APPLICATION NOTE

I²C/SPI Boot Loader

Ultra-Low Power Apollo MCU Family

A-MCUAP3-ANGA01EN v1.3

Legal Information and Disclaimers

AMBIQ MICRO INTENDS FOR THE CONTENT CONTAINED IN THE DOCUMENT TO BE ACCURATE AND RELIABLE. THIS CONTENT MAY, HOWEVER, CONTAIN TECHNICAL INACCURACIES, TYPOGRAPHICAL ERRORS OR OTHER MISTAKES. AMBIQ MICRO MAY MAKE CORRECTIONS OR OTHER CHANGES TO THIS CONTENT AT ANY TIME. AMBIQ MICRO AND ITS SUPPLIERS RESERVE THE RIGHT TO MAKE CORRECTIONS, MODIFICATIONS, ENHANCEMENTS, IMPROVEMENTS AND OTHER CHANGES TO ITS PRODUCTS, PROGRAMS AND SERVICES AT ANY TIME OR TO DISCONTINUE ANY PRODUCTS, PROGRAMS, OR SERVICES WITHOUT NOTICE.

THE CONTENT IN THIS DOCUMENT IS PROVIDED "AS IS". AMBIQ MICRO AND ITS RESPECTIVE SUPPLIERS MAKE NO REPRESENTATIONS ABOUT THE SUITABILITY OF THIS CONTENT FOR ANY PURPOSE AND DISCLAIM ALL WARRANTIES AND CONDITIONS WITH REGARD TO THIS CONTENT, INCLUDING BUT NOT LIMITED TO, ALL IMPLIED WARRANTIES AND CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHT.

AMBIQ MICRO DOES NOT WARRANT OR REPRESENT THAT ANY LICENSE, EITHER EXPRESS OR IMPLIED, IS GRANTED UNDER ANY PATENT RIGHT, COPYRIGHT, MASK WORK RIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT OF AMBIQ MICRO COVERING OR RELATING TO THIS CONTENT OR ANY COMBINATION, MACHINE, OR PROCESS TO WHICH THIS CONTENT RELATE OR WITH WHICH THIS CONTENT MAY BE USED.

USE OF THE INFORMATION IN THIS DOCUMENT MAY REQUIRE A LICENSE FROM A THIRD PARTY UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF THAT THIRD PARTY, OR A LICENSE FROM AMBIQ MICRO UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF AMBIQ MICRO.

INFORMATION IN THIS DOCUMENT IS PROVIDED SOLELY TO ENABLE SYSTEM AND SOFTWARE IMPLEMENTERS TO USE AMBIQ MICRO PRODUCTS. THERE ARE NO EXPRESS OR IMPLIED COPYRIGHT LICENSES GRANTED HEREUNDER TO DESIGN OR FABRICATE ANY INTEGRATED CIRCUITS OR INTEGRATED CIRCUITS BASED ON THE INFORMATION IN THIS DOCUMENT. AMBIQ MICRO RESERVES THE RIGHT TO MAKE CHANGES WITHOUT FURTHER NOTICE TO ANY PRODUCTS HEREIN. AMBIQ MICRO MAKES NO WARRANTY, REPRESENTATION OR GUARANTEE REGARDING THE SUITABILITY OF ITS PRODUCTS FOR ANY PARTICULAR PURPOSE, NOR DOES AMBIQ MICRO ASSUME ANY LIABILITY ARISING OUT OF THE APPLICATION OR USE OF ANY PRODUCT OR CIRCUIT, AND SPECIFICALLY DISCLAIMS ANY AND ALL LIABILITY, INCLUDING WITHOUT LIMITATION CONSEQUENTIAL OR INCIDENTAL DAMAGES. “TYPICAL” PARAMETERS WHICH MAY BE PROVIDED IN AMBIQ MICRO DATA SHEETS AND/OR SPECIFICATIONS CAN AND DO VARY IN DIFFERENT APPLICATIONS AND ACTUAL PERFORMANCE MAY VARY OVER TIME. ALL OPERATING PARAMETERS, INCLUDING “TYPICALS” MUST BE VALIDATED FOR EACH CUSTOMER APPLICATION BY CUSTOMER’S TECHNICAL EXPERTS. AMBIQ MICRO DOES NOT CONVEY ANY LICENSE UNDER NEITHER ITS PATENT RIGHTS NOR THE RIGHTS OF OTHERS. AMBIQ MICRO PRODUCTS ARE NOT DESIGNED, INTENDED, OR AUTHORIZED FOR USE AS COMPONENTS IN SYSTEMS INTENDED FOR SURGICAL IMPLANT INTO THE BODY, OR OTHER APPLICATIONS INTENDED TO SUPPORT OR SUSTAIN LIFE, OR FOR ANY OTHER APPLICATION IN WHICH THE FAILURE OF THE AMBIQ MICRO PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR. SHOULD BUYER PURCHASE OR USE AMBIQ MICRO PRODUCTS FOR ANY SUCH UNINTENDED OR UNAUTHORIZED APPLICATION, BUYER SHALL INDEMNIFY AND HOLD AMBIQ MICRO AND ITS OFFICERS, EMPLOYEES, SUBSIDIARIES, AFFILIATES, AND DISTRIBUTORS HARMLESS AGAINST ALL CLAIMS, COSTS, DAMAGES, AND EXPENSES, AND REASONABLE ATTORNEY FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PERSONAL INJURY OR DEATH ASSOCIATED WITH SUCH UNINTENDED OR UNAUTHORIZED USE, EVEN IF SUCH CLAIM ALLEGES THAT AMBIQ MICRO WAS NEGLIGENT REGARDING THE DESIGN OR MANUFACTURE OF THE PART.

