합니다.
2. 먼저 인터페이스를 별도로 테스트한 후 전체 브리지 기능을 구현하는 것이 좋습니다. 또한 통신 실패, 과부하, 프레임 형식 오류 등과 같은 비정상적인 상황의 처리를 고려합니다.
3. 인터럽트를 통해 인터페이스 기능을 구현하여 적시에 통신을 보장하도록 하는 것이 좋습니다. 예제 코드에서 인터페이스 기능은 보통 인터럽트에 구현되고, 정보 전송은 main() 함수에서 완료됩니다.

소프트웨어 흐름도

그림 35에서는 메시지가 한 인터페이스에서 수신되고 다른 인터페이스에서 전송되는 방식을 설명하는 *CAN-I2C 브리지*에 대한 코드 흐름 다이어그램을 보여줍니다. *CAN-I2C 브리지*는 I2C에서 수신, CAN에서 수신, CAN을 통한 전송, I2C를 통한 전송 등 4개의 독립적인 작업으로 나눌 수 있습니다. 두 개의 FIFO는 양방향 메시지 전송 및 메시지 캐싱을 구현합니다.

I2C는 I2C 컨트롤러가 전송 및 수신을 제어하는 통신 방법입니다. 일반적으로 I2C 대상은 통신을 시작할 수 없습니다. I2C 대상-컨트롤러 통신의 경우 **그림 35**에 표시된 것처럼 I2C 대상은 메시지를 전송해야 하는 경우 IO를 풀다운할 수 있습니다.

그림 36에 표시된 것처럼 IO가 로우로 감지되면 I2C 컨트롤러는 IO 인터럽트에서 I2C 읽기 명령을 시작할 수 있습니다. 이 데모에서는 I2C를 I2C 대상 또는 컨트롤러로 구성할 수 있습니다.

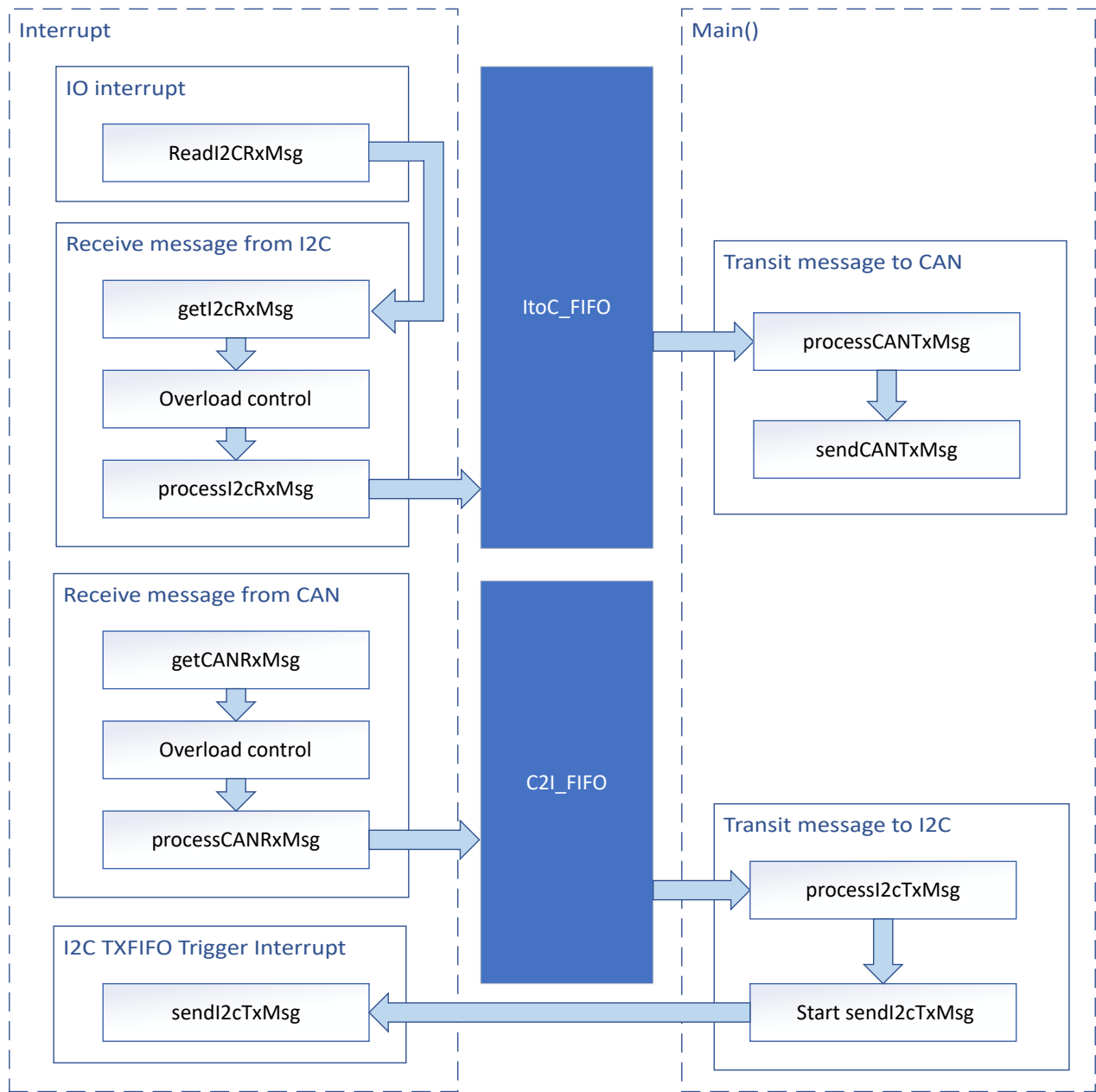


그림 35. CAN-I2C(I2C 컨트롤러) 브리지에 대한 애플리케이션 소프트웨어 흐름도

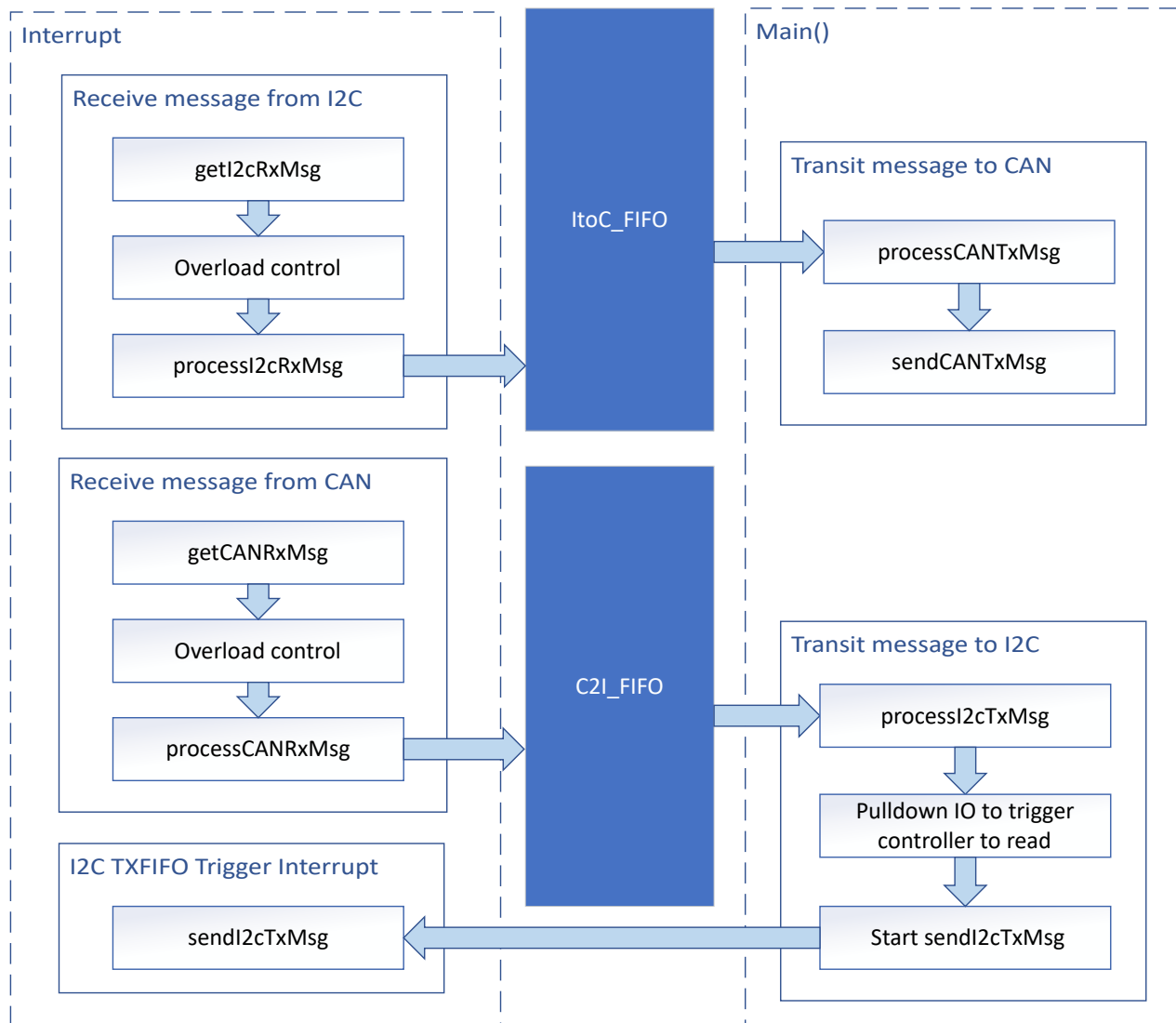


그림 36. CAN-I2C(I2C 대상) 브리지에 대한 애플리케이션 소프트웨어 흐름도

장치 구성

이 애플리케이션은 TI 시스템 구성 툴(SysConfig) 그래픽 인터페이스를 사용하여 CAN 및 I2C의 구성 코드를 생성합니다. 그래픽 인터페이스를 사용하여 장치 주변 기기를 구성하면 애플리케이션 프로토타입 제작 프로세스를 간소화할 수 있습니다.

그림 35에 설명된 코드는 그림 37에 표시된 것처럼 파일의 예제 코드에서 확인할 수 있습니다.

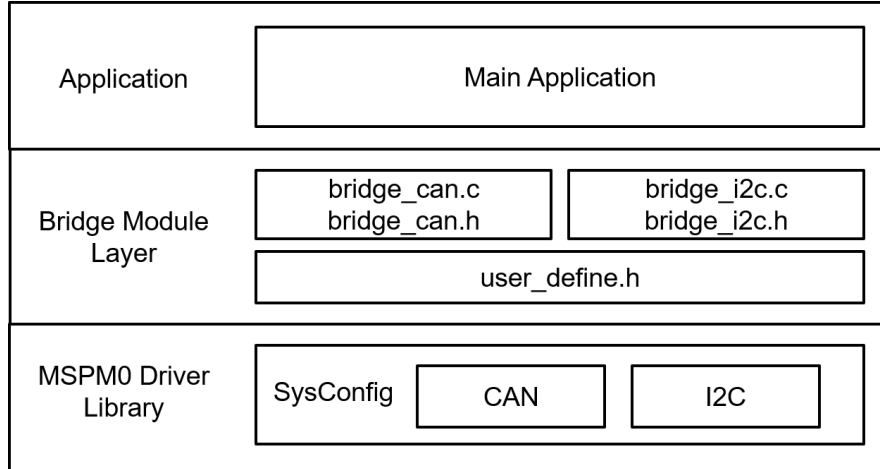


그림 37. 파일 구조

애플리케이션 코드

다음 코드 스니펫에서는 인터페이스 기능을 수정하는 위치를 보여 줍니다. 표의 함수는 서로 다른 파일로 분류됩니다. I2C 수신 및 전송을 위한 함수는 bridge_i2c.c 및 bridge_i2c.h에 포함되어 있습니다. CAN 수신 및 전송을 위한 함수는 bridge_can.c 및 bridge_can.h에 포함되어 있습니다. FIFO 요소의 구조는 user_define.h에 정의되어 있습니다.

사용자는 파일별로 함수를 쉽게 분리할 수 있습니다. 예를 들어 I2C 함수만 필요한 경우 사용자는 bridge_i2c.c 및 bridge_i2c.h를 예약하여 함수를 호출할 수 있습니다.

주변 기기 기본 구성은 MSPM0 SDK 및 DriverLib 문서를 참조하세요.

표 22. 함수 및 설명

작업	함수	설명	장소
I2C 수신	readI2CRxMsg_controller()	슬레이브에 읽기 요청 전송(I2C 마스터만 해당)	bridge_i2c.c
	getI2CRxMsg_controller()	수신한 I2C 메시지 받기(I2C 마스터만 해당)	bridge_i2c.h
	getI2CRxMsg_target()	수신한 I2C 메시지 받기(I2C 슬레이브만 해당)	
	processI2cRxMsg()	수신한 I2C 메시지 포맷을 변환하여 gI2C_RX_Element에 저장	
I2C 전송	processI2cTxMsg()	I2C를 통해 전송할 gI2C_TX_Element 포맷 변환	
	sendI2CTxMsg_controller()	I2C를 통해 메시지 전송(I2C 마스터만 해당)	
	sendI2CTxMsg_target()	I2C를 통해 메시지 전송(I2C 슬레이브만 해당)	
CAN 수신	getCANRxMsg()	수신한 CAN 메시지 가져오기	bridge_can.c
	processCANRxMsg()	수신한 CAN 메시지 포맷을 변환하여 메시지를 gCAN_RX_Element에 저장	bridge_can.h
CAN 전송	processCANTxMsg()	CAN을 통해 전송할 gCAN_TX_Element 포맷 변환	
	sendCANTxMsg()	CAN를 통해 메시지 전송	

Custom_Element는 user_define.h에 정의된 구조입니다. Custom_Element는 FIFO 요소의 구조, I2C/CAN 전송의 출력 요소 및 I2C/CAN 수신 입력 요소로 사용됩니다. 사용자는 필요에 따라 구조를 수정할 수 있습니다.

```

typedef struct {
    /*! Identifier */
    uint32_t id;
    /*! Data Length Code*/
    uint32_t dlc;
    /*! Data bytes*/
    uint16_t data[64];
} Custom_Element;

```

FIFO의 경우 FIFO로 사용되는 글로벌 변수 2개가 있습니다. 6개의 글로벌 변수가 FIFO를 추적하는 데 사용됩니다.

```

Custom_Element ItoC_FIFO[ItoC_FIFO_SIZE];
Custom_Element C2I_FIFO[C2I_FIFO_SIZE];
uint16_t ItoC_in = 0;
uint16_t ItoC_out = 0;
uint16_t ItoC_count = 0;
uint16_t C2I_in = 0;
uint16_t C2I_out = 0;
uint16_t C2I_count = 0;

```

결과

CAN 분석기를 사용하면 CAN 측에서 메시지를 전송하고 수신할 수 있습니다. 데모로서 예를 들면 두 개의 런치패드를 두 개의 CAN-I2C 브리지(1개의 I2C 마스터 및 1개의 I2C 슬레이브)로 사용하여 루프를 형성할 수 있습니다. CAN 분석기가 마스터 런치패드를 통해 CAN 메시지를 전송하면 분석기가 슬레이브 런치패드로부터 CAN 메시지를 수신할 수 있습니다.

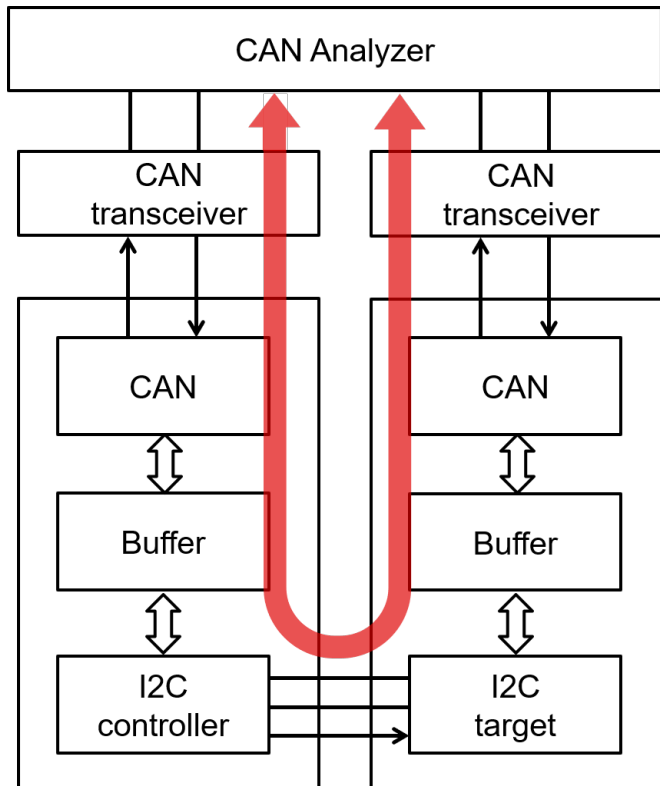


그림 38. 데모

Index	Time	Device	Channel	Frame ID	Type	CANType	RT	Len	Data
0	0.000000	Device0	0	0x1	StandardFrame	CANFD Accelerate	Tx	16	00 11 22 33 44 53 66 77 88 99 AA BB CC DD EE FF
1	0.000900	Device0	1	0x1	StandardFrame	CANFD Accelerate	Rx	16	00 11 22 33 44 53 66 77 88 99 AA BB CC DD EE FF
2	75.392500	Device0	1	0x2	StandardFrame	CANFD Accelerate	Tx	16	00 11 22 33 44 53 66 77 88 99 AA BB CC DD EE FF
3	75.393400	Device0	0	0x2	StandardFrame	CANFD Accelerate	Rx	16	00 11 22 33 44 53 66 77 88 99 AA BB CC DD EE FF
4	96.807600	Device0	1	0x3	StandardFrame	CANFD Accelerate	Tx	12	00 11 22 33 44 53 66 77 88 99 AA BB
5	96.808400	Device0	0	0x3	StandardFrame	CANFD Accelerate	Rx	12	00 11 22 33 44 53 66 77 88 99 AA BB
6	111.433500	Device0	0	0x4	StandardFrame	CANFD Accelerate	Tx	8	00 11 22 33 44 53 66 77
7	111.434100	Device0	1	0x4	StandardFrame	CANFD Accelerate	Rx	8	00 11 22 33 44 53 66 77
8	127.068700	Device0	1	0x5	StandardFrame	CANFD Accelerate	Tx	4	00 11 22 33
9	127.069200	Device0	0	0x5	StandardFrame	CANFD Accelerate	Rx	4	00 11 22 33
10	137.580700	Device0	0	0x6	StandardFrame	CANFD Accelerate	Tx	4	00 11 22 33
11	137.581200	Device0	1	0x6	StandardFrame	CANFD Accelerate	Rx	4	00 11 22 33
12	160.259200	Device0	0	0x7	StandardFrame	CANFD Accelerate	Tx	1	00
13	160.259700	Device0	1	0x7	StandardFrame	CANFD Accelerate	Rx	1	00

그림 39. 데모용 CAN 분석기에서 송수신하는 메시지

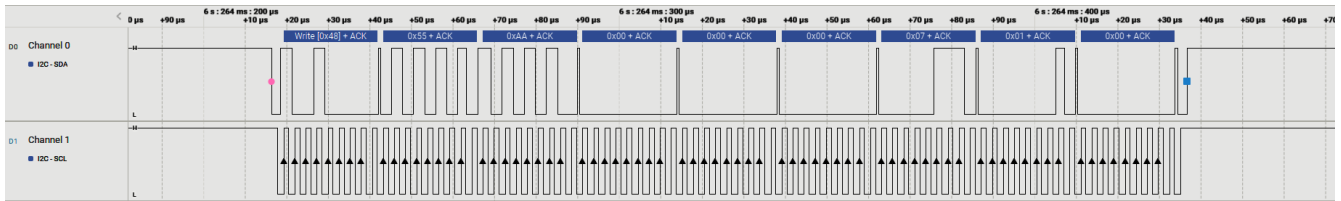


그림 40. 로직 분석기의 PC 터미널 프로그램

추가 리소스

- 텍사스 인스트루먼트, [MSPM0 SDK 다운로드](#)
- 텍사스 인스트루먼트, [SysConfig에 대해 자세히 알아보기](#)
- 텍사스 인스트루먼트, [MSPM0 G 시리즈 80MHz 마이크로컨트롤러](#), 기술 레퍼런스 매뉴얼
- 텍사스 인스트루먼트, [MSPM0G LaunchPad 개발 키트](#)
- 텍사스 인스트루먼트, [MSPM0 CAN 아카데미](#)
- 텍사스 인스트루먼트, [MSPM0 I2C 아카데미](#)

I2C-UART 서브시스템 설계

설계 설명

이 서브시스템은 I2C-UART 브리지 역할을 합니다. 이 서브시스템에서 MSPM0 장치는 I2C 대상 장치입니다. I2C 컨트롤러가 I2C 대상으로 전송되면 해당 대상은 수신된 모든 데이터를 수집합니다. 대상이 정지 상태를 감지하면 해당 대상은 UART 인터페이스를 사용하여 데이터를 전송합니다. I2C 컨트롤러가 브리지에서 읽기를 시도하면 브리지는 UART 장치에서 수신한 마지막 바이트를 전송합니다. I2C 컨트롤러가 2바이트를 읽으면 브리지는 UART 장치에서 수신한 마지막 바이트와 브리지에서 생성된 최근 오류 코드를 전송합니다.

MSPM0은 I2C SCL 및 SDA 라인으로 I2C 컨트롤러에 연결됩니다. MSPM0은 UART TX 및 RX 라인을 사용하여 UART 장치에도 연결됩니다.

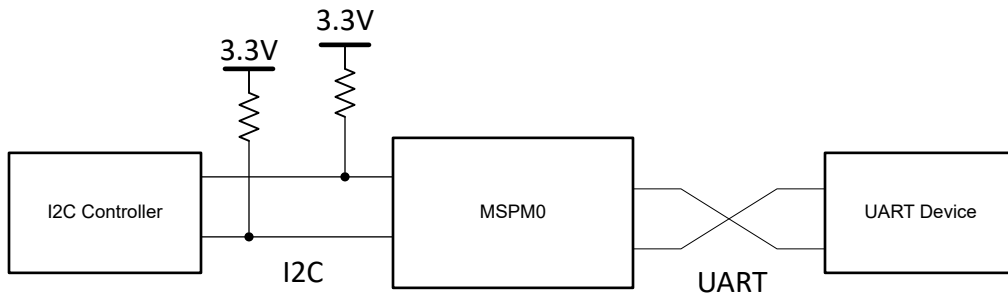


그림 41. 시스템 기능 블록 다이어그램

필요한 주변 기기

사용되는 주변 기기	참고
I2C	코드에서 I2C_INST라고 부름
UART	코드에서 UART_INST라고 부름

호환 가능 장치

필요한 주변 기기의 요구 사항에 따라 이 예제는 **호환 가능 장치**에 표시된 장치와 호환됩니다. 해당 EVM은 프로토타입 제작에 사용할 수 있습니다.

호환 가능 장치	EVM
MSPM0Lxxxx	LP-MSPM0L1306
MSPM0Gxxxx	LP-MSPM0G3507

설계 단계

1. SysConfig에서 I2C 모듈을 설정합니다. 장치를 대상 모드로 설정하고 RX FIFO 트리거, 시작 감지, 정지 감지, 대상 중재 손실, TX FIFO 언더플로, RX FIFO 오버플로 및 인터럽트 오버플로 인터럽트를 활성화합니다.
2. SysConfig에서 UART 모듈을 설정합니다. 장치에 대해 원하는 보 레이트를 선택합니다. 수신, 전송, 오버런 오류, 중단 오류, 프레임 오류, 패리티 오류, 잡음 오류 및 RX 시간 초과를 활성화합니다.

설계 고려 사항

1. 애플리케이션 코드에서 I2C_MAX_PACKET_SIZE가 전송할 패킷을 포함할 수 있을 만큼 충분히 큰지 확인합니다.
2. 사용 중인 I2C 모듈에 적합한 풀업 저항 값을 선택해야 합니다. 일반적으로 10kΩ은 100kHz에 적합합니다. I2C 버스 속도가 높을수록 더 낮은 값의 풀업 저항이 필요합니다. 400kHz 통신의 경우 4.7kΩ에 가까운 저항을 사용합니다.
3. UART 보 레이트를 높이려면 *대상 보 레이트*라고 표시된 SysConfig UART 탭에서 값을 조정합니다. 이 아래에서 대상 보 레이트를 반영하기 위해 계산된 보 레이트 변화를 관찰합니다. 이는 사용 가능한 클럭 및 분할기를 사용하여 계산됩니다.
4. 오류 플래그를 확인하고 적절하게 처리합니다. UART 및 I2C 주변 기기는 모두 진단에 도움이 되는 오류 인터럽트를 발생시킬 수 있습니다. 손쉬운 디버깅을 위해 이 서브시스템은 열거형 및 전역 변수를 사용하여 오류 코드가 발생할 때 오류 코드를 저장합니다. 실제 애플리케이션에서는 오류로 인해 프로젝트가 중단되지 않도록 코드의 오류를 처리합니다.

소프트웨어 흐름도

그림 42에서는 이 예제에 대한 코드 흐름 다이어그램을 보여주고, 장치가 수신된 I2C 데이터로 데이터 버퍼를 채운 후 UART를 통해 데이터를 전송하는 방법을 설명합니다.

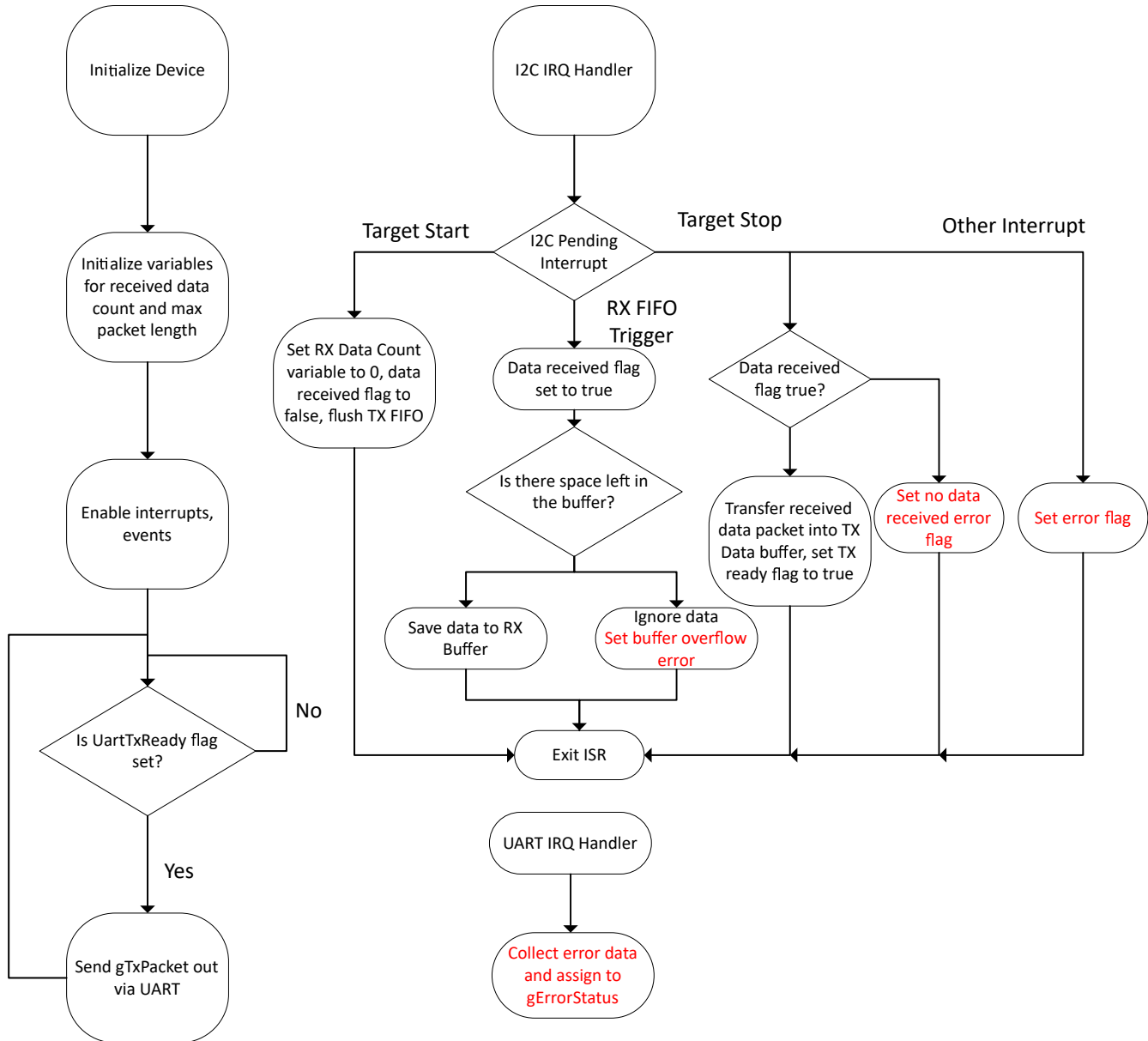


그림 42. 애플리케이션 소프트웨어 흐름도

장치 구성

이 애플리케이션은 TI 시스템 구성 툴(SysConfig) 그래픽 인터페이스를 사용하여 장치 주변 기기의 구성 코드를 생성합니다. 그래픽 인터페이스를 사용하여 장치 주변 기기를 구성하면 애플리케이션 프로토타입 제작 프로세스를 간소화할 수 있습니다.

그림 42에 설명된 코드는 `i2c_to_uart_bridge.c` 파일의 `main()` 시작 부분에서 확인할 수 있습니다.

