**Freescale Semiconductor**
Application Note

Document Number: AN3076
Rev. 0, 05/2006

# Tuning an Application to Prevent M1 Memory Contention

*by* *David Schuchmann*
*Networking and Communication Systems Group*
*Freescale Semiconductor, Inc.*
*Austin, TX*

For applications requiring full utilization of the DSP's performance, tuning for contention avoidance is critical. Depending on the application, M1 memory access contention can contribute heavily to DSP performance degradation. Tuning an application based on a thorough understanding of the M1 memory organization can reduce or completely eliminate contention-based degradation. In this paper, M1 contention is shown to cause a 54 percent performance degradation using a simple algorithm, that represents a worst-case scenario. After tuning, the algorithm executes with 0 percent degradation.

**Contents**



## 1 Determining Contention as a Problem

To determine whether contention is significantly degrading SC1400 performance, an evaluation can be done by counting the M1 contention occurrence with respect to algorithm execution time. Using the event port and a timer, the M1 contention occurrence can be counted. A second timer can be configured to count clock cycles used as a reference to measure the execution time of the algorithm.



freescale semiconductor

As shown in Figure 1, both timers should be enabled just before the start of algorithm execution. The timers should then be disabled and/or the count values read immediately after the algorithm executes.

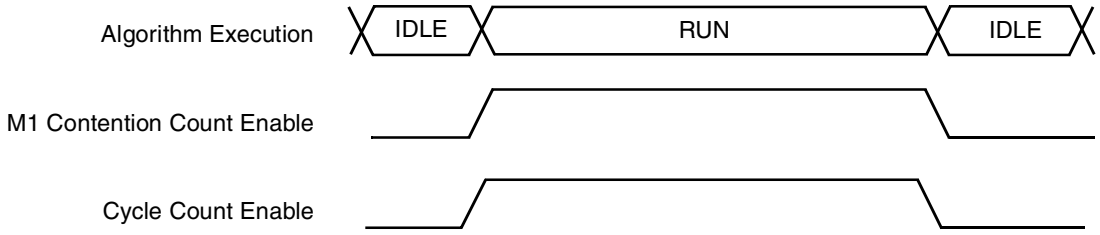


**Figure 1. Measuring Contention Occurrence**

Based on the counter values at the end of the code execution, the performance degradation due to M1 contention can be calculated as follows:

*% Degradation = (M1 Contention Cycles) / (Total Execution Cycles) x 100*

***Eqn. 1***

## 1.1 Event Port Configuration

The event port works closely with the internal timers and debug port (EOnCE block) to allow events on the MSC711x EVNT pins or on-chip events (that is, M1 contention, DMA activity, I-Cache misses, etc.) to interact with the SC1400 core, DMA controller, and other devices. Event port inputs are combined in a program to enable a desired action, such as a timer counting, an interrupt, a DMA transfer, or a halt of the SC1400 core. Figure 2 shows an event multiplexer in the event port configured so that the M1 contention input drives the TIN0 signal, which can be used as a timer count source.

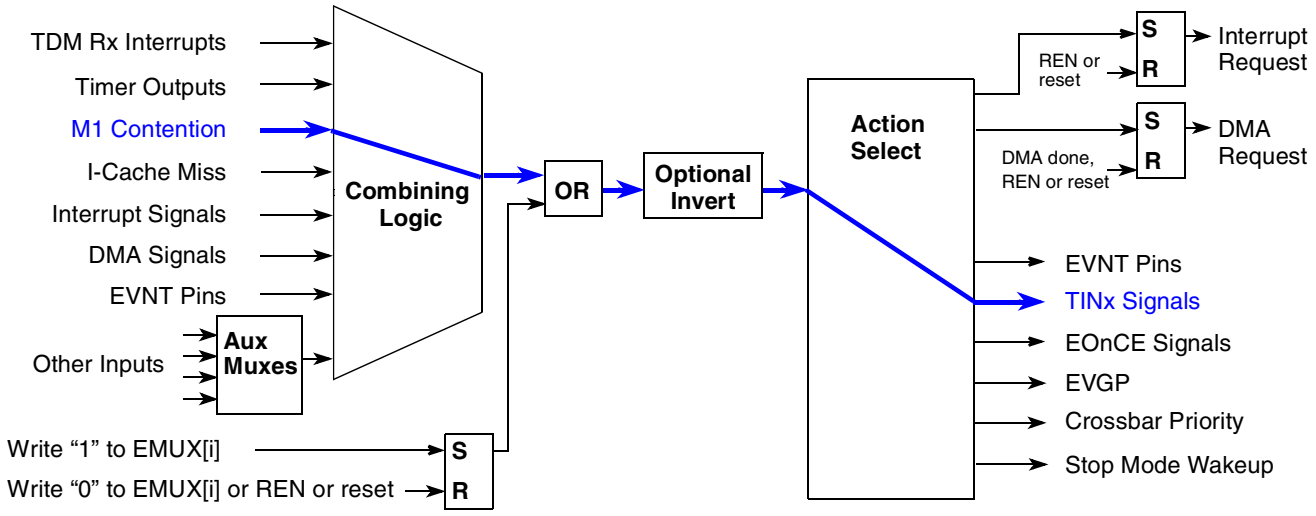


**Figure 2. Single Event Mux**

The contention signal used by the event port and timer asserts once for every two cycles that M1 contention occurs. The following register setting achieves the desired event port configuration:

```
; --- Setup Event Port
      ; Select M1 Contention to drive TIN0 for the Timer
        MOVELIA ($78014E38),(EVOUT0)
```

## 1.2 Timer Configuration

The performance measurement requires the use of two timers. The IP Bus clock should be selected as the timer clock source. When clock cycles and M1 contention cycles are counted, the counter values must be multiplied by 2 to get the values in terms of SC1400 clock cycles. Timer A0 can be used for counting IP Bus clock cycles, and timer A1 can be used to count M1 contentions. The following register settings achieve the desired timer configuration:

```
; --- Setup Timer A0
      ; Select "IP Bus Clk" to drive timer clock
        MOVELIA ($C0000000),(CLKCTRL)
      ; Use Timer for counting AHB Clock Cycles
        MOVEWIA ($0000),(TIMERA_TMR0_CNTR)
      ; Enable Timer A0: IP Bus Clk Counter
        MOVEWIA ($3000),(TIMERA_TMR0_CTRL)
; --- Setup Timer A1
      ; Use Timer for counting M1 Contention
        MOVEWIA ($0000),(TIMERA_TMR1_CNTR)
         ; Enable Timer A1: M1 Contention Counter
        MOVEWIA ($7000),(TIMERA_TMR1_CTRL)
```

This assembly code excerpt uses the MOVELIA and MOVEWIA macros, which are defined as follows:

```
MOVEWIA macro   IMM,DST         ; IMM = immediate value, DST = abs16
        move.w  #?IMM,d6
        move.w  d6,(?DST)
        endm
MOVELIA macro   IMM,DST         ; IMM = immediate value, DST = abs32
        move.l  #?IMM,d6
        move.l  d6,(?DST)
        endm
```

# 2 M1 Memory Organization

The 256 kbyte M1 memory consists of four 64 kbyte groups connected in parallel to the XA, XB, P, and ASM1 buses. Each group consists of two half-groups, each containing four 8 kbyte single-port SRAM modules and a memory wrapper, as shown in Figure 3. Modules 0–3 reside in half-group A, and modules 4–7 reside in half group B. The memory structure allows multiple simultaneous accesses issued from the XA, XB, P, and ASM1 buses to be served without stalls. However, to ensure that stalls are not introduced, the applications must be tuned to avoid memory contentions. The memory contention rules are provided in Section 3, "Contention Rules and Avoidance." Each half-group allows two simultaneous 64-bit accesses or a single 128-bit access to be served.

