Reimagining EDSAC in open source: A valve computer reimplemented using FPGAs, Arduinos, 3D printing and discrete electronics

Slide 1: Title
(Up during setup and heralding)

Slide 2: Overview
(new slide, picture with EDSAC, Wilkes and our reimagining)

Good afternoon

Today I am going to tell you about one of the earliest general purpose computers, EDSAC, its creator Maurice Wilkes, and our ongoing project to reimagine EDSAC using modern free and open source technology.

Our goal is to introduce school students, who will be the engineers of the future, to the reality of a computer which shaped modern society. Pure software simulations of EDSAC have existed for many years. Crucially our project is about recreating a physical version of EDSAC. One where connections can come loose, where paper tape jams and sometimes mis-reads, where initial orders can be mis-entered and where teleprinters can run out of paper at just the wrong time.

There will be demonstrations. With these I'll be helped with these by my glamorous assistant, Jeremy.

(Jeremy takes a bow)

These should work perfectly, due to my hard work, but of course if anything goes wrong, it will be Jeremy's fault.

Slide 3: EDSAC
(existing slide 7, retitled)

EDSAC was one of the first general purpose computers to be built and used. It represented the beginning of computing being used to share and develop ideas from a large group of people. This is why EDSAC was chosen to be explored during the open source hardware tutorial that was run in the summer of 2017.

Plus the Computer Conservation Society was also building a replica which shall be open to the public at Bletchley, England very soon.

Slide 3: Pre-EDSAC Timeline
Computers have not always been around, nor have the basic principles of computing that we take for granted today.

Computing in the 1940's was changing from electromechanical machines using switches and relays to electronic computation using valves. Relays were slow and temperamental and so could not be used for general purpose machines although a few valiant attempts have been made to prove otherwise.
All computers used decimal systems. Von Neumann was the person that suggested 2's-complements use in his proposal for EDVAC. Computers also did not store programs in memory. To change the tasks in some computers the engineers had to change the internal wiring.

Slide: ENIAC
EDSAC's grandfather was called ENIAC. This computer was notable for being one of the first totally electronic, valve driven, digital, program-controlled computers. It was unveiled in February 1946. It was moved to Maryland where it grew up, by getting a memory upgrade, married and became a proud father and called his son EDVAC.

Slide: EDVAC
John Von Neumann designed EDVAC in 1945 but it was only finished in 1951. The main difference between ENIAC and EDVAC was that ENIAC used a decimal system whereas EDVAC was binary, one of the traits inherited by EDSAC.

Slide 4: Wilkes
Maurice Wilkes was born in England 1913. He studied maths and physics at the University of Cambridge from 1931 to 1934 and completed a PhD in physics in 1936. In 1945, Wilkes was appointed as the second director of the University of Cambridge Mathematical Laboratory later known as the Computer Lab. The Computer Lab initially had many different computing devices, including a differential analyser, but these were often too difficult or too niche to use.

Slide 5: Wilkes and EDSAC
One day Leslie Comrie visited Wilkes and lent him a copy of John von Neumann's prepress description of EDVAC under construction at the Moore School of Electrical Engineering. This report includes the first published description of the design of a stored-program computer, giving rise to the term von Neumann architecture. Comrie was leaving Cambridge the next day so Wilkes had to read it overnight because there were no photocopying machines in those days. Wilkes decided immediately that the document described the logical design of future computing machines, and that he wanted to be involved in the design and construction of such machines.
In August 1946, Wilkes travelled by ship to the United States to enroll in the Moore School Lectures. He was only able to attend the final two weeks because of various travel delays.

During the return voyage to England, Wilkes sketched out in some detail the logical structure of the machine which would become EDSAC.

He immediately started work on a small practical machine once back at Cambridge. Wilkes decided that his mandate was not to invent a better computer, but simply to make one available to the university. Therefore, his approach was relentlessly practical. He used only proven methods for constructing each part of EDSAC. The resulting computer was slower and smaller than other planned contemporary computers. However, EDSAC was operated successfully from May 1949, well over a year before the much larger and more complex EDVAC.

