University of Southampton

Faculty of Physical Sciences and Engineering
School of Electronics and Computer Science

Mercury Delay Line Emulation: Reconstructing Memory for EDSAC

by

Joshua Tyler
August 23, 2017

A dissertation submitted in partial fulfilment of the degree of
MSc Embedded Systems

Abstract

Contents

List of Acronyms

AC alternating current.

ARM advanced RISC machine.

CPLD complex programmable logic device.

DC direct current.

EDSAC electronic delay storage automatic calculator.

ENIAC electronic numerical integrator and computer.

FIFO first in, first out.

FPGA field programmable gate array.

GPIO general purpose input/output.

IC integrated circuit.

LC logic cell.

LUT look-up table.

PCB printed circuit board.

PLL phase locked loop.

RAM random access memory.

RC resistor-capacitor.

TQFP thin quad flat package.

VHDL VHSIC hardware definition language.

List of Figures

List of Tables

Chapter 1

INTRODUCTION

The National Museum of Computing is currently hosting a project to reconstruct a very early computer: electronic delay storage automatic calculator (EDSAC), pictured with two of its creators in Figure 1.1 [3]. EDSAC was the first practical digital stored program computer. This means that it was the first practical computer able to accept a program from the user, store it in memory, and execute it on the fly. In contrast to this earlier computers, such as electronic numerical integrator and computer (ENIAC), used programs hard-coded using switches, in the case of ENIAC using 3,600 ten 10-way switches [4]. The only digital stored program computer earlier than EDSAC wa the Manchester small-scale experimental machine. This machine was not intended for general purpose computation however, but rather for testing of a new type of memory [5].

Figure 1.1: W.Renwick and M.Wilkes standing with EDSAC [1]

EDSAC ran its first program in May 1949. This posed a significant design challenge for the memory of the machine. Transistors were not commercially available at all until 1951 [6], and valves, whilst available, were physically large, and were fairly expensive. This meant that creation of even a modest amount of storage would not have been feasible. ENIAC used valves, but it also had very little memory. This was not a problem for its intended application, but would have posed a problem for a general purpose computing platform, such as EDSAC [7, p.208].

The solution chosen for EDSAC's storage problem was delay line memory. This was common with other early computers, and works via a fairly simple mechanism. Given a medium able to delay a pulse train by a certain amount, memory can be created by feeding the output of that delay medium back into the input. If the delay time is tuned to be an integer multiple of the system clock frequency, the system is able to store a sequence of bits proportional in length to the delay time. This principle is illustrated in Figure 1.2.

Delay lines exist in various forms, from magnetorestrictive delay lines which function by twisting one end of a coil of wire, then waiting for the stress to propagate to the other end of the wire, to electric delay lines which provide much smaller delays by sending electrical impulses down a length of coaxial wire or a printed circuit board (PCB) microstrip trace.

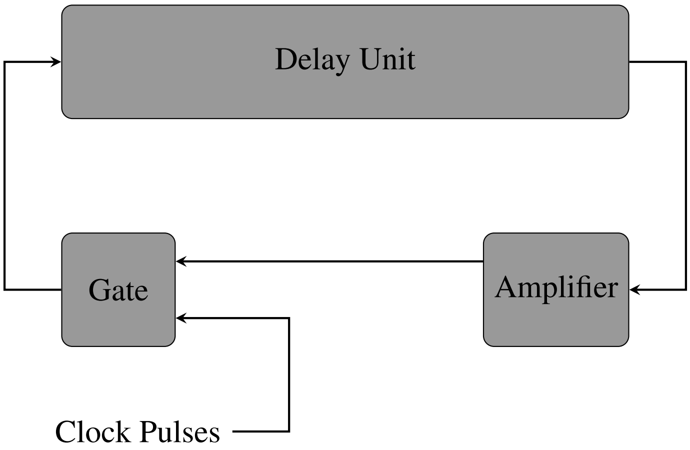

Figure 1.2: Demonstration of the principle of delay line memory, adapted from [?]

EDSAC used acoustic delay lines in the form of steel tubes, approximately 1.5 m long, and full of mercury. Impulses were inserted into one end of the tube via a quartz transducer, and reach the other end of the tube after a delay of approximately 1 ms. Here a second quartz transducer converts the incoming acoustic impulse into an electronic impulse.

