Understanding the TMS320F240 External Memory Interface

APPLICATION REPORT: SPRA452

Jeff Crankshaw
Digital Signal Processor Products
Semiconductor Group

Digital Signal Processing Solutions
June 1998

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current and complete.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

TRADEMARKS

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Contents

Figures

Tables

Understanding the TMS320F240 External Memory Interface

Abstract

This application report describes the features and operation of the Texas Instruments (TI™) TMS320F240 digital signal processing (DSP) external memory interface. First, a simple interface example is provided to show the considerations for connecting SRAMs to the ‘F240. Following this, the basic operation of a single read and a single write access is described to provide an understanding of the signal relationships and characteristics unique to each type of operation. This analysis forms the foundation upon which the interaction of reads and writes are described in all three program spaces: data, program, and I/O spaces. All descriptions include waveforms captured using a digitizing oscilloscope and assembly code sequences used to generate the waveforms. In addition, examples are included for both zero and one software wait state modes of operation.

In addition to the description of the basic modes of operation, a description of the address visibility mode is provided. This mode enables the user to trace the instruction fetch address at the external address pins when the program code resides in the internal ROM of the ‘240.

The program name, code excerpts, test setup, and oscilloscope plots of the signals of interest are provided for each case. Actual assembly programs, assemble/link options, and sample debugger initialization files are provided in the appendices.

Product Support

Related Documentation

The following list specifies product names, part numbers, and literature numbers of corresponding TI documentation.

- ❑ *TMS320C24x DSP Controllers Reference Set Volume 1: CPU System, and Instruction Set*, September 1997, Literature number SPRU160 revision B or later
- ❑ *TMS320C24x DSP Controllers Reference Set Volume 2: Peripheral Library and Specific Devices*, December 1997, Literature number SPRU161 revision B or later
- ❑ *TMS320C240, TMS320F240 DSP Controllers* Data Sheet, December 1997, Literature number SPRS042 revision B or later.

World Wide Web

Our World Wide Web site at **www.ti.com** contains the most up to date product information, revisions, and additions. Users registering with TI&ME can build custom information pages and receive new product updates automatically via email.

Email

For technical issues or clarification on switching products, please send a detailed email to **dsph@ti.com**. Questions receive prompt attention and are usually answered within one business day.

Introduction

Traditionally, the DSP external memory interface is documented in two places: the user’s guide and the data sheet. Both of these document forms provide the user of the DSP with a focused description of the external memory interface. The *user’s guide* provides a general overview of the memory interface features such as pin names, pin descriptions, and address reach. The *data sheet* provides the formal specification of the memory interface and includes important parameters such as operating temperature range, switching characteristics, and timing parameters. Both documents satisfy important needs; however, neither provides all the information a designer may want. This application report will draw information from both sources and expand on several key areas to provide a complete description on the use of the external memory interface.

This report is organized into several sections which cover the basic read and write bus cycles, specific bus cycles resulting from commonly used instructions in all three address spaces, and a description of the address visibility mode and examples on how that may be used for internal program address tracing. The flow of information of the report as a whole and within the individual sections begins with simple concepts and builds on those to cover more advanced topics. The intention is to provide a description suitable for both beginners and experienced users.

As an introduction to the external memory interface, it is first useful to understand how external accesses relate to internal accesses. Table 1 provides a summary of the different types of accesses to program, data, and I/O space, with the external address ranges highlighted in gray. The minimum number of wait states, the minimum number of execution cycles, and the number of delay cycles due to an external read immediately following an external write are included. The minimum number of execution cycle counts are obtained from Chapter 8, “Assembly Language Instructions,” of the *TMS320C24X DSP Controllers Reference Set Volume 1: CPU System Instruction Set Volume 1*.

Table 1. Summary of Read/Write Execution Cycles for ‘240 Address Space

Space	State of $MP/\overline{MC}$	Address Range	Min. # of Wait States	Access Type	Min. # of Execution Cycles	Write-Read Delay Cycles
Program	1	0x0 - 0xffff (ext.)	0	fetch	1+ws	2
				TBLR	3+ws	-
				TBLW	4+ws	-
	0	0x0 - 0x3fff (ROM)	0	fetch	1	-
		0x4000 - 0xffff	0	fetch	1+ws	2
				TBLR	3+ws	-
				TBLW	4+ws	-
I/O	1 or 0	0x0 - 0xffff	0	IN	2+ws	2
				OUT	3+ws	-
Data	1 or 0	0x0 - 0x6fff	0	read	1	-
		(DARAM blocks)		write	1	-
		0x7000 - 0x73ff	2	read	1+2	2
		(peripherals)		write	2+2	-
		0x7400 - 0x7fff	0	read	1	2
		(Event Manager)		write	2	-
		0x8000 - 0xffff	0	read	1+ws	2
		(external)		write	2+ws	-

The Write-Read delay column provides an indication of the ranges where a CPU delay may be incurred if a read operation immediately follows a write operation. This delay is due to a bus conflict between competing read and write bus cycles. The write operation occurs in the Execute stage of the pipeline, while the read operation occurs in the preceding Operand Fetch stage. In the case of a load accumulator instruction from external memory, the bus cycle associated with the external memory read occurs in the Operand Fetch stage. This places the external operand fetch in conflict with the external write of the store accumulator instruction, which happens in the Execute Stage.

The pipeline flow diagram in Table 2 illustrates how this delay occurs for the following write-read instruction sequence:

```
Write    SACL 8000    ; write to external memory at 0x8000
Read     LACL 8001    ; read from external memory at 0x8001
```

Table 2. Pipeline Flow Diagram of Write-Read Delay

	1	2	3	4	5	6	7	8	9	10
Fetch	SACL	LACL								
Decode		SACL	LACL	LACL	LACL	LACL				
Operand			SACL	x	x	dly	LACL			
Execute				SACL	SACL	x	x	LACL		

During cycles 4 and 5, the external bus is used for the write operation, and the operand fetch of the LACL instruction must wait for the write operation to clear the bus. A single delay is inserted in the operand fetch stage for cycle 6 to prevent bus contention between the read and the write. In cycle 7, the LACL operand fetch proceeds and subsequently completes execution in cycle 8.

External Memory Interface Example

In this section, the external memory interface will be described by picking a specific example. The timing diagrams and tables are based on those in the *TMS320C240, TMS320F240 DSP Controllers Data Sheet*. In this case, the external clock frequency of the TMS320F240 is programmed to be 20Mhz (CLKOUT = CPUCLK).

The timing parameters contained in Table 3, through Table 5 are derived from those in the *TMS320C240, TMS320F240 DSP Controllers Data Sheet*, with the exception that all references to the parameter, H, have been resolved.

The timing diagrams in Figure 1 show a write-write-read sequence and a read-read-write sequence in external data space with zero wait states. The timing diagrams in Figure 2 are modified to show the same sequence, except with one software wait state. In both cases, the external pin READY, though not shown, is tied to V_{CC} through a pull-up resistor.

The memory interface design task may be broken into two phases. The first involves narrowing down the search requirement by determining the access time requirement for the memory. The key access time parameter is generally the access time from valid address, and is usually named $t_{a(A)}$. For the TMS320F240 device, this parameter is specified by the following equation:

$$t_{a(A)} = t_{c(CO)} - t_{d(CO\text{-}A)} - t_{su(D\text{-}COL)RD} + N \times t_{c(CO)}$$

where N is the number of wait states.

By substituting the numbers from Table 3 and Table 4, the access times for zero and one wait states can be calculated as follows:

$t_{a(A)}$ = 50ns - 17ns - 15ns + 0ns = 18ns, for zero wait states.

$t_{a(A)}$ = 50ns - 17ns - 15ns + 1× 50ns = 68ns, for one wait state.

These values are also provided in Table 4. With this information, it is possible to select a category of memory devices and move to the second phase, which involves a detailed timing analysis.

Figure 1. Typical Timing Sequence for ‘F240 Connected to SRAM in Data Space: (a) Write-Write-Read and (b) Read-Read-Write; Zero Wait States

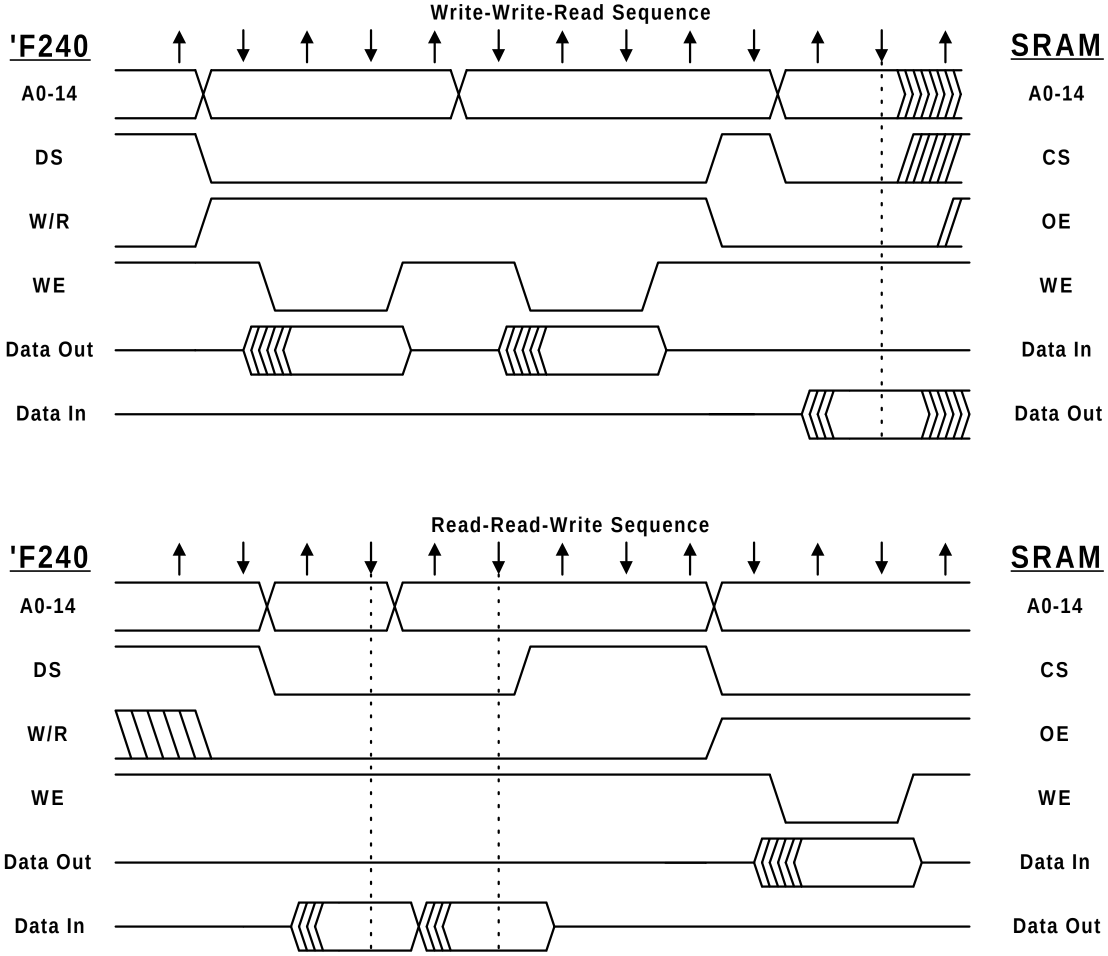


Table 3. Read Switching Characteristics, Adapted from TMS320F240 Data Sheet

Parameter		MIN	MAX	UNIT
$t_{c(CO)}$	Cycle time, CLKOUT	50		ns
H	Cycle time, one half CLKOUT (H=0.5 $t_{c(CO)}$)	25		ns
$t_{d(CO-A)RD}$	Delay time, address valid after CLKOUT/IOPC1 low		17	ns
$t_{d(CO-SL)RD}$	Delay time, STRB low after CLKOUT/IOPC1 low		10	ns
$t_{d(CO-SH)RD}$	Delay time, STRB high after CLKOUT/IOPC1 high		6	ns
$t_{d(CO-ACTL)RD}$	Delay time, PS, DS, IS, and BR low after CLKOUT/IOPC1 low		10	ns
$t_{d(CO-ACTH)RD}$	Delay time, PS, DS, IS, and BR high after CLKOUT/IOPC1 low		10	ns

Table 4. Read Timing Requirements, Adapted from TMS320F240 Data Sheet

Parameter			MIN	MAX	UNIT
$t_{a(A)}$	Access time, read data from address valid†	zero wait states		18	ns
		one wait state		68	
$t_{su(D-COL)RD}$	Setup time, data read before CLKOUT/IOPC1 low		15		ns
$t_{h(D-COL)RD}$	Hold time, data read after CLKOUT/IOPC1 low		2		ns

† Calculated value is based on other characterized data and not tested.

Figure 2. Typical Timing Sequence for 'F240 Connected to SRAM in Data Space: (a) Write-Write-Read and (b) Read-Read-Write; One Wait State

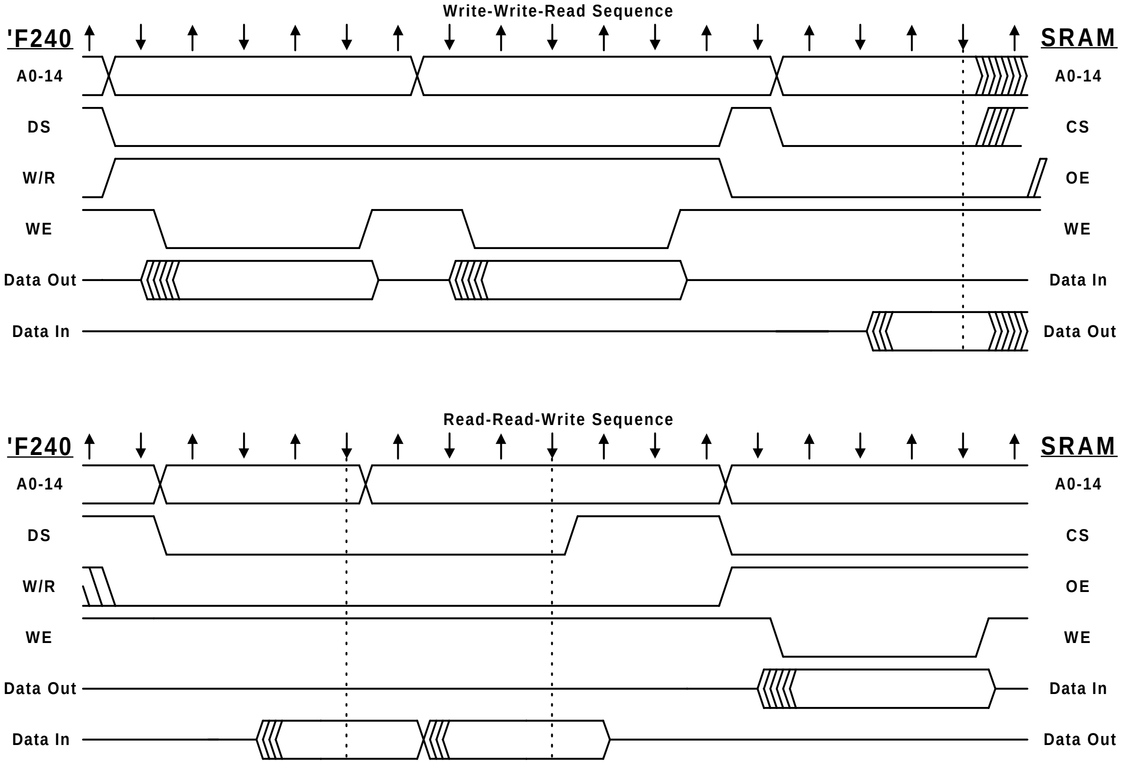


Table 5. Write Switching Characteristics (from TMS320F240 Data Sheet)

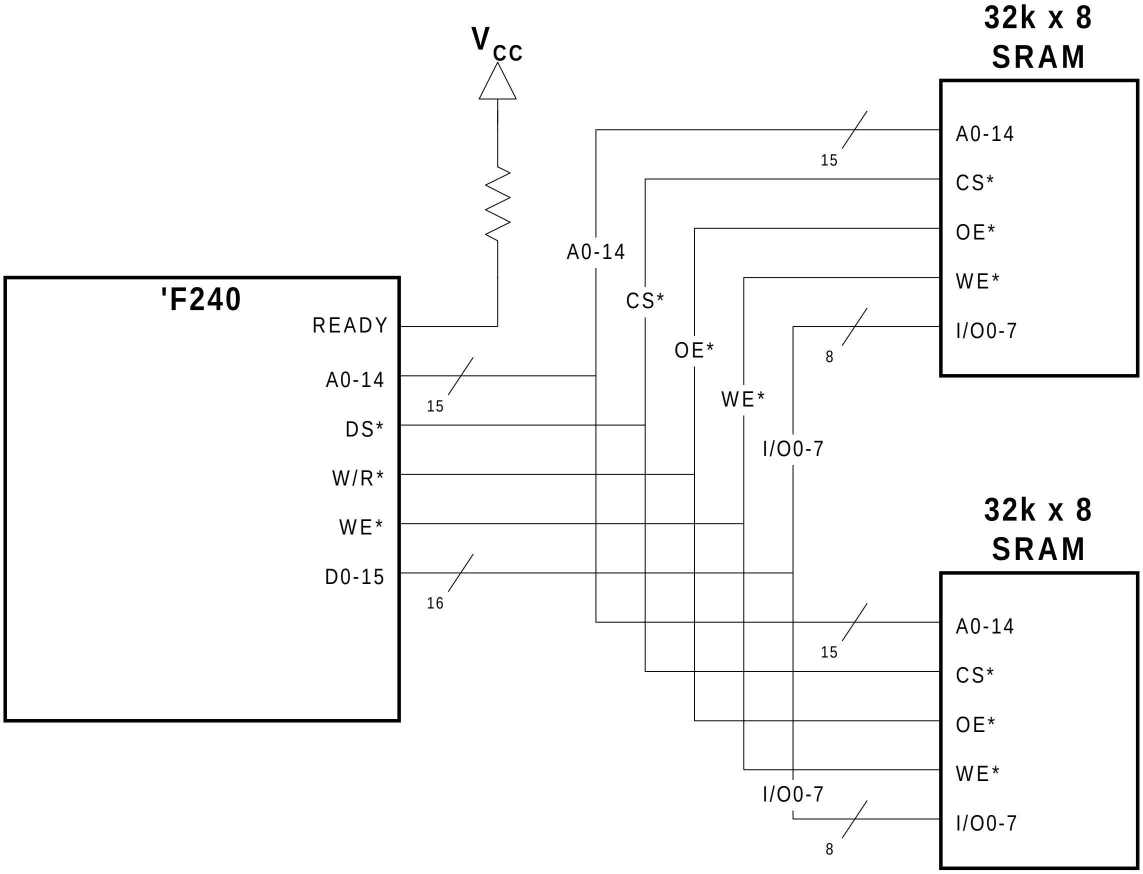
	Parameter	MIN	MAX	UNIT
$t_{d(CO\text{-}A)W}$	Delay time, address valid after CLKOUT/IOPC1 high		17	ns
$t_{d(CO\text{-}D)}$	Delay time, data bus driven after CLKOUT/IOPC1 low		15	ns
$t_{h(A)W}$	Hold time, address valid after WE high†	25		ns
$t_{w(WH)}$	Pulse width, WE high	39		ns
$t_{w(WL)}$	Pulse width, WE low	39		ns
$t_{d(CO\text{-}WL)}$	Delay time, WE low after CLKOUT/IOPC1 low		9	ns
$t_{d(CO\text{-}WH)}$	Delay time, WE high after CLKOUT/IOPC1 low		9	ns
$T_{su(D)W}$	Setup time, write data valid before WE high†	44		ns
$T_{hz(D)W}$	High-impedance time, WE high to data bus Hi-Z‡	0	5	ns
$T_{d(CO\text{-}SL)W}$	Delay time, STRB low after CLKOUT/IOPC1 low		10	ns
$T_{d(CO\text{-}SH)W}$	Delay time, STRB high after CLKOUT/IOPC1 low		6	ns
$T_{d(CO\text{-}ACTL)W}$	Delay time, IS, PS, DS, and BR low after CLKOUT/IOPC1 high		10	ns
$T_{d(CO\text{-}ACTH)W}$	Delay time, IS, PS, DS, and BR high after CLKOUT/IOPC1 high		10	ns
$t_{d(CO\text{-}RWL)W}$	Delay time, R/W low after CLKOUT/IOPC1 high		10	ns
$t_{d(CO\text{-}RWH)W}$	Delay time, R/W high after CLKOUT/IOPC1 high		10	ns

† Calculated value is based on other characterized data and not tested.
‡ Parameter is specified by design and not tested.

