The Super Happy Fun Game:
A Text-Based Adventure Game

Final Project Report
December 6, 2000
E155

Ari Moradi and Ryan Stuck

Abstract:

An interesting problem that comes up quite often in industry is the problem of interfacing with a user. This particular design issue, coupled with the fun of a text-based adventure game, has spawned this project, the Super Happy Fun Game. In this project, we have created a short, text-based adventure game, which allows users to input desired commands on a standard 4x4 keypad, and outputs game information onto a 4 line by 20 character liquid crystal display. The game is implemented as a finite state machine on our Xilinx FPGA, which in turn sends data to an HC11 Evaluation Board that interprets the state data given from the FPGA and displays appropriate data to the user.

Introduction

We have designed and implemented a text adventure game called “The Super Happy Fun Game.” The game uses a 68HC11 Evaluation Board (EVB), a Xilinx Spartan FPGA, a keypad, and a LCD display. All of the parts necessary for our project have been checked out of the Engineering Stockroom.

The FPGA holds the game data while the HC11 takes as input from the FPGA the current state of the game and then outputs text to the LCD display.

Detailed descriptions of how the FPGA and the HC11 work follow.

New Hardware

The creation of the Super Happy Fun Game required the use of a dot matrix style liquid crystal display (LCD). The LCD employed is a 4 line by 20 character display. Each character is made up of a 5x11 dot matrix. The actual number of dots employed in displaying a character is configurable, as noted below. The LCD chosen is one of the smart LCD variety, being that it has its own controller on board, namely a Hitachi 44780XX controller. Thus, one need only send pre-defined commands to the LCD to operate it. Below can be found notes that may aid future groups in the implementation of this type of LCD, including a wiring diagram, the fundamental instruction set, and some trouble shooting tips. Two appendices at the end of this document will include one, code implementing the LCD with an HC11 Evaluation Board (EVB), and two, timing diagrams. Now, find below a wiring diagram of the LCD and description of the pin out.

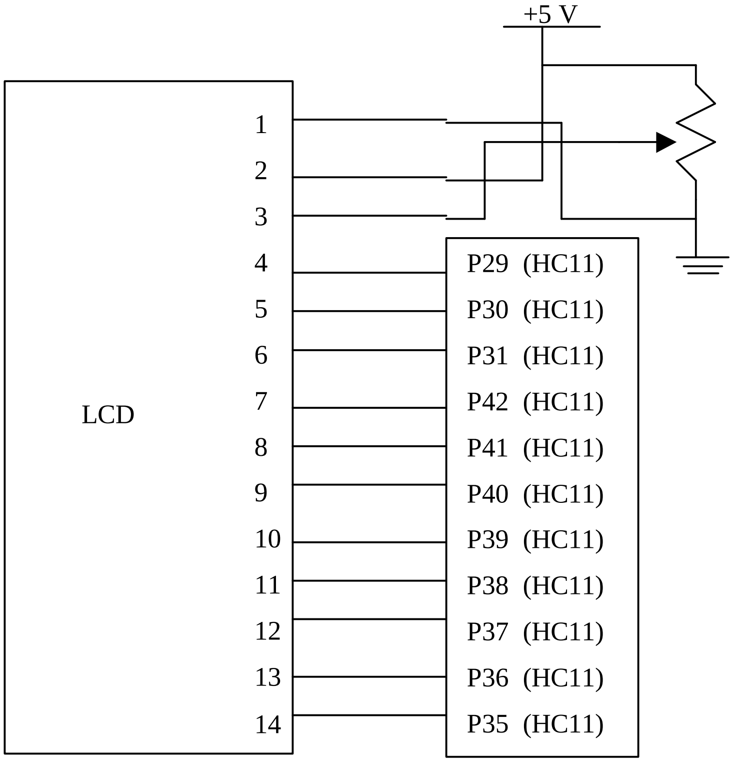


Figure 1: Wiring Diagram of LCD. Shows contrast adjustment circuit and pin out to HC11 EVB. A description of the pins can be found below.

Below is a table describing the pin out of the previous wiring diagram; it contains the pin number, connection, name and function of each pin on the LCD.

Pin Number	*Name*	*Function*	*Connection*
1	V_{ss}	Ground	Ground
2	V_{dd}	+5V	+5V power supply
3	V_{ee}	Contrast	Potentiometer
4	RS	Register Select	P29 HC11 port A, bit 5
5	R/W	Read/Write	P30 HC11 port A, bit 4
6	E	Enable	P31 HC11 port A, bit 3
7	D0	Data bit 0	P42 HC11 port B, bit 0
8	D1	Data bit 1	P41 HC11 port B, bit 1
9	D2	Data bit 2	P40 HC11 port B, bit 2
10	D3	Data bit 3	P39 HC11 port B, bit 3
11	D4	Data bit 4	P38 HC11 port B, bit 4
12	D5	Data bit 5	P37 HC11 port B, bit 5
13	D6	Data bit 6	P36 HC11 port B, bit 6
14	D7	Data bit 7	P35 HC11 port B, bit 7

Table 1: Pin out of LCD. Table shows pin number, name, function, and connection to circuit.

On the following page can be found a table describing the fundamental command set to control the LCD. More commands exist, however they are a bit more exotic, and not relevant to the functionality of this design. See references to find more resources on implementing these other instructions. Also, to write an ASCII character to the LCD, the write data command must be given. Attached is an ASCII character table, giving the character and which byte is used to denote it. This byte is what is sent along with the write data command. Also note that R/W was tied high, as no reading from the LCD was ever necessary.

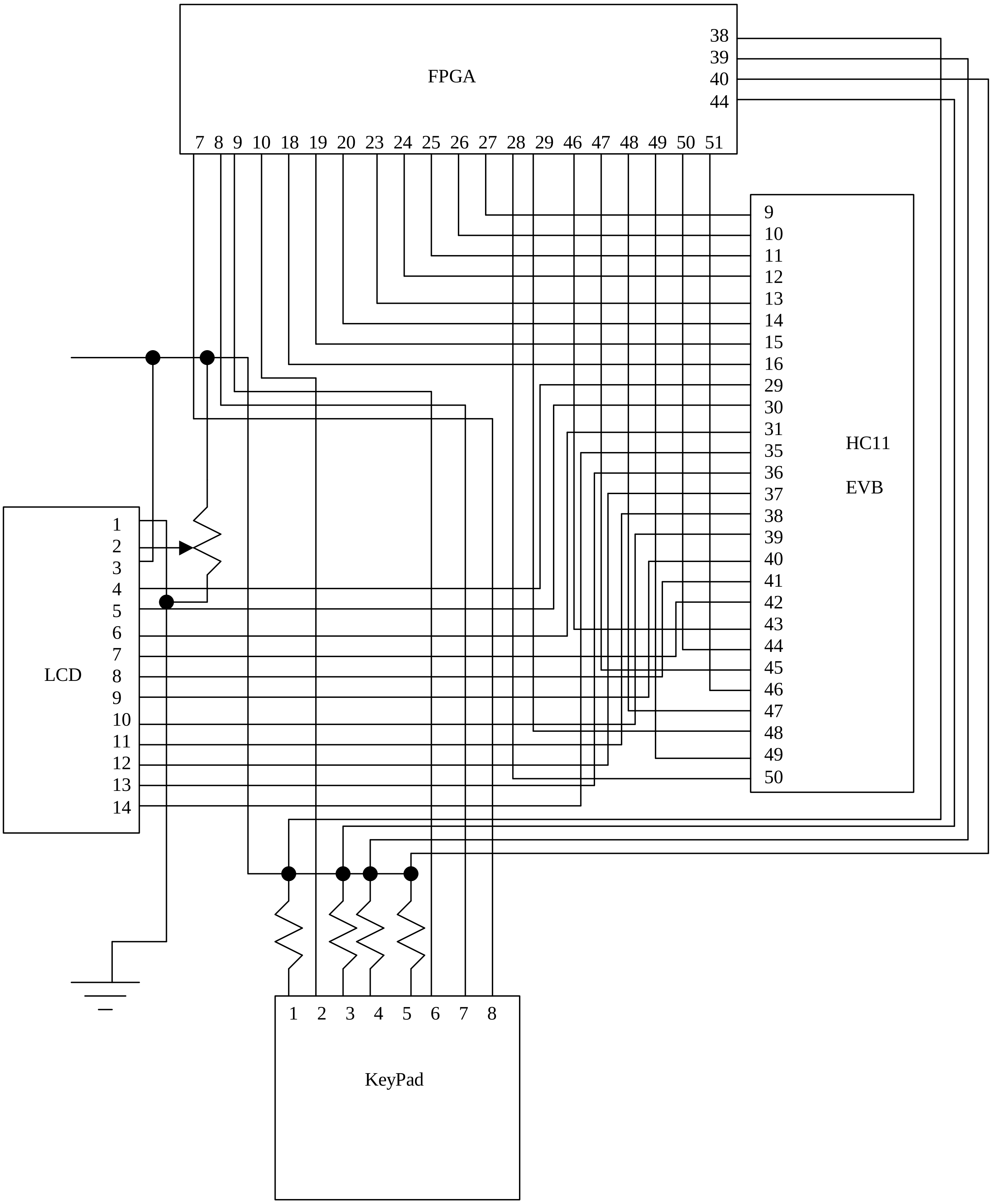
Command	*RS*					*Binary*				*Hex*
		D7	*D6*	*D5*	*D4*	*D3*	*D2*	*D1*	*D0*	
Clear Display	0	0	0	0	0	0	0	0	1	01
Display and Cursor Home	0	0	0	0	0	0	0	1	X	02 or 03
Character Entry Mode	0	0	0	0	0	0	1	I/D	S	04 to 07
Display/Cursor On/Off	0	0	0	0	0	1	D	U	B	08 to 0F
Display/Cursor Shift	0	0	0	0	1	D/C	R/L	X	X	10 to 1F
Function Set	0	0	0	1	8/4	2/1	10/7	X	X	20 to 3F
Set CGRAM Address	0	0	1	A	A	A	A	A	A	40 to 7F
Set Display Address	0	1	A	A	A	A	A	A	A	80 to FF
Write Data	1	D	D	D	D	D	D	D	D	00 to FF
I/D: 1=Increment, 0=Decrement				***R/L: 1=Right shift, 0=Left Shift***						
S: 1=Display shift on, 0=Display shift off				***8/4: 1=8 bit interface, 0=4 bit interface***						
D: 1=Display on, 0=Display off				***2/1: 1=2 line mode, 0=1 line mode***						
U: 1=Cursor underline on, 0=off				***10/7: 1=5x10 dot format, 0=5x7 dot format***						
B: 1=Blinking Cursor on, 0=no blinking				***D/C: 1=Set Display shift, 0=Set Cursor***						

Table 2: Command control codes. This table lists the commands necessary to operate the LCD. Setting appropriate bits sends recognizable commands to the LCD on board controller.

