ssp_lab.h

```
//
// Created by nick_ on 9/27/2020.
//

#pragma once

#include <stdint.h>
#include <stdlib.h>

void ssp2__init(uint32_t max_clock_mhz);

uint8_t ssp2__swap_byte(uint8_t data_out);
```

ssp_lab.c

```
//
// Created by nick_ on 9/27/2020.
//

#include "ssp_lab.h"
#include "clock.h"
#include "gpio.h"
#include "lpc40xx.h"
#include <stdint.h>
#include <stdio.h>

void ssp2__init(uint32_t max_clock_mhz) {
 // Refer to LPC User manual and setup the register bits correctly

 //  LPC_IOCON->P1_0 &= ~(0x7); // ssp2 sck
 //  LPC_IOCON->P1_1 &= ~(0x7); // ssp2 mosi
 //  LPC_IOCON->P1_4 &= ~(0x7); // ssp2 miso
 //  LPC_IOCON->P1_0 |= (0x4);
 //  LPC_IOCON->P1_1 |= (0x4);
 //  LPC_IOCON->P1_4 |= (0x4);
 gpio__construct_with_function(GPIO__PORT_1, 0, GPIO__FUNCTION_4);
 gpio__construct_with_function(GPIO__PORT_1, 1, GPIO__FUNCTION_4);
 gpio__construct_with_function(GPIO__PORT_1, 4, GPIO__FUNCTION_4);

 // a) Power on Peripheral
 LPC_SC->PCONP |= (1 << 20); // bit 20 is ssp2
 // b) Setup control registers CR0 and CR1
 // page 610
 LPC_SSP2->CR0 = 0x7;      // 8 bit mode, SPI
 LPC_SSP2->CR1 = (1 << 1); // normal operation, ssp enabled
                           //  LPC_SSP2->CPSR = 96;
 // c) Setup prescalar register to be <= max_clock_mhz
 uint8_t divider = 2;
 const uint32_t cpu_clock_mhz = clock__get_core_clock_hz() / 1000000UL;
 while (max_clock_mhz < (cpu_clock_mhz / divider) && divider < 255) {
   divider += 2;
 }
 fprintf(stderr, "Clock = %d\nDivider: %d\nSCK: %d\n", cpu_clock_mhz, divider,
cpu_clock_mhz / divider);
 LPC_SSP2->CPSR = divider;
}

uint8_t ssp2__swap_byte(uint8_t data_out) {
 // Configure the Data register(DR) to send and receive data by checking the SPI
peripheral status register
 LPC_SSP2->DR = data_out; // set the data register 8bits
 while (LPC_SSP2->SR & (1 << 4)) {
   ;
 } // wait for bit 4 (busy signal)

 return (uint8_t)(LPC_SSP2->DR & 0xff); // return the data register
}
```

