Hardware design in education using CλaSH

Rinse Wester   Jan Kuper   Christiaan Baaij
Dep. of Electrical Engineering, Mathematics and Computer Science
University of Twente
Enschede, The Netherlands
{r.wester, c.p.r.baaij, j.kuper}@utwente.nl

In order to bring the theory of embedded computer architectures into practice, a practical assignment has been developed where students design digital hardware using the functional hardware description language CλaSH. The assignment focuses on specifying the structure of FIR filters and analyzing the resulting hardware. By working through the assignment, students gain insight in the use of higher-order functions in the context of hardware design. It also shows that the declarative aspect of functional programming also applies to functional hardware design.

1 Introduction

At the University of Twente, students of the master program Embedded Systems can take a course on advanced computer architecture. The course focuses on designing regular architectures for signal processing applications. In order to bring the theory into practice, a practical assignment has been developed. The assignment involves the design of several regular architectures such that students become familiar with the implementation aspects of these architectures.

During the course, architectures are either expressed in mathematics or in Haskell. Choosing a functional language for the implementation on hardware is therefore logical. The language of choice is CλaSH, a functional hardware description language developed internally.

The rest of the paper is structured as follows: Section 2 gives some background information on the material covered in the course and on the CλaSH language. Section 3 covers the different architectures to be implemented by students using CλaSH. Finally, in Section 4 conclusions are drawn and plans for the future are presented.

2 Background

As mentioned before, students implement different filter architectures on an FPGA platform using the functional hardware description language CλaSH [2]. CλaSH descriptions are translated to synthesizable VHDL by the CλaSH compiler.

The CλaSH language has many features that can also be found in Haskell. Examples are Polymorphism, pattern matching, type derivation and higher-order functions. These features allow circuit designers to describe parameterizable circuits in a natural way. Especially higher-order functions provide a lot of insight into the structure of the resulting hardware.

CλaSH is a synchronous hardware description language where every component is expressed in terms of a Mealy machine i.e. every output and new state is a function of the current state and input. Listing 1 shows an example of a multiply accumulate expressed in CλaSH.

All hardware components described using CλaSH have a structure as expressed in Listing 1. In order to distinguish between inputs for the component and data coming from registers, a keyword *State* is used.

Submitted to:
TFPIE 2013

Listing 1 Multiply accumulate in CλaSH

```
$mac\ (State\ s)\ (a,b) = (State\ s',out)$
  where
    $s' = s + a * b$
    $o = s'$
```

Every variable preceded by *State* will be translated to a register by the CλaSH compiler. In Listing 1, the first occurrence of state s is the current state (the output of the registers) while the second occurrence s' is the new state of the register (the input). Figure 1 shows the architecture described by the CλaSH code of Listing 1.

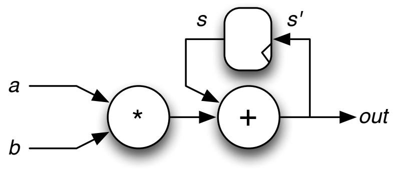

Figure 1: Multiply accumulate architecture

The practical assignment is given during the course Embedded Computer Architectures 2 which focuses on regular digital signal processing architectures like systolic arrays. These architectures are derived from a mathematical definition taking into account resource sharing and scheduling. Due to regularity of the resulting architectures, they can be nicely formulated in Haskell using higher-order functions. Several architectures are derived during the course, including: FIR filters, pattern matchers and auto-regressive filters. In order to bring the theory, presented in the course, to use in practice, students are performing digital filter design using CλaSH.

3 The practical assignment

The goal of the the practical assignment is to let students experiment with different, mathematically equivalent, FIR filter hardware structures in CλaSH. Students are given a framework in which they can instantiate these FIR filters. This framework targets the DE1 Development and Education board [3]. Central is the FPGA around which peripherals like an audio codec, buttons and LEDs are placed. By programming the FPGA with the complete framework, the filters can be physically tested by listening to the filtered audio.

The assignment consists of three exercises where every exercise covers a distinct FIR filter architecture. A standard, a transposed and a symmetrical filter have to be implemented, simulated and synthesized. The assignments focus especially on, a limited set of, higher-order functions to express the structure of the FIR filters. When the filters are implemented using CλaSH and synthesized using the Quartus FPGA tooling [1], students are asked to describe the relation between the higher-order functions in the CλaSH description and in the resulting hardware on FPGA.

3.1 Standard FIR filter

A FIR filter is a streaming architecture that calculates a weighted sum over current and previous input samples according to the formula:

$$y_k = \sum_{n=0}^{N-1} h_n \times x_{k-n}$$

The corresponding hardware structure is depicted in Figure 2.

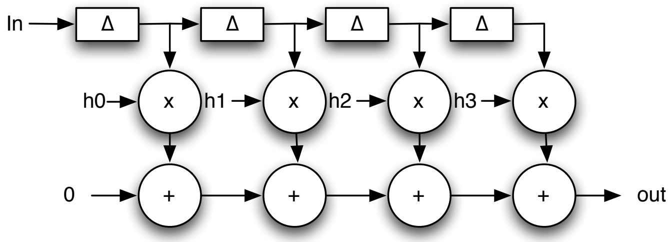

Figure 2: Standard FIR filter

To become familiar with CλaSH and the toolchain, the CλaSH code for the standard FIR filter is already given. The first step is to simulate the design using Haskell to determine the impulse response of the filter. The second step is to compile the design using the CλaSH compiler to generate VHDL code. This VHDL code is then integrated in the DE1 framework and synthesized using the Quartus FPGA synthesis tooling. Finally, the structure of the design on FPGA has to be related to the initial CλaSH description. Listing 2 shows the CλaSH function describing a standard FIR filter.