Creating a faithful reproduction of this system poses challenges, namely: the expense of mercury, the health and safety implications of using mercury in a museum environment, and the technical challenges of the precise machining necessary for the steel tubes. As a result of this the reconstruction project currently intends to use magnetorestrictive delay lines [8].

As discussed at length in [?], this method is non-ideal since it is: anachronistic for the time, dissimilar in appearance to the original delay lines, and dissimilar in terms of electrical interface to the original. For this reason it was decided to investigate the use of modern technology to emulate the original delay lines. This emulation is required to be indistinguishable from the original in terms of appearance and electrical interface. The design of this system is the goal of this project.

Chapter 2

TECHNOLOGY REVIEW

This Chapter presents a review of the relevant literature surrounding the delay lines used in EDSAC, and uses it to derive a specification for the recreated delay line. Much of the literature presented comes from original documentation produced by Maurice Wilkes, the man in charge of the original EDSAC project.

2.1 EDSAC Delay Line Specification

As briefly discussed in Chapter 1, EDSAC uses mercury delay lines as memory. These are used in two ways:

1. Short tubes used for the results of calculations.
2. Batteries of longer tubes used as the main memory store. This is analogous to random access memory (RAM) in a modern computer.

EDSAC stores words in units of 36 pulses, which is referred to as a minor cycle. 34 pulses are used to store the magnitude of a number, one stores the sign, and one acts as a space between numbers. This system was chosen to allow storage of ten digit numbers.

The shorter tubes are sized to store only a single minor cycle, but the longer tubes are long enough to store 576 pulses (16 minor cycles). The batteries of each of these tubes each contained 16 tubes, and EDSAC originally had two batteries, allowing for a total storage capacity of $16 \times 16 \times 2 = 512$ numbers.

2.1.1 Timing

The memory in EDSAC uses a circulating bit rate of 500 kHz. This is made up of a 0.9 µs pulse, and a 1.1 µs gap for each bit. The pulse is a burst of 13.5 MHz carrier frequency if the bit is a logical 1, or it is 0V if the bit is a logical 0. It should be noted that the reconstruction will use a bit rate slightly faster or slower than 500 kHz, because 500 kHz is an international distress frequency, and given the large wiring looms in EDSAC, it would be quite easy for EDSAC to become an unintentional transmitter of this frequency.

In the regeneration portion of the circuitry, the pulses are demodulated from the 13.5 MHz carrier and stretched to approximately 1.9 µs long, i.e. just long enough that each pulse fails to overlap it's neighbour [7, p.212].

This is critical because it means that the pulses that have passed through the delay line are effectively resynchronised to the system clock. If the system did not work in this way, then the delay lines in the the original EDSAC would never have worked. The propagation time of acoustic waves through mercury varies slightly with temperature, and the cumulative effect of this through all the delay lines of the store would mean that pulses would be unlikely to align with the system clock at the other end.

Despite this resynchronisation after each delay line, the original EDSAC suffered a great deal from the variation of delay with temperature. Originally the system clock was adjusted whenever the temperature of the delay lines drifted out of synchronization, but later on in development the delay lines were put inside a temperature controlled oven.

The regeneration system implies that the maximum acceptable skew of the delay line from its nominal delay is $\pm$ 0.5 µs. This does not, however, take into account other factors such as the jitter and longer term drift of the system clock, as well the slew rates of the analogue circuitry. Whilst the demodulating pulse is lengthened to 1.9 µs, it is unlikely to be consistently at it's peak voltage for this time, and so the output pulse is likely to have a better shape if the delay line produces an output in the middle of this period.

