**Digital Design: Peak detector Project**

**Introduction**

This digital design project explored the design of a peak detector that was implemented to successfully sort and retrieve the greatest data value from a stream of inputs. The peak data value would then be delivered to the user along with the six surrounding data values (three above and three below) of the peak. The purpose of this project was to practice writing well-structured and organised code in VHDL that could later by synthesised in Xilinx and the implemented onto a FPGA. The project also involved the use of teamwork in order to distribute the overall development and allow for collaboration and problem solving with respect to particular tasks.

**Group Composition**

The group was divided so that two people worked on the data processor and two people worked on the command processor. The data processor team worked on the communication between itself and the data generator in which it received a byte every time a handshake (a change in the input to the data generator leading to a change in the output of the data generator) took place. In addition the team also worked on communication with the command processor in order to receive the start command and required bytes to process and also outputting the peak byte and seven byte sequence upon completion of the processing.

The data processing team was subdivided such that one person worked on the how the overall process would interact and developed an FSM chart and sequential logic in in order to develop it. In addition they also worked on the interaction between the data generator and the data processor in order to relay a correct hand shaking protocol in order to retrieve a data byte. Finally this person also developed a comparator to compare the latest byte of data with the current peak value to see if it was greater then, equal to or less then it.

The other person in the data processing team developed a counter system using three counters that each incremented from 0 to 9 and would track the index of the peak byte and surrounding data bytes. The counter implementation was also used in order to count the number of times the process should run with respect to ‘numwords’. In addition to this they also worked on a subtractor that was used in order to determine the correct value of the peak index as the index value was one value greater than the actual index of the stored data bytes.

**Project Plan**

After the team was subdivided into the command processing team and data processing team, a project plan was developed in order to meet the overall goals of the project. To do this, each team developed a set of design goals and tasks that related to their specific process. In addition the overall deliverables of the project were found and all of the goals were developed into a Gantt chart. The Gantt (see figure X) chart allowed the team to plan how long each task was expected to take thus allowing us to maintain a similar amount of work throughout completion of the entire task. The chart also allowed us to plan team meetings that could be used for the discussion and development of certain tasks.

**Architecture of Data Processor**

The algorithm of Data Processor module is divided into 6 states. The processes are triggered with enable signals that are defined in each of the states.

Each of the individual processes are described below:

Counter: The command processor sends the number of bytes to be processed in BCD format, at port ‘numWords\_bcd’. The algorithm used in the Data processor needs to know what the value of ‘numWords-bcd’ as it only stops asking for new bytes from the Data Generator and processing the peak value when the number of iterations equals the value of ‘numWords\_bcd’ sent by the command processor. The Counter process is designed to run three counters in the form of hour, minute and second hand of a clock. ‘Counter1’, ‘Counter2’, ‘Counter3’ are 4 bit std\_logic\_vectors, that are respectively used to compare each of the 4-bit digits of numWords\_bcd.  The signal ‘counter2’ is incremented only when the signal counter1 completes iteration from value 0 to 9.

Subtractor: