Minesweeper

Final Project Report
December 12, 2002
E155

Gigi Au and Daniel Vaughan

Abstract:

Minesweeper is a game where the user is challenged to identify hidden mines and clear safe cells. Since the existing version of the game is only available for a computer, users who are bored of staring at the monitor may want to play the game using a different display. This project prototypes minesweeper using a keypad, an HC11 microcontroller, a Spartan FPGA, and a perforation board containing a grid of 'minefield' LEDs, six seven-segment game status displays, and win/lose LEDs. The user uses a keypad to navigate through the grid, clear or flag mines, or reset the game. The FPGA decodes the keypress and sends it to the microcontroller, which controls the game logic. The microcontroller, in turn, sends data back to the FPGA, which decodes the input signals and routes them to the LED grid and game status displays.

Introduction

The game of minesweeper requires a user to navigate through a grid, while systematically flagging "mines" and clearing safe cells. The user wins by successfully flagging all mines within the grid. The user loses by either flagging a cell that does not contain a mine *or* clearing a cell that does contain a mine.

This project involves implementing minesweeper using an M68HC11 evaluation board (EVB), a Spartan xcs10-3pc84 FPGA, and an external perforated board. The EVB controls the game logic, while the FPGA sends user input to the EVB and game display updates to the perforation board. Mounted on the perforation board is a keypad, a 5x6 grid of 'minefield' LEDs, a 'win' LED, a 'lose' LED, and four dual 7-segment displays to show current game information (row and column indices, time, and number of mines adjacent to current position). Figure 1 below illustrates the layout of the game. Please refer to Appendix A for a top level block diagram of the game control.

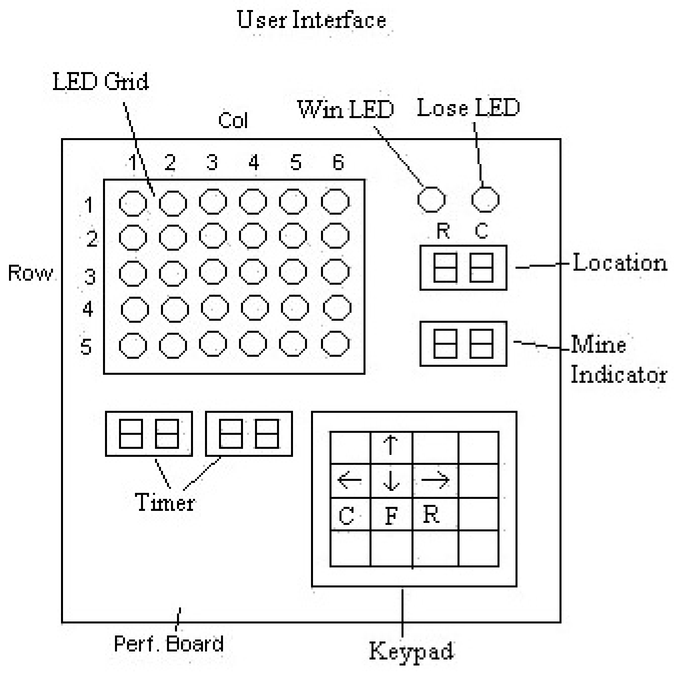


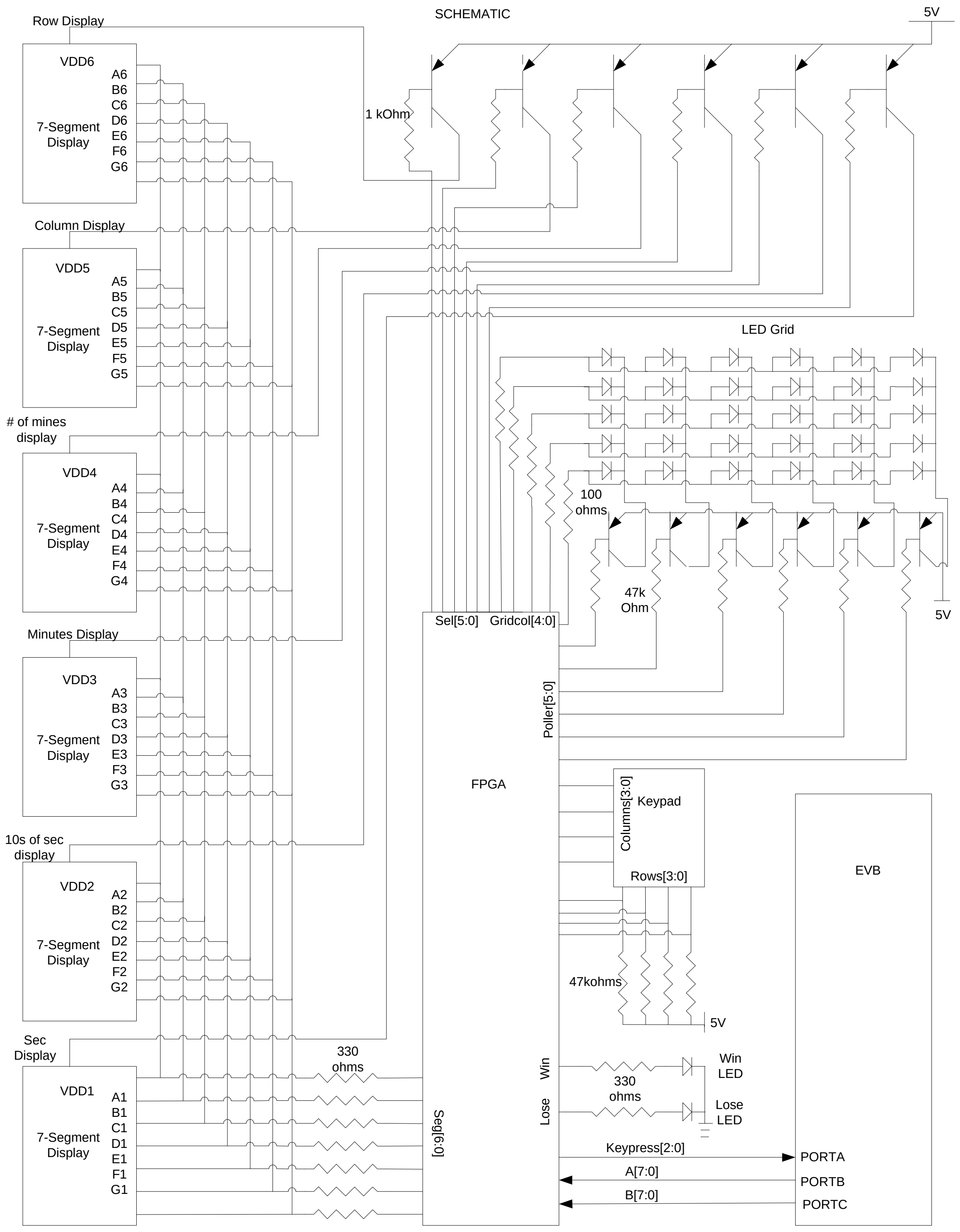
Figure 1: Minesweeper Layout

Schematics

The hardware for the project is placed on a breadboard and a perforation board. The breadboard consists of the FPGA, transistors, and resistors. The perforation board consists of the 5x6 grid of LEDs, six seven-segment displays, a keypad, a win LED and a lose LED. The LED grid is illuminated using a polling method: LED status information is sent to the appropriate column while that column is being polled. The six seven-segment displays are time-multiplexed using a selector signal driven by the system clock and transistors to switch between the displays. A similar polling method is also used to interpret the button press on the keypad.

The FPGA and EVB communicate through Ports A, B and C. The FPGA sends the keypress signal to the EVB through Port A. The EVB determines the new game status update and then sends 2 bytes of data, A and B, back to the FPGA through Ports B and C, respectively.

A schematic of the integrated system appears on the following page.


SCHEMATIC
5V
Row Display
VDD6
A6
B6
C6
D6
E6
F6
G6
7-Segment Display
1 kOhm
Column Display
VDD5
A5
B5
C5
D5
E5
F5
G5
7-Segment Display
LED Grid
# of mines display
VDD4
A4
B4
C4
D4
E4
F4
G4
7-Segment Display
100 ohms
47k Ohm
5V
Minutes Display
Sel[5:0]
Gridcol[4:0]
VDD3
A3
B3
C3
D3
E3
F3
G3
7-Segment Display
Poller[5:0]
FPGA
Columns[3:0]
Keypad
Rows[3:0]
EVB
10s of sec display
VDD2
A2
B2
C2
D2
E2
F2
G2
7-Segment Display
47kohms
5V
Sec Display
330 ohms
VDD1
A1
B1
C1
D1
E1
F1
G1
7-Segment Display
Seg[6:0]
Win
Lose
330 ohms
Win LED
Lose LED
Keypress[2:0]
A[7:0]
B[7:0]
PORTA
PORTB
PORTC

Microcontroller Design

Logical control of minesweeper is conducted within the HC11 Microcontroller. The breakdown of the HC11 game management code is organized as follows:

1. Definitions and Initializations
2. Pattern Generation
3. Start Routine
    a. Position Check Routine
    b. Write output data
    c. Read input data
    d. Determine Keypress
4. Clear Routine
5. Flag Routine
6. Navigation Routines
7. Win/Lose Routines

An .rst file of our assembly code appears in Appendix B, while a block diagram of the EVB game logic appears in Appendix C. The following descriptions will walk the reader through the primary routines and algorithms used in the HC11 code.

Port Definitions

Define memory locations for PORTA, PORTB, PORTC, DDRC, and TCNT.

Variable Definitions

Patterns are defined for: navigation directions, clear, flag, reset, loser, winner, and various masks. Memory locations $C100 - $C167 are set aside for the mine pattern plus a perimeter of empty cells, where each memory location contains a 'cell cleared' indicator and a 'mine' indicator in the least two significant bits, respectively. Memory locations are also reserved for storing the win/lose signal (WINLOSE), row and column indices (RINDEX, CINDEX), blink and status signals (BLINK, STATUS), the number of mines surrounding a cell (NUMMINE), and the signal that enables the number of mines to be displayed (NUMEN). Additional locations are reserved for various counters used internally to generate the output signals.

Pattern Generator

We have developed a routine for generating a seemingly random mine sequence. A mine pattern is generated by iteratively examining the lowest 8 bits of the EVB timer for 30 cycles. For a single cycle, the timer bits are added to a seed, and this value becomes the new seed for each successive cycle. A mine will be placed in the memory location corresponding to the cycle number only if the seed is less than decimal# –78. This corresponds to a probability of about 1/5 because the range of seed values is (-128:127). Alternatively, a mine will not be designated in that location if the seed is greater than or equal to -78. After 30 cycles, the total number of mines should average 6 (but it is not restricted to this value!). At the end of the final cycle, the information for each cell will be stored in the least significant bit of the appropriate memory location.

The pattern generator could be improved by multiplying the timer by a relatively prime seed, adding an additional relatively prime number, and then setting that value as the new seed for each cycle.