TMS320F240 Interfaced to Two 32k x 8-Bit SRAMs

In this section, a detailed timing analysis will be carried out for the ‘F240 interfaced to two generic 32k x 8-bit SRAMs in external data space using zero software wait states. A simplified block diagram of the interface is shown in Figure 3.

Figure 3. Simplified Block Diagram of TMS320F240 Interface to a Pair of 32k x 8 SRAMs



A comparison of typical SRAM switching characteristics versus the comparable ‘F240 values is listed in Table 6 and Table 7. In general, the ‘F240 can meet a typical 15ns SRAM specification. Of course, SRAM specifications vary from manufacturer to manufacturer, so be sure to check with the specification for the SRAM selected.

Table 6. Typical SRAM Read Switching Characteristics

			MIN or MAX	Typ –15 SRAM Value	‘F240 Typ Value	Margin	Unit
1	t_{RC}	Read cycle time	min	15	50	35	ns
2	t_{AA}	Data access time from address valid	max	15	18	3	ns
3	t_{OH}	Output hold time from address change	min	3	0	3	ns
4	t_{AC}	Data access time from chip select low	max	15	25	10	ns
5	t_{OE}	Data access time from output enable low	max	7	50	43	ns
6	t_{LZC}	Output in low z from chip select low	min	3	-	-	ns
7	t_{HZC}	Output in high z from chip select high	max	7	-	-	ns
6	t_{LZO}	Output in low z from output enable low	min	0	-	-	ns
7	t_{HZO}	Output in high z from output enable high	max	7	-	-	ns

Table 7. Typical SRAM Write Switching Characteristics

			MIN or MAX	Typ –15 SRAM Value	‘F240 Typ Value	Margin	Unit
8	t_{WC}	Write cycle time	min	15	100	85	ns
9	t_{AW}	Address valid to end of write	min	10	67	57	ns
10	t_{CW}	Chip select low to end of write	min	10	74	64	ns
11	t_{AS}	Address setup to start of write	min	0	17	17	ns
12	t_{WP}	Pulse width, write enable low	min	9	39	39	ns
13	t_{AH}	Address hold time from write end	min	0	25	25	ns
14	t_{DW}	Data setup time to end of write	min	7	44	33	ns
15	t_{DH}	Data hold time from end of write	min	0	0	0	ns
16	t_{WLZ}	Output in low z from write enable high	min	4	-	-	ns
16	t_{HZW}	Output in high z from write enable low	max	6	-	-	ns

The ‘F240 typical values shown in the preceding two tables were determined using the following analysis:

1) t_{RC} The minimum read cycle time on the ‘F240 is one CPU clock cycle, which is given by the spec value $t_{c(CO)}$ as 50ns.

2) t_{AA} The maximum time from address valid before data is latched by the ‘F240 is calculated by the following equation:

$t_{c(CO)}$ - $t_{d(CO-A)RD}$ -$t_{su(D-COL)RD}$ = 50ns - 17ns - 15ns = 18ns

3) t_{OH} For the ‘F240, the minimum time data must remain valid after the address changes, $t_{h(D-COL)RD}$, gives a minimum hold time after clock out low. Since the address switches after clock out low, the soonest the address will change is 0ns.

4) t_{AC} The maximum time from data strobe low before data is latched by the ‘F240 is given by the following equation:

   $t_{c(CO)}$ -$t_{d(CO\text{-}ACTL)RD}$ - $t_{su(D\text{-}COL)RD}$ = 50ns - 10ns -15ns = 25ns.

   The memory timing number must be less than this value.

5) t_{OE} The maximum time from write/read low before data is latched into the ‘F240 is given by the following equation:

   1.5$t_{c(CO)}$ - $t_{d(CO\text{-}RWL)W}$ - $t_{su(D\text{-}COL)RD}$ = 75ns - 10ns -15ns = 50ns.

   The memory timing number must be less than this value.

6) t_{LZC},t_{LZO} The times from chip select and/or output enable to low-z of the data pins are used to determine the length of time the data bus will be in high impedance state between the end of a write cycle and the start of a read cycle. These timings determine when the memory will begin driving outputs to the ‘F240 data pins. The ‘F240 parameter, $t_{hz(D)W}$, determines when the ‘F240 ceases to drive data out. The bus hi-z time is determined by the difference between these two events. For a memory interface where $\overline{DS}$ is connected to the memory chip select input and $R/\overline{W}$ is connected to the memory output enable input, the following equations can be used to determine the minimum hi-z time between a write and a read:

   $t_{hz(W\text{-}RD)}$ = min($t_{d(WH\text{-}ACTL)}$ + t_{LZC} - $t_{hz(D)W}$, $t_{d(WH\text{-}RWL)}$ + t_{LZO} - $t_{hz(D)W}$)

   where,

   $t_{d(WH\text{-}ACTL)}$ = $t_{c(CO)}$ + $t_{d(CO\text{-}ACTL)RD}$ - $t_{d(CO\text{-}WH)}$ = 50ns + 10ns - 9ns = 51ns

   and

   $t_{d(WH\text{-}RWL)}$ = 0.5$t_{c(CO)}$ + $t_{d(CO\text{-}RWL)}$ - $t_{d(CO\text{-}WH)}$ = 25ns + 10ns - 9ns = 24ns.

   $\therefore$ $t_{hz(W\text{-}RD)}$ = min(51 + 3 - 5, 24 + 0 - 5) = min (49ns, 19ns) = 19ns.

7) t_{HZC},t_{HZO} The times from chip select and/or output enable to high-z of the data pins are used to determine the length of time the data bus will be in high impedance state between the end of a read cycle and the start of a write cycle. These timings determine when the memory will stop driving outputs to the ‘F240 data pins. The ‘F240 parameter, $t_{d(CO\text{-}D)}$, provides an indication of when the ‘F240 begins to drive data out. Since data out is delayed from the falling edge of clock out, the time to low-z can be determined no sooner than that, or 0ns from clock out low. The bus hi-z time is determined by the difference between the maximum hi-z time of the memory and the minimum low-z time of the ‘F240. For a memory interface where $\overline{DS}$ is connected to the memory chip select input and $R/\overline{W}$ is connected to the memory output enable input, the following equations can be used:

For $\overline{CS} = \overline{DS}$,

$t_{hz(RD\text{-}W)} = 2t_{c(CO)} - t_{HZC} - t_{d(CO\text{-}ACTH)W}$ = 100ns -7ns -10ns = 83ns.

8) t_{WC} The minimum write cycle time for the ‘F240 is two CPU clock cycles, which is given by the spec value $2t_{c(CO)}$, and is 100ns.

9) t_{AW} The minimum time from address valid either $\overline{WE}$ or $\overline{DS}$ high can be calculated by the following equations:

For $\overline{WE}$ -controlled writes,

$1.5t_{c(CO)} - t_{d(CO\text{-}A)W} + t_{d(CO\text{-}WH)}$ = 75ns - 17ns + 9ns = 67ns

For $\overline{DS}$ -controlled writes,

$2t_{c(CO)} - t_{d(CO\text{-}A)W} + t_{d(CO\text{-}ACTH)W}$ = 100ns - 17ns + 10ns = 93ns

$\therefore$ t_{AW} = min(67ns, 93ns) = 67ns.

10) t_{CW} For the ‘F240, the minimum time from data strobe low to write enable low is calculated by the following equation:

$1.5t_{c(CO)} - t_{d(CO\text{-}ACTL)W} + t_{d(CO\text{-}WH)}$ = 75ns - 10ns + 9ns = 74ns

11) t_{AS} The minimum time from address valid to write enable low is calculated by the following equation:

$0.5t_{c(CO)} + t_{d(CO\text{-}WL)} - t_{d(CO\text{-}A)W}$ = 25ns + 9ns - 17ns = 17ns.

The memory timing number must be less than this value.

12) t_{WP} The ‘F240 parameter, $t_{w(WL)}$, provides this value directly, and is 2H-11, or 39ns.

The memory timing number must be less than this value.

13) t_{AH} The ‘F240 parameter, $t_{h(A)W}$, provides this value directly, and is H, or 25ns.

The memory timing number must be less than this value.

14) t_{DW} The ‘F240 parameter, $t_{su(D)W}$, provides this value directly, and is 2H-6, or 44ns.

The memory timing number must be less than this value.

15) t_{DH} The ‘F240 parameter, $t_{hz(D)W}$, provides this value directly, and is 0ns.

The memory timing number must be less than or equal to this value.

16) t_{WLZ},t_{HZW} These memory parameters are for interfaces where $\overline{OE}$ is tied low, and are not applicable to this interface example.

External Memory Interface Operation

This section discusses the range of external memory interface operations beginning with the definition of an external bus cycle and its attributes. This is followed by a description of the basic read and write bus cycles. Specific examples are then provided which show the read and write bus cycles in context with instructions in each address space, such as table reads/writes in program space and in/out operations in I/O space. For each example, the code sequences used to create each access is presented and described. All accesses are shown for both zero and one wait state. Each description is supported by oscilloscope waveforms, which illustrate the relationship between the key signals.

External Bus Cycle Definition

External memory accesses are defined in terms of bus cycles, where a bus cycle is the period of time between consecutive falling edges of the CPU clock as observed on the $CLKOUT$ pin. By this definition, one bus cycle equals one CPU cycle. A bus cycle is further defined as either active or inactive. An active bus cycle generally occurs when one of the address strobes, $\overline{DS}$, $\overline{PS}$, or $\overline{IS}$, is in its active low state. By definition then, both read and write bus cycles begin with the falling edge of $CLKOUT$. For read bus cycles, or hereafter just reads, all address, data and control signals also switch or are latched by the falling edge of $CLKOUT$. In contrast, for writes, only data, $\overline{STRB}$, and $\overline{WE}$ switch with respect to the falling edge of $CLKOUT$. The address and remaining control signals switch with respect to the rising edge of $CLKOUT$. This relationship between transition of the clock output and the external memory interface signals is summarized in Table 8.

Table 8. Summary of External Memory Interface Signal Transitions

Bus Cycle	Clock Out Transition	Signal Name
Read	Falling edge	$A0-15$, $D0-15$, $\overline{STRB}$, $\overline{DS}$, $\overline{PS}$, and $\overline{IS}$.
Write	Falling edge	$D0-15$, $\overline{STRB}$, and $\overline{WE}$.
	Rising edge	$A0-15$, $\overline{DS}$, $\overline{PS}$, $\overline{IS}$, $W/\overline{R}$ and $R/\overline{W}$

The instruction set of the ‘C2xx CPU includes many instructions which will generate external read or write bus cycles. Read bus cycles can be generated in data space by load accumulator, and bit test instructions, or in program space by instruction fetches or table read instructions, or in I/O space by the in instruction. Similarly, write bus cycles can be generated by many different instructions and in all three address spaces. The simplest way to generate reads and writes is to use the load/store accumulator instructions to access external data space. The basic read and write bus cycle will be described using these two instructions as the metric.

Read Cycles

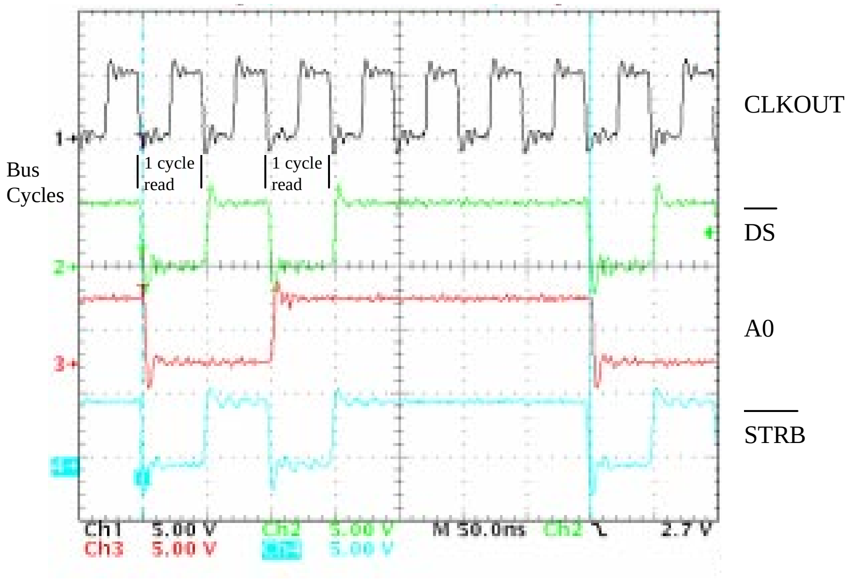
The simple code sequence in Example 1 is written to generate two distinct read bus cycles on the external memory interface. This code provides a tight loop for synchronization of the oscilloscope and capture of the key signal transitions that define the read bus cycle.

Example 1. External Read to Data Space Code Sequence - xidstime.asm

```
R_TIMING:
      LDP #08000h/80h      ; use Data Page of first location in
                              ; external data space

RLOOP
      LACL   0         ; read from external memory 0x8000
      NOP
      LACL   1         ; read from external memory 0x8001
      B      RLOOP
```

The resulting signal trace, shown in Figure 4, captures a one-cycle read, one inactive cycle (NOP), another one-cycle read, four inactive cycles (branch) and then repeats the first one-cycle read. Every read cycle begins when $\overline{DS}$ ($\overline{PS}$ or $\overline{IS}$) and $\overline{STRB}$ transition to their active low states and addresses becoming valid with the falling edge of $CLKOUT$. The read cycle ends when $\overline{DS}$ ($\overline{PS}$ or $\overline{IS}$) and $\overline{STRB}$ switch back to their inactive high states, again coincident with the falling edge of $CLKOUT$. The incoming data is also latched by this last falling edge of $CLKOUT$. Note that the address pins retain the last valid address until the next valid bus cycle, which in this case is the second read. All signals transition based on the falling edge of $CLKOUT$, as previously defined in Table 8.

Figure 4. Trace of External Data Space Reads with Zero Wait States



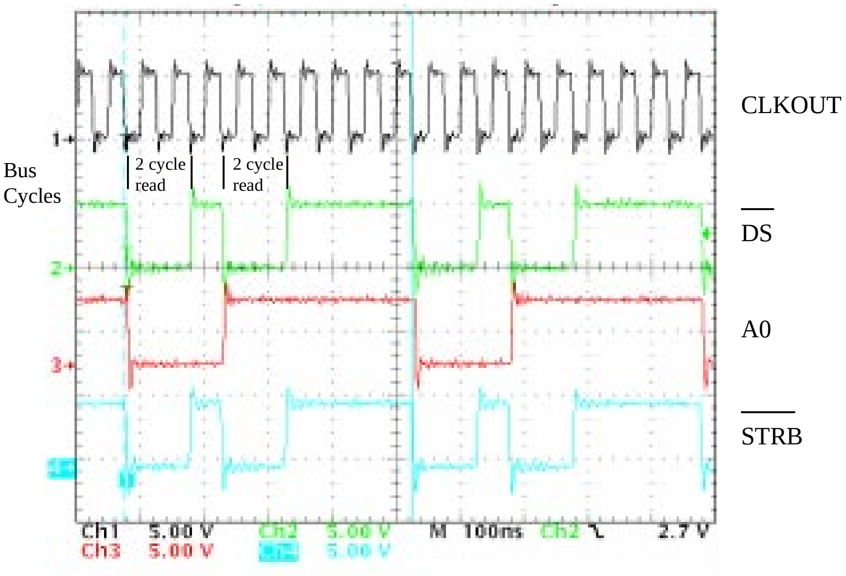
The one wait state read bus cycle is very similar to the zero wait state case just discussed. The simple code change shown in Example 2 will reconfigure the software wait state generator for one wait state in data space.

Example 2. Code Example for One Software Wait State in Data Space

```
* Load Address of wait-state generator register, WSGR, into AR1,
* and turn off address visibility mode, AVIS.
     LDP  #0             ; set DP to page zero, includes B2 RAM
     SPLK #02h,070h      ; 70h <= 0002h
     OUT  70h,0ffffh     ; WSGR <= (07h). AVIS off; 1 ws for DS, 0
                           ; ws for IS & PS
```

The result of this change can be seen in Figure 5. The only difference from the zero wait state case is that each read cycle now takes two *CLKOUT* cycles to complete. The addition of one wait state has done just that: added one additional cycle to each access. All other signal transitions remain the same.

Figure 5. Trace of External Data Space Reads with One Wait States



Write Cycles

The discussion of the basic write bus cycle proceeds using the same approach to the previous read discussion. The code sequence in Example 3 generates two distinct write bus cycles on the external memory interface.

Example 3. External Write to Data Space Code Sequence - xidstime.asm

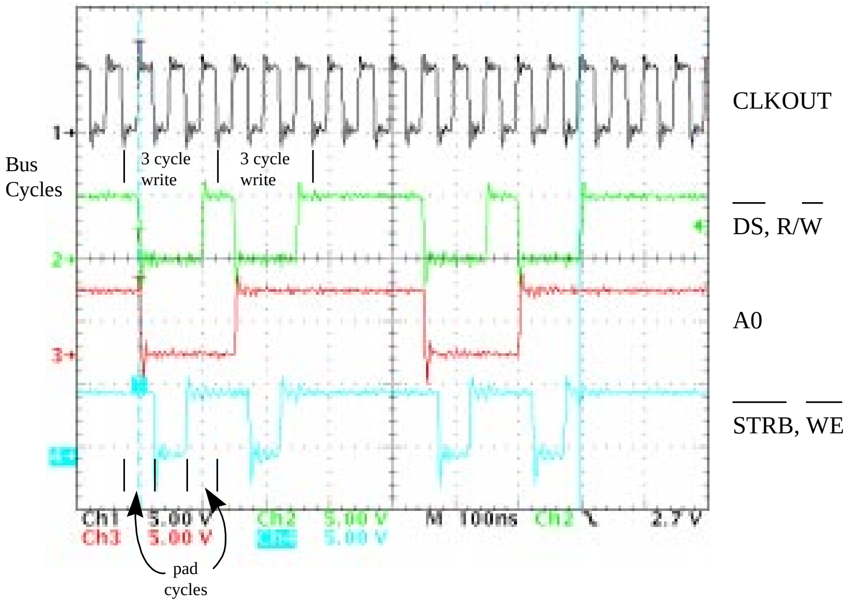
```
W_TIMING:
     LDP  #08000h/80h    ; use Data Page of first location in external ds

WLOOP
       SACL   0         ; write to external memory 0x8000
     NOP
     SACL 1               ; write to external memory 0x8001
     B    WLOOP
```

The resulting signal trace, shown in Figure 6, captures two consecutive three-cycle writes, three inactive bus cycles (branch), and then repeats. The write bus cycle is different from the read bus cycle described in the previous section. The main difference is that write bus cycles always have one inactive (pad) cycle of *CLKOUT* surrounding a write operation; one cycle before and one cycle after. This pad allows a smooth transition between the write operation and any adjacent bus operations, including other writes, and prevents bus contention between read and write operations.

