Application Report
SPRA531

TMS320C54x Interface with SDRAM

Vivian Shao/Soon Chye *C5000*

Abstract

This application report provides a comprehensive guide into the design of the hardware interface between the Texas Instruments (TI™) TMS320C54x digital signal processor (DSP) and the TMS626812A 2MX8 synchronous dynamic random-access memory (SDRAM) using field programmable gate arrays (FPGA). A test algorithm is also written, in assembly, to test the hardware design and the SDRAM. This design can be applied to systems that need external memory as a data buffer, for example, a digital still camera.

Contents

Figures

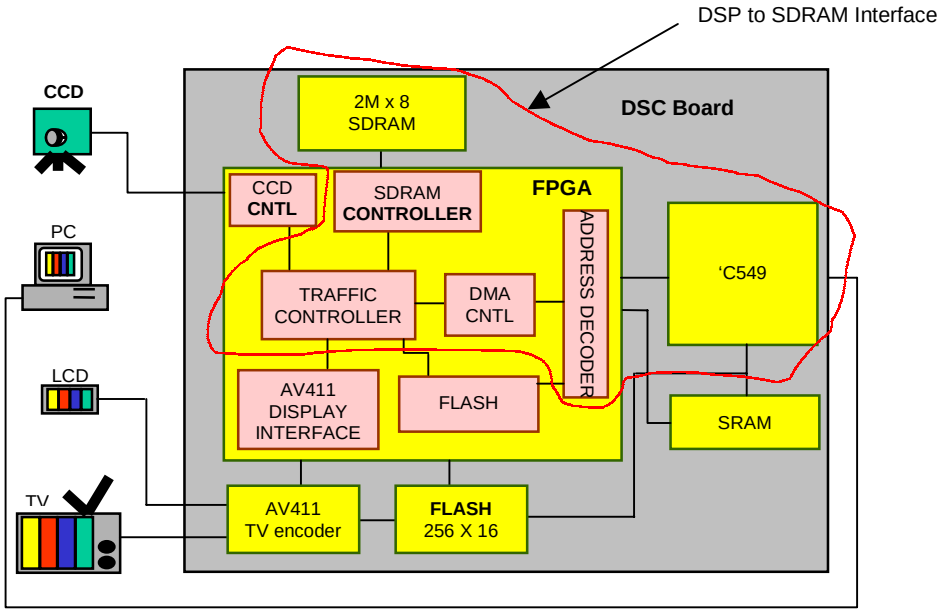
Tables

Introduction

In the development of the Digital Still Camera Demonstration Board, (referred as Project) a 2 Mbytes SDRAM is used as a temporary storage space for raw video data from the camera as well as manipulated video data to the display devices such as an LCD or television. Since the DSP cannot be directly interfaced with the SDRAM, an FPGA is used as the hardware interface to provide the necessary control signals and timing requirements so that the DSP can write data to and read data from the SDRAM. This application report focuses on the design and testing of the interface between the DSP and SDRAM, which is highlighted in Figure 1.

This application report consists of two sections. The first section discusses the hardware design, the use of the FPGA to provide the timing and controls to access SDRAM. The second section describes the testing algorithm used by the DSP to test the FPGA and the SDRAM.

Figure 1. Digital Still Camera Demonstration Board System Block Diagram



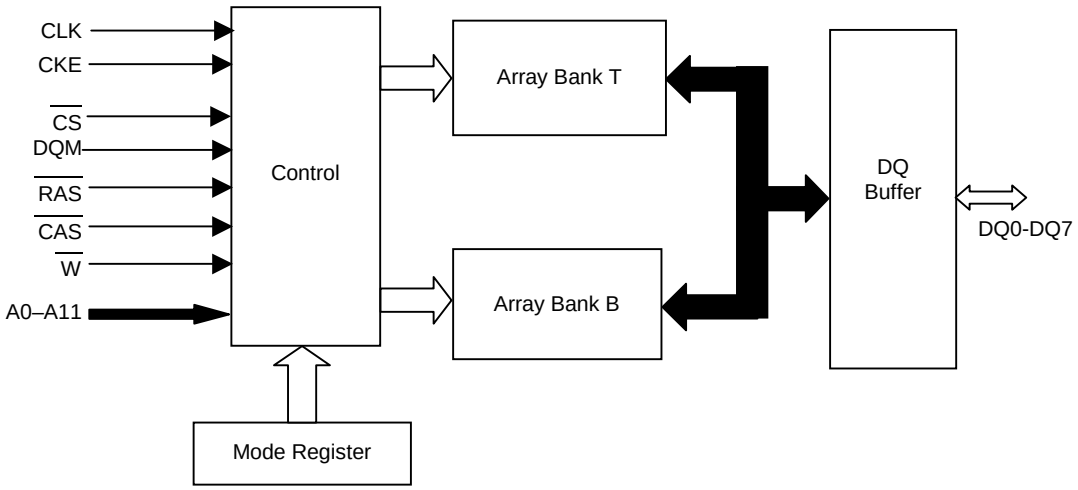
Hardware Interface

The key issue in the design of the hardware interface between the DSP and SDRAM is timing. To design the SDRAM interface, the first step is to understand the SDRAM commands and timing requirements. Due to the complexity of the timing required, this interface is designed using FPGA. In the following sections, SDRAM timing and commands are introduced first, followed by the design of the interface between the SDRAM and ‘C549.

SDRAM Timing and Command

The SDRAM used in this application report is the TMS626812A. The TMS626812A is a high-speed, 16 777 216-bit SDRAM device organized as two banks of 1 048 576 words with 8-bits per word. The SDRAM employs state-of-the art technology for high performance, reliability, and low power. All inputs and outputs are synchronized with the system clock (CLK) input to simplify system design and enhance the use with high-speed microprocessors and caches. The functional block diagram is shown in Figure 2.

Figure 2. SDRAM Functional Block Diagram



All inputs to the ‘626812A SDRAM are latched on the rising edge of the synchronous system clock (CLK). The outputs, DQx, also are referenced to the rising edge of CLK. The ‘626812A has two banks that are accessed independently. A bank must be activated before it can be accessed (read from or written to). Refresh cycles refresh both banks alternately.

Six basic commands or functions control most operations of the ‘626812A:

- Bank activate/row-address entry
- Column-address entry/write operation
- Column-address entry/read operation
- Bank deactivate

- Auto-refresh
- Self-refresh

The basic command functional table is shown in Table 1.

Table 1. SDRAM Basic Command Functional Table

Command	State of Bank(S)	$\overline{CS}$	$\overline{RAS}$	$\overline{CAS}$	$\overline{W}$	A11	A10	A0-A9	Mnemonic
Mode Register Set	T = deac	L	L	L	L	X	X	A0-A6, A9 = V	MRS
	B = deac							A7-A8 = L	
Bank Deactivate (precharge)	X	L	L	H	L	BS	L	X	DEAC
Deactivate all banks	X	L	L	H	L	X	H	X	DCAB
Bank activate/row-address entry	SB = deac	L	H	L	L	BS	V	V	ACTV
Column-address entry/write operation	SB = actv	L	H	L	L	BS	L	V	WRT
Column-address entry/write operation with automatic deactivate	SB = actv	L	H	L	L	BS	H	V	WRT-P
Column-address entry/read operation	SB = actv	L	H	L	H	BS	L	V	READ
Column-address entry/read operation with automatic deactivate	SB = actv	L	H	L	H	BS	H	V	READ-P
No operation	X	L	H	H	H	X	X	X	NOOP
Control-input inhibit/no operation	X	H	X	X	X	X	X	X	DESL
Auto-refresh	T = deac	L	L	L	H	X	X	X	REFR
	B = deac								

The commands used in this application report for interfacing SDRAM with ‘C54x are MRS, DEAC, ACTV, WRT-P, READ-P, and REFR. The goal of the design is to generate the control signals to meet the timing requirement of these commands. See the TMS626812A data sheet for the operation timing.

Interface Between SDRAM and TMS320C54x DSP

Figure 3 shows the general block diagram of the interface between the DSP and the SDRAM. The SDRAM is configured to read or write data in bursts of 128 bytes; whereas, the DSP reads or writes data one byte at a time. Due to these differences, two buffers (B0 and B1) are required to store the data. Buffers B0 and B1 are each RAM of 128 bytes. Data transfers to and from the SDRAM through B0 and B1. The two buffers are mapped to the same address location FF00h-FF80h in DSP data space and they are mutually exclusive meaning that the DSP cannot access both B0 and B1 at the same time.

In fact, when B0 is access by the DSP, B1 is access by the SDRAM; and, when B1 is accessed by the DSP, B0 is accessed by the SDRAM. Thus, while the DSP is writing data to B0, the SDRAM is reading data from B1. Alternatively, while the DSP is reading data from B0, the SDRAM is writing data to B1. These events are listed in Table 2.

Figure 3. SDRAM Interface with TMS320C54x Block Diagram

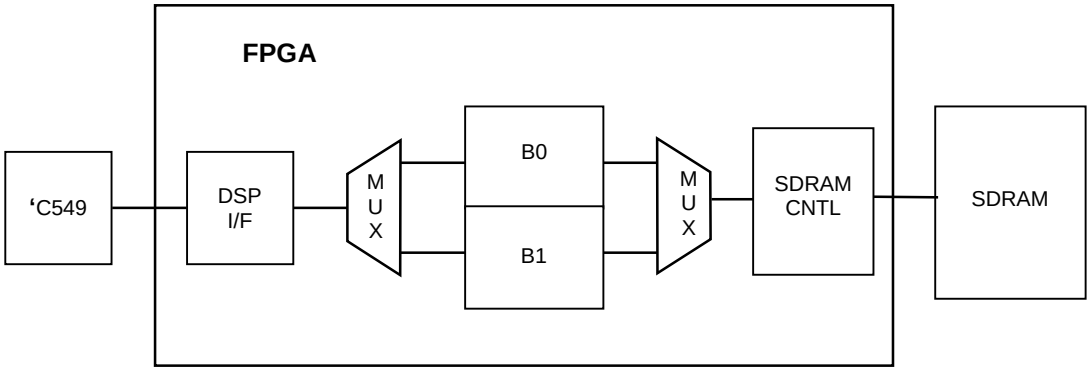


Table 2. Data Transfer Events

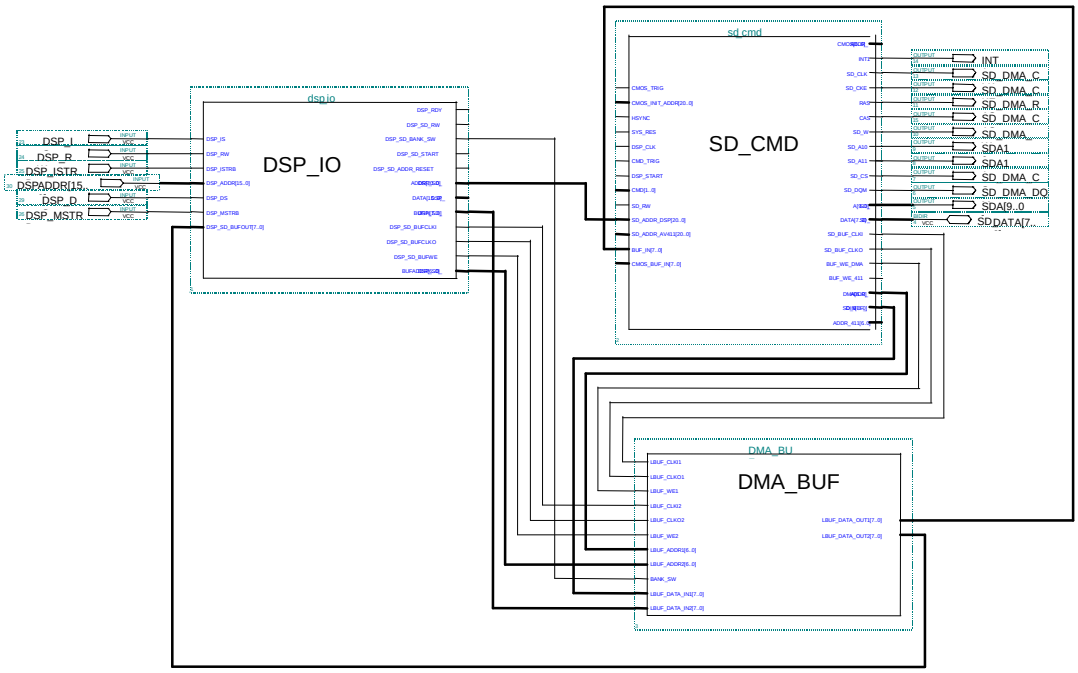
Event			Errors
DSP writes to B0	---	SDRAM reads from B1	No
DSP writes to B1	---	SDRAM reads from B0	No
DSP reads from B0	---	SDRAM writes to B1	No
DSP reads from B1	---	SDRAM writes to B0	No
DSP writes to B0	---	SDRAM writes to B1	Yes
DSP writes to B1	---	SDRAM writes to B0	Yes
DSP reads from B0	---	SDRAM reads from B1	Yes
DSP reads from B1	---	SDRAM reads from B0	Yes

Certain events, for example, both the SDRAM and DSP writing to the buffers at the same time or both the SDRAM and DSP reading from the buffers at the same time results in an error. This method of data transfer achieves two major benefits: speeding up 'C54x accessing time of burst data and solving the problem of different clocking rates between 'C549 and SDRAM.

Hardware Design in FPGA

The 'C54x provides the following signals for external memory control: CLK, CS, A0-A15, D0-D15, RW, MSTRB, ISTRB, and IS, but the SDRAM accepts the following control signals: CLK, CKE, CS, DQM, W, RAS, CAS, and A0-A11. Due to the differences in control signals, there is a need to interface these two devices with some hardware control logic to generate SDRAM control signals from the command of the DSP. The complexity of this data transfer interface warrants the need to use FPGA. Observing Figure 3, the DSP I/F (interface) unit and the SDRAM CNTL (control) unit are designed to receive SDRAM accessing commands from the 'C54x and then generate control signals to the SDRAM. Figure 4 shows the top-level view of an FPGA design.

