Timing and Synchronous Serial Communication

- Finish up timing discussion:
  - Compare Mode
  - Capture Mode
  - Scheduling
- Synchronous serial communication
  - SPI protocol

Administrivia

- Lab:
  - Lab exercise 3 this week (you must be checked off on this)
  - Friday: Milestone 3 (PWM control). See the main web page for some suggestions
- Homework:
  - Homework 1 solutions are available on the web page
- Talk:
  - " Privacy Protection for Ubiquitous RFID Tags"

    Ari Juels

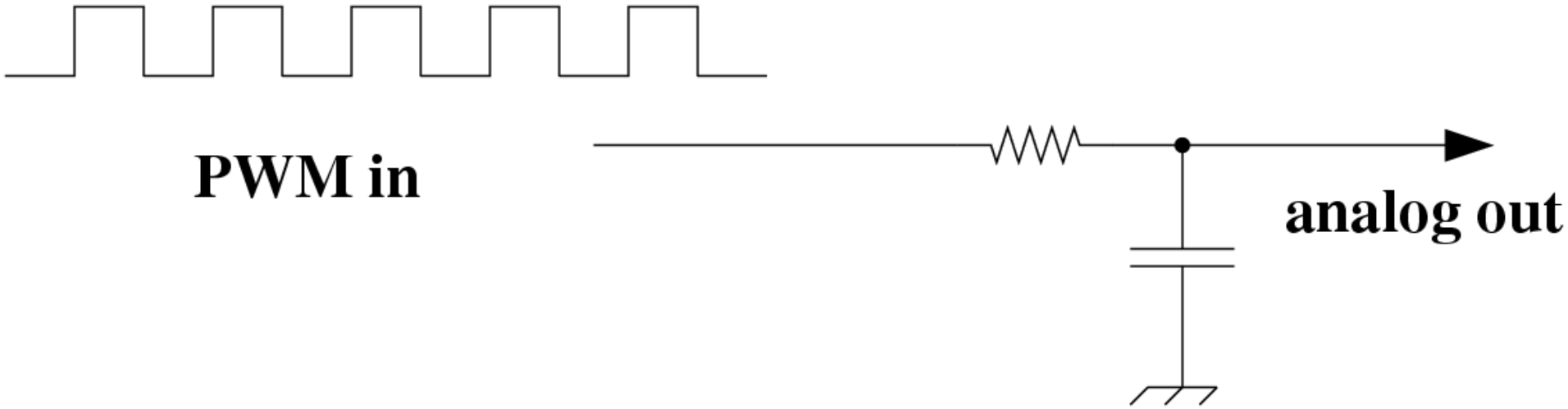
    Friday, Oct 24, 11:00, CS 151

Actuator of the Day...

Actuator of the Day

Piezoelectric speaker...

