# Chapter 4
# Interrupts

Interrupts gain the attention of a processor. They are used for task switching, message passing and obtaining CPU time. For example, a peripheral (like a disk controller) can interrupt a processor when it needs attention. After the peripheral generates an interrupt, the processor saves its current state and jumps to a program called an interrupt service routine. There it services the peripheral and then returns to what it was doing just before the interrupt.

Interrupts are analogous to reading the newspaper at home on your day off. Several things can happen that could cause you to interrupt your reading. If the telephone rings you make a mental note of where you are reading, and then answer the phone. When you are done with the call, you return to reading the newspaper. In this case a peripheral (the telephone) needs attention, causing you to stop what you are doing (reading the newspaper). You then perform a service (answer the phone), and then return to what you were doing.

Multiple interrupt levels are available on VMEbus. These can be prioritized so that certain events get serviced before others.

For example, in our newspaper analogy, assume that the telephone and the doorbell ring at the same time. You may wish to answer the door before the telephone. In this case the doorbell has a higher priority than the telephone.

Microcomputers prioritize interrupts the same way. This also gives the ability to ignore (mask off) some interrupt sources, an important feature in some applications.

## 4.1 VMEbus Interrupts

VMEbus has a seven level prioritized interrupt architecture. The seven levels are called IRQ1* - IRQ7*, with IRQ7* having the highest priority. Modules that generate interrupts, such as serial I/O boards, use a functional module called an interrupter. Modules that service interrupts, such as CPUs, do so with a functional module called an interrupt handler.

To initiate an interrupt, the interrupter asserts one of the seven interrupt request lines. Each is wire 'or'ed together (open collector) and may be driven by any number of modules. Interrupt handlers monitor these signals and generate an interrupt acknowledge cycle in response to the requests. Each interrupt request line can be monitored by a single interrupt handler.

The interrupt acknowledge cycle performs two important functions. These include interrupt arbitration and a STATUS/ID (interrupt vector) read cycle. To generate an interrupt acknowledge cycle the handler first acquires the data transfer bus. This is necessary because it must obtain a STATUS/ID from the interrupter. All interrupt handlers must have a bus requester to acquire the data transfer bus. Interrupt acknowledge cycles that do not fetch a STATUS/ID are not allowed.

Once the bus has been acquired, the handler asserts address lines A01-A03, IACK* and AS*. The assertion of IACK* notifies all modules that the current cycle is an interrupt acknowledge cycle. A block diagram of the priority interrupt bus is shown in Figure 4-1.

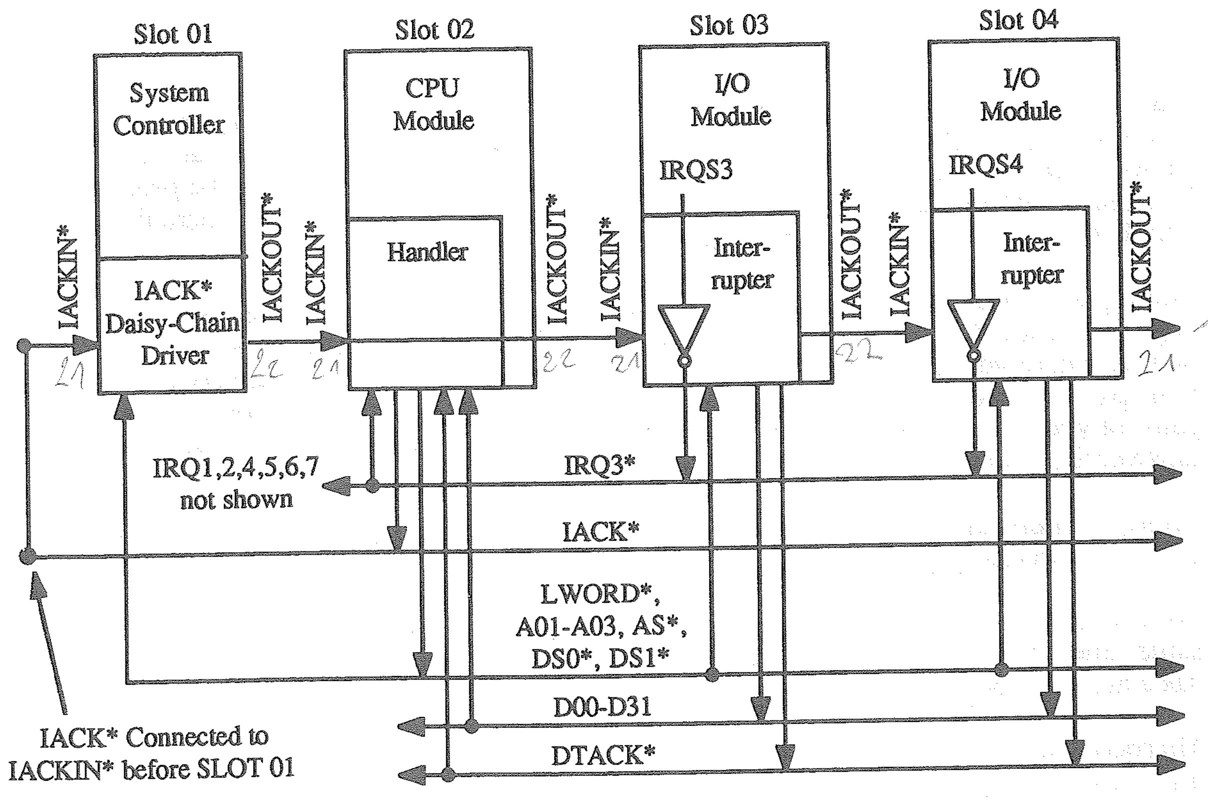


Figure 4-1. Block diagram of the priority interrupt bus.

The three bit address code A01-A03 notifies all interrupters of the priority level being acknowledged. Table 4-1 shows how they are driven.

Table 4-1. Three bit code (A01-A03) used during interrupt acknowledge cycle.

| Interrupt line being acknowledged | 3 bit code | | |
|---|---|---|---|
| | A03 | A02 | A01 |
| IRQ7* | 1 | 1 | 1 |
| IRQ6* | 1 | 1 | 0 |
| IRQ5* | 1 | 0 | 1 |
| IRQ4* | 1 | 0 | 0 |
| IRQ3* | 0 | 1 | 1 |
| IRQ2* | 0 | 1 | 0 |
| IRQ1* | 0 | 0 | 1 |

Address modifiers AM0-AM5 are not driven during the interrupt acknowledge cycle. Handlers must, however, assert IACK* during the cycle.

When AS* and DS0* (and/or DS1*) are asserted, the IACK* daisy-chain is driven by the slot 01 IACK* daisy-chain driver. The daisy-chain propagates from module to module until it reaches the interrupter that initially requested the interrupt. The requester then places an 8, 16 or 32 bit STATUS/ID (vector) onto data lines D00-D31, and terminates the cycle with DTACK*. The STATUS/ID can be used by the interrupt handler to determine which interrupter initiated the interrupt. This speeds up interrupts because the handler does not need to poll the interrupt sources to determine which one requires service.

CPU modules perform interrupt service routines in response to interrupt requests. The VMEbus specification does not define what must happen during the interrupt service routine; this is completely user defined. This interrupt service routine may or may not require use of the bus.

To illustrate an interrupt acknowledge cycle, consider again the block diagram of Figure 4-1. If the interrupter located in slot 04 generates an interrupt, it initiates an interrupt acknowledge cycle like the one shown in the timing diagram of Figure 4-2. Here the interrupt handler, located in slot 02, monitors and responds to interrupt requests on level IRQ3*. When it monitors that IRQ3* is asserted it arbitrates for the data transfer bus. Once it has obtained the bus it asserts IACK*, and places the level of interrupt it is acknowledging on A01-A03. The handler then asserts AS* and one (or both) data strobe.

Figure 4-2 shows a D08(O) interrupt acknowledge cycle. The levels of DS0* and DS1* indicate the width of STATUS/ID that the handler expects to receive from the requester. 8, 16 or 32 bit STATUS/ID words are allowed. Mnemonics describing the interrupter and handler types, and the selection of DS0* and DS1*, are shown in Table 4-2.

The VMEbus Revision B specification allowed only the D08(O) STATUS/ID. For this reason, interrupters may provide fewer bytes than requested by the handler. Un-driven data lines are pulled up to a logic '1' by the backplane termination networks.

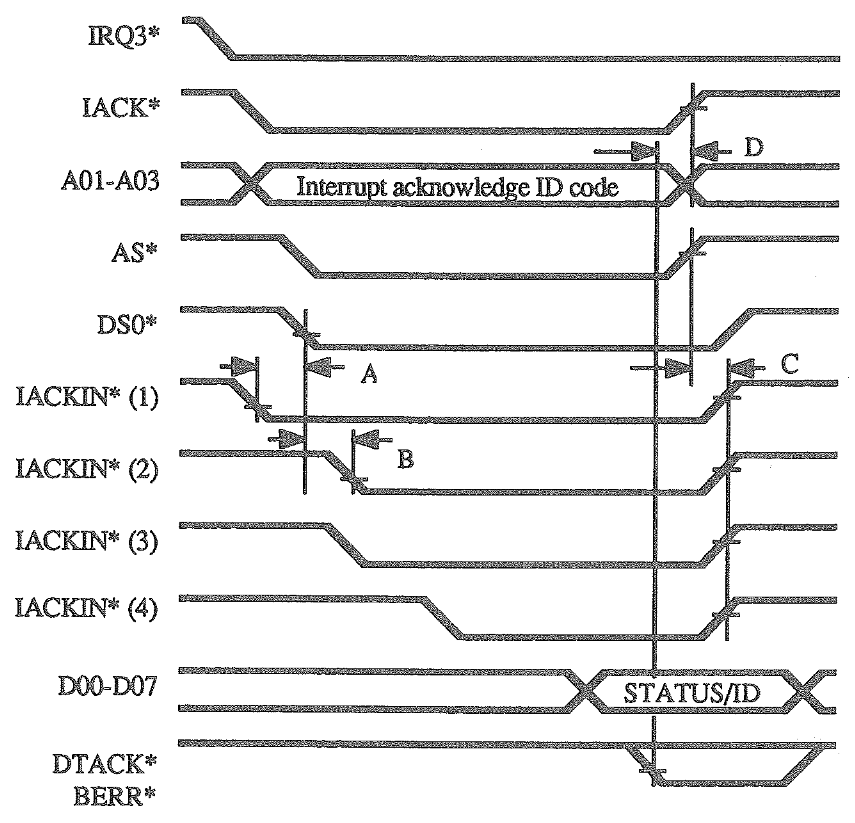


Figure 4-2. D08(O) interrupt acknowledge cycle.

