TamaMudder!

Final Project Report
December 8, 2000
E155

Tina Wang and Carrie Chung

Abstract:
The TamaMudder is a virtual pet whose life is based on the life of a Mudder. The goal is to survive long enough to get to graduation. The owner of the TamaMudder must see that the TamaMudder gets adequate amounts of Food, Sleep and maintains a healthy amount of Tan and Happiness. They do so by making the TamaMudder do various activities by pushing the different activity buttons on the game board. The health of the TamaMudder is displayed on a grid of LED's. Another set of LED's will indicate which, if any, activity is currently being executed. The game is controlled and executed by a microcontroller. The microcontroller receives inputs from the game board and outputs display data to an FPGA. The FPGA decodes the data and outputs it to the LED display.

Table of Contents

Introduction... 4
Schematics.. 5
Microcontroller Design 7
FPGA Design.. 9
Results 11
Appendix A – HC11 assembly code 12
Appendix B-Verilog.. 18

Figures and Tables

Figure 1: Overall System Block Diagram 4
Figure 2: FPGA Interface 5
Figure 3: Breadboard Schematic.. 5
Figure 4:Printed Circuit Board and HC11 Schematics Pin Assignments 6
Figure 5:Shift Register 9
Table 1: Inputs and outputs of HC11 7
Table 2: Game Parameters – Initial Settings 7
Table 3: Inputs and Outputs of FPGA... 9
Table 4: Information Bits Used 10

Introduction

In looking at the many of the other projects and ideas, all seemed interesting but few of them were cute. So we thought a Tamagotchi-style game would be a cute contrast to the many gadgets and games that we saw in both this year and last year's reports.

The TamaMudder is a game based on the Tamagotchi virtual pets. The basic premise is that the player needs to take care of the virtual pet, as they would a real pet, and gets to watch it grow and evolve. For the TamaMudder, the player will help the TamaMudder survive long enough to reach graduation. As with the Tamagotchi pets, the TamaMudder will die if the player neglects to take care of it. For example, if the player never sends the TamaMudder to Platt, it will starve to death.

In the TamaMudder world, each hour lasts ½ second in the real world and it only takes 4 weeks to reach graduation, as opposed to 4 years in the real world. Like a real Mudder, it must sleep, eat, do homework, and maintain a decent tan in order to survive to reach graduation.

The game hardware consists of a Motorola HC11 controller, a Xilinx Spartan FPGA, toggle switches and several LED's assembled together on a protoboard. The HC11 controller polls for user input, keeps track of which activity is running and the different health levels of the TamaMudder as well as the overall life/grade level. The FPGA decodes data from the HC11 controller and controls the LED display.

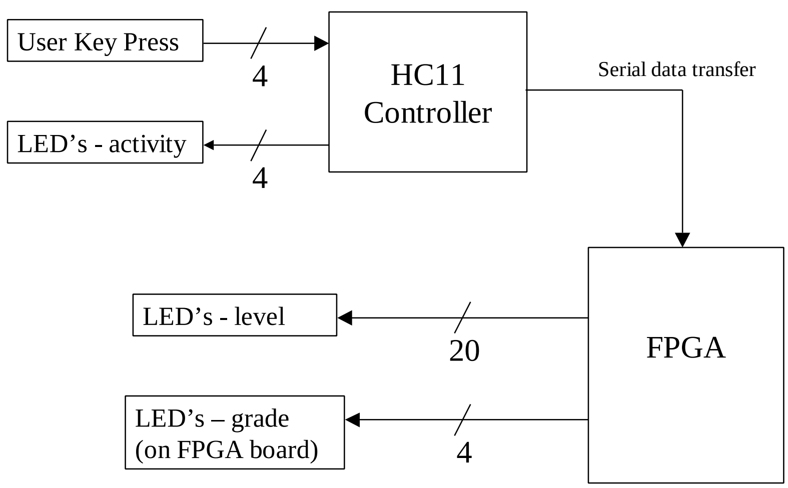

Figure 1: Overall System Block Diagram

Schematics

We tried to keep the game design as simple as possible. Therefore all the LED's connected directly to a signal pin on the FPGA instead of having the LED's multiplexed. We were worried that we were going to run out of pins, but our design was small enough to allow us the luxury of controlling the LED's directly. Since the HC11 controls the user input, we connected the user input switches and activity display LED's directly to the HC11 controller, bypassing the FPGA which would have acted as a wire anyway.

