Homework 1: UART Arithmetic Logic Unit

Eliah Reeves

1 Introduction

In this assignment I created an Arithmetic Logic Unit (ALU) that can perform a variety of operations on integers. The ALU supports the following operations:

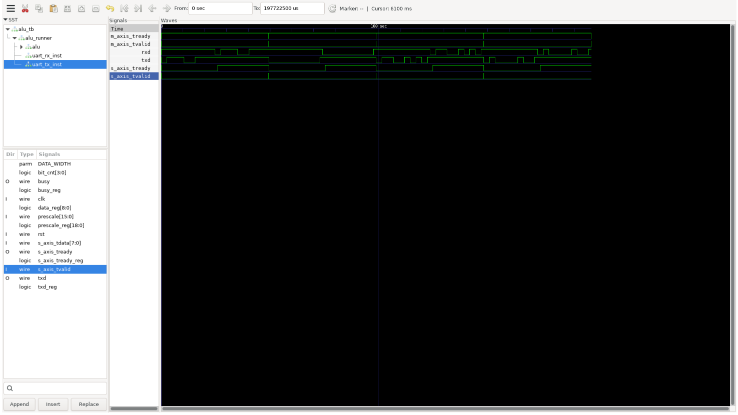
- Echo back input.
- Addition on a list of 32-bit integers.
- Multiplication on a list of 32-bit signed integers.
- Division on a pair of 32-bit signed integers.

The ALU is designed to be implemented on an iCEBreaker FPGA board and communicates with a host computer via a UART interface. This project is implemented in SystemVerilog.

2 Procedure

2.1 UART Echo

The first step I took approaching this problem was to implement a UART echo module. For this part I simply connected the RX and TX pine of Alex Forencish's UART modules. Output waveform of SystemVerilog Simulation:

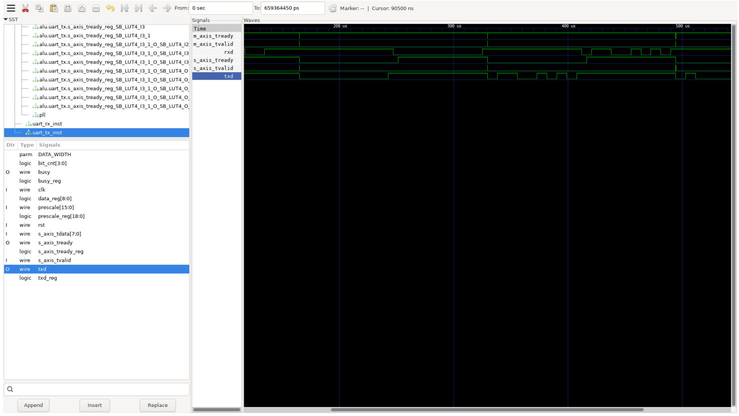


From: 0 sec To: 197722500 us Marker: -- | Cursor: 6100 ms

Dir	Type	Signals
	parm	DATA_WIDTH
	logic	bit_cnt[3:0]
O	wire	busy
	logic	busy_reg
I	wire	clk
	logic	data_reg[8:0]
I	wire	prescale[15:0]
	logic	prescale_reg[18:0]
I	wire	rst
I	wire	s_axis_tdata[7:0]
O	wire	s_axis_tready
	logic	s_axis_tready_reg
I	wire	s_axis_tvalid
O	wire	txd
	logic	txd_reg

Figure 1: UART echo simulation

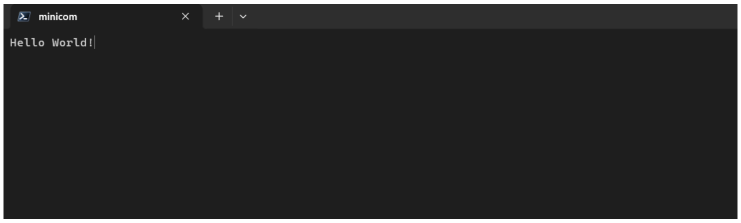
After the success of my initial simulation I ran the same test useing gate level synthesis:


From: 0 sec
To: 659364450 ps
Marker: -- | Cursor: 90500 ns
SST
alu.uart_tx.s_axis_tready_reg_SB_LUT4_I3
alu.uart_tx.s_axis_tready_reg_SB_LUT4_I3_1
alu.uart_tx.s_axis_tready_reg_SB_LUT4_I3_1_O_SB_LUT4_I2
alu.uart_tx.s_axis_tready_reg_SB_LUT4_I3_1_O_SB_LUT4_I3
alu.uart_tx.s_axis_tready_reg_SB_LUT4_I3_1_O_SB_LUT4_I3
alu.uart_tx.s_axis_tready_reg_SB_LUT4_I3_1_O_SB_LUT4_O
alu.uart_tx.s_axis_tready_reg_SB_LUT4_I3_1_O_SB_LUT4_O
alu.uart_tx.s_axis_tready_reg_SB_LUT4_I3_1_O_SB_LUT4_O
alu.uart_tx.s_axis_tready_reg_SB_LUT4_I3_1_O_SB_LUT4_O
alu.uart_tx.s_axis_tready_reg_SB_LUT4_I3_1_O_SB_LUT4_O
pll
uart_rx_inst
uart_tx_inst
Signals
Time
m_axis_tready
m_axis_tvalid
rxd
s_axis_tready
s_axis_tvalid
txd
Waves
200 us
300 us
400 us
500 us
Append
Insert
Replace

