目 录

1. 实验目的 1

2. 程序流程示意图 2

3. 代码及实现思路分析 3

3.1 实验要求二 3

3.2 实验要求三 3

4. 实验结果 4

4.1 实验二效果 4

4.2 实验三效果 4

5. 感想与收获 5

6. 讨论题 6

7. 源代码 8

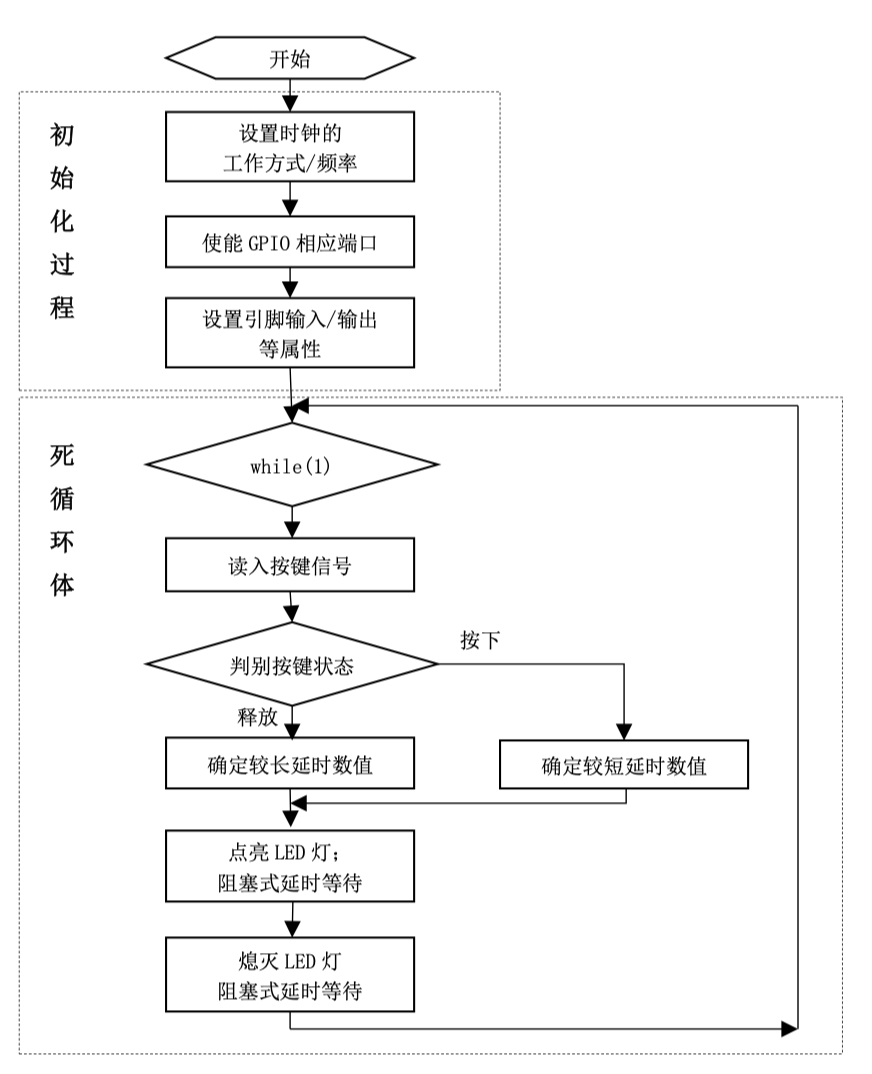
# 实验目的

熟悉KEILuVision5，自行建立工程项目

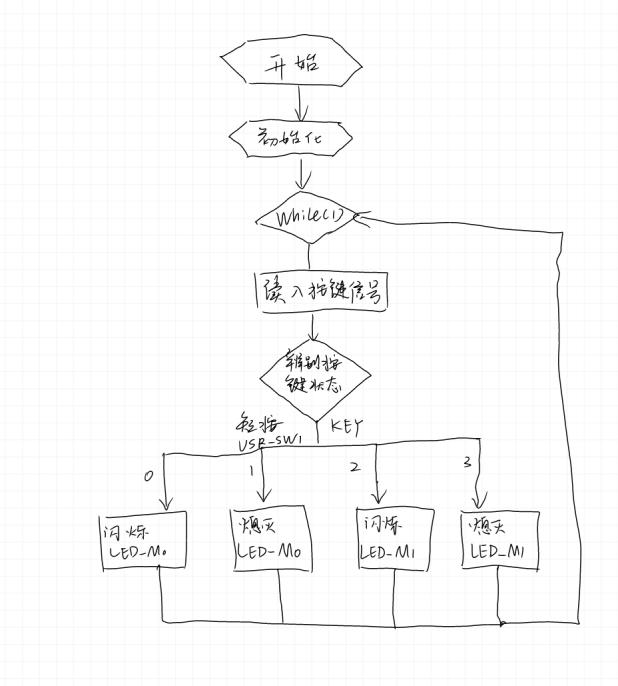
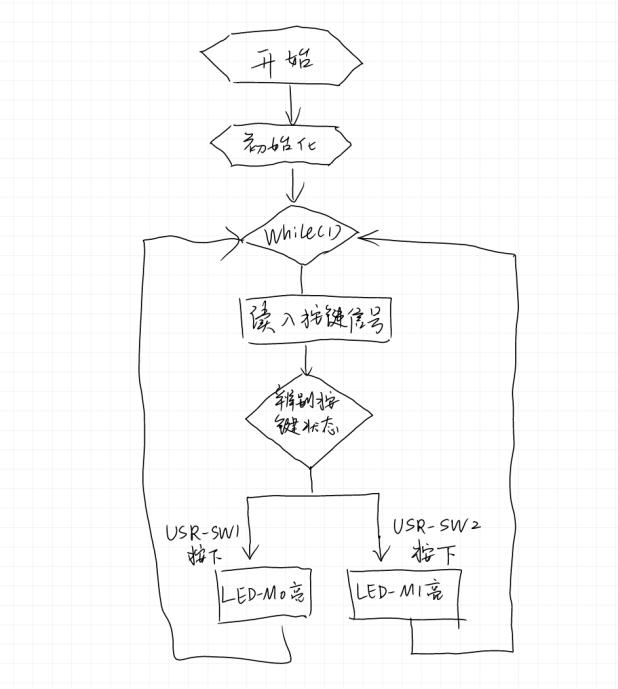
掌握GPIO的工作原理，能够结合GPIO的输入与输出功能进行实验

理解CPU的时钟信号，了解不同时钟对电源消耗的不同。

