UART implementation for PRU-ICSS

Adam Balawender
Faculty of Electronics
Wroclaw University of Science and Technology

Instructor:
Witold Paluszynski, Ph.D.
Class:
Intermediate Project

Abstract

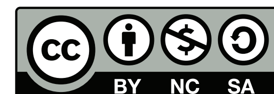
The goal of this project is to use the PRU-ICSS to create a fast (> 1Mbps) peripheral serial communication device, entirely in software. For the sake of testability, UART has been chosen as the implemented device. Although PRU-ICSS may include a hardware UART peripheral, it is not used in this project.

Hard real-time capabilities of the PRU were combined with the extended I/O hardware and a carefully optimized critical section. As a result, the new peripheral has a theoretical maximum speed limit of 4.347MBaud. However, due to clock divisor limitations, the maximum configurable speed is 4.168MBaud.

The new peripheral has been tested to work reliably up to that speed, which is beyond the reach of the FT232RL, a popular UART interface chip. It may be used as a base for implementing more complex protocols, such as CAN. This document contains the design considerations and the testing setup.

2018 January

This work is licensed under a Creative Commons "Attribution-NonCommercial-ShareAlike 4.0 International" license.



1 Introduction

The goal of this project is to use PRU-ICSS to create a fast (> 1Mbps) peripheral serial communication device, entirely in software. PRU is an interesting platform, but there are not many projects involving it. Perhaps this work may improve the situation and become a building block for the next, more complex one.

UART has been chosen mostly because of its low complexity, allowing to focus on platform-related problems, instead of the protocol caveats. UART can also be easily tested and verified by communicating it with other commonly available, inexpensive UART devices.

1.1 UART

Universal Asynchronous Receiver-Transmitter is a hardware device for asynchronous serial communication. Each character is framed as a start bit, 5-9 data bits, optionally a parity bit and one or more stop bits. Typically, the idle state is high and the start bit is low, as visible in the figure 1.

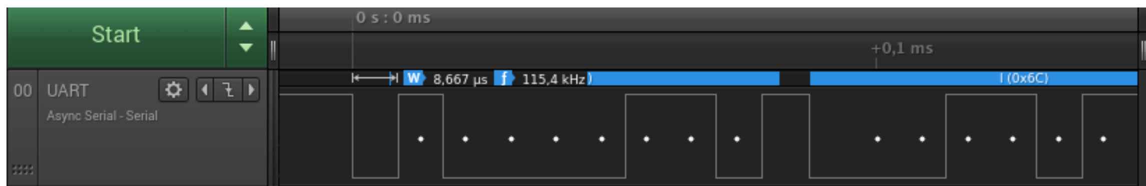


Figure 1: Typical UART configuration: 115200 8N1 (115200 Baud, 8-bit of data, no parity, 1 stop bit)

Communicating UARTs do not share timing system apart from the communication signal, so both devices need to have the same configuration to be able to synchronize and communicate.

1.2 PRU-ICSS

The Programmable Real-Time Unit and Industrial Communication Subsystem (PRU-ICSS) consists of two 32-bit RISC cores (PRUs).

The PRUs can be used to implement custom peripheral interfaces, especially in situations where real-time responses are crucial. The PRU cores can be programmed with a small, deterministic instruction set, and can operate independently or in coordination with each other, as well as in coordination with the device-level host CPU[2].

1.2.1 Hardware details

Texas Instruments AM335x chip, used to develop this project, is the second-generation PRU (PRUSSv2) with the following specification[1]:

- 200MHz clock, no pipeline, executing each instruction in a single clock cycle (5ns),
- 8KB data and 8KB instruction memories per core, 12KB shared RAM,
- enhanced GPIO with the ability of operation in serial or parallel modes,
- scratchpad shared by the PRU cores,
- multiplier with optional accumulation unit (MAC),
- internal peripherals (UART, eCAP, MII_RT, MDIO, and IEP),
- interrupt controller with the possibility of handling and posting events to the host CPU.

1.2.2 Development environment

Recently, a C/C++ cross-compiler has been developed for the PRUs. The compiler supports C99 and C++03 standards[3]. The compiler, as well as GCC toolchain are now a part of the *TI processor SDK*[4]. Example code, as well as setup and compilation instructions can be found in the *pru-software-support-package*[5]. Most of the software provided by TI is licensed as TSPA and is publicly available to use for free.