The following are some troubleshooting tips that have helped us implement the LCD with the HC11 EVB. To fully implement the LCD, the RS, E, and R/W signals must be timed appropriately, through the proper use of delays for setup and hold times. Attached in an appendix the reader can find the general timing diagrams to implement an LCD. However, through experimentation, we have found that these timing specs are inaccurate when applied to the HC11. Thus, in our code, the reader will note that we have employed considerably longer setup and hold times to actually operate the LCD. Generally speaking, we have found that delays between instructions should be around 1 to 2 mS, otherwise the display will not act properly. See the assembly code for more details. For testing the LCD, we found that using a simple protoboard and DIP switches worked quite well, as we just set the data and set the enable signal when necessary.

Schematics

The breadboard layout of our project is shown below. The HC11 and FPGA communicate over a 16 bit parallel connection. The FPGA sends data to the HC11. The HC11 sends no information back to the FPGA. The HC11 communicates to the LCD using 11 parallel bits. The keypad is connected to the FPGA using eight wires. Four bits are input; four bits are output. The pin outs of all devices can be seen in appendix C.


FPGA
38
39
40
44
7
8
9
10
18
19
20
23
24
25
26
27
28
29
46
47
48
49
50
51
HC11
EVB
9
10
11
12
13
14
15
16
29
30
31
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
LCD
1
2
3
4
5
6
7
8
9
10
11
12
13
14
KeyPad
1
2
3
4
5
6
7
8

Microcontroller Design

The following section will describe the use of the HC11 EVB in our design. The EVB in our design acted mainly as a LCD control look up table. State information was sent from the FPGA, and appropriate text was sent to the LCD. Thus, internally, the HC11 acted as follows. First, the HC11 polled 2 8 bit input ports, namely port C and port E. Once information was received from the FPGA via these 2 ports, the data received was assembled as a 16 bit state. The state was interpreted by a routine on the HC11. Once interpreted, other appropriate routines were called that effectively set a pointer to an appropriate place in memory. Then, ASCII characters were read from memory and sent via an 8 bit output port to the LCD, along with the corresponding control bits. The LCD then displayed the characters sent to it, and the HC11 went back to polling for a new state. The major subroutines listed above will be discussed in further detail below.

First, the polling routine will be discussed. As mentioned above, port C, configured as input, and port E were used in tandem to gather 16 bit data from the FPGA. As shown previously, port C and port E are both hard-wired to output pins on our FPGA board. The polling routine would store the data coming into these ports, and check them against the last state processed. If the new data were the same as the old, the routine would continue to poll port C and E. Once a new state had been received by the polling routine, the rest of the program would then be executed.

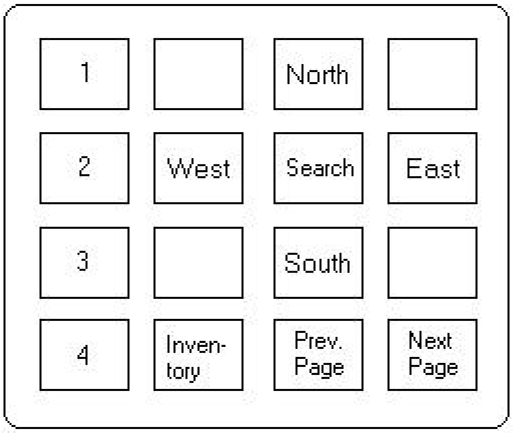
The next large routine is the interpreter. This routine took the 16 bit state data from the polling routine and operated on it to interpret the state information. For example, if the top nibble of the top byte of the two was set to 0x0, then the interpreter would decide it was a story line state, and set the pointer to the appropriate place in memory to display story line data. Other possible states include death states, search states, and various error states, such as bad key presses.

Thus, the interpreter would jump to appropriate subroutines based upon what kind of state was delivered from the FPGA. These subroutines all acted fundamentally the same, by generally looking at the lower 12 bits of state data and place a pointer at the place in memory that referenced the text for that particular state. For example, if the interpreter decided it was a story line state, it would jump to the story line subroutine. In this routine, the lower 12 bits of state would be interpreted further into which actual story page was being accessed, then move a pointer to the correct place in memory to send the correct ASCII characters to the LCD to display this pages story line.

Once the pointer had been set, the write data to LCD routine was accessed. This routine started at the place in memory where the pointer had been set by the previous routines, and simply sequentially sent 80 bytes of data to the LCD, corresponding to 80 characters. Thus, a full screen of text was sent to the LCD for every state processed. The pointer would simply be incremented to the next address, and the byte in memory written to the LCD via port B. Within this routine, various command signals such as Enable and Register Select were also sent via port A to the LCD. Please see the new hardware section for more information on command signals for the LCD.

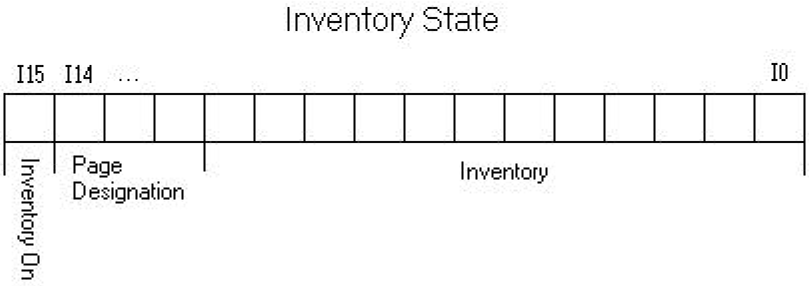
FPGA Design

The FPGA stores the finite state machine (FSM) data and takes keypad input. The keypad is the standard 16-button keypad used previously in an E155 lab. The keys are as follows:



Any buttons that are not labeled are not used. The FPGA de-bounces the keypad signal and interprets which key has been pressed and then decides the next state. The FPGA effectively has 3 state machines, the room state machine, the inventory state machine, and the error state machine. All the state machines depend on the keypad input and what state the game is in. Each state is 16 bits long. The FPGA sends only one of the three states at a time. Inventory states all have a leading 1 (i.e. 0x8001). Error states are 3FFF, 3FFE, and 3FFD. All other outputs denote room states. The logic in the FPGA determines which state to send to the HC11 depending on current state and user input. Two bits are saved to tell the FPGA whether it should send the Error State or the Inventory State. The error message takes precedence. If neither are to be sent, then the room state is sent. A description of the state machines follow.

Inventory State



The Inventory State saves the items the user has in inventory. The inventory on bit lets the HC11 know that it is looking at the inventory state. The page designation bits tell the HC11 which page of inventory to display. The last 12 bits designate what items the player has. If a bit is on, the user has the item. If it is off, the user does not have the item. If a user has an item, then it will be listed in inventory. If he does not have it, a blank line will be shown. As of now the bits in inventory represent the following items:

I0 : Rubber Ducky
I1 : 2.2 kOhm Resistor
I2 : Hamster
I3 : Bra
I4 : Flower
I5 : Sword
I6 : HC11 Reference Manual
I7 : Torch

Error State

The Error State gets sent to the HC11 when the ErrorMessage signal is high. The three possible error messages are :

3FFF: You cannot use that key.
3FFE: You cannot use that item.
3FFD: You do not have that item.

Room States

The room states are simply encoded starting at 0x0000 and counting up in the order in which you can encounter them. The room state by default gets sent to the HC11. Each room state has 80 characters in memory associated with it. Each room state has another room state associated with it, the searching the room state. The search state is encoded by a leading 0x4 (i.e. the search state for 0x0005 is 0x4005). Other special room state encodings are ones with leading 0x2s. These represent game endings. A 0x1 at the beginning of a room state represents that the user has picked up an item. In such a state bits 0 – 11 one-hot encode which item was just picked up. The method for choosing these state encodings is based on making it easy for us to distinguish types of state and to make it easier to reference locations in memory. The room state machine is given in Appendix B.

Verilog Implementation

The verilog code that implements the state machines is attached in Appendix B.

Results

We created a fun game. It is in fact super happy fun. The hardest part of the project was conserving memory. There is limited memory available to the HC11 and to one FPGA. Trying to make a game that was large enough to be fun required looking carefully at our resources and how to store data effectively. Overall we think our project turned out very well for us designing it and for everyone who has played it.

