EE344:ELECTRONIC DEVICES LAB

Annie D'souza 20D070028

Raavi Gupta 200070064

Sakshi Heda 200070071

April 28, 2023

Final Report

DEPARTMENT OF ELECTRICAL ENGINEERING
IIT BOMBAY

RFID Based Detector

Table of contents:

Introduction

The aim of this project is to develop an RFID (Radio Frequency Identification) system. RFID is a form of wireless communication that incorporates the use of electromagnetic or electrostatic coupling in the radio frequency portion of the electromagnetic spectrum to uniquely identify an object or a tag. An RFID system basically consists of a reader (or interrogator) and tags. The RFID reader uses radio waves to transmit signals that activate the tag. Once activated, the tag sends a wave back to the antenna, where it is translated into data.

In this project, we will be developing an RFID reader module using the relevant IC, antenna and circuitry. The tags will be ready made.

Working of an RFID system

Radio frequency identification (RFID) is employed where data encoded in “tags” are attained by a reader by the use of radio frequencies. It is similar to using bar codes that are seen in many everyday applications such as inventory tracking with the advantage that the tag doesn’t have to be in direct line of sight of the reader. RFID scanner recognizes locations and identification of tagged items. Moreover, more than one item can be identified (as long as it’s in the range of the reader) automatically without the need of human intervention which was not the case in a barcode scanner.

RFID systems mainly consists of three components:

- RFID reader
    - Induces enough power into tag
    - Provides syncronization clock to tag
    - Acts as a carrier for return data from tag
- RFID tags
- Antenna

RFID Reader:

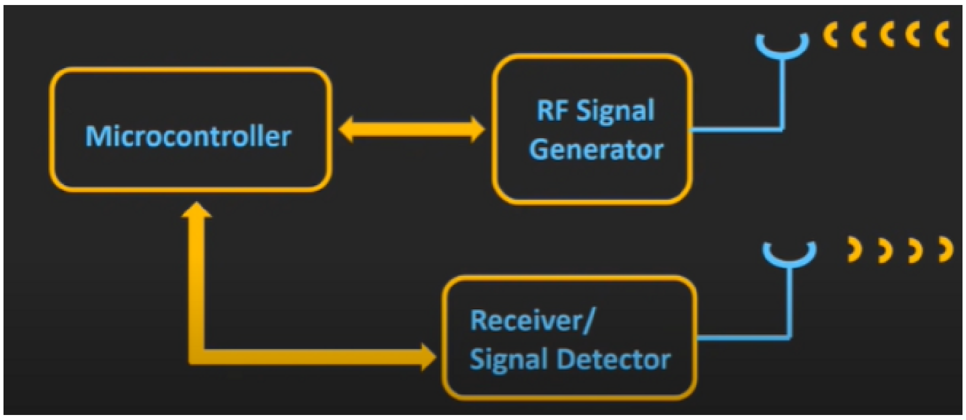


Figure 1: RFID Reader

RFID Tag:

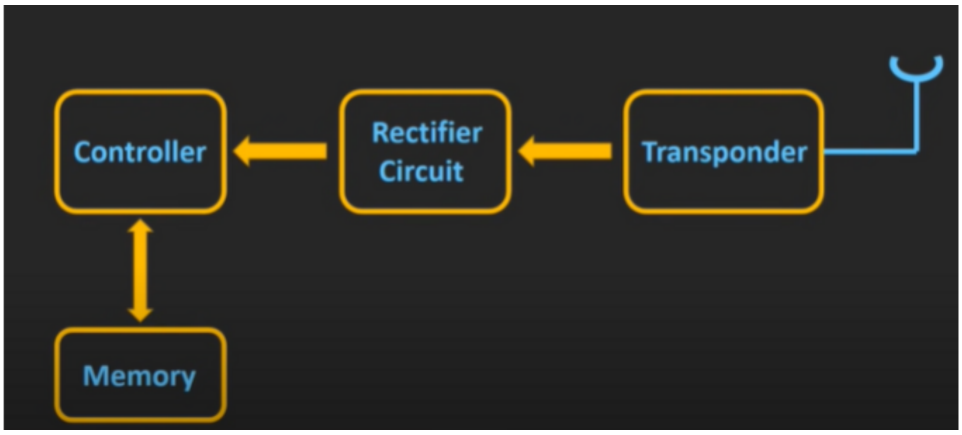


Figure 2: RFID Tag

Problem Statement

To design an RFID reader that works on 125kHz frequency (Low Frequency).

When an RFID tag is brought within close proximity to the reader, the reader emits a electromagnetic field that energizes the tag's antenna. The tag then responds by modulating the field and sending back its UID to the reader.

The reader must have an antenna that is tuned to the correct frequency to properly read the tag's UID. The reader also requires a microcontroller to decode the UID and make decisions based on that information. In this case, the reader is designed to determine whether the UID is valid or invalid, and then open or keep the lock closed accordingly.

Approaches

Approach 1

Interfacing the antenna with the microcontroller via a reader IC (HTRC110)

The schematic can be divided into 5 major components. These components are as follows:

1. HTRC110 Reader IC and its connections
2. Impedance matching circuit
3. STM32 μcontroller and its connections
4. The antenna
5. Buzzers and LEDs

Schematic:

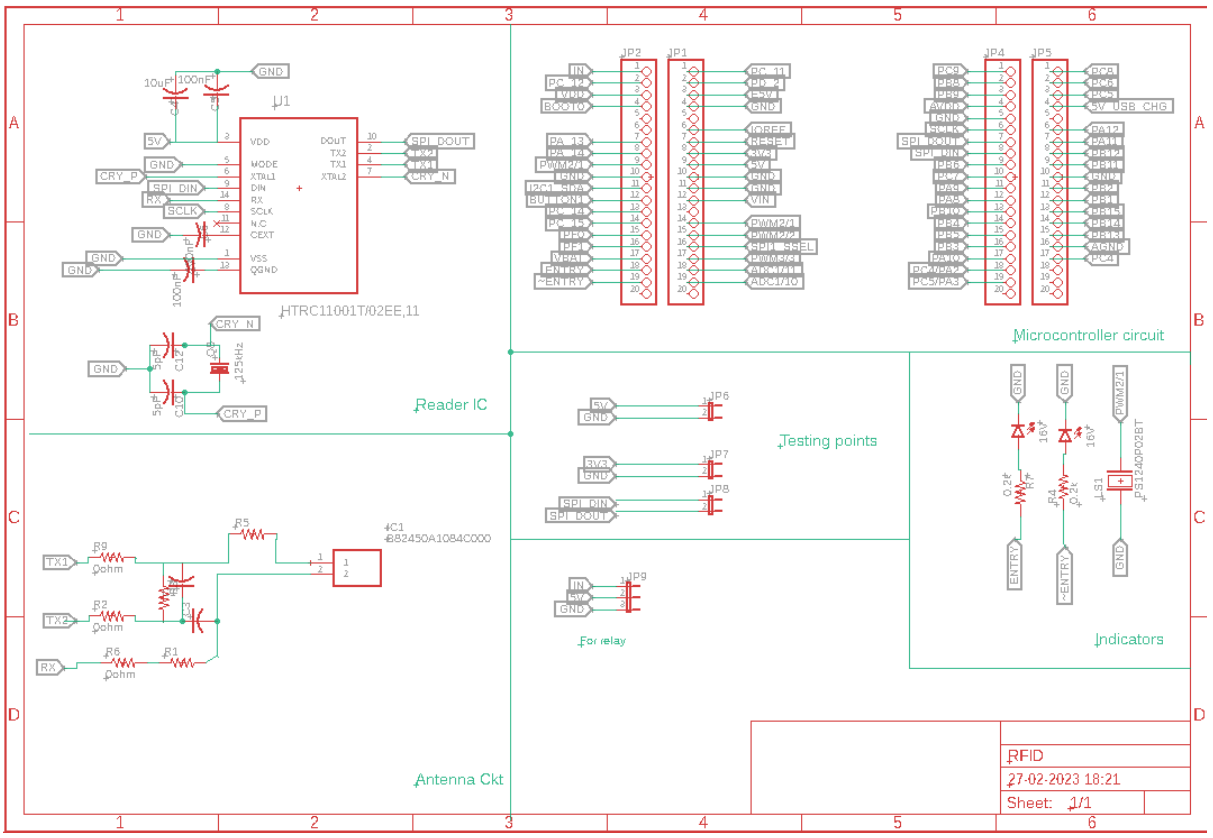


Figure 3: RFID circuit schematic

1. **Reader IC circuit**:
    a. All the connections of the reader IC were done in accordance to the specifications mentioned in the datasheet.
    b. The value of the capacitances connected to the oscillator has been also acquired from the datasheet
    c. The communication between the Reader IC and the microcontroller is (or was initially thought to be) via SPI (Serial Peripheral Interphase)

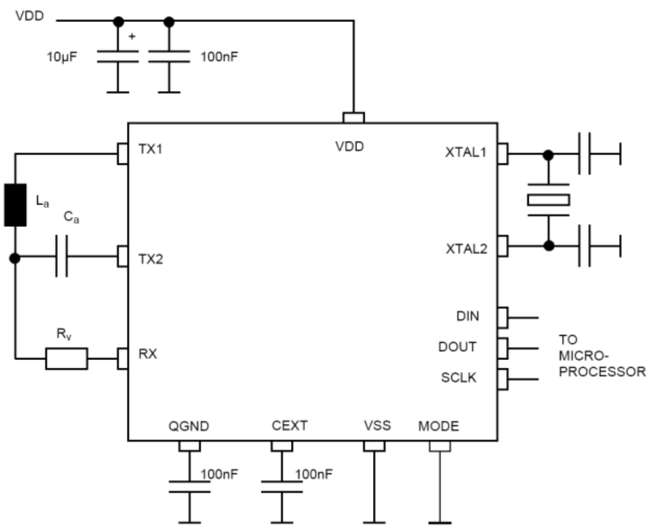


Figure 4: HTRC connections as given in the datasheet

2. **Antenna circuitry**

The three 0 ohm have been placed for debugging purposes, so that the antenna circuit can be isolated from the reader IC circuit while testing.

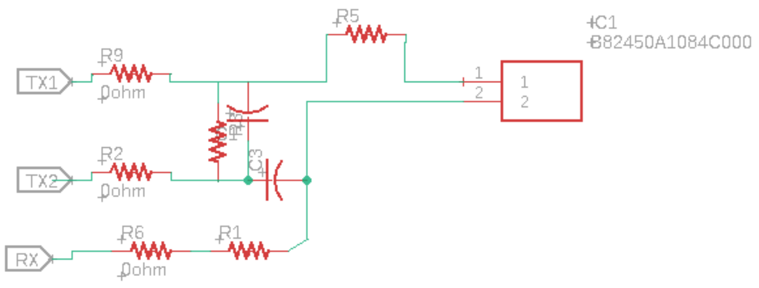
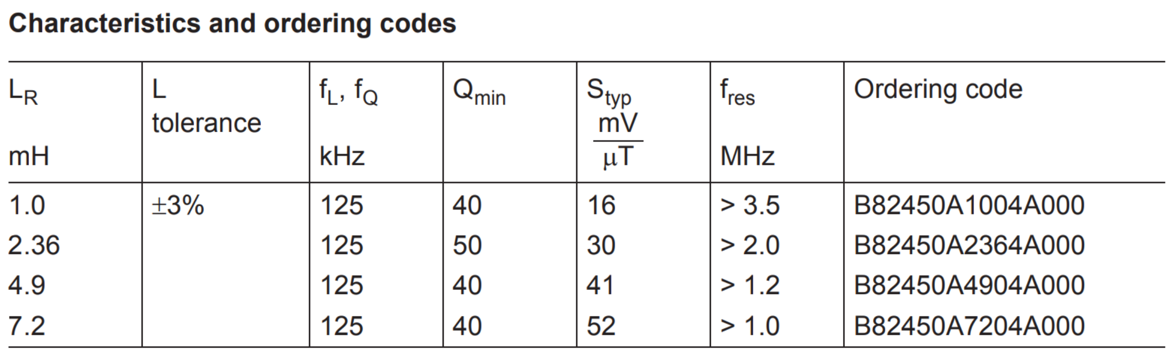


Figure 5: Antenna matching



Characteristics and ordering codes

L_R mH	L tolerance	f_L, f_Q kHz	Q_{min}	S_{typ} $\frac{mV}{\mu T}$	f_{res} MHz	Ordering code
1.0	±3%	125	40	16	> 3.5	B82450A1004A000
2.36		125	50	30	> 2.0	B82450A2364A000
4.9		125	40	41	> 1.2	B82450A4904A000
7.2		125	40	52	> 1.0	B82450A7204A000

Figure 6: Characteristics given in antenna datasheet

To measure the R and L values of our antenna we used the network analyzer at the lab which gave the following results at $125kHz$.

