04-1779-SDS-ZFR11
8.0
December 20th, 2006

SDMA Scripts Library – Software Design Specification

Abstract:

This document describes the list of the available SDMA scripts

Keywords:

SDMA – I.API - Scripts

APPROVED:

Author	Sign-off Signature 1	Sign-off Signature 2	Sign-off Signature 3
Gregory Meunier	Arnaud Devreese	Claude Brouat	
Sign-off Signature 4	**Sign-off Signature 5**	**Sign-off Signature 6**	

REVISION HISTORY

Version	Date	Name	Reason
0.01	March 22, 04	Laurent Tacquet Gregory Meunier	First version from Tortola requirement
0.02	April 29, 04	Gregory Meunier	Add new scripts (dsp to peripheral, dsp to peripheral dma) and more comments on the IPC scripts.
0.03	May 19,04	Gregory Meunier	Add diagram of the SDMA connected to the EMI, SMIF and MAX
1.0 – D01	June 29, 04	Gregory Meunier	Change document number to the one given by doclet. Version for Dissect review
1.0 – D02	July 09, 04	Gregory Meunier	Document changes after dissect review TLS703
1.0	August 03,04	Gregory Meunier	Change document status: Publish
2.0-D01	August 13, 04	Gregory Meunier	Modification in the scripts that read data from the UART
2.0-D02	Sept 10, 04	Gregory Meunier	Modification in the scripts that read data from the UART after review.
2.0-D03	Oct 14, 04	Jean-Pascal Blondin	Add description of new generic scripts from/to (shared) peripherals to/from per/burst memories
2.0	Oct 15, 04	Jean-Pascal Blondin	Change document status : publish for SDMA_SR01
3.0-D01	Oct 29, 04	Jean-Pascal Blondin	Updated the table on which the available scripts are mentionned
4.0	Dec 16, 04	Gregory Meunier	Add byte swapping support in scripts used for transfering data between DSP and MCU memory.
5.0-D1	Jan 6, 05	Nicuşor Penişoara	Added ap_2_ap script for AP internal data transfers with memory type detection. Added bp_2_ap and ap_2_bp, for IPC data transfers with endiannes support and AP memory type detection.
5.0-D2	Feb 21, 2005	Carmen Marin	Updated dsp_2_dsp script: changed

Version	Date	Name	Reason
			name into bp_2_bp, modified registers' usage.
5.0-D3	Feb 24, 2005	Carmen Marin	Added the dptc_dvfs script.
5.1	Aug 1, 2005	Jean-Pascal Blondin	Update uart script sequence by clearing the ageing interrupt flag
5.2	Aug 1, 2005	Jean-Pascal Blondin	Added byte swapping capability for Dsp mem from/to periphs scripts
6.0	Aug 2, 2005	Jean-Pascal Blondin	Rewrote the table of contents by separating ROMv1 and v2
7.0	Fev 10, 2006	Jean-Pascal Blondin	Details on peripheral size passed through BD mode
7.1	Dec 20, 2006	Neha Agarwal	Details of the additions in patched bp_2ap and ap_2_bp scripts
8.0-D01	Mar 8,2007	Sachin Modi	Updated ap_2_bp and bp_2_ap scripts for IPCv2.
8.0	Mar 23,2007	Sachin Modi	Updated after the review comments.

CONTRIBUTORS

Ownership and maintenance of this document belongs to BSP Team. The author of this document was Gregory Meunier, but the primary contact from January 2005 is Jean-Pascal Blondin.

The following engineers have contributed to the creation of this document through authoring, editing, answering questions, participating in information gathering meetings, and/or reviewing documentation. All input into the creation of this document is greatly appreciated.

Gregory Meunier – BSP team

Laurent Tasquet – E3S team

Jean-Pascal Blondin – BSP team

Nicusor Penisoara - PK986

Carmen Marin - PK986

TABLE OF CONTENTS

List of figures

List of tables

1 Introduction

1.1 Audience Description

This document is for software and hardware engineer working in architecture, design, implementation and test of any platform where SDMA is present.

1.2 Purpose Statement

The purpose of this document is to present the SDMA scripts library to perform point-to-point data transfers in a multi-core silicon system on a chip. The library supports data transfer from core memory space to dedicated core memory space (DSP), from core memory space to peripherals and vice-versa.

1.3 References

All references listed in this table are for Freescale Semiconductor Usage only

Id	Title	Reference	Version	Location
1	I.API Software Architecture Specification	MOT-SFS-IAPI-SAS-001	Version 0.04	http://compass.freescale.net/go/148531376
2	I.API Software Design Specification	MOT-SFS-IAPI-SDS-001	Version 0.05	http://compass.freescale.net/go/148531376
3	SDMA Boot Code specification	SDMA-BOOT-0.02	Version 0.02	http://compass.freescale.net/doc/149133100/SDMA_BOOT_CODE.doc
4	SDMA RAM Builder Software Design Specification	04-1722-SDS-ZFR11	Version 1.0-D01	http://compass.freescale.net/doc/148806070/RAM_Builder_Software_Design_Specification.doc

1.4 Conventions

No special conventions are used in this document.

1.5 Definitions, Acronyms, and Abbreviations

BD: Buffer Descriptor. Data structure located in Host or dedicated host memory space and used for point-to-point data transfer with the SDMA.

CCB: Control Channel Block. Data structure located in Host or dedicated host memory space, each SDMA channel has a dedicated CCB that points to the array of buffer descriptors

SDMA: Smart Direct Memory Access module.

Rainbow: Silicon first generation baseband System on a Chip (SOC) for third generation wireless protocol 3G products.

POG: Pot Of Gold, hip7 version of Rainbow.

ZATS: Zeus, Argon+, Tortola, SCM-A11 baseband or application processor for wireless solution.

I.API: Inter Processor Communication Module Application Programming Interface.

IPCM: Inter Processor Communication Module. First version of the SDMA.

ROM: Read Only Memory.

RAM: Random Access Memory.

Scripts: SDMA program executed on a channel.

WML: Watermark level, lower or upper threshold that triggers a DMA request to the SDMA.

1.6 Overview

The SDMA module is responsible to perform data transfer inside a multi-core platform. Scripts, written in SDMA assembly, have been developed to cover many kind of data transfer. The second section of the document presents a table that indicates which data transfers are possible with the current implementation of the SDMA scripts library. The third section gives the name of the script dedicated to a specific data transfer, but also its size and the requisite input parameters. The fourth section will go back to the parameters and will fully explain their meaning. The fifth section will provide a detailed description of all the scripts of the library. The sixth and last section will show the current image of the SDMA memory.

1.7 Document Location

http://compass.freescale.net/doc/148524911/SDMA_SCRIPTS.doc

1.8 Problem Reporting Instructions

Problems or corrections to this document should be reported by email to Sachin Modi (sachin@freescale.com).

2 DMA channels supported by SDMA scripts library

TO / FROM	FIRI	ATA	UART	UART shared	SIM shared	SSI	SSI shared	CSPI	CSPI shared	SDHC shared	MSHC	CSI	External Memory	ARM internal Memory	DSP Mem
FIRI													✓	✓	✓
ATA													✓		
UART													✓	✓	
UART shared													✓	✓	
SIM shared													✓	✓	✓
SSI													✓	✓	✓
SSI shared													✓	✓	✓
CSPI													✓	✓	✓
CSPI shared													✓	✓	✓
SDHC shared													✓	✓	✓
MSHC													✓	✓	
CSI													✓		
External Memory	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓
ARM internal Memory	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓
DSP Mem			✓	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓

The ✓ means one of the SDMA script library allows the data transfer.

The framed cells :

mean that the corresponding scripts are located in SDMA ROM v1 and v2.

The framed cells :

mean that the corresponding scripts are located in SDMA ROM v2.

3 Scripts library overview

The next three tables present the script library organized in three sections:

1. memory to memory scripts
2. memory to peripheral scripts
3. peripheral to memory scripts

Note: The scripts in the tables above without any link (transfers peripheral <-> memory) are not yet developped but it can be done easily by making them from the existing ones of their family: most of non existing scripts are with the keyword “dsp”; the scripts which must be the references for them are the scripts whose word “dsp” is replaced by “mcu” in their names.

The data transfer column lists the possible types of dma channels that involve the SDMA; to each one a specific script is attached. It has to be noticed that some scripts cover several dma channels, it deals with channel with generic peripheral where only watermark level is needed (not ageing timer, no end_of_packet feature,…). The location means if script resides in ROM or in RAM and the size is given in instructions. The parameters columns are explained in next chapter.

The following terms are used in the previous table that lists all the supported data transfer and also in the bext three tables.

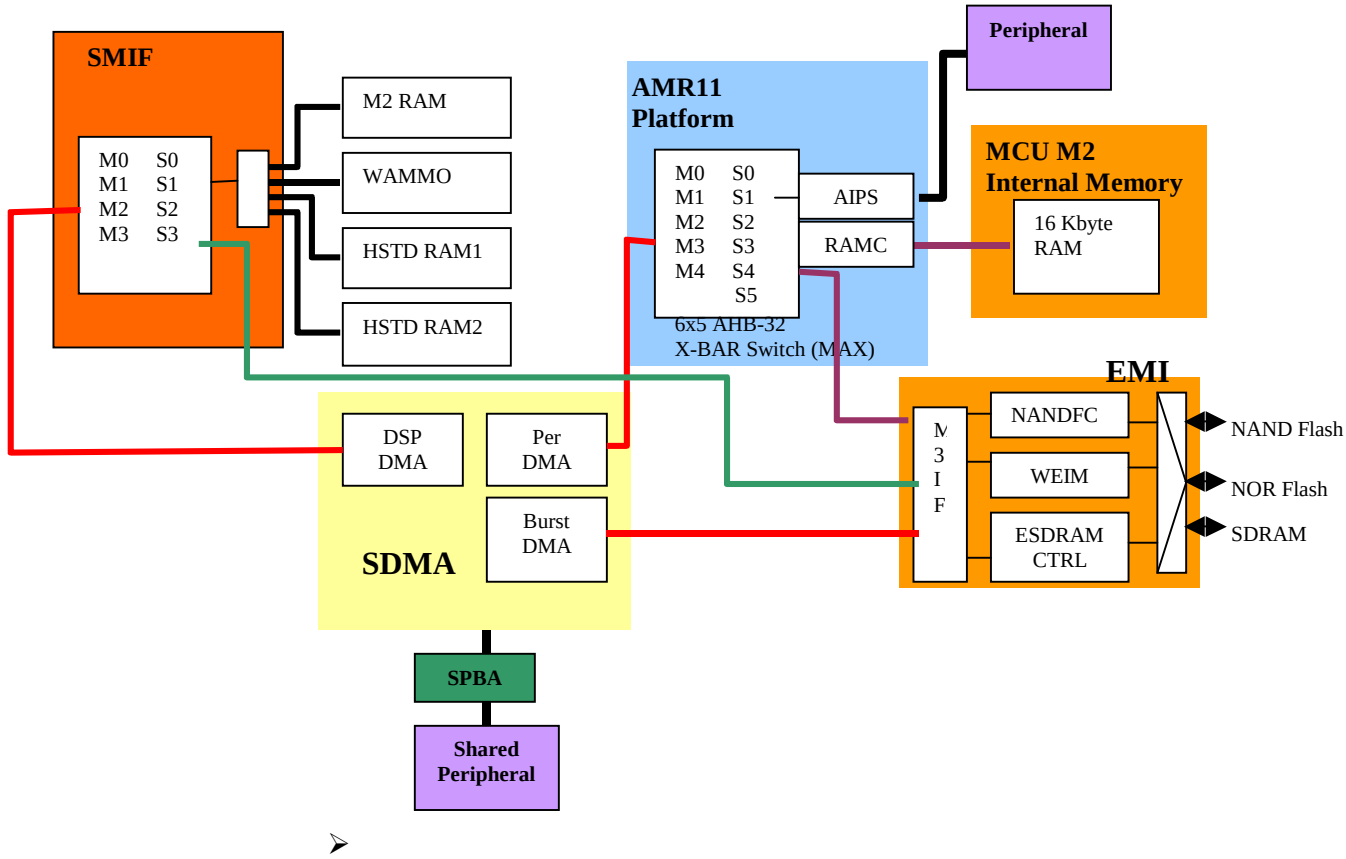
DSP or DSP mem: mean the DSP memories accessible through the SMIF.

EMI or External Memory: mean the external memories connected to the External Memory Interface. Thus, a data transfer to EMI means a data transfer to any of the external memories (NAND Flash, NOR Flash, PSRAM, SDRAM,…).

Host mem or AMR internal memory: mean the memory space accessible through the MAX of the ARM platform. It does not correspond to the peripherals although they are accessible through the MAX too. A data transfer from Host mem means data are read from internal memory of the ARM or from external memory because it is possible to point to an external memory through the MAX as it is also connected to EMI module. The next diagram replaces the SDMA and its DMA port in the hardware architecture. The Host mem is accessible by the Per DMA port of the SDMA, the EMI is accessible by the Burst DMA and the DSP mem by the DSP port.

Shared Peripheral: A same peripheral can be connected to the shared peripheral bus (output of SPBA module) and to the ARM platform; it is the case for UART, CSPI, and SSI in Argon/Tortola/SCM-A11. Therefore *shared UART* indicates the UART connected to the shared peripheral, whereas UART means the UART connected to the ARM platform.

The next diagram gives an overview of the SDMA in its hardware environment.



3.1 Memory to memory scripts

Table 1 Memory to memory scripts

Data transfer	Script to use	DMA ports used	Parameters through context	Parameters through buffer descriptor
Dsp mem→EMI	dsp_2_mcu	DSP DMA/BURST DMA	None	Counter Extended BD, with two address fields. First for source address of the transfer, second for destination address of the transfer.
	dsp_2_mcu_2buf	DSP DMA/BURST DMA	None	For each buffer: Size first address used second addres not used
EMI→Dsp mem	mcu_2_dsp	BURST DMA/ DSP DMA	None	Counter Extended BD, with two address fields. First for source address of the transfer, second for destination address of the transfer.
	mcu_2_dsp_2buf	BURST DMA/ DSP DMA	None	For each buffer: Size first address used second addres not used
EMI→EMI	mcu_2_mcu	BURST DMA	Base address for 8 words of scratch memory (r7)	Counter Extended BD, with two address fields.

				First for source address of the transfer, second for destination address of the transfer.
Host mem→EMI	per_2_mcu	PER DMA/BURST DMA	None	Counter Extended BD, with two address fields. First for source address of the transfer, second for destination address of the transfer.
Host mem → Dsp mem	per_2_dsp_2buf	PER DMA/ DSP DMA	None	For each buffer: Size first address used second addres not used
Host mem→Host mem	per_2_per	PER DMA	None	Counter Extended BD, with two address fields. First for source address of the transfer, second for destination address of the transfer.
EMI→Host mem	mcu_2_per	BURST DMA/PER DMA	None	Counter Extended BD, with two address fields. First for source address of the transfer, second for destination address of the transfer.
Dsp mem → Host mem	dsp_2_per_2buf	DSP DMA/PER DMA	None	Counter Address of the 2 buffers
AP mem → AP mem	ap_2_ap	BURST DMA /PER DMA	M3 start address (r7)	Counter Source address Destination address
AP mem -> BP mem	ap_2_bp	BURST,PER DMA/ DSP DMA	M3 start address (r7)	Byte swapping (ap side only), Counter Source address (ap side)/ Destination address (bp side) Second address not used
AP mem -> BP mem	bp_2_ap	DSP DMA/ BURST,PER DMA	M3 start address (r7)	Byte swapping (ap side only), Counter Source address (ap side)/ Destination address (bp side) Second address not used
Dsp mem → Dsp mem	bp_2_bp	DSP DMA	None	Counter Extended BD, with two address fields. First for source address of the transfer, second for destination address of the transfer.

3.2 Memory to peripheral

Table 2 Memory to peripheral

Data transfer	Script to use	DMA ports used	Parameters through context	Parameters through buffer descriptor
FIRI	mcu_2_firi	BURST DMA/ PER DMA DSP DMA/ PER DMA	FIRI Tx fifo addr (r6) Event_mask (r1) Watermark level (r7)	Counter Memory source address Second address not used
	per_2_firi	PER DMA	FIRI Tx fifo addr (r6) Event_mask (r1) Watermark level (r7)	Counter Memory source address Second address not used
UART	mcu_2_app	BURST DMA/ PER DMA	Uart Tx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b01), Counter Memory source address Second address not used
	per_2_app	PER DMA	Uart Tx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b01), Counter Memory source address Second address not used
	dsp_2_app	DSP DMA/ PER DMA	Uart Tx fifo address (r6) Event_mask (r1) Watermark level (r7)	Byte swapping, Periph size (2'b01), Counter Memory source address Second address not used
SSI	mcu_2_app	BURST DMA/ PER DMA	SSI Tx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b01 or 2'b10 or 2'b11 or 2'b00), Counter Memory source address Second address not used
	per_2_app	PER DMA	SSI Tx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b01 or 2'b10 or 2'b11 or 2'b00), Counter Memory source address Second address not used
	dsp_2_app	DSP DMA/ PER DMA	SSI Tx fifo address (r6) Event_mask (r1) Watermark level (r7)	Byte swapping, Periph size (2'b01 or 2'b10 or 2'b11 or 2'b00), Counter Memory source address Second address not used
CSPI	mcu_2_app	BURST DMA/ PER DMA	CSPI Tx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b00), Counter Memory source address Second address not used
	per_2_app	PER DMA	CSPI Tx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b00), Counter Memory source address Second address not used

	dsp_2_app	DSP DMA/ PER DMA	CSPI Tx fifo address (r6) Event_mask (r1) Watermark level (r7)	Byte swapping, Periph size (2'b00), Counter Memory source address Second address not used
SDHC/ SIM	mcu_2_shp	BURST DMA	SDHC/SIM Tx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b10), Counter Memory source address Second address not used
	per_2_shp	PER DMA	SDHC/SIM Tx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b10), Counter Memory source address Second address not used
	dsp_2_shp	DSP DMA	SDHC/SIM Tx fifo address (r6) Event_mask (r1) Watermark level (r7)	Byte swapping, Periph size (2'b10), Counter Memory source address Second address not used
Shared SSI	mcu_2_shp	BURST DMA	SSI Tx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b01 or 2'b10 or 2'b11 or 2'b00), Counter Memory source address Second address not used
	per_2_shp	PER DMA	SSI Tx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b01 or 2'b10 or 2'b11 or 2'b00), Counter Memory source address Second address not used
	dsp_2_shp	DSP DMA	SSI Tx fifo address (r6) Event_mask (r1) Watermark level (r7)	Byte swapping, Periph size (2'b01 or 2'b10 or 2'b11 or 2'b00), Counter Memory source address Second address not used
Shared CSPI	mcu_2_shp	BURST DMA	CSPI Tx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b00), Counter Memory source address Second address not used
	per_2_shp	PER DMA	CSPI Tx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b00), Counter Memory source address Second address not used
	dsp_2_shp	DSP DMA	CSPI Tx fifo address (r6) Event_mask (r1) Watermark level (r7)	Byte swapping, Periph size (2'b00), Counter Memory source address Second address not used
Shared UART	mcu_2_shp	BURST DMA	Uart Tx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b01), Counter Memory source address Second address not used
	per_2_shp	PER DMA	Uart Tx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b01), Counter Memory source address Second address not used

	dsp_2_shp	DSP DMA	Uart Tx fifo address (r6) Event_mask (r1) Watermark level (r7)	Byte swapping, Periph size (2'b01), Counter Memory source address Second address not used
MSHC	mcu_2_mshc	BURST DMA	MSHC Tx fifo address (r6) Event_mask (r1)	Counter Memory source address Second address not used
	per_2_mshc	PER DMA	MSHC Tx fifo address (r6) Event_mask (r1)	Counter Memory source address Second address not used
ATA	mcu_2_ata	BURST DMA	ATA Tx fifo address (r6) Event_mask1(r0) Event_mask2 (r1) Watermark level (r7)	Counter Memory source address Second address not used

3.3 Peripheral to memory

Table 3 Peripheral to Memory

Data transfer	Script to use	DMA port used	Parameters through context	Parameters through buffer descriptor
… → EMI/DSP mem				
FIRI	firi_2_mcu	PER DMA BURST DMA	FIRI base address (r6) Event_mask (r1) Watermark level (r7)	Counter Mem destination addr Second address not used Suspend_On_EOP (bit 24 of command)
	firi_2_per	PER DMA	FIRI base address (r6) Event_mask (r1) Watermark level (r7)	Counter Mem destination addr Second address not used Suspend_On_EOP (bit 24 of command)
	firi_2_dsp	PER DMA DSP DMA	FIRI base address (r6) Event_mask (r1) Watermark level (r7)	Byte swapping, Counter Mem destination addr Second address not used Suspend_On_EOP (bit 24 of command)
UART	uart_2_mcu	PER DMA BURST DMA	Uart Rx fifo address (r6) Event_mask (r1) Watermark level (r7)	Counter Mem destination addr Second address not used
	uart_2_per	PER DMA	Uart Rx fifo address (r6) Event_mask (r1) Watermark level (r7)	Counter Mem destination addr Second address not used
SSI	app_2_mcu	PER DMA BURST DMA	SSI Rx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b01 or 2'b10 or 2'b11 or 2'b00), Counter Mem destination addr Second address not used

	app_2_per	PER DMA	SSI Rx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b01 or 2'b10 or 2'b11 or 2'b00), Counter Mem destination addr Second address not used
	app_2_dsp	PER DMA DSP DMA	SSI Rx fifo address (r6) Event_mask (r1) Watermark level (r7)	Byte swapping, Periph size (2'b01 or 2'b10 or 2'b11 or 2'b00), Counter Mem destination addr Second address not used
CSPI	app_2_mcu	PER DMA BURST DMA	CSPI Rx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b00), Counter Mem destination addr Second address not used
	app_2_per	PER DMA	CSPI Rx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b00), Counter Mem destination addr Second address not used
	app_2_dsp	PER DMA DSP DMA	CSPI Rx fifo address (r6) Event_mask (r1) Watermark level (r7)	Byte swapping, Periph size (2'b00), Counter Mem destination addr Second address not used
CSI	csi_2_mcu	BURST DMA	CSI Rx fifo address (r6) Event_mask (r1) Watermark level (r7)	Counter Mem destination addr Second address not used
Shared SSI	shp_2_mcu	BURST DMA	SSI Rx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b01 or 2'b10 or 2'b11 or 2'b00), Counter Mem destination addr Second address not used
	shp_2_per	PER DMA	SSI Rx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b01 or 2'b10 or 2'b11 or 2'b00), Counter Mem destination addr Second address not used
	shp_2_dsp	DSP DMA	SSI Rx fifo address (r6) Event_mask (r1) Watermark level (r7)	Byte swapping, Periph size (2'b01 or 2'b10 or 2'b11 or 2'b00), Counter Mem destination addr Second address not used
Shared CSPI	shp_2_mcu	BURST DMA	CSPI Rx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b00), Counter Mem destination addr Second address not used
	shp_2_per	PER DMA	CSPI Rx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b00), Counter Mem destination addr Second address not used
	shp_2_dsp	DSP DMA	CSPI Rx fifo address (r6) Event_mask (r1) Watermark level (r7)	Byte swapping, Periph size (2'b00), Counter Mem destination addr Second address not used

SDHC/SIM	shp_2_mcu	BURST DMA	SDHC/SIM Rx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b10), Counter Mem destination addr Second address not used
	shp_2_per	PER DMA	SDHC/SIM Rx fifo address (r6) Event_mask (r1) Watermark level (r7)	Periph size (2'b10), Counter Mem destination addr Second address not used
	shp_2_dsp	DSP DMA	SDHC/SIM Rx fifo address (r6) Event_mask (r1) Watermark level (r7)	Byte swapping, Periph size (2'b10), Counter Mem destination addr Second address not used
MSHC	mshc_2_mcu	BURST DMA	MSHC Rx fifo address (r6) Event_mask (r1)	Counter Mem destination addr Second address not used
	mshc_2_per	PER DMA	MSHC Rx fifo address (r6) Event_mask (r1)	Counter Mem destination addr Second address not used
Shared UART	uartsh_2_mcu	BURST DMA	UART Rx fifo address (r6) Event_mask (r1) Watermark level (r7)	Counter Mem destination addr Second address not used
	uartsh_2_per	PER DMA	UART Rx fifo address (r6) Event_mask (r1) Watermark level (r7)	Counter Mem destination addr Second address not used
ATA	ata_2_mcu	BURST DMA	ATA base address (r6) Event_mask1(r0) Event_mask2 (r1) Watermark level (r7)	Counter Mem destination addr Second address not used

4 Scripts parameters: definition & mechanism

From the table of the previous chapter, the reader may notice some scripts involve only memories, meaning source and destination addresses point to a memory (DSP memory space, ARM internal memory or external memory; and some scripts use a peripheral as a source or a destination of the data transfer. All the scripts of the SDMA scripts library are platform independent but they need parameters to work - it can be seen as the arguments of a C function – and the parameters are platform dependent. Indeed, for the second scripts "family", those that use peripheral, the peripheral address, the watermark level, the event mask have to be passed to the script. The present chapter explains the parameters meaning and also details the software mechanism, based on buffer descriptor scheme, to "pass" them to the script.

4.1 Parameters definition

4.1.1 Parameters required by data transfer script involving a peripheral

The scripts that have a peripheral as a player of the data transfer need to have the following input parameters:

4.1.1.1 Watermark level (WML)

It is the upper or the lower threshold of the receive or transmit fifo of the peripheral that triggers a DMA request to the SDMA. The WML determines the data transfer loop size, meaning the number of bytes that will be read/write from/to the receive/transmit FIFO. This parameter must be a **multiple** of the peripheral FIFO data size. As an example, if UART1 is the data transfer destination, data must be written to the TX FIFO of the peripheral. The watermark level can be set to 8 bytes for instance (the uart is a 1 byte FIFO data size), this means the UART_TX DMA request will be sent to the SDMA each time there are more than 8 free rooms in the fifo; therefore SDMA loop size will be set to 8 and 8 bytes will be written to the UART_TX fifo. In fact, the loop size is the minimum between the WML and the count field of the buffer descriptor attached to the channel. This concept is later on explained in the present document.

Similarly, if the UART is the source of a data transfer and if the WML is set to 16, each time the DMA request is received by the SDMA, which triggers the associated channel, the data transfer loop size is set to 16, meaning 16 bytes will be read from the UART_RX FIFO.

The watermark level is a “long” (32-bit data) programmed by the ARM and is not supposed to evaluate a lot during application. The WML is given in bytes. Note:

For the transmit fifo, the WML = FIFO_SIZE – FIFOTX_THRESHOLD

For the receive fifo, the WML = FIFORX_THRESHOLD

The FIFOTX_THRESHOLD/FIFORX_THRESHOLD are the values of the peripheral transmit and receive fifo level at which a DMA request is triggered. For instance, if TX fifo is 32 bytes depth and if the DMA request is sent when the number of bytes present in the fifo is below a threshold of 10, the WML will be 22. It means that when the SDMA will receive the DMA request from the peripheral, it will be possible to store 22 bytes in the TX fifo. For the receive fifo, the WML equals the threshold.

4.1.1.2 Event_mask

As previously said, when the WML threshold is over or under passed, a DMA request is sent to the scheduler part of the SDMA. The SDMA scheduler has 32 input pins, called “events”, which are connected to the different DMA request. The relationship between the DMA request name and the event number is project dependent. For instance, in platform A, the UART_RX DMA request is connected to events pin 5; whereas for platform B, it may be connected to events pin 15. The script reads the EVENTS register to check if the received DMA request is compliant with the expected one and it guaranteed that a second DMA request sent during the data transfer is not lost.

For instance, while SDMA is reading the UART_RX FIFO (again, the number of data to be read is given by WML value), the peripheral may receive new data from its input ports. Therefore after the first data transfer, the number of data available in the UART received FIFO can be still higher than the WML threshold, so a second DMA request may be sent.

The event_mask value is generally a 1-bit hot register, which means that if the script attached to the channel must be triggered by DMA request number I, event_mask[I] must be set to 1. Some scripts (like ata_2_mcu) may be triggered by several DMA requests; in that case several bits of the event_mask must be asserted. The event_mask is a long (32-bit) data.

4.1.1.3 Peripheral address

The scripts of the SDMA scripts library are platform independent but as their goal is to address peripheral, they required the base address or the FIFO address of the peripheral. Passing FIFO or peripheral base address depends on the scripts.

4.1.1.4 Data length

The scripts whose name contains the key word "app" or "shp" are generic: they are used to transfer data from/to a 8/16/24 or 32bits peripheral data size. Some jumps to some routines of these scripts depend on this peripheral size. So this parameter is required as input of these scripts. This parameter is passed through the command field of the buffer descriptor and is coded on bits 25, 24 of the mode:

00: 32-bit data transfer.
01: 8-bit data transfer.
10: 16-bit data transfer.
11: 24-bit data transfer.

4.1.2 Parameters required by generic memory to memory (with ap) scripts

The memory to memory (ap source or/and destination) scripts need to have the following parameter set:

- **M3 Start Address**: This parameter specifies the M3 ap start address for the generic memory to memory ap scripts. These new memory to memory scripts are doing transfers from/to ap to/from bp side. On the ARM side, the M3 start address is required to determine which dma port (peripheral/burst DMA) has to be used to do the transfer. On the DSP side, whatever the BP address is (M1, M2 or M3), the dspdma port is always used for the transfer.

4.1.3 Parameters required by all scripts

- **Count**: This parameter specifies the total number of bytes that must be transferred by the associated channel before it is switched-off. In case a peripheral is involved in the data transfer, the count must be a **multiple** of the peripheral FIFO data size like the watermark level.
- **Memory addresses**: These 2 word parameters of the Buffer Descriptor structure are both decoded by all the scripts even if only one is useful for the transfer.

  For memory to memory transfers, the first one specifies the source address of the transfer, the second one the destination address.

  For transfers involving a peripheral, only the first one is useful. It specifies the source or the destination address in the memory space. The second one is useless but is always caught by the script. For instance, with the uart_2_mcu script, which performs data transfer from UART Rx FIFO to external memory, the MCU destination address is passed through the first *memory address* parameter.

- **Command**: Some scripts need some specific information passed through

this Command parameter.

- The scripts performing transfers from/to BP side to/from AP side catch the **byte_swapping** bit of the command (MSB bit) to determine if some software byte swapping is required by the script or not. If mode[31] set (command[7]), byte swapping is enabled. By default, it is disabled. It is only the AP mode byte swapping bit which is decoded by the scripts.

  The scripts which support this feature are the followings:

  Memory to memory transfer type: ap_2_bp, bp_2_ap

  Peripheral scripts: app_2_dsp, firi_2_dsp, dsp_2_app, dsp_2_shp, shp_2_dsp

4.2 Parameter passing mechanism

The six parameters previously described are set by the Core (ARM or DSP) that initiates the data transfer. They are passed to the script according two mechanisms: by channel context and by buffer descriptor as showed in following diagram.

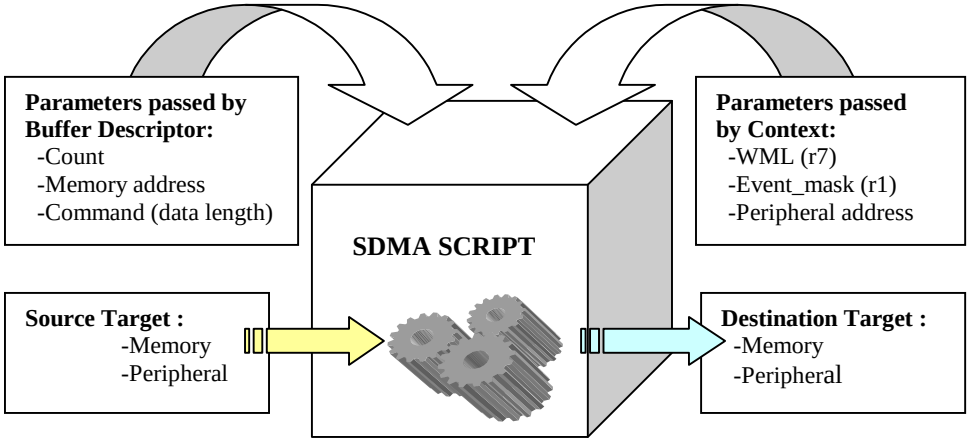


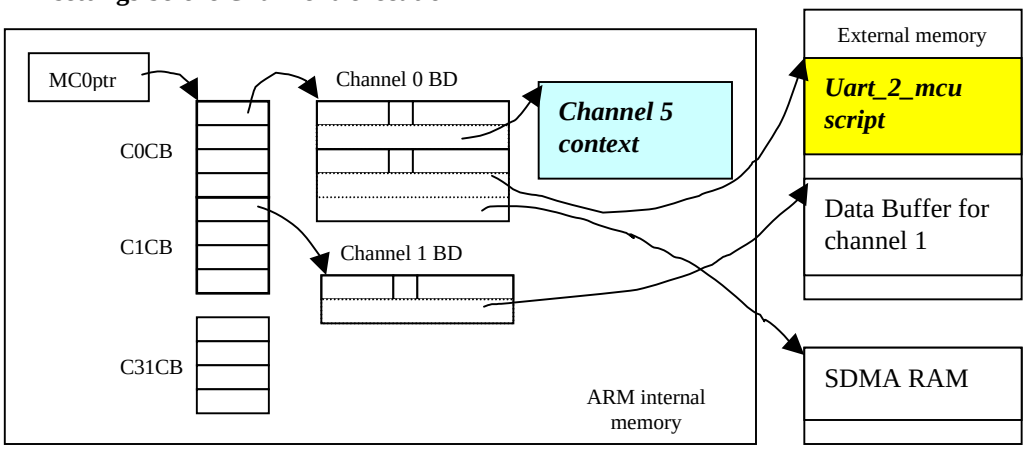
Figure 1 Context & Buffer Descriptor Parameter

4.2.1 Parameters transmit through the Context

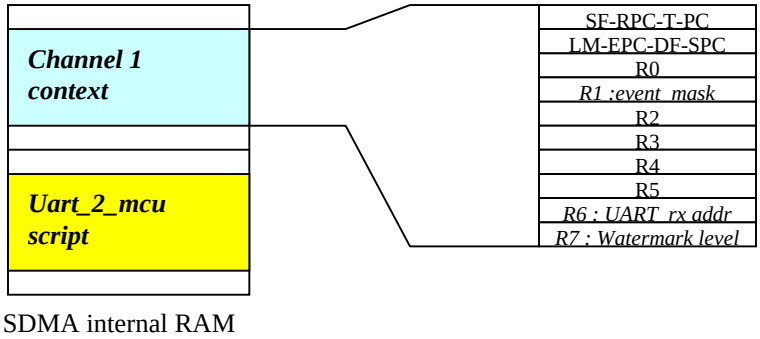
The SDMA channel 0 is dedicated to the boot session, its goal is to download into the SDMA RAM the code and the context of the different SDMA scripts that will be later used during the application. More information about boot code can be found in [3]. The channel 0 is started by ARM after reset when all the data structures (CCB and array of BD) are ready but this channel can also be run each time the ARM updates the data structures, in other word, channel 0 is not only dedicated to be run after reset.
When a channel is started by the SDMA for the very first time, the channel context is restored. It means the SDMA internal registers (General registers, pc, epc, rpc, DF, SF, T, MSA, MDA,…) are overwritten by the values stored in the SDMA RAM in the context area reserved for the channel.

The next picture illustrates the boot sequence to download the context for channel 1 on which the uart_2_mcu script (firstly located in external memory) must be run when UART_RX DMA request connected to event 3 is received.

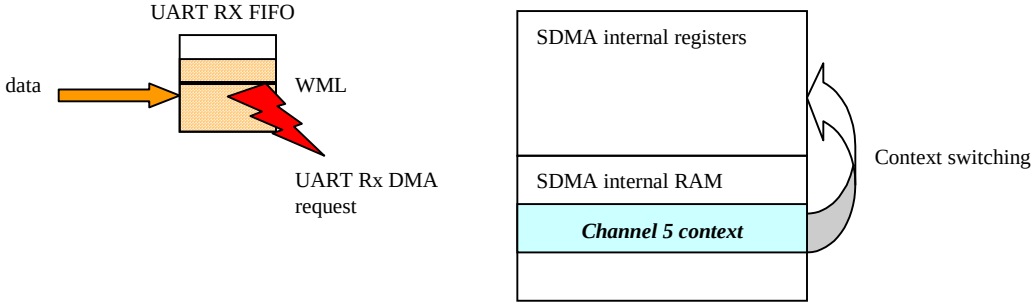
1- ARM settings before Channel 0 execution



2 – after channel 0 execution (ARM triggered)



3 – UART Rx DMA request is sent…



1. ARM first sets up the Channel Control block for channel 0 and 2 buffer descriptors: one to download the context and one to download the script into SDMA RAM. In the channel 1 context, *WML*, *event_mask* and *UART_RX address* are stored respectively in r7, r1 and r6.
   ARM also prepared the channel 1 buffer descriptor where the memory source address is stored in the address buffer field of the BD, the *count* is written into the count field of the BD and for this example no command are needed. The ARM core must also program the SDMA channel event matrix that establishes relationship between an event number and the channels to be triggered. In our example, event 3 must trigger channel 1. Besides, the core must also program the priority of the channel and the DSP enable, Host Enable, Event Override register as explained in [1] (section on scheduler and

host control register).

