Lattice™
Semiconductor
Corporation

ispLSI® Configurable Memory Controller

Introduction

There are many advantages in using In-System Programmable® ispLSI devices. In board level designs, as well as during manufacturing, the flexibility of hardware reconfiguration can lead to many innovative system designs. Once configured, the ispLSI devices' non-volatile E^2CMOS® cells will retain their configuration even when the power is turned off. The guaranteed 10,000 programming cycles and 20 year data retention of the ispLSI device will allow the user to reliably reconfigure the device as often as required.

This application note highlights the advantages of designing with ispLSI devices and how they can lead to innovative design ideas which translate to ease of use and instant updates without board layout changes. The flexibility of design is illustrated with the use of the Dynamic Random Access Memory (DRAM) controller. This example shows a typical microprocessor and memory interface with the memory controller controlling the DRAM access and refresh timing requirements. The use of pLSI® and ispLSI Development System (pDS®) Software is also illustrated in this application note. The Lattice Design File (.ldf) listing file generated by the software is also attached at the end of this section.

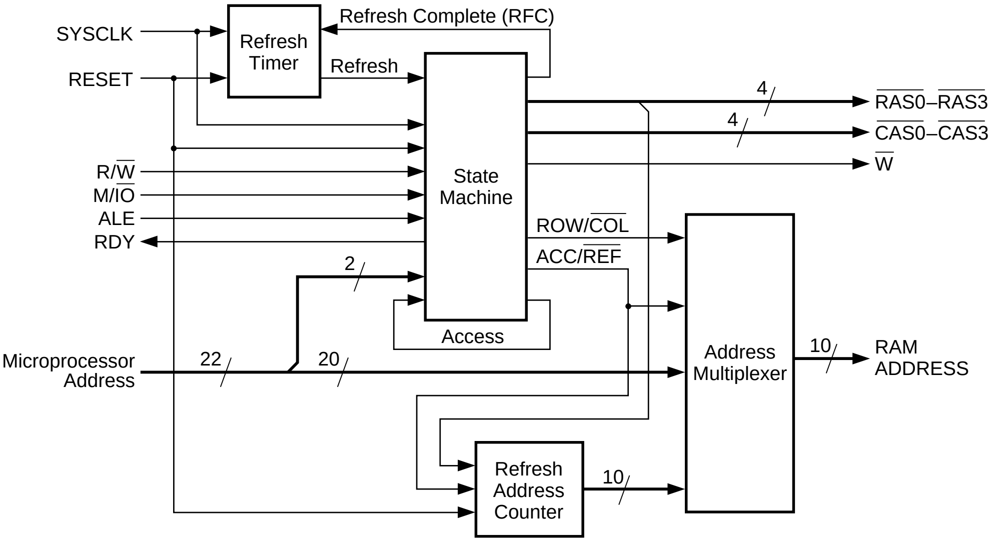
Memory Controller Logic Overview

When interfacing the microprocessor to the DRAM, the control signal and timing requirements of both the processor and the DRAM must be satisfied. In order to satisfy these requirements, the external timing controller must take the processor address, data and control signals and translate them into the control signals for the DRAM. At the same time, the DRAM timing controller must take into account the refresh requirements of the DRAM.

Figure 1 shows the block diagram of the DRAM timing controller that is implemented in the ispLSI 1032. The state machine and address multiplexer blocks are used to control the memory access request of the processor and supply the DRAM with the necessary address and control signals. DRAM refresh requirements are controlled by the refresh timer block, refresh address counter block and the address multiplexer block.

Any access request from the processor is processed by the state machine based on the processor control signals such as Read/Write (R/W), Memory/IO access (M/IO), Address Latch Enable (ALE) and the microprocessor address signals. The Ready (RDY) signal is used to

Figure 1. DRAM Timing Controller Block Diagram



inform the processor the status of the requested data. In other words, it is used to acknowledge the processor that the memory is ready to respond to the processor. The address multiplexer generates the row and column addresses necessary for the memory access cycle. The appropriate Row Address Strobe (RAS), Column Address Strobe (CAS), and Write (W) signals are also generated by the state machine based on the processor inputs. To arbitrate between the memory access request and the refresh request, the state machine also generates the status signal called Access. The purpose of this signal is to keep track of an access cycle when the refresh sequence is in progress. This status signal is then used to determine whether or not to begin an access sequence after the refresh sequence. As part of the access/refresh arbitration, the state machine also issues an Access/Refresh (ACC/REF) signal to the address multiplexer logic block. Based on this signal the address multiplexer block routes the appropriate access or refresh address on to the external DRAM address bus.

As for all DRAMs, memory refresh must be completed within a specified time. This process is completely controlled by the DRAM timing controller. The refresh timer block generates the internal refresh request signal according to the system clock speed and the DRAM refresh rate requirements. When the state machine detects this refresh request signal, the refresh sequence for the DRAM is generated as soon as time permits. This means that the refresh sequence is generated right after the refresh request or if the timing controller is in the middle of a memory access cycle the refresh sequence is generated right after the memory access cycle is complete. During the refresh sequence, the row address and all the RAS signal must be activated to perform the basic RAS-only refresh. The row addresses are supplied by the refresh address counter logic block. This logic block keeps track of the rows that are being refreshed and it gets incremented every time a refresh sequence is performed. All the RAS signal are activated for refresh by the state machine.

With the basic understanding of the DRAM timing control logic complete, the next section will discuss the implementation of the logic in an ispLSI device and how to take advantage of the ISP™ features to make the system design, manufacturing and field updates easy and flexible.

Taking Advantage of ISP Features

Implementing a basic DRAM timing control logic in the ispLSI 1032 takes up approximately 65% of the total logic available in the device. (It is with this in mind that the features needed for a specific design can be added to these basic logic blocks). With ISP capability, many features can be added to accommodate the ever-changing requirements of the system, microprocessor speeds, availability of DRAMs, and the memory configurations. Moreover, the changes are made only under the software control. Instead of having different production runs for various different options, the options are added at the in-system programming stage.

