

**VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY**



EMBEDDED SYSTEM PROJECT

TOPIC:

RFID CARD READER

CLASS TT01 --- GROUP 04 --- SEM 232

SUBMISSION DATE: 07/05/2024

Instructor: MSc. BÙI QUỐC BẢO

Student	Student ID
Nguyễn Nhật Tuấn Minh	2151229
Vũ Huy Bảo	2151177
Châu Hồng Phước	2051175

CONTENTS

	Page
I. OVERVIEW	5
1. RFID information	5
1.1. RFID	5
1.2. RFID operation	5
1.3. RFID practical application	5
2. RFID tags.....	7
3. Types of RFID system.....	8
4. Choice of components	9
1.4.1 Project components	9
1.4.2 Components reasoning.....	9
II. MFRC522 AND MIFARE CARD IMPLEMENTATION IN RFID SYSTEM.....	10
2.1. MFRC522 chip	10
2.1.1. Feature and benefits	11
2.1.2. Block diagram	13
2.1.3. Functional description	14
2.1.4. Digital interface.....	15
2.1.4.1. Serial peripheral interface	15
2.1.4.2. SPI read data.....	16
2.1.4.3. SPI write data.....	16
2.1.4.4. SPI address byte	16
2.1.5. Analog interface and contactless UART	17
2.1.5.1. General.....	17
2.1.5.2. TX p-driver	17
2.1.5.3. Serial data switch	19
2.1.5.4. MFIN and MFOUT interface support	20
2.1.6. FIFO buffer.....	20
2.1.6.1. Accessing the FIFO buffer	21
2.1.6.2. FIFO buffer status information	21
2.1.7. Interrupt request system.....	22
2.1.8. Timer unit	22
2.1.9. Power reduction mode (please read reference).....	23

2.1.10. Timer unit (please read reference)	23
2.2. MIFARE card (MF1S50)	23
2.2. MIFARE card (MF1S50)	24
2.2.1. Block diagram	24
2.2.1.1. Block desription.....	24
2.2.2. Communication principle	25
2.2.2.1. Request standard/all.....	26
2.2.2.2. Anticollision loop	26
2.2.2.3. Select card.....	26
2.2.2.4. Three pass authentication	26
2.2.3. Memory operations	27
2.2.4. Data integrity	28
2.2.5. Three pass authentication sequence	28
2.2.6. RF interface	29
2.2.7. Memory organization	30
2.2.7.1. Manufacturer block	30
2.2.7.2. Data block	31
2.2.7.3. Value block.....	31
2.2.7.4. Sector trailer	32
2.2.8. Memory access	33
2.2.8.1. Access conditions	34
2.2.8.2. Access conditions for the sector trailer	35
2.2.8.3. Access condition for data block.....	36
2.2.9. Command overview	37
2.2.10. ATQA and SAK responses	38
2.3. Interfacing with MFRC522 module and MF1S50	38
2.4. Other functions	44
III. DESIGN SCHEMATIC	53
IV. PROJECT RESULT	57
V. FUTURE DEVELOPMENT	61
VI. REFERENCE	74

I. OVERVIEW

1. RFID information

1.1. RFID

RFID (radio frequency identification) 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, animal or person.

1.2. RFID operation

Every RFID system consists of three components: a scanning antenna, a transceiver and a transponder:

- When the scanning antenna and transceiver are combined together, they are referred to as an RFID reader or interrogator. There are two types of RFID readers - fixed readers and mobile readers. The RFID reader is a network-connected device that can be portable or permanently attached. It uses radio waves to transmit signals based on their protocol(decoding process) which is that activate the tag and give permission to interrogator to access data inside memory of tag. Once activated, the tag sends a wave back to the antenna, where it is translated into data.
- The transponder is in the RFID tag itself. The read range for RFID tags varies based on factors including the type of tag, type of reader, RFID frequency and interference in the surrounding environment or from other RFID tags and readers. Tags that have a stronger power source also have a longer read range.

1.3. RFID practical application

These are the areas of greatest application of RFID technology:

- Payments in means of transport: public transports and tolls

In order to eliminate the delays on road tolls, Electronic Tolls Collection (ETC) collects tolls electronically. Cars do not need to stop when they pass through tolls, the payment is automatic if the car is enrolled in the program and if it is not, the system sends an event which triggers an action which refuses to let the car pass or it takes a crime photo.

If you do not drive, when you are travelling around the city you also use RFID in public transport, because of the entrance payment and the exit control can be done thanks to RFID tags in the prepaid transport passes.



- Logistics

Tagging goods and pallets, you can instantly know what goods you have, how many they are and how much space they occupy in your warehouse or back store just when they are entered.

Can you imagine what would happen if huge food companies did not have constant and instant control of the products they have in stock? When you shop at the supermarket you are not aware of how much the product has travelled, but thanks to RFID technology there is a trace of its movements.



2. RFID tags

RFID tags are made up of an integrated circuit (IC), an antenna and a substrate. The part of an RFID tag that encodes identifying information is called the RFID inlay.

There are two main types of RFID tags:

- **Active RFID.** An active RFID tag has its own power source, often a battery. this type of RFID tag is used in identify the location of the object at a far distance and bring about 98% accuracy.
- **Passive RFID.** A passive RFID tag receives its power from the reading antenna, whose electromagnetic wave induces a current in the RFID tag's antenna. This type is often used in low power application and have a small distance as compare to active RFID tag

There are also semi-passive RFID tags, meaning a battery runs the circuitry while communication is powered by the RFID reader.

Low-power, embedded non-volatile memory plays an important role in every RFID system. RFID tags typically hold less than 2,000 KB of data, including a unique identifier/serial number. Tags can be read-only or read-write, where data can be added by the reader or existing data overwritten.

The read range for RFID tags varies based on factors including type of tag, type of reader, RFID frequency, and interference in the surrounding environment or from other RFID tags and readers. There are some examples: MIFARE card: MF1S50 (MIFARE classic card),...

