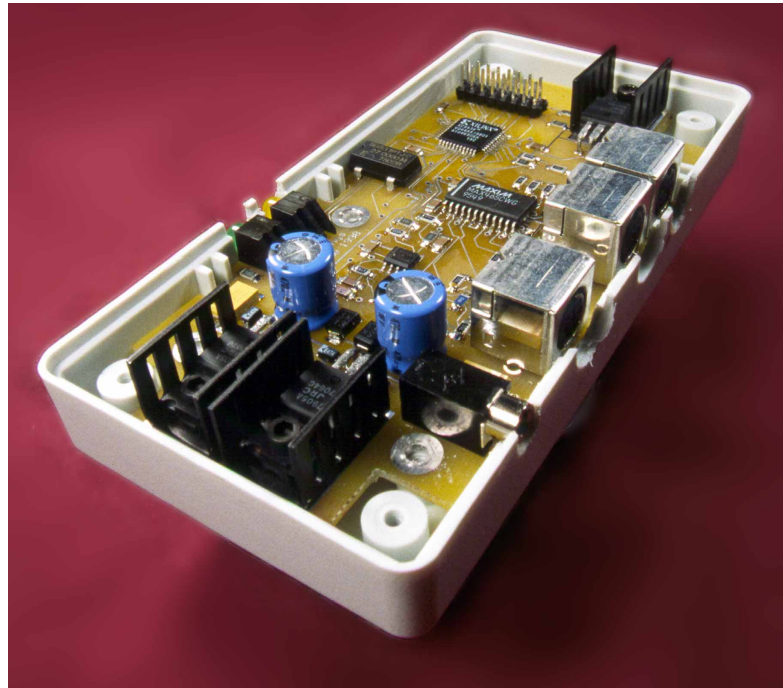


Video Switch

Cullen Jennings

FIGURE 1. Completed Project



In a large computer vision application I am working on, I am using multiple video cameras to grab stereo pairs of images. To do this I needed a video multiplexer that takes inputs from multiple cameras, detects the video sync signals, and switches between the multiple inputs on each field. I have constructed such a video multiplexer: it accepts 6 inputs and switches them to 3 outputs. It does sync detection and uses a Xilinx 9500 series PLD to implement the switching logic. This PLD is “in system programmable” so it can be changed to implement various switching schemes such as alternating between camera inputs every three seconds, every frame, on each field, or on a certain area of the image. This last scheme would allow a banner from one video input to be superimposed on top of another video image - and various other picture in picture type applications are also possible. Although this video multiplexer is primarily designed for NTSC video applications, the bandwidth is adequate for switching video signals for high resolution monitors. The switch also implements a simple DAC, which is used to output a video

Concept

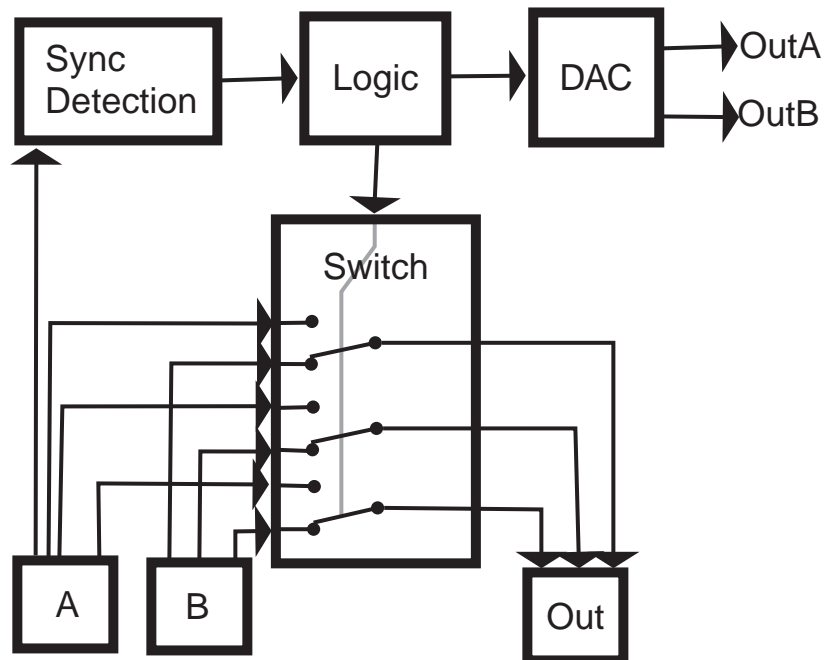
test signal that produces an image with a few simple gray bars. This is a useful test image and demonstrates the simplicity of synthesizing a video signal using a relatively inexpensive PLD.

This project has been interesting from several points of view. The programming of the PLD is all done in VHDL, which is very powerful and is becoming much more available to engineers with small budgets. The project is constructed using surface mount components - I will address some of the issues that arise when working with these using only the type of equipment one might have at home. The PCB is only two layers, which keeps the price down. Video cameras have become so inexpensive lately that I am sure their usage will increase in both commercial and noncommercial projects. This circuit is a useful tool for any project involving multiple video cameras. It is cheap (approximately \$50), the components are widely available, and it is not particularly difficult to build.

Concept

The block diagram for the circuit is shown in Figure 2. There are three video outputs, which can choose either three A or three B inputs. The logic block selects the inputs to use and receives information from the sync detection circuit. The logic block also synthesizes two output video signals that can be used as sync channels or other video test signals.

FIGURE 2. Block diagram for video switch



I connect the A and B channels to two RGB video cameras that are genlocked together, and connect the output to a video digitizer card in a computer. The PLD is programmed to switch on each field. When I capture a frame, the even field is from one camera and the odd field is from the other. In software I separate these into two images and process them through a stereo correlation algorithm to compute the distance from the cameras to the objects viewed.

Another application would be to take a certain region of the screen from input A part of the time and from input B the rest of the time. This would allow a banner or rectangle from the image A to be overlaid on the image B. The system could also be programmed to switch inputs every few seconds to allow viewing through two cameras or to put the view from the first camera in the top half of the image and from the second in the bottom half so that two cameras could be watched at once.

The ability to generate test patterns can be used to time stamp the video. Basically you generate a test pattern that encodes the frame number as a binary pattern on the image and then you use the switcher to overlay this image onto the left edge of the input video. This is really convenient for finding out exactly how many frames your video capture card is really dropping and whether it ever duplicates frames.

A slight modification of the circuit is required (diode and two resistors) to feed the AC signal into one of the inputs of the PLD so that it can synchronize the output sync signals with the AC line signals. Video cameras can then be genlocked to this input signal, and the video camera captures will synchronize with the flicker of the AC fluorescent lights. This will significantly improve the image in some cases.

Something I have not looked into but am curious about is whether one could regenerate portions of the sync signal that had been corrupted by, say, a video copy protection scheme. It seems like it would be easy just to switch over to the synthesized sync on those scan lines having a corrupted sync signal.

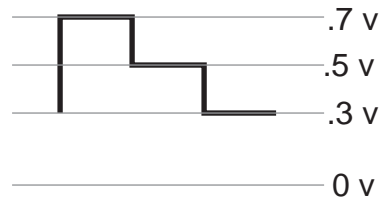
The project is also quite convenient if your scope does not have a TV trigger. Feed a video signal into this device, and then use one of the digital outputs from the PLD as a trigger for your scope. When it comes to video, it slices, it dices; but wait, there's more - it comes with a free TV trigger for your scope.

Brief review of RS170 video signals

A video sequence is a series of still images called frames. Each frame consists of 525 horizontal lines called scan lines. The frame is split into two fields with the odd scan lines in the first field and the even in the second. The video signal transmits one frame after the next. For each frame, the first field is transmitted followed by the second. The fields are transmitted by transmitting each of the scan lines with appropriate delays and signals between each scan line, field, and frame so that the receiving system can tell what part of the image is being received. The grayscale value of the image at a given location along the scan line is transmitted by encoding it as a voltage between 0.357 and 1.0v. Consider the example shown in Figure 3. At the start of the scan line, the image is

white so the output voltage is 0.7v. As we move across the image, we hit the gray region where the output voltage drops to 0.5v, then the black where it drops to 0.3v. The scan lines occur at a rate of 15,734Hz or about every 63.5555 μ S.

