LAB4

Program the on chip SPI module

Outline

- Learn to utilize the on-chip SPI module
- Implement it in C
- Translate it to ARM Assembly
- Test and verify the result using oscilloscope and shift register.

Serial Peripheral Interface Bus (SPI)

- The SPI bus is a **synchronous serial data link** standard. It is mainly used for communication between a host processor and its peripheral devices.
- SPI is operated in **full duplex** mode: device can send and receive data at the same time.
- A SPI device (a device with SPI interface) can operate in either **master mode** or **slave mode**. There can be only 1 SPI master, while can be multiple SPI slaves.
- Master device generates control signal including **clock signal** and **slave selection signal**. Slave devices are controlled by master device.

Serial Peripheral Interface Bus (SPI)

- The SPI interface contains 4 input/output pins:
  - **SCLK:** Serial Clock
    1. SPI Master output serial clock signal
    2. SPI Slave input serial clock signal
  - **MOSI:** Master Output Slave Input
    1. SPI Master output (send) serial data
    2. SPI Slave input (receive) serial data
  - **MISO:** Master Input Slave Output
    1. SPI Master input (receive) serial data
    2. SPI Slave output (send) serial data
  - **SS(SSEL):** Slave Select
    1. SPI Master output selection signal
    2. SPI Slave input selection signal

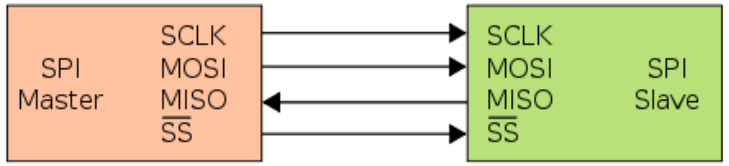


Fig 1. 1 SPI master to 1 SPI slave

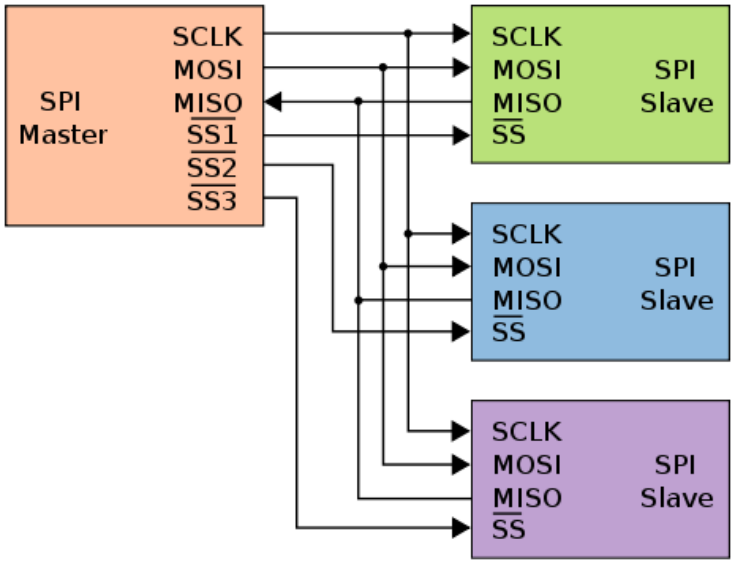


Fig 2. 1 SPI master to multiple SPI slave

Clock Polarity and Clock Phase

- In addition to setting the clock frequency, the master device must also configure the **Clock Polarity** (**represented by CPOL**) and **Clock Phase** (**represented by CPHA**) with respect to the data.
- CPOL determines the **base voltage level** (0: base level HIGH; 1:base level LOW)
- CPHA determines the **sampling clock edge** (0: sample at first clock edge; 1: sample at second clock edge)
- In other words:
  - At CPOL = 0 the base value of the clock is LOW
    - For CPHA = 0, data are read on the clock's rising edge (low->high transition) and data are changed on a falling edge (high->low clock transition).
    - For CPHA = 1, data are read on the clock's falling edge and data are changed on a rising edge.
  - At CPOL = 1 the base value of the clock is HIGH (inversion of CPOL=0)
    - For CPHA = 0, data are read on clock's falling edge and data are changed on a rising edge.
    - For CPHA = 1, data are read on clock's rising edge and data are changed on a falling edge.

