

# 嵌入式系统导论实验报告

姓名	学号	班级	电话	邮箱
曹广杰	15352015	1501	13727022190	<a href="mailto:1553118845@qq.com">1553118845@qq.com</a>

第14周，实验10——Performance debugging

## 计算优化时间

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。

优化级别有4个选项，从0到3，在计算分别对100和230400进行开方的运算下得到的计算时间如下：

优化程度	被开方数 <b>100</b> 运算时间	被开方数 <b>230400</b> 运算时间
0	200	210
1	193	195
2	191	195
3	191	195

从以上优化情况对应的运行结果可以看出，对编译操作的适当优化可以缩短计算的时间。随着优化程度的提升，运算的时间也对应地缩短。

实验结果截图见附录。

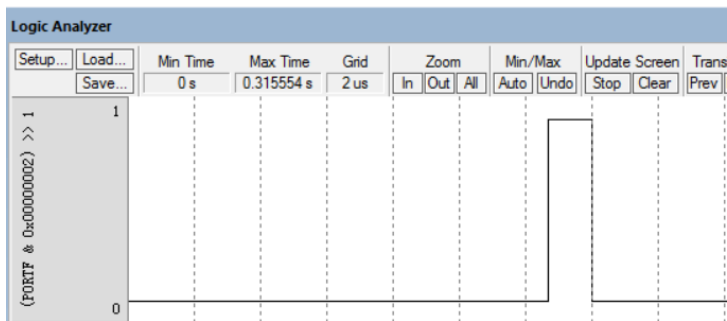
## 观察PF1的波形

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）。

把上文中使用的计时语句注释掉之后，换用对PF1的控制代码，得到实际操作如下：

```
1 GPIO_PORTF_DATA_R= 0x02;    // turn on led LED
2 tt = sqrt(ss);
3 GPIO_PORTF_DATA_R = 0x00; // turn off led LED
```

在计算开方前后进行输出出口的控制操作，此时得到的输出波形如下：



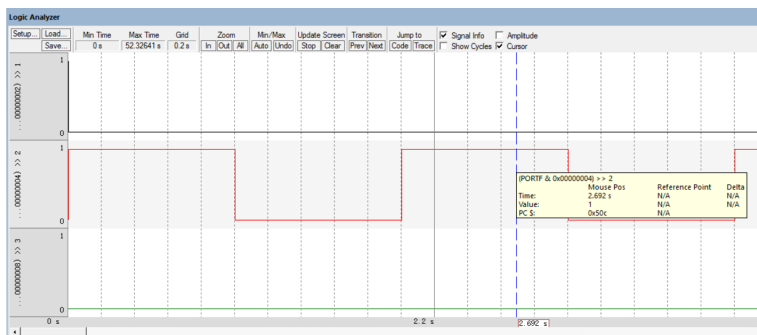
这是PF1的输出端口波形，该波形只有这一处方波信号，因为之前的代码在循环之外，运行之后就进入循环，故而这里只运行一遍。

## 查看内存记录

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。

在实验板上运行代码，得到的运行结果如下：

PF2的输出情况：



这种输出是因为在while循环中的语句：

```
1 while(1){
2     Led = GPIO_PORTF_DATA_R;    // read previous
3     Led = Led^0x04;             // toggle red LED
4     GPIO_PORTF_DATA_R = Led;    // output GPIO_PORTF_DATA_R;
5     /**/
6     SysTick_Wait10ms(100);      // wait 1s
7 }
```

这里的LED对应着输出的端口，随着循环的每一次运行，都对 0x04 进行异或运算，于是每次循环端口2都会与上次不同，由是得出矩形方波。由矩形方波的周期可以看出周期为2s，但是在循环的结尾部分添加了一个等待函数，恰好为1s，这说明用于运算的时间几乎为0。

内存记录的PF2状态值：

Call Stack + Locals		
Name	Location/Value	Type
Data	0x200003A0	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

对于PF2的计算过程如下：

```

1  while(1){
2      if(i<50){
3          Data[i] = GPIO_PORTF_DATA_R&0x04; // record PF2
4      }
5  }

```

在前50次循环内，每一次的运算都对Data序列进行异或运算，所以Data的变化趋势应该是非常明显，是典型的矩形方波。而数值变化也是在 `0x00` 与 `0x04` 之间波动。