During the pad cycles, $\overline{STRB}$ and $\overline{WE}$ are always in their inactive high state. The entry pad cycle begins at the falling edge of $CLKOUT$ a half cycle before the transition of $\overline{DS}$ (or $\overline{PS}$ / $\overline{IS}$) and $R/\overline{W}$ (also $W/\overline{R}$) on the rising edge of $CLKOUT$. The address pins, $A0$ shown, also become valid on the rising edge of $CLKOUT$. This relationship between $CLKOUT$ and address is valid for all write cycles. The address strobes ($\overline{DS}$, $\overline{PS}$ and $\overline{IS}$), $W/\overline{R}$ and $R/\overline{W}$ always change state on the rising edge of $CLKOUT$ during the pad cycle before and after a write or series of writes.

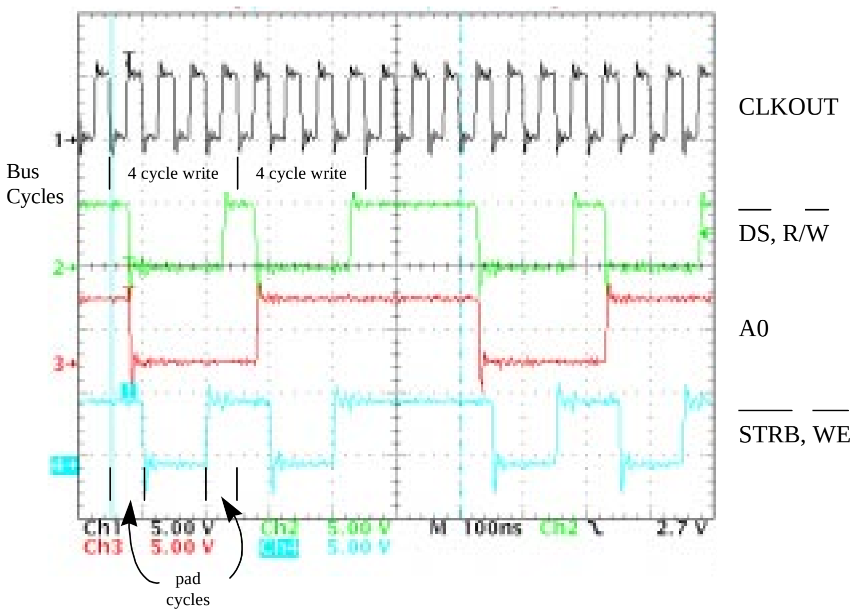
For the actual write operation, $\overline{STRB}$ and $\overline{WE}$ both go active low on the falling edge of $CLKOUT$ and stay low for one cycle, returning high on the next falling edge of $CLKOUT$. Write data is driven by the ‘240 on the same falling edge of $CLKOUT$ that cause $\overline{STRB}$ and $\overline{WE}$ to go low, and is held for a little more than one cycle of $CLKOUT$. The exit pad cycle for the first write occurs during the next cycle of $CLKOUT$, and is immediately followed by the entry pad cycle of the second write.

Figure 6. Trace of External Data Space Writes with Zero Wait States



The one wait state write bus cycle is very similar to the zero state bus cycle. The resulting trace is shown in Figure 7. With the addition of one state, the active low time of both $\overline{STRB}$ and $\overline{WE}$ has been extended by one cycle for a total of two $CLKOUT$ cycles. The entry and exit pad cycles remain one cycle each, for a four bus cycle write.

Figure 7. Trace of External Data Space Writes with One Wait States



One final note on the $W/\overline{R}$ and $R/\overline{W}$ signals. The decoding logic used for $W/\overline{R}$ is different than that used for $R/\overline{W}$. For accesses in data space between 0x7000 and 0x7fff, the $W/\overline{R}$ signal is not blocked, and transitions of this signal for internal accesses in this range are driven external to the device. All other control, address and data signals are in their inactive state when this occurs.

Typical External Data Space Access Sequences

To this point, both read and write bus cycles have been described and examples presented which show how these bus cycles look. Rarely are only reads or only writes performed in isolation. A more common and practical case would involve both reads and writes in varying combinations. The next two subsections discuss examples that show different interactions between read and write bus cycles in external data space. The read-read-write sequence will demonstrate bus behavior for consecutive reads and also for the read-write boundary. Similarly, the write-write-read sequence demonstrates the bus behavior for consecutive writes and for the write-read boundary. All other bus activity in external data space can be extrapolated based on these two sequences.

Read-Read-Write Sequence

The simple code sequence in Example 4 is written to generate the consecutive read-read-write bus cycles on the external memory interface. This code provides a tight loop for synchronization of the oscilloscope and capture of the signal key signal transitions which define the bus cycles.

Example 4. External Data Space Read-Read-Write Sequence - xidstime.asm

```
RRW_TIMING:
       LDP #08000h/80h      ; use Data Page of first location in
                               ;external ds
RLOOP2
       LACL   0                ; read from external memory 0x8000
       LACL   1                ; read from external memory 0x8001
       SACL   2                ; write to external memory 0x8002
       B      RLOOP2
```

The resulting signal traces for zero and one wait states are shown in Figure 8 and Figure 9. Notice that the $\overline{STRB}$ and $\overline{DS}$ signals do not change from the first to the second read cycle. This is typical for consecutive reads. Only the address and data signals switch between consecutive reads. For zero wait states, each read completes in one cycle, as expected. In general, n reads require n bus cycles. In the case of one wait state, both reads are extended by one additional cycle, for two two-cycle reads.

The read-write boundary is defined by the single inactive bus cycle between the read and write bus cycles. This inactive bus cycle is due to the pipeline execution of the read, which occurs one cycle after the read bus cycle. The write bus cycle actually begins immediately after the read completes execution, even though it appears as if a one-cycle delay occurs.

Figure 8. Trace of External Data Space Read-Read-Write Sequence, Zero Wait States

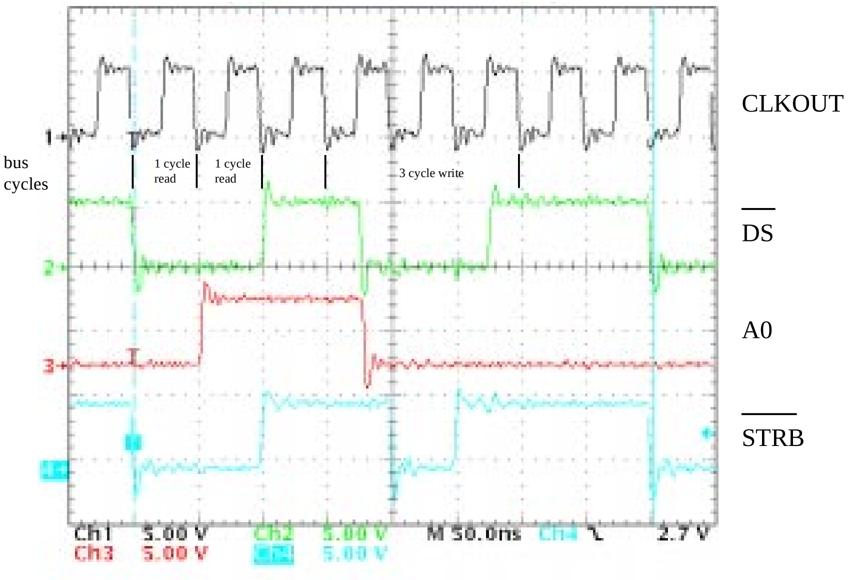
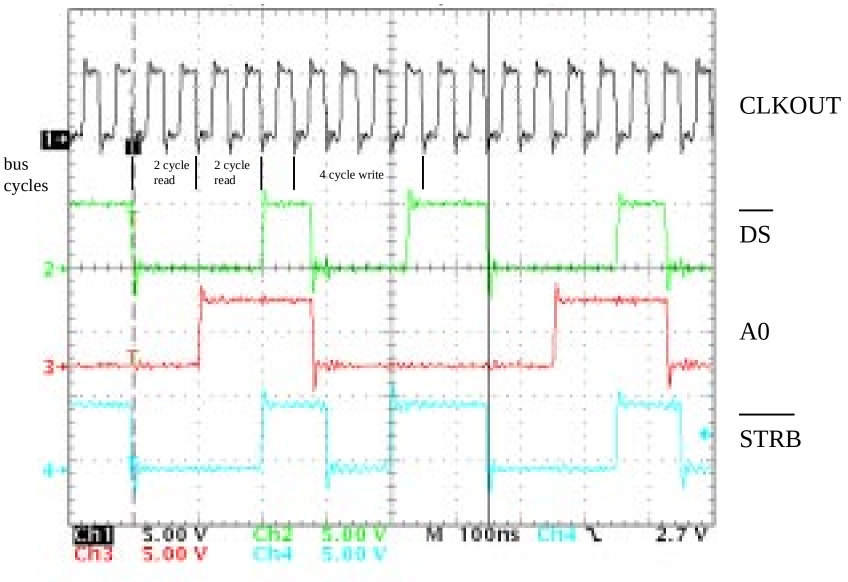


Figure 9. Trace of External Data Space Read-Read-Write Sequence, One Wait State



Write-Write-Read Sequence

The write-write-read sequence covers the opposite situation as presented by the read-read-write sequence. In this case, consecutive writes and the write-read boundary are discussed. The simple code sequence in Example 5 generates consecutive write-write-read bus cycles on the external memory interface.

Example 5. External Data Space Write-Write-Read Sequence - xidstime.asm

```
WWR_TIMING:
       LDP #08000h/80h      ;use Data Page of first location in
                               ;external ds

WLOOP2
       SACL   0             ; write to external memory 0x8001
       SACL   1             ; write to external memory 0x8001
       LACL   2             ; read from external memory 0x8002
       B      WLOOP2
```

The resulting signal traces for zero and one wait states are shown in Figure 10 and Figure 11. Notice that the $\overline{STRB}$ and $\overline{WE}$ signals switch from their active to inactive state between the first and second write cycles. This is the pad cycle described previously, and is typical for consecutive writes. Only the $\overline{DS}$ and read/write ($W/\overline{R}$ and $R/\overline{W}$) signals remain in their active state. For zero wait states, single writes require three bus cycles. For two consecutive writes, however, only five bus cycles are required; two for the first write and three for the last write. This is because consecutive writes share a pad cycle. In general, n consecutive writes require 2n+1 bus cycles. In the case of one wait state, both writes are extended by one additional cycle, for one three-cycle and one four-cycle write.

The transition from write to a read bus cycle in this case is immediate. The read bus cycle begins as soon as the write bus cycle completes.

Figure 10. Trace of External Data Space Write-Write-Read Sequence, Zero Wait States

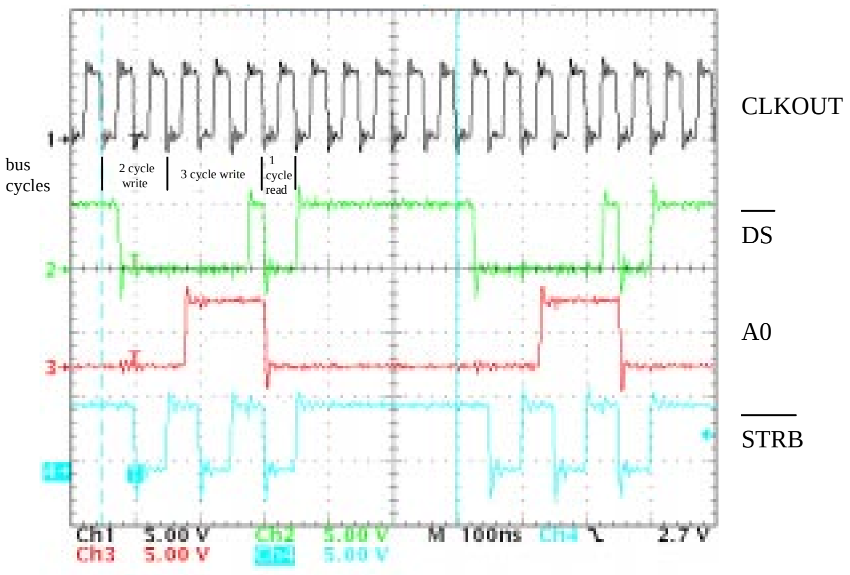
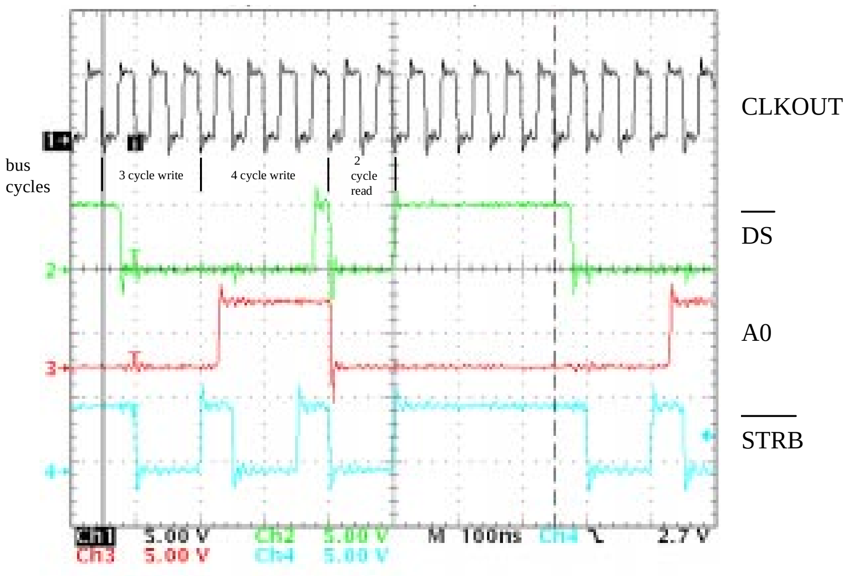


Figure 11. Trace of External Data Space Write-Write-Read Sequence, One Wait State



Bus Cycle vs. Execution Cycle Correlation

The definition and duration of an external bus cycle provides a consistent and easy to understand method of describing external memory accesses. However, the number of bus cycles per access is not necessarily equal to the number of execution cycles of the associated instruction. In the case of load accumulator instructions, the number of bus cycles is consistent with the execution time. But, in the case of a table read instruction, there will be one bus cycle for the external read, while the number of execution cycles is typically three (depending on operand source, destination and wait states). The same difference exists for write bus cycles. For a store accumulator instruction, the write bus cycle count is three, while the execution cycle count is only two. However, in the case of a write to I/O space, both bus cycle count and execution cycle count are three cycles. Clearly, there is no easily stated, general relationship between the number of bus cycles and execution cycles for an instruction.

The purpose of this section is to provide a reference for determining the difference between the execution and bus cycle counts. Since the load and store accumulator instructions have been described in detail with regard to bus cycles, these instructions will be used to compare execution and bus cycle counts. A comparison of execution and bus cycles for the key boundary conditions is provided in Table 9.

Table 9. Bus Cycle vs. Execution Cycle Correlation

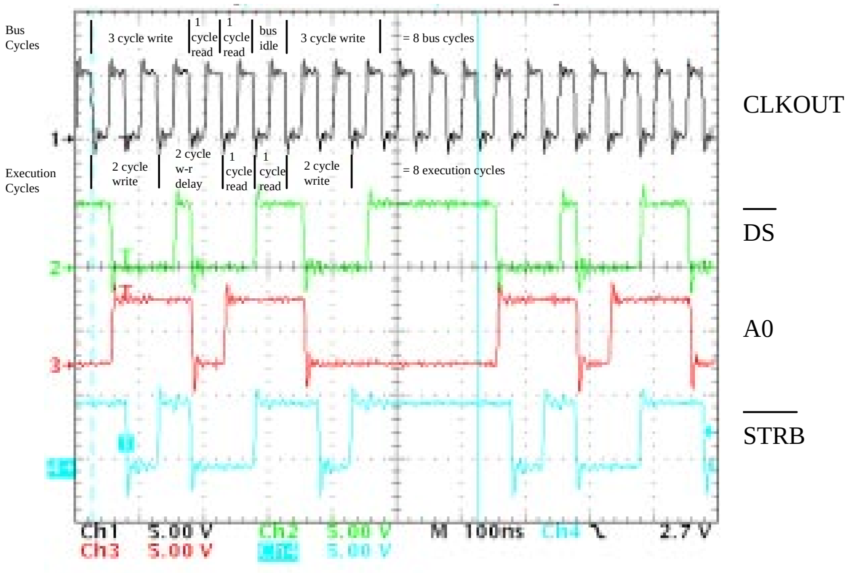
Bus Operation (loads and stores only)	# Bus Cycles	# Execution Cycles
single Read	1	1
n consecutive Reads	n	n
single Write	3	2
n consecutive Writes	2n+1	2n
consecutive Write-Write-Read	3+3+1	2+(2+2)+1
consecutive Read-Read-Write	1+1+3	1+1+2

A write-read-read-write sequence is presented to illustrate how this comparison is performed. The resulting signal trace is shown in Figure 12. There are eight active bus cycles: two three-cycle writes and two one-cycle reads. From an execution point of view, external reads take one cycle and writes take two cycles. This leaves two additional cycles unaccounted for. These two cycles are due to the write-read delay discussed in the introduction. If correlation between the bus cycle and execution cycle is needed, this same method can be applied to other code sequences.

Example 6. External Write-Read-Read-Write Code Sequence - xidstime.asm

```
WRRW_TIMING:
       LDP #08000h/80h      ; use Data Page of first location in
                               ;external ds

RLOOP2
       SACL   3         ; write to external memory 0x8003
       LACL   0         ; read from external memory 0x8000
       LACL   1         ; read from external memory 0x8001
       SACL   2         ; write to external memory 0x8002
       B      RLOOP2
```

Figure 12. Trace of External Data Space Write-Read-Read-Write Sequence



External Program Space Accesses

Accesses to external program space are essentially no different from data space accesses. The same basic read and write operations take place, just the instructions used to perform these operations are different. For external program space, there are three types of accesses. To perform read operations, the program fetch and table read instruction are used. For write operations, the table write instruction is used. The discussion of external program space accesses will focus on these three types.

