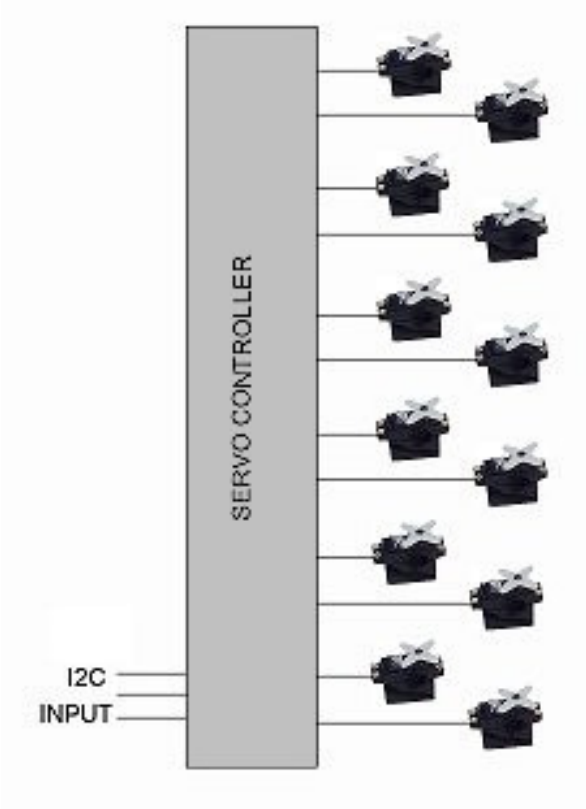
12 Channel I2C Servo Controller
Pro* Version

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 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.

Specifications

- 12 Servo output channels (SERVO0 through SERVO11)
- I2C interface
- Configurable Slave Address allows up to 4 units to be added to your project
- Accurate Servo Pulse Width generation
- 5V or 3.3V operation.
- Configurable **standard** or **extended** modes
- Programmable Servo Speed
- PWM frequency rates from 50Hz to 400Hz
- Servo Position Query
- Output can be turned ON or OFF

Additional Features of the Pro* version over Standard Version

The main features of the Pro* version over the Standard version. These features will be discussed in greater detail later in the document.

Adjustable Pulse Frequency
Over and above the standard servo pulse frequency of 50Hz, the frequency can be increased to 66Hz, 80Hz, 100Hz, 133Hz, 200Hz or 400Hz. This is a very useful feature when controlling electronic speed controllers (ESC's) as it enables a faster output response to the motors. This is needed in projects such as Quadrocopters.

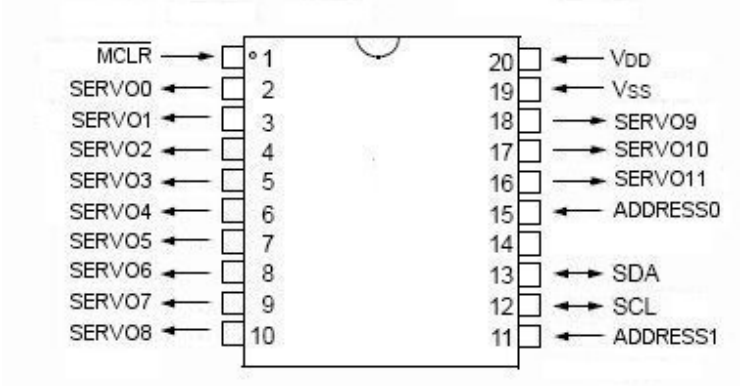
Servo Speed
This command limits the *speed* at which a servo channel's output value changes, i.e.the rate that the pulse changes from one extreme to the other (servo through 180 degrees). This can be configured in 1/10 second increments from 0 seconds up to 25 seconds. This is individually configurable for each servo. The controller will output all the intermediate pulse values.
This is useful for robotics as it enables the Servo Controller to move a servo to an end position at a fixed rate without the controlling microcontroller having to do anything.

Servo Position Query
You can query the position value of each servo to see its current (electrical) position. This position value represents the pulse width that the controller is transmitting on the channel. Very useful for robots, especially when Servo Speed is being used.

Output ON/OFF
The output of all the servo channels can be turned ON and OFF with a single command. The default startup value is OFF. This allows for configuration and default values to be sent before outputs are turned ON.

Pinout

The pinout is different to the Standard Servo Controller. They use different chips and are not compatible.



Hardware Configuration

MCLR
The MCLR pin should be connected to VDD by a suitable pullup resistor (10k).
The MCLR pin can be pulled low to reset the device

Slave Address

Pins 15 and 11 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	Address1 (pin11)	Address0 (pin 15)
40	0	0
41	0	1
42	1	0
43	1	1

SERVO Values

Servo pulses are generated at 20ms intervals (default 50Hz). Each servo pulse duration is 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 centre position is achieved with a value of 127 which gives a pulse duration of 1.5ms

The servo value can thus be controlled by the value of a single unsigned char (byte) value.

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 power for all the servos. The black/brown servo wire(s) are connected to earth.

I2C

I2C will function 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.

Setting the Servo value

To set the value for each servo position, first send the starting servo register value. This is a value between 0 and 11. **This value must always be sent**. Followed by one or more bytes to set the subsequent servos values

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.

Turning the Output ON/OFF (Register Address 60)

To turn the output to ALL servo's ON or OFF, send a 1 (On) or 0 (Off) to register address 60
The default value is 0 (OFF)
See example code later in document

Servo Mode (Register Address 61)

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 value is 1 (Extended)

To configure the mode send a value to register address 61

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.

PWM Pulse Frequency (Register Address 62)

There are seven PWM frequency settings. The frequency is selected by sending a value between 0 and 6 to register address 62 as in the table below.
The default value is 0 (50Hz)

Value	Frequency	Available Servo Outputs	
0	50Hz	12	(0 – 11)
1	66Hz	12	(0 – 11)
2	80Hz	12	(0 – 11)
3	100Hz	12	(0 – 11)
4	133Hz	12	(0 – 11)
5	200Hz	8	(0 – 7)
6	400Hz	4	(0 – 3)

With frequencies between 50Hz and 133Hz, all servo outputs are available. At 200 Hz, only 8 servo outputs are available, and at 400Hz only 4 servo outputs are available.

Servo Speed (Register Address 63 – 74)

This command limits the *speed* at which each servo channel's output value changes. I.e.the rate that the pulse changes from one extreme to the other (servo through 180 degrees). This can be configured in 1/10 second increments from 0 seconds up to 25 seconds. This is individually configurable for each servo.

Register Address	Servo
63	Servo 0
64	Servo 1
65	Servo 2
66	Servo 3
67	Servo 4
68	Servo 5
69	Servo 6
70	Servo 7
71	Servo 8
72	Servo 9
73	Servo 10
74	Servo 11

Servo Position Query (Register Address 0 – 11)

As well as setting each servo's pwm value, you can query the current pwm value for each servo. This is most useful when using the Servo Speed settings.
Start by setting the register address for the starting servo to be read, just as you would to write the servo value.
Then request one or more values for the servo(s) you want to read the position of.

See the examples below to see how this works in more detail

Example Arduino Sketches

SERVO 1 - SWEEP
The following Arduino sketch sweeps each servo from left position to full right, then back to full left and repeats. It is an ideal test to make sure everything is connected properly.

```
/*
** SERVO 1 - SWEEP
**
** Wire Master of Hobbytronics 12 channel SERVO Controller
** Sweeps all 12 servos from one end to the other continuously
** Created 04 Nov 2011
**
** This example code is in the public domain.
** www.hobbytronics.co.uk
*/

#include <Wire.h>

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

  Wire.beginTransmission(servoslave_address); // transmit to device
  Wire.send(60);             // servo register address 60
  Wire.send(1);              // Turn Servo Outputs ON
  Wire.endTransmission();    // stop transmitting
}


void loop()
{
    unsigned char j;

    Wire.beginTransmission(servoslave_address); // transmit to device
    Wire.send(0);              // servo register to start from
    for(j=0;j<12;j++)
    {
       Wire.send(0);           // send each servo left
    }
    Wire.endTransmission();    // stop transmitting
    delay(2000);               // wait 2 seconds to allow servos to reach end


    Wire.beginTransmission(servoslave_address); // transmit to device
    Wire.send(0);              // servo register to start from
    for(j=0;j<12;j++)
    {
       Wire.send(255);         // send each servo right
    }
    Wire.endTransmission();    // stop transmitting
    delay(2000);               // wait 2 seconds to allow servos to reach end

}
```