Appendix A: Assembly Code

The following assembly file includes the interpreter and polling routines, as well as the LCD driver routines.

```
*********************************
*   MicroP's Final            *
*   The Super Happy Fun Game  *
*   November 16, 2000         *
*   Authored by:        *
*      Ari Moradi       *
*      Ryan Stuck       *
*********************************

** Register Addresses

PORTA EQU   $1000
PORTB EQU   $1004
PORTC EQU   $1003
DDRC  EQU   $1007
PORTE EQU   $100A

** Port direction mask

PCCFG EQU   %00000000   *configs port c as input

** Inventory Page Mask

PMASK EQU   %01110000   *masks page bits in state data
IMASK EQU   %00000111   *masks inventory bits of interest
P3MASK      EQU   %00000001   *masks bit of interest from top bits

** Constants

CX11  EQU   $C4C4       *the CXnn's are compare values
CX12  EQU   $C4D8       *to see when to move on to the next
CX13  EQU   $C4EC       *inventory item.  these are used in
CX21  EQU   $C500       *set inventory functions
CX22  EQU   $C514
CX23  EQU   $C528
CX31  EQU   $C53C
CX32  EQU   $C550
CX33  EQU   $C564
CX41  EQU   $C578
CX42  EQU   $C58C
CX43  EQU   $C5A0
CXN   EQU   $C370

BADK  EQU   $3FFF       *badkey state compare value
BADI  EQU   $3FFE       *cant use item state compare value
DONTI EQU   $3FFD       *no item state compare value

BKOFF EQU   $C370       *all values labeled xxOFF are offsets
```

```
BIOFF EQU   $C3C0       *to memory locations where text is located
NIOFF EQU   $C410
STOFF EQU   $C700
SEOFF EQU   $D200
ENDOFF      EQU   $D8E0
INOFF EQU   $C5A0
ITOFF1      EQU   $C4B0
ITOFF2      EQU   $C4EC
ITOFF3      EQU   $C528
ITOFF4      EQU   $C564
BLOFF EQU   $C474
NPOFF EQU   $C35C

FSIZE EQU   $50         *size for full screen (80 characters) of text
ISIZE EQU   $14         *size for 1 line (20 characters) of text

BIADDR      EQU   $04         *all values labeled xxADDR are offsets on
the
TIADDR      EQU   $03         *zero page for temporary memory to store
AADDR EQU   $05         *values in the accumulators A,B,D, indices
BADDR EQU   $06         *X and Y, and inventory bytes
DADDR EQU   $07
XADDR EQU   $09
LXADDR      EQU   $0A
YADDR EQU   $0B
LYADDR      EQU   $0C

PAGE1 EQU   $C5A0       *PAGEx refers to a page of inventory
PAGE2 EQU   $C5F0
PAGE3 EQU   $C640
PAGE4 EQU   $C690

IT    EQU   $C460       *Got Item offset in memory
ITWR  EQU   $C488       *Where to write item received offset

** Ouput Masks
** b5 = register select command=0/data=1
** b4 = read=1/write=0
** b3 = enable=1

WRD   EQU   %00100000
WRDEN EQU   %00101000
WRC   EQU   %00000000
WRCEN EQU   %00001000

** Command Signals

CLEAR EQU   $01
HOME  EQU   $02
ENTRY EQU   $06
DISPON      EQU   $0C
FUNCT EQU   $38
SETCUR      EQU   $14

** Time delays for proper setup

HTIME EQU   $02
```

```
DTIME EQU   $40

** Main Function - calls necessary subroutines

      ORG   $C000
MAIN  JSR   INITLCD           *Initialize LCD
      LDAA  #PCCFG            *Port C config bits
      STAA  DDRC        *Store config in DDRC
      LDD   #$0001            *Initialize D register
      STD   DADDR       *Save in memory
      LDAA  #$00        *Put 0 in AA
      STAA  TIADDR            *Initialize Inventory
      STAA  BIADDR            *to empty
AGAIN LDX   #$0000            *Initialize X index
      JSR   POLL        *Poll for new state data
      JSR   CLEARI            *If new game, clear inventory
      JSR   SETIP1            *Set up inventory page 1
      JSR   SETIP2            *Set up inventory page 2
      JSR   SETIP3            *Set up inventory page 3
      JSR   SETIP4            *Set up inventory page 4
      LDD   DADDR       *Retrieve state from D reg
      JSR   INTERP            *Interpret state data
      ADDD  #FSIZE            *add frame size to D
      STAB  BADDR       *store end of frame in AB
      SUBD  #FSIZE            *subtract frame size
NEXT  XGDX              *put pointer to frame in index X
      LDAB  0,X         *load character from mem at X
      JSR   WRITED            *write character to LCD
      LDAA  #HTIME            *load delay time
      JSR   IDELAY            *delay for LCD setup time
      INX               *increment pointer
      XGDX              *put X in D
      CMPB  BADDR       *see if at end of frame
      BNE   NEXT        *if not at end of frame, next character
      BRA   AGAIN       *else search for new state
      SWI

** Write Data Function

*     ORG   $C000
WRITED      LDAA  #WRD        *Send write data to LCD
      STAA  PORTA
      JSR   STALL       *pause for hold time
      LDAA  #WRDEN            *Send enable data to LCD
      STAA  PORTA
      JSR   STALL       *pause for hold time
      STAB  PORTB       *Load data for LCD
      LDAA  #WRD        *Drop enable signal to LCD
      STAA  PORTA
      RTS

** Write Command Function

*     ORG   $C100
WRITEC      LDAA  #WRC        *Send write command to LCD
      STAA  PORTA
      JSR   STALL       *pause for hold time
      LDAA  #WRCEN            *Send enable command to LCD
      STAA  PORTA
      JSR   STALL       *pause for hold time
      STAB  PORTB       *Load command for LCD
      LDAA  #WRC        *Drop enable signal to LCD
      STAA  PORTA
      RTS

** Stall Function - to account for hold time

*     ORG   $C050
STALL LDY   #$0100
LOOP    DEY
      CPY   #$0000
      BNE   LOOP
      RTS

** Delay Function - to allow instruction completion
** lasts approx. 1 mS
*     ORG   $C150       *# of cycles
DELAY LDY   #$01E8            *1000 loops
MORE  DEY               *4
      NOP               *2
      NOP               *2
      NOP               *2
      NOP               *2
      CPY   #$0000            *5
      BNE   MORE        *3
      RTS

** Instruction Delay Function - delays AA mS
** for this instruction

*     ORG   $C200
IDELAY      DECA
      JSR   DELAY
      CMPA  #$00
      BNE   IDELAY
      RTS

** Initialization Function - inits LCD to write
** to 4x20 mode, and to increment address counter

*     ORG   $C220
INITLCD     LDAB  #DISPON           *Turn on display
      JSR   WRITEC
      LDAA  #HTIME
      JSR   IDELAY
      LDAB  #ENTRY            *Set entry mode
      JSR   WRITEC
      LDAA  #HTIME
      JSR   IDELAY
      LDAB  #FUNCT            *Set cursor/shift
      JSR   WRITEC
      LDAA  #HTIME
      JSR   IDELAY
      LDAB  #CLEAR            *Clear screen
      JSR   WRITEC
      LDAA  #HTIME
      JSR   IDELAY
      LDAB  #HOME       *Send cursor home
      JSR   WRITEC
      LDAA  #HTIME
      JSR   IDELAY
      RTS

** Polling Function - to poll for input state

*     ORG   $C300
POLL  LDAA  PORTC       *Load top bits
      LDAB  PORTE       *Load bottom bits
      CPD   DADDR       *Compare to see if changed
      BEQ   POLL        *If no change, continue to poll
      STD   DADDR       *else store in mem and continue
      RTS

** Interpret Function - decodes input state

*     ORG   $C400
INTERP      CPD   #BADK       *check if bad key press
      BEQ   BADKEY
      CPD   #BADI       *check if wrong item
      BEQ   BADIT
      CPD   #DONTI            *check if no item yet
      BEQ   DONTIT
      CMPA  #$00        *check if story line state
      BEQ   ST
      CMPA  #$10        *check if got item state
      BEQ   GET
      CMPA  #$20        *check if ending state
      BEQ   GEND
      CMPA  #$40        *check if search state
      BEQ   SE
      CMPA  #$80        *check if inventory state
      BGE   IN
BADKEY      JSR   KEYCHK
      BRA   BACK
BADIT JSR   CANTIT
      BRA   BACK
DONTIT      JSR   NOITEM
      BRA   BACK
ST    JSR   STORY
      BRA   BACK
GET   JSR   GETITEM
      BRA   BACK
GEND  JSR   GAMEEND
      BRA   BACK
SE    JSR   SEARCH
      BRA   BACK
IN    JSR   INVEN
BACK  RTS

** The following functions set the pointer in memory
** to an appropriate frame to display the proper message
** Bad Key Function - displays bad key message

*     ORG   $C450
KEYCHK      LDD   #BKOFF
      RTS

** Can't Item Function - displays cant use item message

*     ORG   $C610
CANTIT      LDD   #BIOFF
      RTS

** Don't Have Function - displays dont have item message

*     ORG   $C620
NOITEM      LDD   #NIOFF
      RTS

** Set Inventory Function - sets memory
** to display inventory properly

*     ORG   $C800
SETIP1      LDAB  BIADDR            *load bottom inventory bits
      ANDB  #IMASK            *mask for bottom 3 bits
      STAB  BADDR
      LDY   #PAGE1            *load inventory page 1 offset
      LDX   #ITOFF1           *load item 1 offset
NEXTS11     LDAA  0,X         *load character from mem at index x
      ANDB  #$01
      CMPB  #$01        *if this item is flagged as gotten
      BEQ   SETI11            *then save it in memory
      LDAA  BLOFF       *else write a blank to memory
SETI11      STAA  0,Y
      INY               *This actually continues for each item
      INX               *in the same manner, so no more commenting
      CPX   #CX11       *for this function or the next three like
      BNE   NEXTS11           *it will be noted.  Just know that they
      LDAB  BADDR       *all act the same, just with different
NEXTS12     LDAA  0,X         *offsets and inventory bit checks
      ANDB  #$02
      CMPB  #$02
      BEQ   SETI12
      LDAA  BLOFF
SETI12      STAA  0,Y
      INY
      INX
      CPX   #CX12
      BNE   NEXTS12
      LDAB  BADDR
NEXTS13     LDAA  0,X
      ANDB  #$04
      CMPB  #$04
      BEQ   SETI13
      LDAA  BLOFF
SETI13      STAA  0,Y
      INY
      INX
      CPX   #CX13
      BNE   NEXTS13
      LDX   #NPOFF            *load next/prev line offset
NEXTS14     LDAA  0,X
SETI14      STAA  0,Y         *save characters into inventory
      INY
      INX
      CPX   #CXN
      BNE   NEXTS14
      RTS

** Set Inventory Function - sets memory
** to display inventory properly

*     ORG   $C900
SETIP2      LDAB  BIADDR
      LSRB
      LSRB
      LSRB
      ANDB  #IMASK
      STAB  BADDR
      LDY   #PAGE2
      LDX   #ITOFF2
NEXTS21     LDAA  0,X
      ANDB  #$01
      CMPB  #$01
      BEQ   SETI21
      LDAA  BLOFF
SETI21      STAA  0,Y
      INY
      INX
      CPX   #CX21
      BNE   NEXTS21
      LDAB  BADDR
NEXTS22     LDAA  0,X
      ANDB  #$02
      CMPB  #$02
      BEQ   SETI22
      LDAA  BLOFF
SETI22      STAA  0,Y
      INY
      INX
      CPX   #CX22
      BNE   NEXTS22
      LDAB  BADDR
NEXTS23     LDAA  0,X
      ANDB  #$04
      CMPB  #$04
      BEQ   SETI23
      LDAA  BLOFF
SETI23      STAA  0,Y
      INY
      INX
      CPX   #CX23
      BNE   NEXTS23
      LDX   #NPOFF
```

```
NEXTS24     LDAA  0,X
SETI24      STAA  0,Y
      INY
      INX
      CPX   #CXN
      BNE   NEXTS24
      RTS

** Set Inventory Function - sets memory
** to display inventory properly

*     ORG   $CA00
SETIP3      LDAB  BIADDR
      LSRB
      LSRB
      LSRB
      LSRB
      LSRB
      LSRB
      ANDB  #IMASK
      LDAA  TIADDR
      ANDA  #P3MASK
      LSLA
      LSLA
      STAA  AADDR
      ORAB  AADDR
      STAB  BADDR
      LDY   #PAGE3
      LDX   #ITOFF3
NEXTS31     LDAA  0,X
      ANDB  #$01
      CMPB  #$01
      BEQ   SETI31
      LDAA  BLOFF
SETI31      STAA  0,Y
      INY
      INX
      CPX   #CX31
      BNE   NEXTS31
      LDAB  BADDR
NEXTS32     LDAA  0,X
      ANDB  #$02
      CMPB  #$02
      BEQ   SETI32
      LDAA  BLOFF
SETI32      STAA  0,Y
      INY
      INX
      CPX   #CX32
      BNE   NEXTS32
      LDAB  BADDR
NEXTS33     LDAA  0,X
      ANDB  #$04
      CMPB  #$04
      BEQ   SETI33
      LDAA  BLOFF
SETI33      STAA  0,Y
```

```
      INY
      INX
      CPX   #CX33
      BNE   NEXTS33
      LDX   #NPOFF
NEXTS34     LDAA  0,X
SETI34      STAA  0,Y
      INY
      INX
      CPX   #CXN
      BNE   NEXTS34
      RTS

** Set Inventory Function - sets memory
** to display inventory properly

*     ORG   $CB00
SETIP4      LDAB  TIADDR
      LSRB
      ANDB  #IMASK
      STAB  BADDR
      LDY   #PAGE4
      LDX   #ITOFF4
NEXTS41     LDAA  0,X
      ANDB  #$01
      CMPB  #$01
      BEQ   SETI41
      LDAA  BLOFF
SETI41      STAA  0,Y
      INY
      INX
      CPX   #CX41
      BNE   NEXTS41
      LDAB  BADDR
NEXTS42     LDAA  0,X
      ANDB  #$02
      CMPB  #$02
      BEQ   SETI42
      LDAA  BLOFF
SETI42      STAA  0,Y
      INY
      INX
      CPX   #CX42
      BNE   NEXTS42
      LDAB  BADDR
NEXTS43     LDAA  0,X
      ANDB  #$04
      CMPB  #$04
      BEQ   SETI43
      LDAA  BLOFF
SETI43      STAA  0,Y
      INY
      INX
      CPX   #CX43
      BNE   NEXTS43
      LDX   #NPOFF
NEXTS44     LDAA  0,X
```

```
SETI44      STAA  0,Y
      INY
      INX
      CPX   #CXN
      BNE   NEXTS44
      RTS


** Story Function - displays storyline

*     ORG   $C500
STORY LDAA  #FSIZE
      MUL
      ADDD  #STOFF
      RTS

**  Got Item Function - displays got item message
**  This function checks to see which item you received
**  then prints out a message saying you received it

*     ORG   $C550
GETITEM     ANDA  #$0F
      ORAB  BIADDR
      STAB  BIADDR
      ORAA  TIADDR
      STAA  TIADDR
      LDD   DADDR
      LDY   #$0000
      ANDA  #$0F
CHECKI      CMPB  #$01
      BEQ   DISPI
      INY
      LSRD
      CPY   #$000B
      BEQ   ENDI
      BRA   CHECKI
DISPI STY   YADDR
      LDAA  LYADDR
      LDAB  #ISIZE
      MUL
      ADDD  #ITOFF1
      XGDY
      LDAB  #$00
      LDX   #ITWR
MOREI LDAA  0,Y
      CMPB  #$02
      BGT   WRIT
      LDAA  BLOFF
WRIT  STAA  0,X
      INX
      INY
      INCB
      CMPB  #$14
      BNE   MOREI
      LDD   #IT
ENDI  RTS
```

```
** Game End Function - displays game over message

*     ORG   $C600
GAMEEND     LDAA  #FSIZE
      MUL
      ADDD  #ENDOFF
      RTS

** Search Function - displays search options

*     ORG   $C700
SEARCH      LDAA  #FSIZE
      MUL
      ADDD  #SEOFF
      RTS

** Invetory Function - displays current inventory

*     ORG   $C750
INVEN ANDA  #PMASK
      LSRA
      LSRA
      LSRA
      LSRA
      LDAB  #FSIZE
      MUL
      ADDD  #INOFF
      RTS

** Clear Inventory Function - if you die, this clears the
** inventory information

      ORG   $CE00
CLEARI      CPD   #$0000
      BNE   ENDCL
      LDAA  #$00
      STAA  TIADDR
      STAA  BIADDR
ENDCL RTS
```