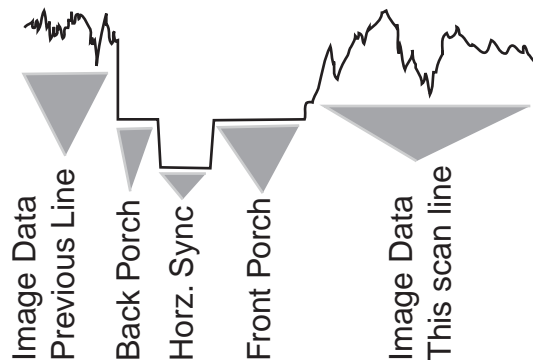
FIGURE 3. Video signal for a single scan line.



HORIZONTAL SYNC

At the start of each scan line, a horizontal sync is transmitted so that the system will know where the scan line starts. This signal is shown in Figure 4. Here we have a back porch section that is 1.18 μ S long at 0.306v, followed by the actual sync pulse that is 4.7 μ S long at 0.020v. Finally there is the front porch for 3.14 μ S at 0.306v.

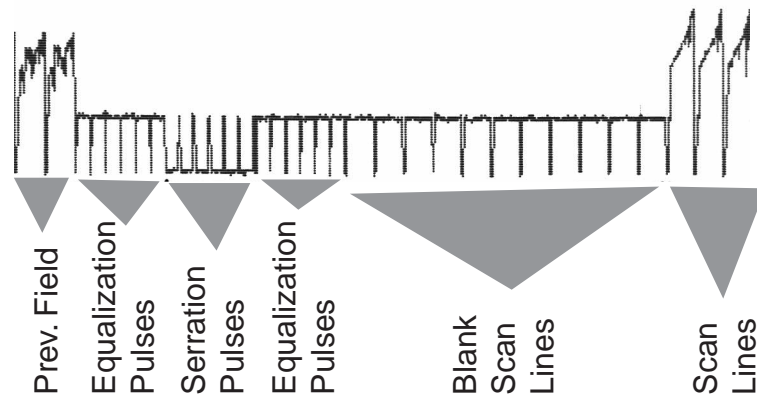
FIGURE 4. Horizontal sync signal between two scan lines.



VERTICAL SYNC

In addition, the first several scan lines in each field are not used for image data but are used for vertical sync and other purposes. The first three scan lines of the field contain equalizing pulses, followed by three lines of serration pulses, then three more lines of equalizing pulses, as shown in Figure 5. The next eleven lines generally contain black scan lines but might have other information such as closed caption. The equalizing pulses are 2.3 μ S wide and the serration pulses are 4.7 μ S wide.

FIGURE 5. Vertical sync between two fields.



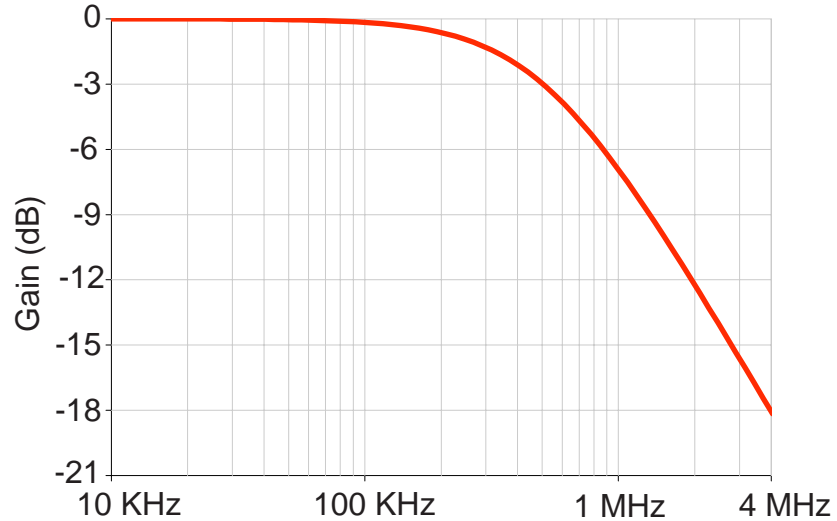
PAL and SECAM are very similar, but the timings change. This is described in detail in *Video Demystified* [1].

Schematics

The video output by the system can be selected from either the A or B video input. The logic block selects whether to use the A or B input. It can use the information provided by the sync detection circuit to decide when to switch video inputs. The logic block also drives a simple DAC that can be used to generate a video output signal on the two sync out lines. The schematics are shown in Figure 17.

All video signals coming into the system are terminated with a 75 ohm resistor. Depending on the jumper settings of JP1 and JP2, either the green A signal or the sync A is fed into the sync detection circuit. First the signal is filtered in the passive RC circuit formed by R3, C7 and C9. The filtered signal is then passed into the U5 chip. U5 is a National LM1881 chip [2] that does the sync detection. The frequency response curve for this filter is shown in Figure 6. This was computed using Electronics Workbench [3], which forms a convenient front end to SPICE. It has about an -18db attenuation of the color burst signals but very little effect on low frequency sync signals. This filter is only required if color signals are being fed into the system. It can be removed from the system by simply removing C7 and C9. There is really no need to short R3 because it has little effect relative to the high input impedance of U5.

FIGURE 6. Calculated filter response. This will have a nearly -18 dB effect on the color burst signal while having little effect on the sync signal which is mostly under 500KHz.



The various sync signals are fed into the PLD (U8 in the schematic), which can be programmed to select the correct input - more on this later. The PLD is a Xilinx 9536 which is discussed in the Xilinx *Data Book* [4] and Xilinx web page [5]. The PLD also receives a clock signal from the oscillator and controls two LEDs. JP6 allows the logic section to be easily connected up to another circuit or to a logic analyzer. Data lines 0 to 7 can be used as general purpose I/O. Data lines 9 and 10 are pulled to certain logic levels with the resistors R22 and R23. They can be pulled to the opposite levels by putting jumpers on JP6. Digital power is also available on the jumper to power some other circuit, if desired.

The PLD controls U6, the video switch chip. U6 is a Maxim MAX465 [6] video switch which controls three channels and can select each channel from one of two inputs. It amplifies the signal by a factor of two, which conveniently allows us to divide the signal by two by running it through a 75 ohm resistor before driving the 75 ohm transmission line. This makes it easy to match impedance and to avoid overdriving the output amplifier even if the output line is shorted to ground - something that seems to happen when I build cables. It also double terminates the transmission line, greatly reducing reflections.

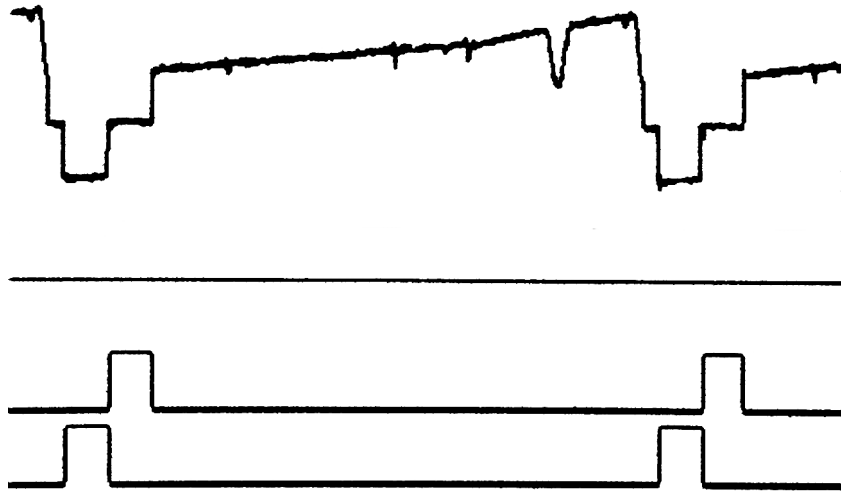
The logic block also outputs a two-bit signal into the DAC built by a two resistor network. Resistors R9 and R13 form the DAC for the B input sync while R19 and R20 form the DAC for the output sync. Since the outputs on the PLD can be set to 0, 1, or Z (tri-stated), there are six possible output voltages from this DAC.