Initializations

The row and column indices are initially both set to '1' (corresponding to the upper left corner of the grid). The LEDs are all initialized to '1' (on), blink and win/lose signals to '0' (off) and the enable signal to display the number of mines to '0' (off). Various counters, such as the total mine counter and the flag counter are also initially set to zero. In addition, a value of zero is stored in a perimeter of memory locations around the mine grid – this allows for the use of a common procedure to recursively count adjacent mines.

Start Routine

Generation of 8-bit data (A)

The program begins by generating an 8-bit output that consists of the column and row indices in the 6 most significant bits, followed by the blink signal and on/off status signal in the least two significant bits. This data is sent through PORTB to the FPGA to control the LED grid.

A = [Col(2:0) Row(2:0) BLINK STATUS]

Check for Cleared Cell

The program uses the current row and column indices to reference the byte stored in that particular memory location. Since the number of mines around a cell is not supposed to be displayed if that cell has not yet been uncovered, the second least significant bit must be examined for a '1' to determine whether NUMMINE should be displayed. If this bit is a '1,' then the user has uncovered this cell previously and the NUMEN signal will be set to 1 so that the number of mines can be displayed. In addition, the value of NUMMINE is also determined so that the appropriate value can be sent to the FPGA.

To achieve full functionality, this routine must also include a check for whether a particular mine has been flagged. Currently, once a user flags a mine, the BLINK value is set to 1 but is not stored in the byte of information corresponding to that grid location. As a result, when a user moves away from the current location after flagging a mine, the BLINK signal remains on and causes every cell the user lands on to blink. To fix this problem, we could simply store a “flagged” bit in the memory location for each cell and use that value to adjust the blink signal appropriately.

Generation of 8-bit data (B)

First, all bits of PORTC are set to output mode. The 8-bit output data (B) contains a reset command in the most significant bit, the number of mines in the next four bits, the enable signal needed to display the number of mines, and the WINLOSE signal in the two least significant bits. This data is sent through PORTC to the FPGA.

B = [Reset Adj(3:0) AdjMineEnable WinLose(1:0)]

Determination of Keypress

The FPGA sends data containing the keypress through PORTA to the EVB. The 3 least significant bits of the data are examined to determine if the key pressed was ‘reset’, ‘flag’, ‘clear’, or one of the navigation directions. Depending on the keypress, the program jumps to the respective routine.

In order to ensure that the HC11 only executes one routine each time a button is pressed, we implemented a “handshaking” routine. This block of code stores the current keypress in memory and keeps the output data the same as long as the user is holding down a button. Once the user lets up on the button, a zero is sent to the EVB. The next keypress will then be evaluated appropriately.

Check for Reset

If the reset button has been pressed, the program jumps back to the pattern generation routine and a “Reset” signal is sent through Port C to tell the FPGA to reset all the displays and the LED grid to their initial states.

Flag Routine

The flag routine uses the row and column indices to determine the referenced memory location that stores the mine status for each cell. If the flagged cell is not a mine, the program jumps to the lose routine and the game resets. Otherwise, the blink signal is set to ‘1’ and the flag count is incremented to keep track of the number of correctly flagged mines. If the flag count equals the total number of mines in the grid, the user has completed the game, and the program jumps to the win routine, after which the game resets. As long as the number of flagged mines is less than the number of total mines, the flag count is stored in FLGCNT and the program returns to the start routine.

As mentioned above, the flag status is not stored into memory for each grid location. We should have added a few lines of code in this routine to store the flag status of each grid point to the associated cell.

Clear Routine

The clear routine first uses the row and column indices to determine the referenced memory location that stores the mine status for each cell. If the cleared cell is a mine, the program jumps to the lose routine and the game resets. Otherwise, the user has cleared a safe cell, and the status signal is set to '0' to turn off the respective LED in the grid that corresponds to the referenced memory location. In addition, a '1' is loaded into the second least significant bit (recall that least significant bit contains 'mine status (0 or 1)') to indicate that the cell has been uncovered. This data is stored into memory.

To check for adjacent mines, the clear routine first initializes a counter 'Y' and then determines the current user position. Because we have generated a perimeter of zeros around the grid, we simply use one routine that checks for mines in each of the 8 adjacent grid locations. At the end of this routine, the number of adjacent mines is stored into the memory location for the current grid position.

Navigation Routine

If a navigation button is pressed, the program determines the direction in which to move the current position. This is accomplished by changing the column or row index in accordance with the navigation direction. The program also prohibits the row and column indices from exceeding the size of the grid. In other words, if the user position were along the leftmost column, pressing the left navigation button would have no effect. After the routine has determined the new position, it adjusts the appropriate index and returns to the main program.

Win/Lose Routines

As described above, flagging incorrectly or clearing a space containing a mine sends the program to the 'Lose' routine. Likewise, correctly flagging all the mines sends the program to the 'Win' routine. In the lose routine, a 'Lose' LED lights up on the game board to remind the user they've just been blown to pieces, after which the game should be reset. Similarly, in the win routine, a 'Win' LED is illuminated for one second to indicate a successful mission, after which the user resets the game.

FPGA Design

(Please refer to Appendix D for the FPGA block diagram and associated FSM digrams)

Introduction

The FPGA will be divided into two primary sections. One section will decode a keypad input into a specified format and send this information to the EVB. The other section will input control signals from the EVB and use this information to update the LED grid and the 7-segment displays on the perforation board.

Section 1: Key decoder

Upon receiving a row input from the keypad, the key decoder will determine the corresponding column position and decode this information once the signal has been debounced. The decoded keypress will then be sent directly to the EVB for processing.

Section 2: Display Multiplexer and Grid Poller

After processing the keypress, the EVB will send two bytes of data back to the FPGA containing information on how to control the grid and the various displays. The first byte (A), as described in the Assembly Code section, contains bits that specify which grid point to change (RINDEX, CINDEX), and how to change it (BLINK, STATUS). The second byte (B) contains a reset signal, the number of adjacent mines and a corresponding enable signal, and a two-bit win/lose signal. The FPGA signal control can be divided into three parts: the clock divider-selector, the grid controller, and the display multiplexer.

Clock Divider/Selector

To ensure that blurring does not occur, a divided clock signal will be used to multiplex the displays. The displays are controlled by a six-input multiplexer while the LED grid is illuminated using a polling method. A one-hot logic scheme is used to select among the inputs. To generate the selector signal for each multiplexer, the first step is to tap out the lowest 11 bits of the system clock. This will yield a divided clock signal operating at approximately 1 kHz. By choosing six 11-bit reference numbers that are approximately evenly spaced apart, it is ensured that the multiplexer inputs will be selected at even intervals. Before entering the multiplexers, the six 11-bit numbers will be encoded using one-hot logic so that the appropriate signals can be selected.

Grid Controller

The grid controller inputs the A byte and outputs six column vectors to be multiplexed and sent to the LED grid. The grid controller uses the reference position information stored in byte A to determine which LED to alter, and the blink and status bits are used to determine how the selected LED is to be altered (e.g. ON → OFF or ON→ BLINK).

Display Multiplexer

The display multiplexer extracts the row and column indices from byte A, the number of mines from byte B, and timer information (minutes, 10's of seconds, and seconds) from the time decoder. These six inputs are multiplexed using selector signals that are generated by slowing down the system clock. The selected signal is then decoded and sent to the appropriate seven-segment display.

Results

For the most part, the game meets the original specifications: the perforation board and game components behave correctly, the FPGA interfaces correctly with the EVB and perforation board, and the HC11 code is reliable except for the flag routine. After loading the .s19 file into the HC11 and running power to the FPGA, the game is ready to operate. Upon reset, the LEDs all turn on, and the game timer starts over.

Perforation Board

The perforation board served as an excellent template for mounting the various displays and routing many of the wires. Color-coded wires were used to tie together common segments of the digital displays, which greatly reduced the number of wires crossing between the board and the FPGA. A similar scheme was used to illuminate the LED grid. In retrospect, a better-planned wiring diagram would have eliminated much of the wire clutter. Most importantly, though, the board and game components operated appropriately.

FPGA

The FPGA also operates properly. The keypress is successfully decoded and sent to the EVB. The seven segment displays show the appropriate values of row and column index, number of adjacent mines, and game time. The win and lose LEDs both function appropriately. The grid of LEDs accurately displays the current game status – LEDs can be turned on, off, or to a blinking state. One unusual characteristic of the display that could not be accounted for is the tendency for the ‘seconds’ timer to bleed onto the ‘row index’ display. This may have to do with the time multiplexing rate used to illuminate the 6 displays.

EVB

Within the EVB, the mine pattern is generated and stored in the proper memory locations. Each of the EVB routines except for the flag routine behaves normally: the navigation routines adjust user position appropriately, the clear routine sends the correct status signal to the FPGA and outputs the appropriate number of adjacent mines plus an enable signal. The win/lose routines also function as planned. The position check routine accurately determines whether the current position has been cleared and decides whether to display the number of adjacent mines.

The primary unresolved problem with the HC11 code lies in the flag routine. By not storing the flag status of each grid point, the BLINK signal that is sent to the FPGA remains high indefinitely following the first successful flag. As mentioned before, storing flag data in the memory location for each grid point could easily fix this issue.

References:

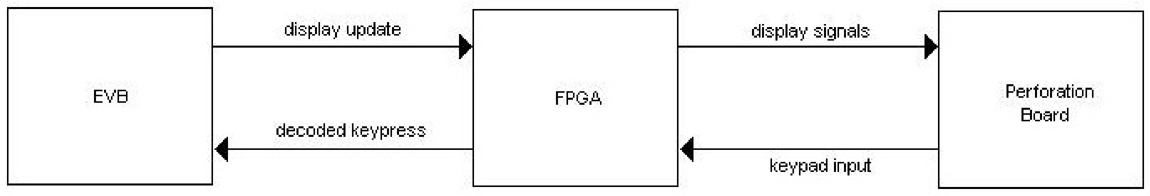
[1] Franzon, Paul D. and David R. Smith. *Verilog Styles for Synthesis of Digital Systems.* Upper Saddle River, New Jersey: Prentice Hall, 2000.

[2] *M68HC11EVB Evaluation Board User's Manual*. First Edition. Motorola, Inc. 1986.

