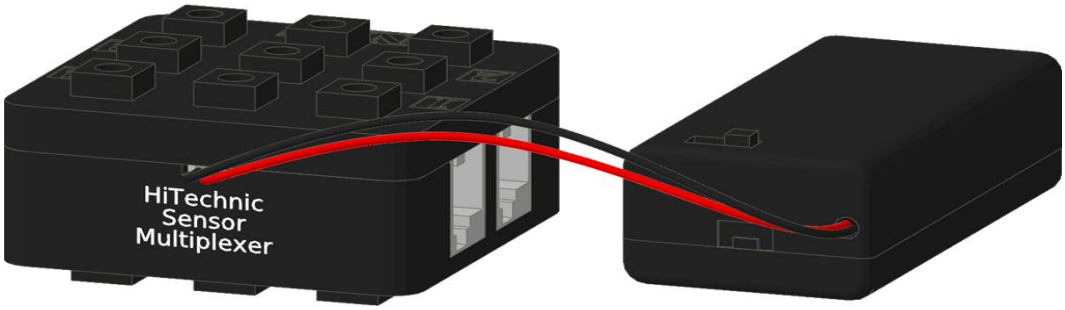
HiTechnic Sensor Multiplexer Programmer's Guide



Version 1.0.1

November 25, 2009

By Xander Soldaat

Contents

Thanks to John Hansen for reviewing the NXC code and John, Steve and Gus from HiTechnic for reviewing this document and providing me with all the necessary information.

The HiTechnic Sensor Multiplexer

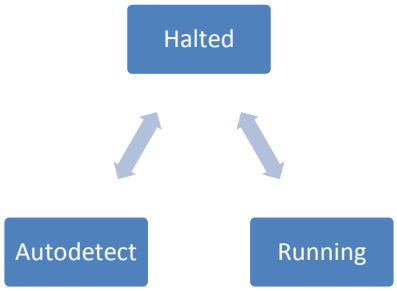
Introduction

The HiTechnic Sensor Multiplexer (SMUX) allows you to read up to 4 supported analogue or digital sensors via a single NXT sensor port.

The SMUX itself is a digital sensor that can be configured and queried through I2C. To use the SMUX in programming environments such as ROBOTC or NXC, it's necessary to have a basic understanding of its operation.

Operational states

The SMUX has 3 basic operational states; **halted**, **autodetect** and **running**.



The **halted** state has to be entered before any other command is given to the SMUX.

When the **autodetect** state is entered, the SMUX will attempt to probe the sensors currently attached to its ports. If the sensor is a digital one, the information returned to the SMUX is compared to an internal lookup table. If the sensor is found in this table, it populates the internal configuration registers (See: Appendix A) with the information required to read that sensor. If the sensor is not found or doesn't respond to I2C queries, that SMUX port is configured as analogue. The list of supported sensors can be found in Appendix E. The SMUX takes approximately 500ms to complete the scan. Care should be taken not to issue any new commands during this cycle. The **autodetect** state may only be entered when the SMUX is in a **halted** state. When the **autodetect** state exits, the SMUX will automatically revert to the **halted** state.

To start reading the sensors connected to the SMUX, it must enter the **running** state. This state may only be entered from the **halted** state.

Multiplexer Operation

How does it do what it does?

The SMUX works by querying the connected sensors on behalf of the user. The results from these queries are stored in the registers 40-4FH, 50-5FH, 60H-6FH, 70-7FH for sensors 1, 2, 3 and 4 respectively. In the case of an analogue sensor, the 10 bit values can be retrieved from registers 36-37H, 38-39H, 3A-3BH and 3C-3DH for analogue sensors 1, 2, 3 and 4 respectively. So how does the SMUX know what to ask for? Parameters such as the sensor's I2C address, the number of bytes to ask for and the register to be queried are all stored in their respective registers as specified in Appendix A.

Depending on the type of sensor and the number of bytes that are queried, the SMUX may poll the sensor at different frequencies. Below is a table of the polling frequencies.

Sensor Type	I2C Byte Count	Polling Frequency
Analogue	-	300Hz (roughly every 3ms)
I2C	>= 8	50Hz (every 20ms)
I2C	< 8	100Hz (every 10ms)

Programming examples

Since theory is nothing without practice, here are some examples of how to use this SMUX using both NXC and ROBOTC.

