Hands-on Lab

NXC Programming – RS-485 Communications

The Lego NXT Brick can communicate with other peripherals via RS-485 serial communications. Port 4 on the Brick provides this high-speed full-duplex capability. Many peripherals like scanners, joysticks, keypads, and the XL-320 servo use this serial communication protocol. RS-485 can also connect the NXT Brick to other micro-processors and computers that have this port. This lab explores RS-485 NXC programming to send and receive packets.

Preliminary: NXT and Laptop Setup

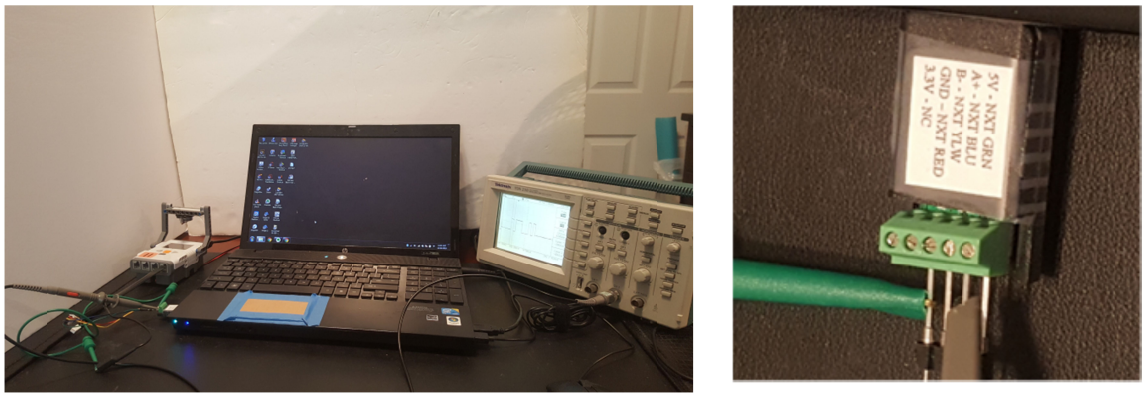

Figure A: NXT communicating to a Windows-based laptop via RS-485 (left). A USB-to-RS485 (right) connects into the laptop.

Most laptops today feature USB ports for serial communications. As such, a USB-to-RS485 dongle can plug into the laptop. **Figure A** (right) shows one purchased from Amazon.com; it is labeled to indicate the function of each screw terminal line. A spliced NXT cable (**Figure B left**) was then inserted into the dongle (**Figure B right**).

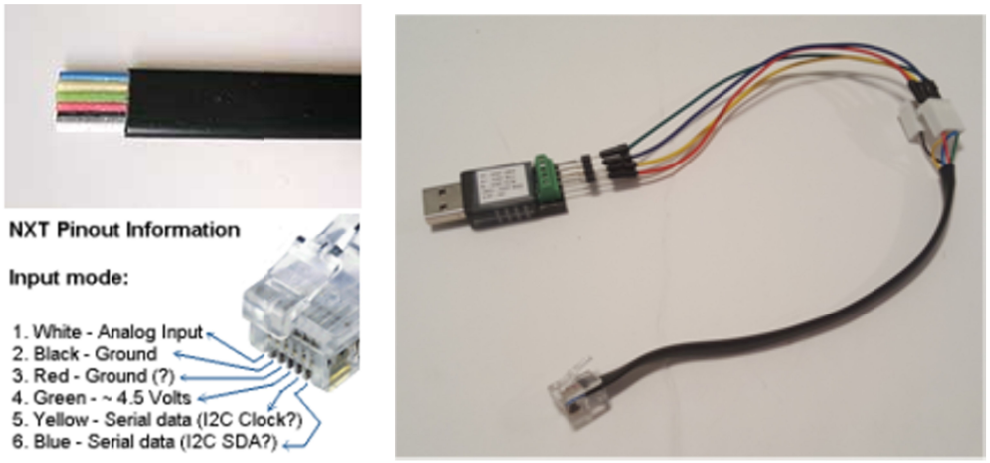

Figure B: NXC Cable pinout (left). NXC cabled spliced and inserted into USB-to-RS485 dongle (right)

Lastly, a terminal emulator is needed to establish serial communications. Hercules is an example of a free Windows-based emulator. Installing this executable allows one to choose settings like baud rate and parity.

