QUALCOMM® CDMA Technologies

MDM6200 and MDM6600 Mobile Data Modem

User Guide

80-VR001-3 Rev. A

October 13, 2009

Submit technical questions at:
https://support.cdmatech.com

Qualcomm Confidential and Proprietary

QUALCOMM Incorporated
5775 Morehouse Drive
San Diego, CA 92121-1714
U.S.A.

Revision history

Revision	Date	Description
A	October 2009	Initial release

Contents

Figures

Tables

1 Introduction

1.1 Documentation overview

This user addresses the multiple Mobile Data Modem™ (MDM™) devices – MDM6200™ and MDM6600™ devices. They are jointly referred to as the MDM6x00™ device in this document.

Technical information for the MDM6x00™ devices are primarily covered by the documents listed in Table 1-1. All these documents should be studied to gain a thorough understanding of the device and its applications. Released MDM6x00 documents are posted on the CDMA Tech Support website (https://support.cdmatech.com) and are available for download.

Table 1-1 Primary MDM6200 and MDM6600 documentation

Document number	Title/description
80-VR001-1	*MDM6200 and MDM6600 Mobile Data Modem Device Specification (Advanced Information)* Conveys all MDM6x00 device electrical and mechanical specifications. Additional material includes pin assignments; shipping, storage, and handling instructions; PCB mounting guidelines; and part reliability. This document can be used by company purchasing departments to facilitate procurement.
80-VR001-1A	*MDM6200 and MDM6600 Mobile Data Modem Pin Assignment Spreadsheet* An Excel spreadsheet that defines all pin assignment details, including software-supported GPIO configurations.
80-VR001-2	*MDM6200 and MDM6600 Mobile Data Modem Software Interface* Provides detailed information about the MDM6x00 device software interface and its clocks, security, user interface, and registers
80-VR001-3 (this document)	*MDM6200 and MDM6600 Mobile Data Modem Chip User Guide* Provides detailed descriptions of all MDM6200 device functions and interfaces, including its various operating modes.
80-VR001-4	*MDM6200 and MDM6600 Mobile Data Modem Device Revision Guide* Provides a history of MDM6x00 device revisions and changes to its device specification. It explains how to identify the various device revisions, describes known issues (or bugs) for each revision and how to work around them, and lists performance specification changes between each revision of the device specification (80-VR001-1).

The organization of this MDM6x00 user guide is summarized in Table 1-2.

Table 1-2 Chapter summaries

Chapter	Description
Chapter 1	Provides an overview of MDM6x00 IC documentation, gives an introductory functional description of the IC and a typical application, and lists terms and acronyms used throughout this document. Whole chapters (listed below) are dedicated to describing major functional blocks and their interface requirements.
Chapter 2	Defines how to set the basic MDM6x00 operating modes and describes the resulting IC operation.
Chapter 3	Device architecture – detailed functional descriptions of the on-chip processors, buses, and memory that form the IC's digital core, plus other key internal functions and their interfaces (such as boot, security, clocks, and test ports)
Chapter 4	Memory support – memory controller and external bus interface (EBI) functional descriptions and interface requirements
Chapter 5	Air interfaces – functional descriptions of the supported air interfaces
Chapter 6	Connectivity – supported interfaces like USB, UART, UIM, SDIO, PCM, I^2C– functional descriptions and interface requirements
Chapter 7	BB / RF /PM interfaces – how the MDM communicates with other key handset functional blocks like the RF front end, off-chip analog circuits, and power management circuits.
Chapter 8	General-purpose input/output (GPIO) pins – pad structure, configuration options, power-up defaults, multiplexing, and the top-level mode multiplexer (TLMM).
Chapter 9	RF transceivers – functional descriptions and interface requirements for the UMTS and GSM transceivers and the GNSS receiver
Chapter 10	Power and ground – lists of power and ground connections and associated recommendations

1.2 Device variants

The MDM6x00 device variants are primarily defined by the RF operating bands and air interfaces supported by hardware (Table 1-2). The bands supported in a particular application, and the corresponding RF Rx and Tx port assignments, are defined by software.

Table 1-3 MDM6x00 device variants defined by RF support

	CDMA air interfaces				CDMA operating bands											WCDMA air interfaces				WCDMA operating bands								GSM air interfaces			GSM op bands				GNSS	
Device	1x	1x-EVDOr0	1x-EVDOrA	1x-EVDOrB	BC0 - Cellular	BC0 SC3 - JCDMA	BC1 - US PCS	BC3 - JCDMA	BC4 - KPCS	BC5 - 450	BC6 - IMT	BC7 - 700	BC10 - 800	BC14 - PCS	BC15 - AWS	WCDMA Rel 99	HSDPA	HSUPA	HSPA+	B1 - UMTS-2100	B2 - UMTS-1900	B4 - UMTS-AWS	B5 - UMTS-850	B6 - UMTS-800	B8 - UMTS-900	B9 - UMTS-1700	B11 - UMTS-1500	GSM Rel 99	GPRS	EDGE	GSM-850	GSM-900	GSM-1800	GSM-1900	GPS	Glonass
MDM6200																X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
MDM6600	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

A convenient summary of the device variants:

- WCDMA + GSM + GNSS = MDM6200 device
- CDMA + WCDMA + GSM + GNSS = MDM6600 device

NOTE Table 1-1 lists all the bands supported by the MDM6x00 device, but only specific band combinations are supported by each software build. See Section 9.1 for details.

1.3 Example applications

Figure 1-1 shows the MDM6x00 device in an example application. Example RF front-end details is shown in Figure 1-2 and Figure 1-3.

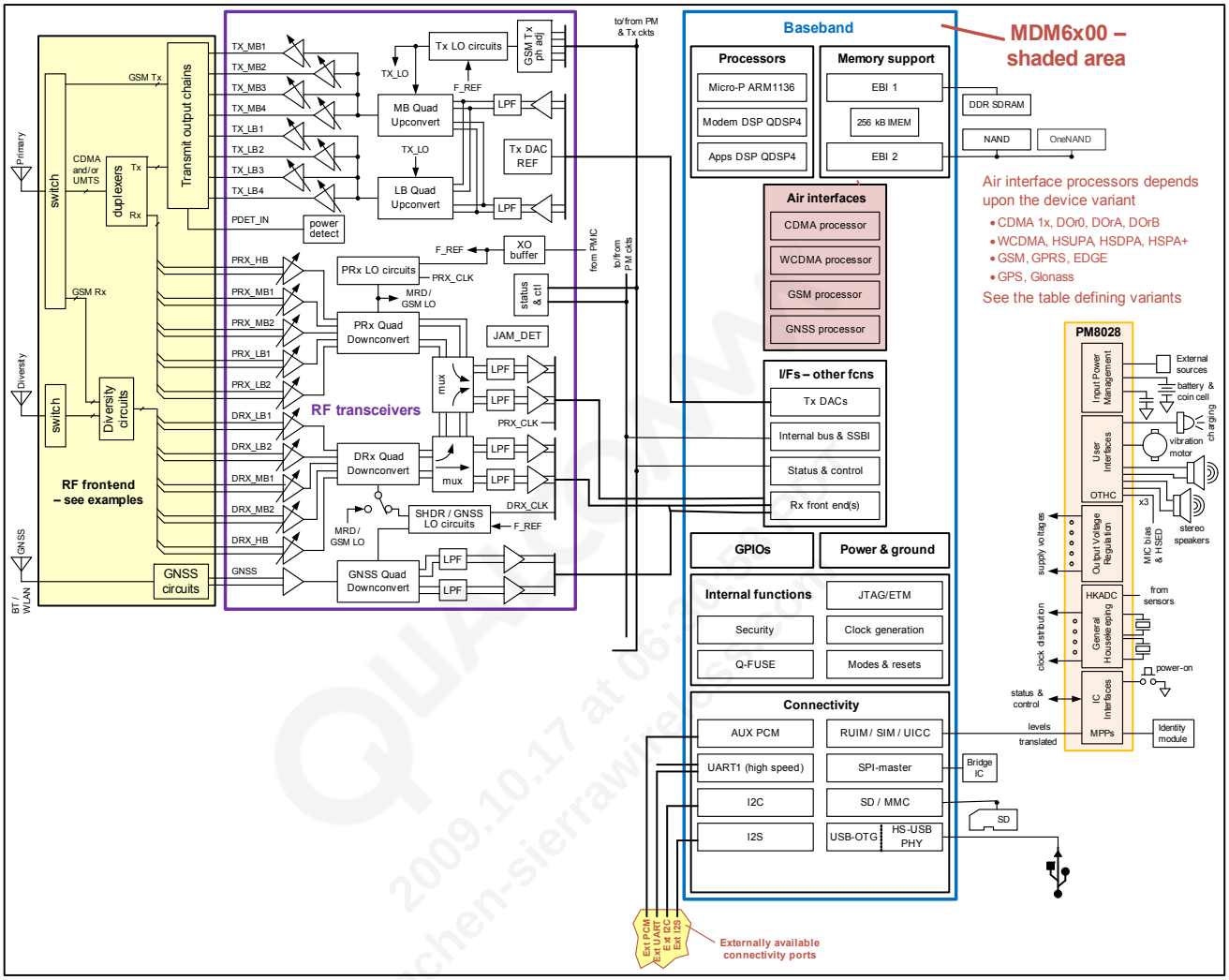


Figure 1-1 MDM6x00 functional block diagram and example application

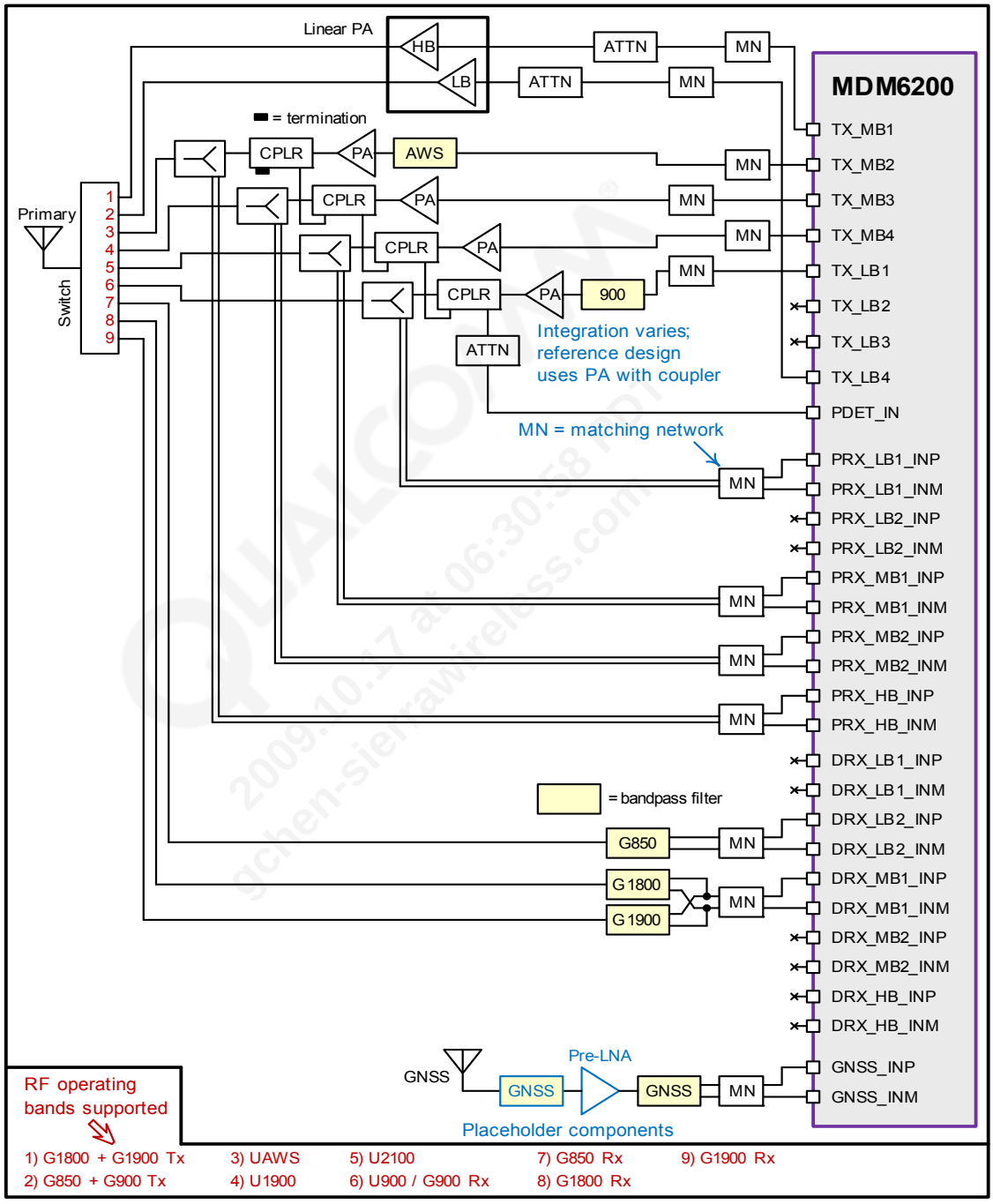


Figure 1-2 RF front-end example 1 – WCDMA + GSM + GNSS (MDM6200 device)

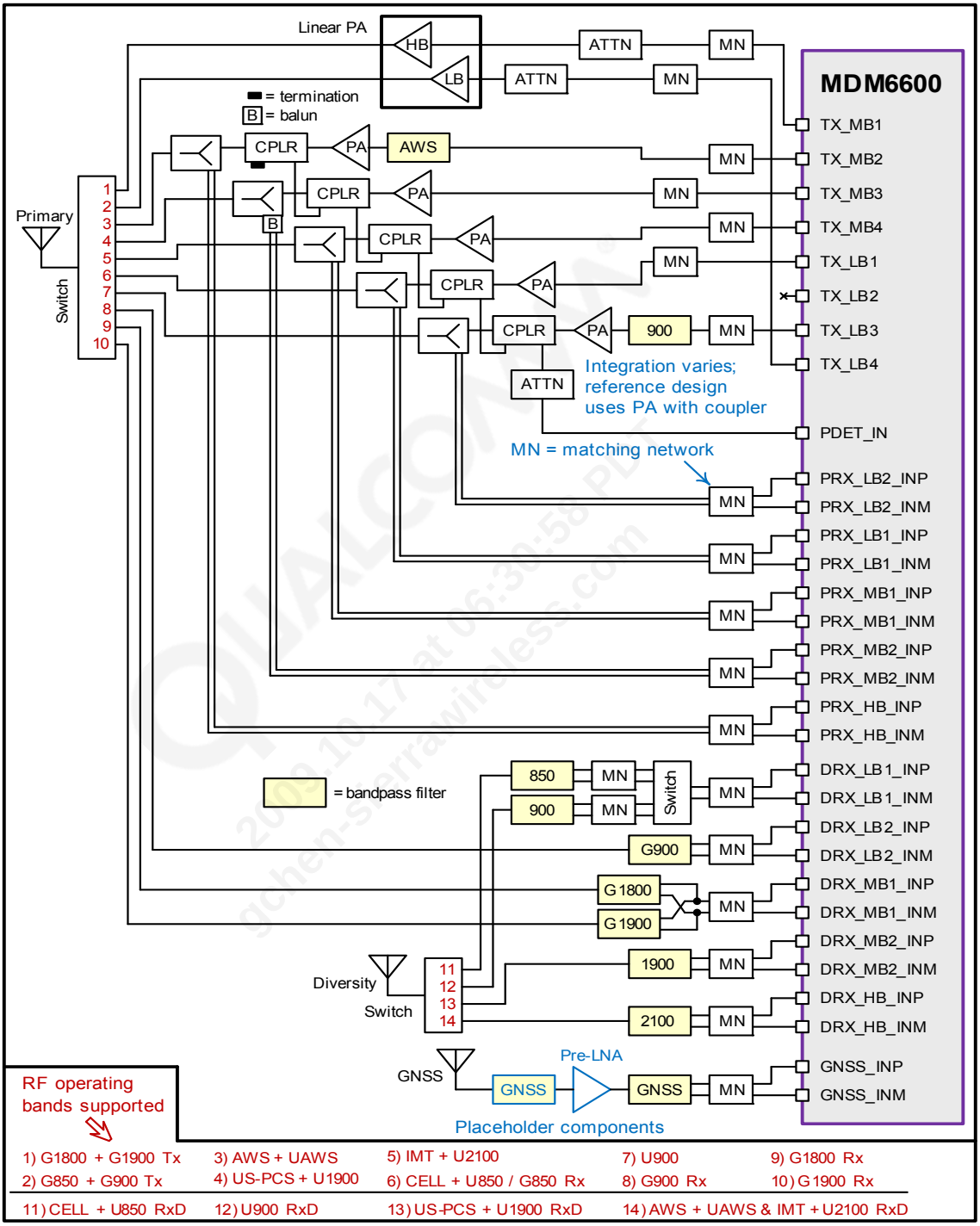


Figure 1-3 RF front-end example 3 – CDMA + WCDMA + GSM + GNSS (MDM6600)

MDM6x00 solution benefits

- Higher integration to reduce PCB surface area, power consumption, time-to-market, and BOM costs while adding capabilities and processing power
  - Baseband functions, including multiple hardware cores
  - radioOne® RF transceiver functions (Rx and Tx, both eliminating their intermediate frequency [IF] components)
  - Rx inter-stage SAW filters are not required.
  - Longer run-time for mobile devices over other industry solutions that use companion processors
- Primary and diversity receive paths are designed for equivalent noise-figure performance.
- Location-based services and applications, including points of interest, personal navigation, and friend finder; Gen 8 (GPS and GLONASS) are supported.
- Ample processing power to support third-party operating systems (OSs)
- HS-USB core with built-in PHY eliminates additional USB components.
- USB-UICC support
- DC power reduction using innovative techniques

1.3.1 Processing power

Processors

The MDM6x00 device integrates three processors:

1. Industry standard ARM1136-JS™ embedded microprocessor system (with ARM® Jazelle™ Java® hardware accelerator)
2. A second QDSP4000 processor as the modem digital signal processor (mDSP)

By removing the need for costly applications coprocessors and memory subsystems, the MDM6x00 solution reduces BOM costs and increases standby and talk times.

SecureMSM™ platform

The MDM device includes support for the SecureMSM™ security solution, thereby enabling the following features:

- Secure boot – protects against reflashing attacks
- Robust protection of subsidy locks and phone identifiers such as the IMEI, ESN, and SIMLock
- Compliance with CMLA and OMTP TR0 specification
- Unchangeable hardware ID unique to each MDM device

The result is a very robust platform securing OMA DRM v1.0 and v2.0 services and e-commerce transactions.

1.3.2 Memory support

Dual-memory buses separate the high-speed memory subsystem (EBI1) from low-speed memory and peripheral devices (EBI2). The example application supports 160 MHz DDR SDRAM memory on EBI1, and adds NAND or OneNAND™ memory via EBI2 (1.8 V devices). EBI2 memory is used for system boot.

1.3.3 Air interfaces

Multimode phones

The MDM6x00 device supports CDMA (1X, 1xEV-DOr0, 1xEV-DOrA, 1xEV-DOrB), WCDMA (Rel '99, HSDPA, HSUPA, HSPA+) and GSM (Rel '99, GPRS, and EDGE). These enable rich wireless multimedia services, such as high-speed upload of multimedia files and attachments, interactive gaming, and a variety of IP-based services such as voice over internet protocol (VoIP). Real-time conversation services such as push-to-talk, video telephony, and instant multimedia (an extension of push-to-talk that combines immediate voice with simultaneous delivery of video and pictures) are also supported.

Very high peak data rates are supported: up to 14.7 Mbps on the CDMA forward link and 5.4 Mbps on the reverse link; up to 14.4 Mbps on the WCDMA downlink and 5.76 Mbps on the uplink. Such high data rates enable network operators to provide an even broader range of wireless multimedia and other data services for consumers and enterprise customers. Data-only (DO), HSDPA, HSUPA, and HSPA+ are optimized for packet data service and provide some of the lowest costs per bit when compared with other wireless wide area network (WWAN) technologies. The MDM6x00 solution integrates powerful applications processors into the Qualcomm market-proven wireless modem, offering increased processing capacity combined with lower power consumption. With this solution, handset manufacturers can design sleek, converged consumer wireless devices that boast the industry's most advanced image quality and resolution to provide enhanced 3D animation, gaming, streaming video, videoconferencing, broadcast reception, and more.

GNSS position location and navigation systems

Gen8 (GPS L1 + GPS L1 wide + GLONASS) is supported.

A global navigation satellite system (GNSS) is a satellite navigation system that provides autonomous geospatial positioning with global coverage. The integrated GNSS functionality allows the handset to determine its location (longitude, latitude, and altitude) to within a few meters using time signals transmitted along a line-of-sight by radio from satellites.

Two examples of a GNSS is the fully functional United States NAVSTAR global positioning system (GPS) and the Russian GLONASS, which is in the process of being restored to full operation. The MDM supports both systems.

GPS consists of up to 32 medium-earth-orbit satellites in six different orbital planes, with the exact number of satellites varying as older satellites are retired and replaced. It is currently the world's most used satellite navigation system. GLObal'naya NAvigatsionnaya Sputnikovaya Sistema (GLONASS) currently has gaps in coverage, but is on its way to full global availability.

The MDM solution merges satellite and network information to provide a high-availability solution that offers industry-leading accuracy and performance. Multiple modes are supported:

- Mobile-assisted mode – When a request for position location is issued, the available network information is provided to the location server (cell-ID), and assistance is requested from the location server. The location server sends the assistance information to the handset. The handset/mobile unit measures the satellite observables and provides those measurements along with available network data (that is appropriate for the given air interface technology) to the location server. The location server then calculates the position location and returns the results to the requesting entity.
- Mobile-based mode – The assistance data provided by the location server encompasses not only the information required to assist the handset in measuring the satellite signals, but also the information required to calculate the handset's position. Therefore, rather than provide the satellite measurements and available network data back to the location server, the mobile calculates the location on the handset and passes the results to the requesting entity.
- Standalone (autonomous) mode – The handset demodulates the data directly from the satellites. This mode has some reduced cold-start sensitivity and a longer time to first fix when compared to the assisted modes. However, it requires no server interaction and works in out-of-network coverage situations.

This combination of satellite measurements and available network information provides:

- A high-sensitivity solution that works in all terrains: indoor, outdoor, urban, and rural
- High availability that is enabled by using both satellite and network information

Therefore, while network solutions typically perform poorly in rural areas and areas of poor cell geometry/density, and while unassisted, satellite-only solutions typically perform poorly indoors, the Qualcomm gpsOne® solution provides optimal time-to-fix, accuracy, sensitivity, availability, and reduced network utilization in both of these environments, depending on the given condition.

Additional information about Qualcomm GNSS solutions and implementations is available in the *radioOne Solutions for GPS Position Location Application Note* (80-V1647-1).

1.3.4 Connectivity

In addition to the basic handset-control interfaces like the keypad, the MDM device supports a variety of consumer electronics with its connectivity ports:

- Full-speed universal serial bus (only via HS-USB)
- High-speed USB on-the-go (USB-OTG) with built-in physical layer (PHY)
- USB universal integrated circuit card (USB-UICC)
- Universal asynchronous receiver transmitter (UART) serial ports. Each GSBI port has a dedicated UART making the total UARTs available for the GSBIs ports equal 5
- UMTS subscriber identity module (USIM); 1.8 V support (3.3 V support with PMIC-level translators)
- Three integrated secure digital (SD) controllers (two at 1.8 V, one at 2.85 V)
- Pulse-coded modulation (PCM) interface

- Inter-integrated circuit (I^2C) interface for peripheral devices
- Inter-IC sound (I^2S) interface for digital audio
- Serial peripheral interface (SPI)

Various combinations of connectivity ports can be used to support features like near-field communicator (NFC) – with the addition of external devices.

1.3.5 RF transceivers

The MDM RF transceiver functions are outlined in purple within Figure 1-1.

radioOne technology

The MDM6x00 device incorporates Qualcomm's radioOne technology – the technology for RF transceivers that converts received signals directly from RF to baseband and transmit signals directly from baseband to RF (also known as direct conversion or zero intermediate frequency [ZIF] processing). The LO sources and distribution circuits are also integrated on-chip, enabling network compatibility around the world while simplifying parts procurement.

Supported operating bands

The MDM transceiver circuits are leveraged from Qualcomm's latest generation RF technology to support CDMA, WCDMA, and GSM operation in the RF bands identified within Table 1-2. Note that the air interface technology and operating bands supported depends upon the MDM device variant.

In addition, position location and navigation is supported by a dedicated receiver operating within the GPS and GLONASS L1 bands.

RF transmitters

A single analog baseband Tx input is shared by all the MDM transmit paths. Two quadrature upconverters, one dedicated for each band category (low band and high band), translate the Tx waveform directly to the desired RF band. The RF signal is then routed to the appropriate driver amplifier output. There are four outputs for each band category, for a total of eight Tx outputs.

See Section 9.3 for additional RF transmitter details.

RF receivers

During CDMA operation, the MDM primary, diversity, and GNSS receiver paths enable advanced techniques such as mobile receive diversity (MRD), simultaneous hybrid dual receiver (SHDR), full-time SHDR (FTS), and simultaneous GNSS operation. Standalone GPS is supported regardless of the airlink mode. CDMA operation also uses Qualcomm's IntelliCeiver™ technology to minimize power dissipation while achieving adequate linearity for changing jammer conditions. IntelliCeiver functionality is incorporated on the primary receive path.

There are eleven RF input ports, all using a differential configuration to maximize common-mode rejection, Tx isolation, out-of-band suppression, and second-order intermodulation performance, thereby eliminating the need for inter-stage SAW filters. There are three quadrature

downconverters and three separate interfaces to the baseband circuits – primary, diversity, and GNSS. Although GSM signals can use DRx paths, the PRx LO is used and the diversity baseband signals are routed to the PRx output port to allow the primary baseband Rx circuits to process the GSM signals even when using DRx paths. A dedicated signal path is provided for GNSS, but its downconverter shares a PLL with the DRx downconverter; this means SHDR and GNSS operation cannot occur simultaneously.

See Section 9.6 for additional RF receiver details.

1.3.6 Power management

The MDM6x00 device is powered by a Qualcomm powerOne™ series PM8028™ power management IC. The power source is selected from either the battery, external charger, adaptor, or USB device, and then the IC generates all the regulated voltages needed to power the handset electronics. The main battery and backup coin-cell voltages are monitored, and charging is conducted as needed.

In addition to these basic power functions, the PMIC also provides:

- Housekeeping functions
  - Analog multiplexer and ADC that allows monitoring of IC-level and handset-level sensors, including the XO thermal status
  - System clocks, including the MDM IC's 32 kHz sleep clock and 19.2 MHz operating clock
  - Real-time clock and associated alarms
- User interfaces
  - Current drivers for an auto trickle-charge indicator LED
  - Pulse generator output for controlling external LED and backlight drivers
  - Vibration motor driver for silent incoming call alerts
  - Two-channel (stereo, 500 mW per channel) speaker driver for far-field speakers
  - Headset send/end detection and microphone biasing
- IC-level interfaces
  - Monitoring of triggering events and coordination of the power-on and power-off sequences
  - Level translators for dual-voltage UIM support
  - PA controllers

See the PM8028 document set (80-VN204-x) for more details.

1.3.7 Software

Qualcomm provides a complete software suite that extends handset capabilities and expedites product development.

Advance mobile subscriber software (AMSS™)

Many hardware features and operational parameters are enabled by the proper versions of AMSS software. This software is designed to run on a subscriber unit reference (SURF™) phone platform, an optional development platform used to evaluate, test, and debug AMSS software.

Launchpad™ Suite

The Launchpad Suite of integrated applications offers wireless operators and manufacturers a cost-effective, scalable, and timely solution for providing advanced wireless data services. This seamlessly integrated solution enables advanced next-generation applications and services that incorporate multimedia, position location, connectivity, customized user interface, and storage capabilities. Launchpad features are available for each Qualcomm chipset, closely matching the specific functionality and cost-target objectives agreed upon in joint product planning with manufacturers and wireless service operators worldwide.

The MDM6x00 solution supports Launchpad's advanced feature set, including streaming video and audio, still-image and video encoding and decoding, and an 8-megapixel camera interface. This solution also integrates GNSS functionality to support assisted and standalone operation – enabling a wide variety of location-based services and applications, including points of interest, personal navigation, and friend finder.

Seamless communication directly with printers, digital cameras, keyboards, and other accessories is also supported via the integrated wireless Bluetooth and USB controller functions.

BREW Mobile Platform

The BREW® Mobile Platform (Brew MP) design allows manufacturers to deliver differentiated devices to the market through their control over the platform and user experience. This provides the power and control of a proprietary platform with the features, support and *open* OS access.

Brew MP provides an extensive set of APIs into features at all layers of the device. Additionally, new features and APIs can be added without Qualcomm's involvement.

Most of the features and customizations to the platform are implemented as:

- Digitally signed binary modules (modular and secure)
- Demand-loaded (minimizes RAM)
- Language-abstracted APIs (C/C++, Flash, Lua, and Trig)

Brew MP is optimized to leverage chipset modem and multimedia services. These services are exposed through immutable object-oriented APIs that support seamless access to hardware-accelerated implementations. The platform supports both cooperative and preemptive threading. It also supports process-based memory protection that complements the platforms robust, least-privileged execution model.

Refer to the BREW Mobile Platform website for more details:
https://brewmobileplatform.qualcomm.com/devnet/index.jsp

1.4 Terms and acronyms

Table 1-4 defines terms and acronyms commonly used throughout this document.

Table 1-4 Terms and acronyms

Term	Definition
AAD	Address address data
AMSS	Advanced mobile subscriber software
ADC	Analog-to-digital converter
AEC	Acoustic echo cancellation
AGC	Automatic gain control
BER	Bit error rate
BPF	Bandpass filter
bps	Bits per second
BT	Bluetooth
CAGC	CDMA AGC
CELP	Code excited linear prediction
CMX	Compact Media Extension
Codec	Coder decoder
CSP	Chip scale package
CRC	Cyclic redundancy code
DDR	Dual data rate
DRM	Digital rights management
DSP	Digital signal processor
DTMF	Dual-tone multiple frequency
EBI	External bus interface
ETM	Embedded trace macrocell
EVRC	Enhanced variable rate codec
GNSS	Global navigation satellite system
GPIO	General-purpose input/output
HPF	Highpass filter
HSDPA	High speed downlink packet access
IF	Intermediate frequency
JPEG	Joint Photographic Experts Group
JTAG	Joint Test Action Group (ANSI/ICEEE Std. 1149.1-1990)
kbps	Kilobits per second

Table 1-4 Terms and acronyms (cont.)

Term	Definition
LCD	Liquid crystal display
LNA	Low noise amplifier
LPF	Lowpass filter
LSBit or LSByte	Defines whether the LSB is the least significant bit or least significant byte. All instances of LSB used in this manual are assumed to be LSByte, unless otherwise specified.
MMC	Multimedia card
MPEG	Motion picture experts group
MSBit or MSByte	Defines whether the MSB is the most significant bit or most significant byte. All instances of MSB used in this manual are assumed to be MSByte, unless otherwise specified.
MSM™	Mobile Station Modem™
OMA	Open Mobile Alliance
PA	Power amplifier
PCM	Pulse-coded modulation
PDM	Pulse-density modulation
PSRAM	Pseudo-static random access memory
QCELP®	Qualcomm Code Excited Linear Prediction
QCIF	Quarter common intermediate format (176 pixels/line, 144 lines/frame)
QLIC	Quasi-linear interference cancellation
RC	Resistance-capacitance
RF	Radio frequency
RMS	Root mean squared
RUIM	Removable user interface module
Rx	Receive
SDAC	Stereo digital-to-analog converter
SDRAM	Synchronous dynamic random access memory
SoC	System-on-Chip
SPI	Serial peripheral interface
Sps	Symbols per second (or samples per second)
TAP	Test access port
TCXO	Temperature-compensated crystal oscillator
Tx	Transmit
UART	Universal asynchronous receiver transmitter
UICC	Universal integrated circuit card
USB	Universal serial bus
USB-OTG	Universal serial bus on-the-go
USIM	UMTS subscriber interface module

Table 1-4 Terms and acronyms (cont.)

Term	Definition
WCDMA	Wideband code division multiple access
XO	Crystal oscillator
ZIF	Zero intermediate frequency

1.5 Special marks

Table 1-5 defines special marks used in this document.

Table 1-5 Special marks

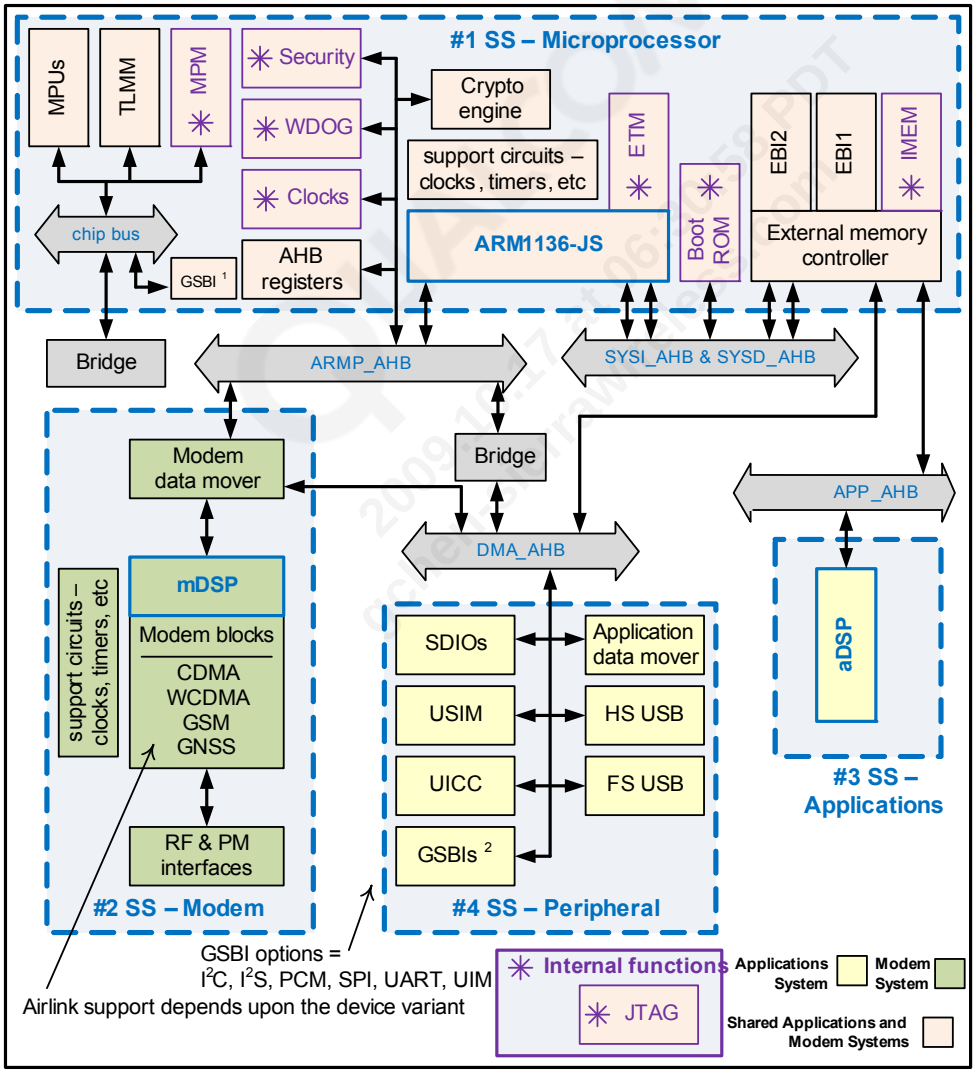
Mark	Definition
[ ]	Brackets ([ ]) sometimes follow a pin, register, or bit name. These brackets enclose a range of numbers. For example, GPIO_INT[7:0] may indicate a range that is 8 bits in length, or DATA[7:0] may refer to all eight DATA pins.
_N	A suffix of _N indicates an active low signal. For example, RESIN_N.
0x0000	Hexadecimal numbers are identified with an x in the number (for example, 0x0000). All numbers are decimal (base 10) unless otherwise specified. Non-obvious binary numbers have the term binary enclosed in parentheses at the end of the number, for example, 0011 (binary).
|	A vertical bar in the outside margin of a page indicates that a change was made since the previous revision of this document.

2 Device Operating Modes

This information will be included in future revisions of this document.

3 Device Architecture

This chapter includes the on-chip processors, buses, and internal memory that form the IC's digital core, plus other key internal functions and their interfaces (boot, security, clocks, test ports, etc.). The MDM6x00 device architecture is shown in Figure 3-1.



GSBI notes:
1. I²S/PCM GSBI cores are configured by the ARM processor through the chip bus.
2. Data movement of I²S/PCM is done through the DMA_AHB bus. Configuration and data movement of I²C, SPI, and UART are also done through the DMA_AHB bus.

Figure 3-1 MDM6x00 device architecture

The architecture includes two major systems comprised of five subsystems:

- Shared between the modem and applications systems (shared blocks are highlighted in pink)
  - #1 subsystem – the microprocessor subsystem
- Modem system (its blocks are highlighted in green)
  - #2 subsystem – the modem subsystem
- Applications system (its blocks are highlighted in yellow)
  - #3 subsystem – the applications subsystem
  - #4 subsystem – the peripherals subsystem
  - #5 subsystem – the display subsystem

The MDM hardware function described within this chapter is assigned to one of these five subsystems in order to match the software organization defined within the *MDM6200 and MDM6600 Mobile Data Modem Software Interface* (80-VR001-2).

Both systems are supported by six internal advanced high-performance buses:

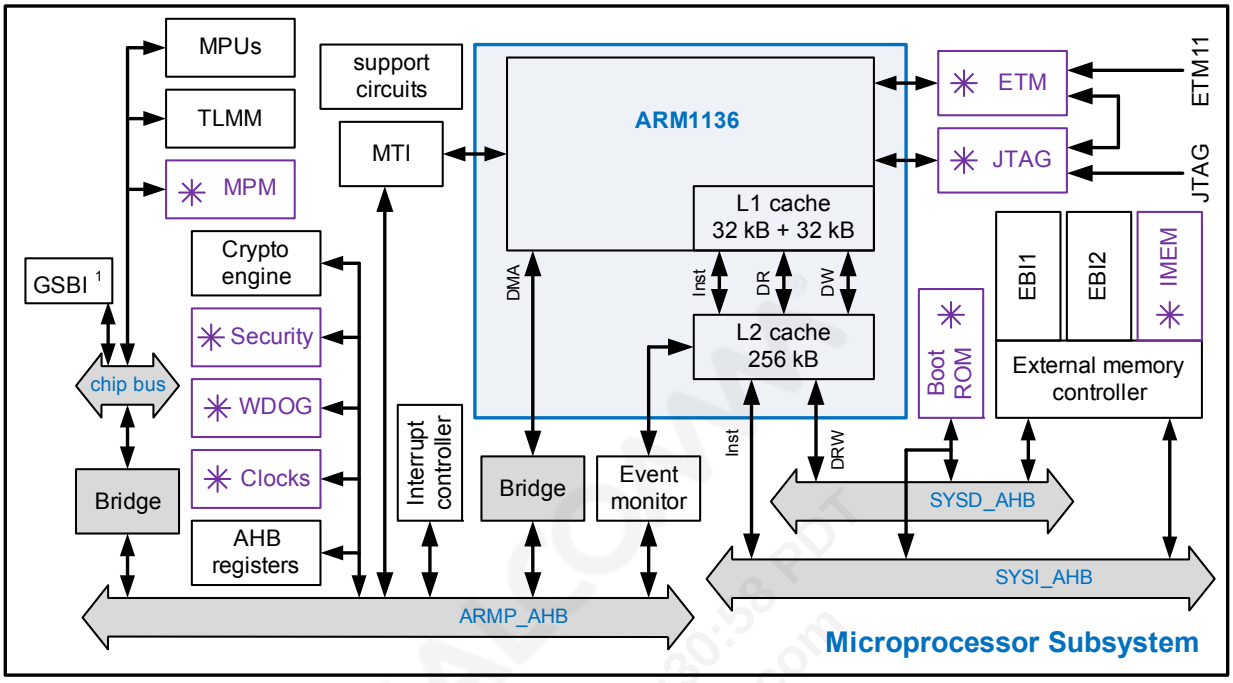
- ARMP_AHB – ARM peripheral bus
- SYSI_AHB – system instruction bus
- SYSD_AHB – system data bus
- APP_AHB – application bus
- DMA_AHB – auxiliary bus for DMA transfers
- APP-IMEM_AHB – unused
- Plus local interconnects and bridges distributed throughout the IC

This architecture chapter provides details about two functional blocks within the IC's top-level block diagram (Figure 1-1) presented in Chapter 1:

1. Processors – which includes both major systems and the internal buses.
2. Internal functions – internal memory, security (including Qfuses and boot), clock generation and distribution, power optimization, and test ports (JTAG/ETM) – these functions are outlined in purple within Figure 3-1.

3.1 Microprocessor subsystem

The MDM6x00 device is a single-microprocessor device, so the ARM1136-JS processor and its subsystem (Figure 3-2) support both the modem and applications systems. It provides floating-point capability, including the execution of real-time, third-party operating systems, and multimedia applications.



GSBI notes:
1. I²S/PCM GSBI cores are configured by the ARM processor through the chip bus .

Figure 3-2 Microprocessor subsystem functional block diagram

The microprocessor subsystem includes processor-specific blocks that are described in the following subsections, plus other blocks that are addressed elsewhere:

- Security and related blocks (MPUs, crypto engine, and boot ROM) – Section 3.6
- Clock generation and distribution – Section 3.7
- GPIOs, GSBIs, and TLMM – Chapter 8
- Memory blocks – Section 3.5 and Chapter 4
- Test interfaces (ETM and JTAG) – Section 3.9
- Buses – Section 3.4

3.1.1 ARM1136-JS processor

The ARM1136-JS processor is a single-clock, high-performance, Java-enabled, synthesizable core. It includes an eight-stage pipelined RISC architecture, supporting both 32-bit ARM and 16-bit Thumb instruction sets, a 32-bit address bus, and a 32-bit internal data bus. Not all ARM1136-JS features are used in the MDM device.

Key improvements over previous-generation products include:

- Microprocessor core with L2 cache controller
- Separate 64-bit instruction and data bus interfaces (SYSI_AHB and SYSD_AHB)
- All buses include grant-table programmability-priority arbiters to allow more flexibility in priority and bandwidth assignments.

3.1.1.1 Reset and pause

The microprocessor includes reset and pause functions (Figure 3-3) that are responsible for:

- Generating synchronous reset signals for modem processing and the other bus architecture peripherals.
- Providing status information concerning the source of the last reset (pin-generated or watchdog-generated).
- Controlling the clocks, thereby providing reset or pause mechanisms for modem processing.
- Allowing software to control this pause mechanism to halt the ARM for a specified duration.

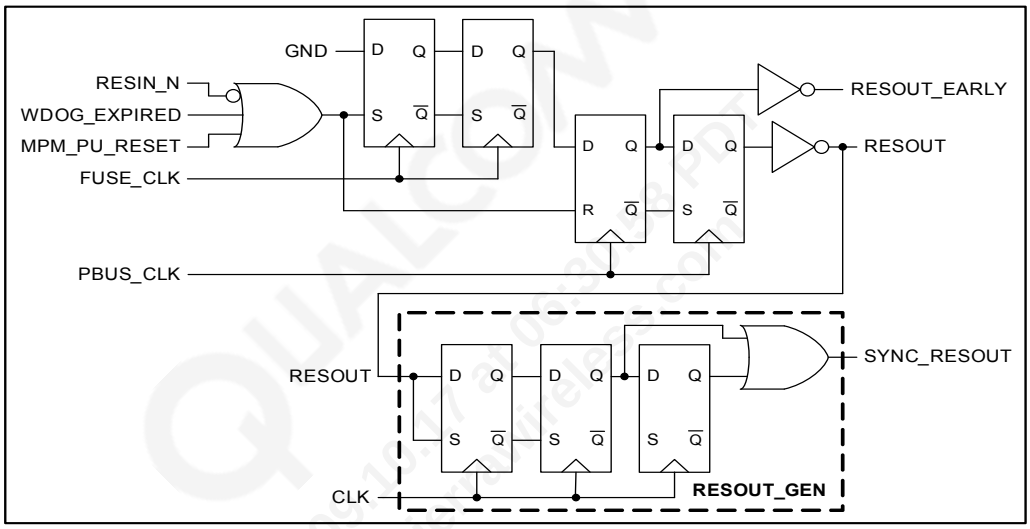


Figure 3-3 Reset generation

3.1.1.2 Watchdog timer

Watchdog timer characteristics:

- 14-bit counter running off the sleep-controller clock regime
  - Enables the handset to recover from unexpected hardware or software anomalies
  - If the microprocessor fails to periodically reset the watchdog timer, the timer expires and the handset is reset.
- Disabled and/or reset by grounding the WDOG_EN pin or by asserting the RESIN_N pin
  - Hardware disabling example: The ARM processor enters debug mode (indicated by the DBGACK pin of the processor transitioning from low to high)
- Enabled by leaving the WDOG_EN pin unconnected (it has an internal pull-up) or by connecting it to VDD
- If temporarily disabled, it can be re-enabled by applying a system reset that clears the state of the internal wdog_disable signal

When enabled, the watchdog circuit pulses the WATCHDOG_EXPIRED signal whenever its timer has expired. In NATIVE mode, this signal is combined with the RESIN_N pin to generate the RESOUT_N signals and the MDM device's internal reset.

3.1.1.3 Sleep mode

The microprocessor cannot reset the watchdog timer while in its sleep mode, but the sleep controller includes an auto-kicker circuit that resets the watchdog timer every 32 sleep-clock cycles. This circuit is enabled when the microprocessor writes a particular bit sequence to the AUTOKICK_START: AUTO_KICK_START bit. When the sleep counter expires, the auto-kicker is disabled.

When the sleep crystal is not being used (SLEEP_XTAL_EN = 0), the watchdog timer is disabled.

3.1.1.4 Non-sleep mode

In non-sleep mode, the microprocessor must reset the watchdog timer at least once every 213 ms [1]. If not, the watchdog timer expires and asserts an internal RESIN_N signal to the system. The WDOG_EXPIRED signal lasts about 100 ms [1]. To reset the watchdog timer, the microprocessor must write a particular bit sequence to the WDOG_RESET:WATCH_DOG bit.

3.1.2 Vectored interrupt controller (VIC)

The vectored interrupt controller prioritizes all pending sources and provides a vector to the ARM processor having the highest-priority enabled interrupt.

- If the ARM processor is not currently processing an interrupt service routine (ISR), then the vector availability is indicated with an interrupt request (IRQ) or fast interrupt request (FIQ).
- If the ARM processor is in the process of an ISR, then any new IRQ is issued only for a higher-priority enabled ISR.

FIQ sources are not prioritized in hardware, and any enabled source results in an FIQ_N indication to the processor. Tracking the nesting of stacked interrupts in hardware allows software to re-enable IRQs while processing a current ISR, thereby allowing higher-priority IRQs to be serviced without waiting to finish lower-priority ISRs.

When comparing the VIC to a prioritized interrupt controller (PIC):

- The PIC and the VIC both prioritize IRQs in hardware and provide a vector along with the highest-priority IRQ.
- When the vector presented to the ARM core is the index number of the associated interrupt source, the interrupt controller (IRC) is referred to as a PIC.
- When the vector is the full-ISR address that the ARM processor can jump to immediately, the interrupt controller is referred to as a VIC.

1. Stated times assume a 32.768 kHz sleep clock.

Features of the VIC include:

- Eight programmable priority levels for interrupt sources
- The interrupt vector of the highest-priority source can be returned through AHB reads to the VICIRQ_VEC_RD or VICIRQ_VEC_PEND_RD registers.
- Allows for nested interrupt services
- Round-robin dispatch scheme for same-level sources
- Maximum delay of 6 HCLK cycles from the assertion of the interrupt sources to the assertion of the IRQ
- Combinational paths from interrupt sources to the clock controller for initiating an exit from the clock-halted state
- Support for a NO_NESTING_MODE (plain IRC)
- Supports both edge-triggered and level-triggered interrupt sources
- Configuration of the interrupt controller is accomplished over the P_AHB bus and includes the following features:
  - Marking of interrupt sources as either IRQ or FIQ
  - Detection of interrupt sources using either level or edge sensing
  - Priority-level specification of each IRQ marked interrupt source
  - Enables masking of each interrupt source
  - Refer to the *MDM6200 and MDM6600 Mobile Data Modem Software Interface* (80-VR001-2) for more details.

3.1.3 Software interface options for interrupts

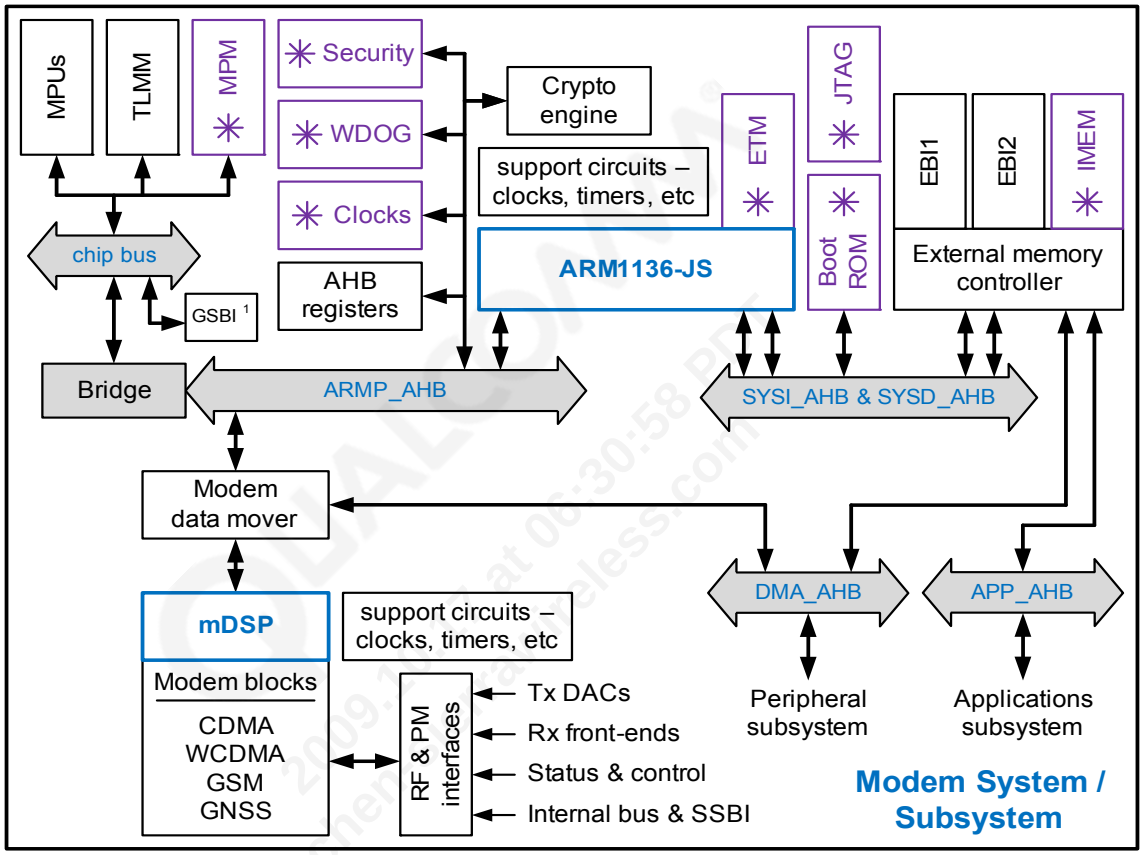
This information will be included in a future revision of this document.

3.2 Modem system / subsystem

The modem subsystem (Figure 3-4) includes all blocks that provide modem functionality directly (such as airlink modulation and demodulation, and searching), plus supporting functions such as:

- An ARM1136-JS processor as the main modem processor.
- Connections to key functions such as the crypto engine and EBI via AHB buses.
- The modem DSP (mDSP) that controls modem hardware blocks and assists with symbol-level processing of physical layer traffic. All modem functions require mDSP support to operate. It has a limited amount of processing power and program/data storage; it must meet the sum of all modem MIPS and memory requirements (plus some overhead) for concurrent operation.
- The modem data mover (MDM) – a direct memory access (DMA) engine that can drive several AHB buses.
- The modem cores – airlink processing capabilities that include CDMA, WCDMA, GSM, and GNSS.

