VIC64 to Motorola 68040 Interface

Purpose

This application note shows how the VIC64 can be interfaced to a Motorola 68040 microprocessor operating at 40 MHz. The issues and assumptions that go into designing such an interface are considerable and complex; thus, this application note will not attempt to design a complete VME board that can do everything. It will cover some of the issues that are pertinent when designing a 68040-based VMEbus board and will focus on the circuitry required for VIC64 to 68040 interfacing.

Design Issues

Asynchronous Bus (VIC64) to Synchronous Bus (68040) Interfacing

With the 68040 microprocessor, Motorola radically changed its bus architecture. With the 68030 and prior processors, Motorola used an asynchronous bus protocol. The 68040, on the other hand, uses a synchronous bus protocol. The VIC64, being an extension of the VIC068A architecture, retains the asynchronous bus protocol that is compatible with the 68030 and prior microprocessors. This makes the VIC64 and 68040 bus protocols incompatible.

For the most part, the VIC64 is a peripheral to the 68040. The 68040 generates read and write cycles to the VIC64 and the VIC64 responds. There is only one case where the 68040 would act as a peripheral to the VIC64 and that is if the 68040's snooping capability were turned on and the 68040 was required to supply data from its internal cache for a VIC64 cycle. To simplify the snooping interface, there are memory design strategies described later in this application note that can isolate memory accessed by the VIC64 from the 68040 internal cache. Thus, whenever the VIC64 were to act as master on the bus, the 68040 would never need to respond to a VIC64 cycle. No memory area that the VIC64 can access would be cached by the 68040.

To allow the 68040 and VIC64 to communicate, the VIC64 must be synchronized to the 68040. The primary signals that undergo this synchronization are the handshaking signals, DSACK0* and DSACK1*, that the VIC64 sends to the 68040 to indicate the completion of a register transfer or a VMEbus transfer.

Putting a "Slow" VIC64 on the 68040's Bus

The 68040 synchronous bus can transfer data at a rate of 1 transfer per 2 cycles of the 40-MHz bus clock when running in single-cycle mode. The transfer can either be a byte, word, or longword in length. This translates to 1 transfer every 50 ns. The VIC64 responds to a request for a data transfer to its internal registers no quicker than 67.5 ns. When the 68040 accesses the VMEbus via the VIC64, the transfer can be considerably slower since the VMEbus slave controls the progress of the transfer. To pace the transfer without losing data, the 68040 allows a slow peripheral to hold off on asserting $\overline{TA}$ until it has its data available on a read, or can accept data on a write. The interface designed in this application note synchronizes the DSACK1* and DSACK0* signals from the VIC64 and uses them to generate $\overline{TA}$ to the 68040.

Bus Contention – Peripheral Write after Read

When designing with a high-speed processor and a slow peripheral, bus contention is always a concern. Bus contention comes into play when a slow peripheral is being read by the processor in the current bus

cycle and in the next cycle, the processor executes a write. Typically, the slow peripheral cannot be disabled off of the bus before the processor begins driving the bus. The VIC64 to 68040 interface is no exception.

Figure 1 shows the timing of the contention. The VMEbus interface used in this application note is the full functional D64 VMEbus interface using the VIC64 and 3 CY7C964s as shown in *Figure 4* of the Cypress application note titled, "Using the CY7C964 with VIC." At the end of the 68040 cycle where data is read, it takes up to 5 ns for PAS*, DS*, and CS* to deassert (using a PALC22V10D−7), up to 23 ns from DS* deasserted to ISOBE* deasserted, up to 12 ns from ISOBE* deasserted to CISOBE deasserted, and then 7.5 ns for the '245 to disable assuming a 74FCT16245T is used. The next cycle can begin and write data can be presented to the bus as early as 5.25 ns after the BCLK following the cycle when PAS*, DS*, and CS* were deasserted. This creates over 15 ns of contention!

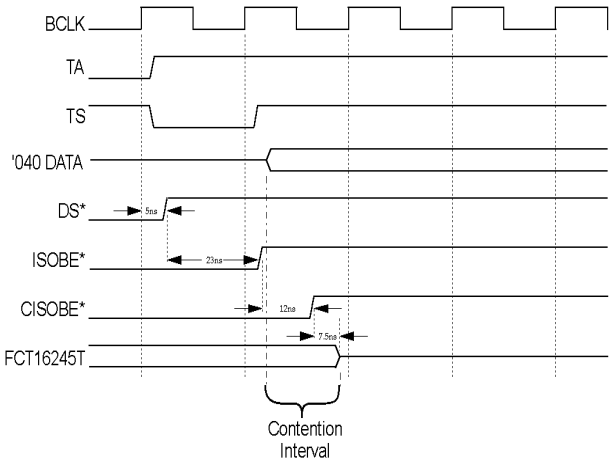

Figure 1. Contention for a Read Followed by a Write

Solving Bus Contention with Arbitration

The solution to the contention is easy considering the bus arbitration scheme of the 68040. In prior members of the 68k family, the processor also contained a bus arbiter on the same chip. Any peripheral that wanted to get access to the bus was required to request the bus from the processor. The 68040 relies on the designer to implement an external bus arbiter. All devices that can be masters on the bus must request the bus from the arbiter and the 68040 is no exception.

A way to eliminate the contention is to not allow the processor to begin a write cycle immediately after it has read the VMEbus or the VIC64 registers. The arbitration states of the 68040 make this possible. The timing of the arbitration is shown in *Figure 2*. At the beginning of the read cycle, the 68040 asserts $\overline{TS}$ along with an address that indicates either a VMEbus cycle or a VIC64 register access. The progression then is as follows:

1. The address is decoded and CS*, STROBE*, or MWB* is asserted along with PAS*.
2. The arbiter deasserts $\overline{BG}$ in response to CS*, STROBE*, or MWB* assertion. The 68040 will complete its current cycle. It is assumed that the 68040 does not want to relinquish the bus and will continue to drive $\overline{BR}$ asserted.
3. After receiving $\overline{TA}$, the 68040 is forced off the bus since the $\overline{BG}$ signal had been previously deasserted. However, when the arbiter sees that

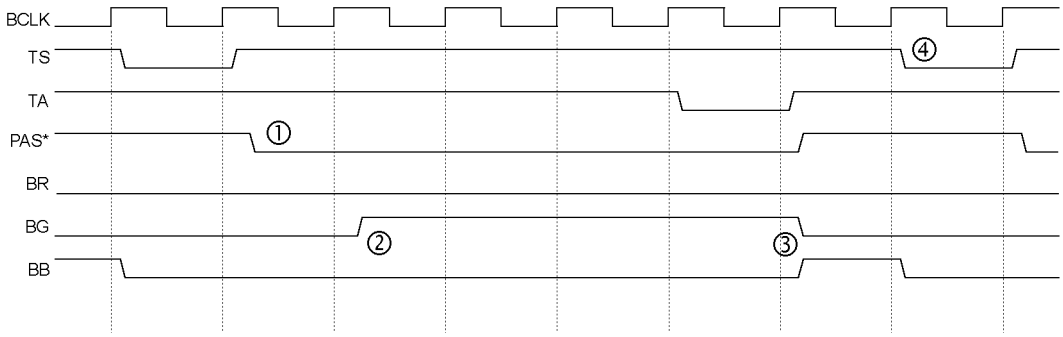

Figure 2. Arbitration Used to Eliminate Contention

$\overline{TA}$ has been asserted it grants the bus back to the 68040.

4. The 68040 on the next clock rising edge can assert a $\overline{TS}$ to begin the new cycle since $\overline{BG}$ is seen asserted.

With this method, there is no possibility of contention since 25 ns has been added to the contention resolution time.

For this method to be effective, good board layout and decoupling must be used. Taking the bus away from the 68040 causes its bus buffers to go high-impedance and then low-impedance in a single bus cycle. This can cause significant ground bounce and noise if the proper design practices are not used. Also, the signals that go high-impedance must be pulled up to V_{CC} to prevent them from floating.

Slave Access Implementation

Regardless of the memory map of the board, there are common issues pertaining to slave access of the board from the VMEbus. The slave interface is highly dependent on the function of the board. If the board is a memory array, chances are the board will primarily be accessed as a slave. However, if the board is a general purpose microprocessor, it will probably spend most of its time as a master on the VMEbus.

Since the slave interface is variable from board to board, the details of a slave interface to onboard circuitry will not be covered. The information in the *VIC068A/VAC068A User's Guide* and the *VIC64/CY7C964 Design Notes* contain ample information on using the VIC64 and CY7C964s for slave accesses. The next three sections address issues necessary for designing the slave circuitry on the board.

Bus Snooping

The 68040 can be configured to snoop cycles on its bus when it is not a master. Snooping is only a concern if the 68040 and the VIC64 share a common memory subsystem. If the VIC64 has its own dedicated memory which is gated off from the 68040's memory, snooping is not an issue (unless, of course, multiple bus masters reside on the bus with the 68040).

When the 68040 finds a cycle that requires data to be supplied from its internal cache, it will inhibit the memory subsystem and provide the requested data. The timing of this operation is synchronous to the BCLK and thus, if snooping were configured, the VIC64, when acting as a bus master, must have its signals synchronized to properly meet the 68040 timing.

Inhibiting Cache Transfers From Shared Memory

To avoid the timing difficulties that arise when snooping is enabled with a common memory subsystem, snooping can be disabled! This would also require that data areas on the board accessed by the VIC64 cannot be cached internally by the 68040. To disable caching of VIC64 register data or read VMEbus data, and disable snooping, accesses to the VIC64 and CY7C964 circuitry that generate STROBE*, CS*, and/or MWB* would cause $\overline{TCI}$, SC0, and SC1 signals to go to the 68040 in their inhibiting states. This would disallow the current cycle from being cached internal to the 68040.

To prevent the caching of data written to the board when the VIC64 is acting as a slave or a block transfer controller, the 68040's memory map decoder must assert $\overline{TCI}$ when any location the VIC64 can access when in that mode is requested by the 68040.

Memory Map Decoding and Remapping

Another design issue when implementing slave access logic is that of memory map decoding and remapping. When an address is provided from the VMEbus, it may not correspond to the same physical address on the board. Through the use of PLDs for decoding and shifting addresses, the VMEbus address can map to an on-board address.

Design Assumptions

Other than the design issues covered above, there are two assumptions that have been made in the design of the circuits herein. The first pertains to the memory system design and the second pertains to the buffer-type selection of the 68040.

Memory System Design

The goal in any memory system design is to match the performance of the memory to the masters that access it. This presents a problem in the design since the 68040 and VIC64 have vastly different bus structures. The 68040 is based on a synchronous bus and supports high-speed burst transfers as well as single-cycle transfers with all data and control signals synchronized to a common bus clock (BCLK). However, the VIC64 relies on asynchronous bus transfers that are paced by asynchronous data accesses and acknowledgements. There must be an assumption made by the board designer of one of the following memory strategies. Based on the typical application of the board, the designer can select a memory strategy to maximize data throughput. Two designs are presented here but many more are possible. In each case, the block labeled "VME Interface" contains the circuit shown in *Figure 4* of the Cypress application note titled, "Using the CY7C964 with VIC."

Two Memory Banks Architecture with No Caching of Shared Bank

Figure 3 shows a memory system design that is split into two separate banks. The first bank of memory is dedicated to the 68040 and runs synchronously. The second bank of memory is dedicated to the VIC64 and runs asynchronously. By having two separate memory banks, each can be designed to run optimally with its corresponding bus master. This would offer the best performance for the 68040 for its burst mode, and for the VIC64 for its burst mode. The gate between the two memory buses allows the 68040 access to the VIC's memory and to the VMEbus for single-cycle transfers. Access to the VIC64's memory bus is controlled by the arbiter and is granted to the 68040 when the VIC64 is not active on its bus.

Under normal operation, the gate opens when requested by the 68040, allowing the 68040 free access onto the VIC64's memory bus and onto the VMEbus. Only when the VIC64 is accessed as a slave or it is controlling burst transfers would the gate be closed. The VIC64 would request access to its bus via its LBR* signal. A memory configuration like this would allow both the VIC64 and the 68040 the most bandwidth on their respective busses. Both could be operating as bus masters at the same time. The application note titled "Interfacing the CY7C611A with the VIC64" uses this type of

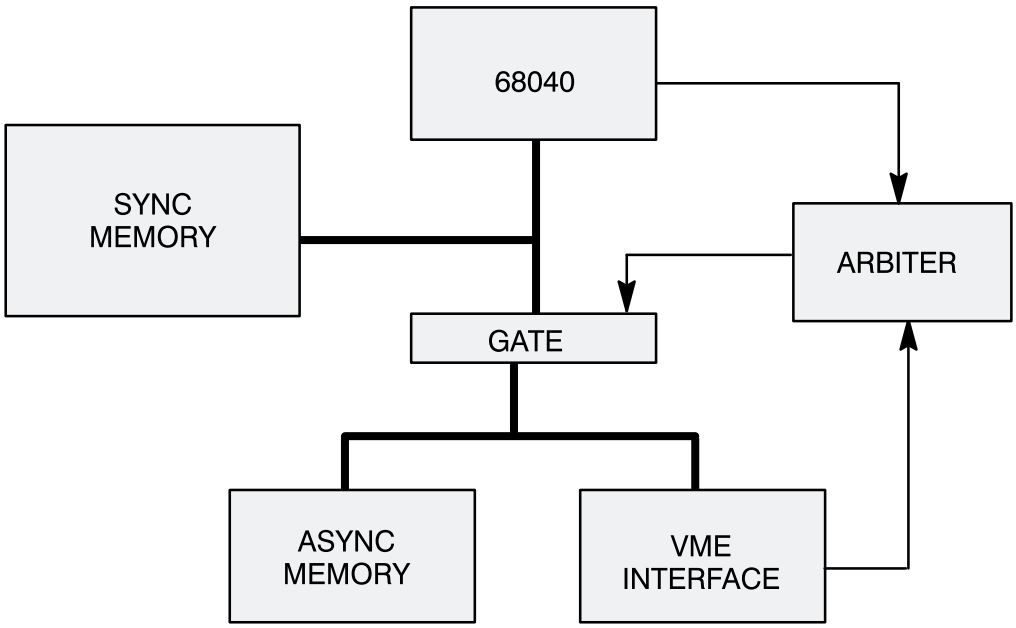

Figure 3. Two Memory Banks Architecture

memory scheme. The only caveat with this type of memory scheme is that when the 68040 is accessing the VIC64's memory, the data and acknowledgements from the VIC64 or its memory must be synchronized to the 68040's bus requirements.

Shared Memory with No Caching of VME Area

The other type of memory subsystem would be one that can act synchronously or asynchronously depending on whether the 68040 or the VIC64 was on the bus. This is illustrated in *Figure 4*. Both the 68040 and the VIC64 would share the same address and data buses and the arbiter would be used to grant access to one or the other. The arbiter could also indicate to the memory subsystem who has access to the bus. Although this simplifies the bus structure, it could complicate the memory design. It could also limit the bandwidth of the 68040 and the VIC64 to unacceptable levels.

However, this memory design might be perfectly acceptable for certain applications. The VIC068A and earlier 68K-family processors were able to share the same bus due to their compatible bus structures. Many designs allowed both the VIC64 and, for example, a 68020 to share bus bandwidth without detrimental effects. It is assumed that since this design revolves around a 68040 at 40 MHz, bus bandwidth for the processor is important! For this application note, it will be assumed that the separate memories strategy is used.

