# 1. 实验题目

- **Performance Debugging**

# 2. 实验内容

- **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**
- **用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**
- **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**

# 3. 实验原理

**系统计时器(*TI官方文档*)**

Cortex-M4 includes an integrated system timer, SysTick, which provides a simple, 24-bit clear-on-write, decrementing, wrap-on-zero counter with a flexible control mechanism.

The timer consists of three registers:

- SysTick Control and Status (STCTRL): A control and status counter to configure its clock, enable the counter, enable the SysTick interrupt, and determine counter status.
- SysTick Reload Value (STRELOAD): The reload value for the counter, used to provide the counter's wrap value.
- SysTick Current Value (STCURRENT): The current value of the counter.

When enabled, the timer counts down on each clock from the reload value to zero, reloads (wraps) to the value in the STRELOAD register on the next clock edge, then decrements on subsequent clocks. Clearing the STRELOAD register disables the counter on the next wrap. When the counter reaches zero, the COUNT status bit is set. The COUNT bit clears on reads.

Writing to the STCURRENT register clears the register and the COUNT status bit. The write does not trigger the SysTick exception logic. On a read, the current value is the value of the register at the time the register is accessed.

The SysTick counter reload and current value are undefined at reset; the correct initialization sequence for the SysTick counter is:

1. Program the value in the STRELOAD register.
2. Clear the STCURRENT register by writing to it with any value.
3. Configure the STCTRL register for the required operation.

**Keil中的优化等级**

- O0  applies minimum optimizations.  Most optimizations are switched off, and the code generated has the best debug view.
- O1  applies restricted optimization.  For example, unused inline functions and unused static functions are removed. At this level of optimization, the compiler also applies automatic optimizations such as

removing redundant code and re -ordering instructions s o as to avoid an interlock situation. The code generated is reasonably optimized, with a good debug view.

- O2 applies high optimization (This is the default setting). Optimizations applied at this level take advantage of ARM's in-depth knowledge of the processor architecture, to exploit processor -specific behavio r of the given target. It generates well optimized code, but with limited debug view.
- O3 applies th e most aggressive optimization. The optimization is in accordance with the user's –Ospace/- Otime choice . By default, multi - file compilation is enabled, which leads to a longer compile time, but gives the highest levels of optimization.

# 4. 实验过程

## 代码理解

```
void SysTick_Init(void){
  NVIC_ST_CTRL_R = 0;               // disable SysTick during setup
  NVIC_ST_RELOAD_R = 0x00FFFFFF;    // maximum reload value
  NVIC_ST_CURRENT_R = 0;            // any write to current clears it
  NVIC_ST_CTRL_R = 0x05;            // enable SysTick with core clock

}
//test code
// The delay parameter is in units of the 80 MHz core clock(12.5 ns)
void SysTick_Wait(uint32_t delay){
   NVIC_ST_RELOAD_R = delay-1;  // number of counts
  NVIC_ST_CURRENT_R = 0;       // any value written to CURRENT clears
  while((NVIC_ST_CTRL_R&0x00010000)==0){ // wait for flag
    }
}
// Call this routine to wait for delay*10ms
void SysTick_Wait10ms(uint32_t delay){
 unsigned long i;
 for(i=0; i<delay; i++){
   SysTick_Wait(800000); // wait 10ms
    }
}
```

此部分为上述原理中系统计时器的使用，开始是初始化系统寄存器，之后使用计时器进行延迟的代码。

```
  PLL_Init();                       // bus clock at 80 MHz
  SYSCTL_RCGCGPIO_R |= 0x20;        // activate port F
  while((SYSCTL_PRGPIO_R&0x20)==0){};
  GPIO_PORTF_DIR_R |= 0x0E;         // make PF3-1 output (PF3-1 built-in LEDs)
  GPIO_PORTF_AFSEL_R &= ~0x0E;      // disable alt funct on PF3-1
  GPIO_PORTF_DEN_R |= 0x0E;         // enable digital I/O on PF3-1
                                    // configure PF3-1 as GPIO
  GPIO_PORTF_PCTL_R = (GPIO_PORTF_PCTL_R&0xFFFF000F)+0x00000000;
  GPIO_PORTF_AMSEL_R = 0;           // disable analog functionality on PF

  SysTick_Init(); // initialize SysTick timer, see SysTick.c
```