SERVO 2 - SWEEP 2

We are now going to do the same Servo Sweep, but are going to tidy things up a bit. We will initialize the start values and create a few functions to make the coding easier.

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

#include <Wire.h>

const int  servoslave_address=40;   // I2C Address of Servo Chip

// Write a config value to address register on device
void servoConfig(int device, byte address, byte val) {
  Wire.beginTransmission(device); // start transmission to device
  Wire.send(address);             // send register address
  Wire.send(val);                 // send value to write
  Wire.endTransmission();         // end transmission
}

// Write Startup values for each Servo
void servoStartup(int device) {
  unsigned char i;
  Wire.beginTransmission(device); // start transmission to device
  Wire.send(0);                   // send register address
  for(i=0;i<12;i++) Wire.send(0); // send 0 as startup value for each servo
                                  // We could easily send different values for
                                  // each servo
  Wire.endTransmission();         // end transmission
}

// Setup our Servo Configuration
void setup()
{
  Wire.begin();              // join i2c bus (address optional for master)

  // Set Servo Config and Startup values
  servoConfig(servoslave_address, 61, 1);    // Extended Mode
  servoConfig(servoslave_address, 62, 0);    // Servo Update Rate (0-6)
  servoStartup(servoslave_address);          // Send Startup values for servos
  servoConfig(servoslave_address, 60, 1);    // Servo Output ON
}

void loop()
{
    unsigned char j;

    Wire.beginTransmission(servoslave_address); // transmit to device
    Wire.send(0);              // servo register to start from
    for(j=0;j<12;j++)
    {
       Wire.send(0);           // send each servo left
```

```
    }
    Wire.endTransmission();     // stop transmitting
    delay(2000);                // wait 2 seconds to allow servos to reach end


    Wire.beginTransmission(servoslave_address); // transmit to device
    Wire.send(0);               // servo register to start from
    for(j=0;j<12;j++)
    {
       Wire.send(255);          // send each servo right
    }
    Wire.endTransmission();     // stop transmitting
    delay(2000);                // wait 2 seconds to allow servos to reach end

}
```

SERVO 3 - SWEEP 3

We are now going to do the same Servo Sweep, but are going to add **Servo Speed** of 2 seconds to Servo0 output. You will notice that a servo connected to Servo 0 output will take 2 seconds to turn from one end to the other.

```
/*
** SERVO 3 - SWEEP 3
**
** Wire Master of Hobbytronics 12 channel SERVO Controller
** Sweeps all 12 servos from one end to the other continuously
** Created 04 May 2011
**
** This example code is in the public domain.
** www.hobbytronics.co.uk
*/

#include <Wire.h>

const int  servoslave_address=40;   // I2C Address of Servo Chip

// Write a config value to address register on device
void servoConfig(int device, byte address, byte val) {
  Wire.beginTransmission(device); // start transmission to device
  Wire.send(address);             // send register address
  Wire.send(val);                 // send value to write
  Wire.endTransmission();         // end transmission
}

// Write Startup values for each Servo
void servoStartup(int device) {
  unsigned char i;
  Wire.beginTransmission(device); // start transmission to device
  Wire.send(0);                   // send register address
  for(i=0;i<12;i++) Wire.send(0); // send 0 as startup value for each servo
                                  // We could easily send different values for
                                  // each servo
  Wire.endTransmission();         // end transmission
}

// Setup our Servo Configuration
void setup()
{
  Wire.begin();              // join i2c bus (address optional for master)