Slide 6: About EDSAC: General
EDSAC used delay lines for memory and vacuum tubes for logic. Thanks to these 3000 odd vacuum tubes and the 5 foot or longer delay lines, EDSAC could just about fit into a 5 meter by 4 meter room. But EDSAC couldn't store that much information in the beginning, the available main memory was only 512 18-bit words.

Slide 7: Punched tape
Punched tape was a popular way to load programs into older computers. To make the punched tape, a large, typewriter controlled tape punch would be operated by a trained typist.

For an EDSAC programmer, their preliminary program would have been given to the typist who converted the program into standard Baudot code and punched the five hole wide tape. The tape would have been handed back to them. The punching machine was on a floor below EDSAC so the programmers wouldn't have too far to travel with their precious tape before handing to one of the computer engineers.

The computer engineer would then hang the tape up until it was its turn to be read into EDSAC. If there were errors in the tape, a hand punch could be used to do small corrections.

Slide: Teleprinter
If the program instructed a printed output, the teleprinter would be used. Teleprinters were electro-mechanical, typewriter-like machines. First EDSAC would send the least significant bit of the word being printed to an internal buffer. When a new character was received in the internal buffer, the old character would be printed.

Slide: Cycle time
Most components in EDSAC were clocked using a square wave that was generated from a sinusoidal signal with the wanted frequency and a very large amplitude. The signals would hit the positive and negative voltage rails of the circuit behind me, giving the resulting waves a square like appearance.

There are some remaining drawings of EDSAC but even these are sometimes incomplete. Sometimes something needed an extra tweak hear or an extra wire there to make the circuit

work. These changes were not noted down. But Wilkes was incredibly neat with his drawings. I don’t think my circuit diagrams would ever be that neat even if I tried.

Cycle time was 1.5 milliseconds for all ordinary instructions, such as addition, and about 6 milliseconds for multiplication. To put that into perspective, we currently use MIPS to measure the amount of millions of instructions per second. This means that it is normal to take hundreds or tens of nanoseconds per instruction.

Slide: Debuggers
Debuggers were some time away, but one of the cathode ray tube screen could be set to display the contents of a particular piece of memory. This was used to see if a number was converging, for example. A loudspeaker was connected to the accumulator's sign bit; experienced users knew healthy and unhealthy sounds of programs, particularly programs 'hung' in a loop.

Slide 9: A night (Story)
When EDSAC broke down during the day, the engineers would fix it.
After office hours certain "Authorised Users" were allowed to run the machine for themselves, which went on late into the night until a valve blew. They would then have to turn off, go home and wait till late the next morning for EDSAC to work again.

There are a few known instances where EDSAC worked through the night. One such night happen in March 1950. D H Shinn worked through the night and said that when the engineers arrived in the morning they were surprised and delighted to find that they did not have to go through the time-consuming commissioning and testing routine; they just had to wait until the next breakdown! Shinn was the only programmer left because all the others had gone home to have their breakfast and a good sleep.

Slide 10: Instructions
Due to timing issues, the top most bit in every word was always unavailable. Short words were 17-bit long. Long words were 35 bits long and consisted of two consecutive little-endian short words, minus the final bit. All instruction codes were by design represented by one mnemonic letter, so that the *Add* instruction, for example, used the EDSAC character code for the letter A.

The EDSAC used two's complement, binary numbers. Unusually, the multiplier was designed to treat numbers as fixed-point fractions in a range between one and minus one. Therefore the binary point was immediately to the right of the sign. The accumulator could hold 71 bits, including the sign. This allowed two long numbers to be multiplied without losing any precision.

There was no division instruction and no way to directly load a number into the accumulator. Instead a "sTore and zero accumulator" instruction followed by an "Add" instruction were necessary for the accumulator. There was no unconditional jump instruction, nor was there a procedure call instruction as it had not yet been invented.

Slide 11: Initial Orders
All computers, including the ones of today, need bootstrapped orders and start up instructions to function. EDSAC required 32 instructions, each instruction consisted of 17-bits. The first order described where in memory the rest should be stored, the first 31 locations.
During the testing period of EDSAC, a large amount of time was wasted manually entering each instruction. Wheeler noticed this problem and decided to fix it. He had the 32 initial instructions and orders were hard-wired on a set of uniselector switches. By May 1949, the initial orders provided a primitive relocating assembler taking advantage of the mnemonic design. This was the world's first assembler.

