ispLSI® and pLSI®: A Multiple Function Solution

Introduction

As high-density programmable logic becomes more common place, determining exactly which functions to integrate and how to integrate these functions becomes more challenging. Some of the obvious considerations when integrating a design include speed and density. Beyond these concerns, several other design and system details must be evaluated. In the following example, these design details will be examined and fully addressed. Design considerations can be broken into the following hierarchy: 1) System considerations including technology, reliability, and testability; 2) Design considerations which include partitioning a design for a specific architecture, determining I/O, and speed concerns; and 3) Integration of the design into an ispLSI device. This includes optimizing the ispLSI and pLSI architecture for the best speed and efficient random logic consolidation.

A Dual Processor Controller

The design shown in Figure 1 is a dual processor controller which sits on a backplane bus to which other CPUs have access. All of the CPUs communicate via the backplane bus by sending interrupts back and forth. This design also contains an independent 32-bit general purpose counter along with CPU control logic for memory and I/O.

Before partitioning the design, one must consider board space and reliability. For example, in some systems where board space and reliability are at a premium, it may be desirable to surface mount all components. In these cases, using sockets may be necessary to minimize manufacturing problems for the programmable devices. All Lattice Semiconductor ispLSI devices are In-System Programmable®, so removing devices from the board is not necessary if reprogramming is required. Another benefit to directly soldering components on the board is that less board space is needed and less capacitance will load the outputs. Therefore, soldering devices directly on the board will not increase propagation delay. To reprogram the ispLSI device, 5 volts and a five-wire interface are all that is needed. In addition, choosing an instantly reprogrammable technology allows complete testability. Lattice tests for and guarantees 100% AC, DC, functional and programming yields.

Having considered these overall issues, we can now look at partitioning the design. Many designers partition by using GAL devices or other PLDs for speed and fast state machine control, and FPGAs for interface and random logic. The Lattice ispLSI family rewrites these basic design rules. With the Lattice ispLSI and pLSI families of high-density programmable logic devices, the designer acquires speed and density in one device! The design must still be partitioned, but within the Generic Logic Blocks of the device rather than between several discrete PLDs and/or FPGAs.

This design can be broken into three major blocks: the two interrupt and bus random logic blocks, the data block consisting of a 32-bit counter with a 32-to-16 multiplexer, and the memory and I/O control logic state machine.

Figure 1. Dual Process Controller Block Diagram

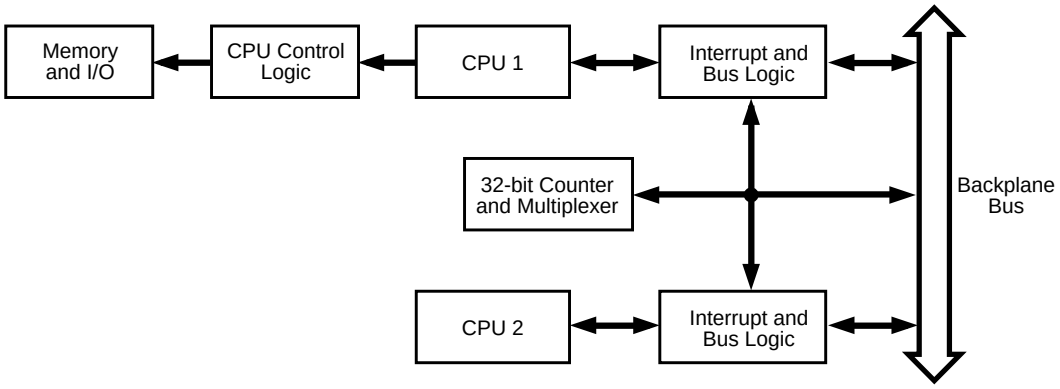
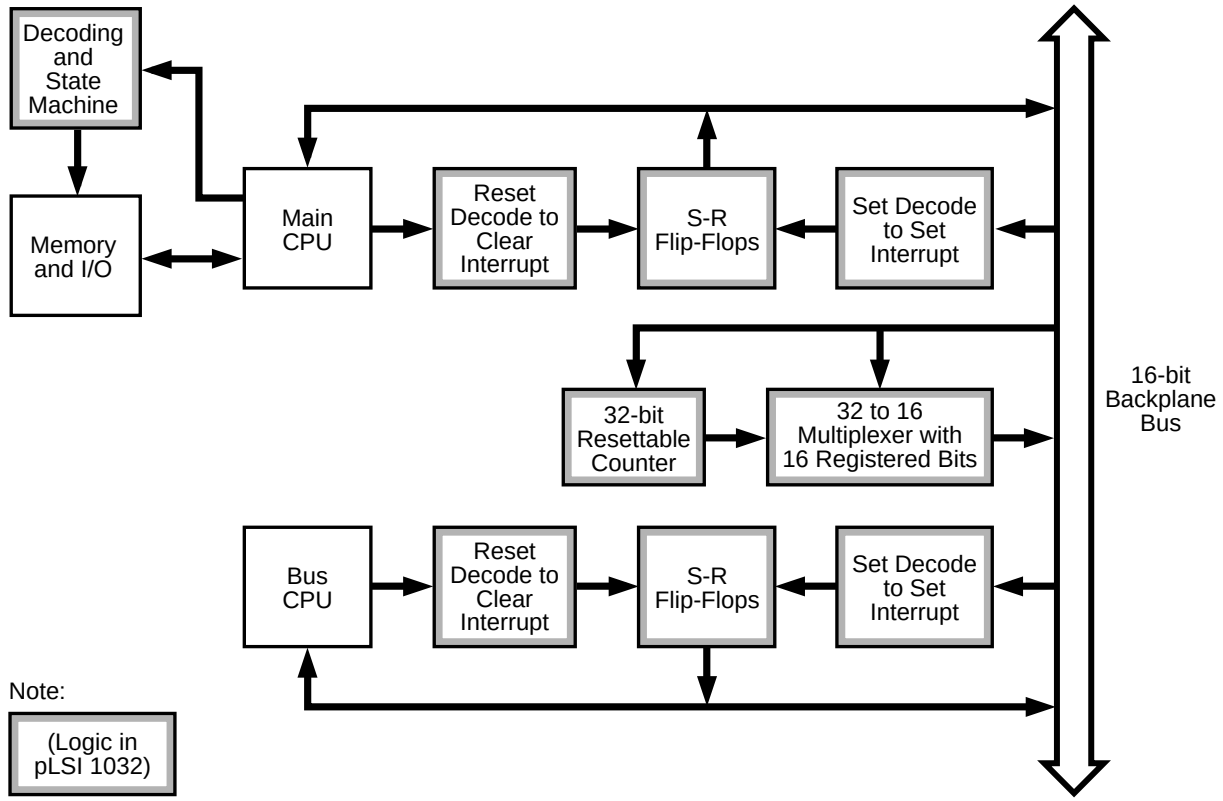


Figure 2. The Partitioned Design



Traditional FPGA devices would integrate the interrupt and bus logic and the 32-bit counter since the speed is not critical. The memory and I/O control logic would be left to the GAL® devices. With the Lattice pLSI® architecture's density and speed, many designs of this type can be fully integrated into one device.

