Logic Analyzer with Teensy 4.0

Contents

Introduction

The Teensy 4.0 uses a 600MHz CPU which should make it possible to get faster data acquisition then with the Attiny44 and a Raspberry. However, I don’t expect learning as much about SPI and assembly programming this way.

Prerequisites

Hardware

- Teensy 4.0

Software

- Git
- Visual Studio Code
- PlatformIO

Description

I uploaded the code to GitHub.
To get started I first created a simple blink sketch, using the Windows Version of VSCode with PlatformIO installed.

```
#include <Arduino.h>

int led_pin = 13;

void setup()
{
 // put your setup code here, to run once:
 pinMode(led_pin, OUTPUT);
}

void loop()
{
 // put your main code here, to run repeatedly:
 digitalWrite(led_pin, HIGH);
 delay(500);
 digitalWrite(led_pin, LOW);
 delay(500);
}
```

Then I uploaded it to confirm the connection to the Teensy. It worked.

Next up I ported all to Linux since I want to use that as my main development operating system (at least at home).

First I had to clone the GitHub repository.

1. Open the VSCode command line using Ctrl-Shift-P and enter git clone.
2. Copy the URL of the repo from the GitHub website.
3. Select a location to clone to, in my case “Documents/PlatformIO/Projects”. Git will create a new directory and put all the cloned files in there.
4. Open PlatformIO Home.
5. Open a project and select the newly generated folder.

But when I compiled and uploaded it, two things happened:

- The line “#include <Arduino.h>” becomes an include error and no fix is in sight, but it still works.
- The upload failed. Even when putting the Teensy into sleep mode.

The upload failing is because of the way Linux handles new USB devices. By default, a device is generated that only has read permissions. We need to add a new rule.

To fix this, first create a file called “49-teensy.rules” on the desktop.

```
cd Desktop
sudo nano 49-teensy.rules
```

Then copy this code or get the most recent version using the link in the link section.

```
# UDEV Rules for Teensy boards, http://www.pjrc.com/teensy/
#
# The latest version of this file may be found at:
#   http://www.pjrc.com/teensy/49-teensy.rules
#
# This file must be placed at:
#
# /etc/udev/rules.d/49-teensy.rules   (preferred location)
#   or
# /lib/udev/rules.d/49-teensy.rules   (req'd on some broken systems)
#
# To install, type this command in a terminal:
#   sudo cp 49-teensy.rules /etc/udev/rules.d/49-teensy.rules
#
# After this file is installed, physically unplug and reconnect
Teensy.
#
ATTRS{idVendor}=="16c0", ATTRS{idProduct}=="04[789B]?",
ENV{ID_MM_DEVICE_IGNORE}="1", ENV{ID_MM_PORT_IGNORE}="1"
```

```
ATTRS{idVendor}=="16c0", ATTRS{idProduct}=="04[789A]?",
ENV{MTP_NO_PROBE}="1"
SUBSYSTEMS=="usb", ATTRS{idVendor}=="16c0",
ATTRS{idProduct}=="04[789ABCD]?", MODE:="0666"
KERNEL=="ttyACM*", ATTRS{idVendor}=="16c0",
ATTRS{idProduct}=="04[789B]?", MODE:="0666"
#
# If you share your linux system with other users, or just don't
like the
# idea of write permission for everybody, you can replace
MODE:="0666" with
# OWNER:="yourusername" to create the device owned by you, or with
# GROUP:="somegroupname" and mange access using standard unix
groups.
#
# ModemManager tends to interfere with USB Serial devices like
Teensy.
# Problems manifest as the Arduino Serial Monitor missing some
incoming
# data, and "Unable to open /dev/ttyACM0 for reboot request" when
# uploading.  If you experience these problems, disable or remove
# ModemManager from your system.  If you must use a modem, perhaps
# try disabling the "MM_FILTER_RULE_TTY_ACM_INTERFACE" ModemManager
# rule.  Changing ModemManager's filter policy from "strict" to
"default"
# may also help.  But if you don't use a modem, completely removing
# the troublesome ModemManager is the most effective solution.
```

Write out and exit.

Copy this file in the udev folder.

```
sudo cp 49-teensy.rules /etc/udev/rules.d/
```

Now programming works (at least for me).

Next step was to confirm a connection from the Teensy to the PC. For this I used the serial interface over USB.

The standard Arduino Serial is directly connected to the USB interface. When you want to access hardware serial connections, you have to use Serial1 to Serial6.