Concept 1 ASCII Character Transmission: Read from PC

RS-232 and its descendent RS-485 employ HI and LO voltages to represent ON and OFF binary states. In RS-232, the voltages are -12V to +12V values. RS-485 uses lower values, but more importantly, use voltage differences in order to offer more robust transmission. Most micro-processor based systems employ TTL (transistor-to-transistor logic) voltages where LO and HI are respectively 0 to +5V. As such, a converter is employed to transform RS-232 or RS-485 states to TTL.

Step 1: In BricxCC open and compile `nxtReadFromPC1_0b.nxc`

Figure 1A depicts an NXC program that transmits ASCII characters from the PC to an NXT Brick. After declaring variables, the four steps involve (see yellow highlight): (1) configuring Port 4 on the Brick, for RS-485 communications; (2) activating RS-485 protocol; (3) setting the baud rate and parity; and (4) waiting at least 1 millisecond for these calls to be established.

```
// FILE: nxtReadFromPC1_0b.nxc - Works!
// DATE: 06/08/19 15:01
// AUTH: P.Oh
// DESC: PC terminal program sents ASCII string to NXT.  NXT displays string
// REFS: ME425 notes 485Master1_0.nxc nd 485Slave1_0.nxc
//       Hercules terminal PC program:
//       https://www.hw-group.com/software/hercules-setup-utility
// NOTE: Serial setup Baud 115200, no parity, no handshaking

task main() {

  byte readBuffer[];          // array to store bytes received from PC
  string charsRead;           // string of ASCII characters read from PC

  UseRS485();                                 // (1) Configure S4 for RS-485
  RS485Enable();                              // (2) Activate RS-485
  RS485Uart(HS_BAUD_4800, HS_MODE_DEFAULT);   // (3) Baud 4800 and default parity

  Wait(MS_1);                // (4) Wait briefly for port settings to be ready

  while(true) { // keep reading and displaying strings received from PC until abort
    while(!RS485DataAvailable()) {
      // if no ASCII chars available, then do nothing
    };
    // Bytes ready, so now them
    RS485Read(readBuffer);
    // Convert bytes into ASCII string
    charsRead = ByteArrayToStr(readBuffer);
    // Display on middle of Brick's screen
    ClearScreen();
    TextOut(0, LCD_LINE4, charsRead);
    // Clear buffer
    readBuffer = 0;
    // Wait briefly
    Wait(100);
  }; // end while
} // end main
```

Figure 1A: Listing for `nxtReadFromPC1_0b.nxc` for transmitting from the PC to the Brick

A `while`-loop checks for any transmitted data with an NXC function call (`RS485DataAvailable`). When data is available, the NXC function `RS485Read` collects that data into the array `readBuffer`. This data is converted into a character string and displayed on the Brick. The array is then reset to 0 and the program loops.

