THIS MONTH'S PROJECTS

LEVEL RATING SYSTEM

To find out the level of difficulty for each of these projects, turn to our ratings for the answers.

●○○○ . . . . Beginner Level
●●○○ . . . . Intermediate Level
●●●○ . . . . Advanced Level
●●●● . . . . Professional Level

I have always had an interest in animation and time lapse photography. With a digital camera, I found that making time lapse movies became fun and economical.

Time lapse photography is the process of taking pictures at regular intervals.

When you combine the pictures in sequence, they create a movie.

FIELD PROGRAMMABLE INTERVALOMETER

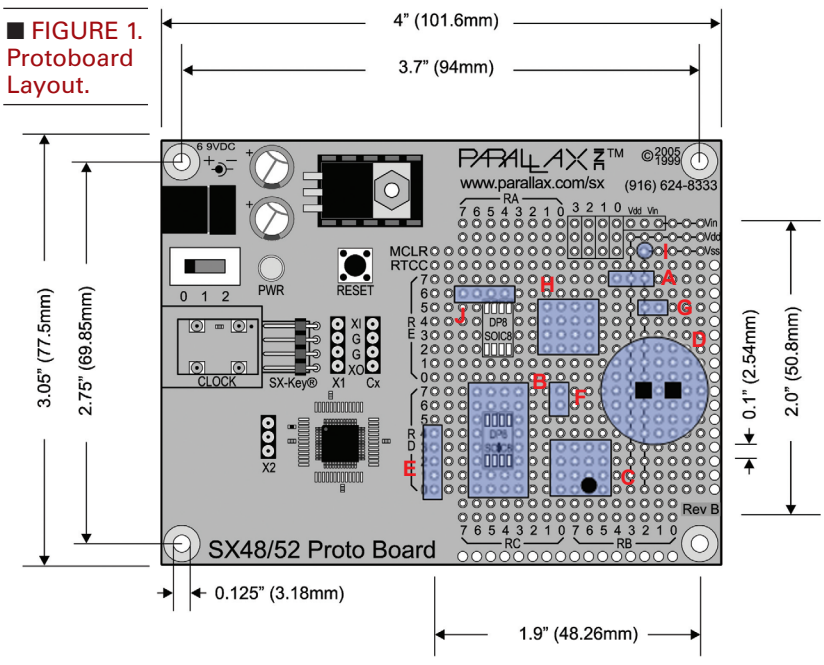
For instance, set up a camera to take two pictures of a blooming flower every minute for three hours. Convert the resulting 360 photographs to an MPEG movie, and watch the flower bloom in under a half minute. I made short time-lapse movies of a moon rise and my drive to work. But after swatting mosquitoes and driving distracted, I realized I needed to automate.

Initial Investigations

I reviewed the remote control available for my Panasonic DMC-FZ20, but I wasn't impressed with a switch on a wire. As I searched the Internet for ideas, I found a resistor array schematic that controls the camera through its remote jack on **www.steves-digicams.com** — a good start. I also found information about intervalometer functions on Nikon cameras. An intervalometer signals the camera to take a picture at regular intervals.

A. LCD Display Connector
B. Component Carrier
C. Date/Time Chip
D. Back-up Supply
E. Switch Connector Block
F. Camera Control Connector
G. LED Connector
H. Optoisolators
I. Ground Pin for Switch
J. Resistor for LED

FIGURE 1. Protoboard Layout.



I Googled "intervalometer" and found a few commercially built units. Many were for laboratory use, and a few were for specific cameras. The most intriguing and flexible was the Time Machine (**www.bmumford.com/photo/camctlr.html**). It had great features including motion sensor inputs, and a range of interval setups from simple to complex. However, it did not interface to my camera and was more than I wanted. Something about the Time Machine's package reminded me that I had built a cookie timer with a BASIC Stamp a few years ago.

At the Parallax website, I reviewed the BASIC Stamp, but thought I might require more program space and memory for my project. This led me to consider the SX microcontroller. After I read about other SX projects, I felt that it would be a platform I could use now and in the future! In addition to a free compiler, the price for the prototyping board made it very attractive to start experimenting.

Design Approach

In addition to setting the interval between pictures, I wanted to specify when to start and stop taking pictures. I also wanted to program these functions in the field.

The number of pictures — or frames — per time period varies based on the event. Longer events, such as a change in season, require one frame per day where a blooming flower requires two frames per minute (frames/min). I planned to use the intervalometer for events that last from 24 to 48 hours so I specified the slowest rate as one frame per minute. My camera required some time to recover between pictures, so my highest rate was 20 frames/min.

