EELE 466 Class Project
Task #2 Assignment

Due Wed April 8, 2015

For Task #2, you will run the Workflow Advisor in Matlab's HDL Coder toolbox on the first segment of Madgwick's code (Madgwick_qDot.m). This is a similar flow to the tutorial found in Help → HDL Coder → Examples (click tab) → *Getting Started with Matlab to HDL Workflow*

Step 1 : Create a new directory \hdl_coder_Madgwick_qDot

Step 2 : Open the HDL Coder Workflow Advisor

1. In Matlab (not the Editor) click on the *APPS* tab
2. You will see some apps, but you will need to expand the window by clicking on the small down arrow at the right.
3. Scroll down to the Code Generation row and click on HDL Coder
4. Name the project Madgwick_qDot.prj
5. Specify the Location by browsing to the directory you just created.
6. You will need to do this for each *New* project and make sure that the Type is specified to be *HDL Code Generation*.

Step 3 : Add Matlab files.

1. Under the Matlab Function header in the HDL Code Generation Panel, add Madgwick_qDot.m by clicking on the Add Matlab function link. This will be the function that is converted to VHDL code.
2. Under the Matlab Test Bench header in the HDL Code Generation Panel, add Madgwick_qDot_tb.m by clicking on the Add files link. This test bench script drives the inputs of the Madgwick_qDot function:

```
[qDot1, qDot2, qDot3, qDot4] = Madgwick_qDot(q0, q1, q2, q3, gx, gy, gz)
```

The test bench script must generate the input vectors that span all the values that the function to be converted to VHDL is likely to see. In our case, the quaternion is assumed to be normalized, so we will just generate values between 0 and 1, i.e. [0 1]. The gyro in the LSM9DS0 (for example) can have a full scale reading of +- 2000 dps which is 2000/180*pi rad/s.

Step 4 : Workflow Advisor

1. Click on the Workflow Advisor button in the HDL Code Generation panel.
2. In the panel on the left, click on **Define Input Types**. Run this step by clicking Run in the main window or right clicking and selecting Run this task. This runs the test bench script and determines that Matlab input data types (doubles).
3. In the panel on the left, click on **Fixed-Point Conversion**. Run this step by clicking on Run Simulation in the main window or right clicking and selecting Run this task.
    a. In the main window you will see the min and max values for each input/output that have been found when the simulation was run (which is why the test bench needs to cover all possible input values).
    b. It determines the appropriate fix-point representations as numerictype(S,W,F) where S=signed/unsigned, W=word width, F=fraction width. If you want to see what has been determined for each internal Matlab variable, you can find this information located at: \hdl_coder_Madgwick_qDot\codegen\Madgwick_qDot\fixpt\Madgwick_qDot_fixpt_Report
    c. You can control the rounding method by clicking on the Advanced button at the top of the main window and then clicking on the value to change it. You can also change the overflow action as well this way.
4. In the panel on the left, click on **Select Code Generation Target**. We won't change anything here, just keeping the workflow as Generic ASIC/FPGA. However, here we could be more specific if we wanted to specify the Quartus II synthesis tool and a specific FPGA device.
5. In the panel on the left, click on **HDL Code Generation**.
    a. In the Target tab, you can specify the language. We will keep it as VHDL.
    b. In the Coding Style tab, select Include Matlab source code as comments. This way we can see how the Matlab code was converted to VHDL. Select Generate report. Also select Industry as the HDL coding standard. This will produce shorter and more readable VHDL code.
    c. In the Clocks and Ports tab leave everything unchanged.
    d. In the Optimization tab select Register inputs. This will allow us to connect and pipeline our components together and allow the design to run at a faster clock speed. Later (don't do it right now) we will change the Resource sharing factor under Area Optimizations. Notice that adding a register adds a pipeline delay of 1.
    e. We will also leave the entries the same in the tabs Advanced and Script Options.
6. Right click on **HDL Code Generation** and select Run to Selected Task to run everything and generate the VHDL code.

7. In the main response window, click on the Madgwick_qDot_fixpt.vhd link to view the resulting VHDL code.
8. In the main response window, click on the Resource Utilization Report resource_report.html Make a note of the summary of how many Multipliers, Adders, Registers, RAMs, and Multiplexers are needed.   Notice that we require 12 multipliers and 9 adders.
9. Now go to the Optimization tab in the HDL Code Generation window and change the Resource sharing factor to 2.   This tells the code generator that we want to reuse the multipliers by running the clock speed twice as fast. Click Run in the main window to rerun the code generator. Scroll to the top of the output window and notice that it says: ### MESSAGE: The design requires 2 times faster clock with respect to the base rate = 1. Also note that it created several VHDL components that it now instantiates. Look at the VHDL code now and notice how much more complicated it is.   The resource summary now says that it will take 7 Multipliers, 9 Adders, 42 registers, and 30 multiplexers.

Task #2 Deliverables

Upload to D2L the following files:

1. Madgwick_qDot_fixpt.vhd (Resource sharing factor = 0)
2. resource_report.html (Resource sharing factor = 0)
3. Madgwick_qDot_fixpt.vhd (Resource sharing factor = 2)
4. resource_report.html (Resource sharing factor = 2)

Modify the file names so that the file name includes the sharing factor.