

It will definitely help if you have a copy of the schematic to refer to in this discussion.

RV1 - RV4 are potentiometers. I used rotary, but if you have them, slide pots (faders) work just as well. The ones I happened to have handy were 50K Ohms, but any value between 5K and 100K should work well. The higher values reduce the drain on the battery.

The wiper (the arrowhead coming in from the right) picks up a voltage in the range 0 to 5 volts, depending on its position. The combination of R1 and C1 acts as a filter, to keep this signal from picking up any noise on the way to the ITTYBITTYCx chip. R2 and C2 do the same thing for the signal from RV2, and so on and so forth.

The ITTYBITTYCx chip is a microcontroller - a single integrated circuit that includes a microprocessor, memory and some peripherals, all on the same sliver of silicon. The particular microcontroller I used is a Philips Semiconductor 87LPC767. I put a link to its 55-page data sheet in my first post on ControlBooth.com.

The three peripherals we are using are an analog-to-digital converter, a timer, and a serial port.

The analog-to-digital converter takes the voltage from each potentiometer and turns it into a binary number in the range 0 (0 volts) to 255 (5 volts). When it has finished converting one voltage, it interrupts the microprocessor.

An interrupt is a hardware signal that tells the microprocessor "stop what you're doing, mark your place, and come do this for a while." "This" is to take the number from the A to D converter and put it into one of two memory locations (depending on the setting of SW1), then set up the A to D converter for the next conversion. It only takes the microprocessor about 15 microseconds to service the interrupt. After that, the microprocessor goes back and picks up where it left off. Interrupts are a way of making a microprocessor appear to be doing two or more things at once. We'll go into it a little deeper when we analyze the program.

The serial port takes 8-bit data and sends it out on pin 12, one bit at a time. It automatically adds a start bit and two stop bits, creating serial data in proper DMX-512 frames. Part of the serial port's operation is controlled by the main part of the program and part of it is an interrupt service routine... the serial port has an interrupt signal, too.

In the main part of the program, the microprocessor reads the address switches and puts address into a location it can use as a counter. It generates the MARK (8 or more microseconds with the serial port output high, or "logic 1", then tells the serial port to send a "null-start" frame. Then it waits while the serial port does so. Then it tells the serial port to send a zero data frame for each address from 1 to whatever was set when it read the address switches. For each zero data frame it has to wait until the serial port has sent one before telling it to send another. It only takes about 7 microseconds to tell the serial port to send a frame, but it takes the serial port 44 microseconds to do so. During that 44 microseconds, it can be interrupted by the A to D converter, service it, come back, and still have to wait.

Once the main program has told the serial port to send the right number of zero data frames, it sets an address register used by the serial interrupt. It points to the first of eight memory locations, two per fader, that are being continuously updated in the A to D converter interrupt routine. Then it enables the serial port interrupt and waits for the serial interrupt to disable itself.

When the serial port finishes sending a frame, it interrupts the microprocessor. The serial interrupt service routine takes a number from memory and tells the serial port to send it out. It points the address register to the next memory location and checks to see if it's gone past the end of the 8 addresses used for fader data. If so, it disables the hardware interrupt. Then it lets the microprocessor go back to what it was doing (waiting). While it's waiting, the processor is also looking to see if the serial interrupt is still enabled. If so, it keeps waiting.

When the processor sees that the serial interrupt has disabled itself, it knows the last frame has been sent. It pulls the serial port output pin low, creating the "break" required between DMX-512 packets. It sets the timer for a time long enough to fulfill the minimum break time requirement of the DMX-512 specification, then waits for the timer to tell it "time's up." Then it goes back to the beginning, reading the address switches again.

As part of this "main loop," the processor decrements an 8-bit register. Every time the register hits zero, it complements the "ALIVE" output on pin 10. "Complement" means to change it to its opposite. If it was a logic 0 (0 volts) it becomes logic 1 (5 volts). If it was logic 1, it becomes logic 0. This has the effect of causing the "ALIVE" LED to blink slowly, indicating the processor is doing its job properly.

The output of the serial port is single-ended, meaning it's just one pin that swings to +5 volts for a "1" bit and to ground (zero volts) for a "0" bit. DMX-512 calls for a differential signal. U2, the SN75176, converts the single-ended logic signal to the proper differential signal.

U3, the LM7805CT, is a voltage regulator. Its output will be a rock-steady 5 volts for any input between about 7.2 volts and 30 volts. This is necessary because the logic chips, both the 87LPC767 and the SN75176, want to see 5 volt power, pretty tightly regulated, on their VCC pins. The 87LPC767 can operate down to 3.3 volts comfortably, but the SN75176 isn't as forgiving, and either one will smoke at voltages above 7.

Y1 is an 8 MHz crystal. It's used as a frequency reference for all the timing. The bit clock for the serial port is 1/32 the crystal frequency. 1/32 of 8,000,000 is 250,000 - the precise bit rate required for DMX-512. Because this 8 MHz is a pretty high frequency, it's important to keep the wires between Y1 and U1 as short as possible.