# 程序流程示意图



图一、程序总体逻辑



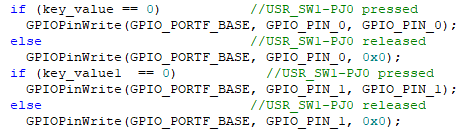
图二、实验二与实验三的逻辑

# 代码及实现思路分析

## 实验要求二

对于实验要求二：

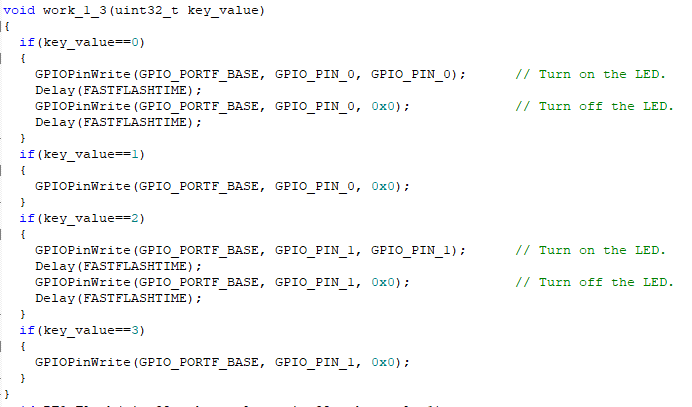
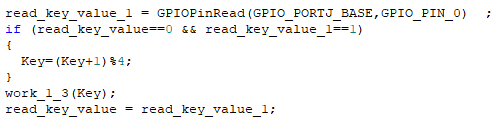
268a17a458aece4f71963ee9fb34340



用read\_key\_value和read\_key\_value\_1分别记录USR\_SW1和USR\_SW2是否被按下，将参数传入PF0\_Flash以后再判定LED\_M0与LED\_M1是处于亮或灭的状态。