Table 4-2. STATUS/ID options on interrupt acknowledge cycles.

| Mnemonic | When applied to a | Means that |
|---|---|---|
| D08(O) | Interrupter | Responds to 8, 16 and 32 bit interrupt acknowledge cycles. These modules place an 8 bit STATUS/ID onto data lines D00-D07. They must monitor DS0* but not DS1* or LWORD*. They must not drive D08-D31. |
| | Interrupt handler | Generates 8 bit interrupt acknowledge cycles and reads an 8 bit STATUS/ID on D00-D07. |
| D16 | Interrupter | Responds to 16 and 32 bit interrupt acknowledge cycles. These modules place a 16 bit STATUS/ID onto data lines D00-D15. They must monitor DS1* but do not drive data lines D16-D31. |
| | Interrupt handler | Generates a 16 bit interrupt acknowledge cycle and reads a 16 bit STATUS/ID on D00-D15. |
| D32 | Interrupter | Responds to 32 bit interrupt acknowledge cycles with a 32 bit STAUTS/ID on D00-D31. These modules must monitor DS0*, DS1* and LWORD*. |
| | Interrupt handler | Generates a 32 bit interrupt acknowledge cycle and reads a 32 bit STATUS/ID on data lines D00-D31. |

### 4.1.1 The IACK* Daisy-chain

The assertion of IACK* and one or both of the data strobe starts the IACKIN*/IACKOUT* daisy-chain. The daisy-chain is driven by the slot 01 IACK* daisy-chain driver. As shown in the timing diagram of Figure 4-2 (points A & B) it delays IACKOUT* until a data strobe has been asserted for some minimum time. Its purpose is to allow back-to-back interrupt acknowledge cycles, and to guarantee IACKIN* set-up times for interrupters.

The IACKIN*/IACKOUT* daisy-chain propagates from module to module until it reaches the correct interrupter. When the responding interrupter receives IACKIN*, it places the STATUS/ID onto the data bus, and terminates the cycle with DTACK*.

As Figure 4-2 shows (point C), IACKIN*/IACKOUT* from each VMEbus module must be negated within some specified time after AS* is negated, regardless of the state of DS0* or DS1*. This provision is made so that back-to-back interrupt acknowledges may be performed. If not met, this timing parameter (40 ns.) could cause IACKIN* to remain asserted until the start of another interrupt acknowledge cycle, and cause a system crash. When evaluating or designing VMEbus modules, make sure that IACKOUT* is negated within the minimum specified time after AS* is negated.

All commercial VMEbus backplanes provide IACK* daisy-chain jumpers like those shown for the bus grant daisy-chains (Chapter 3). These jumpers allow propagation of the daisy-chain if one or more slots in the backplane are empty. For example, consider a 21 slot VMEbus system with only two modules: a slot 15 interrupter and a slot 01 handler. During the interrupt acknowledge cycle there would be no way for the daisy-chain to propagate between slots 01 and 15 if the intermediate slots were empty, and the system would crash. To prevent this, IACKIN* and IACKOUT* at each empty slot should be shorted together with a backplane jumper. This allows the daisy-chain to propagate in a normal fashion.

It is not necessary to place jumpers after the last module in the system. In our example, jumpers do not have to be placed in slots 16-21.

Most vendors of VMEbus modules include a printed circuit trace between IACKIN* and IACKOUT* on all non-interrupter modules. This eliminates the need to install jumpers in the backplane when modules are installed. The VMEbus specification does not require this trace, however. If the vendor did not do this, the jumper must be installed if an interrupter resides farther down the system. If you are unsure if a module has this trace refer to the manufacturer's data sheet or use a ohm meter to determine if IACKIN* and IACKOUT* are shorted.

A daisy-chain jumper must not be installed in a slot where an interrupter resides. Since interrupters assert IACKOUT*, the jumper may crash the system.

### 4.1.2 Interrupt Acknowledge Cycle Timing

In general, the interrupt acknowledge cycle will follow the same timing rules as read/write cycles. This includes address rot as Figure 4-2 shows (point D). When DTACK* is asserted by the interrupter, the handler may (but does not have to) release AS*, IACK* and A01-A03. The interrupter, however, must continue to drive the valid STATUS/ID onto the data bus until the handler negates both data strobes. When evaluating or designing VMEbus interrupters verify that they do this.

## 4.2 Interrupter

The interrupter functional module generates interrupts to handlers. The levels that the interrupter uses are given by the I(x) and I(x-y) mnemonics.

There are two classes of interrupters: release-on-acknowledge and release-on-register-access. The mnemonics ROAK and RORA are used to describe them. The ROAK interrupter negates its interrupt request line in response to an interrupt acknowledge cycle. The interrupt release is shown in the timing diagram of Figure 4-3. The ROAK mechanism works with all handlers.

The RORA interrupter releases its request when the handler accesses an on-board register during the interrupt service routine. As shown in Figure 4-4, the handler performs the acknowledge cycle, but the interrupter does not immediately negate its request. Sometime during the service routine the handler writes to a register on the interrupter. This causes it to negate the request.

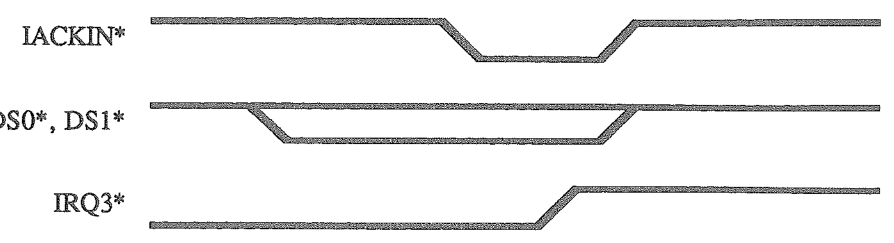


Figure 4-3. Timing for ROAK release mechanism. The interrupt request, in this case IRQ3*, is negated in response to the interrupt acknowledge cycle.

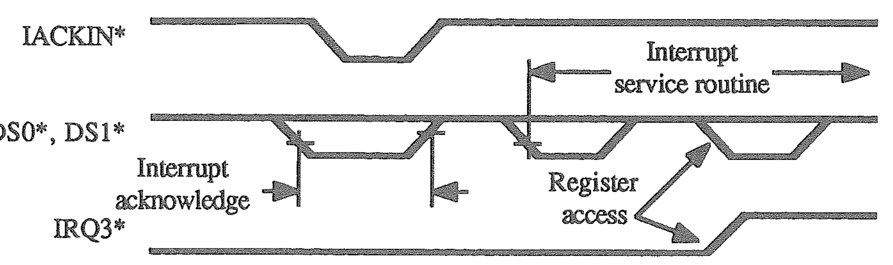


Figure 4-4. Timing for RORA release mechanism. The interrupt request, in this case IRQ3*, is negated when a register on the interrupter module is accessed by the handler.

The RORA release mechanism works with most CPU handlers because the CPU can access the interrupter's internal registers during the service routine. Handlers, however, are not required to have the capability to obtain the data transfer bus and do read/write cycles (they are only required to obtain the bus to read a STATUS/ID byte). If the handler cannot perform read/write cycles it is impossible to force the RORA interrupter to release its interrupt. In these cases an ROAK interrupter would have to be used.

The Revision B VMEbus specification allowed only the ROAK mechanism. This caused problems with some VLSI chips (such as serial and parallel I/O ports) and the RORA mechanism was added in Revision C. For example, the MC68681 serial port IC generates an interrupt from several sources. If two or more of these were to happen at the same time it would assert and hold its interrupt request until all were serviced. This type of request is called a level sensitive request because it is intended that a host CPU keep servicing interrupts as long as a valid request is present. Before the RORA interrupter the logic necessary to interface this device to the VMEbus was cumbersome (and often impossible) to build. A good rule of thumb is that the ROAK release mechanism should be used only with edge sensitive interrupt requests.

The mnemonics used to describe the interrupt release mechanisms are summarized in Table 4-3.

### 4.2.1 Circuit Example - Interrupter

Figure 4-5 shows a circuit for a simple ROAK interrupter. It is a clock generator which produces an interrupt 100 times each second. It can be used for a real time clock source, a task switching (heartbeat) timer or for any other timing purpose. The circuit can be easily adapted to handle other applications where edge-triggered interrupts are needed.

Table 4-3. Mnemonics that describe interrupt release capabilities.

| The Mnemonic | When Applied to | Means that |
|---|---|---|
| RORA | Interrupter | Releases its interrupt request line when some master accesses an on-board status or control register. |
| ROAK | Interrupter | Releases its interrupt request line when its STATUS/ID is read during the interrupt acknowledge cycle. |

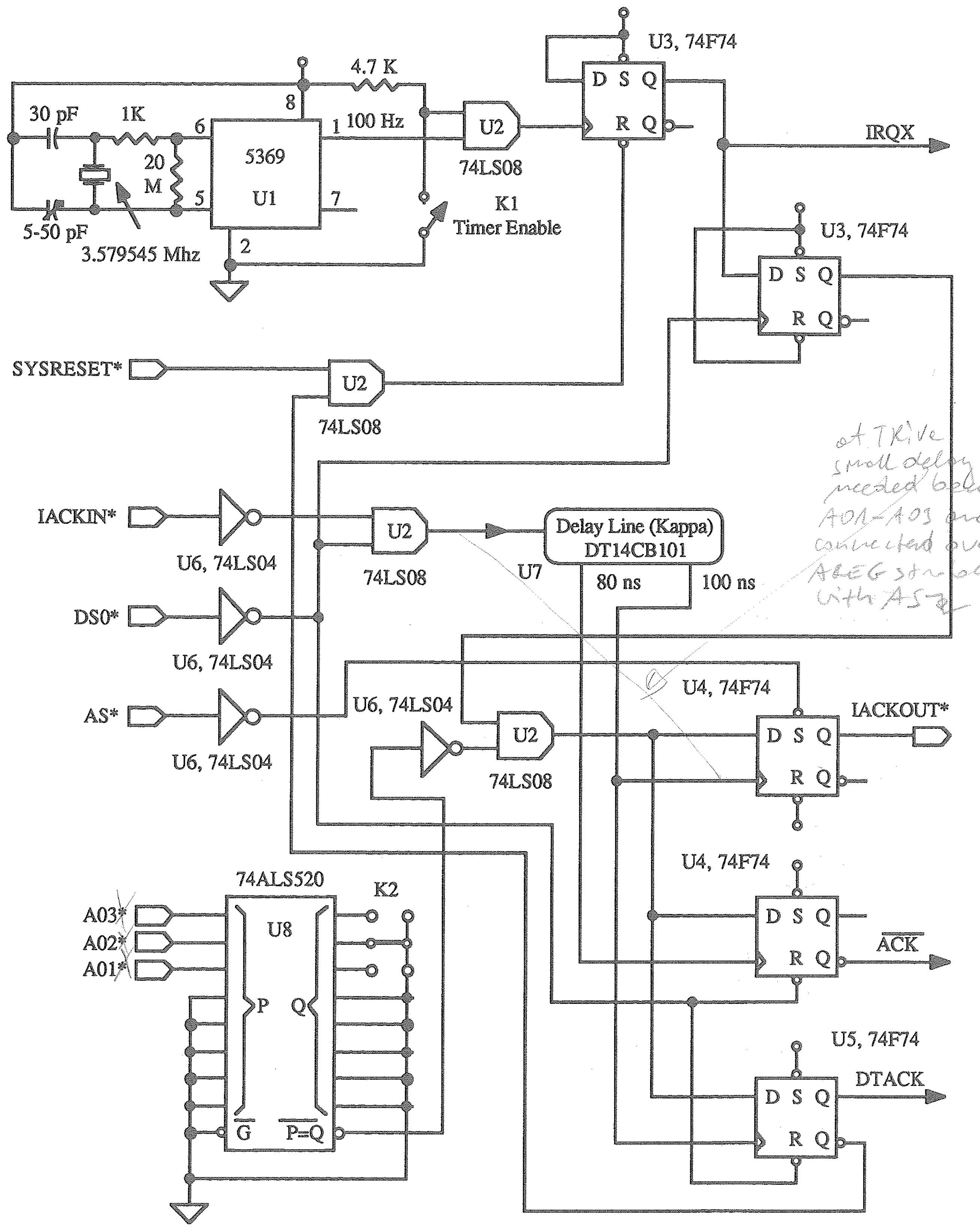
This circuit illustrates some of the key concepts of VMEbus interrupt generation. It shows how to initiate an interrupt, why the local interrupt must be latched during the interrupt acknowledge cycle, how to compare the acknowledge level, how to control timing and prevent race conditions, how to place a STATUS/ID byte onto the bus and when to negate DTACK* and IACKOUT*. The circuit also illustrates how the ROAK release mechanism works.

