Practical assignment ECA2:
Polynomial CλaSH

November 21, 2018

Introduction

Below is the set of polynomial assignments of homework exercises for ECA-2, to be handed in by groups of two students. We assume you successfully completed the tutorial (tutorial.pdf).

deadline for handing in your solutions: 2018-02-02, on blackboard.

Remarks on delivery

- Add your names and student numbers to the front page of your report, and at the top of the code file.
- We have provided a file `Polynomial.hs`, please fill in your student last names and numbers at the top.
- Write your CλaSH code in the provided `Polynomial.hs` file. `topEntity` functions are commented at the bottom of the file. comment and uncomment when necessary.
- For all assignments you are asked to deliver CλaSH code, please include your CλaSH code for that specific assignment in a text box in your report.
- There is no need to deliver VHDL code.
- In some assignments you are asked to describe/show the combinational path, we prefer for you to draw this path in a figure.
- If you are asked to comment about clock cycles, we are looking for comments describing the amount of clock cycles delay for latency and throughput.
- You can score 2 points in this assignment, this is 0.2 points of the final grade.
- Deliver both your report (pdf) and the CλaSH code (`Polynomial.hs`)
- Report name: `ECA2_Polynomial_lastname1_lastname2.pdf`.

Preliminary Remarks on Haskell and CλaSH

- A filename of a Haskell/CλaSH program should be of the form `<Filename>.hs`, starting with a capital letter.
- You can transform Haskell to CλaSH by the following steps:
  - add the line

    import `Clash.Prelude`

    as the second line of your file. Among others, this redefines many standard list functions in Haskell towards corresponding vector functions in CλaSH. For example, in Haskell functions such as *take* and *map* work for lists, whereas in CλaSH they work for vectors. If you need such a function for lists, use *Data.List.take*, *Data.List.map*, etc. However, this only works in the `clash` interpreter, and not in a program file.
  - define the hardware types needed, using `Signed`, `Unsigned`, `Vec`, etc.
  - Instructions how to install CλaSH on your own system can be found on `clash-lang.org`.
- Finally, to generate VHDL code using CλaSH, you should define the variable *topEntity* and make sure that its type is not polymorphic and fully determined. Note that Haskell (and thus CλaSH also) can derive the type of an expression, to be checked in Haskell/CλaSH with:

  ```
  :t <expression>
  ```

  VHDL code is generated by CλaSH using the command[1]

  ```
  :vhdl
  ```

  This will put the resulting VHDL code in a subdirectory `vhdl/<Filename>`. We assume that you have access to *Quartus* for synthesizing the VHDL generated by CλaSH.

[1] Don't bother about possible "`Can't make testbench`" errors, they are not relevant in our context.

1 Standard polynomial

The general format of a polynomial of one variable is:

$$a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0 \tag{1}$$

The degree of this polynomial is n. In this assignment we consider $n = 4$ so:

$$f_0(x) = a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x + a_0 \tag{2}$$

For these assignments you may use `Signed 16` as number type. Choose any arbitrary numbers as coefficients.

Assignment 1 (0.5 pts):

Implement the f_0 function in CλaSH in two different ways, one by using the power function (^), the other by only using multiplication (so $x^4 = x * x * x * x$). Generate VHDL and show the RTL schematics and Flow summaries of both implementation, comment on the differences. Show the combinational path.

2 Factorizing

Factorization provides a mathematically equivalent but structurally different variants of polynomial functions, for example $ax^2 + bx \Rightarrow (ax + b)x$

Assignment 2 (0.5 pts):

Write in CλaSH the factorized version of f_0, produce RTL schematic, and comment on the combinational path.

3 Higher-order functions

The structure from Assignment 2 can also be implemented using a higher-order function.

Assignment 3 (0.5 pts):

Give a definition of the polynomial using a higher-order function, provide the RTL schematic.

4 Time-area trade off

Let us consider an FPGA with only 1 adder and 1 multiplier.

Assignment 4 (0.5 pts):

Transform the higher-order function from Assignment 3 to a mealy machine, simulate the functionality and show the simulation function and its result. Show the RTL schematic and Flow summary, comment on resource usage.