```
title "V123 beam synchronous encoder module";
%drawing 94028037
 next assembly 94028034
 by C. R. Conkling Jr.
 conversion October 9, 1997 V101 input modules.
 conversion October 29, 1997 28.152 MHz
 conversion November 4, 1997 56.304 MHz
 conversion December 8, 1997 28.152 MHz

 drawing number 94028037: compile with vmeid.mif that contains the
 V123 module serial number and revision number. Also compile with a
 SRAM.mif  that contains the initial state of the event trigger to
 event code translation table - normally SRAM contents equal SRAM
 address.

event encoder/input module A16 addressing
|-----------|-----------|-----------|-----------|
|15|14|13|12|11|10|09|08|07|06|05|04|03|02|01|00|
-------------------------------------------------
|----SWR----| 0| ENC_ADDR
               | 0  0  0  0  0  X  X  X  X  X  0|R/- BYTE status/ID PROM
               | 0  0  0  0  1  0  0  0  0  0  1|R/W BYTE CSR
               | 0  0  0  0  1  0  0  0  0  1  1|R/W BYTE interrupt vector
               | 0  0  0  0  1  0  0  0  1  0  1|R/W BYTE interrupt level
               | 0  0  0  0  1  0  0  0  1  1  1|R/W BYTE interrupt vector
               | 0  0  0  0  1  0  0  1  0  0  1|R/W BYTE FIFO
               | 0  0  0  0  1  0  0  1  1  0  0|R/W LWORD revolution
               | 0  0  1  X  X  X  X  X  X  X  X|R/W WORD event code SRAM
|           | 1| INP_ADDR  |
               |--CHANNEL--| 16-channel codes from P2 backplane
                           | 0  X  X  X  X  X  X|R/- BYTE VME status/id
                           | 1  0  0  0  0  0  0|R/W HI-BYTE channel enable
                           | 1  0  0  0  0  0  1|R/W LO-BYTE channel enable
                           | 1  0  0  0  0  1  0|R/W HI-BYTE interrupt enable
                           | 1  0  0  0  0  1  1|R/W LO-BYTE interrupt enable
                           | 1  0  0  0  1  0  0|-/W HI-BYTE event trigger
                           | 1  0  0  0  1  0  1|-/W LO-BYTE event trigger
                           | 1  0  0  0  1  1  0|R/- HI-BYTE lost event
                           | 1  0  0  0  1  1  1|R/- LO-BYTE lost event
                           | 1  0  0  0  1  1  1|BYTE unused
                           | 1  0  0  0  1  1  1|R/- BYTE interrupt vector
                           | 1  0  0  0  1  1  1|BYTE unused
                           | 1  0  0  0  1  1  1|R/W BYTE interrupt level%
%-------------------------------------------------------------------------------%

%LAST_V101 = the numeric value of the last V101 trigger channel%
constant LAST_V101 = 63; %four V101 modules%

%Compiler options:
 connected pin CLOCK_OUT to CLOCK_IN for simulation
 logic synthesis FAST, and ONE-HOT%

%Altera LPM's used in this design%
include "lpm_rom"; %VMEid PROM%
include "lpm_ram_dq"; %256 x 8 SRAM%
```

```
include "csfifo"; %256 x 8 FIFO%

subdesign 94028037

(/SYSRST, %VMEbus system reset%
 28_MHz, %28 MHz BEAM_CLOCK backup clock%
