

Using a Xilinx FPGA to Beat Your Son at Guitar Hero

Electronically monitor the video signal from a Nintendo Wii console to operate a Guitar Hero game in real time.

by Michael Seedman
Designer
michael@seedman.org

This project started as a way to hang out with my 16-year-old son, Alex. “Dad, Guitar Hero rocks. We’ve got to get one”—simple words that would unknowingly set me on a quest to play that perfect game of Guitar Hero myself. Simple, I thought, because on and off, I’ve had a guitar in my hands for upwards of 45 years. How hard could this be? Surely I was capable of pressing a plastic fret button and a plastic strum button on a plastic guitar.

Well, although my son and I started at the same level, it wasn’t long before he had moved up the difficulty scale and placed his initials on the top of the “high score” display of every song. Try as I might, there was just no way I could get my four fingers to the right place, at the right time, and play those five buttons on that plastic fret board anywhere close to the speed my son was able to do it. A quick search of YouTube confirmed my suspicions. Thousands of kids could beat me in their sleep.

There was only one way I was going to beat Alex at this game—I was going to have to cheat. I was going to have to put my engineering background to work to build my surrogate.

My first thoughts about a system to conquer Guitar Hero on my behalf were simple. Some kind of light sensors stuck to the front of the TV, connected to some kind of microcontroller-based computing platform, driving some kind of solenoids to actuate the physical buttons on the guitar. Rube Goldberg would have been proud.

Many of my projects start as a kludge and get—well, let’s say refined over time. This one was no exception. It didn’t take long for my friend Steve to say, “Just look at the composite video signal and electronically operate the buttons,” quickly followed by another piece of advice: “Use a Xilinx FPGA as the computing engine.” It’s like building a project the old-fashioned way, Steve told me.

“You go into your box of TTL parts and wire them together into any function you need, except you do it on your computer

and you have an almost unlimited number of gates,” he said. “You want a 13-bit shift register, make one. You want a 5-bit adder, make one. Rewiring the project takes about 30 seconds. And everything works at gate speeds—no more counting cycles to see if you can get a task done in a certain period of time.”

Steve, you see, is a video engineer who was responsible for building one of the first nonlinear editing systems in the 1980s. No self-respecting video guy would let anyone build a kludge system like the one I was describing.

After doing some research, I found that not only can you build arbitrarily complex logic, but you could download a microprocessor core with code into the FPGA and get the best of all worlds. Long ago, gates were my crayons. But gates were expensive. They were packaged in their own plastic case with their own leads and in need of external connection, which was provided with a wire-wrap gun and a handful of Kynar-covered wire. It was time-consuming and error-prone. Changing the design was painful. Unwrap, wrap, unwrap, wrap.

This was going to be different. Write some code, compile it, download it to the part and try it out. Need an 11-bit counter? Just write a few lines of code. Problem with the design? Redo it. Want to extend it? Don’t move it from the test bench to the prototype area, just download new code one more time. A full loop takes a couple of minutes.

It’s not like this capability hasn’t been available for years—it’s just that I hadn’t tried it. I’ve been a microprocessor guy. Well, now I’ve tried it and I like it—a lot.

Soon the system design was starting to take shape. Composite video in, a little analog processing and level shifting, an FPGA-based computing engine and drivers for the buttons.

I know, I know—it’s a completely ridiculous project. Why waste time building a computer to play a computer video game? The enormity of commitment of both resources and time was baldly apparent. So as I have done countless times in the past, I simply disguised the exercise as a learning opportunity. Not knowing much about the inner details and intricacies of either com-

posite video or FPGAs, I decided this was my kind of project. Not to mention, I’d get the chance to melt some solder.

What is Guitar Hero?

It’s an insidiously simple game that runs (in this case) on a Nintendo Wii. Guitar Hero has been a huge commercial success. The game comes with a plastic guitar that has



Figure 1 – As the pucks move down the screen they expand, giving the graphic impression of coming toward you. I found scan line 198 was about the right place to look for pucks; notice the green puck passing through the scan. By having five instances of the puck locator running in the FPGA, the computer effortlessly found and stored the presence or absence of a puck. Its “highlight” signal generated the highlighted line.

only five buttons on the neck and a plastic bar on the guitar body that you use to “strum” the instrument. When you start the game, you’re presented with a list of songs. After you select one—say, “Purple Haze”—a vanishing-point guitar neck appears down the center of the screen. Around it swirl animated graphics and a scoreboard.

