SPI.hpp

```
#ifndef LPC_SPI_H__
#define LPC_SPI_H__
#include "LPC17xx.h"
#include "stdint.h"
#include "printf_lib.h"
#include "gpio_1.hpp"
    enum FrameModes
    {
        /* Fill this out based on the datasheet. CPHA:CPOL */
        mode0,                                      // 00
        mode1,                                      // 01
        mode2,                                      // 10
        mode3,                                      // 11
    };


typedef union
{
    uint16_t stsmem;
    struct
    {
        uint8_t erss : 1;
        uint8_t ps1 : 1;
        uint8_t ps2 : 1;
        uint8_t sle : 1;
        uint8_t res1 : 1;
        uint8_t epe : 1;
        uint8_t res2 : 1;
        uint8_t rdy1 : 1;

        uint8_t pgsize : 1;
        uint8_t protect : 1;
        uint8_t dens : 4;
        uint8_t comp : 1;
        uint8_t rdy2 : 1;
    } __attribute__((packed));
} adesto_t;

class LabSpi
{
 public:

    bool initialize(uint8_t data_size_select, FrameModes format,
uint8_t divide);
```

```
    uint8_t transfer(uint8_t send);
    uint8_t transfersts(uint8_t sendsts);

    LabSpi();
    ~LabSpi();

 private:

};

#endif
```

SPI.cpp

```
#include "SSP_1.hpp"
#include "gpio_1.hpp"

uint8_t LabSpi :: transfersts(uint8_t sendsts)
{
    LPC_SSP1->DR = sendsts;
    while(LPC_SSP1->SR & (1 << 4));
    return LPC_SSP1->DR;
}

bool LabSpi :: initialize(uint8_t data_size_select, FrameModes format,
uint8_t divide)
{
    bool f = 0;

    //Power on SSP1
    LPC_SC->PCONP |= (1 << 10);

    //Pin configuration 15:14 - SCK1, 17:16 - MISO, 19:18 - MOSI
    LPC_PINCON->PINSEL0 |= (0 << 14) | (1 << 15) | (0 << 16) | (1 <<
17) | (0 << 18) | (1 << 19);

    LPC_GPIO0->FIODIR |= (1 << 6) | (1 << 7) | (0 << 8) | (1 << 9);

    //Initially CS = 1
    LPC_GPIO0->FIOSET |= (1 << 6);

    LPC_SSP1->CR1 |= (1 << 1);
    LPC_SSP1->CR0 = 0x00;

    if((data_size_select <=15) && (data_size_select != 0) &&
(data_size_select != 1))
        {
        LPC_SSP1->CR0 |= (data_size_select-1) | (format << 6);
        u0_dbg_printf("%d\n", LPC_SSP1->CR0);
        }

    else
        u0_dbg_printf("Invalid data size \n");

    if(divide > 1 && (divide & 1) != 1)
        LPC_SSP1->CPSR = divide;
    else {
        u0_dbg_printf("Invalid Clock divide\n");
```

```
        f=0;
    }
    u0_dbg_printf("Init Done\n");

    return f;
}

uint8_t LabSpi :: transfer(uint8_t send)
{
    LPC_SSP1->DR = send;
    while(LPC_SSP1->SR & (1 << 4));
    uint32_t stclr = LPC_SSP1->SR;
    return LPC_SSP1->DR;
}

LabSpi :: LabSpi(){}
LabSpi :: ~LabSpi(){}
```

Main.cpp

```
#include "LPC17xx.h"
#include "stdint.h"
#include "printf_lib.h"
#include <FreeRTOS.h>
#include "task.h"
#include "semphr.h"
#include "SSP_1.hpp"

SemaphoreHandle_t spi_bus_lock;

LabSpi obj;
GPIO_Lab_0 CS(6);

uint8_t d[6];
uint16_t s[4];

void adesto_chip_select(void)
{
    CS.setLow();
}

void adesto_deselect(void)
{
    CS.setHigh();
}

void vReadFromSSP(void *pvParameter)
{
    while(1) {
        if(xSemaphoreTake(spi_bus_lock, 1000))
        {
            // Use Guarded Resource
           adesto_chip_select();
           d[0] = obj.transfer(0x09f);
           d[1] = obj.transfer(0x09f);
           d[2] = obj.transfer(0x09f);
           d[3] = obj.transfer(0x09f);
           d[4] = obj.transfer(0x09f);
           d[5] = obj.transfer(0x09f);
           adesto_deselect();

           adesto_chip_select();
           s[0] = obj.transfer(0xd7);
```

