July 8, 2009

EE 456 **Fall, 2009**
Notes on SPI Bus **Blandford/Mitchell**

The Serial Peripheral Interface (SPI) bus was created by Motorola and has become a defacto standard on many microcontrollers. This is a four wire bus and always transmits data in full duplex mode. Unlike the RS-232 bus and the USB bus, the SPI bus is synchronous so that a clock is sent along with the data.

In the simplest mode of operation the SPI bus facilitates communication between a master and a slave module. The master module initiates the transmission and for every bit transmitted a corresponding bit must be received. The individual modules determine whether or not both bits are needed and are free to discard either. For transmission only, the master transmits a bit but the slave must transmit a dummy reply bit.

There are four signals on the SPI bus as shown in Table 1. If there is only one slave device in the system the SS pin may be tied low making this a 3-wire system. (On some devices which implement this interface, such as some A to D converters, a falling edge is required on SS.) Typically, the MISO line from the slave is in a tri-state mode and is only active when the slave is activated. This permits multiple slaves to share a common MISO line.

SCLK	Serial Clock (from the master)
MOSI	Master Out, Slave In (from the master)
MISO	Master In, Slave Out (from the slave)
SS (or $\overline{SS}$)	Slave Select (from the master and active low)

Table 1
SPI bus signals.

To transmit data the master must be running a clock which can go from 1MHz to 70MHz. The master begins transmission by pulling the SS line low for the slave. The master then sends a bit on the MOSI line which the slave can read. The slave then sends a bit on the MISO line which the master can read. Typically the master and slave have shift registers each of which stores one word but there is no standard word length.

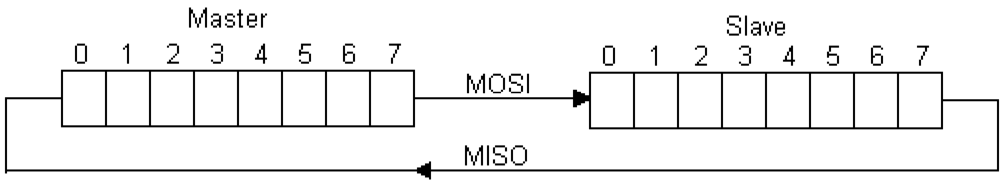


Figure 1
SPI Bus exchange of information between a master and a slave.

There are two SPI configuration signals that must be set up by the user. These two signals have to do with the clock polarity and the clock phase. They are called CPOL and CPHA. CPOL defines the default state of the clock signal in the idle state. These states are defined in Table 2.

The critical factor is that the master and slave must have CPOL and CPHA in the same state.

Mode	CPOL	CPHA	
0	0	0	Data clocked on rising edge and data can change on the falling edge
1	0	1	Data clocked on the falling edge and data can be changed on the rising edge
2	1	0	Data clocked on the falling edge and data can be changed on the rising edge
3	1	1	Data clocked on rising edge and data can change on the falling edge

Table 2
Mode definitions for signal polarity on the SPI bus.

Philips P89LPC916

Figure 2 shows the logic diagram for the LPC916 which implements a 4-wire version of the SPI interface. Many of the pins on the LPC916 are multiplexed to serve multiple functions. The pin diagrams are shown in Figure 3.

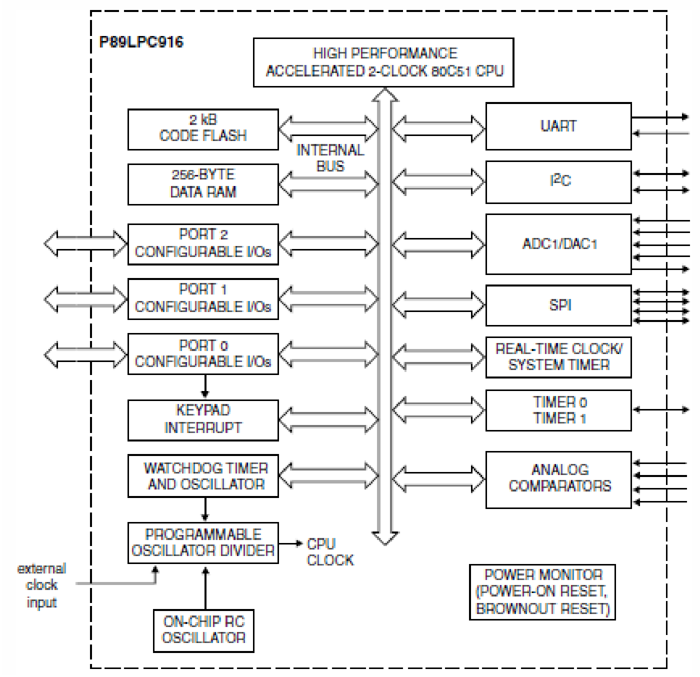


Figure 2
The P89CLPC916 logic diagram.

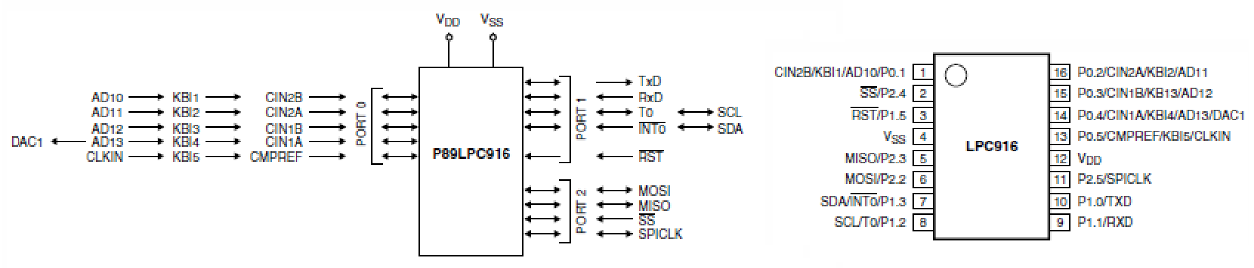


Figure 3
Pin assignments and packaging for the LPC916.

Note that the LPC916 also has a 4-input multiplexed 8-bit A/D converter, a single DAC output (also 8-bit), 2K of flash program memory, and operates from 2.4 to 3.6 volts with 5 volt tolerant I/O pins.

Figure 4 shows the SPI interface logic and pin connections which are multiplexed with pins from Port 2. To configure the SPI channel you need to assign values to the bits in the SPI Control register in Figure 5. In addition to CPOL and CPHA, there is an SPEN bits which enables the channel (SPEN = 1 to enable), a MSTR bit to determine if this module is a master (set to 1) or a slave (set to 0), plus two bits to set the clock rate. SPR1-SPR0 = 00(divide by 4), 01 (divide by 16), 10 (divide by 64) or 11 (divide by 128). The signal SSIG must be set to 0 if the SS line is going to be ignored (3-wire interface). In addition, the DORD bit determines the order in which the data is sent (1 = LSB first, 0 = MSB first).

This implementation has a SPI status register which has a "Finished" bit named SPIF. This bit is set to 1 when serial transmission finishes and can be used to generate an interrupt (Interrupt 9 at 0x4B). Note that the transmission complete flag also indicates that a byte has been received since every bit is sent in full duplex mode.

