High-level synthesis

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 a by-hand synthesis of an FFT butterfly (see Figure 1) from a specification in pseudocode (high-level) to an RTL-level code in SystemVerilog. The SystemVerilog implementation should be demonstrated on an FPGA.

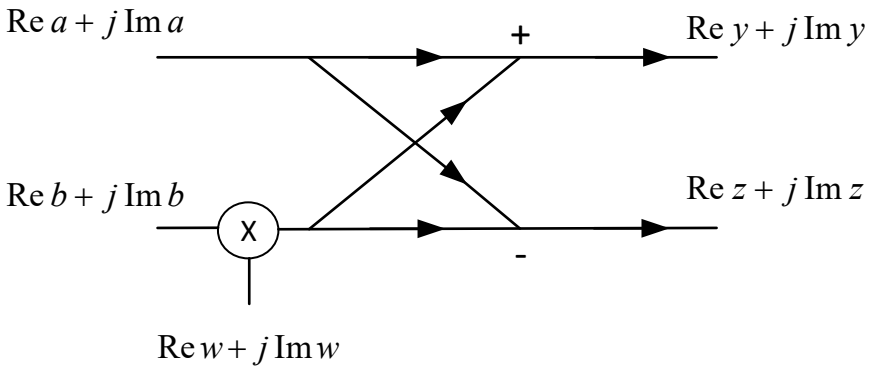

Figure 1. FFT butterfly.

The equations represented by the butterfly diagram in Figure 1 are as follows:

$$Re\ y + j\ Im\ y = (Re\ a + j\ Im\ a) + (Re\ b + j\ Im\ b) * (Re\ w + j\ Im\ w)$$
$$Re\ z + j\ Im\ z = (Re\ a + j\ Im\ a) - (Re\ b + j\ Im\ b) * (Re\ w + j\ Im\ w)$$

Where a and b represent a pair of the input samples, y and z are the corresponding output samples and w is the 'twiddle' factor, i.e. the corresponding trigonometric constant in the Fast Fourier Transform that is multiplied by the data in the course of the FFT transformation.

Your design should be a synchronous sequential system with an 8-bit data input connected to switches SW0..SW7 on the FPGA board, a clock a handshaking input signal ReadyIn connected to switch SW8. The design should have an 8-bit data output connected to LEDs LED0..LED7. Switch SW9 should act as an active-low master reset.

Pseudocode

The butterfly calculation algorithm executed by your design should run as follows:

0. Assert master reset (SW9=0)
1. Deassert master reset (SW9=1)
2. Wait for ReadyIn =0
3. Wait for ReadyIn =1
4. Read *Re w* from switches SW0..SW7
5. Wait for ReadyIn =0
6. Wait for ReadyIn =1
7. Read *Im w* from switches SW0..SW7
8. REPEAT
    a. Wait for ReadyIn =0
    b. Wait for ReadyIn =1
    c. Read *Re b* from switches SW0..SW7
    d. Wait for ReadyIn =0
    e. Wait for ReadyIn =1
    f. Read *Im b* from switches SW0..SW7
    g. Wait for ReadyIn =0
    h. Wait for ReadyIn =1
    i. Read *Re a* from switches SW0..SW7
    j. Wait for ReadyIn =0
    k. Wait for ReadyIn =1
    l. Read *Im a* from switches SW0..SW7
    m. Display *Re y* on LED0..LED7
    n. Wait for ReadyIn =0
    o. Wait for ReadyIn =1
    p. Display *Im y* on LED0..LED7
    q. Wait for ReadyIn =0
    r. Wait for ReadyIn =1
    s. Display *Re z* on LED0..LED7
    t. Wait for ReadyIn =0
    u. Wait for ReadyIn =1
    v. Display *Im z* on LED0..LED7
9. UNTIL forever

Synthesis steps

Step 1. From the specification above produce a control/data flow diagram (C/DFG) representing the butterfly algorithm and the pseudocode above.

Step 2. By hand perform scheduling, i.e. the order in which the operation should executed and determine data path resources.