The 100 Hz time base is generated with a MM5369 timer U1 and a 3.57 Mhz crystal (television color burst crystal in the US and Canada). The square wave output of U1 is latched by flip-flop U3 when switch K1 is open. U3 drives any of the seven interrupt lines IRQ1* - IRQ7* depending upon the state of jumper block K3. Figure 4-5 is set for level IRQ5*. The interrupt lines are driven by an open-collector driver IC, U9. This is a 74F38 device that can sink up to 48 mA of current as required by the VMEbus specification.

After the interrupter generates the interrupt, the handler responds with an interrupt acknowledge cycle. The interrupt handler initiates this cycle by placing the acknowledging interrupt level on A01 - A03, and asserts IACK*, AS* and the data strobe(s) DS0*/DS1*. The IACKIN*/IACKOUT* daisy-chain then propagates from module to module until it reaches the interrupter. When IACKIN* is asserted, the interrupter decides whether to acknowledge the interrupt or pass the daisy-chain. The decision flow for this is shown in Figure 4-6.

The circuit of Figure 4-5 uses a 74ALS520 comparator (U8) and a 74LS08 AND gate (U2) to determine when a valid interrupt acknowledge is in progress. At the falling edge of DS0* (on every bus cycle) the state of the local interrupt request line IRQX is sampled using flip-flop U3. When IACKIN* and DS0* are both asserted, a rising edge propagates through delay line U7.

If this interrupter initiated an interrupt, and the handler is issuing an interrupt acknowledge cycle on the same interrupt level as was requested, the output of U2 is asserted (active high). Flip-flops U4 and U5 latch the state of U2 at the rising edges of the 80 and 100 nanosecond taps of U7. If the interrupter participates in the cycle, it places a STATUS/ID byte onto the data bus and asserts DTACK*. If it does not participate in the cycle, then IACKOUT* is asserted.



at Trive small delay needed because A01-A03 are connected over A&EG strobed with AS*

Figure 4-5. Simple ROAK interrupter circuit.

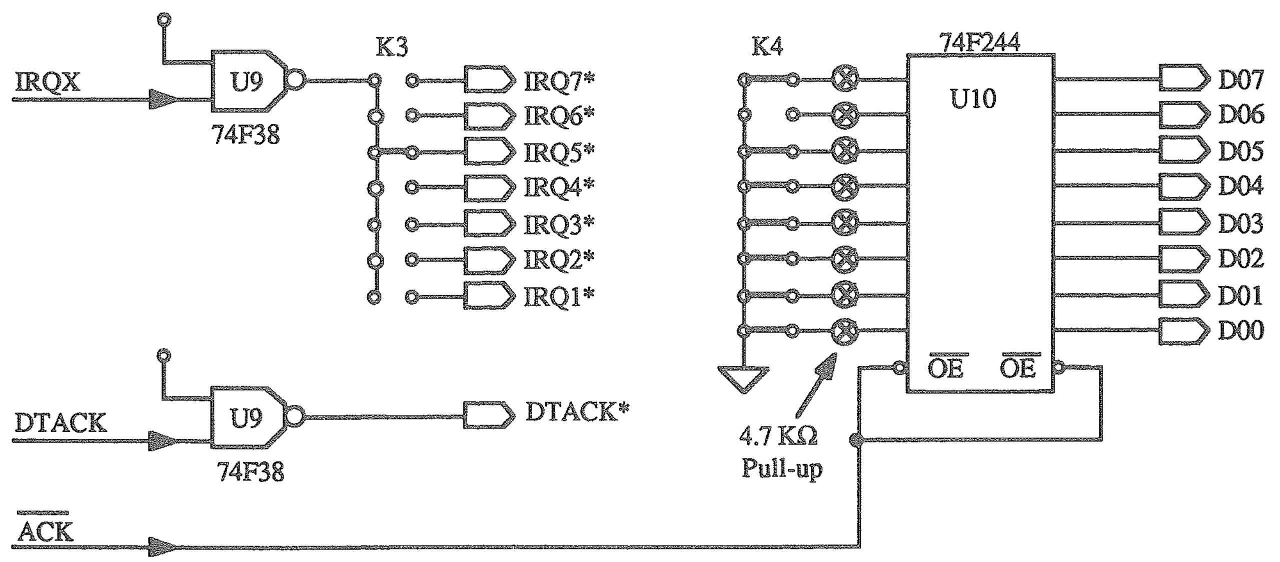


Figure 4-5 (con't).

A metastable state can appear on the output of U3 because it latches the state of the interrupt request asynchronously to the interrupt acknowledge cycle (the interrupt request could happen just as IACKIN* is driven in response to some other interrupter). The 100 nanosecond delay provided by U7 allows this metastable state to damp out before the module asserts IACKOUT* or DTACK*.

DTACK* and IACKOUT* are negated under different circumstances. At the end of the interrupt acknowledge cycle, DTACK* is negated after the handler negates DS0*. IACKOUT*, on the other hand, must be negated within 30 nanoseconds after AS* is negated. This permits back-to-back interrupt acknowledge cycles.

This circuit does not determine if it should participate in the interrupt acknowledge cycle until IACKIN* has been asserted. Faster versions could latch the interrupt request and determine whether the module participates in the cycle by monitoring IACK*. All metastable states could damp out before the IACKIN* daisy-chain reaches the module.

This module responds to all widths of STATUS/ID, but some modules return only 16 or 32 bit status IDs. If it were designed to return a D16 or D32 STATUS/ID, and the interrupt acknowledge cycle required a D08(O) STATUS/ID, then the module would not respond to the cycle and would pass the IACK* daisy-chain. The flowchart of Figure 4-6 shows the decision tree used by this interrupter.

When evaluating or designing modules with interrupters, verify that race conditions do not occur on the IACK* daisy-chain. Just as with requesters, interrupters must have an internal arbiter. In the case of the interrupter, the arbiter must determine whether to return a STATUS/ID or assert IACKOUT*. In Figure 4-5 this arbitration is provided by the delay line and flip-flops U4 and U5.

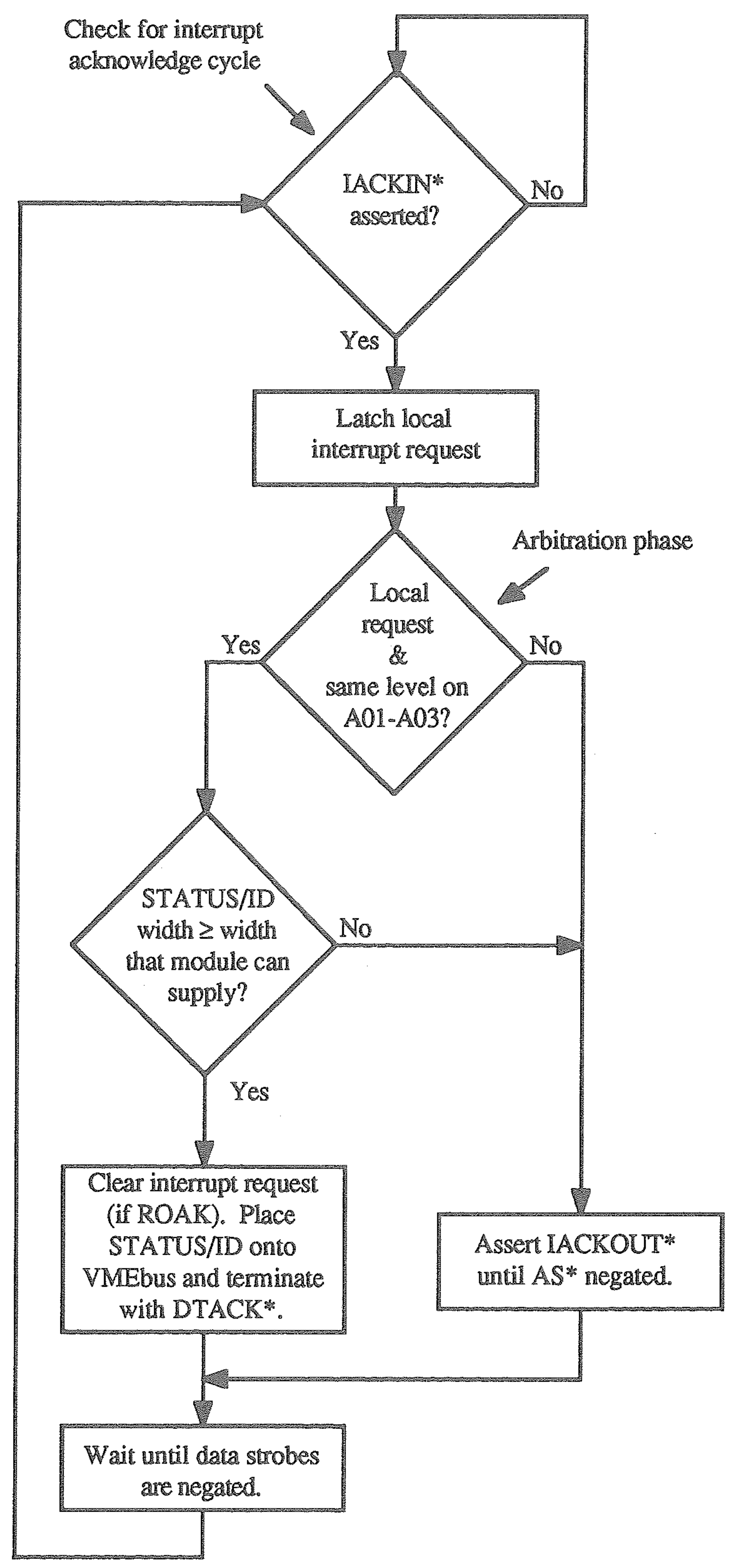

Check for interrupt acknowledge cycle
IACKIN* asserted?
No
Yes
Latch local interrupt request
Arbitration phase
Local request & same level on A01-A03?
Yes
No
STATUS/ID width ≥ width that module can supply?
No
Yes
Clear interrupt request (if ROAK). Place STATUS/ID onto VMEbus and terminate with DTACK*.
Assert IACKOUT* until AS* negated.
Wait until data strobes are negated.