 SWR[15..12], %base address comparison%
 BA[15..1], %VMEbus address lines%
 BAM[5..0], %VMEbus address type%
 /DS0, /DS1, /LWORD, %VMEbus address control%
 /AS, /IACK, /WRITE, /IACKIN, %VMEbus control%
 /TAKE_DATA, %input module bus data strobe%
 TRIGGER[7..0], %input module bus data%
 REVOLUTION_TICK, %RF system pulse indicating beam reference point%
 BEAM_CLOCK, %28 MHz RF system beam synchronous clock%
 CLOCK_IN : input; %BEAM_CLOCK if available, otherwise 28_MHz backup%

 D[31..0] : bidir; %local data & address buses%

 /DTACK, %acknowledge transfer%
 /IRQ[7..1], %select interrupt request level%
 /IACKOUT, %VMEbus interrupt control%
 /INP_ADDR, %input module bus MSB's of module address%
 INP_INT_VEC[7..4], %input module bus MSB's of module interrupt vector%
 /BUSY, %beginning of input module priority chain%
 14_MHz, %input module bus clock%
 /OFFLINE, /SELECT, /INTERRUPT_LED, %OFF LINE, VME SELECT, and INTERRUPT LEDs%
 /EVENT_LED, /V101_EVENT_LED, /FIFO_EVENT_LED, %event trigger LEDs%
 /BEAM_CLOCK, /REVOLUTION_TICK, %beam LEDs%
 /HI_ENA, /LO_ENA, /LW_ENA, %module VMEbus data control%
 CLOCK_OUT, %link to CLOCK_IN%
 EVENT_LINK[3..1] : OUTPUT;) %event TIMELINE output, bi-phase mark
transmission%

variable

SHORT_AD, SWR_COMP, LINK, CLOCKFAIL, INDEXFAIL, REV_RESET, CHANGE, EMPTY,
FULL, CLEAR, SRAM_ADDRESS[7..0], CLEAR_EXT_EVENT : node;

MODULE, SR[2..0], AL[10..0], DDTACK, IRQX, CR[7..0], IDTACK, STATUS, IACKOUT,
LOCAL, INT_SR[2..0], EVENT_LINK[3..1], EXT_EVENT[8..0],
COUNT24[4..0], DISPLAY[15..0], SELECT[1..0], INP_ADDR, EXTERNAL,
INT_VEC[7..0], INT_LEV[7..0], INP_INT_VEC[7..4], ENA_VME, ER_OS[3..0],
CLOCK_FAIL[2..0], TICK_FAIL[2..0], REVOLUTION[31..0], SYNC_OS[1..0], READY,
EVENT_LED[1..0],COUNT15[3..0], COUNT2421, FIFO_WRITE[1..0], FIFO_REG_IN[7..0],
14_MHz, EVENT_CODE[7..0], FIFO_REG[8..0], FIFO_RESET,
V101_EVENT_LED[1..0], FIFO_EVENT_LED[1..0], INTERRUPT_LED[1..0] : dffe;

TRI_DATA[31..0] : tri;

%LPM functions%
PROM : lpm_rom with (LPM_WIDTH = 8, LPM_WIDTHAD = 5,
       LPM_FILE = "vmeid.mif",
       LPM_ADDRESS_CONTROL = "unregistered", LPM_OUTDATA = "unregistered");
```

```
SRAM : lpm_ram_dq with
      (LPM_WIDTH = 8, LPM_WIDTHAD = 8,
       LPM_FILE = "sram.mif",
       LPM_INDATA = "unregistered",
       LPM_ADDRESS_CONTROL = "unregistered",
       LPM_OUTDATA = "unregistered",
       USE_EAB);

FIFO : csfifo with (LPM_WIDTH = 8, LPM_NUMWORDS = 256);

%state machine, compiled one-hot%
FIFO_READ: machine with states (S0, S1, S2);

begin
%-----------------------------------------------------------------------------%
%3-bit shift register, SR[2..0], clocked by 14_MHz to develop local data
 strobes, and VMEbus response, DTACK. SR[2..0] is held reset by DS[1..0].
 When a VMEbus cycle starts, DS[1..0] enable the shift register, the cycle
 is delayed until VME_ENA indicates that the event link generator will not
 need the SRAM for a VMEbus cycle time.
