

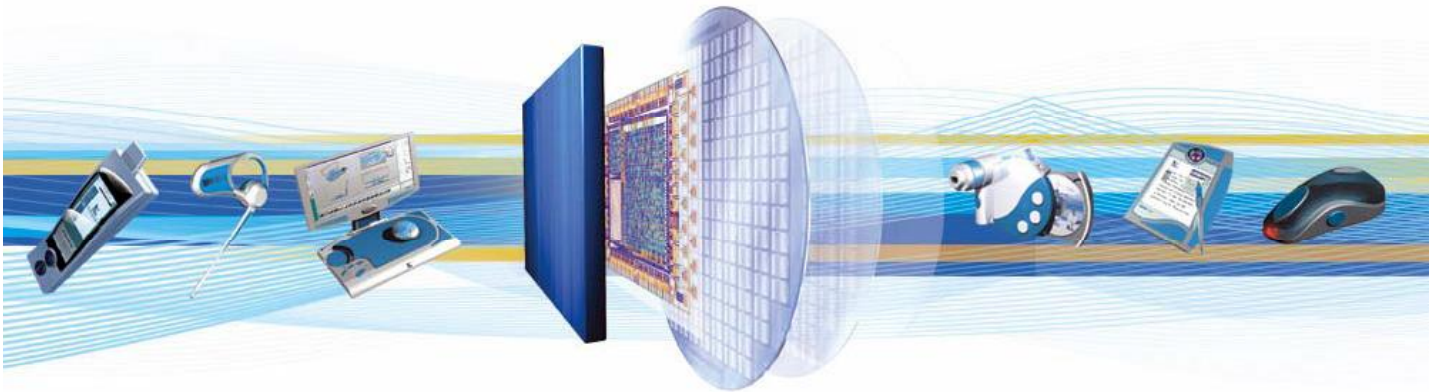


BlueCore™5-Multimedia

Kalimba DSP

User Guide

July 2006



CSR

Cambridge Science Park
Milton Road
Cambridge CB4 0WH
United Kingdom

Registered in England 4187346

Tel: +44 (0)1223 692000
Fax: +44 (0)1223 692001

www.csr.com



Contents

- 1 Introduction 6**
- 2 Key Features 7**
- 3 System Overview 8**
 - 3.1 Kalimba DSP Core 9
 - 3.2 Kalimba DSP Memory 9
 - 3.3 Kalimba DSP Peripherals 9
 - 3.3.1 Memory Management Unit Interface 9
 - 3.3.2 Programmable I/O Control 9
 - 3.3.3 Interrupt Control 9
 - 3.3.4 Clock Source Select and Timer 9
 - 3.3.5 Debug Interface 9
- 4 Kalimba DSP Core Architecture 10**
 - 4.1 Arithmetic Logic Unit 10
 - 4.2 Address Generators 10
 - 4.3 Registers 11
 - 4.4 Bank 1 Registers 11
 - 4.5 rFlags Register 12
 - 4.5.1 Negative Flag (N) 12
 - 4.5.2 Zero Flag (Z) 12
 - 4.5.3 Carry Flag (C) 12
 - 4.5.4 Overflow Flag (V) 12
 - 4.5.5 Sticky Overflow Flag (SV) 13
 - 4.5.6 User Definable Flag (UD) 13
 - 4.5.7 Bit Reverse Flag (BR) 13
 - 4.5.8 User Mode Flag (UM) 13
 - 4.5.9 Condition Codes 13
 - 4.6 rMAC Register 14
 - 4.7 Bank 2 Registers 15
 - 4.7.1 Index Registers 15
 - 4.7.2 Modify Registers 15
 - 4.7.3 Length Registers 15
 - 4.8 Special Bank 3 registers 16
 - 4.9 Program Flow 16
 - 4.10 Debug 17
- 5 Memory Organisation 18**
 - 5.1 Memory Map 18
 - 5.1.1 PM Memory Map 18
 - 5.1.2 DM1 Memory Map 19
 - 5.1.3 DM2 Memory Map 19
- 6 Instruction Set Description 21**
 - 6.1 ADD and ADD with CARRY 22
 - 6.2 SUBTRACT and SUBTRACT With Borrow 23
 - 6.3 Bank1/2 Register Operations: ADD and SUBTRACT 24
 - 6.4 Logical Operations: AND, OR and XOR 25
 - 6.5 Shifter: LSHIFT and ASHIFT 26
 - 6.6 rMAC Move Operations 27
 - 6.7 Multiply: Signed 24-Bit Fractional and Integer 28
 - 6.8 MULTIPLY and ACCUMULATE (56-bit) 29
 - 6.9 LOAD / STORE with Memory Offset 30
 - 6.10 Sign Bits Detect and Block Sign Bits Detect 31



6.11 Divide Instruction	32
6.12 PUSH, POP and PLOOK Stack Instructions.....	33
6.13 Program Flow: CALL, JUMP, RTS, RTI, SLEEP, DO...LOOP and BREAK.....	34
6.14 Indexed MEM_ACCESS_1 and MEM_ACCESS_2.....	35
7 Instruction Coding	37
7.1 Type A Instruction.....	38
7.2 Type B Instruction.....	38
7.3 Type C Instruction	38
7.4 Special Cases.....	38
7.5 OP_CODE Coding.....	39
7.6 AM Field	40
7.7 Carry Field (C Field)	40
7.8 Bank 1/2 Register Select Field (B2RS Field).....	40
7.9 Saturation Select Field (V Field).....	41
7.10 Sign Select Field (S Field).....	41
7.11 k_{16} Coding for LSHIFT and ASHIFT.....	41
7.12 rMAC Sub Registers.....	41
7.13 ASHIFT.....	42
7.14 LSHIFT	43
7.15 k_{16} Coding Divide Instructions.....	43
7.16 StackBankSelect field encoding	44
8 Kalimba DSP Peripherals	45
8.1 MMU Interface	46
8.1.1 Read Ports	46
8.1.2 Write Ports	46
8.2 DSP Timers	46
8.3 Kalimba Interrupt Controller.....	47
8.3.1 DSP Core Functionality During Interrupt	47
8.3.2 Interrupt Controller Functionality	47
8.4 Generation of MCU Interrupt	48
8.5 PIO Control.....	48
8.6 MCU Memory Windows in DM2.....	48
8.7 Flash Memory Windows in DM2.....	48
8.8 PM Window in DM1	49
8.9 PM Flash Window with 64-word Direct Cache.....	49
8.10 MCU I/O Map Memory-Mapped Interface.....	49
8.11 General Registers.....	49
8.12 Clock Rate Divider Control	49
8.13 Debugging	49
Appendix A: Number Representation	50
A.1 Binary Integer Representation	50
A.2 Binary Fractional Representation	50
A.3 Integer Multiplication.....	50
A.4 Fractional Multiplication	51
Appendix B: DSP Memory Mapped Registers	52
B.5 DSP Memory Mapped I/O.....	52
B.2 MMU Interface DSP Registers.....	57
B.3 Timers DSP Registers	59



B.4 Interrupt Controller DSP Registers 60

B.5 MCU Interrupt DSP Registers..... 68

B.6 PIO Control DSP Registers 68

B.7 MCU Window DSP Registers 72

B.8 Flash Window DSP Registers..... 72

B.9 PM Window DSP Registers 74

B.10 MCU IO Map Interface Registers..... 74

B.11 General DSP Registers 76

B.12 PM Flash Window Control Registers 77

B.13 Clock Divide Rate DSP Registers..... 78

B.14 Core State DSP Registers 79

B.15 Bitreverse Function Memory-mapped Register 81

B.16 Kalimba Add/Subtract Core Configuration Register..... 81

B.17 Stack Memory-mapped Registers 82

B.18 Debug Profiling Memory-mapped Registers 83

Appendix C: Software Examples 86

 C.1 Double-Precision Addition 86

 C.2 Fractional Double-Precision Multiply 86

 C.3 Integer Double-Precision Multiply 86

 C.4 FIR Filter..... 87

 C.5 Cascaded Bi-Quad IIR Filter..... 88

 C.6 Radix-2 FFT..... 89

Document References 92

Terms and Definitions 93

Document History 94

List of Figures

Figure 3.1: Kalimba DSP Co-Processor Subsystem 8

Figure 4.1: Kalimba DSP Core Base Architecture..... 10

Figure 4.2: rMAC Register 14

Figure 5.1: Memory Organisation..... 18

Figure 8.1: Kalimba DSP Peripheral Interfaces..... 45

Figure 8.2: Example of MMU Interface Usage for a Wireless MP3 Player 46

**List of Tables**

Table 4.1: Bank 1 Registers	11
Table 4.2: rFlags Register	12
Table 4.3: Condition Codes	13
Table 4.4: Bank 2 Registers	15
Table 4.5: Bank 3 Registers	16
Table 5.1: PM Memory Map	19
Table 5.2: DM1 Memory Map	19
Table 5.3: DM2 Memory Map	20
Table 6.1: Notational Convention	21
Table 7.1: Instruction Coding Format	37
Table 7.2: OPCODE Coding Format	39
Table 7.3: AM Field	40
Table 7.4: C Field Options	40
Table 7.5: B2RS Field	40
Table 7.6: V Field	41
Table 7.7: S Field	41
Table 7.8: k_{16} Coding Shift Format	41
Table 7.9: rMAC Sub-Registers	41
Table 7.10: ASHIFT	42
Table 7.11: LSHIFT	43
Table 7.12: Divide Field	43
Table 7.13: Divide Field States	43



1 Introduction

The **BlueCore™5-Multimedia** Kalimba DSP User Guide is for developers of software applications and algorithms for the DSP (Digital Signal Processor) co-processor on the BlueCore5-Multimedia device. It documents the architecture of the Kalimba DSP, instruction set mnemonics and peripheral features, and includes some example code. Read this document in conjunction with the other Kalimba tools documents that are available:

- BlueLab Kalimba DSP Assembler User Guide (CSR reference bc5-ug-002P) which describes the assembler software
- BlueLab xIDE User Guide (CSR reference blab-ug-002P) which describes xIDE, the software debugging tool

One of the features of the BlueCore5-Multimedia device is an on-chip DSP co-processor, Kalimba. The Kalimba DSP particularly targets audio processing applications for BlueCore. The likely audio processing applications include:

- Sub-Band Coding (SBC) encoding and decoding, as defined in the Bluetooth Advanced Audio Distribution Profile
- MP3 encoding and decoding, as defined in ISO/IEC 11172-3, and the sample rate extensions defined in ISO/IEC 13818-3
- Advanced Audio Coding (AAC) encoding and decoding, as defined in ISO/IEC 13818-7
- Alternative voice/Hi-Fi CODECs
- Echo and noise cancellation
- Audio signal enhancement
 - Stereo enhancement
 - Equaliser
 - Lost packet concealment
- Text to speech
- Voice recognition



2 Key Features

The key features of the Kalimba DSP core include:

- 24-bit fixed point Kalimba DSP core
- 64MIPS performance which can be divided down for power saving
- 32-bit instruction word, dual 24-bit data memory:
 - 16-bit program address space with 6Kword (6K x 32-bits) of physical RAM, external flash is addressable with a 64-word direct cache
 - 16-bit data address space in two banks with 16Kword and 12Kword (x 24-bits) of physical RAM
- Up to two data memory accesses can be performed in the same cycle as a program memory read
- Single-cycle 24 x 24-bit multiply with 56-bit accumulator
- Single-cycle barrel shifter with 56-bit input and 24-bit output
- 12-cycle divide (performed in the background)
- Majority of instructions can be conditional
- Zero overhead ring buffer indexing
- Zero overhead looping and unconditional branching
- Bit reversed addressing capability, and bit reverse data function
- Largely orthogonal instruction set, which is quick to learn and easy to write in algebraic assembler language
- Stack instructions: PUSH, POP, PLOOK and overflow detection
- Low-power internal architecture

The key features of the Kalimba DSP peripherals include:

- Close integration into the rest of the BlueCore5-Multimedia
- Eight low-overhead read/write ports to transfer streaming data to and from the BlueCore5-Multimedia subsystem
- Four shared memory-mapped registers
- Two memory-mapped windows into the BlueCore5-Multimedia MCU RAM for data exchange
- Three windows for access to the Flash data memory
- Flash Program memory interface with 64-word direct cache
- Memory-mapped interface to BlueCore5-Multimedia I/O address map
- Multiple interrupt sources including two 24-bit timers
- Read, write and direction control access to external programmable I/O (PIO) lines

3 System Overview

The BlueCore5-Multimedia contains the Kalimba DSP shown in Figure 3.1 and consists of the following functional elements:

- Kalimba DSP core
- DSP memory, this RAM is used for:
 - Data memory
 - Program memory
 - Memory mapped I/O
- Memory management unit (MMU) interface
- Programmable I/O control
- Interrupt control
- Clock source
- Timer
- Debug interface

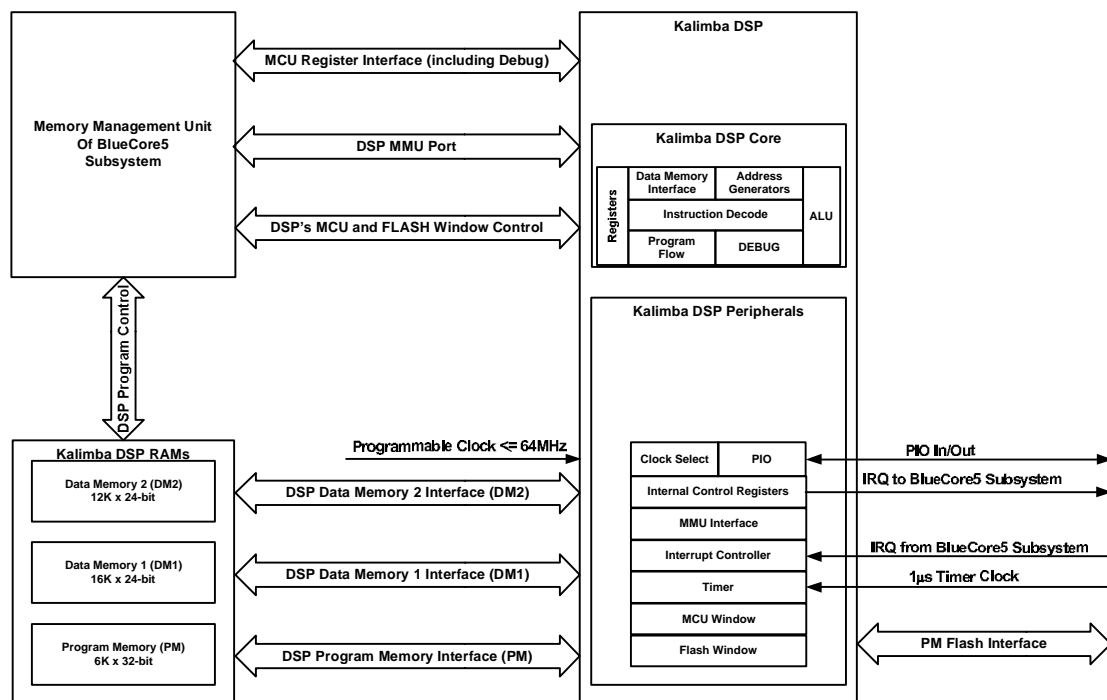


Figure 3.1: Kalimba DSP Co-Processor Subsystem

3.1 Kalimba DSP Core

The Kalimba DSP core is an open-platform DSP that can perform signal processing functions on over-air data or audio CODEC data to enhance audio applications. Figure 3.1 and Section 0 show how the DSP interfaces to other functional blocks within BlueCore5-Multimedia.

3.2 Kalimba DSP Memory

The Kalimba DSP contains the following on-chip RAM:

- 16K x 24-bit for data memory 1 (DM1)
- 12K x 24-bit for data memory 2 (DM2)
- 6K x 32-bit for program memory (PM) with 64-word direct cache for Flash program space

3.3 Kalimba DSP Peripherals

3.3.1 Memory Management Unit Interface

The MMU Interface consists of a series of virtual read and write ports that stream transfers to/from the rest of the BlueCore5-Multimedia.

3.3.2 Programmable I/O Control

BlueCore5-Multimedia has up to 24 programmable I/O lines controlled by firmware running on the device. The Kalimba DSP core can read any digital I/O directly but can only write to or change the pin direction of digital outputs that the MCU has enabled. (This is done through the VM application.) A full description of the I/O control is in Section 8.5.

3.3.3 Interrupt Control

The interrupt controller function within the Kalimba DSP covers interrupt control of the Kalimba DSP core. It allows interrupt sources selection and control of their priority setting within three levels. Alongside the interrupts caused by hardware, there are four software event interrupts available, with a further 16 interrupt events that directly correspond to the 16 interrupt events available to the Bluecore5-Multimedia MCU.

3.3.4 Clock Source Select and Timer

The Kalimba DSP consists of a clock source select interface, which is a clock-rate divider circuit that is controllable from both the Kalimba DSP core and the on-board MCU. The Kalimba DSP also has two timers with a 1 μ s time base available.

3.3.5 Debug Interface

The BlueCore5-Multimedia contains a hardware interface that assists in the debugging of applications running on the Kalimba DSP core.

4 Kalimba DSP Core Architecture

Figure 4.1 shows the Kalimba DSP core architecture.

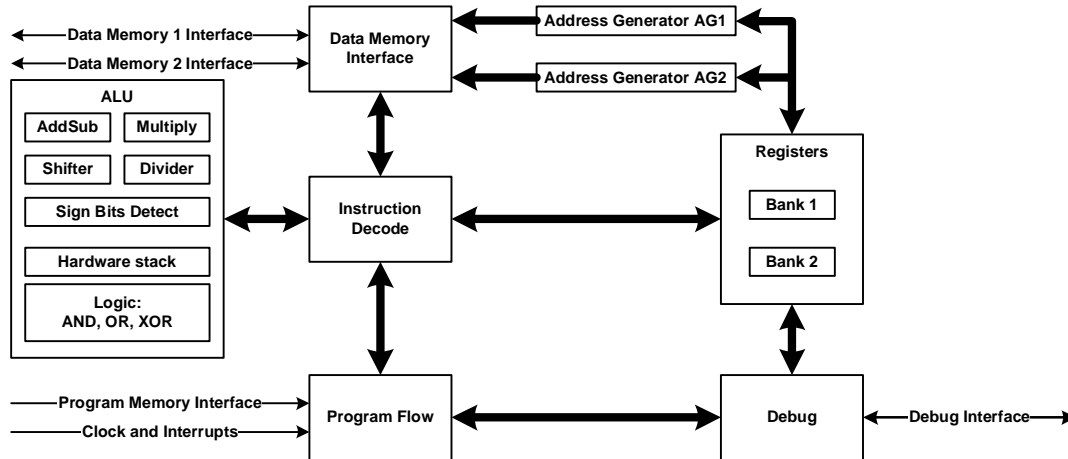


Figure 4.1: Kalimba DSP Core Base Architecture

4.1 Arithmetic Logic Unit

The arithmetic logic unit (ALU) performs the following functions:

- Add and subtract arithmetic
- Logic operations:
 - AND, OR, and XOR
- Single-cycle multiply, multiply/add and multiply/subtract
- Logical and arithmetic shift operations with a 56-bit input and 24-bit output
- Derive exponent and block derive exponent operations, which detect the number of redundant sign bits
- Signed divide, taking a 48-bit dividend (numerator) and a 24-bit divisor (denominator)⁽¹⁾
- Stack instruction control: PUSH, POP and PLOOK with overflow detection

Note:

- ⁽¹⁾ This operation is performed in the background, taking 12 clock cycles.

4.2 Address Generators

The address generators (AGs) form data memory addresses for indexed memory reads or writes. Each AG has four associated address pointers (index registers). When an index register is used for a memory access, it is post-modified by a value in a specified modify register, or by a 2-bit constant. With two independent AGs, the DSP can generate two addresses simultaneously for dual indexed memory accesses.

Length values may be associated with four of the index registers to implement automatic modulo addressing for circular buffers.

When the appropriate mode bit is set in the `rFlags` register, the output of AG1 is bit-reversed then driven on to the address bus. This feature enables addressing in radix-2 Fast Fourier Transform (FFT) algorithms; see the Radix-2 FFT code in appendix C for an example of its use.

4.3 Registers

There are two banks of 16 registers, with an additional bank of special registers for use with the stack instructions:

- Bank 1 are general registers described in Section 4.4 that can be used by virtually all instructions.
- Bank 2 registers are used for the control of index memory accesses described in Section 4.5.

4.4 Bank 1 Registers

Bank 1 registers are general registers used by virtually all instructions.

No.	Name	# bits	Description
0	Null	NA	Always read as zero, writing only affects flags (hence can be used for condition testing)
1	rMAC	56	The 56-bits are used for multiply accumulate instructions and the input to shift operations. For 24-bit operations: <ul style="list-style-type: none"> ▪ Read as bits [47:24] (with saturation and unbiased rounding)⁽¹⁾, ▪ Written as bits [47:24] with sign extension and trailing zero padding to make 56-bits
2	r0	24	General register
3	r1	24	General register
4	r2	24	General register
5	r3	24	General register
6	r4	24	General register
7	r5	24	General register
8	r6	24	General register
9	r7	24	General register
10	r8	24	General register
11	r9	24	General register
12	r10	24	General register and is used as the loop counter for zero overhead loops
13	rLink	16	Call instructions put the return PC address in this register for use by rts instructions ⁽²⁾
14	rFlags	16	Status and mode flags see below for a description
15	rIntLink	16	The return PC address is stored in this register for use by rti instructions

Table 4.1: Bank 1 Registers

Notes:

Only registers rMAC, r0-r5 can be used by indexed memory access instructions.

16-bit registers (rLink, rFlags and rIntLink) are zero padded to 24-bit. Writing to them does not affect the flags register.

All registers are set to 0 (zero) on DSP reset:

- ⁽¹⁾ Unbiased rounding is as follows:

```
rMACrounded = rMAC[47:24] + rMAC[23];
if (rMAC[23:0] == 0x800000) then rMACrounded[0] = 0;
```

This has the effect of rounding odd rMAC[47:24] values away from zero and even rMAC[47:24] value towards zero, yielding a zero large sample bias assuming uniformly distributed values.

- ⁽²⁾ There is no hardware subroutine stack, so to allow multi depth subroutine calls a software stack implementation or equivalent is required.

4.5 rFlags Register

The `rFlags` register is a 16-bit register that is located in register bank 1 of Kalimba shown in Table 4.1. The individual bits that make up the `rFlags` register and their value after reset are as shown in Table 4.2. This register has a natural split into two bytes:

- The least significant byte contains the presently active flags used by Kalimba.
- The most significant byte contains a stored value of the flags.

The flags are stored when an interrupt has occurred and then restored back to these values after Kalimba has finished servicing the interrupt. Interrupts are serviced three instructions after the interrupt request line goes high (not including prefix instructions). The `INT_` versions of the various flags are the copies that are stored at the point of interrupt service. The `rti` instruction then automatically restores the flags to their previous value.

Name	INT_UM_FLAG	INT_BR_FLAG	INT_SV_FLAG	INT_UD_FLAG	INT_V_FLAG	INT_C_FLAG	INT_Z_FLAG	INT_N_FLAG	UM_FLAG	BR_FLAG	SV_FLAG	UD_FLAG	V_FLAG	C_FLAG	Z_FLAG	N_FLAG
Reset State	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Table 4.2: rFlags Register

4.5.1 Negative Flag (N)

This is set if the result of the instruction is negative (most significant bit set), and cleared otherwise.

4.5.2 Zero Flag (Z)

This is set if the result of the instruction is zero, and cleared otherwise.

4.5.3 Carry Flag (C)

The state of the carry flag is:

- For an addition, C is set if the addition produced a carry (that is, an unsigned overflow), and is cleared otherwise.
- For a subtraction, C is cleared if the subtraction produced a borrow (that is, an unsigned underflow), and is set otherwise.
- For other operations (including multiply accumulate), C is left unchanged.

4.5.4 Overflow Flag (V)

The state of the overflow flag is:

- For addition, subtraction, arithmetic shifts, integer multiplies and multiply accumulates, V is set if signed overflow occurred, regarding the operands and result as two's complement signed integers, and is cleared otherwise.
- The setting/clearing of the V flag for the `rMAC` register occurs if there is overflow past the 56th bit, whereas for the 24-bit registers it is if overflow occurs past the 24th bit. When writing to the 16-bit registers overflow has limited meaning so the V flag remains unchanged.
- For other operations, V is left unchanged.



4.5.5 Sticky Overflow Flag (SV)

Set whenever the V flag is set but can only be cleared in software by explicitly writing to the rFlags register.

4.5.6 User Definable Flag (UD)

A special USERDEF condition code is TRUE if this flag is set and FALSE if this flag is clear. Use it in the code sections to improve speed and code clarity where optionally a particular instruction needs to execute.

4.5.7 Bit Reverse Flag (BR)

If set, the output of Address Generator 1 (index registers I0-I3) is bit-reversed before being driven to the address bus.

4.5.8 User Mode Flag (UM)

If set, interrupts are serviced. On entry to the interrupt service routine (PC address 0x0002), this flag is cleared and so no further interrupts are serviced unless the flag is manually set for example to support interrupt priority. Execution of a return from interrupt (rti) instruction sets this flag to the value of INT_UM_FLAG (normally set unless altered in software). During an interrupt service routine, the INT_UM_FLAG contains the stored value of the user mode flag prior to the interrupt.

4.5.9 Condition Codes

The state of the flags present in the rFlags register forms the basis of the condition codes in Table 4.3 for the Kalimba DSP.

