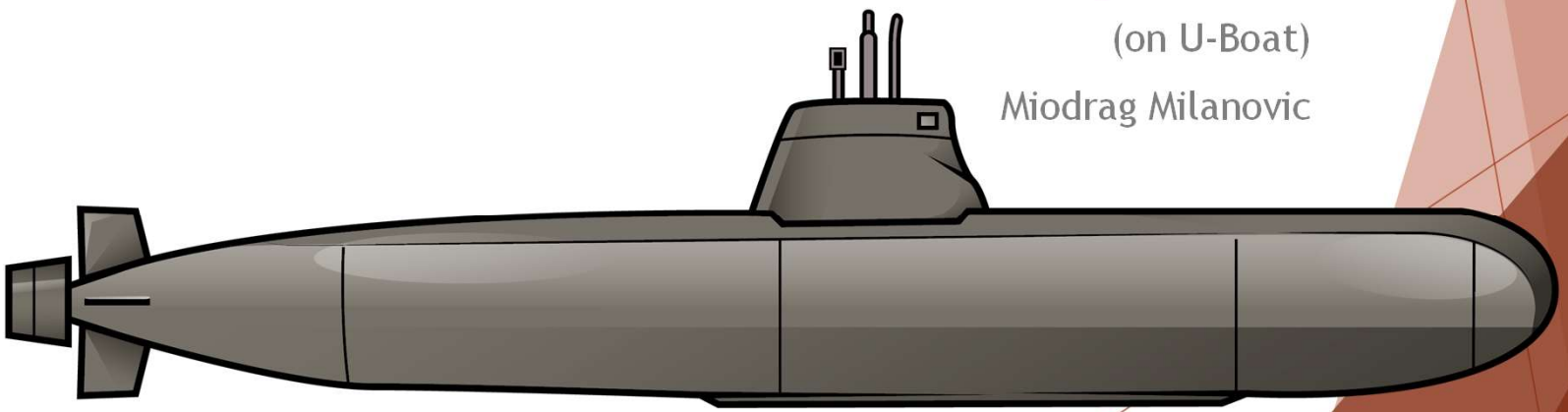
FPGA for Software Developers

(on U-Boat)

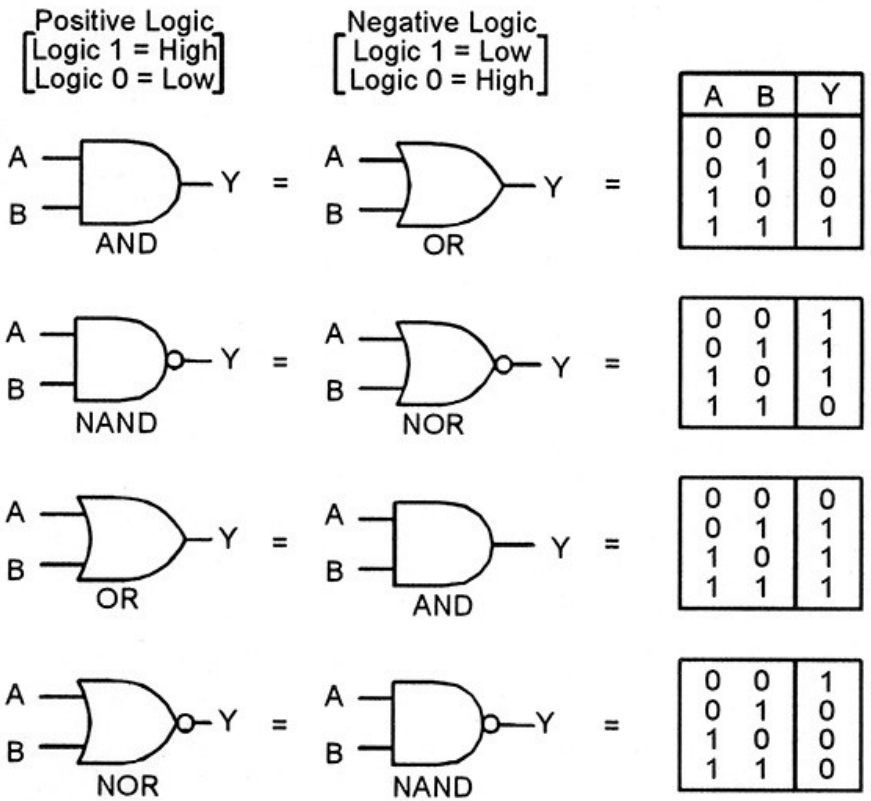
Miodrag Milanovic



Scenario

- We are reverse engineers from present time
- Sent back in time to U-1206 during WWII
- Find Enigma and create own copy of it
- Only thing that they gave us is FPGA and laptop
- But we do not know how Enigma works
- We do not even understand how FPGA works
- And we have 1 hour limit for doing it
- For fun sake we are going to make a diversion at the end

Boolean logic



A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

0	0	1
0	1	1
1	0	1
1	1	0

0	0	0
0	1	1
1	0	1
1	1	1

0	0	1
0	1	0
1	0	0
1	1	0

D Flip-Flop

D Flip-flop

Symbol

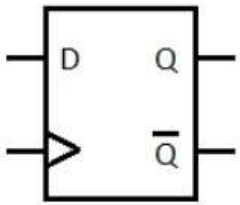
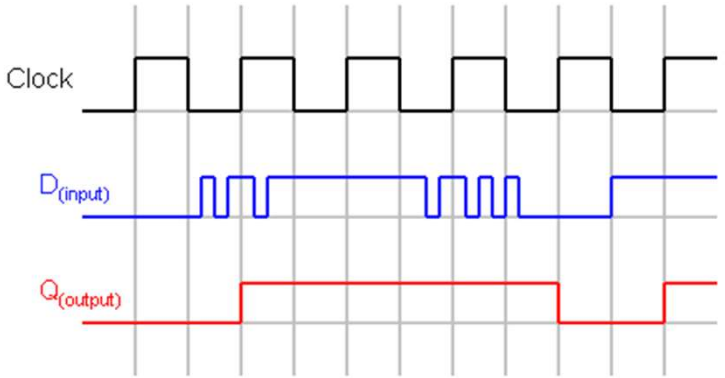
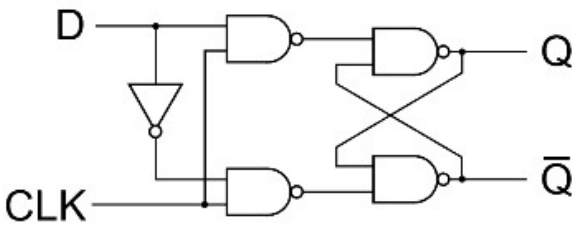


