To: Prof. Ross Snider
From: Matthew Handley and David Keltgen
Regarding: EELE 466 Final Project
Date: May 5, 2015

Summary:
For this project, we converted a MATLab script into VHDL and tested the VHDL implementation using the MATLab Cosimulation toolbox. The MATLab script was based on work by Sebastian Madgwick in the document *An Efficient Orientation Filter for Inertial and Inertial/Magnetic Sensor Arrays*. The VHDL was generated using MATLab's HDL Coder toolbox.

Introduction:
The project was divided into 4 tasks. The first two tasks consisted of dividing the supplied MATLab script into four separate sub-functions (Madgwick_correction, Madgwick_normalize, Madgwick_qDot, and Madgwick_update) and a top-level function Madgwick_segments. Task 3 involved converting each of the four sub-functions into VHDL using MATLab's HDL Coder toolbox. In task 4 the top-level Madgwick_segments.m was converted into VHDL by manually writing a Madgwick_segments.vhdl entity to call the auto-generated sub-functions.

Body:
The first two tasks were very straight-forward to complete as it was simply reorganizing MATLab code. Task 3 was also straight-forward, as detailed instructions were provided for using the HDL Coder. The HDL Coder converts all variables to fixed point objects before converting the fixed point MATLab code into VHDL.

Table 1: Madgwick Sub-Function Resources

MATLab Function	**Madgwick_qDot**			**Madgwick_normalize**			**Madgwick_correction**		**Madgwick_update**	
Register Inputs	No	Yes	Yes	No	Yes	Yes	No	Yes	No	Yes
Resource Sharing Factor	0	0	4	0	0	4	0	4	0	4
Multipliers	12	12	4	6	6	10	124	49	8	3
Adders	9	9	9	3	3	3	133	133	8	8
Registers	0	7	55	0	4	20	0	734	0	59
RAMs	0	0	0	0	0	0	0	0	0	0
Multiplexer	0	0	35	6	6	28	12	361	0	24

Table 1 shows the resources used by each of the four sub-functions for resource sharing factors of both 0 and 4. The resource sharing factor is intended to reduce number of logic units required on the FPGA by increasing the number of clock cycles required to complete a calculation. The table shows that while some resources decrease for certain sub-functions, other resources actually increase with the increased sharing factor. For this reason, the final implementations all had the resource sharing factor set to 0 and register inputs enabled.

Task 4 took the longest to complete by far, due to several factors. In addition to implementing the Madgwick_segments vhdl entity, a top-level entity was created for Quartus compilation along with a MATLab Cosimulation test bench to drive the inputs of Madgwick_segments in ModelSim.

One of several issues encountered during this task was that Madgwick_normalize MATLab function required a Reciprocal Square Root (RSR) operation. Initially, this was implemented using the MATLab sqrt() function, however the HDL Coder was not able to synthesize this. The problem was solved by manually wiring our reciprocal square root VHDL from a previous lab into the auto-generated VHDL from the HDL Coder.

A follow-on issue involved the RSR which used a two-byte lookup table. The lookup table was over 65 thousand lines of VHDL. While this was easily compiled in ModelSim, Quartus was unable to synthesize it in a reasonable amount of time. To fix this issue, the resolution of the lookup table was reduced by ignoring the least significant nibble of the two-byte input.

Figure 1 shows the Resource Report from Quartus after compiling the final implementation. Only 784 logic elements were required to implement the Madgwick code on the DE2 board's Cyclone IV FPGA.

Resource	Usage
Estimated Total logic elements	784
Total combinational functions	421
◢ Logic element usage by number of LUT inputs	
-- 4 input functions	148
-- 3 input functions	150
-- <=2 input functions	123
◢ Logic elements by mode	
-- normal mode	360
-- arithmetic mode	61
◢ Total registers	633
-- Dedicated logic registers	633
-- I/O registers	0
I/O pins	422
Total memory bits	86016
Embedded Multiplier 9-bit elements	0
Maximum fan-out node	QIC_SIGNALTAP_GND
Maximum fan-out	400
Total fan-out	4159
Average fan-out	2.73