Because of the architecture of the ispLSI® and pLSI devices, the key concern for engineers will be I/O pin conservation. Counting the I/Os in this design (62 including the clocks), the pLSI 1032 with 64 I/O pins and 8 dedicated inputs will fit this application nicely. There are four types of input/output configurations which can be implemented by the pLSI 1032 architecture. These configurations are input only, output only, 3-state output, and bi-directional I/O. In addition, input registers and latches are also available. When executing designs with Lattice software, it is necessary to label all of the I/O signals. I/O examples will follow later in this application note. All Boolean equations are in a syntax format which can be used in an ASCII text file and then imported into, or used directly in the Lattice ispLSI and pLSI Development Systems (pDS®) environment. Figure 2 shows the portion of the design implemented in the pLSI 1032 device.

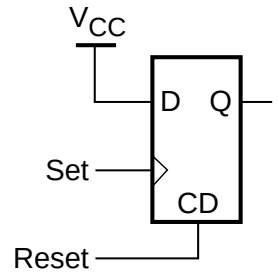
Interrupt and Bus Random Logic

Let us examine the details of each of the three sections to see how they would be implemented into the pLSI architecture. First, how to implement the decoding and latching of the interrupts. For this design, integration of the decoding logic for the set and reset terms and set/reset flip-flops is necessary.

The decoding logic is easily integrated, because the architecture has the familiar AND-OR structure. The less obvious detail which must be dealt with is exactly how to perform the Set/Reset flip-flop function. There are two choices to be explored. The first would be to use the product term reset in the Generic Logic Block, or GLB, as reset and use the product term clock as the preset with the 'D' input tied to a '1' (see Figure 3).

This approach works for implementing a small number of unique S-R flip-flops. If many unique S-R flip-flops are needed (this example requires 12), a different implementation must be used. This is because all four registers share a commong clock in each Generic Logic Block. Therefore, using just one register would require the other

Figure 3. D-Type flip-flop Configured as an S-R flip-flop



three outputs to be combinatorial, or registers with the same clock and reset. If there are several unique S-R flip-flops, each would have to exist in separate GLBs. This is not an efficient use of the architecture, unless the other outputs can be used as combinatorial logic. For this example, a more effective use of the GLBs can be achieved by making an S-R flip-flop from gates. The logic equations necessary are shown in Listing 1.

With this implementation, two S-R registers can fit into one GLB. The limiting factor in deciding whether two registers will fit, is the number of inputs necessary to perform the S-R function. Each GLB has a maximum of 18 inputs. If the number of inputs (including fast feedbacks), for the two registers is 18 or less, then both equations can be used in one GLB. In this design we have a total of 12 S-R registers. Listing 1 shows the equations for two S-R registers from the design, followed by the same equations reconfigured using the gate S-R implementation in Listing 2.

The number of unique input and feedback signals in the equations above, is 14. Since this is less than 18, the equations will fit into one GLB. To implement the other ten S-R registers, simply use the same strategy and partition the logic into five GLBs.

Data Path: 32-bit Counter and 32-to-16 Multiplexer

The next task is deciding how to build the 32-bit counter and the 32-to-16 multiplexer data latch. Using the ispLSI architecture, counter implementations up to 16 bits are straightforward. For up to 16 bits, the counter can run at the full speed of the device. There are two reasons the counter is able to execute at full speed: 1) wide input GLBs; and 2) T-type flip-flops configurable in the architecture. The T-type flip-flop is created by inserting an XOR gate before a D-type flip-flop and feeding back the D output into one of the two inputs to the XOR gate. The other input to the XOR gate becomes the T-type flip-flop input. Beyond 16 bits, a counter must be cascaded into another level of logic since the total number of inputs needed exceeds the maximum allowed by the GLB architecture. Recall that each GLB has an 18 input limit. Two of the inputs are dedicated input pins and the other 16 are I/O pins or fast feedbacks. Therefore, to implement a 32-bit counter, we must use two more GLBs to decode the point at which the counter has reached the full 16 bit mark. This is accomplished by setting an output true when all bits (0-15) are a '1'. Also, it is necessary to decode the point at which the counter has reached the full 24 bit mark. This is done by setting an output true when all bits (0-23) are a '1'. Using these intermediate terminal count outputs, a 32-bit counter can be implemented in 10 GLBs. This 32-bit counter can run at 40 MHz as implemented here, or up to 80 MHz if the carry out is pipelined. The equations for this counter are shown in Listing 3.

Listing 1.

```
Q = !Set # !Qbar;      // Q is the output of the S-R  flip-flop
Qbar = !Reset # !Q;   // Qbar is the inversion of Q
```

Listing 2.

```
reset1 = bp_int_clr & bp_data12 # bp_reset;
reset2 = bp_int_clr & bp_data11 # bp_reset;
set1 = !m_as & !ipc_int & mdata8 & !mdata10 & !mdata11 & !mdata12;
set2 = !m_as & !ipc_int & mdata8 & mdata10 & !mdata11 & !mdata12;
These equations are now optimized to combine the logic in one GLB:
Q1 = !(!m_as & !ipc_int & mdata8 & !mdata10 & !mdata11 & !mdata12) # !Q1bar;  //
!set1
Q1bar = !(bp_int_clr & bp_data12 # bp_reset) # !Q1 ;  //!reset1
Q2 = !(!m_as & !ipc_int & mdata8 & mdata10 & !mdata11 & !mdata12) #!Q2bar;  //
!set2
Q2bar = !(bp_int_clr & bp_data11 # bp_reset) # !Q2;  // !reset2
```

