```
===============================================================================
RS-485 Controller Solution

2016-10-08  Peter S'heeren, Axiris
-------------------------------------------------------------------------------
```

1. Overview

The RS-485 Controller is designed for use in a circuit that bridges between an RS-485 bus and a serial Rx/Tx interface. The controller controls the direction of the RS-485 transceiver and is particularly useful in a system where the serial interface doesn't expose a direction pin.

A command line tool accompanies the controller. This program complements the hardware. Together the controller and the program form a complete solution.

```
2. Controller
-------------

2.1. Function Block
-------------------

The controller is based on an ATtiny45 microcontroller. An SPI or I2C interface
enables access to a register map from software.

High-level view of the controller:

    RS485CTRL (ATtiny45)
    +-------------------+
    |               TXD|<--- Serial TxD line probe
    |                   |
    |    RS-485     DIR|---> RS-485 direction
    |  CONTROLLER       |
    |               I/F|<==> I2C/SPI interface
    |                   |
    |               DBG|---> Debug output
    +-------------------+

  Pins:

    TXD - Serial Transmit Data Line Probe

      Type:   input

      Input:  0 - Corresponds with RS-232 logic 0, space, +3..+15V.
              1 - Corresponds with RS-232 logic 1, mark, -15..-3V.

      This pin probes the serial Tx line.

    DIR - Direction

      Type:   output

      Reset:  1

      Output: 0 - Force transceiver to receive from RS-485 bus.
              1 - Force transceiver to transmit to RS-485 bus.

      This pin controls the direction of the RS-485 transceiver.

    I/F - Interface

      The design supports multiple interfaces. Each interface implies a
      specific set of pins:
      * SPI: SCLK, MOSI, MISO (no slave select).
      * I2C: SCL, SDA.

    DBG - Debug Output

      The controller can optionally transmit debug information to a serial
      monitor program (e.g. PuTTY) for debugging purposes.
```

Typical application:

```
           XCVR                3.3V or 5V level       UART
          +-----+                                     +---+
   RS-485 |  >  |------------------------------------>|Rx |
  <~~~~~~>|  <  |<------------+-------------------+---|Tx |
          |  DIR|<--+         |                   |   +---+
          +-----+   |  NAND   |                   |
                    |  +--+   |    RS485CTRL      |
                    +--|  |<--+  +------------+   |
                       |  |<-----|DIR      TXD|<--+   HOST
                       +--+      |            |       +---+
                                 |            |  SPI  |   |
                                 |         I/F|<=====>|   |
                                 +------------+       |   |
                                                      +---+
```

XCVR: RS-485 bus transceiver.

UART: Serial interface. Typically part of the host.

HOST: System with SPI master interface.

NAND: 2-input NAND gate.

When the UART transmits a data word over the Tx line, the NAND gate will immediately instruct the transceiver to transmit. Doing so gives the controller enough time to detect the start of the data word and take over control of the direction of the transceiver.

2.2. Reset Sequence

1. All registers are set to zero.
2. DIR is set to one, so the RS-485 transceiver is set to receive data from the RS-485 bus.
3. The contents of the EEPROM are read into the register map, given that the contents of the EEPROM are valid.