2. When the data setup is over, the channel 0 is executed and channel 1 context and scripts are loaded in SDMA internal RAM.
3. When UART Rx FIFO overpasses the watermark level, the corresponding DMA request is sent to SDMA and channel 1 context is restored, Program counter (PC) is modified to point to the start address of the channel 1 script and all internal registers are overwritten by their corresponding value in the channel 1 context. Therefore, *WML*, *event_mask* and *peripheral address* parameters are passed to the channel 1 script.

4.2.2 Parameters transmit through the Buffer Descriptor

All the scripts of the library use the same buffer descriptor mechanism as the channel 0. When a script is started for the first time, the first subroutine that is executed is the *getcb* function (see [3] section 5.2 getcb routine) to retrieve pointer to the array of buffer descriptors attached to the script. Then the *getbd* routine is called to read a buffer descriptor. The parameter command must be present in the command field of the BD; the count parameter must be in BD count field and memory address parameter in buffer address field. Extended bit may be used for some scripts as described in next section that details all the script.

To conclude, with the *getbd* routine, the second parameter family (command, count and memory address) is passed to the SDMA script.

5 Current Scripts library

The following scripts are all based on buffer descriptor mechanism, therefore they all use getcb, getbd and setbd sub-routine (getdcb, getdbd and setdbd on ARM side). A clear understanding of these subroutines is prerequisite for a good comprehension of the different SDMA script. It has to be kept in mind that when *setbd* (or *setdbd*) function is called and executed, an interrupt is sent to the ARM (or DSP) according to the potential presence of the Interrupt flag in the flag field of the opened buffer descriptor.

5.1 SDMA ROM V1 scripts

The ROM V1 (scma11 pass1, argon 1-2, tortola 1.0) holds all memory to memory scripts available for these platforms tapeouts.

Note1: In these memory to memory scripts, the number of bytes to transfer from the source memory to the destination is divided by 4 to get the number of word (32-bit) that can be transferred. Most of the time, the transfer will be a transfer of words. There will be a byte or a half-word data transfer (for the latest byte or half-word) if the total number of bytes to transfer can not be divided by 4.

Note2: For these memory to memory scripts of ROM v1, there are no byte manipulations of the data that transferred through one of the general register of the SDMA. Therefore, endianness is taken into account only on the boundary of the DMA ports (BURST, Peripheral and DSP DMA)

5.1.1 mcu_2_mcu

This script is used to transfer data inside the external memory using the burst DMA.

Using this script there is no restriction on the number of byte to transmit or on the source address and destination address alignment. But these parameters could impact the performance. In fact the copy capability of the burst DMA could only be used if the two addresses are word aligned. When it is not the case the SRAM of the SDMA is used as temporary buffer.

5.1.1.1 Parameters transmit through the context:

R7: Base address of the scratch area. This buffer must be 8 words long.

5.1.1.2 Parameters transmit trough the Buffer descriptor:

The number of bytes to transmit is stored in the first Buffer descriptor word

The source address is stored in the second Buffer descriptor word (address field).

The destination address is stored in the Extended Buffer address

5.1.1.3 Use of the General Register during the execution

- R0: counter
- R1: scratch
- R2: channel block descriptor address / base address of the scratch buffer (in SDMA's SRAM)
- R3: current buffer descriptor address
- R4: mode
- R5: source address / scratch
- R6: destination address / scratch
- R7: pointer in the scratch buffer

5.1.1.4 Overview of script functionality

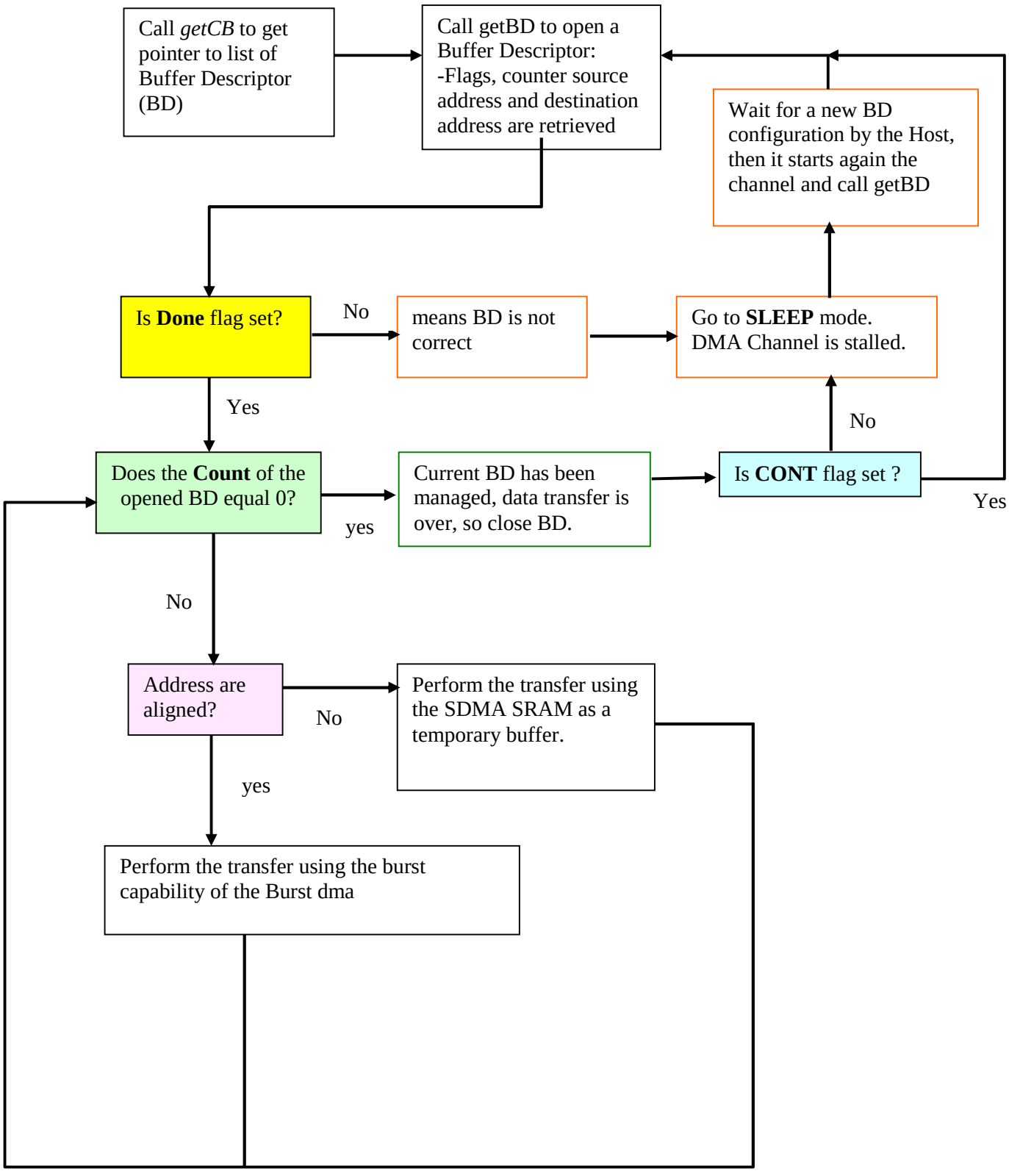
The parameters of the transfer (number of bytes to transfer, source address and destination address) are retrieved from the buffer descriptor.

Depending on the address alignment, the transfer is performed using the burst capability of the burst dma, or using the SDMA SRAM as a temporary buffer.

When the transfer is finished, this algorithm restarts (a new buffer descriptor is read).

5.1.1.5 Script sequences

Return to DMA channels supported by SDMA library



5.1.2 dsp_2_mcu

This script is used to transfer data from the dsp memory space to the external memory.

Using this script there is no restriction on the number of byte to transmit or on the source address and destination address alignment.

5.1.2.1 Parameters transmit through the context

None

5.1.2.2 Parameters transmit trough the Buffer descriptor:

The number of bytes to transmit is stored in the first Buffer descriptor word

The source address in dsp memory space is stored in the second Buffer descriptor word (address field).

The destination address in the external memory is stored in the Extended Buffer address

5.1.2.3 Use of the General Register during the execution

- R0: counter
- R1: scratch
- R2: channel block descriptor
- R3: current buffer descriptor address
- R4: mode
- R5: source address / scratch
- R6: destination address / scratch
- R7: count remainder used by subroutine

5.1.2.4 Overview of script functionality

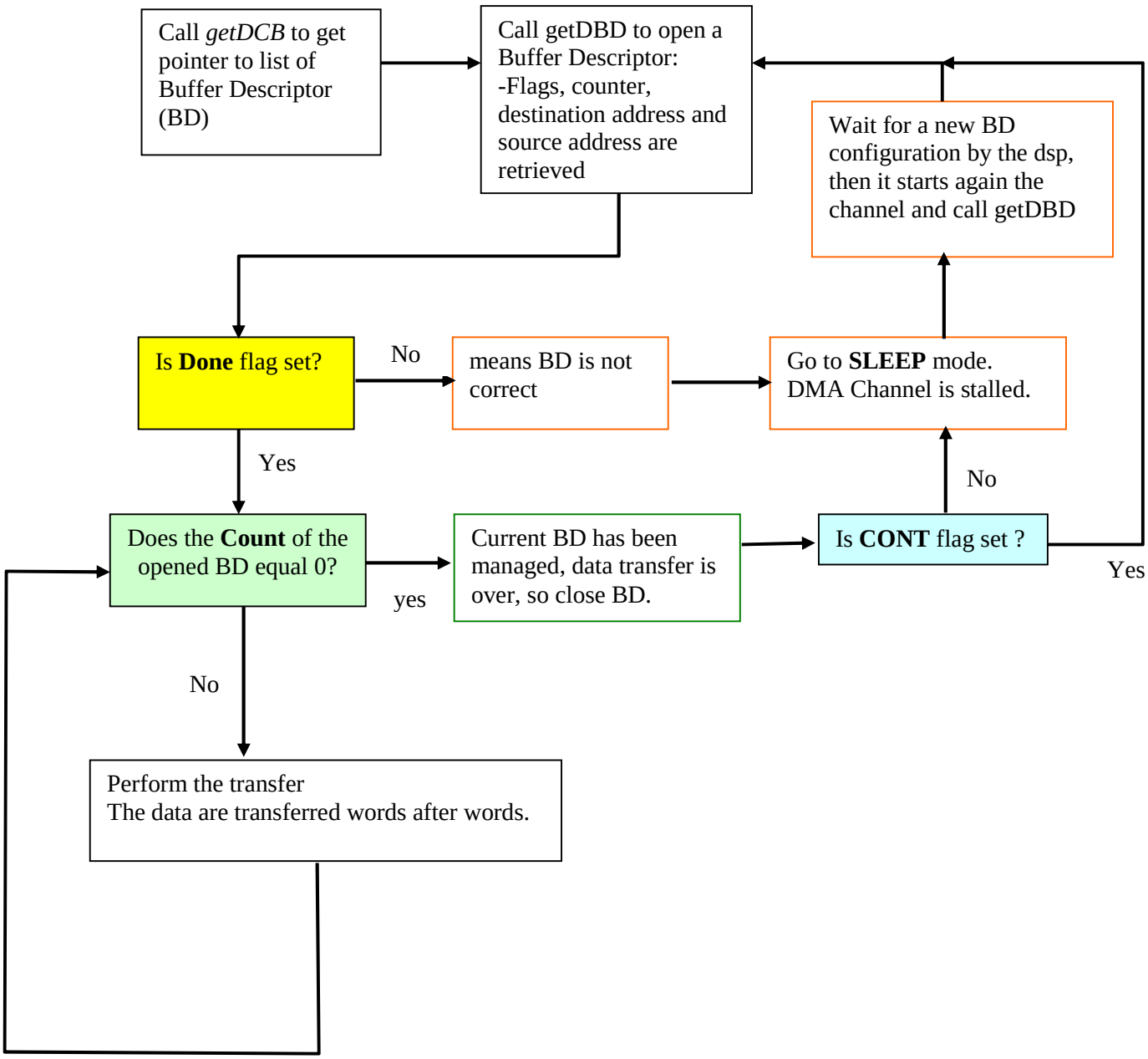
The parameters of the transfer (number of bytes to transfer, source address and destination address) are retrieved from the buffer descriptor.

Then the transfer is performs, the data as transfer as 32 bits long data. So if the endian modes are different on the two sides, there is no rotation inside the word.

When the transfer is performs this algorithm is restarted (a new buffer descriptor is read)

5.1.2.5 Script sequences

Return to DMA channels supported by SDMA library



5.1.3 mcu_2_dsp

This script is used to transfer data from the external memory to the dsp memory space.

Using this script there is no restriction on the number of byte to transmit or on the source address and destination address alignment.

5.1.3.1 Parameters transmit through the context

None

5.1.3.2 Parameters transmit trough the Buffer descriptor:

The number of bytes to transmit is stored in the first Buffer descriptor word

The source address in the external memory is stored in the second Buffer descriptor word (address field).

The destination address in dsp memory space is stored in the Extended Buffer address

5.1.3.3 Use of the General Register during the execution

- R0: counter
- R1: scratch
- R2: channel block descriptor
- R3: current buffer descriptor address
- R4: mode
- R5: source address / scratch
- R6: destination address / scratch
- R7: count remainder used by subroutine

5.1.3.4 Overview of script functionality

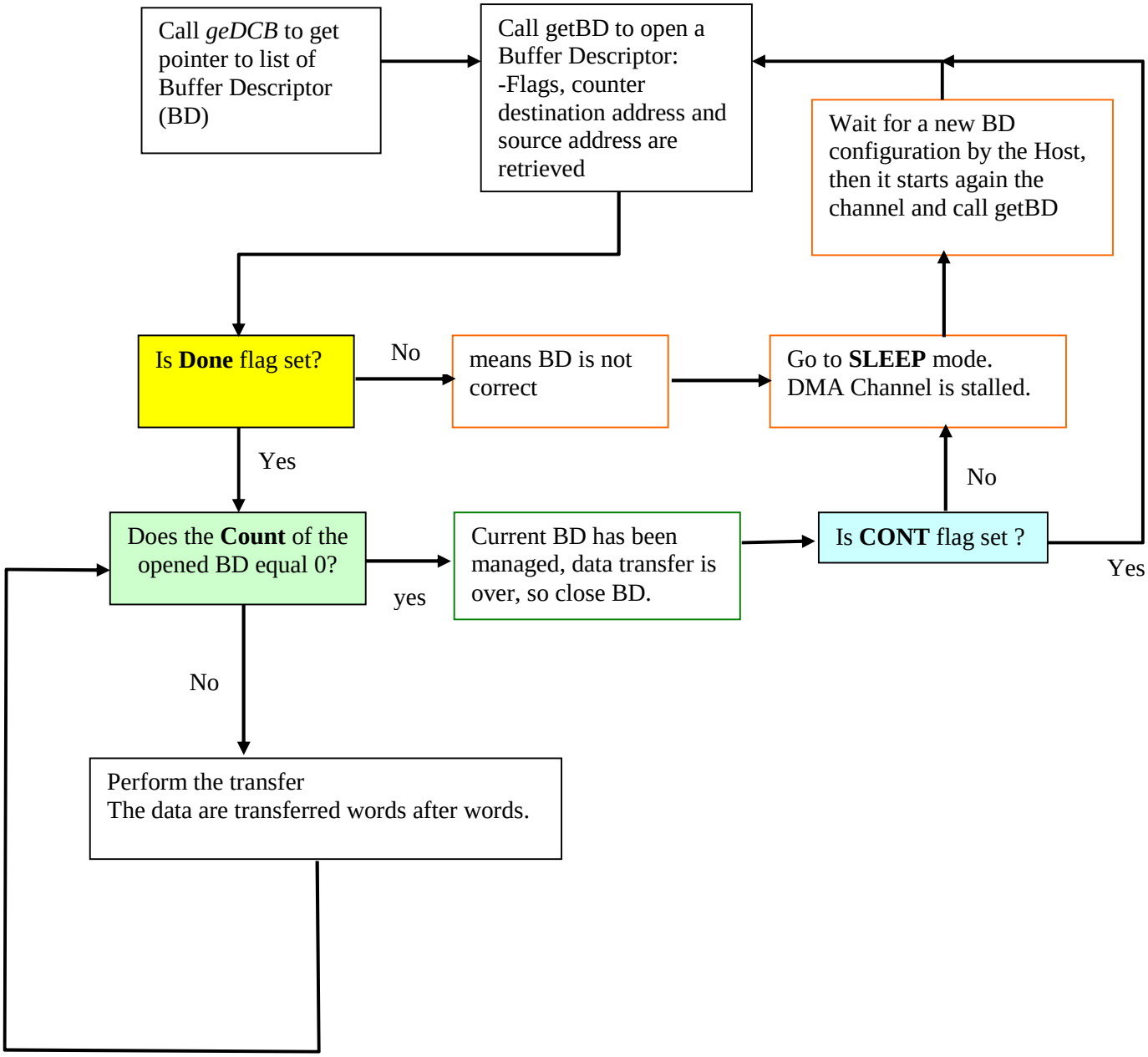
The parameters of the transfer (number of bytes to transfer, source address and destination address) are retrieved from the buffer descriptor.

Then the transfer is performs, the data as transfer as 32 bits long data. So if the endian modes are different on the two sides, there is no rotation inside the word.

When the transfer is performs this algorithm is restarted (a new buffer descriptor is read)

5.1.3.5 Script sequences

Return to DMA channels supported by SDMA library



5.1.4 mcu_2_per

This script is used to transfer data from the external memory to the host memory space.

Using this script there is no restriction on the number of byte to transmit or on the source address and destination address alignment.

5.1.4.1 Parameters transmit through the context

None

5.1.4.2 Parameters transmit trough the Buffer descriptor:

The number of bytes to transmit is stored in the first Buffer descriptor word

The source address in the external memory is stored in the second Buffer descriptor word (address field).

The destination address in the host memory space is stored in the Extended Buffer address

5.1.4.3 Use of the General Register during the execution

- R0: counter
- R1: scratch
- R2: channel block descriptor
- R3: current buffer descriptor address
- R4: mode
- R5: source address / scratch
- R6: destination address / scratch
- R7: count remainder

5.1.4.4 Overview of script functionality

The parameters of the transfer (number of bytes to transfer, source address and destination address) are retrieved from the buffer descriptor.

Firstly the necessary number of data to word align the destination address are transferred.

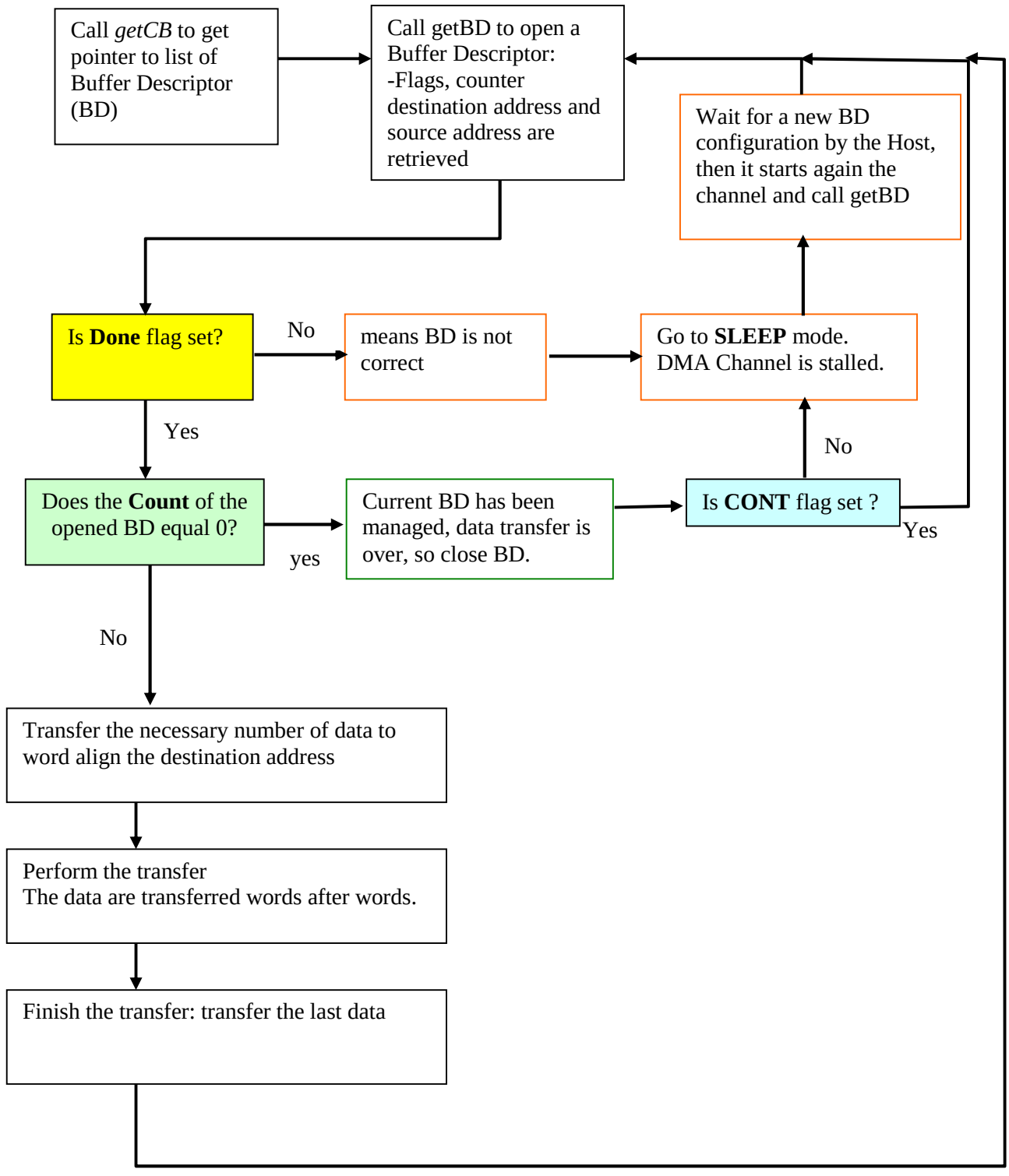
Then the main part of transfer is performs, the data as transfer as 32 bits long data.

The last data are transferred before closing the buffer.

When the transfer is performs this algorithm is restarted (a new buffer descriptor is read)

5.1.4.5 Script sequences

Return to DMA channels supported by SDMA library



5.1.5 per_2_mcu

This script is used to transfer data from the host memory space to the external memory.

Using this script there is no restriction on the number of byte to transmit or on the source address and destination address alignment.

5.1.5.1 Parameters transmit through the context

None

5.1.5.2 Parameters transmit trough the Buffer descriptor:

The number of bytes to transmit is stored in the first Buffer descriptor word

The source address in the host memory space is stored in the second Buffer descriptor word (address field).

The destination address in the external memory is stored in the Extended Buffer address

5.1.5.3 Use of the General Register during the execution

- R0: counter
- R1: scratch
- R2: channel block descriptor
- R3: current buffer descriptor address
- R4: mode
- R5: source address / scratch
- R6: destination address / scratch
- R7: count remainder

5.1.5.4 Overview of script functionality

The parameters of the transfer (number of bytes to transfer, source address and destination address) are retrieved from the buffer descriptor.

Firstly the necessary number of data to word align the source address are transferred.

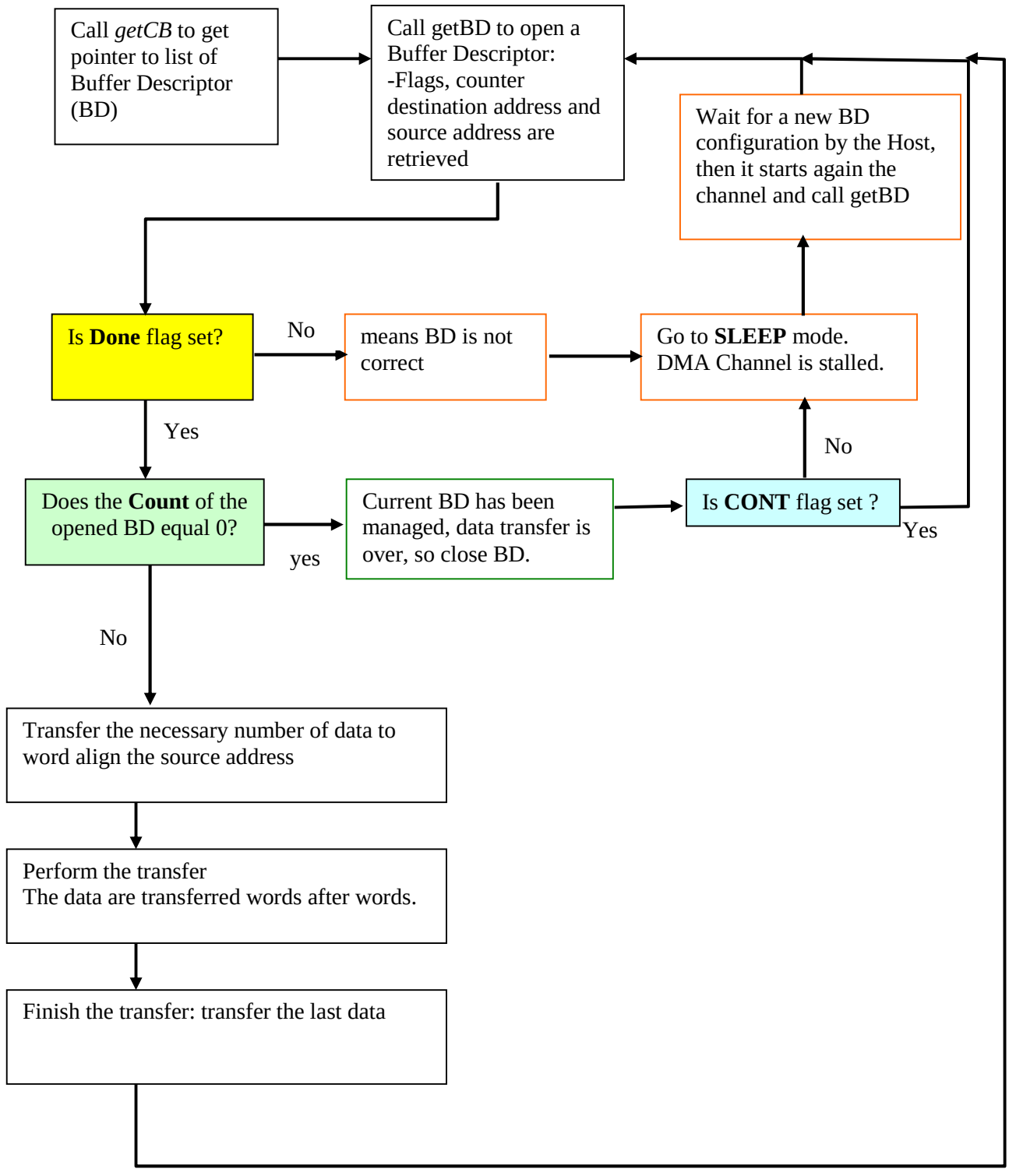
Then the main part of transfer is performs, the data as transfer as 32 bits long data.

The last data are transferred before closing the buffer.

When the transfer is performs this algorithm is restarted (a new buffer descriptor is read)

5.1.5.5 Script sequences

Return to DMA channels supported by SDMA library



5.1.6 mcu_2_dsp_2buf

This script is used to transfer data from the external memory to the dsp memory space.

Using this script there is no restriction on the number of byte to transmit or on the source address and destination address alignment. And the buffer are defined on each side, they do not need to have the same size.

5.1.6.1 Parameters transmit through the context

None

5.1.6.2 Parameters transmit trough the Buffer descriptor:

In the buffer descriptor on the dsp side:

The number of bytes to transmit is stored in the first Buffer descriptor word

The destination address in the external memory is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used

In the buffer descriptor on the host side:

The number of bytes to transmit is stored in the first Buffer descriptor word

The source address in the external memory is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used

5.1.6.3 Use of the General Register during the execution

- R0: channel context base address, scratch, loop count
- R1: channel scratch area base address (set by getsw)
- R2: channel control block pointer (set by getcb/getdcb)
- R3: current buffer descriptor pointer (set by getcb/getdcb)
- R4: bd mode word (set by getbd/getdbd), MCU bd count
- R5: bd address word (set by getbd/getdbd), MCU bd address
- R6: DSP bd address
- R7: DSP bd count, data

5.1.6.4 Overview of script functionality

The parameters of the transfer on the dsp side (size of the buffer, destination address) are retrieved from the buffer descriptor.

The parameters of the transfer on the host side (size of the buffer, source address) are retrieved from the buffer descriptor.

Then a transfer loop is performed based on the size of the smallest buffer.

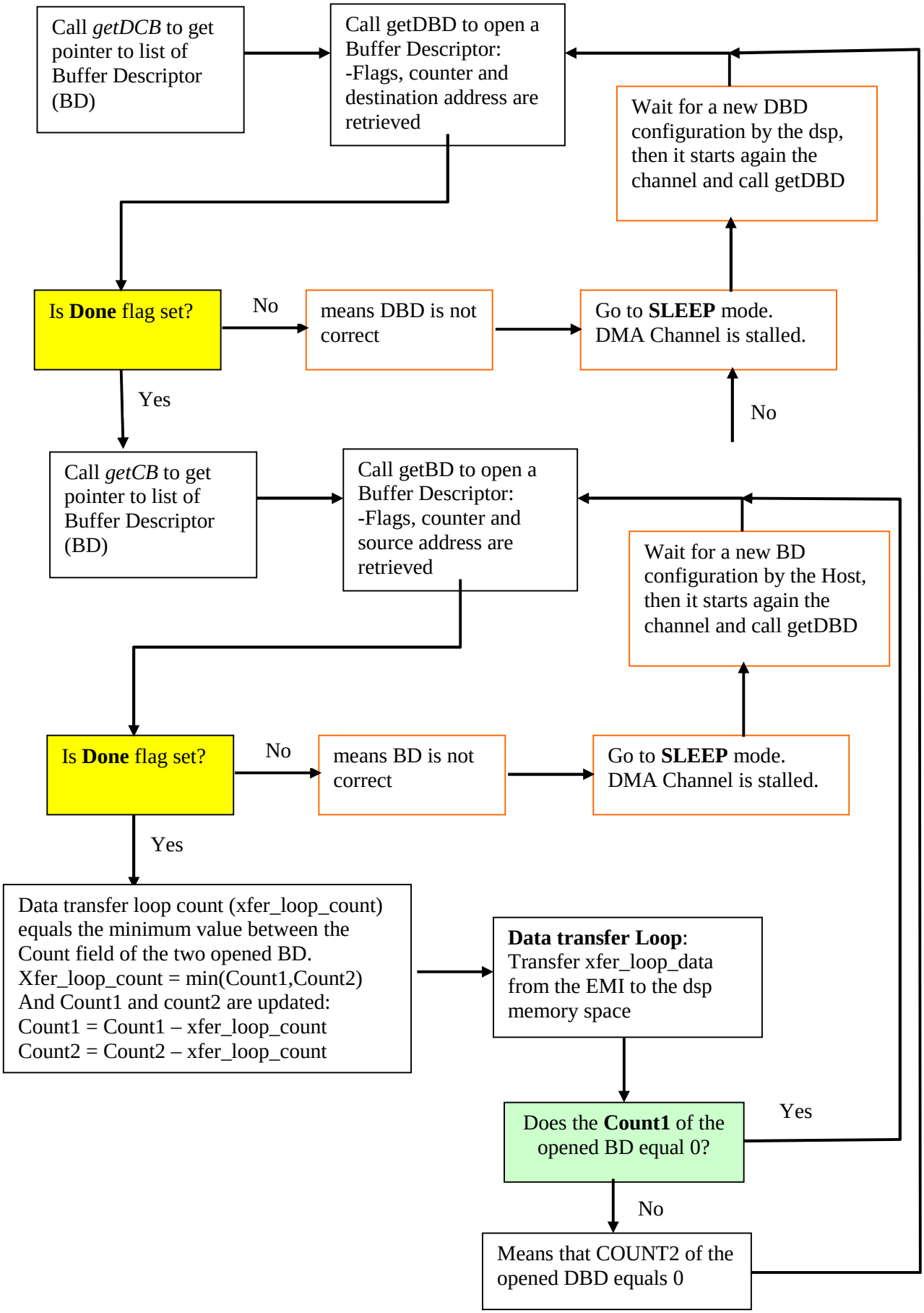
When the SDMA has finished to use a buffer (buffer on the host side is empty, or buffer on the dsp side is filled), this buffer is closed, and a new one is opened. And a new transfer loop is executed.

The channel is closed when all the buffers on a side have been used. The script can

handle source and destination buffers of different size as described in IPC scripts and use cases section that gives more details about the Last bit feature.

5.1.6.5 Script sequences

Return to DMA channels supported by SDMA library



5.1.7 dsp_2_mcu_2buf

This script is used to transfer data from the dsp memory space to the external memory.

Using this script there is no restriction on the number of byte to transmit or on the source address and destination address alignment. And the buffer are defined on each side, they do not need to have the same size.

5.1.7.1 Parameters transmit through the context

None

5.1.7.2 Parameters transmit trough the Buffer descriptor:

In the buffer descriptor on the host side:

The number of bytes to transmit is stored in the first Buffer descriptor word

The destination address in the external memory is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used

In the buffer descriptor on the dsp side:

The number of bytes to transmit is stored in the first Buffer descriptor word

The source address in the external memory is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used

5.1.7.3 Use of the General Register during the execution

- R0: channel context base address, scratch, loop count
- R1: channel scratch area base address (set by getsw)
- R2: channel control block pointer (set by getcb/getdcb)
- R3: current buffer descriptor pointer (set by getcb/getdcb)
- R4: bd mode word (set by getbd/getdbd), DSP bd count
- R5: bd address word (set by getbd/getdbd), DSP bd address
- R6: MCU bd address
- R7: MCU bd count, data

5.1.7.4 Overview of script functionality

The parameters of the transfer on the host side (size of the buffer, destination address) are retrieved from the buffer descriptor.

The parameters of the transfer on the dsp side (size of the buffer, source address) are retrieved from the buffer descriptor.

Then a transfer loop is performed based on the size of the smallest buffer.