Figure 4-6. Decision flow diagram for an interrupter.

## 4.2.2 Circuit Idea - The MC68153 Bus Interrupter Module

When a sophisticated interrupter is needed, the MC68153 Bus Interrupter Module (BIM) IC from Motorola can be used. This device is a four channel interrupter for VMEbus. The block diagram of Figure 4-7 shows it configured as a slave in the short I/O address space. VMEbus masters can program the device using the registers shown in Figure 4-8. This programmability eliminates the need for jumpers on the module.

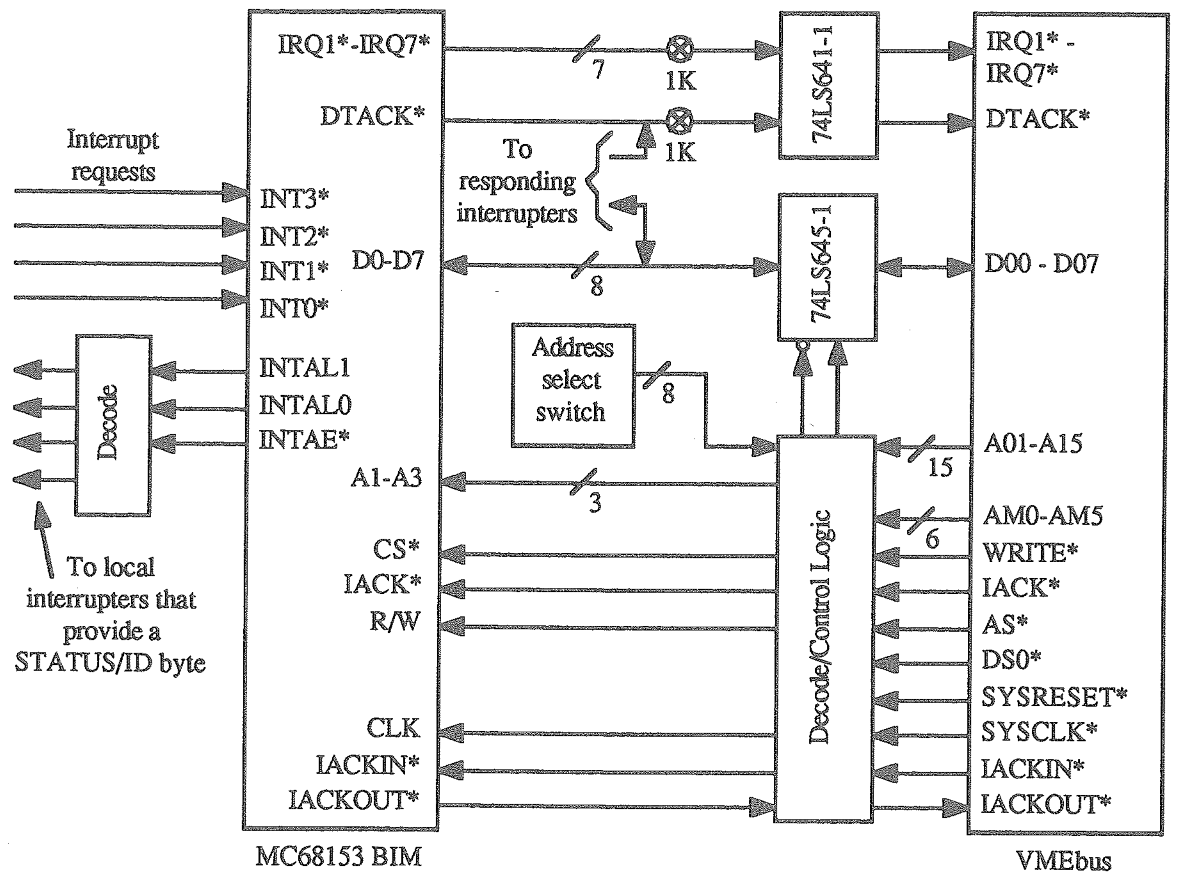


Figure 4-7. Block diagram of an interrupter using the MC68153 Bus Interrupter Module. The device is shown as a slave in the short I/O address space.

Interrupts are generated by asserting a local request on INT0*-INT3*. The MC68153 BIM monitors these and asserts a VMEbus interrupt request on IRQ1* - IRQ7*. The level at which this interrupt is generated depends upon how the control register (bits D0-D2) for that channel are set. Each interrupt may be disabled by clearing bit D4 of the associated control register.

When the MC68153 monitors an interrupt acknowledge cycle (IACK* and IACKIN* asserted) it responds by providing an 8 bit STATUS/ID or by passing the IACKOUT* daisy-chain to the next module. If it participates in the cycle it will respond in one of two ways. How it responds depends upon the state of the X/IN bit (D5) in the control register. If it is programmed for an external STATUS/ID it will place the device number (0-3) onto INTAL0-INTAL1 and assert INTAE*. This forces the interrupting peripheral to supply its own STATUS/ID byte. If pro-

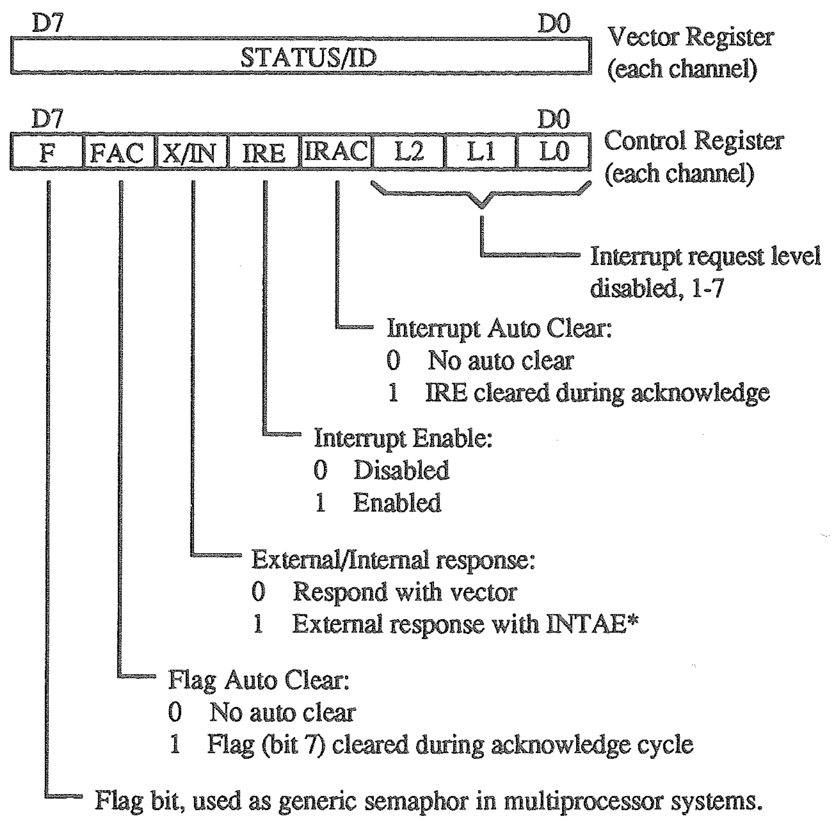
grammed for an internal STATUS/ID it will place the byte in the vector register onto the data bus.

The MC68153 automatically clears its interrupt request during the interrupt acknowledge cycle. It is classified as an ROAK interrupter.

The external decoder/control logic shown in Figure 4-7 must guarantee some bus timing for the MC68153. For example, since the MC68153 has no AS* input, IACK* must be qualified in the external logic. This external logic must guarantee that IACKIN*-IACKOUT* timing is correct. If the device is to reside in the slot 01 backplane position an external IACK* daisy-chain driver must also be added.

MC68153 Register Set

| Address | | | Description |
|---|---|---|---|
| A3 | A2 | A1 | |
| 0 | 0 | 0 | Channel 0 Control |
| 0 | 0 | 1 | Channel 1 Control |
| 0 | 1 | 0 | Channel 2 Control |
| 0 | 1 | 1 | Channel 3 Control |
| 1 | 0 | 0 | Channel 0 Vector |
| 1 | 0 | 1 | Channel 1 Vector |
| 1 | 1 | 0 | Channel 2 Vector |
| 1 | 1 | 1 | Channel 3 Vector |



D7
D0
STATUS/ID
Vector Register (each channel)
D7
D0
F
FAC
X/IN
IRE
IRAC
L2
L1
L0
Control Register (each channel)
Interrupt request level disabled, 1-7
Interrupt Auto Clear:
0 No auto clear
1 IRE cleared during acknowledge
Interrupt Enable:
0 Disabled
1 Enabled
External/Internal response:
0 Respond with vector
1 External response with INTAE*
Flag Auto Clear:
0 No auto clear
1 Flag (bit 7) cleared during acknowledge cycle
Flag bit, used as generic semaphor in multiprocessor systems.

Figure 4-8. MC68153 Programmer's model.

### 4.2.3 Circuit Idea - The SCB68154 Interrupt Generator

The SCB68154 interrupt generator IC allows local masters to initiate VMEbus interrupts. As shown in the block diagram of Figure 4-9 a local master, such as a 68000, communicates with the device over a private data bus. The local master initiates interrupts on any level by writing to the SCB68154 internal control registers. This device was not intended to be used with slave modules since it does not have interrupt request inputs. It is useful for communication and message passing between processors.