 SR2 LATCH MODULE & ADDRESS 0-71ns after start
 SR1 MODULE to WRITE = 71ns write setup
 SR0 MODULE to DTACK = 142ns read access%
SR2.d = ENA_VME # SR2;
SR[1..0].d = SR[2..1];
SR[2..0].clk = 14_MHz;
SR[2..0].clrn = /SYSRST & (!/DS0 # !/DS1);

%Address latch%
AL[10..0].d = (BA[10..1], (/DS1 & !/DS0));
AL[].clk = SR[2];
AL[].clrn = global(/SYSRST);

SWR_COMP = SWR[15..12] == BA[15..12]; %encoder base address%
SHORT_AD = BAM[5..0] == B"101X01"; %VMEbus short address%

%latch encoder module base address
 module will respond byte between BASE_ADDRESS and + 0x7F
 module will respond LWORD at BASE_ADDRESS + 0xC%
MODULE.d = SWR_COMP & !BA11 & SHORT_AD & /IACK & (
           /LWORD & (/DS0 $ /DS1)
         # !/LWORD & BA[10..2] == b"000010011");
MODULE.clk = SR[2];
MODULE.clrn = /SYSRST & (!/DS0 # !/DS1);

%latch five MSB's of input module base address%
INP_ADDR.d = SWR_COMP & BA11 & SHORT_AD & /IACK;
INP_ADDR.clk = SR[2];
INP_ADDR.clrn = /SYSRST & (!/DS0 # !/DS1);

/INP_ADDR = !INP_ADDR;

%VMEbus data bus buffer enables%
/LW_ENA = !(MODULE & !/LWORD);
/HI_ENA = !(MODULE & (!/DS1 & /DS0 # !/LWORD));
```

```
/LO_ENA = !(MODULE & (/DS1 & !/DS0 # !/LWORD) # STATUS);

%VMESEL LED driver output%
SELECT[0].d = VCC;
SELECT[0].clk = MODULE;
SELECT[0].clrn = !SELECT[1];
SELECT[1].d = SELECT[0];
SELECT[1].clk = DISPLAY[15];
SELECT[1].clrn = global(/SYSRST);

/SELECT = !SELECT[1];

%Low frequency clock for LED displays%
DISPLAY[15..0].clk = 14_MHz;
DISPLAY[].clrn = global(/SYSRST);
DISPLAY[] = DISPLAY[] + 1;

%-----------------------------------------------------------------------------%
%Module registers%
%VME ID prom contained in one of the 10K10 EAB's%
PROM.address[4..0] = AL[5..1];

%Command register
 CR7 = CLOCKFAIL read only
 CR6 = INDEXFAIL read only
 CR5 = unused
 CR4 = unused
 CR5 = unused
 CR6 = unused
 CR1 = INTERRUPT ENABLE
 CR0 = MODULE ENABLE
interrupt on command register status bit change%
CR[7..6].d = (CLOCKFAIL, INDEXFAIL);
CR[7..6].clk = CHANGE;

CHANGE = /SYSRST & (CLOCKFAIL, INDEXFAIL) != CR[7..6];

%read-write command register%
CR[5..0].d = D[5..0];
CR[5..0].clk = MODULE & (AL[10..0] == h"41") & SR[1] & !/WRITE;
CR[5..0].clrn = global(/SYSRST);

%drives OFF LINE LED%
/OFFLINE = CR[0];

%Module interrupt status/ID register. The interrupt status/ID register
 contains the interrupt vector which is returned during an interrupt cycle.%
INT_VEC[7..0].d = D[7..0];
INT_VEC[7..0].clk = MODULE & (AL[10..0] == h"43") & SR[1] & !/WRITE;
INT_VEC[7..0].clrn = global(/SYSRST);

%Module interrupt request level register. The interrupt request is made
 on the level selected in this register%
INT_LEV[7..0].d = D[15..8];
INT_LEV[7..0].clk = MODULE & (AL[10..0] == h"45") & SR[1] & !/WRITE;
```

```
INT_LEV[7..0].clrn = global(/SYSRST);

%generate seven interrupt request level outputs%
for LEVEL in 7 to 1 generate
/IRQ[LEVEL] = !(INT_LEV[2..0] == LEVEL & IRQX);
end generate;

%MS nibble of input module interrupt vector. Output on input module bus.
