Final Project: Delay Line Memory

Zoe Fiddler, Ezra Varady

December 2015

1 Introduction

In this project we investigated delay line memory, one of the oldest forms of memory used in computers, and its affect on computer architecture. Delay line memory stores data in a transmission media which creates a phase delay, allowing it to be accessed at a later time. The number of bits that can be stored in a delay line is dependent on the phase offset of the transmission line. It is a first in, first out memory, as opposed to the random access memories we have studied in class. This creates a number of interesting differences in the architecture of the device. It is also a refreshable memory, meaning it must be constantly rewritten back into the line or the data will be lost.

We investigated several methods of creating a delay line as well as some historical devices which used this form of memory. Some of the first computers, such as the EDVAC or the UNIVAC I, used a delay line. We specifically looked into the Olivetti Programa 101, as this had the simplest architecture. We analyzed the architecture of this CPU and compared it to the processors we studied in class. The architecture differed in many ways, both due to the limitations of the delay line and the limited availability of transistors at the time.

We also implemented a delay line memory in Verilog, which can read and write to individual registers in the line. This was modeled after the structure of the Friden EC130 calculator. This code would be easily integrated into a calculator or a larger CPU. This allowed us to explore many of the timing issues involved with delay line memory.

2 Background

Delay lines were first used for clutter detection in radar applications during WWII. They allowed a sweeping radar to compare a previous sample with the current sample to look for changes and track moving objects. Delay line memory was first proposed by Eckert and Mauchly in the 1940s. At the time, computers were built with vacuum tubes which were both large and costly. Delay Line memory offered a smaller, cheaper way to store data than the vacuum tube based memory used on previous computers. They used a mercury delay line to build EDVAC, one of the first stored program computers. Delay lines, both mercury and magnetostrictive, went on to be used in many computers during this era, but were eventually replaced by faster more efficient forms of memory.

There are many different ways to implement a delay line. The first was with a tube of mercury with piezoelectric transducers at each end. The waves propagated slower in the mercury and the impedance of the transmission media closely matched that of the transducers. The mercury had to be held at a specific temperature to keep impedance

and transmission speed uniform. Another early implementation was the magnetostrictive delay line. This consisted of a metal wire with a magnetostrictive transducer at either end. A torsional wave representing the signal would pass along the wire at a set rate. This had the advantage of being much smaller, as the wire could be wound. Other acoustic implementations include a piezoelectric transmission line or transmission through free space. A method has been proposed to bounce the signal off the moon, which would allow for up to 1 GB of storage.

Smaller electronic delay lines are still used in modern applications. These come in two forms, distributed and lumped element. Distributed delay lines, implemented in microstrip or coax, are used to create phase delay in modern computers. Lumped element delay lines, implemented with inductors and capacitors, can also be used to create small phase delays.

3 Verilog Implementation

To aid in understanding the design challenges associated with constructing a system around a delay line we implemented a memory system using one. We used an available block diagram of the Friden EC-130, an early digital calculator. The EC-130 used Reverse Polish notation, and this is reflected in our implementation. Our memory behaves like a stack, and supports two basic functions, pushing memory onto the stack, and popping the top two values off. The memory is divided into 4 registers, mirroring the EC-130. While there is currently no addressing scheme beyond an internal state keeping track of the next register to be written, the same technique used to align the input's phase with this register could be expanded to make the memory addressable. While we have no formal concept of an address, the serial access pattern of a delay line requires tracking certain analogous states. An input can be received at any point in the delay line's cycle but must be written to the correct register. We chose to store an entire word in each register, so we had to track both the current register, and the the current bit within that register. When we receive a write command we buffer the input word, and register the command internally, preventing further memory access until the write is completed. At the point when the current register is positioned such that it can be written to we convert parallel input to serial output, and upon completion unlock memory. Writes are performed using a mux. Typically it feeds the delay line's output back into the input, but when the desired register is aligned this input is replaced with the serialized input data. Reading memory follows a very similar pattern, however the need to decrement the current register twice adds complexity and in our implementation requires the use of additional solid state memory. When a read command is received, assuming the memory is in a state in which it can perform one, it will store both the registers in solid state memory for use by an ALU or whatever hardware is attached. These values are read in serial, and provided to the user in parallel. Because we read the higher register first, read operations require two full cycles of the delay line. Examples of reading and writing to the delay line can be seen in the waveforms section. While we use a considerable amount of solid state memory to support the delay line, increases in the length of the line do not require significantly more.

3.1 Future Work

Design requirements were very different in the early days of computing due to the high cost of transistors. Future work would include paring down the number of transistors used.

