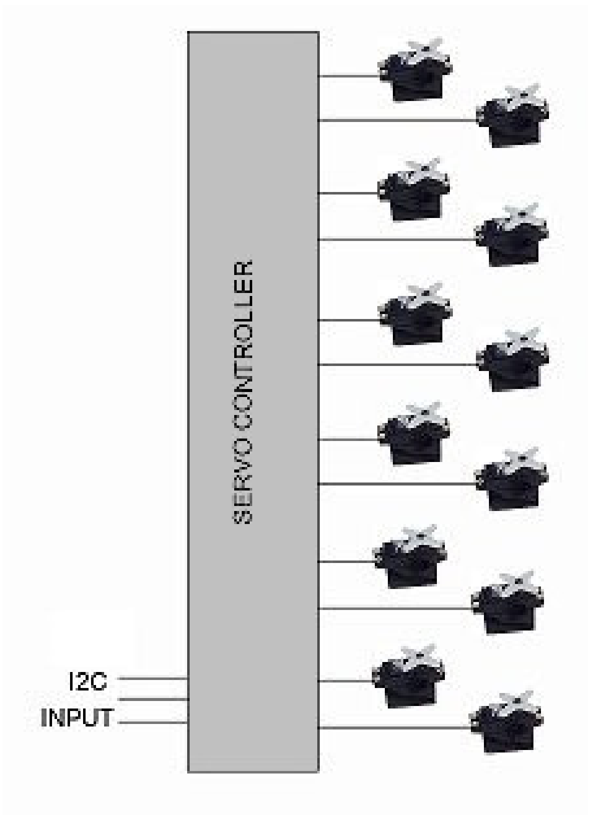
12 Channel I2C Servo Controller

Data Sheet

12 Channel Servo Controller via I2C

Introduction



Many microcontroller projects involve the use of external servos for mechanical control, robotics and in continuous rotation mode for driving wheels. Servos typically require the use of one of the use pulse width modulation (PWM) channels in the microcontroller. But microcontrollers usually have only a handful of PWM outputs, and sometimes this is not enough for the project in hand.

We have programmed a microcontroller to act as a slave servo controller which can individually generate 12 pwm output channels suitable for driving servos (or speed controllers which use servo signals). Data is easily sent to this slave device via the I2C protocol.

Each servo can be individually controlled.

No additional components are required as it has its own built in oscillator clock circuit.

Modes

There are two modes of operation, **standard** and **extended**
Standard mode sends pwm signals that vary between 1ms and 2ms. This is the norm for radio control models and allows the servo to rotate through approx 90 degrees

Extended mode sends pwm signals that vary between 0.6ms and 2.4ms. This allows most servos to rotate through almost 180 degrees and is more useful in robotics projects.

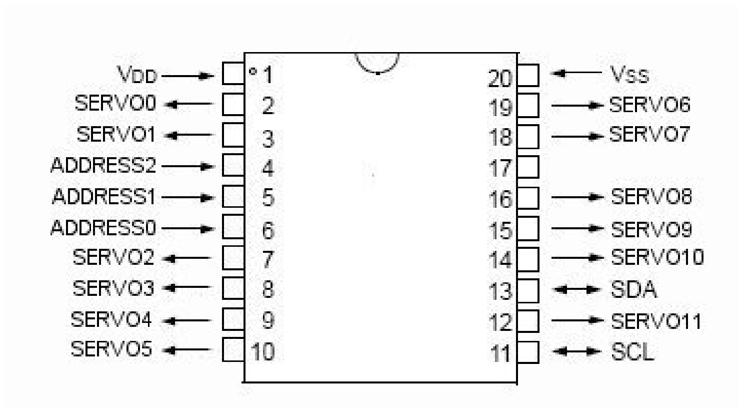
The default mode is extended

To configure the mode send a value to register address 20

To set mode standard, send value 0
To set mode extended send value 1

See the example Arduino sketch for an example of altering the mode.

Pinout



Specifications

- 12 Servo output channels (SERVO0 through SERVO11)
- I2C interface
- Configurable Slave Address allows up to 8 units to be added to your project
- 64MHz Clock Speed for accurate pwm generation
- 5V or 3.3V operation (If operating at 3.3V, the servos will still require a 5V supply)
- Configurable **standard** or **extended** modes

Hardware Configuration

Slave Address

Pins 4, 5 and 6 are used to configure the slave address of the device. The device has a default slave address of 40, but by tying these pins high or low, the address can be altered. For the default address tie all the pins low.