In some cases, I wanted to start taking pictures at a certain time of day and stop a few hours later. For example, the intervalometer should start triggering pictures of a sunrise at 4 AM and stop at 8 AM. With a timed session, I could set the start time the night before. When I return the next morning, the camera and intervalometer would have captured the sunrise (I called this the Delay mode). In addition, I wanted to be able to start taking pictures immediately and continue for a set period of time (I called this the Now mode).

In order to make my intervalometer field programmable, I selected a four direction switch and an LCD screen. I used the switch to navigate and select options from a menu presented on the screen.

The final design included a Parallax SX-48, a DS1302 Real Time Clock (RTC) with battery back-up, optocouplers, and menu-driven interface presented on a screen. I considered using the SX-48's real time clock counter for timed sessions. This would have required some familiarity with interrupts. Since this was my first project with the SX-48, I decided to use the RTC.

After completing the design, I placed a graphic of the SX-48 Protoboard into a Word document. Using items from the Word Drawing toolbar, I laid out components onto the graphic. When I was done, I was confident they would fit on the Protoboard (see Figure 1).

As a final step before starting, I estimated power requirements. When the project was complete, I measured the current required for the circuit quiescently and in operation. The results are in Figure 2.

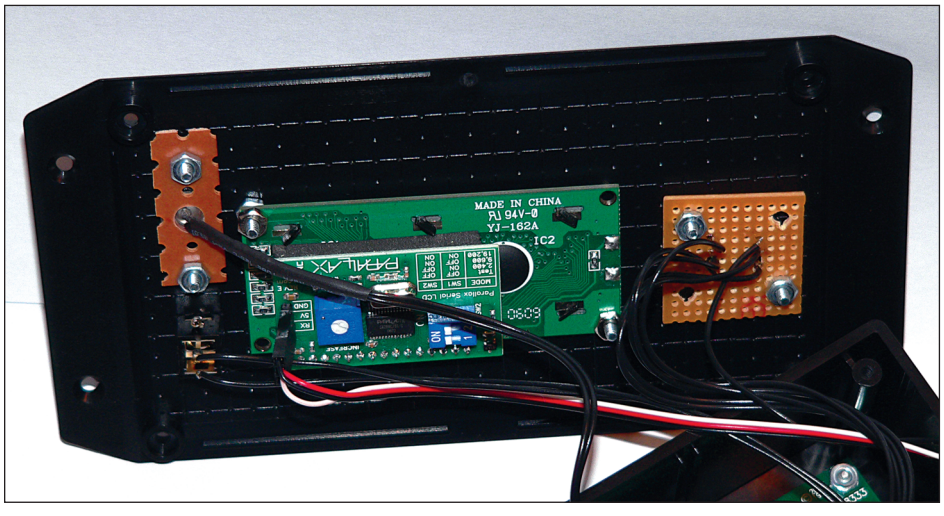
	Budget	Measured
Intervalometer	131 mA	108 mA
Intervalometer (picture triggered)	201 mA	125 mA

■ FIGURE 2. Power Budget. I estimated the power consumption from specifications of the components. At the end of the project, I measured the power consumption.

Circuit Construction

I believe in breaking down a large project into small chunks. For the circuit construction, I installed components on the Protoboard and made a few small programs to help evaluate their operation. The simplest place to start was the LCD. By having a working display, I thought I would also be able to resolve programming issues by displaying information as the program executed.

I soldered a three pin header to the Protoboard so that power and ground pins lined up with the Vdd and Vss bus lines; a third pin would be used to com-



■ FIGURE 3. The back of the front panel shows how the four-direction switch was mounted. When the intervalometer was complete, I thought the switch connections should have had heat shrink tubing to help reduce stress.

Set Up	The Set Up sub menu provides for setting the start and stop time for the Delay mode, the duration in minutes for the Now mode, the frames/min, and setting the time and date of the Real Time Clock. The navigation switch moves the cursor on the screen to indicate what is under edit. It also adjusts the value of the parameter. Pushing the navigation switch exits the edit mode and saves the value.
Run	The Run sub menu provides for automatic or manual operation. Manual operation focuses or triggers the camera from the front panel. Automatic operation starts the Delay or Now mode, or cancels a mode in operation.
Options	The Options sub menu provides for toggling the use of an LED during a session (the LED lights when a frame is triggered), toggling the backlight of the LCD Display, and toggling the Super Cap charging circuit.
Power Down	The Power Down item places the intervalometer in Sleep mode. The power switch must be cycled to display the main menu again.