  // Set Servo Config and Startup values
  servoConfig(servoslave_address, 61, 1);    // Extended Mode
  servoConfig(servoslave_address, 62, 0);    // Servo Update Rate (0-6)
  servoConfig(servoslave_address, 63, 20);   // Servo Speed set to 2 seconds (Servo 0)
  servoStartup(servoslave_address);          // Send Startup values for servos
  servoConfig(servoslave_address, 60, 1);    // Servo Output ON
}

void loop()
{
    unsigned char j;

    Wire.beginTransmission(servoslave_address); // transmit to device
    Wire.send(0);              // servo register to start from
    for(j=0;j<12;j++)
    {
```

```
      Wire.send(0);            // send each servo left
    }
    Wire.endTransmission();    // stop transmitting
    delay(2000);               // wait 2 seconds to allow servos to reach end


    Wire.beginTransmission(servoslave_address); // transmit to device
    Wire.send(0);              // servo register to start from
    for(j=0;j<12;j++)
    {
      Wire.send(255);          // send each servo right
    }
    Wire.endTransmission();    // stop transmitting
    delay(2000);               // wait 2 seconds to allow servos to reach end

}
```

SERVO 4 - QUADCOPTER

We will now ramp up the frequency to 400Hz. Only 4 Servo outputs (0 through 3) are available at this frequency.
You will have to check the output on an oscilloscope to see the frequency.

```
/*
** SERVO 4 - QUADCOPTER
**
** Wire Master of Hobbytronics 12 channel SERVO Controller
** Sweeps all 12 servos from one end to the other continuously
** at 400Hz
** Created 04 Nov 2011
**
** This example code is in the public domain.
** www.hobbytronics.co.uk
*/

#include <Wire.h>

const int  servoslave_address=40;   // I2C Address of Servo Chip

// Write a config value to address register on device
void servoConfig(int device, byte address, byte val) {
  Wire.beginTransmission(device); // start transmission to device
  Wire.send(address);             // send register address
  Wire.send(val);                 // send value to write
  Wire.endTransmission();         // end transmission
}

// Write Startup values for each Servo
void servoStartup(int device) {
  unsigned char i;
  Wire.beginTransmission(device); // start transmission to device
  Wire.send(0);                   // send register address
  for(i=0;i<12;i++) Wire.send(0); // send 0 as startup value for each servo
                                  // We could easily send different values for
                                  // each servo
  Wire.endTransmission();         // end transmission
}

// Setup our Servo Configuration
void setup()
{
  Wire.begin();              // join i2c bus (address optional for master)

  // Set Servo Config and Startup values
  servoConfig(servoslave_address, 61, 1);    // Extended Mode
  servoConfig(servoslave_address, 62, 6);    // Servo Update Rate (6=400Hz)
  servoStartup(servoslave_address);          // Send Startup values for servos
  servoConfig(servoslave_address, 60, 1);    // Servo Output ON
}

void loop()
{
    unsigned char j;

    Wire.beginTransmission(servoslave_address); // transmit to device
    Wire.send(0);              // servo register to start from
    for(j=0;j<4;j++)
    {
```

```
      Wire.send(0);            // send each servo left
    }
    Wire.endTransmission();    // stop transmitting
    delay(2000);               // wait 2 seconds to allow servos to reach end


    Wire.beginTransmission(servoslave_address); // transmit to device
    Wire.send(0);              // servo register to start from
    for(j=0;j<4;j++)
    {
      Wire.send(255);          // send each servo right
    }
    Wire.endTransmission();    // stop transmitting
    delay(2000);               // wait 2 seconds to allow servos to reach end

}
```

SERVO 5 – Read Servo Position

In this last example we are going to slowly sweep servo 0 from full left to full right and back again.
We will use the ability to read the servo position and use this to control servo 1.
When servo 0 is going right and gets to halfway we will turn servo 1 fully right.
When servo 0 is going left and gets to halfway we will turn servo 1 fully left.

