# Using the Slave VIC (CY7C960/961)

Many VME boards, especially I/O boards, need only be aware of VME Slave transactions. Most commercially available VME interface chips are capable of both Master and Slave VME transactions and require some local intelligence, such as a microprocessor, to reset and program the interface chip. I/O-only boards do not need a microprocessor since information is simply passed to and from the I/O without being processed in between (at least in the simplest case) so the addition of a microprocessor, or any other kind of intelligence such as a state machine, only adds to the cost of the interface in design time, board space, and money. The most common solution to the problem of a slave-only interface is an FPGA, which still adds extra cost in the form of design time, board space, and the cost of the FPGA.

A better solution to this problem is Cypress's Slave VME Interface Controller (SVIC) Family: the CY7C960 and the CY7C961. An SVIC, along with four Bus Interface Logic chips (CY7C964), implements a complete VME64-compliant slave-only VME interface that requires no microprocessor and occupies minimum board space.

## Index



## CY7C960/961 Features

- Full VME64 Slave transaction support
- DRAM/Refresh Controller
- CY7C964 Control Interface
- I/O (Chip Select Output) Controller
- VMEbus Interrupter
- Address Modifier (AM) Code Discriminator
- Slave Address Region Decoder
- Limited Master Support (CY7C961 only)

## Slave VIC Operation Overview

*Figure 1* shows the internal blocks that comprise the CY7C960. The CY7C960 Slave VMEbus Interface Controller (SVIC) provides the board designer with an integrated, full-featured VME64 interface. This 64-pin device can be programmed to handle every transaction defined in the VME64 specification. The CY7C960 contains all the circuitry needed to control large DRAM arrays and local I/O circuitry without the intervention of a local CPU. There are no registers to read or write, and no complex command blocks to be constructed in memory. The CY7C960 simply fetches its own configuration parameters during the power-on reset period.

After reset, the CY7C960 responds appropriately to VMEbus activity and controls local circuitry transparently. The CY7C960 controls a bridge between the VMEbus and the local DRAM and I/O. Once programmed, the CY7C960 provides activities such as DRAM refresh and local I/O handshaking in a manner that requires no additional local circuitry.

The VMEbus control signals are connected directly to the CY7C960. The VMEbus address and data signals are connected to companion address/data transceivers which are controlled by CY7C960. The CY7C964 VMEbus Interface Logic Circuit is an ideal companion device. The CY7C964 provides 8 bits of data and address logic that has been optimized for VME64 transactions. In addition to providing the specified drive strength and timing for VME64 transactions, the CY7C964 contains all of the circuitry needed to multiplex the address/data bus for multiplexed VMEbus transactions. It contains counters and latches needed during BLT (Block Transfer) operations. It also contains address comparators which can be used in the board's Slave Address Decoder. For a 6U or 9U application, four CY7C964 devices are controlled by a single CY7C960. For 3U applications, the CY7C960 controls two CY7C964 devices and an address latch.

The design of the CY7C960 makes it unnecessary to know the details of the VMEbus transaction timing and protocol. The complex VMEbus activities are translated by the CY7C960 to be simple local cycles involving a few familiar control signals. Similarly, it is not necessary to understand the operation of the companion device, the CY7C964; all control

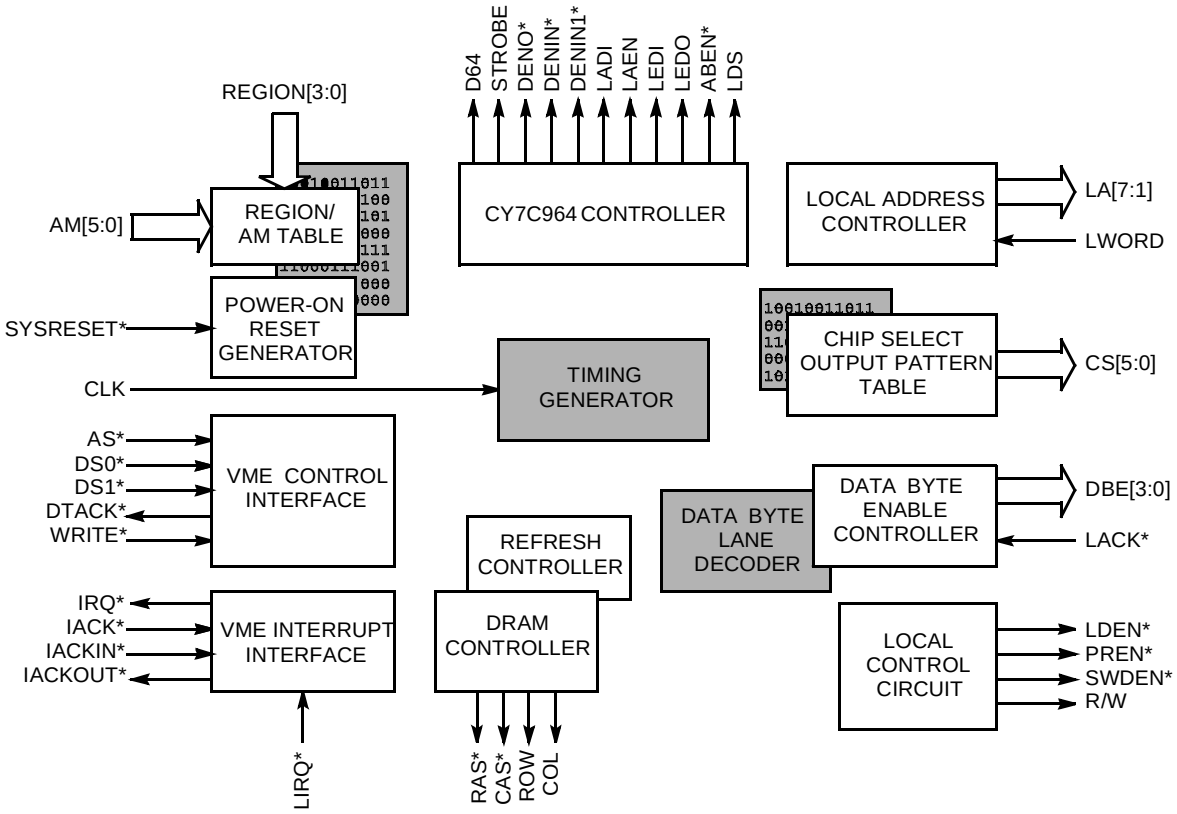


**Figure 1. Internal Block Diagram of the CY7C960**

sequences for the part are generated automatically by the CY7C960 in response to VMEbus or local activity.

VMEbus transactions supported by CY7C960 include D8, D16, D32 (including UAT), MD32, D64, A16, A24, A32, A40, A64 single cycle and block transfer reads and writes, Read-Modify-Write cycles (including multiplexed), and Address-only (with or without Handshake). The CY7C960 functions as a VMEbus Interrupter, and supports the Auto Slot ID standard and CR/CSR space. The CY7C960 also handles LOCK cycles, although full LOCK support is not possible within the constraints of the CY7C960 pinout (full LOCK support is included in the CY7C961).

On the local side, no CPU is needed to program the CY7C960 nor to manage transactions. All programmable parameters are initialized through the use of either the VMEbus or a serial PROM. As the CY7C960 incorporates a reliable power-on reset circuit, parameters are self-loaded by the device at power-up or after a system reset. If the VMEbus is used to provide parameters, a VMEbus Master provides the programming information using a protocol that is compliant with the Auto Slot ID protocol from the VME64 specification.

The architecture of the SVIC includes several functions that remove most of the VMEbus problems from the board designer's shoulders. All VMEbus control and response is automatic; the user loads the Region/AM table during configuration, and the CY7C960 then handles all appropriate VMEbus transactions. The CY7C964 controller works in lock step with the VMEbus Control Interface, providing the correct timing and control for the transaction in process. Local circuitry such as DRAM or I/O 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. Local timing is determined during configuration, and handshaking is available from the Data Byte Enable Controller. Local Interrupts are supported through the VME Interrupt Interface. The CY7C960 contains an internal Power-on Reset circuit, and also responds to a VMEbus SYSRESET*.

## General Overview

*Figure 2* illustrates a block diagram of a slave-only VME interface using one CY7C960/961 and four CY7C964s. No external glue logic is required when using the SVIC. The SVIC directly drives up to 6 Chip Selects (CSs) and four Data Byte Enables (DBEs) for interfacing to local resources. Depending on the requirements of your design, there may be a need for some external logic to implement a SWAP buffer, DRAM address interface, interrupt generation, and/or REGION decoding. The extent of this external logic would consist mainly of buffers (244s and 245s) and a PLD. The amount and complexity of external logic required is scalable depending on the requirements of your design. This application note concentrates on the design of these external logic components and on the interconnection of these components to the SVIC.