3. Types of RFID system

There are three main types of RFID systems: low frequency (LF), high frequency (HF) and ultra-high frequency (UHF). Microwave RFID is also available:

- Low-frequency RFID systems. These range from 30 KHz to 500 KHz, though the typical frequency is 125 KHz. LF RFID has short transmission ranges, generally anywhere from a few inches to less than six feet.
- High-frequency RFID system These range from 3 MHz to 30 MHz, with the typical HF frequency being 13.56 MHz. The standard range is anywhere from a few inches to several feet.
- UHF RFID systems. These range from 300 MHz to 960 MHz, with the typical frequency of 433 MHz and generally can be read from 25-plus feet away.
- Microwave RFID systems. These run at 2.45 GHz can be read from 30-plus feet away.

The frequency used will depend on the RFID application, with actual obtained distances sometimes varying from what is expected.

RFID frequencies and ranges		
FREQUENCY	BAND	RANGE
LF RFID	30-500 KHz, typically 125 KHz	Less than three feet
HF RFID	3-30 MHz, typically 13.56 MHz	Less than six feet
UHF RFID	300-960 MHz, typically 433 MHz	25+ feet
Microwave	2.45 GHz	30+ feet

4. Choice of components

1.4.1 Project components

There are some module that we will use in our mini project: MFRC522 module, MIFARE card (MF1S50) and some other things like buzzer, LEDs and the main things is microcontroller which is ATMEGA328P to handle the middle process: the transmission and receiving data process between reader and tag in our RFID system.

1.4.2 Components reasoning

- Easy to buy in electronics shop.
- Cheap.
- Already built device as module. Just only find the way to communicate with each device.
- Small distance application (13.56MHz for standard communication), low power (Passive tag) and limit data transfer (106 Kbit data in each second).

II. MFRC522 AND MIFARE CARD IMPLEMENTATION IN RFID SYSTEM

2.1. MFRC522 chip



The MFRC522 is a highly integrated reader/writer IC for contactless communication at 13.56 MHz. The MFRC522 reader supports ISO/IEC 14443 A/MIFARE mode.

The MFRC522's internal transmitter is able to drive a reader/writer antenna designed to communicate with ISO/IEC 14443 A/MIFARE cards and transponders without additional active circuitry. The receiver module provides a robust and efficient implementation for demodulating and decoding signals from ISO/IEC 14443 A/MIFARE compatible cards and transponders. The digital module manages the complete ISO/IEC 14443 A framing and error detection (parity and CRC) functionality.

The MFRC522 supports MF1xxS20, MF1xxS70 and MF1xxS50 products. The MFRC522 supports contactless communication and uses MIFARE higher transfer speeds up to 848 kBd in both directions.

The following host interfaces are provided:

- Serial Peripheral Interface (SPI)
- Serial UART (similar to RS232 with voltage levels dependent on pin voltage supply).
- I2C-bus interface.

2.1.1. Feature and benefits

- Highly integrated analog circuitry to demodulate and decode responses
- Buffered output drivers for connecting an antenna with the minimum number of external components
- Supports ISO/IEC 14443 A/MIFARE
- Typical operating distance in Read/Write mode up to 50 mm depending on the antenna size and tuning
- Supports MF1xxS20, MF1xxS70 and MF1xxS50 encryption in Read/Write mode
- Supports ISO/IEC 14443 A higher transfer speed communication up to 848 kBd
- Supports MFIN/MFOUT

- Additional internal power supply to the smart card IC connected via MFIN/MFOUT

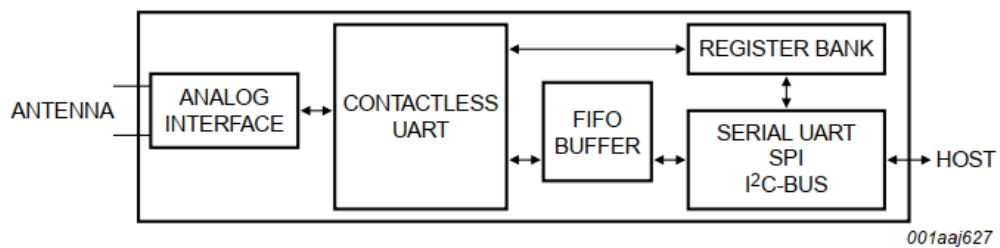
- Supported host interfaces
- SPI up to 10 Mbit/s(fastest)
- I2C-bus interface up to 400 kBd in Fast mode, up to 3400 kBd in High-speed mode
- RS232 Serial UART up to 1228.8 kBd, with voltage levels dependant on pin
- voltage supply
- FIFO buffer handles 64 byte send and receive
- Flexible interrupt modes
- Hard reset with low power function
- Power-down by software mode
- Programmable timer
- Internal oscillator for connection to 27.12 MHz quartz crystal
- 2.5 V to 3.3 V power supply
- CRC coprocessor
- Programmable I/O pins
- Internal self-test

2.1.2. Block diagram

The analog interface handles the modulation and demodulation of the analog signals.

The contactless UART manages the protocol requirements for the communication protocols in cooperation with the host. The FIFO buffer ensures fast and convenient data transfer to and from the host and the contactless UART and vice versa.

Various host interfaces are implemented to meet different customer requirements.



Detail block diagram:

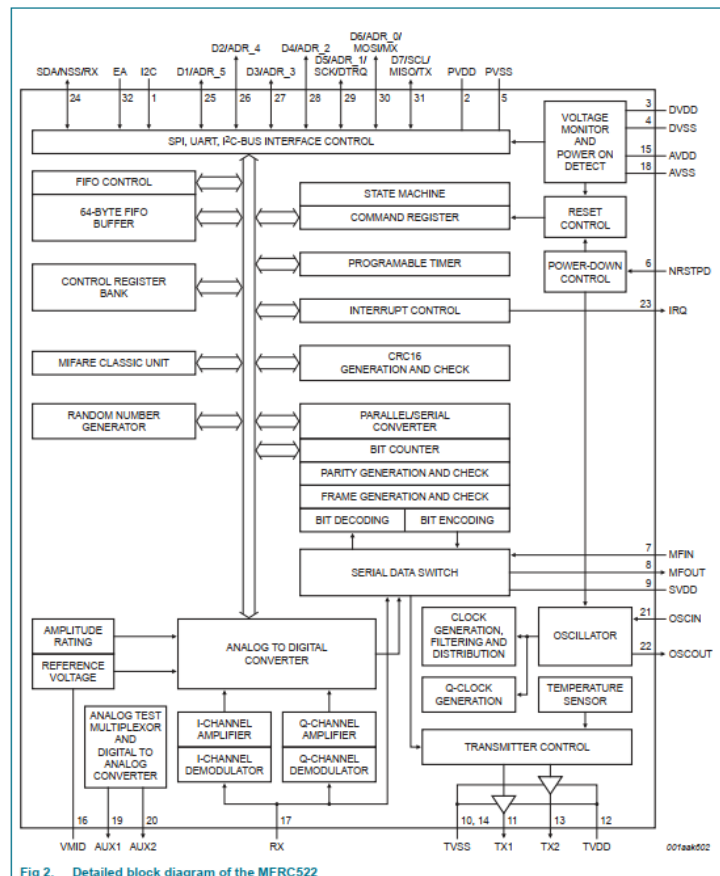


Fig 2. Detailed block diagram of the MFRC522

2.1.3. Functional description

The MFRC522 transmission module supports the Read/Write mode for ISO/IEC 14443 A/MIFARE using various transfer speeds and modulation protocols.

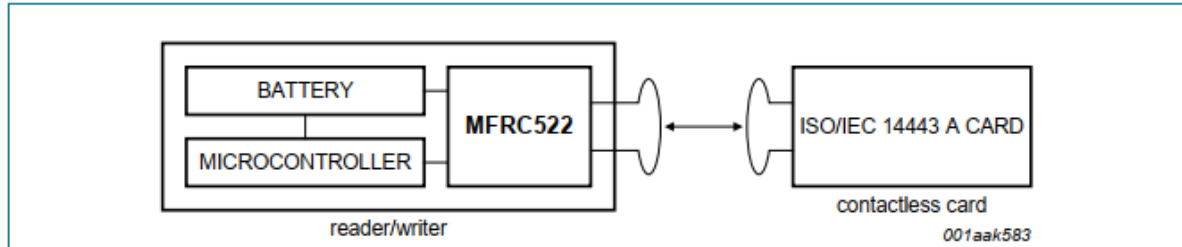
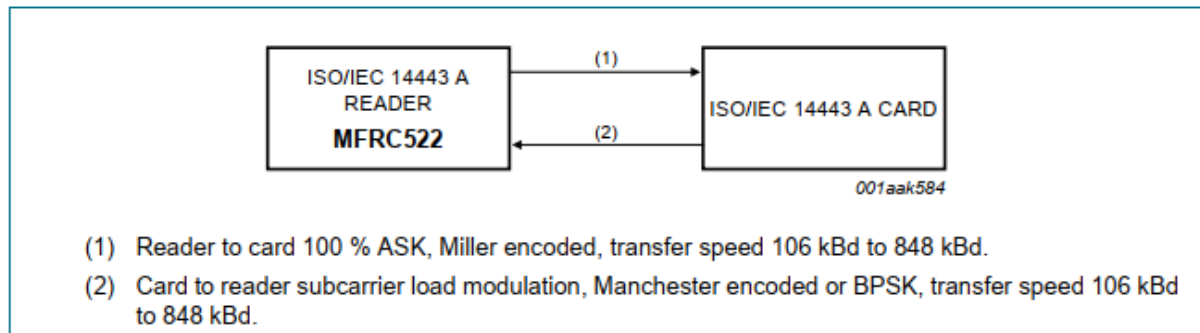
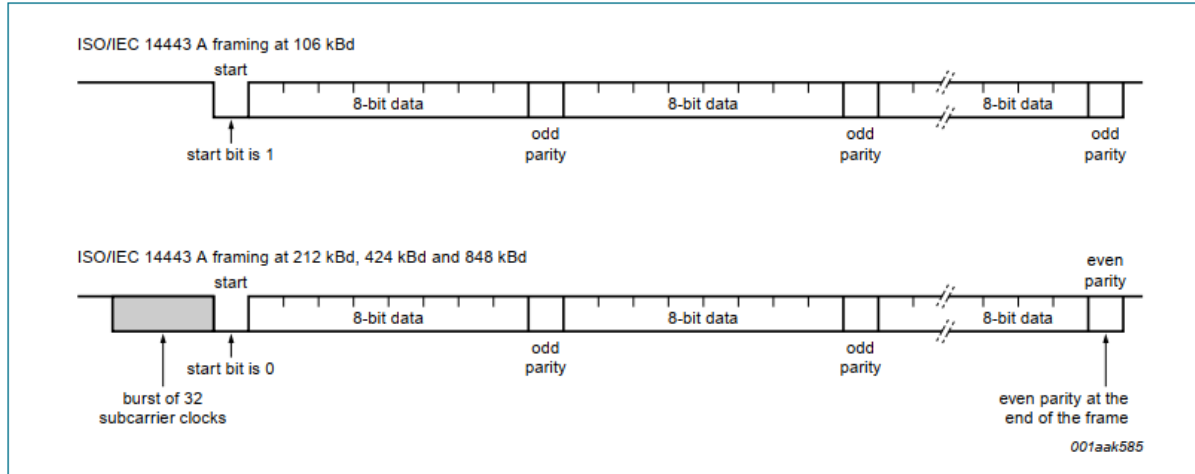


Table 4. Communication overview for ISO/IEC 14443 A/MIFARE reader/writer

Communication direction	Signal type	Transfer speed			
		106 kBd	212 kBd	424 kBd	848 kBd
Reader to card (send data from the MFRC522 to a card)	reader side modulation	100 % ASK	100 % ASK	100 % ASK	100 % ASK
	bit encoding	modified Miller encoding	modified Miller encoding	modified Miller encoding	modified Miller encoding
	bit length	128 (13.56 μ s)	64 (13.56 μ s)	32 (13.56 μ s)	16 (13.56 μ s)
Card to reader (MFRC522 receives data from a card)	card side modulation	subcarrier load modulation	subcarrier load modulation	subcarrier load modulation	subcarrier load modulation
	subcarrier frequency	13.56 MHz / 16	13.56 MHz / 16	13.56 MHz / 16	13.56 MHz / 16
	bit encoding	Manchester encoding	BPSK	BPSK	BPSK



Data coding and framing according to ISO/IEC 14443 A using 106 KBd (which is used in our project) and some other communication at different rate. Automatic parity can be switched off using parity disable's register.



