Fermi National Accelerator Laboratory

SVX II Silicon Strip Detector Upgrade Project Readout Electronics

Programmable Logic ABEL files of the FIB SUBRACK FANOUT MODULE (FFO) Revision B

November 15, 1999

John Anderson, Kerry Woodbury, John Chramowicz, Stefano Rapisarda

Document # ESE-SVX-990908

```
"==================================================
" FIB FANOUT CLOCK CONTROLLER
" ABEL MODULE CLK_MUX
"
" MASTER STATE MACHINE
"==================================================

MODULE clk_mux

TITLE 'Clock Multiplexer for FIB Fanout'


"inputs to this block

GLINK_IN_LOCK pin;
LOCAL_CLOCK pin;
RUN_REQ pin;
SLAVE_DONE pin;
LOCAL_SYNC pin;
POWER_ON_RESET pin;


"outputs from this block

SET_MCLK_LOCAL pin;
SET_SYNC_LOCAL pin;
RUN_REQ_ACCEPT pin;
ENABLE_SLAVE pin;
DISABLE_SYNC pin;
MSTR_J3_DISABLE pin;

MX3,MX2,MX1,MX0 pin;

"internal nodes - in this block only

mstr3,mstr2,mstr1,mstr0 node;
mstr3,mstr2,mstr1,mstr0 istype 'reg,buffer';
MASTER = [mstr3,mstr2,mstr1,mstr0];
MWATCH = [MX3,MX2,MX1,MX0];


equations
"
"       Necessary clock junk
"
MASTER.CLK = LOCAL_CLOCK;
MASTER.AR = !POWER_ON_RESET;
"
"       For analyzer use only.
"
MWATCH = MASTER;
"
"       Clock control section.
"
"       The MCLK and SYNC start out selecting the internal
"       53 MHz oscillator and the LOCAL_SYNC.  A state machine
"       allows them to progress to the GLINK selection
"       (GLINK clock and SYNC) if the GLINK is locked, and there
"       is a RUN REQUEST from VME or from the SRC.
"
"       In order to insure that the switchover occurs in SYNC
"       with the GLINK's clock, which will never be in phase with
"       the local oscillator, the MCLK and SYNC are first frozen,
"       then a subsidiary state machine which is run off the GLINK
"       clock (SLAVE) is used to move control over to the GLINK.
"       The main state machine (MASTER) then waits for the
"       GLINK-controlled state machine (SLAVE) to finish, which
"       then tells the 'master' to start switching back to the local.
"
"
"       'Master' state machine
```

```
"
STATE_DIAGRAM MASTER

"
"       State 0: wait for RUN_REQUEST to be asserted, either by
"       vme or by SRC.  Check for the presence of GLINK IN LOCK.
"       If RUN_REQUEST & GLINK IN LOCK go to state 1.
"
state 0:  ENABLE_SLAVE = 0;
          SET_MCLK_LOCAL = 1;
          MSTR_J3_DISABLE = 0;
          SET_SYNC_LOCAL = 1;
          DISABLE_SYNC = 0;
           RUN_REQ_ACCEPT = 0;
          if (RUN_REQ == 1)&(GLINK_IN_LOCK == 1) then 1 else 0;
"
"
"       state 1: RUN_REQ is in the RUN mode, meaning that either
"       the SRC or VME wants the GLINK to drive things.  Before we
"       switch, wait for SYNC to be a one.
"
state 1: ENABLE_SLAVE = 0;
          SET_MCLK_LOCAL = 1;
          SET_SYNC_LOCAL = 0;
          MSTR_J3_DISABLE = 1;
          DISABLE_SYNC = 0;
          RUN_REQ_ACCEPT = 0;
          if (GLINK_IN_LOCK == 0) then 0
          else if (LOCAL_SYNC == 0) then 1
          else 2;
"
"       State 2: sync now one, so wait for a zero, then it can be safely disabled.
"
state 2: ENABLE_SLAVE = 0;
          SET_MCLK_LOCAL = 1;
          SET_SYNC_LOCAL = 1;
          MSTR_J3_DISABLE = 0;
          DISABLE_SYNC = 0;
          RUN_REQ_ACCEPT = 0;
          if (GLINK_IN_LOCK==0) then 0
          else if (LOCAL_SYNC == 1) then 2
          else 3;
"
"       State 3: It was a SYNC falling edge, so disable generator and J3 drivers.
"
"
state 3: ENABLE_SLAVE = 0;
           SET_MCLK_LOCAL = 1;
           SET_SYNC_LOCAL = 1;
           MSTR_J3_DISABLE = 1;
           DISABLE_SYNC = 1;
           RUN_REQ_ACCEPT = 0;
          IF (GLINK_IN_LOCK) then 4 else 3;
"
"       State 4: through state 10: wait a full sync cycle (7 ticks) to insure
"       the FIB doesn't see glitches.  After that, enable the
"       slave machine to run and wait for it to finish.

state 4: ENABLE_SLAVE = 0;
           SET_MCLK_LOCAL = 0;
           SET_SYNC_LOCAL = 0;
          MSTR_J3_DISABLE = 1;
           DISABLE_SYNC = 1;
           RUN_REQ_ACCEPT = 1;
           goto 5;

state 5: ENABLE_SLAVE = 0;
           SET_MCLK_LOCAL = 0;
           SET_SYNC_LOCAL = 0;
          MSTR_J3_DISABLE = 1;
           DISABLE_SYNC = 1;
```

```
           RUN_REQ_ACCEPT = 0;
           goto 6;

state 6: ENABLE_SLAVE = 0;
           SET_MCLK_LOCAL = 0;
           SET_SYNC_LOCAL = 0;
          MSTR_J3_DISABLE = 1;
           DISABLE_SYNC = 1;
           RUN_REQ_ACCEPT = 0;
           goto 7;

state 7: ENABLE_SLAVE = 0;
           SET_MCLK_LOCAL = 0;
           SET_SYNC_LOCAL = 0;
          MSTR_J3_DISABLE = 1;
           DISABLE_SYNC = 1;
           RUN_REQ_ACCEPT = 0;
           goto 8;

state 8: ENABLE_SLAVE = 0;
           SET_MCLK_LOCAL = 0;
           SET_SYNC_LOCAL = 0;
          MSTR_J3_DISABLE = 1;
           DISABLE_SYNC = 1;
           RUN_REQ_ACCEPT = 0;
           goto 9;

state 9: ENABLE_SLAVE = 0;
           SET_MCLK_LOCAL = 0;
           SET_SYNC_LOCAL = 0;
          MSTR_J3_DISABLE = 1;
           DISABLE_SYNC = 1;
           RUN_REQ_ACCEPT = 0;
           goto 10;

state 10: ENABLE_SLAVE = 0;
           SET_MCLK_LOCAL = 0;
           SET_SYNC_LOCAL = 0;
          MSTR_J3_DISABLE = 1;
           DISABLE_SYNC = 1;
           RUN_REQ_ACCEPT = 0;
           goto 11;
"
"       State 11
"       Turn on slave machine, wait for it to finish.
"       The slave machine will finish in one of the following cases:
"
"       a) GLINK fails.
"       b) SRC sets RUN/INIT back to INIT.
"       c) VME request INIT mode.
"
state 11:
          ENABLE_SLAVE = 1;
           SET_MCLK_LOCAL = 0;
           SET_SYNC_LOCAL = 0;
          MSTR_J3_DISABLE = 1;
           DISABLE_SYNC = 1;
           RUN_REQ_ACCEPT = 0;
           if (SLAVE_DONE == 1) then 12 else 11;

"
"       Once slave is done, wait again then re-enable local sync generator.  At this point
"       the MCLK and SYNC are still coming from the GLINK, so we can re-enable the internal
"       sync generator.  The generator will re-start with sync off, so can be gated out
"       immediately.  However, to be really conservative, we wait for an up-and-down of the
"       local sync anyway and do not re-enable the local until then.
"
"
"       Wait for the sync to be high, then proceed.
"
state 12: ENABLE_SLAVE = 0;
```

```
           SET_MCLK_LOCAL = 0;  "do not switch clock yet
           SET_SYNC_LOCAL = 0;  " do not switch sync yet
          MSTR_J3_DISABLE = 1;
           DISABLE_SYNC = 0;
           RUN_REQ_ACCEPT = 0;
           if (LOCAL_SYNC == 0) then 12 else 13;
"
"       Now wait for the sync to be high
"
state 13: ENABLE_SLAVE = 0;
           SET_MCLK_LOCAL = 0;  " do not switch clock yet
           SET_SYNC_LOCAL = 0;  " do not switch sync yet
          MSTR_J3_DISABLE = 1;
           DISABLE_SYNC = 0;
           RUN_REQ_ACCEPT = 0;
           if (LOCAL_SYNC == 1) then 13 else 14;
"
"       Now re-enable the local sync generator and the drivers.
"

state 14: ENABLE_SLAVE = 0;
           SET_MCLK_LOCAL = 1;
           SET_SYNC_LOCAL = 1;
           DISABLE_SYNC = 0;
          MSTR_J3_DISABLE = 0;
           RUN_REQ_ACCEPT = 0;
           goto 15;

state 15: ENABLE_SLAVE = 0;
           SET_MCLK_LOCAL = 1;
           SET_SYNC_LOCAL = 1;
           DISABLE_SYNC = 0;
          MSTR_J3_DISABLE = 0;
           RUN_REQ_ACCEPT = 0;
           goto 0;

END
```

```
"=================================================
" FIB FANOUT CLOCK CONTROLLER
" ABEL MODULE FAKESYNC
"
" LOCAL SYNC GENERATOR
"=================================================


MODULE fakesync

TITLE 'Local Sync generator for FIB Fanout'


"inputs

LOCAL_CLOCK pin;
DISABLE_SYNC pin;


" outputs

LOCAL_SYNC pin istype 'reg,buffer';


FS7,FS6,FS5,FS4,FS3,FS2,FS1 node istype 'reg,buffer';

EQUATIONS

"  Local sync generator used to create SYNC pulse when
"  Fanout is in INIT mode.
