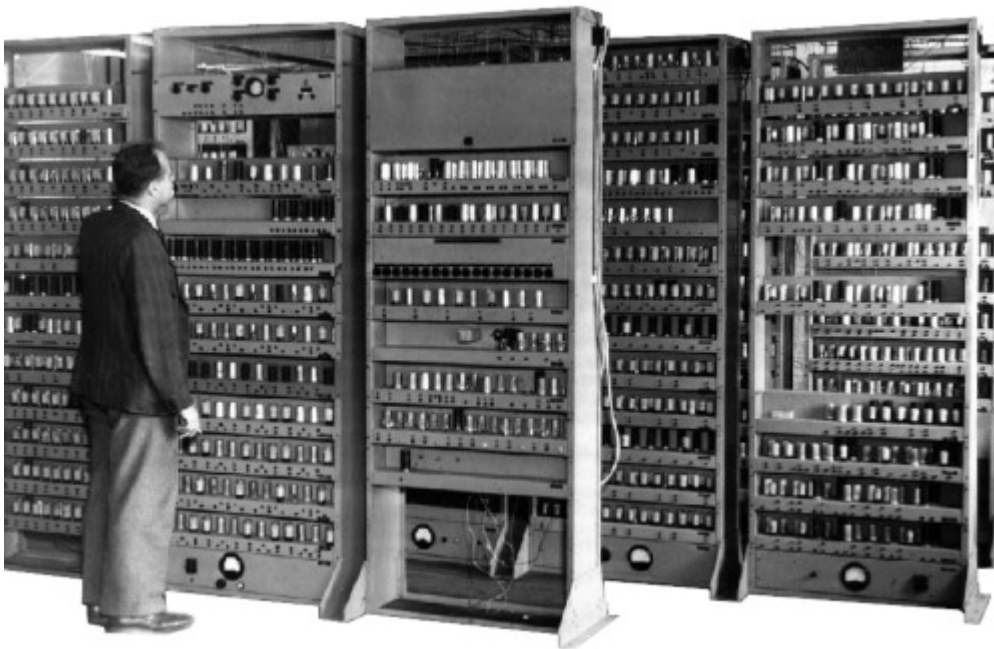
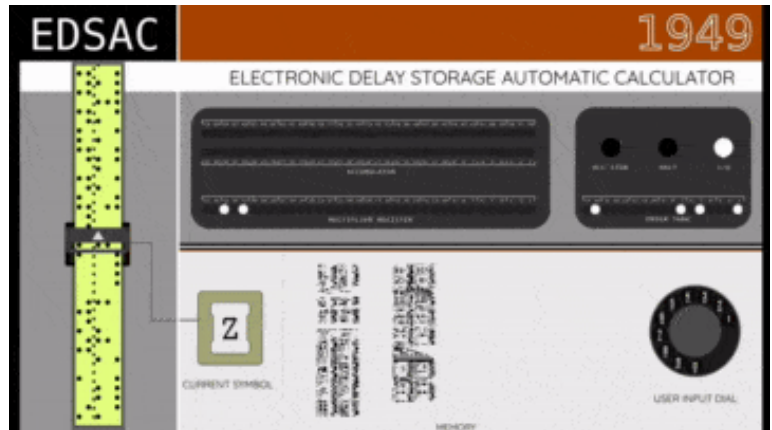


EDSAC FPGA core for MiSTer

This is a FPGA implementation of [EDSAC](#) (Electronic delay storage automatic calculator), the first practical general purpose stored program electronic computer in the world. Construction began in 1947 and it executed its first program in May, 1949. Featuring a mercury delay line memory, it had around 3400 vacuum tubes and consumed roughly 15 kW of power. This fascinating computer also pioneered the concept of assembly language, using library routines and the first book about programming was written to accompany this cool machine. Fun fact - there is no procedure call instruction as the concept hasn't even been invented yet!