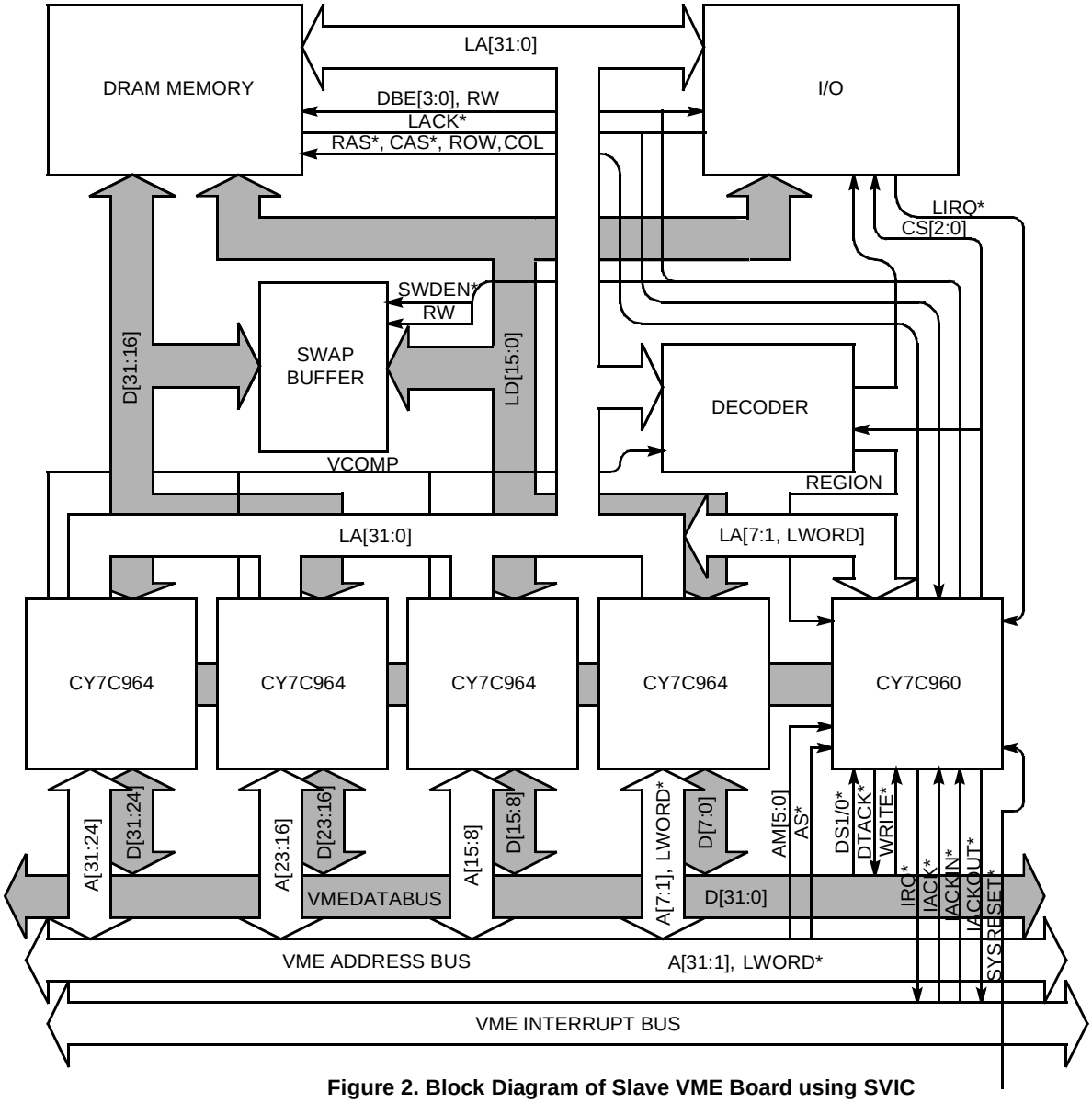


**Figure 2. Block Diagram of Slave VME Board using SVIC**

## Design Issues

### DRAM Interface

The SVIC can be programmed (through the use of the WINSVIC software, as explained in the SVIC Users Guide) to operate in one of two modes: DRAM/IO or I/O Only. While in DRAM/IO mode the SVIC is capable of controlling a bank of DRAM through the use of RAS* (Row Address Strobe) and CAS* (Column Address Strobe) signals along with performing DRAM refresh (programmable timings). In order to speed up the access to DRAM, every time the AS* (Address Strobe) goes LOW on the VMEbus, the RAS* signal goes LOW on the SVIC, causing the row address to be pre-latched into the DRAM. If the cycle was not meant for the DRAM then no harm was done, since a RAS-only cycle does not cause any reading or writing from/to the DRAM. But if the cycle was meant for the DRAM then half of the DRAM access has already occurred with only the CAS part of the cycle remaining.

Due to the fact that the address passes through the CY7C964s and not the SVIC itself, external buffers (244s) are required to separate the row and column address from the full address. Enabling these 244s at the proper time is accomplished by the ROW and COL outputs from the SVIC.

Another important issue to deal with is distinguishing DRAM accesses from I/O accesses (when DBE[3:0] are used as CAS*). If the DBEs (Data Byte Enables) are programmed to act as CAS*, an assertion of DBE due to an I/O access will look like an assertion of CAS* to the DRAM, and will thus complete a RAS-CAS DRAM access. A solution to this issue is to gate a Chip Select from the SVIC with the DBEs to determine when the CAS* input on the DRAM should be driven LOW. An example of how this can be accomplished is found in the Design Examples section that follows.

**Swap Buffer**

Most modern designs utilize memories that are 32 bits wide. The VME64 Specification allows for transactions that are 8, 16, 32, and 64 bits wide, which require reads and writes to resources in 8-, 16- and 32-bit-wide slices that may or may not be aligned to word boundaries. If 8- or 16-bit-wide transactions to 32-bit-wide local resources are to be allowed on your board, a Swap Buffer, comprised of 245s and controlled by the SVIC, needs to be included in the design of the slave board. If transactions are to be limited to the size of the local data size (i.e. only D32 to 32-bit-wide local data or only D16 to 16-bit-wide local data) the Swap Buffer can be omitted and the local data bus can be tied directly to the CY7C964s. An example of how to implement the Swap Buffer can be found in the Design Examples section that follows.

**Region Decoder**

One of the most flexible features of the SVIC is the ability to react differently depending on where in the slave board's local address map a VME transaction is destined. Think of the local address map as being logically broken up into blocks of space referred to as regions. The local address map can be broken up into as many regions (up to 16) as required by your design. The size of each region is completely arbitrary and each region need not be of the same size. For example, 4 MBs of DRAM may sit in one region while 32K of SRAM may sit in another. The SVIC is told which region of the local memory map is being addressed based on what value is being asserted onto the REGION inputs.

The SVIC has four REGION inputs when in I/O Mode and three REGION inputs when in DRAM Mode. The value that is asserted on the REGION inputs is the job of the Region Decoder. The most common method used to determine which REGION value should be asserted to the SVIC is VME address decoding.

A comparison between the VME address that is placed on the VMEbus by the Master, and the VME address space in which the Slave board sits (Slave Base Address) will determine if the current VME transaction is destined for this particular Slave board. If the SVIC is to handle one and only one set of VME transactions (i.e., always A16 and A24 transactions), a comparison of the VME address and the Slave Base Address will be all that is required when deciding which REGION value to assert. In this example, a 'true' from the comparison logic will indicate that it is this board that is being addressed and that the region that has been programmed to allow A16 and A24 transactions should be asserted to the SVIC's REGION inputs.

If the SVIC is required to react differently when accessing different local resources, i.e. A16 (but not A24 or A32) transactions when addressing SRAM space and A16 and A32 (but not A24) transactions to DRAM space, the fact that it is this board being addressed is not enough to determine which REGION value to assert to the SVIC since the SVIC is required to react differently depending on which part of SVIC local address map is being addressed. In this case, further VME address decoding must be done by the Region Decoder to determine which region of the SVIC board is being addressed.

During initialization the SVIC is loaded with its configuration parameters. The configuration parameters are chosen using a free, Cypress-supplied software called WINSVIC. The WINSVIC software allows you to choose the configuration that is applicable to your design and outputs a file consisting of your chosen parameters encoded into 380 bits. These 380 bits are fed into the SVIC during initialization to fully configure the device. These configuration parameters consist of global parameters (those parameters that define the general operation of the chip) and Region parameters (those that define what type of VME transactions that the SVIC is allowed to handle and which Chip Selects will be driven if the current VME transaction is handled by the SVIC).