Different System Speed

Designing with a different speed microprocessor requires a different DRAM timing controller. The adjustments must be made in the state machine and refresh timer logic of the controller to account for the difference in speed. Without the capabilities of the ISP features, different boards with different PLD codes must be built to work with different processor speeds. By providing a simple programming circuitry on board to support the isp programming, the logic adjustments for different speed processor can be accomplished by in-system programming the different patterns via software control. Manufacture of these options are made simple and cost effective by not having to keep an inventory of prepatterned devices.

DRAM Feature Flexibility

DRAMs have many features from which the system designer can select. For the same DRAM configuration, the system designer can select from DRAMs that have different access schemes such as nibble mode, static column mode and page mode. Similarly, different memory refresh schemes can be chosen. The two choices of refresh schemes include the simple RAS only refresh and the option to perform hidden refresh with the CAS before RAS refresh scheme. Most of these various DRAM options can be supported by programming the ispLSI devices in-system. Again, the flexibility lies in the fact that the decision about what function the ispLSI device will perform on board can be made after the type of DRAMs used on board have been determined.

Different DRAM Configuration

The ispLSI implementation of the DRAM timing controller makes the change of memory configuration very simple. Reprogramming of the address decoding and turning on the appropriate address strobe signals for different memory configuration can be done by in-system reconfiguration of the state machine and the address decoding of the ispLSI device. All of these changes can be accomplished under software control.

Memory Timing Controller Details

As shown in Figure 1, the memory timing controller consists of four different logic blocks. The refresh timer, state machine, refresh address counter and memory address multiplexer. All Boolean equations for the logic blocks are developed within the pDS software. The entire memory timing controller design assumes that all the processor signals are typical of a commercially available processor with a clock speed of 25MHz. DRAMs are arranged in four banks of 1M X 32-bit arrangement. All timing for the access and refresh sequences are shown in the timing diagram.

Refresh Timer

The function of the refresh timer is to generate a refresh request signal every 15.5 µs. This refresh period is derived from the DRAM refresh requirement of 512 rows of refresh every 8ms for the 1M X 1 DRAM. Based on the 25MHz system clock frequency, the count value to divide the clock period to the refresh period is 200. Changing processor speed will only require a change of count value. Once the count value expires, the refresh timer generates an internal refresh signal to inform the state machine to perform a refresh cycle. When the state machine completes the refresh cycle, a refresh complete (RFC) signal is generated for the refresh timer. The refresh timer then resets the internal counter for the next refresh period.

ispLSI implementation of the refresh timer takes up three GLBs (A0-A2) within the device. The system clock is used to run the 9-bit counter, RFC is the input signal to this block and REFRESH is the output signal of this logic block.

State Machine

The state machine can be further divided into four different sub-logic blocks. These sub-logic blocks consists of a RAS generator, CAS generator, four-bit state machine which is divided into two state variable bits and two counter bits, and control signal generator. In the ispLSI 1032 implementation, the state machine logic block takes up nine GLBs.

The four-bit state machine is divided into a two-bit state variable, named ST0 and ST1, and two-bit state counter, named SCNT0 and SCNT1. The state diagram with its state transitions are shown in Figure 2. In each of the access and refresh states, the state counter sequences through the operation until the sequence is complete. The purpose of the state variable bits are only to keep track of the state transitions. Once the state transition has occurred, the state counter bits take the responsibility of sequencing through the state.

The three states are divided as idle state, access state and refresh state. Based on the processor control signal and the internal refresh request signal, the state transition occurs from idle state to either access state or refresh state. If the refresh and access request happen at the same time, refresh request takes precedence over access request. When the refresh request is asserted during an access cycle, the refresh cycle follows right after the access cycle. The only other condition between the access and refresh request that the state machine needs to arbitrate is when the access request occurs during the refresh sequence. The access feedback signal of the state machine is activated when the access request occurs during the refresh cycle. When the refresh cycle is complete, the access feedback signal is used to determine whether or not the access sequence needs to begin. The timing diagrams in Figures 3 and 4 illustrate the control signal sequence for the access and refresh cycles, respectively.

In addition to the external DRAM control signals, the state machine also generates the control signal for the address multiplexer and the refresh address counter. The ROW/COL signal directs the address multiplexer to output the appropriate row and column address during the access cycle. Furthermore, the address multiplexer accepts the