Step 3. Perform binding, i.e. determine which operations execute on which resources.

Step 4. Produce an RTL-level block diagram and generate an RTL model in SystemVerilog. In this step show the datapath registers, register inputs, control signals and the controller FSM.

Implementation Suggestions

Your design could be implemented as a control/data path sequential system illustrated by the two examples in Figures 2 and 3. This is what most high-level synthesis systems do.

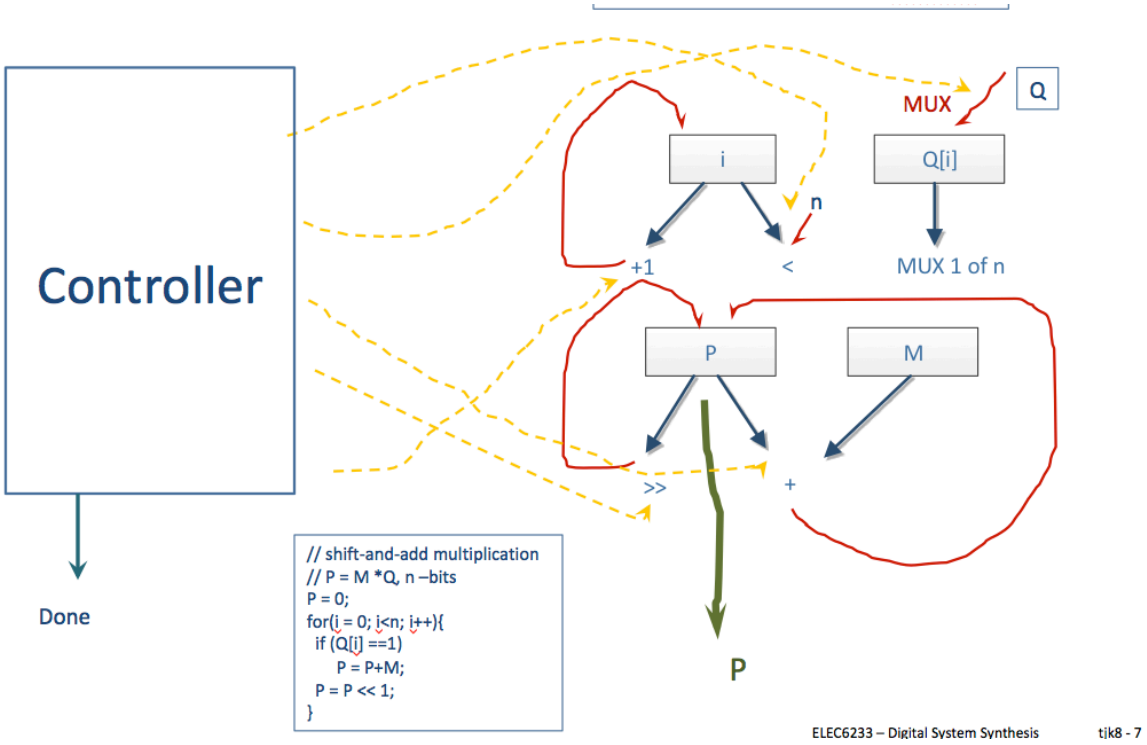

Figure 2. High-level synthesis example of a shift and add multipler.