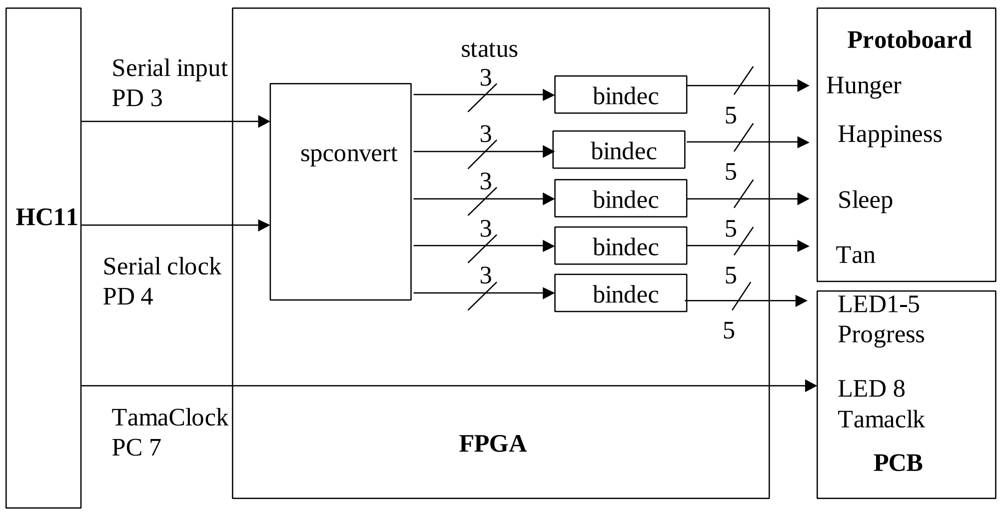

Figure 2: FPGA Interface

The HC11 outputs data to be decoded in the FPGA through both the serial port and a parallel port pin. The serial port sends all the LED display data to the FPGA in groups of 5 8-bit numbers. The parallel port sends the TamaClk signal to the FPGA which acts as a wire to display the pulse on the 8th LED of the FPGA board.