"
FS7.D = FS6.Q;
FS7.CLK = LOCAL_CLOCK;
FS7.RE = DISABLE_SYNC;

FS6.D = FS5.Q;
FS6.CLK = LOCAL_CLOCK;
FS6.RE = DISABLE_SYNC;

FS5.D = FS4;
FS5.CLK = LOCAL_CLOCK;
FS5.RE = DISABLE_SYNC;

FS4.D = FS3.Q;
FS4.CLK = LOCAL_CLOCK;
FS4.RE = DISABLE_SYNC;

FS3.D = FS2.Q;
FS3.CLK = LOCAL_CLOCK;
FS3.RE = DISABLE_SYNC;

FS2.D = FS1.Q;
FS2.CLK = LOCAL_CLOCK;
FS2.RE = DISABLE_SYNC;

FS1.D = !FS7;
FS1.CLK = LOCAL_CLOCK;
FS1.RE = DISABLE_SYNC;

LOCAL_SYNC.D = (!FS7.Q & FS5.Q) # (FS7.Q & !FS5.Q);
LOCAL_SYNC.CLK = LOCAL_CLOCK;


END
```

```
"=====================================================
" FIB FANOUT CLOCK CONTROLLER
" ABEL MODULE RUNMODE
"
" RUN STATE MACHINE
"=====================================================


MODULE runmode

TITLE 'run mode control'

SRC_RUN_REQ pin;
WRT_0020 pin;
DB30 pin;
RUN_REQ_ACCEPT pin;
RUN_MODE pin;
LOCAL_CLOCK pin;
POWER_ON_RESET pin;
CMD_3D pin;
GLINK_FAIL pin;

"
"
AUTO_RESET_GLINK pin;

RM2,RM1,RM0 pin istype 'reg,buffer';

RUN_STATE = [RM2,RM1,RM0];

EQUATIONS

RUN_STATE.CLK = LOCAL_CLOCK;
RUN_STATE.AR = !POWER_ON_RESET;

STATE_DIAGRAM RUN_STATE
"
"       State 0: Idle state.  If SRC_RUN_REQ is high and
"       RUN_REQ_ACCEPT is high (master machine responded
"       to SRC run request) go to State 1.
"       If master responds but SRC_RUN_REQ is low,
"       go to state 4 (VME requested).
"
state 0:
        AUTO_RESET_GLINK = 0;
        RUN_MODE = 0;
        if (RUN_REQ_ACCEPT & SRC_RUN_REQ) then 1
        else if (RUN_REQ_ACCEPT & !SRC_RUN_REQ) then 4
        else goto 0;

"
"       state 1: entered run mode because of SRC_RUN_REQ.
"       Will exit if:
"       a) VME sets bit 30 in diagnostic register;
"       b) SRC sends command #0x3D over GLINK.
"       c) G-LINK lost lock.
"
state 1:
        AUTO_RESET_GLINK = 0;
        RUN_MODE = 1;
        if (POWER_ON_RESET == 0) then 0
        else if (!CMD_3D) then 2
        else if (!WRT_0020 & DB30) then 3
        else if (GLINK_FAIL) then 5
        else 1;
"
"       State 2: leaving run mode because of SRC command.
"
state 2:
        AUTO_RESET_GLINK = 0;
        RUN_MODE = 0;
```

```
        goto 0;

"
"       State 3: leaving run mode because VME asked us to.
"
state 3:
        AUTO_RESET_GLINK = 0;
        RUN_MODE = 0;
        goto 0;

"
"       State 4: entering RUN mode because of VME request.
"       Because VME asked us to run, only VME can ask
"       to leave. Run mode can be left also because of
"       detected GLINK error.
"       Reset the GLINK on the way out (throught state 5)
"
state 4:
        AUTO_RESET_GLINK = 0;
        RUN_MODE = 1;
        if (GLINK_FAIL) then 5
        else if (!WRT_0020 & DB30) then 0
        else 4;
"
"       state 5: leaving run mode because of detected GLINK error.
"       Reset the GLINK on the way out.
"
state 5:
        AUTO_RESET_GLINK = 1;
        RUN_MODE = 0;
        goto 6;
state 6:
        AUTO_RESET_GLINK = 1;
        RUN_MODE = 0;
        goto 0;

"
"       Unused state
"
state 7:
        AUTO_RESET_GLINK = 0;
        RUN_MODE = 0;
        goto 0;

END
```

```
"=================================================
" FIB FANOUT CLOCK CONTROLLER
" ABEL MODULE SLVMACH
"
" SLAVE STATE MACHINE
"=================================================
"
"    This machine is clocked by the GLINK clock.
"    It remains in state 0 until the ENABLE_SLAVE
"    input goes high, indicative that the MASTER
"    machine wants the clock and sync to start
"    coming from the GLINK, not the local oscillator.
"    At the point of entry the local sync generator
"    has been disabled so there is no sync pulse going
"    to the FIB, but the MCLK is still running from
"    the local oscillator.
"
"    When this machine is enabled it first checks for
"    the presence of the GLINK lock.  If the GLINK
"    is there, it then synchronizes itself to the SYNC
"    pulse from the GLINK, then switches the MCLK and
"    SYNC drivers over to the GLINK synchronous with
"    the falling edge of the sync from the GLINK.
"
"    The machine then persists in this state until
"    RUN_INIT goes back to the INIT mode.
"    If this happens, this machine waits for a
"    falling edge of the GLINK SYNC, then returns
"    back to the master machine.
"
MODULE slvmach

TITLE 'Slave machine for sync/clk control.'

"inputs

GLINK_IN_LOCK pin;   "signal indicating GLINK locked.
GLDB19 pin;
RUN_MODE pin;
ENABLE_SLAVE pin;
GLINK_STB pin;
SLV_J3_DISABLE pin;
POWER_ON_RESET pin;
DAV_T pin;

"outputs

SLAVE_DONE pin;
SET_MCLK_GLINK pin;
SET_SYNC_GLINK pin;
slv3,slv2,slv1,slv0 node;
slv3,slv2,slv1,slv0 istype 'reg,buffer';
SX3,SX2,SX1,SX0 pin;

"Macros

SLAVE = [slv3,slv2,slv1,slv0];
SX = [SX3,SX2,SX1,SX0];

EQUATIONS

SLAVE.CLK = GLINK_STB;
SLAVE.AR = !POWER_ON_RESET;
SX = SLAVE;

STATE_DIAGRAM SLAVE


"
"       Wait in state 0 until you see ENABLE_SLAVE.
"       If we're in RUN mode, start progressing
```

```
"       towards a GLINK clock.  If not in RUN mode,
"       there's an internal error in the logic,
"       so hang up via state 14.
"
"
state 0:      SLAVE_DONE = 0;
              SLV_J3_DISABLE = 1;
              SET_MCLK_GLINK = 0;
              SET_SYNC_GLINK = 0;

              if (RUN_MODE & ENABLE_SLAVE & GLINK_IN_LOCK) then 1
        else  if (!RUN_MODE & ENABLE_SLAVE) then 14;
        else  goto 0;

"
"       When machine enters state 1, SYNC & MCLK are already off
"       from the actions of the master machine.  We need wait for
"       a falling edge of the GLINK's SYNC, and then turn on the
"       buffers to drive the GLINK data.
"
"       First wait for GLDB19 to be a one, as qualified by DAV_T == 0.
"       DAV_T is the TTL copy of the GLINKs DAV* signal, indicating
"       'data available' if a zero.
"
state 1:
              SLAVE_DONE = 0;
              SLV_J3_DISABLE = 1;
              SET_MCLK_GLINK = 0;
              SET_SYNC_GLINK = 0;
        if (GLINK_IN_LOCK == 0) then 0
        else if ((DAV_T == 0) & (GLDB19 == 1)) then 2
        else 1;

"
"       State 2: Wait for GLDB19 to be zero.
"
state 2:
              SLAVE_DONE = 0;
              SLV_J3_DISABLE = 1;
              SET_MCLK_GLINK = 0;
              SET_SYNC_GLINK = 0;
        if (GLINK_IN_LOCK == 0) then 0
        else if ((DAV_T == 0) & (GLDB19 == 0)) then 3
        else 2;
"
"       State 3: Now, coincident with the falling edge, turn on the
"       ECL drivers to the GLINK signals.  Wait here until one of
"       the legit exit conditions occurs: RUN_MODE goes to zero.

state 3:
              SLAVE_DONE = 0;
              SLV_J3_DISABLE = 0;
              SET_MCLK_GLINK = 1;
              SET_SYNC_GLINK = 1;
        if (RUN_MODE == 1) then 3 else 4;


state 4:
              SLAVE_DONE = 0;
              SLV_J3_DISABLE = 0;
              SET_MCLK_GLINK = 1;
              SET_SYNC_GLINK = 1;
          goto 5;

state 5:
              SLAVE_DONE = 0;
              SLV_J3_DISABLE = 1;
              SET_MCLK_GLINK = 0;
              SET_SYNC_GLINK = 0;
         goto 6;
```

```
state 6:
              SLAVE_DONE = 1;
              SLV_J3_DISABLE = 1;
              SET_MCLK_GLINK = 0;
              SET_SYNC_GLINK = 0;
         goto 7;

state 7:
              SLAVE_DONE = 1;
              SLV_J3_DISABLE = 1;
              SET_MCLK_GLINK = 0;
              SET_SYNC_GLINK = 0;
         goto 0;

"
"       Unused states.
"

state 8:
              SLV_J3_DISABLE = 1;
              SLAVE_DONE = 1;
              SET_MCLK_GLINK = 0;
              SET_SYNC_GLINK = 0;
         goto 0;
state 9:
              SLV_J3_DISABLE = 1;
              SLAVE_DONE = 1;
              SET_MCLK_GLINK = 0;
              SET_SYNC_GLINK = 0;
         goto 0;
state 10:
              SLV_J3_DISABLE = 1;
              SLAVE_DONE = 1;
              SET_MCLK_GLINK = 0;
              SET_SYNC_GLINK = 0;
         goto 0;
state 11:
              SLV_J3_DISABLE = 1;
              SLAVE_DONE = 1;
              SET_MCLK_GLINK = 0;
              SET_SYNC_GLINK = 0;
         goto 0;
state 12:
              SLV_J3_DISABLE = 1;
              SLAVE_DONE = 1;
              SET_MCLK_GLINK = 0;
              SET_SYNC_GLINK = 0;
         goto 0;
state 13:
              SLV_J3_DISABLE = 1;
              SLAVE_DONE = 1;
              SET_MCLK_GLINK = 0;
              SET_SYNC_GLINK = 0;
         goto 0;


"
"       End of unused states.