Figure 4. Hardware Interface Design in FPGA-Top Level



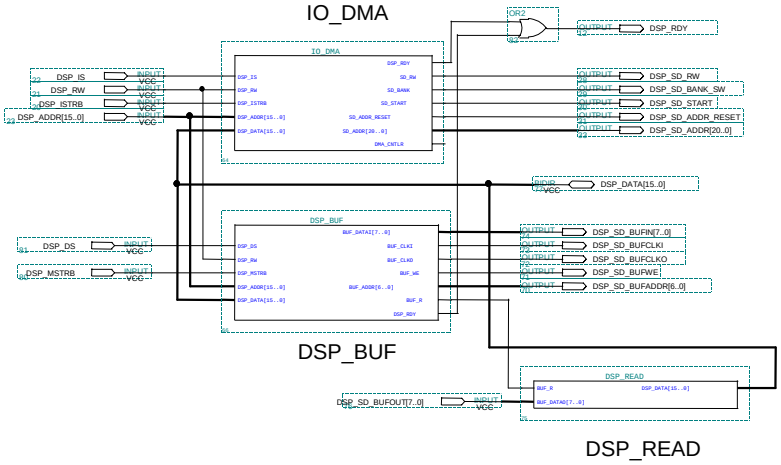
The SDRAM is dynamic RAM; therefore, it requires refreshing to prevent data loss. The refresh circuit is designed into the FPGA, but is not shown in this section because it is incorporated in another circuit not related to the SDRAM portion. The refresh cycle is hardware controlled.

There are three main blocks: DSP_IO is the interface with the ‘C54x that decodes the SDRAM address and SDRAM commands, DMA_BUF is the B0 and B1 buffers, SD_CMD generates control signals of the SDRAM. These three blocks are described in more detail in the following sections.

DSP_IO

The components in DSP_IO include: IO_DMA, DSP_BUF, and DSP_READ. Figure 5 shows the interconnection of these three components, IO_DMA generates the SDRAM command signals; DSP_BUF generates B0 and B1 address, data, and control signals; and DSP_READ controls the DSP read and write direction.

Figure 5. Detailed Design of DSP_IO (DSP_IO.GDF)



Three I/O-mapped registers (address) from the DSP are assigned for SDRAM control, as listed in Table 3. The bit definition for the DMA_CTL register is listed in Table 4.

Table 3. I/O-Mapped Control Registers

I/O Address	Definition	Description
0000h	SW_RESET	Clear to 0 and then set to 1 to restart DSP
0003h	DMA_ADDH	SDRAM absolute address A20-A16
0004h	DMA_ADDL	SDRAM absolute address A15-A0
0005h	DMA_CTL	SDRAM read and write control. See Table 4 for a description of each bit.

Table 4. DMA_CTL Register Bit Description

Bit	Definition	Description
0	R/W_	0: DSP write mode; SDRAM read mode
		1: DSP read mode; SDRAM write mode
1	B0/B1	0: DSP uses B0, SDRAM uses B1
		1: DSP uses B1, SDRAM uses B0
2	START	0: stop SDRAM R/W operation
		1: start SDRAM R/W operation
3	Polling	Reserved
4	SD_ADDR_RESET	1: set SDRAM burst start address
		0: no operation

The number of addressable memory locations in the SDRAM is from 00000h-FFFFFh, which is 2 Mbytes; however, the ‘C54x only has a 16-bit address line for data space. Therefore, the ‘C54x has to send addresses to the SDRAM in two steps: send DMA_ADDH followed by DMA_ADDL.

The DMA_CTL register is used by the ‘C54x to control hardware operations. Bit 0 sets the SDRAM read/write (R/W) mode. Bit 1 switches B0 and B1, that is, defines which buffer is used by the ‘C54x and which buffer is used by the SDRAM. Bit 2 starts a burst of data, 128 words, reading from or writing to SDRAM through the B0 or B1 buffer. To set a new address of SDRAM, bit 4 is set to 1 before setting DMA_ADDH and DMA_ADDL.

DMA_BUF (B0 and B1)

Two buffers, B0 and B1, are used for data transmission. The input and output signals for each buffer include CLKI, CLKO, WE, ADDR[6-0], DATA_IN[7-0], and DATA_OUT[7-0]. BANK_SW, an input signal of DMA_BUF, switches the signal connection between B0 and B1 with the ‘C549 and SDRAM.

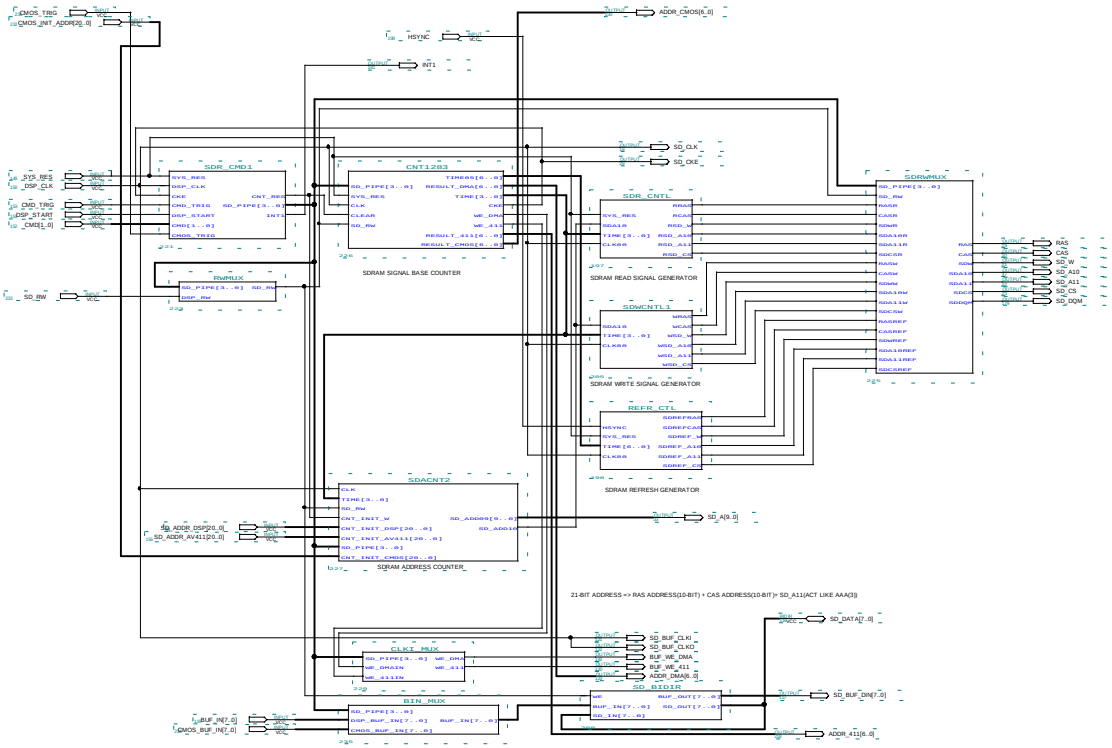
The VHDL file, RAM_BUF.VHD, is attached in Appendix A.

SD_CMD

Figure 6 shows the detailed design of SD_CMD. Three SDRAM functions are implemented here: refresh, read, and write. When the ‘C54x sets the READ command, 128-byte data is read from the SDRAM and stored to the B0 or B1 buffer. When the ‘C54x sets the WRITE command, 128-byte data is transferred from the B0 or B1 buffer and written to the SDRAM.

The VHDL files of BIN_MUX.VHD, CLKI_MUX.VHD, SDACNT2.VHD, CNT1283.VHD, SDRWMUX.VHD, RWMUX.VHD, SDR_CMD1.VHD, SDWCNTL1.VHD, SD_BIDIR.VHD, REFR_CTL.VHD, and SDR_CNTL.VHD are attached in Appendix A.

Figure 6. Detailed Design of SD_CMD (SD_CMD.GDF)



TMS320C549 Software

Data transfer between the DSP and the SDRAM has to go through the buffers B0 and B1; therefore, the DSP must notify the FPGA where to store the data in the SDRAM or where to extract the data from the SDRAM. To accomplish this task, two dedicated I/O addresses are used by the DSP to transfer the address of the SDRAM data location to the FPGA since there are 2 Mbytes of addressable space. These two I/O addresses are 03h (DMA_ADDH) and 04h (DMA_ADDL). In addition, another I/O address, 05h (DMA_CTL), is used by the DSP to send commands to the FPGA to generate the necessary control signals for controlling the SDRAM.

To perform data transfers, the following sequence must be used for writing to the SDRAM.

1) The data must first be placed into either buffer B0 or B1.

2) The address of the SDRAM must be sent to FPGA via I/O addresses DMA_ADDH and DMA_ADDL.

3) The DSP sends a command to FPGA (I/O address DMA_CTL) to generate the necessary control signals to enable the SDRAM to read the data from buffer B0 or B1.

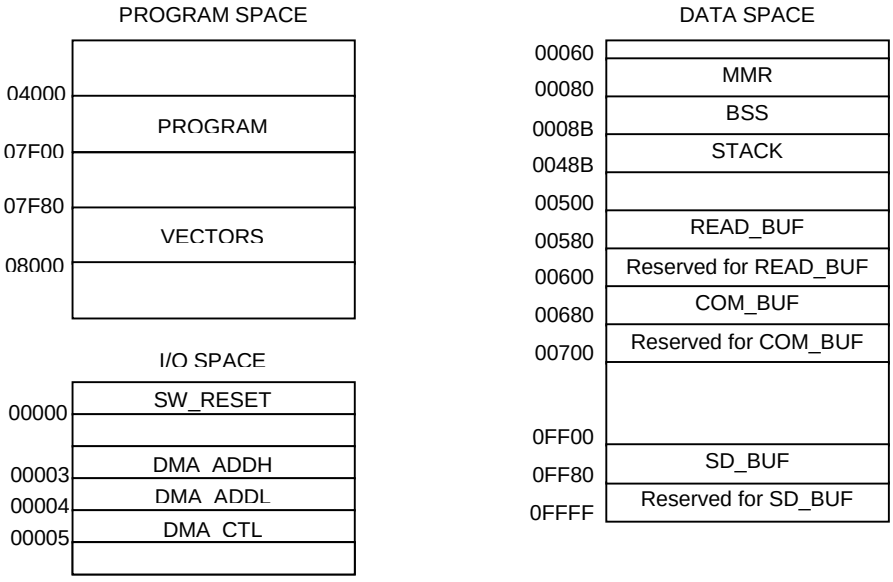
The following sequence must be used for reading from the SDRAM.

1) The DSP sends the address.
2) The DSP sends a command to SDRAM to begin writing data to buffer B0 or B1.
3) The DSP reads from buffer B0 or B1.

Memory Map of the DSP

Figure 7 shows the memory map of the DSP. Note that this memory map only shows the program, data, and I/O spaces used for data transfers between the SDRAM and the DSP, and not the memory spaces used for the entire project.

Figure 7. Memory Map of Program, Data, and I/O Space



Program Space

- PROGRAM – code for the test algorithm is stored here
- VECTORS – contains interrupt branch routines

Data Space

- MMR – contains the memory-mapped registers
- BSS – contains all the data variables
- STACK – software stack
- READ_BUF – contains data values to be sent to SDRAM and also used as reference for verification of data read back from SDRAM.

- COM_BUF – contains the data read back by DSP from SDRAM
- SD_BUF – maps to buffer B0 and B1 of FPGA. The gateway for data transfer to and from SDRAM

I/O Space

- SW_RESET – contains the commands to perform a software reset to the project
- DMA_ADDH – contains the higher bits of the SDRAM address bus (A20-A16)
- DMA_ADDL – contains the lower bits of the SDRAM address bus (A15-A0)
- DMA_CTL – contains the commands to perform SDRAM read/write, etc.

Software Flow Diagram

The basic flow of the program is discussed in detail rather then the algorithm. The algorithm for testing the SDRAM is provided in Appendix B. Note that this algorithm is specifically written to test the SDRAM used in this project and is not meant to be a general testing algorithm. Basically, the testing algorithm is divided into two main sections. The first section shows the code for writing data to all the available locations of the SDRAM. In this project, a running number sequence (01h to 80h) totaling 128 bytes is used as a test pattern. This test pattern tests the physical data lines from the DSP to the FPGA to the SDRAM. In addition, this number sequence is able to find errors due to timing deviation during the transfer, as shown in Figure 8.

Figure 8. Errors Seen during Testing

Timing is not Deviated: 1,2,3,4,5,…….3a,3b,3c,3d,3e,3f,…….,7a,7b,7c,7d,7e,7f

Timing is Deviated: 1,2,3,4,5.,……3a,**3a,3a**,3d,3e,3f,……..,7a,**7a**,7c,7d,**75**,7f

Figure 8 is only one type of error that can occur; other types of errors are not shown here. In the second section, the data written to the SDRAM is read back by the DSP and compared with a set of reference data. The Error Counter is incremented if an error is encountered. The flow diagram is shown in Figure 9.