 LS nibble is input module input bus position%
INP_INT_VEC[7..4].d = D[7..4];
INP_INT_VEC[7..4].clk = MODULE & (AL[10..0] == h"47") & SR[1] & !/WRITE;
INP_INT_VEC[7..4].clrn = global(/SYSRST);

%-------------------------------------------------------------------------------%
%256-byte FIFO input logic. FIFO uses one the three 10K10 EAB's. FIFO_WRITE
 one-shot triggers if VMEbus event trigger > LAST_V101.%
FIFO.clock = 14_MHz;
FIFO.clockx2 = 28_MHz;

%FIFO input register%
FIFO_REG_IN[7..0].d = D[7..0];
FIFO_REG_IN[7..0].clk = MODULE & (AL[6..0] == h"49") & SR[1] & !/WRITE;
FIFO_REG_IN[7..0].clrn = global(/SYSRST);
FIFO.data[7..0] = FIFO_REG_IN[7..0];

%FIFO write request - FIFO_REG_IN must be > LAST_V101 to load FIFO%
FIFO_WRITE[0].ena = D[7..0] > LAST_V101;
FIFO_WRITE[0].d = VCC;
FIFO_WRITE[0].clk = MODULE & (AL[6..0] == h"49") & SR[1] & !/WRITE;
FIFO_WRITE[0].clrn = !FIFO_WRITE[1];
FIFO_WRITE[1].d = FIFO_WRITE[0];
FIFO_WRITE[1].clk = 14_MHz;
FIFO_WRITE[1].clrn = global(/SYSRST);

FIFO.wreq = FIFO_WRITE[1];

%FIFO data status flags%
EMPTY = FIFO.empty;
FULL = FIFO.full;

%If FIFO output is available, unload FIFO and store in FIFO_REG[]%
FIFO_READ.clk = 14_MHz;
FIFO_READ.reset = global(!/SYSRST);

case FIFO_READ is
 when S0 => if  !CR[0] # EMPTY # FIFO_REG[8] then FIFO_READ = S0;
               else FIFO_READ = S1;
            end if;
            FIFO.rreq = GND;
 when S1 => FIFO_READ = S2;
            FIFO.rreq = VCC;
 when S2 => FIFO_READ = S0;
            FIFO.rreq = GND;
end case;
```

```
%register FIFO output - FIFO_REG[8] is data available flag%
FIFO_REG[].ena = S2;
FIFO_REG[].d = (VCC, FIFO.q[7..0]);
FIFO_REG[].clk = 14_MHz;
FIFO_REG[].clrn = /SYSRST & !FIFO_RESET;

%FIFO_RESET resets FIFO_REG[] after FIFO event code is sent%
FIFO_RESET.ena = COUNT2421;
FIFO_RESET.d = SRAM_ADDRESS[7..0] > LAST_V101;
FIFO_RESET.clk = global(CLOCK_IN);
FIFO_RESET.clrn = global(/SYSRST);

%-----------------------------------------------------------------------------%
%REVOLUTION counter%
REVOLUTION[31..0].ena = COUNT24[] == 23 & COUNT15[] == 14;
REVOLUTION[31..0].clk = global(CLOCK_IN);
REVOLUTION[31..0].clrn = !REV_RESET;
REVOLUTION[31..0] = REVOLUTION[31..0] + 1;
REV_RESET = !/SYSRST # (MODULE & (AL[6..0] == h"48") & SR[1] & !/WRITE);

%-----------------------------------------------------------------------------%
%EVENT_CODE 256 x 8 SRAM translation table. Programmed on EPLD initialization
 for contents == address by file SRAM.mif (no trigger code translation).
 SRAM.address and SRAM.q shown in EVENT_CODE logic%
%Write data from VMEbus - SRAM.address[] is AL[7..0]%
SRAM.we = MODULE & (AL[10..8] == 1) & SR[1] & !/WRITE;

%Multiplex VMEbus odd-even byte data into SRAM%
if AL[0] == 0 then SRAM.data[7..0] = D[15..8];
   else SRAM.data[7..0] = D[7..0];
end if;

%-----------------------------------------------------------------------------%
%Acknowledge a VMEbus read/write cycle%
DDTACK.d = MODULE;
DDTACK.clk = SR[0];
DDTACK.clrn = /SYSRST & (!/DS0 # !/DS1);

%VMEbus DTACK is a logic OR of read/write and interrupt acknowledgements%
