Application Note

AWR2188 Boot Process

Roy Zhuang *Central FAE*
Allen Yin

ABSTRACT

This document was translated from a simplified Chinese source. (ZHCAG83)

MmWave imaging radar is a core technology in fields such as Advanced Driver Assistance Systems (ADAS), autonomous driving, and industrial automation. Compared with traditional radar, imaging radar utilizes multi-channel antenna arrays to achieve higher angular resolution and can generate high-precision point clouds, thereby more accurately identifying and classifying objects. The high-performance single-chip mmWave radar AWR2188 introduced by Texas Instruments (TI) operates in the 76–81GHz frequency band, integrates 8 transmit and 8 receive channels, and has a built-in complete radio frequency (RF) front end, phase-locked loop (PLL), and analog-to-digital converter (ADC), achieving high integration and compact packaging. This article will introduce the boot process of the AWR2188 chip in detail. We will walk through the process from hardware power-on to firmware loading, aiming to provide developers with a clear and complete operation guide, helping them quickly get started with AWR2188 system development, and accelerating the deployment of imaging radar applications.

Table of Contents

List of Figures

List of Tables

1 Introduction to AWR2188

The imaging radar market is in a rapid evolution stage driven by high-performance demands, especially in the fields of Advanced Driver Assistance Systems (ADAS), autonomous driving, and industrial applications. In the early days, to overcome the limitations of traditional radar such as low resolution and the inability to form fine point clouds, the industry began to explore imaging radar solutions. Taking Texas Instruments' (TI) second-generation AWR2243 mmWave sensor as an example, it requires a multi-chip cascade method to increase the number of antennas, thereby obtaining more virtual channels to meet higher-performance imaging needs, such as longer distance and higher resolution. This cascade solution achieved a breakthrough in performance, but at the same time, it brought challenges such as increased system complexity, rising power consumption, and longer development cycles.

The driving force of the market promotes technology to develop towards higher integration and better cost-effectiveness, so as to achieve the popularization of imaging radar. Under this background, TI launched a new generation of AWR2188 mmWave radar chip, representing the development direction of radar technology towards single-chip high-performance integration. The AWR2188 integrates up to 8 transmitters and 8 receivers. On a single chip, it outperforms the two-chip AWR2243 cascade solution in terms of channel count, performance, and power consumption, thereby significantly simplifying hardware design and reducing system cost and power consumption. This fundamental architectural innovation makes the development of high-performance imaging radar more efficient and convenient, accelerating its deployment in various application scenarios.

The advent of AWR2188 not only marks a leap in the integration level of imaging radar technology, but also provides more efficient and reliable perception solutions for autonomous driving and industrial applications. Its high-channel-count design can provide finer point cloud data, improving the ability to recognize and distinguish small objects and complex scenes in the environment. In addition, the advantages brought by single-chip integration are also reflected in smaller size, lower system-level power consumption, and higher production efficiency, which are particularly critical for space-constrained automotive electronics and cost-sensitive industrial applications. Therefore, the emergence of AWR2188 is not only a technological iteration, but also a precise response to market demands, paving the way for the widespread application of the next generation of high-performance forward-looking imaging radars.

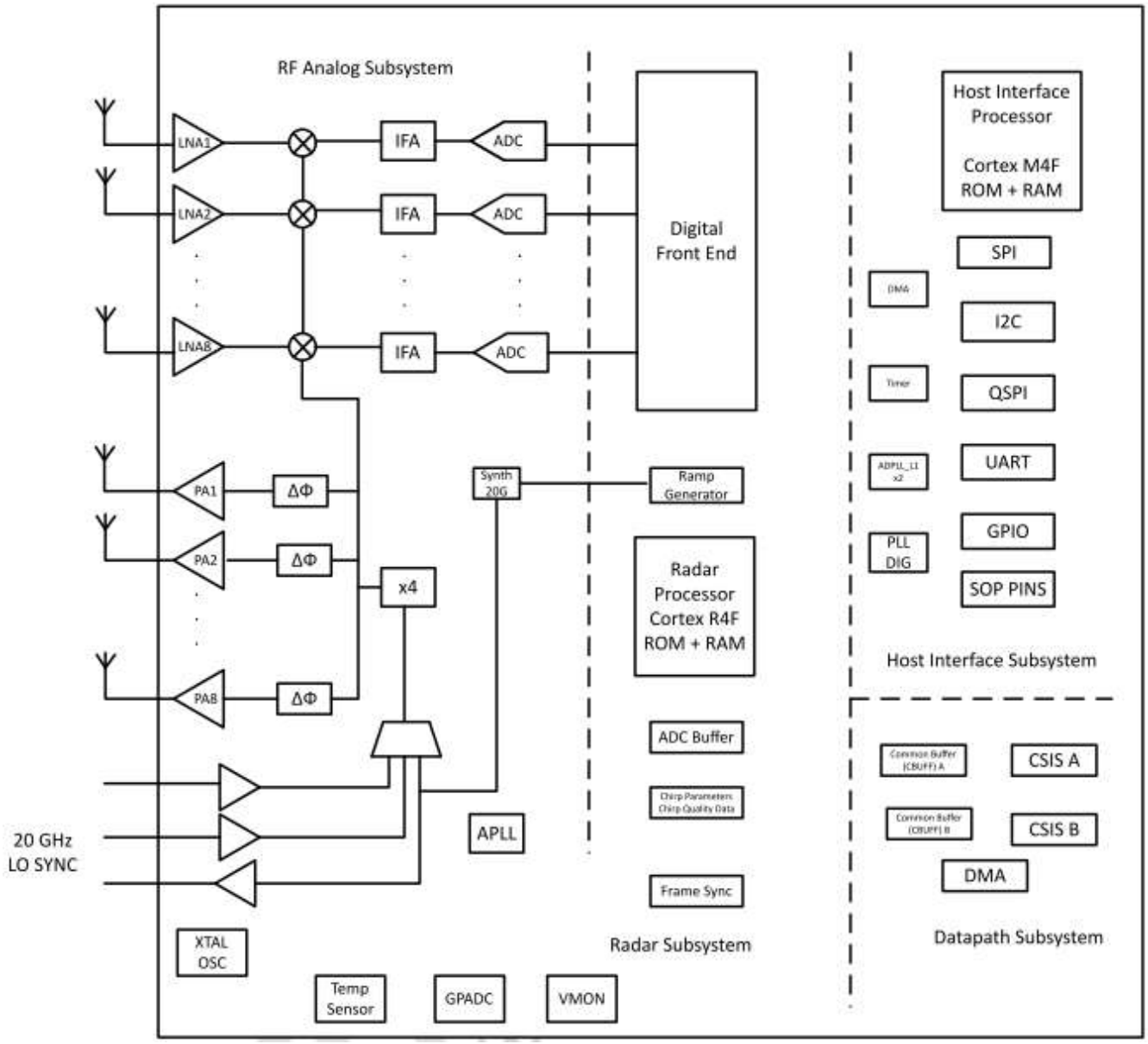

Figure 1-1. AWR2188 Chip System Block Diagram

The internal structure of AWR2188 mainly consists of the following parts:

- mmWave RF Module: Mainly includes the RF transceiver and sampling module, containing the receive low noise amplifier (LNA), transmit power amplifier (PA), mixer, analog-to-digital conversion (ADC), local oscillator generator (Synthesizer), and phase-locked loop (APLL), etc.;
- Radar Front-End Controller: The firmware of the RF controller runs on the built-in Cortex R4F (RSS, Radar Subsystem) processor, used to control the parameter configuration, calibration, and monitoring of the RF front end, and perform decimation filtering on the sampled signals;
- Host Interface Control Module: The built-in Cortex M4F (MSS) runs the firmware provided by TI, used to provide SPI or I2C interfaces to communicate with external processors, and send ADC data over the CSI interface via DMA, as well as interact with the radar front-end controller;

This article will introduce the hardware configuration and boot process of AWR2188, aiming to help customers quickly get started with and debug AWR2188 to the greatest extent.

2 Hardware Architecture and Design

2.1 Radar System Architecture

Currently, there are two mainstream mmWave radar system architectures on the market: one is the satellite radar architecture. The mmWave front-end sensor chip MMIC directly transmits the collected raw data to the vehicle's central domain controller processor through a high-speed interface (Serdes/Ethernet), and the raw data processing is performed in the central domain controller. The other is the edge radar architecture. That is, the MMIC and the processor are integrated on one board or the same chip. The MMIC hands over the collected raw data to the processor to be processed into point clouds or target data, and then transmits it to the vehicle's central domain controller processor through a CAN-FD or Ethernet interface for the next step of decision-making. Figure 2-1 and Figure 2-2 are the hardware block diagrams of these two architectures.

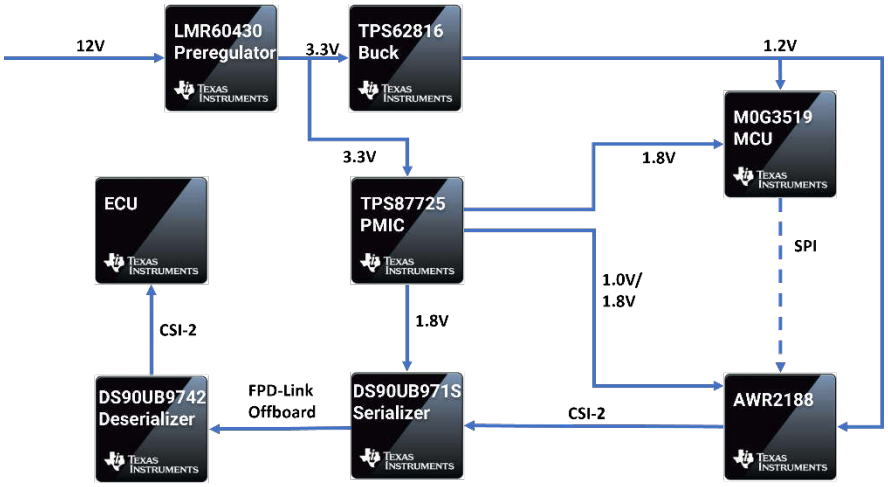

Figure 2-1. Satellite Radar Architecture

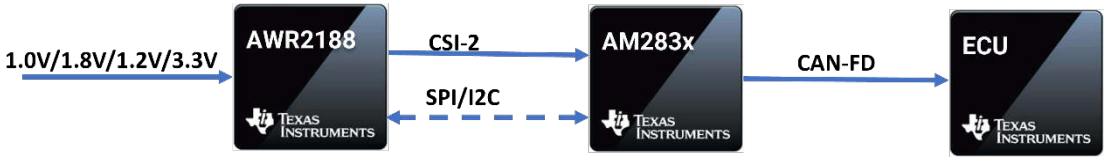

Figure 2-2. Edge Radar Architecture

2.2 AWR2188 Radar Hardware Design

Below, taking the satellite architecture of AWR2188 as an example, its hardware design is introduced. Figure 2-3 draws all the signal pins and power-related devices required by AWR2188 (among them, TPS628503, TPS628501, TLV70723, LMR60430 are buck converters, M0G3519 is an MCU external micro-controller, LP87745 is a PMIC power management chip, DS90B971 is an FPDLINK serializer). Among them, the pins that must be connected to connect AWR2188 are: nRESET, nError, SPI_BUSY, HOST_IRQ, SPI_MOSI, SPI_MISO, and SPI_CS (if it is I2C mode, see Section 4.2 I2C Configuration for details).