Figure 9. Flow Diagram

DSP writes data to SDRAM

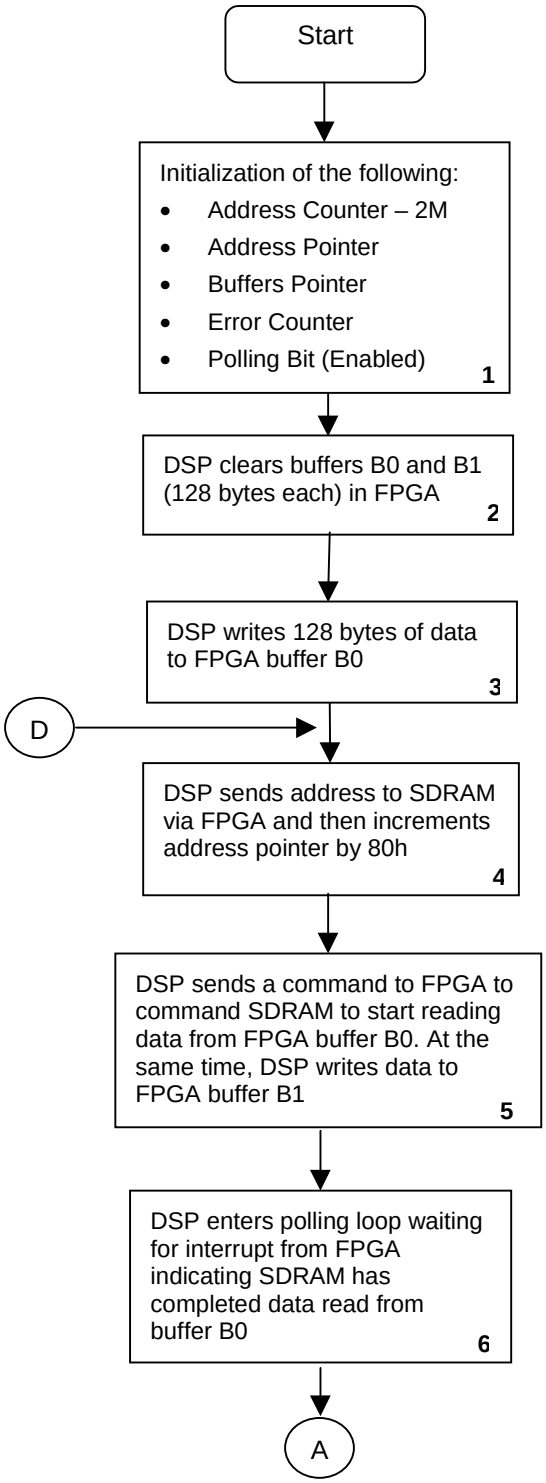


Figure 9. Flow Diagram (Continued)

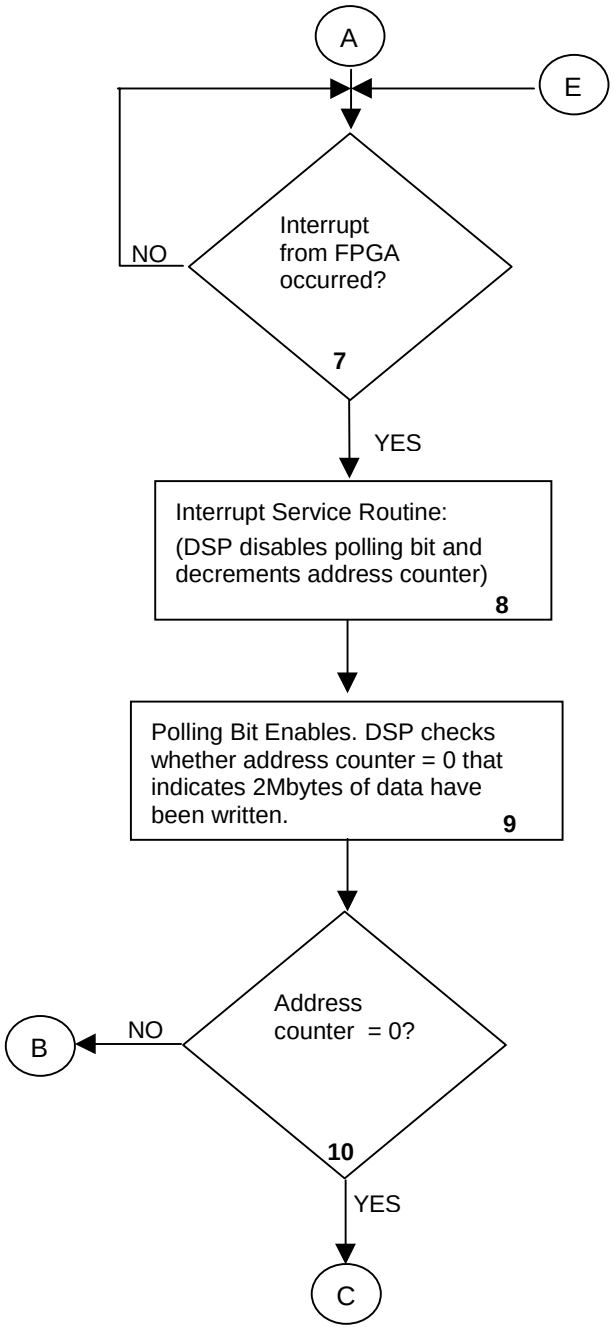


Figure 9. Flow Diagram (Continued)

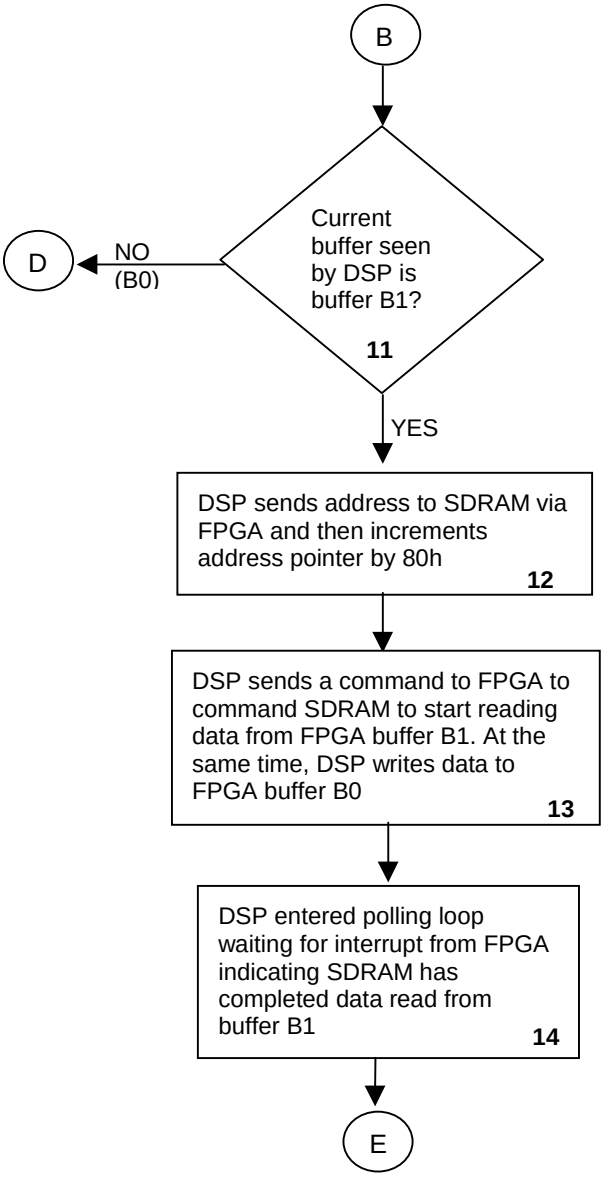


Figure 9. Flow Diagram (Continued)

DSP reads data from SDRAM

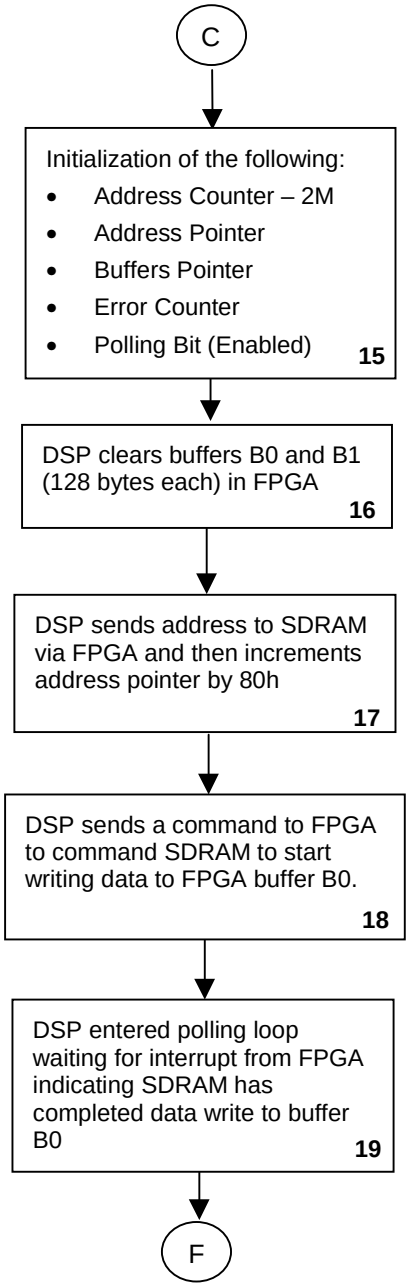


Figure 9. Flow Diagram (Continued)

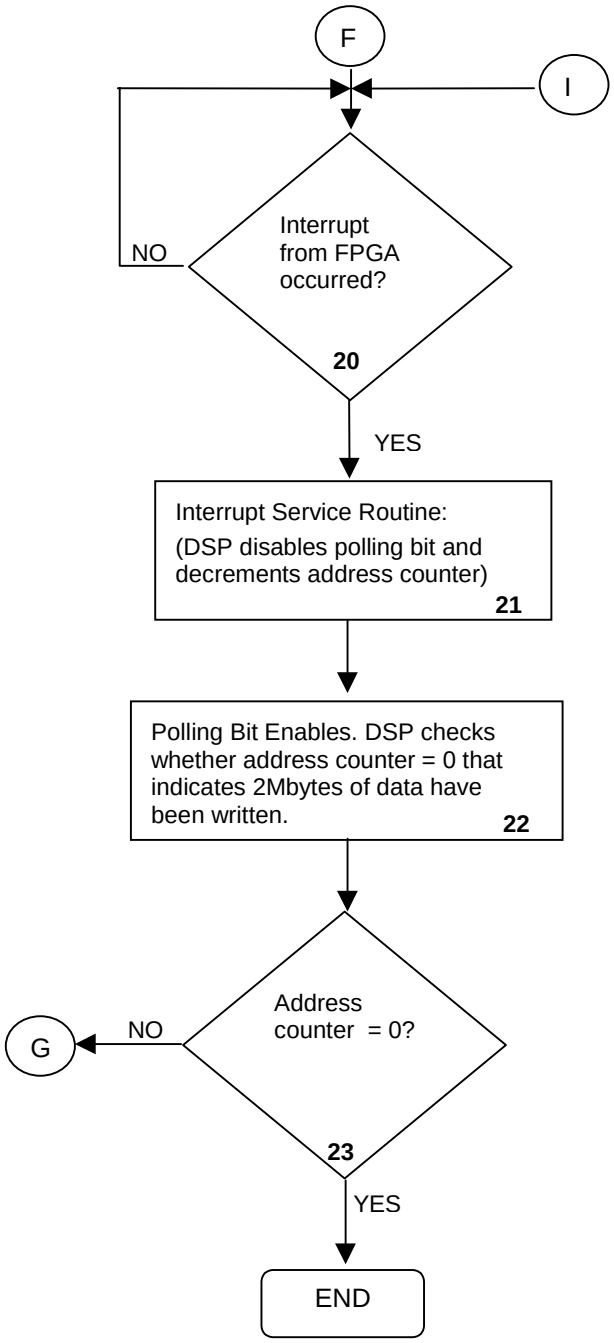


Figure 9. Flow Diagram (Continued)

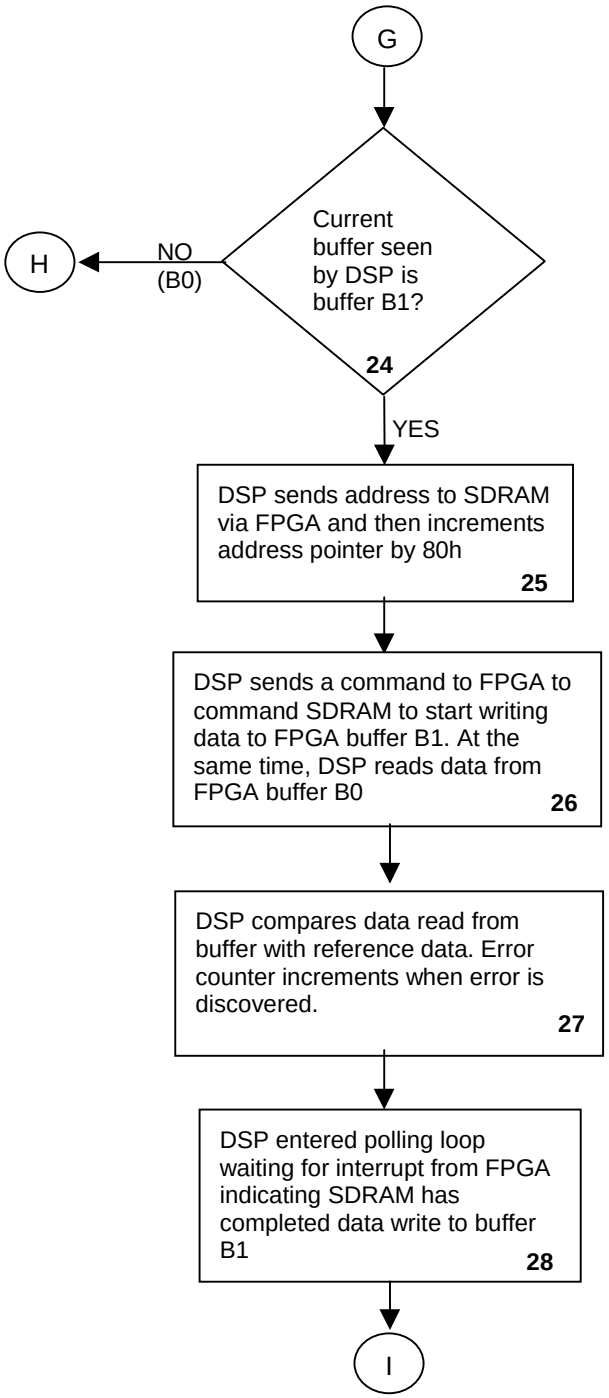
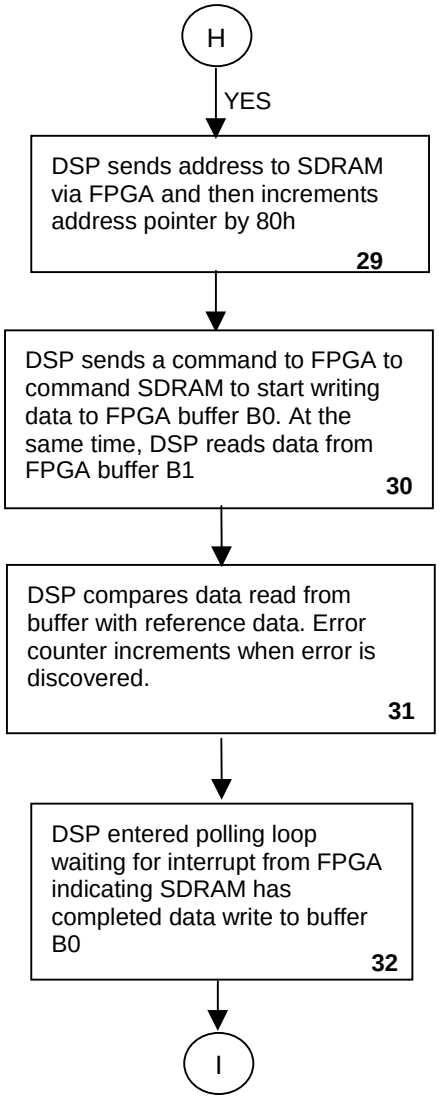


Figure 9. Flow Diagram (Continued)



As shown in Figure 9, the flow of the codes is in sequence from 1–32. The following details are in accordance to the sequence.