Revision History

Revision	Date	Description
1.0	August 25, 2015	Initial release
1.1	October 26, 2015	Updated for runtime selection between I^2C and SPI bus modes.
1.2	April 5, 2016	Updated to change run selection of I^2C versus SPI bus modes back to a complete option and other minor clean-ups.
1.3	April 12, 2022	Updated document template.

Reference Documents

Document ID	Description

Table of Contents

List of Tables

List of Figures

SECTION

1

Introduction

This document describes the protocol used on a SPI or I^2C bus that connects the Apollo (sensor hub), as a bus slave, to an Application Processor (host), acting as the bus master when used with an OTA boot loader. Apollo is supported by both a secure and a non-secure bootloader supporting the Apollo I/O Slave. This document describes the protocol with respect to the ios_-boot example in the **apollo_evk_base/examples** directory. This is the non-secure bootloader. Everything described here also applies to the secure bootloader.

The **ios_boot bootloader** can run in either an I^2C mode or SPI mode. The choice is made at compile time. The initial description of the boot loader protocol will focus on the SPI protocol. The I^2C protocol will be discussed later in t the document.

The SPI interface to the slave hardware in the Apollo MCU gives the host access to 128 registers that are shared between the host and the sensor hub. The host writes commands in to the shared registers and can read responses from the shared registers.

The first 120 bytes of the shared register space are simply shared storage while the final 8 registers implement an interrupt controller for the host's use. This interrupt controller can enable from 0 to 8 interrupt status bits to drive GPIO pin 4 high when an interrupt is asserted. Six (6) of the interrupt status bits are purely software interrupts which are set by the Apollo MCU to assert a GPIO interrupt. The interrupt enable register can on only be set or cleared from the host. Similarly, the interrupt status bits are cleared by SPI transactions from the host's SPI master, see Figure 1-1 Host View of 128 Byte Shared Register Space, on page 8.

The host can write to one or more of the shared register bytes by using a SPI write transaction such as the one sketched in Figure 1-2 SPI Write Transaction from Host to Shared Registers, on page 8. Notice that the first byte transmitted on the MOSI pin after the Chip Select has gone low is an addressing byte (offset byte) that contains both the direction for the rest of the transaction, (read or write), and a 7 bit offset value. The offset value tells the SPI slave on Apollo which byte is to be over-written by the first write data byte following the offset byte. A write transaction can write from one to 128 bytes. For the purposes of this protocol, the offset value used to actually write bytes to shared registers is automatically incremented after each byte is

written. Thus one can write an entire 48 byte command to the Apollo sensor hub in a single SPI transaction.

> **NOTE:** The chip select line must remain low between the offset byte and all subsequent data bytes in the same transaction.