애플리케이션 코드

이 애플리케이션은 수신된 데이터와 전송할 데이터에 메모리를 할당해야 합니다. 또한 이 애플리케이션은 수신 및 전송된 데이터의 양을 파악해야 합니다. 수신되는 데이터가 완료되고 UART를 통해 전송할 준비가 되었는지 여부를 확인하기 위해 플래그가 필요합니다. 또한 오류 코드를 저장할 변수와 함께 오류 코드에 사용하는 열거형도 있습니다. 버퍼, 카운터, 열거형 및 플래그의 초기화는 아래에 표시되어 있습니다.

```
#include "ti_msp_dl_config.h"

/* Maximum size of TX packet */
#define I2C_TX_MAX_PACKET_SIZE (1)

/* Maximum size of RX packet */
#define I2C_RX_MAX_PACKET_SIZE (16)

/* Data sent to Controller in response to Read transfer */
uint8_t gTxPacket[I2C_TX_MAX_PACKET_SIZE] = {0x00};

/* Counters for TX length and bytes sent */
uint32_t gTxLen, gTxCount;

/* Data received from Controller during a Write transfer */
uint8_t gRxPacket[I2C_RX_MAX_PACKET_SIZE];
/* Counters for TX length and bytes sent */
uint32_t gRxLen, gRxCount;

enum error_codes{
    NO_ERROR,
    DATA_BUFFER_OVERFLOW,
    RX_FIFO_FULL,
    NO_DATA_RECEIVED,
    I2C_TARGET_TXFIFO_UNDERFLOW,
    I2C_TARGET_RXFIFO_OVERFLOW,
    I2C_TARGET_ARBITRATION_LOST,
    I2C_INTERRUPT_OVERFLOW,
    UART_OVERRUN_ERROR,
    UART_BREAK_ERROR,
    UART_PARITY_ERROR,
    UART_FRAMING_ERROR,
    UART_RX_TIMEOUT_ERROR
};

uint8_t gErrorStatus = NO_ERROR;

/* Buffer to hold data received from UART device */
uint8_t gUARTRxData = 0;
/* Flags */
bool gUartTxReady = false; /* Flag to start UART transfer */
bool gUartRxDone = false; /* Flag to indicate UART data has been received */
```

애플리케이션 코드의 본문은 상대적으로 짧습니다. 먼저 장치와 주변 기기가 초기화됩니다. 그러면 인터럽트와 이벤트가 활성화됩니다. 카운터 값도 초기화됩니다. 마지막으로 메인 루프에 도달하면 수신된 데이터가 UART를 통해 다시 전송할 준비가 되었는지 여부를 감지하기 위해 플래그가 폴링됩니다.

```
int main(void)
{
    SYSCFG_DL_init();

    gTxCount = 0;
    gTxLen = I2C_TX_MAX_PACKET_SIZE;
    DL_I2C_enableInterrupt(I2C_INST, DL_I2C_INTERRUPT_TARGET_TXFIFO_TRIGGER);

    /* Initialize variables to receive data inside RX ISR */
    gRxCount = 0;
    gRxLen = I2C_RX_MAX_PACKET_SIZE;

    NVIC_EnableIRQ(I2C_INST_INT_IRQN);
    NVIC_EnableIRQ(UART_INST_INT_IRQN);

    while (1) {
```

```

        if(gUartTxReady){
            gUartTxReady = false;
            for(int i = 0; i < gRxCOUNT; i++){
                /* Transmit data out via UART and wait until transfer is complete */
                DL_UART_Main_transmitDataBlocking(UART_INST, gTxPacket[i]);
            }
        }
    }
}

```

이 코드의 다음 부분은 I2C IRQ 처리기입니다. 이 코드는 데이터 수집을 시작하고 중지하는 데 사용됩니다. 그런 다음, 데이터가 수신될 때 코드가 데이터를 저장합니다. 보류 중인 인터럽트가 감지된 I2C 시작 조건인 경우 장치가 카운터 변수를 초기화합니다. 보류 중인 인터럽트가 RX FIFO에 사용할 수 있는 데이터가 있음을 알려주는 경우 장치가 데이터 버퍼에 공간이 남아 있는지 확인합니다. 공간이 있으면 수신된 값이 저장됩니다. 더 이상 공간이 없으면 수신된 값이 무시됩니다. 보류 중인 인터럽트가 TX FIFO 트리거인 경우 장치가 전송된 바이트 수를 확인합니다. 장치가 이미 바이트를 전송한 경우 FIFO는 가장 최근에 보고된 오류 코드로 채워집니다. 보류 중인 인터럽트가 I2C 정지 조건인 경우 장치는 데이터가 수신되었는지 확인합니다. 데이터가 수신되었으면 수신된 데이터 버퍼가 전송 데이터 버퍼로 복사되고 UART TX 준비 플래그가 true로 설정됩니다. 수신된 데이터가 없으면 장치는 아무것도 전송하지 않습니다. 또한 이 ISR은 gErrorStatus 변수에 적절한 오류 코드를 할당하여 I2C 오류 인터럽트를 처리합니다.

```

void I2C_INST_IRQHandler(void)
{
    static bool dataRx = false;

    switch (DL_I2C_getPendingInterrupt(I2C_INST)) {
        case DL_I2C_IIDX_TARGET_START:
            /* Initialize RX or TX after Start condition is received */
            gTxCount = 0;
            gRxCount = 0;
            dataRx = false;
            /* Flush TX FIFO to refill it */
            DL_I2C_flushTargetTXFIFO(I2C_INST);
            break;
        case DL_I2C_IIDX_TARGET_RXFIFO_TRIGGER:
            /* Store received data in buffer */
            dataRx = true;
            while (DL_I2C_isTargetRXFIFOEmpty(I2C_INST) != true) {
                if (gRxCount < gRxLen) {
                    gRxPacket[gRxCount++] = DL_I2C_receiveTargetData(I2C_INST);
                } else {
                    /* Prevent overflow and just ignore data */
                    DL_I2C_receiveTargetData(I2C_INST);
                }
            }
            break;
        case DL_I2C_IIDX_TARGET_TXFIFO_TRIGGER:
            /* Fill TX FIFO if there are more bytes to send */
            if (gTxCount < gTxLen) {
                gTxCount += DL_I2C_fillTargetTXFIFO(
                    I2C_INST, &gUARTRxData, (gTxLen - gTxCount));
            } else {
                /*
                 * Fill FIFO with error status after sending latest received
                 * byte
                 */
                while (DL_I2C_transmitTargetDataCheck(I2C_INST, gErrorStatus) != false)
                    ;
            }
            break;
        case DL_I2C_IIDX_TARGET_STOP:
            /* If data was received, echo to TX buffer */
            if (dataRx == true) {
                for (uint16_t i = 0;
                    (i < gRxCount) && (i < I2C_TX_MAX_PACKET_SIZE); i++) {
                    gTxPacket[i] = gRxPacket[i];
                    DL_I2C_flushTargetTXFIFO(I2C_INST);
                }
                dataRx = false;
            }
    }
}

```

```

    }
    /* Set flag to indicate data ready for UART TX */
    gUartTxReady = true;
    break;
case DL_I2C_IIDX_TARGET_RX_DONE:
    /* Not used for this example */
case DL_I2C_IIDX_TARGET_RXFIFO_FULL:
    /* Not used for this example */
case DL_I2C_IIDX_TARGET_GENERAL_CALL:
    /* Not used for this example */
case DL_I2C_IIDX_TARGET_EVENT1_DMA_DONE:
    /* Not used for this example */
case DL_I2C_IIDX_TARGET_EVENT2_DMA_DONE:
    /* Not used for this example */
case DL_I2C_IIDX_TARGET_TXFIFO_UNDERFLOW:
    gErrorStatus = I2C_TARGET_TXFIFO_UNDERFLOW;
    break;
case DL_I2C_IIDX_TARGET_RXFIFO_OVERFLOW:
    gErrorStatus = I2C_TARGET_RXFIFO_OVERFLOW;
    break;
case DL_I2C_IIDX_TARGET_ARBITRATION_LOST:
    gErrorStatus = I2C_TARGET_ARBITRATION_LOST;
    break;
case DL_I2C_IIDX_INTERRUPT_OVERFLOW:
    gErrorStatus = I2C_INTERRUPT_OVERFLOW;
    break;
default:
    break;
}
}
}

```

이 예제에서 마지막 코드 조각은 UART IRQ 처리기입니다. UART IRQ 처리기는 수신된 데이터를 저장하고 오류를 확인하는 데만 사용됩니다. UART RX 인터럽트가 보류 중이면 장치는 수신된 데이터를 버퍼, gUARTRxData에 저장한 다음, 저장된 새 RX 데이터가 있음을 나타내는 플래그를 설정합니다. UART 오류가 발생하면 이 ISR이 실행되어 gErrorStatus에 올바른 오류 코드를 할당합니다.

```

void UART_INST_IRQHandler(void)
{
    switch (DL_UART_Main_getPendingInterrupt(UART_INST)) {
        case DL_UART_MAIN_IIDX_RX:
            DL_UART_Main_receiveDataCheck(UART_INST, &gUARTRxData);
            gUartRxDone = true;
            break;
        case DL_UART_INTERRUPT_OVERRUN_ERROR:
            gErrorStatus = UART_OVERRUN_ERROR;
            break;
        case DL_UART_INTERRUPT_BREAK_ERROR:
            gErrorStatus = UART_BREAK_ERROR;
            break;
        case DL_UART_INTERRUPT_PARITY_ERROR:
            gErrorStatus = UART_PARITY_ERROR;
            break;
        case DL_UART_INTERRUPT_FRAMING_ERROR:
            gErrorStatus = UART_FRAMING_ERROR;
            break;
        case DL_UART_INTERRUPT_RX_TIMEOUT_ERROR:
            gErrorStatus = UART_RX_TIMEOUT_ERROR;
            break;
        default:
            break;
    }
}

```

추가 리소스

1. 텍사스 인스트루먼트, [MSPM0 SDK 다운로드](#)
2. 텍사스 인스트루먼트, [SysConfig에 대해 자세히 알아보기](#)
3. 텍사스 인스트루먼트, [MSPM0L LaunchPad™](#)

4. 텍사스 인스트루먼트, [MSPM0G LaunchPad™](#)
5. 텍사스 인스트루먼트, [MSPM0 I2C 아카데미](#)
6. 텍사스 인스트루먼트, [MSPM0 UART 아카데미](#)

CAN-SPI 브리지

설계 설명

이 서브시스템은 CAN-SPI 브리지를 구축하는 방법을 보여줍니다. CAN-SPI 브리지를 사용하면 장치가 한 인터페이스에서 정보를 송신 또는 수신하고 다른 인터페이스에서 정보를 수신 또는 전송할 수 있습니다. [이 예제의 코드를 다운로드하세요.](#) 서브시스템은 컨트롤러 모드 또는 주변 기기 모드에서 작동하도록 SPI를 지원합니다.

그림 43에서는 이 서브시스템의 기능 다이어그램을 보여줍니다.

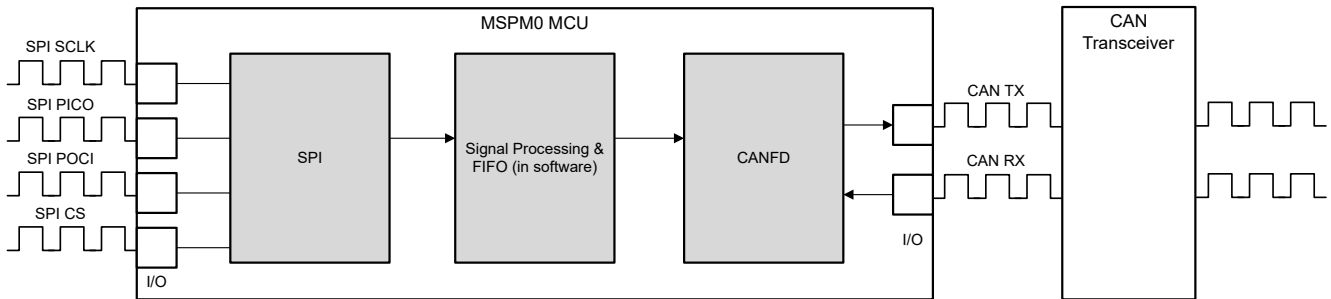


그림 43. 서브시스템 기능 블록 다이어그램

필요한 주변 기기

이 애플리케이션은 CANFD 및 SPI가 필요합니다.

표 23. 필요한 주변 기기

하위 블록 기능	사용되는 주변 기기	참고
CAN 인터페이스	(1x) CANFD	코드에서 <code>MCANO_INST</code> 라고 부름
SPI 인터페이스	(1x) SPI	코드에서 <code>SPI_0_INST</code> 라고 부름

호환 가능 장치

표 23의 요구 사항에 따라 이 예제는 표 24의 장치와 호환됩니다. 해당 EVM은 프로토타입 제작에 사용할 수 있습니다.

표 24. 호환 가능 장치

호환 가능 장치	EVM
MSPM0G35xx	LP-MSPM0G3507

설계 단계

- CAN 모드, 비트 타이밍, 메시지 RAM 구성 등을 포함한 CAN 인터페이스의 기본 설정을 결정합니다. 고정된 설정과 애플리케이션에서 변경되는 설정을 고려합니다. 예제 코드에서 CANFD는 250kbit/s 중재 비율 및 2Mbit/s 데이터 전송률로 사용됩니다.
 - CAN-FD 주변 기기의 주요 특징은 다음과 같습니다.
 - ECC가 있는 전용 1KB 메시지 SRAM
 - 구성 가능한 전송 FIFO, 전송 대기열 및 이벤트 FIFO(최대 32개 요소)
 - 최대 32개의 전용 전송 버퍼 및 64개의 전용 수신 버퍼가 있습니다. 2개의 구성 가능한 수신 FIFO(각각 최대 64개 요소)

- iv. 최대 128개의 필터 요소
 - b. CANFD 모드가 활성화된 경우:
 - i. 64바이트 CAN-FD 프레임을 완벽하게 지원
 - ii. 최대 8Mbit/s의 비트 레이트
 - c. CANFD 모드가 비활성화된 경우:
 - i. 8바이트 클래식 CAN 프레임을 완벽하게 지원
 - ii. 최대 1Mbit/s의 비트 레이트
2. 데이터 길이, 비트 레이트 전환, 식별자, 데이터 등을 포함한 CAN 프레임을 결정합니다. 고정된 부품과 애플리케이션에서 변경해야 하는 부분을 고려합니다. 예제 코드에서 식별자, 데이터 길이 및 데이터는 서로 다른 프레임에서 변경되는 반면 다른 항목은 고정되어 있습니다. 프로토콜 통신이 필요한 경우 사용자는 코드를 수정해야 합니다.

```

/**
 * @brief Structure for MCAN Rx Buffer element.
 */
typedef struct {
    /* Identifier */
    uint32_t id;
    /* Remote Transmission Request
     * 0 = Received frame is a data frame
     * 1 = Received frame is a remote frame
     */
    uint32_t rtr;
    /* Extended Identifier
     * 0 = 11-bit standard identifier
     * 1 = 29-bit extended identifier
     */
    uint32_t xtd;
    /* Error State Indicator
     * 0 = Transmitting node is error active
     * 1 = Transmitting node is error passive
     */
    uint32_t esi;
    /* Rx Timestamp */
    uint32_t rxts;
    /* Data Length Code
     * 0-8 = CAN + CAN FD: received frame has 0-8 data bytes
     * 9-15 = CAN: received frame has 8 data bytes
     * 9-15 = CAN FD: received frame has 12/16/20/24/32/48/64 data bytes
     */
    uint32_t dlc;
    /* Bit Rat Switching
     * 0 = Frame received without bit rate switching
     * 1 = Frame received with bit rate switching
     */
    uint32_t brs;
    /* FD Format
     * 0 = Standard frame format
     * 1 = CAN FD frame format (new DLC-coding and CRC)
     */
    uint32_t fdf;
    /* Filter Index */
    uint32_t fidx;
    /* Accepted Non-matching Frame
     * 0 = Received frame matching filter index FIDX
     * 1 = Received frame did not match any Rx filter element
     */
    uint32_t anmf;
    /* Data bytes.
     * Only first dlc number of bytes are valid.
     */
    uint16_t data[DL_MCAN_MAX_PAYLOAD_BYTES];
} DL_MCAN_RxBufElement;

```

3. SPI 모드, 비트 레이트, 프레임 크기, FIFO 등을 포함한 SPI 인터페이스의 기본 설정을 결정합니다. 고정된 설정과 애플리케이션에서 변경되는 설정을 고려합니다. 예제 코드에서 SPI는 컨트롤러 또는 주변 기기로 설정할 수 있습니다. SPI는 컨트롤러 모드에서 500k 비트 레이트로 작동합니다.
 - a. SPI의 주요 특징은 다음과 같습니다.
 - i. 컨트롤러 또는 주변 기기로 구성 가능
 - ii. 프로그래밍 가능 클럭 비트 레이트 및 프리스케일러
 - iii. 별도의 전송(TX) 및 수신(RX) FIFO(선입 선출 버퍼)
 - iv. PACKEN 기능 및 단일 비트 패리티 지원
 - v. 프로그래밍 가능 데이터 프레임 크기 및 프로그래밍 가능 SPI 모드
 - vi. 전송 및 수신 FIFO, 오버런 및 시간 초과 인터럽트, DMA 완료에 대한 인터럽트

4. SPI 프레임을 결정합니다. 일반적으로 SPI는 바이트 단위로 전송됩니다. 고급 통신을 달성하기 위해 사용자는 소프트웨어를 통해 프레임 통신을 구현할 수 있습니다. 필요한 경우 사용자는 특정 통신 프로토콜을 도입할 수도 있습니다. 예제 코드에서 메시지 형식은 < 55 AA ID1 ID2 ID3 ID4 Length Data1 Data2...>입니다. 사용자는 동일한 형식으로 데이터를 입력하여 터미널에서 CAN 버스로 데이터를 전송할 수 있습니다. 55 AA는 헤더입니다. ID 영역은 4바이트입니다. 길이 영역은 데이터 길이를 나타내는 1바이트입니다. 사용자가 SPI 프레임을 수정해야 하는 경우 프레임 수집 및 구문 분석에 대한 코드도 수정해야 합니다.

표 25. SPI 프레임 양식

헤더	주소	데이터 길이	데이터
0x55 0xAA	4바이트	1바이트	(데이터 길이) 바이트

5. 변환해야 할 메시지, 메시지 변환 방법 등을 포함하여 브리지 구조를 결정합니다.
 - a. 브리지가 단방향인지 또는 양방향인지를 고려합니다. 일반적으로 각 인터페이스에는 수신 및 전송이라는 두 가지 기능이 있습니다. 일부 기능만 포함해야 하는지 고려합니다(예: SPI 수신 및 CAN 전송). 예제 코드에서 CAN-SPI 브리지는 양방향 구조입니다.
 - b. 변환할 정보 및 해당 캐리어(변수, FIFO)를 고려합니다. 예제 코드에서 식별자, 데이터 및 데이터 길이는 한 인터페이스에서 다른 인터페이스로 변환됩니다. **그림 44**에 나와 있는 것처럼 코드에 정의된 두 개의 FIFO가 있습니다.

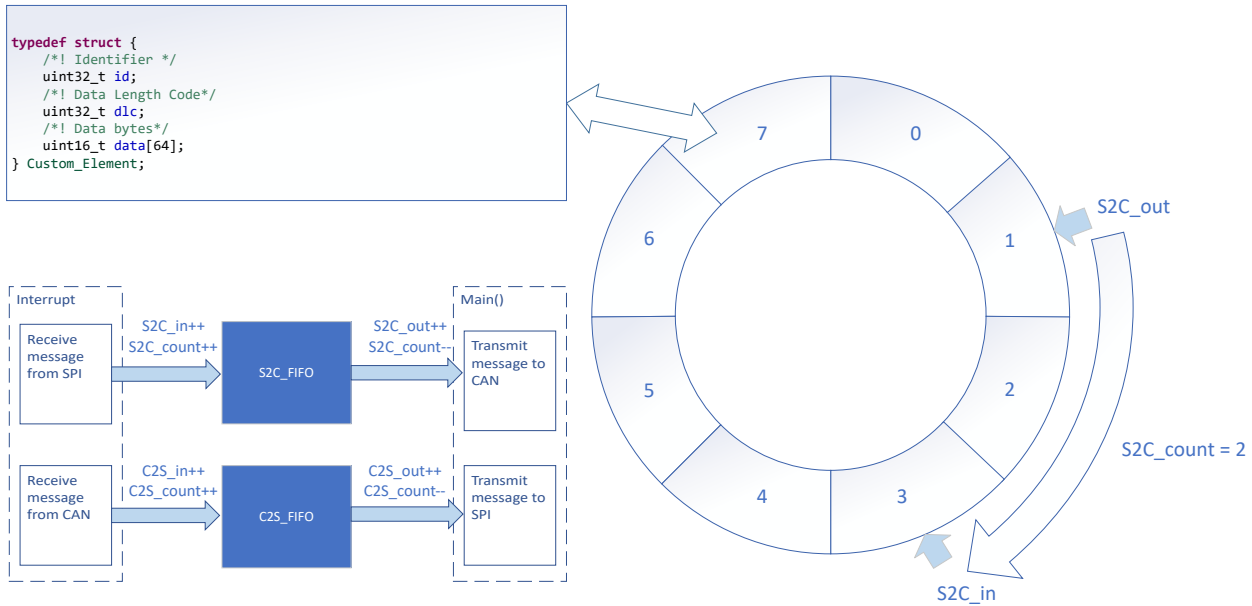


그림 44. 브리지 구조

6. (옵션) 우선 순위 설계, 정체 상황, 오류 처리 등을 고려합니다.

설계 고려 사항

1. 애플리케이션의 정보 흐름을 고려하여 각 인터페이스에서 수신 또는 전송할 정보, 따라야 할 프로토콜을 결정하고, 서로 다른 인터페이스를 연결하기 위한 적절한 정보 전송 캐리어를 설계합니다.
2. 먼저 인터페이스를 별도로 테스트한 후 전체 브리지 기능을 구현하는 것이 좋습니다. 또한 통신 실패, 과부하, 프레임 형식 오류 등과 같은 비정상적인 상황의 처리를 고려합니다.
3. 인터럽트를 통해 인터페이스 기능을 구현하여 적시에 통신을 보장하도록 하는 것이 좋습니다. 예제 코드에서 인터페이스 기능은 보통 인터럽트에 구현되고, 정보 전송은 main() 함수에서 완료됩니다.

소프트웨어 흐름도

다음 그림은 메시지가 한 인터페이스에서 수신되고 다른 인터페이스에서 전송되는 방식을 설명하는 CAN-SPI 브리지에 대한 코드 흐름 다이어그램을 보여줍니다. CAN-SPI 브리지는 SPI에서 수신, CAN에서 수신, CAN을 통한 전송, SPI를 통한 전송 등 4개의 독립적인 작업으로 나눌 수 있습니다. 두 개의 FIFO는 양방향 메시지 전송 및 메시지 캐싱을 구현합니다.

SPI는 동시에 송수신하는 통신 방법입니다. 컨트롤러가 바이트를 전송하기 시작하면 컨트롤러는 바이트를 수신할 것으로 예상합니다. 이 문서의 설계에서 SPI RX 인터럽트는 SPI 수신뿐 아니라 TX 데이터를 SPI TX FIFO에 채우는 데에도 사용됩니다. SPI가 컨트롤러 모드에서 작동하는 경우 SPI TX FIFO가 데이터에 의해 저장된 후 즉시 SPI 통신이 시작됩니다. SPI가 주변 기기 모드에서 작동하는 경우 SPI는 데이터가 저장된 후 컨트롤러가 통신을 시작하길 기다릴 수 있습니다. 이 데모에서 사용자는 SPI 모드를 선택할 수 있습니다.

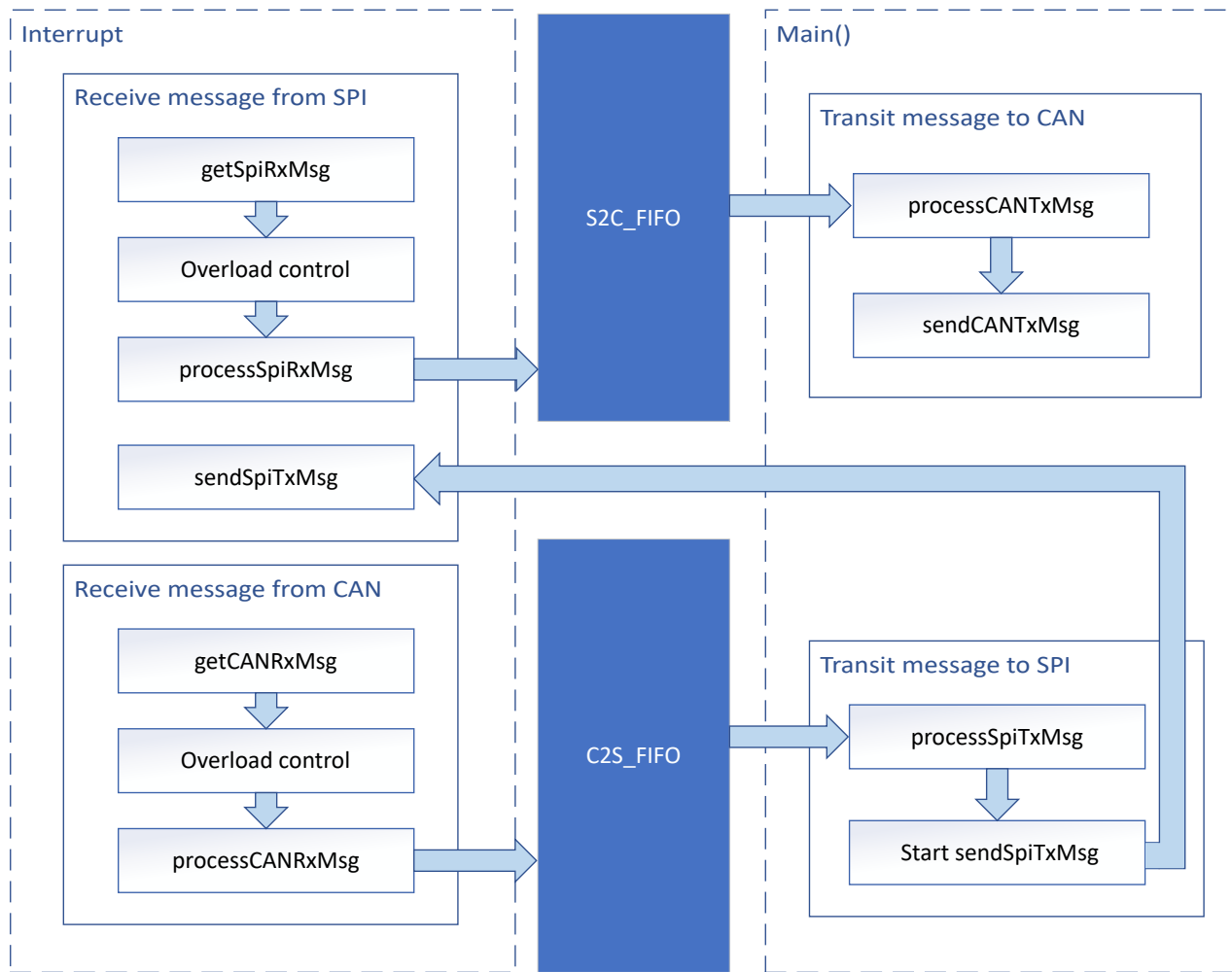


그림 45. 애플리케이션 소프트웨어 흐름도

장치 구성

이 애플리케이션은 TI 시스템 구성 툴(SysConfig) 그래픽 인터페이스를 사용하여 CAN 및 SPI의 구성 코드를 생성합니다. 그래픽 인터페이스를 사용하여 장치 주변 기기를 구성하면 애플리케이션 프로토타입 제작 프로세스를 간소화할 수 있습니다.

사용자는 SysConfig에서 SPI를 컨트롤러 또는 주변 기기로 구성할 수 있습니다.

그림 45에 설명된 코드는 그림 46에 표시된 것처럼 파일의 예제 코드에서 확인할 수 있습니다.

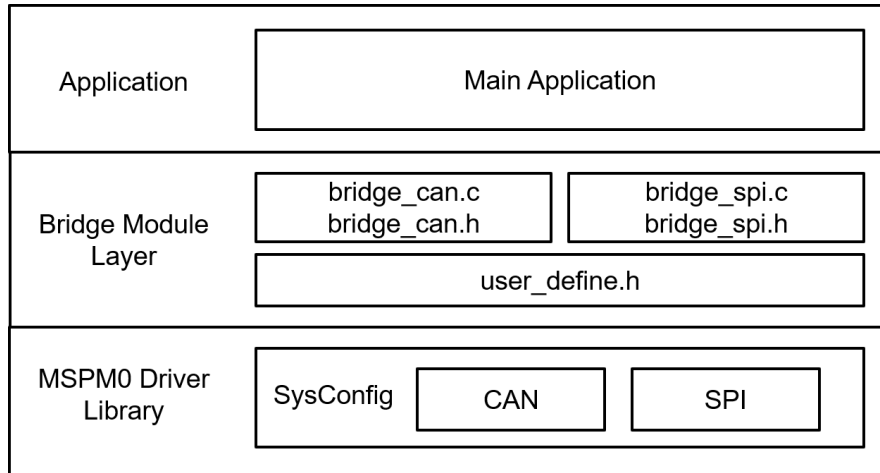


그림 46. 파일 구조

애플리케이션 코드

다음 코드 스니펫에서는 인터페이스 기능을 수정하는 위치를 보여 줍니다. 표의 함수는 서로 다른 파일로 분류됩니다. SPI 수신 및 전송을 위한 함수는 bridge_spi.c 및 bridge_spi.h에 포함되어 있습니다. CAN 수신 및 전송을 위한 함수는 bridge_can.c 및 bridge_can.h에 포함되어 있습니다. FIFO 요소의 구조는 user_define.h에 정의되어 있습니다.

사용자는 파일별로 함수를 쉽게 분리할 수 있습니다. 예를 들어 SPI 함수만 필요한 경우 사용자는 bridge_spi.c 및 bridge_spi.h를 예약하여 함수를 호출할 수 있습니다.

주변 기기 기본 구성은 MSPM0 SDK 및 DriverLib 문서를 참조하세요.

표 26. 함수 및 설명

작업	함수	설명	장소
SPI 수신	getSpiRxMsg()	수신한 SPI 메시지 가져오기	bridge_spi.c bridge_spi.h
	processSpiRxMsg()	수신한 SPI 메시지 포맷을 변환하여 gSPI_RX_Element에 저장	
SPI 전송	processSpiTxMsg()	SPI를 통해 전송할 gSPI_TX_Element 포맷 변환	
	sendSpiTxMsg()	SPI를 통해 메시지 전송	
CAN 수신	getCANRxMsg()	수신한 CAN 메시지 가져오기	bridge_can.c bridge_can.h
	processCANRxMsg()	수신한 CAN 메시지 포맷을 변환하여 메시지를 gCAN_RX_Element에 저장	
CAN 전송	processCANTxMsg()	CAN을 통해 전송할 gCAN_TX_Element 포맷 변환	
	sendCANTxMsg()	CAN를 통해 메시지 전송	

Custom_Element는 user_define.h에 정의된 구조입니다. Custom_Element는 FIFO 요소의 구조, SPI/CAN 전송의 출력 요소 및 SPI/CAN 수신 입력 요소로 사용됩니다. 사용자는 필요에 따라 구조를 수정할 수 있습니다.

```

typedef struct {
    /*! Identifier */
    uint32_t id;
    /*! Data Length Code*/
    uint32_t dlc;
    /*! Data bytes*/
    uint16_t data[64];
} Custom_Element;

```

FIFO의 경우 FIFO로 사용되는 글로벌 변수 2개가 있습니다. 6개의 글로벌 변수가 FIFO를 추적하는 데 사용됩니다.

```

Custom_Element S2C_FIFO[S2C_FIFO_SIZE];
Custom_Element C2S_FIFO[C2S_FIFO_SIZE];
uint16_t S2C_in = 0;
uint16_t S2C_out = 0;
uint16_t S2C_count = 0;
uint16_t C2S_in = 0;
uint16_t C2S_out = 0;
uint16_t C2S_count = 0;

```

결과

CAN 분석기를 사용하면 CAN 측에서 메시지를 전송하고 수신할 수 있습니다. 데모로서 예를 들면 두 개의 런치패드를 두 개의 CAN-SPI 브리지(1개의 SPI 컨트롤러 및 1개의 SPI 주변 기기)로 사용하여 루프를 형성할 수 있습니다. CAN 분석기가 컨트롤러 런치패드를 통해 CAN 메시지를 전송하면 주변 기기 런치패드로부터 CAN 메시지를 수신합니다.

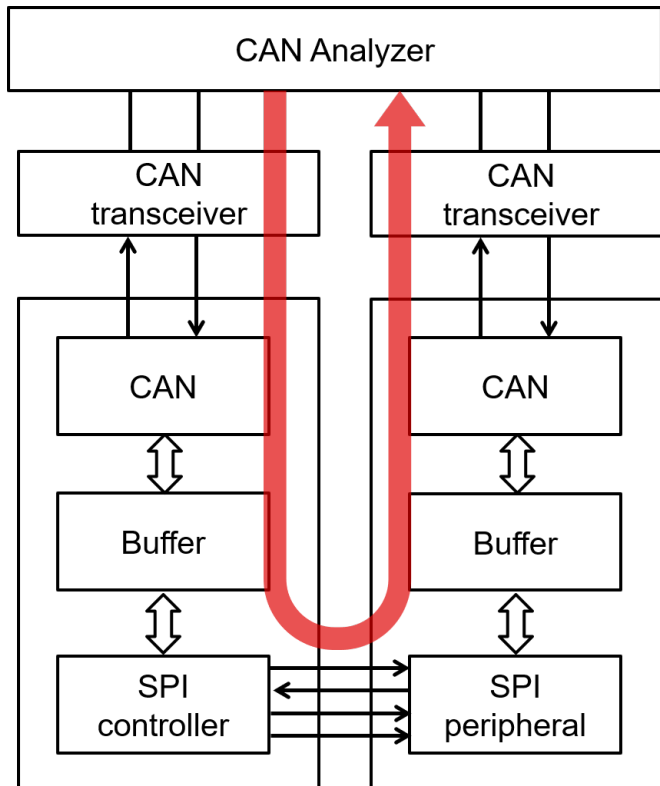


그림 47. 데모

Index	Time	Device	Channel	Frame ID	Type	CANType	RT	Len	Data
					ALL	ALL	ALI		
0	0.000000	Device0	0	0x1	StandardFrame	CANFD Accelerate	Tx	1	00
1	0.000300	Device0	1	0x1	StandardFrame	CANFD Accelerate	Rx	1	00
2	18.323700	Device0	0	0x2	StandardFrame	CANFD Accelerate	Tx	2	00 11
3	18.324100	Device0	1	0x2	StandardFrame	CANFD Accelerate	Rx	2	00 11
4	33.411500	Device0	0	0x3	StandardFrame	CANFD Accelerate	Tx	4	00 11 22 33
5	33.411900	Device0	1	0x3	StandardFrame	CANFD Accelerate	Rx	4	00 11 22 33
6	50.216400	Device0	0	0x4	StandardFrame	CANFD Accelerate	Tx	8	00 11 22 33 44 55 66 77
7	50.216900	Device0	1	0x4	StandardFrame	CANFD Accelerate	Rx	8	00 11 22 33 44 55 66 77
8	67.378700	Device0	0	0x5	StandardFrame	CANFD Accelerate	Tx	12	00 11 22 33 44 55 66 77 88 99 AA BB
9	67.379400	Device0	1	0x5	StandardFrame	CANFD Accelerate	Rx	12	00 11 22 33 44 55 66 77 88 99 AA BB
10	344.182200	Device0	0	0x6	StandardFrame	CANFD Accelerate	Tx	32	00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF...
11	344.183400	Device0	1	0x6	StandardFrame	CANFD Accelerate	Rx	32	00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF...