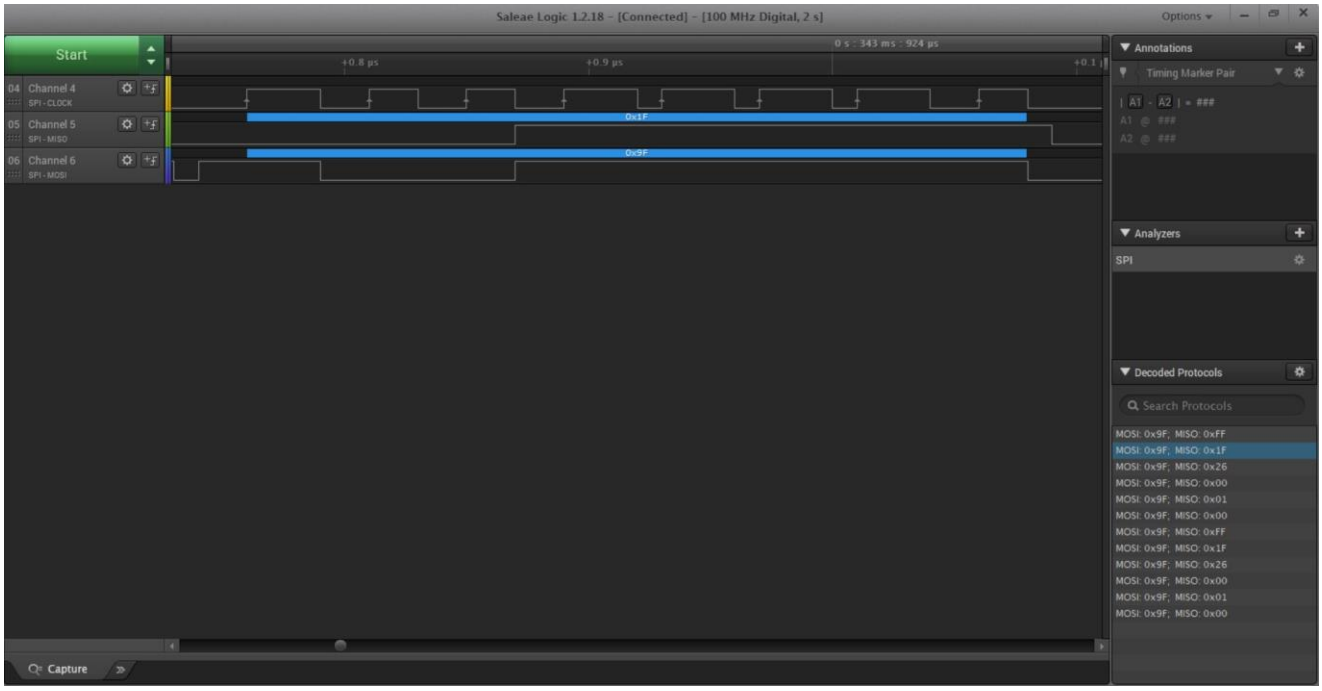
```
            s[1] = obj.transfer(0xff);
            s[2] = obj.transfer(0xff);

            adesto_deselect();

             // Give Semaphore back:
            xSemaphoreGive(spi_bus_lock);
        }
        else u0_dbg_printf("Failed to get the semaphore for read
task");
        vTaskDelay(1000);
    }
}

void vOutputFromSSP(void *pvParameter)
{
    while(1) {
        if(xSemaphoreTake(spi_bus_lock, 1000))
        {
            adesto_t status;
            status.stsmem = s[2] | (s[1] << 8);

            u0_dbg_printf("\nReturned Data: %x %x %x %x
%x\n",d[1],d[2],d[3],d[4],d[5]);

            u0_dbg_printf("\nManufacturer ID: %x\n", d[1]);
            u0_dbg_printf("\nDevice ID: %x\n", d[2]);
            u0_dbg_printf("\nDevice ID: %x\n", d[3]);
            u0_dbg_printf("\nEDI String length: %x\n", d[4]);
            u0_dbg_printf("\nEDI byte1: %x\n", d[5]);

            u0_dbg_printf("\nStatus: %x\n", status.stsmem);

            if(status.rdy2 == 1)
                u0_dbg_printf("\nDevice is ready \n");
            else u0_dbg_printf("\nDevice is busy \n");
            if(status.erss == 1)
                u0_dbg_printf("\nA sector is erase suspended \n");
            else u0_dbg_printf("\nNo sectors are erase suspended \n");
            if(status.ps1 == 1)
                u0_dbg_printf("\nA sector is program suspended while
using Buffer 1 \n");
            else u0_dbg_printf("\nNo program operation has been
suspended while using Buffer 1 \n");
            if(status.ps2 == 1)
```

```
                u0_dbg_printf("\nA sector is program suspended while
using Buffer 2 \n");
            else u0_dbg_printf("\nNo program operation has been
suspended while using Buffer 2 \n");
            if(status.sle == 1)
                u0_dbg_printf("\nSector Lckdwn EN \n");
            else u0_dbg_printf("\nSector Lckdwn ~EN \n");
            if(status.epe == 1)
                u0_dbg_printf("\nErase or program error detected \n");
            else u0_dbg_printf("\nErase or program successful \n");
            if(status.comp == 1)
                u0_dbg_printf("\nMain memory page data matches buffer
data \n");
            else u0_dbg_printf("\nMain memory page data does not match
buffer data \n");
            if(status.pgsize == 1)
                u0_dbg_printf("\npower of 2 binary page size \n");
            else u0_dbg_printf("\nStandard DataFlash page size \n");
            if(status.protect == 1)
                u0_dbg_printf("\nSector protection is EN \n");
            else u0_dbg_printf("\nSector protection is ~EN \n");
            u0_dbg_printf("\nDensity: %x \n", status.dens);

            xSemaphoreGive(spi_bus_lock);

        }
        else u0_dbg_printf("\nFailed to get the semaphore for display
task \n");
        vTaskDelay(1000);
    }
}

int main(void)
{
    spi_bus_lock = xSemaphoreCreateMutex();

    obj.initialize(8, mode0, 2);

    xTaskCreate(vReadFromSSP, (const char*) "Read_data", 512, NULL, 1,
NULL);
    xTaskCreate(vOutputFromSSP, (const char*) "Output_data", 512,
NULL, 1, NULL);
    vTaskStartScheduler();
}
```

