6.2050 Enigma and Bombe Machine Final Report

Kevin Min
Department of EECS
Massachusetts Institute of Technology
Cambridge, MA
km1317@mit.edu

Anna Simmons
Department of EECS
Massachusetts Institute of Technology
Cambridge, MA
annasim@mit.edu

Taylor Wagner
Department of EECS
Massachusetts Institute of Technology
Cambridge, MA
twagner5@mit.edu

***Abstract*—The Enigma Machine is an encryption machine that was used by Germany during World War II to safely send coded messages. In response to this, Alan Turing designed the Bombe machine to decode the encrypted messages and break the Enigma Machine, ending the war years earlier. In this project we attempt to model both the Enigma Machine and the Bombe machine, using 3 FPGAs. One of these FPGAs sends encoded messages as the Enigma machine would have done, while another will be recieves and decodes these messages using the same process, with both FPGAs knowing the encoding process. This transmission will take place using infrared signals. The third FPGA simulates the Bombe machine and attempts to intercept these signals and break the encryption, without knowing the specific encoding settings that the two communicating FPGAs are using.**

***Index Terms*—FPGA, Enigma, Bombe, encryption, IR**

I. Background

This project explores cryptography, specifically the Enigma machine encoding process and the Bombe machine decryption process, using three FPGAs. The FPGAs will communicate using an infra-red transmitter and receiver, and will be hooked up to HDMI displays to display the messages. The current input method is using the on-board buttons and switches. The main benefit of using an FPGA is to allow for quick encoding/decoding, particularly with the large amount of computation needed for the decoding process in the Bombe machine.

II. Sending the Message

The first FPGA is the message sender and is responsible for encoding the message using a known rotor configuration. After encoding each letter they transmit the message letter by letter over an infrared signal to be picked up intentionally by the message receiver and then decoded using the enigma module settings.

A. Data Module (Kevin)

This module takes in the inputs from the switches and buttons, and processes the information to send to the enigma modules when all the information is available. Depending on the value of $sw[15:14]$, the remaining switches will get stored as either the rotor choices, rotors initial rotations, or the letter to send. We considered also implementing functionality using an external PS/2 keyboard for easier input, but ultimately we ran out of time to add this.

B. Enigma Module (Taylor)

This module constitutes the main body of the project. It is designed to simulate the original Enigma machine, using a series of three rotors and a reflector to take in letters one at a time. Each unencoded input letter passes through the three rotors in order, then the reflector, and passes back through the rotors in the opposite direction to yield the output, where each rotor forms a bijection between the 26 letters. The rotors can be selected from a pre-chosen set of five, and set to any of the 26 possible initial rotations, for a total of $5 \cdot 4 \cdot 3 \cdot 26^3 = 1054560$ initial settings. As with the actual Enigma machine, the rotors will turn after every letter to make the encryption harder to break, with the first rotor turning after every letter, the second one turning when the first one completes one full rotation, i.e. every 26 letters, and the third one turning when the second one completes one full rotation. This ensures that the encoding effectively changes with every letter passed in, so common decryption methods like frequency analysis fail. The reflector ensures that the encoding and decoding processes are the same, provided that the initial rotor settings are the same, so this module can be used for both the transmitting FPGA and the receiving one, provided both agree on the initial rotor settings.

This module takes input wires for rotor selection, rotor initial rotation, data valid in, reset in, data in, and sends back a ready signal alongside the encoded letter and a data valid signal, where rotor selection corresponds to the $5 \cdot 4 \cdot 3$ choices of which rotors we use and in what order, and rotor initial rotation corresponds to the 26^3 total rotations at the start. When the module sends a ready signal, it can take in 1 letter, which it will take 8 cycles to output the encoded letter. During these 8 cycles, the module is putting the letter through the 3 rotor mappings, the reflector, and backwards mappings of the 3 rotors. It also will perform the necessary rotor shifts during this period.