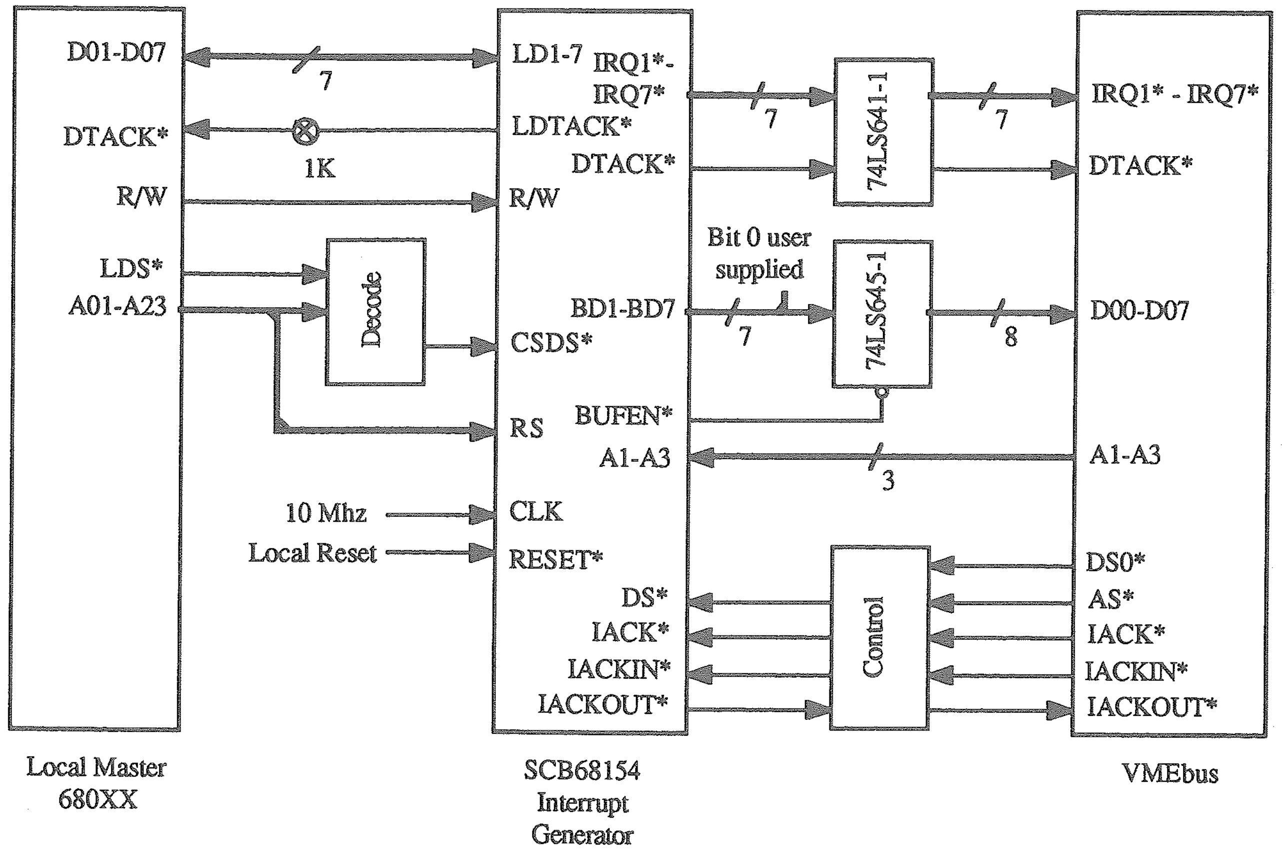
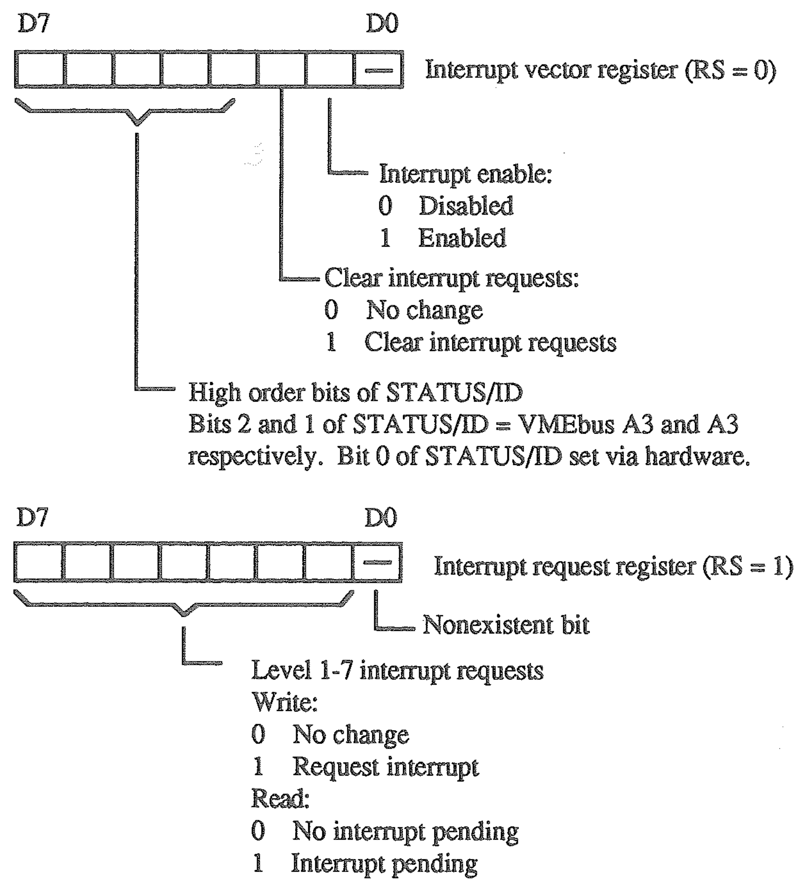


Figure 4-9. Block diagram of the SCB68154 interrupt generator IC.

Before generating an interrupt the local master sets up the interrupt vector register shown in the programmer's model of Figure 4-10. This register is used both to latch the STATUS/ID that will be sent during the interrupt acknowledge cycle and to enable interrupts. Only the highest five bits (D3 - D7) of the STATUS/ID are latched into the register. Bits D2 and D1 are set to the level of address lines A02 and A01 during the interrupt acknowledge cycle. This means that the STATUS/ID byte will vary depending upon what level the interrupt is generated. Bit D0 is set by hardware external to the SCB68154.

Once the interrupt vector register is set an interrupt may be triggered by writing to the appropriate bit in the interrupt request register. When an interrupt has been initiated, the local master should not change the interrupt vector register as this could cause the STATUS/ID to change during the acknowledge cycle. The master can check the status of an interrupt by reading interrupt request register.



D7
D0
Interrupt vector register (RS = 0)
Interrupt enable:
0 Disabled
1 Enabled
Clear interrupt requests:
0 No change
1 Clear interrupt requests
High order bits of STATUS/ID
Bits 2 and 1 of STATUS/ID = VMEbus A3 and A3 respectively. Bit 0 of STATUS/ID set via hardware.
D7
D0
Interrupt request register (RS = 1)
Nonexistent bit
Level 1-7 interrupt requests
Write:
0 No change
1 Request interrupt
Read:
0 No interrupt pending
1 Interrupt pending

Figure 4-10. Programmer's model for the SCB68154 Interrupt Generator IC.

The SCB68154 automatically clears its interrupt request during the interrupt acknowledge cycle. It is classified as an ROAK interrupter.

When designing circuits using the SCB68154, care should be taken to insure proper timing of the IACK* input and the IACKOUT* daisy-chain driver. As with the MC68153, IACK* must be qualified with AS*. IACKOUT* must also be negated less than 30 nanoseconds after AS* has been removed. The SCB68154 was designed to negate IACKOUT* only after DS0* is negated. The control block in Figure 4-9 should incorporate this fix. In addition, the circuit does not provide the IACK* daisy-chain driver.

## 4.3 IACK* Daisy-chain Driver

The IACK* daisy-chain driver was first introduced in the Revision C VMEbus specification and permits back-to-back interrupt acknowledge cycles. It is located in slot 01 and is part of the system controller. Under the Revision B specification the handler was not required to negate IACK* between consecutive interrupt acknowledge cycles, and the interrupt acknowledge daisy-chain would not always get negated between cycles (IACK* is connected to IACKIN* before slot 01 of the backplane). The net result could be a crashed system. This timing is shown in Figure 4-11.

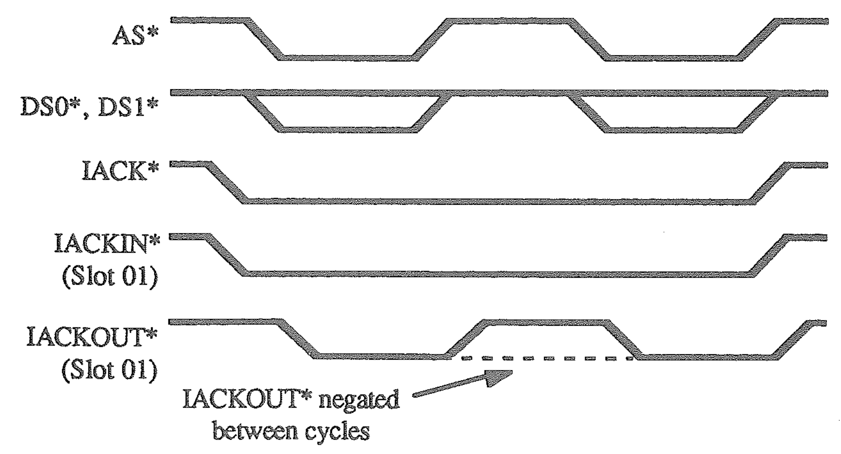


Figure 4-11. Timing diagram showing back-to-back interrupt acknowledge cycle. Without an IACK* daisy-chain driver IACKIN*/IACKOUT* would not be negated between cycles and could crash the system.

The IACK* daisy-chain driver must be located in the Slot 01 backplane position. When a VMEbus board maker claims to have a system controller on a module, it should have an IACK* daisy-chain driver circuit.

The IACK* daisy-chain driver also guarantees that IACKIN*/IACKOUT* is negated for a minimum time between cycles, and that IACKIN*/IACKOUT* will not be asserted until after the data strobes have been asserted. This simplifies the design of interrupters.

Most interrupt handlers, including MC680XX based CPU modules, negate IACK* between two interrupt acknowledge cycles, and therefore do not require the IACK* daisy-chain driver. It is necessary, however, that it be used in all systems since most manufacturers do not specify whether or not their modules produce back-to-back acknowledge cycles. As more VMEbus modules are developed they may take advantage of this cycle to speed up multiple interrupt handling.

## 4.4 Circuit Example - IACK* Daisy-chain Driver

A simple circuit for an IACK* daisy-chain driver is shown in Figure 4-12. 40 nanoseconds after either data strobe (DS0* and/or DS1*) is asserted, flip-flop U2 latches the state of IACKIN*. If the cycle is an interrupt acknowledge cycle then IACKOUT* is asserted until AS* is negated. If it is not an interrupt acknowledge cycle, then IACKOUT* is not asserted.

If the circuit resides on a module located in slot 01, then S1 is configured as shown in Figure 4-12. If in any other slot, it should be configured to pass the IACKIN*/IACKOUT* daisy-chain.

The circuit of Figure 4-12 is part of the simple system controller circuit of Figure 3-3.

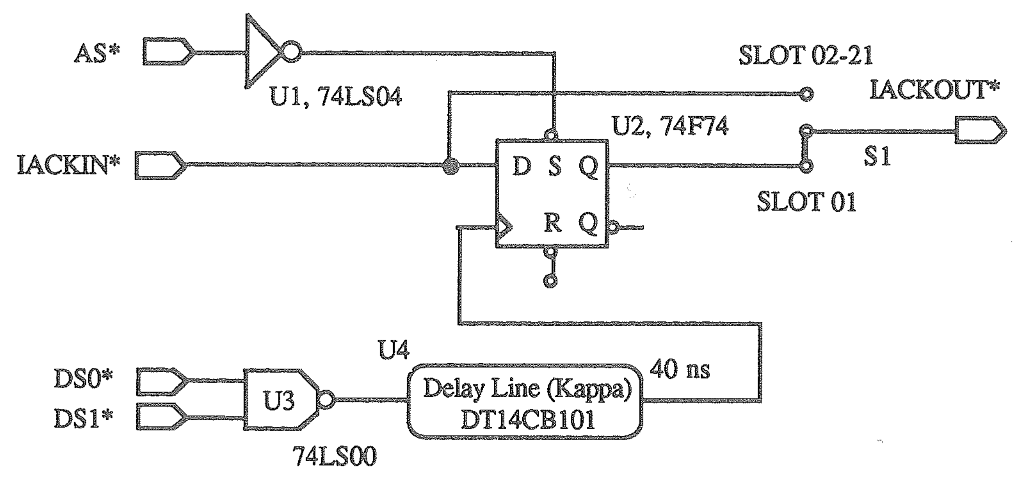


Figure 4-12. Simple circuit for an IACK* daisy-chain driver.