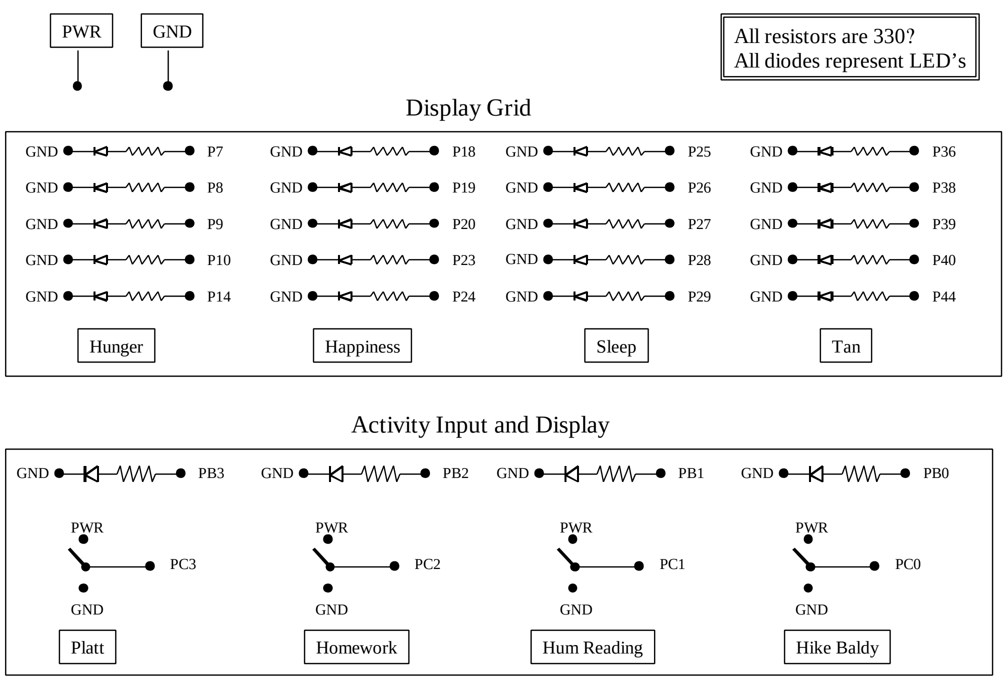

Figure 3: Breadboard Schematic

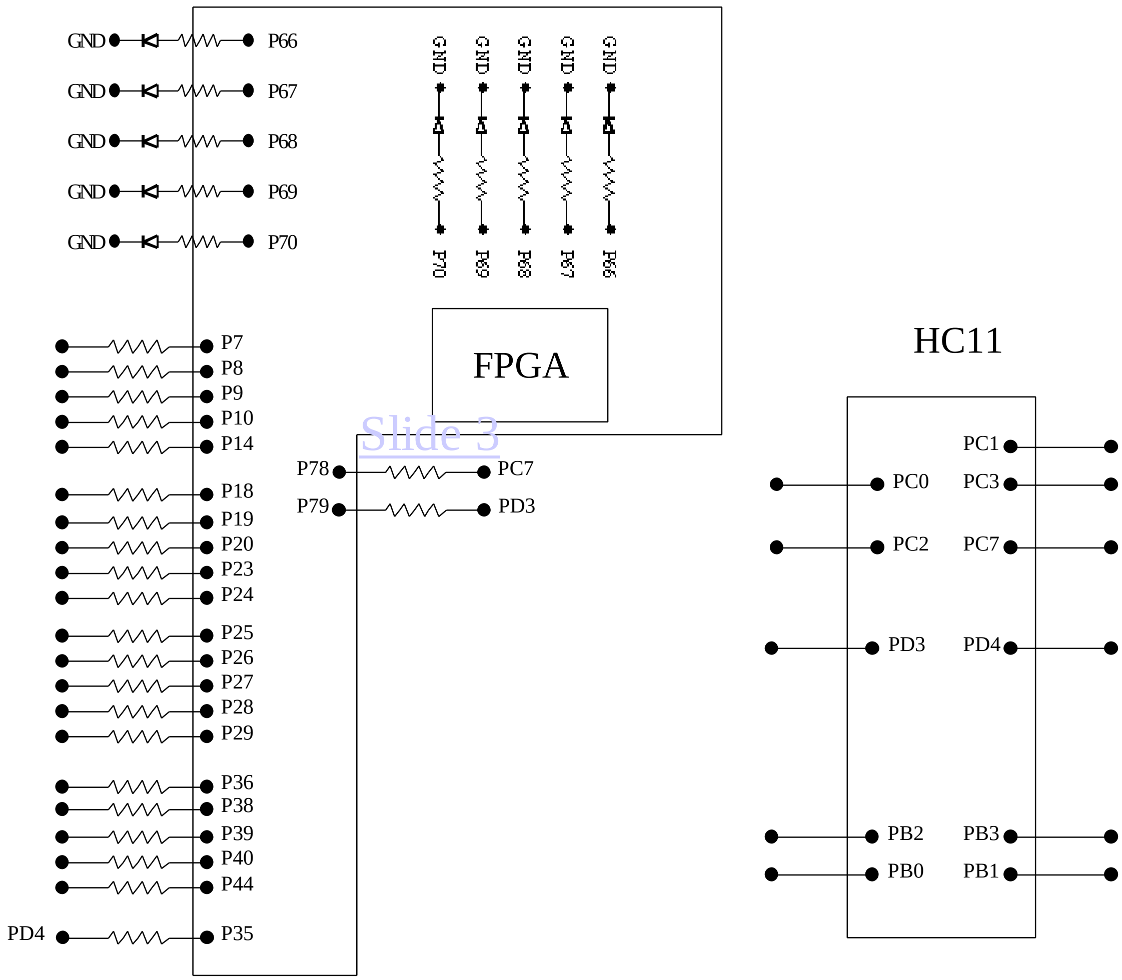

Figure 4:Printed Circuit Board and HC11 Schematics Pin Assignments

Microcontroller Design

Please refer to Final2.asm in Appendix A when reading this section.

As previously stated, the HC11 takes in a 4-bit, one-hot, parallel user input and outputs a 4-bit, one-hot, parallel display showing which, if any, activity is currently being played. The HC11 will also output the current health levels and grade at every TamaClk tick, which means the display will refresh every half second. The display output is sent serially to the FPGA. The TamaClk is sent via a single output parallel pin to the FPGA where is it displayed on an LED on the FPGA board.

INPUT	OUTPUT
Parallel port C – Activity/User Input (PC 0-3)	Serial port, Serial clk (PD 3,4)
	Parallel port B – activity output (PB 0-3)
	Parallel port C – TamaClk (PC 7)

Table 1: Inputs and outputs of HC11

<u>Variables</u>

The HC11 assembly code mainly consists of a series of counters with different events occurring at different times and different dependencies on other counters. There is a life counter (LIFE), that counts by the hour up to 1 week and then the life week (LIFEWK) counter will increment while the life counter resets. Each health level has it's own counter because they increment and decrement at different rates. Those counters are stored in Game Parameters table stored in memory. When an activity starts, it also depends on a counter to keep track of the Activity until it reaches the set Duration. These counters are also stored in the Game Parameters table. While an activity is being played, an internal flag (AFLG) is toggled to prevent any further key presses from registering until the activity is done with. When the activity is done, it will increment the associated health Level up to the Max Level. The user input is always stored in a temporary memory location (TPORTC) and if an activity is being played, the program will bypass scanning for a user input and just read from the temporary memory location, in effect, freezing the user input until the activity is finished.

