**18.1 Introduction**

\*이 장에서는 CAN 컨트롤러의 아키텍처와 기능, 그리고 설계시 다양한 레지스터의 기능에 대해 설명

\*PS에는 2 개의 거의 동일한 CAN 컨트롤러가 독립적으로 작동

\*CAN 프로토콜을 정의하는 것은이 문서의 범위를 벗어나며 사양에 대한 지식이 있다고 가정한다.

**18.1.1 Features 특징**

• ISO 11898 -1, CAN 2.0A 및 CAN 2.0B 표준과 호환 가능  
• 표준 (11 비트 식별자) 및 확장 (29 비트 식별자) 프레임  
• 최대 1Mb / s의 비트 전송률  
• 64개의메세지를 갖는 송신 메시지 FIFO (TxFIFO)  
• 하나의 우선 순위가 높은 전송 버퍼 (TxHPB)를 통해 우선 순위를 전송합니다.  
• TxFIFO 및 RxFIFO에 대한 워터 마크(경계선) 인터럽트  
정상 모드에서의 오류 또는 중재 손실시 자동 재전송  
• 64개의 메시지를 갖는 수신 메시지 FIFO (RxFIFO)  
인 에이블, 마스크 및 ID가있는 4 개의 Rx 수용 필터  
• 특수 애플리케이션을위한 루프백 및 스눕 모드  
• 마스크 가능한 오류 및 상태 인터럽트  
수신 메시지를위한 16 비트 타임 스탬핑  
• 읽기 가능한 Rx / Tx 오류 카운터

**18.1.2 System Viewpoint 시스템 관점**

그림 18-1은 CAN 컨트롤러의 시스템 관점을 보여줌

**18.1.3 Block Diagram 블록 다이어그램**

그림 18-2는 CAN 코어의 상위 아키텍처를 보여줍니다. 서브 모듈은 후속 절에서 설명

**\*Configuration Registers 구성레지스터**CAN 컨트롤러 구성 레지스터는 구성 레지스터를 정의한다. 이 모듈은 APB 인터페이스를 통해 레지스터에 대한 읽기 및 쓰기 액세스를 허용한다. CAN 컨트롤러 레지스터의 개요는 18.3.8 레지스터 개요에 나와 있다  
**\*Transmit and Receive Messages 메시지 전송 및 수신**  
FIFO 구조를 통해 전송 (TxFIFO) 및 수신 (RxFIFO) 메시지를위한 별도의 저장 버퍼가 존재한다. 각 버퍼는 최대 64 개의 메시지를 저장할 수 있다 . 메시지 가 TxFIFO에 기록되면 CAN 버스를 통해 전송하는 데 2 ​​\* (Tx 드라이버 지연 + 전파 지연 + Rx 드라이버 지연)의 총 지연이 발생한다

**\*Tx High Priority Buffer Tx 우선 순위가 높은 버퍼**  
각 컨트롤러에는 전송 우선 순위가 높은 버퍼 (TxHPB)가있어 하나의 전송 메시지에 대한 저장소를 제공합니다. 이 버퍼에 쓰여진 메시지는 최대 전송 우선 순위를가집니다. TxFIFO에서 메시지를 선점하여 현재 전송이 완료된 직후 전송을 위해 큐에 대기합니다.

**\*Acceptance Filters 수용필터**  
수용 필터는 사용자정의 수용 마스크 및 ID 레지스터(누가쓴거지구분해주는레지스터)를 사용하여 들어오는 메시지를 정렬하여 RxFIFO에 메시지를 저장할지 여부를 결정하거나 메시지를 확인하고 무시합니다. 승인 필터를 통해 전달 된 메시지는 RxFIFO에 저장됩니다.

**18.1.4 Notices**

**\*Restrictions 제한사항**  
두 컨트롤러를 위한 단일PS(processor system)클럭생성기가 있다. 내부 클럭이 사용될 때,동일한 클럭 주파수가 되지만, 각 컨트롤러에 대한 클럭은 slcr(system level control register 슬기부분 슬기한테물어보기)레지스터를 사용하여 개별적으로 활성화 될 수있다

<>안의 섹션 참고

< Quad-SPI클럭은 Quad-SPI보드속도분배기를 사용하여 적어도 2분의1로 나뉘어져있다. 12.4.1절참조.>

<마스터모드에서 SPI 클럭은 SPI보드속도분배기를 사용하여 4분의1로 나뉘어져 있다. 17.4.2절참조.>

또한, MIO핀을 통해 외부소스로부터 한쪽 또는 양쪽 제어장치가 클록 될 수 있다(18.4.1 절 참조).

권장 사항 :모든 클럭킹소스에 대해 can.BRPR 레지스터를 최대2이상으로 설정하고 프리스케일러 값을 최소3 이상으로 설정해야함 .

**18.2 Functional Description 기능설명**

\*각 컨트롤러는 독립적으로 구성 및 제어된다

\*두 개의 CAN 컨트롤러 (CANx, 여기서 x = 0 또는 1)가 있다

\*CAN레지스터에 대한 레지스터 머리글은 ‘can’(예: can,MSR 레지스터)이다

**18.2.1 Controller Modes 제어모드**

**\*** CAN컨트롤러는 다음과 같은 작동모드를 지원

• Configuration

• Normal

• Sleep

• Loop Back

• Snoop mode

**\*Configuration Mode 구성모드**

CAN 컨트롤러는 동작 모드에 관계없이 다음 동작 중 하나가 수행 될 때 구성 모드로 들어간다

• SRR 레지스터의 CEN 비트에 0을 쓴다.  
• SRR 레지스터의 SRST 비트에 1을 쓴다. 코어는 소프트웨어 재설정 직후 구성 모드로 들어간다.  
• 리셋 입력에서 0을 구동. 코어는 리셋이 0인 한 계속 재설정된다. 코어는 리셋이 1로 무효화 된 후 구성 모드로 들어간다

**\*Normal Mode 정상모드**

정상모드는 Bosch 및 IEEE 사양에 정의 된대로 Tx 및 Rx I / O 신호에서 메시지를 전송 및 수신한다

**\*Sleep Mode 절전모드**

\*절전모드는 유휴(?)시간동안 적은양의 전력을 절약하는데 사용

\*절전모드에 있을 때 컨트롤러는 정상모드 또는 구성모드로 전환할수 있다

<절전모드일 때>

• 다른 노드가 메시지를 전송하면 컨트롤러는 메시지를 수신하고 절전 모드를 종료

• 새로운 Tx 요청이있는 경우 하드웨어는 컨트롤러를 일반 모드로 전환하고 컨트롤러가 요청을 처리  
• 컨트롤러가 절전 모드로 들어가면 인터럽트가 생성 될 수 있다  
• 컨트롤러가 깨어 나면 인터럽트가 생성 될 수 있다.