Halting the SMUX

ROBOTC

```
void HTSMUXhalt(tSensors smux) {
  ubyte sendMsg[4];

  sendMsg[0] = 3;      // size of I2C message
  sendMsg[1] = 0x10;   // Address of SMUX
  sendMsg[2] = 0x20;   // Command register
  sendMsg[3] = 0;      // Command to be sent
                       // (halt)

  // Send the command to the SMUX to put it
  // in halted state
  sendI2CMsg(smux, sendMsg, 0);

  // wait 50ms for the SMUX to clean up
  Wait1Msec(50);
}
```

NXC

```
void HTSMUXhalt (const byte smux) {
  byte sendMsg[3];

  sendMsg[0] = 0x10;   // Address of SMUX
  sendMsg[1] = 0x20;   // Command register
  sendMsg[2] = 0;      // Command to be sent
                       // (halt)

  // Send the command to the SMUX to put it
  // in halted state
  I2CWrite(smux, 0, sendMsg);

  // Wait 50ms after SMUX is halted
  Wait(50);
}
```

Initiating autodetect state

ROBOTC

```
void HTSMUXautodetect(tSensors smux) {
  ubyte sendMsg[4];

  sendMsg[0] = 3;      // size of I2C message
  sendMsg[1] = 0x10;   // Address of SMUX
  sendMsg[2] = 0x20;   // Command register
  sendMsg[3] = 1;      // Command to be sent
                       // (autodetect)

  // Send the command to the SMUX to
  // start probing sensors.
  sendI2CMsg(smux, sendMsg, 0);

  // Wait 500ms for autodetect to complete
  wait1Msec(500);
}
```

NXC

```
void HTSMUXautodetect (const byte smux) {
  byte sendMsg[3];

  sendMsg[0] = 0x10;   // Address of SMUX
  sendMsg[1] = 0x20;   // Command register
  sendMsg[2] = 1;      // Command to be sent
                       // (autodetect)

  // Send the command to the SMUX to put it
  // in halted state
  I2CWrite(smux, 0, sendMsg);

  // Wait 500ms for autodetect to complete
  Wait(500);
}
```

Start normal multiplexer operation

ROBOTC

```
void HTSMUXrun (tSensors smux) {
  ubyte sendMsg[4];

  sendMsg[0] = 3;      // size of I2C message
  sendMsg[1] = 0x10;   // Address of SMUX
  sendMsg[2] = 0x20;   // Command register
  sendMsg[3] = 2;      // Command to be sent
                       // (run)

  // Send the command to the SMUX to put it
  // in running state
  sendI2CMsg(smux, sendMsg, 0);
}
```

NXC

```
void HTSMUXrun (byte smux) {
  byte sendMsg[3];

  sendMsg[0] = 0x10;   // Address of SMUX
  sendMsg[1] = 0x20;   // Command register
  sendMsg[2] = 2;      // Command to be sent
                       // (run)

  // Send the command to the SMUX to put it
  // in running state
  I2CWrite(smux, 0, sendMsg);
}
```

We can optimize the three functions into one with the following code

Sending a command to the SMUX

ROBOTC

```
#define HTSMUX_CMD_HALT         0x00
#define HTSMUX_CMD_AUTODETECT   0x01
#define HTSMUX_CMD_RUN          0x02

void HTSMUXsendCmd(tSensors smux, byte cmd) {
  ubyte sendMsg[4];

  sendMsg[0] = 3;       // size of I2C message
  sendMsg[1] = 0x10;    // Address of SMUX
  sendMsg[2] = 0x20;    // Command register
  sendMsg[3] = cmd;     // Command to be sent

  // Send the command to the SMUX
  sendI2CMsg(smux, sendMsg, 0);

  // if the HTSMUX_CMD_AUTODETECT command has
  // been given, wait 500 ms
  if (cmd == HTSMUX_CMD_AUTODETECT) {
    wait1Msec(500);

  // Wait 50ms after SMUX is halted
  } else if (cmd == HTSMUX_CMD_HALT) {
    wait1Msec(50);
  }
}
```

NXC

```
#define HTSMUX_CMD_HALT         0x00
#define HTSMUX_CMD_AUTODETECT   0x01
#define HTSMUX_CMD_RUN          0x02

void HTSMUXsendCmd(byte smux, byte cmd)  {
  byte sendMsg[3];

  sendMsg[0] = 0x10;   // Address of SMUX
  sendMsg[1] = 0x20;   // Command register
  sendMsg[2] = cmd;    // Command to be sent

  // Send the command to the SMUX
  I2CWrite(smux, 0, sendMsg);

  // if the HTSMUX_CMD_AUTODETECT command has
  // been given, wait 500 ms
  if (cmd == HTSMUX_CMD_AUTODETECT) {
    Wait(500);

  // Wait 50ms after SMUX is halted
  } else if (cmd == HTSMUX_CMD_HALT) {
    Wait(50);
  }
}
```

Now that we know how to get the SMUX into the three states, **halted**, **autodetect** and **running**, let's see about making it work for us.

If you look at Appendix A, you'll see that registers 40-4FH, 50-5FH, 60-6FH and 70-7FH, the I2C buffers, are all 16 bytes wide. This is no coincidence, it is the maximum size read reply the NXT can handle. Each time the SMUX polls a sensor, the data returned from it is placed in its respective I2C buffer.