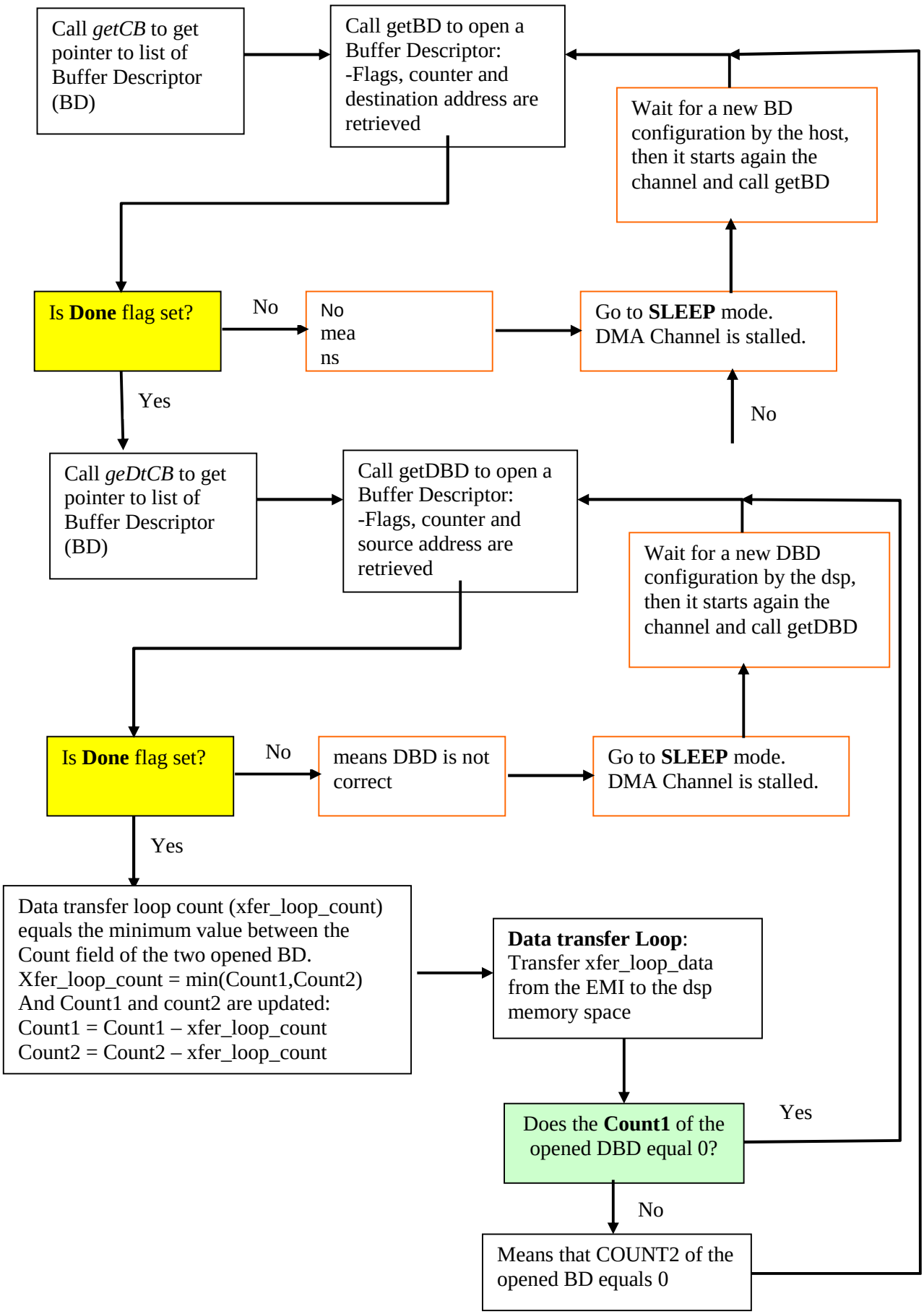
When the SDMA has finished to use a buffer (buffer on the dsp side is empty, or buffer on the host side is filled), this buffer is closed, and a new one is opened. And a new transfer loop is executed.

The channel is closed when all the buffers on a side have been used. The script can

handle source and destination buffers of different size as described in IPC scripts and use cases section that gives more details about the Last bit feature.

5.1.7.5 Script sequences

Return to DMA channels supported by SDMA library



5.1.8 per_2_dsp_2buf

This script is used to transfer data from the AMR internal memory to the DSP memory space.

Using this script there is no restriction on the number of byte to transmit or on the source address and destination address alignment. And the buffer are defined on each side, they do not need to have the same size.

5.1.8.1 Parameters transmit through the context

None

5.1.8.2 Parameters transmit trough the Buffer descriptor:

In the buffer descriptor on the dsp side:

The number of bytes to transmit is stored in the first Buffer descriptor word

The destination address in the external memory is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used

In the buffer descriptor on the host side:

The number of bytes to transmit is stored in the first Buffer descriptor word

The source address in the external memory is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used

5.1.8.3 Use of the General Register during the execution

- R0: channel context base address, scratch, loop count
- R1: channel scratch area base address (set by getsw)
- R2: channel control block pointer (set by getcb/getdcb)
- R3: current buffer descriptor pointer (set by getcb/getdcb)
- R4: bd mode word (set by getbd/getdbd), MCU bd count
- R5: bd address word (set by getbd/getdbd), MCU bd address
- R6: DSP bd address
- R7: DSP bd count, data

5.1.8.4 Overview of script functionality

The parameters of the transfer on the dsp side (size of the buffer, destination address) are retrieved from the buffer descriptor.

The parameters of the transfer on the host side (size of the buffer, source address) are retrieved from the buffer descriptor.

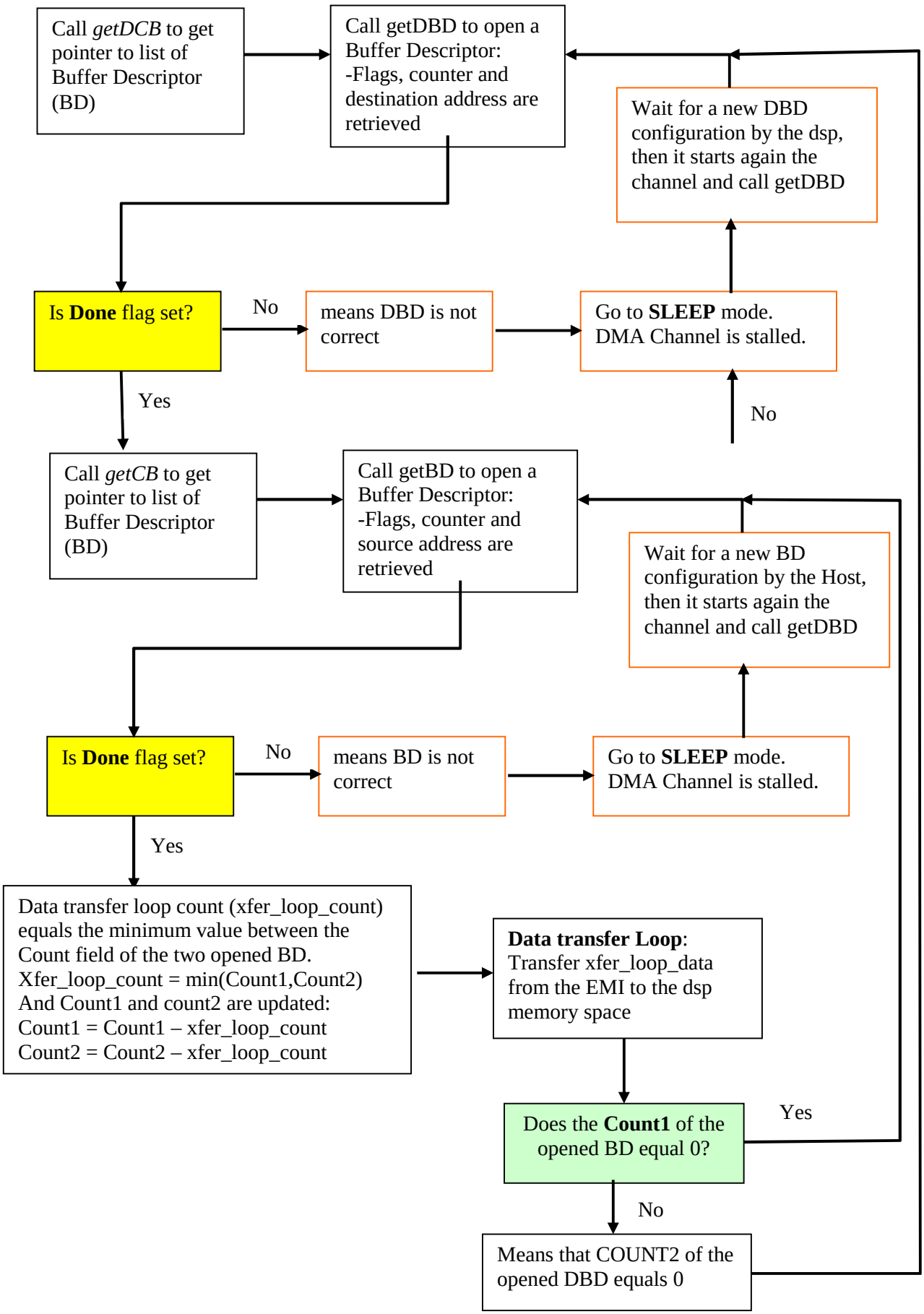
Then a transfer loop is performed based on the size of the smallest buffer.

When the SDMA has finished to use a buffer (buffer on the host side is empty, or buffer on the dsp side is filled), this buffer is closed, and a new one is opened. And a new transfer loop is executed.

The channel is closed when all the buffers on a side have been used.

5.1.8.5 Script sequences

Return to DMA channels supported by SDMA library



5.1.9 dsp_2_per_2buf

This script is used to transfer data from the dsp memory space to the ARM internal memory.

Using this script there is no restriction on the number of bytes to transmit or on the source address and destination address alignment. And the buffer are defined on each side, they do not need to have the same size.

5.1.9.1 Parameters transmit through the context

None

5.1.9.2 Parameters transmit trough the Buffer descriptor:

In the buffer descriptor on the host side:

The number of bytes to transmit is stored in the first Buffer descriptor word

The destination address in the external memory is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used

In the buffer descriptor on the dsp side:

The number of bytes to transmit is stored in the first Buffer descriptor word

The source address in the external memory is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used

5.1.9.3 Use of the General Register during the execution

- R0: channel context base address, scratch, loop count
- R1: channel scratch area base address (set by getsw)
- R2: channel control block pointer (set by getcb/getdcb)
- R3: current buffer descriptor pointer (set by getcb/getdcb)
- R4: bd mode word (set by getbd/getdbd), DSP bd count
- R5: bd address word (set by getbd/getdbd), DSP bd address
- R6: MCU bd address
- R7: MCU bd count, data

5.1.9.4 Overview of script functionality

The parameters of the transfer on the host side (size of the buffer, destination address) are retrieved from the buffer descriptor.

The parameters of the transfer on the dsp side (size of the buffer, source address) are retrieved from the buffer descriptor.

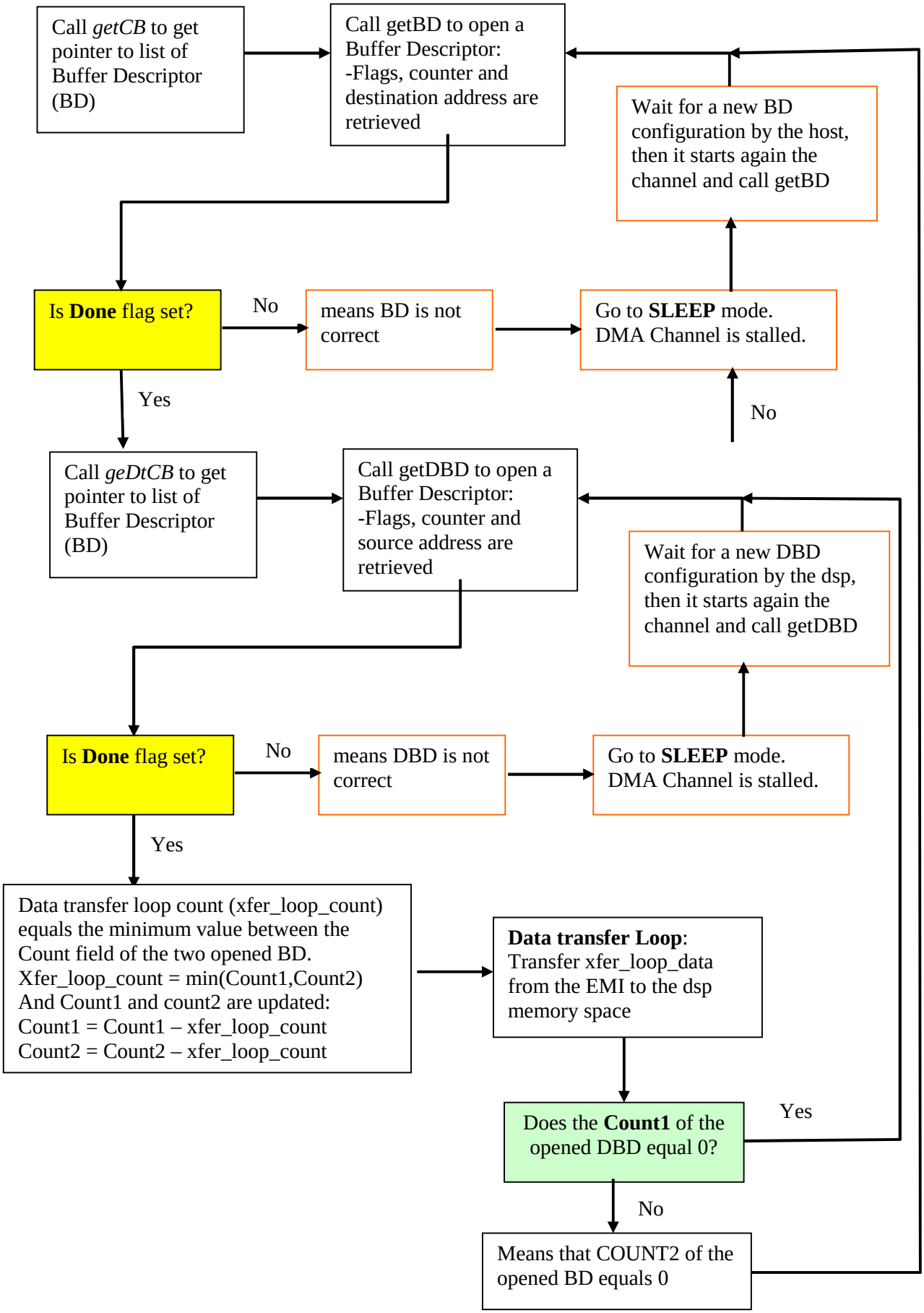
Then a transfer loop is performed based on the size of the smallest buffer.

When the SDMA has finished to use a buffer (buffer on the dsp side is empty, or buffer on the host side is filled), this buffer is closed, and a new one is opened. And a new transfer loop is executed.

The channel is closed when all the buffers on a side have been used.

5.1.9.5 Script sequences

Return to DMA channels supported by SDMA library



5.1.10 per_2_per

This script is used to transfer data inside the ARM internal memory using the Peripheral DMA port of the SDMA.

Using this script there is no restriction on the number of byte to transmit or on the source address and destination address alignment. The copy mode of the peripheral DMA is always used but if addresses are not aligned to a word boundary, the copy mode will do byte data transfers. If addresses are word aligned, word data are transferred, which improves data throughput.

5.1.10.1 Parameters transmit through the context:

None

5.1.10.2 Parameters transmit trough the Buffer descriptor:

The number of bytes to transmit is stored in the first Buffer descriptor word

The source address is stored in the second Buffer descriptor word (address field).

The destination address is stored in the Extended Buffer address

5.1.10.3 Use of the General Register during the execution

- R0: counter
- R1: scratch
- R2: channel block descriptor address / scratch
- R3: current buffer descriptor address
- R4: mode
- R5: source address / scratch
- R6: destination address / scratch
- R7: pointer in the scratch buffer

5.1.10.4 Overview of script functionality

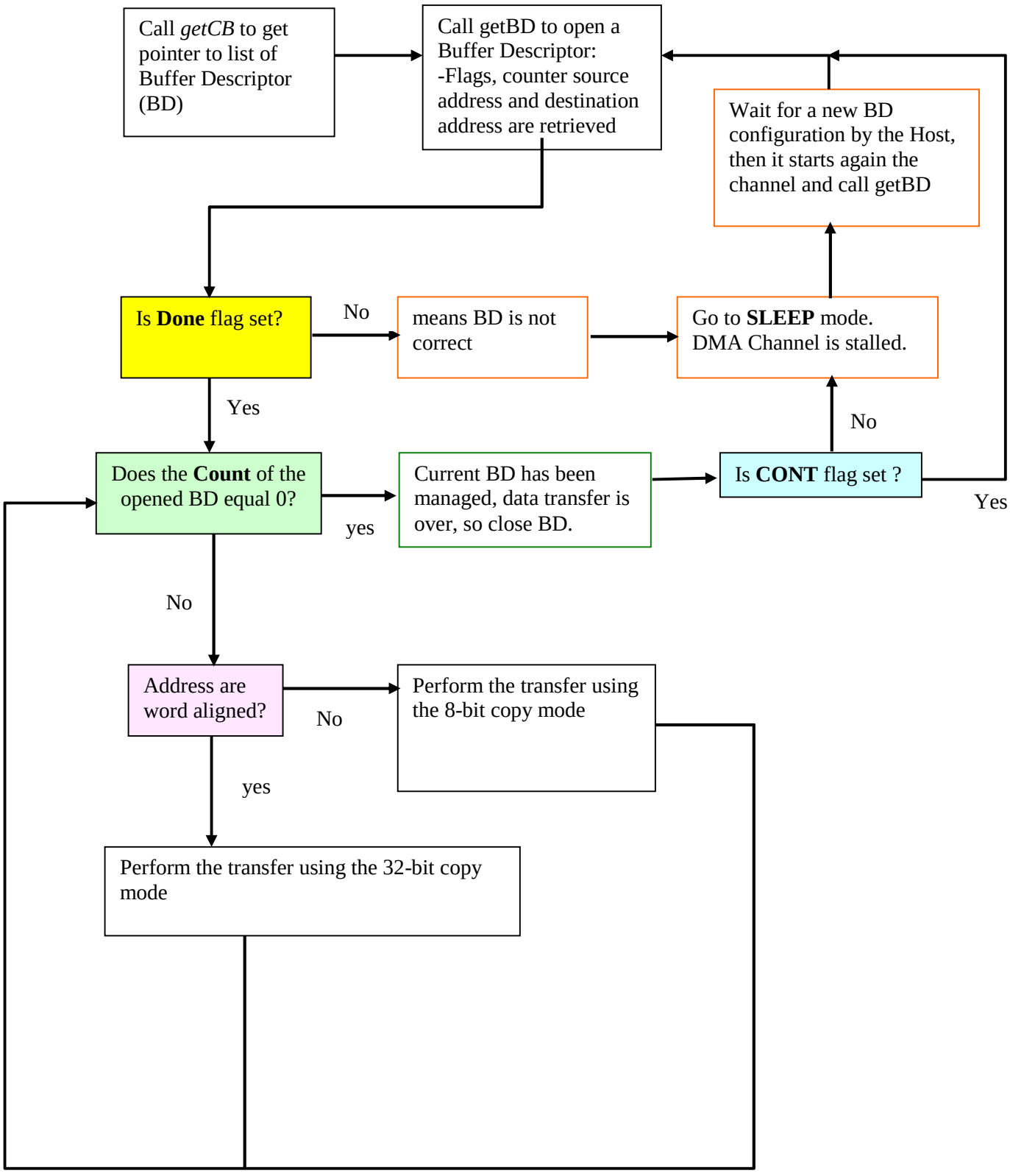
The parameters of the transfer (number of bytes to transfer, source address and destination address) are retrieved from the buffer descriptor.

Depending on the address alignment, the transfer is performed using the copy capability of the peripheral dma.

When the transfer is finished, this algorithm restarts (a new buffer descriptor is read).

5.1.10.5 Script sequences

Return to DMA channels supported by SDMA library



5.1.11 dsp_2_dsp

This script is used to transfer data inside the DSP memory space using the DSP DMA port of the SDMA.

Using this script there is no restriction on the number of byte to transmit or on the source address and destination address alignment. But these parameters could impact the performance. In fact the copy capability of the DSP DMA could only be used if the two addresses are aligned; meaning source and destination address must have the same modulo. For instance, a copy mode is used if source address equals 0[4] and destination address equals 0[4]; or source is 1[4] and destination is 1[4]. When it is not the case the SRAM of the SDMA is used as temporary buffer.

5.1.11.1 Parameters transmit through the context:

R7: Base address of the scratch area. This buffer must be 8 words long.

5.1.11.2 Parameters transmit trough the Buffer descriptor:

The number of bytes to transmit is stored in the first Buffer descriptor word

The source address is stored in the second Buffer descriptor word (address field).

The destination address is stored in the Extended Buffer address

5.1.11.3 Use of the General Register during the execution

- R0: counter
- R1: scratch
- R2: channel block descriptor address / base address of the scratch buffer (in SDMA's SRAM)
- R3: current buffer descriptor address
- R4: mode
- R5: source address / scratch
- R6: destination address / scratch
- R7: pointer in the scratch buffer

5.1.11.4 Overview of script functionality

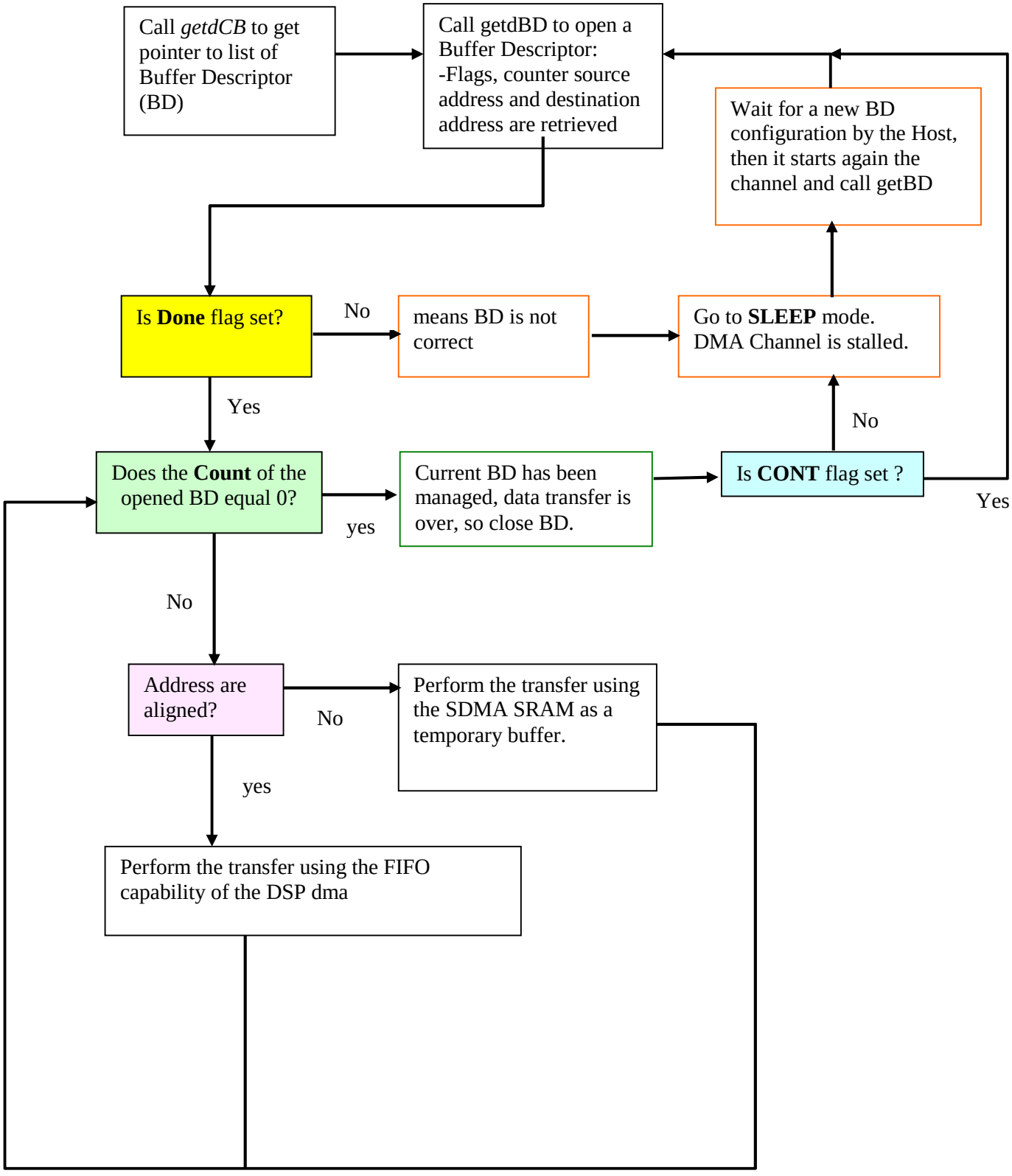
The parameters of the transfer (number of bytes to transfer, source address and destination address) are retrieved from the buffer descriptor.

Depending on the address alignment, the transfer is performed using the FIFO capability of the DSP DMA, or using the SDMA SRAM as a temporary buffer.

When the transfer is finished, this algorithm restarts (a new buffer descriptor is read).

5.1.11.5 Script sequences

Return to DMA channels supported by SDMA library



5.2 SDMA ROM V2 scripts

The ROM V2 (scma11 pass2, argon LV, tortola 1.1) holds the new generic memory to memory scripts (ap_2_ap, ap_2_bp, bp_2_ap, bp_2_bp) and the peripheral most useful scripts (app_2_mcu, mcu_2_app, uart_2_mcu, uartsh_2_mcu, mcu_2_shp, shp_2_mcu).

In these memory to memory scripts, the number of bytes to transfer from the source memory to the destination is divided by 4 to get the number of word (32-bit) that can be transferred. Most of the time, the transfer will be a transfer of words. There will be a byte or half data transfer at the end of the transfer if the total number of bytes to transfer can not be divided by 4.

The generic memory to memory scripts **ap_2_bp, bp_2_ap** support the software **byte swapping** feature to solve endianness issue (ARM and DSP in a different endianness => data needs to be swapped as words transfers happen most of the time). This feature is available in the Command field of the buffer descriptor. (see details in command paragraph of 4.1.3 section)

5.2.1 ap_2_bp

This generic script is used to transfer data from the AP(host) to the BP (DSP memory) space.

Using this script there is no restriction on the number of byte to transmit, the source address and destination address alignment, source memory type. The buffers defined on each side do not need to have the same size. This script offers the byte swapping feature which can be enable/disable through command passed in the Command field of the Buffer Descriptor located on MCU side.

5.2.1.1 Parameters transmit through the context

R7: The AP M3 start address.

5.2.1.2 Parameters transmit trough the Buffer descriptor:

In the buffer descriptor on the dsp side:

The number of bytes to transmit is stored in the first Buffer descriptor word

The destination address in the BP memory is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used.

In the buffer descriptor on the host side:

The Command field is used to enable/disable byte swapping feature (command == 0x80: byte swapping ON – Default is OFF).

The number of bytes to transmit is stored in the first Buffer descriptor word

The source address in the AP memory is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used.

5.2.1.3 Use of the General Register during the execution

- R0: counter during loop, base address of the context when needed
- R1: BP current buffer descriptor count (set by getcb/getdcb)
- R2: channel control block pointer (set by getcb/getdcb), scratch
- R3: current buffer descriptor pointer, scratch
- R4: AP bd mode word, count, scratch
- R5: destination address loaded by getdb, scratch
- R6: base address of context, scratch
- R7: M3 start address

5.2.1.4 Overview of script functionality

The parameters of the transfer on the dsp side (size of the buffer, destination address) are retrieved from the buffer descriptor

The parameters of the transfer on the host side (size of the buffer, source address) are retrieved from the buffer descriptor and also the Byte swapping flag.

Then a transfer loop is performed based on the size of the smallest buffer. If Byte swapping is ON, each time a 32-bit data is exchanged between Per/Burst DMA and DSP DMA port, a byte swapping is done (B3B2B1B0 → B0B1B2B3).

The DMA unit used to get the data from the AP is determined comparing the source address with the start address of the M3 memory in the AP memory space. If the source address is above or equal, the Burst DMA unit is used to read the data from the AP. If the source address is below, the Peripheral DMA unit is used.

In case the Burst DMA unit is used, the data is transferred on 32bits. If the number of bytes to be transferred is not a multiple of 4, the last bytes are transferred using 8bit transfers.

In case Per DMA is used, if the source/destination address is not aligned on a 4 byte boundry, there is a byte transfer to have the address aligned. Then a 32bit transfer is performed, finishing with another 8bit transfer in case there are remaining bytes.

When the SDMA has finished to use a buffer (buffer on the host side is empty, or buffer on the dsp side is filled), this buffer is closed, and a new one is opened. And a new transfer loop is executed.

The channel is closed when all the buffers on a side have been used. The script can handle source and destination buffers of different size as described in IPC scripts and use cases section that gives more details about the Last bit feature.

5.2.1.5 Patched version:

Please see 5.2.2.5 for more details.

5.2.1.6 New Feature:

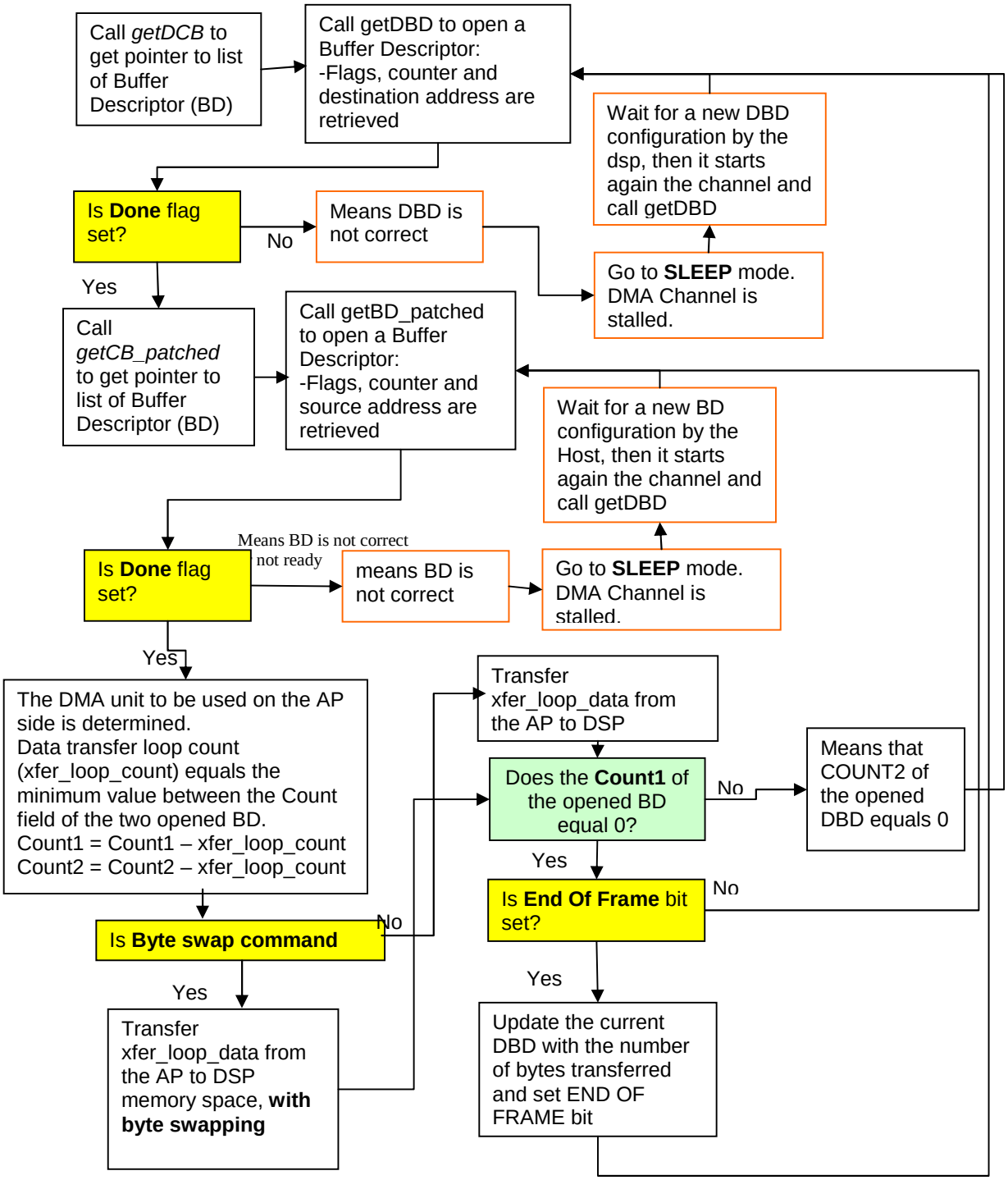
The patched version of this script has been modified to support IPcv2. The purpose of IPCv2 is to transfer Mutiple frame in one transaction from AP memory to BP memory and vice versa.

To achieve this, a new status bit has been included in the Buffer Desctiptors which goes as an input to the script. The new status Bit is called as End Of Frame(F) bit. When F bit is encountered by the script on the source BD, the script shall copy the

F bit in the destination BD and update the count of current BD in destination. After this operation, the script shall request new BDs from both AP and BP if the LAST bit is not set in the current BD of the source.

Script sequences

Return to DMA channels supported by SDMA library



5.2.2 bp_2_ap

This script is used to transfer data from the BP side (DSP memory) to the AP (host).

Using this script there is no restriction on the number of byte to transmit,on the source address and destination address alignment, or on the destination memory type. The buffers are defined on each side, they do not need to have the same size. This script offers the byte swapping feature which can be enable/disable through command passed in the Command field of the Buffer Descriptor on MCU side.

5.2.2.1 Parameters transmit through the context

R7: The AP M3 start address.

5.2.2.2 Parameters transmit trough the Buffer descriptor:

In the buffer descriptor on the host side:

The Command field is used to enable/disable byte swapping feature (command == 0x80: byte swapping ON – Default is OFF).

The number of bytes to transmit is stored in the first Buffer descriptor word

The destination address in the AP side is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used

In the buffer descriptor on the dsp side:

The number of bytes to transmit is stored in the first Buffer descriptor word

The source address in the BP side is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used

5.2.2.3 Use of the General Register during the execution

- R0: counter during loop, base address of the context when needed
- R1: BP bd mode word, count, scratch
- R2: channel cb address, scratch
- R3: current buffer descriptor pointer, scratch
- R4: AP bd mode word, count, scratch
- R5: destination address loaded by getdb, scratch
- R6: base address of context, scratch
- R7: M3 start address

5.2.2.4 Overview of script functionality

The parameters of the transfer on the host side (size of the buffer, destination address) are retrieved from the buffer descriptor and also the Byte swapping flag.

The parameters of the transfer on the dsp side (size of the buffer, source address) are retrieved from the buffer descriptor.

Then a transfer loop is performed based on the size of the smallest buffer. If Byte swapping is ON, each time a 32-bit data is exchanged between Burst DMA and

DSP DMA port, a byte swapping is done (B3B2B1B0 → B0B1B2B3).

When the SDMA has finished to use a buffer (buffer on the dsp side is empty, or buffer on the host side is filled), this buffer is closed, and a new one is opened. And a new transfer loop is executed.

The channel is closed when all the buffers on a side have been used. The script can handle source and destination buffers of different size as described in IPC scripts and use cases section that gives more details about the Last bit feature.

5.2.2.5 Patched version:

Two new scripts called bp_2_ap_patched and ap_2_bp_patched has been added in RAM.These scripts are basically the same as bp_2_ap and ap_2_bp in ROM with S/W fixes and S/w workarounds for H/W bugs which were encountered as testing was done.

These scripts were added in RAM because fixes made in ROM will need a new chip version. The fix in RAM made it possible to fix the error in all chips irrespective of the ROM version

Summary of the changes in the patched scripts are as follows:

1. In the ROM version, SDMA uses Peripheral DMA to get the BD and CCB structures from AP memory. The patched version uses Burst SDMA which is faster and better suited for SDRAM like devices.

The main reason for making the change from Peripheral DMA to Burst DMA is the behaviour in Low Power Mode.Peripheral DMA communicates through the Cross-bar switch.The clock of the Cross-bar switch is off when AP is in wait mode.Thus, Peripheral DMA cannot work in Low Power Mode.Burst DMA on the other hand uses M3if which is not affected in Low Power Mode. Hence, the change.

2. To support the S/W workaround for the DSPDMA synchronisation H/W bug, a polling mechanism has been added to prevent a context switch due to termination of channel when Done bit is not set for a particular BD on the AP side. This has been reflected in the flowchart below.This is applicable only for bp_2_ap_patched.

