CAB202 Tutorial 10
Communicating over USB Serial

Overview

Interfacing between computer systems is reliant on a consistent and explicitly defined method of communication. This communication process must abide by a specified protocol to successfully communicate between two systems (communication without explicit protocols is analogous to two people trying to communicate with two different languages). In this practical you will learn about serial communication protocols, and how you can send data to your Teensy from a computer, and vice versa. When communicating between the Teensy and a computer, we will use virtual USB serial communication through the Atmega32u4. This tutorial is worth 4% of your final mark for this subject.

USB Serial Library and Functions

The Teensy connects to the computer through a USB connection, and this will be utilised to communicate with a computer. The USB standard provides a multitude of standard device 'interfaces'. One example of these is a serial port (other common examples include keyboard, mice, flash drive, etc.). You will need to add the following line to your main **`*.c`** source file to allow access to these functions:

```
#include “usb_serial.h”
```

You will also need to ensure that you include the **`usb_serial.c`** source file in your compilation command. Navigating the code in the provided USB serial libraries (especially the `usb_serial.h` header file) will provide an overview of the functions available. Functions of interest include: `usb_init()`, `usb_configured()`, `usb_serial_get_control()`, `usb_serial_getchar()`, and `usb_serial_putchar()`. To compile with these libraries you will need to compile against multiple source files. You should use the following (extending the command from last week):

```
avr-gcc <source>.c usb_serial.c –mmcu=atmega32u4 –Os
–DF_CPU=8000000UL –std=gnu99 –I<lib_path> -L<lib_path>
-Wl,-u,vfprintf –lprintf_flt –lcab202_teensy –lm
–o <output>.o
```

USB Drivers and PuTTY

The provided USB serial libraries assume that the computer can correctly interface with the Teensy board as a USB serial device. After creating a program that calls `usb_init()` function and programming it to the Teensy, the board it should present itself as a USB serial device when next restarted. The process of correctly configuring USB Serial driver interfacing has different instructions for the different operating systems (see this page for detailed information and all of the resources - https://www.pjrc.com/teensy/usb_serial.html):

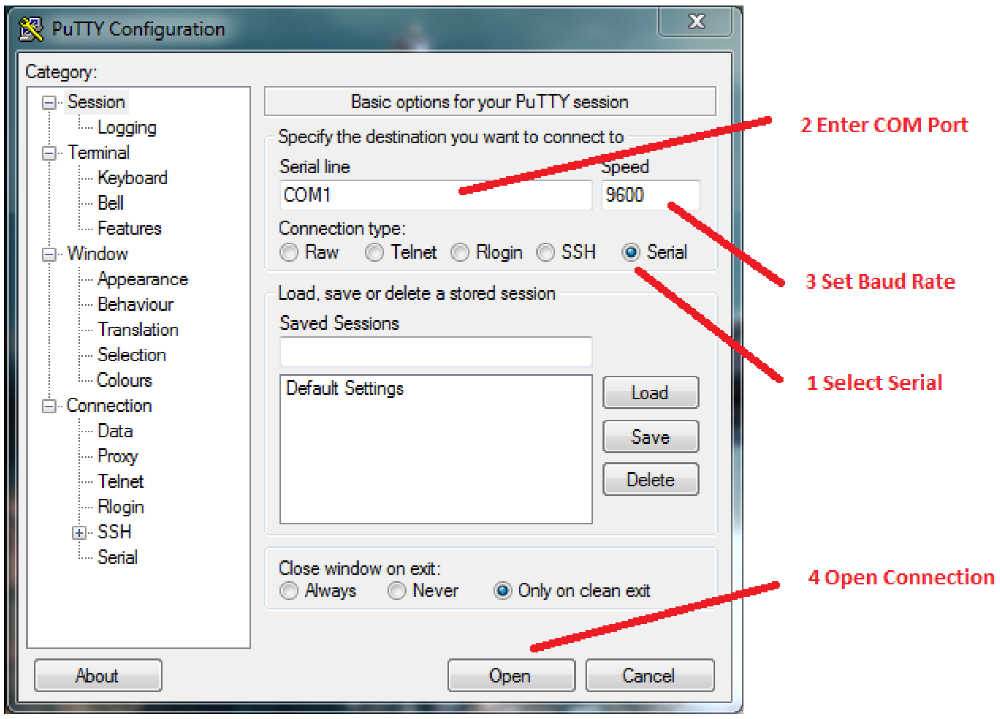
- **Windows**: the automated driver install process will fail in most cases. To overcome this, use the installer executable provided on the website. Running this installer and then reconnecting the Teensy USB cable should be sufficient to get it working.
- **Mac:** the drivers should just work. If you see an unnecessary “modem / network setup window” when you first connect, close it.
- **Linux:** as with all USB devices in Linux, you must correctly configure your UDEV rules. You should understand exactly what these are, exactly what purpose they serve, and exactly how to modify them. The UDEV rules for the Teensy USB device can be found here: https://www.pjrc.com/teensy/49-teensy.rules

To allow the Teensy to communicate with your computer, you will need to use some form of Serial Port Terminal. One of the easiest to use, and most commonly used, is called **PuTTY** (download link is here: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html). There are ports of this software available for both Ubuntu (installable straight from `apt-get`) and Mac. There are a number of other different options for running a Serial Port terminal (including “MTPuTTY” – provides easy reconnect, “KiTTY” – PuTTY but with a number of new features, “screen” – command line serial port terminal, “Arduino Serial Monitor” – very simple to use but requires installation of Aduino IDE). You may use whatever works best for you, but you must follow the correct configuration instructions regardless of the terminal software you are using.

