Embedded Computer Architecture 2

Riccardo Battistig s1716239, Pietro Pennestrì s2382660

February 1, 2020

Contents

Code Reference

You can access the project repository at:

https://gitlab.com/eca2/polynomials

1 Standard Polynomial

The Polynomial given is seen in equation (1).

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

Where we will use the 4^th order polynomial, as seen in equation (2). Where all signals are 16 bit signed numbers.

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

Three permutations of this computation were synthesized in order to analyze their differences. Overall, the three permutations compute the same 4^th order polynomial result, with the similarities that the multiplications with the coefficients a_n and addition of the terms is equivalent, but the only differences are in the computation of the x^n terms. It is therefore recognized already that the combinational paths of these implementations will only differ by their computation of their x^n terms.
The function $f0p$ uses the higher order clash command ”^ to compute the powers of x. It is therefore entirely dependent on the implementation of this function what the speed/area efficiency of this implementation will be. The implementation will become more clear once the generated VHDL is analyzed, the use of this standard higher order function is useful as comparison basis for the other functions. In figure 1 the RTL schematic can be seen. What is apparent from its analysis is that the higher order function automatically forms the shortest multiplication path in a tree-like structure, similarly to summing. A critical path is highlighted in red, another critical path would be through the partial computation of $a_4 x^2$. In computing this term the partial outcome of x^2 is multiplied with itself to safe resources.
The function $f0m1$ implements explicit multiplications in the of the x^n terms, resulting in n times a multiplication of the constants with x. From the figure 1 it can be seen that the x terms are derived more explicitly, resulting in an increase of a factor of 2 of the amount of multiplication blocks needed. Once again a critical path is highlighted, however it can also be said that the critical path should probably actually be the one passing by the multiplier on the bottom, as multipliers generally have a higher latency than adders, but this wasnt taken into account in the making of this particular figure.
The function $f0m2$ is similar to $f0m1$, however, it explicitly indicates that first the x^n terms are computed first an foremost, and then constructed with the coefficients. As can be seen in the bottom of figure 1 this can exactly be seen. As only three multipliers are needed to compute the multiplications needed, as the recurrency of the values is used to full effect and the outputs of the multipliers are used in different parts of the schematic wherever they might reoccur.