Because behavioral Verilog does not provide strong idioms for reasoning about transistor usage, we would also aim to create a fully structural implementation. Additionally memory access patterns could be made more efficient.

4 Delay Line CPU

We explored one implementation of a CPU using a delay line, the Olivetti Programa 101, the first desktop computer, which uses a magnetostrictive delay line. This was a programmable computer which could store up to 120 instructions and do 16 different operations. It could also be programmed using magnetic disks or operated manually. There are many similarities between this CPU and the processors discussed in class. A full block diagram of the system, taken from the patent, can be seen in figure 1 on the following page.

4.1 Memory Organization

The memory consists of 10 registers, each 22 characters long with each character being 8 bits. Two of the registers are reserved for instructions, 3 are used for operational purposes and 4 are used to store values. Each register can represent a single 22 digit number or be split into two 11 digit number. The bits are interwoven in the delay line, meaning the first 10 bits represent the first bit of each register, while the second 10 bits are the second bit of each register.Each character consists of a 4 bit binary number representing a decimal value as well as 4 tag bits, shown in figure 6 on page 9. Three of these bits serve to place the decimal point, indicate sign and flag unused values. The last has many uses depending on the situation. In the instruction registers it acts as the program counter. In the data storage registers it aids in timing and manual data entry. In modern CPUs, these actions are done by separate parts of the system but, because vacuum tubes were so costly, the designers chose to code this information into the delay line memory. Each instruction is 1 character, 8 bits, long so each register can contain 22 instructions. Optionally, some of the registers intended for data storage can be used to hold additional instructions.

4.2 Delay Line, PISO and POSI

The delay line has a serial output with a serial to parallel converter attached to it. A new bit is transmitted every 1 microsecond but the converter updates every 10 microseconds. This means it shows the nth bit of all 10 registers for the entire 10 microsecond period. One interesting side effect of this is that, using an oscilloscope, one could see what is stored in all of the registers at all times, which could aid in debugging. The literature did not describe how this converter is implemented but it could have been done with a shift register or microstrip delay line, then transferred to the output with a latch. Since delay line memory must be refreshed continuously, there is also a parallel to serial converter on the other end of the delay line which does the reverse and writes data into the line. This section is entirely different from the memory we studied in class, as it can see all the registers at the same time, but can only access 1 bit at a time.