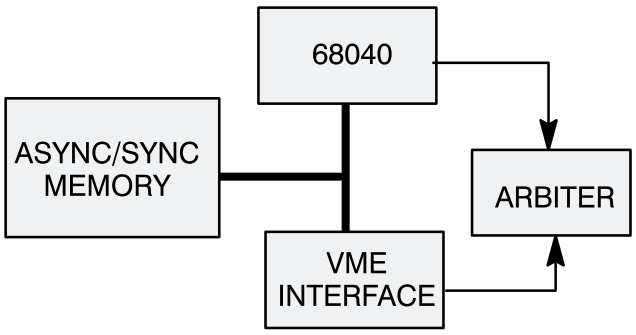

Figure 4. Single Memory Bank Architecture

68040 Configured for Large Buffer Timing Mode

To simplify the timing analysis and insure the peak performance from the design, the 68040 will be used in its Large Buffer Timing mode. This will require the careful layout of the board and the use of signal terminations to prevent adverse results from transmission line effects. Large Buffer mode is entered into during processor reset by pulling the $\overline{IPL2}$, $\overline{IPL1}$, and $\overline{IPL0}$ signals to a logic-one state.

Reset Circuitry

The reset circuitry and its routing is shown in *Figure 5*. There are three possible sources for a reset in this design. The first is a power-up or front panel pushbutton reset. The second is a reset initiated by the VIC64. The third is a 68040-initiated reset. The Reset PLD, a CY7C335-83, controls the sequencing for each of these reset types. The VHDL code describing this PLD is given in Appendix A.

Power-Up or Pushbutton Reset

The timing for the power-up or pushbutton reset is shown in *Figure 6*. While PWRUP_RST_N is LOW from either the pushbutton being depressed or the capacitor in *Figure 5* charging at power-up, BRD_RST_N_OUT is LOW. The capacitor and resistor values are chosen to guarantee that the clock and the board V_{CC} are stable when the rising edge of PWR_RST_N occurs. This insures that the VIC64 will be reset properly with a global reset. When the rising edge of PWRUP_RST_N occurs, the IRESET* signal is pulled LOW to the VIC64. The VIC64 responds with RESET* LOW, which in turn causes IPL0* to be pulled LOW, thus beginning a global reset. The IPL0* signal is then returned HIGH and, after a delay, IRESET* and BRD_RST_N_OUT are brought HIGH, ending the reset.

68040 Mode Selection

The 68040 is reset via the $\overline{RSTI}$ signal. For a valid reset to occur, the $\overline{RSTI}$ signal must be held LOW for a minimum of 10 BCLK cycles. The operation of the Reset PLD guarantees that $\overline{RSTI}$ will be held LOW for greater than this minimum amount of time. On the rising edge of $\overline{RSTI}$, the 68040 reads

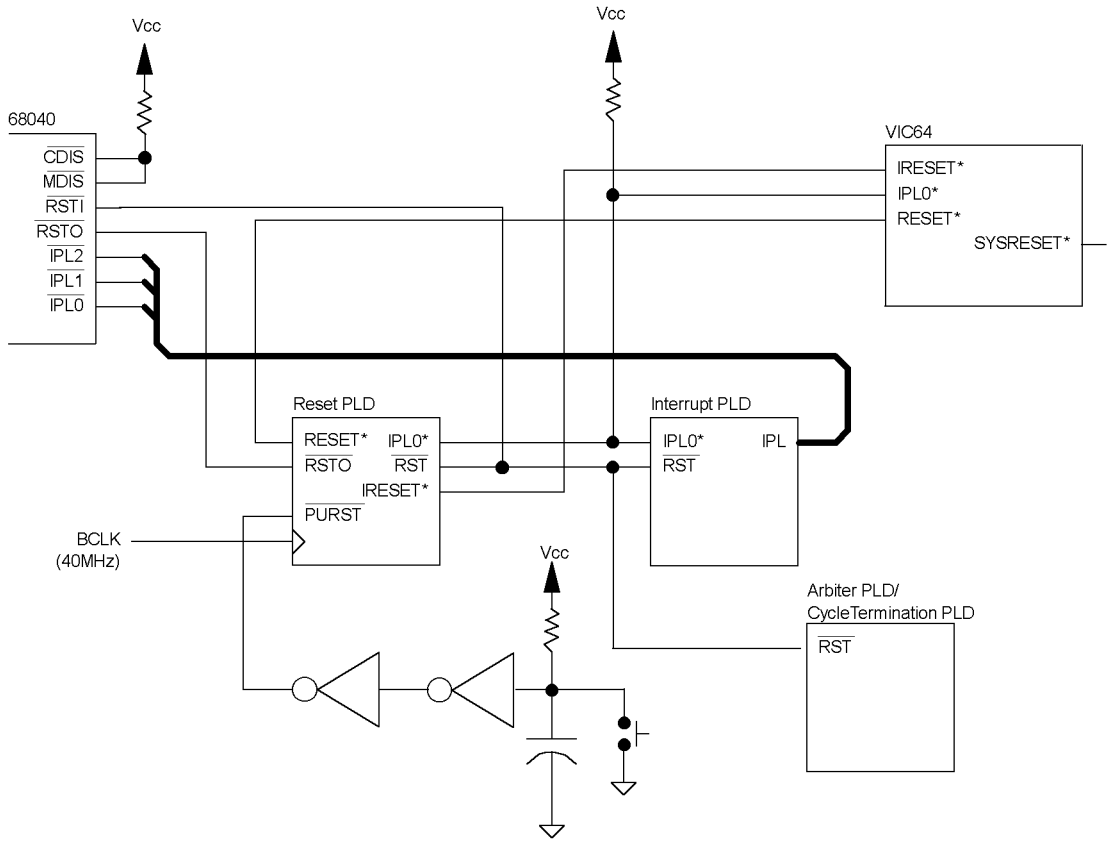

Figure 5. Reset Circuitry

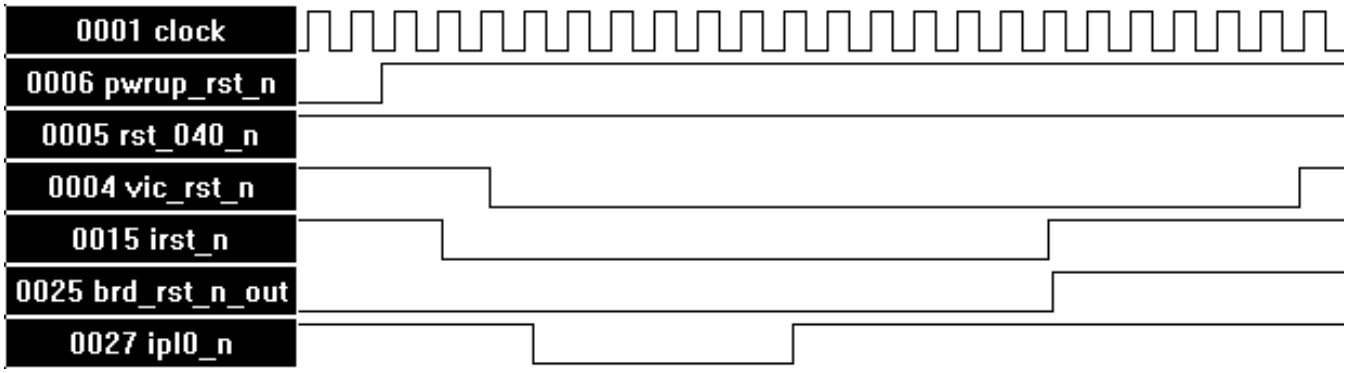

Figure 6. Power-Up or Pushbutton Reset

the current state of the $\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$, $\overline{MDIS}$, and $\overline{CDIS}$ and sets the mode of operation of the 68040.

The $\overline{CDIS}$ and $\overline{MDIS}$ signals are both pulled HIGH through a resistor that, at reset, disables Multiplexed Bus mode and Data Latch Enable mode. During normal operation, pulling $\overline{CDIS}$ and $\overline{MDIS}$ HIGH enables the internal cache of the 68040 and enables its internal MMU. The $\overline{IPLx}$ signals are all pulled HIGH at reset also via the Interrupt PLD. This enables the Large Buffer Timing mode for the data, address, and control signals.

VIC-Initiated Reset (SYSRESET* Active or SRCR Written)

The timing for a VIC-initiated reset is shown in *Figure 7*. A reset from the VIC is caused by one of two events. If the SYSRESET* signal on the VMEbus is driven active, the VIC64 will respond with the RESET* driven active. The VIC64 will also issue a RESET* if the SRR ($E3) is written with a value of $F0. This will cause the Reset PLD to force a full board reset via the BRD_RST_N_OUT signal and a global reset to the VIC64 with the IPL0* and IRESET* signals.

Support for 68040 RESET Instruction

The 68040 has an instruction, RESET, that forces its $\overline{\text{RSTO}}$ signal LOW for 512 BCLK cycles. The internal state of the 68040 is unaffected during this interval, which makes this instruction good for resetting board peripherals during normal processor operation. The implementation in this design, however, forces the board to be reset when the $\overline{\text{RSTO}}$ signal is activated. When the 68040 sees the $\overline{\text{RSTI}}$ signal active during an $\overline{\text{RSTO}}$ LOW interval, it immediately negates $\overline{\text{RSTO}}$ and forces a processor reset. The timing of this reset is shown in *Figure 8*.

Bus Arbitration

Bus Arbitration State Machine

The state machine for the bus arbitration is shown in *Figure 9*. There are essentially three arbitration states in the machine with a fourth being the reset state. The task of the arbiter is to grant access to the VIC64's private bus (*Figure 3*). Thus, it will normally allow the 68040's $\overline{\text{BG}}$ signal to remain active at all times. In fact, the arbiter does not even consider the state of the $\overline{\text{BR}}$ signal from the 68040 in the arbitra-

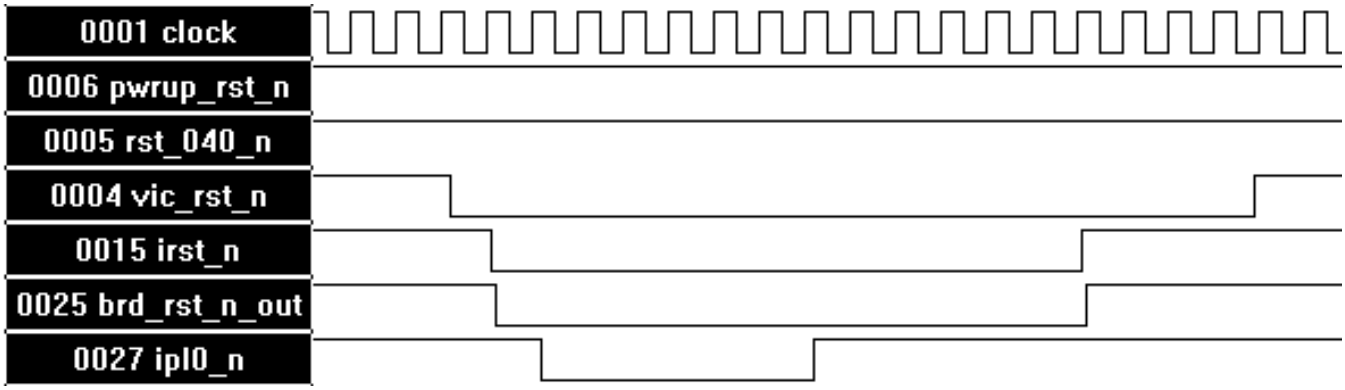

Figure 7. VIC64-Initiated Reset

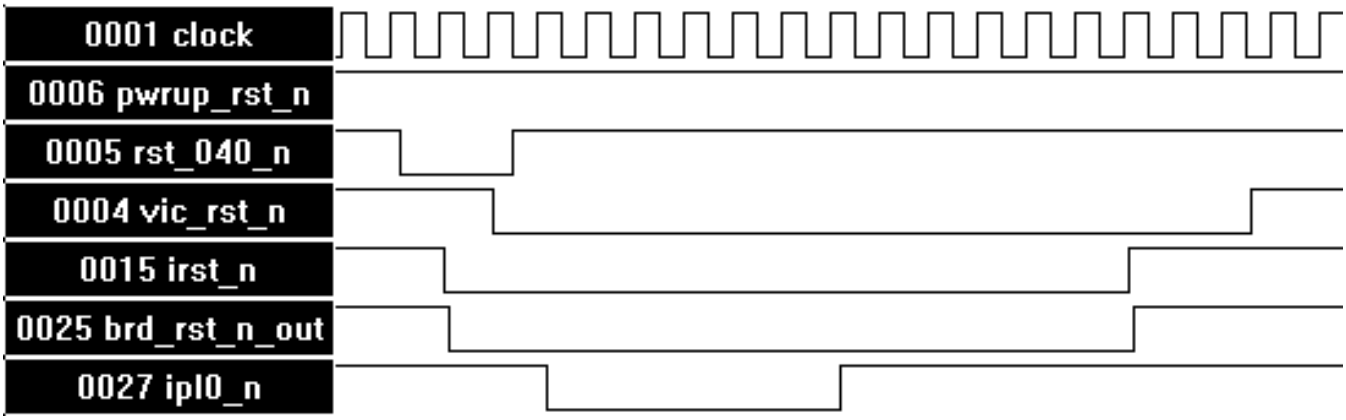

Figure 8. 68040-Initiated Reset

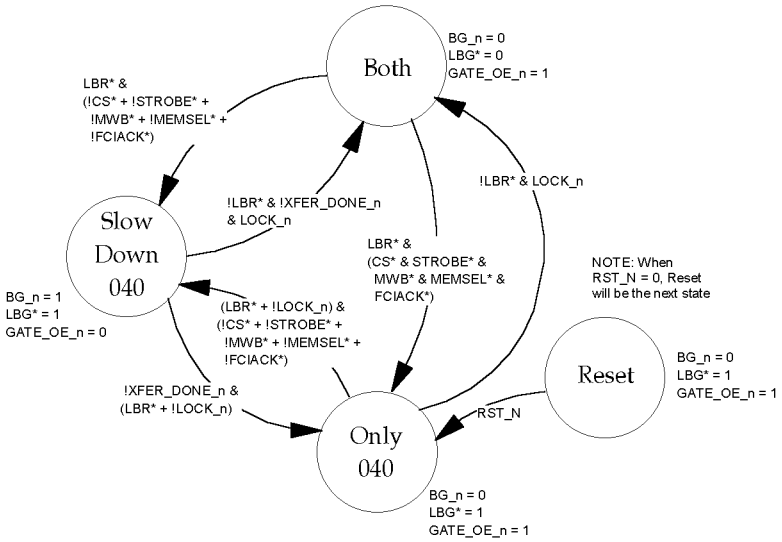

Figure 9. Bus Arbitration State Machine

tion. Rather, the state of the FCIACK*, MEMSEL*, CS*, STROBE*, or MWB* signals determine if the 68040 requires access to the VIC64's bus for either memory access, VIC64 or CY7C964 register access, or VMEbus access.

After a board reset has completed, the state machine transitions from the *Reset* state to the *Only_040* state. In this state, the 68040's $\overline{BG}$ signal is active, granting the 68040 access to its private bus. The LBG* to the VIC64 is inactive and the GATE_OE_N signal is also inactive. The GATE_OE_N signal is used to open the gate between the 68040's bus and the VIC64's bus. This "gate" consists of '245-type bidirectional drivers between the data busses and '244-type drivers for the 68040's address and control signals. It is suggested that FCT-C speed gates be used to insure that the data and address signals from the 68040 are driven to the VIC64 and/or the VMEbus with adequate setup time to DS* and PAS*.

The bus arbitration state machine is implemented in a CY7C335−83 and is named the Bus Arbitration PLD. The VHDL code describing this PLD is in Appendix D. The PLD and its connections within the circuit are shown in the schematic in *Figure 18*.

68040 Request for VIC64 Bus Access

There are two states from which the 68040 can gain access to the VIC64's bus, the *Only_040* state and the *Both* state. From the *Only_040* state, the 68040 would attempt access to the VIC64 bus with either the FCIACK*, CS*, STROBE*, MWB*, or the MEMSEL* going active. Only one of the signals would go active in a given access cycle. If the LBR* from the VIC64 is not active, the 68040 is granted access to the VIC64's bus by transitioning to the *Slow_Down_040* state. Another possible transition into the *Slow_Down_040* state from the *Only_040* state is if the 68040 is currently in the middle of a read-modify-write cycle, indicated by the $\overline{LOCK}$ signal being active. Regardless of the state of LBR*, if $\overline{LOCK}$ is active, the read-modify-write cycle is allowed to continue before the VIC64 can gain control of its bus.

Once in the *Slow_Down_040* state, the BG to the 68040 is driven inactive (to cause the 1-cycle delay in the 68040 bus cycle as described above) and the GATE_OE_N is driven active. When the current cycle completes as indicated by the XFER_DONE_N signal going active, the state machine transitions to either the *Both* state or the *Only_040* state depending on the state of the LBR* and $\overline{LOCK}$ signals.

If the VIC64 currently has ownership of its bus and the 68040 requests the VIC64's bus, the 68040 will not be granted access until the LBR* from the VIC64 has gone inactive. This could pose a problem if the VIC64 were in the midst of a block transfer. The 68040 might not receive ownership of the VIC64 in a timely fashion. Although not implemented in this application note, a bus timeout could be implemented to cancel the 68040's attempt to access the VIC64's data bus, or a method of testing the BLT* signal before initiating a cycle could be used.

VIC64 Bus Requests

The VIC64 requests ownership of its bus via the LBR* going active. If the active state is *Only_040* and the 68040 is currently not in the middle of a read-modify-write cycle (LOCK inactive), the state machine will transition to the *Both* state. In this state, the 68040 will have access to its bus, the VIC64 will be granted access to its bus, and the gate between the two buses will be closed. If the active state is *Slow_Down_040*, indicating that the 68040 currently owns the VIC64's bus, the VIC64 will not be granted its bus until the 68040 finishes its current cycle (assuming that the cycle is not the first half of a read-modify-write cycle). When the cycle completes, the state machine will transition from the *Slow_Down_040* state to the *Both* state.

Once in the *Both* state, the state machine will not transition until the VIC64 finishes its current cycle and releases its bus by driving the LBR* signal inactive. If the 68040 is attempting access to the VIC64's bus via the CS*, STROBE*, MWB*, or MEMSEL* signals, the state machine will transition to the *Slow_Down_040* state; otherwise, it will transition to the *Only_040* state.

Sample Arbitration Timing Diagrams

Figure 10 is a sample arbitration timing diagram. As the state machine exits the *Reset* state, $\overline{BG}$ is active, and GATE_OE_N and LBG* are inactive. When the 68040 attempts access to the VIC64's bus with the MEMSEL* signal, it is granted access and the $\overline{BG}$ signal goes inactive and the GATE_OE_N goes active. During the access, the LBR* signal goes active signifying that the VIC64 wants access to its bus. It is granted access (LBG* goes LOW) after the 68040's cycle completes with the XFER_DONE_N signal pulsing active.

During the VIC64's active time on its bus, the 68040 attempts access to the VIC64's bus via the MWB* signal. The 68040's cycle does not begin until the

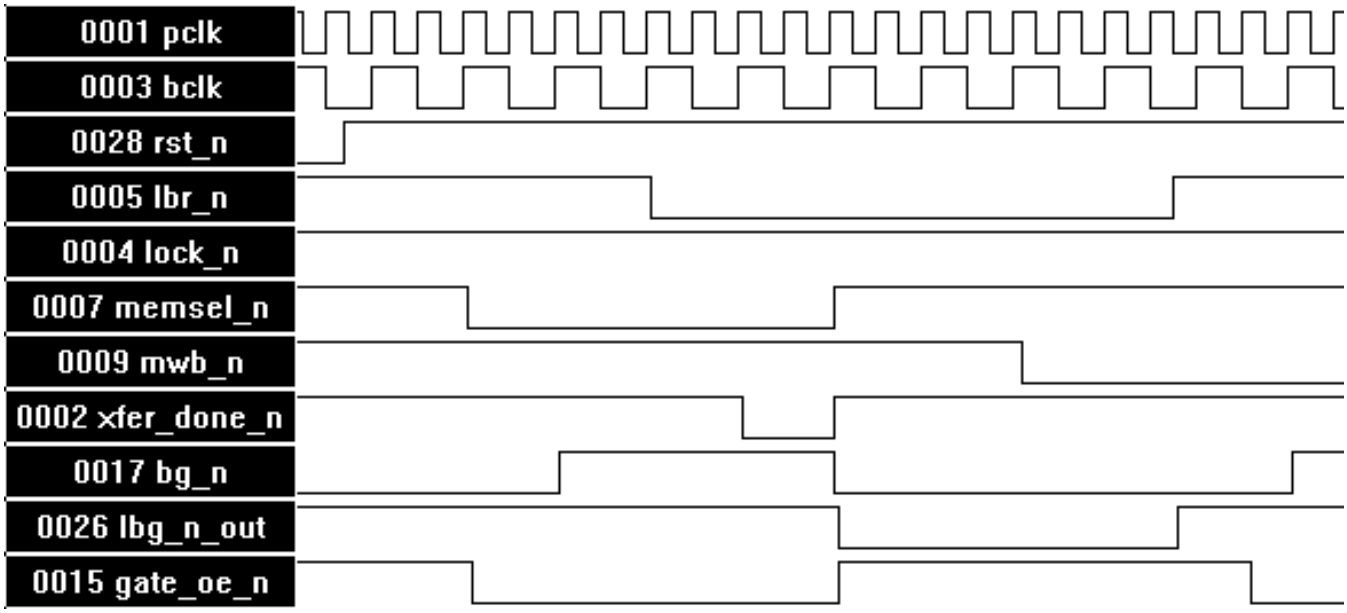

Figure 10. Arbitration Timing Diagram 1

LBR* signal is driven inactive, at which time the LBG* signal is driven inactive along with $\overline{BG}$. The *gate_oe_n* signal is driven active, allowing the 68040 onto the VIC64's bus.

Figure 11 is a continuation of *Figure 10*. The 68040 is completing its access with the MWB* signal and begins a read-modify-write cycle via the MEMSEL* signal. At the same time, the VIC64 requests access to its bus with the LBR* signal. In this case, the 68040 wins the arbitration and is allowed to complete the two cycles of the read-modify-write sequence. Once the sequence completes, the VIC64 is granted access to its bus until it deasserts the LBR* signal.

VIC64 and CY7C964 Register Access Cycles

VIC64 and CY7C964 register access cycles, as well as all other access cycles, are controlled by three PLDs. Two of the PLDs, the Address and Cycle Decode PLDs, control the initiation of a transfer. They are PALC22V10D-7s. The remaining PLD, the Cycle Termination PLD, controls the normal or abnormal completion of a cycle. This PLD is a CY7C335-83. The VHDL code for these PLDs can be found in Appendix B and Appendix C respectively. The PLDs and their connections within the circuit are shown in the schematic in *Figure 18*.

Selection of the PALC22V10D and CY7C335 Devices

The PALC22V10D and CY7C335 were chosen for a single, key reason. Both have a guarantee on their output data stability. The CY7C335 has a parameter, t_{OH}, that guarantees 2 ns of output data stability from the clock supplied to the part. The PALC22V10D-7 also guarantees a minimum on the t_{CO} specification of 2 ns. This is vital to the design because the 68040 running at 40 MHz requires that signals such as $\overline{TA}$, $\overline{TEA}$, $\overline{TBI}$, $\overline{TCI}$, etc., have a hold time of 2 ns from the rising edge of BCLK.

Selecting the VIC Registers vs. the CY7C964 Registers

Both the VIC64 registers and the CY7C964 registers are mapped into the same base address of A31−A28 = "0001." To make the determination between register sets, the lowest-order address bits, A00 and A01, are used in conjunction with the size signals, SIZ0 and SIZ1. When a byte transfer is requested and the lowest address bits are both HIGH, this is decoded as a VIC64 register access. When a longword transfer is requested and the lowest ad-

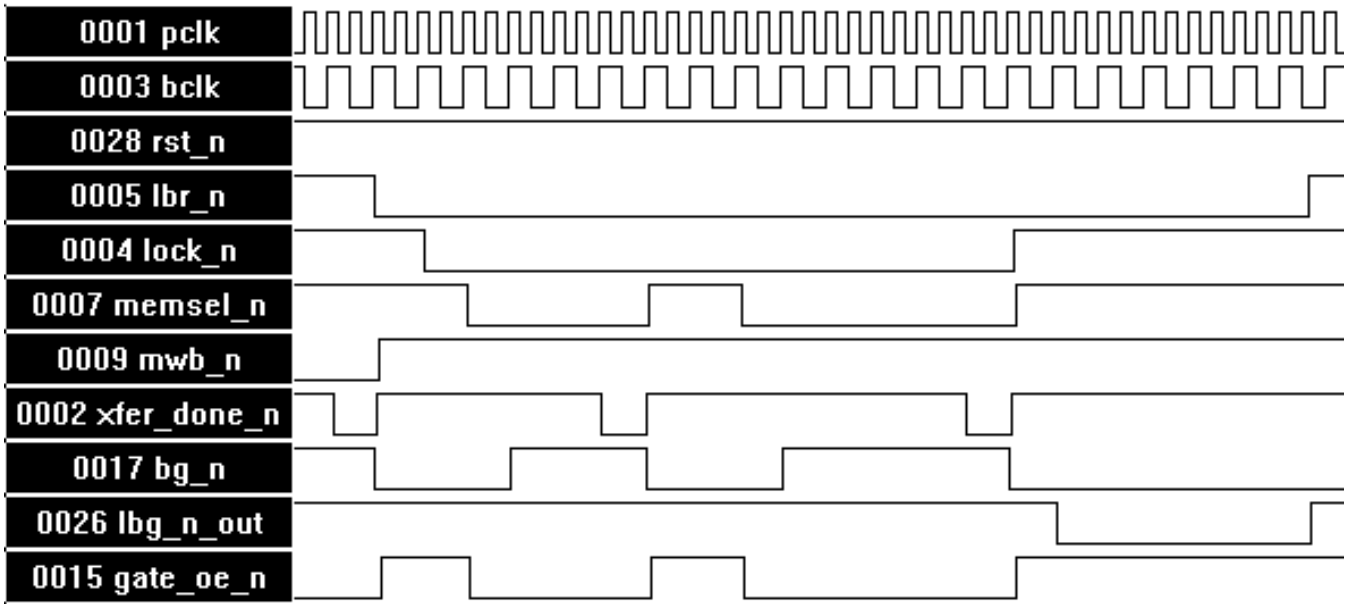

Figure 11. Arbitration Timing Diagram 2

dress bits are both LOW, this is decoded as a CY7C964 register access.

Register Access Cycle Initiation

If an address of $1xxxxxxx_{16}$ is detected when $\overline{TS}$ from the 68040 is active, a register access cycle is initiated. The address is qualified by the lowest address bits, the SIZ signals, and the transfer type from the 68040. For the CY7C964 register access, the PAS* and DS* signals are both kept inactive and only the STROBE* signal is allowed to be driven active. Data on the D31−D08 signals will be written into the CY7C964's while the data on D07−D00 is ignored. For the VIC64 register access, the PAS*, DS*, and CS* are all driven active. Data to be written to the VIC64 would be presented on D07−D00. Data read from the VIC64 would also appear on D07−D00.

To assure the proper timing on the D07−D00 signals with respect to the DS* signal, DS* is driven active on the cycle following PAS* driven active. This guarantees that, during a write cycle, data is present at the VIC64 prior to DS* becoming active.

Register Access Cycle Termination

The end of a register access cycle is indicated differently depending on whether the VIC64 registers were accessed or the CY7C964 registers were accessed. Also, the read or write status of the transfer has a bearing on how the cycle is terminated. For the VIC64 register transfers, the Cycle Termination PLD waits for either DSACK0* or DSACK1* to occur to indicate the end of the transfer. The DSACK1* and DSACK0* signals are registered as they enter the Cycle Termination PLD in order to synchronize them to the BCLK before they are used in output equations.

For the CY7C964 register transfers, the PLD counts three BCLK cycles before ending the cycle. This is because there are no external signals that indicate that the CY7C964s have received data.

In order to allow proper data hold times to the CY7C964 or VIC64, the termination of a write cycle is handled differently from the termination of the read cycle. In a read cycle, all active signals (PAS*, DS*, and CS*) are driven inactive at the same time in response to the XFER_DONE_N signal from the Cycle Termination PLD. However, for writes, an additional signal, XFER_DONE_W_N, is activated a full cycle before XFER_DONE_N. The STROBE* signal for CY7C964 register writes and the DS* for VIC64 register writes are driven inactive in response to this signal. On the subsequent BCLK cycle, the 68040 is given a $\overline{TA}$ signal and the PAS* and CS* signals are driven inactive. This insures that the rising edge of STROBE* or DS* is a cycle before the 68040 can remove data from the bus, thus guaranteeing the necessary data hold time into the CY7C964's and VIC64.

Performance of Register Access Cycles

An example of VIC64 register access is shown in *Figure 12*. An example of CY7C964 register access is shown in *Figure 13*. From these diagrams, the following performance figures are guaranteed for the different types of register access cycles.

VIC64 register write: 11 BCLK cycles assuming the slowest DSACK1/0* response time from the VIC64.

VIC64 register read: 10 BCLK cycles assuming the slowest DSACK1/0* response time from the VIC64.

CY7C964 register write: 10 BCLK cycles.

Master Read Cycles

Master Read cycles are also controlled by three PLDs, two Address and Cycle Decode PLDs and a Cycle Termination PLD. These cycles are very similar to a VIC64 Register read; however, instead of the VIC64 providing data and terminating the cycle, an addressed slave board would provide data and indicate that the data is available with the VMEbus signal, DTACK*. The VIC64 would issue DSACK1* and/or DSACK0* in response to the DTACK* signal.

Master Read Cycle Initiation

If an address of $2xxxxxxx_{16}$, $3xxxxxxx_{16}$, or $Fxxxxxxx_{16}$ is detected, along with R/$\overline{W}$ being in the HIGH state, when $\overline{TS}$ from the 68040 is active, a VMEbus master read access cycle is initiated. The address is qualified by the transfer type from the 68040.

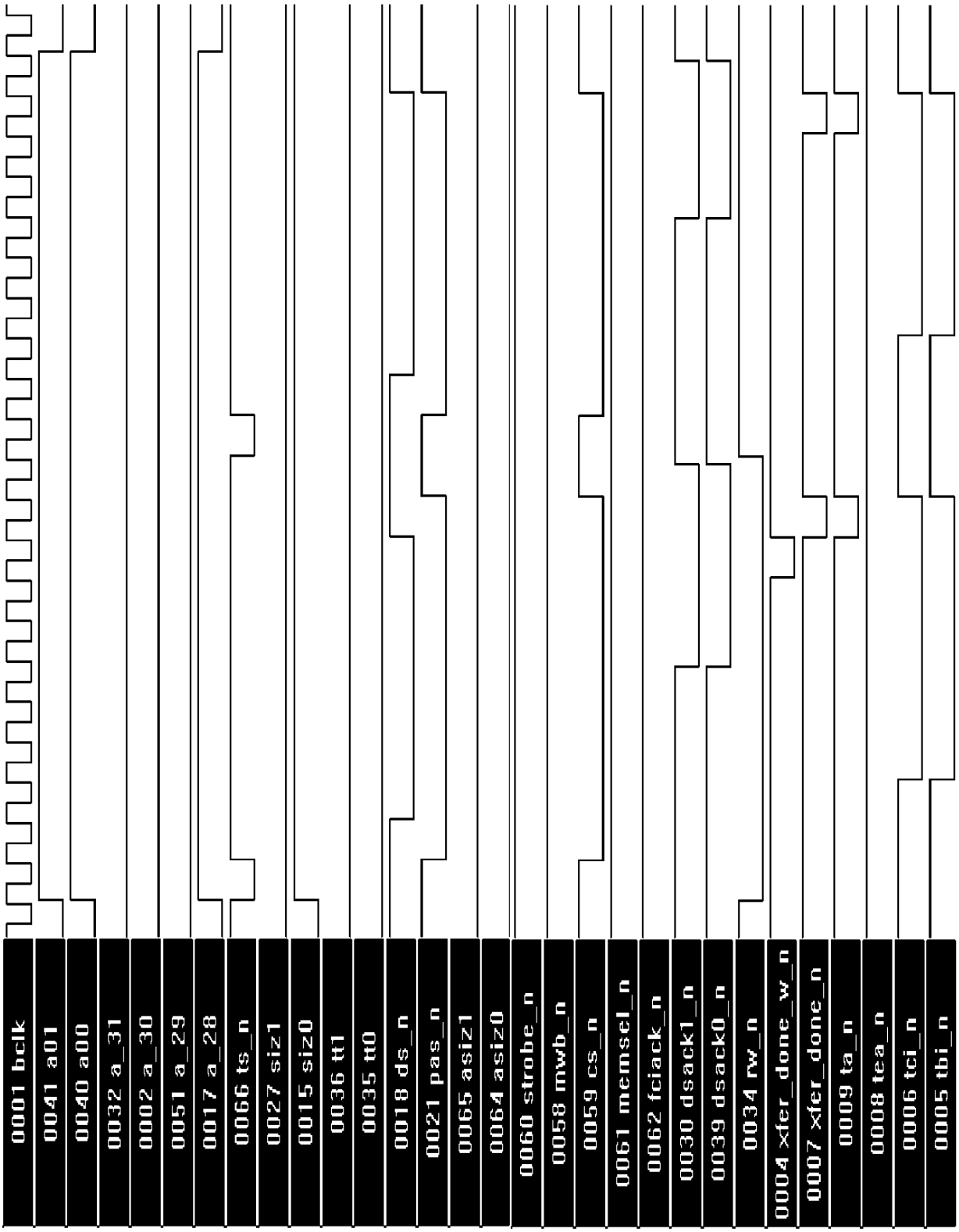

Figure 12. VIC64 Register Access

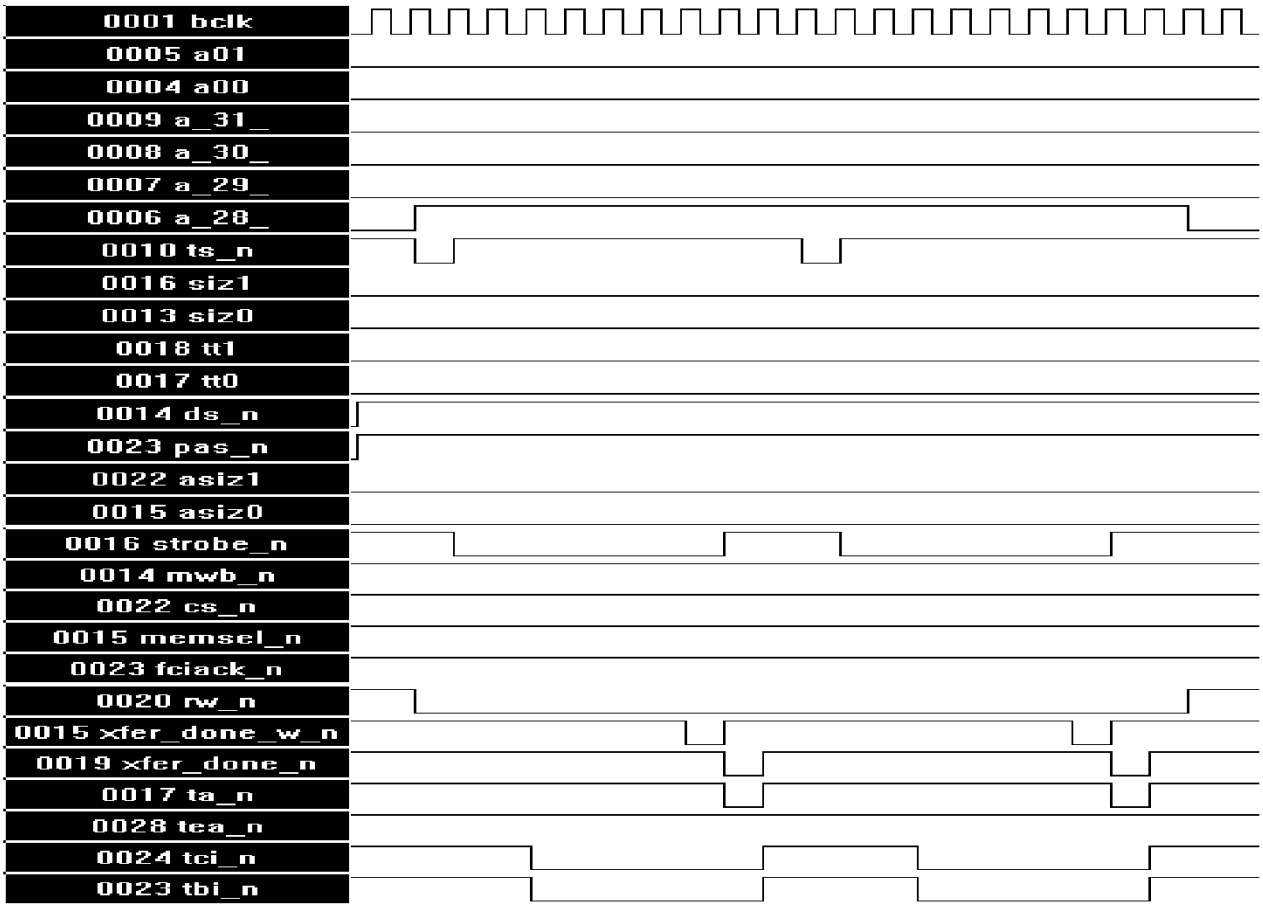

Figure 13. CY7C964 Register Access

MWB*, PAS* and DS* are driven active when the cycle is decoded and the ASIZ1 and ASIZ0 are driven based on the address from the 68040. *Table 1* shows how the address from the 68040 is decoded.

Table 1. 68040 Address Decode

68040 Address	Address Size	ASIZ1/ASIZ0
$2xxxxxxx_{16}$	A16	1/0
$3xxxxxxx_{16}$	A24	1/1
$Fxxxxxxx_{16}$	A32	0/1

The 68040 also indicates the memory space that is to be accessed with its TM2−TM0 lines. These signals are driven the the VIC64's FC2 and FC1 signals for generating AM codes on the VMEbus. The VIC64 will control the buffer control signals to the CY7C964's based on the size of the data that is to be transferred (indicated by the SIZ1 and SIZ0 signals from the 68040) and will initiate a VMEbus read with the appropriate VMEbus signals.

Master Read Cycle Termination

Once there has been a VMEbus read cycle initiated by the VIC64, there are three typical ways the cycle can be terminated. The cycle can be ended normally, be deadlocked and retried, or be terminated abnormally via a bus error.

Master Read Cycle Normal Termination

A normal master read will be terminated when the Cycle Termination PLD receives one or both of the DSACK0* or DSACK1* signals from the VIC64. These signals are driven by the VIC64 in response to a DTACK* signal from the addressed slave on the VMEbus backplane. The performance of a normal-

ly terminated cycle can vary due to the response time of the slave board being addressed and whether or not the VIC64 was granted access to the VMEbus quickly. *Figure 14* illustrates two back to back read cycles on the VMEbus that are terminated normally.

Master Read Cycle Deadlock/Retry Termination

A master read that ends in deadlock occurs when a slave cycle and a master cycle are asserted to the VIC64 at the same time. The VIC64 indicates that a deadlock has occurred by asserting the DEDLK* signal when the local side attempts access during a slave transaction. In response to the DEDLK* signal from the VIC64, the Cycle Termination PLD drives both the $\overline{\text{TEA}}$ and $\overline{\text{TA}}$ signals active to the 68040. This will cause the 68040 to end its cycle, wait one BCLK cycle (due to the Bus Arbitration PLD), and then attempt the cycle again.

The 68040 will continue to retry the cycle until the cycle is ended either with $\overline{\text{TEA}}$ or $\overline{\text{TA}}$ only. On each attempt the 68040 makes to the VIC64, the Address and Cycle Decode PLDs look at the state of the DEDLK_S signal from the Cycle Termination PLD. DEDLK_S is a double-registered version of the DEDLK* signal from the VIC64. If the DEDLK_S is active on an otherwise valid attempt to access the VIC64's private bus, the DEAD_N signal will activate instead of the normal signal (MWB*, CS*, etc.). The assertion of *DEAD*_N will not affect the Bus Arbitration state machine but will allow the Cycle Termination PLD to again cause a retry to the 68040 with $\overline{\text{TEA}}$ and $\overline{\text{TA}}$ together.

This method is used because there is a possibility that DEDLK* could go inactive *during* a 68040

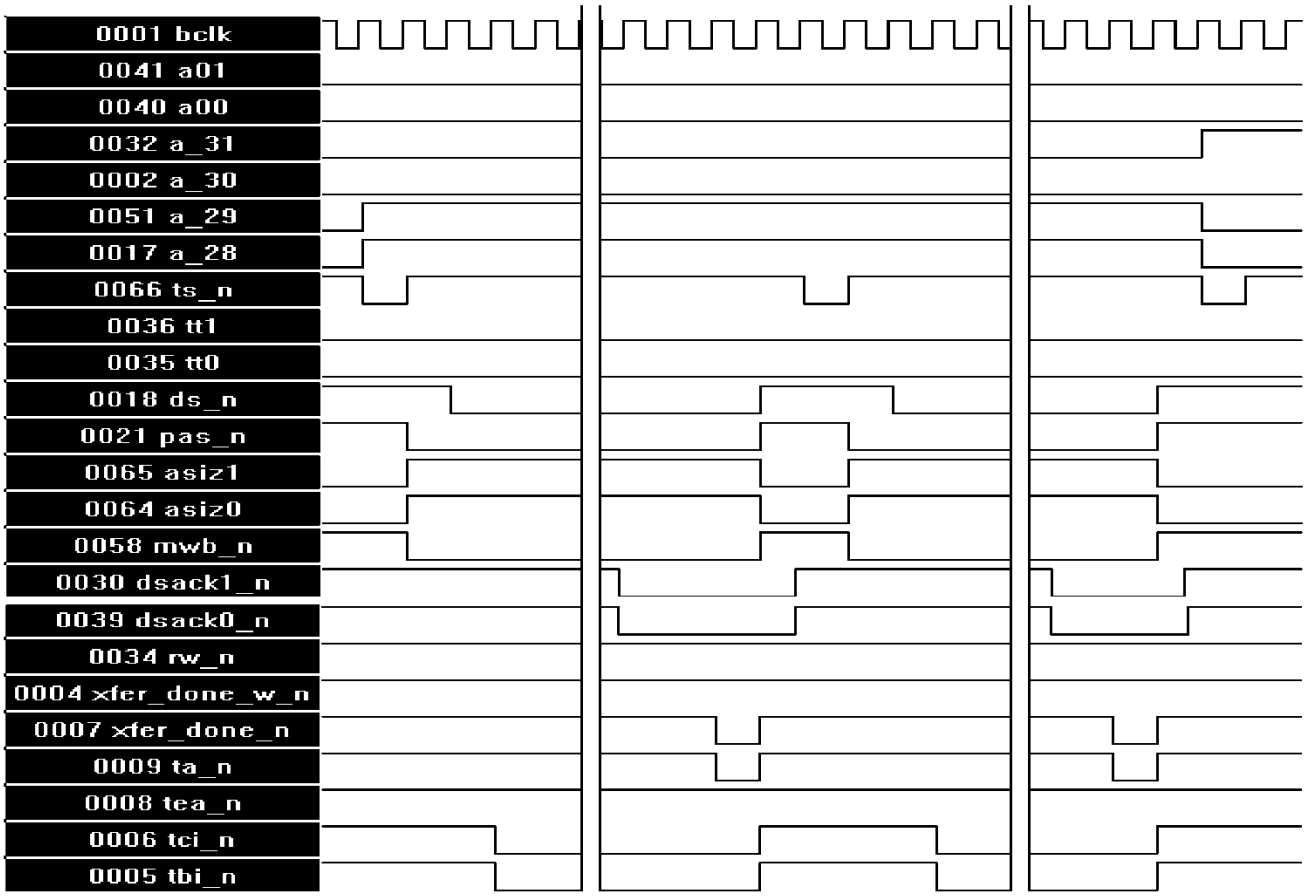

Figure 14. Master Reads

cycle. If this occurs, the Cycle Termination PLD could see DEDLK* active and terminate the cycle with $\overline{TEA}$ and $\overline{TA}$ active, thus indicating a retry to the 68040. However, the Bus Arbitration PLD may see that the Address and Cycle Decode PLD is signaling a valid cycle with LBR* inactive and an active select signal. This would cause the Cycle Termination PLD and the Bus Arbitration PLD to lose synchronization with each other. By preventing the Bus Arbitration PLD from even seeing a cycle that potentially could have a deadlock (with the *dead_n* signal), it will not arbitrate that cycle and the Cycle Termination PLD will cause a retry.

