Interfacing the VIC068A to the MC68020

This application note explains some of the features of the Cypress VIC068A and provides the first-time VIC068A user with simple implementations of these features. The VIC068A offers the most highly integrated VMEbus interface available today. It reduces the number of parts needed and saves board space. The emphasis in this application note is on interfacing the VIC068A as VMEbus A24/A16 D16/D08(E0) master/slave to the Motorola 68020.

Reset Operation

The VIC068A performs three distinct reset operations:

- Internal reset, activated by the IRESET pin, which initializes most of the internal registers
- System reset, essentially the same as IRESET, but is activated by writing ($F0) to the system reset register, or by asserting IRESET when the VIC068A is the VMEbus controller (SCON pin asserted)
- Global reset initializes all the VIC068A registers

After a reset, the 680X0 processor reads its initial stack pointer (SSP) and program counter (PC) from addresses $0 through $7. One way to handle this is to remap the boot-up ROMs to the low addresses for the first few cycles of the processor.

Figure 1 shows a circuit you can use to do this. The circuit uses a serial-in/parallel-out shift register (the 74HC164) to generate the MAP signal. This active-Low signal can be used with address-decode logic to force boot ROM access to the lower addresses during initial power up. Asserting the 74HC164 CLEAR pin drives all the parallel outputs Low, which asserts the selected MAP signal. With the two serial inputs tied High, each Low-to-High transition of the 68020 AS clocks the High through the shift register and out each of the parallel outputs. By picking the proper output for the MAP signal, you can decode from 1 to 8 of the initial processor cycles. You can use the MAP signals on memory configurations that are 8, 16, or 32 bits wide by using the QH, QD, or QB outputs, respectively.

Using the Processor RESET Instruction

The OR gate in *Figure 1* ensures that the 74HC164 is cleared only when HALT and RESET are both asserted. This allows the use of the 68020 RESET instruction without inadvertently reasserting MAP. An alternative to this approach is to use two small-signal diodes (1N4148) and a pull-down resistor in place of the OR gate. This change reduces the design's parts count by eliminating the 74HC32.

A ROM remapping circuit must be used whether the RESET instruction is issued or not because of the way the VIC068A arbitrates local bus contention between the 68020 and the VMEbus. Contention occurs when both master and slave operations are requested concurrently (MWB asserted and SLSEL0, SLSEL1, or IFCSEL asserted). The VIC068A indicates this contention by asserting DEDLK. You can deal with the condition by setting bit 4 of the VIC068's interface configuration register ($AF) to assert HALT along with LBERR when DEDLK occurs (68020 bus retry sequence). The VIC06 then waits for the 68020 to deassert the MWB input. Once this happens, the VIC068A releases LBERR but continues to assert HALT to keep the 68020 off the local bus. The VIC068A then allows the slave operation to complete and deasserts HALT. The 68020 can now retry the contested bus cycle.

Internal Reset

At first glance, the IRESET might seem the logical choice for implementing the power-on reset. Because the IRESET input has some built-in hysteresis, a simple RC circuit would be appropriate for applying the power-on signal.

IRESET does not initialize the local bus timing register nor any of the slave select registers, however. Additionally, the VIC068A powers-up with the DRAM refresh option enabled (bit 4 of the arbiter/requester configuration register $B3 High). This condition is acceptable if you are using DRAM but adversely affects the external reset circuit in *Figure 1*. Specifically, for the first DRAM refresh cycle, the VIC068A deasserts RESET but maintains HALT in the active (Low) state and toggles AS. This action causes shift operations in the 74HC164.