此部分进行了3个初始化，首先是80MHz时钟给计时器使用，之后是PF3-1接口的初始化，用于控制LED，之后是系统计时器的初始化。

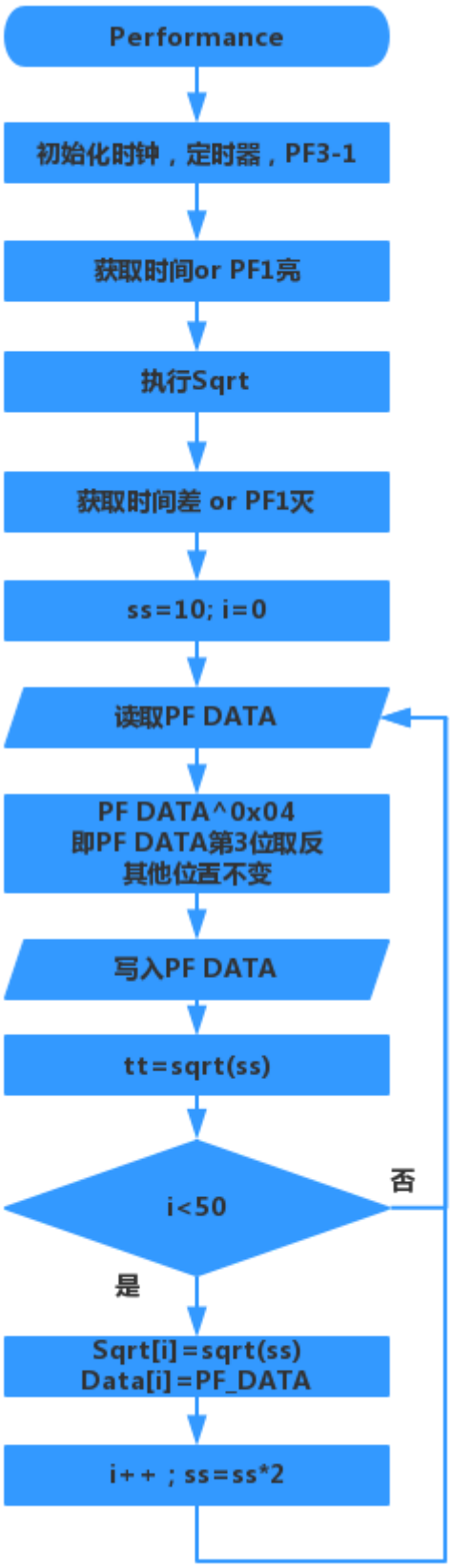
```
ss = 230400;
//GPIO_PORTF_DATA_R= 0x02;   // turn on led LED
  before = NVIC_ST_CURRENT_R;
tt = sqrt(ss);
  //GPIO_PORTF_DATA_R = 0x00; // turn off led LED
elapsed = (before-NVIC_ST_CURRENT_R)&0x00FFFFFF;
```

此部分为计算开方执行的时间，分别使用了两种方式：1.使用计时器进行时间计算，此处一点因为计时器只有24bit,所以最后结果只有24bit有用,此处需要&0x00FFFFFF

```
  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
  }
```

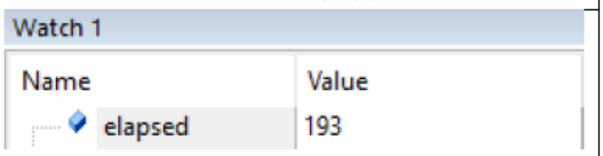
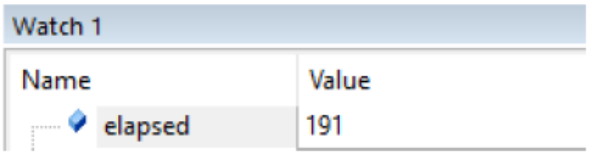
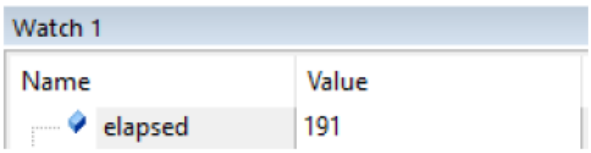
此处实现的是记录50次开方的结果，同时使led灯1秒亮灭转换一次。

## 程序流程

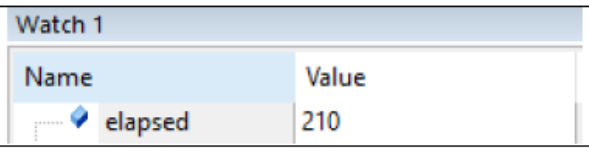
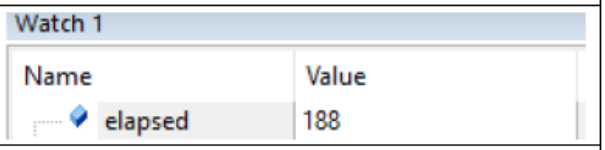
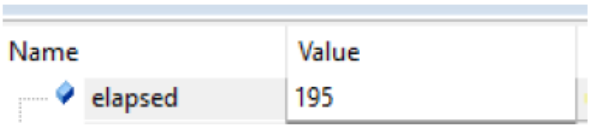
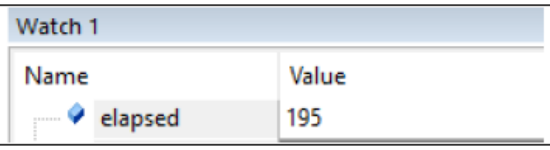