Status:		**Hunger**	**Happiness**	**Sleep**	**Tan**
Max Level	CONST	5	5	5	5
Level	VAR	5	5	5	5
Lcount	VAR	0	0	0	0
Time2Drop	CONST	8	24	24	48
Activity:		**Platt**	**Homework**	**Hum Reading**	**Hike Baldy**
Duration	CONST	1	4	8	6
Activity	VAR	0	0	0	0

Table 2: Game Parameters – Initial Settings

<u>Program Structure:</u>

Initialization:

- Store Game Parameters into memory
- Clear all variables (LIFE, LIKEWK, AFLG, TPORTC, TAMACLK)
- Initialize parallel port outputs

Main Program Loop

- Poll for user input if Activity Flag != 0
- Call `activity` subroutine if valid key press detected. If not a valid key press, TPORTC = 0.
- Toggle TamaClk
- Call `Main2` subroutine.
- Output TPORTC to activity LED display.

- Call `sec` subroutine for ½ second delay

Main2 subroutine – (computes the 5 8-bit serial outputs for the LED display, 4 levels + 1 grade)

- Starts the beginning of the Game Parameter table. For each category, calls `level` to compute the current level and increment the counter. Then calls `increm` to increment the pointer stored in accumulator X to the next category.
- Call `grade` subroutine to compute current grade.

Grade subroutine

- If LIFE counter = 168 hours (= 24 hours x 7 days), reset LIFE to 0 and increment LIFEWK.
- Else LIFE = LIFE+1
- Send LIFEWK to serial output by calling `serial`
- If LIFEWK = 5, then call `win`

Serial subroutine

- Outputs data stored at 0,X to serial port

Activity subroutine – (is only called when an activity is occurring)

- Uses the category parameters stored starting at 0,X
- If Activity variable < Duration constant, increment Activity and toggle Activity Flag on.
- Else if Activity variable >= Duration constant, activity is finished, reset parameters and flag, increment corresponding health level by one unless already at Max Level.

Level subroutine -

- Uses the category parameters stored starting at 0,X.
- Checks if Level Counter (LCount) is equal to Time2Drop. If equal, the decrement Level and reset Level Counter. Else increment Level Counter by one.
- If the TamaMudder stays at Level 1 for 24 hours, then call `die` subroutine.
- Send Level to serial output by calling `serial`

FPGA Design

*Please refer to Appendix B when reading this section.

The main functions of the FPGA board are

? Conversion of the serial input from HC11 into 5 8-bit binary numbers.
? Decoding the binary input from HC11 and output 4 5-bit decimal display.

INPUT	OUTPUT
Serial port, Serial clk(PD3,4)	4 5-LED Level Display (to protoboard)
Parallel port C-TamaClk(PC 7)	5-LED Progress Display (PCB LED1-5)
	TamaCLk (PCB LED8)

Table 3: Inputs and Outputs of FPGA

To perform these functions, the following modules were created in verilog.

1. *spconvert.v*: input: serial data, serial clock
   output: 40-bit data
   spconvert takes in groups of 5-8 bit serial data and converts them into 40-bit data using a series of shift registers and a serial clock (from HC11).

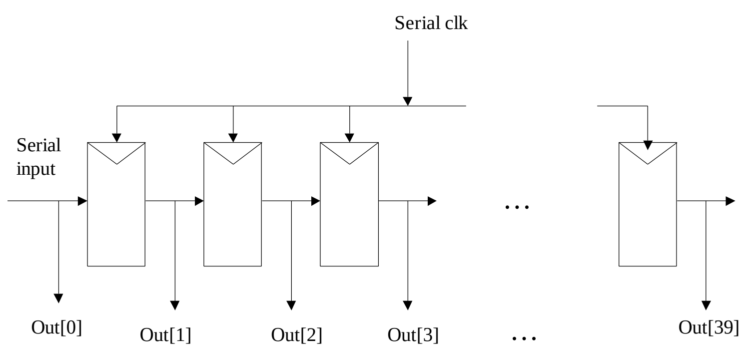

Figure 5:Shift Register

2. *bindec.v*: input: 3-bit binary
   output: 5-bit decimal representation
   This module takes in a 3-bit binary number and converts into a 5-bit decimal display. For example, inputs 011, 101 are converted into 00111,11111, respectively.

