The Wake-Up-With-Britney-Spears Alarm Clock

Final Project Report
December 9, 1999
E157
Professor Harris

Cody Machler and Tom Heberlein

Abstract

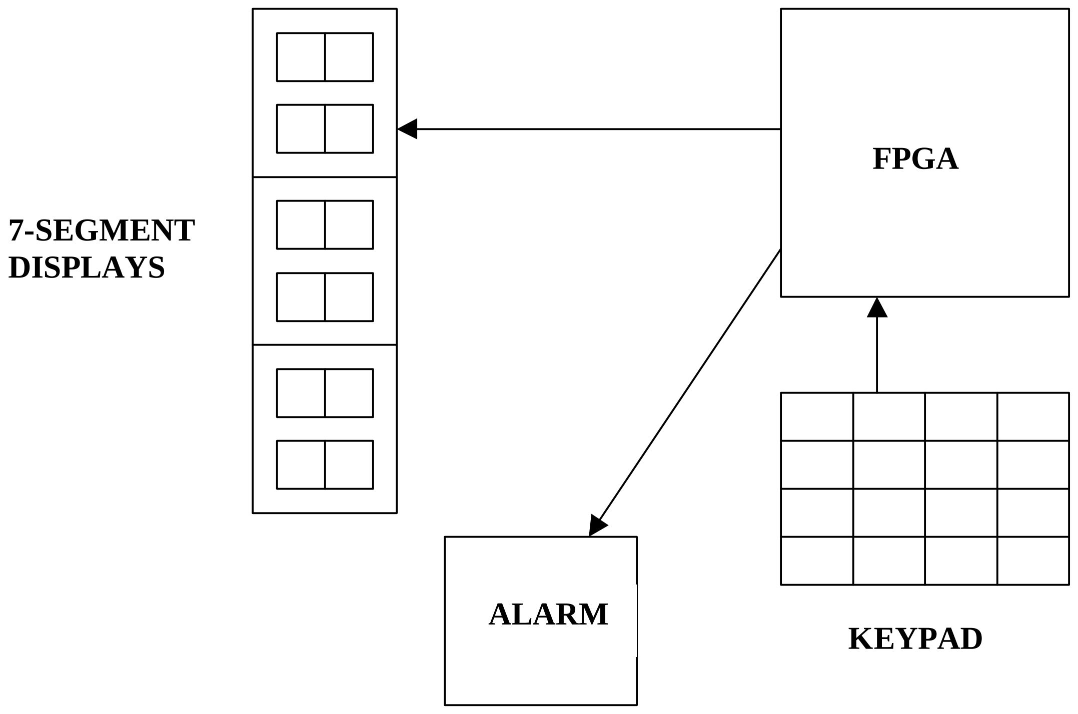
The scope of this project is to build a digital alarm clock, operating on a 24-hour clock display. It will be constructed from the E157 FPGA board, standard resistors and transistors, a 16-button keypad, three E157 7-segment displays, and an alarm, built out of a Britney Spears keychain. All components will be connected through a breadboard. Excluding the alarm, they will also operate on the same power supply. The alarm will have its own battery based power source, supplied with the alarm hardware. All hardware, such as decoders and counters, will be coded for in Verilog and loaded onto the FPGA. The user will be able to set the clock time and alarm time, as well as other check the alarm time and turn the alarm on and off.

Introduction

Most people rely on some model of alarm clock to get their day started. However, whether the alarm is a buzzer, a beeper, or a radio, everybody has most likely experienced sleeping through the alarm. The cause of this problem may be that the alarm is not annoying enough or the user hit the snooze button. For these reasons, we planned to build an alarm clock with a very annoying alarm and no snooze button.

Our digital alarm clock, operating on a 24-hour clock (1:00PM = 13:00) will feature many of the same functions found on standard alarm clocks, such as settable clock time, settable alarm time, and alarm display.

The major hardware components are the FPGA, the keypad, the alarm, and three 7-segment display panels. The overall interaction of the hardware can be seen in the following block diagram.