Reading polled data from an I2C sensor

ROBOTC

```
typedef struct  {
  ubyte arr[16];
} byte_array;

void HTSMUXreadI2C (tSensors smux, byte chan,
                    byte offset, byte length,
                    byte_array &array) {

  ubyte sendMsg[3];

  sendMsg[0] = 2;        // size of I2C message
  sendMsg[1] = 0x10;     // Address of SMUX
                         // Buffer register
  sendMsg[2] = 0x40 + (chan * 16) + offset;

  // Query the SMUX and read the response
  sendI2CMsg(smux, sendMsg, length);
  wait1Msec(10);
  readI2CReply(smux, array.arr[0], length);
}
```

NXC

```
bool HTSMUXreadI2C (byte smux, byte chan,
                    byte offset, byte length,
                    byte &data[]) {

  byte sendMsg[2];

  sendMsg[0] = 0x10;     // Address of SMUX
                         // Buffer register
  sendMsg[1] = 0x40 + (chan * 16) + offset;

  // Query the SMUX and read the response
  return I2CBytes(smux, sendMsg, length, data);
}
```

If the sensor is an analogue one, you have to query different buffers. Judging by Appendix A, the registers we're going to want to read are 36-37H, 38-39H, 3A-3BH and 3C-3DH, 2 bytes for each channel. The SMUX's analogue data is 10 bits wide, the same as the native NXT bit width. The first byte contains the upper 8 bits and the second byte contains the lower 2 bits.

Reading polled data from an analogue sensor

ROBOTC

```
int HTSMUXreadAD (tSensors smux, byte chan) {

  ubyte sendMsg[3];
  ubyte readMsg[2];

  sendMsg[0] = 2;        // size of I2C message
  sendMsg[1] = 0x10;     // Address of SMUX
                         // Buffer register
  sendMsg[2] = 0x36 + (chan * 2);

  // Query the SMUX and read the response
  sendI2CMsg(smux, sendMsg, 2);
  wait1Msec(10);
  readI2CReply(smux, readMsg, 2);

  // ensure no weirdness from signed/unsigned
  // conversions
  return (readMsg[0] & 0x00FF) * 4 +
         (readMsg[1] & 0x00FF);
}
```

NXC

```
int HTSMUXreadAD (byte smux, byte chan) {

  byte sendMsg[2];
  byte readMsg[2];
  const byte count = 2;

  sendMsg[0] = 0x10;     // Address of SMUX
                         // Buffer register
  sendMsg[1] = 0x36 + (chan * 2);

  // Query the SMUX and read the response
  if (I2CBytes(smux, sendMsg, count, readMsg))
    // ensure no weirdness from signed/unsigned
    // conversions
    return (readMsg[0] & 0x00FF) * 4 +
           (readMsg[1] & 0x00FF);
  else
    return 0;
}
```

Putting it all together

Now that we know how to work the basics of the SMUX, it's time to put it all together. I've used a HiTechnic Colour Sensor V2 and a Gyro in my tests; you can use whatever you'd like, as long as it's supported by the SMUX. You may need to modify the program to properly reassemble the data from the sensors in question.

Reading a HiTechnic colour sensor (I2C) attached to port 1 of a SMUX

ROBOTC

```
#define HTSMUX_CMD_HALT         0x00
#define HTSMUX_CMD_AUTODETECT   0x01
#define HTSMUX_CMD_RUN          0x02

typedef struct  {
  ubyte arr[16];
} byte_array;

void HTSMUXsendCmd(tSensors smux, byte cmd) {
  ubyte sendMsg[4];

  sendMsg[0] = 3;       // size of I2C message
  sendMsg[1] = 0x10;    // Address of SMUX
  sendMsg[2] = 0x20;    // Command register
  sendMsg[3] = cmd;     // Command to be sent

  // Send the command to the SMUX
  sendI2CMsg(smux, sendMsg, 0);

  // if the HTSMUX_CMD_AUTODETECT command has
  // been given, wait 500 ms
  if (cmd == HTSMUX_CMD_AUTODETECT) {
    wait1Msec(500);

  // Wait 50ms after SMUX is halted
  } else if (cmd == HTSMUX_CMD_HALT) {
    wait1Msec(50);
  }
}

void HTSMUXreadI2C (tSensors smux, byte chan,
                    byte offset, byte length,
                    byte_array &array) {

  ubyte sendMsg[3];

  sendMsg[0] = 2;       // size of I2C message
  sendMsg[1] = 0x10;    // Address of SMUX
                        // Buffer register
  sendMsg[2] = 0x40 + (chan * 16) + offset;

  // Query the SMUX and read the response
  sendI2CMsg(smux, sendMsg, length);
  wait1Msec(10);
  readI2CReply(smux, array.arr[0], length);
}

task main () {
  byte_array data;
  SetSensorType(S1, sensorLowSpeed);
  wait1Msec(100);

  // first send the halt command to the SMUX
  HTSMUXsendCmd(S1, HTSMUX_CMD_HALT);
  // Initiate scan
  HTSMUXsendCmd(S1, HTSMUX_CMD_AUTODETECT);
  // Start normal operation
  HTSMUXsendCmd(S1, HTSMUX_CMD_RUN);

  while (true) {
     // Read a single byte from the I2C
     // buffer for channel 0 (SMUX port 1)
     HTSMUXreadI2C(S1, 0, 0, 1, data);
     nxtDisplayTextLine(2, "%d", data.arr[0]);
     wait1Msec(100);
  }
}
```