When the DEDLK* has been released by the VIC64, the 68040 will be able to finally complete the cycle that it has been retrying. The cycle will be a normal master read. *Figure 15* shows 4 cycles attempted by the 68040. The first cycle ends in retry when a DEDLK* is recognized in the middle of the

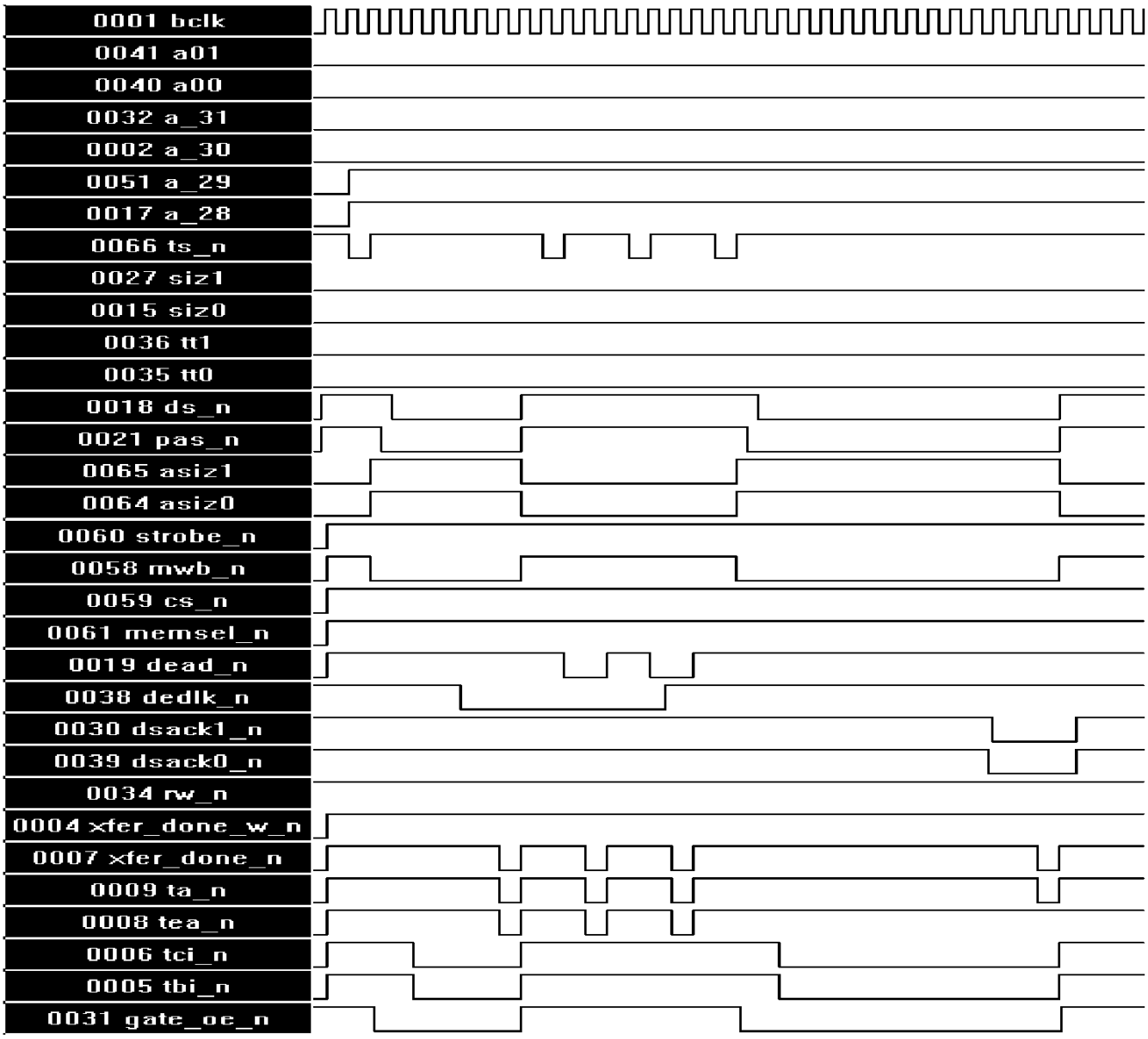

Figure 15. Master Reads with Deadlock

cycle. The next two cycles begin as deadlocked cycles and thus are immediately forced to be retried by the Cycle Termination PLD. The last cycle occurs after the deadlock and thus begins and ends as a normal master read.

Master Read Cycle Bus Error Termination

A master read will be terminated as a bus error to the 68040 when the Cycle Termination PLD receives the LBERR* signal from the VIC64. This signal is driven by the VIC64 in response to a BERR* signal from the addressed slave on the VMEbus backplane or a VMEbus timeout (based on the configuration of the TTR, register $A3 in the VIC64). A cycle terminated with a Bus Error will look similar to the timing shown in *Figure 14*. The differences will be twofold. First, the LBERR* signal will be driven by the VIC64 instead of DSACK1* and DSACK0*. Second, the $\overline{\text{TEA}}$ signal will be driven to the 68040 instead of the $\overline{\text{TA}}$ signal. Other than these differences, the cycles are equivalent.

Master Write, Writepost, and BLT Initiation Cycles

Like the Register Access cycles and Master Read cycles, Master Write cycles are also controlled by three PLDs, two Address and Cycle Decode PLDs and a Cycle Termination PLD. These cycles are very similar to a VIC64 Register write; however, instead of the VIC64 providing data and terminating the cycle, an addressed slave board would provide data and indicate that the data is available with the VMEbus signal, DTACK*. The VIC64 would issue DSACK1* and/or DSACK0* in response to the DTACK* signal.

Commonality Between the Various Write Cycles

Each of the cycles, Master Write, Writepost, and BLT initiation are subtly different. However, each shares the common trait that they are all write cycles from the 68040's perspective and all produce an MWB* signal to the VIC64. In each case however, the data is dissimilar. The Master Write and Writepost actually provide data that is transferred to an addressed slave, while the data from the BLT initiation cycle is the local address where the block transfer begins.

Write Cycle Initiation

If an address of $2xxxxxxx_{16}$, $3xxxxxxx_{16}$, or $Fxxxxxxx_{16}$ is detected, along with R/$\overline{\text{W}}$ being in the LOW state, when $\overline{\text{TS}}$ from the 68040 is active, a VMEbus master write-access cycle is initiated (or a block transfer initiation cycle if bit 6 of the BTCR is set). The address is qualified by the transfer type from the 68040. MWB*, PAS* and DS* are driven active when the cycle is decoded and the ASIZ1 and ASIZ0 are driven based on the address from the 68040. *Table 2* shows how the address from the 68040 is decoded.

Table 2. 68040 Address Decode

68040 Address	Address Size	ASIZ1/ASIZ0
$2xxxxxxx_{16}$	A16	1/0
$3xxxxxxx_{16}$	A24	1/1
$Fxxxxxxx_{16}$	A32	0/1

The 68040 also indicates the memory space that is to be accessed with its TM2−TM0 lines. These signals are driven as the VIC64's FC2 and FC1 signals for generating AM codes on the VMEbus. The VIC64 will control the buffer control signals to the CY7C964's based on the size of the data that is to be transferred (indicated by the SIZ1 and SIZ0 signals from the 68040) and will initiate a VMEbus write with the appropriate VMEbus signals.

To assure the proper timing on the D07−D00 signals with respect to the DS* signal, DS* is driven active on the cycle following PAS* driven active. This guarantees that during a write cycle data is present at the VIC64 prior to DS* active.

Write Cycle Termination

As with a VMEbus read cycle, once there has been a VMEbus write cycle initiated by the VIC64, there are three typical ways the cycle can be terminated. The cycle can be ended normally, be deadlocked and retried, or be terminated abnormally via a bus error.

Write Cycle Normal Termination

A normal master write will be terminated when the Cycle Termination PLD receives one or both of the

DSACK0* or DSACK1* signals from the VIC64. These signals are driven by the VIC64 in response to a DTACK* signal from the addressed slave on the VMEbus backplane. The performance of a normally terminated cycle can vary due to the response time of the slave board being addressed and whether or not the VIC64 was granted access to the VMEbus quickly.

In order to allow proper data hold times to the VIC64 for BLT initiation cycles and Master Write-posts, the termination of a write cycle is handled differently from the termination of the read cycle. In a read cycle, all active signals (PAS*, DS*, and MWB*) are brought inactive at the same time in response to the XFER_DONE_N signal from the Cycle Termination PLD. However, for writes, an additional signal, XFER_DONE_W_N, is activated a full cycle before XFER_DONE_N. The DS* signal is brought inactive in response to this signal. On the subsequent BCLK cycle, the 68040 is given a $\overline{TA}$ signal and the PAS* and MWB* signals are driven inactive. This insures that the rising edge of DS* is a cycle before the 68040 removes data from the bus, thus guaranteeing the necessary data hold time into the VIC64.

This timing is not an issue for Master writes since the VMEbus specification states that a slave will only issue a DTACK* after it has accepted the data written to it. Thus, hold time on the data is inherent in the delay from DTACK* on the VMEbus to DSACKx* on the local bus to $\overline{TA}$ from the cycle termination PLD. *Figure 16* illustrates two back-to-back write cycles on the VMEbus that are terminated normally.

Write Cycle Deadlock/Retry Termination

A write that ends in deadlock occurs when a slave cycle and a master cycle are asserted to the VIC64 at the same time. The VIC64 indicates that a deadlock has occurred by asserting the DEDLK* signal when the local side attempts access during a slave transaction. In response to the DEDLK* signal from the VIC64, the Cycle Termination PLD drives both the $\overline{TEA}$ and $\overline{TA}$ signals active to the 68040. This will cause the 68040 to end its cycle, wait one BCLK cycle (due to the Bus Arbitration PLD), and then attempt the cycle again.

As with deadlock on a read cycle, there is a timing relationship between the bus arbitration PLD and the cycle termination PLD that must be maintained. This timing is discussed in the Master Read Cycle Deadlock/Retry Termination section above. Timing for write cycles that deadlock is identical to the timing shown in *Figure 16*. The only difference is the relationship of DS* to both MWB* and PAS* as described above.

Write Cycle Bus Error Termination

A master write will be terminated as a bus error to the 68040 when the Cycle Termination PLD receives the LBERR* signal from the VIC64. This signal is driven by the VIC64 in response to a BERR* signal from the addressed slave on the VMEbus backplane or a VMEbus timeout (based on the configuration of the TTR, register $A3 in the VIC64). A cycle terminated with a Bus Error will look similar to the timing shown in *Figure 16*. The differences will be twofold. First, the LBERR* signal will be driven by the VIC64 instead of DSACKx*. Second, the $\overline{TEA}$ signal will be driven to the 68040 instead of the $\overline{TA}$ signal. Other than these differences, the cycles are equivalent.

Interrupt Acknowledge Cycles

Interrupt Acknowledge cycles are controlled by the Interrupt PLD, Cycle Termination PLD, and Address and Cycle Decode PLDs. Typical functionality of the Interrupt PLD is shown in *Figure 17*. Operation of the Address Decode PLDs and the Cycle Termination PLD is comparable to a Master Read Cycle except that FCIACK* is active rather than MWB*.

Operation At Reset

Although not an interrupt-related function, the Interrupt PLD controls the configuration of the 68040's buffer mode via the $\overline{IPL2}$, $\overline{IPL1}$, and $\overline{IPL0}$ signals. During a board reset, the signals are all driven to a HIGH state to configure the 68040's signals to Large Buffer mode.

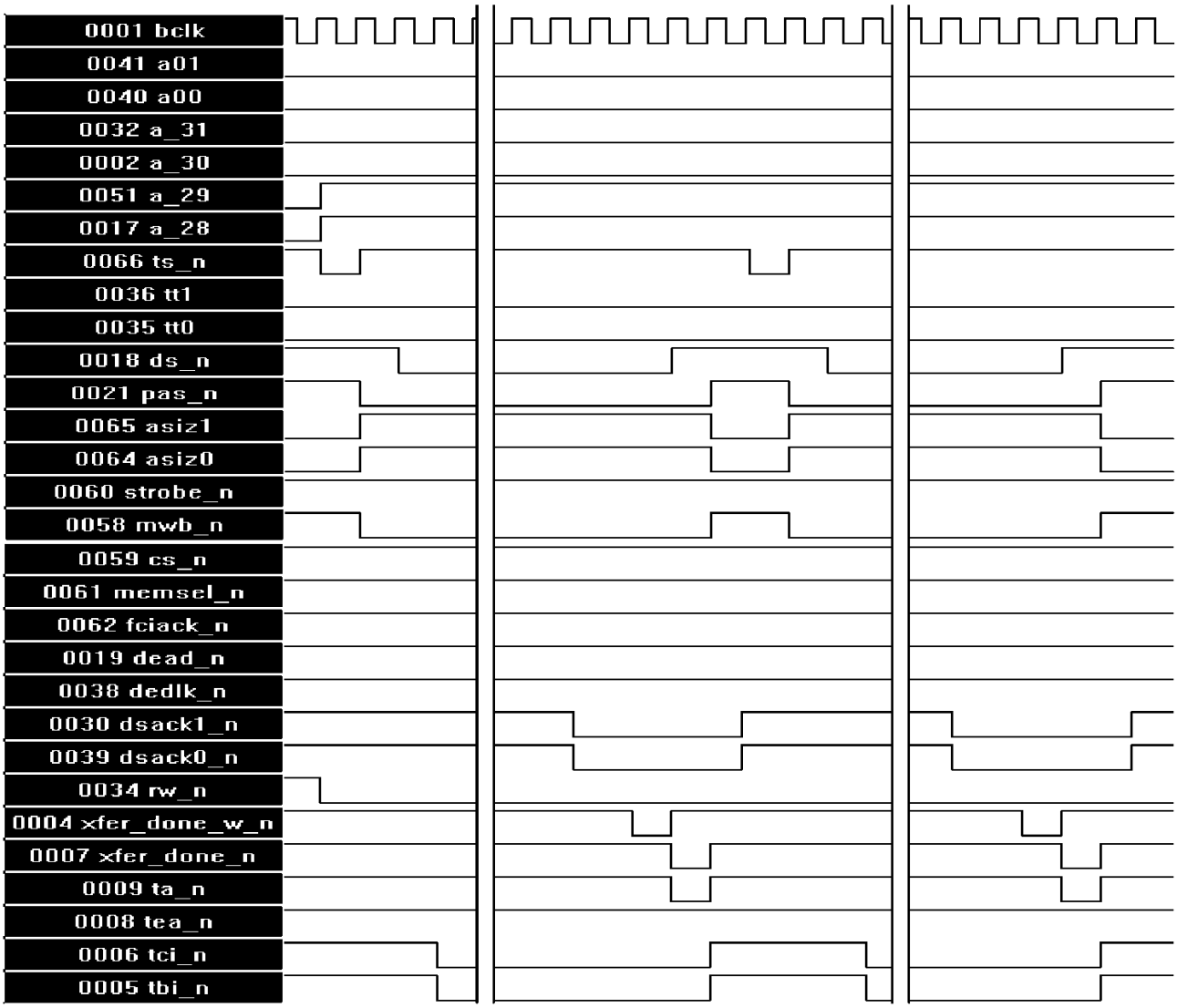

Figure 16. Master Writes

VMEbus vs. Local Interrupts

There are two possible sources for interrupts, the VMEbus and local interrupts. For VMEbus interrupts, the 68040 will only be involved if the VIC64 is configured as an interrupt handler. When the VMEbus interrupter generates an interrupt, the VIC64 will assert an interrupt to the 68040 via the IPL2*−IPL0* lines. The 68040 will respond with an interrupt acknowledge cycle. When the VIC64 sees the interrupt acknowledge cycle from the 68040, it obtains the VMEbus to request the Status/ID vector from the Interrupter. As the Status/ID vector is placed on data bus, it is passed through to the 68040 and the VIC64 terminates the cycle.