\*슬립모드는 TxFIFO 또는 TxHPB중 한쪽에서 CAN버스활동 또는 요청이있을때 하드웨어에 의해 종료

\*컨트롤러가 슬립모드를 종료하면 하드웨어에서 can.MSR[SLEEP]을 0으로 설정하고 인터럽트를 생성할 수 있다

\*CAN 컨트롤러는 MSR의 LBACK 비트가 0, MSR의 SLEEP 비트가 1, SRR의 CEN 비트가 1 일 때 구성 모드에서 슬립 모드로 전환

\*CAN 컨트롤러는 TX FIFO 또는 TX High Priority Buffer에서 보류중인 전송 요청이 없는 경우에만 Sleep 모드로 들어간다

\*CAN 컨트롤러는 SLEEP 비트가 1이고 CAN 버스가 유휴 상태이고 TX FIFO 또는 TX 높은 우선 순위 버퍼로부터 보류중인 전송 요청이 없는 경우에만 정상 모드에서 슬립 모드로 전환된다

\*다른 노드가 메시지를 전송하면 CAN 컨트롤러는 전송 된 메시지를 수신하고 절전 모드를 종료한다 컨트롤러가 슬립모드에 있을 때, TX FIFO 또는 TX 고High Priority Buffer로부터 새로운 전송 요청이 있으면, 이러한 요청은 처리되고 CAN 컨트롤러는 슬립 모드에서 빠져 나온다 인터럽트는 CAN 컨트롤러가 슬립 모드로 들어가거나 슬립 모드에서 깨어날 때 생성된다. 슬립 모드에서 CAN 컨트롤러는 구성모드 나 일반모드로 들어갈 수 있다

**Loop Back Mode (Diagnostics) 루프백모드(진단)**

\*루프백모드는 진단목적으로 사용된다 루프백모드에 있을 때 컨트롤러는 반드시 구성모드로 들어가거나 재설정을 실행하도록 프로그래밍 해야한다.

<루프백모드>

• 컨트롤러는 열성 비트 스트림을 CAN\_TX 버스 신호로 전송한다  
• Tx 메시지는 내부적으로 Rx 회선으로 되돌아와 확인 응답한다  
• Tx 메시지는 CAN\_TX 버스 신호로 전송되지 않는다  
• 컨트롤러는 전송하는 모든 메시지를 수신한다  
• 컨트롤러는 다른 CAN 노드가 전송 한 메시지를 수신하지 않는다

**Snoop Mode (Diagnostics) 스눕모드(진단). 검시모드**

\*스눕 모드는 진단 목적으로 사용된다

\*스눕모드에 있을때 컨트롤러는 꼭 구성모드로 들어가거나 재설정상태로 유지되도록 프로그램해야한다

<스눕모드>

• 컨트롤러는 열성 비트 스트림을 CAN 버스로 전송합니다.  
• 컨트롤러가 일반 버스 통신에 참여하지 않습니다.  
• 컨트롤러는 다른 CAN 노드가 전송 한 메시지를 수신합니다.  
• 소프트웨어는 허용 필터를 프로그래밍하여 기준을 동적으로 활성화 / 비활성화하고 변경합니다. 에러 카운터는 비활성화되고 0으로 클리어됩니다. 에러 카운터 레지스터에 대한 읽기는 0을 반환합니다

**Mode Transitions 모드전환**

\*지원되는 모드전환이 그림18-3에 나옴

\*첫번째 컨트롤러는 주로 리셋, CEN 비트, MSR 레지스터 설정하고 변화한다 하드웨어 웨이크업 메커니즘에 의해 제어된다

\*구성모드에서 일반모드로 들어가는방법>

•CAN.MSR [LBACK, SNOOP, SLEEP] = 0 으로 클리어  
•CAN.SRR [CEN] = 1로 셋팅 정상 모드에서 슬립모드로 들어간다(인터럽트 생성) :  
•CAN.MSR [SLEEP] = 1로 셋팅 컨트롤러가 절전 모드를 종료하도록하는 이벤트 (인터럽트가 생성됨) :  
• Rx 신호 동작 (하드웨어 CAN.MSR [SLEEP] = 0 셋팅)

•TxFIFO 또는 TxHPB 동작 (하드웨어 CAN.MSR [SLEEP] = 0 셋팅)

• 소프트웨어는 CAN.MSR [SLEEP]에 0을 씁니다.

**Mode Settings 모드세팅**

표 18-1은 CAN 컨트롤러 작동 모드 및 해당 제어 및 상태비트를 정의

**18.2.2 Message Format 메시지 형식**

\*동일한 메시지 형식이 RxFIFO 및 Tx (FIFO 및 HPB)에 사용

\*각 메시지는 4 개의 단어 (16 바이트)를 포함

\*소프트웨어는 메시지의 실제 데이터 바이트 수와 유효한 필드 수에 관계없이 네 단어 모두를 읽고 쓸 수 있어야한다

\*메시지 단어, 필드 및 구조는 표 18-2에 나와 있다 각 필드의 세부 사항은 표 18-3에 나와 있다

**Bit Field Details 비트필드세부정보**

<Writes>

비트 필드 또는 데이터 바이트가 필요하지 않으면 0을 씁니다. 소프트웨어는 사용되지 않는 기능에 대해서는 표 18-3에 표시된 디폴트값(기본값)을 작성해야합니다

< Reads>

데이터는 바이트 0에서 시작하여 DLC의 카운트 수만큼 계속됩니다. 소프트웨어는 두 데이터 단어를 모두 읽어야하지만 유효한 바이트 만 DLC에 의해 결정됩니다.  
표 18-3은 식별자 워드 비트, DLC 워드 비트, 데이터 워드 1 및 데이터 워드 2 비트에 대한 비트 설명을 제공합니다.

**18.2.3 Message Buffering 메시지 버퍼링**

**Rx Messages 수신메세지**

\*RxFIFO는 수신되어 선택적으로 필터링 된 최대 64 개의 Rx CAN 메시지를 저장할 수 있다

\* 수용 필터 중 하나를 통과하는 Rx메시지는 RxFIFO에 저장된다 수용필터가 선택되지않으면 수신된 모든메세지가 RxFIFO에 저장된다

\* 성공적으로 저장된 Rx 메시지가 타임 스탬프에 각각 추가된다

\* Rx 메시지를 타임 스탬프하는 규칙은 다음과 같습니다

•카운터가 롤오버된다 롤오버 조건이 발생했음을 나타내는 상태 비트는 없다

• Rx 메시지를 성공적으로 수집 할 때 포함 된 타임 스탬프입니다. 카운터의 샘플링은 EOF의 마지막 비트에서 수행됩니다.  
• 카운터는 CEN = 0이거나 can.TCR 레지스터에 1을 쓰는 소프트웨어에 의해 지워집니다

소프트웨어는 메시지의 데이터 바이트 수에 관계없이 RxFIFO에서 Rx 메시지의 4 개 레지스터를 모두 읽어야합니다 첫 번째 단어는 RXFIFO\_ID 레지스터를 사용하여 읽히고 수신 된 메시지의 식별자 (IDR)를 포함합니다 두 번째 단어는 RXFIFO\_DLC 레지스터를 사용하여 읽혀지며 16 비트 타임 스탬프와 DLC (Data Length Code) 필드를 포함합니다. 세 번째와 네 번째 단어는 데이터 단어 1 (DW1R)과 데이터 단어 2 (DW2R)를 포함합니다. RxFIFO 레지스터에 대한 쓰기는 무시한다. 빈 RxFIFO의 읽기 데이터가 유효하지 않으며 인터럽트를 생성 할 수 있습니다.  
RxFIFO의 메시지는 CAN 컨트롤러가 Bus off 상태 또는 Configuration 모드로 들어가도 유지된다