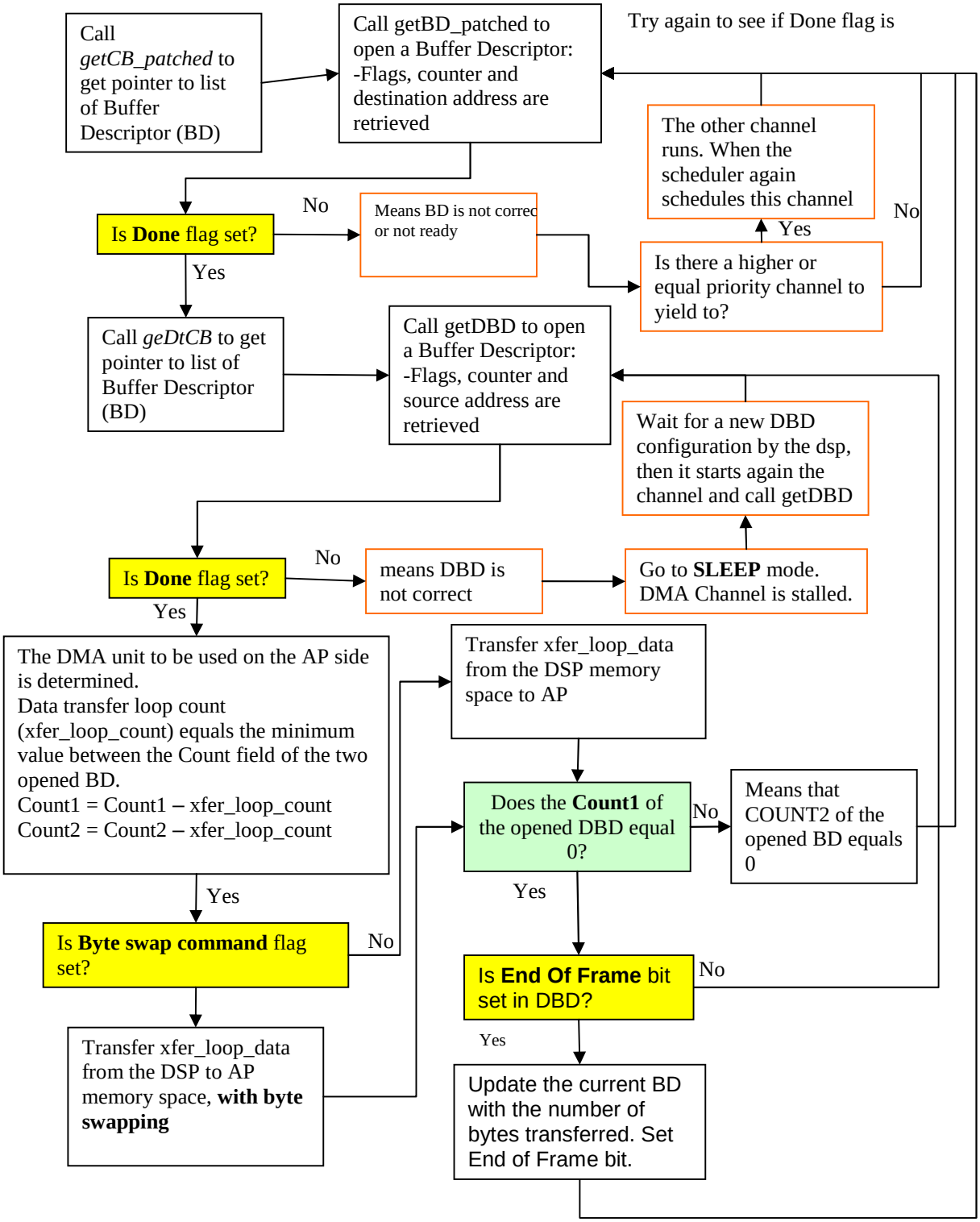
5.2.2.6 New Feature:

The patched version of this script has been modified to support IPcv2. The purpose of IPCv2 is to transfer Mutiple frame in one transaction from AP memory to BP memory and vice versa.

To achieve this, a new status bit has been included in the Buffer Desctiptors which goes as an input to the script. The new status Bit is called as End Of Frame(F) bit. When F bit is encountered by the script on the source BD, the script shall copy the F bit in the destination BD and update the count of current BD in destination. After this operation, the script shall request new BDs from both AP and BP if the LAST bit is not set in the current BD of the source.

Script sequences

Return to DMA channels supported by SDMA library



5.2.3 bp_2_bp

This script is used to transfer data inside the DSP memory space using the DSP DMA port of the SDMA.

Using this script there is no restriction on the number of byte to transmit or on the source address and destination address alignment. But these parameters could impact the performance. In fact the copy capability of the DSP DMA could only be used if the two addresses are aligned; meaning source and destination address must have the same modulo. For instance, a copy mode is used if source address equals 0[4] and destination address equals 0[4]; or source is 1[4] and destination is 1[4]. When it is not the case the SRAM of the SDMA is used as temporary buffer.

5.2.3.1 Parameters transmit through the context:

N/A

5.2.3.2 Parameters transmit trough the Buffer descriptor:

The number of bytes to transmit is stored in the first Buffer descriptor word

The source address is stored in the second Buffer descriptor word (address field).

The destination address is stored in the Extended Buffer address

5.2.3.3 Use of the General Register during the execution

- R0: counter
- R1: scratch, context base
- R2: channel block descriptor address / base address of the scratch buffer (in SDMA's SRAM)
- R3: current buffer descriptor pointer
- R4: mode
- R5: source address loaded by getdcb / scratch
- R6: destination address loaded by getdbd / scratch
- R7: N/A

5.2.3.4 Overview of script functionality

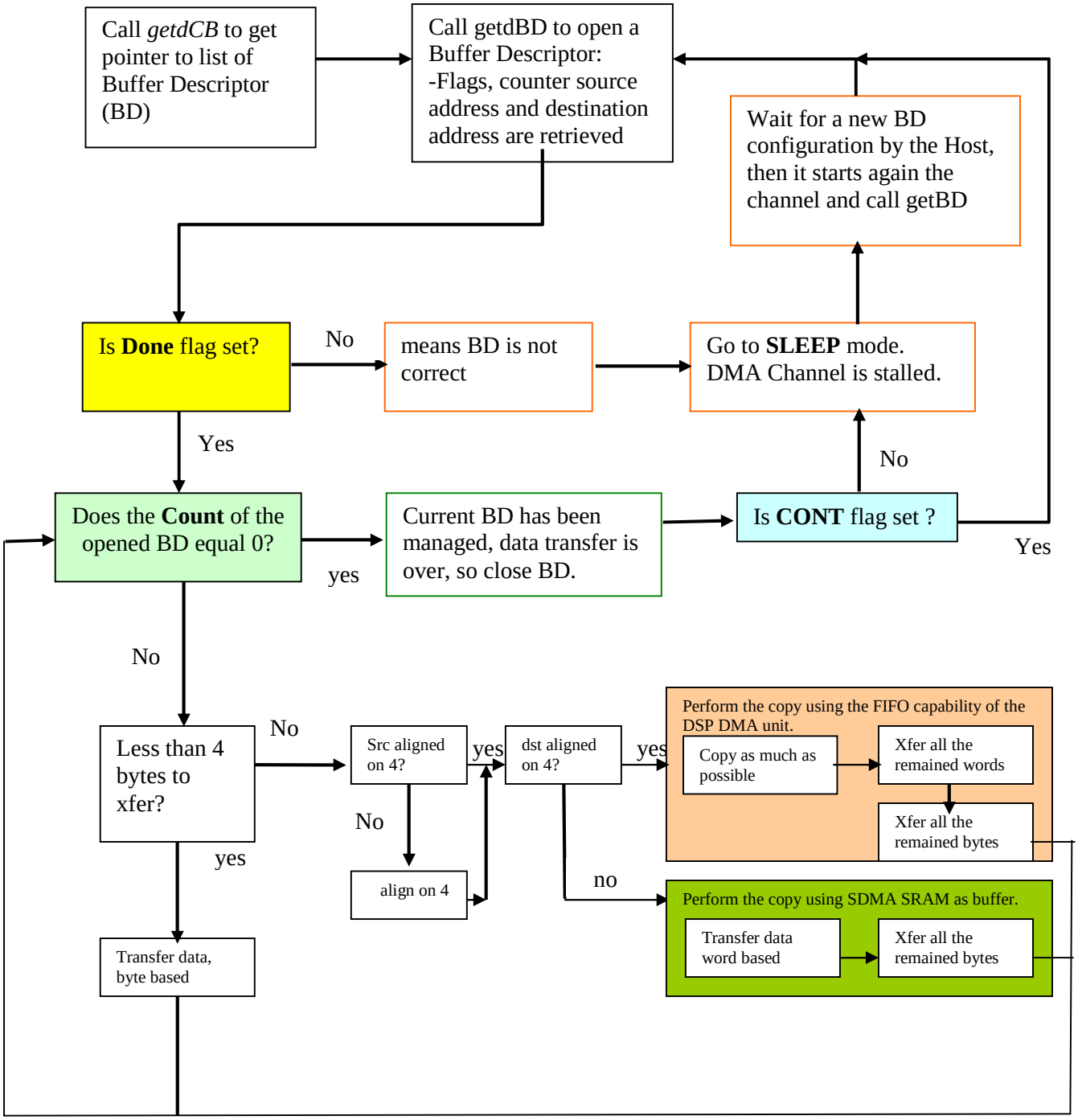
The parameters of the transfer (number of bytes to transfer, source address and destination address) are retrieved from the buffer descriptor.

Depending on the address alignment, the transfer is performed using the FIFO capability of the DSP DMA, or using the SDMA SRAM as a temporary buffer.

When the transfer is finished, this algorithm restarts (a new buffer descriptor is read).

5.2.3.5 Script sequences

Return to DMA channels supported by SDMA library



5.2.4 ap_2_ap

This script is used to transfer data inside the application processor memory using whatever DMA unit or units necessary, depending on the source memory type and destination memory type.

The source and destination memory can be in M2 RAM or M3 RAM. In order to access the M2 RAM, the Peripheral DMA unit must be used, while for the M3 RAM the Burst DMA unit must be used. This results in 4 possible combinations of DMA units to be used, as presented in the table below:

Source address in	Destination address in	DMA units transfer
M2	M2	From Per DMA to Per DMA
M2	M3	From Per DMA to Burst DMA
M3	M2	From Burst DMA to Per DMA
M3	M3	From Burst DMA to Burst DMA

In order for the script to detect the apartenence of source/destination address to M2 or M3, it must have access to the start address of M3 RAM. This value it is passed to the script by context.

Using this script there is no restriction on the number of byte to transmit, the source address and destination address alignment, or on the source memory and destination memory type. But these parameters could impact the performance. In case that the source and destination are in the same memory type, being word aligned allows the SDMA to use the copy capability of the DMA units. When it is not the case the SRAM of the SDMA is used as temporary buffer.

5.2.4.1 Parameters transmit through the context:

R7: AP M3 start address.

5.2.4.2 Parameters transmit trough the Buffer descriptor:

The number of bytes to transmit is stored in the first Buffer descriptor word

The source address is stored in the second Buffer descriptor word (address field).

The destination address is stored in the Extended Buffer address

5.2.4.3 Use of the General Register during the execution

- R0: counter
- R1: scratch
- R2: channel block descriptor address / base address of the scratch buffer (in SDMA's SRAM)
- R3: current buffer descriptor address
- R4: mode
- R5: source address / scratch
- R6: destination address / scratch
- R7: pointer in the scratch buffer

5.2.4.4 Overview of script functionality

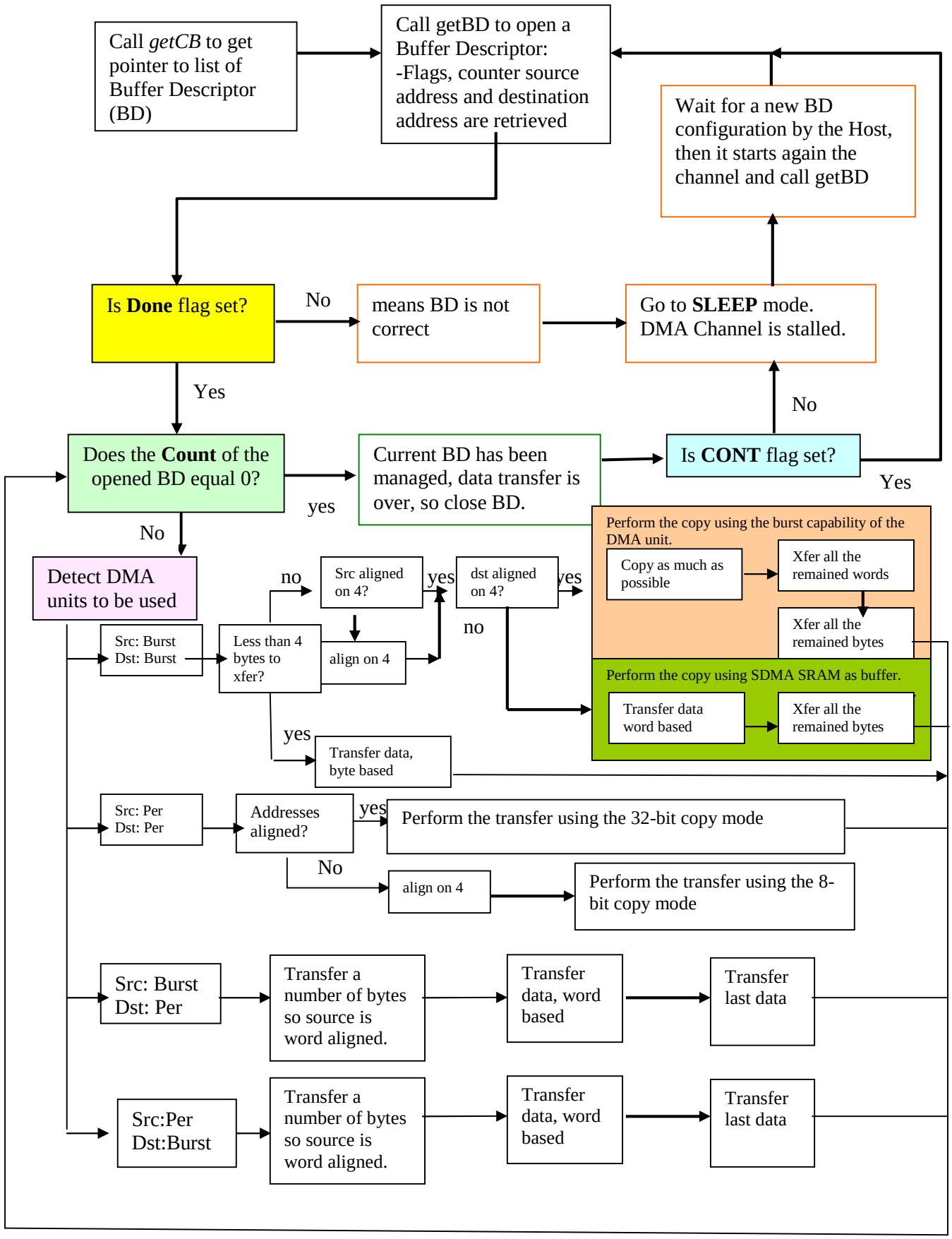
The parameters of the transfer (number of bytes to transfer, source address and destination address) are retrieved from the buffer descriptor.

Depending on the source and destination address memory domain, a combination of DMA units is chosen to perform the transfer.

Depending on the address alignment and the DMA units involved, if possible, the transfer is performed using the copy capability of the DMA unit, or using the SDMA SRAM as a temporary buffer.

When the transfer is finished, this algorithm restarts (a new buffer descriptor is read).

5.2.4.5 Script sequences



5.2.5 app_2_mcu

This generic script is used to transfer data from a 8/16/24 or 32 bits peripheral connected to the ARM platform to memories accessed by the BurstDMA (External memories). It can be used for SSI(8/16/24 or 32bits data size) or for CSPI(32bits data size), these peripherals being connected to the ARM platform peripheral bus.

Note: All the peripherals connected to the ARM platform peripheral bus are considered as 32-bit peripherals in term of connectivity. Therefore all SDMA accesses are 32-bit read accesses. However, the peripheral data size is taken into account for the SDMA write access to the external memory.

5.2.5.1 Parameters transmit through the context

r1: mask to check event – If script is triggered by event I, r1[I] must be set to 1. (Project dependent)

r6: address of the peripheral Rx fifo (project dependent)

r7: Watermark level – Used to determine the maximum of data that can be sent to the peripheral each time the channel is started.

5.2.5.2 Parameters transmit trough the Buffer descriptor

The peripheral size is set in the command field of the first Buffer descriptor word, specially bits 24,25.

The number of bytes to transmit is stored in the first Buffer descriptor word (count field)

The destination address in the M3 ARM memory is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used.

5.2.5.3 Use of the General Register during the execution

- r0: counter during loop, base address of the context when needed
- r1: event mask
- r2: channel control block (CCB) address when needed otherwise scratch
- r3: current buffer descriptor (BD) pointer, base address used to read events
- r4: mode (Command, Flags, count fields of the buffer descriptor)
- r5: destination address (in the external memory) loaded by getdb, number of bytes in buffer otherwise.
- r6: Rx fifo address (when context loaded)
- r7: watermark level

5.2.5.4 Overview of script functionality

The parameters of the transfer (peripheral size, number of bytes to transfer and source address) are retrieved from the buffer descriptor.

When the peripheral sends an event, a data transfer loop is executed. This event is set when there is at least "watermark level" data in the FIFO.

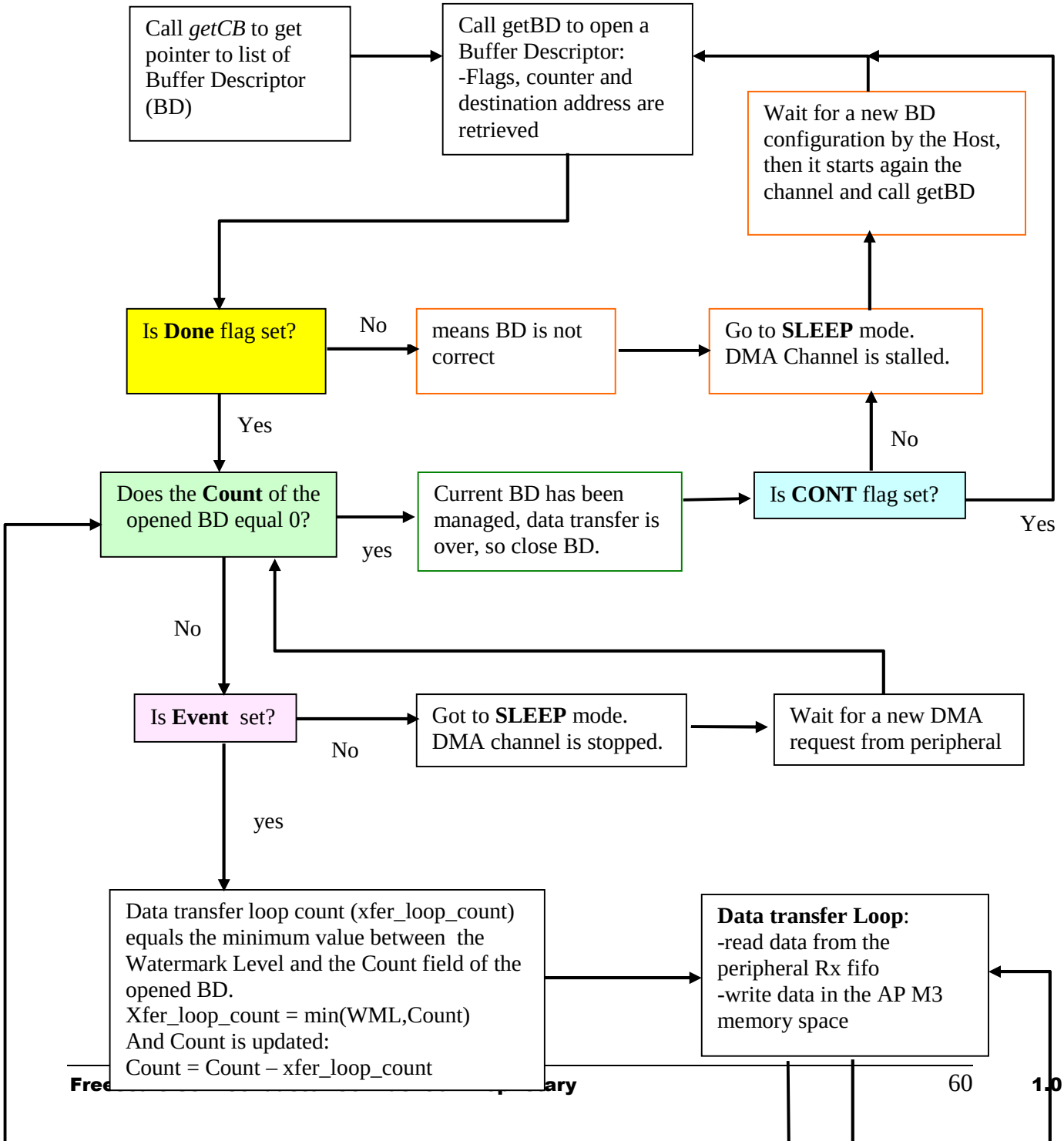
During this data transfer loop, the number of bytes that are transferred from the RX FIFO of the peripheral to the EMI is equal to the minimum value between the "watermark level" and the number of data that have not been transferred yet. For each data transfer (one read, one write), the size of the memory write access is fixed by the parameter peripheral size.

The necessary number of transfer loops are executed in order to fill the buffer, but they are executed only if the event is set. While the event is not set, the channel is not executable.

When the buffer is filled, this algorithm restarts (a new buffer descriptor is read).

5.2.5.5 Script sequences

Return to DMA channels supported by SDMA library



5.2.6 mcu_2_app

This generic script is used to transfer data from memories accessed by the BurstDMA (External memories) to a 8/16/24 or 32 bits peripheral connected to the ARM platform. It can be used for SSI(8/16/24 or 32bits data size), for CSPI(32bits data size) or for UART1,2 (8bits data size), these peripherals being connected to the ARM platform peripheral bus.

Note: All the peripherals connected to the ARM platform peripheral bus are considered as 32-bit peripherals in term of connectivity. Therefore all SDMA accesses to them are 32-bit accesses (in write for this script). However, the peripheral real size, meaning the size of the data stored in their FIFO, is taken into account for the SDMA read access to the external memory. For instance, in the case of the UART, it deals with a 8-bit peripheral; therefore, 8-bit are read from the Burst DMA (that stored data from External memory after the first prefetch); the 8-bit data are 0 extended in the SDMA and then, a 32-bit write access is done to the UART Tx FIFO.

5.2.6.1 Parameters transmit through the context

r1: mask to check event – If script is triggered by event I, r1[I] must be set to 1. (Project dependent)

r6: address of the peripheral Tx fifo (project dependent)

r7: Watermark level – Used to determine the maximum of data that can be retrieved from the peripheral each time the channel is started.

5.2.6.2 Parameters transmit trough the Buffer descriptor

The peripheral size is set in the command field of the first Buffer descriptor word, specially bits 24,25.

The number of bytes to transmit is stored in the first Buffer descriptor word (count field)

The source address in the external memory is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used.

5.2.6.3 Use of the General Register during the execution

- r0: counter during loop, base address of the context when needed
- r1: event_mask
- r2: channel control block (CCB) address when needed otherwise scratch
- r3: current buffer descriptor (BD) pointer, base address used to read events
- r4: mode (Command, Flags, count fields of the buffer descriptor)
- r5: source address (in the external memory) loaded by getdb, number of bytes in buffer otherwise.
- r6: Tx fifo address (when context loaded)
- r7: watermark level

5.2.6.4 Overview of script functionality

The parameters of the transfer (peripheral size, number of bytes to transfer and source address) are retrieved from the buffer descriptor.

When the peripheral sends an event, a data transfer loop is executed. This event is set when there is at least "watermark level" empty rooms in the FIFO.

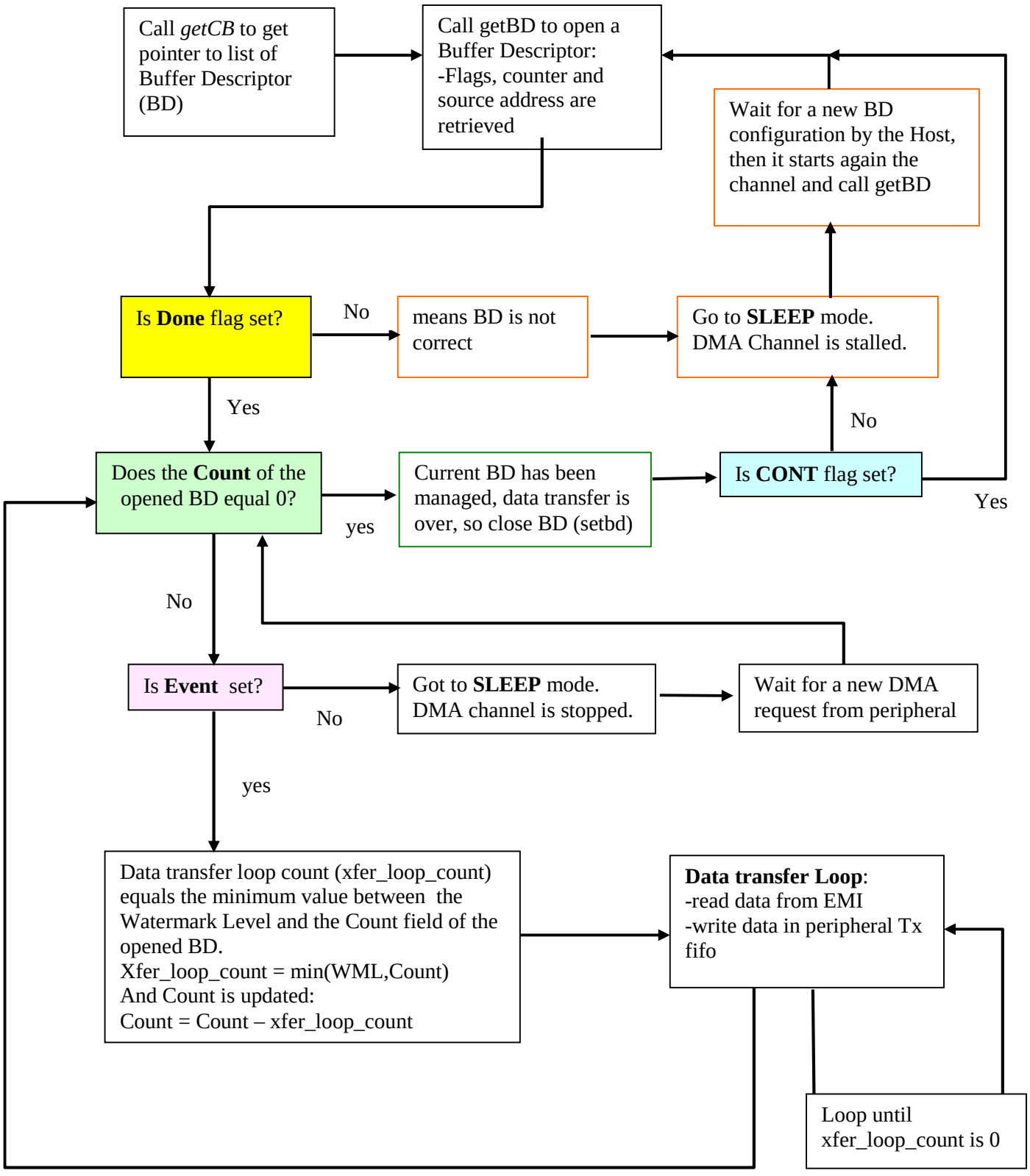
During this data transfer loop, the number of bytes that are transferred from the EMI to the TX FIFO of the peripheral is equal to the minimum value between the "watermark level" and the number of data that have not been transferred yet. For each data transfer (one read, one write), the size of the read access to the external memory is fixed by the parameter peripheral size.

The necessary number of transfer loops are executed in order to empty the buffer, but they are executed only if the event is set. While the event is not set the channel is not executable.

When the buffer is empty, this algorithm restarts (a new buffer descriptor is read).

5.2.6.5 Script sequences

Return to DMA channels supported by SDMA library



5.2.7 uart_2_mcu

This script is used to transfer data from the UART to the External memory. The UART is connected to the ARM platform. The UART is a 8-bit peripheral. But when reading the FIFO 16 bits are retrieved, in fact the 8 MSB are the status bytes for the current data. At each access the value of this byte is checked to verify that a valid data was retrieved. If an error is detected the transfer is stopped and the information is sent back to the Host through the buffer descriptor whith the byte count field updated.

5.2.7.1 Parameters transmit through the context

R1: mask to check event

R6: address of the UART RX FIFO

R7: Watermark level

5.2.7.2 Parameters transmit trough the Buffer descriptor

The number of bytes to transmit is stored in the first Buffer descriptor word

The destination address is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used

5.2.7.3 Use of the General Register during the execution

- R0: counter
- R1: mask to check event
- R2: channel block descriptor address
- R3: current buffer descriptor address
- R4: mode
- R5: destination address / scratch
- R6: source address / scratch
- R7: watermark level

5.2.7.4 Overview of script functionality

When the UART sends a DMA request, it can be for three different reasons:

1- When the RX threshold is overpassed. It deals with the most frequent event.

2- When AGEING timer reached its final value, meaning there are less data than the RX threshold in the RX FIFO and they are present since a too long time. It is called the ageing dma request.

3- When IDLE timer reached its final value; meaning the RX FIFO is empty since a too long time. It is called the iddle dma request. The IDLE timer is not used, so only normal and AGEING DMA requests are supported by the script.

The first time the script is executed, a buffer descriptor is opened, the buffer destination address and its size are retreived from the BD.
Then the script checks if RRDY bit of USR1 is set. If yes, it means that the Watermark level has been reached, there are “WML” bytes in RX FIFO. If the count field of the opend BD is higher than WML-1, the SDMA script will perform

WML-1 read accesses to the RX fifo and WML write access to the destination buffer (a read access then a write access is call a data transfer loop). If the count field is lesser than the WML-1 value, the count field will be the data transfer loop size. So the data transfer loop size equals min(WML-1,BD count)

So, there will be always one data remaining in the UART RX fifo after every data transfer loop and the ageing timer will always trigger a DMA request. It means that the channel on which the script is run will be always closed on an AGEING DMA request.

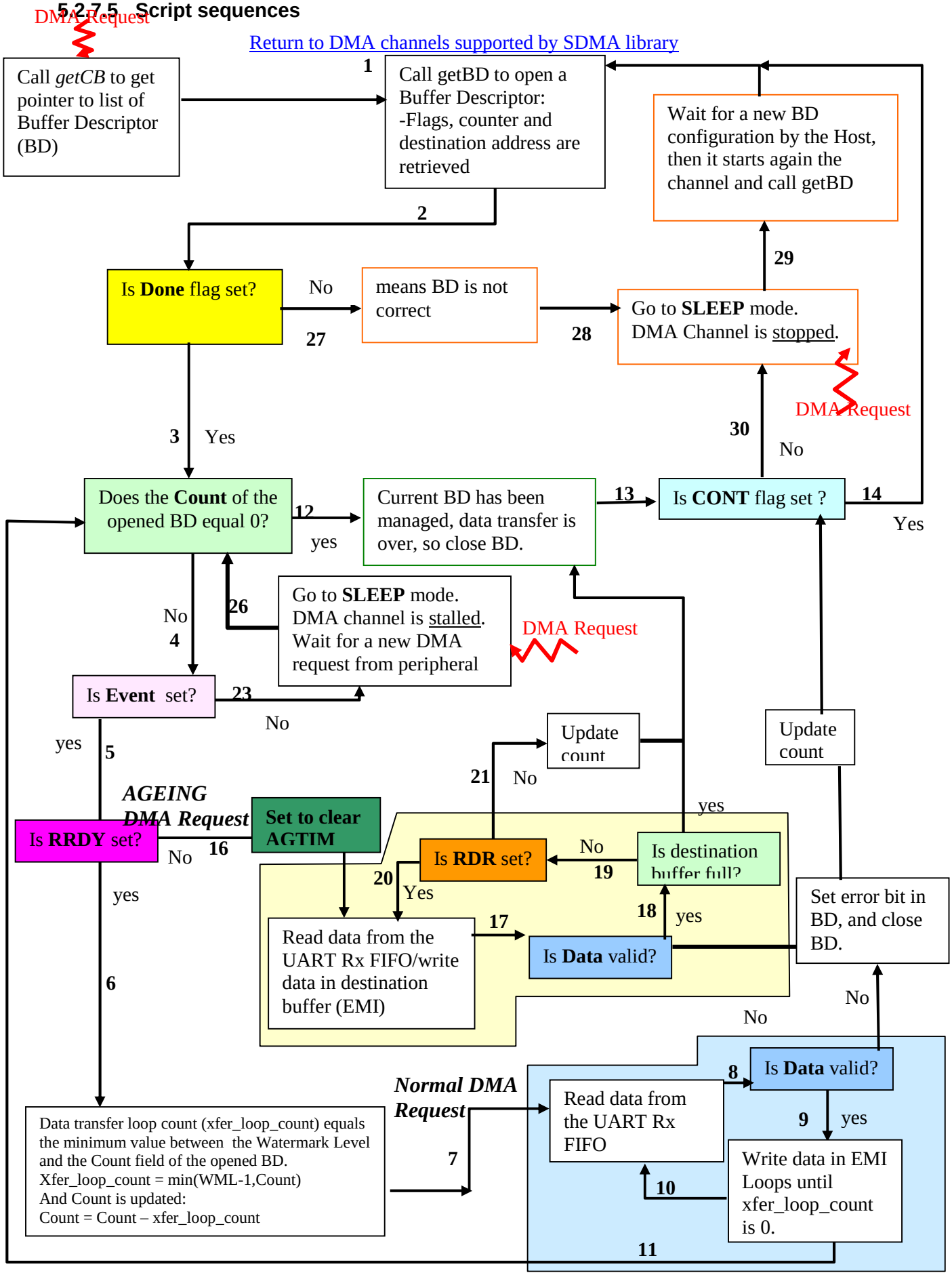
Now, let's assume the script is triggered by a DMA request, but RRDY is not set. It means that it deals with an AGEING DMA request; there is at least one data in the RX FIFO. The data is read and written into the destination buffer, and then the RDR is checked again. If it is set and if the destination data buffer is not full (BD count is not 0), a new data is read and so on until RDR is 0. When RDR is 0, the ageing timer interrupt flag is disabled by setting the AGTIM bit, the count field of BD is updated and the current BD is closed. Then the script tries to open the next BD if the Continuous bit of the current BD was set, if there is no more BD to open, the channel is stopped.

When reading a data from the UART RX FIFO, the MSB are check to verify that a valid data has been retrieved. If an error is detected the transfer is stopped, the error bit is set in the BD, the count is updated, the BD is closed and an interrupt is sent to the ARM.

When the buffer is full, this algorithm restarts (a new buffer descriptor is read, if it exists).

The next diagram illustrates the script algorithm, it is quite complex but things become more obvious when we notice that there is a loop for the normal DMA request (in sky-blue) and on when the AGEING DMA request was detected (in yellow).

5.2.7.5 Script sequences



5.2.8 uartsh_2_mcu

This script is based on the uart_2_mcu script, but it is targeted to the shared UART located on the shared peripheral bus. The shared UART is accessed through the sysbus of the SDMA and through the spba, the destination buffer is located in the external memory. The Burst DMA port of the SDMA is used.

5.2.9 mcu_2_shp

This generic script is used to transfer data from memories accessed by the BurstDMA (External memories) to a 8/16/24 or 32 bits peripheral connected to the shared peripheral bus accessed through the SPBA. It can be used for SSI(8/16/24 or 32bits data size), for CSPI(32bits data size), for SDHC(MMC)/SIM (16bits data size) or for UART3,4 (8bits data size), these peripherals being connected to the shared peripheral bus.

5.2.9.1 Parameters transmit through the context

r1: mask to check event – If script is triggered by event I, r1[I] must be set to 1. (Project dependent)

r6: address of the peripheral Tx fifo (project dependent)

r7: Watermark level – Used to determine the maximum of data that can be retrieved from the peripheral each time the channel is started.

5.2.9.2 Parameters transmit trough the Buffer descriptor

The peripheral size is set in the command field of the first Buffer descriptor word, specially bits 24,25.

The number of bytes to transmit is stored in the first Buffer descriptor word (count field)

The source address in the external memory is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used.

5.2.9.3 Use of the General Register during the execution

- r0: counter during loop, base address of the context when needed
- r1: event_mask
- r2: channel control block (CCB) address when needed otherwise scratch
- r3: current buffer descriptor (BD) pointer, base address used to read events
- r4: mode (Command, Flags, count fields of the buffer descriptor)
- r5: source address (in the external memory) loaded by getdb, number of bytes in buffer otherwise.
- r6: Tx fifo address (when context loaded)
- r7: watermark level

5.2.9.4 Overview of script functionality

The parameters of the transfer (peripheral size, number of bytes to transfer and

source address) are retrieved from the buffer descriptor.

When the peripheral sends an event, a data transfer loop is executed. This event is set when there is at least "watermark level" empty rooms in the FIFO.