/DTACK = !(DDTACK # IDTACK);

%-----------------------------------------------------------------------------%
%VMEbus interrupt logic%

%IRQX is set to request an interrupt if MISSED # CLOCKFAIL # INDEXFAIL%
IRQX.d = CR[1] & CR[0];
IRQX.clk = CHANGE;
IRQX.clrn = !IDTACK;

%LOCAL is set by any following DS0%
LOCAL.d = IRQX;
LOCAL.clk = !/DS0;
LOCAL.clrn = global(/SYSRST);

%Enable interrupt status/ID byte%
```

```
STATUS.d = LOCAL & (BA[3..1] == INT_LEV[2..0]);
STATUS.clk = INT_SR1;
STATUS.clrn = !/DS0;

%Interrupt cycle acknowledgement%
IDTACK.d = LOCAL & (BA[3..1] == INT_LEV[2..0]);
IDTACK.clk = INT_SR0;
IDTACK.clrn = !/DS0;

%IACKOUT is set if the interrupt is not local or not level%
IACKOUT.d = !LOCAL # (BA[3..1] != INT_LEV[2..0]);
IACKOUT.clk = INT_SR0;
IACKOUT.clrn = !/AS;
/IACKOUT = !IACKOUT;

%3-bit shift register, clocked by 28_MHz to develop VMEbus interrupt delays%
INT_SR2.d = !/IACKIN;
INT_SR[1..0].d = INT_SR[2..1].q;
INT_SR[2..0].clk = 14_MHz;
INT_SR[2..0].clrn = !/DS0;

INTERRUPT_LED[0].d = VCC;
INTERRUPT_LED[0].clk = CHANGE;
INTERRUPT_LED[0].clrn = !INTERRUPT_LED[1];
INTERRUPT_LED[1].d = INTERRUPT_LED[0];
INTERRUPT_LED[1].clk = DISPLAY[15];
INTERRUPT_LED[1].clrn = global(/SYSRST);

/INTERRUPT_LED = !(INTERRUPT_LED[0] # INTERRUPT_LED[1]);
%-----------------------------------------------------------------------------%
%32-bit T/S local VMEbus data bus for data, interrupt vector, or VME status/ID
 byte.  VME status/ID even byte = ".", 0x2E tri-state bus multiplexer, odd
 byte in 10K10 PROM. Odd data bus output is also controlled by the interrupt
 STATUS flip/flop%

%LWORD addresses%
TRI_DATA[31..16].in = REVOLUTION[31..16];

case (AL[10..0]) is %even byte addresses and LWORD%
 when b"00000xxxxxx" => TRI_DATA[15..8].in = (GND, GND, VCC, GND, VCC, VCC,
VCC, GND);
 when b"000010010xx" => TRI_DATA[15..8].in = REVOLUTION[15..8];
 when b"001xxxxxxxx" => TRI_DATA[15..8].in = SRAM.q[7..0];
 when others => TRI_DATA[15..8].in = 0;
end case;

case (STATUS, AL[10..0]) is %odd byte addresses and LWORD%
 when b"000000xxxxxx" => TRI_DATA[7..0].in = (PROM.q[7..0]);
 when b"000001000001" => TRI_DATA[7..0].in = (CR[7..6], GND, GND, GND, GND,
CR[1..0]);
 when b"000001000011" => TRI_DATA[7..0].in = INT_VEC[7..0];
 when b"000001000101" => TRI_DATA[7..0].in = (GND, GND, GND, GND, GND,
INT_LEV[2..0]);
 when b"000001000111" => TRI_DATA[7..0].in = (INP_INT_VEC[7..4], GND, GND,
GND, GND);
```

```
 when b"000001001001" => TRI_DATA[7..0].in = (GND, GND, GND, GND, GND, GND,
GND, FULL);
 when b"0000010011xx" => TRI_DATA[7..0].in = REVOLUTION[7..0];
 when b"0001xxxxxxxx" => TRI_DATA[7..0].in = SRAM.q[7..0];
 when b"1xxxxxxxxxxx" => TRI_DATA[7..0].in = INT_VEC[7..0];
 when others => TRI_DATA[15..8].in = 0;
end case;

%internal register connections to tri-state VMEbus data%
TRI_DATA[31..0].oe = STATUS # MODULE & /WRITE;
D[31..0] = TRI_DATA[31..0].out;

%-----------------------------------------------------------------------------%
%BEAM_CLOCK failure, CLOCKFAIL, CSR7, is determined by counter, CLOCK_FAIL[].