Parts List

Part	**Source**	**Vendor Part #**	**Price**
Perforated Board	MarVac	4700T	$15.97

Appendix A: Top Level Block Diagram



Appendix B: HC11 Assembly Code

```
0001                      * mineswpr.asm -- Minesweeper Game Control
0002                      *
0003                      * Authors: Daniel Vaughan (dvaughan@hmc.edu) and Gigi Au (gwau@hmc.edu)
0004                      * Date:     December 10, 2002
0005                      *
0006                      * This file inputs a keypad command through Port A from the FPGA and determines
0007                      * how to update the LED grid based upon the nature of the command.
0008                      * Output is stored in two bytes and sent out through ports B and C.
0009                      *
0010                      *
0011                      * Code Layout:
0012                      *
0013                      *       Subsection                           Location
0014                      *______________________________________________________
0015                      *       Port Definitions....................0031
0016                      *       Reserved Memory Locations...........0042
0017                      *       Program Masks.......................0073
0018                      *       Pattern Generator...................0100
0019                      *       Initializations.....................0152
0020                      *       Start routine.......................0200
0021                      *       Flag routine........................0308
0022                      *       Clear routine.......................0344
0023                      *       Win/Lose routine....................0522
0024                      *       Navigation routines.................0535
0025                      *______________________________________________________
0026                      *
0027                      *
0028                      *
0029
0030                      **************************
0031                      *   Port Definitions     *
0032                      **************************
0033
0034 1000                 PORTA    EQU     $1000
0035 1004                 PORTB    EQU     $1004
0036 1003                 PORTC    EQU     $1003
0037 1007                 DDRC     EQU     $1007
0038 100f                 TCNT     EQU     $100F
0039
0040
0041                      ************************************
0042                      *    Reserved Memory Locations     *
0043                      ************************************
0044
0045                      * Registers are reserved for the mine pattern
0046
0047 c032                 WINLOSE  EQU     $C032
0048
0049 c033                 RINDEX   EQU     $C033          * Store row number in memory location $33
0050 c043                 RITEMP   EQU     $C043          * Temporary rindex storage
0051 c034                 CINDEX   EQU     $C034          * Store col number in memory location $34
0052 c044                 CITEMP   EQU     $C044          * Temporary cindex storage
0053 c045                 KPTEMP   EQU     $C045
0054
0055 c035                 BLINK    EQU     $C035          * Blink LED command
0056 c036                 STATUS   EQU     $C036          * LED on/off command
0057 c037                 REF1     EQU     $C037          * index reference
0058 c038                 REF2     EQU     $C038          * index reference
0059 c03a                 NUMMIN1  EQU     $C03A          * least significant byte of NUMMINE, number of adjacent mines
0060 c039                 NUMMINE  EQU     $C039          * number of adjacent mines
0061 c03b                 TOTMINE  EQU     $C03B          * total number of mines
0062 c03c                 NUMEN    EQU     $C03C          * enable number of adj. mines display (on/off)
0063 c03d                 FLGCNT   EQU     $C03D          * number of flagged mines counter
0064 c03e                 MINECT   EQU     $C03E          * number of adj. mines counter
0065
0066 c046                 SEED     EQU     $C046          * pattern generator seed number
0067 c048                 MEMX     EQU     $C048          * memory to store current row position
0068 c049                 MEMX1    EQU     $C049
0069 c050                 LOOPNUM  EQU     $C050          * memory to store loop number in pattern gen.
```

```
c052                    TEMP     EQU     $C052

                        *********************************
                        *        Program Masks        *
                        *********************************

0002                    UP               EQU     %00000010       * Masks for navigation directions
0005                    DOWN     EQU     %00000101
0004                    LEFT     EQU     %00000100
0001                    RIGHT    EQU     %00000001

0003                    CLEAR    EQU     %00000011       * Masks for action selections
0007                    FLAG     EQU     %00000111
0006                    RESET    EQU     %00000110
0000                    NOPRESS  EQU     %00000000

0002                    CLRMASK  EQU     %00000010       * Cleared location check mask
0003                    CLRMSK2  EQU     %00000011
00ff                    PCMASK   EQU     %11111111       * Port C --> output mask
0007                    NAVMASK  EQU     %00000111       * navigation button mask
0005                    FIVE     EQU     %00000101       * number five
0006                    SIX      EQU     %00000110       * number six
0095                    TESTNUM  EQU     %10010101       * Pattern generator seed (dec. #149)

0002                    LOSER    EQU     %00000010       * lose mask
0001                    WINNER   EQU     %00000001       * win mask



                        **************************
                        *   Pattern Generator    *
                        **************************

d100                             ORG     $D100

d100 ce 00 00           pattern  LDX     #$00            * Initialize row counter
d103 18 ce 00 01                 LDY     #$01            * Initialize row + column counter
d107 86 95                       LDAA    #TESTNUM        * Store test seed (#149) in memory
d109 b7 c0 46                    STAA    SEED
d10c 86 00                       LDAA    #$00
d10e b7 c0 3b                    STAA    TOTMINE
d111 b6 10 0f           rowloop  LDAA    TCNT            * Load lower 8 bits of counter
d114 8b 95                       ADDA    #TESTNUM        * Add to test seed
d116 b7 c0 52                    STAA    TEMP            * Store in memory
d119 c6 10                       LDAB    #$10            * Increment row address (2nd least sign. hex digit)
d11b 3a                          ABX
d11c 8c 00 50                    CPX     #$50            * Stop incrementing at bottom row of grid
d11f 2e 4d                       BGT     start
d121 18 ce c0 01                 LDY     #$C001          * Initialize row + column counter
d125 ff c0 48                    STX     MEMX            * Store row number
d128 f6 c0 49                    LDAB    MEMX1
d12b 18 3a                       ABY                     * Use Y to store row + column
d12d 86 00                       LDAA    #$00            *set column loop counter to 0
d12f b7 c0 50                    STAA    LOOPNUM
d132 b6 10 0f           colloop  LDAA    TCNT
d135 8b 95                       ADDA    #TESTNUM
d137 b7 c0 52                    STAA    TEMP
d13a b6 c0 50                    LDAA    LOOPNUM         * stop incrementing at far right column of grid
d13d 4c                          INCA
d13e 81 06                       CMPA    #$06
d140 2e cf                       BGT     rowloop
d142 b7 c0 50                    STAA    LOOPNUM
d145 c6 01                       LDAB    #$01            * Increase column position by 1
d147 18 3a                       ABY
d149 b6 10 0f                    LDAA    TCNT            * Load bottom 8 bits of timer in accumulator A
d14c f6 c0 46                    LDAB    SEED            * Load "random" seed into accumulator B
d14f 1b                          ABA
d150 b7 c0 46                    STAA    SEED            * Add timer and seed to form new seed
d153 81 b2                       CMPA    #$B2            * Compare (-128 -- 127) to (-78)
d155 2e 0f                       BGT     setzero         * If greater than -78 (4/5 probability), set mine = 0
d157 86 01                       LDAA    #$01            *
d159 18 a7 ff                    STAA    $FF,Y           * Otherwise (1/5 probability), set mine = 1
d15c f6 c0 3b                    LDAB    TOTMINE
d15f 5c                          INCB
d160 f7 c0 3b                    STAB    TOTMINE         * Store the number of total mines in TOTMINE
d163 7e d1 32                    JMP     colloop         * This will be generally around 6, but the unknown
```

```
d166 86 00              setzero  LDAA    #$00            * number makes the game a little more challenging
d168 18 a7 ff                    STAA    $FF,Y
d16b 7e d1 32                    JMP     colloop


                        *****************************************
                        *                Initializations             *
                        *****************************************

d16e 86 01              start    LDAA    #$01            *
d170 b7 c0 33                    STAA    RINDEX          * Set initial row index to one
d173 b7 c0 34                    STAA    CINDEX          * Set initial column index to one
d176 b7 c0 36                    STAA    STATUS          * Set initial Status signal to one (all LEDs on)


d179 86 00                       LDAA    #$00            *
d17b b7 c0 35                    STAA    BLINK           * Set Blink command to 'off'
d17e b7 c0 32                    STAA    WINLOSE         * Set WIN/LOSE command to 'off'
d181 b7 c0 3c                    STAA    NUMEN           * Set number of adjacent mines display to 'off'
d184 b7 c0 3d                    STAA    FLGCNT          * Set flag counter to 0
d187 b7 c0 3e                    STAA    MINECT          * Set adjacent mine counter to 0
d18a b7 c0 45                    STAA    KPTEMP          * Set initial keypress to 0
d18d b7 c0 50                    STAA    LOOPNUM         * Set LOOPNUM to 0
d190 b7 c0 39                    STAA    NUMMINE         * Set number of mines to 0
d193 b7 c0 3a                    STAA    NUMMIN1         *

d196 b7 c1 00                    STAA    $C100           * Create perimeter of zeros around mine grid to
d199 b7 c1 01                    STAA    $C101           * simplify adjacent mine counting procedure
d19c b7 c1 02                    STAA    $C102           *
d19f b7 c1 03                    STAA    $C103           * Actual mine sequence stored in
d1a2 b7 c1 04                    STAA    $C104           *       Row 1:  $C111 - $C116
d1a5 b7 c1 05                    STAA    $C105           *       Row 2:  $C121 - $C126
d1a8 b7 c1 06                    STAA    $C106           *       Row 3:  $C131 - $C136
d1ab b7 c1 07                    STAA    $C107           *       Row 4:  $C141 - $C146
d1ae b7 c1 10                    STAA    $C110           *       Row 5:  $C151 - $C156
d1b1 b7 c1 20                    STAA    $C120
d1b4 b7 c1 30                    STAA    $C130
d1b7 b7 c1 40                    STAA    $C140
d1ba b7 c1 50                    STAA    $C150
d1bd b7 c1 60                    STAA    $C160
d1c0 b7 c1 17                    STAA    $C117
d1c3 b7 c1 27                    STAA    $C127
d1c6 b7 c1 37                    STAA    $C137
d1c9 b7 c1 47                    STAA    $C147
d1cc b7 c1 57                    STAA    $C157
d1cf b7 c1 67                    STAA    $C167
d1d2 b7 c1 61                    STAA    $C161
d1d5 b7 c1 62                    STAA    $C162
d1d8 b7 c1 63                    STAA    $C163
d1db b7 c1 64                    STAA    $C164
d1de b7 c1 65                    STAA    $C165
d1e1 b7 c1 66                    STAA    $C166

                        ************************************
                        *                                         *
                        *        Start routine           *
                        *                                         *
                        ************************************

                        *******************************
                        *  Current Position Cleared?  *
                        *******************************

d1e4 bd d3 ce           nav      JSR     posclr          * check if current position has been cleared

                        ********************************
                        * send data out through PORT B *
                        ********************************

d1e7 b6 c0 34                    LDAA    CINDEX          * Generate an eight bit output with
d1ea 48                          LSLA                    * column index in [7:5], row index in
d1eb 48                          LSLA                    * [4:2], Blink in bit 1, and Status in
d1ec 48                          LSLA                    * bit 0
d1ed bb c0 33                    ADDA    RINDEX
d1f0 48                          LSLA
d1f1 bb c0 35                    ADDA    BLINK
```

```
d1f4 48                          LSLA
d1f5 bb c0 36                    ADDA    STATUS
d1f8 b7 10 04                    STAA    PORTB           * Send information out over port B



                        ********************************
                        * send data out through PORT C *
                        ********************************

d1fb 86 ff              skipc    LDAA    #PCMASK         * Set port C to all output mode
d1fd b7 10 07                    STAA    DDRC
d200 b6 c0 3a                    LDAA    NUMMIN1         * Generate a seven bit output with the number of mines
d203 48                          LSLA                    * adjacent to current position in [6:3]
d204 bb c0 3c                    ADDA    NUMEN           * NUMEN in bit 2
d207 48                          LSLA
d208 48                          LSLA
d209 bb c0 32                    ADDA    WINLOSE         * WINLOSE in bits [1:0]
d20c b7 10 03                    STAA    PORTC           * Send information out over port C
d20f 86 00                       LDAA    #$00            * Set enable signal to 0
d211 b7 c0 3c                    STAA    NUMEN

d214 b6 c0 32                    LDAA    WINLOSE         * Branch to keypress if WINLOSE = 0
d217 81 00                       CMPA    #$00
d219 27 1b                       BEQ     keyprs

d21b 18 ce 00 00        onesec   LDY     #$0000          * Display the win or lose LED for 1 second
d21f ce 00 00           one_st   LDX     #$0000          * after user wins or loses
d222 18 8c 00 2a                 CPY     #$002A
d226 27 0b                       BEQ     pat2
d228 18 08                       INY
d22a 8c 0e 4e           one_lp   CPX     #$0E4E
d22d 27 f0                       BEQ     one_st
d22f 08                          INX
d230 7e d2 2a                    JMP     one_lp
d233 7e d1 00           pat2     JMP     pattern



                        ************************
                        *  Determine Keypress  *
                        ************************

                        * Evaluates keypress and jumps to appropriate subroutine


d236 b6 10 00           keyprs   LDAA    PORTA           * Load value sent from FPGA
d239 84 07                       ANDA    #NAVMASK        * Only look at bottom three bits
d23b 81 00                       CMPA    #NOPRESS        * If no button being pressed, keep sending same data
d23d 26 06                       BNE     noprs           * out ports B and C
d23f b7 c0 45                    STAA    KPTEMP
d242 7e d1 e4                    JMP     nav
d245 f6 c0 45           noprs    LDAB    KPTEMP          * Load previous keypress
d248 11                          CBA                     * If button being held down, keep sending same data
d249 26 03                       BNE     holdon          * out ports B and C
d24b 7e d1 e4                    JMP     nav
d24e b7 c0 45           holdon   STAA    KPTEMP          * Store current keypress in temporary location
d251 81 06                       CMPA    #RESET          * If reset pressed, then return to pattern generator
d253 27 1b                       BEQ     pat
d255 81 07                       CMPA    #FLAG           * If flag pressed, go to FLAG routine
d257 27 1a                       BEQ     flg
d259 81 03                       CMPA    #CLEAR          * If clear pressed, go to CLEAR routine
d25b 27 19                       BEQ     clr
d25d 81 02                       CMPA    #UP             * If up pressed, go to UP subroutine
d25f 27 18                       BEQ     up1
d261 81 05                       CMPA    #DOWN           * If down pressed, go to DOWN subroutine
d263 27 17                       BEQ     down1
d265 81 04                       CMPA    #LEFT           * If left pressed, go to LEFT subroutine
d267 27 16                       BEQ     left1
d269 81 01                       CMPA    #RIGHT          * If right pressed, go to RIGHT subroutine
d26b 27 15                       BEQ     right1
d26d 7e d1 e4                    JMP     nav             * Otherwise, return to start routine

d270 7e d1 00           pat      JMP     pattern
d273 7e d2 88           flg      JMP     flagged
d276 7e d2 c3           clr      JMP     cleared
```