**Tx Messages 전송메세지**

컨트롤러는 최대 64 개의 Tx CAN 메시지 버퍼를 사용할 수있는 구성 가능한 TxFIFO를 가지고 있습니다. 컨트롤러에는 또한 우선 순위가 높은 전송 버퍼 (Tx HPB)가 있으며 하나의 메시지 저장 공간이 있습니다. 우선 순위가 더 높은 메시지를 전송해야 할 때 소프트웨어는 우선 순위가 높은 전송 버퍼가 사용 가능한 경우 메시지를 메시지에 씁니다. TxHPB의 메시지는 TxFIFO의 메시지보다 우선 순위가 높습니다.  
메시지 전송 중에 중재 손실이나 오류가 발생하면 컨트롤러는 메시지를 다시 전송하려고합니다. 후속 메시지가 없으며 원래의 메시지가 오류 또는 중재 손실없이 전송 될 때까지 더 새롭고 높은 우선 순위의 메시지가 전송됩니다

컨트롤러는 IDR 워드의 비트 31부터 시작하여 메시지를 전송합니다. 식별자 워드가 전송 된 후 DLCR 워드가 전송됩니다. 다음에 DB0, DB1, ... DB7 순서로 데이터 바이트가옵니다. 메시지의 데이터 부분의 마지막 비트는 DB7, 비트 0입니다. 표 18-2, 563 페이지를 참조하십시오

상태 비트 인 can.ISR [TXOK]는 컨트롤러가 TxFIFO 또는 TxHPB에서 메시지를 성공적으로 전송 한 후 1로 설정됩니다.  
TxFIFO 및 TxHPB의 메시지는 CAN 컨트롤러가 Bus off 상태 또는 Configuration 모드로 들어도 유지됩니다.  
메시지 형식은 18.2.2 메시지 형식에서 설명합니다

**Reads from RxFIFO RxFIFO를 읽기**

전체 메시지를 수신하려면 RxFIFO에서 16 바이트를 모두 읽어야합니다. 첫 번째 단어 읽기 (4 바이트)는 수신 된 메시지의 식별자 (IDR)를 반환합니다. 두 번째 읽기는받은 메시지 (DLCR)의 16 비트 수신 타임 스탬프 및 DLC (Data Length Code) 필드를 반환합니다. 세 번째 읽기는 데이터 워드 1 (DW1R)을 반환하고 네 번째 읽기는 데이터 워드 2 (DW2R)를 반환합니다.  
자유 실행 16 비트 카운터는 메시지가 성공적으로 수신 된 시간을 기준으로 타임 스탬프를 제공합니다.  
메시지에 8 바이트 미만의 데이터가 포함되어 있어도 각 메시지에 대해 4 개의 단어를 모두 읽어야합니다. RxFIFO에 대한 쓰기 처리가 무시됩니다. 빈 RxFIFO에서 읽기가 잘못된 데이터를 반환하고 Rx 언더 플로우 인터럽트를 생성합니다

**Rx and Tx Error Counters Rx 및 Tx 오류 카운터**

Rx 또는 Tx 오류가 발생하면 프로토콜 엔진의 관련 오류 카운터 (18.2.6 절 참조)가 증가합니다. 두 오류 카운터는 8 비트 폭이며 읽기 전용 can.ECR 레지스터, 비트 필드 REC 및 TEC를 사용하여 읽습니다.  
다음과 같은 상황이 발생하면 Rx 및 Tx 카운터가 재설정됩니다

•can.Srr[SRST] field=1이된후1이기록된다 이비트 쓰는 것은 자동소거된다

•언제든지 can.SRR[CEN]=0 (구성모드)

•컨트롤러는 버스가 꺼진상태로 들어간다

**18.2.4 Interrupts**

각 CAN 컨트롤러에는 GIC인터럽트컨트롤러에 대한 단일 인터럽트 신호가 있습니다. CAN 0은 IRQ ID # 60에 연결하고 CAN 1은 ID # 83에 연결한다 인터럽트의 소스는 다음 중 하나로 그룹화 할 수 있다.  
• TxFIFO 및 TxHPB • RxFIFO  
• 메시지 전달 및 중재  
• 수면 모드 및 버스 끄기  
can.IER 레지스터를 사용하여 인터럽트를 활성화 및 비활성화한다. can.ISR을 사용하여 인터럽트의 초기상태를 확인할수있다. can.ICR에 1을 써서 인터럽트를 지울수있다 일부 인터럽트 소스에는 표 18-4와 같이 인터럽트를 지우는 추가 방법이 있다  
**List of Interrupts 인터럽트목록**

모든 CAN 인터럽트는 고정적이다. 즉, 일단 하드웨어가 설정하면 소프트웨어에 의해 지워질 때까지 설정된 상태를 유지합니다. CAN 상태와 인터럽트는 표 18-4에 나와 있습니다

**RxFIFO and TxFIFO Interrupts**

FIFO 워터 마크 레벨과 모든 FIFO 인터럽트는 그림 18-4, CAN RxFIFO 및 TxFIFO 워터 마크 인터럽트에 설명되어 있습니다

**Example: Program RxFIFO Watermark Interrupt (12)**

**예 : RxFIFO 워터 마크 인터럽트 프로그램 (12)**다음 단계를 사용하여 RxFIFO 워터 마크 인터럽트를 설정하고 제어 할 수 있습니다. 그림 18-4를 참조하십시오. 워터 마크 상태 및 제어 인터럽트는 572 페이지의 프로토콜 엔진에 설명되어 있습니다.  
1. RxFIFO 워터 마크 인터럽트를 비활성화합니다. can.IER [12]에 0을 씁니다.  
2. RxFIFO 전체 워터 마크 레벨을 프로그램하십시오. can.WIR [FW].에쓰기  
3. RxFIFO 워터 마크 인터럽트를 지우십시오. can.ICR [12]에 1을 씁니다.  
4. RxFIFO 워터 마크 상태를 읽습니다. 읽기 can.ISR [12].  
5. RxFIFO 워터 마크 인터럽트를 활성화합니다. can.IER [12]에 1을 씁니다.

**Example: Program TxFIFO Watermark Interrupt (13)**

**예 : TxFIFO 워터 마크 인터럽트 프로그램 (13)**  
다음 단계를 사용하여 TxFIFO 워터 마크 인터럽트를 설정하고 제어 할 수 있습니다. 그림 18-4를 참조하십시오. 워터 마크 상태 및 제어 인터럽트는 572 페이지의 프로토콜 엔진에 설명되어 있습니다.  
1. TxFIFO 워터 마크 인터럽트를 비활성화하십시오. can.IER [13]에 0을 씁니다.  
2. TxFIFO 빈 워터 마크 레벨을 프로그래밍하십시오. can.WIR [EW]쓰기   
3. TxFIFO 워터 마크 인터럽트를 지우십시오. can.ICR [1]에 1을 쓴다.  
4. TxFIFO 워터 마크 상태를 읽습니다. can.ISR [13]읽기.  
5. TxFIFO 워터 마크 인터럽트를 활성화합니다. can.IER [13]에 1을 씁니다

**Example: Program TxFIFO Empty Interrupt (14)**

**예 : 빈 TxFIFO 인터럽트 프로그램 (14)**  
다음 단계를 사용하여 빈 TxFIFO 인터럽트를 제어 할 수 있습니다.  
1. 빈 TxFIFO 인터럽트를 비활성화하십시오. can.IER [14]에 1을 씁니다.  
2. 빈 TxFIFO 인터럽트를 지우십시오. can.ICR [1]에 1을 씁니다.  
3. 빈 TxFIFO 인터럽트를 활성화합니다. can.IER [14]에 1을 씁니다.  
4. 빈 TxFIFO 상태를 읽습니다. 읽기 can.ISR [14].

TxFIFO가 비어있는지 안비어있는지 상태를 나타낸다

**18.2.5 Rx Message Filtering 수신 메시지 필터링**

수신메세지를 필터링 하려면 수신 마스크 및 ID레지스터가 있는 최대 4개의 수신필터를 구성하고 활성화하여 메시지를 RxFIFO에 저장할지 여부를 결정하거나 메시지를 확인하고 무시한다

수신필터링은 다음순서로 수행된다

1. 들어오는 식별자는 수신필터마스크레지스터의 비트로 마스킹된다
2. 수신필터 ID레지스터는 또한 수신필터마스크레지스터의 비트로 마스킹된다
3. 두결과값이 비교된다
4. 이 두값이 동일하면 메시지가 RxFIFO에 저장된다
5. 수신필터링은 정의 된 각 필터로 처리된다 수신 식별자가 수용필터를 통과하면 메시지는 RxFIFO에 저장된다

**Acceptance Filter Enable 수신필터 사용**

수신필터레지스터(Acceptance Filter Register AFR)는 사용할 수신필터를 정의한다 여기에는 4개의 수신필터에 해당하는 4개의 활성화 비트가 포함된다 각각의 수신필터ID레지스터(AFIR)및 수신필터마스크레지스터(AFMR)한쌍은(두개) 사용수신필터(UAF)비트와 관련된다

UAF비트가1일 때 수신필터링을 위해 해당 수신필터 쌍이 수신필터링에 사용되지 않는다

일반모드에서 수신필터 쌍을 수정하려면 이 레지스터의 해당 UAF비트를 먼저 0으로 설정해야 한다 수신필터를 수정한 후에는 해당 UAF 비트를 1로 설정해야 필터를 사용할 수 있다

can.AFR 레지스터의 UAF 비트는 Rx 수용 필터를 활성화한다

• 모든 UAF 비트가 0으로 설정된 경우 수신 된 모든 메시지는 RxFIFO에 저장됩니다.  
• CAN 메시지 수신 중 UAF 비트가 1에서 0으로 변경되면 메시지는 RxFIFO에 저장되지 않습니다.  
  
활성화 된 필터 (최대 4 개) 중 하나라도 이 방정식을 만족하면 Rx 메시지가 RxFIFO에 저장됩니다.  
If (AFMR & Message\_ID) == (AFMR & AFIR) then Capture Message 인 경우 메시지캡쳐

각 수신 필터는 독립적으로 활성화됩니다. 필터는 can.AFR 레지스터에 의해 선택됩니다.  
• AFMR4 및 AFID4를 활성화하려면 can.AFR [UAF4] = 1로 설정하십시오.  
• AFMR3 및 AFID3을 활성화하려면 can.AFR [UAF3] = 1로 설정하십시오.  
• AFMR2 및 AFID2를 활성화하려면 can.AFR [UAF2] = 1로 설정하십시오.  
• AFMR1과 AFID1을 활성화하려면 can.AFR [UAF1] = 1로 설정하십시오.  
모든 can.AFR [UAFx] 비트가 0으로 설정된 경우 수신 된 모든 메시지는 RxFIFO에 저장됩니다. UAF 비트는 수신 메시지의 시작 부분에서 하드웨어에 의해 샘플링됩니다

**Acceptance Filter Mask 수신 필터 마스크**

수신필터마스크레지스터(AFMR)는 수신필터링에 사용되는 마스크비트를 포함합니다 메시지프레임의 수신메세지식별자부분은 수신필터 ID레지스터에 저장된 메세지 식별자와 비교됩니다 마스크 비트는 수신필터ID레지스터에 저장된 어떤 식별자비트가 수신메세지식별자와 비교되는지 정의한다 4 개의 AFMR이 있다. 이 레지스터는 메모리에 저장된다. 메모리가 초기화되지 않은 경우 AFMR에서 읽기는 'X'를 반환한다. 소프트웨어 리셋 또는 하드웨어 리셋을 선언해도 레지스터 내용이 지워지지 않는다. 이 레지스터는 읽고 쓸 수 있다. 이 레지스터는 can의 해당 UAF 비트가 있을 때만 기록된다.  
AFR 레지스터는 0이고 can.SR 레지스터의 ACFBSY 비트는 0이다.

AFMR에 적용되는 조건은 다음과 같다.  
확장 된 프레임 => 모든 비트 필드 (AMID [28:18], AMSRR, AMIDE, AMID [17 : 0] 및 AMRTR)를 정의해야한다.  
표준 프레임 => AMID [28:18], AMSRR 및 AMIDE 만 정의해야한다. AMID [17 : 0] 및 AMRTR은 0으로 기록해야한다

**Acceptance Filter Identifier 수신필터식별자**

수신필터ID레지스터(AFIR)는 수신필터링에 사용되는 식별자비트를 포함한다. 4개의 Acceptance Filter ID 레지스터가 있다  
이 레지스터는 읽고 쓸 수 있습니다. 이 레지스터는 SR의 해당 UAF 비트가 0이고 SR의 ACFBSY 비트가 0 인 경우에만 기록되어야한다.

AFIR의 사용에 적용되는 조건은 다음과 같습니다.  
확장 된 프레임  
 모든 비트 필드 (AIID [28..18], AISRR, AIIDE, AIID [17 : 0] 및 AIRTR)가 정의되어야한다.  
표준 프레임  
AIID [28:18], AISRR 및 AIIDE 만 정의해야한다. AIID [17 : 0] 및 AIRTR은 0으로 작성해한다.