NXC

```
#define HTSMUX_CMD_HALT         0x00
#define HTSMUX_CMD_AUTODETECT   0x01
#define HTSMUX_CMD_RUN          0x02

void HTSMUXsendCmd(byte smux, byte cmd)  {
  byte sendMsg[3];

  sendMsg[0] = 0x10;   // Address of SMUX
  sendMsg[1] = 0x20;   // Command register
  sendMsg[2] = cmd;    // Command to be sent

  // Send the command to the SMUX
  I2CWrite(smux, 0, sendMsg);

  // if the HTSMUX_CMD_AUTODETECT command has
  // been given, wait 500 ms
  if (cmd == HTSMUX_CMD_AUTODETECT) {
    Wait(500);

  // Wait 50ms after SMUX is halted
  } else if (cmd == HTSMUX_CMD_HALT) {
    Wait(50);
  }
}

bool HTSMUXreadI2C (byte smux, byte chan,
                    byte offset, byte length,
                    byte &data[]) {

  byte sendMsg[2];

  sendMsg[0] = 0x10;    // Address of SMUX
                        // Buffer register
  sendMsg[1] = 0x40 + (chan * 16) + offset;

  // Query the SMUX and read the response
  return I2CBytes(smux, sendMsg, length, data);
}

task main () {
  byte data[16];
  SetSensorLowspeed(S1);
  Wait(100);

  // first send the halt command to the SMUX
  HTSMUXsendCmd(IN_1, HTSMUX_CMD_HALT);
  // Initiate scan
  HTSMUXsendCmd(IN_1, HTSMUX_CMD_AUTODETECT);
  // Start normal operation
  HTSMUXsendCmd(IN_1, HTSMUX_CMD_RUN);

  while (true) {
     // Read a single byte from the I2C
     // buffer for channel 0 (SMUX port 1)
     HTSMUXreadI2C(IN_1, 0, 0, 1, data);
     NumOut(0, LCD_LINE2, data[0], true);
     Wait(100);
  }
}
```

Reading a HiTechnic Gyro sensor (analogue) attached to port 1 of a SMUX

ROBOTC

```
#define HTSMUX_CMD_HALT         0x00
#define HTSMUX_CMD_AUTODETECT   0x01
#define HTSMUX_CMD_RUN          0x02

void HTSMUXsendCmd(tSensors smux, byte cmd) {
  ubyte sendMsg[4];

  sendMsg[0] = 3;       // size of I2C message
  sendMsg[1] = 0x10;    // Address of SMUX
  sendMsg[2] = 0x20;    // Command register
  sendMsg[3] = cmd;     // Command to be sent

  // Send the command to the SMUX
  sendI2CMsg(smux, sendMsg, 0);

  // if the HTSMUX_CMD_AUTODETECT command has
  // been given, wait 500 ms
  if (cmd == HTSMUX_CMD_AUTODETECT) {
    wait1Msec(500);

  // Wait 50ms after SMUX is halted
  } else if (cmd == HTSMUX_CMD_HALT) {
    wait1Msec(50);
  }
}

int HTSMUXreadAD (tSensors smux, byte chan) {

  ubyte sendMsg[3];
  ubyte readMsg[2];

  sendMsg[0] = 2;       // size of I2C message
  sendMsg[1] = 0x10;    // Address of SMUX
                        // Buffer register
  sendMsg[2] = 0x36 + (chan * 2);

  // Query the SMUX and read the response
  sendI2CMsg(smux, sendMsg, 2);
  wait1Msec(10);
  readI2CReply(smux, readMsg, 2);

  // ensure no weirdness from signed/unsigned
  // conversions
  return (readMsg[0] & 0x00FF) * 4 +
         (readMsg[1] & 0x00FF);
}

task main () {
  int data = 0;
  SetSensorType(S1, sensorLowSpeed);
  wait1Msec(100);

  // first send the halt command to the SMUX
  HTSMUXsendCmd(S1, HTSMUX_CMD_HALT);
  // Initiate scan
  HTSMUXsendCmd(S1, HTSMUX_CMD_AUTODETECT);
  // Start normal operation
  HTSMUXsendCmd(S1, HTSMUX_CMD_RUN);
  wait1Msec(50);

  while (true) {
     // Read a single byte from the AD
     // buffer for channel 0 (SMUX port 1)
     data = HTSMUXreadAD (S1, 0, 0, 1, data);
     nxtDisplayTextLine(2, "%d", data);
     wait1Msec(100);
  }
}
```