For local interrupts, a device on the board requiring service will assert an interrupt to the VIC64 via the LIRQ7*−LIRQ0* lines. The VIC64 will then assert an interrupt to the 68040 via the IPL2*−IPL0* lines. The 68040 will respond with an interrupt acknowledge cycle. There are two possible responses to the interrupt acknowledge cycle from the 68040. If the VIC64 is enabled to supply a vector for the current interrupt, it will do so and terminate the cycle with the DSACKx* signals. If the VIC64 is not

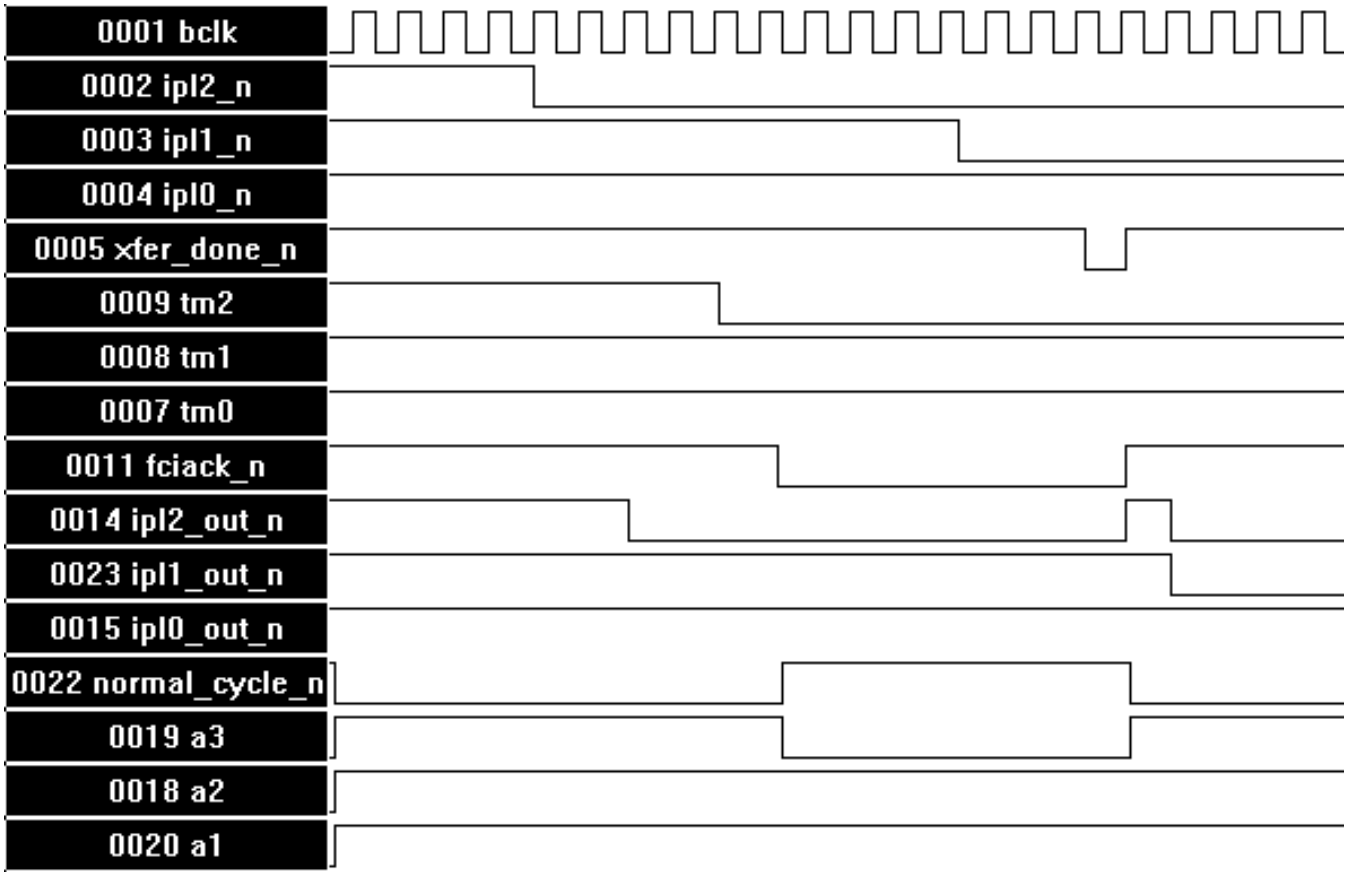

Figure 17. Interrupt Initiation and Acknowledge

enabled to provide a vector, it will assert the LIACKO* signal instead. The Cycle Termination PLD asserts AVEC to the 68040 in response to LIACKO* active and then terminates the cycle.

Another possible configuration for local interrupts is to have the LIACKO* from the VIC64 tell the interrupting device to supply a Status/ID vector to the 68040 and then terminate the cycle with a $\overline{TA}$ to the 68040.

Interrupt Initiation from the VIC64

As described above, the VIC64 issues an interrupt via its IPL2*–IPL0* signals. The IPL2*–IPL0* signals are normally in a HIGH state and are pulled LOW to request interrupt service from the 68040. When the IPL2*–IPL0* signals are pulled LOW, the Interrupt PLD synchronizes the signals before providing them to the 68040. The VIC64 may have up to 10 ns of skew in the IPL2*–IPL0* signals and that skew could be expanded to a full BCLK cycle through synchronization. However, the 68040 must see the interrupt level for two full BCLK cycles before it is considered valid so the skew is inconsequential.

Interrupt Cycle Initiation by the 68040

When the 68040 begins a bus cycle with $\overline{TS}$ active and the TT1 and TT0 signals are both HIGH, an interrupt acknowledge cycle is indicated. The SIZ1 and SIZ0 signals are also qualified to make sure they are indicating a byte-width operation. The Address Decode PLDs respond to the cycle by issuing FCIACK*, PAS*, and DS*. The VIC64 recognizes the beginning of an interrupt acknowledge cycle on the overlap of FCIACK*, PAS*, and DS* active.

Interrupt Cycle Decode

When FCIACK* goes active, the Interrupt PLD captures the current state of the $\overline{IPL2}-\overline{IPL0}$ signals and holds them throughout the cycle. The use of a Cypress PALC22V10D for this PLD guarantees the required hold times on the $\overline{IPL2}-\overline{IPL0}$ signals to the 68040 are met. When the cycle terminates, the $\overline{IPL2}-\overline{IPL0}$ signals are driven inactive for at least one BCLK cycle before a new interrupt level can be driven.

Another function that the Interrupt PLD performs is steering the TM2−TM0 signals from the 68040 onto the A3−A1 address lines on the VIC64. The TM2−TM0 signals from the 68040 contain the level of the interrupt being acknowledged and the VIC64 requires that information be passed on address lines A3−A1.

Interrupt Cycle Termination

The interrupt cycle is terminated in one of two ways from the VIC64. If the VIC64 is configured to supply a Status/ID vector, it will place that vector on D7−D0 and supply DSACKx* to the Cycle Termination PLD. If the VIC64 is not configured to supply a vector, it will issue a LIACKO* signal which will cause the Cycle Termination PLD to issue AVEC to the 68040 and then terminate the cycle with a $\overline{TA}$.

Summary

This application note has designed a possible VIC64 to Motorola 68040 interface. The issues and assumptions that must be addressed in the interface have been covered. The circuitry required for bus arbitration, resets, reads, writes, and interrupts has been designed. VHDL code for the PLDs used in the application note as well as timing diagrams and schematics have been provided in the following Appendices.

References

1. Cypress Semiconductor, *VIC068A/VAC068A User's Guide*, June, 1992.
2. Cypress Semiconductor, *VIC64/CY7C964 Design Notes*, October, 1993.
3. Motorola, Inc., *MC68040 Microprocessors User's Manual (M68040UM/AD)*, 1992.
4. Mazor, S., and P. Langstraat, *A Guide to VHDL*, Boston: Kluwer Academic, 1992.

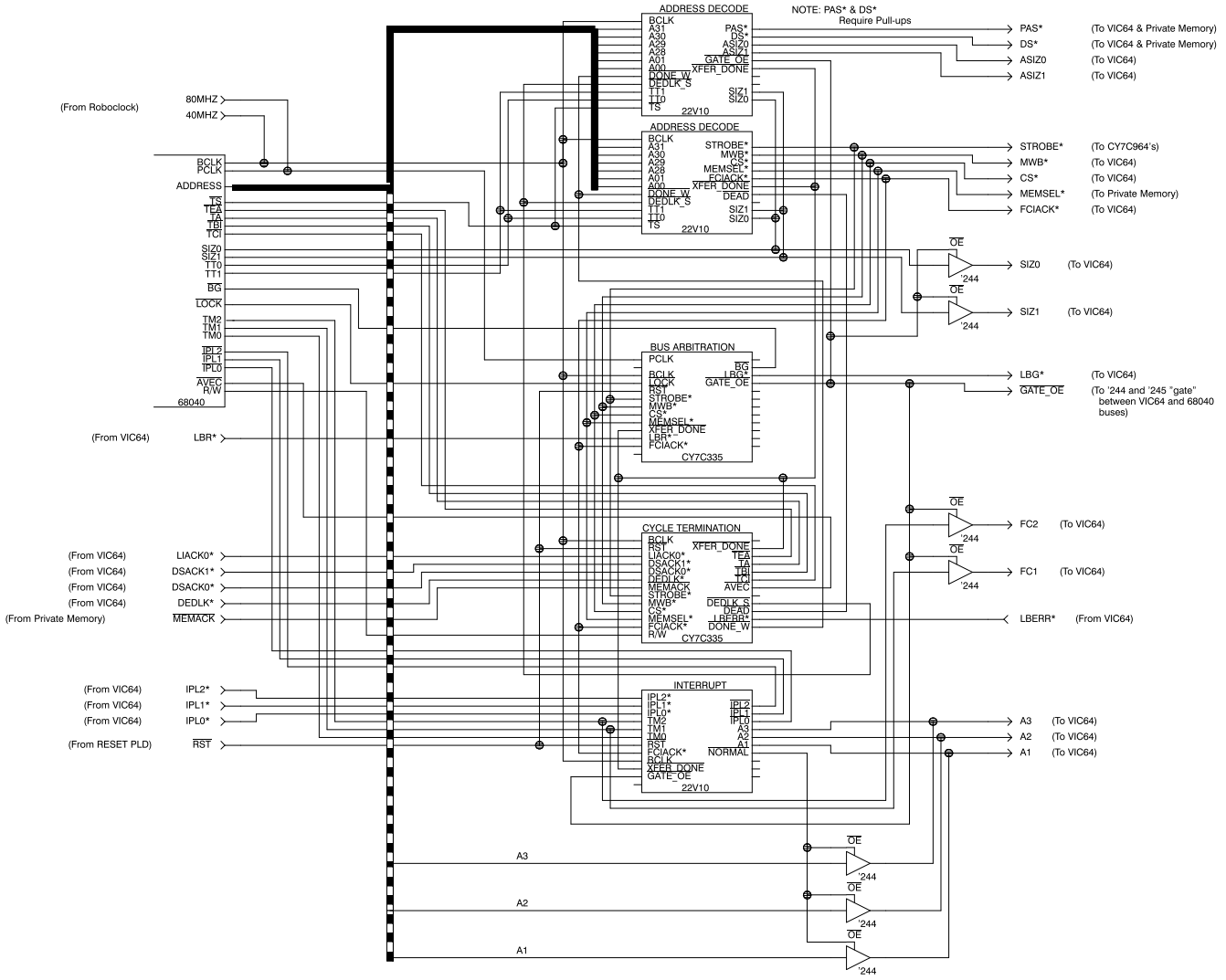

Figure 18. Schematic of Control Logic Circuitry

Appendix A. Reset Control PLD (CY7C335)

```
-- RESET PLD design
--
-- The following table is a cross reference between the PLD port names and
-- those on the schematic in the application note text.
--
--    clock            =  BCLK
--    vic_rst_n        =  RESET*
--    rst_040_n        =  RSTO "bar"
--    pwrup_rst_n      =  PURST "bar"
--    irst_n           =  IRESET*
--    brd_rst_n_out    =  RST "bar"
--    ipl0_n           =  IPL0*

ENTITY rst_ctrl IS
        PORT (clock, vic_rst_n, rst_040_n, pwrup_rst_n : in bit;
              irst_n, brd_rst_n_out : out bit;
              ipl0_n : inout x01z);
attribute part_name of rst_ctrl:entity is "c335";
attribute pin_numbers of rst_ctrl:entity is "clock:1 vic_rst_n:4 "
                                    & "rst_040_n:5 pwrup_rst_n:6";
END rst_ctrl;

USE work.rtlpkg.all;

ARCHITECTURE operation OF rst_ctrl IS
        signal pwrup_rst_n_rising:bit;
        signal pwrup_rst_n_reg:bit;
        signal pwrup_rst_n_reg1:bit;
        signal brd_rst_n:bit;
        signal start:bit;
        signal expired: bit;
        signal timer_count:integer(0 to 4);
        signal ipl0_oe: bit;
        signal ipl0_sig: bit;
        type states is (idle, rst1, rst2, rst3, wait_for_no_rst);
        signal rst_state: states;

        BEGIN

-- This process captures the power-up or pushbutton reset in two registers
-- in order to synchronize the signal and create a pulse on the rising edge
-- of the reset

        sync_rst: PROCESS BEGIN
            WAIT UNTIL clock = '1';
            pwrup_rst_n_reg <= pwrup_rst_n;
            pwrup_rst_n_reg1 <= pwrup_rst_n_reg;
        END PROCESS;

-- This concurrent assignment creates a pulse when the rising edge of the
-- power-up or pushbutton reset occurs.  This is used to begin a global reset
-- to the VIC which must be reset AFTER the Vcc and Oscillator are stable.
```

Appendix A. Reset Control PLD (CY7C335) (continued)

```
        pwrup_rst_n_rising <= '1' WHEN ((pwrup_rst_n_reg = '1') AND
                                        (pwrup_rst_n_reg1 = '0'))
                              ELSE '0';

-- This concurrent assignment guarantees that the reset out of this part will
-- be low when either the power-up/pushbutton reset is low or the brd_rst_n
-- signal generated in the process below is low.  This assures that as this
-- PLD powers up, the reset out of it will be low even if the process below
-- has not begun stable operation.

        brd_rst_n_out <= '0' WHEN ((brd_rst_n = '0') OR
                                   (pwrup_rst_n = '0'))
                         ELSE '1';

        rst: PROCESS BEGIN
            WAIT UNTIL clock = '1';

-- Do the following while the board is powering up or the pushbutton
-- is depressed

            IF (pwrup_rst_n_reg = '0') THEN
                  rst_state <= idle;
                  brd_rst_n <= '0';
                  ipl0_sig <= '1';
                  irst_n <= '1';
                  start <= '0';
            END IF;
            CASE (rst_state) IS

-- When the state machine is in the idle state, we look for the VIC to
-- initiate a reset or the 68040 to initiate a reset.  We also look for the
-- end of the powerup reset.  Remember that even though this state machine
-- is waiting for the end of power-up/pushbutton reset, the board is still
-- held in reset by the brd_rst_n_out signal above.

                    WHEN idle =>
                            IF ((pwrup_rst_n_rising = '1') OR
                                 (vic_rst_n = '0') OR
                                 (rst_040_n = '0')) THEN
                                    rst_state <= rst1;
                                    start <= '1';
                                    irst_n <= '0';
                                    brd_rst_n <= '0';
                            END IF;

-- In the rst1 state, we wait for one of two events.  If the VIC responds to
-- the reset from this PLD before the timer expires, we pull ipl0_n low and
-- continue to the rst2 state.  This would be the normal procedure.  If for
-- some unknown reason the VIC doesn't respond, we would wait for the timer
-- to expire and then assert ipl0_n.
```

Appendix A. Reset Control PLD (CY7C335) (continued)

```
                    WHEN rst1 =>
                            start <= '0';
                            IF (vic_rst_n = '0') THEN
                                    rst_state <= rst2;
                                    ipl0_sig <= '0';
                                    start <= '1';
                            ELSIF (expired = '1') THEN
                                    rst_state <= rst2;
                                    ipl0_sig <= '0';
                                    start <= '1';
                            END IF;

-- Just wait around in this state until the timer expires.  Remove the ipl0_n
-- signal at the end of this state.

                    WHEN rst2 =>
                            start <= '0';
                            IF ((expired = '1') AND (start = '0')) THEN
                                    rst_state <= rst3;
                                    ipl0_sig <= '1';
                                    start <= '1';
                            END IF;

-- Just wait around in this state until the timer expires.  Remove resets at
-- the end of this state.

                    WHEN rst3 =>
                            start <= '0';
                            IF (expired = '1') THEN
                                    rst_state <= wait_for_no_rst;
                                    irst_n <= '1';
                                    brd_rst_n <= '1';
                            END IF;

-- We remain in this state until the VIC comes out of reset.

                    WHEN wait_for_no_rst =>
                            IF (vic_rst_n = '1') THEN
                                    rst_state <= idle;
                            END IF;

-- Make the state machine complete.

                    WHEN others => rst_state <= idle;
            END CASE;
        END PROCESS;

-- The following makes the ipl0_sig signal a three-state signal on the
-- pin of the device.  This is required since ipl0 is normally driven by
-- the VIC64 but needs to be driven by this PLD during global reset.

        ipl0_oe <= NOT ipl0_sig;
        ipl0: bufoe port map (ipl0_sig, ipl0_oe, ipl0_n, open);
```

