g04- Lab 5 – Enigma Machine

Malkolm Alburquenque – 260562740

Ryan Xu – 260553466

The g04_enigma_machine is a high level encryption mechanism allowing important messages to be sent from one user to another. The encryption mechanism changes every day, thus making it nearly impossible to decode. After powering on the Enigma machine there will be a series of inputs required. These inputs are to be prearranged between the parties sending encrypted messages. Without all 10 inputs being initialized correctly, the message will forever remain a mystery.

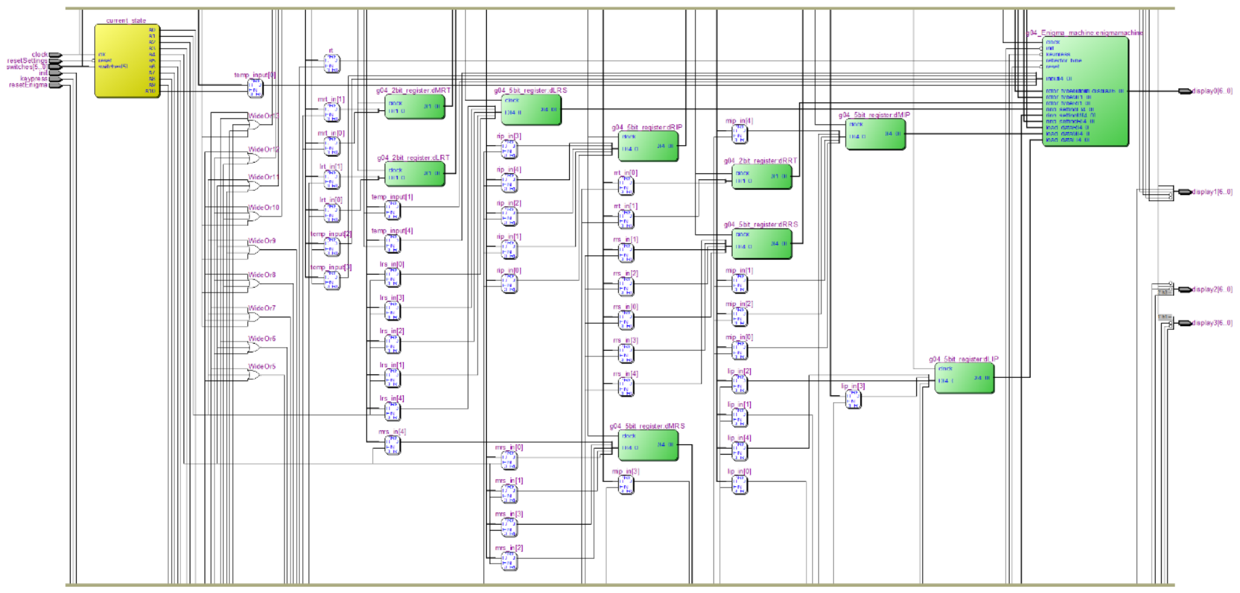


Fig. 1: Above is the block diagram for the complete enigma machine with FPGA implementation.

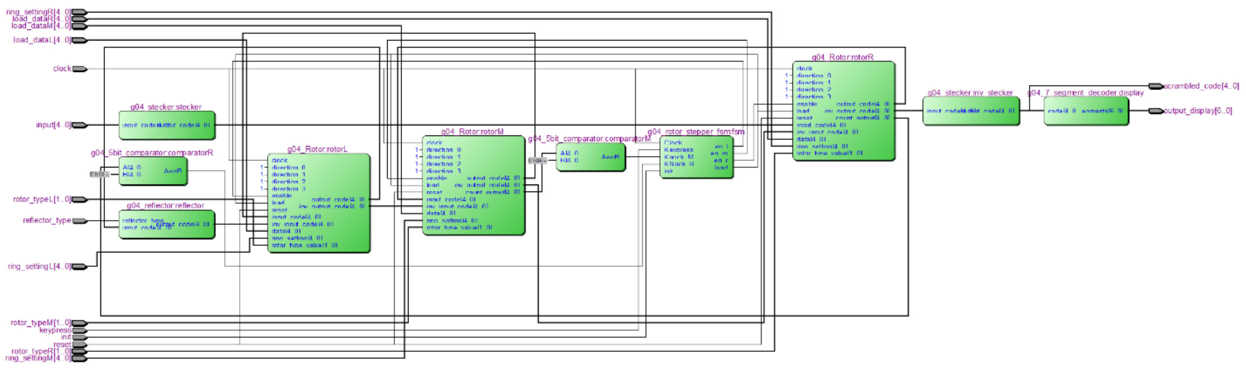


Fig. 2: Block diagram for the component labelled "g04_Enigma_machine:enigmamachine" from Fig. 1.