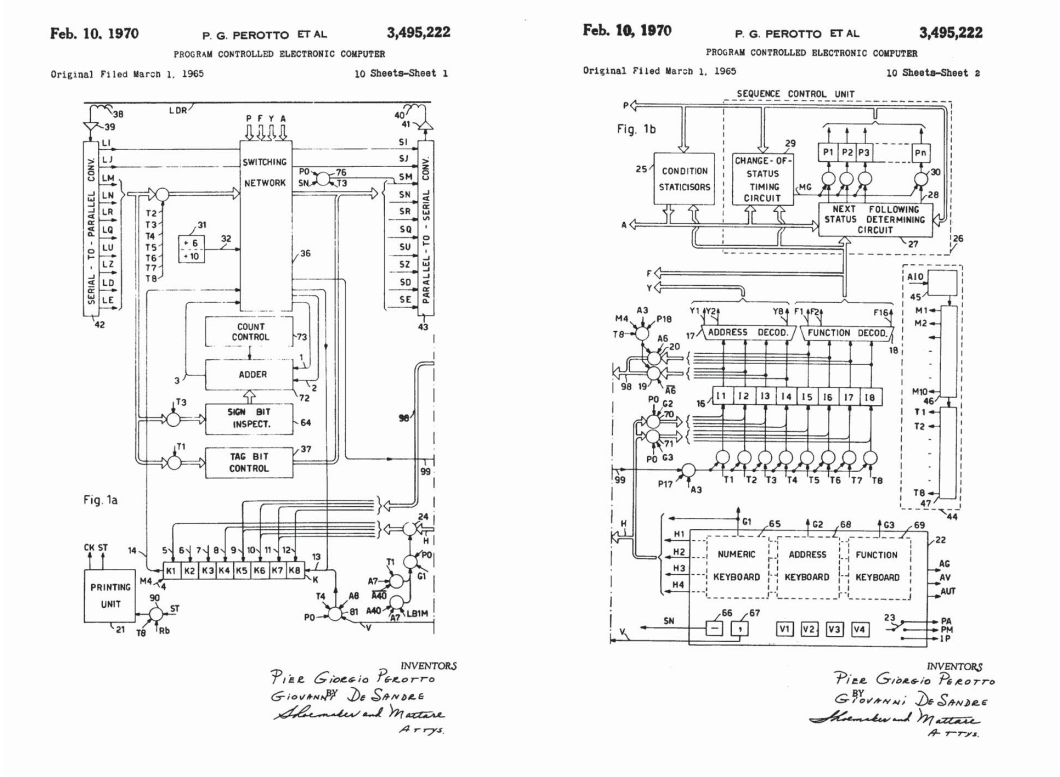

Figure 1: Block diagram of the Olivetti Programa 101

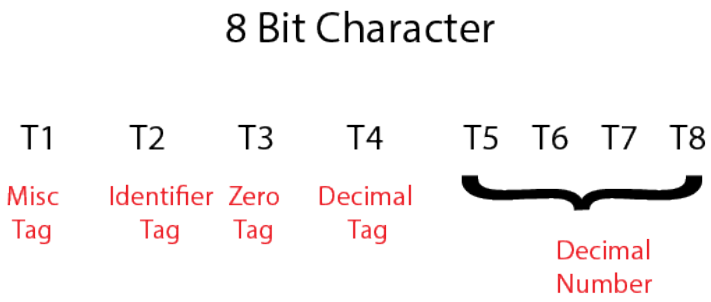

Figure 2: Decimal Value Character

4.3 Adder

The adder in this CPU is implemented differently than the adder we created for this class. We built our adder (and our ALU) using bit slice but, with a delay line, only one bit is available at a time. This means we only need a single binary adder with carryout. The addition is done near instantaneously compared to the clock cycle so two registers can be added together and written to another register in a single cycle. The carryout is stored in a bistable device and used during the next bit cycle or character. Subtraction is done in 1's complement (inverting each bit) and is noted by whether or not the negative tag is high.

4.4 Status Update and Sequencing

This section plays the role of the finite state machine and look up table we have seen in previous devices. For each instruction, there is a sequence of statuses which encode some of the bistable devices and control the behavior of the processor. The transitions between statuses are determined by the states of the bistable devices. The change of state then, in turn modifies those devices and executed the instruction. This is one section of the CPU which is very similar to the processor we studied in class, with no major differences due to the delay line memory.

4.5 Instruction Decode

Each instruction contains a 4 bit address followed by a 4 bit operation. There are two decoders, one for each part. The decode phase happens in much the same way as we learned it but the instructions themselves are quite different. Some of the possible operations are as follows,

- Addition
- Subtraction
- Multiplication
- Division
- Transfer from M
- Transfer into N

- Exchange
- Print
- Print and Zeroizes
- Program Stop
- Extract from Register I
- Jump Unconditional
- Jump Conditional

These operations are different from the set we used in class because the processor is much more limited and must be able to handle the hardware requirements of the system.

4.6 Switching Network

Th ere is a complex logic network inside of this component which is not described in the literature. Its purpose is to determine which signals should be routed where depending on the instruction, state of the bistable devices and location in the sequence. This is somewhat analogous to the ALU and write to register section of our previous CPU. This is the part of the system which determines whether the signal is passed through the system or modified by the instruction or user input. It also allows for the tag bits to be changed when necessary. This is useful because the program counter is included as a tag bit in the instruction register. We can update the program counter, in the case of a jump, without modifying any of the other registers.

4.7 Programming

The CPU can be operated manually, from an inputted program or from a magnetic card. There are switches to change between the modes of operation. Unlike the MIPS architecture, there is no operation to add an integer or write an integer to a register. If a constant is needed, it must be entered manually before the program is run. There are several keypads on the computer which correspond the the available registers and functions.

5 Conclusion

This project was very interesting for us for several reasons. We were able to look at a processor with a different architecture and analyze the differences. Through this we could better understand some of the benefits to the modern architecture, such as random access memory and more operations. While this processor is clearly no longer a viable option, there were many lessons to be learned from it. The Programa 101 used many creative strategies to reduce size including reusing already built components and eliminating vacuum tubes or transistors wherever possible. While implementing our Verilog code, we also tried to reduce the number of gates and transistors used, although this was not always possible. The next step could be to implement this whole processor in Verilog, although we doubt the exercise would be helpful, as the architecture is obsolete. To improve our system it would be better to attempt to build and test an actual delay line. Ours is simulated using behavioral Verilog

but it would be interesting to see how phase errors and timing issues increase the difficulty of the problem.