그림 48. 데모용 CAN 분석기에서 송수신하는 메시지

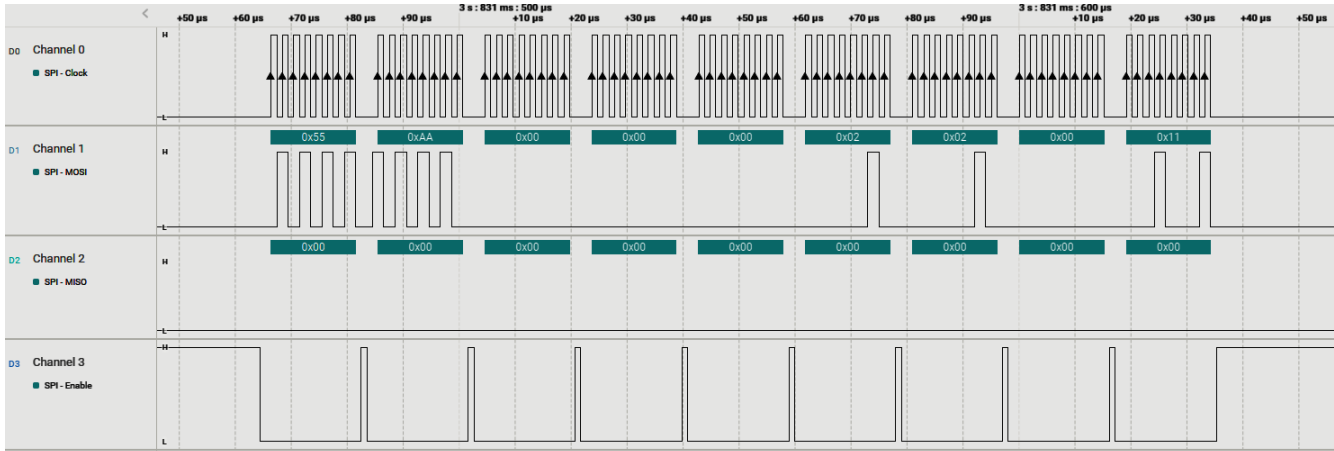


그림 49. 로직 분석기의 PC 터미널 프로그램

추가 리소스

- 텍사스 인스트루먼트, [MSPMO SDK 다운로드](#)
- 텍사스 인스트루먼트, [SysConfig에 대해 자세히 알아보기](#)
- 텍사스 인스트루먼트, [MSPMO G 시리즈 80MHz 마이크로컨트롤러](#), 기술 레퍼런스 매뉴얼
- 텍사스 인스트루먼트, [MSPMOG LaunchPad 개발 키트](#)
- 텍사스 인스트루먼트, [MSPMO CAN 아카데미](#)
- 텍사스 인스트루먼트, [MSPMO SPI 아카데미](#)

CAN-UART 브리지

설계 설명

이 서브시스템은 CAN-UART 브리지를 구축하는 방법을 보여줍니다. CAN-UART 브리지를 사용하면 장치가 한 인터페이스에서 정보를 송신 또는 수신하고 다른 인터페이스에서 정보를 수신 또는 전송할 수 있습니다. [이 예제의 코드를 다운로드하세요.](#)

그림 50에서는 이 서브시스템의 기능 다이어그램을 보여줍니다.

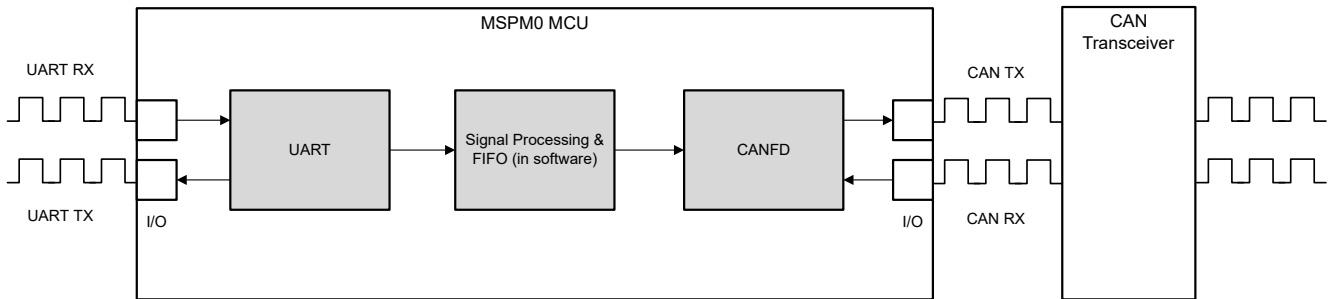


그림 50. 서브시스템 기능 블록 다이어그램

필요한 주변 기기

이 애플리케이션은 CANFD 및 UART가 필요합니다.

표 27. 필요한 주변 기기

하위 블록 기능	사용되는 주변 기기	참고
CAN 인터페이스	(1x) CANFD	코드에서 <code>MCAN0_INST</code> 라고 부름
UART 인터페이스	(1x) UART	코드에서 <code>UART_0_INST</code> 라고 부름

호환 가능 장치

표 27의 요구 사항에 따라 이 예제는 표 28의 장치와 호환됩니다. 해당 EVM은 프로토타입 제작에 사용할 수 있습니다.

표 28. 호환 가능 장치

호환 가능 장치	EVM
MSPM0G35xx,	LP-MSPM0G3507

설계 단계

1. CAN 모드, 비트 타이밍, 메시지 RAM 구성 등을 포함한 CAN 인터페이스의 기본 설정을 결정합니다. 고정된 설정과 애플리케이션에서 변경되는 설정을 고려합니다. 예제 코드에서 CANFD는 250kbit/s 중재 비율 및 2Mbit/s 데이터 전송률로 사용됩니다.
 - a. CAN-FD 주변 기기의 주요 특징은 다음과 같습니다.
 - i. ECC가 있는 전용 1KB 메시지 SRAM
 - ii. 구성 가능한 전송 FIFO, 전송 대기열 및 이벤트 FIFO(최대 32개 요소)
 - iii. 최대 32개의 전용 전송 버퍼 및 64개의 전용 수신 버퍼가 있습니다. 2개의 구성 가능한 수신 FIFO(각각 최대 64개 요소)

- iv. 최대 128개의 필터 요소
 - b. CANFD 모드가 활성화된 경우:
 - i. 64바이트 CAN-FD 프레임을 완벽하게 지원
 - ii. 최대 8Mbit/s의 비트 레이트
 - c. CANFD 모드가 비활성화된 경우:
 - i. 8바이트 클래식 CAN 프레임을 완벽하게 지원
 - ii. 최대 1Mbit/s의 비트 레이트
2. 데이터 길이, 비트 레이트 전환, 식별자, 데이터 등을 포함한 CAN 프레임을 결정합니다. 고정된 부품과 애플리케이션에서 변경해야 하는 부분을 고려합니다. 예제 코드에서 식별자, 데이터 길이 및 데이터는 서로 다른 프레임에서 변경되는 반면 다른 항목은 고정되어 있습니다. 프로토콜 통신이 필요한 경우 사용자는 코드를 수정해야 합니다.

```

/**
 * @brief Structure for MCAN Rx Buffer element.
 */
typedef struct {
    /* Identifier */
    uint32_t id;
    /* Remote Transmission Request
     * 0 = Received frame is a data frame
     * 1 = Received frame is a remote frame
     */
    uint32_t rtr;
    /* Extended Identifier
     * 0 = 11-bit standard identifier
     * 1 = 29-bit extended identifier
     */
    uint32_t xtd;
    /* Error State Indicator
     * 0 = Transmitting node is error active
     * 1 = Transmitting node is error passive
     */
    uint32_t esi;
    /* Rx Timestamp */
    uint32_t rxts;
    /* Data Length Code
     * 0-8 = CAN + CAN FD: received frame has 0-8 data bytes
     * 9-15 = CAN: received frame has 8 data bytes
     * 9-15 = CAN FD: received frame has 12/16/20/24/32/48/64 data bytes
     */
    uint32_t dlc;
    /* Bit Rate Switching
     * 0 = Frame received without bit rate switching
     * 1 = Frame received with bit rate switching
     */
    uint32_t brs;
    /* FD Format
     * 0 = Standard frame format
     * 1 = CAN FD frame format (new DLC-coding and CRC)
     */
    uint32_t fdf;
    /* Filter Index */
    uint32_t fidx;
    /* Accepted Non-matching Frame
     * 0 = Received frame matching filter index FIDX
     * 1 = Received frame did not match any Rx filter element
     */
    uint32_t anmf;
    /* Data bytes.
     * Only first dlc number of bytes are valid.
     */
    uint16_t data[DL_MCAN_MAX_PAYLOAD_BYTES];
} DL_MCAN_RxBufElement;

```

3. UART 모드, 보 레이트, 워드 길이, FIFO 등을 포함한 UART 인터페이스의 기본 설정을 결정합니다. 고정된 설정과 애플리케이션에서 변경되는 설정을 고려합니다. 예제 코드에서 UART는 9600보 레이트로 사용됩니다.
 - a. UART 주변 기기의 주요 특징은 다음과 같습니다.
 - i. 시작, 중지 및 패리티를 위한 표준 비동기 통신 비트
 - ii. 완전히 프로그래밍 가능한 직렬 인터페이스
 - iii. 별도의 전송 및 수신 FIFO가 DAM 데이터 전송 지원
 - iv. 전송 및 수신 루프백 모드 작동 지원
4. UART 프레임을 결정합니다. 일반적으로 UART는 바이트 단위로 전송됩니다. 고급 통신을 달성하기 위해 사용자는 소프트웨어를 통해 프레임 통신을 구현할 수 있습니다. 필요한 경우 사용자는 특정 통신 프로토콜을 도입할 수도 있습니다. 예제 코드에서 메시지 형식은 < 55 AA ID1 ID2 ID3 ID4 Length Data1 Data2...>입니다. 사용자는 동일한 형식으로 데이터를 입력하여 터미널에서 CAN 버스로 데이터를 전송할 수 있습니다. 55 AA는 헤더입니다. ID 영역은 4바이트입니다. 길이 영역은 데이터 길이를 나타내는 1바이트입니다. 사용자가 UART 프레임을 수정해야 하는 경우 프레임 수집 및 구문 분석에 대한 코드도 수정해야 합니다.

표 29. UART 프레임 양식

헤더	주소	데이터 길이	데이터
0x55 0xAA	4바이트	1바이트	(데이터 길이) 바이트

5. 변환해야 할 메시지, 메시지 변환 방법 등을 포함하여 브리지 구조를 결정합니다.
 - a. 브리지가 단방향인지 또는 양방향인지를 고려합니다. 일반적으로 각 인터페이스에는 수신 및 전송이라는 두 가지 기능이 있습니다. 일부 기능만 포함해야 하는지 고려합니다(예: UART 수신 및 CAN 전송). 예제 코드에서 CAN-UART 브리지는 양방향 구조입니다.
 - b. 변환할 정보 및 해당 캐리어(변수, FIFO)를 고려합니다. 예제 코드에서 식별자, 데이터 및 데이터 길이는 한 인터페이스에서 다른 인터페이스로 변환됩니다. 아래에 나와 있는 것처럼 코드에 정의된 두 개의 FIFO가 있습니다.

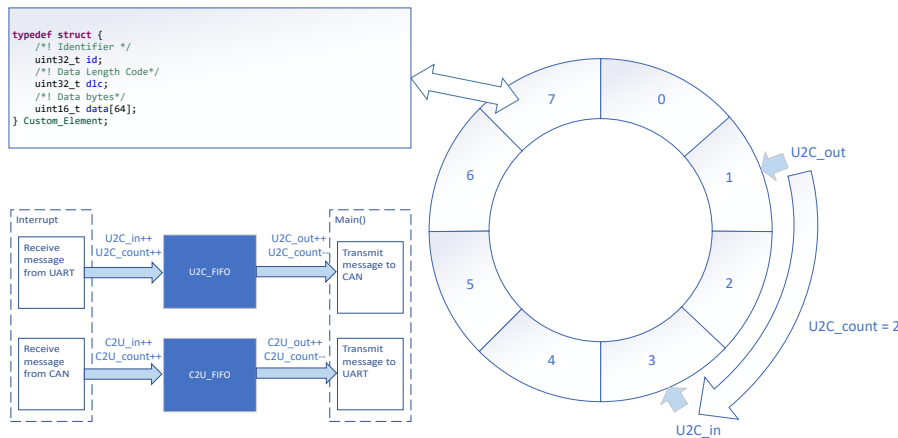


그림 51. 브리지 구조

6. (옵션) 우선 순위 설계, 정체 상황, 오류 처리 등을 고려합니다.

설계 고려 사항

1. 애플리케이션의 정보 흐름을 고려하여 각 인터페이스에서 수신 또는 전송할 정보, 따라야 할 프로토콜을 결정하고, 서로 다른 인터페이스를 연결하기 위한 적절한 정보 전송 캐리어를 설계합니다.
2. 먼저 인터페이스를 별도로 테스트한 후 전체 브리지 기능을 구현하는 것이 좋습니다. 또한 통신 실패, 과부하, 프레임 형식 오류 등과 같은 비정상적인 상황의 처리를 고려합니다.
3. 인터럽트를 통해 인터페이스 기능을 구현하여 적시에 통신을 보장하도록 하는 것이 좋습니다. 예제 코드에서 인터페이스 기능은 보통 인터럽트에 구현되고, 정보 전송은 main() 함수에서 완료됩니다.

소프트웨어 흐름도

다음 그림은 메시지가 한 인터페이스에서 수신되고 다른 인터페이스에서 전송되는 방식을 설명하는 CAN-UART 브리지에 대한 코드 흐름 다이어그램을 보여줍니다. CAN-UART 브리지는 UART에서 수신, CAN에서 수신, CAN을 통한 전송, UART를 통한 전송 등 4개의 독립적인 작업으로 나눌 수 있습니다. 두 개의 FIFO는 양방향 메시지 전송 및 메시지 캐싱을 구현합니다.

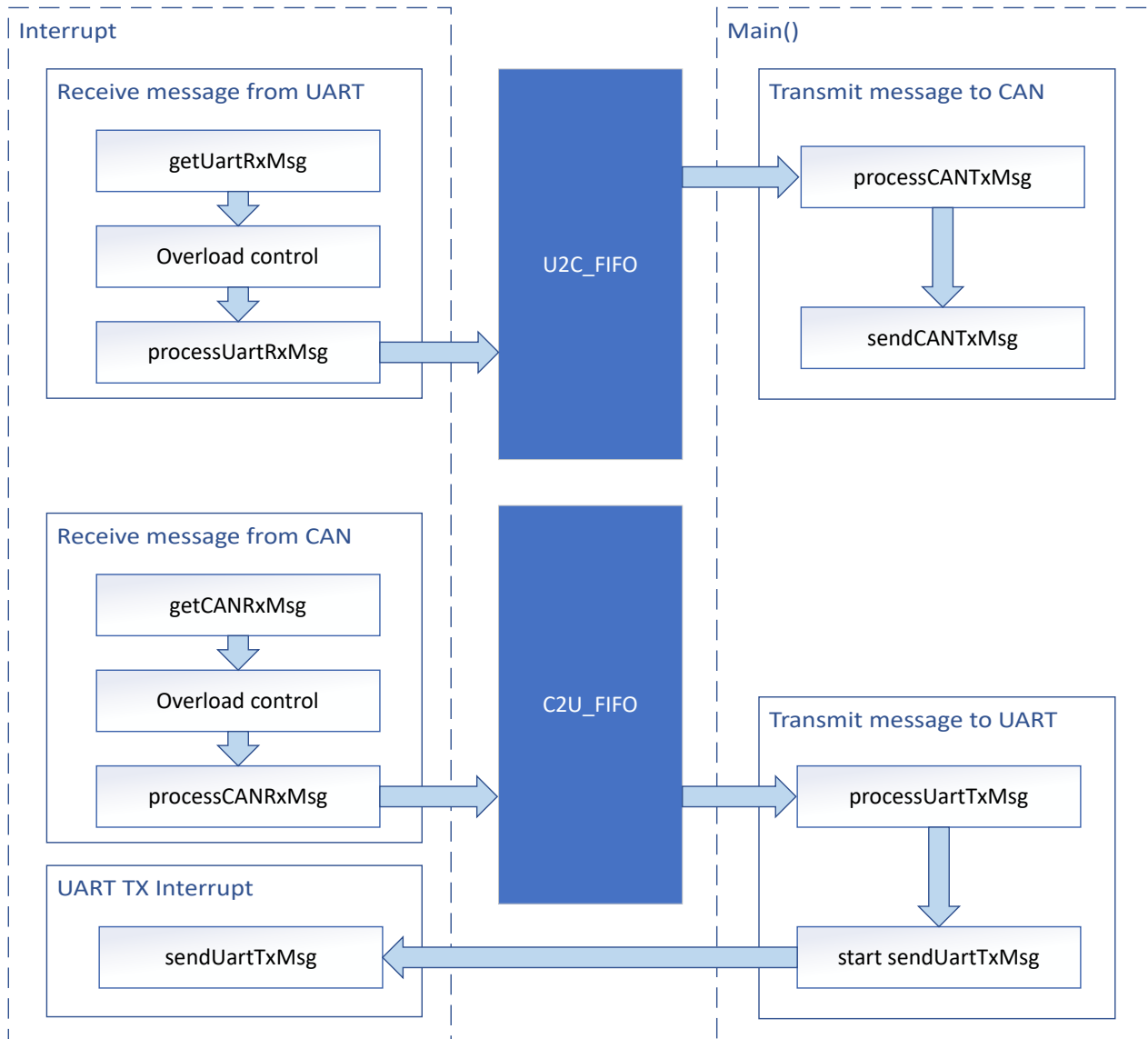


그림 52. 애플리케이션 소프트웨어 흐름도

장치 구성

이 애플리케이션은 TI 시스템 구성 툴(SysConfig) 그래픽 인터페이스를 사용하여 CAN 및 UART의 구성 코드를 생성합니다. 그래픽 인터페이스를 사용하여 장치 주변 기기를 구성하면 애플리케이션 프로토타입 제작 프로세스를 간소화할 수 있습니다.

그림 52에 설명된 코드는 그림 53에 표시된 것처럼 파일의 예제 코드에서 확인할 수 있습니다.

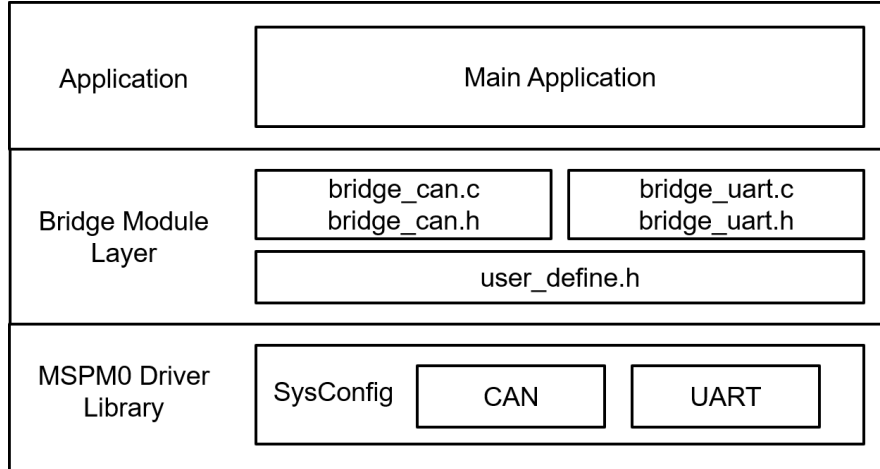


그림 53. 파일 구조

애플리케이션 코드

다음 코드 스니펫에서는 인터페이스 기능을 수정하는 위치를 보여 줍니다. 표의 함수는 서로 다른 파일로 분류됩니다. UART 수신 및 전송을 위한 함수는 bridge_uart.c 및 bridge_uart.h에 포함되어 있습니다. CAN 수신 및 전송을 위한 함수는 bridge_can.c 및 bridge_can.h에 포함되어 있습니다. FIFO 요소의 구조는 user_define.h에 정의되어 있습니다.

사용자는 파일별로 함수를 쉽게 분리할 수 있습니다. 예를 들어 UART 함수만 필요한 경우 사용자는 bridge_uart.c 및 bridge_uart.h를 예약하여 함수를 호출할 수 있습니다.

주변 기기 기본 구성은 MSPM0 SDK 및 DriverLib 문서를 참조하세요.

표 30. 함수 및 설명

작업	함수	설명	장소
UART 수신	getUartRxMsg()	수신한 UART 메시지 가져오기	bridge_uart.c bridge_uart.h
	processUartRxMsg()	수신한 UART 메시지 포맷을 변환하여 메시지를 gUART_RX_Element에 저장	
UART 전송	processUartTxMsg()	UART를 통해 전송할 gUART_TX_Element 포맷 변환	
	sendUartTxMsg()	UART를 통해 메시지 전송	
CAN 수신	getCANRxMsg()	수신한 CAN 메시지 가져오기	bridge_can.c bridge_can.h
	processCANRxMsg()	수신한 CAN 메시지 포맷을 변환하여 메시지를 gCAN_RX_Element에 저장	
CAN 전송	processCANTxMsg()	CAN을 통해 전송할 gCAN_TX_Element 포맷 변환	
	sendCANTxMsg()	CAN를 통해 메시지 전송	

Custom_Element는 user_define.h에 정의된 구조입니다. Custom_Element는 FIFO 요소의 구조, UART/CAN 전송의 출력 요소 및 UART/CAN 수신 입력 요소로 사용됩니다. 사용자는 필요에 따라 구조를 수정할 수 있습니다.

```

typedef struct {
    /*! Identifier */
    uint32_t id;
    /*! Data Length Code*/
    uint32_t dlc;
    /*! Data bytes*/
    uint16_t data[64];
} Custom_Element;

```

FIFO의 경우 FIFO로 사용되는 글로벌 변수 2개가 있습니다. 6개의 글로벌 변수가 FIFO를 추적하는 데 사용됩니다.

```

Custom_Element U2C_FIFO[U2C_FIFO_SIZE];
Custom_Element C2U_FIFO[C2U_FIFO_SIZE];
uint16_t U2C_in = 0;
uint16_t U2C_out = 0;
uint16_t U2C_count = 0;
uint16_t C2U_in = 0;
uint16_t C2U_out = 0;
uint16_t C2U_count = 0;

```

결과

런치패드에서 XDS110을 사용하여 PC를 통해 UART 측에서 메시지를 전송 및 수신할 수 있습니다. 데모로서 예를 들면 두 개의 런치패드를 두 개의 CAN-UART 브리지로 사용하여 루프를 형성할 수 있습니다. PC가 런치패드 중 하나를 통해 UART 메시지를 전송하면 XDS110은 다른 런치패드로부터 UART 메시지를 수신할 수 있습니다.

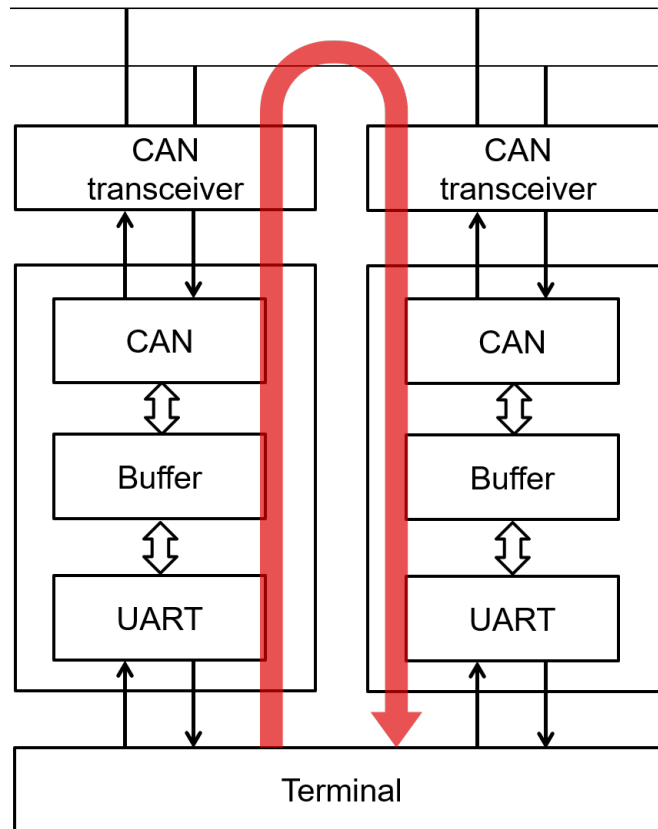


그림 54. 데모

```
55 AA 00 00 00 01 10 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  
55 AA 00 00 00 02 10 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  
55 AA 00 00 00 03 10 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F  
55 AA 00 00 00 04 10 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F  
55 AA 00 00 00 FF 08 00 11 22 33 44 55 66 77
```

그림 55. PC 터미널 프로그램

추가 리소스

- [MSPM0 SDK 다운로드](#)
- [SysConfig에 대해 자세히 알아보기](#)
- [MSPM0G TRM\(기술 레퍼런스 매뉴얼\)](#)
- [MSPM0G LaunchPad 개발 키트](#)
- [MSPM0 CAN 아카데미](#)
- [MSPM0 UART 아카데미](#)

병렬 IO-UART 브리지

설계 설명

많은 애플리케이션은 동시에 여러 GPIO의 상태 변경을 캡처하고, 업데이트한 후 UART를 통해 상태를 호스트로 전송해야 합니다. GPIO 리소스가 충분한 비용 효율적인 MCU(마이크로컨트롤러)는 병렬-직렬로 구현하고 UART를 통해 PC 측과 같이 호스트로 데이터를 실시간으로 전송할 수 있습니다. [이 예제의 코드를 다운로드하세요.](#)

그림 56에서는 이 서브시스템의 기능 다이어그램을 보여줍니다.

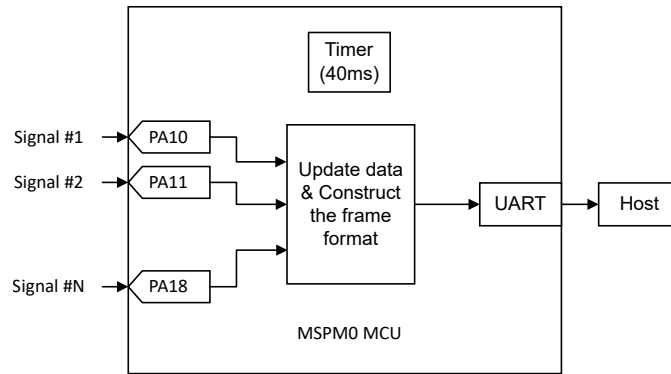


그림 56. 서브시스템 기능 블록 다이어그램

필요한 주변 기기

이 애플리케이션에는 GPIO 9개, 타이머 1개 및 UART 1개가 필요합니다.

표 31. 필요한 주변 기기

하위 블록 기능	사용되는 주변 기기	참고
IO 입력	9핀	코드에서 <code>GROUP1_IRQHandler</code> 라고 부름
타이머 간격	TIMG0	코드에서 <code>TIMG0_IRQHandler</code> 라고 부름
UART 출력	UART0	코드에서 <code>transmitPacketBlocking0</code> 이라고 부름

호환 가능 장치

표 31의 요구 사항에 따라 이 예제는 표 32의 장치와 호환됩니다. 해당 EVM은 프로토타입 제작에 사용할 수 있습니다.

표 32. 호환 가능 장치

호환 가능 장치	EVM
MSPM0L1xx	LP-MSPM0L1306
MSPM0G3xx/1xx	LP-MSPM0G3507

설계 단계

- 9개의 GPIO 스위치의 상태를 캡처합니다.
- 데이터 세그먼트에 이러한 9개의 비트를 채우고 UART를 통해 완료된 프레임 하나를 PC로 전송합니다.
- 작업이 감지될 때 또는 40ms마다 데이터를 업데이트합니다.

설계 고려 사항

이 구현은 9개의 GPIO 핀(PA10~PA18)을 사용하여 표 33에 표시된 해당 작업을 나타내는 스위치의 상태를 캡처합니다.

표 33. 핀과 작업 사이의 통신

GPIO 핀	작동
PA10	GPIO_Signal_10
PA11	GPIO_Signal_11
PA12	GPIO_Signal_12
PA13	GPIO_Signal_13
PA14	GPIO_Signal_14
PA15	GPIO_Signal_15
PA16	GPIO_Signal_16
PA17	GPIO_Signal_17
PA18	GPIO_Signal_18

상기 핀에서 PA14는 런치패드에서 S2에 고정적으로 연결되고 S2를 누르면 PA14가 접지로 풀다운됩니다. 다른 핀의 경우 각 핀을 J11을 통해 S1에 연결할 수 있으며 S1을 누르면 핀이 3V3까지 풀업될 수 있습니다. 예를 들어, S1이 PA18에 연결되어 있고 두 SW를 동시에 누르면 표 34에 표시된 것처럼 데이터가 업데이트됩니다.

표 34. 9개 핀의 데이터 형식

	Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
GPIO 핀								PA18	PA17	PA16	PA15	PA14	PA13	PA12	PA11	PA10
기본값	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
PA18 및 14	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

SW를 누르면 MCU는 데이터 세그먼트(2바이트)를 업데이트하고 즉시 체크섬을 확인한 후 UART를 통해 다음 형식으로 구성된 새 데이터를 PC로 전송할 수 있습니다.

40ms마다 SW를 누르지 않으면 MCU는 현재 상태를 PC로 전송할 수 있습니다. PC로 전송된 패키지의 형식은 표 35에 표시된 것과 같습니다.

표 35. UART에서 전송하는 패키지 형식

바이트	헤더(2바이트)	데이터 길이 (1바이트)	소스 ID(1바이트)	대상 ID(1바이트)	명령(1바이트)	데이터 색인 (1바이트)	데이터(N 바이트)	체크섬(2바이트)
값	0x5A 0x5A	N	0~63	0~63	0~255	0~255	데이터	CsumL CsumH

소프트웨어 흐름도

그림 57에서는 메인 함수인 메인 루프와 GROUP1_IRQHandler 함수인 GPIO 인터럽트 처리에 대한 코드 흐름 다이어그램을 보여줍니다.

TIMG0 인터럽트 처리는 타이머 인터럽트로 전환되어 40ms마다 전류 데이터를 전송하는 매우 간단한 방식입니다.

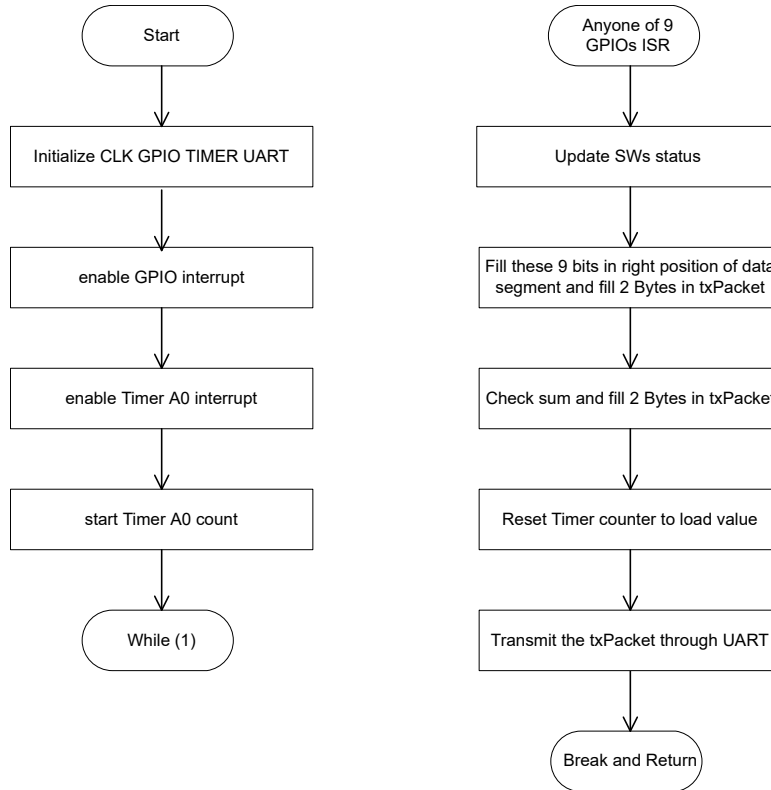


그림 57. 애플리케이션 소프트웨어 흐름도

애플리케이션 코드

메인 루프

```

SYSCFG_DL_init();
NVIC_EnableIRQ(GPIO_MULTIPLE_GPIOA_INT_IRQN);
NVIC_EnableIRQ(TIMER_0_INST_INT_IRQN);
DL_TimerG_startCounter(TIMER_0_INST);
while (1) {
    __WFI();
}
    
```

TIMG0_IRQHandler

```

switch (DL_TimerG_getPendingInterrupt(TIMER_0_INST)) {
    case DL_TIMER_IIDX_ZERO:
        transmitPacketBlocking(gTxPacket, UART_PACKET_SIZE);
        break;
}
    
```

GPIO GROUP1_IRQHandler

```

if (DL_Interrupt_getPendingGroup(DL_INTERRUPT_GROUP_1)) {
    dataStatus = (GPIOA->DIN31_0);
    dataTemp = (dataStatus >> 10);
}
    
```

```

gTxPacket[7] = dataTemp >> 8;
gTxPacket[8] = dataTemp & 0xFF;

siganlChecksum = checkSum1ByteIn2ByteOut((gTxPacket+2),7);

gTxPacket[10] = siganlChecksum >> 8;
gTxPacket[9] = siganlChecksum & 0xFF;

DL_TimerG_stopCounter(TIMER_0_INST);
DL_TimerG_setTimerCount(TIMER_0_INST, TIMER_0_INST_LOAD_VALUE);
DL_TimerG_startCounter(TIMER_0_INST);

transmitPacketBlocking(gTxPacket, UART_PACKET_SIZE);
}

```

결과

로직 분석을 사용하여 데이터 흐름을 캡처하고 더 자세한 정보를 표시합니다.