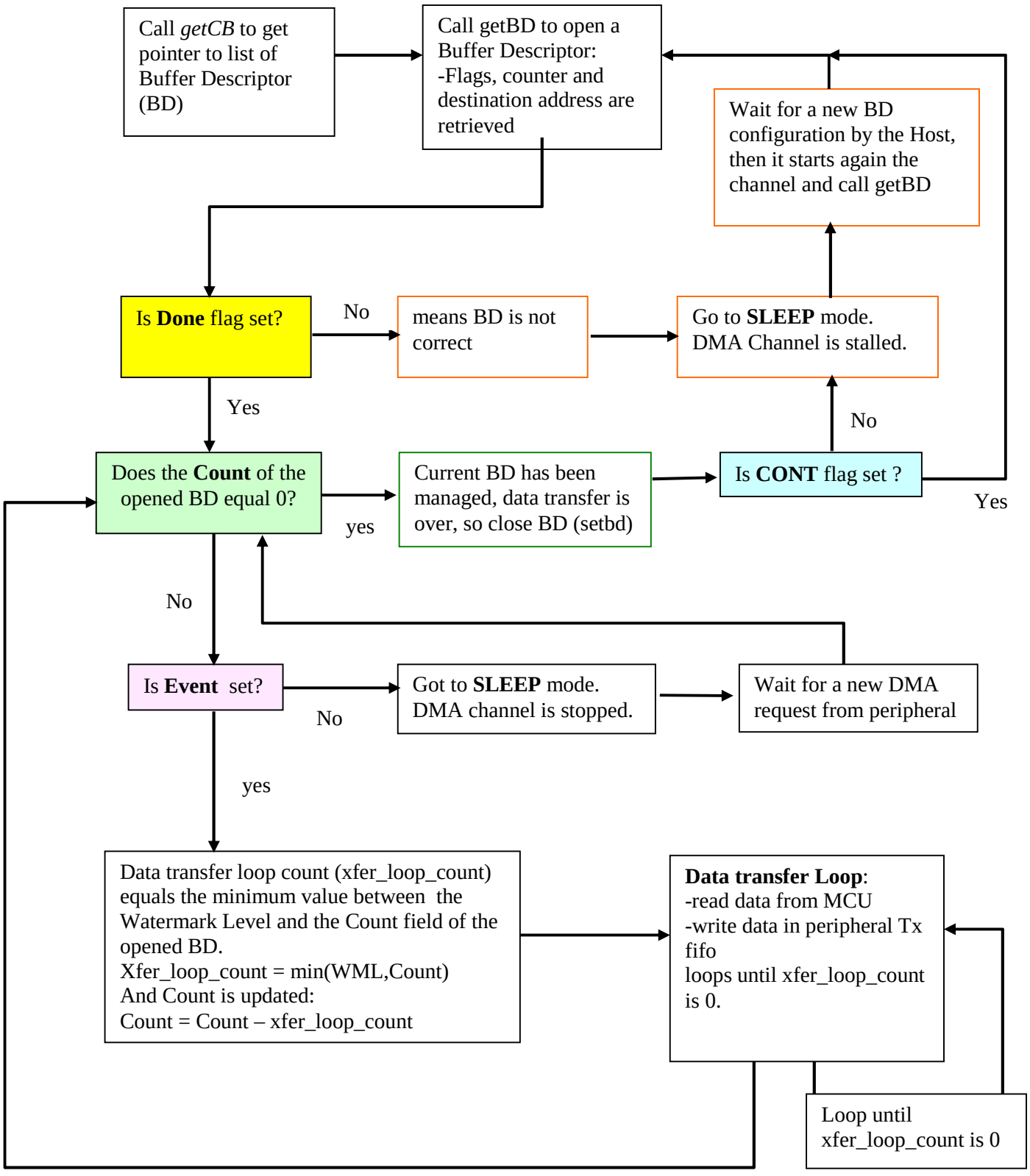
During this data transfer loop, the number of bytes that are transferred from the EMI to the TX FIFO of the peripheral is equal to the minimum value between the "watermark level" and the number of data that have not been transferred yet. For each data transfer (one read, one write), the size of the memory read access is fixed by the parameter peripheral size.

The necessary number of transfer loops are executed in order to empty the buffer, but they are executed only if the event is set. While the event is not set, the channel is not executable.

When the buffer is empty, this algorithm restarts (a new buffer descriptor is read).

5.2.9.5 Script sequences

Return to DMA channels supported by SDMA library



5.2.10 shp_2_mcu

This generic script is used to transfer data from a 8/16/24 or 32 bits peripheral connected to the shared peripheral bus accessed through the SPBA to memories accessed by the BurstDMA (External memories). It can be used for SSI(8/16/24 or 32bits data size), for CSPI(32bits data size) or for SDHC(MMC)/SIM (16bits data size), these peripherals being connected to the shared peripheral bus.

5.2.10.1 Parameters transmit through the context

r1: mask to check event – If script is triggered by event I, r1[I] must be set to 1. (Project dependent)

r6: address of the peripheral Rx fifo (project dependent)

r7: Watermark level – Used to determine the maximum of data that can be sent to the peripheral each time the channel is started.

5.2.10.2 Parameters transmit trough the Buffer descriptor

The peripheral size is set in the command field of the first Buffer descriptor word, specially bits 24,25.

The number of bytes to transmit is stored in the first Buffer descriptor word (count field)

The destination address in the external memory is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used.

5.2.10.3 Use of the General Register during the execution

- r0: counter during loop, base address of the context when needed
- r1: event_mask
- r2: channel control block (CCB) address when needed otherwise scratch
- r3: current buffer descriptor (BD) pointer, base address used to read events
- r4: mode (Command, Flags, count fields of the buffer descriptor)
- r5: destination address (in the external memory) loaded by getdb, number of bytes in buffer otherwise.
- r6: Rx fifo address (when context loaded)
- r7: watermark level

5.2.10.4 Overview of script functionality

The parameters of the transfer (peripheral size, number of bytes to transfer and source address) are retrieved from the buffer descriptor.

When the peripheral sends an event, a data transfer loop is executed. This event is set when there is at least "watermark level" data in the FIFO.

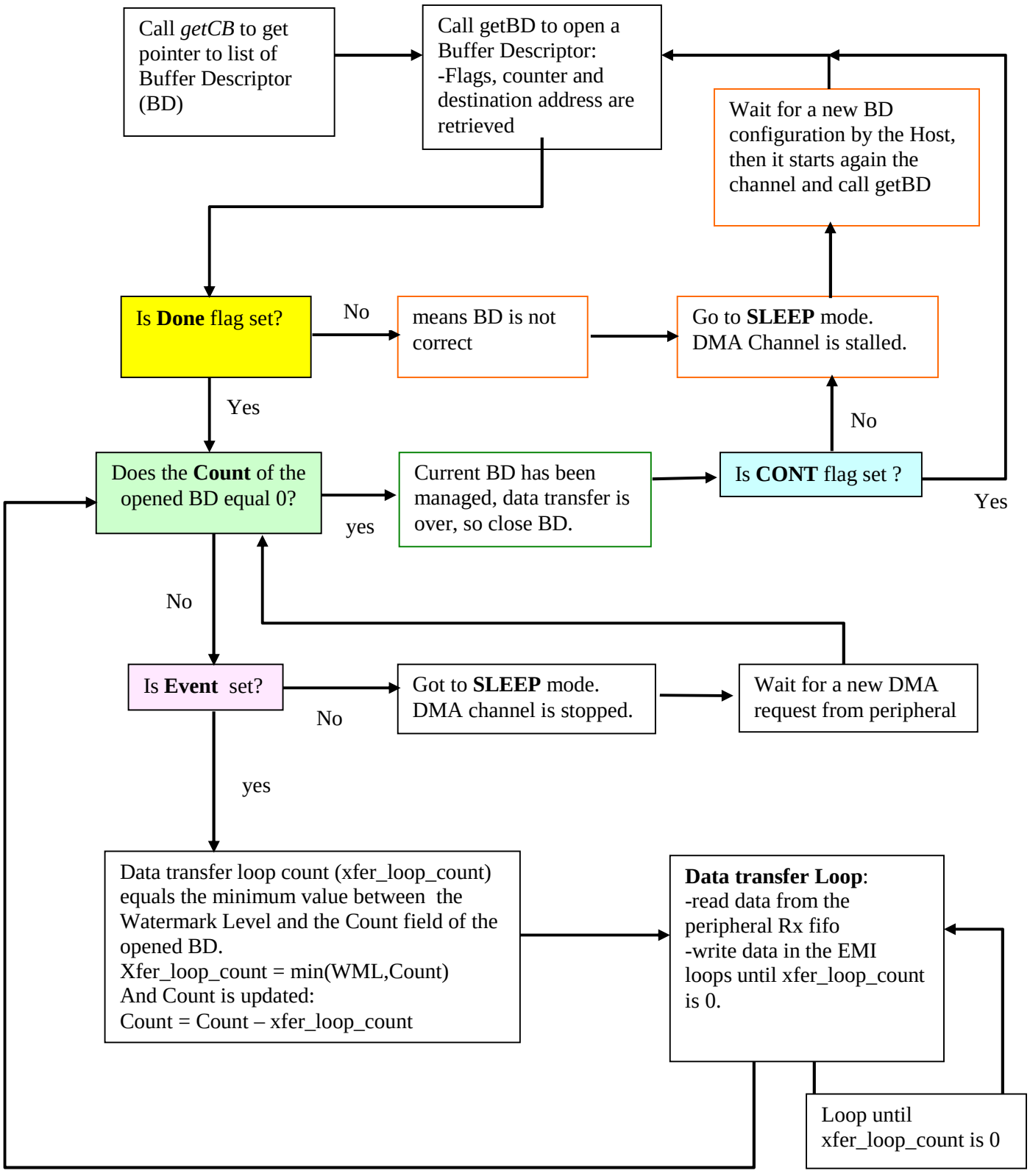
During this data transfer loop, the number of bytes that are transferred from the RX FIFO of the peripheral to the EMI is equal to the minimum value between the "watermark level" and the number of data that have not been transferred yet. For each data transfer (one read, one write), the size of the memory write access is fixed by the parameter peripheral size.

The necessary number of transfer loops are executed in order to fill the buffer, but they are executed only if the event is set. While the event is not set, the channel is not executable.

When the buffer is filled, this algorithm restarts (a new buffer descriptor is read).

5.2.10.5 Script sequences

Return to DMA channels supported by SDMA library



5.3 SDMA RAM scripts

In this section, are listed and detailed all scripts which have to be loaded in the SDMA RAM. They are divided into 3 categories: memory to peripheral, peripheral to memory, others.

In the other subsection, there is only the dptc_dvfs script which is sending measurements to the Clock Controller Module according to a software buffer input.

All necessary memory to memory scripts are located in ROM; except platforms built with the ROMpass1 and requiring some byte swapping.

5.3.1 Memory to peripheral

In this part, all scripts to do transfers from a memory (AP or BP side) to a peripheral (on AP bus or on shared bus) are detailed below.

5.3.1.1 mcu_2_app

This script is loaded only in the SDMA RAM for platforms built with the ROMpass1. See details in the SDMA ROM v2 section mcu_2_app.

5.3.1.2 per_2_app

This generic script is used to transfer data from some memory space accessed with the PeripheralDMA (Host internal or external memory) to a 8/16/24 or 32 bits peripheral. connected to the ARM platform. It can be used for SSI(8/16/24 or 32bits data size), for CSPI(32bits data size) or for UART1,2 (8bits data size), these peripherals being connected to the ARM platform peripheral bus.

Although this script can be used to access the external memory, it is less efficient (in term of data throughput) than the mcu_2_app script.

Note: All the peripherals connected to the ARM platform peripheral bus are considered as 32-bit peripherals in term of connectivity. Therefore all SDMA accesses to them are 32-bit accesses (in write for this script). However, the peripheral real size, meaning the size of the data stored in their FIFO, is taken into account for the SDMA read access through the PeripheralDMA.

5.3.1.2.1 Parameters transmit through the context

r1: mask to check event – If script is triggered by event I, r1[I] must be set to 1. (Project dependent)

r6: address of the peripheral Tx fifo (project dependent)

r7: Watermark level – Used to determine the maximum of data that can be retrieved from the peripheral each time the channel is started.

5.3.1.2.2 Parameters transmit trough the Buffer descriptor

The peripheral size is set in the command field of the first Buffer descriptor word, specially bits 24,25.

The number of bytes to transmit is stored in the first Buffer descriptor word (count field)

The source address in the Host memory space is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used.

5.3.1.2.3 Use of the General Register during the execution

- r0: counter during loop, base address of the context when needed
- r1: event_mask
- r2: channel control block (CCB) address when needed otherwise scratch
- r3: current buffer descriptor (BD) pointer, base address used to read events
- r4: mode (Command, Flags, count fields of the buffer descriptor)
- r5: source address (in the external memory) loaded by getdb, number of bytes in buffer otherwise.
- r6: Tx fifo address (when context loaded)
- r7: watermark level

5.3.1.2.4 Overview of script functionality

The parameters of the transfer (peripheral size, number of bytes to transfer and source address) are retrieved from the buffer descriptor.

When the peripheral sends an event, a data transfer loop is executed. This event is set when there is at least "watermark level" empty rooms in the FIFO.

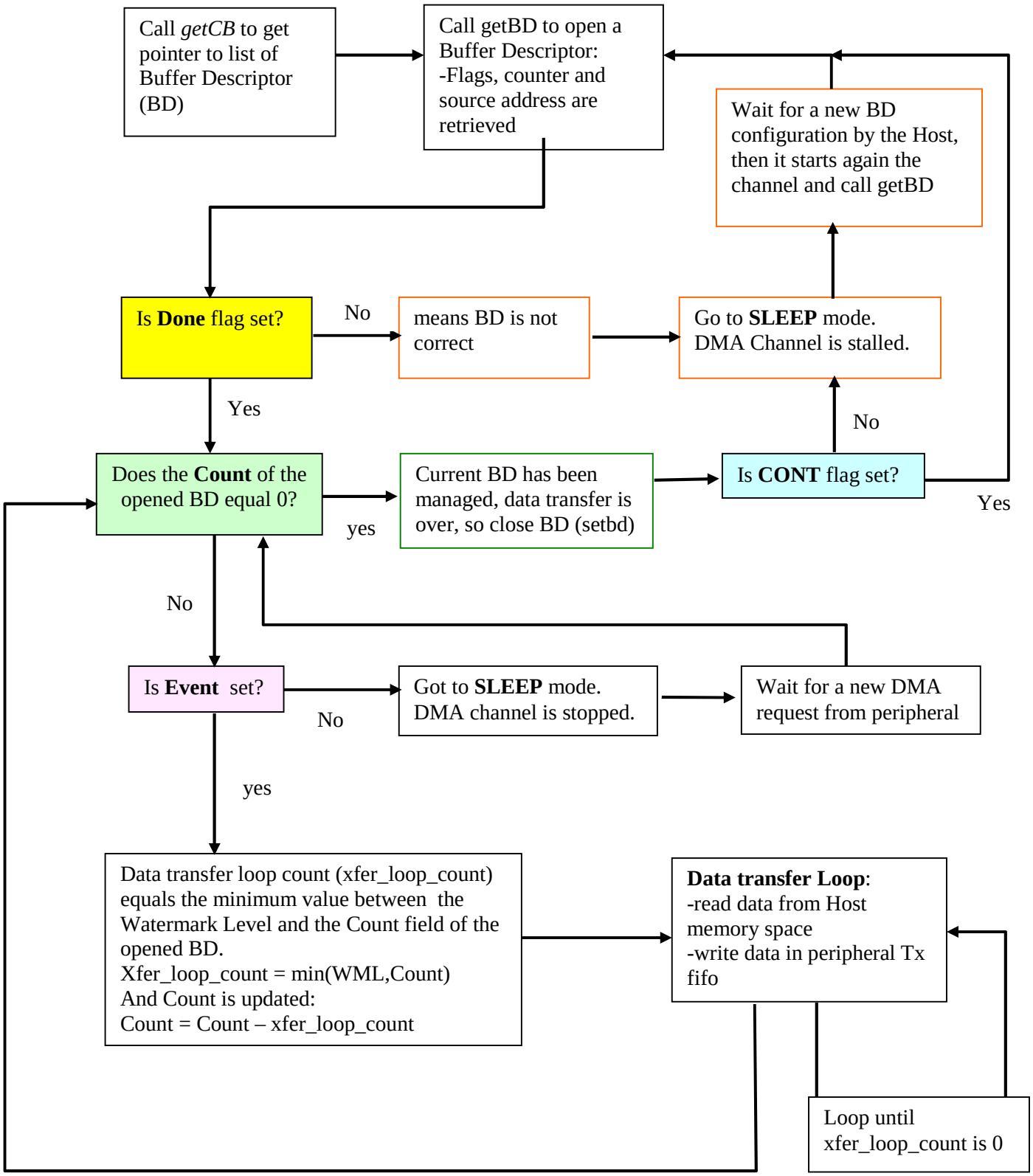
During this data transfer loop, the number of bytes that are transferred from the memory space accessed through the peripheral dma to the TX FIFO of the peripheral is equal to the minimum value between the "watermark level" and the number of data that have not been transferred yet. For each data transfer (one read, one write), the size of the memory read access is fixed by the parameter peripheral size.

The necessary number of transfer loops are executed in order to empty the buffer, but they are executed only if the event is set. While the event is not set, the channel is not executable.

When the buffer is empty, this algorithm restarts (a new buffer descriptor is read).

5.3.1.2.5 Script sequences

Return to DMA channels supported by SDMA library



5.3.1.3 mcu_2_firi

This script is used to transfer data to the FIRI from the external memory. The FIRI is an 8 bits peripheral but the FIRI has the capability to pack several bytes (32 bits access and 16 bits access to the FIRI are supported by the script).

5.3.1.3.1 Parameters transmit through the context

r1: mask to check event – If script is triggered by event I, r1[I] must be set to 1. (Project dependent)

r6: address of the FIRI TX FIFO (project dependent)

r7: Watermark level – Used to determine the maximum of data that can be retrieved from the FIRI each time the channel is started.

5.3.1.3.2 Parameters transmit through the Buffer descriptor

The number of bytes to transmit is stored in the first Buffer descriptor word (count field)

The source address in the external memory is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used.

5.3.1.3.3 Use of the General Register during the execution

- r0: counter during loop, base address of the context when needed
- r1: event_mask
- r2: channel control block (CCB) address when needed otherwise scratch
- r3: current buffer descriptor (BD) pointer, base address used to read events
- r4: mode (Command, Flags, count fields of the buffer descriptor)
- r5: source address (in the external memory) loaded by getdb, number of bytes in buffer otherwise.
- r6: FIRI TX fifo address (when context loaded)
- r7: watermark level

5.3.1.3.4 Overview of script functionality

The parameters of the transfer (number of bytes to transfer and source address) are retrieved from the buffer descriptor.

When the FIRI sends an event, a data transfer loop is executed. This event is set when there is at least "watermark level" empty rooms in the FIFO.

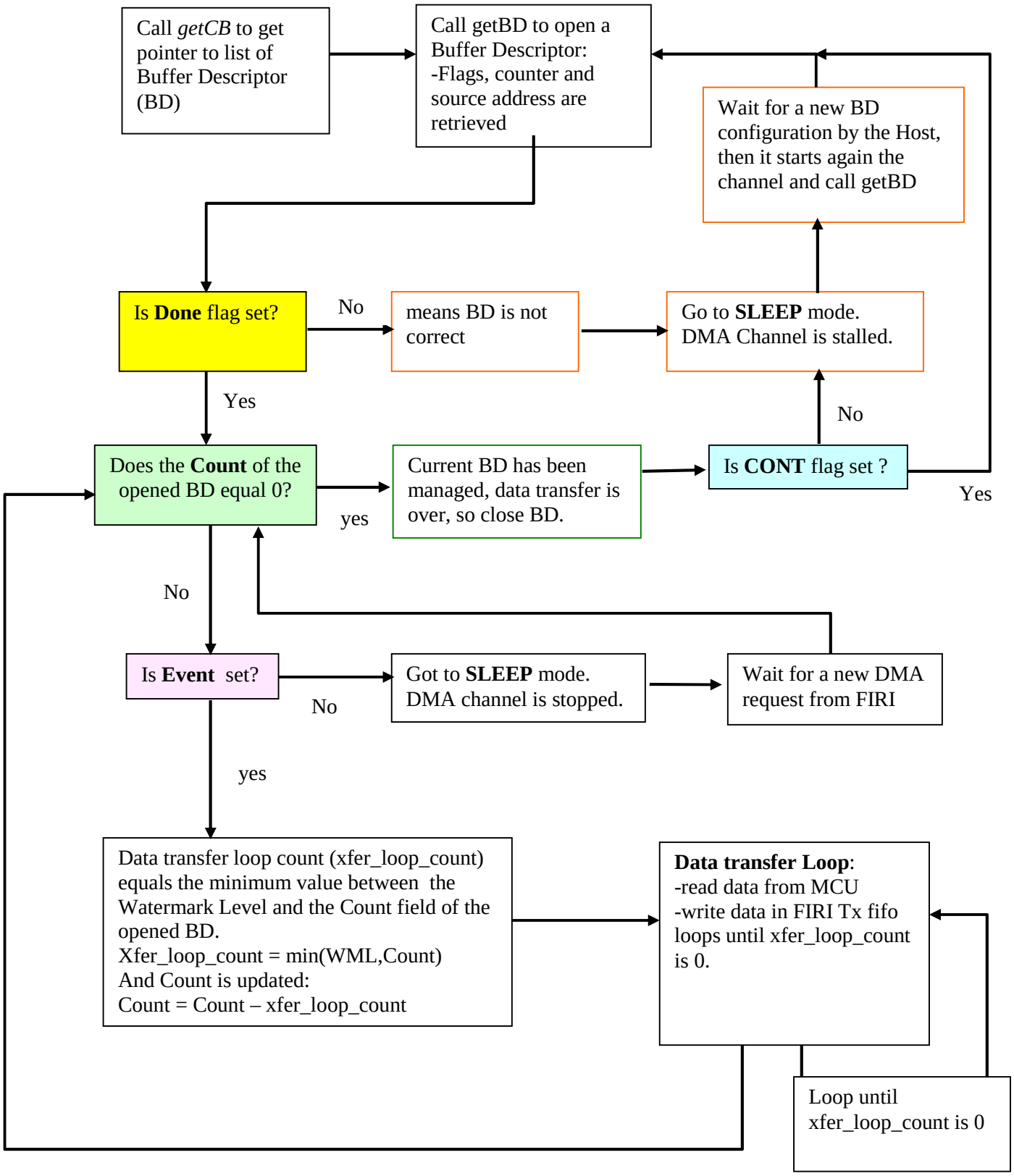
During this data transfer loop, the number of bytes that are transferred from the EMI to the TX FIFO of the FIRI is equal to the minimum value between the "watermark level" and the number of data that have not been transferred yet.

The necessary number of transfer loops is executed in order to empty the buffer, but they are executed only if the event is set. When the event is not set the channel is not run able.

When the buffer is empty, this algorithm restarts (a new buffer descriptor is read).

5.3.1.3.5 Script sequences

Return to DMA channels supported by SDMA library



5.3.1.4 per_2_firi

This script is used to transfer data to the FIRI from the Host memory space. The FIRI is a 8 bits peripheral but the FIRI has the capability to pack several bytes (32 bits access and 16 bits access to the FIRI are supported by the script).

5.3.1.4.1 Parameters transmit through the context

r1: mask to check event – If script is triggered by event I, r1[I] must be set to 1. (project dependent)

r6: address of the FIRI TX FIFO (project dependent)

r7: Watermark level – Used to determine the maximum of data that can be retrieved from the FIRI each time the channel is started.

5.3.1.4.2 Parameters transmit through the Buffer descriptor

The number of bytes to transmit is stored in the first Buffer descriptor word (count field)

The source address in the host memory space is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used.

5.3.1.4.3 Use of the General Register during the execution

- r0: counter during loop, base address of the context when needed
- r1: event_mask
- r2: channel control block (CCB) address when needed otherwise scratch
- r3: current buffer descriptor (BD) pointer, base address used to read events
- r4: mode (Command, Flags, count fields of the buffer descriptor)
- r5: source address (in the host memory space) loaded by getdb, number of bytes in buffer otherwise.
- r6: FIRI TX fifo address (when context loaded)
- r7: watermark level

5.3.1.4.4 Overview of script functionality

The parameters of the transfer (number of bytes to transfer and source address) are retrieved from the buffer descriptor.

When the FIRI sends an event, a data transfer loop is executed. This event is set when there is at least "watermark level" empty rooms in the FIFO.

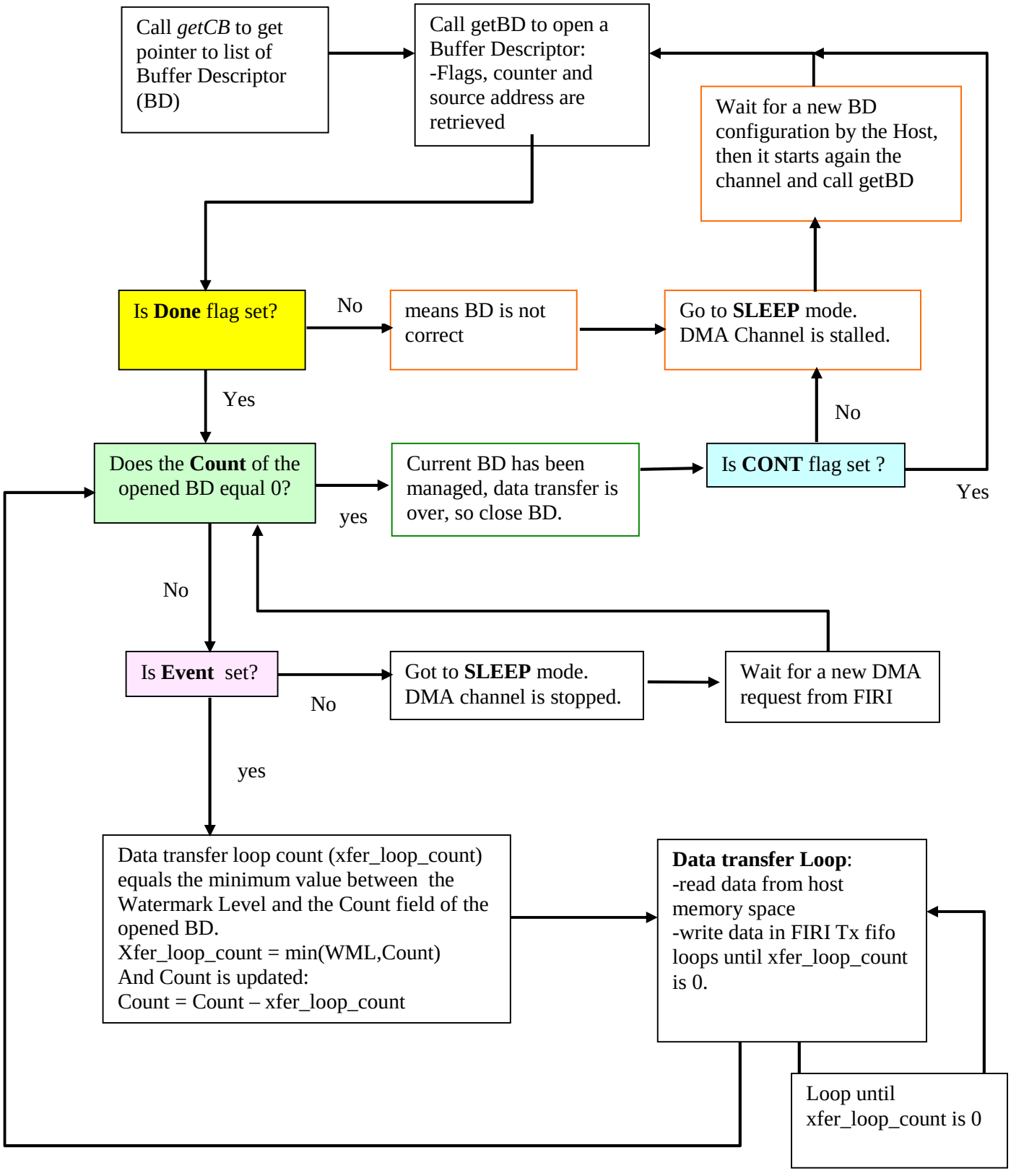
During this data transfer loop, the number of bytes that are transferred from the host memory space to the TX FIFO of the FIRI is equal to the minimum value between the "watermark level" and the number of data that have not been transferred yet.

The necessary number of transfer loops is executed in order to empty the buffer, but they are executed only if the event is set. When the event is not set the channel is not runnable.

When the buffer is empty, this algorithm restarts (a new buffer descriptor is read).

5.3.1.4.5 Script sequences

Return to DMA channels supported by SDMA library



5.3.1.5 mcu_2_shp

This script is loaded only in the SDMA RAM for platforms built with the ROMpass1. See details in the SDMA ROM v2 section mcu_2_shp.

5.3.1.6 per_2_shp

This generic script is used to transfer data from memories accessed by the PeripheralDMA (Host internal or External memories) to a 8/16/24 or 32 bits peripheral connected to the shared peripheral bus accessed through the SPBA. It can be used for SSI(8/16/24 or 32bits data size), for CSPI(32bits data size), for SDHC(MMC)/SIM (16bits data size) or for UART3,4 (8bits data size), these peripherals being connected to the shared peripheral bus.

Although this script can be used to access the external memory, it is less efficient (in term of data throughput) than the mcu_2_app script.

5.3.1.6.1 Parameters transmit through the context

r1: mask to check event – If script is triggered by event I, r1[I] must be set to 1. (Project dependent)

r6: address of the peripheral Tx fifo (project dependent)

r7: Watermark level – Used to determine the maximum of data that can be retrieved from the peripheral each time the channel is started.

5.3.1.6.2 Parameters transmit trough the Buffer descriptor

The peripheral size is set in the command field of the first Buffer descriptor word, specially bits 24,25.

The number of bytes to transmit is stored in the first Buffer descriptor word (count field)

The source address in the host memory space memory is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used.

5.3.1.6.3 Use of the General Register during the execution

- r0: counter during loop, base address of the context when needed
- r1: event_mask
- r2: channel control block (CCB) address when needed otherwise scratch
- r3: current buffer descriptor (BD) pointer, base address used to read events
- r4: mode (Command, Flags, count fields of the buffer descriptor)
- r5: source address (in the external memory) loaded by getdb, number of bytes in buffer otherwise.
- r6: Tx fifo address (when context loaded)
- r7: watermark level

5.3.1.6.4 Overview of script functionality

The parameters of the transfer (peripheral size, number of bytes to transfer and source address) are retrieved from the buffer descriptor.

When the peripheral sends an event, a data transfer loop is executed. This event is set when there is at least "watermark level" empty rooms in the FIFO.

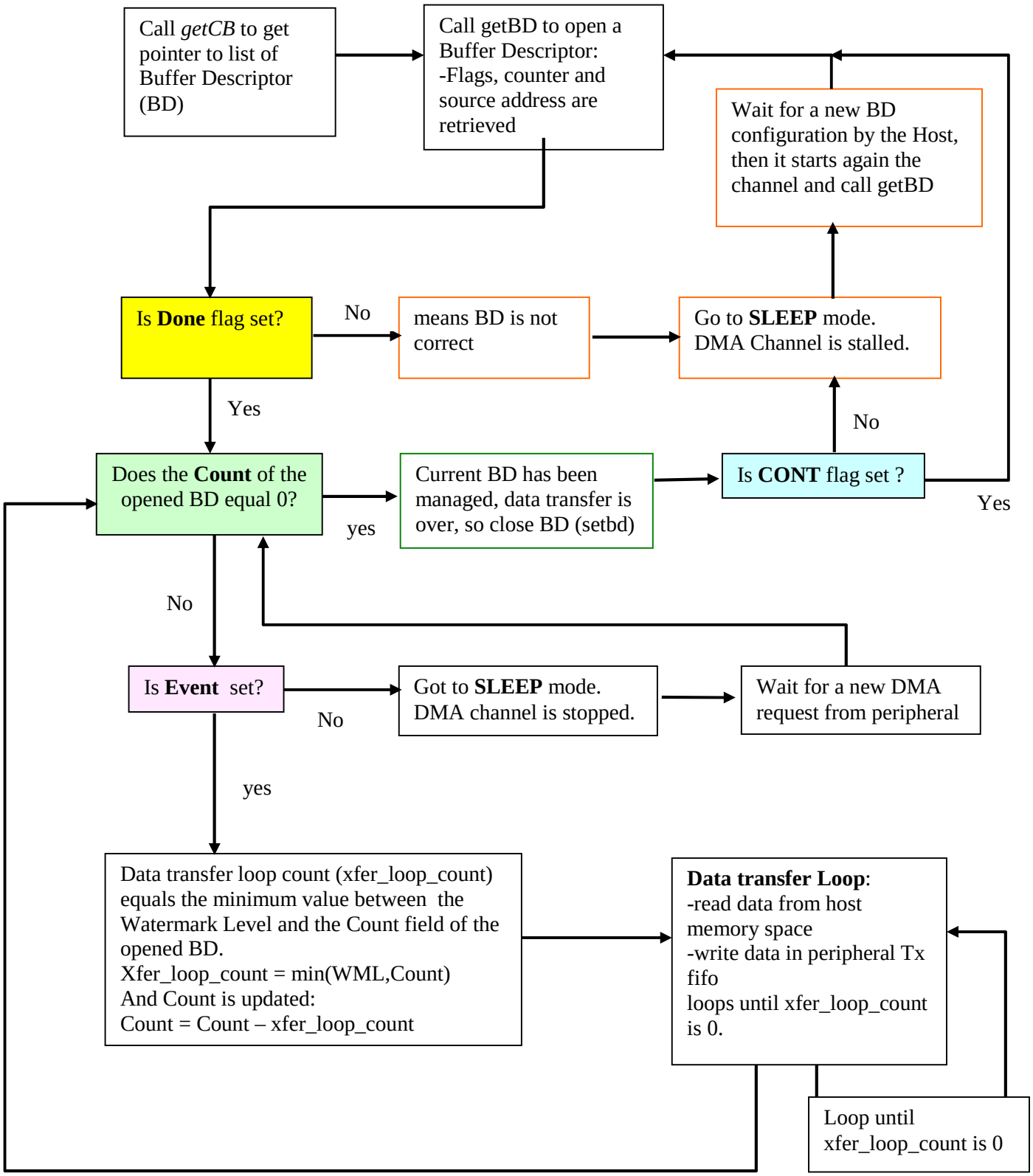
During this data transfer loop, the number of bytes that are transferred from a memory accessed through the PeripheralDMA to the TX FIFO of the peripheral is equal to the minimum value between the "watermark level" and the number of data that have not been transferred yet. For each data transfer (one read, one write), the size of the memory read access is fixed by the parameter peripheral size.

The necessary number of transfer loops are executed in order to empty the buffer, but they are executed only if the event is set. While the event is not set, the channel is not executable.

When the buffer is empty, this algorithm restarts (a new buffer descriptor is read).

5.3.1.6.5 Script sequences

Return to DMA channels supported by SDMA library



5.3.1.7 mcu_2_ata

This script is used to transfer data to the ATA peripheral from the external memory. The ATA is a 32 bits peripheral but it can be accessed in 16 bits. This peripheral has two DMA requests, one is dedicated to flag that the Tx fifo watermark has been under passed (fifo_Tx_alarm), the second warns when a end of transfer occurs (fifo_txfer_end_alarm).

5.3.1.7.1 Parameters transmit through the context

r0: mask to check the fifo_Tx_alarm event, meaning the event (a.k.a DMA request) that is set by the ATA when its Tx fifo under passed its watermark.

r1: mask to check the fifo_txfer_end_alarm event

r6: address of the ATA TX FIFO

r7: watermark level of the Tx fifo (WML)

5.3.1.7.2 Parameters transmit trough the Buffer descriptor

The number of bytes to transmit is stored in the first Buffer descriptor word

The source address in the external memory is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used

5.3.1.7.3 Use of the General Register during the execution

- r0: counter during loop / mask for the Tx fifo event
- r1: mask for the end of transmit event
- r2: channel control block descriptor address /scratch
- r3: current buffer descriptor address / mask for the Tx fifo event
- r4: mode field of BD / buffer size (BD count field)
- r5: source address (in external memory) / base address for the event register
- r6: Tx fifo address of the ATA
- r7: watermark level

5.3.1.7.4 Overview of script functionality

The parameters of the transfer (number of bytes to transfer and source address) are retrieved from the buffer descriptor.

When the peripheral sends an event (fifo_Tx_alarm), a data transfer loop is executed. This event is set when there is at least "watermark level" empty rooms in the FIFO.