**1.择不同优化级别，运行程序，观察elapsed记录的对100和230400两个数求平方根的执行时间**

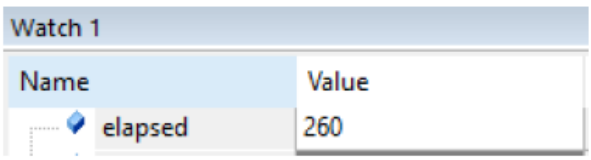
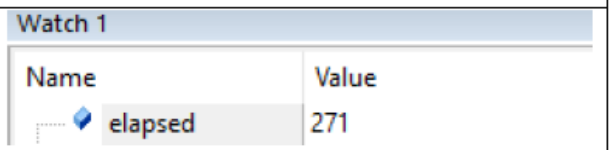
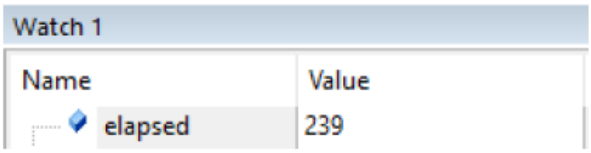
- 100

| O0，100 开方 | O1，100 开方 |
| --- | --- |
|  |  |
| **O2，100 开方** | **O3，100 开方** |
|  |  |

- 230400

| O0，230400 开方 | O1，230400 开方 |
| --- | --- |
|  |  |
| **O2，230400 开方** | **O2，230400 开方** |
|  |  |

- Newton's method：100

| O0，100 开方 | O1，100 开方 |
| --- | --- |
|  |  |
| **O2，100 开方** | **O3，100 开方** |
|  |  |

- Newton's method：230400

| O0，230400 开方 | O1，230400 开方 |
| --- | --- |
| Watch 1<br>Name: elapsed<br>Value: 292 | Watch 1<br>Name: elapsed<br>Value: 272 |
| **O2，230400 开方** | **O3，230400 开方** |
| Watch 1<br>Name: elapsed<br>Value: 271 | Watch 1<br>Name: elapsed<br>Value: 241 |

- **分析**

  此时我们看到，如果优化等级越高对应的执行时间也就越短，因为在不同等级，对应的O0-3,每一级都比上一级的优化有提升，但是总体来说还是Newton's method快。

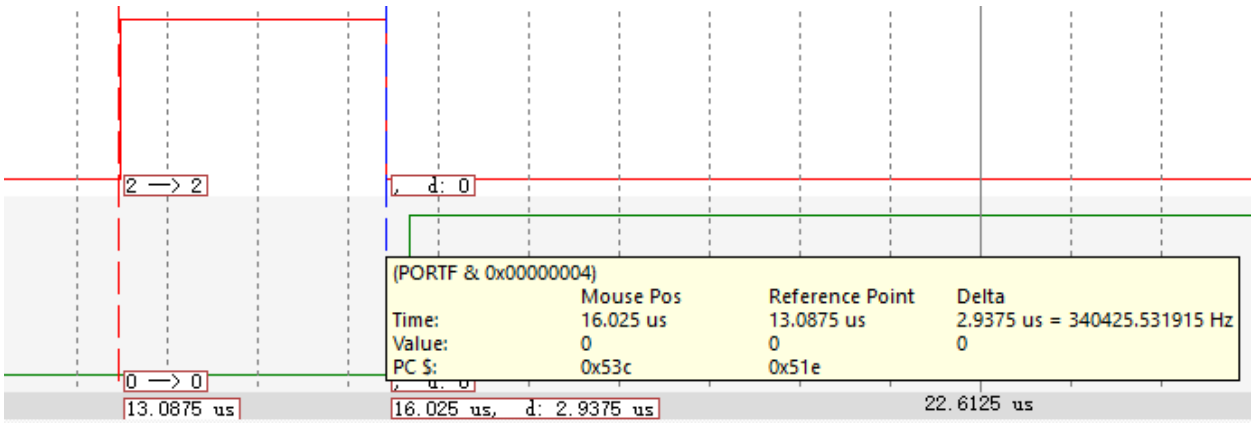
## 用PF1开灯和关灯语句取代用定时器测量的elapsed语句，用逻辑分析仪观察PF1波形，计算函数运算时间