Figure 2. DRAM Timing Controller State Machine

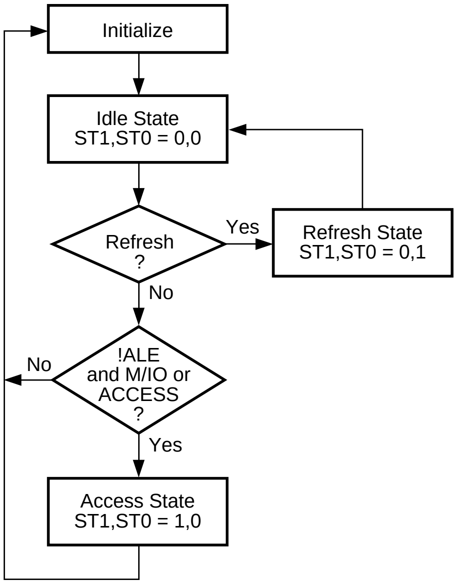


Figure 3. Access Cycle Timing

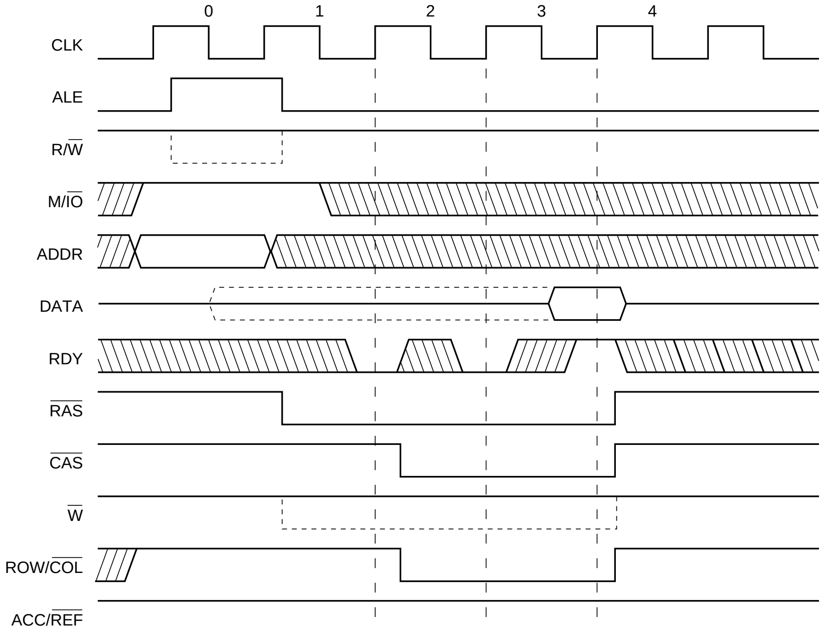
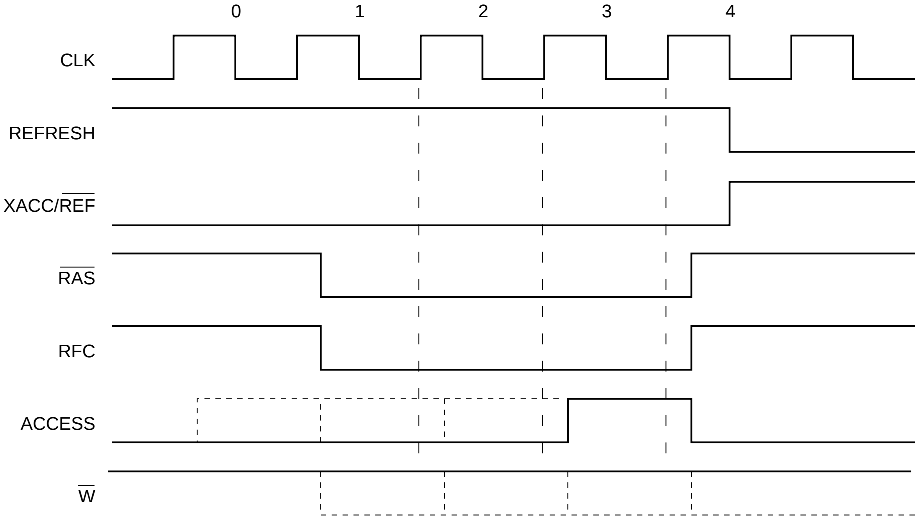


Figure 4. Refresh Cycle Timing



access/refresh (ACC/REF) control signal to either direct the memory access address from the processor or direct the refresh row address from the refresh address counter to the DRAM.

Refresh Address Counter

The refresh address counter keeps track of the rows of DRAM to be refreshed. This counter is only incremented on the falling edge of the RAS signal during refresh sequence. The ispLSI device implementation of this counter takes three GLBs.

Memory Address Multiplexer

In access mode, determined by the ACC/REF internal signal, the memory address multiplexer multiplexes between the row and column address. Once in the refresh cycle, the refresh address comes from the refresh address counter. It takes three GLBs to implement the multiplexer in the ispLSI 1032.

Summary

This application note uses the example of a generalized DRAM timing controller to demonstrate how ISP features can be used to improve design features and the manufacturing process. The software example provides a starting point for designers implementing state machine based designs. With the flexibility of ispLSI devices, innovative design possibilities are limitless.

The following section lists the Lattice Design file with the Boolean Equations and pinout for the ispLSI 1032.