Listing 3.

```
// 0-7 decode
TC_1 = (QQ_0 & QQ_1 & QQ_2 & QQ_3 & QQ_4 & QQ_5 & QQ_6 & QQ_7);
// 0-15 decode
TC_2 = (QQ_8 & QQ_9 & QQ_10 & QQ_11 & QQ_12 & QQ_13 & QQ_14 & QQ_15 & TC_1);
// 0-23 decode
TC_3 = (QQ_16 & QQ_17 & QQ_18 & QQ_19 & QQ_20 & QQ_21 & QQ_22 & QQ_23 & TC_2);
// The QQ_0 to QQ_31 signals are the 32 counter output bits.
QQ_0 = QQ_0 $$ VCC ;
QQ_1 = QQ_1 $$ QQ_0;
QQ_2 = QQ_2 $$ QQ_1 & QQ_0 ;
QQ_3 = QQ_3 $$ QQ_2 & QQ_1 & QQ_0 ;
QQ_4 = QQ_4 $$ QQ_3 & QQ_2 & QQ_1 & QQ_0 ;
QQ_5 = QQ_5 $$ QQ_4 & QQ_3 & QQ_2 & QQ_1 & QQ_0 ;
QQ_6 = QQ_6 $$ QQ_5 & QQ_4 & QQ_3 & QQ_2 & QQ_1 & QQ_0 ;
QQ_7 = QQ_7 $$ QQ_6 & QQ_5 & QQ_4 & QQ_3 & QQ_2 & QQ_1 & QQ_0 ;
QQ_8 = QQ_8 $$ TC_1 ;
QQ_9 = QQ_9 $$ QQ_8 & TC_1 ;
QQ_10 = QQ_10 $$ QQ_9 & QQ_8 & TC_1 ;
QQ_11 = QQ_11 $$ QQ_10 & QQ_9 & QQ_8 & TC_1 ;
QQ_12 = QQ_12 $$ QQ_11 & QQ_10 & QQ_9 & QQ_8 & TC_1 ;
QQ_13 = QQ_13 $$ QQ_12 & QQ_11 & QQ_10 & QQ_9 & QQ_8 & TC_1 ;
QQ_14 = QQ_14 $$ QQ_13 & QQ_12 & QQ_11 & QQ_10 & QQ_9 & QQ_8 & TC_1 ;
QQ_15 = QQ_15 $$ QQ_14 & QQ_13 & QQ_12 & QQ_11 & QQ_10 & QQ_9 & QQ_8 & TC_1 ;
QQ_16 = QQ_16 $$ TC_2 ;
QQ_17 = QQ_17 $$ QQ_16 & TC_2 ;
QQ_18 = QQ_18 $$ QQ_17 & QQ_16 & TC_2 ;
QQ_19 = QQ_19 $$ QQ_18 & QQ_17 & QQ_16 & TC_2 ;
QQ_20 = QQ_20 $$ QQ_19 & QQ_18 & QQ_17 & QQ_16 & TC_2 ;
QQ_21= QQ_21 $$ QQ_20 & QQ_19 & QQ_18 & QQ_17 & QQ_16 & TC_2 ;
QQ_22= QQ_22 $$ QQ_21 & QQ_20 & QQ_19 & QQ_18 & QQ_17 & QQ_16 & TC_2;
QQ_23= QQ_23 $$ QQ_22 & QQ_21 & QQ_20 & QQ_19 & QQ_18 & QQ_17 & QQ_16 & TC_2;
QQ_24= QQ_24 $$ TC_3 ;
QQ_25= QQ_25 $$ QQ_24 & TC_3 ;
QQ_26= QQ_26 $$ QQ_25 & QQ_24 & TC_3 ;
QQ_27= QQ_27 $$ QQ_26 & QQ_25 & QQ_24 & TC_3 ;
QQ_28= QQ_28 $$ QQ_27 & QQ_26 & QQ_25 & QQ_24 & TC_3 ;
QQ_29= QQ_29 $$ QQ_28 & QQ_27 & QQ_26 & QQ_25 & QQ_24 & TC_3 ;
QQ_30= QQ_30 $$ QQ_29 & QQ_28 & QQ_27 & QQ_26 & QQ_25 & QQ_24 & TC_3 ;
QQ_31= QQ_31 $$ QQ_30 & QQ_29 & QQ_28 & QQ_27 & QQ_26 & QQ_25 & QQ_24 & TC_3 ;
```

The 32-to-16 multiplexer latch is the next logic block to be constructed. In this design, the multiplexer allows the system bus access to 16 bits of the counter at a time. Either the high 16 bits (16-31) or the low 16 bits (0-15) are enabled to the bus. Since this multiplexer latch is a simple OR gate control function into a register, these 16 bits can be placed into four GLBs. Recall that each GLB has a maximum of four outputs. The equations for one GLB are shown in Listing 4.

These 16 bits are also 3-stated by a control pin. In the ispLSI 1032 architecture, four unique output enable terms are allowed. Each output enable can control up to 16 outputs or bi-directional pins. For example, a design could have 64 3-state outputs, but four output enable control signals would be used to control 16 outputs each. It is important to note that if an output enable signal is to control more than 16 outputs, the output enable signal will need to be defined more than once. In this design only 16 outputs are controlled by one output enable signal, therefore only one output enable is used. This signal is provided by defining the output enable in a GLB as shown in Listing 5.

Memory and I/O State Machine

Considering the memory and I/O state machine and decoding logic, the ispLSI GLB architecture has a path which is optimal for decoding logic. This path is utilized by choosing the four product term bypass mode. This mode allows an output with four product terms or less to exhibit input pin to output pin propagation delays of no more than 15 ns. Since decoding logic typically uses four product terms or less, this mode can be used for the critical propagation delay paths. The designer is cautioned to use the Four Product Term Bypass Mode sparingly, because too many paths designated as critical in any one design may result in a failure of the Place and Route algorithm. The syntax necessary to invoke the four product term bypass mode is shown in Listing 6.

Listing 4.

```
OMDATA0I.CLK = CLK;
OMDATA0I = (!CNTELO & QQ_16 )  # (CNTELO & QQ_0  ); //select high or low word
OMDATA1I = (!CNTELO & QQ_17 )  # (CNTELO & QQ_1  );
OMDATA2I = (!CNTELO & QQ_18 )  # (CNTELO & QQ_2  );
OMDATA3I = (!CNTELO & QQ_19 )  # (CNTELO & QQ_3  );
                        *
                        *
                        *
OMDATA31I = (!CNTELO & QQ_31 )  # (CNTELO & QQ_15);
```

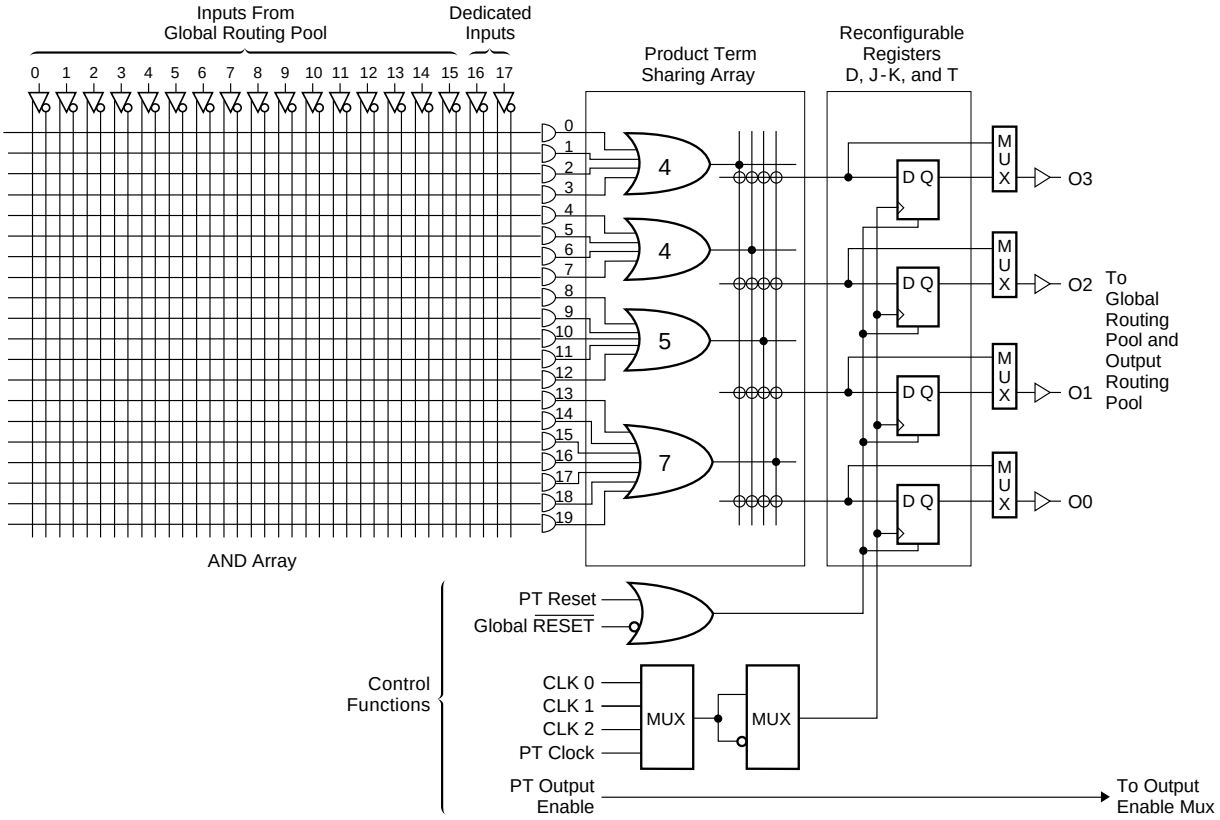
Listing 5.

```
BP_INT_RDI.OE = BP_INT_RD0; //PROGRAMMABLE OUTPUT ENABLE SIGNAL
```