As you cross a line at the bottom of the guitar neck, the player’s job is to press the correct button or combination of buttons and hit the strum bar. Success gains you points at an ever-increasing multiplier, failure gets you an annoying sound and a multiplier reset. A whammy bar augments your score during sustained notes, and shaking the guitar at certain times during a song increases the point multiplier even further.

It takes consistency to build big scores. Miss enough notes, and the game delivers a notice of failure along with degrading

We could build a system that could potentially turn Dad into a Guitar Hero master—the gaming equivalent to my real-life guitar heroes, Eric Johnson, Jimmy Page and Eddie Van Halen. But I had to surmount a few engineering hurdles first.

boos from the (virtual) crowd. There's nothing like a game of Guitar Hero to keep you humble.

Sizing up the Design Challenge

The Guitar Hero game is, in concept, a simple one. Video is presented on a monitor in two basic sections. By using a DVD recorder I was able to capture, frame by frame, a game being played. Upon review, it was easy to see the step-by-step progress the game makes.

In general, there are two areas of the screen (see Figure 1). Down the center is the guitar neck, starting small at the top of the screen and getting progressively larger at the bottom, presented in just the same way as a vanishing-point drawing exercise you might have done in elementary school. On the left and right, graphics add excitement to the game, but they are extraneous and can be disregarded. Frets start at the top of the neck and travel down the screen, getting longer horizontally as they make their trek.

Overlaid on the neck, and traveling at the same speed as the frets, are colored pucks that look like cream-centered doughnuts, tilted in perspective. Designed to signify necessary button presses, these colored pucks start from the top of the screen as small circles and get wider and wider as they move down, spreading out and expanding in diameter, to give the illusion of moving toward the player.

As each puck approaches a corresponding cylinder at the bottom of the screen, if you're fast enough to press the correct button on the plastic guitar neck and hit the strum bar at just the right moment, it will disappear inside. A small flame emanating from the cylinder signals success. Blow it and the sound of a muted guitar string echoes your failure as the multiplier counter resets. Blow it enough times and you're booted off the stage by an angry crowd.

It seemed to me that by sensing the presence or absence of a puck and timing its progression down the neck of the guitar, we could build a system that could poten-

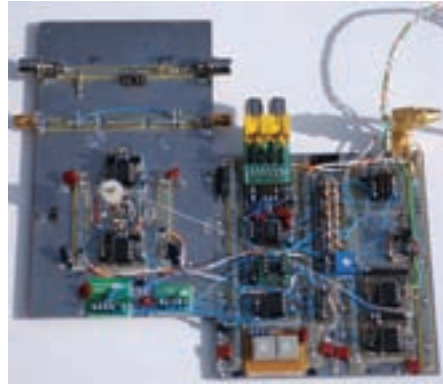


Figure 2 – Here's the prototype of the analog board. It seems that I always make my prototype area just a little too small for the circuit I'm designing. Just as in software, "I'm not quite done, I have to add just one more section..." thus the two boards tacked together. The yellow RCA connectors in the middle of the board are video-in. The video signal moves down the center of the board, through sync separator and Zetex amp, to the row of resistors that shift the 5-V TTL levels to 3.3 V for the FPGA. The small header at the top right corner connects this board to the Digilent board (not shown). The board on the left is the 3.58-MHz color trap and comparator. The long white wire going from the bottom right to the middle left is video.

tially turn Dad into a Guitar Hero master—the gaming equivalent to my real-life guitar heroes, Eric Johnson, Jimmy Page and Eddie Van Halen. But I had to surmount a few engineering hurdles first.

Challenge No. 1: There's no still spot on the screen you can look at to gather the important information. Guitar Hero isn't like yesterday's Pac-Man or Pong. Graphics are in motion all over the place. Overlaid frames of moving frets, lightning bolts, flashes of fire and strobes may make for exciting game play, but they were going to make finding those pucks difficult.

I guess this would normally be the job for a frame store and a computer running complex algorithms, but buying or building a frame store was slightly out of the question. Finding those pucks was going to be the job of a single-pixel comparator. Yes, we could pick the line, and yes, we could pick the position on that line, but the only information available was going to be "is the signal above or below this value?" White, or less than white?

