An SVIC to 68020 Arbiter Design

Introduction

VME board functionality and their interfaces vary quite widely from application to application. The most complex type of VME interface is a VMEbus System Controller, which has complete VME master and slave capability and is the VME Interrupt handler. There are many devices on the market that can satisfy this need and Cypress has devices that can perform this function, namely the VIC068A and VAC068A 32-bit VMEbus Interface Controllers. In addition to this, the VIC64 provides all the functionality of the VIC068A but with the addition of D64 VME block transfer capability.

However there are many applications that do not require the complexity of the VIC/VAC products. These VME boards might often be slave-only type applications. Cypress has introduced the Slave VIC devices (SVIC for short), the CY7C960 and CY7C961. These devices are simple VME interface controllers, without having any of the complexity of being a VME System Controller or VME Interrupt Handler. The CY7C960 is a slave and the CY7C961 is a slave with DMA master.

Typical applications for slave-only products are memory boards and I/O boards. Memory boards can be as diverse as SRAM, DRAM, UVEPROM or FLASH EPROM (in, say, solid state mass storage). The I/O type applications could be for Ethernet, SCSI, FDDI, MIL STD 1553, RACE, Parallel/Serial I/O or even a VSB bridge. Memory boards do not require the use of a microprocessor, as they invariably rely on the VME master to initiate either a read or a write. Local timing and bank switching, etc., can be controlled with programmable logic devices (either CPLDs or FPGAs) and a microcontroller may also be needed.

Again, most I/O applications operate in a similar way to the memory card, in that reads and writes are initiated by the VME master. However, if there are several interfaces on the I/O card, then a local microprocessor may be useful for reducing the overhead of the main system processor. If the local processor could take over much of this overhead, such as pre-processing, then the VME master may only be required to extract data on a block transfer basis. Such a set-up could allow data to be transferred at up to 80 Mbytes/second.

This application note provides an example of how to design the arbiter between one of the SVIC devices and a microprocessor. It has been assumed that the local microprocessor is a Motorola 68020. The arbitration associated with this device is fairly standard with most of the Motorola processors. Also, the Motorola processors are well suited to the VMEbus, requiring some byte swapping for 8- or 16-bit transfers, but little else.

The SVIC Devices (CY7C960 and CY7C961)

Features List

- 80 Mbyte per second Block Transfer Rates
- VME64 compliance (A64, A40, A32, A24, A16)
- Auto Slot ID
- All standard VMEbus transactions implemented
- VMEbus Interrupter
- No Local CPU necessary
- Programmable from VMEbus or Serial PROM

- DRAM Controller including refresh
- Local I/O Controller
- Flexible VMEbus address scheme
- User-configured VMEbus personality
- Limited VME Master support (CY7C961 Only)
- TQFP, PQFP, CQFP packaging

Slave VIC Operational Overview

The Slave VMEBus Interface Controller (SVIC) provides the board designer with an integrated, full-featured VME64 Interface. This device can be programmed to handle every transaction defined in the VME64 specification (as a slave device). The SVIC contains all the circuitry needed to control large DRAM arrays and local I/O circuitry without the necessity of complex programmable logic to drive the timing. There are no registers to read or write and no complex command blocks to be constructed in memory. The SVIC simply fetches its own configuration parameters during the power-on reset period. After reset, the SVIC responds to VMEBus activity and local circuitry transparently.

The SVIC acts as a bridge between the VMEbus and the local DRAM, as well as the local I/O. The VMEbus control signals are connected directly to the SVIC. The VMEbus address and data signals are connected to address and data transceivers that are controlled by the SVIC. Typically, these are devices such as the FCT543T. The SVIC may also be seamlessly connected to the ideal companion device, the CY7C964 VMEbus Interface Logic Circuit from Cypress. For an A32/D32 application, there is one CY7C964 required per byte width of address and data. Thus a total of four devices are required—maximum. The CY7C964 provides a slice of data and address logic that has been optimized for VME64 transactions. As well as providing the required drive strength and timing for VME64 transactions, the CY7C964s contain all the circuitry needed to multiplex the address/data bus functions for multiplexed VMEbus transactions. The CY7C964 contains counters and latches needed during block transfer operations. It also contains the address comparators that are used in the board's Slave Address Decoder. For an A32 or larger application four CY7C964 devices are required. For A24/D32 applications, then, three CY7C964s and the SVIC are required. For A24/D16 applications, only two CY7C964s, the SVIC and an FCT543T (or equivalent) are required. For A16/D16 applications, only two CY7C964s and the SVIC are required.

VMEbus transactions supported by the SVIC include D8, D16, D32 (include unaligned transfers (UAT)), MD32, D64, A16, A24, A32, A40, A64 single cycle and block transfer reads and writes.

Figure 1 shows the internal blocks that comprise the SVIC. The architecture includes several functions that remove most of the VMEbus problems from the board designer's shoulders. All VMEbus signals are handled automatically. The user has to program the Region AM table during configuration and then the SVIC handles the transactions as defined by the table set-up. Local circuitry is simplified by the Refresh Controller, the DRAM Controller, and the output pattern table. Block transfers are supported by the local address controller together with the CY7C964 circuitry (if used). Local timing is determined during initial configuration and the handshaking is determined from the Data Byte Enable Controller. Local interrupts are supported through the VME Interrupt Interface. The SVIC contains an internal Power-On Reset circuit and also responds to the VME SYSRESET* signal.

Design Example

Introduction

The design example has been chosen as a typical example of a VME board design. *Figure 2* shows that the design is based on a Motorola 68020 microprocessor. The processor has boot software located in the Boot EEPROM. After setting up the stack and implementing the reset exception routine, the processor would normally jump to running code from the EPROM. This will allow the processor to set up the DUART, RTC and any other programmable functions within the peripherals. This may well include setting up the SVIC, even though this is normally performed by either a serial EPROM or, alternatively, via the VMEbus.

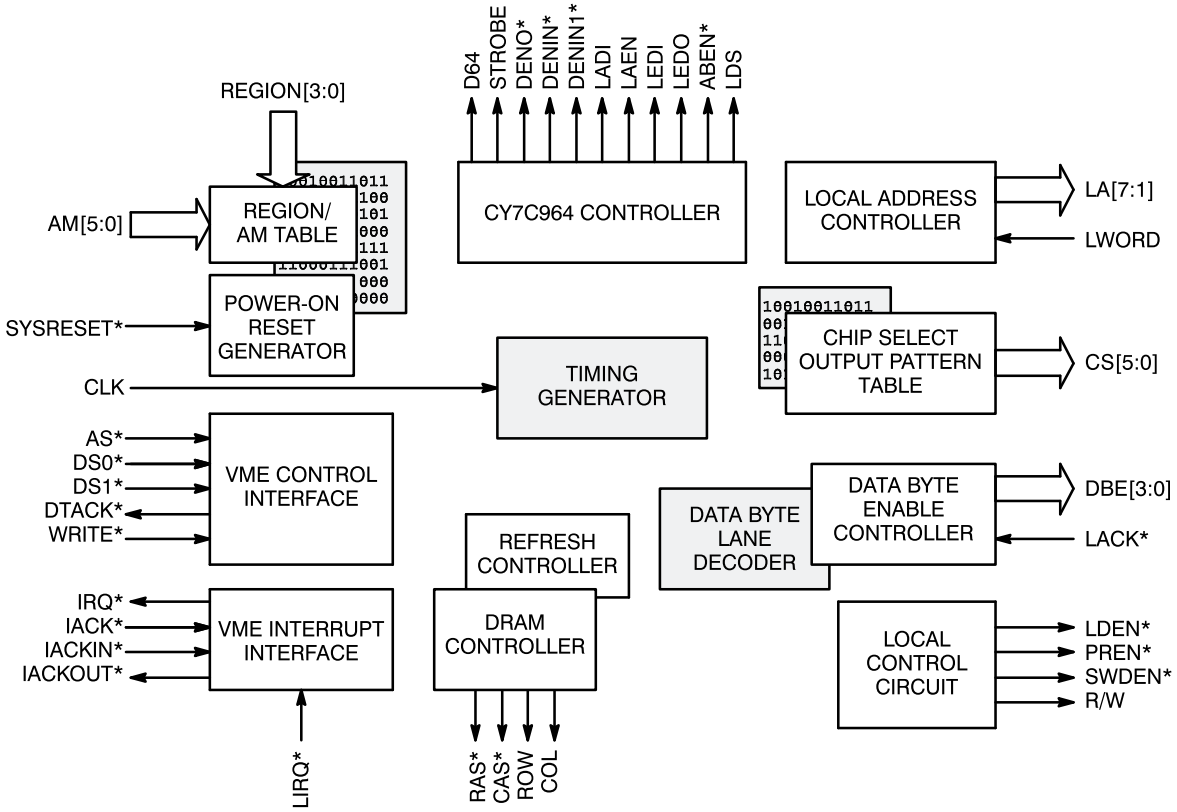

Figure 1. SVIC Block Diagram

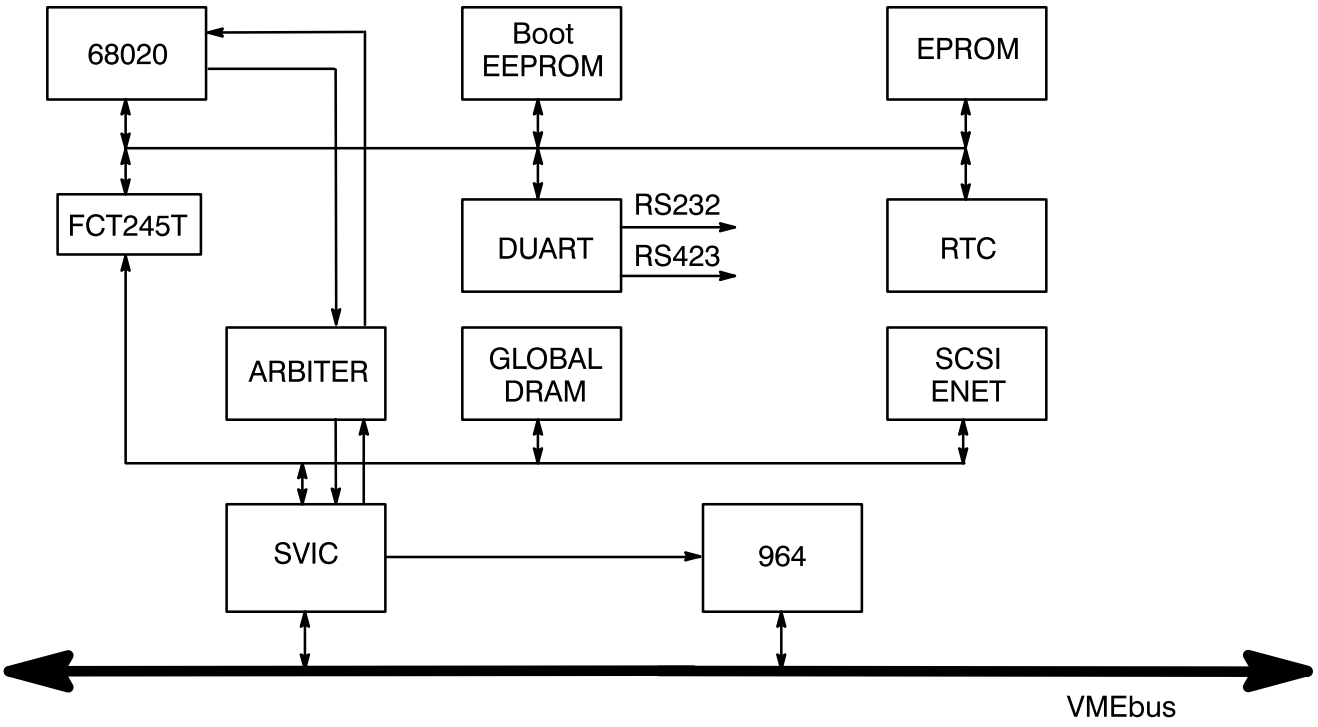

Figure 2. Typical VMEbus Design

There are two potential local bus masters in this design—either the 68020 or the SVIC. The task of arbitration (i.e., determining which master has control) is done by the Arbiter. The design is based on a single bus structure. The presence of the FCT245T devices reduces capacitive bus loading to maintain better performance.

Overview of the Motorola 68020

The Motorola 68020 was the first 32-bit implementation of the M68000 family of microprocessors from Motorola. The 68020 is object-code compatible with other members of the 68000 family. The non-multiplexed bus structure of the 68020 uses 32 bits of data and 32 bits of address. This lends itself very well to the VMEbus architecture, which is based on a 32-bit data and 32-bit address structure. For the purposes of data transfers, a D64 block transfer on the VMEbus is automatically split up into two 32-bit data transfers on the local bus, which keeps the 68020 compliant even in a D64 environment as provided by the SVIC.

The 68020 provides support for a dynamic bus sizing arrangement where the processor can transfer operands to or from devices while dynamically allowing the local bus logic to determine the port width for the 68020 on a cycle by cycle basis. This allows for access to devices of differing port width without the software engineer having to take special care over data alignment restrictions.

68020 Arbitration Methodology

Bus arbitration is the process in which a device on a bus may become bus master. The 68020 has a bus controller that controls the bus arbitration for the local bus that the processor sits on. This means that the 68020 has the lowest priority on the local bus. The design of the 68020 allows for a single bus master to be on the local bus at any one time. This includes an external device or the processor itself.

68020 Bus Arbitration Sequence

The bus arbitration sequence for the 68020 is:

1. An external device asserts the BR* signal.
2. The processor asserts the BG* signal to indicate that the local bus will become available at the end of the current bus cycle.
3. Once the local bus is released, the external device asserts the BGACK* signal back to the processor to indicate that it has assumed bus mastership.

The 68020 Bus Request Mechanism

Any devices on the local bus that are capable of becoming a local bus master must assert the BR* signal to the processor. The BR* signals from many potential bus masters can be arranged in a wire-ORd fashion even though they need not be open collector signals. (Rescinding three-statable signals are preferable to wire-ORd as the circuit does not rely on RC effects for the signal to drift up to an inactive level.) Once BR* has been asserted to the processor, this indicates that some external device wants control of the local bus. The design of the 68020 is such that it is always at a lower level bus priority than the external device that wants control of the bus and so the processor is compelled to relinquish the bus after it has completed its current cycle. If the BGACK* signal is inactive while the BR* signal is asserted, then the processor remains the bus master once BR* is negated. This feature reduces unnecessary interruptions in ordinary processing if the arbitration circuitry inadvertently responds to noise or if the alternate bus master decides that it doesn't need to be bus master before it has been granted bus mastership.

The 68020 Bus Grant Mechanism

The processor issues a bus grant in response to the bus request issued by the external device. BG* assertion immediately follows after internal synchronization. However, if the processor is performing a read-modify-write cycle or has already made an internal decision to perform a single bus cycle, then it must complete that operation first. During a read-modify-write cycle, the processor cannot assert the BG* signal unless the entire cycle has completed. The RMC* signal is asserted to indicate that the bus has been locked. When an internal decision has been made to execute another bus cycle, then

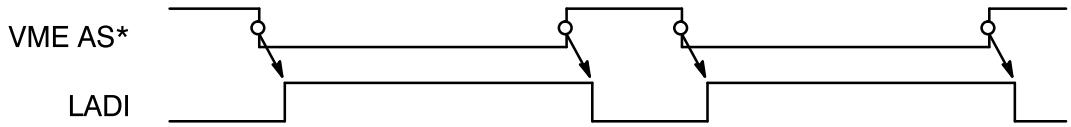

Figure 3. LADI with BUS HOLD OFF Disabled

the BG* cannot be asserted to the external device until the bus cycle has begun. The 68020 design allows the BG* signal to be routed through a daisy-chained network or, alternatively, through a priority encoded network such as an external arbiter. (The 68020 allows any kind of external arbiter as long as the arbitration sequencing is followed precisely.)

The 68020 Bus Grant Acknowledge Mechanism

Once the external device has received the BG* from the 68020, then it must wait until the local AS*, DSACK0*, DSACK1*, and BGACK* are negated before asserting its own BGACK* to the processor. The removal of the AS* signal indicates that the previous master has released the bus. The negation of the DSACK0* and DSACK1* signals indicates that the previous slave has terminated the cycle with the previous master.

The SVIC Local Bus Philosophy

The bus arbitration of the SVIC is much simpler than the 68020. This is known as the BUS HOLD-OFF feature of the SVIC.

The SVIC is intended to be the highest priority on the local bus. This implies that when a VME slave transaction occurs, then nothing will prevent the SVIC from reading or writing to local resources. Normally the SVIC starts a local cycle assuming that no other master may be in control of the local bus. This optimizes the response time of the SVIC by preventing the VME cycle being extended by local bus contention. This philosophy is not beneficial in all cases, such as where there is a local processor to consider. Some rudimentary control of the local bus shall be required from time to time by other devices.

SVIC Local Bus Arbitration Methodology

The SVIC can be prevented from starting a local cycle or a refresh of any local DRAM by using a BUS HOLD OFF function. To explain how this works, first consider the VMEbus activity. Without the bus hold function being enabled, whenever the AS* is asserted by the VME master, the SVIC will drive LADI HIGH and RAS* LOW (Row Address Strobe to DRAM) (see *Figure 3*). Then the VMEbus address is driven onto the local address bus under control of the SVIC. This happens for all VMEBus cycles whether the cycle is intended for the slave or not (the reason for this is to reduce bus latency).

When the BUS HOLD OFF feature is enabled, LADI is a 'local bus busy' signal. It indicates to the local arbitration logic that the SVIC has control of the local bus for either VME slave accesses or when the SVIC is performing DRAM refresh cycles.

As can be seen from *Figure 4*, the LADI signal goes HIGH when there is a VME AS* signal. If the cycle is not intended for the SVIC then the LADI signal is deasserted. It can be seen that LADI is also used to indicate to the local bus arbiter that a DRAM refresh cycle is taking place.

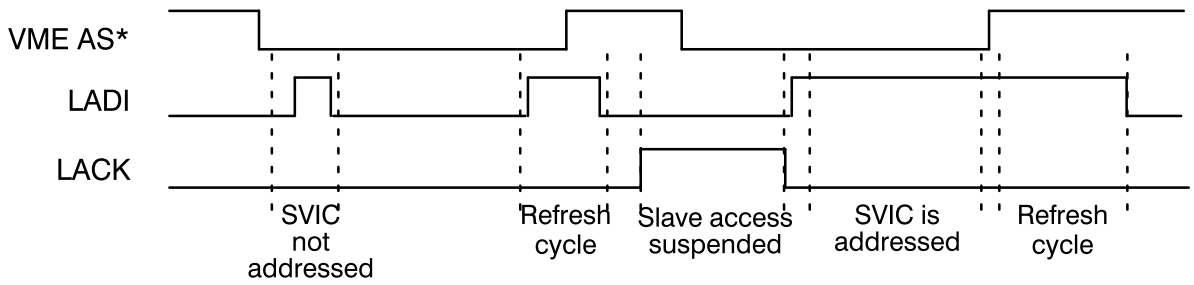

Figure 4. LADI with BUS HOLD OFF Enabled

The local bus arbiter monitors the LADI signal to determine when the SVIC does not have control of the local bus. Once the LADI signal is LOW, there is no current VME slave cycle or DRAM refresh taking place.

There are two scenarios that need to be considered for holding off the SVIC from further accesses:

1. Once the LADI signal goes LOW, the local arbiter is able to prevent the SVIC from regaining control of the local bus. As can be seen in *Figure 4*, the local bus arbiter sets the LACK signal to a '1' to 'hold off' the SVIC from regaining control of the local bus. If the LACK signal is set to a '1' by the **SECOND RISING EDGE** of the SVIC clock, then the local arbiter is guaranteed to have prevented the SVIC from getting control of the local bus.

2. The other condition for an alternate master gaining control of the local bus is when the LADI signal has been set LOW for greater than two clock cycles (i.e., when there is little VMEbus traffic). When the alternate master desires control of the local bus, the local arbiter drives the LACK signal to be a '1'. However, after **TWO RISING EDGES** of the SVIC clock signal, the arbiter must sample the LADI signal to make sure that it is still LOW. This takes account of potential metastable conditions as a result of a VME AS* being asserted to the SVIC at the same time as LACK is asserted.

When the local bus is not available to the SVIC, all VME slave transactions and will hold until the local bus is made available again. Once control of the local bus has been returned to the SVIC the refresh engine shall have priority and burst all the missed refresh cycles up to modulo 64. After this, the SVIC will respond to a pending VME slave request.

The 'bus hold off' function is enabled by a bit in the configuration bit stream. If the bit is not enabled then the SVIC cannot be prevented from performing DRAM refresh or from starting a local cycle. The function of LACK* is then simply to extend the completion of local cycles, allowing for slow local peripherals.

Design Considerations

There are certain special cases that the design engineer must consider when designing the SVIC into a VME board that can have more than one local bus master.

In the most basic applications where the SVIC is the only bus master, slave select logic is straightforward. *Figure 5* shows how this might be accomplished.

As can be seen from *Figure 5*, the three most significant address bytes are permanently enabled by connecting the LAEN inputs of the three most significant 964s to V_{CC}. This allows the VME addresses to flow directly from the VMEbus and onto the local bus. The region decoder then decodes the local addresses and the four REGION bits are fed directly into the SVIC. When a VME address appears which targets the VME board, one of the REGION bits becomes active which is then validated by the falling edge of VME AS*.

If the VME board design is such that there may be more than one bus master, then a more suitable arrangement can be seen in *Figure 6*.

Figure 6 shows that when the SVIC does not have control of the local bus, then the local addresses become isolated from the VME interface. The VME addresses are still monitored, however, by the region decoder. The output of the region decoder can then be used as the SVIC local bus request signals. These signals can be fed into the arbiter along with VME AS* to qualify the local bus request.

SVIC to 68020 Arbiter Design

The arbiter design represents a challenge to the designer. The reason for this is that the assumption of the SVIC is that it requires the highest priority and normally has control of the local bus all of the time. On the other hand, the 68020, which contains its own arbitration circuit, has the lowest priority.

The arbiter design must allow control of the local bus to default to the 68020. In addition the SVIC must not be allowed to take control of the local bus if there is any activity on the VME AS* signal unless the VME cycle is targeted towards the SVIC itself.

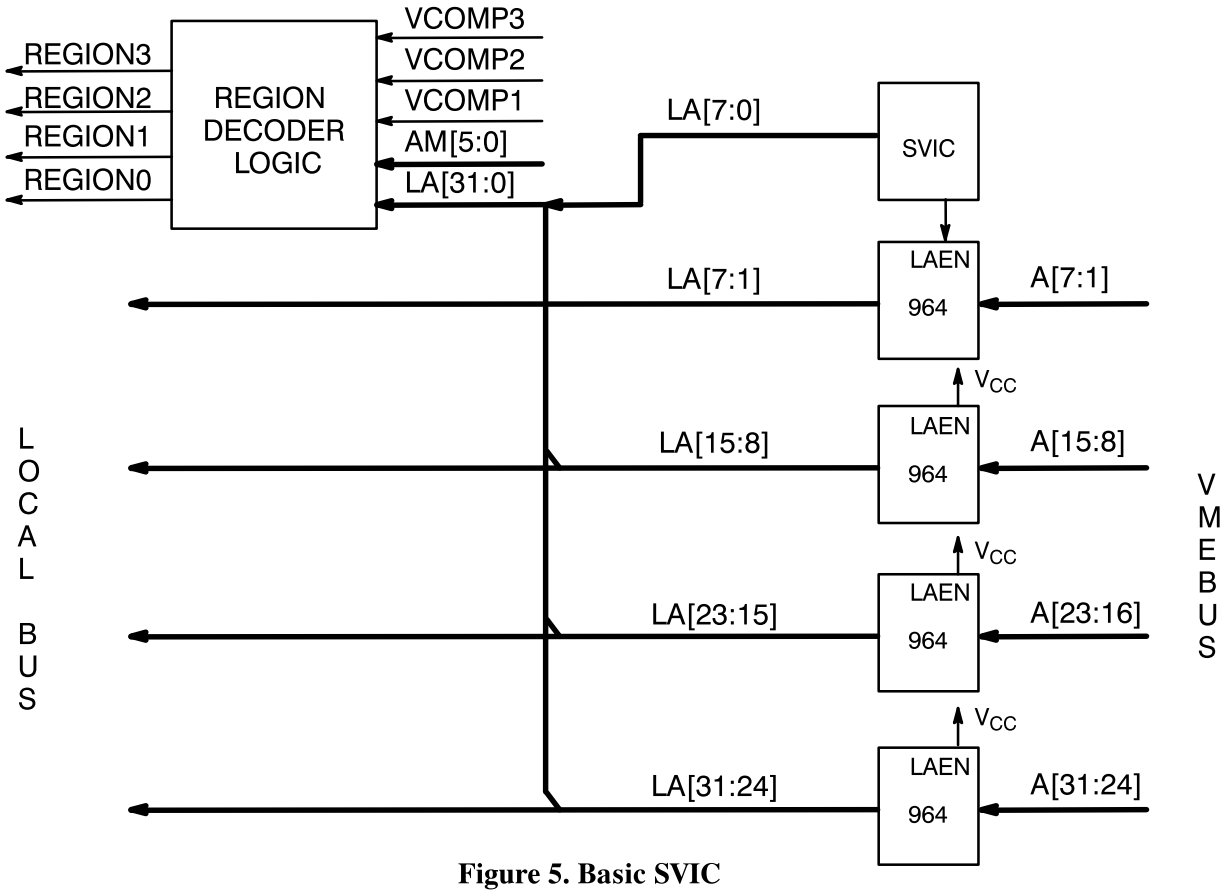

Figure 5. Basic SVIC

Coping with Metastable Events

The arbiter design is based on a state machine. The state machine is driven at the processor bus clock frequency of 20 MHz. The SVIC is driven at a higher frequency of 80 MHz. The design will require the use of RoboClock to keep the rising edges of the two clock frequencies aligned. This will greatly reduce the instances of metastability. The crystal oscillator required to drive the RoboClock will be 20 MHz (see *Figure 7*). This crystal oscillator frequency is a common frequency and is easily obtained from many crystal oscillator vendors. In addition, this frequency oscillator is easily available to military specifications.

Using RoboClock is one method of reducing potential metastable events by using clock edges that line up. There are signals, however, that are totally unpredictable as to when they arrive. One of these is the VME AS* signal. The VMEbus is totally asynchronous to both the SVIC 80-MHz clock and also the processor 20-MHz clock. To make sure that these types of signals don't make the arbiter metastable, one of two methods should be employed. One is to utilize a register that is resilient to being metastable, (i.e., it catches an event or doesn't); the other more straightforward method is to use double registering. This saves on board space and can easily be implemented in programmable logic devices. The FLASH370™ series of CPLDs supports double registering at the dedicated inputs.

Handling the DRAM Refresh

Once the SVIC has been put into holdoff mode, it has no way of indicating to the local logic that there are any pending DRAM refreshes. The SVIC can store up to 64 refresh events while it is held off, (if the number of pending refreshes exceeds 64 then the count will roll around to 0 again and 64 pending refreshes will be lost). Once the SVIC gets control of the local bus, it will initiate a burst of refresh pulses. The most straightforward way for the SVIC

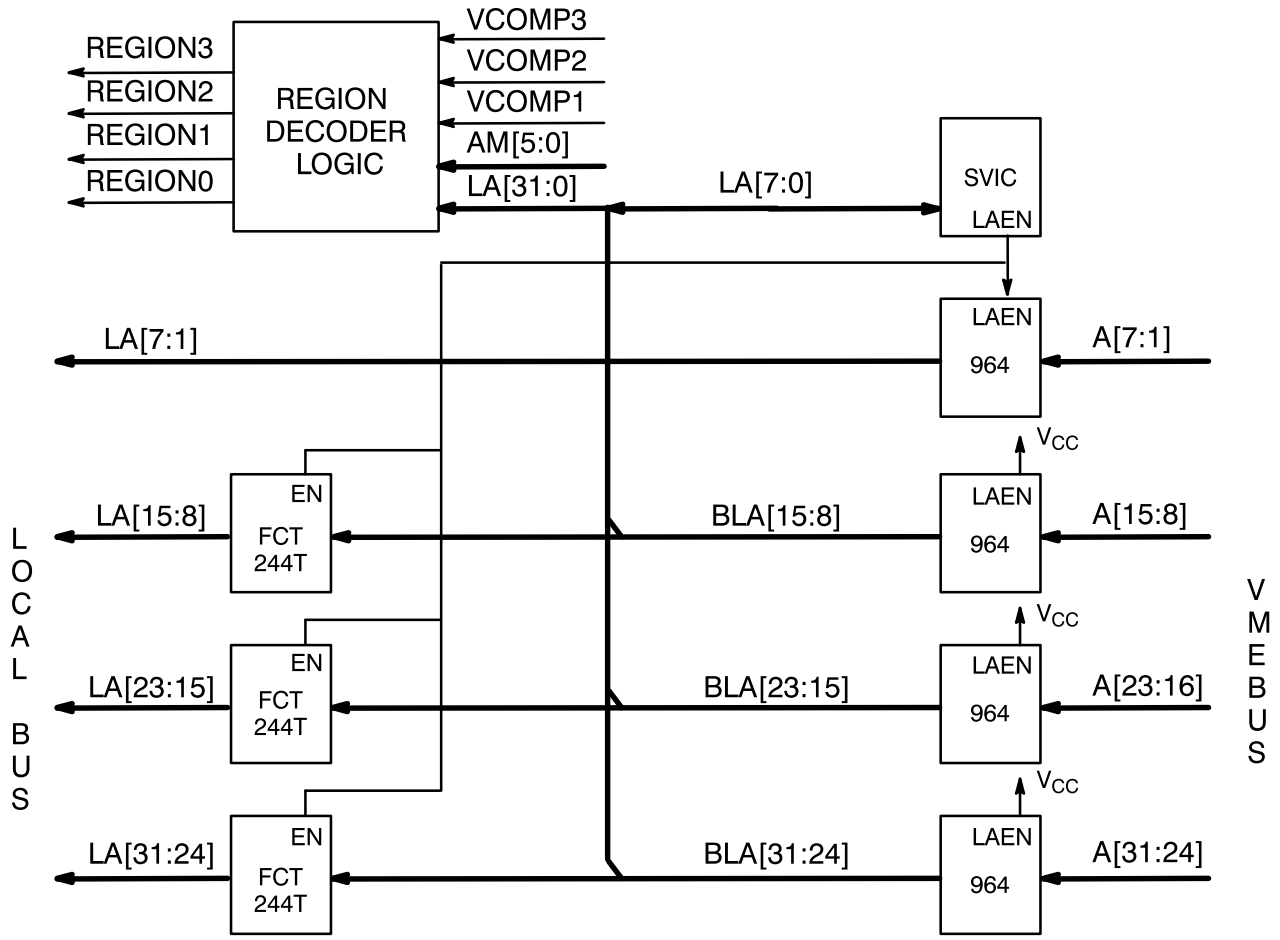

Figure 6. SVIC Implementation with More than One Bus Master

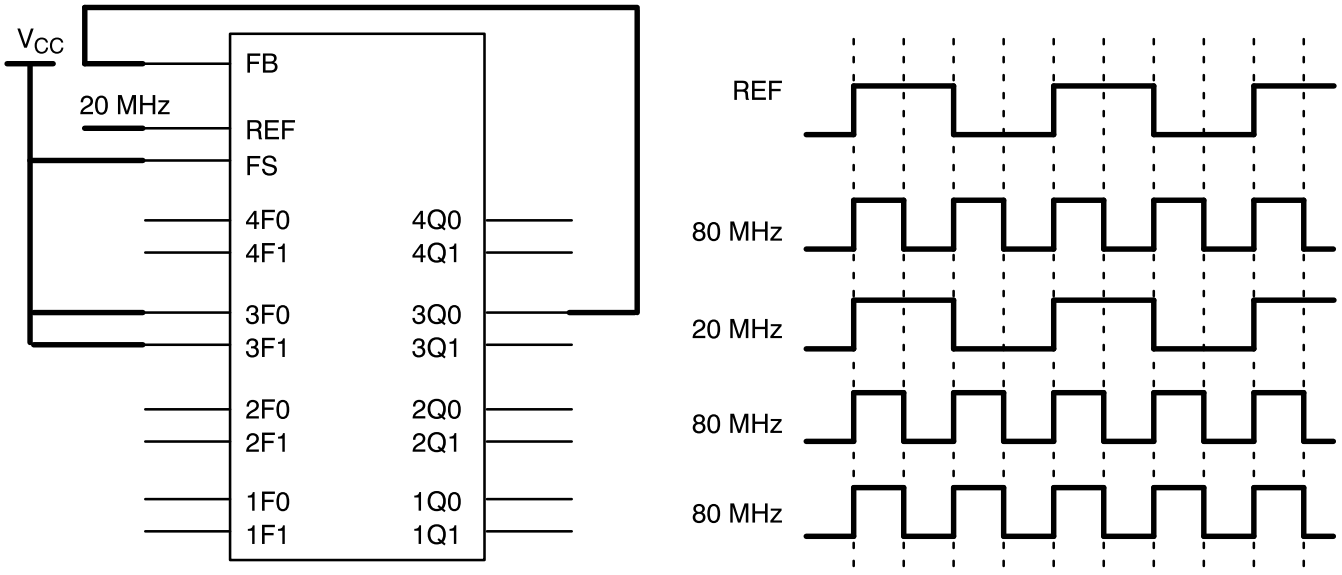

Figure 7. RoboClock Frequency

to get hold of the local bus is when a slave access takes place from the VMEbus. Once the SVIC has been granted control of the local bus, the SVIC will perform the pending DRAM refresh cycles as a higher priority. Once all of the pending refreshes have been done, then the VME master is allowed to proceed with the data transfer.