The power supply circuitry was designed to keep the noise on the power to the analog chips low. I considered using a switching supply because of concerns about heat dissipation, but I decided the linear power regulator would cause less noise on the video output.

The digital circuits have their own supply because this was a simple way to reduce transient noise on the analog lines caused by high speed digital switching. When substituting capacitors or transformers on the input side of the power regulators, keep in mind that a 12v AC transformer means 12 RMS, resulting in a DC voltage of 17v. If the voltage rating of your capacitor is too low, it will fail much sooner than it should.

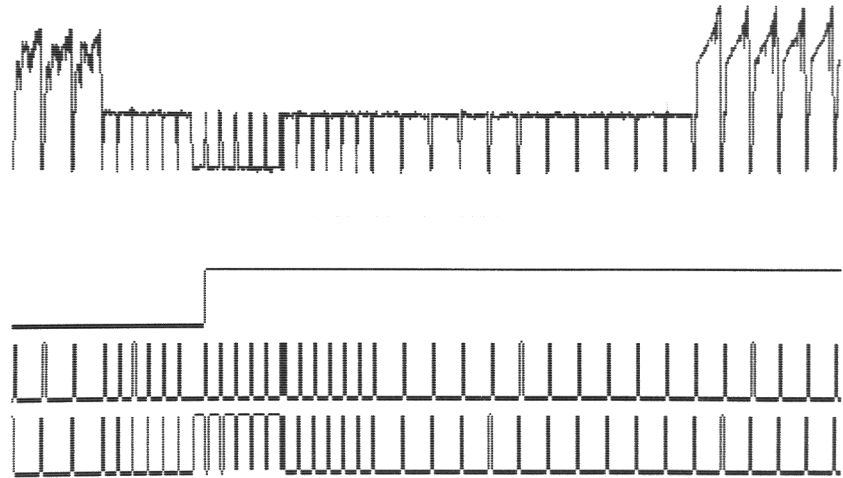
The input video signal and the associated outputs of the sync detection circuit are shown in Figure 7. The top trace is the video input for one scan line with a horizontal sync at both ends. The second trace is the field which only changes on a vertical sync. The next trace is the burst which indicates when the color burst signal is active. Finally the bottom trace is the sync, which is active during the horizontal sync.

FIGURE 7. Horizontal sync signals.



A longer segment of the signal around a vertical sync is shown in Figure 8. This shows the same signals as Figure 7 but over several lines and a vertical sync. The field changes at the beginning of the vertical pulse.

FIGURE 8. Vertical sync signals.



Right now the board uses a mini-din style connector that is similar to an SVideo connector. The advantage of this is that it does not take up much space on the board. The disadvantage is that making cables is a pain. It might have been better just to put a bunch of BNC connectors on the board.

I have designed this circuit such that it should work for NTSC, PAL, and SECAM style video signals, but I have only tested it with NTSC. The only values that deserve much consideration are C11 and R10, which form the time constant for vertical sync detection. The voltage levels generated for output signals by R9, R13 and R19, R20 would also need to be changed.

This project could easily be done using a Cypress ISP PLD or even an AMD Mach ISP PLD instead of the Xilinx.

Board layout

The PCB layout was a bit of a challenge. I wanted a ground plane under the whole analog section, but I also wanted to keep costs low by only using a two-layer board. I am very happy with the results. There are almost no breaks in the ground plane, and I got it on two layers. The jumper JP5 provides a simple way of controlling where the analog and digital grounds get connected. This digital lines are kept well separated from the analog lines. Excellent advice on minimizing noise by proper layout can be found in *High-Speed Digital Design* [7] and *Noise Reduction Techniques in Electronic Systems* [8].

The actual board layouts are shown in Figure 9 and Figure 10. The placement of the components is shown in Figure 11.

FIGURE 9. PCB top layer. Note the careful separation of analog and digital signals.

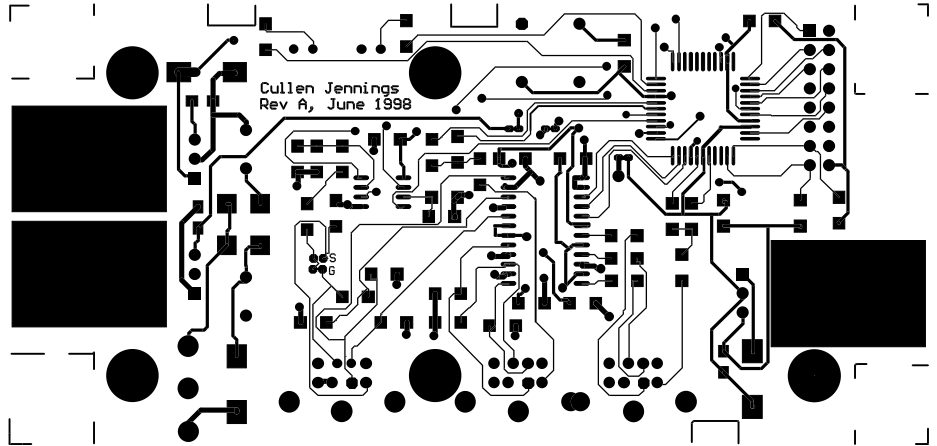


FIGURE 10. PCB bottom layer. Note the ground plane under the whole analog section.

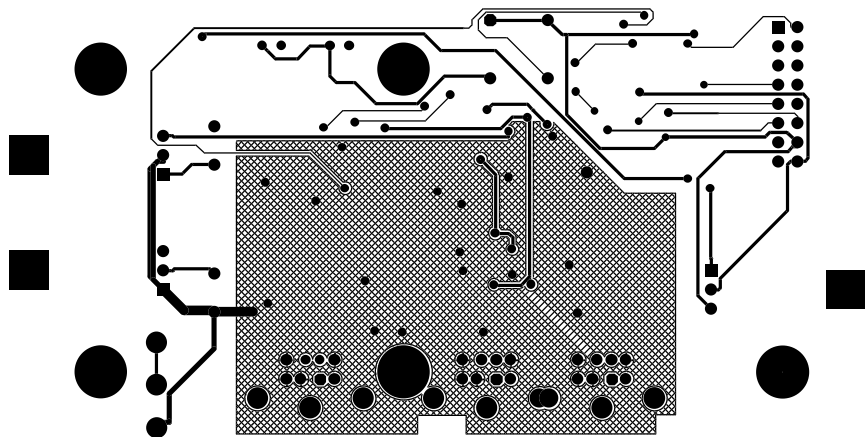
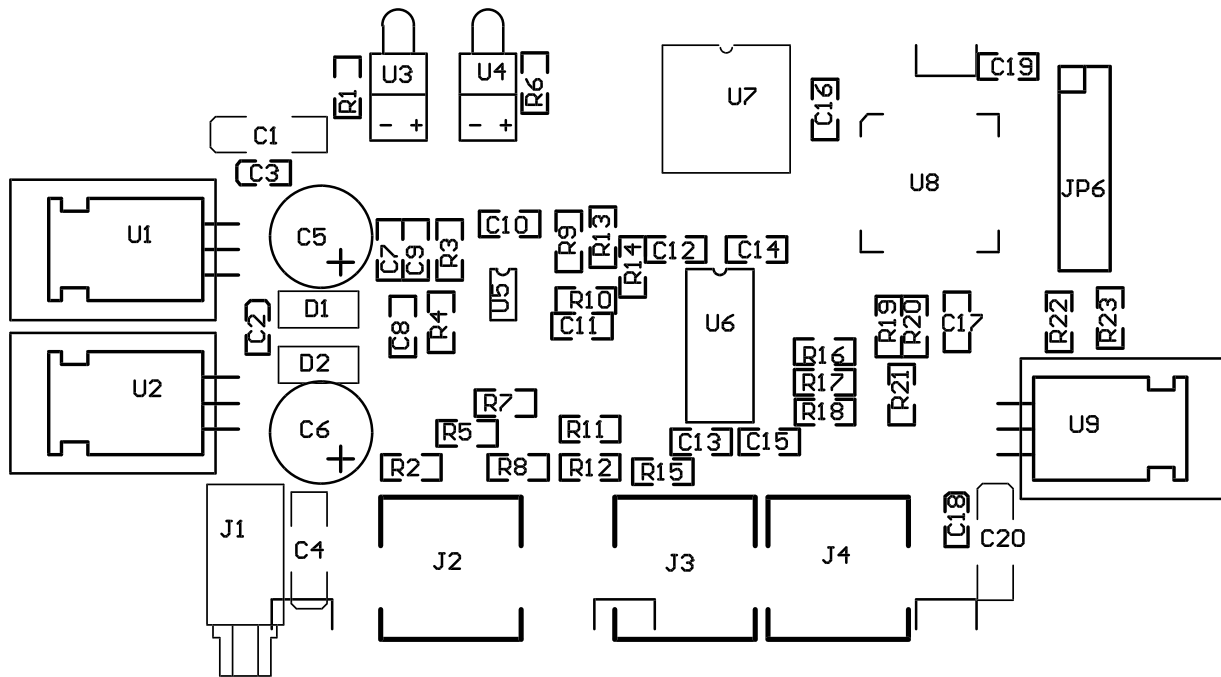