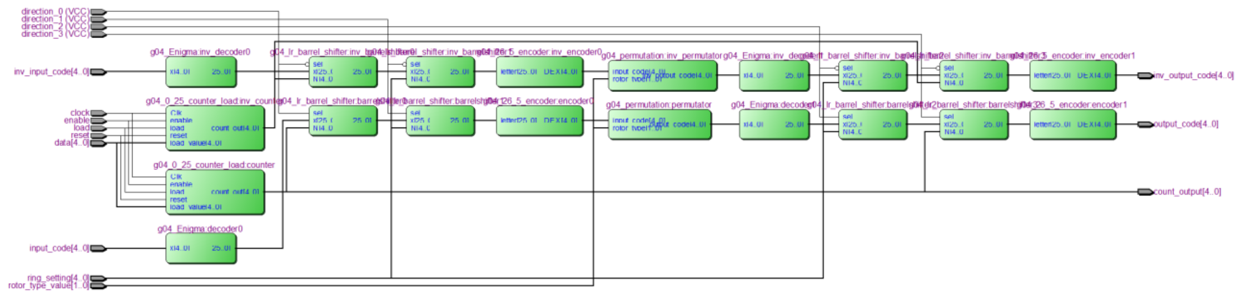


Fig. 3: Block diagram for the component labelled "g04_Rotor:RotorR" from Fig. 2.

Fig. 1 shows the complete implementation of the Enigma with the Altera Cyclone II FPGA board (a description on this will be given later). Fig. 2 shows the complete concept and design of the Enigma machine. As you can see from Fig. 2, the Enigma machine consists of three Rotors (labelled "RotorR", "RotorM", and "RotorL"), two Steckers (labelled "Stecker" and "inv_Stecker"), a Reflector, a Finite State Machine (labelled "rotor_stepper_fsm") , two Comparators (labelled "ComparatorR" and "ComparatorM"), and a 7 Segment Decoder.

The 7 Segment Decoder takes a 5 bit input representing a letter (with "A" being of value 0 to "Z" of value 25) and converts it to a 7 bit output representing the input visually according to Fig. 4.

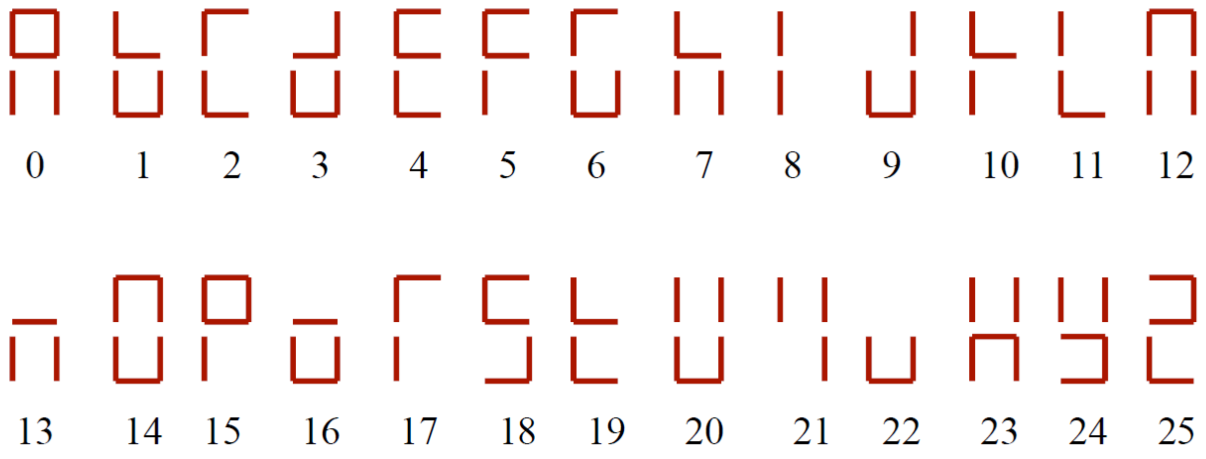


Fig. 4: Above is the 7 segment display for the entire English alphabet. The number underneath each letter represents the matching input to the 7 Segment Decoder.

The Reflector circuit takes as input a 1 bit signal "reflector_type" which chooses one of two reflectors, and a 5 bit "input_code" and outputs a 5 bit "output_code". The function of the Reflector is to pair two letters together such that when one is received in the input, the other becomes the output. The "reflector_type" chooses "Reflector B" when 0 and "Reflector C" when 1. The pairs for each reflector type is shown below in Fig. 5.

reflector B	(AY) (BR) (CU) (DH) (EQ) (FS) (GL) (IP) (JX) (KN) (MO) (TZ) (VW)
reflector C	(AF) (BV) (CP) (DJ) (EI) (GO) (HY) (KR) (LZ) (MX) (NW) (TQ) (SU)

Fig. 5: Displays the pairs set in reflector B as well as reflector C (i.e In reflector B, A maps to Y and Y maps to A).

The Stecker (and inv_Stecker) takes a 5 bit input “input_code” and outputs a 5 bit output “output_code”. The purpose of the Stecker is similar to the reflector, where it takes a letter as an input, potentially pairs it with another letter, and assigns the pair to the output. The Stecker mapping we have chosen is seen below in Fig. 6.

Input	A	B	C	D	E	F	G	H	I	J	K - Z
Output	B	A	D	C	F	E	H	G	J	I	Input

Fig. 6: Above is the Stecker we have chosen to implement. Only inputs A-J are effected by the Stecker. Inputs K-Z are ignored and assigned immediately to the output.

The Rotor Stepper FSM’s function is to rotate the rotors at specific times to increase complexity. It has 1 bit inputs “Clock”, “Keypress”, “init”, “Knock_M”, “Knock_R”, and 1 bit outputs “en_l”, “en_m”, “en_r”, and “load”. The Rotor Stepper FSM has 8 states shown below in Fig. 7.