사용자는 표준 및 확장 프레임에 대해 IDE 비트를 올바르게 프로그래밍해야한다. 사용자가 AMIR의 IDE 비트를 0으로 설정하면 표준 프레임 ID 검사로 간주된다.  
예 : 프로그램 수신 필터  
각 수신 필터는 자체 마스크 can.AFMR {1,2,3,4} 및 ID 레지스터 can.AFIR {1,2,3,4}을가집니다.  
1. 수신 필터를 비활성화를 한다. can.AFR 레지스터에 0을 쓴다.  
2. 필터가 사용 중이 지 않을 때까지 기다린다. Can.SR [ACFBSY] 를 0으로 설정한다   
3. 필터 마스크와 ID를 작성한다. 한 쌍의 AFMR 및 AFIR 레지스터에 쓰기 (아래 예제 참조).  
4. 필터 마스크와 ID를 추가로 작성한다. 2 단계로 이동하십시오.  
5. 하나 이상의 필터를 활성화한다. 모든 필터를 활성화하려면 0x0000\_000F를 can.AFR 레지스터에 쓴다

**Program the AFMR and AFIR Registers AFMR 및 AFIR 레지스터 프로그래밍**

Tx 메시지를 컨트롤러에 보내는 데 유효한 필드는 표 18-5에 요약되어 있다. 이 필드는 18.2.2 메시지 형식에서 설명한다

AFMR 마스크 레지스터에서 비트 필드에 1을 기록하여 수신 Rx CAN 메시지의 각 필드에 대한 비교 기능을 활성화 (마스크 해제)한다. AFIR 레지스터에서 들어오는 Tx CAN 메시지와 비교할 값을 작성해야한다 .

예 : 표준 프레임에 대해 AFMR 및 AFIR 프로그래밍  
이 예제는 표준 프레임용 수신필터를 설정한다. 프레임 ID 번호는 0x5DF로 표시되지만 응용 프로그램에 대해 원하는 값으로 설정할 수 있다.  
1. 표준 프레임에 대해 필터 마스크를 구성한다.

1. can.AFMR 레지스터에 0xFFF8\_0000을 쓴다. 표준 메시지 ID 인 [AMIDE] = 1에 대한 비교를 활성화한다.
2. 표준 메시지 ID 인 [AMIDH] = 0x7FF의 모든 비트를 비교해야한다.
3. 대체 원격 전송 요청 인 [AMSRR] = 1에 대한 비교를 활성화합니다.
4. 확장 된 프레임 비트를 제로 아웃해야한다. [AMIDL, AMRTR] = 0

2. 표준 프레임에 대해 필터 ID를 구성하십시오. 0xABC0\_0000을 can.AFIR 레지스터에 쓴다

a. 표준 프레임 메시지 모드, [AIIDE] = 0을 선택해야 한다

b. 표준 메시지 ID, [AIIDH] = 0x55E를 프로그램해야 한다.

c. 대체 원격 전송 요청, [AISRR] = 0을 비활성화한다.

d. 확장 프레임 비트를 제로 - 아웃, [AIIDL, AIRTR] = 0.

**Example: Program the AFMR and AFIR for Extended Frames**

**=>예 : 확장 된 프레임에 대해 AFMR 및 AFIR 프로그래밍**

이 예에서는 확장 된 프레임에 대한 승인 필터를 설정해야한다. 프레임 ID 번호는 0x5DF로 표시되지만 응용 프로그램에 대해 원하는 값으로 설정할 수 있다.

1. 확장 된 프레임에 대해 필터 마스크를 구성해야한다. can.AFMR 레지스터에 0xFFFF\_FFFF를 쓴다.

a. 프레임에 대해 대체 원격 전송 요청 마스크 인 [AMSRR] = 1을 사용가능하게 한다.  
b. 표준 메시지 ID 인 [AMIDH] = 0x7FF에 대한 비교의 모든 비트를 비교해야한다 .  
c. 확장 된 프레임 인 [AIIDE] = 1을 사용한다.  
d. 확장 ID, [AIIDL] = 0x3FFFF  
e. 확장 프레임에 대한 원격 전송 요청 비트, [AIRTR] = 1.

2. 확장 된 프레임에 대한 필터 ID를 구성한다. 0xABDF\_9BDE를 can.AFIR 레지스터에 쓴다.  
a. 표준 ID, [AIIDH] = 0x55E.  
b. 표준 프레임에 대한 원격 전송 요청 비트, [AISRR] = 1.  
c. 표준 / 확장 프레임, [AIIDE] = 1을 선택한다.

d. 확장 ID, [AIIDL] = 3CDEF.  
e. 확장 프레임에 대한 원격 전송 요청 비트, [AIRTR] = 0

**Protocol Engine 프로토콜 엔진**

CAN 프로토콜 엔진은 주로 비트 타이밍 로직 (BTL)과 비트 스트림 프로세서 (BSP) 모듈로 구성된다. 그림 18-5는 CAN 프로토콜 엔진의 블록 다이어그램을 보여준다

**Rx/Tx Bit Timing Logic Rx / Tx 비트 타이밍 로직**

비트타이밍로직(BLT)모듈의 주요기능은 다음과 같다

\*비트스트림프로세서(BSP)에 대한 Rx샘플링 클럭생성

\*CAN컨트롤러를 버스상의 CAN트랙피과 동기화 한다

\*버스 샘플링 및 수신 중 버스에서 데이터 스트림 추출

\*전송중에 전송비트 스트림을 버스에 삽입한다

비트타임의 클록 주기의 공칭길이는 CAN\_REF\_CLK 클록주파수, 보 레이트(보레이트:데이터통신에서 직렬전송의 변조속도를 1초간에 전송되는 신호의 수로 나타낸 값 즉 직렬데이터의 전송속도를 나타내는 단위이다) 생성기 디바이더(can.BPRP 레지스터)와 세그먼트 길이(can.BTR레지스터)를 기반으로 한다

비트타이밍로직모듈은 can,BPR[SJW]빝 필드의 sync width 매개 변수를 사용하여 CAN에 대한 재 동기화 기능을 관리합니다 CAN비트 타이밍은 그림 18-6에 나와 있다

.동기화 세그먼트 수는 항상 한 퀀텀주기(프로그램에 주어지는 최소의 시간단위)와 같다 TS1과TS2 주기 카운트는 can.BPR[TS1,TS2]비트 필드를 사용하여 프로그래밍 할수 있다 이 레지스터느 컨트롤러가 구성모드에 있을 때 기록된다 전파세그먼트(PROP\_SEG)의 너비는 실제 전파지연보다 작아야한다

**Time Quanta Clock 시간 퀀타 클럭**

시간 퀀텀 클럭 (TQ\_CLK)은 컨트롤러 기준 클럭 (CAN\_REF\_CLK)을 보레이터 프리스케일러 (BRP)로 나눈 값에서 나온다

팁:주어진 비트율은 여러 비트 시간 구성으로 달성 할 수 있지만 오실레이터 허용 오차 및 CAN 전파 지연을 신중하게 고려한 후에 값을 선택해야한다. CAN 비트 타임 레지스터 설정에 대한 자세한 내용은 CAN 2.0A, CAN 2.0B 및 ISO 11898-1 사양을 참조.

**Bitstream Processor 비트 스트림 프로세서**

비트 스트림 프로세서 (BSP) 모듈은 CAN 메시지를 송수신하는 동안 몇 가지 기능을 수행합니다. BSP는 TxFIFO 또는 TxHPB로부터 전송할 메시지를 획득하고 비트 스트림을 BTL로 전달하기 전에 다음 기능을 수행합니다.  
• 메시지 직렬화  
• 전송 중에 스터프 비트, CRC 비트 및 기타 프로토콜 정의 필드 삽입  
  
전송 중에 BSP는 Rx 데이터를 동시에 모니터링하고 버스 중재 작업을 수행합니다. 그런 다음 중재가 이루어질 때 전체 프레임을 전송하고 중재가 손실되면 다시 시도합니다.