Condition	Condition Flag State	Condition Code
Z (Zero) / EQ (Equal)	Z = 1	0 0 0 0
NZ (Not Zero) / NE (Not equal)	Z = 0	0 0 0 1
C (ALU carry) / NB (Not ALU borrow)	C = 1	0 0 1 0
NC (Not ALU carry) / B (ALU borrow)	C = 0	0 0 1 1
NEG (Negative)	N = 1	0 1 0 0
POS (Positive)	N = 0	0 1 0 1
V (ALU overflow)	V = 1	0 1 1 0
NV (Not ALU overflow)	V = 0	0 1 1 1
HI (unsigned higher)	C = 1 AND Z = 0	1 0 0 0
LS (unsigned lower or same)	C = 0 OR Z = 1	1 0 0 1
GE (Signed Greater than or equal)	N = V	1 0 1 0
LT (Signed Less than)	N != V	1 0 1 1
GT (Signed Greater than)	Z = 0 AND N = V	1 1 0 0
LE (Signed Less than or equal)	Z = 1 OR N != V	1 1 0 1
USERDEF (user defined)	USERDEF = 1	1 1 1 0
Always true	don't care	1 1 1 1

Table 4.3: Condition Codes

4.6 rMAC Register

The rMAC register is a 56-bit register that is located in register bank 1 of Kalimba indicated in Table 4.1. The rMAC register splits into a set of separately accessible sub-registers. Figure 4.2 lists the size of these registers:

- rMAC is the overall 56-bit register
- rMAC0 is a 24-bit register that forms the lower part of the rMAC register
- rMAC1 is a 24-bit register that forms the middle part of the rMAC register
- rMAC2 is a 8-bit register that forms the higher part of the rMAC register
- rMAC12 is a 32-bit register that is a combination of rMAC2 and rMAC1 that forms part of the rMAC register

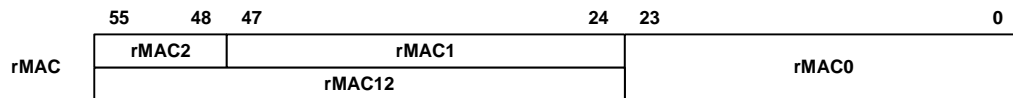


Figure 4.2: rMAC Register

4.7 Bank 2 Registers

Bank 2 registers are for the control of index memory accesses, and set up modulo addressing through a circular memory buffer.

No.	Name	# bits	Description
0	I0	16	Index register for Address Generator 1 (AG1)
1	I1	16	Index register for AG1
2	I2	16	Index register for AG1
3	I3	16	Index register for AG1
4	I4	16	Index register for Address Generator 2 (AG2)
5	I5	16	Index register for AG2
6	I6	16	Index register for AG2
7	I7	16	Index register for AG2
8	M0	16	Modify register for any index register
9	M1	16	Modify register for any index register
10	M2	16	Modify register for any index register
11	M3	16	Modify register for any index register
12	L0	16	Length register for Index register I0
13	L1	16	Length register for Index register I1
14	L4	16	Length register for Index register I4
15	L5	16	Length register for Index register I5

Table 4.4: Bank 2 Registers

Note:

- All Bank 2 registers are sign extended to 24-bit for arithmetic operations
- All registers are set to 0 (zero) on DSP reset

4.7.1 Index Registers

The index register contains the address pointers to data memory used with indexed addressing. These registers allow transfer of data to and from selected Bank1 registers using the address contained within this register. See Section 6.14 for further details. The index registers I0 to I3 are associated with AG1 and I4 to I7 are associated with AG2.

4.7.2 Modify Registers

When an index register is used for a memory access, it can be post-modified by a value contained in the modify register, or by a 2-bit constant.

4.7.3 Length Registers

Length register values are associated with four of the index registers, as listed in Table 4.4, to implement automatic modulo addressing for circular buffers. To disable automatic modulo addressing set the corresponding length register to zero.

4.8 Special Bank 3 registers

These additional registers can be used by the stack instructions to accelerate pushing and popping registers to the stack, particularly for interrupt routine entry and exit.

No.	Name	# bits	Description
0	RMAC0	24	The LS 24 bits of rMAC
1	RMAC1	24	The middle 24 bits of rMAC
2	RMAC2	8	The MS 8 bits of rMAC
3	DoLoopStart	16	Start address for do Loops. Identical to memory-mapped MM_DOLOOP_START
4	DoLoopEnd	16	End address for do Loops. Identical to memory-mapped MM_DOLOOP_END
5	DivResult	24	Divider result
6	DivRemainder	24	Divider remainder

Table 4.5: Bank 3 Registers

Note:

All special Bank3 registers are only accessible with the stack instructions. Pushing `DivResult` or `DivRemainder` will stall the DSP until any background divides have completed, popping will cancel any current background divides.

All registers are set to 0 (zero) on DSP reset.

4.9 Program Flow

The program counter (PC) supplies addresses to the program memory. An instruction register holds the currently executing instruction. This instruction register introduces a single level of pipelining into the program flow. During one clock cycle, the instruction register has instructions fetched and loaded into it and during the following cycle they are executed. To allow zero overhead branching and no pipeline hazards with memory reads the processor also acts on the direct signal from the memory, i.e. the next instruction executed, to set up the address bus and control lines for a memory read. However, certain instructions have stalls imposed to reduce long combinatorial paths within the DSP. These stalls occur whenever any of the following happens:

- A conditional branch occurs
- A main-instruction memory read occurs where the result of the previous instruction is used as the address for the next instruction
- A conditional memory read occurs
- The value of `r10` is set immediately before the start of a `Do ... Loop`
- An `rts` occurs immediately after changing `rLink`
- An `rti` occurs immediately after changing `rIntLink`
- An index memory read occurs on either AG1 or AG2 where the corresponding index register is modified immediately before the indexed read
- The bitreverse (BR) flag is altered immediately before an AG1 indexed memory access

Hence, there are stall cycles in the following code examples:

- Stall due to conditional branch:

- ```

r0 = r0 - 1;
if POS jump dont_add_ten;
 r0 = r0 + 10;
dont_add_ten:

```
- Stall due to main-instruction memory read:
 

```

r0 = 100;
r1 = M[r0];

```
  - Stall due to modifying r10 before a do ... loop:
 

```

r10 = 100;
do Loop; //Stall here due to modifying r10 before do..loop
 r0 = r0 + 1;
Loop:

```
  - Stall due to setting index register before a memory read:
 

```

I0 = 100;
r1 = r1 + r2
 r0 = M[I0, 1]; //Stall here since I0 is set up immediately before

```
  - Stall due to changing rLink before an rts:
 

```

POP rLink;
rts; //stall here due to modifying rLink before rts

```

Another feature of the program flow of the Kalimba DSP is the hardware zero overhead looping instruction. The register `r10` is loaded with the number of times that the code between the `do` instruction and the `loop` label executes. `r10` is then automatically decremented and a jump taken (if needed) at the same time as executing the instruction before the 'loop' label. For example:

- Zero overhead looping:
 

```

r10 = 10;
r1 = 4; // A stall would occur if this wasn't here
do loop; // copies 10 words of data from
 r0 = M[I0,1]; // address I0 to address I2.
 M[I2,1] = r0; // Takes 22 cycles in total.
loop:

```

## 4.10 Debug

There is debugging hardware in the Kalimba DSP used by the debugger, `xIDE` or `kaldbg`. It provides the following features:

- Reset, Run, Stop, Step
- Setting and reading of the program counter (PC)
- Program breakpoint
- Data memory breakpoint (read, write, or read/write)
- Instruction Break
- Read/write of register values
- External to the DSP core itself there is debug circuitry to read/write memory locations as seen by the DSP on either of its 3 memory buses PM, DM1 and DM2
- Read/clear profiling counters:
  - Number of clocks
  - Number of instructions executed
  - Number of stall cycles

## 5 Memory Organisation

The Kalimba DSP core has two 24-bit data memory banks, DM1 and DM2, each having a 15-bit address space, and a single 32-bit program memory (PM) having a 16-bit address space. Accessing all three memories simultaneously is possible in the same clock cycle, assuming no conflicts; this is a three-bank Harvard architecture. Conflicts introduce an appropriate number of wait cycles. Figure 5.1 is a view of the Kalimba DSP memory organisation containing size information and peripheral memory mapping.

BlueCore5-Multimedia has the following physical RAM for the Kalimba DSP:

- DM1 = 16K x 24-bit
- DM2 = 12K x 24-bit
- PM = 6K x 32-bit

The BlueCore5-Multimedia subsystem MCU initialises the Kalimba DSP. During initialisation program and data coefficient download to the DSP occurs through auto-incrementing memory-mapped registers in the MCU. The MCU then sets the initial clock frequency for the Kalimba DSP to use before starting it running. API calls from the virtual machine (VM) running on the MCU invokes the program download, and Kalimba DSP initialisation.

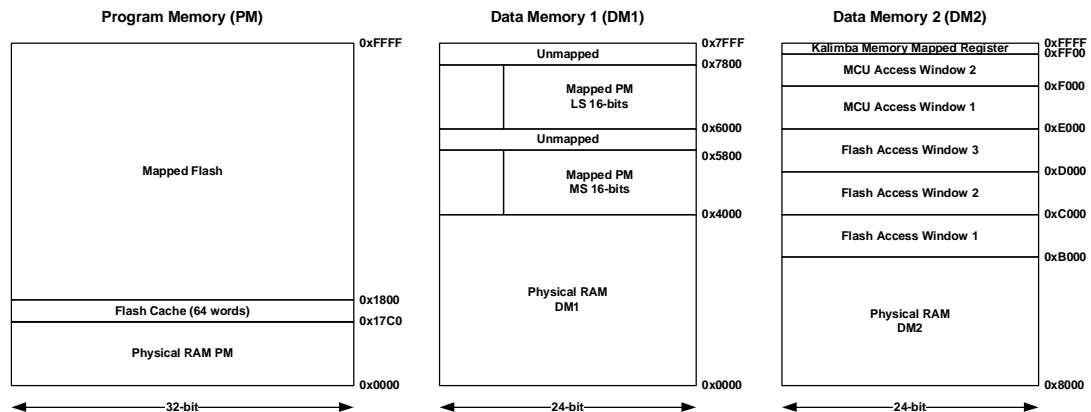


Figure 5.1: Memory Organisation

### 5.1 Memory Map

The memory organisation shown in Figure 5.1 can be broken down into their individual memory maps as shown in the tables in Section 5.1.1 to Section 5.1.3 (and Appendix B Table 1 for the DSP memory map).

#### 5.1.1 PM Memory Map

The program memory map for Kalimba shown in Table 5.1 contains 6Kwords (6K x 32-bits) of physical program memory RAM. The highest 64 words of RAM may be used as cache for the PM flash space. The remaining 58Kwords can be mapped into flash. Memory-mapped registers in the DSP decide the flash start address and size of the region of flash that is mapped. The MCU can control the priority given to flash accesses between the Kalimba DSP and MCU.

| Data   |        | Length (Words) | Description                                                        |
|--------|--------|----------------|--------------------------------------------------------------------|
| Start  | End    |                |                                                                    |
| 0x0000 | 0x17BF | 6K (-64words)  | General-purpose RAM (program memory)                               |
| 0x17C0 | 0x17FF | 64words        | Cache for flash PM space, or general-purpose RAM if flash not used |
| 0x1800 | 0xFFFF | 58Kwords       | Mapped to flash                                                    |

**Table 5.1: PM Memory Map**

### 5.1.2 DM1 Memory Map

The first data memory bank DM1 has a memory map shown in Table 5.2 containing:

- 16Kwords (16K x 24-bits) of data memory for the DSP
- A window of 6Kwords of program memory split into the most and least significant halves
- The remaining areas of the memory map allow for future variants

| Data   |        | Length (Words) | Description                                    |
|--------|--------|----------------|------------------------------------------------|
| Start  | End    |                |                                                |
| 0x0000 | 0x3FFF | 16K            | General purpose RAM (data memory 1)            |
| 0x4000 | 0x57FF | 6K             | Mapped to program memory MS-16bits             |
| 0x5800 | 0x5FFF | 2K             | Available for RAM expansion in future variants |
| 0x6000 | 0x77FF | 6K             | Mapped to program memory LS-16bits             |
| 0x7800 | 0x7FFF | 2K             | Available for RAM expansion in future variants |

**Table 5.2: DM1 Memory Map**

### 5.1.3 DM2 Memory Map

The second data memory bank DM2 memory map shown in Table 5.3 has the following distinct areas:

- 12Kwords (12K x 24-bits) of general purpose data RAM for the DSP.
- Three 4Kwords windows into the flash memory, which could be used by the DSP for items such as slower access coefficient tables.
- There are two MCU windows 4K and 3.75Kwords into the MCU memory to allow for control information and message passing.
- The final 256words of the memory map are reserved for the memory mapped I/O for the DSP that is explained further in Appendix B: DSP Memory Mapped Registers.



| Data   |        | Length<br>(Words) | Description                                                                               |
|--------|--------|-------------------|-------------------------------------------------------------------------------------------|
| Start  | End    |                   |                                                                                           |
| 0x8000 | 0xAFFF | 12K               | General-purpose RAM (data memory 2)                                                       |
| 0xB000 | 0xBFFF | 4K                | Flash window 1                                                                            |
| 0xC000 | 0xCFFF | 4K                | Flash window 2                                                                            |
| 0xD000 | 0xDFFF | 4K                | Flash window 3                                                                            |
| 0xE000 | 0xEFFF | 4K                | Window into MCU memory 1                                                                  |
| 0xF000 | 0xFEFF | 4K-256            | Window into MCU memory 2                                                                  |
| 0xFF00 | 0xFFFF | 256               | DSP Memory-mapped I/O registers (See Appendix B: DSP Memory Mapped Registers for details) |

Table 5.3: DM2 Memory Map

## 6 Instruction Set Description

The instruction set for the BlueCore5-Multimedia is an algebraic assembler instruction set compared to the mnemonic assemblers found in traditional microcontrollers. Algebraic assemblers suit the architecture of a DSP, as it is able to express complex and parallel instructions in an understandable way. This section describes each instruction in more detail. Table 6.1 outlines the notation conventions used in describing the syntax:

|                                    |                                                                                                                                                                                                                    |
|------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Parallel lines                     | Vertical parallel bars enclose lists of syntax options. One of the choices listed must be chosen.                                                                                                                  |
| Angled brackets <non bold italics> | Anything in <i>non bold italics</i> enclosed by angled brackets is an optional part of the instruction statement.                                                                                                  |
| A, B, C                            | Denotes a register operand. By default, the register must be chosen from the list of Bank1 registers. If subscribed with 'Bank1/2' then the register can be chosen from either Bank1 or Bank2                      |
| $k_{16}, k_8$                      | Denotes a constant, the subscripted number being the size of the number in bits.                                                                                                                                   |
| M[x]                               | Means the data in the memory location with address 'x'.                                                                                                                                                            |
| M[i,m]                             | Means the data in the memory location with address 'i', and that after the read/write the value of the register 'i' is modified according to the equation: $i=i+m$ ;                                               |
| cond                               | A condition code from Table 4.3, e.g. NZ                                                                                                                                                                           |
| MEM_ACCESS_1<br>MEM_ACCESS_2       | Represents a memory access instruction that can be appended to an instruction where indicated. The valid memory access instructions that can be appended are covered by the Indexed MEM_ACCESS_1/2 in Section 6.14 |

**Table 6.1: Notational Convention**

**Important Note:**

The Kalimba DSP architecture has been designed to carry out single-cycle instructions. Any exceptions are noted in the description for the instruction, e.g. the divide instruction and in section 4.9.

## 6.1 ADD and ADD with CARRY

### Syntax:

|                 |                                                                                       |                           |                                                                        |
|-----------------|---------------------------------------------------------------------------------------|---------------------------|------------------------------------------------------------------------|
| <b>Type A:</b>  | $C = A + B$<br>$C = A + M[B]$<br>$C = M[A] + B$<br>$M[C] = A + B$                     | $\langle +Carry \rangle$  | $\langle MEM\_ACCESS\_1 \rangle;$                                      |
| <b>Example:</b> | $if\ Z\ r3 = r1 + M[r2] + Carry$<br>$r4 = M[I0, M0];$                                 |                           |                                                                        |
| <b>Type B:</b>  | $C = A + k_{16}$<br>$C = A + M[k_{16}]$<br>$C = M[A] + k_{16}$<br>$M[k_{16}] = A + C$ | $\langle +Carry \rangle;$ |                                                                        |
| <b>Example:</b> | $r3 = M[r1] + 10 + Carry;$                                                            |                           |                                                                        |
| <b>Type C:</b>  | $C = C + A$<br>$C = C + M[A]$                                                         | $\langle +Carry \rangle$  | $\langle MEM\_ACCESS\_1 \rangle \quad \langle MEM\_ACCESS\_2 \rangle;$ |
| <b>Example:</b> | $r3 = r3 + M[r1] + Carry$<br>$r4 = M[I0, M0]$<br>$r5 = M[I4, M1];$                    |                           |                                                                        |

### Description:

Test the optional condition and, if TRUE, perform the addition. If the condition is FALSE, perform a no-operation (NOP) but *MEM\_ACCESS\_1* still carried out. Omitting the condition performs the addition unconditionally. The addition operation adds the first source operand to the second source operand and, if designated by the “+Carry” notation, adds the ALU carry bit, C. The result is stored in the destination operand. The operands may be either one of the 16 Bank1 registers, a 16-bit sign extended constant (24-bit with prefix instruction), or memory pointed to by the register or a constant.

### Flags Generated:

|   |                                                            |   |                                                     |
|---|------------------------------------------------------------|---|-----------------------------------------------------|
| Z | Set if the result equals zero and cleared otherwise        | N | Set if the result is negative and cleared otherwise |
| V | Set if an arithmetic overflow occurs and cleared otherwise | C | Set if a carry is generated and cleared otherwise   |

### Note:

If one of the source operands is Null then a load/store is assumed. Therefore, the C and V flags are unchanged.

If all operands are Null then a NOP is assumed. Therefore, all flags are unchanged.

When writing to *rLink* the result is stored in *rLink* and so the flags are not applicable.

Additions and subtractions may be saturated on overflow by setting the ADDSUB\_SATURATE\_ON\_OVERFLOW memory-mapped register. This causes saturation of signed numbers (positive saturation to 0x7FFFFFFF, negative saturation to 0x800000).

## 6.2 SUBTRACT and SUBTRACT With Borrow

### Syntax:

**Type A:**  $\langle \text{if cond} \rangle$   $\left\{ \begin{array}{l} C = A - B \\ C = A - M[B] \\ C = M[A] - B \\ M[C] = A - B \end{array} \right. \left| \begin{array}{l} \langle \text{-Borrow} \rangle \\ \langle \text{MEM\_ACCESS\_1} \rangle; \end{array} \right.$

**Example:**  $\text{if } Z \text{ } r3 = r1 - r2 - \text{Borrow}$   
 $r4 = M[I0, M0];$

**Type B:**  $\left\{ \begin{array}{l} C = A - k_{16} \\ C = A - M[k_{16}] \\ C = M[A] - k_{16} \\ M[k_{16}] = A - C \\ C = k_{16} - A \end{array} \right. \left| \begin{array}{l} \langle \text{-Borrow} \rangle; \end{array} \right.$

**Example:**  $r3 = M[r1] - 10 - \text{Borrow};$

**Type C1/2:**  $\left\{ \begin{array}{l} C = C - A \\ C = C - M[A] \end{array} \right. \left| \begin{array}{l} \langle \text{-Borrow} \rangle \\ \langle \text{MEM\_ACCESS\_1} \rangle \\ \langle \text{MEM\_ACCESS\_2} \rangle; \end{array} \right.$

**Example:**  $r3 = r3 - r1 - \text{Borrow}$   
 $r4 = M[I0, M0]$   
 $r5 = M[I4, M1];$

### Description:

Test the optional condition and, if TRUE, perform the subtraction. If the condition is FALSE, perform a NOP, but *MEM\_ACCESS\_1* still carried out. Omitting the condition performs the subtraction unconditionally. The subtraction operation subtracts the second source operand from the first source operand and optionally, if designated by the “- Borrow” notation, subtracts the inverse of the ALU carry bit, C. The result is stored in the destination operand. The operands may be either one of the 16 Bank1 registers, a 16-bit sign extended constant (24-bit with prefix instruction), or memory pointed to by the register or constant.

### Flags Generated:

|   |                                                            |   |                                                     |
|---|------------------------------------------------------------|---|-----------------------------------------------------|
| Z | Set if the result equals zero and cleared otherwise        | N | Set if the result is negative and cleared otherwise |
| V | Set if an arithmetic overflow occurs and cleared otherwise | C | Cleared if a borrow is generated and set otherwise  |

### Note:

If one of the source operands is Null then a negate is assumed. The C and V flags are set or cleared as appropriate. Note the difference between BlueCore5-Multimedia and BlueCore3-Multimedia, where the C and V flags are left unchanged.

When writing to rLink the result is stored in rLink and so the flags are not applicable.

Additions and subtractions may be saturated on overflow by setting the ADDSUB\_SATURATE\_ON\_OVERFLOW memory mapped register. This causes saturation of signed numbers (positive saturation to 0x7FFFFFFF, negative saturation to 0x800000).

### 6.3 Bank1/2 Register Operations: ADD and SUBTRACT

#### Syntax:

**Type A:**  $\langle \text{if cond} \rangle \left\{ \begin{array}{l} C_{\text{BANK1/2}} = A_{\text{BANK1/2}} + B_{\text{BANK1/2}} \\ C_{\text{BANK1/2}} = A_{\text{BANK1/2}} - B_{\text{BANK1/2}} \end{array} \right. \langle \text{MEM\_ACCESS\_1} \rangle;$

**Example:** `if Z I0 = I4 + r2  
r1 = M[I1, M1];`

**Type B:**  $\left\{ \begin{array}{l} C_{\text{BANK1/2}} = A_{\text{BANK1/2}} + k_{16} \\ C_{\text{BANK1/2}} = A_{\text{BANK1/2}} - k_{16} \\ C_{\text{BANK1/2}} = k_{16} - A_{\text{BANK1/2}} \end{array} \right. ;$

**Example:** `I0 = r2 + 5;`

**Type C1/2:**  $\left\{ \begin{array}{l} C_{\text{BANK1/2}} = C_{\text{BANK1/2}} + A_{\text{BANK1/2}} \\ C_{\text{BANK1/2}} = C_{\text{BANK1/2}} - A_{\text{BANK1/2}} \end{array} \right. \langle \text{MEM\_ACCESS\_1} \rangle \langle \text{MEM\_ACCESS\_2} \rangle;$

**Example:** `r2 = r2 + I2  
r0 = M[I0, M0]  
r1 = M[I4, M1];`

#### Description:

Test the optional condition and, if TRUE, perform the specified cross-bank addition or subtraction. If the condition is FALSE, perform a NOP, but *MEM\_ACCESS\_1* still carried out. Omitting the condition performs the addition or subtraction unconditionally. The operands may be either one of the 16 Bank1 or 16 Bank2 registers or a 16-bit sign extended constant (24-bit with prefix instruction).

#### Flags Generated:

|   |                                                                        |   |                                                                                                          |
|---|------------------------------------------------------------------------|---|----------------------------------------------------------------------------------------------------------|
| Z | Set if the result equals zero and cleared otherwise                    | N | Set if the result is negative and cleared otherwise                                                      |
| V | Set if an arithmetic overflow occurs and cleared otherwise (see Note). | C | For addition: set if a carry generated.<br>For subtraction: cleared if a borrow is generated (see Note). |

#### Note:

If one of the source operands is Null then a load/store assumed. The C and V flags are left unchanged.  
If the destination register is from bank2 (i.e. 16-bit) then the C and V flags are left unchanged.  
Additions and subtractions may be saturated on overflow by setting the ADDSUB\_SATURATE\_ON\_OVERFLOW memory mapped register. This causes saturation of signed numbers (positive saturation to 0x7FFFFFFF, negative saturation to 0x800000). This will only occur where the destination register is a Bank1 24-bit register.



## 6.4 Logical Operations: AND, OR and XOR

### Syntax:

**Type A:**       $\langle \text{if cond} \rangle$      $\left\{ \begin{array}{l} C = A \text{ AND } B \\ C = A \text{ OR } B \\ C = A \text{ XOR } B \end{array} \right. \quad \langle \text{MEM\_ACCESS\_1} \rangle;$

**Example:**     $\text{if } Z \text{ } r3 = r1 \text{ AND } r2$   
                    $r0 = M[I0, M0];$

**Type B:**                     $\left\{ \begin{array}{l} C = A \text{ AND } k_{16} \\ C = A \text{ OR } k_{16} \\ C = A \text{ XOR } k_{16} \end{array} \right. ;$

**Example:**     $r3 = r1 \text{ XOR } 10;$

**Type C1/2:**                 $\left\{ \begin{array}{l} C = C \text{ AND } A \\ C = C \text{ OR } A \\ C = C \text{ XOR } A \end{array} \right. \quad \langle \text{MEM\_ACCESS\_1} \rangle \quad \langle \text{MEM\_ACCESS\_2} \rangle;$

**Example:**     $r3 = r3 \text{ OR } r1$   
                    $r0 = M[I0, M0]$   
                    $r2 = M[I4, M1];$

### Description:

Test the optional condition and, if TRUE, perform the specified bit wise logical operation (logical AND, OR, or XOR). If the condition is FALSE, perform a NOP, but *MEM\_ACCESS\_1* still carried out. Omitting the condition performs the operation unconditionally. The operands may be either one of the 16 Bank1 registers or a 16-bit sign extended constant (24-bit with prefix instruction).