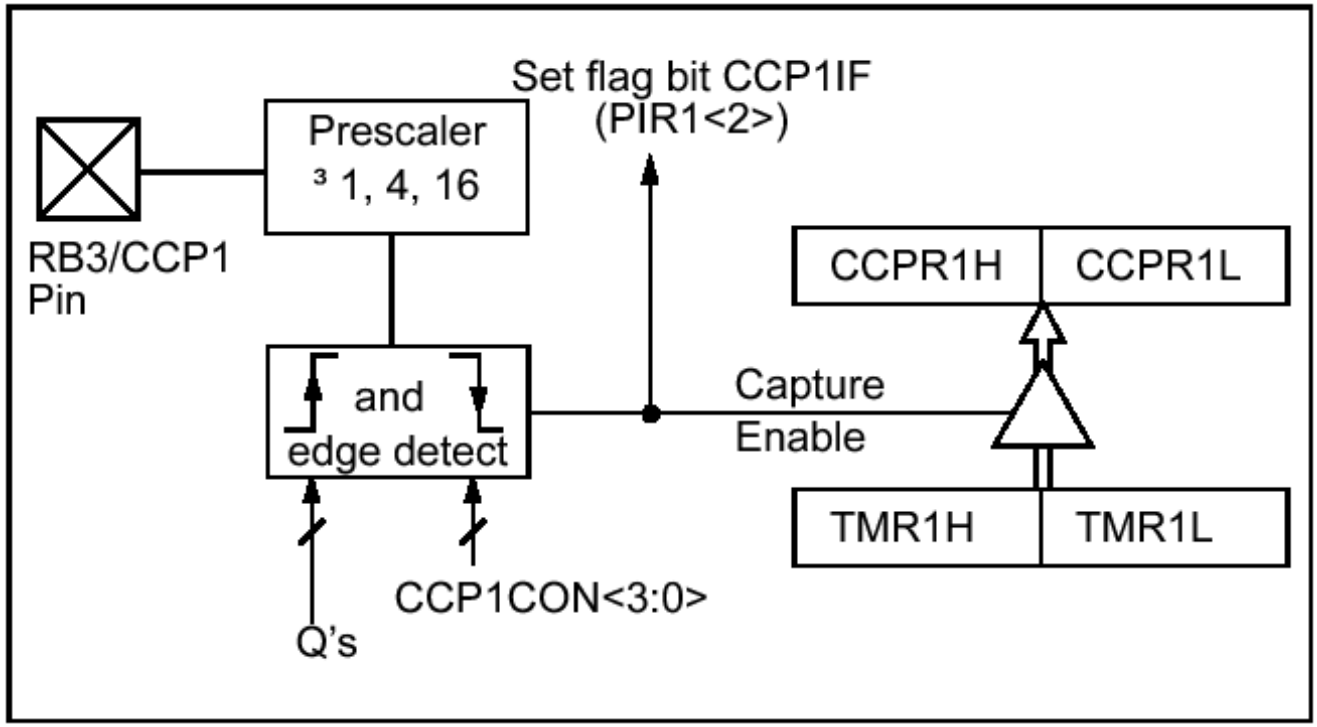
PWM Approach to Digital-to-Analog Conversion



- R-C circuit smooths the digital input
- Tune R-C parameters to match the frequency of the PWM signal (lab exercise 3)

CCP Capture Mode

- Timer 1 must be running
- When the specified event occurs on RB3, the Timer 1 register value is copied into CCP_x



CCP Capture Mode

```
long tmp;
CCP_1 = 0; tmp = 0;
setup_ccp1(CCP_CAPTURE_RE);
setup_timer_1(T1_INTERNAL |
  T1_DIV_BY_8);

while(tmp == CCP_1){
    // Do something
};
// Capture has happened
```

- CCP_1 is a global long defined by the compiler

PWM Measurement

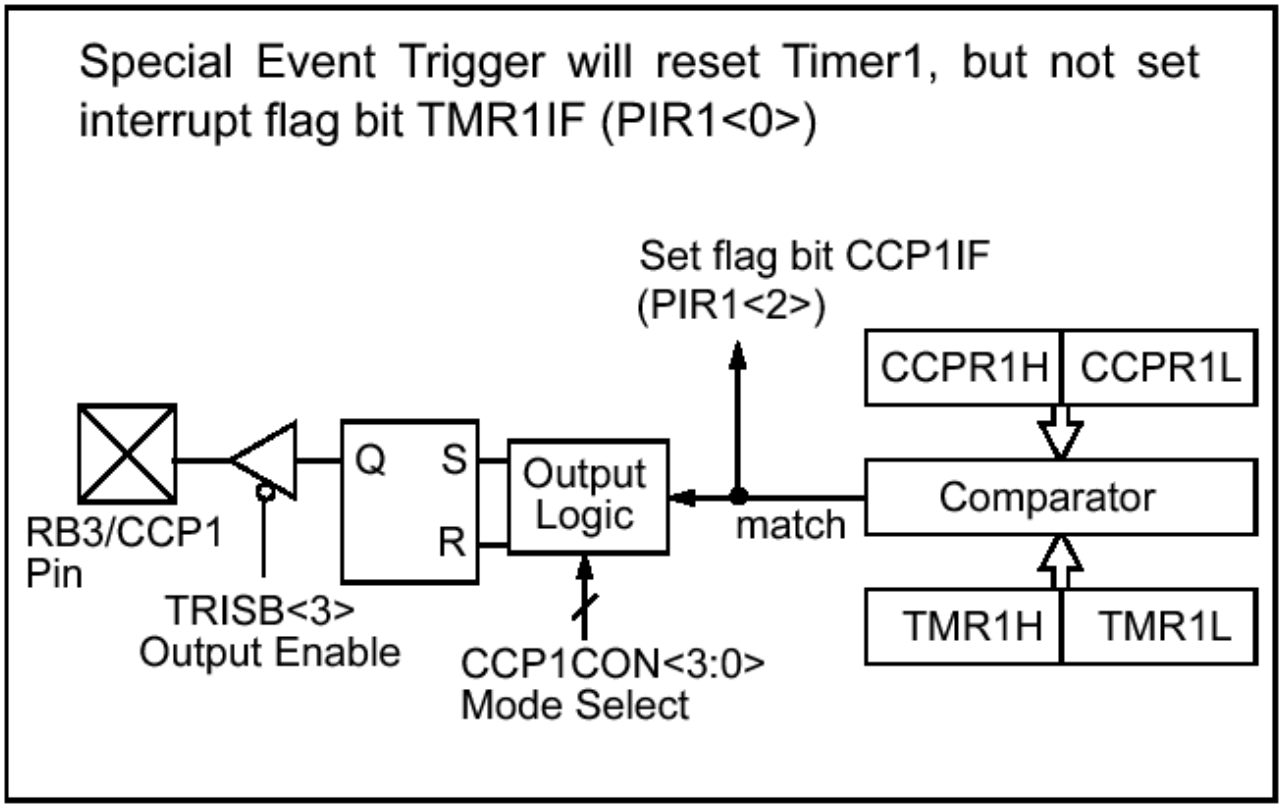
- Configure CCP_1 to capture on rising edge
- Configure CCP_2 to capture on falling edge (note that the 628s only have one CCP module)