Figure 1-1: Host View of 128 Byte Shared Register Space

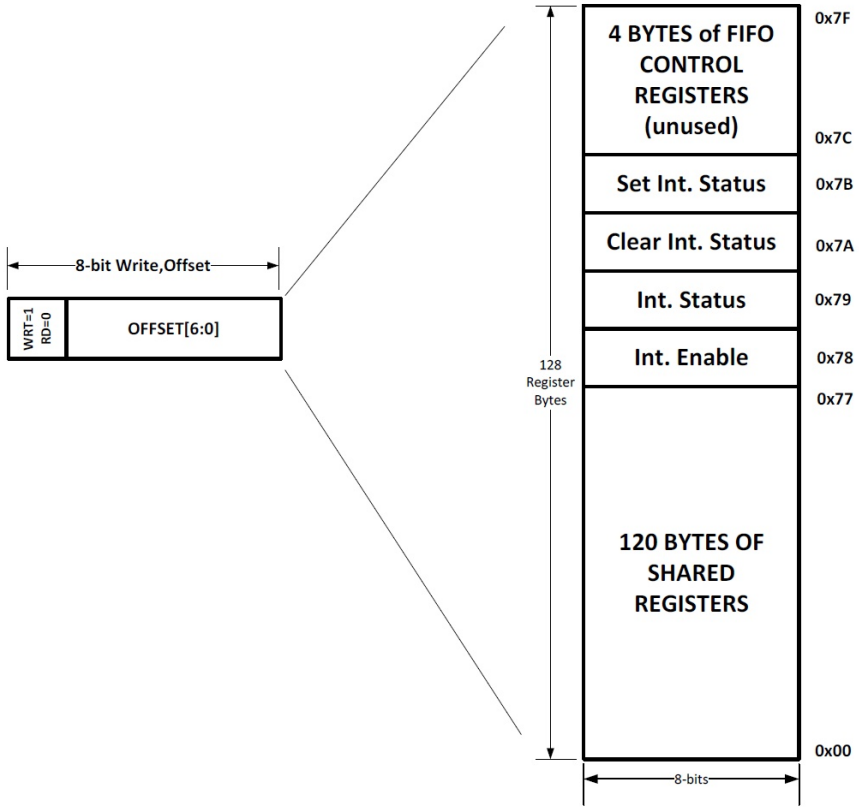
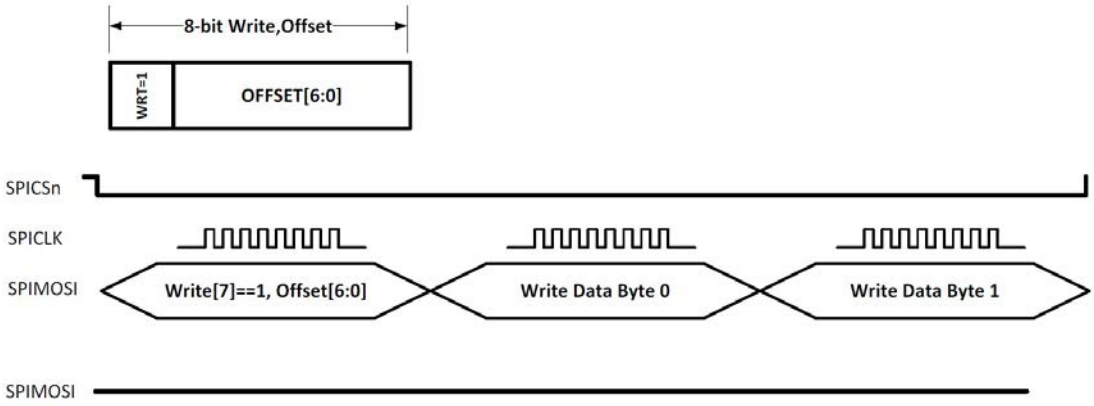


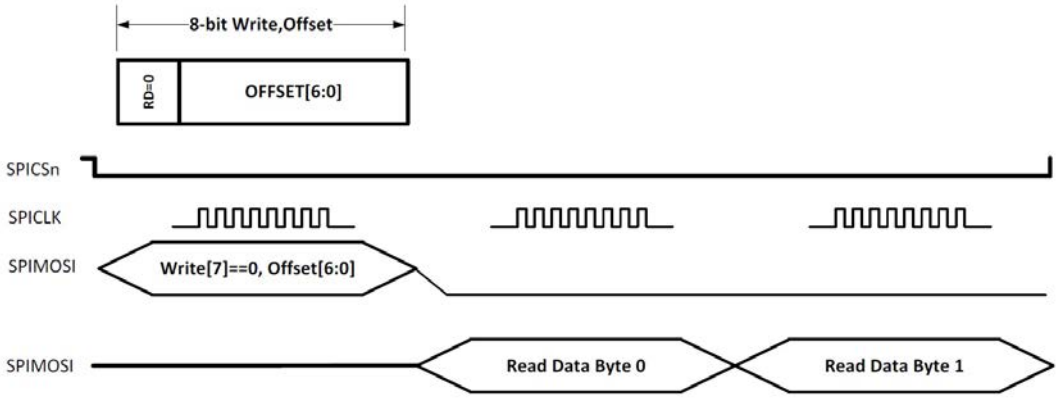
Figure 1-2: SPI Write Transaction from Host to Shared Registers



NOTE: Every time the chip select pin goes from high to low then the state machine in the Apollo slave assumes the next byte written over the SPI bus MOSI pin will be an offset byte.

In a similar way, a read transaction allows for from 1 to 128 bytes to be read from the Apollo I/O slave in a single SPI bus transaction as shown in Figure 1-3.