- **代码修改为**

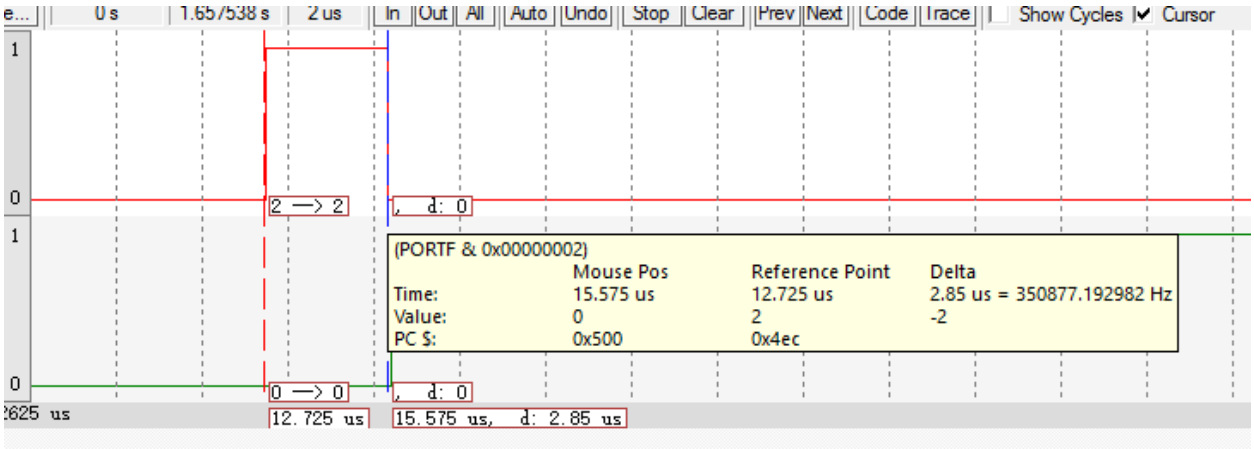
```
SysTick_Init(); // initialize SysTick timer, see SysTick.c
ss = 230400;
GPIO_PORTF_DATA_R= 0x02;   // turn on led LED
      //before = NVIC_ST_CURRENT_R;
tt = sqrt(ss);
  GPIO_PORTF_DATA_R = 0x00; // turn off led LED
//elapsed = (before-NVIC_ST_CURRENT_R)&0x00FFFFFF;
```