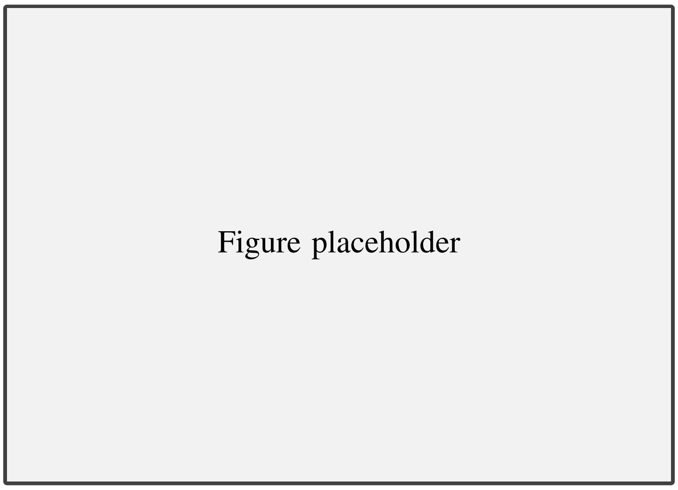

Figure 2.1: EDSAC pulse timing

2.1.2 Electrical

Electrically speaking, EDSAC originally drove the delay lines with a nominal voltage of 25 V peak, through a 70 Ω terminated transmission line. The loss in the delay lines was 69 dB, leading to an output voltage of approximately 10 mV.

Despite this, the recreation project has discovered that the regeneration circuitry actually feed the delay lines with a decreasing voltage as the pulses propagate through the lines. The voltage starts off at approximately 35 V for a signal fed to a delay line at the start of the store, but is reduced to approximately 25 V peak for the tubes at the end of the store.

In addition to this, problems were experienced with amplifying the low signal level output by the delay lines. Because the current wire delay line solution has flexibility in its output voltage, currently the delayed signal is output at 100 mV peak.

The 75 Ω transmission lines are alternating current (AC) coupled to the main EDSAC chassis, and the shield of the coaxial cable is referenced directly to earth. This poses a small problem for the reconstruction, because modern health and safety regulations dictate that the chassis of EDSAC should be earthed, but the electronics themselves are powered from an isolated 0 V rail with earth leakage protection in place.

This wouldn't be a problem except for the fact that the coaxial shield is earth referenced, meaning the coaxial return current goes to earth, not the isolated 0 V rail. In order to combat this, several capacitors have been added between ground and the 0 V rail, in order to allow the AC current to pass, but still provide direct current (DC) isolation. This means that care will be necessary when implementing the circuitry to power the delay line, as it cannot draw any DC power from the line.

2.1.3 Mechanical

The store delay lines originally consisted of banks of steel tubes, each tube having an outer diameter of 4.44 cm, and an inner diamter of 2.86 cm [7, p. 213]. The tubes are then held in an array using machined end-plates. An illustration of this is shown in Figure 2.2.

The main implication of these dimensions for the reconstructed system is that the electronics must be more narrow than the inner diameter of the tube, so that it can fit inside.

Figure 2.2: A battery of mercury delay lines in EDSAC [2]

Chapter 3

SPECIFICATION

The research of Chapter 2 has led to the derivation of a specification for the delay line I will produce. This specification is detailed in Table 3.1.

Table 3.1: Delay line specification

Item	Specification	Justification
1	**Must** be capable of producing a delayed copy of the EDSAC pulse train presented to it's input	This is the primary function of the device.
2	**Must** be powered from the input signal driven by EDSAC, with only minimal non-intrusive modificaions made to EDSAC.	This is the primary function of the device.
3	**Must** be able to have an adjustable nominal delay in the range of **[xx]** to **[xx]** .	An adjustable delay allows synchronisation with the system clock, with may vary.
4	**Must** have a skew against the nominal delay of no greater than **[xx]** , and a maximum per cycle jitter of. **[xx]**	This ensures that the delay line output will be able to synchronise with the clock of EDSAC.
5	**Must** be able to interface with input waveforms AC coupled bursts of 13.5 MHz carrier, with peak voltages in the range of **[xx]** to **[xx]** .	This is necessary to mimic the performance of the original delay line.
6	**Must** be able to have an adjustable nominal output voltage in the range of **[xx]** to **[xx]** , driving into 70 Ω.	An adjustable output voltage in this range allows compatibility with both the original electrical interface, and that used by the reconstruction effort.
7	**Must** be able to fit inside a tube of **[xx]** internal diameter.	This diameter allows the design to fit inside the steel tubes originally filled with mercury.
8	**Must** be accompanied by a testing device capable of emulating the signals produced by EDSAC.	This allows the delay line to be tested separately to the reconstruction project.

Chapter 4

DELAY LINE DEVELOPMENT

This Chapter describes the development of the delay line itself, covering the architectural choices made, as well as the detailed design, development and testing. The overall block diagram of the delay line is shown in Figure 4.1.