Figure 1: Quartus Resource Report

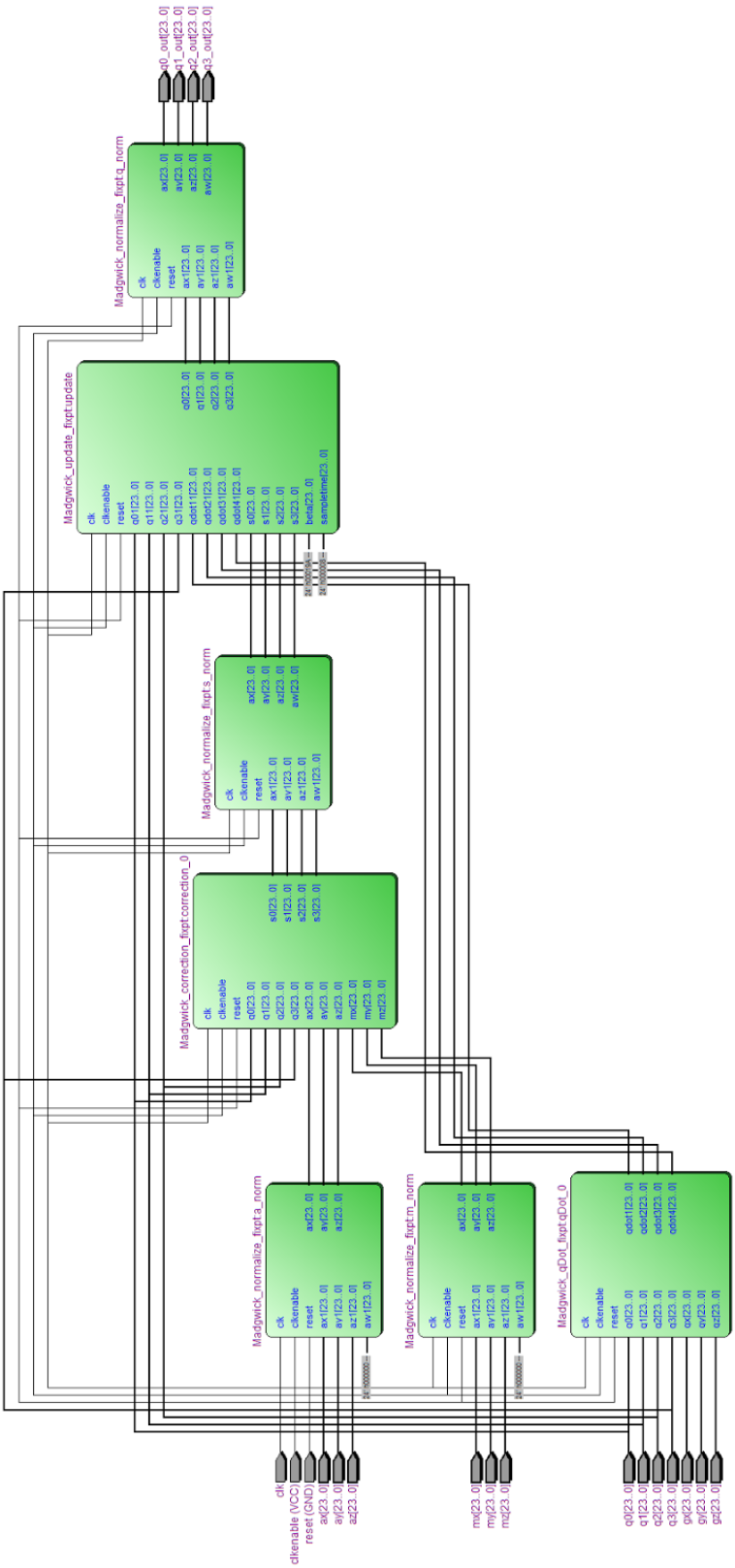

Figure 2: VHDL Block Diagram

Figure 2 shows the block diagram and wiring of the Madgwick sub-functions in the final implementation. Because each sub-function was implemented without resource sharing they only require 1 clock cycle to complete their calculation. The exception is the Madgwick_normalize function which utilized the RSR code. This code uses an iterative process and takes an average of 10 of clock cycles to complete.

The final VHDL implementation was tested and verified using the MATLab script Madgwick_segments_verification_tb.m. This script ran the VHDL in ModelSim using the Cosimulation toolbox and compared the output of the VHDL simulation to that of the original Madgwick MATLab code for a given set of inputs. After 100 cycles of testing, the maximum difference between the output of the VHDL and the MATLab was 0.0003. This small error is to be expected, as the output of the VHDL was a 24 bit fixed point number with 12 fractional bits. 12 fractional bits have a resolution of 0.0002441.

Conclusion:
The Madgwick MATLab code was successfully converted into an equivalent fixed point VHDL implementation. Testing showed that the output of the VHDL closely matched that of the original MATLab.