Design LDF Listing

```
//isp_app.ldf generated using Lattice pDS Version 2.50
LDF 1.00.00 DESIGNLDF;
DESIGN DRAM CONTROLLER 1.00;
PROJECTNAMEispAPPLICATIONS;
DESCRIPTION
DRAM CONTROLLER DESIGN FORispAPPLICATION.
IT INCLUDES FOUR MAJOR BLOCKS.
   - REFRESH TIMER
   - REFRESH ROW ADDRESS COUNTER
   - ADDRESS MUX
   - STATE MACHINE;

PART ispLSI 1032-90LJ;

DECLARE
END;  //DECLARE

SYM GLB  C2  1  ;
   /////    ROW ADDRESS STROBE (RAS1,RAS0) GLB      /////
   SIGTYPE IRAS1 REG OUT;
   SIGTYPE IRAS0 REG OUT;
   EQUATIONS
      IRAS1.CLK = ICLK;
      IRAS1 = !ST0 & !IA20 & IRAS1 & !IRESET   /////   REDUCED RAS1    /////
           # !ST1 & IA21 & IRAS1 & !IRESET
           # !ST0 &  ST1 & SCNT0 & SCNT1 & IA20 & !IA21 & !IRESET
           #  ST0 & !ST1 & SCNT0 & SCNT1 & !IRESET
           # !ST0 & !ST1 & IRAS1 & !IRESET
           #  ST0 & ST1 & IRAS1 & !IRESET
           #  SCNT1 & IRAS1 & !IRESET
           #  SCNT0 & IRAS1 & !IRESET;

      IRAS0 = !ST0 & IA20 & IRAS0 & !IRESET    /////   REDUCED RAS0    /////
           # !ST1 & IA21 & IRAS0 & !IRESET
           # !ST0 &  ST1 & SCNT0 & SCNT1 & !IA20 & !IA21 & !IRESET
           #  ST0 & !ST1 & SCNT0 & SCNT1 & !IRESET
           # !ST0 & !ST1 & IRAS0 & !IRESET
           #  ST0 & ST1 & IRAS2 & !IRESET
           #  SCNT1 & IRAS0 & !IRESET
           #  SCNT0 & IRAS0 & !IRESET;
   END
END;

SYM GLB  A2  1  ;
   /////     REFRESH TIMER GLB2     /////
   SIGTYPE RQ8 REG OUT;
   SIGTYPE RQ9 REG OUT;
   SIGTYPE REFRESH REG OUT;
   FJK11 (REFRESH,R_RATE,RFC,ICLK);  /////   REFRESH REQUEST SIGNAL   /////
   EQUATIONS
      RQ8.CLK = ICLK;
      RQ8 = (RQ8 & !RFC)
         $$ (RQ7 & RQ6 & RQ5 & RQ4 & RQ3 & RQ2 & RQ1 & RQ0 & !RFC);
      RQ9 = (RQ9 & !RFC)
         $$ (RQ8 & RQ7 & RQ6 & RQ5 & RQ4 & RQ3 & RQ2 & RQ1 & RQ0 & !RFC);
      R_RATE = RQ7 & RQ6 & !RQ5 & !RQ4 & RQ3 & !RQ2 & !RQ1 & !RQ0;
   END
END;
```

```
SYM GLB  A1  1  ;
   /////     REFRESH TIMER GLB1     /////
   SIGTYPE RQ4 REG OUT;
   SIGTYPE RQ5 REG OUT;
   SIGTYPE RQ6 REG OUT;
   SIGTYPE RQ7 REG OUT;
   EQUATIONS
      RQ4.CLK = ICLK;
      RQ4 = (RQ4 & !RFC)
         $$ (RQ3 & RQ2 & RQ1 & RQ0 & !RFC);
      RQ5 = (RQ5 & !RFC)
         $$ (RQ4 & RQ3 & RQ2 & RQ1 & RQ0 & !RFC);
      RQ6 = (RQ6 & !RFC)
         $$ (RQ5 & RQ4 & RQ3 & RQ2 & RQ1 & RQ0 & !RFC
      RQ7 = (RQ7 & !RFC)
         $$ (RQ6 & RQ5 & RQ4 & RQ3 & RQ2 & RQ1 & RQ0 & !RFC);
   END
END;

SYM GLB  A0  1  ;
   //////  REFRESH TIMER GLB0   /////
   SIGTYPE RQ0 REG OUT;
   SIGTYPE RQ1 REG OUT;
   SIGTYPE RQ2 REG OUT;
   SIGTYPE RQ3 REG OUT;
   EQUATIONS
      RQ0.CLK = ICLK;
      RQ0 = !RQ0 & !RFC;
      RQ1 = (RQ1 & !RFC)
         $$ (RQ0 & !RFC);
      RQ2 = (RQ2 & !RFC)
         $$ (RQ1 & RQ0 & !RFC);
      RQ3 = (RQ3 & !RFC)
         $$ (RQ2 & RQ1 & RQ0 & !RFC);
   END
END;

SYM GLB  D0  1  ;
   /////     ADDRESS MUX GLB0      /////
   SIGTYPE IRAM0 ASYNC OUT;
   SIGTYPE IRAM1 ASYNC OUT;
   SIGTYPE IRAM2 ASYNC OUT;
   SIGTYPE IRAM3 ASYNC OUT;
   EQUATIONS
      IRAM0 = ROW_COL & ACC_REF & IA0    /////  ROW SELECT   /////
         #  !ROW_COL & ACC_REF & IA10 /////  COLUMN SELECT   /////
         #  !ACC_REF & RCNTR0;               /////   REFRESH ADDR SELECT  /////
      IRAM1 = ROW_COL & ACC_REF & IA1
         #  !ROW_COL & ACC_REF & IA11
         #  !ACC_REF & RCNTR1;
      IRAM2 = ROW_COL & ACC_REF & IA2
         #  !ROW_COL & ACC_REF & IA12
         #  !ACC_REF & RCNTR2;
      IRAM3 = ROW_COL & ACC_REF & IA3
         #  !ROW_COL & ACC_REF & IA13
         #  !ACC_REF & RCNTR3;
   END
END;
```

```
SYM GLB  D1  1  ;
   /////     ADDRESS MUX GLB1      /////
   SIGTYPE IRAM4 ASYNC OUT;
   SIGTYPE IRAM5 ASYNC OUT;
   SIGTYPE IRAM6 ASYNC OUT;
   SIGTYPE IRAM7 ASYNC OUT;
   EQUATIONS
      IRAM4 = ROW_COL & ACC_REF & IA4    /////  ROW SELECT   /////
          #  !ROW_COL & ACC_REF & IA14 /////  COLUMN SELECT   /////
          #  !ACC_REF & RCNTR4;               /////   REFRESH ADDR SELECT  /////
      IRAM5 = ROW_COL & ACC_REF & IA5
          #  !ROW_COL & ACC_REF & IA15
          #  !ACC_REF & RCNTR5;
      IRAM6 = ROW_COL & ACC_REF & IA6
          #  !ROW_COL & ACC_REF & IA16
          #  !ACC_REF & RCNTR6;
      IRAM7 = ROW_COL & ACC_REF & IA7
          #  !ROW_COL & ACC_REF & IA17
          #  !ACC_REF & RCNTR7;
   END
END;

SYM GLB  D2  1  ;