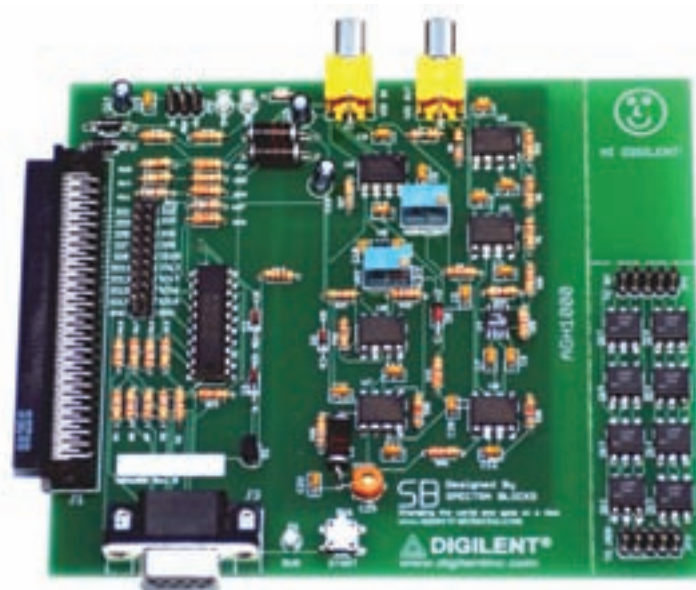


Figure 2A – This is the completed PCB that my friend Steve laid out for the Digilent project. The connector on the left now plugs directly into the Digilent Nexys2 FPGA Demo board. Counterclockwise from there are the DB9 connector that drives the guitar, the optoisolator subassembly and the two video connectors—one for in and one for out. I'd say this board is a little cleaner than the prototype shown in Figure 2.

Finding a puck on the screen was like trying to find a doughnut on a conveyor belt by looking through a pinhole. By choosing an imaginary line right above the flame tips that jump from the cylinders at the bottom of the screen, we found a relatively quiet spot from which to watch for the white centers of the pucks.

The only input to the system is a single, 1-volt peak-to-peak composite video signal containing all the information necessary to drive a video monitor. By separating the composite video into horizontal and vertical sync, I could control counters in the computing platform to locate any spot on the screen. Programmatically, I could say “look at line 198, position 1340, and tell me if it’s white or dark.”

Composite video in, comparator out at a single spot (see Figure 2). I used a National LM1881 sync separator and a Zetex ZXFV089 video amplifier with dc restore to generate the timing signals for the computing platform. Between these two parts, I generated horizontal and vertical sync and a referenced copy of the original video signal.

From here, the video signal takes two paths. Path one is through an op-amp-based adder, so that I could selectively add a small offset voltage before the video passed through an amplifier/buffer and out to a monitor. By using the computing platform to signal “highlight,” I could make areas of the screen lighter than normal. This came in handy when I was troubleshooting the system, since it let me highlight, for instance, “areas where the comparator sees high values” or “this is where I’m sampling the screen.”

Path two is to a 3.58-MHz trap to remove the color information from the signal. Color information rides on top of luminance information, and all I wanted to do was look at the brightness of a spot on the screen. If I left the color information in the signal, the next stage would have had a very difficult time dealing with it. Next, it was on to a very fast (7-nanosecond) comparator to generate the “white/not white” signal.

Challenge No. 2: Controlling the plastic guitar. There are five buttons on the guitar’s neck and a strum bar in its body. Each of

