RPI 2 – I^2C and SIGINT

I^2C interface, SIGINT

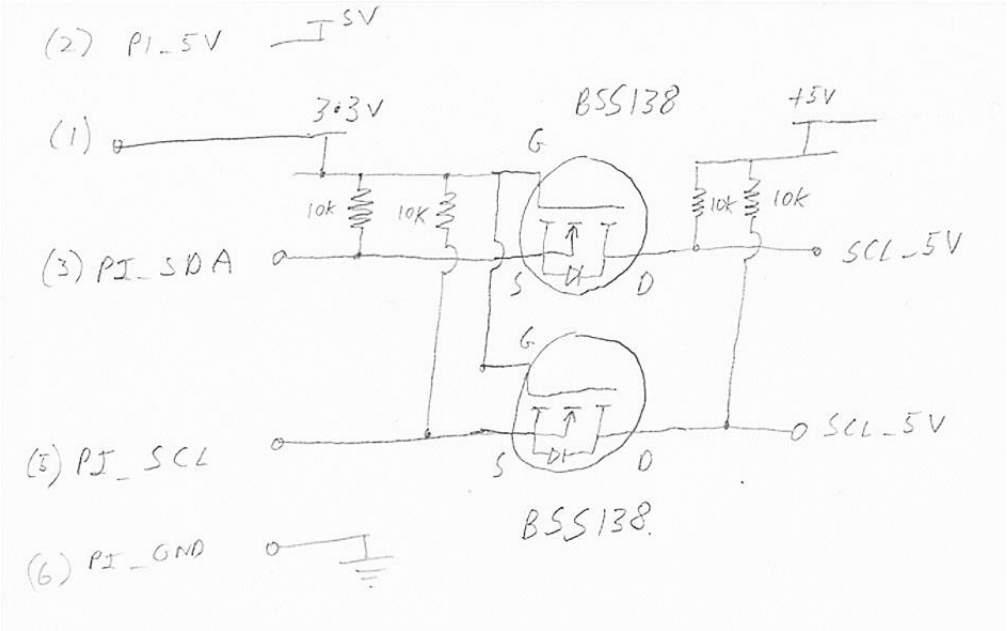
- Heather Lomond

I²C Driver

I2C is a serial, 2 wire, bi-directional communications bus
Usually Single Master, Multiple Slave configuration.
Requires pullup resistors on all lines.
Devices only ever pull the bus low and never drive it high.
Devices are addressed by a single byte (usually hard coded into the device)
Devices are register mapped

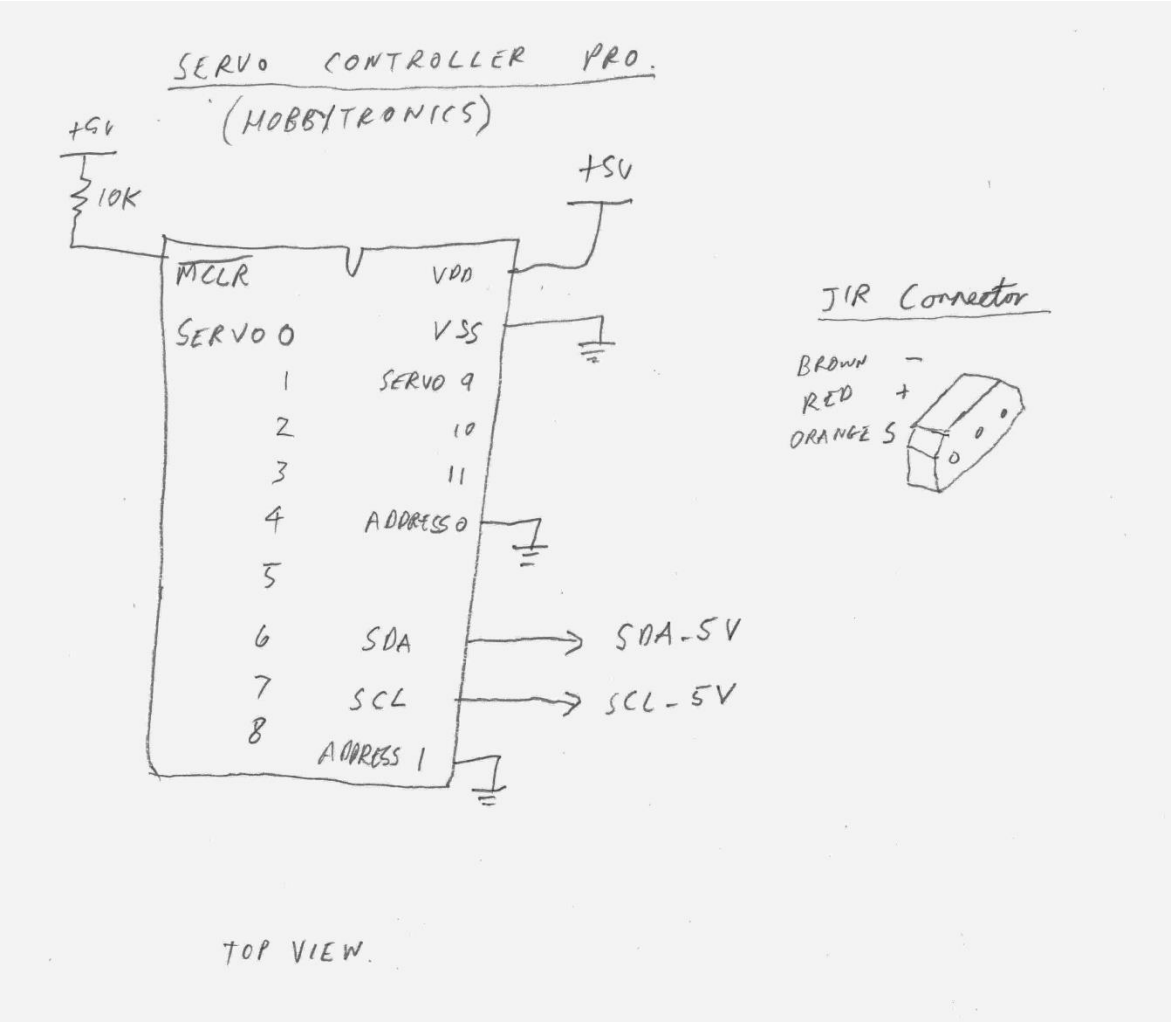
Pi versions have changed how this interface works, so here we are only considering the Pi 2 that uses the full Linux I2C driver.

Pi is 3.3V I/O, so we need level Shifters for 5V devices.



I²C – Hardware - Servo

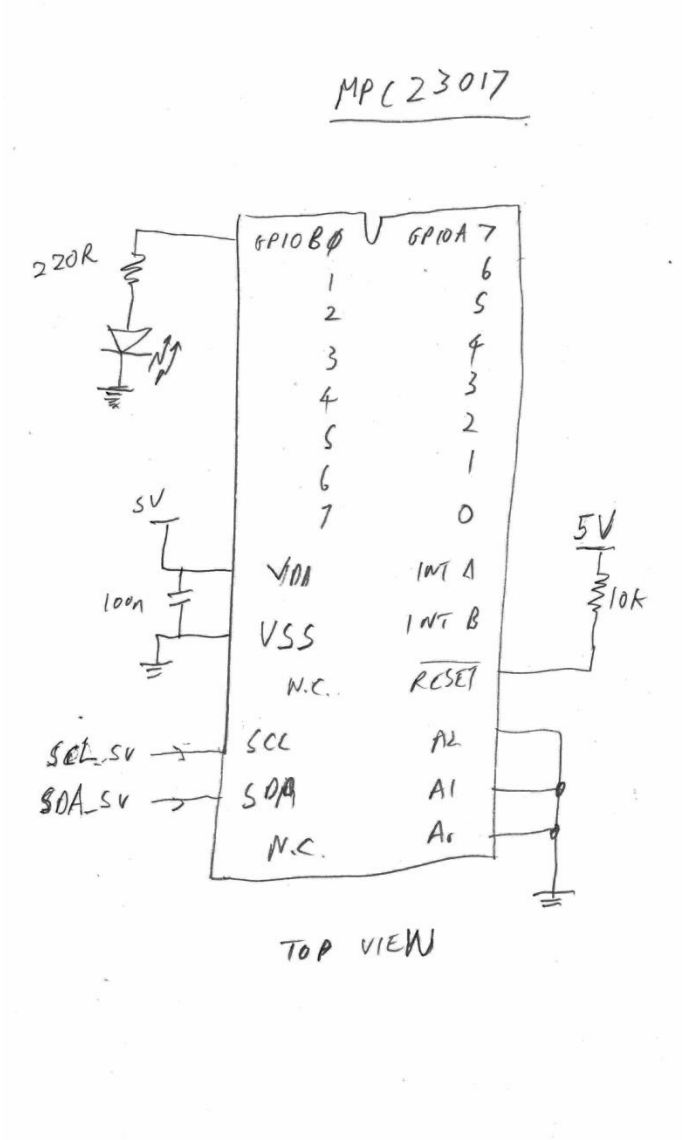
This uses the Hobbytronics Servo_Pro 12 channel servo controller.



I²C – Hardware - GPIO

Just a simple example of I2C hardware, a 16 channel I/O expander – the MCP23017.
A 5V device

Attached is a relay controlled via a BC557 PNP transistor



I²C – Enabling it on the Pi

Basic delivered Pi doesn't have I2C enabled (we want to see it appear as a device in /dev). Use

```
sudo rpi-config
```

And set option 8 (advanced), go into i2c and enable it as well as enabling the module loading, then go for "finish" to get it all set. Next you need to edit the modules file (which tells Linux what modules to load at boot time)

```
sudo vi /etc/modules
```

and add these two lines:

```
i2c-bcm2708
i2c-dev
```

Then also check there is no blacklisting in

```
/etc/modprobe.d/raspi-blacklist.conf
```

Now reboot and you should be set up.