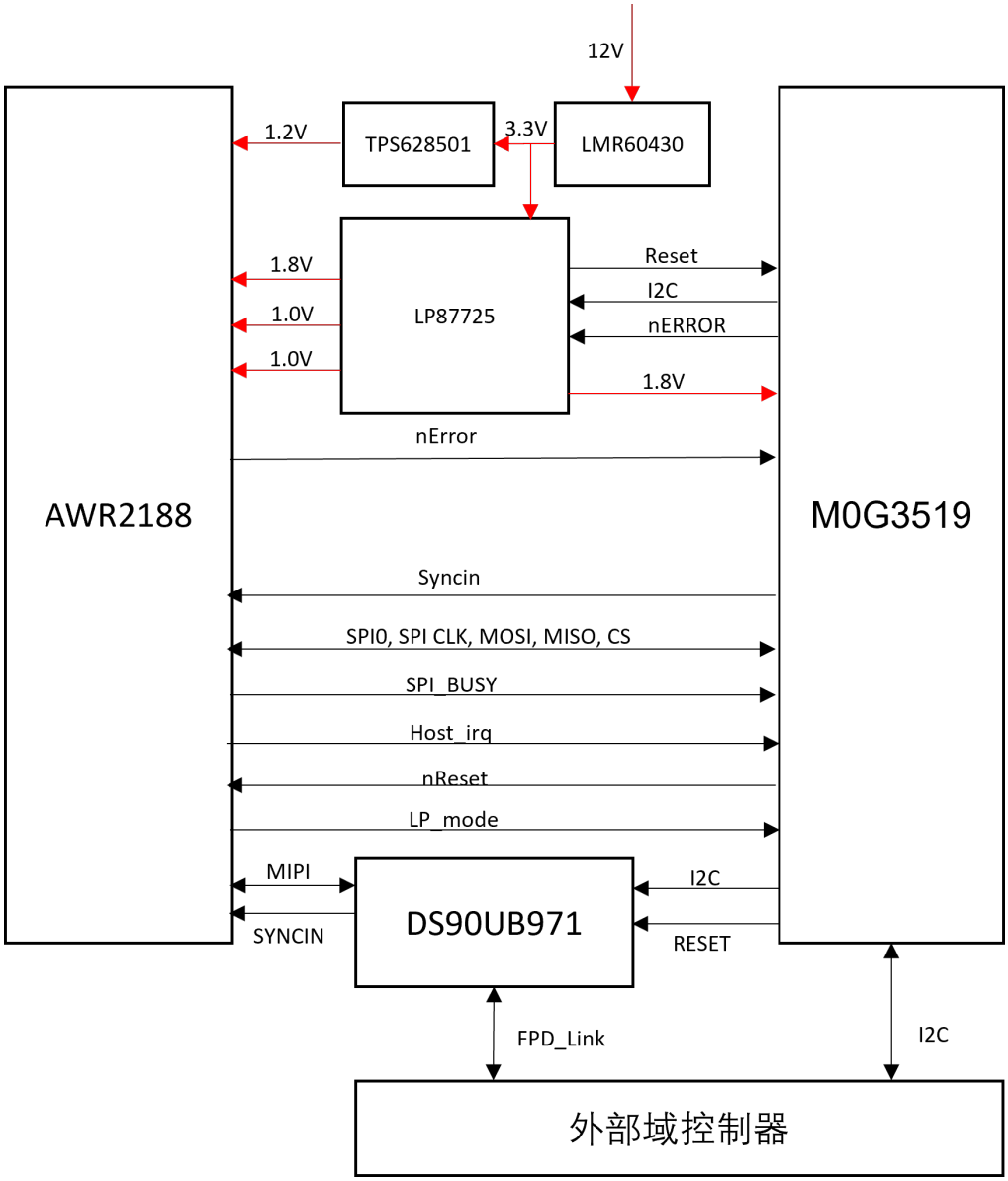

Figure 2-3. Satellite Radar Architecture Hardware Design

For more detailed schematics and reference designs, please refer to AWR2188 Satellite Radar Hardware Design [4] .

2.3 SOP Pin Configuration

The first operation of AWR2188 after power-on is to read the state of the Sense-on-Power (SOP) pins. Table 2-1 are the SOP modes and corresponding pins for AWR2188.

Table 2-1. Sense-on-Power (SOP) Lines and Boot Modes

Functions	Signal	Description	Pin Number
SOP – SOP[4:0]	SOP0 (Multiplexed with TDO)	SOP 0 - controls boot behavior	AF8
	SOP1 (Multiplexed with FRM_SYNCOUT)	SOP 1 - controls boot behavior	AF3
	SOP2 (Multiplexed with PMIC_CLKOUT)	SOP 2 - controls boot behavior	AG5
	SOP3 (Multiplexed with MCU_CLKOUT)	SOP 3 - controls cascade/non-cascade	AG6
	SOP4 (Multiplexed with QSPI_0)	SOP 4 - controls 40 MHz or 50 MHz crystal	AG27

SOP Mode	Description	SOP[2:0]
Functional Mode SPI	Main deployment mode. Firmware patches are loaded from an external processor via SPI or from Flash via QSPI, functional firmware execution begins and device is controlled via SPI commands	0 0 1
Device Management Mode	Flash programming mode. Image files (patches) are downloaded to a serial flash (SFLASH) via a burn tool that transfers the image files over UART	1 0 1
Functional Mode I2C	Load firmware patches via I2C, and then functional firmware begins execution	1 1 1

SOP Mode	Description	SOP[3]
Cascade Mode	0: Single-chip AWR2188 1: Multi-chip cascade mode	1 or 0

SOP Mode	Description	SOP[4]
Crystal Selection	0: 40MHz 1: 50MHz	1 or 0

2.4 AWR2188 Reset and RBL Boot Process

After AWR2188 powers on, during the process where the nReset signal changes from low to high, the SOP hardware configuration will be recorded. After the reset ends, it will enter the RBL (ROM Bootloader, executing the Bootloader from the TI solidified ROM area after power-on) stage.