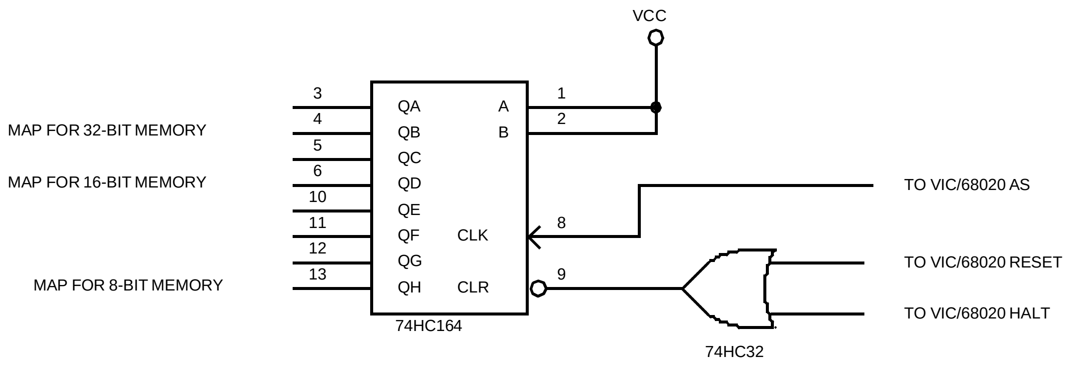

Figure 1. ROM Remapping Circuit

You can activate DRAM refresh after reset by writing a 1 to bit 4 of the arbiter/requester configuration register ($B3).

System Reset

The assertion of SYSRESET on the VMEbus typically activates system reset, but only when a global reset is not taking place. When the VIC068A is configured as the system controller (SCON pin asserted), it drives the SYSRESET pin for the required 200 ms during an internal or global reset.

Global Reset

The global reset is the most useful for power-up purposes because it places all the VIC068A registers in a known state. You initiate a global reset by asserting IPL(0) concurrent with or just after asserting IRESET. These reset signals should not be asserted until the VCC power source has stabilized at 5 volts. Because IPL(0) is also one of the encoded interrupt lines for the 68020, you must assert this signal with an open-collector or three-state device.

In using global reset, bear in mind that when the VIC068A powers-up it ignores the VMEbus SYSRESET. The VIC068A releases HALT and RESET after the 200-ms time out even if the current VMEbus master asserts SYSRESET past this required minimum time. This automatic release is a useful feature because it eliminates reliance on the system controller to release SYSRESET to start the power-up sequence. Refer to the *VIC068A/VAC068A User's Guide* for more information on global reset.

The VIC068A generates a LBERR if you try to access the VMEbus or any of the VIC068A registers before SYSRESET is deasserted. One solution to this problem is to structure the software so that the VIC068A registers are set up as late as possible in the power-up sequence. You can also temporarily point the 68020 BERR exception vector to an address containing an RTE instruction and let the 68020 cycle in a BERR/RTE loop until SYSRESET is deasserted. The latter approach provides an opportunity to be the first board in a system to request VMEbus mastership.

Connecting the Bus Lines

Figure 2 shows the standard buffer configuration for an A24/D16 VMEbus connection. This design also supports A16 and D08(E0) operation.

The D16/D08(E0) Data Bus

Connect the VIC068A to the 68020 as you would any 16-bit peripheral device. The 74FCT543 data buffer connects between the 68020 data bus's upper byte (D31–24) and the VMEbus D15 - 8 data lines. The lower byte (LD7–LD0) is buffered through the VIC068A to the VMEbus low byte (D7–D0). Several control signals connect directly from the VIC068A to the 74FCT543: DENO (data enable out) to OEAB (Output enable A-to-B), LWDENIN (lower word data enable) to OEBA (Output enable B-to-A), LEDO (latch enable data out) to LEAB (Latch enable A-to-B), and LEDI (latch enable data in) to LEBA (latch enable B-to-A).

The Address Bus

The A24/A16 configuration requires the use of two more 74FCT543 devices to buffer and control the VMEbus A23 through A8 signals. The 74FCT543 LEAB, LEBA, and OEBA inputs connect directly to the VIC068A LADO (latch address out control), LADI (latch address in control), and ABEN (enable address out control) outputs, respectively. The output of the VIC068A LAEN (local-address enable control) must be connected to the 74FCT543 OEBA input through an inverter because LAEN is an active-High output and OEBA is an active-Low input.

Connecting the DSACK Lines

During the normal local bus operation, the 68020's slave devices (i.e., memory, UART, parallel port) must tell the processor the size of their data bus. This is done by asserting the DSACK1 inputs, which tells the 68020 that the port is a 16-bit device. Asserting DSACK0 instead indicates that the port is an 8-bit device. Asserting both DSACK1 and DSACK0 indicates that the port is 32 bits wide. To configure the VIC068A as a 16-bit port, simply connect the 68020 DSACK1 to the VIC068A DSACK1.