Table of truth:

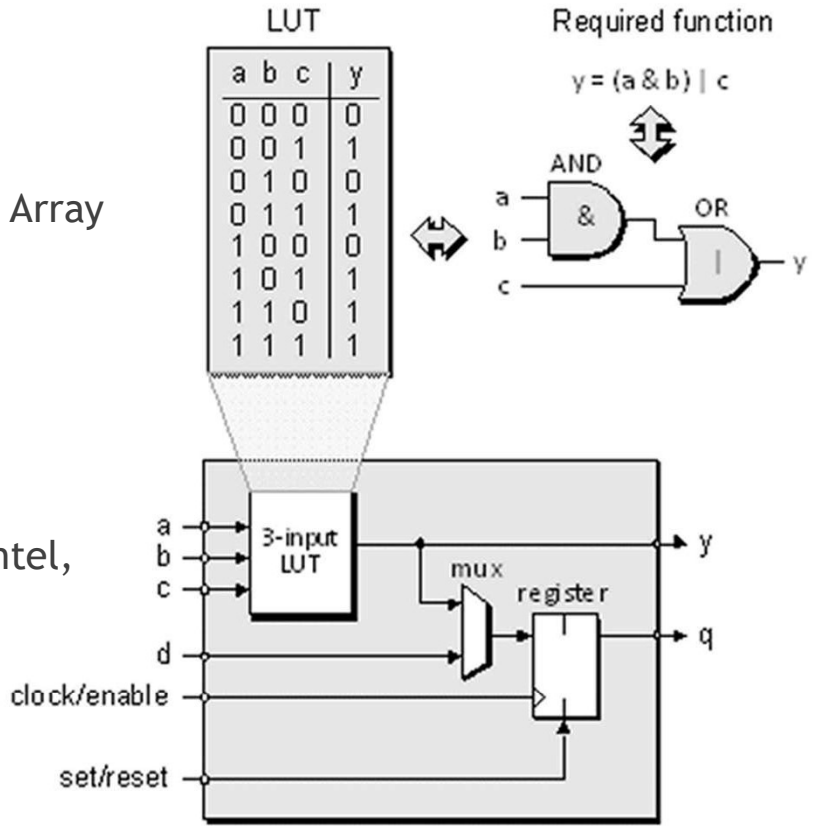
clk	D	Q	$\overline{Q}$
0	0	Q	$\overline{Q}$
0	1	Q	$\overline{Q}$
↓	0	0	1
↓	1	1	0





What is FPGA ?

- Field Programmable Gate Array
- LUT – Look-Up Tables
- Clock Tree
- I/O Blocks
- Dedicated RAM
- Vendors : Xilinx, Altera/Intel, Lattice, Microsemi …



Where FPGA is used ?

- Prototyping
- Small series commercial products
- Where ever power consumption is critical and no budget for ASIC
- Implementing security algorithms
- Fast network packet analysis
- Many, many more

Process of development

- Analysis : parsing and validation of HDL (Verilog, VHDL …)
- Synthesis : HDL or schematics -> netlist
- Place-and-route : netlist -> specific FPGA technology
- Assembler : specific FPGA technology -> bitstream
- Timing Analysis
- Simulation

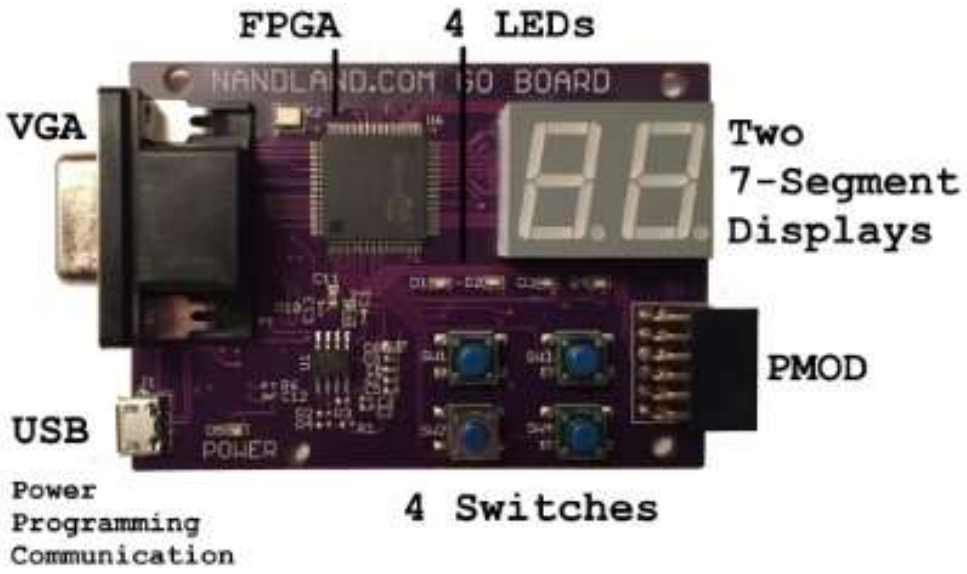
Open Source Tools

- Yosys - Verilog synthesis tool by Clifford Wolf
- Arachne PnR - Place and route tool by Cotton Seed
- Project IceStorm – Assembler tool by Clifford Wolf and Mathias Lasser

