Strange inputs when reading data over serial communication with ATTiny85

dan1994 **sept. '20 #1**

I'm trying to get 2 way serial communication working over ATTiny85.

My setup: The ATtiny85 is connected through an UNO which is tri-stated (RESET connected to GND). The connections between the UNO and ATTiny are VCC (5V), GND, TX to TX and RX to RX. I'm using ATTinyCore, and its built-in Serial object.

I'm running the following code:

```
unsigned char *buffer = Serial._rx_buffer->buffer;
int *head = &Serial._rx_buffer->head;
int *tail = &Serial._rx_buffer->tail;

void setup() {
    Serial.begin(115200);
}

void loop() {
    Serial.print(*head);
    Serial.print(",\t");
    Serial.print(*tail);
    Serial.print(",\t");
    for (size_t index = 0; index < SERIAL_BUFFER_SIZE; index++) {
        const size_t adjusted_index = (index + *tail) % SERIAL_BUFFER_SIZE;
        const unsigned int value = static_cast(buffer[adjusted_index]);
        Serial.print(value);
        Serial.print("\t");
    }
    Serial.print(",\t");
    Serial.print(Serial.available());
    Serial.print(",\t");
    Serial.print(Serial.read());
    Serial.print("\n");
    delay(5000);
}
```

As can be seen, I'm inspecting the circular buffer used internally by the serial object. The circular buffer has the following structure:

```
struct soft_ring_buffer
{
  unsigned char buffer[SERIAL_BUFFER_SIZE];
  int head;
  int tail;
};
```

The tail points to the next byte to be read, while the head points after the last byte written. Both wrap around when reaching SERIAL_BUFFER_SIZE (32 in my case).

I've run the code and received the following output:

```
14, 1,  248 248 168 173 22  87  229 206 143 184 25  215 164 0   0   0   0   (
14, 2,  248 168 173 22  87  229 206 143 184 25  215 164 0   0   0   0   0   (
14, 3,  168 173 22  87  229 206 143 184 25  215 164 0   0   0   0   0   0   (
14, 4,  173 22  87  229 206 143 184 25  215 164 0   0   0   0   0   0   0   (
14, 5,  22  87  229 206 143 184 25  215 164 0   0   0   0   0   0   0   0   (
14, 6,  87  229 206 143 184 25  215 164 0   0   0   0   0   0   0   0   0   (
14, 7,  229 206 143 184 25  215 164 0   0   0   0   0   0   0   0   0   0   (
14, 8,  206 143 184 25  215 164 0   0   0   0   0   0   0   0   0   0   0   (
14, 9,  143 184 25  215 164 0   0   0   0   0   0   0   0   0   0   0   0   (
14, 10, 184 25  215 164 0   0   0   0   0   0   0   0   0   0   0   0   0   (
14, 11, 25  215 164 0   0   0   0   0   0   0   0   0   0   0   0   0   0   (
14, 12, 215 164 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   (
14, 13, 164 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   (
```

```
14, 14, 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
14, 14, 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
17, 14, 152 67  225 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
17, 15, 67  225 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2
17, 16, 225 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   240 2
17, 17, 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   240 248 2
17, 17, 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   240 248 2
20, 17, 152 67  225 0   0   0   0   0   0   0   0   0   0   0   0   0   240 248 2
20, 18, 67  225 0   0   0   0   0   0   0   0   0   0   0   0   0   240 248 248 1
20, 19, 225 0   0   0   0   0   0   0   0   0   0   0   0   0   240 248 248 168 1
20, 20, 0   0   0   0   0   0   0   0   0   0   0   0   0   240 248 248 168 173 2
20, 20, 0   0   0   0   0   0   0   0   0   0   0   0   0   240 248 248 168 173 2
23, 20, 152 67  225 0   0   0   0   0   0   0   0   0   0   240 248 248 168 173 2
23, 21, 67  225 0   0   0   0   0   0   0   0   0   0   240 248 248 168 173 22  8
23, 22, 225 0   0   0   0   0   0   0   0   0   0   240 248 248 168 173 22  87  2
23, 23, 0   0   0   0   0   0   0   0   0   0   240 248 248 168 173 22  87  229 2
23, 23, 0   0   0   0   0   0   0   0   0   0   240 248 248 168 173 22  87  229 2
26, 23, 153 67  225 0   0   0   0   0   0   0   240 248 248 168 173 22  87  229 2
26, 24, 67  225 0   0   0   0   0   0   0   240 248 248 168 173 22  87  229 206 1
26, 25, 225 0   0   0   0   0   0   0   240 248 248 168 173 22  87  229 206 143 1
26, 26, 0   0   0   0   0   0   0   240 248 248 168 173 22  87  229 206 143 184 2
26, 26, 0   0   0   0   0   0   0   240 248 248 168 173 22  87  229 206 143 184 2
26, 26, 0   0   0   0   0   0   0   240 248 248 168 173 22  87  229 206 143 184 2
29, 26, 176 67  225 0   0   0   0   240 248 248 168 173 22  87  229 206 143 184 2
29, 27, 67  225 0   0   0   0   240 248 248 168 173 22  87  229 206 143 184 25  2
29, 28, 225 0   0   0   0   240 248 248 168 173 22  87  229 206 143 184 25  215 1
29, 29, 0   0   0   0   240 248 248 168 173 22  87  229 206 143 184 25  215 164 1
```