- **运行效果**

  O0优化时函数运行时间如下：




  O3优化时函数运行时间如下：



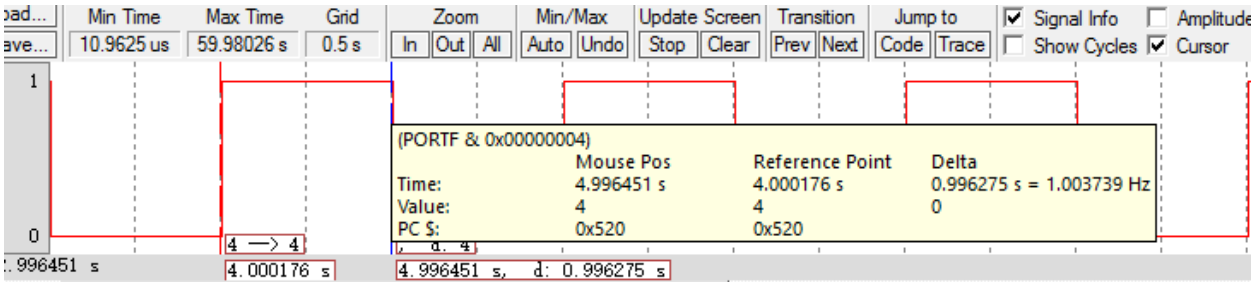

- **分析**

  我们可以通过逻辑分析以直接读出开方所需的时间，通过对于优化的对比我们也能看到此时不同级别的优化还是有所差距的。

  此时我们O3使用的时间是2.85us，而使用O0优化的使用了2.9375us。

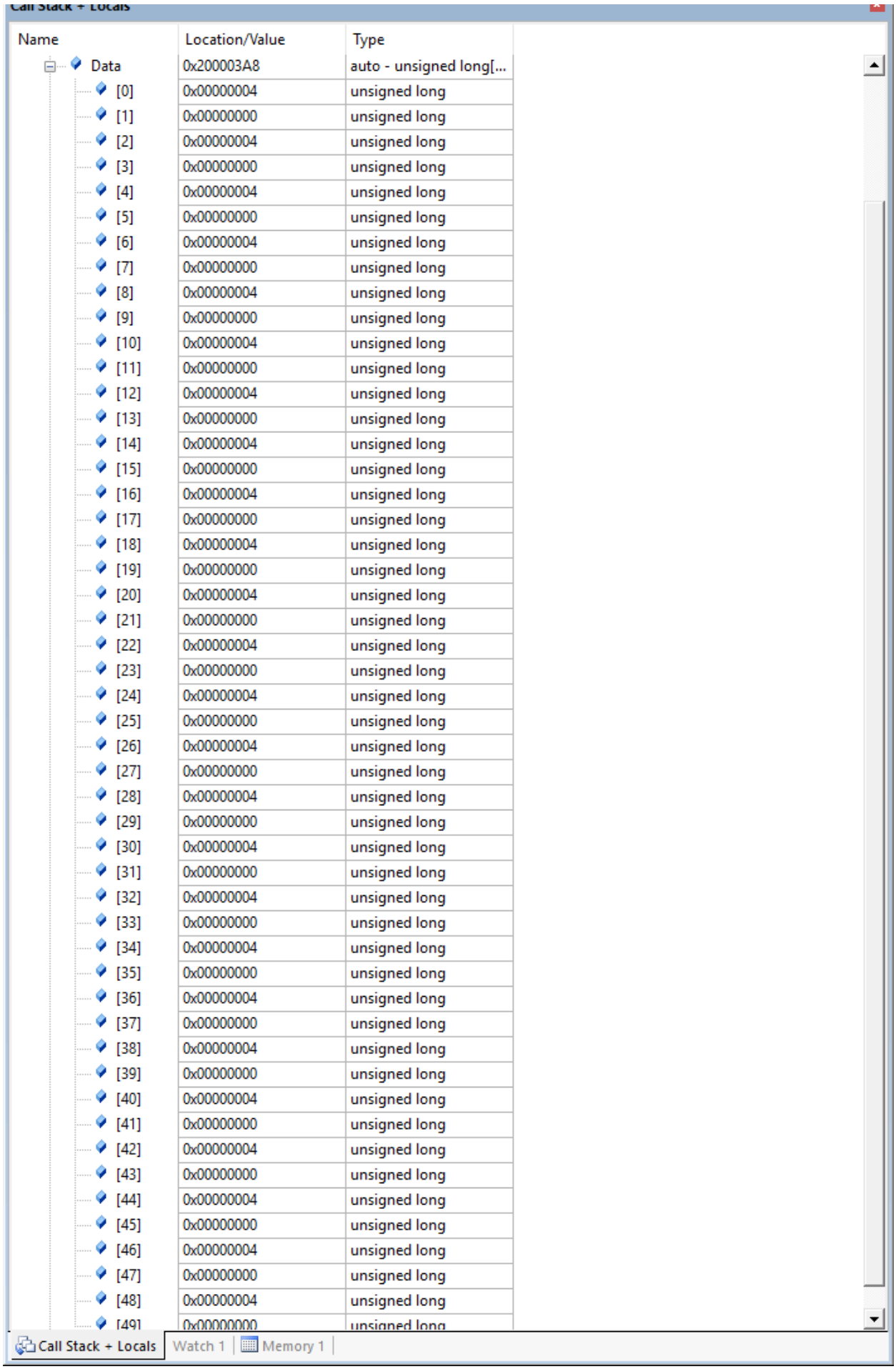
## 板级运行程序，观察pf2灯的亮灭、观察存入内存记录的PF2状态值，以及tt所求的ss平方根值