Each wrapper has two late-write buffers, one serving XA write accesses and one serving XB write accesses. The late-write buffers accept data from XA and XB during the core access when there is a potential contention, and they are flushed to SRAM when there is a free cycle (non-conflicting data read cycle). These late-write buffers reduce contention probability due to their ability to flush to SRAM during these idle cycles.

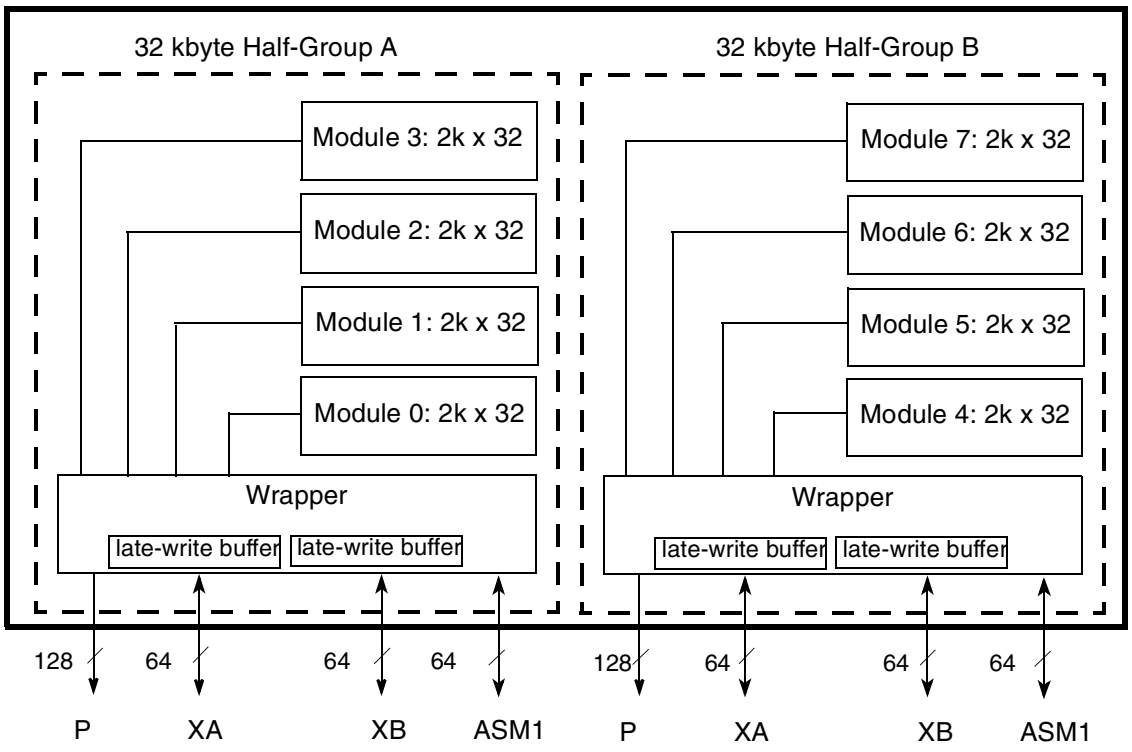


**Figure 3. 64 kbyte Group**

Figure 4 shows the organization of the M1 memory so that there is module interleaving within each memory group, which decreases the probability of access contention.

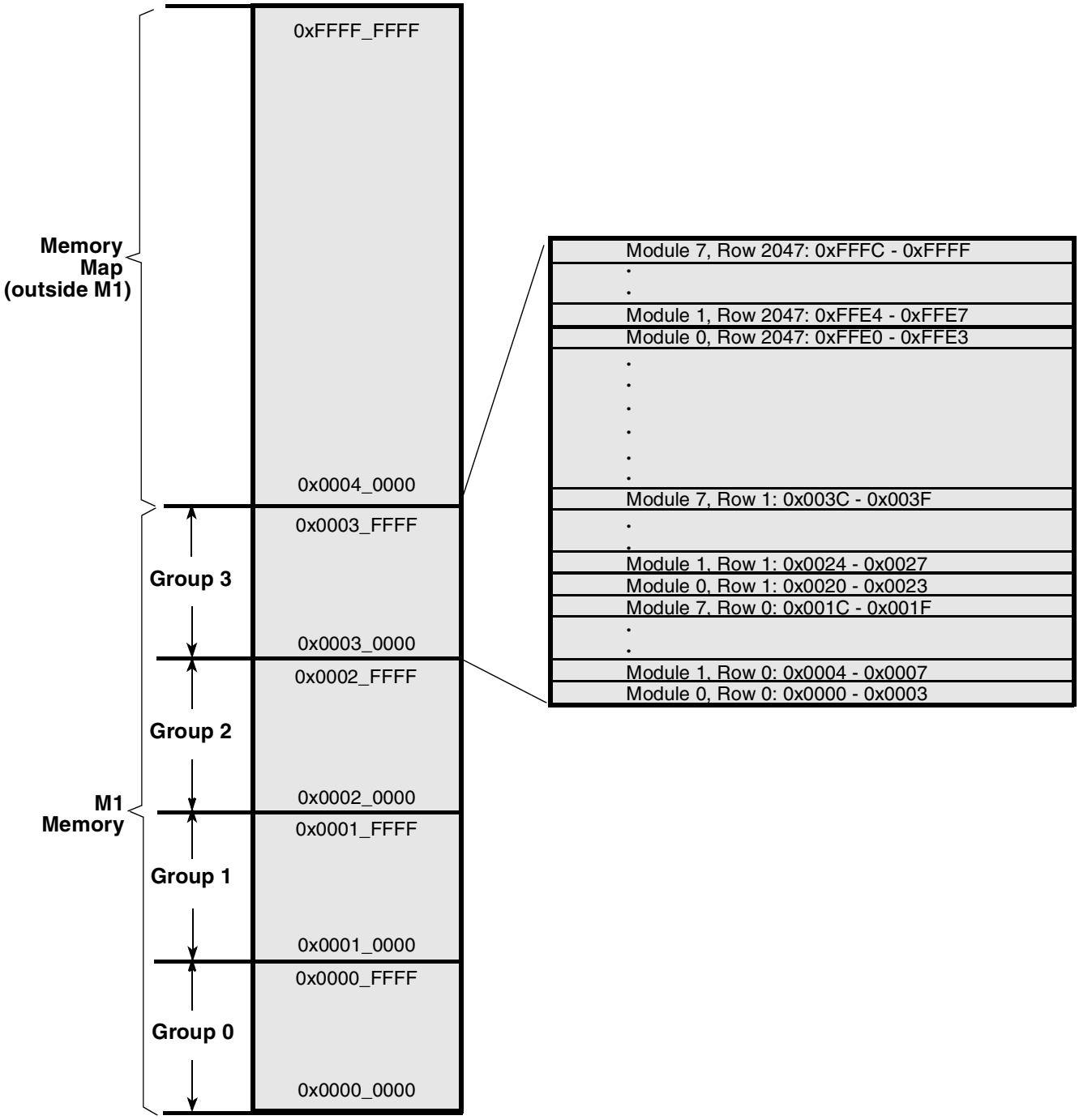


**Figure 4. M1 Memory Organization**

M1 memory uses the SC1400 data (XA, XB) and program (P) addresses to control group, module, and offset selection, as shown in Figure 5.