- Lattice iCE40 FPGA only

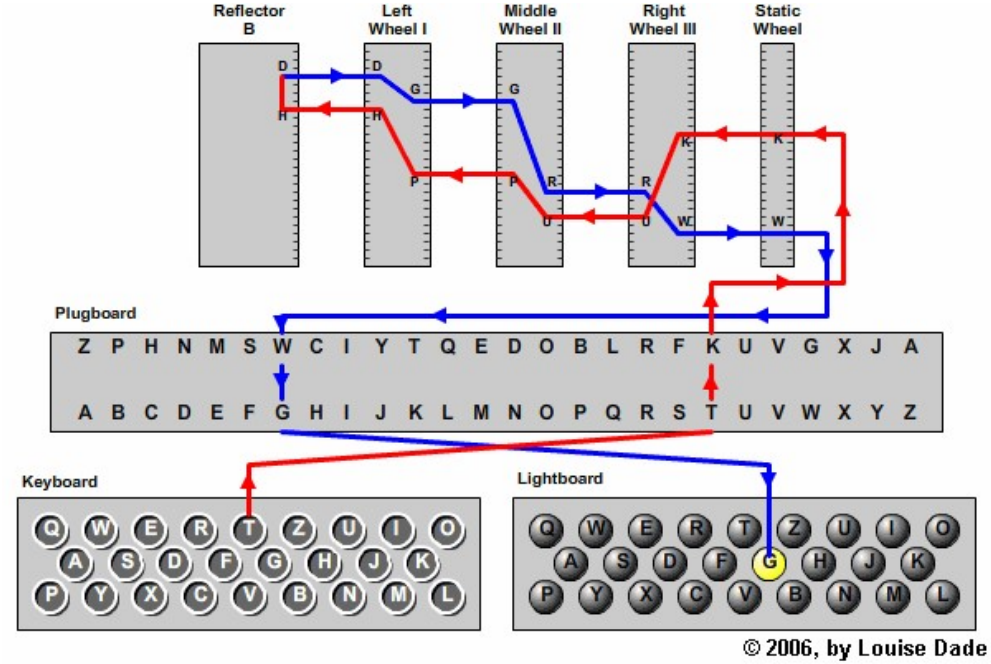
- Icarus Verilog by Stephen Williams
- Verilator by Wilson Snyder with Duane Galbi and Paul Wasson

The Go Board by Russell Merrick



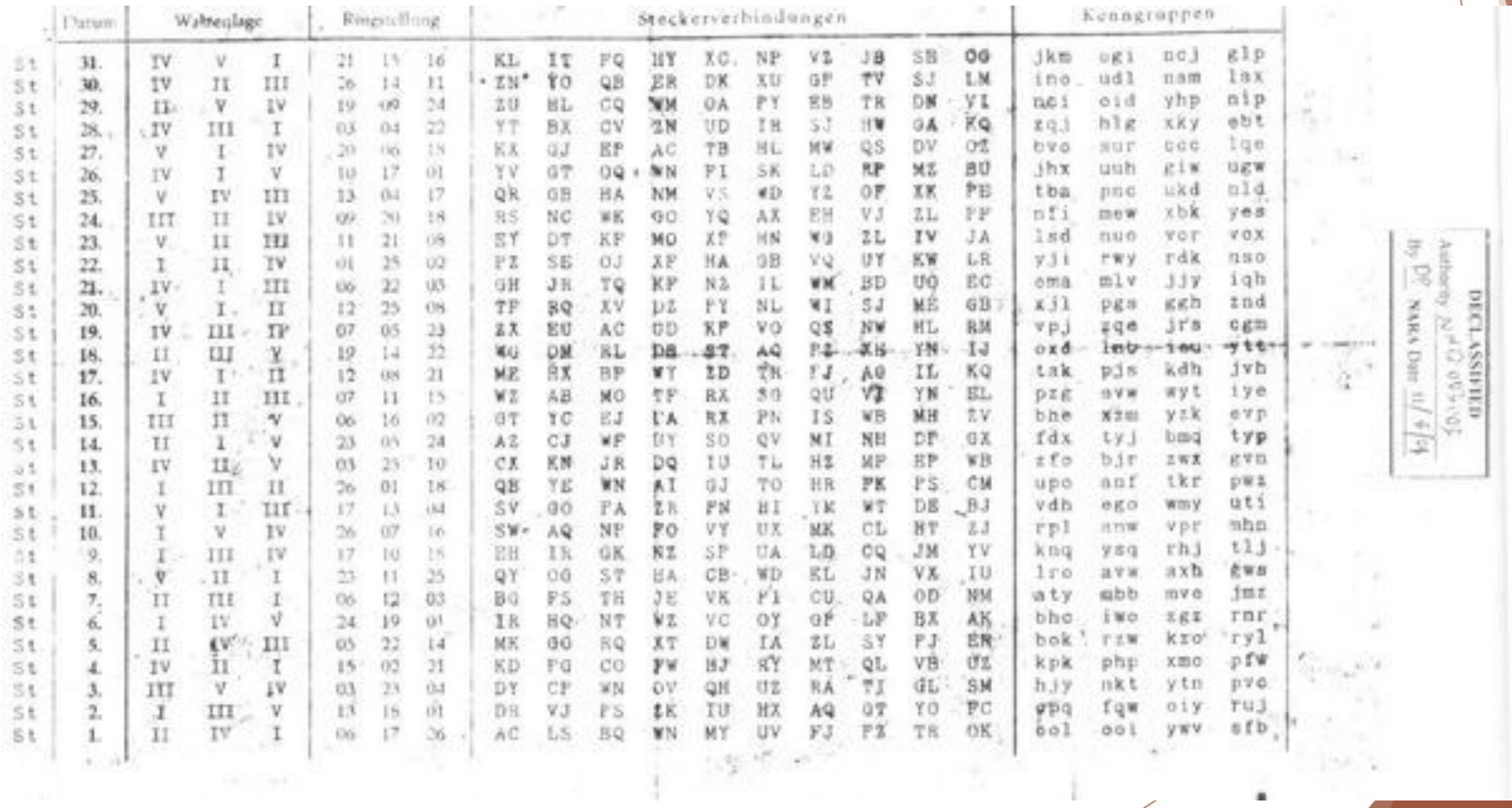
Enigma Machine

How Enigma works ?