## 实验要求三

对于实验要求三：



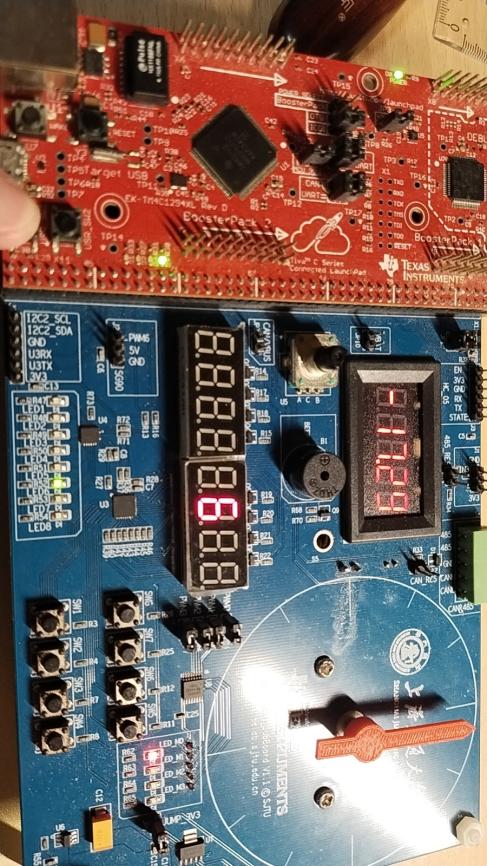
用Key维护此时处于第几次按下USR\_SW1的状态（从1到4循环，分别为：闪烁LED\_M0，熄灭LED\_M0，闪烁LED\_M1，熄灭LED\_M1），切换状态的判定条件为：前一刻USR\_SW1还未被按下，此刻USR\_SW1已经被按下。外部系统不断将Key传入work\_1\_3，再判断此时应该控制板子完成何种操作。