To configure a serial connection correctly for the Teensy you **must** do the following:

- Select the serial option in connection type.
- Enter the COM port that your Teensy is currently connected to.
- Enter the correct Baud Rate – must be 9600.
- **Force local echoing** (this means when you press a key, it is echoed to the screen)**.** In PuTTY, this is found under “Terminal -> Line Discipline Options -> Local Echo”.

Below is a screenshot of everything except local echoing configured in PuTTY:


Specify the destination you want to connect to
Serial line: COM1
Speed: 9600
Connection type: Raw Telnet Rlogin SSH [✓ Serial]
Saved Sessions: Default Settings
Close window on exit: Always Never [✓ Only on clean exit]

Assessable Exercises

NOTE: You should use code you wrote from previous questions in the later questions where it applies. All questions are to be marked in your allocated tutorial session only!

1. Demonstrating basic bidirectional communication (1 mark)

Download the **`question_1.c`** source file from Blackboard. The program is complete, it waits for the computer user to initialise a serial connection, greets and asks their name, thanks them, and waits for them to press `'q'` to exit. Demonstrate this USB serial communication process with a computer and Teensy.

2. Simple debugging (1 mark)

Download the **`question_2_3_template.c`** source file from Blackboard. The program is complete but needs debugging code added to it to test all of the functionality is working correctly. Modify the template, so that the following debugging strings are printed on a connected computer (via USB serial) when the appropriate conditions are met:

- When the Teensy detects that the computer is successfully connected:

  `Debugger initialised. Debugging strings will appear below:`

- When the main loop is **first** entered (i.e. only once):

  `[DEBUG] Entering main loop...`

- When `LED0` is toggled:

  `[DEBUG] LED0 was toggled.`

- When `LED1` is toggled:

  `[DEBUG] LED1 was toggled.`

- When `SW0` (or `SW3` on new Teensy - i.e. far right) is pressed:

  `[DEBUG] Press of SW0 detected.`

- When `TIMER1` overflows:

  `[DEBUG] TIMER1 overflowed.`

Note: the `[DEBUG]` prefix is provided by the `send_debug_string()` function – you should make use of it where possible!

3. Advanced debugging (1 mark)

Debugging can often be a painful process, and can require more complicated forms of feedback to isolate the cause of issues. Take your code from question 2, and make the following extensions:

- Add timing information to the **`[DEBUG]`** prefix in **`send_debug_string()`** function. This prefix should now display **`[DBG @ XXX.XX]`**, where **`XXX.XX`** is current system time to 2 decimal places (with a width of 6 and leading zeros included).
- Use this new debugging tool to profile how long a *`show_screen()`* function call takes. You should send the following two messages before and after the *`show_screen()`* call respectively:

```
[DBG @ XXX.XX] Calling show_screen()...
[DBG @ XXX.XX] Finished show_screen().
```

- Implement the ***`enter_breakpoint()`*** function so that it halts the execution of the program, and waits for the computer user to enter a '***b***' character. This breakpoint **must** be system-wide and must remain until a '***b***' key press. The **only** things that should still be running is the serial connection and incrementing of the `ovf_count` variable (i.e. pressing the push buttons should not generate a debug message, and there should be no overflow debug message). When `enter_breakpoint()` is run, the following should be printed in the serial debug log (where **<N>** is the line number passed to the function):

```
[DBG @ XXX.XX] Entered breakpoint @ line #<N>. Press b to
continue...
```

Note: completing and demonstrating this question will award you marks for questions 1 and 2 as well.

4. Complete the lucky dip game (1 mark)

Download the **`question_4_template.c`** source file from Blackboard. The program provides skeleton code for a 'lucky dip' type game where the player enters a box number (1-9) on the computer that they would like to search for the gold ('$') – and they keep searching until they find it. The Teensy screen is only used for displaying the current state of the game (i.e. the open and closed boxes) – **all** prompting, feedback, and input from the user **must** happen in the serial console! Complete the source code for the game such that:

- The game starts with all of the closed boxes on the screen.
- The user is prompted to enter a number between 1 and 9 to select a box. The program then waits until a valid number is entered.
- If the number has already been selected, the console displays a message informing them that the box is already open, and then they are again prompted to enter a number.
- Otherwise, the box corresponding to the number entered is opened and displayed on the screen.
- If the box had the gold ('$'), your code should finish and continue onto the 'winner' code already provided in the template.

Note: the **`draw_box()`** ***function is already implemented for you – make sure you use it!***