- When CCP_2 captures, the pulse width can be measured using:

```
pulse_time = CCP_2 - CCP_1;
```

CCP Compare Mode

When Timer 1 register matches a predefined value, then set or reset the CCP pin



CCP Compare Mode

Initialize:

```
setup_ccp1(CCP_COMPARE_CLR_ON_MATCH);
setup_timer_1(T1_INTERNAL |
  T1_DIV_BY_8);
// Make sure that the CCP_1 pin is
//   configured as an output

// Remember to initialize the pin
//   to low
```

CCP Compare Mode

On some event, generate a pulse of a specific width:

```
setup_ccp1(CCP_COMPARE_SET_ON_MATCH);
CCP_1 = 0;
set_timer1(0);  // Force match: pin
                //   goes high
CCP_1 = 500;  // New timeout
setup_ccp1(CCP_COMPARE_CLR_ON_MATCH);
// Pin will go low after delay
```

Scheduling A Long Task at Regular Intervals

If we want to keep ISRs short, how do we schedule a long task at regular intervals?

- For example, a 7-9 ms task at intervals of 10 ms

Scheduling A Long Task at Regular Intervals

One solution:

- Configure Timer 1 to generate an interrupt at 1 ms intervals
- int_timer1: increment a clock that turns over at 250

Scheduling Long Tasks

The main program:

```
while(1) {
  // Wait for clock to turn over
  while(clock % 10){};
  do_the_task();
}
```

Must ensure that the task does not require more than 10 ms (including background activities)

Scheduling Multiple Tasks for Different Intervals

Suppose two different tasks:

- One that must be executed at intervals of 10 ms
- One that must be executed every 5 ms

- How do we handle this in the main program?

Scheduling Multiple Tasks

```
while(1) {
   if(clock % 5){
      do_task_1();
   }
   if(clock % 10) {
      do_task_2();
   }
}
```

Will this work?

Scheduling Multiple Tasks

```
while(1) {
   if(clock % 5){
      do_task_1();
   }
   if(clock % 10) {
      do_task_2();
   }
}
```

Only if we can ensure that both tasks require less than 1 ms. Why?

Scheduling Multiple Tasks

```
next_1 = next_2 = clock;
while(1) {
   if(clock >= next_1){
      do_task_1();
      next_1 += 5;
   }
   if(clock >= next_2) {
      do_task_2();
      next_2 += 10;
   }
}
```

Scheduling Multiple Tasks

- One caveat: must also handle the roll over of the clock
  - Make sure to handle the critical section problem

- How would we handle tasks (differently) that need to be scheduled every 5 and 7 ms?