Program Fetch Cycles

External program, or instruction, fetch cycles take place whenever the program code is located in external program space. Any code sequence located as such would result in program space read operations. The code used to generate the trace in this discussion is shown in Example 7. The basic instruction sequence in the LOOP section of code is:

- Store accumulator to external data space
- Load accumulator from external data space

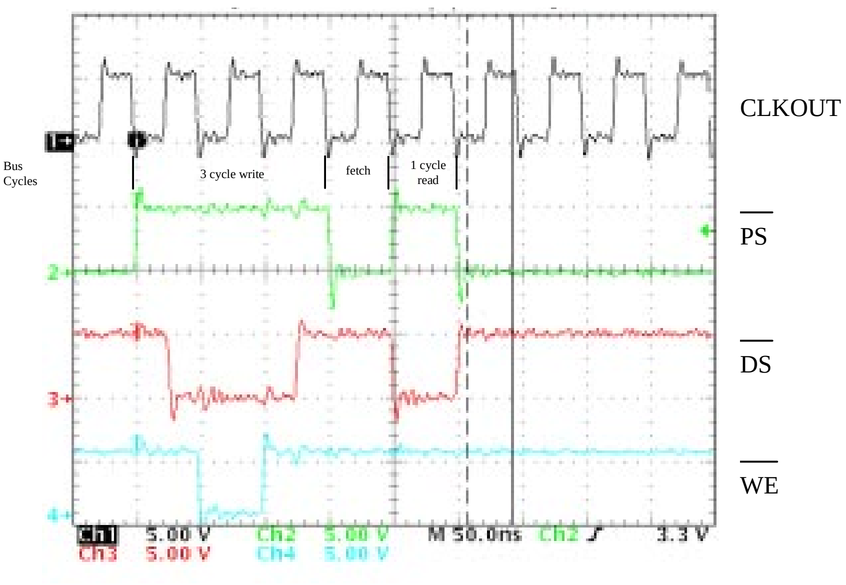
The purpose of this particular instruction sequence is to show how program fetches are interleaved with other external memory accesses.

Example 7. External Program Instruction Fetch Code Sequence - xi_fetch.asm

```
FETCH_TEST:
          LAR AR1,#8700h
          LACC   #05555h
LOOP      SACL   *+
          LACL   *-
          B   LOOP
```

The resulting signal trace from this code loop is captured in Figure 13. This trace shows $CLKOUT$ (Ch.1), $\overline{PS}$ (Ch.2), $\overline{DS}$ (Ch.3), and $\overline{WE}$ (Ch.4). Before discussing the interleave aspect of this signal trace, it is important to point out that all signals switch with respect to $CLKOUT$ as described by the guidelines provided in Table 8. For reads, both $\overline{PS}$ and $\overline{DS}$ falling edge transitions are with respect to $CLKOUT$ falling. For writes, $\overline{DS}$ falling edge occurs on the rising edge of $CLKOUT$ and $\overline{WE}$ falling edge occurs on the falling edge of $CLKOUT$.

Figure 13. Trace of External Program Space Instruction Fetch with Zero Wait States



The program fetch operation works like any read operation in data space. The only difference is that $\overline{PS}$ is active instead of $\overline{DS}$. The alternating active low periods for $\overline{PS}$ and $\overline{DS}$ show how the interleaving of both access types occurs when using the external memory interface. Normally, a write-read sequence takes three cycles for the write and one cycle for the read. In the case where an external instruction fetch coincides with an external data space access, the effective execution time of the write-read sequence is extended by one cycle for both the write and the read due to the one-cycle instruction fetch required for both instructions.

The effective execution of the data space accesses is further delayed by the addition of one wait state in program space. Example 8 shows the code modification necessary to configure the external memory interface for one program space and zero data and I/O space wait states.

Example 8. Modification of Example 7 Code for One Software Wait State

```
* Load Address of wait-state generator register, WSGR, into AR1,
* and turn off address visibility mode, AVIS.
      LDP #0              ; set DP to page zero, includes B2 RAM
      SPLK   #01h,070h    ; 70h <= 0001h
      OUT 70h,0ffffh      ; WSGR <= (07h). AVIS off; 1 ws for PS, 0
                             ; ws for IS & DS
```

The resulting signal trace in Figure 14 shows the same signals as in Figure 13. In this trace, the instruction fetches take two cycles, extending the effective execution times to five cycles for the data space write and three cycles for the data space read.

Figure 14. Trace of External Program Space Instruction Fetch with One Wait State

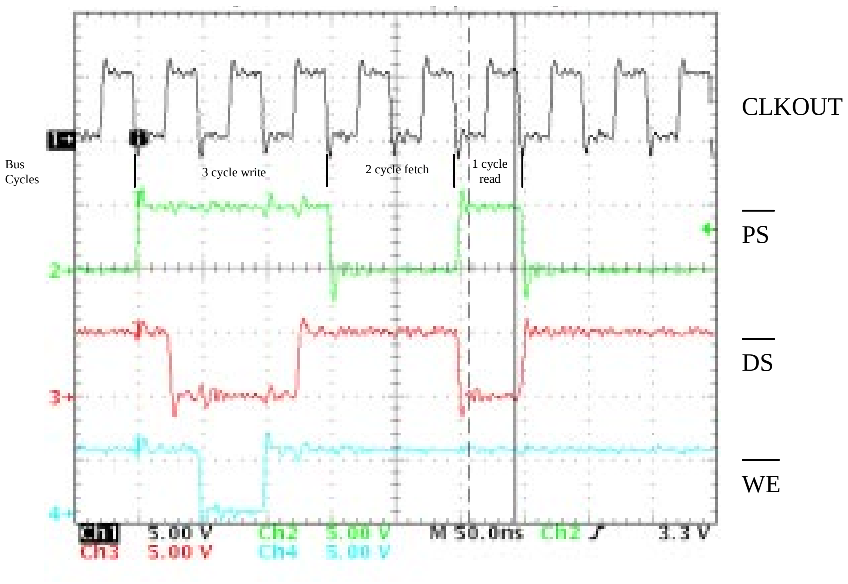


Table Read and Write Cycles

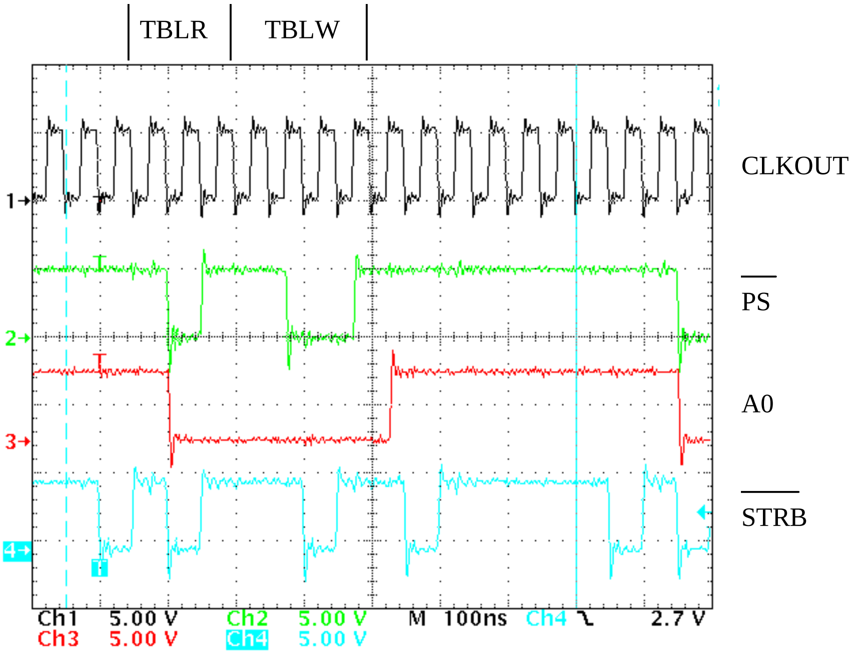
Besides the instruction fetch, the table read and table write operations are commonly used to access program space memory. Example 9 shows the code sequence used in this description. This sequence loads the external program space address 0xce00 into the accumulator for later use by the table read (TBLR) and table write (TBLW) instructions. The store accumulator (SACL) instruction that surrounds the table instructions provides an externally visible indication of when the TBLR instruction begins execution and the TBLW instruction ends execution. This makes for easier interpretation of the trace results shown in Figure 15, which shows *CLKOUT* (Ch.1), $\overline{PS}$ (Ch.2), $\overline{WE}$ (Ch.3), and $\overline{STRB}$ (Ch.4).

Example 9. External Table Read and Table Write Code Sequence - xi_table.asm

```
************************************************************
* Begin main body of code.                                 *
*                                                          *
* Access external program space                            *
************************************************************
*
     LAR  AR1,#08701h    ; AR1 <= External Data RAM
       MAR *,AR1

       LACC   #0ce00h      ;
PS     SACL   *
       TBLR   060h
       TBLW   060h
       SACL   *
       B   PS
```

The resulting trace actually shows three write operations in data space, which occur when $\overline{WE}$ and $\overline{STRB}$ are low, and $\overline{PS}$ is high. The first two $\overline{DS}$ write operations bound the table read and write instructions. The third is a repeat of the first.

Figure 15. Trace of External Table Read and Write Operations with Zero Wait States



The table read instruction begins with the first falling edge of $\overline{PS}$. It behaves just like the program fetch and data read accesses described in previous sections. All control signals switch as described in Table 8.

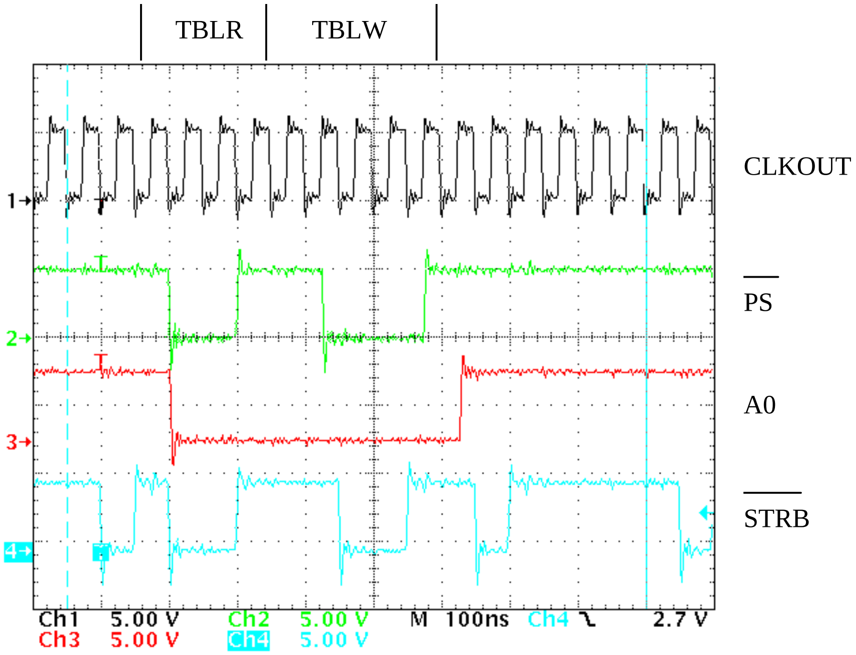
To correctly interpret the trace results for table reads or writes, it is necessary to understand how these instructions are executed by the 'C2xx CPU. The TBLR and TBLW instructions execute in a minimum of three cycles. This breaks down into three tasks:

- ❑ Program space address calculation
- ❑ Program space access
- ❑ Data space access

In a zero wait state system, each of these three tasks will take one cycle, which results in the three cycle minimum. The order of execution for these tasks is different, depending on whether a TBLR or TBLW instruction is executed. For TBLR, where data from program space (read) is copied to data space (write), the order is as presented. Tasks two and three switch places for table writes, where data from data space (read) is copied to program space (write).

In the case of an external source/destination for a TBLR/TBLW operation, the number of wait states associated with the read/write to program space will be added to the minimum cycle count. In the case shown in Figure 15, both program and data space are configured for zero wait states. However, since all external writes take two cycles, the TBLW instruction takes four cycles to complete. The labels at the top of the trace show the clock cycles during which the TBLR and TBLW instructions execute.

Figure 16. Trace of External Table Read and Write Operations with One Wait State



The impact of one wait state for program memory accesses can be seen in Figure 16. The table read operation now takes four cycles and the table write operation takes five cycles.

External I/O Space Accesses

At this point, the simple read and write accesses have been described for the data space accesses. Then, these same basic operations were shown to be “dressed-up” within the more complicated table read and write operations in program space. I/O space accesses are accomplished with the IN (read) and OUT (write) instructions, which are similar to the program space TBLR and TBLW instructions. Both are multi-task, multi-cycle instructions that include the basic read and write operations.

IN and OUT Cycles

Each IN or OUT instruction consists of two accesses: one in data space and one in I/O space. In the case of the IN instruction, the read from I/O space takes place first, followed by the writing of that value into a data space location. For the OUT instruction, a read from a data space address is performed first, followed by a write to I/O space.

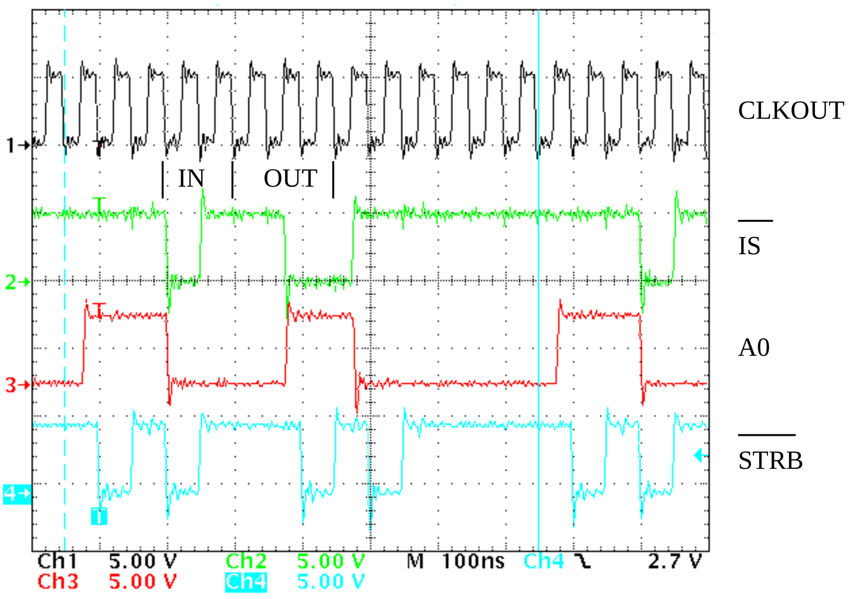
Example 10. External IN and OUT Code Sequence - xi_inout.asm

```
************************************************************
* Access external I/O space                                *
************************************************************
INOUT:
       LAR  AR1,#08701h    ;AR1 <= External Data RAM
       MAR *,AR1

IS     SACL   *+
       IN  060h,8876h          ;Read I/O at 0x8876 and write to
                               ;  DARAM at 0x60
       OUT 060h,8877h      ;Read DARAM at 0x60 and write to
                               ;  I/O at 0x8877
       SACL   *-
       B   IS
```

The code sequence used for this description is shown in Example 10. The store accumulator (SACL) instruction that surrounds the IN and OUT instructions provides an externally visible indication of when the IN instruction begins execution and the OUT instruction ends execution. This makes for easier interpretation of the trace results shown in Figure 17, which shows *CLKOUT* (Ch.1), $\overline{IS}$ (Ch.2), $\overline{WE}$ (Ch.3), and $\overline{STRB}$ (Ch.4).

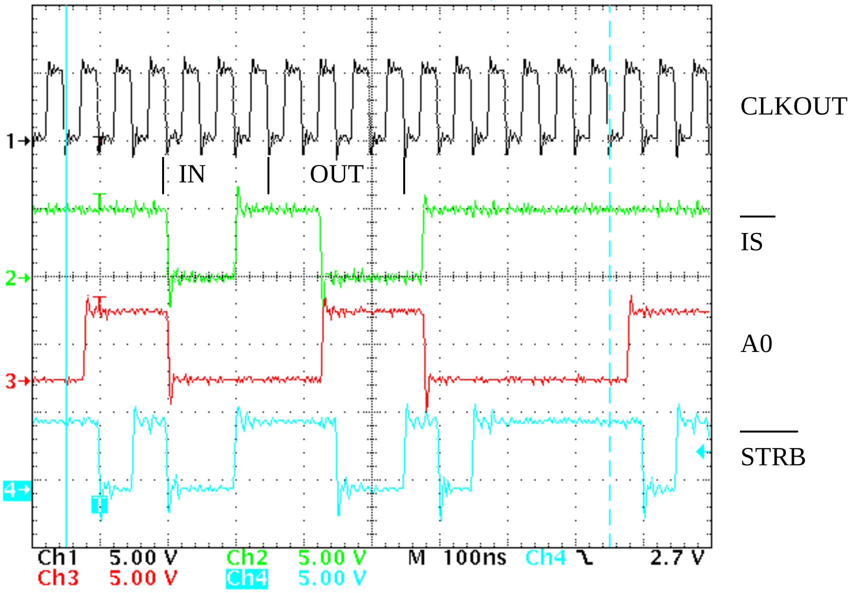
The IN instruction begins with the first falling edge of $\overline{IS}$. It behaves just like the program fetch and data read accesses described in previous sections. All control signals switch as described in Table 8.

Figure 17. Trace of External IN and OUT Operations with Zero Wait States



The best case execution cycle counts for the IN and OUT instructions are two and three cycles, respectively. In the case of an external source/destination for an IN/OUT operation, the number of wait states associated with the read/write to I/O space will be added to the minimum cycle count. In the case shown in Figure 17, both I/O and data space are configured for zero wait states. The IN instruction will still take two cycles, and the OUT instruction will take three cycles to complete. The labels at the top of the trace show the clock cycles during which the IN and OUT instructions execute.

Figure 18. Trace of External IN and OUT Operations with One Wait State



The impact of one wait state for I/O memory accesses can be seen in Figure 18. The IN instruction now takes three cycles and the OUT instruction takes four cycles.

IO Space Decoding for the Range 0xff00-0xfffe

When the external memory interface was designed, it was decided to only block $\overline{IS}$ in the address range 0xff00 to 0xfffe in IO space. Decoding and blocking the other signals ($\overline{WE}$, $\overline{STRB}$, $W/\overline{R}$, $R/\overline{W}$, and *A0-A15*) would have added additional delays to all other memory accesses. Since the address range 0xff00 to 0xfffe is reserved, it was decided not to add the additional decode logic required to prevent these pins from switching. Any external peripheral or memory located in I/O space must use the $\overline{IS}$ signal as part of its chip select logic. An inactive high state on $\overline{IS}$ is sufficient to block any accesses to I/O space above 0xff00. For accesses to I/O space in the range 0xff00-0xffff, the pins $\overline{WE}$, $\overline{STRB}$, $W/\overline{R}$, $R/\overline{W}$, and *A0-A15* switch, even when $\overline{IS}$ is inactive.

External Access Summary

The previous three sections presented an in-depth look at the external memory interface characteristics for the key instructions and/or access types for each of the three address spaces: program, data, and I/O.

The results from each section are summarized in Table 10.

Table 10. Execution Cycle Count for External Memory Accesses[1]

Address Space	Instruction	Zero Wait States	One Wait State
Data	LACL	1	2
	SACL	2	3
Program	Fetch	1	2
	TBLR	3	4
	TBLW	4	5
I/O	IN	2	3
	OUT	3	4

[1] The execution cycle counts in this table are for the specific examples presented in this report. These counts will vary depending on the source and destination of the data.