- **PF2灯的亮灭**




  此时这个亮灭过程所花的时间近似1s,由于开发板的80MHz不是准确的80,000,000Hz,所以此时的偏差可以理解。

- **存入的FP2值**

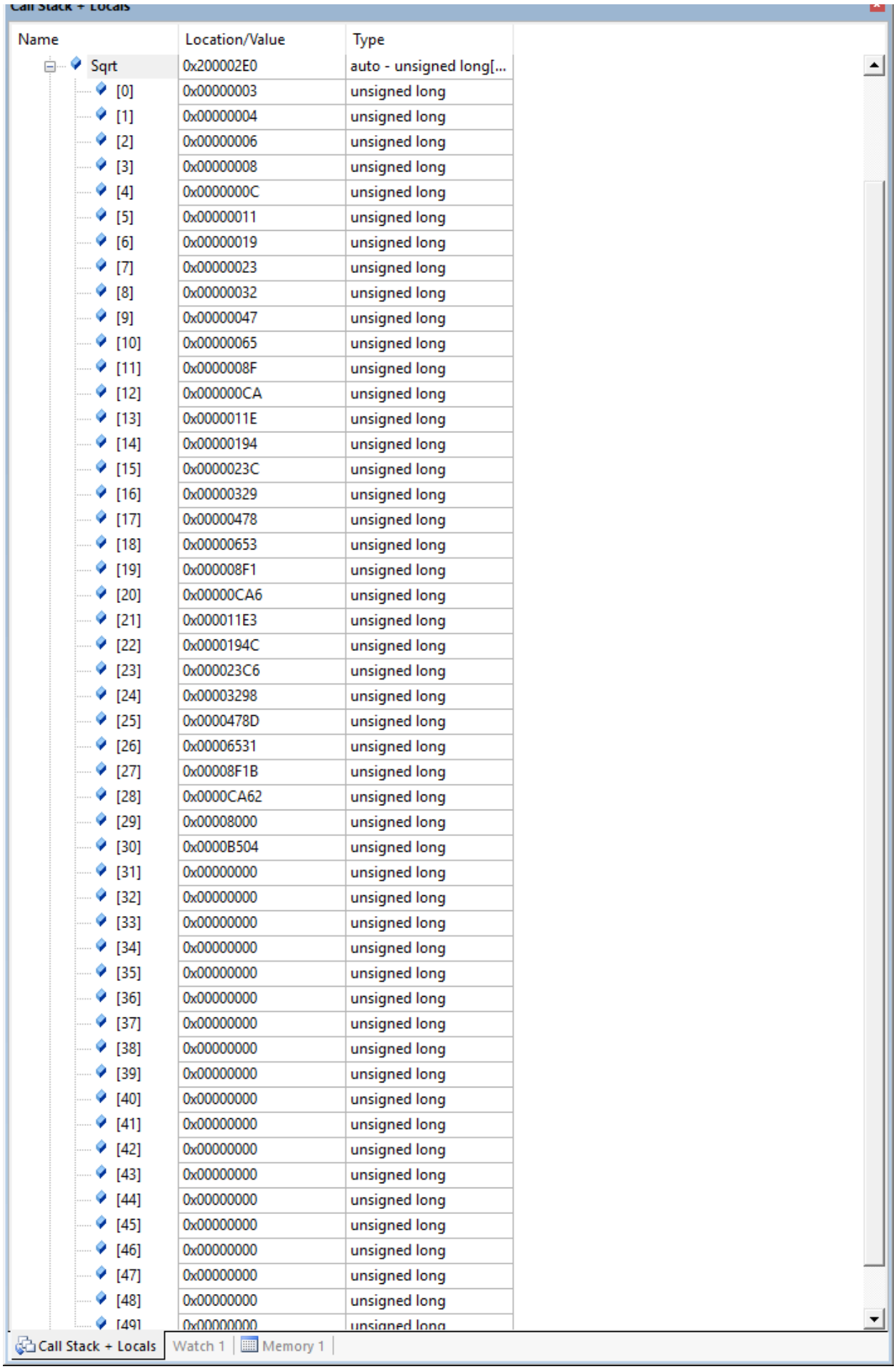

Call Stack + Locals
Name
Location/Value
Type
Data
0x200003A8
auto - unsigned long[...
[0] 0x00000004 unsigned long
[1] 0x00000000 unsigned long
[2] 0x00000004 unsigned long
[3] 0x00000000 unsigned long
[4] 0x00000004 unsigned long
[5] 0x00000000 unsigned long
[6] 0x00000004 unsigned long
[7] 0x00000000 unsigned long
[8] 0x00000004 unsigned long
[9] 0x00000000 unsigned long
[10] 0x00000004 unsigned long
[11] 0x00000000 unsigned long
[12] 0x00000004 unsigned long
[13] 0x00000000 unsigned long
[14] 0x00000004 unsigned long
[15] 0x00000000 unsigned long
[16] 0x00000004 unsigned long
[17] 0x00000000 unsigned long
[18] 0x00000004 unsigned long
[19] 0x00000000 unsigned long
[20] 0x00000004 unsigned long
[21] 0x00000000 unsigned long
[22] 0x00000004 unsigned long
[23] 0x00000000 unsigned long
[24] 0x00000004 unsigned long
[25] 0x00000000 unsigned long
[26] 0x00000004 unsigned long
[27] 0x00000000 unsigned long
[28] 0x00000004 unsigned long
[29] 0x00000000 unsigned long
[30] 0x00000004 unsigned long
[31] 0x00000000 unsigned long
[32] 0x00000004 unsigned long
[33] 0x00000000 unsigned long
[34] 0x00000004 unsigned long
[35] 0x00000000 unsigned long
[36] 0x00000004 unsigned long
[37] 0x00000000 unsigned long
[38] 0x00000004 unsigned long
[39] 0x00000000 unsigned long
[40] 0x00000004 unsigned long
[41] 0x00000000 unsigned long
[42] 0x00000004 unsigned long
[43] 0x00000000 unsigned long
[44] 0x00000004 unsigned long
[45] 0x00000000 unsigned long
[46] 0x00000004 unsigned long
[47] 0x00000000 unsigned long
[48] 0x00000004 unsigned long
[49] 0x00000000 unsigned long
Call Stack + Locals
Watch 1
Memory 1

此时我们因为代码中每次都读入整个Port F DATA的值然后异或0x04,此时只会出现两种情况也就是Port F DATA的值在0000,0100和0000,0000之间变化，所以我们存入的值就是0x00000004和0x00000000。