2.1.4. Digital interface

In this project, we only use SPI protocol to interface with MFRC522 module so that we only discuss about this protocol.

2.1.4.1. Serial peripheral interface

A serial peripheral interface (SPI compatible) is supported to enable high-speed communication to the host. The interface can handle data speeds up to 10 Mbit/s. When communicating with a host, the MFRC522 acts as a slave, receiving data from the external host for register settings, sending and receiving data relevant for RF interface communication.

An interface compatible with SPI enables high-speed serial communication between the MFRC522 and a microcontroller. The implemented interface is in accordance with the SPI standard.

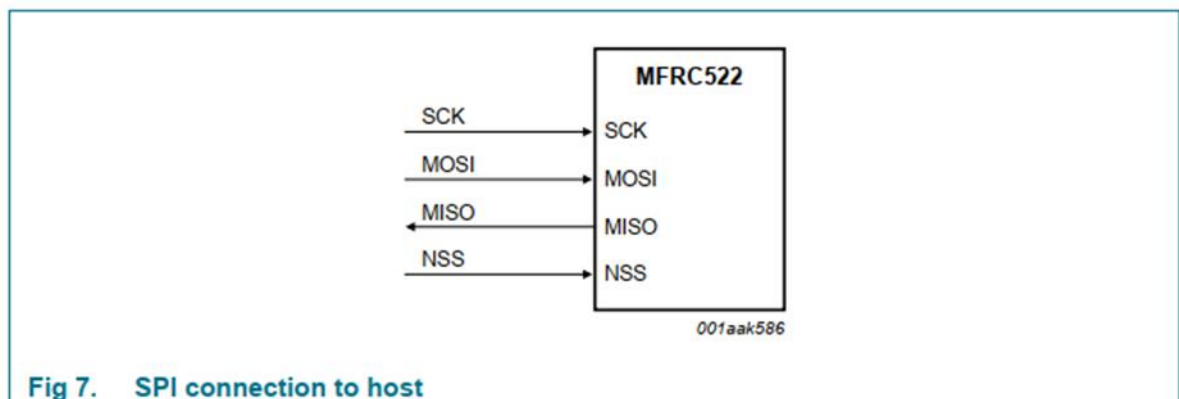


Fig 7. SPI connection to host

The MFRC522 acts as a slave during SPI communication. The SPI clock signal SCK must be generated by the master. Data communication from the master to the slave uses the MOSI line. The MISO line is used to send data from the MFRC522 to the master.

Data bytes on both MOSI and MISO lines are sent with the MSB first. Data on both MOSI and MISO lines must be stable on the rising edge of the clock and can be changed on the falling edge. Data is provided by the MFRC522 on the falling clock edge and is stable during the rising clock edge => (CPOL = 0 and CPHA = 0).

2.1.4.2. SPI read data

It is possible to read out up to n-data bytes.

The first byte sent defines both the mode and the address byte.

Line	Byte 0	Byte 1	Byte 2	To	Byte n	Byte n + 1
MOSI	address 0	address 1	address 2	...	address n	00
MISO	X ^[1]	data 0	data 1	...	data n – 1	data n

2.1.4.3. SPI write data

It is possible to write up to n data bytes by only sending one address byte.

The first byte sent defines both the mode and the address byte.

Line	Byte 0	Byte 1	Byte 2	To	Byte n	Byte n + 1
MOSI	address 0	data 0	data 1	...	data n – 1	data n
MISO	X ^[1]	X ^[1]	X ^[1]	...	X ^[1]	X ^[1]

2.1.4.4. SPI address byte

The address byte must meet the following format.

The MSB of the first byte defines the mode used. To read data from the MFRC522 the

MSB is set to logic 1. To write data to the MFRC522 the MSB must be set to logic 0. Bits 6 to 1 define the address and the LSB is set to logic 0.

7 (MSB)	6	5	4	3	2	1	0 (LSB)
1 = read 0 = write	address						0

2.1.5. Analog interface and contactless UART

2.1.5.1. General

The integrated contactless UART supports the external host online with framing and error checking of the protocol requirements up to 848 kBd. An external circuit can be connected to the communication interface pins MFIN and MFOUT to modulate and demodulate the data.

The contactless UART handles the protocol requirements for the communication protocols in cooperation with the host. Protocol handling generates bit and byte-oriented framing. In addition, it handles error detection such as parity and CRC, based on the various supported contactless communication protocols.

Remark: The size and tuning of the antenna and the power supply voltage have an important impact on the achievable operating distance.

2.1.5.2. TX p-driver

The signal on pins TX1 and TX2 is the 13.56 MHz energy carrier modulated by an envelope signal. It can be used to drive an antenna directly using a few passive components for matching and filtering. The signal on pins TX1 and TX2 can be configured using the TxControlReg register.

The modulation index can be set by adjusting the impedance of the drivers. The impedance of the p-driver can be configured using registers CWGsPReg and

ModGsPReg. The impedance of the n-driver can be configured using the GsNReg register. The modulation index also depends on the antenna design and tuning.

The TxModeReg and TxSelReg registers control the data rate and framing during transmission and the antenna driver setting to support the different requirements at the different modes and transfer speeds.

Table 15. Register and bit settings controlling the signal on pin TX1

Bit Tx1RFEn	Bit Force 100ASK	Bit InvTx1RFOOn	Bit InvTx1RFOff	Envelope	Pin TX1	GSPMos	GSNMos	Remarks
0	X ^[1]	X ^[1]	X ^[1]	X ^[1]	X ^[1]	X ^[1]	X ^[1]	not specified if RF is switched off
1	0	0	X ^[1]	0	RF	pMod	nMod	100 % ASK: pin TX1 pulled to logic 0, independent of the InvTx1RFOff bit
				1	RF	pCW	nCW	
	0	1	X ^[1]	0	RF	pMod	nMod	
				1	RF	pCW	nCW	
	1	1	X ^[1]	0	0	pMod	nMod	
				1	RF_n	pCW	nCW	