2.3. Register Map

The controller's register map is accessible via the SPI or I2C interface.

```
        7     6     5     4     3     2     1     0
     +-----+-----+-----+-----+-----+-----+-----+-----+
+00h |                   CNT[7..0]                   |-+
     +-----+-----+-----+-----+-----+-----+-----+-----+ |
+01h |                  CNT[15..8]                   |
     +-----+-----+-----+-----+-----+-----+-----+-----+ CNT
+02h |                  CNT[23..16]                  |
     +-----+-----+-----+-----+-----+-----+-----+-----+ |
+03h |                  CNT[31..24]                  |-+
     +-----+-----+-----+-----+-----+-----+-----+-----+
+04h |           reserved          |RSOTL| OOD | DEN | CTRL
     +-----+-----+-----+-----+-----+-----+-----+-----+
+05h |           reserved                | TLD | DTD | STATUS
     +-----+-----+-----+-----+-----+-----+-----+-----+
+06h |           reserved                | RRE | WRE | CMD
     +-----+-----+-----+-----+-----+-----+-----+-----+
```

```
reserved - Reserved bit field

  Reset value:  0

  Read access:  Read state.

  Write access: Always write zero.


CNT - Count Register

  Reset value:  00000000_00000000_00000000_00000000b

  Read access:  Read one byte of the count value.

  Write access: Write one byte of the count value.


CTRL - Control Register

  DEN - Detection Enabled

    Reset value:  0

    Read access:  0 - Detection is disabled.
                  1 - Detection is enabled.

    Write access: 0 - Disable detection.
                  1 - Enabled detection.


  OOD - One-off Detection

    Reset value:  0

    Read access:  0 - One-off detection is disabled.
                  1 - One-off detection is enabled.

    Write access: 0 - Disable one-off detection.
                  1 - Enabled one-off detection.
  RSOTL - Reset SPI on TxD Low

    Reset value:  0

    Read access:  0 - Feature is disabled.
                  1 - Feature is enabled.

    Write access: 0 - Disable feature.
                  1 - Enabled feature.

    This flag is only available with the SPI interface.


STATUS - Status Register

  DTD - Detected

    Reset value:  0

    Read access:  0 - TxD high-to-low transition detected.
                  1 - TxD high-to-low transition not detected.

    Write access: 0 - Clear flag.
                  1 - Set flag.
  TLD - TxD Low Detected

    Reset value:  0

    Read access:  0 - TxD low detected.
                  1 - TxD low not detected.

    Write access: 0 - Clear flag.
                  1 - Set flag.


CMD - Command Register

  WRE - Write registers to EEPROM

    Reset value:  0

    Read access:  0 - Always zero.

    Write access: 0 - No effect.
                  1 - Start command.

    Set WRE.1 to write registers CNT and CTRL to EEPROM.


  RRE - Read registers from EEPROM

    Reset value:  0

    Read access:  0 - Always zero.

    Write access: 0 - No effect.
                  1 - Start command.

    Set RRE.1 to read registers CNT and CTRL from EEPROM.


2.4. EEPROM
-----------

        7     6     5     4     3     2     1     0
     +-----+-----+-----+-----+-----+-----+-----+-----+
+00h |                    CNT[7..0]                  |-+
     +-----+-----+-----+-----+-----+-----+-----+-----+ |
+01h |                   CNT[15..8]                  |
     +-----+-----+-----+-----+-----+-----+-----+-----+ CNT
+02h |                   CNT[23..16]                 |
     +-----+-----+-----+-----+-----+-----+-----+-----+ |
+03h |                   CNT[31..24]                 |-+
     +-----+-----+-----+-----+-----+-----+-----+-----+
+04h |           reserved          |RSOTL| OOD | DEN | CTRL
     +-----+-----+-----+-----+-----+-----+-----+-----+

The EEPROM stores a subset of the register map: CNT and CTRL.

After reset, the controller reads the EEPROM into the registers. This is
functionally equivalent as setting CMD.RRE to one.

When the host sets CMD.WRE to one, the controller writes registers CNT and CTRL
to EEPROM.

When the host sets CMD.RRE to one, the controller reads the EEPROM contents
into the corresponding registers, given that the contents of the EEPROM are
valid.
```

2.5. TxD Detection

When CTRL.DEN is one, the controller continuously samples the TXD pin. When the controller detects a transition from high to low on the TXD pin, a series of actions are set in motion. The controller will:

1. Set the DIR output low.
2. Delay for a period determined by the CNT register.
3. Sample the TXD pin once more. If the input state is low, CTRL.TLD is set to one.
4. Set the DIR output high.
5. Set CTRL.DTD to one.
6. If CTRL.OOD is one, set CTRL.DEN to zero.

2.6. Count-based Delay

```
Time    --->

RS485   ----+---------------------+----
TXD         |XXXXXXXXXXXXXXXXXXXXX|       Transmission of a data word.
input       +---------------------+

XCVR    ----+                    +-----
DIR         |                    |        Intended state of XCVR's DIR input.
input       +--------------------+

RS485   -----+                   +-----
DIR          |                   |        Software-controlled DIR output.
output       +-------------------+

Delay        |-------------------|        Delay period based on CNT register.
```

The total delay is the sum of:

1. A fixed overhead delay.
2. A period that's proportional with register CNT.

There's no relation between the count register CNT and absolute time i.e. don't try to translate CNT to a period in seconds.