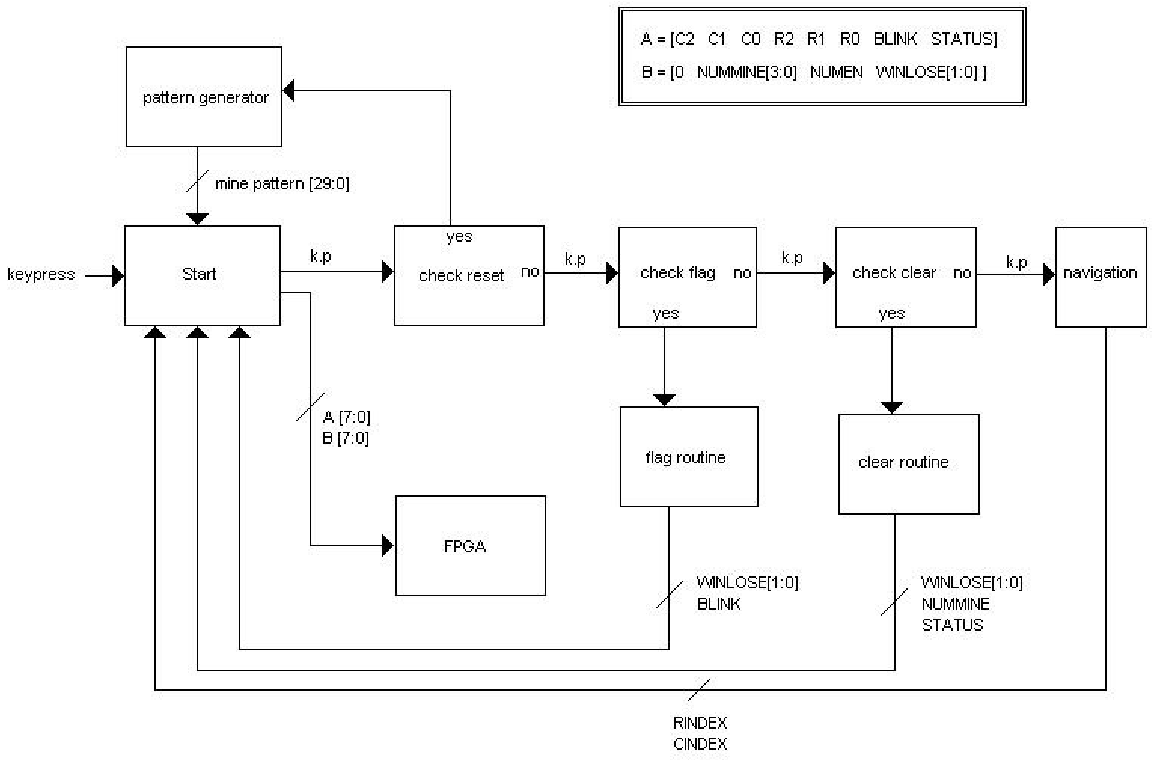
```
d279 7e d4 12           up1      JMP     navup
d27c 7e d4 20           down1    JMP     navdown
d27f 7e d4 2e           left1    JMP     navleft
d282 7e d4 3c           right1   JMP     navrt

d285 7e d1 e4                    JMP     nav             * Once done with routine, return to start


                        ****************************
                        *                                *
                        *      Flag Routine      *
                        *                                *
                        ****************************

                        * Checks to see if mine in current location. If no, user loses. If yes, stores
                        * a 1 in the blink signal, increments the flag counter, and returns to start

d288 b6 c0 33           flagged  LDAA    RINDEX
d28b 48                          LSLA
d28c 48                          LSLA
d28d 48                          LSLA
d28e 48                          LSLA                    * Shift RINDEX 2^4 = 1 hex number to left
d28f bb c0 34                    ADDA    CINDEX
d292 8b 01                       ADDA    #$01
d294 b7 c0 37                    STAA    REF1
d297 f6 c0 37                    LDAB    REF1
d29a ce c0 00                    LDX     #$C000          * Set X to zero
d29d 3a                          ABX
d29e a6 ff                       LDAA    $FF,X           * Load B'th element of X (B'th element of mine pattern)
d2a0 81 01                       CMPA    #$01            *
d2a2 27 03                       BEQ     flag1           * If it was not a mine, they lose
d2a4 7e d4 05                    JMP     lose
d2a7 86 01              flag1    LDAA    #$01            * Otherwise, load BLINK with blink signal
d2a9 b7 c0 35                    STAA    BLINK
d2ac 86 00                       LDAA    #$00
d2ae b7 c0 36                    STAA    STATUS
d2b1 b6 c0 3d                    LDAA    FLGCNT          * Load current flag count
d2b4 4c                          INCA                    * Increment flag count by one
d2b5 b1 c0 3b                    CMPA    TOTMINE         * Compare it to total number of mines
d2b8 27 06                       BEQ     winner          * If user has flagged every mine, they win
d2ba b7 c0 3d                    STAA    FLGCNT          * If not, then store new flag count and return to start
d2bd 7e d1 e4                    JMP     nav
d2c0 7e d4 0a           winner   JMP     win


                        *************************
                        *                             *
                        *     Clear Routine     *
                        *                             *
                        *************************

                        * Checks to see if mine in current location. If yes, user loses. If no, checks adjacent
                        * spaces for mines, records number of adjacent mines, and returns to start

                        ******************************
                        *          Mine Check         *
                        ******************************

d2c3 b6 c0 33           cleared  LDAA    RINDEX
d2c6 48                          LSLA
d2c7 48                          LSLA
d2c8 48                          LSLA
d2c9 48                          LSLA                    * Shift RINDEX 2^4 = 1 hex number to left
d2ca bb c0 34                    ADDA    CINDEX
d2cd 8b 01                       ADDA    #$01
d2cf b7 c0 37                    STAA    REF1
d2d2 f6 c0 37                    LDAB    REF1
d2d5 ce c0 00                    LDX     #$C000          * Set X to zero
d2d8 3a                          ABX
d2d9 a6 ff                       LDAA    $FF,X           * Load B'th element of X (B'th element of mine pattern)
d2db 81 01                       CMPA    #$01            * If it was a mine, go to lose (you lose)
d2dd 26 03                       BNE     clear1
d2df 7e d4 05                    JMP     lose
d2e2 c6 00              clear1   LDAB    #$00            *
d2e4 f7 c0 36                    STAB    STATUS          * Store 0 into STATUS to turn off LED
d2e7 8b 02                       ADDA    #CLRMASK        *
```

```
d2e9 a7 ff                       STAA    $FF,X          * Store 1 into "cleared" to indicate spot has been cleared


                        ************************
                        *      Mine Count      *
                        ************************

                        * Check mine pattern to see if there's a mine. Earlier routines choose which
                        * mine count routines to jump to, depending on current grid position


d2eb 18 ce 00 00         LDY     #$00            * Initialize mine counter


d2ef b6 c0 33           nextl    LDAA    RINDEX          * look in cell left of current position
d2f2 b7 c0 43                    STAA    RITEMP
d2f5 b6 c0 34                    LDAA    CINDEX
d2f8 8b ff                       ADDA    #$FF
d2fa b7 c0 44                    STAA    CITEMP
d2fd bd d3 ad                    JSR     adj_chk
d300 b6 c0 33           nextr    LDAA    RINDEX          * look in cell right of current position
d303 b7 c0 43                    STAA    RITEMP
d306 b6 c0 34                    LDAA    CINDEX
d309 8b 01                       ADDA    #$01
d30b b7 c0 44                    STAA    CITEMP
d30e bd d3 ad                    JSR     adj_chk
d311 b6 c0 33           nextul   LDAA    RINDEX          * look in cell above, left of current position
d314 8b ff                       ADDA    #$FF
d316 b7 c0 43                    STAA    RITEMP
d319 b6 c0 34                    LDAA    CINDEX
d31c 8b ff                       ADDA    #$FF
d31e b7 c0 44                    STAA    CITEMP
d321 bd d3 ad                    JSR     adj_chk
d324 b6 c0 33           nextu    LDAA    RINDEX          * look in cell above current position
d327 8b ff                       ADDA    #$FF
d329 b7 c0 43                    STAA    RITEMP
d32c b6 c0 34                    LDAA    CINDEX
d32f b7 c0 44                    STAA    CITEMP
d332 bd d3 ad                    JSR     adj_chk
d335 b6 c0 33           nextur   LDAA    RINDEX          * look in cell to right of current position
d338 8b ff                       ADDA    #$FF
d33a b7 c0 43                    STAA    RITEMP
d33d b6 c0 34                    LDAA    CINDEX
d340 8b 01                       ADDA    #$01
d342 b7 c0 44                    STAA    CITEMP
d345 bd d3 ad                    JSR     adj_chk
d348 b6 c0 33           nextbl   LDAA    RINDEX          * look in cell below, left of current position
d34b 8b 01                       ADDA    #$01
d34d b7 c0 43                    STAA    RITEMP
d350 b6 c0 34                    LDAA    CINDEX
d353 8b ff                       ADDA    #$FF
d355 b7 c0 44                    STAA    CITEMP
d358 bd d3 ad                    JSR     adj_chk
d35b b6 c0 33           nextb    LDAA    RINDEX          * look in cell below current position
d35e 8b 01                       ADDA    #$01
d360 b7 c0 43                    STAA    RITEMP
d363 b6 c0 34                    LDAA    CINDEX
d366 b7 c0 44                    STAA    CITEMP
d369 bd d3 ad                    JSR     adj_chk
d36c b6 c0 33           nextbr   LDAA    RINDEX          * look in cell below, right of current position
d36f 8b 01                       ADDA    #$01
d371 b7 c0 43                    STAA    RITEMP
d374 b6 c0 34                    LDAA    CINDEX
d377 8b 01                       ADDA    #$01
d379 b7 c0 44                    STAA    CITEMP
d37c bd d3 ad                    JSR     adj_chk

d37f 18 ff c0 39                 STY     NUMMINE         * Store total mine count in NUMMIN1
d383 b6 c0 3a                    LDAA    NUMMIN1
d386 48                          LSLA
d387 48                          LSLA
d388 b7 c0 3a                    STAA    NUMMIN1
d38b b6 c0 33                    LDAA    RINDEX
d38e 48                          LSLA
d38f 48                          LSLA
d390 48                          LSLA
```