http://enigma.louisedade.co.uk/ Enigma Emulator by Louise Dade

Enigma settings

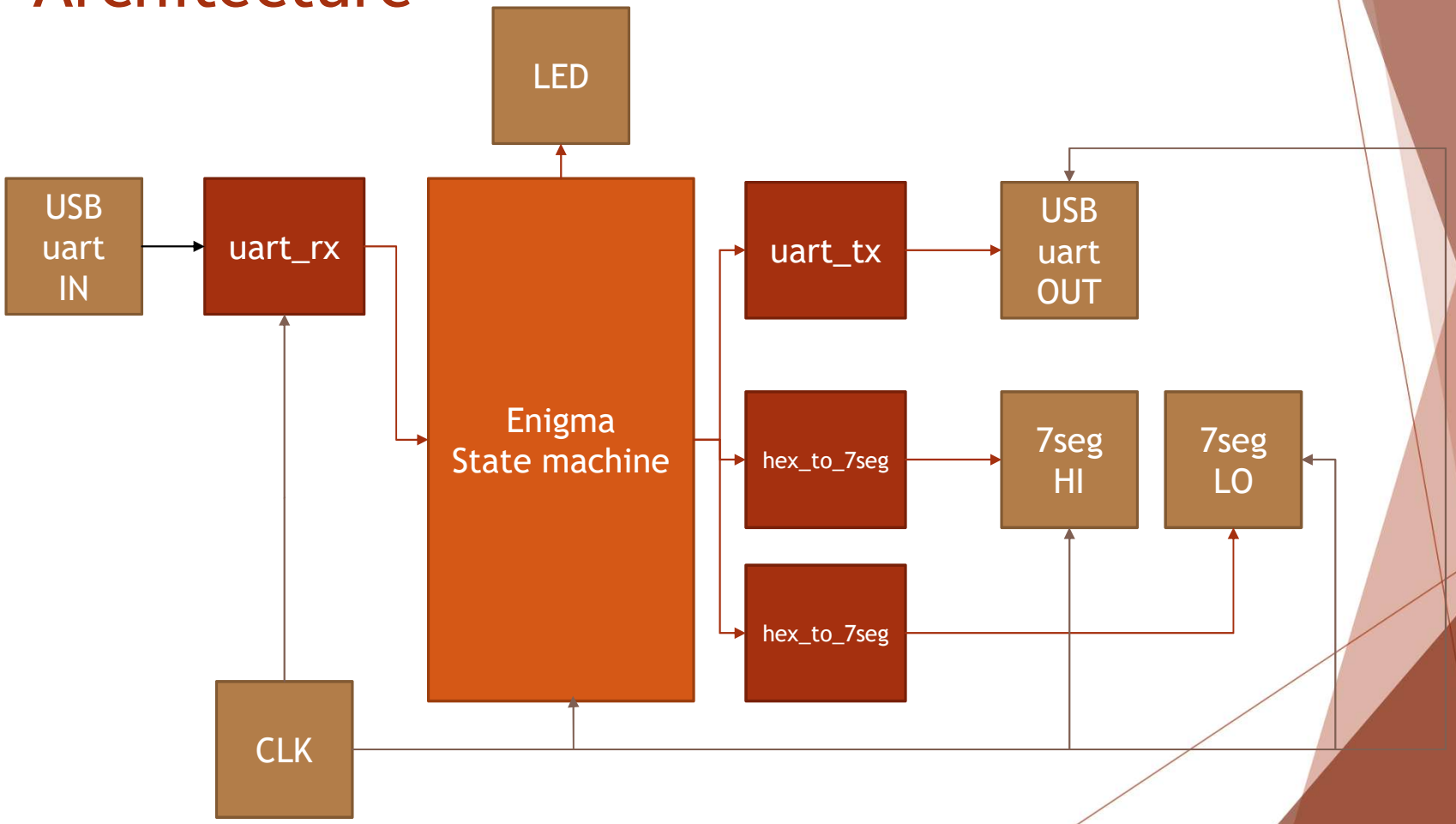


	Datum	Walzenlage			Ringstellung			Steckerverbindungen										Kenngruppen			
St	31.	IV	V	I	21	15	16	KL	IT	FQ	HY	XC	NP	VZ	JB	SB	OG	jkm	ogi	ncj	glp
St	30.	IV	II	III	26	14	11	ZN	YO	QB	ER	DK	XU	GP	TV	SJ	LM	ino	udl	nam	lax
St	29.	II	V	IV	19	09	24	ZU	HL	CQ	WM	OA	PY	EB	TR	DW	YI	nci	oid	yhp	nip
St	28.	IV	III	I	03	04	22	YT	BX	CV	ZN	UD	IR	SJ	HW	GA	KQ	zqj	hlg	xky	ebt
St	27.	V	I	IV	20	06	18	KX	QJ	EP	AC	TB	HL	MW	QS	DV	OZ	bvo	sur	ccc	lqe
St	26.	IV	I	V	10	17	01	YV	GT	OQ	WN	PI	SK	LD	RP	MZ	BU	jhx	uuh	giw	ugw
St	25.	V	IV	III	13	04	17	QR	GB	HA	NM	VS	WD	YZ	OF	XK	PE	tba	pnc	ukd	nld
St	24.	III	II	IV	09	20	18	RS	NC	WK	GO	YQ	AX	EH	VJ	ZL	PF	nfi	mew	xbk	yes
St	23.	V	II	III	11	21	08	SY	DT	KP	MO	XI	HN	WQ	ZL	IV	JA	lsd	nuo	ver	vox
St	22.	I	II	IV	01	25	02	PZ	SE	OJ	XF	HA	GB	VQ	UY	KW	LR	yji	rwy	rdk	nso
St	21.	IV	I	III	06	22	03	GH	JR	TQ	KP	NS	IL	WM	BD	UO	EC	oma	mlv	jjy	iqh
St	20.	V	I	II	12	25	08	TF	BQ	XV	DZ	FY	NL	WI	SJ	ME	GB	xjl	pgs	gch	znd
St	19.	IV	III	II	07	05	23	ZX	EU	AC	OD	KP	VO	QS	NW	HL	RM	vpj	zqe	jrs	cgm
St	18.	II	III	V	19	14	22	WG	DM	BL	DB	ST	AQ	PZ	XS	YN	IJ	oxd	lnb	ieu	ytt
St	17.	IV	I	II	12	08	21	ME	SX	BP	WY	ID	TR	FJ	AO	IL	KQ	tak	pjs	kdh	jvb
St	16.	I	II	III	07	11	15	WZ	AB	MO	TF	RX	SG	QU	VJ	YN	EL	pzg	avw	wyt	iye
St	15.	III	II	V	06	16	02	GT	YC	EJ	LA	RX	PN	IS	WB	MH	ZV	bhe	xzm	yzk	evp
St	14.	II	I	V	23	03	24	AZ	CJ	WF	UY	SO	QV	MI	NH	DP	GX	fdx	tyj	bmq	typ
St	13.	IV	II	V	03	23	10	CX	KN	JR	DQ	IU	TL	HZ	MP	EP	WB	zfo	bjr	zwx	gvn
St	12.	I	III	II	26	01	18	QB	YE	WN	AI	GJ	TO	HR	PK	PS	CM	upo	anf	tkr	pwz
St	11.	V	I	III	17	13	04	SV	GO	FA	ZR	FN	HI	YK	WT	DE	BJ	vdh	ego	wmy	uti
St	10.	I	V	IV	26	07	16	SW	AQ	NP	PO	VY	UX	MK	CL	HT	ZJ	rpl	anw	vpr	mhn
St	9.	I	III	IV	17	10	15	EH	IK	GK	KZ	SP	UA	LD	CQ	JM	YV	lro	ysq	rhj	tlj
St	8.	V	II	I	23	11	25	QY	OG	ST	HA	CB	WD	EL	JN	VX	IU	aty	mbb	mve	jmz
St	7.	II	III	I	06	12	03	BQ	FS	TH	JE	VK	PI	CU	QA	OD	NM	bho	iwo	xgz	rnr
St	6.	I	IV	V	24	19	01	IR	HQ	NT	WZ	VC	OY	GF	LP	BX	AK	bok	rzw	kzo	ryl
St	5.	II	IV	III	05	22	14	MK	GO	RQ	XT	DW	IA	ZL	SY	FJ	ER	kpk	php	xmo	pfw
St	4.	IV	II	I	15	02	21	KD	FG	CO	PW	BJ	SY	MT	QL	VB	UZ	hjy	nkt	ytn	pvc
St	3.	III	V	IV	03	23	04	DY	CF	WN	OV	QH	UZ	RA	TJ	GL	SM	gpq	fqw	oiy	ruj
St	2.	I	III	V	13	16	01	DR	VJ	PS	EK	IU	HX	AQ	GT	YO	FC	eol	eoi	ywv	sfb
St	1.	II	IV	I	06	17	26	AC	LS	BQ	WN	MY	UV	FJ	PZ	TR	OK				