수신하는 동안 BSP는 수신 비트 스트림에서 스터프 비트, CRC 비트 및 기타 프로토콜 필드를 제거합니다. BSP 상태 머신은 또한 폼, CRC, ACK, Stuff 및 Bit 위반에 대한 송수신 중 버스 트래픽을 분석합니다. 그런 다음 상태 시스템은 오류 신호 및 오류 제한 태스크를 수행합니다. CAN 컨트롤러는 자발적으로 과부하 프레임을 생성하지 않지만 버스에서 감지 된 과부하 플래그에 응답합니다.  
이 모듈은 CAN 컨트롤러의 오류 상태를 결정합니다 : 오류 활성, 오류 수동 또는 버스 오프. 버스에서 Tx 또는 Rx 오류가 관찰되면 BSP는 CAN 2.0 A, CAN 2.0 B 및 ISO 11898-1 표준에 정의 된 규칙에 따라 전송 및 수신 오류 카운터를 업데이트합니다. 이 카운터의 값에 따라 CAN 컨트롤러의 오류 상태가 BSP에 의해 업데이트됩니다.

**CAN0-to-CAN1 Connection CAN0과 CAN1 연결**

PS의 두 CAN 컨트롤러의 I / O 신호는 서로 연결될 수 있습니다. 이 모드에서는 하나의 CAN 컨트롤러의 RX 신호가 다른 컨트롤러의 TX 신호에 연결됩니다. 이러한 연결은 slcr.MIO\_LOOPBACK [CAN0\_LOOP\_CAN1] 비트를 사용하여 활성화됩니다.

**Programming Guide 프로그래밍 가이드**

**Overview 개요**

컨트롤러에는 메시지를 수신하고 전송하는 여러 가지 작동 모드와 다른 방법이 있습니다. 하위 수준 기능은 18.2 기능 설명에 설명되어 있습니다. 시스템 레벨 연산은 18.4 절 시스템 함수에서 설명한다.  
모든 컨트롤러 레지스터는 표 18-6에 나열되어 있으며 부록 B, 레지스터 세부 정보에서 자세히 설명합니다

**Configuration Mode State 구성 모드 상태**

CAN 컨트롤러는 다음과 같은 동작이 수행 될 때 작동 모드와 관계없이 구성 모드로 들어갑니다.  
• SRR 레지스터의 CEN 비트에 0을 씁니다.  
• SRR 레지스터의 SRST 비트에 1을 씁니다. 컨트롤러는 소프트웨어 재설정 직후에 구성 모드로 들어갑니다.  
• SLCR을 통해 제어되는 리셋 입력에서 1을 구동. 컨트롤러는 리셋이 1 인 동안 리셋 상태를 계속 유지합니다. 컨트롤러는 리셋이 0으로 무효화 된 후 구성 모드로 들어갑니다

구성 모드에서 다음이 적용됩니다.  
• CAN 컨트롤러는 CAN 버스와의 동기화를 잃고 버스 라인에서 일정한 열성 비트를 구동합니다.  
• 에러 카운트 레지스터 (ECR)가 리셋됩니다.  
• 오류 상태 레지스터 (ESR)가 재설정됩니다.  
• 비트 타이밍 레지스터 (BTR) 및 보드 율 프리스케일러 레지스터 (BRPR)를 수정할 수 있습니다.  
• CAN 컨트롤러는 새로운 메시지를 수신하지 않습니다.  
• CAN 컨트롤러는 어떤 메시지도 전송하지 않습니다. TxFIFO 및 TxHPB의 메시지가 추가됩니다. 이러한 패킷은 정상 작동이 재개 될 때 전송됩니다.  
• RxFIFO에서 읽기를 수행 할 수 있습니다.  
• TxFIFO 및 TxHPB에 대한 쓰기가 수행 될 수 있습니다 (SNOOP 비트가 설정되지 않은 경우).  
• 인터럽트 상태 레지스터 비트 ARBLST, TXOK, RXOK, RXOFLW, ERROR, BSOFF, SLP 및 WKUP이 해제됩니다.  
• 인터럽트 상태 레지스터 비트 RXNEMP 및 RXUFLW는 RxFIFO에 대한 읽기 작업으로 인해 설정할 수 있습니다.  
• 인터럽트 상태 레지스터 비트 TXBFLL 및 TXFLL 및 상태 레지스터 비트 TXBFLL 및 TXFLL은 각각 TX HPB 및 TX FIFO에 대한 쓰기 작업으로 인해 설정할 수 있습니다.  
• 인터럽트 활성화 레지스터 (IER)의 해당 비트가 1이면 인터럽트가 발생합니다.  
• 모든 구성 레지스터에 액세스 할 수 있습니다

구성 모드에서 CAN 컨트롤러는 SRR 레지스터의 CEN 비트가 1로 설정 될 때까지 이 모드로 유지됩니다. CEN 비트가 1로 설정된 후 CAN 컨트롤러는 구성 모드를 종료하기 전에 11 개의 열성비트 결과를 기다립니다.  
CAN 컨트롤러는 MSR 레지스터의 LBACK, SNOOP 과 SLEEP 비트에 따라 구성 모드에서 정상, 루프백, 스누핑 또는 슬립 모드로 전환됩니다

**Start-up Controller 스타트 업 컨트롤러**

컨트롤러는 Normal, Sleep, Snoop 및 Loop Back 모드에서 작동 할 수 있습니다. 지원되는 변환에 대해서는 그림 18-3을 참조하십시오. 시동시 컨트롤러 클록 및 구성 비트가 프로그래밍됩니다. 그런 다음 작동 모드가 선택되고 활성화됩니다.  
예 : 시작 순서  
1. 시계를 구성하십시오. 18.4.1 절을 참조하십시오.  
2. Tx / Rx 신호를 구성하십시오. 18.5.1 절 MIO 프로그래밍을 참조하십시오.  
3. 구성 모드를 기다립니다. 읽기 can.SR [CONFIG] 때까지 1 같음.  
4. 컨트롤러를 재설정하십시오. 컨트롤러가 구성 모드로 표시됩니다. 18.4.2 리셋을 참조하십시오.  
5. 비트 샘플링 클럭을 프로그래밍하십시오. Rx / Tx 비트 타이밍 로직 섹션을 참조하십시오.  
6. 필요에 따라 인터럽트를 프로그래밍하십시오. 18.2.4 인터럽트 섹션을 참조하십시오.  
7. 수용 필터를 프로그램하십시오. 18.2.5 Rx 메시지 필터링 섹션을 참조하십시오.  
8. 작동 모드를 선택하십시오. 정상, 수면, 스눕 또는 루프백. 18.3.4 절 작동 모드 변경을 참조하십시오.  
9. 컨트롤러를 활성화하십시오. can.SRR [CEN]에 1을 씁니다.