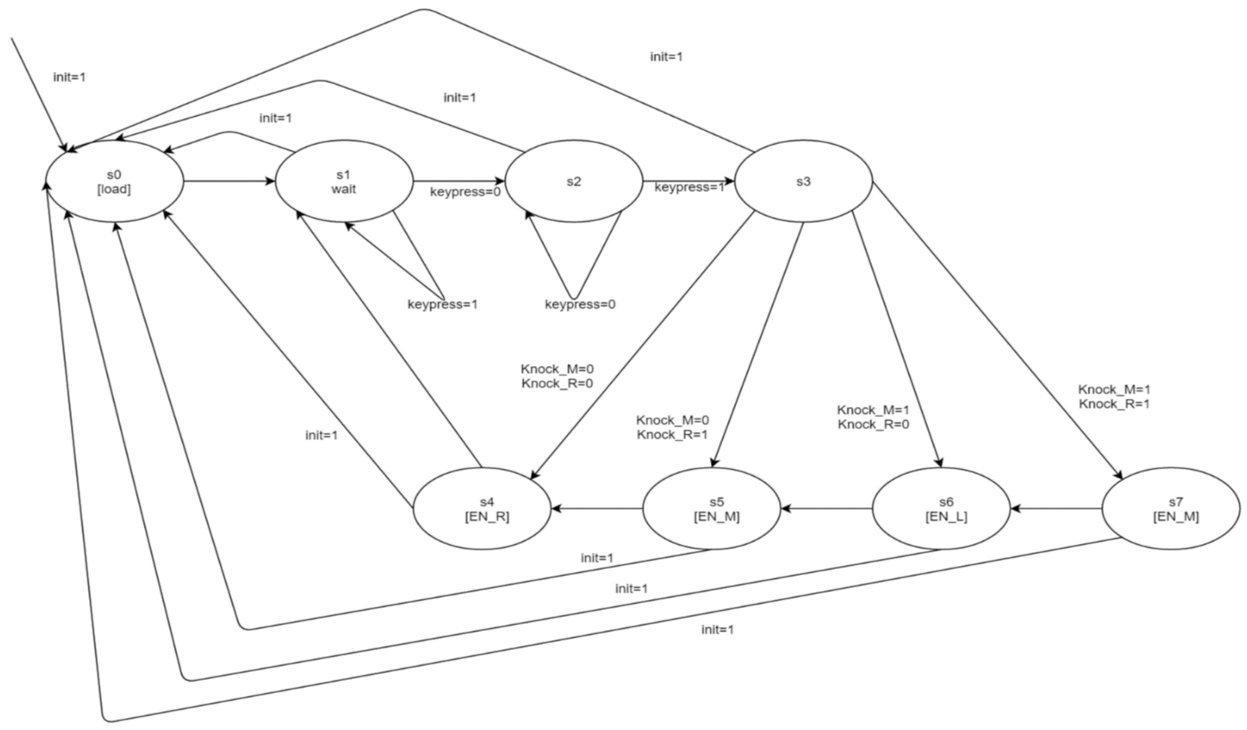
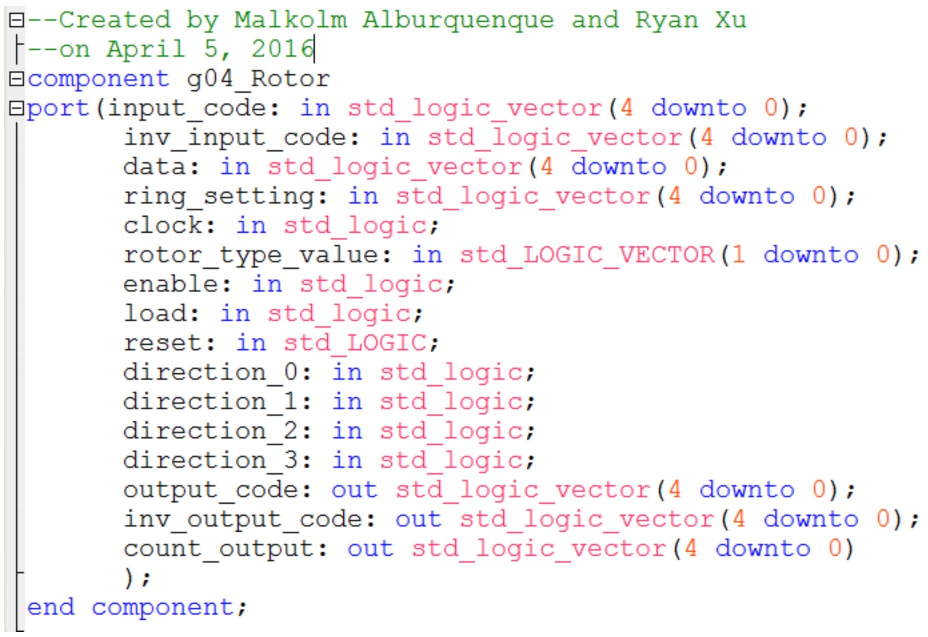


Fig. 7: State diagram of the Rotor Stepper FSM.

The functionality of the Rotor Stepper FSM is clearly displayed in Fig. 7. The outputs "en_l", "en_m", "en_r" are passed into the input of RotorL, RotorM, RotorR respectively. Depending on which "Notch_M" or "Notch_R" are high, the appropriate enable signal will be high, thus shifting a rotor, greatly increasing complexity thus improving the overall encryption of the Enigma machine.

