Debugging With a Serial Port

- Debugging using a serial port is an simple and reliable technique.
- It focuses on behavior *observability*.
- Its a much lower overhead way to implement a "printf()" into your code.
  - The printf() code is nearly one quarter the size of our final project!
- The presence of the "instrument" ideally would not pertubate the time behavior of your code, otherwise your code will not behave as expected.
- Don't create a *Heisenbug*!

Debugging With a Serial Port

- We will use the ATMega128's UART1.
- The UART will be covered in detail later, but for now you can just insert a little code and things will work fine.
- The output of the UART will be directed to a USB to Serial module.
- The debug messages are displayed via a serial terminal on your laptop.

Debugging With a Serial Port

- We will add some initalization code for the UART that runs outside of any critical timing areas.
- Where we wish, we write a single byte to the UART data register. This will be displayed on the laptop.
- The key to keeping the timing pertubation small is that it only takes the time to write one byte, 125nS, to an on-chip register.
- Done correctly, the change to timing is very small.
- If you need more information than a single byte can carrry, this can be done, but where you implement the multiple writes must be chosen carefully.

Debugging With a Serial Port

- The connection between your AVR board, the module, and laptop is simple.

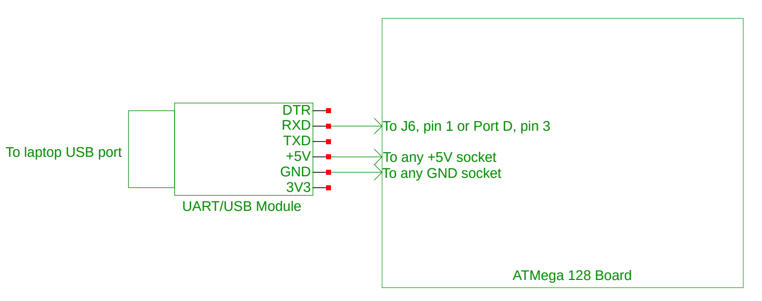

Figure 1: UART1 to USB Module Connections

Debugging With a Serial Port

- The code you need is simple. Using lab1 code a test case, let’s look at the code in parts.

```
#include <avr/io.h>
#include <util/delay.h>
#include "uart_functions.h"

#define DEBUG  1
```

- We have to include the uart functions header file so we can use functions in there.
- We set the preprocessor variable DEBUG to conditionally ”turn on” our debug code. I chosse ’1’ to indicate that debug code is enabled.

Debugging With a Serial Port

```
int8_t debounce_switch() { .......snip!.........

int main() {
  uart1_init();
  DDRB = 0xFF;  //set port B to all outputs
  while(1){     //do forever
    if(debounce_switch())
      {
      PORTB++; //increment PORTB LEDs
#if DEBUG == 1
      if(PORTB % 2){uart1_putc(’o’);} //check odd
      else         {uart1_putc(’e’);} //check even
#endif
    } //if debounce_switch()
  _delay_ms(2);  //keep in loop to debounce 24ms
  } //while
} //main
```

- Outside the while(1) loop, UART1 is initalized by `uart1_init();`
- The `#if` statement checks the value of `DEBUG` variable to conditionally compile the `if,else` and link the `uart1_putc()` statements which enables writing to the UART.

Debugging With a Serial Port

- If `DEBUG` was equal to '0', the `if,else` is still compiled but is not linked in the executable.
- Although leaving in debug code is less aesthetically pleasing, it is strongly advised that you leave it in place as written.
- If I had a dollar for everytime I've had to go back and put back debug code that I'd deleted, I'd be rich!"
- Write your debug code carefully and cleanly. It probably will be used again.

Debugging With a Serial Port

- Your Makefile will also need to be aware of the uart code. You need to add the uart object code to your makefile object file list. If the DEBUG statement is set to '1', the linker will include the debug code, else not.
- For example, if the `lab1_code.c` was renamed to `debug_w_uart.c`, you would need to change the object file list as follows:

```
OBJS                  = debug_w_uart.o uart_functions.o
```

Debugging With a Serial Port

- You can send multiple characters by using the `uart_puts()` function. This function takes a string or pointer as an argument.

```
   uart1_puts("o\n\r");
```

- Remember though, that you will take longer to execute the debug code.
- At 9600 baud, the first character takes about 125nS. Two characters take about 1mS to send. Three characters would take about 2.2mS to send.
- In 2.2mS, a 16MHz AVR can execute roughly 32,000 instructions!

Debugging With a Serial Port

- On the laptop, we need a terminal program to communicate with the UART/USB module.
- Most any terminal program will do but I'll use "gtkterm".
- Gtkterm is downloaded and installed with:

```
sudo apt-get install gtkterm
```

- Since Gtkterm "talks" with a hardware port, you will need to invoke it with sudo.

```
sudo gtkterm
```