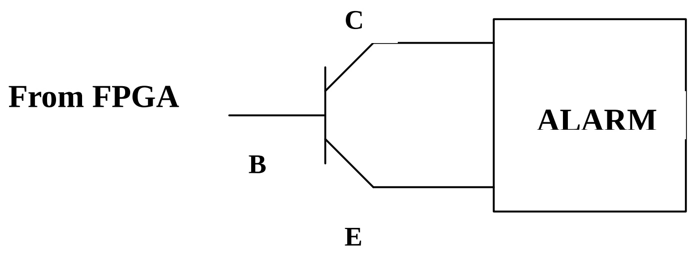


New Hardware

The only hardware implemented in our design that was not used in the E157 labs is the Britney Spears musical key chain that we used for our alarm. The key chain operates on battery power, so it wasn't necessary to draw power from the project's power supply (Three PCS Type L1154 1.5V batteries). The retail key chain can play one of two songs, depending on the position of a switch. To play the selected song, the user pushes a button. To stop the song, the user pushes that same button again.

In order to incorporate Britney into our alarm clock, we had to design a way to bypass the key chain's mechanical switch and use an electrical switch in its place. With the electrical switch in place, a signal from the FPGA could then turn the song on and off. We soldered a wire to each lead of the latching on/off switch. When these wires were connected, the song was activated. After disconnecting them, when they were connected again, the song was deactivated.

To hardwire the alarm to the clock, we attached the leads to a transistor on the breadboard. The transistor base was connected to an FPGA output, while the collector and emitter were connected to the leads from the alarm. The FPGA signal controlled when the transistor opened and closed. Because the alarm is latching, it changes state every time the transistor turns on and completes the circuit. To sound the alarm, we sent our 1 Hz clock signal to the transistor, which turned on and then off. As a result, the alarm sounded. To turn the alarm off, the transistor receives another clock signal.



Schematics

All of our project's components interfaced with each other through the breadboard. The setup is very similar to that of Lab 4. The main difference is the manner in which the displays were wired. We used seven output pins on the FPGA board to drive six digits. We could have used 21, but we chose to use seven to decrease the number of active pins.

FPGA Design

The entire logic is encoded onto the FPGA. The top level module is in alarmclock.v.

This module has the following inputs (note all the pins are summarized in Appendix A):

- clk – This is the 2 MHz oscillator on the FPGA board, hardwired to pin 13. We programmed our system to just use the 1 MHz jumper on it.
- reset – This is the reset button on the board, hardwired to pin 7.
- alarmoff – This is DIP switch 1, hardwired to pin 3. It is actually the alarm on/off switch, but when it is high, we want the alarm off.
- timeset – This is DIP switch 2, hardwired to pin 4. It is used to enable the keypad input as a new time, and display this input on the 7-segment displays.
- alarmset – This is DIP switch 3, hardwired to pin 5. It is used to enable the keypad input as a new alarm time, and display this input on the 7-segment displays.
- alarmcheck – This is DIP switch 4, hardwired to pin 6. It is used to check the alarm time by displaying it to the 7-segment displays.
- rows [3:0] – These are the four wires from the keypad that are used to check which keys are being pressed. They are wired from pins 29, 26, 25, and 27, respectively.

It has the following outputs:

- columns [3:0] – These are the columns being polled to the keypad. They are wired from pins 20, 23, 24, and 28, respectively.
- sevenseg [6:0] – These are the 7 segments of the display, which are wired to all six 7-segment displays, with synchronized transistor base outputs. They are wired from pins 77-83, respectively.
- hourleft – The base driver for the leftmost display transistor. Pin 35.
- hourright – The base driver for the 2nd display from the left. Pin 36.
- minleft – The base driver for the 3rd display from the left. Pin 37.
- minright – The base driver for the 4th display from the left. Pin 38.
- secleft – The base driver for the 5th display from the left. Pin 39.
- secright – The base driver for the rightmost display transistor. Pin 40.
- decimalpoint – The output to H1 on the rightmost 7-segment display. This corresponds to the decimal point on the rightmost display, which we use to indicate the alarm is on. Pin 14.
- secondclk – This is a 1 Hz output, used for debugging purposes. Outputs to LED1, which is hardwired to pin 61.
- alarm – This is the signal that is supposed to drive the alarm. It is wired to LED 6, which is hardwired to pin 68.

The top level module has the following functions:

- It handles inputs and outputs from the board.