We originally wanted to also add the plugboards to our module, which were an additional feature in most military-grade Enigma machines that swapped around up to 10 pairs of letters at the very start of the encryption process, yielding an extra factor of $\frac{26!}{2^{10} \cdot 6! \cdot 10!} \approx 1.5 \cdot 10^14$ possibilities, making a brute force code-breaking method nearly impossible, which would necessitate a smarter approach that used the fact that

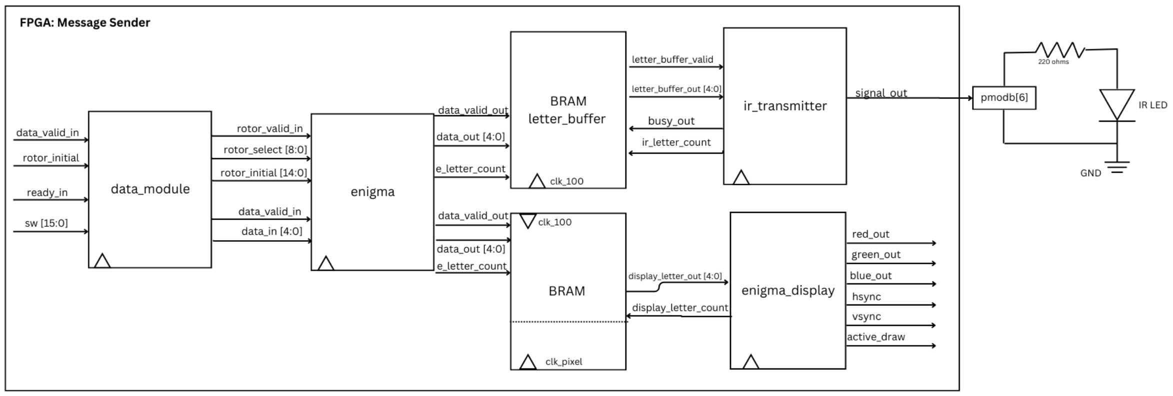

Fig. 1. System Diagram for Message Sender

letters cannot get encoded to themselves due to the way the reflector worked. However, we ultimately ran out of time for this and settled for the simpler version that is much easaier to brute force.

The configuration of the enigma rotors can be seen in Figure 5. This displays all 5 rotors as well as the reflector. Note in the implemented enigma module only 3 of these rotors are chosen and their order can also be reconfigured.

Our implementation of the Enigma is modeled after the Norenigma that was used by the Norwegian Police Security Service after WWII.

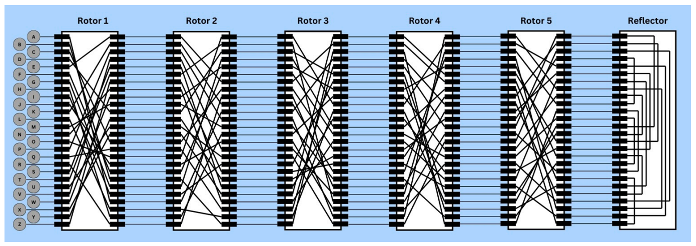

Fig. 2. Rotor Configuration for all 5 rotors plus the reflector

C. IR Transmission (Anna)

Messages are sent over infrared signals. One by one, letters are encoded by the enigma module and these letters are stored in a 5×1024 BRAM. When it is ready to transmit a signal, the $ir\_transmitter$ module takes in 1 letter at a time, 5 bits, and transmits the corresponding signal to that 5 bits over an IR LED based on specific bursts of lights and silence. The specific signal encoding starts with a synchronization period of 2.4 milliseconds of light followed by a sequence of 5 bits where a '0' is encoded by 600 microseconds of silence and 600 microseconds of light and a '1' is encoded by 600 microseconds of silence and 1200 microseconds of light. The end of the message has an extra 600 microseconds of silence period. See Figure 3 for an example of this signal for the letter 'J'. Because the IR receiver is tuned to infra-red lights of $40kHz$ frequency, the 'on' periods of the IR signal are bursts of a 40kHz frequency at 50% duty cycle. We use Pulse Width Modulation to achieve this frequency. Using IR signals lets us communicate across different FPGA boards to send this message. When not transmitting a message, this signal stays 'off' i.e. silent.

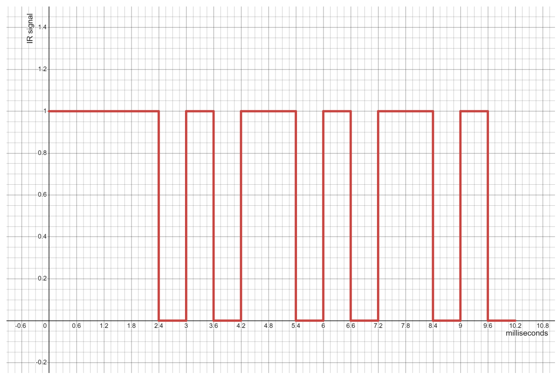

Fig. 3. Infra-red Signal for the letter J: '01010'

D. Display (Kevin)

This display is designed to look similar to the actual enigma machine, with a "keyboard" that darkens the letter being pressed, and another set of "lights" that shows the encoded letter being sent. This display is designed for a 1024×720p screen and runs on a 74.25 MHz pixel clock.

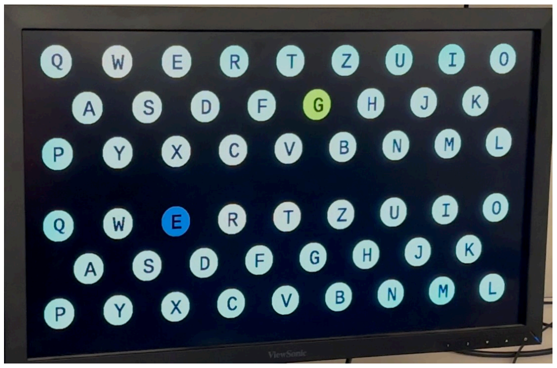

Fig. 4. Message Sender Display

III. Receiving the Message

A. IR Decoding (Anna)

We decode the infra-red signal using an IR sensor that detects IR light at $40kHz$ frequency. This receiver interprets the light as either 'on' or 'off' where, for our purposes, 'on' means the IR light is oscillating at 40kHz and 'off' means it is silent. This IR receiver decodes the message through the $ir\_decoder$ module. It decodes 5 bits at a time, one letter, by sensing the IR light transmission from the $ir\_transmitter$ signal output over the LED. This module is expecting the specific periods of 'on' and 'off' for the light for the synchronization period and for interpreting '1's and '0's as outlined in the IR Transmission section. Once it successfully receives a full 5 bit message it outputs that message with a data_valid signal. This output is the encoded letter and will then be run back through the enigma module to get the original letter. If the decoder interprets any IR signal that doesn't follow the expected protocol, it will return to an 'idle' state and wait until a signal with the expected protocol is received.

B. Display (Kevin)

This module uses a 5×1024 BRAM to store messages of up to 1024 letters at a time, with the letters being taken from the decoding Enigma module one at a time. It then outputs said message on the screen using a pre-generated sprite sheet of letters, displaying $32 \times 16 = 512$ letters on a 1024×720p screen, and a 74.25 MHz pixel clock. It also has an input wire that allows "scrolling" one line at a time, in order to display all 1024 letters. This same display will also be used for the Bombe machine after breaking the message, and there will potentially be a module that can write the decoded messages to a text file using UART and Python for the final project, for displaying large messages in a more convenient format.

IV. Breaking the Message

A. Bombe Module (Kevin)

The Bombe module is our stretch focus for this project. Unfortunately we did not entirely finish it, but we got the module working in simulation, and it was not flashing properly onto the FPGA for unknown reasons. It utilizes the FPGA's ability to process lots of data in parallel, by simulating 8 simultaneous Enigma machines. The original plan was to have 32 simultaneous Enigmas, which was what the original Bombe actually had, but the FPGA did not have enough resources for this. At 9 cycles for each letter, messages of up to 1024 letters, and $5 \cdot 4 \cdot 3 \cdot 26^3$ possible settings for the Enigmas, and 8 simultaneous Enigmas this is a maximum of roughly 120 million cycles to decode a message, or just over a second.

The module takes in the same inputs as the enigma module, except for the rotor signals, which will be replaced with a known phrase. The known phrase is a phrase that is likely to be found in the message; with the actual Bombe machine used in WWII, this would be a phrase that the British would manually guess, using external sources. For example the Germans would often send a weather report in the morning, so the code breakers would use the phrase "weather report" in their Bombe machine. This word is then used to test the possible scramblings of the Enigma machine, and to quickly rule out the overwhelming majority of possibilities.

There will also be an output code valid signal, which is high when the Bombe machine finds a possible combination. One of the issues with the Bombe is while it can usually reduce the possible decodings to a single digit number, it is still up to human decoders to finish the process. This signal is here to tell the user when the Bombe has found a possible encoding, so they can manually check it before the Bombe module continues the search.

V. Design Evaluation

A. Timing of Modules

- **Enigma**: This module is clocked on the onboard 100MHz clock. The module can take in an input every 8 cycles, meaning it has a latency of 80ns and a throughput of 1/80ns.
- **IR**: For the IR signals, sending one letter, a 5 bit message, takes between 9 and 12 milliseconds, which means it has a latency of 9-12 milliseconds. The throughput is between $\frac{1}{9}$ and $\frac{1}{12}$ letters per millisecond. The variability in this timing is because a '1' and a '0' are encoded with different lengths of time.
- **Display**: The display operates on a 74.25Mhz clock, drawing 60 frames per second. The transmitting Enigma display has a throughput of 15 cycles, and the receiving display has a throughput of 12 cycles.

Currently, the slowest part of our system comes from inputting the letters, because we use the onboard switches. In the future this can be improved by integrating an PS2 keyboard to type the messages instead.

B. BRAMs

In total we use two BRAMs on the FPGA that sends the message, one BRAM on the FPGA that receives the message, and one BRAM for the FPGA that breaks the message with the Bombe. All of the BRAMs store letters and have a width of 5 bits and depth of 1024.

For transmitting the message, both BRAMs are written to with encoded letters from the $enigma$ module. The first