There is a case, however, when there are minimal VMEbus access requests to the SVIC. Such a situation would mean that the pending DRAM refresh cycles would build up without any chance of the SVIC of being granted control of the local bus. Hence part of the arbiter design requires the use of a counter timer that counts 125 µs. If there have been no VME cycles targeted towards the SVIC in this time (which is quite possible), then the arbiter needs to hand over control of the local bus to the SVIC and then monitor the LADI signal being inactive. Once LADI is inactive, this will indicate to the arbiter that the DRAM refresh cycles are complete and control can be taken from the SVIC. *Figure 8* shows the state diagram that is the basis of the state machine.

The source code for the design has been written in VHDL. The target device is a FLASH371−110 device. However if more registers and/or combinatorial logic is required for future upgrades or additions then the designer can migrate to a FLASH372 without having to change the real estate in the PCB that is already being used.

The flow of the state machine is shown in the timing diagram shown in *Figure 9*.

Appendix A shows the VHDL source code for the double buffering section. This was designed as hierarchical VHDL (the designer only has to instantiate the function as a single line of VHDL in the main code). This will be especially useful if a 25-MHz or 33-MHz 68020 is used. These frequencies are not a multiple of 2 so the clock domain of the 68020 and the clock domain of the SVIC (80 MHz) will be entirely asynchronous. The method of instantiating the double buffer saves time and effort.

Appendix B shows the main source code which contains the state machine design and also the DRAM refresh holdoff timeout counter.

Signal definition (*=active LOW)
BR* = bus request to the 68020
BG* = bus grant from the 68020
BGACK* = bus grant acknowledge to the 68020
SVICREQ* = SVIC requests local bus (VME cycle targets SVIC or refresh hold off times out)
SVICPROC* = SVIC granted local bus (VME cycle targets the SVIC or DRAM refresh hold off times out)
LACK* = SVIC bus grant (input to SVIC)
LADI* = SVIC bus busy (output from SVIC
dsacks = dsack0 AND dsack1
as* = 68020 address strobe

