Module 11. Intro to Digital Interfaces

Digital Interface: Connection between two or more digital devices.

Types of Digital Interfaces

There are two classifications for how data is transmitted in a digital interface: serial and parallel

Parallel Interfaces = Each bit has its own electrical connection (interconnect, trace, wire)

>> Advantages = Fast, easier to synchronize (1 clock edge transfers all bits together)

>> Used almost exclusively inside a CPU (or other) chip

>> Disadvantage = Each bit must have its own electrical connection

Serial Interfaces – Bits sent one after another along a single connection

>> Used almost exclusively to make connections off-chip (and off-computer, through the Internet, out to the Mars Rover, etc.)

>> Advantages = Simpler/fewer connections between CPU and peripheral (2-4 lines)

Examples include:

Device Select/Enable (CS)

Synchronizing CLK (SCLK)

Data Line(s) (SDI and SDO)

Common ground (GND): Often already established if devices are on the same board or IC

>> "Connection" = PCB trace, wire, RF, acoustic, optical, etc.
>> Disadvantages = "Slower", more complicated synchronization, potential timing issues

How are digital interfaces implemented?

Most microcontrollers contain hardware peripherals that implement the interface in hardware. On the MSP430, we have hardware peripherals to implement a few types of serial interfaces.

On the MSP430: Universal Serial Communications Interface (UCSI)

>> Basically acts as a parallel-to-serial and serial-to-parallel converter

>> Most modern microprocessors/microcontrollers will have built-in Serial interface
-- MSP430F5229 has a total of four, in two types

>> Role of Serial interfaces has grown with growing sophistication and speed of serial links (SPI and I^2C to USB and others)

We will discuss three types of interfaces: UART, SPI, and I2C. There are many more types of digital interfaces!

What kind of information is exchanged?

Just like in a CPU, information in a digital interface is represented in bits. The interface defines the manner in which bits are transmitted:

Types of Interfaces

UART

>> UART: Universal Asynchronous Receiver/Transmitter
>> UART mode configures basic 2-wire *asynchronous* serial communications
>> Connect to UCSI with 2 external pins (MSP430: `UCAxRXD` and `UCAxTXD`)

>> Not synchronous (no shared clock) = Asynchronous

>> To use serial communications both devices must know data format and baud rate

- These are set using UARTs control registers
- Implies make data format & baud rate decisions at design time

UART: Fundamental Parameters

In order to use UART for an application, you need to determine the following parameters:

- Start Bit = 1 bit (“low”)
- Data Bits = 7 or 8 bits
- Parity = Even, Odd, or None
  >> Even Parity = 1 when number of 1's including parity is even
  >> Odd Parity = 1 when number of 1's including parity is odd
- Stop bit(s) = 1 or 2 bits (“high”)
- Baud rate (bits/sec): Common rates are 200, 2400, 9600, 19200, 115200

RS-232 – “Old standby” for serial format →Actually is a specific standard with associated voltage ranges and baud rated now often misused to mean any asynchronous serial communication

-->Data sent Least Significant Bit (LSB) first**

** Most UARTs send asynchronous serial data LSB first (because RS-232 is LSB first) but MSP430F5229 UCSI_A is configurable and can be set to send MSB first, so that it can be compatible with various types of serial interfaces.