Figure 1-3: SPI Read Transaction



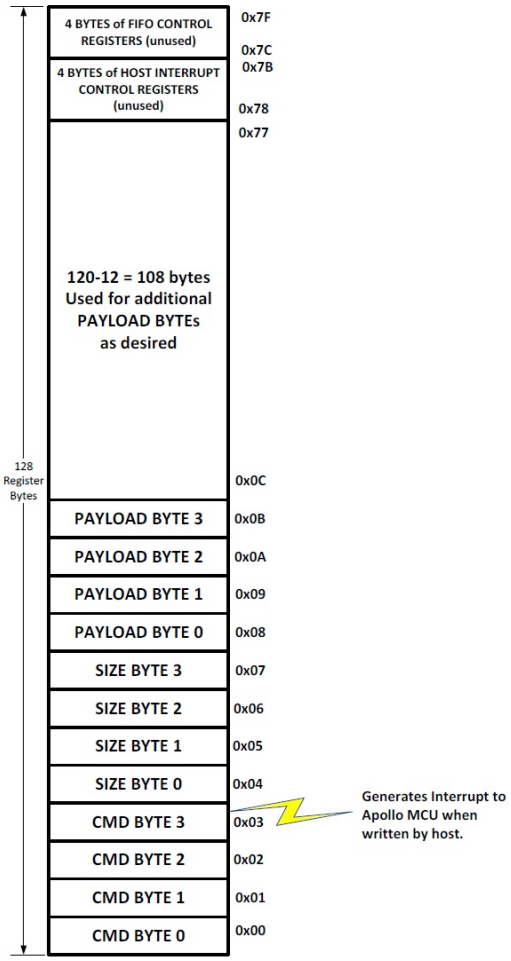
SECTION

2

OTA Boot Loader Protocol

The general layout of the shared register space is the same for all boot loader commands (and in either I^2C or SPI modes) and essentially looks like that shown in Figure 2-1.

Figure 2-1: Shared Register View of Protocol



The first thing to notice about the bootloader protocol is that all data items are multiples of 4 bytes and are always aligned on 4 byte boundaries within the shared register space.

The second thing to notice is that the Apollo firmware has initialized the I/O slave hardware so that an access interrupt is generated to the Apollo MCU whenever the host writes to byte 3 of the shared register space. This should always be the last byte written in any transaction, otherwise the Apollo MCU may way wake up too quickly and grab stale data from the shared registers before the host has finished writing them.

Thus one should always break the command write operations up in to two transactions:

1. Write all of the bytes of the payload in one transaction.
2. Write the 4 bytes of the command to shared registers 0 through 3 as the very last transaction.

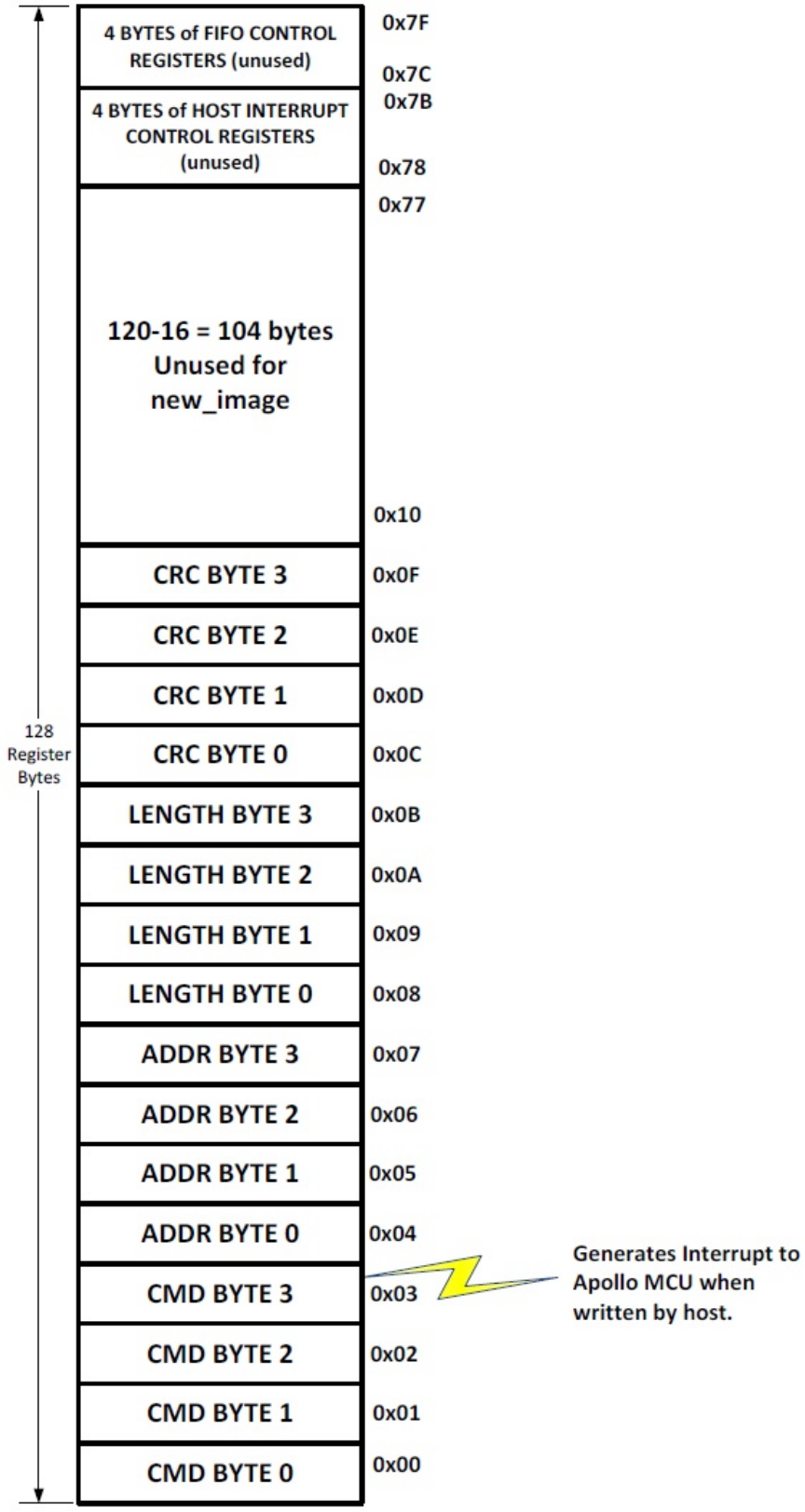
This will trigger the Apollo to wake up and process a command from the host.