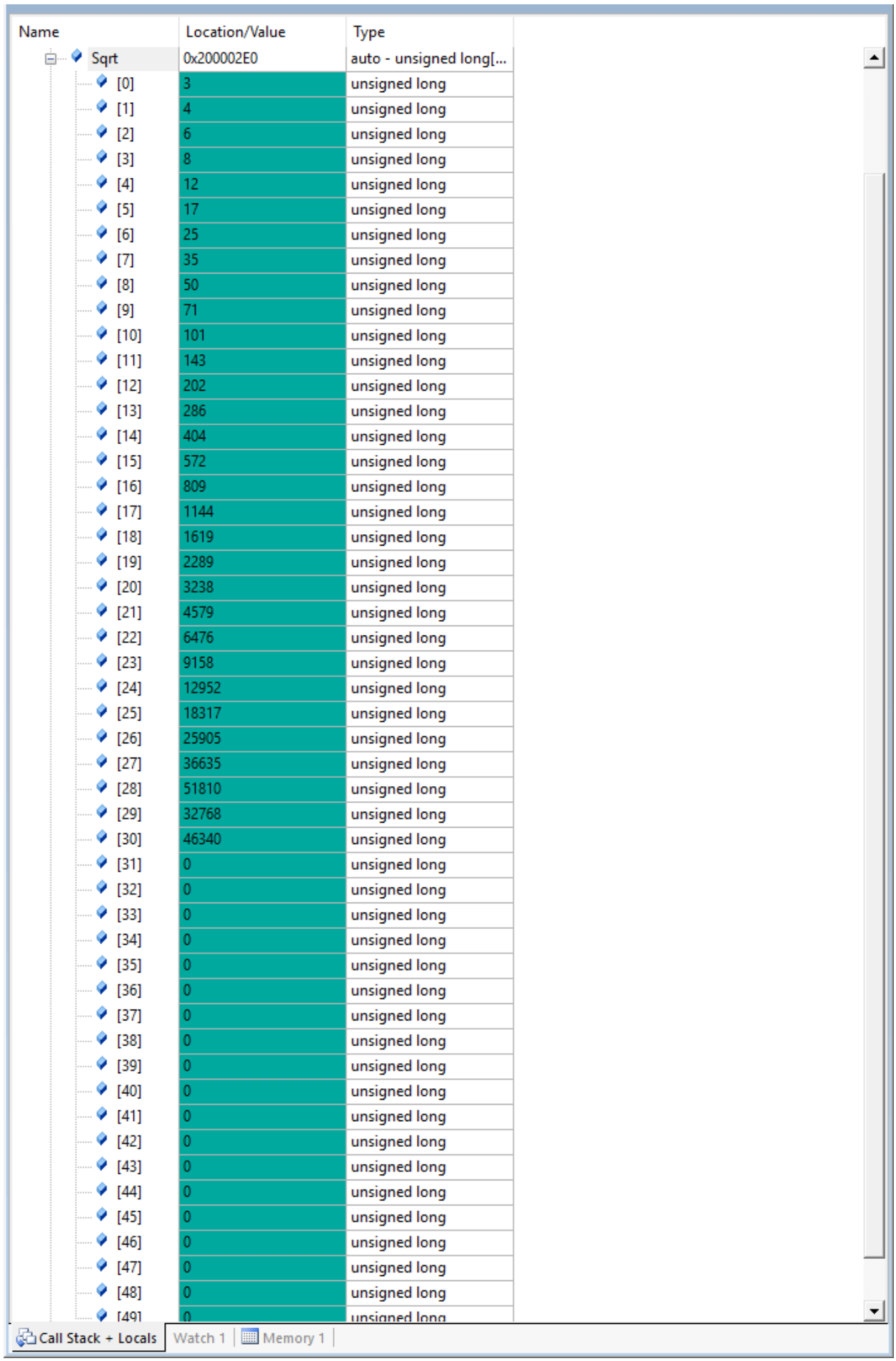
因为PF2位于第三位，所以这个值表现的就是PF2的0,1变化，也就是我们看到灯会闪烁的原因。