EDSAC was also a home to one of the first games ever for a digital computer, *noughts and crosses* (known as tic-tac-toe across the pond), written by [A.S.Douglas](#) in 1952. This FPGA core can run the original code and enables you to play a game that's almost 70 years old, making it one of the oldest original games for a digital computer that survive to this day. One of the

obvious but overlooked facts is that the game was a single player variant and you were competing against the machine AI.

Despite being unbeatable at tic-tac-toe, EDSAC's primary design objective was mathematics and many daunting engineering tasks and scientific problems were solved using this technological breakthrough.

Check out the [installation guide](#) for instructions how to set it up, or see a [DEMO VIDEO](#).

Implementation details

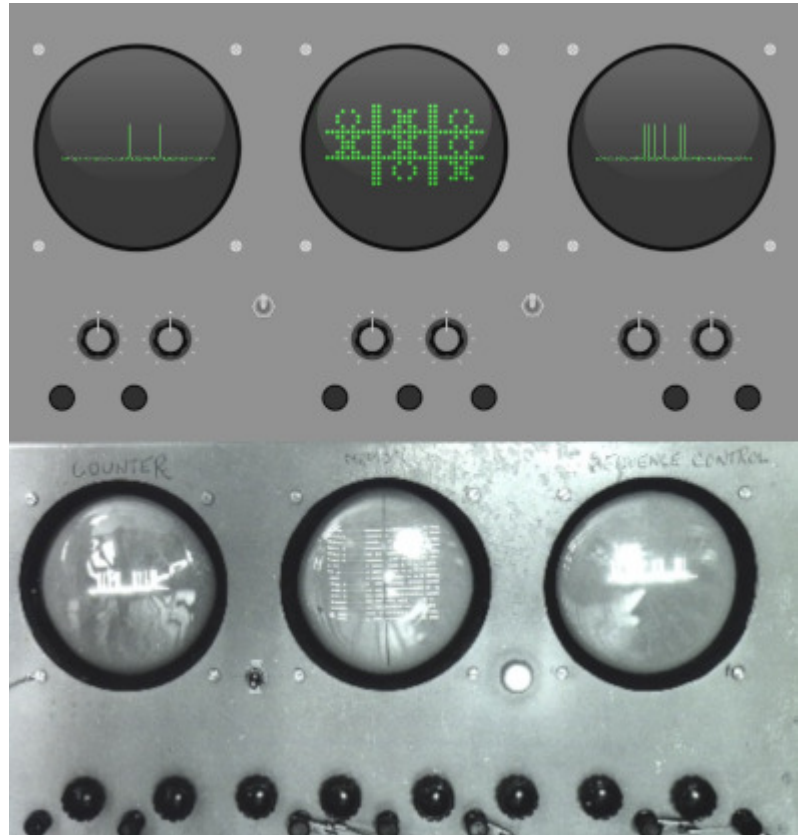
Scope Display

Scope background is implemented as an on-the-fly image decompression with information drawn on top. Original EDSAC had two of these scope racks with three CRT tubes each, but only the top one is implemented due to screen size considerations. Left scope mirrors an internal counter and the right one displays the sequence control tank (SCT). The most interesting is the one in the middle - it displays the contents of a selected memory tank.

EDSAC had 16 35-bit words per delay line memory tank and eventually reached 32 tanks, amounting to 2240 bytes of memory altogether. When powered up for the first time in May 1949, it had only 4 tanks operational but that was enough to demonstrate the ability and importance of the machine.

Displaying individual bits of a single delay line tank made for a simple 35x16 display which could be used to show some meaningful information. This was used to display the tic-tac-toe game output.

The "noise" on left and right scope was made with a LFSR, linear feedback shift register. All of the examples I've found seemed very complex so I implemented a short and simple solution following an [application note from Xilinx](#).