The layout of the shared register space for a NEW IMAGE command is shown in Figure 2-2 New Image Command As Seen in the Shared Registers, on page 12. Whenever a new binary image is to be downloaded to Apollo's integrated flash memory, one must send the NEW IMAGE command telling the bootloader where to place the image and how big the image will be in flash memory. In addition an expected CRC value for the new image is provided with this command. When the entire image has been downloaded and stored in flash, the boot loader will re-compute the CRC and make sure that it matches the one provided with this command.

Finally, the NEW IMAGE command can tell the boot loader which pin to monitor at reset to enter the boot loading state (boot loader override) instead of launching the pre-installed program. The NEW IMAGE command can also tell the boot loader which polarity to use for detecting the boot loader override function.

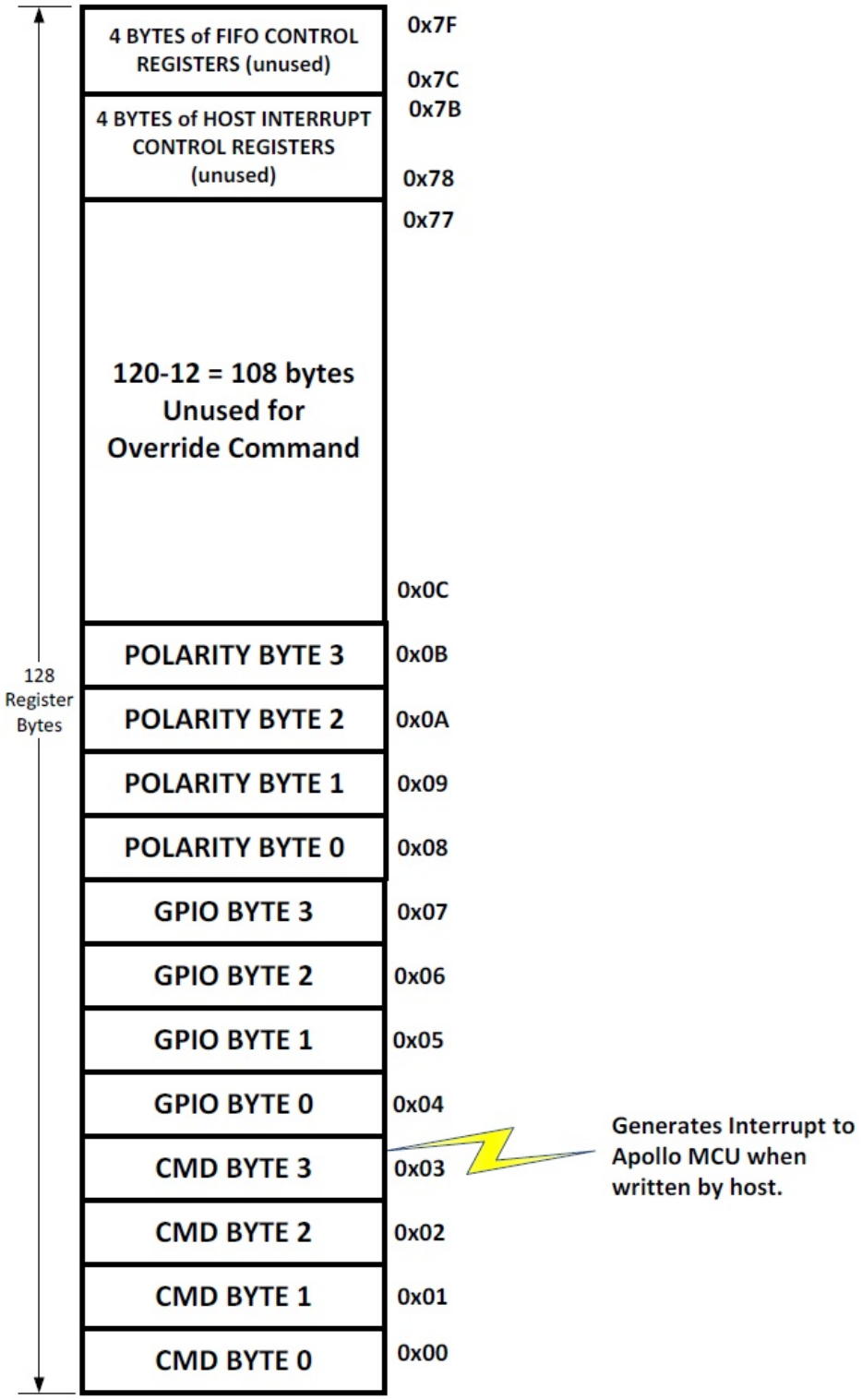
It cannot be emphasize enough that byte 3 MUST be the last byte written for any boot loader command sent to the Apollo MCU.