채널 0 ----> UART Tx

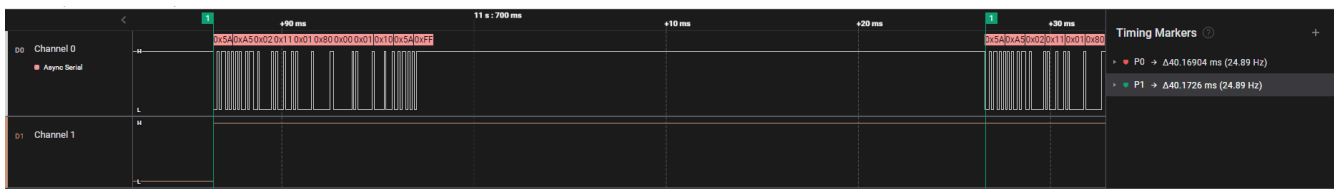
채널 1 ----> PA18

아래 이미지는 다음 사항을 보여줍니다.

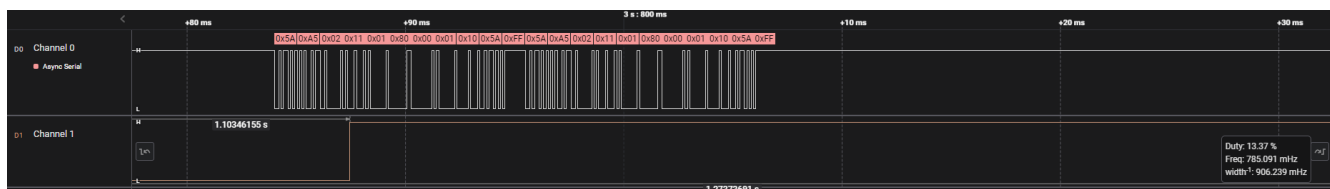
SW를 누르지 않으면 MCU는 기본값을 40ms마다 전송합니다.



S1을 누르면 PA18에서 상승 에지와 데이터 업데이트가 발생합니다. 그런 다음, MCU는 40ms마다 업데이트 데이터를 전송합니다.



상승 에지가 발생하지만 마지막 패키지가 완료되지 않은 경우 MCU는 마지막 전송이 완료된 후 데이터 업데이트를 전송합니다.



추가 리소스

- [MSPM0 SDK 다운로드](#)
- [SysConfig에 대해 자세히 알아보기](#)
- [MSPM0G TRM\(기술 레퍼런스 매뉴얼\)](#)

- [MSPMOL TRM\(기술 레퍼런스 매뉴얼\)](#)
- [MSPMOG LaunchPad 개발 키트](#)
- [MSPMOL LaunchPad 개발 키트](#)
- [MSPMO 타이더 아카데미](#)
- [MSPMO UART 아카데미](#)

UART 브리지를 통한 I2C 확장기

설명

그림 58에서는 MSPM0을 I2C 확장기로 사용하여 UART(범용 비동기 리시버-트랜스미터) 인터페이스에서 여러 대상 I2C 컨트롤러로 데이터 또는 명령을 전송하는 방법을 보여줍니다. 수신 UART 패킷은 I2C 통신으로 원활하게 전환할 수 있도록 특별한 형식으로 구성됩니다. 그림 58에서는 오류를 호스트 장치로 다시 전달하는 방법에 대해서도 설명합니다. 이 예제의 코드는 MSPM0 SDK에서 확인할 수 있습니다.

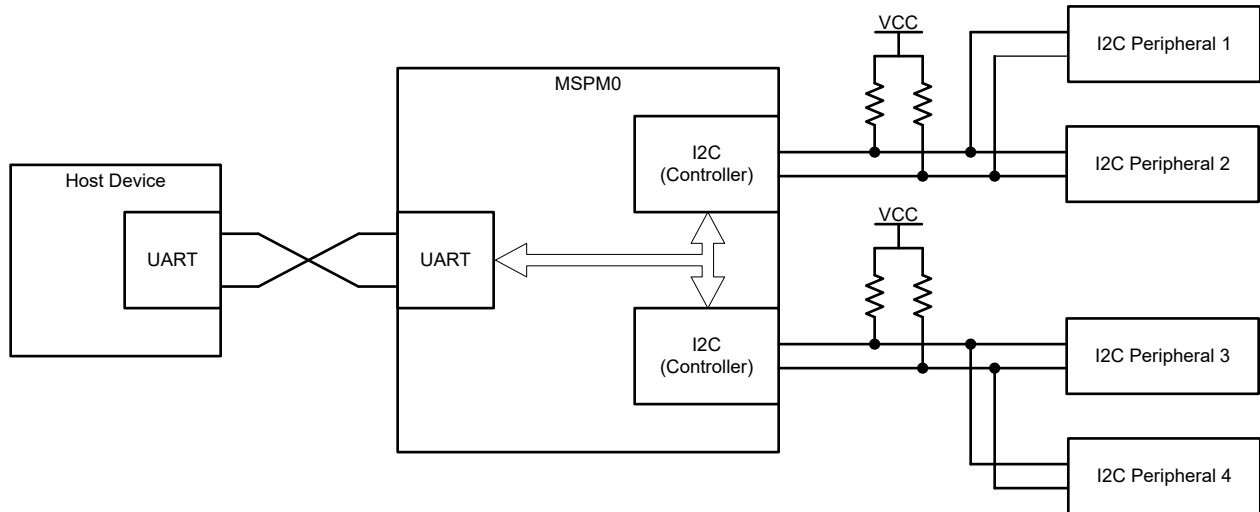


그림 58. 서브시스템 기능 블록 다이어그램

필요한 주변 기기

이 애플리케이션에는 UART 1개 및 I2C 주변 기기가 필요합니다.

표 36. 필요한 주변 기기

하위 블록 기능	사용되는 주변 기기	참고
UART TX-RX 인터페이스	(1x) UART	코드에서 UART_BRIDGE_INST라고 부름
I2C 컨트롤러	(2x) I2C	코드에서 I2C_BRIDGE_INST 및 I2C_BRIDGE2_INST라고 부름

호환 가능 장치

표 37에는 표 36의 요구 사항에 따라 해당 EVM과 호환되는 장치가 열거되어 있습니다. 표 36의 요구 사항이 충족되면 다른 MSPM0 장치 및 해당하는 EVM을 사용할 수 있습니다.

표 37. 호환 가능 장치

호환 가능 장치	EVM
MSPM0Lxxxx	LP-MSPM0L1306
MSPM0Gxxxx	LP-MSPM0G3507

설계 단계

1. SysConfig에서 UART 주변 기기 인스턴스, I2C 주변 기기 인스턴스 및 원하는 장치 핀에 대한 핀 출력을 설정합니다.
2. SysConfig에서 UART 보 레이트를 설정합니다. 기본값은 9600보입니다.
3. SysConfig에서 I2C 클럭 속도를 설정합니다. 기본값은 100kHz입니다.
4. 브릿지가 처리하는 최대 I2C 패킷 크기를 정의합니다.
5. 주요 UART 헤더 값을 정의합니다(옵션).
6. 오류 처리를 사용자 지정합니다(옵션).

설계 고려 사항

- **통신 속도:** 속도를 높이면 데이터 처리량이 증가하고 충돌 가능성이 줄어듭니다. I2C 속도가 빨라진 경우 통신을 위해서는 I2C 사양에 따라 외부 풀업 저항을 조정해야 합니다. 최적화에는 더 높은 장치 작동 속도, 다중 전송 버퍼, 헤더 크기 감소 또는 상태 시스템 간소화가 포함됩니다.
- **UART 헤더:** UART 패킷 헤더 및 시작 바이트는 애플리케이션에 대해 사용자 지정할 수 있습니다. 텍사스 인스트루먼트에서는 일반적인 데이터 전송 시작 시 발생할 가능성이 더 적은 값을 할당할 것을 권장합니다.
- **오류 처리:** UART 버스를 컴퓨터 터미널로 모니터링하는 경우 ASCII 수치 값에 오류 값을 대응시킵니다. 호스트가 적절한 작업을 수행할 수 있도록 호스트 UART 장치가 오류 값을 읽을 수 있고 관련 의미를 알고 있는지 확인합니다. ErrorFlags 구조 유형을 수정하여 추가 오류 유형을 추가하고 Uart_Bridge() 내에 추가 오류 감지 코드를 추가합니다. 현재 구현에서는 제한된 오류를 감지하고 UART 인터페이스에 해당 코드를 다시 보고합니다. 그런 다음, 애플리케이션 코드가 현재 통신 상태 시스템에서 중단됩니다. 사용자는 오류 발생 시 브릿지의 동작을 변경하기 위해 오류 처리 코드를 추가할 수 있습니다. 예를 들어 NACK 발생 후 I2C 패킷을 재전송합니다.

소프트웨어 흐름도

그림 59, 그림 60 및 그림 61에서는 그림 58에 대한 메인 UART 브리지 기능, Main() 및 UART ISR, I2C ISR의 코드 흐름 다이어그램을 각각 보여줍니다.

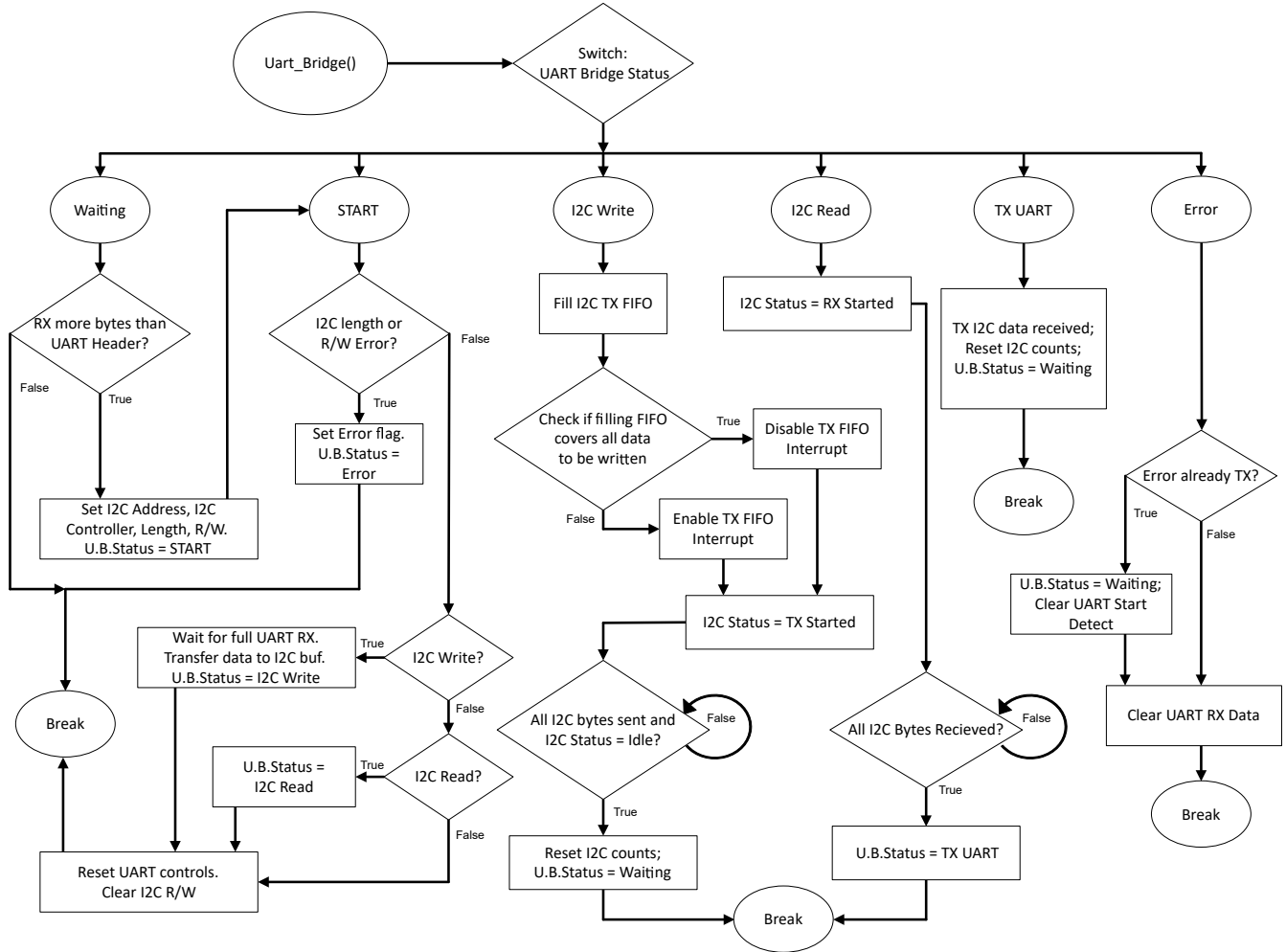


그림 59. 다음에 대한 소프트웨어 흐름 다이어그램 Uart_Bridge()

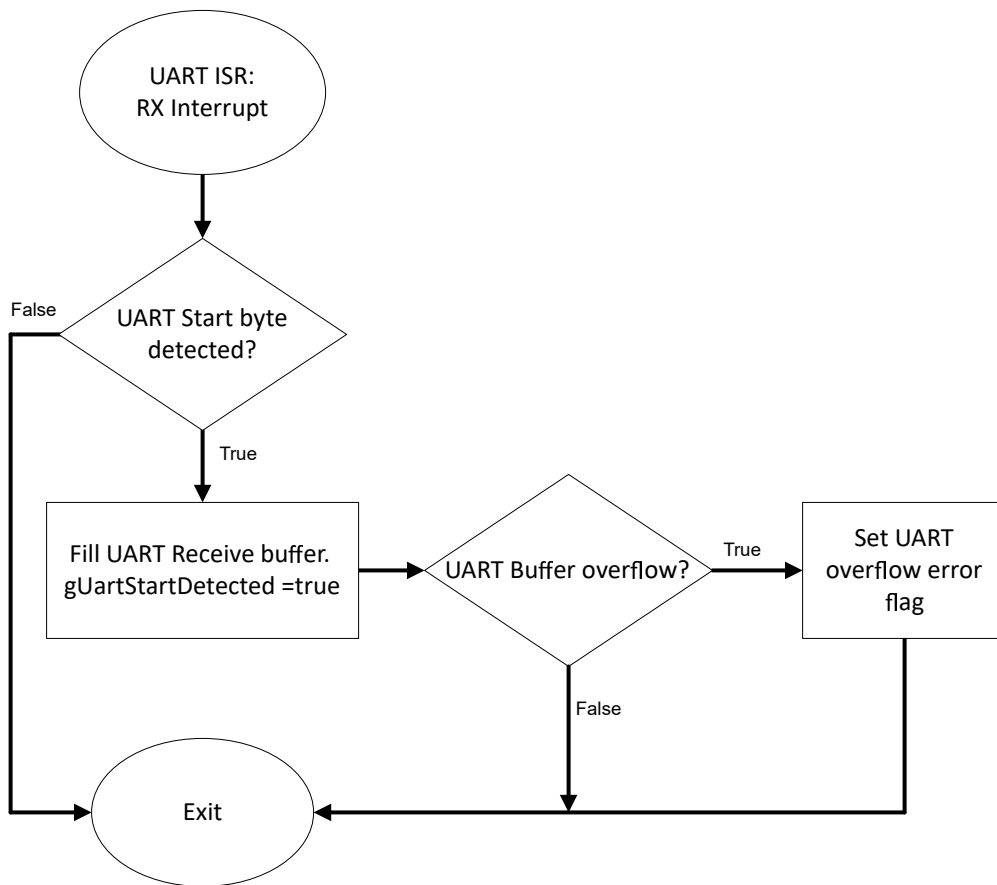
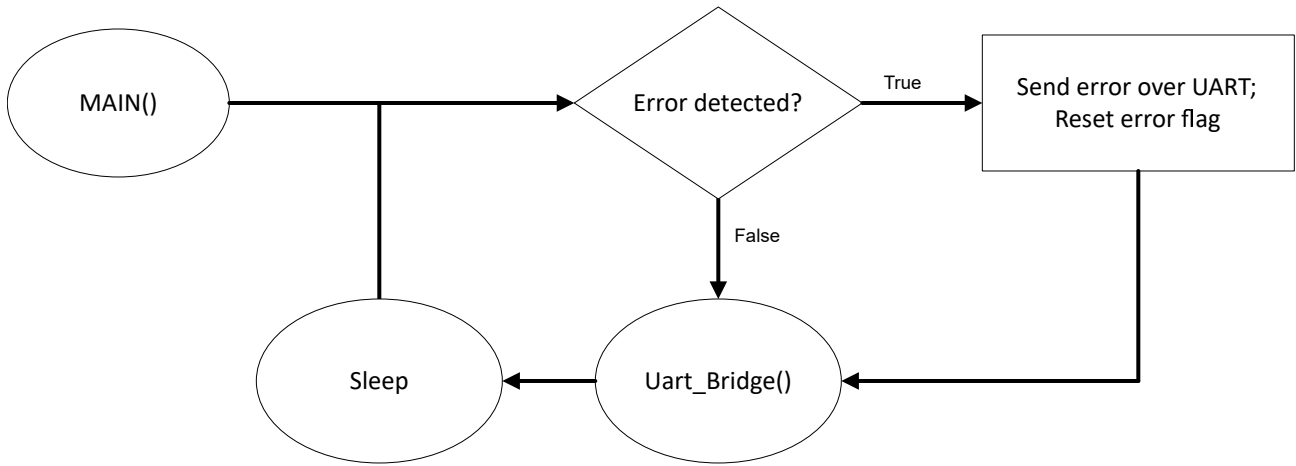


그림 60. 메인 루프 및 UART ISR의 소프트웨어 흐름 다이어그램

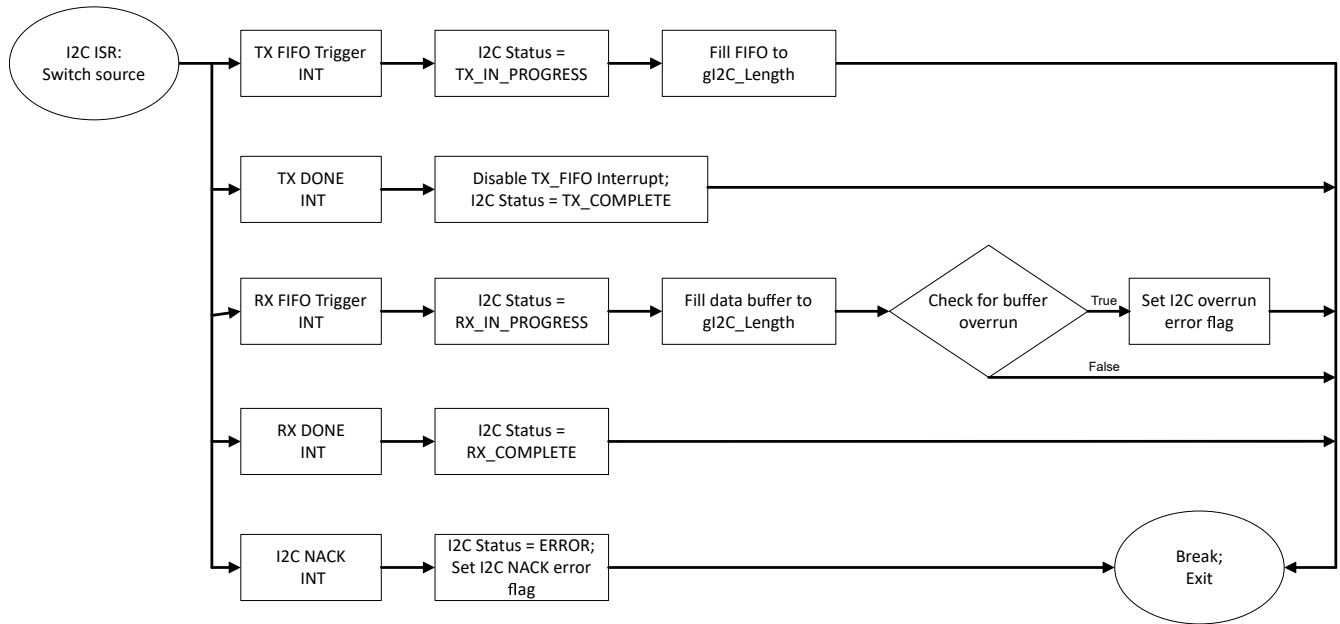


그림 61. I2C ISR의 소프트웨어 흐름 다이어그램

필수 UART 패킷

그림 62에서는 I2C 인터페이스를 올바르게 브리징하는 데 필요한 UART 패킷을 보여줍니다. 표시된 값은 그림 58 내에 정의된 기본 헤더 값입니다.

- 시작 바이트: 브리지에서 새 트랜잭션이 시작되었음을 나타내기 위해 사용하는 값입니다. 브리지에서 이 값을 승인할 때까지 UART 전송은 무시됩니다.
- I2C 주소: 호스트가 통신하는 I2C 대상의 주소입니다.
- I2C 읽기 또는 쓰기 표시기: 대상 I2C 장치에 대해 읽거나 쓰기 위해 브리지를 작동하는 값입니다.
- 메시지 길이 N: 전송되는 데이터의 길이(바이트)입니다. 이 값은 정의된 I2C 최대 패킷 길이보다 클 수 없습니다.
- 브리지 색인: 호스트가 통신하는 I2C 컨트롤러입니다.
- D0, D1..., Dn: 브리지 내에서 전송되는 데이터입니다.

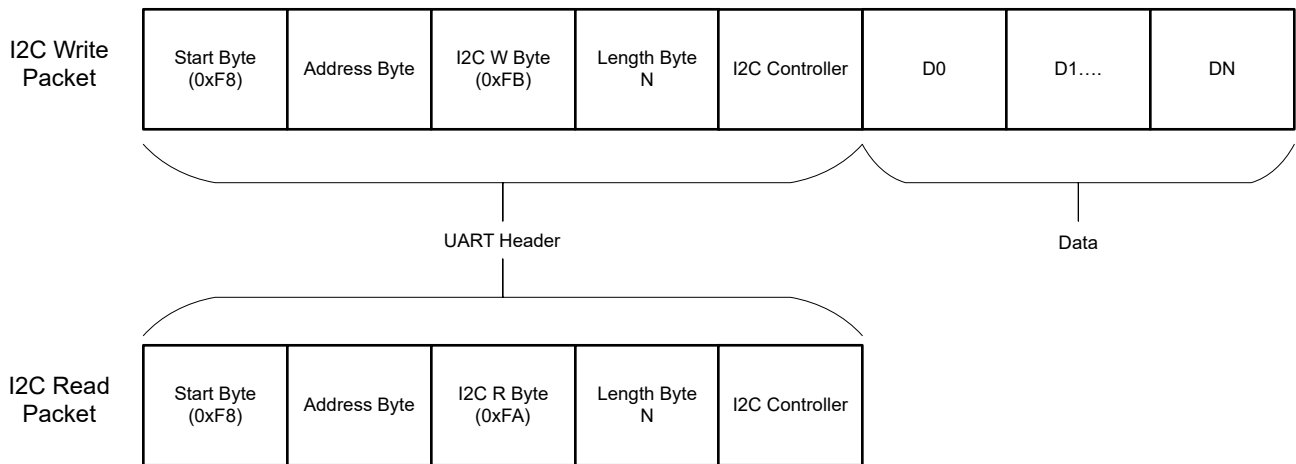


그림 62. UART 브리지 패킷 설명

장치 구성

이 애플리케이션은 TI **시스템 구성 툴**(SysConfig)의 그래픽 인터페이스를 사용하여 비교기 및 타이머 모듈 2개의 구성 코드를 생성합니다. 그래픽 인터페이스를 사용하여 장치 주변 기기를 구성하면 애플리케이션 프로토타입 제작 프로세스를 간소화할 수 있습니다.

애플리케이션 코드

UART 패킷 또는 최대 I2C 패킷 크기에 사용되는 특정 값을 변경하려면 다음 코드 블록에 설명된 대로 코드 예제의 시작 부분에서 다음 #defines를 수정합니다.

```
/* Define UART Header and Start Byte*/
#define UART_HEADER_LENGTH 0x04
#define UART_START_BYTE 0xF8
#define UART_READ_I2C_BYTE 0xFA
#define UART_WRITE_I2C_BYTE 0xFB
#define ADDRESS_INDEX 0x00
#define RW_INDEX 0x01
#define LENGTH_INDEX 0x02
#define BRIDGE_INDEX 0x03

/*Define max packet sizes*/
#define I2C_MAX_PACKET_SIZE 16
#define UART_MAX_PACKET_SIZE (I2C_MAX_PACKET_SIZE + UART_HEADER_LENGTH)
```

추가 리소스

- 텍사스 인스트루먼트, [I2C 확장기 서브시스템 코드](#)
- 텍사스 인스트루먼트, [MSPM0 SDK 다운로드](#)
- 텍사스 인스트루먼트, [SysConfig에 대해 자세히 알아보기](#)
- 텍사스 인스트루먼트, [MSPM0L LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0G LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0 UART 아카데미](#)
- 텍사스 인스트루먼트, [MSPM0 I2C 아카데미](#)

E2E

TI의 **E2E™** 지원 포럼 토론을 보고 새로운 스레드를 게시하여 설계에서 MSPM0 장치를 활용하는 데 필요한 기술 지원을 받으세요.

UART-I2C 브리지

설명

그림 63에서는 MSPM0을 I2C 컨트롤러로 사용하여 UART 인터페이스의 데이터 또는 명령을 여러 대상 I2C 주변 기기로 전송하는 방법을 보여줍니다. 수신 UART 패킷은 I2C 통신으로 원활하게 전환할 수 있도록 특별한 형식으로 구성됩니다. 그림 63은(는) 통신에서의 오류를 호스트 장치에 다시 전달할 수 있습니다. 이 예제의 코드는 **UART-I2C 브리지 서브시스템 코드**에서 확인할 수 있습니다.

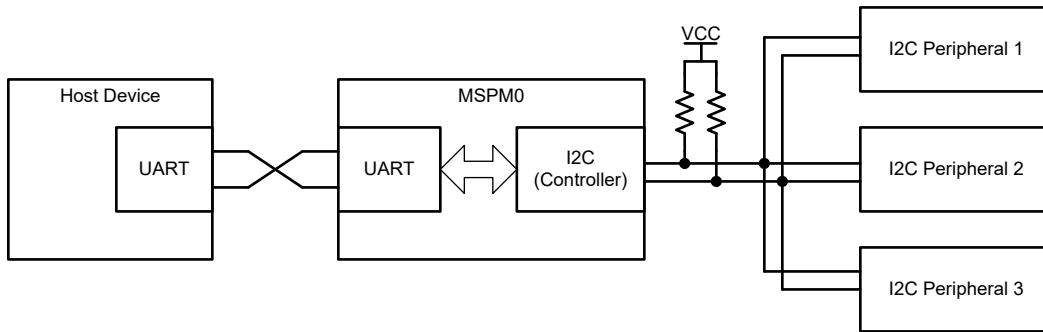


그림 63. 서브시스템 기능 블록 다이어그램

요구 사항

이 애플리케이션을 적용하려면 UART 및 I2C 주변 기기가 필요합니다.

표 38. 필요한 주변 기기

하위 블록 기능	사용되는 주변 기기	참고
UART TX/RX 인터페이스	UART	코드에서 UART_Bridge_INST라고 부름. 기본값은 9600보 레이트입니다.
I2C 컨트롤러	I2C	코드에서 I2C_Bridge_INST라고 부름. 기본적으로 전송 속도는 100kHz입니다.

호환 가능 장치

표 38의 요구 사항을 바탕으로 호환 가능 장치가 표 39에 해당하는 EVM과 함께 열거되어 있습니다. 표 38의 요구 사항이 충족되면 다른 MSPM0 장치 및 해당하는 EVM을 사용할 수 있습니다.

표 39. 호환 가능 장치

호환 가능 장치	EVM
MSPM0Lxxxx	LP-MSPM0L1306
MSPM0Gxxxx	LP-MSPM0G3507

설계 단계

1. **SysConfig**에서 UART 주변 기기 인스턴스, I2C 주변 기기 인스턴스 및 원하는 장치 핀에 대한 핀 출력을 설정합니다.
2. **SysConfig**에서 UART 보 레이트를 설정합니다. 기본값은 9600보입니다.
3. **SysConfig**에서 I2C 클럭 속도를 설정합니다. 기본값은 100kHz입니다.
4. 브리지가 처리하는 최대 I2C 패킷 크기를 정의합니다.
5. 주요 UART 헤더 값을 정의합니다(옵션).
6. 오류 처리를 사용자 지정합니다(옵션).

설계 고려 사항

1. 통신 속도.
 - a. 두 인터페이스 속도를 높이면 데이터 처리량이 증가하고 데이터 충돌 가능성이 줄어듭니다.
 - b. I2C 속도가 빨라진 경우 통신을 위해서는 I2C 사양에 따라 외부 풀업 저항을 조정해야 합니다.
 - c. 더 빠른 속도로 반복되는 대용량 데이터 패킷은 전체 시스템 성능에 영향을 줄 수 있습니다. 증가된 브리지 사용에 부합하려면 이 코드를 추가로 최적화해야 할 수 있습니다. 추가 최적화에는 더 높은 장치 작동 속도, 다중 전송 버퍼, 헤더 크기 감소 또는 상태 시스템 간소화가 포함됩니다.

주

그림 63 예제는 기본 속도 9600보(UART) 및 100kHz(I2C) 속도에서만 테스트되었습니다.

2. UART 헤더.
 - a. UART 패킷 헤더 및 시작 바이트 값은 애플리케이션에 대해 사용자 지정할 수 있습니다. 텍사스 인스트루먼트에서는 일반적인 데이터 전송 시작 시 발생할 가능성이 더 적은 값을 할당할 것을 권장합니다.
3. 오류 처리.
 - a. UART 버스를 컴퓨터 터미널로 모니터링하는 경우 ASCII 수치 값에 오류 값을 대응시킵니다.
 - b. 호스트가 적절한 작업을 수행할 수 있도록 호스트 UART 장치가 오류 값을 읽을 수 있고 관련 의미를 알고 있는지 확인합니다.
 - c. *ErrorFlags* 구조 유형을 수정하여 추가 오류 유형을 추가하고 *Uart_Bridge()* 내에 추가 오류 감지 코드를 추가합니다.
 - d. 현재 구현에서는 제한된 오류를 감지하고 UART 인터페이스에 해당 코드를 다시 보고합니다. 그런 다음, 애플리케이션 코드가 현재 통신 상태 시스템에서 중단됩니다. 사용자는 오류 발생 시 브리지의 동작을 변경하기 위해 오류 처리 코드를 추가할 수 있습니다. 예를 들어 NACK 발생 후 I2C 패킷을 재전송합니다.

주

그림 63은(는) 현재 일반적인 오류에 플래그를 지정하고 *ErrorFlags* 구조 형식에 정의된 대로 숫자 값을 할당합니다.