- An always-on modem power manager (MPM) that controls power collapsing of the modem.
- Memory protection units (MPUs) that prevent certain bus masters from accessing sensitive memory regions.
- More microprocessor subsystem blocks that are cross-referenced within Section 3.1



GSBI notes:
1. I^2S/PCM GSBI cores are configured by the ARM processor through the chip bus .

Figure 3-4 Modem subsystem functional block diagram

3.2.1 Modem subsystem bus interfaces

Bus interfaces provide the necessary connections to other functions:

- The ARM1136-JS modem processor is accessed via the 32-bit ARMP_AHB.
- The peripheral and applications subsystems, and the external memory controller, are accessed via the DMA_AHB and APP_AHB buses for internal and external communications.
- Local AHB buses bridge to asynchronous buses to communicate with the modem hardware blocks, and a slow-speed request/acknowledge-based bus is used to configure the modem cores.

See Section 3.4 for more bus information.

3.2.2 Modem processor

The ARM1136-JS processor supports both the modem and applications systems; microprocessor information is presented in Section 3.1.

3.2.3 Modem data mover (MDM)

The MDM engine is a general-purpose DMA engine used to transfer data internally and to external memory. It is connected to the APP_AHB, ARMP_AHB, and DMA_AHB buses. ARMP_AHB is the host interface that is used to configure the MDM. Transfers can be initiated through any of these three buses for both read and write.

3.2.4 Modem DSP (mDSP)

One of the integrated QDSP4000 cores is dedicated to modem functions and airlink processing (CDMA, WCDMA, GSM, GNSS – in their various modes). This DSP is capable of running at up to 147.5 MHz.

3.2.5 Modem core

The modem core performs the processing necessary to support the air interfaces and features.

3.3 Peripheral system

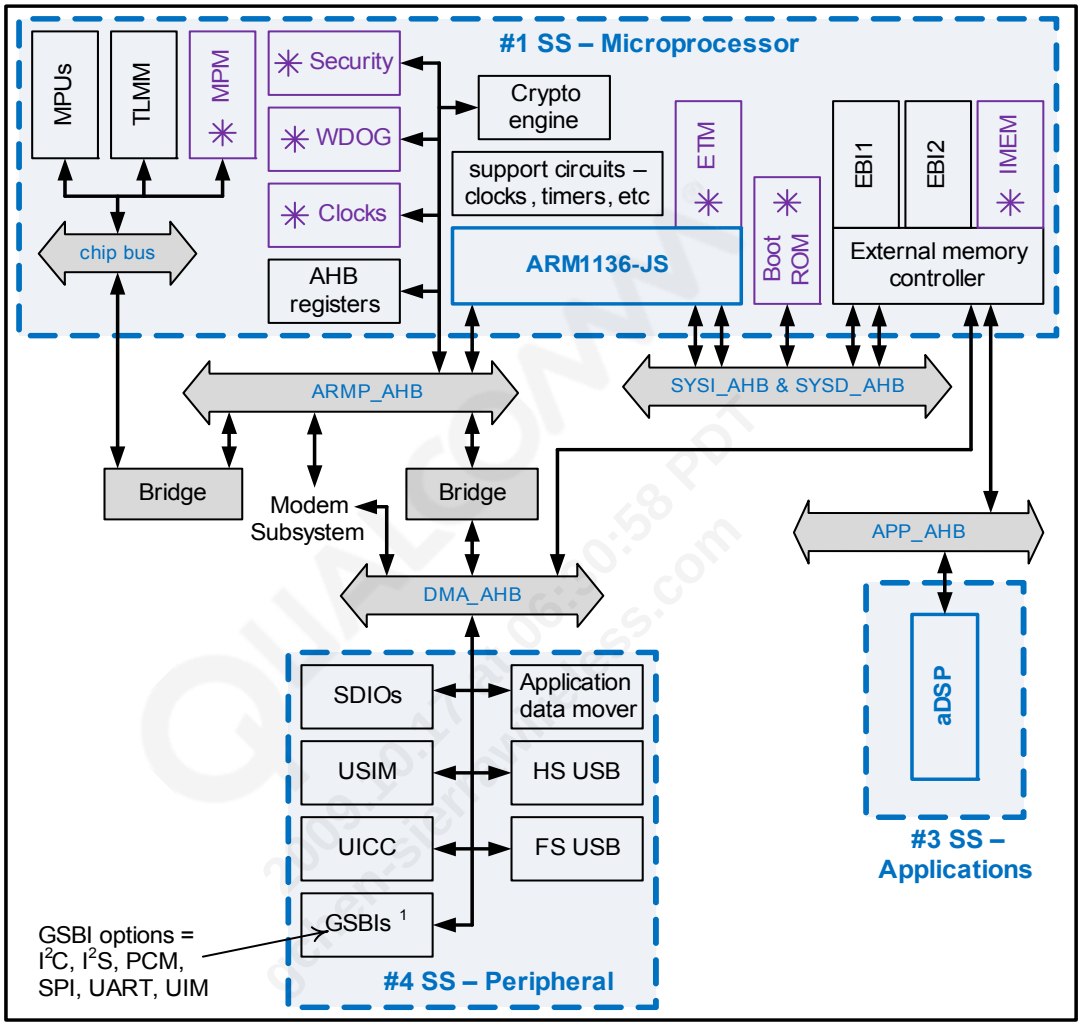
The MDM6x00 peripheral system consists of the following subsystems and major functional blocks (Figure 3-5):

- The ARM1136-JS microprocessor subsystem (shared with the modem) with supporting watchdog, clock, and timers (Section 3.1)
- The applications subsystem (Section 3.3.1) that includes aDSP
- A dedicated applications data mover (ADM) that resides within the peripheral system
- Other peripheral subsystem blocks including SDIOs, USIM, UICC, HS-USB, FS-USB, keypad interface, and GSBIs that can be configured for I^2C, I^2S, PCM, SPI, UART, and UIM operation. These functions are described within the connectivity chapter (Chapter 6).
- Access to the APP_AHB and APP-IMEM_AHB buses that connect the applications subsystem to the others

The applications system includes the aDSP bus masters. The buses are described in Section 3.4, but a few comments about the applications system masters are highlighted here:

- The ADM engine supports up to 16 channels. Under control of the ARM, it handles data transfers between EBI1 and other peripherals as well as external memory controller, SD controller, and USB RLP data transfers.
- The data mover is used for byte alignment/packing of USB RLP data and putting it back into system memory, and then the USB DMA master transfers the byte-aligned data from system

memory. The USB AHB slave interface is on the peripheral bus and is used for configuring the USB registers.



GSBI notes:
1. Data movement of I^2S/PCM is done through the DMA_AHB bus. Configuration and data movement of I^2C UART are also done through the DMA_AHB bus.

Figure 3-5 Peripheral system functional block diagram

3.3.1 Applications subsystem

Two functional blocks within the applications subsystem are described here: the digital signal processor and the data mover. Its other functional blocks are described in other chapters:

3.3.1.1 Applications DSP (aDSP)

The applications subsystem uses the applications DSP (aDSP) block located within the peripheral subsystem. A dedicated QDSP4000 processor running at up to 162 MHz is used to support applications such as vocoder, video, and graphics.

3.3.1.2 Applications data mover (ADM)

The ADM is a multichannel DMA engine capable of direct, scatter/gather, and box-mode data transfers. It executes programming structures placed by one or several hosts in memory. Each programming structure may include several lists of commands, where each command specifies certain types of data transfers between memory-mapped ADM clients. The ARM processor controls the ADM to enable transfers between EBI1 memory and peripheral devices.

3.4 Advanced high-performance bus (AHB) system

The bus connections are shown in Figure 3-6, with all bus masters highlighted in red. Brief descriptions of each bus are given below:

- SYSI_AHB – system instruction AHB, 64 bits wide; dedicated to ARM11™ instruction fetch operations.
- SYSD_AHB – system data read/write AHB, 64 bits wide; dedicated to ARM11 data read/write operations.
- ARMP_AHB – ARM11 peripheral port AHB, 32 bits wide; mainly used to perform configuration operations of various cores. Also includes the chip-bus bridge as a slave. This slow-speed bridge interfaces the AHB to multiple slow-speed slave interfaces that work on chip-bus protocol.
- DMA_AHB – direct memory access AHB, 32 bits wide; primarily performs DMA operations without ARM intervention, connecting the ARMP_AHB masters (through bridges) with various chip peripherals for data transfer to the peripherals.
- APP-IMEM_AHB – application internal memory AHB, 64 bits wide; allows accesses to IMEM through the external memory controller.
- APP_AHB – application AHB, 32 bits wide; handles data traffic to and from external EBI1 memory through the external memory controller from and to aDSP.

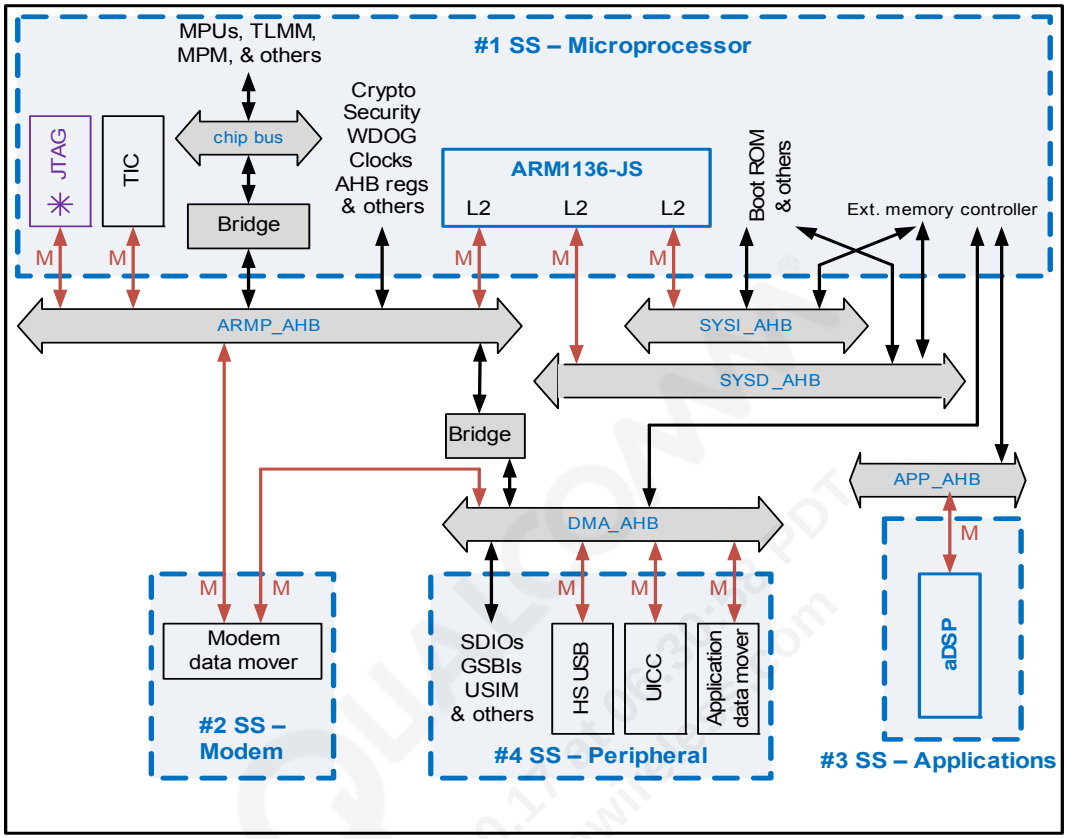


Figure 3-6 Bus connection diagram

Table 3-1 AHB bus master summary

Bus	BM #	Bus master (BM)
SYSI_AHB	1	ARM11 instruction
SYSD_AHB	2	ARM11 D-RW port
ARMP_AHB	3	ARM11 peripheral port (default)
	4	TIC
	5	MDM2
	6	JTAG2AHB
	7	HBFC
DMA_AHB	8	AHB2AHB bridge
	9	MDM2
	10	BPM
	11	USB (default)
	12	UICC
	13	ADM2
APPI_AHB	Unused	
APP_AHB	14	SW1

Table 3-1 AHB bus master summary

Bus	BM #	Bus master (BM)
	15	ADSP
	16	MDM2

For more AHB information than what is provided here, refer to the *TBD document*.

3.5 Internal memory

In addition to the boot ROM within the microprocessor subsystem, the MDM device includes one 256 kB internal memory (IMEM) aDSP, and various internal functions. This memory is random access, so it is also known as IRAM (internal RAM). It is divided into two 128 kB custom memories (Figure 3-7).

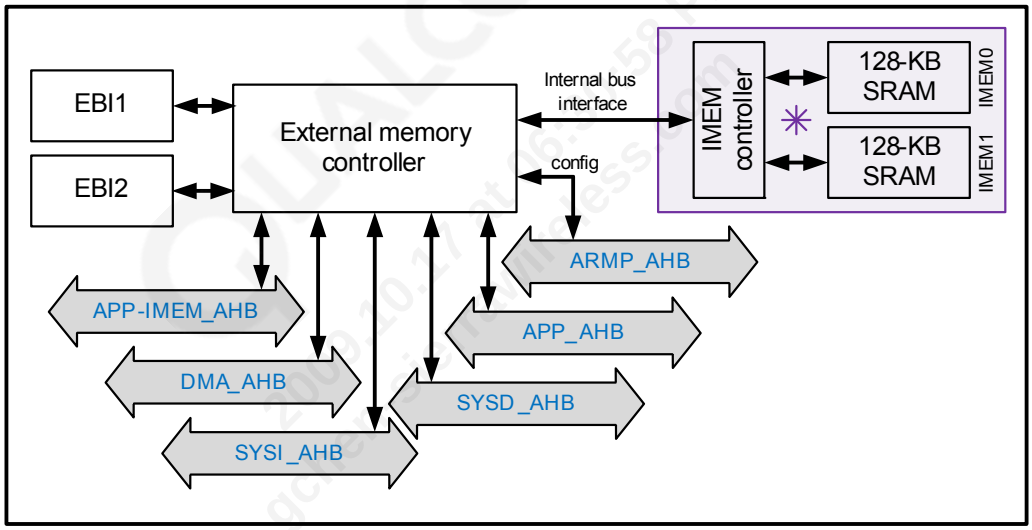


Figure 3-7 Internal memory data flow

An internal bus interface (IBI) is used to access the internal memory. Each memory macro can be accessed independently; two masters on different ports can access two macros in parallel. For example, the ARM can read instructions from IMEM0 via SYSI_AHB, while the aDSP can write to IMEM1 via APP_AHB – both in the same clock cycle.

The IMEM controller translates data requests of various sizes to the data width of IMEM.

Benefits of using the internal IMEM rather than external memory include:

- Reduced power due to saved cycles accessing external memory.
- Improved performance, since the internal memory can be accessed with low latency.

3.6 Security (including Qfuses and boot)

Security goals

1. Enforce protection of modem operation – make it sufficiently hard for an attacker to launch a successful software-based or physical attack that would gain any degree of control over modem functionality.
2. Enable secure implementation of digital rights management (DRM) applications – likewise protect against unauthorized copies or otherwise misusing audio or video DRM content.
3. Enable secure implementation of M-commerce applications – likewise protect against access to a user's financial information, or otherwise compromise M-commerce protocols.
4. Be compatible with Federal Information Processing Standard (FIPS), Level 2 (L2) security requirements.

Security trust zones

Five trust zones are defined according to their trust level:

Zone 1 ARM supervisory mode execution. This is a completely secure zone, having the smallest possible code size. It executes core operating-system functions and crypto operations involving the hardware key or private keys from the secure file system.

Zone 2 aDSP RTOS execution. A completely secure zone that controls the access rights of individual aDSP applets.

Zone 3 ARM user-mode execution. A secure zone with access to the MDM device's radio control registers; it contains all AMSS information.

Zone 4 aDSP applet execution. A secure zone used for certain address ranges, depending upon the applet. Zone 2 (RTOS) is responsible for Zone 4 access rights.

Zone 5 ARM application-mode execution. This is a completely unsecured zone.

Protecting memory functions

Memory functions are also protected according to these zone definitions:

- ARM memory management unit (MMU)
  - Application execution is unsecured in any mode. The application MMU settings are not secure.
  - ARM supervisory-mode execution is completely secure. Therefore, the modem MMU is used to guard between individual AMSS processes and their access rights within the MDM memory space.
- aDSP RTOS: The aDSP can execute any mix of secure and non-secure applets. To support individual access rights per applet, RTOS provides driver functionality to request memory transfers or accesses. RTOS is trusted to identify access rights per applet and configure the assigned DM channel's memory protection unit (MPU) accordingly.
- MDM memory protection units: Since the application MMU is not secure, separate MPUs protect zones 1 and 2 from invalid accesses coming from zones 3, 4, and 5.

Summary of key security features

The MDM6x00 system provides seven key security features (Table 3-2) that ensure the security goals explained earlier.

Table 3-2 Security goals and features

Goal	Related features
Modem operation	**Memory protection** is essential to most MDM security goals, but it does not provide any resistance to physical attack, so it does not meet the FIPS L2 compatibility security goal. A **secure time source** is required for timely expiration of DRM content certificates and temporary debugging-right certificates. The latter are used to temporarily grant specific users system debugging capabilities that go beyond the capabilities specified by the secure model for this user. While the secure time source may, under certain conditions, get behind real (network) time, the MDM device ensures that system time always advances and can never be reset to any past value, or otherwise be compromised by an unauthorized user through a software-based physical attack.
DRM	**Memory protection** and **secure time source** were described above.
M-commerce	**Memory protection** and **secure time source** were described above. A software-based **random number generator (RNG)** supplies the cryptographically strong random numbers required by M-commerce cryptographic algorithms. The MDM device ensures that the RNG cannot be compromised by unauthorized users through software-based physical attack.
FISP L2 compliance	As required by M-commerce and DRM applications, the **secure file system (SFS)** provides long-term storage of user and device identity information in external non-volatile memory. This information must be encrypted, otherwise the FIPS L2 compliance goal is violated. Secure software encrypts sensitive data before storing it in external non-volatile memory using a device-specific hardware key. Since the keys are device-specific, if a key is compromised, only one device's security is breached (not all the devices ever manufactured). **Secure boot** uses strong cryptographic protocols. These protocols guarantee that the software image loaded during boot is the authentic image supplied by Qualcomm or its partners. Secure boot ensures FIPS L2 compliance. **Embedded memory** is used in the cryptographic algorithm computations that underline most MDM security features. When these algorithms are executed, either by software or hardware, secret information (cryptographic keys) has to be dynamically stored in memory. Embedded memory is used to ensure FIPS L2 compliance. **Secure debug** is supported, as required to ensure FIPS L2 compliance.

The following material provides an overview of these security features and related topics.

3.6.1 Secure time source

The secure time source implementation does not require any special hardware; it is achieved using memory protection (Section 3.6.4) and SFS (Section 3.6.2) security features. The secure time source includes these components:

- Maintains the local time reference even while the network time source is not available. The sleep-controller hardware timer maintains local time, and it is active as long as the phone is on.
- Periodically saves and restores time in external NV memory to prevent local time from being reset to zero when power is turned off
- Synchronizes with network time whenever possible

3.6.2 Secure file system

Hardware keys are used to facilitate the SFS. The crypto engine and data mover blocks are used to accelerate the encoding and decoding of data stored in the SFS.

The hardware keys are stored in on-chip NV memory (Qfuses) and are each 128-bits long, plus 65 bits for the FEC. Only secure software can read the hardware key values (by addressing their registers in the secure mode control module's memory map). Software must not store hardware keys in any memory except IMEM, where a secure partition must exist at all times while the key is held there.

Once a hardware key is provisioned, it cannot be altered by blowing the remaining hardware key fuses. A dedicated fuse is provided to disable programming of any hardware key fuses.

3.6.3 Embedded memory

For improved security, the MDM6x00 IC includes two types of embedded memory: boot ROM and IMEM.

Boot ROM

The 64 kB boot ROM holds the ARM primary boot loader (described in Section 3.6.6) and the eleven 2048-bit public keys. Eleven keys are believed sufficient, but as many as 16 can be supported by adding more keys to the boot ROM. Public keys are selected by the 4-bit PUBLIC_KEY_SELECT fuses in the CONFIG chain. Boot ROM implements a slave AHB interface and is connected to the modem bus.

IMEM

IMEM (Section 3.5) includes a 16 kB memory space that supports security-related applications. The ARM uses IMEM for secure computations involving long-term keys (device keys or user private keys). This IMEM implements a slave AHB interface and is connected to the modem bus.

3.6.4 Secure debug

Secure debug material is included within other sections: hardware requirements for secure boot (Section 3.6.6.2) and Qfuses (Section 3.6.7.2).

3.6.5 Memory protection

There are two types of memory protection:

1. The main type, a bus-centric mechanism, relies upon dedicated software-configurable memory protection units (MPUs) that define and enforce security partitions.
   - Only trusted software executing on the ARM processor AMSS kernel is allowed to configure and modify memory protection configurations.

- Each partition is characterized by:
  - Two boundaries whose alignment is specified by generics in MPU code
  - A set of up to 64 different permissions, each applying to a specific bus master
- Each permission is set to one of four possible values: No access, read only, write only, or full access.
- To allow MPU operation, the bus system includes a master identity with each bus request delivered to the MPU. Master identities are propagated through the MDM device's bus infrastructure using standard bus signals in combination with various subsystem-specific sideband signals.

2. A slave-centric memory protection mechanism:
   - Some slaves use specifically designed register maps; these maps allow MPUs to protect portions of the registers (such as DM and NAND).
   - In more complex cases, slave devices implement specialized logic (a register protection unit) that protects selected registers, or even portions of the register (like GPIO blocks).

3.6.5.1 EBI1 memory / AHB protection

EBI1 memory slave partitions

- AMSS partition – used by the AMSS trusted kernel; typically only accessible by the mDSP, ARM processor, and modem data mover channels that are designated as secure
- AMSS externalized state partition – used by the AMSS trusted kernel to externalize its state; adds read-only permission for all masters
- Remaining memory not belonging to any partitions should also be configured with the correctly chosen set of permissions. For instance, aDSP blocks should be prevented from accessing this memory to ensure that these modules cannot be used to copy DRM content to this otherwise unprotected memory.

AHB MPU

- Defines boundaries for eight security partitions:
  - Partition boundaries are 8 kB aligned.
  - Partitions may have adjacent boundaries and may overlap.
- Defines a set of 48 access permissions for each partition and a set of 48 access permissions for the remaining space (memory that does not belong to any partition)
- Supports four permission types: full access, read only, write only, and no access
- Detects burst requests that cross a 1 kB boundary and treats them as a security violation – otherwise, checking both the starting and ending addresses of each burst would be required
- Implements configuration registers for partition boundaries and partition permissions
- At reset, it enters a disabled state; when the MPU is disabled, it does not block any requests.

3.6.6 Secure boot

MDM power-up is coordinated by the PMIC. Once the supply voltages, clock signals, and PON_RESET_N signal are all established, the MDM device begins its start-up (or *boot*) process. The ARM processor controls the boot; it starts executing code from the internal boot ROM. This code is called the primary boot loader (PBL).

As the PBL runs, it accounts for some external control settings defined by the IC's pins and Qfuses (Section 3.6.7.2). These variations are ignored in the following description.

The PBL is entirely contained within the MDM device; it is preprogrammed in silicon and cannot be changed. This has two consequences:

1. The PBL cannot be altered or hacked, thereby enabling a trusted or secure boot.
2. Since it cannot be changed, it does not allow handset designers to customize the boot process for their unique products.

The first consequence is highly desirable – secure boot. The second is easily accommodated by an external boot memory that can be customized by the handset designer. This external memory is a flash device (either NAND or OneNAND) that interfaces with the ARM processor via EBI2 (Figure 3-8). Its boot-supporting code is called the secondary boot loader (SBL). Since this code is external, it cannot be trusted and must be authenticated by the security authentication code that resides in the internal boot ROM's PBL. Once authenticated, control is transferred from the PBL to the SBL.

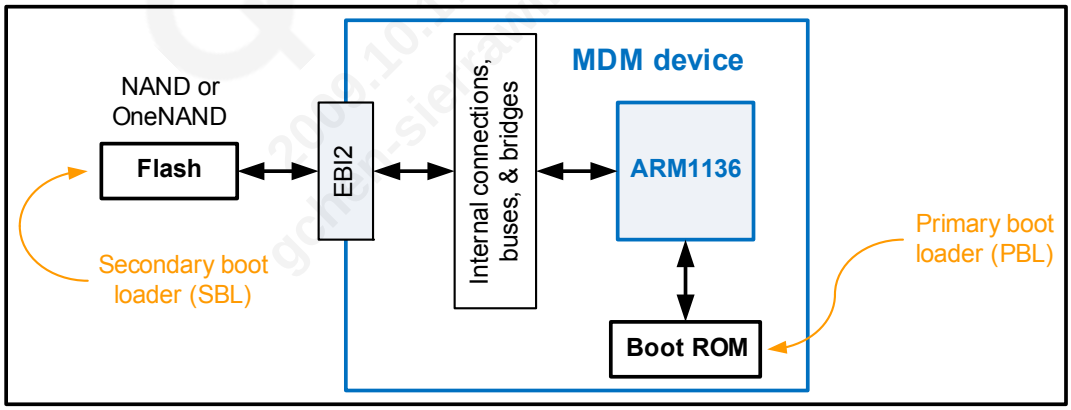


Figure 3-8 Boot loaders – internal and external memory for PBL and SBL code

3.6.6.1 Boot process

Start-up and the primary boot loader are described in Figure 3-9.

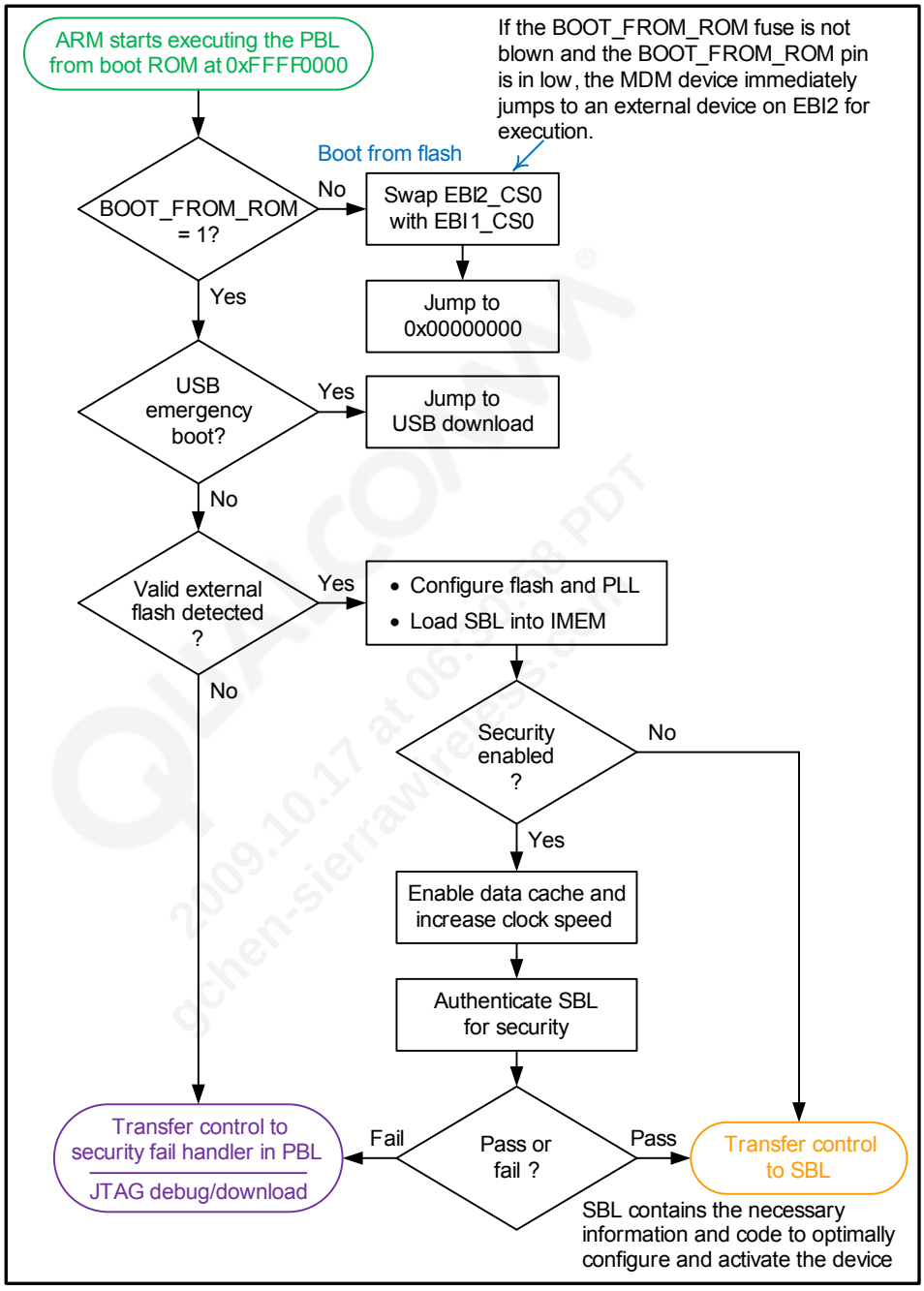


Figure 3-9 Start-up and PBL

3.6.6.2 Hardware requirements for secure boot

Boot ROM

Security requires booting from code that cannot be altered or hacked. Only on-chip memory meets this need, so the MDM device includes 64 kB for boot ROM. This ROM contains the PBL that is mapped to address 0xFFFF_0000, and since the ARM processor uses the PBL, it always boots from 0xFFFF_0000.

IMEM

During boot, basic configuration information is loaded from flash to IMEM. IMEM is not required to retain its state during shutdown, so it does not reside within the always-on domain.

Qfuses and mode pins

Two on-chip Qfuses and one external mode control pin help configure MDM security:

- SECURE_BOOT[1:0] Qfuses
  - If bit 0 is not blown, MDM security is determined by the external mode control bit (below) and bit 1 is inconsequential
  - If bit 0 is blown, MDM security is determined by bit 1 as defined below and the external mode control bit is free to be used as a normal GPIO port
    - If bit 1 is not blown – non-trusted boot
    - If bit 1 is blown – trusted boot (SBL and any subsequent code is authenticated for security)
- Pin D17 (GPIO_88 configured as BOOT_SCUR) – If this pin is enabled (high), it forces a trusted boot (the SBL or any subsequent code to be authenticated for security). This pin is meaningful only if security-enable bit 0 fuse is not blown.

MDM security configuration control is further defined within Table 3-3.

Table 3-3 MDM security configuration control [1]

SECURE_BOOT [1:0]	BOOT_SCUR at pin D17	Type of boot	Comments
X0	0	non-trusted	Pin D17 cannot be used as a normal GPIO
	1	trusted	
01	X	non-trusted	Pin D17 is available to be used as a normal GPIO
11	X	trusted	

1. X = don't care.

Hardware expected to be functional at boot (when ARM reset is released)

- The PMIC is active and provides:
  - An established clock – the SYS_CLK signal.
  - The MDM supply voltages at their correct levels.

- All AHB buses are active, with their clocks derived from the basic clock.
- All blocks are in their default power-up or reset state with their clocks off, except for the memory controllers that are active with their clocks derived from the basic clock.
- The NAND controllers are functional in some basic configuration.

Memory map requirements

The following address is a key memory-address location:

- 0xFFFF_0000 to 0xFFFF_FFFF – 64 kB space for the ARM boot code; the boot ROM is permanently mapped to this location

3.6.7 Other security topics

Key security topics were listed in Table 3-2 and described in earlier sections. Supplemental security-related topics are presented in the following subsections.

3.6.7.1 Modem subsystem security

The modem subsystem is protected from all outside masters by the unidirectional modem bridge. In addition, there are non-trusted masters inside the modem subsystem:

- ARM11 processor when running in user mode
- Data mover channels (MDM and ADM) controlled by ARM11 in user mode
- SMC register-map protection is ensured by checking the HPROT[1] bit. This bit is set during accesses produced by the ARM processor while in supervisory mode and during accesses produced by DM channels in security domain 0. ARM software must ensure that DM security domain 0 is only accessible to the ARM processor in supervisory mode. No access to the SMC is allowed in user mode.
- MPU configuration-register protection is also ensured by checking the HPROT[1] bit. This prevents access to MPU registers by masters like JTAG-to-AHB over a peripheral bus.

3.6.7.2 Qfuses

The MDM device includes on-chip non-volatile (NV) one-time programmable memory to store device configurations, hardware key values, boot ROM patch information, and memory redundancy information. These NV bits are electronic fuses called Qfuses.

Security-related NV content

Seven Qfuse chains are used for security-related applications; two must be considered by handset designers. If compromised (modified and/or observed), they may cause a security breach:

CONFIG (130 bits + FEC) – contains miscellaneous configuration information that can be programmed after packaging. Additional programming by handset designers (or their production staff) is allowed, if they were not already blown by Qualcomm. A fuse value of 1 means it has been blown. Security-related CONFIG bits are listed in Figure 3-4.

Table 3-4 Security-related CONFIG Qfuses

Name	Description
SECURE_BOOT[1:0]	Overrides the mode control pin (GPIO_88 = BOOT_SCUR) when blown and forces trusted boot
PUBLIC_KEY_SELECT	Public key select; selects one of 16 keys stored in boot ROM
ALL_DEBUG_DISABLE	Overrides the following fuse-controlled settings and forces them to disabled states: TAP_INSTR_DISABLE ARM_ANY_MODE_DEBUG_DISABLE TIC_DISABLE
TAP_INSTR_DISABLE	Disables up to 14 TAP instructions. JTAG access to each Qfuse chain requires a dedicated TAP instruction that is subject to being disabled by one of these bits. When the TAP controller decodes the disabled instruction, it treats it as a NOOP instruction.
ARM_ANY_MODE_DEBUG_DISABLE	Disables access to the ARM debug TAP in all modes
SW_FUSE_PROGRAM_DISABLE	Disables the ability to program fuses through the software interface
TIC_DISABLE	Disable ARM TIC
REQUIRE_UNLOCK	Requires that the unlock value be scanned in via JTAG or written via secure software to the SMC block before any of the debug features are enabled

Two MDM features ensure that it is safe to always allow JTAG access to the CONFIG chain:

- All CONFIG chain fields are defined in such a way that blowing additional fuses makes the device configuration more secure and results in a more restrictive setting. For example, blowing the ALL_DEBUG_DISABLE fuse causes all debug features to become permanently disabled, but there is not a fuse such as ‘DEBUG_ENABLE’ that increases debug capabilities.
- The CONFIG chain includes shadow registers that re-latch sensed fuse values upon completion of fuse-sense operations. This latching is done in the security mode control block, and is necessary because with JTAG access to the CONFIG chain, it is possible to scan any value into fuse cell latches, making a blown fuse appear unblown to the rest of the logic. Shadow registers store the actual fuse values immediately after sensing and maintain them even if different values are scanned into the fuse cells.

3.7 Clock generation and distribution

MDM clock circuits (Figure 3-10) provide single-phase clock generation and distribution for the entire system. These circuits distribute the operating clock signals for all internal circuits and external interfaces except the modem core. Rather than supplying the modem’s operating clock, several raw clock signals are provided so that the modem can generate its required operating clocks on its own.

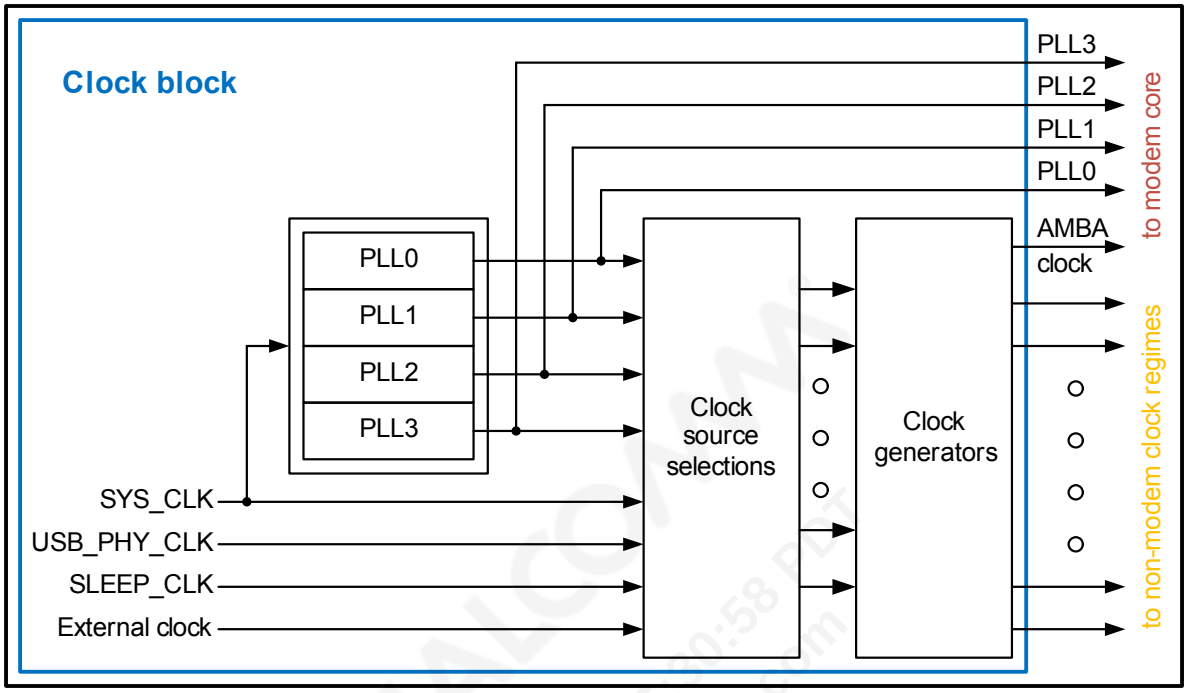


Figure 3-10 Clock-block basic architecture

The operating clock outputs meet the average frequency, jitter, duty cycle, phase, and spectrum requirements of each core or interface. Local gating reduces power consumption by disabling clocks to inactive circuits. All MDM clock circuits are designed to minimize power dissipation while meeting the necessary performance requirements.

All clock outputs are derived from a selection of input sources:

- An external clock
- SLEEP_CLK – a 32.768 kHz signal from the PMIC that drives the sleep controller when most MDM circuits are off and the 19.2 MHz XO is disabled
- SYS_CLK signal – a buffered 19.2 MHz signal from the PMIC; the primary operating clock source
- The on-chip PLLs (PLL0 through PLL3)

During normal operation, primary MDM processing blocks like CDMA, WCDMA, GSM, GNSS, ARM, and QDSP run off clocks derived from the PLLs. PLLs are used to generate several different clock frequencies that are stable, with little jitter.

3.7.1 Clock sources

The four PLL sources are designed to support a variety of frequency plans, but the primary plan uses just three:

- PLL0 – 294.912 MHz output; used for peripherals, GPS, and mDSP
- PLL1 – 235.954/245.786 MHz for 1X/WCDMA modem requirements
- PLL2 – 384/480 MHz for microprocessor, AHB buses, and memory controller requirements
- PLL3 – 405/480 MHz for aDSP

Due to the tight jitter requirements on some clock regimes and the wide frequency ranges needed to support various modes, alternate frequency plans may be necessary. There are many options for generating the required frequencies from one, two, three, or four PLLs. Any unused PLLs are turned off to conserve DC power.

The most basic application requirements might be fulfilled using a single PLL solution. In this ideal case, only PLL0 is active and it is used to generate all of the primary clock sources, including the low-jitter modem clock. In this case, PLL0 is supplemented by M/N:D counters (Section 3.7.3) that generate any other necessary clock sources.

PLLs in addition to PLL0 are turned on if the application requires:

- Higher clock frequencies with low jitter

  or

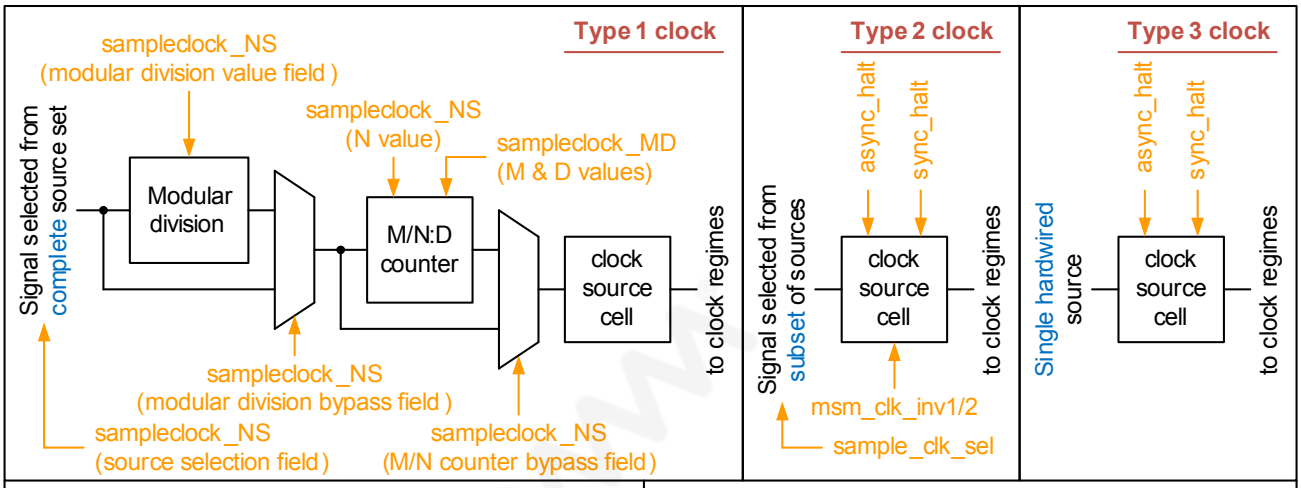
- Better jitter performance than achieved using the M/N:D counters

Other input sources and their uses are:

- SYS_CLK – frequency reference to PLL circuits; input clock to M/N:D circuits
- USB_PHY_CLK – generated by the PLL within the USB PHY block; used for USB circuits
- SLEEP_CLK – a digital clock input that is only used when the other clocks are disabled to conserve power
- External clock – only used for test purposes

3.7.2 Clock types

The MDM device includes three types of clock circuits (Figure 3-11) to address a wide range of requirements with the least possible complexity and power dissipation. In all cases, the clock source cell also serves as the output driver.



Type 1 clock

- Select from all available sources
- Highly configurable; note controls
- Modular division – div by 2, 3, or 4, or bypassed (1)
- Modular division = P = 1, 2, 3, or 4
- Modular division reduces M/N counter input frequency
  - Into M/N operating range (if needed)
  - Less power dissipation if just a divide is needed
- Overall output frequency = $\frac{M}{PxN}$ x source frequency

Type 2 clock

- Not configurable, but input source is selectable
- Select from a subset of sources
- Once source is selected frequency cannot be changed

Type 3 clock

- Source is hardwired; not selectable
- Output frequency cannot be changed

Figure 3-11 Three clock types

3.7.3 M/N:D counter

The MDM M/N:D counter circuit is shown and described in Figure 3-12.

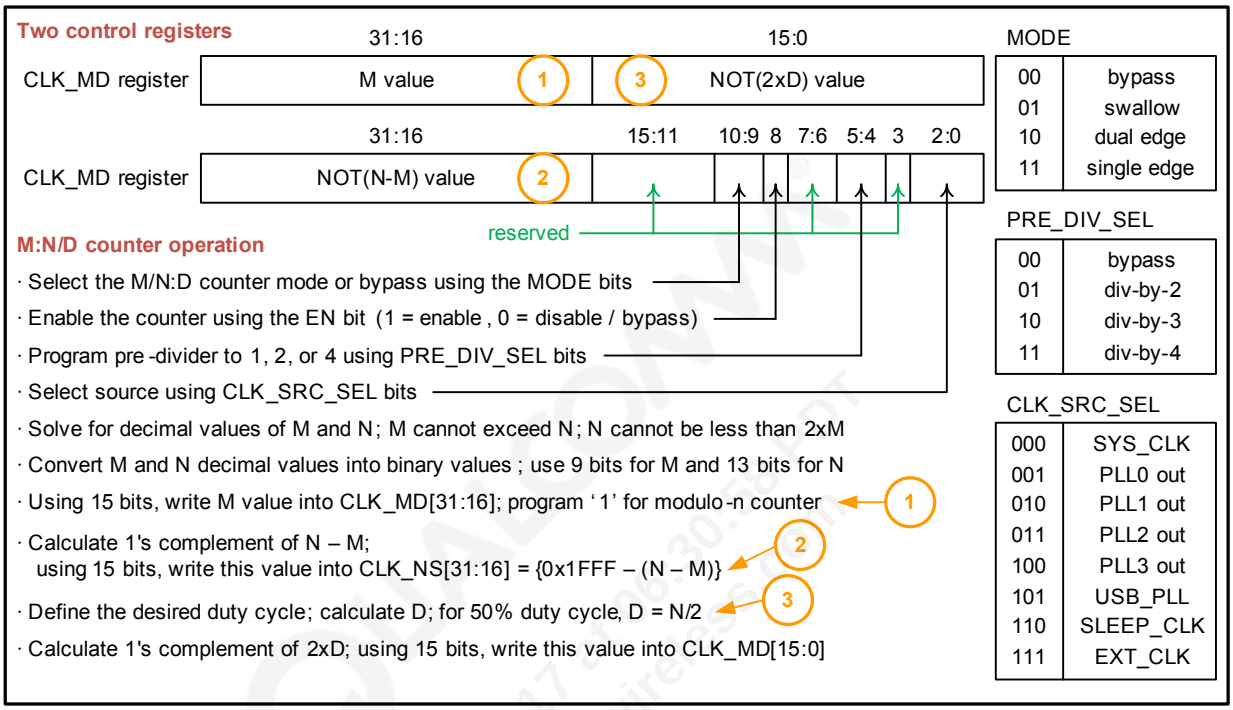


Figure 3-12 M/N:D counter

The M/N:D counter cannot divide by every fractional value. When non-integer divisions are used, the output jitter can vary greatly – from negligible to very serious jitter. The GP_CLK and GP_MN output signals must be evaluated carefully at the intended settings to verify jitter and overall quality.

NOTE For guaranteed minimal jitter, use even-integer divisors. For outputs that require non-integer divisors, request assistance at https://support.cdmatech.com (the CDMA Tech Support website) for jitter and signal-quality evaluation.

3.7.4 General-purpose clock outputs

The MDM6x00 IC provides two GPIOs that can be configured as general-purpose clock outputs:

- GPIO_17 = GP_CLK. This function was fully explained in Section 3.7.3 as an example using the M/N:D counter.
- GPIO_83 = GP_MN. This is very similar to GP_CLK, except that the only input available is TCXO/4.

3.7.5 Clock regimes

The MDM6x00 clock-generator block has more than 50 clock regimes, each with its own clock source cell that generates a single-phase clock from the desired source frequency. The clock source for each regime is chosen by a programmable multiplexer structure under microprocessor control. Even though several of the clock regimes have similar sources, they are separated to provide independent sleep control and gating.

3.7.6 Microprocessor and bus clocks

The microprocessor enters its idle (standby) mode when it has nothing to do. In this mode, the microprocessor clock and (optionally) the bus clocks are halted by writing a 1-0 sequence to the MICRO_CLK_OFF register. To resume operating, the clock is turned on when an unmasked interrupt is detected. However, if an interrupt is detected while the processor is in its clock-off sequence, the interrupt is saved and the clock-on process is held until clock-off is finished.

3.7.7 Codec clocks

A fractional-N PLL is available to generate the clocks for the different audio blocks in the MDM6x00 device.

- The wideband codec Tx path and Rx path clocks can be configured for OSR * F_s, where OSR is the over sample rate and F_s is the sampling frequency. The MDM6x00 device includes programmable OSR of 256, 128 and 64, and supports sampling frequencies of 8 kHz, 11.025 kHz, 12 kHz, 16 kHz, 22.05 kHz, 24 kHz, 32 kHz, 44.1 kHz, and 48 kHz.
- When the MDM6x00 device is in I^2S Tx master mode, it can generate the 32*F_s I^2S serial clock, where F_s is the sample rate. In I^2S mode, the MDM6x00 device can support rates of 8 kHz, 16 kHz, 24 kHz, 32 kHz, 44.1 kHz, and 48 kHz.
- When the MDM6x00 device is the master for both short-sync and long-sync PCM, the external codec PCM interface (PCM_CLK) can support the following rates:
  - In long-sync mode, the PCM_CLK is 128 kHz.
  - In short-sync mode, the PCM_CLK is N*8 kHz, where N can be 256, 128, 64, and 32.

3.7.8 Clock connections

An example application showing typical clock connections with comments is presented in Figure 3-13.

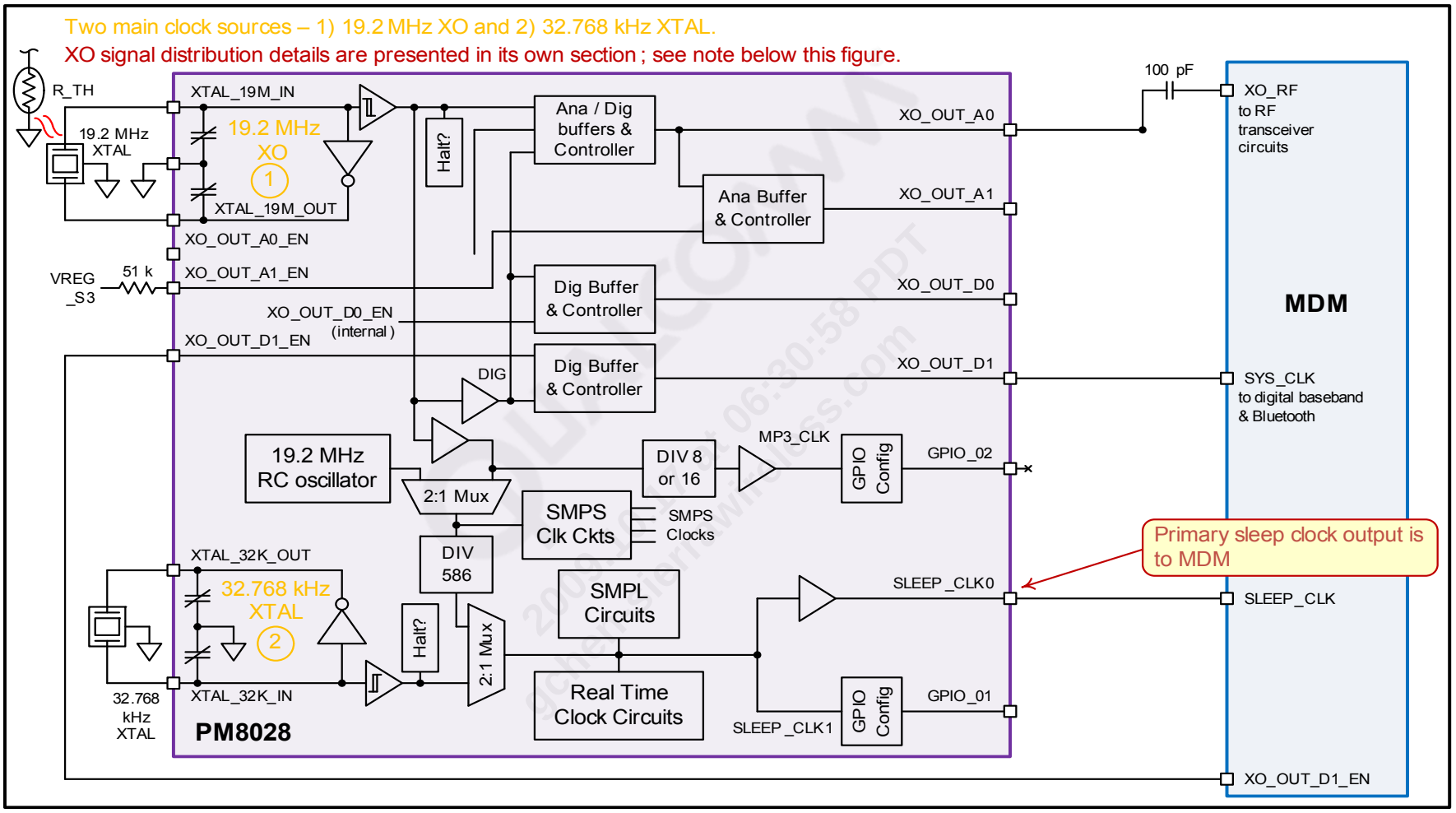


Figure 3-13 Example clock connections

3.8 Power optimization

- The MDM device includes power-control hardware that reduces its DC power consumption. Techniques include:
- Dynamic clock and voltage scaling (DCVS)
- Sleep mode with XO shutdown coordinated by the modem power manager (MPM)

3.8.1 Dynamic clock and voltage scaling (DCVS)

DCVS is a technique whereby the clock frequency and supply voltage of a digital circuit is dynamically adjusted in discrete steps to meet the processing performance requirements while minimizing DC power consumption.