Clock Polarity and Clock Phase

- The following figure shows the details of **CPOL** and **CPHA**

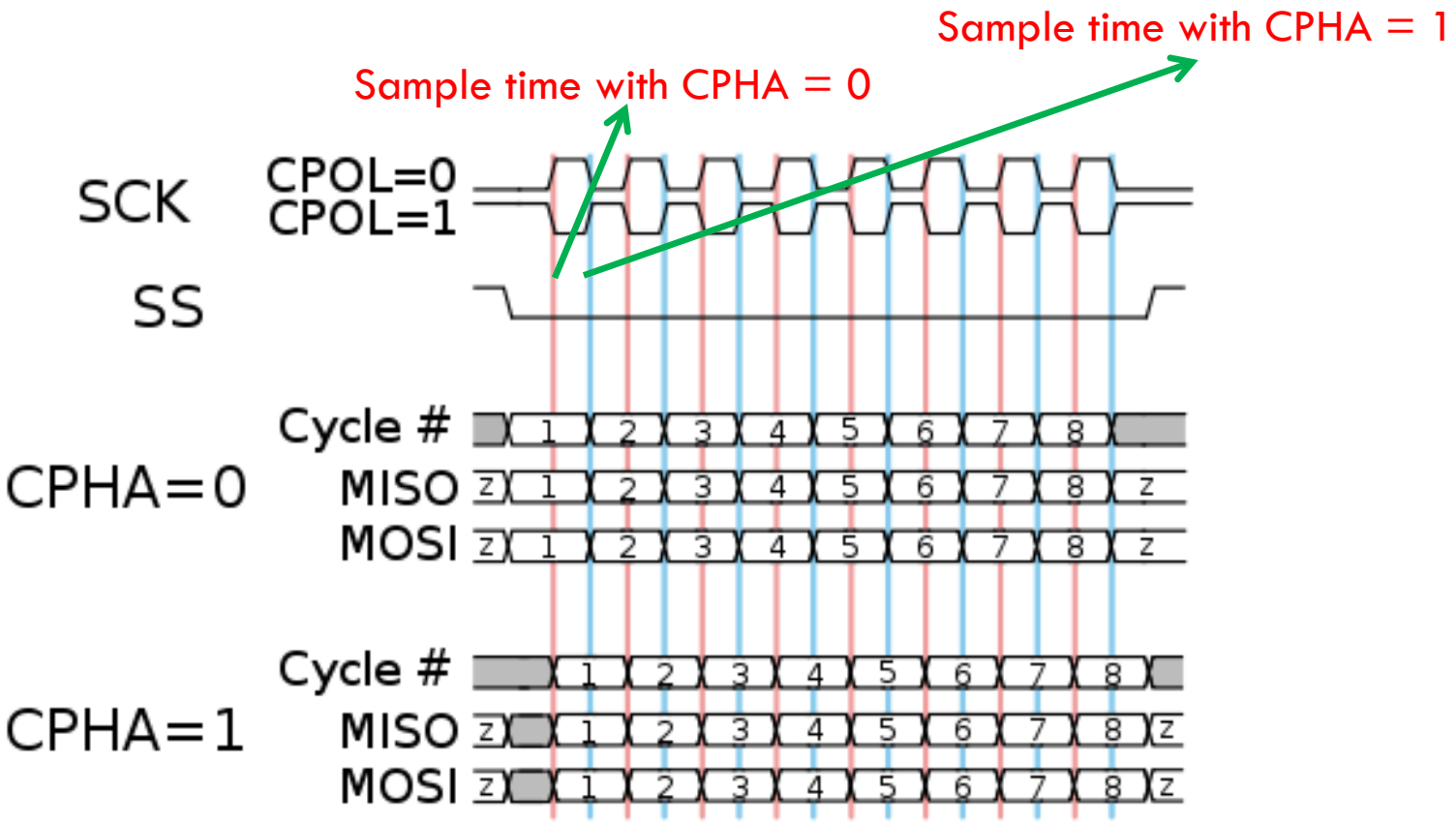


Fig 3. Relations between CPOL and CPHA

Utilize the On-board SPI module

- Our goal in this lab is to configure the on-board SPI module as a master device, and infinitely output serial data on MOSI pin. Finally, we verify the output signal using oscilloscope and shift register.
- In this lab, we set the SPI transfer rate to 1.25MHz, and select both CPOL and CPHA to 0.
- To achieve this goal, we need to:
- 1) Initialize the SPI module
  - Setup pin select to enable SPI
  - Set frequency of the SPI clock signal to 1.25MHz (set transfer rate of SPI to 1.25MHz)
  - Configure the SPI module as a master device
- 2) Send the serial data on MOSI pin by writing it to SPI data register
- 3) Wait until the data transfer is completed (by testing the SPI status register), and then transfer the next byte of data