FIGURE 11. PCB component placement



Once my board layout was done, I printed it on a laser printer and then carefully checked that all the parts fit their footprints. Putting the paper on a piece of styrofoam and pushing the “through hole” components through the paper is a good way to check the jacks. The easiest way to get the footprints correct is to take some calipers and measure the components. Once I created the Gerber plots and NC drill files, I checked them using GC Preview. This is a great little program for viewing Gerber plot files that you can download [9]. This is an important step: you check that the Gerber output formats are as required by the PCB manufacturer and that the drill files align on top of the Gerber plots.

Construction

PARTS

The first stage of building the board is to order all the components. I tried to select parts that were available from mail order houses such as Digikey [10], Marshall [11], Insight [12], Arrow [13], and Nu Horizons [14]. The MAX465 is available from Arrow and Nu Horizons. With the exception of the filter formed by C7, C9 and R3, none of the values are super critical. Even the filter values are not that critical, but play with a SPICE model of the filter before you change them. The data sheet for the LM1881 explains the goals of this filter. In Table 1 I have listed very approximate costs for the components -

Construction

these could vary widely for low volume purchases. I have also added a flux pen which is very helpful in soldering the QFP.

TABLE 1. Part List

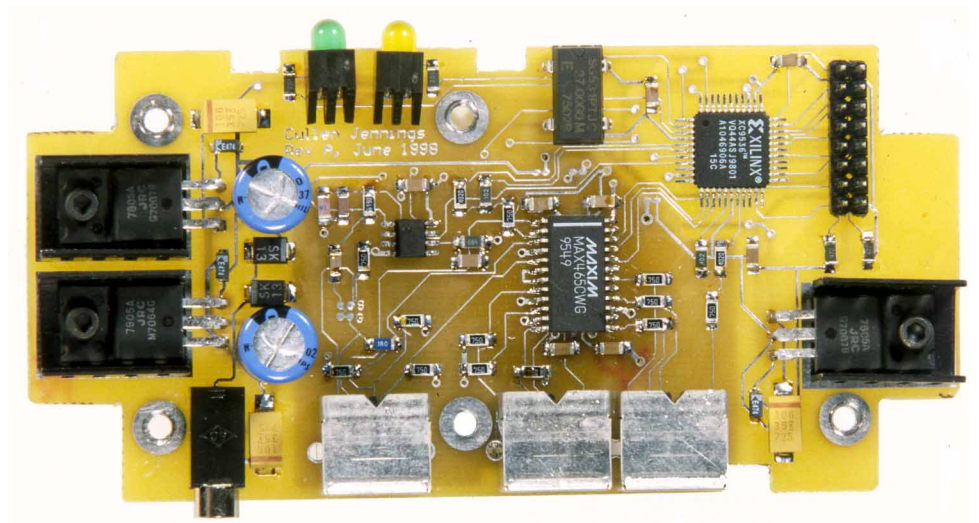
Count	Part	Designator	Footprint	Cost Each
10	0.1uF	C10 C11 C12 C13 C14 C15 C16 C17 C19 C8	1206	0.02
1	0R	R5	1206	0.02
3	10uF 35V	C1 C20 C4	7243	0.37
2	1K	R13 R19	1206	0.02
2	200R	R1 R6	1206	0.02
1	33pF	C7	1206	0.12
2	400R	R20 R9	1206	0.02
1	470pF	C9	1206	0.11
2	470uF 35V	C5 C6	Radial0.2	0.30
2	4K7	R22 R23	1206	0.02
1	619R	R3	1206	0.14
1	680K	R10	1206	0.02
12	75R0	R14 R16 R17 R18 R21 R11 R12 R15 R2 R4 R7 R8	1206	0.11
3	DIN8	J2 J3 J4		2.05
2	DIODE	D1 D2	SMB	0.58
1	HEADER 8X2	JP6	IDC16	0.25
1	JACK	J1 (MODE 3.5MM R/A PC Mount)		0.22
1	LED Green	U3 (IDT 5300H5_		0.63
1	LED Yellow	U4 (IDT 5300H7)		0.63
1	LM1881	U5	SO-8	3.33
1	MAX465	U6	SOL-24	10.79
2	MC7805	U1 U9	TO-220	0.51
1	MC7905	U2	TO-220	0.12
1	OSC-27Mhz	U7	OSC-DIP8	3.47
1	XC9536	U8 (XC9536-15VQ44C(44))	QFP44	4.70
1	PCB	(ordered in quantity 10)		13.00
1	Box	(part number Hammond 1599 B)		3.20
1	Flux Pen	(Kester 450-B-FLU-PEN)		3.80

The next stage is to order the PCB. Gerber and drill plots can be downloaded, or you can generate your own. I ordered the PCB from Alberta Printed Circuits [15], who did a nice job. They do not router out the shape of the board, so the first step in construction is to chop out the cutouts on the board with a dremel tool.

BUILDING THE BOARD

The completed board is shown in Figure 12.

FIGURE 12. Completed board



The next step is to solder on all the SMT resistors and capacitors and check that there are no shorts between VCC, +5, -5, and GND. Then add all the components in the power supply. Do the big capacitors last - they make it harder to reach other components. Check to make sure all the voltages are correct before you start adding expensive stuff. Next add the chips. At this point I applied power to the board and used my finger to check that none of the chips got hot. After a few minutes, I was satisfied that nothing was seriously wrong, so I added all the connectors and remaining components and programmed the PLD. Programming the PLD is described later in this article. Check that the input resistance on all the video inputs is 75 ohms. If everything looks good, connect up a bunch of video signals and go for it.

If it doesn't work, start tracing the signal through from the start. I powered up the board and fed a video signal into the sync input and checked a few points: first, the output of the filter on pin 2 of U5; then that the field line (pin 7 of U5) was a 30 Hz square wave; next, the PLD should output the same square wave to pin 2 of connector JP6 - it will also cause the green LED to flicker at 30 Hz. At this point, check that a signal applied to the A input gets chopped up at 30 Hz and similarly for B. Check this same signal is going to U6. Now trace the signals for the video in and out of U6.