DCVS adjustments are based upon combinations of requirements from different masters. A centralized clock and voltage-changing scheme is implemented that requires a client to register with its clock driver in order to enter or exit the DCVS mechanism. When all registrations favor DCVS, the relevant clocks and voltages are centrally lowered. Likewise, when any registration does not favor DCVS, all clocks return to their higher frequencies, and normal operating voltages are restored.

For additional information about DCVS, refer to *Application Note: MDM6x00 Dynamic Clock and Voltage Scaling* (TBD).

3.8.2 Modem power manager (MPM)

The MDM6x00 IC includes the modem power manager (MPM) function that allows it to enter a sleep mode in order to help minimize its DC power dissipation during long periods of phone inactivity. This sleep mode's power is reduced by:

- Turning off clocks to unused blocks.
- Turning off non-essential PMIC regulators.
- Reducing the MDM essential core voltage.
- Turning off the 19.2 MHz XO and running off the sleep clock.
- Maintaining the SSBI communication link with the PMIC to enable and disable the XO functions and voltage regulators.

MDM-to-PMIC connections

The MDM-to-PMIC connections required for MPM operation are shown in Figure 3-14.

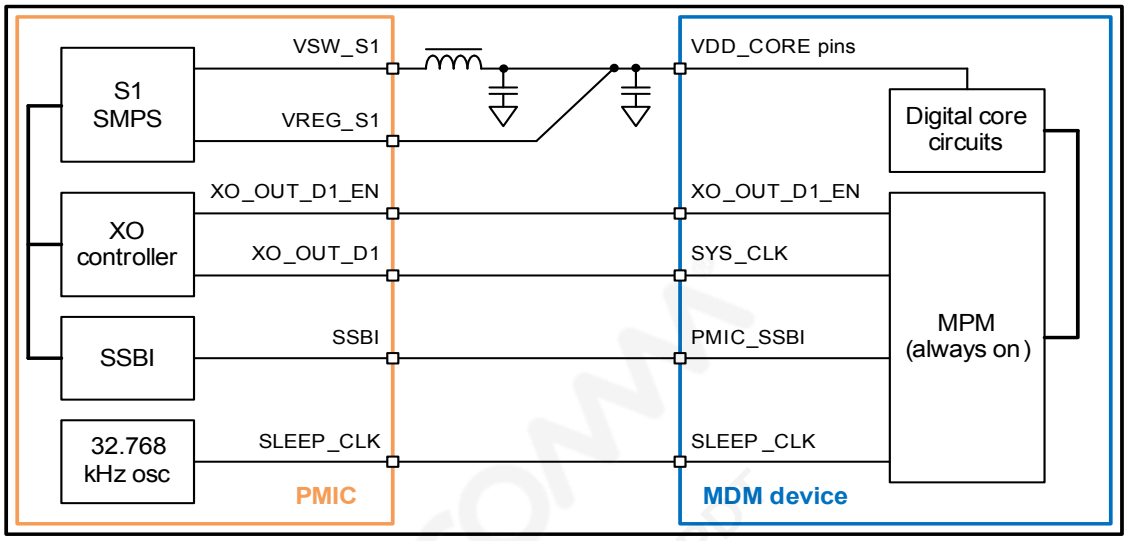


Figure 3-14 PMIC connections for MPM support

Entering Power savings mode

MDM software recognizes when its system-level operations will allow it to enter a sleep mode. Power savings is entered following the procedure outlined in Table 3-5.

Table 3-5 ARM shutdown procedure

Step #	Step description	Comments
1	Configure MDM I/Os to their power-saving states	ARM writes to control registers in the modem SS and MPM to set up and start sleep mode
2	Program MPM with wake-up timeout.	Timer prevents failed, unending sleep state
3	Save clock states.	In preparation for coming out of sleep
4	Disable all modem and peripheral clocks.	Power savings and prevent errors
5	Reroute all interrupts to the MPM.	To ensure continued monitoring of interrupts
6	If ARM processor is running SDRAM code, copy the final code segment to IMEM.	Allows the SDRAM to enter its self-refresh mode
7	Put SDRAM in self-refresh mode.	Power savings and fast recovery
8	Switch ARM clock to XO.	32 kHz sleep clock is not necessary, because MPM performs the XO power savings mode
9	ARM processor disables global clocks.	Writes to GLBL_SLEEP_ENA and executes SWFI instruction
10	Execute SWFI to trigger MPM state machine a) MPM sends PMIC commands to disable XO & lower MDM supply voltage. b) MPM waits for interrupt or expired timer.	Enables ARM processor to gate internal clocks, flush the write buffers, and assert signal indicating standby mode has been entered. When interrupt is received, the wakeup (or exiting power savings mode) procedure begins. [1]

1. If an interrupt is detected during the *entering shutdown* process, it is saved, and the wakeup process starts as soon as the shutdown procedure concludes.

Exiting power savings mode

Exiting power savings mode reverses the procedure (Table 3-6).

Table 3-6 ARM wakeup procedure

Step #	Step description	Comments
1	Interrupt or timeout occurs a) MPM sends PMIC commands to raise MDM supply voltage and enable XO. b) Duplicate interrupt from MPM to ARM	ARM processor accepts the interrupt.
2	The ARM processor enables global clocks.	By triggering the clk_ctl block
3	Switch ARM clock to full speed.	By bringing back PLLs and clock regimes, switching the arm_clk to a faster clock, and begin executing code
4	Bring SDRAM out of self-refresh mode.	Before accesses are made to it
5	Restore all modem and peripheral clocks.	
6	Restore MDM I/O states.	

3.9 Test ports (JTAG/ETM)

The MDM6x00 IC uses JTAG and ETM functions for test and debug.

3.9.1 JTAG

The MDM6x00 JTAG interface conforms to the IEEE 1149.1A-1993 standard specifying components that accept test instruction and data inputs, then provide the respective results as outputs in a serial format. The standard requires a test access port and a boundary-scan architecture to fulfill these requirements.

This test circuitry is used for board-level testing, and achieves the following objectives:

- Confirms that each component on the board performs its function correctly
- Confirms that all components are interconnected in the correct manner
- Confirms that the entire design behaves as intended

These confirmations are achieved using a boundary-scan architecture that includes a shift register stage (or cell) adjacent to each component pin so that signals at the component's boundaries can be tested, controlled, and observed. The boundary-scan cells are connected serially as a long chain, and behave as an overall shift register. More information about the boundary-scan cells is provided in Section 3.9.1.4.

For more detailed JTAG information, refer to *JTAG/ETM Interface for ARM11-based MSM Devices* (TBD) and the *IEEE Standard 1149.1A-1993*.

3.9.1.1 JTAG interface

The MDM JTAG port communicates with either the ARM1136-JS processor or the rest of the MDM device, depending upon the logic levels applied to the three MODE_X pins.

As defined by the standard, this JTAG interface allows test instructions and data to be shifted into the MDM device, and allows the test results to be read out in a serial format. The interface consists of four main functional elements (Figure 3-15): a test access point (TAP), a TAP controller, an instruction register, and three test data registers.

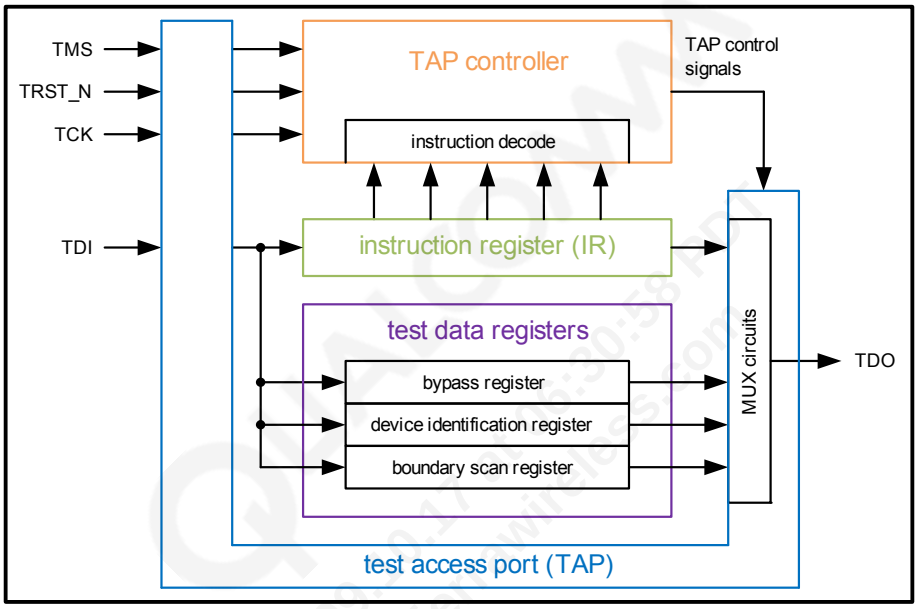


Figure 3-15 JTAG interface

3.9.1.2 Test access port

Descriptions of each TAP pin are given below.

- Test clock (TCK) user-defined input – independent of the system clock, it supplies a clock input to the serial test path between TDI and TDO. It permits shifting of test data concurrently with MDM device operation. Independence ensures that shifting of test data in and out of the TAP does not change the state of the MDM device. If TCK is stopped at zero, stored-states within the test logic are retained indefinitely.
- Test mode select (TMS) input – sampled on the rising edge of TCK, decoded by the TAP controller to set up all operations. Pulled-high internally to ensure that the MDM device continues operating normally if unconnected. A logic low must be applied externally to enter test modes.
- Test data input (TDI) – TAP input for test instruction or test input (serial format), sampled on the rising edge of TCK. Data is pushed into the TAP through TDI and propagates to TDO without inversion. Pulled high internally, so an open-circuit fault at the board-level within the serial data path will force a defined logic state into the TAP.

- Test data output (TDO) – TAP serial data output. States only change on the falling edge of TCK to ensure race-free operation (since changes on TMS and TDI occur only on the rising edge). The TDO driver is inactive (tri-state condition), unless data scanning is in progress.
- Test reset (TRST_N) input – when at logic low: 1) asynchronously resets the TAP controller, 2) drives it to the test-logic-reset state, and 3) asynchronously initializes all other test logic (as is required by the test-logic-reset state). It is pulled-down internally to ensure that the MDM device continues operating normally if unconnected.
- Return TCK (RTCK) output – used to synchronize the TCK clock with the core clock. The external device issues a TCK signal and waits for the RTCK response. Synchronization is preserved, since progress is not allowed unless each TCK output is followed by a RTCK input.

3.9.1.3 TAP controller

The TAP controller is a synchronous finite state machine that responds to TMS and TCK changes to control a sequence of operations. It has 16 states for capturing, shifting, and updating data and instructions; a state for running tests; plus a reset state. For a detailed description of the different states and a state diagram, refer to *IEEE Standard 1149.1a-1993*.

The actions of test logic (both instruction and data registers) occur on either the rising or falling edge of TCK in each respective controller state; again, for more information, see the standard given above.

The TAP controller initializes when it is in the test-logic-reset state; this state is achieved either by:

- Applying a logic low to TRST_N to asynchronously enter the test-logic-reset state

  or by

- Holding the TMS high for at least five rising TCK edges to synchronously enter the test-logic-reset state (worst-case time from any other state).

Once the test-logic-reset state is reached, all test logic disables and normal MDM operations resume.

For board-level designs where JTAG testing is not used, TRST_N must be a logic low. Before using the MDM device in a mobile station application, TRST_N must be pulled low at powerup, or the TAP controller must be in the test-logic-reset state.

3.9.1.4 Data registers

Device identification register

The device identification register (Table 3-7) provides a way for the manufacturer, part number, and version of a component to be determined through the TAP. An example application is to distinguish component manufacturers on an assembled board when more than one source is used.

Table 3-7 JTAG device identification register

Bits	Name	Description
31:28	VERSION_ID	Device version data
27:12	PARTNUM	A Qualcomm-generated device part number
11:1	QUALCOMM_MFG_ID	Device manufacturer identification code (administered by JEDEC); Qualcomm's code is 000_0111_0000 (0x070)
0	START_BIT	Register's start bit – always set (1)

Selecting the device identification register loads its value into the shift register on the rising edge of TCK in the capture-DR state. It has a parallel input, but it does not have a parallel output. Figure 3-16 shows the structure of the device ID register.

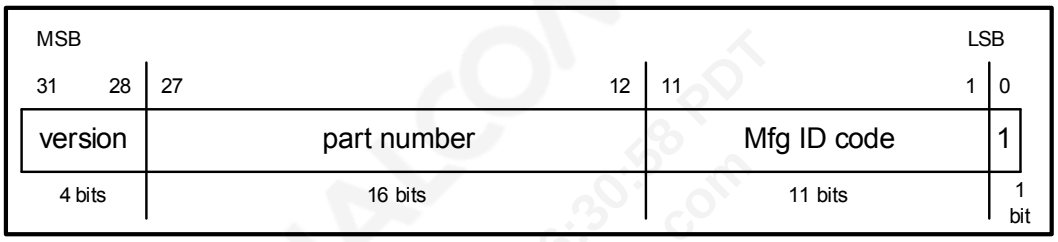


Figure 3-16 Device identification register data structure

Bypass register

The bypass register is a single-stage shift register located between TDI and TDO that is used to route the scan data directly from TDI to TDO without going through the entire boundary-scan register. Bypassing the boundary-scan register allows quicker movement of data to and from other board components. The bypass register operation has no effect on the operation of the on-chip system logic. It does not have a parallel output.

Selecting the bypass register loads the shift register with a logic low on the rising edge of TCK, following entry into the capture-DR state.

Boundary-scan register

The boundary-scan register allows board interconnection testing in order to detect typical defects like opens and shorts. The boundary-scan register provides connections, via boundary-scan cells (Figure 3-17), between each digital I/O pin and the MDM-internal circuits. This gives the tester the capability to access and/or sample system I/O signals when testing system logic. The boundary-scan register also handles parallel inputs and outputs.

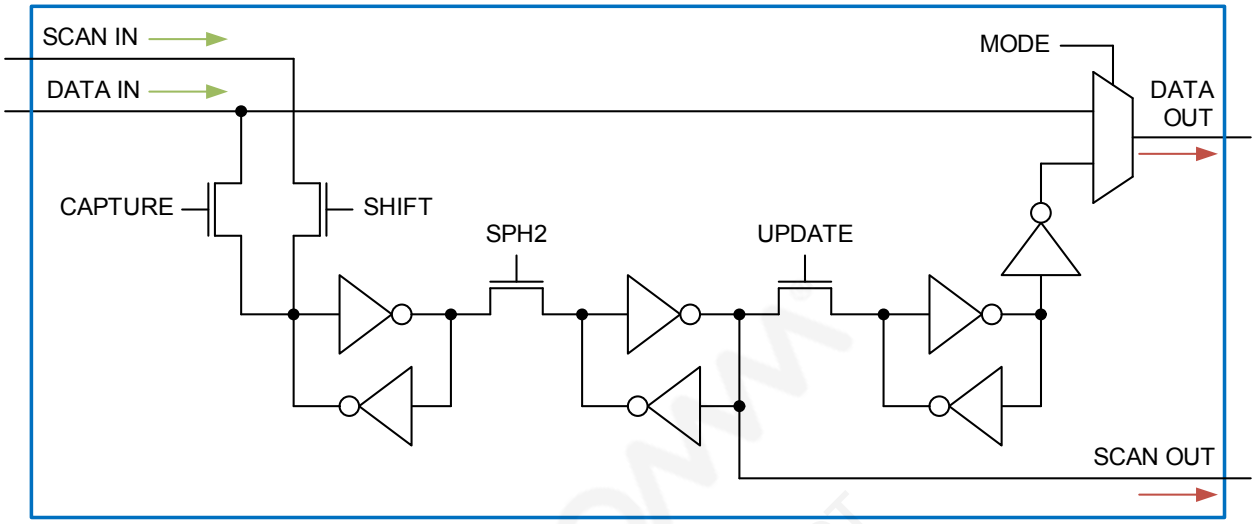


Figure 3-17 Typical structure of a boundary-scan cell

The boundary-scan register is comprised of boundary-scan cells that access digital signals within the MDM device. Each cell has a single-shift register stage, has single serial input and output terminals, and is connected to the one before it and after it within the boundary-scan register. In addition, each cell has a parallel input/output terminal through which data is sampled and/or latched (this is how the entire register handles parallel input/output).

3.9.1.5 Instruction register

The instruction register (IR) is a 7-bit shift register having a serial input and parallel latched output. It holds JTAG instructions that were shifted into the JTAG interface, and allows instructions to be serially loaded into the test logic during an IR scan cycle. A JTAG instruction shifts through TDI (LSB first) until it is in the IR. Once in the IR, the instruction is decoded and the test to be performed and the data register to be used are selected. The parallel output holds the current instruction, and is updated synchronously (on the falling edge of TCK when in the update-IR state) or asynchronously (upon entry into the test-logic-reset state). The basic functions used and supported on the JTAG interface are BYPASS, SAMPLE/PRELOAD, EXTEST, and IDCODE.

3.9.1.6 JTAG selection

Since the embedded processor and the MDM device as a whole have JTAG capability, users must select the desired operating mode. Two important examples are shown and described in Figure 3-18.

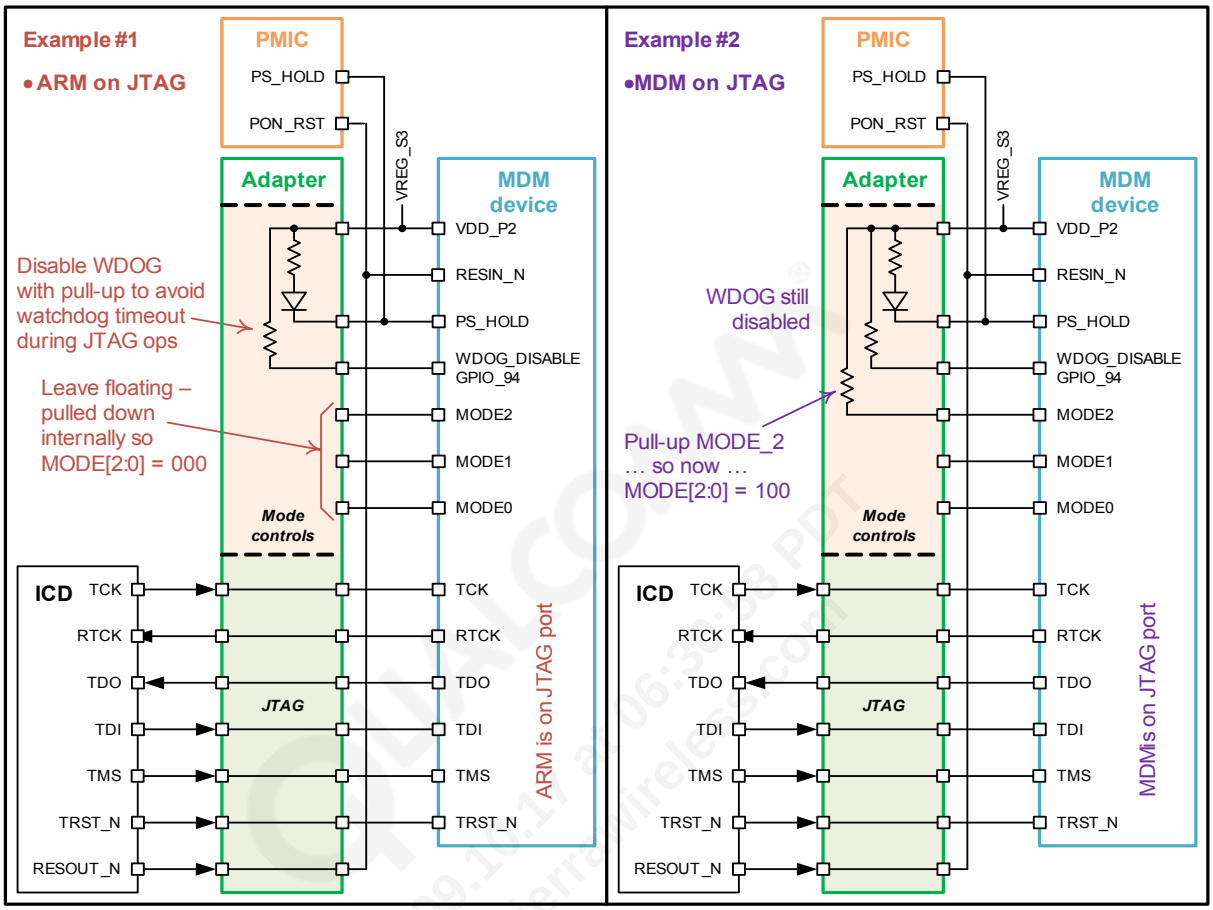


Figure 3-18 JTAG mode selection examples

3.9.2 Embedded trace macrocell (ETM)

This section describes the operation of the embedded trace macrocell (ETM) in the ARM family of processors, and how it is implemented and used on the MDM device. The ETM block provides instructions and data trace that allows real-time system debugging.

The ETM consists of two parts:

- Trace port – broadcasts trace information (either instruction trace or data trace)
- Triggering facilities – controls the ETM to filter and control trace operations

The ETM is connected directly to the ARM core it is tracing. The trace port connects to a software debugger that configures the ETM and processes the information provided. An example debugging environment is shown in Figure 3-19.

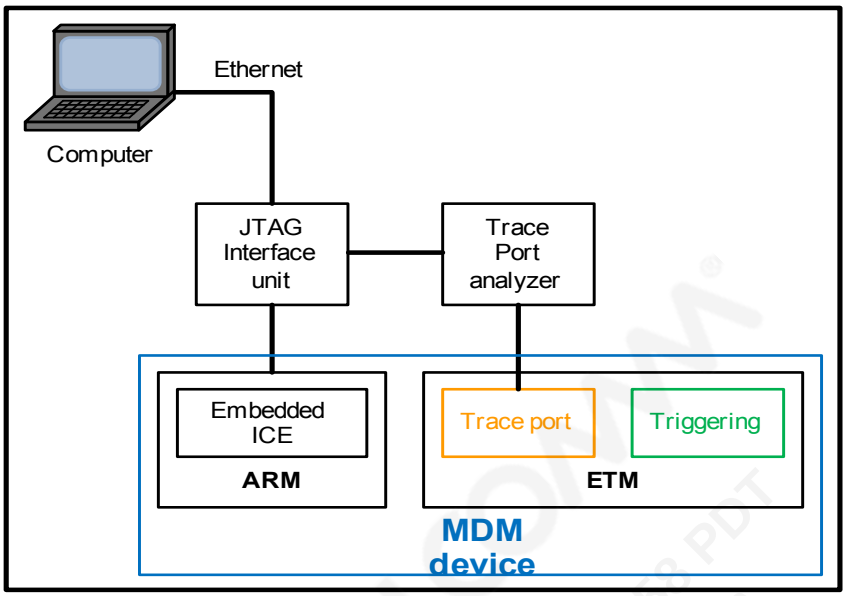


Figure 3-19 Example debugging environment using ETM

Visit the ARM website (www.arm.com) to download more detailed information about the ETM, or refer to *JTAG/ETM Interface for ARM11-based MSM Devices* (TBD).

3.9.2.1 ETM architecture

The ARM11 ETM architecture is shown and described in Figure 3-20.

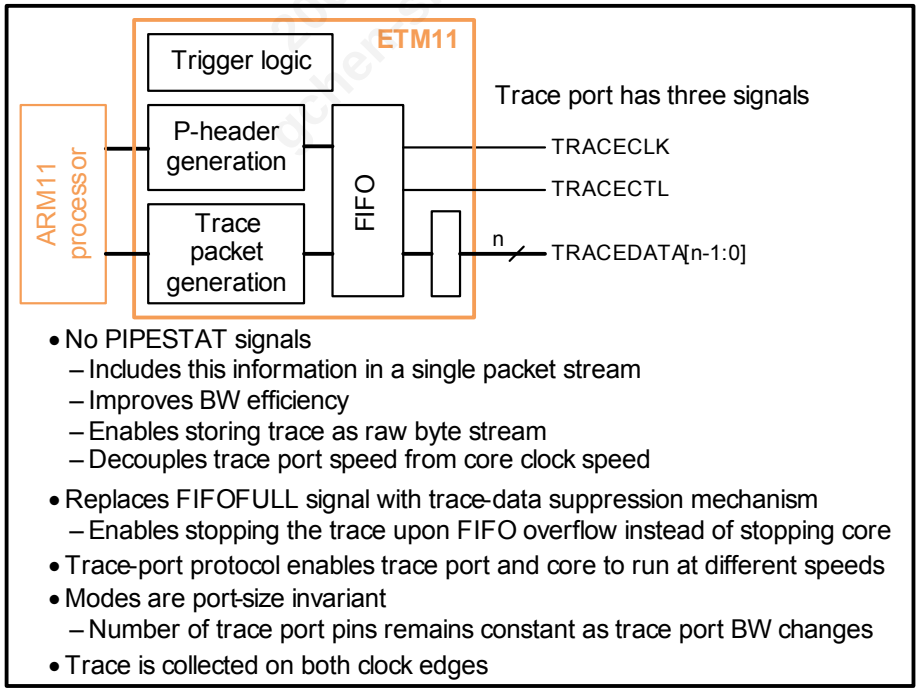


Figure 3-20 Basic ETM11 architecture

3.9.2.2 ETM connections

All ETM pin assignments are listed in the *MDM6200 and MDM6600 Mobile Data Modem Device Specification (Advance Infomation)* (80-VR001-1). Visit the ARM website (www.arm.com) for more detailed information about ETM connections.

The ETM11 supports two modes: 1) single-processor trace mode, and 2) 16-bit normal mode. Two trace clock (TRACECLK) modes are also supported: *1:2* and *1:4*. The ETM11 always transfers data on both edges of TRACECLK. Therefore, when 1:2 clocking mode is selected, TRACECLK is a divide-by-four of the core clock (CLK). However, data is transferred on both edges of TRACECLK, so there is effectively one data transfer every two CLK cycles, yielding the 1:2 ratio.

In the case of the 1:4 clocking mode, TRACECLK is CLK/8, but since data is being transferred on both edges of TRACECLK, there is a data transfer every four CLK cycles, yielding the 1:4 ratio.

The FIFO size for the ETM11RV is fixed at 69 bytes. With the slower TRACECLK frequency (1:4 clocking mode), the effective data transfer rate going off chip is reduced. This makes a FIFO overflow more likely, resulting in lost trace data. FIFO overflow can be reduced by carefully setting up the trace filter rules to minimize the amount of trace captured. There is also a trade-off against port size: choosing a larger port size increases the data throughput and reduces the likelihood of causing a FIFO overflow, but requires more MDM pins.

Table 3-8 lists the pinout for a single-processor ETM11 connector. If a wider trace port (more than 16 bits) is required, a second 38-pin trace-port connector must be used.

Table 3-8 ETM11 single-processor connector pin assignments

Pin #	Signal name		Pin #	Signal name
1	NC		20	TRACE_DATA[5]
2	NC		21	mTRST
3	NC		22	TRACE_DATA[4]
4	NC		23	TRACE_DATA[15]
5	GND		24	TRACE_DATA[3]
6	TRACECLK		25	TRACE_DATA[14]
7	DBGRQ		26	TRACE_DATA[2]
8	DBGACK		27	TRACE_DATA[13]
9	nSRST		28	TRACE_DATA[1]
10	EXTTRIG		29	TRACE_DATA[12]
11	TDO		30	GND
12	VT_Ref		31	TRACE_DATA[12]
13	RTCK		32	GND
14	V_Supply		33	TRACE_DATA[10]
15	TCK		34	VDD
16	TRACE_DATA[7]		35	TRACE_DATA[9]
17	TMS		36	TRACE_CTL
18	TRACE_DATA[6]		37	TRACE_DATA[8]
19	TDI		38	TRACE_DATA[0]

4 Memory Support

The MDM6x00 IC external memory support (Figure 4-1) includes the following major functions:

- External memory controller that handles all communications between the AHB masters, the low-latency internal memory (IMEM), and the external bus interfaces
- For high-speed memory devices – the EBI1 port
- For lower-speed memory devices, and other peripheral devices – the EBI2 port

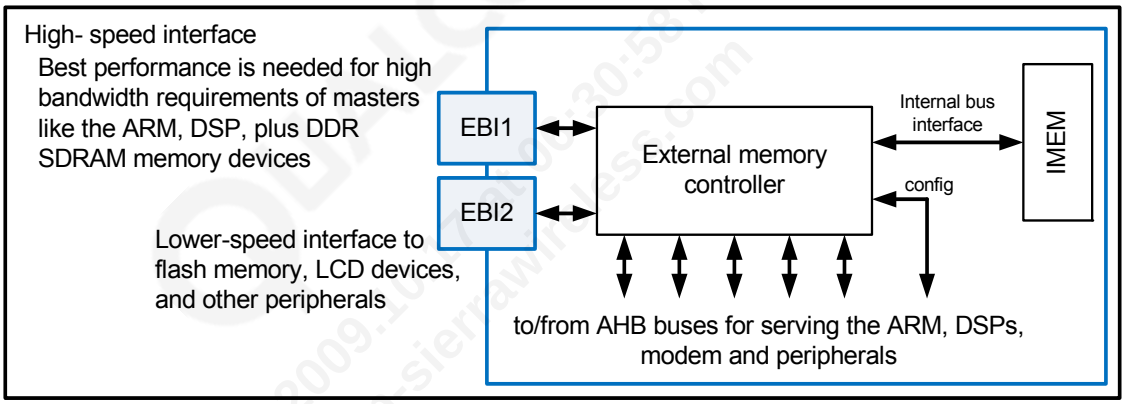


Figure 4-1 External memory support high-level block diagram

4.1 Supported external devices

External device configurations supported by the EBI1 and EBI2 ports are listed in Table 4-1.

Table 4-1 Memory configurations

CFG	EBI1 support	EBI2 support [1]
1	16-bit DDR SDRAM	NAND flash (boot device)
2	16-bit DDR SDRAM	OneNAND flash (boot device)

1. Asynchronous peripheral or memory devices are also supported (optional).

A section dedicated to each major memory function is presented after Section 4.2.

4.2 Memory map

Table 4-2 Memory map (addressable)

Region	Base address	Max address	Range (MB)
EBI1_CS0	0x00000000	0x0FFFFFFC	256
EBI1_CS1	0x10000000	0x1FFFFFFC	256
IMEM	0x20000000	0x27FFFFFC	128
EBI2_CS0	0x28000000	0x2FFFFFFC	128
EBI2_CS1	0x30000000	0x37FFFFFC	128
EBI2_CS2	0x38000000	0x3FFFFFFC	128
EBI2_CS3	0x40000000	0x47FFFFFC	128
EBI2_CS4	0x48000000	0x4FFFFFFC	128
EBI2_CS5	0x50000000	0x57FFFFFC	128
EBI2_CS6	0x58000000	0x5FFFFFFC	128
RESERVED	0x60000000	0x67FFFFFC	128
RESERVED	0x68000000	0x6FFFFFFC	128
NANDC	0x70000000	0x77FFFFFC	128
RESERVED	0x78000000	0x79FFFFFC	32
	0x7A000000	0x7BFFFFFC	32
	0x7C000000	0x7DFFFFFC	32
	0x7E000000	0x7FFFFFFC	32
	0x80000000	0x83FFFFFC	64
	0x84000000	0x87FFFFFC	64
	0x88000000	0x8FFFFFFC	128
	0x90000000	0x97FFFFFC	128
	0x98000000	0x9FFFFFFC	128
	0xA0000000	0xA7FFFFFC	128
	0xA8000000	0xAFFFFFFC	128
	0xB0000000	0xB7FFFFFC	128
	0xB8000000	0xBFFFFFFC	128
	0xC0000000	0xC7FFFFFC	128
	0xC8000000	0xC9FFFFFC	32
	0xCA000000	0xCBFFFFFC	32
	0xCC000000	0xCDFFFFFC	32
	0xCE000000	0xCFFFFFFC	32
	0xD0000000	0xD7FFFFFC	128
	0xD8000000	0xD9FFFFFC	32
	0xDA000000	0xDBFFFFFC	32

Table 4-2 Memory map (addressable) (cont.)

Region	Base address	Max address	Range (MB)
RESERVED	0xDC000000	0xDDFFFFFC	32
	0xDE000000	0xDFFFFFFC	32
	0xE0000000	0xE0FFFFFC	16
	0xE1000000	0xE1FFFFFC	16
	0xE2000000	0xE2FFFFFC	16
	0xE3000000	0xE3FFFFFC	16
	0xE4000000	0xE4FFFFFC	16
	0xE5000000	0xE5FFFFFC	16
	0xE6000000	0xE6FFFFFC	16
	0xE7000000	0xE7FFFFFC	16
	0xE8000000	0xEFFFFFFC	128
	0xF0000000	0xFFFEFFFC	384 - 0.064
BOOT UP REGION	0xFFFF0000	0xFFFFFFFC	0.064

4.3 External memory controller

This material will be included in a future revision of this document.

4.4 High-speed memory interface (EBI1)

The EBI1 port connects the external DDR SDRAM with the memory controller, which then makes the desired transfers to and from the internal application and modem systems via the AHB buses.

4.4.1 EBI1 connections

All EBI1-related pins are located in the same general area of the IC package. This allows optimal routing, and should minimize interference from this bus to more sensitive functions (such as analog baseband). EBI1 layout guidelines are given in Section 4.4.3.2.

DDR SDRAM memory options are shown in Figure 4-2.

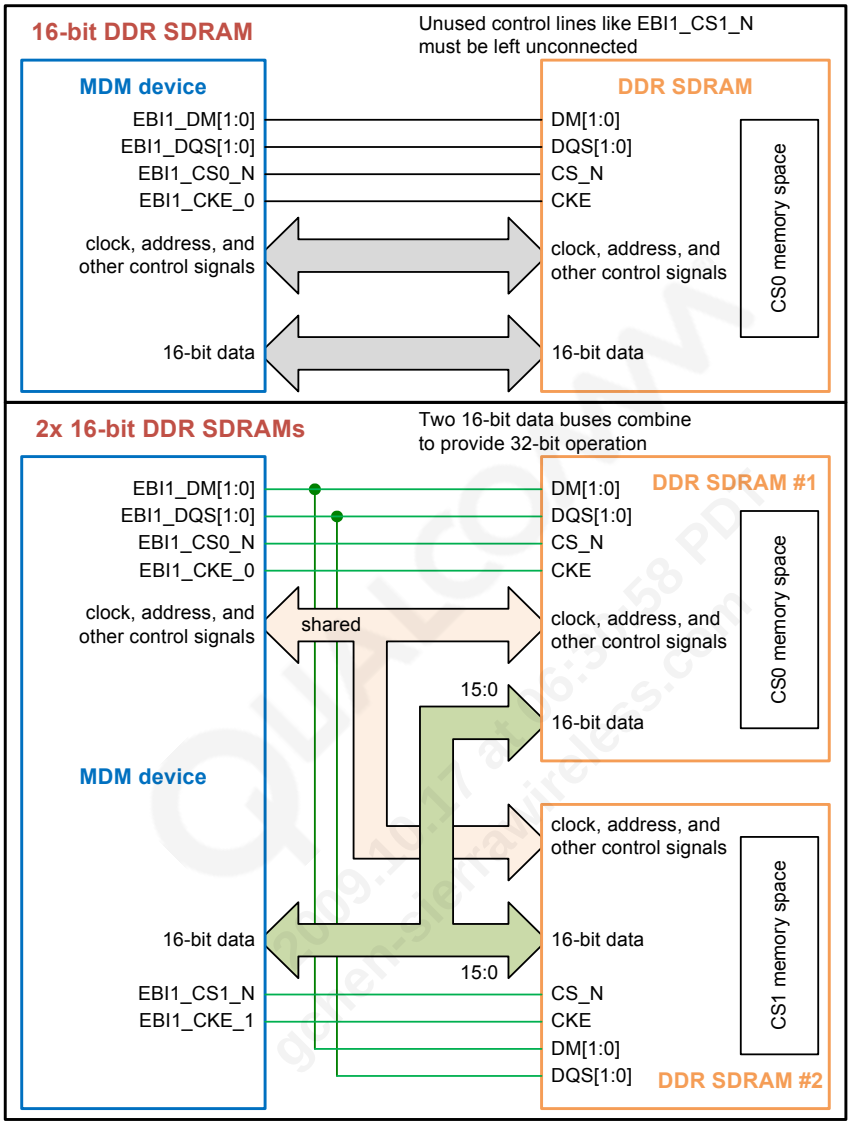


Figure 4-2 EBI1 interface options

4.4.2 External DDR SDRAM memory

The SDRAM's double-data-rate (DDR) architecture enables high-speed operation by transferring two data words per clock cycle (one word with each clock edge) at its I/O pins. DDR SDRAM is a source synchronous interface – the device supplying the data must also supply the clock. This is achieved by using the bidirectional DQ strobe (DQS) signal as the clock during data transfers. During writes, the MDM device drives data on the DQ lines and supplies a clock on the DQS line; the DQS signal is used to capture data at the DDR SDRAM receiver end.

4.4.2.1 DDR SDRAM features

- Clock rates from 9.6 MHz to 160 MHz
- Low-power, 1.8 V I/O

- 16-bit
- Separate CKE pin for each CS pin, for partial operation
  - Two DDR SDRAM devices are connected.
  - One is set to self-refresh mode, while the other is active.
- Programmable configurations
  - Memory organization (interleaving or linear)
  - Address widths for row, column, and address bits
  - CAS latency (3) clock cycles
  - Burst length (x4, x8)
  - Auto or manual precharge
- Partial refresh
- Idle power-down to reduce idling power consumption
- Self-refresh power-down to sleep the entire system
- Most SDRAM commands are supported by the MDM device (issued through configuration registers within the external memory controller)

For more register information, see the *MDM6200 and MDM6600 Mobile Data Modem Software Interface* (80-VR001-2).

4.4.2.2 DDR SDRAM initialization

After MDM power-up, the DDR SDRAM device is configured as part of the EBI1 configuration sequence – two DDR SDRAM registers (mode register and extended mode register) must be initialized. There are two different ways to initialize these registers:

- Upon software request to write to the register, the controller finite state machine (FSM) generates the command sequence and initializes the register (the command sequence for configuring DDR SDRAM registers takes one clock).
- The controller provides a mechanism for the software to generate the command sequence to the DDR SDRAM through a register write.

The following procedure describes the initialization flow of the DDR SDRAM device. The initialization sequence is device-specific and the flow given here is just an example. Refer to the DDR SDRAM datasheet for a detailed description of the initialization flow sequence.

1. The system is initialized.
2. The system is idle for 200 μs.
3. Issue at least one NOP command (CS[1:0] = 1, RAS = 1, CAS = 1, WE = 1).
4. Issue a precharge command to all of the banks.
5. Initialize the SDRAM device mode register (per the device datasheet).
6. Initialize the SDRAM device extended-mode register (per the device datasheet).
7. Issue an auto-refresh command to all banks.

The DDR_SDRAM_INIT_CTRL register contains all the signals required to initialize the external SDRAM device. When software writes a value into this register, the controller FSM goes into its configuration mode as early as possible and maps the bits of this register to the SDRAM interface pins. Using this mechanism, the software can update the mode and extended mode registers within any SDRAM device. For more details, see the SDRAM initialization flow chart in the *MDM6200 and MDM6600 Mobile Data Modem Software Interface* (80-VR001-2).

The following procedure outlines the sequence for a mode register access:

1. The software writes data to this register.
2. The software writes to the configuration access register.
3. The state machine completes its pending transaction and moves to the configuration state.
4. The state machine controls the interface module to take the MRS values from the register storage.
5. The interface module outputs the contents of this register to the I/O pads.
6. The state machine clears the contents of this register.
7. The state machine completes the MRS access.

4.4.2.3 DDR SDRAM refresh logic

DDR SDRAM devices must be refreshed at regular intervals to retain the stored data. This can be done using auto-refresh or self-refresh modes; the MDM device supports both. Self-refresh is used when the MDM device is in a power-down mode; the SDRAM retains its data without external clocking. The CS1_SELF_RFRSH and CS0_SELF_RFRSH bits in the DDR_SDRAM_DPD_CTRL register indicate whether the SDRAM is in the self-refresh mode or active mode.

4.4.3 DDR SDRAM design considerations

Due to its high data rates, the DDR SDRAM interface requires very careful timing calculations and controlled impedance routing. For example, the time skew allowed between DQS asserted and valid data is 0.6 ns. This suggests that signal integrity can be violated with a long PCB trace, excessive parasitic capacitance, or similar layout issues.

Key design issues are described in this section. Additional details are provided in *Application Note: DDR SDRAM Design Considerations for Mobile Station Modem (MSM) Devices* (80-V9038-54).

4.4.3.1 Package and PCB impedances

The MDM6x00 package and the PCB it is mounted on have impedance characteristics that should be considered when designing its high-speed interface to external memory devices (Figure 4-3).

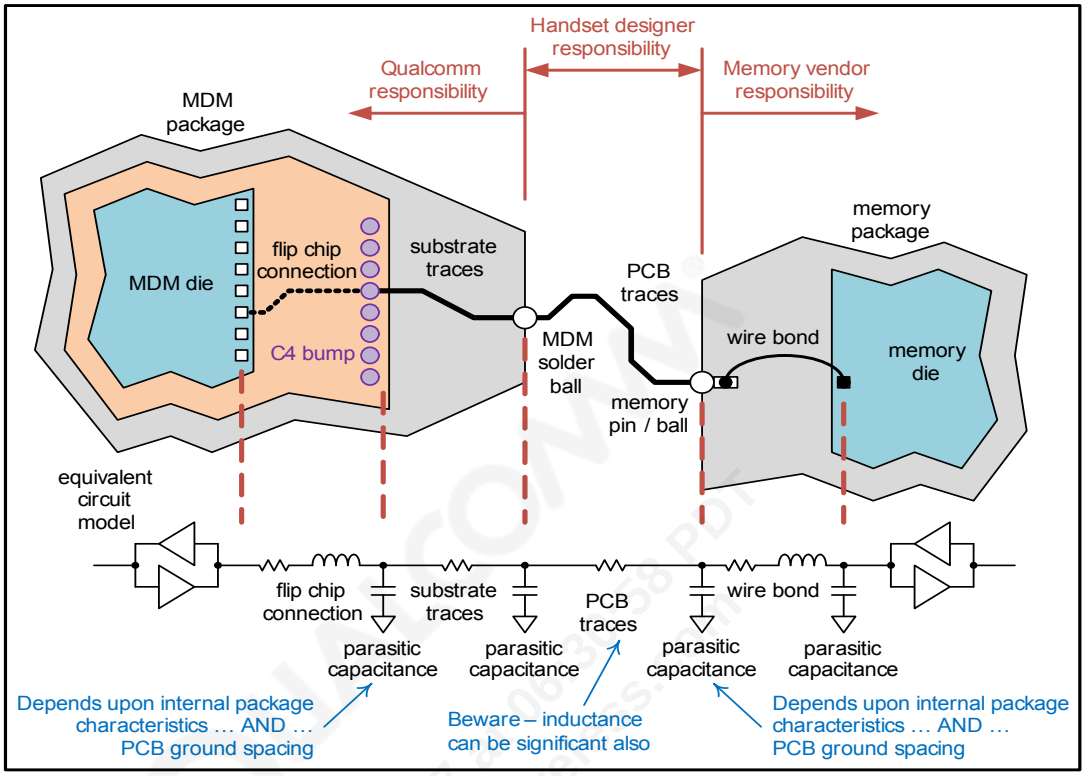


Figure 4-3 EBI1 output impedance modeling

All of the modeled components can factor into the interface's signal integrity. For example, series resistance, series inductance, and shunt capacitance slow the rising and falling edges and distort the waveform. Since the specified clock rates are guaranteed by Qualcomm and the memory vendor, those component designs should be adequate. It is left to the handset designer to maintain interface performance with a quality PCB design. Some guidelines are given next.

4.4.3.2 EBI1 layout guidelines

The PCB layout requires extra care and attention to preserve the EBI1 high-speed performance; in particular, the loads seen by the bus must be minimized and the jitter on all signals must be limited.

- Minimize parasitic capacitance at all component signal pads by clearing internal layer ground directly below the pads on one or two adjacent layers (Figure 4-4). The board's metal layers are like a parallel-plate capacitor whose capacitance is inversely proportional to the separation between the metal plates. Clearing the inner-layer ground increases separation, thereby decreasing capacitance.

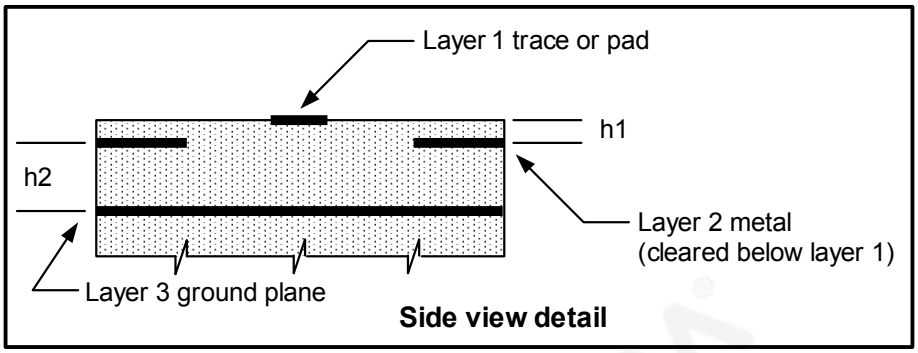


Figure 4-4 Clearance below component pads and signal traces

- Use controlled-impedance traces (microstrip or stripline) between the MDM and memory devices. The line impedance depends upon several variables – trace width and thickness, height of dielectric material between the trace and ground plane(s), and dielectric constant of the PCB material. Given the PCB material selected, the geometry of the microstrip or stripline elements must be designed properly to provide the desired controlled impedance value.
  - When routing on an external layer, use microstrip techniques where possible.
  - When routing on internal layers, use stripline techniques where possible.
  - Maintain continuous ground below microstrip traces and above and below stripline traces.
  - Keep traces short and direct to minimize loss and undesired coupling.
  - Fill the areas along both sides of traces with ground to improve isolation, providing adequate clearance to minimize coplanar capacitance and leakage.
  - Use several ground vias to connect outer ground fill areas to the internal ground planes.
  - Avoid crossing high-speed traces if possible.
- The differential clock signal (EBI1_DCLK and EBI1_DCLKB) requires special consideration (in addition to those listed above).
  - Use controlled impedance lines for each of the two differential traces.
  - Route the two traces in parallel and close to each other. If the trace width is W, the spacing between traces should be on the order of 2W to 3W.
  - The two traces must have nearly equal length, otherwise the complementary phase relationship between the two is degraded (impacting the timing, duty cycle, and possibly the detection margins).
- The DQS signals are like clocks between the MDM device and DDR SDRAM circuits. The timing difference (delay) between the DQ and DQS signals needs to be minimized – the timing must be well matched. All DQ and DQS signals must also be well shielded from any noise sources.
- There are four DQS lines on the EBI1 bus, with each DQS signal associated with its own group of data (DQ) lines:
  - EBI1_DQS_1 for EBI1_DQ[15:8]
  - EBI1_DQS_0 for EBI1_DQ[7:0]

- The following signals should have similar PCB characteristics (such as trace length, capacitive loading, and logic delay) if the EBI1 rates are to be achieved:
  - As described earlier, the differential clock signals (EBI1_DCLK, EBI1_DCLKB) should be routed together and have the same length between the MDM and the memory device(s).
  - Every 8 bits of EBI1 data (DQ[15:8], DQ[7:0]) and their corresponding DQS and DQM bits on the SDRAM interface. Each of these signal sets are a group; for a 16-bit DDR SDRAM, there are two groups.
  - All other EBI1 signals (RAS_N, CAS_N, WE_N, CS1_N, CS0_N, CKE1, CKE0, BA1, BA0, and ADR[n:0]) should have similar delays.

To supplement these line-by-line guidelines, a high-level view of the EBI1 connections is shown in Figure 4-5. The memory device should be placed directly adjacent to the EBI1 pins, close to the MDM device, but with enough room for bypass capacitors between the two.

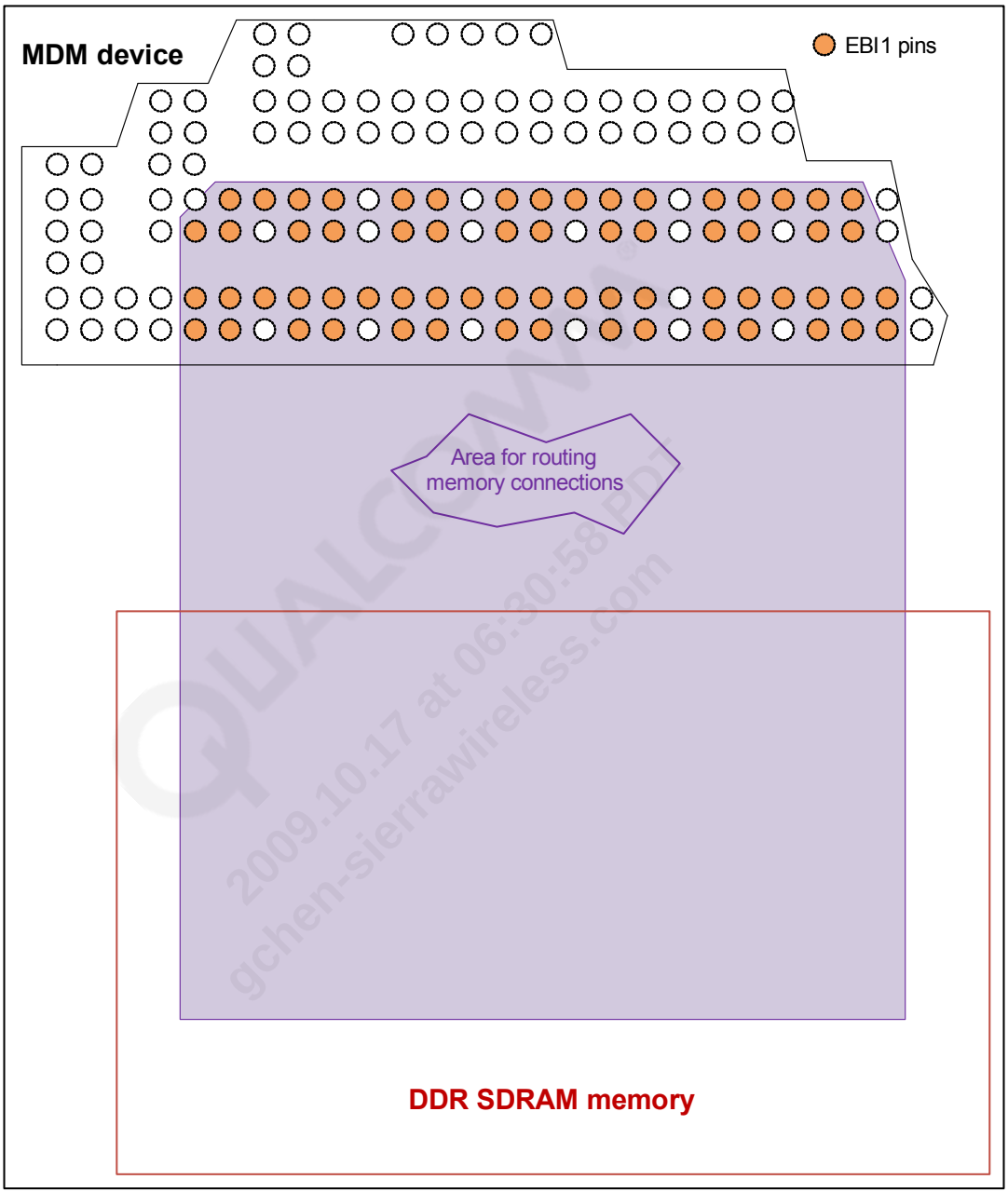


Figure 4-5 EBI1 memory placement and connections

Since the EBI1 memory is one of the most constrained routing areas in the phone, very specific routing guidelines should be followed:

- EBI1 signals should not be routed on surface layers.
  - Due to high clock rates and fast transitions, EBI1 signals will radiate high-order harmonics that can cause RF desensitization issues through EMI. Routing EBI1 signals on inner layers should reduce EMI issues.
  - Breakout will require some layer 1 routing, but these are dropped to inner layers as soon as possible.