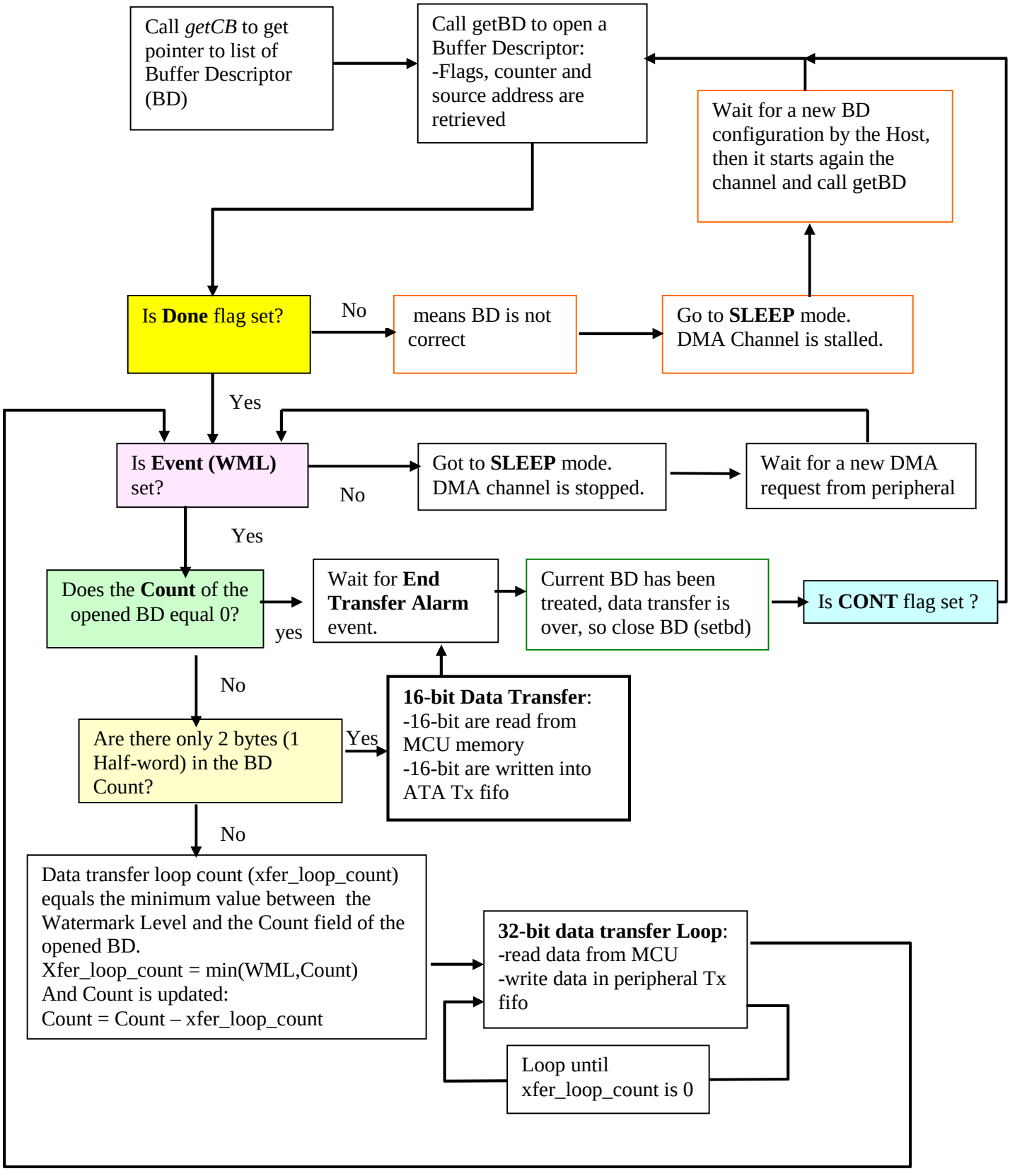
During this data transfer loop, the number of bytes that are transferred from the EMI to the TX FIFO of the peripheral is equal to the minimum value between the "watermark level" and the number of data that have not been transferred yet.

The necessary number of transfer loops are executed in order to empty the buffer, but they are executed only if the event is set. When the event is not set the channel is not executable.

When the buffer is empty, the SDMA waits for the transfer completion before restarting this algorithm (a new buffer descriptor is read). When the transfer is finished the ATA send an event (fifo_txfer_end_alarm) to the SDMA.

5.3.1.7.5 Script Sequences

Return to DMA channels supported by SDMA library



5.3.1.8 mcu_2_mshc

This script is used to transfer data from the external memory to the MSHC. The MSHC is connected to the ARM platform. The MSHC is a 32 bits.

5.3.1.8.1 Parameters transmit through the context

R1: mask to check event

R6: address of the MSHC TX FIFO

5.3.1.8.2 Parameters transmit trough the Buffer descriptor

The number of bytes to transmit is stored in the first Buffer descriptor word

The source address is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used

5.3.1.8.3 Use of the General Register during the execution

- R0: base address of the context or base address used to read events
- R1: mask to check event
- R2: channel block descriptor address / scratch
- R3: current buffer descriptor address
- R4: mode
- R5: destination address loaded by getdb, number of bytes in buffer otherwise
- R6: RX FIFO address
- R7:

5.3.1.8.4 Overview of script functionality

The parameters of the transfer (number of bytes to transfer and source address) are retrieved from the buffer descriptor.

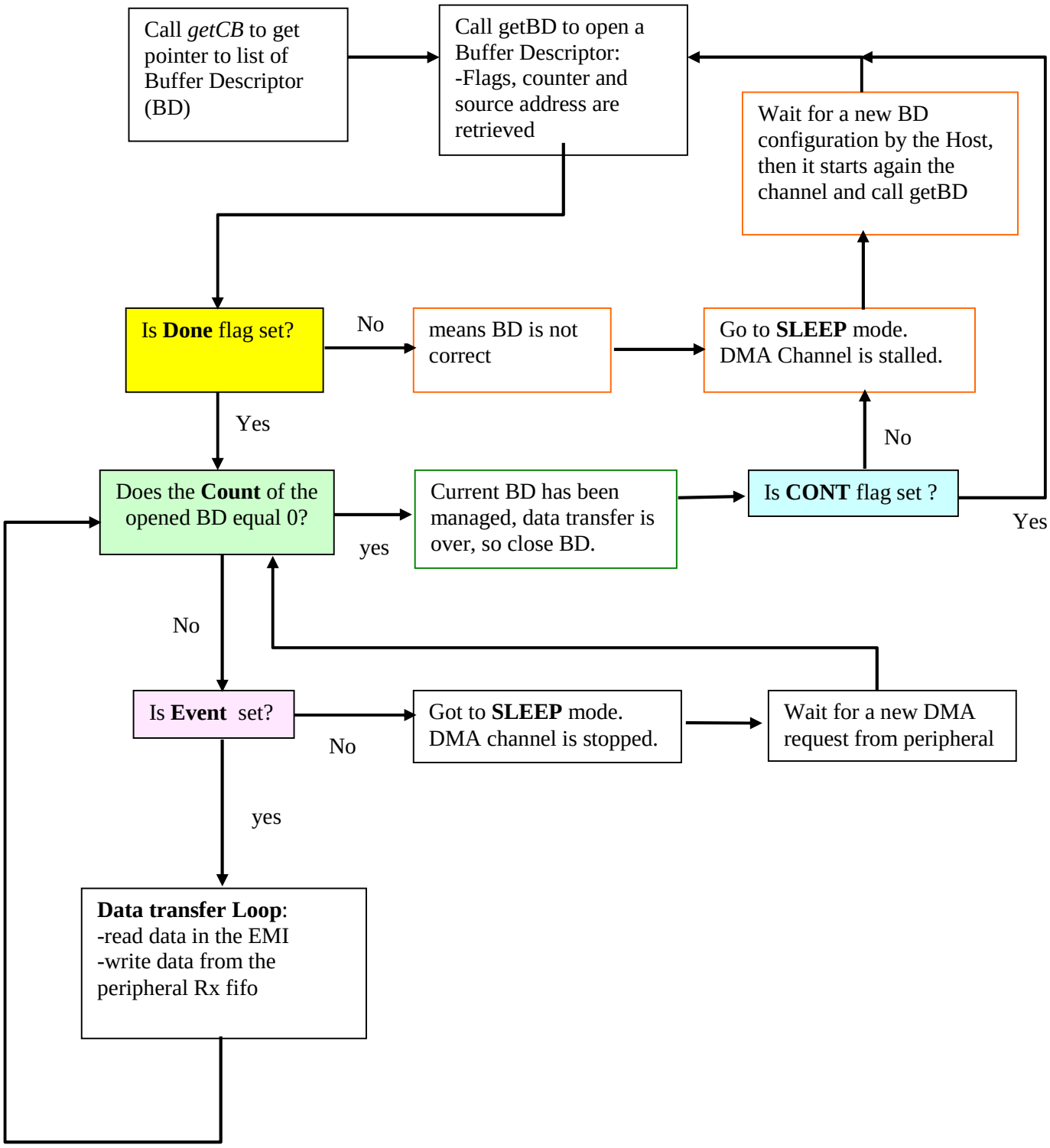
When the MSHC sends an event, a data transfer is executed. This event is set when the FIFO is not full.

The necessary number of transfers are executed in order to empty the buffer, but they are executed only if the event is set. When the event is not set the channel is not runnable.

When the buffer is empty, this algorithm restarts (a new buffer descriptor is read).

5.3.1.8.5 Script sequences

Return to DMA channels supported by SDMA library



5.3.1.9 per_2_mshc

This script is used to transfer data from the host memory space to the MSHC. The MSHC is connected to the ARM platform. The MSHC is a 32 bits.

5.3.1.9.1 Parameters transmit through the context

R1: mask to check event

R6: address of the MSHC TX FIFO

5.3.1.9.2 Parameters transmit trough the Buffer descriptor

The number of bytes to transmit is stored in the first Buffer descriptor word

The source address is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used

5.3.1.9.3 Use of the General Register during the execution

- R0: base address of the context or base address used to read events
- R1: mask to check event
- R2: channel block descriptor address / scratch
- R3: current buffer descriptor address
- R4: mode
- R5: destination address loaded by getdb, number of bytes in buffer otherwise
- R6: RX FIFO address
- R7:

5.3.1.9.4 Overview of script functionality

The parameters of the transfer (number of bytes to transfer and source address) are retrieved from the buffer descriptor.

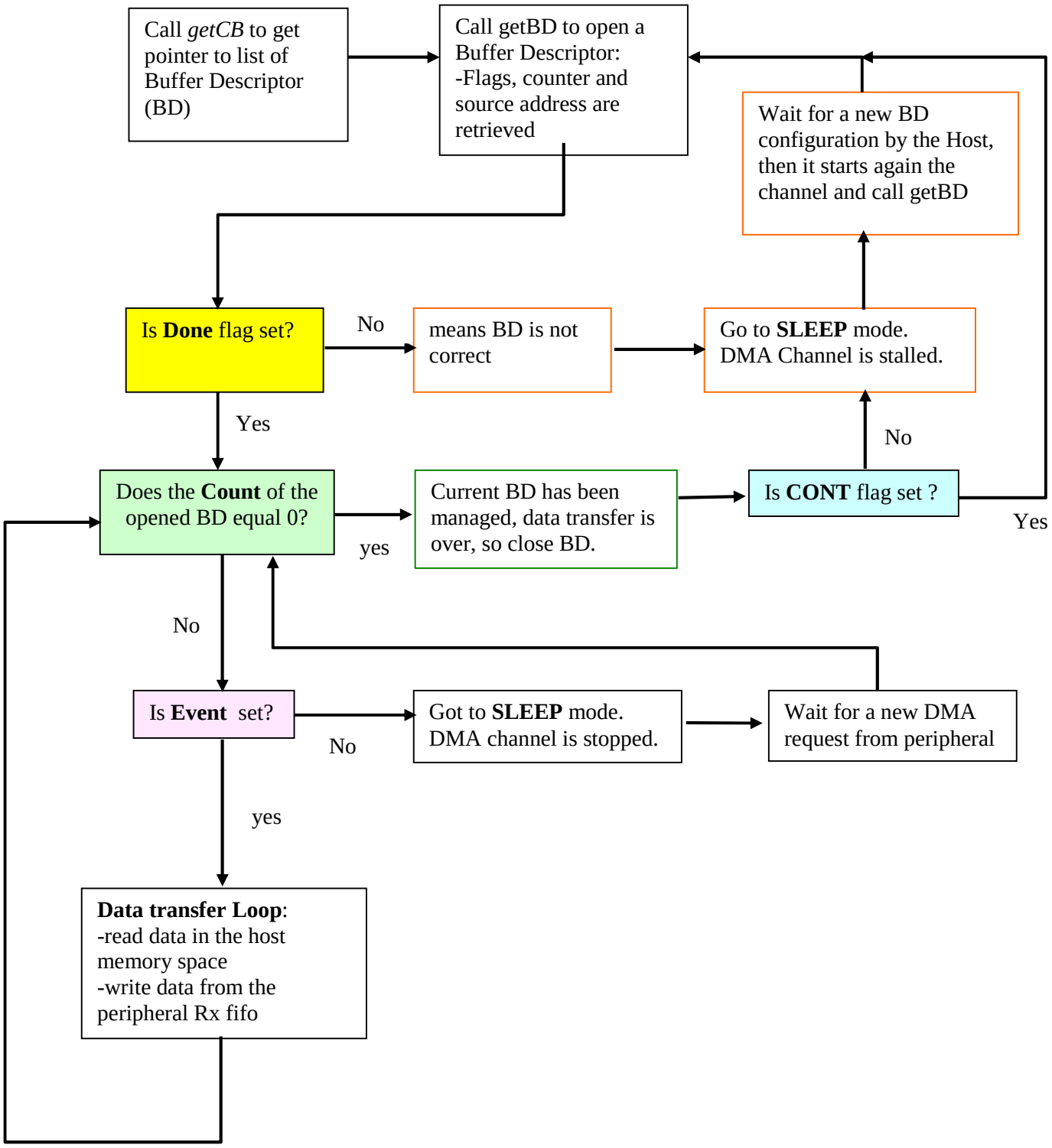
When the MSHC sends an event, a data transfer is executed. This event is set when the FIFO is not full.

The necessary number of transfers are executed in order to empty the buffer, but they are executed only if the event is set. When the event is not set the channel is not runnable.

When the buffer is empty, this algorithm restarts (a new buffer descriptor is read).

5.3.1.9.5 Script sequences

Return to DMA channels supported by SDMA library



5.3.1.10 Dsp to peripherals

There is no detailed paragraph for each script doing transfers from Dsp memory to peripheral because the scripts that perform such transfer are the same as the script for transferring data from MCU memory (External) to peripheral. In other word, mcu_2_app and dsp_2_app. They only differ for the DMA port used, therefore MDA/MSA/MD/MS registers are replaced by DDA/DSA/DD/DS but the rest of the script is identical. So to have details on dsp_2_app, just replace "external memory" by "DSP memory" and MCU by DSP.

But the **byte swapping** feature has also been implemented in **dsp_2_app**, **dsp_2_shp** scripts, on the same way as for ap_2_bp, bp_2_ap scripts. The DSP can be in big endian and the peripherals are always seen as being in little endian. A swapping of the data can be required. It can be enabled/disabled though the MSB bit of the command field of the buffer descriptor located on MCU side.

5.3.2 Peripheral to memory

In this part, all scripts to do transfers from a peripheral (on AP or on shared bus) to a memory (AP or BP side) are detailed below.

5.3.2.1 app_2_mcu

This script is loaded only in the SDMA RAM for platforms built with the ROMpass1. See details in the SDMA ROM v2 section app_2_mcu.

5.3.2.2 app_2_per

This generic script is used to transfer data from a 8/16/24 or 32 bits peripheral connected to the ARM platform to memories accessed by the PerDMA (Host internal or External memories). It can be used for SSI(8/16/24 or 32bits data size) or for CSPI(32bits data size), these peripherals being connected to the ARM platform peripheral bus.

Note: All the peripherals connected to the ARM platform peripheral bus are considered as 32-bit peripherals in term of connectivity. Therefore all SDMA accesses are 32-bit read accesses. However, the peripheral data size is taken into account for the SDMA write access through the PeripheralDMA.

Although this script can be used to access the external memories, it is less efficient (in term of data throughput) than the app_2_mcu script.

5.3.2.2.1 Parameters transmit through the context

r1: mask to check event – If script is triggered by event I, r1[I] must be set to 1. (Project dependent)

r6: address of the peripheral Rx fifo (project dependent)

r7: Watermark level – Used to determine the maximum of data that can be sent to the peripheral each time the channel is started.

5.3.2.2.2 Parameters transmit trough the Buffer descriptor

The peripheral size is set in the command field of the first Buffer descriptor word, specially bits 24,25.

The number of bytes to transmit is stored in the first Buffer descriptor word (count field)

The destination address in the host memory space is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used.

5.3.2.2.3 Use of the General Register during the execution

- r0: counter during loop, base address of the context when needed
- r1: event mask
- r2: channel control block (CCB) address when needed otherwise scratch
- r3: current buffer descriptor (BD) pointer, base address used to read events
- r4: mode (Command, Flags, count fields of the buffer descriptor)
- r5: destination address (in the external memory) loaded by getdb, number of bytes in buffer otherwise.
- r6: Rx fifo address (when context loaded)
- r7: watermark level

5.3.2.2.4 Overview of script functionality

The parameters of the transfer (peripheral size, number of bytes to transfer and source address) are retrieved from the buffer descriptor.

When the peripheral sends an event, a data transfer loop is executed. This event is set when there is at least "watermark level" data in the FIFO.

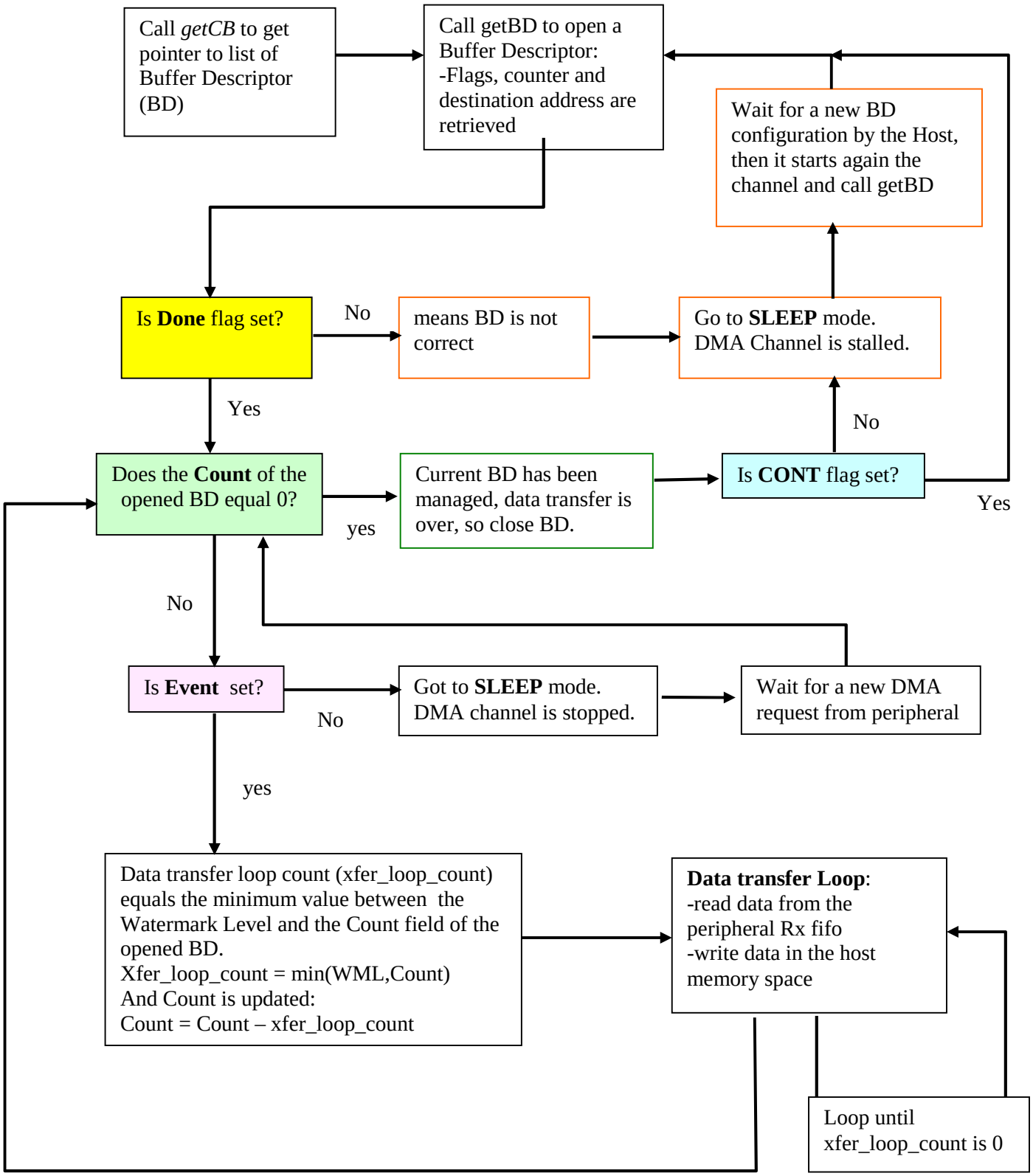
During this data transfer loop, the number of bytes that are transferred from the RX FIFO of the peripheral to the EMI is equal to the minimum value between the "watermark level" and the number of data that have not been transferred yet. For each data transfer (one read, one write), the size of the memory write access is fixed by the parameter peripheral size.

The necessary number of transfer loops are executed in order to fill the buffer, but they are executed only if the event is set. While the event is not set, the channel is not executable.

When the buffer is filled, this algorithm restarts (a new buffer descriptor is read).

5.3.2.2.5 Script sequences

Return to DMA channels supported by SDMA library



5.3.2.3 csi_2_mcu

This script is used to transfer data from the CSI to the external memory. The CSI is a 32 bits peripheral on the AHB bus, that is why the FIFO is accessed using the Burst DMA in Frozen mode.

5.3.2.3.1 Parameters transmit through the context

R1: mask to check event

R6: address of the CSI RX FIFO

R7: Watermark level

5.3.2.3.2 Parameters transmit trough the Buffer descriptor

The number of bytes to transmit is stored in the first Buffer descriptor word

The destination address is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used

5.3.2.3.3 Use of the General Register during the execution

- R0: counter
- R1: mask to check event
- R2: channel block descriptor address / scratch
- R3: current buffer descriptor address
- R4: mode
- R5: destination address
- R6: source address
- R7: watermark level

5.3.2.3.4 Overview of script functionality

The parameters of the transfer (number of bytes to transfer and destination address) are retrieved from the buffer descriptor.

When the CSI sends an event, a data transfer loop is executed. This event is set when there is at least "watermark level" data in the FIFO.

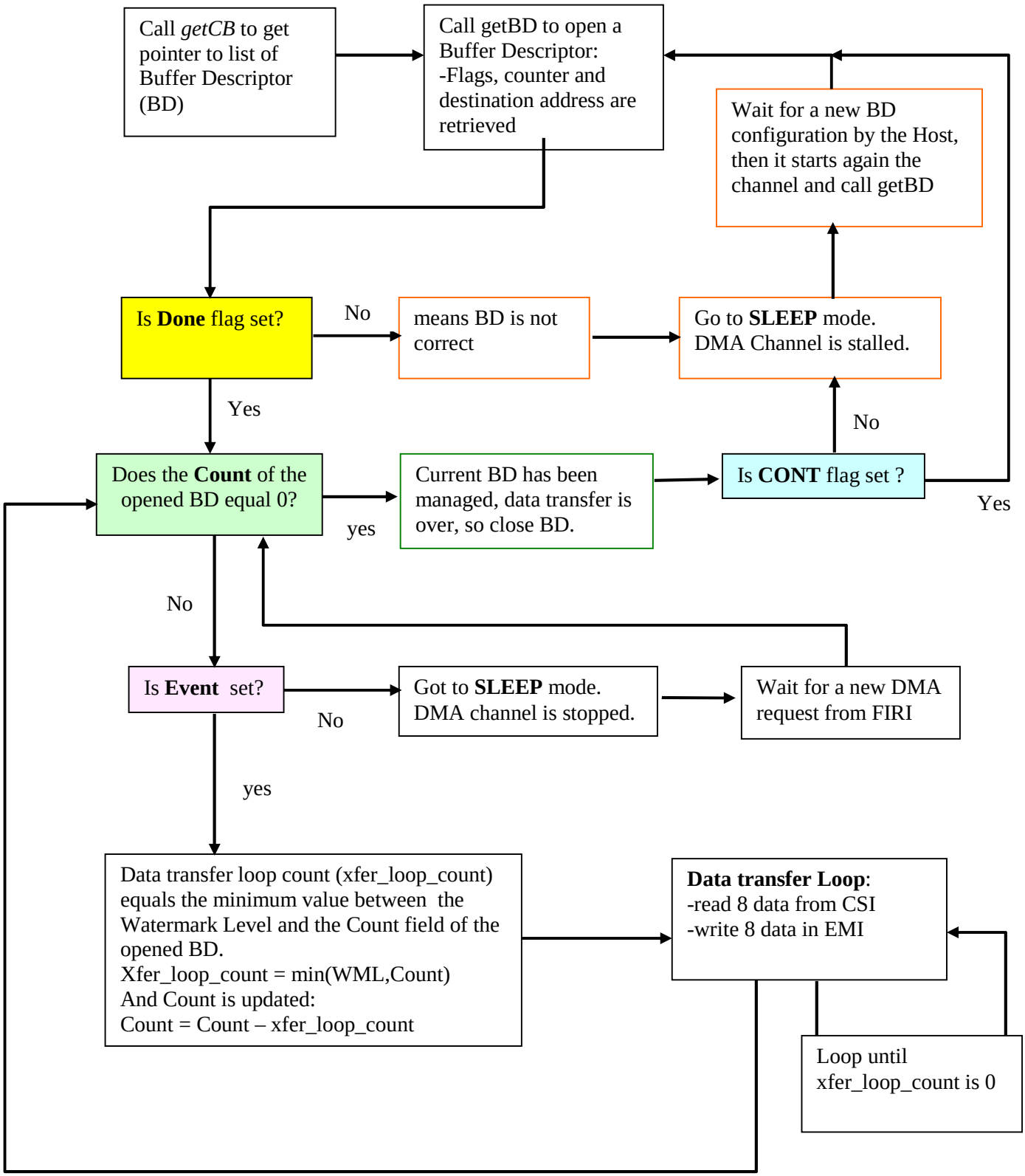
During this data transfer loop, the number of bytes that are transferred from the CSI to the EMI is equal to the minimum value between the "watermark level" and the number of data that have not been transferred yet. This transfer is perform using the burst capability of the burst dma. The 8 words fifo of the SDMA is loaded with data retrieved from the CSI, and then this data are written in the EMI.

The necessary number of transfer loops are executed in order to fill the buffer, but they are executed only if the event is set. When the event is not set the channel is not runnable.

When the buffer is full, this algorithm restarts (a new buffer descriptor is read).

5.3.2.3.5 Script sequences

Return to DMA channels supported by SDMA library



5.3.2.4 firi_2_mcu

This script is used to transfer data from the FIRI to the external memory. The FIRI is a 8 bits peripheral but the FIRI has the capability to pack several bytes (32 bits access and 16 bits access are supported).

5.3.2.4.1 Parameters transmit through the context:

R1: mask to check event

R6: base address of the FIRI

R7: Watermark level

5.3.2.4.2 Parameters transmit trough the Buffer descriptor

The number of bytes to transmit is stored in the first Buffer descriptor word

The destination address is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used

A flag SUSPEND_ON_EOP (bit 24 of the first buffer descriptor) could also be set to specify that the buffer descriptor must be closed when an End-Of-Packet is detected.

5.3.2.4.3 Use of the General Register during the execution

- R0: counter
- R1: mask to check event
- R2: channel block descriptor address / scratch
- R3: current buffer descriptor address / scratch
- R4: mode
- R5: destination address
- R6: source address
- R7: watermark level

5.3.2.4.4 Overview of script functionality

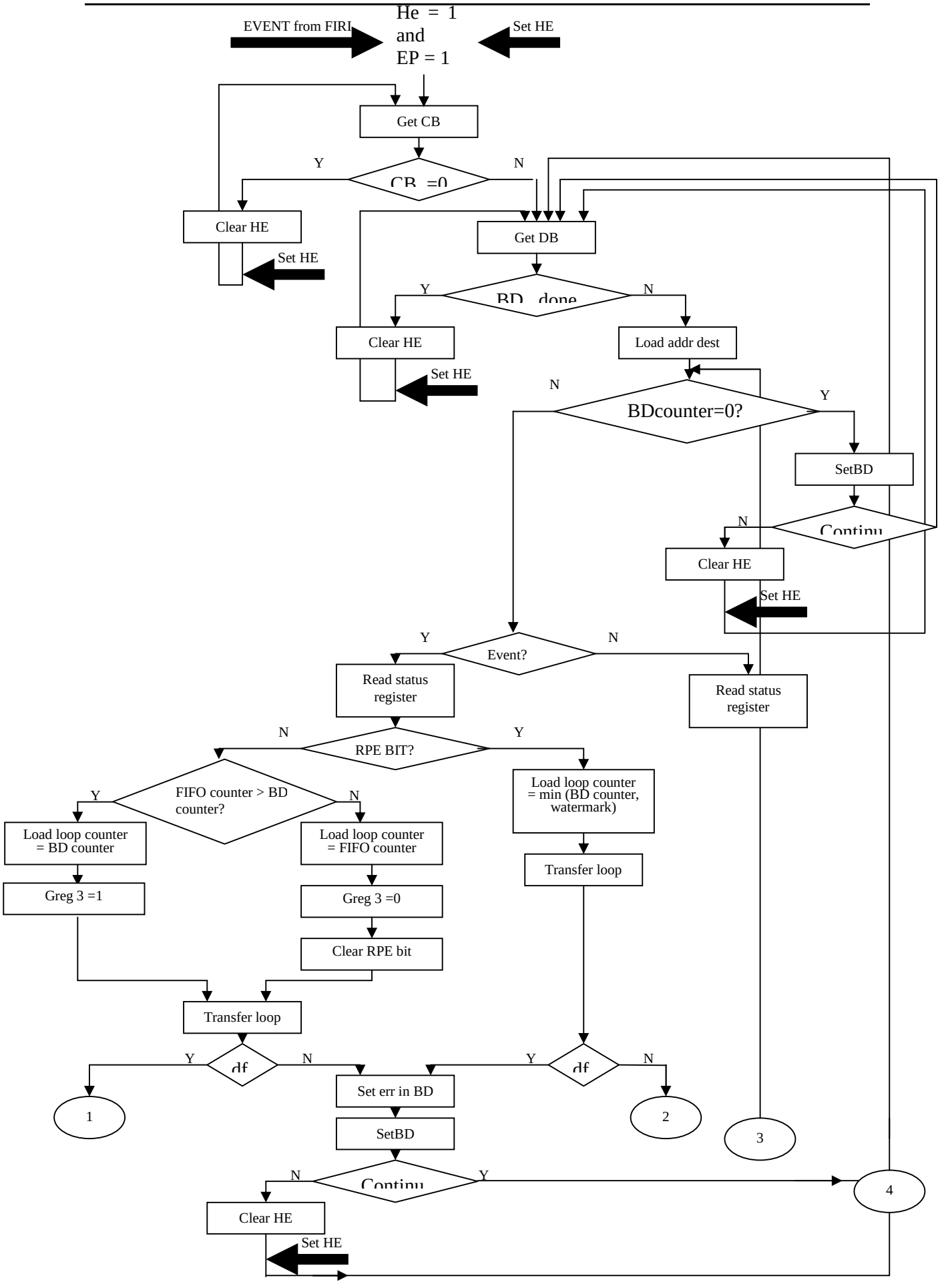
The parameters of the transfer (number of bytes to transfer and source address) are retrieved from the buffer descriptor.

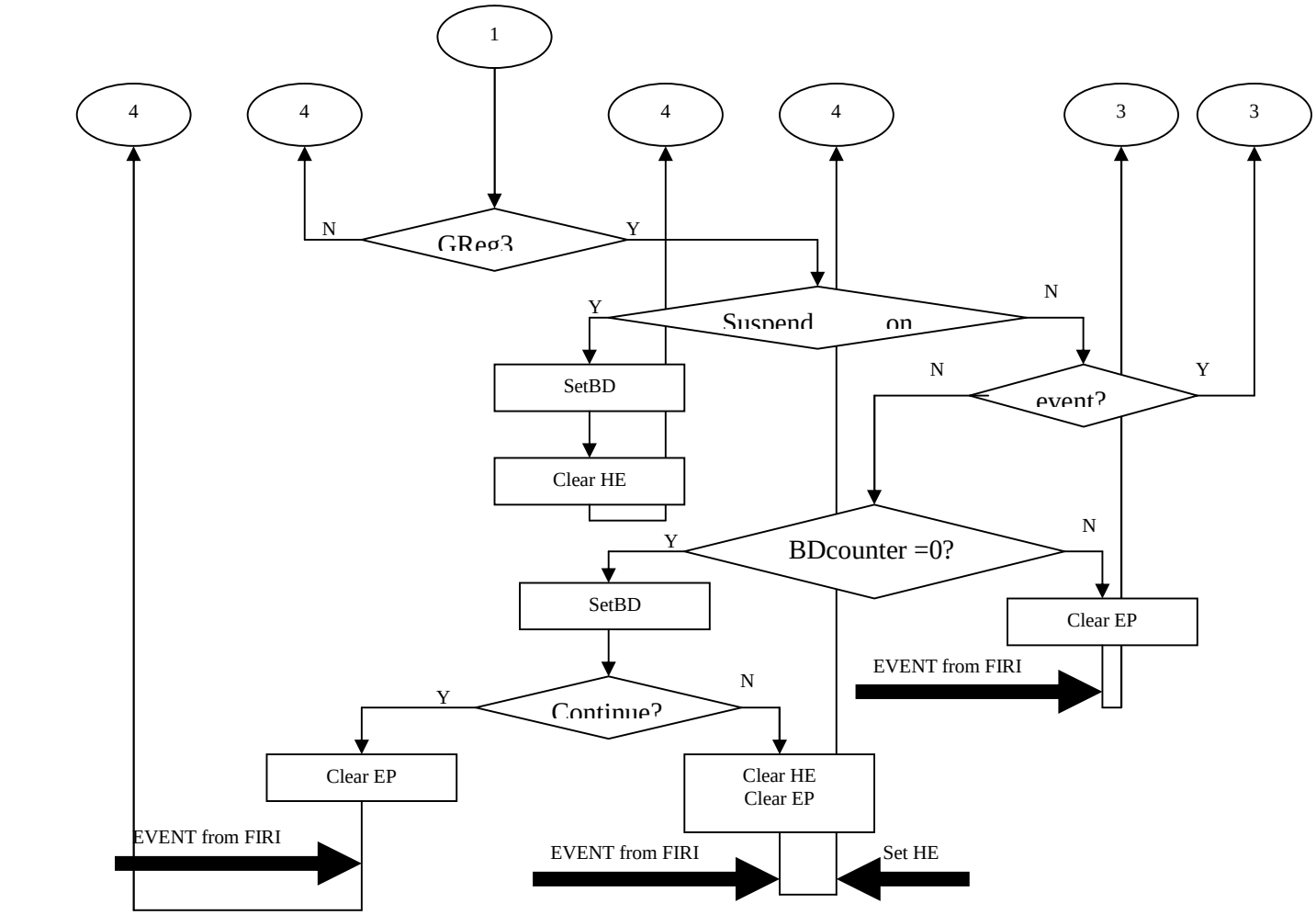
When the FIRI sends an event or when the ARM forces the SDMA to empty the FIFO, a data transfer loop is executed. The event is set when there are at least "watermark level" data in the FIFO, or when an End Of Packet has been detected by the FIRI.