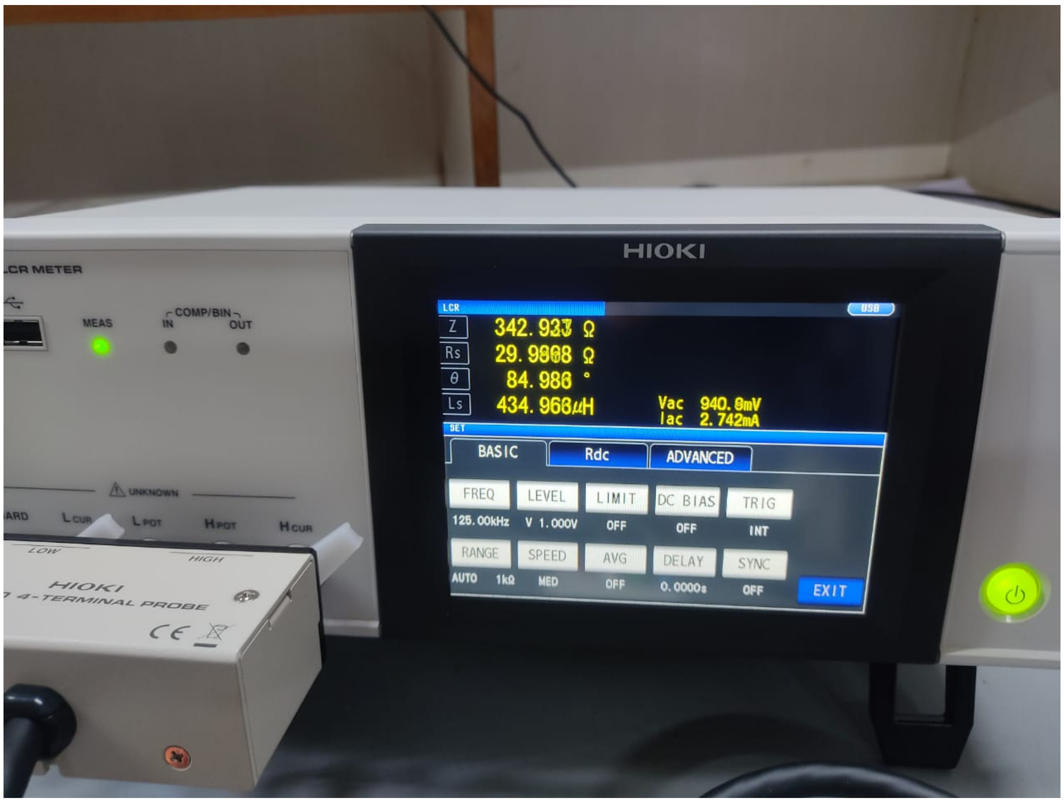


Figure 7: LCR meter reading

$Z = 342.9\Omega$, $\theta = 84.98^\circ$. From this we get $R = 23.86\Omega$, $X_L = 341.16\Omega$. Since we are doing this at operating frequency $125kHz$, we get $L = 434.38\mu H$ which is also verified by the analyzer reading.

Next, from the HTRC110 (Reader IC) datasheet, we have $I_{ant_{max}} = \frac{\hat{U}_{dr}}{R_{ant}} = \frac{4}{\pi} \cdot \frac{V_{DD}}{R_{ant}}$

$V_{DD} = 5V$ and $I_{ant_{max}} = 200mA$

$\therefore$ we get $R_{ant_{IC}} = 31.83\Omega$

This is the required impedance of antenna as seen by the reader IC. For maximum power transmission, we need to match the antenna impedance to the impedance as seen by the reader IC, i.e., 31.83Ω

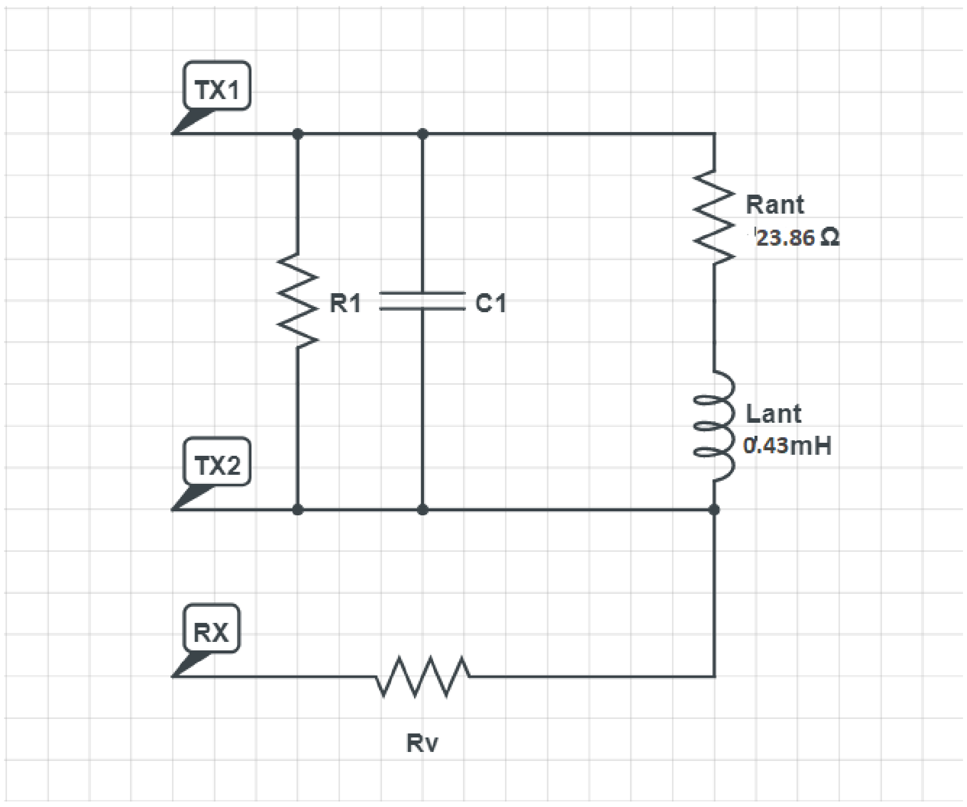


Figure 8: Impedance matching circuit

$(R_{ant} + jX_L)||C_1||R_1 = R_{ant_{IC}}$

Solving this with the above values, we get $C_1 = 3.71pF$, $R_1 = 32.00\Omega$

The receiver terminal of the reader IC is internally connected to ground via resistance, R_{demin}. Since the tap of the antenna will be at very high voltage, an external resistance (named R_v) will be required. The RX terminal of the IC should be maintained at voltage ranging from $-8V$ to $+8V$

Taking the extreme case, we have the equation $\frac{U_{tap_{max}}}{R_v+R_{demin}} = \frac{U_{RX_{max}}}{R_{demin}}$

On solving we get $R_v = R_{demin}(\frac{U_{tap_{max}}}{U_{RX_{max}}} - 1)$

$U_{tap_{max}} = 2\pi f_o L_{ant} I_{ant_{max}} = 68.23V$

Also taking into account experimental and design errors, we take $U_{RX_{max}} = 7.5V$

Then, $R_v = 202.44k\Omega$ where $R_{demin} = 25k\Omega$ as given in the datasheet

Thus, we got the values for the original antenna

3. **Microcontroller:**

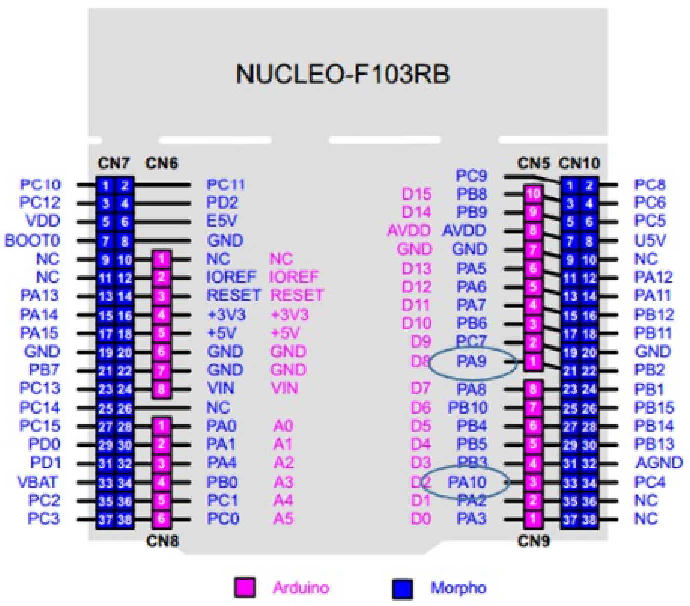


Figure 9: Pinout of STM32F1

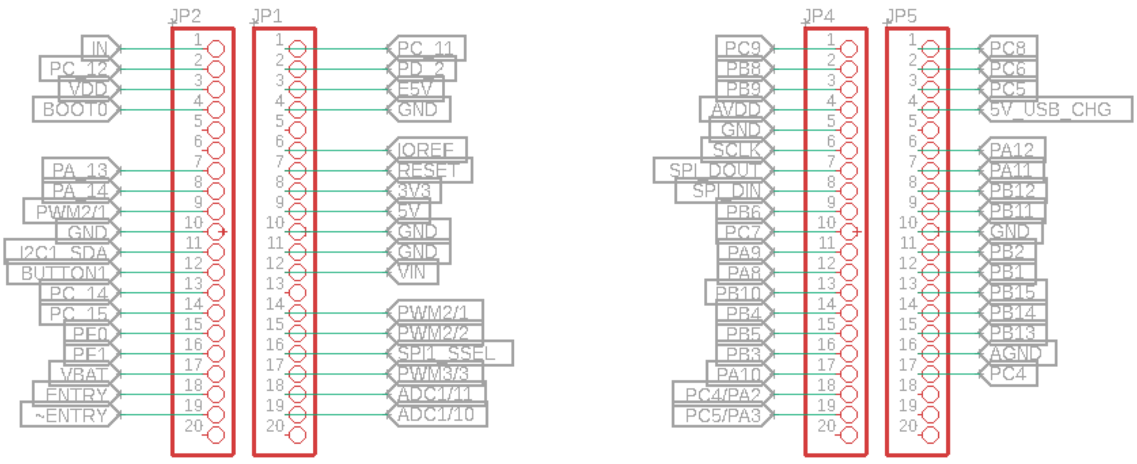


Figure 10: Microcontroller

a. Since we are being provided a nucleo development board, the four twenty-jumper headers are kept so that the microcontroller fits snugly on top of the board.

b. The last pin of all the headers are empty as the nucleo rails consist of 19 pins whereas we have 20 pin jumpers

c. The pink rails shown in the left figure are present for Arduino compatibility and have been scrapped for our purposes

4. **Relay and lock circuit:**

a. The electromechanical relay has to be externally connected to the lock which in turn operates based whether the card (tag) has valid entry or not

5. **Testing points and Indicators:**

   a. The testing points are kept for easy debugging and to ensure that proper signals are coming into the microcontroller when a tag is tapped

   b. There are two LEDs and one buzzer present on the board to provide visual and auditory stimuli when a tag comes into the range of the reader

The individual layers are as follows:

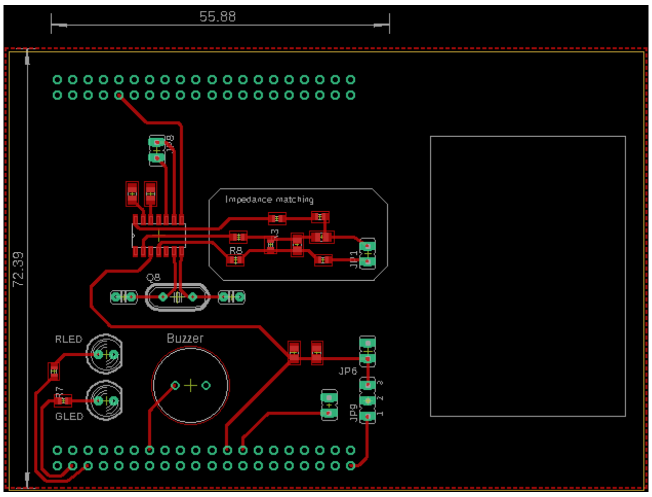


Figure 11: Routed board top layer

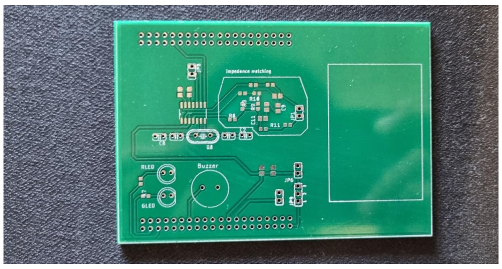


Figure 12: PCB top layer

On the left is the routed top layer of the board and above the printed PCB can be seen