■ FIGURE 4. Intervalometer Operation. After turning it on, the intervalometer presents the main menu. The menu items and their functionality appear in the table here.

municate with the LCD. After I verified the LCD operation, I connected a microcontroller port pin to the third pin on the pin header, and verified that I could write text to the display. But I discovered that the text started wrapping after 20 characters rather than 16. Parallax Support said I should remove a surface mount jumper from the back of the LCD PC board. With this fix, the display wrapped the text correctly.

With the LCD in place, I wanted to display the time of day using the RTC. I soldered it and its crystal to the Protoboard. I completed the connections from the SX-48 to the RTC, and ported a BASIC Stamp program to S/X Basic. The program initialized and displayed the time on the LCD.

At first, the time did not appear because the RTC was initialized incorrectly. The syntax was correct for the BASIC Stamp but not for the SX-48 (the compiler did not detect the sytax errors). The Parallax online discussion Forum helped me correct the program. I was relieved to see the RTC operating because I thought I had burned it or the crystal when I soldered them to the board!

To complete the RTC development, I soldered a Super Cap to the Protoboard. I added code to my program to operate a charging function through the RTC. I have found that the RTC accurately maintains the time and date for more than a month on just the Super Cap current.

I selected an ALPS four direction switch with center push for menu navigation. Since the small switch had no mounting fixtures, I mounted it to a one inch square piece of Vectorboard. I enlarged some of the holes, and mounted the switch in the center. I drilled a hole in each corner, and used two of them to mount the Vectorboard with the switch on the front panel of my enclosure (see Figure 3).

I verified the switch operation with a program that displayed the closed connection on the LCD. I placed all of the switch connections on a single port of the microcontroller. In this manner, I only have to read the port, mask the reading, and compare the result to constants in order to determine the position of the switch.

Firmware Development

I prefer to develop programs in small steps to both gain confidence in the system and to minimize code defects. When there is a defect, then I have only to look at the most recent changes to correct it. As the program grew in size, I found that locating defects, in some cases, was a little more difficult as more parts of the program interacted.

The main loop of the program monitors the four position switch. The up and down switch positions navigate the menu like a Windows list box. The left and right positions move between menu levels and select menu items, respectively. I used the push button connection to finalize editing of parameters.

I found it challenging to detect a number less than zero. For example, the index of the current menu item is zero and I want to navigate up. The program decrements the index and its value becomes 255. The SX-48 does not recognize the value as -1, and the following SX/BASIC statement did not evaluate to true:

```
IF MenuIndex < 0 THEN
```

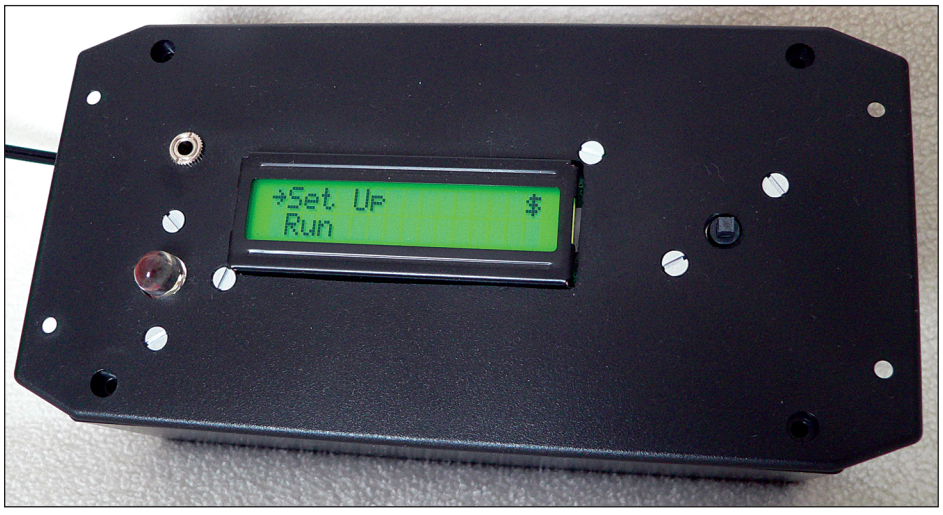
When I needed to detect a negative number, I checked for 255 explicitly. Fortunately, I did not have a case where 255 was a valid value for any of my indexes. Once I recognized this, I was comfortable adopting it as a standard for this project. I would not recommend it for all projects.

As I started adding more functionality, I found odd things happening on the display. The cursor would be in the wrong place or a value would not display correctly. For example, when editing frames per minute, I wanted to write "Frames/min" and "10" on the first line, and clear the second line. The label would appear but the value did not.