**Change Operating Mode 작동 모드 변경**

예 : 표준 모드에서 절전 모드  
절전 모드는 다음 조건이 충족 될 때 표준 모드에서 시작됩니다.  
1. 절전 모드를 선택하십시오. can.MSR [SLEEP]에 1을 씁니다.  
2. CAN 버스가 유휴 상태가 될 때까지 기다립니다.  
3. 모든 TxFIFO 및 TxHPB 메시지가 전송 될 때까지 기다립니다.  
참고 : 정상 모드에서 can.MSR [LBACK] = 0이고 can.SSR [CEN] = 1입니다. 또한 can.MSR [SNOOP] = 상관 없습니다.  
예 : 절전 모드로 구성  
다음 조건이 충족되면 절전 모드가 구성 모드에서 시작됩니다.  
1. 절전 모드를 선택하십시오. can.MSR [SLEEP]에 1을 기록하고 can.MSR [LBACK]에 0을 씁니다.  
2. 컨트롤러를 활성화하십시오. can.SSR [CEN]에 1을 기록하십시오.  
3. TxFIFO 또는 TxHPB가 비어있을 때까지 기다립니다.  
참고 : 구성 모드에서 can.MSR [SNOOP] = 상관 없습니다.  
절전 모드는 I / O 버스 활동이 감지되거나 소프트웨어가 TxFIFO 또는 TxHPB에 메시지를 쓸 때 종료됩니다. 컨트롤러가 sleep 모드를 종료하면 하드웨어에서 can.MSR [SLEEP]을 0으로 설정하고 인터럽트를 생성 할 수 있습니다.

**Write Messages to TxFIFO TxFIFO에 메시지 쓰기**

두 옵션 중 하나를 사용하면 메시지를 작성하기 전에 can.SR [TXFLL]을 폴링 할 수 있습니다.  
  
TxFIFO에 쓰여지는 모든 메시지는 메시지 구조에 정의 된 형식을 따라야합니다.  
  
예제 : 폴링 방법을 사용하여 TxFIFO에 메시지 쓰기  
  
1. TxFIFO 상태를 폴링 합니다. Read can.SR [TXFLL] = 0, can.SR [TXFEMP] = 1이면 메시지를 TxFIFO에 쓸 수 있습니다.  
2.TxFIFO에 메시지를 씁니다. 4개의 데이터레지스터(can.TXFIFO\_ID, can.TXFIFO\_DLC, can.TXFIFO\_DATA1 및 can.TXFIFO\_DATA2) 모두 쓴다.  
  
예 :인터럽트 방법을 사용하여 TxFIFO에 메시지 쓰기  
  
인터럽트 모드에서는 can.ISR [TXFLL]이 인터럽트를 생성 할 때까지 쓰기를 계속할 수 있습니다.  
  
TxFIFO가 가득 찰 때까지 메시지를 계속 TxFIFO에 쓸 수 있습니다. TxFIFO가 가득 차면 can.ISR [TXFLL] 및 can.SR [TXFLL]이 1로 설정됩니다. TxFIFO가 비어 있으면 can.ISR [TXFEMP]이 1로 설정됩니다

**Write Messages to TxHPB TxHPB에 메시지 쓰기**

TxHPB에 쓰여진 모든 메시지는 폴링 방법을 사용합니다. 형식은 18.2.2 메시지 형식을 따라야합니다.  
  
예 : TxHPB에 메시지 쓰기  
  
1. TxHPB 상태를 폴링하십시오. can.SR [TXBFLL]이 0이 될 때까지 읽은 다음 메시지를 TxHPB에 씁니다.  
2. TxHPB에 메시지를 씁니다. 네 개의 모든 데이터 레지스터 (can.TXHPB\_ID, can.TXHPB\_DLC, can.TXHPB\_DATA1 및 can.TXHPB\_DATA2)에 기록하십시오.

**Read Messages from RxFIFO RxFIFO에서 메시지 읽기**

RxFIFO에서 메시지 읽기  
새로운 메시지가 수신되어 RxFIFO에 입력 될 때마다 can.ISR [RXNEMP] 및 can.ISR [RXOK] 비트는 1로 설정됩니다. 메시지를 읽을 때 RxFIFO가 비어 있으면 can.ISR [RXUFLW ]도 1로 설정됩니다.  
  
예 : 폴링 방법을 사용하여 RxFIFO에서 메시지 읽기  
1. RxFIFO 상태를 폴링하십시오. 메시지를 수신 할 때까지 can.ISR [RXOK] 또는 can.ISR [RXNEMP] 레지스터를 읽습니다. 비트가 설정되면 2 단계로 진행하십시오.

2. RxFIFO에서 메시지를 읽습니다. 4개의레지스터 모두를 읽습니다 (can.RXFIFO\_ID,can.RXFIFO\_DLC, can.RXFIFO\_DATA1, can.RXFIFO\_DATA2).  
3. RxFIFO에 더 많은 메시지가 있는지 판별하십시오. can.ISR [RXNEMP]. 읽기

예 : 인터럽트 방법을 사용하여 RxFIFO에서 메시지 읽기  
can.ISR [RXOK] 및 / 또는 can.ISR [RXNEMP] 비트 필드는 인터럽트를 생성 할 수 있습니다.  
1. RxFIFO 워터 마크 레벨 인터럽트를 프로그램하십시오. WIR [FW]로 워터 마크를 설정할 수 있습니다. ISR [RXFWMFLL] interrupt.  
2. 인터럽트가 수신되면 3 단계로 진행합니다.  
3. 메시지가 수신 될 때까지 기다리십시오. 읽기 can.ISR [RXOK] 또는 can.ISR [RXFWMFLL].  
4. RxFIFO에서 메시지를 읽는다. 4개의 레지스터 모두를 읽는다 (can.RXFIFO\_ID, can.RXFIFO\_DLC, can.RXFIFO\_DATA1, can.RXFIFO\_DATA2).  
5. RxFIFO가 비어 있지 않은지 판별하십시오. 읽기 can.ISR [RXNEMP].  
6. RxFIFO가 비어있을 때까지 반복하십시오.  
7. 인터럽트를 지우십시오

**Register Overview 레지스터개요**

제어 및 상태 레지스터는 표 18-6을 참조. 이 레지스터들 각각은 32 비트 폭이다. 제한되지 않은 비트 또는 사용되지 않은 비트에 대한 읽기 작업은 0을 반환한다. 0은 제한되지 않은 비트 및 사용되지 않는 비트 필드에 써야한다. 제한된 위치에 대한 쓰기는 무시된다

**System Functions**

**Clocks**

컨트롤러 및 I / O 인터페이스는 기준 클럭 (CANx\_REF\_CLK)에 의해 구동됩니다. 컨트롤러의 상호 연결에는 APB 인터페이스 클록이 필요합니다. APB 상호 연결 클럭 (CPU\_1x)은 항상 PS 클럭 하위 시스템에서 발생합니다.  
  