NXC

```
#define HTSMUX_CMD_HALT         0x00
#define HTSMUX_CMD_AUTODETECT   0x01
#define HTSMUX_CMD_RUN          0x02

void HTSMUXsendCmd(byte smux, byte cmd)  {
  byte sendMsg[3];

  sendMsg[0] = 0x10;   // Address of SMUX
  sendMsg[1] = 0x20;   // Command register
  sendMsg[2] = cmd;    // Command to be sent

  // Send the command to the SMUX
  I2CWrite(smux, 0, sendMsg);

  // if the HTSMUX_CMD_AUTODETECT command has
  // been given, wait 500 ms
  if (cmd == HTSMUX_CMD_AUTODETECT) {
    Wait(500);

  // Wait 50ms after SMUX is halted
  } else if (cmd == HTSMUX_CMD_HALT) {
    Wait(50);
  }
}

int HTSMUXreadAD (byte smux, byte chan) {

  byte sendMsg[2];
  byte readMsg[2];
  const byte count = 2;

  sendMsg[0] = 0x10;    // Address of SMUX
                        // Buffer register
  sendMsg[1] = 0x36 + (chan * 2);

  // Query the SMUX and read the response
  if (I2CBytes(smux, sendMsg, count, readMsg))
    // ensure no weirdness from signed/unsigned
    // conversions
    return (readMsg[0] & 0x00FF) * 4 +
           (readMsg[1] & 0x00FF);
  else
    return 0;
}

task main () {
  int data = 0;
  SetSensorLowspeed(S1);
  Wait(100);

  // first send the halt command to the SMUX
  HTSMUXsendCmd(IN_1, HTSMUX_CMD_HALT);
  // Initiate scan
  HTSMUXsendCmd(IN_1, HTSMUX_CMD_AUTODETECT);
  // Start normal operation
  HTSMUXsendCmd(IN_1, HTSMUX_CMD_RUN);
  Wait(50);

  while (true) {
     // Read a single byte from the I2C
     // buffer for channel 0 (SMUX port 1)
     data = HTSMUXreadAD (IN_1, 0);
     NumOut(0, LCD_LINE2, data, true);
     Wait(100);
  }
}
```

Advanced operations

So what to do if you are using the Lego Light sensor and you want to turn the light on or off? Well, it's not as tricky as you might think. You can use the channel mode registers, 22H, 27H, 2CH and 31H to enable or disable the dig0 pin for each individual SMUX channel. This is the pin used by the Light Sensor to check if it should enable or disable the LED. Incidentally, this is also the method for switching a sound sensor from dB to dBA mode and the HiTechnic EOPD sensor from Long Range to Short Range mode. For this example we'll use the Light Sensor, however.

Das blinkenlichten – switching the light on and off

ROBOTC

```
#define HTSMUX_CHAN_NONE        0x00
#define HTSMUX_CHAN_DIG0_HIGH   0x04

void HTSMUXenableActive(tSensors smux,
                        byte chan,
                        bool active)  {

  ubyte sendMsg[4];

  sendMsg[0] = 3;      // size of I2C message
  sendMsg[1] = 0x10;   // Address of SMUX
                       // Channel mode register
  sendMsg[2] = 0x22 + (chan * 5);

  if (active)
    sendMsg[3] = HTSMUX_CHAN_DIG0_HIGH;
  else
    sendMsg[3] = HTSMUX_CHAN_NONE;

  // Send the command to the SMUX
  sendI2CMsg(smux, sendMsg, 0);
}
```

NXC

```
#define HTSMUX_CHAN_NONE        0x00
#define HTSMUX_CHAN_DIG0_HIGH   0x04

void HTSMUXenableActive(byte smux,
                        byte chan,
                        bool active)  {

  byte sendMsg[3];

  sendMsg[0] = 0x10;   // Address of SMUX
                       // Channel mode register
  sendMsg[1] = 0x22 + (chan * 5);

  if (active)
    sendMsg[2] = HTSMUX_CHAN_DIG0_HIGH;
  else
    sendMsg[2] = HTSMUX_CHAN_NONE;

  // Send the command to the SMUX
  I2CWrite(smux, 0, sendMsg);
}
```

When you call the `HTSMUXenableActive()` function, it is important that the SMUX is in a **halted** state. That means you must halt the SMUX using `HTSMUXsendCmd(S1, HTSMUX_CMD_HALT)`, issue the `HTSMUXenableActive()` for the right channel and the tell the SMUX to resume polling by putting it back in the **running** state with `HTSMUXsendCmd(S1, HTSMUX_CMD_RUN)`. That might seem awfully complicated but it's not that hard. Just always remember to **halt** the SMUX before modifying registers and put it back in a **running** state when you want to resume polling.