When I stepped through the program, I discovered that it wrote the label on the first line, the value on the second line, and then cleared the second line. I was not able to see this under normal execution because it happened so quickly. The debugger was a valuable tool to help resolve this issue.

As the code grew larger, I noticed that the menu text was not displaying correctly. I reviewed the S/X BASIC READ command that helps move the text to the display. The command expects to find menu text on a single page of 256 bytes. My menu data crossed a page boundary, and was more than 256 bytes. To resolve this, I moved the menu text to its own file and gave it a starting point on a page boundary. Also, my menu became simpler (and smaller) which focused the program on basic intervalometer functions.

I completed the programming for editing parameters and started working on the camera interface. In addition to a resistor array, I added a set of optocouplers (one for focus and one to trigger the picture). The optocouplers isolate the power supply of



■ FIGURE 5. The Completed Project.

the camera from the intervalometer.

I proved the interface operation on a breadboard by placing the resistors on a DIP header. On the breadboard, the camera operated correctly. I inserted the DIP header on the Protoboard and connected the optocouplers. Disappointingly, the camera did not react.

I reviewed the interface circuit and the placement of the DIP header on the Protoboard. In that exact spot, the Protoboard had holes shorted by design. This was causing a short in my resistor array. I changed the resistors on the DIP to take advantage of the Protoboard design. With corrections in place, the camera operated as expected.

My final programming task was to place time intervals between frames. The SX-48 on-board timers seemed an obvious choice. My first approach was to monitor the number of timeouts on a timer, and trigger the camera when the timeouts equaled a pre-calculated count. I was starting to run low on code space, so I designed a more efficient solution.

I set up Timer1 in pulse width mode (50% duty) and the other, Timer2, in External Trigger mode. The pulse from Timer1 triggers Timer2. Next, I set the Timer2's compare register to the number of pulses I needed before I wanted a frame triggered. When the number of counts was the same as the compare register, the SX-48 generated an interrupt. The code in my interrupt routine became very simple: reset Timer2 and trigger the camera.

Code Optimizations

I made changes to subroutines which freed up code space and helped me add extra, unexpected functionality. These changes included code reuse, my own PAUSE command, display optimizations, and removing functionality.

I saved code space when I wrote code that could be reused. For example, I needed to set the date and time of the Real Time Chip and for the Delay mode. I wrote the subroutine that sets the date and time generically so that the settings could be used for either. I also reused camera control subroutines for both manual and automatic operation.

SystemPause

The SX/B PAUSE command compiles to a few lines of assembly everywhere it's used. I copied the assembly code of the PAUSE command to a single SX/B subroutine — named SystemPause — and passed in a single argument. The argument set the number of milliseconds to pause. The overhead of calling into and returning from my subroutine was not critical in my application, and I saved code space.

Display Optimization

I needed to indicate the operational status on the display. Custom characters required code space, so it was more economical to use existing characters. Also, I found that I could manipulate the displayed character with masking to indicate different status. For example, the $ character has a binary value of 00100101. By manipulating the byte that stores this value, I could change it from 00100000 (space) to 00100101 by changing two bits. On the display, the status toggles between 'space' and '$.'

