EE 281 October 21, 2002
Embedded Systems Design Laboratory Handout #7

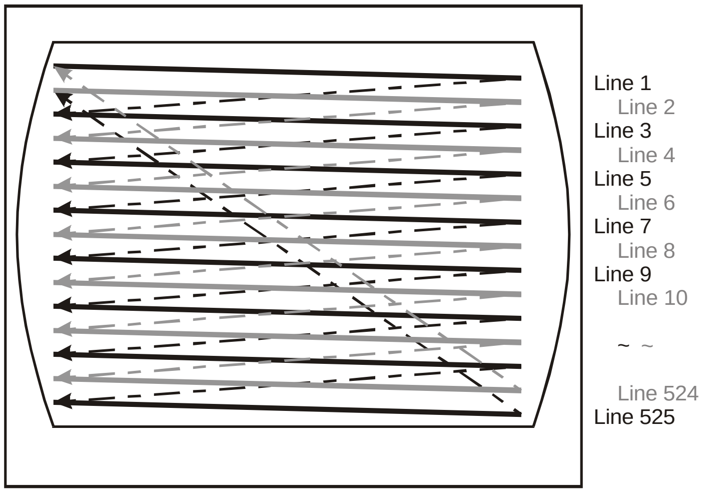
Laboratory Assignment #4:
"TV Paint"

Due Wednesday, October 30, 2002

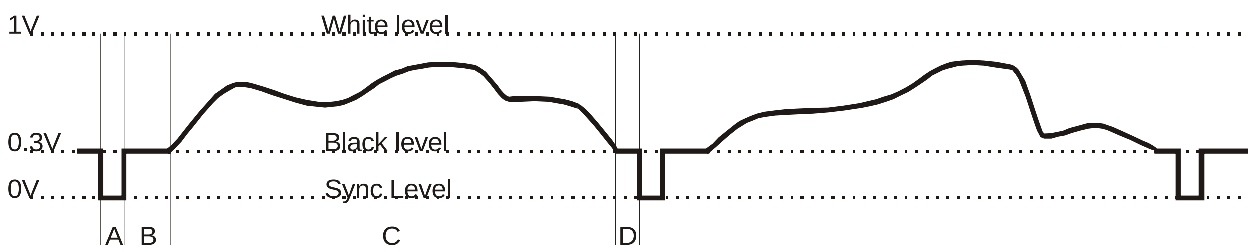
1. How To Use Your TV

Have you ever imagined using your TV as a display for a home project? If not, then perhaps you should have. Televisions, like most mass-produced technological wonders, have standard interfaces. For example, those little yellow RCA connectors on the back of your TV and/or VCR are designed to accept a video signal that's described by the US standard RS-170A (NTSC). If you build something which outputs video in accordance with the standard, you can plug it into a TV and it's more or less guaranteed to work. What's even more interesting is that a carefully programmed AVR processor can quite easily generate just such a video signal! (Did you see that coming?) This section will briefly describe the details of video signals and how they're displayed.

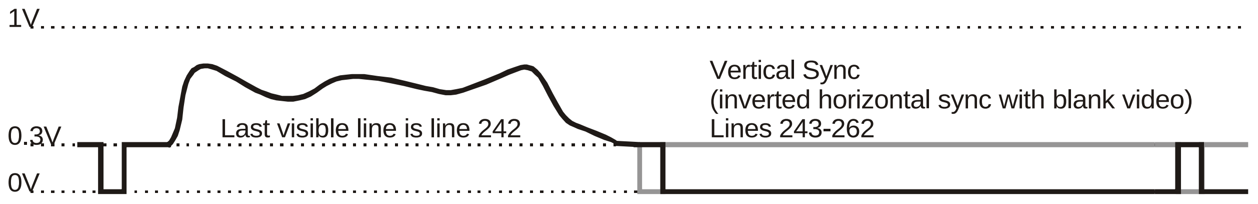
Televisions are raster-scan devices, which mean that they trace out their images by quickly moving a single electron beam in a "Z" pattern across the display tube. The beam starts in the upper left corner of the display, and traces quickly across to the upper right "painting" the first horizontal line of the TV image. When it reaches the right side, the beam turns off, returns to the left side, and paints the second horizontal line. This process continues until all the lines in the TV frame have been painted. A complete TV image is 525 lines tall and is fully painted 30 times per second. However, in the early days of TV (1930s), it was discovered that the 30Hz refresh rate produced too much flicker to watch comfortably. To reduce this flicker, each full image is split into two frames of 262.5 lines each, and thus frames are displayed at a rate of 60Hz. The first frame contains lines 1,3,5,7… of the original image, and the second frame contains lines 2,4,6,8, etc. Each frame fills the entire screen, but only paints every other line. For practical purposes, you can discard the final 0.5 line of each frame to make it an even 262 lines/frame. If you take a moment to do the math, you will find that each line takes 63.5us to paint (15.75kHz).