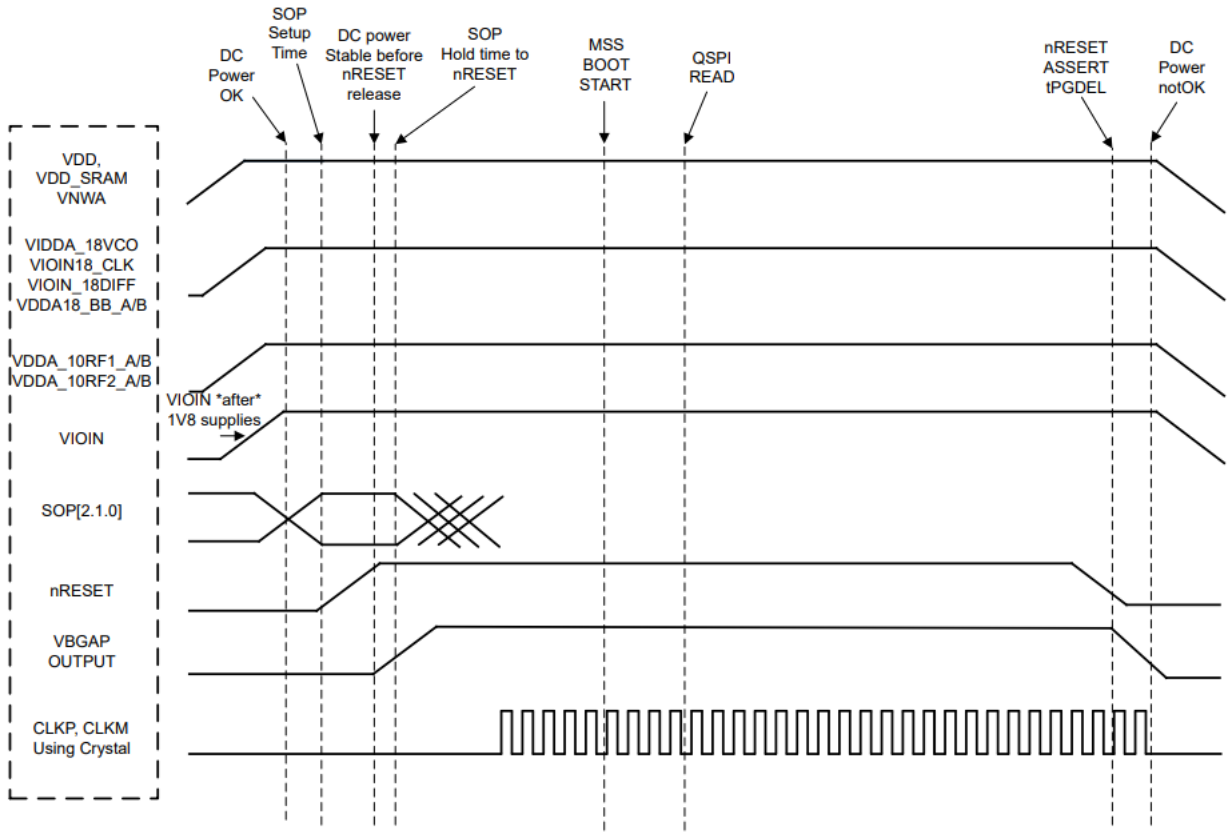

Figure 2-4. AWR2188 Power-on Timing

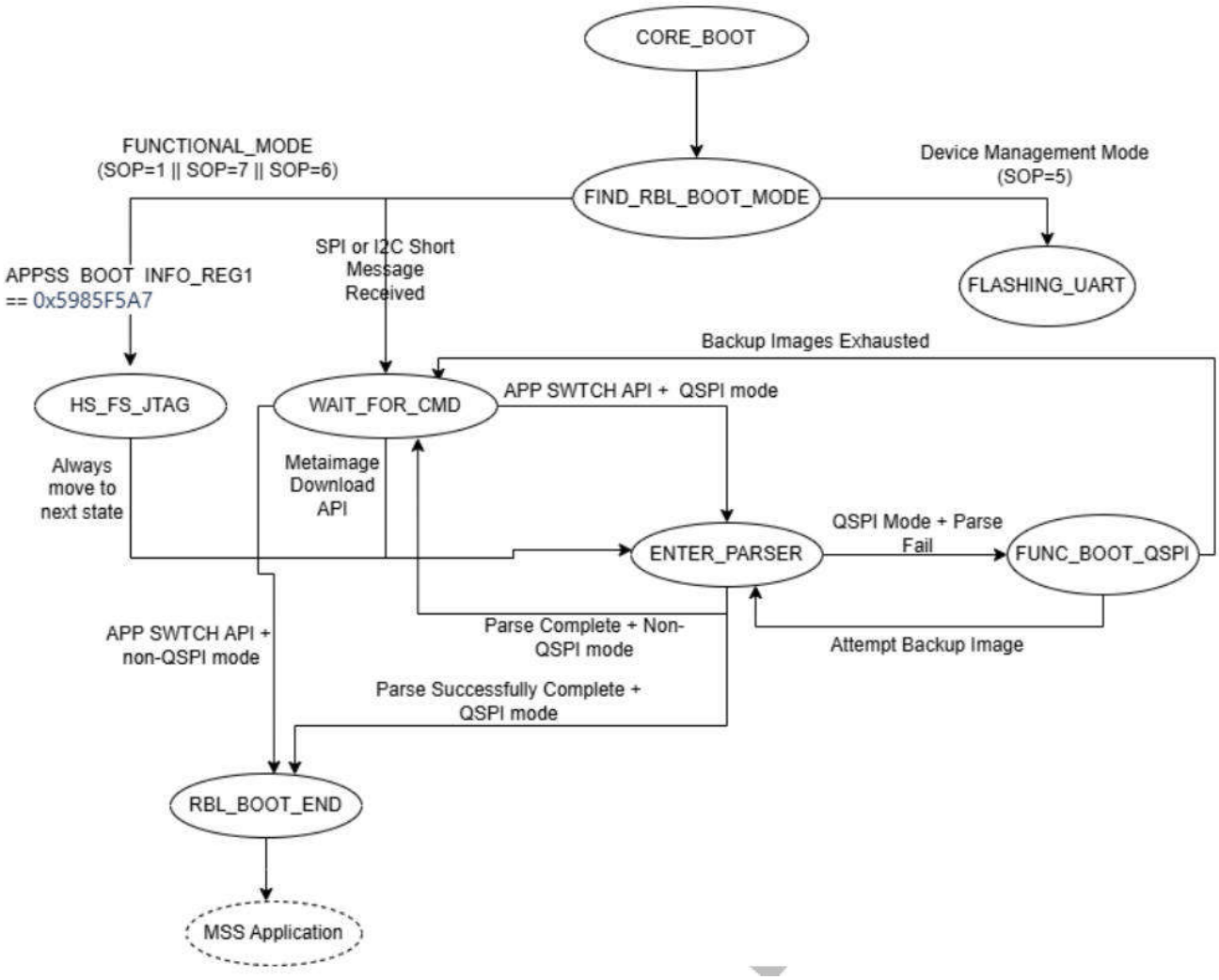

Figure 2-5. AWR2188 Power-on Timing

The flowchart of the entire RBL stage is shown in Figure 2-5. In the SPI/I2C functional mode, after the nReset of AWR2188 goes from low to high, AWR2188 enters the RBL stage. The RBL will choose to use QSPI from Flash or use SPI/I2C from an external processor to download the AWR2188 firmware according to the user's configuration. After the firmware download is complete, the AWR2188 external processor will set whether the application program boots from QSPI or boots from the external processor via SPI/I2C according to the software configuration.