Upon power up, the DSP initializes (#1) the following:

- Address Counter – holds the count value for testing 2 Mbytes of SDRAM.
- Address Pointer – holds the address to be sent to the SDRAM.
- Buffers Pointers – point to the reference data buffer and location FF00h that is mapped to the FPGA buffer.
- Error Counter – holds the numbers of errors found.
- Polling Bit – held the DSP in a polling loop
- Stores the reference data into the reference buffer. Performed by subroutine REF_DATA

Once initialization is complete, the DSP clears the two buffers B0 and B1 (#2) in the FPGA. To differentiate the two buffers, two clear values are used; for buffer B0, 00h is used and for buffer B1, FFh is used. Note that these two values used are not within the test sequence. Since both buffers are mutually exclusive, a command must be sent to the FPGA to switch to the correct buffer. These commands are shown in the subroutine DSPRB1 (DSP see B0 and SDRAM see B1) and DSPRB0 (DSP see B1 and SDRAM see B0). These commands are also present in the following subroutines: DSP_WB0W, DSP_WB0, DSP_WB1W, DSP_WB1, DSP_RB0, and DSP_RB1. The difference between these two groups is that the first group (DSPRB0 and DSPRB1) does not contain commands to activate the SDRAM.

After clearing the buffers, the DSP starts to write the first 128 bytes of the test sequence into buffer B0 (#3). This is performed by the subroutine WRITE_BUF. After writing the data, the DSP sends the address to the FPGA (#4) and then increments the address pointer by 80h (128 bytes). The DSP then sends a command to start the SDRAM reading the data from buffer B0 (#5). Once the command is sent, the DSP starts writing the test sequence into buffer B1. All these steps are performed by the subroutine DSP_WB0. When buffer B1 is fully written, the DSP enters into a polling loop (#6) waiting for SDRAM to complete the read sequence at buffer B0. When the SDRAM completes reading 128 bytes, the FPGA sends an interrupt (INT1) to the DSP (#7).

When the DSP receives this interrupt, the DSP enters into an ISR (interrupt service routine). In this ISR (#8), the polling bit is disabled and the address counter is decremented by 1. A single decrement indicates 128 bytes of data have been transferred. When returned from the ISR, the polling bit is enabled and the address counter is checked (equal to 0) to determine if 2 Mbytes of data have been written (#9). If the address counter is not 0, the DSP checks whether the buffer presently seen by it is B0 or B1. At this time, the DSP should see buffer B1 and so the DSP sends the address for the next block of data to FPGA and increments the address pointer (#12). The DSP then sends a command to start the SDRAM reading the data from buffer B1 (#13). Once the command is sent, the DSP starts writing the data to buffer B0. The writing process continues until the address counter reaches zero.

When the address counter reaches zero, indicating 2 Mbytes of data have been written, the DSP proceeds to the second portion of the routine that is to read back the data and perform a data comparison (#10).

Before the DSP starts reading data back from the SDRAM, it reinitializes all the necessary pointers and counters (#15). Once initialization is complete, the DSP again clears the two buffers B0 and B1 (#16). This prevents the DSP from reading back data that is inside the FPGA buffer during the write phase. The DSP sends the address of the data it wants to read back to the SDRAM (#17) and then increments the address pointer. The DSP then sends a command to start the SDRAM writing data into FPGA buffer B0 (#18). The DSP enters into a polling loop (#19) waiting for the SDRAM to complete writing data into buffer B0. When the SDRAM completes writing data to buffer B0, FPGA sends an interrupt (INT1) to the DSP (#20).

When the DSP receives this interrupt, the DSP enters into an ISR. In this ISR (#21), the polling bit is disabled and the address counter is decremented by 1. When returned from the ISR, the polling bit is enabled and the address counter is checked (equal to 0) to determine if 2 Mbytes of data have been written (#22). If the address counter is not 0, the DSP checks whether the buffer presently seen by it is B0 or B1 (#23). At this time, the DSP should see buffer B1 and so the DSP sends the address for the next block of data to SDRAM and increments the address pointer (#25). The DSP then sends a command to start the SDRAM writing the data to buffer B1 (#26). Once the command is sent, the DSP starts reading the data from buffer B0. After reading the data, the DSP compares the data with the reference data stored. When there is a mismatch, the error counter is incremented (#27). The DSP enters into a polling loop (#28) waiting for the SDRAM to complete writing data into buffer B1. When the SDRAM completes writing data to buffer B1, FPGA sends an interrupt (INT1) to the DSP (#20).

This read process continues until the address counter is 0. When the address counter is 0, the test routine terminates and you can check the error counter for any errors that have occurred.

Conclusion

This application report provides a detailed look into the design of a hardware interface between a DSP and SDRAM using FPGA. In addition, a test algorithm is provided to test the hardware design and at the same time shows how data is transferred to and from the SDRAM.

References

1) TMS320C54x Reference Set Volume 1, CPU and Peripherals (SPRU131)
http://www.ti.com/sc/docs/dsps/literatu.htm

2) TMS320C54x Reference Set Volume 2, Mnemonic Instruction Set (SPRU172)
http://www.ti.com/sc/docs/dsps/literatu.htm

3) TMS320C54x Assembly Language Tools (SPRU102)
http://www.ti.com/sc/docs/dsps/literatu.htm

4) TMS626812A Synchronous Dynamic Random-Access Memories data sheet
http://www.ti.com/sc/docs/memory/sdram/16mb.htm

5) ALTERA FPGA user's guide
http://www.altera.com

Appendix A FPGA VHDL Files

DMA_BUF.VHD

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
LIBRARY lpm;
USE lpm.lpm_components.ALL;
-- LIBRARY work;
-- USE work.ram_constants.ALL;

ENTITY DMA_BUF IS
  GENERIC( ADDR_WIDTH : INTEGER := 7 );
    PORT(
  LBUF_CLKI1, LBUF_CLKO1, LBUF_WE1 : IN STD_LOGIC;
  LBUF_CLKI2,LBUF_CLKO2, LBUF_WE2  : IN STD_LOGIC;
  LBUF_ADDR1,LBUF_ADDR2  : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
  BANK_SW : IN STD_LOGIC;
  LBUF_DATA_IN1,LBUF_DATA_IN2  : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  LBUF_DATA_OUT1,LBUF_DATA_OUT2 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
  );
END DMA_BUF;

ARCHITECTURE example OF DMA_BUF IS
  SIGNAL DATA1,DATA2     : STD_LOGIC_VECTOR (7 DOWNTO 0);
     SIGNAL ADDRESS1,ADDRESS2 : STD_LOGIC_VECTOR (ADDR_WIDTH-1 DOWNTO 0);
     SIGNAL WE1,WE2,INCLOCK1,INCLOCK2,OUTCLOCK1,OUTCLOCK2 : STD_LOGIC;
     SIGNAL Q1,Q2 : STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN

  OUTCLOCK1 <= LBUF_CLKO1 WHEN ( BANK_SW = '0') ELSE LBUF_CLKO2;
  INCLOCK1 <= LBUF_CLKI1 WHEN ( BANK_SW = '0') ELSE LBUF_CLKI2;
  WE1 <= LBUF_WE1 WHEN ( BANK_SW = '0') ELSE LBUF_WE2;
  ADDRESS1 <= LBUF_ADDR1 WHEN(BANK_SW='0') ELSE LBUF_ADDR2;

  OUTCLOCK2 <= LBUF_CLKO1 WHEN ( BANK_SW = '1') ELSE LBUF_CLKO2;
  INCLOCK2 <= LBUF_CLKI1 WHEN ( BANK_SW = '1') ELSE LBUF_CLKI2;
  WE2 <= LBUF_WE1 WHEN ( BANK_SW = '1') ELSE LBUF_WE2;
  ADDRESS2 <= LBUF_ADDR1 WHEN(BANK_SW='1') ELSE LBUF_ADDR2;

  LBUF_DATA_OUT1 <= Q1 WHEN (BANK_SW='0') ELSE Q2;
  LBUF_DATA_OUT2 <= Q1 WHEN (BANK_SW='1') ELSE Q2;
  DATA1 <= LBUF_DATA_IN1 WHEN (BANK_SW='0') ELSE LBUF_DATA_IN2;
  DATA2 <= LBUF_DATA_IN1 WHEN (BANK_SW='1') ELSE LBUF_DATA_IN2;

PING_RAM: lpm_ram_dq
      GENERIC MAP (lpm_widthad => ADDR_WIDTH, lpm_width => 8)
      PORT MAP (data => DATA1, address => ADDRESS1, we => WE1,
         inclock => INCLOCK1, outclock => OUTCLOCK1,  q => Q1);
PONG_RAM: lpm_ram_dq
      GENERIC MAP (lpm_widthad => ADDR_WIDTH, lpm_width => 8)
      PORT MAP (data => DATA2, address => ADDRESS2, we => WE2,
         inclock => INCLOCK2, outclock => OUTCLOCK2,  q => Q2);
END example;
```

DSP_BUF.VHD

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;


ENTITY DSP_BUF IS
  PORT(
    DSP_DS,DSP_RW,DSP_MSTRB    : IN STD_LOGIC;
    DSP_ADDR         : IN STD_LOGIC_VECTOR(15 downto 0);
    DSP_DATA         : IN STD_LOGIC_VECTOR(15 downto 0);
--   DSP_CLK          : IN STD_LOGIC;

    BUF_DATAI          : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
    BUF_CLKI,BUF_CLKO,BUF_WE   : OUT STD_LOGIC;
    BUF_ADDR         : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
    BUF_R          : OUT   STD_LOGIC;

    DSP_RDY          : OUT STD_LOGIC);

-- CONSTANT DSP_RESET : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0000";
-- FF00H IN DATA SPACE
  CONSTANT DSP_SDRAM : STD_LOGIC_VECTOR(7 DOWNTO 0) := "11111111";
END DSP_BUF;

ARCHITECTURE a OF DSP_BUF IS
SIGNAL  BUF_W : STD_LOGIC;

BEGIN
-- KEEP DATA BUS AT 'Z' WHEN THERE IS NO I/O READ ACCESS FROM DSP TO FPGA
-- DSP_DATA(15 DOWNTO 8) <= "ZZZZZZZZ" WHEN (BUF_W ='1');
-- DSP_DATA(7 DOWNTO 0) <= BUF_DATAO WHEN(BUF_R = '1') ELSE
--     "ZZZZZZZZ";

-- IO IS ONE WAIT STATE(DEFAULT OF DSP)
  DSP_RDY <='1';
-- IO ADDRESS R/W DECODER
  BUF_W <='1' WHEN (DSP_DS = '0' AND DSP_RW = '0' AND
    DSP_ADDR(15 DOWNTO 8) = DSP_SDRAM ) ELSE '0';
  BUF_R <='1' WHEN (DSP_DS = '0' AND DSP_RW = '1' AND
    DSP_ADDR(15 DOWNTO 8) = DSP_SDRAM ) ELSE '0';
  BUF_WE <= BUF_W;
  BUF_CLKI <= DSP_MSTRB; -- WHEN (BUF_W='1') ELSE (DSP_MSTRB);
  BUF_CLKO <= '0' WHEN (BUF_W='1') ELSE (NOT DSP_MSTRB);
  BUF_ADDR <= DSP_ADDR(6 DOWNTO 0);
  BUF_DATAI <= DSP_DATA(7 DOWNTO 0) WHEN(BUF_W='1');

END a;
```

DSP_READ.VHD

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;


ENTITY DSP_READ IS
  PORT(
    BUF_R,DMA_CNTLR,I2C_CSR,CMOS_STATUSR  : IN STD_LOGIC;
    BUF_DATAO      : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    SD_END        : IN  STD_LOGIC;
    I2C_DAT       : IN STD_LOGIC;
    CMOS_DIO1      : IN STD_LOGIC;
    DSP_DATA      : OUT STD_LOGIC_VECTOR(15 downto 0));

END DSP_READ;