Listing 6.

```
SIGTYPE  IO_SELECT 1 CRIT;  // CRIT - TELLS THE SOFTWARE TO USE THE  4  PRODUCT
TERM                            // BYPASS MODE FOR THIS COMBINATORIAL OUTPUT
EQUATIONS
IO_SELECT = MA23 & MPA22 & !MPA21 & MPIO_MEM & !MPWR_RD #
           MPA23 & !MPA22 & MPA21 & MPIO_MEM & MPWR_RD;
END;
```

Figure 4. GLB Product Term Sharing Array



The ispLSI 1032 is ideal for state machine applications because of two specific features. First, the I/O cell can be used to register or latch input signals. This attribute gives designers assurance that setup times to GLB registers will not be violated and metastability concerns are greatly diminished.

The second feature which configures efficient state machines is the standard GLB configuration with four, four, five and seven product terms (see Figure 4). The product terms can be tied together to perform wider product term functions which are always needed for complex state machines. For example, in a state machine which has an output consisting of nine product terms, the architecture will allow four of the product terms to be tied to five additional product terms, to add up to the total of nine, which is required by the state machine output. Any configuration of product term grouping is possible, including all 20! That is, if the design needs 20 product terms for one output, this is handled in one pass through just one GLB.

The key to successfully implementing state machines into the ispLSI 1032 is to utilize the 18 maximum inputs with up to four outputs, and the ability to tie the 20 product terms together. Intelligent use of these features permit the designer to streamline state machine design.

Summary

The ispLSI architecture provides designers with unparalleled flexibility, density and speed. ispLSI devices are dense and flexible enough to incorporate random logic. The architecture also contains 18 wide inputs and XOR capability in each GLB which enable counters to be effortlessly implemented. The four product term bypass mode allows designers to successfully realize high speed applications. Finally, the ability to tie product terms together along with the input registers available at each I/O pin make this device ideal for state machine designs.

The complete Lattice Design File containing the Boolean equations for this design appears on the following pages.