The rectangle represents the space for antenna to be placed on the PCB

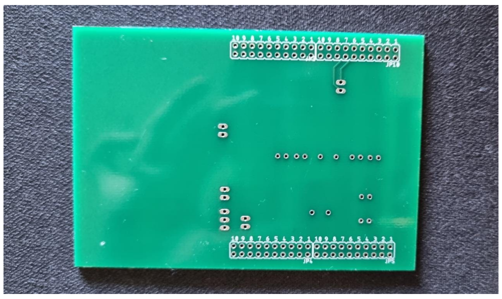


Figure 14: PCB bottom layer

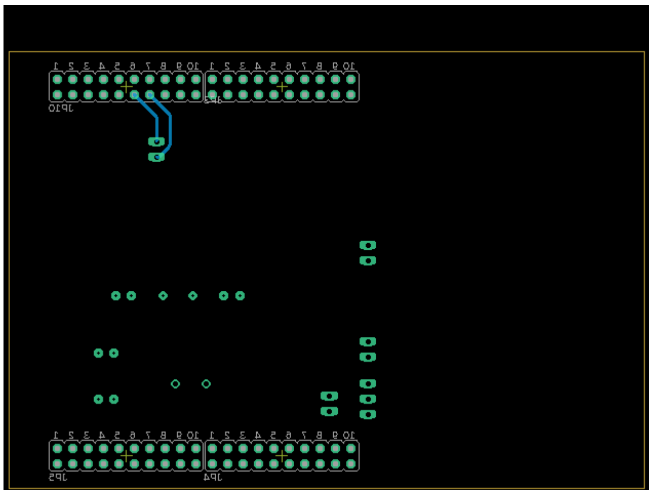


Figure 13: Routed board bottom layer

On the left is the routed bottom layer of the board and above the printed PCB can be seen

Code

The code testing was the most tedious and time consuming part of the milestone. The HTRC was assumed to communicate with the microcontroller through SPI.

The configuration used for this project has been described as follows:

- For SPI clock pin PB3 has been used.
- For MISO and MOSI, pins PB4 and PB5 have been used.
- The GPIO output pins are pins PC2 and PC3 - these pins are for controlling the relay module and LEDs.

The code can be further divided into the following subparts:

void rfid_read_tag (void):

This function is called to start the process of reading tag. In this, the *rfid_read_tag_pending* flag is set to 1.

BYTE rfid_get_read_tag_result (void):

This function is used to get the current status of the program whether the card has been successfully read, read is in process or read has failed. It is used to check for the end of the read tag process.

This function returns 0 if still reading, 1 if completed. However, bit 7 will be set if the read failed.

After a successful read the ASCII tag ID will be in rfid_data_nibbles[0] | rfid_data_nibbles[9] (which is a variable of the required data type).

void rfid_process (void):

This functions comprises of different stages in the process of reading the tag. This sequence of these stages is mentioned in the datasheet of our reader IC HTRC110.

The stages have been mentioned as follows:

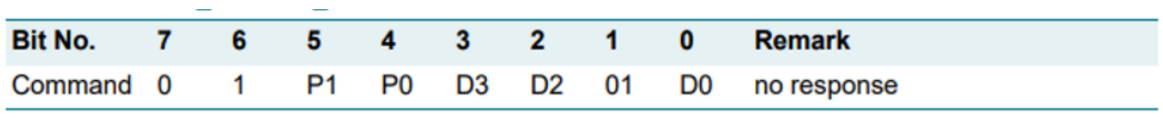
- **RFID_INITIALISE**

Here, we pause for >= 10mS for HTRC110 to be ready. (specified in section 11.1 of the Philips "Read/Write Devices based on the HITAG Read/Write IC HTRC110" application note AN97070 rev 1.1). Therefore we added a delay of 11ms in this state before moving on to the next state

We also set *rfid_state* to the RFID_SET_IDLE state.

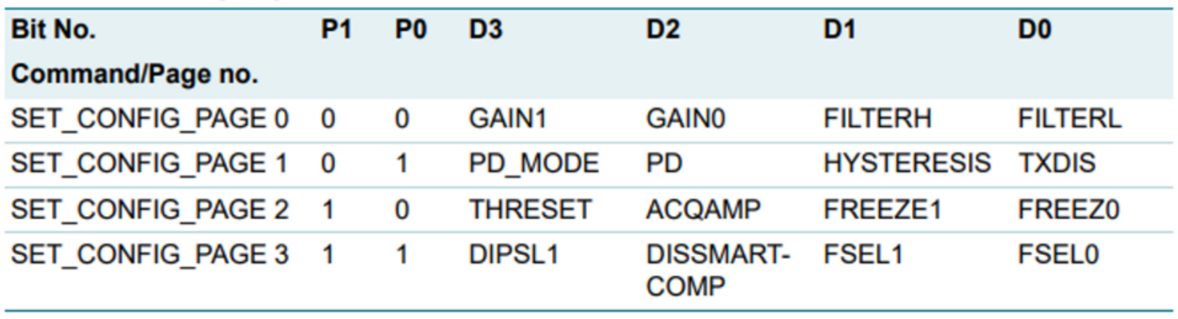
- **RFID_SET_IDLE**

Next we need to set amplifier and filter parameters for the different operating modes. We have four operating modes which are represented by two bits P1 and P0.



Bit No.	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Remark**
Command	0	1	P1	P0	D3	D2	01	D0	no response

Figure 15: SET_CONFIG_PAGE command



Bit No. **Command/Page no.**	**P1**	**P0**	**D3**	**D2**	**D1**	**D0**
SET_CONFIG_PAGE 0	0	0	GAIN1	GAIN0	FILTERH	FILTERL
SET_CONFIG_PAGE 1	0	1	PD_MODE	PD	HYSTERESIS	TXDIS
SET_CONFIG_PAGE 2	1	0	THRESET	ACQAMP	FREEZE1	FREEZ0
SET_CONFIG_PAGE 3	1	1	DIPSL1	DISSMART-COMP	FSEL1	FSEL0

Figure 16: Config Page

1. For CONFIG_PAGE 3, DISLP1=0, DISSMARTCOMP=0, FSEL=11(for 16MHz clock) - 0x73

2. For CONFIG_PAGE 0, we have a default value of gain as 2 (will be represented as 10), FILTERH=1, FILTERL=1 - 0x43
3. For CONFIG_PAGE 2, THRESET=1, ACQAMP=1, FREEZE=11 - 0x6f
4. For CONFIG_PAGE 1, PD_MODE=0(idle mode), PD=1(device power down), HYSTERESIS=0(off), TXDIS=1(coil driver inactive) - 0x55

Each command is of 8 bits which is transmitted serially from the microcontroller to HTRC in the order mentioned above (from the datasheet). This transmission is done using the function rfid_tx_rx_byte

After doing this HTRC enters the IDLE state

- **RFID_IDLE:**

Here the reader is in inactive low power state and waits for the read tag request (from *rfid_read_tag_pending* flag). If this flag is 1, the reader enters into the reading state

- **RFID_READ_TAG**

For read mode of the RFID reader, we use the following commands:

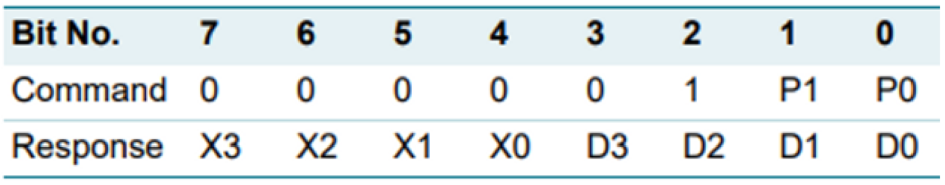
1. SET_CONFIG_PAGE 3 = 0x73 (same as idle state)
2. SET_CONFIG_PAGE 0 = 0x43 (same as idle state)
3. SET_CONFIG_PAGE 1 = 0x50 (PD=0, TXDIS=0)

Then following sequence should be carried out for CONFIG_PAGE 2 (HITAG Pg46)

1. SET_CONFIG_PAGE 2 = 0x6B (THRESET=1, ACQAMP=0, FREEZE=11)
2. SET_CONFIG_PAGE 2 = 0x68 (FREEZE = 00)
3. SET_CONFIG_PAGE 2 = 0x60 (THRESET = 0)

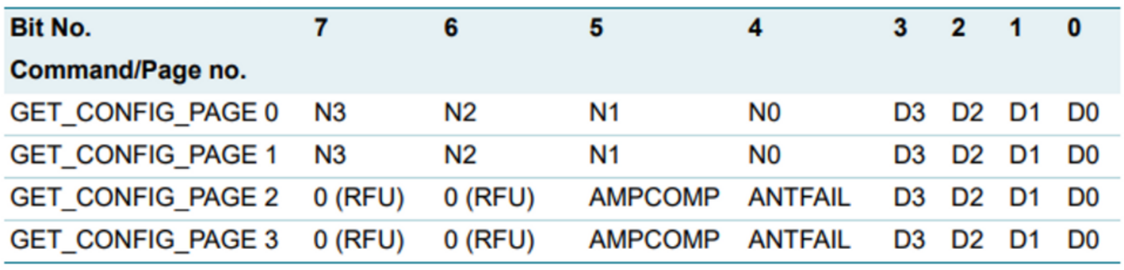
There is a delay of 4ms and 1ms between these commands respectively

Next we will check whether the antenna is connected properly(not short or open). For this we will use GET_CONFIG_PAGE command



Bit No.	7	6	5	4	3	2	1	0
Command	0	0	0	0	0	1	P1	P0
Response	X3	X2	X1	X0	D3	D2	D1	D0

Figure 17: GET_CONFIG_PAGE command



Bit No. Command/Page no.	7	6	5	4	3	2	1	0
GET_CONFIG_PAGE 0	N3	N2	N1	N0	D3	D2	D1	D0
GET_CONFIG_PAGE 1	N3	N2	N1	N0	D3	D2	D1	D0
GET_CONFIG_PAGE 2	0 (RFU)	0 (RFU)	AMPCOMP	ANTFAIL	D3	D2	D1	D0
GET_CONFIG_PAGE 3	0 (RFU)	0 (RFU)	AMPCOMP	ANTFAIL	D3	D2	D1	D0

Figure 18. Config Page

We will send GET_CONFIG_PAGE 2 command (0x06) and then observe the 4th bit of the received byte. If ANTFAIL == 1, we will declare RFID_READER_FAIL and exit the state, otherwise proceed further.

Next step is setting the sampling time of the reader.

Following are the steps involved:

1. Read Phase: For this we transmit 0x08 over the SPI and observe the received byte.
2. Left shift the received time by 1 bit (that is multiplication by 2)
3. Add the offset compensation value to it
4. Mask the 2 MSBs to get modulo 64

We will set this as the sampling time of the reader.

Now we are ready to start reading the tag and move to the state RFID_READ_TAG_READING

This tag reading process will take place in parallel to this current RFID_process. So here we will enable the MISO rising edge interrupt which will start reading the tag.

- **RFID_READ_TAG_READING**

  We will check the state as from the interrupt function. If it gives success, we call the function rfid_end_read_tag() to declare the end of process, sets the rfid_read_tag_success flag to 1 and then convert the received value to ASCII and then again enter the RFID_SET_IDLE state

  Otherwise if the reading fail, still it ends the process declaring failure and gets back to the RFID_SET_IDLE state.

- **RFID_READ_FAIL**

  This state directs the code back to the RFID_SET_IDLE state.

void rfid_sampling_rising_edge (WORD us_since_last_rising_edge)

This is the function called in the interrupt service routine to read the tag.

Here, *us_since_last_rising_edge* is the time in microseconds that passes after the last card reading till the next time (present) when it is called.

Following are the different states for the state *rfid_capture_state*:

```
typedef enum _SM_RFID_CAPTURE
{
  RFID_CAPTURE_INITIALISE,
  RFID_CAPTURE_GET_BIT_RATE,
  RFID_CAPTURE_GET_HEADER,
  RFID_CAPTURE_GET_DATA,
  RFID_CAPTURE_SUCCESS,
  RFID_CAPTURE_FAILED,
  RFID_CAPTURE_FAILED_GIVEN_UP
} SM_RFID_CAPTURE;
```

We use the enum datatype for these states which assigns an integer value to these states which makes it easier to compare them.

