sparkfun

Shift Registers

CONTRIBUTORS: *JORDANDEE*

FAVORITE 2

Overview

Have you ever found yourself wanting to control lots of LED's? Or just needed more I/O in general? Well, this tutorial will cover the basics you need to know about a technology that will let you do just that. It's called the shift register. So what exactly is it? Why are they useful? How do I use it? These are all questions we will attempt to answer in this tutorial.

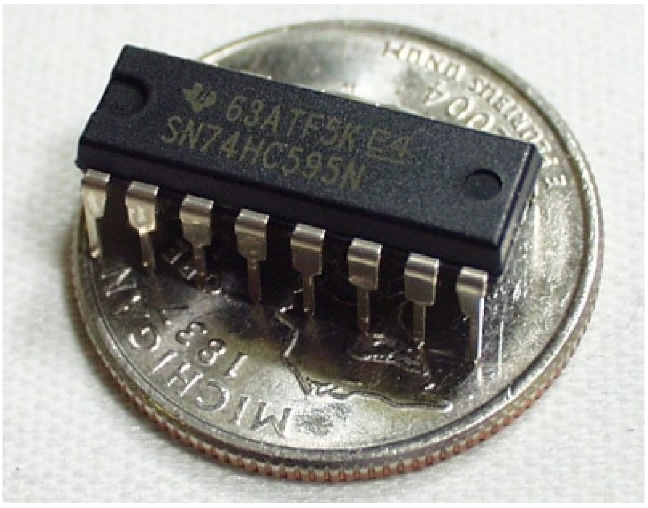
Suggested Reading

Here is some background material you may consider reading before moving on:

- Logic Levels
- Binary
- Serial Communication
- Series vs Parallel
- How to Use a Breadboard

What is a Shift Register

A shift register is a device that allows additional inputs or outputs to be added to a microcontroller.

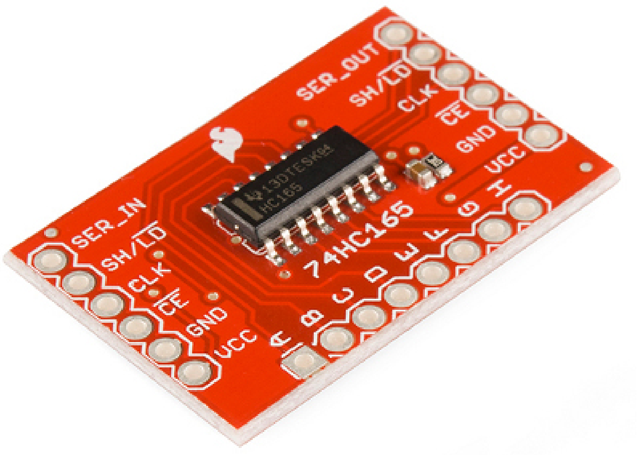


The 74HC595 Shift Register in a DIP package.

This is accomplished by converting data between parallel and serial formats. A microprocessor communicates with a shift register using serial information, and the shift register gathers or outputs information in a parallel (multi-pin) format.

Got a lot of buttons? Consider using a shift register.

Shift registers come in two basic types, either SIPO, Serial-In-Parallel-Out, or PISO, Parallel-In-Serial-Out. SparkFun carries both types. Here is a SIPO, the 74HC595, and the PISO, the 74HC165. The first type, SIPO, is useful for controlling a large number of outputs, including LEDs, while the latter type, PISO, is good for gathering a large number of inputs, like buttons.



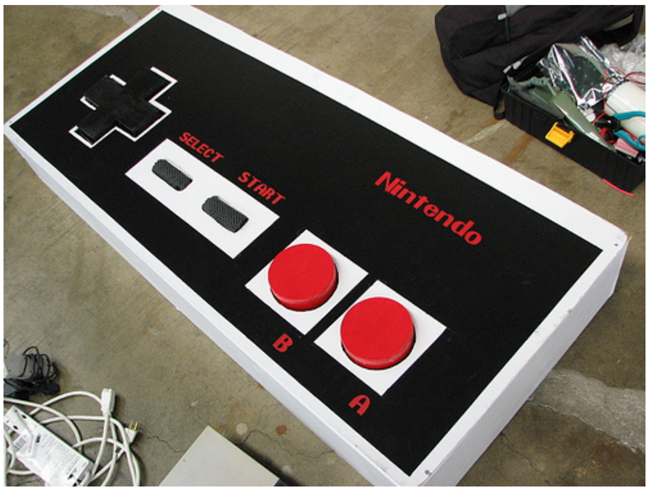
A breakout version of the 74HC165.