The SVIC is loaded with 16 sets of Region parameters when in I/O Mode and 8 sets of Region parameters when in DRAM Mode. Out of these many sets of Region parameters only one set is valid and being used to define the operation of the SVIC at any one time. Which set of Region parameters that the SVIC should consider valid is determined by the user through the use of the REGION inputs (i.e., placing 3H on the REGION inputs will tell the SVIC to use the Region number 3 parameters when deciding if the current VME transaction should be handled).

The role that the Region parameters play in determining the operation of the SVIC is as follows:

1. Master places VME address, VME data (if a write), Address Modifier Codes (AM Codes), and strobes onto the VMEbus.
2. SVIC sees the strobes, waits a programmed period of time (known as the Decode Delay) and samples the REGION inputs.

   At this time the SVIC knows what type of VME transactions it will respond to.
3. SVIC looks at the AM Codes on the VMEbus (which define what type of transaction the Master is requesting) and compares the type of transaction requested with the types of transactions that it is allowed to handle (based on Region parameters).
4. If there was a match between requested and allowed transactions, the SVIC will drive the programmed Chip Selects (CS) and will handle the requested transaction. If there was not a match the SVIC would ignore this VME transaction.

Because the REGION inputs are driven by local logic, the determination of which region is being addressed at any given time is determined by the designer of the Region Decoder. The purpose of the Region Decoder is to determine if the address on the VMEbus falls into the address map of the SVIC. The address map of the SVIC can consist of up to 16 different regions, each of which can be of different sizes.

*Figure 3* is an example of how a VME address can be mapped into regions. The first thing to note is that at least one region must not exist in the local address map. In this example, Regions 0 and 3 and Regions 7 through 15 do not exist in the local address map. The SVIC should be programmed to ignore all AM codes when the REGION inputs are being driven with 0 or 3 or 7 through 15. When the VME address does not fall within the Slave board's address space, it is one of these unused or 'turned-off' regions that should be asserted to the SVIC.

Another thing to notice is how the address map is decoded into regions. This example assumes that the SVIC is being addressed when the most significant byte (A[31:24]) of the address is FF (Slave Base Address = FFxxxxxx). The next

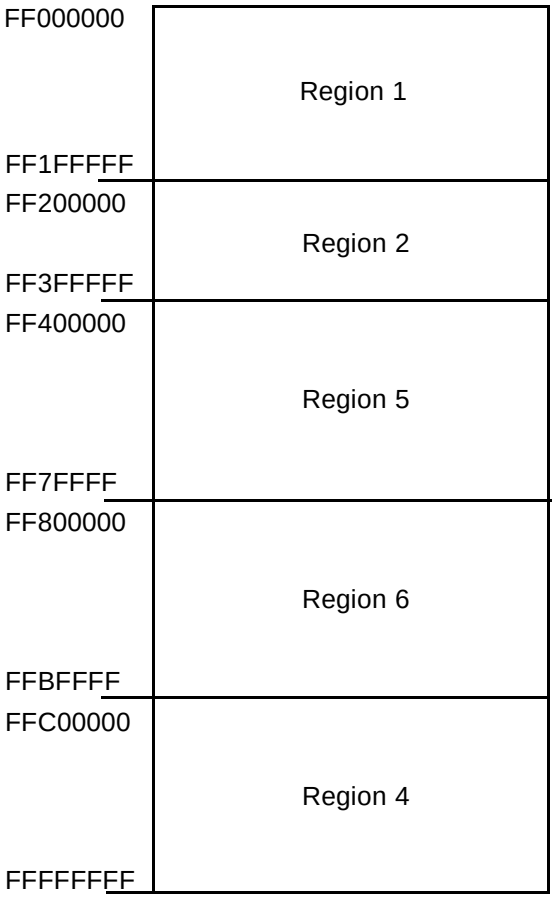


**Figure 3. Example of an SVIC Address Map**

nibble (A[23:20]) determines what region is being addressed and the rest of the address (A[19:0]) is decoded as the offset within the region. This address decoding scheme assumes 32-bit addresses. Because VME addresses can be of varying sizes, a design that would allow accesses in different address modes (A16, A32, etc.) will need to be aware of what address mode is being used for each transaction. Because this information is encoded in the AM Codes, the easiest thing to do is to include the VMEbus AM Code along with the address when decoding the region.

As this address map illustrates, regions need not be of the same size. The regions do not need to be in numerical order nor do all the regions need to appear in the address map.

### Local Interrupts

The SVIC has one interrupt request pin (LIRQ*) available to local resources. Assertion of the LIRQ* pin by local resources causes a VME interrupt to occur. Upon acknowledgment of the VME interrupt by a master, through the use of the IACK daisy chain, the SVIC informs the local logic to place a Status/ID word onto the local data bus. This Status/ID word is read by the responding master and the interrupt acknowledge sequence is complete.

If more than one interrupter exists on the local side of the SVIC, each interrupter must share the LIRQ* pin but can drive a different Status/ID word. It is the Status/ID word that truly distinguishes one interrupter from another. If more than one interrupt is pending at the same time it is up to local logic to perform interrupt priority. The complexity and size of the local interrupt logic is a function of the number of interrupters on the local side and the priority algorithm being implemented.

### A64/A40 Support

The SVIC is capable of performing transactions in A64 and A40 address space. A64 addresses are transmitted over the VMEbus by multiplexing the 32-bit address and the 32-bit data buses that are available to 6U and larger VME cards. A40 addresses are transmitted over the VMEbus by multiplexing the 24-bit address and the 16-bit data buses that are available to 3U and larger VME cards. To support A64/A40 BLTs, the upper bits of the address (which are carried on the data bus) must be latched into external buffers for use in later cycles. The address is latched into and driven out of these latches (373s) at the proper time by signals that are sourced by the SVIC. If the SVIC is not programmed to handle A64 or A40 transactions then these external latches can be omitted from the design.

### CY7C964 Interface

CY7C964s are directly controlled by the SVIC for use as the address and data glue logic between the VME and Local buses. The actual interconnections between the SVIC and the four CY7C964s is documented in the next section (Design Examples).

### MD32 Support

Additionally, the VME64 Specification supports 32-bit-wide data transfers on 3U VME cards known as MD32 transactions. 3U VME cards only have a 16-bit data bus and a 24-bit address bus available to them. In order to transfer 32 bits of data at a time, the two buses are multiplexed with two bytes of data carried on the data bus and the other two bytes of data being carried on the address bus. Additions to a design for support of MD32 transactions include the control of the upper two CY7C964's DENIN* and DENIN1* (Data Enable In) inputs. The DENIN* and DENIN1* pins on 964-2 and 964-3 should be connected to the modified DENIN signals (MOD_DENIN* and MOD_DENIN1*, respectively, see *Figure 4*) and are only required if D64 transactions are to be supported on the same board.

## Design Examples

### DRAM Interface

*Figure 5* illustrates how an SVIC can be interfaced to a bank of DRAM. This example uses a 4-MB 70-ns SIMM as the DRAM bank. This 4-MB SIMM requires ten bits of address and uses a 32-bit (4-byte) data word. The SIMM also has a separate CAS* (which is generated by the FLASH375) and RAS* for each data byte. The FLASH375 filters out DBE[3:0] assertions due to I/O access and allows DBE[3:0] assertions meant for the DRAM to be passed out to the CAS[3:0] lines.

Three buffers (244s) are used for separating the row and column address from the local address. Enabling of the row and column address buffers is accomplished by the SVIC by the assertion of ROW and COL. The latching of the address into the DRAM is controlled by the SVIC with the RAS* and CAS* signals.