3. *status.v*: input: serial data, serial clk, tamaclk
   output: 4 5-bit Status, 1 5-bit Progress, 1-bit tamaclk.
   This is the top-level module of the FPGA where the previous modules, *spconvert* and *bindec*, are called and used for all the inputs and outputs on Table3.
   First, *status* takes in serial data from the HC11 serial port (PD3). This data and Serial clk (PD 4) go into *spconvert*, and become 40-bit (5 8-bit data). Then, *bindec* takes the least significant 3 bits of each group of 8 to control 4 5-level Status displays and a 5-level Progress display. For example, Bits 0,1,2 from the first eight of the 40 bits (7:0), and Bits 8,9,10 from the second 8 bits (15:7) are used to control Progress and Tan, respectively. The table below shows the bits used for Status and Progress displays. Also, Tamaclk, which is used to display the time in TamaMudder world,

BIT NUMBER	OUTPUT
34:32	Hunger
26:24	Happiness
18:16	Sleep
10:8	Tan
2:0	Progress

Table 4: Information Bits Used

Results

TamaMudder was successfully built as planned and meets all the requirements stated in our proposal. However, while testing the system, we realized that the initial game specifications were not sufficient to make the game as challenging as we expected.. Fortunately the game parameters are easy to change. In order to increase the level of difficulty, all one needs to do is modify the numbers stored into memory.(See Appendix A) For example, one could make the TamaMudder deteriorate faster, or have activities take longer to complete, to make it more interesting.

One deviation from the proposal is the usage of the FPGA board. Initially, we planned to have the FPGA freeze the activity control input from the breadboard upon detecting a valid user input, then releasing it when the HC11 signaled the completion of the activity. Eventually it turned out that we were trying to design an asynchronous circuit. We decided that it was less complicated to delegate the activity/user input detection to the HC11 board.

Another minor change we made is time scaling. Our proposal stated that 1 second in the real world would equal 1 hour in the TamaMudder world. This turned out to be too long to keep the player engaged in the game. Thus we modified the game parameter so that 1 TamaMudder hour equals ½ second in real time. We also switched to toggle switches instead of push-buttons to accept user inputs on the breadboard. This was done so that we could easily tie the input to Vdd or GND. We could have still used the toggle switch if we had added resistors to the outputs.
`