The delay itself is implemented in the digital domain, with the goal of accurately emulating the analogue domain of the original design. This development is detailed in Section 4.1. The surrounding analogue circuitry translates between the logic level signal signals of the digital circuit, and the input and output.

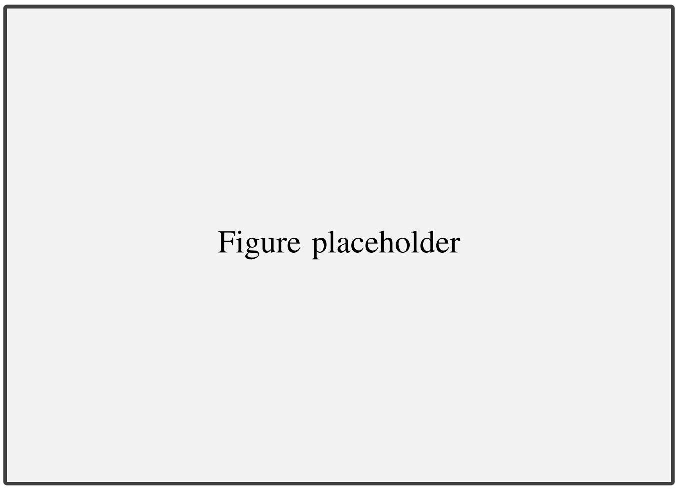

Figure 4.1: The overall architecture of the delay line

4.1 Digital Design

The goal of the digital design is to delay the signal on it's input by a certain amount. If the input signal were arbitrary, then the optimal solution would be to clock a 1 b wide first in, first out (FIFO) buffer with a sampling clock. The delay would then be given by Equation 4.1, where t_d represents the delay time, f_s sampling clock frequency, and N the depth of the buffer.

$$t_d = \frac{1}{f_s} \times N \tag{4.1}$$

This solution can be simplified by consideration of the fact that the input signal is not arbitrary, the characteristics or the signal are known from the research detailed in Section 2.1.1. It is known that:

- The signal will consist of 0.9 µs pulses of 13.5 MHz tone.
- Each tone burst will be separated by 1.1 µs.
- Each delay line can store a maximum of 576 pulses.

Using these characteristics it can be seen that a digital delay line only needs to sample store the time at which the rising edge of an incoming pulse is received. Since the packet length and modulation frequency is fixed, this can be asserted on the line a fixed delay later.

4.1.1 Architecture selection

Various architectures could be used to implement the system described in the previous section. The three most obvious methods of implementation being

- A microcontroller design.
- A discrete logic design
- A field programmable gate array (FPGA) design

Microcontrollers have the advantage of being comparatively cheap and readily available, in addition they typically have more than enough RAM available to implement the memory to store the array of times at which pulses arrived.

The principle disadvantage is that microcontrollers inherently to their architecture process a single thread of data at once. This means that even a tight processing loop which samples the input would add a much larger amount of jitter to the input compared to a hardware solution with the same clock rate. Fortunately however modern microcontroller architectures, typically have a large number of peripherals embedded in the silicon, which can remove computation from the core. This combined with interrupts can provide an architecture with very predictable latency.

An example implementation is detailed in Figure 4.2. This details a system, based upon a typical microcontroller in the advanced RISC machine (ARM) Cortex-M0 family. A pin change interrupt interrupts the main execution thread, and branches to an interrupt handler. This interrupt handler reads the value of a free-running timer peripheral that counts up using the microcontroller master clock, adds a fixed delay value to it, and appends it to a queue of timer counts held in RAM. This queue contains the values of the counter which the output should be asserted at.

The main execution thread of the microcontroller configures the timer peripheral to trigger a second timer peripheral once its count equals the value on the top of the queue. This second counter is connected directly to an output general purpose input/output (GPIO) pin, and is clocked such that the output toggles at 13.5 MHz.

This system can therefore meet the requirements of the delay line, with a worst case jitter of one system clock period (since the Cortex-M0 family allows for deterministic latency interrupts **[cite]**). Microcontrollers with timers capable of being configured as described above are readily available also **[cite STM32 reference manual here]** .