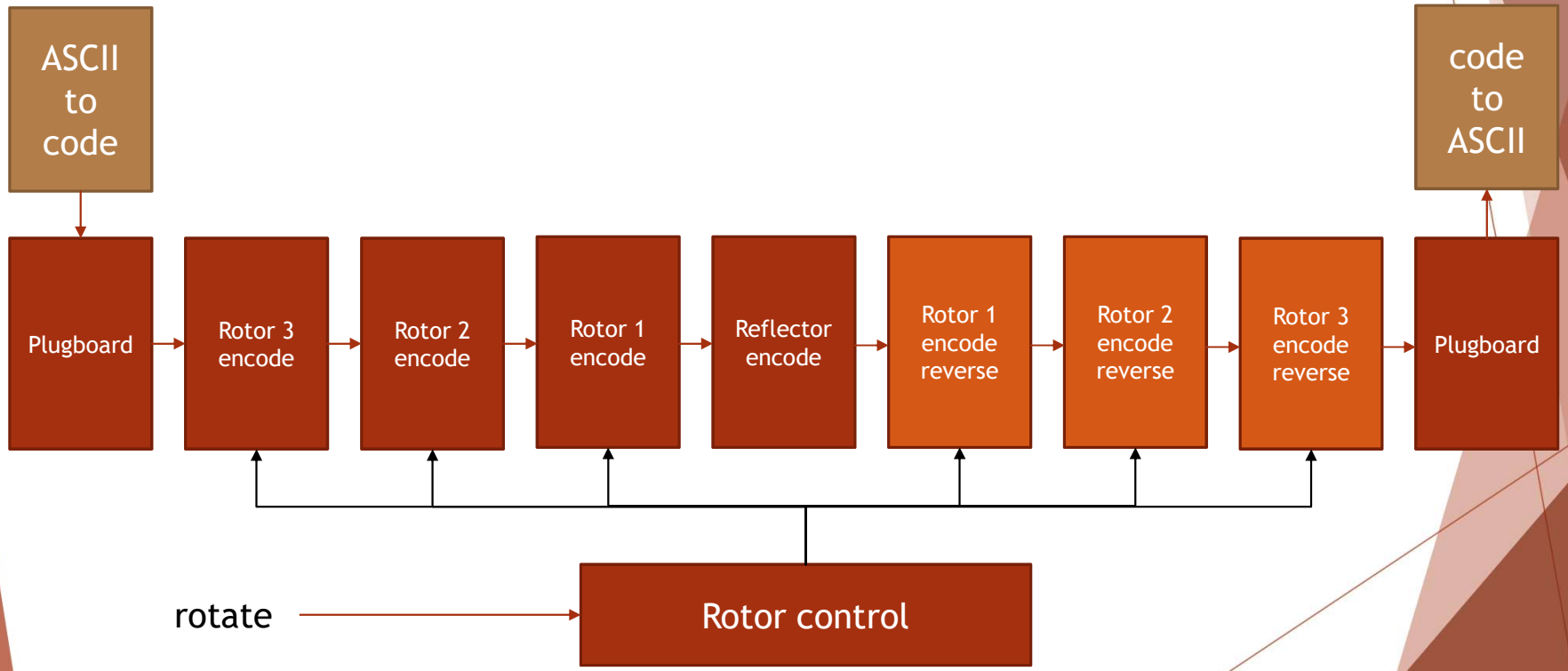
Planning

WEEKS OF PROGRAMMING CAN SAVE YOU HOURS OF PLANNING

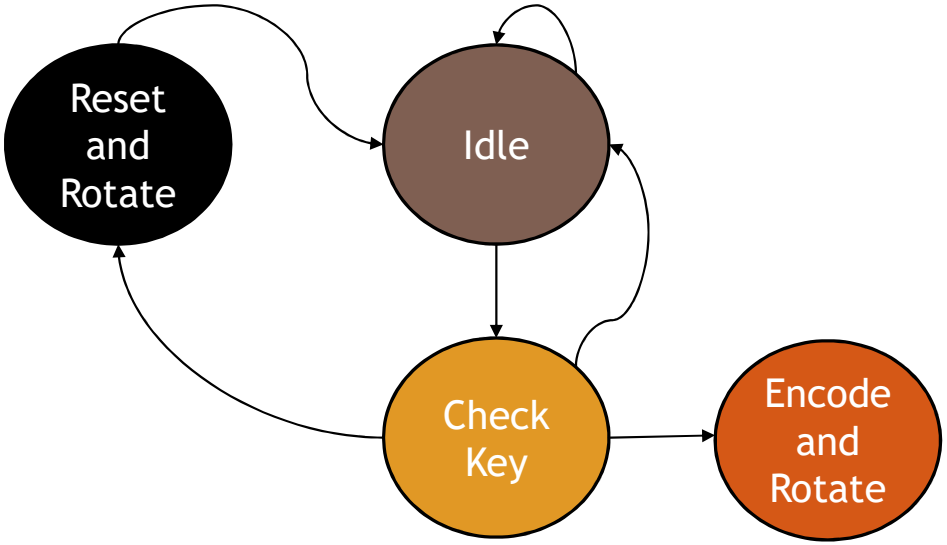
Architecture



Encode section



State machine



Verilog introduction

```
module decodeASCII(input [4:0] code, output [7:0] ascii);
	assign ascii = 8'h41 + code;
endmodule

module encodeASCII(ascii, code, valid);
	input [7:0] ascii;
	output [4:0] code;
	output valid;

	assign valid = ((ascii < 8'h41 || ascii > 8'h5A) && (ascii < 8'h61 || ascii > 8'h7A)) ? 0 : 1;
	assign code = (ascii > 8'h5A) ? ascii - 8'h61 : ascii - 8'h41;
endmodule
```

- Module – Each building block is a module
- Inputs/outputs
- Combinational logic

Complex combinational logic

```
module reflectorEncode (code, val, reflector_type);
        input [4:0] code;
        output reg [4:0] val;
        input reflector_type;

   always @*
   begin
          if (reflector_type == 1'b0)
          begin
                    // Reflector B
                    case (code)
                              0 : val = "Y" - 8'h41;
                              1 : val = "R" - 8'h41;
                              2 : val = "U" - 8'h41;
                              ....
                              24: val = "A" - 8'h41;
                              25: val = "T" - 8'h41;
                    endcase
          end
          else
          begin
                   // Reflector C
                   case (code)
                              0 : val = "F" - 8'h41;
                              1 : val = "V" - 8'h41;
                               ....
                              24: val = "H" - 8'h41;
                              25: val = "L" - 8'h41;
                   endcase
          end
   end
endmodule
```

```
'Y','R','U','H','Q','S','L','D','P','X','N','G','O','K','M','I','E','B','F','Z','C','W','V','J','A','T'  // M3 B
'F','V','P','J','I','A','O','Y','E','D','R','Z','X','W','G','C','T','K','U','Q','S','B','N','M','H','L'  // M3 C
```

Memory

```
module rotorEncode #(parameter REVERSE = 0) (code, rotor_type, val);
    input [4:0] code;