# 实验结果

## 实验二效果

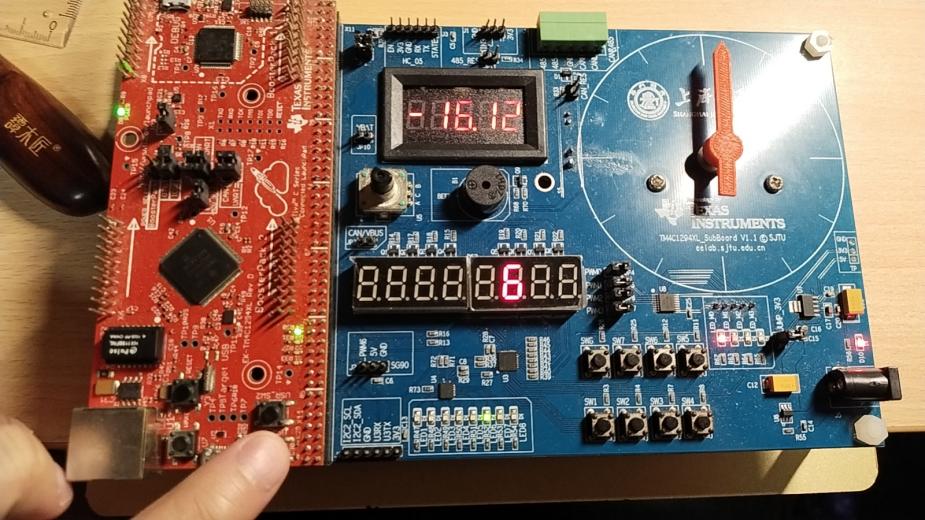
当按下USR\_SW1键时，点亮LED\_M0，放开时，熄灭LED\_M0

当按下USR\_SW2键时，点亮LED\_M1，放开时，熄灭LED\_M1

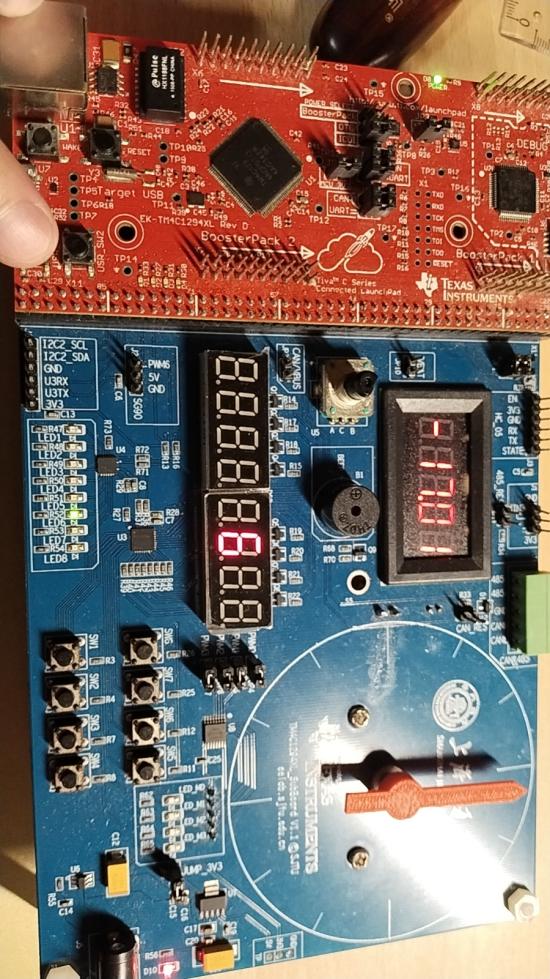


图三、按下USR\_SW1,点亮LED\_M1

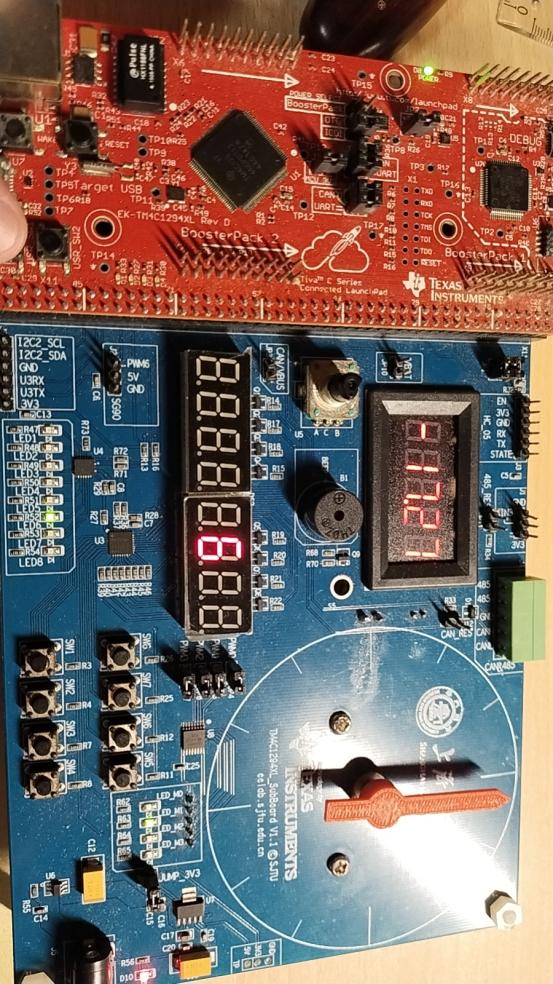
## 实验三效果



图四、第一次短按USR\_SW1键时，闪烁LED\_M0



图五、第二次短按USR\_SW1键时，熄灭LED\_M0



图六、第三次短按USR\_SW1键时，闪烁LED\_M1

当第一次短按USR\_SW1键时，闪烁LED\_M0，

当第二次短按USR\_SW1键时，熄灭LED\_M0，

当第三次短按USR\_SW1键时，闪烁LED\_M1，

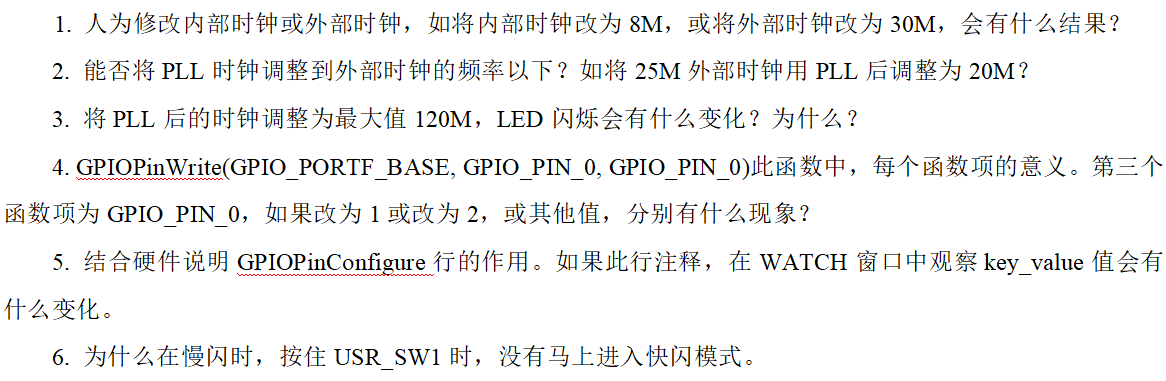
当第四次短按USR\_SW1键时，熄灭LED\_M1

# 感想与收获

在本次实验中，笔者首次使用了KEILuVision5，并自行完成了建立工程项目的任务。明确了使用Keil进行编程的基本思路：在主函数（main）中调用其他已经实现好并完成封装的功能。养成了：分析电路，明确需求，分解操作为基本操作组合；初始化（如设置GPIO状态等）；根据需求设置外设状态的基本操作流程习惯。