```
/*
** SERVO 5 - Read Servo Position
**
** Wire Master of Hobbytronics 12 channel SERVO Controller (Version 2)
** Created 04 Nov 2011
**
** This example moves the servo 0 from full left to full right and back
** again with a speed setting of 4 seconds.
** By reading the current position of servo 0 we can determine when it is
** at the halfway point and move servo 1 full right immediately.
** A similar function is applied when the servo 0 is on the way back
** when it reaches half way, servo 1 is returned back to full left.
** 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
unsigned char servo_pos;

// Write a config value to address register on device
void servoConfig(int device, byte address, byte val) {
  Wire.beginTransmission(device); // start transmission to device
  Wire.send(address);             // send register address
  Wire.send(val);                 // send value to write
  Wire.endTransmission();         // end transmission
}

// Send a position command to a Servo
void servoPos(int device, byte servo, byte pos) {
  Wire.beginTransmission(device); // start transmission to device
  Wire.send(servo);               // send servo register address
  Wire.send(pos);                 // send position value for servo
  Wire.endTransmission();         // end transmission

}

// Write Startup values for each Servo
void servoStartup(int device) {
  unsigned char i;

  // Clear any speed settings for all servos
  Wire.beginTransmission(device); // start transmission to device
  Wire.send(63);                  // send register address
  for(i=0;i<12;i++) Wire.send(0); // send 0 as speed value for each servo
  Wire.endTransmission();         // end transmission

  // Set every servo at position 0 - change this if you want start positions
  // to be different.
  Wire.beginTransmission(device); // start transmission to device
  Wire.send(0);                   // send register address
  for(i=0;i<12;i++) Wire.send(0); // send 0 as startup value for each servo
```

```
  Wire.endTransmission();         // end transmission

}

// Read current position for a Servo
unsigned char getservoPos(int device, byte servo_num) {
  unsigned char servoPosition;

  //Send the Servo number to read
  Wire.beginTransmission(device); // start transmission to device
  Wire.send(servo_num);           // send register address (servo number)
  Wire.endTransmission();         // end transmission

  //Request servo position
  Wire.requestFrom(device, 1);    // request 1 byte from slave device
  if(Wire.available())
  {
    servoPosition = Wire.receive(); // receive a byte as character
  }
  return servoPosition;

}

void setup()
{
  Wire.begin();              // join i2c bus (address optional for master)

  // Set Servo Config and Startup values
  servoConfig(servoslave_address, 61, 1);    // Extended Mode
  servoConfig(servoslave_address, 62, 0);    // Servo Update Rate (0-6)
  servoStartup(servoslave_address);          // Send Startup values for servos
                                             // and clear speed settings
  servoConfig(servoslave_address, 63, 40);   // Servo Speed set to 4 seconds (Servo 0)
  servoConfig(servoslave_address, 60, 1);    // Servo Output ON

  // Add a delay here to give servos time to move to startup position

}

void loop()
{
  unsigned char i,j;

  // Send servo 0 full right (speed is 4 seconds)
  servoPos(servoslave_address, 0, 255);

  // Read position of servo 0, when it gets to center move servo 1 full right
  while(1)
  {
    delay(2);               // waits
    servo_pos = getservoPos(servoslave_address, 0);
    if(servo_pos>=125) break;
  }

  // move servo 1 full RIGHT
  servoPos(servoslave_address, 1, 255);

  // Read position of servo 0, when it gets to full right, send it back to full left
  while(1)
  {
    delay(2);               // waits
```

```
    servo_pos = getservoPos(servoslave_address, 0);
    if(servo_pos>=255) break;
  }

  // send servo 0 back to full left
  servoPos(servoslave_address, 0, 0);

  // Read position of servo 0, when it gets to center move servo 1 full LEFT
  while(1)
  {
    delay(2);                // waits
    servo_pos = getservoPos(servoslave_address, 0);
    if(servo_pos<=125) break;
  }

  // move servo 1 full LEFT
  servoPos(servoslave_address, 1, 0);

  // wait unti servo 0 gets back to 0, then loop
  while(1)
  {
    delay(2);                // waits
    servo_pos = getservoPos(servoslave_address, 0);
    if(servo_pos<=0) break;
  }

}
```