3 AWR2188 External Processor Software Configuration

To help users better understand the boot process and configuration of AWR2188, this chapter takes M0 as an example of an external processor to introduce the software configuration of the external processor for AWR2188 in SPI (SOP = 001) mode. Figure 3-1 gives the software execution sequence after AWR2188 powers on and boots.

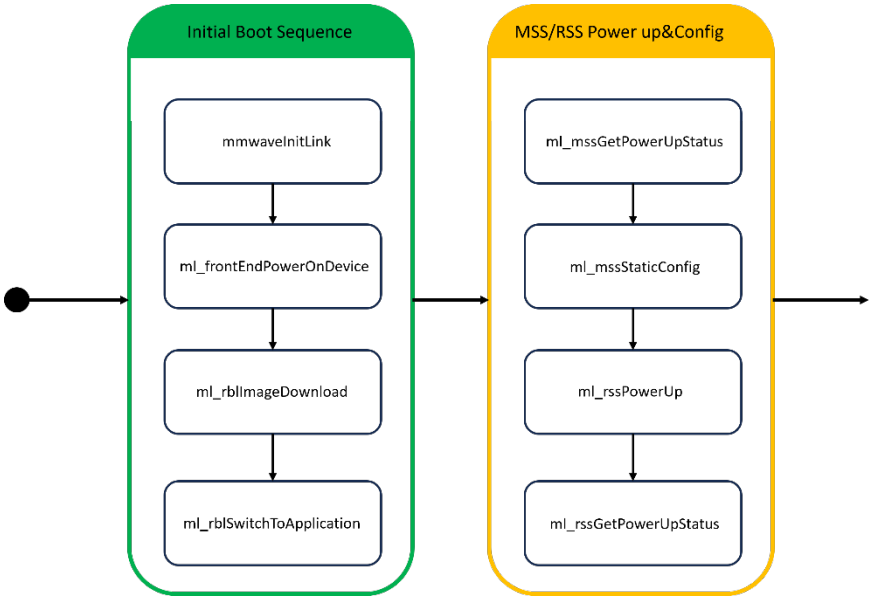

Figure 3-1. AWR2188 Software Execution Flow

The functions in the figure are executed in the external processor, controlling various functions in AWR2188 by calling the interface functions in the mmWaveLink framework. The firmware package and execution code running on AWR2188 are all contained in the DFP (Device Firmware Package, which contains firmware, and code required to be executed by the RSS and MSS systems).

3.1 External Processor Powers on and Configures MSS

On the software side, after AWR2188 powers on, the first operation of the external processor software is to reset AWR2188 through the *ml_frontEndPowerOnDevice* function (controlling the nReset pin level of AWR2188 from high to low and then to high), and during the reset period, by controlling the GPIO output level, configure the SOP and clock of AWR2188, where the SOP configuration is seen in Table 3-1, and the Clock configuration is seen in Table 2-1. After this function finishes execution, AWR2188 leaves the reset state and formally enters the RBL stage, executing the code solidified on the ROM side, performing initializations such as clock on AWR2188, and waiting to download the DFP image from the external processor.

```
    /*Power up*/
    pwrup.c_PowerControl = M_T_ML_FE_DEVICE_POWER_UP;·  //上电宏
    pwrup.c_SopBootCfg = M_ML_DEVICE_SOP_BOOT_SPI_MODE_0;  //SPI 模式
    pwrup.c_SopClkCfg =M_ML_DEVICE_SOP_CLOCK_50MHZ_OSC_OUT_DISABLE;  //时钟配置

status = ml_frontEndPowerOnDevice(DEVICE_INDEX, &pwrup);
```

Table 3-1. External Processor Software SOP Configuration

Configuration	Description
0x01	Functional Mode using SPI0
0x06	Functional Mode using SPI1
0x07	Functional Mode using I2C
0x05	Device Management Mode using UART
0x03	Debug Mode

3.2 DFP Image Download

AWR2188 will judge whether it is QSPI mode or SPI/I2C mode based on SOP during the RBL stage. If it is QSPI mode, AWR2188 copies the DFP image from Flash to AWR2188's RAM; if it is SPI/I2C mode, the external processor needs to transmit the DFP image to AWR2188 via SPI or I2C. Below, taking the SPI mode as an example, the DFP image download is introduced.

```
    /* Offset in the M0 flash where the metaimage for the 2188 is stored*/
    imagecfg.w_ImageSrcAddress = 0x6F80;
    /* Size of the metaimage*/
    imagecfg.w_ImageSize = 0x32F48;
    status = ml_rblImageDownload(DEVICE_INDEX,&imagecfg);
    while(status!=0);
```

The external processor will download the DFP image into AWR2188 through this function ml_rblImageDownload. There are two points to note when using this function to download the image:

1. The address of the image in the external processor: the location of the DFP image on the Flash or external processor side;
2. The size of the image. For example, the image size is 208712 Bytes, and the sample code uses hexadecimal to represent the size, which is 0x32F48.

Generally speaking, the downloaded firmware is the DFP firmware package provided by TI, and the local storage path is: "C:\ti\mmWave_DFP_04_00_xx_xx\firmware\xWR2188\metaimage\mmwave_dfp_metaimage.bin".

After downloading the image, AWR2188 will return a verification value, saved in the structure T_ML_MSS_APPLICATION_ENTRY_RSP.w_ParseStatus. Users can judge whether the image download is normal based on this return value. If the image is downloaded normally to AWR2188, this verification value is 0x3E1A95C5.

Since the image download and subsequent control are all completed through the SPI/I2C protocol, the SPI/I2C protocol analysis of AWR2188 is placed in Chapter 4 for separate discussion.

3.3 MSS Application Boot and Configuration

The power-on of AWR2188's MSS is completed during the process of the external processor calling frontEndPowerOnDevice. When the image download is complete, AWR2188 will switch to the corresponding application boot mode through ml_rblSwitchToApplication. Currently, there are two application boot modes that can be configured, which are QSPI mode and serial data interface mode. The difference between the two lies in whether the MSS application program is loaded from Flash through the QSPI interface, or loaded by the external processor through the SPI/I2C protocol. After AWR2188 switches to the corresponding boot mode, it will trigger an asynchronous event to notify the external processor. Whether this asynchronous event is triggered normally can be confirmed by checking if the return value aehandler.h_MssAsyncStatus is 1.

```
/* Switch to Application*/
    entryptr.h_BootMode = 0x00;
    status = ml_rblSwitchToApplication(DEVICE_INDEX,&entryptr,&entryrspptr);
    /* Ensure device power up*/
    while((aehandler.h_MssAsyncStatus & 1) != 1);
```

Table 3-2. AWR2188 Application Boot Mode Configuration

Configuration	Description
0xAACE	Boot from QSPI
0x0000	Boot from SPI/I2C Interface

After switching to the corresponding application boot mode, AWR2188 will check the power-on status of MSS and configure the parameters of MSS. After MSS configuration is completed, RSS will also be configured in sequence according to the flow of powering on, checking power-on status, and configuring RSS. Among them, the key configurations required for MSS and RSS power-on are seen in Table 3-3, Table 3-4, and Table 3-5.