Check the LPC2103 Spec

- Setup pin select to enable SPI:

  → Set **PINSEL0** to 0101010100000000 (0x5500)

- Set SPI clock frequency to 1.25MHz:

  →According to the lpc2103 spec: **Speed of SPI = APB clock / S0SPCCR value**

  APB clock is the clock frequency for peripheral devices. S0SPCCR is the SPI clock counter register, which indicates the number of APB clock cycles that make up an SPI clock. We set APB to be the same as system clock (10MHz), and S0SPCCR to 8. Therefore, Speed of SPI = 10/8 = 1.25MHz.

  →Set **APBDIV** to 0x1 (the same as system clock)

  →Set **S0SPCCR** to 0x8

- Configure the on-board SPI module as SPI Master:

  →Set **S0SPCR (SPI Control Register)** to 00100000 (0x20)

- Send serial data on MOSI pin:

  →Write serial data byte by byte into the register **S0SPDR (SPI Data Register)**

- Wait until the data transfer is completed before transferring the next byte of data

  →Test 7th bit of **S0SPSR (SPI Status Register)**. If 1, completed. If 0, not completed.

Implement in C

```
#include <arch\philips\lpc2103_k9supd.h>
#include "coridium.h"
void Initialize(void);
int main()
{
    Initialize();

    /* Do forever */
    while(1)
    {
        /* Write data into data register, in this case we use 0xA2 */
        S0SPDR=0xA2;

        /* Wait for transfer to be completed */
        while((S0SPSR & 0x80) == 0){}
    }
}
```

Implement in C

```
void Initialize()
{
    /* Select pins for SPI */
    PINSEL0=0x5500;

    /* Set peripheral clock to same as system clock */
    APBDIV=0x01;

    /* Set the speed of SPI to 1.25 MHz */
    S0SPCCR=0x08;

    /* Device selected as master */
    S0SPCR=0x20;
}
```

Implement in ARM Assembly

```
                    .text  /* Define register addresses to make the code more readable and portable */
                    .equ        PINSEL0, 0xE0020008
                    .equ        APBDIV, 0xE01FC100
                    .equ        S0SPDR, 0xE0020008
                    .equ        S0SPSR, 0xE0020004
                    .equ        S0SPCCR, 0xE002000C
                    .equ        S0SPCR, 0xE0020000
                    .equ        DATA, 0xA2
_start:             .global     _start  /* Define the required routine for ARM processor */
                    .global     main
                    .global     UNDEF_Routine
                    .global     SWI_Routine
                    .global     PAbt_Routine
                    .global     DAbt_Routine
                    .global     FIQ_Routine
                    b           main
UNDEF_Routine:
SWI_Routine:
PAbt_Routine:
DAbt_Routine:
FIQ_Routine:
```

Implement in ARM Assembly

```
main:
        bl      INITIALIZE  /* call initialize(), and store return address in lr */
infinite:
        ldr     r2, =S0SPDR
        ldr     r1, =DATA   /* write the DATA (0xC1) into S0SPDR */
        str     r1, [r2, #0]

        ldr     r1, =S0SPSR /* r1 points to S0SPSR */
waitempty:
        ldr     r2, [r1, #0]  /* test if the 7th bit in S0SPSR is equal to 0*/
        mov     r3, #1
        and     r2, r3, LSL #7
        cmp     r2, #0
        bne     infinite      /* if not equal, break the waitempty, and jump to
        b       waitempty infinite; if equal, keep executing waitempty until
                              not equal */
```

Implement in ARM Assembly

```
INITIALIZE:
        ldr     r2, =PINSEL0
        mov     r1, #0x5500         /* PINSEL0 = 0x5500 */
        str     r1, [r2, #0]

        ldr     r2, =APBDIV
        mov     r1, #0x01           /* APBDIV = 0x01 */
        str     r1, [r2, #0]

        ldr     r2, =S0SPCCR
        mov     r1, #0x08           /* S0SPCCR = 0x08 */
        str     r1, [r2, #0]

        ldr     r2, =S0SPCR
        mov     r1, #0x20           /* S0SPCR = 0x20 */
        str     r1, [r2, #0]

        mov     pc, lr              /* return to main */
```

Verify the Result – Oscilloscope

- For SPI2103, we should **set the SSEL to HIGH** if no slave device is connected.
- Display the waveform of **MOSI** and **SCLK** using oscilloscope.

