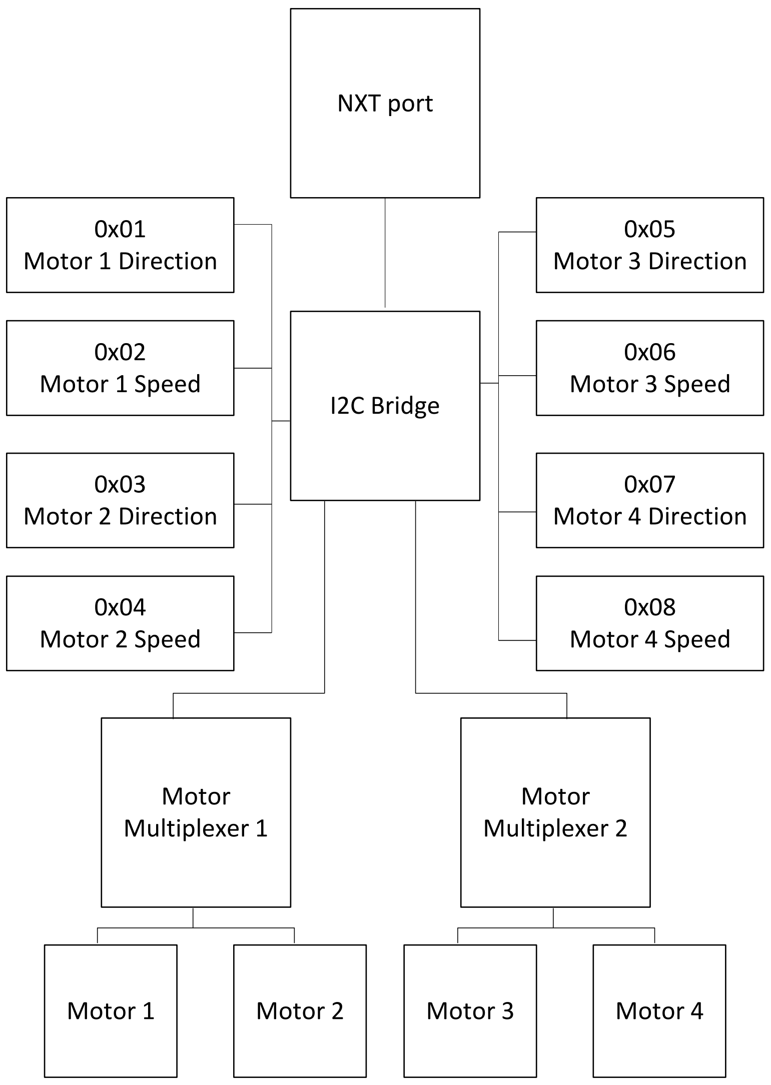
I2C Motor Multiplexer

The I2C motor multiplexer board can be used to drive up to 4 DC motors from a NXT sensor port. The motors are able to be controlled independently by using 2 registers: one for direction and another for speed. There are 2 motor controller chips on each board, while this is not important from a design point of view it is worth noting that even if one chip is broken the other will be able to function.

Fig 1. Representation of I2C Motor Multiplexer



Motor Control

The direction registers should hold values between 0 and 3 corresponding to the following actions. It must be noted that should the value 3 be put in the register then the motor multiplexer will impose 0V across the motor. This means that should the motor be turned when the motor is in the break state that the motor multiplexer will drive reverse current in order to impose this 0V. While this may not appear to be a huge issue it can lead to the boards overloading if the motor is externally driven while in the break state.

Particular care should be taken if this board is being used to power a solenoid as the back EMF is more than capable of blowing the motor controller chips. One other consequence of break is that it will drain the battery very quickly due to the imposition of 0V.

This is not to say that break cannot be used but it does mean that very careful consideration if break is really needed or if float is a sufficient to accomplish the task.

Fig 2. Direction look up table

Direction Value	Result
0	Float (use this for off)
1	Forwards
2	Backwards
3	Break (BE VERY CAREFUL WITH THIS)

The speed register can take a value from 0 to 255 and controls the speed of the motor in whatever direction it is spinning.

Fig 3. Lejos I2C methods used

Method	Description
i2cEnable	Tells the I2C port how to function. Use *I2CPort.STANDARD_MODE* for the motor multiplexer
setAddress(int)	Sets the address of the I2C to be used. In this case hex 5A or 90
sendData(int,byte)	The byte is transferred to the register int. This returns a 0 if completed non 0 if an error

Sample Lejos code

Fig 4. Lejos sample code

```
I2CPort I2Cport; //Create a I2C port
I2Cport = SensorPort.S1; //Assign port
I2Cport.i2cEnable(I2CPort.STANDARD_MODE);
//Initialize port in standard mode

I2CSensor I2Csensor = new I2CSensor(I2Cport);
//Creates an I2CSensor

byte direction = (byte)1;
byte speed = (byte)200;

I2Csensor.setAddress(0x5A);
I2Csensor.sendData(0x01,direction);
I2Csensor.sendData(0x02,speed);
```

This code will turn motor 1 forward at a reasonably fast speed The code could then be expended to perform a kicking action by having it wait a short period of time, reversing , waiting again, and then finally going into float mode.

Fig 5. Sample Lejos kicker code

```
I2CPort I2Cport; //Create a I2C port
I2Cport = SensorPort.S1; //Assign port
I2Cport.i2cEnable(I2CPort.STANDARD_MODE);
//Initialize port in standard mode

I2CSensor I2Csensor = new I2CSensor(I2Cport);
//Creates an I2CSensor

byte forward = (byte)1;
byte backward = (byte)2;
byte off = (byte)0;
byte speed = (byte)200;

I2Csensor.setAddress(0x5A);
I2Csensor.sendData(0x01,forward);
I2Csensor.sendData(0x02,speed);

Thread.sleep(20);
I2Csensor.sendData(0x01,backward);

Thread.sleep(20);
I2Csensor.sendData(0x01,off);
I2Csensor.sendData(0x02,off);
```

There was some initial trouble with the I2C code, this appears to have been caused by the batter not being powerful enough to drive the motor through the I2C board despite it being powerful enough to drive the motor on its own.

The problem was initially thought to lie with the information from mindsensors with regard to the registers being incorrect; the information for the address of the board was incorrect so this assumption was reasonable at the time however after performing multiple register sweeps. Putting 1 in every odd register and 200 in ever even the board refused to work. However when this was revered the board started to operate despite the registers being revered. The exact reason for this is not known.

Fig 6. I2C register sweep

```
I2CPort I2Cport; //Create a I2C port
I2Cport = SensorPort.S1; //Assign port
I2Cport.i2cEnable(I2CPort.STANDARD_MODE);
//Initialize port in standard mode

I2CSensor I2Csensor = new I2CSensor(I2Cport);
//Creates an I2CSensor


int counter;
byte direction = (byte)1;
byte speed = (byte)200;

I2Csensor.setAddress(0x5A);

for( counter = 0; counter <65; counter ++){
I2Csensor.sendData(0x01 +
(2*counter),direction);
I2Csensor.sendData(0x02 + (2*counter),speed);
}

 Thread.sleep(1000)

for( counter = 0; counter <65; counter ++){
I2Csensor.sendData(0x02 +
(2*counter),direction);
I2Csensor.sendData(0x01 + (2*counter),speed);
}
```

The code will do a complete register sweep putting 1 in each odd register and 200 in each even register, then wait 10 seconds and do the reverse.