ARCHITECTURE a OF DSP_READ IS

BEGIN
DSP_DATA <= "000000000000" & SD_END & "000" WHEN (DMA_CNTLR='1') ELSE
     "00000000" & BUF_DATAO WHEN (BUF_R='1') ELSE
     "00000000000000" & I2C_DAT & "0" WHEN (I2C_CSR='1') ELSE
     "000000000000000" & CMOS_DIO1 WHEN (CMOS_STATUSR='1') ELSE
     "ZZZZZZZZZZZZZZZZ" ;
END a;
```

IO_DMA.VHD

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY IO_DMA IS
  PORT(
    DSP_IS,DSP_RW,DSP_ISTRB    : IN STD_LOGIC;
    DSP_ADDR         : IN STD_LOGIC_VECTOR(15 downto 0);
    DSP_DATA         : IN STD_LOGIC_VECTOR(15 downto 0);
    DSP_RDY          : OUT STD_LOGIC;
    SD_RW,SD_BANK,SD_START    : OUT STD_LOGIC;
    SD_ADDR_RESET       : OUT STD_LOGIC;
    SD_ADDR          : OUT STD_LOGIC_VECTOR(20 DOWNTO 0);
    DMA_CNTLR         : OUT STD_LOGIC
);

  CONSTANT DMA_ADDRH : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0011";
  CONSTANT DMA_ADDRL : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0100";
  CONSTANT DMA_CNTL  : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0101";
  CONSTANT DSP_RESET : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0000";
END IO_DMA;

ARCHITECTURE a OF IO_DMA IS
SIGNAL  DMA_ADDRHW,DMA_ADDRLW,DMA_CNTLW,SW_RESETW : STD_LOGIC;

BEGIN
-- IO IS ONE WAIT STATE(DEFAULT OF DSP)
  DSP_RDY <='1';
-- IO ADDRESS R/W DECODER
  DMA_ADDRHW <='1' WHEN (DSP_IS = '0' AND DSP_RW = '0' AND
    DSP_ADDR(3 DOWNTO 0) = DMA_ADDRH) ELSE '0';
  DMA_ADDRLW <='1' WHEN (DSP_IS = '0' AND DSP_RW = '0' AND
    DSP_ADDR(3 DOWNTO 0) = DMA_ADDRL) ELSE '0';
  DMA_CNTLW <='1' WHEN (DSP_IS = '0' AND DSP_RW = '0' AND
    DSP_ADDR(3 DOWNTO 0) = DMA_CNTL) ELSE '0';
  DMA_CNTLR <='1' WHEN (DSP_IS = '0' AND DSP_RW = '1' AND
    DSP_ADDR(3 DOWNTO 0) = DMA_CNTL) ELSE '0';

  PROCESS
  BEGIN
    WAIT UNTIL (DSP_ISTRB'EVENT AND DSP_ISTRB='1');
    IF(DMA_ADDRHW='1') THEN
      SD_ADDR(20 DOWNTO 16) <= DSP_DATA(4 DOWNTO 0);
    ELSIF(DMA_ADDRLW='1') THEN
      SD_ADDR(15 DOWNTO 0) <= DSP_DATA;
    ELSIF(DMA_CNTLW='1') THEN
      SD_RW <= DSP_DATA(0);
      SD_BANK <= DSP_DATA(1);
      SD_START <= DSP_DATA(2);
      SD_ADDR_RESET <= DSP_DATA(4);
    END IF;
  END PROCESS;
END a;
```

BIN_MUX.VHD

```
library ieee;
use   ieee.std_logic_1164.all;

ENTITY BIN_MUX IS
  PORT (  SD_PIPE  :  IN   STD_LOGIC_VECTOR(3 DOWNTO 0);
     DSP_BUF_IN :  IN   STD_LOGIC_VECTOR(7 DOWNTO 0);
     CMOS_BUF_IN :  IN   STD_LOGIC_VECTOR(7 DOWNTO 0);
     BUF_IN   :  OUT  STD_LOGIC_VECTOR(7 DOWNTO 0)
   );
END BIN_MUX;

ARCHITECTURE dataflow OF BIN_MUX IS
BEGIN
  BUF_IN <= DSP_BUF_IN WHEN (SD_PIPE="0001") ELSE
       CMOS_BUF_IN WHEN (SD_PIPE(3 DOWNTO 1)="001") ELSE
       "00000000";
END dataflow;
```

CLKI_MUX.VHD

```
library ieee;
use   ieee.std_logic_1164.all;

ENTITY CLKI_MUX IS
  PORT (  SD_PIPE  :  IN STD_LOGIC_VECTOR(3 DOWNTO 0);
     WE_DMAIN,WE_411IN    :  IN STD_LOGIC;
     WE_DMA   :  OUT STD_LOGIC;
     WE_411   :  OUT STD_LOGIC
    );
END CLKI_MUX;

ARCHITECTURE dataflow OF CLKI_MUX IS
BEGIN
  WE_411 <= WE_411IN  WHEN ( SD_PIPE(2)='1') ELSE '0';
  WE_DMA <= WE_DMAIN  WHEN ( SD_PIPE="0001") ELSE '0';
END dataflow;
```

SDACNT2.VHD

```
library ieee;
use   ieee.std_logic_1164.all;
use   ieee.std_logic_arith.all;

ENTITY SDACNT2 IS
  PORT(
    CLK      : IN  STD_LOGIC;
    TIME    : IN  INTEGER RANGE 0 TO 15;
    SD_RW    : IN  STD_LOGIC;
    CNT_INIT_W  : IN  STD_LOGIC;
    CNT_INIT_DSP  : IN  STD_LOGIC_VECTOR(20 DOWNTO 0);
    CNT_INIT_AV411 : IN  STD_LOGIC_VECTOR(20 DOWNTO 0);
    SD_PIPE    : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
    CNT_INIT_CMOS : IN  STD_LOGIC_VECTOR(20 DOWNTO 0);

    SD_ADD09   : OUT   STD_LOGIC_VECTOR(9 DOWNTO 0);
    SD_ADD10   : OUT   STD_LOGIC
    );
end SDACNT2;

ARCHITECTURE a OF SDACNT2 IS
SIGNAL  CNT_INIT : INTEGER RANGE 0 TO 31 ;
SIGNAL  RADD010  : STD_LOGIC_VECTOR(10 DOWNTO 0);
SIGNAL  CADD010  : STD_LOGIC_VECTOR(10 DOWNTO 0);
SIGNAL  RESULT_BIT : STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL  ADDR97   : STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL  TIME_BIT : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL  RESULT   : INTEGER RANGE 0 TO 31;
SIGNAL  TIME_BIT2  : STD_LOGIC;
SIGNAL  TIME_DLYH  : INTEGER RANGE 0 TO 15;
BEGIN
  RADD010 <= CNT_INIT_AV411(20 DOWNTO 10) WHEN (SD_PIPE(3 DOWNTO 2)="01") ELSE
       CNT_INIT_CMOS(20 DOWNTO 10)  WHEN (SD_PIPE(3 DOWNTO 1)="001") ELSE
       CNT_INIT_DSP(20 DOWNTO 10)  WHEN (SD_PIPE="0001") ELSE
       "10000110011";
  ADDR97 <=  CNT_INIT_AV411(9 DOWNTO 7) WHEN (SD_PIPE(3 DOWNTO 2)="01") ELSE
       CNT_INIT_CMOS(9 DOWNTO 7)  WHEN (SD_PIPE(3 DOWNTO 1)="001") ELSE
       CNT_INIT_DSP(9 DOWNTO 7)   WHEN (SD_PIPE="0001") ELSE
       "000";
-- LATENCY=3, BURST=8
  RESULT_BIT <= CONV_STD_LOGIC_VECTOR(RESULT,3);
  CADD010 <=  "10000110011" WHEN ( SD_PIPE(3)='1') ELSE
       "00" & ADDR97 & RESULT_BIT & "000";

  TIME_BIT <= CONV_STD_LOGIC_VECTOR(TIME,4);
-- ADJUST COUNTER TRANSISTION TO RAS TIMING
  PROCESS(CLK)
  BEGIN
    IF (CLK'EVENT AND CLK='1') THEN
      TIME_BIT2 <= TIME_BIT(3);
    END IF;
  END PROCESS;

  PROCESS (TIME_BIT2,CNT_INIT_W)
  VARIABLE        CNTR    : INTEGER RANGE 0 TO 7;
  BEGIN
    IF (CNT_INIT_W='1') THEN
      CNTR := 7;
    ELSIF (TIME_BIT2'EVENT AND TIME_BIT2='1') THEN
      CNTR := CNTR+1;
    END IF;
    RESULT <= CNTR;
  END PROCESS;
```

```
-- ADJUST THE ADDRESS TIMING WITH RAS TIMING
  PROCESS(CLK)
  BEGIN
    IF(CLK'EVENT AND CLK='0') THEN
      TIME_DLYH <= TIME;
    END IF;
  END PROCESS;
-- Synchronize address output
  PROCESS(CLK)
  BEGIN
    IF(CLK'EVENT AND CLK='1') THEN
      IF (((TIME_DLYH=1 OR TIME_DLYH=9) AND SD_RW='1') OR
         ((TIME_DLYH=4 OR TIME_DLYH=12) AND SD_RW='0'))
      THEN
        SD_ADD09 <= RADD010(9 DOWNTO 0);
        SD_ADD10 <= RADD010(10);
      ELSE
        SD_ADD09 <= CADD010(9 DOWNTO 0);
        SD_ADD10 <= RADD010(10);
      END IF;
    END IF;
  END PROCESS;

END a;
```

CNT1283.VHD

```
-- CNT128.vhd
-- JAN 23, 1998
-- Vivian Shao
-- Counter for periodic operation : COUNTING RANGE FROM 0 TO 15
-- Mandy Tsai APR,10th,98 modify
library ieee;
use   ieee.std_logic_1164.all;
USE  ieee.std_logic_arith.all;

ENTITY CNT1283 IS
  PORT(
    SD_PIPE    : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
    SYS_RES    : IN  STD_LOGIC;
    CLK              : IN    STD_LOGIC;
    CLEAR            : IN    STD_LOGIC;
    SD_RW    : IN STD_LOGIC;
    TIME05     : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
    RESULT_DMA      : OUT   STD_LOGIC_VECTOR(6 DOWNTO 0);
    TIME    : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
    CKE,WE_DMA,WE_411: OUT  STD_LOGIC;
    RESULT_411      : OUT   STD_LOGIC_VECTOR(6 DOWNTO 0);
    RESULT_CMOS   : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
  );
end CNT1283;

ARCHITECTURE a OF CNT1283 IS
SIGNAL  CNT_ENABLE,CNT_ENABLE_DLY1,CNT_ENABLE_DLY2, CNT_END: STD_LOGIC;
SIGNAL  CLEAR_DLY  : STD_LOGIC;
SIGNAL  SD_PIPE_RW : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL  CNT_INIT_VALUE, CNT_END_VALUE : INTEGER RANGE 0 TO 511;
SIGNAL  CKE_WE_TEMP,CKE_WE_TEMP1,CKE_DLY1,CKE_DLY2,CKE_DLY11 : STD_LOGIC;

SIGNAL  RESULT_BIT  : STD_LOGIC_VECTOR(8 DOWNTO 0);
SIGNAL  RESULT1  : INTEGER RANGE 0 TO 511;
SIGNAL  CNT_EN,CNT_CLR : STD_LOGIC;
SIGNAL  CKE_DLY3,CKE_DLY4,CKE_DLY5 : STD_LOGIC;
SIGNAL  CNT_EN2 : STD_LOGIC;
SIGNAL  RESULT2_BIT  : STD_LOGIC_VECTOR(8 DOWNTO 0);
SIGNAL  RESULT2,CNT_INIT_VALUE2   : INTEGER RANGE 0 TO 511;
SIGNAL  CKE_DLY12  : STD_LOGIC;
SIGNAL  CNT_EN3 : STD_LOGIC;
SIGNAL  RESULT3_BIT  : STD_LOGIC_VECTOR(8 DOWNTO 0);
SIGNAL  RESULT3,CNT_INIT_VALUE3   : INTEGER RANGE 0 TO 511;
SIGNAL  CNT_EN4 : STD_LOGIC;
SIGNAL  RESULT4_BIT  : STD_LOGIC_VECTOR(8 DOWNTO 0);
SIGNAL  RESULT4,CNT_INIT_VALUE4   : INTEGER RANGE 0 TO 511;
SIGNAL  CLK_NOT : STD_LOGIC;

COMPONENT CNT4
  GENERIC (N     : INTEGER :=7);
  PORT(
    ENABLE   : IN STD_LOGIC;
    LOAD_VALUE : IN INTEGER RANGE 0 TO N;
    CLK,CLEAR  : IN STD_LOGIC;
    RESULT   : OUT INTEGER RANGE 0 TO N
  );
end COMPONENT;
```

```
BEGIN
  CNT_INIT_VALUE <= 128  WHEN (SD_PIPE(3)='1') ELSE
           123  WHEN (SD_PIPE(3)='0' AND SD_RW='0') ELSE
           120 WHEN (SD_PIPE(3)='0' AND SD_RW='1');

  CKE <= CKE_DLY1;
  PROCESS(CLK)
  BEGIN
    IF(CLK'EVENT AND CLK='1') THEN
      CKE_DLY1 <= CNT_EN;
    END IF;
  END PROCESS;

  RESULT_BIT <= CONV_STD_LOGIC_VECTOR(RESULT1,9);
  TIME05 <= RESULT_BIT(6 DOWNTO 0);
-- RESULT <= RESULT_BIT(6 DOWNTO 0);
-- INHIBIT THE LAST COMMAND OUT AFTER FINISHED 128-BYTE OPERATION
  TIME <= RESULT_BIT(3 DOWNTO 0) WHEN(RESULT_BIT(7 DOWNTO 3)/="11111")
     ELSE "0000" ;
  CNT_EN <= NOT RESULT_BIT(8);
  CNT_CLR <= NOT CLEAR AND SYS_RES;

CNT128: CNT4
  GENERIC MAP(511)
    PORT MAP(CNT_EN,CNT_INIT_VALUE,CLK,CNT_CLR,RESULT1);

  WE_DMA <= SD_RW AND CKE_DLY1;

  CNT_INIT_VALUE2 <=  128  WHEN (SD_PIPE(3)='1') ELSE
           125 WHEN (SD_PIPE(3)='0' AND SD_RW='0') ELSE
           119 WHEN (SD_PIPE(3)='0' AND SD_RW='1');
  RESULT2_BIT <= CONV_STD_LOGIC_VECTOR(RESULT2,9);
  RESULT_DMA <= RESULT2_BIT(6 DOWNTO 0);
  CNT_EN2 <= NOT RESULT2_BIT(8);
-- CNT_INIT_VALUE1 <=0;

CNT1282: CNT4
  GENERIC MAP(511)
    PORT MAP(CNT_EN2,CNT_INIT_VALUE2,CLK,CNT_CLR,RESULT2);

  WE_411 <= SD_RW AND CNT_EN3;

  CNT_INIT_VALUE3 <=  128  WHEN (SD_PIPE(3)='1') ELSE
           125  WHEN (SD_PIPE(3)='0' AND SD_RW='0') ELSE
           119 WHEN (SD_PIPE(3)='0' AND SD_RW='1');
  RESULT3_BIT <= CONV_STD_LOGIC_VECTOR(RESULT3,9);
  RESULT_411 <= RESULT3_BIT(6 DOWNTO 0);
  CNT_EN3 <= NOT RESULT3_BIT(8);
  CLK_NOT <= NOT CLK;  -- TIMING ADJUST FOR AV411 BUF

CNT1283: CNT4
  GENERIC MAP(511)
    PORT MAP(CNT_EN3,CNT_INIT_VALUE3,CLK_NOT,CNT_CLR,RESULT3);

  CNT_INIT_VALUE4 <=  128  WHEN (SD_PIPE(3)='1') ELSE
           125  WHEN (SD_PIPE(3)='0' AND SD_RW='0') ELSE
           119 WHEN (SD_PIPE(3)='0' AND SD_RW='1');
  RESULT4_BIT <= CONV_STD_LOGIC_VECTOR(RESULT4,9);
  RESULT_CMOS <= RESULT4_BIT(6 DOWNTO 0);
  CNT_EN4 <= NOT RESULT4_BIT(8);

CNT1284: CNT4
  GENERIC MAP(511)
    PORT MAP(CNT_EN4,CNT_INIT_VALUE4,CLK,CNT_CLR,RESULT4);

END a;
```