 The CLOCK_FAIL[] clock is the backup 56_MHz input. CLOCK_FAIL[] is reset by
 the 56 MHz BEAM_CLOCK which clocks a one-shot ER_OS[1..0], and generates a
 CLOCK_FAIL[] reset pulse. The one-shot, is used as BEAM_CLOCK could be
 "stuck" at one or zero level. If BEAM_CLOCK is lost, CLOCK_FAIL[] will begin
 to count. At CLOCK_FAIL[] == 7 (CLOCKFAIL), CSR7 is held set, and a VMEbus
 interrupt is requested. CLOCKFAIL is reset when the BEAM_CLOCK is restored.%
CLOCK_FAIL[].clk = global(28_MHz);
CLOCK_FAIL[].clrn = /SYSRST & !ER_OS1;
if CLOCK_FAIL[] == 7 then CLOCK_FAIL[] = 7;
 else CLOCK_FAIL[] = CLOCK_FAIL[] + 1;
end if;

CLOCKFAIL = CLOCK_FAIL[] == 7;

CLOCK_OUT = BEAM_CLOCK & !CLOCKFAIL # 28_MHz & CLOCKFAIL;

%LED indicator%
/BEAM_CLOCK = CLOCKFAIL;

%reset CLOCK_FAIL[] if BEAM_CLOCK is active%
ER_OS0.d = VCC;
ER_OS0.clk = BEAM_CLOCK;
ER_OS0.clrn = !ER_OS1;
ER_OS1.d = ER_OS0;
ER_OS1.clk = global(28_MHz);
ER_OS1.clrn = global(/SYSRST);

%-----------------------------------------------------------------------------%
%Generate a 14_MHz clock from the 28 MHz CLOCK_IN%
14_MHz.d = !14_MHz;
14_MHz.clk = global(28_MHz);
14_MHz.clrn = global(/SYSRST);

%-----------------------------------------------------------------------------%
%REVOLUTION_TICK failure, CSR6, is determined by counter TICK_FAIL[].
 The TICK_FAIL[] clock is the local index derived from CLOCK_IN.
 TICK_FAIL[] is reset by REVOLUTION_TICK which clocks a one-shot ER_OS[2..3],
 and generates an TICK_FAIL[] reset pulse. The one-shot, is used as the
 input could be "stuck" at one or zero level. If REVOLUTION_TICK is lost,
 TICK_FAIL[] will begin to count. At TICK_FAIL[] == 3 (INDEXFAIL), CSR6
 is held set, and a VMEbus interrupt is requested. However, as long as the
```

```
 BEAM_CLOCK continues, the clock dividers will continue to transmit the
 revolution event code.%
TICK_FAIL[].clk = COUNT15[3];
TICK_FAIL[].clrn = /SYSRST & !ER_OS3;

if TICK_FAIL[].q == 3 then TICK_FAIL[].d = 3;
 else TICK_FAIL[].d = TICK_FAIL[].q + 1;
end if;

INDEXFAIL = TICK_FAIL[] == 3;

%LED indicator%
/REVOLUTION_TICK = INDEXFAIL;

%reset TICK_FAIL[] if REVOLUTION_TICK is active%

ER_OS2.d = VCC;
ER_OS2.clk = REVOLUTION_TICK;
ER_OS2.clrn = !ER_OS3;
ER_OS3.d = ER_OS2;
ER_OS3.clk = global(28_MHz);
ER_OS3.clrn = global(/SYSRST);

%-----------------------------------------------------------------------------%
%THE REVOLUTION_TICK INPUT MUST BE ADJUSTED TO OCCUR AFTER THE BEAM_CLOCK%
%-----------------------------------------------------------------------------%
%When RF BEAM_CLOCK and REVOLUTION_TICK are present, without synchronization
 there could be two phases of the 14.076 MHz EVENT_LINK clock derived from
 BEAM_CLOCK. The beam synchronous encoder module EVENT_LINK output could be
 a half cell off as it is restarted. The REVOLUTION_TICK provides the
 synchronization. REVOLUTION_TICK is a pulse indicating the beam passage at
 the 4 o'clock wall current monitor. 28.152 MHz represents 360 RF buckets.%
%-----------------------------------------------------------------------------%
%Synchronize REVOLUTION_TICK to CLOCK_IN%
SYNC_OS[0].d = VCC;
SYNC_OS[0].clk = REVOLUTION_TICK;
SYNC_OS[0].clrn = !SYNC_OS[1];
SYNC_OS[1].d = SYNC_OS[0];
SYNC_OS[1].clk = global(CLOCK_IN);
SYNC_OS[1].clrn = global(/SYSRST);

%Synchronize beam synchronous event link to REVOLUTION_TICK.
 COUNT24[] generates a single event cycle.
 COUNT15[] generates 15 event cycles/revolution%
COUNT24[].clk = global(CLOCK_IN);
COUNT24[].clrn = global(/SYSRST);

if SYNC_OS[1] then COUNT24[] = 2;
 elsif COUNT24[] == 23 then COUNT24[] = 0;
 else COUNT24[] = COUNT24[] + 1;
end if;

COUNT15[].ena = COUNT24[] == 23;
COUNT15[].clk = global(CLOCK_IN);
COUNT15[].clrn = /SYSRST & !SYNC_OS[1];
```

```
if COUNT15[] == 14 then COUNT15[] = 0;
 else COUNT15[] = COUNT15[] + 1;
end if;

%-----------------------------------------------------------------------------%
%V101 input module bus:
 See INP_ADDR for the five MSB's of a V101 input module address.