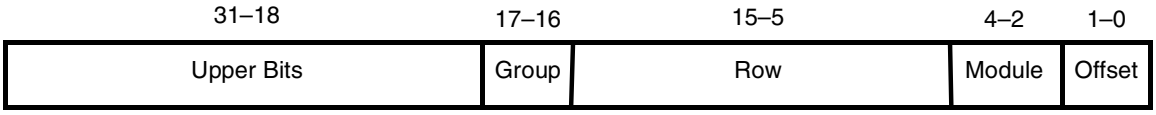


**Figure 5. M1 Memory Address Fields**

# 3 Contention Rules and Avoidance

Due to the M1 memory organization and single-port SRAM modules, memory contention can occur when more than one access targets the same memory module. Because the late-write buffers decrease the contention probability of writes, the contention rules are focused on read accesses.

Contention occurs when any one of the following is true:

- (P read) and [(XA read) or (XB read) or (ASM1)] accesses target the same half-group.
- (ASM1) and [(XA read) or (XB read)] accesses target the same half-group.
- (XA read) and (XB read) accesses target different rows of the same module.

There are similar rules for write accesses but with the following modifications:

1. They are based on the late-write buffer contents for XA and XB write accesses.
2. They apply only for a write to a full late-write buffer, necessitating a flush to SRAM.

In the M1 memory group organization, the addressing interleaves between all 8 modules in the group (across both half-groups), as shown in Figure 4.

The memory contention avoidance rules are as follows:

1. Program code should reside in a memory group not accessed by XA, XB, or ASM1.
2. Data accessed by the DMA or FEC through ASM1 should reside in a group not accessed by XA or XB.
3. Data accessed by XA and XB in parallel ideally should reside in separate groups. If this is not possible, the data should be offset so that XA and XB do not simultaneously access the same module.

# 4 Application Scenario

A simple sum-of-products example algorithm, as follows, can be implemented using parallel SC1400 data accesses to illustrate the impact of M1 contention on DSP performance.

$$Q = X(0)Y(0) + X(1)Y(1) + X(2)Y(2) + ... + X(N)Y(N)$$

***Eqn. 2***

In the example application, buffers X[0:N], Y[0:N], and Z[0:N] are located in M1 memory. Buffers X and Y are accessed by the SC1400 core, and buffer Z is accessed by the DMA controller through ASM1. In this scenario, program code is located in DDR memory and served by the I-Cache; therefore, SC1400 program accesses do not contribute to M1 contention.

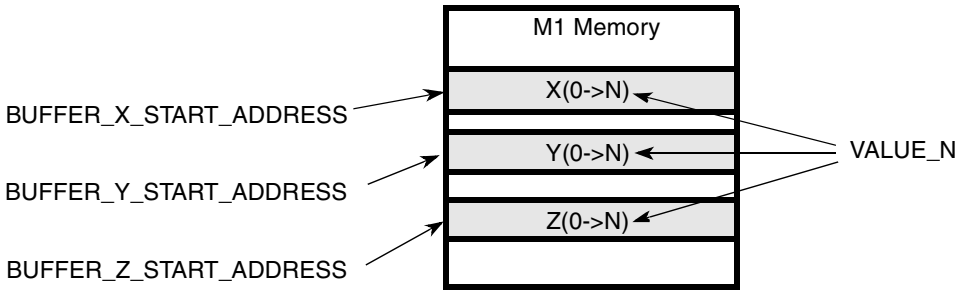


**Figure 6. Buffer Locations**

The example algorithm can be implemented on an MSC711x device through the following assembly code, where the data locations are as shown in Figure 6.

```
; Simple Test Algorithm
    move.l  #$BUFFER_X_START_ADDRESS,r6
    move.l  #$BUFFER_Y_START_ADDRESS,r7
    nop
    ; Parallel XA and XB Data Read Accesses from X(0) and Y(0)
    move.l (r6)+,d2    move.l (r7)+,d3
    move.w  #$VALUE_N,d1
    dosetup1 _start1
    doensh1    d1
    nop
    skipls   _end1
loopstart1
_start1
    ; Calculate X(i)*Y(i) and Accumulate
    ; Parallel XA and XB Data Read Accesses from X(i+1) and Y(i+1)
    mac d2,d3,d4      move.l (r6)+,d2      move.l (r7)+,d3
    loopend1
_end1
```

# 5 Case 1: Non-Optimized Application

The application discussed in this section is not optimized because buffers X, Y, and Z are placed in the same memory group, as shown in Figure 7. Additionally, buffers X and Y are offset by 0x2000, as shown in Figure 8, resulting in the following parallel access in the algorithm to cause XA versus XB contention each time it executes (VALUE_N times).

```
mac d2,d3,d4      move.l (r6)+,d2      move.l (r7)+,d3
```

## 5.1 Configuration

```
BUFFER_X_START_ADDRESS = $00016000
BUFFER_Y_START_ADDRESS = $00018000
BUFFER_Z_START_ADDRESS = $0181A000 (same as $0001A000 when accessed by SC1400)
VALUE_N = $FF
```

Because buffers, X, Y, and Z, are placed in the same group, M1 contention results because:

1. [ASM1 and (XA or XB)] accesses target the same half-group.
2. XA and XB accesses target different rows of the same module.