笔者在此次实验中掌握GPIO的工作原理，能够完成GPIO的端口配置与初始化，能够结合GPIO的输入（GPIOPinWrite）与输出（GPIOPinRead）功能进行实验操作。

# 讨论题



1.修改内部时钟或外部时钟的结果

内部时钟改为8M:意味着MCU的系统时钟从默认的频率（比如50MHz）降低到8MHz。

(1).系统运行速度变慢：所有定时和操作都会变慢。例如，计时器的时间间隔会变长。

(2).功耗降低：低频率下运行会减少功耗。

(3).外设时钟影响：所有依赖系统时钟的外设时钟也会随之变慢。

外部时钟改为30M:意味着使用外部时钟源提供的30MHz作为系统时钟或PLL的输入。

(1).系统运行速度改变：如果PLL配置得当，系统时钟可能提高或保持稳定。

(2).外设时钟影响：与外部时钟源同步的外设时钟会相应调整。

(3),稳定性和精度：外部时钟源的质量会直接影响系统的稳定性和时间精度。

2.PLL时钟调整到外部时钟的频率以下

外部时钟25M，通过PLL调整为20M：理论上可以，但这取决于具体的PLL配置能力。

PLL（Phase-LockedLoop）通常用于将一个输入频率乘以某个倍数，再除以另一个倍数来获得目标频率。如果PLL可以支持合适的倍频和分频因子，是可以将输入的25MHz调整为低于25MHz的频率，如20MHz。

影响：

(1).系统时钟降低：导致系统运行速度变慢。

(2).准确配置：需要正确计算PLL倍频和分频值，以得到精确的目标频率。

3.PLL后的时钟调整为最大值120M，LED闪烁的变化和原因

LED闪烁变化：

(1).频率增加：如果LED闪烁由定时器控制，定时器会更快地达到设定值，从而导致LED闪烁频率增加。

(2).周期变短：LED的闪烁周期变短，闪烁速度加快。

原因：

(1)时钟频率增大：系统时钟加快，所有基于时钟的操作都会变快。

(2)定时器和延时函数：在相同的计数值下，定时器和延时函数的时间间隔变短。

4.GPIOPinWrite函数中各参数的意义和变化

对于函数GPIOPinWrite(GPIO\_PORTF\_BASE,GPIO\_PIN\_0,GPIO\_PIN\_0)

(1).GPIO\_PORTF\_BASE：GPIO端口F的基地址。

(2).GPIO\_PIN\_0：GPIO端口F的第0引脚。

(3).GPIO\_PIN\_0：设置第0引脚为高电平。

修改第3个参数：

改为1：设置引脚电平为1（相当于高电平），引脚输出高电平。

改为2：设置引脚电平为2，这对于GPIO\_PIN\_0是不合适的，因为GPIO\_PIN\_0只有高电平和低电平，可能导致不正确的行为或无效操作。

改为其他值：具体值取决于GPIO引脚支持的电平值范围和含义，通常0为低电平，1为高电平，其他值可能无效。

5.GPIOPinConfigure行的作用及其注释影响

作用：GPIOPinConfigure用于配置GPIO引脚的模式，如其工作模式（如数字输入/输出、模拟功能等）。

注释掉的影响：

(1).如果GPIOPinConfigure被注释，可能导致引脚没有正确配置为所需的功能。

(2).key\_value可能不会反映实际的引脚状态，可能始终为默认值或未定义状态。

(3).影响调试和故障排除。

6.按住USR\_SW1时没有马上进入快闪模式的原因

(1).去抖动延迟：按键去抖动处理可能会有延迟，避免按键抖动导致误动作。

(2).软件逻辑延迟：软件可能有一定的延迟逻辑，例如按住一定时间后才进入快闪模式。

(3).状态检测周期：如果程序周期性检测按键状态，周期较长可能导致检测滞后。

# 源代码

#include <stdint.h>

#include <stdbool.h>

#include "hw\_memmap.h"

#include "debug.h"