Scheduling Multiple Tasks

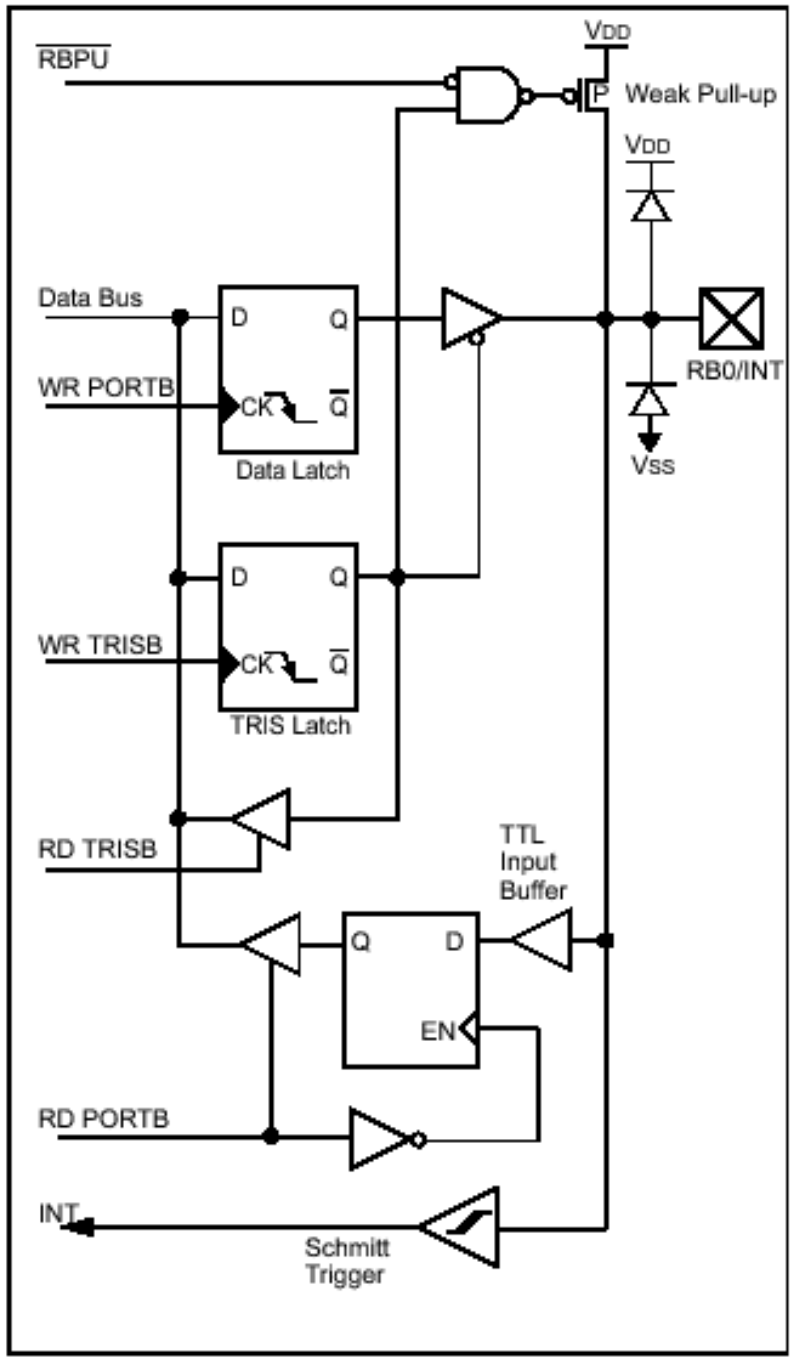
How would we handle tasks that need to be scheduled every 5 and 7 ms?

- Make sure that the clock rolls over at an increment of 35

External Interrupt Pin

- RB0/INT
- Can generate interrupt on either transition (up or down)
- Schmitt triggered input

FIGURE 5-8: BLOCK DIAGRAM OF RB0/INT PIN



Timer 0 Note

- Can handle external interrupts of up to 50 Mhz (remember that it has a prescaler of up to 256)

Port B Change Interrupt

If pins RB4 – RB7 change, an interrupt may be generated:

- Define the INT_RB ISR
- Enable INT_RB and INT_GLOBAL
- In order to determine which pin changed, you must poll them and compare their state against your own record of the prior state

Digital Communication

- What is serial communication?
- How does it differ from parallel communication?
- Why would we consider using serial communication?

