In Lab02 the goal was to create a system that toggled the light at 5 Hz. This means the red LED should be on for 0.1 sec and off for 0.1 sec. If we look at the LED with our eyes it looks like it is running correctly. If we look at the signal on the oscilloscope or logic analyzer, again it looks correct. But can we prove it? In this first debugging process we will dump the value of PF1 in one array and the time difference in a second array (Program 9.9). When we run this program, it reveals all 49 measurements of time difference are 1,599,996 bus cycles, which is 0.09999975 seconds, which is close to the desired time of 0.1 second.

```
// first data point is wrong, the other 49 will be correct
unsigned long Time[50];
unsigned long Data[50];
int main(void){  unsigned long i,last,now;
                  // initialize PF0 and PF4 and make them inputs
  PortF_Init();   // make PF3-1 out (PF3-1 built-in LEDs)
  SysTick_Init(); // initialize SysTick, runs at 16 MHz
  i = 0;          // array index
  last = NVIC_ST_CURRENT_R;
  while(1){
    Led = GPIO_PORTF_DATA_R;   // read previous
    Led = Led^0x02;            // toggle red LED
    GPIO_PORTF_DATA_R = Led;   // output
    if(i<50){
      now = NVIC_ST_CURRENT_R;
      Time[i] = (last-now)&0x00FFFFFF;  // 24-bit time difference
      Data[i] = GPIO_PORTF_DATA_R&0x02; // record PF1
      last = now;
      i++;
    }
    Delay();
  }
}
```

*Program 9.9. Instrumentation to record the first 49 time differences (debugging shown in bold).*

However compelling this data is, it doesn't prove it is always 0.1 sec. Let's further specify the desire is to create a system that is accurate to ±0.01%. This means any time difference 1,600,000±160 cycles is acceptable. In Program 9.10 we will count the number of times the time difference is unacceptable. If we run this program for a month we can observe its behavior over 25 million times. Furthermore, we can leave this debugging code into the deployed system, and verify the system is running as expected for the entire life of the system.

```
// first data point is wrong, the others will be correct
long Errors;
#define CORRECT 1600000
#define TOLERANCE 160
int main(void){  unsigned long last,now,diff;
                  // initialize PF0 and PF4 and make them inputs
  PortF_Init();   // make PF3-1 out (PF3-1 built-in LEDs)
  SysTick_Init(); // initialize SysTick, runs at 16 MHz
  Errors = -1;    // no errors (ignore first measurement)
  last = NVIC_ST_CURRENT_R;
  while(1){
    Led = GPIO_PORTF_DATA_R;   // read previous
    Led = Led^0x02;            // toggle red LED
    GPIO_PORTF_DATA_R = Led;   // output
    now = NVIC_ST_CURRENT_R;
    diff = (last-now)&0x00FFFFFF;  // 24-bit time difference
    if((diff<(CORRECT-TOLERANCE))||(diff>(CORRECT+TOLERANCE)){
      Error++;
    }
    last = now;
    Delay();
  }
}
```
*Program 9.10. Instrumentation to count the number of mistakes (debugging shown in bold).*

A **black box recorder** stores strategic debugging information in permanent memory so that if an accident were to occur, we could recover the debugging information to see why the accident occurred. We could make a black box recorder out of the programs in this section by storing the data in ROM. The flash ROM allows the running program to change its value. The disadvantage of storing data in ROM is it takes over 1ms to cause the change. In situations where we have infrequent but important data, this 1ms overhead is not significant. The advantage of storing infrequent but important debugging information in ROM is that this data is available even if power is removed and restored. For example, if the embedded system were to be involved in a loss of life accident, the data stored in ROM could be recovered to determine if any run-time errors might have contributed to the accident. Conversely, this data stored in ROM could verify that no errors in the operation of the embedded system had occurred prior to the accident. If you wish to write to flash ROM, look at example projects called “flash” on **http://users.ece.utexas.edu/~valvano/arm/** (http://users.ece.utexas.edu /%7Evalvano/arm/)

Help