"

"
"       Error exit.  Assert SLAVE_DONE, do nothing else, then
"       loop back to state 0.  By the time you get to zero,
"       the master machine should have taken ENABLE_SLAVE away.
"
state 14:
              SLV_J3_DISABLE = 1;
                SLAVE_DONE = 1;
                SET_MCLK_GLINK = 0;
                SET_SYNC_GLINK = 0;
                goto 15;
```

```
state 15:
                SLAVE_DONE = 1;
              SLV_J3_DISABLE = 1;
                SET_MCLK_GLINK = 0;
                SET_SYNC_GLINK = 0;
                goto 0;

END
```

```
"==================================================
" FIB FANOUT CLOCK CONTROLLER
" ABEL MODULE ttl161
"
" Synchronous 4-bit binary up counter with
" asynchronous clear, parallel data load
" and 2 enables
"==================================================


MODULE ttl161

TITLE 'Synchronous 4-bit binary up counter with asynchronous clear, parallel data load and 2 ena
bles'

        D0..D3,EN1,EN2,CLK,LD,CD        PIN;
        Q0..Q3                          PIN ISTYPE 'buffer,reg';
        CAO                             PIN;

        EN      NODE;

        COUNT = [Q3..Q0];
        DIN   = [D3..D0];

EQUATIONS
        COUNT.CLK = CLK;
        COUNT.RE  = !CD;
        EN = EN1 & EN2;

        WHEN (!LD) THEN
                COUNT.D = DIN;
        ELSE WHEN (EN) THEN
                COUNT.D = COUNT + 1;
        ELSE
                COUNT.D = COUNT;

        CAO = EN2 & Q0 & Q1 & Q2 & Q3;
END
```

```
"==================================================
" FIB FANOUT ERROR PIPE
" ABEL MODULE errencd
"
" ERROR ENCODER
"==================================================

MODULE errencd
"
"  Priority encoder for 3-bit error code
"


SYNC_ERR,PD2_ERR,L1A_ERR,ADVP_ERR,CXQT_ERR,RDQ_ERR,BC_ERR pin;

ERROR2,ERROR1,ERROR0 pin istype 'com';

INDIV_ERR = [SYNC_ERR,PD2_ERR,L1A_ERR,ADVP_ERR,CXQT_ERR,RDQ_ERR,BC_ERR];
ERROR = [ERROR2,ERROR1,ERROR0];

X = .x.;


TRUTH_TABLE (INDIV_ERR -> ERROR)

[1,X,X,X,X,X,X] -> [1,1,1];
[0,1,X,X,X,X,X] -> [1,1,0];
[0,0,1,X,X,X,X] -> [1,0,1];
[0,0,0,1,X,X,X] -> [1,0,0];
[0,0,0,0,1,X,X] -> [0,1,1];
[0,0,0,0,0,1,X] -> [0,1,0];
[0,0,0,0,0,0,1] -> [0,0,1];
[0,0,0,0,0,0,0] -> [0,0,0];

END
```

```
"==================================================
" FIB FANOUT ERROR PIPE
" ABEL MODULE SYNC_CK
"
" SYNC ERROR DETECTOR
"==================================================

MODULE SYNC_CK
"
" SYNC ERROR DETECTION
"
"
"       Inputs
"
SYNC7,SYNC6,SYNC5,SYNC4,SYNC3,SYNC2,SYNC1 pin;

"
"       Outputs
"
SYNC_ERR pin istype 'com,buffer';

"
"       Macro groups for equation convenience.
"

SYNCGRP = [SYNC7,SYNC6,SYNC5,SYNC4,SYNC3,SYNC2,SYNC1];


EQUATIONS

"
"       Across seven SYNC bits only two should be set at any time.
"

!SYNC_ERR =   (SYNCGRP == [0,0,0,0,0,1,1])
             #(SYNCGRP == [0,0,0,0,1,1,0])
             #(SYNCGRP == [0,0,0,1,1,0,0])
             #(SYNCGRP == [0,0,1,1,0,0,0])
             #(SYNCGRP == [0,1,1,0,0,0,0])
             #(SYNCGRP == [1,1,0,0,0,0,0])
             #(SYNCGRP == [1,0,0,0,0,0,1]);

END
```

```
"=====================================================
" FIB FANOUT ERROR PIPE
" ABEL MODULE ttl138
"=====================================================


MODULE ttl138

TITLE '3-to-8-line decoder with enable'

        A2..A0,G1,G2A,G2B       PIN;
        Y0..Y7                  PIN;

EQUATIONS
        Y0 = !(G1 & !(G2A # G2B) & !A2 & !A1 & !A0);
        Y1 = !(G1 & !(G2A # G2B) & !A2 & !A1 & A0);
        Y2 = !(G1 & !(G2A # G2B) & !A2 & A1 & !A0);
        Y3 = !(G1 & !(G2A # G2B) & !A2 & A1 & A0);
        Y4 = !(G1 & !(G2A # G2B) & A2 & !A1 & !A0);
        Y5 = !(G1 & !(G2A # G2B) & A2 & !A1 & A0);
        Y6 = !(G1 & !(G2A # G2B) & A2 & A1 & !A0);
        Y7 = !(G1 & !(G2A # G2B) & A2 & A1 & A0);
END
```

```
"=====================================================
" FIB FANOUT ERROR PIPE
" ABEL MODULE ttl174
"=====================================================

MODULE ttl174

TITLE '6-bit D flip-flop with asynchronous clear'

        D0..D5,CLK,CD   PIN;
        Q0..Q5          PIN ISTYPE 'buffer,reg_D';

        DIN = [D5..D0];
        QO  = [Q5..Q0];

EQUATIONS
        QO.CLK = CLK;
        QO.RE  = !CD;
        QO.D   = DIN;
END
```

```
"=====================================================
" FIB FANOUT ERROR DECODER
" ABEL MODULE BIGLATCH
"=====================================================

MODULE biglatch

PLSI PROPERTY 'ISP ON';

"inputs

D0,D1,D2,D3,D4,D5,D6,D7,D8,D9,D10,D11,D12,D13,D14,D15 pin;
D16,D17,D18,D19,D20,D21,D22,D23,D24,D25,D26,D27,D28 pin;

REGCLK pin;
REGCLR pin;

"outputs

Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8,Q9,Q10,Q11,Q12,Q13,Q14,Q15 pin istype 'reg';
Q16,Q17,Q18,Q19,Q20,Q21,Q22,Q23,Q24,Q25,Q26,Q27,Q28 pin istype 'reg';

INPUT = [D0,D1,D2,D3,D4,D5,D6,D7,D8,D9,D10,D11,D12,D13,D14,D15,D16,D17,D18,D19,D20,D21,D22,D23,D
24,D25,D26,D27,D28];
OUTPUT = [Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8,Q9,Q10,Q11,Q12,Q13,Q14,Q15,Q16,Q17,Q18,Q19,Q20,Q21,Q22,Q23,
Q24,Q25,Q26,Q27,Q28];


EQUATIONS

Q0 := D0;
Q0.CLK = REGCLK;
Q0.RE = REGCLR;

Q1 := D1;
Q1.CLK = REGCLK;
Q1.RE = REGCLR;

Q2 := D2;
Q2.CLK = REGCLK;
Q2.RE = REGCLR;

Q3 := D3;
Q3.CLK = REGCLK;
Q3.RE = REGCLR;

Q4 := D4;
Q4.CLK = REGCLK;
Q4.RE = REGCLR;

Q5 := D5;
Q5.CLK = REGCLK;
Q5.RE = REGCLR;

Q6 := D6;
Q6.CLK = REGCLK;
Q6.RE = REGCLR;

Q7 := D7;
Q7.CLK = REGCLK;
Q7.RE = REGCLR;

Q8 := D8;
Q8.CLK = REGCLK;
Q8.RE = REGCLR;

Q9 := D9;
Q9.CLK = REGCLK;
Q9.RE = REGCLR;

Q10 := D10;
```

```
Q10.CLK = REGCLK;
Q10.RE = REGCLR;

Q11 := D11;
Q11.CLK = REGCLK;
Q11.RE = REGCLR;

Q12 := D12;
Q12.CLK = REGCLK;
Q12.RE = REGCLR;

Q13 := D13;
Q13.CLK = REGCLK;
Q13.RE = REGCLR;

Q14 := D14;
Q14.CLK = REGCLK;
Q14.RE = REGCLR;

Q15 := D15;
Q15.CLK = REGCLK;
Q15.RE = REGCLR;

Q16 := D16;
Q16.CLK = REGCLK;
Q16.RE = REGCLR;

Q17 := D17;
Q17.CLK = REGCLK;
Q17.RE = REGCLR;

Q18 := D18;
Q18.CLK = REGCLK;
Q18.RE = REGCLR;

Q19 := D19;
Q19.CLK = REGCLK;
Q19.RE = REGCLR;

Q20 := D20;
Q20.CLK = REGCLK;
Q20.RE = REGCLR;

Q21 := D21;
Q21.CLK = REGCLK;
Q21.RE = REGCLR;

Q22 := D22;
Q22.CLK = REGCLK;
Q22.RE = REGCLR;

Q23 := D23;
Q23.CLK = REGCLK;
Q23.RE = REGCLR;

Q24 := D24;
Q24.CLK = REGCLK;
Q24.RE = REGCLR;

Q25 := D25;
Q25.CLK = REGCLK;
Q25.RE = REGCLR;

Q26 := D26;
Q26.CLK = REGCLK;
Q26.RE = REGCLR;

Q27 := D27;
Q27.CLK = REGCLK;
Q27.RE = REGCLR;
```

```
Q28 := D28;
Q28.CLK = REGCLK;
Q28.RE = REGCLR;



END
```

```
"==================================================
" FIB FANOUT ERROR DECODER
" ABEL MODULE BIGMUX
"==================================================

MODULE bigmux

"inputs

D0,D1,D2,D3,D4,D5,D6,D7,D8,D9,D10,D11,D12,D13,D14,D15 pin;
D16,D17,D18,D19,D20,D21,D22,D23,D24,D25,D26,D27,D28,D29,D30,D31 pin;
E0,E1,E2,E3,E4,E5,E6,E7,E8,E9,E10,E11,E12,E13,E14,E15 pin;