So long as there you have no requirement for VMEbus access to 8-bit devices on the local bus, you do not need to do anything with the VIC068A DSACK0 pin except terminate it (pull it High).

When you do need to access 8-bit devices, a small problem arises with the way the VIC068A acknowledges register accesses and interrupt-acknowledge cycles. During these cycles, the VIC068A always asserts both DSACK1 and DSACK0, whether the WORD input is asserted or not. And in VMEbus master cycles, when talking to an 8-bit device on the VMEbus, the VIC068A responds with DASCK0 to acknowledge the 8-bit transfer completion.

The solution to the DSACK0 problem is simple but can be complicated to implement: You must break the DASCK0 connection between the VIC068A and the 68020 during interrupt acknowledge or VIC068A register access (CS) cycles. The circuit needed to do this is a bidirectional, open-collector buffer between the VIC068A and 68020. The buffer should be inactive in both directions only when the VIC068A FCIACK or CS inputs are asserted. In *Figure 3*'s PAL equations, the DSACK0_020 and VIC068A DSACK0 equation illustrates how to handle the DSACK0 connection.

Master Operation

VMEbus master operation with the VIC068A is easily accomplished with the use of the MWB (module-wants-bus) input. The VMEbus can be requested at any level (0–3). The VMEbus can also be dynamically changed via the arbiter/requester configuration register ($B3), which eliminates the need for hardware jumpers. All VMEbus release modes are supported through the release control register ($D3). Support for write posting means that the local processor can write to the VMEbus without having to wait for the current bus master to release the bus or for the arbitration logic to assert the correct BGIN 9 (bus grant in) line. The VIC068A takes cares of this overhead for the local processor, improving system throughput.

To request VMEbus mastership, the 68020 asserts the MWB input. You can think of MWB as a VMEbus chip select. When interfacing to the VMEbus as an A24 or A16 device, you can have access to the whole VMEbus address space by decoding a 32-Mbyte area of the 68020 address space for VMEbus operations. The ASIZ1–0 pins tell the VIC068A whether the current cycles represent an A32, A24, or A16 operation. You can use the upper 16-Mbyte address space (A24 High) for VMEbus A23 operation and the lower half (A24 Low) for VMEbus A16 operation by following three steps: decode A31 through A25 to generate MWB, tie the ASIZ1 input High, and