### Flags Generated:

|   |                                                     |   |                                                     |
|---|-----------------------------------------------------|---|-----------------------------------------------------|
| Z | Set if the result equals zero and cleared otherwise | N | Set if the result is negative and cleared otherwise |
| V | Left unchanged                                      | C | Left unchanged                                      |

## 6.5 Shifter: LSHIFT and ASHIFT

### Syntax:

**Type A:** `<if cond> C = A OP B <MEM_ACCESS_1>;`

**Example:** `if Z r3 = r2 LSHIFT r1  
r0 = M[I0,M0];`

**Type B:**

|                                             |
|---------------------------------------------|
| <code>C = A OP k<sub>7</sub></code>         |
| <code>rMAC0 = A OP k<sub>7</sub></code>     |
| <code>rMAC12 = A OP k<sub>7</sub></code>    |
| <code>rMAC2 = A OP k<sub>7</sub></code>     |
| <code>rMAC = A OP k<sub>7</sub> (LO)</code> |
| <code>rMAC = A OP k<sub>7</sub> (MI)</code> |
| <code>rMAC = A OP k<sub>7</sub> (HI)</code> |
| <code>C = k<sub>16</sub> OP A</code>        |

**Example:** `rMAC0 = r2 LSHIFT 4;`

**Type C1/2:** `C = C OP A <MEM_ACCESS_1> <MEM_ACCESS_2>;`

**Example:** `r2 = r2 ASHIFT r7  
r5 = M[I0,M2]  
r1 = M[I4,M1];`

### Description:

Test the optional condition and, if TRUE, perform the specified shift operation (arithmetic or logical). If the condition is FALSE, perform a NOP, but `MEM_ACCESS_1` still carried out. Omitting the condition performs the shift unconditionally. A positive number causes a shifting to the left and a negative number causes a shifting to the right. For an arithmetic shift to the right sign extension bits added as needed. If overflow occurs in an arithmetic shift, i.e. non-sign bits being shifted out, then the overflow flag is set and the result is saturated to  $2^{23}-1$  or  $-2^{23}$  depending on the sign of the input. No rounding occurs for ASHIFT or LSHIFT. The operands may be either one of the 16 Bank1 registers or a constant specified in the instruction.

### Flags Generated:

|                                                                                                                                                                        |                                                                                      |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|
| <p>Z Set if the result equals zero and cleared otherwise</p> <p>V ASHIFT: Set if an arithmetic overflow occurs and cleared otherwise</p> <p>LSHIFT: Left unchanged</p> | <p>N Set if the result is negative and cleared otherwise</p> <p>C Left unchanged</p> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|

### Note:

OP is either ASHIFT (arithmetic) or LSHIFT (logical).

If `rMAC` is the source operand, the full 56 bits are used as input to the shifter. (The output of the shifter is always 24-bit.)

For Type B instructions the destination operand can be:

- Either, `rMAC0`, `rMAC12`, or `rMAC2`, causing the other bits of `rMAC` to be unaffected (writing to `rMAC12` writes the data into `rMAC1` and causes sign extension (ASHIFT) or zero fill (LSHIFT) into `rMAC2`)
- Or, `rMAC`, with a data format tag (LO, MI, HI) to select which word of `rMAC` the 24-bit result from the shifter should be written to, the other bits of `rMAC` are sign-extended / zero-padded as appropriate

The source register may also be a 16-bit sign-extended constant (or 24-bit constant with prefix).

## 6.6 rMAC Move Operations

Syntax:

**Type B:**  $\begin{bmatrix} \text{rMAC0} \\ \text{rMAC2} \end{bmatrix} = \begin{bmatrix} \text{rMAC0} \\ \text{rMAC1} \\ \text{rMAC2} \\ \text{A} \end{bmatrix};$

**Example:**  $\text{rMAC0} = \text{rMAC1};$

$\text{rMAC12} = \begin{bmatrix} \text{rMAC0} \\ \text{rMAC1} \\ \text{rMAC2} \\ \text{A} \end{bmatrix} \begin{matrix} (\text{SE}) \\ (\text{ZP}) \end{matrix};$

**Example:**  $\text{rMAC12} = \text{rMAC0} (\text{SE});$

$\text{C} = \begin{bmatrix} \text{rMAC0} \\ \text{rMAC1} \end{bmatrix};$

**Example:**  $\text{r3} = \text{rMAC0};$

$\text{C} = \begin{bmatrix} \text{rMAC2} \end{bmatrix} \begin{matrix} (\text{SE}) \\ (\text{ZP}) \end{matrix};$

**Example:**  $\text{r3} = \text{rMAC2} (\text{ZP});$

**Description:**

These are move instructions, implemented as a special case of LSHIFT and ASHIFT, to support loading and reading of the individual sections of the rMAC register (rMAC2, rMAC1, and rMAC0) shown in Figure 4.2. When writing to rMAC1, the data is either sign extended or zero padded into rMAC2. The format specifiers, SE and ZP, are required to specify how rMAC2 filled.

**Flags Generated:**

|   |                                                     |   |                                                     |
|---|-----------------------------------------------------|---|-----------------------------------------------------|
| Z | Set if the result equals zero and cleared otherwise | N | Set if the result is negative and cleared otherwise |
| V | Left unchanged                                      | C | Left unchanged                                      |

**Note:**

These operations are always implemented as a Type B instruction, i.e. no parallel memory reads can be performed.

## 6.7 Multiply: Signed 24-Bit Fractional and Integer

### Syntax:

**Type A:** `<if cond> | C = A * B | (frac) | <(sat)> <MEM_ACCESS_1>`

**Example:** `if Z r3 = r2 * r1 (int) (sat)  
r5 = M[I0, M2];`

**Type B:** `| C = A * k16 | (frac) | <(sat)>`

**Example:** `r6 = r2 * 0.34375 (frac);`

**Type C1/2:** `| C = C * B | (frac) | <(sat)> <MEM_ACCESS_1> <MEM_ACCESS_2>`

**Example:** `r2 = r2 * r7 (int) (sat)  
r0 = M[I0, 1]  
r1 = M[I4, -1];`

### Description:

Test the optional condition and, if TRUE, perform the specified multiply operation (fractional or integer). If the condition is FALSE, perform a NOP, but `MEM_ACCESS_1` still carried out. Omitting the condition performs the operation unconditionally. A fractional multiply, (`frac`), treats the source and destination operands as fractional numbers with  $2^{23}$  representing +1 and  $-2^{23}$  representing -1. An integer multiply, (`int`), treats the source and destination operands as integer numbers. Optionally, if designated by the (`sat`) notation, the result of an integer multiply is saturated if overflow occurs. Unbiased rounding is always done for a fractional multiply operation. The operands may be either one of the 16 Bank1 registers or a 16-bit left justified constant (24-bit with prefix instruction). See Appendix A on number representation for information on multiply operations.

### Flags Generated:

|   |                                                                        |   |                                                     |
|---|------------------------------------------------------------------------|---|-----------------------------------------------------|
| Z | Set if the result equals zero and cleared otherwise                    | N | Set if the result is negative and cleared otherwise |
| V | frac: Left unchanged                                                   | C | Left unchanged                                      |
|   | int: Set if an arithmetic signed overflow occurs and cleared otherwise |   |                                                     |

### Note:

A saturated fractional multiplication has no significance therefore (`sat`) is not an option with (`frac`).

The 16-bit constant is left justified to 24-bits by adding 8 zeros as the LSBs. This allows the 16-bit constant to represent fixed point fractional numbers between +1 and -1.

Unbiased rounding is as follows:

```
rMACrounded = rMAC[47:24] + rMAC[23];
if (rMAC[23:0] == 0x800000) then rMACrounded[0] = 0;
```

This has the effect of rounding odd `rMAC[47:24]` values away from zero and even `rMAC[47:24]` values towards zero, yielding a zero large sample bias assuming uniformly distributed values.

## 6.8 MULTIPLY and ACCUMULATE (56-bit)

### Syntax:

|                   |                                                                                 |                                                                                                      |                                                                   |
|-------------------|---------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------|
| <b>Type A:</b>    | $rMAC = A * B$<br>$rMAC = rMAC + A * B$<br>$rMAC = rMAC - A * B$                | $\langle (SS) \rangle$<br>$\langle (SU) \rangle$<br>$\langle (US) \rangle$<br>$\langle (UU) \rangle$ | $\langle MEM\_ACCESS\_1 \rangle;$                                 |
| <b>Example:</b>   | $if\ Z\ rMAC = rMAC + r1 * r2\ (SS)$<br>$r5 = M[I0, M0];$                       |                                                                                                      |                                                                   |
| <b>Type B:</b>    | $rMAC = A * k_{16}$<br>$rMAC = rMAC + A * k_{16}$<br>$rMAC = rMAC - A * k_{16}$ | $\langle (SS) \rangle$<br>$\langle (SU) \rangle$<br>$\langle (US) \rangle$<br>$\langle (UU) \rangle$ | ;                                                                 |
| <b>Example:</b>   | $rMAC = rMAC + r1 * 0.24254\ (SS);$                                             |                                                                                                      |                                                                   |
| <b>Type C1/2:</b> | $rMAC = C * A$<br>$rMAC = rMAC + C * A$<br>$rMAC = rMAC - C * A$                | $\langle (SS) \rangle$<br>$\langle (SU) \rangle$<br>$\langle (US) \rangle$<br>$\langle (UU) \rangle$ | $\langle MEM\_ACCESS\_1 \rangle\ \langle MEM\_ACCESS\_2 \rangle;$ |
| <b>Example:</b>   | $rMAC = rMAC - r3 * r1\ (SS)$<br>$r2 = M[I0, 1]$<br>$r1 = M[I4, -1];$           |                                                                                                      |                                                                   |

### Description:

Test the optional condition and, if `TRUE`, perform the specified multiply/accumulate. If the condition is `FALSE`, perform a NOP, but `MEM_ACCESS_1` still carried out. Omitting the condition performs the operation unconditionally. The data format field to the right of the operands specifies whether each respective operand is in signed (S) or unsigned (U) format. The effective binary point is between bits 47 and 46. The operands may be either one of the 16 Bank1 registers or a 16-bit left -justified constant (24-bit with prefix instruction). See Appendix A on number representation for information on multiply operations.

### Flags Generated:

|   |                                                                            |   |                                                     |
|---|----------------------------------------------------------------------------|---|-----------------------------------------------------|
| Z | Set if the result equals zero and cleared otherwise                        | N | Set if the result is negative and cleared otherwise |
| V | Set if overflow occurs past the 56 <sup>th</sup> bit and cleared otherwise | C | Left unchanged                                      |

### Note:

Where  $(SS)$  is the default if no data format is specified.

The 16-bit constant is left justified to 24-bits by adding 8 zeros as the LSBs. This allows the 16-bit constant to represent fixed point fractional numbers between +1 and -1.

To get the result of the equivalent integer multiplication, the result should be shifted to the right by 1-bit.

## 6.9 LOAD / STORE with Memory Offset

### Syntax:

**Type A:**  $\langle \text{if cond} \rangle \left\{ \begin{array}{l} C = M[A + B] \\ M[C + A] = B \end{array} \right. \langle \text{MEM\_ACCESS\_1} \rangle;$

**Example:**  $\text{if } Z \text{ } r3 = M[r1 + r2]$   
 $r4 = M[I0, M0];$

**Type B:**  $\left\{ \begin{array}{l} C = M[A + k_{16}] \\ M[C + k_{16}] = A \end{array} \right. ;$

**Example:**  $M[r3 + 6] = r1;$

**Type C:**  $\left\{ \begin{array}{l} C = M[C + A] \end{array} \right. \langle \text{MEM\_ACCESS\_1} \rangle \langle \text{MEM\_ACCESS\_2} \rangle;$

**Example:**  $r3 = M[r3 + r2]$   
 $r4 = M[I0, 1]$   
 $r5 = M[I4, -1];$

### Description:

Test the optional condition and, if TRUE, perform the specified load/store, including memory offset. If the condition is FALSE, perform a NOP, but *MEM\_ACCESS\_1* still carried out. Omitting the condition performs the load/store unconditionally. The operands may be either one of the 16 Bank1 registers or a 16-bit constant specified in the instruction.

### Flags Generated:

|   |                                                             |   |                                                     |
|---|-------------------------------------------------------------|---|-----------------------------------------------------|
| Z | Set if the result operand equals zero and cleared otherwise | N | Set if the result is negative and cleared otherwise |
| V | Left unchanged                                              | C | Left unchanged                                      |



## 6.10 Sign Bits Detect and Block Sign Bits Detect

### Syntax:

Type A: `<if cond> | C = SIGNDET A | <MEM_ACCESS_1>;`

**Example:** `if Z r3 = SIGNDET rMAC  
r4 = M[I0, M0];`

Type C: `| C = BLKSIGNDET A | <MEM_ACCESS_1> <MEM_ACCESS_2>;`

**Example:** `r3 = BLKSIGNDET r1  
r4 = M[I0, 1]  
r5 = M[I4, -1];`

### Description:

`SIGNDET` returns the number of redundant sign bits of the source operand. For example:

|                               |   |                                           |
|-------------------------------|---|-------------------------------------------|
| 0000 1101 0101 0101 1100 1111 | - | has 3 redundant sign bits                 |
| 1001 0101 0101 0100 0111 1111 | - | has 0 redundant sign bits                 |
| 0000 0000 0000 0000 0000 0001 | - | has 22 redundant sign bits                |
| 1111 1111 1111 1111 1111 1111 | - | has 23 redundant sign bits                |
| 0000 0000 0000 0000 0000 0000 | - | has 23 redundant sign bits (special case) |

Valid results are 0 to 23 for the 24-bit registers, and -8 to 47 for `rMAC`.

`BLKSIGNDET` returns the smaller of the result of `SIGNDET` and the present value of the destination operand. When performed on a series of numbers, it can derive the effective exponent of the number largest in magnitude.

### Flags Generated:

|   |                                                     |   |                                                     |
|---|-----------------------------------------------------|---|-----------------------------------------------------|
| Z | Set if the result equals zero and cleared otherwise | N | Set if the result is negative and cleared otherwise |
| V | Left unchanged                                      | C | Left unchanged                                      |

## 6.11 Divide Instruction

### Syntax:

**Type B:** `Div = rMAC / A ;`  
`C = DivResult`  
`C = DivRemainder`

**Example:** `Div = rMAC / r1;`  
`r2 = DivResult;`  
`r3 = DivRemainder;`

### Description:

The `Div = rMAC/A` instruction initiates the divide block to start its multi-cycle 48-bit / 24-bit integer divide. The overflow flag is set if the `DivResult` is wider than 24-bits. In this case, the result is saturated to  $-2^{23}$  or  $2^{23} - 1$  and the remainder is invalid. The result and/or remainder of the divide is available after 12 cycles. If the result or remainder is requested before the 12 cycles has elapsed then program flow is suspended until the result is ready. To carry out a fractional divide the value in `rMAC` needs to be first right shifted by 1 bit before carrying out the divide operation. The result of the divide may also be PUSHed directly onto the stack, which will stall the DSP if the divide is not yet completed.

### Integer Divide example (86420 / 7 = 12345 remainder 5):

```
rMAC = 0; // clear rMAC
r0 = 86420;
rMAC0 = r0; // LS word of rMAC now equals 86420;
r0 = 7;
Div = rMAC / r0;
r1 = DivResult; // r1 = 12345
r2 = DivRemainder; // r2 = 5
```

### Fractional Divide example (0.25/0.75 = 0.3333):

```
rMAC = 0.25;
r0 = 0.75;
rMAC = rMAC ASHIFT -1;
Div = rMAC / r0;
r1 = DivResult; // r1 = 0.33333;
```

### Flags Generated:

After: `Div = rMAC/A;`

|   |                                                                                                                |   |                |
|---|----------------------------------------------------------------------------------------------------------------|---|----------------|
| Z | Left unchanged                                                                                                 | N | Left unchanged |
| V | Set if a divide exception occurs (divide by zero or overflow in <code>DivResult</code> ) and cleared otherwise | C | Left unchanged |

After: `C = DivResult;` or `C = DivRemainder;`

|   |                                                     |   |                                                     |
|---|-----------------------------------------------------|---|-----------------------------------------------------|
| Z | Set if the result equals zero and cleared otherwise | N | Set if the result is negative and cleared otherwise |
| V | Left unchanged                                      | C | Left unchanged                                      |



## 6.12 PUSH, POP and PLOOK Stack Instructions

### Syntax:

|                   |                                                                                     |                                                                       |
|-------------------|-------------------------------------------------------------------------------------|-----------------------------------------------------------------------|
| <b>Type A:</b>    | $\langle if\ cond \rangle$<br>$push\ C_{BANK1/2/3}$<br>$pop\ C_{BANK1/2/3}$         | $\langle MEM\_ACCESS\_1 \rangle;$                                     |
| <b>Examples:</b>  | $if\ Z\ push\ r0$<br>$r5 = M[I0, M2];$<br>$If\ NZ\ pop\ DivResult;$                 |                                                                       |
| <b>Type B:</b>    | $push\ k_{16}$<br>$push\ C_{BANK1/2/3} + k_{16}$<br>$C_{BANK1/2/3} = plook\ k_{16}$ | $;$                                                                   |
| <b>Examples:</b>  | $push\ I0 + 4;$<br>$M3 = plook\ 2;$                                                 |                                                                       |
| <b>Type C1/2:</b> | $push\ C_{BANK1/2/3}$<br>$pop\ C_{BANK1/2/3}$                                       | $\langle MEM\_ACCESS\_1 \rangle\ \ \ \langle MEM\_ACCESS\_2 \rangle;$ |
| <b>Examples:</b>  | $push\ r0$<br>$r5 = M[I0, 1]$<br>$r1 = M[I4, -1];$                                  |                                                                       |

### Description:

Test the optional condition and, if TRUE, perform the specified stack operation. If the condition is FALSE, perform a NOP, but *MEM\_ACCESS\_1* still carried out. Omitting the condition performs the operation unconditionally. The operands may be any one of the 16 Bank1, any of the 16 Bank2 and any of the special Bank3 registers. Memory-mapped registers set the stack start address and end address. Type B *push* pushes the result of a 16 bit constant (or 24-bit with prefix) and the selected register. A *plook* returns the contents of  $M[STACK\_POINTER - k_{16}]$  without modifying the stack pointer. The memory-mapped stack pointer is incremented with a *push* and decremented with a *pop*, and holds the current stack write address.

### Flags Generated:

|   |                                                                                                                                                          |   |                                                                                                                                                              |
|---|----------------------------------------------------------------------------------------------------------------------------------------------------------|---|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Z | Set if the operand ( <i>push</i> ) or result ( <i>pop</i> / <i>plook</i> ) is Zero, cleared otherwise, left unchanged if operand/result is <i>rFlags</i> | N | Set if the operand ( <i>push</i> ) or result ( <i>pop</i> / <i>plook</i> ) is negative, cleared otherwise, left unchanged if operand/result is <i>rFlags</i> |
| V | <i>plook</i> : Left unchanged<br><br><i>push</i> : Set if a stack overflow occurs<br><br><i>pop</i> : Set if a stack underflow occurs                    | C | Left unchanged                                                                                                                                               |

### Notes:

A Conditional *pop* with condition FALSE carries out the read regardless, but destination register, flags and *STACK\_POINTER* remain unchanged

There is no stack underflow flag when destination is *rFlags*

*push* of *DivResult* or *DivRemainder* whilst a Divide is busy will stall program flow until divide completes and then push the divide result. A conditional *push* of *DivResult* or *DivRemainder* with condition FALSE will cause the program flow to stall anyway until divide completes, but not push the result, set flags or increment stack pointer

*pop* of *DivResult* or *DivRemainder* whilst the divider is busy will fail to alter the value of *DivResult* or *DivRemainder*, but the stack pointer will be decremented and flags will be set accordingly.

When Popping *DoLoopStart*, *DoLoopEnd* or *r10*, all values must be popped at least 2 instructions before the end of a DO...LOOP.

Writing to memory-mapped  $M[ \$STACK\_START\_ADDR ]$  sets  $M[ \$STACK\_POINTER ]$  equal to  $M[ \$STACK\_START\_ADDR ]$ .

Popping to `rMAC12` sign-extends into `rMAC2`.

### 6.13 Program Flow: CALL, JUMP, RTS, RTI, SLEEP, DO...LOOP and BREAK

Syntax:

**Type A:**

|                              |                                         |  |   |
|------------------------------|-----------------------------------------|--|---|
| <code>&lt;if cond&gt;</code> | jump A<br>call A<br>rts<br>rti<br>sleep |  | ; |
|------------------------------|-----------------------------------------|--|---|

**Example:** `if Z jump r1;`

**Type B:**

|                              |                                              |  |   |
|------------------------------|----------------------------------------------|--|---|
| <code>&lt;if cond&gt;</code> | jump k <sub>16</sub><br>call k <sub>16</sub> |  | ; |
| do k <sub>16</sub>           |                                              |  |   |

**Example:**

```

r10 = 100;
do loop;
 rMAC = rMAC + r0 * r1
 r3 = M[I4,1];
loop:

```

**Type C:**

|                              |            |  |   |
|------------------------------|------------|--|---|
| <code>&lt;if cond&gt;</code> | rts<br>rti |  | ; |
|------------------------------|------------|--|---|

**Example:** `if NZ rts;`

|  |       |  |   |
|--|-------|--|---|
|  | break |  | ; |
|--|-------|--|---|

**Example:** `break;`

#### Description:

Omitting the condition performs the program flow. If the condition is `TRUE` perform program flow below, else execute a `NOP`:

- `jump` Program execution jumps to the address in operand (either a register or a constant, e.g. an address label).
- `do` For zero overhead looping, instructions between `DO` and `loop` are executed until register `r10` is zero, `r10` is decremented by one each loop. If `r10` is zero at start then no loop instructions executed and a jump to `loop` occurs. If a `do...loop` is executed in an interrupt service routine (ISR), `r10` and the memory mapped registers `MM_DOLOOP_START` and `MM_DOLOOP_END` should be saved.
- `sleep` Program execution paused and the DSP put in lower power mode. Interrupts still handled in sleep. The ISR is responsible for issuing a software event that causes wake up from a sleep instruction.
- `call` Loads `rLink` register with return address (`PC+1`) and jumps to address in operand (either a register or a constant e.g. an address label).
- `rti` Sets `PC` equal to value of `rIntLink` register, and restores flags to their pre-interrupt status, i.e. the MS byte of `rFlags` register is copied to the LS byte.
- `rts` Sets `PC` equal to value of `rLink` register. Stack depths greater than one must be implemented in software.
- `break` For program debug, used by `xIDE`, a break instruction either acts as a `nop`, or jump to self.

#### Flags Generated:

Flags Z, N, V and C left unchanged

**Note:** See Section 10.4 for information about stalls.

## 6.14 Indexed MEM\_ACCESS\_1 and MEM\_ACCESS\_2

**Syntax:**

|                 | MEM_ACCESS_1                                                                                                    |   | MEM_ACCESS_2                                                                                                    |
|-----------------|-----------------------------------------------------------------------------------------------------------------|---|-----------------------------------------------------------------------------------------------------------------|
| <b>Type A:</b>  | $\text{Reg}_{AG1} = M[\text{I}_{AG1}, \text{M}_{AG1}]$ $M[\text{I}_{AG1}, \text{M}_{AG1}] = \text{Reg}_{AG1}$   | ; |                                                                                                                 |
| <b>Example:</b> | $M[\text{I}0, \text{M}0] = r1;$                                                                                 |   |                                                                                                                 |
| <b>Type C:</b>  | $\text{Reg}_{AG1} = M[\text{I}_{AG1}, \text{M}_{AG1}]$ $M[\text{I}_{AG1}, \text{M}_{AG1}] = \text{Reg}_{AG1}$   | ; | $\text{Reg}_{AG2} = M[\text{I}_{AG2}, \text{M}_{AG2}]$ $M[\text{I}_{AG2}, \text{M}_{AG2}] = \text{Reg}_{AG2}$   |
|                 | $\text{Reg}_{AG1} = M[\text{I}_{AG1}, \text{MK}_{AG1}]$ $M[\text{I}_{AG1}, \text{MK}_{AG1}] = \text{Reg}_{AG1}$ | ; | $\text{Reg}_{AG2} = M[\text{I}_{AG2}, \text{MK}_{AG2}]$ $M[\text{I}_{AG2}, \text{MK}_{AG2}] = \text{Reg}_{AG2}$ |
| <b>Example:</b> | $r0 = M[\text{I}0, \text{M}0]$<br>$M[\text{I}4, \text{M}1] = r1;$                                               |   |                                                                                                                 |

### Permitted Registers

|                                       |                                 |                  |                   |
|---------------------------------------|---------------------------------|------------------|-------------------|
| $\text{Reg}_{AG1} / \text{Reg}_{AG2}$ | r0, r1, r2, r3, r4, r5 and rMAC | $\text{I}_{AG1}$ | I0, I1, I2 and I3 |
| $\text{M}_{AG1} / \text{M}_{AG2}$     | M0, M1, M2 and M3               | $\text{I}_{AG2}$ | I4, I5, I6 and I7 |
| $\text{MK}_{AG1} / \text{MK}_{AG2}$   | -1, 0, 1 and 2                  |                  |                   |

### Description:

Any Type C instruction can also perform up to two memory reads/writes in the same instruction cycle as the main ALU part of the instruction.  $\text{Reg}_{AG1/AG2}$  selects the source or destination register for the memory read or write.  $\text{I}_{AG1/AG2}$  selects the index register to use for the memory read, and either  $\text{M}_{AG1/AG2}$  selects the modify register or  $\text{MK}_{AG1/AG2}$  selects the modify constant (-1, 0, +1, or +2) to use for the post modify of the index register. Type A instructions can perform a single memory read/write with the limitation that the modify operand is not a constant, i.e. must be  $\text{M}_{AG1/AG2}$ .