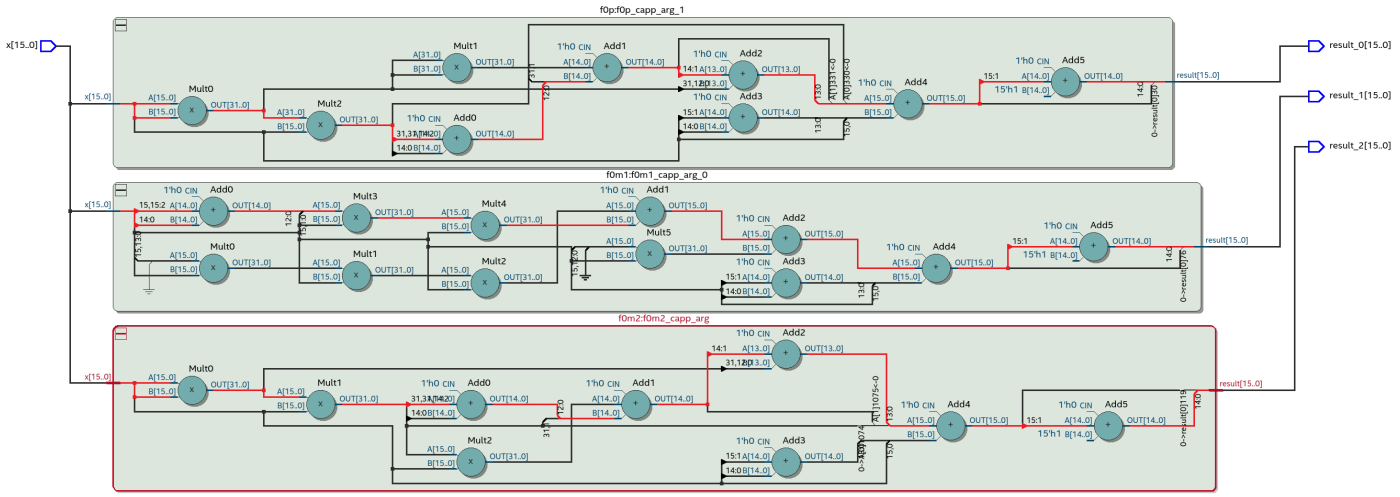

Figure 1: RTL Schematic of Equation (2) in all three permutations, from f0p down to f0m1 and f0m2.

2 Factorizing

By factorization the polynomial in equation (2) can be expressed as in equation (3).

$$f_1(x) = (((a_4x + a_3)x^3 + a_2)x^2 + a_1)x + a_0 \tag{3}$$

The factorization results in a great reduction of the needed amount of operations for the function, as there is only a need to multiply by x 4 times. In comparison this is 6 less multiplications by x compared to f0m2. The price to pay however is an increase in combinational path, as the initial x is part of a chain which totals 4 multipliers and 4 adders.

3 Higher-order Functions

The higher order functions used to implement the equation in (3) are: scanl, sum and zipWith. First a vector of x is created by using the repeat function, this is scanned from left over the multiplication, so that there are intermediate outputs creating the x^n outputs. This result is then multiplied using zipWith together with the coefficients, and finally summed.

The RTL schematic of this function is seen in figure 2, where the longest combinational path is highlighted in red. From the figure it is clearly visible that there is first a tapped chain of x multiplications (scanl), which each lead into a multiplication with the constants (zipWith), and then are summed in a tree-like structure (sum).

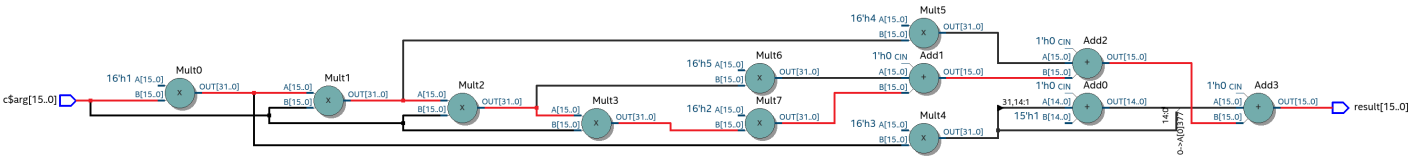

Figure 2: RTL Schematic of Equation (3) using Higher Order Functions

4 Time-area Trade Off

When the equation (3) is implemented using a mealy machine, we can take advantage of the added memory in order to reduce the area cost of the circuit. Namely, in theory we implement the calculation by only one Digital Signal Processing (DSP) operation at a time. DSP operations are multiplication or addition, since they are necessarily different DSP blocks, they can be scheduled in the same clock cycle (as they don't share resources). Each clock cycle an intermediate result is computed and stored in the accumulator register, until the final value is outputted. The resulting implementation now has a very small combinational path: of either the multiplication block or adder block (whichever has the longest latency; probably the multiplier). The design pays for it's increase in area efficiency by computation time, as the final result takes 8 clock cycles to compute. However, since the critical path is reduced, this design allows for a higher maximum clock speed compared to the longer combinational path in the trivial implementation. The RTL schematic is too big too show, so was moved to be seen in the appendix in figures 5 and 6. The Flow summary however can be seen below in figure 4, as well as a zoom of the important parts of the design 3. As can be seen in the flow summary, and less well so in the design figure there is only 1 DSP block, which is most probably as the implemented design uses a single ALU able to compute multiplications as well as additions.