I²C – Tools

If you go in the `/dev` directory, when it is all enabled you should see a file `i2c-1`

```
ls /dev/
```

A good tool to check your hardware is i2cdetect

```
sudo apt-get install i2c-tools
```

And run it using device 1 (this tells it to use i2c-1 /dev file). For older RPIs the I2C was on i2c-0 but they moved it in newer hardware.

```
sudo i2cdetect -y 1
```

This will look at all 128 addresses and see if it can find any hardware responding to that address. You should see your device appear under it's address

I²C – Pi Version Pt1

What we want to do is open the `cpuinfo` file on the pi. This contains various details about the CPU and versions of things.
In there is a line defining the hardware, and we are interested in the SOC (i.e all the peripherals of the Broadcom chip).
For earlier PIs the SOC was labeled “BCM2708” and on RPI 2 it is upgraded to the “BCM2709”

```
int soc_pi_revision = -1;

/*-------------------------------------------------------------*/
void soc_get_pi_revision()
/*-------------------------------------------------------------*/
{
    FILE *cpuinfo;
    char line[120];

    /* open the info file to find what we need to know */
    if ((cpuinfo = fopen ("/proc/cpuinfo", "r")) == NULL) {
        printf("Can't open /proc/cpuinfo\n");
    }
```

I²C – Pi Version Pt2

```
    /* Now read the file till we find the Hardware line */
    while (fgets (line, 120, cpuinfo) != NULL)
        if (strncmp (line, "Hardware", 8) == 0)
        break;
    if (strncmp (line, "Hardware", 8) != 0) {
        printf("Can't find the Hardware line in the file /proc/cpuinfo\n");
        exit (-1);
    }

    /* See what SOC we have, BCM2708 or BCM2709 */
    if (strstr (line, "BCM2709") != NULL) {
        soc_pi_revision=2;
        printf("Found an RPI 2\n");
    } else {
        soc_pi_revision=1;
        printf("Found an original RPI\n");
    }
    fclose (cpuinfo);
}
```

I²C – Open the File

Now, to, initialise the RPI I2C interface we need to …. Guess what …. Open a file!

```
int i2c_file;

/* open the I2C bus */
char *filename = (char*)"/dev/i2c-1";
if ((i2c_file = open(filename, O_RDWR)) < 0) {
   printf("Failed to open the i2c bus\n");
}
```

I²C – Write out a byte

Sending data simply involves knowing the bus address of the device, the register you want to write to and the value you want to send in.

```
/*-----------------------------------------------------------------*/
int i2c_send_one_byte_to_reg(uint8 i2c_addr, uint8 i2c_reg, uint8 i2c_val)
/*-----------------------------------------------------------------*/
{
   if (ioctl(i2c_file, I2C_SLAVE, i2c_addr) < 0) {
       printf("Failed to acquire bus access and/or talk to slave\n");
       return -1;
   }
   int length = 2;
   unsigned char buffer[2];
   buffer[0]=i2c_reg;
   buffer[1]=i2c_val;
   if (write(i2c_file, buffer, length) != length) {
       printf("Failed to write to the i2c bus\n");
   }
}
```

I²C – Read in a byte

Reading is pretty much the same but in reverse

```
/*-------------------------------------------------------------*/
int i2c_read_one_byte_from_reg(uint8 i2c_addr, uint8 i2c_reg, uint8 *i2c_value)
/*-------------------------------------------------------------*/
{
   if (ioctl(i2c_file, I2C_SLAVE, i2c_addr) < 0) {
        printf("Failed to acquire bus access and/or talk to slave\n");
        return -1;
   }
   int length = 1; /* first write out the register address we want to read */
   if (write(i2c_file, &i2c_reg, length) != length) {
        printf("Failed to write the reg addr to the i2c bus\n");
        return -1;
   }
   if (read(i2c_file, i2c_value, length) != length) {
         printf("Failed to read from the i2c bus\n");
         return -1;
   }
   return 0;
}
```

SIGINT – trapping Ctrl-C

So, you have your program running and you want to Ctrl-C but all the GPIOs are still setup, I2C files open etc. We need a signal handler to trap Ctrl-C and end the program cleanly!

```
#include<signal.h>
#include<stdlib.h>
#include<unistd.h>

void sig_handler(int signal_number)
{
  if (signal_number == SIGINT) {
      close(i2c_file);
      exit(-1);
  }
}
```

And then we can link our program to it with

```
  if (signal(SIGINT, sig_handler) == SIG_ERR)
      printf("\ncan't catch SIGINT\n");
```

Now, whenever a signal, like Ctrl-C come in, it won't shut us down, it will just call the sig_handler routine and we can do good stuff in there.

Questions