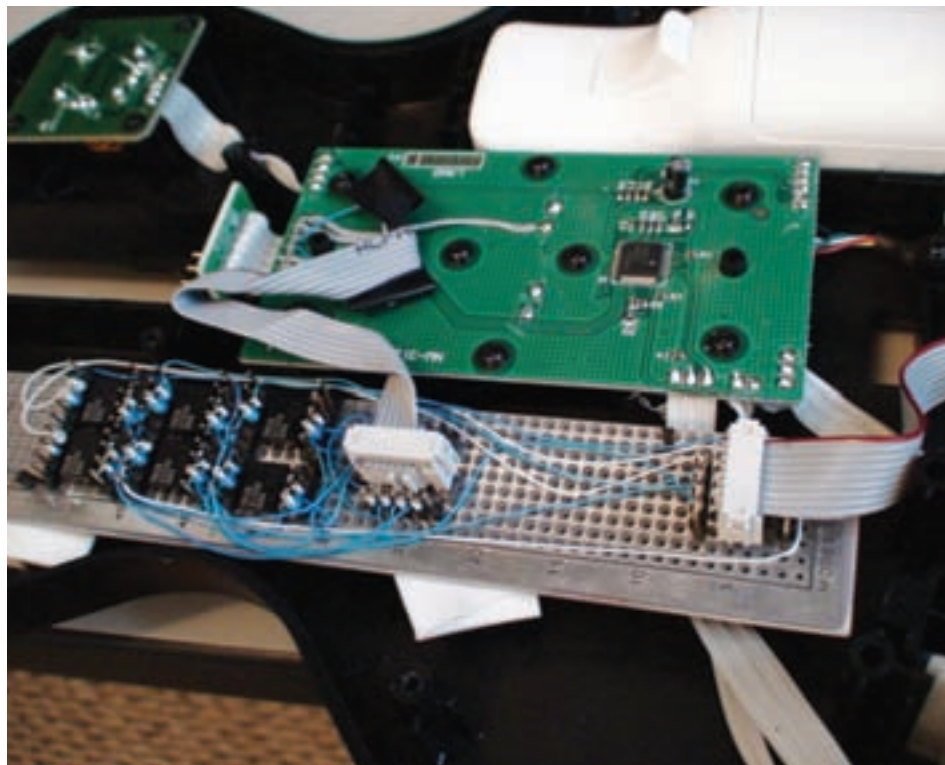


Figure 3 – The smaller green board is the guitar controller. Below it is the prototype board with the six optoisolators, used to electrically press the buttons on the guitar. The ribbon cable on the right goes to a DB9 connector mounted on the side of the guitar.

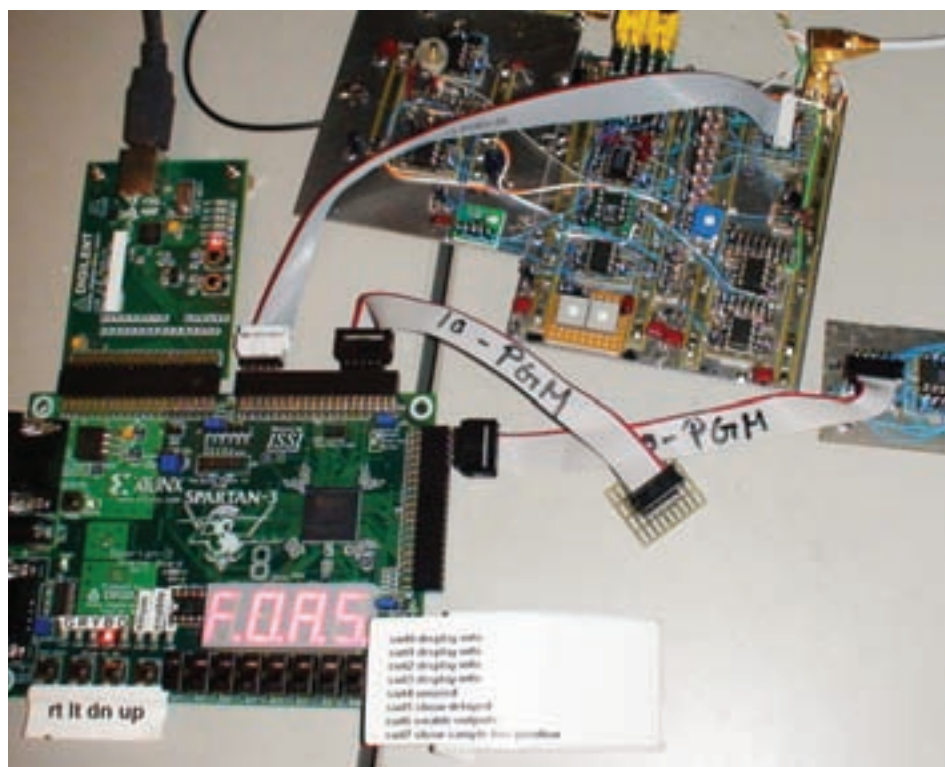


Figure 4 – Here’s the entire AutoGuitarHero system. Counterclockwise from the left, boards include the Digilent USB interface, Digilent Spartan-3 FPGA demo board, driver board and analog interface board. The small board in the middle of the picture is used to attach scope probes to the system for troubleshooting. Notice the yellow video-in connector and the SMA video-out connector.