The following assembly code is the storyboard, which will be written to memory to be accessed by the assembly file above.

```
*********************************
*   MicroP's Final            *
*   Story Line (in ASCII)     *
*   November 19, 2000         *
*   Authored by:        *
*   Ryan Stuck                *
*   Ari Moradi                *
*********************************

** Blank to be repeated when necessary
*     ORG   $DF50
*     FCC   " "

** Next/Previous page lines

      ORG   $C35C
      FCC   "prev            next"

** Bad Key Press Message

      ORG   $C370
      FCC   "You can't do that   "
      FCC   "                    "
      FCC   "here!               "
      FCC   "                    "

** Wrong Item Press Message

      ORG   $C3C0
      FCC   "You can't use that  "
      FCC   "                    "
      FCC   "item here !         "
      FCC   "                    "
** Don't Have Item Press Message

      ORG   $C410
      FCC   "You can't use what  "
      FCC   "Dummy !             "
      FCC   "you don't have,     "
      FCC   "                    "

** Item Pick Up screens

      ORG   $C460

* Any Item
      FCC   "You got the         "
      FCC   "                    "
      FCC   "                    "
      FCC   "                    "

** Total Inventory to be written later
```

```
*  by proggie in $DD00

      ORG   $C4B0

      FCC   "1) Rubber ducky     "
      FCC   "2) 2.2 kOhm Resistor"
      FCC   "3) Hamster          "
      FCC   "1) Sexy bra         "
      FCC   "2) Flower           "
      FCC   "3) Angry Axe        "
      FCC   "1) HC11 Manual      "
      FCC   "2) Torch            "
      FCC   "3) Item 9           "
      FCC   "1) Item 10          "
      FCC   "2) Item 11          "
      FCC   "3) Item 12          "

** Story screens

      ORG   $C700

* State 0 c700
      FCC   "   The Super Happy  "
      FCC   "         by         "
      FCC   "       Fun Game     "
      FCC   " A Moradi & R Stuck "
* State 1 c750
      FCC   "You wake up in a tub"
      FCC   "naked. There is a   "
      FCC   "and notice you are  "
      FCC   "door to the east.  E"
* State 2 c7a0
      FCC   "You find yourself   "
      FCC   "break from your     "
      FCC   "enjoying a nice     "
      FCC   "hectic morning.     "
* State 3 c7f0
      FCC   "You find yourself   "
      FCC   "spaceship.          "
      FCC   "on the deck of a    "
      FCC   "                   W"
* State 4 c640
      FCC   "You find yourself   "
      FCC   "rhinogooserufulus.  "
      FCC   "confronted by a mad "
      FCC   "                NSEW"
* State 5 c690
      FCC   "The rhino is happy. "
      FCC   "empty field.        "
      FCC   "You are now in an   "
      FCC   "                NSEW"
* State 6 c6e0
      FCC   "You find yourself   "
      FCC   "                    "
      FCC   "in a cabin.         "
      FCC   "                  EW"
* State 7 c730
```

```
      FCC   "You are in the      "
      FCC   "                    "
      FCC   "cabin's kitchen.    "
      FCC   "                   W"
* State 8 c780
      FCC   "You see a bridge.  A"
      FCC   "asks: What is your  "
      FCC   "troll comes out and "
      FCC   "favorite color?     "
* State 9 c7d0
      FCC   "You find yourself at"
      FCC   "moat.  The way over "
      FCC   "the foot of a giant "
      FCC   "is a drawn bridge. N"
* State 10 c820
      FCC   "You are confronted  "
      FCC   "of a forboding      "
      FCC   "by the giant doors  "
      FCC   "castle.           EW"
* State 11 c870
      FCC   "You are in the main "
      FCC   "castle.  So now     "
      FCC   "hall of an ancient  "
      FCC   "what to do?       NS"
* State 12 c8c0
      FCC   "You find yourself in"
      FCC   "room surrounded by  "
      FCC   "a medieval weapon's "
      FCC   "axes and swords.   N"
* State 13 c910
      FCC   "You are now in an   "
      FCC   "of forgotten things "
      FCC   "old library.  Tomes "
      FCC   "surround you.      S"
* State 14 c960
      FCC   "You find yourself in"
      FCC   "laboratory.  Bottles"
      FCC   "a magician's        "
      FCC   "are all about.    SE"
* State 15 c9b0
      FCC   "You are now in the  "
      FCC   "You see many flowers"
      FCC   "castle's courtyard. "
      FCC   "and benches.       N"
* State 16 ca00
      FCC   "You step into a dark"
      FCC   "man mumbles insanely"
      FCC   "dungeon.  A crazy   "
      FCC   "in the corner.     E"
* State 17 ca50
      FCC   "You step into a room"
      FCC   "gears and strange   "
      FCC   "filled with grinding"
      FCC   "bottles.            "
* State 18 caa0
      FCC   "You enter a tower   "
      FCC   "magician staring at "
      FCC   "room and find a     "
      FCC   "you angrily.        "
* State 19 caf0
      FCC   "Pieces of the       "
      FCC   "you.  You still feel"
      FCC   "magician lie about  "
      FCC   "uncomfortable.      "
* State 20 cb40
      FCC   "You see before a    "
      FCC   "who seems to have   "
      FCC   "beautiful princess  "
      FCC   "lost her top.       "
* State 21 cb90
      FCC   "The princess, now   "
      FCC   "at you.  What should"
      FCC   "decent, smiles shyly"
      FCC   "you do now?         "

** Search screens

      ORG   $D200

* Search 0 d200
      FCC   "You can:            "
      FCC   "  ducky             "
      FCC   "1 Pick up a rubber  "
      FCC   "2 Use toilet        "
* Search 1 d250
      FCC   "You can:            "
      FCC   "  ducky             "
      FCC   "1 Pick up a rubber  "
      FCC   "2 Use toilet        "
* Search 2 d2a0
      FCC   "You do not find     "
      FCC   "                    "
      FCC   "anything.           "
      FCC   "                    "
* Search 3 d2f0
      FCC   "You can:            "
      FCC   "2 Push FIRE button  "
      FCC   "1 Push LAND button  "
      FCC   "3 Get 2kOhm Resistor"
* Search 4 d340
      FCC   "You do not find     "
      FCC   "                    "
      FCC   "anything.           "
      FCC   "                    "
* Search 5 d390
      FCC   "You do not find     "
      FCC   "                    "
      FCC   "anything.           "
      FCC   "                    "
* Search 6 d3e0
      FCC   "You can:            "
      FCC   "2 Pick up a sexy bra"
      FCC   "1 Pick up a hamster "
      FCC   "                    "
```

```
* Search 7 d430
      FCC   "You see a microwave."
      FCC   "tasty, furry thing  "
      FCC   "If only you had some"
      FCC   "to eat right now.   "
* Search 8 d480
      FCC   "You can answer:     "
      FCC   "2 blue              "
      FCC   "1 yellow            "
      FCC   "3 fart              "
* Search 9 d4d0
      FCC   "You see that the    "
      FCC   "and a hamster wheel "
      FCC   "drawbridge is broken"
      FCC   "and plug are nearby."
* Search 10 d520
      FCC   "You can:            "
      FCC   "2 Knock on the door "
      FCC   "1 Pick up the flower"
      FCC   "                    "
* Search 11 d570
      FCC   "You can:            "
      FCC   "2 Go downstairs     "
      FCC   "1 Go upstairs       "
      FCC   "                    "
* Search 12 d5c0
      FCC   "You can:            "
      FCC   "2 Pick up the sword "
      FCC   "1 Pick up the axe   "
      FCC   "                    "
* Search 13 d610
      FCC   "You can:            "
      FCC   "  Reference Manual  "
      FCC   "1 Pick up HC11      "
      FCC   "2 Pick up SpaceQuest"
* Search 14 d660
      FCC   "You can:            "
      FCC   "  labeled 'Drink Me'"
      FCC   "1 Drink bottle      "
      FCC   "2 Eat the burrito   "
* Search 15 d6b0
      FCC   "You can:            "
      FCC   "2 Sit on a bench    "
      FCC   "1 Pick up the torch "
      FCC   "                    "
* Search 16 d700
      FCC   "You can:            "
      FCC   "2 Talk to crazy man "
      FCC   "1 Go back upstairs  "
      FCC   "                    "
* Search 17 d750
      FCC   "You see an HC11 on  "
      FCC   "You can:            "
      FCC   "the geared machines."
      FCC   "1 Press reset button"
* Search 18 d7a0
      FCC   "You do not find     "
      FCC   "                    "
      FCC   "anything.           "
      FCC   "                    "
* Search 19 d7f0
      FCC   "You see the magician"
      FCC   "but you feel he is  "
      FCC   "lying before you,   "
      FCC   "not yet dead.       "
* Search 20 d840
      FCC   "The princess seems  "
      FCC   "standing there with-"
      FCC   "very embarrassed    "
      FCC   "out a shirt.        "
* Search 21 d890
      FCC   "The princess looks  "
      FCC   "smile on her glowing"
      FCC   "at you with a happy "
      FCC   "face.               "

** Game Ending screens

      ORG   $D8E0

* Ending 1 d8e0
      FCC   "You accidentally    "
      FCC   "Oops.  GAME OVER !  "
      FCC   "blew up the earth ! "
      FCC   "                    "
* Ending 2 d930
      FCC   "You try to run, but "
      FCC   "pain of a horn      "
      FCC   "you feel the sharp  "
      FCC   "impaling you.       "
* Ending 3 d980
      FCC   "You pass over a hill"
      FCC   "Candyland.  You live"
      FCC   "and find you are in "
      FCC   "happily ever after. "
* Ending 4 d9d0
      FCC   "You wander into the "
      FCC   "recall you are naked"
      FCC   "frozen mountains,   "
      FCC   "and freeze and die. "
* Ending 5 dac0
      FCC   "As you watch the    "
      FCC   "see it expand and   "
      FCC   "hamster cooking, you"
      FCC   "explode into bits.  "
* Ending 5 dac0
      FCC   "You try to run past"
      FCC   "catches, kills, and "
      FCC   "the troll.  He      "
      FCC   "eats you.           "
* Ending 6 da20
      FCC   "You answer incorrect"
      FCC   "to disembowel you.  "
      FCC   "and the troll starts"
      FCC   "You die painfully.  "
* Ending 7 da70
      FCC   "You place the 2 kohm"
      FCC   "and feel electricity"
      FCC   "resistor in the plug"
      FCC   "cook your brain.    "
* Ending 8 db10
      FCC   "You put the hamster "
      FCC   "runs, the dawbridge "
      FCC   "in the wheel.  As it"
      FCC   "begins to drop.     "
* Ending 9 db60
      FCC   "A frenchman leans   "
      FCC   "taunt you in a very "
      FCC   "out and proceeds to "
      FCC   "unkind fashion.     "
* Ending 10 dbb0
      FCC   "You reach for the   "
      FCC   "blade slips through "
      FCC   "sword and slip.  The"
      FCC   "you like butter.    "
* Ending 11 dc00
      FCC   "You drink the bottle"
      FCC   "You are lucky that  "
      FCC   "and feel a bit sick."
      FCC   "didn't kill you.    "
* Ending 12 dc50
      FCC   "You scarf down the  "
      FCC   "you need that toilet"
      FCC   "burrito.  Suddenly  "
      FCC   "again and rush back."
* Ending 13 dca0
      FCC   "You take a seat on a"
      FCC   "enjoy the scenery   "
      FCC   "nearby bench and    "
      FCC   "around you.         "
* Ending 14 dcf0
      FCC   "You try to navigate "
      FCC   "dark but slip and   "
      FCC   "the stairs in the   "
      FCC   "smash your skull.   "
* Ending 15 dd40
      FCC   "The old man says:   "
      FCC   "Halitosis Man?  I   "
      FCC   "What is it you want,"
      FCC   "see, Mr. Stinkmouth."
* Ending 16 dd90
      FCC   "As you press the    "
      FCC   "begin to explode.   "
      FCC   "button, the bottles "
      FCC   "You die in flames.  "
* Ending 17 dde0
      FCC   "You reference the   "
      FCC   "you think is a bomb "
      FCC   "manual, defuse what "
      FCC   "and leave the room. "
* Ending 18 de30
```

```
      FCC   "You wield the torch "
      FCC   "manage to catch your"
      FCC   "bravely, but only   "
      FCC   "self on fire and die"
* Ending 19 de80
      FCC   "As you turn to walk "
      FCC   "stands up and blasts"
      FCC   "away, the magician  "
      FCC   "you to pieces.      "
* Ending 20 ded0
      FCC   "The princess smiles!"
      FCC   "the evil wizard and "
      FCC   "You have defeated   "
      FCC   "have won the game!  "
```