Then it basically works just like an Arduino.

```
#include <Arduino.h>

int led_pin = 13;
int counter = 0;

void setup()
{
  // put your setup code here, to run once:
  pinMode(led_pin, OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  // put your main code here, to run repeatedly:
  digitalWrite(led_pin, HIGH);
  delay(500);
  digitalWrite(led_pin, LOW);
  delay(500);
  counter = counter + 1;
  Serial.print("Hi, counter is at: ");
  Serial.println(counter);
}
```

I connect to it via Putty. On windows, open the device manager and look for the right serial port (COM port). On Linux I found it to be called “/dev/ttyACM0”. To check for serial devices, use this command.

```
dmesg | grep tty
```

Writing a single byte (captured logic analyzer information) can be done with the Serial.write(); command.

The basic structure of the logic analyzer code is as follows:

1. Initialize a timer with a set frequency (maybe read serial?)
2. Sample pins as fast as possible and write the most new value to a buffer (one byte buffer only, overwrite with new values)
3. Repeat from step 2.

If the timer fires:

1. Transmit the most recent buffer via USB serial.
2. Return from interrupt.

Doing so should get me pretty high with the sampling rate.

There is a bonus feature: The Teensy Serial.write(); does not wait until the transmission is complete but just writes the values into the transmission buffer.
Using a high speed serial baud rate (maybe for really high speeds use a shielded USB cable) this should get pretty high.
If I’m not mistaken the USB serial port actually uses 12Mbit/s by default so with one byte per transmission I should get somewhere close to 1Mb/s which should translate to 1Msps which again is pretty dope.
However one bottleneck could be the wait time to initialize a new transmission for every bit instead of writing an array as a block which is said to be a little faster but we’ll see.

```
#include <Arduino.h>

unsigned char buffer;

void setup()
{
  // input pins = PORT D register
  pinMode(0, INPUT);
  pinMode(1, INPUT);
  pinMode(2, INPUT);
  pinMode(3, INPUT);
  pinMode(4, INPUT);
  pinMode(5, INPUT);
  pinMode(6, INPUT);
  pinMode(7, INPUT);

  Serial.begin(9600); // uses maximum frequency for USB = 12MHz
}

void loop()
{
  buffer = digitalReadFast(0) << 0;
  buffer |= digitalReadFast(1) << 1;
  buffer |= digitalReadFast(2) << 2;
  buffer |= digitalReadFast(3) << 3;
  buffer |= digitalReadFast(4) << 4;
  buffer |= digitalReadFast(5) << 5;
  buffer |= digitalReadFast(6) << 6;
  buffer |= digitalReadFast(7) << 7;

  Serial.println(buffer, BIN);
}
```

This works, but it is slow, mainly because of two points:

- The buffer is not filled when sending (nor is it updated)
- The buffer fills slowly since each pin is read and shifted one at a time and not port wise

```
#include <Arduino.h>

IntervalTimer sampleTimer;
void sendSample();

volatile unsigned char buffer;

void setup()
{
  // input pins = PORT D register
  pinMode(0, INPUT);
  pinMode(1, INPUT);
  pinMode(2, INPUT);
  pinMode(3, INPUT);
  pinMode(4, INPUT);
  pinMode(5, INPUT);
  pinMode(6, INPUT);
  pinMode(7, INPUT);

  Serial.begin(9600); // uses maximum frequency for USB = 12MHz

  sampleTimer.begin(sendSample, 2);
}

void loop()
{
  buffer |= digitalReadFast(0) << 0;
  buffer |= digitalReadFast(1) << 1;
  buffer |= digitalReadFast(2) << 2;
  buffer |= digitalReadFast(3) << 3;
  buffer |= digitalReadFast(4) << 4;
  buffer |= digitalReadFast(5) << 5;
  buffer |= digitalReadFast(6) << 6;
  buffer |= digitalReadFast(7) << 7;
}

void sendSample()
{
  Serial.write(buffer);
```

```
  buffer = 0;
}
```

Now every 2us, so 500 000 times a second, the function is called. It writes the last known state of “buffer” via USB at 12Mbit/s over a serial connection.

Note, that I write a single byte here, so the transmission takes about 1us. This is the maximum speed I was able to reach.

Results summarized

I was able to get up to 500ksps with the Teensy 4.0.
Now only a analyser software is missing.
It would also be nice if I could set the sampling rate via serial. I’m currently working on that and putting it up on GitHub once it’s finished.

Links to Sources and more Information

- Teensy rules: https://www.pjrc.com/teensy/loader_linux.html
- GitHub repo test code: https://github.com/MxFxM/TeensyUSB_Test
- GitHub repo logic analyzer code:
- Teensy serial: https://www.pjrc.com/teensy/td_uart.html
- Teensy schematic: https://www.pjrc.com/teensy/schematic.html
- Teensy pinout: https://www.pjrc.com/teensy/pinout.html
-