- It calls all the other modules

- It decodes the 3-bit transistor signal from the multiplexeddisplays module, into 6 individual signals. (Lines 30-35)

- It determines when to set the alarm off (this *SHOULD* happen when the alarm time equals the clock time, but we didn't get this working). (Line 38)

- It toggles the alarm. Because our alarm uses a latching switch, we need to turn the alarm signal off right after turning it on. This happens as soon as the alarm time no longer equals the clock time. This ensures we get a one-second switch signal. When the user sets alarmoff to high, it should simply asserts the alarm signal again. There may be a problem with this. If the user turns the alarmoff to high, a constant current will flow through the transistor and alarm. If this is too much of a current, the alarm may get hot and die. We didn't experiment with the upper limitations of the time to drive the alarm. (Lines 40-48)

- It controls the outputs to the display and new time registers. It picks which set of numbers should be displayed, and then these numbers are decoded and displayed. Depending on the mode, the display shows the current time, the alarm time, or the input from the keypad. We did not implement the storage of keypad input into the new time registers. We did not figure out how to do this until after the project was due. (Lines 52-108)

All the other modules were relatively simple, and some were modifications of E157 lab modules, so we will not go into depth here. A quick listing follows:

- onesec – gives a 1 Hz signal
- keypad – handles keypad in/outs
- pollkeypad – obtains keypad in/outs
- clkslow – a slow clock for debouncing use
- decode_row_column – decodes keypad in/outs into hex numbers
- multiplexeddisplays – takes 6 decimal inputs and outputs them on 7-segment LED's using only one 7-bit output, and 6 transistor wires (an extension of the multiplexed display)
- decadecounter – counts to ten, starts over
- sixcounter – counts to six, starts over
- hourscounter – counts to 24, starts over

Results

The clock works. We ran the clock for 02 hours: 03 minutes: 00 seconds, with no detectable error. However, we can not set the clock as implemented. This could be rather easily implemented with an enabled set of registers, similar to a multi-bit adder, with carryin/out signals. Unfortunately, we realized this too late. This was the most difficult part of the design, as everything else is either working or almost working. This we did not get coded at all.

The alarm works when triggered. However, we were unable to get the trigger working. We are not sure why this is the case.

The alarm time was not checkable, but this was an error noted in the comments in the code. In our implemented version, we had line 90-96 of the alarmclock module taking illegal inputs. We believe this is fixed in the code, but it is untested.

The alarm time was programmable though.

We changed several things from our original proposal. The first was our alarm. We originally proposed using some kind of D/A conversion, but when we thought about the prospect of waking up with Britney Spears, we could not resist the urge to tear apart one of her products and plug it into an electrical circuit.

The other main thing we changed was our hardware. We originally proposed to use two HC11 boards, and quickly realized we only needed one. When we realized we did not need the HC11 boards at all, we continued trying to figure out how we would use them, but after some frustrating coding attempts, finally decided to get something working on the FPGA boards, since we understand those best.

We also changed the inputs from the user, originally intending to use the DIP switches, changing our minds to use the keypad input (which we said in our presentation), and then changed back to the DIP switches.

Parts List

Part	**Source**	**Vendor Part #**	**Price**
MCD, Music on the Go Keychain (Britney Spears: "…I Will Be There")	WalMart	ST-85001	$10.74

Appendices

Pinout List

```
NET "alarmoff"        LOC =  "P3";
NET "timeset"         LOC =  "P4";
NET "alarmset"        LOC =  "P5";
NET "alarmcheck"      LOC =  "P6";
NET "reset"           LOC =  "P7";
NET "clk"             LOC =  "P13";
NET "decimalpoint"    LOC =  "P14";
NET "columns<3>"      LOC =  "P20";
NET "columns<2>"      LOC =  "P23";
NET "columns<1>"      LOC =  "P24";
NET "rows<1>"         LOC =  "P25";
NET "rows<2>"         LOC =  "P26";
NET "rows<0>"         LOC =  "P27";
NET "columns<0>"      LOC =  "P28";
NET "rows<3>"         LOC =  "P29";
NET "hourleft"        LOC =  "P35";
NET "hourright"       LOC =  "P36";
NET "minleft"         LOC =  "P37";
NET "minright"        LOC =  "P38";
NET "secleft"         LOC =  "P39";
NET "secright"        LOC =  "P40";
NET "secondclk"       LOC =  "P61";
NET "alarm"           LOC =  "P68";
NET "sevenseg<0>"     LOC =  "P77";
NET "sevenseg<1>"     LOC =  "P78";
NET "sevenseg<2>"     LOC =  "P79";
NET "sevenseg<3>"     LOC =  "P80";
NET "sevenseg<4>"     LOC =  "P81";
NET "sevenseg<5>"     LOC =  "P82";
NET "sevenseg<6>"     LOC =  "P83";
```