Address Visibility Mode

The external memory interface has the capability to provide program address trace through the use of address visibility mode. This mode is controlled by the AVIS bit in the wait state generator register. At reset, the AVIS bit is set to 1 and address visibility mode is enabled.

Due to the pipelined architecture of the ‘C2xx CPU, interpretation of the program address trace can be difficult. A few simple programs are provided to aid in this interpretation. These programs allow the scope to be used to trace the program fetch sequence. The control signals, $\overline{DS}$ and $CLKOUT$, and the address pins, $A0$, $A1$, and $A2$, are monitored to indicate the access type. Examples of the following three ranges in data space are presented and analyzed:

❑ Internal data space at 0x0060 (B2 DARAM)

❑ Internal data space at 0x7098 and 0x709a (I/O port register PADATDIR and PBDATDIR)

❑ External data space at 0x8000 and 0x8001

To interpret the program address trace, it is also helpful to understand the time delay between the fetch and execution of each instruction. In the ‘C2xx CPU, there are four pipeline stages:

❑ Instruction Fetch

❑ Instruction Decode

❑ Operand Fetch

❑ Instruction Execute

In the following three sections, a program code segment and resulting signal trace will be presented, followed by an instruction address to execute a correlation exercise. This exercise will make use of a code analysis which includes the instruction address, the predetermined cycle count for each instruction, and an abbreviation of the instruction. The code analysis and the signal trace data are then used to generate an idealized version of the execution path of each instruction, presented with respect to the four pipeline stages. This idealized pipeline view is provided to assist in the interpretation of the address trace but is not necessarily an exact representation of how the ‘C2xx pipeline processes instructions. Even so, this approximation does provide a reasonable foundation on which to base an understanding of the fetch address traces presented herein.

Address Trace and Internal Accesses

The simplest case to interpret the program address trace provided by the Address Visibility feature is when all program activity is confined within the CPU. This includes accesses to the CPU registers ST0 & ST1, internal DARAM blocks B0, B1 and B2, and flash EEPROM. Since the peripheral and external buses are not in use, all program address bus activity can be sent to the external address bus.

The code sequence used for this first case is presented in Example 11.

Example 11. Internal B2 DARAM Code Sequence - avis_chk.asm

```
AVIS_B2:
* Load Address of wait-state generator register, WSGR, into AR1,
* and turn off address visibility mode, AVIS.
       LDP #0          ; set DP to page zero, includes B2 RAM
       SPLK  #08h,070h     ; 70h <= 0000h
       OUT 70h,0ffffh  ; WSGR <= (07h). AVIS ON; 0 ws for DS, IS
                              ;& PS
       SPLK  #03h,071h     ;
       MAR *,AR1
LOOP
       LACL  60h           ; read from B2 DARAM
       LACL  61h           ;
       RPT 71h         ; stop fetching for 4 cycles
       NOP
       SACL  64h           ; write to B2 DARAM
       SACL  67h           ;
       B     LOOP
```

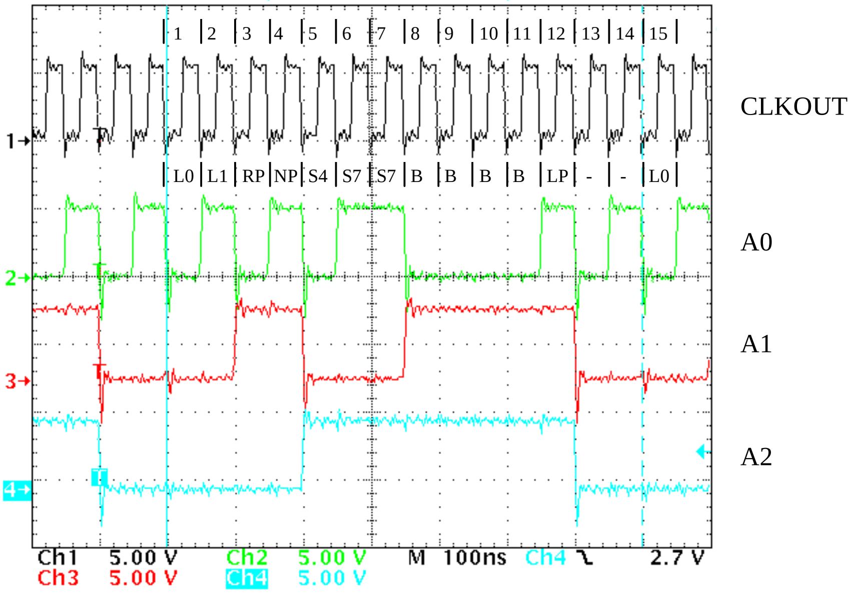
Based on the linked output file and the instruction execution times, a table summarizing the key attributes of this program excerpt is compiled in Table 11. Included in this table are the instruction, an abbreviation used in the pipeline chart, the linked program address, and the execution cycle count of each instruction. The total execution time for one iteration of this code loop is 14 cycles.

Table 11. Analysis of AVIS_B2 Code

Abbreviation	Instruction/Operand	Program Address	# Cycles
L0	LACL 60h	0x30	1
L1	LACL 61h	0x31	1
RP	RPT 71h	0x32	2
NP	NOP	0x33	4x1
S4	SACL 64h	0x34	1
S7	SACL 67h	0x35	1
B	B	0x36	4
LP	LOOP	0x37	-
		Total Loop Execution Cycle Count	14

The resulting signal trace in Figure 19 shows the address pins, $A0$, $A1$, and $A2$, and confirms the execution cycle count of 14 cycles. Each address corresponding to the instruction fetch is present on the external address pins, starting with 0x30 for L0 and continuing through 0x37 for LP.

Figure 19. Address Visibility Trace on A0-2 for Internal DARAM Accesses



This trace also provides visibility into the relationship between instruction fetch and execution. For example, the address 0x35 of S7 remains on the external address pins for two cycles, which corresponds to the execution of the RPT instruction. Likewise, the address of 0x36 of B remains on the external address pins for four cycles, which corresponds to the execution of the NOP instruction, repeated four times. This relationship is shown graphically in Table 12. Each cycle of the code loop is numbered from 1 to 14. The address present on the external address pins and the approximate instruction path through the pipeline are listed under the corresponding cycle.

Table 12. Instruction Fetch vs. Execution of AVIS_B2 Code over Time

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Address	0	1	2	3	4	5	5	6	6	6	6	7	8	9	0
Fetch	L0	L1	RP	NP	S4	S7	S7	B	B	B	B	LP	-	-	L0
Decode	-	L0	L1	RP	NP	S4	S4	S7	S7	S7	S7	B	LP	-	-
Operand	-	-	L0	L1	RP	NP	NP	S4	S4	S4	S4	S7	B	LP	-
Execute	LP	-	-	L0	L1	RP	RP	NP	NP	NP	NP	S4	S7	B	LP

In this case when only internal resources are accessed by the program code, interpretation of the address trace provided by address visibility mode is easily understood by examination of a few external pins using an oscilloscope. The pipeline chart and code analysis table provide additional support and confirmation of the expected operation.

Address Trace with Peripheral Accesses

A somewhat more complicated example is provided which demonstrates how multi-cycle instructions affect instruction fetch. The program code accesses only on-chip resources, but the destination of the accesses has changed to the peripheral registers. As in the previous case, the code sequence (Example 12), code analysis (Table 13), signal trace (Figure 20), and pipeline chart (Table 14) are provided as reference.

Example 12. Peripheral Register Access Code Sequence - avis_chk.asm

```
AVIS_PER
* Load Address of wait-state generator register, WSGR, into AR1,
* and turn off address visibility mode, AVIS.
       LDP #0              ; set DP to page zero, includes B2 RAM
       SPLK   #08h,070h    ; 70h <= 0000h
       OUT 70h,0ffffh      ; WSGR <= (07h). AVIS ON; 0 ws for DS, IS
                              ;& PS
       LDP #07080h/80h     ; use Data Page of first location in
                              ; external ds
       MAR *,AR1
LOOP1
       LACL   PADATDIR     ;
       SACL   PADATDIR
       SACL   PBDATDIR
       LACL   PBDATDIR
       B      LOOP1
```

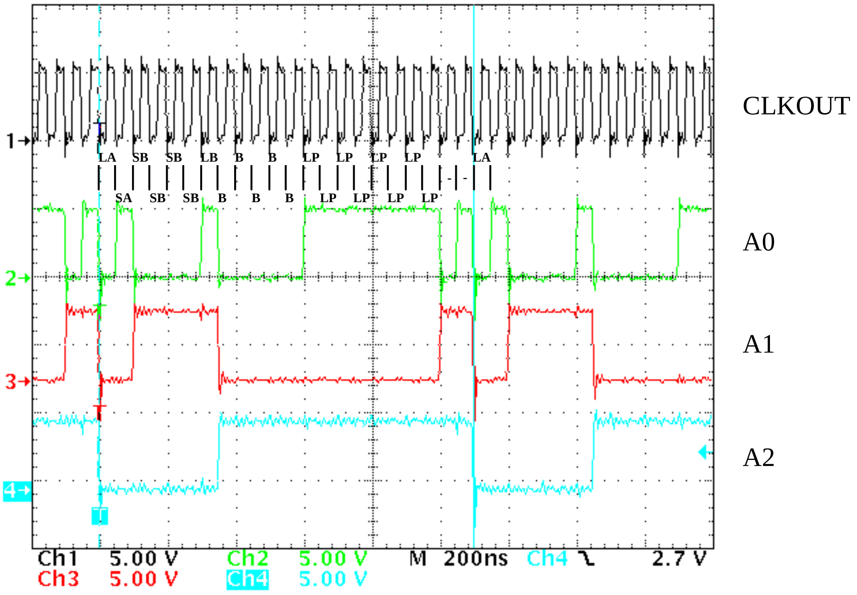
The code sequence requires 22 cycles per iteration. Since all accesses are internal to the device, every instruction fetch is visible on the external address pins. Due to the many multi-cycle instructions, most instruction fetch addresses are present for more than one cycle. This makes interpretation of the execution cycle time of the instruction more difficult, but the instruction fetch sequence is not interrupted.

Table 13. Analysis of AVIS_PER Code

Abbreviation	Instruction/Operand	Program Address	# Cycles
LA	LACL PADATDIR	0x40	4
SA	SACL PADATDIR	0x41	5
SB	SACL PBDATDIR	0x42	4+1
LB	LACL PBDATDIR	0x43	4
B	B	0x44	4
LP	LOOP1	0x45	-
	Total Loop Execution Cycle Count		22

The resulting signal trace in Figure 20 shows the address pins, $A0$, $A1$, and $A2$, and confirms the execution cycle count of 22 cycles. Each address corresponding to the instruction fetch is present on the external address pins, starting with 0x40 for LA and continuing through 0x45 for LP.

Figure 20. Address Visibility Trace on A0-2 for Peripheral Register Accesses



The pipeline chart is constructed by observing the fetched address in each of the 22 cycles and then estimating the flow of each instruction through the pipeline based on its execution time. Again, the idealized pipeline representation does not allow for an exact description. However, it does provide a reasonable approximation for the purpose of interpreting the fetch trace observed on the external address pins.

Table 14. Instruction Fetch vs. Execution of AVIS_PER Code over Time

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
Address	0	1	2	2	2	2	3	4	4	4	4	4	5	5	5	5	5	5	5	5	6	7	0
Fetch	LA	SA	SB	SB	SB	SB	LB	B	B	B	B	B	LP	LP	LP	LP	LP	LP	LP	LP	-	-	LA
Decode	-	LA	SA	SA	SA	SA	SB	LB	LB	LB	LB	LB	B	B	B	B	B	B	B	B	LP	-	-
Operand	-	-	LA	LA	LA	LA	SA	SB	SB	SB	SB	SB	LB	LB	LB	LB	LB	LB	LB	LB	B	LP	-
Execute	LP	-	-	-	-	-	LA	SA	SA	SA	SA	SA	SB	SB	SB	SB	x	-	-	-	LB	B	LP

Both this and the previous example show the case where the program code does not access external address space. In these examples, the instruction address fetch sequence is uninterrupted and can be taken directly from the external address pins.

Address Trace with External Accesses

The purpose of this section is to show the effect external accesses have on the instruction address trace when address visibility mode is enabled. The instruction address trace will not be continuous, since the address of the external access will take precedence over the fetch address. This case also allows for a comparison of external address bus activity for address visibility mode enabled and disabled.

The code sequence (Example 13), code analysis (Table 15), signal trace (Figure 21), and pipeline chart (Table 16) are again provided as reference.

Example 13. External Memory Access Code Sequence - avis_chk.asm

```
AVIS_EXT
* Load Address of wait-state generator register, WSGR, into AR1,
* and turn off address visibility mode, AVIS.
       LDP #0              ; set DP to page zero, includes B2 RAM
       SPLK   #08h,070h    ; 70h <= 0000h
       OUT 70h,0ffffh      ; WSGR <= (07h). AVIS ON; 0 ws for DS, IS
                              ;& PS
       LDP #08000h/80h     ; use Data Page of first location in
                              ;external ds
       LAR AR2,#PADATDIR ; AR2 <= Port A DAT/DIR register
       MAR *,AR1
LOOP2
       LACL   0               ; read from external memory
       RPT #3              ; stop instruction fetch for 4 cycles
       NOP
       SACL   1               ; write to external memory
       B      LOOP2
```

The code sequence requires 13 cycles per iteration. However, two of the instructions access external data space at addresses 0x8000 (L0) and 0x8001 (S1). These external accesses interrupt the internal program address stream on the external address pins. This makes interpretation of the instruction fetch sequence more difficult, since there is a resource conflict on the external address pins.

This conflict is automatically resolved by the decode logic in the external memory interface. The external memory address will always take priority over the program fetch address. This priority is implemented based on the state of the address space strobes, $\overline{DS}$, $\overline{PS}$, and $\overline{IS}$. Program fetch addresses are output on the external address pins whenever AVIS bit is set AND all three strobes are inactive high. As soon as one of the strobes goes active low, the address pins will switch from the program fetch address to the external memory address. The external address pins switch back to the current program fetch address when the strobe goes back to its inactive high state.

Table 15. Analysis of AVIS_EXT Code

Abbreviation	Instruction/Operand	Program Address	# Cycles
L0	LACL 0	0x50	1
RP	RPT #3	0x51	2
NP	NOP	0x52	4
S1	SACL 1	0x53	2
B	B	0x54	4
LP	LOOP2	0x55	-
E0	Execution Address of LACL 0	-	-
E1	Execution Address of SACL 1	-	-
	Total Loop Execution Cycle Count		13

The resulting signal trace in Figure 21 shows the address pins, $A0$, $A1$, and $A2$, and confirms the execution cycle count of 13 cycles. The trace in Figure 22 shows the address pins, $A0$ and $A1$, but replaces $A2$ with $\overline{DS}$ to provide visibility between the fetch address and the external memory address. In this case, not all instruction fetch addresses are present on the external address pins. In cycle 3, the execution address of the L0 instruction (0x8000) replaces the fetch address of the NOP instruction (0x52). The glitch on $A1$ that occurs on the first falling edge of $\overline{DS}$ (cycle 3) provides an indication of the switch between fetch address and external memory address. The half cycles that precede and follow the second low pulse of $\overline{DS}$ (first half of cycle 11 and last half of cycle 13) also represent the switch between fetch and external memory addresses.

In both cases, the conflict caused by external memory access and address visibility mode result in undesirable bus activity. This underscores the fact that address visibility mode is intended for use when no external memory accesses will occur. In applications where external memory is used, the address visibility mode should be disabled to prevent this undesired address bus activity.

Figure 21. Address Visibility Trace on A0-2 for External Memory Accesses

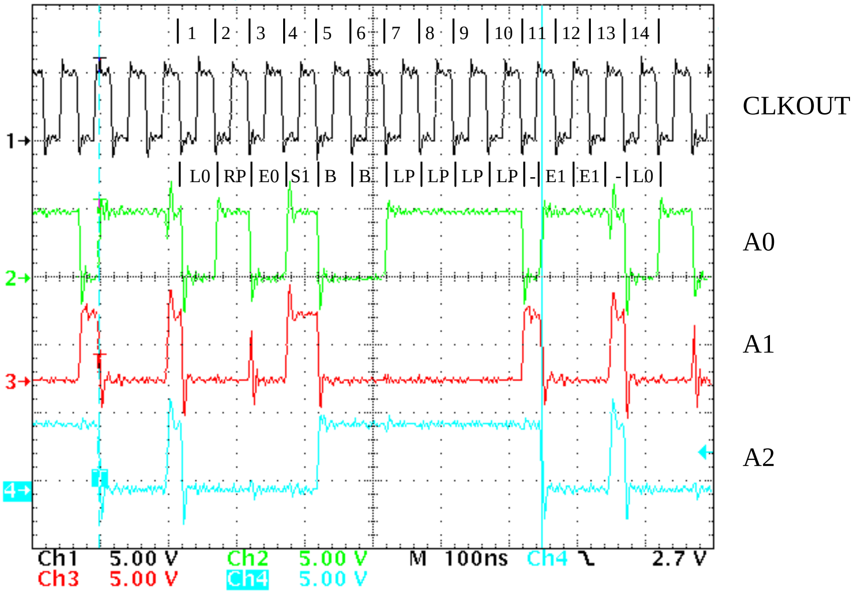
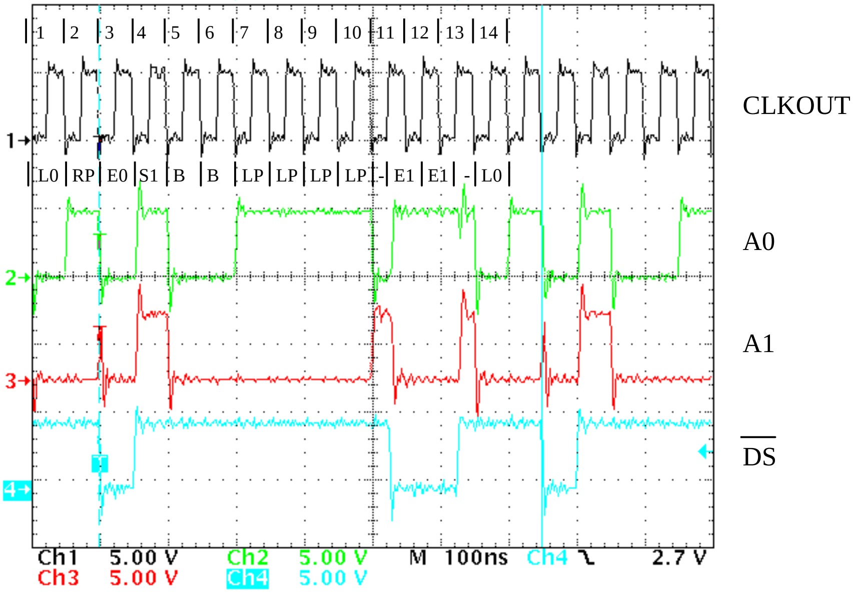


Figure 22. Address Visibility Trace on A0-1 and $\overline{DS}$ for External Memory Accesses



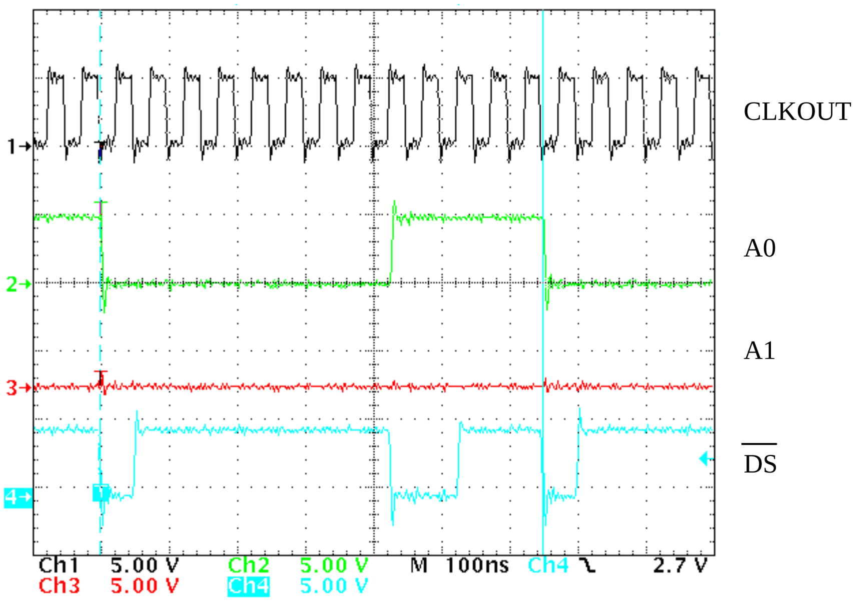
The pipeline chart is constructed by observing the address and data strobe in each of the 13 cycles and then estimating the flow of each instruction through the pipeline based on its execution time. In cycle 3, the execution address of the L0 instruction appears on the external address pins. This is because the operand is located in external memory and must be fetched prior to execution of the instruction. The pipeline chart is subdivided into halves for cycles 10 through 12. This is to represent the bus activity on the external pins as observed in Figure 22. The half cycles during the address fetch for two discarded instructions (due to the branch) are the pad cycles described in the Write section. Again, the idealized pipeline representation does not provide an exact description of pipeline operation, but does provide sufficient information to interpret the fetch trace observed on the external pins.

