ECE 4514 Fall 2019: Homework 4

Assignment posted on February 17
Assignment due on February 24 11:59 PM

Homework weight: 100 points

- Homework 4 is a simulation-based assignment on FSMD design. You will need to study and implement the CORDIC algorithm in fixed-point precision. Refer to lecture 7 and 8,

A Direct Digital Synthesis Design

In this Homework, you will design an FSMD implementation for a reference algorithm provided in C. The reference is written using fixed-point precision. The design is a Direct Digital Synthesis algorithm (DDS) for a sine wave. The output amplitude is normalized using fixed-point precision, and the implementation output encodes the waveform samples using `fix<16,14>.`

The DDS computes a sampled sine wave:

```
DDS(increment) returns output {
     output = sin(phase);
     phase = phase + increment;
}
```

The increment is a fixed-point number in radians, and it is encoded as `fix<16,15>` precision. Repeated execution of the DDS will thus run return the samples of a sine wave. A larger increment value corresponds to a bigger step. When the DDS samples would be played back at a fixed sample frequency, then a bigger step corresponds to a higher sine wave frequency (i.e., a higher pitch).

The top-level Verilog module interface of the DDS is the following:

```
module ddstop(input wire clk,
              input wire         reset,
              input wire  [15:0] increment,
              input wire         update,
              output wire [15:0] q,
              output wire        ready);
```

where

- `clk` is the system clock and reset is the system reset
- `increment` is the 16-bit phase increment (as `fix<16,15>`)
- `update` is a control pulse that initiates the computation of the next output sample
- `q` is the 16-bit output (as `fix<16,14>`)
- `ready` is a control output pulse that indicates completion of the computation

The module works as follows. When update is high at an upgoing clk, the DDS will sample the increment input, will set the ready output low, and will compute the output as the sine of the current accumulated phase. The output should be computed in less than 20 clock cycles. When the output is ready, ddstop will update q to the output value and set the ready output high for at least one clock cycle.

Reference implementation in C

To help you design the DDS, you get a reference implementation in C. This C implementation, dds.c, is a bit-accurate version of the Verilog design. This means that the Verilog design must be able to compute the same output for the same input increment value, as the C algorithm.

The reference implementation can be run from the command line, and it generates test vectors for the DDS testbench in Verilog:

```
dds samples increment float
```

where

- `samples` is the number of samples created by the algorithm
- `increment` is the phase increment (as a fix<16,15> integer)
- `float` is 1 or 0, for floating point output and Verilog test-bench testvector output, respectively

To compile dds.c, you should use a C compiler such as gcc (available in Cygwin) or MSVC (available under Windows).

Here is a sample output, for 15 samples with a phase step of 8000 (hex 1F40 = fix<16,15>(0.24414) ~ 0.0777 pi).

$ ./dds.exe 15 8000 1

```
a  1f40 s       1ef2 ( sin( 0.24414) =  0.24176 ) sin  0.24172 err  0.000037730345704
a  3e80 s       3c0b ( sin( 0.48828) =  0.46909 ) sin  0.46911 err -0.000022990819536
a  5dc0 s       5597 ( sin( 0.73242) =  0.66867 ) sin  0.66867 err -0.000001742444443
a  7d00 s       6a0d ( sin( 0.97656) =  0.82852 ) sin  0.82858 err -0.000055974000447
a  9c40 s       783c ( sin( 1.22070) =  0.93933 ) sin  0.93934 err -0.000009695388839
a  bb80 s       7f47 ( sin( 1.46484) =  0.99435 ) sin  0.99439 err -0.000038026637246
a  dac0 s       7ec8 ( sin( 1.70898) =  0.99048 ) sin  0.99047 err  0.000011299676612
a  fa00 s       76bf ( sin( 1.95312) =  0.92770 ) sin  0.92780 err -0.000094508434441
a 11940 s       67af ( sin( 2.19727) =  0.81003 ) sin  0.81010 err -0.000074485901196
a 13880 s       5279 ( sin( 2.44141) =  0.64432 ) sin  0.64436 err -0.000042618420986
a 157c0 s       385a ( sin( 2.68555) =  0.44025 ) sin  0.44040 err -0.000154873445106
a 17700 s       1aee ( sin( 2.92969) =  0.21039 ) sin  0.21032 err  0.000065364080685
a 19640 s   fffffbdf ( sin( 3.17383) = -0.03226 ) sin -0.03223 err -0.000027191161885
a 1b580 s   ffffdd12 ( sin( 3.41797) = -0.27289 ) sin -0.27287 err -0.000017114302286
a 1d4c0 s   ffffc056 ( sin( 3.66211) = -0.49738 ) sin -0.49733 err -0.000046996040890
```