Slide 12: Subroutine
The subroutine is a concept that many of us use today. But back before 1950, it didn't exist. David Wheeler was the first person to come up with the concept and he implemented it on EDSAC. Users wrote programs that called a routine by jumping to the start of the subroutine with the return address in the accumulator. By convention the subroutine expected this and the first thing it did was to modify its concluding jump instruction to that return address. He called them the Wheeler jump, wonder why.

Multiple and nested subroutines could be called so long as the user knew the length of each one in order to calculate the location to jump to. Recursive calls were forbidden.

To add a subroutine to a paper tape, the user would have to either copy the routine to the end of their tapes or feed it in after their program.

The subroutine concept led to the availability of a substantial subroutine library. By 1951, 87 subroutines were available for general use. These included arithmetic operations on complex numbers, logarithms, and counting operations that simulated repeat until loops, while loops and for loops**.**

Slide 13: Famous programs
Since EDSAC was a general purpose computer, many advances throughout the sciences were assisted by it. The most famous was in 1951 when Wilkes and David Wheeler used EDSAC to solve differential equations relating to gene frequencies.

EDSAC was also used to discover a prime number with 79 digits, the largest known at the time. They discovered it using the 127th Mersenne prime. For those of you who don't know it off the top of your head it is behind me.

The only Nobel prize to be won thanks to the computer was for Nerve cell signalling. Once all numbers and constants had been solved Hodgkin and Huxley hoped to used EDSAC as soon as possible but quickly found out that it was off air for 6 months or so while it underwent some major modifications.

The first graphical video game was written and played on EDSAC. It was tic-tac-toe written by Sandy Douglas for his PhD. The game produced a graphical output sent to a 6 inch

VCR97. It allowed the player to choose who would go first; them or the computer. To enter their next move, the rotary telephone dial was used. The game was also called ‘OXO’. Sandy’s PhD was a success however he would never again program another computer game.

Slide 14: Joke (story)
There are a few short stories that the users of EDSAC remember fondly. Dr David Hartley collected these stories from the early days of the Computer Laboratory.

A group of research students and other EDSAC uses arranged an April Fool’s Day surprise for one of their colleagues, Ken Dodds. They asked the typist to add an extra hole in Dodds paper tape. This hole transferred control to instructions that had not been cleared from the EDSAC store.

On the morning on April first, there was a particularly long queue of programs. One of the research students had to surrendered his place to ensure that the doctored program would run before noon, as was traditional with April Fool’s. When it came to Dodds turn, the tape was read in and EDSAC printed out: “Ease up Ken, I've been working all night. EDSAC.” Poor Dodds was not amused.

Slide 15: Rheostats (story)
Another story is about the surprising effect of EDSAC’s unprotected mains. Jenifer Leech, previously Jenifer Haselgrove, was an authorised user and research student using EDSAC over 1953 to 1956. Since EDSAC’s memory wasn’t very large, some of her programs had ‘pending-put’ tapes, meaning that her program used output tapes with intermediate results on them which were read in soon afterwards as input for the next stage. The tape punch used to miss out a hole occasionally, and when this was detected on input corrections would be made using a hand punch which was kept nearby. The punch had once been chained to a table to stop it wandering, but by then it was loose, with the chain dangling from it.

One day the usual error occurred, time was valuable, Leech seized the hand punch, the chain swung under the table, there was a loud bang, and all the oscilloscope displays went out. Leech can’t remember what she put in the log book.

EDSAC required a specific amount of electricity to run. The amount was controlled by rheostats dotted throughout the Computer Lab. Most of them had slider arrows drawn on them, one pointing up labelled ‘up’ which increase the voltage level and the other pointing down and labelled ‘down’ which decreased the voltage level. One day Jenifer Leech added two new labels; ‘left’ and ‘right’. By the morning they were gone. She admits that perhaps the joke wasn’t all that funny.