Code

```
module alarmclock (clk, reset, timeset, alarmset, alarmcheck, alarmoff,
rows, columns, sevenseg, hourleft, hourright, minleft, minright,
secleft, secright, decimalpoint, secondclk, alarm) ;

input clk, reset;
input alarmoff, timeset, alarmset, alarmcheck; // The 4 DIP switch
inputs for the 4 display modes
input [3:0] rows;  // rows sensed in the keypad polling
output [3:0] columns;  // columns polled on the keypad
output [6:0] sevenseg; // 7-segments (a-g)
output hourleft, hourright, minleft, minright, secleft, secright ; //
transistors
output decimalpoint ;          // Represents alarm On
output secondclk ;             // 1 Hz clock to drive HC11 counter (when
it is implemented)
output alarm ;                 // high when alarm sounds

wire secondclk ;        // 1 Hz clock

onesec seconds (reset, clk, secondclk) ;

wire hourleft, hourright, minleft, minright, secleft, secright ;
wire [6:0] sevenseg;
wire [2:0] transistor ; //picks which of 6 transistors is active
wire [3:0] columns;
reg [3:0] hourtens, hourones, mintens, minones, sectens, secones ;  //
4-bit values of display digits
wire [3:0] hourten, hourone, minten, minone, secten, secone ;  // 4-bit
values of counter digits (for clk only)
wire alarmen ; // Alarm Enable
reg alarm ;

reg [3:0] alarmhourten, alarmhourone, alarmminten, alarmminone,
alarmsecten, alarmsecone ; // registers that hold the alarm time
reg [3:0] timehourten, timehourone, timeminten, timeminone, timesecten,
timesecone ;  // the 6 digits of the clock in normal operation
wire [3:0] timesethourten, timesethourone, timesetminten,
timesetminone, timesetsecten, timesetsecone ; //the 6 digits of the
clock in timeset or alarmset operation

assign hourleft = ~(transistor[2] & ~transistor[1] & transistor[0]) ;
// These 6 signals are used for multiplexing the display
assign hourright = ~(transistor[2] & ~transistor[1] & ~transistor[0]) ;
assign minleft  = ~(~transistor[2] & transistor[1] & transistor[0]) ;
assign minright = ~(~transistor[2] & transistor[1] & ~transistor[0]) ;
assign secleft  = ~(~transistor[2] & ~transistor[1] & transistor[0]) ;
assign secright = ~(~transistor[2] & ~transistor[1] & ~transistor[0]) ;

assign decimalpoint = ~alarmoff ;
assign alarmen = ~alarmoff & ((alarmhourten && hourten) & (alarmhourone
&& hourone) & (alarmminten && minten) & (alarmminone && minone) &
(alarmsecten && secten) & (alarmsecone && secone)) ;  // Should alarm
when alarm time == clock time
```

```
always @(alarmen)  // alarm toggling (doesn't work -- unsimulated)
begin          // we used both edges of alarmen, because the alarm is
latching, meaning it turns on/off only on posedge of its switch
      if (alarm)
            alarm <= 1'b0 ;
      else if (alarmoff)
            alarm <= 1'b0 ;
      else
            alarm <= 1'b1 ;
end

// The following handles the clock display.  Hourtens, etc is what is
sent to the decoder for display

always @(posedge clk or posedge reset)
begin
      if (reset)
      begin
            hourtens <= 4'b0000 ;
            hourones <= 4'b0000 ;
             mintens <= 4'b0000 ;
             minones <= 4'b0000 ;
             sectens <= 4'b0000 ;
             secones <= 4'b0000 ;
      end
      else if (timeset)
      begin
//???????????????????????????????????????????????????????????? (How do we
store a time -- figured out, but not implemented, 12/9 -TMH)
            hourtens <= timesethourten ;  // Timesethourten, etc is the
last 6 digits of the keypad input
            hourones <= timesethourone ;
            mintens      <= timesetminten ;
            minones  <= timesetminone ;
            sectens  <= timesetsecten ;
            secones  <= timesetsecone ;
      end
      else if (alarmset)
      begin
            hourtens <= timesethourten ;
            hourones <= timesethourone ;
            mintens      <= timesetminten ;
            minones  <= timesetminone ;