Appendix A – HC11 assembly code

```
*****************************************
* Final2.asm
*****************************************
* Tina Wang
* 11/19/00
* MicroP's Final Project: TamaMudder!
* A game where you take care of a Virtual Mudder.
*
*    This portion is the assembly code that programs the Motorola HC11
* controller. The controller contains the control logic of the
* game. It interfaces with the FPGA board which contains the display
* logic.
*    In theory, each delay time unit should be equivalent to an hour
* but for practical purposes, each delay will only be 1/2 sec. real time.
*
* Interfaces
* - serial output to FPGA for satisfaction display
* - 4-pin parallel input from user keypress for activity input,
*   used 1-hot encoding
* - 4-pin parallel output to LED display for current activity
* - 1-pin parallel output to FPGA for TamaClk, equivalent to slowclk,
*   ticks on and off for every hour in the TamaMudder world (~ 1/2 sec)
*
* HC11 Pin Assignments
* - serial port, serial clk (PD3, PD4)
* - parallel port C, activity input (PC 0-3), TamaClk (PC7)
* - parallel port B, activity LED output (PB 0-3)
*
* Subroutines
* - main2: Calls Level for each catagory and then grade.
* - grade: Increments Life counter and computes & displays grade.
*          Calls win when user has won the game
* - win: Blinks display in an infinite loop
* - sec: Creates a 1/2-sec delay
* - increm: Increments X by 6
* - serial: sends 8-bit output number of data found in 0,X
* - activity: computes duration of an activity, increments level
*             when done
* - level: computes level, drops at appropriate times
* - dead: Clears entire display, infinite loop
*
*****************************************
* Assign Register locations

DDRD  EQU $1009        * Data direction control for Port D
SPCR  EQU $1028        * SPI Control Register
SPSR  EQU $1029        * SPI Status Register
SPDR  EQU $102A        * SPI Data Register
TOF   EQU %10000000    * Timer Overflow Flag mask
TFLG2 EQU $1025
PORTC EQU $1003        * port C, 8-bit bidirectional parallel port
DDRC  EQU $1007        * Data direction control for Port C
PORTB EQU $1004        * port B, 8-bit output port

* Assign Variable locations
LIFE   EQU $0021       * Holds overall life counter
LIFEWK EQU $0022       * Holds life week counter
AFLG   EQU $0023       * Holds internal flag for activity duration
TPORTC EQU $0024       * Temporary storage for Port C input
TAMACLK EQU $0025      * Holds TamaClock counter


*****************************************
*                  Game Level Table ( in Hex )
*                (also Table 1 in Status Report)
*  ==============================================================
*  |  Status:  |   Hunger  | Happiness |    Sleep    |    Tan     |
*  --------------------------------------------------------------
*  | Max Level |    05     |    05     |     05      |     05     |
*  --------------------------------------------------------------
*  |   Level   |    05     |    05     |     05      |     05     |
*  --------------------------------------------------------------
*  |  LCount   |    00     |    00     |     00      |     00     |
*  --------------------------------------------------------------
*  | Time2Drop |    08     |    18     |     18      |     30     |
*  ==============================================================
*  | Activity: |   Platt   | Homework  | Hum Reading | Hike Baldy |
```

```
*  -------------------------------------------------------------
*  | Duration  |    01     |    04     |     08     |     06     |
*  -------------------------------------------------------------
*  | Activity  |    00     |    00     |     00     |     00     |
*  ==============================================================
*
* Max Level: CONST, corresponds to maximum level of health
* Level:     VAR, corresponds to TamaMudder's current level of health
* LCount:    VAR, corresponds to amount of time that TamaMudder has
*               been at current level
* Time2Drop: CONST, except at Level 1, if Drop Time = Time2Drop, then
*               Level = Level-1
* Duration:  CONST, corresponds to the time it takes to complete
*               activity
* Activity:  VAR, counter corresponding the the time spent on an
*               activity
* All CONST remain constant throughout the program. VAR's are
* initiallized to the values in the table.

* Store table into memory starting at $0000, reading down (columns)
       ORG $0000
       FCB #$05          * Column 1 - Hunger
       FCB #$05
       FCB #$00
       FCB #$08
       FCB #$01
       FCB #$00

       FCB #$05          * Column 2 - Happiness
       FCB #$05
       FCB #$00
       FCB #$18
       FCB #$04
       FCB #$00

       FCB #$05          * Column 3 - Sleep
       FCB #$05
       FCB #$00
       FCB #$18
       FCB #$08
       FCB #$00

       FCB #$05          * Column 4 - Tan
       FCB #$05
       FCB #$00
       FCB #$30
       FCB #$06
       FCB #$00

       FCB #$00          * Holds life display

******************************************
* Memory pointer X at location $0000
* Start program at $D000

       ORG $D000
       LDAA #$00         * Initialize Life Counter to 0
       STAA LIFE

       LDAA #$01         * Initialize LifeWeek Counter to 0
       STAA LIFEWK

       LDAA #$00         * Initialize ActivityFlag to 0
       STAA AFLG

       LDAA #$00         * Initialize TemporaryPortC to 0
       STAA TPORTC

       LDAA #$00         * Initialize TamaClk to 0
       STAA TAMACLK
       LDAA #$80         * Set data direction for parallel port
       STAA DDRC         *    1=output, 0=input
       LDAA TAMACLK
       STAA PORTC        * Intialize PortC output bit 7 = 0

       LDAA #$00         * Initialize PortB output to 00
       STAA PORTB

******************************************
* Main program, start by reading parallel port
```

```
main:    LDX #$0000         * Start with X pointing to $0000

         LDAA TPORTC        * Output TemporaryPortC to PortB
         STAA PORTB
         LDAA AFLG
         CMPA #$00
         BNE food           * If ActivityFlag=0 (no activity now)

         LDAA #$80          *   Set data direction for parallel port
         STAA DDRC          *      1=output, 0=input
         LDAA PORTC         *   Load data from parallel port C
         STAA TPORTC        *   Store user input in TemporaryPortC

food:    LDAA TPORTC        * Load data from memory
         ANDA #%00001111    * Mask A to read bottom 4 bits only
         CMPA #%00001000
         BNE hw             * If not food keypress, check next
         JSR activity
         BRA none

hw:      JSR increm
         LDAA TPORTC        * Load data from memory
         ANDA #%00001111    * Mask A to read bottom 4 bits only
         CMPA #%00000100
         BNE read           * If not hw keypress, check next
         JSR activity
         BRA none

read:    JSR increm
         LDAA TPORTC        * Load data from memory
         ANDA #%00001111    * Mask A to read bottom 4 bits only
         CMPA #%00000010
         BNE hike           * If not read keypress, check next
         JSR activity
         BRA none

hike:    JSR increm
         LDAA TPORTC        * Load data from memory
         ANDA #%00001111    * Mask A to read bottom 4 bits only
         CMPA #%00000001
         BNE invalid        * If not hike keypress, then invalid