Digital Communication

- With parallel communication:
  - A set of bits is transmitted simultaneously over a set of signal lines
- With serial, we order the bits in time
  - How they are ordered depends on the protocol
  - Requires more time to send the same amount of data
  - But: only requires a small number of signal lines
  - The difference in hardware cost implies serial is most common

Translating a Bit into a Signal on a Line

How do we do this?

Translating a Bit into a Signal on a Line

How do we do this?

- Hold the line low or high depending upon the value of the bit
- We do this for some amount of time

- Challenge:
  - How does the receiver know when to read the bit value from the line?

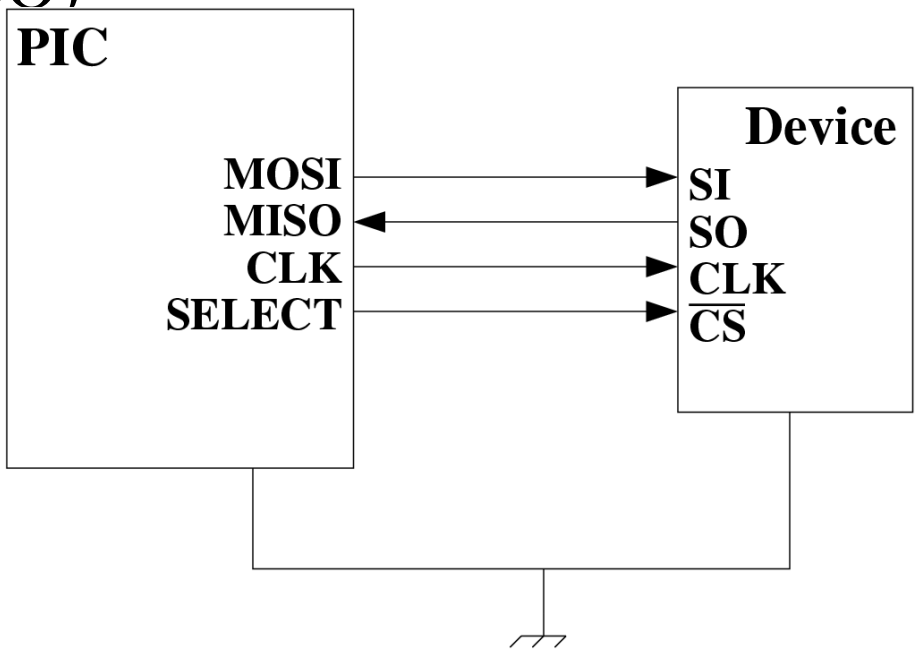
Translating a Bit into a Signal on a Line

How does the receiver know when to read the bit value from the line?

- We must have some way to agree on when a value is valid
- In **synchronous serial** systems:
  - Another signal line carries a **clock signal**
  - The two sides agree that when the clock transitions, the bit value is valid

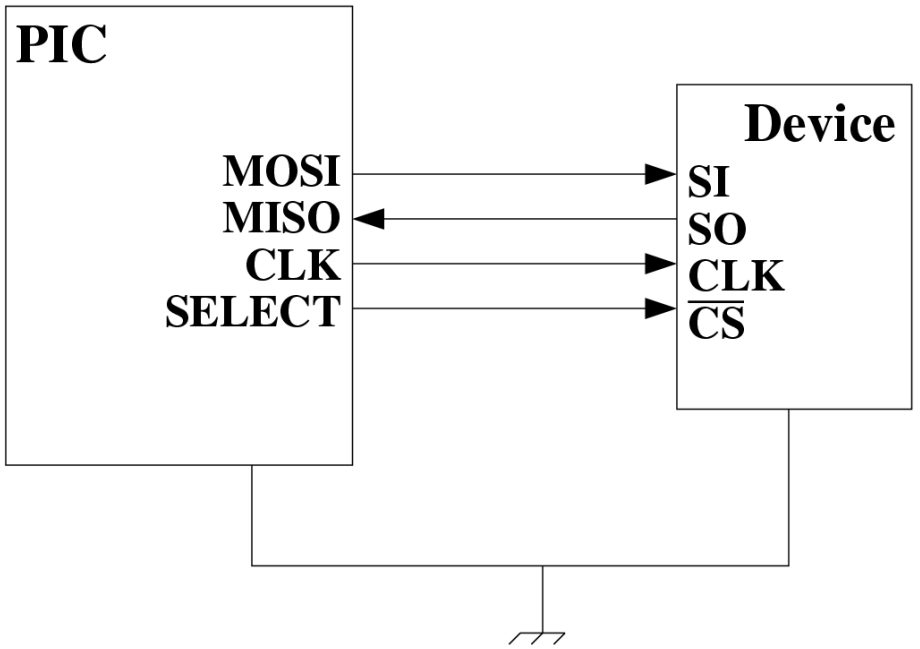
Serial Peripheral Interface (SPI)

