CS473: Avalon slave laboratory
Calculator through custom UART controller

Olivér Facklam

November 29, 2020

Contents

1 Problem statement

The goal of this project is to create a simple text-based calculator supporting additions, using a custom UART controller as input/output mechanism.

The UART controller must be designed as an Avalon slave peripheral, allowing to integrate it onto the FPGA. It must be able to communicate with an external device through the UART protocol on a pair of TX/RX pins. The controller should also support configurable baud rate and parity settings.

The "calculator" application must receive an arithmetic expression (consisting of a single addition), parse it, and send the result back to the external device's terminal for display. All this communication must happen through the custom UART controller.

2 System overview

The overall system consists of 3 components: the NIOS-II processor, an on-chip memory, and our custom UART controller. The latter is connected to the host computer through a pair of TX/RX pins and a UART-to-USB converter. The overview of the system is shown in figure 1.

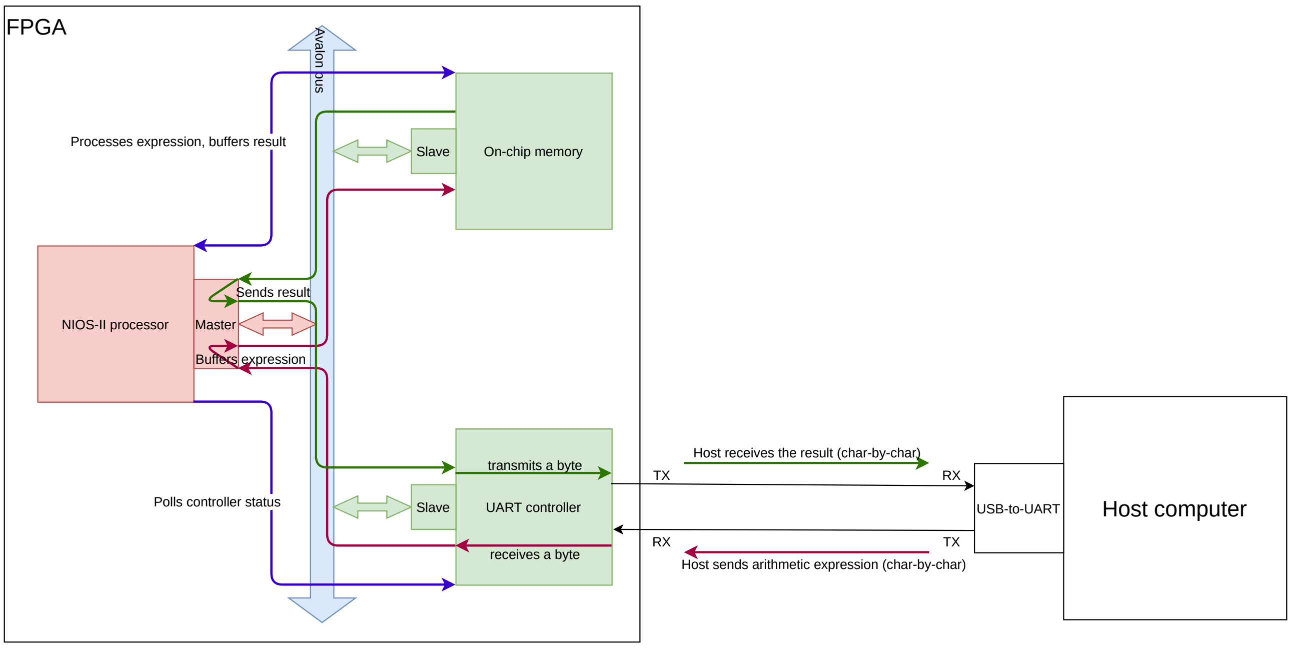

Figure 1: Overview of system architecture and operation

The standard operation of the application is the following:

1. The user types an arithmetic expression on the host terminal (used as an input device to the calculator).
2. The host computer sends the sequence of bytes through the UART protocol to the controller on the FPGA.
3. The UART controller receives these characters one-by-one, updating its status registers accordingly.
4. The NIOS-II processor polls the controller to determine when new data has arrived, and buffers the received expression into memory.
5. The processor parses the expression and prepares the response string, which is buffered into memory waiting for transmission.
6. The processor polls the controller to determine when it is ready to transmit the next character, and sends the bytes one-by-one.
7. The controller transmits each byte through the UART protocol to the host computer.
8. The response is received by the host and displayed on the terminal (the output device of the calculator).