- **ss平方根值**

  tt所求的ss平方根的值存在Sqrt中，所以我们之间看Sqrt的值


Call Stack + Locals
Name
Location/Value
Type
Sqrt 0x200002E0 auto - unsigned long[...
[0] 0x00000003 unsigned long
[1] 0x00000004 unsigned long
[2] 0x00000006 unsigned long
[3] 0x00000008 unsigned long
[4] 0x0000000C unsigned long
[5] 0x00000011 unsigned long
[6] 0x00000019 unsigned long
[7] 0x00000023 unsigned long
[8] 0x00000032 unsigned long
[9] 0x00000047 unsigned long
[10] 0x00000065 unsigned long
[11] 0x0000008F unsigned long
[12] 0x000000CA unsigned long
[13] 0x0000011E unsigned long
[14] 0x00000194 unsigned long
[15] 0x0000023C unsigned long
[16] 0x00000329 unsigned long
[17] 0x00000478 unsigned long
[18] 0x00000653 unsigned long
[19] 0x000008F1 unsigned long
[20] 0x00000CA6 unsigned long
[21] 0x000011E3 unsigned long
[22] 0x0000194C unsigned long
[23] 0x000023C6 unsigned long
[24] 0x00003298 unsigned long
[25] 0x0000478D unsigned long
[26] 0x00006531 unsigned long
[27] 0x00008F1B unsigned long
[28] 0x0000CA62 unsigned long
[29] 0x00008000 unsigned long
[30] 0x0000B504 unsigned long
[31] 0x00000000 unsigned long
[32] 0x00000000 unsigned long
[33] 0x00000000 unsigned long
[34] 0x00000000 unsigned long
[35] 0x00000000 unsigned long
[36] 0x00000000 unsigned long
[37] 0x00000000 unsigned long
[38] 0x00000000 unsigned long
[39] 0x00000000 unsigned long
[40] 0x00000000 unsigned long
[41] 0x00000000 unsigned long
[42] 0x00000000 unsigned long
[43] 0x00000000 unsigned long
[44] 0x00000000 unsigned long
[45] 0x00000000 unsigned long
[46] 0x00000000 unsigned long
[47] 0x00000000 unsigned long
[48] 0x00000000 unsigned long
[49] 0x00000000 unsigned long
Call Stack + Locals
Watch 1
Memory 1

换一下显示方式



| Name | Location/Value | Type |
|---|---|---|
| Sqrt | 0x200002E0 | auto - unsigned long[... |
| [0] | 3 | unsigned long |
| [1] | 4 | unsigned long |
| [2] | 6 | unsigned long |
| [3] | 8 | unsigned long |
| [4] | 12 | unsigned long |
| [5] | 17 | unsigned long |
| [6] | 25 | unsigned long |
| [7] | 35 | unsigned long |
| [8] | 50 | unsigned long |
| [9] | 71 | unsigned long |
| [10] | 101 | unsigned long |
| [11] | 143 | unsigned long |
| [12] | 202 | unsigned long |
| [13] | 286 | unsigned long |
| [14] | 404 | unsigned long |
| [15] | 572 | unsigned long |
| [16] | 809 | unsigned long |
| [17] | 1144 | unsigned long |
| [18] | 1619 | unsigned long |
| [19] | 2289 | unsigned long |
| [20] | 3238 | unsigned long |
| [21] | 4579 | unsigned long |
| [22] | 6476 | unsigned long |
| [23] | 9158 | unsigned long |
| [24] | 12952 | unsigned long |
| [25] | 18317 | unsigned long |
| [26] | 25905 | unsigned long |
| [27] | 36635 | unsigned long |
| [28] | 51810 | unsigned long |
| [29] | 32768 | unsigned long |
| [30] | 46340 | unsigned long |
| [31] | 0 | unsigned long |
| [32] | 0 | unsigned long |
| [33] | 0 | unsigned long |
| [34] | 0 | unsigned long |
| [35] | 0 | unsigned long |
| [36] | 0 | unsigned long |
| [37] | 0 | unsigned long |
| [38] | 0 | unsigned long |
| [39] | 0 | unsigned long |
| [40] | 0 | unsigned long |
| [41] | 0 | unsigned long |
| [42] | 0 | unsigned long |
| [43] | 0 | unsigned long |
| [44] | 0 | unsigned long |
| [45] | 0 | unsigned long |
| [46] | 0 | unsigned long |
| [47] | 0 | unsigned long |
| [48] | 0 | unsigned long |
| [49] | 0 | unsigned long |

此时29处变为32768，是因为28处$10 \times 2^{28} = 2684354560$，此时在乘2为 5368709120，已经超过了$2^{32} - 1 = 4294967295$超过了unsigned long的表示范围，所以炸了，此时ss值二进制为1 0100 0000 0000 0000 0000 0000 0000 0000取前32位，所以得到的ss值为1073741824，再开方得到32768，所以29为32768，此时$ss = ss \times 2 = 2147483648$，开方为46340，所以30为46340。此时$ss = ss \times 2 = 4294967296$，二进制为1 0000 0000 0000 0000 0000 0000 0000 0000，再次超过32位，取前32位，得到0，所以31为0，因为此时ss=0，之后乘上0都不会有变化，所以之后的值全部为0。

所以i=30之后tt的值一直为0。

# 5. 实验心得

这次实验没有什么代码需要分析的，都是一些流程，主要还是学一下优化选项，如何使用计时器，以及如何观察衡量代码的运行时间，最后的地方才需要分析代码的运行结果，这个结果也是简单的数学计算，没有什么困难的，主要就是一些之前的C的内容。