            sectens  <= timesetsecten ;
            secones  <= timesetsecone ;

            alarmhourten <= timesethourten ;  // This stores the keypad
input into the alarm time
            alarmhourone <= timesethourone ;
            alarmminten  <= timesetminten ;
            alarmminone  <= timesetminone ;
            alarmsecten  <= timesetsecten ;
            alarmsecone  <= timesetsecone ;
      end
      else if (alarmcheck)
      begin
```

```
            hourtens <= alarmhourten ; // Changed 12/9/99 to
alarmhourten (this may work for display)
            hourones <= alarmhourone ;
            mintens      <= alarmminten ;
            minones  <= alarmminone ;
            sectens  <= alarmsecten ;
            secones  <= alarmsecone ;
      end
      else
      begin
            hourtens <= hourten ;  // Simple 1 second ticking clock!
            hourones <= hourone ;
            mintens  <= minten  ;
            minones  <= minone  ;
            sectens  <= secten  ;
            secones  <= secone  ;
      end

end

// The keypad module polls the keypad for settable times.  It is always
active, and any check of the outputs displays the 6 digits on displays
keypad getnewtime (reset, clk, rows, columns, timesethourten,
timesethourone, timesetminten, timesetminone, timesetsecten,
timesetsecone) ;
// multiplexeddisplays decodes the 6 digits to a single 7-bit output
and a 3-bit transistor value
multiplexeddisplays hourminsec (clk, reset, hourtens, hourones,
mintens, minones, sectens, secones, transistor, sevenseg) ;

// The following is what we used for our 00:00:00 to 23:59:59 counter.
The bad design of this didn't allow settable inputs
decadecounter seco (secondclk, reset, secone) ;
sixcounter sect (~secone[3], reset, secten) ;
decadecounter mino (~secten[2], reset, minone) ;
sixcounter mint (~minone[3], reset, minten) ;
hourscounter hours (~minten[2], reset, hourone, hourten) ;


endmodule
```

```
module onesec (reset, clk, slowclk) ;

input  reset, clk; //input clock should be 1 MHz
output slowclk ;  // runs at 1 Hz

wire halfsecreset ;  // resets the counter every half-second
reg slowclk ;
reg [18:0] slowclkbits ;  // 19 bits > 500,000 = 7A120

assign halfsecreset = slowclkbits[18] && slowclkbits[17] &&
slowclkbits[16] && slowclkbits[15]&& slowclkbits[13]&& slowclkbits[8]&&
slowclkbits[5];
// Use below for sim, above for implementation
//assign halfsecreset = slowclkbits[5];

always @(posedge clk)
begin
if (reset)
begin
      slowclkbits <= 19'h00000 ;
      slowclk <= 0 ;
end
else
if (halfsecreset)
begin
      slowclkbits <= 19'h00000 ;  // begin count until slowclk toggles
again (half-second)
      slowclk <= ~slowclk ;
end
else
      slowclkbits <= slowclkbits + 1 ;
end

endmodule
```

```
module keypad (reset, clk, rows, columns, display5, display4, display3,
display2, display1, display0) ;

input  reset ;
input  clk ;
input  [3:0] rows ; // row sensor for keypad
output [3:0] columns ; // column polling for keypad
output [3:0] display5, display4, display3, display2, display1, display0
;  // These output the last 6 keypresses

wire [3:0] columns ;
wire slowclk ;  // If this runs slower than 5 ms, but fast enough to
catch any keypress, it debounces!

reg rowlow, previousrowlow ;  // FSM states that establish whether a
key was pressed on the last cycle or not
reg [3:0] newnumber ;  // most recent keypress
reg [3:0] display5, display4, display3, display2, display1, display0 ;

//newdisplay == display0, last display ==> display1, etc

always @(posedge slowclk)
begin
      rowlow = ~&rows ;  // if a key is pressed, a row falls low, and
this stops polling
end

always @(posedge slowclk or posedge reset)
begin
      if (reset)
      begin
            display5 = 4'b1000 ;
            display4 = 4'b1000 ;
            display3 = 4'b1000 ;
            display2 = 4'b1000 ;
            display1 = 4'b1000 ;
            display0 = 4'b1000 ;