3 UART controller design

In this section, we will describe the design and implementation of our custom UART controller. The goal is to create an Avalon memory-mapped slave peripheral, supporting configurable baud rate and parity, pollable status signals, and separate TX and RX buffers. The controller will then communicate using the UART protocol through a conduit consisting of a pair of TX/RX pins.

Some additional high-level design choices were made. The only supported data modes are:

- 8-bit data without parity and 1 stop bit (8N1 in minicom)
- 7-bit data with even parity and 1 stop bit (7E1)
- 7-bit data with odd parity and 1 stop bit (7O1).

In addition, a double-buffering system was chosen. This means that the controller doesn't have an internal FIFO; however it can store the next byte to transmit and the last byte received, while an active transmission and reception are going on.

3.1 Programmable interface

The physical interface of the peripheral is shown in figure 2. The controller has a clock and reset input, a memory-mapped slave interface, and a conduit for TX/RX.

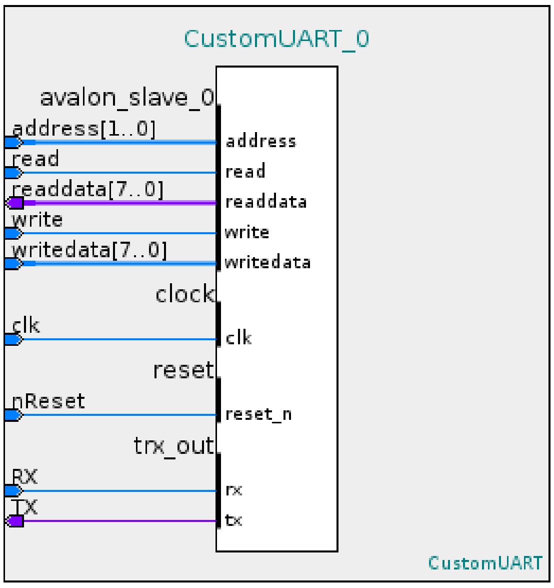

Figure 2: Interface of the UART controller

In table 1, you can find the controller's register map. We can see that the combined settings of clock divider (ranging from 1 to 128) and baud interval (ranging from 4 to 254 ticks/bit) allow – assuming a 50 MHz primary clock – baud rates between 1.54 kBaud and 12.5 MBaud.

Address	Register	Bits	Description
0x00	CTRL A	0	**RX available** (read-only): set when a new received byte is available.
		1	**TX ready** (read-only): set when the controller is ready to accept the next byte for transmission.
		2	**parity enable**: 1 to enable the parity bit (7-bit data + 1 parity bit), 0 to disable (8-bit data).
		3	**parity odd**: 1 to use odd parity, 0 to use even parity (ignored if parity is disabled).
		4 - 6	**clock divider**: $\log_2$ of the clock divider value, i.e. values between 0-7 are mapped to dividers 1-128. Transmission and reception logic (including the baud rate setting) are with respect to the slow clock.
		7	unused, always returns 0.
0x01	CTRL B	0-7	**baud rate**: represents the baud interval in terms of slow clock ticks per bit. Only values ≥ 4 are accepted. Since this setting must be even, the LSB is always 0.
0x02	RX DATA	0-7	**RX data** (read-only): last correctly received byte. Reading from this register also clears the *RXavailable* flag
0x03	TX DATA	0-7	**TX data**: next byte to transmit. Writing to this register also clears the *TXready* flag.

Table 1: Register map of the UART controller

3.2 Internal architecture

3.3 TX design

3.4 RX design

3.5 Timing of operations

3.6 Implementation details

4 Calculator software

4.1 UART driver

4.2 Main program

5 Operation & results

6 Conclusion