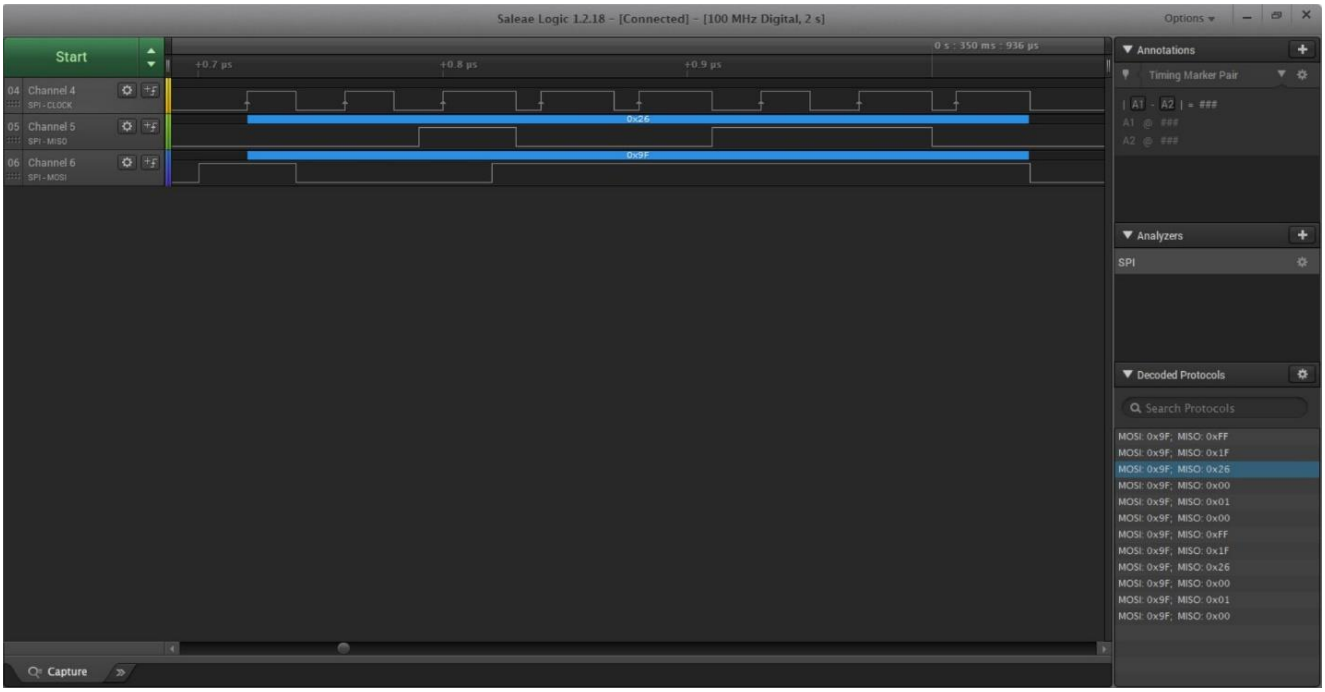
Output Terminal:

Logic Analyzer Output:

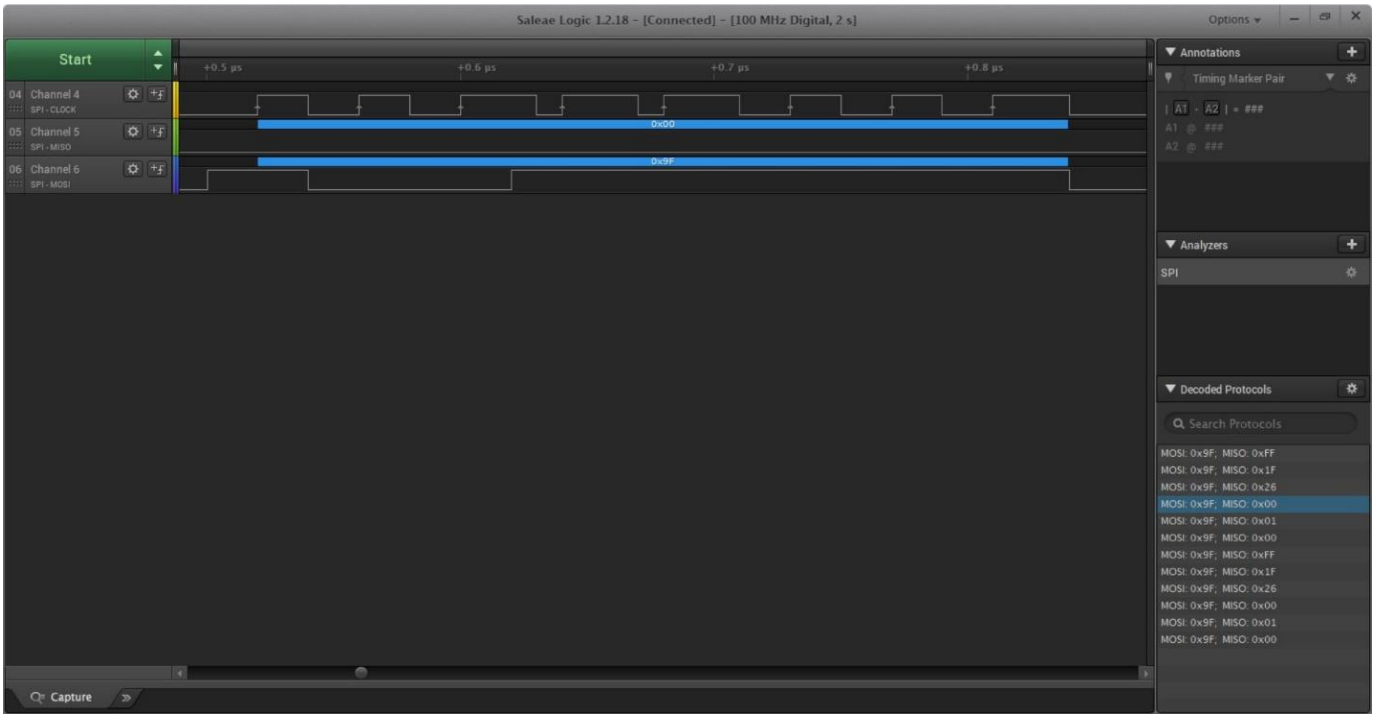
Data returned: 1f