## 4.5 Handler

Interrupt handlers can be designed to accept interrupts on levels one to seven. While any number of interrupters may reside on any level, only one interrupt handler may monitor each level.

Table 4-4 shows the mnemonics describing the handler options. Interrupt handlers with the mnemonic IH(x) can monitor a single level of interrupt. For example, a module that monitors level 5 would be called an IH(5) handler. When more than one level is handled the IH(x-y) mnemonic is used, where x and y are a range of interrupts. For example, a handler which monitors levels 2, 3, 4 and 5 is called an IH(2-5) handler.

Table 4-4. Interrupt handler options and their mnemonics.

| Mnemonic | Description |
|---|---|
| IH(x) | Monitors and generates interrupt acknowledge cycles in response to interrupt requests on level IRQx. |
| IH(x-y) | Monitors and generates interrupt acknowledge cycles in response to interrupt requests on levels IRQx to IRQy. |

By strict interpretation of the VMEbus specification, a IH(x-y) handler can only monitor consecutive interrupt request lines. For example, it may monitor levels 2, 3, 4 and 5 but may not levels 2, 3, 5 and 6. In most cases it can be done, however. The only reason for this requirement is so the IH(x-y) mnemonic will apply in all situations.

### 4.5.1 Circuit Example - Handler for a MC680XX CPU Module

An example of a VMEbus interrupt handler is shown in the block diagram of Figure 4-13. Interrupts IRQ1* - IRQ7* are encoded to a three bit code (IPL0* - IPL2*) for use by the

MC680XX microprocessor. When the MC680XX processor monitors a pending interrupt on these inputs, it initiates an interrupt acknowledge cycle.

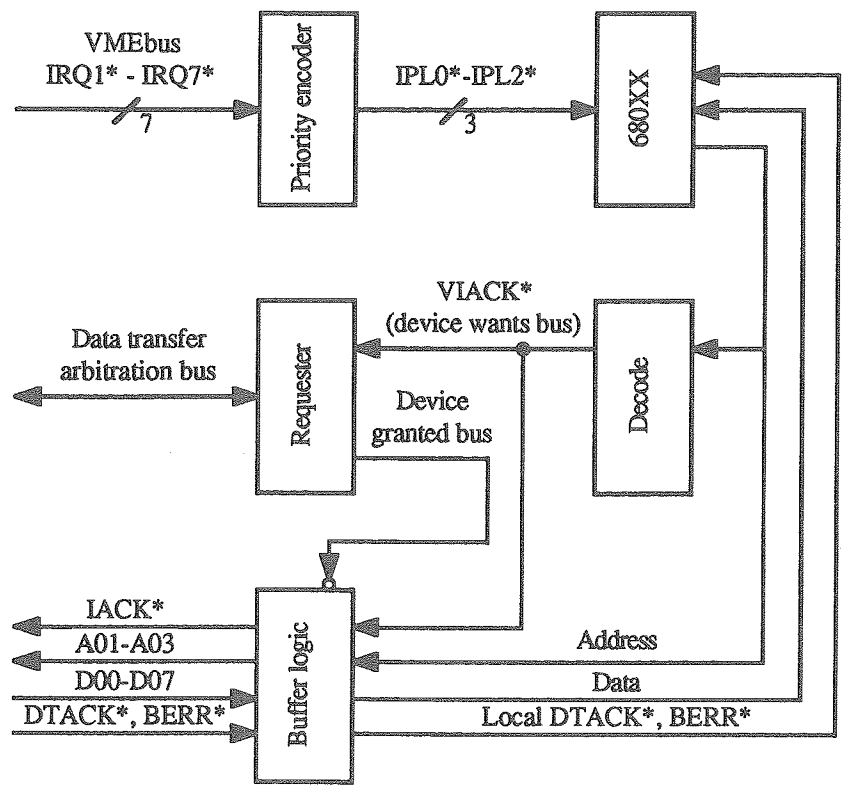


Figure 4-13. Block diagram of an interrupt handler on an MC680XX based CPU module. Portions have been omitted for clarity.

During the interrupt acknowledge cycle VIACK* is asserted by a decoder. After VIACK* is asserted, the local MC680XX processor expects to fetch a STATUS/ID byte from VMEbus. Before the byte can be fetched, however, VMEbus mastership must first be obtained using the bus requester. In most cases VIACK* is 'or'ed with off-board memory access requests. For simplicity this logic is not shown.

Once VMEbus ownership is obtained, an interrupt acknowledge cycle is generated. During the interrupt acknowledge cycle the local master asserts IACK*, sets A01-A03 to the level that is being acknowledged, and asserts AS* and DS0*. The interrupter then places the STATUS/ID byte onto the bus and asserts DTACK*. The local processor reads the byte and terminates the cycle.

When evaluating or designing interrupt handlers be careful with the 7 to 3 priority encoder between IRQ1* - IRQ7* and IPL0* - IPL2*. A proven solution is the circuit of Figure 4-14. Here a PAL16L8B is programmed as shown in the truth table of Table 4-5 and the equations of Figure 4-15. This circuit has a jumper block which allows programming of the interrupt levels to be handled. The example is set to accept interrupts on levels IRQ1*-IRQ5*, and to ignore levels IRQ6* and IRQ7*. A common mistake is to design the jumper block with a pull-up resistor as shown in the inset of Figure 4-14. The pull-up resistor violates the maximum input

low current allowed for IRQ1* - IRQ7*. Chapter 6 describes the electrical characteristics of the bus interface in more detail.

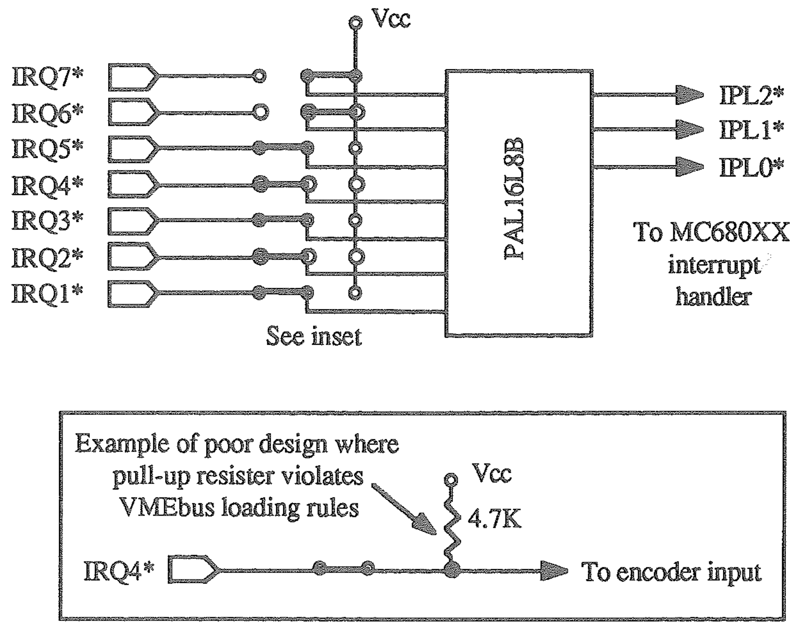


Figure 4-14. Simple VMEbus priority encoder used with MC680XX handlers.

Table 4-5. Truth table for PLD shown in Figure 4-14.

| Truth Table | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| IRQ7* | IRQ6* | IRQ5* | IRQ4* | IRQ3* | IRQ2* | IRQ1* | IPL2* | IPL1* | IPL0* |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | X | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | X | X | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | X | X | X | 0 | 1 | 1 |
| 1 | 1 | 0 | X | X | X | X | 0 | 1 | 0 |
| 1 | 0 | X | X | X | X | X | 0 | 0 | 1 |
| 0 | X | X | X | X | X | X | 0 | 0 | 0 |

A second problem has become more prevalent as faster microprocessors come onto the market. This problem is a result of the sampling mechanism used by the MC680XX interrupt inputs. Consider the timing waveforms for the MC68020 IPL0* - IPL2* inputs shown in Figure 4-16(a). Since the IPL0* - IPL2* inputs are completely asynchronous to the microprocessor clock, they must be synchronized before its internal logic can use them. The MC68020 does this by clocking the inputs on two consecutive negative clock edges. In addition to synchronizing the inputs, the microprocessor also verifies that the same interrupt level is pending on two consecutive clocks. This is because priority encoders of the type shown in Figure 4-14 take time for their outputs to settle. This settling time is shown in the timing waveform of Figure 4-16(a). Here two interrupt requests occur on levels IRQ2* and IRQ5*. The first negative clock latches the IRQ2* input, and the second IRQ5*. Since they were not the same, the MC68020 rejects the request until the next negative clock edge. In general this clocking mechanism works quite well. It can cause problems, however, if a slow priority encoder is used.

```
TITLE       INTERRUPT ENCODER
PATTERN     VH0005.PDS
REVISION    A
AUTHOR      WADE PETERSON
COMPANY     (C) 1988 WADE PETERSON, ALL RIGHTS RESERVED
DATE        MARCH 27, 1988

CHIP ENCODE PAL16L8

IRQ7 IRQ6 IRQ5 IRQ4 IRQ3 IRQ2 IRQ1 NC NC GND
OE IPL2 IPL1 IPL0 NC NC NC NC NC VCC

EQUATIONS

/IPL2        = /IRQ7
       +  IRQ7 */IRQ6
       +  IRQ7 * IRQ6 */IRQ5
       +  IRQ7 * IRQ6 * IRQ5 */IRQ4

/IPL1 = /IRQ7
       +  IRQ7 */IRQ6
       +  IRQ7 * IRQ6 * IRQ5 * IRQ4 */IRQ3
       +  IRQ7 * IRQ6 * IRQ5 * IRQ4 * IRQ3 */IRQ2

/IPL0 = /IRQ7
       +  IRQ7 * IRQ6 */IRQ5
       +  IRQ7 * IRQ6 * IRQ5 * IRQ4 */IRQ3
       +  IRQ7 * IRQ6 * IRQ5 * IRQ4 * IRQ3 * IRQ2 */IRQ1
```

Figure 4-15. Logic equations for the PLD shown in Figure 4-14.

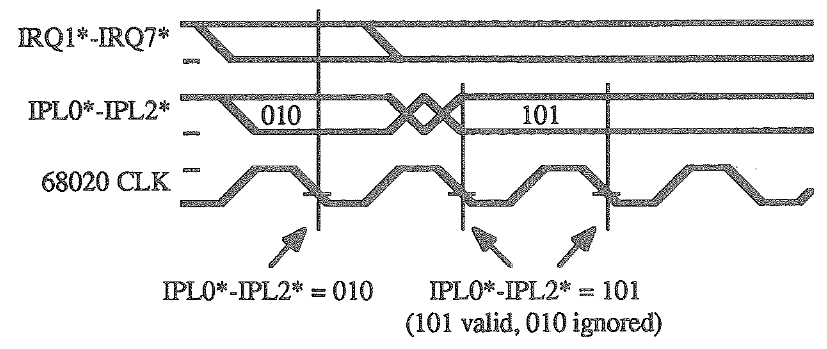
When an encoder such as that shown in Figure 4-14 changes states it can go through intermediate transient states. In our example where IRQ5* is asserted just after IRQ2*, IPL0* - IPL2* can occupy the IRQ3* state shown in Figure 4-16(b). If the encoder were slow it could occupy this intermediate state for some time and trigger a level three interrupt as shown in Figure 4-16(c). The result would be a random system crash (probably rare) when a level three interrupt acknowledge cycle occurs.