Appendix B: Verilog Code

```
// final.v
// top level module for e155 final project
// Ari Moradi and Ryan Stuck

module final (Clk, Reset, LED, Pollout, KeypadIn, ParallelOut) ;

input [3:0] KeypadIn;
input Clk, Reset ;

output [3:0] Pollout;
output [15:0] ParallelOut;
output [7:0] LED;

wire myclk;                   // myclk signal;  clock for all flops
wire NewData;                 // tells if a new button has been pressed
wire [3:0]data;               // keypadin data when newdata
wire [7:0]Count;

assign Count = 8'b10000000;   // delay for slowing down clock

// LED's show the bottom 8 bits of parallel data
assign LED = ParallelOut[7:0];

// creates myclk signal; sequential
assignMyClk amc(Clk, Count, myclk, Reset);

// takes myclk and input to do debouncing and stop/continue
// polling; sequential
getInput gi(myclk, Reset, KeypadIn, Pollout, NewData, data);

// interprets data for output to HC11; sequential
assignOuts ao(myclk, Reset, NewData, data, Pollout, ParallelOut);


endmodule


// assignmyclk.v
// slows down clock to help debounce keypad signal
// Ari Moradi and Ryan Stuck

module assignMyClk (Clk, Count, myclk, Reset) ;

input Clk, Reset ;
input [7:0] Count ;
output myclk ;

reg [12:0]myCount;
reg myclk;

// counts up until myCount reaches Count, then toggles myclk to slow
down clock
always@(posedge Clk or posedge Reset)
      if(Reset)
```

```
            begin
            myclk <= 0;
            myCount <= 0;
            end
      else if (myCount == {Count[7:0], 5'b00000})
            begin
            myclk <= ~myclk;
            myCount <= 0;
            end
      else
            myCount <= myCount + 1;

endmodule

// getinput.v
// module that debounces and detects the keypad signal
// Ari Moradi and Ryan Stuck

module getInput (myclk, Reset, KeypadIn, PollOut, NewData, Data) ;

input myclk ;
input Reset ;
input [3:0]KeypadIn;                      // row input from keypad
output [3:0]PollOut;                      // polling output to keypad
output NewData;                           // if a new button has been
pressed
output [3:0]Data;                   // row input on new keypress

reg PollOut;
reg KeepPolling;
reg [3:0]Data;
reg NewData;
reg sameKey;                              // if the user is holding
down a button

// takes pollout and keypadin to determine if a key has been pressed
always@(posedge myclk or posedge Reset)
      if(Reset)
            begin
            KeepPolling <= 1;
            NewData <= 0;
            Data <= 4'b1111;
            sameKey <= 0;
            PollOut <= 4'b1110;
            end
      // continues to poll if keepPolling
      else if (KeepPolling)
            // checks if a button has been pressed
            // 4'b111 means a button has not been pressed
            if (KeypadIn != 4'b1111)
                  begin
                  KeepPolling <= 0;
                  Data <= KeypadIn;
                  NewData <= 0;
                  end
            else
                  begin
                  KeepPolling <= 1;
                  NewData <= 0;
                  // cycles pollout
                  case(PollOut)
                        4'b1110: PollOut <= 4'b1101;
                        4'b1101: PollOut <= 4'b1011;
                        4'b1011: PollOut <= 4'b0111;
                        4'b0111: PollOut <= 4'b1110;
                        default: PollOut <= 4'b1110;
                  endcase
                  end
      // this is the check for the user holding down the key
      else if (KeypadIn == Data)
            if (~sameKey)
                  begin
                  NewData <= 1;
                  sameKey <= 1;
                  end
            else
                  NewData <= 0;
      else
            begin
            KeepPolling <= 1;
            sameKey <= 0;
            end

endmodule

// assignouts.v
// module that takes keypad input, determines what key was pressed,
// then determines the next state.  all of the state machine info
// is in this module.
// Ari Moradi and Ryan Stuck

module assignOuts (myclk, Reset, NewData, Data, Pollout, ParallelOut);

input myclk ;
input Reset ;
input NewData;                // tells if a new button has been pressed

input [3:0]Data;              // data in from keypad
input [3:0]Pollout;           // Polling signal for keypad
output [15:0]ParallelOut;     // the parallel data that goes to HC11

wire [7:0]signal;
// what button has been pressed

reg [15:0]RoomState;
// stores what room the player is in

reg [15:0]InventoryState;
// stores the inventory information

reg Inventory;
// tells if the user is looking at inventory

reg [15:0]ErrorState;
```

```
// any error message state that needs to be displayed

reg ErrorMessage;
// tells if an error message needs to be displayed

// signal is the combination of the rows in from the keypad and the
// pollout signal from the FPGA
assign signal = {Data[3:0], Pollout[3:0]};

// definintions of the keys
parameter ONE       = 8'b0111_0111; // 1
parameter UNUSED1   = 8'b0111_1011; // 2
parameter NORTH     = 8'b0111_1101; // 3
parameter UNUSED2   = 8'b0111_1110; // 12
parameter TWO       = 8'b1011_0111; // 4
parameter WEST      = 8'b1011_1011; // 5
parameter SEARCH    = 8'b1011_1101; // 6
parameter EAST      = 8'b1011_1110; // 13
parameter THREE     = 8'b1101_0111; // 7
parameter UNUSED3   = 8'b1101_1011; // 8
parameter SOUTH     = 8'b1101_1101; // 9
parameter UNUSED4   = 8'b1101_1110; // 14
parameter FOUR      = 8'b1110_0111; // 10
parameter INVENTORY = 8'b1110_1011; // 0
parameter PREVPAGE  = 8'b1110_1101; // 11
parameter NEXTPAGE  = 8'b1110_1110; // 15

// this is the finite state machine
always@(posedge myclk or posedge Reset)
      begin
      // on reset, the game starts at the start screen and the user has
      // no inventory
      if(Reset)
            begin
            RoomState   <= 16'h0000;
            InventoryState <= 16'h8000;
            Inventory <= 0;
            ErrorState <= 16'h3FFF;
            ErrorMessage <= 0;
            end
      // if there is a new button press then it interprets what
      // should happen
      else if (NewData)
            begin
            // if an error message is being displayed, then it returns
            // to the game
            if (ErrorMessage)
                  begin
                  ErrorMessage <= 0;
                  end
            // if the user is looking at inventory
            else if (Inventory)
                  // switches for which page the user is looking at
                  case (InventoryState[14:12])
                        // switches for keypresses
                        3'b000: case (signal)
                                    INVENTORY: Inventory <= 0;
```

```
NEXTPAGE: InventoryState[14:12]
      <= 001;
ONE:  if (InventoryState[0])
            if (RoomState ==
                   16'h0004)
                  begin
                  Inventory <= 0;
                  RoomState
                  <= 16'h0005;
                  end
            else
                  begin
                  Inventory <= 0;
                  ErrorMessage
                  <= 1;
                  ErrorState
                  <= 16'h3ffe;
                  end
      else
            begin
            Inventory <= 0;
            ErrorMessage <= 1;
            ErrorState <= 16'h3ffd;
            end
TWO:  if (InventoryState[1])
            if (RoomState ==
                   16'h0009)
                  begin
                  Inventory <= 0;
                  RoomState
                  <= 16'h2007;
                  end
            else
                  begin
                  Inventory <= 0;
                  ErrorMessage
                  <= 1;
                  ErrorState
                  <= 16'h3ffe;
                  end
      else
            begin
            Inventory <= 0;
            ErrorMessage <= 1;
            ErrorState <= 16'h3ffd;
            end
THREE:      if (InventoryState[2])
            if (RoomState ==
                   16'h0007)
                  begin
                  Inventory <= 0;
                  RoomState
                  <= 16'h2004;
                  end
            else if (RoomState ==
                   16'h0009)
                  begin
```

```
                              Inventory <= 0;
                              RoomState
                              <= 16'h2008;
                              end
                        else
                              begin
                              Inventory <= 0;
                              ErrorMessage
                              <= 1;
                              ErrorState
                              <= 16'h3ffe;
                              end
                  else
                        begin
                        Inventory <= 0;
                        ErrorMessage <= 1;
                        ErrorState <= 16'h3ffd;
                        end
            // default is for bad key press
            default:
                  begin
                  ErrorMessage <= 1;
                  ErrorState <= 16'h3fff;
                  end
      endcase
3'b001: case (signal)
            INVENTORY: Inventory <= 0;
            PREVPAGE: InventoryState[14:12]
                        <= 000;
            NEXTPAGE: InventoryState[14:12]
                        <= 010;
            ONE:  if (InventoryState[3])
                        if (RoomState ==
                               16'h0014)
                              begin
                              Inventory <= 0;
                              RoomState
                              <= 16'h0015;
                              end
                        else
                              begin
                              Inventory <= 0;
                              ErrorMessage
                              <= 1;
                              ErrorState
                              <= 16'h3ffe;
                              end
                  else
                        begin
                        Inventory <= 0;
                        ErrorMessage <= 1;
                        ErrorState <= 16'h3ffd;
                        end
            TWO:  if (InventoryState[4])
                        if (RoomState ==
                               16'h0015)
                              begin
                              Inventory <= 0;
                              RoomState
                              <= 16'h2014;
                              end
                        else
                              begin
                              Inventory <= 0;
                              ErrorMessage
                              <= 1;
                              ErrorState
                              <= 16'h3ffe;
                              end
                  else
                        begin
                        Inventory <= 0;
                        ErrorMessage <= 1;
                        ErrorState <= 16'h3ffd;
                        end
            THREE:      if (InventoryState[5])
                        if (RoomState ==
                               16'h0012)
                              begin
                              Inventory <= 0;
                              RoomState
                              <= 16'h0013;
                              end
                        else
                              begin
                              Inventory <= 0;
                              ErrorMessage
                              <= 1;
                              ErrorState
                              <= 16'h3ffe;
                              end
                  else
                        begin
                        Inventory <= 0;
                        ErrorMessage <= 1;
                        ErrorState <= 16'h3ffd;
                        end
            default:
                  begin
                  ErrorMessage <= 1;
                  ErrorState <= 16'h3fff;
                  end
      endcase
3'b010: case (signal)
            INVENTORY: Inventory <= 0;
            PREVPAGE: InventoryState[14:12]
                  <= 001;
            NEXTPAGE: InventoryState[14:12]
                  <= 011;
            ONE:  if (InventoryState[6])
                        if (RoomState ==
                               16'h0011)
                              begin
                              Inventory <= 0;
```

```
                                                      RoomState
                                                      <= 16'h2011;
                                                      end
                                                else
                                                      begin
                                                      Inventory <= 0;
                                                      ErrorMessage
                                                      <= 1;
                                                      ErrorState
                                                      <= 16'h3ffe;
                                                      end
                                          else
                                                begin
                                                Inventory <= 0;
                                                ErrorMessage <= 1;
                                                ErrorState <= 16'h3ffd;
                                                end
                                    TWO:  if (InventoryState[7])
                                                if (RoomState ==
                                                       16'h0013)
                                                      begin
                                                      Inventory <= 0;
                                                      RoomState
                                                      <= 16'h0014;
                                                      end
                                                else
                                                      begin
                                                      Inventory <= 0;
                                                      ErrorMessage
                                                      <= 1;
                                                      ErrorState
                                                      <= 16'h3ffe;
                                                      end
                                          else
                                                begin
                                                Inventory <= 0;
                                                ErrorMessage <= 1;
                                                ErrorState <= 16'h3ffd;
                                                end
                                    default:
                                          begin
                                          ErrorMessage <= 1;
                                          ErrorState <= 16'h3fff;
                                          end
                              endcase
                        3'b011: case (signal)
                                    INVENTORY: Inventory <= 0;
                                    PREVPAGE: InventoryState[14:12]
                                                <= 010;
                                    default:
                                          begin
                                          ErrorMessage <= 1;