SDRWMUX.VHD

```
library ieee;
use   ieee.std_logic_1164.all;

ENTITY SDRWMUX IS
  PORT (  SD_PIPE : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
     SD_RW : IN STD_LOGIC;
     RASR : IN STD_LOGIC;
     CASR : IN STD_LOGIC;
     SDWR : IN STD_LOGIC;
     SDA10R  : IN STD_LOGIC;
     SDA11R  : IN STD_LOGIC;
     SDCSR : IN STD_LOGIC;
     RASW : IN STD_LOGIC;
     CASW : IN STD_LOGIC;
     SDWW : IN STD_LOGIC;
     SDA10W  : IN STD_LOGIC;
     SDA11W  : IN STD_LOGIC;
     SDCSW : IN STD_LOGIC;
     RASREF  : IN STD_LOGIC;
     CASREF  : IN STD_LOGIC;
     SDWREF  : IN STD_LOGIC;
     SDA10REF: IN STD_LOGIC;
     SDA11REF: IN STD_LOGIC;
     SDCSREF : IN STD_LOGIC;
     RAS   : OUT STD_LOGIC;
     CAS   : OUT STD_LOGIC;
     SDW  : OUT STD_LOGIC;
     SDA10   : OUT STD_LOGIC;
     SDA11   : OUT STD_LOGIC;
     SDCS  : OUT STD_LOGIC;
     SDDQM : OUT STD_LOGIC
   );
END SDRWMUX;

ARCHITECTURE dataflow OF SDRWMUX IS

BEGIN
  SDDQM <= '0';

  RAS <=  RASREF  WHEN ( SD_PIPE(3)='1') ELSE
     RASW WHEN ( SD_PIPE(3)='0' AND SD_RW='0') ELSE
     RASR;

  CAS <=  CASREF  WHEN ( SD_PIPE(3)='1') ELSE
     CASW WHEN ( SD_PIPE(3)='0' AND SD_RW='0') ELSE
     CASR;

  SDW <=  SDWREF  WHEN ( SD_PIPE(3)='1') ELSE
     SDWW WHEN ( SD_PIPE(3)='0' AND SD_RW='0') ELSE
     SDWR;

  SDA10 <=SDA10REF WHEN ( SD_PIPE(3)='1') ELSE
     SDA10W  WHEN ( SD_PIPE(3)='0' AND SD_RW='0') ELSE
     SDA10R;

  SDA11 <=   SDA11REF WHEN ( SD_PIPE(3)='1') ELSE
     SDA11W  WHEN ( SD_PIPE(3)='0' AND SD_RW='0') ELSE
     SDA11R;

  SDCS <=  SDCSREF WHEN ( SD_PIPE(3)='1') ELSE
     SDCSW WHEN ( SD_PIPE(3)='0' AND SD_RW='0') ELSE
     SDCSR;

END dataflow;
```

RWMUX.VHD

```
library ieee;
use   ieee.std_logic_1164.all;

ENTITY RWMUX IS
  PORT (  SD_PIPE : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
     DSP_RW  : IN STD_LOGIC;
     SD_RW   : OUT STD_LOGIC
   );
END RWMUX;

ARCHITECTURE dataflow OF RWMUX IS

BEGIN
  SD_RW <=DSP_RW WHEN ( SD_PIPE="0001") ELSE   -- DSP
     '1'   WHEN ( SD_PIPE(3 DOWNTO 2)="01") ELSE  -- AV411
     '0'   WHEN ( SD_PIPE(3 DOWNTO 1)="001") ELSE '1'; --CMOS

END dataflow;
```

SDR_CMD1.VHD

```
PACKAGE SDRAM_CMD_PKG IS
  TYPE SDRAM_STATE_TYPE IS (REFR, WRITE128, READ128,READ32);
  TYPE PULSE_SW_STATE IS (ZERO,CKEE,DSP,AV411,CMOS);
END SDRAM_CMD_PKG;

USE WORK.SDRAM_CMD_PKG.ALL;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY SDR_CMD1 IS
  PORT(
-- LET ENABLE(SD_RW, SD_REF) SYNC WITH DSP_CLK
-- TO MAKE SURE THAT ALL COUNTER CAN BE SYNC
   SYS_RES          : IN STD_LOGIC;
   DSP_CLK,CKE         : IN STD_LOGIC;
   CMD_TRIG,DSP_START      : IN STD_LOGIC;
   CMD            : IN SDRAM_STATE_TYPE;
   CMOS_TRIG          : IN STD_LOGIC;

   CNT_RES          : OUT STD_LOGIC;
   SD_PIPE          : OUT   STD_LOGIC_VECTOR(3 DOWNTO 0);
   INT1          : OUT STD_LOGIC
  );
end SDR_CMD1;

ARCHITECTURE a OF SDR_CMD1 IS
SIGNAL  SD_LAT1,SD_LAT2,SD_LAT3 : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL  CKE_AV411_RES, CKE_DSP_RES, CKE_REFR_RES : STD_LOGIC;
SIGNAL  CMD_TRIG_DLY1, CMD_TRIG_DLY2 : STD_LOGIC;
SIGNAL  DSP_START_DLY1, DSP_START_DLY2  : STD_LOGIC;
SIGNAL  CKE_DLY1,CKE_DLY2      : STD_LOGIC;
SIGNAL  CMD_TRIG_PULSE,DSP_START_PULSE  : STD_LOGIC;
SIGNAL  CKE_RES_PULSE,CKE_RES_PULSE_DLY : STD_LOGIC;
SIGNAL  CNT_PULSE_SW        : PULSE_SW_STATE;
SIGNAL  CKE_CLR_PULSE, DSP_CLR_PULSE, AV411_CLR_PULSE : STD_LOGIC;
SIGNAL  DSP_START_CLK,CMD_TRIG_CLK   : STD_LOGIC;
SIGNAL  CKE_DSP_RES1        : STD_LOGIC;
SIGNAL  CMOS_TRIG_CLK, CMOS_TRIG_PULSE : STD_LOGIC;
SIGNAL  CMOS_TRIG_DLY1, CMOS_TRIG_DLY2  : STD_LOGIC;
SIGNAL  CKE_CMOS_RES        : STD_LOGIC;
BEGIN
-- SYNC TRIG SIGNALS WITH DSP_CLK
  PROCESS(DSP_CLK)
  BEGIN
    CMD_TRIG_CLK <= CMD_TRIG;
    DSP_START_CLK <= DSP_START;
    CMOS_TRIG_CLK <= CMOS_TRIG;
  END PROCESS;

  SD_PIPE <= SD_LAT1;
  PROCESS(CMD_TRIG_CLK,CKE_REFR_RES)
  BEGIN
    IF(CKE_REFR_RES = '1') THEN
      SD_LAT1(3) <= '0';
    ELSIF(CMD_TRIG_CLK'EVENT AND CMD_TRIG_CLK='1') THEN
      IF(CMD=REFR) THEN
        SD_LAT1(3) <= '1';
      ELSE SD_LAT1(3) <= '0';
      END IF;
    END IF;
  END PROCESS;
```

```
 PROCESS(CMD_TRIG_CLK,CKE_AV411_RES)
 BEGIN
   IF(CKE_AV411_RES = '1') THEN
     SD_LAT1(2) <= '0';
   ELSIF(CMD_TRIG_CLK'EVENT AND CMD_TRIG_CLK='1') THEN
     IF(CMD/=REFR) THEN
       SD_LAT1(2) <= '1';
     ELSE SD_LAT1(2) <= '0';
     END IF;

   END IF;
 END PROCESS;

 PROCESS(CMOS_TRIG_CLK,CKE_CMOS_RES)
 BEGIN
   IF (CKE_CMOS_RES='1') THEN
     SD_LAT1(1) <= '0';
   ELSIF(CMOS_TRIG_CLK'EVENT AND CMOS_TRIG_CLK='1') THEN
     SD_LAT1(1) <= '1';
   END IF;
 END PROCESS;

 PROCESS(DSP_START_CLK,CKE_DSP_RES)
 BEGIN
   IF (CKE_DSP_RES='1') THEN
     SD_LAT1(0) <= '0';
   ELSIF(DSP_START_CLK'EVENT AND DSP_START_CLK='1') THEN
     SD_LAT1(0) <= '1';
   END IF;
 END PROCESS;

 PROCESS(DSP_CLK)
 BEGIN
   IF(DSP_CLK='1') THEN
     SD_LAT2 <= SD_LAT1;
     SD_LAT3 <= SD_LAT2;    -- PREVIOUS SD STATE
     CMD_TRIG_DLY1 <= CMD_TRIG;
     CMD_TRIG_DLY2 <= CMD_TRIG_DLY1;
     CMOS_TRIG_DLY1 <= CMOS_TRIG;
     CMOS_TRIG_DLY2 <= CMOS_TRIG_DLY1;
     DSP_START_DLY1 <= DSP_START;
     DSP_START_DLY2 <= DSP_START_DLY1;
     CKE_DLY1 <= CKE;
     CKE_DLY2 <= CKE_DLY1;
   END IF;
 END PROCESS;

-- PULSE GENERATOR
 CKE_RES_PULSE <= NOT CKE AND CKE_DLY1;
 CKE_RES_PULSE_DLY <= NOT CKE_DLY1 AND CKE_DLY2;
 CMD_TRIG_PULSE <= CMD_TRIG_DLY1 AND NOT CMD_TRIG_DLY2;
 CMOS_TRIG_PULSE <= CMOS_TRIG_DLY1 AND NOT CMOS_TRIG_DLY2;
 DSP_START_PULSE <= DSP_START_DLY1 AND NOT DSP_START_DLY2;
-- CLEAR SD_LAT STATE WHEN CKE OCCURS
 CKE_REFR_RES <= (CKE_RES_PULSE AND SD_LAT3(3)) OR NOT SYS_RES;
 CKE_AV411_RES <= (CKE_RES_PULSE AND ( NOT SD_LAT3(3) AND SD_LAT3(2)))
       OR NOT SYS_RES;
 CKE_CMOS_RES <= (CKE_RES_PULSE AND
       (NOT SD_LAT3(3) AND NOT SD_LAT3(2) AND SD_LAT3(1)))
       OR NOT SYS_RES;
 CKE_DSP_RES1 <= CKE_RES_PULSE AND
       (NOT SD_LAT3(3) AND NOT SD_LAT3(2) AND NOT SD_LAT3(1) AND SD_LAT3(0));
 CKE_DSP_RES <= CKE_DSP_RES1 OR NOT SYS_RES;
```

```
-- CNT_RES ABITRATOR
 CNT_RES <=  '0'        WHEN SD_LAT1="0000" ELSE
      CKE_RES_PULSE_DLY WHEN (SD_LAT1(0)='1' AND
      ((SD_LAT1(3)='0' AND SD_LAT3(3)='1') OR
       (SD_LAT1(2)='0' AND SD_LAT3(2)='1') OR
       (SD_LAT1(1)='0' AND SD_LAT3(1)='1'))) ELSE
      CKE_RES_PULSE_DLY   WHEN (SD_LAT1(1)='1' AND
      ((SD_LAT1(3)='0' AND SD_LAT3(3)='1') OR
       (SD_LAT1(2)='0' AND SD_LAT3(2)='1'))) ELSE
      CMD_TRIG_PULSE   WHEN (SD_LAT1(3)='1' OR
       SD_LAT1(2)='1')          ELSE
      CMOS_TRIG_PULSE   WHEN (SD_LAT1(3 DOWNTO 1)="001" AND
        SD_LAT3(3 DOWNTO 1)="000")    ELSE
      DSP_START_PULSE   WHEN ( SD_LAT1="0001" AND
       SD_LAT3="0000")        ELSE '0';

-- GENERATE INT1 TO DSP (20 CLK WIDTH)
 PROCESS(DSP_CLK,SYS_RES)
 VARIABLE        CNTR    : INTEGER RANGE 0 TO 31;
 BEGIN
   IF (SYS_RES = '0') THEN
     CNTR:=20;
     INT1 <= '1';
   ELSIF (DSP_CLK'EVENT AND DSP_CLK='0') THEN
     IF (CKE_DSP_RES1 = '1') THEN
       CNTR:=0;
     ELSIF (CNTR = 20) THEN
       INT1 <= '1';
     ELSE
       INT1 <= '0';
       CNTR := CNTR+1;
     END IF;
   END IF;
 END PROCESS;

END a;
```