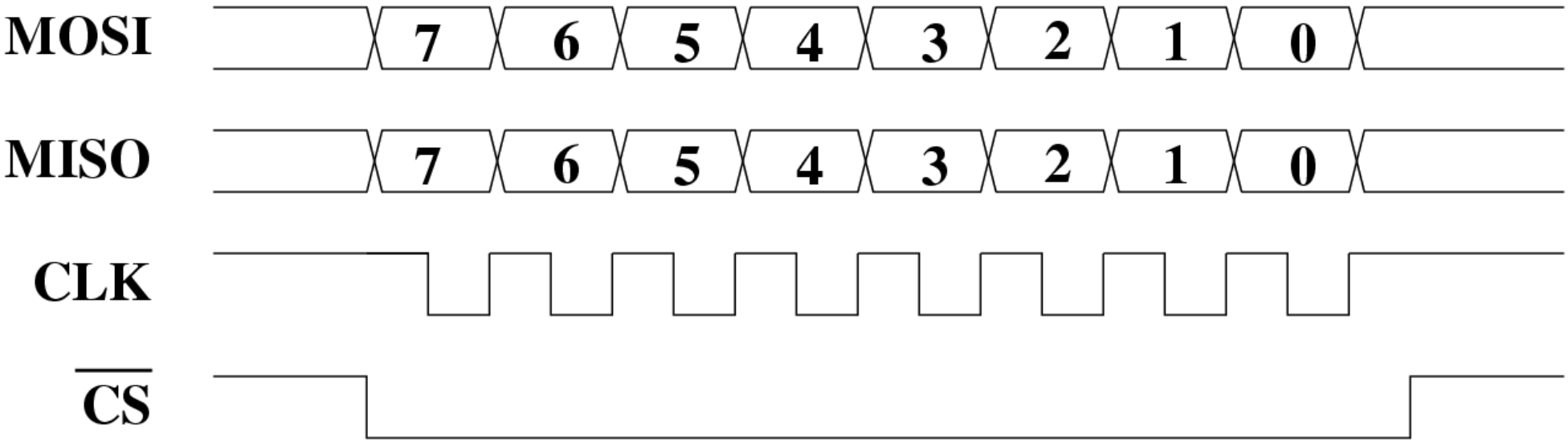
- Bidirectional transmission of data
- 4 connections:
  - Master-Out-Slave-In (MOSI)
  - Master-In-Slave-Out (MISO)
  - Clock
  - Select
- Don't forget the common



SPI Timing

- The master and slave transmit their bytes simultaneously
- CS = chip select





SPI Protocol Variations

- Some devices can accept multiple bytes
  - EEPROM devices might accept many 1000s of bytes
- Clock polarity can be reversed
- Data shift can occur on a downward clock edge or an upward edge
- In some cases, we do not use the chip select signal

SPI Support in the PICs

- Only the 16f876 and the 16f88 chips support SPI (the 628s do not)
- Pins of interest:
  - SCK: clock in/out
  - SDO: serial data out
  - SDI: serial data in
  - SS: slave select (slaves only). For a master, you must allocate some digital output(s) to generate the select signals.

Next Time

- Lecture Thursday:
  - SPI and I2C
  - Make sure you have read DEH chapter 9 and DwPICM chapters 7 and 9
- Lab Friday:
  - Milestone 3 is due
  - Lab exercise 3 must be completed