소프트웨어 흐름도

그림 64, 그림 65 및 그림 66에서는 그림 63에 대한 메인 UART 브리지 기능, Main() 및 UART ISR, I2C ISR의 코드 흐름 다이어그램을 각각 보여줍니다.

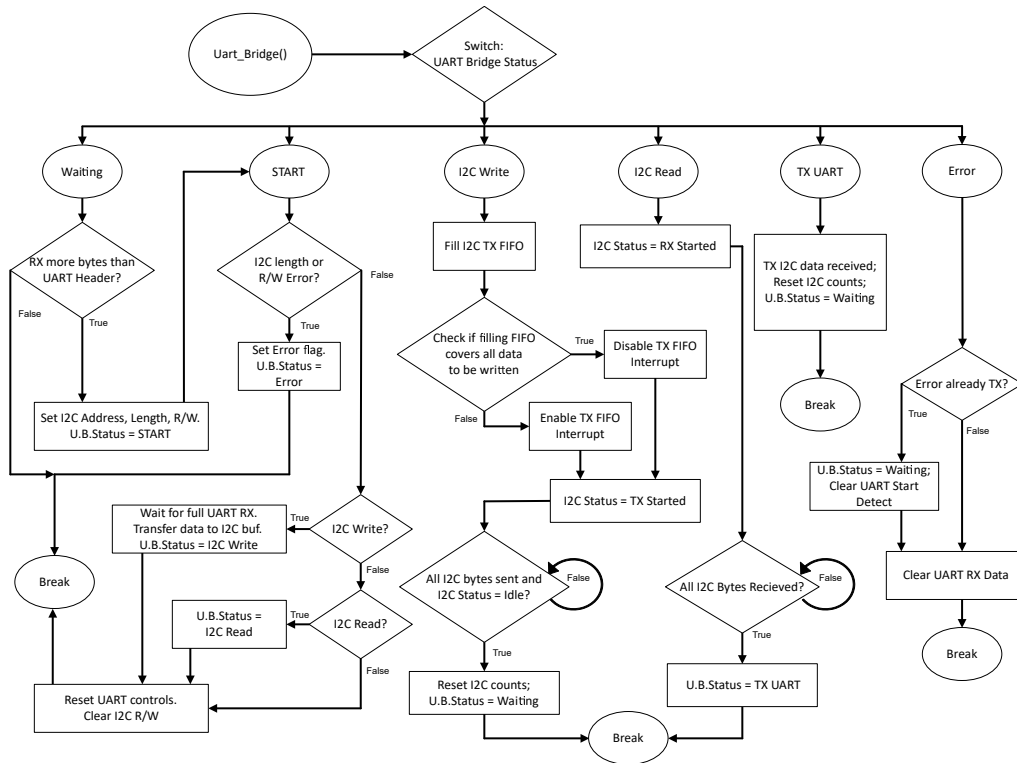


그림 64. Uart_Bridge()에 대한 소프트웨어 흐름 다이어그램

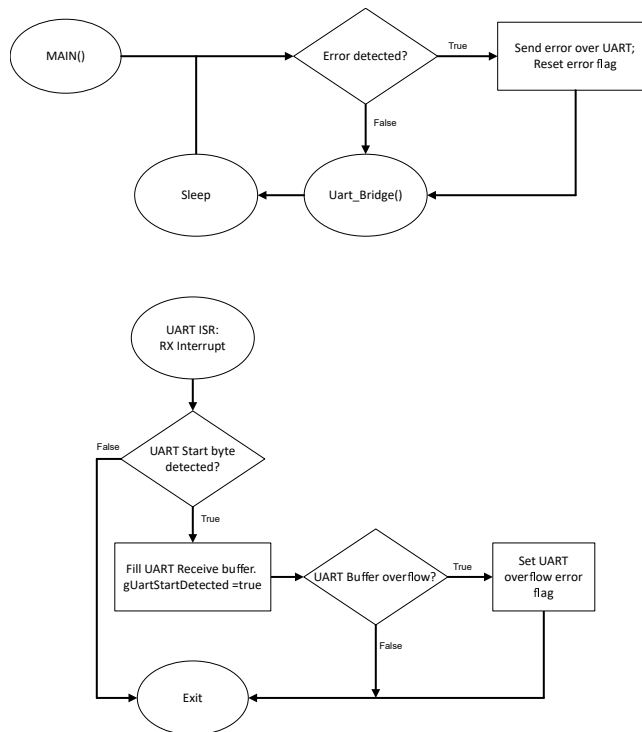


그림 65. 메인 루프 및 UART ISR의 소프트웨어 흐름 다이어그램

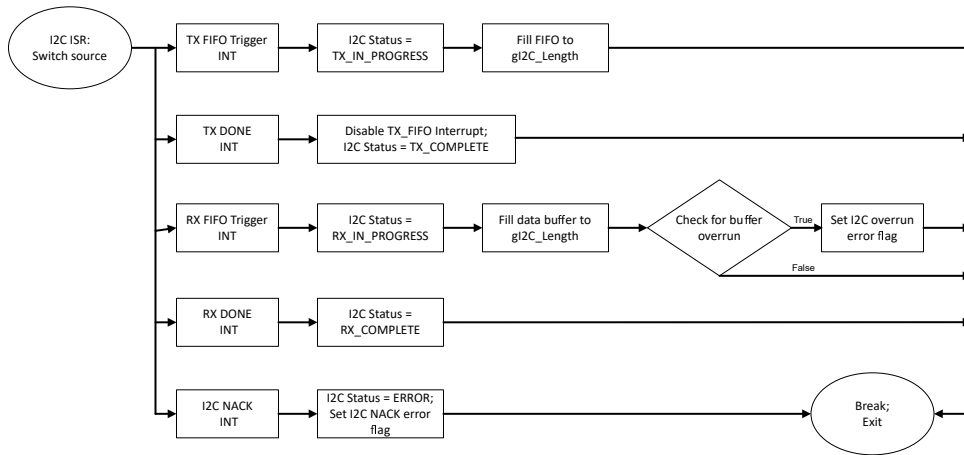


그림 66. I2C ISR의 소프트웨어 흐름 다이어그램

필요한 UART 패킷

그림 67에서는 I2C 인터페이스를 올바르게 브리징하는 데 필요한 UART 패킷을 보여줍니다. 표시된 값은 그림 63 내에 정의된 기본 헤더 값입니다.

- 시작 바이트: 브리지에서 새 트랜잭션이 시작되었음을 나타내기 위해 사용하는 값입니다. 브리지에서 이 값을 승인할 때까지 UART 전송은 무시됩니다.
- I2C 주소: 호스트가 통신하는 I2C 대상의 주소입니다.
- I2C 읽기 또는 쓰기 표시기: 대상 I2C 장치에 대해 읽거나 쓰기 위해 브리지를 작동하는 값입니다.
- 메시지 길이 N: 전송되는 데이터의 길이(바이트)입니다. 이 값은 정의된 I2C 최대 패킷 길이보다 클 수 없습니다.
- D0, D1..., Dn: 브리지 내에서 전송되는 데이터입니다.

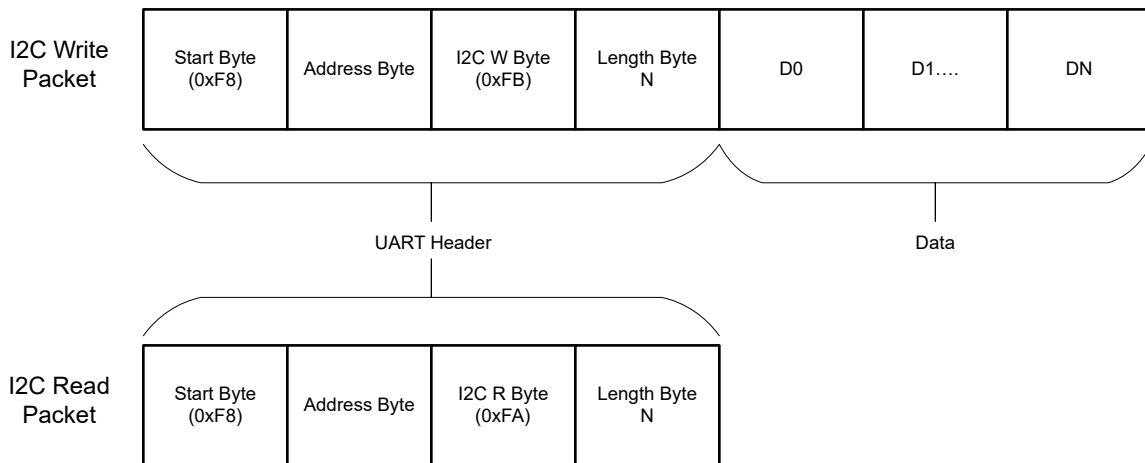


그림 67. UART 브리지 패킷 설명

장치 구성

그림 63 애플리케이션은 **TI 시스템 구성 툴(SysConfig)** 그래픽 인터페이스를 사용하여 장치 주변 기기의 구성 코드를 생성합니다. 그래픽 인터페이스를 사용하여 장치 주변 기기를 구성하면 애플리케이션 프로토타입 제작 프로세스를 간소화할 수 있습니다.

애플리케이션 코드

UART 패킷 또는 최대 I2C 패킷 크기에 사용되는 특정 값을 변경하려면 다음 코드 블록에 설명된 대로 문서의 시작 부분에서 #defines를 수정합니다.

```
/* Define UART Header and Start Byte*/
#define UART_HEADER_LENGTH 0x03
#define UART_START_BYTE 0xF8
#define UART_READ_I2C_BYTE 0xFA
#define UART_WRITE_I2C_BYTE 0xFB
#define ADDRESS_INDEX 0x00
#define RW_INDEX 0x01
#define LENGTH_INDEX 0x02

/*Define max packet sizes*/
#define I2C_MAX_PACKET_SIZE 16
#define UART_MAX_PACKET_SIZE (I2C_MAX_PACKET_SIZE + UART_HEADER_LENGTH)
```

코드의 여러 포인트는 오류 감지와 관련된 주석입니다. 사용자는 코드의 이러한 포인트에서 사용자 지정 오류 처리 및 추가 오류 보고를 추가할 수 있습니다. 간단하게 모든 오류 처리 코드 교차점이 여기에 포함되지는 않습니다. 실제에서는 다음 코드 블록에 설명된 것과 유사한 코드에서 주석을 검색합니다.

```
while (DL_I2C_isControllerRXFIFOEmpty(I2C_BRIDGE_INST) != true) {
    if (gI2C_Count < gI2C_Length) {
        gI2C_Data[gI2C_Count++] =
            DL_I2C_receiveControllerData(I2C_BRIDGE_INST);
    } else {
        /*
         * Ignore and remove from FIFO if the buffer is full
         * Optionally add error flag update
         */
        DL_I2C_receiveControllerData(I2C_BRIDGE_INST);
        gError = ERROR_I2C_OVERUN;
    }
}
```

추가 리소스

- 텍사스 인스트루먼트, [UART-I2C 브리지 서브시스템 코드](#)
- 텍사스 인스트루먼트, [TI SysConfig에 대해 자세히 알아보기](#), 툴
- 텍사스 인스트루먼트, [MSPM0 지원 개발 키트](#), 툴
- 텍사스 인스트루먼트, [MSPM0 아카데미: UART](#), 교육
- 텍사스 인스트루먼트, [MSPM0 아카데미: I2C](#), 교육

UART-SPI 브리지

설명

이 서브시스템은 UART(범용 비동기 리시버-트랜스미터)-SPI(직렬 주변 기기 인터페이스) 브리지로 MSPM0 장치를 구현하는 방법을 보여줍니다. 수신되는 UART 패킷은 SPI 통신을 원활하게 하기 위해 특정 형식을 따라야 합니다. 또한 이 예제에서는 오류 상태를 확인하고 UART 장치에 다시 전달하는 기능도 있습니다. 이 예제의 코드는 **MSPM0 SDK**에서 확인할 수 있습니다.

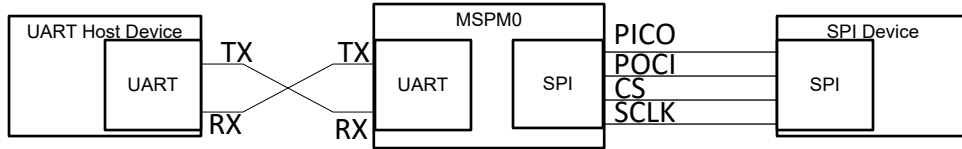


그림 68. 시스템 기능 블록 다이어그램

필요한 주변 기기

표 40. 필요한 주변 기기

사용되는 주변 기기	참고
UART	코드에서 UART_BRIDGE_INST라고 부름
SPI	코드에서 SPI_0_INST라고 부름

호환 가능 장치

표 40의 요구 사항에 따라 이 예제는 표 41에 표시된 장치와 호환됩니다. 일반적으로 필요한 주변 기기 표에 열거된 기능이 있는 모든 장치에서 이 예제를 지원할 수 있습니다.

표 41. 호환 가능 장치

호환 가능 장치	EVM
MSPM0Lxxxx	LP-MSPM0L1306
MSPM0Gxxxx	LP-MSPM0G3507

설계 단계

1. SysConfig에서 SPI 모듈을 설정합니다. 장치를 컨트롤러 모드로 설정하고 나머지 설정은 기본값으로 둡니다. 고급 구성 탭에서 RX FIFO 임계값 레벨이 *RX FIFO*에 ≥ 1 개 항목 포함으로 설정되어 있는지 확인합니다. TX FIFO 임계값 레벨이 *TX FIFO*에 ≤ 2 개 항목 포함으로 설정되어 있는지 확인합니다. 이제 *인터럽트* 구성 탭으로 이동하고 수신, 전송, RX 시간 초과, 패리티 오류, 수신 FIFO 오버플로, 수신 FIFO 가득 참, 전송 FIFO 언더플로 인터럽트를 활성화합니다.
2. SysConfig에서 UART 모듈을 설정합니다. 9600보 레이트로 설정합니다. 수신/인터럽트를 활성화합니다.

설계 고려 사항

1. 애플리케이션 코드에서 SPI 및 UART 최대 패킷 크기가 애플리케이션의 요구 사항과 일치하는지 확인합니다.
2. UART 보 레이트를 높이려면 *대상 보 레이트*라고 표시된 SysConfig UART 탭에서 값을 조정합니다. 이 아래에서 대상 보 레이트를 반영하기 위해 계산된 보 레이트 변화를 관찰합니다. 이는 사용 가능한 클럭 및 분할기를 사용하여 계산됩니다.
3. 오류 플래그를 확인하고 적절하게 처리합니다. UART 및 I²C 주변 기기는 모두 진단에 도움이 되는 오류 인터럽트를 발생시킬 수 있습니다. 손쉬운 디버깅을 위해 이 서브시스템은 열거형 및 전역 변수를 사용하여 오류 코드가 발생할 때 오류 코드를 저장합니다. 실제 애플리케이션에서는 오류로 인해 프로젝트가 중단되지 않도록 코드의 오류를 처리합니다.
4. 프로젝트의 현재 형식은 *UART_START_BYTE*, *UART_READ_SPI_BYTE* 및 *UART_WRITE_SPI_BYTE*와 같이 패킷의 형식이 지정된 모든 부분을 정의합니다. 여기에는 패킷 헤더에서 이러한 명령이 발견되는 위치를 지정하기 위한 정의가 함께 제공됩니다. 구현에서 값을 변경할 수 있습니다. UART 시작 및 읽기 또는 쓰기 바이트가 애플리케이션에서 예상되지 않은 바이트인지 확인합니다.

소프트웨어 흐름도

그림 69에서는 이 예제에 대한 코드 흐름 다이어그램을 보여주고, 다양한 UART 브리지 대기 상태 및 장치가 각 상태에서 수행하는 동작에 대해 설명합니다. 흐름도에는 UART 및 SPI에 대한 인터럽트 서비스 루틴도 표시됩니다.

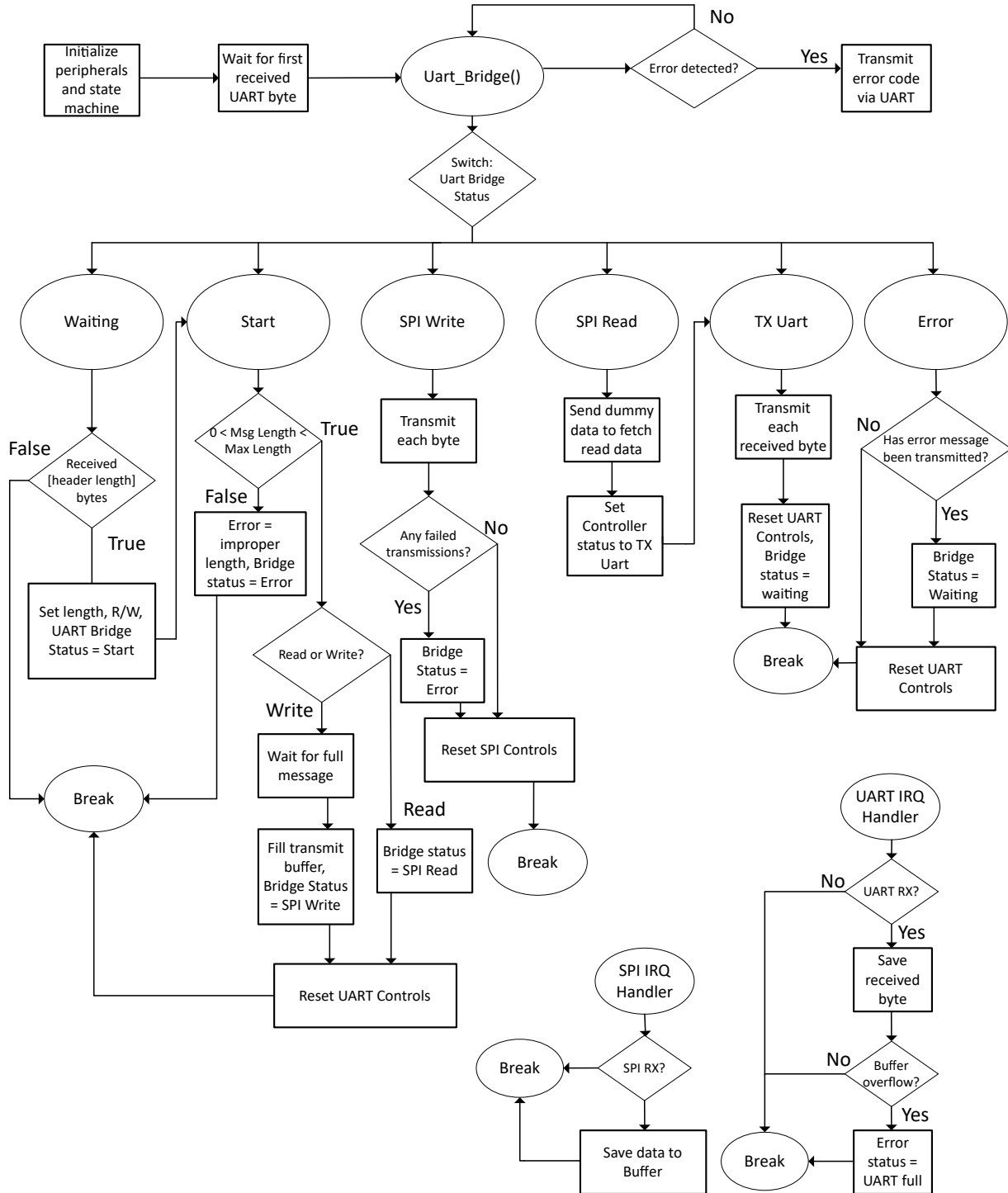


그림 69. 소프트웨어 흐름도

장치 구성

이 애플리케이션은 TI 시스템 구성 툴(**SYSCONFIG**) 그래픽 인터페이스를 사용하여 장치 주변 기기의 구성 코드를 생성합니다. 그래픽 인터페이스를 사용하여 장치 주변 기기를 구성하면 애플리케이션 프로토타입 제작 프로세스를 간소화할 수 있습니다.

소프트웨어 흐름도에 설명된 코드는 `uart_to_spi_bridge.c` 파일에서 확인할 수 있습니다.

필요한 UART 패킷

그림 70에서는 SPI로 읽기 및 쓰기를 수행하는 데 필요한 UART 패킷을 보여줍니다. 표시된 값은 예제에서 정의된 기본 헤더 값입니다.

- **시작 바이트:** 브리지에서 새 트랜잭션이 시작되었음을 나타내기 위해 사용하는 값입니다. 브리지에서 이 값을 감지할 때까지 UART 전송은 무시됩니다.
- **SPI 읽기 또는 쓰기 표시기:** 이 값은 브리지에 SPI 장치에 대한 읽기 또는 쓰기를 수행할지 여부를 알려줍니다.
- **메시지 길이 N:** 전송되는 데이터의 길이(바이트)입니다.
- **D0, D1, ..., D(N - 1):** 브리지로 전송되는 데이터입니다.

주

읽기 패킷에는 헤더만 포함됩니다. 읽기를 수행할 때는 패킷 이후에 데이터를 전송할 필요가 없습니다. 브리지 장치가 SPI 주변 기기로 올바른 양의 더미 데이터를 자동으로 전송하여 읽기 데이터를 가져옵니다.

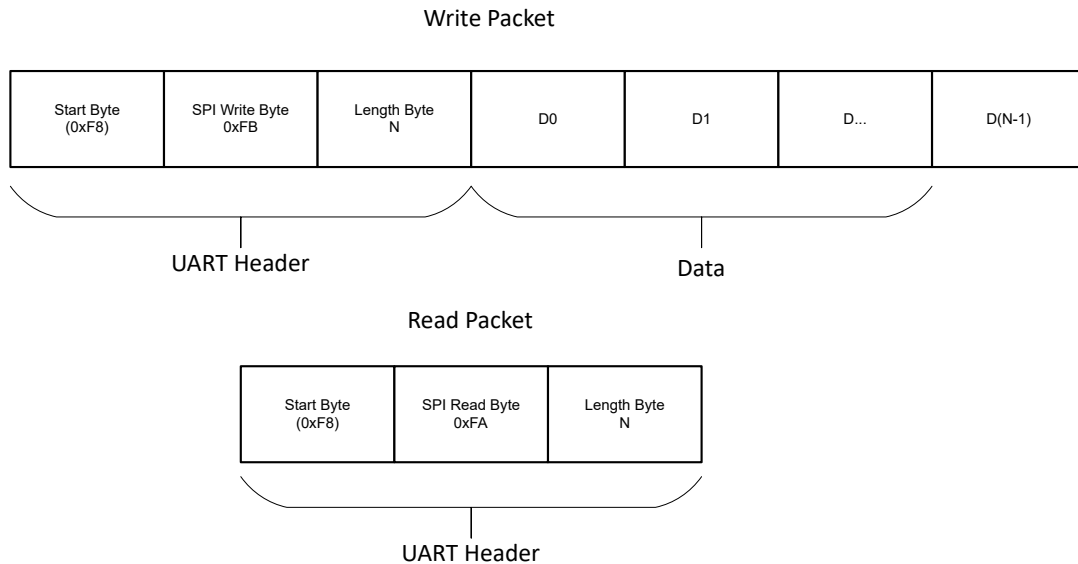


그림 70. UART 쓰기 및 읽기 패킷 형식

애플리케이션 코드

일부 사용자는 UART 패킷 헤더에서 사용되는 특정 값 또는 최대 패킷 크기를 변경하고자 합니다. 이러한 변경 작업은 다음 코드와 같이 `uart_to_spi_bridge.c` 파일의 시작 부분에 있는 `#define` 값을 수정하여 수행됩니다.

```
/* Define UART Header and Start Byte*/
#define UART_HEADER_LENGTH 0x02
#define UART_START_BYTE 0xF8
#define UART_READ_SPI_BYTE 0xFA
#define UART_WRITE_SPI_BYTE 0xFB
#define RW_INDEX 0x00
#define LENGTH_INDEX 0x01

/*Define max packet sizes*/
#define SPI_MAX_PACKET_SIZE (16)
#define UART_MAX_PACKET_SIZE (SPI_MAX_PACKET_SIZE + UART_HEADER_LENGTH)
```

코드의 많은 부분은 오류 감지 및 처리에 사용하기 위한 것입니다. 코드의 이러한 부분에서 사용자는 추가 오류 처리 또는 보고를 사용하여 보다 강력한 애플리케이션을 만들 수 있습니다. 예를 들어 다음 코드 세그먼트는 SPI 전송의 오류를 확인하고 오류 발생 시 오류 플래그를 설정하는 방법을 보여줍니다. 사용자는 전송을 종료하고 오류를 반영하기 위해 여기에서 UART 브리지 상태를 변경할 수 있습니다. 이 영역과 코드의 다른 많은 영역에는 오류를 고려하는 옵션이 있습니다.

```
for(int i = 0; i < gMsgLength; i++){
    if(!DL_SPI_transmitDataCheck8(SPI_0_INST, gSPIData[i])){
        gError = ERROR_SPI_WRITE_FAILED;
    }
}
```

추가 리소스

- 텍사스 인스트루먼트, [MSPM0 SDK 다운로드](#)
- 텍사스 인스트루먼트, [SysConfig에 대해 자세히 알아보기](#)
- 텍사스 인스트루먼트, [MSPM0L LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0G LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0 SPI 아카데미](#)
- 텍사스 인스트루먼트, [MSPM0 UART 아카데미](#)

E2E

TI의 E2E 지원 포럼 토론을 보고 새로운 스레드를 게시하여 설계에서 MSPM0 장치를 활용하는 데 필요한 기술 지원을 받으세요.

기타 MCU 기능

- 디지털 MUX 에뮬레이션
- 5V 인터페이스
- 작업 스케줄러

디지털 MUX 에뮬레이션

설명

디지털 멀티플렉서 소프트웨어 에뮬레이션에서는 GPIO 인터럽트를 사용하여 디지털 멀티플렉서를 에뮬레이션하는 방법을 보여줍니다. 로직 기반 멀티플렉서와 유사하게 MCU는 선택 신호(S0 및 S1)를 사용하여 주어진 시간에 어떤 입력 채널(C0, C1, C2 및 C3)이 출력이 되는지를 결정합니다. MCU를 통해 이를 수행하면 외부 멀티플렉서가 필요하지 않을 뿐만 아니라 PCB 라우팅을 지원할 수 있는 유연한 핀 할당이 가능합니다. 이 구체적인 예제에서는 4입력 채널, 2선택 신호 디지털 멀티플렉서를 에뮬레이션합니다.

그림 71에서는 이 서브시스템의 기능 블록 다이어그램을 보여줍니다.

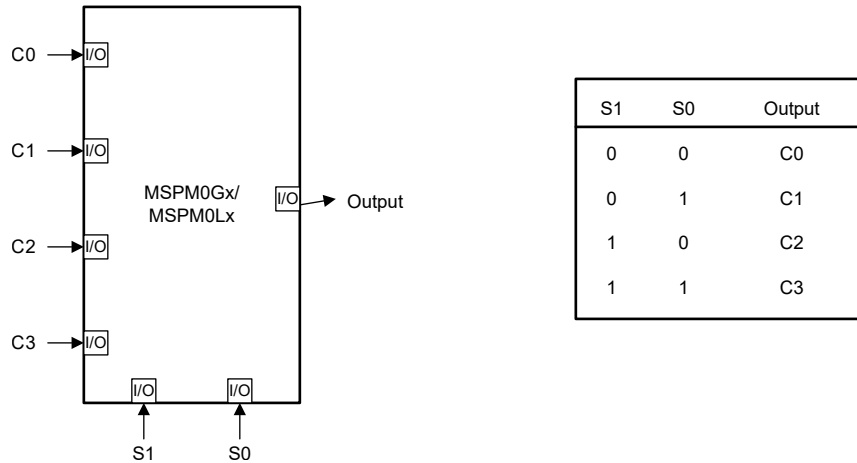


그림 71. 서브시스템 기능 블록 다이어그램

필요한 주변 기기

이 애플리케이션에는 GPIO 핀 7개 및 GPIO 인터럽트가 필요합니다.

표 42. 필요한 주변 기기

하위 블록 기능	참고
GPIO	핀 그룹은 코드에서 INPUT, OUTPUT 및 SELECT라고 합니다.

호환 가능 장치

표 42의 요구 사항을 바탕으로 호환 가능 장치가 **표 43**에 열거되어 있습니다. 해당 EVM은 빠른 평가를 위해 사용할 수 있습니다.

표 43. 호환 가능 장치

호환 가능 장치	EVM
MSPM0C	LP-MSPM0C1104
MSPM0Lx	LP-MSPM0L1306
MSPM0Gx	LP-MSPM0G3507

설계 단계

1. 애플리케이션에 필요한 GPIO의 수를 결정합니다. 이 예제에서는 입력 채널 핀 4개, 선택 핀 2개, 출력 핀 1개가 있습니다.
2. SysConfig에서 GPIO 출력 핀을 출력으로 구성합니다.

- 3. GPIO 입력 채널 핀을 구성하고 SysConfig에서 인터럽트 기능이 있는 입력으로 핀을 선택합니다.
- 4. 채널 및 SELECT 디지털 신호에 따라 출력을 변경하는 인터럽트에 대한 애플리케이션 코드를 작성합니다.

설계 고려 사항

- 1. 입력 채널 수 및 선택 핀 수: 4입력 멀티플렉서에는 선택 핀 2개가 필요합니다. 그러나 8입력 멀티플렉서에는 3개의 선택 핀이 필요합니다.
- 2. 로직 테이블: 선택 핀 구성에 따라 출력으로 선택되는 입력 채널이 결정됩니다.
- 3. 인터럽트: 출력 신호가 선택한 입력 채널에 따라 출력 신호를 설정하거나 지움으로써 생성되므로 모든 입력 채널 및 선택 핀에 인터럽트가 배치되어야 합니다.
- 4. 전파 지연: 인터럽트로 인한 전파 지연이 발생할 가능성이 있습니다. 전파 지연은 클록 속도에 따라 달라집니다.

소프트웨어 흐름도

그림 72에서는 이 서브시스템 예제의 소프트웨어 흐름도를 보여주고, 디지털 멀티플렉서를 에뮬레이션하는 데 사용되는 GPIO 인터럽트 루틴을 설명합니다.

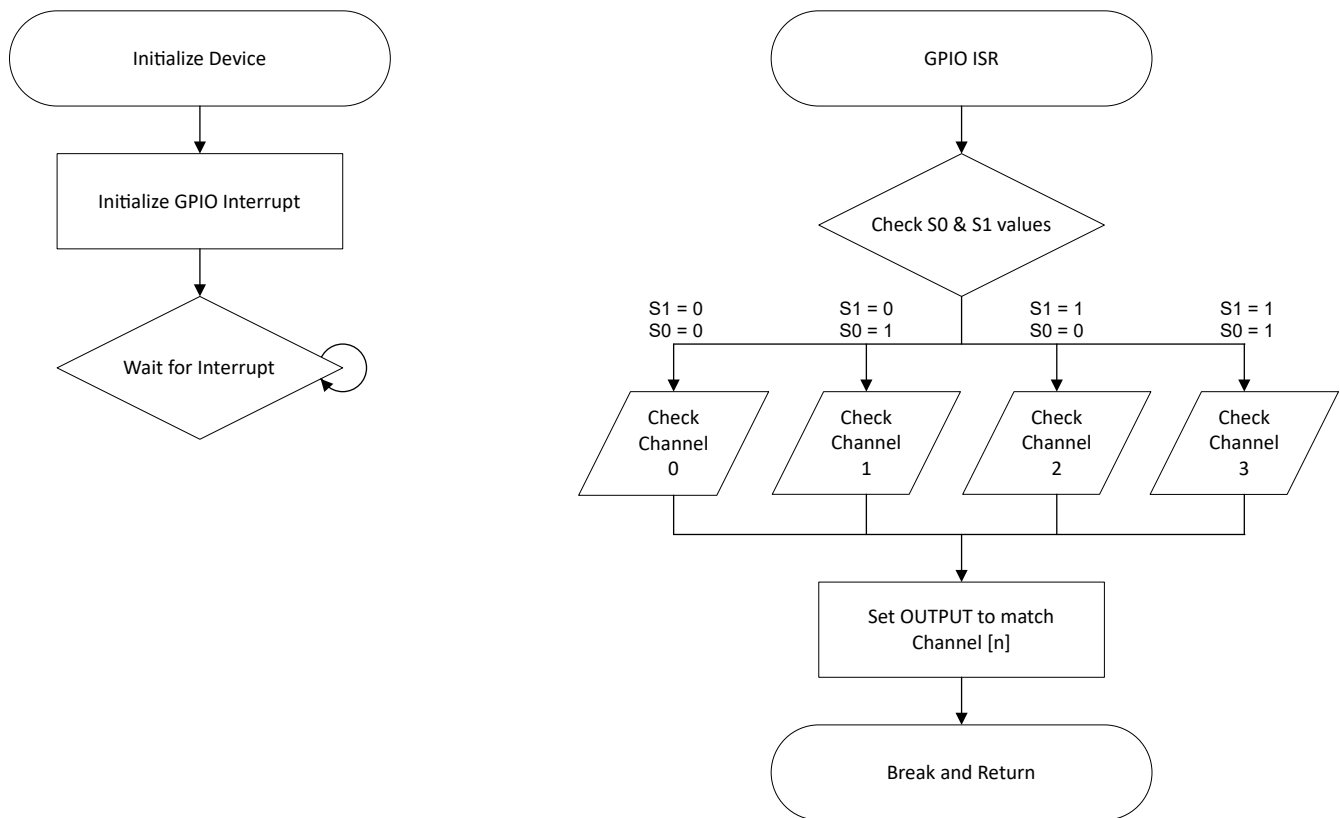


그림 72. 애플리케이션 소프트웨어 흐름도

애플리케이션 코드

이 애플리케이션은 **TI 시스템 구성 툴(SysConfig)** 그래픽 인터페이스를 사용하여 장치 주변 기기에 대한 구성 코드를 생성합니다. 그래픽 인터페이스를 사용하여 장치 주변 기기를 구성하면 애플리케이션 프로토타입 제작 프로세스를 간소화할 수 있습니다.

