Dr. Edgar Huckert
05/2018 V1.1

The simplest serial connection

If you need a simple connection between two computers the RS232 is still a good choice. Depending on the baud rate it allows distances between several meters and up to 100 m. In the embedded world it has however some drawbacks:

- on both sides special chips (UARTS) are needed
- the transmission level is -12V/+12V: this requires often additional chips (MAX...)

I tested if on low level MCUs (ex. Arduino, NXP LCP1768 etc.) a much simpler transmission is possible. Normally I2C or SPI is implemented on many small MCUs. These protocols require TTL (=5V) or lower levels (3.3V). 3 or 4 lines are necessary to rune these protocols. This seemed to much for me. For one-direction (typically connection between a sensor to a consumer) communication 2 lines should be enough.

The method

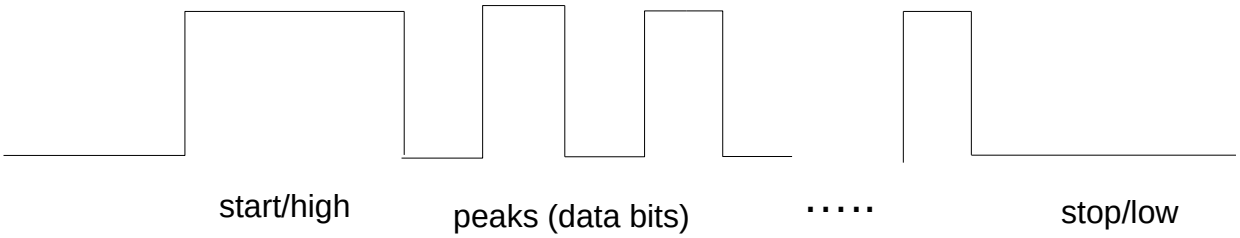
My method (certainly not new) is simple because it does not require UART chips nor libraries like the softSerial library on Arduino. Simulating a UART would require exact timing on both sides: the bits in a byte transmitted via RS232 or similar serial protocols must have exact distances in time.

The method described here should not be confused with the more complicated and more powerful *1-wire protocol*. Like with the 1-wire protocol in fact **two lines** are needed on both sides connected to a digital GPIO and to GND.

Instead of sending small **fixed length** packages - a UART byte is a small package (frame) consisting of start bit, data bits and stop bit(s) - I send **variabe length** packages. Each package starts with a start bit (high level) and ends with a stop bit (low level). Start bits and stop bits are longer then the data bits so that they can be distinguished from data bits.

In order to avoid precise timing as in UARTs I do not send 8-bits per byte but exactly **as many peak bytes as the byte value** says: for a blank (space, 0x30) I send 48 peaks, for a 'A' (0x41) I send 65 peaks. This method avoids the detection of high bits and low bits. It just requires the detection of high levels and low levels (peaks) and the transitions between them.

Here is a diagram showing this method. Note the longer Start bit (high level) and the longer stop bit (low level):



My implementation does not care for basic **data integrity.** In normal serial transmissions **parity bits** can be used to provide a basic level of integrity. You can add this here also - but I didn't do that.

The main **advantages** of my method are:

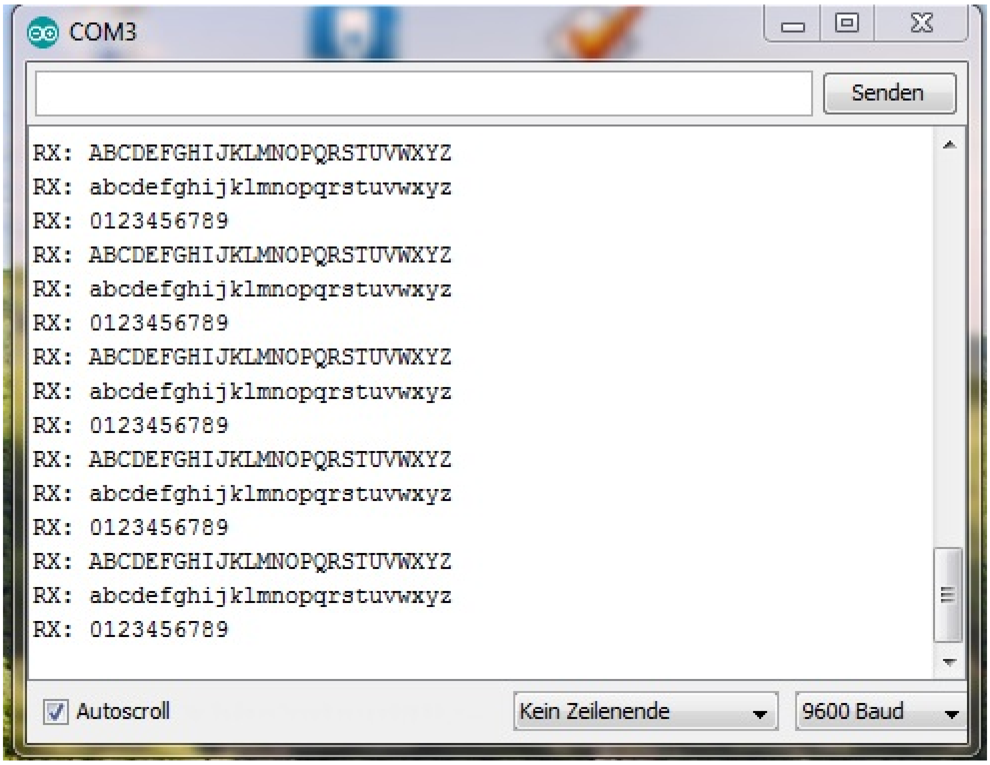
- No UART needed (cheap!)
- Only 2 lines for onedirectional commnunication need
- TTL or 3.3V levels: no level shifter for 12/-12V needed
- Very simple protocol (around 40-80 lines of code)
- Can be implemented on very small MCUs (Arduino, NXP1868) as on larger CPUs (Raspberry)
- no drivers needed

The main **drawbacks**:

- Does not allow larger distances (several meters)
- Not very performant (but sufficient for displays etc.)
- Polling - no interrupts required
- May be unsure on multitasking OS
- Should not be used for reliable communctions (this is also true for standard RS232)
- You cannot intermix this method with UART driven protocols

I have tested my method between a Rasperry 3 Zero (sender) and an Arduino Nano: the results were good. The Rasberry 3 ran under Linux - a multi-tasking and multi-user OS. These types of operating systems normally do not guarantee exact timing on the user level as hidden tasks (like flushing to disks) or other user tasks may influence the scheduling of tasks. With a real time system (like RTOS) or with a simple event loop (Arduino, basic NXP LCP1868) the method should even be more reliable.

The image below shows the reception of a string in the Arduino IDE. Note that on the receiver side the polling process should not be interrupted by longer tasks (like calling *printf()* or *writeln()*). Using a small block protocoll (sending length and data bytes) may help here as the polling can be interrupted at the end of a block for other tasks like *printf()*.