When the program is run with the float parameter 0, it produces a test-vector for a Verilog testbench:

$ ./dds.exe 15 8000 0

```
f          // ignore this line, it is used by the testbench
1f40       // ignore this line, it is used by the testbench
f79        // first output
1e05       // second output
2acb       // third output
3506       // fourth output
3c1e       // fifth output
3fa3
3f64
```

```
3b5f
33d7
293c
1c2d
d77
fdef
ee89
e02b
```

These values are the expected output of the DDS implementation. For example, the first DDS output is 0xf79, which is fix<16,14>(0.2417), while the fifth DDS output is 0x3c1e, which is fix<16,14>(0.9393).

The reference implementation in C serves, therefore, a dual purpose:

1. To specific the functionality of the DDS
2. To generate testvectors for the Verilog design of the DDS

By redirecting the output of reference implementation into a file vector.txt, you obtain a test vector that can be used by the Verilog testbench:

```
$ ./dds.exe 15 8000 0 >vector.txt
```

Verilog testbench

You will only receive the Verilog testbench for ddstop, and you have to design the complete module, including the phase accumulator and the cordic algorithm. Refer to lectures 8 and 9 for a discussion of fixed-point arithmetic, the CORDIC algorithm and its use in a DDS.

Here is a sample run of a working solution.

```
# vsim -c ddstoptb -do "run -all"
# Start time: 21:30:35 on Feb 17,2019
# Loading work.ddstoptb
# Loading work.ddstop
# --> Testing       1024 vectors for angle step   200
# --> Simulation Complete.           0 errors
# ** Note: $stop    : ddstoptb.v(69)
#    Time: 491510 ns  Iteration: 2  Instance: /ddstoptb
# Break in Module ddstoptb at ddstoptb.v line 69
# Stopped at ddstoptb.v line 69
```

Note that the testbench requires the file vector.txt (produced using the C reference implementation) to run.

Verilator

You must run all of the Verilog code that you design, with exception of the testbench, through the lint checker of verilator. To install Verilator, follow the instructions on
https://www.veripool.org/projects/verilator/wiki/Installing

Installation under Cygwin is straightforward.

To run verilator in link checker mode, on e.g. ddstop.v, use the following command line:

```
verilator –lint-only ddstop.v
```

The tool generates warning messages when your code is not consistent. Here is an example of such a warning message:

```
$ verilator --lint-only ddstop.v
%Warning-WIDTH: ddstop.v:40: Operator ASSIGNW expects 17 bits on the
Assign RHS, but Assign RHS's COND generates 18 bits.
%Warning-WIDTH: Use "/* verilator lint_off WIDTH */" and lint_on
around source to disable this message.
%Error: Exiting due to 1 warning(s)
%Error: Command Failed /usr/local/bin/verilator_bin --lint-only
ddstop.v
```

To receive full grade on this assignment, none of the files that you turn in (with exception of the testbench `ddstoptb.v`) is allowed to generate a warning under verilator.

What to turn in

You need to design the DDS using one or more finite state machines with datapath. Push your solution before the deadline to github.

Coding Guidelines

Please refer to Chapter “RTL Coding Guidelines “ of the Reuse Methodology Manual (accessible for free on the Tech network: https://link.springer.com/book/10.1007%2Fb116360). These are generic guidelines for HDL coding; I expect you to aim to follow at least Chapter 5.2 and Chapter 5.5.

I value code clarity but I do not intend to penalize you if you don’t follow these guidelines to the letter. In the other hand, if you turn in messy code with obvious violations against the principles and main directives in this guideline, you will receive a penalty of up to 25% of the points (even if everything works perfectly).