Step 2: Configure Hercules

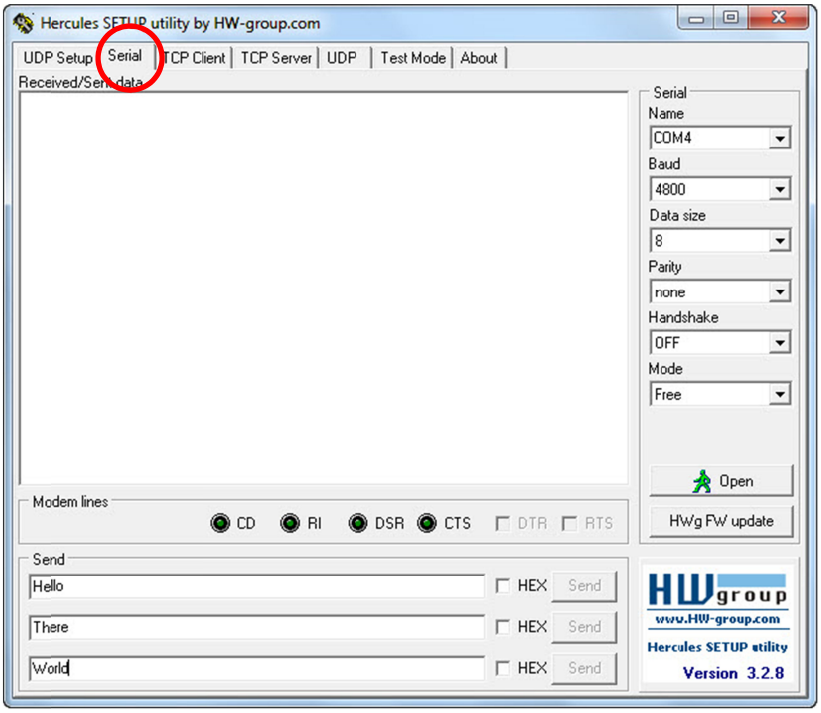

Figure 1B: Hercules Terminal Emulator with Baud set to 4800 BPS and 8N1

A Terminal Emulator, like Hercules, allows one to establish serial communications between the PC and any serially connected devices. **Figure 1B** shows the `Serial` tab selected (red circle). The right pull-down fields show that that `COM4` is selected. In Windows, one can use Device Manager to see port names for any serially connected devices. In this particular example, The NXT Brick was connected to COM4. Next, a 4800 BPS (bits-per-second) baud rate was chosen. This is because the listing in **Figure 1A** established `RS485Uart` at 4800. 8N1 refers to 8-bits, no parity, and 1 stop bit. This is a common setting where 8-bits represents data, the data is sent asynchronously, and a stop bit is used to flag the end of transmission.

Step 3: Run `nxtReadFromPC1_0b.nxc` and Open the Port in Hercules

Once the Brick runs the NXT program, it waits for a connection. Clicking the `Open` button in Hercules confirms this connection. At the bottom of **Figure 1B**, one can enter text and clicking Send, transmits their byte equivalents to the NXT Brick. Recall the NXT program in **Figure 1A** uses the NXC functions `ByteArrayToStr` to convert any received bytes into string characters. These string characters are displayed on the Brick using `TextOut`.

Step 4: Connect Oscilloscope

ASCII is the international standard to represent alpha-numeric characters. For example, the uppercase letter G has the ASCII value of 71 Decimal (0x47) or 0100 0111 in binary. One can visually see how G is represented in binary by computers, using an oscilloscope.

Recall that a 4800 bit/second baud rate was selected. This is 0.21 seconds/bit. Hence, setting an oscilloscope at this rate will capture this transmission rate. The improve visibility one should set the oscilloscope to trigger when voltages are above 0 Volts (e.g. 100 mV). **Figure 1C** shows a capture of the oscilloscope's display when Hercules transmits the letter G to the NXT Brick.

