ECE 3770 Final Lab Report

Trevor Crone

6848906

April 4 2012

Abstract

This report details the design, implementation and operation of the ECE 3770 (Digital System Design 2) final lab project called GRAFIX. GRAFIX is an image display program written in Veriglog. GAFIX is capable of producing pixel coordinates for dots and lines, generated by serialized X-Y coordinates and gray scale color. This is achieved using 3 main componets: Fist the ALU, responsible for logic operations. Next the DPU is responsible for data collection and marshaling. Finally the CCU is responsible for sending commands to the DPU and ALU via the DPU.

# – Introduction

The goal of the final three laboratories of ECE 3770 is the design implementation and operation of graphics processor called GRAFIX. The GRAFIX system produces X-Y pixel coordinates and grey scale colors of pixels. Theses pixel can then be plotted on to a screen. GRAFIX can produce these pixels in one of two ways: point plotting and line plotting. In point plotting, a single pixel is produced. In line plotting, an algorithm is used to plot a straight line between two X-Y points. The GRAFIX system is to be written in a Hardware Description Language (HDL) to simulate a physical circuit implementation.

GRAFIX is written in the HDL language Verilog. Verilog is a C based HDL, well suited for the context of this project. The GRAFIX system is made up of three main components; The ALU, the DPU and the CCU. The ALU is the preforms logic operations such as addition and subtraction, which is required to plot lines. The DPU moves data to and from the ALU. The DPU also stores the data required for GRAFIX to function and plot lines. Finally, the CCU sends instructions to the DPU so that the DPU in conjunction with the ALU can plot points and lines. The 3 components are it’s own Verilog file. They are designed to by module in their operation, such that the ALU and the DPU can work independently. This allows the GRAFIX system to be expanded or modified if needed.

The report that follows will detail the deign decisions and Verilog code written to achieve the GRAFIX system. The next chapter will discuss the design of the main components of GRAFIX. The chapter after that will then detail how the modules were implemented in Verilog. And finally, the second to last chapter will discuss the final results and performance of the GRAFIX system.

# – Design

This chapter details how the three main components of GRAFIX were designed. This chapter will discuss how the three components come together to achieve complete the system. The way each component is design is crucial because they need to function in harmony. As a result, the planning and design of the entire system needed to be done first with the three components in mind before any one part could be implemented.

## 2.1 ALU

The ALU or Arithmetic Logic Unit is very similar to a typical ALU. The ALU preforms a set of logic function from two inputs, the A accumulator and the B accumulator. In this case, both accumulators are 8-bit registers. The resulting data is then stored within the DPU, which will be discussed later. The ALU also contains a Control Code or CC register. This register displays information about the logic operations the ALU preforms. Finally, the ALU receives commands from a 3-bit N bus.

The ALU used in GRAFIX is capable of 8 logic operations. The 8 operations are addition, subtraction, shift left one bit, shift right one bit, maximum and minimum comparisons and finally, a function to pass either accumulator A or B through the ALU unchanged. Theses are commands are less numerous than what most ALUs are capable of, but for the purposes of GRAFIXs it is all that is needed. Each operation is easily implemented in Verilog. Using C operators like +, -, \*, / and conditional IF statements, all 8 operations can be done simply.

When the ALU is needed to preform one of its eight operations, a 3-bit command is send via the N bus. The ALU will then receive a binary number from 0 to 7, with each number corresponding to a certain operation. The op-code table is shown below in Table 2-1.

Table 2‑1: ALU Operations and Op-codes

|  |  |
| --- | --- |
| **Operation** | **Op-code (Binary)** |
| Addition (A + B) | 0 0 0 |
| Subtraction (A – B) | 0 0 1 |
| Shift right 1 bit (A \* 2) | 0 1 0 |
| Shift left 1 bit (A / 2) | 0 11 |
| Pass A to output | 1 0 0 |
| Pass B to output | 1 0 1 |
| Maximum (Greatest of A or B) | 1 1 0 |
| Minimum (Smallest of A or B) | 1 1 1 |

In addition to the eight commands above, the ALU will also ignore any number grater that seven. This is done so that data can be passed to the DPU without signaling the ALU. This will be discussed more in section 2.2.

