LabSPI.hpp

```
/*
 * labspi.hpp
 *
 *  Created on: 21-Feb-2018
 *      Author: Kaustubh
 */


#ifndef LABSPI_HPP_
#define LABSPI_HPP_

#include "FreeRTOS.h"
#include "LPC17xx.h"
#include "labgpio.hpp"

class LabSPI
{
private:

    //SSP register lookup table structure
   static constexpr LPC_SSP_TypeDef * SSP[] =
   {LPC_SSP0,
    LPC_SSP1
   };

public:
```

```
enum FrameModes
{
    /* Fill this out based on the datasheet. */
    SPI = 00,
    TI = 01,
    Microwire = 10
};

enum Peripheral
{
    SSP0 = 0,
    SSP1 = 1
};
Peripheral periph;

/**
 * 1) Powers on SPPn peripheral
 * 2) Set peripheral clock
 * 3) Sets pins for specified peripheral to MOSI, MISO, and SCK
 *
 * @param peripheral which peripheral SSP0 or SSP1 you want to select.
 * @param data_size_select transfer size data width; To optimize the code, look for a pattern in the datasheet
 * @param format is the code format for which synchronous serial protocol you want to use.
 * @param divide is the how much to divide the clock for SSP; take care of error cases such as the value of 0, 1, and odd numbers
 *
 * @return true if initialization was successful
 */
```

```
    bool init(Peripheral peripheral, uint8_t data_size_select, FrameModes format, uint8_t divide);

    /**
     * Transfers a byte via SSP to an external device using the SSP data register.
     * This region must be protected by a mutex static to this class.
     *
     * @return received byte from external device via SSP data register.
     */
    uint8_t transfer(uint8_t send);

    void cs();
    void ds();

    LabSPI();
    ~LabSPI();
};
#endif /* LABSPI_HPP_ */
```

LabSPI.cpp

```
/*
 * labspi.cpp
 *
 *  Created on: 21-Feb-2018
 *      Author: Kaustubh
 */

#include "labspi.hpp"
#include "LPC17xx.h"
#include "stdio.h"
```

```
LabSPI::LabSPI()
{}

bool LabSPI::init(Peripheral peri,uint8_t dss, FrameModes mode, uint8_t divideby)
{
   periph = peri;
   if(peri == SSP1 && mode == SPI && 3<dss && dss<15 && 1<divideby && divideby<254 && divideby%2 == 0)
   {
      LPC_SC->PCONP &= ~(1<<10);
      LPC_SC->PCONP |= (1<<10);

      LPC_SC->PCLKSEL0 &= ~(3<<21);
      LPC_SC->PCLKSEL0 |= (3<<21);

      LPC_PINCON->PINSEL0 &= ~((3<<14)|(3<<16)|(3<<18));
      LPC_PINCON->PINSEL0 |= ((2<<14)|(2<<16)|(2<<18));

      SSP[peri]->CR0 = dss;
      SSP[peri]->CR1 |= (1<<1);
      SSP[peri]->CPSR = divideby;

      LPC_PINCON->PINSEL0 &= ~(3<<12);
      LPC_GPIO0->FIODIR |= (1<<6);

      //LPC_SSP1->DR = 0x0000;

      return 1;
   }
```

```
   else if(peri == SSP0 && mode == SPI && 3<dss && dss<15 && 1<divideby && divideby<254 && divideby%2
== 0)
   {
      LPC_SC->PCONP &= ~(1<<21);
      LPC_SC->PCONP |= (1<<21);

      LPC_SC->PCLKSEL1 &= ~(3<<10);
      LPC_SC->PCLKSEL1 |= (3<<10);

      LPC_PINCON->PINSEL0 &= ~(3<<30);
      LPC_PINCON->PINSEL0 |= (2<<30);
      LPC_PINCON->PINSEL1 &= ~((3<<2)|(3<<4));
      LPC_PINCON->PINSEL1 |= ((2<<2)|(2<<4));

      SSP[peri]->CR0 = dss;
      SSP[peri]->CR1 |= (1<<1);
      SSP[peri]->CPSR = divideby;

      LPC_PINCON->PINSEL1 &= ~(3<<0);
      LPC_GPIO0->FIODIR |= (1<<16);

      return 1;
   }

   else
   {
      return 0;
   }
}
```

```
void LabSPI::cs()
{
   if(periph == 1)
   {
      LabGPIO_0 obj1(0,6);
      LabGPIO_0 obj(0,0);
      obj1.setHigh();
      obj.setHigh();
   }
   else if(periph == 0)
   {
      LabGPIO_0 obj1(0,6);
      obj1.setHigh();
      LabGPIO_0 obj(0,1);
      obj1.setHigh();
   }
}

void LabSPI::ds()
{
   if(periph == 1)
   {
      LabGPIO_0 obj1(0,6);
      obj1.setLow();
      LabGPIO_0 obj(0,0);
      obj1.setLow();
   }
   else if(periph == 0)
```

```
    {
        LabGPIO_0 obj1(0,16);
        obj1.setLow();
        LabGPIO_0 obj1(0,1);
        obj1.setLow();
    }
}

uint8_t LabSPI::transfer(uint8_t send)
{
    //printf("transferring\n");
    //printf("Current DR contains:%02X\n", LPC_SSP1->DR);
    if(periph == 1)
    {
        LPC_SSP1->DR = send;
        while(LPC_SSP1->SR & (1<<4));
        return LPC_SSP1->DR;
    }
    else
    {
        LPC_SSP0->DR = send;
        while(LPC_SSP0->SR & (1<<4));
        return LPC_SSP0->DR;
    }
}

LabSPI::~LabSPI()
{}
```

Main.cpp:

```
/*
 *     SocialLedge.com - Copyright (C) 2013
 *
 *     This file is part of free software framework for embedded processors.