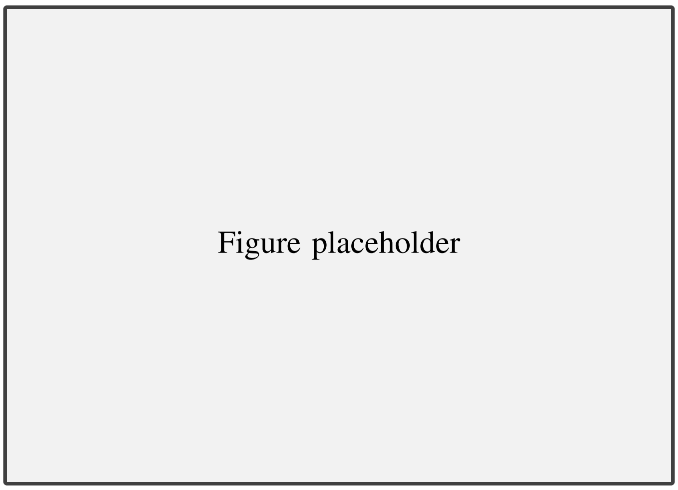

Figure 4.2: Proposed architecture for a microcontroller based system

The other two alternatives, a discrete logic system, and a FPGA based system would be similar in architecture, but differ in implementation, with the FPGA based system implementing the function using the programmable fabric of a FPGA, whereas the discrete implementation would use individual integrated circuits (ICs). A block diagram of the proposed architecture is shown in Figure 4.3.

The concept of this architecture is similar to that of the microcontroller based system. There is a free running counter inside the FPGA, when a pulse is detected on the input, the value of the counter, added to a fixed delay is saved into a FIFO. A second hardware module outputs a pulse train whenever the value of the counter matches the value on the top of the FIFO.

The difference between the hardware implementation and the microcontroller implementation is that there isn't a processor core to set up the hardware blocks, instead each hardware block is designed to perform the correct function, and interacts with the other modules using logic signals. In addition there is no need for an external modulator, as a harware block can be created to output the modulated 13.5 MHz signal.

The FIFO is the centre of this system. It is set to be the width of the free-running counter. At its input is the current value of the counter, added to a fixed constant that represents the required delay. This value is saved into the FIFO when the rising edge detector produces a pulse on its output.

The rising edge detector is a fairly simple hardware block that outputs a pulse for a single clock cycle when it detects a rising edge on its input. It then times out for a fixed interval, to avoid triggering on the remaining pulses in the same packet, and resets.

At the output of the FIFO feeds into a comparator. This comparator compares the value on the top of the FIFO with the current value of the counter, and triggers the pulse generator when the two values are equal.

The pulse generator outputs a fixed length burst of 13.5 MHz tone whenever it is triggered. This could be implemented simply by choosing a clock frequency that is 2^n times greater than 13.5 MHz. Thus the carrier frequency can be generated by a counter clocked from the input clock.

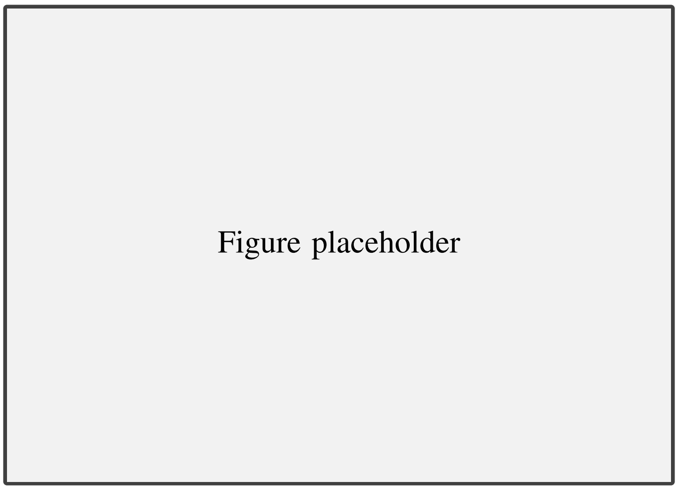

Figure 4.3: Proposed architecture for a FPGA or discrete system

[Could add power in here]

The advantage of the microcontroller system over a FPGA or discrete solution is twofold. Firstly the simplicity of the physical circuit necessary. A microcontroller based design would require a IC for the microcontroller itself, and an external crystal oscillator (although some microcontrollers do have a reduced precision internal resistor-capacitor (RC) oscillator, removing the need even for this), and are generally powered from a single 3.3 V supply. This is in contrast to FPGAs which typically require more than one supply rail, with extensive decoupling, and a discrete solution which would require many ICs.