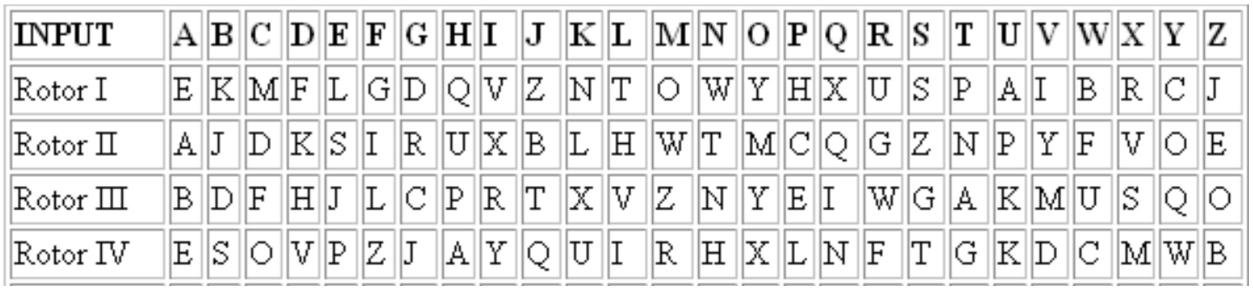
To determine when the notch values are high, two 5 bit comparator circuits will be used (one for Notch_M and another for Notch_R). The input to "ComparatorR" will be a 5 bit input "Notch_R" which represents the letter at which the "RotorR" will rotate, and the current letter of the same rotor. "ComparatorM" will be used in the same manner. The output of the comparators is a 1 bit signal which maps to the appropriate knock value in the Rotor Stepper FSM.



```vhdl
--Created by Malkolm Alburquenque and Ryan Xu
--on April 5, 2016
component g04_Rotor
port(input_code: in std_logic_vector(4 downto 0);
      inv_input_code: in std_logic_vector(4 downto 0);
      data: in std_logic_vector(4 downto 0);
      ring_setting: in std_logic_vector(4 downto 0);
      clock: in std_logic;
      rotor_type_value: in std_LOGIC_VECTOR(1 downto 0);
      enable: in std_logic;
      load: in std_logic;
      reset: in std_LOGIC;
      direction_0: in std_logic;
      direction_1: in std_logic;
      direction_2: in std_logic;
      direction_3: in std_logic;
      output_code: out std_logic_vector(4 downto 0);
      inv_output_code: out std_logic_vector(4 downto 0);
      count_output: out std_logic_vector(4 downto 0)
      );
end component;
```

Fig. 8: The VHDL description of the inputs and outputs of the rotor component.

Fig. 8 shows the inputs and outputs of one rotor. The inputs direction_0, direction_1, direction_2, direction_3, ring_setting, and rotor_type_value all contribute to the encryption and must match the values of the person you are communicating with. Fig. 3 shows the block diagram of RotorR (all rotors function in the same matter). As you can see in Fig. 3 for every component within the Rotor, there is a matching inverse component to account for the portion of the enigma machine post reflector. The function of the rotor is to implement the permutation circuit based off of the chosen rotor_type. The permutation circuit is a more complicated version of the reflector circuit. Instead of having a direct pair, the letters map to each other but no in pairs. See Fig. 9 for the permutation mapping depending on the 2 bit "rotor_type".



INPUT	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Rotor I	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 II	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 III	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 IV	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

Fig. 9 Mappings of the permutation circuit for each of the four rotor_types.

The entire enigma machine is then mapped to match Fig. 2, thus completing the circuit. When the user inputs a letter, it will be scrambled 9 times before it yields an output.

In order for users to choose different settings (e.g. ring setting for each rotor), we implemented a finite state machine contains 11 states (shown in Fig. 1). State s0 to s9 are reserved for machine settings, including ring settings, rotor type and initial position for each rotor and reflector type. We used D flip flop as registers to store those setting. Settings in different states are stored in independent registers to be used in state s10. After all settings have been declared, the system will stay at state s10 unless a reset signal is received. At state s10, input from the board will be assigned to the input of component "enigmamachine". Corresponding output will then be displayed.

User interface:

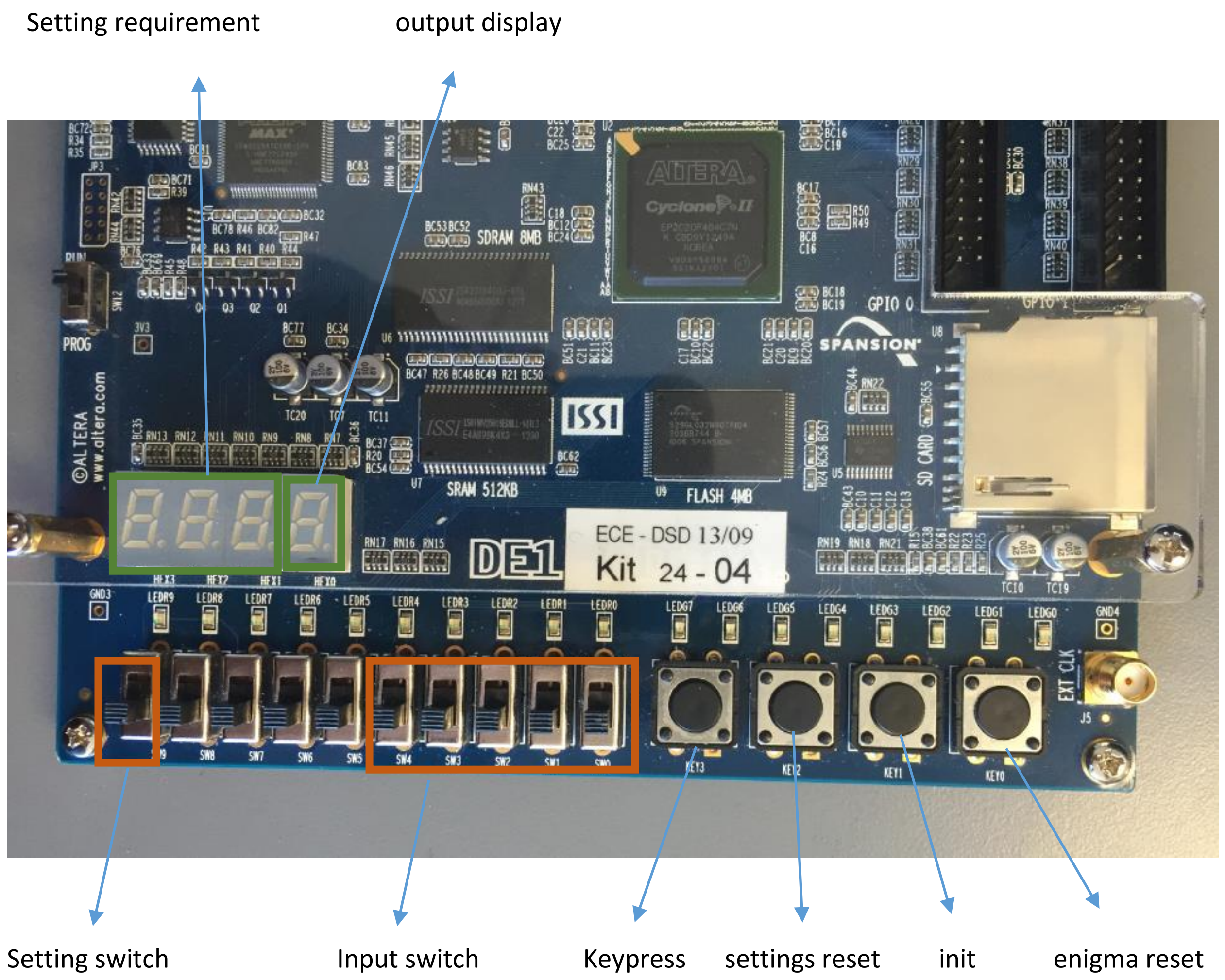


Fig. 10 Board view

After the board has been powered up and the program has been loaded. Setting requirement section will light up. It indicates what input, if any, needs to be entered.

LED display Bitwise requirement	LRT 2 bits	LRS 5 bits	LIP 5 bits	MRT 2 bits	MRS 5 bits	MIP 5 bits	RRT 5 bits	RRS 2 bits	RIP 5 bits	RET 1 bit
LED display translation	Left Rotor Type	Left Ring Setting	Left Initial Position	Middle Rotor Type	Middle Ring Setting	Middle Initial Position	Right Rotor Type	Right Ring Setting	Right Initial Position	Reflector Type

Fig. 11 Machine input requirements

Initially, LED should display LRT (refer to Fig.12). According to Fig.11, the machine now requires 2 bit input. All input switches being at ‘0’ as default, therefore “00” will be assigned to the rotor type of the left rotor. Users are allowed to select different inputs by assigning SW1 and SW0 different values. Setting will be stored automatically and only values after the final change will be recorded.

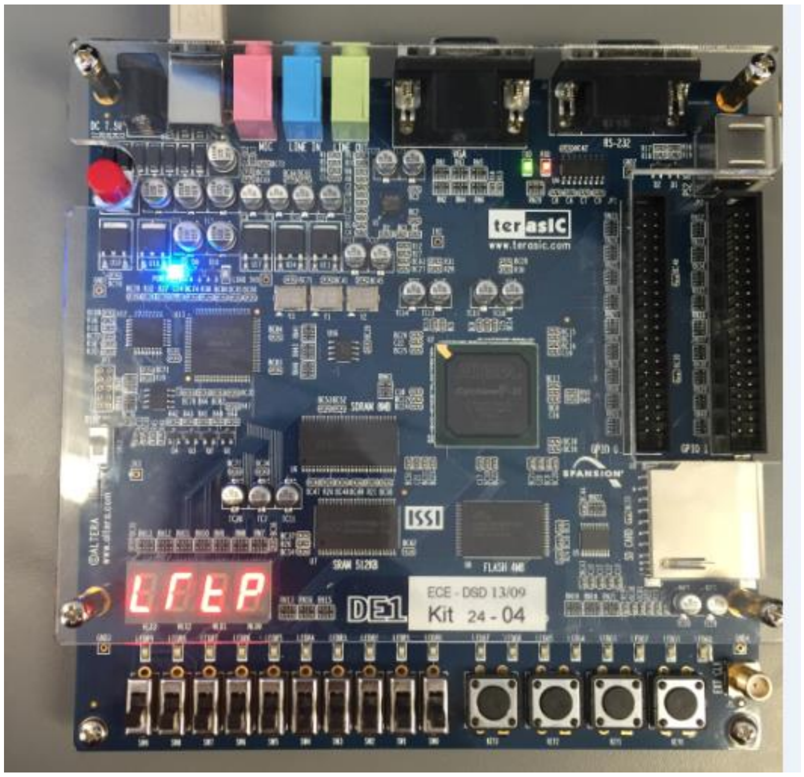


Fig. 12: system requires setting for LRT(left rotor type).