FPGA version vs. original

```
reg [30:0] lfsr;

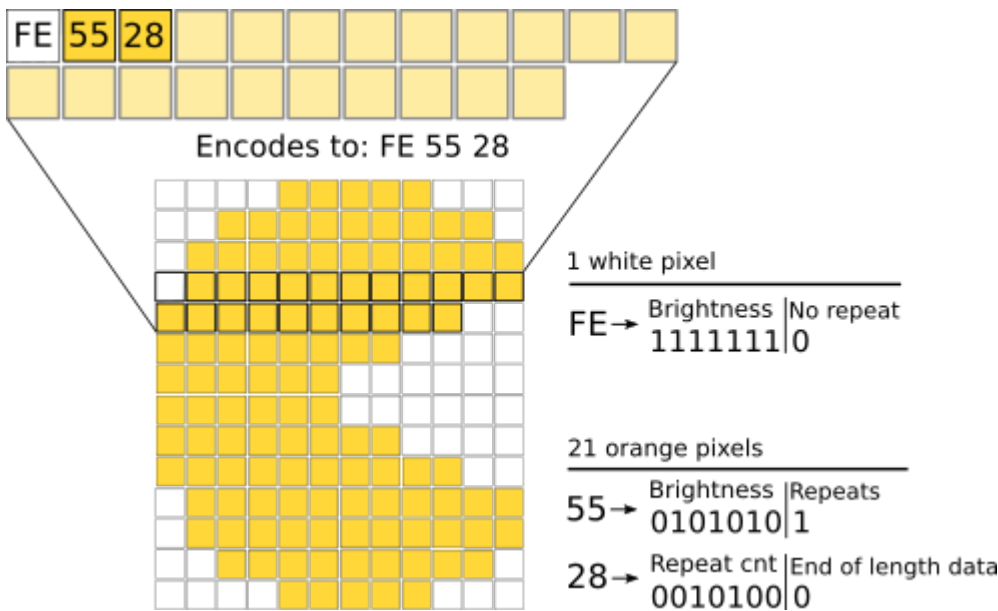
always @(posedge clock) begin
    lfsr <= {lfsr[29:0], lfsr[30] ^~ lfsr[27]};
end
```



Much simpler!

Background images

Because of the limited memory within the chip, some compression scheme is needed as not to waste memory. Therefore, a simple RLE compression variant is implemented, making it possible to fit an entire background image for both scope and panel into 64K.



1. Read character
2. If the least significant bit is a 0, it is not compressed and the remaining 7 bits are used as luma. Re-start.
3. Otherwise, we need to fetch the repeat count. It is permissible to use up to 3 following bytes for the size parameter. If the character read has a 1 in its LSB position, there is more size data to read and next character is read in as well. This enables to store $7 \times 3 = 21$ bits of size data.
4. Keep outputting the initial character until the size counter runs out, then re-start the algorithm.

Color was not considered initially, but was added later as a simple palette lookup table with just a few colors.

Teletype

Teletype was mostly borrowed from my [PDP-1 project](#), with scroll improvements and added printing sound for slightly more realistic feel. The sound clip was ~170 ms, therefore the output speed was capped so the clip can play without being interrupted by another one starting.

Screen is divided into a 64 x 32 pixel matrix, and each field contains one character. To store this data, a 2048 byte RAM is instantiated. As lines are drawn for each frame, the read address is changed when the horizontal counter "jumps" from one grid element to the next. Each character is then looked up in the ROM, depending on the value read and the grid element line number we are currently plotting.

1	1	1
2	4	3
3	9	5
4	16	7
5	25	9
6	36	11
7	49	13
8	64	15
9	81	17
10	100	19
11	121	21
12	144	23
13	169	25
14	196	27
15	225	29
16	256	31
17	289	33
18	324	35
19	361	37
20	400	39

There is a character set ROM and a frame buffer RAM. After choosing a suitable font which was used on contemporary teletypes, it was exported to individual bitmaps and processed with Python image library to produce a memory image format file. This file is stored in ROM and looked up to draw individual characters.

```

D6A5 : 0000000000000000;
D6A6 : 0000000000000000;
D6A7 : 0001111111111000;
D6A8 : 0011111111111000;
D6A9 : 0011000000000000;
D6AA : 0011000000000000;
D6AB : 0011000000000000;
D6AC : 0011000000000000;
D6AD : 0011000110000000;
D6AE : 0011011111110000;
D6AF : 0011110000111000;
D6B0 : 0011100000011100;
D6B1 : 000000000001100;
D6B2 : 000000000001100;
D6B3 : 000000000001100;

```

	!	"	#	\$	%
5	6	7	8	9	:
J	K	L	M	N	O

A more accurate look might be to have several variations of the same font to account for minute differences in the same letter and then use a *lfsr* pseudorandom generator to select the current variant.