         JSR activity
         BRA none

invalid: LDAA #$00
         STAA TPORTC

none:    LDAA TAMACLK       * Increment TamaClk
         ADDA #$80
         STAA TAMACLK
         LDAA #$80          * Set data direction for parallel port
         STAA DDRC          *    1=output, 0=input
         LDAA TAMACLK
         STAA PORTC         * Output TamaClk to LEDs

         JSR main2

         LDAA TPORTC        * Output TemporaryPortC to PortB
         STAA PORTB

         JSR sec
         BRA main


******************************************
* Main2 SUBROUTINE
*
* Keeps track of category levels and displays levels to user
*

main2:   LDX #$0000
         JSR level         * Level - Hunger
         JSR increm        * Increment to next category
         JSR level         * Level - Happiness
         JSR increm        * Increment to next category
         JSR level         * Level - Sleep
         JSR increm        * Increment to next category
         JSR level         * Level - Tan
         JSR increm
         JSR grade         * Life status
         RTS
```

```
******************************************

activity: JSR act0
          RTS

******************************************
* Grade subroutine
*
* This increments and displays the life counter

grade:   LDAA LIFE
         LDAB #$A8        * 24 hours x 7 days = 168 hours = A8
         CBA              * If equal to 1 week (if LIFE < A8)
         BNE weekup
         LDAA LIFEWK      *    Week = Week+1
         ADDA #$01
         STAA LIFEWK
         LDAA #$00        *    Life = 0
         STAA LIFE
         BRA grade1

weekup:  LDAA LIFE        * Else Add one to life counter
         ADDA #$01
         STAA LIFE

grade1:  LDAA LIFEWK      * Load current grade
         STAA 0,X         *
         JSR serial       *    Send to display

         LDAB LIFEWK
         LDAA #$05        * If week=5
         CBA
         BEQ win          *    Jump to WIN

         RTS              * Else return from subroutine

******************************************
* Win subroutine
*
* Blinks the display on and off when user has won the game
* Infinite loop until manual reset

win:     LDAA #$00        * Clear activity display
         STAA PORTB

         LDX #$0000       * Blink display when game over
         LDAA #$05        *  infinate loop
         STAA 0,X
         STAA 1,X
         STAA 2,X
         STAA 3,X
         STAA 4,X
         JSR win2

         LDX #$0000
         LDAA #$00
         STAA 0,X
         STAA 1,X
         STAA 2,X
         STAA 3,X
         STAA 4,X
         JSR win2

         BRA win

win2:    LDX #$0000
         JSR serial
         INX
         JSR serial
         INX
         JSR serial
         INX
         JSR serial
         INX
         JSR serial
         JSR sec
         RTS

******************************************
* 1/2-sec delay subroutine
```

```
*
sec:    LDAB #15           * Load B=number of times to repeat for 1 sec.
sdelay: LDAA #TOF          * Clear TOF
        STAA TFLG2
sspin:  TST TFLG2          * test if flag is all 0's
        BPL sspin          * branch on plus, spin until overflow
        DECB               * B=B-1
        BNE sdelay         * if B!=0, repeat
        RTS                * else B=0, return from subroutine

******************************************
* increm
* increment X by 6 subroutine
increm:  INX
         INX
         INX
         INX
         INX
         INX
         RTS

******************************************

level:   JSR level1
         RTS

******************************************
* Serial Port - LED display
*
* Sends data at 0,X to serial port

serial: LDAA #%00111000    * Set data direction for serial port
        STAA DDRD
        LDAA #%01011100    * Set serial port control
        STAA SPCR

        LDAB SPSR          * Check SPSR, will clear SPIF (7th) bit

        LDAB 0,X           * Load current level
        STAB SPDR          * Send data to serial port
        RTS

******************************************
* Activity subroutine
*
* Keeps track of the duration of an activity
* Increments activity counter, compares it to activity duration
* Increments level when counter=duration
* Toggles flag on through the cycle of an activity
* Toggles flag off at the end of an activity
*
* 0,X Max Level
* 1,X Level
* 2,X LCount
* 3,X Time2Drop
* 4,X Duration
* 5,X Activity
*
*

act0:     LDAB 5,X          * Load activity counter
          LDAA 4,X          * Load duration time
          SBA
          BLE act1          * If activity < duration

          LDAA 5,X          *    Activity+1
          ADDA #$01
          STAA 5,X
          LDAA #$01         *    ActivityFlag=1
          STAA AFLG
          RTS               *    and Return

act1:     LDAA #$00         * Else activity >= duration
          STAA 5,X          *    Activity=0
          LDAA #$00         *    ActivityFlag=0
          STAA AFLG
          LDAA #$00         *    LCount=0
          STAA 2,X          *

          LDAB 1,X          *    Load Level
          LDAA 0,X          *    Load Max Level
```

```
         SBA
         BLE act2          *    If Level < Max Level

         LDAA 1,X
         ADDA #$01         *        Level+1
         STAA 1,X
         RTS

act2:    LDAA 0,X          *    Else
         STAA 1,X          *        Level=MaxLevel
         RTS


******************************************
* Leveling Subroutine
*
* Keeps track of the amount of time a user has been at a
* certain level.
* If a user is at Level 1 of any category for 24 hours
*   then TamaMudder dies.
* Level drops when LCount=Time2Drop
* LCount starts counting up from 0 whenever there is a
*   level change
*
* 0,X Max Level
* 1,X Level
* 2,X LCount
* 3,X Time2Drop
* 4,X Duration
* 5,X Activity
*
* If LCount < Time2Drop
*   increment LCount
* Else if Level = 1
*   if LCount < 24
*   increment LCount
*   else DEAD
* Else (LCount >= Time2Drop) and (Level != 1)
*   Level = Level - 1
*   LCount = 0
level1:  LDAB 2,X          * Load LCount into B
         LDAA 3,X          * Load Time2Drop in A
         SBA               * Subtract LCount from Time2Drop
         BLE level2        * If LCount < Time2Drop
         LDAA 2,X          *      Load LCount into A
         ADDA #$01         *      LCount+1
         STAA 2,X          *      Store back into table
         BRA endl          *      Skip to next catagory
level2:  LDAA 1,X          * Load Level into A
         CMPA #$01         * If Level = 1
         BNE level3
         LDAA 2,X          *    If LCount = 24 (hours in a day)
         CMPA #$18
         BEQ dead          *    Then DEAD
         LDAA 2,X
         ADDA #$01         *    Else LCount+1
         STAA 2,X
         BRA endl          *    Skip to next catagory
level3:  LDAA 1,X          * Load Level into A
         SUBA #$01         * Level-1
         STAA 1,X
         LDAA #$00         * LCount=0
         STAA 2,X

endl:    INX
         JSR serial      * Send to display
         DEX
         RTS

******************************************
* Dead subroutine
*
* Blanks out entire display and holds in an infinite loop
* until manual reset.

dead:    LDAA #$00       * Clear display if dead
         STAA 0,X        *   infinate loop
         JSR serial
         LDAA #$00       * Clear activity display
         STAA PORTB
         BRA dead
```