The fix for this potential problem is to use a fast encoder. If we were to use a 25 Mhz MC68020 this means that we would need an encoder that would change states within 40 nanoseconds (1/25 Mhz = 40 ns). If this requirement is met we should have no problems with our circuit.

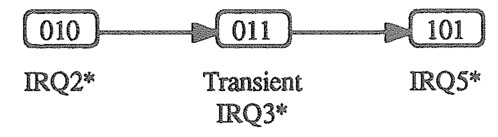
While the MC68020 clocks the IPL0* - IPL2* inputs on every negative edge, the MC68000 does so on a positive and negative edge. Since this represents twice the sampling frequency of

the MC68020 a 12.5 Mhz MC68000 would also need a 40 nanosecond encoder. The situation is exacerbated if the microprocessor clock duty cycle degrades.

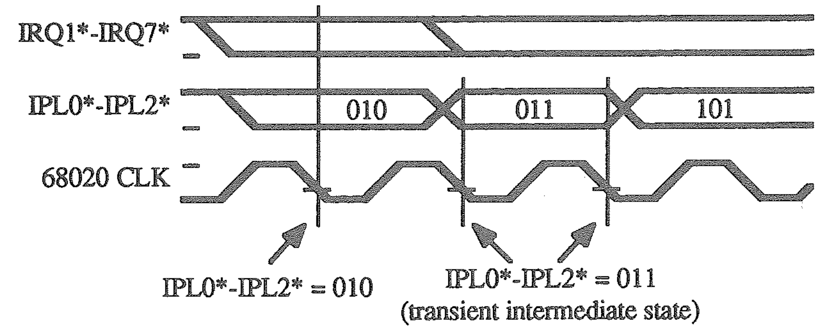
When evaluating or designing VMEbus interrupt handlers, be sure to check them for the wire-or glitch problem. The wire-or glitch was described in Chapter 3, and comes from using open-collector (wire-or) logic. MC680XX microprocessors guard against this by double clocking their inputs. When using other microprocessors on VMEbus they should be evaluated for compatibility with wire-or interrupt logic.



(a) Normal IPL0*-IPL2* transitions generate valid interrupt.



(b) One scenario of IPL0*-IPL2* transition between 010 and 101 generates unwanted level 3 interrupt request.



(c) A slow encoder with a fast MC68020 may cause a system crash due to transient interrupt request.

Figure 4-16. Poor choice of priority encoder may cause problems on a MC680XX μP.

### 4.5.2 Circuit Idea - The SCB68155 Interrupt Handler

VMEbus handler circuits can be simplified using the SCB68155 interrupt handler IC. This device allows up to seven VMEbus, six local and one non-maskable interrupt to be handled by a microprocessor. Figure 4-17 shows one way to use the SCB68155. Table 4-6 shows how the interrupt inputs are prioritized.

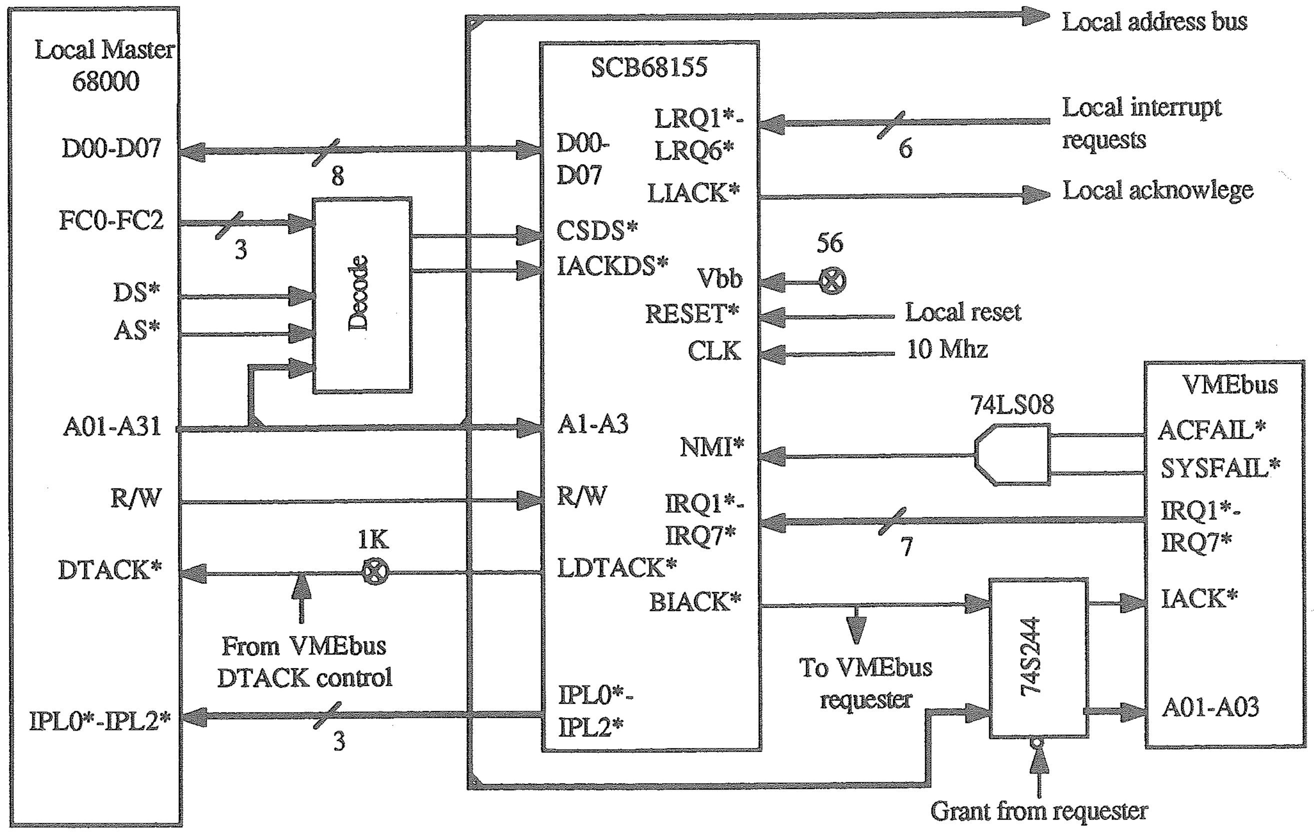
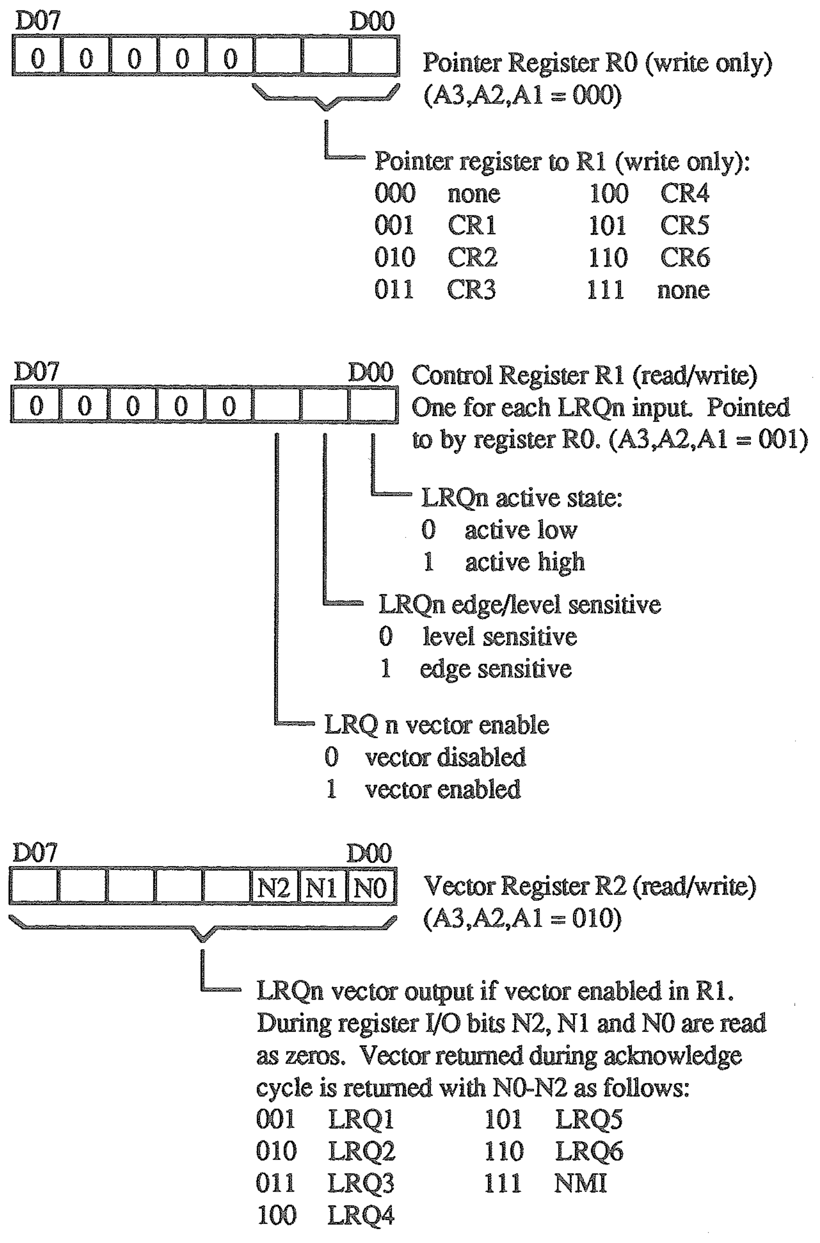


Figure 4-17. Block diagram showing a typical SCB68155 configuration.

Table 4-6. Prioritization of SCB68155 interrupt inputs.