Design LDF Listing

```
// tedfulla.ldf generated using Lattice pDS Version 2.50

LDF 1.00.00 DESIGNLDF;
DESIGN cdx_design 1.00;
PART pLSI 1032-90LJ;
DECLARE
END;  //DECLARE

SYM GLB  A4  1  INTA52; // Here are 2 S-R flip-flops
                      // OUT signifies a combinatorial output
SIGTYPE INTA2I  OUT;
SIGTYPE INTA3I  OUT;
SIGTYPE INTA2IBAR  OUT;
SIGTYPE INTA3IBAR  OUT;
SIGTYPE BP_INT_RDI OE;  // OE signifies Output Enable
EQUATIONS
BP_INT_RDI = BP_INT_RDO;
INTA2I     = !(!MAS & !IPC_INTI & !MDATA12I & !MDATA11I & !MDATA10I & MDATA8I)
           # !INTA2IBAR.PIN;
INTA2IBAR  = !INTA2I.PIN
           # !(BP_INT_CLRI & BP_DATA10I RSETI);
INTA3I     = !(!MAS & !IPC_INTI &
           # !MDATA12I & !MDATA11I & MDATA10I & MDATA8I)
           # !INTA3IBAR.PIN;
INTA3IBAR  = !INTA3I.PIN
           # !(BP_INT_CLRI & BP_DATA15I RSETI);
END;
END;

SYM GLB  A5  1  INTA52;
SIGTYPE INTA4I  OUT;
SIGTYPE INTA5I  OUT;
SIGTYPE INTA4IBAR  OUT;
SIGTYPE INTA5IBAR  OUT;
EQUATIONS
INTA4I     = !(!MAS & !IPC_INTI & !MDATA12I & !MDATA11I & !MDATA10I & MDATA8I)
           # !INTA4IBAR.PIN;
INTA4IBAR  = !INTA4I.PIN
           # !(BP_INT_CLRI & BP_DATA14I RSETI);
INTA5I     = !(!MAS & !IPC_INTI & !MDATA12I & !MDATA11I & MDATA10I & MDATA8I)
           # !INTA5IBAR.PIN;
INTA5IBAR  = !INTA5I.PIN
           # !(BP_INT_CLRI & BP_DATA13I RSETI);
END;
END;
```

```
SYM GLB  B3  1  INTAMP45;
SIGTYPE INTAMP4I  OUT;
SIGTYPE INTAMP5I  OUT;
SIGTYPE INTAMP4IBAR  OUT;
SIGTYPE INTAMP5IBAR  OUT;
EQUATIONS
INTAMP4I    = !(!MAS & !IPC_INTI & !MDATA12I & !MDATA11I & !MDATA10I & MDATA8I)
            # !INTAMP4IBAR.PIN;
INTAMP4IBAR = !INTAMP4I.PIN
            # !(MP_INT_CLRI & MP_DATA14I RSETI);
INTAMP5I    = !(!MAS & !IPC_INTI & !MDATA12I & !MDATA11I & MDATA10I & MDATA8I)
            # !INTAMP5IBAR.PIN;
INTAMP5IBAR = !INTAMP5I.PIN
            # !(MP_INT_CLRI & MP_DATA13I RSETI);
END;
END;

SYM GLB  B4  1  INTAMP23;
SIGTYPE INTAMP2I  OUT;
SIGTYPE INTAMP3I  OUT;
SIGTYPE INTAMP2IBAR  OUT;
SIGTYPE INTAMP3IBAR  OUT;
SIGTYPE MP_INT_RDI OE;
EQUATIONS
MP_INT_RDI  = MP_INT_RDO;
INTAMP2I    = !(!MAS & !IPC_INTI & !MDATA12I & !MDATA11I & !MDATA10I & MDATA8I)
            # !INTAMP2IBAR.PIN;
INTAMP2IBAR = !INTAMP2I.PIN
            # !(MP_INT_CLRI & MP_DATA10I # RSETI);
INTAMP3I    = !(!MAS & !IPC_INTI & !MDATA12I & !MDATA11I & MDATA10I & MDATA8I)
            # !INTAMP3IBAR.PIN;
INTAMP3IBAR = !INTAMP3I.PIN
            # !(MP_INT_CLRI & MP_DATA15I RSETI);
END;
END;

SYM GLB  B5  1  INTAMP01;
SIGTYPE INTAMP0I  OUT;
SIGTYPE INTAMP1I  OUT;
SIGTYPE INTAMP0IBAR  OUT;
SIGTYPE INTAMP1IBAR  OUT;
EQUATIONS
INTAMP0I    = !(!MAS & !IPC_INTI & !MDATA12I & !MDATA11I & !MDATA10I & !MDATA8I)
            # !INTAMP0IBAR.PIN;
INTAMP0IBAR = !INTAMP0I.PIN
            # !(MP_INT_CLRI & MP_DATA12I RSETI);
INTAMP1I    = !(!MAS & !IPC_INTI & !MDATA12I & !MDATA11I & MDATA10I & MDATA8I)
            # !INTAMP1IBAR.PIN;
INTAMP1IBAR = !INTAMP1I.PIN
            # !(MP_INT_CLRI & MP_DATA11I RSETI);
END;
END;
```

```
SYM GLB  B6  1  INTA01;
SIGTYPE INTA0I  OUT;
SIGTYPE INTA1I  OUT;
SIGTYPE INTA0IBAR  OUT;
SIGTYPE INTA1IBAR  OUT;
EQUATIONS
INTA0I    = !(!MAS & !IPC_INTI & !MDATA12I & !MDATA11I & !MDATA10I & MDATA8I)
          # !INTA0IBAR.PIN;
INTA0IBAR = !INTA0I.PIN
          # !(BP_INT_CLRI & BP_DATA12I RSETI);
INTA1I    = !(!MAS & !IPC_INTI & !MDATA12I & !MDATA11I & MDATA10I & MDATA8I)
          # !INTA1IBAR.PIN;
INTA1IBAR = !INTA1I.PIN
          # !(BP_INT_CLRI & BP_DATA11I RSETI);
END;
END;

SYM GLB  A7  1  BPIPLS;
SIGTYPE BP_IPL0I OUT;
SIGTYPE BP_IPL1I  OUT;
SIGTYPE BP_IPL2I  OUT;
EQUATIONS
BP_IPL0I  = !INTA0I & !INTA2I & !INTA4I & BP_NMII & !DSP_INTI
          # !INTA0I & !INTA2I & !INTA4I & OS_TICKI & BP_NMII
          # BP_NMII & !TMS_INTI;
BP_IPL1I  = !INTA1I & !INTA3I & !INTA5I & OS_TICKI & BP_NMII & TMS_INTI
          # BP_NMII & TMS_INTI & !DSP_INTI
          # INTA4I & BP_NMII & TMS_INTI
          # INTA2I & BP_NMII & TMS_INTI
          # INTA0I & BP_NMII & TMS_INTI;
BP_IPL2I  = !INTA0I & !INTA2I & !INTA4I & BP_NMII & TMS_INTI & DSP_INTI;
END;
END;

SYM GLB  B0  1  MPIPLS;
SIGTYPE MP_IPL0I OUT;
SIGTYPE MP_IPL1I  OUT;
SIGTYPE MP_IPL2I  OUT;
EQUATIONS