#include "gpio.h"

#include "hw\_types.h"

#include "pin\_map.h"

#include "sysctl.h"

#define   FASTFLASHTIME         (uint32\_t) 300000

#define   SLOWFLASHTIME         (uint32\_t) FASTFLASHTIME\*20

void        Delay(uint32\_t value);

void        S800\_GPIO\_Init(void);

void        PF0\_Flash(uint32\_t key\_value, uint32\_t key\_value1);

void        work\_1\_3(uint32\_t key\_value);

int main(void)

{

    uint32\_t read\_key\_value=0;

    uint32\_t read\_key\_value\_1=0;

    uint32\_t Key=3;

    S800\_GPIO\_Init();

    while(1)

  {

        //1.2

        read\_key\_value = GPIOPinRead(GPIO\_PORTJ\_BASE,GPIO\_PIN\_0)    ;               //read the PJ0 key value

        read\_key\_value\_1 = GPIOPinRead(GPIO\_PORTJ\_BASE,GPIO\_PIN\_1)  ;

        PF0\_Flash(read\_key\_value, read\_key\_value\_1);

        //1.3

        /\*

        read\_key\_value\_1 = GPIOPinRead(GPIO\_PORTJ\_BASE,GPIO\_PIN\_0)  ;

        if (read\_key\_value==0 && read\_key\_value\_1==1)

        {

            Key=(Key+1)%4;

        }

        work\_1\_3(Key);

        read\_key\_value = read\_key\_value\_1;

        \*/

   }

}

void work\_1\_3(uint32\_t key\_value)

{

    if(key\_value==0)

    {

        GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_0, GPIO\_PIN\_0);          // Turn on the LED.

        Delay(FASTFLASHTIME);

        GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_0, 0x0);                         // Turn off the LED.

        Delay(FASTFLASHTIME);

    }

    if(key\_value==1)

    {

        GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_0, 0x0);

    }

    if(key\_value==2)

    {

        GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_1, GPIO\_PIN\_1);          // Turn on the LED.

        Delay(FASTFLASHTIME);

        GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_1, 0x0);                         // Turn off the LED.

        Delay(FASTFLASHTIME);

    }

    if(key\_value==3)

    {

        GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_1, 0x0);

    }

}

void PF0\_Flash(uint32\_t key\_value, uint32\_t key\_value1)

{

        /\* 1.1

    uint32\_t delay\_time;

        if (key\_value   == 0)                       //USR\_SW1-PJ0 pressed

            delay\_time                          = FASTFLASHTIME;

        else                                                    //USR\_SW1-PJ0 released

            delay\_time                          = SLOWFLASHTIME;

        GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_0, GPIO\_PIN\_0);          // Turn on the LED.

        Delay(delay\_time);

        GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_0, 0x0);                         // Turn off the LED.

        Delay(delay\_time);

    \*/

    //1.2

        if (key\_value   == 0)                       //USR\_SW1-PJ0 pressed

            GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_0, GPIO\_PIN\_0);

        else                                                    //USR\_SW1-PJ0 released

            GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_0, 0x0);

        if (key\_value1  == 0)                       //USR\_SW1-PJ0 pressed

            GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_1, GPIO\_PIN\_1);

        else                                                    //USR\_SW1-PJ0 released

            GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_1, 0x0);

}

void Delay(uint32\_t value)

{

    uint32\_t ui32Loop;

    for(ui32Loop = 0; ui32Loop < value; ui32Loop++){};

}

void S800\_GPIO\_Init(void)

{

    SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);                        //Enable PortF

    while(!SysCtlPeripheralReady(SYSCTL\_PERIPH\_GPIOF));         //Wait for the GPIO moduleF ready

    SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOJ);                        //Enable PortJ

    while(!SysCtlPeripheralReady(SYSCTL\_PERIPH\_GPIOJ));         //Wait for the GPIO moduleJ ready

  GPIOPinTypeGPIOOutput(GPIO\_PORTF\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);          //Set PF0 as Output pin

    GPIOPinTypeGPIOInput(GPIO\_PORTJ\_BASE,GPIO\_PIN\_0 | GPIO\_PIN\_1);//Set the PJ0,PJ1 as input pin

    GPIOPadConfigSet(GPIO\_PORTJ\_BASE,GPIO\_PIN\_0 | GPIO\_PIN\_1,GPIO\_STRENGTH\_2MA,GPIO\_PIN\_TYPE\_STD\_WPU);

}