Drivers to load firmware to the PRUs and control execution are included in the Linux kernel (4.4.91-ti). Also, a message queue implementation exists to share data between PRUs and the host CPU[6].

2 Design considerations

2.1 Using shift mode for I/O

The PRU's extended GPIO modes include shift-in and shift-out. This means that only one pin is used externally to read or output signals, but internally they are visible as 16-bits registers. This is not that important for the transmitter module, but due to oversampling in the receiver (see below), the shift-in mode is crucial in achieving high speed and reliability.

2.2 Using separate cores for transmitting and receiving

Shift modes only use the first pin of input or output bank, accordingly. Also, PRUs have separate pins for input and output. However, due to the external pin multiplexer's limitation, the first pin of both banks of a given PRU is the same physical pin, so only one of those will be routed out. Hence the decision to use separate cores.

2.3 Using inverted signals

The PRU's shift-in input mode has a counter that posts an event every 16 input clock cycles after a 1 is read. The UART's standard start bit is 0, so to be able to use the built-in 16-counter, all signals used in this projects are inverted. FT232RL chip can also be programmed to work with inverted signals.

2.4 Oversampling in the receiver

Due to clock jitter, the receiver needs to read data with a higher frequency. Otherwise, it may happen that some bits will be skipped. Again, due to the 16-counter and 10-bit size of a data unit, this implementation will use 8x oversampling, which means that to receive 1 unit of data, the receiver reads 16 bits, 5 times. Then, it splits every received 16 bits in two groups and does a majority vote. If there are 4 or more **1**s read among the 8 bit, it understands it as a **1**.

2.5 Clock divisor constraints

Shift modes are triggered by the pulses of the internal clock going through two divisors. Each divisor can be set between 1 and 16 with the step of 0.5. This significantly limits the number of available frequencies.

3 Results

3.1 Testing setup

The testing setup used consists of:

1. BeagleBone Black - a development board based on TI AM335x chip with PRU-ICSS,
2. Saleae Logic 8 - 8-channel logic analyzer capable of 24MHz capture and protocol analysis,
3. FTDI FT232RL - USB to UART interface capable of working with inverted signals, up to 3MBaud.

The BeagleBone Black was configured to connect necessary physical pins to PRUs.

Firstly, the input pin of the receiver was connected to the TX pin of the USB-UART interface, so that the receiver can be tested. The Logic analyzer was connected go *spy* on the transmitted signal to verify the correctness of timing. Note that this approach only let the receiver to be tested up to 3MBaud mode, due to the USB-UART limitation.

Then, the input pin of the receiver was connected to the output pin of the transmitter, instead of the USB-UART module, forming a loop. This way, the implemented transmitter

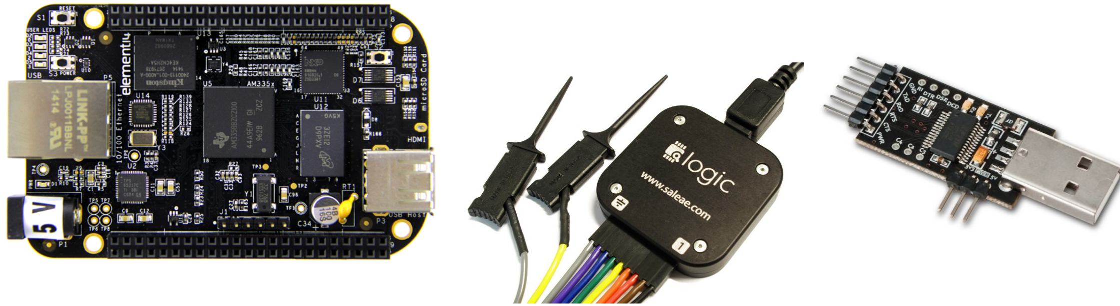


Figure 2: Devices used in the project: BeagleBone Black[a], Saleae Logic 8[b], USB↔UART[c].