### Memory Access Timing:

Only one access (read or write) permitted per memory bank per clock cycle. If AG1 and AG2 both access DM1 in the same instruction then the two memory accesses are queued with the AG1 access occurring first. Memory reads set up the memory bus the instruction before, whereas memory writes take place at the end of the current instruction. This means that if the previous instruction does a memory write, then the current instruction delays by one clock cycle if it tries to read the same memory bank as the previous instruction wrote to. External wait signals from peripherals may slow down the instruction cycle.

### Flags Generated:

No flags affected by the memory access part of instructions.

### Note:

$\text{Reg}_{AG1/AG2}$  selects one of the first eight Bank1 register, i.e. Null, rMAC, r0–r5. If the Null register is selected then no memory read/write is performed.

Type A instructions use Address Generator 1 (AG1) so can only use index registers I0-I3. They must use a modify register rather than a modify constant.



Type C instructions use AG1 and AG2 so one memory access must use I0-I3 and the other must use I4-I7. They must either both use a modify register or both use a modify constant.



## 7 Instruction Coding

Instruction coding is relatively simple and orthogonal so that the instruction decode is efficient. There are three basic coding formats: Type A, B, and C. Table 7.1 outlines the format of the instruction with corresponding definitions described in Section 7.5 to Section 7.15.

|         |    |    |    |      |    |    |    |                         |    |    |    |                 |        |      |       |           |        |      |       |                    |    |   |   |                           |   |   |   |      |   |   |   |      |
|---------|----|----|----|------|----|----|----|-------------------------|----|----|----|-----------------|--------|------|-------|-----------|--------|------|-------|--------------------|----|---|---|---------------------------|---|---|---|------|---|---|---|------|
| 31      | 30 | 29 | 28 | 27   | 26 | 25 | 24 | 23                      | 22 | 21 | 20 | 19              | 18     | 17   | 16    | 15        | 14     | 13   | 12    | 11                 | 10 | 9 | 8 | 7                         | 6 | 5 | 4 | 3    | 2 | 1 | 0 | Type |
| OP_CODE |    |    |    | RegC |    |    |    | RegA<br>StackBankSelect |    |    |    | AG1 Write       | RegAG1 | IAG1 | MAG1  | RegB      |        |      |       | cond               |    |   |   | A                         |   |   |   |      |   |   |   |      |
|         |    |    |    |      |    |    |    |                         |    |    |    | K <sub>16</sub> |        |      |       |           |        |      |       |                    |    |   |   |                           |   |   |   | B    |   |   |   |      |
|         |    |    |    |      |    |    |    |                         |    |    |    | AG1 Write       | RegAG1 | IAG1 | MAG1  | AG2 Write | REGAG2 | IAG2 | MAG2  | C <sub>REG</sub>   |    |   |   |                           |   |   |   |      |   |   |   |      |
|         |    |    |    |      |    |    |    |                         |    |    |    | AG1 Write       | RegAG1 | IAG1 | MKAG1 | AG2 Write | REGAG2 | IAG2 | MKAG2 | C <sub>CONST</sub> |    |   |   |                           |   |   |   |      |   |   |   |      |
| 1       | 1  | 1  | 1  | 1    | 1  | 0  | 1  | 0                       | 0  | 0  | 0  | 0               | 0      | 0    | 0     | 0         | 0      | 0    | 0     | 0                  | 0  | 0 | 0 | K <sub>PREFIX</sub> [7:0] |   |   |   | PFIX |   |   |   |      |

Table 7.1: Instruction Coding Format

**Notes:**

- OP\_CODE           Selects the instruction operation, see Section 7.5
- RegA               Selects a register to use as the first source operand for instructions. Bank1 registers are used by default, Bank 2 is selected within OP\_CODE for certain instructions
- RegB               Selects a register to use as the first source operand for instructions. Bank1 registers are used by default, Bank 2 is selected within OP\_CODE for certain instructions
- RegC               Selects 1 of 16 registers to use as the destination register for instructions. For Type C instructions RegC also defines the first source operand, see Table 7.2. Bank1 registers are used by default, Bank2 is selected within OP\_CODE for certain instructions
- cond               Selects an optional condition to be met for the instruction to be executed, otherwise a no-operation will be executed
- K<sub>16</sub> / K<sub>PREFIX</sub>       A 16-bit / 8-bit constant used by Type B instructions
- AG1 / AG2         Selects whether the indexed memory access is a read (0) or a write (1)
- Reg<sub>AG1</sub> / Reg<sub>AG2</sub>   Selects one of the first eight Bank1 registers: rMAC, r0-r5; for the source/destination register of multifunction memory reads and writes. If Null is selected then no read or write is performed
- I<sub>AG1</sub> / I<sub>AG2</sub>       Selects one of the index registers, I0-I3 for AG1, and I4-I7 for AG2, for multifunction memory reads and writes
- M<sub>AG1</sub> / M<sub>AG2</sub>       Selects one of the modify registers, M0-M3, for multifunction memory reads and writes
- Mk<sub>AG1</sub> / Mk<sub>AG2</sub>     Selects a fixed constant to use for the modify: -1, 0, +1, or +2
- StackBankSelect   Selects the bank of registers to use for stack operations



## 7.1 Type A Instruction

Type A is a conditional instruction, with an additional single memory read or write operation. The instruction can accept two input operands and one output operand (which may be different). Operands can be any of the 16 Bank1 registers (Load/Store instructions permit the usage of Bank2 registers and PUSH/POP Instructions permit the use of Bank1, 2 and special bank3 registers). For the memory access, index registers I0-I3 select the address, the destination or source register can be any of the first eight Bank1 registers: rMAC, r0 - r5. At the end of the instruction the index register is post modified by one of the modify registers M0-M3.

A Type A instruction has the following syntax:

```
<if cond> RegC = RegA OP(1) RegB <MEM_ACCESS_1>;
```

## 7.2 Type B Instruction

Type B is a non-conditional instruction similar to Type A, but with one of the operands being a 16-bit constant stored in the instruction word. To use a 24-bit constant prefix the Type B instruction with the prefix (PFI) instruction. No additional memory access operation permitted. The PFI instruction is automatic if needed by the assembler *kalasm2*.

A Type B instruction has the following syntax:

```
RegC = RegA OP(1) constant;
```

## 7.3 Type C Instruction

Type C is a non-conditional instruction similar to a Type A, except that one of the input operands is also the output operand. In addition, two memory reads or writes may occur in the same clock cycle; one memory access uses Address Generator 1 (index registers I0-I3) and the other uses Address Generator 2 (index registers I4-I7). The index registers can be either post modified by the M0-M3 registers or by a 2-bit signed modify constant (valid values: -1, 0, 1, or 2).

A Type C instruction has the following syntax:

```
RegC = RegC OP(1) RegA <MEM_ACCESS_1> <MEM_ACCESS_2>;
```

## 7.4 Special Cases

Program flow instructions such as *jump*, *call*, *rts*, etc; use a slight variation on the above types, as they can always be conditional. Also the multiply accumulate instructions all write their result to the rMAC register.

**Note:**

<sup>(1)</sup> OP refers to operation which can include Addition, Subtraction, Multiplication, OR, AND, Exclusive OR.



## 7.5 OP\_CODE Coding

Table 7.2 shows the instruction decoding of the OP\_CODE field shown in Table 7.1

| OP_CODE |      |   | Action (Type A) <sup>(1)</sup>                                     | Action (Type B)                                                                                                                | Action (Type C <sub>REGICONST</sub> ) <sup>(2)</sup>                                                       | Description                                                          |                                                                                                                                                  |
|---------|------|---|--------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| 000     | AM   | C | RegC = RegA + RegB                                                 | RegC = RegA + k <sub>16</sub>                                                                                                  | RegC = RegC + RegA                                                                                         | Add <sup>(3)</sup>                                                   |                                                                                                                                                  |
| 001     | AM   | C | RegC = RegA – RegB                                                 | RegC = RegA - k <sub>16</sub>                                                                                                  | RegC = RegC – RegA                                                                                         | Subtract                                                             |                                                                                                                                                  |
| 010     | B2RS |   | RegC <sub>B1/2</sub> = RegA <sub>B1/2</sub> + RegB <sub>B1/2</sub> | RegC <sub>B1/2</sub> = RegA <sub>B1/2</sub> + k <sub>16</sub>                                                                  | RegC <sub>B1/2</sub> = RegC <sub>B1/2</sub> + RegA <sub>B1/2</sub>                                         | Bank1/2 Add                                                          |                                                                                                                                                  |
| 011     | B2RS |   | RegC <sub>B1/2</sub> = RegA <sub>B1/2</sub> - RegB <sub>B1/2</sub> | RegC <sub>B1/2</sub> = RegA <sub>B1/2</sub> - k <sub>16</sub><br>RegC <sub>B1/2</sub> = k <sub>16</sub> - RegA <sub>B1/2</sub> | RegC <sub>B1/2</sub> = RegC <sub>B1/2</sub> - RegA <sub>B1/2</sub>                                         | Bank1/2 Subtract                                                     |                                                                                                                                                  |
| 100     | 0    | 0 | 0                                                                  | RegC = RegA AND RegB                                                                                                           | RegC = RegA AND k <sub>16</sub>                                                                            | RegC = RegC AND RegA                                                 | Logical AND                                                                                                                                      |
| 100     | 0    | 0 | 1                                                                  | RegC = RegA OR RegB                                                                                                            | RegC = RegA OR k <sub>16</sub>                                                                             | RegC = RegC OR RegA                                                  | Logical OR                                                                                                                                       |
| 100     | 0    | 1 | 0                                                                  | RegC = RegA XOR RegB                                                                                                           | RegC = RegA XOR k <sub>16</sub>                                                                            | RegC = RegC XOR RegA                                                 | Logical XOR                                                                                                                                      |
| 100     | 0    | 1 | 1                                                                  | RegC = RegA LSHIFT RegB                                                                                                        | RegC = RegA LSHIFT k <sub>16</sub> <sup>(4)</sup>                                                          | RegC = RegC LSHIFT RegA                                              | Logical Shift                                                                                                                                    |
| 100     | 1    | 0 | 0                                                                  | RegC = RegA ASHIFT RegB                                                                                                        | RegC = RegA ASHIFT k <sub>16</sub> <sup>(4)</sup>                                                          | RegC = RegC ASHIFT RegA                                              | Arithmetic Shift                                                                                                                                 |
| 100     | 1    | 1 | V                                                                  | RegC = RegA * RegB (int)                                                                                                       | RegC = RegA * k <sub>16</sub> (int)                                                                        | RegC = RegC * RegA (int)                                             | Integer signed multiply                                                                                                                          |
| 100     | 1    | 0 | 1                                                                  | RegC = RegA * RegB (frac)                                                                                                      | RegC = RegA * k <sub>16</sub> (frac)                                                                       | RegC = RegC * RegA (frac)                                            | Fractional signed multiply                                                                                                                       |
| 101     | 0    | S | S                                                                  | rMAC = rMAC + RegA * RegB                                                                                                      | rMAC = rMAC + RegA * k <sub>16</sub>                                                                       | rMAC = rMAC + RegC * RegA                                            | Multiply accumulate (56-bit)                                                                                                                     |
| 101     | 1    | S | S                                                                  | rMAC = rMAC – RegA * RegB                                                                                                      | rMAC = rMAC – RegA * k <sub>16</sub>                                                                       | rMAC = rMAC – RegC * RegA                                            | Multiply subtract (56-bit)                                                                                                                       |
| 110     | 0    | S | S                                                                  | rMAC = RegA * RegB                                                                                                             | rMAC = RegA * k <sub>16</sub>                                                                              | rMAC = RegC * RegA                                                   | Multiply (48-bit)                                                                                                                                |
| 110     | 1    | 0 | 0                                                                  | RegC = M[RegA + RegB]                                                                                                          | RegC = M[RegA + k <sub>16</sub> ]                                                                          | RegC = M[RegC + RegA]                                                | Load with offset                                                                                                                                 |
| 110     | 1    | 0 | 1                                                                  | M[RegA + RegB] = RegC                                                                                                          | M[RegA + k <sub>16</sub> ] = RegC                                                                          | -                                                                    | Store with offset                                                                                                                                |
| 110     | 1    | 1 | 0                                                                  | RegC = SignDet RegA                                                                                                            | Div = rMAC / RegA<br>RegC = DivResult <sup>(2)</sup><br>RegC = DivRemainder <sup>(2)</sup>                 | RegC = BlkSignDet RegA                                               | Sign detect / Divide / Block sign detect                                                                                                         |
| 110     | 1    | 1 | 1                                                                  | JUMP RegA                                                                                                                      | if [RegC=cond] JUMP k <sub>16</sub>                                                                        | if [RegC = cond] RTS                                                 | Jump to program address / return from subroutine                                                                                                 |
| 111     | 0    | 0 | 0                                                                  | CALL RegA                                                                                                                      | if [RegC=cond] CALL k <sub>16</sub>                                                                        | if [RegC = cond] RTI                                                 | Call subroutine / return from interrupt                                                                                                          |
| 111     | 0    | 0 | 1                                                                  | SLEEP                                                                                                                          | DO .... LOOP                                                                                               | BREAK                                                                | Go into sleep mode / Do Loop / Debug Break point                                                                                                 |
| 111     | 0    | 1 | 0                                                                  | FUTURE USE <sup>(5)</sup>                                                                                                      | RegC = k <sub>16</sub> LSHIFT RegA                                                                         | FUTURE USE <sup>(5)</sup>                                            | Logical Shift of a constant                                                                                                                      |
| 111     | 0    | 1 | 1                                                                  | FUTURE USE <sup>(5)</sup>                                                                                                      | RegC = k <sub>16</sub> ASHIFT RegA                                                                         | FUTURE USE <sup>(5)</sup>                                            | Arithmetic Shift of a constant                                                                                                                   |
| 111     | 1    | 0 | 0                                                                  | PUSH RegC <sub>B1/2</sub><br>POP RegC <sub>B1/2</sub> <sup>(6)</sup>                                                           | PUSH RegC <sub>B1/2</sub> + k <sub>16</sub><br>RegC <sub>B1/2</sub> = PLOOK k <sub>16</sub> <sup>(6)</sup> | PUSH RegC <sub>B1/2</sub><br>POP RegC <sub>B1/2</sub> <sup>(6)</sup> | Stack Operations                                                                                                                                 |
| 111     | 1    | 0 | 1                                                                  | FUTURE USE <sup>(5)</sup>                                                                                                      |                                                                                                            |                                                                      |                                                                                                                                                  |
| 111     | 1    | 1 | x                                                                  | FUTURE USE <sup>(5)</sup>                                                                                                      |                                                                                                            |                                                                      |                                                                                                                                                  |
| 111     | 1    | 1 | 1                                                                  |                                                                                                                                | PREFIX instruction                                                                                         |                                                                      | Allows 24-bit constants for Type B operations – by prefixing the following instruction's k <sub>16</sub> by the 8-bit k <sub>PREFIX</sub> value. |

Table 7.2: OPCODE Coding Format

**Note:**

- (1) Type A is conditional with if [ cond ] and can contain single memory access MEM\_ACCESS\_1
- (2) Type C permits two simultaneous memory access MEM\_ACCESS\_1 and MEM\_ACCESS\_2
- (3) The add instruction is also used to implement load/stores to registers/memory by setting one of the source registers as Null. Setting all 3 operands as Null implements a no-operation (NOP) instruction
- (4) See section 7.11 and section 7.15 for encoding of k<sub>16</sub> for rMAC shift instructions and Divide instructions
- (5) There are three spare OP\_CODES plus various other spare coding space for future instructions
- (6) PUSH/POP instruction is chosen in theStackBankSelect Field, see Section 7.16

## 7.6 AM Field

Selects different memory addressing modes

| AM | Type A                   | Type B                  | Type C <sub>REG/CONST</sub> |
|----|--------------------------|-------------------------|-----------------------------|
| 00 | RegC = RegA [OP] RegB    | RegC = RegA [OP] K16    | RegC = RegC [OP] RegA       |
| 01 | RegC = RegA [OP] M[RegB] | RegC = RegA [OP] M[K16] | RegC = RegC [OP] M[RegA]    |
| 10 | RegC = M[RegA] [OP] RegB | RegC = M[RegA] [OP] K16 | -                           |
| 11 | M[RegC] = RegA [OP] RegB | M[K16] = RegC [OP] RegA | -                           |

Table 7.3: AM Field

## 7.7 Carry Field (C Field)

Selects whether addition/subtraction performed with carry and the appropriate state shown in Table 7.4.

| C | Description               |
|---|---------------------------|
| 0 | Do not use carry / borrow |
| 1 | Use carry / borrow        |

Table 7.4: C Field Options

## 7.8 Bank 1/2 Register Select Field (B2RS Field)

Bank 1/2 register select for add/subtract instructions are shown in Table 7.5. Subtract from constant Type B instructions are only valid for SUBTRACT opcodes.

| B2RS | Type A                                                                                   | Type B                                                               | Type C <sub>REG/CONST</sub>                                                              |
|------|------------------------------------------------------------------------------------------|----------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| 000  | $\text{RegC}_{\text{Bank1}} = \text{RegA}_{\text{Bank1}} \pm \text{RegB}_{\text{Bank1}}$ | $\text{RegC}_{\text{Bank1}} = \text{RegA}_{\text{Bank1}} \pm K_{16}$ | $\text{RegC}_{\text{Bank1}} = \text{RegC}_{\text{Bank1}} \pm \text{RegA}_{\text{Bank1}}$ |
| 001  | $\text{RegC}_{\text{Bank1}} = \text{RegA}_{\text{Bank1}} \pm \text{RegB}_{\text{Bank2}}$ | $\text{RegC}_{\text{Bank1}} = K_{16} - \text{RegA}_{\text{Bank1}}$   | $\text{RegC}_{\text{Bank1}} = \text{RegC}_{\text{Bank1}} \pm \text{RegA}_{\text{Bank2}}$ |
| 010  | $\text{RegC}_{\text{Bank1}} = \text{RegA}_{\text{Bank2}} \pm \text{RegB}_{\text{Bank1}}$ | $\text{RegC}_{\text{Bank1}} = \text{RegA}_{\text{Bank2}} \pm K_{16}$ | -                                                                                        |
| 011  | $\text{RegC}_{\text{Bank1}} = \text{RegA}_{\text{Bank2}} \pm \text{RegB}_{\text{Bank2}}$ | $\text{RegC}_{\text{Bank1}} = K_{16} - \text{RegA}_{\text{Bank2}}$   | -                                                                                        |
| 100  | $\text{RegC}_{\text{Bank2}} = \text{RegA}_{\text{Bank1}} \pm \text{RegB}_{\text{Bank1}}$ | $\text{RegC}_{\text{Bank2}} = \text{RegA}_{\text{Bank1}} \pm K_{16}$ | -                                                                                        |
| 101  | $\text{RegC}_{\text{Bank2}} = \text{RegA}_{\text{Bank1}} \pm \text{RegB}_{\text{Bank2}}$ | $\text{RegC}_{\text{Bank2}} = K_{16} - \text{RegA}_{\text{Bank1}}$   | -                                                                                        |
| 110  | $\text{RegC}_{\text{Bank2}} = \text{RegA}_{\text{Bank2}} \pm \text{RegB}_{\text{Bank1}}$ | $\text{RegC}_{\text{Bank2}} = \text{RegA}_{\text{Bank2}} \pm K_{16}$ | $\text{RegC}_{\text{Bank2}} = \text{RegC}_{\text{Bank2}} \pm \text{RegA}_{\text{Bank1}}$ |
| 111  | $\text{RegC}_{\text{Bank2}} = \text{RegA}_{\text{Bank2}} \pm \text{RegB}_{\text{Bank2}}$ | $\text{RegC}_{\text{Bank2}} = K_{16} - \text{RegA}_{\text{Bank2}}$   | $\text{RegC}_{\text{Bank2}} = \text{RegC}_{\text{Bank2}} \pm \text{RegA}_{\text{Bank2}}$ |

Table 7.5: B2RS Field



## 7.9 Saturation Select Field (V Field)

Selects whether to enable saturation on the result and the options shown in Table 7.6. Note also that additions and subtractions may be saturated on overflow by setting the ADDSUB\_SATURATE\_ON\_OVERFLOW memory mapped register bit.

| V | Description   |
|---|---------------|
| 0 | No saturation |
| 1 | Saturation    |

Table 7.6: V Field

## 7.10 Sign Select Field (S Field)

Selects signed/unsigned multiplies and the various options shown in Table 7.7.

| S  | Description         |
|----|---------------------|
| 00 | unsigned x unsigned |
| 01 | unsigned x signed   |
| 10 | signed x unsigned   |
| 11 | signed x signed     |

Table 7.7: S Field

## 7.11 $k_{16}$ Coding for LSHIFT and ASHIFT

$k_{16}$  coding section from the instruction coding format shown in Table 7.1 splits into its individual bits and their functionality listed in Table 7.8

| 15 | 14 | 13 | 12 | 11 | 10 | 9        | 8 | 7 | 6           | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----------|---|---|-------------|---|---|---|---|---|---|
| 0  | 0  | 0  | 0  | 0  | 0  | Dest_Sel |   |   | ShiftAmount |   |   |   |   |   |   |

Table 7.8:  $k_{16}$  Coding Shift Format

Note:

- ShiftAmount: Is a signed 7-bit number that is the amount to shift the input.  
Positive is a left shift, negative is a right shift.
- Dest\_Sel: When the destination register is rMAC, selects how the 24-bit output from the shifter is used.

## 7.12 rMAC Sub Registers

The full 56-bits of the rMAC register are accessible as their individual sub-registers outlined in Table 7.9. See Section 7.13 and Section 7.14 for further details on how to load the individual sub registers.

| Bit           | 55 – 48 | 47 – 24 | 23 – 0 |
|---------------|---------|---------|--------|
| rMAC Register | rMAC2   | rMAC1   | rMAC0  |

Table 7.9: rMAC Sub-Registers



### 7.13 ASHIFT

Table 7.10 represents how the arithmetic shift instruction is encoded within the instruction coding format shown in Table 7.1 for further information on the LSHIFT instruction see Section 6.5.

| Dest_Sel | New rMAC2      | New rMAC1      | New rMAC0      | Example                       |
|----------|----------------|----------------|----------------|-------------------------------|
| 001      | Sign extend    | Sign extend    | SHIFTER_OUTPUT | $rMAC = r? ASHIFT k_{16} (0)$ |
| 000      | Sign extend    | SHIFTER_OUTPUT | Trailing zeros | $rMAC = r? ASHIFT k_{16} (1)$ |
| 010      | SHIFTER_OUTPUT | Trailing zeros | Trailing zeros | $rMAC = r? ASHIFT k_{16} (2)$ |
| 101      | Old rMAC2      | Old rMAC1      | SHIFTER_OUTPUT | $rMAC0 = r? ASHIFT k_{16}$    |
| 100      | Sign extend    | SHIFTER_OUTPUT | Old rMAC0      | $rMAC1 = r? ASHIFT k_{16}$    |
| 110      | SHIFTER_OUTPUT | Old rMAC1      | Old rMAC0      | $rMAC2 = r? ASHIFT k_{16}$    |

Table 7.10: ASHIFT

## 7.14 LSHIFT

Table 7.11 represents how the logical shift instruction is encoded within the instruction coding format shown in Table 7.1, for further information on the LSHIFT instruction see Section 6.5.

| Dest_Sel | New rMAC2      | New rMAC1      | New rMAC0      | Example                              |
|----------|----------------|----------------|----------------|--------------------------------------|
| 001      | Zero fill      | Zero fill      | SHIFTER_OUTPUT | rMAC = r? LSHIFT k <sub>16</sub> (0) |
| 000      | Zero fill      | SHIFTER_OUTPUT | Trailing zeros | rMAC = r? LSHIFT k <sub>16</sub> (1) |
| 010      | SHIFTER_OUTPUT | Trailing zeros | Trailing zeros | rMAC = r? LSHIFT k <sub>16</sub> (2) |
| 101      | Old rMAC2      | Old rMAC1      | SHIFTER_OUTPUT | rMAC0 = r? LSHIFT k <sub>16</sub>    |
| 100      | Zero fill      | SHIFTER_OUTPUT | Old rMAC0      | rMAC1 = r? LSHIFT k <sub>16</sub>    |
| 110      | SHIFTER_OUTPUT | Old rMAC1      | Old rMAC0      | rMAC2 = r? LSHIFT k <sub>16</sub>    |

Table 7.11: LSHIFT

Both the LSHIFT and ASHIFT instruction enable the user to write to the three individual sub-registers rMAC2/1/0, hence providing a way of loading rMAC with a double precision number. It also allows shift operations of rMAC with the destination being the rMAC register, which speeds up double precision calculations. See rMAC move operations in Section 0 for more details.