- EBI1 signals should be routed adjacent to a solid power fill and ground fill areas to provide proper return paths and maximize shielding.
  - Fill the surface layer between the MDM and memory devices with ground.
  - Fill layer 3 between the MDM and memory devices with power subplane(s).
- Carefully control the function of each PCB layer in the EBI1 area.
- The recommended layer 3 power fill area should begin at the EBI1 portion of the MDM device and extend well into the memory device. Most importantly, this fill area must cover the EBI1 signals so that their entire routing is sandwiched between the layer 1 ground and the layer 3 power fill. If necessary to cover all the listed signals, the power fill may have to extend all the way across the memory device. If coverage does not require the fill to go so far, it can be stopped part of the way under the memory device (this would allow other signals to be routed on layer 3).
- Differential clock signals must be routed on an inner layer with utmost caution due to their fast edges. Provide at least 8 mil separation from all other signals to reduce potential crosstalk.
- Signals not included within the EBI1 bus should be routed with at least 8 mil spacing from all EBI1 signals for adequate coplanar isolation. In particular, sensitive analog and RF signals ***should not be routed near*** EBI1 signals – particularly not on the same layer or an adjacent layer within the vicinity of EBI1 routing.
- Timing and signal integrity simulations should be performed using an actual board routing to ensure that the time specifications of the selected memory device are met. This ensures that any crosstalk, simultaneous switching, or trace-length skew among matched-length nets will not cause timing or signal integrity issues.

In some cases, the EBI1 bus might require routing to multiple devices. If so, all EBI1 traces should be routed using a T configuration, with both traces after the split having equal lengths (Figure 4-6). This helps maintain signal integrity by reducing signal reflections from the loads.

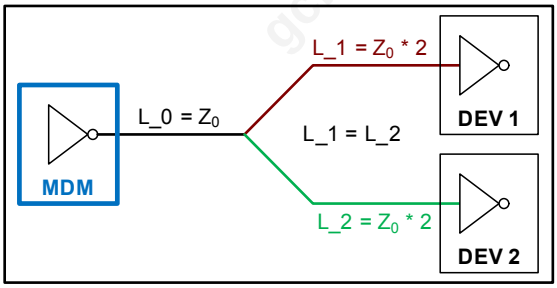


Figure 4-6 T routing to multiple EBI1 devices

4.5 Lower-speed memory interfaces (EBI2)

The EBI2 port connects the slow peripheral devices such as, flash memory devices, and asynchronous SRAM-type devices with the memory controller, which then makes the desired transfers to and from the internal subsystems via the AHB buses.

4.5.1 General EBI2 features

- 1.8 V memory interface support
- Four categories of devices are supported
  - NAND
  - OneNAND
  - Asynchronous devices
- Boot from NAND or OneNAND

Additional features are listed later in application-specific subsections.

4.5.2 EBI2 connections

Similar to EBI1 signals, all EBI2-related pins are located in the same general area of the IC package. This allows optimal routing, and should minimize interference from this bus to more sensitive functions (such as analog baseband).

An example EBI2 interface is shown in Figure 4-7; this example supports NAND flash memory (the boot device), and an asynchronous SRAM-type device.

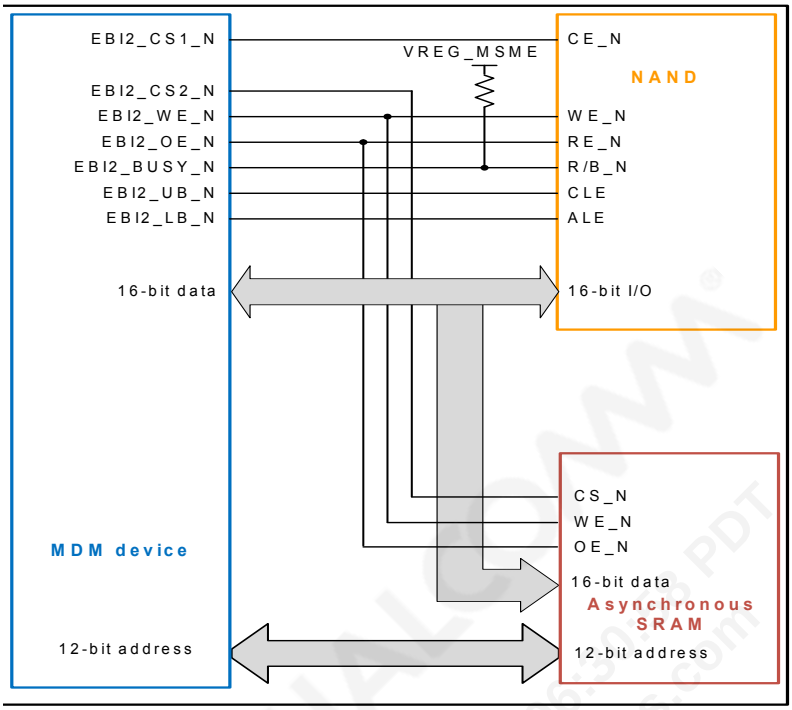


Figure 4-7 EBI2 example application

4.5.3 NAND devices

NAND flash memory devices are supported to improve data-storage capability:

- The generic flash memory interface supports 16-bit and 8-bit NAND devices from manufacturers such as Samsung, Toshiba, etc.
- Programmable page sizes of 512-byte and 2-kbyte (256-byte is not supported).
- The interface performs data transfers one page at a time to and from the flash device. Sequential page accesses are not supported.
- The interface has an internal SRAM buffer for one data page; the ARM processor accesses this internal buffer memory through the AHB bus.
- Supports both MLC and SLC mass-storage technologies

NOTE Although MLC NAND is supported in hardware by MDM devices, software support for these devices is currently not available.

- Error correction coding (Section 4.5.3.4)
- Support NAND flash devices with the following operations:
  - Reset flash device
  - Read status
  - Read ID
  - Read page

- Program page
- Erase block

For more information regarding the flash configuration registers, refer to the *MDM6200 and MDM6600 Mobile Data Modem Software Interface* (80-VR001-2).

4.5.3.1 Booting from flash

During power-up, the MDM device downloads the secondary boot loader from the flash device. The connected device's page size and bus width must be known, otherwise its contents cannot be downloaded. The accepted EBI2 flash devices have these characteristics:

- 512 bytes per page, 8 bits wide
- 512 bytes per page, 16 bits wide
- 2 kbytes per page, 8 bits wide
- 2 kbytes per page, 16 bits wide

4.5.3.2 Support for two flash devices

The serial flash controller treats the whole flash storage space as a series of *pages*. By default, it is configured to operate as though there is just a single device connected. If there are two, the configuration of each device (I/O width and page size) and the boundary between them have to be programmed by software. The boundary is defined as the first page of the second device.

The controller's access to each device is completely page-based; once a page access to a device begins, the access must be completed before the controller can start another page access to the same (or other) device. Therefore, the controller never operates on two devices simultaneously.

4.5.3.3 Flash page arrangements

In flash devices, data is stored in sectors (pages). Any access from the controller to the flash device accesses the entire page. A write to the flash device is always page-oriented. Each page can have multiple codewords, so the controller data transfer is always in multiples of a codeword. A codeword for the controller is 512 bytes of user data plus 10 bytes of parity data. A flash device has room for 512 bytes of user data plus 16 bytes of spare area for the codeword. Each page has one byte (or one word, depending upon the I/O interface size) reserved for the bad block status. To make it simple, the controller always reserves 1 byte (or 1 word) per codeword for the bad block status. The bad-block status location is not fixed; it varies with different flash devices.

Figure 4-8 and Figure 4-9 show the page arrangements of different NAND flash devices. The controller has programmable registers to indicate the bad-block status byte location.

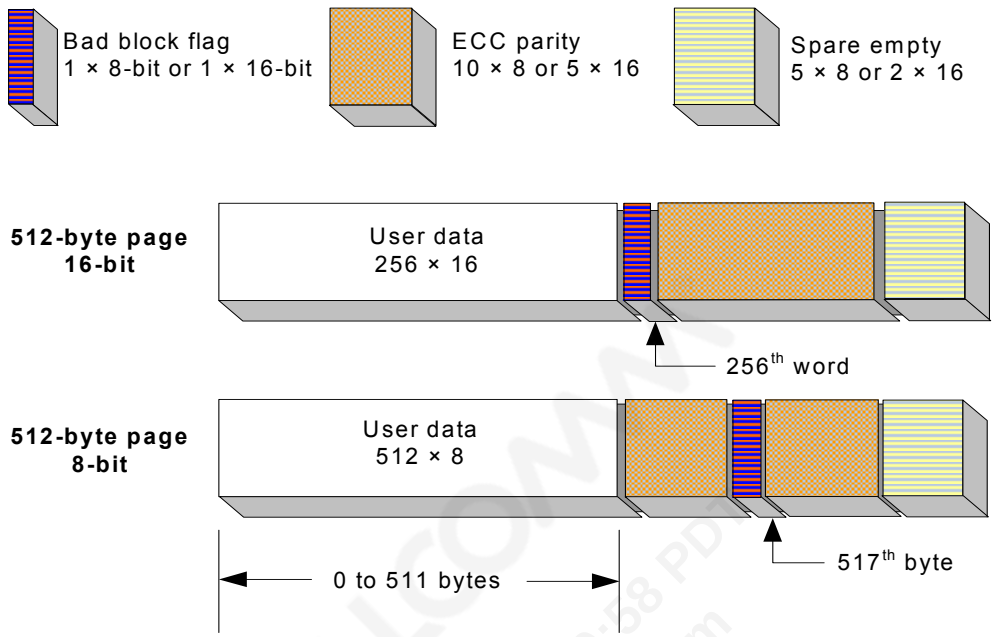


Figure 4-8 Page format description (512 B per page, 8-bit and 16-bit)

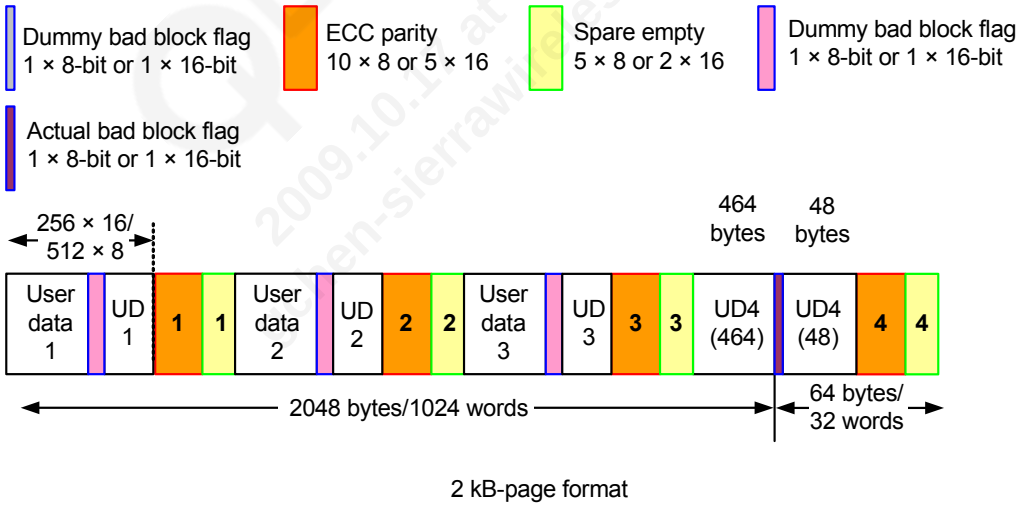


Figure 4-9 Page format description (2 kB per page, 8-bit and 16-bit)

4.5.3.4 Error-correction coding (ECC)

The Reed-Solomon error-correction coding and decoding provided by the controller always uses codewords that cover 512 bytes of user data. Since both 512-byte and 2 kB page sizes are supported, the 512-byte page is arranged to store one ECC codeword, and the 2 kB page contains four.

ECC for 512 B page (8-bit or 16-bit interfaces)

Two types of 512-byte flash devices are supported with Reed-Solomon ECC (Figure 4-8):

- 8-bit wide 512-byte flash device
  - The 512-byte page actually has (512 + 16 =) 528 bytes of storage space.
  - Each page is treated as one ECC codeword space.
  - The bytes are stored in this order: 512 bytes for user data; 5 bytes of ECC parity code; a 1-byte bad-block status; 5 more bytes of parity; and the last 5 bytes are unused.
- 16-bit wide 512-byte flash device
  - The 512-byte page actually has (256 + 8 =) 264 16-bit words of storage space.
  - Each page is treated as one ECC codeword space.
  - The bytes are stored in this order: 256 16-bit words for user data; 1 16-bit word bad-block status; 5 16-bit words of ECC parity code; and the last 2 16-bit words are unused.

NOTE The bad-block indicator is moved to word '0' in the 16-bit interface (compared to byte '5' in the 8-bit interface) when connected to 512-byte per page flash devices.

The ECC allocations are given in Table 4-3.

Table 4-3 ECC allocation – 512 B page flash

Byte or word	ECC allocation	Notes
8-bit width		
Byte_00	ECC Parity_0	10 parity length for Reed-Solomon decoder (8-bit interface)
Byte_01	ECC Parity_1	
Byte_02	ECC Parity_2	
Byte_03	ECC Parity_3	
Byte_04	ECC Parity_4	
Byte_05	Bad block indicator (0xFF)	
Byte_06	ECC Parity_5	
Byte_07	ECC Parity_6	
Byte_08	ECC Parity_7	
Byte_09	ECC Parity_8	
Byte_10	ECC Parity_9	
Byte_11 to Byte_15	Unused	
16-bit width		
Word_00	Bad block indicator (0xFFFF)	
Word_01	ECC Parity_0	5 parity length for Reed-Solomon decoder (16-bit interface)
Word_02	ECC Parity_1	
Word_03	ECC Parity_2	
Word_04	ECC Parity_3	
Word_05	ECC Parity_4	
Word_6 to Word_7	Unused	

ECC for 2 kB page (8- or 16-bit interfaces)

Two types of 512 byte flash devices are supported with Reed-Solomon ECC (Figure 4-9):

- 8-bit wide 2 kB flash device
  - 8 bits x 512 input block length (plus 8 bits x 10 parity length for the decoder)
- 16-bit wide 2 kB flash device
  - 16 bits x 256 input block length (plus 16 bits x 5 parity length for the decoder)

The first codeword is organized as follows:

- 8-bit wide page
  - Four 528-byte sections
  - The bytes are stored in this order: 512 bytes for user data; a 1-byte bad-block status; 10 bytes of parity code; the last 5 bytes are unused.
- 16-bit wide page
  - 4 x 264 16-bit word sections
  - The bytes are stored in this order: 256 16-bit words for user data; a 16-bit word bad-block status; 5 16-bit words of ECC parity code; the last 2 16-bit words are unused.

The second codeword continues as shown in Figure 4-9.

4.5.4 OneNAND devices

OneNAND flash devices are supported in either sync mode; available operations include:

- Reset flash device
- Read page
- Program page
- Erase block
- Lock block
- Unlock block

4.5.4.1 Synchronous operation

Each access to a OneNAND device operating in a synchronous (burst) mode consists of a series of steps. These steps are atomic operations that are common to every access, but depending upon how they are ordered, more complex operations like page reads, page writes, cache reads, synchronous-burst block reads, block erase, and read-while-programming tasks can be performed. Example sequences are shown in Figure 4-10 for page program and page read operations where a single OneNAND page is 2 kB, so each page requires four reads through the 512 B internal data buffer.

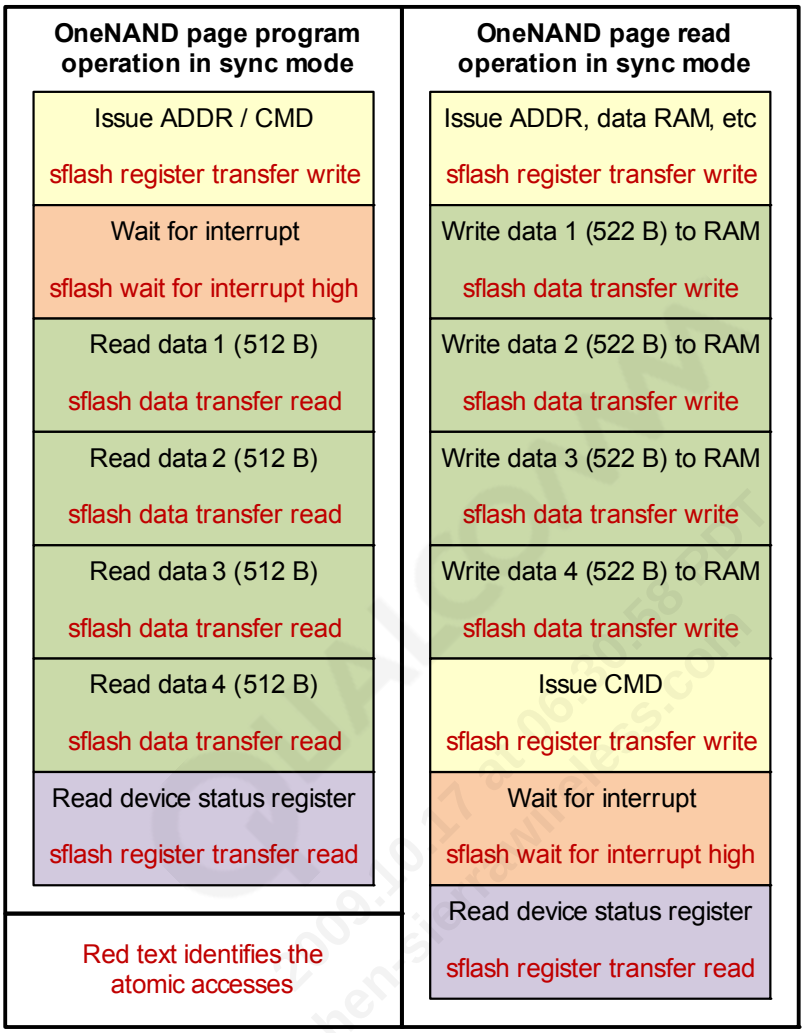


Figure 4-10 Sequences for OneNAND page program and page read operations

All operations can be defined as a sequence of atomic accesses performed by software (instead of predefined sequences in the controller), thereby increasing scalability and reducing verification complexity. The available atomic accesses are:

- Register transfer read – reads data from the flash device and transfers it to any of fourteen 16-bit register sets within the controller
- Register transfer write – transfers data from any of fourteen 16-bit register sets within the controller to the flash device
- Data transfer read – reads data from the flash device and transfers it to the 512-byte data buffer within the controller
- Data transfer write – transfers data from the internal 512-byte data buffer to the flash device
- Wait for interrupt high – polls the interrupt pin from the flash device and indicates to software that it is high
- Wait for interrupt low – polls the interrupt pin from the flash device and indicates to software that it is low

Refer to the *MDM6200 and MDM6600 Mobile Data Modem Software Interface* (80-VR001-2) to program the controller to perform other complex operations using only these six atomic accesses.

4.5.4.2 OneNAND-specific topics

Features that are specific to OneNAND devices include:

- Boot RAM and boot-loader
- Total 4-kByte data RAM buffer
- 16-bit external NOR-like interface
- On-chip error correction
- Command support through on-chip registers
- Support for dual modes (asynchronous mode and burst mode) – AMSS support is required for both modes
- Address and data multiplexed mode support only

To support all the OneNAND device features, software-programmable registers are added. For details, refer to the *MDM6200 and MDM6600 Mobile Data Modem Software Interface* (80-VR001-2).

Example OneNAND device connections to the MDM6x00 IC are shown in Figure 4-11.

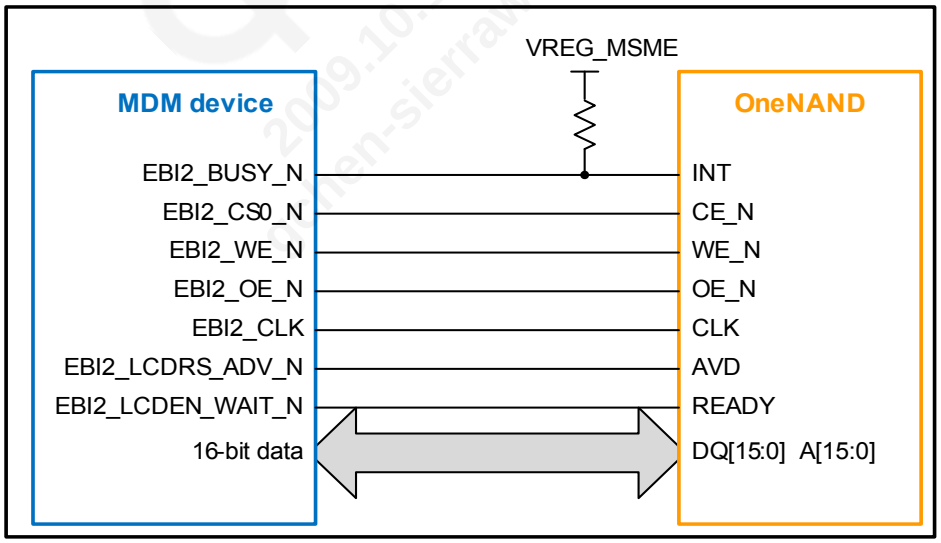


Figure 4-11 Synchronous OneNAND connections

4.5.5 Asynchronous devices

The EBI2 interface supports asynchronous memories and other devices like the UBM receiver. Features include:

- EBI2 supports any generic external peripheral whose interface timing is similar to asynchronous memories.
- Programmable wait states for access, hold, and recovery on all chip selects
- Bus-sizing operations allow WORD, HWORD, and BYTE accesses to 16-bit and 8-bit memory.
- Insertion of recovery/turnover wait states (RECOVERY) to the beginning of the current chip-select access when the previous access was a read to a different chip select, or a when the current access is a write and the previous access was a read to the same chip select
- Addressing to chip-select assertion setup wait states (INIT_LATENCY_RD/WR) at the beginning of the access. This supports some types of pseudo-RAMs on the market by providing the required high time for chip select during consecutive accesses. In general, this feature can be used to provide extra setup time between address and chip-select assertion if needed.
- WAIT_RD and WAIT_WR define the minimum assertion time for read and write, respectively.
- Since the RECOVERY and INIT_LATENCY_RD/WR wait states are applied at the beginning of the access, only the field with the greatest number of wait states takes effect.
- Hold time between WE_N rising and the address/CS_N signals changing (HOLD_RD wait states)
- Hold time between OE_N rising and the address/CS_N signals changing (HOLD_RD wait states), as required by specific types of peripherals. The minimum value (and most commonly used) is HOLD_RD = 0.
- Supports byte-addressable 16-bit devices (UB_N and LB_N signals)
- Page mode is not supported.

4.5.5.1 Configuration registers

EBI2 configuration registers (Table 4-4) must be initialized before the selected chip is used.

Table 4-4 Timing parameters for asynchronous device specifications

Symbol	Write cycle definition	Read cycle definition	Width
i	Initial latency (write cycle) INIT_LATENCY_WR: EBI2_CSn_CFG0[23:16]	Initial latency (read cycle) INIT_LATENCY_RD: EBI2_CSn_CFG0[15:8]	8 bits
h	Write hold HOLD_WR: EBI2_CSn_CFG0[27:24]	Read hold HOLD_RD: EBI2_CSn_CFG1[27:24]	4 bits
r	Recovery cycles [1] RECOVERY: EBI2_CSn_CFG0[31:28]	Recovery cycles [1]	4 bits

Table 4-4 Timing parameters for asynchronous device specifications (cont.)

Symbol	Write cycle definition	Read cycle definition	Width
a_{or}	–	ADV to OE recovery cycles [1] ADV_OE_RECOVERY: EBI2_CSn_CFG1[17:16]	2 bits
T	EBI2 clock cycle		–

1. Recovery cycles are only inserted under certain circumstances, as described in the *MDM6200 and MDM6600 Mobile Data Modem Software Interface* (80-VR001-2). Precharge cycles and recovery cycles may also impact the chip-select time (also described in that document).

4.5.5.2 INIT_LATENCY and WAIT cycles

INIT_LATENCY cycles are inserted in the first access when a particular chip select is enabled for page or synchronous burst operation, or when the access crosses a page boundary. These cycles also refer to the cycles inserted at the beginning of every access to an asynchronous memory. A minimum of one INIT_LATENCY cycle must be programmed for all accesses.

The controller supports independent programming of INIT_LATENCY cycles for read and write accesses.

4.5.5.3 HOLD cycles

HOLD cycles are the extra cycles inserted after every read/write access to an asynchronous memory. A minimum value of 1 HOLD_WR value must be programmed for every write access. The data from the chip is driven from the time WE_N strobe goes low to the time when CS_N goes high. When a hold cycle value of 1 is programmed, then the CS_N stays active for one extra clock cycle.

Read accesses typically do not require HOLD cycles. The external memory controller, however, supports HOLD_RD and uses it to avoid bus contention in multiplexed address/data devices.

4.5.5.4 Recovery cycles

Recovery cycles are required to ensure that there is no contention on the data bus. This value can be obtained from the memory data sheet; it is typically defined as the *amount of time that the memory continues driving the databus after OE_N deassertion.*

4.5.5.5 ADV_OE_RECOVERY

This bit field sets the OE_N timing with respect to ADV_N. A value of 0 means that the ADV_N rising edge and the OE_N falling edge will occur on the same clock cycle. A value of 1 ensures OE_N will be asserted one clock cycle after ADV_N deasserts.

This bit field must be programmed appropriately to avoid bus contention after the chip stops driving the address bus. This address bus is reused as the data bus when supporting multiplexed address/data memories.

NOTE INIT_LATENCY_RD must always be programmed to a value greater than the ADV_OE_RECOVERY value.

4.5.5.6 WE_TIMING

Setting this bit (1) ensures that WE_N remains active for the duration of the burst access. By default, the controller only supports a single clock pulse on WE_N when performing burst-write operations.

4.5.5.7 IGN_WAIT_FOR_WR

This bit causes the EBI2_WAIT_N pin to be ignored during burst-write operation. This is useful for burst memories that do not drive the WAIT_N pin during a burst-write operation. By default, the controller expects the burst memory to drive the WAIT_N, even for a write access. So, this bit must be set to 1 for memories that set the WAIT_N pin to its high-Z state during a burst-write operation.

It is important to program the INIT_LATENCY_WR value to match the write-latency corresponding to the memory device.

NOTE If this bit is programmed to 1, ensure that PAGE_SIZE is programmed to an appropriate value.

4.5.5.8 IGN_WAIT_FOR_RD

This bit causes the EBI2_WAIT_N pin to be ignored during burst-read operation. This is useful for burst memories that do not drive or incorrectly drive the WAIT_N pin during a burst-read operation. By default, the controller expects the burst memory to drive the WAIT_N for a read access. So, this bit must be set to 1 for memories that set the WAIT_N pin to its high-Z state during a burst-read operation.

It is important to program the “INIT_LATENCY_RD” value to match the read-latency corresponding to the memory device.

NOTE If this bit is programmed to 1, ensure that PAGE_SIZE is programmed to an appropriate value.

5 Air Interfaces

This information will be included in future revisions of this document.

6 Connectivity

The MDM6x00 IC supports many interfaces that provide handset users with connectivity to other people and data systems. The following connectivity functions are described in this chapter:

- Universal serial bus (USB)
  - High-speed USB port with integrated physical layer (PHY)
  - USB-UICC port with support for host-mode functionality
- Secure digital (SD) ports
- General serial bus interface (GSBI) ports – five ports that are exclusively used with external interfaces of the MDM device are capable of supporting the following serial protocols:
  - Universal asynchronous receiver transmitter (UART) serial ports
  - User identity module (UIM) ports
  - Pulse-code modulation (PCM) interface
  - Inter-integrated circuit (I^2C) interface
  - Inter-IC sound (I^2S) interface
  - Serial peripheral interface (SPI)

  The GSBI block is described in its own section, followed by a separate section for each available configuration.
- Near field communicator (NFC)

6.1 USB interfaces

The universal serial bus (USB) is an interconnection standard widely supported by the electronics industry. The USB 2.0 specification defines three operating data rates: low-speed (1.5 Mbps); full-speed (12 Mbps); and high-speed (480 Mbps).

When two devices are connected via a USB interface, one device must act as a host, and the other device must act as a peripheral. The host is responsible for initiating and controlling traffic on the bus. The USB specification requires personal computers (PCs) to act as hosts, and other devices such as printers, keyboards, and mice to act as peripherals. The OTG supplement of the USB 2.0 specification creates a new class of devices called OTG devices. OTG devices can act as either hosts or peripherals, depending on how they are connected and/or used. The MDM6x00 device supports host and peripheral modes within the OTG supplement, with the exception of session request protocol (SRP) and host negotiation protocol (HNP).

The MDM6x00 IC supports one HS USB (USBH) port with built-in PHY, and one FS USB-UICC port. Links to this and related specifications can be found at www.usb.org/developers/.

The MDM6x00 IC supports USB interfaces using two controllers:

- The primary controller is the high-speed USB (HS-USB) port with an integrated physical layer (PHY). This HS-USB port is also capable of supporting USB operations at low speed and full speed (Section 6.1.1).
- The secondary controller is the full-speed USB-UICC port, which only supports host-mode functionality (Section 6.1.2).

Additional USB information is available at www.usb.org/developers/.

6.1.1 High-speed USB port with integrated PHY

The high-speed USB module contains an embedded UTMI+ core with a built-in transceiver (eliminating the need for an external PHY). The HS-USB port is a standard 4-pin interface that connects directly to the USB connector (USBPHY_DP, USBPHY_DN, USBPHY_ID, and USBPHY_VBUS). Two additional pins are required for PHY operations: an external reference resistor pin (USBPHY_REXT); and a USB system-clock pin that the USB PHY uses to lock its internal PLL (SYS_CLK). This interface and architecture are shown and described in Figure 6-1.

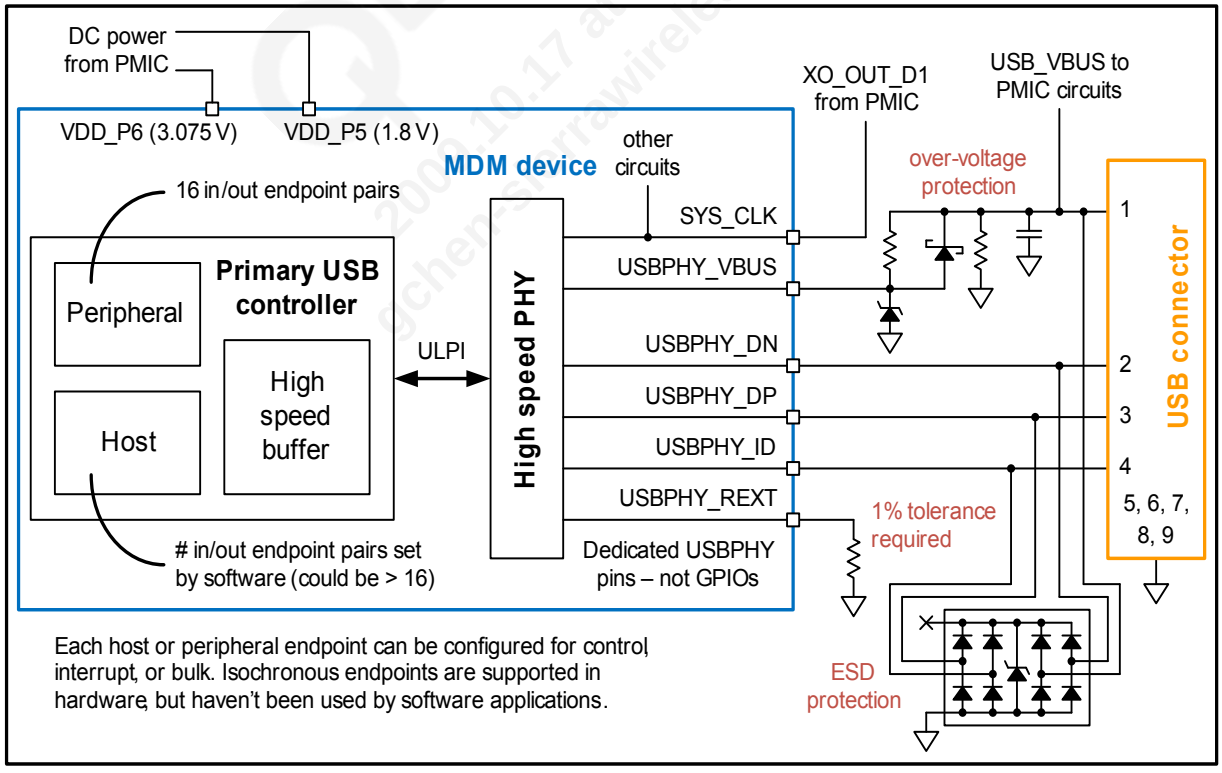


Figure 6-1 HS-USB connections and architecture

With the appropriate software, the HS USB hardware supports many applications, including:

- Peripheral mode

- Mass storage
- MTP for music and video content transfers
- CDC/ACM for modem
- CDC/ECM for data services
- CDC/OBEX for NMEA and diagnostic
- SICD for PictBridge

- Host mode
  - HID supporting keyboard
  - Mouse and gamepad controller connectivity
  - Mass storage supporting USB flash drive and HDD connectivity

6.1.2 USB-UICC port

The MDM device supports USB subscriber identity module (SIM) operation, also known as USB universal integrated circuit card (USB-UICC). An example application is shown in Figure 6-2, with the PMIC and another external IC providing level translations between the 1.8 V and 2.85 V domains. The secondary USB controller has the same logic and software interface as the primary USB port, but does not have the integrated high-speed PHY. Key limitations are noted below:

- Full-speed transceiver interface and smaller buffers limit the speed at 12 Mbps (FS-USB).
- Supports the USB-UICC host mode only

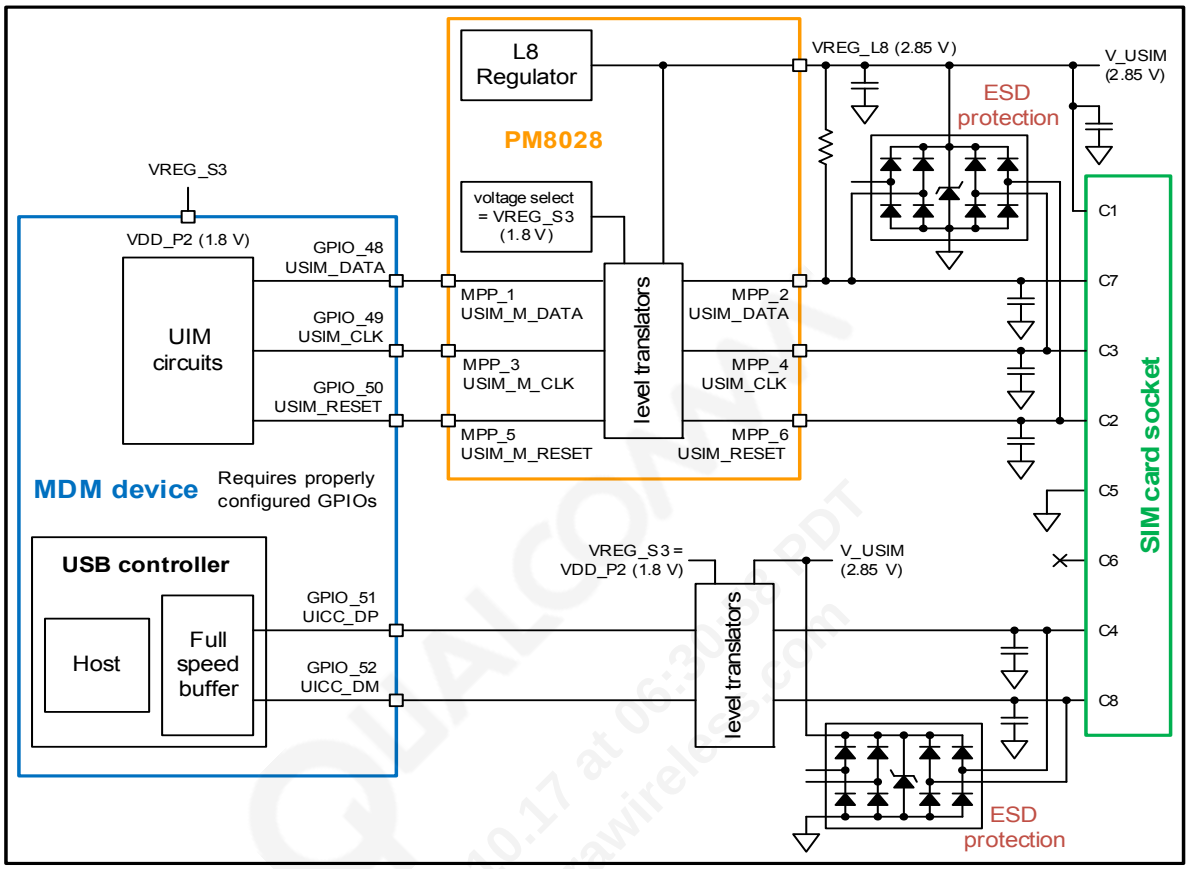


Figure 6-2 Example USB-UICC application

6.2 Secure digital (SD)

Secure digital (SD) products are flash-based memory cards specifically designed to meet the security, capacity, performance, and environmental requirements of popular audio and video consumer devices. Their physical form factors, pin assignments, and data transfer protocols are forward-compatible with the multimedia card (MMC), but with higher bandwidth, copyright protection (via secure version), and functional expansion through secure data input/output (SDIO) capability. SDIO enables external devices to communicate with a host like the MDM6x00 IC through its SD interface. An example MDM SDIO application is 802.11 WLAN support.

The MDM6x00 IC provides up to three SD interfaces (Table 6-1) that provide:

- Clock output up to 52 MHz
- 2.85 V operation on SDCC1; 1.8 V operation on SDCC2 and SDCC3
- SDIO host and client and modes are supported
- SDIO compatible WLAN (802.11)
- Interface with SD/MMC memory cards up to 32 GB
- Internal 10k to 100k pull-up resistors on the command and data pins; see the *Application Note: Multimedia Card/Secure Digital Card* (80-V7837-1)

Table 6-1 Supported SD interfaces

Standard	Compatible MDM SD ports [1]		Maximum clock frequency
	1-bit	4-bit	
SDIO v2.0	SDC1, 2, 3	SDC1, 2, 3	50 MHz
SD v2.0	SDC1, 2, 3	SDC1, 2, 3	50 MHz
MMC v4.2	SDC1, 2, 3	SDC1, 2, 3	52 MHz

1. See Section 6.2.1 for SDC pin assignments and an example application.

The secure digital card controller architecture is illustrated and explained in Figure 6-3.

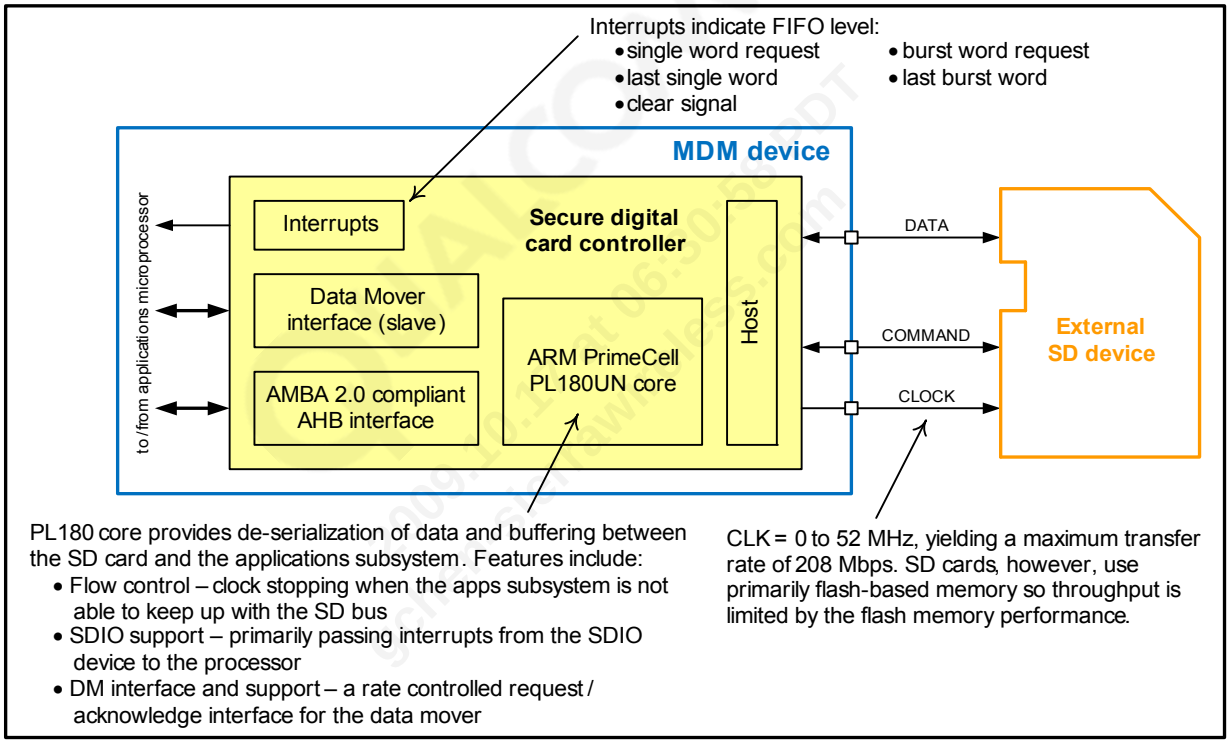


Figure 6-3 SD card controller architecture

6.2.1 Secure digital connections

The three secure digital ports are shown in Figure 6-4. This figure also shows the SD connections as implemented within the MDM reference designs.

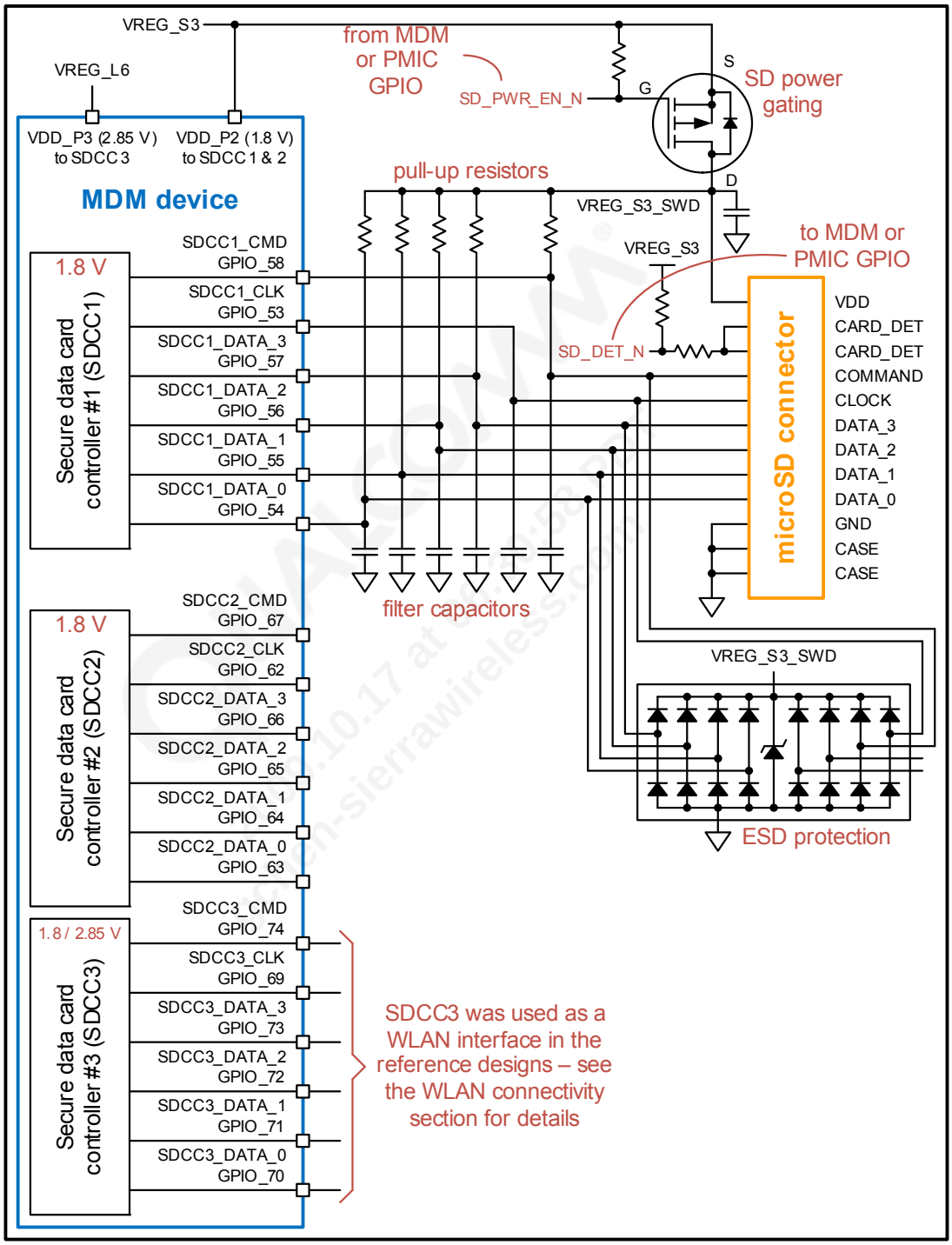


Figure 6-4 Secure digital connections

6.2.2 SD interrupts

When an SD port is used to interface with an SDIO card, a MDM or PMIC GPIO is used to pass SD interrupts to the MDM processor.

6.2.3 SD clocks

Two clock sources are required for the SD card controllers:

- HCLK for all system bus interfacing, DMA, and FIFO control logic
- MCI_CLK (SDCCx_CLK) for the SD interface (external) and other control logic

The SDCCx_CLK has to be 52 MHz or less. The SD card controller inputs read data on the rising edge of SDCCx_CLK and outputs write data on the falling edge of SDCCx_CLK. Further information is available in *Application Note: Multimedia Card/Secure Digital Card* (80-V7837-1).

6.3 General serial bus interface (GSBI) ports

The MDM device includes five four bit GSBI blocks that can be configured to support combinations of six different serial protocols (Table 6-2).

Table 6-2 GSBI configurations

GSBI option	Configuration	Interface details
1	4-pin UART	Section 6.4
2	2-pin UART + 2 GPIOs	Section 6.4 and Chapter 8
3	3-pin UIM + 1 GPIO	Section 6.5 and Chapter 8
4	4-pin SPI (full duplex)	Section 6.9
5	3-pin SPI + 1 GPIO	Section 6.9 and Chapter 8
6	4-pin I^2S (half duplex)	Section 6.8
7	3-pin I^2S + 1 GPIO	Section 6.8 and Chapter 8
8	4-pin AUX_PCM	Section 6.6
9	2-pin I^2C + 2-pin UART	Section 6.7 and Section 6.4
10	2-pin I^2C + 2 GPIOs	Section 6.7 and Chapter 8
11	4 GPIOs	Chapter 8

NOTE To assign GSBIs to particular functions (such as the options listed in Table 6-2), designers must identify all their application's requirements and map each GSBI to its function – carefully avoiding conflicts in their assignments.

The five GSBI ports can be diagrammatically represented as shown in fig Figure 6-5.

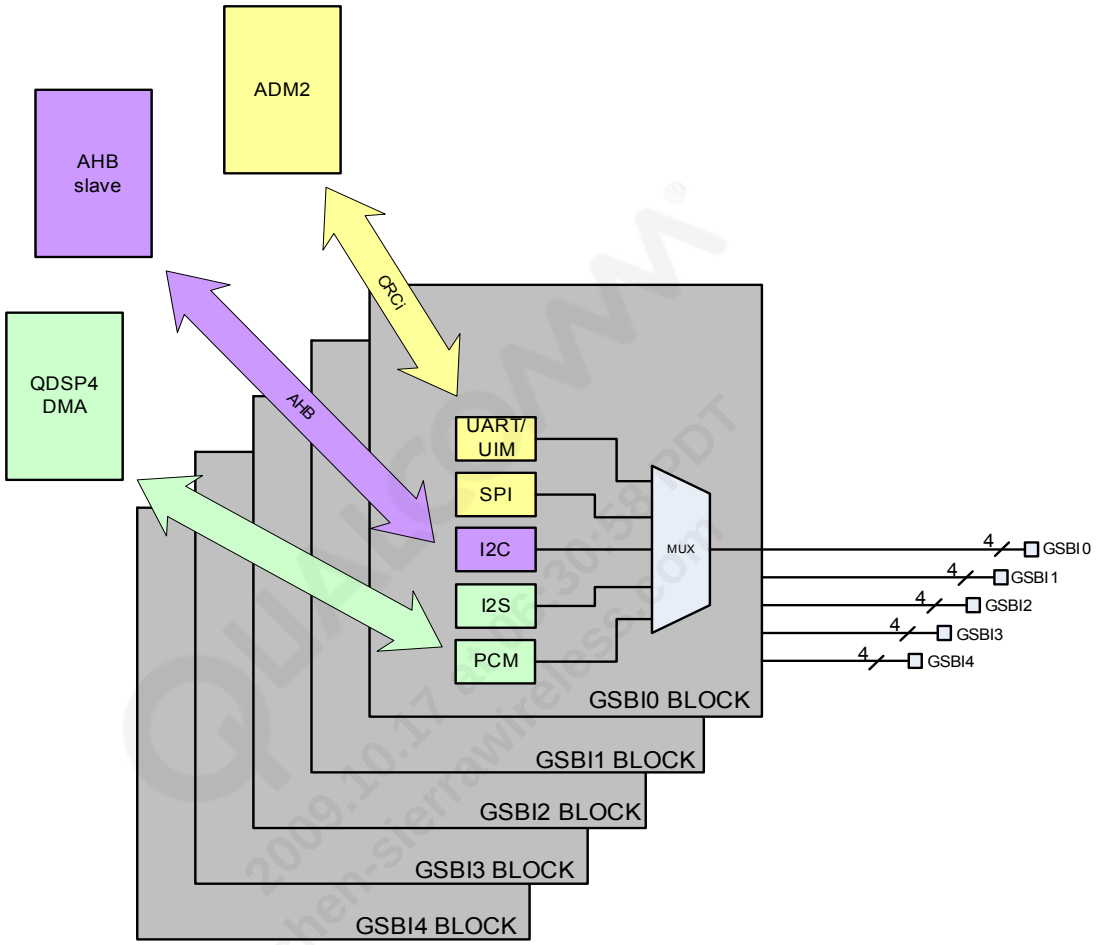


Figure 6-5 GSBI block diagram

- Each of the GSBI blocks can be independently and simultaneously configured as a UART, SPI, I^2C, I^2S, PCM and UIM interface.
- The UART and SPI blocks use the CRCi bus to connect to ADM2
- The I^2S and PCM block use the QDSP4 DMA
- The I^2C block is connected to the AHB bus
- The UIM block a separate UART DM

6.4 Universal asynchronous receiver transmitter (UART)

When a GSBI is configured for UART operation, it can be used for applications such as:

- Serial data interface that conforms to the RS-232 interface protocol
- Handset's serial data port for test and debug

- External keypad or ringer interface
- With flash memory – used to load and/or upgrade system software

All are capable of basic UART operation (64-byte FIFO, up to 230 kb/s rate) or high-speed operation (TBD-byte FIFO, up to 4 Mbps rate).

High-level UART features include:

- Hardware handshaking
- Programmable parameters
  - Data size
  - Stop bits
  - Parity
  - Bit rate
  - Clock source
  - High-level functions such as enable/disable, start break, stop break, and reset

UART details are presented in the rest of this section.

6.4.1 UART connections

The reference design did not include any external UART connections. A generic UART interface is shown in Figure 6-6.

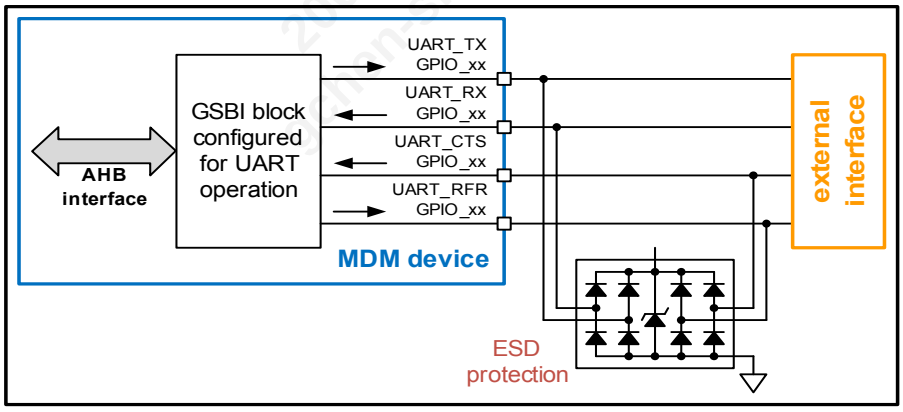


Figure 6-6 Example UART connections

6.4.2 UART setup

Since the GSBI blocks can be used for protocols other than UART, some setup is required. Details will be provided in a future revision of this document.