E16,E17,E18,E19,E20,E21,E22,E23,E24,E25,E26,E27,E28,E29,E30,E31 pin;
F0,F1,F2,F3,F4,F5,F6,F7,F8,F9,F10,F11,F12,F13,F14,F15 pin;
F16,F17,F18,F19,F20,F21,F22,F23,F24,F25,F26,F27,F28,F29,F30,F31 pin;
G0,G1,G2,G3,G4,G5,G6,G7,G8,G9,G10,G11,G12,G13,G14,G15 pin;
G16,G17,G18,G19,G20,G21,G22,G23,G24,G25,G26,G27,G28,G29,G30,G31 pin;

SEL0,SEL1 pin;

"outputs

Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8,Q9,Q10,Q11,Q12,Q13,Q14,Q15 pin;
Q16,Q17,Q18,Q19,Q20,Q21,Q22,Q23,Q24,Q25,Q26,Q27,Q28,Q29,Q30,Q31 pin;

INPUTA = [D0,D1,D2,D3,D4,D5,D6,D7,D8,D9,D10,D11,D12,D13,D14,D15,D16,D17,D18,D19,D20,D21,D22,D23,
D24,D25,D26,D27,D28,D29,D30,D31];
INPUTB = [E0,E1,E2,E3,E4,E5,E6,E7,E8,E9,E10,E11,E12,E13,E14,E15,E16,E17,E18,E19,E20,E21,E22,E23,
E24,E25,E26,E27,E28,E29,E30,E31];
INPUTC = [F0,F1,F2,F3,F4,F5,F6,F7,F8,F9,F10,F11,F12,F13,F14,F15,F16,F17,F18,F19,F20,F21,F22,F23,
F24,F25,F26,F27,F28,F29,F30,F31];
INPUTD = [G0,G1,G2,G3,G4,G5,G6,G7,G8,G9,G10,G11,G12,G13,G14,G15,G16,G17,G18,G19,G20,G21,G22,G23,
G24,G25,G26,G27,G28,G29,G30,G31];
OUTPUT = [Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8,Q9,Q10,Q11,Q12,Q13,Q14,Q15,Q16,Q17,Q18,Q19,Q20,Q21,Q22,Q23,
Q24,Q25,Q26,Q27,Q28,Q29,Q30,Q31];

EQUATIONS

OUTPUT = (INPUTA & !SEL0 & !SEL1)
        #(INPUTB &  SEL0 & !SEL1)
        #(INPUTC & !SEL0 &  SEL1)
        #(INPUTD &  SEL0 &  SEL1);

END
```

```
"=====================================================
" FIB FANOUT ERROR LATCH
" ABEL MODULE LATCHX
"=====================================================
"
"     A latch which is clocked regularly, and once set,
"     stays set until explicitly cleared.
"
MODULE latchx

PLSI PROPERTY 'ISP ON';

"inputs
D,CLK,RESET pin;

"outputs
Q pin istype 'reg';

EQUATIONS

Q := D # Q;
Q.CLK = CLK;
Q.RE = !RESET;

END
```

```
"==================================================
" FIB FANOUT FIFO CONTROLLER
" ABEL MODULE EFIFO
"
" ERROR FIFO STATE MACHINE
"==================================================
MODULE efifo

TITLE 'U3 Error FIFO state control'

"
"       This module controls the error fifo.
"       Control will be provived to:
"       a) trap and store GLINK and TEST-FIFO data errors,
"       b) loading of the error fifo with the contents of
"          the test fifo
"
"       The error fifo will be read via VME.
"
"       The state machine has two forms of operation:
"       Normal mode (EF_TEST_MODE == 0) in which the FIFO
"       gets almost full (8185 data words) and clocks
"       forever until an error is detected or a VME trigger
"       is received, at which point the FIFO count 1000
"       words and stops. Then, the machine waits for the
"       FIFO be emptied or reset by VME.
"
"       Test mode (EF_TEST_MODE == 1) in which the ERROR
"       FIFO gets loaded with data from the TEST FIFO.
"       The ERROR FIFO start storing data as soon as the
"       TEST FIFO starts dumping data. The ERROR FIFO
"       will stops storing data when the Test FIFO stops
"       dumping and then the state machine will waits
"       until is reset by VME.
"

"inputs

EF_TEST_MODE pin; "Error FIFO TEST MODE
PIPE_CLK,RESET,EFAFF,EFEF pin;
TRIGGER pin;

"outputs

EFOE_EN,EF_RCLK_SEL,EF_WEN,EF_REN,EF_HAS_DATA,FORCE_RESET_EF pin;
RESET_LATCHES pin;

"DLYCNT         delay counter.
"FORCE_RESET_EF active low, reset the Error FIFO logic.
"               and the ERROR FIFO.
"EF_WEN         active low, ERROR FIFO write enable.
"EF_REN         active low, ERROR FIFO read enable.
"EFOE_EN        active low , ERROR FIFO output enable.
"EF_RCLK_SEL    select EFIFO Read CLOCK. 0:pipe_clk, 1:VME read clock.
"RESET_LATCHES  reset latches signal.
"EF_HAS_DATA    EF has data stored ready to be read by VME.

"nodes

S2,S1,S0 pin istype 'reg,buffer';       "state bits
DLY13,DLY12,DLY11,DLY10,DLY9,DLY8,DLY7,DLY6,DLY5,DLY4,DLY3,DLY2,DLY1,DLY0 node istype 'reg,buffe
r';


DLYCNT = [DLY13,DLY12,DLY11,DLY10,DLY9,DLY8,DLY7,DLY6,DLY5,DLY4,DLY3,DLY2,DLY1,DLY0];


EFIFO_SM = [S2,S1,S0];

EQUATIONS
```

```
EFIFO_SM.AR = !RESET;
EFIFO_SM.CLK = PIPE_CLK;

DLYCNT.AR = !RESET;
DLYCNT.CLK = PIPE_CLK;

STATE_DIAGRAM [S2,S1,S0]

"
"       State 0: initial state. The state machine force the error FIFO
"       logic and the ERROR FIFO to reset.
"
"



" State 0: Initialization state
" The EFIFO and the FIFO logic are forced to reset
"
"
state 0:
        DLYCNT := 0;
        FORCE_RESET_EF = 0;
        EF_WEN = 1;
        EF_REN = 1;
        EFOE_EN = 1;
        EF_RCLK_SEL = 0;
        RESET_LATCHES = 0;
        EF_HAS_DATA = 0;
        if (EF_TEST_MODE == 1) then 5
        else goto 1;

" State 1: Enable the write and start filling the Error FIFO. Wait till the
" Error FIFO is almost full (FIFO capacity - 7) and jump to state 2.
state 1:
        DLYCNT := 0;
        FORCE_RESET_EF = 1;
        EF_WEN = 0;
        EF_REN = 1;
        EFOE_EN = 1;
        EF_RCLK_SEL = 0;
        RESET_LATCHES = 0;
        EF_HAS_DATA = 0;
        if (EFAFF == 0) then 2
        else goto 1;

" State 2: Wait for a trigger
" Both FIFO read and write are enabled in order to
" keep the FIFO almost full.
state 2:
        DLYCNT := 0;
        FORCE_RESET_EF = 1;
        EF_WEN = 0;
        EF_REN = 0;
        EFOE_EN = 1;
        EF_RCLK_SEL = 0;
        RESET_LATCHES = 0;
        EF_HAS_DATA = 0;
        if (TRIGGER == 1) then 3
        else goto 2;

" A trigger has been received, the machine will count 1000 words
" and then jump to state 4 (stop).
" Both FIFO read and write are enabled in order to
" keep the FIFO almost full.
state 3:
        DLYCNT := DLYCNT + 1;
        FORCE_RESET_EF = 1;
        EF_WEN = 0;
        EF_REN = 0;
        EFOE_EN = 1;
```

```
        EF_RCLK_SEL = 0;
        RESET_LATCHES = 1;
        EF_HAS_DATA = 0;
        if (DLYCNT >= 1000) then 4
        else goto 3;

" The FIFO wait to be reset.
" The read clock is that coming from VME reads and
" the FIFO OE is enabled allowing the FIFO to be read by VME.
" FIFO_HAS_DATA is set to let the system know that there
" is a sequence of word stored.
" Trigger latches are reset.

state 4:
        DLYCNT := 0;
        FORCE_RESET_EF = 1;
        EF_WEN = 1;
        EF_REN = 1;
        EFOE_EN = 0;
        EF_RCLK_SEL = 1;
        RESET_LATCHES = 1;
        EF_HAS_DATA = 1;
        goto 4;

"========================================
"===============TEST MODE===============
"========================================
" Error FIFO Test Mode has been enabled.
" The state machine is waiting for a trigger
" (Test FIFO Dumping or VME Trigger)

state 5:
        DLYCNT := 0;
        FORCE_RESET_EF = 1;
        EF_WEN = 1;
        EF_REN = 1;
        EFOE_EN = 1;
        EF_RCLK_SEL = 0;
        RESET_LATCHES = 0;
        EF_HAS_DATA = 0;
        if (TRIGGER == 1) then 6
        else goto 5;

" Trigger has been received.
" The Error FIFO is write enabled.