 See INP_INT_VEC[7..4] for register containing the four MSB's of the V101
 input module interrupt vector.%
%V101 input module bus clock which drives the input module priority logic%

%EXT_EVENT[7..0] latches the highest priority V101 module trigger byte.
 EXT_EVENT[8] is the input data bus busy flag.%
EXT_EVENT[8..0].d = (VCC, TRIGGER[7..0]);
EXT_EVENT[8..0].clk = !/TAKE_DATA;
EXT_EVENT[8..0].clrn = !CLEAR_EXT_EVENT & /SYSRST;

%BUSY output starts the V101 input modules priority daisy chain. BUSY means
 that EXT_EVENT[] contains an event trigger, wait till it empties%
/BUSY = EXT_EVENT[8] == 0;

%If SRAM_ADDRESS[] is in the input bus range, reset EXT_EVENT[] register.%
CLEAR_EXT_EVENT = COUNT2421 & SRAM_ADDRESS[7..0] > 1
                            & SRAM_ADDRESS[7..0] < LAST_V101 + 1;
COUNT2421.ena = READY;
COUNT2421.d = COUNT24[] == 20;
COUNT2421.clk = global(CLOCK_IN);
COUNT2421.clrn = global(/SYSRST);

%Synchronize EXT_EVENT[8] flag to local clock%
EXTERNAL.ena = COUNT24[] == 23;
EXTERNAL.d = EXT_EVENT[8];
EXTERNAL.clk = global(CLOCK_IN);
EXTERNAL.clrn = global(/SYSRST);

%-----------------------------------------------------------------------------%
%Pass event trigger code through SRAM to translate into event code. See
 module registers for SRAM.we and SRAM.data[]. SRAM.address is VMEbus,
 derived from REVOLUTION_TICK (COUNT15[] == 0), EXT_EVENT[], or FIFO outputs.%
if MODULE then SRAM_ADDRESS[7..0] = AL[7..0];
   elsif COUNT15[] == 0 then SRAM_ADDRESS[7..0] = 1;
   elsif EXTERNAL then SRAM_ADDRESS[7..0] = EXT_EVENT[7..0];
   elsif FIFO_REG[8] then SRAM_ADDRESS[7..0] = FIFO_REG[7..0];
   else SRAM_ADDRESS[7..0] = 0;
end if;

SRAM.address[7..0] = SRAM_ADDRESS[7..0];

EVENT_CODE[].ena = COUNT24[] == 4;
EVENT_CODE[].d = SRAM.q[7..0];
EVENT_CODE[].clk = global(CLOCK_IN);
EVENT_CODE[].clrn = global(/SYSRST);

%READY to transmit if EVENT_CODE[] != 0 at COUNT24[] == 3%
READY.ena = COUNT24[] == 3;
```

```
READY.d = SRAM_ADDRESS[7..0] != 0;
READY.clk = global(CLOCK_IN);
READY.clrn = global(/SYSRST);

%Allow VMEbus access if EVENT_CODE[] == 0%
ENA_VME.d = (!ENA_VME & !READY & COUNT24[] == 4)
           # (ENA_VME & COUNT24[] != 16);
ENA_VME.clk = global(CLOCK_IN);
ENA_VME.clrn = global(/SYSRST);

%-----------------------------------------------------------------------------%
%light BEAM_LINK LED as event is transmitted%
EVENT_LED[0].d = VCC;
EVENT_LED[0].clk = READY;
EVENT_LED[0].clrn = !EVENT_LED[1];
EVENT_LED[1].d = EVENT_LED[0];
EVENT_LED[1].clk = DISPLAY[15];
EVENT_LED[1].clrn = global(/SYSRST);

/EVENT_LED = !(EVENT_LED[0] # EVENT_LED[1]);

%light V101 trigger LED as event is transmitted%
V101_EVENT_LED[0].d =  SRAM_ADDRESS[7..0] > 1
                     & SRAM_ADDRESS[7..0] < LAST_V101 + 1;
V101_EVENT_LED[0].clk = READY;
V101_EVENT_LED[0].clrn = !V101_EVENT_LED[1];
V101_EVENT_LED[1].d = V101_EVENT_LED[0];
V101_EVENT_LED[1].clk = DISPLAY[15];
V101_EVENT_LED[1].clrn = global(/SYSRST);

/V101_EVENT_LED = !(V101_EVENT_LED[0] # V101_EVENT_LED[1]);

%light FIFO trigger LED as event is transmitted%
FIFO_EVENT_LED[0].d =  SRAM_ADDRESS[7..0] > LAST_V101;