NOTE 64-byte high-speed FIFO has been confirmed thus far. The final HS FIFO size will be stated in future revisions of this document

6.4.3 Basic UART

The UART (Figure 6-7) processes transmit and receive data with separate Tx and Rx channels. The receive path begins with serial data from the UART_RX pin, and the Rx channel provides a serial-to-parallel conversion. Conversely, the UART Tx channel provides a parallel-to-serial conversion and outputs serial data to the UART_TX pin. Supporting circuits include channel controller, registers, clock generator, bit-rate generator, and bus interface.

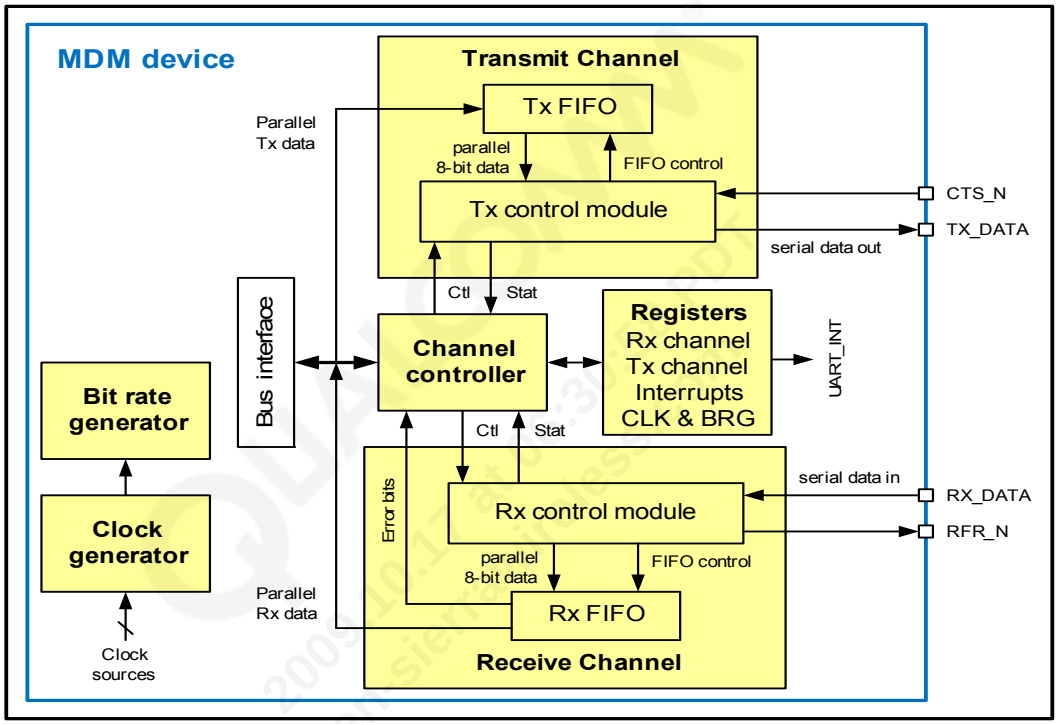


Figure 6-7 Basic UART functional block diagram

6.4.3.1 UART transmitter

The main UART transmit-channel components are its FIFO and control module. Normal operation is illustrated and explained in Figure 6-8.

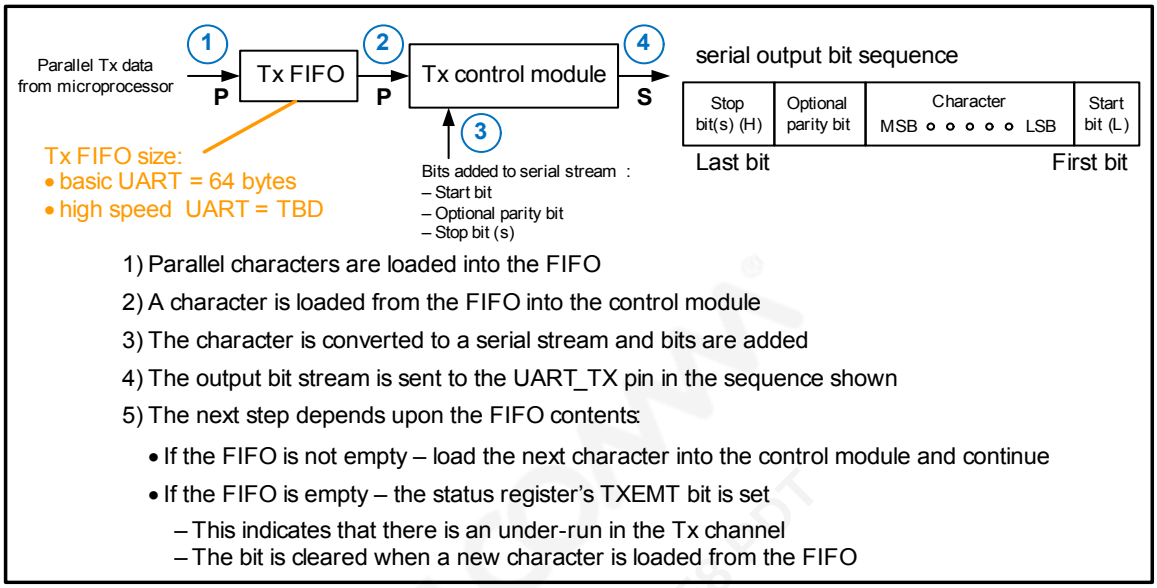


Figure 6-8 Normal UART Tx operation

Further UART Tx operation depends upon its registers, as summarized in Table 6-3.

Table 6-3 UART Tx registers and operational description

Register type / key bits	Description
Command register	■ Enable or disable the Tx channel □ If a character is waiting in the Tx FIFO when enabled, normal operation begins (Figure 6-8). □ When disabled, the Tx channel continues transmitting until all loaded characters are sent. When transmission ends, the UART_TX pin goes into a marking state. [1] ■ Start break command – after the Tx channel transmits all the characters in the FIFO, the UART_TX pin is forced low. ■ Stop break command – ends the break, thereby allowing the UART_TX pin to change from its low state. ■ Reset transmitter command – forces the Tx channel to cease; the UART_TX pin enters its marking state and the Tx FIFO is flushed.
Status register TXRDY bit TXLEV bit TXEMT bit	Set whenever the Tx FIFO has space available. Asserted when the Tx FIFO has fewer characters than the number programmed into the Tx FIFO watermark register (below). If the FIFO is empty when Tx is enabled, this bit is set (1).
Tx FIFO watermark register	Number of characters in the FIFO before a warning is generated

1. Idling or disabling the Tx channel holds the UART_TX pin in a high state – its *marking* state.

UART Tx operation is supplemented by the clear-to-send (CTS) control bit:

- The CTS control feature is turned on via the UART mode register 1 (MR1).
- When on, the Tx channel checks the CTS_N input before transmitting a character.

- If CTS_N is high, the Tx channel stops transmitting and continues marking.
- If CTS_N is low, transmission begins or continues.

- If CTS_N goes high in the middle of character transmission, the Tx channel waits for a completed transmission before entering the marking state.
- The Tx channel can generate a programmable interrupt whenever CTS_N changes states.

6.4.3.2 UART receiver

The main UART receive-channel components are its FIFO and control module. Normal operation is illustrated and explained in Figure 6-9.

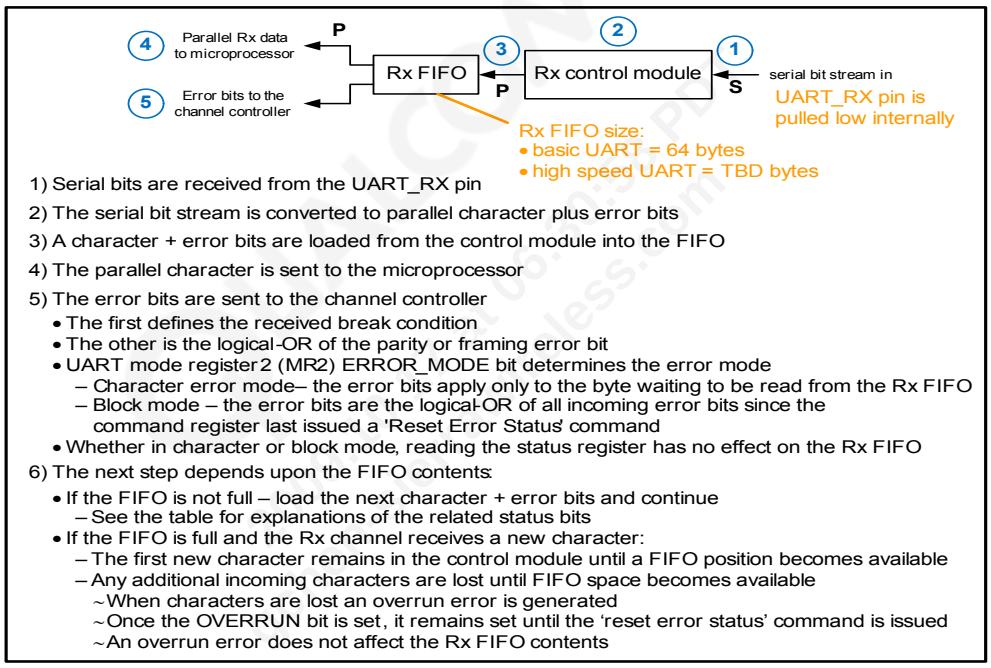


Figure 6-9 Normal UART Rx operation

Further UART Rx operation depends upon its registers, as summarized in Table 6-4.

Table 6-4 UART Rx registers and operational description

Register type / key bits	Description
Command register	■ Enable or disable the Rx channel □ If a serial bit stream is applied to the UART_RX pin when enabled, normal operation begins (Figure 6-9). □ When disabled, incoming characters are ignored. Characters already in the Rx FIFO remain unchanged and can be read by the microprocessor. The channel remains disabled until the enable command is issued. ■ Reset error-status command – resets the error bits received since the previous *reset error-status* command was issued. ■ Reset receiver command – forces the Rx channel to cease reception; the Rx FIFO is flushed and the status bits are cleared.
Status register	
RXRDY bit	Set whenever the control module writes a character into an empty Rx FIFO; clears when the FIFO becomes empty again.
RXFULL bit	Set when the Rx FIFO becomes full; cleared when a character is read out.
RXLEV bit	Asserted when the Rx FIFO has more characters than the number programmed into the Rx FIFO watermark register (below).
RXSTALE bit	If a character is waiting in the FIFO longer than the timeout interval (below), this interrupt is generated.
RXHUNT bit	Asserted when the hunt character (below) is received.
OVERRUN bit	Asserted when the FIFO and control module are full and characters are lost; remains set until a *reset error-status* command is issued.
Rx FIFO watermark register	Number of characters in the FIFO before a warning is generated
STALE_TIMEOUT register	Programmable wait interval for how long a character is in the FIFO
Hunt character register	Character that the Rx channel is hunting (searching for)

UART Rx operation is supplemented by the ready-for-receive (RFR) status bit:

- The automatic RFR feature is turned on via the UART mode register 1 (MR1).
- When on, the Rx channel checks the Rx FIFO level to determine the RFR_N state.
  - If the Rx FIFO level is the same or greater than the value programmed in MR1, RFR_N is forced high.
  - If the Rx FIFO level is below the programmed level, RFR_N is forced low.
- This feature prevents overruns when the MDM RFR_N pin is connected to the transmitting device CTS_N pin.

6.4.3.3 UART clock source and bit-rate generator

Each basic UART block includes the clock source and bit-rate generator illustrated and described in Figure 6-10; this figure includes an example calculation.

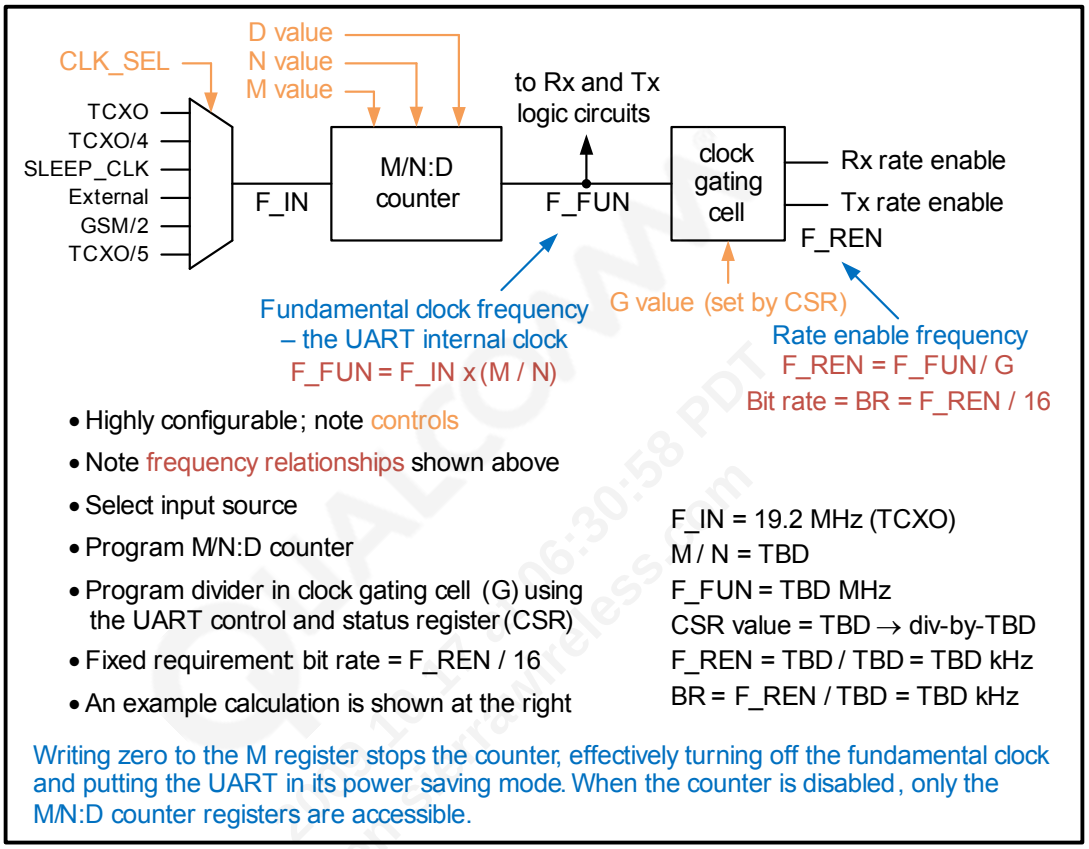


Figure 6-10 Basic UART clock source and bit-rate generator

A generic explanation of M/N:D counters is provided in Section 3.7.3. In addition:

- Once the input source is selected, the M/N ratio is calculated.
  - For basic UART operation, 1.8432 MHz is recommended for the fundamental clock rate.
- Select the M and N values.
  - Note that the ideal M and N values might exceed the register width, and alternate values would need to be selected.
- Select the D value to set the duty cycle.
- Once these values are selected, the actual register values are calculated. To simplify the hardware, the N register is loaded with the NOT(N-M) value; this conversion is provided by the script.

Additional UART-related programming details will be included in a future revision of this document (CSR values and clock selection).

6.4.3.4 UART interrupts

An interrupt status register (ISR) for each UART reports status through six separate bits:

- TXLEV
- RXLEV
- RXSTALE
- RXHUNT
- RXBREAK
- DELTA_CTS

An interrupt mask register (IMR) includes bits for the same six functions; the ISR status bits can be enabled or disabled by setting the corresponding bits in the IMR. A third register, the mask interrupt status register (MISR), returns the bit-wise AND of the ISR and IMR registers.

Conditions for the RXSTALE interrupt can be defined by the user on the UART_IPR register.

6.4.4 High-speed UART

Much of the information presented earlier for the basic UART block applies to the high-speed UART (UART with data mover) also. Additional UARTDM material is presented below.

The UARTDM is used to support high-speed UART operation up to 4 Mbps and medium data-rate IrDA operation up to 1.152 Mbps. The basic UART blocks cannot operate at these data rates. Other advantages of the UARTDM blocks include:

- Rate-controlled data mover with separate CRCI channels for Rx and Tx
- Larger Rx and Tx FIFOs that are implemented in one SRAM
  - 512 bytes vs. 64 bytes
- Access to the fast peripheral bus (32-bit wide AHB interface) rather than the slow bus
- Maintains traditional level interrupts directly to the microprocessor when the data mover is not available

The UARTDM architecture is illustrated and explained in Figure 6-11. Note that the UARTDM Tx and Rx channels are similar to the basic UART channels, except that the FIFOs are implemented in SRAM and the FIFO controls and IRQ generation are in the DM controller block.

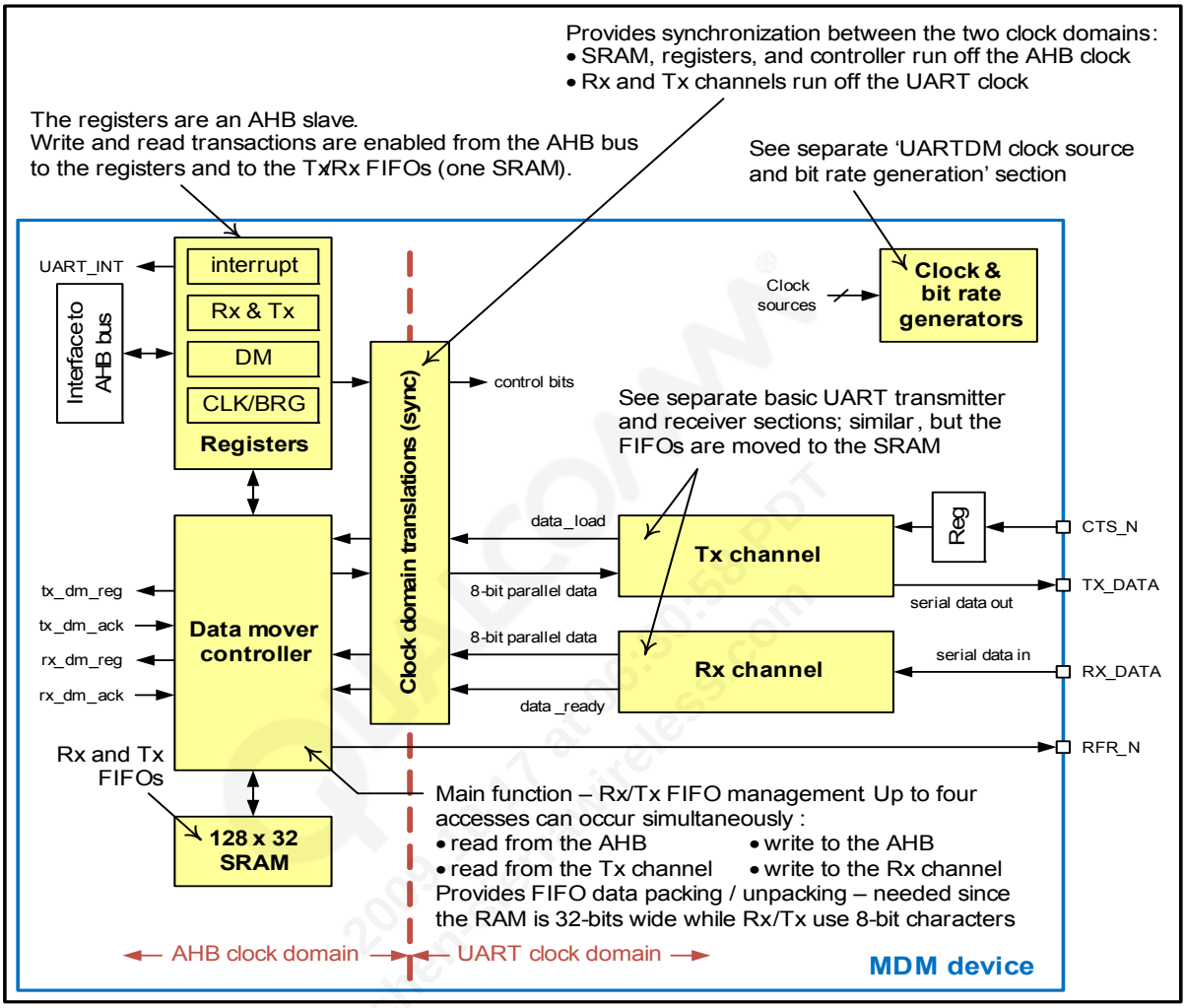


Figure 6-11 High-speed UART (UARTDM) architecture

6.4.4.1 UARTDM clock source and bit-rate generator

Most of the material presented in Section 6.4.3.3 applies to the UARTDM as well. The differences will be described in future revisions of this document.

6.4.4.2 UARTDM operation

The UARTDM is connected to the AHB bus and includes two rate-controlled (CRCI) data mover interfaces (one for Rx transfers and one for Tx transfers). All configuration registers are in the AHB clock domain, thereby optimizing the AHB bus utilization throughout the configuration phase. Most registers are configuration registers that do not require synchronization; they undergo a soft reset along with the UART block after the configuration phase, and are enabled after a transfer.

The Rx path is totally independent from the Tx path, so the UART block can receive data during an active Tx transfer. The only Rx/Tx commonality is the shared SRAM that implements the FIFOs, and the two memory spaces are not necessarily equal.

Four methods can be used to set up a Tx or Rx transfer; two with the CRCI DM interface and two without:

1. Tx transfers without the Tx DM CRCI interface
   a. Initialize the UART.
   b. Soft reset the UART.
   c. The UART sends an interrupt each time the FIFO holds less than a preprogrammed number of characters. This interrupt signals the microprocessor that a new data burst can be sent to the UART Tx block.
   d. The UART transmits character-by-character as long as there is a valid character in the transmit FIFO.
2. Tx transfer with the Tx DM CRCI interface
   a. Initialize the UART.
   b. Soft reset the UART.
   c. Initialize the DM and make sure that the EBI memory includes the transfer.
   d. Enable the Tx DM mode, and then initialize the UART's TX_data_length counter.
   e. The UART sends a tx_dm_req to the DM each time there is enough space for a new data burst in the transmit FIFO. The DM responds to the request by sending an acknowledge, and then sends the data burst via the AHB bus to the Tx FIFO. The UART continues to send requests until the TX_data_length counter is zero; the transfer is then complete.
   f. The UART transmits character-by-character as long as there is a valid character in the Tx FIFO.
3. Rx transfer without the Rx DM CRCI interface
   a. Initialize the UART.
   b. Soft reset the UART.
   c. The UART starts collecting characters as they are received via the receive channel and writes them to the Rx FIFO. The UART sends an interrupt each time the FIFO holds more than a preprogrammed number of characters (level interrupt), or whenever no characters are received over a preprogrammed duration while the FIFO is empty (stale interrupt). This interrupt signals the CPU that newly received data can be read from the Rx FIFO.

4. Rx transfer with the Rx DM CRCI interface
   a. Initialize the UART.
   b. Soft reset the UART.
   c. Initialize the DM.
   d. Enable the Rx DM mode.
   e. Enable the UART Rx path.
   f. The UART starts receiving characters, and a request is sent to the DM each time the Rx FIFO is ready for a new data burst. After each character transfer, the stale timer is loaded; when a stale timeout occurs (no new characters were received), the last burst is filled with non-valid characters and a request is sent to the DM. An interrupt is sent to the microprocessor immediately after transferring the last burst to the DM. The UART also stores the number of valid characters received up to the point that the stale interrupt bit was cleared or reset.

6.5 User identity module (UIM)

MDM support of USB-UICC operation uses dedicated USIM pins, as described in Section 6.1.2. In addition, one or more of the GSBI blocks can be configured for UIM operation, as described in this section.

The MDM6x00 IC supports user identity modules for all its air interfaces. In GSM and UMTS applications, the MDM interfaces with a subscriber interface module (SIM for GSM or USIM for UMTS). To support dual-voltage UIM modules, the PMIC provides the necessary level translation from 1.8 to 2.85 V (as shown in Section 6.1.2).

6.5.1 UIM connections

The reference design did not include any UIM connections. A generic UIM interface is shown in Figure 6-12. An example that includes PMIC-level translation is shown in Figure 6-2.

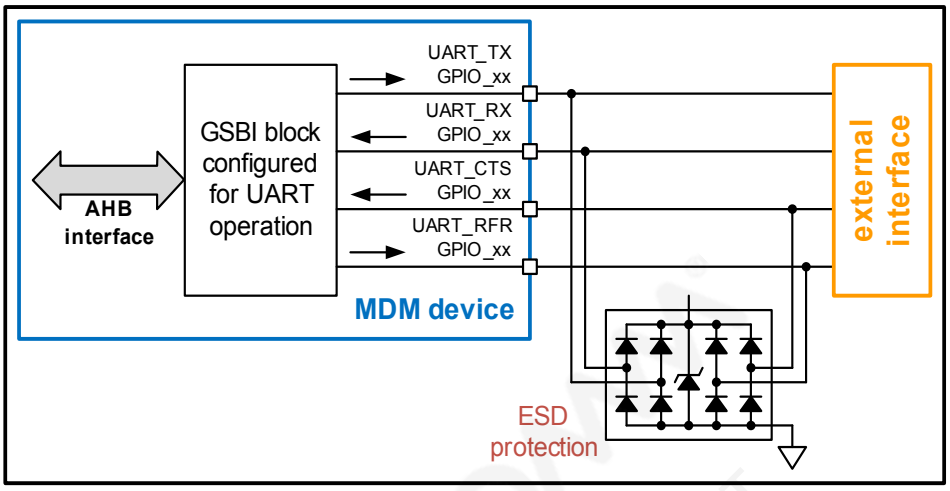


Figure 6-12 Example UIM connections

6.5.2 UIM setup

Since the GSBI blocks can be used for protocols other than UIM, some setup is required. Details will be provided in a future revision of this document.

6.5.3 UIM clock-source selection

The UIM block uses the same clock source as the basic UART block (Section 6.4.3.3). For UIM operation, the selected clock source is either used directly or divided by two. For ISO/IEC 7816-3 compliance, the highest frequency configuration (3.25 MHz) uses the GSM/2 source and divides it by two. UIM clock configuration is controlled by three bits of the SIM_CFG registers (Table 6-5).

Table 6-5 UIM clock selection – register details

Control bit within SIM_CFG register	Setting and functional description
7	UIM clock enable 0 = disable UIM clock and hold per bit 5 1 = enable UIM clock at selected frequency
6	UIM frequency divider 0 = divide-by-1 (use selected source directly) 1 = divide the selected source by 2
5	UIM clock off state 0 = hold UIM clock output low when disabled 1 = hold UIM clock output high when disabled

6.5.4 UIM initialization

The MDM device can recognize and initialize a UIM ***only during its power-up sequence***, not during regular operation.

During a power-up sequence, whether a true power-up or a soft-reset, the UIM clock and data lines are active as they execute their initialization process one UIM slot at a time (if more than one is used). After initialization, the slots' operation depends upon whether a module is detected or not:

- If a module is detected:
  - The data line stays *active*, even if data is not being transmitted (between accesses). It maintains its marking state (logic high) between accesses.
  - The clock is only active during accesses; it is turned off between accesses to save power. The state of the clock when off is programmable (Section 6.5.3), and must be selected to support the module's characteristics.
  - Even though the clock is turned off between accesses, the interface is still active. The data, reset, and power lines all remain high (assuming an active-low reset).
  - Note that the interface stays on once the module is detected, even during MDM sleep modes; the current consumption continues.
- If a module is not detected (module not inserted or not recognized, broken connection, etc.):
  - The interface is deactivated.
  - All lines are low and there is no chance to operate or communicate with a module until the next power-up sequence.

6.6 PCM interfaces

The PCM interface can be used to connect to an external codec in one of two distinct modes:

- primary PCM (short sync) or
- auxiliary PCM (long sync).

Both modes support 8-bit A-law and m-law, and also 16-bit linear data formats; they both connect to an external device via a 4-pin interface. The MDM device can serve as either master or slave in its primary mode, but must be the master in its auxiliary mode (slave mode is not supported).

6.6.1 PCM connections

The reference design did not include any external PCM connections. Some generic PCM interface examples are shown in Figure 6-13. Note that the source of the clock and sync signals depends upon whether the MDM device is operating as the master or slave.

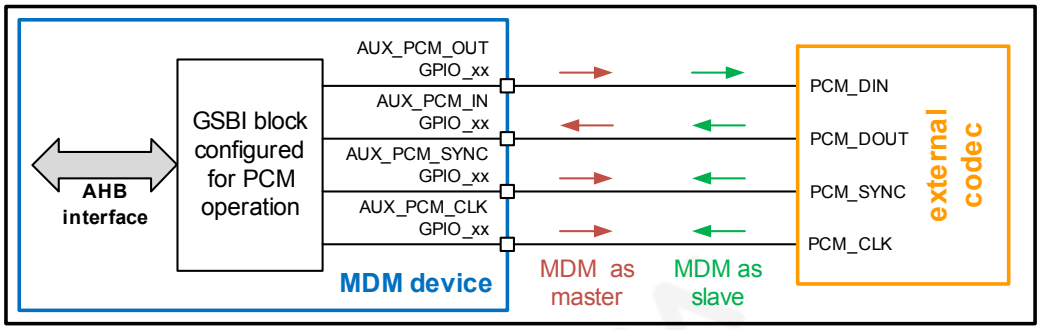


Figure 6-13 Example PCM connections

6.6.2 PCM setup

Since the GSBI blocks can be used for protocols other than PCM, some setup is required. Details will be provided in a future revision of this document.

6.6.3 Primary PCM (short sync) mode

Primary PCM is a slot-based interface where a particular user transmits or receives data within one available slot (Figure 6-14). 8-bit A-law and μ-law, and also 16-bit linear data formats are supported. The slot duration (data width) can be either 8 or 16 cycles. The data is sampled on the falling edge of the PCM_CLK and transmitted on the rising edge; the PCM_SYNC *falling edge* represents the MSB.

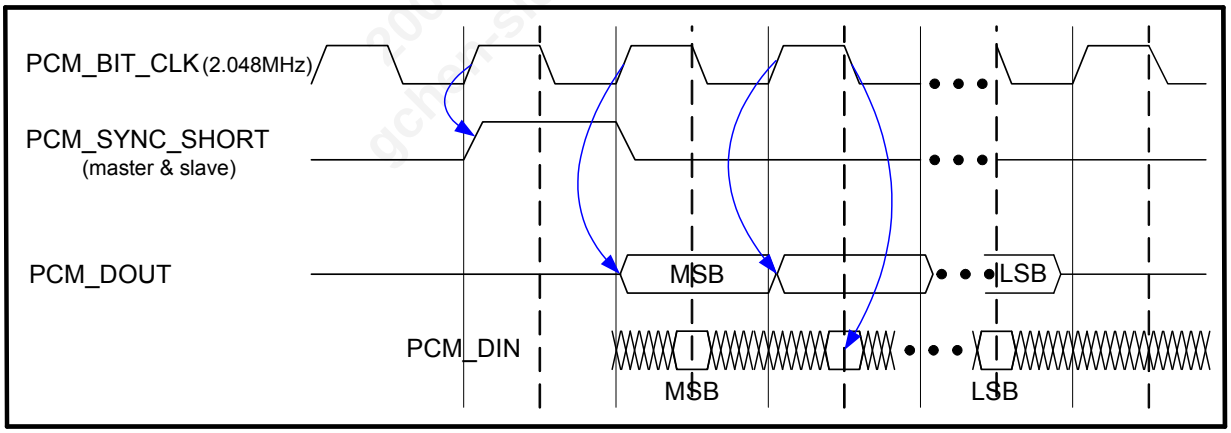


Figure 6-14 Key primary PCM timing relationships

6.6.3.1 Master mode

In master mode, the frame length is software programmable (AUX_CODEC_CTL[14:13]) and can be configured for 256, 128, 64, or 32 PCM_CLK cycles to allow support for different clocks and wideband speech codecs. The PCM_SYNC and number of slots can be calculated as follows:

Number slots = frame length / bit width; PCM_SYNC = PCM_CLK / frame length

Example:

PCM_CLK = 2048 kHz; frame length = 256

Number of slots = 256/8 = 32 slots for A/μ law; or 256/16 = 16 slots for linear format

PCM_SYNC = 2048 kHz/256 = 8 kHz

6.6.3.2 Slave mode

In slave mode, the PCM_CLK and PCM_SYNC are provided by the external device. PCM clock rates of 2048, 1024, 512, 256, and 128 kHz are supported for an 8 kHz sync. The number of slots and frame length can be calculated as follows:

Frame length = PCM_CLK / PCM_SYNC; number of slots = frame length / bit width

Example:

PCM_CLK = 256 kHz, PCM_SYNC = 8 kHz

Frame length: 256 kHz / 8 kHz = 32 cycles

Number of slots: 32/8 = 4 slots for A/μ law; or 32/16 = 2 slots for linear format

Figure 6-15 illustrates the slot and data processing within an 8 kHz frame using a 2.048 MHz PCM_CLK.

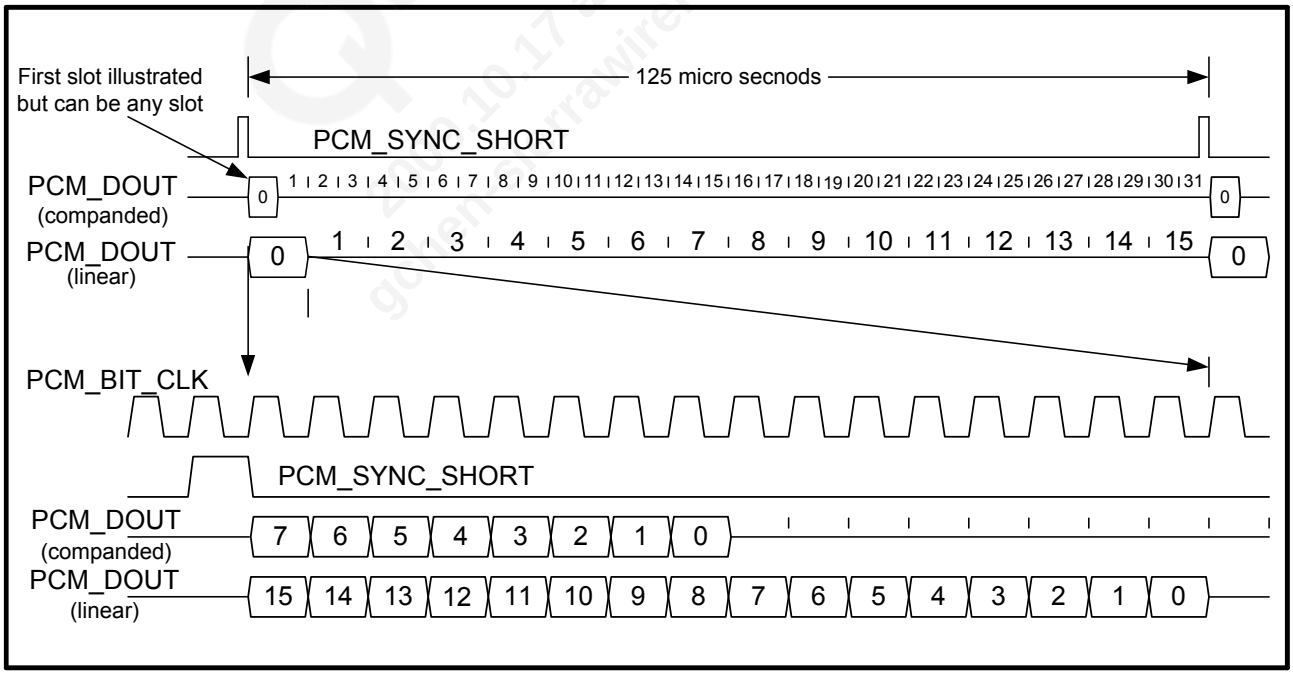


Figure 6-15 Slot and data processing timing example

6.6.4 Auxiliary PCM (long sync) mode

The auxiliary PCM enables communications with an external codec to support hands-free applications; linear, A-law, and μ-law codecs are supported. This interface operates with standard

long-sync timing, a 128 kHz clock, and an 8 kHz, 50% duty cycle PCM_SYNC_LONG. Like the primary PCM, auxiliary data is sampled on the falling edge of CLK and transmitted on the rising edge. However, in this mode, the PCM_SYNC *rising edge* represents the MSB. Figure 6-16 illustrates auxiliary PCM timing and data processing within an 8 kHz frame.

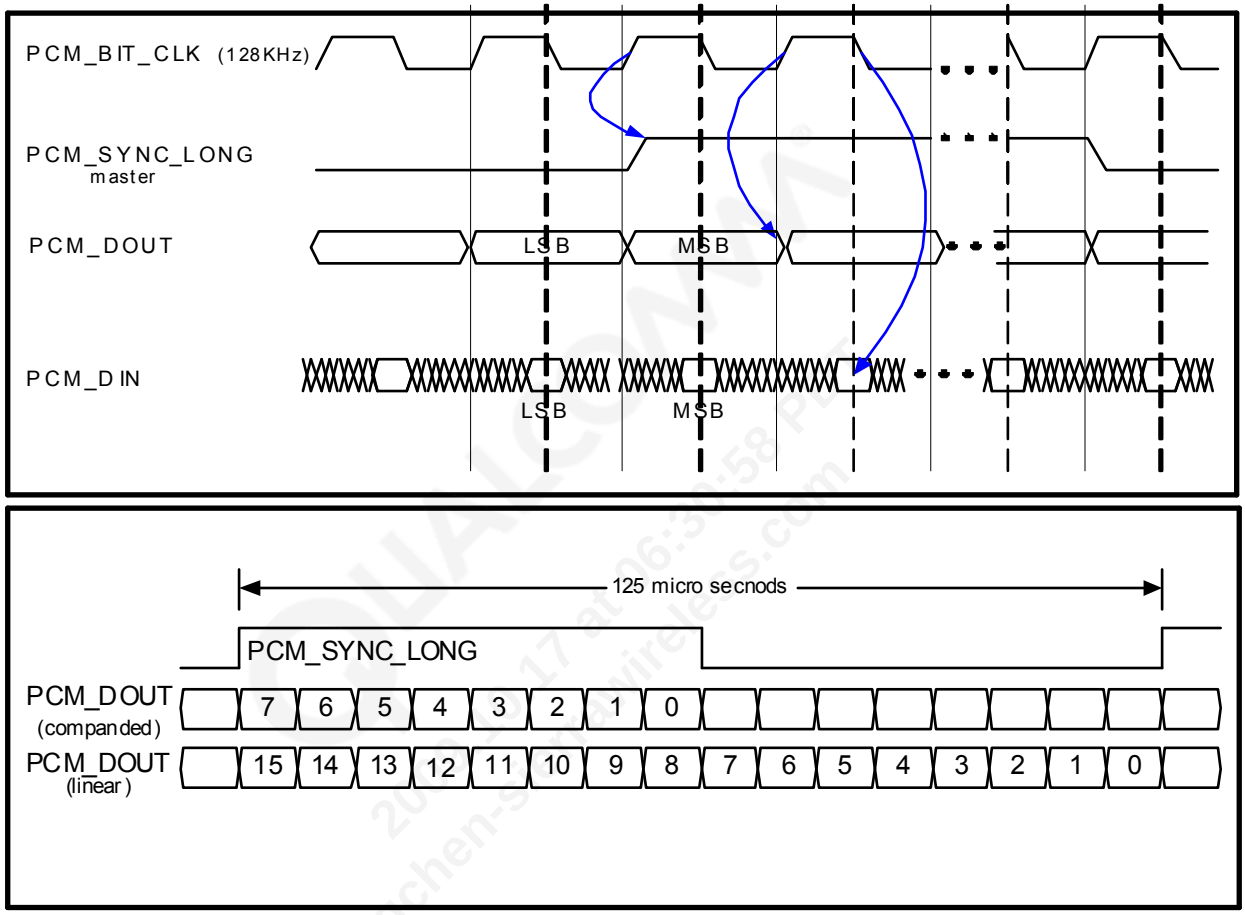


Figure 6-16 Auxiliary PCM timing example

6.6.5 Software control

The external PCM interface is programmed using the AUX_CODEC_CTL register. For detailed bit descriptions, refer to the *MDM6200 and MDM6600 Mobile Data Modem Software Interface* (80-VR001-2). To modify or configure the interface within AMSS, refer to *Application Note: External CODEC PCM Interface Software* (TBD).

6.7 Inter-integrated circuit (I^2C) interface

I^2C is a two-wire bus for inter-IC communication that supports any IC fabrication process (NMOS, CMOS, bipolar, etc.). Two wires (or lines), serial data (SDA) and serial clock (SCL) carry information between the connected devices. Each device is recognized by a unique address (whether it is a microcontroller, memory, LCD driver, stereo DAC, or keyboard interface), and can operate as either a transmitter or receiver, depending upon the device function.

A simplified description of I²C bus operation is given below:

- The master generates a *START* condition, signaling all ICs on the bus to listen for data.
- The master writes a 7-bit address, followed by a read/write bit to select the target device and define whether it is a transmitter or receiver.
- The target device sends an acknowledge bit over the bus. The master must read this bit to determine whether or not the addressed target device is on the bus.
- Depending upon the value of the read/write bit, any number of 8-bit messages can be transmitted or received by the master. These messages are specific to the I²C device used. After 8 message bits are written to the bus, the transmitter will receive an acknowledge bit. This message and acknowledge transfer continue until the entire message is transmitted.
- The message is terminated by the master with a *STOP* condition. This frees the bus for the next master to begin communications.

Both SDA and SCL are bidirectional lines, connected to a positive supply voltage via a current source or pull-up resistor. When the bus is free, both lines are high. The output stages of devices connected to the bus must have an open drain or open collector to perform the wired-AND function. The number of interfaces connected to the bus is solely dependent upon the bus capacitance limit of 400 pF.

The MDM implementation of its I²C controller is described in the rest of this section. For additional I²C interface information, see the *I²C Controller and I²C Bus Specification in MSM6xxx™ Chipsets* (80-V7836-1).

6.7.1 I²C connections

One or more of the GSBI blocks can be configured for I²C operation. For example, the reference design uses two external I²C interfaces:

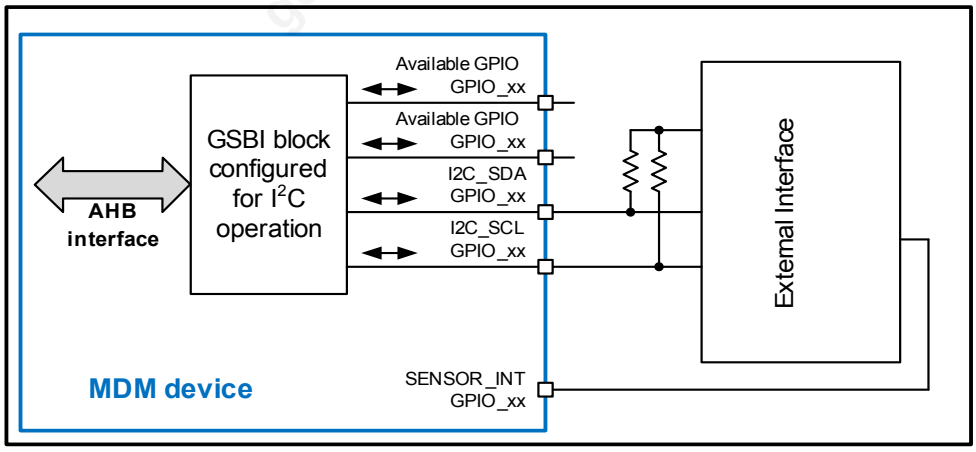


Figure 6-17 Example I²C connection

6.7.2 I²C setup

Since the GSBI blocks can be used for protocols other than I²C, some setup is required. Details will be provided in a future revision of this document.

6.7.3 I²C architecture

The I²C controller provides an interface between the advanced high-performance bus (AHB) and the industry standard I²C serial bus. This controller handles the I²C protocol and frees up the on-chip processor (and AHB) to handle other operations. It is I²C-compliant, high-speed mode (HS-mode) compliant, and a master-only device. This means that the MDM device can access all available I²C slaves on the bus, but cannot be accessed by any other masters. Example MDM-accessible slaves include a stereo codec (control registers only), external LCD controllers, and external touchscreen panels.

Previous-generation MSM and MDM devices supported I²C by multiplexing the I²C signals with USB signals, thereby allowing a single phone connector to be used for USB, I²C, and RS232 communications. The MDM device provides a separate I²C interface, so it supports I²C communications even though they are no longer included in the USB interface specification. A few additional comments about the MDM implementation:

- The I²C signals are generated by the configurable GSBI blocks and accessed via GPIOs.
- Primary I²C pins use GPIOs that can be configured as open-drain outputs; the pull-up resistor for the open-drain output is provided at the slave device.
- Camera auto-focus control through the I²C originates from the aDSP, requiring a separate hardware request port at the I²C controller.

The I²C controller architecture is shown in Figure 6-18. The clock-control and data-control functions are described in detail below.

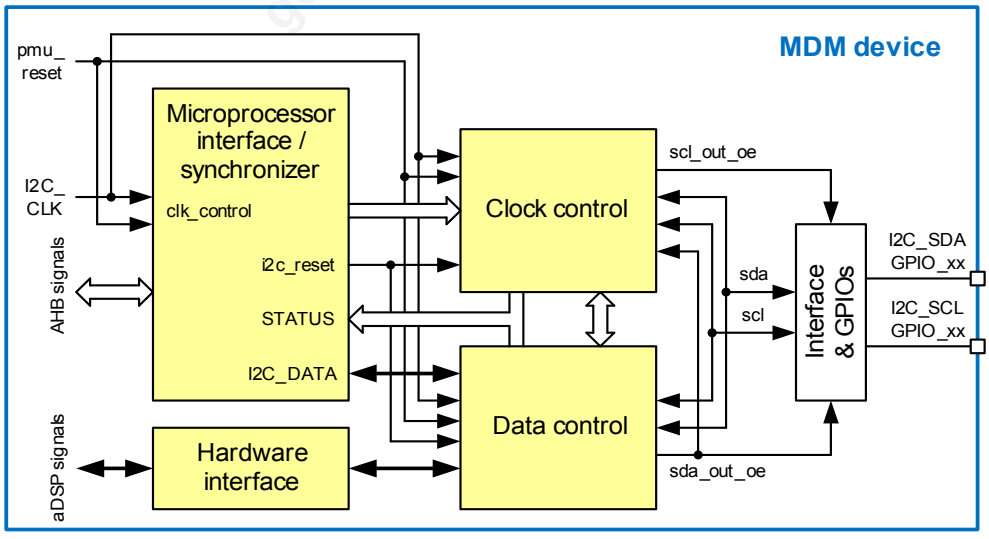


Figure 6-18 I²C controller architecture

6.7.4 I²C clock control

Figure 6-19 shows the various possible transitions in the different states of the clock-control state machine. Descriptions of the states and their transitions are provided below.

1) Reset bus idle state

- No activity on the I²C bus
- Controller leaves this state when START condition is detected; START initiates bus transactions and determines bus master.
  - If sda_out = 0, this controller is the master and transitions to the HIGH state.
  - If sda_out = 1, there is a different master, so this controller transitions to the NOT_MASTER state.
- Bus should be inactive during this state, but if activity is detected prior to START condition (scl_in = 0):
  - An error is generated.
  - The data-control block generates an interrupt.
  - The controller waits until bus activity has ceased.

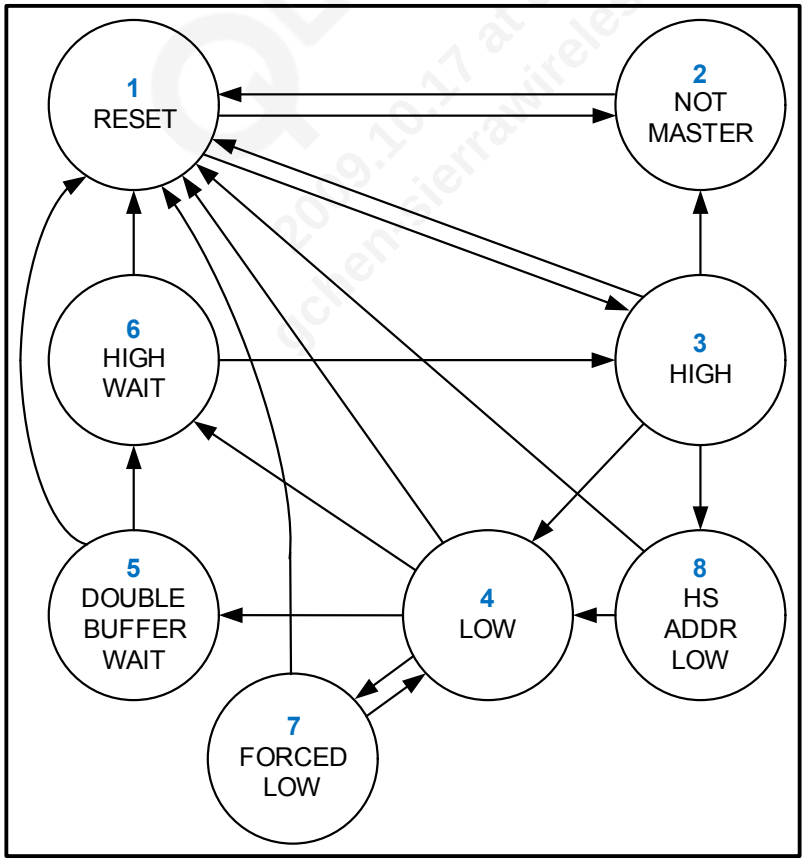


Figure 6-19 I²C clock-control states

2) Not-master state

- Another master is controlling the I^2C bus.
- There are two ways of reaching this state:
  - From the RESET_BUSIDLE state when another master has requested control of the bus.
  - From the HIGH state when this controller has lost arbitration for the bus.
- Once bus control is relinquished by the other master (STOP condition detected), this controller returns to the RESET_BUSIDLE state.

3) High state

- Outputs the high phase of scl_out. Four conditions are checked:

  1. STOP – If detected, this controller releases bus control and returns to RESET_BUSIDLE.
  2. Loss of arbitration – This controller sets arb_lost bit (which triggers an interrupt in the data-control block), releases bus control, transitions to the NOT_MASTER state, and waits for other controller to release the bus.
  3. Unexpected START condition – If detected, this controller releases bus, transitions to the NOT_MASTER state, and flags this as a bus error; there should not be any START conditions detected in this state.
  4. Toggling to low phase of scl_out – This can occur after this controller has counted off its high phase (count_zero = 1), or if another master has pulled the bus low; this occurs during arbitration.

- In HS mode, a 2:1 low-to-high phase ratio is required, so an extra low-phase state has been added (HS_ADDR_LOW state). In normal mode, the transition goes to the LOW state.

4) Low state

- Counts off the low phase of scl_out
- Once the low phase is counted off:
  - If the data-control block requests that the clock be stalled (force_low = 1), the controller goes into FORCED_LOW state and waits until the data-control block can continue its normal operation, at which time the controller returns to this state and counts off another low period.
  - If there is no need to stall the bus, the controller extends the clock low phase by two i2c_clk cycles by transitioning to DOUBLE_BUFFER_WAIT.
    - Double buffering of serial-in signals extends clock high phase by two i2c_clk cycles.
    - When in HS mode, double buffering is bypassed so the controller can transition directly to the HIGH_WAIT state.

5) Double-buffer wait state

- Extends clock low phase by two i2c_clk cycles, because double-buffering inputs causes state machine to extend high phase of the clock's two i2c_clk cycles
- Once the two cycles have been counted off, the controller transitions to the HIGH_WAIT state.

6) High wait state

- Monitors bus and identifies if another controller is holding clock line low:
  - This occurs during arbitration, where the other master's natural scl frequency is slower than this controller's frequency.
- Once the clock line is released, the controller transitions to the HIGH state to begin counting off its high phase.

7) Forced low state

- Holds scl_out low for as long as requested by the data-control block

8) HS address low state

- Counts off an extra low phase of scl_out so that a 2:1 low to high scl_out ratio is achieved, as per the I^2C specification

6.7.5 I^2C data control

Figure 6-20 shows the various possible transitions in the different states of the data-control state machine. Descriptions of the states and their transitions are given below.