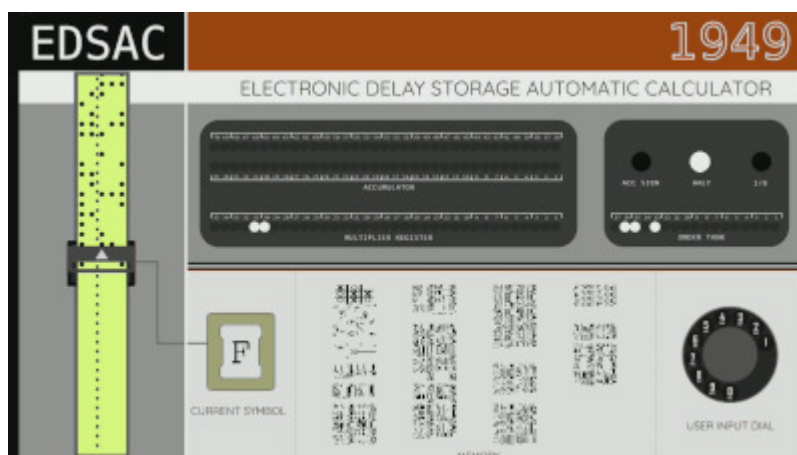
Panel

```

D6B5 : 0000000001110000;
D6B6 : 0000000011110000;
D6B7 : 0001111100000000;
D6B8 : 0001110000000000;

```

While the scope tries to recreate the original piece of hardware, the panel is a made-up concept showing the paper tape reader in action with the symbols decoded, memory contents and register states. I believe it improves the educational aspect of this project, being merely a helpful tool to better understand the architecture and functioning of the machine.









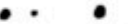

























It is constructed as a RLE compressed background image with register contents displayed on top. Paper tape reader shows the contents of the paper tape buffer memory (where tape images are stored after uploaded to FPGA). As tape is being read, the pointer shifts and the screen is redrawn, creating the illusion of the paper tape moving through the reader. The offset is set so the character being read coincides with the tape read head on the screen.

Middle section displays the entire memory footprint which can provide a good insight into how the computer works and what data is modified. It is divided into four sections of eight delay tanks.

Tape

Note that the EDSAC uses a 5-bit code and knows no difference between letters and numbers - it all depends on current context. Therefore, a "5" could also be a "T" on the current symbol indicator. Note how the first bit is inverted so that blank, unpunched tape isn't confused for a zero.

(1)	(2)	(3)	(4)
Tape	Binary	Punch	Teleprinter
	10001	F	F
	10010	θ	Carr-retn.
	10011	D	D
	10100	φ	Space.
	10101	H	H
	10110	N	N
	10111	M	M
	11000	A	Line feed.
	11001	L	I
	11010	X	X
	11011	G	G
	11100	A	A
	11101	B	B
	11110	C	C
	11111	V	V
	00000	P	P
	00001	Q	Q
	00010	W	W
	00011	E	E
	00100	R	R
	00101	T	T
	00110	Y	Y
	00111	U	U
	01000	I	I
	01001	O	O
	01010	J	J
	01011	π	Fig. shift.
	01100	S	S
	01101	Z	Z
	01110	K	K
	01111	Erase	Let. shift.
	10000	Space	