MP_IPL0I  = !INTAMP0I & !INTAMP2I & !INTAMP4I & MP_NMII & !ROLL_TICKI
          # !INTAMP0I & !INTAMP2I & !INTAMP4I & OS_TICKI & MP_NMII
          # MP_NMII & EXP_TICKI;
MP_IPL1I  = !INTAMP1I & !INTAMP3I & !INTAMP5I & OS_TICKI & MP_NMII & !EXP_TICKI
          # MP_NMII & !EXP_TICKI & ROLL_TICKI
          # INTAMP4I & MP_NMII & !EXP_TICKI
          # INTAMP2I & MP_NMII & !EXP_TICKI
          # INTAMP0I & MP_NMII & !EXP_TICKI;
MP_IPL2I  = !INTAMP0I & !INTAMP2I & !INTAMP4I & MP_NMII & !EXP_TICKI&!ROLL_TICKI;
END;
END;
```

```
// 16 BIT COUNTERS
SYM GLB  D0  1  Q03;           // BITS Q0-Q3
SIGTYPE [QQ_0..QQ_3] REG OUT;        // SIGNIFIES A REGISTERED OUTPUT
EQUATIONS
QQ_0.CLK = CLK;
QQ_1.CLK = CLK;
QQ_2.CLK = CLK;
QQ_3.CLK = CLK;
QQ_0 = QQ_0 $$ VCC ;
QQ_1 = QQ_1 $$ QQ_0;
QQ_2 = QQ_2 $$ QQ_1 & QQ_0 ;
QQ_3 = QQ_3 $$ QQ_2 & QQ_1 & QQ_0 ;
END;
END;

SYM GLB  D1  1  Q47;           // BITS Q4-Q7
SIGTYPE [QQ_4..QQ_7] REG OUT;
EQUATIONS
QQ_4.CLK = CLK;
QQ_5.CLK = CLK;
QQ_6.CLK = CLK;
QQ_7.CLK = CLK;
QQ_4 = QQ_4 $$ QQ_3 & QQ_2 & QQ_1 & QQ_0 ;
QQ_5 = QQ_5 $$ QQ_4 & QQ_3 & QQ_2 & QQ_1 & QQ_0 ;
QQ_6 = QQ_6 $$ QQ_5 & QQ_4 & QQ_3 & QQ_2 & QQ_1 & QQ_0 ;
QQ_7 = QQ_7 $$ QQ_6 & QQ_5 & QQ_4 & QQ_3 & QQ_2 & QQ_1 & QQ_0 ;
END;
END;

SYM GLB  D2  1  Q811;          // BITS Q8-Q11
SIGTYPE [QQ_8..QQ_11] REG OUT;
EQUATIONS
QQ_8.CLK = CLK;
QQ_9.CLK = CLK;
QQ_10.CLK = CLK;
QQ_11.CLK = CLK;
QQ_8 = QQ_8 $$ TC_1 ;
QQ_9 = QQ_9 $$ QQ_8 & TC_1 ;
QQ_10 = QQ_10 $$ QQ_9 & QQ_8 & TC_1 ;
QQ_11 = QQ_11 $$ QQ_10 & QQ_9 & QQ_8 & TC_1 ;
END;
END;

SYM GLB  D3  1  Q1215;         // BITS Q12-Q15
SIGTYPE [QQ_12..QQ_15] REG OUT;
EQUATIONS
QQ_12.CLK = CLK;
QQ_13.CLK = CLK;
QQ_14.CLK = CLK;
QQ_15.CLK = CLK;
QQ_12 = QQ_12 $$ QQ_11 & QQ_10 & QQ_9 & QQ_8 & TC_1 ;
QQ_13 = QQ_13 $$ QQ_12 & QQ_11 & QQ_10 & QQ_9 & QQ_8 & TC_1 ;
QQ_14 = QQ_14 $$ QQ_13 & QQ_12 & QQ_11 & QQ_10 & QQ_9 & QQ_8 & TC_1 ;
```

```
QQ_15 = QQ_15 $$ QQ_14 & QQ_13 & QQ_12 & QQ_11 & QQ_10 & QQ_9 & QQ_8 & TC_1 ;
END;
END;

SYM GLB  D4  1  TC1;           // CARRY OUT OF BITS Q0-Q7 = HIGH
SIGTYPE TC_1 OUT;
EQUATIONS
TC_1 = (QQ_0 & QQ_1 & QQ_2 & QQ_3 & QQ_4 & QQ_5 & QQ_6 & QQ_7);
END;
END;

SYM GLB  C3  1  TC2;           // CARRY OUT OF BITS Q0-Q15 = HIGH
SIGTYPE TC_2 OUT;
EQUATIONS
TC_2 = (QQ_8 & QQ_9 & QQ_10 & QQ_11 & QQ_12 & QQ_13 & QQ_14 & QQ_15 & TC_1);
END;
END;

SYM GLB  D5  1  TC3;           // CARRY OUT OF BITS Q0-Q23 = HIGH
SIGTYPE TC_3 OUT;
EQUATIONS
TC_3 = (QQ_16 & QQ_17 & QQ_18 & QQ_19 & QQ_20 & QQ_21 & QQ_22 & QQ_23 & TC_2);
END;
END;

SYM GLB  D6  1  TERM;          // CARRY OUT OF BITS Q0-Q31 = HIGH
SIGTYPE TERMCNT  REG OUT;
EQUATIONS
TERMCNT.PTCLK =  (TC_1 & TC_2 & TC_3 & QQ_24 & QQ_25 & QQ_26 & QQ_27 & QQ_28 &
QQ_29 &                      QQ_30 & QQ_31);
TERMCNT = VCC;
END;
END;

SYM GLB  D7  1  Q1619;         // BITS Q16-Q19
SIGTYPE [QQ_16..QQ_19] REG OUT;
EQUATIONS
QQ_16.CLK = CLK;
QQ_17.CLK = CLK;
QQ_18.CLK = CLK;
QQ_19.CLK = CLK;
QQ_16 = QQ_16 $$ TC_2 ;
QQ_17 = QQ_17 $$ QQ_16 & TC_2 ;
QQ_18 = QQ_18 $$ QQ_17 & QQ_16 & TC_2 ;
QQ_19 = QQ_19 $$ QQ_18 & QQ_17 & QQ_16 & TC_2 ;
END;
END;
```

```
SYM GLB  C0  1  Q2023;          // BITS Q20-Q23
SIGTYPE [QQ_20..QQ_23] REG OUT;
EQUATIONS
QQ_20.CLK = CLK;
QQ_21.CLK = CLK;
QQ_22.CLK = CLK;
QQ_23.CLK = CLK;
QQ_20 = QQ_20 $$ QQ_19 & QQ_18 & QQ_17 & QQ_16 & TC_2 ;
QQ_21 = QQ_21 $$ QQ_20 & QQ_19 & QQ_18 & QQ_17 & QQ_16 & TC_2 ;
QQ_22 = QQ_22 $$ QQ_21 & QQ_20 & QQ_19 & QQ_18 & QQ_17 & QQ_16 & TC_2;
QQ_23 = QQ_23 $$ QQ_22 & QQ_21 & QQ_20 & QQ_19 & QQ_18 & QQ_17 & QQ_16 & TC_2;
END;
END;

SYM GLB  C1  1  Q2427;          // BITS Q24-Q27
SIGTYPE [QQ_24..QQ_27] REG OUT;
EQUATIONS
QQ_24.CLK = CLK;
QQ_25.CLK = CLK;
QQ_26.CLK = CLK;
QQ_27.CLK = CLK;
QQ_24 = QQ_24 $$ TC_3 ;
QQ_25 = QQ_25 $$ QQ_24 & TC_3 ;
QQ_26 = QQ_26 $$ QQ_25 & QQ_24 & TC_3 ;
QQ_27 = QQ_27 $$ QQ_26 & QQ_25 & QQ_24 & TC_3 ;
END;
END;
SYM GLB  C2  1  Q2831;          // BITS Q28-Q31
SIGTYPE [QQ_28..QQ_31] REG OUT;
EQUATIONS
QQ_28.CLK = CLK;
QQ_29.CLK = CLK;
QQ_30.CLK = CLK;
QQ_31.CLK = CLK;
QQ_28 = QQ_28 $$ QQ_27 & QQ_26 & QQ_25 & QQ_24 & TC_3 ;
QQ_29 = QQ_29 $$ QQ_28 & QQ_27 & QQ_26 & QQ_25 & QQ_24 & TC_3 ;
QQ_30 = QQ_30 $$ QQ_29 & QQ_28 & QQ_27 & QQ_26 & QQ_25 & QQ_24 & TC_3 ;