On another night, the rheostat adjustment to the mains supply wasn’t enough and Fred Hoyle had to ring up the electricity board and said “Can you hike your volts up a bit?”. It is unknown if they did.

If anyone would like to read more stories about EDSAC, EDSAC 2 or Titan, please go to the link on the screen.

Slide 16: Extension
Throughout the lifetime of EDSAC many additions were made. In 1952, another tank was added which doubled the amount of storage. A year later an index register that was designed by David Wheeler was added to EDSAC increasing the number of registers in EDSAC to 3.

Later, to allow a single digit integer to be input into the program part way through, a dial resembling the dialling system on an old phone was added. In late 1955, a full-word delay line store was added allowing program size to increase to 1024 words.

Slide 17: EDSAC 2
EDSAC 2 was the successor to EDSAC. It was designed by the same team that built EDSAC and came into operation in 1958, the year EDSAC was shut down. EDSAC 2 was the first computer to have a microprogrammed control unit. At a mechanical level, EDSAC 2 was packaged with a bit slice manner with interchangeable plug-in units. EDSAC 2 remained in use until 1965.

Slide 18: Reimagine
There are many different simulators that have been made of EDSAC. But there is nothing that has physical components to run alongside the simulator and further the users knowledge. That is why these 5 peripherals were built.

All peripherals are open source. This is because they were designed to be built by anyone and everyone. Schools are the primary target so everything is as cheap as possible and uses resources that are easy to build or buy. Everyone is welcome to add patches to the current designs as well.

The reimagination is based of the original 1949 version of EDSAC.

Slide 19: FPGA
To connect together all of the peripherals, an FPGA was used. FPGAs are Field Programmable Gate Arrays. This means that they are boards who can change their logic depending on the software put into them. FPGAs are programmed in Verilog or VHTML. In these languages everything happens on a clock cycle and all at once as opposed to sequentially.

The MyStorm board is an open source FPGA designed for education. This means that it is cheap and easy to set up for beginners. Ken Boak and Al Wood designed these boards. The MyStorm uses IceStorm and YOSYS, an open source program written by Clifford Wolf for Verilog synthesis. These boards are completely open source. All board schematics are freely available so if you want to make your own version of a MyStorm you can.

Slide 20: EDSAC Verilog

Bill Purvis and Hatim Kanchwala each created a MyStorm version of EDSAC in Verilog. To do this Bill got his hands on the original logic drawings of Wilkes. In those days, logic gates were so new that there was no universal symbol for an AND gate or an OR gate. Instead Wilkes used lettered boxes. Bill redrew the logic system on ELSIE. He then ran gate level simulations to confirm the design.

It was suggested that Bill use a Verilog implementation. As part of the Google Summer of Code, Hatim used Bill's design. They both hand coded I/O modules. Hatim added in a small section to allow an Arduino or Pi to connect to the MyStorm board.

Let's now look at one of the peripheral devices, our reimagining of the paper tape reader. We don't have access to a paper tape punch—or indeed any old paper tape to punch.

Instead we use a thermal printer, of the type used in a shopping till, to print out paper tape

Slide 35: Reimagined Paper Tape
(Assistant holds up paper tape at same time)
Like traditional paper tape we have 3 holes on one side, then a sprocket hole, then 2 holes on the other side. To read the tape we'll shine a light on the paper and detect how much is reflected back using a light dependent resistor.

Slide 36: Reimagined Paper Tape Reader
(Assistant holds up paper tape reader)
The paper tape reader uses a rubber typed wheel on a geared motor to pull the tape through and under an array of light dependent resistors, with the signals processed by an Arduino. A switch and button on the top allow the paper tape to be pulled through in either direction, with a read speed of around 10 rows per second. My assistant is showing one of the readers we have here, and I encourage you to come forward for a closer look after the talk.

Slide 37: Paper Tape Reader Internals
Lifting the lid off we can see the rubber wheel which pulls the paper through and the array of six LEDs and LDRs in a row. My assistant won't take the lid off now, since we want the printer in one piece for the demo shortly.