 *     You can use it and/or distribute it as long as this copyright header
 *     remains unmodified.  The code is free for personal use and requires
 *     permission to use in a commercial product.
 *
 *      THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 *      OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 *      MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
 *      I SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR
 *      CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 *
 *     You can reach the author of this software at :
 *          p r e e t . w i k i @ g m a i l . c o m
 */


/**
 * @file
 * @brief This is the application entry point.
 */


#include "FreeRTOS.h"
#include "task.h"
#include "uart0_min.h"
#include "LPC17xx.h"
#include "labgpio.hpp"
```

```
#include "stdio.h"
#include "semphr.h"
#include "utilities.h"
#include "printf_lib.h"
#include "labspi.hpp"
#include "mutex"

/*Assignment 4- SPI */

const uint8_t size = 2048;
uint8_t a[512];
SemaphoreHandle_t gatekeeper = 0;
LabSPI spiobj;

union
{
   uint8_t byte1;
   struct
   {
      uint8_t  ready:1 ;        // 1 bit field named "ready"
      uint8_t  main_mem_buffer:1 ;
      uint8_t  density:4 ;
      uint8_t  Sector_protection:1 ;
      uint8_t  device_config:1;
   }__attribute__((packed));
}Status1;

union
{
```

```
    uint8_t byte2;
    struct
    {
        uint8_t  ready:1 ;
        uint8_t  future_use:1 ;
        uint8_t  Erase_program:1 ;
        uint8_t  future_use2:1 ;
        uint8_t  sector_lockdown:1 ;
        uint8_t  prog_Buffer2:1 ;
        uint8_t  prog_Buffer1:1 ;
        uint8_t  erase_suspend:1 ;
    }__attribute__((packed));
}Status2;


void vSPITask(void*p)
{
        while(1)
        {
//          /* Insert Loop Code */
            spiobj.cs();
            uint8_t DevIDdummy = spiobj.transfer(0x9F); // Find what to send to read Adesto flash signature
            uint8_t ManufactureID = spiobj.transfer(0x9F); // Find what to send to read Adesto flash signature
            uint8_t DevID1 = spiobj.transfer(0x9F); // Find what to send to read Adesto flash signature
            uint8_t DevID2 = spiobj.transfer(0x9F); // Find what to send to read Adesto flash signature
            printf("Manufacture Id : 0x%02X\n",ManufactureID);
            printf("Device Id : 0x%02X%02X\n",DevID1,DevID2);
            for(int i=0; i<512; i++)
            {
                uint8_t page = spiobj.transfer(0xD2);
```

```
            printf("page data : 0x%02X",page);
        }
        spiobj.ds();
        delay_ms(1000);
    }
}

void vStatusRead(void*p)
{
  while(1)
  {
    spiobj.cs();
    uint8_t StatRegDummy = spiobj.transfer(0xD7); // Find what to send to read Adesto flash signature
    uint8_t StatByte1 = spiobj.transfer(0xD7); // Find what to send to read Adesto flash signature