Main part 1

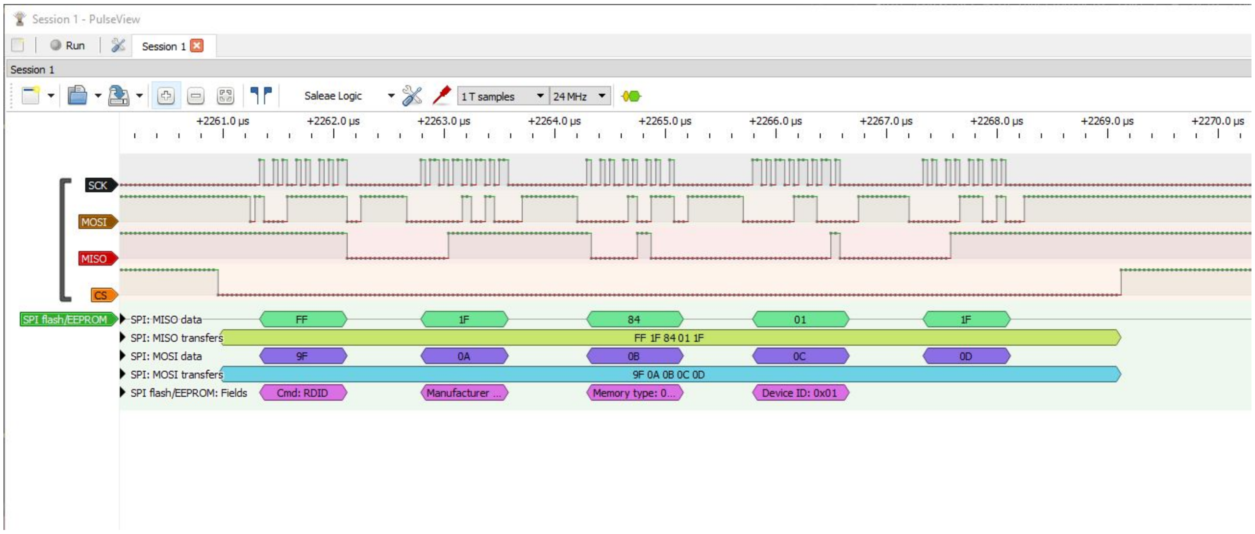
```
#if Lab5
static SemaphoreHandle_t spi_bus_mutex;
typedef struct {
 uint8_t manufacturer_id;
 uint8_t device_id_1;
 uint8_t device_id_2;
 uint8_t extended_device_id;
} adesto_flash_id_s;
gpio_s chipSelect;
void task_one();
void task_two();
void adesto_cs(void) { // LPC_GPIO1->CLR |= (1 << 10);
 gpio__reset(chipSelect);
}
void adesto_ds(void) { // LPC_GPIO1->SET |= (1 << 10);
 gpio__set(chipSelect);
}
void spi_task(void *p);
#endif
```

```
int main(void) { // main function for project
 puts("Starting RTOS");

#if outOfTheBox
 create_blinky_tasks();
 create_uart_task();
#else
 SemaphoreHandle_t spi_bus_mutex = xSemaphoreCreateMutex();
 xTaskCreate(spi_task, /*description*/ "spi_task", /*stack depth*/ 4096 /
sizeof(void *), /*parameter*/ (void *)1,
             /*priority*/ 1, /*optional handle*/ NULL);
#endif
 vTaskStartScheduler(); // This function never returns unless RTOS scheduler runs
out of memory and fails
 return 0;
}

#if Lab5
adesto_flash_id_s adesto_read_signature(void) {
 adesto_flash_id_s data = {0};

 adesto_cs();
 {
   ssp2__swap_byte(0x9F);
   data.manufacturer_id = ssp2__swap_byte(0xa);
   data.device_id_1 = ssp2__swap_byte(0xb);
   data.device_id_2 = ssp2__swap_byte(0xc);
   data.extended_device_id = ssp2__swap_byte(0xd);
 }
 adesto_ds();

 return data;
}

void spi_task(void *p) {
 const uint32_t spi_clock_mhz = 24;
 //  LPC_IOCON->P1_10 = 0;
 //  LPC_GPIO1->DIR |= (1 << 10);
 //  LPC_GPIO1->SET |= (1 << 10);
 chipSelect = gpio__construct_as_output(GPIO__PORT_1, 10);

 ssp2__init(spi_clock_mhz);

 while (1) {
   adesto_flash_id_s id = adesto_read_signature();
   printf("manufacturer_id: %x\n"
          "device_id_1: %x\n"
          "device_id_2: %x\n"
          "extended_device_id: %x\n\n",
          id.manufacturer_id, id.device_id_1, id.device_id_2,
id.extended_device_id);

   vTaskDelay(500);
 }
}
#endif
```

Main part 2b

```
#if Lab5
static SemaphoreHandle_t spi_bus_mutex;
typedef struct {
 uint8_t manufacturer_id;
 uint8_t device_id_1;
 uint8_t device_id_2;
 uint8_t extended_device_id;
} adesto_flash_id_s;
gpio_s chipSelect;
void task_one();
void task_two();
void adesto_cs(void) { // LPC_GPIO1->CLR |= (1 << 10);
 gpio__reset(chipSelect);
}
void adesto_ds(void) { // LPC_GPIO1->SET |= (1 << 10);
 gpio__set(chipSelect);
}
void spi_task(void *p);
#endif

int main(void) { // main function for project
 puts("Starting RTOS");

#if outOfTheBox
 create_blinky_tasks();
 create_uart_task();
#else
 const uint32_t spi_clock_mhz = 16;
 //  LPC_IOCON->P1_10 = 0;
 //  LPC_GPIO1->DIR |= (1 << 10);
 //  LPC_GPIO1->SET |= (1 << 10);
 chipSelect = gpio__construct_as_output(GPIO__PORT_1, 10);
 ssp2__init(spi_clock_mhz);
 spi_bus_mutex = xSemaphoreCreateMutex();
 xTaskCreate(spi_task, /*description*/ "spi_task", /*stack depth*/ 4096 /
sizeof(void *), /*parameter*/ (void *)1,
             /*priority*/ 1, /*optional handle*/ NULL);
 xTaskCreate(spi_task, /*description*/ "spi_task2", /*stack depth*/ 4096 /
sizeof(void *), /*parameter*/ (void *)1,
             /*priority*/ 1, /*optional handle*/ NULL);
#endif
 vTaskStartScheduler(); // This function never returns unless RTOS scheduler runs
out of memory and fails
 return 0;
}

#if Lab5
adesto_flash_id_s adesto_read_signature(void) {
 adesto_flash_id_s data = {0};

 adesto_cs();
 {
   ssp2__swap_byte(0x9F);
   data.manufacturer_id = ssp2__swap_byte(0xa);
   data.device_id_1 = ssp2__swap_byte(0xb);
   data.device_id_2 = ssp2__swap_byte(0xc);
   data.extended_device_id = ssp2__swap_byte(0xd);
 }
 adesto_ds();

 return data;
}

void spi_task(void *p) {

 while (1) {
   if (xSemaphoreTake(spi_bus_mutex, portMAX_DELAY)) {
     // Use Guarded Resource
     adesto_flash_id_s id = adesto_read_signature();
     fprintf(stderr,
             "manufacturer_id: %x\n"
             "device_id_1: %x\n"
             "device_id_2: %x\n"
             "extended_device_id: %x\n\n",
             id.manufacturer_id, id.device_id_1, id.device_id_2,
id.extended_device_id);
     if (id.manufacturer_id != 0x1F) {
       fprintf(stderr, "Manufacturer ID read failure\n");
       vTaskSuspend(NULL); // Kill this task
```

```
      }
      // Give Semaphore back:
      xSemaphoreGive(spi_bus_mutex);
      vTaskDelay(500);
    }
  }
}
#endif
```