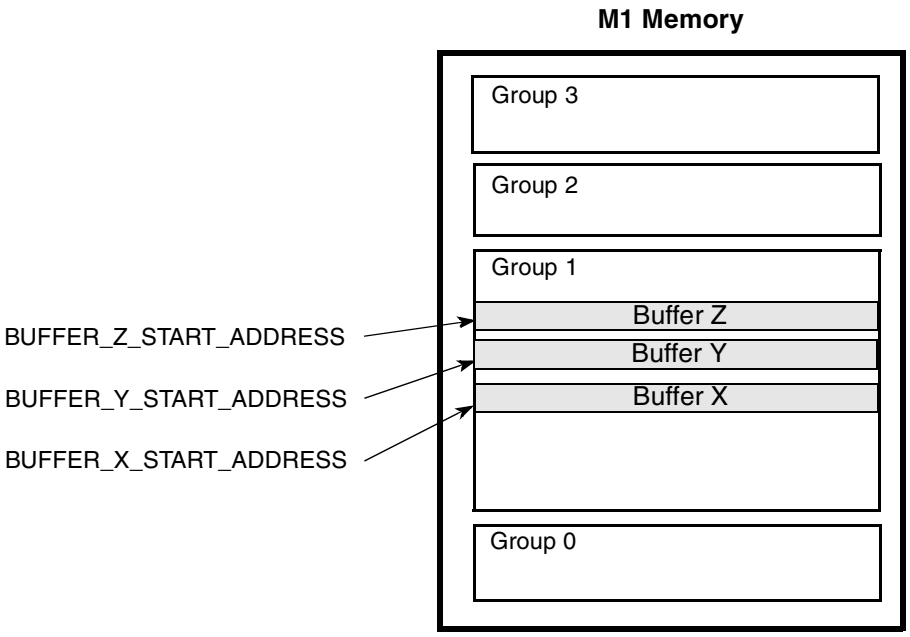


**Figure 7. Case 1 Buffer Placement**

Figure 8 shows the start locations for each buffer in M1. Notice that the start locations for all three buffers are in different rows of module 0. As the algorithm executes, the SC1400 pointers into buffers X and Y increment at the same rate, so the parallel read accesses in the algorithm always target different rows of the same module, causing contention.

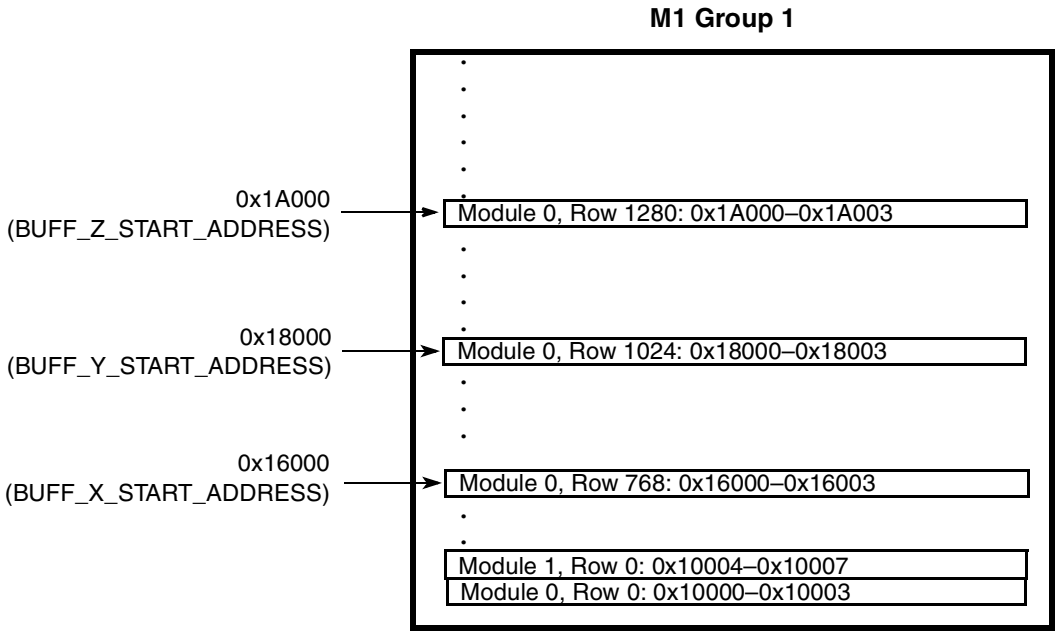


**Figure 8. Case 1 Buffer Offsets**

## 5.2 Results

The algorithm running with performance measurements applied yields the following results:

- Application execution cycles = 287 IP bus cycles = 574 core clock cycles
- M1 contention cycles = 2 × 155 = 310
- Percent degradation = ( 310 / 574 ) × 100 = 54%

Figure 9 and Figure 10 show internal MSC711x signals and buses, including clocks, and SC1400 XA and XB data buses, ASM1 data bus, freeze signals, and counters in timers A0 and A1. The algorithm execution is based on cycle-accurate MSC711x hardware simulation. In Figure 9 the CORE_FREEZE signal, indicating a SC1400 stall, asserts sporadically throughout the algorithm execution (between the markers). The stalls are due to M1 contentions, indicated by assertion of the M1_FREEZE signal. The M1 contention signal used by the event port is similar to the M1_FREEZE signal, but it asserts once for every two M1 contentions, necessitating multiplication by two.