6 Waveforms

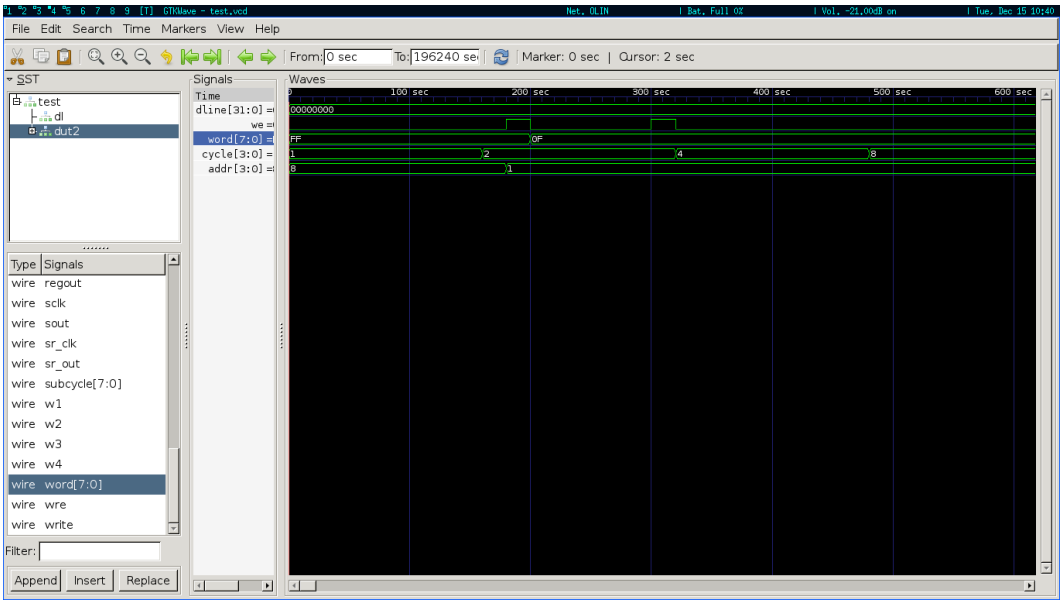

Figure 3: A write attempt is detected.

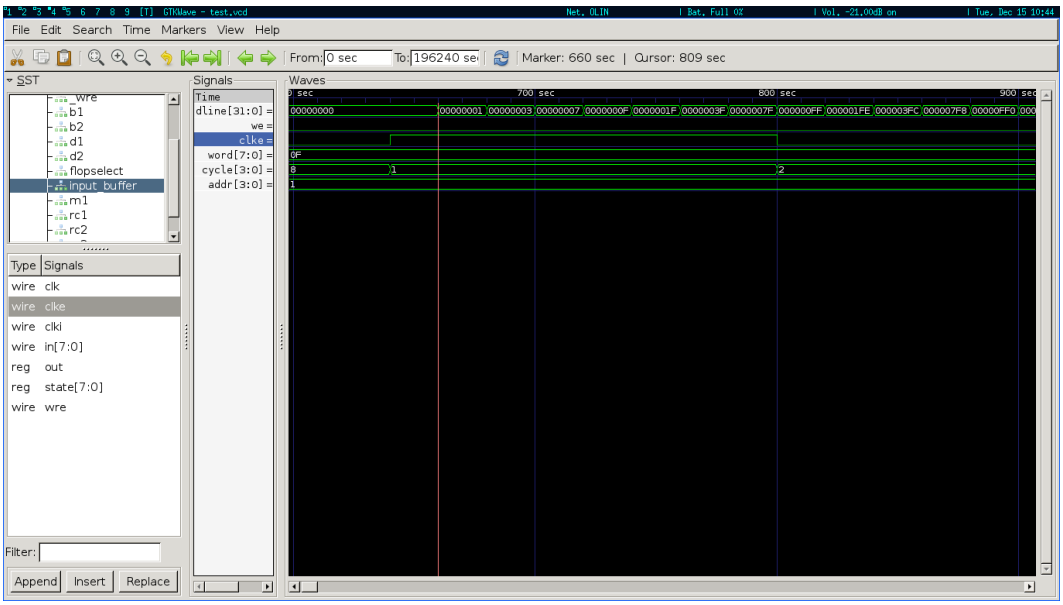

Figure 4: The queued write is performed.

