A Persistence of Vision Clock Face

Final Project Report
Dec 8, 2000
E155

Mike Aung, Brooke Bassage-Glock, and Matthew Gay

Abstract:

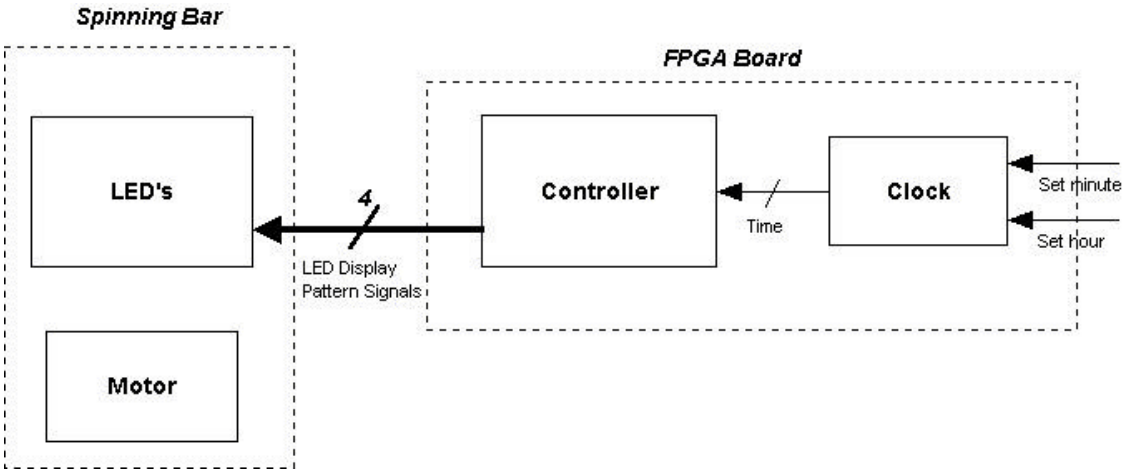
Currently, there are digital persistence of vision clocks on the market. These consist of an inverted pendulum that is driven by springs. There are a few LEDs in a line on the pendulum bar which flash as the bar moves so that there appears to be a digital clock floating in the air. Our project is a prototype of a new style of clock that relies on persistence of vision. However, instead of using a pendulum, we use a bar that spins and outputs an analog clock face. The LEDs are controlled by an FPGA, which also provides the correct time to output.

Introduction

Clocks that rely on persistence of vision are not new products. They can be found in science shops in many malls. However, all of the clocks currently on the market use an inverted pendulum design. A plastic bar is attached to a base at a single pivot point. The user then pulls the bar to one side and releases it, causing it to oscillate back and forth (springs in the base provide the force for oscillation). There are 5 to 10 LEDs arranged in a line on the bar, and as the bar oscillates, the LEDs flash on and off such that there appears to be a digital clock face floating over the base. A good example of a commercially produced clock can be seen at http://www.fantazein.com.

These digital clocks are definitely interesting products, but we wanted out project to be something new and original. So, instead of using an oscillating bar which outputs a digital clock, we decided to use a spinning bar which would then output an analog clock face. Our original goal was to output not only the clock hands, but also the numbers 3, 6, 9, and 12. However, we had some difficulty getting the project to work, so we decided to leave off the numbers, and replace them with tick marks every hour.

There are two main systems which make up the clock. An FPGA is used to control the LEDs, set the time, and keep track of the time. A mechanical system consisting of a motor, a bar, and metal brushes was used to rotate the bar and get the signal from the stationary FPGA to the rotating LEDs. A general block diagram of our project is shown below:

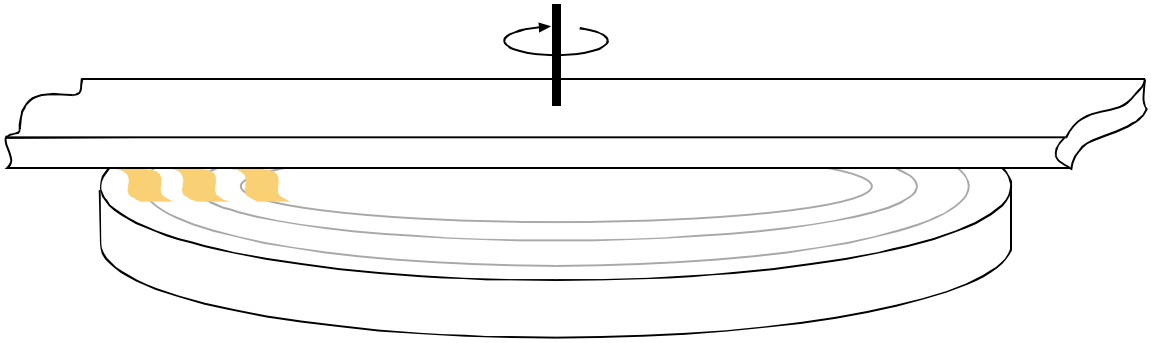


The spinning bar is a long strip of perfboard with the LEDs mounted on it. A motor is used to spin the bar at what we hoped would be a constant rate. The FPGA contains two blocks of logic. First, there is a clock which keeps track of time in hours, minutes, and seconds. There are lines into the clock to set the time, and lines out containing the current time.

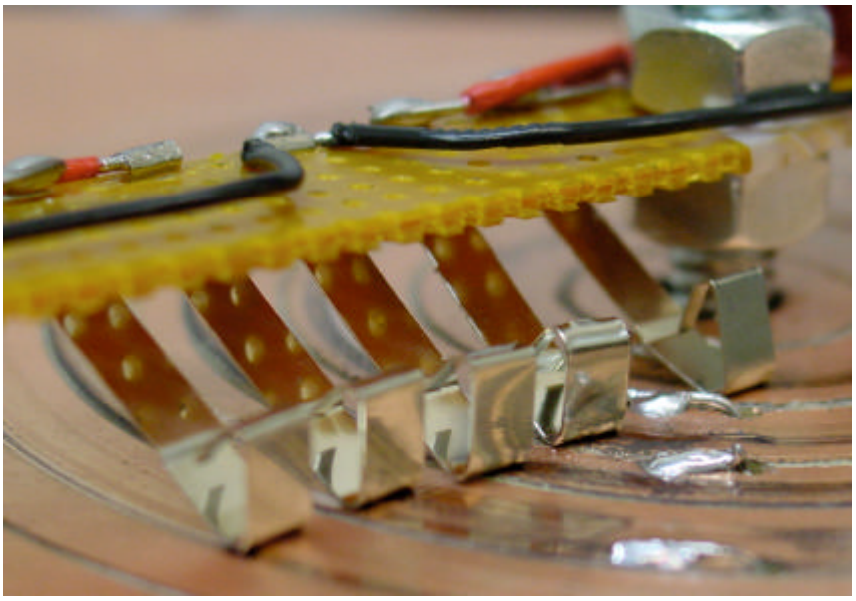
New Hardware

Pick-ups

The most difficult part of this project was designing a rotating bar with LEDs that could be controlled from a stationary point. We started off with a few possibilities for getting signals across a rotating joint. There are commercially available joints able to do this, but they exceeded our budget. Our original suggestion was to use a phone cord detangler. This would give us four lines connected to the board. However, we soon discovered that phone cord detanglers are very high friction devices. After some tests, we decided that the detangler would work, but it would require a very high torque motor. We also considered rotating all of the digital logic so that we would only have to get power across the joint, but this would have prevented us from using the FPGA which was one of the requirements of the project. Our final idea was to use a system similar to the pick-ups used on slot cars. Concentric rings of conducting material were placed under the bar and strips of metal hung down from the bar and stayed in contact with the stationary rings as the bar rotated. A sketch of this system is shown below



Our final idea turned out to be the most successful so that was what we used on the project. A fly cutter was used to cut concentric rings on a PCB so we were left with 5 concentric rings of conducting material. We then drilled a hole through the center of the board for the axle to run through. The pick-ups we used to conduct from the stationary rings to the rotating were slot car brushes. We mounted them at an angle on the bar. Slot car pick-ups were chosen for two reasons. First, they were already in an easy to mount shape, so we didn't have to shape them ourselves which would have made it difficult to mount all the pick ups at the same height. Second, the pick-ups are somewhat springy which means that they are able to stay in contact with the rings all the way around, even when they hit blobs of solder. A close up picture of the pick-up system is shown below:



Motor

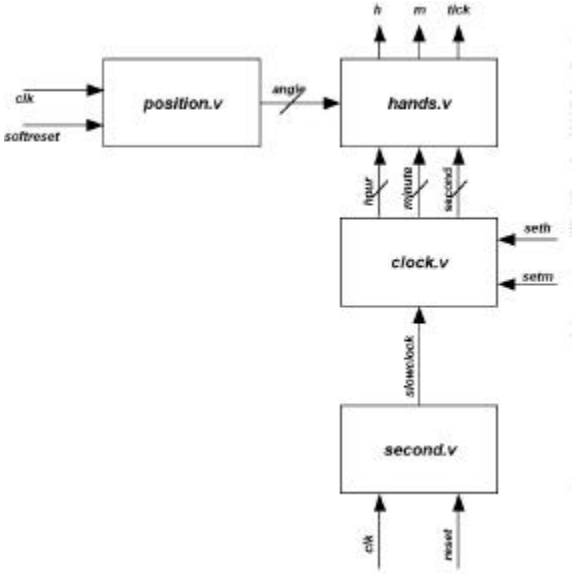
Selecting a motor for this project was not as simple as we originally thought. We needed a motor with the following characteristics: high torque, high RPM, and stable RPM. Our first motor was a floppy drive motor. It had fairly high torque, and reasonably high RPM, and we expected it would be very stable, since it had a built in motor controller. However, after some testing, it became apparent that the floppy drive motor's control system wasn't up to spinning our bar. The motor seemed to have two speeds: fast and slow, unfortunately when it tried to drive our bar, there was enough torque that the controller would jump between the two speeds. Also, there was barely enough torque to overcome the friction between the pick-ups and the rings, so we tried to reduce the pressure between the friction and the rings which resulted in a poor connection. So, overall the floppy drive motor was not successful.

Our next motor was a high speed DC motor from Radio Shack (Cat. #: 273-256). We were somewhat skeptical that it would have enough torque, considering it's low cost of $3.49, but it turned out to be a pretty good motor. We got it to spin at speeds well above 600 RPM (the required speed for our clock to be in phase), and didn't have too much trouble with the torque, although it did sometimes require a little push to get going. There were however two major problems with this motor. First of all, we had difficulty stabilizing the speed of rotation. In fact, we were never able to get a constant speed out f it. This could possibly be fixed with some sort of control system to keep it spinning at constant rate. The second problem was that it tended to overheat and then slow down. It's possible that this was due to the fact that it was having to put out a lot of torque (it was definitely drawing a lot of current), or it could be because the motor was mounted in a block of wood, which didn't allow for much cooling. However, despite these problems, we stuck with this motor because it worked reasonably well, and we didn't have time to find a better solution.

Schematics

FPGA Design

There are two major parts of the FPGA, each of which consists of two modules. The clock modules allow the user to set the time, and keep track of the time. The time is then passed from these modules to the LED control modules. The LED control modules use the time data to flash the LEDs at the appropriate times. A block diagram containing all of the inputs and outputs of the different modules is shown below:



Clock

The clock was fairly simple to design using Verilog. It consisted of two modules, which can be found in the appendix. The first module, Second.v, was a simple clock divider which output a clock with a one second period. This was easy to design since the clock on the board was running at 1 MHz, all this module had to do was invert its output every 500,000 clock cycles. The second module, clock.v, kept track of the time. It took the 1 Hz clock from second.v and used it as the clock for a set of three counters (hours, minutes, and seconds). However, instead of using the standard values for time, it kept track of the position of the hands. This means that if the time is 3:21, it is output as hour=90, minute=126. So, the clock was nothing more than 3 counters. The minute and second counters incremented by 6, and the hour counter incremented by 30. To set the clock, there are two input bits, setm and seth. For normal operation of the clock these bits are kept low. However, if they are set to high, the hours or minutes are incremented once per second, This allows the user to easily set the time by putting the bits to high, and then setting them to low when the desired time is reached.