Secondly a microcontroller design would likely be slightly cheaper than a discrete logic design, as low end microcontrollers are typically only slightly more expensive than individual logic ICs, and are much less costly than typical FPGAs.

Despite these advantages, a hardware solution does have the advantage of elegance. While it has been demonstrated above that a microcontroller solution could achieve single cycle jitter – the same as a hardware solution. It is a lot more difficult to implement and verify, given the fact that configuration of many hardware peripherals are required.

In addition, the cost advantage of the microcontroller solution may be smaller than anticipated. This is because the simplicity of the hardware solution would only require a very small FPGA, which may be competitive in price to a microcontroller. Alternatively a complex programmable logic device (CPLD) with an external RAM IC could be used. A FPGA solution would be preferred to a discrete implementation, due to the ease of testing, and reconfiguring the hardware if the requirements change.

Therefore on balance it has been decided to proceed with a FPGA based solution.

FPGA Selection

The primary requirements for the FPGA are as described in Table 4.1.

Table 4.1: FPGA Requirements

Number	Requirement	Justification
1	**Must** have a moderately low cost	Using the IC for every delay line in the store must not be cost prohibitive.
2	**Must** have enough logic elements and block RAM to implement a single long delay line	This is the proposed function of the delay line
3	**Must** have fabric capable of being clocked fast enough to implement the design with a clock rate of **[xx]**	It was decided at **[xx]** that this is the required sampling speed.
4	**Should** have a development board available with a width less than **[xx]**	As described in Section **[xx]**, **[xx]** is the internal diameter of the delay line tube, it would be ideal if the development board could fit inside the tube, to save a custom PCB being designed.
5	**Should** have a phase locked loop (PLL) to enable to fabric clock to be generated from a crystal oscillator	An internal PLL is ideal, but an external PLL could also be used.

At the time of writing the least expensive FPGA available from component suppliers Farnell, is the a 1280 logic cell variant of the iCE40 FPGA family, produced by Lattice [9]. This is £5.10 in single quantity, which is low cost enough that it could reasonably be used to replace all of the delay lines, thus meeting specification point 1.

This FPGA has 1280 logic cells (LCs), which should be plenty to implement the simple design, given that each LC in this architecture consists of a four input look-up table (LUT), and a flip-flop [10, p.2-2]. The FPGA also has 64 kb of block RAM. This meets specification point 2, when one considers that is enough to store a clock sample for each of the 576 possible pulses, even if a 64 b clock width was used, as demonstrated by Equation 4.2, where S representes the size of memory required. In addition to this, there is one PLL available on the FPGA die, meeting specification point 5.

$$S = 64\,\mathrm{b} \times 576 = 36\,\mathrm{kb} \tag{4.2}$$

It is hard to estimate how fast a design will be able to operate in a FPGA without synthesising it, and running timing analysis. However, despite this we can estimate that the FPGA will easily meet timing at **[xx]** , thus meeting specification point 3, given that the register-to-register performance of the fabric is as good as 403 MHz for a dual-port RAM, 305 MHz for a 16:1 multiplexer, and 105 MHz for a 64 b counter.

In addition to this, a development board is available which is very narrow [11]. The exact dimensions are not provided, but it is barely wider than the 22 mm thin quad flat package (TQFP) of the FPGA itself [11, p.2], meaning it is highly likely to fit in the **[xx]** diameter tube, thus meeting specification point 4.

Based upon the fact that it meets all of the specification points, it has been decided to implement the design using the Lattice iCEstick evaluation board, with the possibility of moving to a custom PCB if many instances of the design were required.

4.1.2 HDL Design

Uniquely, the Lattice iCE40 family of FPGAs has an open-source toolchain available, named Project IceStorm which can be used as an alternative to the toolchain provided by Lattice [12]. Project IceStorm synthesises Verilog natively, and Lattice's iCEcube2 toolchain synthesises both Verilog and VHSIC hardware definition language (VHDL) [13, p.10]. Therefore in order to maintain compatibility with both toolchains, the design will be written using Verilog. Both toolchains were trialled, and the codebase is compatible with both, however Project IceStorm was used in the end as it is the more user-friendly toolchain.