BYTE rfid_sampling_rising_edge_add_bit (BYTE bit_state)

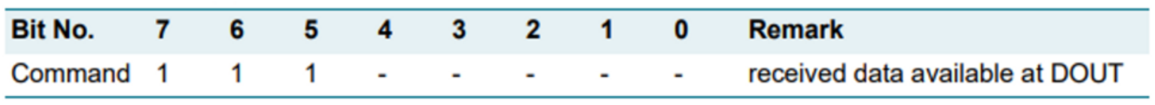
This part of the code uses manchester encoding to decode the received values

BYTE rfid_tx_rx_byte (BYTE tx_data, BYTE get_response)

We transmit and receive data serially using the HAL library transmit and receive function. As an alternative we also plan to use bit by bit transmission on rising edge of MISO pin, i.e., whenever the Slave (HTRC) wants to send the data MISO pin will turn from 0 to 1 and the Master (microcontroller) will start reading the data bit by bit.

void rfid_start_read_tag (void)

This command is used to read the demodulated bit stream from a transponder: After the assertion of the three command bits, the HTRC110 instantaneously switches to READ_TAG-mode and transmits the demodulated, filtered and digitized data from the transponder. Data comes out and should be decoded by the microcontroller. READ_TAG-mode is terminated by a low to high transition at SCLK.



Bit No.	7	6	5	4	3	2	1	0	Remark
Command	1	1	1	-	-	-	-	-	received data available at DOUT

Figure 19: READ_TAG command

void rfid_end_read_tag (void)

To end the process we need to set the clock bit to HIGH.

Testing and debugging

Once all our components had arrived, we started testing the individual subsystems:

1. LEDs (GPIO pins)
2. Buzzer (PWM)
3. Lock System
4. ISR (Interrupt Service Routine) testing
5. Transmitting and Receiving via SPI using Arduino as slave
6. Testing of main code

The detailed description of each of the following experiments have been provided below:

We are using a top down approach while testing starting with the most basic components such as LEDs and scaling up the level. These tests also let us verify whether the PWM or GPIO pins are working as required by our code.

LEDs (GPIO pins)

In our circuit, we make use of two LEDs. The green LED is supposed to glow if the card read is a valid card and access is to be provided. The red LED glows if the RFID card read, is not a valid card. These LEDs are connected to the PC2 and PC3 pins of our STM32 μcontroller. We test to see if these LEDs glow when the appropriate signals are applied to these pins.

This setup was also done to ensure whether all the GPIO pins received the correct signals.

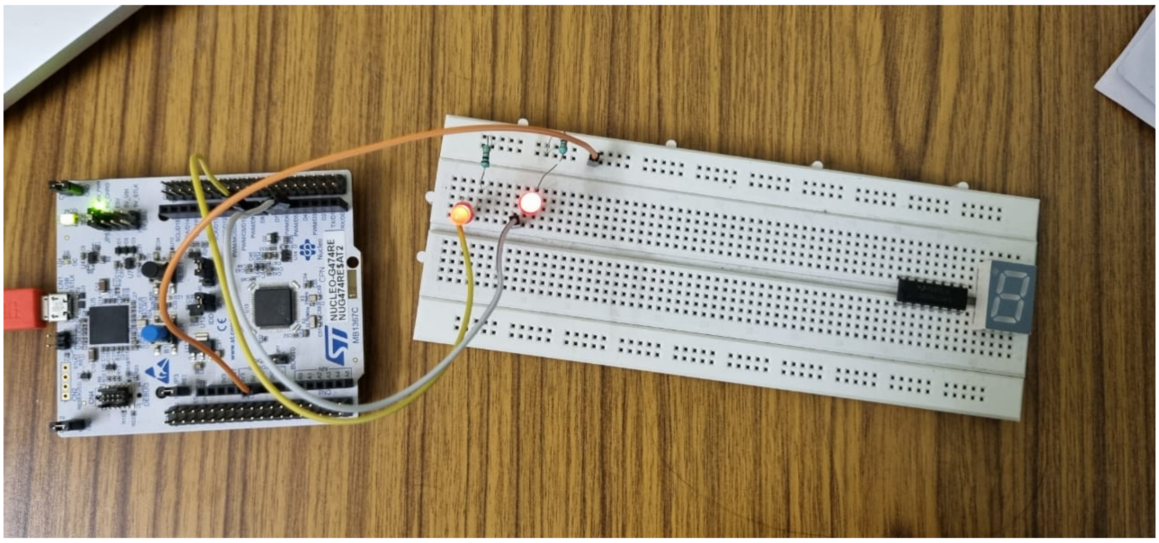


Figure 20: LED Test

Buzzer (PWM)

The buzzer is supposed to buzz differently when access is given and denied. When access is given, the buzzer buzzes with a short beep. When access is denied, the buzzer buzzes with a beep- beep. PWM is used to control these different buzzing. Thus, the buzzer was tested by applying pulses of different duty cycles to the buzzer, supplied from the μcontroller, specifically, the PA0 pin.

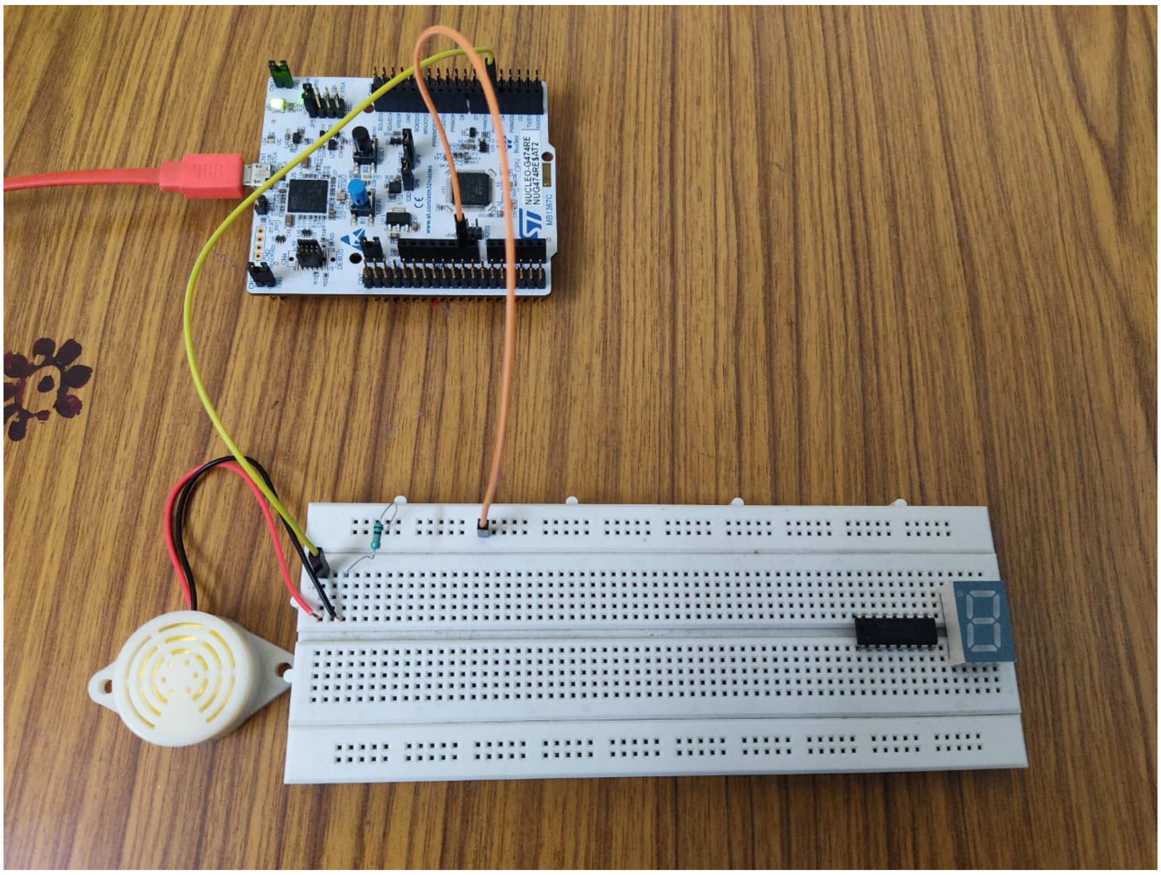


Figure 21: Buzzer test

Lock System

The lock is interfaced with the μcontroller via a relay. An *in* signal is supplied to the relay from the PC10 pin of the μcontroller. When the *in* signal is high, the lock unlocks while the lock stays locked for the duration for which the *in* signal is low.

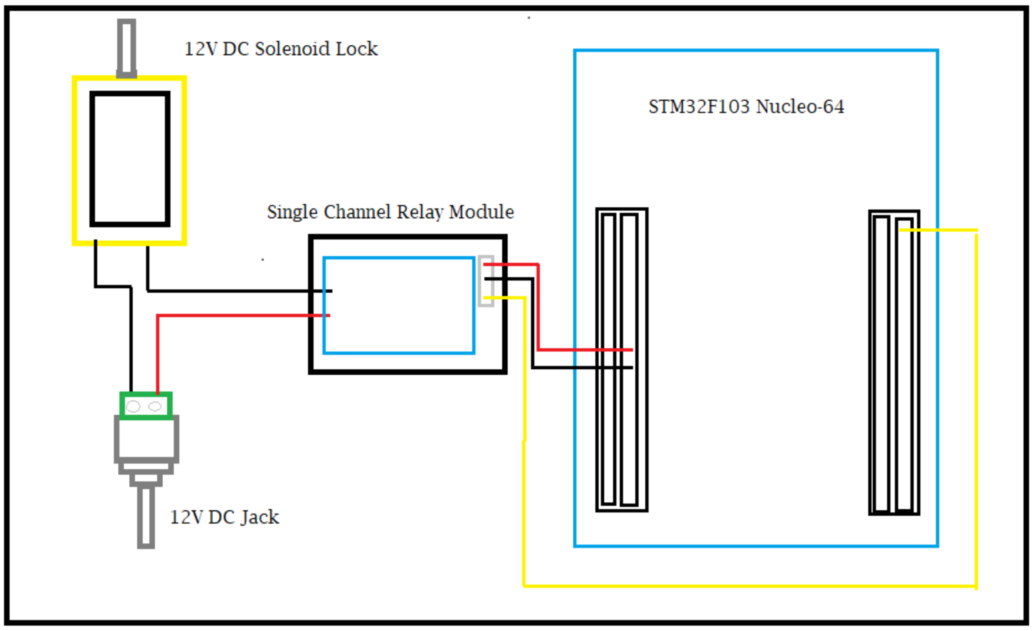


Figure 22: Interfacing of the Lock with the board using a relay

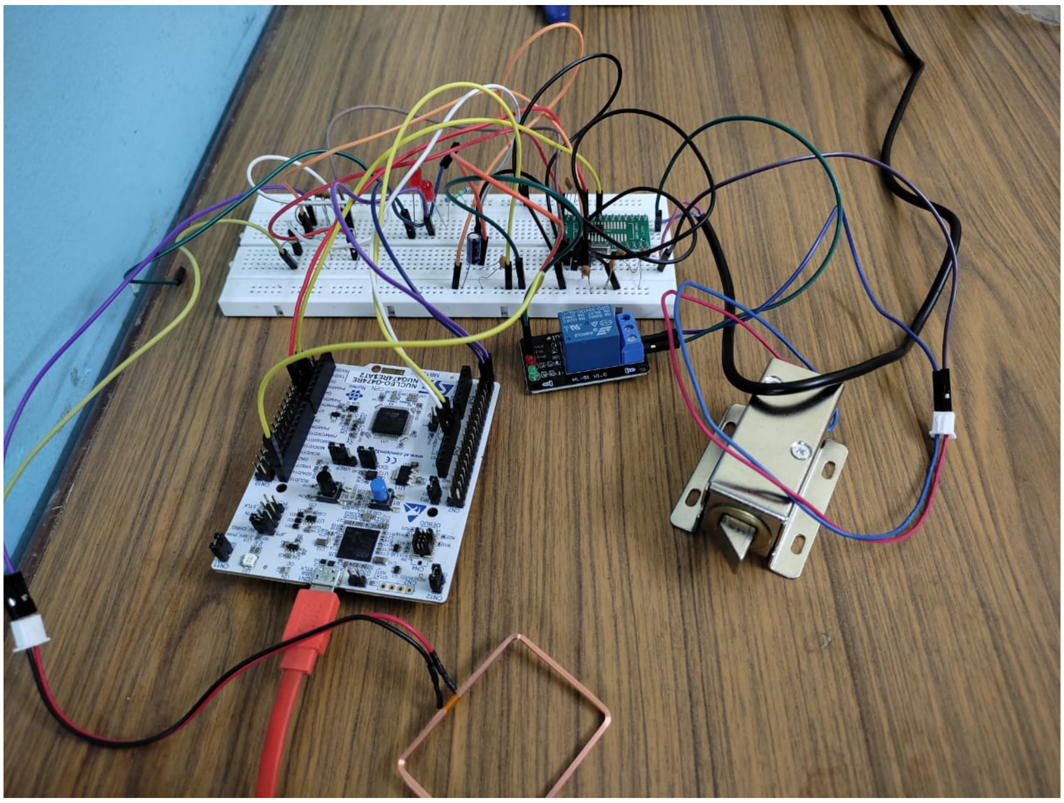


Figure 23: Actual circuit for testing the lock system

Code Testing

We faced a lot of challenges while trying to debug our code and get it to work. Thus, we tested the various sub-sections of our code (and their hardware interfacing) which are as follows:

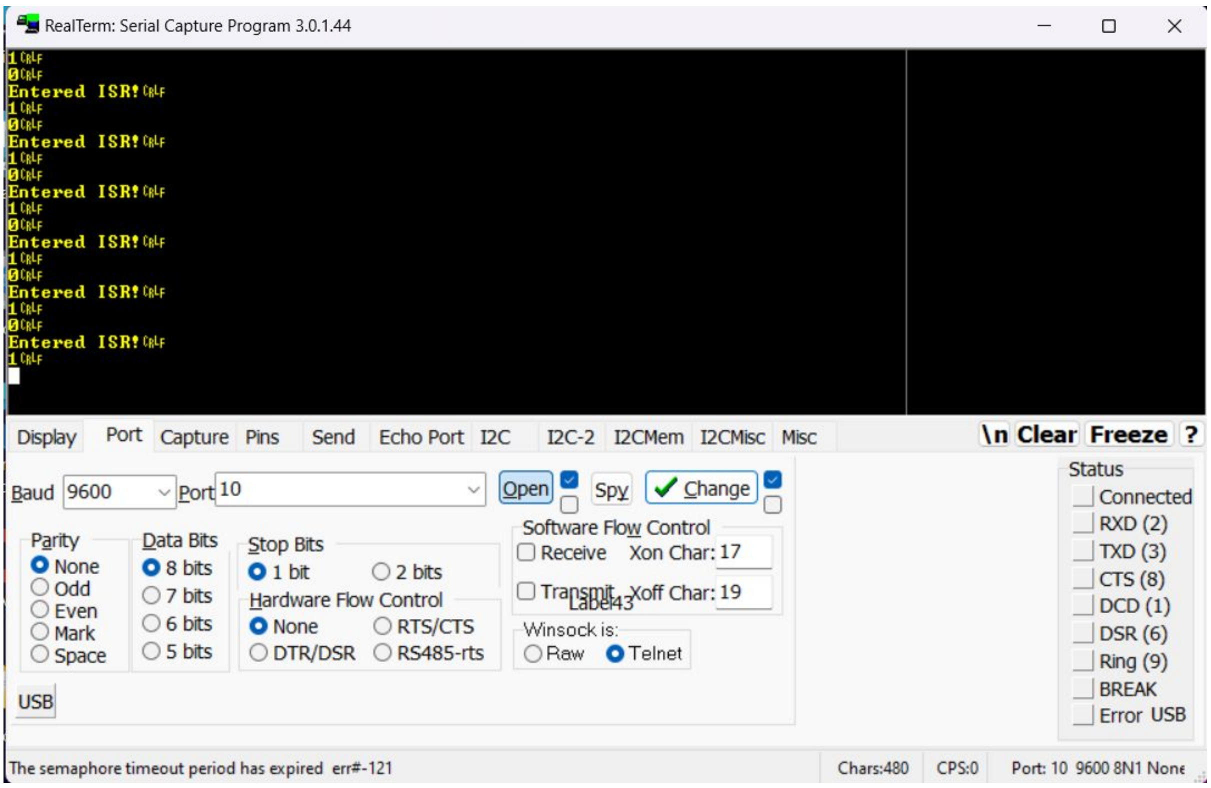
1. ISR (Interrupt Service Routine) testing
2. Transmitting and Receiving via SPI using Arduino as a slave
3. Testing ISR and SPI simultaneously
4. Testing the main code

The detailed description of the tests are as follows:

ISR (Interrupt Service Routine) testing:

The reason for conducting this test is that whenever our HTRC wants to send data to our μ controller, it does so during the rising edge of MISO.

To check whether the interrupts function properly, we decided to test them. Our code requires to enter the ISR routine on the rising edge MISO. In this test, we checked whether the function was being redirected to the ISR function routine when MISO was made low to high externally. Thus, MISO was set as an output pin for this purpose.



RealTerm: Serial Capture Program 3.0.1.44

```
1
0
Entered ISR!
1
0
Entered ISR!
1
0
Entered ISR!
1
0
Entered ISR!
1
0
Entered ISR!
1
0
Entered ISR!
1
```

Baud 9600 Port 10

Parity: [✓ None] Odd Even Mark Space

Data Bits: [✓ 8 bits] 7 bits 6 bits 5 bits

Stop Bits: [✓ 1 bit] 2 bits

Hardware Flow Control: [✓ None] RTS/CTS DTR/DSR RS485-rts

Software Flow Control: Receive Xon Char: 17; Transmit Xoff Char: 19

Winsock is: Raw [✓ Telnet]

Status: Connected, RXD (2), TXD (3), CTS (8), DCD (1), DSR (6), Ring (9), BREAK, Error USB

The semaphore timeout period has expired err#-121 Chars:480 CPS:0 Port: 10 9600 8N1 None

Figure 24: Image showing that the code entered ISR when a rising edge is detected on MISO

Transmitting and Receiving via SPI using Arduino as slave:

In our main code, we transmit and receive data using SPI. However, using our main code, we were receiving a constant 255 from HTRC110. Thus, we needed to figure out why this was happening and also needed to check what it was transmitting to HTRC110.

For this purpose, we decided to connect the STM32 and Arduino boards in a master-slave fashion to check whether our STM32 is able to transmit the right values and correctly receive what is being sent to it from the Arduino board.

The details of this test are as follows:

- STM32 μcontroller was the master
- Arduino μcontroller was the slave
- A baud rate of 9600

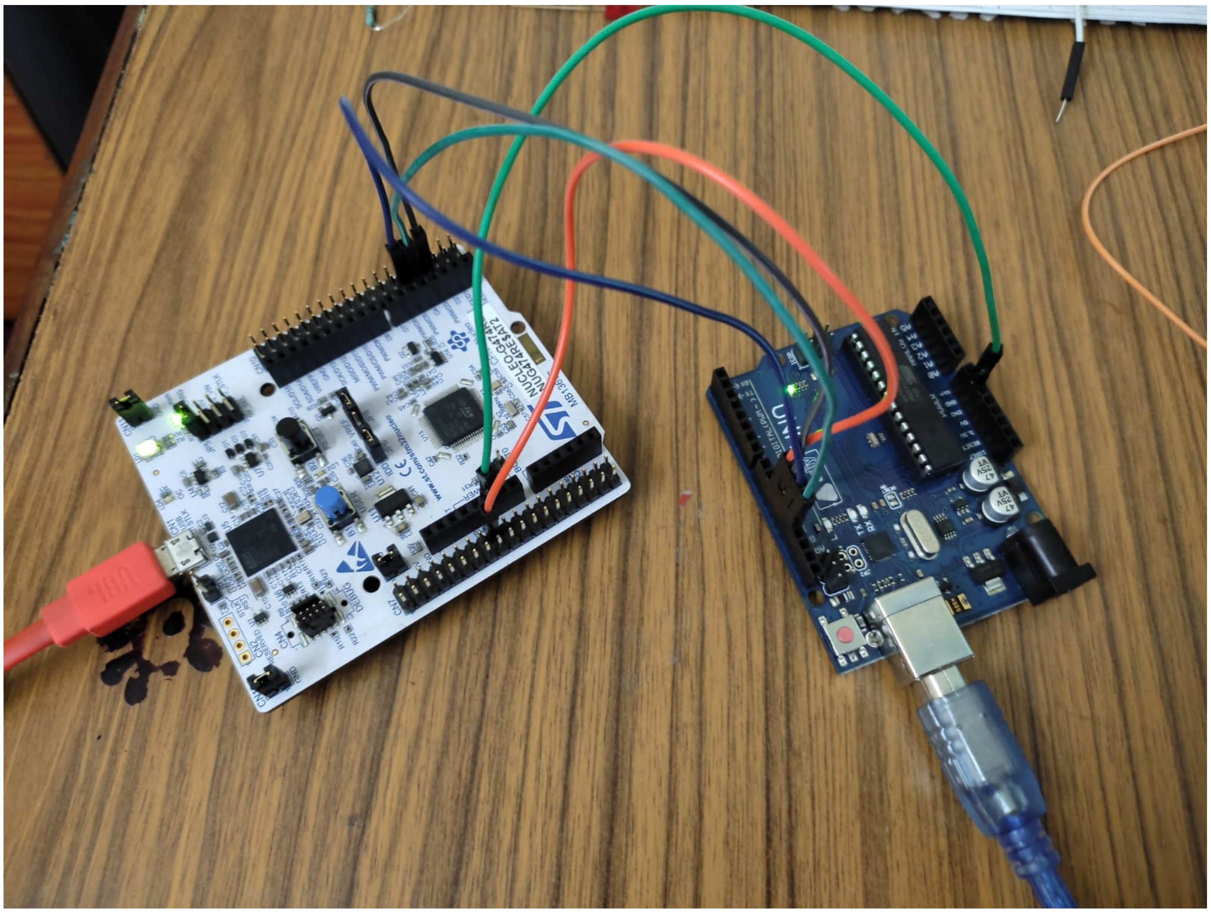


Figure 25: SPI testing using arduino as slave

Testing the Main Code:

We also tried running our main code and were debugging it on the way. The various components of the main code that we checked were as follows:

1. **Initialization** - Whether our HTRC is being initialized correctly
2. **State change** - Whether our state variable correctly transitions between the different states and executes the respective loop.
3. **Communication** between the μcontroller and HTRC
4. **Antenna** - Whether our antenna is able to send out signals correctly, power the tag adequately and read back the data from the tag.

Results

We were able to successfully test cases 1-3 and obtained the desired results. We also partially verified our main code. However, the ISR testing and transmitting and receiving via SPI using arduino as slave did not give the desired outputs.

1. We expected the function to enter the ISR function when MISO was changed from low to high. However, this did not happen. Since MISO had been set as an output pin, we weren't able to read from it and check whether it was transitioning from low to high. Thus, we connected a dummy pin to the MISO pin and set this dummy pin as an input pin. Hence, we used this dummy pin to check for the rising edge of MISO.

   Hence, in the end, we were able to infer that the code was entering into the ISR function during the rising edge of MISO.

2. The HAL_SPI_Transmit_IT function and SPI.transfer functions did not seem to be working. Thus, we replaced these functions and coded it to manually send and receive data bit by bit. This was initially receiving double of the data being sent. Upon further debugging, we realised that the SPI.transfer function did not work because SPI.transfer and Serial.println don't work together as they both make use of interrupts and thus, SPI was unable to go into it's interrupt. Thus, we got rid of the print statements but that made it difficult to monitor what was being transferred and received. Hence, we replaced printing by trying to make LEDs blink but that too got extremely tough to monitor.

Upon further debugging, we realised that we had to get rid of SPI.transfer entirely as HTRC110 doesn't have a chip select. Hence, we then shifted to bit banging - manually sending the data a single bit at a time.