Slide 38 Sensor Design
The sensor is quite simple in design. A light emitting diode in one hole shines light on the paper tape, which is detected by the light dependent resistor in a second hole. The wavelength emitted by the LED is chosen to match the maximum sensitivity of the light dependent resistor, in this case 55 nanonmeter green light.

Slide 39 Sensor Circuit
The light dependent resistor has a resistance which under bright light drops to as little as 5 kiloohm, but rises to 20 Megaohm in the dark. We connect one end of the light dependent resistor to the 5V rail and the other end via a 22 kiloohm pull-down resistor to ground. We then measure the voltage at the mid-connection, connecting it to an analogue input of the

Arduino. We have six of these circuits, five for the data holes and one for the sprocket hole—fortunately the Arduino has six analogue inputs.

When the light dependent resistor is dark, it will have a very high resistance compared to the pull-down resistor. Following Kirchoff's law, we know the voltage will distribute across the resistors proportionately, so the voltage provided to the analogue input will be close to zero volts.

Conversely when the light dependent resistor is illuminated, it will have a low resistance compared to the pull-down resistor. The voltage now distributes mostly across the pull-down resistor, and the voltage provided to the analogue input will be much closer to five volts. We'll be looking for low voltages, which will correspond to the dark holes on the tape.

Slide 40: Paper Tape Reader Internals
Looking again at the paper tape reader internals you can just make out the 22 kiloohm pull-down resistors.

Slide 41: Paper Tape Reader Output
Having built out device, the first step was to feed a trial tape through and put an oscilloscope on the analogue pins. Here we see the sensor for the first hole in yellow and the sprocket hole in blue. Both are providing a clear signal with a similar range—the blue signal appears smaller because its vertical range is set to twice that of the yellow signal.

We were initially somewhat baffled, because we had expected the blue sprocket hole to have a more regular pattern and, due to it smaller dots, a smaller voltage range. The problem was solved when we found a wiring problem, and realized we were actually measuring two data holes.

Slide 42: Raw Sensor Data (ADC)
We then connected the device to the Arduino and used it to collect the raw data from the analogue pins, sampling at 200 hertz. The Arduino analogue-to-digital converters are 10-bit, so will yield values from zero (for zero volts) to 1023 (for five volts).

The plot of all five data pins and the sprocket pin is revealing. As expected after an initial random period while the blank leader passes the sensor, we see a series of minima for each pin, corresponding to dark patches on the tape. The yellow line for the sprocket hole shows a regular pattern with a minimum for each hole.

However, notice that the signals are quite smooth—no sharp edges as we meet a hole. Also note that where a signal has a period of no holes—such as x3—it rises to a higher value than when it has frequent minima. This is because light dependent resistors are quite slow to react. Indeed this would limit the speed at which we could run the reader, rather than any concern about tearing tape.

Secondly notice that the absolute values of individual sensors varies considerably. The minimum for bits one and two are around 200, while the maximum for bits zero and four is

around 150. This is because of stray illumination from neighbouring LEDs, meaning that pins with two neighbours generally get more light and so will show a lower reading.

Finally notice that the yellow sprocket signal has indeed got a smaller range than the other pins, since its dark holes are smaller.

Slide 43: Raw Sensor Data (Resistance)
We'll work with the raw sensor data, but using Ohm's Law we can easily convert back to see how the resistance of the sensors varies. We see a range of 50 killohm to 350 kiloohm, so the choice of 22 kiloohm pull-up resistor is reasonable, although a value closer to the average of the data would have given us a somewhat bigger range on the analogue ports.

Slide 44: Sprocket Hole
Central to our approach will be detecting the sprocket holes. We can't use absolute values, so instead we will look for minima in the signal.

We need to avoid spurious minima which occur during the feed-in period. However we can see that "real" minima are preceded by a very fast change in signal value—something which does not occur with noise.

Slide 45: Calculating Rate of Signal Change
A brief diversion into calculating the rate of change of the signal value. This is the slope of the graph at any one point. If we had a continuous function, we would just use the derivative of the function. However in this case we have discrete data points, so we can measure how much the signal value changes over each time step. Delta x divided by delta t.
Slide 46: Calculating Rate of Signal Change
For adjacent data points we just subtract signal values and time values to yield delta x and delta t.