After inputting selected values to left rotor type, toggle “setting switch” from ‘0’ to ‘1’ and proceed in similar manner. Display shown in Fig.13 should appear after the switch value change.

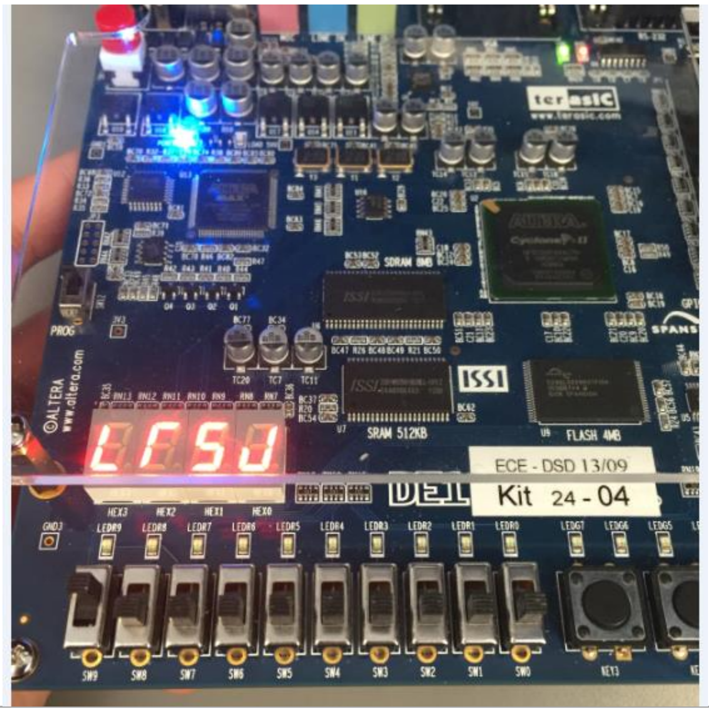


Fig. 13: System requires setting for LRS (left ring setting).

Toggle “setting switch” from ‘1’ to ‘0’ to move to the next setting (LIP). Keep toggling “setting switch” until all 10 settings have be initialized. Note that input may require 1 bit, 2 bit or 5 bit and the length needs to be respected (refer to fig.11 for more information). When the user have completed all 10 settings, “Setting requirements” LEDs will go dark. At this point the enigma machine is ready to take inputs.

Insert a letter by using its corresponding 5 bit representation on input switches. Press “Keypress” pushbutton to scramble the code. The scrambled code will appear on the right most LED “output display”.

System Testing:

We tested each component separately and make sure each one works before we put everything together. Everything done before Lab 5 has been verified with either ModelSim or the FPGA board. We created 2 new circuits “g04_stecker” and “g04_reflector”. We tested all 26 possible inputs in both circuits and received the expected outputs in ModelSim.

After we finished building the rotor circuit, we ran some tests in ModelSim with known inputs. We compared every single intermediate signal to expected output at each stage to verify the correct mapping. In the end, the rotor circuit works as expected.

We then built the complete enigma machine contains components from Lab 1 to Lab 5. We used the floor plan provided in the lab instruction (Fig. 14) as our guideline. We tested the circuit’s functionality in ModelSim. When settings remain unchanged, if we use the scrambled code as our new input code, the enigma machine will return our original input code as output. Thus our rotor circuit works as expected.