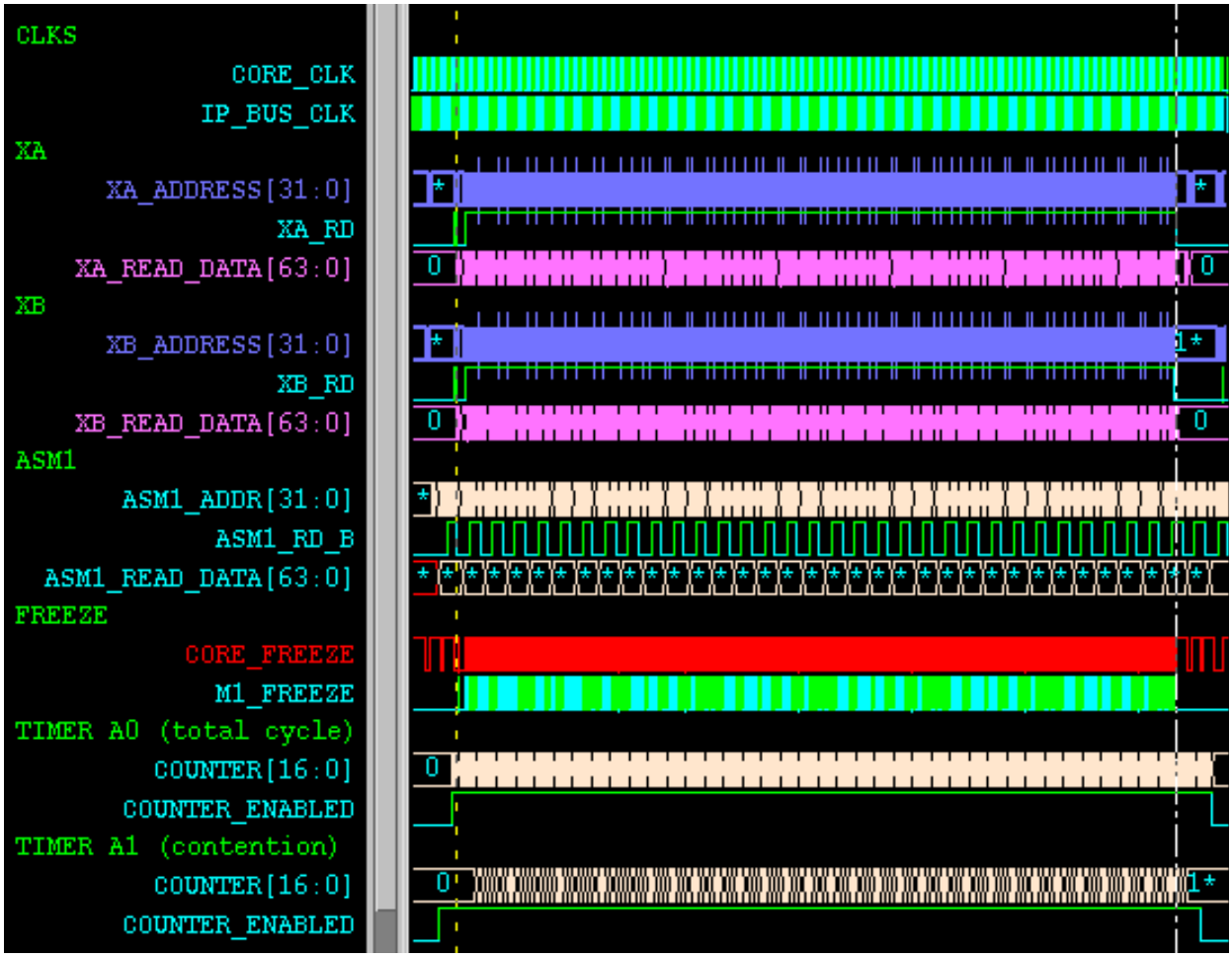


**Figure 9. Case 1 Application Execution**

Figure 10 shows the completion of the algorithm execution at the second marker.

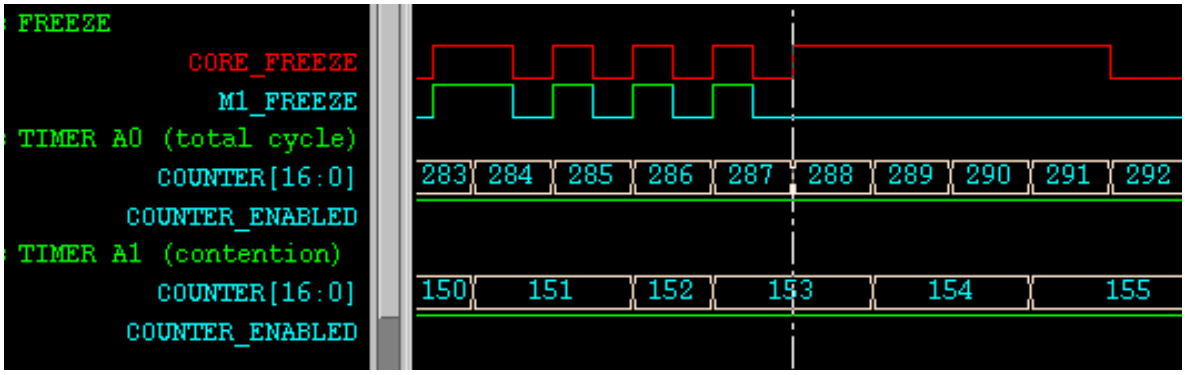


**Figure 10. Case 1 Application Execution Completion**

# 6 Case 2: Partially-Optimized Application

The application discussed in this section is partially optimized because buffer Z is placed into a different memory group than buffers X and Y, as shown in Figure 11. Additionally, buffers X and Y are offset from each other by 0x2000, as shown in Figure 12. As a result, the following parallel access in the algorithm causes XA versus XB contention:

```
mac d2,d3,d4      move.l (r6)+,d2      move.l (r7)+,d3
```

## 6.1 Configuration

```
BUFFER_X_START_ADDRESS = $00016000
BUFFER_Y_START_ADDRESS = $00018000
BUFFER_Z_START_ADDRESS = $01826000 (same as $00026000 when accessed by SC1400)
VALUE_N = $FF
```

Because buffer Z is placed into group 2, M1 contention due to [ASM1 and (XA or XB)] accesses targeting the same half-group is prevented. Since buffers X and Y are still located in the same group, M1 contention results because XA and XB accesses different rows of the same module.

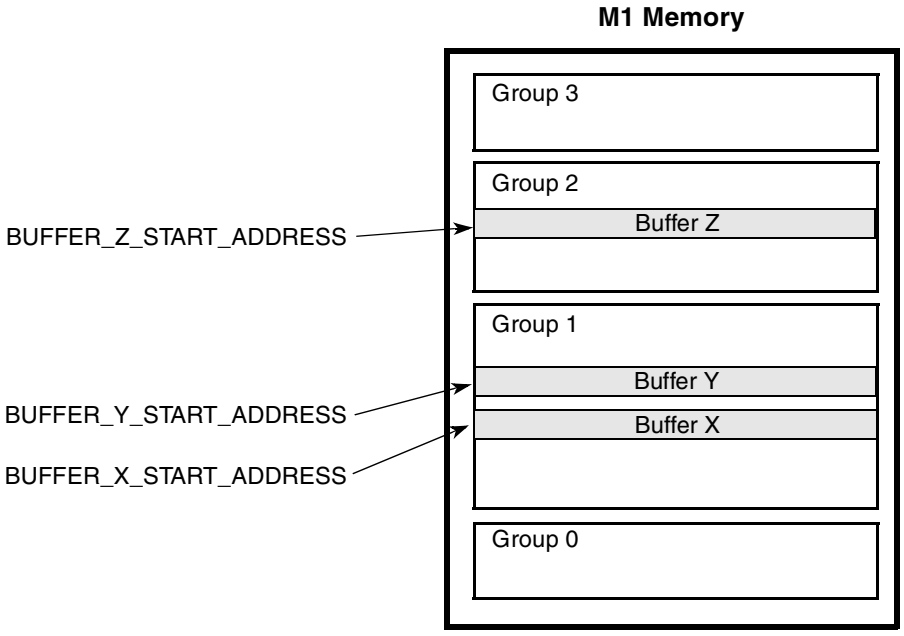


**Figure 11. Case 2 Buffer Placement**

Figure 12 shows the start locations for buffers X and Y in group 1. Note that the start locations are in the same module. As the algorithm executes, the SC1400 pointers into buffers X and Y increment at the same rate so that the parallel read accesses in the algorithm always target the same modules causing contention.