Slide 47: Calculating Rate of Signal Change
However we know that the time step will always be the same. Since we are sampling at 200 Hz it will be 5 milliseconds. We could just divide all the delta in X values by constant 0.05. However the absolute value for rate of signal change doesn't matter, just that it is consistent.

Slide 48: Calculating Rate of Signal Change
So we can just ignore the division and use the delta x value directly. Omitting the division will also make the computation much quicker.

Slide 49: Exponential Smoothing
Calculating slopes in this way relies on having a reasonably smooth signal. Taking a close up look at part of the sprocket signal we can see this is not the case. Random variation in the sensors makes for some roughness.

Slide 50: Exponential Smoothing
The solution is exponential smoothing, where we combine one fifth of the current sensor value with four fifths of the previous signal value.

Slide 51: Exponential Smoothing
This previous value was in turn made from one fifth of its data value and four fifths of the previous value and so on.

Slide 52: Exponential Smoothing
The result is a smoothed signal. We'll get more reliable slope estimates—which is good, but at the expense of reducing the peaks and troughs we wish to detect—which is not so good.

Slide 53: Deltas
Computing the deltas for the five data holes and the sprocket hole, we see two useful properties. First of all they are all centered around the same value, zero, secondly there is a clear distinction between noise—which has small deltas, and signal—which has large deltas.

Slide 54: Sprocket
Since finding the sprocket holes is central to reading the tape, we'll look at that in more detail. Here is part of the sprocket signal as we come to the end of the feed-in tape and see the first few holes.

Slide 55: Sprocket
The deltas show the minima and maxima.

Slide 56: Sprocket
We are interest in the ones where the value transitions from negative to positive, since these will be minima—that is dark holes on the paper tape, but we also get a number of these in the noise period at the start of the tape.

Slide 57: Sprocket
However we can look at the minimum delta achieved since the last minimum, to see if this is a real signal. The blue line traces this value, which resets to zero each time we encounter a minimum. We can see that for noise the value never gets below -2, while for real signals the value is always less than -5. Choosing a cut-off point in the middle—3.5 allows us to simply distinguish signal from noise, and we can see we have found 4 sprocket holes.

Slide 58: Data Bit 2
We now have everything we need to determine if data bits are set. This is the deltas for the sensor for data bit 2.

Slide 59: Data Bit 2
We overlay the sprocket signal and delta. Every time we find a sprocket hole, we look to see whether the delta for data bit 2 has gone below -3.5 since the last sprocket minimum. If it has we have a dark hole, and if not, we don't. We see that data bit 2 has a sequence of four

zeros—i.e. No hole—followed by four ones, then another four zeroes, another four ones, another four zeros, another four ones, another four zeros, then nine ones.

We have a reliable, if relatively slow paper tape reader. It is a simple matter to drive the values out on the Arduino digital pins, via a 5V to 3.3v converter and into the EDSAC running on the MyStorm board. In fact so simple that we haven't actually yet done it, so that will be a subject of a future talk.

Slide 22: Reimagined Tape Reader
The tape reader uses a motor to pull in the tape. The motor was quite cheap and did not pull the tape in at a constant rate so a clock line was added. This is shown by the smaller line of dots. The bigger dots represent a 5 bit number.

To read a single dot, a light is shone on the piece of paper which is reflected either by the white paper or the black dot. The amount of light reflected changes upon the colour of the paper. Therefore if there is a black dot the amount of light reflected is reduced. The reflected light is converted to a voltage using a light sensitive resistor in a potential divider. Using an edge detection algorithm the black dots can be distinguished from the blank paper. The data for each line is gathered and converted into numbers and sent to the MyStorm. Green light was used because the light sensitive resistors were most sensitive to.

After looking at the output of each pin at the output of the potential divider, we noticed ...

Slide 23: 3D printing
A 3D printed case was used to minimise the amount of background light entering the LED chamber where the holes were being sensed. The printer used was a Rep-Rap. This means it was an open source, build-it-yourself 3D printer. A school would have a similar or better type of 3D printer and so would be able to recreate the cases with relative ease. Almost all the other components have used 3D printing as part of their method.