SparkFun carries an easy to use breakout versions for both these chips as well. If you need more than 8 additional I/O lines, you can easily chain multiple shift registers together by connecting the output side of the breakout board to the right side of another board.

Why Shift Bits

Shift registers are often used for the purpose of saving pins on a microcontroller. Every microcontroller has a limited number of pins for general inputs and outputs (GPIO).

If a project needs needs to control 16 LEDs, that would normally require 16 pins of a microcontroller. In the event that you don't have 16 available I/O pins, this is where the shift register comes in handy. With two shift registers connected in series, we can accomplish the task of controlling the 16 LED's with only using 4 I/O pins. That is quite a difference, and you can save even more pins the more shift registers you have chained together.



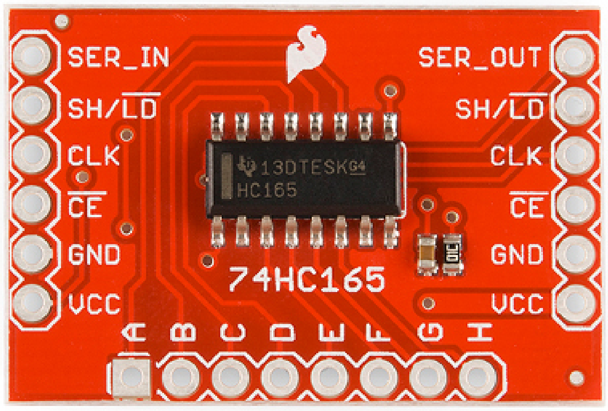
Sparkfun's large NES controller.

A real world example of using a shift register to gather inputs is the original Nintendo controller. The main microcontroller of the NES needed to get button presses from the controller, and it used a shift register to accomplish that task.

Shift Register Example

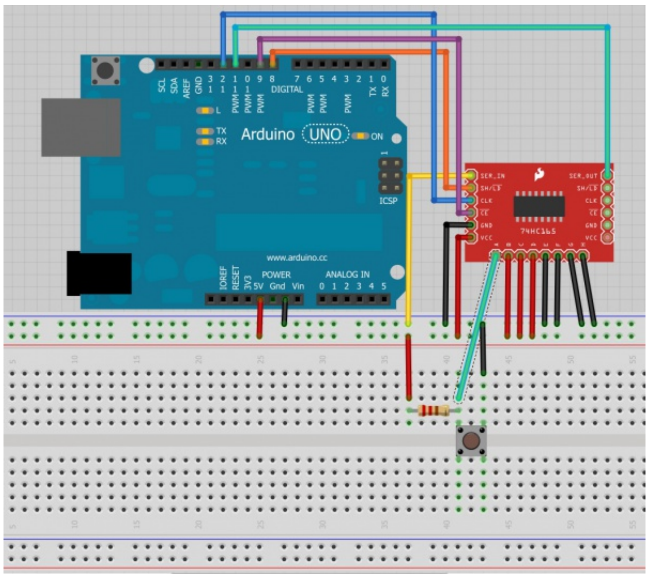
Hardware

For this example, we will use the 74hc165 breakout board and an Arduino Uno.



74hc165 breakout reference for the pinout.

An 8 bit shift register needs 4 lines of a microcontroller. One for the Clock to time the data transfer, one to the enable the clock, one for loading/latching/shifting the bits, and one for the serial data transfer.



Fritzing wiring diagram.

Connect clock (CLK) to pin 12 and clock enable ($\overline{\text{CE}}$) to pin 9. The clock sets the frequency that bits are shifted while the clock enable line allows the clock signal to propagate through to the shifting circuitry.

Connect shift/load (SH/$\overline{\text{LD}}$) to pin 8. A transition to low on the load pin tells the shift register to grab the current state of the 8 input pins(A-H). Pins A-H can be connected to some type of input like buttons, switches, or a digital transistor circuit. If you are testing them, it's recommended to just directly tie them to power or ground to make sure everything is working correctly. For the sake of this example, I'll connect one to a button with a pull up resistor and the rest to power or ground.

Connect the serial out (SER_OUT) to pin 11. This pin is where we receive the serial information from the shift register. Also, connect serial in (SER_IN) to ground. If you were chaining multiple shift registers together, serial in would be attached to the serial out of the last register. The first register in line would still have its serial in pin connected to ground while the last in the chain would have its serial out connected back to the microprocessor instead of another shift register.

Don't forget to connect power(2-6V) and ground as well. With everything hooked up, let's take a look at the firmware.