The last component in the ALU is the CC register. The CC register is a 4 bit register that shows some indication about what is happening during the ALUs logic operations. The purpose of each bit in the CC register is shown in table 2-2.

Table 2‑2: CC Register

|  |  |
| --- | --- |
| **CC Bit** | **Meaning** |
| 0 | Carry bit C. Occurs when during addition subtraction and shift operations |
| 1 | Zero bit Z. Occurs when during addition, subtraction and shift operation when the result is equal to 0. |
| 2 | Greater than bit G. Occurs when A is greater than B during MIN/MAX operations. |
| 3 | Less than bit L. Occurs when A is less than B during MIN/MAX operations. |

The CC register is needed when plotting a line (it is used by the CCU and will be discussed in section 2.3). Thus both the DPU and CCU must be able to see the CC register.

In addition, the ALU is also asynchronous. When new bits are loaded onto either of the accumulator or new commands are sent through the N bus, the ALU will process the result. This means that the DPU has less work to do, but it also means that the ALU may inadvertently try to write data to DPU when it shouldn’t be. For example, when data is passed to the DPU, the ALU may try compute with that data, and that data may be mistakenly written back to the DPU. To remedy this, the ALU has writeEnable pin. This pin is set high when data should be written to the DPU. This pin along with the CC register is reset before each ALU operation.

## 2.2 DPU

The DPU or Data Processing Unit, marshals data to and from the ALU and CCU. The DPU stores data needed by the GRAFIX system. The DPU is responsible for receiving this data from the CCU, sending it to the ALU and the finally saving output from the ALU. The DPU also outputs data to video at the command of the CCU. In terms of hardware, the DPU is the simplest, only registers and buses exist in the DPU. Nonetheless, the DPU proves to be the most critical part of GRAFIX. Since the DPU communicate between both the ALU and the CCU, it must cooperate fully with each. Design decisions made here must match the design of the ALU and CCU.

The main component of the DPU is the register file. The register file contains all the data need for GAFIX to plot points and lines. The composition of the register file is illustrated in Table 2-3.

Table 2‑3: Register File

|  |
| --- |
| Dx |
| Dy |
| Error |
| EInc |
| EnoInc |
| Xs |
| Xe |
| Ys |
| Ye |
| X |
| Y |
| Color |
| 1 |
| 0 |
| Null |

The first nine locations are used by the Bresenham Algorithm to plot lines. While X, Y, and Color are the X-Y coordinates and grey scale color respectively of the pixel to be plotted. When output is sent to video, it is these three locations that are sent. The last location in the register file is left empty, and is the default write location for the ALU. This is done so that if the ALU runs when it should not, it will write data here instead of overwriting important data elsewhere. The second and third last locations are hardcoded as a 1 and 0 respectively. Theses values are there so they can be used in the Bresenham algorithm.

The DPU receives commands from the CCU. Theses commands consist of four parts: Abus address, Bbus address, Rbus address and N. The Abus and Bbus addresses are the register locations in the register file that should be loaded into ALU accumulators A and B respectively. The Rbus then designates where in the register file the ALU output will be written too. Finally, N is a four bit signal that commands the ALU, and is passed through to the ALU. However, in section 2.1, it is described how the ALUs op-codes is a 3 bit signal. The extra bit sent to the DPU is used to signal the other function within the DPU. In this function, data is to be passes from the CCU to the register file. In this case, the ALU needs to do no operation, and since it cannot recognize the 4 bit signal, it does no operation. In this data pass through case, an additional argument is needed. The mData bus is an 8 bit bus connected from the CCU to the DPU. In the case of a data pass through, the DPU will take the data from the mData bus and move it to the register file location identified by the Rbus argument. In this case, the ABus and Bbus arguments are not used. This extra functionality of the DPU is important for the GRAFIX system is it allows the data in register file to be initialized before plotting a point or a line.