Edsac character set

Audio

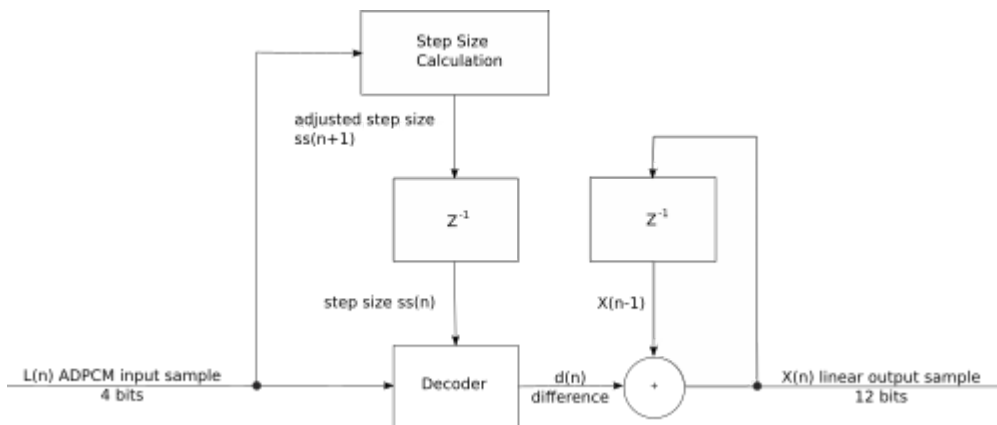
Since memory is not abundant, some form of compression was needed that was effective, yet simple enough to implement in Verilog within a reasonable timeframe. I opted for OKI ADPCM 4bit -> 12bit variant and 12 kHz sampling rate, as it is relatively simple to understand and provides a decent enough ratio of quality vs size.

Once again, I was scared with the complexity of some implementations and (for educational purposes) I decided to implement it from scratch and keep it as simple as possible (50 SLOC).

Sound effects are concatenated and stored within 64K of memory:

Sound	Memory Address	Storage Size
Keypress	0000 - 4160	= 4160 bytes
Bell	4160 - 24014	= 19854 bytes
Tape read	24014 - 65462	= 41448 bytes

When a sound is required, pointer is set to the start of sound block and length is defined. After it runs out, the system is free to play another sound. The notable exception is the bell which sounds when the machine is halted - it can override whatever is playing. There is no multiple voice support (even though it could be done, it would complicate things with little to no benefit).



ADPCM decoder block diagram

CPU implementation

Instructions were implemented by addressing the effect an instruction should have on the system as opposed to achieving a gate-level accuracy, since that would take much more time than was available.

After completing the instruction set according to explanations found in available literature, bug fixing was a tedious task due to some initial assumptions about how certain instructions work were wrong which took time to figure out and fix. As a final test, I've ran a program made by M.Wilkes to calculate the Chapman's Grazing Incidence Integral and compare the output to the table in his published paper [1]. Being computationally intensive, this revealed several issues and after fixing them, the results came out correct as intended. No formal instruction tests exist to the best of my knowledge.

Instruction set (1949)

Instruction	Explanation
A n	Add the number in memory location n to the accumulator
S n	Subtract the number in memory location n from the accumulator
H n	Copy the number in memory location n to the multiplication register RS

Instruction	Explanation
V n	Multiply number in memory location n with number in RS register and store to accumulator
N n	Multiply number in memory location n with number in RS register and subtract from accumulator
T n	Transfer accumulator to memory location n and clear the accumulator
U n	Transfer accumulator to memory location n and <i>don't</i> clear the accumulator
C n	Logical AND of number in memory location n with number in RS, store to accumulator
R n	Arithmetic shift the accumulator right m times, where m is the number of zeros the number ends with
L n	Arithmetic shift the accumulator left m times, where m is the number of zeros the number ends with
E n	Jump to memory location n if the number in accumulator is positive or zero
G n	Jump to memory location n if the number in accumulator is negative
I n	Read a character from paper tape and write it to memory location n as 5 lowest bits
O n	Prints the character formed by the lowest 5 bits in memory location n
F n	Write the last character that was written to the teletype to memory location n
X	No-op
Y	Add 1 to bit 35 of register ABC, rounding ABC to 34 fractional bits
Z	Halt the machine and ring a bell. Microwave style!