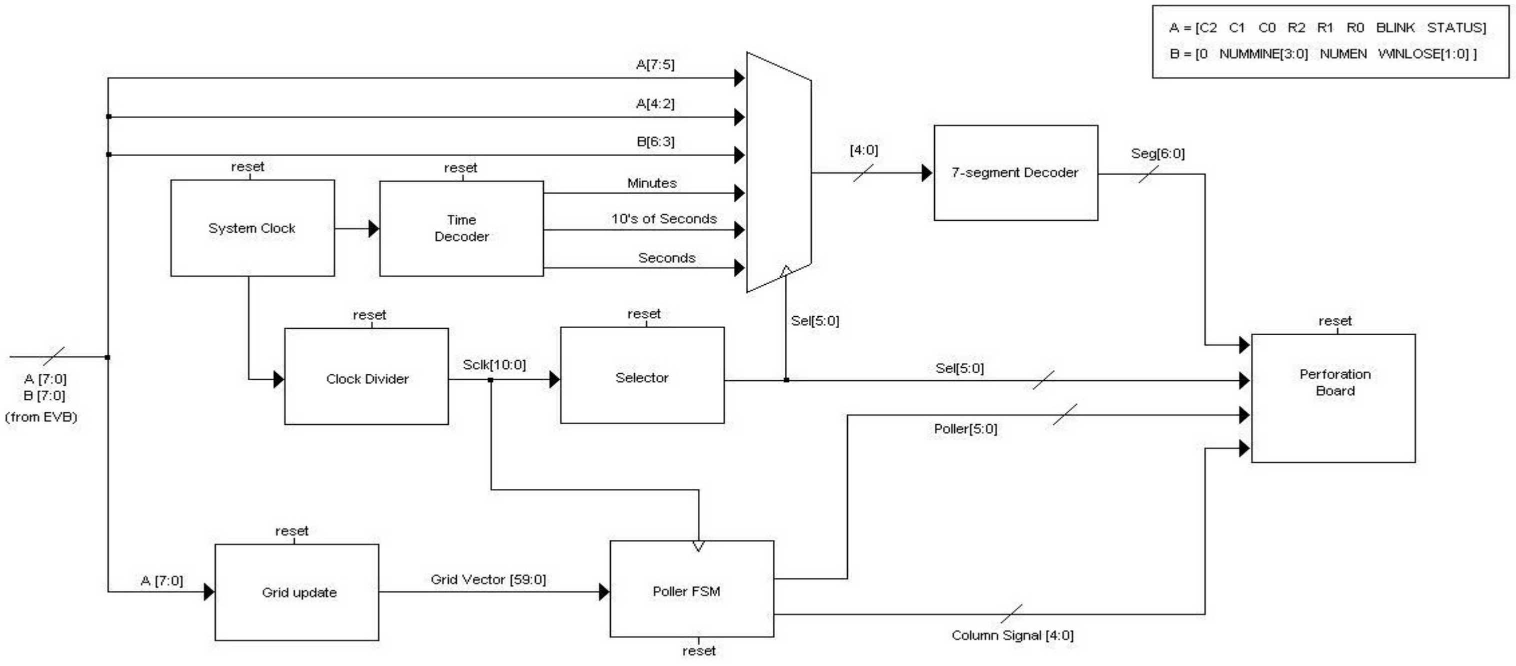
```
d391 48                           LSLA                    * Shift RINDEX 2^4 = 1 hex number to left
d392 bb c0 34                     ADDA    CINDEX
d395 8b 01                        ADDA    #$01
d397 b7 c0 37                     STAA    REF1
d39a f6 c0 37                     LDAB    REF1
d39d ce c0 00                     LDX     #$C000          * Set X to zero
d3a0 3a                           ABX
d3a1 a6 ff                        LDAA    $FF,X           * Load B'th element of X (B'th element of mine pattern)
d3a3 84 03                        ANDA    #CLRMSK2
d3a5 bb c0 3a                     ADDA    NUMMIN1         * store number of mines in memory for this
d3a8 a7 ff                        STAA    $FF,X           * cell location
d3aa 7e d1 e4                     JMP     nav


                        ************************************
                        *  Adjacent Mine Check Subroutine  *
                        ************************************

d3ad b6 c0 43           adj_chk  LDAA    RITEMP
d3b0 48                          LSLA
d3b1 48                          LSLA
d3b2 48                          LSLA
d3b3 48                          LSLA                    * Shift RINDEX 2^4 = 1 hex number to left
d3b4 bb c0 44                    ADDA    CITEMP
d3b7 8b 01                       ADDA    #$01
d3b9 b7 c0 38                    STAA    REF2
d3bc f6 c0 38                    LDAB    REF2
d3bf ce c0 00                    LDX     #$C000          * Set X to zero
d3c2 3a                          ABX                     *
d3c3 a6 ff                       LDAA    $FF,X           * Check mine pattern to see if there's a mine
d3c5 84 01                       ANDA    #$01            * Examine adjacent position for mine
d3c7 81 01                       CMPA    #$01
d3c9 26 02                       BNE     chk_out
d3cb 18 08                       INY                     * Increment mine counter if mine found nearby
d3cd 39                 chk_out  RTS


                        *******************************
                        *  Current Position Cleared?  *
                        *******************************

d3ce b6 c0 33           posclr   LDAA    RINDEX
d3d1 48                          LSLA
d3d2 48                          LSLA
d3d3 48                          LSLA
d3d4 48                          LSLA                    * Shift RINDEX 2^4 = 1 hex number to left
d3d5 bb c0 34                    ADDA    CINDEX
d3d8 8b 01                       ADDA    #$01
d3da b7 c0 37                    STAA    REF1
d3dd f6 c0 37                    LDAB    REF1
d3e0 ce c0 00                    LDX     #$C000          * Set X to zero
d3e3 3a                          ABX
d3e4 a6 ff                       LDAA    $FF,X           * Load B'th element of X (B'th element of mine pattern)
d3e6 84 02                       ANDA    #CLRMASK
d3e8 81 02                       CMPA    #CLRMASK        * Check to see if current position has been cleared
d3ea 26 13                       BNE     skippos         * If not, don't display number of adjacent mines
d3ec 86 01                       LDAA    #$01
d3ee b7 c0 3c                    STAA    NUMEN           * If clear, activate enable signal to display # adj. mines
d3f1 a6 ff                       LDAA    $FF,X
d3f3 44                          LSRA
d3f4 44                          LSRA
d3f5 b7 c0 3a                    STAA    NUMMIN1
d3f8 86 00                       LDAA    #$00
d3fa b7 c0 36                    STAA    STATUS          * Load a zero into status to turn off LED
d3fd 20 05                       BRA     skp2
d3ff 86 01              skippos  LDAA    #$01
d401 b7 c0 36                    STAA    STATUS          * If position has not been cleared, keep LED on
d404 39                 skp2     RTS


                        **************************
                        *                      *
                        *  Win/Lose Subroutine   *
                        *                              *
                        **************************