The design was implemented using the structure of Figure 4.3, with the rising edge detector, FIFO, comparator, and pulse generator implemented as individual Verilog modules.

[Could talk about how the parameters are determined here]

Rising Edge Detector

The rising edge detector is implemented as the state machine of Figure 4.4. In the wait state, the input is sampled on every clock edge, when it is true, the state machine transitions to the assert state, where the output is asserted, until the state machine unconditionally branches to the timeout state. In the timeout state a counter is incremented until it reaches the required limit. At this point the state machine branches back to the wait state.

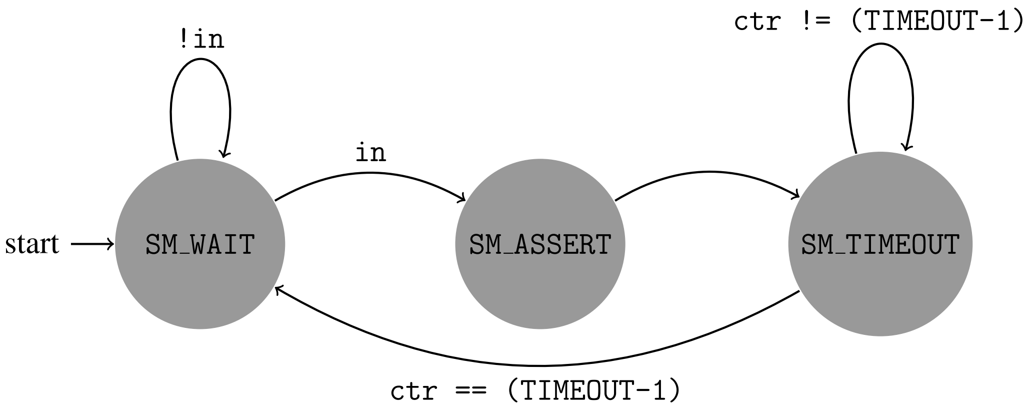

Figure 4.4: Rising Edge Detector State Machine

FIFO

The FIFO is implemented by building logic constructs around a dual port RAM to handle addressing of the read and write ports.

The main part of the FIFO is the read and write addresses. These addresses act as pointers to the current location of the read/write word in RAM. Each address is incremented on when the FIFO it read from/written to, and the address loops around to the start when it reaches the end of the RAM buffer. This 'round robin' approach means that the FIFO can be read from and written to an unlimited amount of times, so long as the total number of words stored is not greater than the depth of the buffer.

In addition to the read/write address pointers, there is a counter which keeps track of the total number of words stored in the buffer. This counter is decremented if a read is requested, and incremented if a write is requested (its value does not change if both a read and a write is requested at the same time, or neither a read or write is requested). This counter is used to generate empty and full signals so that the surrounding logic knows the state of the counter.

Comparator

The comparator is implemented as a state machine that requests data from the FIFO and asserts the output when the counter matches the data word, as illustrated by the state transition diagram of Figure 4.5.

`empty` is the `empty` signal from the FIFO, and the FIFO read request signal is true when the state machine is in the request state.

`count` is the value of the system counter, and `data_in` is the data read from the FIFO. The output trigger is true when the state machine is in the `SM_ASSERT` state.