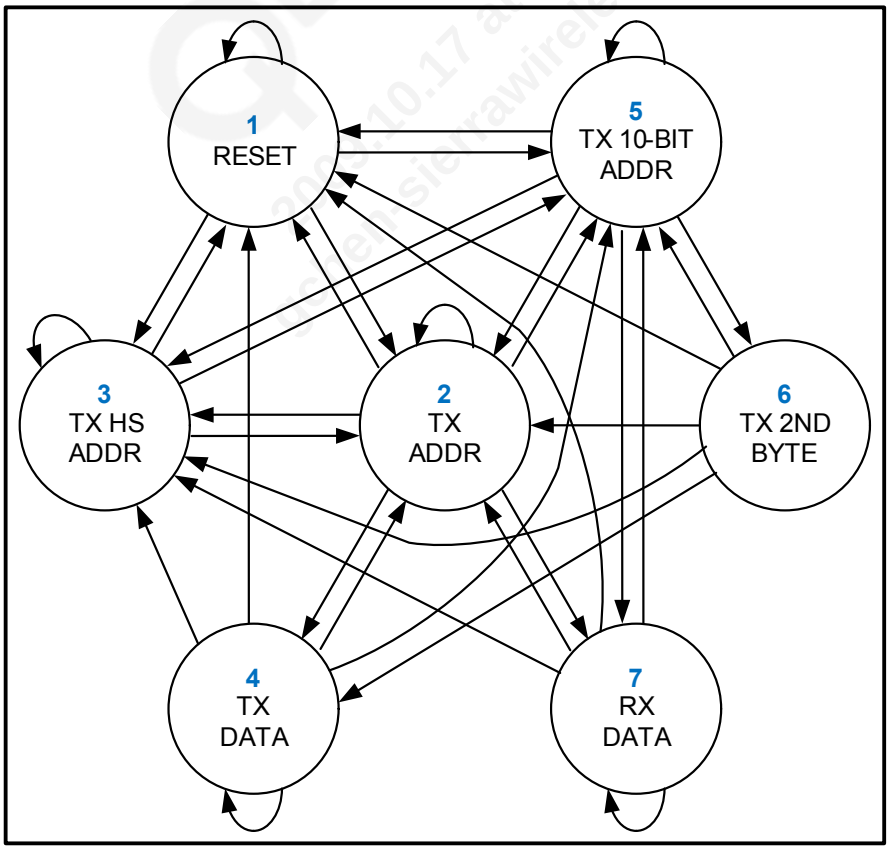


Figure 6-20 I^2C data-control states

1) Reset wait state

- Two main sections:

  1.Issues STOP command to bus to terminate transmissions:

  - When it was determined in the previous state that the STOP command must be issued, the stop signal is set high. Since the transition to the RESET_WAIT state is done during ACK cycle's high-clock phase, the controller must wait until a serial-clock low phase, and then issues the STOP command during the subsequent high phase.

  2.Identifies when the controller can attempt to take control of the bus:

  - Data must be present in the buffer, the bus cannot be presently active, and all interrupts must have been serviced (failed_tmp = 0).
    - If any are not met, the controller remains in the RESET_WAIT state until it can proceed with a transmission.
    - After all are met, the controller decodes data in the I2C_WR_DATA register.
  - Bytes not flagged with addr_byte bit set in this state are considered invalid writes and an interrupt is generated
  - If addr_byte bit is set, the address is decoded and the controller proceeds to the appropriate state.

2) Tx address state

- Transmits the 7-bit slave address; 3 main sections:

  1.Force low (force_low_tmp = 1) – occurs after 7-bit address is transmitted over the bus:

  - Controller enters this section in one of two ways:
    - If address was ACKed and there was no new byte buffered
    - If address was NACKed, but byte in buffer is flagged with the addr_byte set
  - Controller forces serial clock low (force_low is output to clock-control block) and waits for buffer to fill or interrupt to be serviced, whichever is the case.
  - Next state is dictated by write-buffer information:
    - If the addr_byte is not set, then it is data for the accessed slave.
    - If the addr_byte is set, then the address is decoded and the controller selects the appropriate next state.

  2.Issue start – generates the START condition that precedes any I^2C address on the bus

  - When transitioning to this state from any other, start is set and this is the first section of code that is executed.

  3.Transmit byte – shifts data onto the bus

  - The internal wait_sig is used to ensure that events are executed only once per i2c_clk.
    - Width of scl_in will be multiple of i2c_clk.

– Data is updated while serial clock is low (scl_in = 0); only action during serial-clock high is reading ACK (read_ack = 1); at this point, the next transition must be determined.

• If the packet is ACKed, the last_byte_bit has precedence overall because it means software has decided to terminate transmission and release bus control. The 7-bit address byte LSB (rd_wr_n) is checked next to allow the controller to release the serial data line, thereby allowing the slave to take control.

• If data is to be transmitted (last_byte_bit = 0 and rd_wr_n = 0), the controller checks if there is data available in the buffer.

□If not, the controller is set (force_low) and waits until the write buffer is filled (see the first section).

□If the buffer is full, then the next byte is decoded and the appropriate state is selected.

□If 7-bit address was NACKed, controller transitions to the RESET_WAIT state and issues a STOP, unless there is an address byte in the write buffer (wr_buffer_full = 1 and addr_byte = 1), at which time force_low is set and address byte is not transmitted until the interrupt is processed.

• Interrupt is generated on ACK reception regardless of its value – this means that a data transfer was completed on the bus.

3) Tx HS address state

■ Very similar to TX_ADDR state, so only the differences are outlined here.

■ Most significant – the byte following the high-speed master-code transmission must be an address and needs to be flagged with the addr_byte set.

□ In TX_ADDR state, if addr_byte = 0, it produced a transition to the TX_DATA state.

□ In this TX_HS_ADDR state, if addr_byte = 0 for the next byte due for transmission, it is flagged as an invalid write and an interrupt is generated to notify software of the error.

■ The high-speed master code must not be ACKed by any slave device.

□ If ACK occurs, regardless of the buffer's next byte, this causes an error and the controller transitions to the RESET_WAIT state and issues a STOP to release the bus.

□ In the transmit byte section, last_byte and rd_wr_n are not checked.

4) Tx data state

■ Very similar to the TX_ADDR state, so only the major differences are outlined here.

□ The TX_DATA state does not issue START commands on the bus, so start = 1 section is removed.

□ In this TX_DATA state, the rd_wr_n bit state does not need to be examined – the controller already knows it is writing to the slave.

5) Tx 10-bit address state

- Slight variation from TX_ADDR state: If the byte in the write buffer is not flagged with the addr_byte set, transition to the TX_2ND_ADDR_BYTE state.

6) Tx second-address byte state

- Identical to the TX_DATA state; added to make it easier to identify when the second byte of the 10-bit address is being sent, thereby making it easier to debug

7) Rx data state

- I^2C is receiving data from the slave.
- Reception is terminated if either the last_byte is set, or the addr_byte is set (which means a new slave is selected for a new transaction, keeping this I^2C controller as the bus master).

6.8 Inter-IC sound (I^2S) interface

The MDM6x00 device provides an I^2S interface that can be used to transfer serial digital audio to/from an external stereo DAC/ADC. The I^2S interface is a 3-wire interface: serial clock (I2S_SCLK), word select (I2S_WS), and serial data (I2S_SD).

6.8.1 I^2S supported modes and connections

The I^2S interface supports all the Tx/Rx, slave/master modes (Figure 6-21): transmitter-master, transmitter-slave, receiver-master, and receiver-slave. Master or slave is determined by the device driving the I2S_SCLK and I2S_WS; the transmitter sends data to the receiver. This directivity is highlighted within the figure.

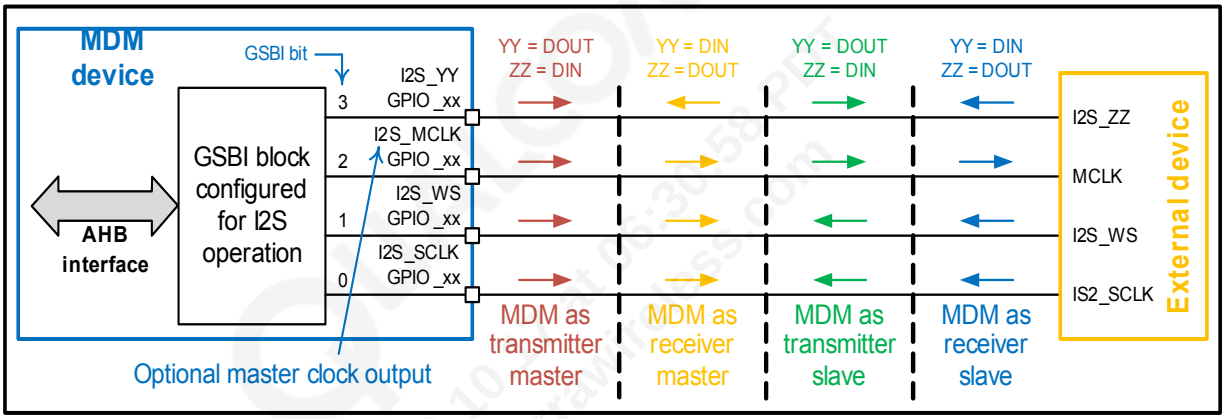


Figure 6-21 Example I^2S connections

Concurrent transmit and receive modes as either a master or slave are supported by configuring two of the five GSBI ports for I^2S operation (Figure 6-22).

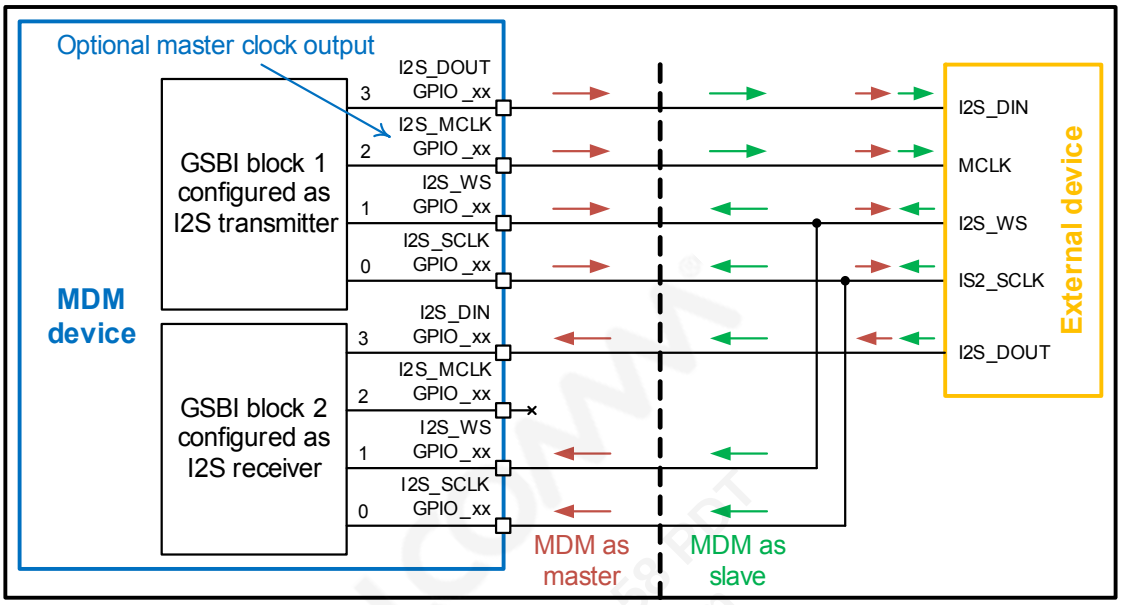


Figure 6-22 I²S concurrent Tx/Rx operation

When MDM-internal FM radio or audio codec functions are used, a dedicated I²S port is configured operation and connected to the WCA circuits internally.

6.8.2 I²S setup

Since the GSBI blocks can be used for protocols other than I²S, some setup is required. Details will be provided in a future revision of this document.

6.8.3 I²S interface details

A high-level timing diagram of the I²S signals is given in Figure 6-23.

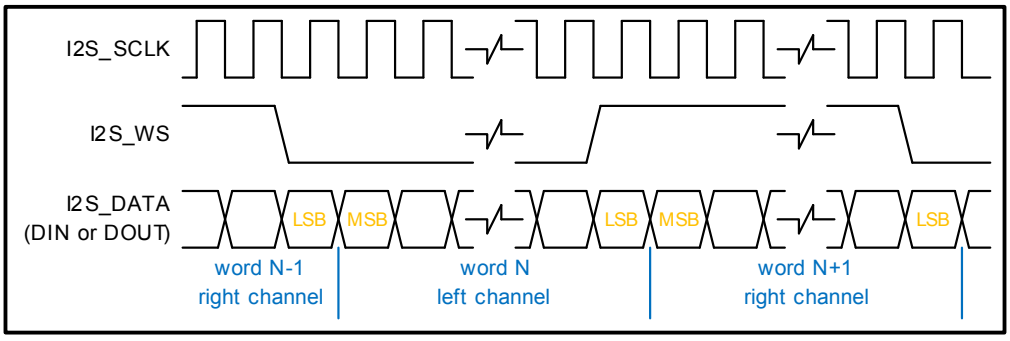


Figure 6-23 High-level I²S timing diagram

I2S_DOUT or I2S_DIN

The serial PCM stereo-data stream for both channels are output from or received by the MDM device through this pin. Serial data is transmitted in two's complement, with the MSB first. The transmitter and receiver are not required to have the same word length:

- When the transmitted word length is greater than the receiver word length, the bits after the receiver's LSB are ignored; the rest of the transmitter's LSBs are ignored.
- When the transmitted word length is less than the receiver word length, the receiver's missing LSB will be set to zero initially, so they will remain at zero.
- The MSB has a fixed position, whereas the LSB position depends upon word length.
- The transmitter always sends the MSB of the next word one clock period after WS changes.
- Serial data sent by the transmitter may be synchronized with either the trailing (H-to-L) or leading (L-to-H) edge of the clock signal.
- Serial data must be latched into the receiver on the leading edge of the serial clock signal.

I2S_WS

The word-select line indicates the channel being transmitted/received:

- 0 specifies the left channel
- 1 specifies the right channel
- The WS signal changes one clock period before the MSB is transmitted.

I2S_SCLK

This is the serial bit clock whose rate is a function of the data width and sample rate:

I2S_SCLK rate = (2 x bit_width) x F_S

where bit_width = 16 bits per channel and F_S is the sample rate, therefore:

I2S_SCLK rate = 32 x F_S

Sample rates of 8, 16, 24, 32, 44.1, and 48 kHz are supported. An example clock rate is:

I2S_SCLK rate = (2 x 16) x 48 kHz = 1.536 MHz

where bit_width = 16 and F_S = 48 kHz.

6.8.4 I²S register settings

The I²S interface is configured using the MI2S_MODE register (Table 6-6).

Table 6-6 I²S register details

Interface	I²S register	Control bit	Functional description
GSBI as I²S	MI2S_MODE	12	Sets the MDM I²S port as master or slave: 0 = slave 1 = master (default)
		3	Sets the MDM I²S port as transmitter or receiver: 0 = receiver 1 = transmitter

For detailed bit descriptions of the I²S interface, refer to the *MDM6200 and MDM6600 Mobile Data Modem Software Interface* (80-VR001-2).

6.9 Serial peripheral interface (SPI)

The SPI bus is a 4-bit synchronous serial data link. Devices communicate in master/slave mode where the master device initiates the data transfers. Multiple slave devices are supported by using chip selects. Since there are no explicit communication framing, error checking, or defined data word lengths, the transfers are strictly at the “raw” bit level.

6.9.1 SPI connections

The reference design did not include any SPI connections. A generic SPI interface is shown in Figure 6-24.

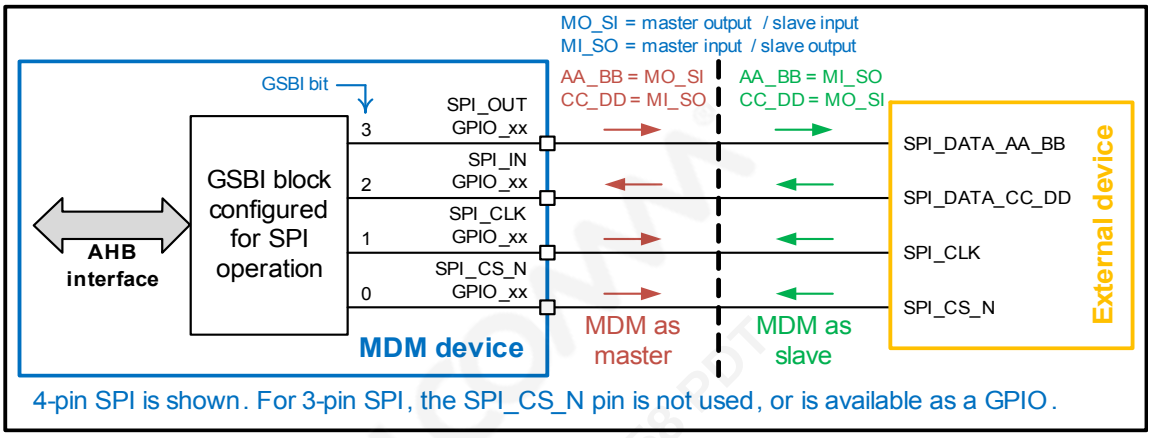


Figure 6-24 Example SPI connections

6.9.2 SPI setup

Since the GSBI blocks can be used for protocols other than SPI, some setup is required. Details will be provided in a future revision of this document.

6.9.3 SPI architecture

Major SPI blocks (Figure 6-25) are:

- Register bank and status and control – provides the internal bus interface, software register interface, and overall core control.
- Output FIFO – holds all data to be output and provides the output data mover interface.
- Shift register – provides the serial-to-parallel and parallel-to-serial conversions necessary for external transfers, and provides loop-back. An spi_toggle_output signal indicates to the output FIFO that an output value has been loaded for shifting. Likewise, an spi_toggle_input signal indicates to the input FIFO that an input value is available for loading.
- Input FIFO – holds all data to be input and provides the input data mover interface.
- Clock control – provides the master clock and a reset signal to the shift register.
- Chip select – drives the chip select signals when the MDM is the SPI master, and receives the CS signal when a slave. Regardless of the CS source, this block provides an spi_transfer signal to tell the shift register that a shift operation should take place.

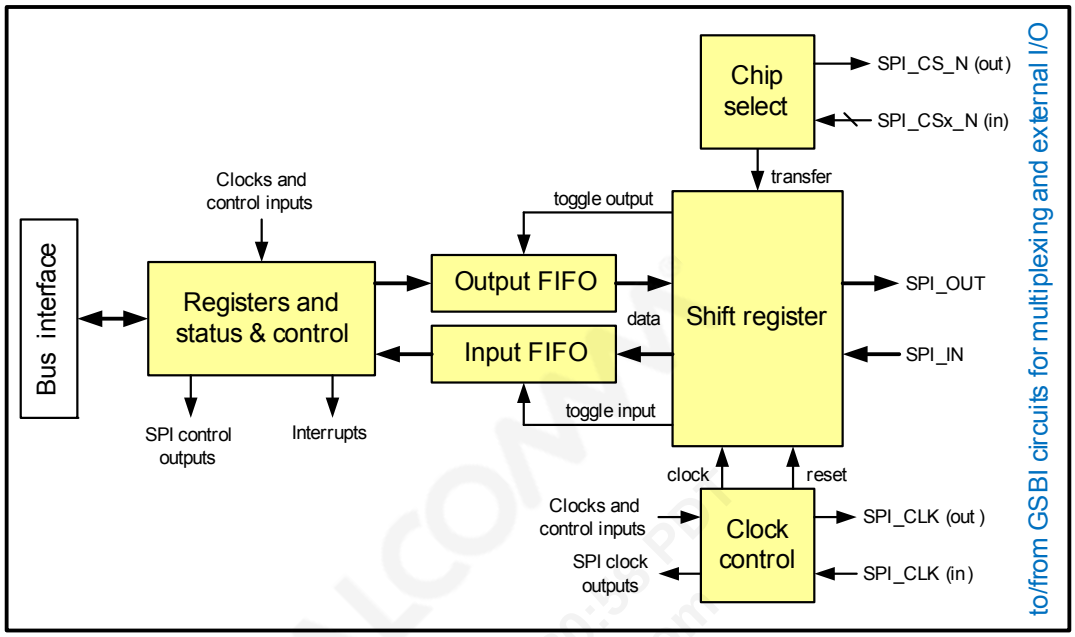


Figure 6-25 SPI functional block diagram

6.9.4 SPI protocol requirements

1. As an SPI master, the core supports several SPI system configurations [1], including 1 through 5 and multi-master.
2. As an SPI master, the core supports SPI_CS0_N, SPI_CS1_N, SPI_CS2_N, and SPI_CS3_N.
3. As an SPI master, the core supports SPI_CLK.
4. As an SPI master, when no transfers are taking place (IDLE), the core supports SPI_CLK_IDLE_LOW and SPI_CLK_IDLE_HIGH.
5. As an SPI master, the core supports leaving the SPI_CLK running when no SPI_CS#_N is asserted (during IDLE).
6. As an SPI master, the core supports SPI_DATA_MO_SI tri-state during IDLE (optional).
7. As an SPI master, the core supports Input_First_Mode.
8. As an SPI master, the core supports Output_First_Mode.
9. As an SPI master, the core supports any value of *N* between 4 and 32.
10. As an SPI master or slave, the core supports the following half-duplex modes:
    a. SPI_DATA_MO_SI only with SPI_DATA_MI_SO held low.
    b. SPI_DATA_MI_SO only with SPI_DATA_MO_SI held low.
    c. Half duplex on a bidirectional signal (defined for system configuration 3 only [1]).
11. As an SPI master, the core supports a mechanism to control the number of SPI_CLK ticks between the assertion of different SPI_CS signals as shown in Figure 6-26. Even though there

1. SPI system configurations as defined in TBD.

is no formal *flow control* mechanism, a slave may require *dead time* between SPI_CS assertions – this capability meets that potential requirement.

12. As an SPI master, the core supports the Qualcomm SPI_CS_N master requirements as discussed in Section 6.9.6.
13. As an SPI master, the core supports assertion of SPI_CS#_N between each transfer of size *N* (CS is normally de-asserted). As an option, SPI_CS#_N can be asserted for a "first transfer" and left asserted for *T* transfers through the "last transfer" *T*. Under this option, requirement #11 above still applies but the SPI_CLK is turned off every N bits while SPI_CS#_N is left asserted. This corresponds to the multi-transfer chip select (MX_CS).
14. As an SPI master, the core supports configuring SPI_CS#_N as active high (optional).

6.9.5 SPI software support

- Except as noted below, output data (SPI_DATA_MO_SI for master or SPI_DATA_MI_SO for slave) is written by software into the core with the *N* bits left (MSB) justified. The core requires no additional shifts to align the data for output.
- The above requirement does not apply when ***N*** equals 8, 16, or 32. For these values of *N*, output shift data written by software into the core shall follow the "natural" right (LSB) justified alignment. Furthermore, the left justification amount is limited as follows:
  - N 01-to-07, software shift justified to 08 bit boundary
  - N 09-to-15, software shift justified to 16 bit boundary
  - N 17-to-31, software shift justified to 32 bit boundary
- The core also provides an option for bit-shifting in hardware to align the MSB left justified.
- Master input shift data (SPI_DATA_MI_SO) is presented from the core for software reads with the ***N*** bits right (LSB) justified. No shift operations are required by software to align the data.
- Using an output FIFO and an output shift register, a write FIFO allows the next output shift value(s) to be staged while the current value is being shifted out.
- Using an input FIFO and input shift register, a read FIFO holds the previous input shift value(s) while the current value is being shifted in.
- An interrupt mechanism indicates when an output FIFO is in need of service.
- An interrupt mechanism indicates when an input FIFO is in need of service.
- An interrupt mechanism indicates the following errors:
  - Output FIFO under-run – indicates the FIFO not serviced before the next output shift was required. When this happens, the previous value is used.
  - Input FIFO over-run – same as output under-run, but for an input instead. When this happens, the previous value is maintained and the newest value discarded.
  - SPI_CLK over-run – the number of SPI_CLK ticks that occurred while SPI_CS#_N was asserted was greater than the programmed value of ***N***. When this error occurs only the first ***N*** bits are passed to the input FIFO and the output shift value is held at zero.
  - SPI_CLK under-run – the number of SPI_CLK ticks that occurred while SPI_CS#_N was asserted was less than the programmed value of ***N***. When this error occurs the bits that were received are passed to the input FIFO and the *current* output bit is forced to zero.

- From a hardware perspective, SPI_CLK is not assumed 100% reliable for completing all SPI transactions, so a time-out mechanism is provided. Low reliability may be due to extreme operating conditions or during debug.
- 32-bit data transfers are supported by two options:
  - When ***N*** equals 8, four values are packed into each 32 bit data transfer
  - When ***N*** equals 16, two values are packed into each 32 bit data transfer.

6.9.6 Chip select example – MDM as master

A timing example is shown in Figure 6-26 that illustrates SPI_CS_N operation when the MDM is the master. This example also includes SPI_DATA_MO_SI operation per protocol requirement #6 (listed in Section 6.9.4).

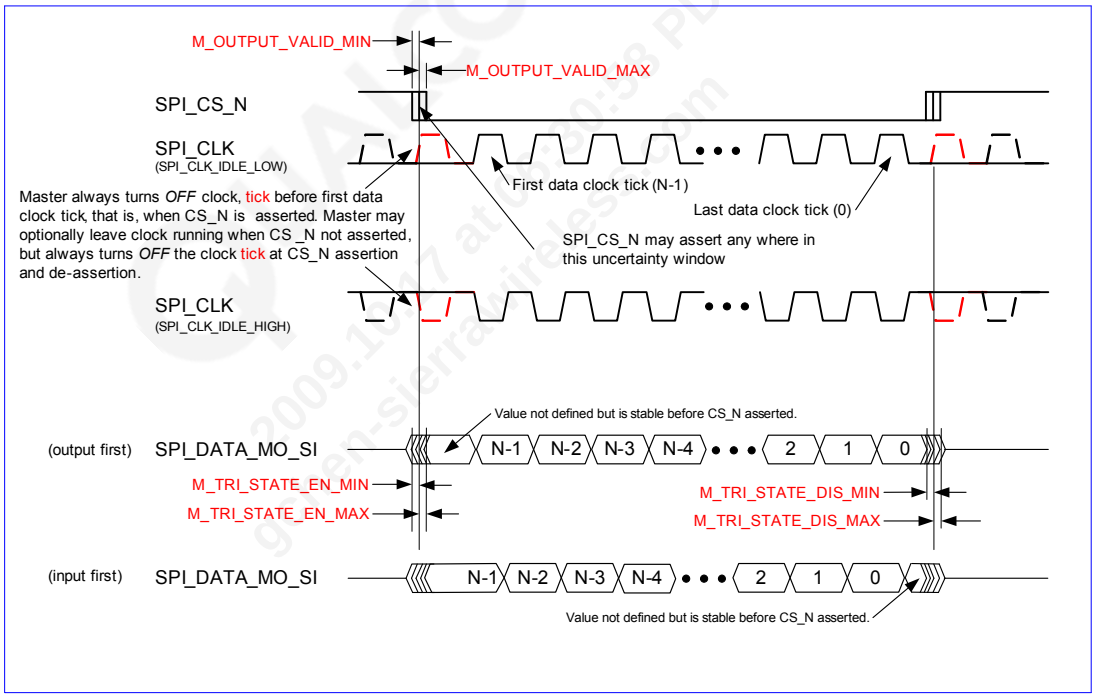


Figure 6-26 SPI_CS_N operation with MDM as master

As seen above, the MDM master always asserts and de-asserts SPI_CS_N on the leading edge of SPI_CLK. Per protocol requirement #7, SPI_CLK can be allowed to run when no SPI_CS_N is asserted. Even under this option, the MDM master always turns off SPI_CLK when SPI_CS_N changes state.

6.10 Near field communicator (NFC)

NFC is a short-range wireless-communication technology that enables the exchange of data between devices without requiring contact (a contactless exchange). A MDM-based NFC device is compatible with existing contactless infrastructure already in use for public transportation and payment. Future applications might include electronic ticketing, travel cards, identity documents, mobile commerce, and electronic keys.

NFC communicates via magnetic-field induction, where two loop antennas are located within each other's near field, effectively forming an air-core transformer. It operates within a globally available and unlicensed radio frequency band.

The MDM6x00 device supports NFC through four GPIO pins that interface with a NFC IC. Two of the four MDM pins are one of the I^2C interfaces available via the GSBI blocks (Section 6.7). An example NFC interface is shown in Figure 6-27.

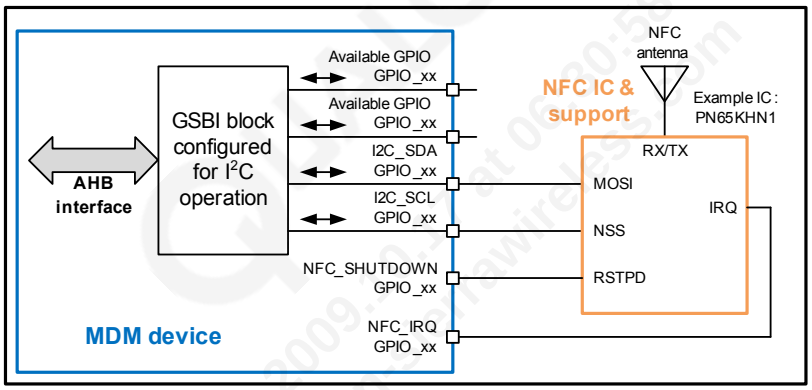


Figure 6-27 Example NFC interface

7 BB/RF/ANA/PM Interfaces

Most MDM interfaces are addressed in the connectivity, RF, and WCA chapters. This chapter covers status and control interfaces not addressed elsewhere: the RF front-end, some PMIC connections, and the 19.2 MHz XO signal distribution.

7.1 RF front-end

The MDM6x00 device supports various airlinks and operating bands, depending upon the MDM variant and the handset's RF design (as discussed in Chapter 9). The MDM to RF front-end interface details depend upon the application, so three examples are listed below:

- CDMA + GSM + GNSS using the MDM6600 device
- WCDMA + GSM + GNSS using the MDM6200 device
- CDMA + WCDMA + GSM + GNSS using the MDM6600 device

7.1.1 Example MDM6600 RF front-end (CDMA)

Figure 7-2 illustrates and describes MDM connections with the RF front-end in a MDM6600-based design.

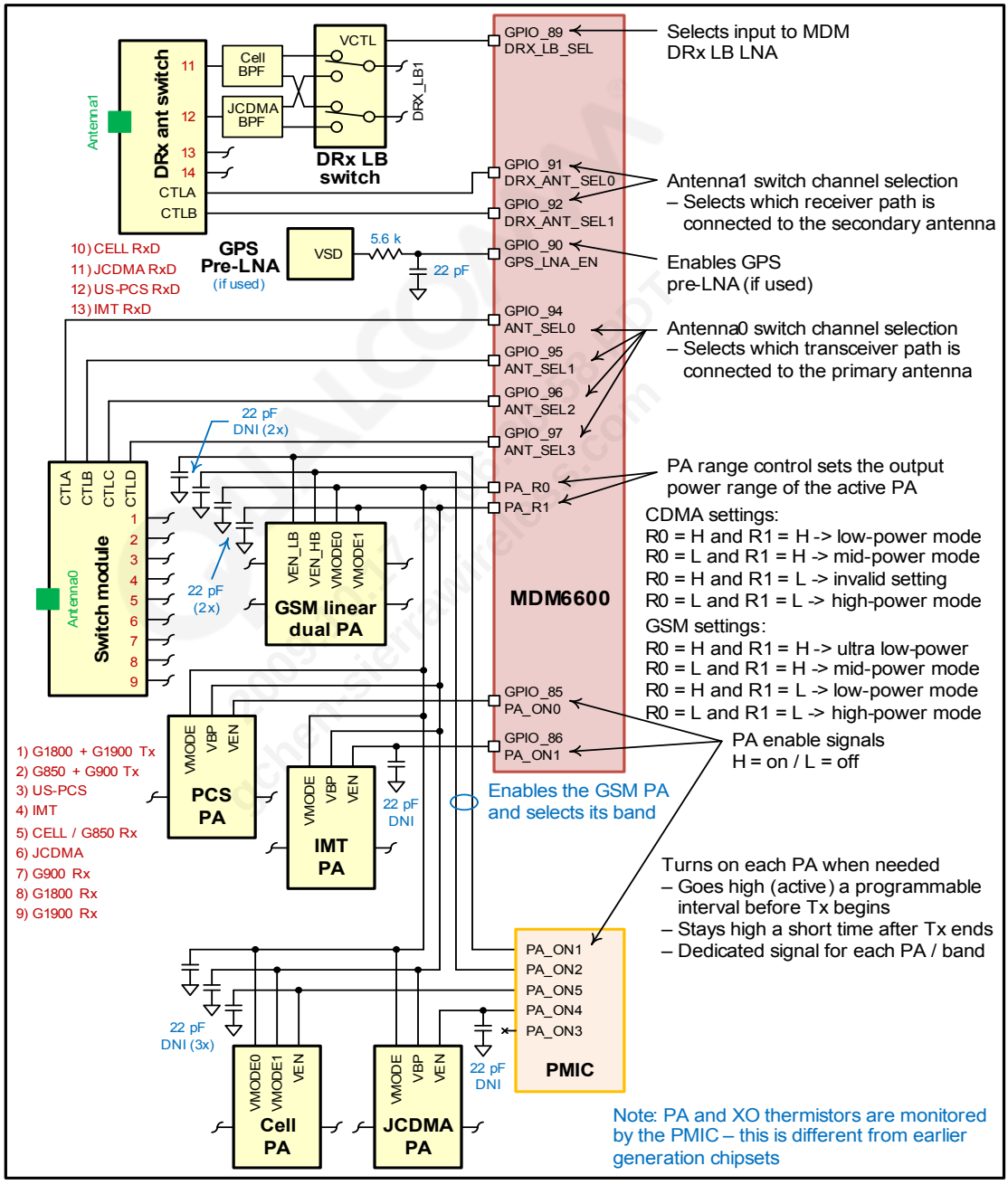


Figure 7-1 Example MDM6600 RF front-end control signal connections (CDMA)

7.1.2 Example MDM6200 RF front-end (WCDMA)

Figure 7-2 illustrates and describes MDM connections with the RF front-end in a MDM6200-based design.

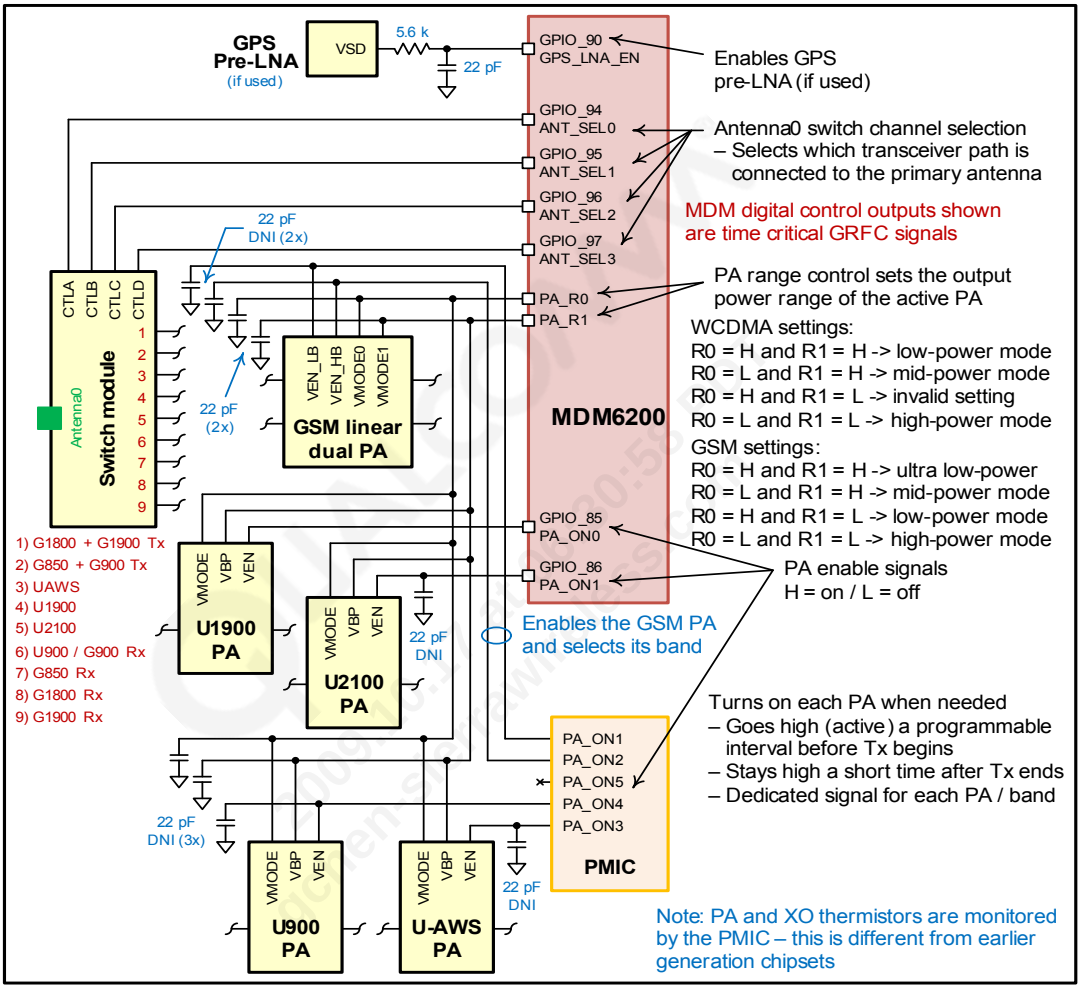


Figure 7-2 Example MDM6200 RF front-end control signal connections (WCDMA)

7.1.3 Example MDM6600 RF front-end (CDMA + WCDMA)

Figure 7-3 illustrates and describes MDM connections with the RF front-end in a MDM6600-based design.

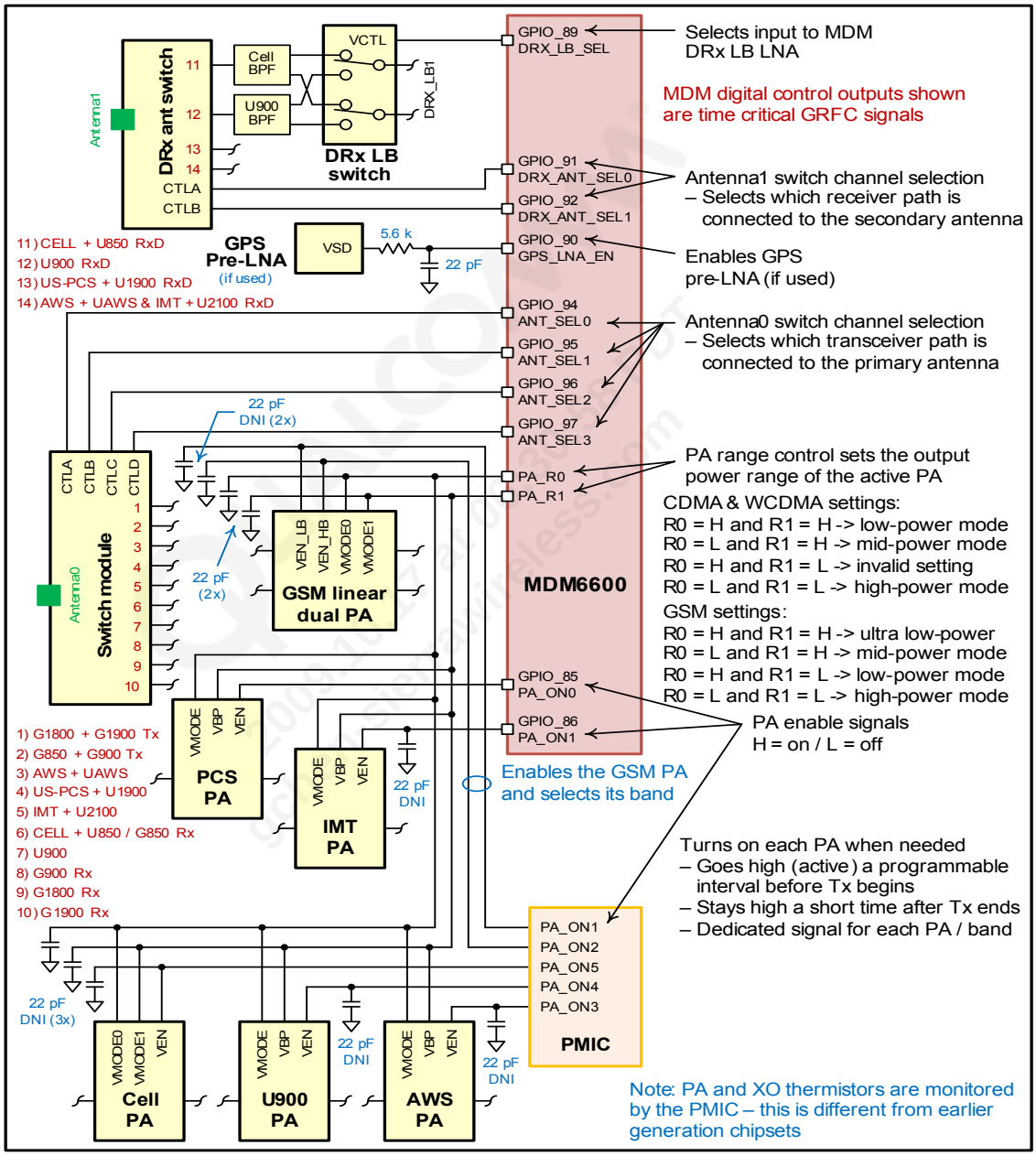


Figure 7-3 Example MDM6600 RF front-end control signal connections (CDMA + WCDMA)

7.2 Power management

The MDM6x00 device relies upon the PM8028 IC for its supply voltages and for many support functions. See the PMIC documentation for full power management details (*PM8028 Power Management Ic Device Specification (Advance Information)* (80-VN204-1); user guide, *PM8028 Power Management IC User Guide* (80-VN204-3); *PM8028 Power Management IC Design Guidelines (*80-VN204-5).

PM-related topics addressed elsewhere within this document include:

- Power supplies
- Clock connections – Section 3.7.8
- Modem power management – Section 3.8.2
- Support for connectivity functions like UIM and NFC
- XO signal distribution – Section 7.3

Other MDM/PM interfaces that are not addressed elsewhere are highlighted in Figure 7-4.

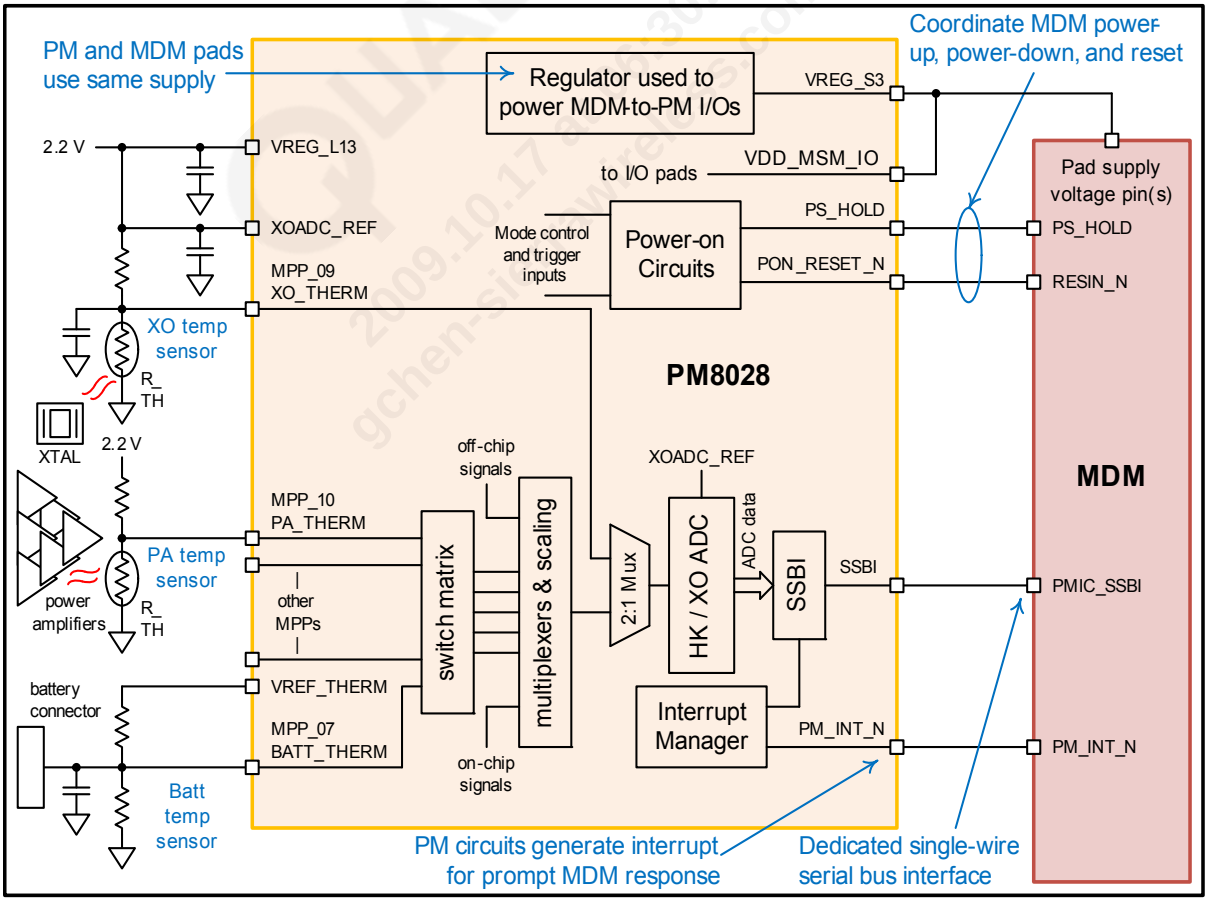


Figure 7-4 Other MDM/PM connections

In earlier generation MSM and MDM devices, the housekeeping ADC was integrated within the packages. This circuit is now included in the PMIC, and is multiplexed to allow XO temperature monitoring as well as the usual HK functions. As usual, the PMIC includes switch and multiplexer circuits for selecting the ADC input. But now, a more direct path is provided from MPP_09 (XO_THERM) to the ADC. The PMIC also includes more channel-dependent scaling options. See the PMIC documents for details (80-VN204-x).

The ADC output is reported to the MDM device via the SSBI.

The PMIC SSBI is also shown in the figure. Handset designers define their applications' PM8028 operating modes and circuit parameters using the application programming interface (API). The API commands are implemented via the MDM/PM single-wire serial bus interface (SSBI). The API is documented in AMSS software; see the applicable AMSS 6600™ documentation for details.

The SSBI provides efficient initialization, control of device operating modes and parameters, and verification of programmed parameters. The MDM is the master while the PMIC is the slave. PM8028 parameters of particular interest that are controlled via SSBI but not discussed in other sections include:

- Switch matrix and analog multiplexer settings (channel selection)
- HK/XO ADC controls
- Reading HK/XO ADC data
- Reading, masking, and clearing interrupts

7.3 XO signal distribution

XO signal distribution is shown as in Figure 7-5.

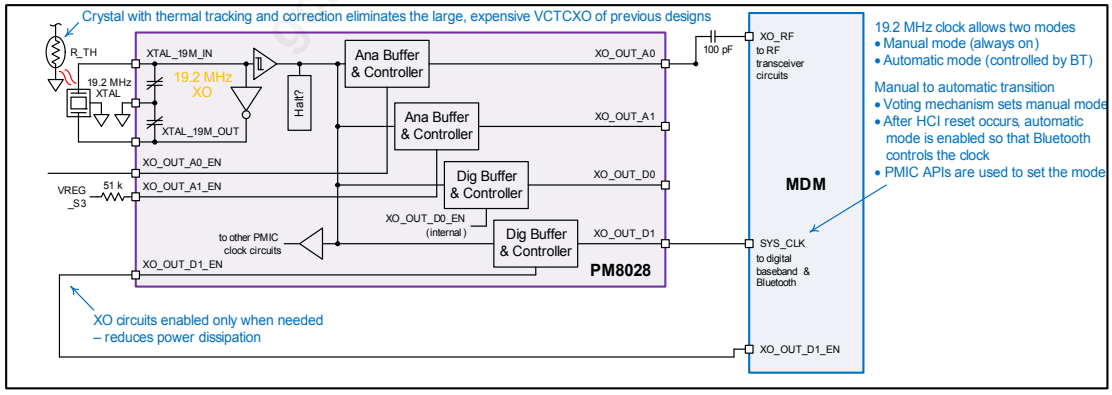


Figure 7-5 XO signal distribution

8 General-Purpose Input/Output Pins

The MDM6x00 IC includes 100 general-purpose input/output (GPIO) pins, and each can be configured as a digital input or digital output. Inputs can be set to have a pull-up, pull-down, keeper, or no-pull. Output drive strength is also programmable. Software assigns functions to the GPIOs, and their configurations are set accordingly. Refer to the *MDM6200 and MDM6600 Mobile Data Modem Device Specification (Advance Infomation)* (80-VR001-1) for functional assignments and more configuration details.

This chapter describes the following GPIO topics:

- Pad structure
  - A functional diagram and description of the GPIO pad structure
  - The state of each GPIO pad at power-up and cold resets; these states are maintained until AMSS software programs them differently through control registers.
  - Wake-up GPIOs for modem power management
  - Programmable pad configurations, including input, output, and pull options
  - GPIO multiplexing
- Top-level mode multiplexer (TLMM) – a convenient way of programming functional groups of GPIO pads
- General serial bus interface (GSBI) ports

8.1 Pad structure

Figure 8-1 provides a conceptual illustration of the GPIO pad structure. The figure includes three segments:

- The top segment shows the output driver circuits; see Section 8.1.3.1 for details.
- The middle segment shows the pull circuits; see Section 8.1.3.3 for details.
- The bottom segment shows the input circuits; see Section 8.1.3.2 for details.

All registers used for GPIO pad configuration and control are asynchronously reset to ensure immediate pad control upon power-up without clock dependency.

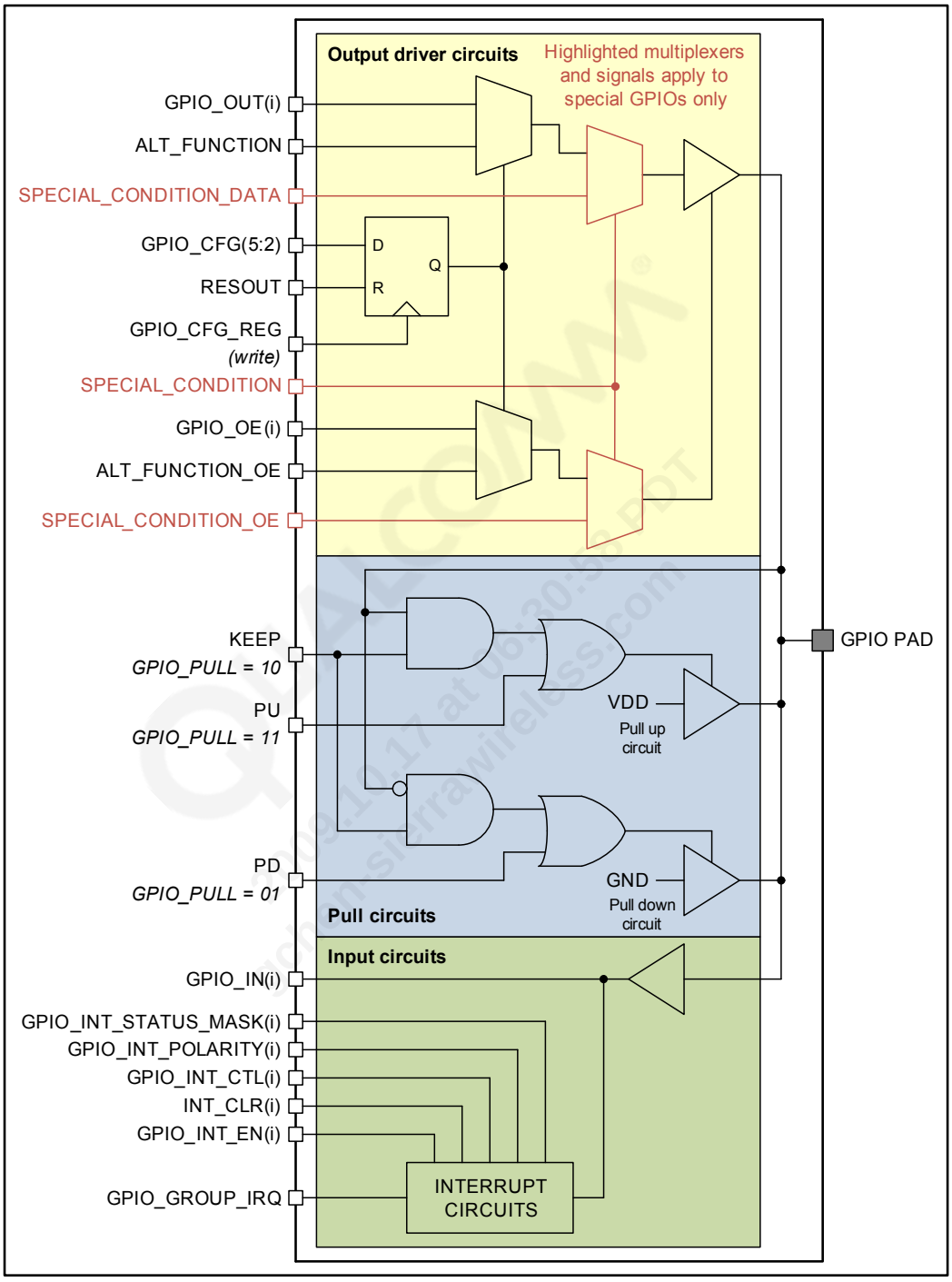


Figure 8-1 Conceptual GPIO pad structure

8.1.1 Power-up states

To avoid GPIO problems, a specific supply sequence is required when powering up the MDM device (Figure 8-2). The core supply (VDD_CORE) must turn on and reach 90% of its programmed value before any of the pad supplies (VDD_Px) are enabled. If this sequence and the timing relationships are not achieved, the GPIO pads might come up in undefined states. The PM8028 IC ensures the proper supply sequence and timing.