```

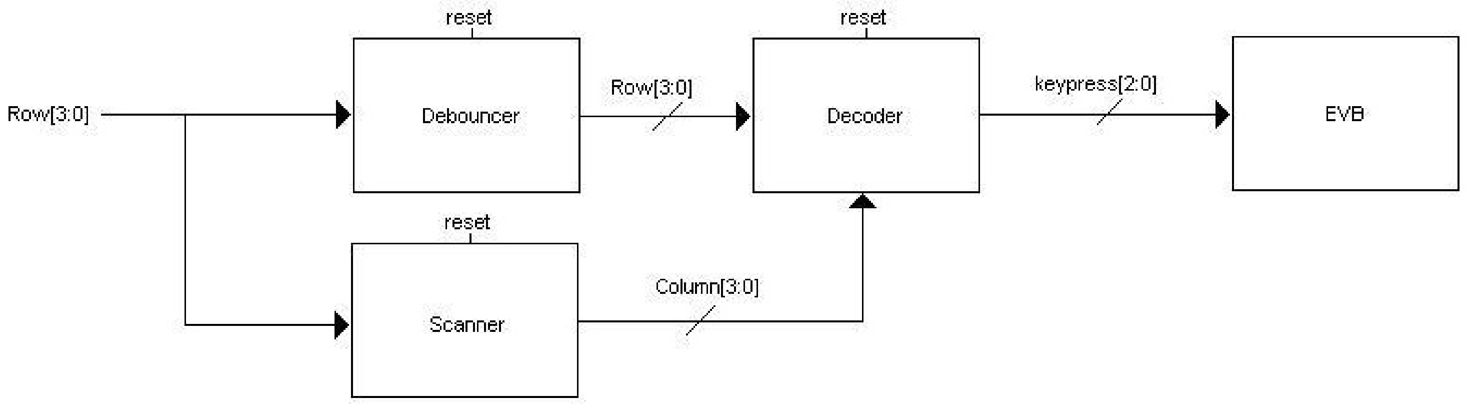
```
d405 86 02              lose     LDAA    #LOSER
d407 b7 c0 32                    STAA    WINLOSE         * Set win/lose to lose
d40a 86 01              win      LDAA    #WINNER
d40c b7 c0 32                    STAA    WINLOSE         * Set win/lose to win
d40f 7e d1 e4                    JMP     nav

                        *******************************
                        *                             *
                        *   Navigation Subroutines    *
                        *                             *
                        *******************************

                        * Checks current grid position and adjusts row/column indices
                        * according to navigation direction specified by user.

                        **********
                        *   up   *
                        **********

d412 b6 c0 33           navup    LDAA    RINDEX
d415 81 01                       CMPA    #$01            *If already at top row, do not decrease row index
d417 27 04                       BEQ     up
d419 4a                          DECA
d41a b7 c0 33                    STAA    RINDEX
d41d 7e d1 e4           up               JMP     nav             * Return to start routine when done


                        **********
                        *  down  *
                        **********

d420 b6 c0 33           navdown  LDAA    RINDEX
d423 81 05                       CMPA    #$05            * If already at bottom row, do not increase row index
d425 27 04                       BEQ     down
d427 4c                          INCA
d428 b7 c0 33                    STAA    RINDEX
d42b 7e d1 e4           down     JMP     nav             * Return to start routine when done


                        **********
                        *  left  *
                        **********

d42e b6 c0 34           navleft  LDAA    CINDEX
d431 81 01                       CMPA    #$01            * If already at leftmost column, do not decrease
d433 27 04                       BEQ     left            * column index
d435 4a                          DECA
d436 b7 c0 34                    STAA    CINDEX
d439 7e d1 e4           left     JMP     nav             * Return to start routine when done


                        ***********
                        *  right  *
                        ***********

d43c b6 c0 34           navrt    LDAA    CINDEX
d43f 81 06                       CMPA    #$06            * If already at rightmost column, do not increase
d441 27 04                       BEQ     right           * column index
d443 4c                          INCA
d444 b7 c0 34                    STAA    CINDEX
d447 7e d1 e4           right    JMP     nav             * Return to start routine when done t increase
```

Appendix C: EVB Block Diagram

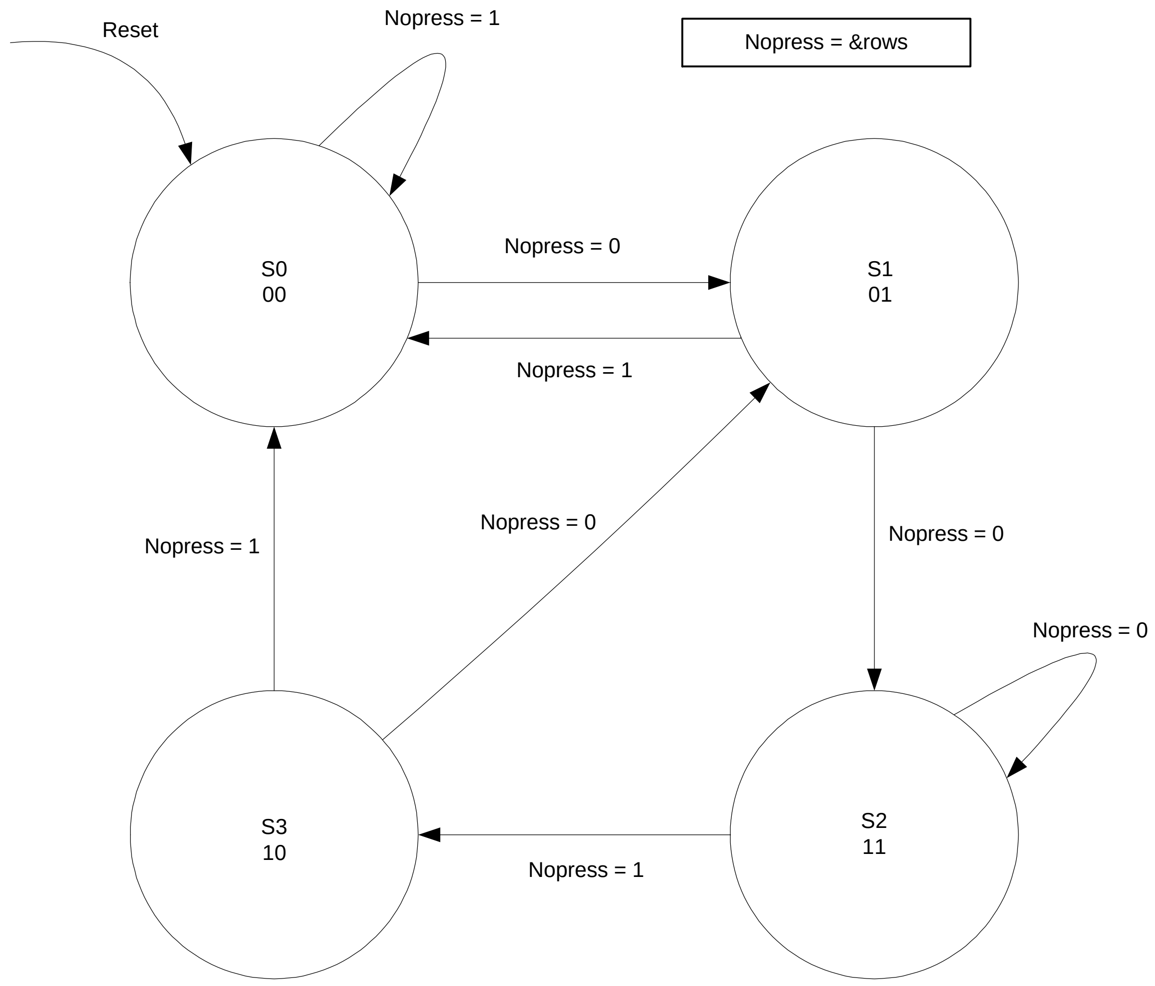


Appendix D: FPGA Block Diagram and FSM Diagrams





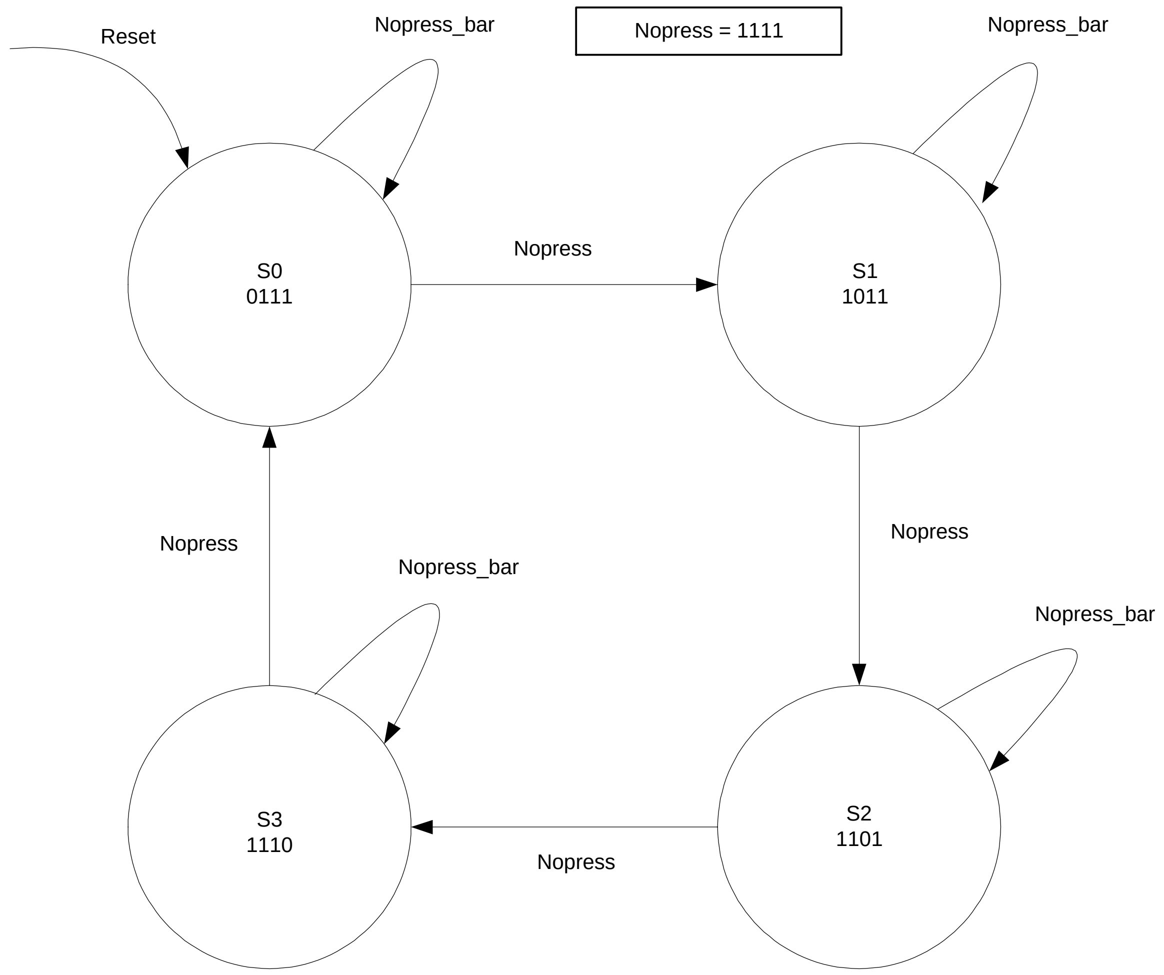
Appendix D (Cont’d): Keypad Debouncer FSM