All 3D prints were designed using the open source program OpenSCAD written by Clifford Wolf. OpenSCAD is quite a mathematical program and deal with shapes very well. This should make adjustments to the designs easy.

Slide 24: Reimagined Teleprinter
This project is aimed at schools, so building an electro-mechanical hole punching machine for the teleprinter would be too difficult. Instead, a thermal printer was used. The thermal printer is an Adafruit Mini thermal printer and prints onto thermal paper which is similar to receipt paper.

An Arduino Uno interfaced to the printer. An Arduino was chosen because it is a cheap microcontroller and is common in schools. Not to mention, the library support and testing Adafruit provided for the printer. Arduino Uno and Cuttlefish are also used on the other peripherals.

The Arduino would then connect to the MyStorm. But, the Arduino runs at 5 volts and the MyStorm at 3.3 volts. It would be dangerous to not include a voltage level shifter in between each connection. Two circuits were made; one that stepped up 3.3 volts to 5 volts and the other to step down. There are many different ways to make these circuits but usually the latter is the easier circuit; a simple potential divider. The step up circuit is two common emitter circuits; two transistors whose emitter pin is connected to ground and hence is common. The base of the second is connected to the collector of the first. The ratio between the resistors controls the step up. The circuit must be powered by the maximum voltage, which in this case is 5 volt.

The MyStorm has a buffer which sends individual characters to the Arduino. In the Arduino libraries there are already bit patterns for letters, numbers and other common characters.

The Verilog program allows a string of less than 16 characters to be sent, one character at a time, into a buffer. Verilog does not have a predefined string type. Instead a register with a width of eight times the number of characters.

Similar to the original teleprinter, when a new character enters the buffer the old character is sent to the Arduino and printer. The printer will only print the line when a carriage return and a new line character is received.

The teleprinter was also used to print the punched tape.

Slide 25: Reimagined Initial orders
To replace the uniselectors on the front panel, the reimagined initial orders used two rows of header pins and a connecter for each pair. Originally the idea was to use individual toggle switches but they were too expensive to use on a teaching resource but you can if you want to.

There are seventeen pairs of headers and each header represents a bit in the current instruction. When a pair is connected, the bit is a 1.

When all the pairs are correct, the user presses one of three buttons. Each button represents a instruction. When an instruction has been loaded the corresponding LED turns on.

Slide 26: Delay line
The original EDSAC delay lines used sound to send the digital signal in loops. Sound is slower than electricity and therefore was easier to control. Once the sound had passed

through the tube sequential logic devices such as flip flops and gates, converted it back to an electric signal.

A piezoelectric transducer was used to generate sounds of 0.5 Mega Hertz. The transducers would also be very large to create a very thin beam of sound which would touch the sides of the tube as little as possible.

The delay lines were filled with Mercury because the acoustic impedance of mercury is almost exactly the same as that of the piezoelectric crystals. This minimized the energy loss and the echoes when the signal was transmitted.

But to get the acoustic impedances to match as closely as possible, the mercury had to be kept at a constant temperature. The system heated the mercury to a uniform above-room temperature setting of 40 degrees Celsius, which made servicing the tubes hot and uncomfortable work.

Slide: Other fluids & Turing
Mercury is not the most accessible fluid for most schools to supply so we had to think up some other fluids to use. On our search we found that Alan Turing suggested Gin. But I doubt schools would stock much of that too. So how about wine? Or home brewed beer? Or for the non-alcohol drinkers, pepsi?

Slide: Reimagined delay line picture
In the end we choose air for a number of reasons. First is that it came ready packaged with the tube. Second the speed of sound is quite low through air allow an audience to be able to hear the bits being sent.

Piezoelectric transducers and receivers are also very, very expensive and so were replaced by a normal speaker and microphone. Which means that the delay line can hold up to 49 bits but there are too many reflections and echoes to rely on even a 10 bit word for very long.

To minimise the harmful echos, we thought of using mirrors at each end of the tube. The speaker and microphone face their respective mirror. When the speaker emits sound, it reflects off of it’s mirror. The reflections that are at a right angle to the mirror have the best chance of reaching the other mirror and being reflected back into the microphone. The other reflections end up bouncing around the tube and very few would make it to the microphone.