    uint8_t StatByte2 = spiobj.transfer(0xD7); // Find what to send to read Adesto flash signature
    printf("Status Register : 0x%02X%02X\n",StatByte1,StatByte2);
    spiobj.ds();

    Status1.byte1 = StatByte1;
    Status2.byte2 = StatByte2;

    (Status1.ready == 0) ? printf("SPI flash is busy.\n"):printf("SPI flash is ready.\n");
    (Status1.main_mem_buffer == 0) ? printf("Main memory page data matches buffer data.\n"):printf("Main memory page data does not match buffer data.\n");
    (Status1.density == 0x0b)?printf("16 Mbit density.\n"):printf("Device density is not 16 Mbits\n");
    (Status1.Sector_protection == 0) ? printf("Sector protection is disabled.\n"):printf("Sector protection is enabled.\n");
    (Status1.device_config == 0) ? printf("Device is configured for standard DataFlash page size (528 bytes).\n"):printf("Device is configured for “power of 2” binary page size (512 bytes).\n");
    (Status2.ready == 0) ? printf("Device is busy with an internal operation.\n"):printf("SPI flash is ready.\n");
```

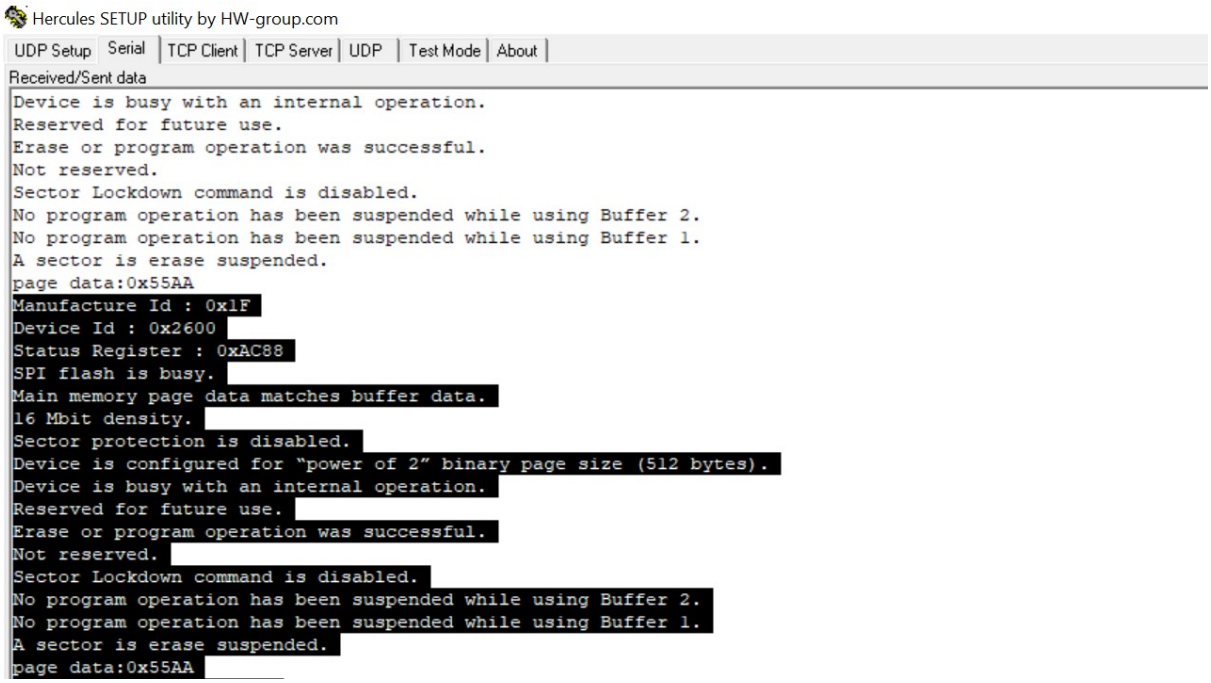
```
    (Status2.future_use == 1) ? printf("Not reserved.\n"):printf("Reserved for future use.\n");

    (Status2.Erase_program == 0) ? printf("Erase or program operation was successful.\n"):printf("Erase or program error detected.\n");

    (Status2.future_use2 == 1) ? printf("Not reserved.\n"):printf("Reserved for future use.\n");

    (Status2.sector_lockdown == 0) ? printf("Sector Lockdown command is disabled.\n"):printf("Sector Lockdown command is enabled.\n");

    (Status2.prog_Buffer2 == 0) ? printf("No program operation has been suspended while using Buffer 2.\n"):printf("A sector is program suspended while using Buffer 2.\n");

    (Status2.prog_Buffer1 == 0) ? printf("No program operation has been suspended while using Buffer 1.\n"):printf("A sector is program suspended while using Buffer 1.\n");

    (Status2.erase_suspend == 0) ? printf("No sectors are erase suspended.\n"):printf("A sector is erase suspended.\n");

    delay_ms(1000);

  }

}


int main(void)

{

  //gatekeeper = xSemaphoreCreateMutex();

   //TODO: Initialize your SPI

  uint8_t x = spiobj.init(LabSPI::SSP1,7,LabSPI::SPI,8);

  while(1)

  {

    if(x==1)

    {

      spiobj.cs();

      uint8_t DevIDdummy = spiobj.transfer(0x9F); // Find what to send to read Adesto flash signature

      uint8_t ManufactureID = spiobj.transfer(0x9F); // Find what to send to read Adesto flash signature

      uint8_t DevID1 = spiobj.transfer(0x9F); // Find what to send to read Adesto flash signature

      uint8_t DevID2 = spiobj.transfer(0x9F); // Find what to send to read Adesto flash signature

      printf("Manufacture Id : 0x%02X\n",ManufactureID);
```

```
printf("Device Id : 0x%02X%02X\n",DevID1,DevID2);

spiobj.ds();


spiobj.cs();

uint8_t StatRegDummy = spiobj.transfer(0xD7); // Find what to send to read Adesto flash signature