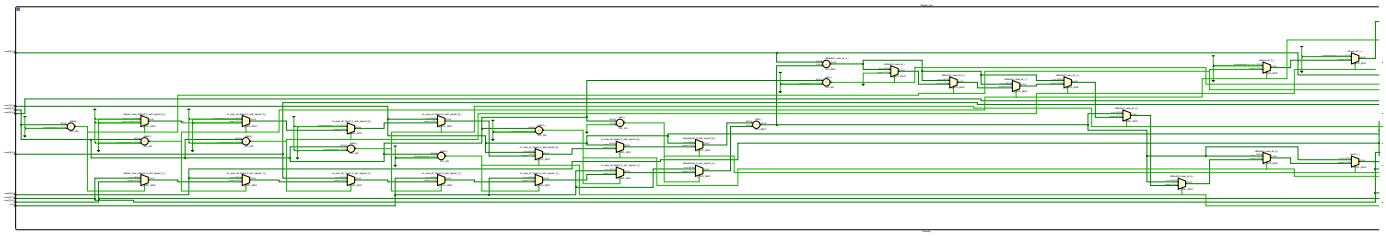

Figure 3: Flow Summary of Equation (3) using a Mealy Machine

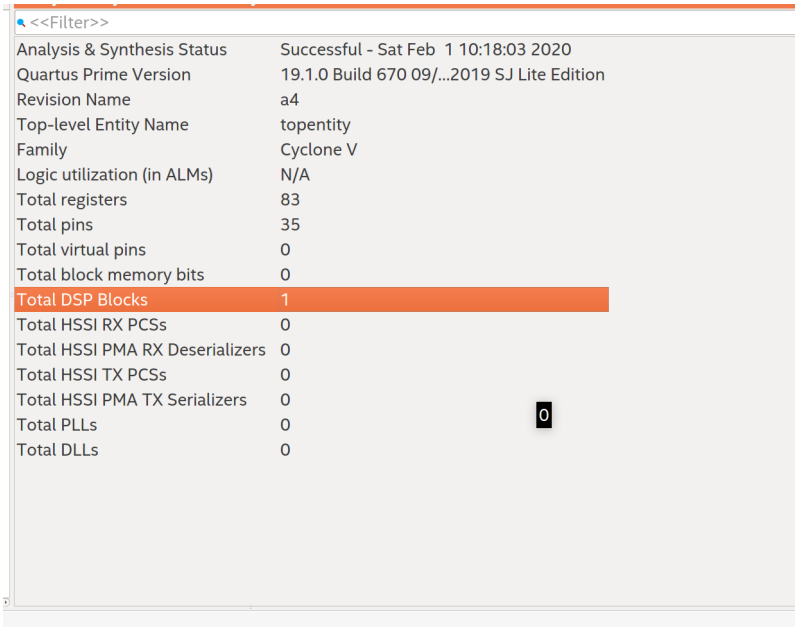

Figure 4: Flow Summary of Equation (3) using a Mealy Machine, yooming in on DSP parts