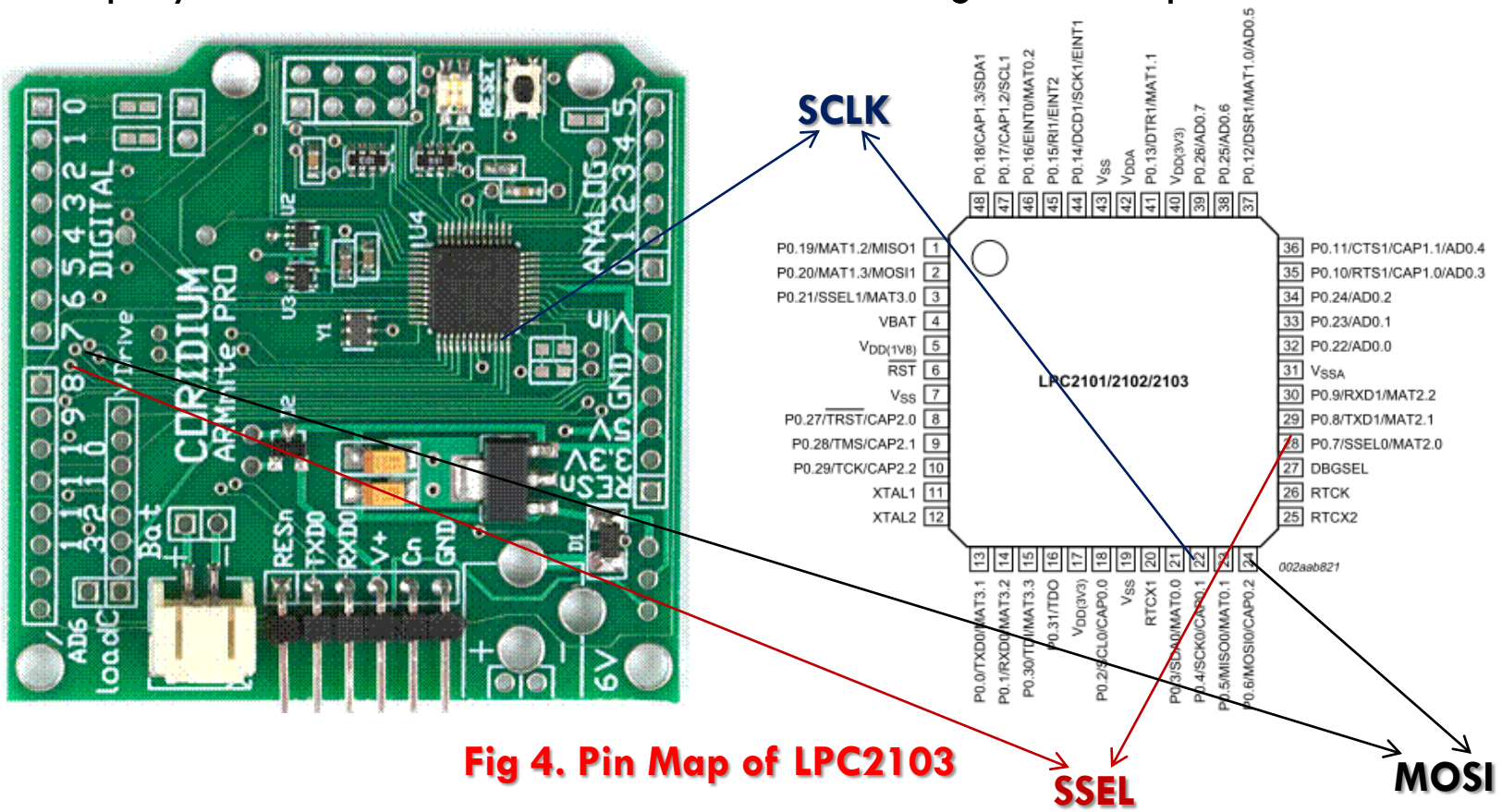
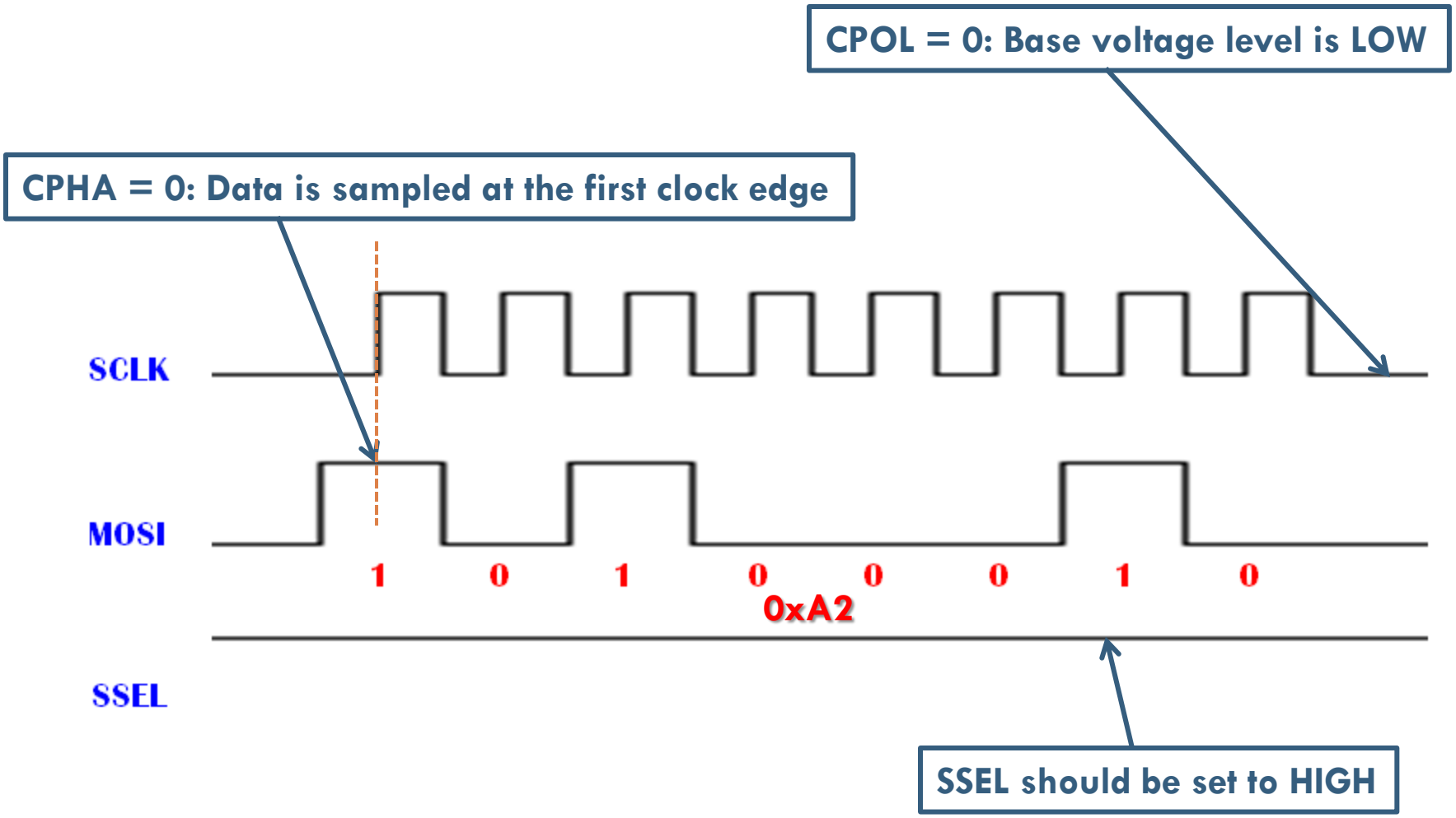


Fig 4. Pin Map of LPC2103

Verify the Result – Oscilloscope



Reference

- http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus
- Philips LPC2103 User Manual
- Philips Application Note 10369
- http://www.allaboutcircuits.com/vol_4/chpt_12/4.html