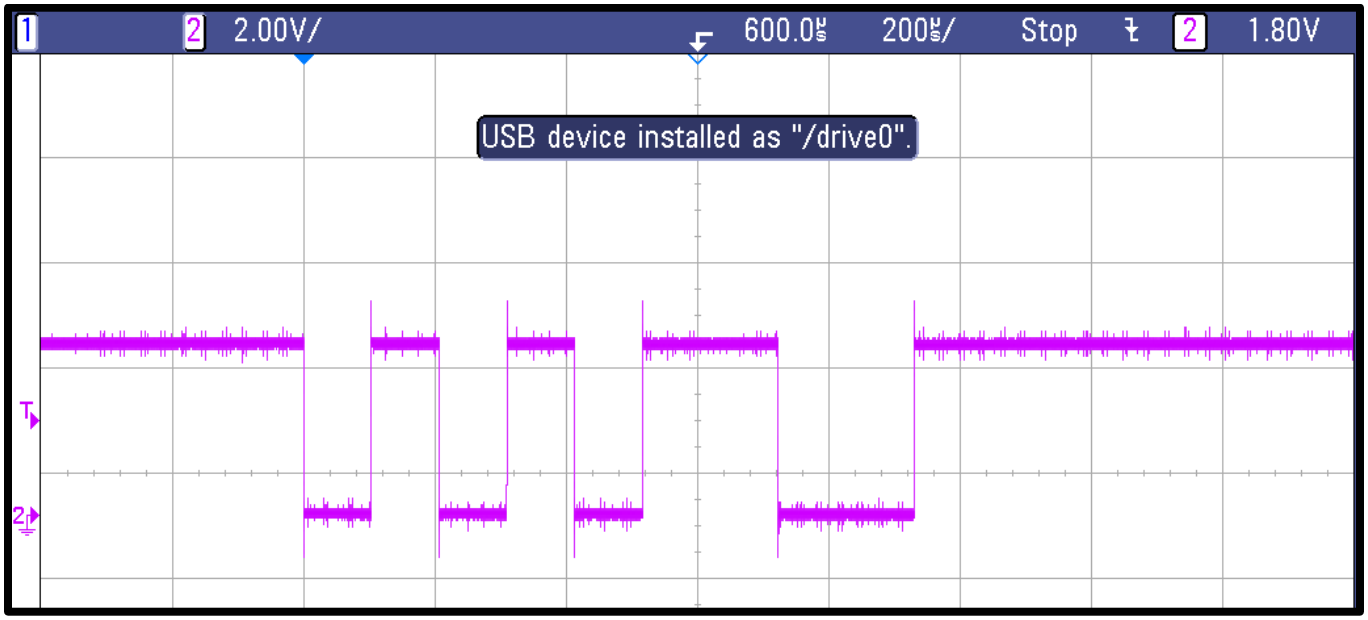
Common Baud Rates

Table 36-4. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 0

BRCLK Frequency (Hz)	Baud Rate (baud)	UCBRx	UCBRSx	UCBRFx	Maximum TX Error (%)		Maximum RX Error (%)	
32 768	1200	27	2	0	-2.8	1.4	-5.9	2.0
32 768	2400	13	6	0	-4.8	6.0	-9.7	8.3
32 768	4800	6	7	0	-12.1	5.7	-13.4	19.0
32 768	9600	3	3	0	-21.1	15.2	-44.3	21.3
1 000 000	9600	104	1	0	-0.5	0.6	-0.9	1.2
1 000 000	19200	52	0	0	-1.8	0	-2.6	0.9
1 000 000	38400	26	0	0	-1.8	0	-3.6	1.8
1 000 000	57600	17	3	0	-2.1	4.8	-6.8	5.8
1 000 000	115200	8	6	0	-7.8	6.4	-9.7	16.1
1 048 576	9600	109	2	0	-0.2	0.7	-1.0	0.8
1 048 576	19200	54	5	0	-1.1	1.0	-1.5	2.5
1 048 576	38400	27	2	0	-2.8	1.4	-5.9	2.0
1 048 576	57600	18	1	0	-4.6	3.3	-6.8	6.6
1 048 576	115200	9	1	0	-1.1	10.7	-11.5	11.3
4 000 000	9600	416	6	0	-0.2	0.2	-0.2	0.4
4 000 000	19200	208	3	0	-0.2	0.5	-0.3	0.8
4 000 000	38400	104	1	0	-0.5	0.6	-0.9	1.2
4 000 000	57600	69	4	0	-0.6	0.8	-1.8	1.1
4 000 000	115200	34	6	0	-2.1	0.6	-2.5	3.1
4 000 000	230400	17	3	0	-2.1	4.8	-6.8	5.8
4 194 304	9600	436	7	0	-0.3	0	-0.3	0.2

Example: How long would it take to transmit “Term is over!” at 9600 baud with 1 start bit 1 stop bit and even parity assume 8-bit ASCII encoding?