Note the comments I added at the end of lines where I sent input.

To sum this up I see 2 strange behaviors:

- The Serial is initialized with 14 bytes of data to be read which seem to be constant. Why is that? I'd expect it to have no data until someone actively writes to it. Might it be my serial monitor sending data when opening the channel?
- Each time I send data I get 3 bytes which don't seem to correspond to the characters I've sent, and are not always consistent. From this example I got:
- '0' -> 152, 67, 225
- '1' -> 153, 67, 225
- 'a' -> 176, 67, 225

Let's assume I should just ignore the last 2 constant numbers. Even then, the first number is not an ascii representation, or even an offset one. How am I supposed to receive data reliably?

---

dan1994 **sept. '20 #2**

I noticed another (probably) important detail: The example output I showed above was captured through Visual Studio Code. I believe it uses the same serial monitor, but just to check I've tried to run it through the Arduino IDE and the output looked different.

Only 2 bytes were in the buffer (248, 248), and when writing, only 2 bytes were added at a time (e.g. '0' -> 152 225).

This is just baffling...

---

hzrnbgy **sept. '20 #3**

> TX to TX and RX to RX

Normally, in serial connections, TX from one device goes to RX of another device

---

dan1994 **sept. '20 #4**

Yes, but in this case, the UNO is in reset so it's just passing the signals through. I was able to compile a sketch that just prints and it works with TX to TX. This is also the case here, as can be seen from the output I'm receiving.

groundFungus Shannon **sept. '20 #5**

I believe that the serial implementation in the tiny core is actually software serial and as such can not be expected to work, reliably, at more that 38400 baud.

dan1994 **sept. '20 #6**

Thank you for your advice. I switched to 9600 and now get correct ASCII values.

My test still shows more than one byte written at a time (3 for VSCode and 2 for the Arduino IDE), but the first byte is the correct ASCII value, and the rest are constant (13, 10 for VSCode; 10 for Arduino IDE).

Also, readString() works correctly, so maybe it's responsible for dealing with the extra values.

Also, when using readString() I see that the first string printed is "►►TestingOpen" in VScode which might explain the "junk bytes" I see when the connection is opened.

groundFungus Shannon **sept. '20 #7**

> My test still shows more than one byte written at a time (3 for VSCode and 2 for the Arduino IDE), but the first byte is the correct ASCII value, and the rest are constant (13, 10 for VSCode; 10 for Arduino IDE).

The **ASCII code** for line feed is 10 (0x0a) and the code for carriage return is 13 (ox0d). That is what you are seeing. I do not know VS but if you want to prevent those from being sent by serial monitor, turn off line endings. |500x285

no line ending.jpg|890x508