5 Appendix - Figures

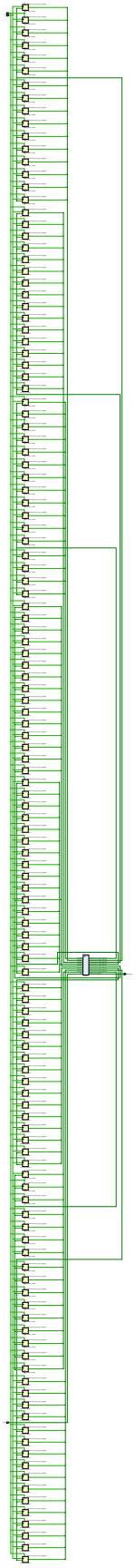

Figure 5: RTL Schematic of Equation (3) using a Mealy Machine Showing Registers

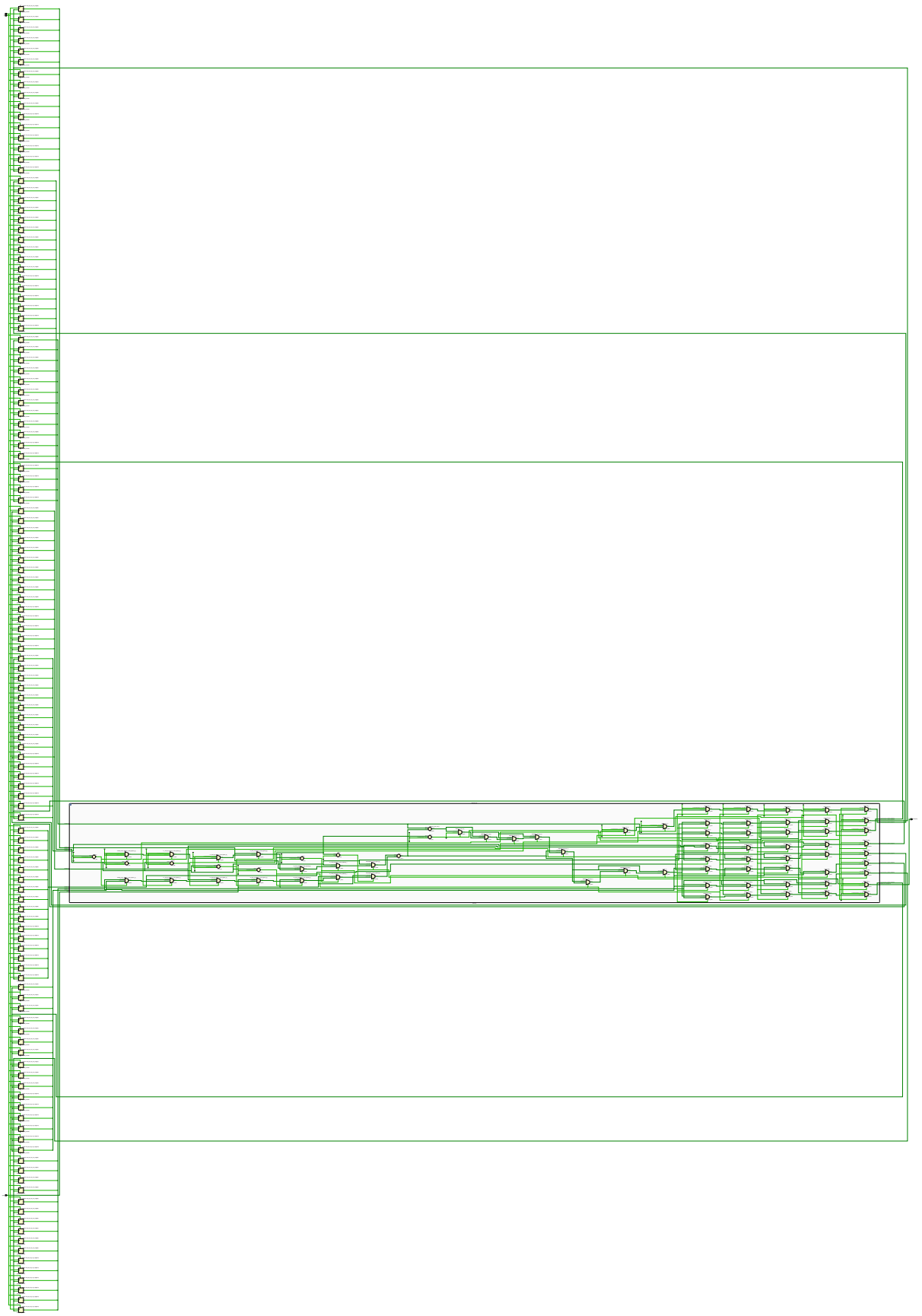

Figure 6: RTL Schematic of Equation (3) using a Mealy Machine Showing Full Function representation with Multiplexers