An Example UART Transmission



What does this mean?

We have no idea unless we know the parameters used for the transmission. In this case, the parameters are: 9600 baud, 1 start bit, 1 stop bit, and no parity bits.
What data is being transmitted?

Serial Peripheral Interface Bus (SPI)

>> Used primarily for synchronous serial communications between a CPU and peripherals "within the box"

-- Synchronous = shared clock (supplied by master device)

>> Usually a 4 wire connection (sometimes 3-wire)

SIMO = Slave In/Master Out data line

SOMI = Slave Out/ Master In data

SCLK = Serial Clock (Called `UCAxCLK` in MSP430 Documentation)

CS = Chip Select

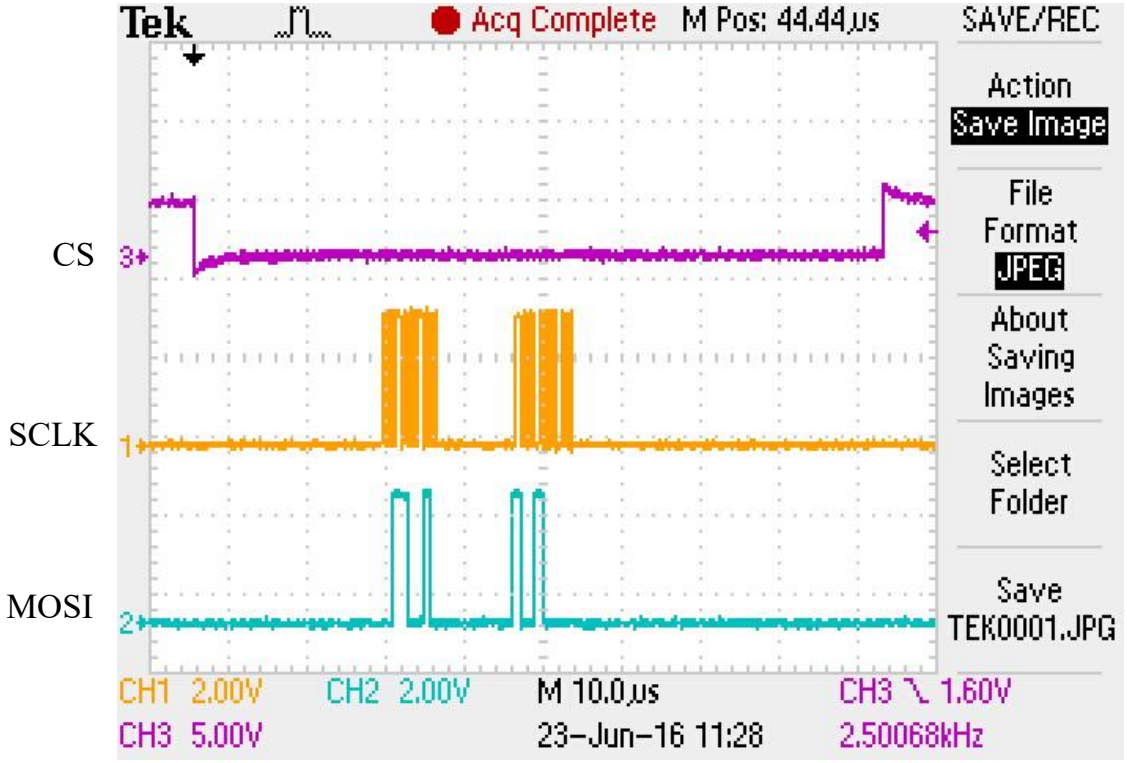
>> SPI is somewhat loose standard --> *Different* from I^2C

How will you know what to use?