    output reg [4:0] val;
    input [2:0] rotor_type;

    parameter MEM_INIT_FILE = "rotors.mem";

    reg [4:0] rotor_data[0:415];

    initial
        if (MEM_INIT_FILE != "")
            $readmemh(MEM_INIT_FILE, rotor_data);

    always @*
        val = rotor_data[((REVERSE) ? 208 : 0) + rotor_type*26 + code];

endmodule
```

```
'E','K','M','F','L','G','D','Q','V','Z','N','T','O','W','Y','H','X','U','S','P','A','I','B','R','C','J'  // Rotor I
'A','J','D','K','S','I','R','U','X','B','L','H','W','T','M','C','Q','G','Z','N','P','Y','F','V','O','E'  // Rotor II
'B','D','F','H','J','L','C','P','R','T','X','V','Z','N','Y','E','I','W','G','A','K','M','U','S','Q','O'  // Rotor III
'E','S','O','V','P','Z','J','A','Y','Q','U','I','R','H','X','L','N','F','T','G','K','D','C','M','W','B'  // Rotor IV
'V','Z','B','R','G','I','T','Y','U','P','S','D','N','H','L','X','A','W','M','J','Q','O','F','E','C','K'  // Rotor V
'J','P','G','V','O','U','M','F','Y','Q','B','E','N','H','Z','R','D','K','A','S','X','L','I','C','T','W'  // Rotor VI
'N','Z','J','H','G','R','C','X','M','Y','S','W','B','O','U','F','A','I','V','L','P','E','K','Q','D','T'  // Rotor VII
'F','K','Q','H','T','L','X','O','C','B','J','S','P','D','Z','R','A','M','E','W','N','I','U','Y','G','V'  // Rotor VIII
```

rotors.mem content

```
04
0a
0c
05
0b
06
03
10
15
19
0d
13
0e
16
18
07
17
14
12
0f
00
08
```

Using registers, and clocked changes

```
module hex_to_7seg
  (
   input       i_Clk,
   input [3:0] i_Value,
   output   o_Segment_A,   output  o_Segment_B,   output  o_Segment_C,   output   o_Segment_D,
   output   o_Segment_E,   output   o_Segment_F,   output  o_Segment_G   );

  reg [6:0]    out = 7'b0000000;

  always @(posedge i_Clk)
    begin
      case (i_Value)
        4'b0000 : out <= 7'b0000001;
        4'b0001 : out <= 7'b1001111;
        …..
      endcase
   end

  assign o_Segment_A = out[6];
  assign o_Segment_B = out[5];
  assign o_Segment_C = out[4];
  assign o_Segment_D = out[3];
  assign o_Segment_E = out[2];
  assign o_Segment_F = out[1];
  assign o_Segment_G = out[0];

endmodule
```