ESD (ELECTRONIC STATIC DISCHARGE)

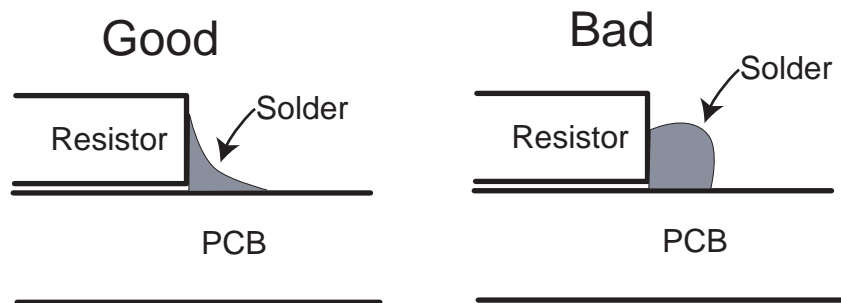
A cautionary observation: I've noticed that people with experience in production electronics tend to be careful with ESD. I guess spending several days tracking down an intermittent problem on a chip with ESD damage makes one believe that ESD problems should be avoided. I work on an ESD safe mat that is properly grounded, and I wear a ground strap. The fact that it rains most of the year where I live means that the relative humidity is great for electronics, but if you live in a dry place, take more care. Handling electronics correctly is not hard, so do it right and when something doesn't work, you won't have to wonder if it is an ESD problem. There is plenty of information about ESD on the web [16] and in books [8].

SOLDERING

One word: flux. It's all easy with flux. Surface mount wasn't designed for hand soldering, but it's easy and quick once you get the hang of it. Very little solder is required; the solder on the board is enough to tack things down.

Let's start with the 1206 resistors and capacitors. The hardest part is holding the component in place while tacking it down. If you have a tweezer solder iron, just put the component on the pad and squeeze the tweezer for a few seconds and release. I use 725°F (385°C). If you don't have a tweezer solder iron, hold the component with tweezers and touch each end with the solder iron. I find a 1/16" chisel tip works well. At this point, we have the component tacked in place but not a good solder. Get the smallest resin core solder you can find and touch the solder iron tip to the PCB and the end of the component, and then feed in a small amount of solder. The goal is a nice concave solder joint, as shown in Figure 13. Easy inspection of the joint really requires a magnifying glass or optical loupe. It is important that everything be clean - use that little sponge by your soldering iron and keep it wet.

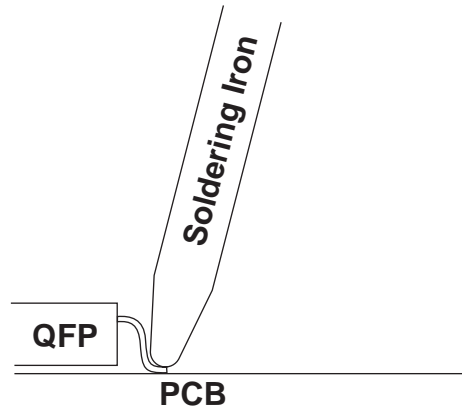
FIGURE 13. Good and bad solder joints for 1206 SMT package



Enough of the easy stuff. Let's solder the surface mount quad flat packs. The magic trick is you need a flux pen. The pen I used is listed in the bill of material in Table 1. I have experimented with a few types and they all seem to work well. Use the flux pen to smear flux all over the pads on the PCB. Set the chip on the board. Make sure it is at the right orientation - pin 1 may not be where you think: the chip has a dimple by pin 1, and the pad on the PCB for pin 1 is square - the rest are rounded. You may laugh, but I have soldered whole QFPs down only to discover that I put them on the PCB wrong - this pretty much writes off the QFP and the PCB. You've been warned.

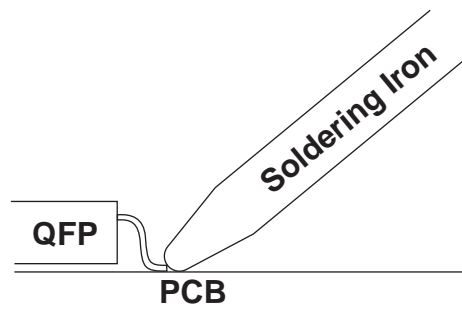
Select the finest soldering tip you can find. I use a 1/64 inch conical tip at 725°F (385°C). Make sure the chip is in about the right place, and then carefully align the two pins in one corner. Tack one of them down by touching the solder iron to the top of the pin above the pad. This will melt the solder on the pin and pad and they will bond (see Figure 14). Now carefully align the opposite corner and tack it down. Carefully check that all the pins are aligned with the pads - this is the last chance to fix anything. Now tack down all the corner pins.

FIGURE 14. Tacking down the QFP pins. Apply pressure straight down.



It is now time to solder down all the pins. Forget the idea of touching both solder and iron to the joint - you will just bridge the joints with too much solder. Get your tip to have a nice tinned coat of solder but not a glob on the end. Touch it to the joint of the pad and the end of the pin as shown in Figure 15.

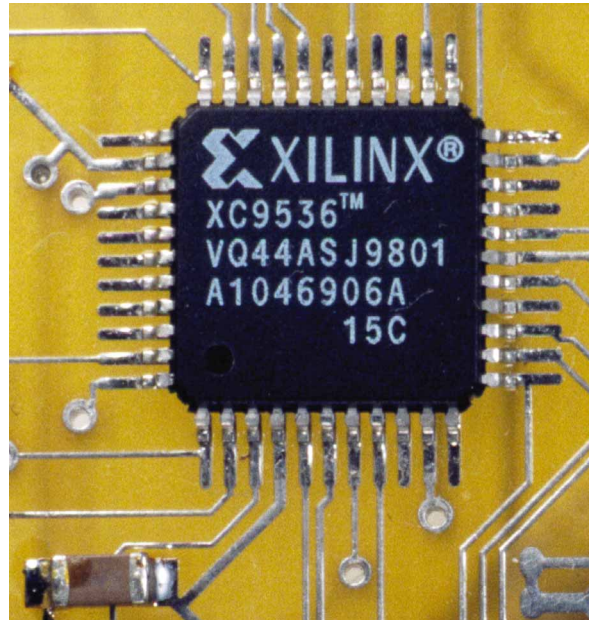
FIGURE 15. Soldering the QFP pins. Touch the soldering iron to both the pin and the pad.



Shortly, you will see the solder along the pin melt and go shiny. Hold for a half-second longer and release. Move on to the next pin. This goes really quickly. The QFPs in this project are not very fine pitch, and you can likely solder them without a magnifying glass, but having one makes the soldering easier to check. The ideal thing is a stereo microscope with a working distance of several inches. This makes the whole process easy. If you go to solder a very fine pitch SMT such as a 208 pin QFP, you pretty much need a microscope or optical loupe to check your work. I check my work by taking a 45° bent dentist probe and pulling sideways on each pin. I pull fairly hard; if it moves, it's bad. Try this on an unsoldered pin to get an idea of what it looks like. I test each pin while visually checking that the solder did not bridge two pins. I add solder to the iron every 10 to 20 pins. Once again, don't try this without flux.

The soldered QFP is shown in Figure 16. Note that there are not huge amounts of solder.

FIGURE 16. Detail of QFP soldering



VHDL

VHDL is a way of describing the desired operation of a logic device. Although it is a standard, vendors all choose different parts to implement and add their own quirks. I chose to use Xilinx PLDs with the Xilinx Foundation tool chain for place and route and Synopsys FPGA Express for synthesis. This is a very nice way of doing things, and it is great for large FPGAs, but it is not exactly the cheapest. On the other hand you might find a free evaluation version of this tool chain [11]. A much cheaper solution that would have worked just as well for projects using small PLDs is the Cypress WARP2 system [17]. For a few hundred dollars you get a VHDL text book [18], a nice VHDL synthesis tool, an in system programming cable, and a few PLDs. On an even cheaper note (like free) one could use the software from AMD and use a MACH PLD. This will not do VHDL but uses a language at about the level of the PLD equations that I provide in the next section.