Appendix B-Verilog

Status.v

```
//This module takes serial clock, serial output from the HC11 board
//and outputs LED display control

module status (clk,hc_sin,sloclk,tamaclk,satled0,satled1,satled2,satled3,progled) ;

input clk ;
input hc_sin;
input sloclk;

output tamaclk;
output [4:0]satled0;
output [4:0]satled1;
output [4:0]satled2;
output [4:0]satled3;
output [4:0]progled;

reg [39:0] out1;
reg [4:0] eat;
reg [4:0] sleep;
reg [4:0] hw;
reg [4:0] tan;
reg [4:0] prog;

spconvert sp0(clk,hc_sin,out1);
bindec decode0(out1[34:32],eat);
bindec decode1(out1[26:24],sleep);
bindec decode2(out1[18:16],hw);
bindec decode3(out1[10:8],tan);
bindec decode4(out1[2:0],prog);

assign satled3=eat;
assign satled2=sleep;
assign satled1=hw;
assign satled0=tan;
assign progled=prog;
assign tamaclk=sloclk;

endmodule
```

Bindec.v

```
// This module takes a 3-bit binary input from and turns it into 5-bit
// decimal representation.

module bindec (fromhc,toled) ;

input [2:0] fromhc;
output [4:0]toled;

reg [4:0]toled;

parameter ONE = 3'b001;
parameter TWO = 3'b010;
parameter THREE = 3'b011;
parameter FOUR = 3'b100;
parameter FIVE =  3'b101;

always @(fromhc)
case(fromhc)
 ONE: toled<=5'b00001;
 TWO:   toled<=5'b00011;
 THREE: toled<=5'b00111;
 FOUR: toled<=5'b01111;
 FIVE: toled<=5'b11111;
 default: toled<=5'b00000;
endcase
 endmodule
```

spconvert.v

```
//takes serial data and converts it so that it could be used as parallel data.
module spconvert (clk,sin,pout) ;

input clk ;
input sin;
output [39:0] pout;
reg [39:0] shiftreg;

always @(posedge clk)
 shiftreg<={shiftreg[38:0],sin};

assign pout=shiftreg;

endmodule
```