Code for a sender

The **C-Code for the sender** (here a version for Raspberry):

```
// module sendChar.c : output on GPIO7
// Must be run als root!
// Sender: Raspberry 3 Zero
```

```
// Receiver side: Arduino Nano, sketch simpleSerial
// Works with values KURZ=35 and LANG=400

#define BCM2708_PERI_BASE    0x20000000
#define GPIO_BASE            (BCM2708_PERI_BASE + 0x200000)

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>

#define PAGE_SIZE (4*1024)
#define BLOCK_SIZE (4*1024)

#define KURZ 35  // adapt this for other MCUs/CPUs
#define LANG 400 // adapt this for other MCUs/CPUs

int  mem_fd;
void *gpio_map;

// I/O access
volatile unsigned *gpio;

// GPIO setup macros. Always use INP_GPIO(x) before using OUT_GPIO(x) or
SET_GPIO_ALT(x,y)
#define INP_GPIO(g) *(gpio+((g)/10)) &= ~(7<<(((g)%10)*3))
#define OUT_GPIO(g) *(gpio+((g)/10)) |=  (1<<(((g)%10)*3))
#define SET_GPIO_ALT(g,a) *(gpio+(((g)/10))) |= (((a)<=3?(a)+4:(a)==4?
3:2)<<(((g)%10)*3))

#define GPIO_SET *(gpio+7)  // sets   bits which are 1 ignores bits which
are 0
#define GPIO_CLR *(gpio+10) // clears bits which are 1 ignores bits which
are 0

// ------------------------------------------------
// Set up a memory region to access GPIO
void setup_io()
{
   /* open /dev/mem */
   if ((mem_fd = open("/dev/mem", O_RDWR|O_SYNC) ) < 0)
   {
      printf("can't open /dev/mem \n");
      exit(-1);
   }

   /* mmap GPIO */
   gpio_map = mmap(
      NULL,             //Any adddress in our space will do
      BLOCK_SIZE,       //Map length
      PROT_READ|PROT_WRITE,// Enable reading & writting to mapped memory
      MAP_SHARED,       //Shared with other processes
      mem_fd,           //File to map
      GPIO_BASE         //Offset to GPIO peripheral
   );

   close(mem_fd); //No need to keep mem_fd open after mmap
```

```
   if (gpio_map == MAP_FAILED) {
      printf("mmap error %d\n", (int)gpio_map);//errno also set!
      exit(-1);
   }

   // Always use volatile pointer!
   gpio = (volatile unsigned *)gpio_map;
} // setup_io

// ------------------------------------------------
// Sends a start bit and then as many highs corresponding
// to the byte value, i.e. for 'A' 0x41=65 peeks
void sendByte(unsigned char by)
{
    unsigned count = by;
    // send 100 us high as start signal
    //GPIO_SET = 1 << 7;
    //^^usleep(500);
    GPIO_CLR = 1 << 7;
    usleep(LANG);
    //
    count = count & (unsigned)0xff;
    //
    for (unsigned n=0; n < count; n++)
    {
      GPIO_SET = 1 << 7;
      usleep(KURZ);
      GPIO_CLR = 1 << 7;
      usleep(KURZ);
    }
    // send low as stop signal
    GPIO_CLR = 1 << 7;
    usleep(LANG);
    //
    return;
}   // end sendByte()

// ------------------------------------------------
int main(int argc, char **argv)
{
  // Set up gpi pointer for direct register access
  setup_io();

  // Switch GPIO 7 to output mode
  printf("setting GPIO 7 for output\n");
  INP_GPIO(7); // must use INP_GPIO before we can use OUT_GPIO
  OUT_GPIO(7);
  //
  printf("start sending...\n");
  //
  for (unsigned char ch='A'; ch <='Z'; ch++)
  {
    sendByte(ch);
    usleep(LANG * 4);
  }
  sendByte('\r');
  sendByte('\n');
```

```
  usleep(LANG * 10);
  for (unsigned char ch='a'; ch <='z'; ch++)
  {
    sendByte(ch);
    usleep(LANG * 4);
  }
  sendByte('\r');
  sendByte('\n');
  usleep(LANG * 10);
  for (unsigned char ch='0'; ch <='9'; ch++)
  {
    sendByte(ch);
    usleep(LANG * 4);
  }
  sendByte('\r');
  sendByte('\n');
  usleep(LANG * 10);
  //
  printf("done...\n");
  //
  return 0;
} // END main()
```

Code for a receiver (Arduino)