But in the end just stuffing the ends with acoustic foam seemed to work well enough in a quiet office.

Slide: Circuit diagram
The microphone requires a small circuit to amplify the output enough to be received by the Arduino. The microphone must be connected to ground and a pull up resistor to work but there are several different variations to the rest of the circuit. Some favour slower frequencies and others favor higher ones. The circuit I used was the simplest.

It used an audio amplifier chip which can be brought from all good electronics retailers. To almost double the gain of the chip, a capacitor is connected between pin 1 and 8. Before the microphone signal reaches the chip, though, there is a variable resistor for controlling the volume.

Slide: Mic Circuit pic
Here is the veroboard version.

Slide: I/p & O/p Mic Circuit
The blue line is the output straight from the microphone and yellow is the output of the amplifier circuit. As you can see there is a large change in amplitude and there would be no way to get the Arduino to read good enough values straight from the microphone.

Slide: Speaker Circuit pic
On the speaker side, the circuit is a lot smaller. To prevent damaging the speaker or the Arduino, the speakers output resistance must match the input resistance of the Arduino. The output resistance is 8 ohm, hence, a 10 ohm resistor is used.

Slide 28: Water/ Beer mess
Instead of air, water could be used. The speed of sound is 1498 meters per second in water which is slightly faster than mercury. Therefore, if the same size tube was used to send 10 bits per second, the pulses would have to be just less than 15 kiloHertz. The Arduino would struggle to keep up so it needs to be replaced by the Raspberry Pi.

Unfortunately it is rather difficult to 3D print a water tight clamp since the printer prints in layers and is only accurate to 0.4 millimeters. This meant that when we put the water into the tube, it all come out the other end.

***Slide 29: Demo ~ 5 min?**
I am now going to do a live demo for two of the peripherals; the tape reader and the delay line. Please keep in mind that this is still a work in progress and combined with the live demonstration curse, things may just refuse to work or blow up in my face.

First is the tape reader.

Now onto the delay line. The delay line will repeat the pattern twice to make sure that the tube has been filled with the information. After 10 loops of the pattern there will be a pause. This is there to clear the echos. The pattern is a 10 bit binary number. In decimal it is 592. You will hear a bleep for every 1 and silence for every 0.

The delay line is quite loud. If you are sensitive to sound feel free to cover your ears. I won't be that offended. But, if you are at the back, you probably won’t be able to hear it.

The repeated patterns are not the same due to echoes and other sounds outside the tube. Sometimes the first repeated pattern is very similar.

<demo delay line>

Thanks for watching the demo. If you had your fingers in your ears for the delay line, now is the time to take them out.

Slide: Chip Hack
These peripherals are meant for education so the best way to use them was to run a tutorial course and include them as a resource. This was the three day Verilog teaching course called Chip Hack. The course used MyStorm boards and had a lot of very interesting talks surrounding the history of EDSAC. Once everyone had tried Verilog programming, they were encouraged to bring up a Verilog EDSAC on their MyStorm and interface it with a peripheral. Chip Hack also coincide with the 60th anniversary of the BCS, a society founded by Wilkes.

Slide 30: Summary
EDSAC was run in May 1949. It was one of the first general purpose computers to be used and was open for University of Cambridge students.
Race forward to 2017, EDSAC was reimagined as an open source, educational project for schools and hobbyists. That uses five physical peripherals to aid in understanding the key parts of the original EDSAC.

Slide: Thank you
I would like to thank the international open source team who helped me put together EDSAC.

- Alan Wood and Ken Boak (MyStorm)
- Bill Purvis and Hatim Kanchwala (Verilog EDSAC)
- Dan Gorringe and Peter Bennett (peripherals)
- Everyone at Chip Hack who put the peripherals through their paces.
- Finally, I would like to thank the Embecosm team; without them my talk and demonstration would be in shambles.

Slide: Contact
I hope I have managed to inspire you to recreate this project at home. This project is always looking to be improved. Any patches are welcome. All resources are on GitHub.
Any questions?