" The machine will wait till the FIFO is almost full
" or the trigger signal is removed
" (if the TEST FIFO stop dumping)

state 6:
        DLYCNT := 0;
        FORCE_RESET_EF = 1;
        EF_WEN = 0;
        EF_REN = 1;
        EFOE_EN = 1;
        EF_RCLK_SEL = 0;
        RESET_LATCHES = 0;
        EF_HAS_DATA = 0;
        if (TRIGGER == 0) # (EFAFF == 0) then 7
        else goto 6;

" The FIFO wait to be reset.
" The read clock is that coming from VME reads and
" the FIFO OE is enabled allowing the FIFO to be read by VME.
" FIFO_HAS_DATA is set to let the system know that there
" is a sequence of word stored.
" Trigger latches are reset.

state 7:
        DLYCNT := 0;
        FORCE_RESET_EF = 1;
```

```
        EF_WEN = 1;
        EF_REN = 1;
        EFOE_EN = 0;
        EF_RCLK_SEL = 1;
        RESET_LATCHES = 1;
        EF_HAS_DATA = 0;
        goto 7;


END
```

```
"=====================================================
" FIB FANOUT LED CONTROLLER
" ABEL MODULE CONTROL
"=====================================================

MODULE CONTROL

TITLE 'FIB FANOUT BOARD LED CONTROL'

"
"This module will control the turning on and off of
"the LEDs on the Fanout. At power up, all leds will
"light for 3 clock ticks of the 10Hz clock.
"This circuit is constructed in such a way as
"the ISP chip will sink the led source current,
"i.e. active low outputs will turn the led on.
"

"inputs

POWER_ON_RESET,PULSE_CLK,LED_TRIGGER pin;

"outputs

LED_DRIVE pin;

"nodes

S1,S0 pin istype 'reg,buffer';  "state bits
LEDSTATE=[S1,S0];

EQUATIONS

LEDSTATE.AR = !POWER_ON_RESET # LED_TRIGGER;
"If reset or valid condition turn on led

LEDSTATE.CLK = PULSE_CLK;
"Onboard Generated 10Hz Clock


STATE_DIAGRAM LEDSTATE

state 0:
        LED_DRIVE = 0;
        if (LED_TRIGGER == 1) then 0
        else 1;
state 1:
        LED_DRIVE = 0;
        goto 2;
state 2:
        LED_DRIVE = 0;
        goto 3;
state 3:
        LED_DRIVE = 1;
        if (LED_TRIGGER == 1) then 0
        else 3;
END
```

```
"==================================================
" FIB FANOUT VME INTERFACE
" ABEL MODULE ADDRCOMP
"
" ADDRESS COMPARATOR
"==================================================

MODULE addrcomp

PLSI PROPERTY 'ISP ON';

"inputs

AD31,AD30,AD29,AD28,AD27,GA4,GA3,GA2,GA1,GA0,AS,LOCAL_CLOCK pin;

POWER_ON_RESET pin;

"outputs

ITS_ME pin istype 'reg,buffer';

"internal nodes

AS2,AS1 node istype 'reg,buffer';
MATCH31,MATCH30,MATCH29,MATCH28,MATCH27 node;


EQUATIONS



AS2.D = AS1.Q;
AS2.CLK = LOCAL_CLOCK;
AS2.AR = !POWER_ON_RESET;

AS1.D = AS;
AS1.CLK = LOCAL_CLOCK;
AS1.AR = !POWER_ON_RESET;

"
" Match equations are for typing convenience only and optimize out.
"
" Note that GA lines are inverted (active low) per VME64x spec.
"

MATCH31 = (AD31 & !GA4) # (!AD31 & GA4);
MATCH30 = (AD30 & !GA3) # (!AD30 & GA3);
MATCH29 = (AD29 & !GA2) # (!AD29 & GA2);
MATCH28 = (AD28 & !GA1) # (!AD28 & GA1);
MATCH27 = (AD27 & !GA0) # (!AD27 & GA0);


ITS_ME.D = MATCH27 & MATCH28 & MATCH29 & MATCH30 & MATCH31;
ITS_ME.CLK = !AS2;
ITS_ME.AR = AS;

END
```

```
"=====================================================
" FIB FANOUT VME INTERFACE
" ABEL MODULE TTL 161
"=====================================================

MODULE ttl161

TITLE 'Synchronous 4-bit binary up counter with asynchronous clear, parallel data load and 2 ena
bles'

        D0..D3,EN1,EN2,CLK,LD,CD        PIN;
        Q0..Q3                          PIN ISTYPE 'buffer,reg';
        CAO                             PIN;

        EN      NODE;

        COUNT = [Q3..Q0];
        DIN   = [D3..D0];

EQUATIONS
        COUNT.CLK = CLK;
        COUNT.RE  = !CD;
        EN = EN1 & EN2;

        WHEN (!LD) THEN
                COUNT.D = DIN;
        ELSE WHEN (EN) THEN
                COUNT.D = COUNT + 1;
        ELSE
                COUNT.D = COUNT;

        CAO = EN2 & Q0 & Q1 & Q2 & Q3;
END
```

```
"=====================================================
" FIB FANOUT VME INTERFACE
" ABEL MODULE GDXSEL
"=====================================================

MODULE gdxsel

RD_ID,RD_TRACK,RD_J3B pin;

GSEL0,GSEL1 pin istype 'com';

TRUTH_TABLE ([RD_ID,RD_TRACK,RD_J3B]->[GSEL0,GSEL1])


[1,1,0] -> [0,0];  "read of J3 backplane
[0,1,1] -> [1,0];  "read of ID register
[1,1,1] -> [1,1];  "read of anything else
[1,0,1] -> [0,1];  "read of tracking register


END
```

```
"==================================================
" FIB FANOUT VME INTERFACE
" ABEL MODULE LAMCOMP
"==================================================

MODULE lamcomp

"inputs

LAM0,LAM1,LAM2,LAM3,LAM4,LAM5 pin;

"output
AM_OK pin;

EQUATIONS

AM_OK = (!LAM5 & !LAM4 & LAM3 & !LAM2 & !LAM1 & LAM0)
      # (!LAM5 & !LAM4 & LAM3 & !LAM2 & LAM1 & !LAM0 );

END
```

```
"=====================================================
" FIB FANOUT VME INTERFACE
" ABEL MODULE TTL138
"=====================================================

MODULE ttl138

TITLE '3-to-8-line decoder with enable'

        A2..A0,G1,G2A,G2B       PIN;
        Y0..Y7                  PIN;

EQUATIONS
        Y0 = !(G1 & !(G2A # G2B) & !A2 & !A1 & !A0);
        Y1 = !(G1 & !(G2A # G2B) & !A2 & !A1 & A0);
        Y2 = !(G1 & !(G2A # G2B) & !A2 & A1 & !A0);
        Y3 = !(G1 & !(G2A # G2B) & !A2 & A1 & A0);
        Y4 = !(G1 & !(G2A # G2B) & A2 & !A1 & !A0);
        Y5 = !(G1 & !(G2A # G2B) & A2 & !A1 & A0);
        Y6 = !(G1 & !(G2A # G2B) & A2 & A1 & !A0);
        Y7 = !(G1 & !(G2A # G2B) & A2 & A1 & A0);
END
```

```
"=====================================================
" FIB FANOUT VME INTERFACE
" ABEL MODULE TTL 174
"=====================================================

MODULE ttl174

TITLE '6-bit D flip-flop with asynchronous clear'

        D0..D5,CLK,CD   PIN;
        Q0..Q5          PIN ISTYPE 'buffer,reg_D';

        DIN = [D5..D0];
        QO  = [Q5..Q0];

EQUATIONS
        QO.CLK = CLK;
        QO.RE  = !CD;
        QO.D   = DIN;
END
```

```
// ################################################################
//          Fanout revision B GDX part.
// ################################################################
//
// Basic functions:
//
//    1) Provide interface to data bus for bus blanking
//    2) Provide interface to data bus for ID register.       (reg address: ox0000)
//    3) Provide interface to data bus for serial # register. (reg address: ox0000)
//    4) Provide interface to logic analyzer connectors
//    5) Replace J3 Backplane monitor register.               (reg address: ox0018)
//    6) Provide programmable interconnect between other PLDs on board
//       (green wire function)
//
// ################################################################


DESIGN fangdx;

//#### Device Section ####
PART ispGDX160-7Q208;

PARAM SECURITY OFF;
PARAM PULLUP OFF;
PARAM SLOWSLEW OFF; //sr


//#### Set/Constant Section ####
SET datbus [dataA0..dataA31];
SET anabus [anlyzr0..anlyzr15];
//SET reroutebus [reroute0..reroute15];
SET GLD [GLDBUS0..GLDBUS19];

//#### Pin Section ####
BIDI datbus {B2..B33} PULLUP;
//OUTPUT datbus {B2..B33} PULLUP;
OUTPUT anabus {D0..D15} PULLUP;

BIDI [bit027en,bit89en,bitcmen] {A37,A33,C13};

//BIDI [bit03en,bit820en] {A37,A33};

BIDI [bitcm14,bitcm13,bitcm12,bitcm11,bitcm10] {C11,C12,C16,C14,C15};

//   various control inputs

INPUT GLD {A0..A19};                // J3 backplane data, for monitor register function.
INPUT [CAV_TTL,DAV_TTL,GLINK_LINKRDY,FF_TTL] {A20,A21,A22,A23}; // G-LINK Status Signals
INPUT [GLINK_ERROR,STAT1_TTL,STAT0_TTL] {A24,A25,A26}; // G-LINK Status Signals
INPUT [FIB_CMD_OVFLW,FIB_FLOW_CTL,FIB_TRUNC,FIB_DEAD_CHAN] {A27,A28,A29,A30};
       // from FIBs on J3 Open Collector signals
INPUT [agnd,bgnd,cgnd,dgnd] {A39,B39,C36,D39};// Grounds for fixed zeroes,to fix GDX bug.
BIDI [gnd_sel0] {A34}; // Ground for SEL0...