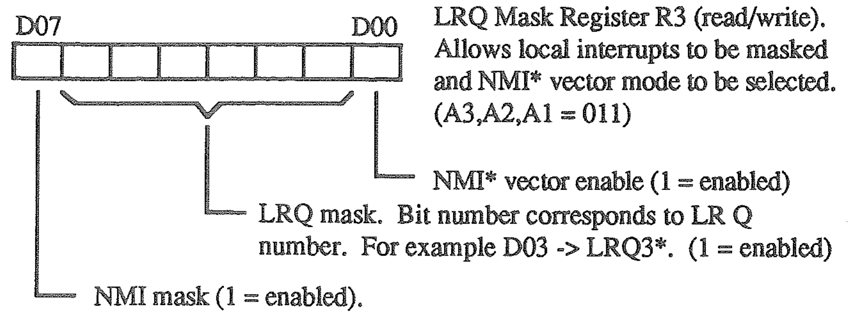
| Interrupt request level | Interrupt priority level outputs. | | | Level acknowledged by local master. | | |
|---|---|---|---|---|---|---|
| | IPL2* | IPL1* | IPL0* | A03 | A02 | A01 |
| NMI*, IRQ7* | 0 | 0 | 0 | 1 | 1 | 1 |
| LRQ6*, IRQ6* | 0 | 0 | 1 | 1 | 1 | 0 |
| LRQ5*, IRQ5* | 0 | 1 | 0 | 1 | 0 | 1 |
| LRQ4*, IRQ4* | 0 | 1 | 1 | 1 | 0 | 0 |
| LRQ3*, IRQ3* | 1 | 0 | 0 | 0 | 1 | 1 |
| LRQ2*, IRQ2* | 1 | 0 | 1 | 0 | 1 | 0 |
| LRQ1*, IRQ1* | 1 | 1 | 0 | 0 | 0 | 1 |

The circuit of Figure 4-17 shows the SCB68155 connected to a local 68000 master. The SCB68155 is controlled via an eight bit I/O interface using D00-D07. This enables the local master to configure the device using the internal registers shown in Figure 4-18. VMEbus interrupts on IRQ1* - IRQ7*, local interrupts on LRQ1* - LRQ6* and the non-maskable interrupt NMI* can be enabled or disabled via the internal registers.



D07
D00
0 0 0 0 0
Pointer Register R0 (write only)
(A3,A2,A1 = 000)
Pointer register to R1 (write only):
000 none
001 CR1
010 CR2
011 CR3
100 CR4
101 CR5
110 CR6
111 none
D07
D00
0 0 0 0 0
Control Register R1 (read/write)
One for each LRQn input. Pointed to by register R0. (A3,A2,A1 = 001)
LRQn active state:
0 active low
1 active high
LRQn edge/level sensitive
0 level sensitive
1 edge sensitive
LRQ n vector enable
0 vector disabled
1 vector enabled
D07
D00
N2 N1 N0
Vector Register R2 (read/write)
(A3,A2,A1 = 010)
LRQn vector output if vector enabled in R1. During register I/O bits N2, N1 and N0 are read as zeros. Vector returned during acknowledge cycle is returned with N0-N2 as follows:
001 LRQ1
010 LRQ2
011 LRQ3
100 LRQ4
101 LRQ5
110 LRQ6
111 NMI

Figure 4-18. SCB68155 programmers model.

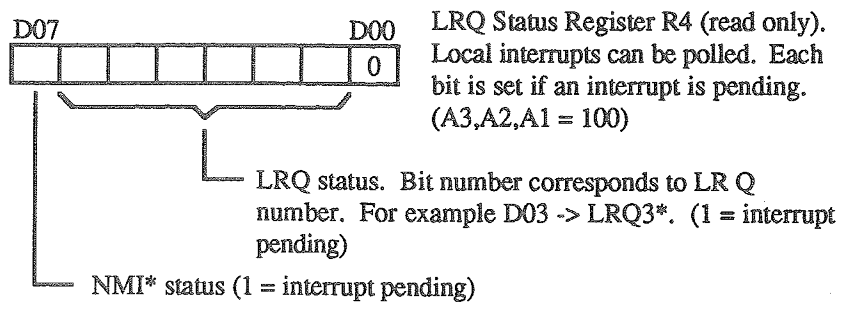


LRQ Mask Register R3 (read/write). Allows local interrupts to be masked and NMI* vector mode to be selected. (A3,A2,A1 = 011)

NMI* vector enable (1 = enabled)

LRQ mask. Bit number corresponds to LR Q number. For example D03 -> LRQ3*. (1 = enabled)

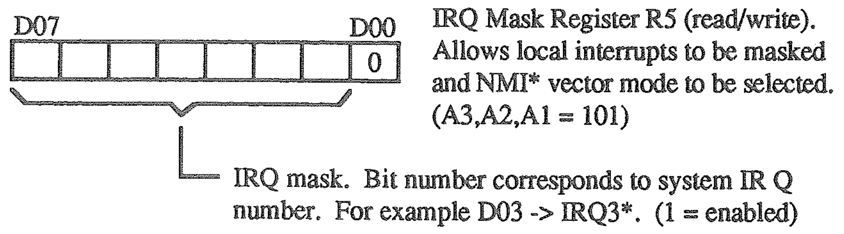
NMI mask (1 = enabled).



LRQ Status Register R4 (read only). Local interrupts can be polled. Each bit is set if an interrupt is pending. (A3,A2,A1 = 100)

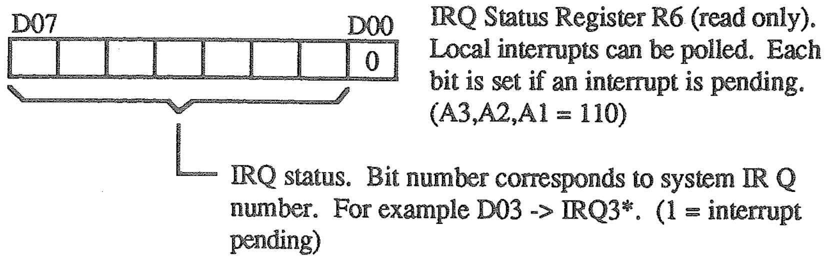
LRQ status. Bit number corresponds to LR Q number. For example D03 -> LRQ3*. (1 = interrupt pending)

NMI* status (1 = interrupt pending)



IRQ Mask Register R5 (read/write). Allows local interrupts to be masked and NMI* vector mode to be selected. (A3,A2,A1 = 101)

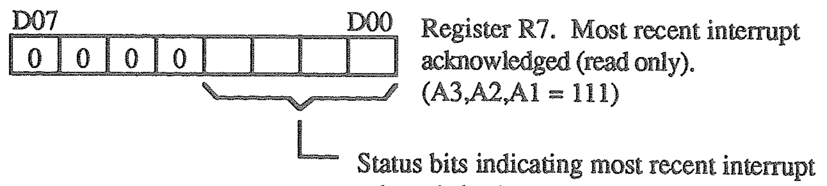
IRQ mask. Bit number corresponds to system IR Q number. For example D03 -> IRQ3*. (1 = enabled)



IRQ Status Register R6 (read only). Local interrupts can be polled. Each bit is set if an interrupt is pending. (A3,A2,A1 = 110)

IRQ status. Bit number corresponds to system IR Q number. For example D03 -> IRQ3*. (1 = interrupt pending)

Figure 4-18 (con't).



Register R7. Most recent interrupt acknowledged (read only). (A3,A2,A1 = 111)

Status bits indicating most recent interrupt acknowledged:

| | | | |
|---|---|---|---|
| 0000 | none | 1000 | none |
| 0001 | IRQ1* | 1001 | LRQ1* |
| 0010 | IRQ2* | 1010 | LRQ2* |
| 0011 | IRQ3* | 1011 | LRQ3* |
| 0100 | IRQ4* | 1100 | LRQ4* |
| 0101 | IRQ5* | 1101 | LRQ5* |
| 0110 | IRQ6* | 1110 | LRQ6* |
| 0111 | IRQ7* | 1111 | NMI* |

Figure 4-18 (con't).

Local interrupts LRQ1* - LRQ6* can be programmed to respond as either edge or level sensitive inputs. They can also be active high or active low. In addition each input can respond with either an internal or external STATUS/ID byte. Auto-vectored interrupts (which do not return a STATUS/ID) are not supported. The non-maskable interrupt NMI* is treated as the highest priority local interrupt, and is negative edge sensitive only.

VMEbus interrupts IRQ1* - IRQ7* are active low, level sensitive only, and must provide an external STATUS/ID byte during their interrupt acknowledge cycles.

When a local or VMEbus interrupt is generated, the SCB68155 prioritizes and notifies the local master using IPL0* - IPL2*. These signals are compatible with all MC680XX type devices. In response to the interrupt request, the local master generates an interrupt acknowledge cycle. It then asserts IACKDS* (to the SCB68155) and places the interrupt level to which it is responding on A01 - A03. If the current acknowledge cycle is in response to a local interrupt the device will assert LIACK* (Local Interrupt Acknowledge). Local interrupt requestors must then decode address lines A01-A03 accordingly. If the particular interrupt request being serviced was so programmed, the SCB68155 will then place an eight bit STATUS/ID byte onto data lines D00-D07 and assert DTACK*. This will notify the local master that it can latch the STATUS/ID byte and continue with the interrupt service routine. If the device was programmed to respond with an external STATUS/ID byte, it is the responsibility of the local interrupter to place the STATUS/ID onto D00-D07 and assert DTACK*.

If the interrupt acknowledge cycle is in response to a VMEbus interrupt, the SCB68155 will respond by asserting BIACK* (Bus Interrupt Acknowledge). This notifies external logic that the VMEbus acknowledge cycle should take place. Note that Figure 4-17 shows BIACK* going to the local bus requester. This is because the module must first acquire VMEbus mastership before it can get the STATUS/ID byte.

Programming the SCB68155 is done with the registers of Figure 4-18. Local interrupt requests are controlled using registers R0 - R4. During the initialization phase registers R0 - R2 must first be configured. R0 and R1 select the active state (high or low), edge or level sensitivity and STATUS/ID source. Register R0 is used as a pointer to select a local interrupt request level. The configuration bits for that particular input are then selected using Control Register R1.

If the SCB68155 was configured to generate an internal STATUS/ID byte in register R1 the value of the STATUS/ID for that level is placed into register R2. In this register, however, the pointer register is not used. When writing to R1 the lower three bits of the STATUS/ID should contain the level of interrupt request. The five high bits should contain the rest of the STATUS/ID byte.

Registers R3 - R6 contain interrupt masks and interrupt status for each request level.

Register R7 holds the state of the last interrupt acknowledged.

For more information about the SCB68155 please refer to its Technical Data Sheet available from Signetics.

## 4.6 References

Motorola Inc. "MC68153 Bus Interrupter Module Technical Data Sheet."

Signetics Inc. Microprocessor Data Manual, 1986.

VMEbus Specification, Revision C.1, VITA 1985

VMEbus Specification, IEEE-1014-87, VITA 1987

# Chapter 5
# Utility Functions

The Utility Bus is used for system initialization, periodic timing and power failure. It's really not really a bus, but the VMEbus specification calls it one.

## 5.1 System Clock

SYSCLK is a general purpose 16 Mhz clock signal. It has no relationship to other bus timing, and can be used for any purpose. Typical uses for SYSCLK include DTACK* generators, bus timers, memory refresh circuits, serial I/O time bases and synchronous state machines.

SYSCLK is generated by the slot 01 system controller, and can be used by any module. It is never turned off, even during system reset.

### 5.1.1 Timing

Figure 5-1 shows SYSCLK timing. At best it has a 50% duty cycle, but this can vary between 40% and 60%. Because SYSCLK is driven over the backplane, its wave shape may not always be ideal. Capacitance and inductance of the backplane, impedance mismatches and bus loading will cause distortion of the waveform as Figure 5-2 shows. SYSCLK is allowed to vary as much as 1.6% from its ideal frequency of 16 Mhz (over voltage and temperature).

The duty cycle of SYSCLK can vary between 40% and 60%, and care should be taken on circuits using both the rising and falling edges. When evaluating or designing VMEbus modules that use SYSCLK, make sure they can can tolerate the entire range of duty cycle.