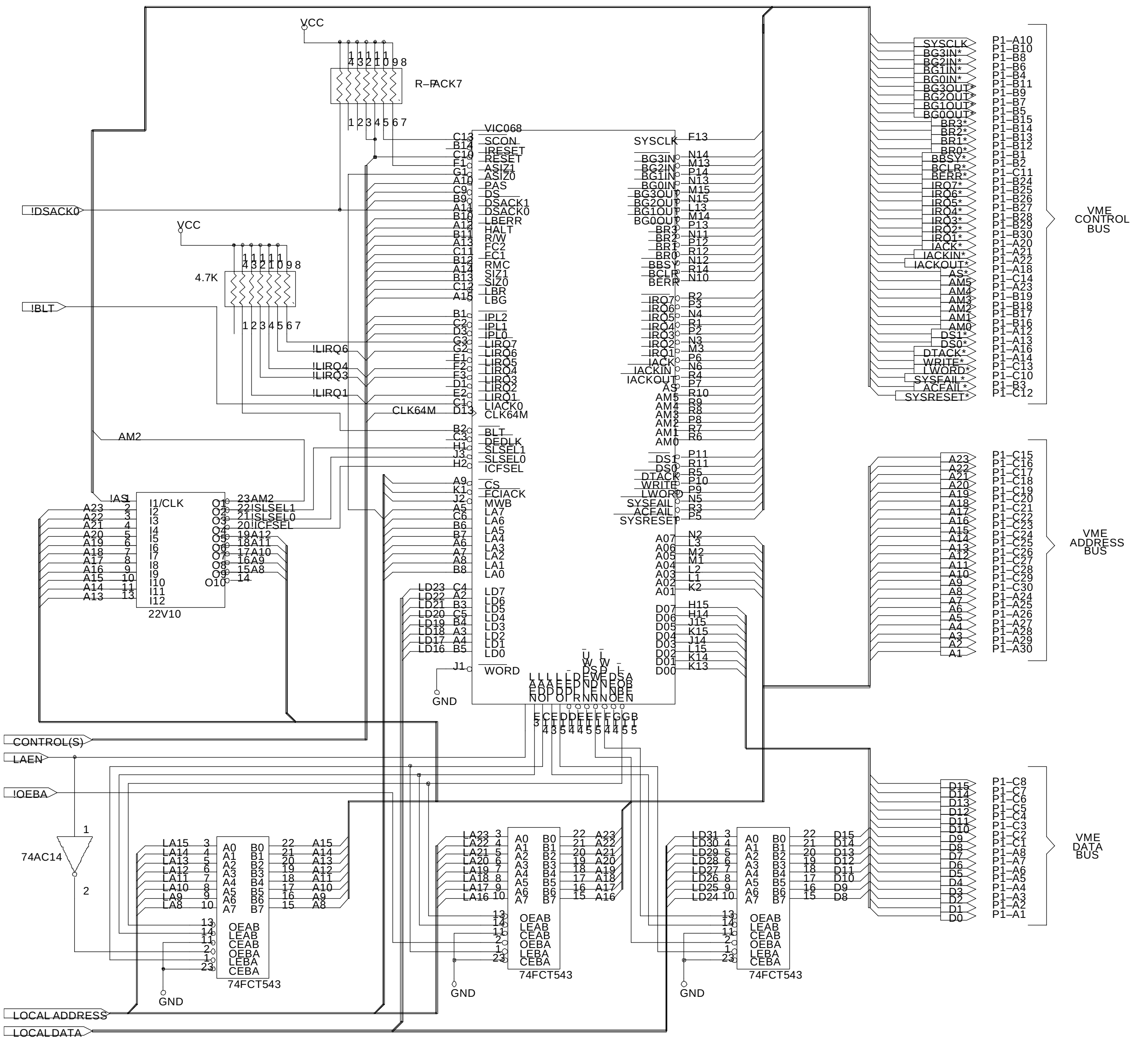

Figure 2. Address and Data Bus Connections

connect the 68020 A24 address line to the VIC068's ASIZ0 input. *Figure 3* demonstrates this way of decoding MWB.

When the VIC068A recognizes a valid slave access, the device asserts LBR (68020 BR input) and waits for LBG assertion (68020 BG output). Once the VIC068A receives LBG, the device becomes the local bus master at the conclusion of the current cycle and completes the requested VMEbus slave operation. If the VIC068A is the only DMA device on the local bus, there is no need to generate BGACK (bus grant acknowledge) for the 68020. But if any other devices are capable of local bus mastership, you have to provide the arbitration logic and the BGACK signal for the 68020. Keep in mind, too, that other DMA devices must be able to recognize and deal appropriately with the 68020 bus-cycle entry operation (BERR and HALT asserted).

Slave Operation

The VIC068A can provide full VMEbus slave operation by dual-porting local memory with little or no 68020 overhead. The normal slave access operation starts by providing SLSEL0 or SLSEL1 through VMEbus address decoding. The circuits in *Figure 2* and *Figure 3* use a 22V10 PAL for this purpose. Always qualify VMEbus address decoding with the AS and/or DS1–0.