FIFO_EVENT_LED[0].clk = READY;
FIFO_EVENT_LED[0].clrn = !FIFO_EVENT_LED[1];
FIFO_EVENT_LED[1].d = FIFO_EVENT_LED[0];
FIFO_EVENT_LED[1].clk = DISPLAY[15];
FIFO_EVENT_LED[1].clrn = global(/SYSRST);

/FIFO_EVENT_LED = !(FIFO_EVENT_LED[0] # FIFO_EVENT_LED[1]);

%-----------------------------------------------------------------------------%
%EVENT_LINK generator%
case COUNT24[] is
when 0 => LINK = GND;                      %stop 1 second cell%
when 1 => LINK = VCC;                      %stop 2 first cell%
when 2 => LINK = GND;                      %stop 2 second cell%
when 3 => LINK = VCC;                      %start first cell%
when 4 => if !READY then LINK = GND;       %start second cell%
            else LINK = VCC;
          end if;
when 5 => if EVENT_LINK1 then LINK = GND; %seven first cell%
            else LINK = VCC;
          end if;
```

```
when 6 => if READY & (EVENT_LINK1 $ EVENT_CODE[7]) then LINK = VCC; %7seccell%
            else LINK = GND;
          end if;
when 7 => if EVENT_LINK1 then LINK = GND; %six first cell%
            else LINK = VCC;
          end if;
when 8 => if READY & (EVENT_LINK1 $ EVENT_CODE[6]) then LINK = VCC; %6seccell%
            else LINK = GND;
          end if;
when 9 => if EVENT_LINK1 then LINK = GND; %five first cell%
            else LINK = VCC;
          end if;
when 10 => if READY & (EVENT_LINK1 $ EVENT_CODE[5]) then LINK = VCC; %5secell%
            else LINK = GND;
          end if;
when 11 => if EVENT_LINK1 then LINK = GND; %four first cell%
            else LINK = VCC;
          end if;
when 12 => if READY & (EVENT_LINK1 $ EVENT_CODE[4]) then LINK = VCC; %4secell%
            else LINK = GND;
          end if;
when 13 => if EVENT_LINK1 then LINK = GND; %three first cell%
            else LINK = VCC;
          end if;
when 14 => if READY & (EVENT_LINK1 $ EVENT_CODE[3]) then LINK = VCC; %3secell%
            else LINK = GND;
          end if;
when 15 => if EVENT_LINK1 then LINK = GND; %two first cell%
            else LINK = VCC;
          end if;
when 16 => if READY & (EVENT_LINK1 $ EVENT_CODE[2]) then LINK = VCC; %2secell%
            else LINK = GND;
          end if;
when 17 => if EVENT_LINK1 then LINK = GND; %one first cell%
            else LINK = VCC;
          end if;
when 18 => if READY & (EVENT_LINK1 $ EVENT_CODE[1]) then LINK = VCC; %1secell%
            else LINK = GND;
          end if;
when 19 => if EVENT_LINK1 then LINK = GND; %zero first cell%
            else LINK = VCC;
          end if;
when 20 => if READY & (EVENT_LINK1 $ EVENT_CODE[0]) then LINK = VCC; %0secell%
            else LINK = GND;
          end if;
when 21 => if EVENT_LINK1 then LINK = GND; %parity first cell%
            else LINK = VCC;
          end if;
when 22 => LINK = GND;                      %parity second cell%
when 23 => LINK = VCC;                      %stop 1 first cell%
end case;

%Three EVENT_LINK outputs to drive two ECL & one TTL buffers%
EVENT_LINK[3..1].d = LINK;
EVENT_LINK[3..1].clk = global(CLOCK_IN);
```

```
EVENT_LINK[3..1].clrn = global(/SYSRST);

end;
```