We can have a little fun with this function. Using four Light Sensors and a SMUX, you can make a Cylon "eye" by switching them on and off in a sequence.

You can watch a video of this in action here: http://www.youtube.com/watch?v=6IniYOOdBOc

By your command

ROBOTC

```
#define HTSMUX_CHAN_NONE        0x00
#define HTSMUX_CHAN_DIG0_HIGH   0x04

#define HTSMUX_CMD_HALT         0x00
#define HTSMUX_CMD_AUTODETECT   0x01
#define HTSMUX_CMD_RUN          0x02

void HTSMUXsendCmd(tSensors smux, byte cmd) {
  ubyte sendMsg[4];

  sendMsg[0] = 3;       // size of I2C message
  sendMsg[1] = 0x10;    // Address of SMUX
  sendMsg[2] = 0x20;    // Command register
  sendMsg[3] = cmd;     // Command to be sent

  // Send the command to the SMUX
  sendI2CMsg(smux, sendMsg, 0);

  // if the HTSMUX_CMD_AUTODETECT command has
  // been given, wait 500 ms
  if (cmd == HTSMUX_CMD_AUTODETECT) {
    wait1Msec(500);
  // Wait 50ms after SMUX is halted
  } else if (cmd == HTSMUX_CMD_HALT) {
    wait1Msec(50);
  }
}

void HTSMUXenableActive(tSensors smux,
                        byte chan,
                        bool active)  {

  ubyte sendMsg[4];

  sendMsg[0] = 3;      // size of I2C message
  sendMsg[1] = 0x10;   // Address of SMUX
  sendMsg[2] = 0x22 + (chan * 5); // channel
                                  // mode reg
  if (active)
    sendMsg[3] = HTSMUX_CHAN_DIG0_HIGH;
  else
    sendMsg[3] = HTSMUX_CHAN_NONE;

  // Send the command to the SMUX
  sendI2CMsg(smux, sendMsg, 0);
}

task main () {
  SetSensorType(S1, sensorLowSpeed);
  wait1Msec (100);

  // first send the halt command to the SMUX
  HTSMUXsendCmd(IN_1, HTSMUX_CMD_HALT);
  // Initiate scan
  HTSMUXsendCmd(IN_1, HTSMUX_CMD_AUTODETECT);

  while (true) {
    for (int i = 0; i < 4; i++) {
      HTSMUXenableActive(IN_1, i, true);
      Wait(20);
      HTSMUXsendCmd(IN_1, HTSMUX_CMD_RUN);
      Wait(100);
      HTSMUXsendCmd(IN_1, HTSMUX_CMD_HALT);
      HTSMUXenableActive(IN_1, i, false);
      Wait(10);
    }
  }
}
```

NXC

```
#define HTSMUX_CHAN_NONE        0x00
#define HTSMUX_CHAN_DIG0_HIGH   0x04

#define HTSMUX_CMD_HALT         0x00
#define HTSMUX_CMD_AUTODETECT   0x01
#define HTSMUX_CMD_RUN          0x02

void HTSMUXsendCmd(byte smux, byte cmd)  {
  byte sendMsg[3];

  sendMsg[0] = 0x10;   // Address of SMUX
  sendMsg[1] = 0x20;   // Command register
  sendMsg[2] = cmd;    // Command to be sent

  // Send the command to the SMUX
  I2CWrite(smux, 0, sendMsg);

  // if the HTSMUX_CMD_AUTODETECT command has
  // been given, wait 500 ms
  if (cmd == HTSMUX_CMD_AUTODETECT) {
    Wait(500);
  // Wait 50ms after SMUX is halted
  } else if (cmd == HTSMUX_CMD_HALT) {
    Wait(50);
  }
}

void HTSMUXenableActive(byte smux,
                        byte chan,
                        bool active) {

  byte sendMsg[3];

  sendMsg[0] = 0x10;   // Address of SMUX
                       // Channel mode register
  sendMsg[1] = 0x22 + (chan * 5);

  if (active)
    sendMsg[2] = HTSMUX_CHAN_DIG0_HIGH;
  else
    sendMsg[2] = HTSMUX_CHAN_NONE;

  // Send the command to the SMUX
  I2CWrite(smux, 0, sendMsg);
}

task main () {
  SetSensorLowspeed(S1);
  Wait(100);

  // first send the halt command to the SMUX
  HTSMUXsendCmd(IN_1, HTSMUX_CMD_HALT);
  // Initiate scan
  HTSMUXsendCmd(IN_1, HTSMUX_CMD_AUTODETECT);

  while (true) {
    for (int i = 0; i < 4; i++) {
      HTSMUXenableActive(IN_1, i, true);
      Wait(20);
      HTSMUXsendCmd(IN_1, HTSMUX_CMD_RUN);
      Wait(100);
      HTSMUXsendCmd(IN_1, HTSMUX_CMD_HALT);
      HTSMUXenableActive(IN_1, i, false);
      Wait(10);
    }
  }
}
```