Table 16. Instruction Fetch vs. Execution of AVIS_EXT Code over Time

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**10.5**	**11.5**	**12.5**	**13**	**14**
Address	0	1	0	3	4	4	5	5	5	5	6	1	1	7	0
Fetch	L0	RP	E0	S1	B	B	LP	LP	LP	LP	-	E1	E1	-	L0
Decode	-	L0	RP	NP	S1	S1	B	B	B	B	LP	LP	LP	-	-
Operand	-	-	L0	RP	NP	NP	S1	S1	S1	S1	B	B	B	LP	-
Execute	LP	-	-	L0	RP	RP	NP	NP	NP	NP	S1	S1	S1	B	LP

Finally, a comparison of the external address activity between address visibility enable and disabled can be observed by comparing the signal trace provided in Figure 23 to those provided in Figure 22. When address visibility mode is disabled, the address pins switch between the read address of 0x8000 and the write address of 0x8001. In between the external bus cycles indicated by an active low $\overline{DS}$, the address bus maintains its previous value.

Figure 23. Address Visibility Mode Disabled for External Memory Accesses



Address Visibility Summary

The address visibility mode implemented on the TMS320x240 provides instruction fetch address trace capability for applications which use the internal flash EEPROM of the ‘F240 or masked ROM of the ‘C240. The instruction address allows the user to determine basic program flow by observing the fetch address of each instruction. This mode can also be used to provide some indication of the execution cycle count by interpretation of the instruction fetch delays caused by multi-cycle instructions. However, address visibility mode was not designed for use in systems with external memory accesses. Enabling address visibility mode in systems that access external address space will reduce the effectiveness of the instruction fetch trace and introduce undesired bus activity on the external address pins.

The examples presented in the previous sections relied on short code sequences that facilitated analysis using a four-channel oscilloscope. For applications which have large program size with multiple discontinuities (branches, calls, interrupts), a logic analyzer with a trace buffer and external trigger input is recommended.

Appendix A. Test Setup

The basic test setup consists of the ‘F240 EVM board configured for 10 MHz external oscillator input, PLL enabled in multiply by 2 mode, which results in DSP Clock rate of 20MHz. The supply voltage is fixed at 5V, and all testing is done at room temperature.

Appendix B. Assemble and Link Options

asmlnk.bat

```
dspa -S %1.ASM -v2xx
dsplnk %1.obj evm_lnk.cmd -o %1.out
```

evm_lnk.cmd

```
/*---------------------------------------------------------------*/
/*  LINKER COMMAND FILE - MEMORY SPECIFICATION for F240 EVM      */
/*  Last update 31 DEC 96                        */
/*---------------------------------------------------------------*/

MEMORY
{
       PAGE 0 :  VECS   : origin =    0h , length =   040h   /* PROGRAM */
                 PROG   : origin =   40h , length =  3FC0h    /* FLASH   */
                 XPM : origin = 4000h , length =  1000h    /* XPM     */
                 ISR : origin = 5000h , length =  1000h    /* ISR    */
                 B0PGM : origin = 0fe00h , length =  0100h    /* B0    */

       PAGE 1 :  MMRS   : origin =    0h , length =   05Fh    /* MMRS    */
                 B2  : origin = 0060h , length =   020h    /* DARAM   */
                 B0  : origin = 0200h , length =  0100h    /* DARAM   */
                 B1  : origin = 0300h , length =  0100h    /* DARAM   */
                 DATA   : origin = 8000h , length =  8000h    /* XDM     */
}

/*-------------------------------------------------------------------------*/
/* SECTIONS ALLOCATION                                                     */
/*-------------------------------------------------------------------------*/
SECTIONS
{
    .reset   : { } > VECS      PAGE 0      /* Interrupt Vector Table     */
    .vectors : { } > VECS      PAGE 0      /* INTERRUPT VECTOR TABLE     */
    .start   : { } > PROG      PAGE 0        /* CODE                       */
    .text    : { } > PROG      PAGE 0      /* CODE                */
    .b0pgm   : { } > B0PGM     PAGE 0      /* CODE                */
    .data    : { } > PROG      PAGE 0      /* INITIALIZATION DATA TABLES */
    .xmem    : { } > XPM       PAGE 0      /* External Program Memory    */
    .xint    : { } > ISR       PAGE 0      /* External Interrupt Routine */
    .mmrs    : { } > MMRS      PAGE 1      /* Memory Mapped Registers    */
    .blk0    : { } > B0        PAGE 1      /* Block B0 - page 4       */
    .bss     : { } > B2        PAGE 1      /* Block B2 - page 0       */
    .blk1    : { } > B1        PAGE 1      /* Block B1 - page ?       */
    .blk3    : { } > DATA      PAGE 1      /* External Data Memory    */
}
```

Appendix C. Header Files

f240regs.h

```
;***********************************************************************
; File Name:  f240reg.h
; Originator: Texas Instruments
;
; Description:   F240 Header file containing all peripheral register
;            declarations as well as other useful definitions.
;
; Last Updated:   27 May 1997
;
;***********************************************************************



;-----------------------------------------------------------------------
; On Chip Periperal Register Definitions (All registers mapped into data
; space unless otherwise noted)
;-----------------------------------------------------------------------

;C2xx Core Registers
;~~~~~~~~~~~~~~~~~~~~
IMR       .set   0004h        ;Interrupt Mask Register
GREG      .set   0005h        ;Global memory allocation Register
IFR       .set   0006h        ;Interrupt Flag Register

;System Module Registers
;~~~~~~~~~~~~~~~~~~~~~~~
SYSCR     .set   07018h    ;System Module Control Register
SYSSR     .set   0701Ah    ;System Module Status Register
SYSIVR .set   0701Eh    ;System Interrupt Vector Register

;Watch-Dog(WD) / Real Time Int(RTI) / Phase Lock Loop(PLL) Registers
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
RTICNTR   .set   07021h    ;RTI Counter Register
WDCNTR    .set   07023h    ;WD Counter Register
WDKEY     .set   07025h    ;WD Key Register
RTICR     .set   07027h    ;RTI Control Register
WDCR      .set   07029h    ;WD Control Register
CKCR0     .set   0702Bh    ;Clock Control Register 0
CKCR1     .set   0702Dh    ;Clock Control Register 1

;Analog-to-Digital Converter(ADC) registers
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
ADCTRL1   .set   07032h    ;ADC Control Register 1
ADCTRL2   .set   07034h    ;ADC Control Register 2
ADCFIFO1  .set   07036h    ;ADC Data Register FIFO1
ADCFIFO2  .set   07038h    ;ADC Data Register FIFO2

;Serial Peripheral Interface (SPI) Registers
```

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
SPICCR .set   07040h    ;SPI Configuration Control Register
SPICTL .set   07041h    ;SPI Operation Control Register
SPISTS .set   07042h    ;SPI Status Register
SPIBRR .set   07044h    ;SPI Baud Rate Register
SPIEMU    .set   07046h    ;SPI Emulation buffer Register
SPIBUF    .set   07047h    ;SPI Serial Input Buffer Register
SPIDAT    .set   07049h    ;SPI Serial Data Register
SPIPC1 .set   0704Dh    ;SPI Port Control Register 1
SPIPC2 .set   0704Eh    ;SPI Port Control Register 2
SPIPRI    .set   0704Fh    ;SPI Priority control Register

;Serial Communications Interface (SCI) Registers
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
SCICCR .set   07050h    ;SCI Communication Control Register
SCICTL1   .set   07051h    ;SCI Control Register 1
SCIHBAUD  .set   07052h    ;SCI Baud Select register, high bits
SCILBAUD  .set   07053h    ;SCI Baud Select register, high bits
SCICTL2   .set   07054h    ;SCI Control Register 2
SCIRXST   .set   07055h    ;SCI Receive Status Register
SCIRXEMU  .set   07056h    ;SCI Emulation data buffer Register
SCIRXBUF  .set   07057h    ;SCI Receiver data buffer Register
SCITXBUF  .set   07059h    ;SCI Transmit data buffer Register
SCIPC2 .set   0705Eh    ;SCI Port Control Register 2
SCIPRI    .set   0705Fh    ;SCI Priority Control Register

;External Interrupt Registers
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~
XINT1CR      .set   07070h    ;Interrupt 1 Control Register
NMICR     .set   07072h    ;Non-maskable Interrupt Control Register
XINT2CR   .set   07078h    ;Interrupt 2 Control Register
XINT3CR      .set   0707Ah    ;Interrupt 3 Control Register

;Digital I/O
;~~~~~~~~~~~
OCRA      .set   07090h    ;Output Control Reg A
OCRB      .set   07092h    ;Output Control Reg B
PADATDIR  .set   07098h    ;I/O port A Data & Direction reg.
PBDATDIR  .set   0709Ah    ;I/O port B Data & Direction reg.
PCDATDIR  .set   0709Ch    ;I/O port C Data & Direction reg.

;General Purpose Timer Registers - Event Manager (EV)
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
GPTCON .set   7400h        ;General Purpose Timer Control Register
T1CNT     .set   7401h        ;GP Timer 1 Counter Register
T1CMPR .set   7402h        ;GP Timer 1 Compare Register
T1PR      .set   7403h        ;GP Timer 1 Period Register
T1CON     .set   7404h        ;GP Timer 1 Control Register
T2CNT     .set   7405h        ;GP Timer 2 Counter Register
T2CMPR .set   7406h        ;GP Timer 2 Compare Register
T2PR      .set   7407h        ;GP Timer 2 Period Register
```

```
T2CON     .set   7408h        ;GP Timer 2 Control Register
T3CNT     .set   7409h        ;GP Timer 3 Counter Register
T3CMPR .set   740Ah        ;GP Timer 3 Compare Register
T3PR      .set   740Bh        ;GP Timer 3 Period Register
T3CON     .set   740Ch        ;GP Timer 3 Control Register

;Full & Simple Compare Unit Registers - Event Manager (EV)
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
COMCON .set   7411h        ;Compare Control Register
ACTR      .set   7413h        ;Full Compare Action Control Register
SACTR     .set   7414h        ;Simple Compare Action Control Register
DBTCON .set   7415h        ;Dead-band Timer Control Register
CMPR1     .set   7417h        ;Full Compare Unit 1 Compare Register
CMPR2     .set   7418h        ;Full Compare Unit 2 Compare Register
CMPR3     .set   7419h        ;Full Compare Unit 3 Compare Register
SCMPR1 .set   741Ah        ;Simple Compare Unit 1 Compare Register
SCMPR2 .set   741Bh        ;Simple Compare Unit 2 Compare Register
SCMPR3 .set   741Ch        ;Simple Compare Unit 3 Compare Register

;Capture & QEP Registers - Event Manager (EV)
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
CAPCON .set   7420h        ;Capture Control Register
CAPFIFO   .set   7422h        ;Capture FIFO Status Register
CAP1FIFO  .set   7423h        ;Capture 1 Two-level deep FIFO Register
CAP2FIFO  .set   7424h        ;Capture 2 Two-level deep FIFO Register
CAP3FIFO  .set   7425h        ;Capture 3 Two-level deep FIFO Register
CAP4FIFO  .set   7426h        ;Capture 4 Two-level deep FIFO Register

;Interrupt Registers - Event Manager (EV)
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
EVIMRA .set   742Ch        ;EV Interrupt Mask Register A
EVIMRB .set   742Dh        ;EV Interrupt Mask Register B
EVIMRC .set   742Eh        ;EV Interrupt Mask Register C
EVIFRA .set   742Fh        ;EV Interrupt Flag Register A
EVIFRB .set   7430h        ;EV Interrupt Flag Register B
EVIFRC .set   7431h        ;EV Interrupt Flag Register C
EVIVRA .set   7432h        ;EV Interrupt Vector Register A
EVIVRB .set   7433h        ;EV Interrupt Vector Register B
EVIVRC .set   7434h        ;EV Interrupt Vector Register C

;Flash Module Registers (mapped into Program space)
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
SEG_CTR   .set   0h        ;Flash Segment Control Register
WADRS     .set   2h        ;Flash Write Address Register
WDATA     .set   3h        ;Flash Write Data Register

;Wait State Generator Registers (mapped into I/O space)
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
WSGR      .set   0FFFFh    ;Wait State Generator Register
```

```
;-----------------------------------------------------------------------
; Constant Definitions
;-----------------------------------------------------------------------

;Data Memory Boundary Addresses
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
B0_SADDR  .set   00200h    ;Block B0 start address
B0_EADDR  .set   002FFh    ;Block B0 end address
B1_SADDR  .set   00300h    ;Block B1 start address
B1_EADDR  .set   003FFh    ;Block B1 end address
B2_SADDR  .set   00060h    ;Block B2 start address
B2_EADDR  .set   0007Fh    ;Block B2 end address
XDATA_SADDR  .set   08000h    ;External Data Space start address
XDATA_EADDR  .set  09FFFh     ;External Data Space end address

;Frequently Used Data Pages
;~~~~~~~~~~~~~~~~~~~~~~~~~~
DP_0      .set 0              ;page 0 of data space
DP_PF1    .set 224            ;page 1 of peripheral file (7000h/80h)
DP_PF2    .set 225            ;page 2 of peripheral file (7080h/80h)
DP_PF3    .set 226            ;page 3 of peripheral file (7100h/80h)
DP_EV     .set 232            ;page 1 of EV reg file (7400h/80h)

;Bit codes for Test Bit instruction (BIT)
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
BIT15     .set   0000h     ;Bit Code for 15
BIT14     .set   0001h     ;Bit Code for 14
BIT13     .set   0002h     ;Bit Code for 13
BIT12     .set   0003h     ;Bit Code for 12
BIT11     .set   0004h     ;Bit Code for 11
BIT10     .set   0005h     ;Bit Code for 10
BIT9      .set   0006h     ;Bit Code for 9
BIT8      .set   0007h     ;Bit Code for 8
BIT7      .set   0008h     ;Bit Code for 7
BIT6      .set   0009h     ;Bit Code for 6
BIT5      .set   000Ah     ;Bit Code for 5
BIT4      .set   000Bh     ;Bit Code for 4
BIT3      .set   000Ch     ;Bit Code for 3
BIT2      .set   000Dh     ;Bit Code for 2
BIT1      .set   000Eh     ;Bit Code for 1
BIT0      .set   000Fh     ;Bit Code for 0

;Bit masks used by the SBIT0 & SBIT1 Macros
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
B15_MSK   .set   8000h     ;Bit Mask for 15
B14_MSK   .set   4000h     ;Bit Mask for 14
B13_MSK   .set   2000h     ;Bit Mask for 13
B12_MSK   .set   1000h     ;Bit Mask for 12
B11_MSK   .set   0800h     ;Bit Mask for 11
B10_MSK   .set   0400h     ;Bit Mask for 10
B9_MSK .set   0200h     ;Bit Mask for 9
```

```
B8_MSK .set   0100h     ;Bit Mask for 8
B7_MSK .set   0080h     ;Bit Mask for 7
B6_MSK .set   0040h     ;Bit Mask for 6
B5_MSK .set   0020h     ;Bit Mask for 5
B4_MSK .set   0010h     ;Bit Mask for 4
B3_MSK .set   0008h     ;Bit Mask for 3
B2_MSK .set   0004h     ;Bit Mask for 2
B1_MSK .set   0002h     ;Bit Mask for 1
B0_MSK .set   0001h     ;Bit Mask for 0

;-----------------------------------------------------------------------
; M A C R O - Definitions
;-----------------------------------------------------------------------

SBIT0     .macro DMA, MASK    ;Clear bit Macro
          LACC   DMA
          AND #(0FFFFh-MASK)
          SACL   DMA
          .endm

SBIT1     .macro DMA, MASK    ;Set bit Macro
          LACC   DMA
          OR #(MASK)
          SACL   DMA
          .endm

KICK_DOG  .macro           ;Watchdog reset macro
          LDP #00E0h       ;DP-->7000h-707Fh
          SPLK   #05555h, WDKEY      ;WDCNTR is enabled to be reset by next AAh
          SPLK   #0AAAAh, WDKEY      ;WDCNTR is reset
          LDP #0h          ;DP-->0000h-007Fh
          .endm
```

Appendix D. Assembly Language Programs

xidstime.asm

```
;****************************************************************************
; File Name:  xidstime.ASM  (File revision 0.1)
; Project:    x240 External Memory Interface Application Note.
; Originator: J.Crankshaw    (Texas Instruments)
;
; Target Sys: F240 EVM
;
; Description:
;
; Status:  Released.
;
; Last Update:   31 Dec 97
; ___________________________________________________________________________
; Date of Mod |         DESCRIPTION
; ------------|--------------------------------------------------------------
;      0.1     | create testcase
;            |
;            |
;            |
;****************************************************************************

     .include "f240regs.h"
*

;----------------------------------------------------------------------------
; Vector address declarations
;----------------------------------------------------------------------------
         .sect  ".vectors"