INPUT [GDX_S0,GDX_S1] {C34,C35};  // from VME Interface PLD
INPUT [RD_DIAG,RD_STAT,RD_0_4_18,RD_0_4_18ms] {C39,C31,C29,C38};
       // because of GDX limitations, signal RD_0_4_18 is brought
```

```
      // in on an oe pin and a mux select pin.
      // It's used for both purposes in the design.

BIDI [RD_DIAG_OR_STAT] {A31};

//
//   Fanout signals of interest, which may be routed out to the analyzer.
//
//    User gets their choice of 16 of these inputs routed out to the
//    analyzer header.
//
//      NOTE: any you don't use in the equations will generate a warning during compile.
//
INPUT [TF_S0,TF_S1,TF_S2,TF_S3] {C7,C8,C9,C10};
//state monitor bits for TFIFO state machine

INPUT [SRC_STAT0,SRC_STAT1,SRC_STAT2,SRC_STAT3]    {C0,C1,C2,C3};// status bits back to SRC

INPUT [SRC_RESET_REQ,SRC_REQ_INIT] {C5,C6};
// interesting SRC command decodes

INPUT [ADC_EOC] {C4};                                  // ADC end of conversion
INPUT [EF_WCLK,EFRCLK,EFREN2,EFEF] {C17,C18,C19,C20};          // error fifo bits
INPUT [TF_REN,TFWCLK,LOCAL_CLOCK,TFEF] {C21,C22,C23,C25};      // test fifo bits
INPUT [ANY_ERROR,ERROR2,ERROR1,ERROR0] {C26,C27,C28,C30};      // decoded protocol errors
INPUT [WRT_MAP0,WRT_MAP1,WRT_MAP2,WRT_MAP3,WRT_TFIFO,WRT_DIAG,WRT_NVRAM]
{D32,D33,D34,D35,D36,D37,D38};
//VME writes

INPUT [RD_MAP0,RD_MAP1,RD_MAP2,RD_EFIFO,WRT_STOP_LOOP,RD_NVRAM]
{A35,A36,A38,B35,B36,B37};
// VME reads

INPUT [AS,DS0,DS1] {C32,C33,C37}; //VME controls

INPUT [J3_DRV_EN,MCLK_SEL,SYNC_SEL] {B0,B1,B38};//clock control bits


//  this leaves 16 free bits, D16..D31, for reroutes.
//
//   These could also be routed out
//   to the analyzer outputs in addition to the fixed set of 52 above
//   by reprogramming the PLDs on the board to put the interesting signals
//   on the reroutebus pins.
//
//    You should also be able to route the data bus signals to the analyzer
//    by redefining the data bus signals as BIDI pins.
//

//   The reroute bus is allocated to the other PLDs on the board this way:
//
//   vme_int : 4 pins  (0,1,2,3)
//   clk_ctl : 4 pins  (4,5,6,7)
//   pipeline: 3 pins  (8,9,10)
//   fifoctl:  4 pins  (11,12,13,14)
//   errlat :  1 pin   (15)
//   errdec1:  0 pins
```

```
//   errdec2:  0 pins
//   ledctl1:  0 pins
//   ledctl2:  0 pins
//
//   TOTAL     16 pins

//BIDI reroutebus {D16..D31};

INPUT [reroute0] {D16};
INPUT [reroute1] {D17}; //USED
INPUT [reroute2] {D18};
INPUT [reroute3] {D19};
INPUT [reroute4] {D20}; //USED
INPUT [reroute5] {D21}; //USED
INPUT [reroute6] {D22};
INPUT [reroute7] {D23}; //USED
INPUT [reroute8] {D24};
INPUT [reroute9] {D25};
INPUT [reroute10] {D26};
INPUT [reroute11] {D27}; //USED
OUTPUT [reroute12] {D28}; //USED
OUTPUT [reroute13] {D29}; //USED
INPUT [reroute14] {D30};
OUTPUT [reroute15] {D31}; //USED

//INPUT reroutebus {D16..D31};
//  On the PC board, things are wired this way for the reroute bus:
//
// reroute4 from CLOCK_CTL ISP is used to provide the signal
// "not(GLINK_DAT_ENBL)" to GDX
// and is used for the generation of Status Register bit 11.
//
// reroute11 from FIFO_CTL ISP is used to provide the signal
// "not(EN_TEST_DATA)" to GDX
// and is used for the generation of Status Register bit 10.
//
// reroute 15 is used to send the signal "RESET_ERROR_LATCHES"
// to the ERROR_LATCH ISP.
// reroute 7 is used to receive the signal "RESET_ERROR_LATCHES"
// from the CLOCK_CTL ISP
//
// reroute 13 is used to send the signal "READ_EFIFO"
// to the FIFO_CTL ISP.
// reroute 1 is used to receive the signal "READ_EFIFO"
// from the VME_INT ISP

//#### Connection Section ####

BEGIN


//  bus blanking.  The GDX must drive all 32 bits of the bus
//  for reads of registers 0,4 or 0x18.  If reading register
//  0x20, must drive 0's on bits 8-20 & 3GND; if reading register
//  0x24, must drive 0's on bits 0-3 only.
//
//
```

```
//  The VME interface PLD provides some decode logic for us by
//  presenting signals GDX_S0 and GDX_S1.  They provide the master
//  mux select for the GDX databus outputs:
//
//    GDX_S0   GDX_S1  databus
//      0        0     read of 0x18, drive with J3 monitor data.
//      0        1     read of 0x0, drive with ID code.
//      1        0     read of some other register, drive bus blank value
//                      (only valid for reads of 0x20 or 0x24).
//      1        1     read of 0x4, drive with tracking number.



datbus.s0 = GDX_S0;
datbus.s1 = GDX_S1;

//
// m3 inputs correspond to group D: GDX_S0 = 0  GDX_S1 = 1.
// This corresponds to a read of the tracking number register.
//
// The serial number is a 32-bit value interpreted as eight-digit BCD.
//
// Example as shown is an ID of 00547893.

//dataA31.m3 = dgnd;
//dataA30.m3 = dgnd;
//dataA29.m3 = dgnd;
//dataA28.m3 = dgnd;

//dataA27.m3 = dgnd;
//dataA26.m3 = dgnd;
//dataA25.m3 = dgnd;
//dataA24.m3 = dgnd;

//dataA23.m3 = dgnd;
//dataA22.m3 = VCC;
//dataA21.m3 = dgnd;
//dataA20.m3 = VCC;

//dataA19.m3 = dgnd;
//dataA18.m3 = VCC;
//dataA17.m3 = dgnd;
//dataA16.m3 = dgnd;

//dataA15.m3 = dgnd;
//dataA14.m3 = VCC;
//dataA13.m3 = VCC;
//dataA12.m3 = VCC;

//dataA11.m3 = VCC;
//dataA10.m3 = dgnd;
//dataA9.m3 =  dgnd;
//dataA8.m3 =  dgnd;

//dataA7.m3 =  VCC;
//dataA6.m3 =  dgnd;
//dataA5.m3 =  dgnd;
```

```
//dataA4.m3 =  VCC;

//dataA3.m3 =  dgnd;
//dataA2.m3 =  dgnd;
//dataA1.m3 =  VCC;
//dataA0.m3 =  VCC;

// PREP ID# 00550316
dataA31.m3 = dgnd;
dataA30.m3 = dgnd;
dataA29.m3 = dgnd;
dataA28.m3 = dgnd;

dataA27.m3 = dgnd;
dataA26.m3 = dgnd;
dataA25.m3 = dgnd;
dataA24.m3 = dgnd;

dataA23.m3 = dgnd;
dataA22.m3 = VCC;
dataA21.m3 = dgnd;
dataA20.m3 = VCC;

dataA19.m3 = dgnd;
dataA18.m3 = VCC;
dataA17.m3 = dgnd;
dataA16.m3 = VCC;

dataA15.m3 = dgnd;
dataA14.m3 = dgnd;
dataA13.m3 = dgnd;
dataA12.m3 = dgnd;

dataA11.m3 = dgnd;
dataA10.m3 = dgnd;
dataA9.m3 =  VCC;
dataA8.m3 =  VCC;

dataA7.m3 =  dgnd;
dataA6.m3 =  dgnd;
dataA5.m3 =  dgnd;
dataA4.m3 =  VCC;

dataA3.m3 =  VCC;
dataA2.m3 =  dgnd;
dataA1.m3 =  dgnd;
dataA0.m3 =  dgnd;



//
//   The m1 select group corresponds to GDX_S0 = 1, GDX_S1 = 0.
//   This selects the board ID register, so the GDX must assert
//   the ID fields onto the data bus.


//
```

```
//   Bits 31-24 are the SVX Module ID code of the module.
//   The FIB Fanout is ID 03.
//
dataA31.m1 = bgnd;
dataA30.m1 = bgnd;
dataA29.m1 = bgnd;
dataA28.m1 = bgnd;
dataA27.m1 = bgnd;
dataA26.m1 = bgnd;
dataA25.m1 = VCC;
dataA24.m1 = VCC;

//
//    Bits 23-20 are the PCB layout revision number
//    This is revision 2 of the FIB Fanout.
//
dataA23.m1 = bgnd;
dataA22.m1 = bgnd;
dataA21.m1 = VCC;
dataA20.m1 = bgnd;


//
//     Bits 19-0 are the Hardware/Firmware revision date.
//     Bits 19-12 are used for the month (BCD). Ex. "ox01"
//                is January, "ox12" is December.
//     Bits 11-4 are used for the day (BCD). Ex. "ox01"
//               is 1, "ox31" is 31.
//     Bits 3-0 are used for the year.
//              Ex. "ox9" is 1999, "ox0" is 2000, "ox8" is 2008,.