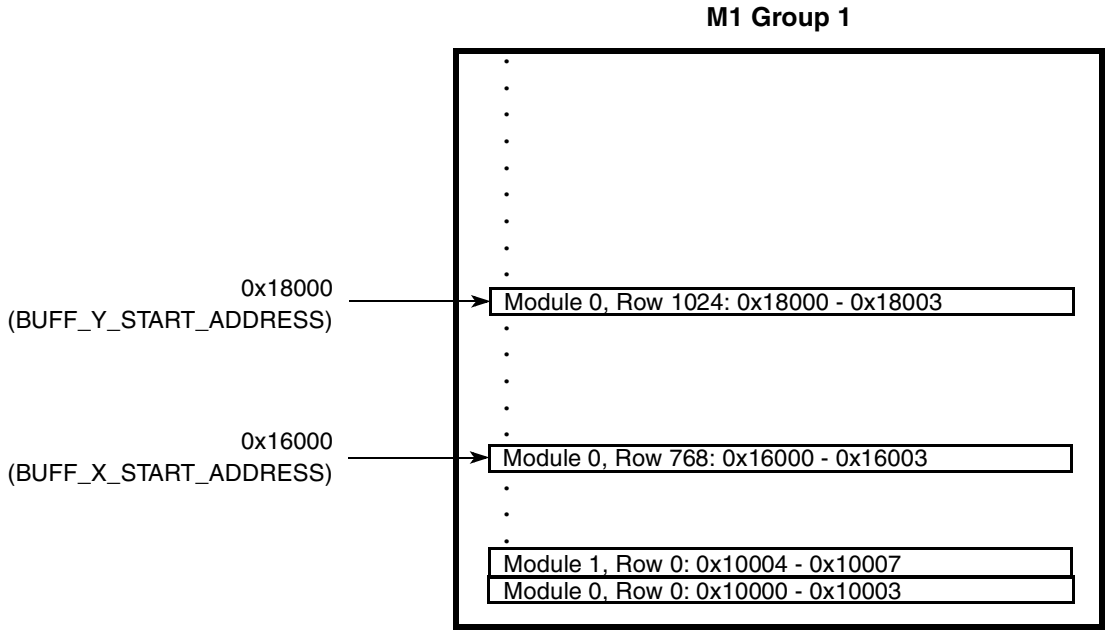


**Figure 12. Case 2 Buffer Offsets**

## 6.2 Results

The algorithm run with performance measurements applied yielded the following results:

- Application execution cycles = 260 IP bus cycles = 520 core clock cycles
- M1 contention cycles = $2 \times 128 = 256$
- Percent degradation = $( 256 / 520 ) \times 100 = 49\%$

Figure 13 and Figure 14 show the algorithm execution on the design based on MSC711x hardware simulation. In Figure 13 the CORE_FREEZE signal, indicating a SC1400 stall, asserts sporadically throughout the algorithm execution (between the markers). The stalls are due to M1 contentions, indicated by assertion of the M1_FREEZE signal.

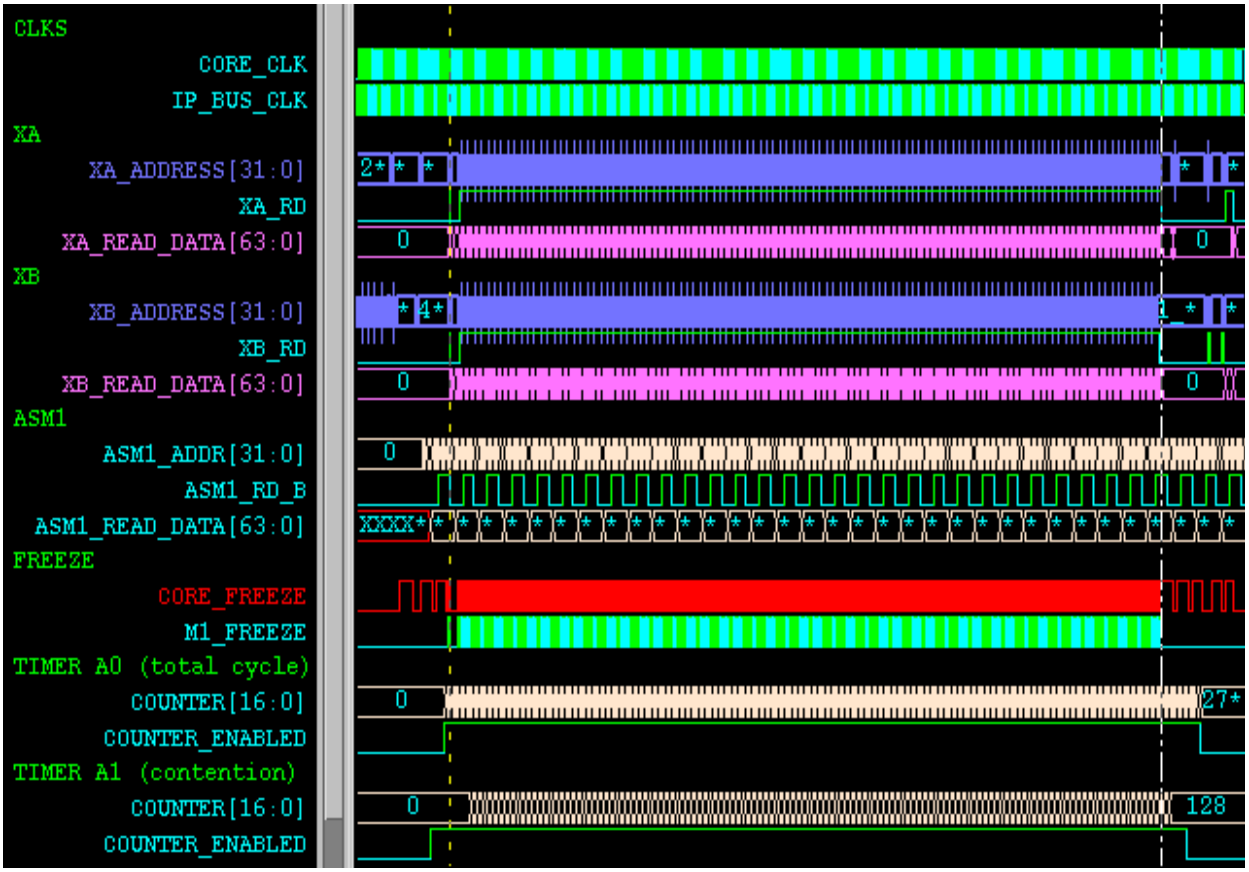


**Figure 13. Case 2 Application Execution**

Figure 14 shows the completion of the algorithm execution at the second marker.

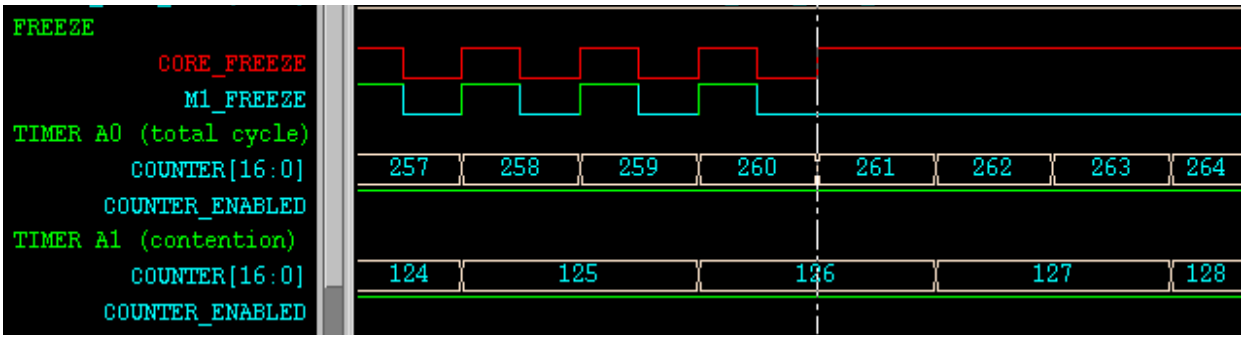


**Figure 14. Case 2 Application Execution Completion**

# 7 Case 3: Optimized Application

The application discussed in this section is optimized because buffer Z is placed in a different memory group from buffers X and Y, as shown in Figure 14. Additionally, buffers X and Y are offset from each other by 0x200C, as shown in Figure 15. As a result, in the following parallel access in the algorithm does not cause XA versus XB contention:

```
mac d2,d3,d4      move.l (r6)+,d2      move.l (r7)+,d3
```