LED control

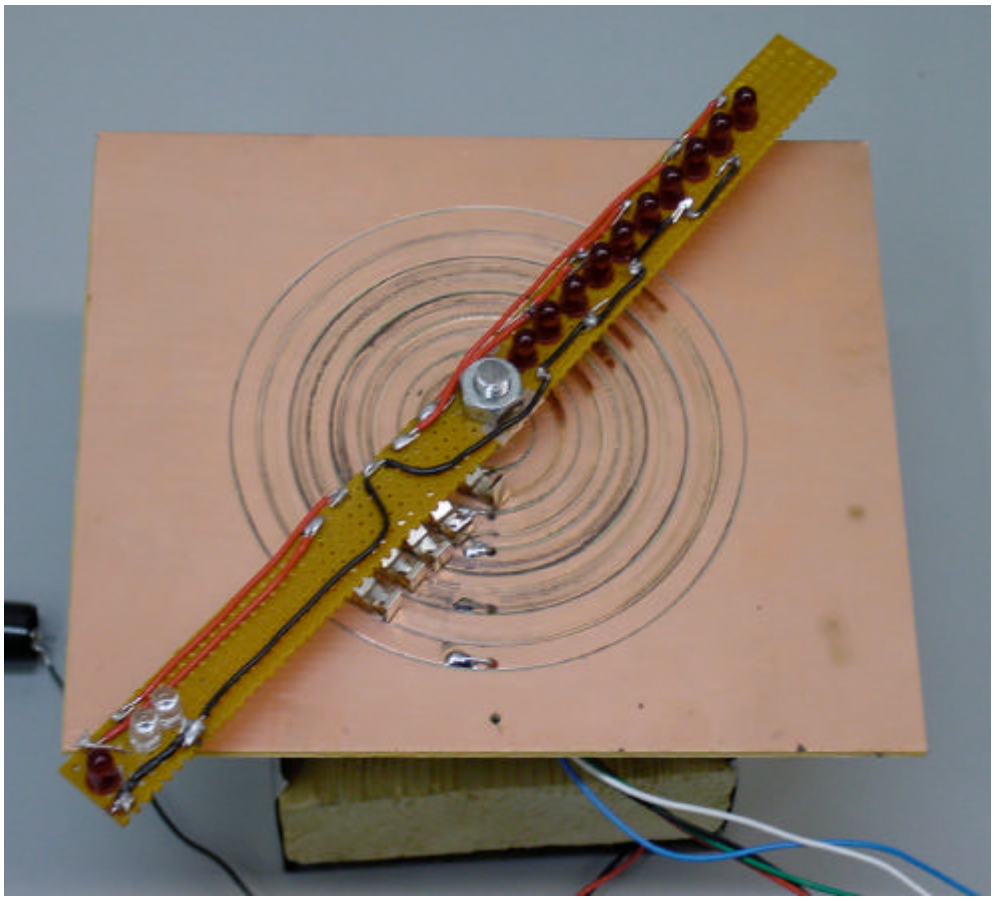
There are two Verilog modules which are used to control when the LEDs flash on. The first module, position.v, keeps track of where the bar is in terms of angle. It is basically a counter that runs from 0 to 360. By adjusting the number of clock cycles

between increments, it is easy to adjust to the speed of the motor. For our project, we assumed that the motor would be spinning at about 10 Hz. There is also a softreset input to position.v, which is used to restart the position counter at 0. This was added in hopes that the motor speed would be stable enough that we could simply run the motor at a speed slightly too slow, and then use a photogate to send a pulse to softreset every time the bar was at noon. This would make it easier to tune the motor speed. Unfortunately, the motor speed was so variable that it was clear that this approach would not be sufficient. The second module, hands.v, is what actually controls the LEDs on the bar. It looks at the angular position output by position.v and compares it to the time output by clock.v. When the two values are the same, it sets h (for the hour hand) or m (for the minute hand).