Removing Functionality

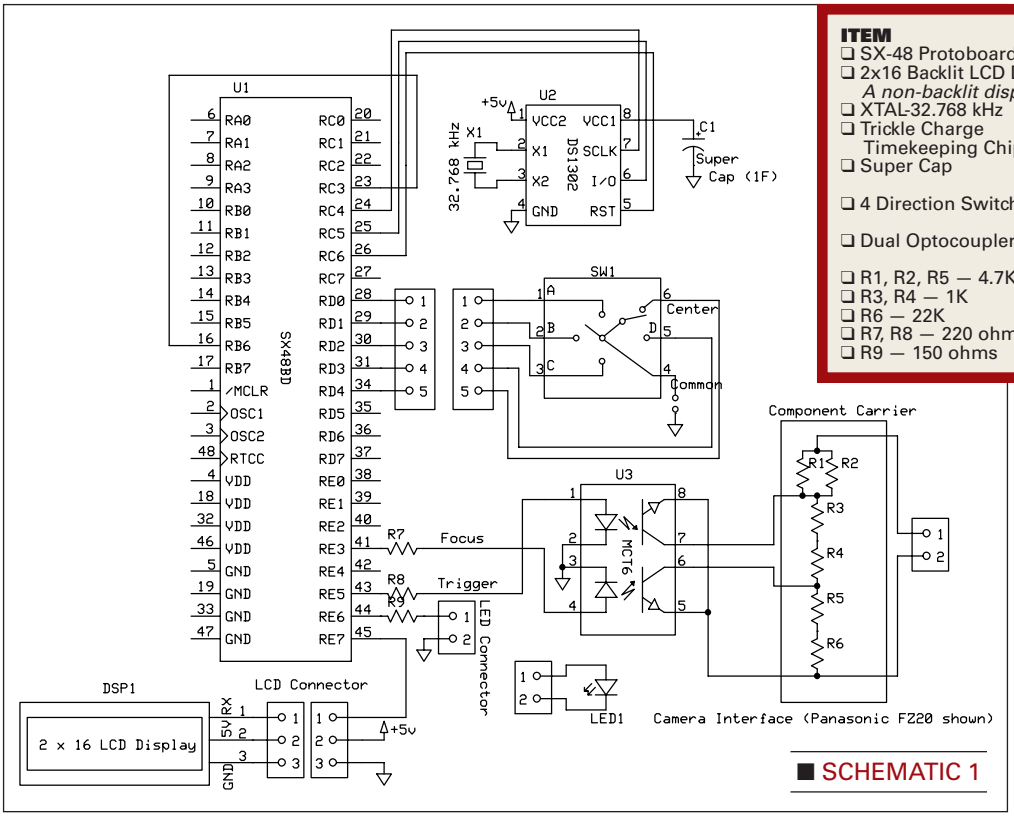
I wrote a subroutine to turn off the display when the intervalometer was waiting to start in Delay mode. When I made current measurements, I found there was very little difference between the display on and off — the LCD still required power. I removed the code and reclaimed the code space for features that made more sense. For example, when setting a date, I didn't check for a valid end of month date. With the extra code space, I checked for things like February 31st and prevented them. Also, I provided for an automatic power-down after two minutes of navigation inactivity.

Testing

Figure 4 describes all of the menu functionality in the intervalometer. I verified most of it on the bench. With the intervalometer complete (see Figure 5), I needed a beta test. Since my "project manager" wanted me to

■ FIGURE 6. Project Alternatives. Here are a few alternatives to the project presented here.

The NE555 Timer	Set up the timer in astable mode. A combination of decade counters (4017) and ripple counters (4020 or 4040) would provide a range of intervals. Make sure the timer pulse width is wide enough for the camera or interface circuit to detect.
The Nikon Coolpix S50C	This model has the intervalometer function built in.
Use your Personal Computer	Most Canon PowerShot models include software which controls the camera through its USB port. The software includes an intervalometer function.



■ SCHEMATIC 1

ITEM	SUPPLIER/PART NO.
❑ SX-48 Protoboard	Parallax/45300
❑ 2x16 Backlit LCD Display	Parallax/27977
A non-backlit display will work just as well.	
❑ XTAL-32.768 kHz	Parallax/251-03230
❑ Trickle Charge Timekeeping Chip	Dallas Semiconductor/DS1302; Parallax/604-00005
❑ Super Cap	Elna/DB-5R5D105; Mouser/555-1.0Z5.5
❑ 4 Direction Switch	ALPS: SKQUCAA010; Mouser/688-SKQUCA
❑ Dual Optocoupler	Fairchild Semiconductor/MCT6; Mouser/512-MCT6
❑ R1, R2, R5 — 4.7K	
❑ R3, R4 — 1K	
❑ R6 — 22K	
❑ R7, R8 — 220 ohms	
❑ R9 — 150 ohms	

paint the kitchen, I set up my camera to film it. I used the Now mode of the intervalometer. In this mode, the intervalometer starts triggering pictures immediately, and continues for four hours. After an hour, a defect caused the triggering to stop.

After debugging through the code, I found that I was using a variable twice which affected the calculation for the Now mode. After making a correction, the intervalometer operated the entire four hours.

I discovered one last defect when setting the date and time. It was difficult to track down because I was using an index to write to memory. The index was one greater than it should have been, and this affected how information was displayed. When I corrected the index, the defect was resolved.

Going Farther

Instead of using elapsed time to trigger frames, the intervalometer could be modified to accept motion sensor information. In this manner, the intervalometer waits until the sensor detects motion, and then takes a picture. This could be used for photographing wildlife in remote places.

Many cameras have a USB port which provides for remote operation. The intervalometer could be modified to support a USB interface with the addition of commercially available integrated circuits. While this would involve hardware and firmware changes, the intervalometer would support a wider range of cameras. **NV**