Dealing with the unexpected

As with all things in life, not everything goes according to plan. The SMUX is no exception. So how do you find out what's bugging the SMUX when you stop getting sane sensor data back from it? How do you check if the batteries powering the SMUX are still good enough? Have a look at the status register and the meaning of its bit fields in Appendix B. You can read this register without needing to be in a **halted** state.

Reading the SMUX status register

ROBOTC

```
#define HTSMUX_STAT_NORMAL      0x00
#define HTSMUX_STAT_BATT        0x01
#define HTSMUX_STAT_BUSY        0x02
#define HTSMUX_STAT_HALT        0x04
#define HTSMUX_STAT_ERROR       0x08

byte HTSMUXreadStatus(tSensors smux)  {

  ubyte sendMsg[3];
  ubyte readMsg[1];

  sendMsg[0] = 2;      // size of I2C message
  sendMsg[1] = 0x10;   // Address of SMUX
  sendMsg[2] = 0x21;   // Status register

  // Query the SMUX and read the response
  sendI2CMsg(smux, sendMsg, 1);
  wait1Msec(10);
  readI2CReply(smux, readMsg, 1);

  return readMsg[0];
}

task main () {
  byte status = 0;
  SetSensorType(S1, sensorLowSpeed);
  wait1Msec (100);

  while (true) {
    eraseDisplay();
    status = HTSMUXreadStatus(S1);
    nxtDisplayTextLine(1, "Status: %d",
                       status);
    if(status & HTSMUX_STAT_BATT ==
       HTSMUX_STAT_BATT)
      nxtDisplayTextLine (3, "No battery");
    wait1Msec (100);
  }
}
```

NXC

```
#define HTSMUX_STAT_NORMAL      0x00
#define HTSMUX_STAT_BATT        0x01
#define HTSMUX_STAT_BUSY        0x02
#define HTSMUX_STAT_HALT        0x04
#define HTSMUX_STAT_ERROR       0x08

byte HTSMUXreadStatus(byte smux)  {

  byte sendMsg[2];
  byte readMsg[1];
  int status = 0;
  const byte count = 1;

  sendMsg[0] = 0x10;   // Address of SMUX
  sendMsg[1] = 0x21;   // Status register

  // Send the command to the SMUX
  if(I2CBytes(smux, sendMsg, count, readMsg))
    return readMsg[0];
  else
    return -1;
}

task main () {
  byte status = 0;
  SetSensorLowspeed(IN_1);
  Wait(100);

  while (true) {
    ClearScreen();
    status = HTSMUXreadStatus(IN_1);
    TextOut(0, LCD_LINE1, "Status: ");
    NumOut(45, LCD_LINE1, status);

    if(status & HTSMUX_STAT_BATT ==
       HTSMUX_STAT_BATT)
      TextOut(0, LCD_LINE3, "No battery");
    Wait(100);
  }
}
```

The status register function is so simple that I've included into a program directly as it does not need any helper functions. Play with it a bit, disconnect the battery pack and see what happens.

3rd Party ROBOTC Driver Suite

Some of the functionality in this document is part of the 3rd Party ROBOTC Driver Suite. The examples given in this document do not have any error checking or error recovery built into them, which the suite obviously does. The suite also handles access to the SMUX transparently, so you don't need to reassemble the I2C data bytes into a 16 or 32 bit value.

The website for this suite can be found here: http://rdpartyrobotcdr.sourceforge.net/, which also has a copy of the complete API documentation and source code for you to browse at your leisure.

I will be publishing an NXC port of my driver suite in the near future, so stay tuned.