Results using bit banging

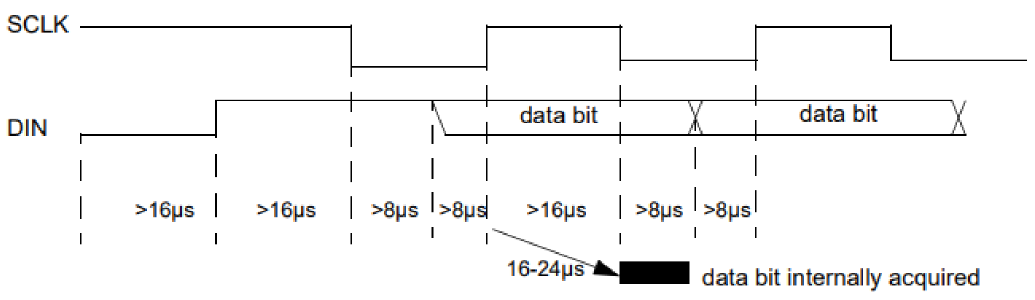


Figure 26: Expected sequence on how the data is supposed to be transferred to HTRC110

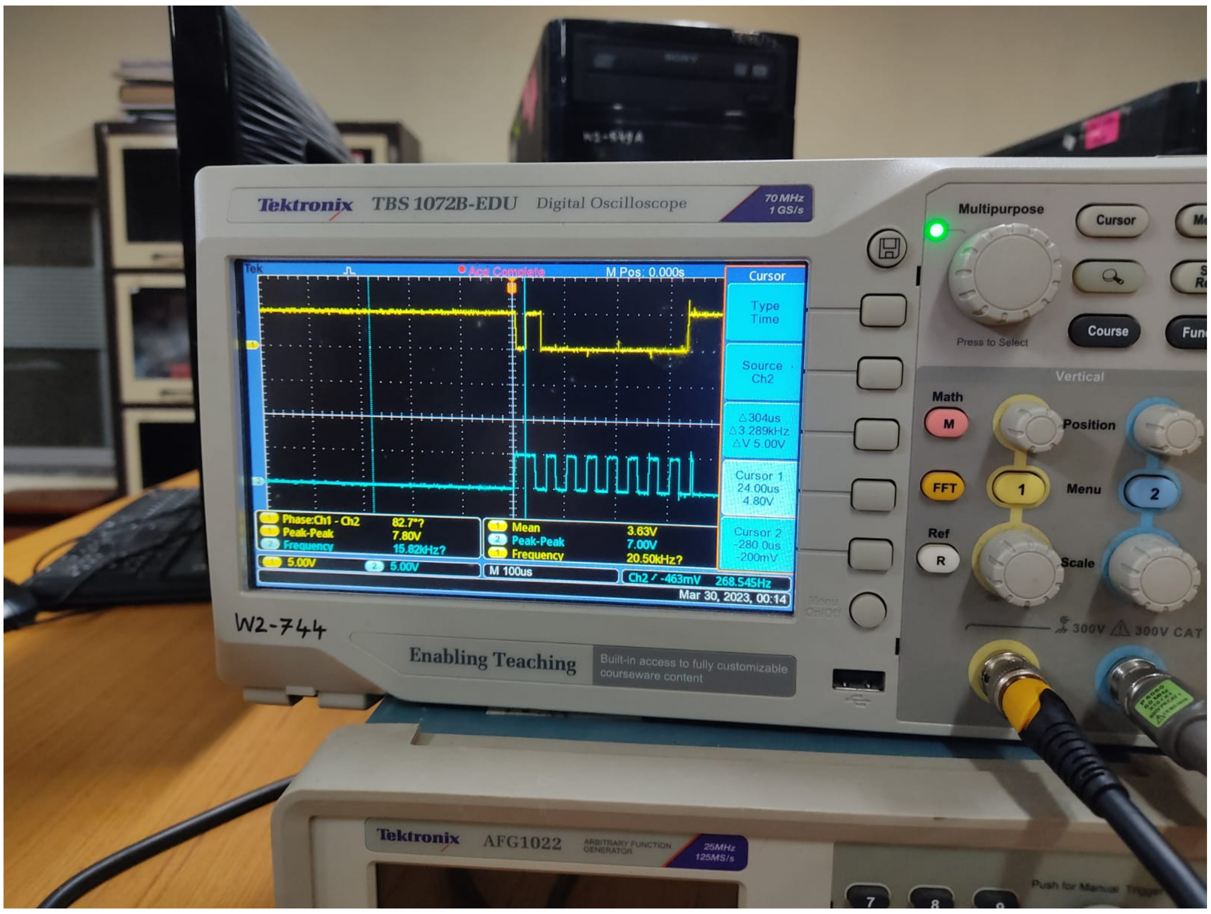


Figure 27: The data sent to HTRC110 using bit banging: Channel 1 has the data and Channel 2 has the clock

We configured the SET_CONFIG_PAGEs and GET_CONFIG_PAGEs as needed and used the oscilloscope to check the clock, the input signal to the HTRC and the output signal obtained from the HTRC. This showed that we were sending correct clock and input signals to the HTRC but were obtaining 0 response from it.

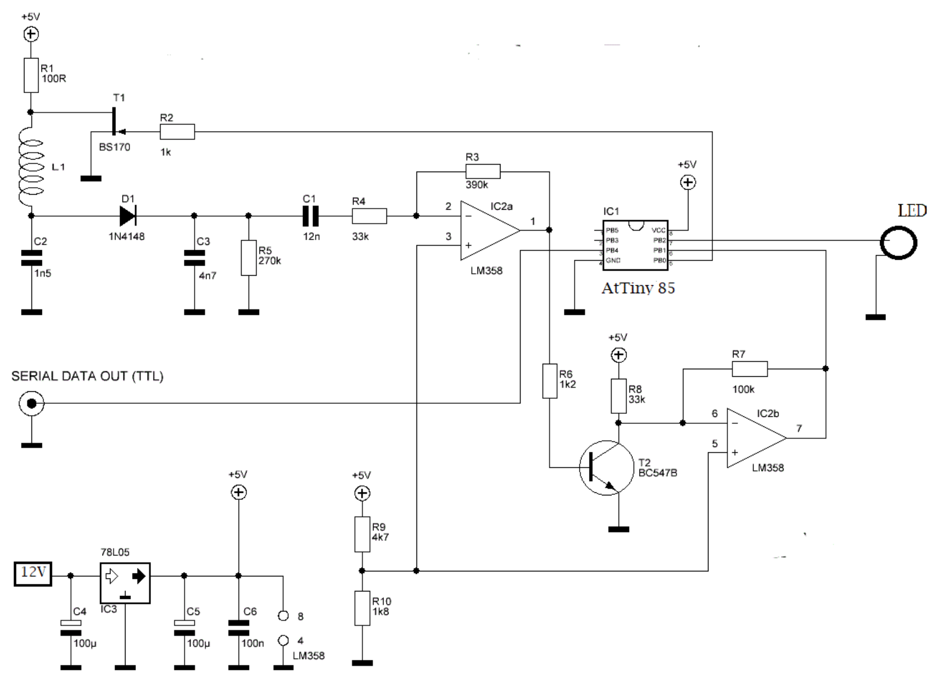
We spent a lot of time trying to debug this including using another HTRC110 (incase the first one had some defect) but that didn’t work either.

Finally, it was pointed out by Tallur Sir that our HTRC was being given 3.3V (since we were using STM32) and thus, the HTRC was probably even reading high signals as low as its V_TH> 3.3V. Thus, we tried using a level shifter to upconvert the signal to 5V before feeding it to the HTRC and continued debugging.

However, none of what we tried seemed to be working and towards the end, we started looking for alternatives to HTRC110. There weren’t any feasible options for RFID reader ICs at 125kHz. We then tried to find out ways in which we won’t be requiring any reader IC and do its functionalities with the microcontroller itself. We then took ATtiny85 as a replacement since it worked on 5V TTL logic and was easily available in WEL.

Approach 2

While looking for alternate approaches we were trying to come up with something so that we can omit the reader IC part of our circuit and use microcontroller directly with the antenna. To power the antenna we require a 125kHz wave which we generated using ATtiny 85 microcontroller. We didn’t use STM32 as its logic levels are 3.3V which does not match with our circuit specifications.



Basic idea

The ATtiny85 uses the PWM function to produce an 125 kHz square wave signal. This signal comes out from PB0 pin. On the falling edge of the PB0 (Logic '0'), the T1 does not conduct. So the antenna is energized through resistor R1 with +5V. When PB0 pin rises (Logic 1) the T1 conducts and one side of L1 goes to GND. The L1 goes in parallel with the capacitor creating an LC oscillator. These transitions of L1 to logic '1' and logic '0' are made at a frequency of 125 kHz.

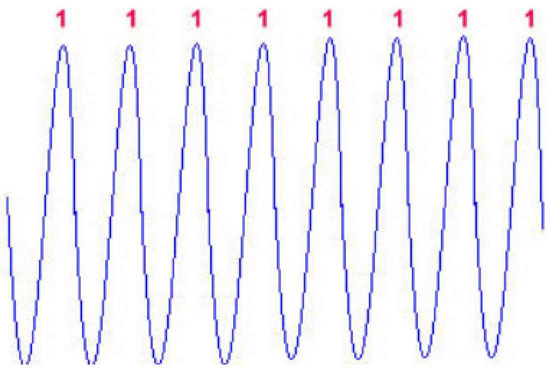


Figure 29: The 125 kHz waveform that is transmitted from antenna and capacitance (C2 in Fig.)

The RFID reader provides energy to the tag by creating an electromagnetic field. The energy transmission between the RFID reader and the tag is the same as transformers convert the voltage from the 220V AC power network to 12V AC, based on the magnetic field that creates the primary coil. In our case the primary coil is the RFID reader (antenna) and the secondary coil is the RFID tag. The only difference is that on RFID circuits there is no iron core between the two coils (one coil is on the reader side and the other coil is in to the RFID tag). The D1, C3 and R5 components constitute an AM signal demodulator.

For reading data from the tag, we employ amplitude modulation. When a tag wants to send a logic '0' to the reader it puts a "load" to its power supply line to request more power from the reader. That makes a small voltage drop on the RFID reader side. That voltage level is logic 0. Simultaneously, as long as the reader transmits the 125 kHz signal it reads the voltage of the transmitted signal through the diode, resistors and capacitors. When the tag drops the voltage, the reader reads this voltage drop as logic '0'. When the Tag doesn't require any additional power, it doesn't make a voltage drop. That is logic 1. The 'Ones' or 'Zeros' length depends on the serial transmission data rate.

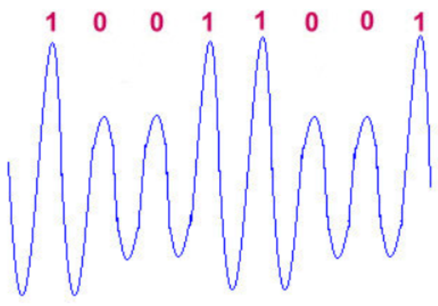


Figure 30: Example of transmitted data

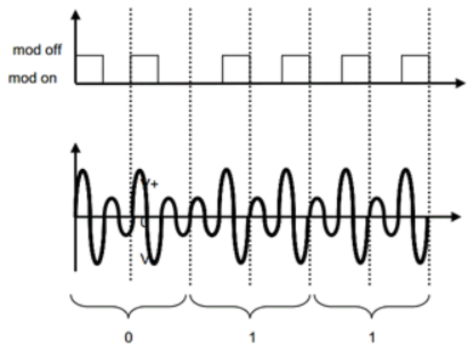
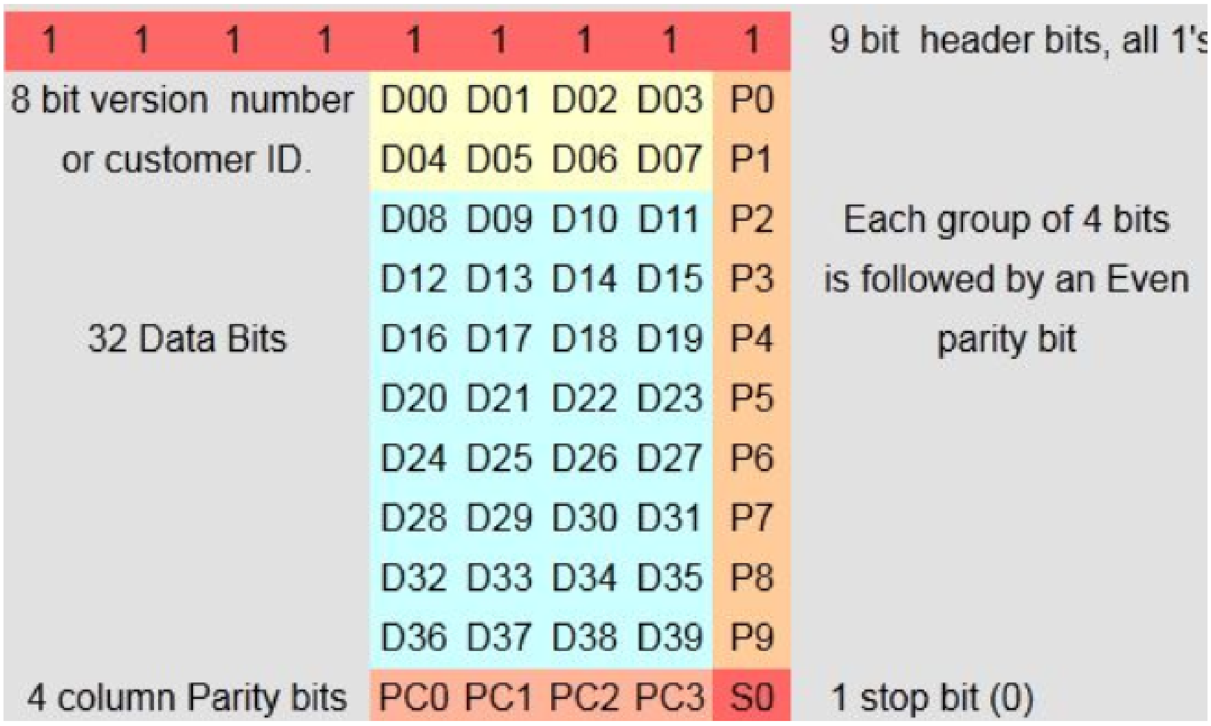


Figure 31: An alternative picture of the PSK modulation.