                                          ErrorState <= 16'h3fff;
                                          end
                              endcase
// this error is for a bad inventory state, and should never happen
                        default:
                              begin
                              ErrorState <= 16'h3fff;
                              ErrorMessage <= 1;
                              Inventory <= 0;
                              end
                  endcase

            else case (RoomState)
// switches on room state if !inventory and !errorMessage
// some rooms are just display messages and automatically go to the
// next state, while others check what key is pressed.  All defaults
// are for bad key presses
                  16'h0000: begin
                          RoomState <= 16'h0001;
                          InventoryState <= 16'h8000;
                          end
                  16'h0001: case (signal)
                              SEARCH:    RoomState <= 16'h4001;
                              EAST:      RoomState <= 16'h0003;
                              INVENTORY: Inventory <= 1;
                              default:
                                    begin
                                    ErrorState <= 16'h3FFF;
                                    ErrorMessage <= 1;
                                    end
                          endcase
                  16'h0002: RoomState <= 16'h0001;
                  16'h0003: case (signal)
                              SEARCH:   RoomState <= 16'h4003;
                              INVENTORY: Inventory <= 1;
                              WEST:     RoomState <= 16'h0001;
                              default:
                                    begin
                                    ErrorState <= 16'h3FFF;
                                    ErrorMessage <=1;
                                    end
                          endcase
                  16'h0004: case (signal)
                              SEARCH:  RoomState <= 16'h4004;
                              INVENTORY: Inventory <= 1;
                              NORTH:   RoomState <= 16'h2001;
                              EAST:    RoomState <= 16'h2001;
                              WEST:    RoomState <= 16'h2001;
                              SOUTH:   RoomState <= 16'h2001;
                              default:
                                    begin
                                    ErrorState <= 16'h3FFF;
                                    ErrorMessage <=1;
                                    end
                          endcase
                  16'h0005: case (signal)
                              SEARCH:  RoomState <= 16'h4005;
                              INVENTORY: Inventory <= 1;
                              NORTH:   RoomState <= 16'h2002;
                              WEST:    RoomState <= 16'h2003;
                              EAST:    RoomState <= 16'h0006;
                              SOUTH:   RoomState <= 16'h0008;
                              default:
                                    begin
                                    ErrorState <= 16'h3fff;
                                    ErrorMessage <= 1;
                                    end
                          endcase
                  16'h0006: case (signal)
                              SEARCH:  RoomState <= 16'h4006;
                              INVENTORY: Inventory <= 1;
                              WEST:    RoomState <= 16'h0005;
                              EAST:    RoomState <= 16'h0007;
                              default:
                                    begin
                                    ErrorState <= 16'h3fff;
                                    ErrorMessage <= 1;
                                    end
                          endcase
                  16'h0007: case (signal)
                              SEARCH:  RoomState <= 16'h4007;
                              INVENTORY: Inventory <= 1;
                              WEST:    RoomState <= 16'h0006;
                              default:
                                    begin
                                    ErrorState <= 16'h3fff;
                                    ErrorMessage <= 1;
                                    end
                          endcase
                  16'h0008: case (signal)
                              SEARCH:  RoomState <= 16'h4008;
                              INVENTORY: Inventory <= 1;
                              SOUTH:   RoomState <= 16'h2005;
                              NORTH:   RoomState <= 16'h0005;
                              default:
                                    begin
                                    ErrorState <= 16'h3fff;
                                    ErrorMessage <= 1;
                                    end
                          endcase
                  16'h0009: case (signal)
                              SEARCH:  RoomState <= 16'h4009;
                              INVENTORY: Inventory <= 1;
                              NORTH:   RoomState <= 16'h0005;
                              default:
                                    begin
                                    ErrorState <= 16'h3fff;
                                    ErrorMessage <= 1;
                                    end
                          endcase

                  16'h000A: case (signal)
                              SEARCH:  RoomState <= 16'h400A;
                              INVENTORY: Inventory <= 1;
                              EAST:  RoomState <= 16'h0009;
                              WEST:  RoomState <= 16'h000B;
                              default:
                                    begin
                                    ErrorState <= 16'h3fff;
                                    ErrorMessage <= 1;
                                    end
                          endcase
                  16'h000B: case (signal)
                              SEARCH:  RoomState <= 16'h400B;
                              INVENTORY: Inventory <= 1;
                              EAST:  RoomState <= 16'h000A;
                              SOUTH: RoomState <= 16'h000C;
                              NORTH: RoomState <= 16'h000D;
                              default:
                                    begin
                                    ErrorState <= 16'h3fff;
                                    ErrorMessage <= 1;
                                    end
                          endcase
                  16'h000C: case (signal)
                              SEARCH:  RoomState <= 16'h400C;
                              INVENTORY: Inventory <= 1;
                              NORTH: RoomState <= 16'h000B;
                              default:
                                    begin
                                    ErrorState <= 16'h3fff;
                                    ErrorMessage <= 1;
                                    end
                          endcase
                  16'h000D: case (signal)
                              SEARCH: RoomState <= 16'h400D;
                              INVENTORY: Inventory <= 1;
                              SOUTH: RoomState <= 16'h000B;
                              default:
                                    begin
                                    ErrorState <= 16'h3fff;
                                    ErrorMessage <= 1;
                                    end
                          endcase
                  16'h000E: case (signal)
                              SEARCH: RoomState <= 16'h400E;
                              INVENTORY: Inventory <= 1;
                              EAST: RoomState <= 16'h000D;
                              SOUTH: RoomState <= 16'h000F;
                              default:
                                    begin
                                    ErrorState <= 16'h3fff;
                                    ErrorMessage <= 1;
                                    end
                          endcase
                  16'h000F: case (signal)
                              SEARCH: RoomState <= 16'h400F;
                              INVENTORY: Inventory <= 1;
                              NORTH: RoomState <= 16'h000E;
                              default:
                                    begin
                                    ErrorState <= 16'h3fff;
                                    ErrorMessage <= 1;
                                    end
                          endcase
                  16'h0010: case (signal)
```

```
            SEARCH: RoomState <= 16'h4010;
            INVENTORY: Inventory <= 1;
            EAST:  RoomState <= 16'h0011;
            default:
                  begin
                  ErrorState <= 16'h3fff;
                  ErrorMessage <= 1;
                  end
        endcase
16'h0011: case (signal)
            SEARCH: RoomState <= 16'h4011;
            INVENTORY: Inventory <= 1;
            default:
                  begin
                  ErrorState <= 16'h3fff;
                  ErrorMessage <= 1;
                  end
        endcase
16'h0012: case (signal)
            SEARCH: RoomState <= 16'h4012;
            INVENTORY: Inventory <= 1;
            default:
                  begin
                  ErrorState <= 16'h3fff;
                  ErrorMessage <= 1;
                  end
        endcase
16'h0013: case (signal)
            SEARCH: RoomState <= 16'h4013;
            INVENTORY: Inventory <= 1;
            NORTH: RoomState <= 16'h2013;
            SOUTH: RoomState <= 16'h2013;
            EAST:  RoomState <= 16'h2013;
            WEST:  RoomState <= 16'h2013;
            default:
                  begin
                  ErrorState <= 16'h3fff;
                  ErrorMessage <= 1;
                  end
        endcase
16'h0014: case (signal)
            SEARCH: RoomState <= 16'h4014;
            INVENTORY: Inventory <= 1;
            default:
                  begin
                  ErrorState <= 16'h3fff;
                  ErrorMessage <= 1;
                  end
        endcase
16'h0015: case (signal)
            SEARCH: RoomState <= 16'h4015;
            INVENTORY: Inventory <= 1;
            default:
                  begin
                  ErrorState <= 16'h3fff;
                  ErrorMessage <= 1;
                  end
        endcase
16'h1001: RoomState <= 16'h0001;
16'h1002: RoomState <= 16'h0003;
16'h1004: RoomState <= 16'h0006;
16'h1008: RoomState <= 16'h0006;
16'h1010: RoomState <= 16'h000A;
16'h1020: RoomState <= 16'h000C;
16'h1040: RoomState <= 16'h000D;
16'h1080: RoomState <= 16'h000F;
16'h2004: RoomState <= 16'h0007;
16'h2008: RoomState <= 16'h000A;
16'h2009: RoomState <= 16'h000A;
16'h200B: RoomState <= 16'h000E;
16'h200C: RoomState <= 16'h0002;
16'h200D: RoomState <= 16'h000F;
16'h200F: RoomState <= 16'h0010;
16'h2011: RoomState <= 16'h0010;
16'h4001: case (signal)
            SEARCH:  RoomState <= 16'h0001;
            ONE:
                  begin
                  RoomState <= 16'h1001;
                  InventoryState[0] <= 1;
                  end
            TWO:  RoomState <= 16'h0002;
            default:
                  begin
                  ErrorState <= 16'h3FFF;
                  ErrorMessage <= 1;
                  end
        endcase
16'h4003: case (signal)
            SEARCH: RoomState <= 16'h0003;
            ONE:  begin
                  RoomState <= 16'h0004;
                  end
            TWO:  begin
                  RoomState <= 16'h2000;
                  end
            THREE:      begin
                  RoomState <= 16'h1002;
                  InventoryState[1] <= 1;
                  end
            default:
                  begin
                  ErrorState <= 16'h3FFF;
                  ErrorMessage <= 1;
                  end
        endcase
16'h4004: case (signal)
            SEARCH: RoomState <= 16'h0004;
            default:
                  begin
                  ErrorState <= 16'h3FFF;
                  ErrorMessage <= 1;
                  end
        endcase
```

```
16'h4005: case (signal)
            SEARCH: RoomState <= 16'h0005;
            default:
                  begin
                  ErrorState <= 16'h3FFF;
                  ErrorMessage <= 1;
                  end
        endcase
16'h4006: case (signal)
            SEARCH: RoomState <= 16'h0006;
            ONE:  begin
                  RoomState <= 16'h1004;
                  InventoryState[2] <= 1;
                  end
            TWO:  begin
                  RoomState <= 16'h1008;
                  InventoryState[3] <= 1;
                  end
            default:
                  begin
                  ErrorState <= 16'h3fff;
                  ErrorMessage <= 1;
                  end
        endcase

16'h4007: case (signal)
            SEARCH: RoomState <= 16'h0007;
            default:
                  begin
                  ErrorState <= 16'h3FFF;
                  ErrorMessage <= 1;
                  end
        endcase
16'h4008: case (signal)
            SEARCH: RoomState <= 16'h0008;
            ONE:    RoomState <= 16'h2006;
            TWO:    RoomState <= 16'h0009;
            default:
                  begin
                  ErrorState <= 16'h3fff;
                  ErrorMessage <= 1;
                  end
        endcase
16'h4009: case (signal)
            SEARCH: RoomState <= 16'h0009;
            default:
                  begin
                  ErrorState <= 16'h3fff;
                  ErrorMessage <=1;
                  end
        endcase
16'h400A: case (signal)
            SEARCH: RoomState <= 16'h000A;
            ONE:        begin
                  RoomState <= 16'h1010;
                  InventoryState[4] <= 1;
                  end
            TWO:  RoomState <= 16'h2009;
            default:
                  begin
                  ErrorState <= 16'h3fff;
                  ErrorMessage <= 1;
                  end
        endcase
16'h400B: case (signal)
            SEARCH: RoomState <= 16'h000B;
            ONE:  RoomState <= 16'h0012;
            TWO:        if (InventoryState[7])
                        RoomState <= 16'h0010;
                  else
                        RoomState <= 16'h200E;
            default:
                  begin
                  ErrorState <= 16'h3fff;
                  ErrorMessage <= 1;
                  end
        endcase
16'h400C: case (signal)
            SEARCH: RoomState <= 16'h000C;
            ONE:  begin
                  RoomState <= 16'h1020;
                  InventoryState[5] <= 1;
                  end
            TWO:  RoomState <= 16'h200A;
            default:
                  begin
                  ErrorState <= 16'h3fff;
                  ErrorMessage <= 1;
                  end
        endcase
16'h400D: case (signal)
            SEARCH: RoomState <= 16'h000D;
            ONE:  begin
                  RoomState <= 16'h1040;
                  InventoryState[6] <= 1;
                  end
            TWO:  RoomState <= 16'h000E;
            default:
                  begin
                  ErrorState <= 16'h3fff;
                  ErrorMessage <= 1;
                  end
        endcase
16'h400E: case (signal)
            SEARCH: RoomState <= 16'h000E;
            ONE:  RoomState <= 16'h200B;
            TWO:  RoomState <= 16'h200C;
            default:
                  begin
                  ErrorState <= 16'h3fff;
                  ErrorMessage <= 1;
                  end
        endcase
16'h400F: case (signal)
```