Sequential logic and instantiation

```
module rotor (
   input clock,
   output reg [4:0] rotor1,  output reg [4:0] rotor2,   output reg [4:0] rotor3,
   input reset, input rotate,  input [2:0] rotor_type_1, input [2:0] rotor_type_2, input [2:0] rotor_type_3,
   input [4:0] rotor_start_1,  input [4:0] rotor_start_2, input [4:0] rotor_start_3
);
        wire knock1;
        wire knock2;

        reg prev_rotate = 1'b0;
        reg prev_knock1 = 1'b0;
        reg prev_knock2 = 1'b0;

        checkKnockpoints checker3(.position(rotor3), .knockpoint(knock1), .rotor_type(rotor_type_3));
        checkKnockpoints checker2(.position(rotor2), .knockpoint(knock2), .rotor_type(rotor_type_2));

        always @(posedge clock)
        begin
                if (reset)
                begin
                        rotor1 <= rotor_start_1;
                        rotor2 <= rotor_start_2;
                        rotor3 <= rotor_start_3;
                        prev_rotate <= 0;
                        prev_knock1 <= 0;
                        prev_knock2 <= 0;
                end  else begin
                        if ((prev_rotate==0) && (rotate==1)) rotor3 <= (rotor3 == 25) ? 0 : rotor3 + 1;
                        if ((prev_knock1==0) && (knock1==1)) rotor2 <= (rotor2 == 25) ? 0 : rotor2 + 1;
                        if ((prev_knock2==0) && (knock2==1)) rotor1 <= (rotor1 == 25) ? 0 : rotor1 + 1;
                        prev_rotate <= rotate;
                        prev_knock1 <= knock1;
                        prev_knock2 <= knock2;
                end
        end
endmodule
```

Test bench

```
module testrotor();
        reg [2:0] rotor_type_3 = 3'b010;  reg [2:0] rotor_type_2 = 3'b001;  reg [2:0] rotor_type_1 = 3'b000;
        reg [4:0] rotor_start_3 = 5'b00000; reg [4:0] rotor_start_2 = 5'b00000; reg [4:0] rotor_start_1 = 5'b00000;
        reg [4:0] ring_position_3 = 5'b00000; reg [4:0] ring_position_2 = 5'b00000; reg [4:0] ring_position_1 = 5'b00000;
        reg reflector_type = 1'b0;
        reg i_clock = 0;
        reg reset;
        reg rotate = 0;
        wire [4:0] rotor1; wire [4:0] rotor2;wire [4:0] rotor3;
        integer i;

        rotor rotorcontrol(.clock(i_clock),.rotor1(rotor1),.rotor2(rotor2),.rotor3(rotor3),
              .reset(reset),.rotate(rotate),
              .rotor_type_1(rotor_type_1),.rotor_type_2(rotor_type_2),.rotor_type_3(rotor_type_3),
              .rotor_start_1(rotor_start_1),.rotor_start_2(rotor_start_2),.rotor_start_3(rotor_start_3));
   always
                #(5) i_clock <= !i_clock;
   initial
        begin
                $dumpfile("testrotor.vcd");
                $dumpvars(0,testrotor);

                #5
                reset = 1;
                #10
                reset = 0;
                for (i = 0; i < 30; i = i + 1)
                begin
                  #10  rotate = 1;
                  #10  rotate = 0;
                  #10  $display("Rotors [%c] [%c] [%c]",rotor1+65,rotor2+65,rotor3+65);
                end
                $finish;
        end
endmodule
```

Running tests

- iverilog testrotor.v rotor.v checkKnockpoints.v
- vvp a.out

VCD info: dumpfile testrotor.vcd opened for output.
Rotors [A] [A] [B]
Rotors [A] [A] [C]
Rotors [A] [A] [D]
Rotors [A] [A] [E]
Rotors [A] [A] [F]
Rotors [A] [A] [G]
Rotors [A] [A] [H]
Rotors [A] [A] [I]
Rotors [A] [A] [J]
Rotors [A] [A] [K]
Rotors [A] [A] [L]
Rotors [A] [A] [M]
Rotors [A] [A] [N]
Rotors [A] [A] [O]
Rotors [A] [A] [P]
Rotors [A] [A] [Q]
Rotors [A] [A] [R]
Rotors [A] [A] [S]
Rotors [A] [A] [T]
Rotors [A] [A] [U]
Rotors [A] [A] [V]
Rotors [A] [B] [W]
Rotors [A] [B] [X]
Rotors [A] [B] [Y]
Rotors [A] [B] [Z]
Rotors [A] [B] [A]
Rotors [A] [B] [B]
Rotors [A] [B] [C]
Rotors [A] [B] [D]
Rotors [A] [B] [E]

State machine

```
always @(posedge i_clock)
begin
        case (state)
                STATE_RESET : // Reset and rotate
                        begin
                                reset <= 1'b0;
                                state <= STATE_IDLE;
                                o_ready <= 1'b0;
                                o_outputData <= 8'b00000000;
                                rotate  <= 1'b1;
                        end
                STATE_IDLE : // Idle –wait for key
                        begin
                                o_ready <= 1'b0;
                                state   <= i_ready ? STATE_CHECKKEY : STATE_IDLE;
                        end
                STATE_CHECKKEY : // Check key value (cut down version)
                        begin
                                o_ready <= 1'b0;
                                rotate  <= 1'b0;
                                state   <= (valid) ? STATE_ENCODE : STATE_IDLE;
                        end
                STATE_ENCODE : // Encode and rotate for next
                        begin
                                rotate  <= 1'b1;
                                o_ready <= 1'b1;
                                o_outputData <= final_ascii;
                                state  <= STATE_IDLE;
                        end
        endcase
End
```