VHDL PRIMER

I am not going to attempt to teach VHDL - get a good book [18][19] to really learn it - but I will show you enough things to make you dangerous. As with all programming languages, a not all bad way to learn VHDL is to examine and modify other VHDL programs. In VHDL synthesis, we describe a program that would have the same behavior as the circuit we wish to create. This program is used by the synthesis tools to create the configuration file that will configure the specific FPGA or PLD into a correct circuit. The syntax is very similar to Ada.

VHDL

The basic building blocks are signals which are virtual wires in a digital circuit. A signal is declared with code something like:

```
signal foo: STD_LOGIC;
```

To take two signals called A and B and feed them into an AND gate and then assign the output of the AND gate to a signal called C, write code that looks like this:

```
C <= A and B;
```

Busses of several wires can also be created with a declaration like:

```
signal bar: STD_LOGIC_VECTOR (7 downto 0);
```

This declares a bus with a seven wires labeled bar(7), bar(6),... bar(0). To AND the first two of these together and assign the result to the signal C, use the following expression:

```
C <= bar(0) and bar(1);
```

Other VHDL operators include:

logical : and, or, nand, nor, xor, xnor, not

relational: =, /=, <, <=, >, >=

arithmetic: +, -, *, /, rem, mod, abs, **

concatenation: &

The concatenation can be used to combine signals into signal vectors. For example, if we had a bus foo and we wanted to shift it left once into a bus called bar and assign the signal bottom to the bottom bit of bar, we could write something like:

```
signal bar: STD_LOGIC_VECTOR (7 downto 0);
signal foo: STD_LOGIC_VECTOR (7 downto 0);
signal bottom: STD_LOGIC;
```

```
bar <= foo(6 downto 0) & bottom;
```

Another useful construct is the WHEN operator. This allows us to do conditional assignment. Take for example a MUX that combines signals a and b depending on the value of the selection signal s into an output signal called result. The code for this would look like:

```
result <= a when (s = '1') else b;
```

AND GATE EXAMPLE

Let's look at complete example that creates an AND gate:

```
library IEEE;
use IEEE.std_logic_1164.all;
library metamor;
use metamor.attributes.all;
```

These lines just include some standard libraries so that things like STD_LOGIC are defined:

```
entity myAndGate is
```



```

port (
    a: in STD_LOGIC;
    b: in STD_LOGIC;
    c: out STD_LOGIC
);

attribute pinnum of a: signal is "8";
attribute pinnum of b: signal is "44";
attribute pinnum of c: signal is "7";
end myAndGate;

```

These lines tell the system that we're going to create an entity called myAndGate. One can think of an entity as a virtual chip. This circuit will have two inputs called a and b, and one output called c. The attribute pinnum commands tell the synthesis tool which physical pin number on the PLD to map a virtual wire to. Different synthesis tools do this in completely different ways, and you will have to figure out how to do it in your tool.

So far we have only told the system that the circuit exists; we have not told it what the circuit does. The following section tells the synthesis tool what the circuit is supposed to do:

```

architecture myAndGate_arch of myAndGate is
    signal result: std_logic;
begin
    result <= a and b;
    c <= result;
end myAndGate_arch;

```

The result signal is not really necessary. I included it simply to show that more signals can be declared and used in the description of the circuit. This forms a complete example.

SEQUENTIAL LOGIC

So I have only described combinatorial logic but have not addressed the issue of sequential logic such as flip flops, counters, and latches. These are described using processes. An example for a D latch would look like:

```

process (GATE, DIN)
begin
    if GATE='1' then --GATE active High
        DOUT <= DIN;
    end if;
end process;

```

Here we declare a process that output depends on to input signals called gate and din. Inside of the process we stipulate that when gate is high the value of din will be assigned to dout. When gate is low, we do not specify what is assigned to dout - it is assumed that dout will not change its value when gate is low, regardless of what happens to din:

```

process (GATE, DIN, RESET)
begin
    if RESET='1' then--RESET active High
        DOUT <= '0';
    end if;
end process;

```

```

    elsif GATE='1' then --GATE active High
        DOUT <= DIN;
    end if;
end process;

```

We can now create latches. Let's move on to look at clocked circuits such as a D flip flop:

```

process (CLK)
begin
    if CLK'event and CLK='1' then --CLK rising edge
        DOUT <= DIN;
    end if;
end process;

```

CLK'event is a special expression that is only true when CLK changes value. On each rising edge of CLK, the value of DIN will be assigned to DOUT.

Next consider a flip flop with asynchronous reset:

```

process (CLK, RESET)
begin
    if RESET='1' then--asynchronous RESET active High
        DOUT <= '0';
    elsif (CLK'event and CLK='1') then --CLK rising edge
        DOUT <= DIN;
    end if;
end process;

```

If we wanted a D flip flop with synchronous reset, the code would look like:

```

process (CLK, RESET)
begin
    if CLK'event and CLK='1' then --CLK rising edge
        if RESET='1' then--synchronous RESET active High
            DOUT <= '0';
        else
            DOUT <= DIN;
        end if;
    end if;
end process;

```

A slightly more complex example strings 4 D flip flops together to form a 4-bit asynchronous counter:

```

process (CLK, COUNT, RESET)
begin
    if RESET='1' then
        COUNT <= "0000";
    else
        if CLK'event and CLK='1' then
            COUNT(0) <= not COUNT(0);
        end if;
        if COUNT(0)'event and COUNT(0)='0' then
            COUNT(1) <= not COUNT(1);
        end if;
    end if;
end process;

```

```

    end if;
    if COUNT(1)'event and COUNT(1)='0' then
        COUNT(2) <= not COUNT(2);
    end if;
    if COUNT(2)'event and COUNT(2)='0' then
        COUNT(3) <= not COUNT(3);
    end if;
end if;
end process;

```

MORE VHDL

This section shows the complete code for the application to switch on each field:

```

library IEEE;
use IEEE.std_logic_1164.all;
library metamor;
use metamor.attributes.all;

```

In this section I simply declare all the signals connected to the PLD and set the pin numbers:

```

entity vid_sw_top is
    port (
        syncNot: in STD_LOGIC;
        odd_even: in STD_LOGIC;
        burstNot: in STD_LOGIC;

        cs_out_bar: out STD_LOGIC;
        sel_out: out STD_LOGIC;
        en_out_bar: out STD_LOGIC;

        test: out STD_LOGIC_VECTOR (7 downto 0);

        syncOutA: out std_logic_vector (1 downto 0);
        syncOutB: out std_logic_vector (1 downto 0);
        jumper: in std_logic_vector(1 downto 0);
        green: out std_logic;
        yellow: out std_logic;

        clk: in std_logic
    );

    attribute pinnum of burstNot: signal is "8";
    attribute pinnum of syncNot: signal is "44";
    attribute pinnum of odd_even: signal is "7";

    attribute pinnum of cs_out_bar: signal is "14";
    attribute pinnum of sel_out: signal is "13";
    attribute pinnum of en_out_bar: signal is "12";

    attribute pinnum of test: signal is
        "27,29,31,36,28,30,32,33";
    attribute pinnum of syncOutA: signal is "6,5";
    attribute pinnum of syncOutB: signal is "16,18";

```

```

attribute pinnum of jumper: signal is "22,21";
attribute pinnum of green: signal is "3";
attribute pinnum of yellow: signal is "2";
attribute pinnum of clk: signal is "43";

end vid_sw_top;

architecture vid_sw_top_arch of vid_sw_top is
signal vert: std_logic;
signal field: std_logic;
signal horz: std_logic;
signal vertStart: std_logic;
signal burst: std_logic;
signal sync: std_logic;
signal syncGen: std_logic;
signal syncOut: std_logic_vector (1 downto 0);
signal line: std_logic_vector (7 downto 0);
begin

```

Here I invert some of the signals and set the fixed outputs:

```

en_out_bar <= '0';
cs_out_bar <= '0';
field <= odd_even;
sync <= not syncNot;
burst <= not burstNot;

```

This line sets the output to switch on fields:

```

sel_out <= field;