Table 3-3. MSS Power-On Configuration

Name	Description	Configuration
c_PowerControl	Set nReset to low level to turn off power	0
	Set nReset to high level to boot the device	1
c_SopBootCfg	See Table 3-1	See Table 3-1
c_SopClkCfg	40MHz XTAL, OCS_CLOCKOUT enabled	0
	40MHz XTAL, OCS_CLOCKOUT disabled	1
	50MHz XTAL, OCS_CLOCKOUT disabled	2
	50MHz XTAL, OCS_CLOCKOUT enabled	3

Table 3-4. RSS Power-On Configuration

Name	Description	Configuration
c_PowerOnMode	In cold boot, RSS will clear all data in all RAM, and firmware variables will also be initialized to default values. Generally, this mode applies to when RSS powers on for the first time or when RSS data cannot be saved after power-off	0x00
c_ClkSourceSel	RSS clock comes from external crystal source	0
	RSS clock comes from PLL-Dig	1
	RSS clock comes from Core-ADPLL clock	2
c_HostSubsystemSel	For an MMIC like AWR2188, RSS can only communicate with one subsystem at a time	0
c_RefClkIndex	50 MHz XTAL	0
	40 MHz XTAL	1
	50 MHz BAW	2
h_CoreClockFreq	The Core Clock of AWR2188 RSS must be 200MHz	200
w_RssPowerUpCfg	See Table 3-5	See Table 3-5

Table 3-5. w_RssPowerUpCfg Power-On Configuration

Bit	Description
Bit[0]	Disable Boot Monitor; 1 is Disable; 0 is Enable
Bit[1]	Disable RSS Logger; 1 is Disable; 0 is Enable
Bit[2]	Disable Configuration Error Check; 1 is Disable; 0 is Enable
Bit[3]	Disable RSS CRC; 1 is Disable; 0 is Enable
Bit[4]	Disable watch dog; 1 is Disable; 0 is Enable
Bit[5]	Disable RSS Core Clock Gating; 1 is Disable; 0 is Enable
Bit[6]	Disable RSS Clock Scaling; 1 is Disable; 0 is Enable
Bit[7]	Enable RSS Debug Log Async-Event; On AWR2188, this function defaults to 0
Bit[8]	Disable Chirp-RAM Memory Initialization; 1 is Disable; 0 is Enable

```
    status = ml_mssStaticConfig(DEVICE_INDEX,&mssstaticcfg);  /*mss Static config*/
    while(status!=0);
    rsspwrupconfig.c_PowerOnMode = M_ML_RSS_POWER_UP_MODE_COLD;
    rsspwrupconfig.c_ClkSourceSel = M_ML_RSS_CLK_SOURCE_DIG_PLL;
    rsspwrupconfig.c_HostSubsystemSel = M_ML_RSS_SUB_SYSTEM_INDEX_DEFAULT;
    rsspwrupconfig.c_RefClkIndex = M_ML_RSS_REF_CLK_INDEX_50MHZ_XTAL;
    rsspwrupconfig.h_CoreClockFreq = 200; //200 MHz is frequency at which the RSS operates
rsspwrupconfig.w_RssPowerUpCfg = M_ML_RSS_PU_CFG_DISABLE_BOOT_MON | M_ML_RSS_PU_CFG_DISABLE_LOGGER
|M_ML_RSS_PU_CFG_ENABLE_ERROR_CHECK| M_ML_RSS_PU_CFG_DISABLE_CRC| M_ML_RSS_PU_CFG_DISABLE_WDT |
M_ML_RSS_PU_CFG_DISABLE_RSS_CLK_SCALING;
    status = ml_rssPowerUp(DEVICE_INDEX,&rsspwrupconfig);      /*RSS power up*/
```

On the external processor side, the user only needs to set the parameters of MSS/RSS well and directly call *mssStaticConfig, rssPowerUp, rssStaticConfig* to send the configuration file to AWR2188. Table 3-3 and Table 3-4 list the parameters that must be configured for MSS and RSS. For more detailed configurations, please refer to each API interface description document of DFP.

4 AWR2188 SPI/I2C Configuration and Signal Analysis

Whether AWR2188 is downloading images, control commands, or command verification, it cannot do without the SPI or I2C protocol. Understanding this part is very helpful for user debugging.

4.1 SPI Boot Configuration and Timing

When AWR2188 SOP[2:0] is configured as 001, AWR2188 will communicate with the external processor through the SPI protocol. The hardware pins used by AWR2188 in SPI mode are shown in Table 4-1.

Table 4-1. SPI Pin Configuration

Signal Name	Description	Pin Number
MSS_GPIO_0	Used by SPI_BUSY, for communication synchronization control between SPI (I2C) and external processor	AF13
MSS_HOSTIRQ	Used by HOST_IRQ, used to trigger asynchronous events	AF17
MSS_McSPIA_CLK	SPI clock signal	AG22
MSS_McSPIA_CS0	SPI chip select signal	AF20
MSS_McSPIA_MISO	AWR2188 SPI output data line	AF22
MSS_McSPIA_MOSI	AWR2188 SPI input data line	AF23

The signals sent by the external processor to AWR2188 via SPI will be divided into two parts: CMD_HEADER and CMD_LONG_MSG. Among them, the data format of CMD_HEADER is fixed, and its main role is to tell AWR2188 the function, data size, CRC, and other information of CMD_LONG_MSG; please refer to Table 4-2 for details. CMD_LONG_MSG is the data or command required to be transmitted in this SPI command transmission. In addition to this, AWR2188 can also use the CMD_PATTERN (Pattern signal, MISO returns a verification signal to notify MOSI of successful communication) function. When the external processor sends commands, it can determine whether the SPI command is normally delivered based on the return value of AWR2188. The flow of command sending and command response is similar. When AWR2188 sends RESP_HEADER, RESP_LONG_MSG to the external processor, the external processor responds with RESP_PATTERN.