## 7.15 k<sub>16</sub> Coding Divide Instructions

Table 7.12 k<sub>16</sub> coding divide instruction represents how the background divide instruction is encoded within the instruction coding format shown in Table 7.1.

| 15       | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1   | 0 |
|----------|----|----|----|----|----|---|---|---|---|---|---|---|---|-----|---|
| Not used |    |    |    |    |    |   |   |   |   |   |   |   |   | Div |   |

Table 7.12: Divide Field

The functionality of the individual divide bits is explained more fully in Table 7.13.

| Div | Assembly Syntax     | Operation                                                                                                                                                                                                                                                          |
|-----|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 00  | Div = rMAC / RegA   | Initiates a 32-bit/16-bit divide. The overflow flag is set if a divide exception occurs. Calculation of the divide takes 12 clock cycles but program execution continues while the divide is calculated.                                                           |
| 01  | RegC = DivResult    | The result and/or remainder of a divide are available 12 cycles after the divide is initiated. In this period, normal program execution continues. If the result is requested early then program execution is automatically delayed until the result is available. |
| 10  | RegC = DivRemainder |                                                                                                                                                                                                                                                                    |
| 11  | NA                  | Not used                                                                                                                                                                                                                                                           |

Table 7.13: Divide Field States

## 7.16 StackBankSelect field encoding

Table 7.12 shows the instruction coding for the StackBankSelectField with stack opcodes.

| StackBankSelect Field |   |   |   | RegC bank               | Operation  |
|-----------------------|---|---|---|-------------------------|------------|
| 0                     | 0 | 0 | 0 | Bank1 registers         | POP/PLOOK  |
| 0                     | 0 | 0 | 1 | Bank1 registers         | PUSH       |
| 0                     | 0 | 1 | 0 | Bank2 registers         | POP/PLOOK  |
| 0                     | 0 | 1 | 1 | Bank2 registers         | PUSH       |
| 0                     | 1 | 0 | 0 | Bank3 special registers | POP/PLOOK  |
| 0                     | 1 | 0 | 1 | Bank3 special registers | PUSH       |
| 0                     | 1 | 1 | X | FUTURE USE              | FUTURE USE |
| 1                     | x | x | X | FUTURE USE              | FUTURE USE |

**Table 7.14: StackBankSelect Field Coding**

## 8 Kalimba DSP Peripherals

The Kalimba DSP for the BlueCore5-Multimedia consists of the DSP core, the DSP peripherals and their associated interfaces. This section covers the peripherals and interfaces.

See Appendix B for information relating to registers.

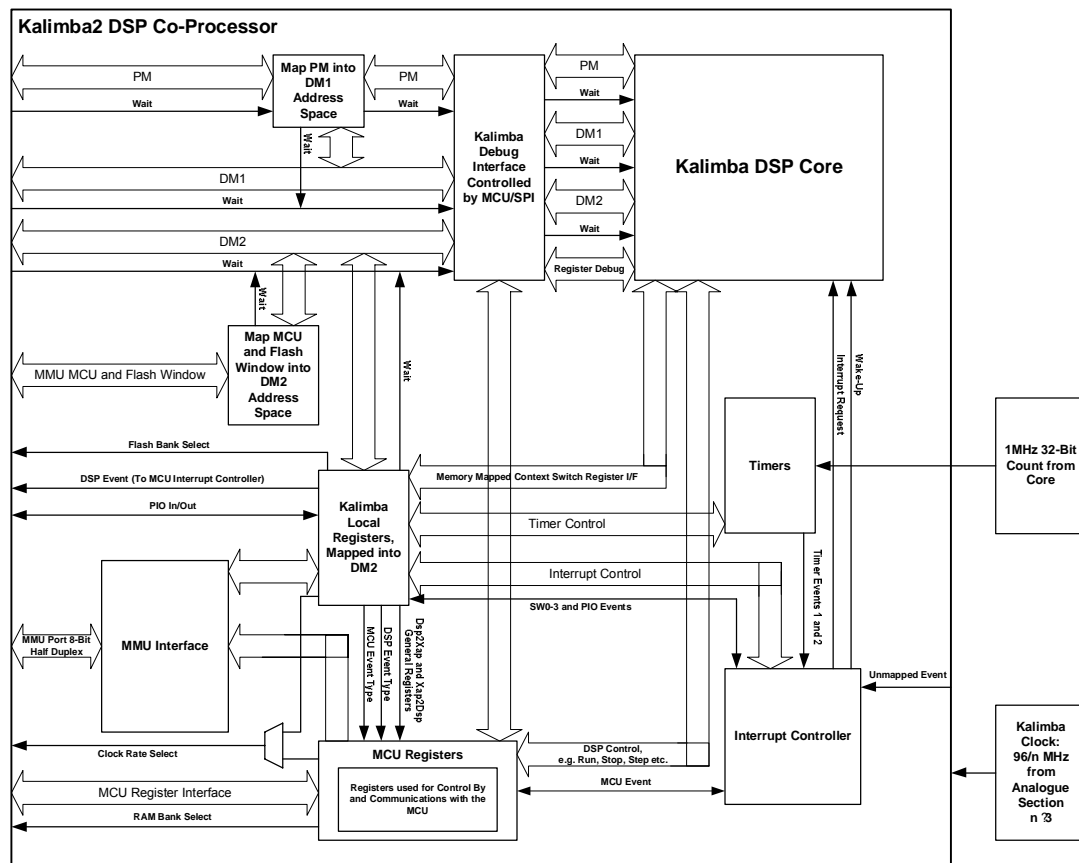


Figure 8.1: Kalimba DSP Peripheral Interfaces

The Kalimba DSP peripherals include:

- Memory management unit (MMU) interface, for stream transfers to/from the BlueCore5-Multimedia subsystem
- Three Memory mapped windows into the flash
- Two memory mapped windows into the MCU RAM
- Memory-mapped register interface into the MCU ram and I/O map
- Program memory window into flash with 64-word direct cache
- Two 1  $\mu$ s timers
- Interrupt controller with three priority levels and wake up from sleep
- Memory mapped access to the DSP program memory through DM1
- Clock rate divider controllable by both the DSP and the MCU
- Debug interface

## 8.1 MMU Interface

The MMU Interface on BlueCore5-Multimedia for the Kalimba DSP contains four virtual read ports and four virtual write ports; an example of the usage of these ports is in Figure 8.2.

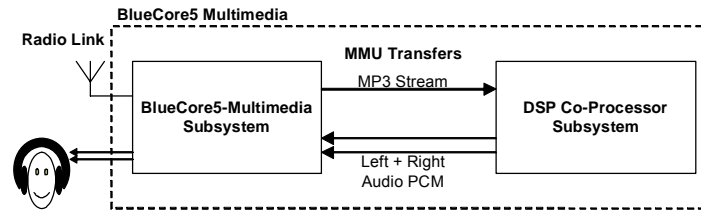


Figure 8.2: Example of MMU Interface Usage for a Wireless MP3 Player

### 8.1.1 Read Ports

Eight virtual read ports appear as memory mapped registers in DM2. The ports have the ability to use:

- 8-bit, 16-bit or 24-bit word size
- Byte swap capability (little endian/big endian)
- Sign-extension, if required

### 8.1.2 Write Ports

Eight virtual write ports appear as memory mapped registers in DM2. The ports have the ability to use:

- 8-bit, 16-bit or 24-bit word size
- Byte swap capability (little endian/big endian)
- 16-bit saturation, if required

## 8.2 DSP Timers

The features of the Kalimba DSP timer are as follows:

- A 48-bit TIMER\_TIME register (read-only) clocked @ 1MHz
- Two trigger value registers, each, if enabled, cause an appropriate interrupt (LS 24 bits of TIMER\_TIME only used for trigger values)

See Appendix B for control register details.

## 8.3 Kalimba Interrupt Controller

This section covers the functionality of Kalimba DSP during interrupts looking at the events of the Kalimba DSP core and the interrupt controller, as well as investigating the interrupt controller registers.

### 8.3.1 DSP Core Functionality During Interrupt

Upon reception of an interrupt, this is when the IRQ is high; the DSP core performs the following:

- `rIntLink` is loaded with the contents of current Program counter
- Program counter is loaded with the address of the interrupt service routine, e.g. `0x0002`
- The flags register is saved
- Perform interrupt service routine
- If enabled, switches to faster interrupt clock rate

When the interrupt service routine completes the DSP needs to return to the routine it had been running prior to the interrupt; this executed with a `rti` instruction. The `rti` instruction carries out the following:

- Restores the `rFlags` register to the non-interrupt state
- Loads the program counter(PC) with the contents of the `rIntLink` register

**Note:**

Saving/restoring of further registers is up to the programmer.

### 8.3.2 Interrupt Controller Functionality

The functionality of the interrupt controller is:

- Selectable interrupt sources:
  - Timer1 and Timer2
  - MCUEvent
  - PIOEvent
  - BufferError/Software error
  - SwEvent0, SwEvent1, SwEvent2 and SwEvent3
  - 16 MCUs interrupt sources made available to the DSP
- Three interrupt priority levels as well as wake-up from sleep
- Registers to save and restore the interrupt controller which allows for nested interrupts
- Optional event signal to cause clock rate change

See Appendix B for control register details.

## 8.4 Generation of MCU Interrupt

Writing to the DSP2MCU\_EVENT\_DATA register causes an interrupt to be sent to the MCU, the value of this register can be seen by the MCU. In a similar way the MCU can generate an interrupt to the DSP, with the event type being stored in the MCU2DSP\_EVENT\_DATA register. These registers can be used to pass messages between MCU and DSP and vice versa.

See Appendix B for control register details.

## 8.5 PIO Control

This section describes the interface between the Kalimba DSP and the programmable I/O (PIO) of the BlueCore5-Multimedia. Control of the PIO from the Kalimba DSP is as follows:

- Kalimba DSP can read BlueCore5-Multimedia PIO lines
- Under the control of the MCU the Kalimba DSP can write to PIO lines
- Under the control of the MCU the Kalimba DSP can change the direction of PIO lines
- PIO line change can generate a Kalimba DSP interrupt

The MCU controls which PIO bits have write access permission for the Kalimba DSP. This information is set through VM functions.

See Appendix B for control register details.

## 8.6 MCU Memory Windows in DM2

Two windows in DM2 shown in Figure 5.1 allow the Kalimba DSP to access MCU memory. The primary use of this memory window is for message passing and control information.

The MCU controls access to this window. A start address and a size for each window can be set, and whether the DSP has read access or read/write access. This information is set by the MCU firmware used.

See Appendix B for control register details.

## 8.7 Flash Memory Windows in DM2

The purpose of the Flash Memory Windows in DM2 is to permit the DSP access to the external flash (up to 32Mbit), this could be, for example, to access further coefficients, download a new program. The window size is 4Kwords and the FLASH\_BANK\_SELECT registers selects which 4K block is visible to the DSP.

See Appendix B for control register details.



## 8.8 PM Window in DM1

The PM window within the DM1 memory bank permits the DSP to change its own program or to be extra data memory (16-bit). For safety, it is possible to disable the window.

The MCU must enable the DSP to have access to the PM mapped into DM1. The DSP must then enable access as well.

See Appendix B for control register details.

## 8.9 PM Flash Window with 64-word Direct Cache

The PM flash window permits the DSP to access program memory directly from flash/ROM. DSP memory-mapped registers control the start address and size of this flash space.

The 32-bit DSP PM bus is mapped into the 16-bit flash space with alternate MS and LS words being stored in flash.

The MCU must enable the DSP to have access to the PM flash window. The DSP must the enable access too. When this interface is enabled, a 64-word direct cache is mapped into the upper 64 words of physical program RAM, allowing zero-overhead accesses for cached instructions.

See Appendix B for control register details.

## 8.10 MCU I/O Map Memory-Mapped Interface

The DSP may read or write to any of the MCU's IO map through a memory-mapped interface. For safety, this interface must first be enabled by the MCU. This interface is utilised by writing the required address to the DSP memory mapped `MCU_REGS_ADDRESS` and then reading from or writing to `MCU_REGS_DATA`

See Appendix B for control register details.

## 8.11 General Registers

The General Registers are for communication between the MCU and the Kalimba DSP.

See Appendix B for control register details.

## 8.12 Clock Rate Divider Control

The Kalimba DSP may control its own clock frequency. It may also force the MCU/MMU clock to operate at a faster, 32MHz clock frequency for accelerated MMU port transfers by setting a DSP memory-mapped register

See Appendix B for control register details.

## 8.13 Debugging

The MCU registers and the serial peripheral interface (SPI) can:

- Read and write any of the DSP core's internal registers
- Control: Single Step, Run, Stop, Breakpoints, etc
- Read and write any individual location in PM, DM1 or DM2, as seen by the DSP

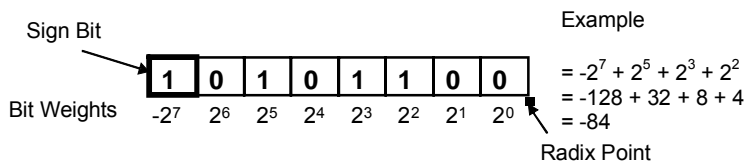
## Appendix A: Number Representation

The number representation used by the Kalimba DSP is outlined in this Appendix.

### A.1 Binary Integer Representation

Two's complement

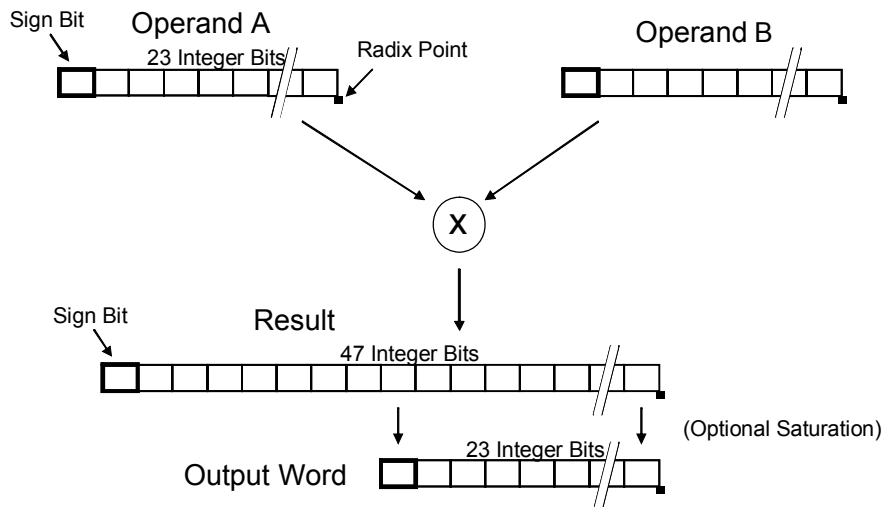
Only 8 bits shown for clarity



### A.2 Binary Fractional Representation

Only 8 bits shown for clarity

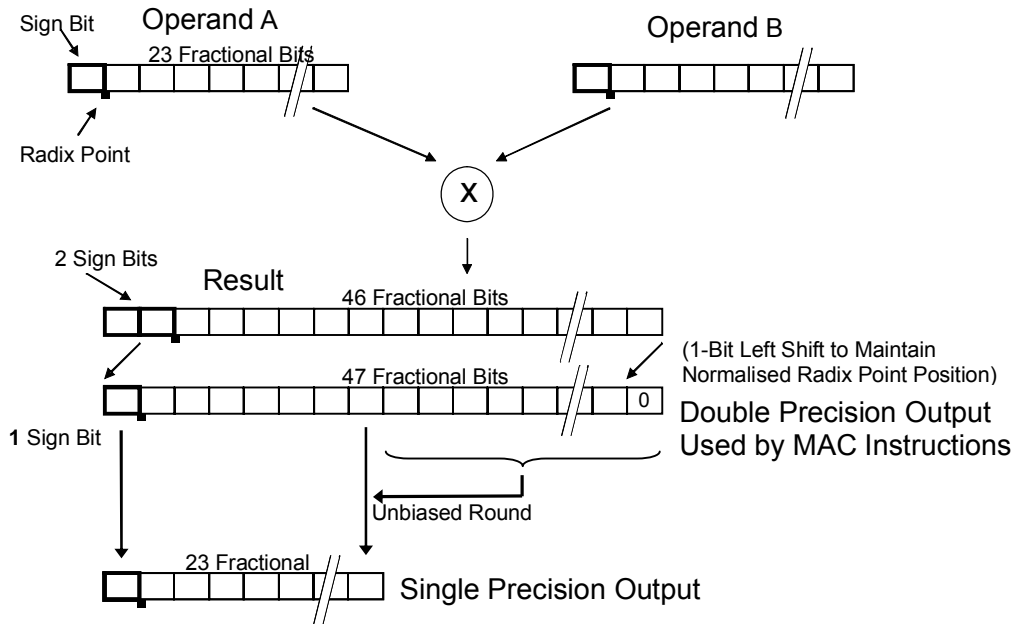
### A.3 Integer Multiplication



Example 1:  
Operand A = 123  
Operand B = 456  
Output Word = 56088

Example 2:  
Operand A = 12345  
Operand B = 67890  
Output Word = 8388607 (With Saturation)  
Output Word = -758750 (Without Saturation)

### A.4 Fractional Multiplication



Example:

Operand A = 0x654321 = 0.79111111  
 Operand B = 0x123456 = 0.14222217  
 Output word = 0x0E66D7F2822C  
               = 0.11251353589 (Double Precision)  
               = 0x0E66D8 (Unbiased Round)  
               = 0.11251354 (Single Precision)



## Appendix B: DSP Memory Mapped Registers

### B.5 DSP Memory Mapped I/O

As described in Section 5.1.3, the DSP memory mapped I/O forms a reserved part of the DM2 memory map. Appendix B Table 1 lists the memory-mapped registers.

| Address | Name                       | Size (Bit) | RW | Description                                                                |
|---------|----------------------------|------------|----|----------------------------------------------------------------------------|
| 0xFF00  | INT_SW_ERROR_EVENT_TRIGGER | 0          | RW | Reading or Writing causes a software error event                           |
| 0xFF01  | INT_GBL_ENABLE             | 1          | RW | Resets interrupt controller state                                          |
| 0xFF02  | INT_ENABLE                 | 1          | RW | Enable searching for an interrupt source                                   |
| 0xFF03  | INT_CLK_SWITCH_EN          | 1          | RW | Enable switching of DSP clock to a special interrupt clock rate            |
| 0xFF04  | INT_SOURCES_EN             | 9          | RW | Enables interrupt sources for Kalimba                                      |
| 0xFF05  | INT_PRIORITIES             | 18         | RW | Set priority levels of individual interrupts                               |
| 0xFF06  | INT_LOAD_INFO              | 14         | RW | Restore information of lowest priority interrupt during a nested interrupt |
| 0xFF07  | INT_ACK                    | 1          | RW | Clears current interrupt request                                           |
| 0xFF08  | INT_SOURCE                 | 5          | R  | Contains current interrupt source                                          |
| 0xFF09  | INT_SAVE_INFO              | 13         | R  | Save information of lowest priority interrupt during a nested interrupt    |
| 0xFF0A  | DSP2MCU_EVENT_DATA         | 16         | RW | Interrupt event data to MCU                                                |
| 0xFF0B  | MCU2DSP_EVENT_DATA         | 16         | R  | Interrupt event data from MCU                                              |
| 0xFF0C  | TIMER1_EN                  | 1          | RW | Enable Timer1 interrupt                                                    |
| 0xFF0D  | TIMER2_EN                  | 1          | RW | Enable Timer2 interrupt                                                    |
| 0xFF0E  | TIMER1_TRIGGER             | 24         | RW | Timer1 trigger value                                                       |
| 0xFF0F  | TIMER2_TRIGGER             | 24         | RW | Timer2 trigger value                                                       |
| 0xFF10  | WRITE_PORT0_DATA           | 8/16/24    | W  | Write port 0                                                               |
| 0xFF11  | WRITE_PORT1_DATA           | 8/16/24    | W  | Write port 1                                                               |
| 0xFF12  | WRITE_PORT2_DATA           | 8/16/24    | W  | Write port 2                                                               |
| 0xFF13  | WRITE_PORT3_DATA           | 8/16/24    | W  | Write port 3                                                               |
| 0xFF14  | WRITE_PORT4_DATA           | 8/16/24    | W  | Write port 4                                                               |
| 0xFF15  | WRITE_PORT5_DATA           | 8/16/24    | W  | Write port 5                                                               |
| 0xFF16  | WRITE_PORT6_DATA           | 8/16/24    | W  | Write port 6                                                               |
| 0xFF17  | WRITE_PORT7_DATA           | 8/16/24    | W  | Write port 7                                                               |
| 0xFF18  | READ_PORT0_DATA            | 8/16/24    | R  | Read port 0                                                                |
| 0xFF19  | READ_PORT1_DATA            | 8/16/24    | R  | Read port 1                                                                |
| 0xFF1A  | READ_PORT2_DATA            | 8/16/24    | R  | Read port 2                                                                |
| 0xFF1B  | READ_PORT3_DATA            | 8/16/24    | R  | Read port 3                                                                |
| 0xFF1C  | READ_PORT4_DATA            | 8/16/24    | R  | Read port 0                                                                |
| 0xFF1D  | READ_PORT5_DATA            | 8/16/24    | R  | Read port 1                                                                |



| Address | Name                     | Size (Bit) | RW | Description                                                                               |
|---------|--------------------------|------------|----|-------------------------------------------------------------------------------------------|
| 0xFF1E  | READ_PORT6_DATA          | 8/16/24    | R  | Read port 2                                                                               |
| 0xFF1F  | READ_PORT7_DATA          | 8/16/24    | R  | Read port 3                                                                               |
| 0xFF20  | PORT_BUFFER_SET          | 0          | RW | Background (write)/Foreground (read) MMU buffer set                                       |
| 0xFF21  | MM_DOLOOP_START          | 16         | RW | Start address of zero overhead loop                                                       |
| 0xFF22  | MM_DOLOOP_END            | 16         | RW | End address of zero overhead loop                                                         |
| 0xFF23  | MM_QUOTIENT              | 24         | RW | Quotient result of division instruction                                                   |
| 0xFF24  | MM_REM                   | 24         | RW | Remainder result of division instruction                                                  |
| 0xFF25  | GENERAL_FROM_MCU0        | 16         | R  | For message passing from on board MCU to the Kalimba                                      |
| 0xFF26  | GENERAL_FROM_MCU1        | 16         | R  | Register for message passing from on board MCU to the Kalimba                             |
| 0xFF27  | GENERAL_FROM_MCU2        | 16         | R  | Register for message passing from on board MCU to the Kalimba                             |
| 0xFF28  | GENERAL_FROM_MCU3        | 16         | R  | Register for message passing from on board MCU to the Kalimba                             |
| 0xFF29  | GENERAL_TO_MCU0          | 16         | RW | Register for message passing from Kalimba to the on board MCU                             |
| 0xFF2A  | GENERAL_TO_MCU1          | 16         | RW | Register for message passing from Kalimba to the on board MCU                             |
| 0xFF2B  | GENERAL_TO_MCU2          | 16         | RW | Register for message passing from Kalimba to the on board MCU                             |
| 0xFF2C  | GENERAL_TO_MCU3          | 16         | RW | Register for message passing from Kalimba to the on board MCU                             |
| 0xFF2D  | CLOCK_DIVIDE_RATE        | 4          | RW | Configuration for clock frequency used by Kalimba during normal operation                 |
| 0xFF2E  | INT_CLOCK_DIVIDE_RATE    | 4          | RW | Configuration for clock frequency used by Kalimba during interrupt service, when selected |
| 0xFF2F  | PIO_IN                   | 24         | R  | Programmable input register used by DSP to read PIO0 –23 and AIO0-3                       |
| 0xFF30  | PIO_OUT                  | 24         | RW | Programmable output register used by DSP to write PIO0 –23 and AIO0-3                     |
| 0xFF31  | PIO_EVENT_EN_MASK        | 24         | RW | Used to select which PIOs and AIOs are used for interrupt sources                         |
| 0xFF32  | INT_SW0_EVENT            | 1          | RW | Select software event 0 to cause interrupt request on Kalimba                             |
| 0xFF33  | INT_SW1_EVENT            | 1          | RW | Select software event 1 to cause interrupt request on Kalimba                             |
| 0xFF34  | INT_SW2_EVENT            | 1          | RW | Select software event 2 to cause interrupt request on Kalimba                             |
| 0xFF35  | INT_SW3_EVENT            | 1          | RW | Select software event 3 to cause interrupt request on Kalimba                             |
| 0xFF36  | FLASH_WINDOW1_START_ADDR | 24         | RW | Start address for 4K block 1 mapped into DM2, see Figure 5.1                              |
| 0xFF37  | FLASH_WINDOW2_START_ADDR | 24         | RW | Start address for 4K block 2 mapped into DM2, see Figure 5.1                              |



| Address | Name                     | Size (Bit) | RW | Description                                                                                                        |
|---------|--------------------------|------------|----|--------------------------------------------------------------------------------------------------------------------|
| 0xFF38  | FLASH_WINDOW3_START_ADDR | 24         | RW | Start address for ~4K block 3 mapped into DM2, see Figure 5.1                                                      |
| 0xFF39  | NOSIGNX_GENREGS          | 1          | RW | General sign extension enable                                                                                      |
| 0xFF3A  | NOSIGNX_MCUWIN1          | 1          | RW | Enable sign extension in MCU window 1                                                                              |
| 0xFF3B  | NOSIGNX_MCUWIN2          | 1          | RW | Enable sign extension in MCU window 2                                                                              |
| 0xFF3C  | NOSIGNX_FLASHWIN1        | 1          | RW | Enable sign extension in flash window 1                                                                            |
| 0xFF3D  | NOSIGNX_FLASHWIN2        | 1          | RW | Enable sign extension in flash window 2                                                                            |
| 0xFF3E  | NOSIGNX_FLASHWIN3        | 1          | RW | Enable sign extension in flash window 3                                                                            |
| 0xFF3F  | NOSIGNX_PMWIN            | 1          | RW | Enable sign extension in PM window                                                                                 |
| 0xFF40  | PM_WIN_ENABLE            | 1          | RW | Allows program memory to be mapped into DM1 memory map see Figure 5.1                                              |
| 0xFF41  | STACK_START_ADDR         | 16         | RW | Stack start address                                                                                                |
| 0xFF42  | STACK_END_ADDR           | 16         | RW | Stack end address (> Start address, stack build upwards)                                                           |
| 0xFF43  | STACK_POINTER            | 16         | RW | Current stack pointer                                                                                              |
| 0xFF44  | NUM_RUN_CLKS_MS          | 8          | RW | MS 8 bits of Number of clocks encountered whilst running since counter last reset. Writing to register resets it.  |
| 0xFF45  | NUM_RUN_CLKS_LS          | 24         | RW | LS 24 bits of Number of clocks encountered whilst running since counter last reset. Writing to register resets it. |
| 0xFF46  | NUM_INSTRS_MS            | 8          | RW | MS 8 bits of Number of instructions encountered since counter last reset. Writing to register resets it.           |
| 0xFF47  | NUM_INSTRS_LS            | 24         | RW | LS 24 bits of Number of instructions encountered since counter last reset. Writing to register resets it.          |
| 0xFF48  | NUM_STALLS_MS            | 8          | RW | MS 8 bits of Number of instruction stalls encountered since counter last reset. Writing to register resets it.     |
| 0xFF49  | NUM_STALLS_LS            | 24         | RW | LS 24 bits of Number of instruction stalls encountered since counter last reset. Writing to register resets it.    |
| 0xFF4A  | TIMER_TIME               | 24         | R  | LS 24 bits of 48-bit timer value increment every 1µs                                                               |
| 0xFF4B  | TIMER_TIME_MS            | 24         | R  | MS 24 bits of 48-bit timer value increment every 1µs                                                               |
| 0xFF4C  | WRITE_PORT0_CONFIG       | 4          | RW | Data size, Endian mode and saturation for write port 0                                                             |
| 0xFF4D  | WRITE_PORT1_CONFIG       | 4          | RW | Data size, Endian mode and saturation for write port 1                                                             |
| 0xFF4E  | WRITE_PORT2_CONFIG       | 4          | RW | Data size, Endian mode and saturation for write port 2                                                             |