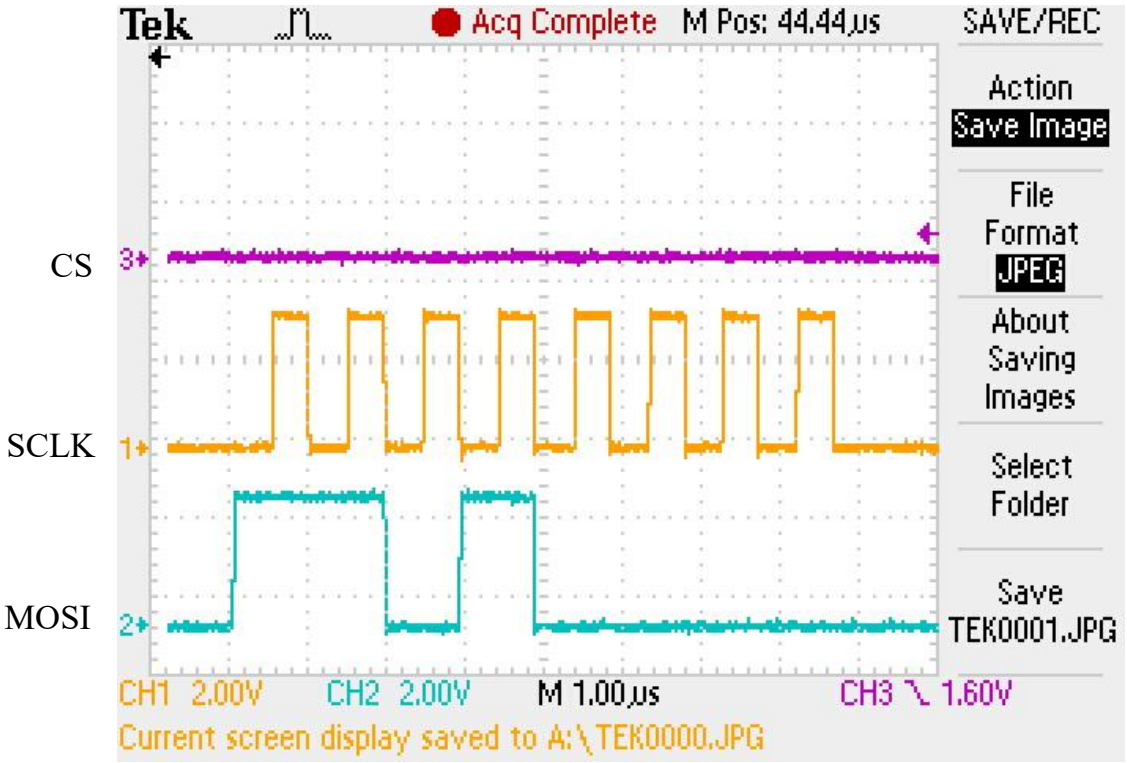
--> SPI, I^2C, asynchronous serial (RS-232), other?
Sensors or other peripheral devices will specify the interfaces with which they are compatible

An Example SPI Transmission

From a high level:



If we zoom in on just 8 bits…



What do we need to know to decode this?

>> ***To use SPI the programmer must...***

1) Enable USCI_A or USCI_B for SPI mode (and set pins for function mode)

2) Select data format (data size, clock edge)

3) Configure a clock frequency

When communicating with a device, the programmer must also ...

4) Select desired SPI peripheral using its chip select (CS)

First, what is the function of a CS?

>> Likely to be multiple peripherals using SPI bus

>> Only 1 Slave device and Master (MSP430) can use SPI bus at a time

>> CS is typically an ACTIVE LOW signal implemented using a Digital IO pins
CS = 0 = Device is Enabled (will read and write to SPI data lines)
CS = 1= Device is Disabled (outputs are high impedance)

Example SPI Peripherals

Our lab boards have 2 peripherals that are SPI devices, the LCD screen and the digital-to-analog converter (DAC). However, we could readily connect others as the USCI pins and some digital IO pins are available through the headers