As you might suspect, how a TV displays images is closely related to the structure of the video signal itself. US standard video (RS-170A) manages to encode sync information, intensity, and color into a single analog signal, and thus into a single wire. This is an amazing feat, but a believable one. Consider that the displaying of TV frames is entirely a serial operation; dot after dot, line after line, frame after frame. So shouldn't we be able to sequentially transmit an image as a single signal. The signal could use a changing voltage to indicate the intensity of pixels as the TV paints out each line, frame, and image. In fact, this is exactly what happens, with some extra sync information added.



A - Horizontal Sync Pulse (4.7us)
B - "Back Porch" Non-visible pre-scan region(5.9us)
C - Visible scan region (51.5us)
D - "Front Porch" Non-visible post-scan region (1.4us)
Total scan time for one line is 63.5us


1V
Vertical Sync
(inverted horizontal sync with blank video)
Lines 243-262
0.3V
Last visible line is line 242
0V

Above is a depiction of four lines of video. Every line begins with a horizontal sync pulse which tells the television to begin a new line. After the sync, the majority of the video signal is simply a varying voltage, between 0.3V and 1V, which controls the intensity of the electron beam as it paints out the line. A voltage of 0.3V gives black, while 1V corresponds to white. Note the regions called "Back Porch" and "Front Porch". These are black-level areas in each line where the electron beam is hidden by the left and right edge of the physical TV tube. Remember that it takes 262 of these lines to make a complete frame, only 242 of which are visible. The final 20 lines are actually a vertical sync. The vertical sync instructs the TV to prepare for the next frame by sweeping up to the top of the display again. You can create a vertical sync simply by sending 20 blank video lines with the horizontal sync inverted. We haven't discussed how color is encoded because it's beyond the scope of this project, but if you're curious, just ask.

2. The Lab Assignment

Your goal for this lab is to program the AVR mega163/323 processor to make a TV "paint" system. The basic idea here is to use a joystick input to allow users to position a dot on the TV screen. Two pushbuttons should allow the user to set the dot at the current position to white or black. To paint, you should be able to hold down the "set to white" button and draw with the joystick. To erase, hold the "set to black" button and erase with the joystick. You may use either assembly or C, or both, to complete the code for this lab. Your paint system must include at least the basic features 1, 2, and 3 below. Any additional features can be implemented for 10% extra credit each. **For you own sake, try to implement your program in blocks that can be tested independently. If you write the entire program all at once, there will be little hope of locating bugs.**

1. Read an X-Y position from a joystick (essentially two variable resistors). (required)

2. Continuously generate a video signal that shows the joystick position as a blinking white dot on a black screen. This is your cursor. Your cursor should travel over the full range of the joystick. **The minimum resolution of your graphic video output should be at least 10x10 pixels**. The video signal must work on any reasonable TV and a TV will be provided in lab for testing. (required)

3. Set up two pushbuttons to set the current dot (as defined by the joystick) to white or black. The dot must retain the setting even when the joystick cursor is moved to another dot. If the "set to white" button is held while the joystick is moved, dots should turn on as the cursor passes over them. Erasing should work in a similar fashion. (required)