Now that I had a working hardware platform, it was time to write some code. I downloaded and installed ISE from the Xilinx site and got the USB interface up and running so I could program the Spartan-3.

those six controls had to be operated under computer control.

I disassembled the guitar body and poked around until I figured out which pads on the circuit board were driven by the various buttons. Six bits from the computing platform drove an LS244 with current-limiting resistors on the outputs. Optoisolators (see Figure 3) mounted on a small circuit board inside the guitar body drove each of the button pads. The computing platform and guitar body are connected together through a nine-conductor RS232 cable.

Challenge No. 3: Having the analog section and I/O prototyped and debugged, my thoughts turned to picking the right computing platform. Since I was an FPGA novice, it had to be reasonably simple, really fast, really powerful, have a stable programming environment and should be something I could grow into and learn from. That's when I found Digilent.

Digilent is a great company, and they offer a Spartan-3®-based FPGA demo board that I thought would be perfect for the application. In fact, there are so many gates and so much I/O that I knew the board would be overkill—but as with all computing platforms today, you get a lot of bang for the buck.

Challenge No. 4: A small matter of programming the system. Now that I had a working hardware platform, it was time to write some code (Figure 4). I downloaded and installed ISE® from the Xilinx site and got the USB interface up and running so I could program the Spartan-3. There's a pretty steep learning curve before you can do anything useful, but once you get your head around the idea of designing with gates, it's incredibly powerful.

In the microprocessor world, everything happens sequentially. Set up this variable. Do this. Do this next. Then do this. If this happens, do something else. It's systematic and progressive, and timing is important. "How long does it take to go through this loop?"

The world of programmable logic is different. Everything happens in parallel—and at almost wire speed. No counting clock cycles. Everything just occurs at the same time. If you need a 13-bit counter, you write a line of code. If you need a six-input gate, you write a line of code. It's fantastic, like reaching into a magical junk box and pulling out the perfect part. Need to build a divide-by-50 million counter? Instead of wire-wrapping pin after pin to make one, just write a line of code. Didn't get it right? No need for an unwrap tool—just rewrite the line.

For instance, here's the VHDL code to set up two counters:

```
signal line_cnt:std_logic_vector(8
downto 0);9 bit counter 0 to 511
signal samp_cnt:std_logic_vector(12
downto 0);13 bit counter 0 to 8191
```

Now, take a look at the VHDL code to count lines. It counts from 0 to 262 and then resets to zero each time vertical sync gets asserted:

```
process(v_sync, burst, line_cnt)
begin
  if burst'event and burst = '0'then
    if v_sync = '0' then
      line_cnt <= "000000000";
    else
      line_cnt <= line_cnt + 1;
    end if;
  end if;
end process;
```

Similarly, here's the counter for position in a line. It counts from 0 to about 3,054 using the on-board 50-MHz clock, and then resets to zero at the beginning of each line:

```
process(clk, samp_cnt, burst)
begin
  if clk'event and clk = '1' then
    if burst = '0' then
```

```
      samp_cnt <= "0000000000000";
    else
      samp_cnt <= samp_cnt + 1;
    end if;
  end if;
end process;
```

Everything in the design is driven off these two counters. We need the program to watch five spots on the screen for a puck. We ask the code to generate five instances of the following code:

```
process(clk)
begin
  if clk'event and clk = '1' then
    if v_sync = '1' then
      if samp_cnt = puck_center_y then
        if line_cnt = sense_line and
window_in = '1' then
          yellow_frame_latched <= 1;
        end if;
      end if;
    end if;
  end if;
end process;
```

"If it's the rising edge of clk (every 20 ns) and we're not in reset, and if we're in the right position (samp_cnt) and we're on the right line (line_cnt), and if the comparator (window_in) has found a white spot, then there's a puck in the right position on the screen. So store the event (in this case, yellow_frame_latched <= 1)."