Appendix A. Reset Control PLD (CY7C335) (continued)

```
-- The timer process runs a counter that times how long the state machine
-- above should remain in a state.

        timer: PROCESS BEGIN
                WAIT UNTIL clock = '1';
                IF (pwrup_rst_n_reg = '0') THEN
                        timer_count <= 0;
                        expired <= '0';
                ELSE
                        IF (timer_count /= 0) THEN
                             timer_count <= timer_count + 1;
                        ELSE
                             timer_count <= 0;
                        END IF;

                    IF start = '1' THEN
                             timer_count <= 1;
                        END IF;

                        IF timer_count = 4 THEN
                             expired <= '1';
                    ELSE
                         expired <= '0';
                        END IF;

                END IF;
        END PROCESS;
END operation;
```

Appendix B. Address and Cycle Decode PLDs (PALC22V10D)

```
-- ADDRESS DECODER design 1
--
-- The following table is a cross reference between the PLD port names and
-- the signals found on the physical IC's.
--
--    bclk             =  BCLK on 68040
--    a(31) to a(28)   =  Address bus on 68040
--    ts_n             =  TS "bar" on 68040
--    tt1              =  TT1 on 68040
--    tt0              =  TT0 on 68040
--    siz1             =  SIZ1 on 68040
--    siz0             =  SIZ0 on 68040
--    xfer_done_n      =  xfer_done_n from TERMINATION PLD
--    xfer_done_w_n    =  xfer_done_w_n from TERMINATION PLD
--    gate_oe_n        =  gate_oe_n from BUS ARBITRATION PLD
--    dedlk_s          =  dedlk_s from TERMINATION PLD
--    asiz1            =  ASIZ1 to VIC64
--    asiz0            =  ASIZ0 to VIC64
--    pas_n            =  PAS* to VIC64
--    ds_n             =  DS* to VIC64

ENTITY address_decoder IS
        PORT (a01, a00, bclk, ts_n, tt1, tt0, siz1, siz0 : in bit;
              xfer_done_n, xfer_done_w_n, gate_oe_n, dedlk_s : in bit;
              a : in bit_vector(31 downto 28);
              asiz1, asiz0 : out bit;
              pas_n, ds_n : inout x01z);
attribute part_name of address_decoder:entity is "c22v10";
END address_decoder;

USE work.rtlpkg.all;

ARCHITECTURE operation OF address_decoder IS
          SIGNAL tt, siz : bit_vector(1 downto 0);
          SIGNAL pas_sig, ds_sig : bit;
          SIGNAL open_gate : bit;
          CONSTANT byte : bit_vector(1 downto 0) := "01";
          CONSTANT word : bit_vector(1 downto 0) := "10";
          CONSTANT lword : bit_vector(1 downto 0) := "00";
          CONSTANT acknow : bit_vector(1 downto 0) := "11";
          CONSTANT normal : bit_vector(1 downto 0) := "00";
BEGIN

     tt <= tt1 & tt0;
     siz <= siz1 & siz0;

     PROCESS BEGIN
          WAIT UNTIL bclk = '1';

-- At the start of a 68040 cycle, determine which signals should be activated
-- However, if the dedlk_s signal from the CYCLE TERMINATION PLD is active,
-- a transfer should not be begun to the VIC64.
```

Appendix B. Address and Cycle Decode PLDs (PALC22V10D) (continued)

```
                IF (ts_n = '0') AND (dedlk_s = '1') THEN
                     -- 964 Registers
                     IF (a = "0001") AND (a01 & a00 = "00") AND (tt = normal) AND
                       (siz = lword) THEN
                          asiz1 <= '0';
                          asiz0 <= '0';
                          pas_sig <= '1';
                     -- VIC64 Registers
                     ELSIF (a = "0001") AND (a01 & a00 = "11") AND (tt = normal) AND
                          (siz = byte) THEN
                          asiz1 <= '0';
                          asiz0 <= '0';
                          pas_sig <= '0';
                     -- A16 Addressing
                     ELSIF (a = "0010") AND (tt = normal) THEN
                          asiz1 <= '1';
                          asiz0 <= '0';
                          pas_sig <= '0';
                     -- A24 Addressing
                     ELSIF (a = "0011") AND (tt = normal) THEN
                          asiz1 <= '1';
                          asiz0 <= '1';
                          pas_sig <= '0';
                     -- A32 Addressing
                     ELSIF (a = "1111") AND (tt = normal) THEN
                          asiz1 <= '0';
                          asiz0 <= '1';
                          pas_sig <= '0';
                     -- VIC64's Private Memory
                     ELSIF (a = "0100") AND (tt = normal) THEN
                          asiz1 <= '0';
                          asiz0 <= '0';
                          pas_sig <= '1';
                     -- Interrupt Acknowledge
                     ELSIF (tt = acknow) AND (siz = byte) THEN
                          asiz1 <= '0';
                          asiz0 <= '0';
                          pas_sig <= '0';
                     -- Not a cycle for us
                     ELSE
                          asiz1 <= '0';
                          asiz0 <= '0';
                          pas_sig <= '1';
                     END IF;
                END IF;

-- DS will follow whatever PAS does on the subsequent cycle

                IF (pas_sig = '0') THEN
                     ds_sig <= '0';
                END IF;
```

Appendix B. Address and Cycle Decode PLDs (PALC22V10D) (continued)

```
-- If the cycle was a write cycle, the ds_sig must be pulled high to latch
-- data into the VIC64.  This is to assure that the local data hold time of
-- 0ns to the VIC64 is not violated.  If this were a cycle sending data
-- across the VMEbus, pulling ds_sig high before pas_n will not cause
-- problems because the slave board that is being written to would have
-- captured data when it asserted DTACK* to the VIC64.

               IF (xfer_done_w_n = '0') THEN
                    ds_sig <= '1';
               END IF;

-- When the cycle has been completed, the signals are all returned to their
-- inactive states.

               IF (xfer_done_n = '0') THEN
                    asiz1 <= '0';
                    asiz0 <= '0';
                    pas_sig <= '1';
                    ds_sig <='1';
               END IF;
          END PROCESS;

-- The pas_n and ds_n are driven by the VIC64 when it has access to its
-- private bus.  By looking at the state of the gate_oe_n signal, the owner
-- of the bus can be determined.  If the gate_oe_n signal is asserted (low),
-- the '040 has control of the bus and the pas_n and ds_n signals must be
-- active.

     open_gate <= NOT gate_oe_n;

     pas: bufoe PORT MAP (pas_sig, open_gate, pas_n, open);
     ds:  bufoe PORT MAP (ds_sig,  open_gate, ds_n,  open);

END operation;
```

Appendix B. Address and Cycle Decode PLDs (PALC22V10D) (continued)

```
-- ADDRESS DECODER design 2
--
-- The following table is a cross reference between the PLD port names and
-- the signals found on the physical IC's.
--
--    bclk             =  BCLK on 68040
--    a(31) to a(28)   =  Address bus on 68040
--    ts_n             =  TS "bar" on 68040
--    tt1              =  TT1 on 68040
--    tt0              =  TT0 on 68040
--    siz1             =  SIZ1 on 68040
--    siz0             =  SIZ0 on 68040
--    xfer_done_n      =  xfer_done_n from TERMINATION PLD
--    xfer_done_w_n    =  xfer_done_w_n from TERMINATION PLD
--    dedlk_s          =  dedlk_s from TERMINATION PLD

--    dead_n           =  dead_n to TERMINATION PLD
--    memsel_n         =  chip select for VIC64's private memory
--    strobe_n         =  STROBE* on CY7C964's
--    mwb_n            =  MWB* to VIC64
--    cs_n             =  CS* to VIC64
--    fciack_n         =  FCIACK* to VIC64

ENTITY address_decoder IS
        PORT (a01, a00, bclk, ts_n, tt1, tt0, siz1, siz0, xfer_done_n : in bit;
              xfer_done_w_n, dedlk_s : in bit;
              a : in bit_vector(31 downto 28);
              memsel_n, strobe_n, mwb_n, cs_n, fciack_n, dead_n : out bit);
attribute part_name of address_decoder:entity is "c22v10";
END address_decoder;

USE work.rtlpkg.all;

ARCHITECTURE operation OF address_decoder IS
          SIGNAL tt, siz : bit_vector(1 downto 0);
          CONSTANT byte : bit_vector(1 downto 0) := "01";
          CONSTANT word : bit_vector(1 downto 0) := "10";
          CONSTANT lword : bit_vector(1 downto 0) := "00";
          CONSTANT acknow : bit_vector(1 downto 0) := "11";
          CONSTANT normal : bit_vector(1 downto 0) := "00";

BEGIN

     tt <= tt1 & tt0;
     siz <= siz1 & siz0;

     PROCESS BEGIN
          WAIT UNTIL bclk = '1';

-- At the start of a 68040 cycle, determine which signals should be activated
-- This will be run only if we are not seeing a deadlock situation via the
-- dedlk_s signal.
```

Appendix B. Address and Cycle Decode PLDs (PALC22V10D) (continued)

```
IF (ts_n = '0') AND (dedlk_s = '1') THEN
     -- 964 Registers
     IF (a = "0001") AND (a01 & a00 = "00") AND (tt = normal) AND
       (siz = lword) THEN
          strobe_n <= '0';
          mwb_n    <= '1';
          cs_n     <= '1';
          memsel_n <= '1';
          fciack_n <= '1';
     -- VIC64 Registers
     ELSIF (a = "0001") AND (a01 & a00 = "11") AND (tt = normal) AND
          (siz = byte) THEN
          strobe_n <= '1';
          mwb_n    <= '1';
          cs_n     <= '0';
          memsel_n <= '1';
          fciack_n <= '1';
     -- A16 Addressing
     ELSIF (a = "0010") AND (tt = normal) THEN
          strobe_n <= '1';
          mwb_n    <= '0';
          cs_n     <= '1';
          memsel_n <= '1';
          fciack_n <= '1';
     -- A24 Addressing
     ELSIF (a = "0011") AND (tt = normal) THEN
          strobe_n <= '1';
          mwb_n    <= '0';
          cs_n     <= '1';
          memsel_n <= '1';
          fciack_n <= '1';
     -- A32 Addressing
     ELSIF (a = "1111") AND (tt = normal) THEN
          strobe_n <= '1';
          mwb_n    <= '0';
          cs_n     <= '1';
          memsel_n <= '1';
          fciack_n <= '1';
     -- VIC64's Private Memory
     ELSIF (a = "0100") AND (tt = normal) THEN
          strobe_n <= '1';
          mwb_n    <= '1';
          cs_n     <= '1';
          memsel_n <= '0';
          fciack_n <= '1';
     -- Interrupt Acknowledge
     ELSIF (tt = acknow) AND (siz = byte) THEN
          strobe_n <= '1';
          mwb_n    <= '1';
          cs_n     <= '1';
          memsel_n <= '1';
          fciack_n <= '0';
```

Appendix B. Address and Cycle Decode PLDs (PALC22V10D) (continued)

```
          -- Not a cycle for us
          ELSE
               strobe_n <= '1';
               mwb_n    <= '1';
               cs_n     <= '1';
               memsel_n <= '1';
               fciack_n <= '1';
          END IF;
     END IF;


-- This is the section of code that will be run if there is a deadlock.
-- If the decoded address/tt/siz information would have normally decoded
-- to a valid cycle, we send out the dead_n signal instead.  This lets the
-- TERMINATION PLD know that the 68040 is issuing a valid request to the
-- VIC64 but that the VIC64 can't be bothered cause it is currently finishing
-- a slave operation.

     IF (ts_n = '0') AND (dedlk_s = '0') THEN
          strobe_n <= '1';
          mwb_n    <= '1';
          cs_n     <= '1';
          memsel_n <= '1';
          fciack_n <= '1';

          -- 964 Registers
          IF (a = "0001") AND (a01 & a00 = "00") AND (tt = normal) AND
            (siz = lword) THEN
               dead_n <= '0';
          -- VIC64 Registers
          ELSIF (a = "0001") AND (a01 & a00 = "11") AND (tt = normal) AND
               (siz = byte) THEN
               dead_n <= '0';
          -- A16 Addressing
          ELSIF (a = "0010") AND (tt = normal) THEN
               dead_n <= '0';
          -- A24 Addressing
          ELSIF (a = "0011") AND (tt = normal) THEN
               dead_n <= '0';
          -- A32 Addressing
          ELSIF (a = "1111") AND (tt = normal) THEN
               dead_n <= '0';
          -- VIC64's Private Memory
          ELSIF (a = "0100") AND (tt = normal) THEN
               dead_n <= '0';
          -- Interrupt Acknowledge
          ELSIF (tt = acknow) AND (siz = byte) THEN
               dead_n <= '0';
          -- Not a cycle for us
          ELSE
               dead_n <= '1';
          END IF;
     END IF;
```

Appendix B. Address and Cycle Decode PLDs (PALC22V10D) (continued)

```
-- If the cycle was a write cycle, the strobe_n must be pulled high to latch
-- data into the '964's.  This is to assure that the local data hold time of
-- 5ns to the 964's is not violated.

               IF (xfer_done_w_n = '0') THEN
                    strobe_n <= '1';
               END IF;

-- When the cycle has been completed, the signals are all returned to their
-- inactive states.

               IF (xfer_done_n = '0') THEN
                    strobe_n <= '1';
                    mwb_n    <= '1';
                    cs_n     <= '1';
                    memsel_n <= '1';
                    fciack_n <= '1';
                    dead_n   <= '1';
               END IF;
          END PROCESS;

END operation;
```

Appendix C. Cycle Termination PLD (CY7C335)

```
-- CYCLE TERMINATION PLD
--
-- The following table is a cross reference between the PLD port names and
-- the signals found on the physical IC's.
--
--    bclk             =  BCLK on 68040
--    rw_n             =  R/W "bar" on 68040
--    rst_n            =  brd_rst_n_out from RESET PLD
--    liacko_n         =  LIACKO* from VIC64
--    dsack1_n         =  DSACK1* from VIC64
--    dsack0_n         =  DSACK0* from VIC64
--    lberr_n          =  LBERR* from VIC64
--    dedlk_n          =  DEDLK* from VIC64
--    memack_n         =  MEMACK* from private memory
--    memsel_n         =  MEMSEL* from ADDRESS DECODE PLD
--    strobe_n         =  STROBE* from ADDRESS DECODE PLD
--    mwb_n            =  MWB* from ADDRESS DECODE PLD
--    cs_n             =  CS* from ADDRESS DECODE PLD
--    fciack_n         =  FCIACK* from ADDRESS DECODE PLD
--    dead_n           =  dead_n from ADDRESS DECODE PLD

--    avec_n           =  AVEC "bar" to 68040
--    xfer_done_n      =  XFER_DONE "bar" to BUS ARBITRATION/ADDRESS DECODE PLD's
--    xfer_done_w_n    =  XFER_DONE_W "bar" to BUS ARBITRATION/ADDRESS DECODE PLD's
--    tea_n            =  TEA "bar" to 68040
--    ta_n             =  TA "bar" to 68040
--    tci_n            =  TCI "bar" to 68040
--    tbi_n            =  TBI "bar" to 68040
--    dedlk_s          =  Double registered (sync'ed) dedlk_n signal to
--                                  ADDRESS DECODE PLDS


ENTITY cycle_termination IS
        PORT (bclk, liacko_n, dsack1_n, dsack0_n : in boolean;
              lberr_n, dedlk_n, memack_n, rst_n, rw_n : in boolean;
              dead_n : in boolean;
              memsel_n, strobe_n, mwb_n, cs_n, fciack_n : in boolean;
              avec_n, tea_n, ta_n, tci_n, tbi_n : out bit;
              dedlk_s : out bit;
              xfer_done_w_n, xfer_done_n : buffer bit);
attribute part_name of cycle_termination:entity is "c335";
END cycle_termination;

USE work.rtlpkg.all;
USE work.table_bv.all;

ARCHITECTURE operation OF cycle_termination IS
     SIGNAL any_access     : boolean;
     SIGNAL any_access_reg : boolean;
     SIGNAL cycle_end      : boolean;
     SIGNAL start, expired : bit;
     SIGNAL timer_count    : integer(0 to 7);
```