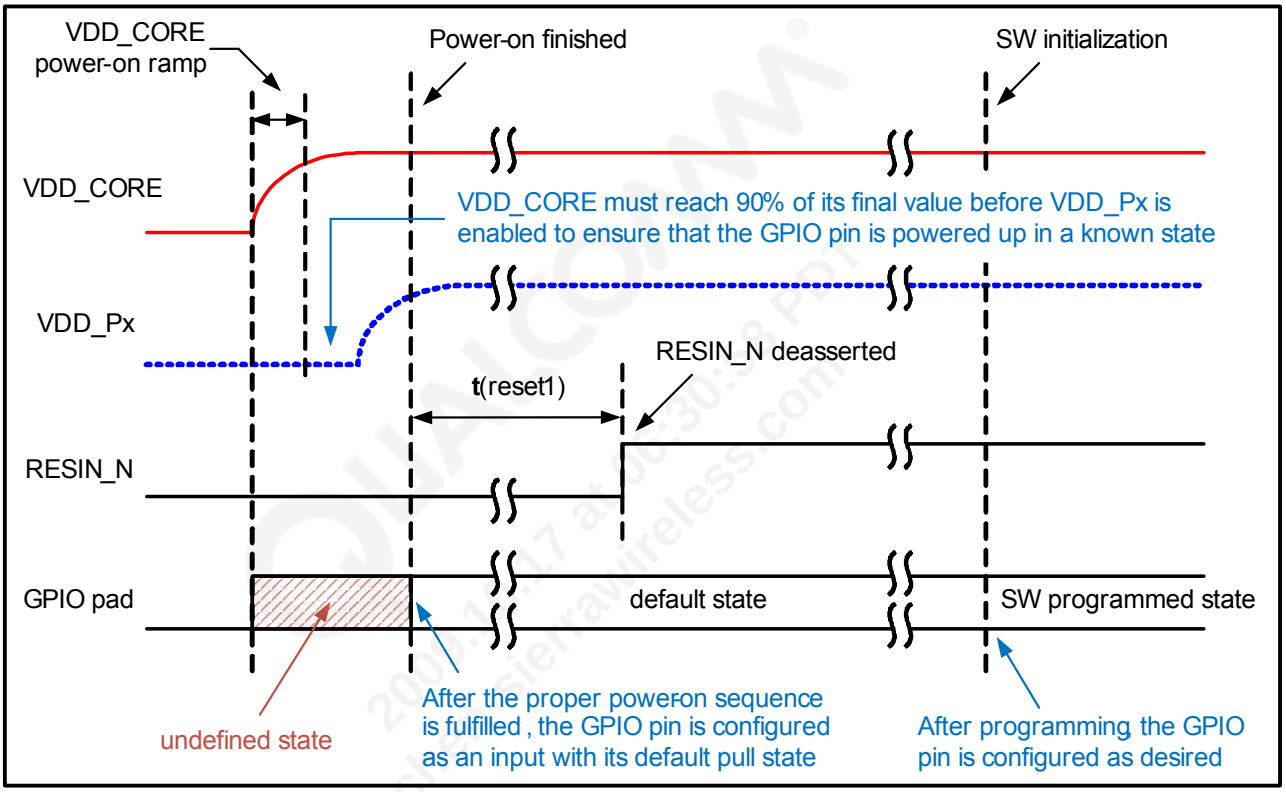


Figure 8-2 GPIO initialization

As the supplies turn on, the GPIO pads go through an interval of undefined states. All GPIO pads are configured as inputs with their default pull states (defined within the *MDM6200 and MDM6600 Mobile Data Modem Device Specification (Advanced Iformation)* (80-VR001-1) when the RESIN_N signal is driven high (deasserted) by power management circuits. After further delay, software initialization occurs for the programmed GPIO values. The GPIO pads are now properly configured, as defined by AMSS software.

8.1.2 Using modem power management (MPM)

If the modem power management feature is enabled by software to reduce power consumption, only the following select GPIO pins can be used to wake up the MDM device:

- GPIO_11, 13, 14, 18, 19, 20, 21, 22, 34, 48, 51, 52, 55, 59, 64, 71, 76, 78, 98, 99
  - GPIO_18, 19, 20, 21, and 22 are ANDed and provide a single MPM interrupt source.

If MPM is enabled, none of the other GPIO pins can be used as a wake-up interrupt.

NOTE GPIOs 11, 13, and 14 do not default to the MPM feature. For these GPIOs to serve as wake up interrupts when the MPM is enabled, the relevant fuse needs to be blown in the CONIF chain.

If MPM is used, external pulls should not be used on digital pads because:

- During MPM operation, all digital pads are held by a keeper.
- If the external pull is in the opposite direction, DC current will flow (highly undesirable during a low-power sleep mode).

The PCB layout must avoid excessive crosstalk on the wake-up GPIOs; coupling might cause an unintentional trigger.

8.1.3 Programmable pad configurations

8.1.3.1 Outputs

Three types of output signals are supported (as seen in the top segment of Figure 8-1):

- A normal GPIO output signal – uses GPIO_OUT(i) and GPIO_OE(i). To drive the output pad as a general-purpose output signal, configure the GPIO for non-alternate function, write the GPIO_OUT_X register with the desired value, and then set the corresponding bit in the GPIO_OE_X register to enable the output path.
- An alternate GPIO output signal – uses ALT_FUNCTION and ALT_FUNCTION_OE. A procedure similar to the one just described is used, but the alternate functions are exercised rather than the normal functions.
- Special GPIO output signals like ETM can override the other GPIO functions (implemented by software).

In addition to these types, each GPIO's output drive strength is programmable (Table 8-1). Pad drive strength is controlled by the 2:0 bit fields within the GPIO_PAD_HDRIVE_MSEL_n registers.

Table 8-1 GPIO drive strength – GPIO_PAD_HDRIVE_MSEL_n register

Register bits	Setting	Drive strength [1]
2:0	000	2 mA
	001	4 mA
	010 (reset state)	6 mA
	011	8 mA
	100	10 mA
	101	12 mA
	110	14 mA
	111	16 mA

1. The stated 2 to 16 mA settings apply when the associated pad-supply voltage is 1.8 V. Higher supply voltage results in slightly higher drive current. The impact of pad voltage to drive strength is explained further in the *MDM6200 and MDM6600 Mobile Data Modem Device Specification (Advance Information)* (80-VR001-1).

8.1.3.2 Input configurations

Two types of input configurations are supported (as seen in the bottom segment of Figure 8-1):

- Buffer – a standard CMOS input buffer; its output is GPIO_IN(i).
- Interrupt – An interrupt circuit allows the input signal's level or edge, with selectable polarity, to generate an interrupt.

A more detailed diagram and explanation is given in Figure 8-3.

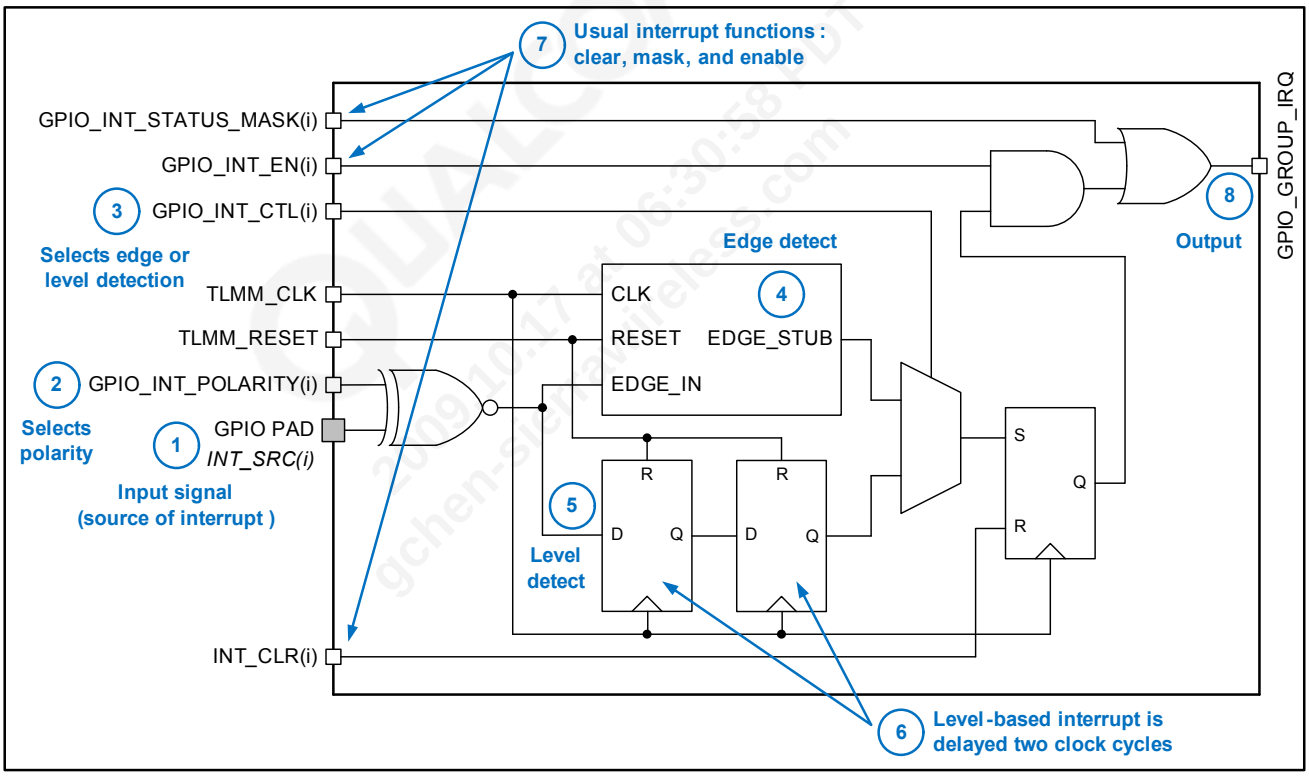


Figure 8-3 GPIO interrupt circuit

8.1.3.3 Pull configurations

Three types of pulls are supported (as seen in the middle segment of Figure 8-1):

- Pull-up – to the pad voltage defined within the *MDM6200 and MDM6600 Mobile Data Modem Device Specification (Advanced Information)* (80-VR001-1).
- Pull-down – to ground.
- Keeper – maintains the pad's last valid logic level, regardless of whether it was an input or an output. Keepers are weak drivers that cannot drive an external bus.

The internal pulls are implemented using JFETs; their strengths will vary between devices, but are not expected to be weaker than about 100k.

8.1.4 GPIO multiplexing

A paging scheme is used to address individual GPIOs and to configure their alternate functions, pulls, and drive strengths. Before a particular GPIO can be configured using the GPIOx_CFG register, its index must be written to the GPIOx_PAGE register. Non-selected interfaces are gated in the GPIOx_CFG register, thereby forcing their inputs low.

There are four GPIO register types:

- Output registers (4) – GPIO_OUT_3 to GPIO_OUT_0; each bit of these ***read/write*** registers corresponds to a specific GPIO.
- Output enable registers (4) – GPIO_OE_3 to GPIO_OE_0; each bit of these ***write*** registers corresponds to a specific GPIO.
- Input registers (4) – GPIO_IN_3 to GPIO_IN_0; each bit of these ***read*** registers corresponds to a specific GPIO's input value.
- Selection registers – GPIOx_PAGE and GPIOx_CFG:
  - GPIOx_PAGE is a 7-bit register whose content determines which GPIO is being programmed
  - GPIOx_CFG is a 6-bit register that is used to configure the settings of the GPIO selected by the GPIOx_PAGE register. It contains the pin's function (bits 5:2) and pull (bits 1:0)

Refer to the *MDM6200 and MDM6600 Mobile Data Modem Software Interface* (80-VR001-2) for more register information.

8.2 Top-level mode multiplexer (TLMM)

The top-level mode multiplexer (Figure 8-4) provides a convenient mechanism for sharing multiple internal functions on the same sets of GPIO pads. The mode assignment for each set of GPIOs is specified using a combination of input pin settings (such as MODE[2:0]) and software-programmable register settings. Using the TLMM method for programming GPIO pads allows higher-level instructions, resulting in faster and easier GPIO assignments. Without the TLMM, each GPIO pad would require individual programming.

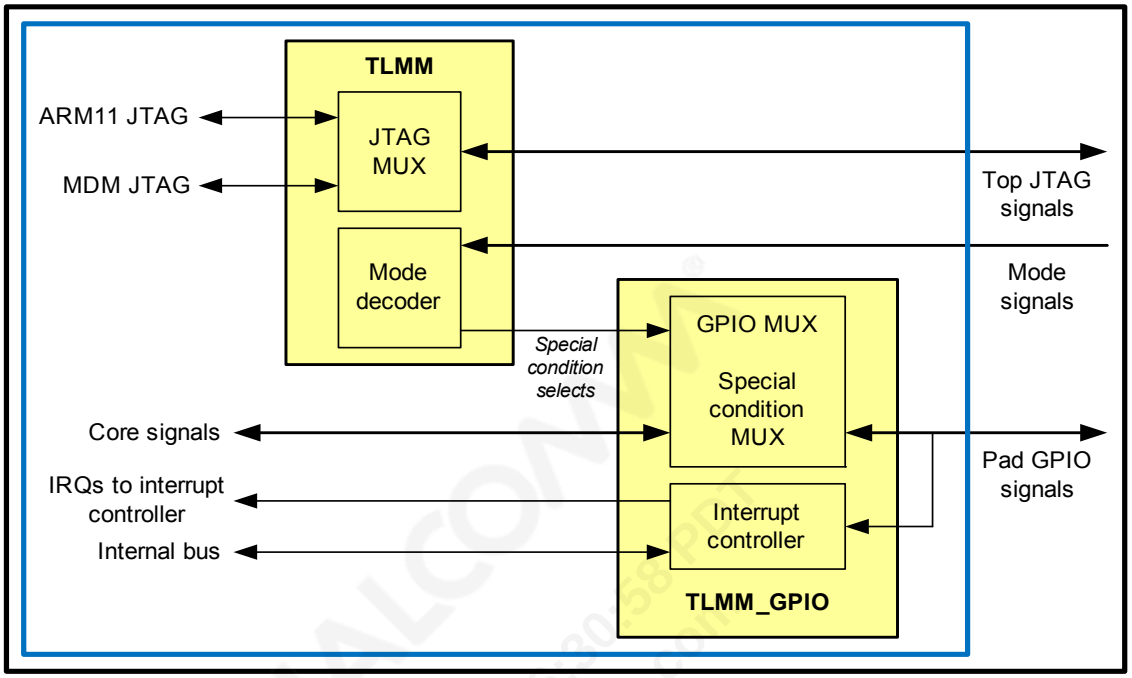


Figure 8-4 TLMM architecture

The TLMM provides software-controlled multiplexing of the MDM device and provides two multiplexing (and demultiplexing) modes:

- Standard – most GPIOs fall into this category. They are configured as inputs at power-on, and then are set by software to their desired functionality. Some example uses include GPIOs supporting different feature sets, such as a phone manufacturer choosing to use the UIM in lieu of UART.
- EBIs – assume their default EBI functions at power-on.

The TLMM module receives mode-select control from the mode pins and from software-writable registers (used to control the GPIO configuration – drive strength, pull direction, and keeper). The GPIO pin values are readable directly as a register-mapped read.

All registers controlling pad configuration and control are asynchronously reset to ensure immediate pad control at power-on without clock dependency.

8.3 General serial bus interface (GSBI) ports

Twenty GPIOs are available in five sets of four pins, with each set connected to a GSBI block. The GSBI configuration options are listed in Section 6.3; each option is described in the subsequent sections listed in Table 6-2.

9 RF Transceivers

The MDM6x00 device includes most of the active RF and LO functions for low-cost handsets operating in the CDMA, UMTS, GSM, and GNSS bands (depending upon the MDM device variant). This chapter describes these on-chip functions in detail and defines their interface requirements. It is organized according to function: shared Rx/Tx functions, RF transmit signal paths, Tx power detector, Tx LO circuits, RF receive signal paths, and Rx LO circuits.

9.1 Supported RF configurations

Table 1-2 lists all the bands supported by the MDM6x00 device, but only specific band combinations are supported by each *software build*.

The software build is defined as follows:

- Each device is supported by its unique software
  - An example: MDM6600 device and its AMSS 6600™ software
- Software reads the MDM hardware ID register to identify the IC version (MDM6200 or MDM6600)
- Using the QCN system, NV items are programmed to define the air interface support
  - The application is selected via QCN.
    - Global, Japan 1, UMTS C, etc.
  - Specific RF Rx input and Tx output ports are enabled via QCN.
    - These are ports allowed to be turned on and off during the phone's operation.
      - PRX_LB1; DRX_LB1; TX_LB1; etc.
  - Based upon the MDM device variant and the selected RF ports, the RF bands are assigned by software and are identified by the QCN system.

The supported RF configurations for each software build are listed in Table 9-1.

Table 9-1 Software supported RF configurations

RF port	6600, 6200 Global SW1 (3)	6600, 6200 Japan 1 SW2 (3)	6600, 6200 Japan 2 SW3 (3)	6200 only Global UMTS C SW4	6600 only CDMA 450 SW5
TX_LB1	BC0 + B5/B6	BC0 + B5/B6	BC0 + B5/B6	B8	BC0
TX_LB2	NC	NC	NC	NC	NC
TX_LB3	B8	BC3	B8	NC	BC5
TX_LB4	G850 + G900	G850 + G900	G850 + G900	G850 + G900	G850 + G900
TX_MB1	G1800 + G1900	G1800 + G1900	G1800 + G1900	G1800 + G1900	G1800 + G1900
TX_MB2	BC15 + B4	B11	B11	B4	NC
TX_MB3	BC1 + B2	BC1 + B2	B9	B2	BC4
TX_MB4	BC6 + B1	BC6 + B1	BC6 + B1	B1	NC
PRX_LB1	BC0 + B5/B6 + G850	BC0 + B5/B6 + G850	BC0 + B5/B6 + G850	B8 + G900	BC0+G850
PRX_LB2	B8	BC3	B8	NC	BC5
PRX_MB1	BC6 + B1	B11	B11	B1	NC
PRX_MB2	BC1 + B2	BC1 + B2	B9	B2	BC4
PRX_HB	BC15 + B4	BC6 + B1	BC6 + B1	B4	NC
DRX_LB1	BC0 + B5/B6 + B8 (sw)	BC0 + B5/B6 + BC3 (sw)	BC0 + B5/B6 + B8 (sw)	NC	BC0
DRX_LB2	G900	G900	G900	G850	G900
DRX_MB1	G1800 + G1900	G1800 + G1900	G1800 + G1900	G1800 + G1900	G1800 + G1900
DRX_MB2	BC1 + B2	BC1 + B2 + B11 (sw)	B9 + B11	NC	BC4
DRX_HB	BC6 + B1 + BC15 + B4	BC6 + B1	BC6 + B1	NC	NC

Notes:
1. Band classes for CDMA (BCx) and WCDMA (Bx) are identified in Table 1-2.
2. NC = no connection (RF port not used); sw = bands are switched at RF port.
3. These software builds support all QSC device variants. The MDM6600 supports all bands listed; MDM6200 supports all but CDMA bands.
4. Red text indicates that Rx diversity functions are supported (MRD and/or SHDR).

9.2 RF transceiver (common to Rx and Tx)

Two pins provide functions that are common to both the receiver and transmitter (Figure 9-1).

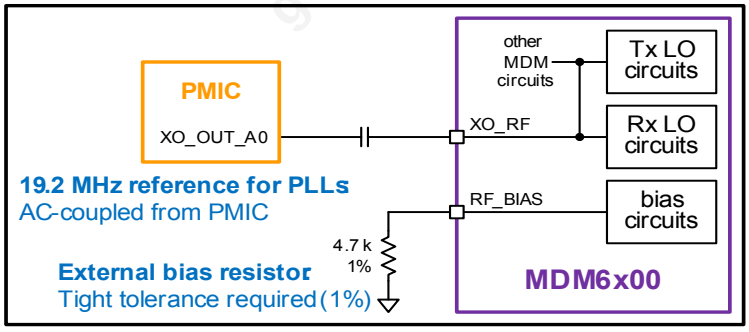


Figure 9-1 Common Rx and Tx circuits

9.3 RF transmit signal paths

The MDM6x00 device provides eight RF transmit output ports that are split into two band categories:

- Low band – CDMA, UMTS, and GSM low bands

- High band – CDMA, UMTS, and GSM high bands

The Tx signal paths are described in the following sections, including band-specific and mode-specific output chains between the MDM output ports and the antenna.

9.3.1 CDMA transmitters

The Tx paths share a common interface from the Tx DAC, and then they share the RF transmitter's analog baseband circuits. Each band category, low and high, has a dedicated baseband-to-RF upconverter and its own set of RF gain stages. A single RF Tx power-detector circuit is integrated on-chip and supports all CDMA bands (only one output is active at a time).

An example CDMA Tx design is shown in Figure 9-2 based upon the MDM6x00 device.

The RF transmit paths begin with a single, shared analog I and Q differential baseband signal from the device's baseband circuits. Each component of the baseband signal is lowpass filtered and amplified to levels sufficient for driving the quadrature upconverters. Only one upconverter is active at a time. The transmitter LO signals are generated by circuits described in Section 9.5, and delivered to the upconverter circuits at the correct frequency, with the proper phase relationship, and with adequate drive level.

The active upconverter's output is at the desired RF channel frequency, and drives its dedicated output circuits. These RF circuits include multiple variable gain stages that provide transmit AGC control. The wide range of driver-amplifier output levels is achieved while supporting the CDMA standard's requirements for ACPR, spurious emissions, Rx-band noise, etc.

All transmitter output ports are single-ended with 50-Ohm nominal impedance. Each requires a simple matching network to interface with its RF front-end circuits' Tx path. See Section 9.3.2 for matching-network topologies and guidelines on handling unused RF output ports.

The driver amplifiers end the signal-path circuitry within the MDM device, but the external transmit paths continue: MDM output — matching network — Tx bandpass filter (450, 700, and AWS bands only) — power amplifier — coupler — duplexer — antenna switch module — antenna.

NOTE Ongoing evaluations will determine if Tx SAW filters are needed in any CDMA paths besides the 450, 700, and AWS bands. The goal is to avoid filters in the PCS, IMT, and Cellular bands.

As noted in Figure 9-2, transmit output chain components are available in varying degrees of integration.

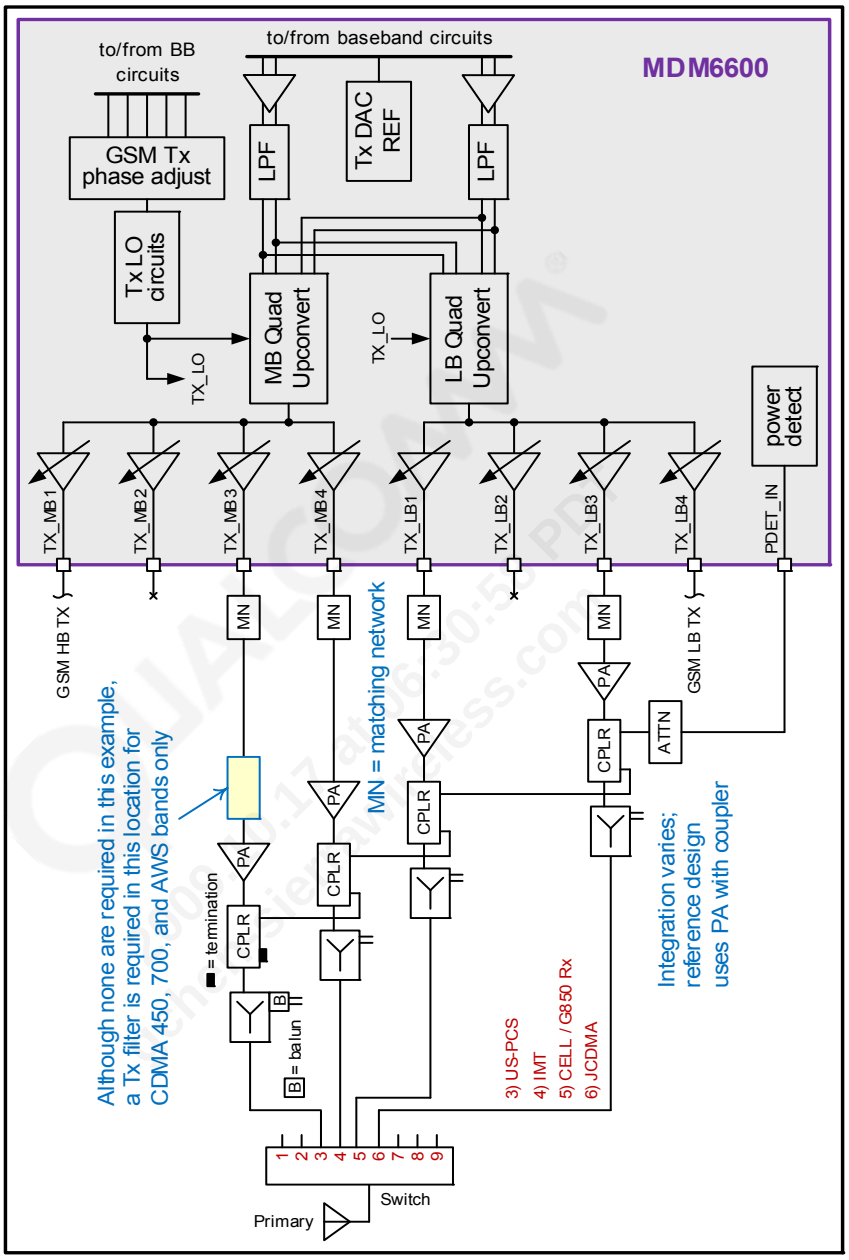


Figure 9-2 CDMA transmitter and example application (MDM6600 device)

Each band's PA drives a coupler that provides a low-level sample of Tx power (only one Tx path is active at a time). An on-chip power detector circuit provides a Tx power estimate that assists in setting the transmit gains and assuring the maximum allowed output power is not exceeded. The couplers use a daisy-chain configuration so that they are able to share the single, on-chip power detector circuit.

9.3.2 CDMA transmitter connections

Recommended CDMA transmitter output matching-network topologies are shown in Figure 9-3, along with guidelines on handling any unused Tx ports.

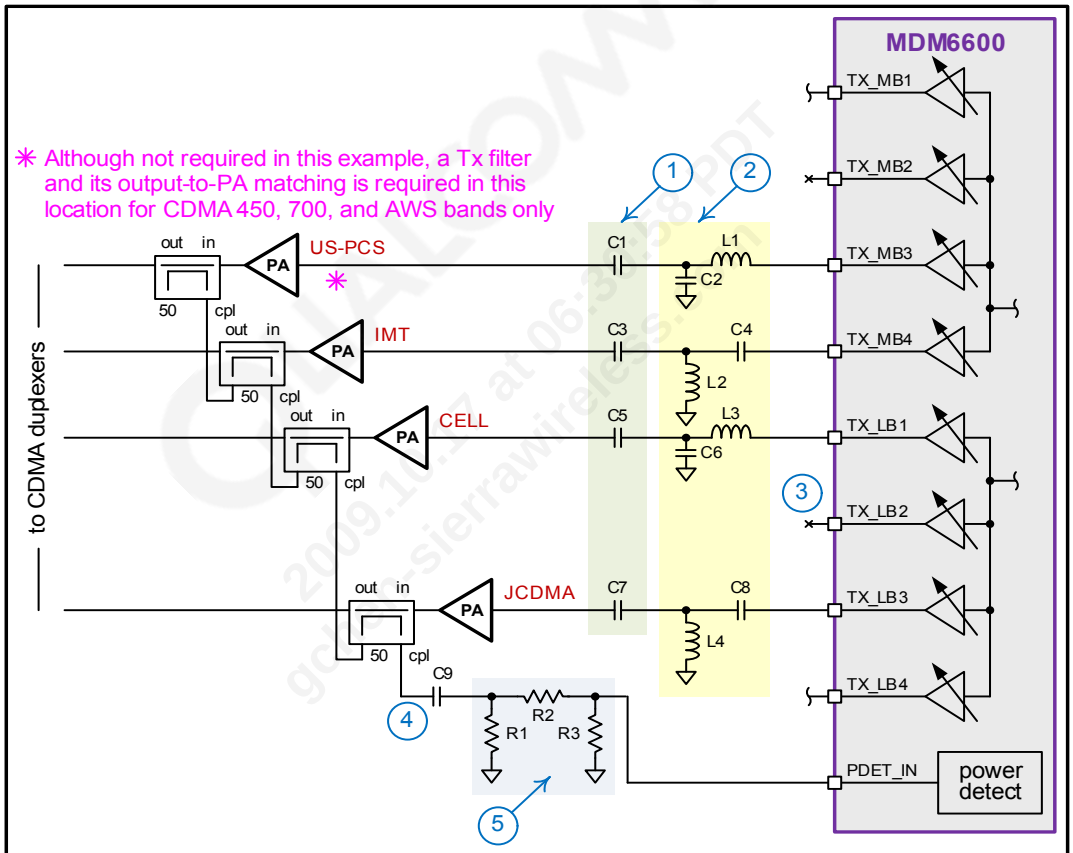


Figure 9-3 Expected CDMA RF transmitter matching-network topologies

The CDMA Tx output matching networks depend upon the operating frequency range, external front-end devices (regardless of their degree of integration), PCB material and stack-up, layout details, and matching components used (which vary from supplier to supplier).

The MDM power detector input (pin U25, PDET_IN) presents a 50 Ω nominal load to the final coupler in the daisy chain. Matching should not be required, though a resistive attenuator is needed to set the proper drive level into the MDM device.

9.3.3 UMTS transmitters

The description presented in Section 9.3.1 for the CDMA transmitters applies to the UMTS transmitters as well. An example application based upon the MDM6200 device is shown in Figure 9-4.

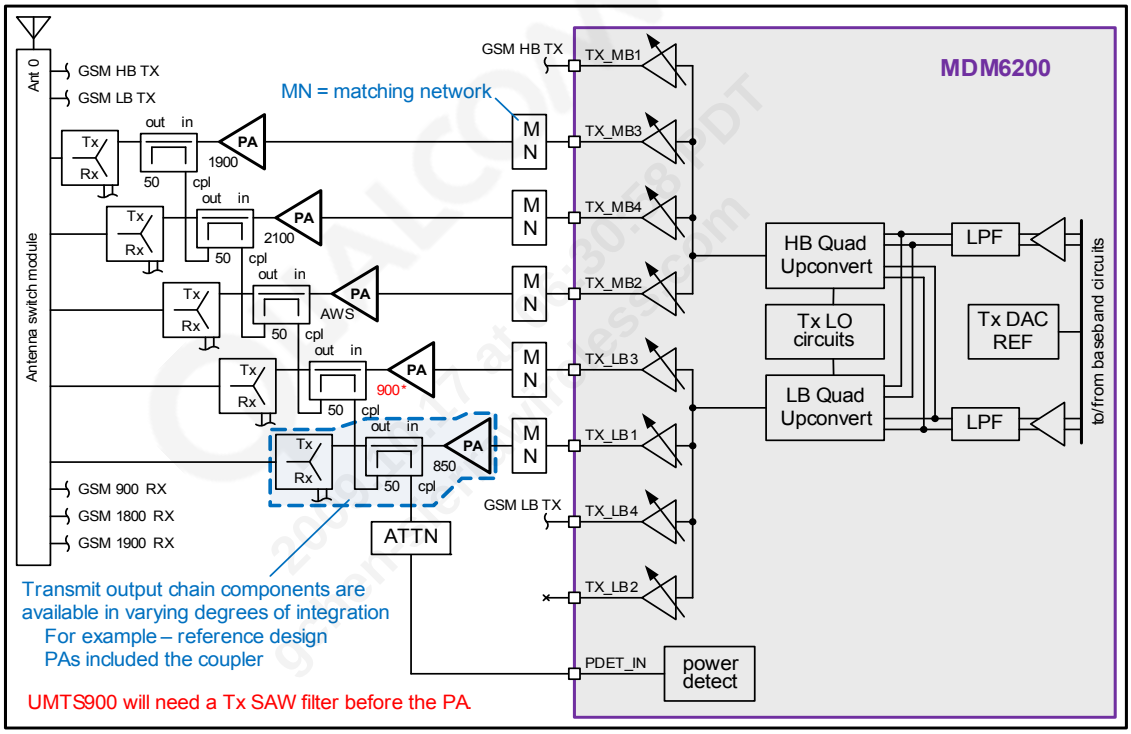


Figure 9-4 UMTS transmitter and example application (MDM6200 device)

9.3.4 UMTS transmitter connections

Recommended UMTS transmitter output matching-network topologies are shown in Figure 9-5.

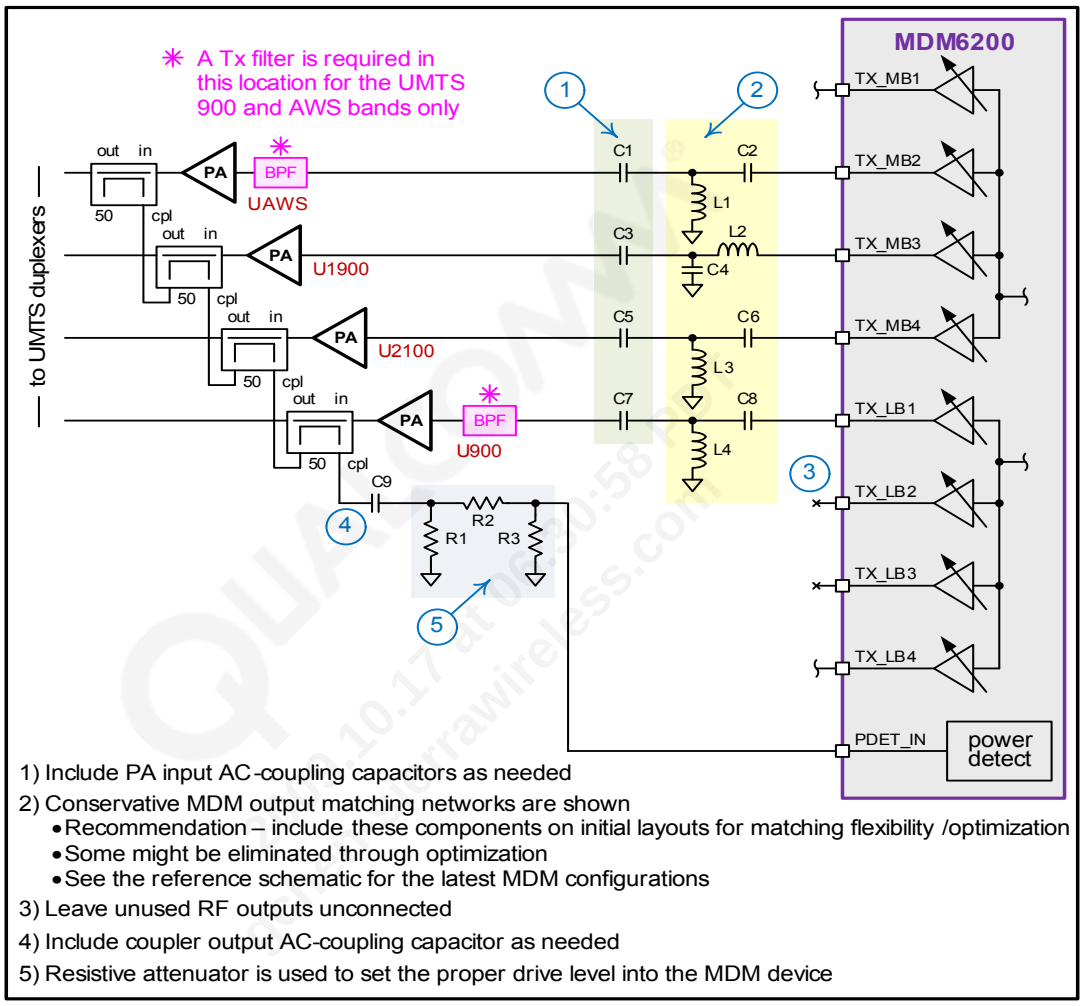


Figure 9-5 Expected UMTS RF transmitter matching-network topologies

9.3.5 GSM transmitters

The GSM Tx paths within the MDM device use small-signal polar, where the phase and envelop signal are combined at the mixer inside the chip. A ***GSM linear PA is required*** for this transmit architecture.

All MDM variants use the same pins for GSM Tx outputs, require using a GSM linear PA, and they all require the same external circuitry to deliver the transmit signals to the primary antenna. An example application based upon the MDM6200 device is shown in Figure 9-6.

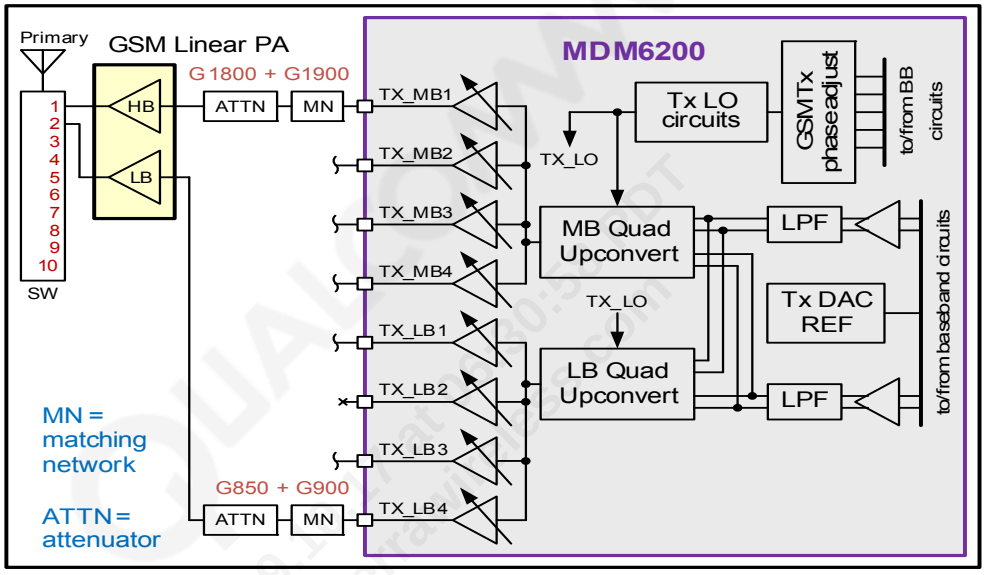


Figure 9-6 GSM transmitter and example application

The GSM Tx output ports are single-ended with 50-Ohm nominal impedance. Each requires a simple matching network and resistive attenuator to interface with its power amplifier. See Section 9.3.6 for matching-network topologies and guidelines on handling unused RF output ports.

The driver amplifiers end the signal-path circuitry within the MDM device, but the external transmit paths continue: MDM output — matching network — resistive attenuator — linear power amplifier — antenna. Note that a linear PA is required.

9.3.6 GSM transmitter connections

Recommended GSM transmitter output matching-network topologies are shown in Figure 9-7, along with guidelines on handling any unused Tx ports.

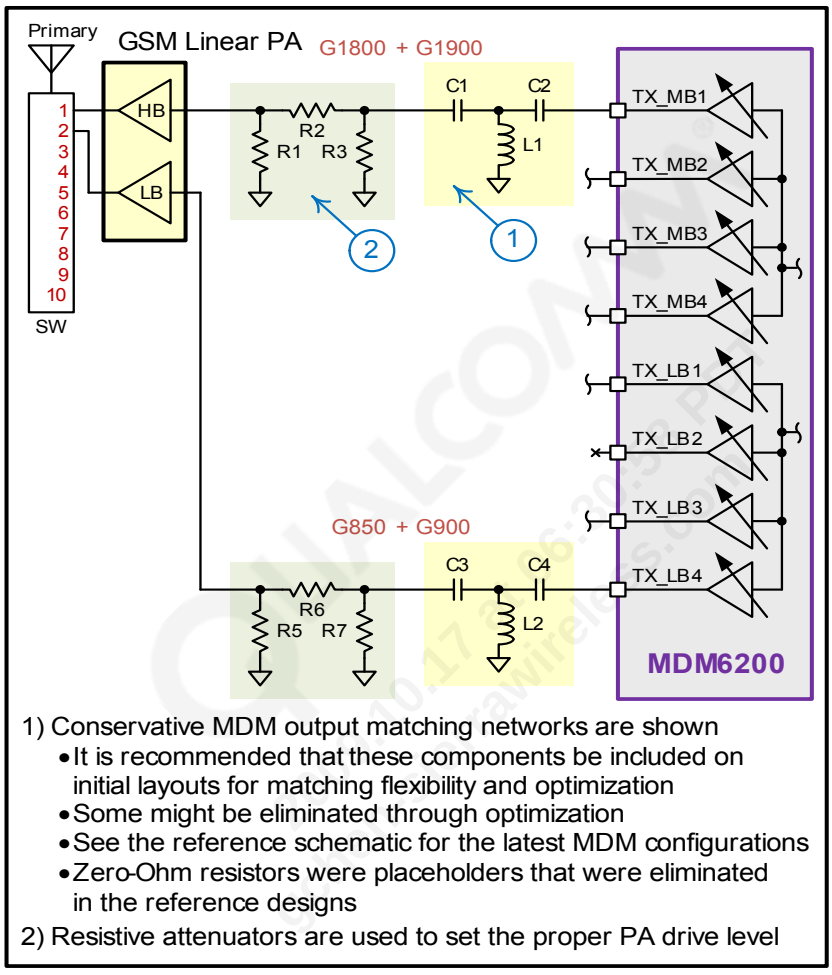


Figure 9-7 Expected GSM RF transmitter matching-network topologies

The GSM Tx output matching networks depend upon the operating frequency range, external front-end devices (regardless of their degree of integration), PCB material and stack-up, layout details, and matching components used (which vary from supplier to supplier).

9.4 Tx power detector

The MDM6x00 device includes an integrated RF transmit power-detector circuit. This function is accessed via the PDET_IN pin; it is described in Section 9.3.1 and Section 9.3.2. The couplers must be connected in a daisy-chain fashion so that a single Tx power detect can support all bands. An example daisy-chain set of port assignments is shown in Figure 9-3.

9.5 Tx LO circuits

The Tx LO frequency synthesizer generates the transmit LO signals, and then distribution circuits deliver the quadrature LO to the upconverters. All PLL circuits are on-chip, including the oscillating signal sources, digital PLL circuits, and circuits for adjusting the GSM transmit phase. External circuits are not required, not even an RC network for the loop filter. A high-level functional block diagram is shown in Figure 9-8.

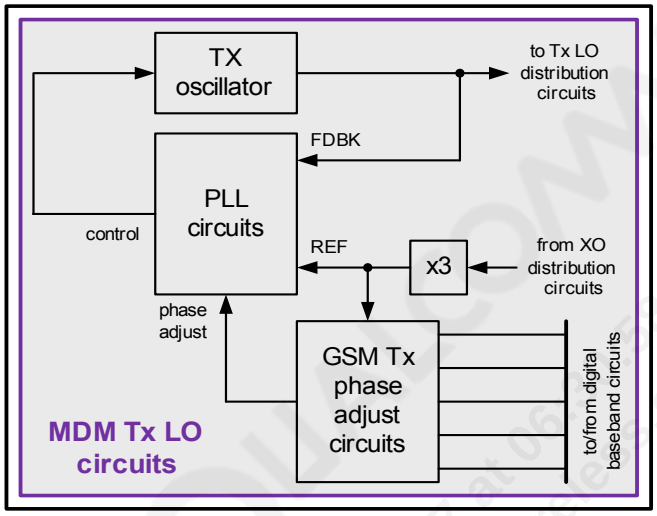


Figure 9-8 Tx LO functional block diagram

The buffered 19.2 MHz XO signal is frequency-multiplied and then used as the synthesizer input (REF), the frequency reference to which the PLL is phase and frequency locked. Another PLL input varies as the loop acquires lock, and is generated by counting the oscillator output pulses (at FDBK). The closed-loop will force the oscillator output to match the programmed value. If the loop is not locked, the error between inputs creates an error signal at the output of the PLL. This error signal tunes the output frequency so that the error is decreased. Ultimately, the loop forces the error to approach zero and the PLL is phase and frequency locked.

The UHF output of the transmit LO synthesizer feeds a distribution network that drives the active upconverter LO ports with the appropriate frequency and amplitude. The LO signals applied to the upconverter are differential with a quadrature phase relationship.

During GSM operation, the phase is dynamically adjusted using a 5-line digital interface with the digital baseband circuits. The GSM phase-adjust circuitry also uses the frequency-multiplied reference signal.

9.6 RF receive signal paths

The MDM6x00 device provides eleven RF receive input ports that are split into four categories:

- CDMA and UMTS primary receivers (PRx)
- CDMA and UMTS diversity receivers (DRx)
- GSM receivers (can use PRx or DRx RF inputs)
- GNSS receiver

The MDM device includes Qualcomm's intelligent receiver (IntelliCeiver) technology. The RF transceiver block supports this CDMA feature with:

- On-chip circuits that detect jammers, allowing adjustment to the receiver paths' bias conditions and linearity (IIP2 and IIP3) for optimal performance.
- Adjustable bias conditions that are set for adequate linearity (given the existing jammer conditions) with the minimum-possible DC power consumption.

IntelliCeiver support is discussed further in Section 9.6.1.3.

The Rx signal paths are described in the following sections, including band-specific and mode-specific input circuits between the antenna and the MDM input ports.

9.6.1 Primary CDMA and UMTS receivers

9.6.1.1 CDMA primary receivers

An example CDMA PRx application based upon the MDM6600 device is shown in Figure 9-9.

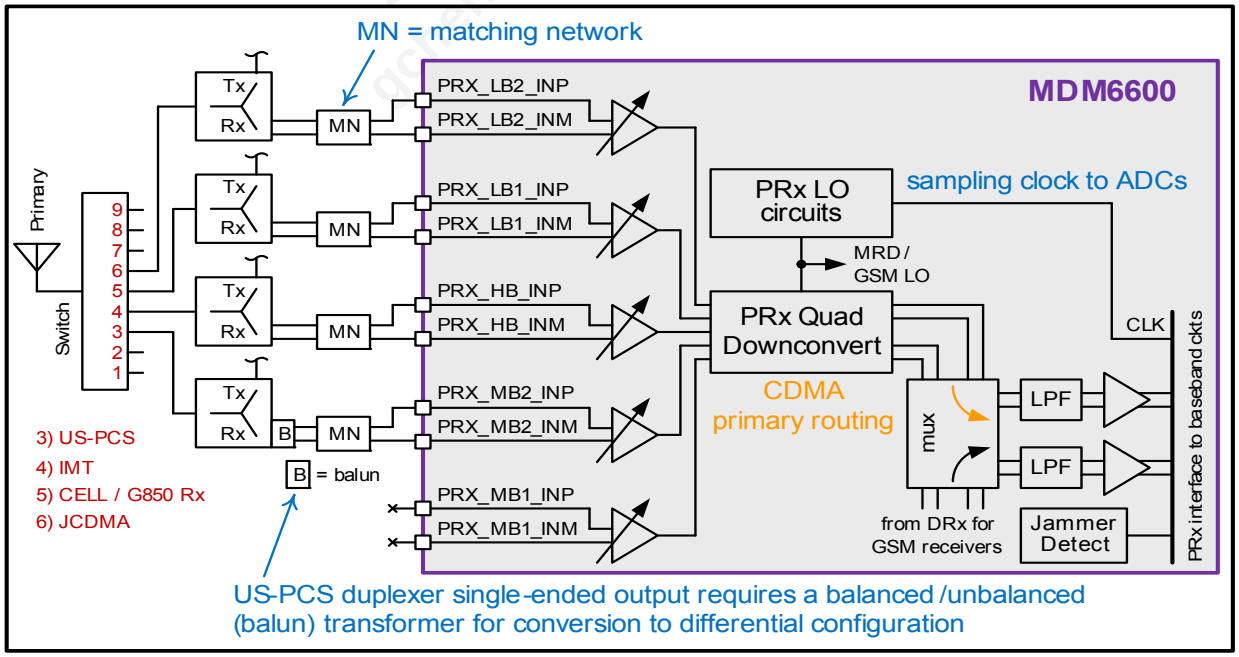


Figure 9-9 CDMA primary receivers and example application (MDM6600 device)

An antenna switch selects the RF operating band, and then band-specific duplexers filter the incoming Rx signal while simultaneously routing the transmit signal to the antenna. The duplexer should provide a differential Rx output. In order to meet single-tone desensitization requirements, the PCS reference design required a FBAR duplexer having a single-ended output. A balanced/unbalanced (balun) transformer was added to convert its single-ended output into the desired differential configuration.

Each primary receive signal is routed from its duplexer to its MDM LNA through a differential matching network (MN) that optimizes the power transfer into the gain-stepped LNA. A carefully controlled differential match is required between the duplexer and the LNA input to maintain the MDM device's high common-mode rejection, Tx isolation, out-of-band suppression, and second-order intermodulation performance.

The MDM device includes advanced receiver technology that eliminates the need for off-chip, inter-stage bandpass filters. Using this technology, the LNA output drives the downconverter directly. Each Rx path includes a dedicated LNA with a specific band assignment.

The downconverter's RF circuitry includes a gain-stepped amplifier that supplements the LNA gain steps to further extend the receiver dynamic range. The quadrature downconverter translates the LNA's RF signal directly to baseband, producing two analog output components: in-phase (I) and quadrature (Q). The baseband signals are routed through multiplexer circuits that only have significance during GSM operation (see Section 9.6.3); the signals continue to lowpass filters whose passband and stopband characteristics are programmed for the CDMA waveform. Both filter outputs are buffered and routed to the baseband circuits' analog-to-digital converters for digitization and further processing.

The primary receiver, diversity receiver, and GNSS receiver can operate simultaneously (except for simultaneous SHDR and GNSS) to enable modes such as the following:

- Simultaneous mobile receive diversity (MRD) and GNSS – enables coherent CDMA processing of the primary and diversity paths by using the same PRx LO signal for both downconverters at the same time as GNSS processing, with the GNSS path using the SHDR/GNSS LO for downconversion.
- Simultaneous hybrid dual receiver (SHDR) operation – enables phone processing in two RF operating bands or channels at the same time (one in the primary receiver and one in the secondary or diversity receiver). See the note below for more details.
- Full-time SHDR (FTS) – FTS details will be provided in subsequent revisions of this document. See *Documentation Definition for HMA Readme* (80-VH591-9) for further FTS-related details.
- Simultaneous GNSS – allows GNSS processing at the same time as phone processing
- Standalone GNSS – allows GNSS processing without location assistance

The receiver LO signals are generated by circuits described in Section 9.7, and delivered to the downconverter circuits at the correct frequency, with the proper phase relationship, and with adequate drive level.

NOTE The circuits that generate the diversity downconverter's LO in SHDR mode share their PLL with the GNSS LO circuits; therefore, GNSS and SHDR functions cannot be supported at the same time. Only a SHDR path or the GNSS path can be active at any one time.

All receiver input ports have a differential configuration with 100 Ω nominal impedance. Each requires a simple matching network to interface with its RF front-end path. See Section 9.6.1.2 for matching-network topologies and design comments.

The PRx matching networks depend upon the operating frequency range, external front-end devices (regardless of their degree of integration), PCB material and stack-up, layout details, and matching components used (which vary from supplier to supplier).

9.6.1.2 CDMA primary-receiver connections

Recommended CDMA primary-receiver matching-network topologies are shown in Figure 9-10, along with some design comments.