Sharp 96x96 LCD Display

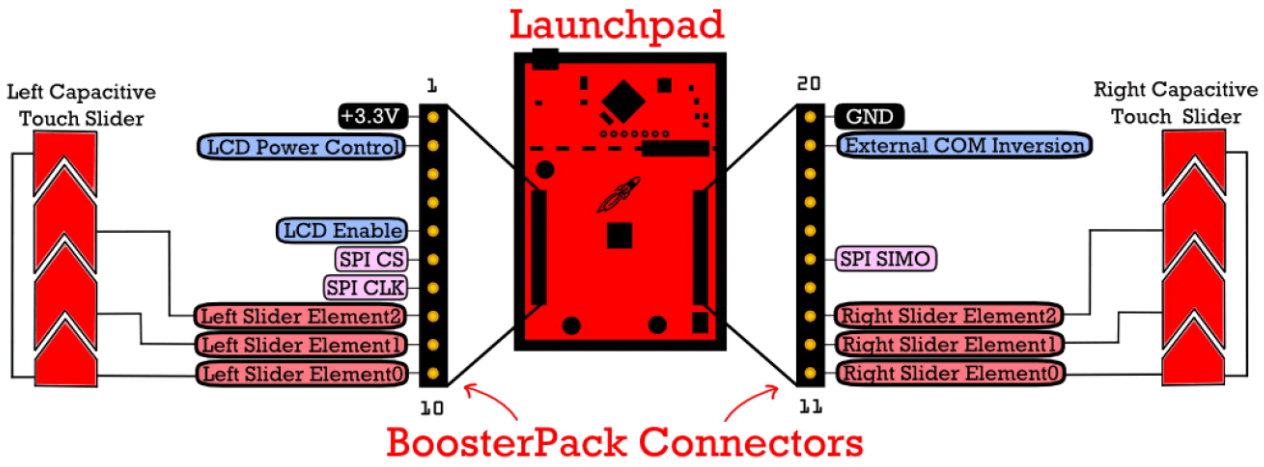


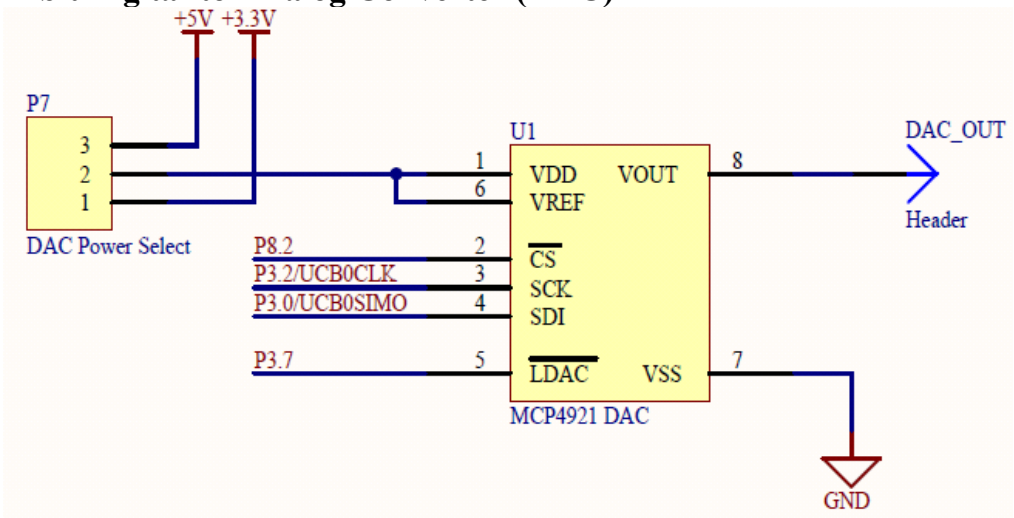
Figure 3. BoosterPack Default Pinout

Interface Pins

- **SPI CS: P6.6 (Digital I/O)**
- **SPI CLK: P3.2 (Function mode, UCB0CLK)**
- **SPI SIMO: P3.0 (Function mode, UCB0SIMO)**
- Two additional digital I/O pins to provide power and enable LCD (P6.5 and P1.6)

Note: LCD does not send data to the MSP430, so the SOMI line is not used!