Table 16. Register and bit settings controlling the signal on pin TX2

Bit Tx1RFEn	Bit Force 100ASK	Bit Tx2CW	Bit InvTx2RFOOn	Bit InvTx2RFOff	Envelope	Pin TX2	GSPMos	GSNMos	Remarks
0	X ^[1]	X ^[1]	X ^[1]	X ^[1]	X ^[1]	X ^[1]	X ^[1]	X ^[1]	not specified if RF is switched off
1	0	0	0	X ^[1]	0	RF	pMod	nMod	-
				X ^[1]	1	RF	pCW	nCW	
			1	X ^[1]	0	RF_n	pMod	nMod	
				X ^[1]	1	RF_n	pCW	nCW	
		1	0	X ^[1]	X ^[1]	RF	pCW	nCW	conductance always CW for the Tx2CW bit
			1	X ^[1]	X ^[1]	RF_n	pCW	nCW	
	1	0	0	X ^[1]	0	0	pMod	nMod	100 % ASK: pin TX2 pulled to logic 0 (independent of the InvTx2RFOOn/InvTx2RFOff bits)
				X ^[1]	1	RF	pCW	nCW	
			1	X ^[1]	0	0	pMod	nMod	
				X ^[1]	1	RF_n	pCW	nCW	
		1	0	X ^[1]	X ^[1]	RF	pCW	nCW	
			1	X ^[1]	X ^[1]	RF_n	pCW	nCW	

The following abbreviations have been used in [Table 15](#) and [Table 16](#):

- RF: 13.56 MHz clock derived from 27.12 MHz quartz crystal oscillator divided by 2
- RF_n: inverted 13.56 MHz clock
- GSPMos: conductance, configuration of the PMOS array
- GSNMos: conductance, configuration of the NMOS array
- pCW: PMOS conductance value for continuous wave defined by the CWGsPReg register
- pMod: PMOS conductance value for modulation defined by the ModGsPReg register
- nCW: NMOS conductance value for continuous wave defined by the GsNReg register's CWGsN[3:0] bits
- nMod: NMOS conductance value for modulation defined by the GsNReg register's ModGsN[3:0] bits
- X = do not care.

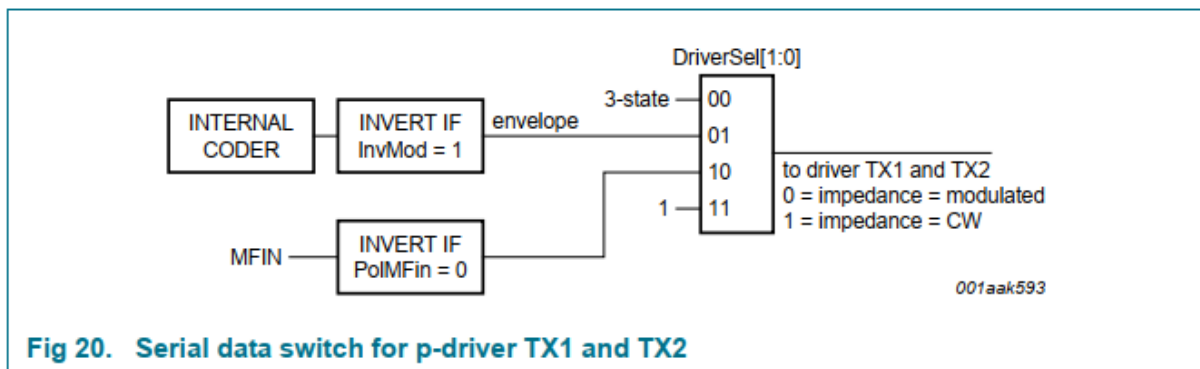
Remark: If only one driver is switched on, the values for CWGsPReg, ModGsPReg and GsNReg registers are used for both drivers.

2.1.5.3. Serial data switch

Two main blocks are implemented in the MFRC522. The digital block comprises the state machines, encoder/decoder logic. The analog block comprises the modulator and antenna drivers, the receiver and amplifiers. It is possible for the interface between these two blocks to be configured so that the interfacing signals are routed to pins MFIN and MFOUT.

This topology allows the analog block of the MFRC522 to be connected to the digital block of another device.

The serial signal switch is controlled by the TxSelReg and RxSelReg registers.



2.1.5.4. MFIN and MFOUT interface support

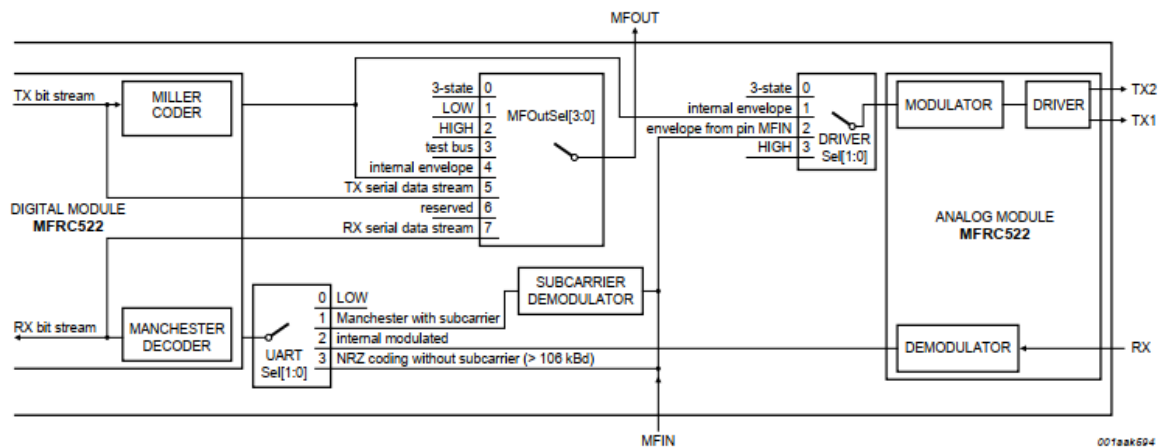
Switch MFOutSel in the TxSelReg register can be used to measure MIFARE and ISO/IEC14443 A related signals. This is especially important during the design-in phase or for test purposes as it enables checking of the transmitted and received data.

The most important use of pins MFIN and MFOUT is found in the active antenna concept.

An external active antenna circuit can be connected to the MFRC522's digital block.

Switch MFOutSel must be configured so that the internal Miller encoded signal is sent to pin MFOUT (MFOutSel = 100b). UARTSel[1:0] must be configured to receive a Manchester signal with subcarrier from pin MFIN (UARTSel[1:0] = 01).