QQ_31 = QQ_31 $$ QQ_30 & QQ_29 & QQ_28 & QQ_27 & QQ_26 & QQ_25 & QQ_24 & TC_3  ;
END;
END;

// MULTIPLEXER GLBs
// SELECT HI ORDER BITS (16-31) IF !CNTEL0
// OR SELECT LOW ORDER BITS (0-15) IF CNTEL0
SYM GLB  B1  1  MDAT01;
SIGTYPE OMDATA0I  REG OUT;
SIGTYPE   OMDATA1I REG OUT;
EQUATIONS
XCNT_SEL1.OE = XCNT_SELI;
OMDATA0I.CLK = CNT_LTCH;
OMDATA0I = (!CNTELO & QQ_16 ) # (CNTELO & QQ_0  );
OMDATA1I = (!CNTELO & QQ_17 ) # (CNTELO & QQ_1  );
END;
END;
```

```
SYM GLB  C4  1  MDATA23;
SIGTYPE   OMDATA2I REG OUT;
SIGTYPE   OMDATA3I REG OUT;
EQUATIONS
OMDATA2I.CLK = CNT_LTCH;
OMDATA2I = (!CNTELO & QQ_18  ) # (CNTELO & QQ_2  );
OMDATA3I = (!CNTELO & QQ_19  ) # (CNTELO & QQ_3  );
END;
END;

SYM GLB  B2  1  MDAT45;
SIGTYPE OMDATA4I  REG OUT;
SIGTYPE   OMDATA5I  REG OUT;
EQUATIONS
OMDATA4I.CLK = CNT_LTCH;
OMDATA4I = (!CNTELO & QQ_20  ) # (CNTELO & QQ_4  );
OMDATA5I = (!CNTELO & QQ_21  ) # (CNTELO & QQ_5  );
END;
END;

SYM GLB  C5  1  MDAT67;
SIGTYPE   OMDATA6I   REG OUT;
SIGTYPE   OMDATA7I   REG OUT;
EQUATIONS
OMDATA6I.CLK = CNT_LTCH;
OMDATA6I = (!CNTELO & QQ_22  ) # (CNTELO & QQ_6  );
OMDATA7I = (!CNTELO & QQ_23  ) # (CNTELO & QQ_7  );
END;
END;

SYM GLB  C6  1  MDAT811;
SIGTYPE OMDATA8I  REG OUT;
SIGTYPE   OMDATA9I  REG OUT;
SIGTYPE   OMDATA10I  REG OUT;
SIGTYPE   OMDATA11I  REG OUT;
EQUATIONS
XCNT_SEL.OE = XCNT_SELI;
OMDATA8I.CLK = CNT_LTCH;
OMDATA8I = (!CNTELO & QQ_24 ) # (CNTELO & QQ_8  );
OMDATA9I = (!CNTELO & QQ_25  ) # (CNTELO & QQ_9  );
OMDATA10I = (!CNTELO & QQ_26 ) # (CNTELO & QQ_10  );
OMDATA11I = (!CNTELO & QQ_27 ) # (CNTELO & QQ_11  );
END;
END;
```

```
SYM GLB  C7  1  MDAT1215;
SIGTYPE OMDATA12I  REG OUT;
SIGTYPE OMDATA13I  REG OUT;
SIGTYPE OMDATA14I  REG OUT;
SIGTYPE OMDATA15I  REG OUT;
EQUATIONS
OMDATA12I.CLK = CNT_LTCH;
OMDATA12I = (!CNTELO & QQ_28) # (CNTELO & QQ_12);
OMDATA13I = (!CNTELO & QQ_29) # (CNTELO & QQ_13);
OMDATA14I = (!CNTELO & QQ_30) # (CNTELO & QQ_14);
OMDATA15I = (!CNTELO & QQ_31) # (CNTELO & QQ_15);
END;
END;

SYM GLB  A1  1  IOMEMOE;
SIGTYPE IO_SELECT0  OUT CRIT;
SIGTYPE IO_SELECT1  OUT CRIT;   // Signifies ORP Bypass
SIGTYPE MEMOE  OUT;
EQUATIONS
IO_SELECT0 = MPA23 & MPA22 & !MPA21 & MPIO_MEM & !MPWR_RD;
IO_SELECT1 = MPA23 & !MPA22 & MPA21 & MPIO_MEM & MPWR_RD;
MEMOE = !MPIO_MEM & !MPWR_RD & MPRDY;
END;
END;

SYM GLB  A0  1  MEMCSWR;
SIGTYPE MEMCS  REG OUT;
SIGTYPE MEMWR  REG OUT;
EQUATIONS
MEMCS.CLK = CLK;
MEMCS = MPA23 & !MPA22 & !MPA21 & !MPIO_MEM # MEMCS & MPRDY;              // CHIP
SELECT
MEMWR = MPA23 & !MPA22 & !MPA21 & !MPIO_MEM & MPWR_RD # MEMWR & MPRDY; // MEMORY
WRITE
     OR READ
END;
END;

// IO CELL ASSIGNMENTS
SYM IOC  IO51  1  MPIPLS;
XPIN IO DSP_INT;
IB11 (DSP_INTI,DSP_INT);             // IB11 = INPUT BUFFER
END;

SYM IOC  IO0  1  MPDAT15;
XPIN IO MP_DATA15           ;
BI11 (MP_DATA15I,MP_DATA15,INTAMP5I,MP_INT_RDI);
END;

SYM IOC  IO1  1  MPDAT14;
XPIN IO MP_DATA14;
BI11 (MP_DATA14I,MP_DATA14,INTAMP4I,MP_INT_RDI);   //BI11 = BIDIRECTIONAL I/O
END;
```

```
SYM IOC  IO2  1  MPDAT13;
XPIN IO MP_DATA13;
BI11 (MP_DATA13I,MP_DATA13,INTAMP3I,MP_INT_RDI);
END;

SYM IOC  IO3  1  MPDAT12;
XPIN IO MP_DATA12;
BI11 (MP_DATA12I,MP_DATA12,INTAMP2I,MP_INT_RDI);
END;

SYM IOC  IO4  1  MPDAT11;
XPIN IO MP_DATA11;
BI11 (MP_DATA11I,MP_DATA11,INTAMP1I,MP_INT_RDI);
END;

SYM IOC  IO5  1  MPDAT10;
XPIN IO MP_DATA10;
BI11 (MP_DATA10I,MP_DATA10,INTAMP0I,MP_INT_RDI);
END;

SYM IOC  IO8  1  MPINTCL;
XPIN IO MP_INT_CLR  LOCK 40;               // LOCK = FIXED PIN
IB11 (MP_INT_CLRI,MP_INT_CLR);
END;

SYM IOC  IO9  1  RSET;
XPIN IO RSET;
IB11 (RSETI,RSET);
END;

SYM IOC  IO10  1  MDATA15;
XPIN IO MDATA15  LOCK 53 ;
OT11 (MDATA15,OMDATA15I,!XCNT_SEL);              // TRISTATE OUTPUT
END;

SYM IOC  IO11  1  MDATA14;
XPIN IO MDATA14  LOCK 54 ;
OT11 (MDATA14,OMDATA14I,!XCNT_SEL);
END;

SYM IOC  IO12  1  MDATA13;
XPIN IO MDATA13  LOCK 55 ;
OT11 (MDATA13,OMDATA13I,!XCNT_SEL);
END;

SYM IOC  IO13  1  MDATA12;
XPIN IO MDATA12  LOCK 56 ;
BI11 (MDATA12I,MDATA12,OMDATA12I,!XCNT_SEL);
END;
```

```
SYM IOC  IO14  1  MDATA11;
XPIN IO MDATA11  LOCK 57 ;
BI11 (MDATA11I,MDATA11,OMDATA11I,!XCNT_SEL);
END;

SYM IOC  IO15  1  MDATA10;
XPIN IO MDATA10  LOCK 58 ;
BI11 (MDATA10I,MDATA10,OMDATA10I,!XCNT_SEL);