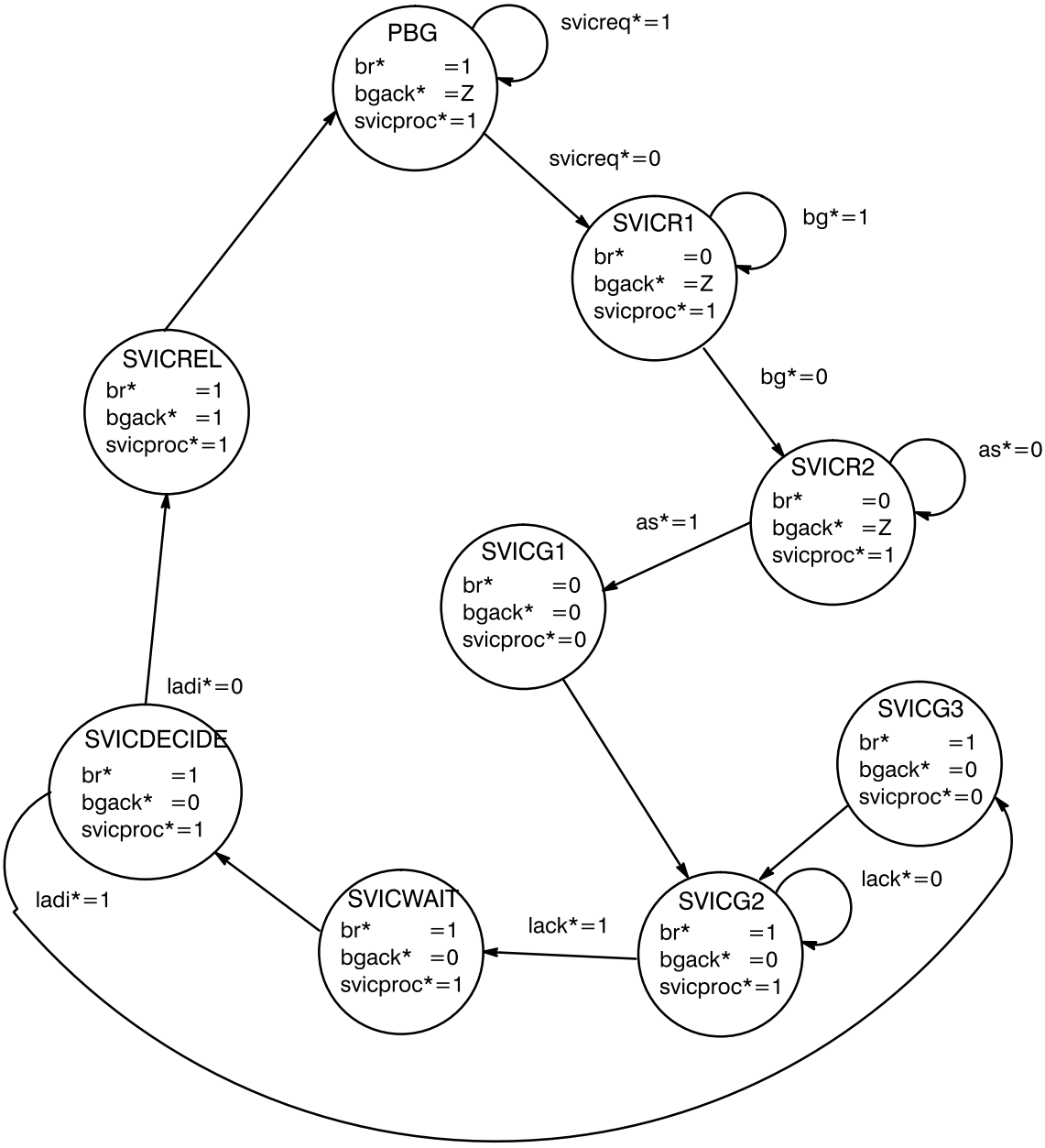

Figure 8. State Diagram of SVIC to 68020 Arbiter

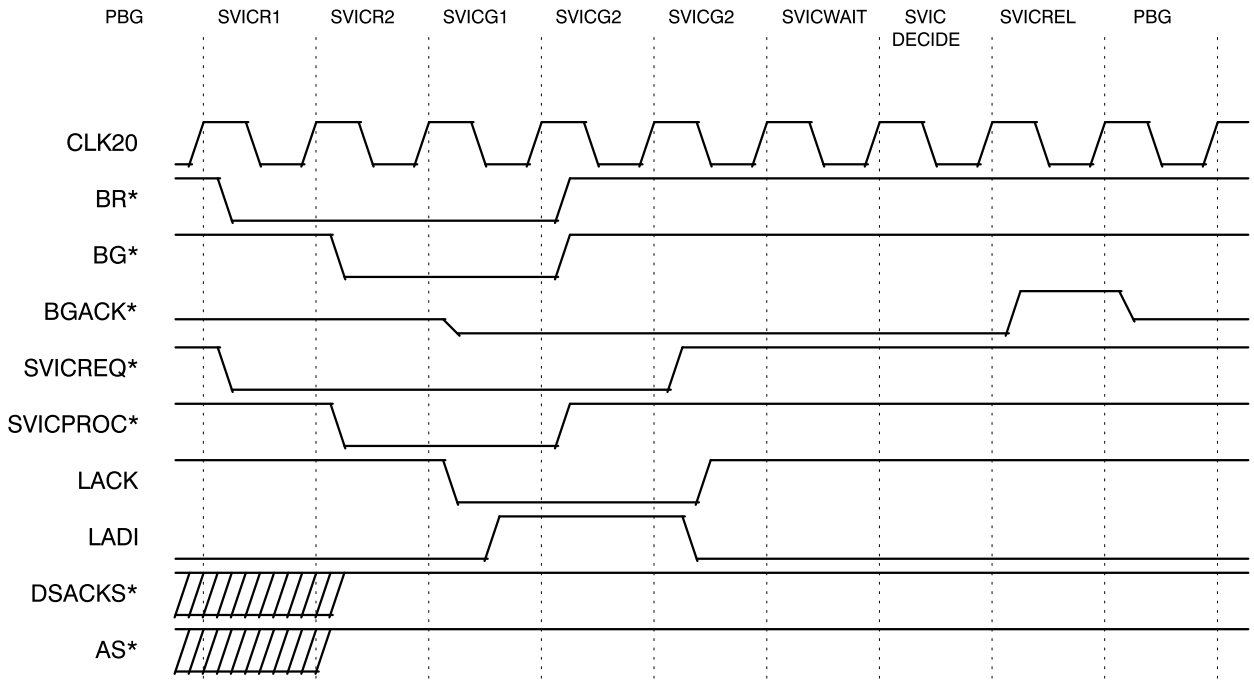

Figure 9. Timing Diagram with States

Appendix A. Source Code for Double Buffering

```
--This package description describes the double buffering technique
--for metastability hardening

PACKAGE sync_tools IS
  COMPONENT synchronise PORT(
  datain,clk:    IN BIT;
  dataout:   OUT BIT);
  END COMPONENT;
END sync_tools;

ENTITY synchronise IS PORT(

datain,clk:    IN BIT;
dataout:   OUT BIT);

END synchronise;

ARCHITECTURE archsynchronise OF synchronise IS

SIGNAL datain1:     BIT;

BEGIN

firstreg:  PROCESS (clk)

BEGIN

  IF clk'EVENT AND clk = '1' THEN
    datain1  <= datain;
  END IF;

end PROCESS firstreg;

secondreg: PROCESS (clk)

BEGIN
  IF clk'EVENT AND clk = '1' THEN
    dataout  <= datain1;
  END IF;

END PROCESS secondreg;

END archsynchronise;
```

Appendix B. Source Code for State Machine and Refresh Hold Off Timer

```
--************************************************************************
--************************************************************************
--**                                                                    **
--**     This design is an arbiter for the SVIC (960 or 961) and        **
--**     and the Motorola MC68020 (20MHz)                               **
--**                                                                    **
--************************************************************************
--************************************************************************

ENTITY arbiter IS PORT(

-- Port list for the 68020

  clk20:            IN BIT;    -- 20MHz Bus clock for the 68020
  dsack0,dsack1:    IN BIT;    -- data strobe acknowledge to 68020
  as:               IN BIT;    -- 68020 address strobe

-- Arbiter signals for the MC68020

  svicbg:           IN BIT;    -- 68020 bus grant to SVIC
  svicack:          INOUT X01Z;  -- SVIC bus grant to 68020
  svicbr:           OUT BIT; -- SVIC bus request
  outpen:           INOUT BIT; -- Output enable for bus grant

-- Arbiter signals for the SVIC

  ladi:             IN BIT;    -- latch address in (SVIC)
  lack:             BUFFER BIT;  -- local data acknowledge (SVIC)

--Port list for the 960

  reset:            IN BIT;    -- reset from reset handler
  clk80:            IN BIT;    -- SVIC 80 MHz clock
  vmeas:            IN BIT;    -- VME address strobe
  region:           IN BIT_VECTOR(2 DOWNTO 0)); -- local VME slave selects

END arbiter;

USE WORK.rtlpkg.ALL;
USE WORK.int_math.ALL;

-- The library sync_tools is a metastability hardening technique utilising
-- double buffering.

USE WORK.sync_tools.ALL;

ARCHITECTURE archarbiter OF arbiter IS

-- Definition of the states for the state machine controlling the
-- arbitration logic
```