It is possible to connect a passive antenna to pins TX1, TX2 and RX (using the appropriate filter and matching circuit) and an active antenna to pins MFOUT and MFIN at the same time. In this configuration, two RF circuits can be driven (one after another) by a single host processor.



2.1.6. FIFO buffer

An 8 x 64 bit FIFO buffer is used in the MFRC522. It buffers the input and output data stream between the host and the MFRC522's internal state machine. This makes it

possible to manage data streams up to 64 bytes long without the need to take timing constraints into account

2.1.6.1. Accessing the FIFO buffer

The FIFO buffer input and output data bus is connected to the FIFODataReg register.

Writing to this register stores one byte in the FIFO buffer and increments the internal FIFO buffer write pointer. Reading from this register shows the FIFO buffer contents pointed by the FIFO buffer read pointer and decrements the FIFO buffer read pointer. The distance between the write and read pointer can be obtained by reading the FIFOLevelReg register.

When the microcontroller starts a command, the MFRC522 can, while the command is in progress, access the FIFO buffer according to that command. Only one FIFO buffer has been implemented which can be used for input and output. The microcontroller must ensure that there are not any unintentional FIFO buffer accesses.

2.1.6.2. FIFO buffer status information

The host can get the following FIFO buffer status information:

- Number of bytes stored in the FIFO buffer: read FIFOLevel 's register.
- FIFO almost full warning: Status1Reg register's HiAlert bit (based on waterlevel 's register).
- FIFO buffer almost empty warning: Status1Reg register's LoAlert bit (based on waterlevel's register).
- FIFO buffer overflow warning: ErrorReg register's BufferOvfl bit. The BufferOvfl bit can only be cleared by setting the FIFOLevelReg register's FlushBuffer bit.

$$HiAlert = (64 - FIFOLength) \leq WaterLevel$$

$$LoAlert = FIFOLength \leq WaterLevel$$

2.1.7. Interrupt request system

The MFRC522 indicates certain events by setting the Status1Reg register's IRq bit and, if activated, by pin IRQ. The signal on pin IRQ can be used to interrupt the host using its interrupt handling capabilities. This allows the implementation of efficient host software.

Table 18. Interrupt sources

Interrupt flag	Interrupt source	Trigger action
IRq	timer unit	the timer counts from 1 to 0
TxIRq	transmitter	a transmitted data stream ends
CRCIRq	CRC coprocessor	all data from the FIFO buffer has been processed
RxIRq	receiver	a received data stream ends
IdleIRq	ComIrqReg register	command execution finishes
HiAlertIRq	FIFO buffer	the FIFO buffer is almost full
LoAlertIRq	FIFO buffer	the FIFO buffer is almost empty
ErrIRq	contactless UART	an error is detected

2.1.8. Timer unit

The timer unit can be used to measure the time interval between two events or to indicate that a specific event occurred after a specific time. The timer can be triggered by events explained in the paragraphs below. The timer does not influence any internal events, for example, a time-out during data reception does not automatically influence the reception process. Furthermore, several timer-related bits can be used to generate an interrupt.

The timer has an input clock of 13.56 MHz derived from the 27.12 MHz quartz crystal oscillator. The timer consists of two stages: prescaler and counter.

The prescaler (TPrescaler) is a 12-bit counter. The reload values (TReloadVal_Hi[7:0] and TReloadVal_Lo[7:0]) for TPrescaler can be set between 0 and 4095 in the TModeReg register's TPrescaler_Hi[3:0] bits and TPrescalerReg register's TPrescaler_Lo[7:0] bits.

The reload value for the counter is defined by 16 bits between 0 and 65535 in the TReloadReg register.

The current value of the timer is indicated in the TCounterValReg register.

When the counter reaches 0, an interrupt is automatically generated, indicated by the ComIrqReg register's TimerIRq bit setting. If enabled, this event can be indicated on pin IRQ. The TimerIRq bit can be set and reset by the host. Depending on the configuration, the timer will stop at 0 or restart with the value set in the TreloadReg register.

The timer status is indicated by the Status1Reg register's TRunning bit

The timer can be started manually using the ControlReg register's TStartNow bit and stopped using the ControlReg register's TStopNow bit.

The timer can also be activated automatically to meet any dedicated protocol requirements by setting the TModeReg register's TAuto bit to logic 1.

The delay time of a timer stage is set by the reload value + 1. The total delay time (t_{d1}) is calculated using [Equation 5](#):

$$t_{d1} = \frac{(TPrescaler \times 2 + 1) \times (TReloadVal + 1)}{13.56 \text{ MHz}} \quad (5)$$

2.1.9. Power reduction mode (please read reference)

2.1.10. Timer unit (please read reference)

2.2. MIFARE card (MF1S50)

NXP Semiconductors has developed the MIFARE Classic EV1 contactless IC MF1S50yyX/V1 to be used in a contactless smart card according to ISO/IEC 14443 Type A.

The MIFARE Classic EV1 with 1K memory MF1S50yyX/V1 IC is used in applications like public transport ticketing and can also be used for various other applications

The MFRC522 indicates certain events by setting the Status1Reg register's IRq bit and, if activated, by pin IRQ. The signal on pin IRQ can be used to interrupt the host

using its interrupt handling capabilities. This allows the implementation of efficient host software.

2.2. MIFARE card (MF1S50)

NXP Semiconductors has developed the MIFARE Classic EV1 contactless IC MF1S50yyX/V1 to be used in a contactless smart card according to ISO/IEC 14443 Type A.

