SOFTWARE OF INTERVALOMETER

The software is a simple bit of assembler compatible with GNU's gpasm. It's not particularly elegant or efficient, but seems to work. You can grabthe source and a hex file from GitHub.

To understand the code, it's helpful to know that Timer1 is configured to generate interrupts at about 16Hz, and most of the code runs inside the Timer1 interrupt. Timer1's is clocked by the system oscillator, prescaled 4:1, and so counts at about 8kHz. Thus, it's synchronous with the instruction clock—instructions take four ticks on this processor. To generate the 16Hz interrupts, we need 512 counts per interrupt. Incidentally 16Hz is somewhat arbitrary: it needs to be fast enough to scan the switches, but that's about all.

When controlling the outputs it's worth remembering that the longest interval between triggers is 30 days, or about 42 million periods of 16Hz. That's just too much for a 24-bit counter:

$$30\times86400\times16=41,472,000,\approx2_{25.3}.$$

However we could use a 24-bit counter if we increment it at 4Hz i.e. on every fourth timer interrupt:

$$30\times86400\times4=10{,}368{,}000,\approx2^{23.3}.$$

So then there's our basic design. Timer1 generates interrupts at 16Hz which we'll use to scan the inputs. Every fourth interrupt we we'll increment a 24-bit counter which controls the outputs. All the output transitions happen at small values of the counter, after which there will be a period of inactivity while we wait for things to start again.

Happily all the interesting transitions happen in the first 64s, so the state machine which drives them can deal with purely 8-bit quantities. It's only the overflow detection which needs the full 24-bit calculation.

The transition state machine can be further simplified because The state of the status LED can be inferred from the other outputs and the clock phase. Accordingly we don't need to explicitly list its transitions. This is both simpler and reduces the chance that the LED doesn't reflect the true status.

If you want to understand the details of the output transitions then read the code, but basic idea is that we first ask the camera to focus, then ask it to take a picture.

If we're controlling power, then we need to apply it before focussing and turn it off again some time after taking the photo.

The status LED is off when nothing's happening then flashes progressively brighter, staying on continuously when the shutter's triggered.

The traces below illustrate this. We begin with the standard behaviour, used when the interval's a minute or longer. As the interval increases the featureless area to the right extends. It's been arbitrarily truncated here.

For shorter intervals the whole sequence is compressed. Here's the 10s version:

Finally the sequence used when we want to control the power is longer still: too long to sensibly display on the page.

Although the pictures are pretty, if you want to explore this in more detail you're better off reading the source code.

Source: http://www.mjoldfield.com/atelier/2011/08/intervalometer.html