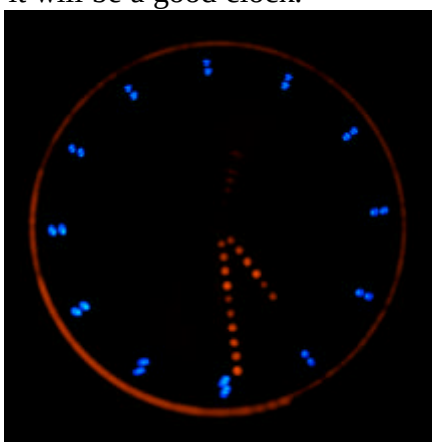
Results

We were never able to get the clock working perfectly, however we were in general pleased with the results of this project. Our final project was able to tell time with significant accuracy, as well as outputting some form of clock face. However, we weren't able to get the motor to spin at a constant rate. As a result the clock face was never perfectly in phase and would instead spin around. We could make the clock face reasonably readable by adjusting the voltage to the motor, but even when we got the motor perfectly in phase with the FPGA, the motor speed would drift. So, our project was somewhat successful, but not completely so.

We have included both a picture of the final design, as well as a movie showing it in action. The first picture shows the stationary design. You can see how the pick-ups contact the track, and in fact leave shiny trails where they have worn down the copper slightly. The design of the base isn't as obvious, but if you look closely at the bottom of the picture, you can see that we used a sheet metal exterior with the motor mounted in a block of wood. The block of wood served two purposes: it provided a secure mount for the motor, and it dampened some of the vibrations.



Next, we have a still shot of the clock, followed by a video of the clock in action. At the beginning of the video, the motor is clearly not tuned, as the hands rotate wildly. However, in the second half of the video, the motor gets more tuned, and you can clearly see the hands. While this movie was being taken the setm bit was high, so the minute hand was moving at a rate of one minute per second. In the movie, it looks like only half of the outer ring of the clock is visible at any time, and there appear to be dark patches. This is an artifact from the camera rather than a real effect. When looking at the clock in person, there are no obvious dark spots, as can be seen on the still image. From this video, it is clear that our clock is nearly fully functional, but that we need to work on the stability of the motor before it will be a good clock.



We had originally thought that we could control for drift in the motor's speed by using the softreset input to position.v. We could then ignore the fact that the motor was changing speed slightly, so noon would always be at the same place. Unfortunately our motor speed was so unstable, that this plan was clearly not going to be successful. This leaves us with two possible alternatives to get this to function correctly. The first is to use some sort of circuit to control the motor so it rotates at a constant speed. This could be done fairly easily with some sort of feedback loop, and would give us more stable motor speed. The other alternative is to not assume that the motor is spinning at a constant speed, and adjust the rate at which the LEDs flash accordingly. Again this could be done with some sort of feedback loop. As a first approximation, we could base the timing on the period of the previous rotation. This idea seems like it will be the most useful to implement, as it will result in a stable clock no matter what the motor speed is. However, this arrangement will result in some drift in the clock face over time, because the LED controller won't be able to adjust instantly, and hence the clock face will rotate over time. To reduce this drift, the softreset can be used, once every hundred cycles or so (possibly more or less depending on the stability of the motor). In addition to some sort of system to get a stable speed, we should probably also either upgrade our motor, or decrease the friction between our pick-ups and the track. With our current set-up, we were unable to get the motor to run for more than about 10 minutes. A higher torque motor would definitely solve this problem, or some sort of better system for conducting across a rotating joint. Changing out the motor and adding in a feedback loop should probably be enough to stabilize the clock face.

We had originally planned to output the numbers 3, 6, 9, and 12 on the clock face. Unfortunately we decided that we weren't going to have time to implement this, and were forced to leave it off of our final design. Adding the numbers requires 7 additional diodes, and 5 additional inputs. This could be done with our current system of tracks and brushes, but it would make for a rather large base. Instead, it would probably be easiest to use some sort of simple encoding to control the LEDS and put a decoder on the bar itself. This would mean that we wouldn't have to add more tracks, and wouldn't be terribly difficult to implement. Another very simple modification would be to add a second hand. This would probably involve nothing more than outputting an "s" bit

similar to the “m” and “h” bits. However, it might be necessary to add another LED on anther circuit so there is no confusion between the minute and second hands.