SDR_CNTL.VHD

```
library ieee;
use ieee.std_logic_1164.all;
--------------------------------------------------------------------
ENTITY SDR_CNTL IS
   PORT (
      SYS_RES    : IN STD_LOGIC;
      SDA10    : IN STD_LOGIC;
      TIME             : IN    INTEGER RANGE 0 TO 15;
      CLK80            : IN    STD_LOGIC;
      RRAS              : OUT   STD_LOGIC;
      RCAS              : OUT   STD_LOGIC;
      RSD_W             : OUT   STD_LOGIC;
      RSD_A10           : OUT   STD_LOGIC;
      RSD_A11           : OUT   STD_LOGIC;
      RSD_CS            : OUT   STD_LOGIC
     );
END SDR_CNTL;
ARCHITECTURE a OF SDR_CNTL IS
SIGNAL  RAS,CAS,CAS1,SD_W,SD_A10,SD_A11,SD_CS,CMD_EN:  STD_LOGIC;
SIGNAL  RSD_A111 : STD_LOGIC;
BEGIN
  WITH TIME SELECT
  RAS <=
     '0' WHEN 1,
     '0' WHEN 9,
     '1' WHEN others;
  WITH TIME SELECT
  CAS <=
     '0' WHEN 4,
     '0' WHEN 12,
     '1' WHEN others;
  SD_W <=  '1';
  WITH TIME SELECT
  SD_A11 <=
     '0' WHEN 9,
     '0' WHEN 12,
     '1' WHEN others;
  SD_CS <= '0';
  WITH TIME SELECT
  SD_A10 <= '1' WHEN 4,
       '1' WHEN 12,
      SDA10 WHEN others;
  PROCESS(CLK80)
  BEGIN
    IF (CLK80'EVENT AND CLK80 ='1') THEN
      RRAS <= RAS;
      RCAS <= CAS;
      RSD_W <= SD_W;
      RSD_A10 <= SD_A10;
      RSD_A11 <= SD_A11;
      RSD_CS <= SD_CS;
    END IF;
  END PROCESS;
END a;
```

Appendix B Test Algorithm

Main Program : Demo.asm

```
*********************************************************************
* This program is written specially for testing the SDRAM used in *
* the Digital Still Camera Demonstration Board.                     *
*********************************************************************

*--------------------------------+
* IO PORT address designation    |
*--------------------------------+
RESET_PORT  .set    0000H
DMA_ADDRH   .set    0003H
DMA_ADDRL   .set    0004H
DMA_CTL     .set    0005H
*************************************************************************

*--------------------------------+
* Constants declaration          |
*--------------------------------+
STARTEND    .set    0FFFBH
ADDR_INC    .set    0128D                   ;Address incremental at 80H steps
ENDLOOP     .set    16384D                  ;Counter for 2 Mbyte write to SDRAM
*************************************************************************

*-----------------------------------------------------------+
* Command to FPGA is in Data Bits of I/O Address DMA_CTL    |
*                                                           |
* Data Bits :   D4        D3       D2      D1      D0       |
* Command   : SD_ADDR   Polling   Start   B0/B1    R/W      |
*                                                           |
* SD_ADDR - Hold SDRAM Address                              |
* Polling - Not Used                                        |
* Start   - Start the read or write cycle                   |
* B0/B1   - Set either B0 or B1 buffer for SDRAM            |
* R/W     - Read or Write command                           |
*-----------------------------------------------------------+

*-----------------------------------+
* Command to toggle the buffer only |
*-----------------------------------+
SW_B0       .set    0FFFDH
SW_B1       .set    00002H

*--------------------------------+
* Command to toggle, Read/Write  |
*--------------------------------+
DSPB0_WS    .set    0004H                   ;DSP write - SDRAM read to B0 w START
DSPB1_WS    .set    0006H                   ;DSP write - SDRAM read to B1 w START
DSPB0_RS    .set    0005H                   ;DSP read - SDRAM write from B0 w START
DSPB1_RS    .set    0007H                   ;DSP read - SDRAM write from B1 w START

*-----------------------------------------+
* Command to toggle, Read/Write & Hold Addr |
*-----------------------------------------+
DSPB0_WAD   .set    0010H                   ;DSP write from B0 w SD_ADDr
DSPB1_WAD   .set    0012H                   ;DSP write from B1 w SD_ADDr
DSPB0_RAD   .set    0011H                   ;DSP read to B0 w SD_ADDr
DSPB1_RAD   .set    0013H                   ;DSP read to B1 w SD_ADDr

*************************************************************************
```

```
*----------------------------------+
* Buffers and Variables Declaration |
*----------------------------------+
READ_BUF    .usect  ".READ_BUF",128,1         ;Store 128 bytes of ref data
COM_BUF     .usect  ".COM_BUF" ,128,1         ;Store 128 bytes of data read from SD_BUF
SD_BUF      .usect  ".SD_BUF"  ,128,1         ;Store 128 bytes of data read from or
                                              ;write to SDRAM
stk         .usect  ".stack",32               ;initializing stack

            .bss    ADDR     ,2  ,1           ;Store address for SDRAM
            .bss    IO_DMA   ,1  ,1           ;Store command to send to SDRAM
            .bss    ERRCOUNT ,1  ,1           ;Store error for data comparison
            .bss    LOOPCT   ,1  ,1           ;Store loop count
            .bss    INTPOLL  ,1  ,1           ;Use for polling
            .bss    IO_DMA1  ,1  ,1           ;Store High and Low SDRAM address byte

*************************************************************************

*--------------+
* MAIM PROGRAM |
*--------------+

            .def    IO_DMA, IO_DMA1, INTPOLL
            .def    READ_BUF, ADDR, ERRCOUNT, LOOPCT, SD_BUF, COM_BUF
            .def    INT1, ENDLOOP
            .def    MAIN

            .mmregs
            .text

MAIN:

      SSBX    INTM                          ;disable interrupt
      STM     #0111111110100000b  ,PMST     ;initial PMST... MP/MC=0,OVLY=1,DROM=0
      STM     #0H                 ,IMR
      STM     #stk+32             ,SP       ;set stack pointer to the bottom of the stack
      STM     #0010001000000000b  ,ST1      ;initial ST1
      STM     #0                  ,SWWSR    ;zero wait state
      STM     #37FEH              ,CLKMD    ;PLL X 4,
                                            ;PLLMUL=3,PLLNDIV=1,PLLCOUNT=0XFF,PLLON=1
      STM     #00D0H              ,PRD      ;GENERATE 4.7US, PRD=000EH, TDDR=1H
      ORM     #8                  ,IMR
      STM     #0020h              ,TCR      ;TRB=1,reload: PRD -> TIM, TDDR -> PSC
      ANDM    #0FFFFH             ,IFR      ;CLEAR PENDING INTRERUPT
      NOP
      NOP
      RSBX    INTM                          ;enable interrupt


      LD      #IO_DMA       ,DP             ;INITIAL DMA CONTROL PORT
      ST      #0            ,IO_DMA         ;START=0
      PORTW   IO_DMA        ,DMA_CTL        ;START=0

*************************************************************
```

```
*-------------------------+
* DSP write to SDRAM Phase |
*-------------------------+

;===Initializing Phase=====;

      LD      #0, DP
      ORM     #2,IMR                      ;ENABLE INT1
      LD      #0            ,B            ;Init pointer of SDRAM addr (WR)
      STM     #ADDR         ,AR1          ;Init pointer of SDRAM addr (WR)
      DST     B             ,*AR1         ;Init pointer of SDRAM addr (WR)
      STM     #READ_BUF     ,AR3          ;Init pointer of READ_BUF
      ST      #0            ,*AR3         ;Init pointer of READ_BUF
      STM     #ERRCOUNT     ,AR5          ;Init Err indicator in COM_DATA
      ST      #0            ,*AR5         ;Init Err indicator in COM_DATA
      STM     #INTPOLL      ,AR0          ;Init Polling Bit
      ST      #0            ,*AR0         ;Init Polling Bit
      STM     #LOOPCT       ,AR4          ;Init loop counter for 2M SDRAM
      ST      #(ENDLOOP)    ,*AR4         ;Init loop counter for 2M SDRAM

      CALL    REF_DATA                    ;Store ref data

;====End Of Initializing Phase====;

;====Clear Buffer B0 and B1=====;

* B0 : 0X00 , B1 : 0XFF
      CALL    DSPRB1                      ;Toggle B1 to B0, DSP see B0
      NOP
      NOP
      CALL    CL_BUF00                    ;DSP clear B0 buffer
      NOP
      NOP
      CALL    DSPRB0                      ;Toggle B0 to B1, DSP see B1
      NOP
      NOP
      CALL    CL_BUFFF                    ;DSP clear B1 buffer
      NOP
      NOP
      CALL    DSPRB1                      ;Toggle B1 to B0, DSP see B0
      NOP
      NOP

;====End of Clear B0 and B1====;

;==============Start of DSP writing data to SDRAM==============;

      CALL    WRITE_BUF                   ;DSP write data to B0
      CALL    DSP_WB0                     ;SDRAM read from B0, DSP see B1

DSP_WRSTART:
      SSBX    INTM                        ;Disable interrupts
      CALL    WRITE_BUF                   ;DSP write data to B1
      RSBX    INTM                        ;Enable interrupts

DSP_WR:
      LD      *AR0          ,A            ;looping for Interrupt
      AND     #01H          ,A            ;looping for Interrupt
      BC      DSP_WR        ,AEQ          ;looping for Interrupt

      SSBX    INTM
      ST      #0            ,*AR0         ;Set INTPOLL=0 to enable looping
      LD      *AR4          ,A            ;Has DSP complete write to 2M SDRAM
      BC      DSP_READ      ,AEQ          ;A=0 => yes, jump to DSP_READ

      LD      IO_DMA        ,A            ;Verify whether B0 or B1 is access
      AND     #00010B       ,A            ;Verify whether B0 or B1 is access
      CC      DSP_WB1W      ,AEQ          ;DSP write data to B1
      CC      DSP_WB0W      ,ANEQ         ;DSP write data to B0
```

```
      RSBX    INTM

      B       DSP_WRSTART

;========End Of DSP writing data to SDRAM============;
*************************************************************

*--------------------------+
* DSP read from SDRAM Phase |
*--------------------------+

;====Initializing Phase====;
DSP_READ:
      LD      #0            ,B            ;Init pointer to SDRAM addr (RD)
      STM     #ADDR         ,AR1          ;Init pointer to SDRAM addr (RD)
      DST     B             ,*AR1         ;Init pointer to SDRAM addr (RD)
      STM     #INTPOLL      ,AR0          ;Init Polling Bit
      ST      #0            ,*AR0         ;Init Polling Bit
      STM     #LOOPCT       ,AR4          ;Init loop counter for 2M SDRAM
      ST      #(ENDLOOP)    ,*AR4         ;Init loop counter for 2M SDRAM

;====End of Initializing Phase====;

;====Clearing Buffer B0 and B1====;

*B0 : 0X00 , B1 : 0XFF
      CALL    DSPRB1                      ;Toggle B1 to B0, DSP see B0
      NOP
      NOP
      CALL    CL_BUF00                    ;DSP clear B0 buffer
      NOP
      NOP
      CALL    DSPRB0                      ;Toggle B0 to B1, DSP see B1
      NOP
      NOP
      CALL    CL_BUFFF                    ;DSP clear B1 buffer
      NOP
      NOP
;====End of Clearing Buffer B0/B1====;

;===============Start of DSP reading data from SDRAM=============;

      CALL    DSP_RB0                     ;SDRAM write to B0, DSP see B1

DSP_RD:
      LD      *AR0          ,A            ;looping
      AND     #01H          ,A            ;looping
      BC      DSP_RD        ,AEQ          ;looping

      ST      #0            ,*AR0         ;Set INTPOLL=0
      LD      *AR4          ,A            ;Has DSP write to 2M SDRAM
      BC      END_RD        ,AEQ          ;A=0 => yes, jump to DSP_READ
      LD      IO_DMA        ,A            ;Verify whether B0 or B1 is access
      AND     #00010B       ,A            ;Verify whether B0 or B1 is access
      CC      DSP_RB1       ,AEQ          ;SDRAM write to B1
      CC      DSP_RB0       ,ANEQ         ;SDRAM write to B0

      SSBX    INTM                        ;Disable Interrupt
      CALL    CL_COMPBUF                  ;Clear COM_BUF
      CALL    COMPBUF                     ;Load data from SD_BUF to COM_BUF
      CALL    COM_DATA                    ;Compare data from COM_BUF to READ_BUF
      RSBX    INTM                        ;Enable Interrupt

      B       DSP_RD                      ;Jump back to DSP_RD
END_RD:
      LD      IO_DMA        ,A            ;Verify whether B0 or B1 is access
      AND     #00010B       ,A            ;Verify whether B0 or B1 is access
      CC      DSPRB1        ,AEQ          ;Toggle B1 to B0, DSP see B0
      CC      DSPRB0        ,ANEQ         ;Toggle B0 to B1, DSP see B1
```

```
      CALL    CL_COMPBUF
      CALL    COMPBUF                    ;Load data from SD_BUF to COM_BUF
      CALL    COM_DATA                   ;Compare data from COM_BUF to READ_BUF

END_TEST:
      B       END_TEST

***********************END OF MAIN PROGRAM*******************************


*****************INTERRUPT SERVICE ROUTINE : INT1*********************

INT1:    ;hardware interrupt from SDRAM;

      NOP
      ST      #1            ,*AR0          ;set INTPOLL=1 to disable looping
      LD      *AR4          ,A             ;Decrement loop count
      SUB     #1D           ,A             ;Decrement loop count
      STL     A             ,*AR4          ;Decrement loop count;
      LD      #0            ,A             ;Clear acc A, errors detected is reduce
      RETE

**************************END OF ISR*********************************

*-------------+
* Subroutines |
*-------------+

;----Subroutine #1----Write REF Data----;
REF_DATA:      ;store ref data into READ_BUF
      LD      #1            ,A
      STM     #READ_BUF     ,AR3
      STM     #128-1        ,BRC
      RPTB    READ-1
      STL     A             ,*AR3+
      ADD     #1            ,A   ,A
READ:
      RET
;----End of Subroutine #1----;

;-------------------------------------------------

;----Subroutine #2----DSP write to Buffer B0/B1----;
WRITE_BUF:     ;write data to buffer

      STM     #READ_BUF     ,AR3
      STM     #SD_BUF       ,AR6
      STM     #128,BK
* MANDY, DSP STRB IN FPGA AND NEED ONE LATENCY FOR WRITING
      STM     #128          ,BRC
      RPTB    WRI-1
      LD      *AR3+%        ,A
      STL     A             ,*AR6+%
WRI:  RET
;----End of Subroutine #2----;

;-------------------------------------------------

;-----Subroutine #3----DSP wrtie data from SD_BUF to COM_BUF----;
COMPBUF:     ;write data to buffer
      STM     #COM_BUF+127  ,AR2
      STM     #SD_BUF+1     ,AR6
      STM     #128,BK
      STM     #128+1        ,BRC
      RPTB    COMB-1
      LD      *AR6+%        ,A
      STL     A             ,*AR2+%
COMB: RET
;----End of Subroutine #3----;
```

```
;-------------------------------------------------

;----Subroutine #4----Clear Data from COM_BUF----;
CL_COMPBUF:  ;Clear data from COM_BUF
      LD      #0            ,A
      STM     #COM_BUF      ,AR2
      STM     #128-1        ,BRC
      RPTB    CL-1
      STL     A             ,*AR2+
CL:   RET
;----End of Subroutine #4----;

;-------------------------------------------------

;----Subroutine #5----Compare data----;
COM_DATA:      ;comparing data from SDRAM with ref data
      STM     #COM_BUF      ,AR2
      STM     #READ_BUF     ,AR3
      STM     #128-1        ,BRC
      RPTB    COM-1
      SUB     *AR2+         ,*AR3+ ,A
      CC      ERRCOUNTER    ,ANEQ
COM:  RET

ERRCOUNTER:
      LD       *AR5          ,A
      ADD      #1            ,A
      STL      A             ,*AR5
      LD       #0            ,A
      RET
;----End of Subroutine #5----;

;-------------------------------------------------

;----Subroutine #6----Send address to SDRAM----;
SDRAM_ADDR:    ;sending addr via I/O to SDRAM
      LD      #IO_DMA1      ,DP
      DLD     *AR1          ,B
      STH     B             ,IO_DMA1
      PORTW   IO_DMA1       ,DMA_ADDRH
      NOP
      NOP
      NOP
      STL     B             ,IO_DMA1
      PORTW   IO_DMA1       ,DMA_ADDRL
      NOP
      NOP
      NOP
      ADD     #(ADDR_INC)   ,B
      DST     B             ,*AR1
      RET
;----End of Subroutine #6----;


************************************************************
;----Subroutine #7----DSP write data to SDRAM thru B0----;
;----Start of DSP_WB0----;
DSP_WB0: ;setup writing of data to B0

      CALL    SDRAM_ADDR

      LD      #IO_DMA       ,DP
      ST      #(DSPB0_WAD)  ,IO_DMA        ;Load SD_ADDr, Read, B0 instr
      PORTW   IO_DMA        ,DMA_CTL       ;Send SD_ADDr, Read, B0 instr
      NOP
      NOP
      NOP
      ST      #(DSPB0_WS)   ,IO_DMA        ; W=0, START=HIGH PULSE, B#=B0
```

```
      PORTW   IO_DMA        ,DMA_CTL
      NOP
      NOP
      NOP
      ANDM    #STARTEND     ,IO_DMA
      PORTW   IO_DMA        ,DMA_CTL
      NOP
      NOP
      NOP
      RET
;----End of DSP_WB0----;


;----Start of DSP_WB0W----;
DSP_WB0W: ;setup writing of data to B0

      CALL    SDRAM_ADDR                  ;Send address to SDRAM

      LD      #IO_DMA       ,DP
      ST      #(DSPB0_WAD)  ,IO_DMA       ;Load SD_ADDr, Read, B0 instr
      PORTW   IO_DMA        ,DMA_CTL      ;Send SD_ADDr, Read, B0 instr
      NOP
      NOP
      NOP
      ST      #(DSPB0_WS)   ,IO_DMA       ;W=0, START=HIGH PULSE, B#=B0
      PORTW   IO_DMA        ,DMA_CTL
      NOP
      NOP
      NOP
      ANDM    #STARTEND     ,IO_DMA       ;W=0. START=LOW PULSE, B#=B0
      PORTW   IO_DMA        ,DMA_CTL
      CALL    WRITE_BUF
      RET
;----End of DSP_WB0W----;
;----End of Subroutine #7----;

;-------------------------------------------------

;----Subroutine #8----DSP write data to SDRAM thru B1----;
;----Start of DSP_WB1----;
DSP_WB1:     ;setup writing of data to B1

      CALL    SDRAM_ADDR

      LD      #IO_DMA       ,DP
      ST      #(DSPB1_WAD) ,IO_DMA        ;Load SD_ADDr, Read, B1 instr
      PORTW   IO_DMA       ,DMA_CTL       ;Send SD_ADDr, Read, B1 instr
      NOP
      NOP
      NOP
      ST      #(DSPB1_WS)   ,IO_DMA       ;W=0, START=HIGH PULSE, B#=B1
      PORTW   IO_DMA        ,DMA_CTL
      NOP
      NOP
      NOP
      ANDM     #STARTEND, IO_DMA          ;W=0, START=LOW PULSE, B#=B1
      PORTW    IO_DMA, DMA_CTL
      NOP
      NOP
      NOP
      RET
;----End of DSP_WB1----;


;----End of DSP_WB1W---;
DSP_WB1W:     ;setup writing of data to B1
      CALL    SDRAM_ADDR
      LD      #IO_DMA       ,DP
      ST      #(DSPB1_WAD) ,IO_DMA        ;Load SD_ADDr, Read, B1 instr
```

```
      PORTW   IO_DMA       ,DMA_CTL       ;Send SD_ADDr, Read, B1 instr
      NOP
      NOP
      NOP
      ST      #(DSPB1_WS)   ,IO_DMA       ;W=0, START=HIGH PULSE, B#=B1
      PORTW   IO_DMA        ,DMA_CTL
      NOP
      NOP
      NOP
      ANDM     #STARTEND, IO_DMA          ;W=0, START=LOW PULSE, B#=B1
      PORTW    IO_DMA, DMA_CTL
      CALL    WRITE_BUF
      RET
;----End of DSP_WB1W----;
;----End of Subroutine #8----;

;--------------------------------------------------

;-----Subroutine #9- SDRAM write to DSP thru B0----;
DSP_RB0: ;sending intr to SDRAM to send data to buffer B0

      CALL    SDRAM_ADDR

      LD      #IO_DMA      ,DP            ;
      ST      #(DSPB0_RAD) ,IO_DMA        ;write & SD_ADDr
      PORTW   IO_DMA       ,DMA_CTL       ;
      NOP
      NOP
      NOP
      ST      #(DSPB0_RS)  ,IO_DMA        ; R=1, START=HIGH PULSE, B#=B0
      PORTW   IO_DMA       ,DMA_CTL       ;
      NOP
      NOP
      NOP
      ANDM    #STARTEND    ,IO_DMA        ; R=1, START=LOW PULSE, B#=B0
      PORTW   IO_DMA       ,DMA_CTL       ;
      NOP
      NOP
      NOP
      RET
;----End of Subroutine #9----;

;--------------------------------------------------

;----Subroutine #10----SDRAM write data to DSP thru B1----;
DSP_RB1: ;sending intr to SDRAM to send data to buffer B1

      CALL    SDRAM_ADDR

      LD      #IO_DMA      ,DP
      ST      #(DSPB1_RAD) ,IO_DMA        ;write & SD_ADDr
      PORTW   IO_DMA       ,DMA_CTL       ;
      NOP
      NOP
      NOP
      ST      #(DSPB1_RS)  ,IO_DMA        ;R=1, START=HIGH PULSE, B#=B1
      PORTW   IO_DMA       ,DMA_CTL       ;
      NOP
      NOP
      NOP
      ANDM     #STARTEND   ,IO_DMA        ;R=1, START=LOW PULSE, B#=B1
      PORTW    IO_DMA      ,DMA_CTL
      NOP
      NOP
      NOP
      RET
;----End of Subroutine #10----;

;--------------------------------------------------
```

```
;----Subroutine 11 ---- Toggling Buffer ----;
;----Start of DSPRB0----;
DSPRB0:  ;To Toggle buffer : SDRAM-B0, DSP-B1
      LD      #IO_DMA      ,DP            ;
      ANDM    #SW_B0       ,IO_DMA        ;
      PORTW   IO_DMA       ,DMA_CTL       ;
      RET
;----End of DSPRB0----;

;----Start of DSPRB1----;
DSPRB1:  ;To Toggle buffer : SDRAM-B1, DSP-B0
      LD      #IO_DMA      ,DP            ;
      ORM      #SW_B1      ,IO_DMA        ;
      PORTW   IO_DMA       ,DMA_CTL       ;
      RET
;----End of DSPRB1----;

;-------------------------------------------------

;----Subroutine 12---Clear Buffer----;
;----Start of CL_BUFFF----;
CL_BUFFF:
      STM     #SD_BUF       ,AR6
      STM     #128-1        ,BRC
      RPTB    CL1-1
      LD      #0FFH         ,A
      STL     A             ,*AR6+
CL1:  RET
;----End of CL_BUFFFF----;

;----Start of CL_BUF00----;
CL_BUF00:
      STM     #SD_BUF       ,AR6
      STM     #128-1        ,BRC
      RPTB    CL2-1
      LD      #00H          ,A
      STL     A             ,*AR6+
CL2:  RET
;----End of CL_BUF00----;

*****************************END OF SUBROUTINE*****************************
```