The order format is:

```

  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|      op      |xx|                address                |SL|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```



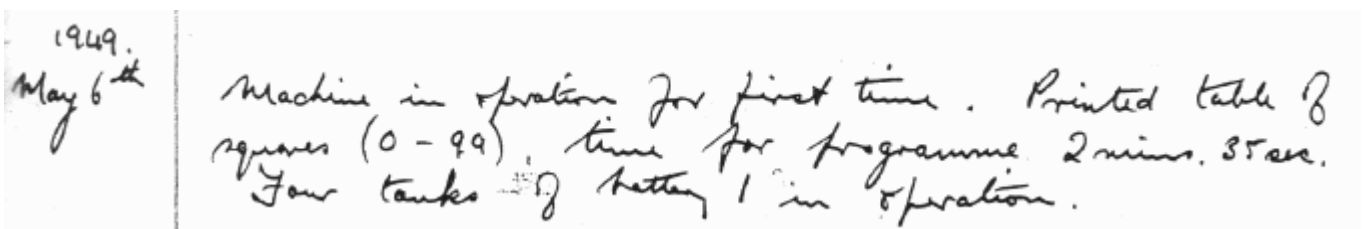
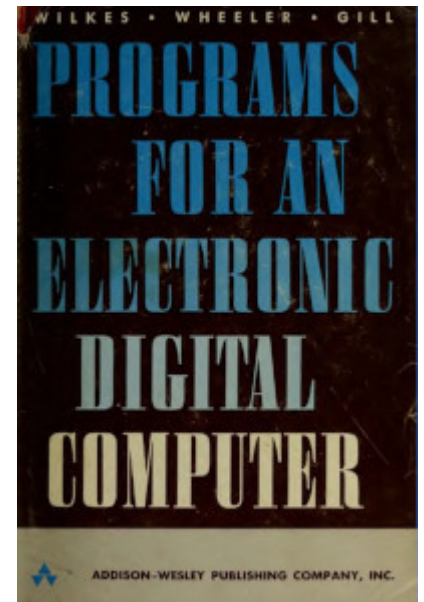
- op = instruction opcode,
- xx = unused,
- address = 10 bit address in memory,
- S/L = length code (short/long, i.e. addressing full or half words)

EDSAC specification

- ~ 650 instructions/sec
- 32 memory tanks, 16 35-bit words of memory per tank, 512 words in total
- Mercury delay line memory
- Paper tape input, 5-track electromechanical reader running at 6 2/3 characters per second
- Teleprinter output at 6 2/3 characters per second
- Roughly 3,400 valves
- Power consumption: ~15 kW
- Area footprint: 5 x 4 meters
- In total there are 142 chassis in 12 racks
- Three rows with four, three and five racks respectively
- Clock speed: 500 kHz
- Made significant contribution to 3 Nobel prizes

Trivia

- First ever practical book on computer programming was published primarily around EDSAC and published in 1951 - "The preparation of programs for an electronic digital computer", M.V. Wilkes, D.J. Wheeler and S. Gill.
- Initial orders (or what we know as bootloader) consisted of 31 instructions in version 1 or 41 instructions in version 2. The latter enabled program relocation and was surprisingly powerful given the code size.
- Operators would wire the accumulator sign to a speaker and could tell by listening if the program was misbehaving or working correctly
- Memory was made out of long tubes filled with mercury. Sound pulses were transmitted on one side and received at another, and as soon as they arrived they were modified as needed, regenerated and sent back through the tube. Simple analogy would be a clown juggling balls and inserting or removing one as he sees fit.
- First output happened on May 6th, 1949 when a list of squares was printed out.



How to install

Download the latest .rbf from releases folder and place it on your SD card. Place some tape images (can be found in the software folder) in /Games/EDSAC folder on the MiSTer. Select the rbf from the

MiSTer menu to execute.

Running the core

This is hardly an interactive computer - the only means of input is a paper tape program and a telephone dial (added later) to enable the user to provide some rudimentary input during program execution. Therefore, to make it do anything you need to load the tape image first.

Pressing F12 will bring up the menu where you can select the tape (.TAP) image to be uploaded. After selecting the image, the panel screen (accessible with F3) will show the paper tape as inserted into the reader and ready to start.