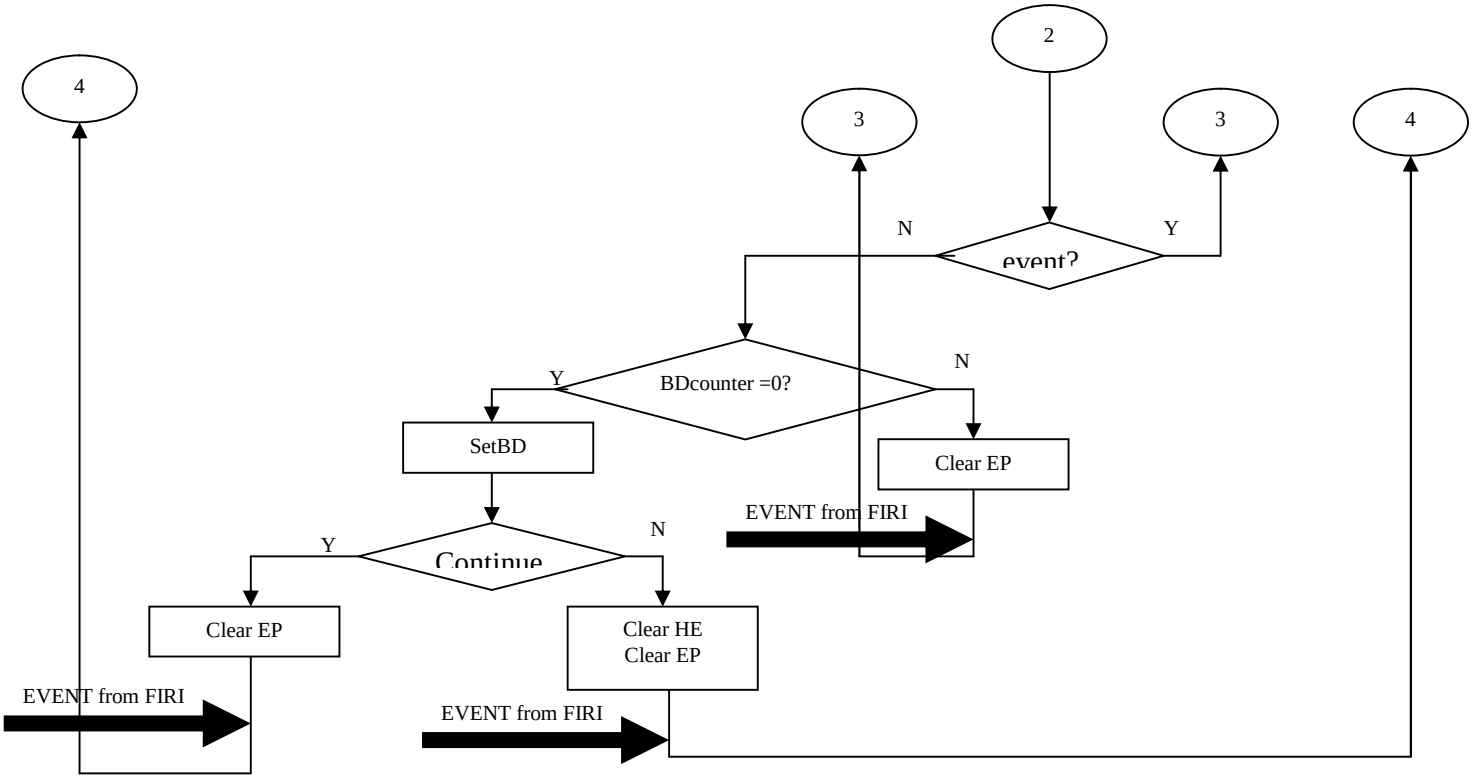
If the SDMA starts a transfer loop because an event has been sent by the FIRI, and if an End Of Packet has not been detected, the number of bytes that are transferred from the RX FIFO of the FIRI to the EMI is equal to the minimum value between the "watermark level" and the number of data that have not been transferred yet. In this case the necessary number of transfer loops are executed in order to fill the buffer, but they are executed only if the event is set. When the event is not set the channel is not runnable. When the buffer is full, this algorithm restarts (a new buffer descriptor is read).

But when an End Of Packet has been detected, the SDMA empties the FIFO. The number of data is read from a register of the FIRI. In this case the RPE bit is also cleared in the FIRI status register at the end of the transfer. This bit is set to indicate that an End Of Packet has been detected. If the SUSPEND_ON_EOP bit (bit 24 of the first word of the buffer descriptor) has been set, the SDMA sends back the buffer descriptor to the ARM.

When the ARM forces the SDMA to empty the FIFO, the SDMA read the number of data to transfer from a register of the FIRI, performs the transfer and sends back the buffer descriptor to the ARM.


EVENT from FIRI
He = 1
and
EP = 1
Set HE
Get CB
Y
CB =0
N
Clear HE
Set HE
Get DB
Y
BD done
N
Clear HE
Set HE
Load addr dest
N
BDcounter=0?
Y
SetBD
N
Continu
Clear HE
Set HE
Y
Event?
N
Read status register
Read status register
N
RPE BIT?
Y
Y
FIFO counter > BD counter?
N
Load loop counter = BD counter
Load loop counter = FIFO counter
Load loop counter = min (BD counter, watermark)
Greg 3 =1
Greg 3 =0
Transfer loop
Clear RPE bit
Transfer loop
Y
df
N
Y
df
N
1
Set err in BD
SetBD
2
3
N
Continu
Y
4
Clear HE
Set HE


1
4
4
4
4
3
3
N
GReg3
Y
Y
Suspend on
N
N
event?
Y
SetBD
Clear HE
Y
BDcounter =0?
N
SetBD
Clear EP
EVENT from FIRI
Y
Continue?
N
Clear EP
Clear HE
Clear EP
EVENT from FIRI
EVENT from FIRI
Set HE


2
4
3
3
4
N
event?
Y
Y
BDcounter =0?
N
SetBD
Clear EP
EVENT from FIRI
Y
Continue
N
Clear EP
Clear HE
Clear EP
EVENT from FIRI
EVENT from FIRI

5.3.2.5 firi_2_per

This script is used to transfer data from the FIRI to the host memory space. The FIRI is a 8 bits peripheral but the FIRI has the capability to pack several bytes (32 bits access and 16 bits access are supported).

5.3.2.5.1 Parameters transmit through the context:

R1: mask to check event

R6: base address of the FIRI

R7: Watermark level

5.3.2.5.2 Parameters transmit trough the Buffer descriptor

The number of bytes to transmit is stored in the first Buffer descriptor word

The destination address is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used

A flag SUSPEND_ON_EOP (bit 24 of the first buffer descriptor) could also be set to specify that the buffer descriptor must be closed when an End-Of-Packet is detected.

5.3.2.5.3 Use of the General Register during the execution

- R0: counter
- R1: mask to check event
- R2: channel block descriptor address / scratch
- R3: current buffer descriptor address / scratch
- R4: mode
- R5: destination address
- R6: source address
- R7: watermark level

5.3.2.5.4 Overview of script functionality

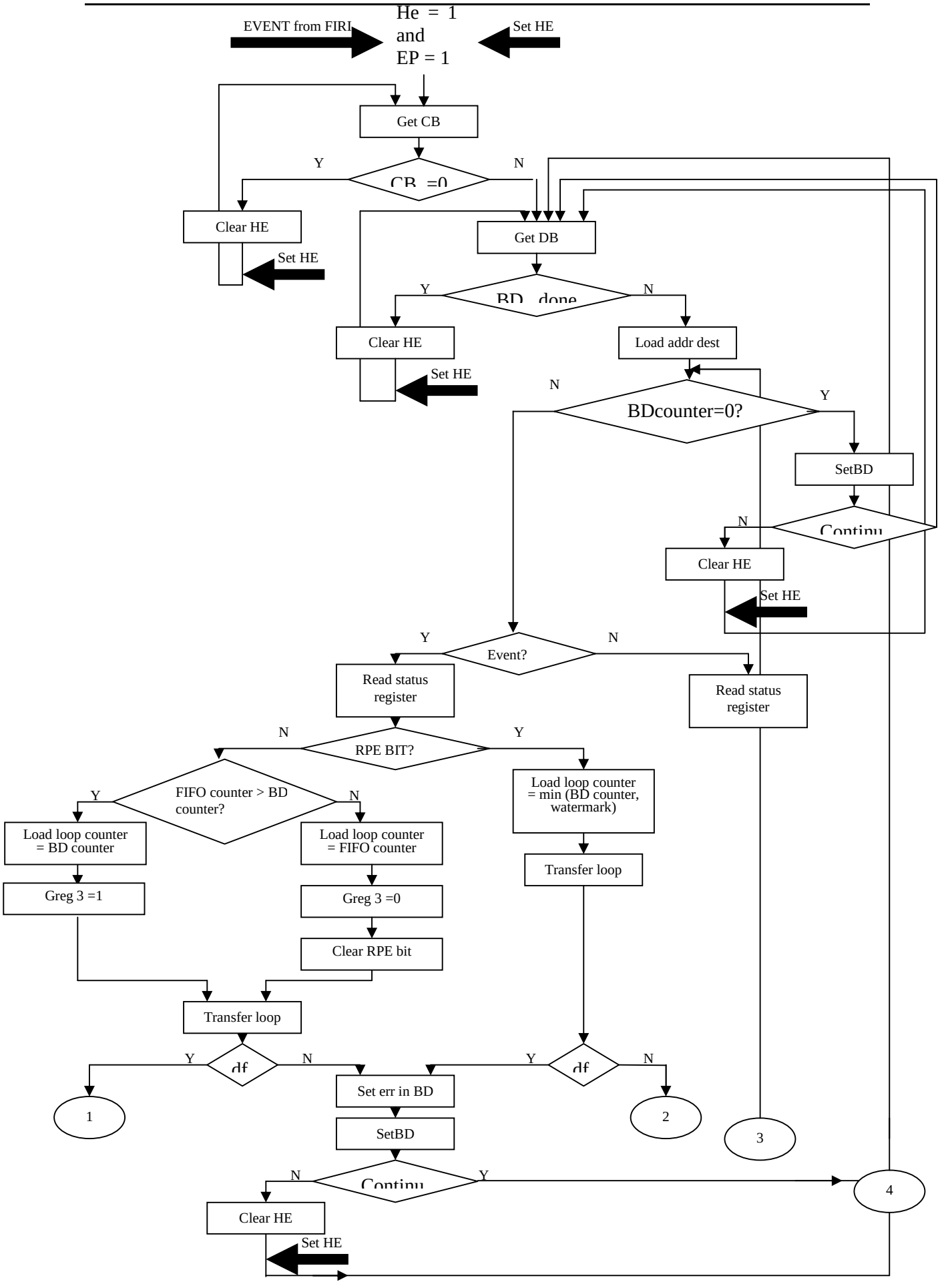
The parameters of the transfer (number of bytes to transfer and source address) are retrieved from the buffer descriptor.

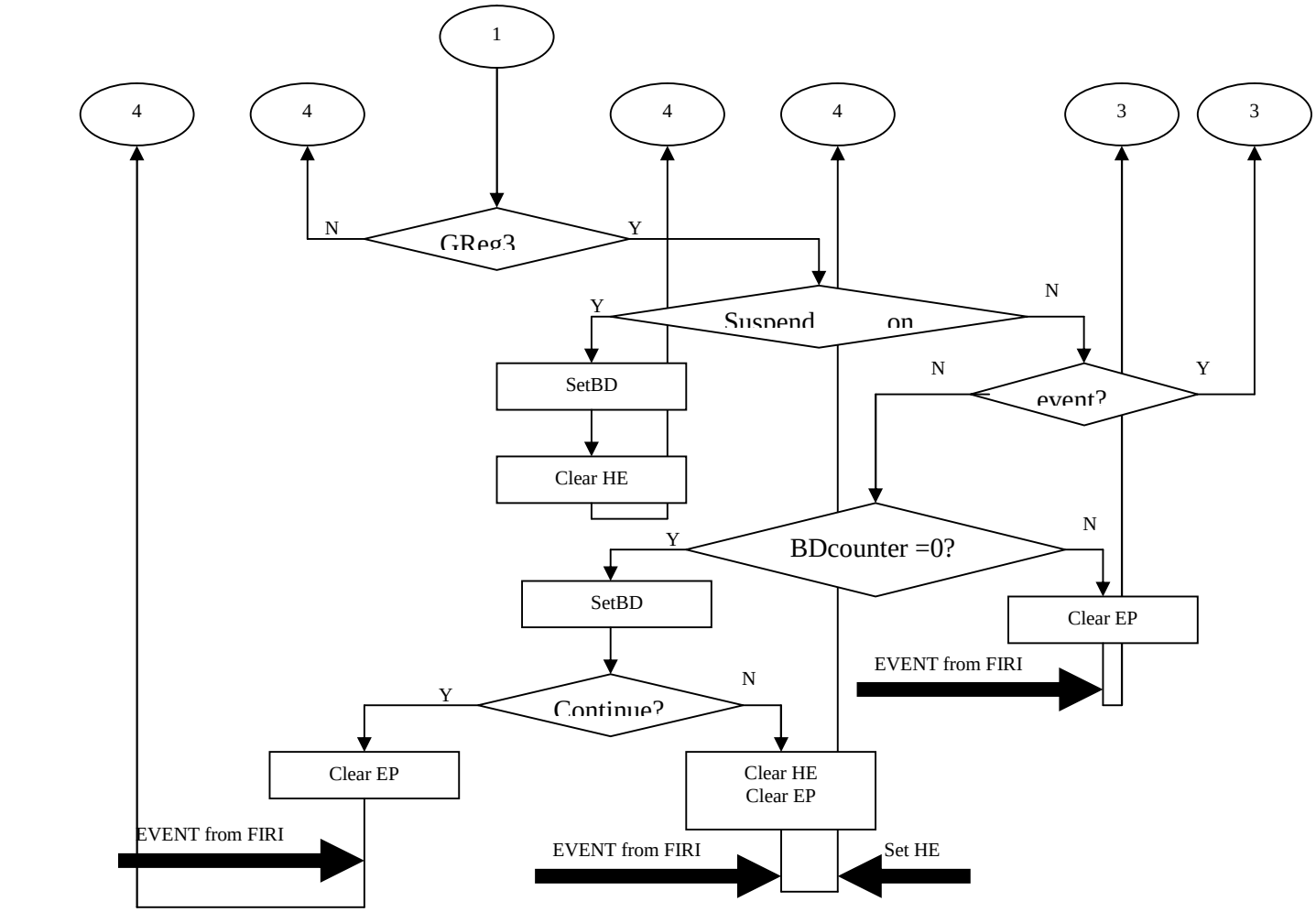
When the FIRI sends an event or when the ARM forces the SDMA to empty the FIFO, a data transfer loop is executed. The event is set when there are at least "watermark level" data in the FIFO, or when an End Of Packet has been detected by the FIRI.

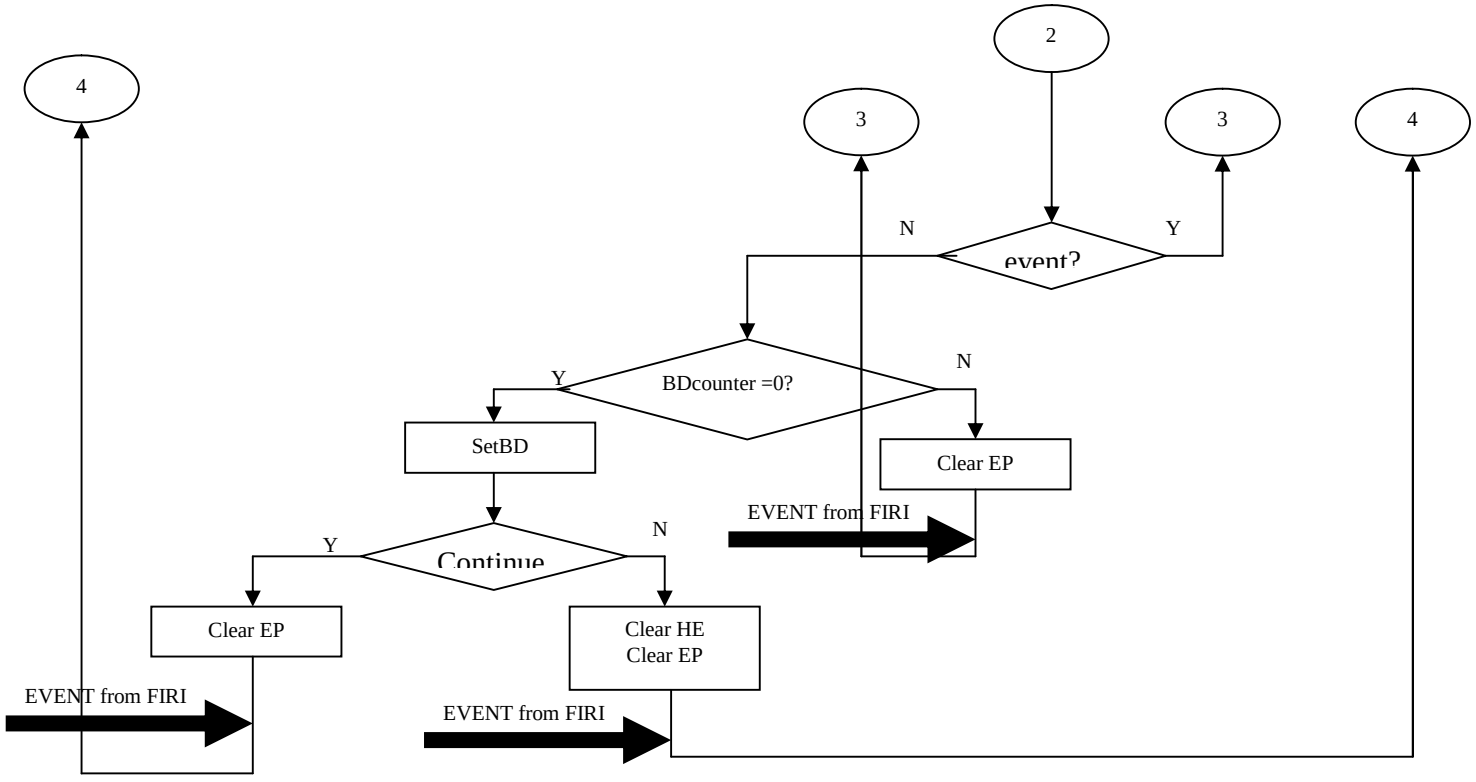
If the SDMA starts a transfer loop because an event has been sent by the FIRI, and if an End Of Packet has not been detected, the number of bytes that are transferred from the RX FIFO of the FIRI to the host memory space is equal to the minimum value between the "watermark level" and the number of data that have not been transferred yet. In this case the necessary number of transfer loops are executed in order to fill the buffer, but they are executed only if the event is set. When the event is not set the channel is not runnable. When the buffer is fill, this algorithm restarts (a new buffer descriptor is read).

But when an End Of Packet has been detected, the SDMA empties the FIFO. The number of data is read from a register of the FIRI. In this case the RPE bit is also cleared in the FIRI status register at the end of the transfer. This bit is set to indicate that an End Of Packet has been detected. If the SUSPEND_ON_EOP bit (bit 24 of the first word of the buffer descriptor) has been set, the SDMA sends back the buffer descriptor to the ARM.

When the ARM forces the SDMA to empty the FIFO, the SDMA read the number of data to transfer from a register of the FIRI, perform the transfer and sends back the buffer descriptor to the ARM.


EVENT from FIRI
He = 1
and
EP = 1
Set HE
Get CB
Y
CB =0
N
Clear HE
Set HE
Get DB
Y
BD done
N
Clear HE
Set HE
Load addr dest
N
BDcounter=0?
Y
SetBD
N
Continu
Clear HE
Set HE
Y
Event?
N
Read status register
Read status register
N
RPE BIT?
Y
Y
FIFO counter > BD counter?
N
Load loop counter = BD counter
Load loop counter = FIFO counter
Load loop counter = min (BD counter, watermark)
Greg 3 =1
Greg 3 =0
Transfer loop
Clear RPE bit
Transfer loop
Y
df
N
Y
df
N
1
Set err in BD
SetBD
2
3
N
Continu
Y
4
Clear HE
Set HE


1
4
4
4
4
3
3
N
GReg3
Y
Y
Suspend on
N
N
event?
Y
SetBD
Clear HE
Y
BDcounter =0?
N
SetBD
Clear EP
EVENT from FIRI
Y
Continue?
N
Clear EP
Clear HE
Clear EP
EVENT from FIRI
EVENT from FIRI
Set HE


2
4
3
3
4
N
event?
Y
Y
BDcounter =0?
N
SetBD
Clear EP
EVENT from FIRI
Y
Continue
N
Clear EP
Clear HE
Clear EP
EVENT from FIRI
EVENT from FIRI

5.3.2.6 uart_2_mcu

This script is loaded only in the SDMA RAM for platforms built with the ROMpass1. See details in the SDMA ROM v2 section uart_2_mcu.

5.3.2.7 uart_2_per

This script is based on the uart_2_mcu script, but the destination buffer is located in the internal host memory. The Peripheral DMA port of the SDMA is used.

5.3.2.8 shp_2_mcu

This script is loaded only in the SDMA RAM for platforms built with the ROMpass1. See details in the SDMA ROM v2 section shp_2_mcu.

5.3.2.9 shp_2_per

This generic script is used to transfer data from a 8/16/24 or 32 bits peripheral connected to the shared peripheral bus accessed through the SPBA to memories accessed by the PeripheralDMA (Host internal or External memories). It can be used for SSI(8/16/24 or 32bits data size), for CSPI(32bits data size) or for SDHC(MMC)/SIM (16bits data size), these peripherals being connected to the shared peripheral bus.

Although this script can be used to access the external memories, it is less efficient (in term of data throughput) than the shp_2_mcu script.

5.3.2.9.1 Parameters transmit through the context

r1: mask to check event – If script is triggered by event I, r1[I] must be set to 1. (Project dependent)

r6: address of the peripheral Rx fifo (project dependent)

r7: Watermark level – Used to determine the maximum of data that can be sent to the peripheral each time the channel is started.

5.3.2.9.2 Parameters transmit trough the Buffer descriptor

The peripheral size is set in the command field of the first Buffer descriptor word, specially bits 24,25.

The number of bytes to transmit is stored in the first Buffer descriptor word (count field)

The destination address in the external memory is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used.

5.3.2.9.3 Use of the General Register during the execution

- r0: counter during loop, base address of the context when needed
- r1: event_mask
- r2: channel control block (CCB) address when needed otherwise scratch
- r3: current buffer descriptor (BD) pointer, base address used to read events
- r4: mode (Command, Flags, count fields of the buffer descriptor)
- r5: destination address (in the external memory) loaded by getdb, number

of bytes in buffer otherwise.

- r6: Rx fifo address (when context loaded)
- r7: watermark level

5.3.2.9.4 Overview of script functionality

The parameters of the transfer (peripheral size, number of bytes to transfer and source address) are retrieved from the buffer descriptor.

When the peripheral sends an event, a data transfer loop is executed. This event is set when there is at least "watermark level" data in the FIFO.

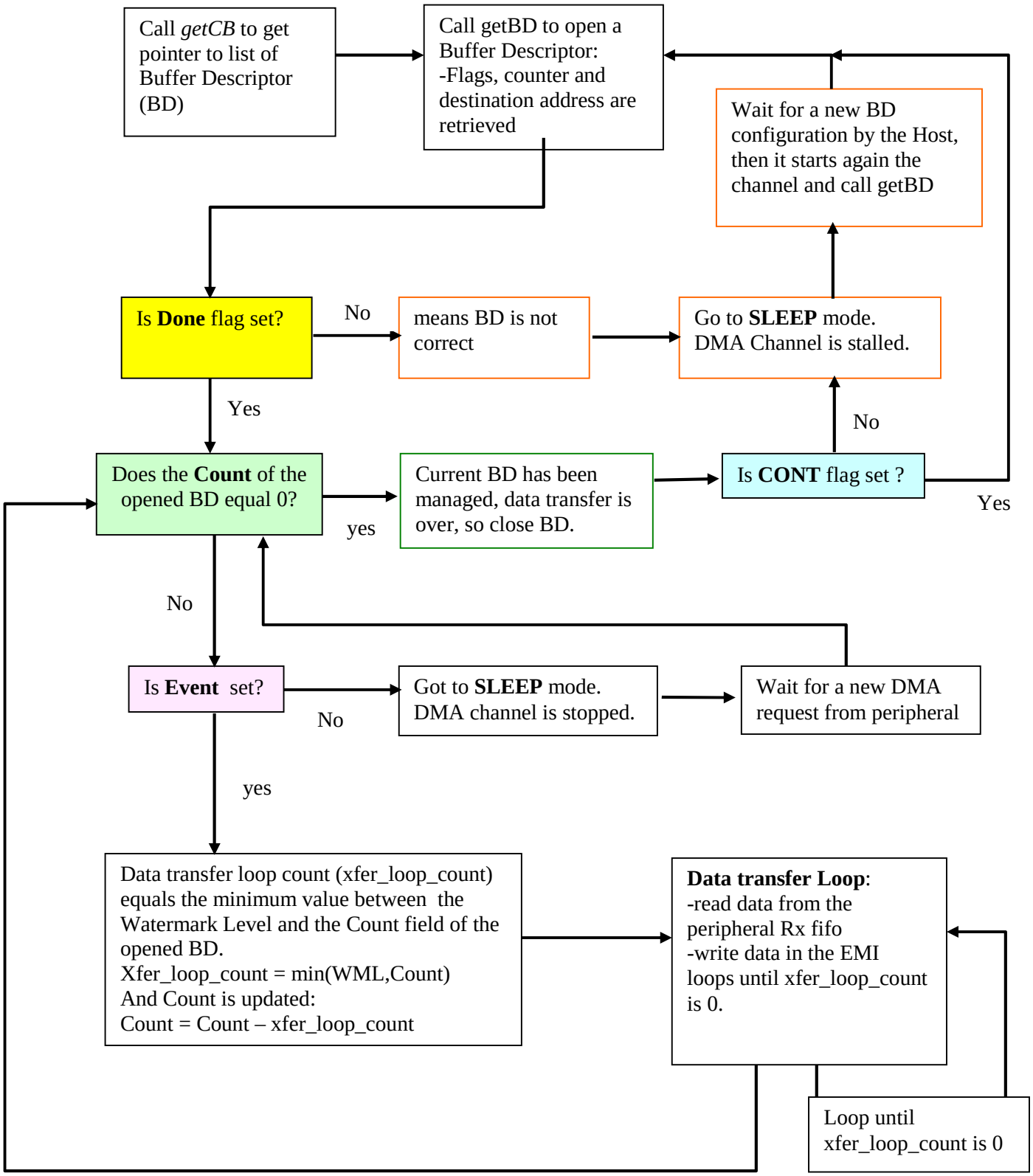
During this data transfer loop, the number of bytes that are transferred from the RX FIFO of the peripheral to the EMI is equal to the minimum value between the "watermark level" and the number of data that have not been transferred yet. For each data transfer (one read, one write), the size of the memory write access is fixed by the parameter peripheral size.

The necessary number of transfer loops are executed in order to fill the buffer, but they are executed only if the event is set. While the event is not set, the channel is not executable.

When the buffer is filled, this algorithm restarts (a new buffer descriptor is read).

5.3.2.9.5 Script sequences

Return to DMA channels supported by SDMA library



5.3.2.10 uartsh_2_mcu

This script is loaded only in the SDMA RAM for platforms built with the ROMpass1. See details in the SDMA ROM v2 section uartsh_2_mcu.

5.3.2.11 uartsh_2_per

This script is based on the uartsh_2_mcu script, but it is targeted to the shared UART located on the shared peripheral bus. The shared UART is accessed through the Peripheral DMA unit of the SDMA and the destination buffer is located in the internal host memory.

5.3.2.12 ata_2_mcu

This script is used to transfer data from the external memory to the ATA peripheral. The ATA is a 32 bits peripheral but it can be accessed in 16 bits. This peripheral has two DMA request, one is dedicated to flag that the Rx fifo watermark has been over passed (fifo_Rx_alarm), the second warns when a end of transfer occurs (fifo_txfer_end_alarm).

5.3.2.12.1 Parameters transmit through the context

r0: mask to check the fifo_Rx_alarm event, meaning the event (a.k.a DMA request) that is set by the ATA when its Rx fifo over passed its watermark.

r1: mask to check the fifo_txfer_end_alarm event

r6: base address of the ATA

r7: watermark level of the Rx fifo (WML)

5.3.2.12.2 Parameters transmit trough the Buffer descriptor

The number of bytes to transmit is stored in the first Buffer descriptor word

The destination address in the external memory is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used

5.3.2.12.3 Use of the General Register during the execution

- r0: counter during loop / mask for the Rx fifo event
- r1: mask for the end of transmit event
- r2: channel control block descriptor address /scratch
- r3: current buffer descriptor address / mask for the Rx fifo event
- r4: mode field of BD / buffer size (BD count field)
- r5: destination address (in external memory) / base address for the event register
- r6: base address of the ATA
- r7: watermark level

5.3.2.12.4 Overview of script functionality

The parameters of the transfer (number of bytes to transfer and destination address) are retrieved from the buffer descriptor.

When the peripheral sends an event (fifo_Rx_alarm or fifo_txfer_end_alarm), a data transfer loop is executed.

The fifo_Rx_alarm event is set when there is at least "watermark level" data in the FIFO. The fifo_txfer_end alarm is set when all the data have been received.

During this data transfer loop, the number of bytes that are transferred from the RX FIFO of the peripheral to the EMI is equal to the minimum value between the "watermark level" and the number of data that have not been transferred yet, when the fifo_RX_alarm is set.

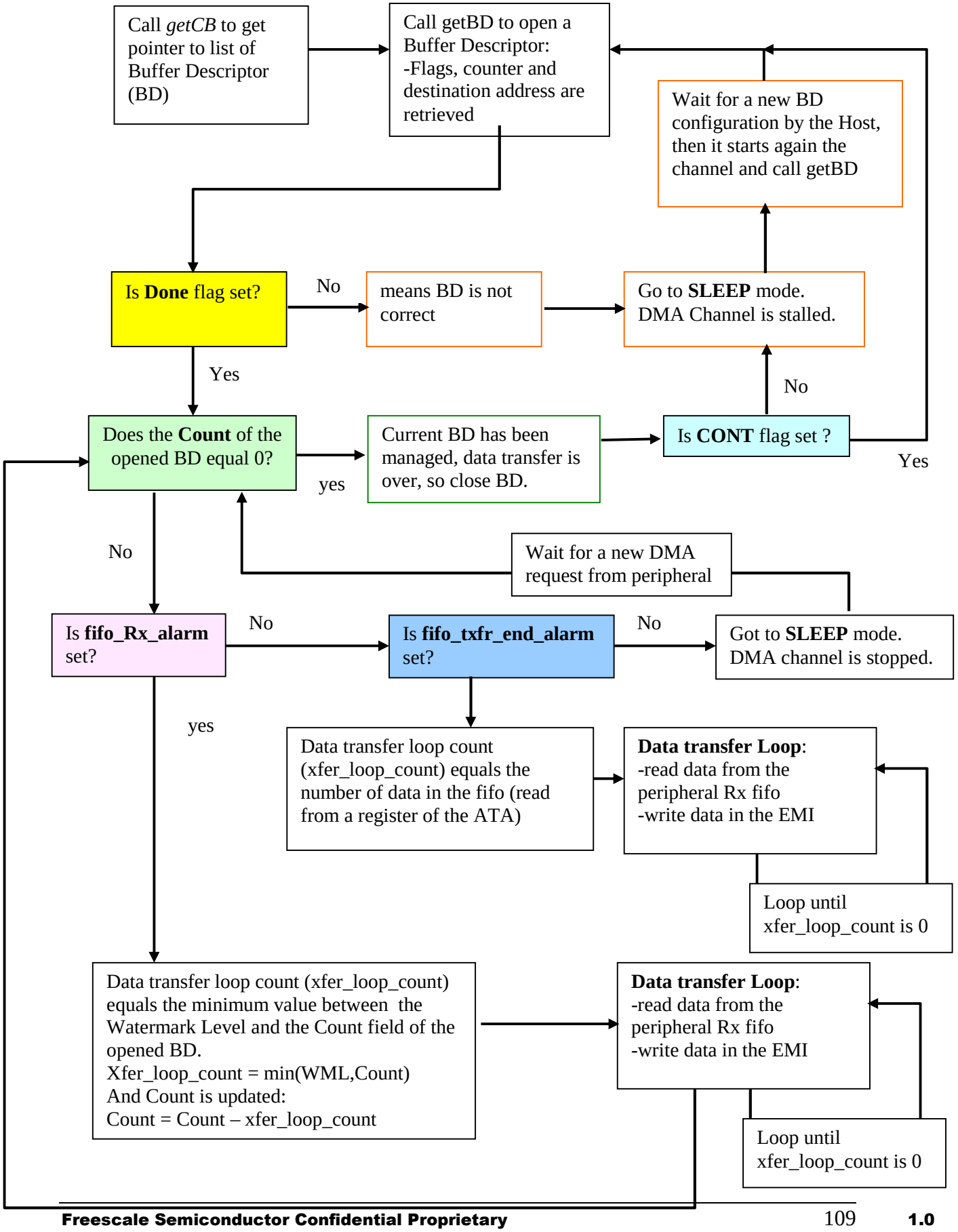
When there is only the fifo_txfer_end_alarm, the SDMA empties the FIFO, the number of data stored in the FIFO is read from a register of the ATA.

The necessary number of transfer loops are executed in order to fill the buffer, but they are executed only if the event is set. When the event is not set the channel is not executable.

When the buffer is filled, this algorithm is restarted (a new buffer descriptor is read).

5.3.2.12.5 Script sequences

Return to DMA channels supported by SDMA library



5.3.2.13 mshc_2_mcu

This script is used to transfer data from the MSHC to the External memory. The MSHC is connected to the ARM platform. The MSHC is a 32 bits.

5.3.2.13.1 Parameters transmit through the context

R1: mask to check event

R6: address of the MSHC RX FIFO

5.3.2.13.2 Parameters transmit trough the Buffer descriptor

The number of bytes to transmit is stored in the first Buffer descriptor word

The destination address is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used

5.3.2.13.3 Use of the General Register during the execution

- R0: base address of the context or base address used to read events
- R1: mask to check event
- R2: channel block descriptor address / scratch
- R3: current buffer descriptor address
- R4: mode
- R5: destination address loaded by getdb, number of bytes in buffer otherwise
- R6: TX FIFO address
- R7:

5.3.2.13.4 Overview of script functionality

The parameters of the transfer (number of bytes to transfer and source address) are retrieved from the buffer descriptor.

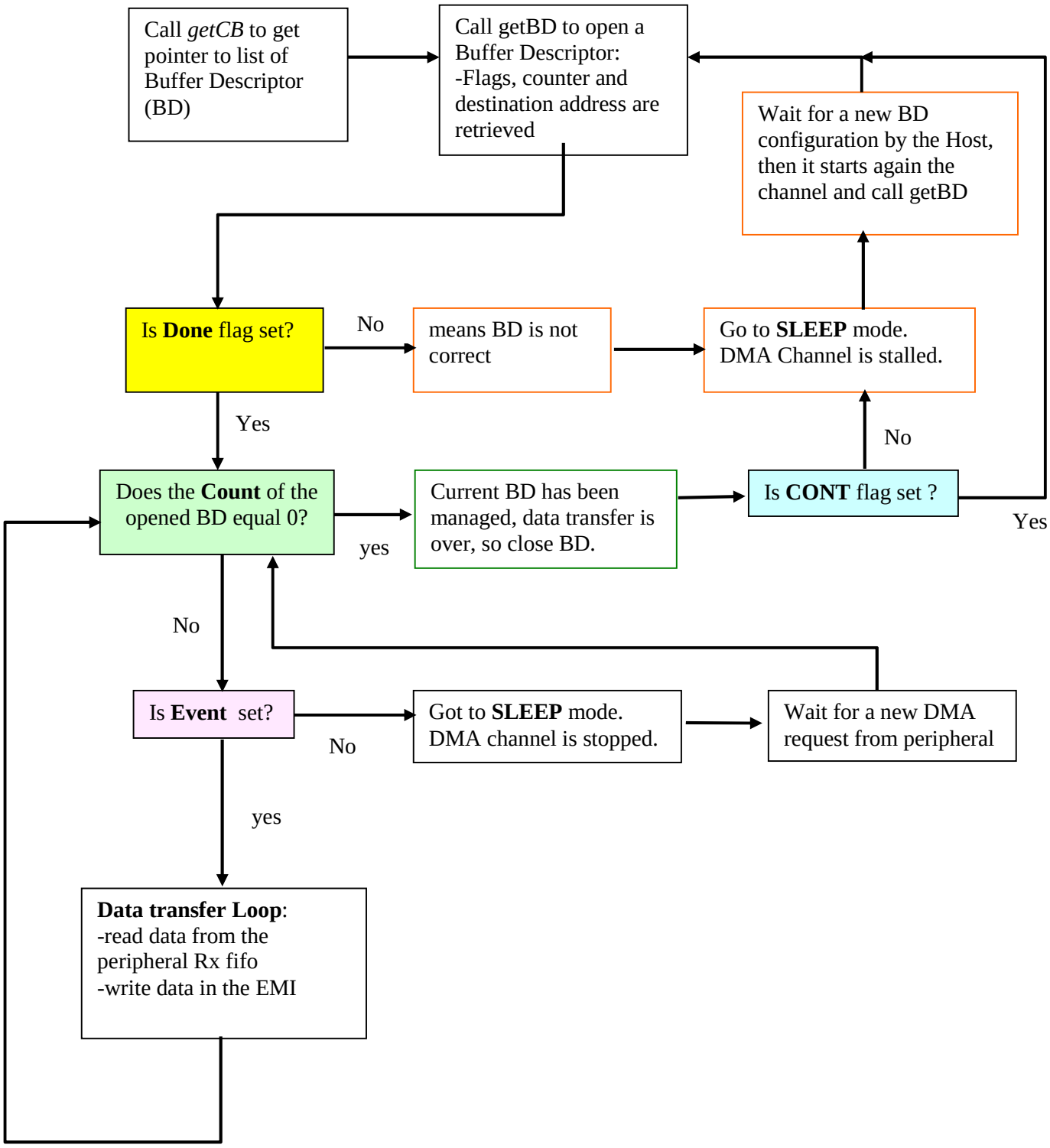
When the MSHC sends an event, a data transfer is executed. This event is set when the FIFO is not empty.