Testing using C++ (Verilator)

```cpp
#include <iostream>
#include <memory>
#include "Venigma.h"
#include "verilated.h"

int main(int argc, char **argv, char **env)
{
        Verilated::commandArgs(argc, argv);
        std::unique_ptr<Venigma> top = std::make_unique<Venigma>();

        top->i_clock = 0;
        top->i_ready = 0;
        top->eval();

        top->i_clock ^= 1; top->eval();
        top->i_clock ^= 1; top->eval();

        for (int i=0;i<20;i++)
        {
                top->i_inputData = 'A';
                top->i_ready = 1;
                top->i_clock ^= 1; top->eval();
                top->i_ready = 0;
                top->i_clock ^= 1; top->eval();

                while (top->o_ready == 0 || top->o_outputData == 0)
                {
                        top->i_clock ^= 1; top->eval();
                }

                cout << "output is " << top->o_outputData << std::endl;

                top->i_clock ^= 1; top->eval();
        }
}
```

Running verilator and building test

- verilator -Wall --cc enigma.vlt state_machine.v encode.v rotor.v encodeASCII.v decodeASCII.v rotorEncode.v reflectorEncode.v plugboardEncode.v checkKnockpoints.v --exe main.cpp
- make -C obj_dir -j -f Venigma.mk Venigma

Output or test running :

```
obj_dir/Venigma
output is B
output is D
output is Z
output is G
output is O
output is W
output is C
output is X
output is L
output is T
output is K
output is S
output is B
output is T
output is M
output is C
output is D
output is L
output is P
output is B
```

Constraints file

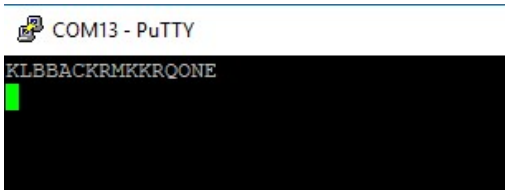
```
### Main FPGA Clock
set_io i_Clk 15

### LED Pins:
set_io o_LED_1 56
set_io o_LED_2 57
set_io o_LED_3 59
set_io o_LED_4 60

### 7 Segment Outputs
set_io o_Segment1_A 3
set_io o_Segment1_B 4
set_io o_Segment1_C 93
set_io o_Segment1_D 91
set_io o_Segment1_E 90
set_io o_Segment1_F 1
set_io o_Segment1_G 2
set_io o_Segment2_A 100
set_io o_Segment2_B 99
set_io o_Segment2_C 97
set_io o_Segment2_D 95
set_io o_Segment2_E 94
set_io o_Segment2_F 8
set_io o_Segment2_G 96

## UART Outputs
set_io i_UART_RX 73
set_io o_UART_TX 74
```

Compile for Go Board

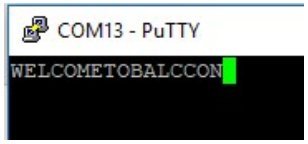
- yosys -q -p "synth_ice40 -abc2 -nocarry -top enigma_top -blif enigma.blif" enigma_top.v state_machine.v encode.v rotor.v encodeASCII.v decodeASCII.v rotorEncode.v reflectorEncode.v plugboardEncode.v checkKnockpoints.v hex_to_7seg.v uart_tx.v uart_rx.v
- arachne-pnr -d 1k -P vq100 -p Go_Board_Constraints.pcf enigma.blif -o enigma.txt
- icepack enigma.txt enigma.bin
- icetime -d hx1k -P vq100 enigma.txt
- iceprog enigma.bin

Running example

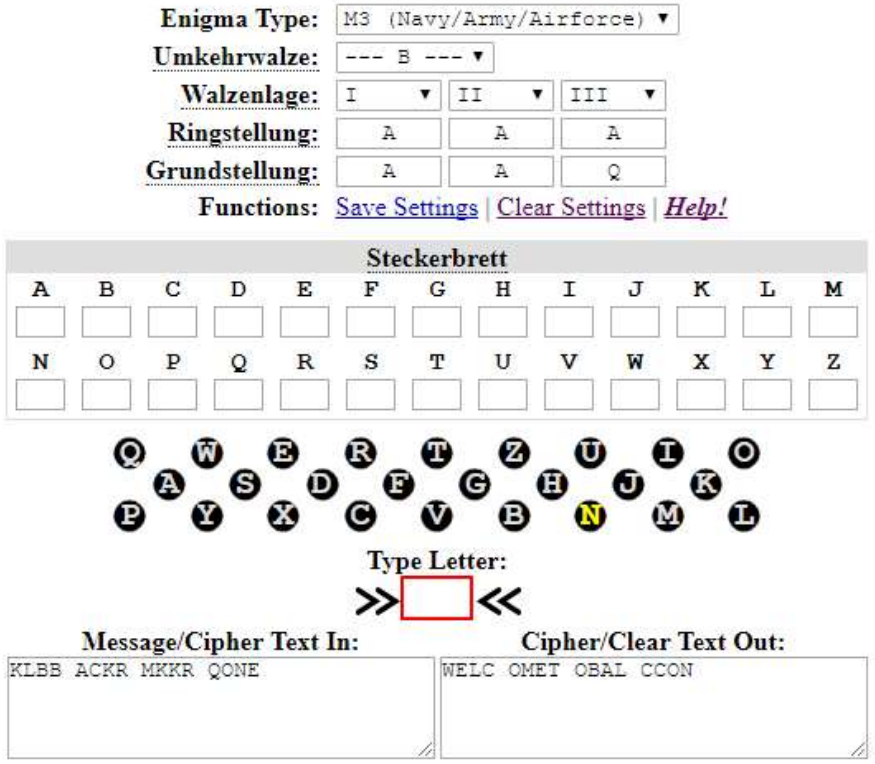
Typing on terminal original text and we get encoded text back



Pressing ESC and resetting terminal
Typing encoded text now, and we get decoded text now



Doing check with same message to see if we did a good job



Live demo

More info

- Full source https://github.com/mmicko/enigmaFPGA
- nandland channel on YouTube with tutorials
- https://www.nandland.com/ to order Go Board

  (code ENIGMA for 10% off till end of September)
- http://www.fpga4student.com/ for more nice tutorials