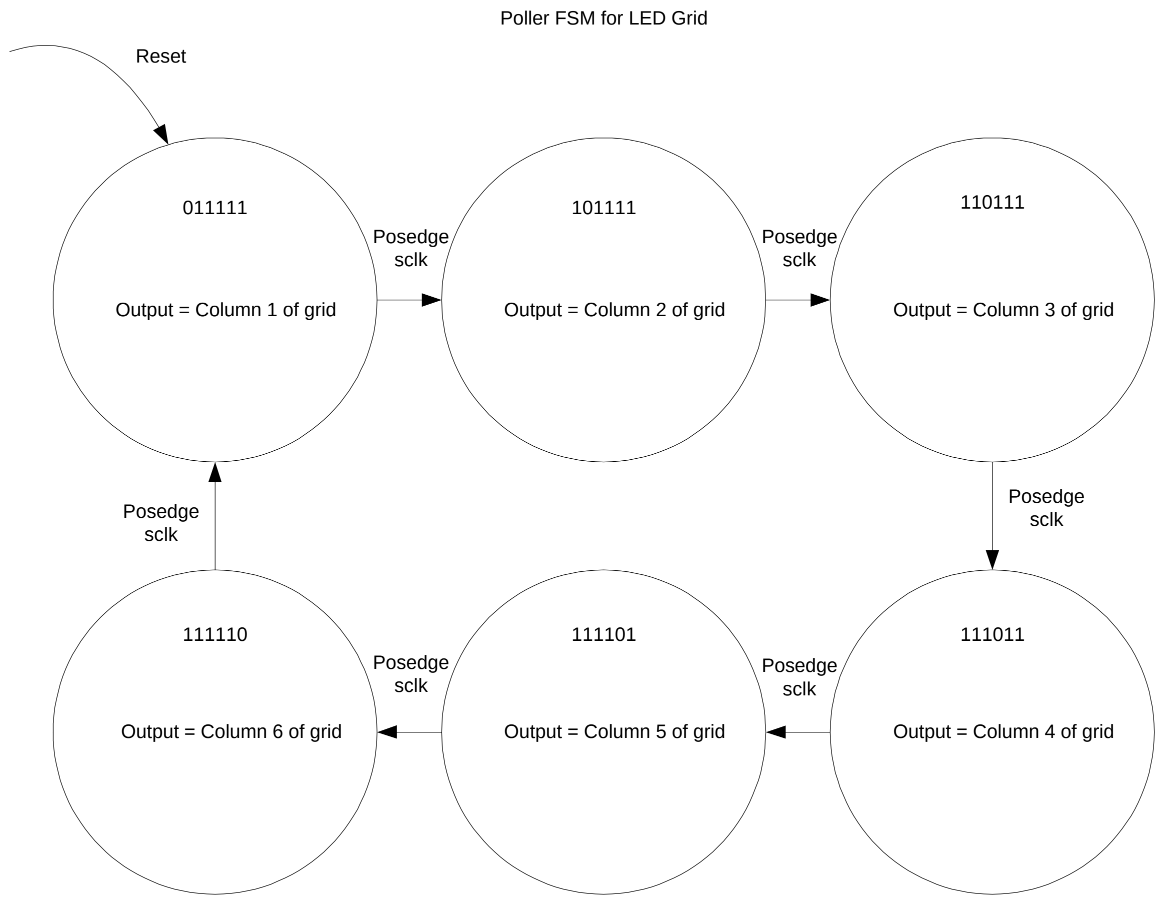
Output Logic

En = state[1]&state[0]

Appendix D (Cont’d): Keypad Scanner FSM



Appendix D (Cont'd): FSM for LED Grid Poller


Poller FSM for LED Grid
Reset
011111
Output = Column 1 of grid
101111
Output = Column 2 of grid
110111
Output = Column 3 of grid
111011
Output = Column 4 of grid
111101
Output = Column 5 of grid
111110
Output = Column 6 of grid
Posedge sclk

