ELEC6234 Embedded Processor Synthesis

SystemVerilog Design of an Embedded Processor: bit-serial processor

Introduction

This exercise is done individually and the assessment is:

- By formal report describing the final design, its development, implementation and testing.
- By a laboratory demonstration of the final design on an Altera FPGA development system

Specification

The objective of this exercise is to design a single-bit processor capable of adding, subtracting and multiplying 8-bit numbers.

The design should be as small as possible in terms of FPGA resources but sufficient to implement the affine transformation algorithm described below. The size cost function of the design is defined as follows:

Cost = number of Logic Elements used +30 x Kbits of RAM used

Each Logic Element has a flip-flop hence flip-flops are included in the above cost figure. The cost figure should be calculated for Altera Cyclone IV EP4CE115 and should be as low as possible. Although Altera Cyclone IV has 266 18x18 bit embedded hardware multipliers, do not use embedded multipliers in your implementation. Bit-serial multiplication should be carried by means of the sequential shift-and-add algorithm ex explained below. To demonstrate the cost figure of your design show in your report Altera Quartus synthesis statistics for Cyclone IV EP4CE115.

To facilitate lab demonstrations and the cost figure calculation, structure your design as shown in Figure 2 below. Calculate the cost figure only for the bitSerialCPU module which you develop.

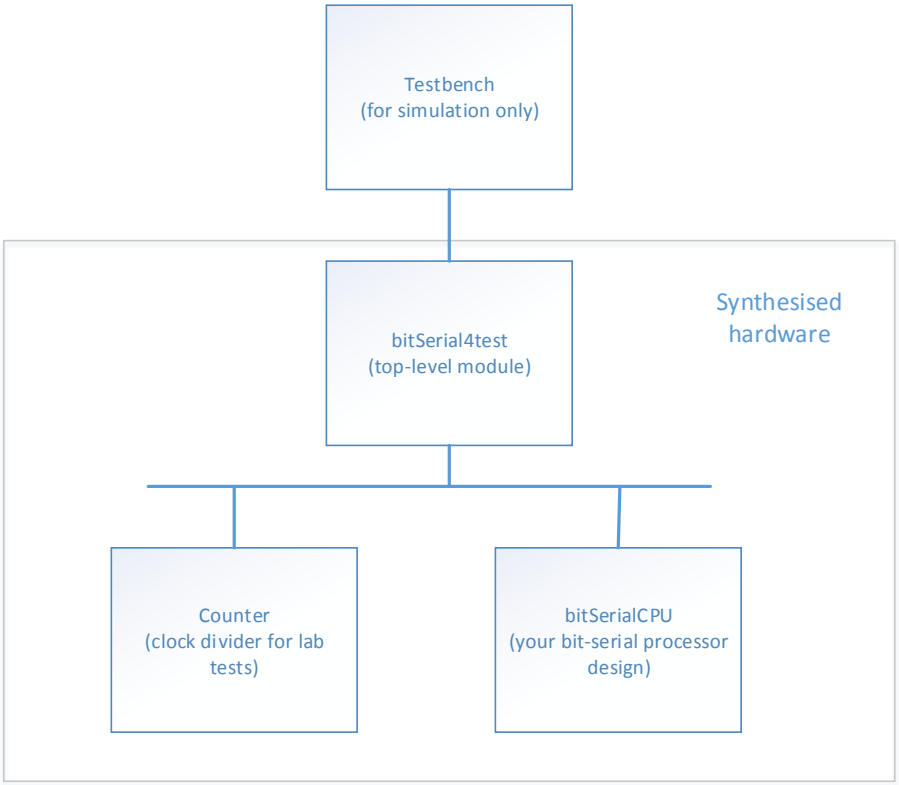

Figure 1. Synthesised design structure.

Use the following code for the top-level module bitSerial.sv and the clock divider counter.sv. The purpose of the clock divider is to eliminate bouncing effects of the mechanical switches which are used to input data as outlined by the pseudocode below.