Slave Address	Address2 (pin4)	Address1 (pin5)	Address0 (pin 6)
40	0	0	0
41	0	0	1
42	0	1	0
43	0	1	1
44	1	0	0
45	1	0	1
46	1	1	0
47	1	1	1

SERVO Signal Output

Servo pulses are generated at 20ms intervals. Each servo pulse duration will be between 1ms and 2ms (standard) or 0.6ms and 2.4ms (extended).

If the data value for the servo channel is 0, it will have a pulse duration of 1ms (0.6ms)
If the data value for the servo channel is 255, it will have a pulse duration of 2ms (2.4ms)

The servo center position is achieved with a value of 127 which gives a pulse duration of 1.5ms

Servo outputs for each channel will default to value 127 until set via I2C

The output from the microcontroller is sufficient to connect directly to the servo signal wire (normally coloured orange). The Red Servo wire(s) should be connected to a 5V supply capable of supplying enough current for all the servos. The black wire(s) are connected to earth

I2C

To set the servo values we need to first send the starting servo register value. **This value must always be sent**. Followed by one or more bytes to set the subsequent servos.

E.g. to set all 12 servos

```
Send 0
Send SERVO0 value
Send SERVO1 value
Send SERVO2 value
Send SERVO3 value
Send SERVO4 value
Send SERVO5 value
Send SERVO6 value
Send SERVO7 value
Send SERVO8 value
Send SERVO9 value
Send SERVO10 value
Send SERVO11 value
```

To set just the first servo

```
Send 0
Send SERVO0 value
```

To set servos 5 and 6

```
Send 5
Send SERVO5 value
Send SERVO6 value
```

The remaining SERVO outputs will default to their previous value (or 127 - center position if not previously set).
I2C will work at the standard data rate of 100kHz rate or less. It may work at the higher data rate of 400kHz but this has not been tested.

Example Arduino Sketch

The following Arduino sketch sweeps each servo from center position to full right, then to full left and back to center approx once per minute. It is an ideal test to make sure everything is connected properly.

```
/*
** Wire Master of Hobbytronics 12 channel SERVO Controller
** Sweeps all 12 servos from one end to the other continuously
** Created 04 May 2011
** Version 2 06-Jun-2011
**
** This example code is in the public domain.
** www.hobbytronics.co.uk
*/

#include <Wire.h>

const int  servoslave_address=40;   // I2C Address of ADC Chip
void setup()
{
  Wire.begin();              // join i2c bus (address optional for master)

  // Optionally set mode to standard
  // - comment out this section is extended mode required
  Wire.beginTransmission(servoslave_address); // transmit to device
  Wire.send(20);             // servo register address 20
  Wire.send(0);              // send value 0 for standard mode
  Wire.endTransmission();    // stop transmitting
  delay(1);                  // waits
}


void loop()
{
  unsigned char i,j;

  for(i = 127; i < 255; i++)      // goes from center to full right
  {
    Wire.beginTransmission(servoslave_address); // transmit to device
    Wire.send(0);               // servo register to start from
    for(j=0;j<12;j++)
    {
       Wire.send(i);           // send 12 bytes of data
    }
    Wire.endTransmission();    // stop transmitting
    delay(1);                  // waits
  }

  for(i = 255; i > 0; i--)     // goes from full right to full left
  {
    Wire.beginTransmission(servoslave_address); // transmit to device
    Wire.send(0);               // servo register to start from
    for(j=0;j<12;j++)
    {
       Wire.send(i);           // send 12 bytes of data
    }
    Wire.endTransmission();    // stop transmitting
    delay(1);                  // waits
  }
  for(i = 0; i < 127; i++)      // goes from full left back to center
  {
    Wire.beginTransmission(servoslave_address); // transmit to device
    Wire.send(0);                // servo register to start from
    for(j=0;j<12;j++)
    {
       Wire.send(i);            // send 12 bytes of data
    }
    Wire.endTransmission();     // stop transmitting
    delay(1);                   // waits
  }
  delay(800);                   // waits for 0.8 seconds
}
```