Parts List

Part	Source	Part #	Price
Motor	Radio Shack	273-256	$3.49
Blue LEDs	Radio Shack	900-7170	$4.19
Perfboard	Radio Shack	276-1395	$2.19
Slot car pick-ups	Pegasus Hobby	Aurora XL400	$2.29

Appendix A – Verilog Code

Hands.v

```
// Matthew Gay

// hands.v outputs a single bit for the hour (h) and minute
hands (m),
// as well as a bit that controls the hourly tick marks
(tick).
// If h=1, then the hour hand is on, otherwise it's off.

module hands(clk,reset,softreset,seth,setm,h,m,tick);

input clk,reset,softreset,seth,setm;
output h,m,tick;

reg h,m,tick;
reg [9:0] hour;
reg [9:0] minute;
reg [9:0] second;
reg [9:0] angle;

clock gettime(clk,reset,hour,minute,second,seth,setm);
position getposition(clk,softreset,angle);

always @ (angle or hour or minute or reset)
 if (reset)
  begin
   h <= 0;
   m <= 0;
   tick <= 0;
  end
 else
  begin
   if ((angle == hour) || (angle == minute)) h <= 1;
   else h <= 0;

   if (angle == minute) m <= 1;
   else m <= 0;

   if (angle == 0) tick <= 1;
   else if (angle == 30) tick <= 1;
   else if (angle == 60) tick <= 1;
   else if (angle == 90) tick <= 1;
   else if (angle == 120) tick <= 1;
```

```
   else if (angle == 150) tick <= 1;
   else if (angle == 180) tick <= 1;
   else if (angle == 210) tick <= 1;
   else if (angle == 240) tick <= 1;
   else if (angle == 270) tick <= 1;
   else if (angle == 300) tick <= 1;
   else if (angle == 330) tick <= 1;
   else tick <= 0;

  end

endmodule
```

Position.v

```
// Matthew Gay

// Position returns the angle of the rotation bar based
// on a 10 Hz angular frequency for the bar's rotation.
// The speed at which the bar is expected to rotated can
// easily be modified by changing the maximum value that
// counter gets. Note that it can be reset once per
revoultion,
// by setting softreset to 1

module position (clk,reset,angle);

input clk,reset;
output [9:0] angle;

reg [9:0] angle;
reg [11:0] counter;

always @ (posedge clk or posedge reset)
  if(reset)
   begin
    counter <= 0;
    angle <= 0;
   end
  else if (counter > 256)
     begin
          counter <= 0;
          if (angle > 358)
               angle <= 0;
          else angle <= angle +1;
     end
```

```
  else counter <= counter +1;

endmodule
```

Clock.v

```
// Matthew Gay

// clock returns the hour, minute and second in terms of
// the angle that the hands make on an analog clock
// (e.g. 2:50:10 = 120:300:60).  It uses the output
// of second.v to keep time.

module clock (clk,reset,hour,minute,second,seth,setm);

input clk,reset;
input seth,setm;
output [9:0] hour;
output [9:0] minute;
output [9:0] second;

wire tick;
reg [9:0] hour;
reg [9:0] minute;
reg [9:0] second;

second secondclk(clk,reset,tick);

always @ (posedge tick or posedge reset)
  if (reset)
   begin
    hour <= 0;
    minute <= 0;
    second <= 0;
   end
  else if (seth)
   if (hour > 300)
    hour <= 0;
   else
    hour <= hour + 30;
  else if (setm)
   if (minute > 348)
    minute <= 0;
   else
    minute <= minute + 6;
  else if (second > 348)
   begin
```

```
    second <= 0;
    if (minute > 348)
     begin
      minute <= 0;
      if (hour > 300)
        hour <= 0;
      else
        hour <= hour + 30;
     end
    else
      minute <= minute + 6;
   end
  else second <= second + 6;

endmodule
```

Second.v

```
// Matthew Gay

// second takes as input a 1MHz clock and outputs
// a 1 Hz clock.  For other clock speeds, you can
// adjust the maximum value of counter

module second(clk,reset,newclk);

input clk;
input reset;
output newclk;


reg [20:0] counter;
reg newclk;

always @ (posedge clk or posedge reset)
  if (reset)
   begin
    counter <= 0;
    newclk <= 0;
   end
  else if (counter < 500000)
    counter <= counter +1;
  else
   begin
    counter <= 0;
    newclk <= ~newclk;
   end
```

```
endmodule
```