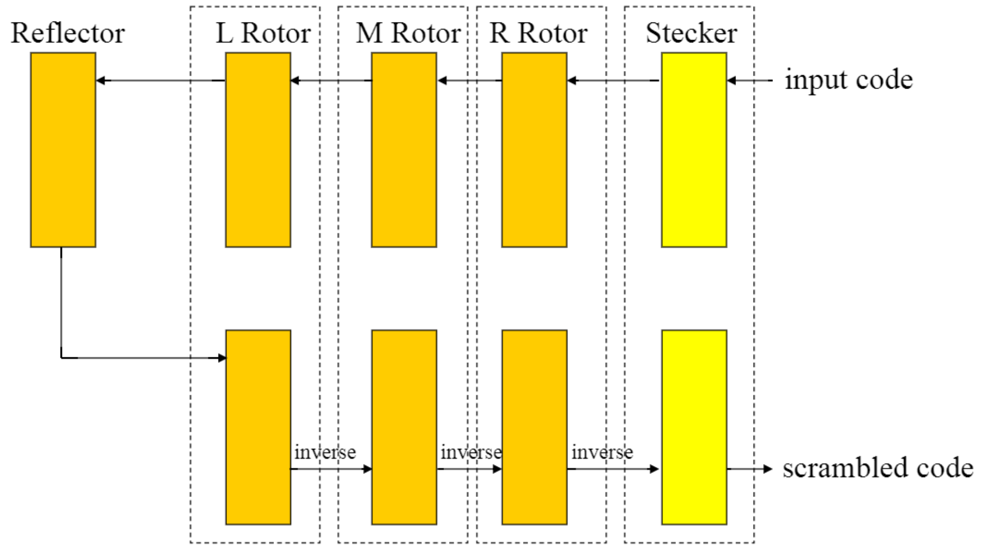
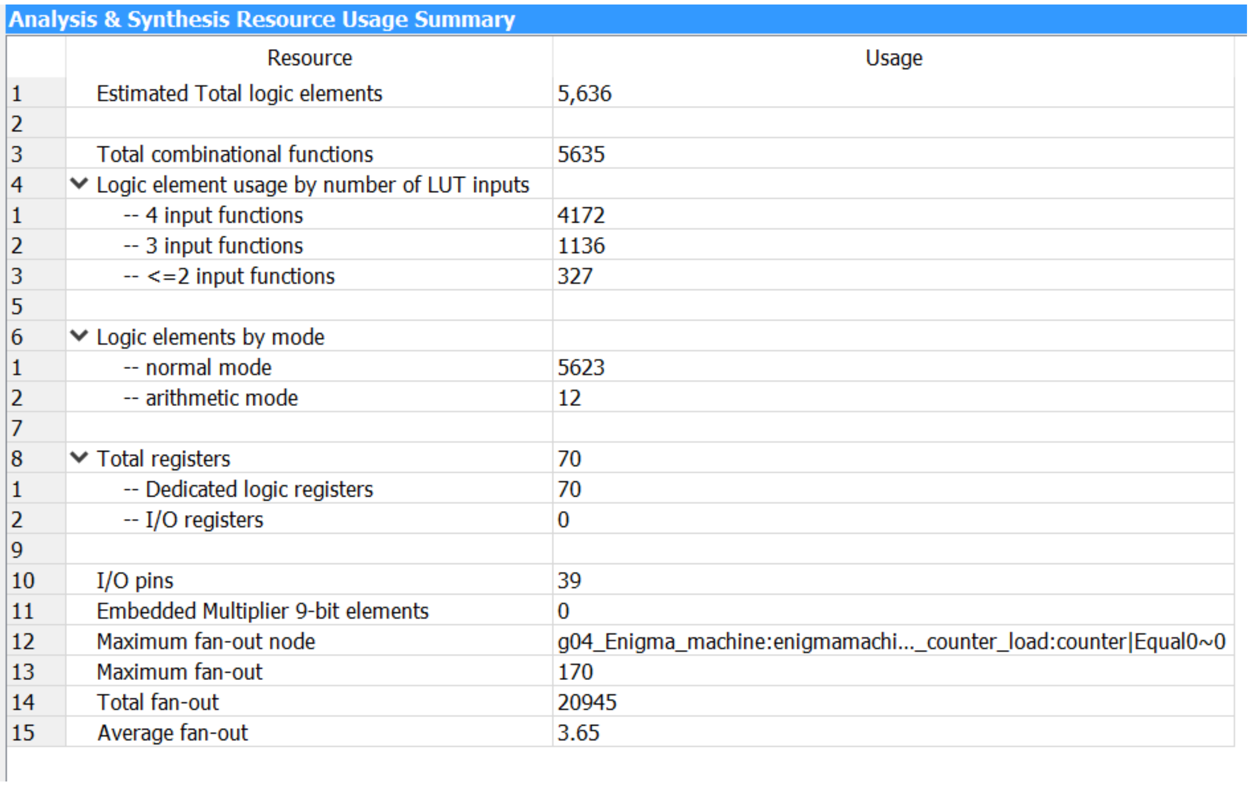


Fig. 14 Complete enigma machine floor plan

Finally we implemented user interface to the enigma machine. The way we checked it is the same way that we demonstrated to the TA. We wrote down the initial 10 settings for the enigma machine. Then we used "keypress" button to encode 'A' through 'D'. We recorded every output and reset the machine. We input the exact same machine settings as the encoding process and entered 4 recorded scrambled code sequentially. We used the pushbutton to encode them one at a time and we observed that output was indeed ABCD. We repeated the same procedure for all 26 letters on board and multiple different initial settings to find the complete circuit with user interface works correctly.

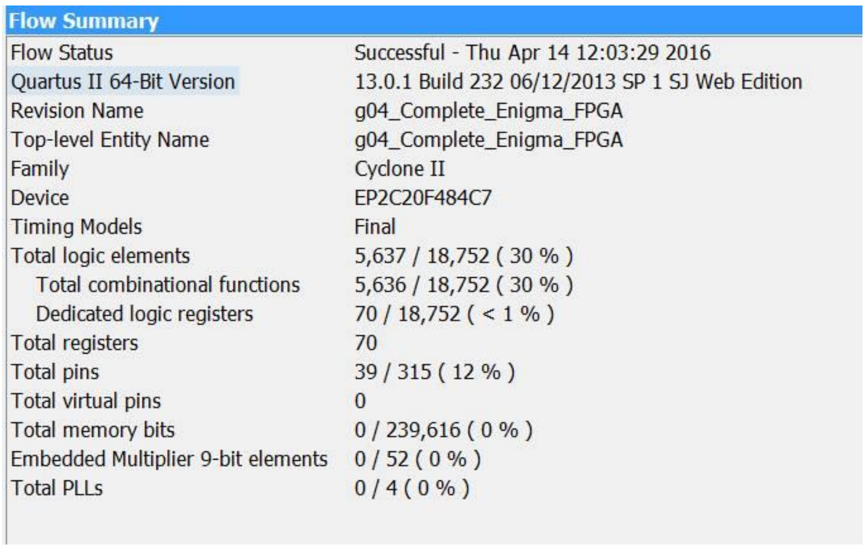
The following section is the summary of FPGA resource utilization and timing analysis.