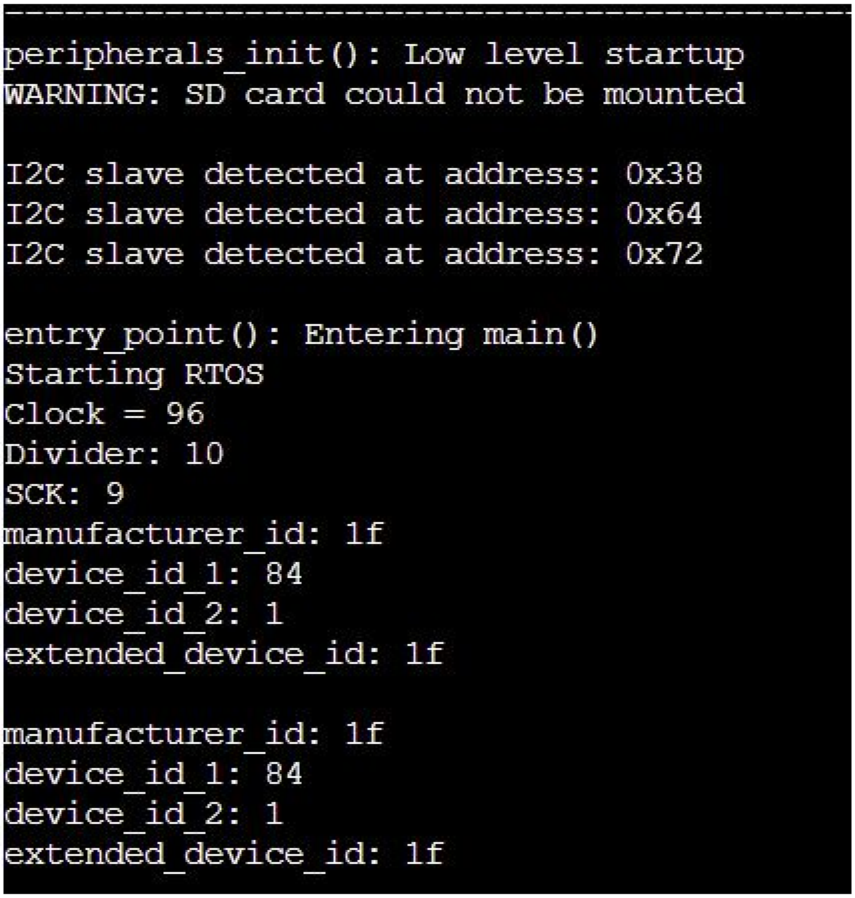
Saleae Logic

1 T samples

24 MHz

```
SPI: MISO data          FF        1F              84               01          1F
SPI: MISO transfers     FF 1F 84 01 1F
SPI: MOSI data          9F        0A              0B               0C          0D
SPI: MOSI transfers     9F 0A 0B 0C 0D
SPI flash/EEPROM: Fields  Cmd: RDID  Manufacturer ...  Memory type: 0...  Device ID: 0x01
```

Logic Analyzer capture



```
------------------------------------------
peripherals_init(): Low level startup
WARNING: SD card could not be mounted

I2C slave detected at address: 0x38
I2C slave detected at address: 0x64
I2C slave detected at address: 0x72

entry_point(): Entering main()
Starting RTOS
Clock = 96
Divider: 10
SCK: 9
manufacturer_id: 1f
device_id_1: 84
device_id_2: 1
extended_device_id: 1f

manufacturer_id: 1f
device_id_1: 84
device_id_2: 1
extended_device_id: 1f
```

Telemetry output