4.1.3 HDL Verification

4.2 Method of Powering

4.3 Analogue Design

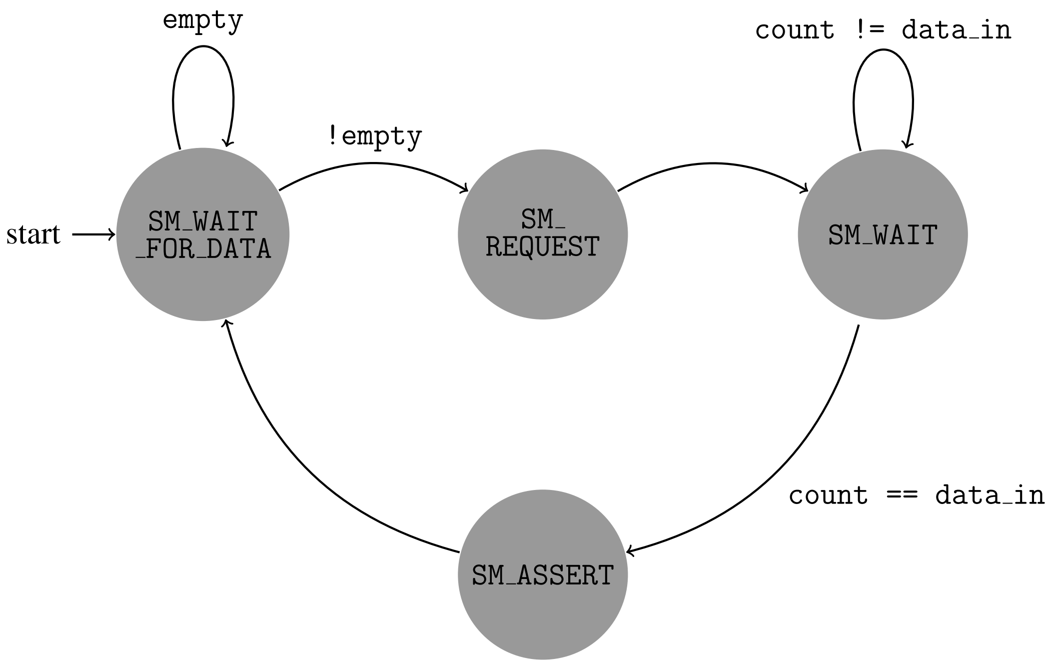

Figure 4.5: Comparator State Machine

Chapter 5

TEST HARNESS DEVELOPMENT

Chapter 6

INTEGRATION AND TESTING

Chapter 7

PROJECT PLANNING

Chapter 8

CONCLUSION

Bibliography

[1] University of Cambridge. (2011, feb) EDSAC I, W.Renwick, M.Wilkes. Copyright Computer Laboratory, University of Cambridge. Reproduced by permission. Cropped from original photo. [Online]. Available: http://www.cl.cam.ac.uk/relics/jpegs/edsac_wilkes.jpg

[2] ——. (2011, feb) M.V.Wilkes, mercury delay line (2). Copyright Computer Laboratory, University of Cambridge. Reproduced by permission. [Online]. Available: http://www.cl.cam.ac.uk/relics/jpegs/delay_lines.jpg

[3] ——, "Pioneer computer to be rebuilt," *CAM*, vol. 62, p. 5, mar 2011.

[4] F. da Cruz. (2013, nov) Programming the ENIAC. Columbia University. [Online]. Available: http://www.columbia.edu/cu/computinghistory/eniac.html

[5] K. S. Jones. (2001, dec) A brief informal history of the computer laboratory. University of Cambridge. [Online]. Available: http://www.cl.cam.ac.uk/events/EDSAC99/history.html

[6] J. Bonne. (2007, march) Transistor transition began in Allentown. The Morning Call. [Online]. Available: http://articles.mcall.com/2007-03-04/features/3712894_1_transistor-production-line-bell-labs-allentown-plant

[7] M. V. Wilkes and W. Renwick, "An ultrasonic memory unit for the EDSAC," *Electronic Engineering*, vol. 20, pp. 208–213, jul 1948.

[8] M. Ward. (2011, jan) Pioneering edsac computer to be built at bletchley park. BBC News. [Online]. Available: http://www.bbc.co.uk/news/technology-12181153

[9] Farnell. (2017, aug) ICE40HX1K-TQ144 -FPGA, iCE40, PLL, 95 I/O's, 133 MHz, 1.14 V to 1.26 V, TQFP-144. [Online]. Available: http://uk.farnell.com/lattice-semiconductor/ice40hx1k-tq144/fpga-1280-luts-1-2v-hx-144tqfp/dp/2362849

[10] *iCE40™LP/HX Family Data Sheet*, 3rd ed., Lattice, mar 2017.

[11] *iCEstick Evaluation Kit*, 1st ed., Lattice, aug 2013.

[12] C. Wolf and M. Lasser, "Project icestorm," http://www.clifford.at/icestorm/.

[13] *iCEcube2 User Guide*, Lattice, feb 2017.

Appendix A

Foo

Bar