The MIFARE Classic EV1 with 1K memory MF1S50yyX/V1 IC is used in applications like public transport ticketing and can also be used for various other applications:

2.2.1. Block diagram

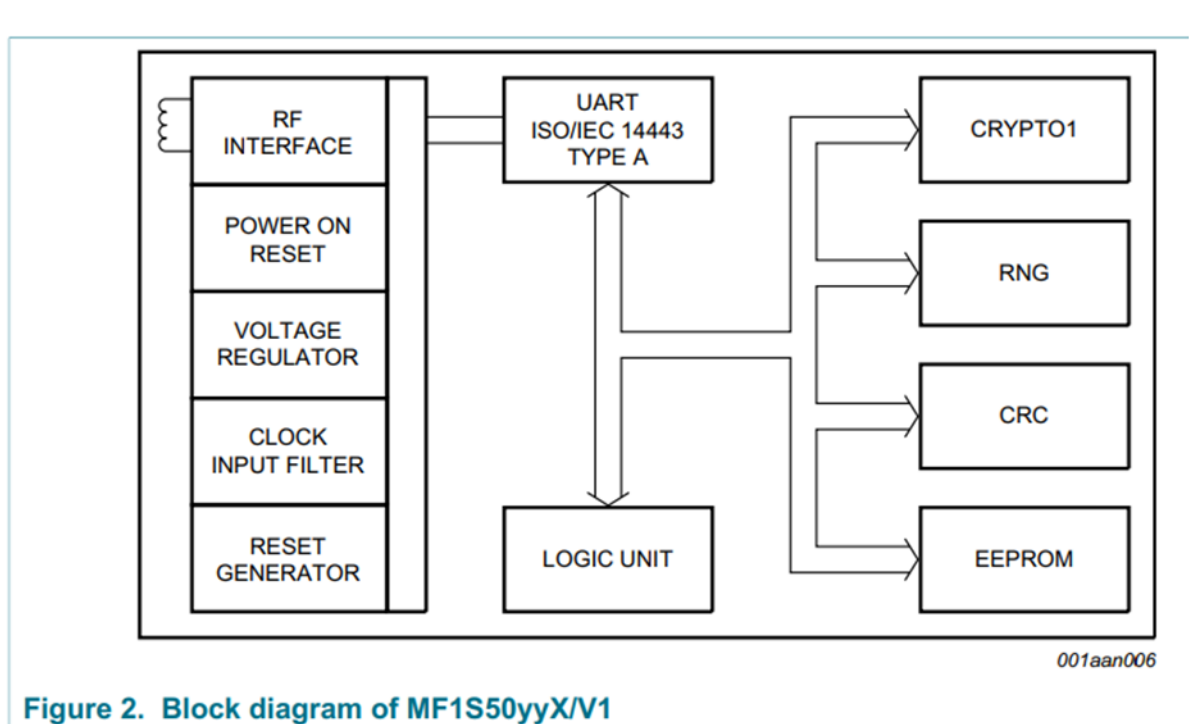


Figure 2. Block diagram of MF1S50yyX/V1

2.2.1.1. Block description

The MF1S50yyX/V1 chip consists of a 1 kB EEPROM, RF interface and Digital Control

Unit. Energy and data are transferred via an antenna consisting of a coil with a small number of turns which is directly connected to the MF1S50yyX/V1. No further external components are necessary.

- RF interface:
 - Modulator/demodulator
 - Rectifier
 - Clock regenerator
 - Power-On Reset (POR)
 - Voltage regulator
- Anticollision: Multiple cards in the field may be selected and managed in sequence
- Authentication: Preceding any memory operation the authentication procedure ensures that access to a block is only possible via the two keys specified for each block.
- Control and Arithmetic Logic Unit: Values are stored in a special redundant format and can be incremented and decremented
- EEPROM interface
- Crypto unit: The CRYPTO1 stream cipher of the MF1S50yyX/V1 is used for authentication and encryption of data exchange.
- EEPROM: 1 kB is organized in 16 sectors of 4 blocks. One block contains 16 bytes. The last block of each sector is called "trailer", which contains two secret keys and programmable access conditions for each block in this sector.

2.2.2. Communication principle

The commands are initiated by the reader and controlled by the Digital Control Unit of the MF1S50yyX/V1. The command response is depending on the state of the IC

and for memory operations also on the access conditions valid for the corresponding sector.

2.2.2.1. Request standard/all

After Power-On Reset (POR) the card answers to a request REQA or wakeup WUPA command with the answer to request code

2.2.2.2. Anticollision loop

In the anticollision loop the identifier of a card is read. If there are several cards in the operating field of the reader, they can be distinguished by their identifier and one can be selected (select card) for further transactions. The unselected cards return to the idle state and wait for a new request command. If the 7-byte UID is used for anticollision and selection.

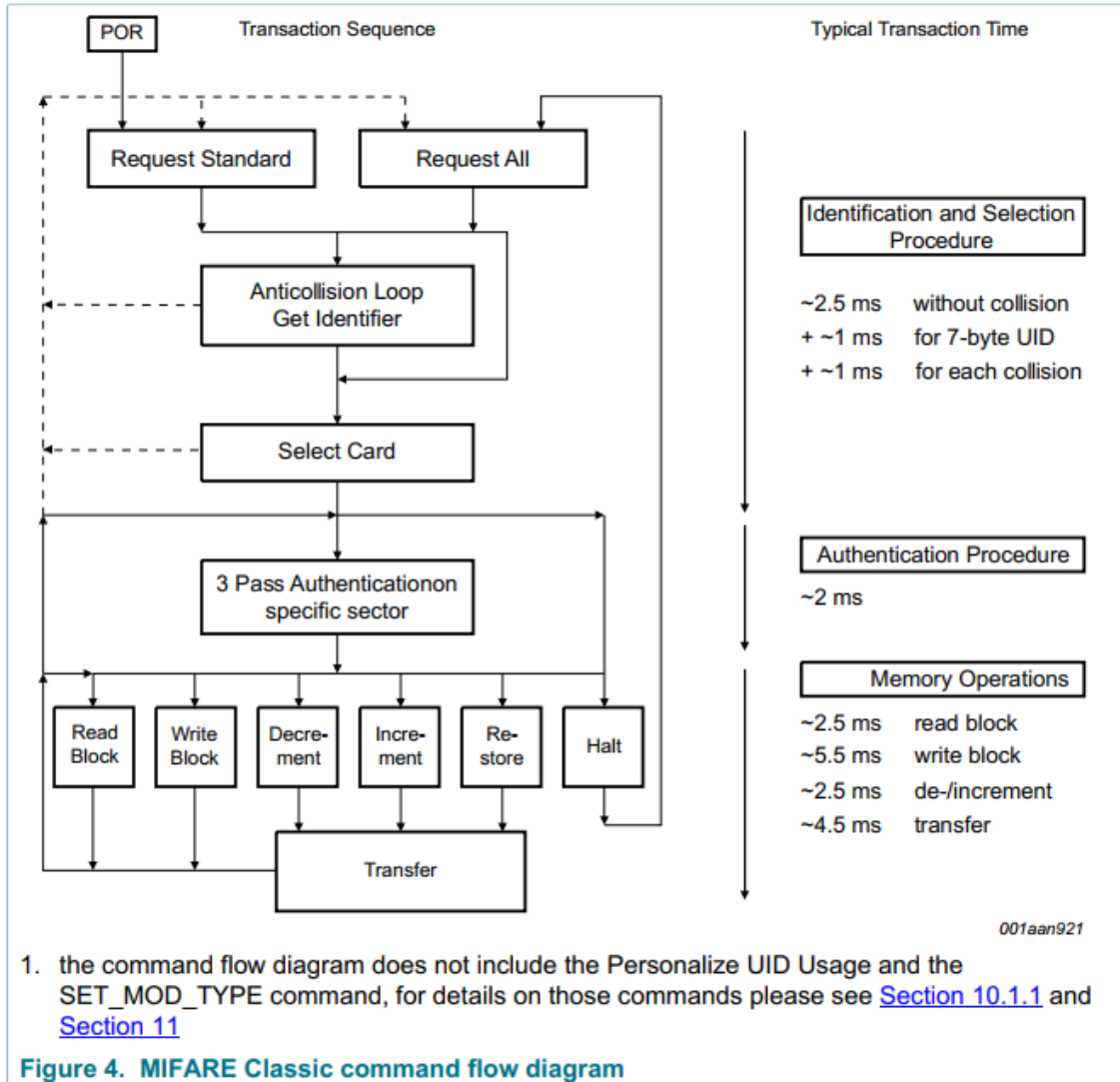
Remark: For the 4-byte non-unique ID product versions, the identifier retrieved from the card is not defined to be unique