Decoding SLSEL0, SLSEL1, and IFCSEL

Figure 3 illustrates a typical PAL specification that you can use to provide address decoding for SLSEL0, SLSEL1, and IFCSEL. The VIC068A uses all the address modifier lines (AM5–0) to qualify the access mode. Address decoding can ignore these inputs. The VIC068A then decides if the access

```
module_CYCLE_DECODE;
   Cycle_decode device 'PV22V10';
   VCC,GND
                                         pin 24,12;
”inputs (15)
   A31,A30,A29,A28,A27,A26,A25,A19    pin 1,2,3,4,5,6,7,8;
   SLSEL1, SLSEL0                     pin 9,10;
   FC2,FC1,FC0,AS,LBG                 pin 13,14,15,16,17 ”for FCIACK and VIC_Cycle output

”outputs (6)
   VIC_DSACK0,DSACK0_020   pin 18,19;   ”To VIC DSACK0 and local system DASCK0
   VIC_CYCLE               pin 20;         ”current   bus cycle is VMEbus
   FCIACK                  pin 21;         ”Interrupt Acknowledge Cycle
   PRE_MWB,MWB             pin 22,23;      ”VIC module-wants-bus (with and without AS)

”output type declarations
   VIC_CYCLE,PRE_MWB,MWB         istype 'com';
   FCIACK,VIC_DSACK0,DSACK0_020  istype 'com';
   VIC_CYCLE.OE,FCIACK.OE        istype 'com';
   PRE_MWB.OE,MWB.OE             istype 'com';
   VIC_DSACK0.OE,DSACK0_020.OE   istype 'com';

equations in CYCLE_DECODE
”Enable ALL outputs except DSACK's
   VIC_CYCLE.OE    =1;
   PRE_MWB.OE      =1;
   MWB.OE          =1;
   FCIACK.OE       =1;

”This signal tells everybody that the VIC068A is controlling the current bus cycle
   !VIC_CYCLE<T>=!LBG & AS<T><T><T><T>”signal is asserted while AS is still high
           #!VIC_CYCLE & !LBG &!AS ”maintain signal through entire cycle

”Interrupt acknowledge cycle (68020 to VIC).  Use VIC_CYCLE to insure this is not a VMEbus
master cycle
   !FCIACK = A31 & A30 & A29 & A28 & A27 & A26 & A25 & A19 & FC2 & FC1 & FC0 & !AS &
VIC_CYCLE;

”VME A24 access is at addresses $04000000 - $04FFFFFF.  A16 access is at addresses $0500000 -
$05FFFFFF (ASIZ0 is tied to LA24)
   !MWB = !A31 & !A30 & !A29 & !A28 & !A27 & A26 & !A25 & VIC_CYCLE &!(FC2 & FC1 & FC0);

”This is the same signal as MWB but the AS input is removed to provide an early VMEbus master
cycle indication input to other PLDS
   !PRE_MWB = !A31 & !A30 & !A29 & !A28 & !A27 & A26 & !A25 & VIC_CYCLE &!(FC2 & FC1 & FC0);

”This signal is connected directly to the VIC DSACK0.  It generates the VIC DSACK0 for VMEbus
slave accesses to 8 bit device
   !VIC_DSACK0 = !VIC_CYCLE & !DSACK0_020;

”This enables VIC_DSACK0 only when VIC is the local bus master (slave accesses)
   VIC_DSACK0.OE = !VIC_CYCLE & (!SLSEL0 # !SLSEL1);

”This signal is connected to the 68020 DSACK).  It generates the 68020 DSACK0 for VMEbus mas-
ter accesses to 8 bit devices
   !DSACK0_020 = !MWB & VIC_CYCLE & !VIC_DSACK0;

”This enables the 68020 DSACK0 only when the VIC is the VMEbus master
   DSACK0_020.020 = !MWB & VIC_CYCLE;
   end_CYCLE_DECODE
```

Figure 3. ABEL Equations for PALC22V10 Cycle Decoding

mode is legal and completes the cycle or generates the VMEbus BERR signal, depending on the value programmed in the slave select registers. You can also qualify the select outputs with the address modifiers and let the initiating device time-out if the access is not legal.

The IFCSEL input gives the VMEbus access to some of the VIC068A control registers and the interprocessor communication registers. These registers are available only through an A16 privileged-mode access.

The PAL specification in *Figure 3* configures SLSEL0 to dual-port a 4-Kbyte (minus 256 bytes) space of local RAM as an A16 non-privileged access input and decodes IFCSEL in the SLSEL0 area's upper 256 bytes. You can use this 256-byte space for mailbox communication between boards in a multi-master system.