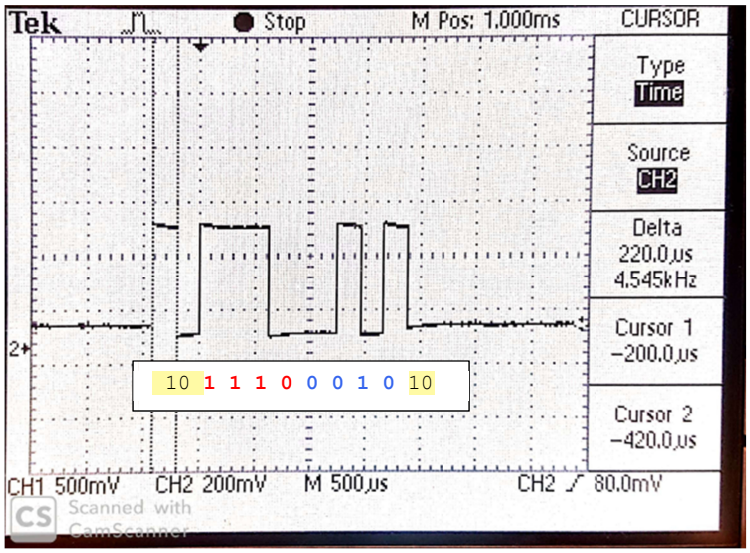

Figure 1C: Oscilloscope display of the uppercase letter G

The two vertical lines are the scope's time cursors and show a `Delta` of 220 microseconds. The first transition from HI to LO is the start bits (represented by the left-most yellow highlighted text in **Figure 1C**). What follows are eight 220 microsecond slices at HI-HI-HI-LO-LO-LO-HI-LO voltage levels (represented by 4 red and 4 blue colored texts). Finally, there is HI-LO transmission which are the stop bits (right-most yellow highlighted text).

Recall that G has binary value 0100 0111. Thus, the scope shows that the least-most-significant bit (right-most) is transmitted first. In other words, G is transmitted as 1110 0010.

Exercises

1.1 What is the ASCII binary value for the uppercase character U? Sketch what should be seen on the scope when U is transmitted. Confirm your sketch with a photo of the oscilloscope display when U is transmitted.

Concept 2 ASCII Transmission – Read from PC

The previous concept allowed the NXT Brick to read characters transmitted by a PC. Vice-versa, this concept allows on the PC to read characters transmitted by the NXT.

Step 1: Open and compile `SendByButton.nxc` file

```
//

task main() {
  bool Rbutton, Lbutton, Mbutton;
  UseRS485();                              // (1) Port S4 configured for RS485
  RS485Enable();                           // (2) turn on RS485
  RS485Uart(HS_BAUD_57600, HS_MODE_DEFAULT); // (3) initialize UART 57600
  Wait(100);                               // (4) Wait at least 1 msec
  TextOut(0, LCD_LINE1, "  NXT <> laptop");
  while (true) {
     if(ButtonPressed(BTNRIGHT, FALSE)){        // if Right Button is pressed
         while(ButtonPressed(BTNRIGHT, FALSE)); // Wait for Button is released
         SendRS485String(" Right "); // then send string
     }else
     if(ButtonPressed(BTNLEFT, FALSE)){
         while(ButtonPressed(BTNLEFT, FALSE));
         SendRS485String(" Left ");
     }else
     if(ButtonPressed(BTNCENTER, FALSE)){
         while(ButtonPressed(BTNCENTER, FALSE));
         SendRS485String(" Center ");
     }
  }
}
```

Figure 2A: Listing for `SendByButton.nxc`

Figure 2A implements the 4-step process for establishing the NXT Brick's Port 4 for RS-485 communications at 8N1 and 4800 baud.

Step 2: Execute Hercules

Like in Concept 1, configure Hercules for the appropriate serial port (e.g. COM4) and settings (e.g. 4800 baud, 8-bits, no parity, 1 stop bit).

Step 3: Execute `SendByButton.nxc` and open port on Hercules

Unlike a PC, the NXT Brick only has 4 buttons. As such, strings to be transmitted are associated with these buttons. **Figure 2A** shows that pushing the left arrow button uses the NXT function SendRS485String to transmit the string " Left " (note the white space before and after the word). Similarly, " Right " is transmitted for the right arrow button.