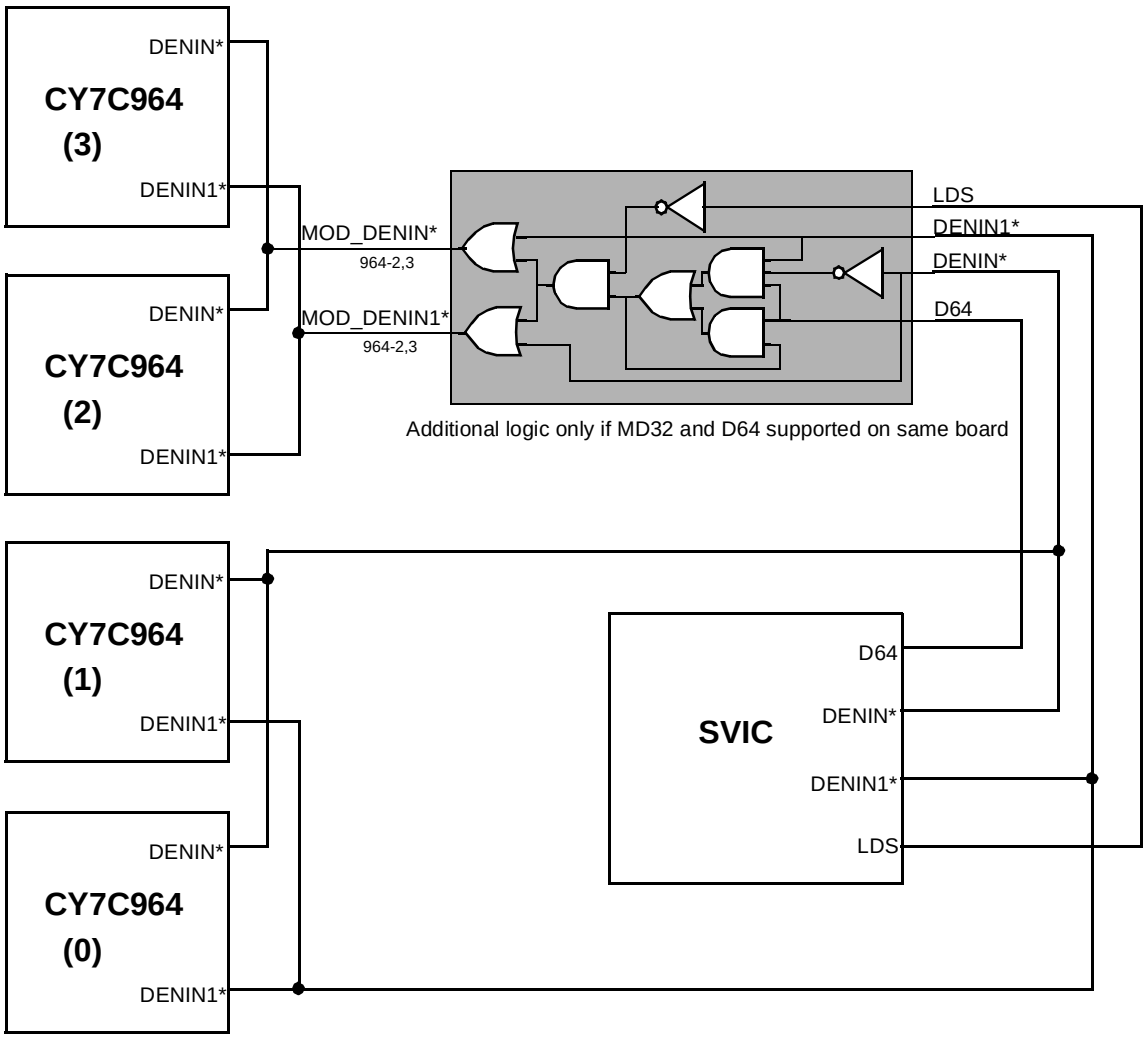


**Figure 4. Additional Logic for MD32 Support**

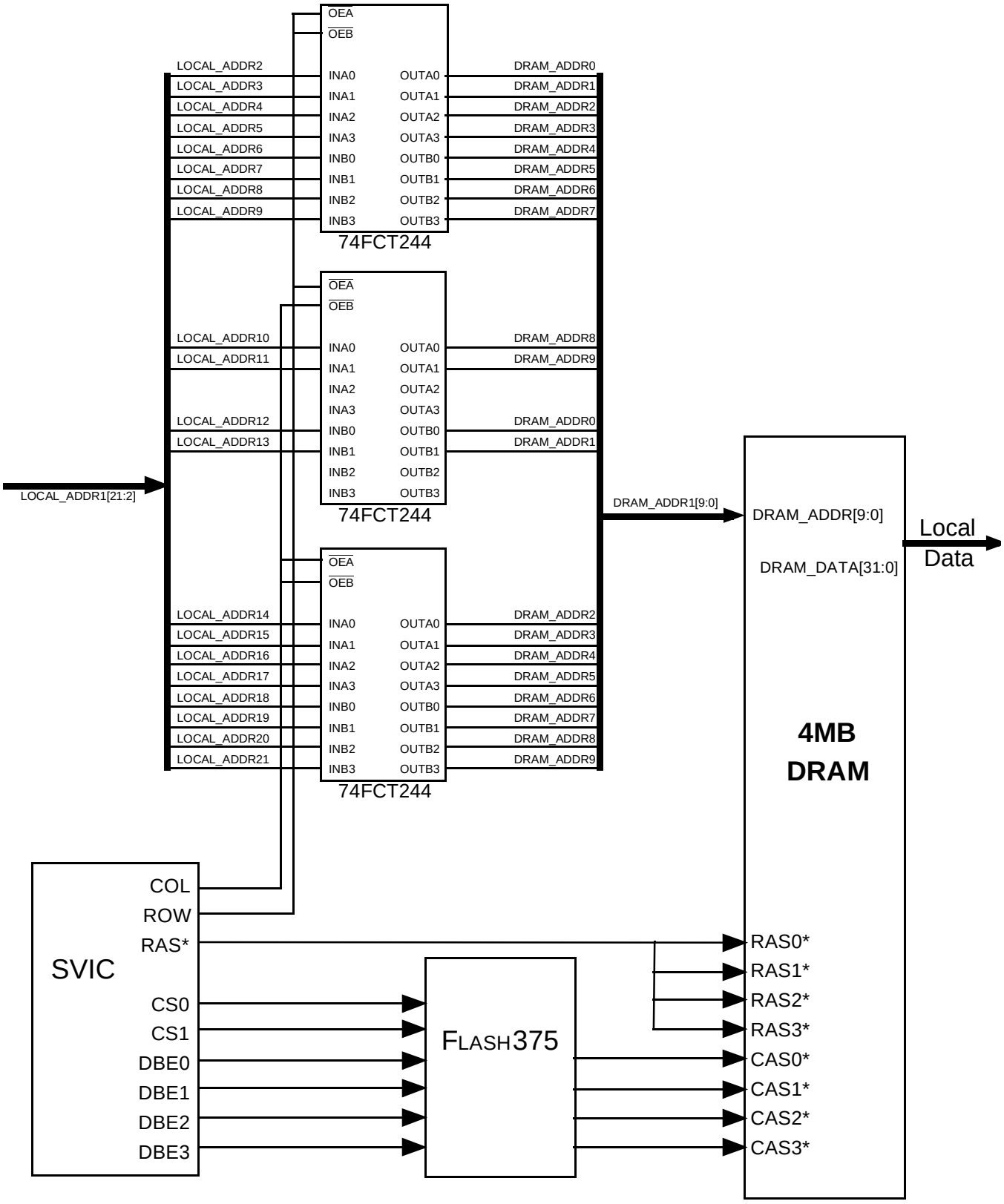


**Figure 5. DRAM Interface Logic Example**

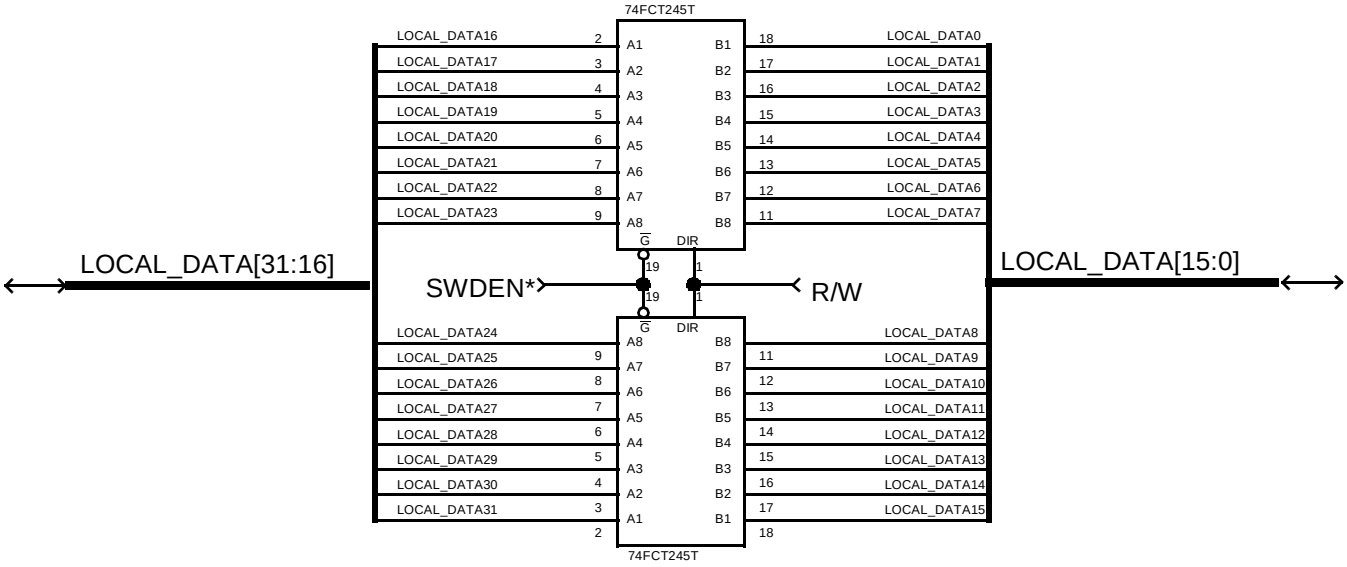


**Figure 6. SWAP Buffer Implementation Example**

### Swap Buffer

*Figure 6* shows the implementation of a Swap Buffer on the SVIC Board. The Swap Buffer is simply two '245 transceivers with the DIR and EN* control lines connected directly to the SVIC. The purpose of the Swap Buffer is to place LD[31:16] onto the LD[15:0] lines, and vice versa, for performing D16 transactions to 32-bit local resources.

### Region Decoder

The Region Decoder for this example is designed to take full advantage of the CY7C960/961. Each of the sixteen possible regions can be individually addressed regardless of the VME address space (A64, A40, A32, A24, and A16) being used. Because of the amount of logic and I/O pins used in the SVIC Board Region Decoder, it was decided to write the decoder in VHDL (see Appendix A, VHDL Code) and program it into a FLASH375 PLD. A simple diagram showing the inputs and outputs to our Region Decoder can be seen in *Figure 7*. The Region Decoder itself would fit into a smaller PLD but since several other parts of the example Board design were placed into a PLD (such as the Interrupt Logic) the FLASH375 was used due to the need for many I/O pins (especially for 32 bits of address and 32 bits of data). Most Region Decoders should require no more than 15–20 I/O pins and 50–100 gates.

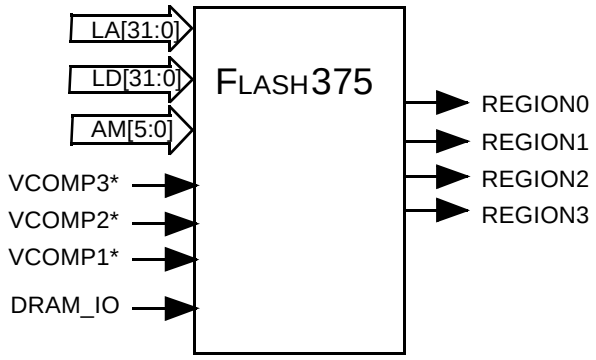


**Figure 7. Inputs and Outputs of the Region Decoder Logic**

We mapped the SVIC Board into the VME address space as follows: the four most significant bits of the VME address are decoded to determine if it is this board that is being addressed. If it is this board that is being addressed then the next four significant bits are decoded as the region.

The challenge is to determine which are the most significant address bits. For example, in A32 space the most significant address bits start at A[31] but in A40 space the most significant bits start at D[15]. The only way to know which address space is being used by the VME Master that is initiating the transaction is to decode the AM Codes from the VMEbus. The AM Codes tell where the most significant VME address bits lie and the address bits tell which region is being addressed (if any).

The Region Decoder VHDL code begins with a CASE statement that uses AM Codes to determine which addressing mode is being used by the VME Master. The use of all 6 bits of the AM Code in the CASE statement was for ease of reading and not by necessity. All that would be required to determine the addressing mode is the three most significant bits of the AM Code.

Once the addressing mode is determined (i.e., the location of the most significant address bits is found) it can be determined if it is this board that is being addressed. Performing an address comparison on the four most significant address bits determines this. For A64 and A40 transfers the address bits themselves must be looked at, but for A32, A24, and A16 transfers the CY7C964s can be used to perform the comparison.