| Address | Name                       | Size (Bit) | RW | Description                                                                                               |
|---------|----------------------------|------------|----|-----------------------------------------------------------------------------------------------------------|
| 0xFF4F  | WRITE_PORT3_CONFIG         | 4          | RW | Data size, Endian mode and saturation for write port 3                                                    |
| 0xFF50  | WRITE_PORT4_CONFIG         | 4          | RW | Data size, Endian mode and saturation for write port 4                                                    |
| 0xFF51  | WRITE_PORT5_CONFIG         | 4          | RW | Data size, Endian mode and saturation for write port 5                                                    |
| 0xFF52  | WRITE_PORT6_CONFIG         | 4          | RW | Data size, Endian mode and saturation for write port 6                                                    |
| 0xFF53  | WRITE_PORT7_CONFIG         | 4          | RW | Data size, Endian mode and saturation for write port 7                                                    |
| 0xFF54  | READ_PORT0_CONFIG          | 4          | RW | Data size, Endian mode and sign extension for read port 0                                                 |
| 0xFF55  | READ_PORT1_CONFIG          | 4          | RW | Data size, Endian mode and sign extension for read port 1                                                 |
| 0xFF56  | READ_PORT2_CONFIG          | 4          | RW | Data size, Endian mode and sign extension for read port 2                                                 |
| 0xFF57  | READ_PORT3_CONFIG          | 4          | RW | Data size, Endian mode and sign extension for read port 3                                                 |
| 0xFF58  | READ_PORT4_CONFIG          | 4          | RW | Data size, Endian mode and sign extension for read port 4                                                 |
| 0xFF59  | READ_PORT5_CONFIG          | 4          | RW | Data size, Endian mode and sign extension for read port 5                                                 |
| 0xFF5A  | READ_PORT6_CONFIG          | 4          | RW | Data size, Endian mode and sign extension for read port 6                                                 |
| 0xFF5B  | READ_PORT7_CONFIG          | 4          | RW | Data size, Endian mode and sign extension for read port 7                                                 |
| 0xFF5C  | PM_FLASH_WINDOW_START_ADDR | 24         | RW | Start address in flash for program memory flash window                                                    |
| 0xFF5D  | PM_FLASH_WINDOW_SIZE       | 16         | RW | Size of program memory window in number of instructions (double for number of flash addresses used)       |
| 0xFF5E  | MCUREGS_ENABLE             | 1          | RW | Enables the DSP memory-mapped access to the MCU's IO map. Reading reads the value of the MCU's own enable |
| 0xFF5F  | NOSIGNX_MCUREGS            | 1          | RW | Enables sign extension of the MCU IO map interface                                                        |
| 0xFF60  | MCUREGS_ADDRESS            | 16         | RW | MCU IO map address to read/write                                                                          |
| 0xFF61  | MCUREGS_DATA               | 16         | RW | MCU IO map data read/write                                                                                |
| 0xFF62  | BITREVERSE_DATA            | 24         | RW | Reading this location reads back the bitreverse of the value written to it                                |
| 0xFF63  | INT_MCU_PRIORITIES_MS      | 16         | RW | Priorities for the first 8 of the MCU's interrupt events                                                  |
| 0xFF64  | INT_MCU_PRIORITIES_LS      | 16         | RW | Priorities for the last 8 of the MCU's interrupt events                                                   |
| 0xFF65  | INT_MCU_SOURCES_EN         | 16         | RW | Enables for the MCU's 16 interrupt events                                                                 |
| 0xFF66  | PIO_DIR                    | 24         | RW | Direction enables for the DSP's PIOs                                                                      |



| Address | Name                        | Size (Bit) | RW | Description                                                               |
|---------|-----------------------------|------------|----|---------------------------------------------------------------------------|
| 0xFF67  | ADDSUB_SATURATE_ON_OVERFLOW | 1          | RW | 1 to enable saturation on overflow for addition or subtraction in the ALU |
| 0xFF68  | FORCE_FAST_MMU_CLOCK        | 1          | RW | 1 to force the MMU/XAP clock to operate at full speed                     |
| 0xFF69  | BITSERIAL_RWB_LENGTH        | 17         | RW | Control register to start bitserial data transfers from the DSP           |
| 0xFF6A  | BITSERIAL_BUFFER_LEVEL      | 16         | R  | Bitserial buffer level status                                             |
| 0xFF6B  | BITSERIAL_EVENT             | 1          | RW | Bitserial Event status                                                    |
| 0xFF6C  | BITSERIAL_BUSY              | 1          | R  | Bitserial Busy status                                                     |

Appendix B Table 1: DSP Memory Mapped I/O





## B.2 MMU Interface DSP Registers

Tables in this section list a group of registers used for communication and control between the Kalimba DSP and the MCU on the BlueCore5-Multimedia.

| WRITE_PORT[0/1/2/3/4/5/6/7]_DATA |                                                       | Description    |
|----------------------------------|-------------------------------------------------------|----------------|
| Bit                              |                                                       |                |
| 23:0                             |                                                       | Data bits 0-23 |
|                                  |                                                       |                |
| <b>Note</b>                      | Memory mapped location for the write port 0 to 7 data |                |

Appendix B Table 2: Data Bits for Write Port 0 to 7

| WRITE_PORT[0/1/2/3/4/5/6/7]_CONFIG                                           |   |   |     | Description                                                                                                                                         |
|------------------------------------------------------------------------------|---|---|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| Bit                                                                          |   |   |     |                                                                                                                                                     |
| 23:4                                                                         | 3 | 2 | 1:0 | 00 = 8-bit write<br>01 = 16-bit write<br>10 = 24-bit write<br>1 = Big endian<br>0 = Little endian<br>1=16-bit saturate<br>0=no saturation<br>Unused |
|                                                                              |   |   |     |                                                                                                                                                     |
| <b>Note</b>                                                                  |   |   |     |                                                                                                                                                     |
| This register is cleared on system reset                                     |   |   |     |                                                                                                                                                     |
| The stereo audio interface of BlueCore5-Multimedia uses Little Endian format |   |   |     |                                                                                                                                                     |

Appendix B Table 3: Configuration Bits for Write Port 0 to 7

| READ_PORT[0/1/2/3/4/5/6/7]_DATA |                                                      | Description    |
|---------------------------------|------------------------------------------------------|----------------|
| Bit                             |                                                      |                |
| 23:0                            |                                                      | Data bits 0-23 |
|                                 |                                                      |                |
| <b>Note</b>                     | Memory mapped location for the read port 0 to 7 data |                |

Appendix B Table 4: Data Bits for Read Port 0 to 7



| READ_PORT[0/1/2/3/4/5/6/7]_CONFIG |                                                                                                                          |   |     | Description                                             |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------|---|-----|---------------------------------------------------------|
| Bit                               |                                                                                                                          |   |     |                                                         |
| 23:4                              | 3                                                                                                                        | 2 | 1:0 |                                                         |
|                                   |                                                                                                                          |   |     | 00 = 8-bit read<br>01 = 16-bit read<br>10 = 24-bit read |
|                                   |                                                                                                                          |   |     | 1 = Big endian<br>0 = Little endian                     |
|                                   |                                                                                                                          |   |     | 1 = No sign extension<br>0 = Sign extension             |
|                                   |                                                                                                                          |   |     | Unused                                                  |
| <b>Note</b>                       | This register is cleared on system reset<br>The stereo audio interface of BlueCore5-Multimedia uses Little Endian format |   |     |                                                         |


Appendix B Table 5: Configuration Bits for Read Port 0 to 7

| PORT_BUFFER_SET |                                                                         | Description                                                                                                                                                                              |
|-----------------|-------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bit             |                                                                         |                                                                                                                                                                                          |
| 23:0            |                                                                         |                                                                                                                                                                                          |
|                 |                                                                         | Reading: Starts foreground buffer set – DSP stalls until complete<br>Writing: Starts background buffer set – DSP only stalls if an MMU port access is made before buffer set is complete |
| <b>Note</b>     | This register is of width zero – always reads value written is not used |                                                                                                                                                                                          |


Appendix B Table 6: Configuration Bits for Read Port 0 to 7

### B.3 Timers DSP Registers

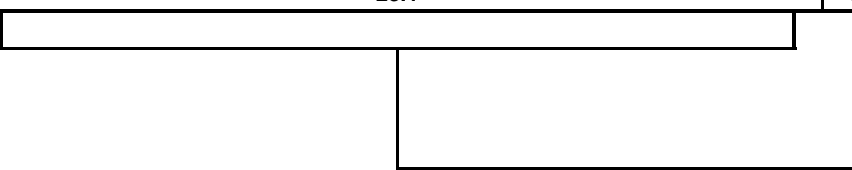
Three groups of registers make up the Kalimba DSP timer, these are:

| TIMER_TIME[MS]                                                                     |                                                                                                      | Description                                 |
|------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|---------------------------------------------|
| Bit                                                                                |                                                                                                      |                                             |
| 23:0                                                                               |                                                                                                      |                                             |
|  |                                                                                                      | Data bits 0-23 or 47-24 for 1µs count value |
| <b>Note</b>                                                                        | A 48-bit read only register, the value is incremented every 1µs, split into two 24-bit DSP registers |                                             |

Appendix B Table 7: The Timer Time Registers

| TIMER[1/2]_TRIGGER                                                                  |                                                                                                                                                              | Description                                |
|-------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------|
| Bit                                                                                 |                                                                                                                                                              |                                            |
| 23:0                                                                                |                                                                                                                                                              |                                            |
|  |                                                                                                                                                              | Data bits 0-23 for 1µs trigger count value |
| <b>Note</b>                                                                         | A 24-bit trigger value. If the appropriate enable register, TIMER[1/2]_EN, is set then an interrupt is generated when $TIMER\_TIME \geq TIMER[1/2]\_TRIGGER$ |                                            |

Appendix B Table 8: The Threshold Value Registers for Timer Trigger 1 and 2

| TIMER[1/2]_EN                                                                        |                                          |   | Description                                               |
|--------------------------------------------------------------------------------------|------------------------------------------|---|-----------------------------------------------------------|
| Bit                                                                                  |                                          |   |                                                           |
| 23:1                                                                                 |                                          | 0 |                                                           |
|  |                                          |   | 1 = Enable timer interrupt<br>0 = Disable timer interrupt |
|                                                                                      |                                          |   | Unused                                                    |
| <b>Note</b>                                                                          | This register is cleared on system reset |   |                                                           |

Appendix B Table 9: Enable Bits for Timer 1 and Timer 2 Interrupts



## B.4 Interrupt Controller DSP Registers

This section covers the registers concerned with interrupt control on the DSP.

| INT_GBL_ENABLE |                                          | Description                                    |
|----------------|------------------------------------------|------------------------------------------------|
| Bit            |                                          |                                                |
| 23:1           | 0                                        | 0 = Reset interrupt controller state<br>Unused |
|                |                                          |                                                |
| <b>Note</b>    | This register is cleared on system reset |                                                |

Appendix B Table 10: Interrupt Controller State Reset Bit

| INT_ENABLE  |                                          | Description                                                                                             |
|-------------|------------------------------------------|---------------------------------------------------------------------------------------------------------|
| Bit         |                                          |                                                                                                         |
| 23:1        | 0                                        | 1 = Enable searching for an interrupt source<br>0 = Disable searching for an interrupt source<br>Unused |
|             |                                          |                                                                                                         |
| <b>Note</b> | This register is cleared on system reset |                                                                                                         |

Appendix B Table 11: Enable Interrupt Searching Bit

| INT_CLK_SWITCH_EN |                                                                                                                                         | Description                                                                                                       |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|
| Bit               |                                                                                                                                         |                                                                                                                   |
| 23:1              | 0                                                                                                                                       | 1 = Enable DSP clock rate switch during interrupt<br>0 = Disable DSP clock rate switch during interrupt<br>Unused |
|                   |                                                                                                                                         |                                                                                                                   |
| <b>Note</b>       | The DSP special clock rate during interrupt is DSP_INT_CLOCK_RATE described in Section 8.12<br>This register is cleared on system reset |                                                                                                                   |

Appendix B Table 12: Enable Interrupt Clock Rate Bit



| INT_SOURCES_EN                                                                                                                                                                                                                                                            |   |   |   |   |   |   |   |   |   | Description                                               |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|---|---|---|---|---|---|---|---|-----------------------------------------------------------|
| Bit                                                                                                                                                                                                                                                                       |   |   |   |   |   |   |   |   |   |                                                           |
| 23:9                                                                                                                                                                                                                                                                      | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |                                                           |
|                                                                                                                                                                                                                                                                           |   |   |   |   |   |   |   |   |   | Timer1 interrupt<br>1 = Enable<br>0 = Disable             |
|                                                                                                                                                                                                                                                                           |   |   |   |   |   |   |   |   |   | Timer2 interrupt<br>1 = Enable<br>0 = Disable             |
|                                                                                                                                                                                                                                                                           |   |   |   |   |   |   |   |   |   | MCU interrupt<br>1 = Enable<br>0 = Disable                |
|                                                                                                                                                                                                                                                                           |   |   |   |   |   |   |   |   |   | PIO line change interrupt<br>1 = Enable<br>0 = Disable    |
|                                                                                                                                                                                                                                                                           |   |   |   |   |   |   |   |   |   | MMU unmapped event interrupt<br>1 = Enable<br>0 = Disable |
|                                                                                                                                                                                                                                                                           |   |   |   |   |   |   |   |   |   | Software0 event interrupt<br>1 = Enable<br>0 = Disable    |
|                                                                                                                                                                                                                                                                           |   |   |   |   |   |   |   |   |   | Software1 event interrupt<br>1 = Enable<br>0 = Disable    |
|                                                                                                                                                                                                                                                                           |   |   |   |   |   |   |   |   |   | Software2 event interrupt<br>1 = Enable<br>0 = Disable    |
|                                                                                                                                                                                                                                                                           |   |   |   |   |   |   |   |   |   | Software3 event interrupt<br>1 = Enable<br>0 = Disable    |
|                                                                                                                                                                                                                                                                           |   |   |   |   |   |   |   |   |   | Unused                                                    |
| <b>Note</b>                                                                                                                                                                                                                                                               |   |   |   |   |   |   |   |   |   |                                                           |
| Bit 0: INT_SOURCE_TIMER1_POSN      5: INT_SOURCE_SW0_POSN<br>1: INT_SOURCE_TIMER2_POSN      6: INT_SOURCE_SW1_POSN<br>2: INT_SOURCE_MCU_POSN          7: INT_SOURCE_SW2_POSN<br>3: INT_SOURCE_PIO_POSN          8: INT_SOURCE_SW3_POSN<br>4: INT_SOURCE_MMU_UNMAPPED_POSN |   |   |   |   |   |   |   |   |   |                                                           |

Appendix B Table 13: Interrupt Source Enable Bits



| INT_PRIORITIES |                                                                                                                |       |       |       |     |     |     |     |     | Description                           |
|----------------|----------------------------------------------------------------------------------------------------------------|-------|-------|-------|-----|-----|-----|-----|-----|---------------------------------------|
| Bit            |                                                                                                                |       |       |       |     |     |     |     |     |                                       |
| 23:18          | 17:16                                                                                                          | 15:14 | 13:12 | 11:10 | 9:8 | 7:6 | 5:4 | 3:2 | 1:0 |                                       |
|                |                                                                                                                |       |       |       |     |     |     |     |     | Timer1 interrupt priority             |
|                |                                                                                                                |       |       |       |     |     |     |     |     | Timer2 interrupt priority             |
|                |                                                                                                                |       |       |       |     |     |     |     |     | MCU interrupt priority                |
|                |                                                                                                                |       |       |       |     |     |     |     |     | PIO line change interrupt priority    |
|                |                                                                                                                |       |       |       |     |     |     |     |     | MMU unmapped event interrupt priority |
|                |                                                                                                                |       |       |       |     |     |     |     |     | Software0 event interrupt priority    |
|                |                                                                                                                |       |       |       |     |     |     |     |     | Software1 event interrupt priority    |
|                |                                                                                                                |       |       |       |     |     |     |     |     | Software2 event interrupt priority    |
|                |                                                                                                                |       |       |       |     |     |     |     |     | Software3 event interrupt priority    |
|                |                                                                                                                |       |       |       |     |     |     |     |     | Unused                                |
| <b>Note</b>    | Priority level for each interrupt is a 2-bit value where the range is 3 (highest priority level) to 0 (lowest) |       |       |       |     |     |     |     |     |                                       |

Appendix B Table 14: Interrupt Priority Level Bits



| INT_MCU_SOURCES_EN |                             |    |    |    |    |    |   |                             |   |   |   |   |   |   |   | Description                       |
|--------------------|-----------------------------|----|----|----|----|----|---|-----------------------------|---|---|---|---|---|---|---|-----------------------------------|
| Bit                |                             |    |    |    |    |    |   |                             |   |   |   |   |   |   |   |                                   |
| 23:16              | 15                          | 14 | 13 | 12 | 11 | 10 | 9 | 8                           | 7 | 6 | 5 | 4 | 3 | 2 | 1 |                                   |
|                    |                             |    |    |    |    |    |   |                             |   |   |   |   |   |   |   | MCU Interrupt source 0<br>Enable  |
|                    |                             |    |    |    |    |    |   |                             |   |   |   |   |   |   |   | MCU Interrupt source 1<br>Enable  |
|                    |                             |    |    |    |    |    |   |                             |   |   |   |   |   |   |   | MCU Interrupt source 2<br>Enable  |
|                    |                             |    |    |    |    |    |   |                             |   |   |   |   |   |   |   | MCU Interrupt source 3<br>Enable  |
|                    |                             |    |    |    |    |    |   |                             |   |   |   |   |   |   |   | MCU Interrupt source 4<br>Enable  |
|                    |                             |    |    |    |    |    |   |                             |   |   |   |   |   |   |   | MCU Interrupt source 5<br>Enable  |
|                    |                             |    |    |    |    |    |   |                             |   |   |   |   |   |   |   | MCU Interrupt source 6<br>Enable  |
|                    |                             |    |    |    |    |    |   |                             |   |   |   |   |   |   |   | MCU Interrupt source 7<br>Enable  |
|                    |                             |    |    |    |    |    |   |                             |   |   |   |   |   |   |   | MCU Interrupt source 8<br>Enable  |
|                    |                             |    |    |    |    |    |   |                             |   |   |   |   |   |   |   | MCU Interrupt source 9<br>Enable  |
|                    |                             |    |    |    |    |    |   |                             |   |   |   |   |   |   |   | MCU Interrupt source 10<br>Enable |
|                    |                             |    |    |    |    |    |   |                             |   |   |   |   |   |   |   | MCU Interrupt source 11<br>Enable |
|                    |                             |    |    |    |    |    |   |                             |   |   |   |   |   |   |   | MCU Interrupt source 12<br>Enable |
|                    |                             |    |    |    |    |    |   |                             |   |   |   |   |   |   |   | MCU Interrupt source 13<br>Enable |
|                    |                             |    |    |    |    |    |   |                             |   |   |   |   |   |   |   | MCU Interrupt source 14<br>Enable |
|                    |                             |    |    |    |    |    |   |                             |   |   |   |   |   |   |   | MCU Interrupt source 15<br>Enable |
| Unused             |                             |    |    |    |    |    |   |                             |   |   |   |   |   |   |   |                                   |
| <b>Note</b>        | Bit 0: INT_MCU_SOURCE_TIMER |    |    |    |    |    |   | 8: INT_MCU_SOURCE_BQI       |   |   |   |   |   |   |   |                                   |
|                    | 1: INT_MCU_SOURCE_TIMER2    |    |    |    |    |    |   | 9: INT_MCU_SOURCE_SPI       |   |   |   |   |   |   |   |                                   |
|                    | 2: INT_MCU_SOURCE_RADIO_TX  |    |    |    |    |    |   | 10: INT_MCU_SOURCE_UNMAPPED |   |   |   |   |   |   |   |                                   |
|                    | 3: INT_MCU_SOURCE_RADIO_RX  |    |    |    |    |    |   | 11: INT_MCU_SOURCE_SW0      |   |   |   |   |   |   |   |                                   |
|                    | 4: INT_MCU_SOURCE_DATA_OUT  |    |    |    |    |    |   | 12: INT_MCU_SOURCE_SW1      |   |   |   |   |   |   |   |                                   |
|                    | 5: INT_MCU_SOURCE_DATA_IN   |    |    |    |    |    |   | 13: INT_MCU_SOURCE_SW2      |   |   |   |   |   |   |   |                                   |
|                    | 6: INT_MCU_SOURCE_VOICE_OUT |    |    |    |    |    |   | 14: INT_MCU_SOURCE_SW3      |   |   |   |   |   |   |   |                                   |
|                    | 7: INT_MCU_SOURCE_VOICE_IN  |    |    |    |    |    |   | 15: INT_MCU_SOURCE_IO_AUX   |   |   |   |   |   |   |   |                                   |

Appendix B Table 15: MCU Interrupt Source Enable Bits



| INT_MCU_PRIORITIES_MS |                                                                                                                |       |       |     |     |     |     |     | Description                    |
|-----------------------|----------------------------------------------------------------------------------------------------------------|-------|-------|-----|-----|-----|-----|-----|--------------------------------|
| Bit                   |                                                                                                                |       |       |     |     |     |     |     |                                |
| 23:16                 | 15:14                                                                                                          | 13:12 | 11:10 | 9:8 | 7:6 | 5:4 | 3:2 | 1:0 |                                |
| □                     | □                                                                                                              | □     | □     | □   | □   | □   | □   | □   | MCU Interrupt source 8 Enable  |
| □                     | □                                                                                                              | □     | □     | □   | □   | □   | □   | □   | MCU Interrupt source 9 Enable  |
| □                     | □                                                                                                              | □     | □     | □   | □   | □   | □   | □   | MCU Interrupt source 10 Enable |
| □                     | □                                                                                                              | □     | □     | □   | □   | □   | □   | □   | MCU Interrupt source 11 Enable |
| □                     | □                                                                                                              | □     | □     | □   | □   | □   | □   | □   | MCU Interrupt source 12 Enable |
| □                     | □                                                                                                              | □     | □     | □   | □   | □   | □   | □   | MCU Interrupt source 13 Enable |
| □                     | □                                                                                                              | □     | □     | □   | □   | □   | □   | □   | MCU Interrupt source 14 Enable |
| □                     | □                                                                                                              | □     | □     | □   | □   | □   | □   | □   | MCU Interrupt source 15 Enable |
| □                     | □                                                                                                              | □     | □     | □   | □   | □   | □   | □   | Unused                         |
| <b>Note</b>           | Priority level for each interrupt is a 2-bit value where the range is 3 (highest priority level) to 0 (lowest) |       |       |     |     |     |     |     |                                |

Appendix B Table 16: MCU Interrupt Source Priorities MS

| INT_MCU_PRIORITIES_LS |                                                                                                                |       |       |     |     |     |     |     | Description                   |
|-----------------------|----------------------------------------------------------------------------------------------------------------|-------|-------|-----|-----|-----|-----|-----|-------------------------------|
| Bit                   |                                                                                                                |       |       |     |     |     |     |     |                               |
| 23:16                 | 15:14                                                                                                          | 13:12 | 11:10 | 9:8 | 7:6 | 5:4 | 3:2 | 1:0 |                               |
| □                     | □                                                                                                              | □     | □     | □   | □   | □   | □   | □   | MCU Interrupt source 0 Enable |
| □                     | □                                                                                                              | □     | □     | □   | □   | □   | □   | □   | MCU Interrupt source 1 Enable |
| □                     | □                                                                                                              | □     | □     | □   | □   | □   | □   | □   | MCU Interrupt source 2 Enable |
| □                     | □                                                                                                              | □     | □     | □   | □   | □   | □   | □   | MCU Interrupt source 3 Enable |
| □                     | □                                                                                                              | □     | □     | □   | □   | □   | □   | □   | MCU Interrupt source 4 Enable |
| □                     | □                                                                                                              | □     | □     | □   | □   | □   | □   | □   | MCU Interrupt source 5 Enable |
| □                     | □                                                                                                              | □     | □     | □   | □   | □   | □   | □   | MCU Interrupt source 6 Enable |
| □                     | □                                                                                                              | □     | □     | □   | □   | □   | □   | □   | MCU Interrupt source 7 Enable |
| □                     | □                                                                                                              | □     | □     | □   | □   | □   | □   | □   | Unused                        |
| <b>Note</b>           | Priority level for each interrupt is a 2-bit value where the range is 3 (highest priority level) to 0 (lowest) |       |       |     |     |     |     |     |                               |