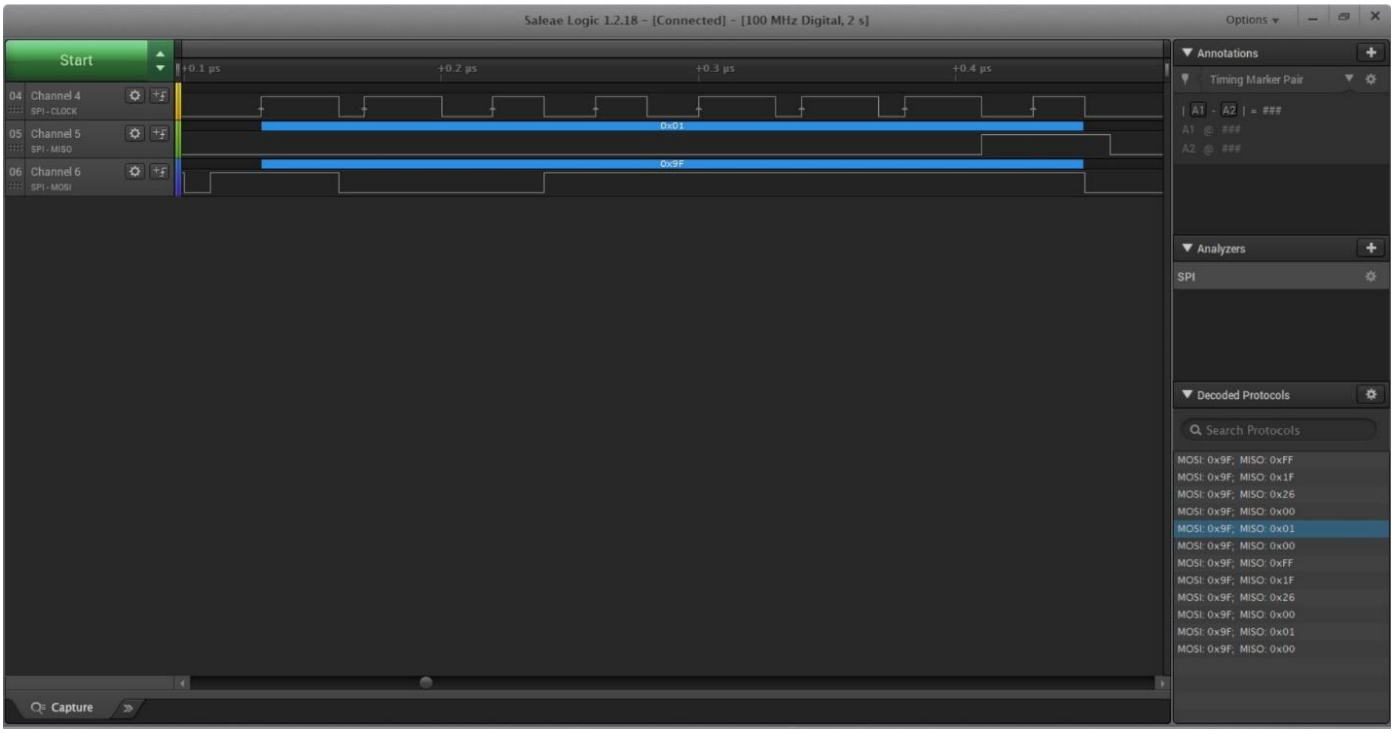
Data returned: 26



Data returned: 00



Data returned: 01



Data printed: 00