            previousrowlow = 0 ;
      end
      else if (rowlow == 1 && previousrowlow == 0 ) // when rowlow goes
high (key is pressed), number shift
      begin
            display5 = display4 ;
            display4 = display3 ;
            display3 = display2 ;
            display2 = display1 ;
            display1 = display0 ;
            display0 = newnumber ;
            previousrowlow = rowlow ;
      end
      else
            previousrowlow = rowlow ;
end

clkslow getnewclock (reset, clk, slowclk) ;  // gets "debouncer" clock
pollkeypad poll (reset, slowclk, rows, columns) ;  // polls the keypad
```

```
decode_row_column RC (reset, slowclk, rows, columns, newnumber) ; //
decodes keypad signals into hex number

endmodule


//////////////////////////////////////////////////////////////////////
//
//////////////////////////////////////////////////////////////////////

module pollkeypad (reset, clk, rows, columns) ;

input  reset ;
input  clk ;
input  [3:0] rows ;
output [3:0] columns ;

wire freeze ;

reg [3:0] columns ;

assign freeze = ~& rows ; // if no debouncing needed, then when a row
falls low, polling is frozen.

always @(posedge clk or posedge reset)
begin
      if (reset)
            columns <= 4'b1110 ;
      else
      if (~freeze)
      begin
            columns[0] <= columns[3] ; // toggles through columns when
no row falls low
            columns[1] <= columns[0] ;
            columns[2] <= columns[1] ;
            columns[3] <= columns[2] ;
      end

end
endmodule
```

```
module clkslow (reset, clk, slowclk) ;

input  reset, clk;
output slowclk ;

reg [13:0] slowclkbits ;  // 14 bits = 8 ms clock (slower than
debouncer need...shouldn't need a debouncer)

assign slowclk = slowclkbits[13] ;

always @(posedge clk or posedge reset)
begin
if (reset)
      slowclkbits = 13'b0000000000000 ;
else
  slowclkbits = slowclkbits + 1 ;

end

endmodule
```

```
module decode_row_column (reset, clk, row, column, number) ;

//  Takes 4-bit row and column inputs, and determines the corresponding
hex number value

input  clk, reset ;
input  [3:0] row, column ;
output [3:0] number ;

reg [3:0] number ;

always @(posedge clk or posedge reset)
  if (reset)
      number = 4'b1000 ;
  else
  if (row == 4'b1110)    // 1, 2, 3, C
    case (column)
      4'b1110:    number = 4'b0001 ;
      4'b1101:    number = 4'b0010 ;
      4'b1011:    number = 4'b0011 ;
      4'b0111:    number = 4'b1100 ;
    endcase             // NOTE:  this still decodes hex values,
although
  else                        // when these are sent to the display
decoder, they are sent blank
  if (row == 4'b1101)    // 4, 5, 6, D
    case (column)
      4'b1110:    number = 4'b0100 ;
      4'b1101:    number = 4'b0101 ;
      4'b1011:    number = 4'b0110 ;
      4'b0111:    number = 4'b1101 ;
    endcase
  else
  if (row == 4'b1011)    // 7, 8, 9, E
    case (column)
      4'b1110:    number = 4'b0111 ;
      4'b1101:    number = 4'b1000 ;
      4'b1011:    number = 4'b1001 ;
      4'b0111:    number = 4'b1110 ;
    endcase
  else
  if (row == 4'b0111)    // A, 0, B, F
    case (column)
      4'b1110:    number = 4'b1010 ;
      4'b1101:    number = 4'b0000 ;
      4'b1011:    number = 4'b1011 ;
      4'b0111:    number = 4'b1111 ;
    endcase

endmodule
```

```
module multiplexeddisplays (clk, reset, hourtens, hourones, mintens,
minones, sectens, secones, transistor, dataout) ;

input  clk, reset;
input  [3:0] hourtens, hourones; // The 6 inputs that drive the
displays
input  [3:0] mintens, minones;
input  [3:0] sectens, secones;
output [2:0] transistor ;        // Picks which of 6 transistors is
active
output [6:0] dataout ;     // The data out from the correctly chosen
decoded data (to the Cathodes).

reg    [3:0] decoderin ;
wire    [2:0] transistor ;
reg    [6:0] dataout ;
reg    [9:0] slowclkbits ;

//                     gfe_dcba  reversed, so bit 6 = F, 0 = A