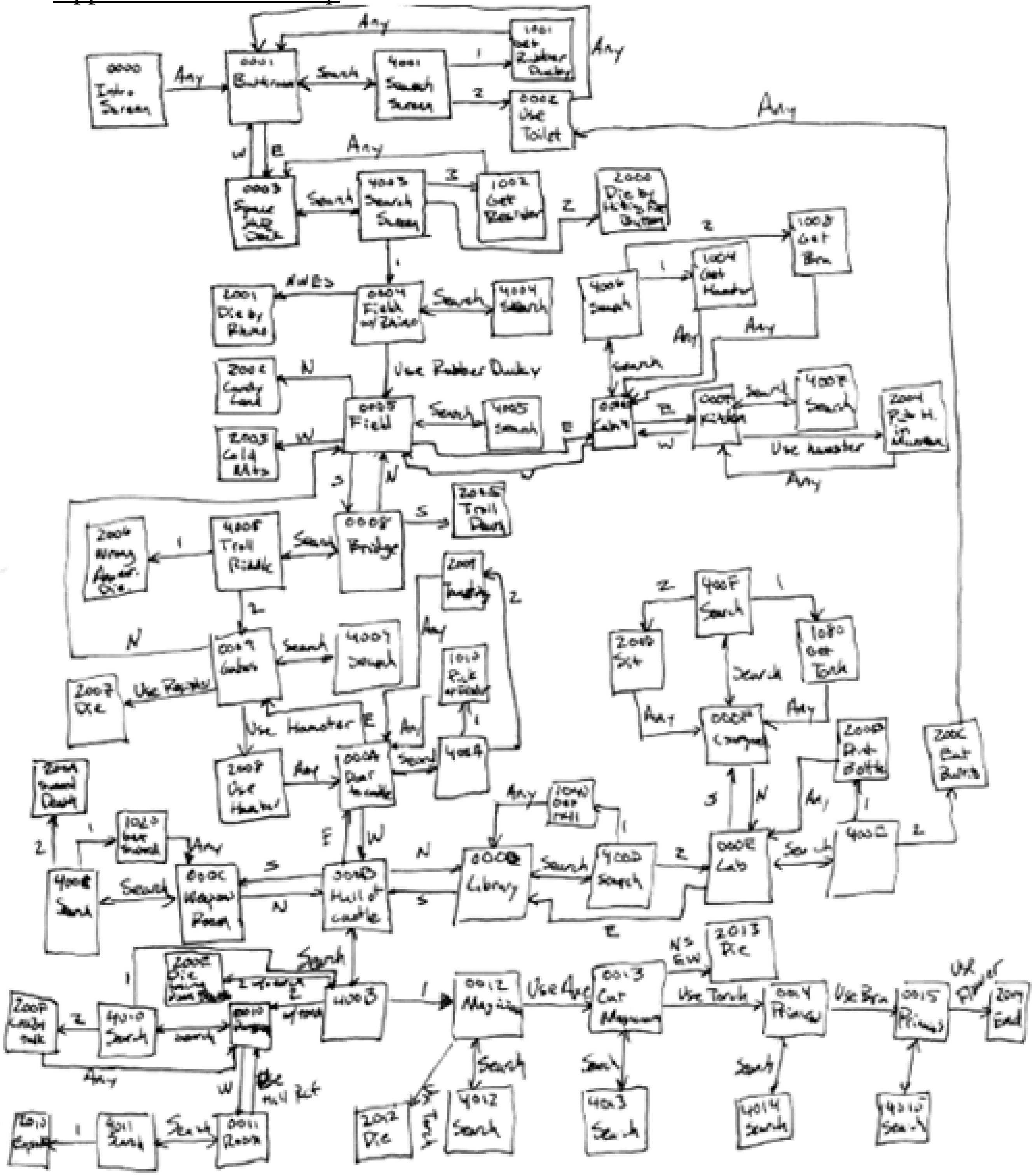
```
            SEARCH: RoomState <= 16'h000F;
            ONE:  begin
                  RoomState <= 16'h1080;
                  InventoryState[7] <= 1;
                  end
            TWO:  RoomState <= 16'h200D;
            default:
                  begin
                  ErrorState <= 16'h3fff;
                  ErrorMessage <= 1;
                  end
        endcase
16'h4010: case (signal)
            SEARCH: RoomState <= 16'h0010;
            ONE:  RoomState <= 16'h000B;
            TWO:  RoomState <= 16'h200F;
            default:
                  begin
                  ErrorState <= 16'h3fff;
                  ErrorMessage <= 1;
                  end
        endcase
16'h4011: case (signal)
            SEARCH: RoomState <= 16'h0011;
            ONE:  RoomState <= 16'h2010;
            default:
                  begin
                  ErrorState <= 16'h3fff;
                  ErrorMessage <= 1;
                  end
        endcase
16'h4012: case (signal)
            SEARCH: RoomState <= 16'h0012;
            default:
                  begin
                  ErrorState <= 16'h3fff;
                  ErrorMessage <= 1;
                  end
        endcase
16'h4013: case (signal)
            SEARCH: RoomState <= 16'h0013;
            default:
                  begin
                  ErrorState <= 16'h3fff;
                  ErrorMessage <= 1;
                  end
        endcase
16'h4014: case (signal)
            SEARCH: RoomState <= 16'h0014;
            default:
                  begin
                  ErrorState <= 16'h3fff;
                  ErrorMessage <= 1;
                  end
        endcase
16'h4015: case (signal)
            SEARCH: RoomState <= 16'h0015;
```

```
                              default:
                                    begin
                                    ErrorState <= 16'h3fff;
                                    ErrorMessage <= 1;
                                    end
                          endcase
                  default: RoomState <= 16'h0000;
            endcase
            end
      else
            begin
            RoomState <= RoomState;
            end
      end

// if (ErrorMessage) ParallelOut = ErrorState;
// else if (Inventory) ParallelOut = InventoryState;
// else ParallelOut = RoomState;
assign ParallelOut = {32{ErrorMessage}}&ErrorState | {32{~ErrorMessage
& Inventory}}&InventoryState | {32{~ErrorMessage&
~Inventory}}&RoomState ;

endmodule
```