Appendix B Table 17: MCU Interrupt Source Priorities LS





| INT_SOURCE  |                                                                                             | Description              |
|-------------|---------------------------------------------------------------------------------------------|--------------------------|
| Bit         |                                                                                             |                          |
| 23:5        | 4:0                                                                                         |                          |
|             |                                                                                             | Current Interrupt Source |
|             |                                                                                             | Unused                   |
| <b>Note</b> | Current Interrupt Source is a 5-bit value that represents the current interrupt as follows: |                          |
|             | 00000 Timer1 interrupt                                                                      |                          |
|             | 00001 Timer2 interrupt                                                                      |                          |
|             | 00010 MCU interrupt                                                                         |                          |
|             | 00011 PIO line change interrupt                                                             |                          |
|             | 00100 MMU unmapped event interrupt                                                          |                          |
|             | 00101 Software0 event interrupt                                                             |                          |
|             | 00110 Software1 event interrupt priority                                                    |                          |
|             | 00111 Software2 event interrupt                                                             |                          |
|             | 01000 Software3 event interrupt                                                             |                          |
|             | 01001 MCU Timer interrupt                                                                   |                          |
|             | 01010 MCU Timer2 interrupt                                                                  |                          |
|             | 01011 MCU Radio TX interrupt                                                                |                          |
|             | 01100 MCU Radio RX interrupt                                                                |                          |
|             | 01101 MCU Data Out interrupt                                                                |                          |
|             | 01110 MCU Data In interrupt                                                                 |                          |
|             | 01111 MCU Voice Out interrupt                                                               |                          |
|             | 10000 MCU Voice In interrupt                                                                |                          |
|             | 10001 MCU BQI/COEX/DSP/BITSERIAL/IO_AUX2 interrupt                                          |                          |
|             | 10010 MCU SPI USER interrupt                                                                |                          |
|             | 10011 MCU PROC Unmapped interrupt                                                           |                          |
|             | 10100 MCU SW0 interrupt                                                                     |                          |
|             | 10101 MCU SW1 interrupt                                                                     |                          |
|             | 10110 MCU SW2 interrupt                                                                     |                          |
|             | 10111 MCU SW3 interrupt                                                                     |                          |
|             | 11000 MCU IO_AUX event/UART interrupt                                                       |                          |

Appendix B Table 18: Current Interrupt Source Bits



| INT_ACK     |                                                                                                                                                                                                                  | Description                                          |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------|
| Bit         |                                                                                                                                                                                                                  |                                                      |
| 23:1        | 0                                                                                                                                                                                                                | 0 = Clears current interrupt request (IRQ)<br>Unused |
|             |                                                                                                                                                                                                                  |                                                      |
| <b>Note</b> | Writing to this register acknowledges an interrupt request. It de-asserts the IRQ line to Kalimba DSP, reverts the Kalimba clock back to the non interrupt version, and initiates the search for a new interrupt |                                                      |

Appendix B Table19: Interrupt Request Acknowledge Bit

| INT_LOAD_INFO                                                                                                                              |    |      |   |     |     | Description                                                                                                                                                                                                                                                                                                                                |
|--------------------------------------------------------------------------------------------------------------------------------------------|----|------|---|-----|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bit                                                                                                                                        |    |      |   |     |     |                                                                                                                                                                                                                                                                                                                                            |
| 23:14                                                                                                                                      | 13 | 12:8 | 7 | 6:2 | 1:0 | Interrupt priority value to restore see Appendix B Tables 14,16 and 17<br>Interrupt source number (0-24) to restore see Appendix B Table 18<br>Interrupt active signal state to restore<br>Interrupt request number (0-24) to optionally clear see Appendix B Table 18<br>0 = Clear the Interrupt request number in bits 8 to 12<br>Unused |
|                                                                                                                                            |    |      |   |     |     |                                                                                                                                                                                                                                                                                                                                            |
| <b>Note</b>                                                                                                                                |    |      |   |     |     |                                                                                                                                                                                                                                                                                                                                            |
| Used to restore information about a previous lower priority interrupt<br>See the nested interrupts example code for an example of its use. |    |      |   |     |     |                                                                                                                                                                                                                                                                                                                                            |

Appendix B Table 20: Restore Information about a Previous Lower Priority Interrupt



| INT_SAVE_INFO |                                                                                                                          |   |     |     | Description                                                                          |
|---------------|--------------------------------------------------------------------------------------------------------------------------|---|-----|-----|--------------------------------------------------------------------------------------|
| Bit           |                                                                                                                          |   |     |     |                                                                                      |
| 23:13         | 12:8                                                                                                                     | 7 | 6:2 | 1:0 |                                                                                      |
|               |                                                                                                                          |   |     |     | Previous lower priority interrupt priority value see Appendix B Tables 14, 16 and 17 |
|               |                                                                                                                          |   |     |     | Previous lower priority interrupt source number (0-24) see Appendix B Table 18       |
|               |                                                                                                                          |   |     |     | Previous lower priority interrupt active signal state                                |
|               |                                                                                                                          |   |     |     | New interrupt source number (0-24) see Appendix B Table 18.                          |
|               |                                                                                                                          |   |     |     | Unused                                                                               |
| <b>Note</b>   | Used to save information about the current interrupt<br>See the nested interrupts example code for an example of its use |   |     |     |                                                                                      |

Appendix B Table 21: Save Information about Current Interrupt

| INT_SW[0/1/2/3]_EVENT |                                          | Description                                                              |
|-----------------------|------------------------------------------|--------------------------------------------------------------------------|
| Bit                   |                                          |                                                                          |
| 23:1                  | 0                                        |                                                                          |
|                       |                                          | 1 = Software Event (causes interrupt if enabled see Appendix B Table 13) |
|                       |                                          | Unused                                                                   |
| <b>Note</b>           | This register is cleared on system reset |                                                                          |

Appendix B Table 22: Software Event 0 to 3 Interrupt Request



| INT_SW_ERROR |                                                                                                                                                                                                         | Description |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| Bit          |                                                                                                                                                                                                         |             |
| 23:0         |                                                                                                                                                                                                         |             |
|              |                                                                                                                                                                                                         | Unused      |
| <b>Note</b>  | Reading or writing this register will cause a software error interrupt event. It is the first memory-mapped register, and is designed to trap rogue software before memory-mapped registers are altered |             |

Appendix B Table 23: Software Event 0 to 3 Interrupt Request

### B.5 MCU Interrupt DSP Registers

| DSP2MCU_EVENT_DATA |                                          |  | Description                |
|--------------------|------------------------------------------|--|----------------------------|
| Bit                |                                          |  |                            |
| 23:16              | 15:0                                     |  |                            |
|                    |                                          |  | Event data from DSP to MCU |
|                    |                                          |  | Unused                     |
| <b>Note</b>        | This register is cleared on system reset |  |                            |

Appendix B Table 24: Interrupt Event Data to MCU

| MCU2DSP_EVENT_DATA |                                          |  | Description                |
|--------------------|------------------------------------------|--|----------------------------|
| Bit                |                                          |  |                            |
| 23:16              | 15:0                                     |  |                            |
|                    |                                          |  | Event data from MCU to DSP |
|                    |                                          |  | Unused                     |
| <b>Note</b>        | This register is cleared on system reset |  |                            |

Appendix B Table 25: Interrupt Event Data from MCU

### B.6 PIO Control DSP Registers

Shown below are the PIO registers accessed by the DSP:



| PIO_IN      |                                                                                    |      | Description                                                                                                                                                      |
|-------------|------------------------------------------------------------------------------------|------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bit         |                                                                                    |      |                                                                                                                                                                  |
| 23:20       | 19:16                                                                              | 15:0 |                                                                                                                                                                  |
| [ ]         | [ ]                                                                                | [ ]  | Reads value on PIO 15 to 0<br>Bit 15 = PIO 15<br>↓ ↓<br>Bit 0 = PIO 0                                                                                            |
| [ ]         | [ ]                                                                                | [ ]  | Reads value on PIO 19 to 16 (depending on firmware control of UART and PCM interfaces)<br>Bit 19 = PIO 19<br>↓ ↓<br>Bit 16 = PIO 16                              |
| [ ]         | [ ]                                                                                | [ ]  | Reads value on PIO 23 to 20 (depending on firmware control of UART and PCM interfaces) or AIO 3 to 0 (if available)<br>Bit 23 = PIO 23<br>↓ ↓<br>Bit 20 = PIO 20 |
| <b>Note</b> | A read only register of the PIO input.<br>This register is cleared on system reset |      |                                                                                                                                                                  |

Appendix B Table 26: PIO Input Register



| PIO_OUT     |                                                                                                                                            |      | Description                                                                                                                                                       |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bit         |                                                                                                                                            |      |                                                                                                                                                                   |
| 23:20       | 19:16                                                                                                                                      | 15:0 |                                                                                                                                                                   |
| [ ]         | [ ]                                                                                                                                        | [ ]  | Writes value on PIO 15 to 0<br>Bit 15 = PIO 15<br>↓ ↓<br>Bit 0 = PIO 0                                                                                            |
| [ ]         | [ ]                                                                                                                                        | [ ]  | Writes value on PIO 19 to 16 (depending on firmware control of UART and PCM interfaces)<br>Bit 19 = PIO 19<br>↓ ↓<br>Bit 16 = PIO 16                              |
| [ ]         | [ ]                                                                                                                                        | [ ]  | Writes value on PIO 23 to 20 (depending on firmware control of UART and PCM interfaces) or AIO 3 to 0 (if available)<br>Bit 23 = PIO 23<br>↓ ↓<br>Bit 20 = PIO 20 |
| <b>Note</b> | Writes to the PIO lines. There is a MCU register which controls which bits of the PIO port are controlled by the MCU and which by Kalimba. |      |                                                                                                                                                                   |

Appendix B Table 27: PIO Output Register

| PIO_DIR     |                                                                                                                                                                                                                                                                                              | Description                                                                   |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------|
| Bit         |                                                                                                                                                                                                                                                                                              |                                                                               |
| 23:0        |                                                                                                                                                                                                                                                                                              |                                                                               |
| [ ]         | [ ]                                                                                                                                                                                                                                                                                          | Writes Drive enable to PIO 23 to 0<br>Bit 23 = PIO 23<br>↓ ↓<br>Bit 0 = PIO 0 |
| <b>Note</b> | Controls direction of the PIO lines. There is a MCU register which controls which bits of the PIO port have direction controlled by the MCU and which by Kalimba.<br>Depending on specific part and firmware, not all PIOs will be available, or available as configurable Inputs or outputs |                                                                               |

Appendix B Table 28: PIO Drive enable Register



| PIO_EVENT_EN_MASK |                                                                                                                                                |      | Description                                                                                                                                                                                        |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bit               |                                                                                                                                                |      |                                                                                                                                                                                                    |
| 23:20             | 19:16                                                                                                                                          | 15:0 |                                                                                                                                                                                                    |
| [ ]               | [ ]                                                                                                                                            | [ ]  | 1 = Select event change to cause an interrupt on PIO 15 to 0<br>Bit 15 = PIO 15<br>↓ ↓<br>Bit 0 = PIO 0                                                                                            |
| [ ]               | [ ]                                                                                                                                            | [ ]  | 1 = Select event change to cause an interrupt on PIO 19 to 16 (depending on firmware control of UART and PCM interfaces)<br>Bit 19 = PIO 19<br>↓ ↓<br>Bit 16 = PIO 16                              |
| [ ]               | [ ]                                                                                                                                            | [ ]  | 1 = Select event change to cause an interrupt on PIO 23 to 20 (depending on firmware control of UART and PCM interfaces) or AIO 3 to 0 (if available)<br>Bit 23 = PIO 23<br>↓ ↓<br>Bit 20 = PIO 20 |
| <b>Note</b>       | A bit mask that selects which bits of the PIO port should cause an interrupt if PIO changes state.<br>This register is cleared on system reset |      |                                                                                                                                                                                                    |

Appendix B Table 29: PIO Event Change Interrupt Mask Register



### B.7 MCU Window DSP Registers

The data word size for the MCU and the DSP differ. The MCU has a 16-bit data width and the DSP has 24-bit data width. A register controls how the MCU presents data to the DSP, i.e. whether it is sign extended or not.

| NOSIGNX_MCUWIN[1/2] |                                                                                                                     | Description                                                                                                                     |
|---------------------|---------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| Bit                 |                                                                                                                     |                                                                                                                                 |
| 23:1                | 0                                                                                                                   | 1 = Disable sign extension in MCU Access Window see Figure 5.1<br>0 = Enable sign extension in MCU Access Window see Figure 5.1 |
| Unused              |                                                                                                                     |                                                                                                                                 |
| <b>Note</b>         | Sign extension is from 16-bit MCU value to the 24-bit Kalimba DSP value<br>This register is cleared on system reset |                                                                                                                                 |

Appendix B Table 30: MCU Access Window 0 to 1 Sign Extension Enable Bit

### B.8 Flash Window DSP Registers

| FLASH_WINDOW[1/2/3]_START_ADDR |                                                                                                                     | Description                                        |
|--------------------------------|---------------------------------------------------------------------------------------------------------------------|----------------------------------------------------|
| Bit                            |                                                                                                                     |                                                    |
| 23:0                           |                                                                                                                     | Flash/ROM start address for flash window 1, 2 or 3 |
| Unused                         |                                                                                                                     |                                                    |
| <b>Note</b>                    | Selects the flash address that maps to the start of flash windows 1, 2 or 3. The windows are always of size 4KWords |                                                    |

Appendix B Table 31: Start address register for flash memory mapped into the three DSP flash windows





| NOSIGNX_FLASHWIN[1/2/3] |                                                                                                                                                           | Description                                                                                                                                       |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| Bit                     |                                                                                                                                                           |                                                                                                                                                   |
| 23:1                    | 0                                                                                                                                                         | 1 = Disable sign extension from flash window 1/2/3 value see Figure 5.1<br>0 = Enable sign extension from flash window 1/2/3 value see Figure 5.1 |
|                         |                                                                                                                                                           |                                                                                                                                                   |
| <b>Note</b>             | Sign extension of the 16-bit flash value to the 24-bit Kalimba DSP value for each of the three flash windows.<br>This register is cleared on system reset |                                                                                                                                                   |

Appendix B Table 32: Flash Access Window Sign Extension Enable Bit



### B.9 PM Window DSP Registers

| NOSIGNX_PMWIN |                                                                                                                                                                        | Description                                                                                                                                   |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| Bit           |                                                                                                                                                                        |                                                                                                                                               |
| 23:1          | 0                                                                                                                                                                      | 1 = Disable sign extension from PM Window value in DM1 see Figure 5.1<br>0 = Enable sign extension from PM Window value in DM1 see Figure 5.1 |
| Unused        |                                                                                                                                                                        |                                                                                                                                               |
| <b>Note</b>   | Sign extension of the PM values when it is windowed as two 16-bit values in the two 24-bit banks of DM in the Kalimba DSP.<br>This register is cleared on system reset |                                                                                                                                               |

Appendix B Table 33: PM Access Window Sign Extension Enable Bit

| PM_WIN_ENABLE |                                          | Description                                             |
|---------------|------------------------------------------|---------------------------------------------------------|
| Bit           |                                          |                                                         |
| 23:1          | 0                                        | 1 = Enable PM mapping into DM1 see Figure 5.1<br>Unused |
| Unused        |                                          |                                                         |
| <b>Note</b>   | This register is cleared on system reset |                                                         |

Appendix B Table 34: Flash Access Window Sign Extension Enable Bit

### B.10 MCU IO Map Interface Registers

The DSP Registers which control the access to the MCU's IO map are:



| MCUREGS_ENABLE |                                                                                                                                                                                                   | Description                                                    |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------|
| Bit            |                                                                                                                                                                                                   |                                                                |
| 23:1           |                                                                                                                                                                                                   | 0                                                              |
|                |                                                                                                                                                                                                   | 1 = Enable MCU register interface<br>0 = Disable MCU Interface |
|                |                                                                                                                                                                                                   | Unused                                                         |
| <b>Note</b>    | This register is cleared on system reset<br>The MCU firmware must also enable the DSP's access to the MCU registers. Reading the value of this register returns the value of the MCU's enable bit |                                                                |

Appendix B Table 35: MCU IO register map interface enable

| NOSIGNX_MCUREGS |                                                                                                                      | Description                                                                                                                                            |
|-----------------|----------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bit             |                                                                                                                      |                                                                                                                                                        |
| 23:1            |                                                                                                                      | 0                                                                                                                                                      |
|                 |                                                                                                                      | 1 = 16-bit MCU registers are sign-extended to 24-bits when read by the DSP<br>0 = 16-bit MCU registers are zero-padded to 24-bits when read by the DSP |
|                 |                                                                                                                      | Unused                                                                                                                                                 |
| <b>Note</b>     | This register is cleared on system reset<br>The MCU firmware must also enable the DSP's access to the MCU registers. |                                                                                                                                                        |

Appendix B Table 36: MCU IO register map sign-extension

| MCUREGS_ADDRESS |                                                                                                                      | Description                                                        |
|-----------------|----------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------|
| Bit             |                                                                                                                      |                                                                    |
| 23:16           | 15:0                                                                                                                 |                                                                    |
|                 |                                                                                                                      | The MCU IO map address to access when reading/writing MCUREGS_DATA |
|                 |                                                                                                                      | Unused                                                             |
| <b>Note</b>     | This register is cleared on system reset<br>The MCU firmware must also enable the DSP's access to the MCU registers. |                                                                    |

Appendix B Table 37: MCU IO register map address



| MCUREGS_DATA |                                                                                                                                                                                                                                                                                                                        | Description                                         |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------|
| Bit          |                                                                                                                                                                                                                                                                                                                        |                                                     |
| 23:16        | 15:0                                                                                                                                                                                                                                                                                                                   |                                                     |
|              |                                                                                                                                                                                                                                                                                                                        | The MCU IO map data to write or read data for reads |
|              |                                                                                                                                                                                                                                                                                                                        | Unused                                              |
| <b>Note</b>  | This register is cleared on system reset<br>The MCU firmware must also enable the DSP's access to the MCU registers.<br>Writing to this register starts the write of given data to the address set up in MCUREGS_ADDRESS<br>Reading from this register returns the data from MCU register addressed by MCUREGS_ADDRESS |                                                     |

Appendix B Table 38: MCU IO register map data

### B.11 General DSP Registers

The General Registers are:

| GENERAL_FROM_MCU[0/1/2/3] |                                                                                                                          | Description                         |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------|-------------------------------------|
| Bit                       |                                                                                                                          |                                     |
| 23:16                     | 15:0                                                                                                                     |                                     |
|                           |                                                                                                                          | 16-bit read only registers from MCU |
|                           |                                                                                                                          | Unused                              |
| <b>Note</b>               | 16-bit read only registers with their value coming from a MCU register. See below for optional sign extension to 24-bits |                                     |

Appendix B Table 39: General Register 0 to 4 from MCU to Kalimba

| GENERAL_TO_MCU[0/1/2/3] |                                                                                              | Description                                         |
|-------------------------|----------------------------------------------------------------------------------------------|-----------------------------------------------------|
| Bit                     |                                                                                              |                                                     |
| 23:16                   | 15:0                                                                                         |                                                     |
|                         |                                                                                              | 16-bit registers written by Kalimba and read by MCU |
|                         |                                                                                              | Unused                                              |
| <b>Note</b>             | 16-bit registers written by Kalimba, read by MCU, and seen by a VM function <sup>(1)</sup> . |                                                     |

Appendix B Table 40: General Register 0 to 4 to MCU from Kalimba

**Note:**

<sup>(1)</sup> Details of this VM function are under development



| NOSIGNX_GENREGS |                                                                                                                                 | Description                                                                                                   |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------|
| Bit             |                                                                                                                                 |                                                                                                               |
| 23:1            | 0                                                                                                                               |                                                                                                               |
|                 |                                                                                                                                 | 1 = Disable sign extension from general MCU registers<br>0 = Enable sign extension from general MCU registers |
|                 |                                                                                                                                 | Unused                                                                                                        |
| <b>Note</b>     | Sign extension of the 16-bit general MCU registers to the 24-bit Kalimba DSP value.<br>This register is cleared on system reset |                                                                                                               |

Appendix B Table 41: General MCU Registers Sign Extension Enable Bit

## B.12 PM Flash Window Control Registers

The registers that control the window into program memory flash are:

| PM_FLASH_WINDOW_START_ADDR |                                                                                                                                               | Description                                 |
|----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------|
| Bit                        |                                                                                                                                               |                                             |
| 23:0                       |                                                                                                                                               |                                             |
|                            |                                                                                                                                               | Flash/ROM start address for PM flash window |
| <b>Note</b>                | Selects the flash address that maps to the start of program memory flash at DSP PM address 0x1800<br>This register is cleared on system reset |                                             |

Appendix B Table 42: Start address register for program memory flash window



| PM_FLASH_WINDOW_SIZE |                                                                                                                | Description                                                                                                                                             |
|----------------------|----------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bit                  |                                                                                                                |                                                                                                                                                         |
| 23:16                | 15:0                                                                                                           |                                                                                                                                                         |
|                      |                                                                                                                | 16-bit register that controls the size of program memory flash space available to the DSP in 32-bit DSP words (double for size of flash space required) |
|                      |                                                                                                                | Unused                                                                                                                                                  |
| <b>Note</b>          | Setting to zero disables the program memory flash window and cache<br>This register is cleared on system reset |                                                                                                                                                         |

Appendix B Table 43: Size register for program memory flash window

### B.13 Clock Divide Rate DSP Registers

| CLOCK_DIVIDE_RATE |                                          | Description                                                                                                                                                                                          |
|-------------------|------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bit               |                                          |                                                                                                                                                                                                      |
| 23:4              | 3:0                                      |                                                                                                                                                                                                      |
|                   |                                          | Is 4-bit value, n, that sets divide ratio from the base clock frequency for the DSP.<br>Divide ratio = $2^n$<br>e.g.<br>0 = ÷ 1 <sup>(2)</sup><br>1 = ÷ 2<br>2 = ÷ 4<br>3 = ÷ 8<br>↓<br>15 = ÷ 32768 |
|                   |                                          | Unused                                                                                                                                                                                               |
| <b>Note</b>       | This register is cleared on system reset |                                                                                                                                                                                                      |

Appendix B Table 44: DSP Clock Rate Register for Normal Operation

**Note:**

- (1) Details of this VM function are under development
- (2) Initial release of BlueCore5-Multimedia and Multimedia External is 64MHz or 64MIPs



| INT_CLOCK_DIVIDE_RATE |                                                                                                                                                                           | Description                                                                                                                                                                                            |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bit                   |                                                                                                                                                                           |                                                                                                                                                                                                        |
| 23:4                  | 3:0                                                                                                                                                                       |                                                                                                                                                                                                        |
|                       |                                                                                                                                                                           | Is 4-bit value, n, that sets divide ratio from the base clock frequency for the DSP during interrupt.<br>Divide ratio = $2^n$<br>e.g.<br>0 = ÷ 1<br>1 = ÷ 2<br>2 = ÷ 4<br>3 = ÷ 8<br>↓<br>15 = ÷ 32768 |
|                       |                                                                                                                                                                           | Unused                                                                                                                                                                                                 |
| <b>Note</b>           | For Kalimba to use this clock frequency during interrupt INT_CLOCK_SWITCH_EN register must be enabled see Appendix B Table 12<br>This register is cleared on system reset |                                                                                                                                                                                                        |

Appendix B Table 45: DSP Clock Rate Register for Interrupt Operation

| FORCE_FAST_MMU_CLOCK |                                                                                                                                                              | Description                                                                                                  |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------|
| Bit                  |                                                                                                                                                              |                                                                                                              |
| 23:1                 | 0                                                                                                                                                            |                                                                                                              |
|                      |                                                                                                                                                              | 1 = Force the MCU and MMU clocks to run at maximum (32MHz) speed<br>0 = Release the MCU and MMU clock speeds |
|                      |                                                                                                                                                              | Unused                                                                                                       |
| <b>Note</b>          | This register can be used to accelerate MMU buffer transfers if firmware permits the MCU clock rate to be forced<br>This register is cleared on system reset |                                                                                                              |

Appendix B Table 46: DSP Register for forcing MCU/MMU clocks

## B.14 Core State DSP Registers

The following registers are used for context saving of the DSP core:



| MM_DOLOOP_START |                                                                                     | Description                                                 |
|-----------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------|
| Bit             |                                                                                     |                                                             |
| 23: 16          | 15: 0                                                                               |                                                             |
|                 |                                                                                     | 16-bit register that defines the start address for do loops |
|                 |                                                                                     | Unused                                                      |
| <b>Note</b>     | This register is memory-mapped for context saving, e.g. for interrupt routine entry |                                                             |

Appendix B Table 47: DSP Do Loop start address register

| MM_DOLOOP_END |                                                                                     | Description                                               |
|---------------|-------------------------------------------------------------------------------------|-----------------------------------------------------------|
| Bit           |                                                                                     |                                                           |
| 23: 16        | 15: 0                                                                               |                                                           |
|               |                                                                                     | 16-bit register that defines the end address for do loops |
|               |                                                                                     | Unused                                                    |
| <b>Note</b>   | This register is memory-mapped for context saving, e.g. for interrupt routine entry |                                                           |