[a]https://techimperial.in/wp-content/uploads/2017/02/BeagleBone-Black-Rev.C_02.jpg
[b]https://www.openlighting.org/wp-content/uploads/2013/12/Saleae-logic.jpg
[c]http://msx-elektronika.pl/pl/wp-content/uploads/2013/10/USB-UART.jpg

module was tested up to 4.168MBaud speed, the maximum. The test was asserted with the Logic analyzer.

3.2 Possible improvements

3.2.1 Timing constraints in the receiver

Every time 16 bits are read (2 bits received), majority voting is performed. Currently, after optimizations using lookup tables for voting, the voting and saving the result take 46 cycles. When running at 4.168MBaud, 2 bits are received every 48 cycles, so there is not much too spare. Perhaps, with further optimizations, a transmitter could be implemented to bit-bang bits while the receiver waits for the next 16-counter event.

3.2.2 Jitter sensitivity

Implementing the receiver to work with transmissions of exacly one byte (10 bits due to UART start/stop bits) allows to easily synchronize with every start bit, and therefore sustain a $\pm 5\%$ jitter. However, in case of a transmission of consecutive bytes, the receiver may not have time to reset the 16-counter event and ensure synchronization. This introduces sensitivity of $\pm 0.006\%$ with transmissions of 64 consecutive bytes (calculation: the receiver may be off by 4 bits maximum to still be able to recover the data). Receiver's code will need to be improved if the user requires higher tolerances for jitter in log messages.

3.3 Conclusions

Despite several challenges, such as the lacking documentation of the PRUs, the project was successful. The critical path in receiver's code only needed 46 clock cycles after opti-

mizations, and at 4.168MBaud transmission, the critical path was executed every 48 clock cycles, which further proofs the hard real-time capabilities of the PRU.

Testing environment chosen turned out to be optimal. Especially helpful was the logic analyzer, with which the timing of the signals could be easily verified. Perhaps, the debugger could be helpful, but at the point of writing this report, the usage of the available debugger for the PRU is not straightforward.

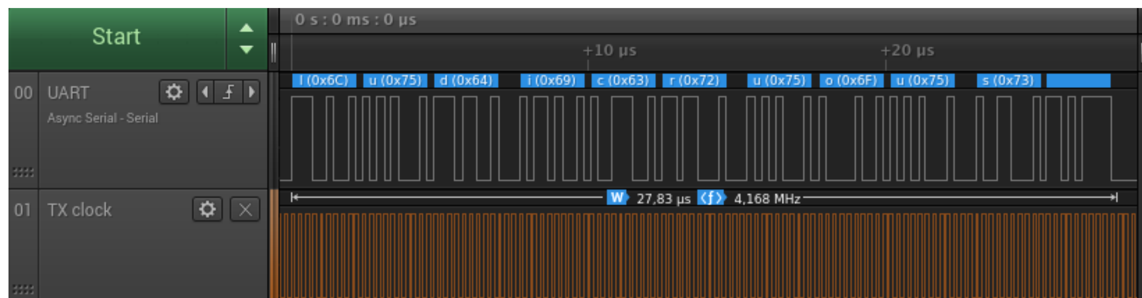


Figure 3: Successful transmission at 4.168MBaud

3.4 Public code repository

The developed code is available under MIT license: `https://github.com/phaezah7/UART`.

References

[1] Texas Instruments. AM335x and AMIC110 Sitara processors Technical Reference Manual. `http://www.ti.com/lit/ug/spruh73p/spruh73p.pdf`.

[2] Texas Instruments. AM335x PRU-ICSS Reference Guide. `https://elinux.org/images/d/da/Am335xPruReferenceGuide.pdf`.

[3] Texas Instruments. PRU optimizing C/C++ compiler. `http://www.ti.com/lit/ug/spruhv7b/spruhv7b.pdf`.

[4] Texas Instruments. PRU SDK Linux installer. `http://processors.wiki.ti.com/index.php/Processor_SDK_Linux_Installer`.

[5] Texas Instruments. PRU software support package. `http://www.ti.com/tool/PRU-SWPKG`.

[6] Texas Instruments. RPMsg quick start guide. `http://processors.wiki.ti.com/index.php/RPMsg_Quick_Start_Guide`.