RSVECT    B    START   ; PM 0    Reset Vector     1
INT1      B    PHANTOM ; PM 2    Int level 1      4
INT2      B    PHANTOM ; PM 4 Int level 2      5
INT3      B    PHANTOM ; PM 6 Int level 3      6
INT4      B    PHANTOM ; PM 8 Int level 4      7
INT5      B    PHANTOM ; PM A Int level 5      8
INT6      B    PHANTOM ; PM C Int level 6      9
RESERVED  B    PHANTOM ; PM E (Analysis Int)    10
SW_INT8   B    PHANTOM ; PM 10   User S/W int     -
SW_INT9   B    PHANTOM ; PM 12   User S/W int     -
SW_INT10  B    PHANTOM ; PM 14   User S/W int     -
SW_INT11  B    PHANTOM ; PM 16   User S/W int     -
SW_INT12  B    PHANTOM ; PM 18   User S/W int     -
SW_INT13  B    PHANTOM ; PM 1A   User S/W int     -
SW_INT14  B    PHANTOM ; PM 1C   User S/W int     -
SW_INT15  B    PHANTOM ; PM 1E   User S/W int     -
SW_INT16  B    PHANTOM ; PM 20   User S/W int     -
TRAP      B    PHANTOM ; PM 22   Trap vector      -
```

```
NMI       B    PHANTOM ; PM 24   Non maskable Int   3
EMU_TRAP  B    PHANTOM ; PM 26   Emulator Trap     2
SW_INT20  B    PHANTOM ; PM 28   User S/W int      -
SW_INT21  B    PHANTOM ; PM 2A   User S/W int      -
SW_INT22  B    PHANTOM ; PM 2C   User S/W int      -
SW_INT23  B    PHANTOM ; PM 2E   User S/W int      -

;==============================================================================
; M A I N   C O D E  - starts here
;==============================================================================
;      .text
       .sect  ".b0pgm"
START:
       CLRC   SXM         ; Clear Sign Extension Mode
       CLRC   OVM         ; Reset Overflow Mode

* Set Data Page pointer to  page 1 of the peripheral frame
       LDP    #DP_PF1       ; Page DP_PF1 includes WET through EINT frames

* initialize WDT registers
       SPLK   #06Fh, WDCR   ; clear WDFLAG, Disable WDT, set 1 sec overflow
       SPLK   #07h, RTICR   ; clear RTI Flag, set RTI for 1 sec overflow

* configure PLL for 10MHz osc, 10MHz SYSCLK and 20MHz CPUCLK
       SPLK   #00B1h,CKCR1 ;CLKIN(OSC)=10MHz,CPUCLK=20MHz
       SPLK   #00C3h,CKCR0 ;CLKMD=PLL Enable,SYSCLK=CPUCLK/2,

* Clear reset flag bits in SYSSR (PORRST, PLLRST, ILLRST, SWRST, WDRST)
       LACL   SYSSR         ; ACCL <= SYSSR
       AND    #00FFh           ; Clear upper 8 bits of SYSSR
       SACL   SYSSR         ; Load new value into SYSSR

* initialize B2 RAM to zero's.
       LAR    AR1,#B2_SADDR  ; AR1 <= B2 start address
       MAR    *,AR1          ; use B2 start address for next indirect
       ZAC                  ; ACC <= 0
       RPT    #1fh           ; set repeat counter for 1fh+1=20h or 32 loops
       SACL   *+             ; write zeros to B2 RAM

* initialize B1 RAM to zero's.
       LAR    AR1,#B1_SADDR  ; AR1 <= B2 start address
       MAR    *,AR1          ; use B2 start address for next indirect
       ZAC                  ; ACC <= 0
       RPT    #0ffh          ; set repeat counter for ffh+1=100h or 256 loops
       SACL   *+             ; write zeros to B1 RAM

* Configure wait-state generator register, WSGR
       LDP    #0            ; set DP to page zero, includes B2 RAM
       SPLK   #02h,070h     ; 70h <= 0000h
       OUT    70h,0ffffh    ; WSGR <= (070h). AVIS OFF; 0 ws for DS, IS & PS
```

```
* Call subroutines
;      CALL R_TIMING
;      CALL W_TIMING
       CALL RRW_TIMING
;      CALL WRRW_TIMING
;      CALL WWR_TIMING
;      CALL RWWWR_TIMING
;      CALL RWR_TIMING
;      CALL RRiWWRiR_TIMING
       B      END          ; end testcase gracefully
**********************************************************
* Complete main body of code.                            *
**********************************************************

END    NOP                 ; Flush pipeline.
       NOP
       NOP
       NOP
       B      END

*******************************************************************************
*                Subroutines                                                  *
*******************************************************************************
;===========================================================================
; Routine Name: x_TIMING        Routine Type: SR
;
; Description:
;
; Originator: J. Crankshaw
;
;===========================================================================
R_TIMING:
       LDP    #08000h/80h    ; use Data Page of first location in external ds

RLOOP
       LACL   0         ; read from external memory 0x8000
       NOP
       LACL   1              ; read from external memory 0x8001
       B      RLOOP

       RET

W_TIMING:
       LDP    #08000h/80h    ; use Data Page of first location in external ds

WLOOP
       SACL   0         ; write to external memory 0x8000
       NOP
       SACL   1              ; write to external memory 0x8001
       B      WLOOP
```

```
        RET

RRW_TIMING:
        LDP  #08000h/80h    ; use Data Page of first location in external ds

RLOOP2
        LACL 0              ; read from external memory 0x8000
        LACL 1              ; read from external memory 0x8001
        SACL 2              ; write to external memory 0x8002
        B    RLOOP2

        RET

WWR_TIMING:
        LDP  #08000h/80h    ; use Data Page of first location in external ds

WLOOP2
        SACL 0              ; write to external memory 0x8001
        SACL 1              ; write to external memory 0x8001
        LACL 2              ; read from external memory 0x8002
        B    WLOOP2

        RET

WRRW_TIMING:
        LDP  #08000h/80h    ; use Data Page of first location in external ds

RLOOP3
        SACL   3            ; write to external memory 0x8003
        LACL 0              ; read from external memory 0x8000
        LACL 1              ; read from external memory 0x8001
        SACL 2              ; write to extenal memory 0x8002
        B    RLOOP3

        RET

RWWWR_TIMING:
        LDP  #08000h/80h    ; use Data Page of first location in external ds

WLOOP3
        LACL 3              ; read from extenal memory 0x8003
        SACL 0              ; write to external memory 0x8000
        SACL 1              ; write to external memory 0x8001
        SACL 5              ; write to external memory 0x8005
        LACL 2              ; read from extenal memory 0x8002
        B    WLOOP3

      RET

RWR_TIMING:
        LDP    #08000h/80h    ; use Data Page of first location in external ds
```

```
       LAR    AR1, #T1PR   ; EV timer1 period register at 0x7403
       MAR    *,AR1

LOOP3
       LACL   0               ; read from external memory 0x8000
       SACL   1               ; write to external memory 0x8001
       LACL   2                ; read from external memory 0x8002
       B      LOOP3

       RET

RRiWWRiR_TIMING:
       LDP    #08000h/80h    ; use Data Page of first location in external ds
       LAR    AR1, #060h   ; DARAM Block B2
       MAR    *,AR1

LOOP4
       LACL   0              ; read from external memory 0x8000
       LACL   *              ; read from internal DARAM 0x60
       SACL   1              ; write to external memory 0x8001
       SACL   2              ; write to external memory 0x8002
       LACL   *              ; read from internal DARAM 0x60
       LACL   3              ; read from external memory 0x8003
       B      LOOP4

       RET

*******************************************************************************
*                ISR's                                                        *
*******************************************************************************
;==============================================================================
; I S R  -  PHANTOM
;
; Description:   ISR used to trap spurious interrupts.
;              Reads System Interrupt Vector register to capture
;              vector of module that caused the interrupt.
;
; Modifies:  Changes DP and ACC. Loads vector into 61h and BADh into 60h.
;
; Last Update:   10-17-96
;==============================================================================
PHANTOM
       LDP    #DP_PF1          ; go to peripheral file data page 1
       LACL   SYSIVR          ; ACC <= [SYSIVR]
       LDP    #0               ; go to first data page, with B2
       SACL   B2_SADDR+1      ; 61h <= [SYSIVR]
       SPLK   #0BADh,B2_SADDR ; 60h <= "BAD" value indicates error
       B      END             ; Terminate gracefully

.end
```

xi_fetch.asm

```
;*****************************************************************************
; File Name:  xi_fetch.ASM  (File revision 0.1)
; Project:    x240 External Memory Interface Application Note.
; Originator: J.Crankshaw    (Texas Instruments)
;
; Target Sys: F240 EVM
;
; Description:
;
; Status:  Released.
;
; Last Update:   31 Dec 97
; ____________________________________________________________________________
; Date of Mod |         DESCRIPTION
; ------------|---------------------------------------------------------------
;      0.1     | create testcase
;            |
;            |
;            |
;*****************************************************************************

     .include "f240regs.h"
*

;-----------------------------------------------------------------------------
; Vector address declarations
;-----------------------------------------------------------------------------
         .sect  ".vectors"

RSVECT    B    START   ; PM 0    Reset Vector      1
INT1      B    PHANTOM ; PM 2    Int level 1       4
INT2      B    PHANTOM ; PM 4 Int level 2      5
INT3      B    PHANTOM ; PM 6 Int level 3      6
INT4      B    PHANTOM ; PM 8 Int level 4      7
INT5      B    PHANTOM ; PM A Int level 5      8
INT6      B    PHANTOM ; PM C Int level 6      9
RESERVED  B    PHANTOM ; PM E (Analysis Int)    10
SW_INT8   B    PHANTOM ; PM 10   User S/W int      -
SW_INT9   B    PHANTOM ; PM 12   User S/W int      -
SW_INT10  B    PHANTOM ; PM 14   User S/W int      -
SW_INT11  B    PHANTOM ; PM 16   User S/W int      -
SW_INT12  B    PHANTOM ; PM 18   User S/W int      -
SW_INT13  B    PHANTOM ; PM 1A   User S/W int      -
SW_INT14  B    PHANTOM ; PM 1C   User S/W int      -
SW_INT15  B    PHANTOM ; PM 1E   User S/W int      -
SW_INT16  B    PHANTOM ; PM 20   User S/W int      -
TRAP      B    PHANTOM ; PM 22   Trap vector       -
NMI       B    PHANTOM ; PM 24   Non maskable Int   3
EMU_TRAP  B    PHANTOM ; PM 26   Emulator Trap     2
SW_INT20  B    PHANTOM ; PM 28   User S/W int      -
```

```
SW_INT21  B    PHANTOM ; PM 2A   User S/W int      -
SW_INT22  B    PHANTOM ; PM 2C   User S/W int      -
SW_INT23  B    PHANTOM ; PM 2E   User S/W int      -

;==============================================================================
; M A I N   C O D E  - starts here
;==============================================================================
       .text
START:
       CLRC   SXM         ; Clear Sign Extension Mode
       CLRC   OVM         ; Reset Overflow Mode

* Set Data Page pointer to  page 1 of the peripheral frame
       LDP    #DP_PF1      ; Page DP_PF1 includes WET through EINT frames

* initialize WDT registers
       SPLK   #06Fh, WDCR  ; clear WDFLAG, Disable WDT, set for 1 sec overflow
       SPLK   #07h, RTICR  ; clear RTI Flag, set RTI for 1 sec overflow

* configure PLL for 10MHz osc, 10MHz SYSCLK and 20MHz CPUCLK
       SPLK   #00B1h,CKCR1 ;CLKIN(OSC)=10MHz,CPUCLK=20MHz
       SPLK   #00C3h,CKCR0   ;CLKMD=PLL Enable,SYSCLK=CPUCLK/2,

* Clear reset flag bits in SYSSR (PORRST, PLLRST, ILLRST, SWRST, WDRST)
       LACL   SYSSR     ; ACCL <= SYSSR
       AND    #00FFh    ; Clear upper 8 bits of SYSSR
       SACL   SYSSR     ; Load new value into SYSSR

* initialize B2 RAM to zero's.
       LAR    AR1,#B2_SADDR  ; AR1 <= B2 start address
       MAR    *,AR1          ; use B2 start address for next indirect
       ZAC                 ; ACC <= 0
       RPT    #1fh           ; set repeat counter for 1fh+1=20h or 32 loops
       SACL   *+             ; write zeros to B2 RAM

* initialize B1 RAM to zero's.
       LAR    AR1,#B1_SADDR  ; AR1 <= B2 start address
       MAR    *,AR1          ; use B2 start address for next indirect
       ZAC                 ; ACC <= 0
       RPT    #0ffh          ; set repeat counter for ffh+1=100h or 256 loops
       SACL   *+             ; write zeros to B1 RAM

* Configure wait-state generator register, WSGR
       LDP    #0             ; set DP to page zero, includes B2 RAM
       SPLK   #00h,070h      ; 70h <= 0000h
       OUT    70h,0ffffh     ; WSGR <= (070h). AVIS OFF; 0 ws for DS, IS & PS

       B      FETCH_TEST

FETCH_TEST:
          LAR    AR1,#8700h
          LACC   #05555h
LOOP      SACL   *+
          LACL   *-
          B   LOOP

;===============================================================================
; I S R  -  PHANTOM
;
; Description:   Dummy ISR, used to trap spurious interrupts.
;
; Modifies:  Nothing
;
; Last Update:   16 June 95
;===============================================================================
PHANTOM   B   PHANTOM
```

xi_table.asm

```
;******************************************************************************
; File Name:  xi_table.ASM  (File revision 0.1)
; Project:    x240 External Memory Interface Application Note.
; Originator: J.Crankshaw    (Texas Instruments)
;
; Target Sys: F240 EVM
;
; Description:
;
; Status:  Released.
;
; Last Update:   31 Dec 97
; _____________________________________________________________________________
; Date of Mod |         DESCRIPTION
; ------------|----------------------------------------------------------------
;      0.1     | create testcase
;            |
;            |
;            |
;******************************************************************************

     .include "f240regs.h"
*

;------------------------------------------------------------------------------
; Vector address declarations
;------------------------------------------------------------------------------
         .sect  ".vectors"

RSVECT    B    START   ; PM 0    Reset Vector      1
INT1      B    PHANTOM ; PM 2    Int level 1       4
INT2      B    PHANTOM ; PM 4 Int level 2      5
INT3      B    PHANTOM ; PM 6 Int level 3      6
```

```
INT4      B    PHANTOM ; PM 8 Int level 4      7
INT5      B    PHANTOM ; PM A Int level 5      8
INT6      B    PHANTOM ; PM C Int level 6      9
RESERVED  B    PHANTOM ; PM E (Analysis Int)    10
SW_INT8   B    PHANTOM ; PM 10   User S/W int      -
SW_INT9   B    PHANTOM ; PM 12   User S/W int      -
SW_INT10  B    PHANTOM ; PM 14   User S/W int      -
SW_INT11  B    PHANTOM ; PM 16   User S/W int      -
SW_INT12  B    PHANTOM ; PM 18   User S/W int      -
SW_INT13  B    PHANTOM ; PM 1A   User S/W int      -
SW_INT14  B    PHANTOM ; PM 1C   User S/W int      -
SW_INT15  B    PHANTOM ; PM 1E   User S/W int      -
SW_INT16  B    PHANTOM ; PM 20   User S/W int      -
TRAP      B    PHANTOM ; PM 22   Trap vector       -
NMI       B    PHANTOM ; PM 24   Non maskable Int   3
EMU_TRAP  B    PHANTOM ; PM 26   Emulator Trap     2
SW_INT20  B    PHANTOM ; PM 28   User S/W int      -
SW_INT21  B    PHANTOM ; PM 2A   User S/W int      -
SW_INT22  B    PHANTOM ; PM 2C   User S/W int      -
SW_INT23  B    PHANTOM ; PM 2E   User S/W int      -

;===============================================================================
; M A I N   C O D E  - starts here
;===============================================================================
;      .text
       .sect  ".b0pgm"
START:
       CLRC   SXM         ; Clear Sign Extension Mode
       CLRC   OVM         ; Reset Overflow Mode

* Set Data Page pointer to  page 1 of the peripheral frame
       LDP #DP_PF1      ; Page DP_PF1 includes WET through EINT frames

* initialize WDT registers
       SPLK   #06Fh, WDCR  ; clear WDFLAG, Disable WDT, set for 1 sec overflow
       SPLK   #07h, RTICR  ; clear RTI Flag, set RTI for 1 second overflow

* configure PLL for 10MHz osc, 10MHz SYSCLK and 20MHz CPUCLK
       SPLK   #00B1h,CKCR1 ;CLKIN(OSC)=10MHz,CPUCLK=20MHz
       SPLK   #00C3h,CKCR0 ;CLKMD=PLL Enable,SYSCLK=CPUCLK/2,

* Clear reset flag bits in SYSSR (PORRST, PLLRST, ILLRST, SWRST, WDRST)
       LACL   SYSSR     ; ACCL <= SYSSR
       AND    #00FFh    ; Clear upper 8 bits of SYSSR
       SACL   SYSSR     ; Load new value into SYSSR

* initialize B2 RAM to zero's.
       LAR  AR1,#B2_SADDR  ; AR1 <= B2 start address
       MAR  *,AR1          ; use B2 start address for next indirect
       ZAC                 ; ACC <= 0
       RPT  #1fh           ; set repeat counter for 1fh+1=20h or 32 loops
```

```
      SACL *+             ; write zeros to B2 RAM

* initialize B1 RAM to zero's.
      LAR  AR1,#B1_SADDR  ; AR1 <= B2 start address
      MAR  *,AR1          ; use B2 start address for next indirect
      ZAC                 ; ACC <= 0
      RPT  #0ffh          ; set repeat counter for ffh+1=100h or 256 loops
      SACL *+             ; write zeros to B1 RAM

* Configure wait-state generator register, WSGR
      LDP  #0             ; set DP to page zero, includes B2 RAM
      SPLK #00h,070h      ; 70h <= 0000h
      OUT  70h,0ffffh     ; WSGR <= (070h). AVIS OFF; 0 ws for DS, IS & PS

************************************************************
* Begin main body of code.                                 *
*                                                          *
* Access external program space                            *
************************************************************
*
      LAR    AR1,#08701h    ; AR1 <= External Data RAM
      MAR    *,AR1

      LACC   #0ce00h      ; 1st read:  ffffh
PS    SACL   *
      TBLR   060h
      TBLW   060h
      SACL   *
      B      PS

      B      END            ; end testcase gracefully
**********************************************************
* Complete main body of code.                            *
**********************************************************

END   NOP                   ; Flush pipeline.
      NOP
      NOP
      NOP
      B      END

*******************************************************************************
*                ISR's                                                        *
*******************************************************************************
;==============================================================================
; I S R  -  PHANTOM
;
; Description:   ISR used to trap spurious interrupts.
;              Reads System Interrupt Vector register to capture
;              vector of module that caused the interrupt.
;
```

```
; Modifies:  Changes DP and ACC. Loads vector into 61h and BADh into 60h.
;
; Last Update:   10-17-96
;==============================================================================
PHANTOM
       LDP    #DP_PF1          ; go to peripheral file data page 1
       LACL   SYSIVR           ; ACC <= [SYSIVR]
       LDP    #0               ; go to first data page, with B2
       SACL   B2_SADDR+1       ; 61h <= [SYSIVR]
       SPLK   #0BADh,B2_SADDR  ; 60h <= "BAD" value indicates error
       B      END              ; Terminate gracefully