   /////     ADDRESS MUX GLB2      /////
   SIGTYPE IRAM8 ASYNC OUT;
   SIGTYPE IRAM9 ASYNC OUT;
   EQUATIONS
      IRAM8 = ROW_COL & ACC_REF & IA8    /////  ROW SELECT   /////
          #  !ROW_COL & ACC_REF & IA18 /////  COLUMN SELECT   /////
          #  !ACC_REF & RCNTR8;               /////   REFRESH ADDR SELECT  /////
      IRAM9 = ROW_COL & ACC_REF & IA9
          #  !ROW_COL & ACC_REF & IA19
          #  !ACC_REF & RCNTR9;
   END
END;

SYM GLB  D5  1  ;
//////  REFRESH ROW COUNTER GLB0   /////
   SIGTYPE RCNTR0 REG OUT;
   SIGTYPE RCNTR1 REG OUT;
   SIGTYPE RCNTR2 REG OUT;
   SIGTYPE RCNTR3 REG OUT;
   EQUATIONS
      RCNTR0.PTCLK = !IRAS0;     /////    USE RAS AS THE COUNTER CLOCK   ////
      RCNTR0 = !RCNTR0 & !ACC_REF    /////   COUNT DURING REFRESH      /////
           #   RCNTR0 & ACC_REF;       /////    HOLD DURING ACCESS          /////
      RCNTR1 = (RCNTR1 & !ACC_REF)
            $$ ((RCNTR0 & !ACC_REF)
           #   (RCNTR1 & ACC_REF));
      RCNTR2 = (RCNTR2 & !ACC_REF)
            $$ ((RCNTR1 & RCNTR0 & !ACC_REF)
           #   (RCNTR2 & ACC_REF));
      RCNTR3 = (RCNTR3 & !ACC_REF)
            $$ ((RCNTR2 & RCNTR1 & RCNTR0 & !ACC_REF)
           #   (RCNTR3 & ACC_REF));
   END
END;
```

```
SYM GLB  D6  1  ;
   //////  REFRESH ROW COUNTER GLB1   /////
   SIGTYPE RCNTR4 REG OUT;
   SIGTYPE RCNTR5 REG OUT;
   SIGTYPE RCNTR6 REG OUT;
   SIGTYPE RCNTR7 REG OUT;
   EQUATIONS
      /////    USE RAS AS THE COUNTER CLOCK   ////
      RCNTR4.PTCLK = !IRAS0;
      RCNTR4 =  (RCNTR4 & !ACC_REF)
      /////  COUNT DURING REFRESH   /////
             $$ ((RCNTR3 & RCNTR2 & RCNTR1 & RCNTR0 & !ACC_REF)
             #   (RCNTR4 & ACC_REF));
      /////    HOLD DURING ACCESS      /////
      RCNTR5 = (RCNTR5 & !ACC_REF)
             $$ ((RCNTR4 & RCNTR3 & RCNTR2 & RCNTR1 & RCNTR0 & !ACC_REF)
             #   (RCNTR5 & ACC_REF));
      RCNTR6 = (RCNTR6 & !ACC_REF)
             $$ ((RCNTR5 & RCNTR4 & RCNTR3 & RCNTR2 & RCNTR1 & RCNTR0 & !ACC_REF)
             #   (RCNTR6 & ACC_REF));
      RCNTR7 = (RCNTR7 & !ACC_REF)
             $$ ((RCNTR6 & RCNTR5 & RCNTR4 & RCNTR3 & RCNTR2 & RCNTR1 & RCNTR0 &
                  !ACC_REF)
             #   (RCNTR7 & ACC_REF));
   END
END;

 SYM GLB  D7  1  ;
 //////  REFRESH ROW COUNTER GLB2   /////
   SIGTYPE RCNTR8 REG OUT;
   SIGTYPE RCNTR9 REG OUT;
   EQUATIONS
      RCNTR8.PTCLK = !IRAS0;     /////    USE RAS AS THE COUNTER CLOCK   ////
      RCNTR8 = (RCNTR8 & !ACC_REF)
             $$ ((RCNTR7 & RCNTR6 & RCNTR5 & RCNTR4
      ///// COUNT DURING REFRESH   /////
     & RCNTR3 & RCNTR2 & RCNTR1 & RCNTR0 & !ACC_REF) # (RCNTR8 & ACC_REF));
      /////   HOLD DURING ACCESS        /////
      RCNTR9 = (RCNTR9 & !ACC_REF)
             $$ ((RCNTR8 & RCNTR7 & RCNTR6 & RCNTR5 & RCNTR4 & RCNTR3 & RCNTR2 &
                   RCNTR1 & RCNTR0 & !ACC_REF)
              #   (RCNTR9 & ACC_REF));
   END
END;

SYM GLB  C7  1  ;
   /////     STATE BITS GLB      /////
   SIGTYPE ST0 REG OUT;
   SIGTYPE ST1 REG OUT;
   FJK11 (ST0,JST0,KST0,ICLK);
   FJK11 (ST1,JST1,KST1,ICLK);
   EQUATIONS
      JST0 = !ST1 & !ST0 & REFRESH;          /////   STATE BIT0 SET INPUT   /////
      KST0 = !ST1 & ST0 & SCNT1 & SCNT0;    /////    STATE BIT0 RESET INPUT   ///
//
      JST1 = !ST1 & !ST0 & !REFRESH & !IALE & IMIO_
```

```
          # !ST1 & !ST0 & !REFRESH & ACCESS;     /////   STATE BIT1 SET INPUT
/////
      KST1 = ST1 & !ST0 & SCNT1 & SCNT0
          # !ST1 & ST0 & SCNT1 & SCNT0;        /////   STATE BIT0 RESET INPUT
/////
   END
END;

SYM GLB  C6  1  ;
/////     STATE COUNTER BITS GLB      /////
   SIGTYPE SCNT0 REG OUT;
   SIGTYPE SCNT1 REG OUT;
   FJK11 (SCNT0,JSCNT0,KSCNT0,ICLK);
   FJK11 (SCNT1,JSCNT1,KSCNT1,ICLK);
   EQUATIONS
      JSCNT0 = !SCNT0 & ST1 & !ST0
           #  !SCNT0 & !ST1 & ST0;  /////   STATE COUNTER BIT0 SET INPUT /////
      KSCNT0 = SCNT0 & ST1 & !ST0
           #  SCNT0 & !ST1 & ST0
           #  ST1 & !ST0 & SCNT1 & SCNT0
           #  !ST1 & ST0 & SCNT1 & SCNT0;  /////STATE COUNTER BIT0 RESET INPUT
/////
      JSCNT1 = !SCNT1 & SCNT0 & ST1 & !ST0
           #  !SCNT1 & SCNT0 & !ST1 & ST0; ///// STATE COUNTER BIT1 SET INPUT
////
      KSCNT1 = SCNT1 & SCNT0 & ST1 & !ST0
           #  SCNT1 & SCNT0 & !ST1 & ST0
           #  ST1 & !ST0 & SCNT1 & SCNT0