Each CY7C964 performs a comparison between the 8 bits of VME address that it is attached to and a Compare Address and Mask value that are written into each CY7C964 during configuration. A comparison between the 8 bits of VME address and the Compare Address (w/Mask) will result in the VCOMP output from the CY7C964 being driven LOW (see the VMEbus Interface Handbook).

If the comparison produces a match it must be determined which region is being addressed. For many designers this

may be a fixed region that will require no further decoding of the address. The SVIC Board allows all 16 regions to be addressed by a VME Master by driving the second most significant nibble of the address onto the REGION inputs. The driving of the REGION3 input of the SVIC is controlled by an input to the Region Decoder on the SVIC Board called DRAM_IO. This functionality was included to allow the SVIC Board to function in both DRAM/IO Mode (3 REGION inputs) and I/O Mode (4 REGION inputs) depending on how the SVIC is programmed. Most slave boards will operate in only one mode, depending on what resources have been designed onto the board, so it will be known how many REGION inputs must be driven by the decoder thus eliminating the need for the DRAM_IO input function.

**Local Interrupts**

The VHDL Code located in Appendix A contains the code used on the SVIC Board for the Interrupt Logic. *Figure 8* shows the inputs and outputs to the Local Interrupt Logic. The SVIC Board is capable of generating VME interrupts from four different local sources, each with its own Status/ID word. The Interrupt Logic VHDL Code also handles AUTO ID and the Compare and Mask loading of the CY7C964s.

LIRQ* (Local Interrupt Request) will be driven LOW when one or more of the LIRQi* inputs on the FLASH375 are driven LOW. When LDEN* (Local Data Enable) is driven LOW and MWB* (Module Wants Bus) is HIGH, a value must be driven onto the Local Data (LD) bus. The value that must be driven onto the LD bus will either be a Status/ID associated with a local interrupt, the STATUS/ID associated with VMEbus Initialization (AUTO ID) or the Compare and Mask for the CY7C964s.

The Local Interrupts have been assigned priority in the VHDL Code with LIRQ1* having the highest priority and LIRQ4* having the lowest. *Table 1* is a summary of what is driven onto the Local Data bus when LDEN*=0.

**Table 1. Summary of What Is Driven onto the Local Data Bus when LDEN*=0**

| What is driven onto the Local Data bus | When it is driven onto the Local Data bus |
|---|---|
| Interrupt Status/ID | LDEN*=0, MWB*=1, PREN*=1, LIRQi*=0 |
| AUTO ID Status/ID | LDEN*=0, MWB*=1, PREN*=1, LIRQi*=1 |
| CY7C964 Compare | LDEN*=0, MWB*=1, PREN*=1, LDS=1 |
| CY7C964 Mask | LDEN*=0, MWB*=1, PREN*=1, LDS=0 |

**A64/A40 Support**

The A64/A40 Support built into the SVIC Board consists of latches ('573/'373s) on the Local Data (LD) bus for use in latching the address bits that are carried on the LD bus during multiplexed address cycles (see *Figure 9*).

A40 support requires the latching of LD[15:0] while A64 support requires the latching of LD[31:0]. The control equations for latching and enabling (LA_UP_ADDR and EN_UP_ADDR) are located in Appendix A, VHDL Code.