Appendix C: Pin Outs

HC11		FPGA	
Pin #	Function	Pin #	Function
9-16	Port C (input) 9=low bit through 16=high bit	7	Column 1 (KP)
29	Register Select 1=instruction 0=data	8	Column 3 (KP)
30	Read/nWrite	9	Column 2 (KP)
31	Enable (high)	10	Column 4 (KP)
35-42	Port B (output) 35=high bit through 42=low bit	18	State b15
43	Port E b0	19	State b14
44	Port E b4	20	State b13
45	Port E b1	23	State b12
46	Port E b5	24	State b11
47	Port E b2	25	State b10
48	Port E b6	26	State b9
49	Port E b3	27	State b8
50	Port E b7	28	State b0
LCD		29	State b2
1	Vss (ground)	37	Row 4 (KP)
2	Vee (0-5V) Contrast adjust	38	Row 3 (KP)
3	Vdd (+5V)	39	Row 2 (KP)
4	Register Select	40	Row 1 (KP)
5	Read/nWrite	46	State b7
6	Enable (high)	47	State b5
7-14	Data (I/O)	48	State b3
KeyPad		49	State b1
1	Row 4	50	State b4
2	Column 1	51	State b6
3	Row 1		
4	Row 3		
5	Row 2		
6	Column 2		
7	Column 3		
8	Column 4		

Appendix D: Game Map


0000 Intro Screen
Any
0001 Bathroom
Search
4001 Search Screen
1
1001 Get Rubber Ducky
2
0002 Use Toilet
Any
Any
Any
W
E
0003 Space SAR Deck
Search
4003 Search Screen
1
1002 Get Raider
2
2000 Die by Hitting Button
Any
1
2001 Die by Rhino
NWES
0004 Field w/ Rhino
Search
4004 Search
Use Rubber Ducky
2002 Candy Land
N
2003 Cold Mts
W
0005 Field
Search
4005 Search
E
0006 Cabin
Search
4006 Search
1
1004 Get Hamster
2
1003 Get Brn
Any
Any
E
W
0007 Kitchen
Search
4007 Search
Use hamster
2004
Any
S
N
0008 Bridge
S
2006 Troll Dies
4008 Troll Riddle
Search
1
2005 Wrong Answer Die
2
N
0009 Gates
Search
4009 Search
Use Hamster
2007 Die
Use Hamster
E
2008 Use Hamster
Any
000A Door to castle
Search
400A
Any
Any
1
1010
2
2009
400F Search
2
1
2008
1080 Get Torch
Search
Any
Any
000F Courtyard
200D Drink Bottle
200C Eat Bats
S
N
Any
1
400E
2
Search
000E Lab
Any
1040
1
400D Search
Search
2
000D Library
N
S
E
E
W
000B Hall of castle
S
N
000C Weapon Room
Search
400B Search
1
1020
Any
2
200A
Search
200E Die
2
4003
1
0012 Magician
Use Axe
0013
Use Torch
N S E W
2013 Die
0014 Prince
Use Brn
0015 Prince
Use
2014 End
200F
2
4010 Search
Search
0010
Any
W
Hall Rat
2012 Die
Search
4012 Search
Search
4013 Search
Search
4014 Search
4015 Search
2010 Escape
1
4011 Search
Search
0011 Room