Finally, there is an outEnable pin connecting the DPU and CCU. When outEnable is high, the X, Y and Color positions in the register file are loaded onto a 24 bit output bus. This bus is then connected to the external video setup.

## 2.3 CCU

The CCU or Central Control Unit, send commands to the DPU and by extension the ALU such that a point or line can be plotted. The CCU receives commands from the user to plot either a point or a line, as well as information needed to preform those operations, such as X-Y coordinates and grey scale color. This information is then used to plot a point or a line and send the information to the DPU and on to the video output. In the case of line plotting, several DPU/ALU operations need to be requested and preformed to plot the line proper.

The CCU is essentially a finite state machine. The CCU has three main states: idle, plot point and plot line. A user command selects a state and the CCU will preform the appropriate action. This command comes in the following serial fashion:

<L or P> <X> <Y> <color> <Xend> <Yend>

The last two arguments are sent only for line operations. In that case, the first X and Y coordinates indicate the starting position of the line, and the last two arguments indicate the ending positions of the line.

In the plot point state, the CCU simply relay the X, Y and color data to the DPU register file via the mData bus and sets the outEnable pin high. In the plot line state however, the CCU moves through a secondary set of sates. In the line state, there are 20 subsates. These substes preform the Bresenham algorithm to plot the line. The algorithm can be seen in appendix ??. The finite sate machine implementation of this algorithm follows the written algorithm very closely, with each line being its own states. All possible state operations can be reduces to either passing data to the register file, or signaling the DPU to process data in the register file through the ALU using a certain logic operation.

## 2.4 GRAFIX System

The GRAFIX system exists as a combination of the ALU, DPU and CCU. As discussed earlier, these components work together to plot points and lines. Figures 2-1, 2-2, and 2-3 illustrate the different state sequences involved in GRAFIX.

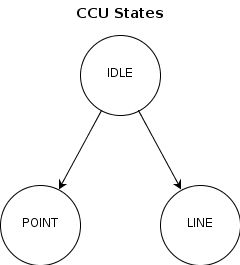


Figure ‑: CCU States

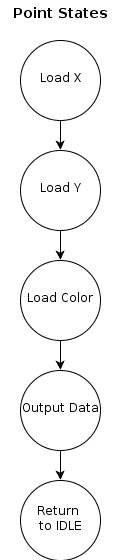


Figure ‑: Plot Point States

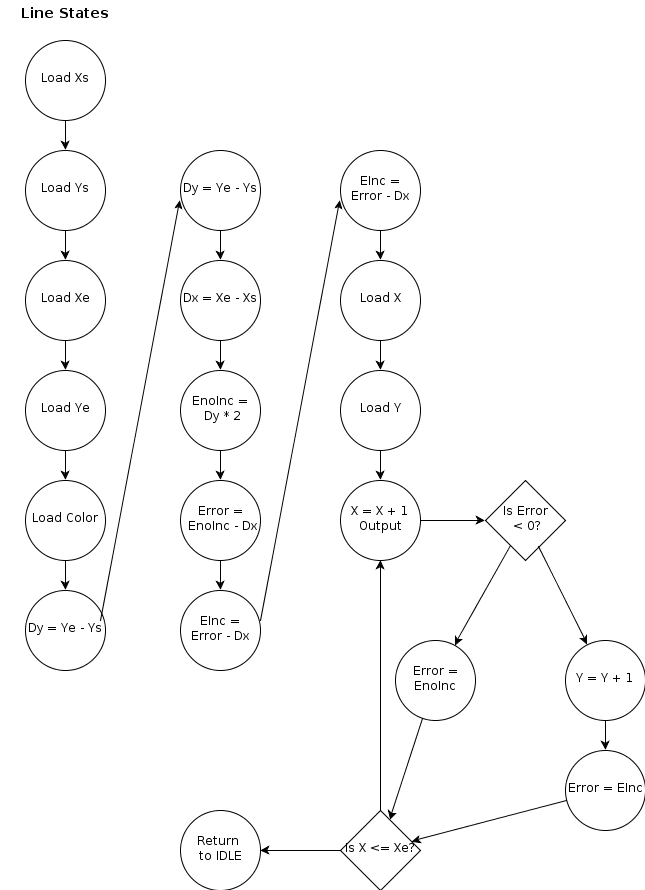


Figure ‑: Plot Line States

# – Implementation

Once an overall system design was made it was time to produce the Verilog code for GRAFIX. During this process, flaws in the initial design would come to light and lead to system improvements. This chapter details how the code writing process lead to system changes that allowed the GRAFIX system to be completed.

The three GRAFIX components (ALU, DPU and CCU) were built to work within one another. Meaning, the ALU runs within the DPU and the DPU runs within the CCU. Data and commands flow downwards from the CCU and certain data, like the CC register, is also read back upwards. This method makes communication between components easy to implement, it also gives the components a degree of independence from the sections above it. Using this architecture, the ALU was coded first, followed by the DPU and finally by the CCU. As each component was coded, it was also thoroughly tested to ensure it would be a robust as possible when as the GRAFIX system was built up.

The first component to be coded was the ALU. The ALU would prove to be the simplest and easiest to produce. Since the ALU needs no awareness of the system built around it, the ALU has none of the complexity associated with data marshaling and sequencing. The ALU code accepts three arguments: data for accumulator A, data for accumulator B, and N op-code. The ALU then returns four arguments: result data from ALU, CC register, and writeEnable. In addition, since the ALU is asynchronous, no timing or clock information is required. The code for the ALU can be seen in Appendix A.1. The only improvement made to the ALU during the testing process was the 2 accumulators. Initially, the ALU would read data directly from the DPUs register file. This however, would cause a race condition during the Bresenham algorithm. During the algorithm, the X coordinate increases by 1 each iteration. When the ALU attempted this, it would read X, increase X, notice the change in X, and run again. This the ALU would run forever. By first loading data from the register file to the accumulators, the ALU became isolated from this increase action as the accumulators are only loaded when the CCU sends a command.

Once the ALU was written and confirmed to work as expected, the DPU was to be written. While the DPU has the least amount of Verilog code in it, it was the hardest to produce conceptually. Since Verilog has no way to implement a traditional data bus, the register file interaction had to be done differently. In the final product, the register file is a 2 dimensional data array, 8 bits wide and 15 registers deep. Data is moved to and from the A, B and R bus using addresses. The CCU will transmit addresses from 0 to 14. This address signifies which register within the register file should be loaded onto each bus, and also where the data on the R bus should be places in the register file. The end result is a very elegant solution to deal with the DPUs busses. For example, to access a register in register file, say position 4, the Verilog code would be:

Abus = 4;

Dataout = registerFile [Abus];

The DPU also needs to pass the N bus and CC register between the ALU and the CCU. The DPU also needs to examine the N bus in the event of a data pass through command.

The data pass through function, as mentioned in section 2.2, is used to load data directly from the CCU to the register file. This data is loaded onto the mData bus and when the appropriate N bus signal is sent, data is taken from the mData bus and placed onto a register in the register file based on the Rbus address also sent from the CCU. This feature, while not explicitly outline in the GRAFIX requirements, is essential to initialize the system. Using an extra N bus op-code and the additional mData bus is the simplest way to load data to the register file and ignore the ALU.