Figure 2-2: New Image Command As Seen in the Shared Registers



The override command allows the host to tell the boot loader which GPIO pin to monitor for forcing a boot load operation even when a valid program is present.

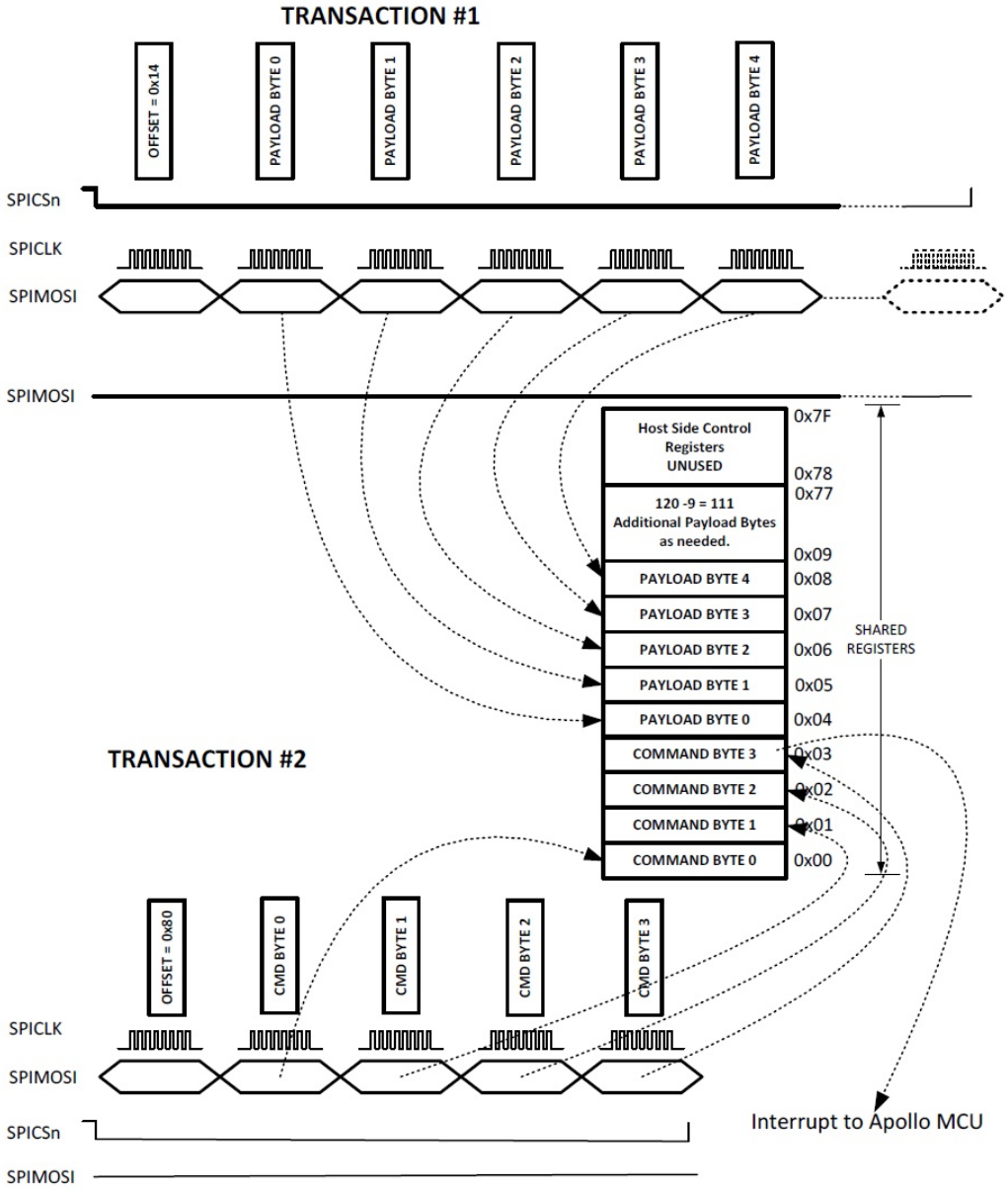
Figure 2-3: New Image Command As Seen in the Shared Registers



2.1 Sending Commands to the Apollo Sensor Hub

When a command is to be sent to the boot loader running on the Apollo, it must sent according to the diagram of Figure 2-4 as seen in the shared registers.