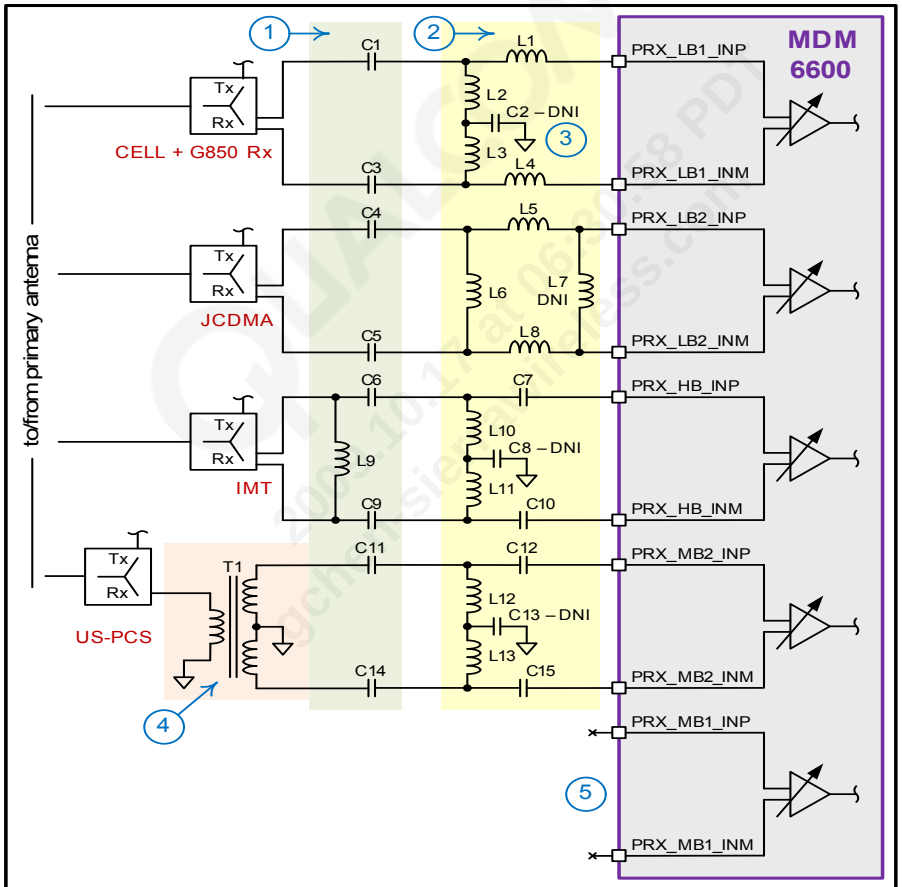


Figure 9-10 Expected CDMA RF primary-receiver matching-network topologies

9.6.1.3 IntelliCeiver support

During CDMA operation, the primary baseband path includes circuits that detect and report the presence of jammers. Jammer conditions are communicated to the baseband circuits as follows:

- When the CDMA receiver is in one of its lower-current modes and a new or increased jammer level exceeds the detection threshold, the JDET_INT interrupt is sent to the baseband circuits. After they receive the interrupt, they communicate through an internal interface to learn that the RF transceiver has detected a jammer. The baseband circuits quickly set the primary receiver into one of its higher-current modes (current modes are explained below).
- When the CDMA receiver is in one of its higher-current modes, the baseband circuits check the RF transceiver for its jammer condition. When the reported jammer level stays below the detection threshold long enough, the baseband circuits set the primary receiver into one of its lower-current modes.

The RF transceiver and baseband circuits work together to incorporate IntelliCeiver technology. Jammer detection is just a small part of this innovative technology. Using the reported jammer conditions, the baseband circuits control the RF transceiver's operational status to optimize receiver performance with the minimum possible DC power consumption. Features of the IntelliCeiver technology include:

- Jammer detecting and reporting methodology to ensure:
  - The primary receivers are always protected against high-level jammers by quickly switching the active Rx path from a lower-current mode to a higher-current mode.
  - The primary receiver is not released from its higher-current mode until an acceptable jammer level is confirmed over a sufficient period of time.
  - When the jammer conditions are not severe, the receiver is allowed to operate in one of its lower-current modes thereby reducing its average DC power dissipation.
- The methodology just described is essentially a fast attack/slow decay algorithm. The attack occurs quickly, setting the receiver to a higher-current mode based on an interrupt signal. The decay is slow, allowing the receiver to operate in a lower-current mode only after careful and repeated confirmation that the jammer environment is not severe.
- The algorithm includes hysteresis to avoid rapidly alternating between higher-current and lower-current modes.
- The RF transceiver is configured via an internal bus to operate in different modes, depending upon the detected jammer levels:
  - High-current modes optimize the primary receiver's linearity (IIP2 and IIP3) performance.
  - Low-current modes minimize DC power consumption while maintaining adequate linearity performance to meet the existing jammer conditions.
- Adaptively adjusting the primary Rx operating mode assures adequate linearity (given the existing jammer conditions) with the minimum possible DC power dissipation.
- Thresholds are set such that all RF transceivers transition from lower-current to higher-current modes at sufficiently low jammer levels to ensure that even worst-case devices have adequate performance to pass all linearity tests required by the standards.

9.6.1.4 UMTS primary receivers

The description presented in Section 9.6.1.1 for the CDMA PRx paths applies to the UMTS PRx paths as well, with a couple exceptions:

1. If the US-PCS band is not supported but the U1900 band is supported, a U1900 duplexer is available with a differential Rx port, so a balun is not required.
2. If CDMA is not supported, the jammer detection output would not be used (unconnected). WCDMA operation in the UMTS bands does not support the IntelliCeiver technology.

An example application based upon the MDM6200 device is shown in Figure 9-11.

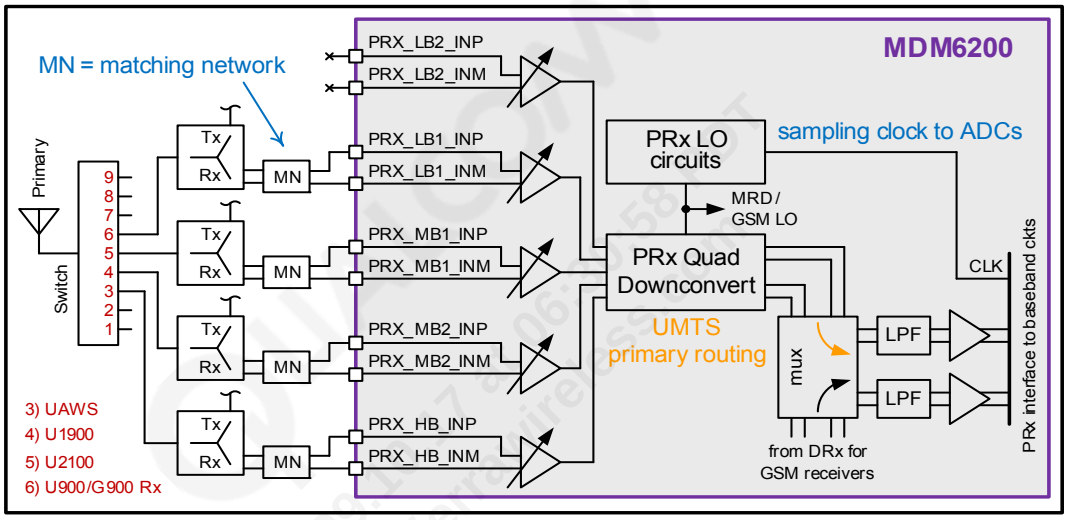


Figure 9-11 UMTS primary receivers and example application (MDM6200 device)

9.6.1.5 UMTS primary-receiver connections

Recommended UMTS primary-receiver matching-network topologies are shown in Figure 9-12.

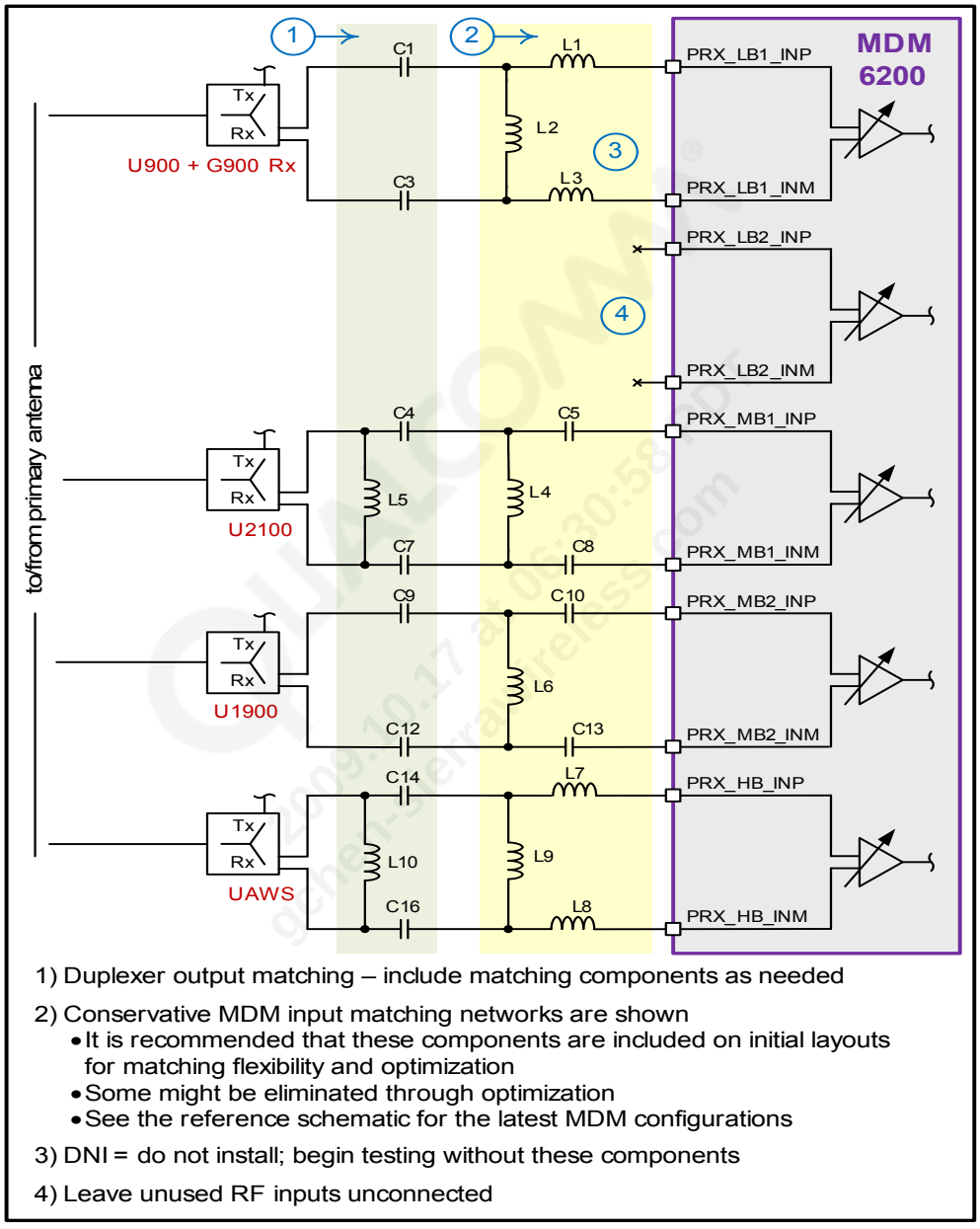


Figure 9-12 Expected UMTS RF primary-receiver matching-network topologies

9.6.2 CDMA and UMTS diversity receivers

9.6.2.1 CDMA diversity receivers

An example CDMA DRx application based upon the MDM6600 device is shown in Figure 9-13.

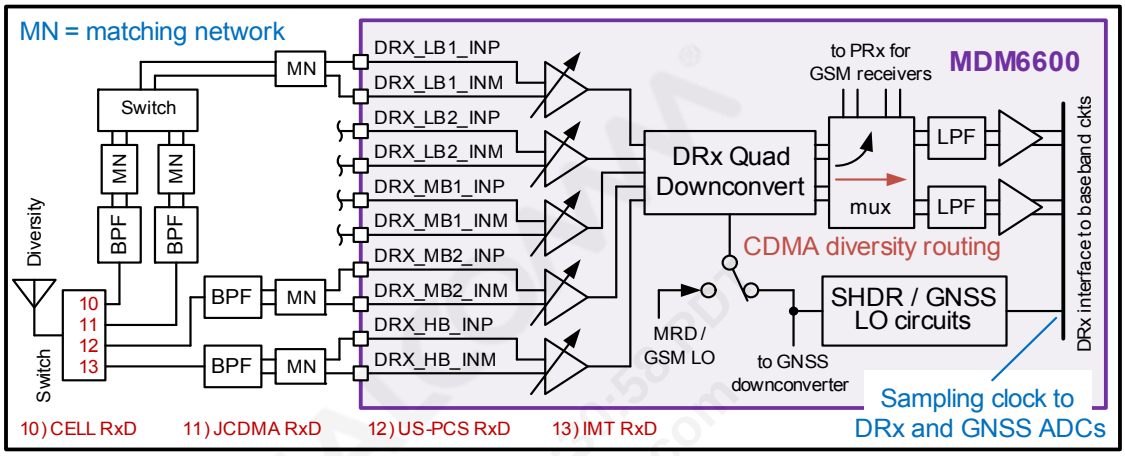


Figure 9-13 CDMA diversity receivers and example application (MDM6600 device)

The diversity (or secondary) receiver paths are similar to the primary paths. The most significant difference between the front-end implementations is the primary paths' ability to support handset transmissions; the diversity paths are receive-only.

An antenna switch selects the RF operating band, and then a bandpass SAW filters the incoming Rx signal. This filter should provide a differential output.

One low band input can support two bands by using an external switch to select the active band. The MB2 input is dedicated to the 1900 MHz band (US-PCS for CDMA operation), while the HB input can be shared by the AWS and 2100 MHz bands (AWS and IMT for CDMA operation). It is possible to share one input between these two bands because their Rx operating range overlaps (AWS = 2110 to 2155 MHz; IMT = 2110 to 2170 MHz).

Each diversity receive signal is routed from its filter (or switch) to its MDM LNA through a differential matching network (MN) that optimizes the power transfer into the gain-stepped LNA. A carefully controlled differential match is required to maintain the MDM device's common-mode rejection, Tx isolation, out-of-band suppression, and second-order intermodulation performance.

Like the primary paths, the diversity paths do not require external filters between their LNAs and downconverter; the LNA outputs drive the downconverter directly. Each Rx path includes a dedicated LNA, but all five diversity paths share a single downconverter.

The downconverter's RF circuitry includes a gain-stepped amplifier that supplements the LNA gain steps to further extend the receiver dynamic range. The quadrature downconverter translates the LNA's RF signal directly to baseband, producing two analog output components: in-phase (I) and quadrature (Q). The baseband signals are routed through multiplexer circuits that only have significance during GSM operation (see Section 9.6.3); the signals continue to lowpass filters whose passband and stopband characteristics are programmed for the CDMA waveform. Both filter outputs are buffered and routed to the baseband circuits' analog-to-digital converters for digitization and further processing.

The receiver LO signals are generated by circuits described in Section 9.7, and delivered to the downconverter circuits at the correct frequency, with the proper phase relationship, and with adequate drive level.

All receiver input ports have a differential configuration with 100 Ω nominal impedance. Each requires a simple matching network to interface with its RF front-end path. See Section 9.6.2.2 for matching-network topologies and design comments.

The DRx matching networks depend upon the operating frequency range, external front-end devices (regardless of their degree of integration), PCB material and stack-up, layout details, and matching components used (which vary from supplier to supplier).

9.6.2.2 CDMA diversity-receiver connections

Recommended CDMA diversity-receiver matching-network topologies are shown in Figure 9-14, along with some design comments.

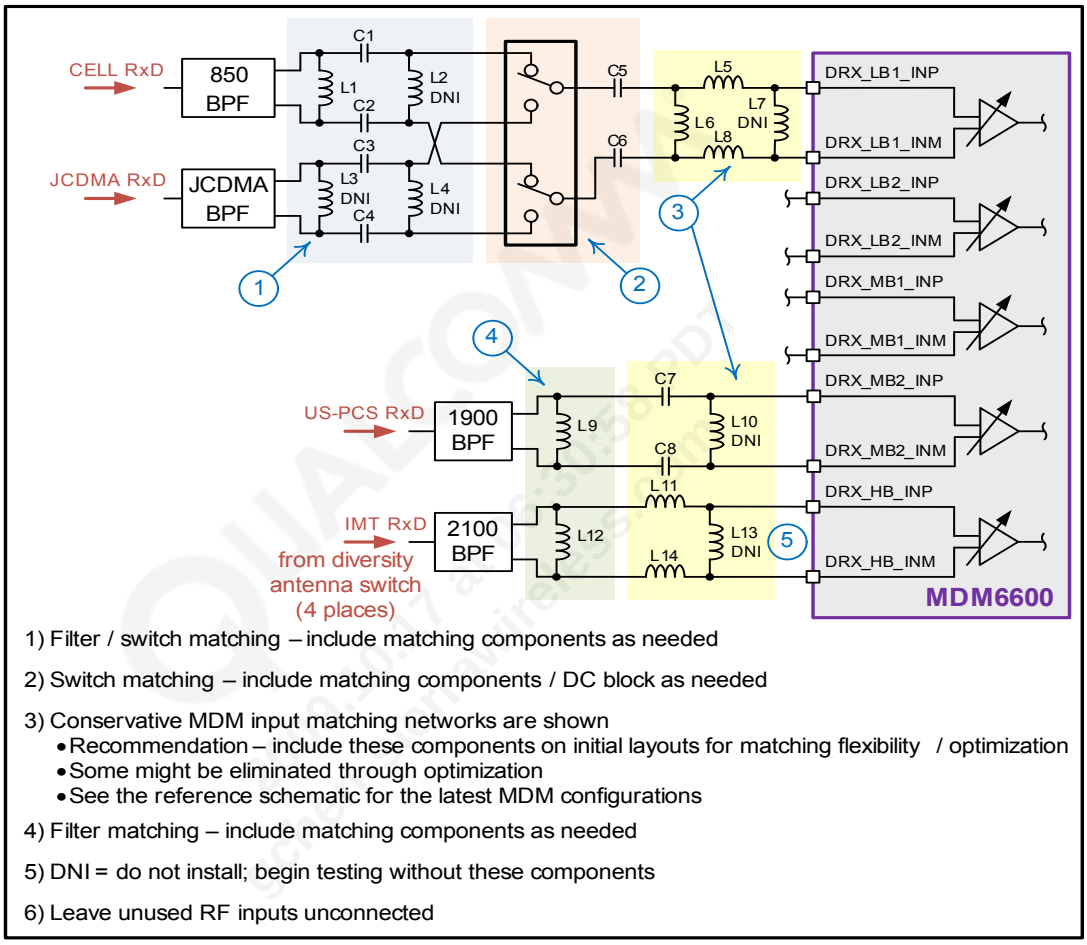


Figure 9-14 Expected CDMA RF diversity-receiver matching-network topologies

9.6.2.3 UMTS diversity receivers

The description presented in Section 9.6.2.1 for the CDMA DRx paths applies to the UMTS DRx paths as well. An example application based upon the MDM6600 device is shown in Figure 9-15.

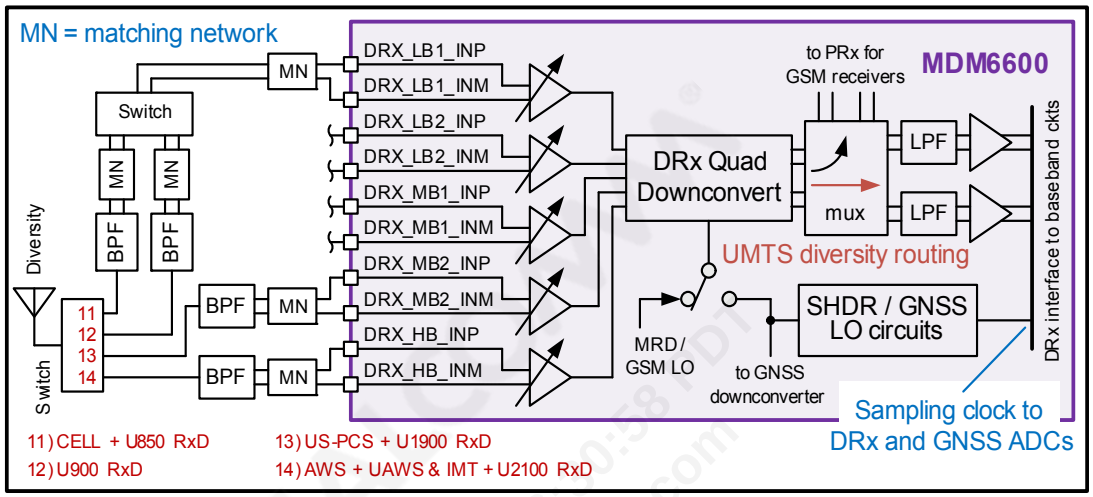


Figure 9-15 UMTS diversity receivers and example application (MDM6600 device)

9.6.2.4 UMTS diversity-receiver connections

Recommended UMTS diversity-receiver matching-network topologies are shown and described in Figure 9-16.

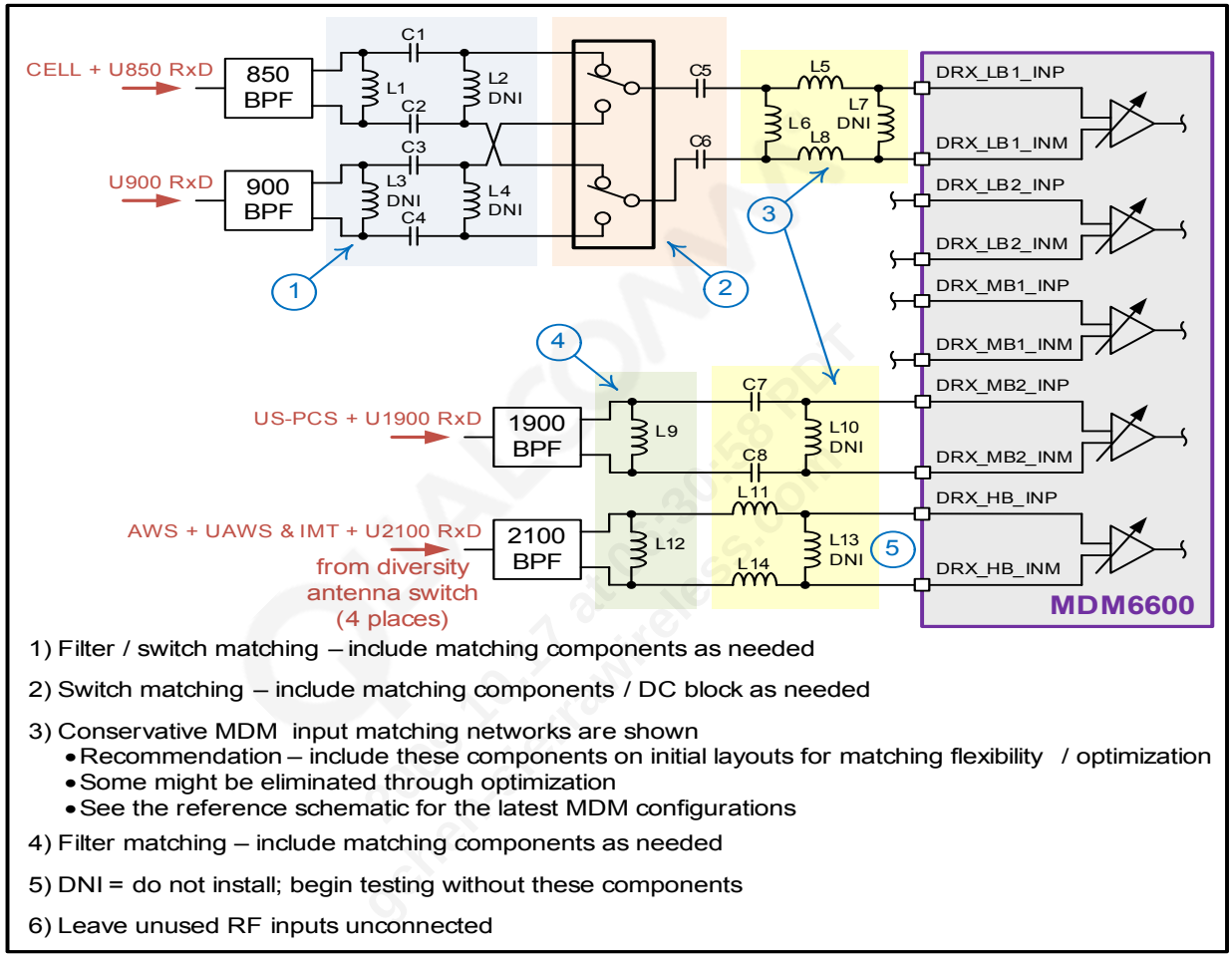


Figure 9-16 Expected UMTS RF diversity-receiver matching-network topologies

9.6.3 GSM receivers

Since the GSM-850 band covers the same frequency range as the CDMA cellular band and the UMTS-850 band, the PRx path defined within Section 9.6.1.1 allows a shared duplexer for GSM-850 operation. Sharing the GSM front-end with a CDMA or UMTS path is called *co-banding*; this technique is discussed further in Section 9.6.3.2.

In the example shown within Figure 9-17, the GSM-900, -1800, and -1900 bands use the MDM diversity circuits that were described within Section 9.6.2.1 for the CDMA DRx paths, with two key exceptions:

- The downconversion LO is sourced by the primary receiver LO circuits, not the SHDR/GNSS LO circuits.
- The baseband multiplexers are used to route the GSM signals to the baseband circuit's primary receiver input (rather than the baseband diversity receiver).

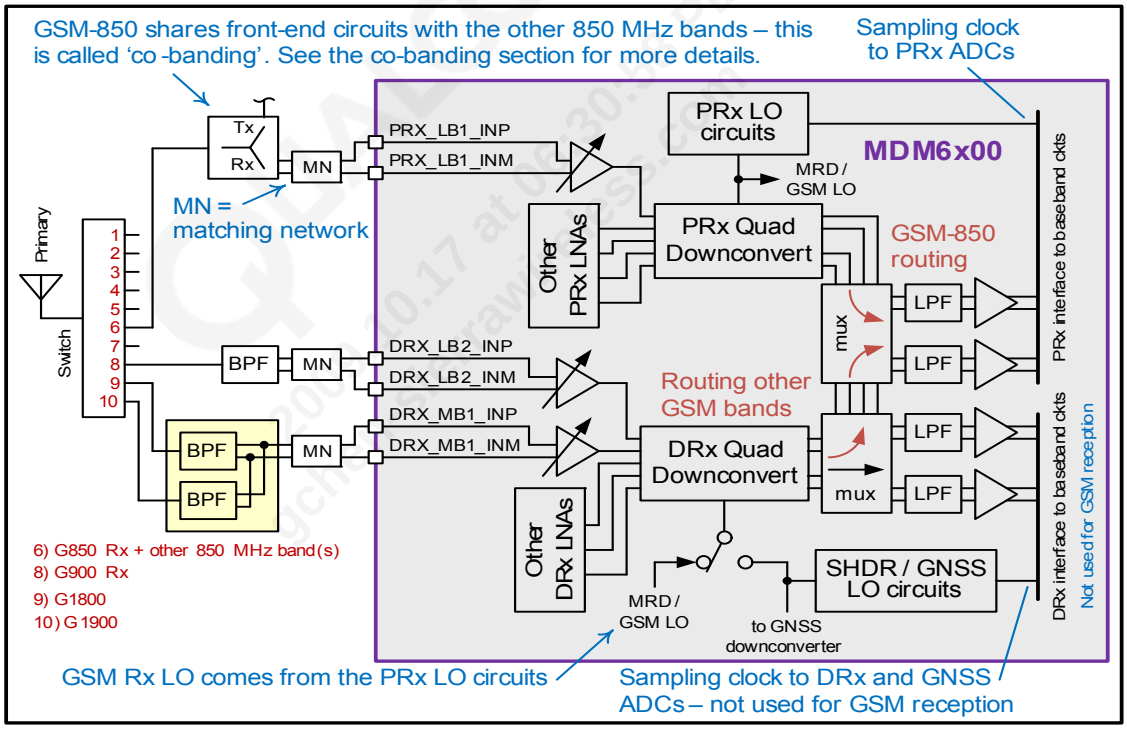


Figure 9-17 GSM receiver and example application

The GSM receivers' front-end circuits are different, depending upon the operating band:

- The GSM 850 path shares the CDMA cellular and/or UMTS-850 path.
- The GSM 900 path is simply a bandpass filter with a matching network into the MDM device; it has single-ended input and a differential output.
- The GSM 1800 and GSM 1900 bandpass filters are combined within a single module. Each filter has a single-ended input and differential output. The outputs are multiplexed internally and routed through a matching network into a single MDM input port

All receiver input ports have a differential configuration with 100 Ω nominal impedance. Each requires a simple matching network to interface with its RF front-end path. See Section 9.6.3.1 for matching-network topologies and design comments.

The GSM Rx matching networks depend upon the operating frequency range, external front-end devices (regardless of their degree of integration), PCB material and stack-up, layout details, and matching components used (which vary from supplier to supplier).

9.6.3.1 GSM receiver connections

Three GSM receiver input matching-network topologies are shown and described in Figure 9-18; the GSM 850 path is not included, since it shares the CDMA cellular and/or UMTS 850 RF front-end (refer to Section 9.6.1.5 and Figure 9-12).

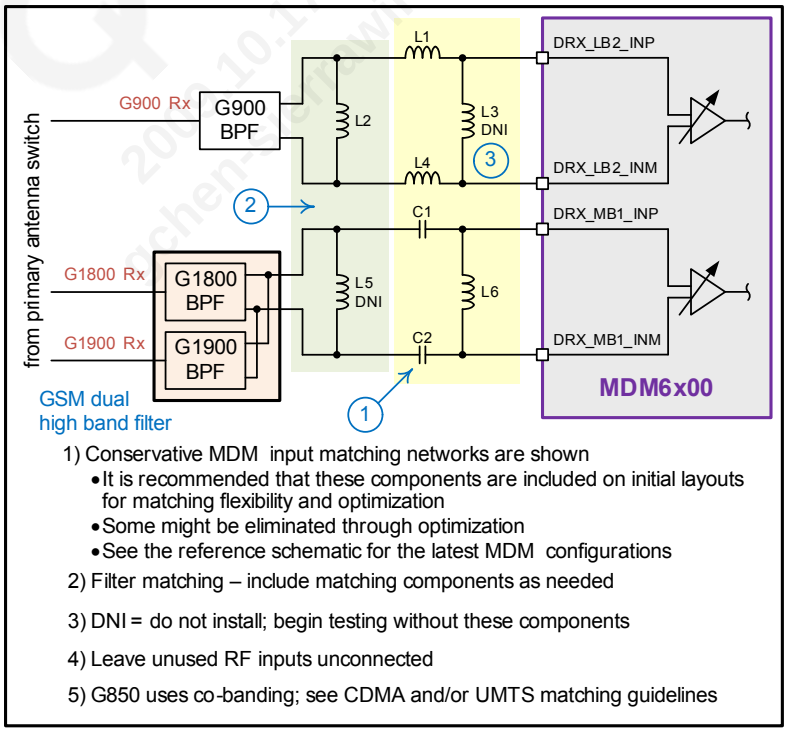


Figure 9-18 Expected GSM RF receiver matching-network topologies

9.6.3.2 GSM co-banding

Software builds might allow up to three GSM bands to share front-end components and MDM paths with the phone's other air interface modes: the 850, 900, and 1900 MHz bands. This technique is called co-banding, and it eliminates external components as shown and discussed in Figure 9-19.

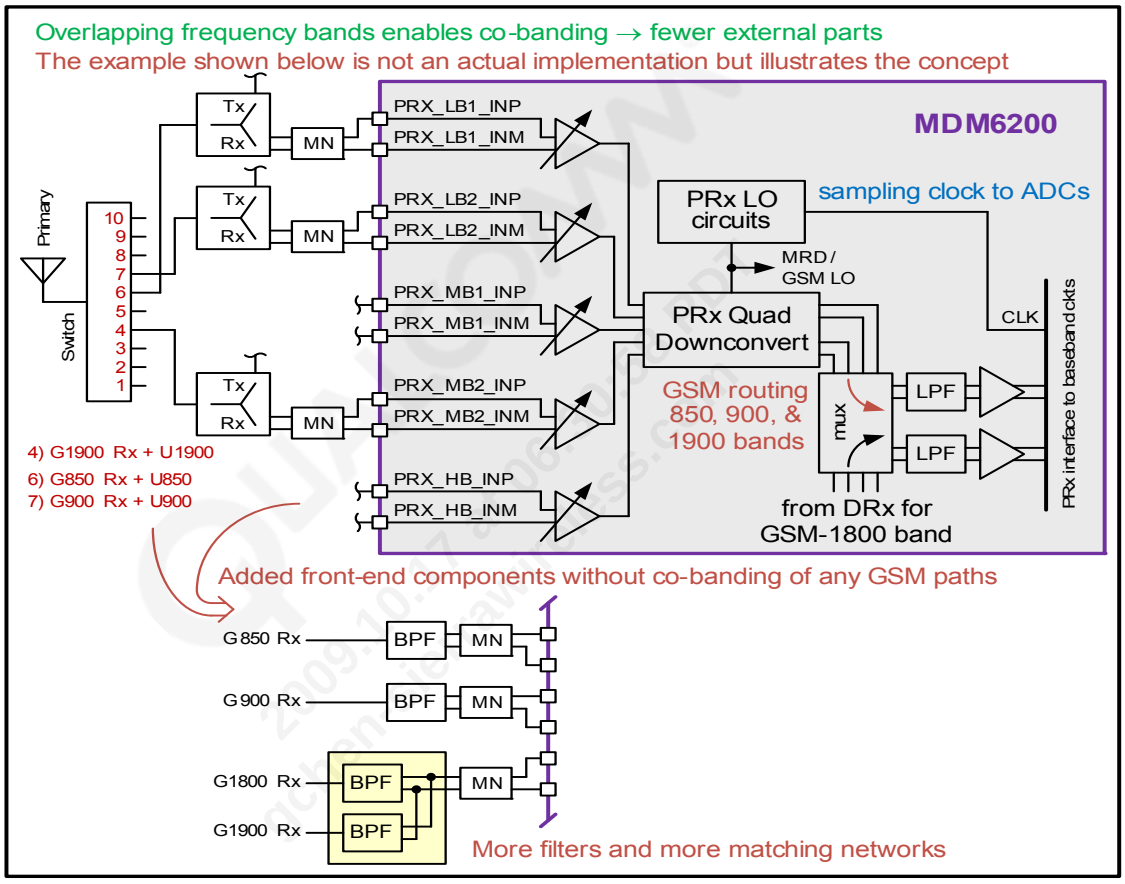


Figure 9-19 GSM co-banding example

NOTE Co-banding can only be used when supported by software.

9.6.4 GNSS receiver

The MDM device includes a fully integrated global navigation satellite system (GNSS) solution that supports generation 8 (Gen8 – GPS L1 + GPS L1 wide + GLONASS). In the most severe condition – during GSM high-power transmit bursts – the GNSS processing is synchronously disabled (blanking mode).

The GNSS path within the MDM device is very similar to that described within Section 9.6.2.1 for the CDMA DRx paths. The only difference: the GNSS path has dedicated RF, downconverter, baseband, and ADC circuits (rather than circuits shared with phone paths). This allows simultaneous MRD plus GNSS operation.

An example application is shown in Figure 9-20.

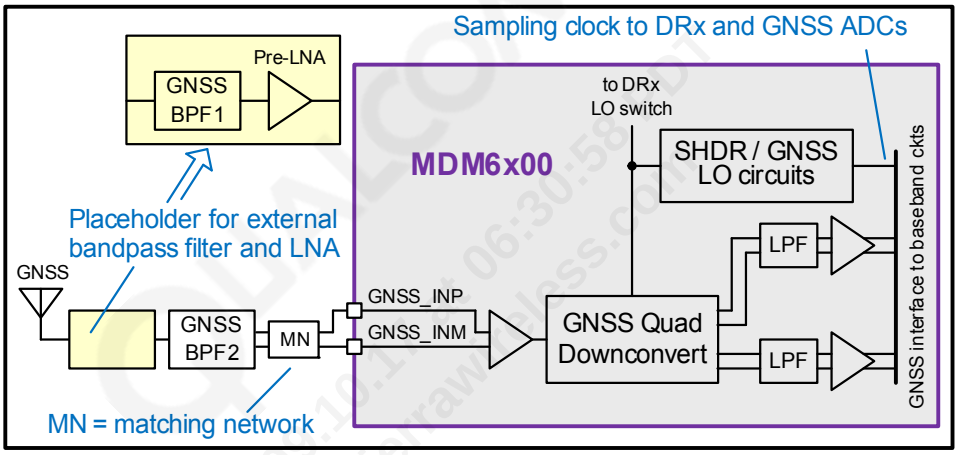


Figure 9-20 GNSS receiver and example application

The GNSS path has its own antenna and receiver front-end circuit. In some applications, only the second bandpass filter and matching network might be needed. This filter is like those filters used at the DRx inputs – a single-ended input and differential output. The GNSS input port has a differential configuration with a 100 Ω nominal impedance. A simple matching network is required; see Section 9.6.4.1 for the matching-network topology and design comments.

The GNSS Rx matching network depends upon the external front-end devices, PCB material and stack-up, layout details, and matching components used (which vary from supplier to supplier).

Qualcomm's design goal is to use just one external bandpass filter, thereby avoiding the external pre-LNA and second bandpass filter. These extra external components should be included as a precaution within initial product designs. These external circuits are detailed in *MDM6200 and MDM6600 Mobile Data Modem Schematic Design Example* (80-VR001-42).

9.6.4.1 GNSS receiver connections

The recommended GNSS receiver input matching-network topology is shown and described in Figure 9-21.

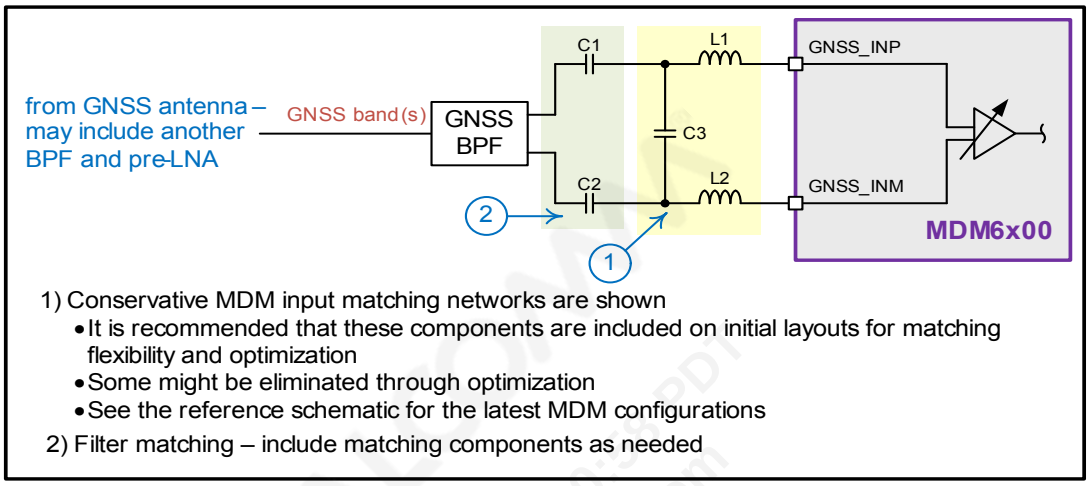


Figure 9-21 Expected GNSS RF receiver matching-network topology

9.7 Rx LO circuits

The MDM device includes two Rx LO frequency synthesizers:

1. The dedicated primary receiver (PRx) LO circuit
2. The shared SHDR plus GNSS LO circuit

A high-level functional block diagram is shown in Figure 9-22. The functional description given in Section 9.5. is valid, but some differences between the Tx LO synthesizer and the Rx LO circuits are discussed below.

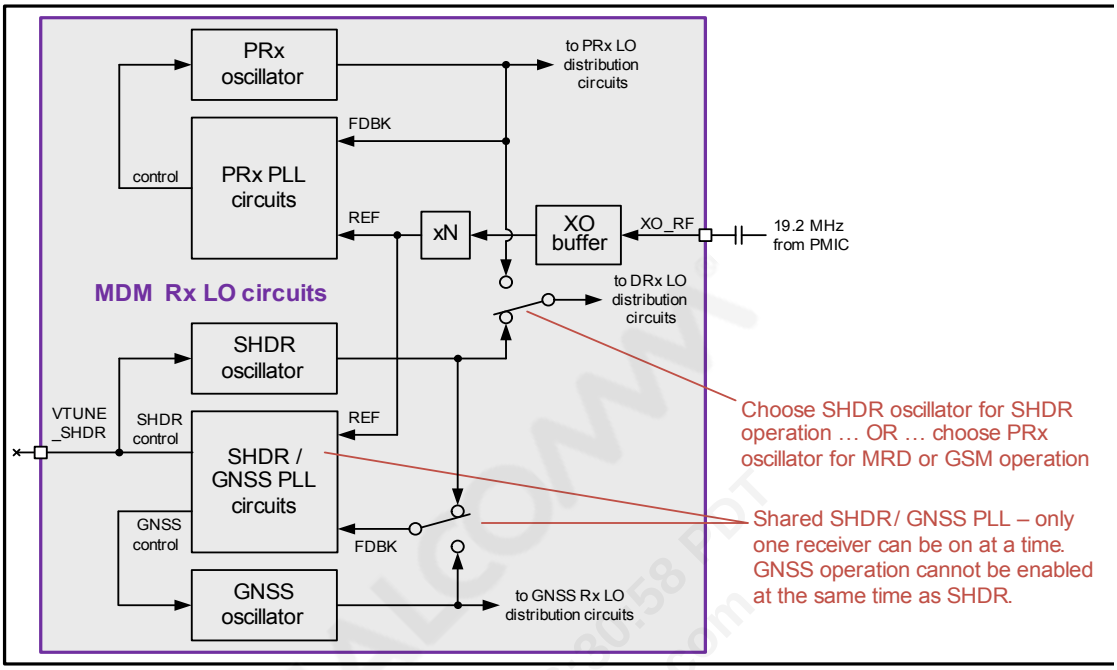


Figure 9-22 Rx LO functional block diagram

In addition to driving the LO distribution circuits for PRx downconversion, the PRx oscillator output is also used as the DRx LO during mobile receive diversity (MRD) or GSM operation.

Although there are separate oscillators for SHDR and GNSS operation, they share a single set of phase-locked loop circuits. Since the PLL is shared, only one receiver can be supported at any one time – either a SHDR path or the GNSS path can be active.

The SHDR oscillator control voltage is brought off-chip to allow supplemental capacitance to be added, though this is not expected to be used.

9.8 Multimode (CDMA + UMTS) designs

The instructions presented individually for CDMA and UMTS receivers and transmitters apply directly to multimode designs.

10 Power and Ground

MDM-appropriate DC voltages are conditioned using the PM8028 IC; see the dedicated PMIC documents (80-VN204-x) for detailed circuit-design suggestions. This chapter details methods for distributing the supply voltages from the PM8028 IC to the MDM6x00 device, and describes the MDM ground connections.

10.1 Power conditioning

The power management circuits use an external supply or the handset's battery to establish the phone's raw DC power. The raw power is applied to regulator circuits, filters, and bypass capacitors that provide clean DC power to all the other handset functions. The MDM6x00 device uses several PM8028 voltage-regulator outputs (as shown in Figure 10-1).

Each regulator must be connected immediately to its recommended output load capacitor (as well as any additional filtering), per the PM8028 documentation (80-VN204-x). Additional filtering and bypassing of the MDM supply voltages are described in Section 10.2.

10.2 Power-supply distribution

The recommended power-supply connections are shown in Figure 10-1. Refer to the *MDM6200 Mobile Data Modem Reference Schematic* (TBD) for bypass capacitor recommendations for each distribution node and power pin. General parts placement and PCB layout guidelines are given in the *MDM6200 and MDM6600 Mobile Data Modem Design Guidelines* (80-VR001-5).

NOTE Initial layouts should include extra bypass capacitors to ensure adequate performance. Extensive testing during product development will prove which components, if any, can be removed without degrading performance. Each handset design should undergo proof-of-design testing to arrive at the optimal set of supply components, depending upon PCB characteristics, trace routings, operating frequencies, etc.

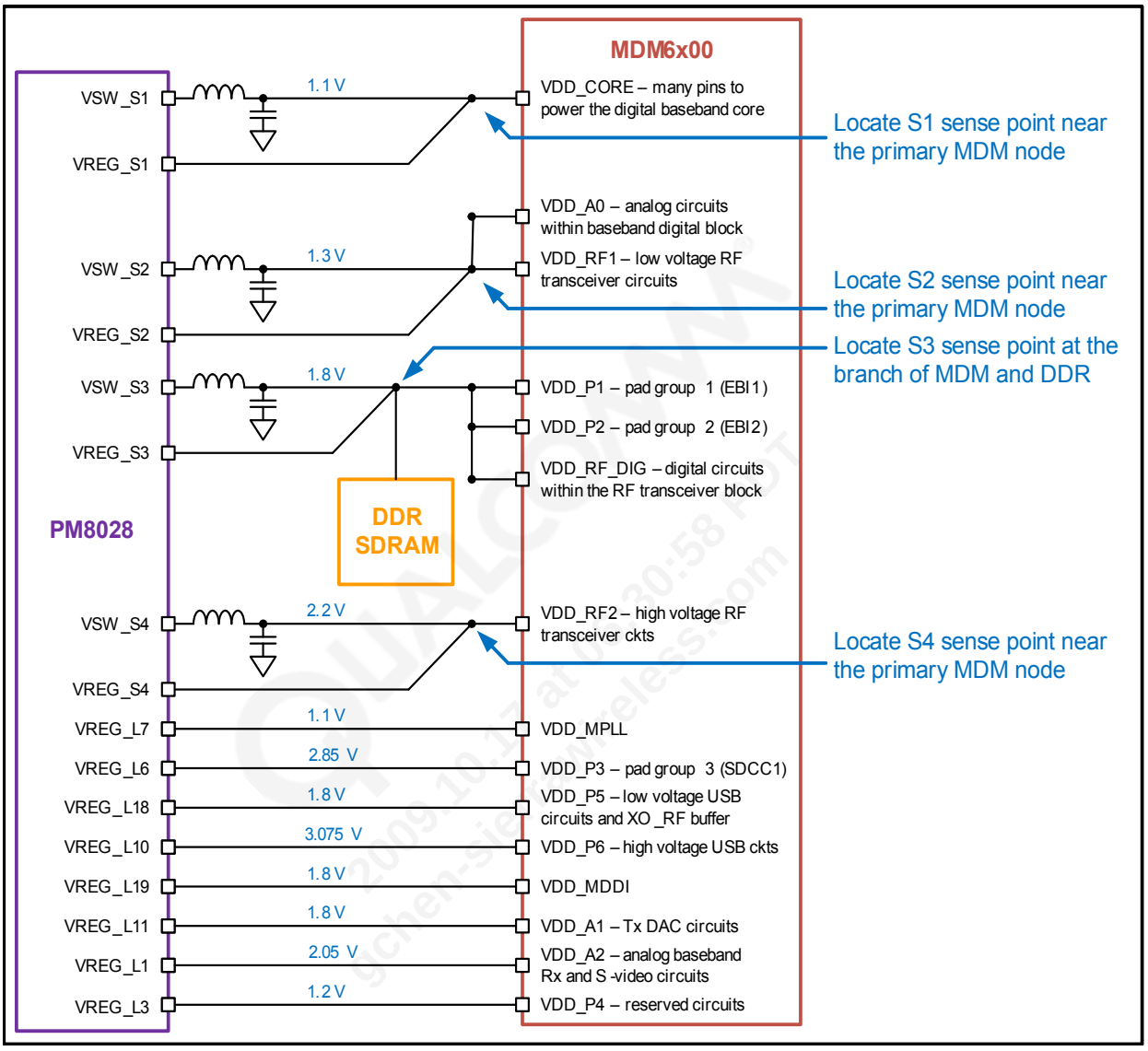


Figure 10-1 Power-supply distribution

10.3 Ground connections

The MDM6x00 device has four types of ground pins (Figure 10-2):

- Digital baseband ground (GND_DIG, shown as grey) – These pins should be connected to a *surface-layer* fill area, and then via down to the primary inner-layer ground plane.
- RF transceiver ground (GND_RF, shown as blue) – These pins should be connected directly to the *inner-layer* ground planes, not the surface-layer digital ground fill.
- Internal ground (GND, shown as orange) – These pins are connected internally to ground, but handset designers have the option to leave them unconnected externally if doing so improves the PCB layout.

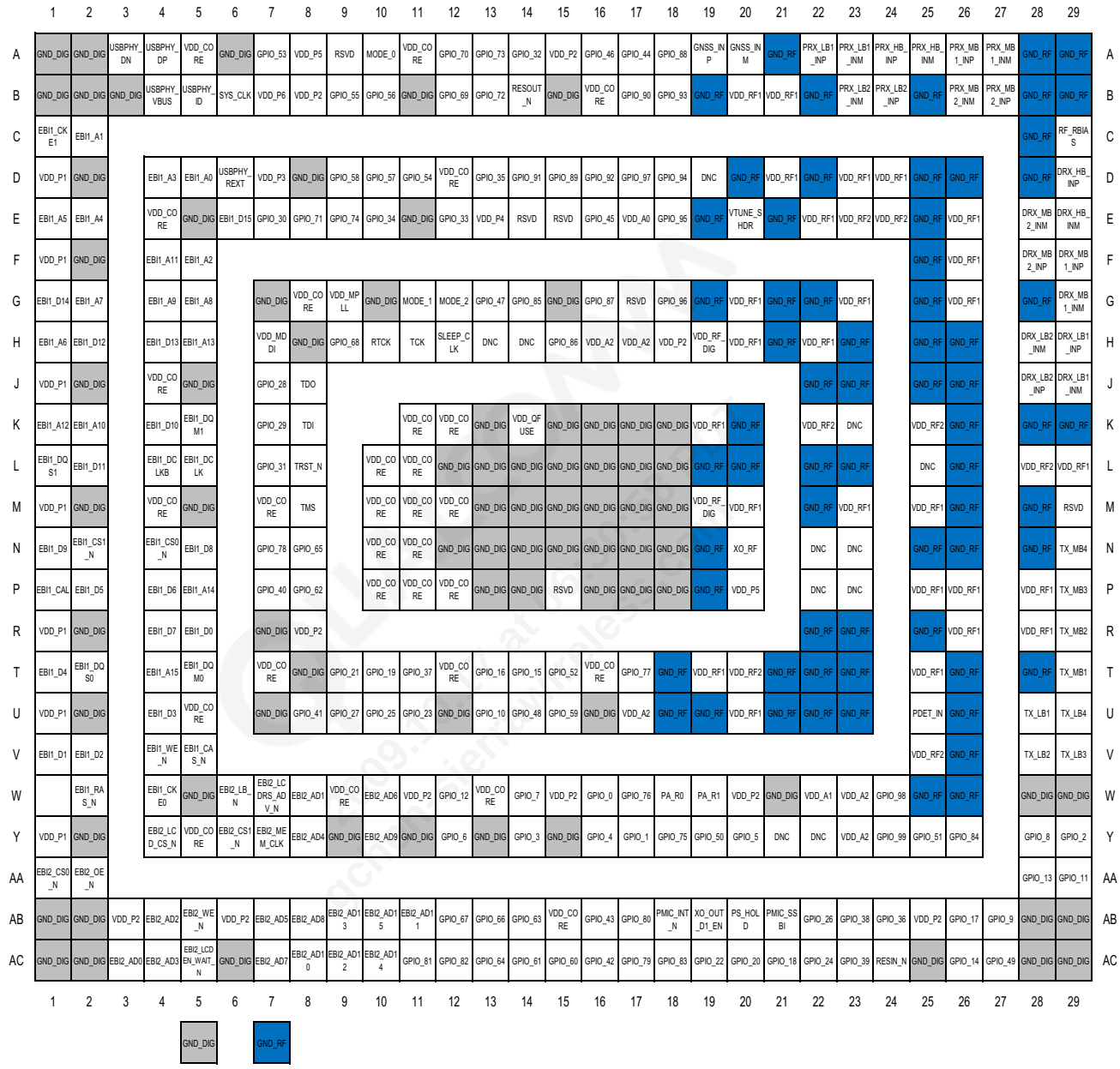


Figure 10-2 Analog and digital ground pins

All ground types must be connected as directly as possible to their PCB ground-fill areas, and ultimately to the primary inner-layer ground plane. Connect IC grounds to as much metal as possible using thick, wide fill areas and sub-planes in addition to the inner-layer ground plane. This proper grounding technique also reduces electromagnetic interference (EMI).