The 125kHz RFID tag transmits 64 bits which are as follows

1. The first 9 bits are the start communication bits (always '1').
2. The next 4 bits are the Low Significant Bits of the customer ID(D00,...,D03).
3. The next 1 bit (P0) is the Even parity bit of the previous 4 bits .
4. The next 4 bits are the High Significant Bits of the customer ID (D04,...,D07).
5. The next 1 bit (P1) is the Even parity bit of the previous 4 bits.
6. The next 4 bits are first part of the 32-bit Tag's serial number (D08,...,D11).
7. The PC0 bit is the Even parity bit of bits D00, D04, D08, D12, D16, D20, D24, D28, D32 and D36 (the bits on the same column).
8. The PC1, PC2, PC3 bits represent the parity bits of the next 3 columns.



1 1 1 1	1	1	1	1	1	9 bit header bits, all 1'
8 bit version number	D00	D01	D02	D03	P0	
or customer ID.	D04	D05	D06	D07	P1	
	D08	D09	D10	D11	P2	Each group of 4 bits
	D12	D13	D14	D15	P3	is followed by an Even
32 Data Bits	D16	D17	D18	D19	P4	parity bit
	D20	D21	D22	D23	P5	
	D24	D25	D26	D27	P6	
	D28	D29	D30	D31	P7	
	D32	D33	D34	D35	P8	
	D36	D37	D38	D39	P9	
4 column Parity bits	PC0	PC1	PC2	PC3	S0	1 stop bit (0)

Figure 32: RFID tag content

The data verification is been done from ATtiny85 by calculating the even parity bit of each line and each column with the parity bits that had been received form the RFID Tag transmitted data.

Working of our circuit

The PWM signal of 125 kHz is generated at PB0 of the ATtiny85 which then passes through a MOSFET switch for improved thresholding for 0 and 5V. We then used and opamp for thresholding the amplitude modulated signals and extract logical data from it. Its output is fed to a transistor which improves the thresholding and makes it 0 and 5V and this signal goes to the pin PB1 of microcontroller

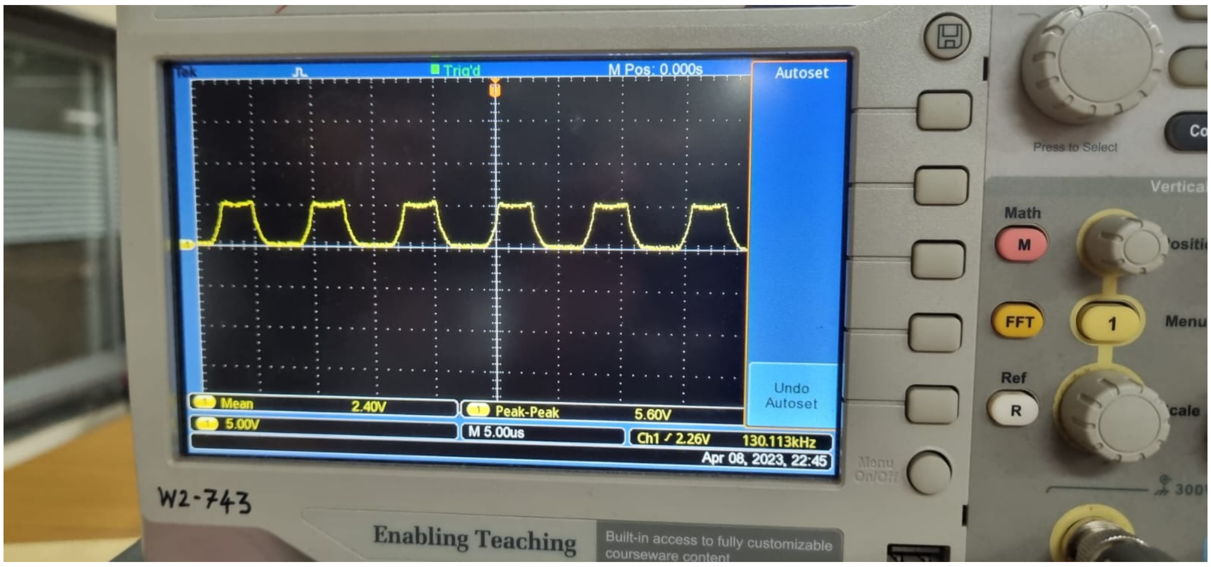


Figure 33: The PWM signal observed at one of the terminals of the Antenna. As can be seen, the frequency is approximately 125kHz

---

Working of code

The code used internal timers of the microcontroller to generate a PWM signal at PB0. Whenever a signal is detected at PB1 an interrupt is fired which sets the flag for input signal to 1. We also had a timer overflow interrupt which is called at regular intervals and checks the value for the input signal flag. If the input signal flag is high for considerable time, we execute the signal reading part. The tag UIDs are read at pin PB4 of the ATtiny which works as Serial Data Out (TTL)

```
ISR(TIM0_OVF_vect)
{
  if(counter1 < 255)
    {counter1++;}
  if((signal_in == TRUE) && counter1 > 60)
  {
    signal_in = FALSE;
    if(bit_is_set(PINB,SIGNAL_IN))
    {
      PORTB |= (1<<PWM_PIN1);
      pulseUpDown = UP;
    }
    else
    {
      PORTB &= ~(1<<PWM_PIN1);
      pulseUpDown = DOWN;
    }
    counter1 = 0;
  }
  signal_in = FALSE;
}
```

When the variable pulseUpDown is UP for a certain time and then goes DOWN, we start reading the tag serial number. The code shifts the contents of the "dataBuffer" array one bit to the left using the "<<=" operator, reads the current status of "pulseUpDown" and checks if it is UP (indicating a logic HIGH signal). If "pulseUpDown" is UP, the code sets the first bit (bit 0) of the "dataBuffer[i/2]" byte to 1, increments a "ones" counter, increments a corresponding column of the "dataBuffer" array, and adds an extra delay by setting the "LED" pin low.

Next code reads the status of the parity bit by checking if "pulseUpDown" is UP. If it is, the code increments the "ones" counter again. The code then calculates the parity of the "ones" counter by taking the modulo 2 of its value and checks if it is odd. If it is odd and the current iteration of the for loop is not the 11th 4-bit array (which is used for column parity), the "parity" variable is set to FALSE. Finally, an extra delay is added using the "LED" pin.

```
void sendTxD (unsigned char txd)
{
  unsigned char bitCounter = 0;
  PORTB &= ~(1 << TXD);
    _delay_us(ONE_BIT_DELAY);
  for(bitCounter=8; bitCounter>0; bitCounter--)
  {
    if (bit_is_clear(txd, 0))
      PORTB &= ~(1 << TXD);
    else
      PORTB |= (1 << TXD);
    _delay_us(ONE_BIT_DELAY);
    txd >>= 1;
  }
  PORTB |= (1 << TXD);
  _delay_us(500);
}
```

The sendTXD function sends a byte of data through a digital pin TXD using a protocol known as asynchronous serial communication. It starts by setting bitCounter to 0 and clearing the TXD pin. After a delay of one bit time, the function enters a loop that iterates eight times, once for each bit of the txd byte.

During each iteration of the loop, the least significant bit (LSB) of txd is checked. If the LSB is clear (i.e., 0), TXD is cleared (i.e., set to logic 0) by clearing the corresponding bit in the PORTB register. If the LSB is set (i.e., 1), TXD is set to logic 1 by setting the corresponding bit in the PORTB register. The function then waits for one bit time and shifts the txd byte one bit to the right, effectively shifting out the previously sent bit.

After all eight bits have been sent, the function sets the TXD pin to logic 1, adds a delay of 500 microseconds, and returns. The delay after sending the byte is to allow the receiving device to process the data before the next byte is sent.

Initially both the interrupts were not working together. So we tested both the interrupts individually. For testing of external interrupt we sent pulses to PB1 pin to check if it was working fine. For the timer we needed two things:

1. Generation of 125 kHz square wave: We calculated and set the value of prescalar to 32
2. Interrupts: We were initially using timer1 which was interfering with the external interrupt. We then shifted to timer0 which solved the problem

Supply

Since our circuit and microcontroller required 5V supply we used a 12V adapter and a voltage converter 78L05 to get 5V. Also for the opamp we needed negative supply which we provided by using an external 9V battery. We used 78L05 for converting to 5V and then connected its positive terminal to ground of our main circuit and got -5V at its negative terminal with respect to our main circuit.

We soldered all the components on a perforated board. We then did short test for all the components to ensure that the circuit works properly

We successfully made an RFID card detector. The LED was attached to pin PB2 of microcontroller which glows whenever a tag is in vicinity. For our application of lock system we required another detector LED which glows when an authorized person taps the card and along with that the lock must unlock.

CAD

We made the case for our project using laser cutting on acrylic sheet. The cad file was made using Makercase