Appendix C. Cycle Termination PLD (CY7C335) (continued)

```
     SIGNAL liacko_n_reg   : boolean;
     SIGNAL dsack0_n_reg   : boolean;
     SIGNAL dsack1_n_reg   : boolean;
     SIGNAL dedlk_n_reg    : boolean;
     SIGNAL lberr_n_reg    : boolean;

BEGIN

     any_access <= NOT memsel_n OR NOT strobe_n OR NOT mwb_n OR NOT cs_n OR
                   NOT fciack_n;

     cycle_end <= (NOT memsel_n AND NOT memack_n) OR
                  (NOT strobe_n AND (timer_count = 3)) OR
                  (NOT mwb_n    AND (NOT dsack0_n_reg OR NOT dsack1_n_reg)) OR
                  (NOT fciack_n AND (NOT dsack0_n_reg OR NOT dsack1_n_reg)) OR
                  (NOT fciack_n AND (NOT liacko_n_reg)) OR
                  (NOT cs_n     AND (NOT dsack0_n_reg OR NOT dsack1_n_reg));

     controller: PROCESS BEGIN
          WAIT UNTIL bclk;
               liacko_n_reg   <= liacko_n;
               dsack0_n_reg   <= dsack0_n;
               dsack1_n_reg   <= dsack1_n;
               dedlk_n_reg    <= dedlk_n;
               IF dedlk_n_reg THEN
                    dedlk_s <= '1';
               ELSE
                    dedlk_s <= '0';
               END IF;
               lberr_n_reg    <= lberr_n;
               start          <= '0';
               xfer_done_n    <= '1';

               IF xfer_done_n = '0' THEN
                    any_access_reg <= FALSE;
               ELSE
                    any_access_reg <= any_access;
               END IF;

-- Normal beginning of a cycle starts the cycle timer and asserts the tbi_n
-- and tci_n to inhibit bursts and caching.

               IF any_access_reg THEN
                    tbi_n <= '0';
                    tci_n <= '0';
                    start <= '1';
               END IF;

-- Normal end to a write cycle will assert the xfer_done_w_n followed by an
-- assertion of xfer_done_n and ta_n.  Normal end to a read cycle is
-- xfer_done_n and ta_n asserted.

               IF cycle_end AND NOT rw_n AND (xfer_done_n = '1') THEN
                    xfer_done_w_n <= '0';
                    start <= '0';
```

Appendix C. Cycle Termination PLD (CY7C335) (continued)

```
                END IF;

                IF (cycle_end AND rw_n) OR (xfer_done_w_n = '0') THEN
                     xfer_done_w_n <= '1';
                     xfer_done_n <= '0';
                     ta_n <= '0';
                     start <= '0';
                END IF;

--  Error endings.  If dedlk_n_reg is active and an access is being attempted,
--  retry the cycle with ta_n and tea_n asserted together.  This will occur
--  only during a cycle.  If dead_n is active, we have already had an
--  initial deadlocked cycle and we are now in a sequence of retries to the
--  68040.
--  If there is a lberr_n assertion, just end the cycle with tea_n to
--  indicate an erred cycle.
--  xfer_done_n is also asserted in either case to shut off the selects in
--  the ADDRESS DECODE PLD's.

                IF (any_access_reg AND (NOT dedlk_n_reg)) OR (NOT dead_n) THEN
                     ta_n <= '0';
                     tea_n <= '0';
                     xfer_done_n <= '0';
                     start <= '0';
                END IF;

                IF any_access_reg AND (NOT lberr_n) THEN
                     tea_n <= '0';
                     xfer_done_n <= '0';
                     start <= '0';
                END IF;

--  liacko_n being asserted means that the processor should autovector the
--  current interrupt.

                IF (NOT liacko_n) THEN
                     avec_n <= '0';
                END IF;

--  Conclusion of the cycle.  xfer_done_n and all other outputs from this
--  PLD are placed in their inactive state.

                IF (xfer_done_n = '0') THEN
                     xfer_done_n <= '1';
                     tbi_n <= '1';
                     tci_n <= '1';
                     ta_n <= '1';
                     tea_n <= '1';
                     avec_n <= '1';
                     start <= '0';
                END IF;

-- Reset condition takes priority over any of the above assignments.
```

Appendix C. Cycle Termination PLD (CY7C335) (continued)

```
               IF (NOT rst_n) THEN
                    xfer_done_n <= '1';
                    xfer_done_w_n <= '1';
                    tbi_n <= '1';
                    tci_n <= '1';
                    ta_n <= '1';
                    tea_n <= '1';
                    avec_n <= '1';
                    start <= '0';
               END IF;

     END PROCESS;

     timer: PROCESS BEGIN
          WAIT UNTIL bclk;
               IF (timer_count /= 0) THEN
                    timer_count <= timer_count + 1;
               ELSE
                    timer_count <= 0;
               END IF;
               -------------------------------------
               IF start = '1' AND (timer_count = 0) THEN
                    timer_count <= 1;
               END IF;
               -------------------------------------
               IF xfer_done_n = '0' OR start = '0' OR (NOT rst_n) THEN
                    timer_count <= 0;
               END IF;
     END PROCESS;
END operation;
```

Appendix D. Bus Arbitration PLD (CY7C335)

```
-- BUS ARBITER PLD design
--
-- The following table is a cross reference between the PLD port names and
-- the signals found on the physical IC's.
--
--    pclk             =  PCLK on 68040
--    bclk             =  BCLK on 68040
--    bg_n             =  BG "bar" on 68040
--    lock_n           =  LOCK "bar" on 68040 (requires external pullup)
--    cs_n             =  cs_n from ADDRESS DECODE PLD
--    strobe_n         =  strobe_n from ADDRESS DECODE PLD
--    mwb_n            =  mwb_n from ADDRESS DECODE PLD
--    memsel_n         =  memsel_n from ADDRESS DECODE PLD
--    fciack_n         =  fciack_n from ADDRESS DECODE PLD
--    xfer_done_n      =  xfer_done_n from TERMINATION PLD
--    rst_n            =  brd_rst_n_out from RESET PLD
--    lbr_n            =  LBR* on VIC64
--    lbg_n_out        =  LBG* on VIC64
--    gate_oe_n        =  OE on GATE between 040 bus and VIC64 bus

ENTITY arbiter IS
        PORT (pclk, bclk, lock_n, cs_n, strobe_n, mwb_n : in bit;
              memsel_n, xfer_done_n, rst_n, lbr_n, fciack_n : in bit;
              bg_n, lbg_n_out, gate_oe_n : out bit);
attribute part_name of arbiter:entity is "c335";
attribute pin_numbers of arbiter:entity is "pclk:1 bclk:3";
END arbiter;

USE work.rtlpkg.all;

ARCHITECTURE operation OF arbiter IS
        signal lbr_n_reg1, lbr_n_reg2:bit;
        signal lbg_n : bit;
        signal selects:bit_vector(4 downto 0);
        type states is (reset, only040, slow_down040, both);
        signal arb_state: states;
        constant no_selects:bit_vector(4 downto 0) := "11111";
BEGIN

-- The local bus grant to the VIC64 must be removed within 1 VIC64 clock
-- cycle or the VIC64 would respond with an unsolicited bus request.

        lbg_n_out <= '0' WHEN ((lbr_n = '0') AND (lbg_n = '0')) ELSE '1';

-- This process captures the lbr_n signal from the VIC64 and double
-- registers it using the pclk signal.

        caputure_lbr: PROCESS BEGIN
            WAIT UNTIL pclk = '1';
            lbr_n_reg1 <= lbr_n;
            lbr_n_reg2 <= lbr_n_reg1;
        END PROCESS;
```

Appendix D. Bus Arbitration PLD (CY7C335) (continued)

```
-- gate_oe_n is triggered in a "Mealy" fashion to begin VIC64 cycles as
-- soon as possible.

        gate_oe_n <= '0' WHEN ((arb_state = slow_down040)
                                    OR
                               ((arb_state = only040) AND
                                   ((lock_n = '0' OR lbr_n_reg2 = '1') AND
                                    (selects /= no_selects)))
                                    OR
                               ((arb_state = both) AND
                                  ((lbr_n_reg2 = '1') AND
                                   (selects /= no_selects))))
                     ELSE '1';

        selects <= cs_n & strobe_n & mwb_n & memsel_n & fciack_n;

        arb_machine: PROCESS BEGIN
            WAIT UNTIL bclk = '1';
                CASE arb_state IS
                    WHEN reset =>
                        IF rst_n = '0' THEN
                            arb_state <= reset;
                            lbg_n <= '1';
                            bg_n <= '0';
                        ELSE
                            arb_state <= only040;
                            lbg_n <= '1';
                            bg_n <= '0';
                        END IF;
                    WHEN only040 =>
                        IF (lbr_n_reg2 = '0' AND lock_n = '1') THEN
                            arb_state <= both;
                            lbg_n <= '0';
                            bg_n <= '0';
                        ELSIF (lock_n = '0' OR lbr_n_reg2 = '1') AND
                              (selects /= no_selects) THEN
                            arb_state <= slow_down040;
                            lbg_n <= '1';
                            bg_n <= '1';
                        ELSE
                            arb_state <= only040;
                            lbg_n <= '1';
                            bg_n <= '0';
                        END IF;
                    WHEN slow_down040 =>
                        IF (xfer_done_n = '0') THEN
                            IF (lbr_n_reg2 = '0' AND lock_n = '1') THEN
                                arb_state <= both;
                                lbg_n <= '0';
                                bg_n <= '0';
                            ELSE
                                arb_state <= only040;
                                lbg_n <= '1';
```

Appendix D. Bus Arbitration PLD (CY7C335) (continued)

```
                        bg_n <= '0';
                    END IF;
                ELSE
                    arb_state <= slow_down040;
                    lbg_n <= '1';
                    bg_n <= '1';
                END IF;
            WHEN both =>
                IF (lbr_n_reg2 = '1') THEN
                    IF (selects /= no_selects) THEN
                        arb_state <= slow_down040;
                        lbg_n <= '1';
                        bg_n <= '1';
                    ELSE
                        arb_state <= only040;
                        lbg_n <= '1';
                        bg_n <= '0';
                    END IF;
                ELSE
                    arb_state <= both;
                    lbg_n <= '0';
                    bg_n <= '0';
                END IF;
            WHEN OTHERS => arb_state <= reset;
                           lbg_n <= '1';
                           bg_n <= '0';
        END CASE;
        IF (rst_n = '0') THEN
            arb_state <= reset;
        END IF;
    END PROCESS;
END operation;
```

Appendix E. Interrupt Synchronizing PLD (22V10D)

```
-- INTERRUPT PLD
--
-- The following table is a cross reference between the PLD port names and
-- the signals found on the physical IC's.
--
--    bclk             =  BCLK on 68040
--    iplx_n           =  IPLx* from VIC64
--    tmx              =  TM2-TM0 on 68040
--    board_reset_n    =  brd_rst_n from RESET PLD
--    fciack_n         =  fciack_n from ADDRESS DECODE PLD
--    gate_oe_n        =  gate_oe_n from BUS ARBITRATION PLD
--    xfer_done_n      =  xfer_done_n from TERMINATION PLD

--    normal_cycle_n   =  OE to '244's driving A3-A1 from 68040
--    a3, a2, a1       =  a3-a1 to VIC64
--    iplx_out_n       =  IPLx "bar" on 68040

ENTITY interrupt_ctrl IS
        PORT (bclk, ipl0_n, ipl1_n, ipl2_n : in bit;
              tm2, tm1, tm0 : in bit;
              board_reset_n, fciack_n : in bit;
              gate_oe_n, xfer_done_n : in bit;
              normal_cycle_n : out bit;
              a3, a2, a1 : inout x01z;
              ipl0_out_n, ipl1_out_n, ipl2_out_n : buffer bit);
END interrupt_ctrl;

use work.cypress.all;
use work.rtlpkg.all;

ARCHITECTURE operation OF interrupt_ctrl IS
     signal addr_oe, ipl0_n_reg, ipl1_n_reg, ipl2_n_reg : bit;
BEGIN

-- Synchronize the incoming ipl signals from the VIC64 to eliminate skew
     PROCESS BEGIN
          WAIT UNTIL bclk = '1';
               ipl0_n_reg <= ipl0_n;
               ipl1_n_reg <= ipl1_n;
               ipl2_n_reg <= ipl2_n;
     END PROCESS;

-- If the board is in reset, the ipl signals must be driven high to configure
-- the driver capability in the 68040.  Otherwise, the following equations
-- will keep the ipl signals from changing to the 68040 during an acknowledge
-- cycle and will synchronize them.  When the acknowledge cycle is finished,
-- the ipl signals will return to inactive state before reading the current
-- input values from the VIC64.

     PROCESS BEGIN
          WAIT UNTIL bclk = '1';
               IF board_reset_n = '0' THEN
                    ipl0_out_n <= '1';
                    ipl1_out_n <= '1';
```

Appendix E. Interrupt Synchronizing PLD (22V10D) (continued)

```
                ipl2_out_n <= '1';
            ELSIF xfer_done_n = '0' THEN
                ipl0_out_n <= '1';
                ipl1_out_n <= '1';
                ipl2_out_n <= '1';
            ELSIF fciack_n = '0' THEN
                ipl0_out_n <= ipl0_out_n;
                ipl1_out_n <= ipl1_out_n;
                ipl2_out_n <= ipl2_out_n;
            ELSE
                ipl0_out_n <= ipl0_n_reg;
                ipl1_out_n <= ipl1_n_reg;
                ipl2_out_n <= ipl2_n_reg;
            END IF;
    END PROCESS;

-- The normal_cycle_n signal is low most of the time to enable the a3-a1
-- signals from the 68040 to the VIC64.  However, if we are in an interrupt
-- acknowledge cycle, we would steer the tm2-tm0 signals from the 68040
-- onto the a3-a1 signals on the VIC64 since the tmx signals indicate which
-- interrupt level is being acknowledged

    normal_cycle_n <= NOT (fciack_n AND (NOT gate_oe_n));

    addr_oe <= NOT fciack_n;
    a3_map: bufoe port map(tm2, addr_oe, a3, open);
    a2_map: bufoe port map(tm1, addr_oe, a2, open);
    a1_map: bufoe port map(tm0, addr_oe, a1, open);

END operation;
```