Appendix A Register Layout + Function

Address	Type	Contents
00 – 07H	chars	Sensor version number
08 – 0FH	chars	Manufacturer
10 – 17H	chars	Sensor type
18 – 1FH	bytes	Not used
20H	byte	Command
21H	byte	Status
22H	byte	Channel 1 mode
23H	byte	Channel 1 type
24H	byte	Channel 1 I2C byte count
25H	byte	Channel 1 I2C device address
26H	byte	Channel 1 I2C memory address
27H	byte	Channel 2 mode
28H	byte	Channel 2 type
29H	byte	Channel 2 I2C byte count
2AH	byte	Channel 2 I2C device address
2BH	byte	Channel 2 I2C memory address
2CH	byte	Channel 3 mode
2DH	byte	Channel 3 type
2EH	byte	Channel 3 I2C byte count
2FH	byte	Channel 3 I2C device address
30H	byte	Channel 3 I2C memory address
31H	byte	Channel 4 mode
32H	byte	Channel 4 type
33H	byte	Channel 4 I2C byte count
34H	byte	Channel 4 I2C device address
35H	byte	Channel 4 I2C memory address
36H	byte	Channel 1 upper 8 bits
37H	byte	Channel 1 lower 2 bits
38H	byte	Channel 2 upper 8 bits
39H	byte	Channel 2 lower 2 bits
3AH	byte	Channel 3 upper 8 bits
3BH	byte	Channel 3 lower 2 bits
3CH	byte	Channel 4 upper 8 bits
3DH	byte	Channel 4 lower 2 bits
3E, 3FH	chars	Reserved
40 – 4FH	bytes	Channel 1 I2C buffer
50 – 5FH	bytes	Channel 2 I2C buffer
60 – 6FH	bytes	Channel 3 I2C buffer
70 – 7FH	bytes	Channel 4 I2C buffer

Field	Function
Sensor version	Reports a revision number in the format “Vn.m” where n is the major version number and m is the revision level. Revision numbers typically reflect the firmware level. The version number is used to indicate the hardware level
Manufacturer	Contains “HiTechnc”.
Sensor type	Contains “SensrMux”.
Status	Used to monitor the operating state of the sensor multiplexer.
Channel 1/2/3/4 mode	Used to either read or set the operating mode for channels 1, 2, 3 or 4.
Channel 1/2/3/4 type	Used to read the sensor type for channels 1, 2, 3 or 4.
Channel 1/2/3/4 I2C byte count	Used to specify the I2C read length for channels 1, 2, 3 or 4.
Channel 1/2/3/4 I2C device address	Used to either read or set the I2C device address for channels 1, 2, 3 or 4.
Channel 1/2/3/4 I2C memory address	Used to either read or set the I2C memory address for channels 1, 2, 3 or 4.
Channel 1/2/3/4 upper 8 bits	Hold the upper 8 bits of the most recent 10 bit value obtained from channels 1, 2, 3 or 4 analog inputs.
The Channel 1/2/3/4 lower 2 bits	Hold the lower 2 bits of the most recent 10 bit value obtained from channels 1, 2, 3 or 4 analog inputs.
The Channel 1/2/3/4 I2C buffer	Hold up to 16 bytes read from channels 1, 2, 3 or 4 I2C interfaces if those channels are set to I2C mode.

Appendix B: SMUX Status

The current status of the SMUX is reported in the 21H register as specified below.

D7	D6	D5	D4	D3	D2	D1	D1
-	-	-	-	Error	Halt	Busy	Batt

The status bits have the following meaning

Bit	Meaning if set
Batt	No/low battery voltage detected
Busy	Autodetect in progress
Halt	SMUX halted
Error	Command error detected

Appendix C: Channel Mode

The channel mode registers (22H, 27H, 2CH and 31H) for each connected are as specified below.

D7	D6	D5	D4	D3	D2	D1	D1
-	-	-	Slow	Dig 1	Dig 0	9V en	I2C

The channel mode bits have the following meaning

Bit	Meaning if set
I2C	The connected sensor is a digital sensor
9v en	The analog pin is used as 9v supply pin
Dig 0	The dig 0 pin is driven high
Dig 1	The dig 1 pin is driven high
Slow	The I2C read rate is decreased from 80Hz to 20Hz

Appendix D: Channel Type

The channel type registers (23H, 28H, 2DH and 32H) contain the type of sensor that was detected during the autodetect phase. The registers can have the following values.

Type	Sensor
0	Not an I2C sensor, also used for analogue sensors
1	LEGO ultrasonic sensor
2	HiTechnic compass sensor
3	HiTechnic color sensor
4	HiTechnic accelerometer sensor
5	HiTechnic IR seeker sensor
6	HiTechnic prototype board
7	HiTechnic new color sensor
8	Reserved
9	HiTechnic new IR seeker sensor

Appendix E: Supported sensors

Digital	Analogue
LEGO ultrasonic sensor	HiTechnic EOPD
HiTechnic compass sensor	HiTechnic Gyro
HiTechnic color sensor	LEGO Light Sensor
HiTechnic accelerometer sensor	LEGO Touch Sensor
HiTechnic IR seeker sensor	
HiTechnic prototype board	
HiTechnic new color sensor	
HiTechnic new IR seeker sensor	