SLSEL1 is decoded as an A24 supervisory-only access and provides full dual-porting of the 68020 board's E2PROM program memory. This allows the VMEbus system controller to put the system in a reset and hold state by asserting bit 6 of the VIC068's interprocessor communications register 7. The VMEbus master can then reprogram the entire program memory space. Once that operation is complete, the controller can use the interprocessor communications register 7 to release the reset and hold state. The board comes up running the newly installed program.

Take care when decoding SLSEL0, SLSEL1, and IFCSEL. The VIC068's operation is undefined when more that one of these inputs is active simultaneously.

Decoding for Supervisor/User Mode

You can use the VMEbus AM2 signal to select user (AM2 Low) or supervisor (AM2 High) modes. The AM2 input is used as part of the slave-select decoding shown in *Figure 3*.

Dealing with A24 and A16 Slave Accesses

Regardless of the access address size, the 74FCT543 address buffer outputs are enabled. Typically, the backplane pulls unused VMEbus address lines High passively, but most masters drive these lines regardless of the bus-cycle-address size. If this is not desirable, control the output-enable signals with the upper address line buffers using the VMEbus address modifiers. *Table 1* illustrates how to use AM5 and AM4 to determine the bus-cycle-address size.

Table 1. Determining Bus-Cycle-Address Size.

AM5	AM4	Cycle
H	H	A24 Access
H	L	A16 Access
L	L	A32 Access

You can derive individual enables for each of the VMEbus address latches by ANDing one or both of these address modifiers with the VIC068A LAEN (local-address enable) signal; modify both if operating in an A32 system.

Remember to provide a stable level for the local-address lines because nothing drives them during VMEbus accesses. You can ensure a stable level using 4.7Ω pull-up or pull-down resistors on the local-bus A31–A16 lines. The local-bus address buffers can be set to the desired address state and enabled with the same latch-enable signals.

Dual-Porting Local Memory

The PAL specification in *Figure 3* generates a signal called VIC_CYCLE than can serve as part of the local-address decoding to re-map local memory for dual-porting on the VMEbus. This approach allows memory placement at a VMEbus address independent of the local address.

Interrupts

The VIC068A interrupt structure is very versatile. One of the most useful features is the ability to redefine interrupt levels, and thus priorities, under normal program control. The VIC068A supports all seven levels of VMEbus interrupt as well as the seven local-interrupt levels. Interrupts are also available to notify the 68020 of VMEbus status and error conditions.

Figure 3 shows how to decode the 68020 interrupt acknowledge bus cycle to generate the VIC068A FCIACK input. You can omit A19-A16 from the equation if you do not use breakpoints, a memory management unit (MMU), or a coprocessor (68881/68882).

Using LIACK0

The LIACK0 output is typically connected to the 68020 AVEC input to initiate autovectoring of interrupts to which the VIC068A has not been programmed to respond. You can also use LIACK0 with the IPL(2–0) outputs to generate interrupt-acknowledge signals to other 680x0-compatible interrupting devices.

LIRQ7–1 Inputs

The LIRQ7–1 inputs are the interrupt request inputs to the VIC068. The control register for each input allows you to determine the input's polarity (high/low) and sensitivity (level or edge). The control register also allows you to define whether the VIC068A supplies the vector during interrupt acknowledge cycles or asserts LIACKO (local-interrupt acknowledge out), sets the level of interrupt the 68020 sees on IPL2–0, and enables or disables the interrupt. You do not need to terminate these inputs if you leave them unconnected, but you must pull them up externally if they are used.

The local interrupts (IPL2–0) are grouped and have a common vector base register ($57). This vector base is added to the encoded interrupt level programmed in each of the interrupt control registers to supply a unique vector to the 68020 for each interrupt input.

LIRQ2 is a special case because it can be used as an interrupt clock tick timer. You enable the timer through bits 2 and 3 of slave-select control register 0($C3). When enabled, LIRQ2 becomes the timer output, and the local-interrupt control register 2 ($2B) becomes the timer's interrupt-control register. The timer's periodic interrupt can be set to 50, 100, or 1000 Hz. If you plan on using the tick timer, do not connect the external interrupts to LIRQ2 because this pin becomes an output.