MCP4921 12-bit Digital-to-Analog Converter (DAC)



Digital to Analog Converter: Converts 12-bit digital code into an analog voltage (in some range V_{Ref+} to V_{Ref-}). Sound familiar?
Can use this to generate analog signals!

Example Data format: MCP4921 Digital to Analog Converter (DAC)

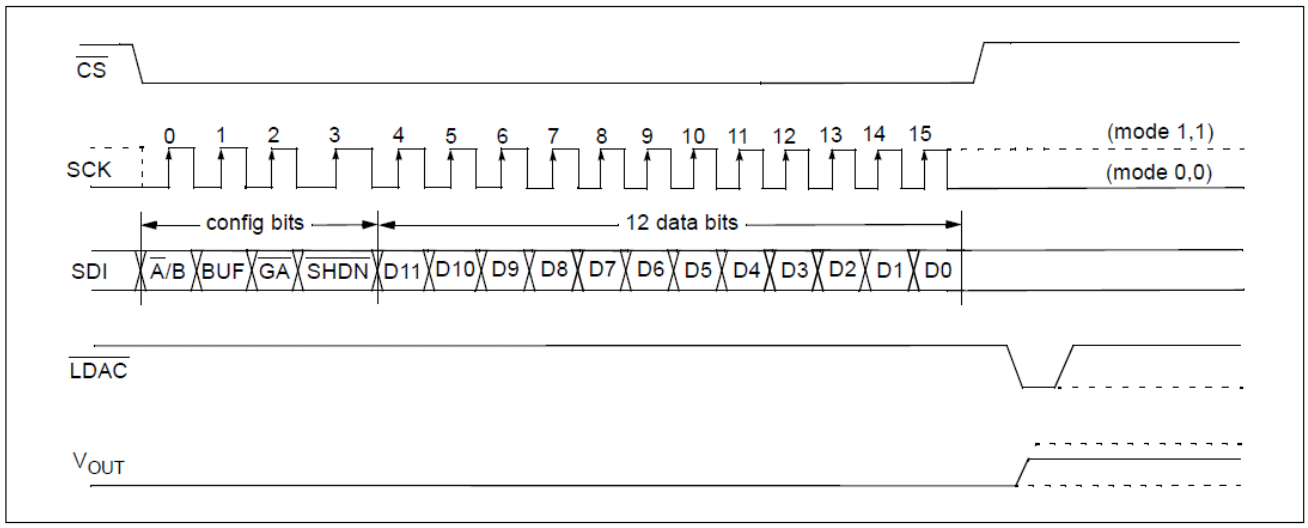


FIGURE 5-1: *Write Command.*

How do we connect multiple devices to the SPI bus?

- The "bus" lines (SCLK, MOSI, MISO) can be shared across all connected devices
- Each device needs its own chip select (CS) line, which tells which device when to wake up and watch the bus for input (or to write some output)

Inter-Integrated Circuit (I^2C)

I^2C (also, I2C) is another increasingly common serial interface. I2C provides a synchronous interface using only two wires!

An I2C interface is comprised of two wires, called the "I2C bus":

- SCL: Serial clock
- SDA: Serial data

I2C bus fundamentals:

- All peripherals are connected to the same two wires
- Both SCL and SDA require *pull-up* resistors such that both lines default to logic high.
- Both lines are *bidirectional*
- Speeds are standardized: typically 100kbps (standard mode) or 400kbps (fast mode)

Unlike SPI, I2C is more rigidly standardized and has strict timing requirements on how the two lines can be used. There are a few standard operations:

- START
- STOP
- ACK: Acknowledge transmission
- NACK: Negative acknowledgement

Every device datasheet will tell you how and when it expects to receive these commands.

How are devices selected?
With only two wires, we have nothing like a chip select (CS) to wake up individual devices! In I2C, every device has an *address*, which is often programmed at the factory.

Before any transmission, the master sends the address of the device it wants to use—the device should only respond to the request if it has a matching address.