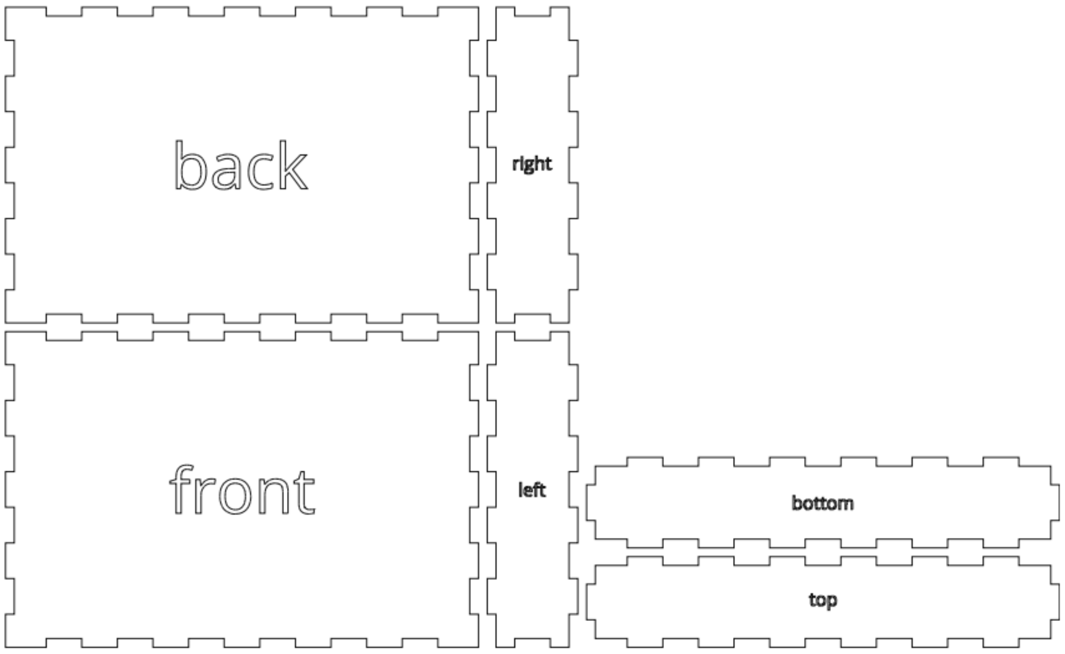


Figure 34: CAD layout

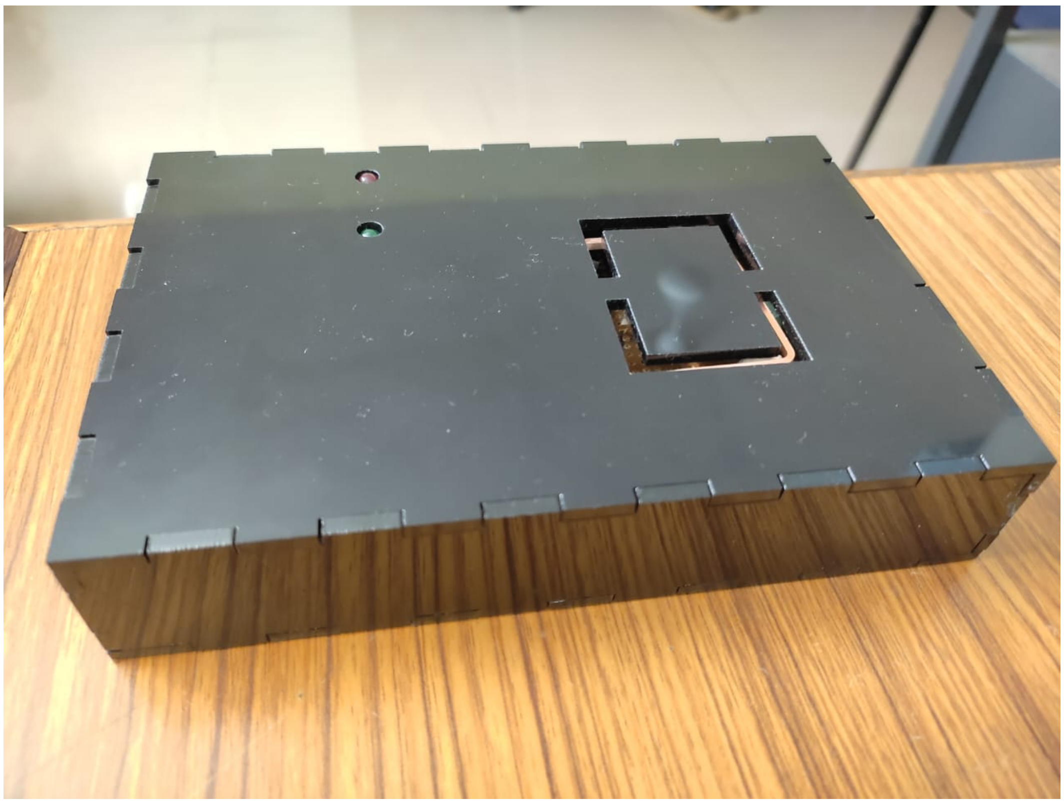


Figure 35: Final CAD

Major problems faced

Problem	Solution
Wrong Antenna received so we bought a new one but since it had no datasheet we were unsure how to proceed	We found the specifications using the LCR meter available at WEL. We consulted Prof. Shevgaonkar regarding how to do impedance matching. He guided us in and gave us two solutions: 1. To remove resistances from our impedance matching circuit since they contribute to power loss 2. To use inductors instead of resistances to sort the above
No chip select in HTRC, SPI protocol could not be used	As suggested by Maheshwar Sir we shifted to bit banging, i.e., bit by bit transfer of data. By looking into the datasheet of HTRC we figured
When a green LED was added at the end the circuit stopped working	We short tested each and every trace to ensure the connections were correct. We found that a a trace wasn't shorted even though it looked like it was. We then resoldered it.

Results and Observations

Thus, our fully functional model performs the following:

- The user brings an access card containing an RFID tag in the vicinity of the antenna. Since the antenna is constantly sending out electromagnetic waves of frequency 125kHz, these waves couple with the coil present in the tag, thus powering it.
- The tag now sends back AM modulated waves which are captured by the antenna.
- The captured signal is demodulated and decoded. AtTiny85 then checks the UID of the tag which is present in the decoded signal.
- If the UID is present in the list of valid UIDs registered in the system, a red LED glows. If the UID isn't one of the valid ones, no LED glows

In this way, we created an RFID based detector.

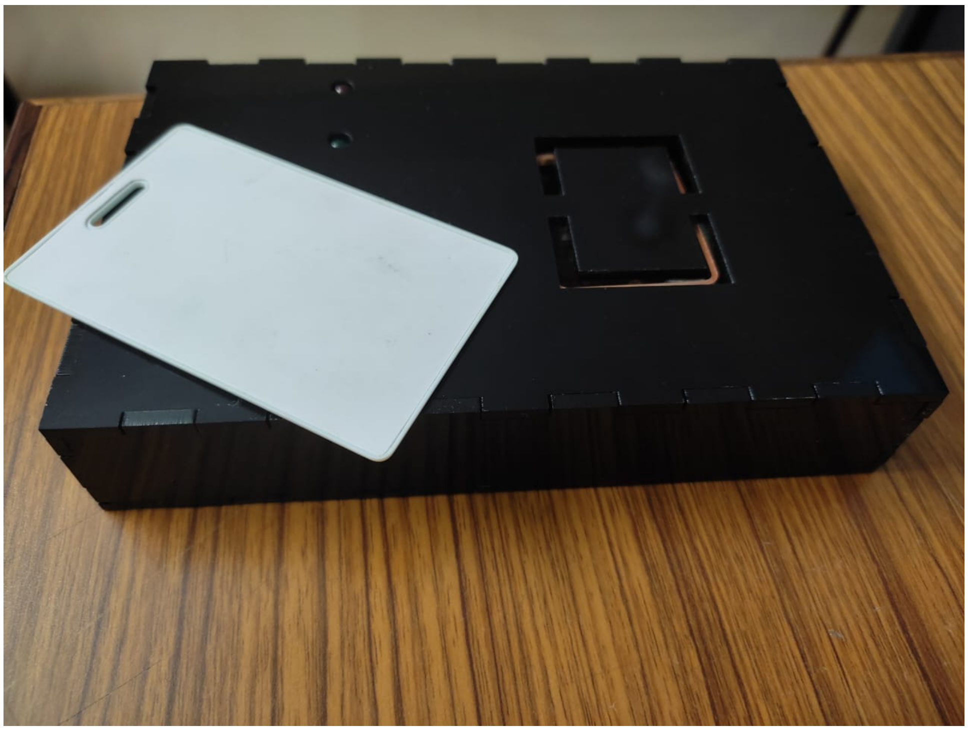


Figure 36

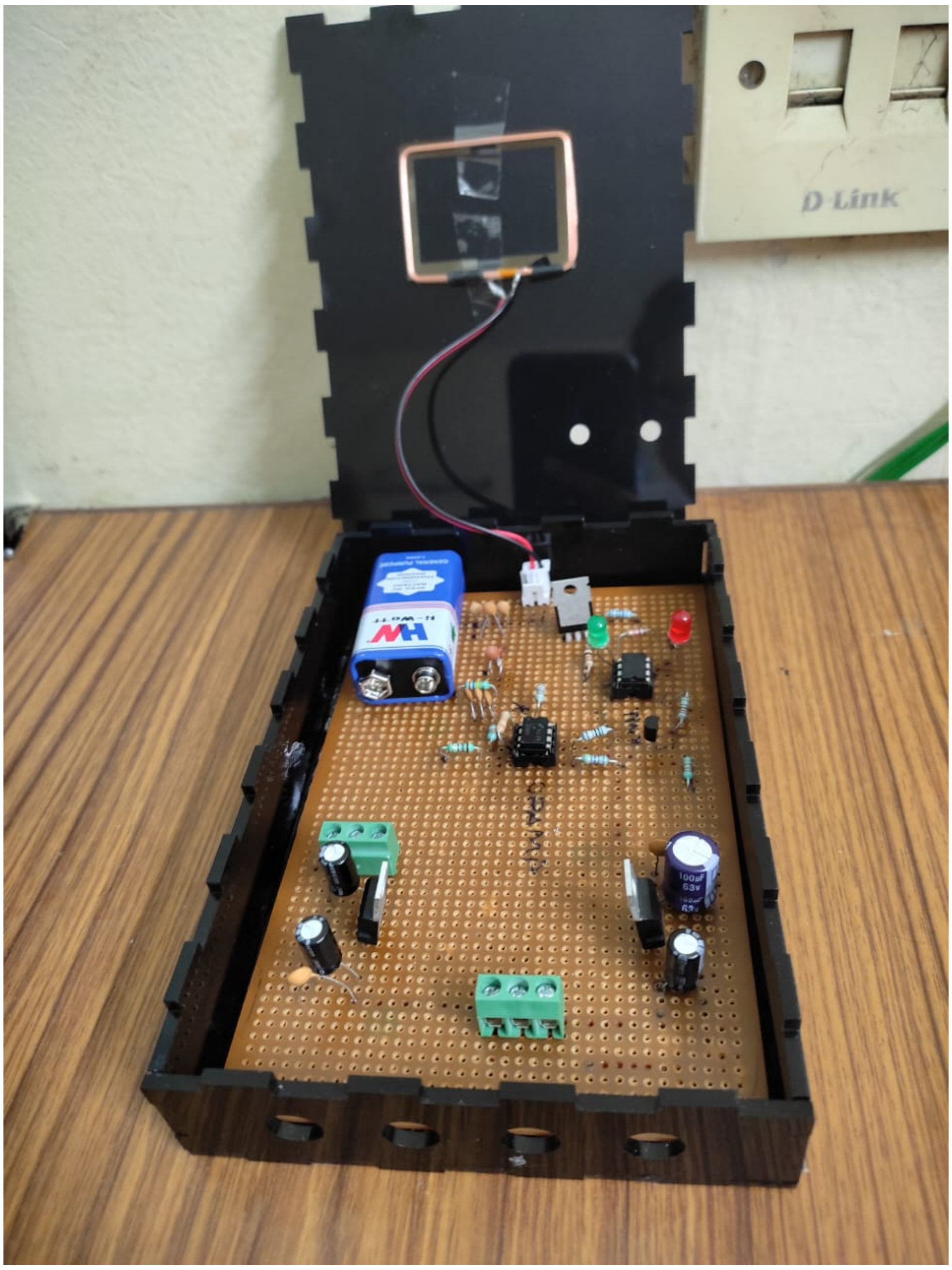


Figure 37

Future Work

Our project was initially an RFID based smart access system. However, due to several challenges faced on the way, our statement then got converted to an RFID based detector.

Thus, there are some improvements that can be made to the current project. They have been stated as follows:

1. Along with the LED that already glows, another LED could be added such that if the card is valid, the green LED glows and if the card is invalid, the red LED glows. This would give a clear distinction between the three cases of a card being valid, the card being invalid and the card not being read at all.
2. A buzzer could be added to the system such that there is a sound indication of access of being denied. For example, if the card is valid, the buzzer sounds a short beep-beep and if the card is invalid, the buzzer sounds a long beep.
3. The detector can be interfaced with a lock. Thus, when the card read is valid, a green LED glows and the buzzer buzzes a beep-beep. In addition to this, the connected lock too unlocks. However, in case if an invalid UID, the lock doesn't unlock.

Bill of Materials

Component	Price
RDM6300 RFID Reader	308/-
Solenoid Lock	510/-
12V DC adapter	299/-
Buzzer	47/-
HTRC110	417/-
ATtiny85 development board	588/-
PCB	3010/-
MOSFET, ATtiny85, 7805 (From Mangaldeep)	315/-
Battery, Connectors, LEDs (From Mangaldeep)	140/-
Total	5634/-

Links to code

- **Project GitHub:** https://github.com/raavi02/RFID-Reader.git
- Main code: https://github.com/raavi02/RFID-Reader/tree/main/Code/RFID_v2
- Code for testing LED and Buzzer: https://github.com/raavi02/RFID-Reader/tree/main/Testing_codes/Blink
- Code for testing lock: https://github.com/raavi02/RFID-Reader/tree/main/Testing_codes/Relay

- Code for ISR testing: https://github.com/raavi02/RFID-Reader/tree/main/Testing_codes/ISR_SPI_test
- Code for transmitting and receiving via SPI using Arduino as a slave
    - Master code: https://github.com/raavi02/RFID-Reader/tree/main/Testing_codes/Master
    - Slave code: https://github.com/raavi02/RFID-Reader/tree/main/Testing_codes/Slave

Links to demo videos

- Video of LED being tested: https://drive.google.com/file/d/1EyMysPxP9u7EGAzq5FO9EJNUUMJc6Ek8/view?usp=share_link
- Video of lock being tested: https://drive.google.com/file/d/148gnD1b-4uz3d8WK9Ean8Pe4PmqgV1bO/view?usp=share_link
- Video of lock being tested:
    - Access denied: https://drive.google.com/file/d/1j0bYYq0CShPTFN3pQQHpTqq3cuelvvto/view?usp=share_link
    - Access given: https://drive.google.com/file/d/1TvtvJnA8GSB0nTYrwB_pOiIY2mv8pnJa/view?usp=share_link

References

- Read/write devices based on HITAG read/write IC HTRC110
- HTRC datasheet
- STM32 SPI Tutorial
- ATtiny 85 datasheet

Conclusions

This lab has been quite a fruitful experience, exposing us to several new concepts and skills. We learned a lot of things on the way including time management, team spirit and several new concepts. However, the major take away for us from this lab would be the importance of a plan B and implementing it on time. We had initially not given any importance to implementing a plan B and just assumed that whatever we had planned in our first milestone would work out. But it is only after what we experienced in the lab do we now realize how important it is to have alternatives already planned in the start and implement these alternatives on time. The professors, TAs and RAs were also very helpful thus, making this lab an excellent learning experience.