## 7.1 Configuration

```
BUFFER_X_START_ADDRESS = $00016000

BUFFER_Y_START_ADDRESS = $0001800C

BUFFER_Z_START_ADDRESS = $01826000 (same as $00026000 when accessed by SC1400)

VALUE_N = $FF
```

Because buffer Z is placed into group 2, M1 contention due to [ASM1 and (XA or XB)] accesses targeting the same half-group is prevented. As an alternative to the buffer X and Y offset of 0x200C, buffers X and Y can be placed into separate groups, preventing M1 contention. In this example algorithm, it is not necessary to locate buffers X and Y in separate groups to eliminate XA versus XB contention.

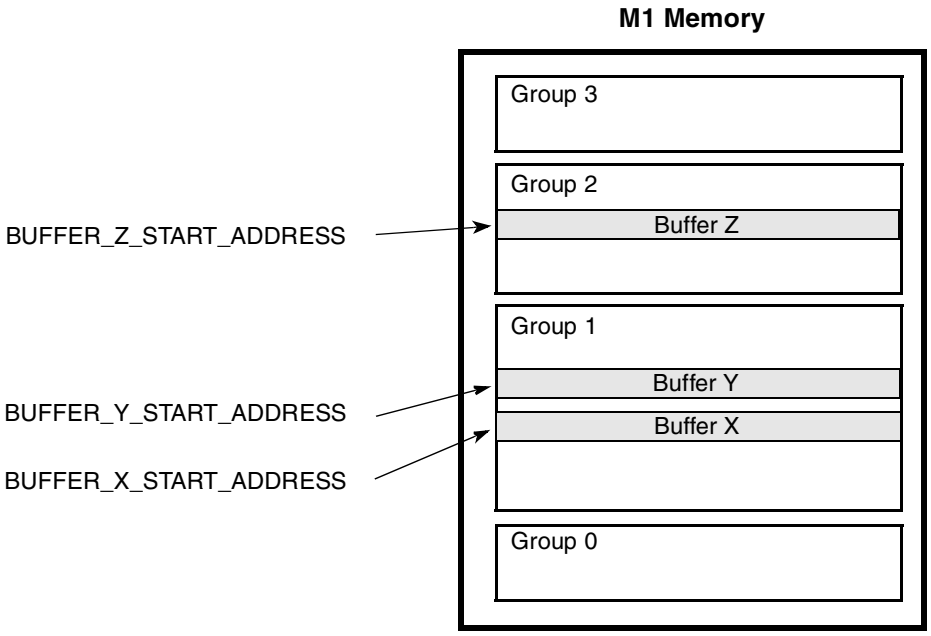


**Figure 15. Case 3 Buffer Placement**

Figure 15 shows the start locations for buffers X and Y in group 1. Note that the start locations are in different modules. As the algorithm executes, the SC1400 pointers into buffers X and Y increment at the same rate so that the parallel read accesses in the algorithm always target different modules, preventing contention.

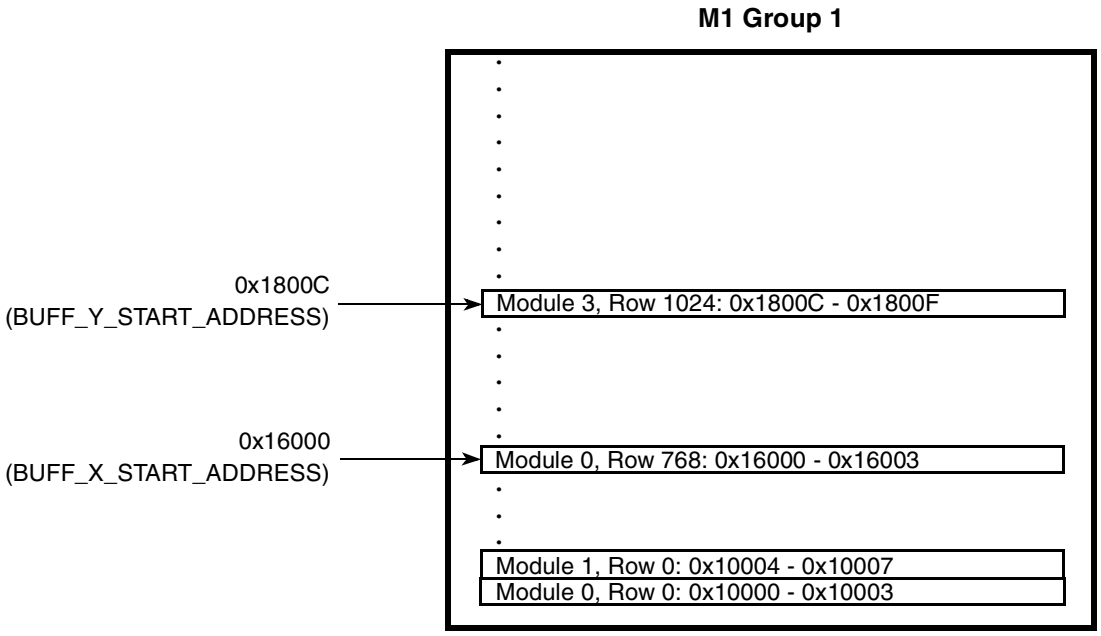


**Figure 16. Case 3 Buffer Offsets**

## 7.2 Results

The algorithm running with performance measurements applied yields the following results:

- Application execution cycles = 132 IP bus cycles = 264 core clock cycles
- M1 contention cycles = 0
- Percent degradation = ( 0 / 264 ) × 100 = 0%

Figure 16 and Figure 17 show the algorithm execution on the design based on a MSC711x hardware simulation. In Figure 16, the CORE_FREEZE signal, indicating a SC1400 stall, does not assert throughout the algorithm execution (between the markers). Also, note that there are no M1 contentions, which are indicated when the M1_FREEZE signal stays deasserted.