또한 SysConfig에서 이 애플리케이션은 GPIO 주변 기기에서 구성 및 활성화된 모든 입력 핀의 GPIO 인터럽트를 사용합니다. SysConfig에서 구성된 GPIO 핀에 따라 개별 GPIO 인터럽트도 NVIC_EnableIRQ(); 함수를 사용하여, 수동으로 코드의 main() 부분에서 활성화해야 합니다. 인터럽트를 활성화하면 main() 코드가 인터럽트를 기다립니다. 즉, 입력 신호 중 하나가 상태를 변경할 때마다 GPIO 인터럽트 서비스 루틴이 시작됩니다. 이 코드의 main() 부분은 다음과 같습니다.

```
int main(void)
{
    SYSCFG_DL_init();
    /* Enable GPIO Port A Interrupts */
    NVIC_EnableIRQ(GPIO_MULTIPLE_GPIOA_INT_IRQN);

    while (1) {
        __WFI();
    }
}
```

다음 코드 스니펫은 GPIO 인터럽트 서비스 루틴을 보여줍니다. 스위치 케이스는 인터럽트 유형에 대한 케이스와 출력으로 선택하는 입력 채널을 결정하는 케이스가 있습니다. 두 번째 스위치 케이스는 먼저 선택 핀을 점검하여 해당 상태를 결정합니다. 이러한 상태를 바탕으로 로직 진리표에 따라 입력 채널이 선택됩니다(**그림 71** 참조). 개별 케이스에 대해 선택한 입력 채널 핀이 확인되고 출력 핀이 일치하도록 설정됩니다. 그런 다음, 코드는 인터럽트 서비스 루틴을 중단하고 다른 인터럽트를 대기하도록 반환됩니다. 또한 이 예제 코드는 LP-MSPM0L1306의 핀 PA0을 출력 핀으로 사용하여 출력 신호를 바탕으로 빨간색 LED를 켜고 끕니다.

```
void GROUP1_IRQHandler(void){
    switch (DL_Interrupt_getPendingGroup(DL_INTERRUPT_GROUP_1)){
        case GPIO_MULTIPLE_GPIOA_INT_IIDX:
            switch (DL_GPIO_readPins(SELECT_PORT, SELECT_S1_PIN | SELECT_S0_PIN)){
                case 0: /* S1 = 0, S0 = 0 */
                    /* Check Channel 0 and set output to match */
                    if (DL_GPIO_readPins(INPUT_PORT, INPUT_CHANNEL_0_PIN)){
                        DL_GPIO_setPins(OUTPUT_PORT, OUTPUT_LED_PIN);
                    } else {
                        DL_GPIO_clearPins(OUTPUT_PORT, OUTPUT_LED_PIN);
                    }
                    break;
                case SELECT_S0_PIN: /* S1 = 0, S0 = 1 */
                    /* Check Channel 1 and set output to match */
                    if (DL_GPIO_readPins(INPUT_PORT, INPUT_CHANNEL_1_PIN)){
                        DL_GPIO_setPins(OUTPUT_PORT, OUTPUT_LED_PIN);
                    } else {
                        DL_GPIO_clearPins(OUTPUT_PORT, OUTPUT_LED_PIN);
                    }
                    break;
                case SELECT_S1_PIN: /* S1 = 1, S0 = 0 */
                    /* Check Channel 2 and set output to match */
                    if (DL_GPIO_readPins(INPUT_PORT, INPUT_CHANNEL_2_PIN)){
                        DL_GPIO_setPins(OUTPUT_PORT, OUTPUT_LED_PIN);
                    } else {
                        DL_GPIO_clearPins(OUTPUT_PORT, OUTPUT_LED_PIN);
                    }
                    break;
                case SELECT_S1_PIN | SELECT_S0_PIN: /* S1 = 1, S0 = 1 */
                    /* Check Channel 3 and set output to match */
                    if (DL_GPIO_readPins(INPUT_PORT, INPUT_CHANNEL_3_PIN)){
                        DL_GPIO_setPins(OUTPUT_PORT, OUTPUT_LED_PIN);
                    } else {
                        DL_GPIO_clearPins(OUTPUT_PORT, OUTPUT_LED_PIN);
                    }
            }
    }
}
```

```

        break;
    }
    break;
}

```

결과

그림 73에서는 다른 입력-출력 신호의 로직 캡처를 보여줍니다. 입력 채널 C0~C3은 각각 흰색, 갈색, 빨간색 및 주황색으로 구분됩니다. S0은 노란색이고 S1은 녹색입니다. 마지막으로 출력 신호는 파란색입니다. 캡처는 다양한 입력이 출력 신호를 어떻게 변경하는지 보여주기 위해 표시됩니다.

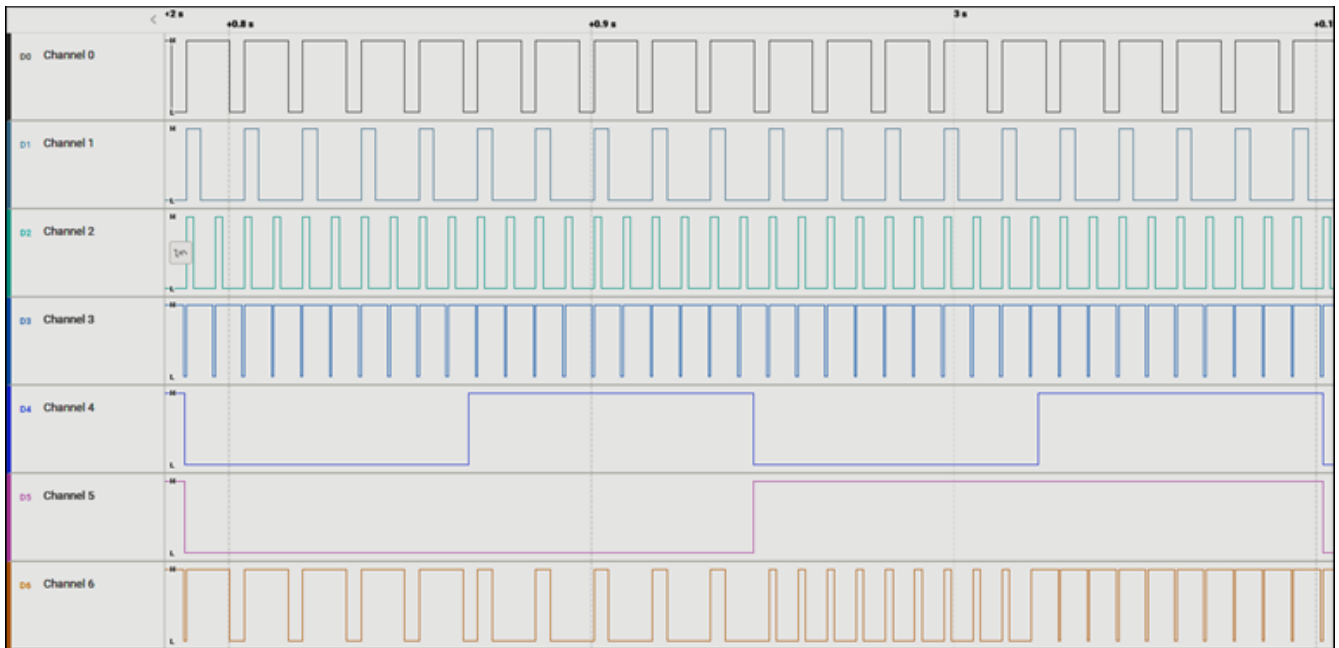


그림 73. 결과

추가 리소스

- 텍사스 인스트루먼트, [MSPM0 SDK 다운로드](#)
- 텍사스 인스트루먼트, [SysConfig에 대해 자세히 알아보기](#)
- 텍사스 인스트루먼트, [MSPM0L LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0G LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0C LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0 아카데미](#)

E2E

TI의 E2E™ 지원 포럼 토론을 보고 새로운 스레드를 게시하여 설계에서 MSPM0 장치를 활용하는 데 필요한 기술 지원을 받으세요.

5V 인터페이스

설명

이 예제에서는 MSPM0 장치에서 ODIO(오픈 드레인 IO)를 사용하여 최대 5V의 신호와 상호 작용하는 방법을 보여줍니다. 외부 풀업 저항을 사용하는 오픈 드레인 IO를 통해 MSPM0 V_{DD} 공급 전압보다 높은 전압 레벨에서 여러 전압 도메인 간에 통신이 가능합니다.

그림 74에서는 이 예제에서 사용된 주변 기기의 기능 블록 다이어그램을 보여줍니다.

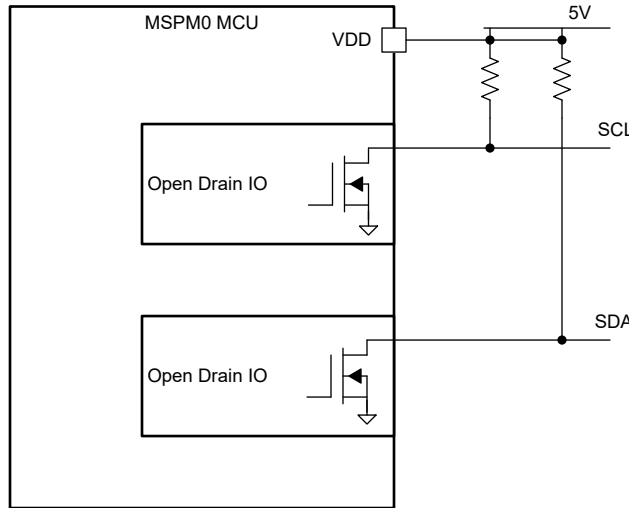


그림 74. 서브시스템 기능 블록 다이어그램

필요한 주변 기기

이 애플리케이션은 최대 2개의 오픈 드레인 IO를 사용할 수 있습니다.

표 44.

하위 블록 기능	사용되는 주변 기기	참고
IO	2개의 GPIO 핀	PA0 및 PA1, 5V 허용 오픈 드레인 IO만 사용 가능

호환 가능 장치

표 44의 요구 사항에 따라 이 예제는 표 45의 장치와 호환됩니다. 해당 EVM은 프로토타입 제작에 사용할 수 있습니다.

표 45.

호환 가능 장치	EVM
MSPM0Lxxx	LP-MSPM0L1306
MSPM0Gxxx	LP-MSPM0G3507

설계 단계

1. 적절한 접퍼를 연결합니다.
2. 애플리케이션에 필요한 풀업 저항을 결정합니다.
 - a. 필요한 풀업 강도는 애플리케이션의 타이밍 요구 사항과 연결 커패시턴스에 따라 달라집니다. 커패시턴스가 클수록 더 강력한(즉, 낮은 저항) 풀업이 필요합니다. 풀업의 정확한 저항 결정에 대한 논의는 이 문서의 범위를 벗어나지만 [I2C 버스 풀업 저항 계산 애플리케이션 노트](#)에서 확인할 수 있습니다.
3. **SysConfig**의 소프트웨어(예: UART, I2C 또는 타이머)에서 이러한 핀에 사용되는 주변 기기를 구성합니다.
4. 사용되는 주변 기기에 따라 애플리케이션 코드를 작성합니다.

설계 고려 사항

1. 풀업 저항: ODIO의 I2C 및 UART 기능을 위한 높은 출력을 위해서는 풀업 저항이 필요합니다.
2. 구동 강도 제어: 이 기능은 ODIO 유형에 사용할 수 없습니다.

추가 리소스

- [MSPM0 SDK 다운로드](#)
- [SysConfig에 대해 자세히 알아보기](#)
- [MSPM0L LaunchPad](#)
- [MSPM0G LaunchPad](#)

작업 스케줄러

설명

이 서브시스템 예제에서는 MSPM0에서 간단한 비선점형 RTC(실행-완료) 스케줄러를 구현하는 방법을 보여줍니다. 이 예제에는 스케줄러와 간단한 작업 헤더 및 소스 파일이 모두 포함되어 있으며, 이러한 종류의 스케줄링 구현을 위한 작업을 빌드하기 위한 최소 요구 사항을 보여줍니다. 시스템에서 시스템에 의해 완료되어야 하고, 임의의 순서로 트리거될 수 있는 여러 작업이 있으며, 이러한 작업의 실제 실행 시간이나 순서가 중요하지 않은 경우 RTC 스케줄러를 사용하는 것이 가장 적절합니다.

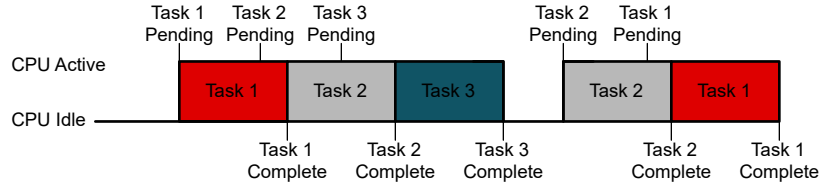


그림 75. 실행-완료 스케줄러

필요한 주변 기기

작업 스케줄러 서브시스템은 일반적이며 MSPM0 포트폴리오의 모든 장치에 적합합니다. 표 46에는 예제 작업에 사용된 주변 기기가 나열되어 있지만, 예제의 스케줄러 부분을 사용하는 데 반드시 필요한 것은 아닙니다.

표 46. 필요한 주변 기기

하위 블록 기능	사용되는 주변 기기	참고
DAC8(옵션)	(1 x) 비교기	코드에서 COMP_0_INST로 표시됨
버퍼(옵션)	(1 x) OPA	코드에서 OPA_0_INST로 표시됨
타이머(옵션)	(1 x) TIMG	코드에서 TIMER_0_INST로 표시됨
LED 출력(옵션)	(1 x) GPIO	코드에서 GPIO_LEDS_USER_LED_1로 표시됨
스위치 입력(옵션)	(1 x) GPIO	코드에서 GPIO_SWITCHES_USER_SWITCH_1로 표시됨

호환 가능 장치

표 46에 표시된 요구 사항에 따라 예제 코드는 표 47에 표시된 장치와 호환됩니다.

표 47. 호환 가능 장치

호환 가능 장치	EVM
MSPM0Lx	LP-MSPM0L1306
MSPM0Gx	LP-MSPM0G3507
MSPM0Cx(DAC8 및 버퍼 미사용)	LP-MSPM0C1104

설계 단계

간단한 스케줄러 애플리케이션을 구현하려면 다음 사항을 완료합니다.

1. 예제 서브시스템 프로젝트로 시작하거나, 기존 프로젝트에 scheduler 소스 및 헤더 파일을 추가합니다.
2. scheduler 함수는 애플리케이션의 메인 소프트웨어 루프 역할을 하도록 구성되었습니다. 초기화 후 [섹션 4.3.7](#)에서와 같이 scheduler 함수에 호출을 추가합니다.
3. 시스템에서 수행할 각 작업에 대해 해당하는 작업의 보류 중인 플래그를 가져오고 설정하고 리셋하는 함수를 만듭니다. 또한 스케줄러가 실행을 시도할 때 실행할 실제 함수도 만듭니다. DAC8Driver 및 SwitchDriver 소스 및 헤더 파일은 이를 수행하는 방법에 대한 간단한 예를 제공합니다.
4. 적절한 IRQ(인터럽트 요청) 처리기를 추가하여 필요한 하드웨어 이벤트를 기반으로 보류 중인 작업을 활성화합니다. IRQ 처리기는 보류 중인 작업 플래그를 설정하고 보류 중인 작업 카운터를 증가시킵니다. 이러한 값은 장치가 시스템 인터럽트에 의해 절전 모드에서 해제될 때 scheduler에 의해 확인됩니다.

설계 고려 사항

작업 스케줄러 서브시스템에 작업을 통합할 때 다음 사항을 고려합니다.

1. 여러 인터럽트 또는 작업이 동시에 대기열에 있는 경우 기본 스케줄러 루프는 작업이 gTasksList에 표시되는 순서대로 작업을 처리합니다. 이는 선점형은 아니지만 단순한 우선 순위 지정 방식으로 간주할 수 있습니다.
2. 모든 작업은 이 아키텍처에서 인터럽트에 의해 처리됩니다. 즉, 적절한 IRQ 처리기가 실행될 작업과 관련된 보류 플래그를 설정해야 합니다. 시스템에서 하나의 이벤트 작업만 가능한 경우 플래그가 아직 설정되지 않은 경우에만 gTasksPendingCounter를 증가시킵니다. 이벤트가 여러 번 동시에 대기열에 있어야 하는 경우 보류 플래그에 엄격한 true 또는 false의 부울 값이 아닌 정수 값을 사용합니다.

소프트웨어 흐름도

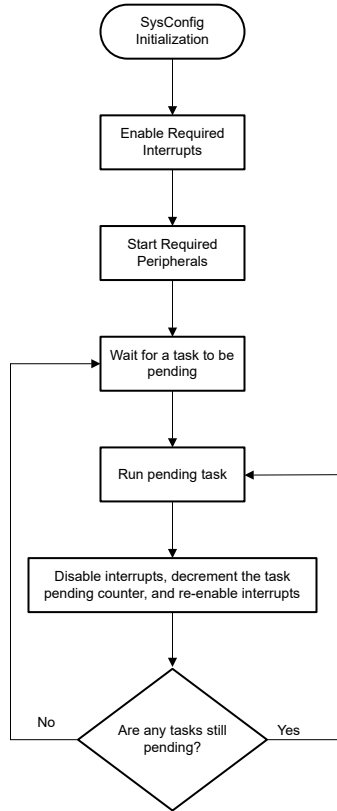


그림 76. 애플리케이션 소프트웨어 흐름도

애플리케이션 코드

스케줄러 코드

스케줄러 코드는 modules/scheduler/scheduler.c 파일에 저장되어 있으며, 스케줄러가 gTasksList에 액세스하는 데 필요한 모든 함수 포인터 목록을 포함합니다. 각 작업은 준비 플래그나 보류 중 플래그를 가져오고 리셋하는 함수, 실행할 작업에 대한 포인터를 제공할 수 있습니다.

스케줄러 루프 내에서 gTasksPendingCounter 값은 보류 중인 작업 수를 추적합니다. 루프가 보류 중인 각 작업 플래그를 순환하면서 루프가 보류 중인 작업 플래그를 발견하면 스케줄러 루프가 이 카운터를 감소시킵니다. 모든 작업이 지워지면 장치는 `__WFI`에 대한 호출을 통해 저전력 모드로 전환됩니다.

```
#include "scheduler.h"
#define NUM_OF_TASKS 2 /* Update to match required number of tasks */
volatile extern int16_t gTasksPendingCounter;

/*
 * Update gTasksList to include function pointers to the
 * potential tasks you want to run. See DAC8Driver and
 * switchDriver code and header files for examples.
 */
static struct task gTasksList[NUM_OF_TASKS] =
{
    { .getRdyFlag = getSwitchFlag, .resetFlag = resetSwitchFlag, .taskRun = runSwitchTask },
    { .getRdyFlag = getDACFlag, .resetFlag = resetDACFlag, .taskRun = runDACTask },
/* { .getRdyFlag = , .resetFlag = , .taskRun = }, */
};

void scheduler() {
    /* Iterate through all tasks and run them as necessary */
    while(1) {
        /*
         * Iterate through tasks list until all tasks are completed.
         * Checking gTasksPendingCounter prevents us from going to
         * sleep in the case where a task was triggered after we
         * checked its ready flag, but before we went to sleep.
         */
        while(gTasksPendingCounter > 0)
        {
            for(uint16_t i=0; i < NUM_OF_TASKS; i++)
            {
                /* Check if current task is ready */
                if(gTasksList[i].getRdyFlag())
                {
                    /* Execute current task */
                    gTasksList[i].taskRun();
                    /* Reset ready for for current task */
                    gTasksList[i].resetFlag();
                    /* Disable interrupts during read, modify, write. */
                    __disable_irq();
                    /* Decrement pending tasks counter */
                    (gTasksPendingCounter)--;
                    /* Re-enable interrupts */
                    __enable_irq();
                }
            }
        }
        /* Sleep after all pending tasks are completed */
        __WFI();
    }
}
```

메인 애플리케이션 코드

스케줄러 작동 및 작업을 위한 장치 초기화는 메인 애플리케이션 소스 코드 파일인 task_scheduler.c 내에서 처리됩니다. SYSCFG_DL_init을 호출하면 예제 코드에 필요한 하드웨어 주변 기기가 구성된 후 인터럽트가 활성화되고 TIMER_0_INST 카운터가 시작됩니다. 그런 다음, 코드가 스케줄러 루프로 들어갑니다.

필요한 IRQ 처리기 내에서 인터럽트 중에 적절한 플래그가 설정되어 스케줄러에게 작업이 보류 중임을 알립니다.

```
#include "ti_msp_dl_config.h"
#include "modules/scheduler/scheduler.h"

/* Counter for the number of tasks pending */
volatile int16_t gTasksPendingCounter = 0;

int main(void)
{
    SYSCFG_DL_init();

    /* Enable IRQs */
    NVIC_EnableIRQ(GPIO_SWITCHES_INT_IRQN);
    NVIC_EnableIRQ(TIMER_0_INST_INT_IRQN);

    /* Start timer to update DAC8 output */
    DL_TimerG_startCounter(TIMER_0_INST);

    /* Enter Task Scheduler */
    scheduler();
}

/* Interrupt Handler for S2 (PB21) button press, toggles LED */
void GROUP1_IRQHandler(void)
{
    switch (DL_Interrupt_getPendingGroup(DL_INTERRUPT_GROUP_1)) {
        /* S2 (PB21) has been pressed execute PB21 task */
        case GPIO_SWITCHES_INT_IIDX:
            /* Increment counter if ready flag is not already set. */
            gTasksPendingCounter += !getSwitchFlag();
            setSwitchFlag();
            break;
    }
}

/* Interrupt Handler for TIM0 zero condition, updates DAC8 value */
void TIMER_0_INST_IRQHandler(void)
{
    switch (DL_TimerG_getPendingInterrupt(TIMER_0_INST)) {
        case DL_TIMER_IIDX_ZERO:
            /* Increment counter if ready flag is not already set. */
            gTasksPendingCounter += !getDACFlag();
            setDACFlag();
            break;
        default:
            break;
    }
}
}
```

추가 리소스

- 텍사스 인스트루먼트, [MSPM0 SDK 다운로드](#)
- 텍사스 인스트루먼트, [SysConfig에 대해 자세히 알아보기](#)
- 텍사스 인스트루먼트, [MSPM0C LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0L LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0G LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0 아카데미](#)

E2E

TI의 **E2E™** 지원 포럼 토론을 보고 새로운 스레드를 게시하여 설계에서 MSPM0 장치를 활용하는 데 필요한 기술 지원을 받으세요.

타이밍 및 제어

- 연결된 다이오드 매트릭스
- 주파수 카운터: 톤 감지
- PWM을 사용하는 LED 드라이버
- 전원 시퀀서
- PWM DAC

연결된 다이오드 매트릭스

설명

연결된 다이오드 매트릭스 예제에서는 6개 이상의 LED를 사용할 때 매트릭스 형식을 사용하여 필요한 GPIO 핀 수를 줄이는 방법을 보여줍니다. 이 구체적인 예제에서는 9개의 LED와 6개의 GPIO를 사용하여 3 × 3 LED 매트릭스를 형성하고 제어합니다. 매트릭스 형식은 LED(또는 다이오드)당 GPIO 2개를 사용하는 그리드를 생성합니다. 이 형식은 LED로 표지판이나 디스플레이를 만들 때 특히 유용합니다. LED 매트릭스용 GPIO 핀은 행 핀과 열 핀으로 나뉩니다. 행 핀이 LED의 음극에 연결되면, **그림 77**에서 볼 수 있듯이 매트릭스는 공통 행 음극입니다. 공통 행 양극은 행 핀이 LED의 양극을 연결하는 경우입니다. LED 매트릭스에서 LED 구성에 따라 행 및 열 핀은 액티브 하이 또는 액티브 로우로 설정됩니다. 이 서브시스템 예제에서 행 핀은 액티브 로우이며 열 핀은 액티브 하이입니다. LED 매트릭스가 제대로 작동하려면 매트릭스의 LED를 한 번에 한 행씩 제어해야 합니다. 이 예제의 애플리케이션 코드는 상태 시스템을 사용하여 행을 지속적으로 순환하면서 LED를 켜고 끕니다.

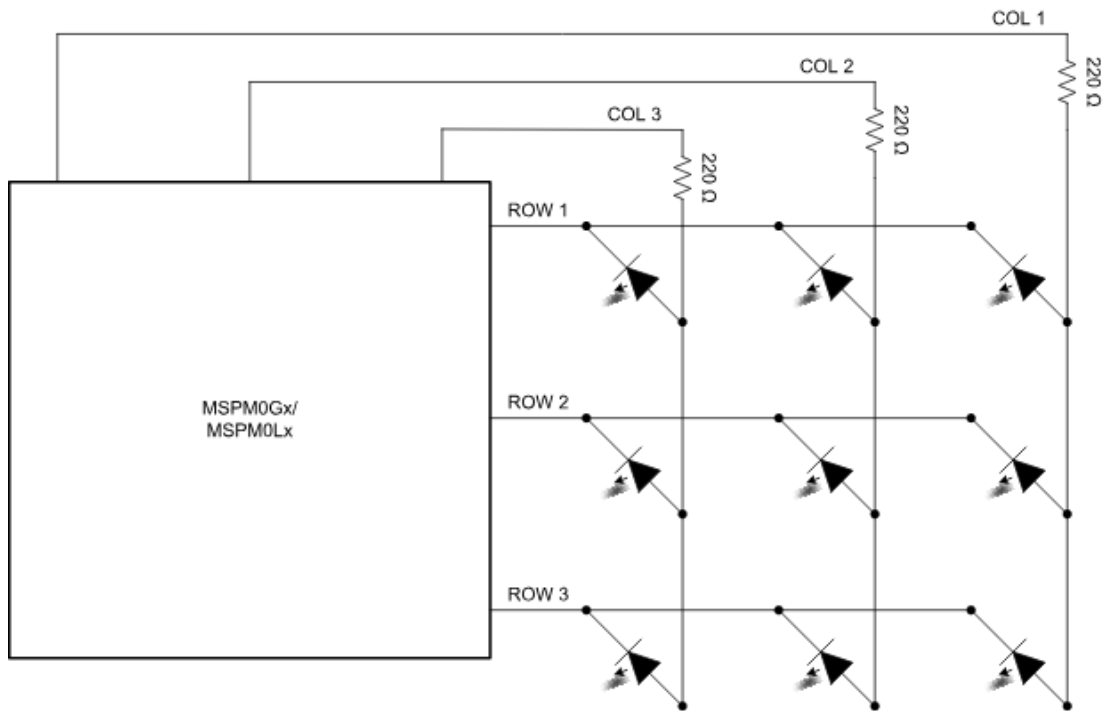


그림 77. 서브시스템 기능 블록 다이어그램

필요한 주변 기기

이 애플리케이션에는 GPIO 핀 6개 및 타이머 인터럽트가 필요합니다.

표 48. 필요한 주변 기기

하위 블록 기능	사용되는 주변 기기	참고
GPIO 하위 블록	6개의 GPIO 핀	이 예제에 사용된 모든 핀은 동일한 포트에 위치함
타이머	타이머 인터럽트	타이머 인터럽트는 LED 매트릭스의 행을 순환하는 데 사용됨

호환 가능 장치

표 48의 요구 사항을 바탕으로 호환 가능 장치가 표 49에 열거되어 있습니다. 해당 EVM은 빠른 평가를 위해 사용할 수 있습니다.

표 49. 호환 가능 장치

호환 가능 장치	EVM
MSPM0Lx	LP-MSPM0L1306
MSPM0Gx	LP-MSPM0G3507

설계 단계

1. 매트릭스에 사용되는 LED 수와 매트릭스 크기를 결정합니다. 매트릭스 크기에 따라 필요한 GPIO 핀 수가 결정됩니다.
2. GPIO 핀을 행 핀과 열 핀으로 분리합니다.
3. 모든 행 및 열 핀을 출력으로 구성합니다.
4. 모든 열 핀 GPIO 값의 비트 단위 OR를 가져와 열 핀의 마스크 값을 결정합니다.
5. 메모리 테이블 및 메모리 테이블 업데이트 기능을 생성합니다.
6. 행 사이를 순환할 행 업데이트 상태 시스템에 대한 열거형 테이블을 생성합니다.
7. 타이머 인터럽트를 구성하고 행 업데이트 상태 시스템에 대한 애플리케이션 코드를 작성하여 LED 상태를 증분시킵니다.
8. 표시 시간을 설정하고 표시가 변경되면 새 열 핀 값으로 메모리 테이블을 업데이트하는 애플리케이션 코드를 작성합니다.

설계 고려 사항

1. **LED 수 및 매트릭스 크기:** 매트릭스 크기는 매트릭스를 실행하는 데 필요한 GPIO 핀의 수를 결정합니다. 예를 들어 16개의 LED 매트릭스는 4 × 4 매트릭스의 8핀 또는 2 × 8 매트릭스의 10핀을 사용할 수 있습니다.
2. **LED 구성:** 행 및 열 핀의 활성 상태는 매트릭스가 공통 행 음극인지 아니면 공통 행 양극인지에 따라 달라집니다.
3. **열 핀 값:** 열 핀 값은 메모리 테이블에 설정됩니다. 정확한 값은 선택된 핀과 각 컬럼 마스크에 따라 결정됩니다. 쉽게 설정할 수 있도록 간격 없이 숫자 순서대로 핀을 선택하는 것이 가장 쉽습니다.
4. **열 및 행 핀 연결:** LED 매트릭스에 핀을 연결할 때 행 핀이 맨 위 행에서 시작되고(아래로 이동) 열 핀이 맨 오른쪽 열에서 시작하는(왼쪽으로 이동) 경우에 애플리케이션 프로그래밍이 가장 쉽습니다.
5. **타이머 인터럽트:** 인터럽트 속도에 따라 표시 시간과 각 LED 행이 상태 시스템 사이클당 얼마나 오래 켜져 있는지에 영향을 미칩니다. 이 구체적인 예제는 5ms마다 인터럽트하여 사람의 눈이 깜빡임을 인식하지 못하게 합니다.
6. **메모리 테이블 업데이트:** 메모리 테이블을 업데이트하는 구체적인 방법은 애플리케이션에 따라 다릅니다. 이 예제에서는 카운터(표시 시간이라고도 함)를 지정된 값까지 증분합니다. 카운터가 이 값에 도달하면 메모리 테이블이 업데이트되어 새로운 표시가 설정됩니다.

소프트웨어 흐름도

그림 78에서는 이 서브시스템 예제의 소프트웨어 흐름도를 보여주고, LED 매트릭스를 제어하는 데 사용되는 타이머 인터럽트 루틴 및 상태 시스템을 설명합니다.