Command File : Demo.cmd

```
demo.obj
SD_VEC.obj
/*-c*/
-o demo.out
-m demo.map
/*-stack 0x100*/
/*-i c:\mandy\c54x\comp1.10*/
/*-l c:\mandy\c54x\comp1.10\rts.lib*/

MEMORY
{
  PAGE 0:
    RAM0 (RIX)      : origin = 04000h  length = 03F00h
    RAM1 (RIX)      : origin = 07F00h  length = 00080h
    VECS (RIX)      : origin = 07F80h  length = 00080h

 PAGE 1:
    DARAM_MMR (RW)  : origin = 00060h  length = 00020h
    DARAM (RW)      : origin = 00080h  length = 01F80h
    SARAM0 (RW)     : origin = 02000h  length = 02000h
    SARAM1 (RW)     : origin = 04000h  length = 04000h
    FLASH (RW)      : origin = 08000h  length = 07F00h
    SDRAM (RW)      : origin = 0FF00H  length = 00100h
}

SECTIONS
{
       .text : {}  > RAM0  PAGE 0
       .data : {}  > SARAM0  PAGE 1
       .vectors : {} > VECS PAGE 0         /* interrupt vector table */
       .bss  : {}  > DARAM PAGE 1
       .CMOS : {}  > SARAM0 PAGE 1
       .stack   : {} > DARAM PAGE 1
       .READ_BUF ALIGN(256): {} > DARAM PAGE 1
       .COM_BUF ALIGN(256): {} > DARAM PAGE 1
       .SD_BUF ALIGN(256): {} > SDRAM PAGE 1
/*       .cinit   : {} > ROM  PAGE 0*/
/*       .const   : {} > ROM  PAGE 0*/
}

****************************END OF COMMAND FILE***************************
```

TI Contact Numbers

INTERNET

TI Semiconductor Home Page
www.ti.com/sc

TI Distributors
www.ti.com/sc/docs/distmenu.htm

PRODUCT INFORMATION CENTERS

Americas
Phone +1(972) 644-5580
Fax +1(972) 480-7800
Email sc-infomaster@ti.com

Europe, Middle East, and Africa
Phone
- Deutsch +49-(0) 8161 80 3311
- English +44-(0) 1604 66 3399
- Español +34-(0) 90 23 54 0 28
- Francais +33-(0) 1-30 70 11 64
- Italiano +33-(0) 1-30 70 11 67

Fax +44-(0) 1604 66 33 34
Email epic@ti.com

Japan
Phone
- International +81-3-3457-0972
- Domestic 0120-81-0026

Fax
- International +81-3-3457-1259
- Domestic 0120-81-0036

Email pic-japan@ti.com

Asia
Phone
- International +886-2-23786800
- Domestic
  - Australia 1-800-881-011
    - TI Number -800-800-1450
  - China 10810
    - TI Number -800-800-1450
  - Hong Kong 800-96-1111
    - TI Number -800-800-1450
  - India 000-117
    - TI Number -800-800-1450
  - Indonesia 001-801-10
    - TI Number -800-800-1450
  - Korea 080-551-2804
  - Malaysia 1-800-800-011
    - TI Number -800-800-1450
  - New Zealand 000-911
    - TI Number -800-800-1450
  - Philippines 105-11
    - TI Number -800-800-1450
  - Singapore 800-0111-111
    - TI Number -800-800-1450
  - Taiwan 080-006800
  - Thailand 0019-991-1111
    - TI Number -800-800-1450
  - Fax 886-2-2378-6808
  - Email tiasia@ti.com

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products 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 APPLICATIONS 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, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

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

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty, or endorsement thereof.