Firmware

Here's a brief rundown of what the code does. It first initializes all the pins we connected to outputs with the exception of the pin we receive serial information on. We set the clock and shift pin to initial states (HIGH) as described by the datasheet. In order to read the state of the pins A-H, we need to tell the shift register to capture the state of the pins. We do this by pulling the load pin LOW briefly (5 microseconds). Once the pins are loaded, we make sure the rest of the pins are in the starting state as described by the datasheet and use the Arduino `shiftIn` function to pull all 8 A-H pin values into a byte called incoming. The values are printed out cleanly on the Serial Terminal. It then waits and repeats. If you're connecting pins like we did above, it should be easy to test if your hardware is working correctly.

Here's the code:

```
// HARDWARE CONNECTIONS
// Connect the following pins between your Arduino and the 74HC165 Breakout Board
// Connect pins A-H to 5V or GND or switches or whatever
```

```
const int data_pin = 11; // Connect Pin 11 to SER_OUT (serial data out)
const int shld_pin = 8; // Connect Pin 8 to SH/!LD (shift or active low load)
const int clk_pin = 12; // Connect Pin 12 to CLK (the clock that times the shifting)
const int ce_pin = 9; // Connect Pin 9 to !CE (clock enable, active low)

byte incoming; // Variable to store the 8 values loaded from the shift register

// The part that runs once
void setup()
{
  // Initialize serial to gain the power to obtain relevant information, 9600 baud
  Serial.begin(9600);

  // Initialize each digital pin to either output or input
  // We are commanding the shift register with each pin with the exception of the serial
  // data we get back on the data_pin line.
  pinMode(shld_pin, OUTPUT);
  pinMode(ce_pin, OUTPUT);
  pinMode(clk_pin, OUTPUT);
  pinMode(data_pin, INPUT);

  // Required initial states of these two pins according to the datasheet timing diagram
  digitalWrite(clk_pin, HIGH);
  digitalWrite(shld_pin, HIGH);

}

// The part that runs to infinity and beyond
void loop() {

  incoming = read_shift_regs(); // Read the shift register, it likes that

  // Print out the values being read from the shift register
  Serial.println("\nThe incoming values of the shift register are: ");
  Serial.print("ABCDEFGH : ");
  print_byte(incoming); // Print every 1 and 0 that correlates with A through H
  //Serial.println(incoming,BIN); // This way works too but leaves out the leading zeros

  delay(2000); // Wait for some arbitrary amount of time

}

// This code is intended to trigger the shift register to grab values from it's A-H inpu
ts
byte read_shift_regs()
{
  byte the_shifted = 0;  // An 8 bit number to carry each bit value of A-H

  // Trigger loading the state of the A-H data lines into the shift register
  digitalWrite(shld_pin, LOW);
  delayMicroseconds(5); // Requires a delay here according to the datasheet timing diagr
am
  digitalWrite(shld_pin, HIGH);
  delayMicroseconds(5);
  // Required initial states of these two pins according to the datasheet timing diagram
  pinMode(clk_pin, OUTPUT);
  pinMode(data_pin, INPUT);
  digitalWrite(clk_pin, HIGH);
  digitalWrite(ce_pin, LOW); // Enable the clock

  // Get the A-H values
  the_shifted = shiftIn(data_pin, clk_pin, MSBFIRST);
  digitalWrite(ce_pin, HIGH); // Disable the clock

  return the_shifted;

}

// A function that prints all the 1's and 0's of a byte, so 8 bits +or- 2
void print_byte(byte val)
{
    byte i;
    for(byte i=0; i<=7; i++)
    {
      Serial.print(val >> i & 1, BIN); // Magic bit shift, if you care look up the <<, >
>, and & operators
    }
    Serial.print("\n"); // Go to the next line, do not collect $200
}
```

Here's example output:

```
The incoming values of the shift register are:
ABCDEFGH : 11110000
```

Now, try connecting each input to buttons or adding another shift register into the mix. If you chain more together, you'll have to modify the code slightly by doing the loading once, then doing a `shiftIn` for each shift register you have before loading again.

Resources and Going Further

Well, now you know what piece of hardware you need to use when you're running short on I/O lines. Hopefully this tutorial has helped you connect more inputs to your project. If you want to add more outputs and need extra help, Bildr has a great tutorial that can help you through the process.

If you're curious as to how these devices work, I recommend studying some digital logic, and eventually you will study a device called the flip-flop which is key to how these things work. Also consider looking at a timing diagram in one of the datasheets to get a sense of what happens during the shift. Enjoy exploring!

You can also explore these other tutorials from SparkFun Electronics:

- Want more information on shift registers? Check out this awesome Bildr tutorial.
- Resistors
- Capacitors
- ICs

- Reading Schematics