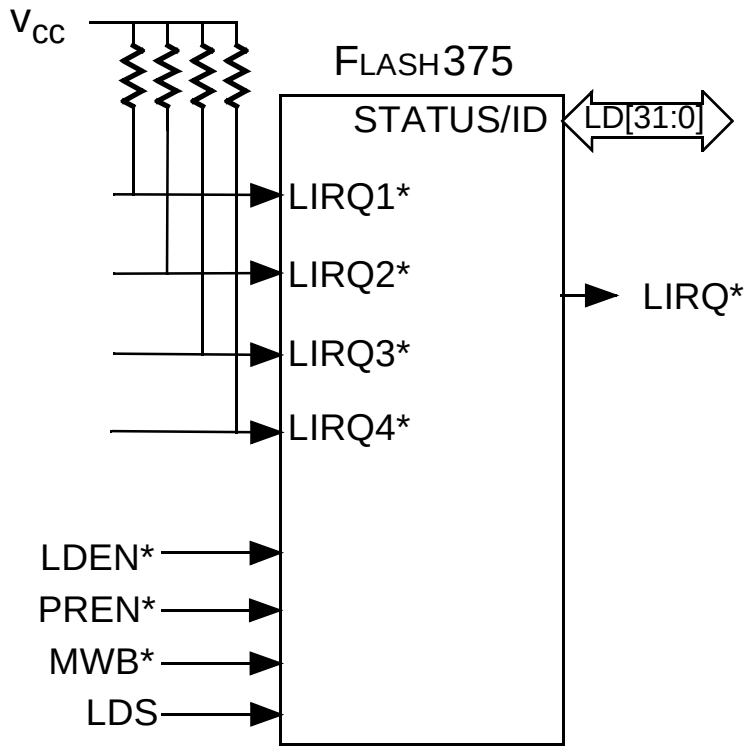


**Figure 8. Inputs and Outputs to Local Interrupt Logic**

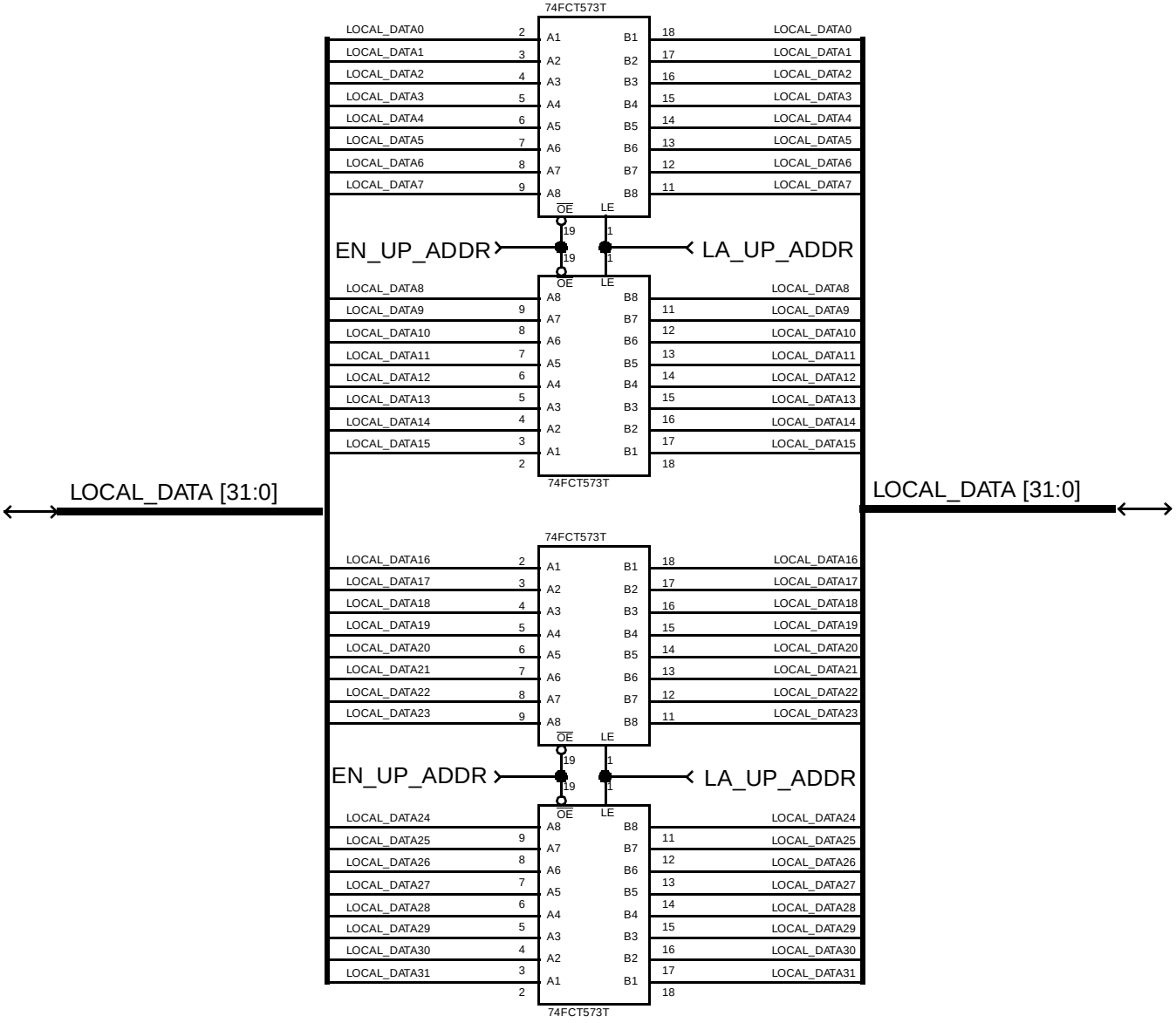


**Figure 9. Additional Logic for A64/A40 Support**

### CY7C964 Interface

The SVIC Board utilizes four CY7C964s to act as the bridge between the VMEbus and the local buses. The interconnections between the CY7C961 and the CY7C964s are summarized in *Table 3*. The table is organized with one row for each CY7C964 pin (or bus for A, D, LA, LD) and one column per each CY7C964 (964-0, 1, 2, 3). The last column of *Table 3* is for users of the CY7C960. An entry in this column should replace the entries in the other columns in that row when the CY7C960 is being used.

All signals are sourced from the SVIC unless the name of a source appears in parentheses under the signal name. For example, in the row below (*Table 2*): the LCIN* pin on the least significant CY7C964 (964-0) should be connected to $V_{CC}$, the LCIN* on the next CY7C964 should be connected to GND, LCIN* on 964-2 should be connected to the LCOUT* pin on 964-1 and LCIN* on 964-3 should be connected to the LCOUT* pin on 964-2. Since the last column is empty there is no difference in the connections to the LCIN* pin when using the CY7C960 as apposed to using the CY7C961.

### MD32 Support

MD32 support on the SVIC Board consists of creating modified DENIN*/DENIN1* (MOD_DENIN*/ MOD_DENIN1*) signals for use in control of the two most significant CY7C964s. If MD32 and D64 transactions are to be supported on the same board, the entries in the CY7C964 Interface table for DENIN* and DENIN1* should be replaced with the entries in *Table 4*.

**Table 2. Example Row from *Table 3***

| CY7C964 Pin | 964-0 LSB | 964-1 | 964-2 | 964-3 MSB | If Using the CY7C960 |
|---|---|---|---|---|---|
| **LCIN*** | $V_{CC}$ | GND | LCOUT* (964-1) | LCOUT* (964-2) | |

**Table 3. Connections Between the SVIC and Four CY7C964s**

| CY7C964 Pin | 964-0 LSB | 964-1 | 964-2 | 964-3 MSB | If Using the CY7C960 |
|---|---|---|---|---|---|
| **A[7:0]** | A[7:1],LWORD (VME) | A[15:8] (VME) | A[23:16] (VME) | A[31:24] (VME) | |
| **D[7:0]** | D[7:0] (VME) | D[15:8] (VME) | D[23:16] (VME) | D[31:24] (VME) | |
| **LA[7:0]** | LA[7:0] (LOCAL) | LA[15:8] (LOCAL) | LA[23:16] (LOCAL) | LA[31:24] (LOCAL) | |
| **LD[7:0]** | LD[7:0] (LOCAL) | LD[15:8] (LOCAL) | LD[23:16] (LOCAL) | LD[31:24] (LOCAL) | |
| **ABEN*** | ABEN* | ABEN* | ABEN* | ABEN* | |
| **BLT*** | BLT* | BLT* | BLT* | BLT* | $V_{CC}$ |
| **D64** | D64 | D64 | D64 | D64 | |
| **DENIN*** | DENIN* | DENIN* | DENIN1* | DENIN1* | |
| **DENIN1*** | DENIN1* | DENIN1* | DENIN* | DENIN* | |
| **DENO*** | DENO* | DENO* | DENO* | DENO* | |
| **FC1** | FC1 | FC1 | FC1 | FC1 | GND |
| **LCOUT*** | N/C | LCIN* (964-2) | LCIN* (964-3) | N/C | |
| **LDS** | LDS | LDS | LDS | LDS | |
| **LADI** | LADI | LADI | LADI | LADI | |
| **LAEN** | LAEN | LAEN321 | LAEN321 | LAEN321 | $V_{CC}$ on 964-1,2,3 LAEN on 964-0 |
| **LEDI** | LEDI | LEDI | LEDI | LEDI | |
| **LEDO** | LEDO | LEDO | LEDO | LEDO | |
| **LADO** | VMECNT | LADO | LADO | LADO | GND |
| **LCIN*** | VCC | GND | LCOUT* (964-1) | LCOUT* (964-2) | |
| **MWB*** | MWB* | MWB* | MWB* | MWB* | VCC |
| **STROBE*** | STROBE* | STROBE* | STROBE* | STROBE* | |
| **VCOMP*** | AS NEEDED | AS NEEDED | AS NEEDED | AS NEEDED | |
| **VCIN*** | GND | GND | VCOUT* (964-1) | VCOUT* (964-2) | |
| **VCOUT*** | N/C | VCIN* (964-2) | VCIN* (964-3) | N/C | |

**Table 4. Modified DENIN connections for MD32 Support**

| CY7C964 Pin | 964-2 | 964-3 |
|---|---|---|
| **DENIN*** | MOD_DENIN* | MOD_DENIN* |
| **DENIN1*** | MOD_DENIN1* | MOD_DENIN1* |

**Required Resistors**

The following signals need pull-up or pull-down resistors:

PULL-UP: BLT*, MWB*, ABEN*, DENO*, PREN*

PULL-DOWN: LAEN, LADI

In addition, if the CY7C960 is being used, FC1 and LADO on the CY7C964s must be tied LOW.