Another important feature of the DPU is the writeEnable and outEnable pins. The writeEnable pin came about to prevent the ALU from overwriting data in the register file. During testing, if the very first command sent to the DPU is a data pass through, then the ALU will attempt to write junk data to the register file because he ALU has only just been initialized. The writeEnable pin prevents this data from being written to the register file. The outEnable pin is used to signal the DPU to push data to the output. This exists so that the CCU algorithm can control when data it is sent to the output. The code for the DPU can be seen in Appendix A.2.

The final part to be coded was the CCU. The CCU is a very large finite state machine. This was accomplished in Verilog by using a series of switch statement and state counters. The CCU moves through the switch statements and the state counter changes accordingly. Each state would be a line in the Bresenham algorithm, and the main while loop in the algorithm would slide back and forth between loop states. At the conclusion of the algorithm the state counters would be reset and the CCU would return to an idle state. The CCU is also the only clock-based part of GRAFIX. The CCU runs on edges of a mater clock. Each state in the CCU runs in a single clock pulse. This clock is also used to read data from the input, which arrives in a serial fashion as described in section 2.3. The code for the CCU can be seen in Appendix A.3.

# – Results

…

###### – Verilog Code

…

###### – Points to Ponder

1. The X-Y and color values can be sent to the frame buffer in a serial or parallel fashion. The data can be sent in 8 it packets one after the other using the ALU by request of the CCU. This is slower and requires more work for the CCU, but it does require less pins and wires between the DPU and frame buffer. The data can also be sent in parallel by load all 24 bits at once. At the CCUs commands, the X, Y, and color registers can be loaded into a single 24bit register and pushed out to the frame buffer. This is faster and means less work for the frame buffer, become requires more hardware cost.
2. In a sense, my GRAFIX system is synchronous. While the DPU and ALU are asynchronous, the CCU uses a clock and is synchronous. However, since the DPU and ALU run at the behest of the CCU, they could be considered synchronous.
3. GRAFIX has 2 external buses. One 8 bit buss comes from the user and instructs the CCU, the other is a 24 bit output bus that loads X, Y and color data to the frame buffer. GRAFIX has several internal buses:

* Abus addr, Bbus Addr, and Rbus addr
  + Sends register file address data from the CCU to the DPU
* Abus, Bbus, and Rbus
  + Loads ALU accumulators A and B and retrieve results from the ALU to the register file.
* N bus
  + Sends op codes from the CCU to the DPU and CCU
* CC bus
  + Sends CC register data back to the CCU

1. If the frame buffer was connected to some sort of video display that would plot its X-Y coordinates, GRAFIX could be used to plot basic shapes. Given enough time, GRAFIX could theoretically draw any image with points and lines. GRAFIX could also be used as a sort of graphing tool. GRAFIX could capture experimental data, record and plot the data in the frame buffer.
2. The CCU in GRAFIX interprets the users commands and can plot points or line. Essentially, the CCU is a FSM machine 3 states, idle, point and line. The CCU sits in idle until a point or a line command is received. At that point, the CCU will move into 2 sub-state machines to plot a point or line. These state machines relay data to the DPU and command the ALU to preform logic operations. The CCU also requests the DPU to output data to the frame buffer.
3. My GRAFIX system does not use the divide (shift right), pass A, pass B, or min functions of the ALU. If theses functions were to be eliminated, the ALU would only have 4 functions. At that point, the N bus could be reduced in size by one bit to facilitate the smaller set of op-codes. While this would reduce the hardware cost of the ALU, it would also greatly reduce the flexibility of the GRAFIX system. I don’t believe that a single bit reduction is worth a much more limited ALU.
4. For the most part, the DPU only serves to pass data from the CCU to the ALU and to store data for them as well. Thus, the DPU is very nearly optimal. However, the DPU could theoretically be eliminated entirely. The CCU could take over the roles of the DPU. The CCU could store data and communicate directly with the ALU. Removing the DPU would eliminate a sort of middleman in GRAFIX. Thus the optimal DPU would be no DPU at all.