# Enigma

Watt, Malcolm
260585950

April 15, 2016

## 1 Features

A sophisticated and easy to use encryption system. Allows easy encryption of messages which can be decoded at any time by an individual with access to your initial settings. There are millions of possible starting configurations, so unscrambling your messages will be practically impossible for unintended third parties.

The intuitive setup mode allows you to set a variety of options, from the type of reflector to use, to the initial position of the rotors of your machine. All of which impact encryption and make your messages more secure.

The user interface allows you to see which letter you are attempting to encrypt, and with the click of a button you can encrypt it and see the output and the input side by side. The converted output remains valid until you click the button again, meaning you do not have to hold the button down to read the encrypted letter, freeing your hands up to work more effectively.

Made a mistake and need to go back to your starting rotor positions? This is accomplished by the simple click of a button thanks to the well organized Enigma setup tools.

# Contents

## 2 System Description

This section is a technical description of the Enigma machine. We're going to take a top down approach to the system and its components, starting at the highest level and working our way down. Naturally let us then start with an explanation of the system's high level inputs and outputs. Figure 1 shows the block diagram of the overall system. This system toggles between operation mode and setup mode based on the *setup_switch*. When in operation mode, the input is determined based on the five slide switches that make up *input_code*, and the letter is encoded by hitting the button routed to *value_key*. A detailed description of how to operate the machine is discussed in section 3.3, while a detailed explanation of how to set up the Enigma is discussed in section 3.5.

Now we will go one step deeper and look at the contents of the Enigma user interface.

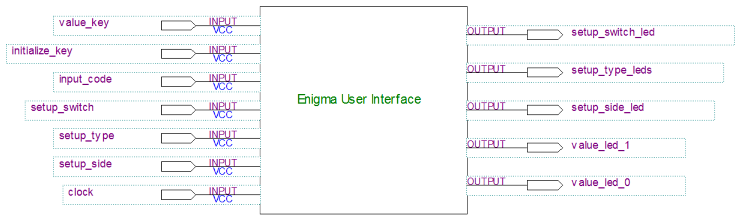


Figure 1: The top level block diagram of the Enigma user interface.

### 2.1 Interface Implementation

Once you have a chance to see how the different setup modes are toggled between in section 3.4, it will become clear that the different operation modes are being chosen based on a multiplexed system where the setup inputs choose between a set of configuration modes on the machine. Knowing this it will not be surprising to see that this is exactly how Quartus interprets the underlying VHDL. In figure 2 we can see the general layout of the user interface, however, there is a lot going on and the details are difficult to interpret. Generally you see the multiple layers of multiplexers that make up the selection of modes and settings, along with a set of D-flip-flops to retain the settings once they have been set. The specifics of this part are not particularly interesting, as they are simply the CAD tools rendering of a set of VHDL **case** statements.

In order to see closer let's examine figures 3 and 4. Here we can see the next layer of depth in both the way that we decode the outputs from 5 bit numbers into letter representations for the 7 segment LEDs, as well as the actual underlying Enigma machine. The Enigma machine's block diagram in figure 4 shows all of the inputs and outputs of the scrambling mechanism. The seven segment decoder's functionality is trivial, in that it converts a 5 bit number between 0 and 25 into its corresponding letter of the alphabet and asserts the corresponding signals for the seven segment display. There is no use discussing the internal workings of this any further and instead, in the following sections, we will discuss the implementation of the scrambling mechanisms within the Enigma.

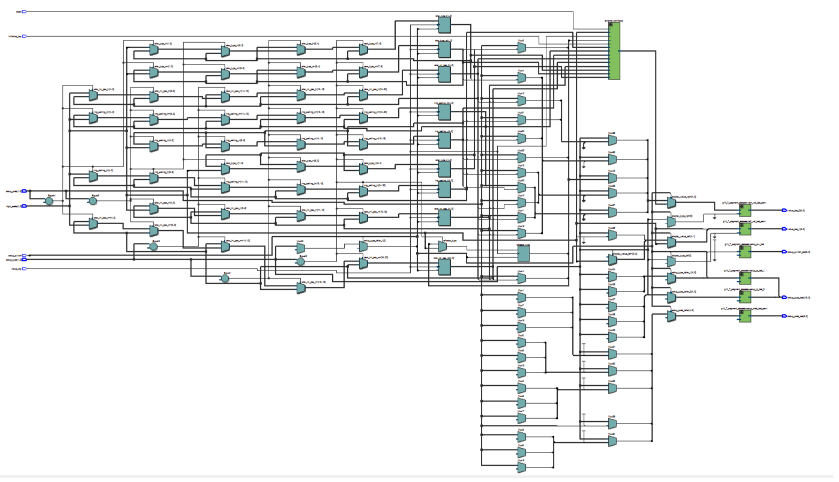


Figure 2: Implementation of the Enigma's user interface hardware.

## 2.2 Enigma Implementation

The actual Enigma machine is composed of several rotors as well as a finite state machine whose job it is to increment the appropriate rotors' rotor positions when key-press is asserted. There are other components within it, such as the stecker and the reflector, which we shall discuss shortly.

Let's first describe the general flow of two things, first the five bit wide *input_code* then the key-press bit. Each of these will be described in the following lists, with heavy references to figures 6 through 8. It is important that the reader sees how these diagrams fit into the overall system diagram of the Enigma in figure 5 before proceeding. In our description of how the input progresses through the circuit, we will ignore the internal architecture and subtleties of the rotors, and simply refer to the outputs as being scrambled versions of the input.

1. The input code is sent to the input of the *forward_stecker* which can be seen in figure 8. The stecker scrambles the input code and produces a temporary output_code which can also be seen in figure 8.

2. The output of the stecker is connected to the *input_code_forward* input of the right_rotor which can be seen in figure 8. Let's not concern ourselves

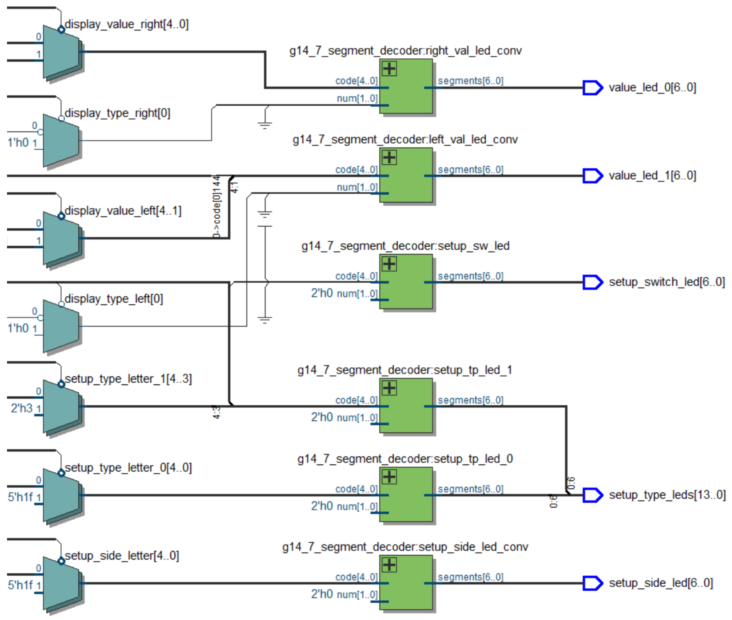

display_value_right[4..0]
g14_7_segment_decoder:right_val_led_conv
value_led_0[6..0]
display_type_right[0]
g14_7_segment_decoder:left_val_led_conv
value_led_1[6..0]
display_value_left[4..1]
g14_7_segment_decoder:setup_sw_led
setup_switch_led[6..0]
display_type_left[0]
setup_type_letter_1[4..3]
g14_7_segment_decoder:setup_tp_led_1
setup_type_letter_0[4..0]
g14_7_segment_decoder:setup_tp_led_0
setup_type_leds[13..0]
setup_side_letter[4..0]
g14_7_segment_decoder:setup_side_led_conv
setup_side_led[6..0]
code[4..0]
num[1..0]
segments[6..0]
2'h0
1'h0
2'h3
5'h1f

Figure 3: Block of seven segment display decoders.

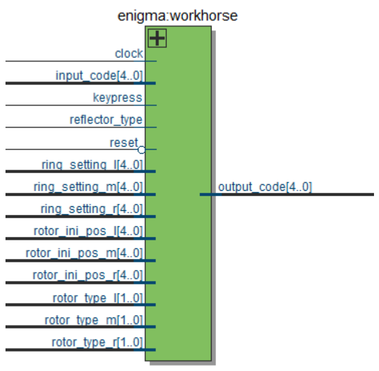


Figure 4: Block diagram of the actual Enigma scrambling mechanism.

just yet with how this rotor scrambles the input, but just know that it scrambles this forward input and the result is seen at *output_code_forward*.

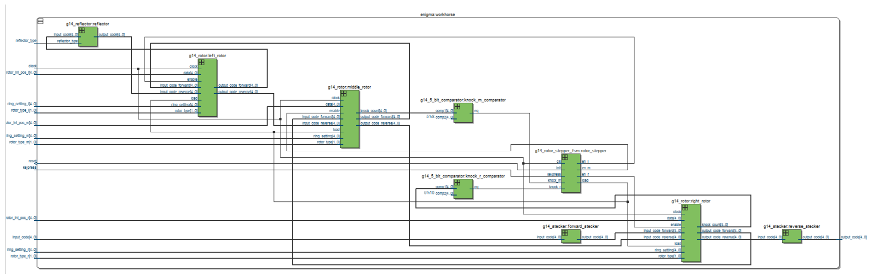


Figure 5: Complete internal block diagram of the Enigma scrambling mechanism. Refer to figures 6 to 8 for a closer look at the components.

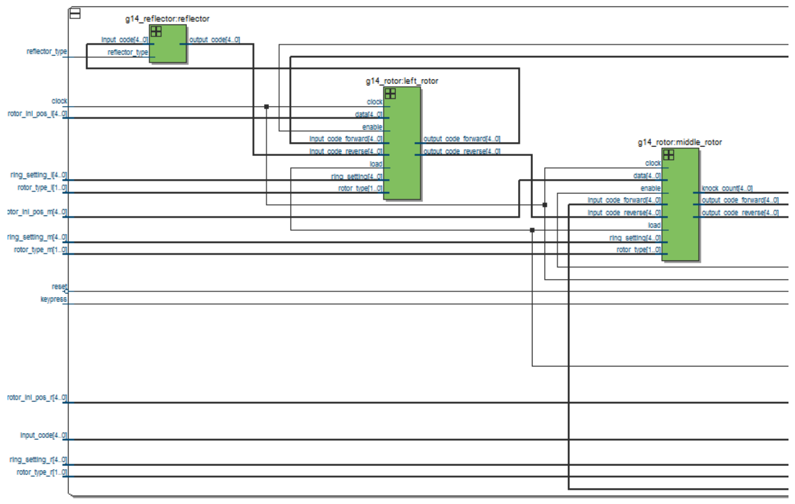


Figure 6: A closer look at the components in the left side of the block diagram in figure 5.

3. This wire is connected to *input_code_forward* of the middle_rotor, which can be seen in figure 7. Again, let's simply consider that this input is scrambled and the output is set at *output_code_forward*.

4. This output code is in turn connected to *input_code_forward* of the left_rotor, which can be seen in figure 6. Again, the scrambled output is set to the *output_code_forward*.

5. This signal is the input to the reflector, which can be seen in figure 6. The reflector consists of an invertible mapping of one letter to another, this means that if A as an input yields C as an output, then C as an input yields A as an output. The scrambled output is set at the output code.

6. From here we can follow the signal from the *input_code_reversed* to the

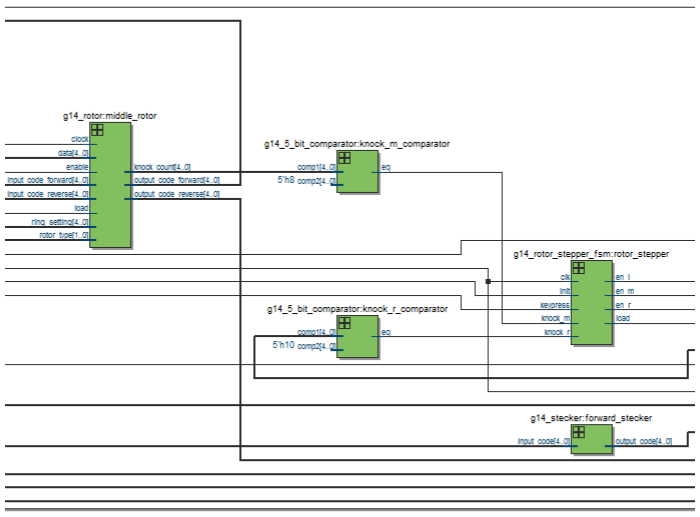


Figure 7: A closer look at the components making up the middle of the block diagram in figure 5.

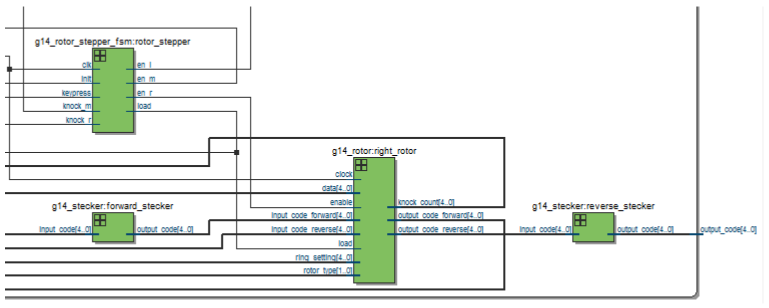


Figure 8: A closer look at the components making up the right side of the block diagram in figure 5.

*output_code_reversed* for the left_rotor, then the middle_rotor, and finaly the right_rotor.

7. As a final scrambling step, the *output_code_reversed* from the right_rotor goes into the reverse_stecker, which applies the inverse transformation of the original stecker on its input value, yielding the final result at its output. This reverse_stecker can be seen in figure 8.

The next thing to understand is how the key-press impacts the Enigma functionality. From a high level, it rotates the rotor's positions, based on the knock values passed to the finite state machine, this changes the scrambling in a predictable manner if you know the correct starting configuration of the machine, and in an uncrackable way if you do not. Before we can fully appreciate this, we need to understand the inner working of the rotors and the rotor stepper finite state machine. Let us elaborate the internal workings of these components so that we may truly grasp the impact of the key-press.

### 2.2.1 Stecker Circuit

The stecker circuit scrambles the input and the reverse stecker scrambles it back the opposite way. Basically the stecker is a one-to-one mapping of the 26 possible inputs onto the 26 possible outputs. The inverse mapping is specified such that in the regular direction if A maps to B then in the inverse mapping B maps to A. Note that this does not say anything about what B maps to in the forward direction, nor what A maps to in the inverse direction.

### 2.2.2 Reflector Circuit

The reflector circuit is similar to the stecker circuit in that it is a simple mapping. The difference is however that the mapping is invertible, that means an input of A yielding B means necessarily that an input of B would yield A.

### 2.2.3 Knock Comparators

These are simple five bit comparator circuits, which take two five bit wide inputs and assert the output if the two five bit words are identical.

### 2.2.4 Rotor Circuit

The rotor circuit has a forward and reverse direction. The actual circuit is fairly linear, especially when compared to the Enigma machine. Figures 9 through 13 show the traversal of the rotor circuit. Let's again follow the input code through the circuit, explaining the various transformations that happen along the way. We will only consider the *input_code_forward*, but the reverse path is pretty much exactly the same except for a difference in values on the shifting directions for some of the barrel shifters. The idea behind this difference in direction is that the input sequence is invertible: if you put a particular letter in in a particular rotor setting, then putting the result in as the input will yield the original input at the output. That being said, let's follow the *input_code_forward* through the circuit.

1. The input code goes into the forward_decoder_0 which decodes the 5 bit input (interpreted as an unsigned) into the appropriate one hot encoded 26 bit binary representation. Ex: If the input code is '00110' (6), then the sixth least significant bit of the output 26 bit vector will be asserted. This can be seen in figure 9.

2. The now decoded signal goes into the forward_barrel_shifter_0 where it is shifted to the right based on the current counter value. This can also be seen in figure 9.

3. The signal travels from the barrel shifter to another barrel shifter, which shifts the signal to the **left** based on the **ring settings** of the Enigma. This can be seen in figure 10.

4. The signal is now re-encoded back into its five bit representation by the forward_encoder_0 which can be seen in figure 10.

5. The signal is now sent to the forward_permuter, which based on the rotor type for this particular rotor, maps each possible 5 bit input to a 5 bit output (both in the range of [0,25]). This can be seen in figure 11.

6. From the permuter the scrambled value is decoded once more, at the forward_decoder_1 (as seen in figure 11).

7. Now the signal passes on to the forward_barrel_shifter_2 where it is shifted to the **right** based on the ring setting for this particular rotor (note that this is the opposite direction from before, and that the ring setting is shifting first. This is part of what makes the rotors invertible). This can be seen in figure 12.

8. The final scramble happens next, the signal is shifter **left** by the count (which we shifted the signal to the right by before). This can be seen in figures 12 and 13.

9. Finally, the 26 bit scrambled signal is encoded into a 5 bit vector and set at the output.

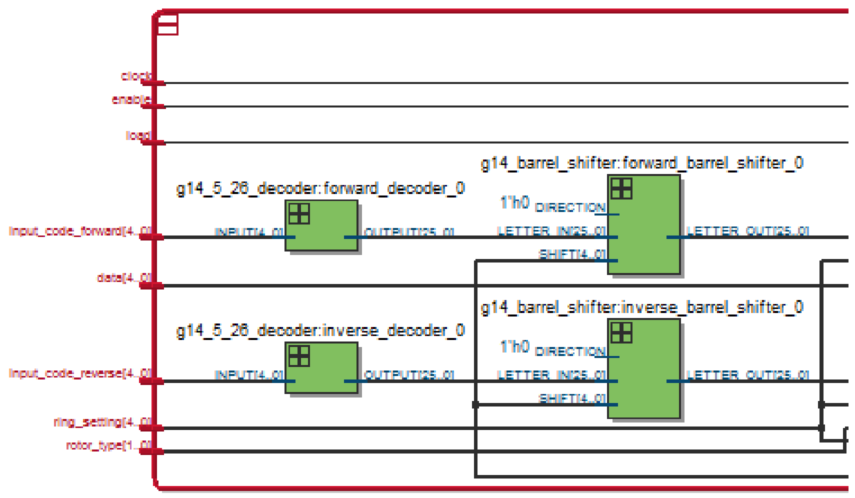


Figure 9: The input ports and first layer of components of the rotor.

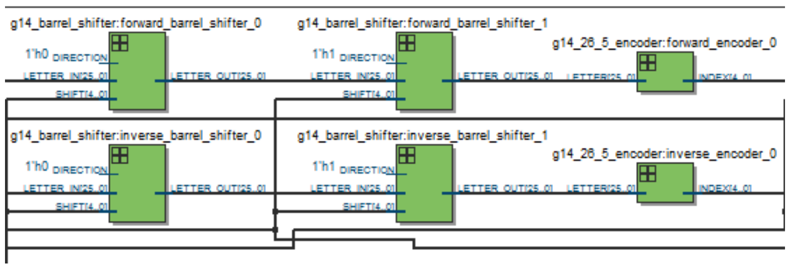

g14_barrel_shifter:forward_barrel_shifter_0
1'h0 DIRECTION
LETTER_IN[25..0]
SHIFT[4..0]
LETTER_OUT[25..0]
g14_barrel_shifter:forward_barrel_shifter_1
1'h1 DIRECTION
g14_26_5_encoder:forward_encoder_0
LETTER[25..0]
INDEX[4..0]
g14_barrel_shifter:inverse_barrel_shifter_0
g14_barrel_shifter:inverse_barrel_shifter_1
g14_26_5_encoder:inverse_encoder_0

Figure 10: The second layer of components of the rotor.

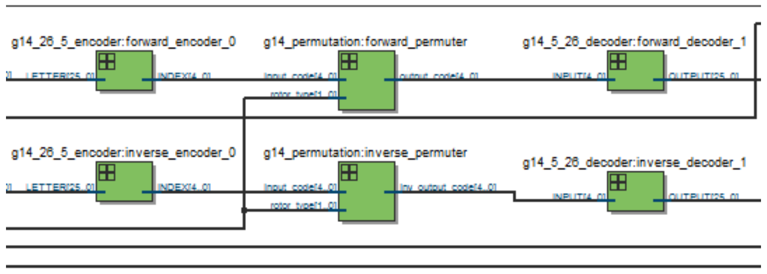

g14_26_5_encoder:forward_encoder_0
LETTER[25..0]
INDEX[4..0]
g14_permutation:forward_permuter
input_code[4..0]
rotor_type[1..0]
output_code[4..0]
g14_5_26_decoder:forward_decoder_1
INPUT[4..0]
OUTPUT[25..0]
g14_26_5_encoder:inverse_encoder_0
g14_permutation:inverse_permuter
inv_output_code[4..0]
g14_5_26_decoder:inverse_decoder_1

Figure 11: The third layer of components of the rotor.

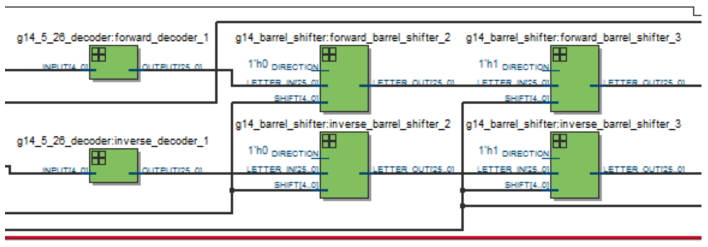

g14_5_26_decoder:forward_decoder_1
INPUT[4..0]
OUTPUT[25..0]
g14_barrel_shifter:forward_barrel_shifter_2
1'h0 DIRECTION
LETTER_IN[25..0]
SHIFT[4..0]
LETTER_OUT[25..0]
g14_barrel_shifter:forward_barrel_shifter_3
1'h1 DIRECTION
g14_5_26_decoder:inverse_decoder_1
g14_barrel_shifter:inverse_barrel_shifter_2
g14_barrel_shifter:inverse_barrel_shifter_3

Figure 12: The fourth layer of components of the rotor.

### 2.2.5 Rotor Stepper FSM

In the rotor circuit, we mentioned a count value which we shifted by in the first and last barrel shifter. These count values are incremented based on the Rotor Stepper FSM. The Finite State machine has 8 possible states. The block diagram of the Rotor Stepper FSM can be seen in figure 14. Overall, there are only two things you need to know about the rotor stepper FSM:

1. Pressing the init button puts the FSM in state 0 where it asserts the load signal which tells all of the rotors to reset their current rotor positions (i.e. the count value we saw before) to the one specified in the settings (i.e. held in the register in the user interface level circuitry).

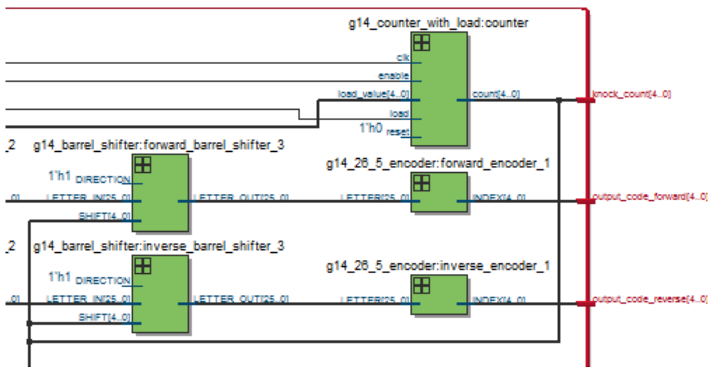

g14_counter_with_load:counter
clk
enable
load_value[4..0]
load
1'h0
reset
count[4..0]
knock_count[4..0]
g14_barrel_shifter:forward_barrel_shifter_3
1'h1
DIRECTION
LETTER_IN[25..0]
SHIFT[4..0]
LETTER_OUT[25..0]
g14_26_5_encoder:forward_encoder_1
LETTER[25..0]
INDEX[4..0]
output_code_forward[4..0]
g14_barrel_shifter:inverse_barrel_shifter_3
1'h1
DIRECTION
LETTER_IN[25..0]
SHIFT[4..0]
LETTER_OUT[25..0]
g14_26_5_encoder:inverse_encoder_1
LETTER[25..0]
INDEX[4..0]
output_code_reverse[4..0]

Figure 13: The final layer and output ports of the rotor.

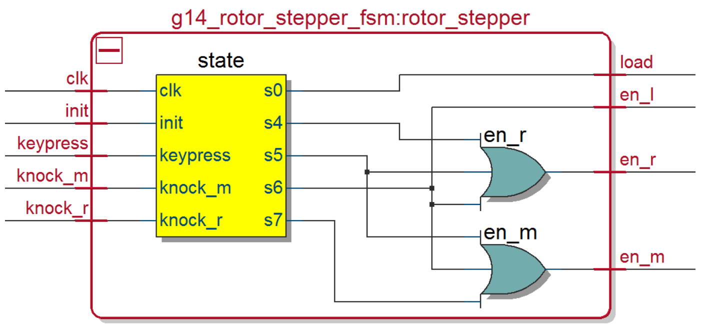


Figure 14: High level block diagram of the finite state machine.

2. Pressing and releasing the action button shifts the FSM through states 2 and 3, and depending on the current count and whether it is a notch in the rotor, it enables the appropriate counter increments.

In effect this is the driver of the count values that we use to shift within each of the rotors.

## 2.3 Final Remarks

We now understand the way the Enigma functions internally. Apart from the initial settings, the only things that is dynamic in the scrambling is the count values which are incremented based on the FSM and triggered by the KeyPress signal. The preceding sections have been fairly involved, describing the implementation behind the functionality. For a better understanding of how to actually use the Enigma, refer to the following section.

# 3 Interface Guide

## 3.1 Labels & Shorthand

Before we get started, it is useful to describe some shorthand notation for displays and switches on the board. You can see the slides switches and their corresponding annotations in figure 15. The seven segment LED displays are labeled, from left to right, HEX5 down to HEX0, which you can see on the board. The push buttons are labeled from left to right KEY3 down to KEY0, which can also be seen on the board. From here on out I will refer to everything by its shorthand name.

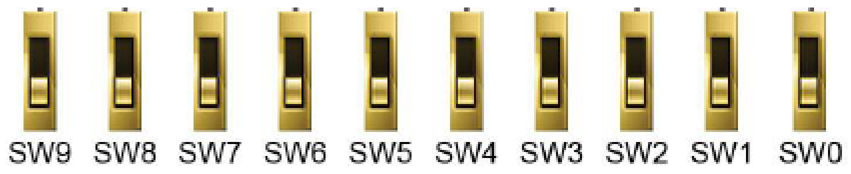


Figure 15: Display of the slide switches and their appropriate labels [1].

## 3.2 Buttons

There are two buttons which impact the functionality of the board. KEY3 is the **action button**. It is the button you hit to set values, and will be referred to as the *action button* from now on. KEY0 is the **reset button**. While encoding or decoding values using the Enigma machine, you can reset the rotors to the daily settings by hitting this button. More details on this will become clear in the following sections. This button will be referred to as the *reset button* from now on.

The two buttons were placed far apart to avoid accidentally hitting the wrong button. Accidentally depressing the reset button would require the user to restart the message encoding / decoding procedure entirely, which is not something that we want to have happen.

## 3.3 Operation Mode

Let's briefly ignore the various setup tools made available to us and use the default settings to get a good idea of how the operation of the Enigma works. In order to get into operation mode, all you need to do is make sure that **SW9** (or the *setup_switch*) is down. You will know you are in operation mode if **HEX5** (the left most display) is displaying the letter O.

### 3.3.1 Operation Displays

In operation mode **HEX1** indicates the current letter that you have selected by your input switches (refer to table 1 for indication of what these switches are).

**HEX0** indicates the current translation of your input value by the Enigma machine. **The general procedure for encoding a message is the following:**

- Push and release the reset button.
- Set the input switches to the value you want to input, which you can verify in **HEX1**.
- Push and release the action button. You will notice that HEX0 will change when you press the action button. Note that it does not necessarily need to change though.
- The encoded letter for your input is the letter in HEX0. Write down HEX0.
- Repeat steps 2 through 5 until you have completed your message.

**Likewise the decoding procedure is as following:**

- Push and release the reset button.
- Set the input switches to the value you want to decode, which you can verify in **HEX1**.
- Push and release the action button. You will notice that HEX0 will change when you press the action button. Once again, it doesn't necessarily need to change.
- The decoded letter for your input is the letter in HEX0. Write down HEX0.
- Repeat steps 2 through 5 until you have completed decoding your message.

Note that the above procedure assumes that the Enigma you are encoding and decoding with are set to the same *initial settings*. We will discuss initial settings later in this guide.

This is all you need to know as far as the general operation of the Enigma is concerned. However, the most important part of the Enigma machine's encryption is being able to change the initial setting of the machine. The setup of the machine is discussed in the following sections.

## 3.4 Switches

Before we can dive into the different setup configurations of the Enigma, it is important to understand the switches and their general purpose. There are nine switches on the FPGA, all of which are used to some extent for the user interface of the Enigma. They are broken up into four distinct groups as you can see in table 1. Each of the groups' options and particularities are explained in the appropriate section.

| Group | Switches | Functionality |
|---|---|---|
| Setup | SW9 | Toggles between setup and operation mode |
| Setup Type | SW8 & SW7 | Allows us to select what we are setting up (4 options) |
| Setup Side | SW6 & SW5 | Allows us to select which side we are setting up |
| Input Value | SW4-SW0 | Allows us to input values between 0 and 31 |

Table 1: Description of the switches on the board and their functionality.

## 3.5 Configuration

There are four different settings of the Enigma, each of which have multiple possible options. The following section explains each of these settings and their possible values, as well as how to set them up. First thing is first, to get into setup mode, put the setup switch high (SW9). Tables 2 and 3 show how the slide switches impact which setup mode we are in, and the following sections describe how to use this to effectively set up the Enigma machine.

| Setup Type | SW8 | SW7 | Displays |
|---|---|---|---|
| Reflector | 0 | 0 | RF |
| Rotor Type | 0 | 1 | RT |
| Rotor Initial Positions | 1 | 0 | IP |
| Ring Setting | 1 | 1 | RS |

Table 2: Setup types and their switch configurations.

| Setup Side | SW6 | SW5 | Displays |
|---|---|---|---|
| Right | 0 | 0 | R |
| Middle | 0 | 1 | M |
| Left | 1 | D | L |

Table 3: Setup sides and their switch configurations.

### 3.5.1 Reflector

In the chain of scrambling of the Enigma, the reflector scrambles the forward going path and send it back on its way down the reverse path. From the users perspective there are two possible values that the reflector can be set to (i.e. two mappings from the forward output to the reverse input). In order to set the reflector, you need to have both of the *setup type* switches down (SW8 and SW7). You will know you are setting up the correct setting if HEX3 and HEX2 read ‘RF’. In this state, you can see the current value of the reflector (0 or 1) in HEX0. You can use SW0 to choose a value for the reflector (which you can verify in HEX1) and you can set the reflector value by pushing the action button.

### 3.5.2 Rotor Type

Each rotor (there are three) has a mapping from one letter to another (or in some cases itself). These mappings are what scramble the input code. There are four different possible mappings for each rotor of this particular Enigma machine. In order to set a particular rotor's type, we must get into the 'rotor type' setup mode, and we must select the appropriate rotor (right, middle, left) at which point we can set the particular value.

Refer to table 2 in order to see where to set switches 7 and 8 to get into *Rotor Type* mode. From there you can pick which side you want to set using switches 5 and 6 (consult table 3 to see how to pick a particular side). Once you have picked a side, HEX0 will show the current rotor type of that side. You can adjust the value sliders (SW4-SW0) to pick a rotor type value between 1 and 4. The current value being interpreted based on your inputs is displayed in HEX1. Set the value to what you want and press the action button to set the rotor type to the value you have selected.

### 3.5.3 Rotor Initial Positions

If each rotor started at the same point every time we encrypted a message, the encryption wouldn't be very difficult to crack at all. Luckily for us, we are able to start each of the three rotors at any of 26 initial positions, we can even change them on the fly using agreed upon techniques if both the person encoding the message and the person decoding the message know how to follow along.

Again refer to table 2 in order to see where to set switches 7 and 8 to get into *Rotor Initial Positions* mode. From there you can once again pick which side you want to set using switches 5 and 6. HEX0 will show the current starting letter (initial position) of the particular rotor and HEX1 will display the currently selected value of the input slide switches (SW4-SW0). Once you have picked the value that you would like, you may press the action button and you will set the rotor initial position.

### 3.5.4 Ring Setting

On top of the initial positions of the rotors, we have an initial scrambling step called the ring setting. There is a particular ring setting for each of the three rotors and each ring setting can be set independently.

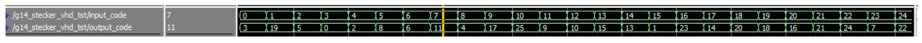
Once again, refer to table 2 in order to see where to set switches 7 and 8 to get into *Ring Setting* mode. From there you can once again pick which side you want to set using switches 5 and 6. HEX0 will show the current ring setting (a letter between A - Z symbolizing the shift) of the particular rotor and HEX1 will display the currently selected value of the input slide switches (SW4-SW0). Once you have picked the value that you would like, you may press the action button and you will set the ring setting for the particular rotor.

# 4 Testing

The system was tested in simulation for each of its components. Most of these components have been shown in previous lab reports, so we will focus on two new components as well as the overall system testing.

## 4.1 Stecker

The stecker circuit scrambles the inputs of the Enigma machine. It provides a one to one mapping from the 26 possible inputs to the 26 possible outputs. You may see the results of all 26 inputs being applied to the stecker circuit in figure 16.

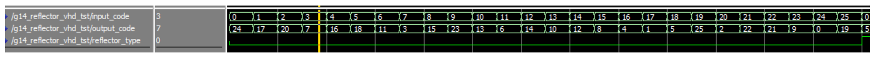


```
/g14_stecker_vhd_tst/input_code    7    0  1  2 3 4 5 6 7  8 9  10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
/g14_stecker_vhd_tst/output_code   11   3  19 5 0 2 8 6 11 4 17 25 9  10 15 13 1  23 14 20 18 16 21 24 7  22
```

Figure 16: Simulation for all possible 26 inputs to the stecker with outputs displayed.

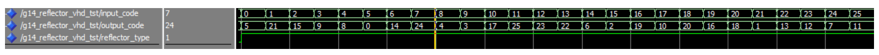
## 4.2 Reflector

The reflector provides an invertible scrambling step to the input. This means that if an input A yields B then an input of B must yield A. You may see the simulation results of these simulations in figures 17 and 18.



```
/g14_reflector_vhd_tst/input_code      3   0  1  2  3 4  5  6  7 8  9  10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 0
/g14_reflector_vhd_tst/output_code     7   24 17 20 7 16 18 11 3 15 23 13 6  14 10 12 8  4  1  5  25 2  22 21 9  0  19 5
/g14_reflector_vhd_tst/reflector_type  0
```

Figure 17: The reflector circuit being tested in its **B** setup.



```
/g14_reflector_vhd_tst/input_code      7    0 1  2  3 4 5 6  7  8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
/g14_reflector_vhd_tst/output_code     24   5 21 15 9 8 0 14 24 4 3 17 25 23 22 6  2  19 10 20 16 18 1  13 12 7  11
/g14_reflector_vhd_tst/reflector_type  1
```

Figure 18: The reflector circuit being tested in its **C** setup.

## 4.3 Enigma Simulation

The overall simulation results of the Enigma machine are shown in two separate cases in figures 19 and 20. As you can see figure 19 we input a sequence of values who upon keypress yield 0 at the output. Then we reset the Enigma, and input 0, at which point the original input sequence arises at the output. We do the same thing for the setup in figure 20, the difference is that we have a different internal setup for the Enigma machine.

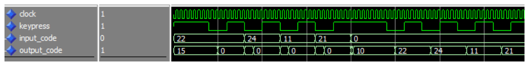


Figure 19: The Enigma simulation with setup **a**.

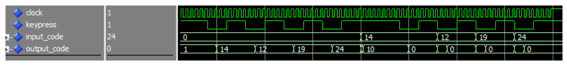


Figure 20: The Enigma simulation with setup **b**.

### 4.4 Full System Testing

The system was tested by using multiple setups and attempting to encode and decode messages using the Enigma.

## 5 Resources & Timing

The system uses a total of 60 registers. These memory elements are comprised of the memory elements required to hold the current setting in the user interface, the current count in the counter mechanisms, and the state machine's state. The design uses 55 (12%) of the pins available on the board, which is not really of much concern as it is not constrained in any way. It uses 2,456 of 32,070 available ALMs in terms of actual combinational logic (which comes out to about 8% of the available logic units on the board). This comes down to the fairly complex user interface logic. The user interface could have been made much more simple in hardware, but it would have been at the loss of ease of use from the end user's perspective. Since the point of the user interface was to be easy to use, and not necessarily the most hardware efficient, this is acceptable. The other combinational logic is happening everywhere inside the rotors appart from the counters.

The slow model's $F_{max}$ was found to be 88.28 MHz based on the hardware implementation. There were no constraints placed on the clock of the board in this analysis, so a discussion of slack and hold times does not add to our understanding of the timing of the circuit. However, from the slow model's $F_{max}$ we can see that it will in fact meet the timing constraints of the board. The frequency of the clock on the board is 50MHz, and since the $F_{max}$ of our circuit is significantly higher than this, we do not need to worry about errors in the sequential logic.

## 6 Conclusion

Now that we have discussed how the system works and how it was tested, let us briefly discuss some of the difficuties and potential enhancements for the Enigma.

## 6.1 Issues

The only major issue which blocked progress and impacted development was the fact that the 5 to 26 decoder was in fact faulty. Originally we wrote the decoder behaviorally with a simple for loop, however we were required to re-implement the decoder using a structural approach (select statements). In doing so we introduced a bug (obviously, since the structural code is unclear and close to 30 lines longer, with each line containing at least 4 or 5 more operations than any given line in the loop). This decoder did not impact anything until the final machine was assembled, at which point it took some serious debugging to find the issue. In the end, we re-wrote the decoder behaviorally in order to avoid the complexity of the structural approach.

## 6.2 Enhancements

There is a lot of room for enhancement on the Enigma. The easiest way for me to elaborate is to simply provide a list of potential enhancements:

- The ability to rotate the rotors back by one if you make a mistake, this prevents you from needing to reset and restart the encoding of the entire message if you make a mistake.
- The ability to see how many characters you have so far encrypted, this allows you to know the current length of your word, which is a way of checking if you are where you expect to be.
- Expanding on the previous idea, the ability to simply see the entire input message and the entire output message side by side would be extremely helpful. Along with this we would require a better display and a more elaborate value-in mechanism.
- Include a few syntax characters in the remaining configurations of the 5 bit word. Notably we could include a space, a comma, a period, and potentially others.

## References

[1] Terasic Technologies and Altera Inc., 9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City, 30070. Taiwan. *Terasic DE1 SoC User Manual*, 2003.