Optical Based Debugging Tool, Version 1.1

By R. G. Sparber

Protected by Creative Commons.[1]

Want to jump to the punchline? Then see

https://www.youtube.com/watch?v=BColki2Hrzo

The Problem

The easiest way to debug code running on a processor is with a full-featured software development environment. It shows you the value of all variables and the path taken as the code executes.

I don't have one of those.

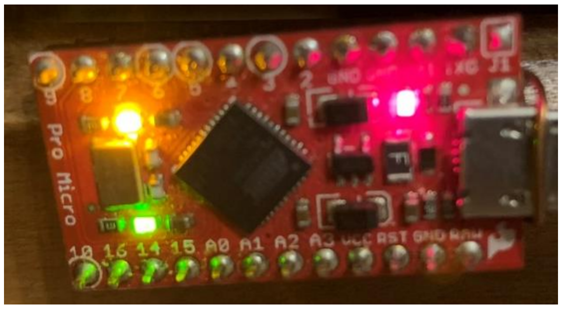
```
#ifdef DiagPrint1

Serial.print(__FUNCTION__);
Serial.print(F("(): "));
Serial.print(__LINE__);
Serial.print(F(".    TS:  "));
Serial.println(millis() - StartForTimeStampULong);
Serial.print(F(" tag "));
Serial.println( data );
#endif
```

Almost all of the time, I use print statements to tell me these same things[2]. These statements are generated with a single keystroke and then populated with any variables I want to see. The subroutine and line number are automatically supplied by the compiler. It calculates a timestamp and prints that too. Of course, this scheme depends on having a terminal emulator to receive all of these serial prints.

[1] This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.
[2] See https://rick.sparber.org/DTFA.pdf

Once in a while, I find myself in an environment where I cannot connect a terminal emulator. Then I have to get creative just to squeeze out, say, 16 different notifications. As with any diagnostic code, I must also minimize my realtime impact.

The Solution

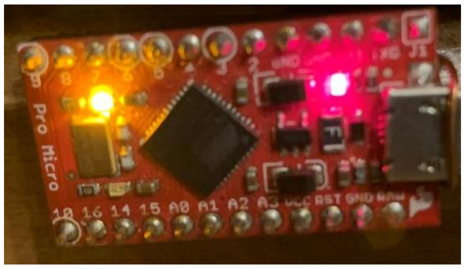


My favorite Arduino compatible is the Sparkfun Pro Micro. It has three LEDs on it. The red one indicates power, so there is no way to control it via software. But the yellow and green LEDs are up for grabs. Usually, they show the state of the serial port.

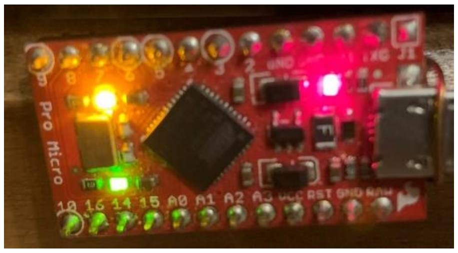
Thanks to Steven Bush, I learned how to control them from my code[3].

Two LEDs may not seem like much, but they provide what I need. Think of the yellow LED as my data and the green LED as my clock. I can flash these LEDs to convey a four-bit quantity. My old brain can remember four bits but would have trouble with eight. If you can handle it, the code can easily be changed to send a full byte.

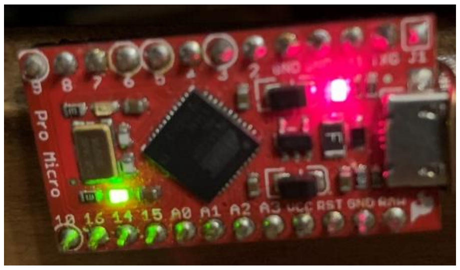
[3] See https://www.electronicsweekly.com/blogs/engineer-in-wonderland/arduino-micro-direct-access-board-leds-2017-08/



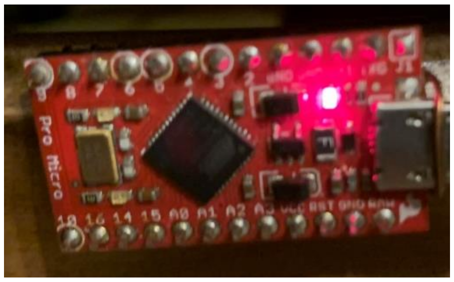
I need a start symbol before I sent my bits. I do this by lighting the data LED without the clock LED once.



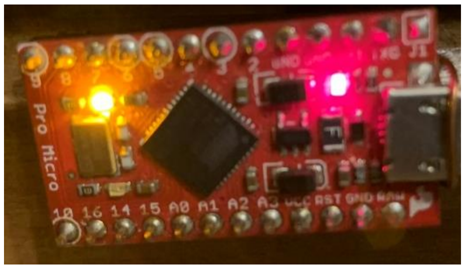
To send a logic one, I light both the clock and data LEDs.



To send a logic zero, I light just the clock LED.



The delimiter between symbols is having neither LED on. It is included at the end of start and each bit.



After the last bit has been transmitted, I append the stop symbol, which is the yellow LED flashed twice.

The entire sequence looks like this:

[start] [bit n] [bit n-1] … [bit 0] [stop]

There are two time intervals. The state active delay is the time that the LEDs are being read. The inter-state delay is the time between symbols. I found that having them equal works well.