### Serial PROM

The SVIC needs to be configured at power-up. The configuration consists of approximately 380 bits of serial data into the part from either the VMEbus or through the use of a serial PROM from the local bus. There are several serial PROMs that are compatible with the SVIC: the AT&T ATT1718 and ATT1736, Xilinx XC1718, XC1736 and XC1765 and Atmel 'Configurator' AT17C65, AT17C128. The numbers following the 17 in each of the part numbers indicate the number of Kbits that the part holds. All of these PROMs have a programmable RESET/Output Enable (R/OE) pin, and the SVIC expects the RESET to be active HIGH. The RESET/OE on these PROMs are programmed to be active HIGH by writing ones into a special memory location. The memory location that must be written (with ones) varies by PROM size. The memory addresses are shown in *Table 5*.

**Table 5. PROM Addresses**

| PROM Size | Address |
|---|---|
| 18K | 8DC-8DF |
| 36K | 11B8-11BB |
| 65K | 2000-2003 |
| 128K | 4000-4003 |

Active HIGH Reset: fill address with ones

*Figure 10* illustrates the connections between the SVIC and the serial PROM. The R/OE pin should be connected to the PREN* output of the SVIC. R/OE should also have a pull-up resistor to ensure that the internal pointer is reset to the first position. The Chip Enable (CE) pin can be either tied LOW or tied to the R/OE pin, the Clock (CLK) pin should be connected to the LA[1]/PCLK pin of the SVIC and, finally, the Data (D) pin should be connected to the LA[2]/PDATA pin on the SVIC. Note that PCLK is a resistive programmable pin, a pull-up resistor is connected to PCLK; this will program the CY7C960 to source PCLK to the PROM.

## Summary

This application note has shown how easy it is to design a fully VME64-compliant Slave VME board using the Cypress Slave VME Interface Controller (SVIC) Family (CY7C960/961). Along with four CY7C964s (Bus Interface Chips), a PLD, and a small amount of TTL logic, a Slave VME board capable of D8 through D64/A16 through A64 transactions can easily be designed in a short amount of time.

Discrete components and VHDL code were used to design the little off-chip logic that was used on the SVIC Board. Along with examples on how to interface the SVIC to the VMEbus, significant examples on how to interface the SVIC to DRAM and I/O were also discussed. The design of optional logic, such as the SWAP Buffer and Local Interrupt logic was explained for those boards requiring it.

A discussion of regions was included to help in the understanding of this topic. Also included for completeness was a discussion on which serial PROMS could be used and where resistors should be added.

The CY7C960 or CY7C961 along with four CY7C964s comprises the most complete and easy to design fully VME64-compliant Slave VME Interface on the market today.

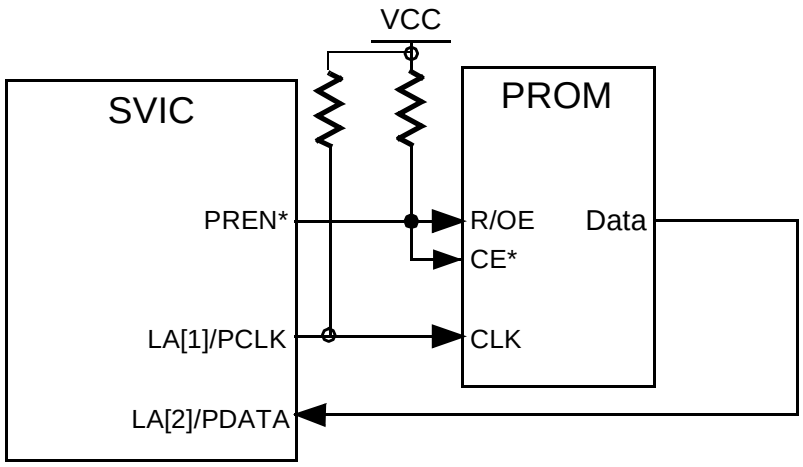


**Figure 10. Connection of SVIC to Serial PROM**

## Appendix A. VDHL Code

```
-- vhdl code for the SVIC Board

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


ENTITY logic IS
    PORT (D64, LDS, DENIN, PREN, DENIN1, LEDI, LDEN, MWB, LIRQ1, LIRQ2, LIRQ3, LIRQ4, DRAM_IO,
      CS0, CS1, DBE0, DBE1, DBE2, DBE3, RW, RESET, SYSRESET: IN BIT;
      MOD_DENIN, MOD_DENIN1, LA_UP_ADDR, EN_UP_ADDR, LIRQ, CE0, CE1, CE2,CE3, CAS0, CAS1,
      CAS2, CAS3, OE, SVIC_RESET: OUT BIT;
      SELECTLM: INOUT BIT;
      LA: IN x01z_VECTOR(31 downto 8);
      AM:  IN BIT_VECTOR(5 downto 0);
      VCOMP: IN BIT_VECTOR(3 downto 1);
      REGION:  OUT x01z_VECTOR(3 downto 0);
      LD:  INOUT x01z_VECTOR(31 downto 0));

    ATTRIBUTE PIN_NUMBERS OF logic : ENTITY IS
      ”LD(0):2 LD(1):3 LD(2):4 LD(3):5 LD(4):6 LD(5):7 LD(6):8 LD(7):9”
      &”LD(8):11 LD(9):12 LD(10):13 LD(11):14 LD(12):15 LD(13):16 LD(14):17 LD(15):18”
      &”LD(16):23 LD(17):24 LD(18):25 LD(19):26 LD(20):27 LD(21):28 LD(22):29 LD(23):30”
      &”LD(24):32 LD(25):33 LD(26):34 LD(27):35 LD(28):36 LD(29):37 LD(30):38 LD(31):39”
      &”LA(8):159 LA(9):158 LA(10):157 LA(11):156 LA(12):155 LA(13):154 LA(14):153
       LA(15):152”
      &”LA(16):150 LA(17):149 LA(18):148 LA(19):147 LA(20):146 LA(21):145 LA(22):144
       LA(23):143”
      &”LA(24):138 LA(25):137 LA(26):136 LA(27):135 LA(28):134 LA(29):133 LA(30):132
       LA(31):131”
      &”AM(0):42 AM(1):43 AM(2):44 AM(3):45 AM(4):46 AM(5):47”
      &”VCOMP(3):122 VCOMP(2):123 VCOMP(1):124”
      &”REGION(0):113”
      &”REGION(1):51 REGION(2):58 REGION(3):53”
      &”LIRQ1:85 LIRQ2:84 LIRQ3:83 LIRQ4:82”
      &”DENIN:119 DENIN1:118 MOD_DENIN:117 MOD_DENIN1:116 LA_UP_ADDR:115 EN_UP_ADDR:114”
      &”D64:139 LDS:129 PREN:72 LEDI:128 LDEN:127 MWB:126 SELECTLM:125”
      &”LIRQ:75 DRAM_IO:98 CE0:89 CE1:88 CE2:87 CE3:86 DBE0:94 DBE1:93 DBE2:92 DBE3:91,
       CS0:19”
      &”RW:77 OE:78  SYSRESET:66 RESET:67 SVIC_RESET:68 CAS0:97 CAS1:96 CAS2:95 CAS3:112”;

END logic;
```