4. Improve the paint program to allow for some grayscale. This isn't as hard as it may sound. (optional/extra credit)

5. Display some dynamic text or numbers on your display, maybe as part of an intro splash screen. By dynamic, I mean that the text is not hard-coded but can be changed by your program. For example, you could display the coordinates of the joystick using two digits in the corner of the screen. The difficulty of doing this will largely be determined by how your other code was written; it can range from easy to very hard. (optional/extra credit)

In order to effectively accomplish your mission, you'll need to put the pedal to the metal and clock your Mega163/323 as fast as it can go. Stop by lab and pick up an 8MHz crystal. You should insert the crystal in the socket marked "CRYSTAL" on the STK500 and move the OSCSEL jumper to cover pins 2-3. Don't forget to make sure that your Mega163/323 is set to "External Clock BOD Enabled" in the "Fuses" section of the STK500 programmer.

You will also need to connect your generated video output to a TV or VCR video input so you can see it. There will be a cable hooked to a TV in lab for this purpose. I will try to make the materials available so that you can make your own RCA cable. Remember, if you need some specific components for EE course work at Stanford, Ed Dillard (Packard 112) may be able to help you. He has a good stock of most standard components.

3. Frequently Asked Questions (and Important Advice)

3.1. My code doesn't work and I can't figure out why!

I'm making this bit of advice a separate paragraph because it's so important. **When you attack a complex problem like this lab, you MUST do it in small pieces**. Don't expect your program to work if you code everything up all at once. Test your A/D code independently of your video generation. You may even test your video generation in several stages. Maybe test your analog output voltages independently of everything else. This goes for troubleshooting too. **Divide possible problems, and conquer them one by one.** The best thing you can do while troubleshooting is to run software or hardware tests that ***prove*** certain parts of your program/system are running correctly. That will let you focus your concentration on where the problem really lies.

3.2. Generating video seems really complex. What are the most important things I need to do to get it right?

Generating the sync signals is the most important thing to do correctly. If the syncs are done right, the rest of the video will be cake. If the sync signals are not correct, then the TV won't be able to sync and will either display what looks like an encoded pay-per-view channel, or the familiar "I have no signal" blue screen. Here are some words of advice: Try to meet the timing spec as closely as possible…but exact pulse widths are actually not that important. You can be off on lots of things by 5-25%. **What's most important is that you're consistent!** Every video line should have the same timing. The horizontal sync (Hsync) is the most sensitive. It's okay if the time between Hsyncs in your signal is really 61us instead of 63.5. What's important is that your Hsyncs do appear every 61us, never late, never early. **One easy mistake:** Don't forget to account for conditional branches: if you branch, it takes 2 clock cycles, if you don't, it takes only one.

3.3. But how can I possibly generate a signal that's as fast and accurate as real video?

True, at first glance, generating accurate 4.7us pulses seems near impossible. But that's far from the truth. Your mega163/323 is running at 8MHz with most instructions being executed in a single clock cycle. That means you can execute 8 instructions every microsecond. Now do you feel like you can do it?

3.4. Wait, I know how to use the A/D converter, but how can I generate analog voltages with the AVR? Don't I need special hardware?

In this case, you should be able to answer your own question, but I'll give some big hints. First, remember that you only really need to generate three voltages: 0V, 0.3V and 1V (these can even be approximate). Second, I'll tell you that one possible solution involves only 2 I/O port pins, and 2-3 resistors. Third, don't forget that most video inputs have an impedance of about 75 ohms to ground. You may need to take this into account. If you need strange resistor values, see Ed Dillard.

3.5. But how can I…

Trust me, you will succeed! In fact, the whole thing can be done in less than 200 lines of assembly. I encourage discussion amongst yourselves regarding various ideas, techniques and concepts. I welcome questions about the lab, especially if you need some conceptual help or clarifications.

*Submit: Your *.asm file(s) and/or *.c and *.h file(s), and a readme.txt explaining your code in reasonable detail (a few paragraphs). Zip these files and submit them to the TA by the due date.*