Listing 2 Standard FIR filter function in CλaSH

```
arch :: State SVect → (Sample, Bit, Bit, Bit, Byte) →              (State SVect, (Sample, Byte, Byte))
arch (State xs)         (audio_in, key1, key2, key3, switches) = (State xs′,    (audio_out, red_leds, green_leds))
  where
    audio_out = if key2 ≡ Low
                  then audio_filtered
                  else audio_in

    xs′ = audio_in +>> xs
    ws = vzipWith fpmult xs filtercoefs
    audio_filtered = vfoldl (+) 0 ws

    red_leds   = switches
    green_leds = switches
```

As shown in Listing 2, the design accessible to students is a single function with predefined in and outputs. The in and outputs will be mapped to the corresponding switches, LEDs and audio port on the DE1 board. The actual filtering part is written using only three lines of CλaSH code. First the next state xs' for the registers is found by shifting in a new audio sample. The second line described the pairwise multiplication of delayed samples with the filter coefficients. Finally, all weighted samples are added together using a *vfoldl* (similar to *foldl* in Haskell but applied to vectors).

3.2 Transposed FIR filter

For the second filter architecture, the transposed form, only a schematic is given and students have to specify the circuit in CλaSH. Figure 3 shows the structure of a transposed FIR filter. After specifying the design in CλaSH, the filter should be simulated to show that the transposed FIR filter behaves exactly the same as the normal filter. If this is the case, VHDL code can be generated and synthesized such that the resulting hardware can be analyzed. The last step is to describe how higher order functions like *vmap* and *vzipWith* in the CλaSH description are mapped to an FPGA.

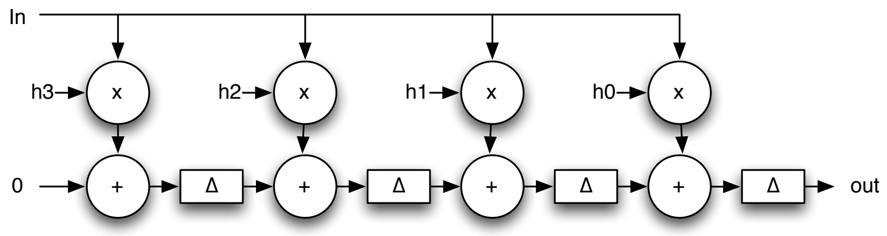

Figure 3: Transposed FIR filter

3.3 Symmetrical FIR filter

FIR filters are usually symmetrical in their coefficients i.e. the coefficients form a palindrome. Due to this symmetry, the formula of the FIR filter can be rewritten such that the number of multipliers is reduced. Consider the first and last coefficient: $h_0 * x_k + h_{N-1} * x_{k-N+1} = h_0 * x_k + h_0 * x_{k-N+1} = h_0 * (x_k + x_{k-N+1})$. Since the whole FIR formula can be rewritten such that first pairs of samples are added before weighted, the number of multiplications is halved. The resulting architecture is shown in Figure 4.

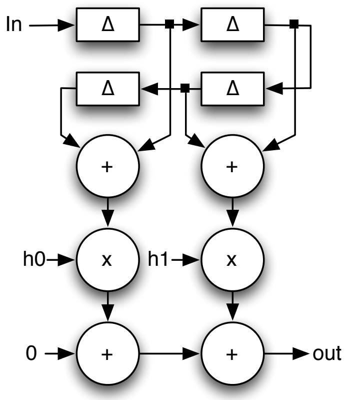

Figure 4: Symmetrical FIR filter

Again, students have to specify the architecture in CλaSH, perform a simulation to verify its correctness and generate hardware that has be analyzed.

4 Conclusion

To bring the theory of regular digital signal processing architectures into practice, a practical assignment has been developed. In this assignment, students have to design different FIR filter architectures using the functional hardware description language CλaSH. Especially higher-order functions are an important theme in the assignment as it an elegant way to express structure.

The practical assignment is now included in the course for two years and feedback from the students is very positive (also from students that had no previous experience with digital hardware design or functional programming). Several students also chose to do their Masters Thesis work on digital hardware design using CλaSH due to the material presented in the assignment. Currently the assignment only consists of FIR filters but this will be extended in the future with a wider range of architectures.

References

[1] Altera (2013): *Quartus design software*. Available at `http://www.altera.com/products/software/sfw-index.jsp`.

[2] C. P. R. Baaij, M. Kooijman, J. Kuper, W. A. Boeijink & M. E. T. Gerards (2010): *CλaSH: Structural Descriptions of Synchronous Hardware using Haskell*. In: *Proceedings of the 13th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools, Lille, France*, IEEE Computer Society, USA, pp. 714–721.

[3] Terasic (2013): *Terasic DE1 board*. Available at `http://www.terasic.com.tw/cgi-bin/page/archive.pl?No=83`.