Dir	Type	Signals
	parm	DATA_WIDTH
	logic	bit_cnt[3:0]
O	wire	busy
	logic	busy_reg
I	wire	clk
	logic	data_reg[8:0]
I	wire	prescale[15:0]
	logic	prescale_reg[18:0]
I	wire	rst
I	wire	s_axis_tdata[7:0]
O	wire	s_axis_tready
	logic	s_axis_tready_reg
I	wire	s_axis_tvalid
O	wire	txd
	logic	txd_reg

Figure 2: UART echo gate level synthesis simulation

Finally the last step was to test the UART echo on the FPGA board. I used a python script and minicom to test.



```
Hello World!
```

Figure 3: UART echo in minicom

2.2 Sending requests

The next step in creating a functional ALU was to create a way to test it. To do this I used the serial package to send a request.

```
Starting Test
```

Figure 4: Request being sent

The instruction follows the following format:

Byte	Field	Description
0	Op code 1	Specifies operation
1	Reserved	currently unused
2	Length LSB	LSB of command length
3	Length MSB	MSB of command length
4-(4-Lenght - 1)	Data	Command payload

Table 1: Request format

The ALU supports the following opcodes:

Code	Description
0xEC	Echos back all data.
0xAD	Adds a list of signed or unsigned 32-bit integers.
0xAF	Multiplies a list of signed 32-bit integers
0xA6	Divides two signed 32-bit integers

Table 2: Op codes

2.3 Implementation

I Implemented the ALU using a state machine. Theses are the possible states:

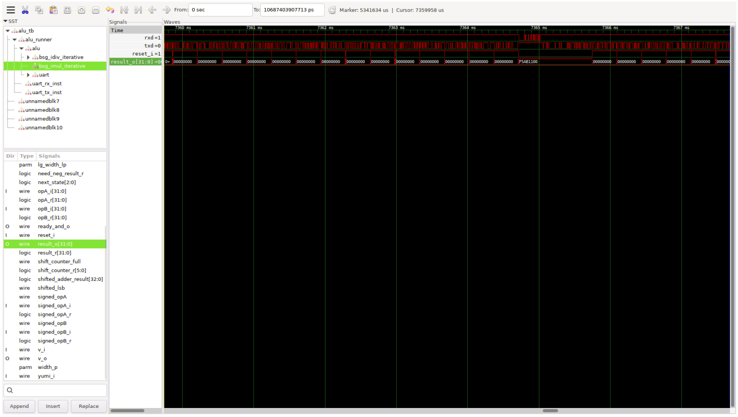
- GET_OP_CODE This stage checks for a valied op code and saves the operation that much be completed. Goes to GET_RESERVED.
- GET_RESERVED This stage doesn't do anything and transitions to GET_LENGTH_LSB. Useful for ignoring a byte.
- GET_LENGTH_LSB Saves the first byte of the length and goes to GET_LENGTH_MSB.
- GET_LENGHT_MSB. Saves the rest of the lenghth and goes to GET_INITIAL_VALUE or ECHO.
- GET_INITIAL_VALUE Saves the first integer into the accumulator. Transitions to GET_OPERAND.
- GET_OPERAND Gets next integer and transitions to the operator being performed.
- ADD Adds operand to accumulator. Transitions to GET_OPERAND or TRANSMIT.
- MUL Multiplies operand and accumulator. Transitions to GET_OPERAND or TRANSMIT.

- DIV Divides accumulator by operand Transitions to TRANSMIT.
- TRANSMIT Transmits accumulator via UART and transitions to GET_OP_CODE.
- ECHO Echos all data received for length number of bytes. Transitions to GET_OP_CODE when complete.

One challenge I encounter while writing this part of the assignment was that the multiplier never sent ready. I worked around this be resetting the multiplier before using it. This worked, but there is liekly a more elegant way.

3 Testing

In order to test my implementation I created a test bench that runs 1000 tests of each operation each test has a random number of inputs with a random value for each input. I also tested the board after it was programmed using minicom and a python script. Simulation waveform:



From: 0 sec To: 10687403907713 ps Marker: 5341634 us | Cursor: 7359958 us

```
rxd =1
txd =0
reset_i =1
result_o[31:0] =0
```

Dir	Type	Signals
	parm	lg_width_lp
	logic	need_neg_result_r
	logic	next_state[2:0]
I	wire	opA_i[31:0]
	logic	opA_r[31:0]
I	wire	opB_i[31:0]
	logic	opB_r[31:0]
O	wire	ready_and_o
I	wire	reset_i
O	wire	result_o[31:0]
	logic	result_r[31:0]
	wire	shift_counter_full
	logic	shift_counter_r[5:0]
	logic	shifted_adder_result[32:0]
	wire	shifted_lsb
	wire	signed_opA
I	wire	signed_opA_i
	logic	signed_opA_r
	wire	signed_opB
I	wire	signed_opB_i
	logic	signed_opB_r
I	wire	v_i
O	wire	v_o
	parm	width_p
I	wire	yumi_i

```
0+ 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 F5AB1100 00000000 00000000 00000000 00000000 00000000 000000
```

Figure 5: ALU testing multiply

The test benches were one of the harder parts to write for me. It seemed like the script wasn't waiting correctly, and I had to move edges around a lot to get to send and receive function to work.