以及开方结果如下：

Call Stack + Locals		
Name	Location/Value	Type
Sqrt	0x200002D8	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

开方运算的结果显示，在前31次运算中，始终有对于开方的运算结果，此后便不再变化，这是因为本次实验中的寄存器是32位的，在前31运算中，已经将32位填满，继续运算的过程中，开方之后的结果已经超出了32位寄存的范围，于是对于溢出操作，开方结果显示不再变化。

## 代码分析

本次实验的代码主要用于计算开方。

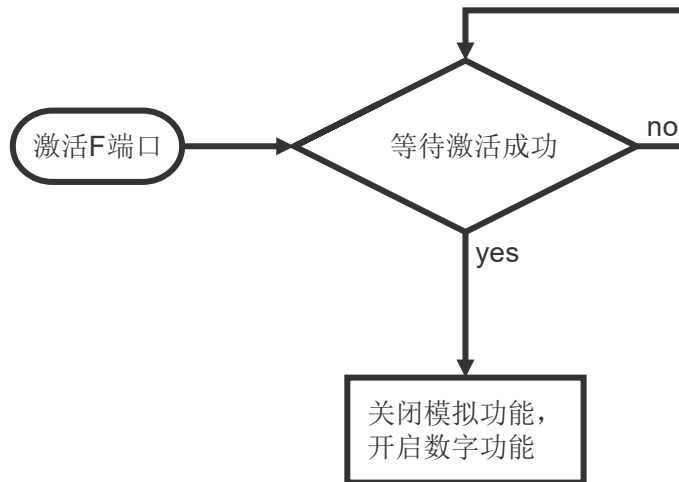
## 端口的设置

```

1  SYSTCL_RCGCGPIO_R |= 0x20;          // activate port F
2  while((SYSTCL_PRGPIO_R&0x20)==0){};
3  GPIO_PORTF_DIR_R  |= 0x0E;          // make PF3-1 output (PF3-1 built-in LEDs)
4  GPIO_PORTF_AFSEL_R &= ~0x0E;       // disable alt funct on PF3-1
5  GPIO_PORTF_DEN_R  |= 0x0E;          // enable digital I/O on PF3-1

```

步骤如下：



接下来，使用开方运算，并将输出的变化添加到开方的前后：

```
1 GPIO_PORTF_DATA_R= 0x02; // turn on led LED
2 tt = sqrt(ss);
3 GPIO_PORTF_DATA_R = 0x00; // turn off led LED
```

在debug的时候，可以通过输出端口的波形变化确定运算的时间与周期。

此后进入死循环：

```
1 while(1){
2     /*更新端口2的输出值*/
3     tt = sqrt(ss);
4     if(i<50){
5         /*使用数组记录输出值变化*/
6         i++;
7         ss=ss*2;
8     }
9     // wait 1s
10 }
```

在这里使用寄存器对迭代相乘的数字开方运算进行记录，并将该操作循环50次。由此可以得出寄存器寄存信息的上限。

## 附录

实验一编译器优化的图片：

对100开方，优化程度为0的实验结果：

Watch 1		
Name	Value	Type
elapsed	200	unsigned long
sqrt	<cannot evaluate>	uchar
data	<not in scope>	unsigned long[50]
Time[i]	<cannot evaluate>	uchar
<Enter expression>		

对100开方，优化程度为3的实验结果：






Watch 1		
Name	Value	Type
elapsed	191	unsigned long
sqrt	<cannot evaluate>	uchar
data	<not in scope>	unsigned long[50]
Time[i]	<cannot evaluate>	uchar
<Enter expression>		

对230400开方，优化程度为0的实验结果：

Watch 1		
Name	Value	Type
elapsed	210	unsigned long
sqrt	<cannot evaluate>	uchar
data	0x200002E0	unsigned long[50]
Time[i]	<cannot evaluate>	uchar
<Enter expression>		

对230400开方，优化程度为3的实验结果：

# Watch 1

Name	Value	Type
 elapsed	195	unsigned long
 sqrt	<cannot evaluate>	uchar
  data	<not in scope>	unsigned long[50]
 Time[i]	<cannot evaluate>	uchar
<Enter expression>		