File bitSerial.sv:

```
// synthesise to run on Altera DE0 for testing and demo
module bitSerial4test(
  input logic fastclk,  // 50MHz Altera DE0 clock
  input logic [9:0] SW, // Switches SW0..SW9
  output logic [7:0] LED); // LEDs

  logic clk; // slow clock, about 10Hz

  counter c (.fastclk(fastclk),.clk(clk)); // slow clk from counter

  // to obtain the cost figure, synthesise your design without the counter
  // and the picoMIPS4test module using Cyclone IV E as target
  // and make a note of the synthesis statistics
  bitSerialCPU myDesign (.clk(clk), .SW(SW),.LED(LED));

endmodule
```

File counter.sv:

```
// counter for slow clock
module counter #(parameter n = 24) //clock divides by 2^n, adjust n if
necessary
```

```
  (input logic fastclk, output logic clk);

logic [n-1:0] count;

always_ff @(posedge fastclk)
    count <= count + 1;

assign clk = count[n-1]; // slow clock

endmodule
```

ELEC6234 lecture slides will describe the bit-serial architecture concept and will provide SystemVerilog coding suggestions. An additional functionality to input 8-bit data and to output 8-bit results will be required as described below. Design your own instruction set and your own architecture. The architecture must be characterised by the presence of a single-bit Arithmetic-Logic Unit in the data processing path.

Bit-serial addition and subtraction

The arithmetic-logic unit of your processor must process one bit at a time. Block diagrams showing how multi-bit addition and multiplication should be performed are shown below.

Bit-serial addition

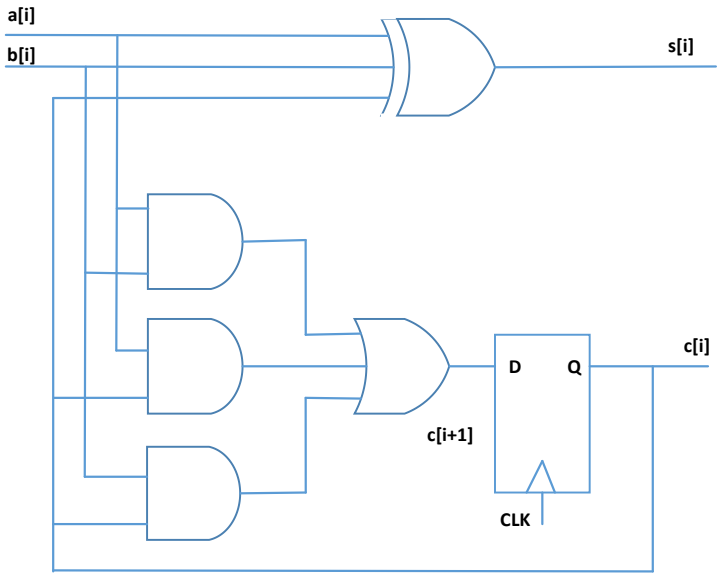

Figure 2. Bit-serial adder.

Bit-serial addition algorithm

- Uses n clock cycles
- Steps for A+B
    1. Clear Carry register
    2. for i = 0 to n-1:
       clock Carry register, read s[i]