The **C-Code for the receiver** (here a version for Arduino). This code is a very simple state machine. The outputs still contained in the code should be eliminated or kept as small as possible:

```
// module simpleSerial (for Arduino)
// Uses digital input D2
// E.Huckert 04-2018
// Funktioniert mit Programm sendChar.c auf Seite Raspberry3/Zero

int sensorPin   = 2;  // = D2 (digital input)
int val         = 0;
unsigned highs = 0;
unsigned lastHighs = 0;
unsigned lows  = 0;
unsigned count = 0;
int      state = 0;

#define LOWPEAKS 150 // adapt this for other computers

// -------------------------------------
void setup()
{
  Serial.begin(9600);
  Serial.println("Start...");
  highs    = 0;
  count    = 0;
  state    = 0;
}
```

```
// ----------------------------------------
void loop()
{
  unsigned char buf[2];
  unsigned char rxBuf[128];
  unsigned rxIdx = 0;
goon:
  val = digitalRead(sensorPin);
  if (state == 0)
  {
    if (val == 0)
      goto goon;
    if (val > 0)
    {
      count = 0;
      state = 1;
      highs = 1;
      lows  = 0;
      goto goon;
    }
  }   // end if state == 0
  if (state == 1)
  {
    // start of rising ramp or on high level
    if (val > 0)
      highs++;
    else
    {
      // end of high level
      if (highs > 0)
      {
        count++;
        lastHighs = highs;
        highs = 0;
      }
      lows = 1;
      state = 2;
    }
    goto goon;
  }    // end if state == 1
  if (state == 2)
  {
    if (val == 0)
    {
      // low level continues
      lows++;
      if (lows > LOWPEAKS)
      {
        // stop bit detected
        state = 0;
        if (rxIdx < (sizeof(rxBuf) - 1))
          rxBuf[rxIdx++] = count;
        // end of a line?
        if (count == '\n')
        {
          rxBuf[rxIdx] = 0;
          Serial.print("RX: ");
```

```
          Serial.print((char *)rxBuf);
          rxIdx = 0;
        }
        count = 0;
        goto goon;
      }
    }   // end  if (lows > LOWPEAKS)
    else
    {
      // high level again
      state = 1;
      highs = 1;
      lows = 0;
      goto goon;
    }
  }     // end if state == 2
  goto goon;
}   // end loop()
```

Code for a receiver (NXP LCP1768)

Here is the very similar **C-Code for the receiver** using the NXP LCP1768. Note the difference in the constant *LOWPEAKS* (1850 for LCP1768 versus 150 for Arduino):

```
// module simpleSerialRx (for NXP LCP1768)
// modelled after simpleSerial on Arduino
// Uses digital input p8
// E.Huckert 04-2018
// Funktioniert mit Programm sendChar.c auf Seite Raspberry3/Zero
// NXP LCP1768: laeuft gut wenn TeraTerm Baudrate=115200

#include "mbed.h"

int val         = 0;
unsigned highs = 0;
unsigned lastHighs = 0;
unsigned lows  = 0;
unsigned count = 0;
int      state = 0;

DigitalIn  pin8(p8);
DigitalOut led(LED1);
Serial serial(USBTX, USBRX);

#define LOWPEAKS 1850 // adapt this for other computers

// ------------------------------------
void initialize()
{
  highs    = 0;
  count    = 0;
  state    = 0;
  led      = 1;
  serial.baud(115200);
  printf("\r\nstarting...\r\n");
```

```
}

// ----------------------------------------
int main()
{
  unsigned char rxBuf[128];
  unsigned rxIdx = 0;
  unsigned loopCount = 0;
  //
  initialize();
  //
goon:
  if ((loopCount++ % 60000) == 0)
    led = ~led;
  val = pin8;  // read from pin8
  //printf("%u:%d:%u ", val, state, lows);
  //if ((loopCount % 10) == 0)
  //  printf("\r\n");
  if (state == 0)
  {
    if (val == 0)
      goto goon;
    if (val > 0)
    {
      count = 0;
      state = 1;
      highs = 1;
      lows  = 0;
      goto goon;
    }
  }   // end if state == 0
  if (state == 1)
  {
    // start of rising ramp or on high level
    if (val > 0)
      highs++;
    else
    {
      // end of high level
      if (highs > 0)
      {
        count++;
        lastHighs = highs;
        highs = 0;
      }
      lows = 1;
      state = 2;
    }
    goto goon;
  }    // end if state == 1
  if (state == 2)
  {
    if (val == 0)
    {
      // low level continues
      lows++;
      if (lows > LOWPEAKS)
      {
```

```
        // stop bit detected
        state = 0;
        if (rxIdx < (sizeof(rxBuf) - 1))
          rxBuf[rxIdx++] = count;
        // end of a line?
        if (count == '\n')
        {
          rxBuf[rxIdx] = 0;
          serial.printf("RX: ");
          serial.printf("%s\r\n", (char *)rxBuf);
          rxIdx = 0;
        }
        count = 0;
        goto goon;
      }
    }   // end  if (lows > LOWPEAKS)
    else
    {
      // high level again
      state = 1;
      highs = 1;
      lows = 0;
      goto goon;
    }
  }     // end if state == 2
  goto goon;
}   // end main()
```