You have a choice of subroutines:

- FlashNibble(x) to send the lower four bits of a byte
- FlashByte(x) to send a byte
- FlashUint(x) to send two bytes
- FlashUlong(x) to send four bytes

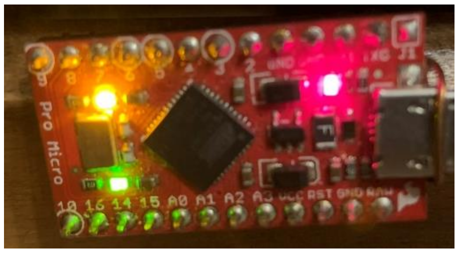
If you want to build your own Flash subroutine:

- StartSymbol()
- SendBits(DiagCode, n) for n less than 33 bits
- StopSymbol()

There is one more piece to the puzzle: minimizing realtime.

To minimize the effect on the program being debugged, I cannot have complexity in my software tool. Therefore, I have simple delays, as needed, to ensure the LEDs are on long enough to be seen.

The key to minimizing these delays is to ask the question – who sees the LEDs? Or what sees them?



If I look at these LEDs with my unassisted eye, the flash rate can't be much faster than one second per bit, or it becomes a blur. That could severely upset a realtime sensitive program.

But I own a smartphone with a built-in camera. One of the features is “slo-mo” with manual playback. It will take 240 frames per second, which means one frame every 4.2 milliseconds. Then I can view one frame at a time.

I am using the iOS app called InShOt, which lets me stop at each frame. This means that I can specify a 9-millisecond interval to show a state plus 9 milliseconds between states and will be guaranteed to see at least one frame for each state. I can, therefore, send a byte in 198 milliseconds.

I am aware that some Android phones can capture up to 1000 frames per second. That will permit the transmission of a byte in about 50 milliseconds.

The Code

If you are not running it on a Pro Micro, the logical pin names associated with LEDs may have to change.

```
int DataLED = 17; // The RX LED has a defined Arduino pin
int ClockLED = 30; // The TX LED has a defined Arduino pin
unsigned int InterStateDelayMsUint = 9;
unsigned long StateActiveIntervalMsUint = 9;
unsigned long StartForTimeStampULong = millis();

void setup() {
pinMode(DataLED, OUTPUT); // Set RX LED as an output and call it the Data
LED which is yellow
pinMode(ClockLED, OUTPUT); // Set TX LED as an output and call it the Clock
LED which is green
}
```

```
void loop(){
//test suproutine calls
FlashNibble(0xa);
//FlashByte(0b10101010);
//FlashUint(0b1010101010101010);
//FlashUlong(0b10101010101010101010101010101010);
delay (2000);
}

void FlashNibble(byte DiagCode){
StartSymbol();//single flash of data LED
SendBits(DiagCode, 4);
StopSymbol();//double flash of data LED
}

void FlashByte(byte DiagCode){
StartSymbol();//single flash of data LED
SendBits(DiagCode, 8);
StopSymbol();//double flash of data LED
}

void FlashUint(unsigned int DiagCode){
StartSymbol();//single flash of data LED
SendBits(DiagCode, 16);
StopSymbol();//double flash of data LED
}

void FlashUlong(unsigned long DiagCode){
StartSymbol();//single flash of data LED
SendBits(DiagCode, 32);
StopSymbol();//double flash of data LED
}


void StartSymbol(){
//single flash of just data LED
DataHigh();
ClockLow();
delay(StateActiveIntervalMsUint);//time to see start symbol
```

```
DataLow();
ClockLow();
delay(InterStateDelayMsUint);//time in idle to separte start symbol from MSB
}

void StopSymbol(){
StartSymbol();
StartSymbol();
}

void SendBits(unsigned long data, byte NumberOfBits){
    //flash out the lower NumberOfBits
    unsigned long BitMaskUlong = 1;//so LSB is 1 and the rest are 0
    BitMaskUlong = BitMaskUlong << NumberOfBits-1;//shift the 1 over to the
MSB position for the data
    for(byte BitCount = 0; BitCount < NumberOfBits; BitCount++){
        if((data & BitMaskUlong) >> (NumberOfBits-1)){//first I set all but
the MSB to 0 and then shift it over to the right by NumberOfBits-1 so it is the
LSB. If this evaluates to 1, I call SendOne(). Else, SendZero().
        SendOne();
        }else{
        SendZero();
        }
    data = data << 1;//shift over 1 bit position to left because I'm sending MSB
first
    }
}

void SendOne(){
DataHigh();
ClockHigh();
delay(StateActiveIntervalMsUint);
DataLow();
ClockLow();
delay(InterStateDelayMsUint);
}

void SendZero(){
DataLow();
ClockHigh();
```

```
delay(StateActiveIntervalMsUint);
DataLow();
ClockLow();
delay(InterStateDelayMsUint);
}

void DataLow(){
digitalWrite(DataLED, HIGH); // set the LED off
}

void DataHigh(){
digitalWrite(DataLED, LOW); // set the LED on
}

void ClockLow(){
digitalWrite(ClockLED, HIGH); // set the LED off
}

void ClockHigh(){
digitalWrite(ClockLED, LOW); // set the LED on
}
```

I welcome your comments and questions.

If you wish to be contacted each time I publish an article, email me with "Subscribe" in the subject line. In the body of the email, please tell me if you are interested in metalworking, software, or electronics so I can put you on the best distribution list.

If you are on a list and have had enough, email me "Unsubscribe" in the subject line.

Rick Sparber
Rgsparber.ha@gmail.com
Rick.Sparber.org