## Appendix A. VDHL Code (continued)

```
ARCHITECTURE arch_logic OF logic IS

    signal VL1N18: bit;
    signal VL1N26: bit;
    signal VL1N28: bit;
    signal VL1N31: bit;
    signal VL1N36: bit;
    signal VL1N40: bit;
    signal STATUS_ID: BIT_VECTOR(31 downto 0) := X"FFFFFFFF";
    signal STATUS_EN: BIT  := '0';
    signal REGION_TEMP:  BIT;

--     for all: AND2 use entity work.AND2(archAND2);

--     for all: INV use entity work.INV(archINV);

--     for all: AND3 use entity work.AND3(archAND3);

--     for all: OR2 use entity work.OR2(archOR2);

begin
-------------------------------------------------------------------------------------------
-- This is the logic for SELECTLM (when writing the REMOTE MASTER -- registers)
-------------------------------------------------------------------------------------------
SELECTLM <= '0' WHEN ((FXB(LA(31)) = '1') AND (FXB(LA(30)) = '1') AND (FXB(LA(29)) = '0') AND
(FXB(LA(28)) = '0')) ELSE '1';

-------------------------------------------------------------------------------------------
-- This is the logic for RESET
-------------------------------------------------------------------------------------------
SVIC_RESET <= RESET AND SYSRESET;

-------------------------------------------------------------------------------------------
-- This is the logic for driving the CASi inputs to DRAM
-- CASi is driven both during DRAM refresh and data access but not during I/O access
-------------------------------------------------------------------------------------------
CAS0 <= DBE3 OR (CS1 AND (NOT CS0));
CAS1 <= DBE2 OR (CS1 AND (NOT CS0));
CAS2 <= DBE1 OR (CS1 AND (NOT CS0));
CAS3 <= DBE0 OR (CS1 AND (NOT CS0));

-------------------------------------------------------------------------------------------
-- This is the logic for the latch and enable signals for A40/A64 UPPER

-- ADDRESS
-------------------------------------------------------------------------------------------
LA_UP_ADDR <= (NOT SELECTLM) AND LEDI;

EN_UP_ADDR <= LDEN OR MWB;
```

## Appendix A. VDHL Code (continued)

```
-------------------------------------------------------------------------------------------
-- This is the logic for controlling the CE*, OE* signals to each bank of DRAM
-- SRAM in I/O space
-------------------------------------------------------------------------------------------
CE0 <=  DBE3 OR CS0;
CE1 <=  DBE2 OR CS0;
CE2 <=  DBE1 OR CS0;
CE3 <=  DBE0 OR CS0;

OE <=   NOT RW;

-------------------------------------------------------------------------------------------
-- This is the cross-connected SWAP BUFFER logic required for MD32 and D64
-- on same board
-------------------------------------------------------------------------------------------
VL1I1: AND2
        port map(A => D64,
                 B => VL1N31,
                 Q => VL1N28);

    VL1I11: INV
        port map(A => DENIN,
                 QN => VL1N18);

VL1I2: AND3
        port map(A => DENIN1,
                 B => VL1N18,
                 C => D64,
                 Q => VL1N26);

    VL1I3: OR2
        port map(A => VL1N26,
                 B => VL1N28,
                 Q => VL1N31);

    VL1I33: AND2
        port map(A => VL1N36,
                 B => VL1N31,
                 Q => VL1N40);

    VL1I38: OR2
        port map(A => DENIN1,
                 B => VL1N40,
                 Q => MOD_DENIN);

    VL1I39: OR2
        port map(A => VL1N40,
                 B => DENIN,
                 Q => MOD_DENIN1);

    VL1I4: INV
        port map(A => LDS,
                 QN => VL1N36);
```

## Appendix A. VDHL Code (continued)

```
-------------------------------------------------------------------------------------------
-- This is the REGION DECODER
-------------------------------------------------------------------------------------------
region:   PROCESS
    BEGIN
    CASE AM is
      WHEN ”000100” | ”000011” | ”000001” | ”000000” =>  --A64 AM Codes
        IF LD(31 downto 28) = ”1110” THEN
          REGION(2 downto 0) <= LD(26 downto 24);
          REGION_TEMP <= FXB(LD(27));
        ELSE REGION(2 downto 0) <= ”000”;
           REGION_TEMP <= ’0’;
        END IF;

      WHEN ”110100” | ”110101” | ”110111” =>   --A40 AM Codes
        IF LD(15 downto 12) = ”1110” THEN
          REGION(2 downto 0) <= LD(10 downto 8);
          REGION_TEMP <= FXB(LD(11));
        ELSE    REGION(2 downto 0) <= ”000”;
          REGION_TEMP <= ’0’;

        END IF;


      WHEN ”001000” | ”001001” | ”001010” | ”001011” | ”001100” | ”001101” | ”001110” |
        ”001111” =>  --A32 AM Codes
        IF VCOMP(3) = ’0’ THEN
          REGION(2 downto 0) <= LA(26 downto 24);
          REGION_TEMP <= FXB(LA(27));
        ELSE    REGION(2 downto 0) <= ”000”;
          REGION_TEMP <= ’0’;
        END IF;


      WHEN ”101111” | ”110010” | ”111000” | ”111001” | ”111010” | ”111011” | ”111100” |
        ”111101” | ”111110” | ”111111” =>  --A24 AM Codes
        IF VCOMP(2) = ’0’ THEN
          REGION(2 downto 0) <= LA(18 downto 16);
          REGION_TEMP <= FXB(LA(19));
        ELSE    REGION(2 downto 0) <= ”000”;
          REGION_TEMP <= ’0’;
        END IF;


      WHEN ”101001” | ”101100” | ”101101” =>  --A16 AM Codes
        IF VCOMP(1) = ’0’ THEN
          REGION(2 downto 0) <= LA(10 downto 8);
          REGION_TEMP <= FXB(LA(11));
        ELSE    REGION(2 downto 0) <= ”000”;
          REGION_TEMP <= ’0’;
        END IF;
```

## Appendix A. VDHL Code (continued)

```
      WHEN ”011000” | ”011001” | ”011010” | ”011011” | ”011100” | ”011101” | ”011110” |
        ”011111” =>  --USER1 AM Codes
        IF VCOMP(3) = ’0’ THEN   --A32 Modes
          REGION(2 downto 0) <= ”101”;  --FORCED TO REGION 5
          REGION_TEMP <= ’0’;
        ELSE    REGION(2 downto 0) <= ”000”;
          REGION_TEMP <= ’0’;
        END IF;

      WHEN ”010000” | ”010001” | ”010010” | ”010011” | ”010100” | ”010101” | ”010110” |
        ”010111” =>  --USER2 AM Codes
        IF VCOMP(2) = ’0’ THEN
          REGION(2 downto 0) <= ”010”;  --FORCED TO REGION 10
          REGION_TEMP <= ’1’;
        ELSE    REGION(2 downto 0) <= ”000”;
          REGION_TEMP <= ’0’;
        END IF;

      WHEN OTHERS =>  --DEFAULT REGION
        REGION(2 downto 0) <= ”000”;
        REGION_TEMP <= ’0’;


    END CASE;
    END PROCESS;

    region_buffer: triout PORT MAP(REGION_TEMP, DRAM_IO, REGION(3));
--DON’T DRIVE REGION(3) WHEN IN DRAM MODE (DRAM_IO = 0)

-------------------------------------------------------------------------------------------
-- This is the INTERRUPT LOGIC
-------------------------------------------------------------------------------------------
    LIRQ <= (LIRQ1 AND LIRQ2) AND (LIRQ3 AND LIRQ4);
    STATUS_EN <= (NOT LDEN) AND MWB;

b1:    FOR i IN 0 TO 31 GENERATE
        bx: triout PORT MAP(STATUS_ID(i), STATUS_EN, LD(i));
       END GENERATE;

interrupt:PROCESS
    BEGIN
    IF LDEN = ’0’ THEN
     IF (LIRQ1 = ’0’ AND PREN =’1’) THEN STATUS_ID(7 downto 0) <= X”01”;
     ELSIF (LIRQ2 = ’0’ AND PREN = ’1’) THEN STATUS_ID(7 downto 0) <= X”02”;
     ELSIF (LIRQ3 = ’0’ AND PREN = ’1’) THEN STATUS_ID(7 downto 0) <= X”03”;
     ELSIF (LIRQ4 = ’0’ AND PREN = ’1’) THEN STATUS_ID(7 downto 0) <= X”04”;
     ELSIF (PREN = ’1’) THEN STATUS_ID(7 downto 0) <=”01010101”;
     ELSIF (PREN = ’1’ AND STROBE = ‘0’ AND LDS = ’1’) THEN STATUS_ID <= X”EEEEEE00”;--compare
     ELSIF (PREN = ’1’ AND STROBE = ‘0’ AND LDS = ’0’) THEN STATUS_ID <= X”0F0F0FFF”; --mask
     END IF;
    END IF;
    END PROCESS;

end arch_logic;
```