Concept 3 Master-Slave Communications

An NXT cable is connected on Port 4 of two NXT Bricks. This allows the two Bricks to communicate via RS-485; one will be called the Master and the other, a Slave.

```
// FILE: 485Master1_0.nxc - Works!
// DATE: 09/26/16 12:45
// AUTH: P.Oh
// DESC: Two NXT bricks connected together on their Port S4 (i.e. RS-485 communications)
//       This code runs on Master brick.  485Slave1_0.nxc runs on Slave brick.
//       As long as Slave is on and sending messages, Master iterates and displays number

inline void WaitForMessageToBeSent()
{
  while(RS485SendingData())
  Wait(MS_1);
}

task main() {
  UseRS485(); // (1) Port S4 configured for RS485
  RS485Enable(); // (2) turn on RS485
  RS485Uart(HS_BAUD_DEFAULT, HS_MODE_DEFAULT); // (3) initialize UART to default values
  Wait(MS_1); // (4) wait a bit so all's activated

  int i;
  byte buffer[];
  string msg;
  byte cnt;

  while (true) {
    msg = "Master " + NumToStr(i);
    TextOut(0, LCD_LINE1, msg);
    // send the # of bytes (5 bytes)
    cnt = ArrayLen(msg);
    SendRS485Number(cnt);
    WaitForMessageToBeSent();

    // wait for ACK from recipient
    until(RS485DataAvailable());
    RS485Read(buffer);

    // now send the message
    SendRS485String(msg);
    WaitForMessageToBeSent();

    // wait for ACK from recipient
    until(RS485DataAvailable());
    RS485Read(buffer);

    i++;
  }

  // disable RS485 (not usually needed)
  RS485Disable();
} // end of main
```

Figure 3A: Listing for `485Master1_0.nxc`

The first Brick will be deemed Master and will execute `485Master1_0.nxc`. The NXC program begins by configure Port 4 for RS485 communications at the Brick's default settings. The NXC constant `HS_BAUD_DEFAULT` represents 921,600 BPS, the fastest rate available.

An endless `while` loop increments a variable (`i`). The string `msg` contains the characters "`Master` " plus the value of the variable (`i`). The NXC function `ArrayLen` calculates the

number of bytes for the resulting string (`msg`) and stores it in the variable cnt. `SendRS485Number` transmits this number to the Slave.

To confirm that the Slave received the transmission, the NXC function `RS485DataAvailable` is called. Once confirmed, the Master sends `msg` via a call to `SendRS485String`.

```
// FILE: 485Slave1_0.nxc - Works!
// DATE: 09/26/16 12:47
// AUTH: P.Oh
// DESC: Two NXT bricks connected together on their Port S4 (i.e. RS-485
communications)
//       This code runs on Slave brick.  485Master1_0.nxc runs on Master brick.
//       When Slave is off, then Master stops.  When Slave is on, the Master iterates

inline void WaitForMessageToBeSent()
{
  while(RS485SendingData())
  Wait(MS_1);
}

task main() {
  UseRS485(); // (1) Port S4 configured for RS485
  RS485Enable(); // (2) turn on RS485
  RS485Uart(HS_BAUD_DEFAULT, HS_MODE_DEFAULT); // (3) initialize UART to default values
  Wait(MS_1); // (4) wait a bit so all's activated

  int i;
  byte buffer[];
  string msg;
  byte cnt;

  while (true) {
    msg = "Slave " + NumToStr(i);
    TextOut(0, LCD_LINE1, msg);
    // send the # of bytes (5 bytes)
    cnt = ArrayLen(msg);
    SendRS485Number(cnt);
    WaitForMessageToBeSent();

    // wait for ACK from recipient
    until(RS485DataAvailable());
    RS485Read(buffer);

    // now send the message
    SendRS485String(msg);
    WaitForMessageToBeSent();

    // wait for ACK from recipient
    until(RS485DataAvailable());
    RS485Read(buffer);

    i++;
  }

  // disable RS485 (not usually needed)
  RS485Disable();
} // end of main
```

Figure 3B: Listing for `485Slave1_0.nxc`

A second Brick runs `485Slave1_0.nxc` given in **Figure 3B** and acts as a Slave. Here, the program begins by configure Port 4 for RS-485 communications at default settings. Like Figure 3A, an endless while loop sends to the Master, the number of bytes for the string it will send, waits for acknowledgement from the Master, and then transmits strings.

Congratulations! You can program the NXT Brick for RS-485 communications.

Exercises

2.1 Write an NXC program to guess a number between 1 and 21. The PC transmits the guessed number to the Brick. The Brick's user uses the left and arrow buttons to respectively transmit "lower" and "higher" strings back to the PC. When the PC transmits the correct number, the Brick's user presses the Orange button to say "Correct!"