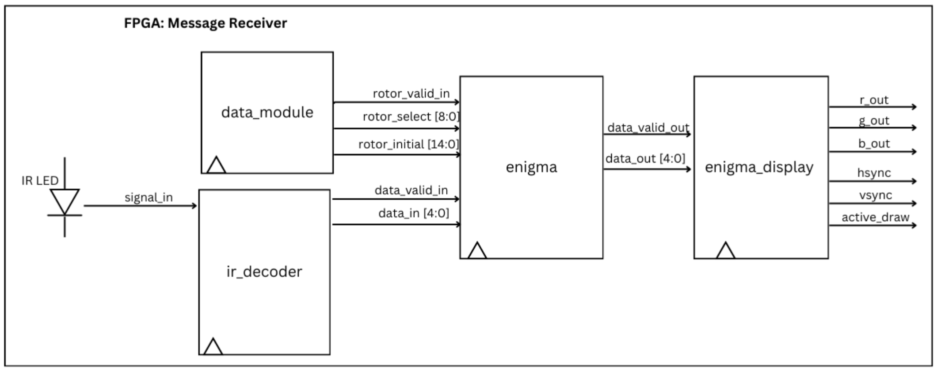

Fig. 5. System Diagram for Message Receiver

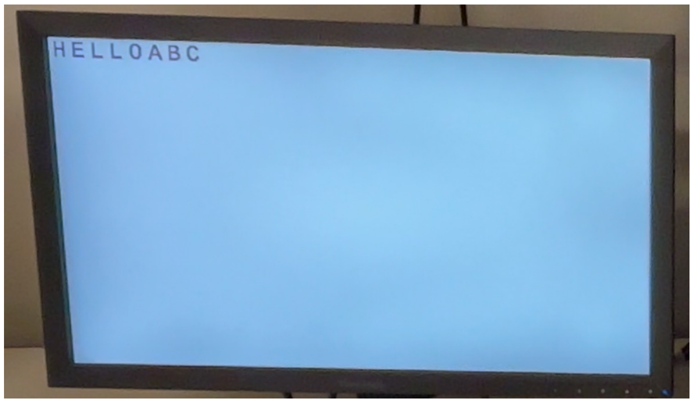

Fig. 6. Message Receiver Display

BRAM operates on a 100MHz clock and is read by the $ir\_transmitter$ module which then transmits these letters over an IR signal. The second BRAM is written to on the 100MHz clock and read on the 74.25MHz clk_pixel clock. The $enigma\_display$ module reads the letters from this BRAM and displays one letter at a time as it is encoded. We chose to have two BRAMs because we have modules operating on different clock domains and this let us meet timing requirements. Ideally we would have liked to have consolidated this into one BRAM because they are both storing the same information, encoded letters, but this required some careful setup to meet the timing constraints, as we would have needed to read and write from the same port, and we decided this was not a priority and did not achieve this.

When receiving the message we use one BRAM which writes decoded letters on a 100MHz clock from the $enigma$ module that is decoding letters. This BRAM is inside the $text\_display$ module which reads from it on the 74.25MHz clock, displaying these letters on the screen as they are received.

The BRAM for the Bombe operates on the 100MHz clock and stores the incoming encoded message from the IR signal. Once the entire message is stored, the Bombe module uses that message to start the decoding process.

C. Timing Requirements

From analyzing the Vivado logs, we succesfully met our timing constraints on sending and receiving the message. For transmitting the message our worst negative slack (WNS) was 1.922 nanoseconds and our total negative slack (TNS) was 0. For receiving the message, our WNS was .331 and TNS was 0.

D. Use Case

- **Commitment**: We successfully met our commitment goal of a functioning system that sends and receives encoded messages using the enigma module. We used IR signals to send these messages and onboard switches for the inputs.
- **Ideal**: We have two functioning displays for sending and receiving the message. We did not incorporate a PS/2 keyboard.
- **Stretch**: We have a Bombe module working in simulation but were not able to integrate it to work on the board successfully.

VI. Insights

Integrating our modules together to create the full transmitting and receiving system took longer than expected. Specifically, since we were working with different clock domains, we struggled with meeting timing and had negative slack at first. After updating the BRAMs to make sure the addresses and input/outputs were updated on the correct clocks we were able to meet the timing constraints and ensure the system worked as expected.

Using onboard switches takes a long time to send messages which also meant that it took a while to check if our system

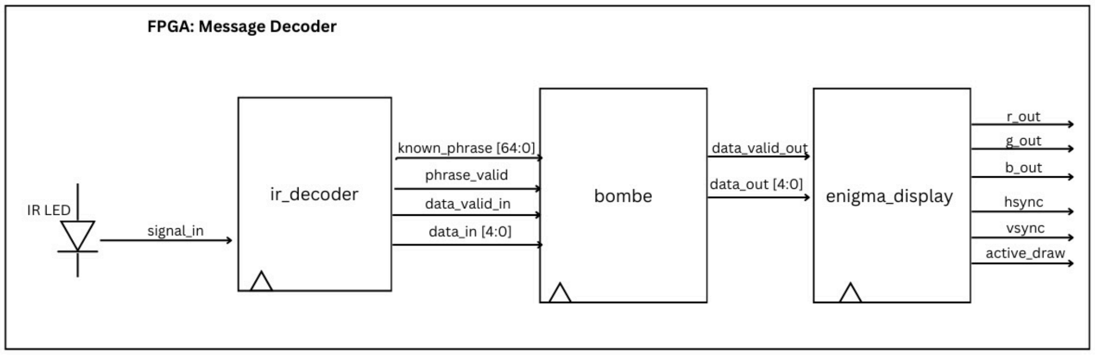

Fig. 7. System Diagram for Message Breaker

was working when we were testing it. If we had focused on incorporating the PS/2 keyboard, inputting messages would have gone much faster and likely would have been easier to test.

VII. References

- [redacted]
- Enigma Wiring