Through various applications, it is necessary to find a path of least resistance through a grid. An obvious application of this problem is finding the fastest path through a topographic map. The analytical approach to solving this problem is to divide the map into grid sections and assign a rating scale (TNC) to each section taking into account gradient and other map features such as terrain or obstacles. By summing up the TNCs of the cells along each possible route, the path of least resistance can be found. To simplify this problem it is necessary to assume that each grid cell can only be accessed by an adjacent grid cell and that it takes the same amount of time to reach any adjacent cell from any starting location within each cell. Theoretically if each cell is made infinitesimally small compared to the size of the total grid these general assumptions will not affect the fastest path through a grid.

After the creation of the grid there are multiple numerical approaches to solving this problem such as an A* search. However, arguably the fastest way is to create a ripple effect that begins at a designated start location and tracks the path taken to reach the end location in the fastest amount of time (See US Patent 5548773 A**)**. Such a design can be implemented into hardware and be universal expanded dependent upon timing and size constraints to a grid of any size desirable. To demonstrate a proof of concept of this design, a 16 x 16 grid was used to find the fastest path along a grid of dynamically assigned cells loaded through a serial interface and outputted to a VGA display.

To implement the design, I began with a basic layout by separating the functions of different modules and shells within the architecture. The top level modules were a serial in parallel out decoder, a word comparator, a FSM, a unit cell grid, retracing logic and a VGA output component. Within these modules were much more basic design units such as 256 address MUXs, decrementers, comparators and large registers to store information. The problem can be separated into three distinct functions: reading in new information, calculating a fastest route and outputting TNC and path information to the VGA. At the center of this design is the unit cell which serves as the bridge between the loading of new information and the output of information in each specific cell. However, since the cell design needed to be duplicated 256 times, each cell needed to optimize space by minimizing the functions of each cell.

In some ways, the unit cell could be seen as a basic synchronous, parallel loaded, clock enabled decrementer. However, the key function of these unit cells was to track a fastest path, so an additional 4 bit register was added to each cell to signal how to find a fastest path to any location on the grid by tracking which adjacent cell triggered its activation. To find the start and stop address external comparators and register are used to activate a single cell as a start or stop address. Once all cells are loaded with data, the grid can be enabled and activated cells will ripple outwards till the stop address is activated. Finally, once the stop address cell is activated by an adjacent cell, a done signal is sent to the FSM to initiate the retrace logic.

The retrace logic is simply a synchronous process activated by the FSM to calculate the next location of a cell on the fastest path and then to activate a high signal in a 256 bit vector for use by the VGA in knowing if a cell is on the fastest path. To find which adjacent cell activated the current cell on the fastest path, a triggered by array is used as a mux to say if the next address is to the north, south, east or west.

To interpret the current TNC value of a cell or to display if a cell is on the fastest path a VGA component was used. The main code for a VGA display and a color design of a 16 x 16 grid was provided. However, the color choices to display a start address, a stop address, a cell on the fastest path, or a cell with a given wait time where open for interpretation. Rather than give an exact TNC in each cell, the TNC groups were binned into 5 categories of roughly 50 TNC

values: 0-56, 57-106, 107-156, 157-206, and 207-256. The VGA logic output an address and received this information for either the cell connection grid or the retrace logic to determine the color of each cell.

After debugging and re-designing, the implementation of the 16x16 grid was successful. However, one undiscovered issue in this project was the definition of how each cell should function, the reason for this issue was a lack of specification in the description of the design of the unit cell. For example, a clock enable would make it possible to overcome possible error that came about as a result of the number of clock cycles it took for a one cell to activate another cell. However, this issue was mitigated by adding a clock enabled tick to minimize the error in this transition. For my implementation my clock enable activated a single decrement of an active cell counter once every 250000 clock cycles slowing a 100 MHz clock down to a 400 Hz clock minimizing the error associated with the transition between cells.

Additionally, since the exact definition of timing was not clearly defined, some of the test maps could not sever as a universal test across all designs. For example, in my logic, a TNC of "00000000" takes 1 enabled clock cycles to execute an activate out tick, a TNC of "00000001" takes 2 enabled clock cycles, and a TNC of "111111111" takes 256 enabled clock cycles (Appendix C). Therefore, the test map logic assumed that a TNC of "00000001" preformed like mine logic does at "00000000" which causes my path to work differently than expected in the test maps. Since this timing was never specified, my program functions within the guidelines and will theoretically decrease the error associated with crossing cells my adding an additional 250000 cycles (caused by an additional required clock enable tick) to each cells activation further reducing error associated with activation delay.

One of the first lessons in the approach to digital design was the need to consider speed, size, and power constraints when implementing architecture. However, through the majority of the course, there was a minimal concern for power and size constraints and merely a focus on timing in order for proper functionality of an implementation. None the less, in such a large scale final design in which components are replicated up to 256 times, the majority of the time in design was spent focusing on the time constraints in order to minimize the size of the unit cell and maximize the effective use of space on the chip. In the first two iterations of design, my implementations used up roughly 180% and then later 130% of the available chip space. These iterations proved successful in the full test bench level, but needed tweaking to minimize space. At first, the two items that I used too much of were registers and LUTs. Both of these were a result of excessive data flow and storage into and out of each unit cell. However, after optimizing critical information stored to only the TNC and “triggered by” values in each cell and removing an address register from the unit cell I was able to minimize my design to 76% of my LUTs and able to implement a final program on my board for demonstration.

By far the most interesting part of this project is this programs ability to demonstrate parallel computing as my program could theoretically load serial data in, calculate a fastest path and output data to the VGA display simultaneously. Further applications of this digital design include possibly finding other paths of least resistance such as flux. All in all this project was an interesting example of large scale implementation of the basic components demonstrated in ENGS 31, and it shows the capability of FPGAs for testing designs and the importance of device optimization for final implementation.