           #  !ST1 & ST0 & SCNT1 & SCNT0;  ///// STATE COUNTER BIT0 RESET INPUT
/////
   END
END;

SYM GLB  C5  1  ;
   /////     CONTROL SIGNALS GLB0      /////
   SIGTYPE RFC REG OUT;
   SIGTYPE ACC_REF REG OUT;
   FJK11 (RFC,JRFC,KRFC,ICLK);
   FJK11 (ACC_REF,JACC_REF,KACC_REF,ICLK);
   EQUATIONS
      JRFC = !ST1 & ST0 & SCNT1 & !SCNT0;      /////  REFRESH COMPLETE SET
INPUT   /////
      KRFC = !ST1 & ST0 & SCNT1 & SCNT0; /////  REFRESH COMPLETE RESET INPUT  //
///
      JACC_REF = !ST1 & ST0 & SCNT1 & SCNT0
            #   IRESET;                /////   ACCESS/REFRESH SET INPUT   ////
/
      KACC_REF = !ST1 & !ST0 & REFRESH & !IRESET;/////ACCESS/REFRESH RESET INPUT
/////
   END
END;

SYM GLB  C1  1  ;
/////    ROW ADDRESS STROBE (RAS3,RAS2) GLB      /////
   SIGTYPE IRAS3 REG OUT;
   SIGTYPE IRAS2 REG OUT;
   EQUATIONS
      IRAS3 = !ST0 & !IA20 & IRAS3 & !IRESET   /////   REDUCED RAS3    /////
           # !ST1 & !IA21 & IRAS3 & !IRESET
           # !ST0 &  ST1 & SCNT0 & SCNT1 & IA20 & IA21 & !IRESET
           #  ST0 & !ST1 & SCNT0 & SCNT1 & !IRESET
           # !ST0 & !ST1 & IRAS3 & !IRESET
           #  ST0 & ST1 & IRAS3 & !IRESET
           #  SCNT1 & IRAS3 & !IRESET
           #  SCNT0 & IRAS3 & !IRESET;
      IRAS3.CLK = ICLK;

      IRAS2 = !ST0 & IA20 & IRAS2 & !IRESET    /////   REDUCED RAS2    /////
           # !ST1 & !IA21 & IRAS2 & !IRESET
           # !ST0 &  ST1 & SCNT0 & SCNT1 & !IA20 & IA21 & !IRESET
           #  ST0 & !ST1 & SCNT0 & SCNT1 & !IRESET
           # !ST0 & !ST1 & IRAS2 & !IRESET
           #  ST0 & ST1 & IRAS2 & !IRESET
           #  SCNT1 & IRAS2 & !IRESET
           #  SCNT0 & IRAS2 & !IRESET;
      IRAS2.CLK = ICLK;

   END
END;
SYM GLB  B7  1  ;
   /////    COLUMN ADDRESS STROBE (CAS0,CAS1) GLB0      /////
   SIGTYPE ICAS0 REG OUT;
   SIGTYPE ICAS1 REG OUT;
   FJK11 (ICAS0,JCAS0,KCAS0,ICLK);
   FJK11 (ICAS1,JCAS1,KCAS1,ICLK);
   EQUATIONS
      /////   CAS0 SET INPUT   /////
      JCAS0 = ST1 & !ST0 & !IA1 & !IA0 & SCNT1 & SCNT0
          #   IRESET;
      /////CAS0 RESET               INPUT /////
      KCAS0 = ST1 & !ST0 & !IA1 & !IA0 & !SCNT1 & SCNT0 & !IRESET;
      /////   CAS1 SET INPUT   /////
      JCAS1 = ST1 & !ST0 & !IA1 & IA0 & SCNT1 & SCNT0
         #   IRESET;
      /////CAS1 RESET INPUT /////
      KCAS1 = ST1 & !ST0 & !IA1 & IA0 & !SCNT1 & SCNT0 & !IRESET;
   END
END;

SYM GLB  B6  1  ;
   /////    COLUMN ADDRESS STROBE (CAS2,CAS3) GLB1      /////
   SIGTYPE ICAS2 REG OUT;
   SIGTYPE ICAS3 REG OUT;
   FJK11 (ICAS2,JCAS2,KCAS2,ICLK);
   FJK11 (ICAS3,JCAS3,KCAS3,ICLK);
   EQUATIONS
     JCAS2 = ST1 & !ST0 & IA1 & !IA0 & !SCNT1 & SCNT0     /////   CAS2 SET INPUT
/////
        #   IRESET;
     ///// CAS2 RESET INPUT              /////
     KCAS2 = ST1 & !ST0 & IA1 & !IA0 & SCNT1 & SCNT0 & !IRESET;
     JCAS3 = ST1 & !ST0 & IA1 & IA0 & !SCNT1 & SCNT0///// CAS3 SET INPUT  /////
        #   IRESET;
     ///// CAS3 RESET INPUT              /////
     KCAS3 = ST1 & !ST0 & IA1 & IA0 & SCNT1 & SCNT0 & !IRESET;
   END
END;
```

```
SYM GLB  B5  1  ;
/////     CONTROL SIGNALS (ACCESS,WRITE) GLB1      /////
   SIGTYPE ACCESS REG OUT;
   SIGTYPE IWREG REG OUT;
   FJK11 (ACCESS,JACCESS,KACCESS,ICLK);
   FJK11 (IWREG,JWREG,KWREG,ICLK);
   EQUATIONS
      JACCESS = !IALE & IMIO_;    /////    MEMORY ACCESS REQUEST SET INPUT   ////
/
      KACCESS = ST1 & !ST0 & SCNT1 & SCNT0;/////MEMORY ACCESS REQUEST RESET
INPUT/////
        JWREG = !ACCESS & IRW_           /////   WRITE REGISTER SET INPUT   /////
             #  ST1 & !ST0 & SCNT1 & SCNT0
             #  IRESET;
       KWREG = !ACCESS & !IRW_ & !IRESET;     /////    WRITE REGISTER RESET INPUT
/////
   END
END;

SYM GLB  B4  1  ;
   /////     CONTROL SIGNALS (ROW/COL,RDY)GLB2      /////
   SIGTYPE ROW_COL REG OUT;
   SIGTYPE IRDY REG OUT;
   FJK11 (ROW_COL,JROW_COL,KROW_COL,ICLK);
   FJK11 (IRDY,JRDY,KRDY,ICLK);
   EQUATIONS
   JROW_COL = ST1 & !ST0 & SCNT1 & SCNT0/////   ROW/COL SELECT SET INPUT   /////
          #  IRESET;
   KROW_COL = ST1 & !ST0 & !SCNT1 & SCNT0 & !IRESET/////ROW/COL SELECT RESET SET
              INPUT/////
       JRDY = ST1 & !ST0 & SCNT1 & !SCNT0;    /////   READY SET INPUT   /////
       KRDY = ST1 & !ST0 & SCNT1 & SCNT0;     /////   READY RESET INPUT   /////
   END
END;

SYM IOC  IO16  1  ;
   //   ADDR 12 I/O CELL W/REG. INPUT //
   XPIN  IO  XA12;
   ID11 (IA12,XA12,IICLK);
END;

SYM IOC  IO15  1  ;
   //  ADDR 11 I/O CELL W/REG. INPUT //
   XPIN  IO  XA11;
   ID11 (IA11,XA11,IICLK);
END;

SYM IOC  IO14  1  ;
   //  ADDR 10 I/O CELL W/REG. INPUT //
   XPIN  IO  XA10;
   ID11 (IA10,XA10,IICLK);
END;

SYM IOC  IO13  1  ;
   //  ADDR 9 I/O CELL W/REG. INPUT //
   XPIN  IO  XA9;
   ID11 (IA9,XA9,IICLK);
END;
```

```
SYM IOC  IO12  1  ;
   //  ADDR 8 I/O CELL W/REG. INPUT //
   XPIN  IO  XA8;
   ID11 (IA8,XA8,IICLK);
END;

SYM IOC  IO11  1  ;
   //  ADDR 7 I/O CELL W/REG. INPUT //
   XPIN  IO  XA7;
   ID11 (IA7,XA7,IICLK);
END;
SYM IOC  IO10  1  ;
   //  ADDR 6 I/O CELL W/REG. INPUT //
   XPIN  IO  XA6;
   ID11 (IA6,XA6,IICLK);
END;

SYM IOC  IO9  1  ;
   //  ADDR 5 I/O CELL W/REG. INPUT //
   XPIN  IO  XA5;
   ID11 (IA5,XA5,IICLK);
END;

SYM IOC  IO8  1  ;

   //  ADDR 4 I/O CELL W/REG. INPUT //
   XPIN  IO  XA4;
   ID11 (IA4,XA4,IICLK);
END;

SYM IOC  IO7  1  ;
   //  ADDR 3 I/O CELL W/REG. INPUT //
   XPIN  IO  XA3;
   ID11 (IA3,XA3,IICLK);

END;
SYM IOC  Y2  1  ;
   //  INPUT REGISTER CLOCK (ALE) //
   XPIN  CLK  XICLK;
   IB11 (IICLK,XICLK);
END;

SYM IOC  IO6  1  ;
   //  ADDR 2 I/O CELL W/REG. INPUT //
   XPIN  IO  XA2;
   ID11 (IA2,XA2,IICLK);
END;

SYM IOC  IO5  1  ;
  //  ADDR 1 I/O CELL W/REG. INPUT //
  XPIN  IO  XA1;
  ID11 (IA1,XA1,IICLK);
END;

SYM IOC  IO4  1  ;
   //  ADDR 0 I/O CELL W/REG. INPUT //
   XPIN  IO  XA0;
   ID11 (IA0,XA0,IICLK);
END;
```

```
SYM IOC  IO3  1  ;
   //  READY I/O CELL, OUTPUT //
   XPIN  IO  XRDY;
   OB11 (XRDY,IRDY);
END;

SYM IOC  IO2  1  ;
   //  ADDRESS LATCH ENABLE I/O CELL /
/
   XPIN  IO  XALE;
   IB11 (IALE,XALE);
END;

SYM IOC  IO1  1  ;
   //  MEMORY OR I/O ACCESS //
   XPIN  IO  XMIO_;
   IB11 (IMIO_,XMIO_);
END;

SYM IOC  IO0  1  ;
   //  READ WRITE SELECTION //
   XPIN  IO  XRW_;
   IB11 (IRW_,XRW_);
END;

SYM IOC  Y0  1  ;
   //  SYSTEM CLOCK INPUT //
   XPIN  CLK  XSYS_CLK LOCK 20;
   IB11 (ICLK,XSYS_CLK);
END;

SYM IOC  IO17  1  ;
   //  ADDR 13 I/O CELL W/REG. INPUT /
/
   XPIN  IO  XA13;
   ID11 (IA13,XA13,IICLK);
END;

SYM IOC  IO18  1  ;
   //  ADDR 14 I/O CELL W/REG. INPUT /
/
   XPIN  IO  XA14;
   ID11 (IA14,XA14,IICLK);
END;

SYM IOC  IO19  1  ;
   //  ADDR 15 I/O CELL W/REG. INPUT /
/
   XPIN  IO  XA15;
   ID11 (IA15,XA15,IICLK);
END;

SYM IOC  IO20  1  ;
   //  ADDR 20 I/O CELLW/REG. INPUT //
   XPIN IO XA20;
   ID11 (IA20,XA20,IICLK);
END;
```

```
SYM IOC IO21 1;
   //  ADDR 21 I/O CELL W/REG.INPUT //
   XPIN IO XA21
   ID11 (IA21,XA21,IICLK);
END;

SYM IOC IO22 1;
   XPIN   IO  XRESET;
   IB11  (IRESET, XRESET);
END;

SYM  IOC  IO23   1 ;
     XPIN   IO  XREFRESH;
     IB11  (REFRESH, XREFRESH);
END;

SYM  IOC  IO24   1 ;
     XPIN   IO  XRAM0;
     OB11  (XRAM0, IRAM0);
END;

SYM  IOC  IO25   1 ;
     XPIN   IO  XRAM1;
     OB11  (XRAM1, IRAM1);
END;

SYM  IOC  IO26   1 ;
     XPIN   IO  XRAM2;
     OB11  (XRAM2, IRAM2);
END;

SYM  IOC  IO27   1 ;
     XPIN   IO  XRAM3;
     OB11  (XRAM3, IRAM3);
END;

SYM  IOC  IO28   1 ;
     XPIN   IO  XRAM4;
     OB11  (XRAM4, IRAM4);
END;
SYM  IOC  IO29   1 ;
     XPIN   IO  XRAM5;
     OB11  (XRAM5, IRAM5);
END;

SYM  IOC  IO30   1 ;
     XPIN   IO  XRAM6;
     OB11  (XRAM6, IRAM6);
END;
```

```
SYM  IOC  IO31   1 ;
     XPIN   IO  XRAM7;
     OB11  (XRAM7, IRAM7);
END;

SYM  IOC  IO32   1 ;
     XPIN   IO  XRAM8;
     OB11  (XRAM8, IRAM8);
END;

SYM  IOC  IO33   1 ;
     XPIN   IO  XRAM9;
     OB11  (XRAM9, IRAM9);
END;

SYM  IOC  IO34   1 ;
     XPIN   IO  XST0;
     OB11  (XST0, ST0);
END;

SYM  IOC  IO36   1 ;
     XPIN   IO  XST1;
     OB11  (XST1, ST1);
END;

SYM  IOC  IO38   1 ;
     XPIN   IO  XSCNT0;
     OB11  (XSCNT0, SCNT0);
END;

SYM  IOC  IO40   1 ;
     XPIN   IO  XSCNT1;
     OB11  (XSCNT1, SCNT1);
END;

SYM  IOC  IO41   1 ;
     XPIN   IO  XACCESS;
     OB11  (XACCESS, ACCESS);
END;
SYM  IOC  IO42   1 ;
     XPIN   IO  XIWREG;
     OB11  (XIWREG, IWREG);
END;

SYM  IOC  IO43   1 ;
     XPIN   IO  XROW_COL;
     OB11  (XROW_COL, ROW_COL);
END;
```

```
SYM  IOC  IO44   1 ;
    XPIN   IO  XIRDY;
    OB11  (XIRDY, IRDY);
END;
SYM  IOC  IO45   1 ;
    XPIN   IO  XRFC;
    OB11  (XRFC, RFC);
END;

SYM  IOC  IO46   1 ;
    XPIN   IO  XACC_REF;
    OB11  (XACC_REF, ACC_REF);
END;
```