The program incorporates a procedure for detecting the correct count value for a given baudrate and a given word size.

2.7. Interfaces

2.7.1. SPI Interface

The SPI interface is implemented using the ATtiny45's USI. The USI hardware directly interfaces with SCLK, MOSI and MISO.

Software is assumed to implement the slave select functionality. However, this may lead to race conditions between software and hardware. Since the controller can't reliably implement slave select, it doesn't use slave select at all.

A slave select input is normally used to determine the boundaries of a data packet. In the absence of a slave select input, the controller uses a different approach: it treats each transfer of eight bits as a 1-byte data packet.

The host sends bytes through the MOSI line and receives bytes through the MISO line. Each byte sent over MOSI is encoded as two nibbles:

```
   7     4 3     0        Packet types:
  +-------+-------+
  |       |       |         0000b - DATA
  +-------+-------+         0001b - WRITE
    |       |               0010b - READ
    value   packet type
```

The WRITE packet puts the controller in write mode allowing the host to write values to the register map by sending DATA packets. The host must send two DATA packets to write a single byte value to the current register.

The WRITE packet specifies the first register to write. After the controller has received two DATA packets and written the register, the register index increments up to the last register and wraps around zero after the last register.

```
        MOSI               MISO          Controller
   7             0    7             0
  +-------+-------+  +---------------+ - 1. Transmits value of previous state.
  |  reg  |0 0 0 1|  |               |   2. Enters write mode.
  +-------+-------+  +---------------+   3. Saves the specified register index.

  +-------+-------+  +---------------+
  |  low  |0 0 0 0|  |   undefined   | - 1. Saves the low nibble.
  +-------+-------+  +---------------+

  +-------+-------+  +---------------+ - 1. Writes low nibble and high nibble
  |  high |0 0 0 0|  |   undefined   |      to register.
  +-------+-------+  +---------------+   2. Points to next register.

  +-------+-------+  +---------------+
  |  low  |0 0 0 0|  |   undefined   | - 1. Saves the low nibble.
  +-------+-------+  +---------------+

  +-------+-------+  +---------------+ - 1. Writes low nibble and high nibble
  |  high |0 0 0 0|  |   undefined   |      to register.
  +-------+-------+  +---------------+   2. Points to next register.
```

The READ packet puts the controller in read mode. The packet specifies the first register to be read.

Upon reception of the READ packet, the controller immediately reads the specified register and returns the value during the reception of the next byte packet. To read subsequent registers, the host must send one or more DATA packets to the controller. As a result, the host receives the result of reading a register when it sends a new byte packet.

```
       MOSI                MISO           Controller
   7             0    7             0
  +-------+-------+  +---------------+ - 1. Transmits value of previous state.
  |  reg  |0 0 1 0|  |               |   2. Enters write mode.
  +-------+-------+  +---------------+   3. Reads the specified register.

  +-------+-------+  +---------------+ - 1. Transmits the register value.
  |0 0 0 0|0 0 0 0|  |   register    |   2. Reads the next register.
  +-------+-------+  +---------------+

  +-------+-------+  +---------------+ - 1. Transmits the register value.
  |0 0 0 0|0 0 0 0|  |   register    |   2. Reads the next register.
  +-------+-------+  +---------------+

  +-------+-------+  +---------------+ - 1. Transmits the register value.
  |       |       |  |   register    |   2. Action depends on packet type.
  +-------+-------+  +---------------+
```

The host inserts a minimum delay between two consecutive 1-byte packets in order to guarantee that the controller can keep up with the incoming data.

The SPI mode settings are:

* Clock polarity: SCK is low when idle.

* Clock phase: sample on leading edge.

* Bit order: most significant bit first.

2.7.2. I2C Interface

The I2C interface is implemented using the ATtiny45's USI.

The controller has an index variable that points to the current register in the register map. This variable is initialized to zero after reset. Each time the current register is read or written, the index variable increments up to the last register and wraps around zero after the last register.