```

If the above line were commented out and the following line commented in, the system would take the first input for scan lines between (binary) 01111111 and 11000000 and the rest of the image from the other input:

```

-- sel_out <= '1' when (( line > "01111111") and (line <
"11000000")) else '0';

```

These lines set the green led to always be on and the yellow led to flash at 30Hz as each field is detected:

```

green <= '1';
yellow <= field;

```

Various signals are put out to the test connector so that they can be monitored with a logic analyzer:

```

test(0) <= sync;
test(1) <= burst;
test(2) <= field;

test(3) <= horz;
test(4) <= vert;

test(5) <= '1';

```

```

test(6) <= '0';

test(7) <= clk;

```

This process detects the vertical sync. It does this by looking at the value of the sync signal during the color burst phase of the sync:

```

process (burst)
begin
    if burst='0' and burst'event then
        vert <= sync;
    end if;
end process;

```

This process generates the horizontal sync signal. It is a flip flop that is set whenever there is a sync that is not the vertical sync and is reset on the start of the next burst event:

```

process (burst,sync,vert)
begin
    if sync='1' and vert='0' then
        horz <= '1';
    elsif (burst'event and burst='0') then
        horz <= '0';
    end if;
end process;

```

This process sets the line to the count of the line in the field that we are on. It counts the horizontal sync pulses and uses the vertical sync pulse to reset to zero:

```

process (horz, line, vert)
begin
    if vert='1' then
        line <= (others => '0');
    else
        if horz'event and horz='1' then
            line(0) <= not line(0);
        end if;
        if line(0)'event and line(0)='0' then
            line(1) <= not line(1);
        end if;
        if line(1)'event and line(1)='0' then
            line(2) <= not line(2);
        end if;
        if line(2)'event and line(2)='0' then
            line(3) <= not line(3);
        end if;
        if line(3)'event and line(3)='0' then
            line(4) <= not line(4);
        end if;
        if line(4)'event and line(4)='0' then
            line(5) <= not line(5);
        end if;
        if line(5)'event and line(5)='0' then
            line(6) <= not line(6);
        end if;
    end if;
end process;

```

```

        if line(6)'event and line(6)='0' then
            line(7) <= not line(7);
        end if;
    end if;
end process;

```

Here the values of the sync out signal are generated. The grey scale value is set depending on which scan line in the field we are at:

```

syncGen <= (not burst) when (vert='1') else (not sync);

syncOut(0) <= line(6) when (horz='0' and syncOut(1)='1'
                        else syncGen;
syncOut(1) <= line(7) when (horz='0') else '0';

```

The next section just drives the DAC resistors to the correct values specified by the syncOut signal:

```

syncOutA(1) <= '1' when (syncOut(0) = '1') else '0'; -- 1K
syncOutA(0) <= '1' when (syncOut(1) = '1') else 'Z'; -- 402R

syncOutB(1) <= '1' when (syncOut(0) = '1') else '0'; -- 1K
syncOutB(0) <= '1' when (syncOut(1) = '1') else 'Z'; -- 402R

end vid_sw_top_arch;

```

THE PLD EQUATIONS

All of this stuff gets collapsed into logic equations for PLD by the synthesis tools. The VHDL code from above turns into the following equations:

```

/cs_out_bar = Vcc
/en_out_bar = Vcc
/green = Gnd
/sel_out = /odd_even
/"syncOutA<0>" = Gnd
    "syncOutA<0>".TRST = /"test<3>" * line_7
/"syncOutA<1>" = /"test<3>" */line_6 * line_7
    + /burstNot * "test<4>" */line_7
    + /syncNot */"test<4>" */line_7
    + /burstNot * "test<3>" * "test<4>"
    + /syncNot * "test<3>" */"test<4>"
/"syncOutB<0>" = Gnd
    "syncOutB<0>".TRST = /"test<3>" * line_7
/"syncOutB<1>" = /"test<3>" */line_6 * line_7
    + /burstNot * "test<4>" */line_7
    + /syncNot */"test<4>" */line_7
    + /burstNot * "test<3>" * "test<4>"
    + /syncNot * "test<3>" */"test<4>"
/"test<0>" = syncNot
/"test<1>" = burstNot
/"test<2>" = /odd_even
/"test<3>" := Vcc
    "test<3>".CLKF = burstNot
    "test<3>".SETF = /syncNot */"test<4>"
    "test<3>".PRLD = GND

```

```
/"test<4>" := syncNot
    "test<4>".CLKF = burstNot
    "test<4>".PRLD = GND
/"test<5>" = Gnd
/"test<6>" = Vcc
/"test<7>" = /clk
/yellow = /odd_even
/line_0 := line_0
    line_0.CLKF = "test<3>"
    line_0.RSTF = "test<4>"
    line_0.PRLD = GND
/line_1 := line_1
    line_1.CLKF = /line_0
    line_1.RSTF = "test<4>"
    line_1.PRLD = GND
/line_2 := line_2
    line_2.CLKF = /line_1
    line_2.RSTF = "test<4>"
    line_2.PRLD = GND
/line_3 := line_3
    line_3.CLKF = /line_2
    line_3.RSTF = "test<4>"
    line_3.PRLD = GND
/line_4 := line_4
    line_4.CLKF = /line_3
    line_4.RSTF = "test<4>"
    line_4.PRLD = GND
/line_5 := line_5
    line_5.CLKF = /line_4
    line_5.RSTF = "test<4>"
    line_5.PRLD = GND
/line_6 := line_6
    line_6.CLKF = /line_5
    line_6.RSTF = "test<4>"
    line_6.PRLD = GND
/line_7 := line_7
    line_7.CLKF = /line_6
    line_7.RSTF = "test<4>"
    line_7.PRLD = GND
```

VARIATIONS

To switch every N frames, just set up a counter to count even fields and then switch when it hits an appropriate value. Similarly, to switch part way through a scan line, set up a counter that is reset by the horizontal pulse and then counts clock ticks from the oscillator, switching the output when it hits an appropriate value.

In system programming of the PLD

JTAG is a protocol originally designed for testing the connection between chips. It has proven to be useful for many things including debugging and programming devices. Basically it is a serial protocol. Each device has 4 wires, a TMS signal that puts the

Downloading the design

device into test mode, a TCK line that clocks the serial data into the system, and a TDI test data in and TDO test data out. The TMS and TCK are all connected together and controlled by the master device doing the testing. The TDI and TDO are daisy chained together to form one long string with all the devices through which the serial data passes. Different commands can be sent to each device, including commands just to pass the data through on the serial line to the next device along the chain. Basic commands allow the chip to set the output level of any pin on the chip and read the logic level of pins on other chips. This allows a JTAG test system to check that all the traces between chips are correctly connected and there are no shorts. Extended commands allow us to program the PLD via the JTAG interface. This in system programming is very convenient: there is no need to own a programmer that has sockets for many different SMT parts, pins don't get bent as parts go in and out of the programmer, and there is no need to socket the part on the board so that it can be reprogrammed. It's all very nice - and, for the development budget-conscious - cheap.

The simplest way to program the PLD is to use the XChecker cable from Xilinx along with their software. Simply connect up the JTAG pins on the cable and the ground but not the Vcc, power up the board, and program it. There are more comments on this in the Xilinx application note XAPP069 [5].

If you do not have the Xilinx software and are going to use the configurations files I have provided, life will be more interesting. Get the application note XAPP058 from the Xilinx web site [5] and the associated software for PCs. Wire up a cable so that you can use your parallel port data lines to drive the TDO, TMS, and TCK lines, and read the TDI line by connecting it to the ERROR line. With this circuit, there is no need to buffer the signals. Have a look in section 10.20 of *The Art of Electronics* [20] for more information. Use the software in the application note to program the PLD. You will need to modify the port access code to match the wiring of your parallel cable. Examples of how to do this can be found in section 29.1.5 of *The Indispensable PC Hardware Book* [21].