Appendix B. Source Code for State Machine and Refresh Hold Off Timer (continued)

```
TYPE state_labels IS (pbg,svicr1,svicr2,svicg1,svicg2,svicg3,svicwait,
          svicdecide,svicrel);
SIGNAL state_bits:    state_labels;

SIGNAL svicreq:          BIT; -- SVIC request to arb logic
SIGNAL svicproc:         BIT; -- SVIC proceed from arb logic
SIGNAL bgack,bgackin:    BIT; -- Bus grant from/to controller direct
SIGNAL count256:         BIT_VECTOR(11 downto 0);  -- Refresh interval timer
SIGNAL co:               BIT; -- carry our from refresh timer
SIGNAL vmeasdel:         BIT; -- synchronised VME AS

--bgack is driven by the CPLD internally. SVICACK is tristate out and
--bgackin is monitored at pin and driven in to device.

BEGIN

--Instantiation pf bufoe to tristate bgack to 68020

  bf:  bufoe PORT MAP (bgack,outpen,svicack,bgackin);

  outpen <= '1' WHEN (state_bits=svicg1) OR (state_bits=svicg2)
           OR (state_bits=svicrel) ELSE '0';

--The following process drives the 68020 arbitration

arbcntrl: PROCESS (reset,clk20)

BEGIN

IF reset = '0' THEN
  state_bits <= pbg;
  svicbr   <= '1';
  bgack    <= '1';
  svicproc <= '1';

    ELSIF (clk20'EVENT AND clk20='1') THEN

    CASE state_bits IS

-- PBG is the idle state where the processor has been granted the bus.

    WHEN pbg   =>IF svicreq = '0'
             THEN state_bits <= svicr1;
               svicbr <= '0';
               bgack  <= '1';
               svicproc<= '1';
             ELSE state_bits  <= pbg;
               svicbr <= '1';
               bgack  <= '1';
               svicproc<= '1';
           END IF;
```

Appendix B. Source Code for State Machine and Refresh Hold Off Timer (continued)

```
-- SVICREQ1 is where the SVIC requires the bus but is waiting for bus grant
-- from the 68020

    WHEN svicr1  =>IF svicbg = '0'
             THEN state_bits  <= svicr2;
               svicbr <= '0';
               bgack  <= '1';
               svicproc<= '1';
             ELSE state_bits <= svicr1;
               svicbr <= '0';
               bgack  <= '1';
               svicproc<= '1';
           END IF;

-- SVICR2 is where the SVIC has been granted the bus but the 68020 is
-- still performing a bus cycle

    WHEN svicr2  =>IF as = '1'
             THEN state_bits  <= svicg1;
               svicbr <= '0';
               bgack  <= '0';
               svicproc<= '0';
             ELSE state_bits  <= svicr2;
               svicbr <= '0';
               bgack  <= '1';
               svicproc<= '1';
           END IF;

-- SVICG1 is where the the 68020 has completed its last cycle, the SVIC
-- has been granted the bus and the arbiter asserts bus grant to the 68020
-- and the SVIC is allowed to proceed

    WHEN svicg1  =>  state_bits <= svicg2;
               svicbr <= '1';
               bgack  <= '0';
               svicproc<= '0';

-- SVICG2 waits for the SVIC to terminate a session

    WHEN svicg2  =>IF lack = '1'
             THEN state_bits  <= svicwait;
               svicbr <= '1';
               svicproc<= '1';
               bgack  <= '0';
             ELSE state_bits  <= svicg2;
               svicbr <= '1';
               bgack  <= '0';
               svicproc<= '1';
           END IF;
```

Appendix B. Source Code for State Machine and Refresh Hold Off Timer (continued)

```
-- SVICG3 allows the SVIC to proceed again in the event of a metastable
-- condition where the SVIC misses the LACK* signal going inactive

    WHEN svicg3  =>  state_bits <= svicg2;
               svicbr <= '1';
               bgack  <= '0';
               svicproc<= '1';

-- SVICWAIT is a timing period before sampling LADI

    WHEN svicwait  =>  state_bits <= svicdecide;
               svicbr <= '1';
               bgack  <= '0';
               svicproc<= '1';

-- SVICDECIDE samples LADI. If LADI is inactive then the SVIC is in
-- hold off mode. If LADI is active then the arbiter failed to hold off
-- the SVIC


    WHEN svicdecide  =>IF ladi = '0' THEN
             state_bits <= svicrel;
               svicbr <= '1';
               bgack  <= '1';
               svicproc<= '1';
          ELSIF ladi = '1' THEN
            state_bits <= svicg3;
               svicbr <= '1';
               bgack  <= '0';
               svicproc<= '0';
          END IF;

-- SVICREL hands control of the local bus back to the 68020

    WHEN svicrel =>state_bits   <= pbg;
               svicbr <= '1';
               bgack  <= '1';
               svicproc<= '1';

-- The when others clause prevents implicit memory generation and copes
-- with any illegal states

    WHEN OTHERS  =>state_bits   <= pbg;
               svicbr <= '1';
               bgack  <= '1';
               svicproc<= '1';

    END CASE;

  END IF;
END PROCESS arbcntrl;
```

Appendix B. Source Code for State Machine and Refresh Hold Off Timer (continued)

```
-- The following process defines the counter that defines 128 uS
-- before control is given to the SVIC for the purposes of DRAM
-- refresh. Making the counter wider increases the time period by a factor
-- of 2 every time, but may make logic synthesis more difficult

cnt: PROCESS (reset,clk20)
BEGIN
  IF (reset = '1') THEN       -- asynch reset
    count256 <= x"000";

  ELSIF (clk20'EVENT AND clk20 = '1') THEN
    IF (state_bits = svicrel) THEN
      count256 <= x"000";
    ELSIF ((state_bits = pbg) AND (co =  '0')) THEN
      count256 <= inc_bv(count256);
    END IF;
  END IF;
END PROCESS cnt;

-- The co signal is used to inhibit the counter when it gets to the
-- terminal count

co<= '1' WHEN (count256 = x"9FF") ELSE '0';

--The following section defines the SVIC arbiter

--The VME AS* is asynchronous to the 80MHz clk so needs to be synchronised

sync1:   synchronise PORT MAP (vmeas,clk80,vmeasdel);

svicreq <= '0'  WHEN (((region /= "000") AND (vmeasdel = '0'))
    OR count256 = x"9FF") ELSE '1';

lack <= '0' WHEN (svicproc = '0')
    OR ((lack = '0') AND (ladi = '1')) else '1';

END archarbiter;
```