parameter   BLNK  = 7'b111_1111;
parameter   ZERO  = 7'b100_0000;
parameter   ONE   = 7'b111_1001;
parameter   TWO   = 7'b010_0100;
parameter   THREE = 7'b011_0000;
parameter   FOUR  = 7'b001_1001;
parameter   FIVE  = 7'b001_0010;
parameter   SIX   = 7'b000_0010;
parameter   SEVEN = 7'b111_1000;
parameter   EIGHT = 7'b000_0000;
parameter   NINE  = 7'b001_1000;

wire six ;

assign transistor = slowclkbits[9:7] ;  // toggles at a fast enough
rate that it fires each transistor more than 30 Hz
assign six = transistor[2] & transistor[1] ;  //at 6, goes back to
transistor 0

always @(posedge clk)
begin
      if (reset)
            decoderin <= secones ;
      else if (six)
            decoderin <= secones ;
      else if (transistor == 3'b000)
            decoderin <= secones ;
      else if (transistor == 3'b001)
            decoderin <= sectens ;
      else if (transistor == 3'b010)
            decoderin <= minones ;
      else if (transistor == 3'b011)
            decoderin <= mintens ;
      else if (transistor == 3'b100)
            decoderin <= hourones ;
      else if (transistor == 3'b101)
            decoderin <= hourtens ;
end
```

```
always @(posedge clk)
begin
      if (reset)
            slowclkbits <= 0 ;
      else if (six)
            slowclkbits <= 0 ;
      else
            slowclkbits <= slowclkbits + 1 ;
end

// This is the 7-Seg Display
always @ (decoderin)
      case (decoderin)
            0:          dataout <= ZERO;
            1:          dataout <= ONE;
            2:          dataout <= TWO;
            3:          dataout <= THREE;
            4:          dataout <= FOUR;
            5:          dataout <= FIVE;
            6:          dataout <= SIX;
            7:          dataout <= SEVEN;
            8:          dataout <= EIGHT;
            9:          dataout <= NINE;
            default:    dataout <= BLNK;
      endcase

endmodule
```

```
// The following counters count our time.  They are NOT programmable,
which caused us trouble

module decadecounter(clk, reset, number);

input  clk, reset ;
output [3:0] number ; // counts from 0-9

wire ten ;
reg [3:0] number ;

assign ten = number[3] && number[1] ;  // At ten, resets to zero

always @(posedge clk or posedge reset or posedge ten)
begin
      if (ten)
      begin
            number <= 4'b0000 ;
      end
      else
      if (reset)
      begin
            number <= 4'b0000 ;
      end
      else
      begin
            number <= number + 1 ;
      end
end

endmodule

module sixcounter(clk, reset, number);

input  clk, reset ;
output [3:0] number ; // counts from 0-6

wire six ;
reg [3:0] number ;

assign six = number[2] && number[1] ; // at six, resets to zero

always @(posedge clk or posedge reset or posedge six)
begin
      if (six)
      begin
            number <= 4'b0000 ;
      end
      else
      if (reset)
      begin
            number <= 4'b0000 ;
      end
      else
      begin
            number <= number + 1 ;
      end
```

```
end

endmodule

module hourscounter(clk, reset, onesdigit, tensdigit);

input  clk, reset ;
output [3:0] onesdigit;
output [3:0] tensdigit;

wire ten ;
wire twentyfour ;
reg [3:0] onesdigit ;
reg [3:0] tensdigit ;

assign ten = onesdigit[3] && onesdigit[1] ;  // at ten, adds one to
tensdigit
assign twentyfour = tensdigit[1] && onesdigit[2] ; // at twentyfour,
resets to zero

always @(posedge clk or posedge reset or posedge ten or posedge
twentyfour)
begin
      if (twentyfour)
      begin
            onesdigit <= 4'b0000 ;
            tensdigit <= 4'b0000   ;
      end
      else
      if (ten)
      begin
            onesdigit <= 4'b0000 ;
            tensdigit <= tensdigit + 1 ;
      end
      else
      if (reset)
      begin
            onesdigit <= 4'b0000 ;
            tensdigit <= 4'b0000   ;
      end
      else
      begin
            onesdigit <= onesdigit + 1 ;
      end
end


endmodule
```