Analysis & Synthesis Resource Usage Summary

	Resource	Usage
1	Estimated Total logic elements	5,636
2		
3	Total combinational functions	5635
4	Logic element usage by number of LUT inputs	
1	-- 4 input functions	4172
2	-- 3 input functions	1136
3	-- <=2 input functions	327
5		
6	Logic elements by mode	
1	-- normal mode	5623
2	-- arithmetic mode	12
7		
8	Total registers	70
1	-- Dedicated logic registers	70
2	-- I/O registers	0
9		
10	I/O pins	39
11	Embedded Multiplier 9-bit elements	0
12	Maximum fan-out node	g04_Enigma_machine:enigmamachi..._counter_load:counter|Equal0~0
13	Maximum fan-out	170
14	Total fan-out	20945
15	Average fan-out	3.65

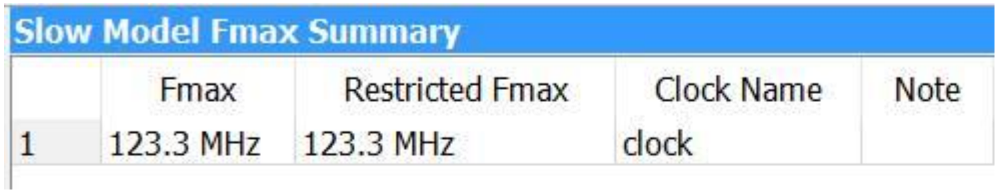
Fig. 15 FPGA resource utilization



Flow Summary

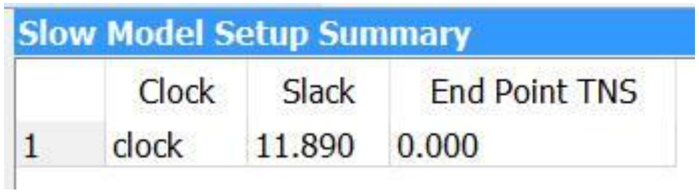
Flow Status	Successful - Thu Apr 14 12:03:29 2016
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	g04_Complete_Enigma_FPGA
Top-level Entity Name	g04_Complete_Enigma_FPGA
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Total logic elements	5,637 / 18,752 (30 %)
Total combinational functions	5,636 / 18,752 (30 %)
Dedicated logic registers	70 / 18,752 (< 1 %)
Total registers	70
Total pins	39 / 315 (12 %)
Total virtual pins	0
Total memory bits	0 / 239,616 (0 %)
Embedded Multiplier 9-bit elements	0 / 52 (0 %)
Total PLLs	0 / 4 (0 %)

Fig. 16 Flow summary



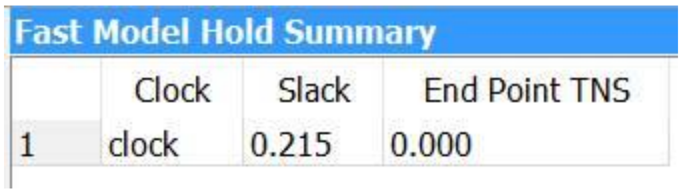
Slow Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	123.3 MHz	123.3 MHz	clock	



Slow Model Setup Summary

	Clock	Slack	End Point TNS
1	clock	11.890	0.000



Fast Model Hold Summary

	Clock	Slack	End Point TNS
1	clock	0.215	0.000

Fig. 17 Timing analysis with 50 MHz clock

Conclusion:

Since we are given the floor plan of the completed enigma machine, the implementation part is fairly straightforward. The challenging part was to understand how the circuit should behave. Outputs are scrambled multiple times and often the encoding process is time sensitive (e.g. rotor behaviour is linked to the counter). At the same time, the instruction did not require to test the rotor functionality before moving on. Therefore we had a hard time debugging the complete system where our rotors failed to yield any signal. We had to go back and test the rotor explicitly and it turned out that we forgot to update the sensitivity list of the barrel shifter circuit after changing it. Other difficulties include making a decision on how to implement the user interface. The board has very limited UI ports. Thus we spent quite some time on coming up a way to support different functions as well as to keep the UI as simple as possible.

Enhancement:

To enhance the difficulty to hack our system, the first thing we could do is to increase the matching pairs of the Stecker circuit. The system has 5 matching pairs at the moment, which corresponding to 5,019,589,575 possible combinations. Theoretically the possible combination can go up to 205,552,193,100,000, if 11 pairs are used. (source: http://www.cryptomuseum.com/crypto/enigma/working.htm, Arthur Bauer, Funkpeilung als alliierte Waffe gegen Deutsche U-Boote 1939-1945). This will for sure enhance the security of the code. It is also possible to add additional stecker or intermediate reflector circuit to further increase the complexity of the encoding process.