Table 4-2. SPI Frame Data Format

Signal Name	Bytes
HDR_CHKSUM (Header Checksum)	2
SPI_CMD_ID (SPI Command ID)	1
CMD_FLAGS (Command Flags)	1
RW_TMT (Read/Write Timeout)	1
AUX_INFO (Auxiliary Information)	1
MSG_SIZE (Message Size)	2
MSG_CRC (Message CRC)	4
SHORT_MSG (Short Message)	4
LONG_MSG (Long Message)	N

Table 4-2 What needs attention is SPI_CMD_ID; the represented meaning of this command can be directly read through SPI_CMD_ID. The meaning of each CMD is shown in Table 4-3.

Table 4-3. SPI_CMD_ID

Interface Function Name	SPI_CMD_ID
SPI error info GET command	0x00
SPI error info GET response	0x80
MSS control SET/GET SPI command	0x01
MSS control SET/GET SPI response	0x81
MSS memory write SPI command	0x02
MSS memory write SPI response	0x82

Table 4-3. SPI_CMD_ID (continued)

Interface Function Name	SPI_CMD_ID
MSS handling RSS control GET SPI command	0x04
MSS handling RSS control GET SPI response	0x84
MSS compute CRC command	0x05
MSS compute CRC response	0x85
MSS image download command	0x06
MSS image download response	0x86

The SPI data timing for the external processor sending to and receiving from AWR2188 is shown in Figure 4-1. Before sending CMD_HEADER, the external processor detects whether SPI_BUSY is at a low level. If it is low, send CMD_HEADER; otherwise, poll and wait until SPI_BUSY is low. After AWR2188 receives CMD_HEADER, it will pull high SPI_BUSY, at which time the external processor will poll and detect whether SPI_BUSY is high. If it is high, the external processor begins to send CMD_LONG_MSG. After AWR2188 receives CMD_LONG_MSG, it pulls low SPI_BUSY, and at this moment, the SPI transmits information once successfully. Whether downloading firmware to AWR2188 or the external processor configuring AWR2188, using the SPI protocol will follow this rule.

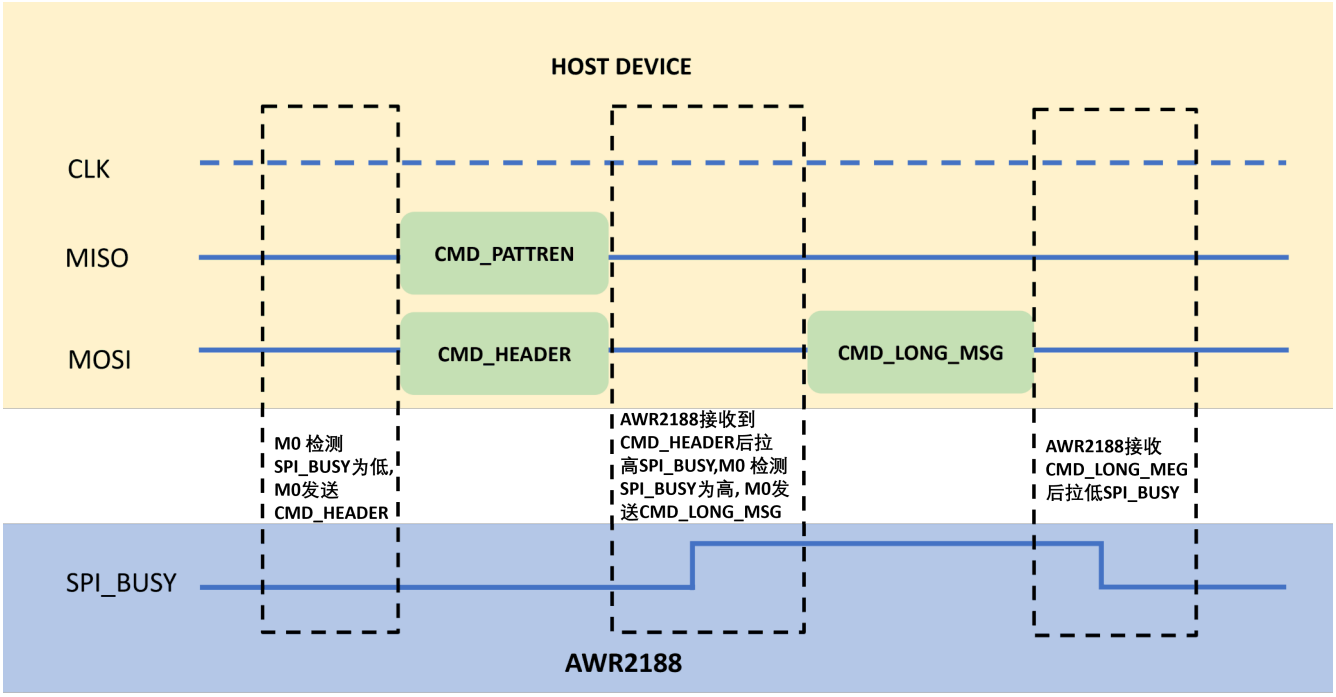

Figure 4-1. AWR2188 SPI Protocol Timing

Another point that needs attention is that if we use CMD_PATTERN/RESP_PATTERN, every time the sending end MOSI/MISO sends data, MISO/MOSI will receive the pre-defined CMD_PATTERN/RESP_PATTERN.