The controller's register map is accessed by means of SLA+W transfers and SLA+R transfers.

The SLA+W transfer sets the index variable and optionally writes one or more registers:

```
       SLA+W             ACK DATA            ACK DATA            ACK DATA
   --> ----------------> <-- --------------> <-- --------------> <--       -->
  +---+-------------+---+---+---------------+---+---------------+---+--   -+---+
  | S |             | 0 | 0 |               | 0 |               | 0 | ... |P/S|
  +---+-------------+---+---+---------------+---+---------------+---+-   --+---+
        7-bit slave   |      register index     |<------------------------>|
          address    write                               data bytes
```

If the register index outranges the register map, the controller ignores the value and uses the current index.

The SLA+R transfer reads one or more registers:

```
       SLA+R             ACK DATA            ACK DATA  DATA             NACK
   --> ----------------> <-- <--------------- -->      <--------------- --> -->
  +---+-------------+---+---+---------------+---+--   -+---------------+---+---+
  | S |             | 1 | 0 |               | 0 | ... |               | 1 |P/S|
  +---+-------------+---+---+---------------+---+-  --+---------------+---+---+
        7-bit slave   |     |<------------------------------------------------>|
          address    read                       data bytes
```

Example transfers:

```
       SLA+W             NACK
   --> ----------------> <-- -->
  +---+-------------+---+---+---+
  | S |1 0 1 1 0 1 0| 0 | 1 | P |
  +---+-------------+---+---+---+
```

The controller doesn't respond.

```
       SLA+W             ACK DATA            ACK
   --> ----------------> <-- ---------------> <-- -->
  +---+-------------+---+---+---------------+---+---+
  | S |1 0 1 1 0 1 0| 0 | 0 |0 0 0 0 0 1 0 1| 0 | P |
  +---+-------------+---+---+---------------+---+---+
```

Set current register index to 5 i.e. the STATUS register.

```
       SLA+W             ACK DATA            ACK DATA            ACK
   --> ----------------> <-- ---------------> <-- ---------------> <-- -->
  +---+-------------+---+---+---------------+---+---------------+---+---+
  | S |1 0 1 1 0 1 0| 0 | 0 |0 0 0 0 0 1 1 0| 0 |0 0 0 0 0 0 0 1| 0 | P |
  +---+-------------+---+---+---------------+---+---------------+---+---+
```

Set CMD.WRE to one.

```
       SLA+R             ACK DATA            NACK
   --> ----------------> <-- <--------------- --> -->
  +---+-------------+---+---+---------------+---+---+
  | S |1 0 1 1 0 1 0| 1 | 0 |1 1 0 0 1 0 1 0| 1 | P |
  +---+-------------+---+---+---------------+---+---+
```

The host reads one register.

Unlike the SPI implementation, there are no race conditions between the I2C bus and the firmware, thanks to clock stretching. Whenever the USI demands software intervention, it stretching the clock line SCL making the I2C interface fully deterministic.

Since the USI requires a great deal of software intervention, clock stretching is virtually always the case.

Note that the I2C interface is not suitable for RPi. The BCM2708/2709 SoC has a bug in its I2C function preventing it from correctly dealing with an I2C slave that performs clock stretching. This is also the reason why the controller on the Swiss Pi board implements the SPI interface.

3. Program

A command line tool that runs in Linux and Windows complements the controller. This program supports all features of the controller and contains the detection procedure for determining the count values for the various baudrates.

The program maintains a settings file called the baud information file.

The program is currently part of the Swiss Pi software and is called swiss485. The baud information file is called swiss485bi.cfg.

```
                     .
           Hardware  .  Software
                     .
    RS485CTRL        .       swiss485
  +------------+     .     +------------+
  |            |     .     |            |
  |            |  I2C/SPI  |            |    +---------------+
  |            |<=========>|            |....| swiss485.cfg  |
  |            |     .     |            |    +---------------+
  |            |     .     |            |
  +------------+     .     +------------+
                     .
```

Note that the RS-485 Controller on the Swiss Pi uses the SPI interface, while the program supports both SPI and I2C.

```
4. Document History
--------------------

2016-10-08  Peter S'heeren, Axiris

  * First release.
```