//
// ***** MONTH ***** ## 11 ##
dataA19.m1 = bgnd;
dataA18.m1 = bgnd;
dataA17.m1 = bgnd;
dataA16.m1 = VCC;

dataA15.m1 = bgnd;
dataA14.m1 = bgnd;
dataA13.m1 = bgnd;
dataA12.m1 = VCC;

// ***** DAY *****  ## 11 ##
dataA11.m1 = bgnd;
dataA10.m1 = bgnd;
dataA9.m1 =  bgnd;
dataA8.m1 =  VCC;

dataA7.m1 =  bgnd;
dataA6.m1 =  bgnd;
dataA5.m1 =  bgnd;
dataA4.m1 =  VCC;

// ***** YEAR ***** ## 9 ##
dataA3.m1 =  VCC;
dataA2.m1 =  bgnd;
dataA1.m1 =  bgnd;
```

```
dataA0.m1 =  VCC;

//
//    The m2 inputs correspond to any read of any other register
//    within the Fanout (GDX_S0 = 1, GDX_S1 = 1)
//    For those registers, drive zeroes.
//    The various output enable controls will select which zeroes
//    are actually driven for bus blanking.

dataA31.m2 = cgnd;
dataA30.m2 = cgnd;
dataA29.m2 = cgnd;
dataA28.m2 = cgnd;
dataA27.m2 = cgnd;
dataA26.m2 = cgnd;
dataA25.m2 = cgnd;
dataA24.m2 = cgnd;
dataA23.m2 = cgnd;
dataA22.m2 = cgnd;
dataA21.m2 = cgnd;
dataA20.m2 = cgnd;
dataA19.m2 = cgnd;
dataA18.m2 = cgnd;
dataA17.m2 = cgnd;
dataA16.m2 = cgnd;
dataA15.m2 = cgnd;
dataA14.m2 = bitcm14;
dataA13.m2 = bitcm13;
dataA12.m2 = bitcm12;
dataA11.m2 = bitcm11;
dataA10.m2 = bitcm10;
dataA9.m2 =  cgnd;
dataA8.m2 =  cgnd;
dataA7.m2 =  cgnd;
dataA6.m2 =  cgnd;
dataA5.m2 =  cgnd;
dataA4.m2 =  cgnd;
dataA3.m2 =  cgnd;
dataA2.m2 =  cgnd;
dataA1.m2 =  cgnd;
dataA0.m2 =  cgnd;


//
//  If GDX_S0 = 0 and GDX_S1 = 0, the board is performing a read of the
//  J3 backplane monitor register.  In this case, route the J3 monitor
//  input pins (GLD bus) to the data bus.



dataA31.m0 = CAV_TTL;
dataA30.m0 = DAV_TTL;
dataA29.m0 = GLINK_LINKRDY;
dataA28.m0 = FF_TTL;
dataA27.m0 = GLINK_ERROR;
dataA26.m0 = STAT1_TTL;
dataA25.m0 = STAT0_TTL;
```

```
dataA24.m0 = FIB_CMD_OVFLW;
dataA23.m0 = FIB_FLOW_CTL;
dataA22.m0 = FIB_TRUNC;
dataA21.m0 = FIB_DEAD_CHAN;
dataA20.m0 = agnd;

dataA19.m0 = GLDBUS19;
dataA18.m0 = GLDBUS18;
dataA17.m0 = GLDBUS17;
dataA16.m0 = GLDBUS16;
dataA15.m0 = GLDBUS15;
dataA14.m0 = GLDBUS14;
dataA13.m0 = GLDBUS13;
dataA12.m0 = GLDBUS12;
dataA11.m0 = GLDBUS11;
dataA10.m0 = GLDBUS10;
dataA9.m0 =  GLDBUS9;
dataA8.m0 =  GLDBUS8;
dataA7.m0 =  GLDBUS7;
dataA6.m0 =  GLDBUS6;
dataA5.m0 =  GLDBUS5;
dataA4.m0 =  GLDBUS4;
dataA3.m0 =  GLDBUS3;
dataA2.m0 =  GLDBUS2;
dataA1.m0 =  GLDBUS1;
dataA0.m0 =  GLDBUS0;



//
//    Output Enable logic, which requires external decode.
//
//
//
//   bit027en is the output enable for bits 0-2 and 7.  These bits need
//   be enabled if reading registers 0,4,18 or 24.  The RD_0_4_18
//   input is low if any of those three registers are being read.
//   RD_STAT is low if register 0x24 is being read.  Data bits 0..2 and 7
//   are enabled onto the data bus if bit027en is high.


bit027en.m3 =  VCC;    // RD_0_4_18 low, RD_STAT high
bit027en.m2 =  cgnd;   // RD_0_4_18 high, RD_STAT high
//Change to disable/enable Masking
//bit027en.m1 =  bgnd; // RD_0_4_18 high, RD_STAT low //Masking DISABLED
bit027en.m1 =  VCC;    // RD_0_4_18 high, RD_STAT low //Masking ENABLED
bit027en.m0 =  VCC;    // RD_0_4_18 low, RD_STAT low
bit027en.s0 = RD_0_4_18ms;
bit027en.s1 = RD_STAT;
bit027en.oe = VCC;

//
//    Similarly, create an output enable for bits 8..9.
//


bit89en.m3 = VCC;
```

```
bit89en.m2 = cgnd;
//Change to disable/enable Masking
bit89en.m1 = bgnd;   //Masking DISABLED
//bit89en.m1 = VCC;  //Masking ENABLED
bit89en.m0 = VCC;
bit89en.s0 = RD_0_4_18ms;
bit89en.s1 = RD_DIAG;
bit89en.oe = VCC;


//
//    Similarly, create an output enable for bits 10..14.
//


bitcmen.m3 = VCC;
bitcmen.m2 = cgnd;
//Change to disable/enable Masking
//bitcmen.m1 = bgnd;   //Masking DISABLED
bitcmen.m1 = VCC;  //Masking ENABLED
bitcmen.m0 = VCC;
bitcmen.s0 = RD_0_4_18ms;
bitcmen.s1 = RD_DIAG_OR_STAT;
bitcmen.oe = VCC;

// Generation of the signal "RD_DIAG_OR_STAT" on pin A31
// Using the mux connected to pin A31 we are able to implement a logic AND
// between the two signals RD_DIAG and RD_STAT.
// The signal is then used for the
// select1 on the bitcmen mux few lines above.
RD_DIAG_OR_STAT.m3 = VCC;      //Not Used
RD_DIAG_OR_STAT.m2 = RD_DIAG;
//RDIAG is on pin C39. Selected when RD_STAT is high.

RD_DIAG_OR_STAT.m1 = bgnd;     //selected when RD_STAT is low.
RD_DIAG_OR_STAT.m0 = VCC;      //Not Used
RD_DIAG_OR_STAT.s0 = !gnd_sel0;
RD_DIAG_OR_STAT.s1 = RD_STAT;
RD_DIAG_OR_STAT.oe = VCC;


//
//   Now assign the oe of each data bit accordingly.
//

dataA31.oe = !RD_0_4_18;
dataA30.oe = !RD_0_4_18;
dataA29.oe = !RD_0_4_18;
dataA28.oe = !RD_0_4_18;
dataA27.oe = !RD_0_4_18;
dataA26.oe = !RD_0_4_18;
dataA25.oe = !RD_0_4_18;
dataA24.oe = !RD_0_4_18;
dataA23.oe = !RD_0_4_18;
dataA22.oe = !RD_0_4_18;
dataA21.oe = !RD_0_4_18;
dataA20.oe = !RD_0_4_18;
```

```
dataA19.oe = !RD_0_4_18;
dataA18.oe = !RD_0_4_18;
dataA17.oe = !RD_0_4_18;
dataA16.oe = !RD_0_4_18;
dataA15.oe = !RD_0_4_18;
dataA14.oe = bitcmen;
dataA13.oe = bitcmen;
dataA12.oe = bitcmen;
dataA11.oe = bitcmen;
dataA10.oe = bitcmen;
dataA9.oe =  bit89en;
dataA8.oe =  bit89en;
dataA7.oe =  bit027en;
dataA6.oe =  !RD_0_4_18;
dataA5.oe =  !RD_0_4_18;
dataA4.oe =  !RD_0_4_18;
dataA3.oe =  !RD_0_4_18;
dataA2.oe =  bit027en;
dataA1.oe =  bit027en;
dataA0.oe =  bit027en;

// Status and Diagnostic registers bits 14..10
//
//                    | Status Register    | Diagnostic Register |
//                    -------------------------------------------
//dataA14.m2 = bitcm14; | GLINK ERROR       |  GND                |
//dataA13.m2 = bitcm13; | /CAV              |  GND                |
//dataA12.m2 = bitcm12; | /DAV              |  GND                |
//dataA11.m2 = bitcm11; | GLINK DATA BUFFER  |  GND               |
//dataA10.m2 = bitcm10; | TEST FIFO BUFFER   |  GND               |
//                    -------------------------------------------
//
//
// bitcm14 is on C11
bitcm14.m3 = dgnd;  //Diagnostic Register, MASKING.
bitcm14.m2 = cgnd; //NOT USED
bitcm14.m1 = bgnd; //NOT USED
bitcm14.m0 = GLINK_ERROR;  //Status Register, GLINK_ERROR is on pin A24
bitcm14.s0 = gnd_sel0;
bitcm14.s1 = RD_STAT;
bitcm14.oe = VCC;

// bitcm13 is on C12
bitcm13.m3 = dgnd;  //Diagnostic Register, MASKING.
bitcm13.m2 = cgnd; //NOT USED
bitcm13.m1 = bgnd; //NOT USED
bitcm13.m0 = CAV_TTL;  //Status Register, CAV_TTL is on pin A20
bitcm13.s0 = gnd_sel0;
bitcm13.s1 = RD_STAT;
bitcm13.oe = VCC;

// bitcm12 is on C16
bitcm12.m3 = dgnd;  //Diagnostic Register, MASKING.
bitcm12.m2 = cgnd; //NOT USED
bitcm12.m1 = bgnd; //NOT USED
bitcm12.m0 = DAV_TTL;  //Status Register, DAV_TTL is on pin A21
bitcm12.s0 = gnd_sel0;
```

```
bitcm12.s1 = RD_STAT;
bitcm12.oe = VCC;

// bitcm11 is on C14
bitcm11.m3 = reroute4;  //Status Register, reroute4 "not(GLINK_DAT_ENBL)"
                        //from CLOCK_CTL is on pin D20
bitcm11.m2 = cgnd; //NOT USED
bitcm11.m1 = bgnd; //NOT USED
bitcm11.m0 = agnd;  //Diagnostic Register, MASKING.
bitcm11.s0 = gnd_sel0;
bitcm11.s1 = !RD_STAT;
bitcm11.oe = VCC;

// bitcm10 is on C15
bitcm10.m3 = reroute11;//Status Register, reroute11
                       // "not(EN_TEST_DATA)" from FIFO_CTL is on pin D27
bitcm10.m2 = cgnd; //NOT USED
bitcm10.m1 = bgnd; //NOT USED
bitcm10.m0 = agnd;  //Diagnostic Register, MASKING.
bitcm10.s0 = gnd_sel0;
bitcm10.s1 = !RD_STAT;
bitcm10.oe = VCC;

// Generation of the GND signal for the "sel0" muxes input...