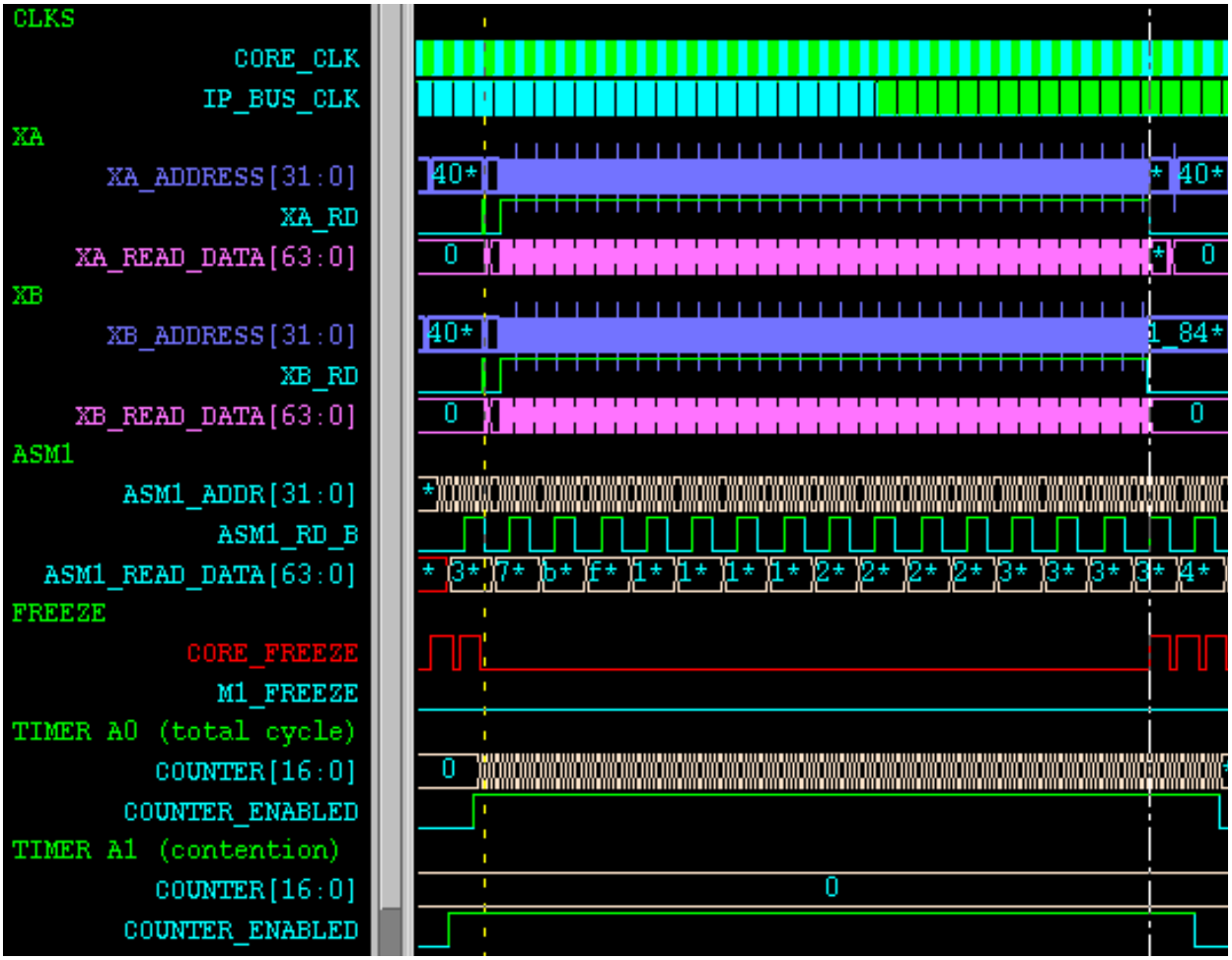


**Figure 17. Case 3 Application Execution**

Figure 18 shows the completion of the algorithm execution at the second marker.

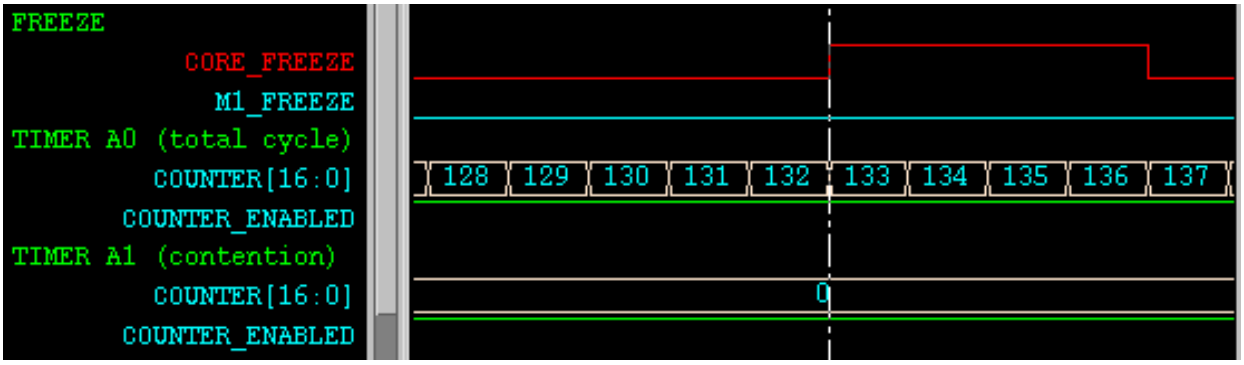


**Figure 18. Case 3 Application Execution Completion**

# 8 Conclusions

The results from the three application cases shown in Table 1 demonstrate the effect of the optimizations. Moving the buffer accessed by the DMA controller (buffer Z) into group 2, which was not accessed by XA or XB, decreased the performance degradation by 5 percent, as shown in case 2 with respect to case 1. By optimizing the buffer placement for buffers operated on by the XA and XB buses (buffers X and Y), performance degradation decreased by an additional 49 percent, as shown in case 3 with respect to case 2. This 49 percent improvement was the result of eliminating M1 contention.

**Table 1. Application Case Results**

| Case | ASM1 vs. [XA,XB] Contention | XA vs. XB Contention | Application Execution Cycles (Core Clk Cycles) | M1 Contentions | % Degradation |
|---|---|---|---|---|---|
| 1: Non-Optimized | Yes | Yes | 574 | 310 | 54 |
| 2: Partially Optimized | No | Yes | 520 | 256 | 49 |
| 3: Optimized | No | No | 264 | 0 | 0 |

In more complicated algorithms, the same percent of improvement may not be achieved by changing the offset of the buffers operated on by XA and XB. For these cases, the buffers should be placed into separate groups to prevent M1 contention. The need for such a change can be determined by measuring the M1 contentions, as described in Section 1, “Determining Contention as a Problem.” Alternatively, degradation due to XA versus XB contention can be estimated by reviewing the SC1400 parallel accesses in the assembly code to determine whether contention results.

In all three cases, program code is located in DDR memory and served by the I-Cache; therefore, SC1400 program accesses do not contribute to M1 contention. If you need to place a program into M1 memory, place it into a group not being accessed by the XA, XB, or ASM1 buses.

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

***How to Reach Us:***

**Home Page:**
www.freescale.com

**email:**
support@freescale.com

**USA/Europe or Locations Not Listed:**
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
1-800-521-6274
480-768-2130
support@freescale.com

**Europe, Middle East, and Africa:**
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

**Japan:**
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064, Japan
0120 191014
+81 3 5437 9125
support.japan@freescale.com

**Asia/Pacific:**
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

**For Literature Requests Only:**
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com



Document Number: AN3076
Rev. 0
05/2006

freescale™
semiconductor