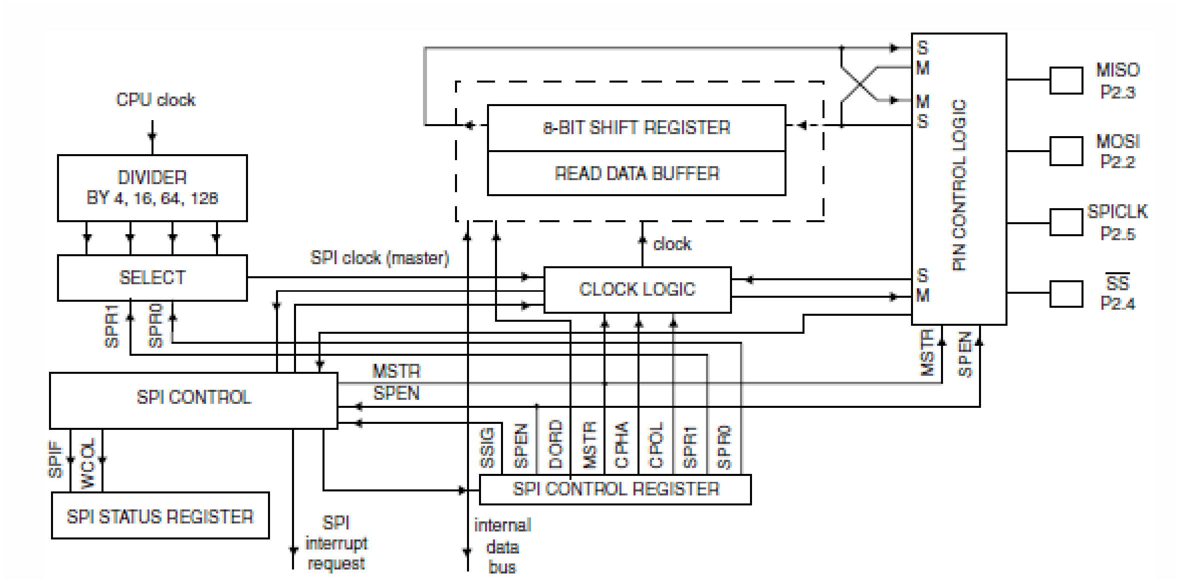
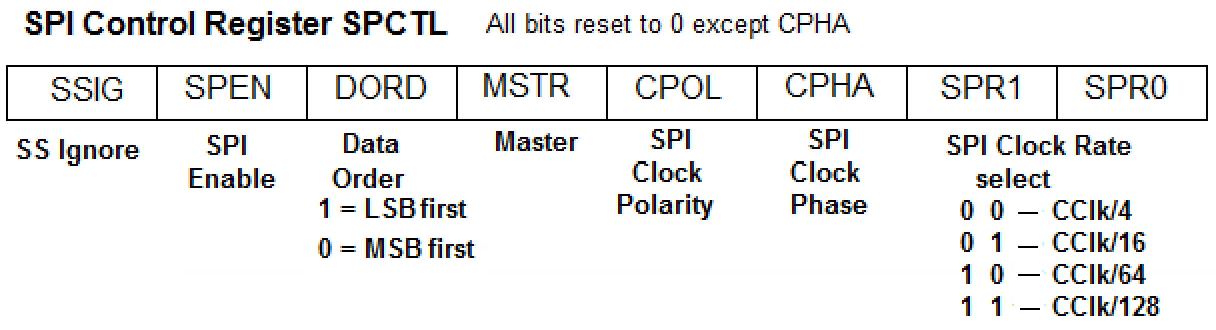


Figure 4
The LPC916 SPI logical interface.



SPI Control Register SPCTL All bits reset to 0 except CPHA

SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
SS Ignore	SPI Enable	Data Order 1 = LSB first 0 = MSB first	Master	SPI Clock Polarity	SPI Clock Phase	SPI Clock Rate select 0 0 — CClk/4 0 1 — CClk/16 1 0 — CClk/64 1 1 — CClk/128	

Figure 5
The SPI Control register (SPCTL). None of the bits in this register are bit addressable.

In addition to the SPCTL register, there are two other registers which are needed to make full use of the SPI interface: SPSTAT is the SPI Status register and SPDATA is the SPI data register. The SPSTAT register has just two bits: SPIF is the *transfer completion flag* which is set to 1 when the present data byte has been transferred. It must be cleared by software and curiously, this is done by writing a one to this bit. WCOL is the *write collision* indicator. This bit is set if the user attempts to write to the SPDATA register before the transfer is complete. This bit too must be cleared by software by writing a 1 to the bit.

The SPDATA register is an 8-bit register that holds the data to be transferred. The user can write and read this register.

The program on the following page can be executed on a P89LPC916. The device is set as a master and the bit pattern 0x55 is sent out along with the clock to the SPI port.

July 8, 2009

```
/* SPISftWrPrj.c
 * This program runs on a P89LPC916.  It sets the processor
 *   up as an SPI master and transmits the bit pattern
 *   0101 01010 out on the SPI port.
*/
#include<Reg916.h>
void InitializeSPI(void);
void main(void)
  {InitializeSPI();
   SPSTAT |= 0x80;    //Clear transfer complete flag
   while(1)
     {SPDAT = 0x55;  //Send out 0101 0101
      while(!(SPSTAT & 0x80)); //Wait for transfer completion
      SPSTAT |= 0x80; //Cleaer transfer complete flag
     }
  }

void InitializeSPI()
  {SPCTL = 0x54;  //use SS, Enable, LSB first, Master, CPOL/CPHA = 01, CClk/4
  }
```