// We are using the pin A34;
gnd_sel0.m3 = dgnd;
gnd_sel0.oe = VCC;

// Logic analyzer rewiring.  Route selected 16 of available
// inputs onto analyzer headers.

//   Designer note:  You can connect any signal to the analyzer outputs,
//   but you have to select the multiplexer settings appropriately.
//   Follow this table:
//
//   SIGNAL COMES FROM       USE MUX INPUT    S0 SETTING  S1 SETTING
//
//      An 'A' pin              .m0           gnd          gnd
//      A  'B' pin              .m1           VCC          gnd
//      A  'C' pin              .m2           VCC          VCC
//      A  'D' pin              .m3           gnd          VCC

//  The help indicates that if you only select one mux imput in the
//  equations it will derive the s0 and s1 for you.  You only need
//  explicitly set the s0 and s1 if you have multiple inputs to
//  the mux. However, if you don't explicitly set the s0 and s1 in
//  the equations it generates warnings.



anlyzr0.m0 = CAV_TTL;   //it's on A20
anlyzr0.s0 = GND;
anlyzr0.s1 = GND;

anlyzr1.m0 = DAV_TTL;   //it's on A21
anlyzr1.s0 = GND;
anlyzr1.s1 = GND;
```

```
anlyzr2.m0 = GLINK_LINKRDY;  //it's on A22
anlyzr2.s0 = GND;
anlyzr2.s1 = GND;

anlyzr3.m0 = STAT1_TTL;     //it's on A25
anlyzr3.s0 = GND;
anlyzr3.s1 = GND;

anlyzr4.m0 = STAT0_TTL;     //it's on A26
anlyzr4.s0 = GND;
anlyzr4.s1 = GND;

anlyzr5.m2 = ERROR0;        //it's on C30
anlyzr5.s0 = VCC;
anlyzr5.s1 = VCC;

anlyzr6.m2 = ERROR1;        //it's on C28
anlyzr6.s0 = VCC;
anlyzr6.s1 = VCC;

anlyzr7.m2 = ERROR2;        //it's on C27
anlyzr7.s0 = VCC;
anlyzr7.s1 = VCC;

anlyzr8.m2 = SRC_STAT0;     //it's on C0
anlyzr8.s0 = VCC;
anlyzr8.s1 = VCC;

anlyzr9.m2 = SRC_STAT1;     //it's on C1
anlyzr9.s0 = VCC;
anlyzr9.s1 = VCC;

anlyzr10.m2 = SRC_STAT2;     //it's on C2
anlyzr10.s0 = VCC;
anlyzr10.s1 = VCC;

anlyzr11.m2 = SRC_STAT3;     //it's on C3
anlyzr11.s0 = VCC;
anlyzr11.s1 = VCC;

anlyzr12.m0 = FIB_CMD_OVFLW;     //it's on A27
anlyzr12.s0 = GND;
anlyzr12.s1 = GND;

anlyzr13.m0 = FIB_FLOW_CTL;     //it's on A28
anlyzr13.s0 = GND;
anlyzr13.s1 = GND;

anlyzr14.m0 = FIB_TRUNC;     //it's on A29
anlyzr14.s0 = GND;
anlyzr14.s1 = GND;

anlyzr15.m0 = FIB_DEAD_CHAN;     //it's on A30
anlyzr15.s0 = GND;
anlyzr15.s1 = GND;
```

```
//   The following is just an example; wire up whatever you need.
//anlyzr15.m2 = TF_S3;   //it's on C10
//anlyzr15.s0 = GND;
//anlyzr15.s1 = VCC;
//
//anlyzr14.m2 = TF_S2;   //it's on C9
//anlyzr14.s0 = VCC;
//anlyzr14.s1 = VCC;
//
//anlyzr13.m2 = TF_S1;     //it's on C8
//anlyzr13.s0 = VCC;
//anlyzr13.s1 = VCC;
//
//anlyzr12.m2 = TF_S0;     //it's on C7
//anlyzr12.s0 = VCC;
//anlyzr12.s1 = VCC;
//
//anlyzr11.m3 = WRT_TFIFO; //it's on D36
//anlyzr11.s0 = GND;
//anlyzr11.s1 = VCC;

//anlyzr10.m2 = TF_REN;   //it's on C21
//anlyzr10.s0 = VCC;
//anlyzr10.s1 = VCC;

//anlyzr9.m2 = TFWCLK;   //it's on C22
//anlyzr9.s0 = VCC;
//anlyzr9.s1 = VCC;

//anlyzr8.m2 = TFEF;     //it's on C25
//anlyzr8.s0 = VCC;
//anlyzr8.s1 = VCC;

//anlyzr7.m3 = reroute3;     //it's on D19 (from VME Interface) VME_WRITE
//anlyzr7.s0 = GND;
//anlyzr7.s1 = VCC;

//anlyzr6.m3 = reroute2;     //it's on D18 (from VME Interface) WRITE_PULSE
//anlyzr6.s0 = GND;
//anlyzr6.s1 = VCC;

//anlyzr5.m3 = reroute1;     //it's on D17 (from VME Interface) G2B102
//anlyzr5.s0 = GND;
//anlyzr5.s1 = VCC;

//anlyzr4.m3 = reroute0;     //it's on D16 (from VME Interface) G2B101
//anlyzr4.s0 = GND;
//anlyzr4.s1 = VCC;

//anlyzr3.m1 = dataA3;
//anlyzr3.s0 = VCC;
//anlyzr3.s1 = GND;

//anlyzr2.m1 = dataA2;
//anlyzr2.s0 = VCC;
//anlyzr2.s1 = GND;
```

```
//anlyzr1.m1 = dataA1;
//anlyzr1.s0 = VCC;
//anlyzr1.s1 = GND;

//anlyzr0.m1 = dataA0;
//anlyzr0.s0 = VCC;
//anlyzr0.s1 = GND;


//###### The following setting is used to monitor
//      the lowest 16 bits of the Data Bus "DB"
//anlyzr15.m1 = dataA15;
//anlyzr15.s0 = VCC;
//anlyzr15.s1 = GND;

//anlyzr14.m1 = dataA14;
//anlyzr14.s0 = VCC;
//anlyzr14.s1 = GND;

//anlyzr13.m1 = dataA13;
//anlyzr13.s0 = VCC;
//anlyzr13.s1 = GND;

//anlyzr12.m1 = dataA12;
//anlyzr12.s0 = VCC;
//anlyzr12.s1 = GND;

//anlyzr11.m1 = dataA11;
//anlyzr11.s0 = VCC;
//anlyzr11.s1 = GND;

//anlyzr10.m1 = dataA10;
//anlyzr10.s0 = VCC;
//anlyzr10.s1 = GND;

//anlyzr9.m1 = dataA9;
//anlyzr9.s0 = VCC;
//anlyzr9.s1 = GND;

//anlyzr8.m1 = dataA8;
//anlyzr8.s0 = VCC;
//anlyzr8.s1 = GND;

//anlyzr7.m1 = dataA7;
//anlyzr7.s0 = VCC;
//anlyzr7.s1 = GND;

//anlyzr6.m1 = dataA6;
//anlyzr6.s0 = VCC;
//anlyzr6.s1 = GND;

//anlyzr5.m1 = dataA5;
//anlyzr5.s0 = VCC;
//anlyzr5.s1 = GND;

//anlyzr4.m1 = dataA4;
//anlyzr4.s0 = VCC;
```

```
//anlyzr4.s1 = GND;

//anlyzr3.m1 = dataA3;
//anlyzr3.s0 = VCC;
//anlyzr3.s1 = GND;

//anlyzr2.m1 = dataA2;
//anlyzr2.s0 = VCC;
//anlyzr2.s1 = GND;

//anlyzr1.m1 = dataA1;
//anlyzr1.s0 = VCC;
//anlyzr1.s1 = GND;

//anlyzr0.m1 = dataA0;
//anlyzr0.s0 = VCC;
//anlyzr0.s1 = GND;


//
//   PLD rewiring.  Here we allow the unused bits of the GDX to connect
//   bits from the PLDs together.
//   This is the place for statements that connect the pins together to compensate for
//   missing connections on the PC board.


// reroute 15 is used to send the signal "RESET_ERROR_LATCHES" to the ERROR_LATCH ISP.
// reroute 7 is used to receive the signal "RESET_ERROR_LATCHES" from the CLOCK_CTL ISP
reroute15.m3 = reroute7;
//reroute15.m2 = cgnd;
//reroute15.m1 = bgnd;
//reroute15.m0 = agnd;
//reroute15.s0 = GND;
//reroute15.s1 = VCC;
//reroute15.oe = VCC;


// reroute 12 is used to send the signal "RUN_STATE" to the FIFO_CTL ISP.
// reroute 5 is used to receive the signal "RUN_STATE" from the CLOCK_CTL ISP
reroute12.m3 = reroute5;

// reroute 13 is used to send the signal "READ_EFIFO" to the FIFO_CTL ISP.
// reroute 1 is used to receive the signal "READ_EFIFO" from the VME_INT ISP
reroute13.m3 = reroute1;

END
```