The necessary number of transfers are executed in order to fill the buffer, but they are executed only if the event is set. When the event is not set the channel is not runnable.

When the buffer is full, this algorithm restarts (a new buffer descriptor is read).

5.3.2.13.5 Script sequences

Return to DMA channels supported by SDMA library



5.3.2.14 mshc_2_per

This script is used to transfer data from the MSHC to the host memory space. The MSHC is connected to the ARM platform. The MSHC is a 32 bits.

5.3.2.14.1 Parameters transmit through the context

R1: mask to check event

R6: address of the MSHC RX FIFO

5.3.2.14.2 Parameters transmit trough the Buffer descriptor

The number of bytes to transmit is stored in the first Buffer descriptor word

The destination address is stored in the second Buffer descriptor word (address field).

The Extended Buffer address is not used

5.3.2.14.3 Use of the General Register during the execution

- R0: base address of the context or base address used to read events
- R1: mask to check event
- R2: channel block descriptor address / scratch
- R3: current buffer descriptor address
- R4: mode
- R5: destination address loaded by getdb, number of bytes in buffer otherwise
- R6: TX FIFO address
- R7:

5.3.2.14.4 Overview of script functionality

The parameters of the transfer (number of bytes to transfer and source address) are retrieved from the buffer descriptor.

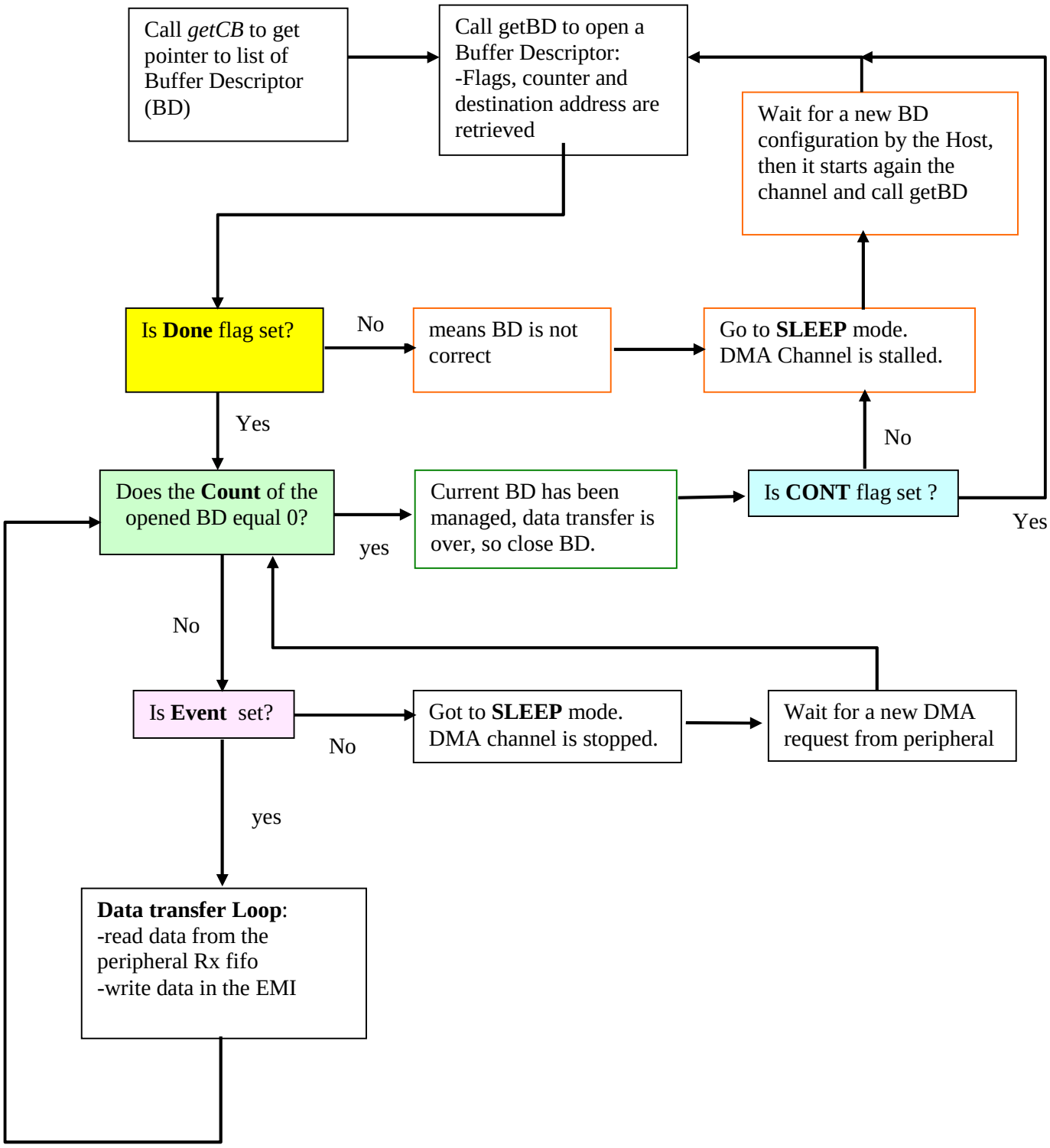
When the MSHC sends an event, a data transfer is executed. This event is set when the FIFO is not empty.

The necessary number of transfers are executed in order to fill the buffer, but they are executed only if the event is set. When the event is not set the channel is not runnable.

When the buffer is full, this algorithm restarts (a new buffer descriptor is read).

5.3.2.14.5 Script sequences

Return to DMA channels supported by SDMA library



5.3.3 Others

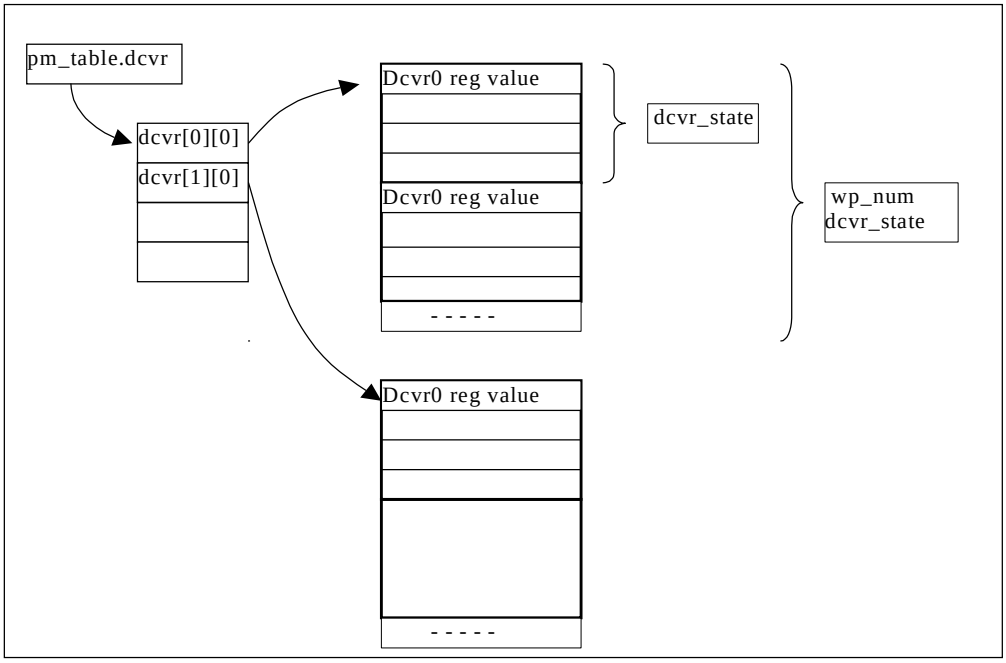
5.3.3.1 dptc_dvfs

This script is used to send measurements to the Clock Controller Module according to a buffer received by software that contains the predicted performance levels.

5.3.3.1.1 Parameters transmitted through the context

R6: The CCM base address (should be 0x53f80000)

R7: The *pm_table* which contains the predicted performance levels is constant for all the buffer descriptors of the channel. So it is passed by channel context.

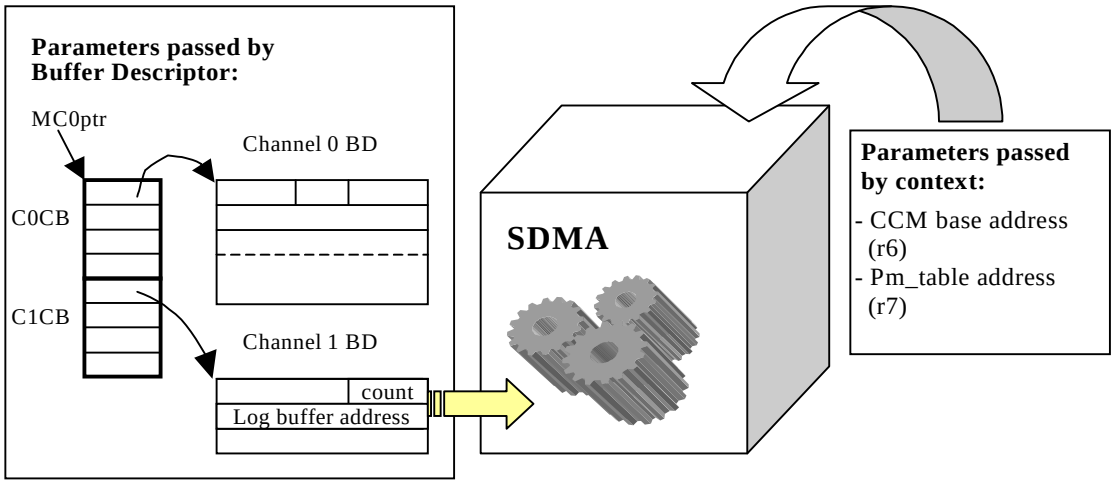


5.3.3.1.2 Parameters transmitted through the Buffer descriptor

- Count in the BD mode: number of load tracking samples

- Log buffer address get from the first address of the BD structure

In the following image there is presented the mode of transmitting the parameters:



5.3.3.1.3 Use of the General Register during the execution

- R0: {1'b0,new dfsup_val[0],new(i,up_val,vscnt_val),18'b0,int_flag,new j}
- R1: RAM base context address of the current channel
- R2: channel block descriptor address
- R3: current BD addr - PMCR0 value
- R4: BD mode
- R5: Log Buffer address
- R6: CCM base start address (should be 0x53f80000); transmitted through context
- R7: pm_table address; transmitted through context

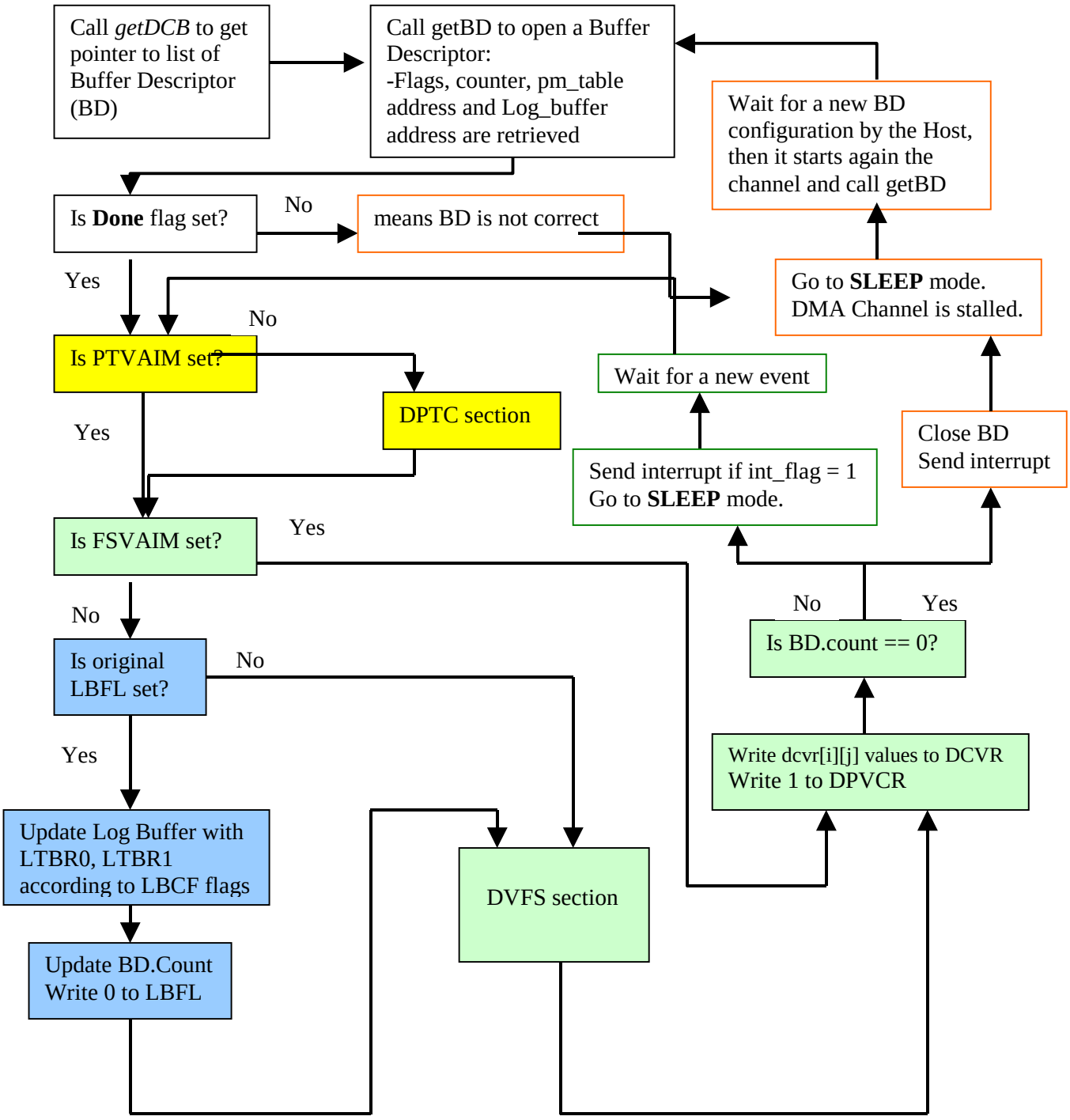
5.3.3.1.4 Overview of script functionality

The main functionality of the DPTC/DVFS SDMA script is to send measurements to the Clock Controller Module according to a buffer received by software that contains the predicted performance levels.

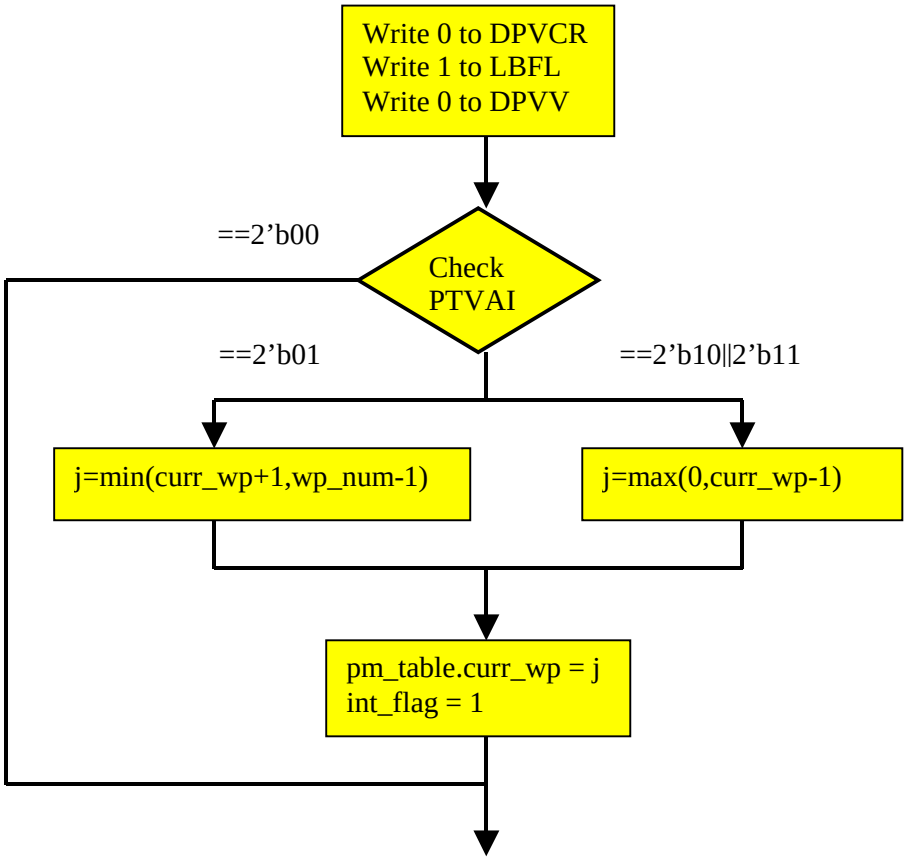
In each SDMA event the SDMA reads the performance level table passed through the context of the channel, updates the current working point index and(or) the current dvfs table index according to the event(s), sends the measurements of these indexes of the table to the CPU.

The SDMA can also read the DVFS log buffer and write it to a buffer received from the software on the Log buffer event

5.3.3.1.5 Script sequences

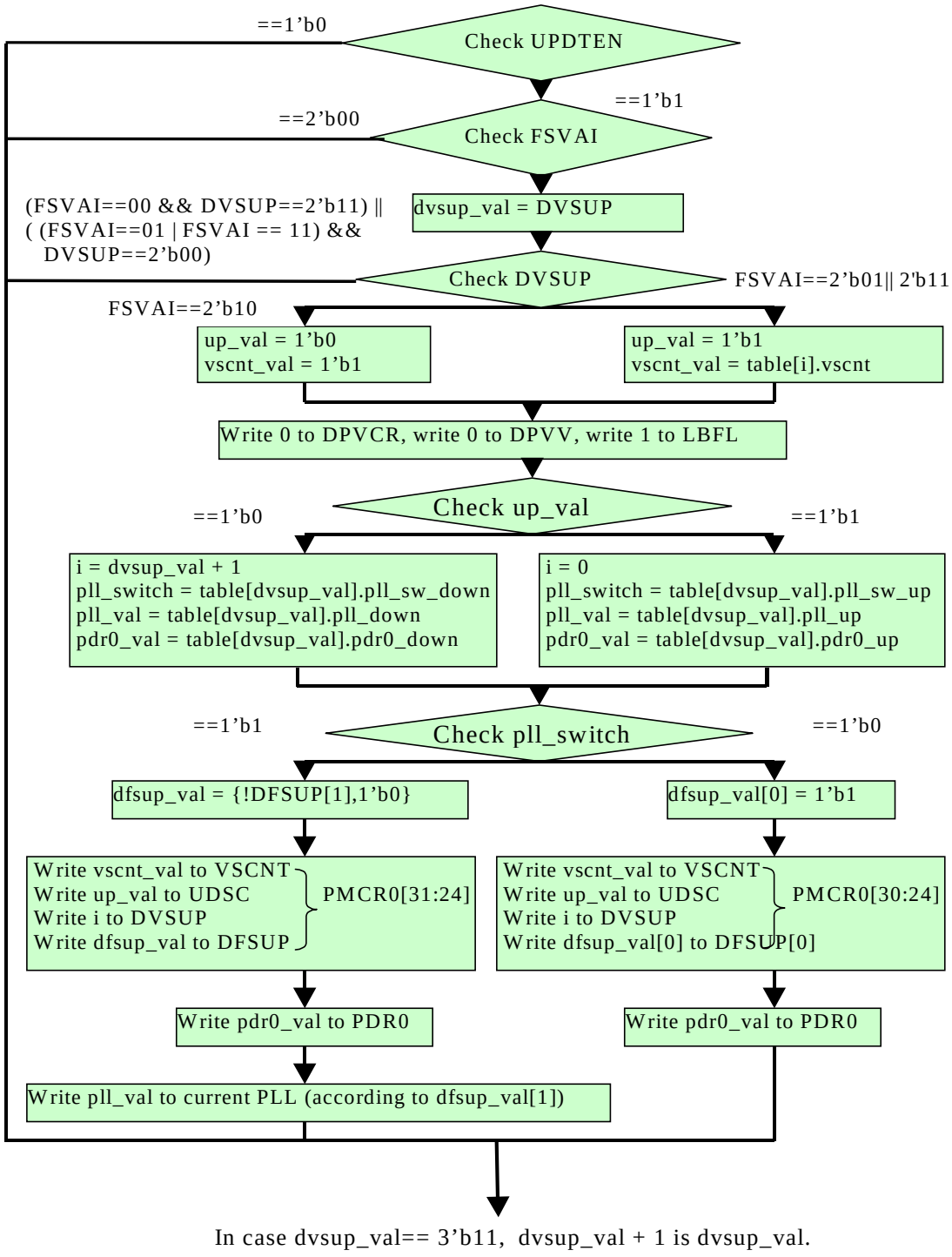

Call *getDCB* to get pointer to list of Buffer Descriptor (BD)
Call getBD to open a Buffer Descriptor:
-Flags, counter, pm_table address and Log_buffer address are retrieved
Wait for a new BD configuration by the Host, then it starts again the channel and call getBD
Is **Done** flag set?
No
means BD is not correct
Yes
Go to **SLEEP** mode.
DMA Channel is stalled.
No
Is PTVAIM set?
DPTC section
Wait for a new event
Yes
Close BD
Send interrupt
Send interrupt if int_flag = 1
Go to **SLEEP** mode.
Is FSVAIM set?
Yes
No
Yes
Is BD.count == 0?
No
Is original LBFL set?
No
Yes
Write dcvr[i][j] values to DCVR
Write 1 to DPVCR
Update Log Buffer with LTBR0, LTBR1 according to LBCF flags
DVFS section
Update BD.Count
Write 0 to LBFL

DPTC section:



In case curr_wp == wp_num, curr_wp+1 is curr_wp. On the same way, if curr_wp == 0, curr_wp-1 is curr_wp.

DVFS section :



In case dvsup_val== 3'b11, dvsup_val + 1 is dvsup_val.

6 SDMA Memories Layout

6.1 ROM V1

The ROM v1 release consists of 5 sections that occupy 3936 bytes of the 4Kbytes. ROM is 97% full.

Note: The test_sram is a built-in-self test that checks correct behaviour of the SDMA internal RAM by writing and reading back data at every address memory. It is used for testing the chip and to determine if SDMA is alive.

The Data Xfer loops consist in several subroutines that may be called by SDMA scripts to perform the data transfer operation.

The following picture presents the ROM organization and also the name of the scripts that constitute each sub-section.

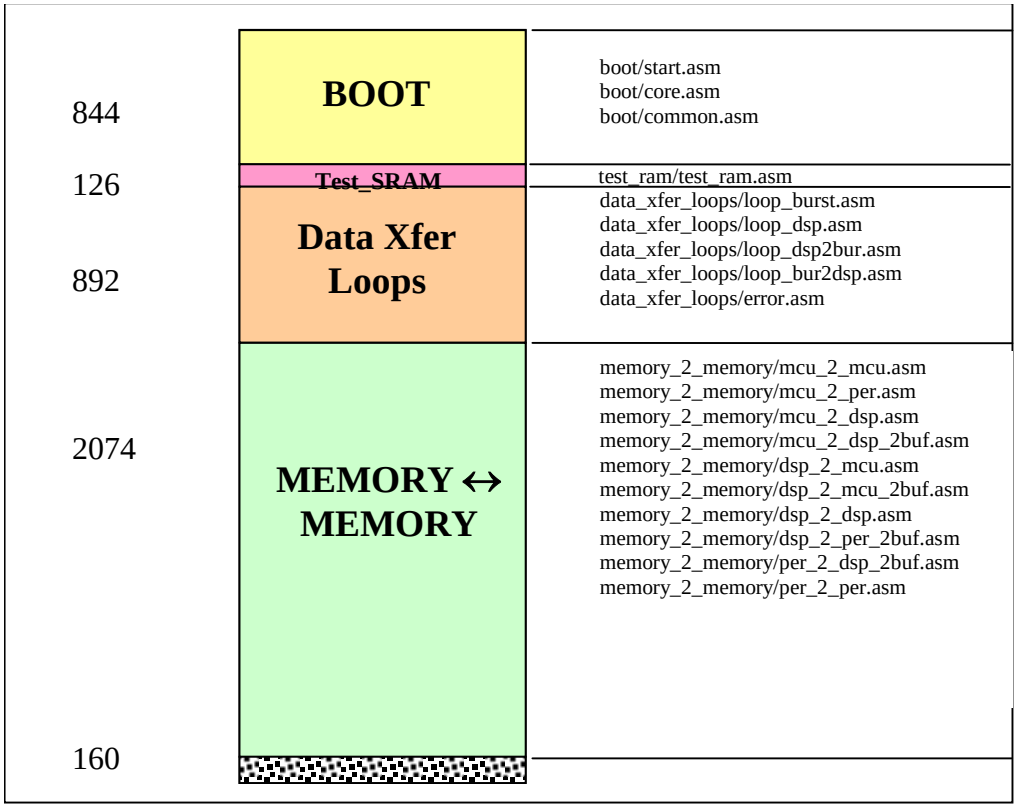


Figure 2 Layout of the 4Kbyte ROM

6.2 ROM V2

The current ROM release consists of 6 sections that occupy 4084 bytes of the 4,096Kbytes. ROM is 99,7% full.

The test_sram is a built-in-self test that checks correct behaviour of the SDMA internal RAM by writing and reading back data at every address memory. It is used for testing the chip and to determine if SDMA is alive.

The Error management consist in several subroutines that may be called by the other SDMA scripts to perform updates in error cases.

The signature is a data equal to the version of the ROM – the latest v2 version is 2.8 (signature equal to 02080208).

The following picture presents the ROM organization and also the name of the scripts that constitute each sub-section.

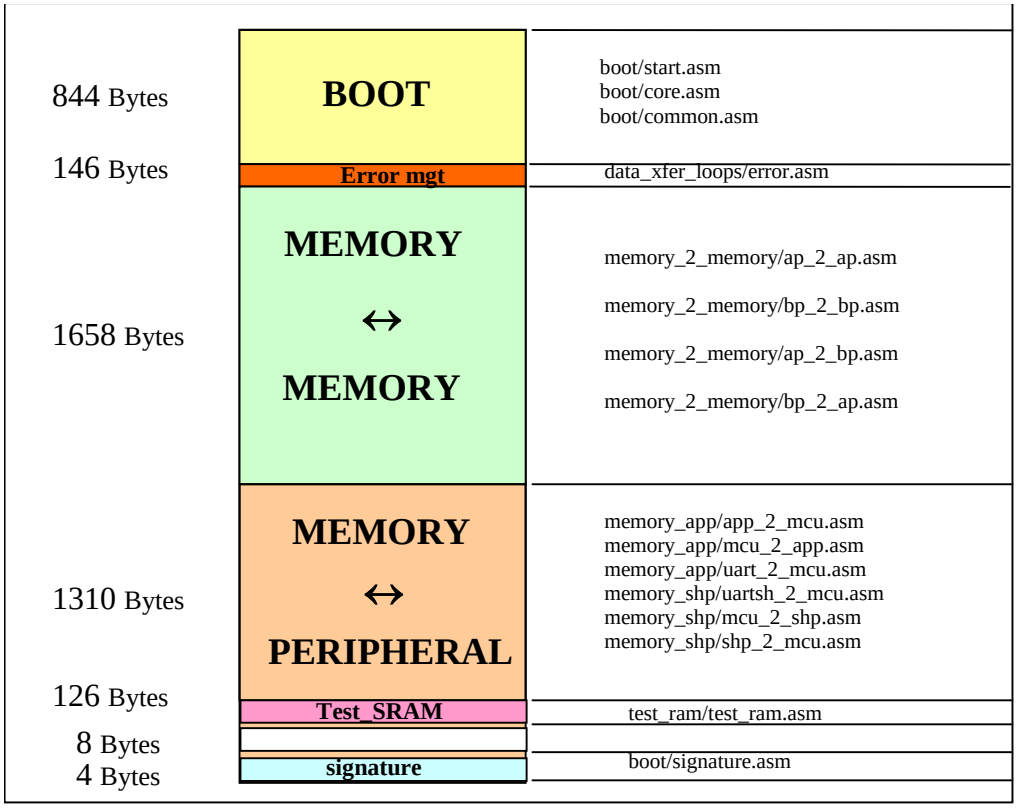
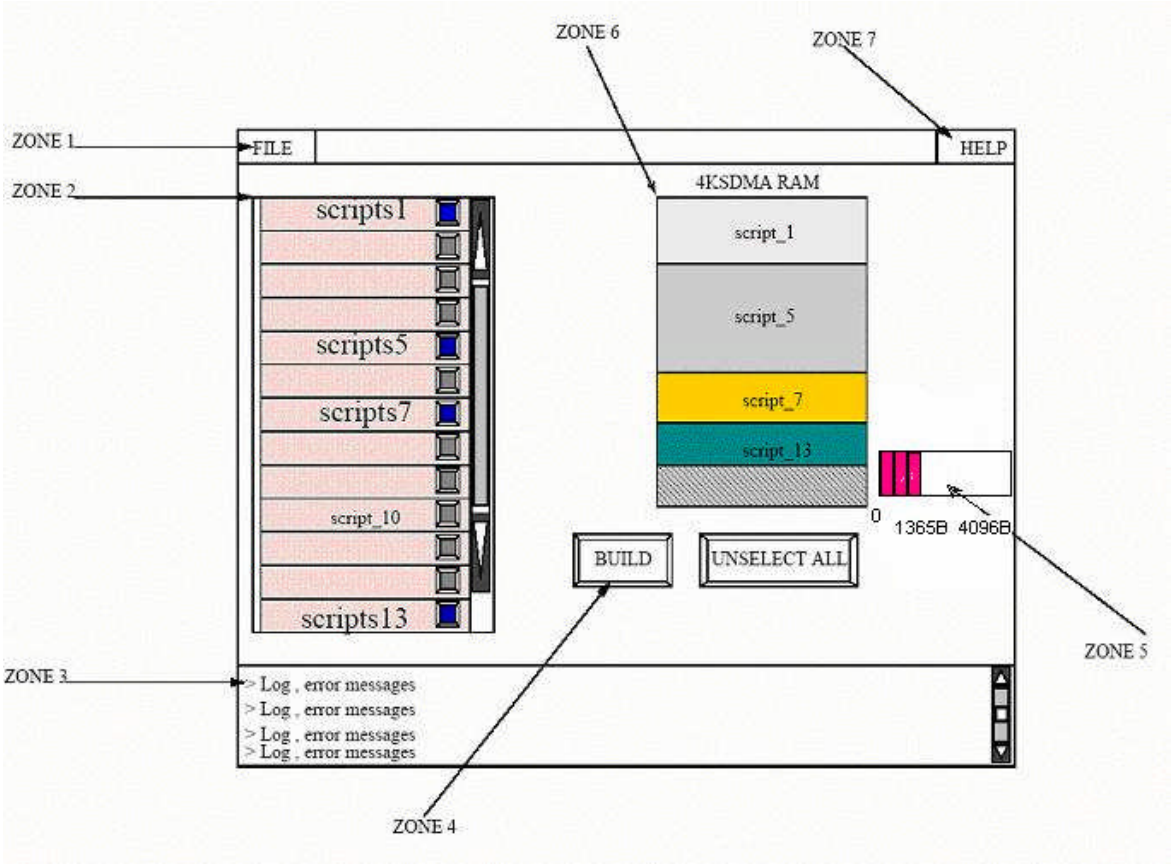


Figure 3 Layout of the 4Kbyte ROM

6.3 RAM

The SDMA RAM is divided into a 4Kbyte area reserved to store the 32 channel context and a 4Kbytes dedicated to store the different SDMA scripts that will be used later on during the application. The ARM is responsible, through channel 0 (see [3]), to download the scripts from the external flash to the SDMA RAM. As it is not possible to store the whole SDMA scripts library inside the SDMA RAM, several RAM images must be present in the external flash memories. For instance, a first image may consist in the build of Peripheral to external memory scripts while a second one could be the build of all the DSP mem to peripheral scripts. To build the RAM image, a tool is available. For more details, please refer to [4]. This tool has a Graphical User Interface to help the user to build the SDMA RAM image, the GUI has the following aspect.



The next diagram illustrates a SDMA RAM image where the memory is 78% full and where the scripts that will be used in the application are organized in two sections: a memory to peripheral section and a custom section. But this image is just given as an example.

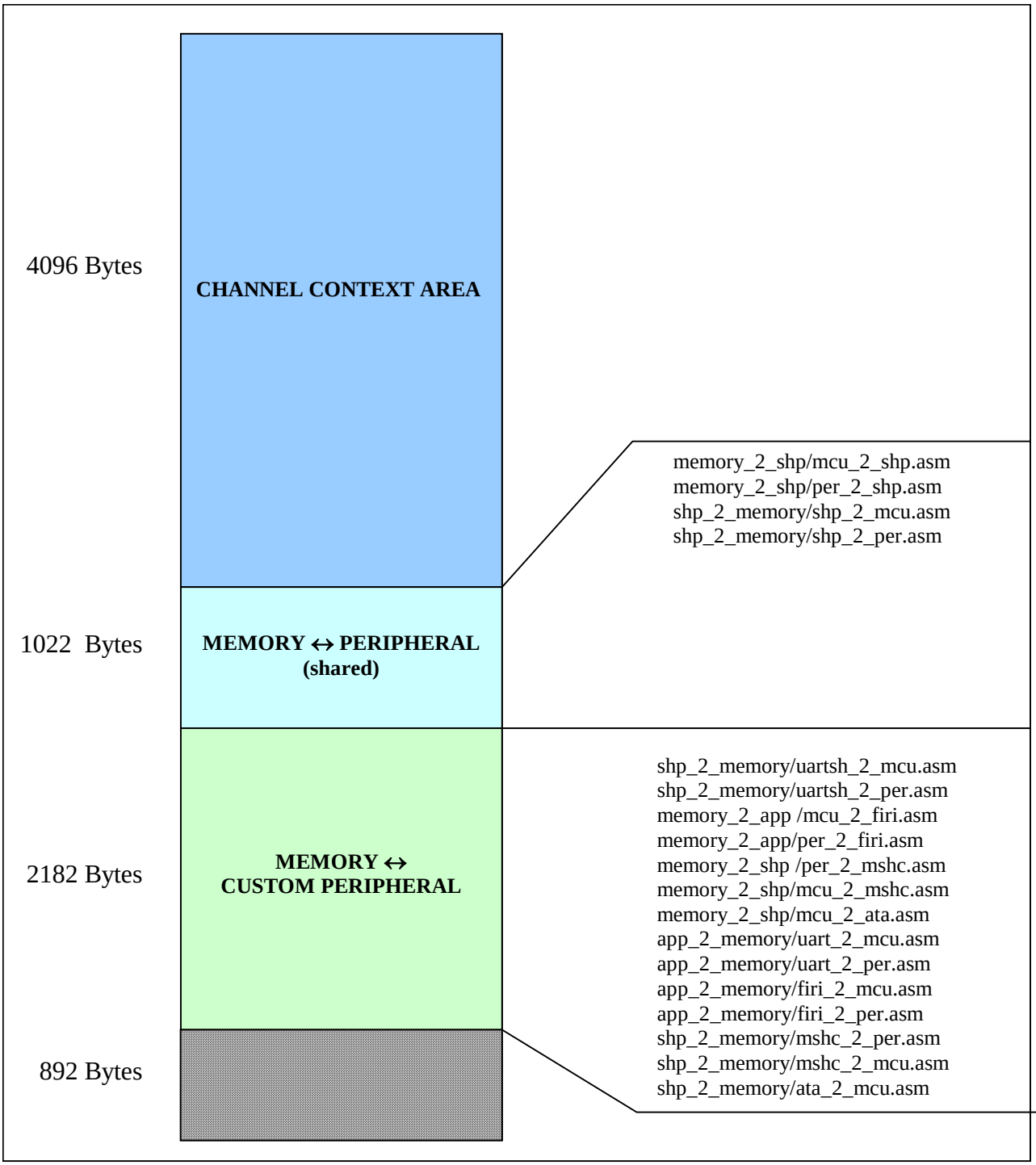


Figure 4 Layout of the 8Kbytes RAM