uint8_t StatByte1 = spiobj.transfer(0xD7); // Find what to send to read Adesto flash signature

uint8_t StatByte2 = spiobj.transfer(0xD7); // Find what to send to read Adesto flash signature

printf("Status Register : 0x%02X%02X\n",StatByte1,StatByte2);

spiobj.ds();


Status1.byte1 = StatByte1;

Status2.byte2 = StatByte2;


(Status1.ready == 0) ? printf("SPI flash is busy.\n"):printf("SPI flash is ready.\n");

(Status1.main_mem_buffer == 0) ? printf("Main memory page data matches buffer
data.\n"):printf("Main memory page data does not match buffer data.\n");

(Status1.density == 0x0b)?printf("16 Mbit density.\n"):printf("Device density is not 16 Mbits\n");

(Status1.Sector_protection == 0) ? printf("Sector protection is disabled.\n"):printf("Sector protection is
enabled.\n");

(Status1.device_config == 0) ? printf("Device is configured for standard DataFlash page size (528
bytes).\n"):printf("Device is configured for “power of 2” binary page size (512 bytes).\n");

(Status2.ready == 0) ? printf("Device is busy with an internal operation.\n"):printf("SPI flash is
ready.\n");

(Status2.future_use == 1) ? printf("Not reserved.\n"):printf("Reserved for future use.\n");

(Status2.Erase_program == 0) ? printf("Erase or program operation was successful.\n"):printf("Erase or
program error detected.\n");

(Status2.future_use2 == 1) ? printf("Not reserved.\n"):printf("Reserved for future use.\n");

(Status2.sector_lockdown == 0) ? printf("Sector Lockdown command is disabled.\n"):printf("Sector
Lockdown command is enabled.\n");

(Status2.prog_Buffer2 == 0) ? printf("No program operation has been suspended while using Buffer
2.\n"):printf("A sector is program suspended while using Buffer 2.\n");
```

```
        (Status2.prog_Buffer1 == 0) ? printf("No program operation has been suspended while using Buffer 1.\n"):printf("A sector is program suspended while using Buffer 1.\n");

        (Status2.erase_suspend == 0) ? printf("No sectors are erase suspended.\n"):printf("A sector is erase suspended.\n");

            spiobj.cs();

            uint8_t page = spiobj.transfer(0xD2);

            spiobj.transfer(0x00);

            spiobj.transfer(0x00);

            spiobj.transfer(0x00);

            spiobj.transfer(0x01);

            spiobj.transfer(0x01);

            spiobj.transfer(0x01);

            spiobj.transfer(0x01);


            for(int i = 0; i<512; i++)

            {

               uint8_t page1 = spiobj.transfer(0x00);

               a[i] = page1;

            }

            printf("page data:0x%02X%02X",a[510],a[511]);

            spiobj.ds();

       delay_ms(1000);

      }


   }

//    xTaskCreate(vSPITask,"SPITask",size,0,1,NULL);

//    xTaskCreate(vStatusRead,"StatusRead",size,0,1,NULL);

//

//    if(x==1)

//    {
```

```
//       vTaskStartScheduler();
//   }
//   else
//       return 0;

   return 0;
}
```

Terminal Output:-



Logic analyzer:-