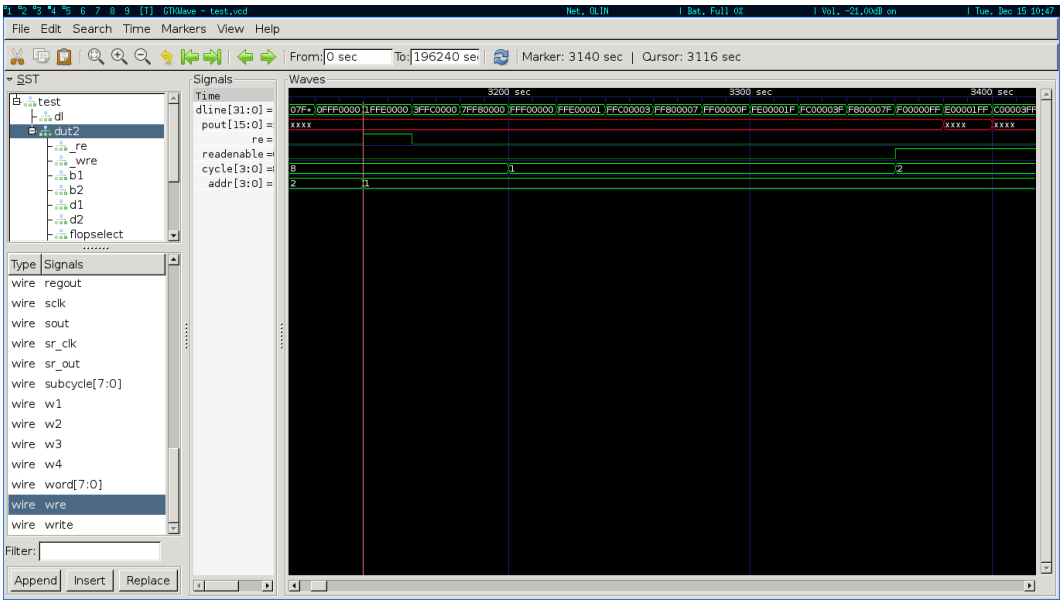

Figure 5: A read is registered.

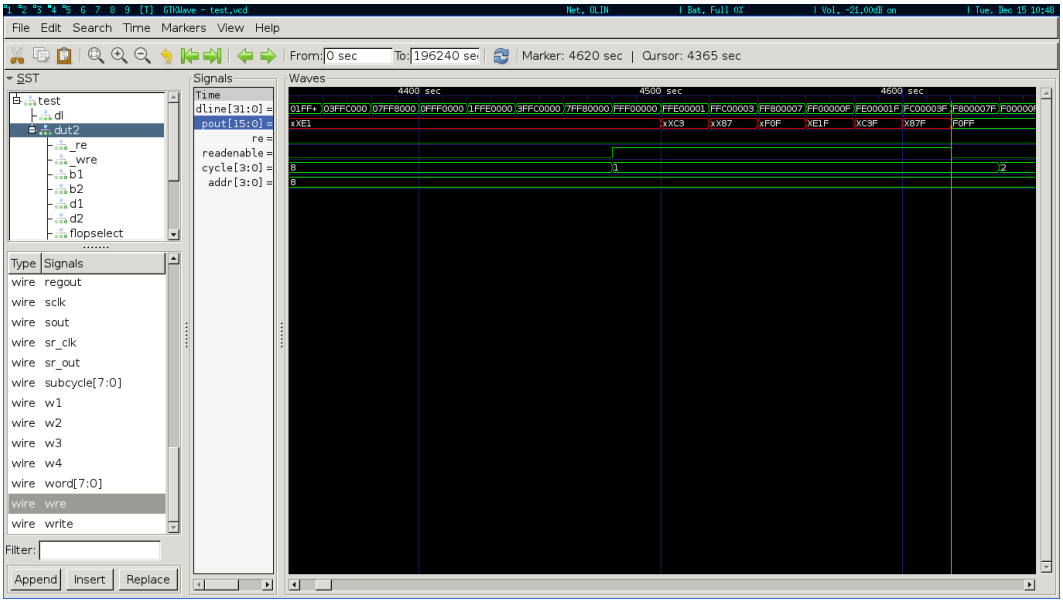

Figure 6: Both words successfully buffered as output.

7 References

- https://www.ieee.org/documents/eckert.pdf
- http://hackaday.com/2015/12/08/retrotechtacular-a-desktop-computer-from-1965/
- http://www.google.com/patents/US3495222
- http://www.classiccmp.org/dunfield/calc/olpro101.pdf
- http://www.oldcalculatormuseum.com/friden130.html
- http://www.vintagecalculators.com/html/friden_ec-130.html
- http://www.sunburst-design.com/papers/CummingsSNUG2002Boston_NBAwithDelays.pdf
- http://ed-thelen.org/comp-hist/vs-univac-mercury-memory.html
- http://www.ece.ucsb.edu/~strukov/ece594BWinter2011/optical.pdf
- https://en.wikipedia.org/wiki/Delay_line_memory