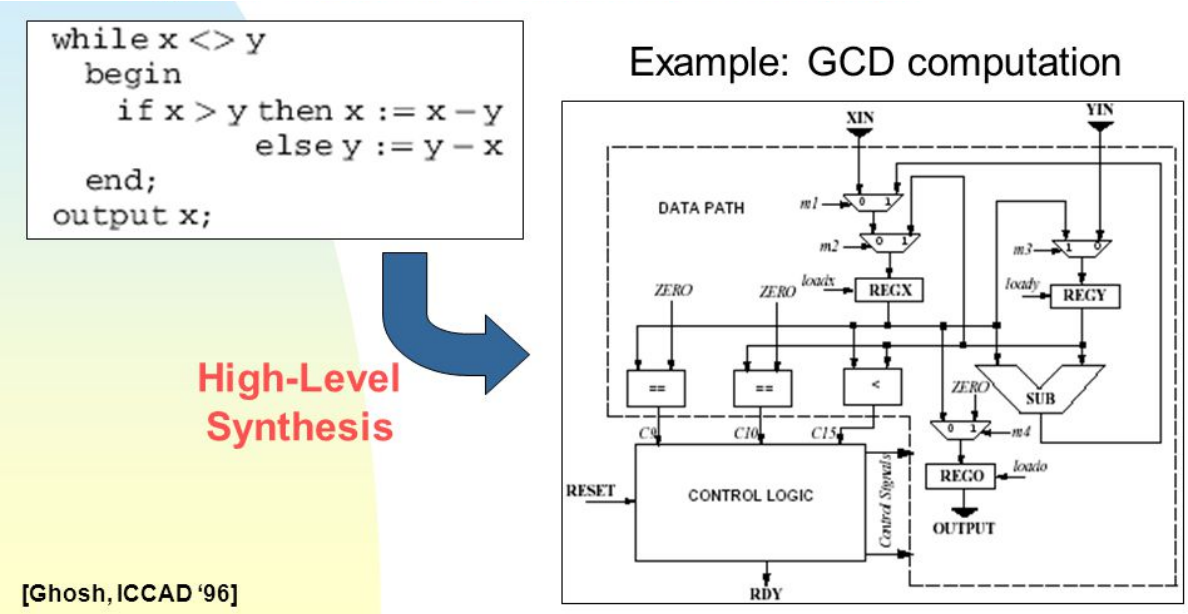

Figure 3. Example of hardware synthesised to implement a calculation represented by C code.

Alternatively, you could implement your design as a NISC (No Instruction Set Processor) shown in Figure 4 or, if you wish, an application-specific picoMIPS where the ALU and instruction set are custom designed for the butterfly operation. A general picoMIPS diagram is shown in Figure 5.