기준 클록은 일반적으로 PS 클럭 서브 시스템에서 발생하지만, 대신에 사용 가능한 MIO 핀을 통해 외부 클럭 소스에 의해 구동 될 수있다. 기준 클럭은 프로토콜 엔진, 보 레이트 생성기 및 데이터 경로에 의해 사용됩니다. 컨트롤러는 PS 클럭 하위 시스템과 동일한 기준 클럭 주파수를 공유합니다. 기준 클럭이 MIO 핀에서 나온 것이라면 주파수는 다를 수 있습니다

**CPU\_1x Clock CPU\_1x클럭**

일반적인 클럭 프로그래밍 정보는 25 장, 클럭을 참조하십시오. CPU\_1x 클럭은 CAN 기준 클럭과 비동기로 실행됩니다.

**Reference Clock 기준 클럭**

CAN\_REF\_CLK는 일반적으로 PS 클럭 하위 시스템으로부터 출처를 나타내지 만 MIO 핀을 통해 외부 클럭 소스에 의해 구동 될 수 있습니다. 내부적으로 PS에는 3 개의 PLL과 2 개의 클록 구분기 쌍이 있습니다. 클록 소스 선택, PS 클럭 서브 시스템 또는 외부 MIO 핀은 CAN\_MIOCLK\_CTRL 레지스터에 의해 제어된다.  
  
PS의 CAN 클럭은 slcr.CAN\_CLK\_CTRL에 의해 제어됩니다. PS에 의한 CAN 기준 클럭의 생성은 섹션에 설명되어 있습니다. Quad-SPI 클럭은 Quad-SPI 보드 속도 분배기를 사용하여 최소 2 개로 나뉘어져 있습니다. 12.4.1 클럭을 참조하십시오. 마스터 모드에서 SPI 클럭은 SPI 보드 분배기를 사용하여 4 분의 1로 나뉘어져 있습니다. 17.4.2 클럭을 참조하십시오. 두 개의 CAN 컨트롤러 모두에 대해 PS에 클럭 발생기가 하나 있습니다. MIO 핀이 대신 사용되면 선택된 MIO\_PIN Mux 레지스터가 입력으로 프로그래밍됩니다.

예 : 기준 클럭에 대한 내부경로 클럭 및 구성

클럭을 구성하고 MIO 경로를 비활성화하십시오. PLL이 1000MHz에서 작동하고 필요한 CAN 기준 클럭이 24MHz (23.8095MHz)라고 가정합니다.  
1. 클럭 서브 시스템을 프로그래밍하십시오.  
slcr.CAN\_CLK\_CTRL 레지스터에 0x0030\_0E03을 씁니다.  
a. 두 CAN 기준클럭 모두를 활성화하십시오

b. I / O PLL 클럭을 42 (0x02A)로 나눕니다. DIVISOR0 = 0x0E 및 DIVISOR1 = 0x03이 두 컨트롤러에서 사용됩니다.  
2. MIO 경로를 비활성화하십시오. slcr.CAN\_MIOCLK\_CTRL 레지스터에 0x0000\_0000을 써 넣어 두 컨트롤러의 내부 클록 하위 시스템 / PLL에서 클록을 선택합니다

예 : MIO 핀의 소스 컨트롤러 클럭  
이 예제는 MIO 핀 45를 컨트롤러 클럭 기준으로 사용합니다

1. MIO 장치 핀을 구성하십시오. slcr.MIO\_PIN\_45 레지스터에 0x0000\_1200을 씁니다.  
a. MIO 핀 45를 GPIO 컨트롤러에 연결하십시오 (다음 단계에서 무시 됨).  
b. 출력 드라이버를 사용하지 않도록 설정하십시오 (TRI\_ENABLE = 1).  
c. LVCMOS18 (다른 전압 옵션의 레지스터 정의 참조).  
d. 느린 CMOS 에지 (양호한 설정).  
e. 내부 풀업 저항을 활성화하십시오.  
f. HSTL 수신기를 비활성화하십시오.  
2. MIO 경로를 활성화하십시오. 이전 단계에서 작성된 MIO PIN 레지스터 설정을 무시하려면 slcr.CAN\_MIOCLK\_CTRL 레지스터에 기록하십시오.  
slcr.CAN\_MIOCLK\_CTRL [CANx\_\_REF\_SEL]에 1을 쓰고 이전 단계의 핀과 일치시키기 위해 slcr.CAN\_MIOCLK\_CTRL [CANx\_MUX] 비트 필드에 원하는 MIO 핀 번호를 씁니다.

**Resets**

각 재설정 유형에 대한 영향은 표 18-7에 요약되어 있습니다

예 : 로컬 CAN 리셋을 사용한 리셋  
1. 로컬 CAN 리셋 레지스터에 기록하십시오. can.SRR [SRST] 비트 필드에 1을 씁니다. 이 비트는 자체 삭제됩니다.  
예 : Reset Subsystem을 사용한 재설정  
1. CAN 용 slcr 리셋 레지스터에 기록하십시오. slcr.CAN\_RST\_CTRL [CANx\_CPU1X\_RST] 비트 필드에 1을, 그 뒤에 0을 씁니다.

**I/O Interface**

**MIO Programming**

컨트롤러 Rx / Tx 신호의 각 세트는 MIO 핀 또는 EMIO 인터페이스에 연결됩니다 (표 18-8, 582 페이지 참조).  
일반적인 라우팅(경로) 개념과 MIO I / O 버퍼 구성은 2.4 PS-PL 전압 레벨 쉬프터 활성화 섹션에서 설명합니다. 외부 기준 클록 (CAN\_REF\_CLK)을 사용하기 위한 MIO경로는 18.4.1 클록에서 설명합니다

예 : MIO 핀에 Rx / Tx 신호 구성  
  
1. Rx 신호에 대해 MIO 핀 46을 구성하십시오. slcr.MIO\_PIN\_46 레지스터에 0x0000\_1221을 씁니다.  
a. CAN0 Rx 신호를 46 번 핀에 연결하십시오.  
b. 출력이 비활성화되었습니다 (TRI\_ENABLE = 1로 설정).  
c. LVCMOS18 (다른 전압 옵션의 레지스터 정의 참조).  
d. 느린 CMOS 엣지 (양호한 설정).  
e. 내부 풀업 저항을 활성화하십시오.  
f. Diable HSTL 수신기.  
2. Tx 신호에 대해 MIO 핀 47을 구성하십시오. slcr.MIO\_PIN\_47 레지스터에 0x0000\_1220을 씁니다.  
a. CAN0 Tx 신호를 47 번 핀에 연결하십시오.  
b. CAN에 의해 ​​제어되는 3- 상태 (TRI\_ENABLE = 0).  
c. LVCMOS18 (다른 전압 옵션의 레지스터 정의 참조).  
d. 느린 CMOS 드라이브 엣지.  
e. 내부 풀업 저항을 활성화하십시오.  
f. HSTL 수신기 사용 안 함

**MIO-EMIO Signals MIO-EMIO 신호**

CAN I / O 신호는 표 18-8에 나와 있습니다. 자세한 라우팅(경로) 정보는 2.4 PS-PL 전압 레벨 쉬프터 활성화를 참조하십시오. 디바이스 버전에 기반한 MIO 핀과 제한 사항은 2.5.4 MIO-at-a-Glance 표 섹션의 MIO 테이블에 나와 있습니다.