The actual algorithm is a bit more complicated, because a fret moving over that area will also meet the above criteria. Thus, the code watches five lines in a row and increments a counter for each line the comparator finds to be white. Then, later in the program, it checks to see the value of the counter. If just a fret passed the spot, there is a count of less than 3. If a puck passes the spot, the counter contains a value of 3 or greater.

At the end of every frame we store the results of the puck hunt in a shift register, clocked every frame (about 1/30th of a second). Notice how you can just copy this code four more times and have a six-deep x1 shift register running to store the puck information in time so we can offset the actual strum button press x/30th second to wait for the puck to be in proper position on the screen.

```
process(v_sync)
begin
  if v_sync'event and v_sync = '0'
  then
    for i in 0 to SR_SIZE - 2 loop
      shift_reg_y(i+1) <=
shift_reg_y(i);
    end loop;

    shift_reg_y(0) <=
yellow_frame_latched;
  end if;
end process;
```

Finally, we use the shift register tap 5 to see if we have to activate the strum bar. If there are pucks in this tap of the shift register, ping the strum bar. If not, check next time.

```
process(clk, strum_stretched_srps_g,
strum_stretched_srps_r,
strum_stretched_srps_y,
strum_stretched_srps_b,
strum_stretched_srps_o)
begin
  if clk'event and clk = '1'then
    strum_center <=
strum_stretched_srps_g
      OR strum_stretched_srps_r
      OR
strum_stretched_srps_y
      OR strum_stretched_srps_b
      OR strum_stretched_srps_o;
  end if;
end process;
```

Although the actual code is about 1,000 lines, the code I showed above does all the heavy lifting. There are some pulse stretchers and timing helpers in there too, but this is the core.

Son of Guitar Hero

I worked on this project on and off, in my spare time, for about three months. Each step was a learning experience. Before Guitar Hero, I had little hands-on knowledge of composite video, FPGAs, VHDL or much else this project took to complete. Afterwards, I can say that I know my way around Xilinx's IDE, and I've learned how unbelievably powerful an FPGA can be. It's a great environment and one that I plan on using in the future.

In fact, after I posted the *autoguitarhero.com* Web site, the folks at Digilent noticed a marked increase in traffic to their site. They found out about the AutoGuitarHero project and wanted to use it in a demo. I agreed to supply them with a system they could show in a booth at an engineering show, and asked to be paid with store credit at Digilent. How's that for commitment?



Figure 5 – Despite my best efforts, it didn't take long for my son, Alex, to trump my best score, as seen in the telltale on-screen leader board, by using a little bit of (uncomputerized) body English.

At this point, the AutoGuitarHero has been taken apart and the various subassemblies are sitting in a box. Its work is done. I put my initials at the top of the leader board and they stayed there for almost a full day. For you see, I didn't computerize the whammy bar or the guitar shake—and by operating the whammy bar and shaking the guitar correctly, you can earn more points.

My wife and I went to dinner the evening that I completed the project and my son went up to my shop. He turned on the AutoGuitarHero, and operated the whammy bar and shook that guitar better than I ever could. When I got home, I found his initials, not mine, at the top of the screen (Figure 5). What's a dad to do?

For the complete blow-by-blow of how Michael Seedman built this project, visit his site www.autoguitarhero.com.

One Board to Rule them All!

X5 GSPS

Lord of RF Signal Capture

Features

- Two 1.5 GSPS, 8-bit A/Ds (Nat ADC08D1500)
- +/-1V, 50 ohm, SMA Inputs
- Xilinx Virtex5, SX95T FPGA
- 512 MB DDR2 DRAM
- 4MB QDR-II SRAM
- 8 Rocket IO Private Links, 2.5 Gbps each
- >1 GB/s, 8-lane PCI Express Host Interface
- Power Management Features
- XMC Module (75x150 mm)
- PCI Express (VITA 42.3)

Perfect for

- Wireless Receiver
- WLAN, WCDMA, WIMAX front end
- RADAR
- Electronic Warfare
- Electronic Counter Measures (ECM)
- High Speed Data Recording
- Electronic Surveillance
- Spectral Analysis
- IP Development

ip cores

Innovative Integration
... real time solutions!

805-578-4260 phone
www.innovative-dsp.com