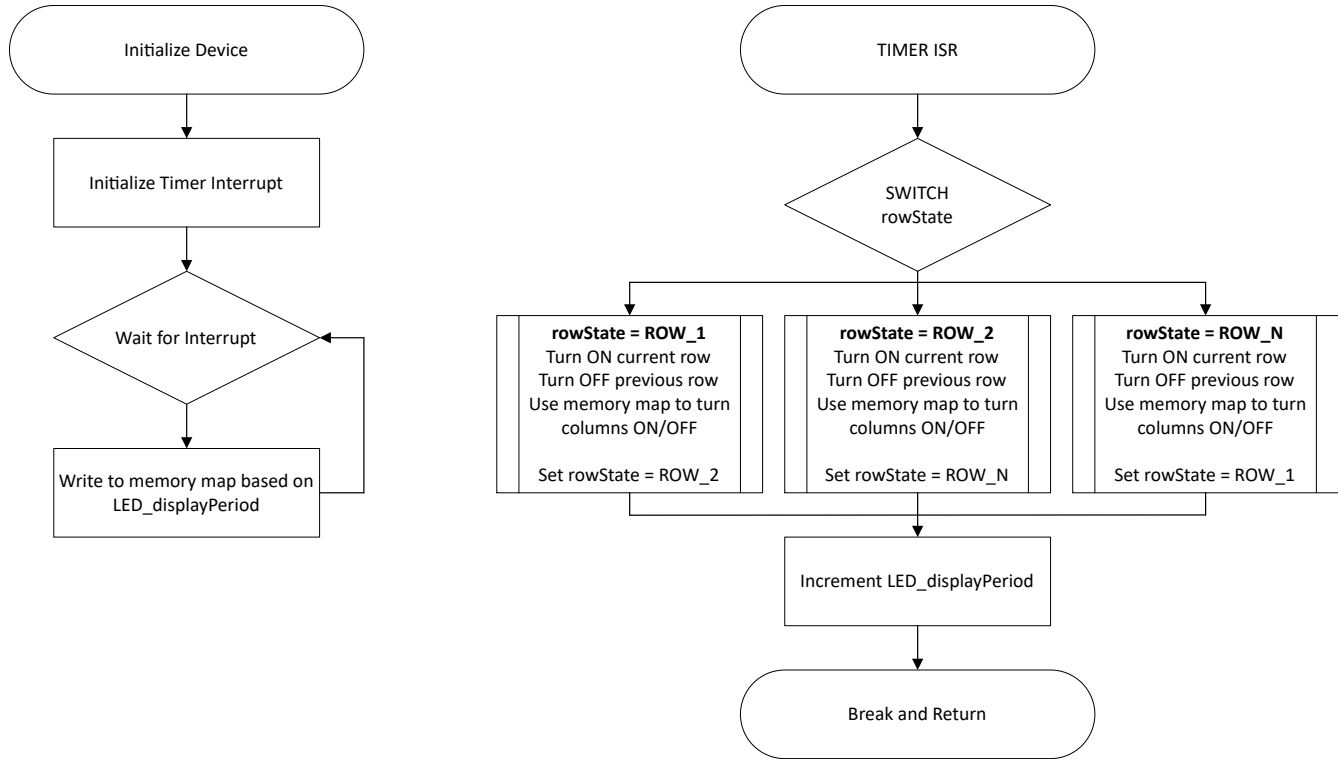


그림 78. 애플리케이션 소프트웨어 흐름도

애플리케이션 코드

이 애플리케이션은 TI 시스템 구성 툴(SysConfig) 그래픽 인터페이스를 사용하여 장치 주변 기기에 대한 구성 코드를 생성합니다. 그래픽 인터페이스를 사용하여 장치 주변 기기를 구성하면 애플리케이션 프로토타입 제작 프로세스를 간소화할 수 있습니다.

이 예제에서는 행 수, 컬럼 마스크 값, 표시 기간 및 인터럽트 수를 추적하는 카운터와 같은 몇 가지 주요 변수를 사용합니다. 행 수는 메모리 테이블 배열을 구축하는 데 사용되는 정의된 값입니다. 컬럼 마스크는 사용되는 모든 컬럼 핀의 GPIO 값의 비트 단위 OR에 상응합니다. 컬럼 마스크는 메모리 테이블과 함께 사용되어 주어진 시간에 행별로 켜거나 꺼야 할 컬럼 핀을 결정합니다. 표시 기간 변수에 타이머 인터럽트당 지속 시간을 곱하여 단일 메모리 테이블 쓰기가 사용되는 시간을 결정합니다. 이 예제에서 표시 기간 값은 100으로 정의되며, 이 값은 0.5초의 표시 기간과 같습니다. 표시 기간 값과 관련된 인터럽트 수를 추적하는 데 사용되는 카운터 또는 gLedState입니다. 이를 통해 모든 표시 기간에 메모리 테이블이 기록됩니다.

```

#define NUMBER_OF_ROWS 3
#define COL_MASK 0x38
#define LED_DISPLAY_PERIOD 100 /* timer period = 5 ms, so display period = 500 ms */
volatile uint32_t gLedState = 0;
void LED_updateTable(uint8_t rowNumber, uint8_t LEDs);
    
```

다음 코드 조각에서는 열거형 테이블과 타이머 IRQ(인터럽트 요청)를 보여줍니다. 열거형 테이블은 rowState 스위치가 타이머 IRQ에서 순환하는 행 상태를 정의합니다. 각 rowState(또는 행 핀)에 대해 현재 행이 켜지고 이전 행은 꺼지며, 열은 컬럼 마스크 값과 메모리 테이블 값을 비교하여 설정됩니다. 그런 다음 rowState가 설정됩니다. 이 예제는 1행에서 N행으로 순차적으로 순환한 후 다시 1행으로 순환합니다. 타이머 IRQ에서 나가기 전에 gLedState가 증가하여 각 표시 기간에 대한 인터럽트 수를 추적합니다.

```
typedef enum {
    ROW_1,
    ROW_2,
    ROW_3
}rowNumber;

rowNumber rowState = ROW_1;

void LED_STATE_INST_IRQHandler(void) {
    switch (DL_TimerG_getPendingInterrupt(LED_STATE_INST)){
        case DL_TIMER_IIDX_ZERO:
            /* State machine to auto cycle from row 1 to row N and repeat */
            switch (rowState){
                case ROW_1:
                    /* Turn on ROW_1, Turn off ROW_3 */
                    DL_GPIO_clearPins(ROW_PORT, ROW_ROW_1_PIN);
                    DL_GPIO_setPins(ROW_PORT, ROW_ROW_3_PIN);

                    /* Set COLUMN values */
                    DL_GPIO_writePinsVal(COLUMN_PORT, COL_MASK, gLedMemoryTable[0]);
                    rowState = ROW_2;
                    break;
                case ROW_2:
                    /* Turn on ROW_2, Turn off ROW_1 */
                    DL_GPIO_clearPins(ROW_PORT, ROW_ROW_2_PIN);
                    DL_GPIO_setPins(ROW_PORT, ROW_ROW_1_PIN);

                    /* Set COLUMN values */
                    DL_GPIO_writePinsVal(COLUMN_PORT, COL_MASK, gLedMemoryTable[1]);
                    rowState = ROW_3;
                    break;
                case ROW_3:
                    /* Turn on ROW_3, Turn off ROW_2 */
                    DL_GPIO_clearPins(ROW_PORT, ROW_ROW_3_PIN);
                    DL_GPIO_setPins(ROW_PORT, ROW_ROW_2_PIN);

                    /* Set COLUMN values */
                    DL_GPIO_writePinsVal(COLUMN_PORT, COL_MASK, gLedMemoryTable[2]);
                    rowState = ROW_1;
                    break;
            }

            /* Increment LED_STATE */
            gLedState++;

            break;
        default:
            break;
    }
}
```

메인 코드에서 수행되는 모든 작업은 표시 기간마다 메모리 테이블에 기록하는 것입니다. 이 작업은 무한 반복됩니다. 1과 0의 레이아웃은 매트릭스 레이아웃을 모방하기 때문에 어느 LED가 켜져 있는지 더 쉽게 확인하기 위해 이 특정 코드에서는 이진을 사용합니다. LED가 켜져 있는 경우 이진 값은 1이고 LED가 꺼진 경우에는 0입니다.

```
while(1){
  __WFI();
  /* Flash TI on repeat in half second increments */
  if (gLedState == LED_DISPLAY_PERIOD){ /* Display "T" for one display period */
    LED_updateTable(1, 0b111);
    LED_updateTable(2, 0b010);
    LED_updateTable(3, 0b010);
  } else if (gLedState == LED_DISPLAY_PERIOD*2){ /* Blank for one display period */
    LED_updateTable(1, 0b000);
    LED_updateTable(2, 0b000);
    LED_updateTable(3, 0b000);
  } else if (gLedState == LED_DISPLAY_PERIOD*3){ /* Display "I" for one display period */
    LED_updateTable(1, 0b111);
    LED_updateTable(2, 0b010);
    LED_updateTable(3, 0b111);
  } else if (gLedState == LED_DISPLAY_PERIOD*4){ /* Blank for one display period */
    LED_updateTable(1, 0b000);
    LED_updateTable(2, 0b000);
    LED_updateTable(3, 0b000);
  } else if (gLedState > LED_DISPLAY_PERIOD*4){ /* Reset gLedState and start over */
    gLedState = 0;
  }
}
```

하드웨어 설계

이 특정 서브시스템 예제에는 9개의 LED, 3개의 저항 및 최소 6개의 와이어가 필요합니다. 매트릭스를 설정하려면 LED를 3 × 3행으로 정렬합니다. 각 LED 행의 음극을 모두 연결합니다. 그런 다음, 각 LED 열의 양극을 모두 연결합니다. 220Ω 저항을 각 열 라인에 연결합니다. 여기서부터 장치 구성에 따라 행 라인과 열 라인을 올바른 장치 핀에 연결합니다. [그림 77](#)에서 연결 가이드라인을 참조하세요.

결과

그림 79에서는 이 애플리케이션에 대한 "T" 표시 기간의 의도된 결과를 보여줍니다. 이 그림의 상단은 애플리케이션 상태 시스템이 인터럽트당 각 행을 순환함에 따른 각 행의 상태를 보여줍니다. 그림의 하단은 전체 사이클에서 합성 이미지가 어떤지 보여줍니다. 이 이미지는 사람의 눈에 보이는 매트릭스의 모습입니다.

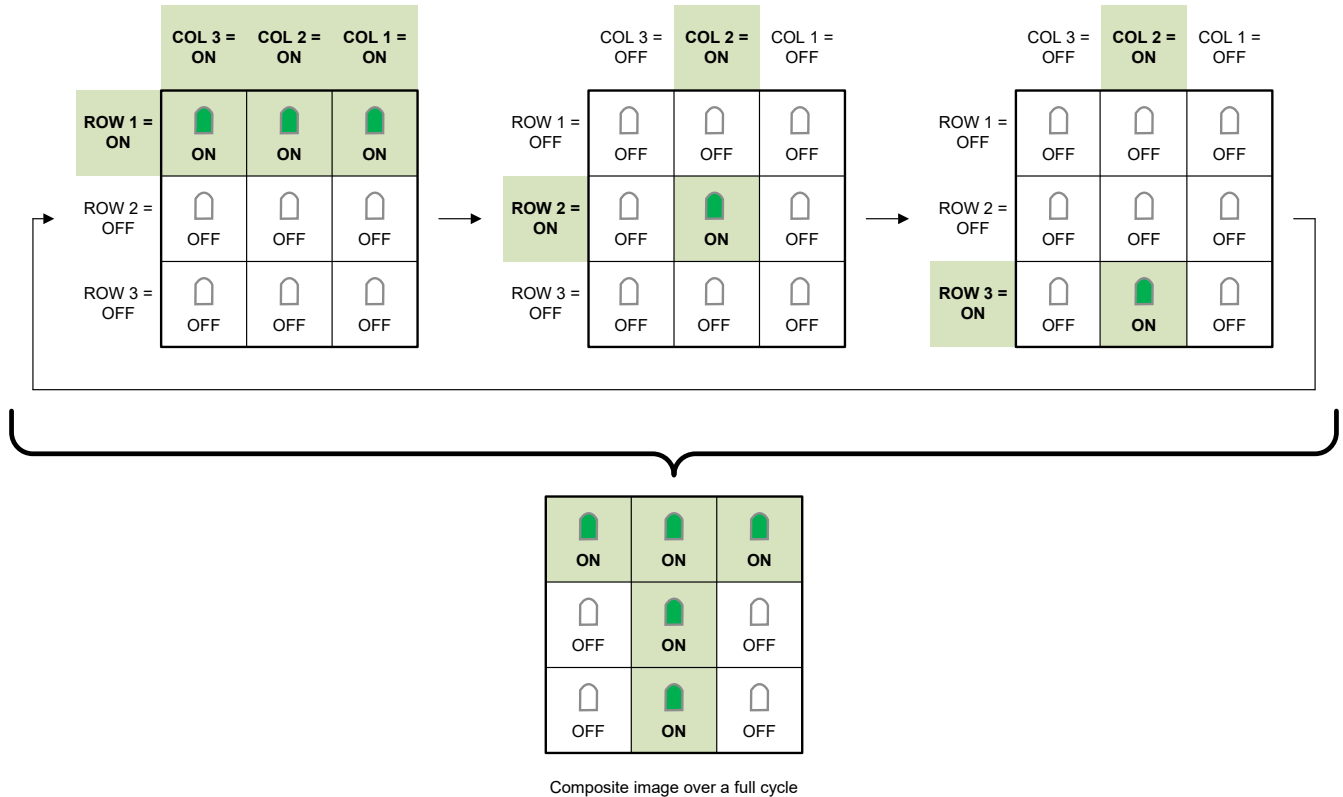


그림 79. 결과

추가 리소스

- 텍사스 인스트루먼트, [MSPM0 SDK 다운로드](#)
- 텍사스 인스트루먼트, [SysConfig에 대해 자세히 알아보기](#)
- 텍사스 인스트루먼트, [MSPM0L LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0G LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0 아카데미](#)

E2E

TI의 E2E™ 지원 포럼 토론을 보고 새로운 스레드를 게시하여 설계에서 MSPM0 장치를 활용하는 데 필요한 기술 지원을 받으세요.

주파수 카운터: 튠 감지

설명

그림 80의 이 서브시스템 예제에서는 간단한 주파수 감지를 구현하기 위해 MSPM0L 및 MSPM0G 장치 제품군에서 내부 비교기 및 타이머를 설정하는 방법을 보여줍니다. 캡처 기간은 다양한 주파수 범위를 허용하도록 구성할 수 있습니다.

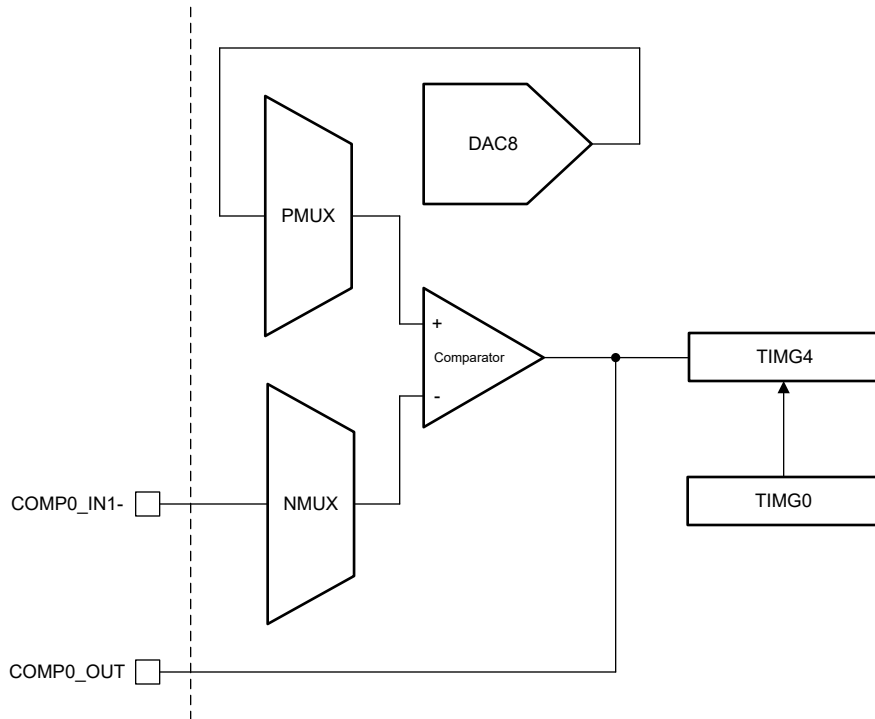


그림 80. 서브시스템 기능 블록 다이어그램

필요한 주변 기기

이 애플리케이션에는 통합 비교기 1개 및 타이머 모듈 2개가 필요합니다.

표 50. 필요한 주변 기기

하위 블록 기능	사용되는 주변 기기	참고
아날로그-디지털 신호 변환	(1 x) 비교기	코드에서 COMP_0_INST라고 부름
디지털 신호 캡처	(2 x) 타이머	코드에서 COMPARE_0_INST 및 PERIOD_TIMER_INST라고 부름

호환 가능 장치

표 50의 요구 사항을 바탕으로 호환 가능 장치가 표 51에 해당하는 EVM과 함께 열거되어 있습니다. 표 50의 요구 사항이 충족되면 다른 MSPM0 장치 및 해당하는 EVM을 사용할 수 있습니다.

표 51. 호환 가능 장치

호환 가능 장치	EVM
MSPM0Lxxxx	LP-MSPM0L1306
MSPM0Gxxxx	LP-MSPM0G3507

설계 단계

1. SysConfig에서 비교기 주변 기기 인스턴스, 타이머 - 비교 인스턴스, 타이머 인스턴스 및 원하는 장치 핀에 대한 핀 출력을 설정합니다.
2. SysConfig에서 비교기 전압을 설정합니다.
3. SysConfig에서 타이머 - 비교 클럭 속도를 설정합니다. 기본값은 4MHz입니다.
4. SysConfig에서 타이머 클럭 속도를 설정합니다. 기본값은 32,768Hz입니다.
5. 원하는 주파수 범위를 정의합니다.
6. 원하는 주파수 범위에 따라 캡처 기간을 정의합니다.
7. SysConfig에서 타이머 - 감지할 에지 수 비교를 설정합니다. 또한 코드에서 MAX_COMPARE_COUNT를 정의합니다. (옵션)

설계 고려 사항

1. **캡처 기간:** 캡처 기간의 길이는 측정할 수 있는 주파수 범위에 영향을 줍니다. 기간이 길어지면 더 느린 주파수를 캡처할 수 있습니다.
2. **클럭 속도:** 이 예제가 올바르게 작동하기 위해서는 주파수를 정확하게 측정하도록 해 주는 클럭 속도를 선택하는 것이 중요합니다.

소프트웨어 흐름도

그림 81에서는 그림 80에 대한 Main() 및 타이머 ISR의 코드 흐름 다이어그램을 보여줍니다.

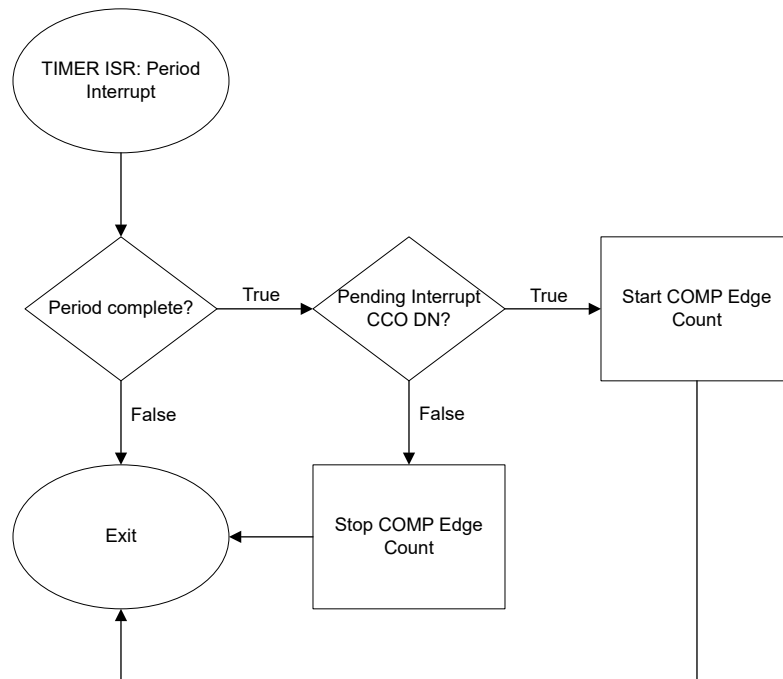
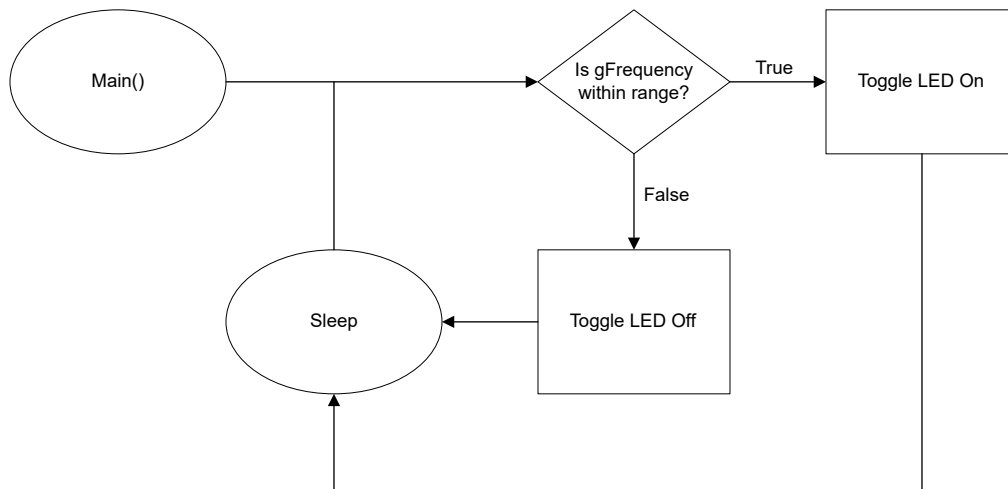


그림 81. 메인 루프 및 TIMER ISR의 소프트웨어 흐름 다이어그램

장치 구성

이 애플리케이션은 TI 시스템 구성 툴(SysConfig)의 그래픽 인터페이스를 사용하여 비교기 및 타이머 모듈 2개의 구성 코드를 생성합니다. 그래픽 인터페이스를 사용하여 장치 주변 기기를 구성하면 애플리케이션 프로토타입 제작 프로세스를 간소화할 수 있습니다.

애플리케이션 코드

타이머 및 원하는 주파수 범위에서 사용되는 특정 값을 변경하려면 다음 코드 블록에 설명된 대로 문서의 시작 부분에서 #defines를 수정합니다.

```

/* Based on required specifications, vary the value
 * between PERIOD_10ms, PERIOD_20ms, and PERIOD_50ms
 * to achieve desired frequency range.
 *
 * RANGES:
 * 10 ms: 100 Hz - 1 MHz
 * 20 ms: 50 Hz - 1 MHz
 * 50 ms: 20 Hz - 1 MHz
 *
 * Please reference [file name] for percent error
 */
#define CAPTURE_PERIOD (PERIOD_20ms) /* CHANGE THIS VARIABLE VALUE */

/* Set the desired frequency range
 * NOTE: see [file name] to ensure proper capture period is set
 * for desired frequency range
 */
#define LOWERBOUND (2000)
#define UPPERBOUND (10000)

/* The maximum amount of rising edge the Timer Compare
 * will read from the COMP. Used as a limit rather than
 * an actual fix value of counts
 */
#define MAX_COMPARE_COUNT 65000

```

추가 리소스

- 텍사스 인스트루먼트, [MSPM0 SDK 다운로드](#)
- 텍사스 인스트루먼트, [SysConfig에 대해 자세히 알아보기](#)
- 텍사스 인스트루먼트, [MSPM0L LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0G LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0 타이머 아카데미](#)
- 텍사스 인스트루먼트, [MSPM0 비교기 아카데미](#)

E2E

TI의 [E2E™](#) 지원 포럼 토론을 보고 새로운 스레드를 게시하여 설계에서 MSPM0 장치를 활용하는 데 필요한 기술 지원을 받으세요.

PWM을 사용하는 LED 드라이버

설명

PWM 듀티 사이클은 LED 밝기와 직접적인 연관이 있습니다. LED를 애플리케이션의 표시등 또는 광원으로 사용하는 경우 PWM 신호를 사용하여 LED 밝기 및 소비 전력을 구동할 수 있습니다. MSPM0의 타이머 모듈을 사용하여 다양한 주파수 및 듀티 사이클로 PWM 신호를 생성할 수 있습니다. 이 예제 코드는 하트비트 방식으로 LED를 어둡게 하고 밝게 하여 LED를 구동하는 데 사용할 수 있는 PWM 듀티 사이클의 전체 범위를 표시합니다.

그림 82에서는 이 예제에서 사용된 주변 기기의 기능 블록 다이어그램을 보여줍니다.

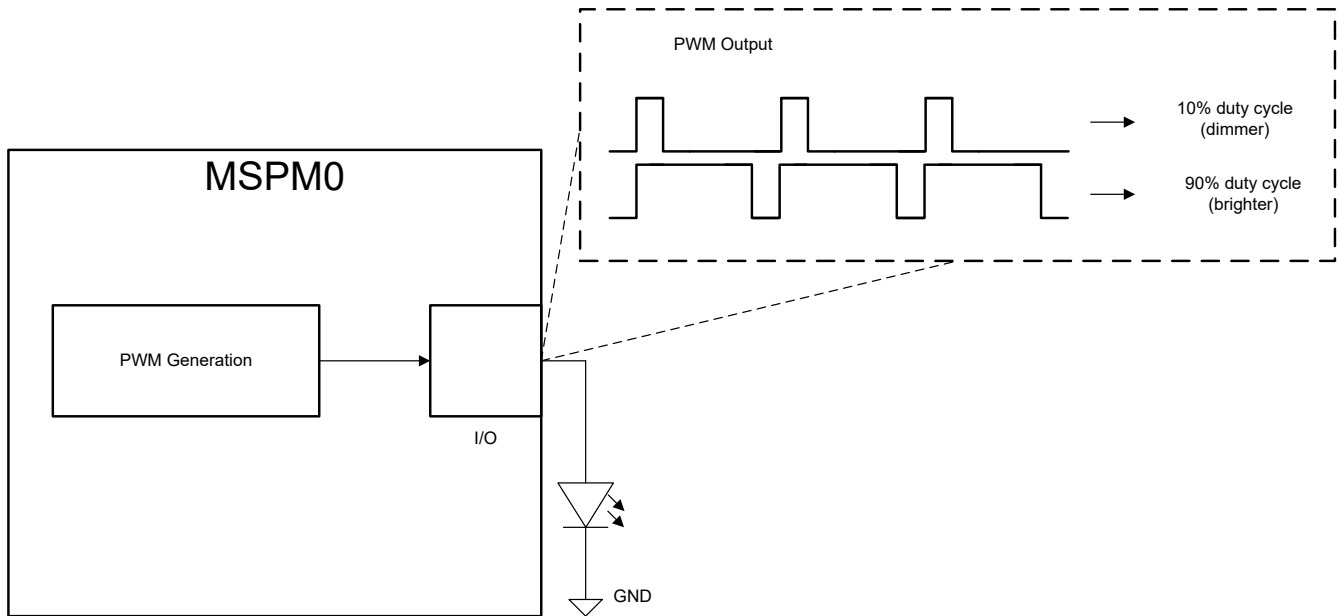


그림 82. 서브시스템 기능 블록 다이어그램

필요한 주변 기기

이 애플리케이션에는 타이머 1개, 장치 핀 1개 및 온보드 LED 1개가 필요합니다.

표 52.

하위 블록 기능	사용되는 주변 기기	참고
PWM 생성	(1x) 타이머 G	코드에서 PWM_0_INST라고 부름
IOMUX 하위 블록	1핀	(1x) PWM 출력

호환 가능 장치

표 52의 요구 사항에 따라 이 예제는 표 53의 장치와 호환됩니다. 해당 EVM은 프로토타입 제작에 사용할 수 있습니다.

표 53.

호환 가능 장치	EVM
MSPM0Lxxx	LP-MSPM0L1306
MSPM0Gxxx	LP-MSPM0G3507

설계 단계

1. 필요한 PWM 출력 주파수 및 해상도를 결정합니다. 이 두 매개 변수는 다른 설계 매개 변수를 계산할 때의 출발점이 되며, 주파수는 외부 부품의 상태를 얼마나 빠르게 업데이트해야 하는지에 따라 결정되어야 합니다. 이 예제에서는 62Hz의 PWM 출력 주파수와 2,000비트의 PWM 해상도를 선택했습니다.
2. 타이머 클럭 주파수를 계산합니다. 다음 방정식을 사용하여 타이머 클럭 주파수를 계산할 수 있습니다. $F_{clock} = F_{pwm} \times resolution$
3. SysConfig에서 주변 기기를 구성합니다. 사용되는 타이머 인스턴스와 PWM 출력에 사용되는 장치 핀을 선택합니다. 이 예제에서는 PWM 출력(TIMG0에 연결됨)에 PA13을 사용합니다.
4. 애플리케이션 코드를 작성합니다. 이 애플리케이션의 나머지 부분은 소프트웨어에서 수행되는 PWM 듀티 사이클을 변경하는 것입니다. 그림 83에서 애플리케이션의 개요를 확인하거나 코드를 직접 살펴봅니다.

설계 고려 사항

1. 최대 출력 주파수: 기본적으로 최대 PWM 출력 주파수는 IO 속도와 선택한 클럭 소스 주파수 모두에 의해 제한됩니다. 그러나 듀티 사이클 해상도는 최대 출력 주파수에도 영향을 미칩니다. 해상도를 높이려면 타이머 수가 더 많아야 하므로 출력 기간이 늘어납니다.
2. 파이프라인: 이 애플리케이션에서 선택한 PWM 타이머는 타이머 비교 값 파이프라인을 지원합니다. 파이프라인을 사용하면 애플리케이션에서 출력에서 글리치를 유발하지 않고 타이머 비교 값에 대한 업데이트를 예약할 수 있습니다.

소프트웨어 흐름도

그림 83에서는 PWM 출력의 듀티 사이클을 변경하기 위해 애플리케이션에서 수행하는 작업을 보여줍니다.