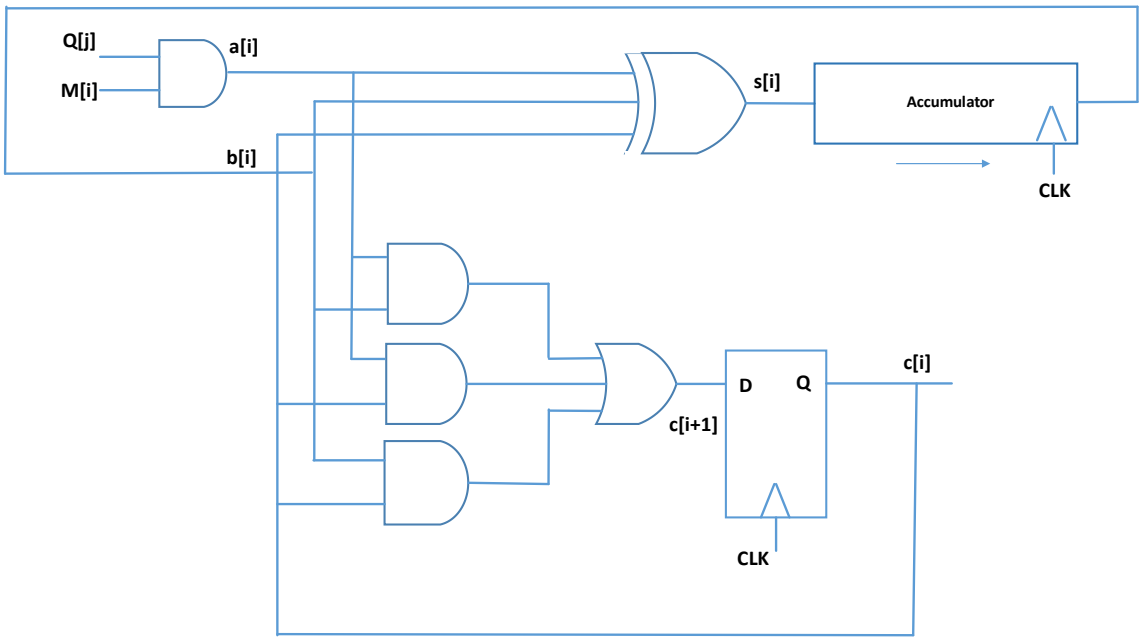

Figure 3. Bit-serial multiplier.

Bit-serial multiplication algorithm

- Uses n^2 clock cycles
- Steps for M x Q:
    1. Clear Product register (Accumulator) and Carry register
    2. for j = 0 to n-1 : --- $a_i = M_i$, if Q_j=1 otherwise no add, shift Accumulator only
       for i =0 to n-1:
       clock Carry register and Product register (Accumulator)

Digital filter algorithm

Implement the following IIR filter:

$Y[n]=X[n]\times(1-d)+Y[n-1]\times d$

where X[n] and Y[n] are input and output values of sample [n], Y[n–1] is an output value of the previous sample [n–1], and d is a constant that determines the filter coefficients. The constant d has the value $0<d<1$ and, physically, it represents the amount of decay between adjacent output samples when the input signal drops from a high level to a low level. The above filter is a digital implementation of a single-pole analogue filter with the time constant τ. A fixed relationship exists between d and τ: $d=e-1/\tau$, where e is the base of natural logarithms. The equation above yields: $Y[n]=Y[n-1]+(1-d)\times(X[n]-Y[n-1])$ which can be interpreted as a crude discrete approximation of the analogue single-pole filter's behaviour.

Implementation of the digital filter algorithm in the bit serial processor

You are required to develop both a smallest possible bit-serial architecture and a machine-level program for the filter presented in the previous section. The coefficient d which defines the filter must be included as an immediate literal in your program. You can choose one of the following values of d: 0.125, 0.375 or 0.875.

8-bit input samples values must be read from the switches SW0-SW7 on the FPGA development system and the resulting pixel coordinates after the transformation displayed on the LEDS LED0-LED7. Switch SW8 provides handshaking functionality as described in the pseudocode below. Switch SW9 should act as an active low reset.

Pseudocode

1. Pre-initialise Y[n-1] to 0
2. Wait for X[n] by polling switch SW8. Wait while SW8=0. When SW8 becomes 1 (SW8=1) read X[n] from SW0-SW7.
3. Calculate Y[n], display it on the LEDs and wait for switch SW8 to become 0
4. Go to step 2.

Representation of fractions in fixed-point 2's complement

Affine transformation and fixed-point representation

Input samples X[n] and output values Y[n] are 8-bit 2's complement signed integers, i.e. their values are in the range -128..+127.

The filter coefficients d and 1-d are 2's complement fixed-point fractions in the range 0 .. +1 – 2^(-8), i.e. it is a 2's complement fractional numbers with the radix point positioned after the most significant bits. A possible suitable representation is:

Bit position	Weight
7 (MSB)	-2^0
6	2^(-1)
5	2^(-2)
4	2^(-3)
3	2^(-4)
2	2^(-5)
1	2^(-6)
0 (LSB)	2(-7)

When the filter coefficients are multiplied by integer samples, a double-length 16-bit product is obtained which is a 2's complement number with the radix point positioned after the 9-th bit. Note however that the result Y[n]] must be an 8-bit 2's complement whole number.

Binary multiplication examples

Binary multiplication of 2's complement 8-bit numbers yields a 16-bit result. As one of the numbers is represented in the range -1..+1-2^{-7} and the other in the range -128..127, it is important to determine correctly which 8-bits of the 16-bit result represent the integer part which should be used for further calculations. The following examples illustrate which result bits represent the integer part.

Example 1. Multiply 0.75 x 6.
In 2's complement 8-bit binary representation these two operands are:

weights:	-2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
0.75 =	0**.**	1	1	0	0	0	0	0

weights:	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
6 =	0	0	0	0	0	1	1	0**.**

The 16-bit result is:

-2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
0	0	0	0	0	0	1	0	0.	1	0	0	0	0	0	0

which represents the value of 4.5. The shaded area shows which bits need to be extracted when the representation is truncated to 8 bits. Note that the fraction part is discarded entirely, so the 8-bit result is now 4. Also note that when the leading bit is discarded, the weight of the new leading bit must now change from 2^7 to -2^7. Why? The importance of the correct interpretation of the leading bit's weight is evident in the following example, where the result is negative.

Example 2. Multiply -0.25 x 20.

The two operands in signed binary are:

weights:	-2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
-0.25 =	1**.**	1	1	0	0	0	0	0

weights:	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
20 =	0	0	0	1	0	1	0	0**.**

The 16-bit result is:

-2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
1	1	1	1	1	1	0	1	1.	0	0	0	0	0	0	0

which is -5 in decimal representation. Again, the shaded area shows which 8 bits to extract when the result is truncated from 16 to 8 bits for further calculations. The truncated 8-bit result has the weight of -2^7 on the most significant bit so that it still correctly represents -5:

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	1	1	0	1	1.

If you would like to experiment more with binary multiplication of signed numbers, run some SystemVerilog simulations of your multiplier in Modelsim, which is what you should do anyway to test your multiplier module before synthesis. If you would like to practice signed binary multiplication by hand, I recommend you use Booth's algorithm (and rather fewer than 8-bits!):

en.wikipedia.org/wiki/Booth's_multiplication_algorithm

The original paper where Andrew Booth published his algorithm back in 1951 can be found here: http://qjmam.oxfordjournals.org/content/4/2/236.short

Design strategy

Develop SystemVerilog code and a separate testbench for each module in your design. Simulate each module in Modelsim. Synthesise each module in Altera Quartus and carefully analyse the synthesis warnings, statistics and RTL diagrams. When you are satisfied that all your modules are correct, write a testbench for the whole design and simulate. Synthesise the whole design and again, carefully analyse the warnings, statistics and RTL diagrams. You will be able to take an FPGA Development System on loan for a few days in Week 6, Week 7 or Week 9. A detailed schedule of loans will be published on the ELEC6234 notes website Test your design either at home or in the laboratory. In Week 9 or 10 (after the Easter Break) you will be asked to demonstrate your design in the Electronics Laboratory.

Formal report

Submit an electronic copy of your report through C-BASS, the electronic handin system, and a printed copy to the ECS front office by the deadline specified on the ELEC6234 notes website. The report should not exceed 3000 words in length. It must contain a full discussion of your design, including the final circuit diagrams, your instruction set and your program. A Word template will be provided with suggested structure of the report. Source files must be packaged in a zip file and submitted electronically as a separate file at the same time. In this exercise, 20% of the marks are allocated to the report, its style and organisation, with the remaining 80% for the technical content. As always, bonus marks are awarded for implementation of novel concepts.

tjk, 9 Feb'15