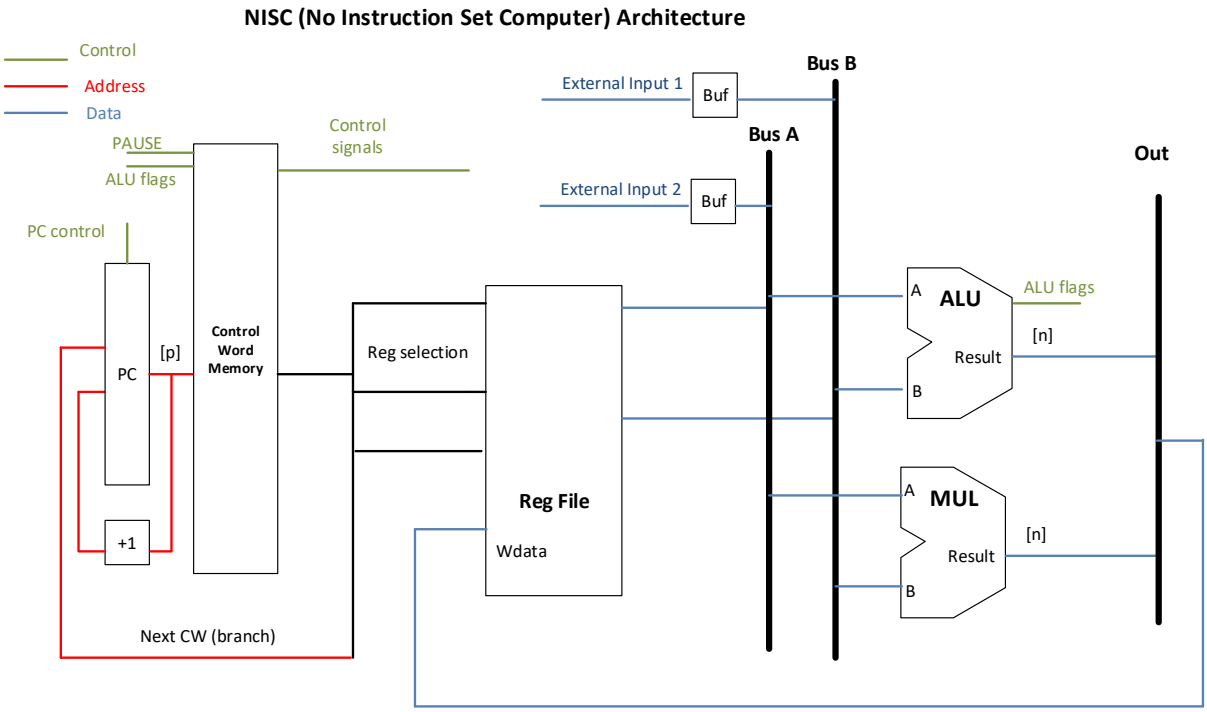

Figure 4. A NISC example.

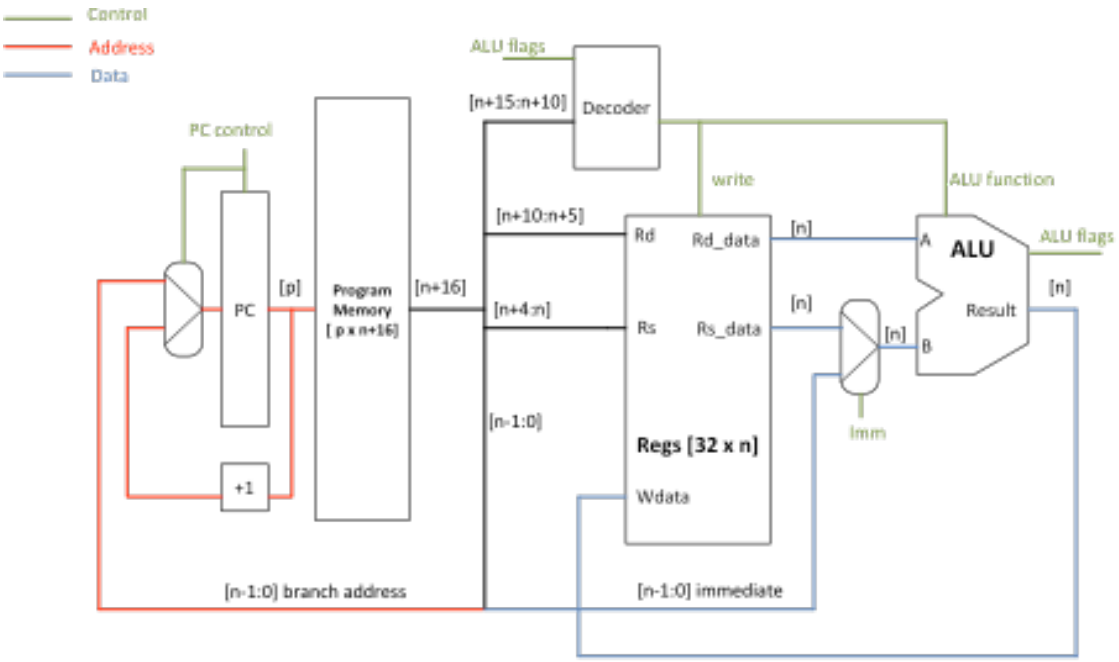

Figure 5. PicoMIPS with no RAM.

Suggested data formats for fixed-point arithmetic using fractions

Fractions and fixed-point representation

The twiddle coefficients are 2's complement signed fixed-point fractions in the range -1 .. +1 – 2^(-8), i.e. they are 2's complement fractional numbers with the radix point positioned after the most significant bits. Therefore the weights of the individual bits are:

Bit position	Weight
7	-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 twiddle factors are multiplied by input samples, which are whole numbers, 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 [x2,y2] 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**.**

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 ELEC6233 notes websites. The report should not exceed 2000 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, 7 Feb'20