Appendix E: Verilog Modules

```
module Final_Project(rows,clk,reset,keypress,columns,A,B,seg,gridcol,poller,sel,win,lose);

   input clk;
   input reset;
   input [3:0] rows;
   input [7:0] A;
   input [6:0] B;
   output [2:0] keypress;
           output [3:0] columns;
   output [6:0] seg;
   output [5:0] poller,sel;
           output [4:0] gridcol;
           output win, lose;

FPGA2EVB FPGA2EVB2(rows,clk,reset,keypress,columns);

minesweeper minesweeper2(clk,reset,A,B,seg,gridcol,poller,sel,win,lose);

endmodule



module FPGA2EVB(rows,clk,reset,keypress,columns);
   input [3:0] rows;
   input clk;
   input reset;
   output [2:0] keypress;
           output [3:0] columns;

           wire sclk, bclk;
           wire [11:0] y;
           wire en;
           wire [3:0] columns;

                   slow_clock slow_clock3(clk,reset,sclk,bclk,y);
                   debouncer debouncer3(reset,rows,sclk,en);
                   keyscanner keyscanner3(sclk,reset,rows,columns);
                   keydecoder keydecoder3(clk,reset,rows,columns,keypress,en);

endmodule
```

```
module debouncer(reset,rows,sclk,en);
   input reset;
   input [3:0] rows;
   input sclk;
   output en;

            wire nopress;

            reg [1:0] state;
            reg [1:0] nextstate;

            assign nopress = (rows[3]&rows[2]&rows[1]&rows[0]);

            parameter S0 = 2'b00;
            parameter S1 = 2'b01;
            parameter S2 = 2'b11;
            parameter S3 = 2'b10;

            // State Register

            always @(posedge sclk or posedge reset)
                     if (reset)           state <= S0;
                     else                          state <= nextstate;

            //Next state logic

            always @(state or nopress)
                     case (state)

                              S0:       if (nopress)        nextstate <= S0;
                                        else                nextstate <= S1;
                              S1:       if (nopress)        nextstate <= S0;
                                        else                nextstate <= S2;
                              S2:       if (nopress)        nextstate <= S3;
                                        else                nextstate <= S2;
                              S3:       if (nopress)        nextstate <= S0;
                                        else                nextstate <= S1;

                              default: nextstate <= S0;

                     endcase

                     // Output Logic

                     assign en = (state[1]&state[0]);
endmodule
```

```
module keydecoder(clk,reset,rows,columns,keypress,en);
            input clk;
            input reset;
   input [3:0] rows;
   input [3:0] columns;
            input en;
   output [2:0] keypress;

            wire [7:0] A;
            reg [2:0] keypress;

parameter UP                = 8'b1011_0111;
parameter LEFT              = 8'b0111_1011;
parameter DOWN              = 8'b1011_1011;
parameter RIGHT             = 8'b1101_1011;
parameter CLEAR             = 8'b0111_1101;
parameter FLAG              = 8'b1011_1101;
parameter RESET             = 8'b1101_1101;

            assign A = {columns[3:0], rows[3:0]};



            always @(posedge clk or posedge reset)
                     if (reset) keypress <= 3'b0;
                     else
                               if (en)
                                         case(A)
                                                   UP:       keypress <= 3'b010;
                                                   LEFT:     keypress <= 3'b100;
                                                   DOWN: keypress <= 3'b101;
                                                   RIGHT: keypress <= 3'b001;
                                                   CLEAR:keypress <= 3'b011;
                                                   FLAG:     keypress <= 3'b111;
                                                   RESET: keypress <= 3'b110;
                                                   default: keypress <= 3'b0;
                                         endcase
                               else keypress <= 3'b0;
endmodule

module keyscanner(clk,reset,rows,columns);
   input clk;
   input reset;
   input [3:0] rows;
   output [3:0] columns;

                     reg [3:0] state;
                     reg [3:0] nextstate;

                     parameter nopress = 4'b1111;
                     parameter S0 = 4'b0111;
                     parameter S1 = 4'b1011;
                     parameter S2 = 4'b1101;
                     parameter S3 = 4'b1110;

                     // State Register
```

```
		always @(posedge clk or posedge reset)
			if (reset)		state <= S0;
			else				state <= nextstate;

		// Next State Logic

		always @(state or rows)
			case (state)
				S0:	if (rows == nopress)	nextstate <= S1;
					else			nextstate <= state;
				S1:	if (rows == nopress)	nextstate <= S2;
					else			nextstate <= state;
				S2:	if (rows == nopress)	nextstate <= S3;
					else			nextstate <= state;
				S3:	if (rows == nopress)	nextstate <= S0;
					else			nextstate <= state;
				default:			nextstate <= S0;
			endcase

		// Output Logic

		assign columns = state;
endmodule



module slow_clock(clk,reset,sclk,bclk,y);
   input clk;
   input reset;
   output sclk;
	output bclk;
	output [10:0] y;

		reg [19:0] q;

		always @(posedge clk or posedge reset)
			if (reset) q <= 20'b0;
			else
				if (q == 20'b1111_1111_1111_1111_1111)  q <= 20'b0;
				else					 q <= q + 1;

		assign sclk = ~q[6]; //q[6]
		assign bclk = q[16];	//q[16]
		assign y = q[10:0];

endmodule
```

```
module minesweeper(clk,reset,A,B,seg,gridcol,poller,sel,win,lose);
   input clk;
   input reset;
   input [7:0] A;
   input [6:0] B;
   output [6:0] seg;
   output [5:0] poller,sel;
                output [4:0] gridcol;
                output win, lose;

                wire [1:0] blinkstatus;
                wire sclk, bclk;
                wire [59:0] L;
                wire [10:0] y;
                wire [4:0] row_in, col_in, mine_num, sec, ten_sec, minutes;
                wire [4:0] loc;
                wire [4:0] q;
                wire [5:0] poller,sel;
                wire en;

                        slow_clock slow_clock2(clk,reset,sclk,bclk,y);                    //input clk,reset
                                                                                          //output sclk,bclk,y

                        loc_decoder loc_decoder2(clk,reset,A,loc,blinkstatus);            //input A
                                                                                          //output loc,blinkstatus

                        index_decoder index_decoder2(clk,reset,A[7:2],B[6:2],row_in,col_in,mine_num);      //input A,B
                                                                                                  //output row,col,mine#

                        blinkoff_fsm blinkoff_fsm2(clk, reset, loc, blinkstatus, L);      //input loc,blinkstatus,reset
                                                                                          //,sclk,bclk,output L

                        gridpoller gridpoller2(sclk,bclk,reset,L,gridcol,poller);         //input sclk,reset,L
                                                                                          //output gridcol,poller

                        sec_decoder sec_decoder2(clk,reset,sec,en);                       //input clk,reset
                        ten_sec_decoder ten_sec_decoder2(clk,reset,en,ten_sec);           //output seconds,ten_sec
                        minute_decoder minute_decoder2(clk,reset,en,ten_sec,minutes);     //,minutes

                        selector selector2(clk,reset,y,sel);

                        mux6 mux62(row_in,col_in,mine_num,sec,ten_sec,minutes,sel,q);     //input row,col
                                                                                          //mine#,sec,
                                                                                          //ten_sec,minutes
                                                                                          //sel,output q

                        sevenseg sevenseg2(q,seg);                                        //input q, output seg

                        winlose winlose2(B[1:0],win,lose);                                //input B, output w/l

endmodule
```

```
module blinkoff_fsm(clk, reset, loc, blinkstatus, L);
//blinkoff_fsm(blinkstatus,loc,reset,clk,bclk,L);

        input clk;
        input reset;
        input [4:0] loc;
        input [1:0] blinkstatus;
        output [59:0] L;

        reg [59:0] L;

        always @(posedge clk or posedge reset)
                if (reset)          L <= 60'b0;
                else if (blinkstatus == 2'b00)
                        begin
                                L[2*(loc-1)] <= 1;  // turn off LED
                                L[2*(loc-1)+1] <= 0;
                  end
                else if (blinkstatus == 2'b01)
                        begin
                                L[2*(loc-1)] <= 0;
                                L[2*(loc-1)+1] <= 0;
                        end
                else if (blinkstatus == 2'b10)
                        begin
                                L[2*(loc-1)+1] <= 1; //blink
                                L[2*(loc-1)] <= 0;
                        end
endmodule

module gridpoller(sclk,bclk,reset,L,gridcol,poller);
   input sclk;
        input bclk;
   input reset;
        input [59:0] L;
   output [4:0] gridcol;
        output [5:0] poller;

        reg [5:0] state;
        reg [5:0] nextstate;
        reg [4:0] gridcol;

        parameter S0 = 6'b011111;
        parameter S1 = 6'b101111;
        parameter S2 = 6'b110111;
        parameter S3 = 6'b111011;
        parameter S4 = 6'b111101;
        parameter S5 = 6'b111110;

        assign poller[5:0] = state[5:0];
        assign poller[4] = state[4];
        assign poller[3] = state[3];
        assign poller[2] = state[2];
        assign poller[1] = state[1];
        assign poller[0] = state[0];

        // State Register

        always @(posedge sclk or posedge reset)
                if (reset)  state <= S0;
                else                        state <= nextstate;

        // Next State Logic
```

```
            always @(state)
                    case (state)
                            S0:         begin
                                                gridcol[4:0] =
{(L[49]&bclk|L[48]),(L[37]&bclk|L[36]),(L[25]&bclk|L[24]),(L[13]&bclk|L[12]),(L[1]&bclk|L[0])};
                                                        nextstate <= S1;
                                                end

                            S1:         begin   gridcol[4:0] =
{(L[51]&bclk|L[50]),(L[39]&bclk|L[38]),(L[27]&bclk|L[26]),(L[15]&bclk|L[14]),(L[3]&bclk|L[2])};
                                                        nextstate <= S2;
                                                end

                            S2:         begin   gridcol[4:0] =
{(L[53]&bclk|L[52]),(L[41]&bclk|L[40]),(L[29]&bclk|L[28]),(L[17]&bclk|L[16]),(L[5]&bclk|L[4])};
                                                        nextstate <= S3;
                                                end

                            S3:         begin   gridcol[4:0] =
{(L[55]&bclk|L[54]),(L[43]&bclk|L[42]),(L[31]&bclk|L[30]),(L[19]&bclk|L[18]),(L[7]&bclk|L[6])};
                                                        nextstate <= S4;
                                                end

                            S4:         begin   gridcol[4:0] =
{(L[57]&bclk|L[56]),(L[45]&bclk|L[44]),(L[33]&bclk|L[32]),(L[21]&bclk|L[20]),(L[9]&bclk|L[8])};
                                                        nextstate <= S5;
                                                end

                            S5:         begin   gridcol[4:0] =
{(L[59]&bclk|L[58]),(L[47]&bclk|L[46]),(L[35]&bclk|L[34]),(L[23]&bclk|L[22]),(L[11]&bclk|L[10])};
                                                        nextstate <= S0;
                                                end

                            default:
                                        begin   gridcol[4:0] =
{(L[49]&bclk|L[48]),(L[37]&bclk|L[36]),(L[25]&bclk|L[24]),(L[13]&bclk|L[12]),(L[1]&bclk|L[0])};
                                                        nextstate <= S0;
                                                end
             endcase

endmodule
```

```
module index_decoder(clk,reset,A,B,row_in,col_in,mine_num);
            input clk;
            input reset;
            input [5:0] A;
   input [4:0] B;
   output [4:0] row_in, col_in, mine_num;

            reg [4:0] row_in,col_in,mine_num;

           always @(posedge clk or posedge reset)

                      if (reset)
                                 begin
                                            col_in <= 5'b0;
                                            row_in <= 5'b0;
                                            mine_num <= 5'b0;
                                 end
                      else
                                 begin
                                            col_in <= {2'b00,A[5:3]};
                                            row_in <= {2'b00,A[2:0]};
                                            mine_num <= {~B[0],B[4:1]};
                                 end
endmodule
```

```
module loc_decoder(clk,reset,A,loc,blinkstatus);
            input clk;
            input reset;
            input [7:0] A;
   output [4:0] loc;
            output [1:0] blinkstatus;

            wire [2:0] row, col;
            reg [4:0] loc;
            reg [1:0] blinkstatus;

            assign col = A[7:5];
            assign row = A[4:2];

            always @(posedge clk or posedge reset)

            if (reset)
                      begin
                                blinkstatus <= 2'b01;
                                loc <= 5'b00001;
                      end

            else
                      begin
                                blinkstatus <= A[1:0];
                                loc <= ((row-1) * 6) + col;
                      end

endmodule

module minute_decoder(clk,reset,en,ten_sec,minutes);
           input clk;
           input reset;
           input en;
           input [4:0] ten_sec;
           output [4:0] minutes;
           reg [4:0] minutes;

           always @(posedge clk or posedge reset)

                     if (reset) minutes <= 5'b0;
                     else if (en && ten_sec == 5'b00101)
                               case (minutes)
                                         5'b00000 :  minutes <= 5'b00001;
                                         5'b00001          :          minutes <= 5'b00010;
                                         5'b00010 :          minutes <= 5'b00011;
                                         5'b00011 :          minutes <= 5'b00100;
                                         5'b00100 :          minutes <= 5'b00101;
                                         5'b00101 :          minutes <= 5'b00110;
                                         5'b00110 :          minutes <= 5'b00111;
                                         5'b00111 :          minutes <= 5'b01000;
                                         5'b01000          :          minutes <= 5'b01001;
                                         5'b01001          :          minutes <= 5'b00000;
                                         default   :          minutes <= 5'b00000;
                               endcase
endmodule
```

```
module mux6(d0,d1,d2,d3,d4,d5,sel,q);
             input [4:0] d0, d1, d2, d3, d4, d5;
    input [5:0] sel;
    output [4:0] q;

parameter sel0 = 6'b011111;
parameter sel1 = 6'b101111;
parameter sel2 = 6'b110111;
parameter sel3 = 6'b111011;
parameter sel4 = 6'b111101;
parameter sel5 = 6'b111110;

             assign    q = (sel == sel0) ? d0 : {5'bzzzzz},
                                          q = (sel == sel1) ? d1 : {5'bzzzzz},
                                          q = (sel == sel2) ? d2 : {5'bzzzzz},
                                          q = (sel == sel3) ? d3 : {5'bzzzzz},
                                          q = (sel == sel4) ? d4 : {5'bzzzzz},
                                          q = (sel == sel5) ? d5 : {5'bzzzzz};

endmodule


module sec_decoder(clk,reset,sec,en);
    input clk;
    input reset;
             output en;
             output [4:0] sec;

                        reg [23:0] q;
                        reg [4:0] sec;
                        reg en;

                        always @(posedge clk or posedge reset)
                                    if(reset) begin
                                                                                sec <= 5'b0;
                                                                                q <= 24'b0;
                                                                                en <= 0;
                                                                    end
                                    else if (q[23:20] == 4'b1010)
                                                                    begin
                                                                                q <= 24'b0;
                                                                                en <= 1;
                                                                    end
                                                        else        begin
                                                                                q <= q + 1;
                                                                                en <= 0;
                                                                                sec <= {1'b0,q[23:20]};
                                                                    end
endmodule
```

```
module selector(clk,reset,y,sel);
   input clk;
              input reset;
   input [10:0] y;
   output [5:0] sel;

              reg [5:0] sel;

              always @(posedge clk or posedge reset)
                          if(reset) sel <= 6'b111111;
                          else
                                      if (y >= 11'b0 && y <= 11'b001_0101_0101)
             sel <= 6'b011111;
                                      else if (y >= 11'b001_0101_0110 && y <= 11'b010_1010_1010)   sel <= 6'b101111;
                                      else if (y >= 11'b010_1010_1011 && y <= 11'b011_1111_1111)   sel <= 6'b110111;
                                      else if (y >= 11'b100_0000_0000 && y <= 11'b101_0101_0100)   sel <= 6'b111011;
                                      else if (y >= 11'b101_0101_0101 && y <= 11'b110_1010_1001)   sel <= 6'b111101;
                                      else if (y >= 11'b110_1010_1010)
                                                  sel <= 6'b111110;
endmodule



module sevenseg(s,seg);
   input [4:0] s;
   output [6:0] seg;

                                      assign seg[0] = (s[4]|((s[3]&s[2]&~s[1]&~s[0]) | (~s[3]&~s[2]&~s[1]) | (~s[3]&s[2]&s[1]&s[0])));
                                      assign seg[1] = (s[4]|((~s[3]&~s[2]&s[1]) | (~s[3]&s[1]&s[0]) | (s[3]&s[2]&~s[1]&s[0])));
                                      assign seg[2] = (s[4]|((~s[3]&s[2]&~s[1]) | (~s[3]&s[1]&s[0]) | (s[3]&~s[2]&~s[1]&s[0])));
                                      assign seg[3] = (s[4]|((s[2]&s[1]&s[0]) | (~s[2]&~s[1]&s[0]) | (s[3]&~s[2]&s[1]&~s[0]) | (~s[3]&s[2]&~s[1]&~s[0])));
                                      assign seg[4] = (s[4]|(((s[3]&s[2])&(s[1] | ~s[0])) | ((~s[3]&~s[2])&(s[1]^s[0]))));
                                      assign seg[5] = (s[4]|((s[0]&((~s[3]&~s[1]) | (s[3]&s[1]))) | (s[2]&~s[0])&(s[1] | s[3])));
                                      assign seg[6] = (s[4]|(((~s[3]&~s[1])&(s[2]^s[0])) | ((s[3]&s[0])&(s[2]^s[1]))));

endmodule



module slow_clock(clk,reset,sclk,bclk,y);
   input clk;
   input reset;
   output sclk;
             output bclk;
             output [10:0] y;

                         reg [19:0] q;

                         always @(posedge clk or posedge reset)
                                     if (reset) q <= 20'b0;
                                     else
                                                 if (q == 20'b1111_1111_1111_1111_1111)  q <= 20'b0;
                                                 else                                    q <= q + 1;

                         assign sclk = ~q[6];
                         assign bclk = q[16];
                         assign y = q[10:0];

endmodule
```

```
module ten_sec_decoder(clk,reset,en,ten_sec);

        input clk;
        input reset;
        input en;
        output [4:0] ten_sec;

        reg [4:0] ten_sec;

        always @(posedge clk or posedge reset)

                if (reset) ten_sec <= 5'b00000;
                else if (en)
                        case (ten_sec)
                                5'b00000 :  ten_sec <= 5'b00001;
                                5'b00001        :       ten_sec <= 5'b00010;
                                5'b00010 :      ten_sec <= 5'b00011;
                                5'b00011 :      ten_sec <= 5'b00100;
                                5'b00100 :      ten_sec <= 5'b00101;
                                5'b00101 :      ten_sec <= 5'b00000;
                                default  :ten_sec <= 5'b00000;
                        endcase
endmodule



module winlose(B,win,lose);
   input [1:0] B;
   output win;
   output lose;

                assign lose = B[1];
                assign win = B[0];

endmodule
```