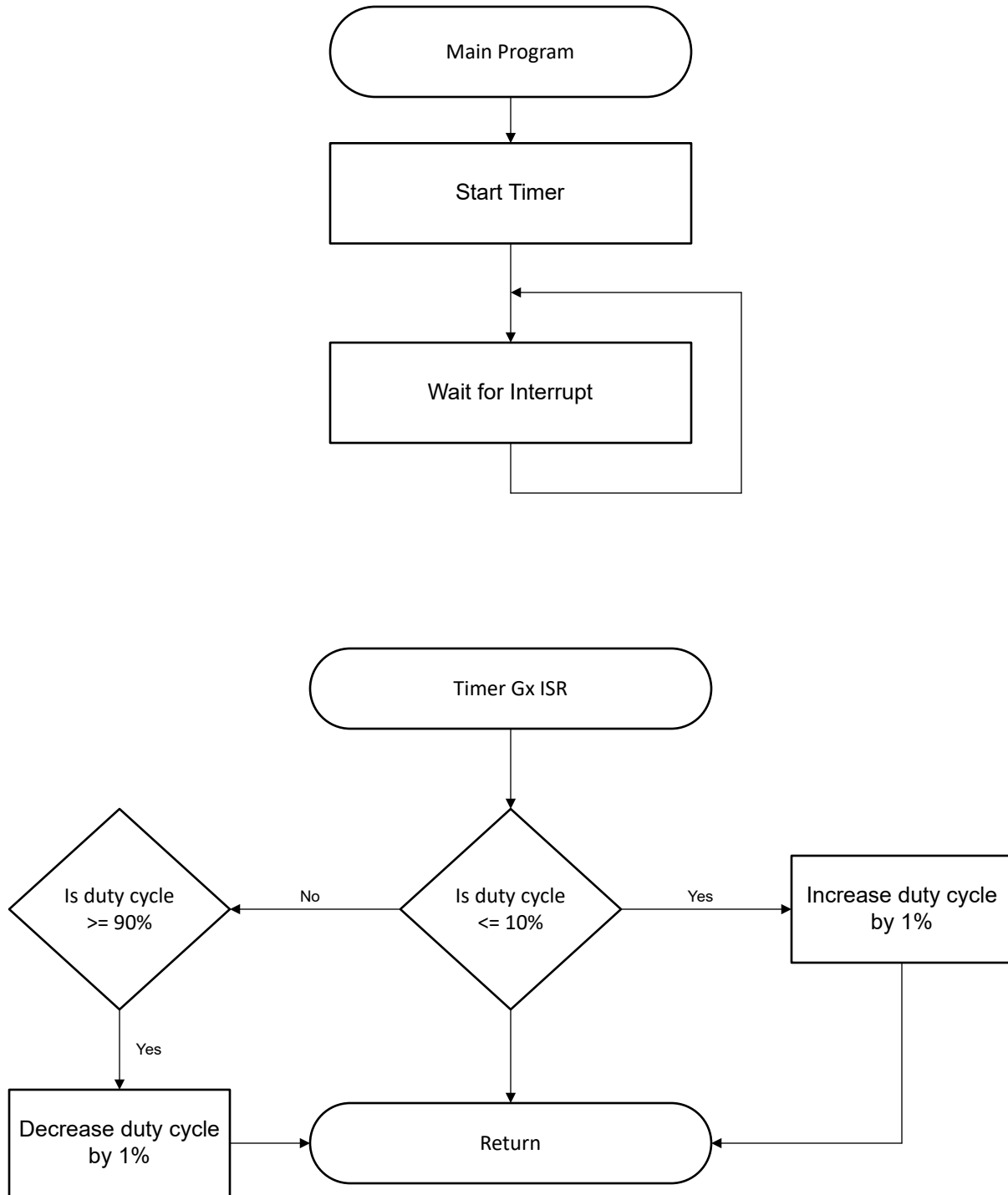


그림 83. 애플리케이션 소프트웨어 흐름도

애플리케이션 코드

애플리케이션 코드에서 타이머가 인터럽트를 트리거할 때마다 PWM 듀티 사이클이 90%에 도달할 때까지 1%씩 증가한 후 듀티 사이클이 10%에 도달할 때까지 1%씩 감소하여 하트비트 효과를 생성합니다. 이 애플리케이션 PWM 출력의 해상도는 2,000비트이므로 *pwm_count* 변수를 20씩 늘리거나 줄이면 듀티 사이클이 1%씩 변경됩니다. 애플리케이션 요구 사항에 따라 다른 확장이 필요할 수 있습니다.

```
void PWM_0_INST_IRQHandler(void){
    switch (DL_TimerG_getPendingInterrupt(PWM_0_INST)){
        case DL_TIMER_IIDX_LOAD:
            if (dc <= 10){mode = 1;} // if reached lowest dc (10%), increase dc
            else if (dc >= 90){mode = 0;} // if reached highest dc (90%), decrease dc
            if (mode){pwm_count -= 20; dc += 1;} // up
            if (!mode){pwm_count += 20; dc -= 1;} // down
            DL_TimerG_setCaptureCompareValue(PWM_0_INST, pwm_count, DL_TIMER_CC_1_INDEX); // update ccr1
value
            break;
        default:
            break;
    }
}
```

결과

추가 리소스

- [MSPM0 SDK 다운로드](#)
- [SysConfig에 대해 자세히 알아보기](#)
- [MSPM0L LaunchPad 개발 키트](#)
- [MSPM0G LaunchPad 개발 키트](#)
- [MSPM0 타이머 PWM 아카데미](#)

전원 시퀀서

설명

전원 시퀀싱 예제에서는 여러 간격으로 한 번의 시동으로 여러 레일을 켜는 방법을 보여줍니다. 이 예방 조치는 시동 중에 전원 스파이크, 버스 경합, 래치 업 오류 및 기타 문제를 일으키는 장치 손상을 방지하는 데 도움이 됩니다. MSPM0에서는 타이머를 하나만 사용하여 각 레일에 대해 서로 다른 간격을 설정할 수 있습니다.

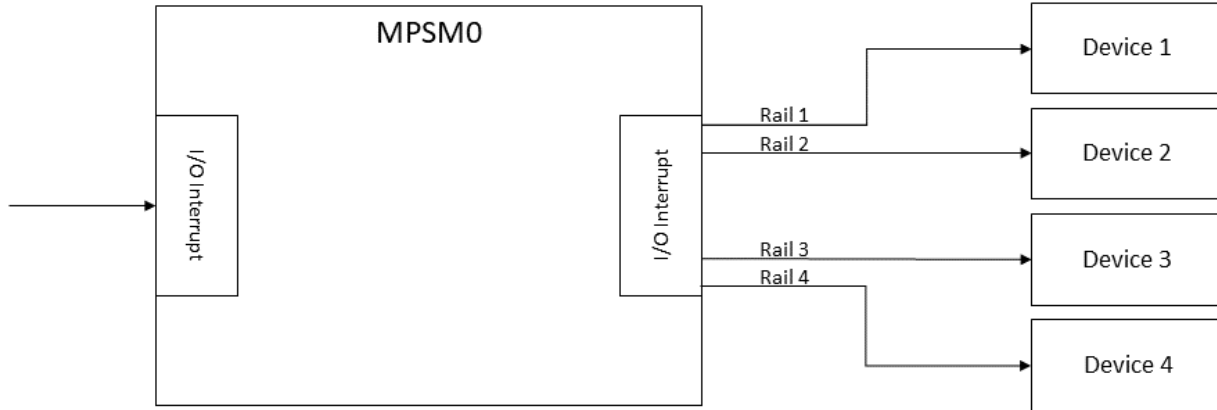


그림 84. 서브시스템 기능 블록 다이어그램

필요한 주변 기기

이 애플리케이션에는 타이머 1개, 출력 핀 4개 및 입력 핀 1개가 필요합니다. 출력 핀 수는 애플리케이션의 요구 사항에 따라 달라질 수 있습니다.

표 54. 필요한 주변 기기

하위 서브블록 기능	사용되는 주변 기기	참고
인터럽트 트리거	1핀	트리거를 위한 신호 입력
출력 신호	4핀	시퀀싱을 위한 출력 신호
시퀀스 생성	타이머 G 1개	코드에서 TIME_SEQUENCE라고 부름

호환 가능 장치

표 54의 요구 사항에 따라 이 예제는 표 55의 장치와 호환됩니다. 해당 EVM은 프로토타입 제작에 사용할 수 있습니다.

표 55. 호환 가능 장치

호환 가능 장치	EVM
MSPM0Cxxx	LP-MSPM0C1104
MSPM0Lxxx	LP-MSPM0L1306
MSPM0Gxxx	LP-MSPM0G3507

설계 단계

1. 시동 시 각 레일 간에 원하는 시간 간격을 결정합니다. 설정된 시간 간격은 시작점이 **아닌** 레일에서 레일까지 순차적으로 계산됩니다. 간격 계산에 대한 지침은 [설계 고려 사항](#)을 참조하세요.
2. 섀다운 시 각 레일 간에 원하는 시간 간격을 결정합니다. 설정된 시간 간격은 시작점이 **아닌** 레일에서 레일까지 순차적으로 계산됩니다. 또는 출력을 동시에 섀다운할 수 있습니다.
3. SysConfig에서 주변 기기를 구성합니다. 타이머를 선택하고, 선택한 주파수로 구성된 후 인터럽트를 제로 이벤트로 설정합니다. 입력 인터럽트를 상승 및 하강 에지로 설정합니다. 입력 전압 및 출력 레일을 위한 핀을 선택합니다.
4. 애플리케이션 코드에서 원하는 시간 간격을 수정합니다. 시간 간격은 .c 파일의 맨 위에 있습니다.

설계 고려 사항

1. **다중 레일:** 이 애플리케이션의 레일 수를 늘리거나 줄일 수 있습니다. 레일 수를 구현하기 위해서는 약간의 편집만 필요합니다.
 - a. 간격 시간 시퀀스의 배열 크기는 선택한 레일 수와 일치해야 합니다. 참조되는 두 배열은 gTimerUp[] 및 gTimerDown[]입니다.
 - b. 레일을 더하거나 빼면 각 GPIO 출력에 대해 pinToggle 함수를 편집해야 합니다.
2. **시퀀스 순서:** 작성된 애플리케이션에는 시퀀스에 대한 특정 순서가 있습니다. 트리거되는 레일의 순서를 변경하려면 pinToggle 함수에서 찾을 수 있는 GPIO_OUT_PIN_#_PIN의 #을 if 명령문의 원하는 순서로 변경합니다.
3. **클럭 설정:** 최대 간격 해상도는 타이머의 주파수에 따라 다릅니다. 타이머 클럭 설정은 시스템 클럭 설정에 따라 조정해야 합니다. 타이머를 얼마나 빨리 클로킹하는지와 레일 간 시간 해상도 간에는 직접적인 관계가 있습니다. 타이머를 빠르게 클로킹할수록 그 사이의 해상도는 높아지지만, 입력 클럭 주파수가 증가하면 레일 간 총 가능한 시간이 감소합니다.
4. **간격 계산:** SysConfig는 MSPM0 제품군의 설정 주파수를 기반으로 기간 범위 및 해상도를 제공합니다. 예제 코드에서 해상도는 7.81ms이고, 클럭 주파수는 128Hz로 설정되어 있습니다. 원하는 간격의 기간은 원하는 시간을 해상도로 나눠 계산할 수 있습니다.
5. **포트 설정:** MSPM0 제품군의 일부 장치는 여러 포트를 제공합니다. 포트가 두 개 이상 사용 중인 경우 여러 포트를 사용하도록 애플리케이션의 GPIO 코드 부분을 수정해야 합니다.
6. **출력 핀에 외부 장치 연결:** 외부 장치는 이러한 유형의 애플리케이션에서 다양한 방식으로 제어할 수 있습니다. 다음 목록에서는 세 가지 일반적인 방법을 설명합니다.
 - a. **활성화 핀:** 출력에 대해 추가 작업이 필요하지 않습니다.
 - b. **직접 전원 공급:** 외부 장치가 출력에 의해 전원을 공급받는 경우 수정이 필요하고 장치 데이터 시트에 있는 출력 전류 제한에 관한 사항을 고려해야 합니다.
 - c. **외부 전원 회로:** 외부 GPAMP와 같은 다른 장치의 전원을 공급하기 위해 외부 회로가 필요한 경우 출력은 [6.a](#)의 활성화 핀 경우와 같습니다. 외부 회로는 각 시스템에 따라 다르며 이 문서의 범위를 벗어납니다.

소프트웨어 흐름도

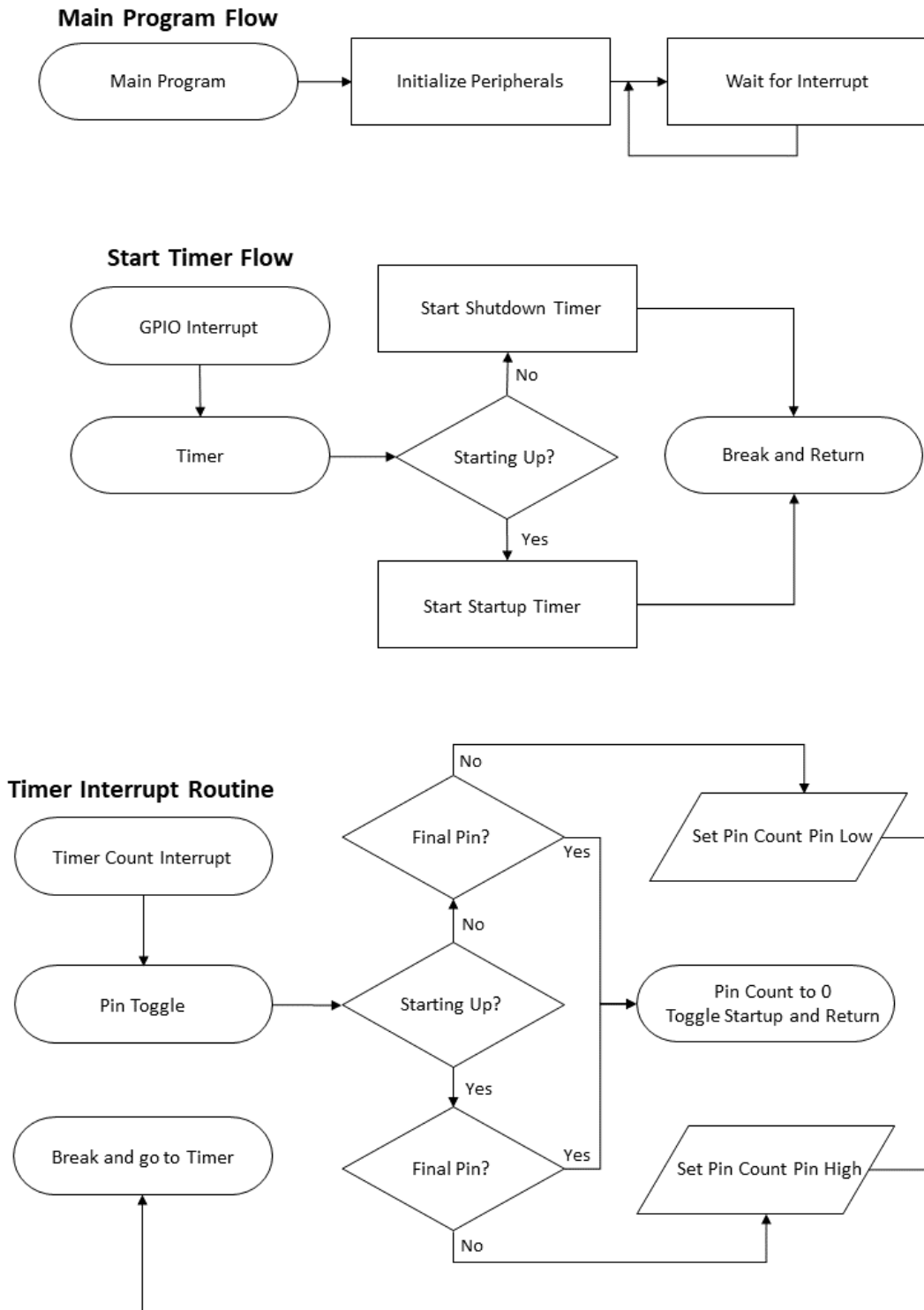


그림 85. 애플리케이션 소프트웨어 흐름도

설계 결과

그림 86에서는 코드 예제 실행의 로직 그래프 결과를 보여줍니다. 마지막에는 핀도 순서대로 턴오프되는 것으로 가정합니다.

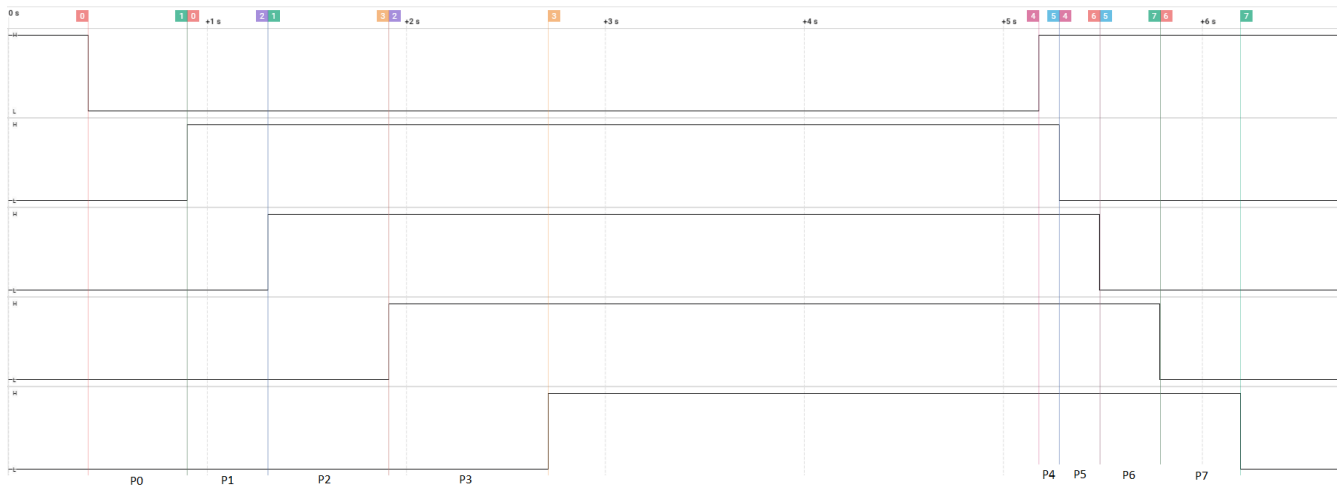


그림 86. 시퀀스 결과 그래프

- P0: 498.6ms(2.01Hz)
- P1: 404.72ms(2.47Hz)
- P2: 607.12ms(1.65Hz)
- P3: 801.68ms(1.25Hz)
- P4: 102.28ms(9.78Hz)
- P5: 202.36ms(4.94Hz)
- P6: 303.56ms(3.29Hz)
- P7: 404.72ms(2.47Hz)

참조

- 텍사스 인스트루먼트, [MSPM0 SDK 다운로드](#)
- 텍사스 인스트루먼트, [SysConfig에 대해 자세히 알아보기](#)
- 텍사스 인스트루먼트, [MSPM0L LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0G LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0 타이머 아카데미](#)

E2E

TI의 [E2E™](#) 지원 포럼 토론을 보고 새로운 스레드를 게시하여 설계에서 MSPM0 장치를 활용하는 데 필요한 기술 지원을 받으세요.

PWM DAC

설명

PWM DAC 서브시스템 예제는 MSPM0 타이머 및 간단한 RC 필터를 사용하여 PWM DAC를 생성하는 방법을 보여줍니다. 예제 소프트웨어를 통해서 PWM 주파수가 31,250Hz인 10비트 DAC를 만듭니다. PWM 신호의 듀티 사이클은 필터 출력에 시누소이드 파형을 생성하기 위해 지속적으로 업데이트됩니다. MSPM0Gx50x 장치에는 12비트 DAC가 포함되어 있고, 내부 비교기에는 OPA를 통해 버퍼링할 수 있는 8비트 레퍼런스 DAC가 포함되어 있지만, PWM DAC를 사용하면 이러한 주변 기기가 없는 장치에서 아날로그 출력 전압을 생성하거나 필요할 때 추가 DAC 출력을 생성할 수 있습니다. **그림 87**에서는 단일 PWM DAC의 블록 다이어그램을 보여줍니다.

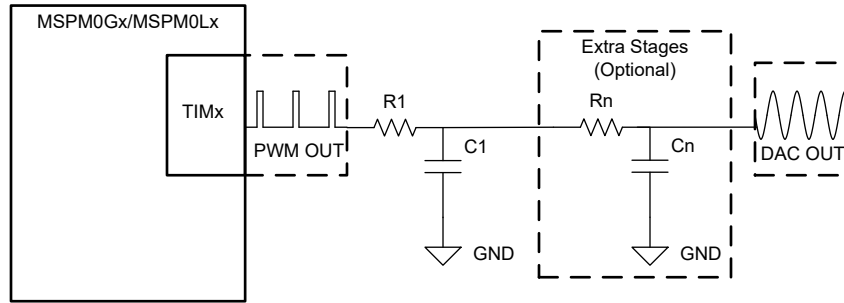


그림 87. 서브시스템 기능 블록 다이어그램

필요한 주변 기기

이 애플리케이션에는 PWM 주변 기기와 새도 캡처 비교 레지스터가 있는 TIMGx 인스턴스가 필요합니다.

표 56. 필요한 주변 기기

하위 블록 기능	사용되는 주변 기기	참고
PWM	TIMGx	예제에서는 TIMG4 새도 레지스터를 사용하여 신호 글리칭을 방지합니다

호환 가능 장치

표 56의 요구 사항을 바탕으로 호환 가능 장치가 표 57에 열거되어 있습니다. 해당 EVM은 빠른 평가를 위해 사용할 수 있습니다.

표 57. 호환 가능 장치

호환 가능 장치	EVM
MSPM0Lx	LP-MSPM0L1306
MSPM0Gx	LP-MSPM0G3507

설계 단계

1. 새도 레지스터와 인터럽트를 사용하도록 PWM을 구성합니다.
2. 원하는 DAC 해상도에 맞게 PWM 주파수를 구성합니다.
3. 듀티 사이클을 조정하는 데 필요한 샘플 수를 결정합니다. 이 서브시스템 예제에서는 배열에 저장된 128개의 샘플을 사용합니다.
4. 샘플 배열을 순환시킵니다. 이 예제에서는 연결된 ISR 동안 배열 색인을 증분시키고 새 비교 값을 로드하여 PWM의 듀티 사이클을 변경합니다.
5. PWM 출력용 저역 필터를 설계하여 아날로그 전압을 생성합니다. 이 예제에서는 단극 RC 필터를 사용합니다.

설계 고려 사항

1. **PWM 주파수:** PWM 주파수는 다음과 같이 DAC 해상도와 관련이 있습니다.

$$2^N = \frac{f_{\text{CLOCK}}}{f_{\text{PWM}}} \tag{15}$$

여기서

- f_{CLOCK} 은 타이머의 클럭 주파수입니다.
- f_{PWM} 은 출력 PWM 주파수입니다.
- N은 PWM DAC(비트 단위)의 듀티 사이클 해상도입니다.

이 서브시스템 예제에서는 32MHz 클럭 주파수 또는 16MHz 클럭 주파수를 사용하여 10비트 DAC를 생성합니다. 표 58에서는 클럭 및 PWM 주파수를 기반으로 하는 일부 PWM DAC 해상도 예제를 자세히 설명합니다.

2. **PWM 구성:** 이 애플리케이션은 에지 정렬 PWM에 대한 타이머를 구성하고, 제로 이벤트 후에 적용되는 캡처 비교 업데이트 값을 설정합니다.
3. **듀티 사이클 업데이트 동기화:** 새도 레지스터는 누락된 카운터 비교 값 업데이트를 방지하는 데 사용됩니다. 이는 MSPM0에서 적절한 타이머 인스턴스의 새도 로드 기능을 활성화하여 수행됩니다. 따라서 듀티 사이클 출력의 글리치 걱정 없이 타이머가 실행되는 동안 듀티 사이클을 업데이트할 수 있습니다.
4. **PWM 인터럽트 구성:** 여기에서 타이머는 카운트다운 모드로 구성되므로 인터럽트가 캡처 또는 다운 이벤트를 발생시키도록 구성됩니다. 다음 사이클에서 듀티 사이클을 업데이트해야 하는 경우, 캡처 비교 다운 또는 업 인터럽트를 사용하면 캡처된 값을 다음 로드 이벤트 또는 제로 이벤트 전에 업데이트할 수 있습니다. 다른 시스템 인터럽트도 사용할 수 있으며, 새도 로드 기능을 활성화하여 동기화해야 합니다.
5. **샘플 배열:** 샘플 수보다 큰 신호 또는 파형을 출력할 경우 해상도 출력이 커집니다. 샘플 값은 PWM DAC의 해상도에 맞게 형식이 지정되어야 합니다.
6. **필터 설계:** 기본 RC 필터는 일반적으로 PWM 출력을 필터링하기에 충분합니다. 필터 차단 주파수는 최소 PWM 주파수보다 낮아야 합니다.

PWM 에지를 더 잘 필터링하기 위해 더 높은 차수나 더 복잡한 필터를 사용할 수 있습니다.

표 58. PWM DAC 해상도

f_{CLOCK}	f_{PWM}	N
32MHz	125kHz	8
32MHz	31.3kHz	10
32MHz	7.8kHz	12

표 58. PWM DAC 해상도 (계속)

f_{CLOCK}	f_{PWM}	N
16MHz	62.5kHz	8
16MHz	15.6kHz	10
16MHz	3.9kHz	12

소프트웨어 흐름도

그림 88에서는 이 서브시스템 예제의 소프트웨어 흐름도와 이 예제에서 PWM DAC를 생성하는 데 사용되는 ISR의 소프트웨어 흐름을 보여줍니다.

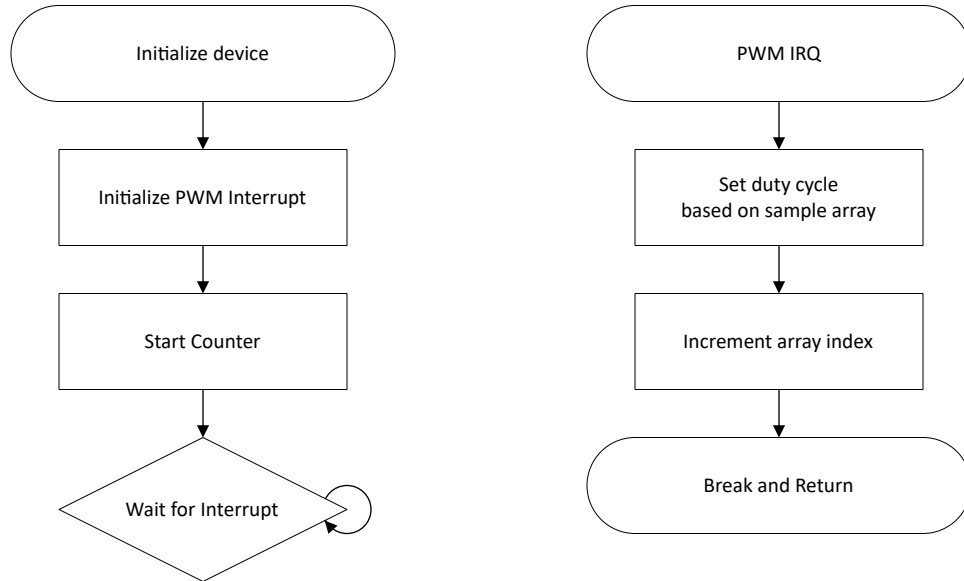


그림 88. 애플리케이션 소프트웨어 흐름도

애플리케이션 코드

이 애플리케이션은 TI 시스템 구성 툴(SysConfig) 그래픽 인터페이스를 사용하여 장치 주변 기기에 대한 구성 코드를 생성합니다. 그래픽 인터페이스를 사용하여 장치 주변 기기를 구성하면 애플리케이션 프로토타입 제작 프로세스를 간소화할 수 있습니다.

이 예제 애플리케이션 코드는 128개의 샘플 배열을 사용하여 단일 PWM 출력의 듀티 사이클을 지속적으로 변경합니다. 이를 통해 필터링 후 시누소이드가 생성됩니다. 듀티 사이클은 타이머 인터럽트 및 새도 레지스터를 통해 변경됩니다. 인터럽트는 카운터 비교 다운 이벤트에서 생성됩니다. 이 인터럽트 동안 배열 색인의 다음 카운터 비교 값이 설정되고 타이머가 0에 도달한 후 다음 TIMCLK 사이클에 로드될 준비가 됩니다. 이로 인해 애플리케이션이 최종 출력에서 글리치를 일으킬 수 있는 PWM 듀티 사이클의 변화를 놓치는 것을 방지하는 데 도움이 됩니다.

```

void PWM_0_INST_IRQHandler(void){
    switch (DL_TimerG_getPendingInterrupt(PWM_0_INST)){
        case DL_TIMERG_IIDX_CC0_DN: /* Interrupt on CC0 Down Event */
            /*Set new Duty Cycle based on sine array sample value */
            DL_TimerG_setCaptureCompareValue(PWM_0_INST, gSine128[gSineCounter%128],
                DL_TIMER_CC_0_INDEX);

            /* Increment gSineCounter value */
            gSineCounter++;

            break;
        default:
            break;
    }
}
    
```

결과

그림 89에서는 32MHz 클럭 주파수를 사용할 때 필터 출력과 비교한 PWM 디지털 출력을 보여줍니다. 상단에서는 최종 사인파 주기의 절반을 확대 표시하여 PWM 신호의 듀티 사이클 변화를 명확하게 표시합니다. 하단에서는 최종 사인파 출력을 명확하게 표시하기 위해 더 축소하여 보여줍니다.

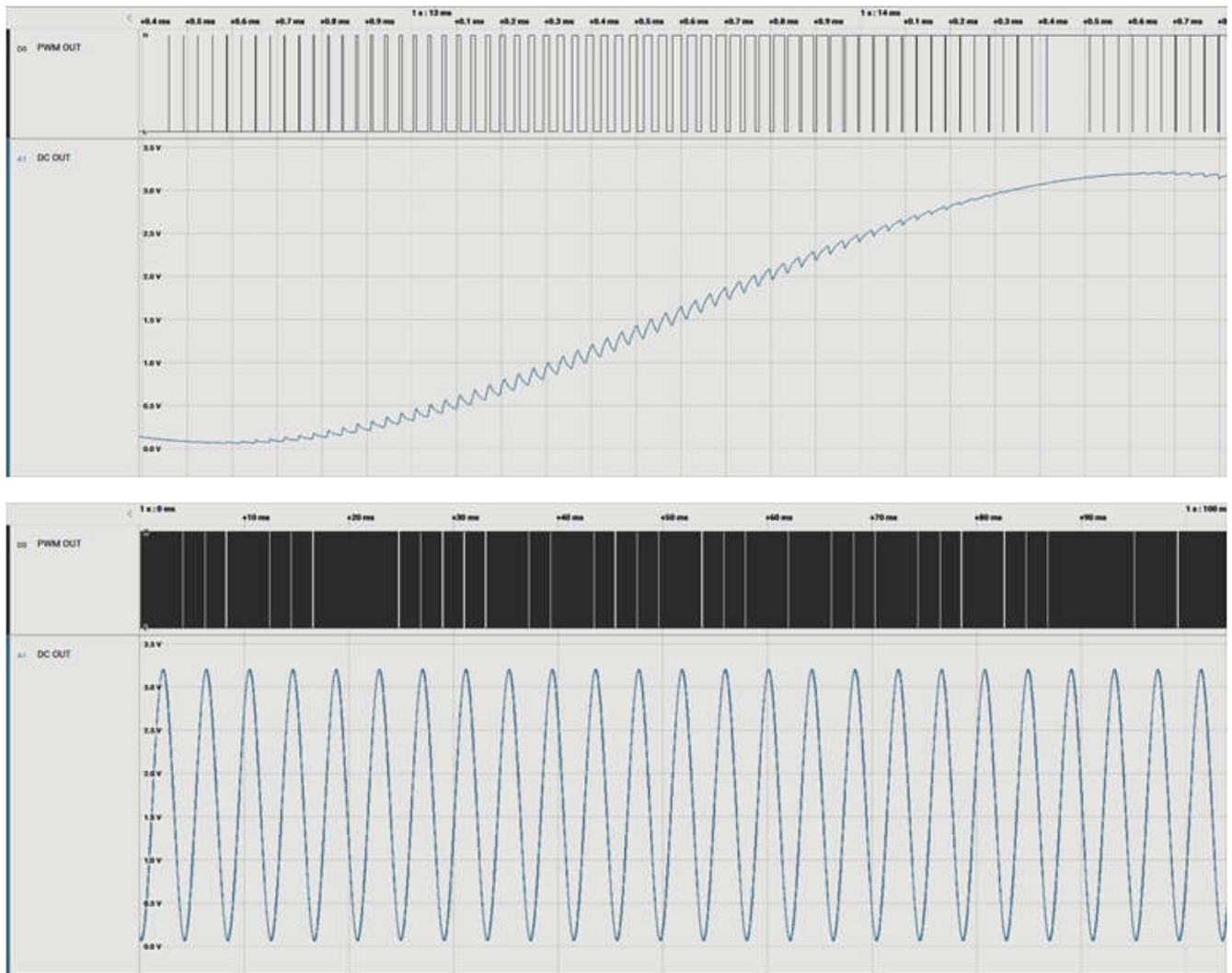


그림 89. 결과

추가 리소스

- 텍사스 인스트루먼트, [MSPM0 SDK 다운로드](#)
- 텍사스 인스트루먼트, [SysConfig에 대해 자세히 알아보기](#)
- 텍사스 인스트루먼트, [MSPM0L LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0G LaunchPad™](#)
- 텍사스 인스트루먼트, [MSPM0 아카데미](#)
- 텍사스 인스트루먼트, [MSP430 고해상도 타이머를 사용하는 PWM DAC 애플리케이션 노트](#)
- 텍사스 인스트루먼트, [PWM Timer_B를 DAC로 사용 애플리케이션 노트](#)
- 텍사스 인스트루먼트, [PWM DAC를 사용하는 음성 대역 오디오 재생](#)

E2E

TI의 [E2E™](#) 지원 포럼 토론을 보고 새로운 스레드를 게시하여 설계에서 MSPM0 장치를 활용하는 데 필요한 기술 지원을 받으세요.

중요 알림: 이 문서에 기술된 텍사스 인스트루먼트의 제품과 서비스는 TI의 판매 표준 약관에 의거하여 판매됩니다. TI 제품과 서비스에 대한 최신 정보를 완전히 숙지하신 후 제품을 주문해 주시기 바랍니다. TI는 애플리케이션 지원, 고객의 애플리케이션 또는 제품 설계, 소프트웨어 성능 또는 특허권 침해에 대해 책임을 지지 않습니다. 다른 모든 회사의 제품 또는 서비스에 관한 정보 공개는 TI가 승인, 보증 또는 동의한 것으로 간주되지 않습니다.

LaunchPad™, E2E™, and BoosterPack™ are trademarks of Texas Instruments.
모든 상표는 해당 소유권자의 자산입니다.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to [TI's Terms of Sale](#) or other applicable terms available either on [ti.com](https://www.ti.com) or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265

Copyright © 2025, Texas Instruments Incorporated