To facilitate users to better understand the previously mentioned CMD_HEADER, CMD_LONGMSG, and CMD_PATTERN, below takes the SPI command in the stage of M0 downloading firmware to AWR2188 as an example to introduce the SPI protocol:

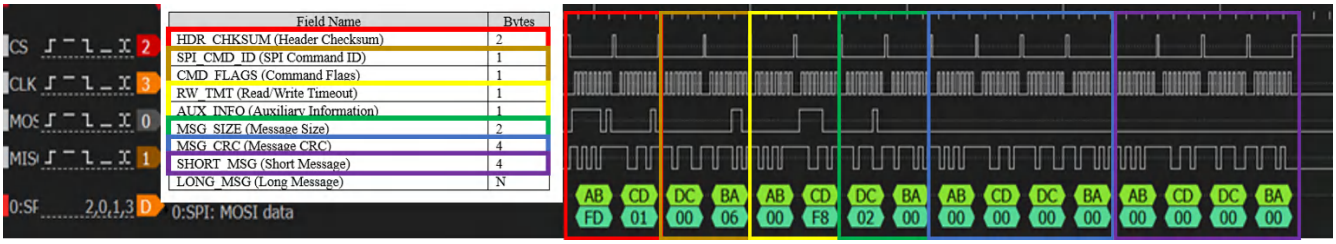

Figure 4-2. AWR2188 CMD_HEADER for Downloading DFP Firmware

AWR2188's SPI will receive and send two bytes each time, and the chip select signal (SPI_CS) can be pulled high once after every 1 byte or every 2 bytes. What needs attention is the external processor in the above example: M0 is a little-endian device, but M0's SPI module is a big-endian device, so every time data is sent, the data needs to undergo byte swapping. When using other big-endian or little-endian external processors, it only needs to ensure that the data on the SPI is in big-endian format.

The SPI of the external processor sends two bytes each time. SPI_CMD_ID and CMD_FLAG are sent at the same time and swap byte order, so the SPI_CMD_ID of the 3rd byte appears as the 4th byte on the SPI MOSI. Every time after CMD_HEADER finishes sending, the external processor will receive the CMD_PATTERN from AWR2188. This verification signal consists of 16 bytes of a fixed format:

```
const UINT8 c_CmdPattern[M_RL_PKT_HDR_SIZE] =
{
    0xCDU, 0xABU, 0xBAU, 0xDCU, 0xCDU, 0xABU, 0xBAU, 0xDCU,
    0xCDU, 0xABU, 0xBAU, 0xDCU, 0xCDU, 0xABU, 0xBAU, 0xDCU
};
```

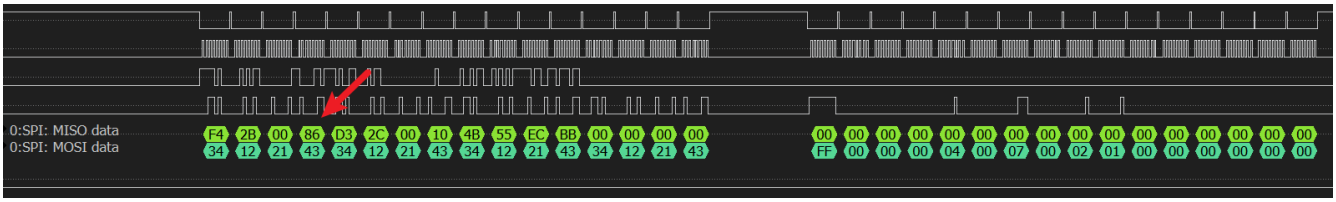

Figure 4-3. AWR2188 CMD_HEADER for Downloading DFP Firmware

By the same token, according to the same SPI command format as shown in Figure 4-3, the response command sent by AWR2188 to the external processor on MISO can be parsed. According to Table 4-2 and the 4th byte 0x86 of AWR2188's RESP_HEADER (MISO line), it can be determined that the image download is complete, and at this time the external processor sends RESP_PATTERN to AWR2188:

```
const UINT8 c_RespPattern[M_RL_PKT_HDR_SIZE] =
{
    0x34U, 0x12U, 0x21U, 0x43U, 0x34U, 0x12U, 0x21U, 0X43U,
    0x34U, 0x12U, 0x21U, 0x43U, 0x34U, 0x12U, 0x21U, 0X43U
};
```

In short, AWR2188 is controlled by the external processor through the SPI protocol, so all control commands can be parsed out using a logic analyzer combined with Table 4-2, thereby judging the current state of AWR2188.

```
const UINT8 c_RespPattern[M_RL_PKT_HDR_SIZE] =
{
    0x34U, 0x12U, 0x21U, 0x43U, 0x34U, 0x12U, 0x21U, 0X43U,
    0x34U, 0x12U, 0x21U, 0x43U, 0x34U, 0x12U, 0x21U, 0X43U
};
```

4.2 I2C Boot Configuration and Timing

When AWR2188 SOP[2:0] is configured as 111, AWR2188 will communicate with the external processor through the I2C protocol. The pins used on AWR2188 are shown in Table 4-4. In the I2C protocol, SPI_BUSY and HOST_IRQ are also both necessary, used to synchronize I2C signals and trigger asynchronous events.

Table 4-4. I2C Hardware Pins

SIGNAL_NAME	Description	APE PIN
MSS_GPIO_0	Used by SPI_BUSY, I2C/SPI control signal	AF13
MSS_HOSTIRQ	Used by HOST_IRQ, used to trigger asynchronous events	AF17
MSS_I2CA_SCL	I2C clock signal	AF22
MSS_I2CA_SDA	I2C data signal	AF23
MSS_GPIO_2	Used by I2C_ADDR_0 in SOP_I2C	AF6
MSS_GPIO_4	Used by I2C_ADDR_1 in SOP_I2C	AF20
MSS_GPIO_5	Used by I2C_ADDR_2 in SOP_I2C	AG22

The I2C address of AWR2188 is configured according to the following format. The last three bits of the I2C address can be customized by the user according to needs, ranging from 0101000 to 0101111.

AWR2188 I2C ADDRESS						
0	1	0	1	MSS_GPIO_5	MSS_GPIO_4	MSS_GPIO_2

The entire I2C protocol is consistent with the SPI protocol. SPI and I2C use exactly the same underlying drivers, so the I2C protocol of AWR2188 can be analyzed with reference to the SPI protocol. However, since AWR2188 has only one data line (I2C_SDA), the verification signal will be turned off in I2C mode, and the external processor will not receive the verification signal returned by AWR2188.

5 References

- AWR2188 Single Chip 8x8 Cascadable 76-to-81 GHz Transceiver in LOP package datasheet
- AWR2243 Bootloader Flow
- AWR2188 EVM Hardware Reference Design
- AWR2188 Satellite Radar Reference Design

6 Modification History

Version	Date	Author	Notes
0.1	Oct 24th 2025	Roy Zhuang / Allen Yin	First draft
0.2	Mar 15th 2026	Roy Zhuang / Allen Yin	Second draft
*	June 12th 2026	Roy Zhuang / Allen Yin	initial release

IMPORTANT NOTICE AND DISCLAIMER

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

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

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

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

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

Last updated 10/2025