Lattice™
Semiconductor
Corporation

E^2CMOS, GAL, ispGAL, ispLSI, pLSI, pDS, Silicon Forest, UltraMOS, Lattice Logo, L with Lattice Semiconductor Corp. and L (Stylized) are registered trademarks of Lattice Semiconductor Corporation (LSC). The LSC Logo, Generic Array Logic, In-System Programmability, In-System Programmable, ISP, ispATE, ispCODE, ispDOWNLOAD, ispGDS, ispStarter, ispSTREAM, ispTEST, ispTURBO, Latch-Lock, pDS+, RFT, Total ISP and Twin GLB are trademarks of Lattice Semiconductor Corporation. ISP is a service mark of Lattice Semiconductor Corporation. All brand names or product names mentioned are trademarks or registered trademarks of their respective holders.

Lattice Semiconductor Corporation (LSC) products are made under one or more of the following U.S. and international patents: 4,761,768 US, 4,766,569 US, 4,833,646 US, 4,852,044 US, 4,855,954 US, 4,879,688 US, 4,887,239 US, 4,896,296 US, 5,130,574 US, 5,138,198 US, 5,162,679 US, 5,191,243 US, 5,204,556 US, 5,231,315 US, 5,231,316 US, 5,237,218 US, 5,245,226 US, 5,251,169 US, 5,272,666 US, 5,281,906 US, 5,295,095 US, 5,329,179 US, 5,331,590 US, 5,336,951 US, 5,353,246 US, 5,357,156 US, 5,359,573 US, 5,394,033 US, 5,394,037 US, 5,404,055 US, 5,418,390 US, 5,493,205 US, 0194091 EP, 0196771B1 EP, 0267271 EP, 0196771 UK, 0194091 GB, 0196771 WG, P3686070.0-08 WG. LSC does not represent that products described herein are free from patent infringement or from any third-party right.

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