Figure 2-4: Sending Command from Host to Sensor Hub



The host initiates a command transfer by using a SPI write transaction to send the payload size and the payload bytes to a message buffer located in the Apollo I/O slave shared register space beginning at offset 0x04 (4).

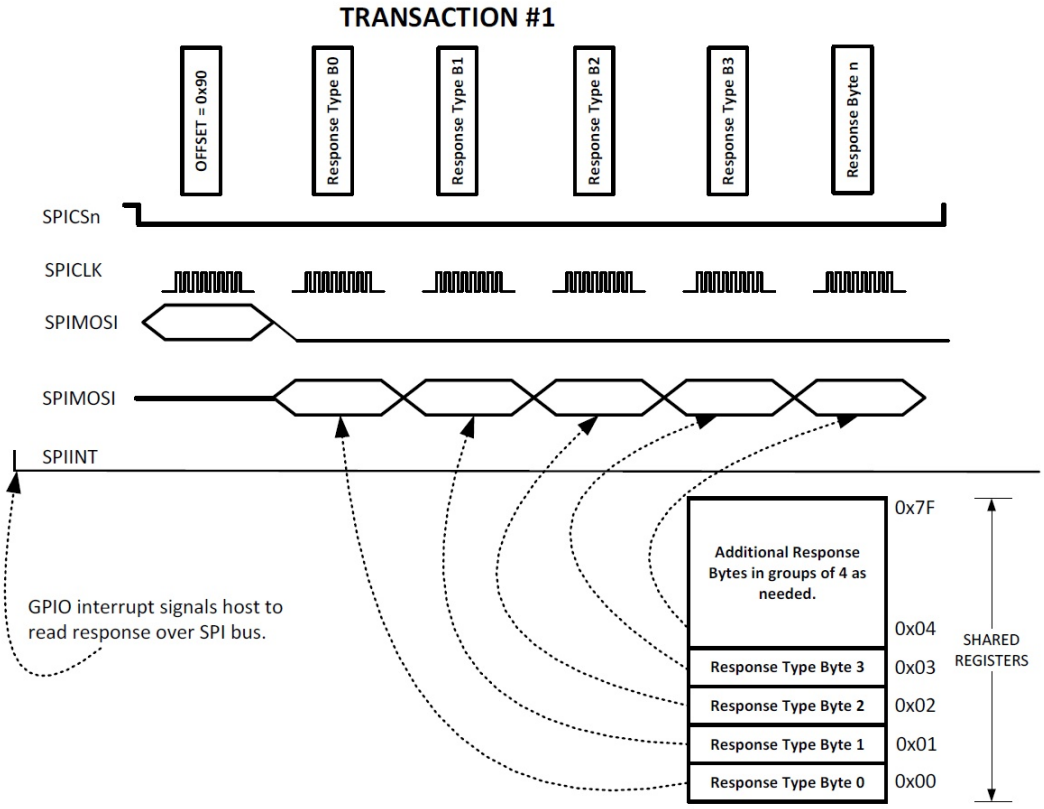
The host then uses a second SPI write transaction to write four bytes containing the command to offset 0x00 (0) in the shared register space. During initialization,

the Apollo firmware has set a special mode of operation on the byte at 0x03. In this mode, any write from the host to offset 0x03 will cause an interrupt to be generated to the Apollo MCU, waking it up to process the command. Since this interrupt tells the Apollo that all bytes of the command are present in the shared register space then this write must occur as the last operation of the command. Thus we need to use two separate SPI write transactions to send a command to the Apollo.

The Apollo firmware will grab the command from the shared register buffer once it wakes up and will examine the command word from the first 4 bytes to determine what is required in response to this command transaction.

2.2 Receiving Responses From the Apollo Sensor Hub

Figure 2-5: Reading a Response From the Apollo Boot Loader



When the Apollo firmware has a response to send to the host, it first loads the response bytes into the shared register buffer located at offset 0x00 (0) in the shared register space. It then asserts a software interrupt to the host on GPIO[4] using a GPIO output control to pull the line low.

Setting this interrupt alerts the host that the response data is ready to be read from the Apollo. As shown in Figure 2-5, the host uses a SPI read transaction to read the

response type bytes and any additional response bytes from the Apollo I/O slave response buffer.