.end
```

xi_inout.asm

```
;*****************************************************************************
; File Name: xi_inout.ASM  (File revision 0.1)
; Project:   x240 External Memory Interface Application Note.
; Originator: J.Crankshaw   (Texas Instruments)
;
; Target Sys: F240 EVM
;
; Description:
;
; Status:  Released.
;
; Last Update:   31 Dec 97
; _____________________________________________________________________________
; Date of Mod |         DESCRIPTION
; ------------|----------------------------------------------------------------
;       0.1     | create testcase
;             |
;             |
;             |
;*****************************************************************************

     .include "f240regs.h"
*

;-----------------------------------------------------------------------------
; Vector address declarations
;-----------------------------------------------------------------------------
         .sect  ".vectors"

RSVECT    B    START   ; PM 0    Reset Vector     1
INT1      B    PHANTOM ; PM 2    Int level 1      4
INT2      B    PHANTOM ; PM 4 Int level 2      5
INT3      B    PHANTOM ; PM 6 Int level 3      6
INT4      B    PHANTOM ; PM 8 Int level 4      7
INT5      B    PHANTOM ; PM A Int level 5      8
INT6      B    PHANTOM ; PM C Int level 6      9
```

```
RESERVED  B    PHANTOM ; PM E (Analysis Int)    10
SW_INT8   B    PHANTOM ; PM 10   User S/W int     -
SW_INT9   B    PHANTOM ; PM 12   User S/W int     -
SW_INT10  B    PHANTOM ; PM 14   User S/W int     -
SW_INT11  B    PHANTOM ; PM 16   User S/W int     -
SW_INT12  B    PHANTOM ; PM 18   User S/W int     -
SW_INT13  B    PHANTOM ; PM 1A   User S/W int     -
SW_INT14  B    PHANTOM ; PM 1C   User S/W int     -
SW_INT15  B    PHANTOM ; PM 1E   User S/W int     -
SW_INT16  B    PHANTOM ; PM 20   User S/W int     -
TRAP      B    PHANTOM ; PM 22   Trap vector      -
NMI       B    PHANTOM ; PM 24   Non maskable Int   3
EMU_TRAP  B    PHANTOM ; PM 26   Emulator Trap    2
SW_INT20  B    PHANTOM ; PM 28   User S/W int     -
SW_INT21  B    PHANTOM ; PM 2A   User S/W int     -
SW_INT22  B    PHANTOM ; PM 2C   User S/W int     -
SW_INT23  B    PHANTOM ; PM 2E   User S/W int     -

;==============================================================================
; M A I N   C O D E  - starts here
;==============================================================================
;      .text
       .sect  ".b0pgm"
START:
       CLRC   SXM         ; Clear Sign Extension Mode
       CLRC   OVM         ; Reset Overflow Mode

* Set Data Page pointer to  page 1 of the peripheral frame
       LDP    #DP_PF1       ; Page DP_PF1 includes WET through EINT frames

* initialize WDT registers
       SPLK   #06Fh, WDCR   ; clear WDFLAG, Disable WDT, set for 1 sec overflow
       SPLK   #07h, RTICR   ; clear RTI Flag, set RTI for 1 second overflow

* configure PLL for 10MHz osc, 10MHz SYSCLK and 20MHz CPUCLK
       SPLK   #00B1h,CKCR1 ;CLKIN(OSC)=10MHz,CPUCLK=20MHz
       SPLK   #00C3h,CKCR0 ;CLKMD=PLL Enable,SYSCLK=CPUCLK/2,

* Clear reset flag bits in SYSSR (PORRST, PLLRST, ILLRST, SWRST, WDRST)
       LACL   SYSSR     ; ACCL <= SYSSR
       AND    #00FFh    ; Clear upper 8 bits of SYSSR
       SACL   SYSSR     ; Load new value into SYSSR

* initialize B2 RAM to zero's.
       LAR  AR1,#B2_SADDR  ; AR1 <= B2 start address
       MAR  *,AR1          ; use B2 start address for next indirect
       ZAC                 ; ACC <= 0
       RPT  #1fh           ; set repeat counter for 1fh+1=20h or 32 loops
       SACL *+             ; write zeros to B2 RAM

* initialize B1 RAM to zero's.
```

```
       LAR  AR1,#B1_SADDR  ; AR1 <= B2 start address
       MAR  *,AR1          ; use B2 start address for next indirect
       ZAC                 ; ACC <= 0
       RPT  #0ffh          ; set repeat counter for ffh+1=100h or 256 loops
       SACL *+             ; write zeros to B1 RAM

* Configure wait-state generator register, WSGR
       LDP  #0             ; set DP to page zero, includes B2 RAM
       SPLK #00h,070h      ; 70h <= 0000h
       OUT  70h,0ffffh     ; WSGR <= (070h). AVIS OFF; 0 ws for DS, IS & PS

;      B  INOUT
;      B  OUT
       B  IN

************************************************************
* Access external I/O space                                *
************************************************************
*
INOUT:
       LAR    AR1,#08701h    ; AR1 <= External Data RAM
       MAR    *,AR1

IS     SACL   *+
       IN     060h,8876h   ; Read I/O at 0x8876 and write to DARAM at 0x60
       OUT    060h,8877h   ; Read DARAM at 0x60 and write to I/O at 0x8877
       SACL   *-
       B      IS

OUT:
       LAR    AR1,#08701h    ; AR1 <= External Data RAM
       MAR    *,AR1

IS1    SACL   *+
       OUT    060h,8877h   ; Read DARAM at 0x60 and write to I/O at 0x8877
       SACL   *-
       B      IS1

IN:
       LAR    AR1,#08701h    ; AR1 <= External Data RAM
       MAR    *,AR1

IS2    LACL   *+
       IN     060h,8876h   ; Read I/O at 0x8876 and write to DARAM at 0x60
       LACL   *-
       B      IS2

       B      END            ; end testcase gracefully
**********************************************************
* Complete main body of code.                            *
**********************************************************
```

```
END    NOP                   ; Flush pipeline.
       NOP
       NOP
       NOP
       B      END

********************************************************************************
*                ISR's                                                         *
********************************************************************************
;===============================================================================
; I S R  -  PHANTOM
;
; Description:   ISR used to trap spurious interrupts.
;              Reads System Interrupt Vector register to capture
;              vector of module that caused the interrupt.
;
; Modifies:  Changes DP and ACC. Loads vector into 61h and BADh into 60h.
;
; Last Update:   10-17-96
;===============================================================================
PHANTOM
       LDP    #DP_PF1         ; go to peripheral file data page 1
       LACL   SYSIVR          ; ACC <= [SYSIVR]
       LDP    #0              ; go to first data page, with B2
       SACL   B2_SADDR+1      ; 61h <= [SYSIVR]
       SPLK   #0BADh,B2_SADDR ; 60h <= "BAD" value indicates error
       B      END             ; Terminate gracefully

.end
```

avis_chk.asm

```
;******************************************************************************
; File Name:  avis_chk.ASM  (File revision 0.1)
; Project:    x240 External Memory Interface Application Note.
; Originator: J.Crankshaw    (Texas Instruments)
;
; Target Sys: F240 EVM
;
; Description:
;
; Status:  Released.
;
; Last Update:   31 Dec 97
; _____________________________________________________________________________
; Date of Mod |         DESCRIPTION
; ------------|----------------------------------------------------------------
;      0.1     | create testcase
;            |
;            |
;            |
;*****************************************************************************

     .include "f240regs.h"
*

;-----------------------------------------------------------------------------
; Vector address declarations
;-----------------------------------------------------------------------------
         .sect  ".vectors"

RSVECT    B    START   ; PM 0    Reset Vector      1
INT1      B    PHANTOM ; PM 2    Int level 1       4
INT2      B    PHANTOM ; PM 4 Int level 2       5
INT3      B    PHANTOM ; PM 6 Int level 3       6
INT4      B    PHANTOM ; PM 8 Int level 4       7
INT5      B    PHANTOM ; PM A Int level 5       8
INT6      B    PHANTOM ; PM C Int level 6       9
RESERVED  B    PHANTOM ; PM E (Analysis Int)    10
SW_INT8   B    PHANTOM ; PM 10   User S/W int      -
SW_INT9   B    PHANTOM ; PM 12   User S/W int      -
SW_INT10  B    PHANTOM ; PM 14   User S/W int      -
SW_INT11  B    PHANTOM ; PM 16   User S/W int      -
SW_INT12  B    PHANTOM ; PM 18   User S/W int      -
SW_INT13  B    PHANTOM ; PM 1A   User S/W int      -
SW_INT14  B    PHANTOM ; PM 1C   User S/W int      -
SW_INT15  B    PHANTOM ; PM 1E   User S/W int      -
SW_INT16  B    PHANTOM ; PM 20   User S/W int      -
TRAP      B    PHANTOM ; PM 22   Trap vector       -
NMI       B    PHANTOM ; PM 24   Non maskable Int   3
EMU_TRAP  B    PHANTOM ; PM 26   Emulator Trap     2
SW_INT20  B    PHANTOM ; PM 28   User S/W int      -
SW_INT21  B    PHANTOM ; PM 2A   User S/W int      -
SW_INT22  B    PHANTOM ; PM 2C   User S/W int      -
SW_INT23  B    PHANTOM ; PM 2E   User S/W int      -

;==============================================================================
; M A I N   C O D E  - starts here
;==============================================================================
;      .text
       .sect  ".b0pgm"
START:
       CLRC   SXM         ; Clear Sign Extension Mode
       CLRC   OVM         ; Reset Overflow Mode

* Set Data Page pointer to  page 1 of the peripheral frame
       LDP    #DP_PF1       ; Page DP_PF1 includes WET through EINT frames

* initialize WDT registers
       SPLK #06Fh, WDCR     ; clear WDFLAG, Disable WDT, set for 1 sec overflow
       SPLK #07h, RTICR     ; clear RTI Flag, set RTI for 1 second overflow
```

```
* configure PLL for 10MHz osc, 10MHz SYSCLK and 20MHz CPUCLK
       SPLK  #00B1h,CKCR1 ;CLKIN(OSC)=10MHz,CPUCLK=20MHz
       SPLK  #00C3h,CKCR0 ;CLKMD=PLL Enable,SYSCLK=CPUCLK/2,

* Clear reset flag bits in SYSSR (PORRST, PLLRST, ILLRST, SWRST, WDRST)
       LACL  SYSSR     ; ACCL <= SYSSR
       AND   #00FFh    ; Clear upper 8 bits of SYSSR
       SACL  SYSSR     ; Load new value into SYSSR

* initialize B2 RAM to zero's.
       LAR  AR1,#B2_SADDR  ; AR1 <= B2 start address
       MAR  *,AR1          ; use B2 start address for next indirect
       ZAC                 ; ACC <= 0
       RPT  #1fh           ; set repeat counter for 1fh+1=20h or 32 loops
       SACL *+             ; write zeros to B2 RAM

* initialize B1 RAM to zero's.
       LAR  AR1,#B1_SADDR  ; AR1 <= B2 start address
       MAR  *,AR1          ; use B2 start address for next indirect
       ZAC                 ; ACC <= 0
       RPT  #0ffh          ; set repeat counter for ffh+1=100h or 256 loops
       SACL *+             ; write zeros to B1 RAM

* Call subroutines
;      CALL AVIS_B2
;      CALL AVIS_PER
;      CALL AVIS_EXT

       B     END             ; end testcase gracefully
**********************************************************
* Complete main body of code.                            *
**********************************************************

END    NOP                  ; Flush pipeline.
       NOP
       NOP
       NOP
       B     END

*******************************************************************************
*                Subroutines                                                  *
*******************************************************************************
;===========================================================================
; Routine Name: AVIS_B2      Routine Type: SR
;
; Description: This subroutine allows the scope to be used to
;            verify the switching of external address pins A0-3 when
;            AVIS is enabled. A sequence of reads and writes are performed
;            on internal B2 block of DARAM. This kernel
;            repeats continuously to view using the scope.
;
```

```
; Originator: J. Crankshaw
;
; Calling Convention:
;
;   Variables    on Entry     on Exit
; ----------------------------------------------------------------
; ----------------------------------------------------------------
; History:
; Last Update:    11-03-97
;===========================================================================
AVIS_B2:
* Load Address of wait-state generator register, WSGR, into AR1,
* and turn off address visibility mode, AVIS.
       LDP  #0             ; set DP to page zero, includes B2 RAM
       SPLK #08h,070h      ; 70h <= 0000h
       OUT  70h,0ffffh     ; WSGR <= (07h). AVIS ON; 0 ws for DS, IS & PS

       SPLK  #03h,071h     ;
       MAR   *,AR1

LOOP
       LACL  60h            ; read from B2 DARAM
       LACL  61h            ;
       RPT   71h            ; stop fetching for 4 cycles
       NOP
       SACL  64h            ; write to B2 DARAM
       SACL  67h            ;
       B     LOOP

       RET

AVIS_PER
* Load Address of wait-state generator register, WSGR, into AR1,
* and turn off address visability mode, AVIS.
       LDP  #0             ; set DP to page zero, includes B2 RAM
       SPLK #08h,070h      ; 70h <= 0000h
       OUT  70h,0ffffh     ; WSGR <= (07h). AVIS ON; 0 ws for DS, IS & PS

       LDP   #07080h/80h    ; use Data Page of first location in external ds
       MAR   *,AR1

LOOP1
       LACL  PADATDIR      ;
       SACL  PADATDIR
       SACL  PBDATDIR
       LACL  PBDATDIR
       B     LOOP1

       RET

AVIS_EXT
```

```
* Load Address of wait-state generator register, WSGR, into AR1,
* and turn off address visability mode, AVIS.
       LDP  #0              ; set DP to page zero, includes B2 RAM
       SPLK #08h,070h       ; 70h <= 0000h
       OUT  70h,0ffffh      ; WSGR <= (07h). AVIS ON; 0 ws for DS, IS & PS

       LDP    #08000h/80h    ; use Data Page of first location in external ds
       LAR    AR2,#PADATDIR  ; AR2 <= Port A DAT/DIR register
       MAR    *,AR1

LOOP2
       LACL   0              ; read from external memory
       RPT    #3
       NOP
       SACL   1              ; read from external memory
       B      LOOP2

       RET

********************************************************************************
*                ISR's                                                         *
********************************************************************************
;===============================================================================
; I S R  -  PHANTOM
;
; Description:   ISR used to trap spurious interrupts.
;              Reads System Interrupt Vector register to capture
;              vector of module that caused the interrupt.
;
; Modifies:   Changes DP and ACC. Loads vector into 61h and BADh into 60h.
;
; Last Update:   10-17-96
;===============================================================================
PHANTOM
       LDP    #DP_PF1         ; go to peripheral file data page 1
       LACL   SYSIVR          ; ACC <= [SYSIVR]
       LDP    #0              ; go to first data page, with B2
       SACL   B2_SADDR+1      ; 61h <= [SYSIVR]
       SPLK   #0BADh,B2_SADDR  ; 60h <= "BAD" value indicates error
       B      END             ; Terminate gracefully

.end
```

Debugger Command Files

f240evm.cmd

```
echo f240evm.CMD for F240 EVM

;Reset Memory Map
mr

;DATA MEMORY
ma 0x00000,1,0x0060,ram   ;MMRs
ma 0x00060,1,0x0020,ram   ;On-Chip RAM B2
ma 0x00200,1,0x0100,ram   ;On-Chip RAM B0 if CNF=0
ma 0x00300,1,0x0100,ram   ;On-Chip RAM B1
ma 0x07010,1,0x0010,ioport   ;Peripheral - System Config & Control
ma 0x07020,1,0x0010,ioport   ;Peripheral - WDT / RTI
ma 0x07030,1,0x0010,ioport   ;Peripheral - ADC
ma 0x07040,1,0x0010,ioport   ;Peripheral - SPI
ma 0x07050,1,0x0010,ioport   ;Peripheral - SCI
ma 0x07070,1,0x0010,ioport   ;Peripheral - Ext Ints
ma 0x07090,1,0x0010,ioport   ;Peripheral - Digital I/O
ma 0x07400,1,0x000D,ioport   ;Peripheral - Event Mgr GPT
ma 0x07411,1,0x000C,ioport   ;Peripheral - Event Mgr CMP,PWM
ma 0x07420,1,0x0007,ioport   ;Peripheral - Event Mgr CAP,QEP
ma 0x0742C,1,0x0009,ioport   ;Peripheral - Event Mgr Int Cntl

; for EVM, external RAM is in memory map
ma 0x08000,1,0x08000,ram  ;Ext SRAM

;PROGRAM MEMORY
;ma 0x00000,0,0x04000, ram ;Internal Prog mem (Flash EEPROM or ROM)
                          ;Available if MPNMC=0.
ma 0x00000,0,0x04000,RAM   ;Internal Program memory - FLASH
ma 0x04000,0,0x0BE00,RAM   ;External Program memory - SRAM
ma 0x0FE00,0,0x00100,RAM   ;Available if CNF=1 i.e. B0

;I/O MEMORY
;~~~~~~~~~~
ma 0x0000,2,0x0008,WOM     ;I/O Memory Mapped DAC Registers
ma 0x0008,2,0x0004,ROM     ;I/O Memory Mapped DIP Switches
ma 0x000C,2,0x0004,WOM     ;I/O Memory Mapped LEDs

mem  0x0200
mem1 0x0300
mem2 0x0060

wa (ST0&(0x03ff))>>9,INTM,d   ;INTM Int Mode Bit
wa ((ST0>>13)&0x07),ARP,d  ;INTM Int Mode Bit
wa (ST0&0x01ff),DP,d
```

```
; External Memory Interface application note programs
;-----------------------------------------------------
;load c:\dspcode\x240\f240evm\xmif_app\xi_fetch.out
;take c:\dspcode\x240\f240evm\xmif_app\xidstime.cmd
;take c:\dspcode\x240\f240evm\xmif_app\xi_rrw.cmd
;take c:\dspcode\x240\f240evm\xmif_app\avis_chk.cmd
;take c:\dspcode\x240\f240evm\xmif_app\xi_table.cmd
;take c:\dspcode\x240\f240evm\xmif_app\xi_inout.cmd

echo   f240evm.CMD HAS BEEN LOADED
```

xidstime.cmd

```
e st1 |= 0x1000

;-----------------;
; load testcase   ;
;-----------------;

load c:\dspcode\x240\f240evm\xmif_app\xidstime.out

?pc = 0xfe00

alias redo,"e st1 |= 0x1000;?pc = 0xfe00"
```

xi_table.cmd

```
e st1 |= 0x1000

;-----------------;
; load testcase   ;
;-----------------;

load c:\dspcode\x240\f240evm\xmif_app\xi_table.out

?pc = 0xfe00

alias redo,"e st1 |= 0x1000;?pc = 0xfe00"
```

xi_inout.cmd

```
e st1 |= 0x1000

;-----------------;
; load testcase   ;
;-----------------;

load c:\dspcode\x240\f240evm\xmif_app\xi_inout.out
```

```
?pc = 0xfe00

alias redo,"e st1 |= 0x1000;?pc = 0xfe00"
```

avis_chk.cmd

```
e st1 |= 0x1000

;-----------------;
; load testcase   ;
;-----------------;

load c:\dspcode\x240\f240evm\xmif_app\avis_chk.out

?pc = 0xfe00

alias redo,"e st1 |= 0x1000;?pc = 0xfe00"
```