END;

SYM IOC  IO16  1  MDATA9;
XPIN IO MDATA9  LOCK 59 ;
OT11 (MDATA9,OMDATA9I,!XCNT_SEL);
END;

SYM IOC  IO17  1  MDATA8;
XPIN IO MDATA8  LOCK 60 ;
BI11 (MDATA8I,MDATA8,OMDATA8I,!XCNT_SEL);
END;

SYM IOC  IO26  1  MAS;
XPIN IO MASX   LOCK 27 ;
IB11 (MAS,MASX);
END;

SYM IOC  IO27  1  IPC_INT;
XPIN IO IPC_INT  LOCK 26 ;
IB11 (IPC_INTI,IPC_INT);
END;

SYM IOC  IO28  1  MPIPL2;
XPIN IO MP_IPL2;
OB11 (MP_IPL2,MP_IPL2I);
END;

SYM IOC  IO29  1  MPIPL1;
XPIN IO MP_IPL1;
OB11 (MP_IPL1,MP_IPL1I);
END;

SYM IOC  IO30  1  MPIPL0;
XPIN IO MP_IPL0;
OB11 (MP_IPL0,MP_IPL0I);
END;

SYM IOC  IO31  1  MPINTRD;
XPIN IO MP_INT_RD;
IB11 (MP_INT_RDO,MP_INT_RD);
END;
```

```
SYM IOC  IO32  1  BPINTRD;
XPIN IO BP_INT_RD;
IB11 (BP_INT_RDO,BP_INT_RD);
END;

SYM IOC  IO33  1  BPDAT15;
XPIN IO BP_DATA15;
BI11 (BP_DATA15I,BP_DATA15,INTA5I,BP_INT_RDI);
END;

SYM IOC  IO34  1  BPDAT14;
XPIN IO BP_DATA14;
BI11 (BP_DATA14I,BP_DATA14,INTA4I,BP_INT_RDI);
END;

SYM IOC  IO35  1  BPDAT13;
XPIN IO BP_DATA13;
BI11 (BP_DATA13I,BP_DATA13,INTA3I,BP_INT_RDI);
END;

SYM IOC  IO36  1  BPDAT12;
XPIN IO BP_DATA12;
BI11 (BP_DATA12I,BP_DATA12,INTA2I,BP_INT_RDI);
END;

SYM IOC  IO37  1  BPDAT11;
XPIN IO BP_DATA11;
BI11 (BP_DATA11I,BP_DATA11,INTA1I,BP_INT_RDI);
END;

SYM IOC  IO38  1  BPDAT10;
XPIN IO BP_DATA10;
BI11 (BP_DATA10I,BP_DATA10,INTA0I,BP_INT_RDI);
END;

SYM IOC  IO41  1  BPINTCL;
XPIN IO BP_INT_CLR;
IB11 (BP_INT_CLRI,BP_INT_CLR);
END;

SYM IOC  IO42  1  BPIPL2;
XPIN IO BP_IPL2;
OB11 (BP_IPL2,BP_IPL2I);
END;

SYM IOC  IO43  1  BPIPL1;
XPIN IO BP_IPL1;
OB11 (BP_IPL1,BP_IPL1I);
END;

SYM IOC  IO44  1  BPIPL0;
XPIN IO BP_IPL0;
OB11 (BP_IPL0,BP_IPL0I);
END;
```

```
SYM IOC  IO45  1  MP_NMI;
XPIN IO MP_NMI;
IB11 (MP_NMII,MP_NMI);
END;

SYM IOC  IO46  1  OS_TICK;
XPIN IO OS_TICK;
IB11 (OS_TICKI,OS_TICK);
END;

SYM IOC  IO47  1  EXPTICK;
XPIN IO EXP_TICK;
IB11 (EXP_TICKI,EXP_TICK);
END;

SYM IOC  IO48  1  ROLTICK;
XPIN IO ROLL_TICK;
IB11 (ROLL_TICKI,ROLL_TICK);
END;

SYM IOC  IO49  1  BP_NMI;
XPIN IO BP_NMI;
IB11 (BP_NMII,BP_NMI);
END;

SYM IOC  IO50  1  TMS_INT;
XPIN IO TMS_INT;
IB11 (TMS_INTI,TMS_INT);
END;

SYM IOC  IO18  1  MPCLR13;
XPIN IO MDATA7;
OT11 (MDATA7,OMDATA7I,!XCNT_SEL1);
END;

SYM IOC  IO19  1  MPCLR13;
XPIN IO MDATA6;
OT11 (MDATA6,OMDATA6I,!XCNT_SEL1);
END;

SYM IOC  IO20  1  MPCLR13;
XPIN IO MDATA5 LOCK 6;
OT11 (MDATA5,OMDATA5I,!XCNT_SEL1);
END;

SYM IOC  IO21  1  MPCLR13;
XPIN IO MDATA4 LOCK 5;
OT11 (MDATA4,OMDATA4I,!XCNT_SEL1);
END;

SYM IOC  IO22  1  MPCLR13;
XPIN IO MDATA3;
OT11 (MDATA3,OMDATA3I,!XCNT_SEL1);
END;
```

```
SYM IOC  IO23  1  MPCLR13;
XPIN IO MDATA2;
OT11 (MDATA2,OMDATA2I,!XCNT_SEL1);
 END;

SYM IOC  IO24  1  MPCLR13;
XPIN IO MDATA1 LOCK 4 ;
OT11 (MDATA1,OMDATA1I,!XCNT_SEL1);
END;

SYM IOC  IO25  1  MPCLR13;
XPIN IO MDATA0 LOCK 3 ;
OT11 (MDATA0,OMDATA0I,!XCNT_SEL1);
END;

SYM IOC  IO63  1  LTCH;
XPIN  IO  XCNTSEL;
IB11  (XCNT_SELI,  XCNTSEL);
END;

SYM IOC  Y1  1  LTCH;
XPIN CLK LTCH;
IB11 (CNT_LTCH,LTCH);
END;

SYM IOC  Y0  1  CLOCK;
XPIN  CLK  XCLK;
IB11 (CLK,  XCLK);
END;

SYM IOC  IO62  1  CNTELO;
XPIN  IO  OCNTELO;
IB11  (CNTELO, OCNTELO);
END;

SYM IOC  IO61  1  TERMCNT;
XPIN   IO  XTERMCNT;
OB11  (XTERMCNT, TERMCNT);
END;

SYM IOC  IO59  1  MPA23;
XPIN IO MPA23O;
IB11 (MPA23, MPA23O);
END;

SYM IOC  IO58  1  MPA21;
XPIN IO MPA22O;
IB11 (MPA22, MPA22O);
END;

SYM IOC  IO57  1  MPA21;
XPIN IO MPA21O;
IB11 (MPA21, MPA21O);
END;
```

```
SYM IOC  IO56  1  MPIO_MEM;
XPIN IO MPIO_MEMO;
IB11 (MPIO_MEM, MPIO_MEMO);
END;

SYM IOC  IO55  1  MPWR_RD;
XPIN IO MPWR_RDO;
ID11 (MPWR_RD, MPWR_RDO,CLK);
END;

SYM IOC  IO54  1  MPRDY;
XPIN IO MP_RDYO;
IB11 (MPRDY, MP_RDYO);
END;

SYM IOC  IO53  1  IO_SELECT;
XPIN IO IO_SELECT0O;
OB11 (IO_SELECT0O, IO_SELECT0);
END;

SYM IOC  IO52  1  IO_SELECT1;
XPIN IO IO_SELECT1O;
OB11 (IO_SELECT1O, IO_SELECT1);
END;

SYM IOC  IO7  1  MEMCS;
XPIN IO MEMCSO  ;
OB11 (MEMCSO, MEMCS);
END;

SYM IOC  IO60  1  MEMOE;
XPIN IO MEMOEO;
OB11 (MEMOEO, MEMOE);
END;
END;  //LDF DESIGNLDF
```

LATTICE SEMICONDUCTOR CORPORATION
5555 Northeast Moore Court
Hillsboro, Oregon 97124 U.S.A.
Tel.: (503) 681-0118
FAX: (503) 681-3037
http://www.latticesemi.com

November 1996