2.3 Commands in the Boot Loader Repertoire

Table 2-1: Commands in the Boot Loader Repertoire

Command	CMD#	# Bytes	Description
ACK	0	4	Acknowledge
NAK	1	4	Negative Acknowledge
NEW_IMAGE	2	4+12	READY for next data packet
NEW_PACKET	3	4+4+n	IMAGE_CMPLT last data packet was good and so was total CRC.
RESET	4	4	Issue a reset to the Apollo MCU and reboot into user program.
OVERRIDE	5	4+8	
BL VERSION	6	4	Respond with boot loader version.
FW VERSION	7	4	Respond with firmware version.
DBG_READ	8	4	Respond with well-known data array
DBG_ECHO	9	4+n	Respond with complement n bytes in command.

Table 2-2: Response in the Boot Loader Repertoire

Response	RSP #	# Bytes	Description
ACK	0	4	Acknowledge successful NEW_IMAGE command
NAK	1	4	Negative acknowledge bad NEW_IMAGE command
READY	2	4	Send next data packet
IMAGE_CMPLT	3	4	Image complete and CRC checks
BAD_CRC	4	4	Bad CRC on image
ERROR	5	4	Error occurred in command processing
BL_VERSION	6	4+4	Boot loader version information
FW_VERSION	7	4+4	Firmware version information
DBG_READ	8	4+20	Respond with 20 well known bytes
DBG_ECHO	9	4+32	Respond with complement of 32 bytes sent with command

2.4 Logic Analyzer Screen Captures

Figure 2-6: From the Beginning of a SPI Write Transaction

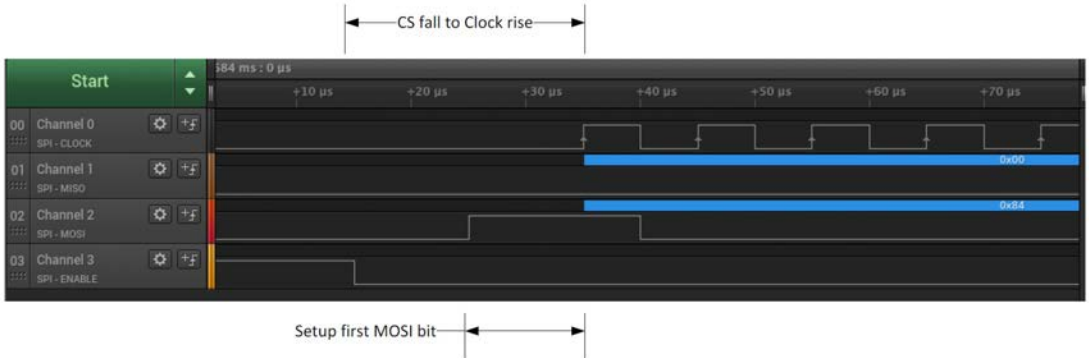
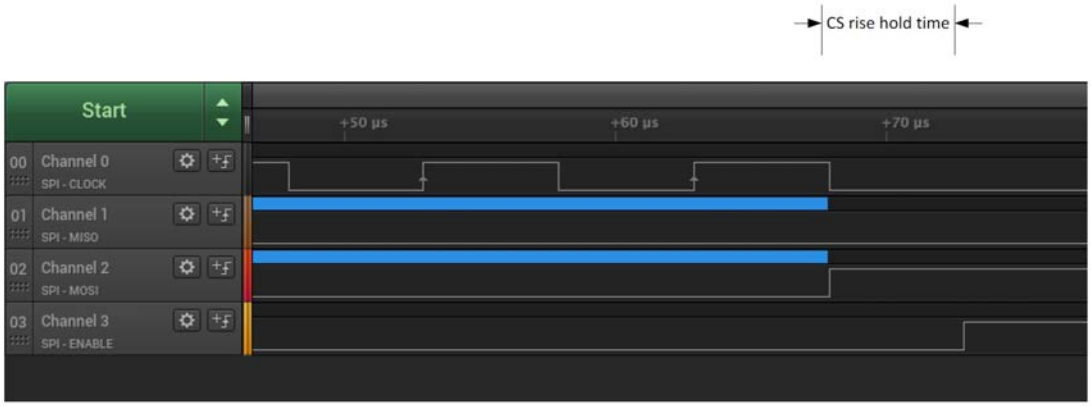


Figure 2-7: From the Middle of a SPI Write Transaction

Figure 2-8: From the End of a SPI Write Transaction



SECTION

3

I²C Mode

The boot loader can be configured at compile time to accept I²C downloads instead of SPI downloads. The boot loader has some minor set up code differences but once the I/O slave is configured and initialized, the rest of the bootloader code is identical regardless of whether I²C or SPI mode is used in the I/O slave.

Refer to the data sheet description of the Apollo MCU to see how the I/O slave behaves in I²C mode versus SPI mode.

6500 River Place Boulevard, Building 7, Suite 200, Austin, TX 78730
www.ambiq.com
sales@ambiq.com
+1 (512) 879-2850

A-MCUAP3-ANGA01EN v1.3
April 2022