Performance Debugging

Performance debugging involves the verification of the timing behavior of our system. Performance debugging is a dynamic process where the system is run and the dynamic behavior of the input/outputs is compared against the expected results. Two methods of are presented, and then the techniques are applied to measure execution speed.

Measurements were performed with inputs 100 and 230400. The C program at optimization level 3 and "optimize level 0 for time" took 178 to 198 cycles. Interestingly, the **sqrt** routine in "math.h" took about 1840 cycles to execute.

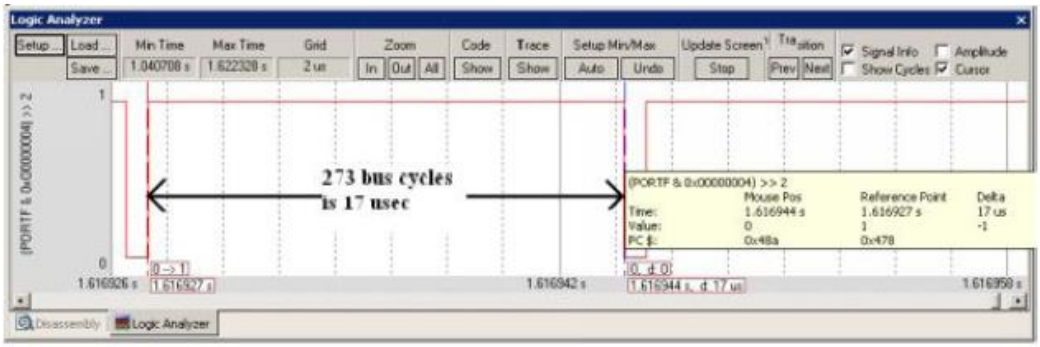
SysTick is a 24-bit counter, decremented at the bus clock frequency. It automatically rolls over when it gets to 0. If we are sure the execution speed of our function is less than (224 bus cycles), we can use this timer to collect timing information with only a modest amount of intrusiveness. We initialize the timer by calling **SysTick_Init** as shown in Program 4.7.

Keep in mind the fact that although the registers and variables are 32 bits, but the data values are only 24 bits. To perform a measurement we first read **NVIC_ST_CURRENT_R** , execute some software, and then read **NVIC_ST_CURRENT_R** again. The difference represents the elapsed time for the executing software. In this example **before** and **elapsed** are 32-bit unsigned variables:

```
before = NVIC_ST_CURRENT_R;
// software we wish to test
elapsed = (before-NVIC_ST_CURRENT_R)&0x00FFFFFF;
void main(void){ uint32_t before,elapsed,ss,tt;
SysTick_Init(); // initialize, Program 4.7
ss = 230400;
//GPIO_PORTF_DATA_R= 0x02;     // turn on led LED
            before = NVIC_ST_CURRENT_R;
            tt = sqrt(ss);
//GPIO_PORTF_DATA_R = 0x00; // turn off led LED
  elapsed = (before-NVIC_ST_CURRENT_R)&0x00FFFFFF;while(1){};
}
```

Another technique is used to measure the execution time of the function.

The third technique can be used in situations where SysTick is unavailable or where the execution time might be larger than 224 cycles. In this empirical technique we attach PF1 output pin to an oscilloscope or to a logic analyzer. We will set the pin high before the call to the function and set the pin low after the function call. Use logic analyzer to observe pin PF2. In this way a pulse is created on the digital output with duration equal to the execution time of the function.



Dump into Array without Filtering

If we add print statements to it that require more than 1 ms to perform, the presence of the print statements will cause the system to crash. In this situation, the print statements would be considered extremely intrusive. Another problem with print statements occurs when the system is using the same output hardware for its normal operation, as is required to perform the print function. we can add a debugger instrument that dumps strategic information into an array at run time. We can then observe the contents of the array at a later time.

The first step when instrumenting a dump is to define a buffer in RAM to save the debugging measurements.

Assume Port F bit 2 is available and connected to the scope. By placing the function call in a loop, the scope can be triggered. With a storage scope or logic analyzer, the function need be called only once.

```
unsigned long i;
unsigned long Sqrt[50];
unsigned long Data[50];
unsigned long Led;
        i = 0;              // array index
        ss=10;
   while(1){
        Led = GPIO_PORTF_DATA_R;     // read previous
       Led = Led^0x04;                // toggle red LED
       GPIO_PORTF_DATA_R = Led;     // output GPIO_PORTF_DATA_R;
        tt = sqrt(ss);
if(i<50){
  Sqrt[i] = sqrt(ss);   // sqrt
           Data[i] = GPIO_PORTF_DATA_R&0x04; // record PF2
      i++;
       ss=ss*2;
       }
       SysTick_Wait10ms(100);                          // wait 1s
```

实验步骤：

1. 运行附件工程文件”Performance”，先选择不同优化级别，运行程序，观察elapsed记录的对100和230400两个数求平方根的执行时间。
2. 把注释语句对PF1开灯和关灯语句取代用定时器测量的elapsed语句，用逻辑分析仪观察PF1波形，计算函数运算时间（仿真模式运行程序）。
3. 板级运行程序，观察pf2灯的亮灭、观察存入内存记录的PF2状态值，以及tt所求的ss平方根值。