After applying power, the computer has no idea what to do so there must be some initial instructions to enable loading program from tape. There are two versions of the initial instruction for EDSAC, the version 1 from May 1949 and version 2 from September of the same year. Most tapes use V2 (and that's why it is default) but some older programs that are still preserved use the V1.

Loading a program

General loading procedure:

- F12, load tape, select tape image
- F3, switch to panel display
- Press "I" on the keyboard to erase memory and load initial orders
- Press "R" to reset
- If HALT light is on, press "C" to start execution

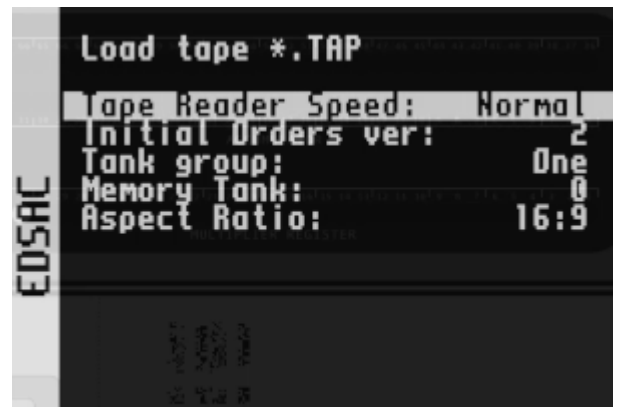
Keyboard shortcuts

- 0-9 - enter number using the telephone dial
- F1 - CRT screen
- F2 - Teletype output
- F3 - Panel
- F12 - On Screen Menu
- I - Clear memory and write initial orders
- E - Erase teletype screen
- R - Reset
- H - Halt computer
- C - Continue execution (resume)

OSD menu options

- **Load tape *.TAP** - Select the tape image to be transferred to EDSAC
- **Tape Reader Speed** - Enables speeding up loading for the impatient

- **Tank Group** - Memory is divided into 2 groups of 16 delay line tanks, this chooses tank group
- **Memory Tank** - Within the tank group, you choose which long tank will be displayed on the middle CRT.
- **Aspect Ratio** - Select which aspect ratio will be used to display the image on your screen. You can choose between 16:9 and 4:3.



Known issues

- Code needs more work and a decent refactoring
- Halt on invalid instruction should not sound the bell, only Z instruction should
- Instruction timing should be historically accurate, but not enough information were available to implement it

License

Available under MIT license.

Credits

A big thanks goes to Andrew Herbert and Bill Purvis for being incredibly helpful and providing a lot of information that would be otherwise inaccessible. Kudos to all TNMOC team doing the EDSAC replica, their engineering skills are only topped by their enthusiasm.

Bibliography

1. Wilkes, M. V. (1954). A Table of Chapman's Grazing Incidence Integral $Ch(x, X)$. Proceedings of the Physical Society. Section B, 67(4), 304–308. doi:10.1088/0370-1301/67/4/304
2. <https://www.cl.cam.ac.uk/events/EDSAC99/statistics.html>
3. Report of a Conference on High Speed Automatic Calculating-machines, University Mathematical Laboratory, Cambridge, June 1949;
4. Martin Campbell-Kelly: Programming the EDSAC, IEEE Annals of the History of Computing, Vol 2(1), 1980.
5. Dasgupta Subrata: It Began with Babbage, Oxford University Press, 2014;
6. Dodd K.N., Glennie A. E. An Introduction to the Use of High-speed Automatic Digital Computing Machines, Ministry of Supply, Memo No. 7/51 of Armament Research Establishment, Physical Research Division, 1951;