Downloading the design

All of the files associated with this project are available for downloading, including the schematics in Protel, net lists, bills of materials, Gerber and Postscript plots of the PCB, drill files, VHDL code for the PLD, reduced equations of the PLD that could be implemented in a language other than VHDL, and configuration images for programming the PLD for those who do not want to synthesize their own PLD programs.

Conclusion

I've enjoyed building this project, and it works great. I hope it works for you - send me some email about new applications you find for it. A few people I would like to thank for help and advice along the way are Stewart Kingdon, Rod Barman, Lyndsay Campbell, and Alan Hawrylyshen who took all the photographs in this article.

References

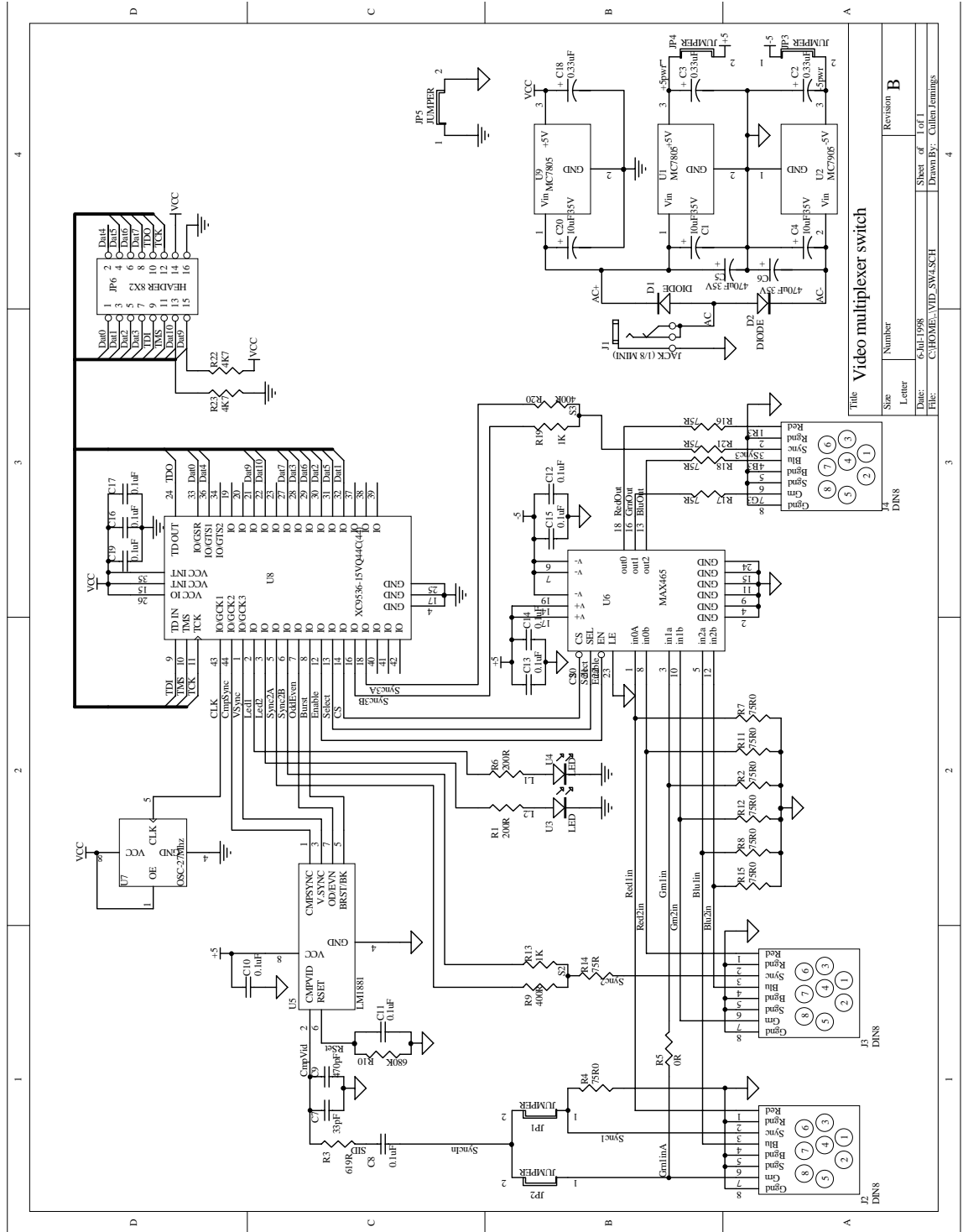
- 1 Keith Jack, *Video Demystified*, 2nd ed. (San Diego: High Text, 1996).
- 2 National Semiconductor LM1881 Video Sync Separator Data Sheet - <http://www.national.com/ds/LM/LM1881.pdf>
- 3 <http://www.interactiv.com>
- 4 Xilinx, *The Programmable Logic Book Data Book* (San Jose: Xilinx, 1998).
- 5 <http://www.xilinx.com>
- 6 Maxim Max463-470 Two Channel, Triple/Quad RGB Video Switches and Buffers <http://www.maxim-ic.com/efp/AllParts.htm>
- 7 Howard W. Johnson and Martin Graham, *High-Speed Digital Design: A Handbook of Black Magic* (Englewood Cliffs, NJ: Prentice Hall, 1993).
- 8 Henry W. Ott, *Noise Reduction Techniques in Electronic Systems*, 2nd ed. (New York: Wiley, 1988).
- 9 <http://www.graphicode.com>
- 10 <http://www.digikey.com>
- 11 <http://www.marshall.com>
- 12 <http://www.insight-electronics.com>
- 13 <http://www.arrowamericas.co>
- 14 <http://www.nuhorizons.com>
- 15 <http://www.apcircuits.com>
- 16 <http://www.eosesd.org>
- 17 <http://www.cypress.com>
- 18 Stefan Sjöholm and Lennart Lindh, *VHDL for Designers* (London: Prentice Hall, 1997).
- 19 Keven Skahill, *VHDL for Programmable Logic* (Reading, MA: Addison-Wesley, 1996).
- 20 Paul Horowitz and Winfield Hill, *The Art of Electronics*, 2nd ed. (Cambridge, MA: Cambridge UP, 1989).
- 21 Hans-Peter Messmer, *The Indispensable PC Hardware Book: Your Hardware Questions Answered*, 2nd ed. (Wokingham, UK: Addison-Wesley, 1995).
- 22 <http://www.iii.ca>
- 23 <http://www.cs.ubc.ca/spider/jennings>

Biography

Cullen Jennings works for Image Integration Inc. [22], a computer consulting company. For the last seven years Image Integration Inc. has developed software and network solutions for oil companies and software for air traffic control. Cullen has an M.Sc. in Computer Science and is currently a Ph.D. candidate at the University of British Columbia [23]. Cullen can be reached at c.jennings@ieee.org.

Appendix A - Schematics

FIGURE 17. Schematics



Appendix B - Data Sheets

Appendix C - Application notes

Appendix D - Figure Captions

FIGURE 1. Completed Project

FIGURE 2. Block diagram for video switch

FIGURE 3. Video signal for a single scan line.

FIGURE 4. Horizontal sync signal between two scan lines.

FIGURE 5. Vertical sync between two fields.

FIGURE 6. Calculated filter response. This will have a nearly -18 dB effect on the color burst signal while having little effect on the sync signal which is mostly under 500KHz.

FIGURE 7. Horizontal sync signals.

FIGURE 8. Vertical sync signals.

FIGURE 9. PCB top layer. Note the careful separation of analog and digital signals.

FIGURE 10. PCB bottom layer. Note the ground plane under the whole analog section.

FIGURE 11. PCB component placement

FIGURE 12. Completed board

FIGURE 13. Good and bad solder joints for 1206 SMT package

FIGURE 14. Tacking down the QFP pins. Apply pressure straight down.

FIGURE 15. Soldering the QFP pins. Touch the soldering iron to both the pin and the pad.

FIGURE 16. Detail of QFP soldering

FIGURE 17. Schematics