Appendix B Table 48: DSP Do Loop end address register

| MM_QUOTIENT |                                                                                     | Description                   |
|-------------|-------------------------------------------------------------------------------------|-------------------------------|
| Bit         |                                                                                     |                               |
| 23:0        |                                                                                     |                               |
|             |                                                                                     | Kalimba core divider quotient |
| <b>Note</b> | This register is memory-mapped for context saving, e.g. for interrupt routine entry |                               |

Appendix B Table 49: DSP Divider Quotient

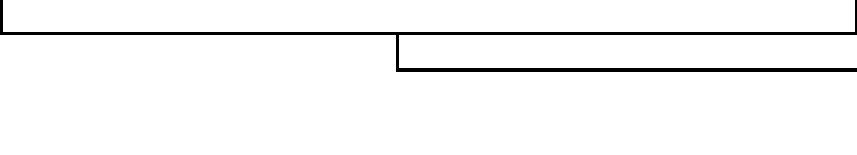
| MM_REM      |                                                                                     | Description                    |
|-------------|-------------------------------------------------------------------------------------|--------------------------------|
| Bit         |                                                                                     |                                |
| 23:0        |                                                                                     |                                |
|             |                                                                                     | Kalimba core divider remainder |
| <b>Note</b> | This register is memory-mapped for context saving, e.g. for interrupt routine entry |                                |

Appendix B Table 50: DSP Divider Remainder



### B.15 Bitreverse Function Memory-mapped Register

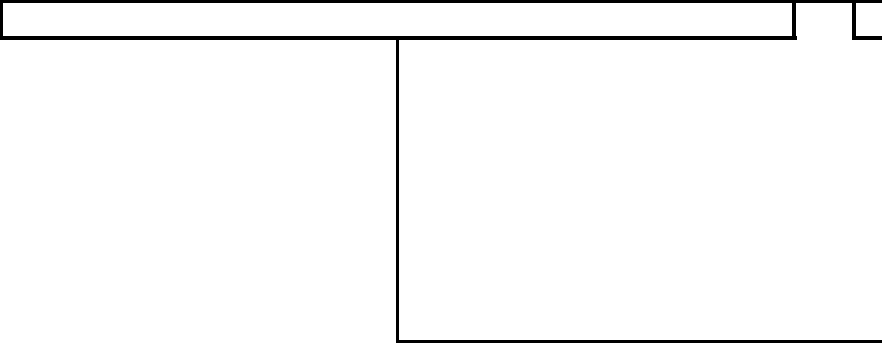
The memory-mapped bitreverse function register is:

| BITREVERSE_DATA                                                                    |                                                           | Description                                                                              |
|------------------------------------------------------------------------------------|-----------------------------------------------------------|------------------------------------------------------------------------------------------|
| Bit                                                                                |                                                           |                                                                                          |
| 23:0                                                                               |                                                           |                                                                                          |
|  |                                                           | Reading this register returns the 24-bit bitreverse of the data previously written to it |
| <b>Note</b>                                                                        | This register should not be used from within an interrupt |                                                                                          |

Appendix B Table 51: DSP Bitreverse data memory-mapped register

### B.16 Kalimba Add/Subtract Core Configuration Register

The memory-mapped register to control saturation in the Kalimba add/subtract core is:

| ADDSUB_SATURATE_ON_OVERFLOW                                                          |                                                                                                                                                                                   |   | Description                                                                                                           |
|--------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|-----------------------------------------------------------------------------------------------------------------------|
| Bit                                                                                  |                                                                                                                                                                                   |   |                                                                                                                       |
| 23:1                                                                                 |                                                                                                                                                                                   | 0 |                                                                                                                       |
|  |                                                                                                                                                                                   |   | 1 = Enable saturation on overflow with DSP add/subtracts<br>0 = Disable saturation on overflow with DSP add/subtracts |
|                                                                                      |                                                                                                                                                                                   |   | Unused                                                                                                                |
| <b>Note</b>                                                                          | Interruptions should be blocked before enabling add/sub saturation, and saturation should be disabled before unblocking interruptions<br>This register is cleared on system reset |   |                                                                                                                       |

Appendix B Table 52: DSP add/subtract core saturation configuration register



### B.17 Stack Memory-mapped Registers

The memory-mapped registers to control the Kalimba's stack instructions are:

| STACK_START_ADDR |                                                                                                                                                            | Description                                                                              |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| Bit              |                                                                                                                                                            |                                                                                          |
| 23:16            | 15:0                                                                                                                                                       |                                                                                          |
| [ ]              | [ ]                                                                                                                                                        | 16-bit register that controls the start address of the stack. This may be in DM1 or DM2. |
| Unused           |                                                                                                                                                            | Unused                                                                                   |
| <b>Note</b>      | Writing to this register sets also STACK_POINTER to STACK_START_ADDR<br>The stack builds upwards. Therefore:<br>$STACK\_END\_ADDR \geq STACK\_START\_ADDR$ |                                                                                          |

Appendix B Table 53: Stack Start address register

| STACK_END_ADDR |                                                                                    | Description                                                                                                                                 |
|----------------|------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| Bit            |                                                                                    |                                                                                                                                             |
| 23:16          | 15:0                                                                               |                                                                                                                                             |
| [ ]            | [ ]                                                                                | 16-bit register that controls the start address of the stack. This may be in DM1 or DM2, but must be in the same bank as the start address. |
| Unused         |                                                                                    | Unused                                                                                                                                      |
| <b>Note</b>    | The stack builds upwards. Therefore:<br>$STACK\_END\_ADDR \geq STACK\_START\_ADDR$ |                                                                                                                                             |

Appendix B Table 54: Stack End address register



| STACK_POINTER |                                                                                                          | Description                                                                                                                                                                                                                  |
|---------------|----------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bit           |                                                                                                          |                                                                                                                                                                                                                              |
| 23:16         | 15:0                                                                                                     |                                                                                                                                                                                                                              |
|               |                                                                                                          | 16-bit register that controls the current value of the stack pointer. This is the address to which the next PUSH instruction will write. The next POP instruction will read address the address one lower than STACK_POINTER |
|               |                                                                                                          | Unused                                                                                                                                                                                                                       |
| <b>Note</b>   | Writing to STACK_START_ADDR sets STACK_POINTER to STACK_START_ADDR too in order to initialize the stack. |                                                                                                                                                                                                                              |

Appendix B Table 55: Stack Pointer memory-mapped register

### B.18 Debug Profiling Memory-mapped Registers

The memory-mapped for debug profiling information are:

| NUM_RUN_CLKS_MS |                                                                                                                                                                                                                                    | Description                                      |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------|
| Bit             |                                                                                                                                                                                                                                    |                                                  |
| 23:8            | 7:0                                                                                                                                                                                                                                |                                                  |
|                 |                                                                                                                                                                                                                                    | 8 MS bits of the 32-bit number of clocks counter |
|                 |                                                                                                                                                                                                                                    | Unused                                           |
| <b>Note</b>     | <p>This register holds the MS 8 bits of the 32-bit number of clocks executed whilst the DSP is running since the counter was reset.</p> <p>The entire 32-bit register is cleared to zero by writing any value to this register</p> |                                                  |

Appendix B Table 56: DSP Number of clocks MS bits register



| NUM_RUN_CLKS_LS |                                                                                                                                                                                                                          | Description                                       |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------|
| Bit             |                                                                                                                                                                                                                          |                                                   |
| 23:0            |                                                                                                                                                                                                                          |                                                   |
|                 |                                                                                                                                                                                                                          | 24 LS bits of the 32-bit number of clocks counter |
| <b>Note</b>     | This register holds the LS 24 bits of the 32-bit number of clocks executed whilst the DSP is running since the counter was reset.<br>The entire 32-bit register is cleared to zero by writing any value to this register |                                                   |

Appendix B Table 57: DSP Number of clocks LS bits register

| NUM_INSTRS_MS |                                                                                                                                                                                                  |  | Description                                            |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--------------------------------------------------------|
| Bit           |                                                                                                                                                                                                  |  |                                                        |
| 23:8          | 7:0                                                                                                                                                                                              |  |                                                        |
|               |                                                                                                                                                                                                  |  | 8 MS bits of the 32-bit number of instructions counter |
|               |                                                                                                                                                                                                  |  | Unused                                                 |
| <b>Note</b>   | This register holds the MS 8 bits of the 32-bit number of instructions encountered since the counter was.<br>The entire 32-bit register is cleared to zero by writing any value to this register |  |                                                        |

Appendix B Table 58: DSP Number of instructions MS bits register

| NUM_INSTRS_LS |                                                                                                                                                                                                         | Description                                             |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------|
| Bit           |                                                                                                                                                                                                         |                                                         |
| 23:0          |                                                                                                                                                                                                         |                                                         |
|               |                                                                                                                                                                                                         | 24 LS bits of the 32-bit number of instructions counter |
| <b>Note</b>   | This register holds the LS 24 bits of the 32-bit number of instructions encountered since the counter was reset.<br>The entire 32-bit register is cleared to zero by writing any value to this register |                                                         |

Appendix B Table 59: DSP Number of instructions LS bits register



| NUM_STALLS_MS |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | Description                                      |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------|
| Bit           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                                                  |
| 23:8          | 7:0                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                  |
|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | 8 MS bits of the 32-bit number of stalls counter |
|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | Unused                                           |
| <b>Note</b>   | <p>This register holds the MS 8 bits of the 32-bit number of stalls encountered since the counter was.</p> <p>This counter only counts stalls due to instruction combinations (e.g. Modifying an index register the instruction before it is used for a read, conditional branches, etc... - see section 4.10 for details), not stalls due to Memory bank conflicts or MMU ports</p> <p>The entire 32-bit register is cleared to zero by writing any value to this register</p> |                                                  |

Appendix B Table 60: DSP Number of stalls MS bits register

| NUM_STALLS_LS |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Description                                             |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------|
| Bit           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                                                         |
| 23:0          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                                                         |
|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | 24 LS bits of the 32-bit number of instructions counter |
| <b>Note</b>   | <p>This register holds the LS 24 bits of the 32-bit number of stalls encountered since the counter was reset.</p> <p>This counter only counts stalls due to instruction combinations (e.g. Modifying an index register the instruction before it is used for a read, conditional branches, etc... - see section 4.10 for details), not stalls due to Memory bank conflicts or MMU ports</p> <p>The entire 32-bit register is cleared to zero by writing any value to this register</p> |                                                         |

Appendix B Table 61: DSP Number of stalls LS bits register



## Appendix C: Software Examples

This section contains samples of example code for the Kalimba DSP.

### C.1 Double-Precision Addition

```
// Double-precision addition: Z = X + Y
//
// Where:
// X = {r0,r1}; - ie. r0 is MSW and r1 is LSW
// Y = {r2,r3};
// Z = {r5,r4};
// Computation time: 2 cycles

r4 = r1 + r3; // add LSWs
r5 = r0 + r2 + Carry; // add MSWs
```

### C.2 Fractional Double-Precision Multiply

```
// Fractional double-precision multiply: Z = X * Y
//
// Where:
// X = {r0,r1}; - ie. r0 is MSW and r1 is LSW
// Y = {r2,r3};
// Z = {r4,r5,r6,r7}; - ie. Z is 96-bit
// Computation time: 12 cycles

rMAC = r1 * r3 (UU); // Compute LSW
r7 = rMAC0; // save Z0
rMAC0 = rMAC1; // shift right rMAC by 24-bits
rMAC12 = rMAC2;
rMAC = rMAC + r0 * r3 (SU); // compute inner products
rMAC = rMAC + r2 * r1 (SU);
r6 = rMAC0; // save Z1
rMAC0 = rMAC1; // shift right rMAC by 24-bits
rMAC12 = rMAC2;
rMAC = rMAC + r0 * r2 (SS); // compute MSWs
r5 = rMAC0; // save Z2
r4 = rMAC1; // save Z3
```

### C.3 Integer Double-Precision Multiply

```
// Integer double-precision multiply: Z = X * Y
//
// Where:
// X = {r0,r1}; - ie. r0 is MSW and r1 is LSW
// Y = {r2,r3};
// Z = {r4,r5,r6,r7}; - ie. Z is 96-bit
// Computation time: 12 cycles

rMAC = r1 * r3 (UU); // Compute LSW
r7 = rMAC LSHIFT 15; // save Z0
rMAC0 = rMAC1; // shift right 24-bits
rMAC12 = rMAC2;
rMAC = rMAC + r0 * r3 (SU); // compute inner products
rMAC = rMAC + r2 * r1 (SU);
r6 = rMAC LSHIFT 15; // save Z1
rMAC0 = rMAC1; // shift right 24-bits
rMAC12 = rMAC2;
rMAC = rMAC + r0 * r2 (SS); // compute MSWs
r5 = rMAC LSHIFT 15; // save Z2
r4 = rMAC LSHIFT -1; // save Z3
```



## C.4 FIR Filter

```
// FIR filter
//
// Input parameters:
// I0 = points to oldest input value in delay line
// L0 = filter length (N)
// I4 = points to beginning of filter coefficient table
// L4 = filter length (N)
// r10 = filter length - 1 (N-1)
// Return values:
// rMAC = sum of products output
// Computation time: N + 2 cycles

fir_filter:
rMAC = 0
r1 = M[I0,1]
r2 = M[I4,1];
do fir_loop;
 rMAC = rMAC + r1 * r2
 r1 = M[I0,1]
 r2 = M[I4,1];
fir_loop:
rMAC = rMAC + r1 * r2;
```



## C.5 Cascaded Bi-Quad IIR Filter

```
// Cascaded biquad IIR filter
//
// Equation of each section:
// $y(n) = (b_0*x(n) + b_1*x(n-1) + b_2*x(n-2) - a_1*y(n-1) - a_2*y(n-2)) \ll \text{scalefactor}$
//
// Input Values:
// r0 = input sample
// I0 = points to oldest input value in delay line
// (no biquads*2 + 2)
// I1 = points to a list of scale factors for each biquad section
// I4 = points to scaled coefficients b2,b1,b0,a2,a1,... etc
// L0 = 2 * num_biquads + 2
// L1 = num_biquads
// L4 = 5 * num_biquads
// M0 = -3
// M1 = 1
// r10 = num_biquads
// Return Values:
// r0 = output sample
// r10 - cleared
// I0,I1,I4,L0,L1,L4,M0,M1 - unaffected
// r1,r2,r3,r4 - affected
//
// Computation time: 8 * num_biquads + 3

biquad_filter:
do biquad_loop;
 r1 = M[I0,1] // get x(n-2)
 r2 = M[I4,1]; // get coef b2
 rMAC = r1 * r2
 r3 = M[I0,1] // get x(n-1)
 r2 = M[I4,1]; // get coef b1
 rMAC = rMAC + r3 * r2
 r4 = M[I1,1] // get scalefactor
 r2 = M[I4,1]; // get coef b0
 rMAC = rMAC + r0 * r2
 r1 = M[I0,1] // get y(n-2)
 r2 = M[I4,1]; // get coef a2
 rMAC = rMAC - r1 * r2
 r1 = M[I0,M0] // get y(n-1)
 r2 = M[I4,M1]; // get coef a1
 rMAC = rMAC - r1 * r2
 M[I0,1] = r3; // store new x(n-2)
 r0 = rMAC ASHIFT r4
 M[I0,M1] = r0; // store new x(n-1)
biquad_loop:
M[I0,1] = r1; // store new y(n-2)
M[I0,1] = r0; // store new y(n-1)
```





## C.6 Radix-2 FFT

```
// An optimised FFT subroutine with a simple interface
//
// Input Values:
// $fft_npts - Number of points (a power of 2)
// $Inputreal - Input array real parts
// $Inputimag - Input array imag parts
// L0,L1,L4,L5 - should be initialised to 0.
//
// Return Values:
// $Refft - Output array real parts
// $Inputreal - Output array imag parts
// All registers altered
//
// Computation time:
//
// No clock cycles = TBA
// where n = log2(fft_npts)
//
// fftnpts: 64 128 256 512
//
// No Clks: TBA

// Declare local variables:
.VAR groups;
.VAR node_space;
#include "twiddle_factors.h"

// -- ENTRY POINT --
fast_fft:

M0 = 0;
M1 = 1;

// -- Process the n-1 stages of butterflies --
r1 = 1;
M[groups] = r1; // groups = 1
r0 = M[$fft_npts];
r0 = r0 ASHIFT -1;
M[node_space] = r0; // node_space = Npts / 2

r0 = SIGNDDET r0;
r1 = 22;
r9 = r1 - r0; // log2(Npts) - 1
stage_loop:

 r10 = M[node_space];
 M2 = r10; // M2 = node_space

 r8 = M[groups];
 r2 = r8 LSHIFT 1;
 M[groups] = r2; // groups = groups * 2;

 I0 = &twid_imag; // I0 -> (-S) of W0
 I2 = &$Inputreal; // I2 -> x0 in 1st group of stage
 I1 = I2 + M2; // I1 -> x1 in 1st group of stage
 I6 = &$Inputimag; // I6 -> y0 in 1st group of stage
 I3 = I6 + M2; // I3 -> y1 in 1st group of stage
 I4 = &twid_real; // I4 -> C of W0
```



```
group_loop:
 r2 = M[I4,1];
 r6 = r2; // r6=C

 r3 = M[I1,0]; // r3=x1

 rMAC = r3 * r6 // rMAC=x1*C
 r5 = M[I3,0]; // r5=y1

 r2 = M[I0,1];
 r7 = r2; // r7=(-S)

 r10 = M[node_space];

 do bfly_loop;
 rMAC = rMAC - r5 * r7 // rMAC=x1*C-y1*-S
 r0 = M[I2,0]; // r0=x0

 r1 = r0 + rMAC // r1=x0'=x0+(x1*C-y1*-S)
 r2 = M[I3,M1]; // r2=y1 (dummy read)

 r1 = r0 - rMAC // r1=x1'=x0-(x1*C-y1*-S)
 M[I2,M1] = r1; // DM=x0'

 rMAC = r3 * r7 // rMAC=x1*(-S)
 M[I1,M1] = r1 // DM=x1'
 r4 = M[I6,M0]; // r4=y0

 rMAC = rMAC + r5 * r6 // rMAC=x1*(-S)+y1*C
 r5 = M[I3,-1]; // r5=next y1

 r1 = r4 + rMAC // r1=y0'=y0+(y1*C+x1*(-S))
 r3 = M[I1,M0]; // r3=next x1

 r4 = r4 - rMAC // r1=y1'=y0-(y1*C+x1*(-S))
 M[I6,M1] = r1; // DM=y0'

 rMAC = r3 * r6 // rMAC=x1*C
 M[I3,M1] = r4; // DM=y1'
 bfly_loop:

 r2 = M[I1,M2]; // move: x1, x0, y1, and y0
 r3 = M[I3,M2]; // onto next group with dummy reads
 r8 = r8 - M1
 r2 = M[I2,M2]
 r3 = M[I6,M2];
 if NZ jump group_loop;

 r10 = M[node_space];
 r10 = r10 ASHIFT -1; // node_space = node_space / 2;
 M[node_space] = r10;

 r9 = r9 - 1;
 if NZ jump stage_loop;

 // -- Process the last stage of butterflies separately --
 I0 = &twid_imag; // I0 -> (-S) of W0
 I5 = &$Inputreal; // I2 -> x0
 I1 = I5 + 1; // I1 -> x1
 M2 = 2;
 I3 = BITREVERSE(&$Refft); // Refft bitreversed

 r0 = M[$fft_npts];
 r0 = SIGNDIT r0;
 r0 = r0 - 7;
 r1 = 1;
 r1 = r1 LSHIFT r0;
 M3 = r1; // Bitreversed modifier

 I4 = &twid_real; // I4 -> C of W0
 I6 = &$Inputimag; // I6 -> y0
 I2 = I6 + 1; // I5 -> y1
```



```
r2 = M[I4,M1]
r5 = M[I2,M2]; // r5=y1
r6 = r2 // r6=C
r3 = M[I1,M2]; // r3=x1

rMAC = r3 * r6 // rMAC=x1*C
r2 = M[I0,1]; // r2=(-S)

r10 = M[$fft_npts];
r10 = r10 ASHIFT -1; // Npts / 2

do last_loop;
 rMAC = rMAC - r5 * r2 // rMAC=x1*C-y1*-S
 r0 = M[I5,M2]; // r0=x0

 r1 = r0 + rMAC; // r1=x0'=x0+(x1*C-y1*-S)

 // enable Bit Reverse addressing on AG1
 rFlags = rFlags OR BR_FLAG;

 r1 = r0 - rMAC // r1=x1'=x0-(x1*C-y1*-S)
 M[I3,M3] = r1; // DM=x0'

 rMAC = r3 * r2 // rMAC=x1*(-S)
 M[I3,M3] = r1 // DM=x1'
 r4 = M[I6,M0]; // r4=y0

 // disable Bit Reverse addressing on AG1
 rFlags = rFlags AND NOT_BR_FLAG;

 rMAC = rMAC + r5 * r6 // rMAC=x1*(-S)+y1*C
 r2 = M[I4,M1] // r2=C;
 r3 = M[I1,M2]; // r3=next x1

 r1 = r4 + rMAC // r1=y0'=y0+(y1*C+x1*(-S))
 r5 = M[I2,M2]; // r5=next y1

 r4 = r4 - rMAC // r1=y1'=y0-(y1*C+x1*(-S))
 M[I6,M1] = r1; // DM=y0'

 r6 = r2 // r6=C
 r2 = M[I0,M1]; // r2=(-S)

 rMAC = r3 * r6 // rMAC=x1*C
 M[I6,M1] = r4; // DM=y1'

last_loop:

I3 = BITREVERSE(&$Inputreal);
I5 = &$Inputimag;

// enable Bit Reverse addressing on AG1
rFlags = rFlags OR BR_FLAG;

r2 = M[I5,1];
r10 = N;
do bit_rev_imag;
 r2 = M[I5,M1]
 M[I3,M3] = r2;
bit_rev_imag:

// disable Bit Reverse addressing on AG1
rFlags = rFlags AND NOT_BR_FLAG;
rts;
```

## Document References

| Document                                                                                                                                      | Reference       |
|-----------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| Bluetooth Advanced Audio Distribution Profile                                                                                                 |                 |
| Information technology -- Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s -- Part 3: Audio | ISO/IEC 11172-3 |
| Information technology -- Generic coding of moving pictures and associated audio information -- Part 3: Audio                                 | ISO/IEC 13818-3 |
| Information technology -- Generic coding of moving pictures and associated audio information -- Part 7: Advanced Audio Coding (AAC)           | ISO/IEC 13818-7 |
| BlueCore5-Multimedia Kalimba DSP Assembler User Guide                                                                                         | bc05-ug-002P    |
| BluLab xIDE User Guide                                                                                                                        | blab-ug-002P    |
| BlueCore5-Multimedia Data Sheet                                                                                                               | bc05-ds-004P    |



## Terms and Definitions

|            |                                                                                          |
|------------|------------------------------------------------------------------------------------------|
| AG         | Address Generator                                                                        |
| ALU        | Arithmetic Logic Unit                                                                    |
| BlueCore™  | Group term for CSR's range of Bluetooth chips                                            |
| Bluetooth® | Wireless technology providing audio and data transfer over short-range radio connections |
| CODEC      | COder DECoder                                                                            |
| CSR        | CSR plc                                                                                  |
| DSP        | Digital Signal Processor                                                                 |
| FFT        | Fast Fourier Transform                                                                   |
| FIR        | Finite Infinite Response filter                                                          |
| IIR        | Infinite Impulse Response filter                                                         |
| ISR        | Interrupt Service Routine                                                                |
| Kalimba    | A CSR DSP core architecture                                                              |
| Kalasm2    | Product name for BlueCore5-Multimedia Kalimba DSP Core Assembler                         |
| LS         | Least Significant                                                                        |
| LSW        | Least Significant Word                                                                   |
| MAC        | Multiply ACcumulate                                                                      |
| MCU        | MicroController Unit                                                                     |
| MS         | Most Significant                                                                         |
| MSW        | Most Significant                                                                         |
| NOP        | No Operation                                                                             |
| PC         | Program Counter                                                                          |
| PIO        | Programmable Input Output                                                                |
| PM         | Program Memory                                                                           |
| RAM        | Random Access Memory                                                                     |
| RTI        | ReTurn from Interrupt                                                                    |
| RTS        | ReTurn from Subroutine                                                                   |
| SPI        | Serial Peripheral Interface                                                              |
| VM         | Virtual Machine                                                                          |



## Document History

| Revision | Date   | Reason for Change:                                                                                              |
|----------|--------|-----------------------------------------------------------------------------------------------------------------|
| a        | OCT 05 | Original draft of this document. (CSR reference: bc05-ug-001Pa)                                                 |
| b        | FEB 05 | Updates throughout                                                                                              |
| c        | JUL 06 | First note (concerns C and V flags) updated in SUBTRACT and SUBTRACT with Borrow. Copyright information updated |

# BlueCore5-Multimedia Kalimba DSP

## User Guide

bc05-ug-001Pc

July 2006

Unless otherwise stated, words and logos marked with <sup>TM</sup> or <sup>®</sup> are trademarks registered or owned by CSR plc or its affiliates. Bluetooth<sup>®</sup> and the Bluetooth logos are trademarks owned by Bluetooth SIG, Inc. and licensed to CSR. Other products, services and names used in this document may have been trademarked by their respective owners.

The publication of this information does not imply that any license is granted under any patent or other rights owned by CSR plc.

CSR reserves the right to make technical changes to its products as part of its development programme.

While every care has been taken to ensure the accuracy of the contents of this document, CSR cannot accept responsibility for any errors.

CSR's products are not authorised for use in life-support or safety-critical applications.