2.2.2.3. Select card

With the select card command the reader selects one individual card for authentication and memory related operations. The card returns the Select AcKnowledge (SAK) code which determines the type of the selected card.

2.2.2.4. Three pass authentication

After selection of a card the reader specifies the memory location of the following memory access and uses the corresponding key for the three pass authentication procedure. After a successful authentication all commands and responses are encrypted.



2.2.3. Memory operations

After authentication any of the following operations may be performed:

- Read block
- Write block
- Decrement: Decrements the contents of a block and stores the result in the internal Transfer Buffer
- Increment: Increments the contents of a block and stores the result in the internal transfer Buffer
- Restore: Moves the contents of a block into the internal Transfer Buffer

- Transfer: Writes the contents of the internal Transfer Buffer to a value block

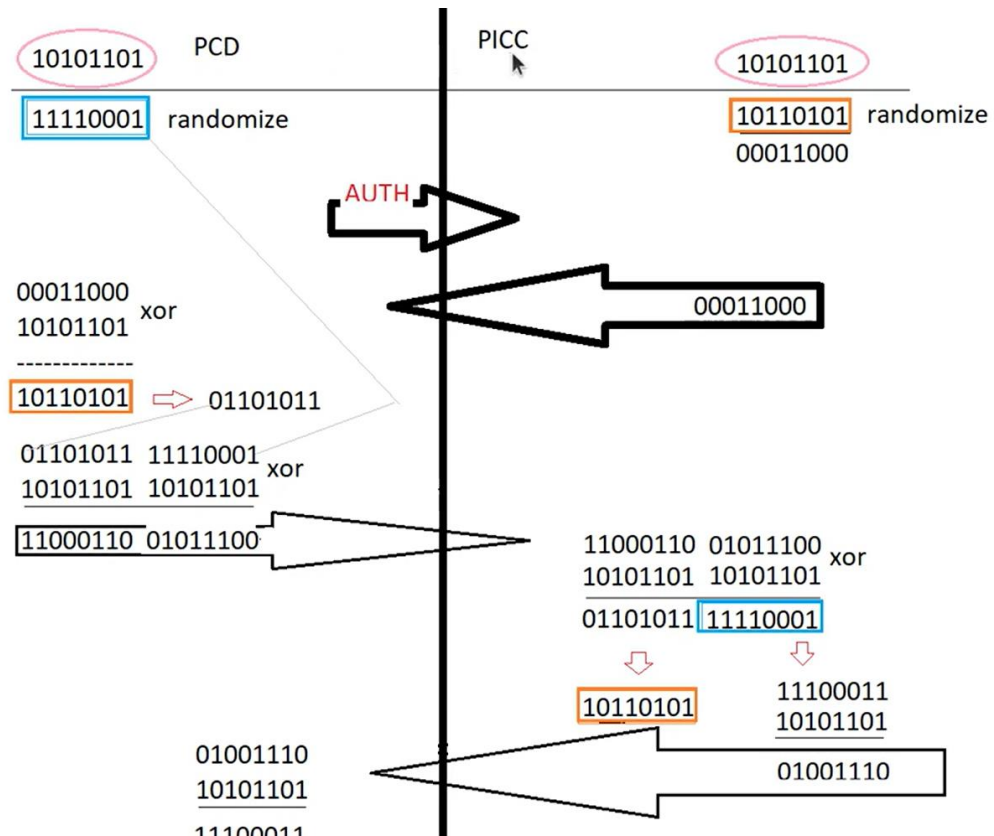
2.2.4. Data integrity

Following mechanisms are implemented in the contactless communication link between reader and card to ensure very reliable data transmission:

- 16 bits CRC per block
- Parity bits for each byte
- Bit count checking
- Bit coding to distinguish between "1", "0" and "no information"
- Channel monitoring (protocol sequence and bit stream analysis)

2.2.5. Three pass authentication sequence

1. The reader specifies the sector to be accessed and chooses key A or B.
2. The card reads the secret key and the access conditions from the sector trailer. Then the card sends a number as the challenge to the reader (pass one).
3. The reader calculates the response using the secret key and additional input. The response, together with a random challenge from the reader, is then transmitted to the card (pass two).
4. The card verifies the response of the reader by comparing it with its own challenge and then it calculates the response to the challenge and transmits it (pass three).
5. The reader verifies the response of the card by comparing it to its own challenge. After transmission of the first random challenge the communication between card and reader is encrypted.



2.2.6. RF interface

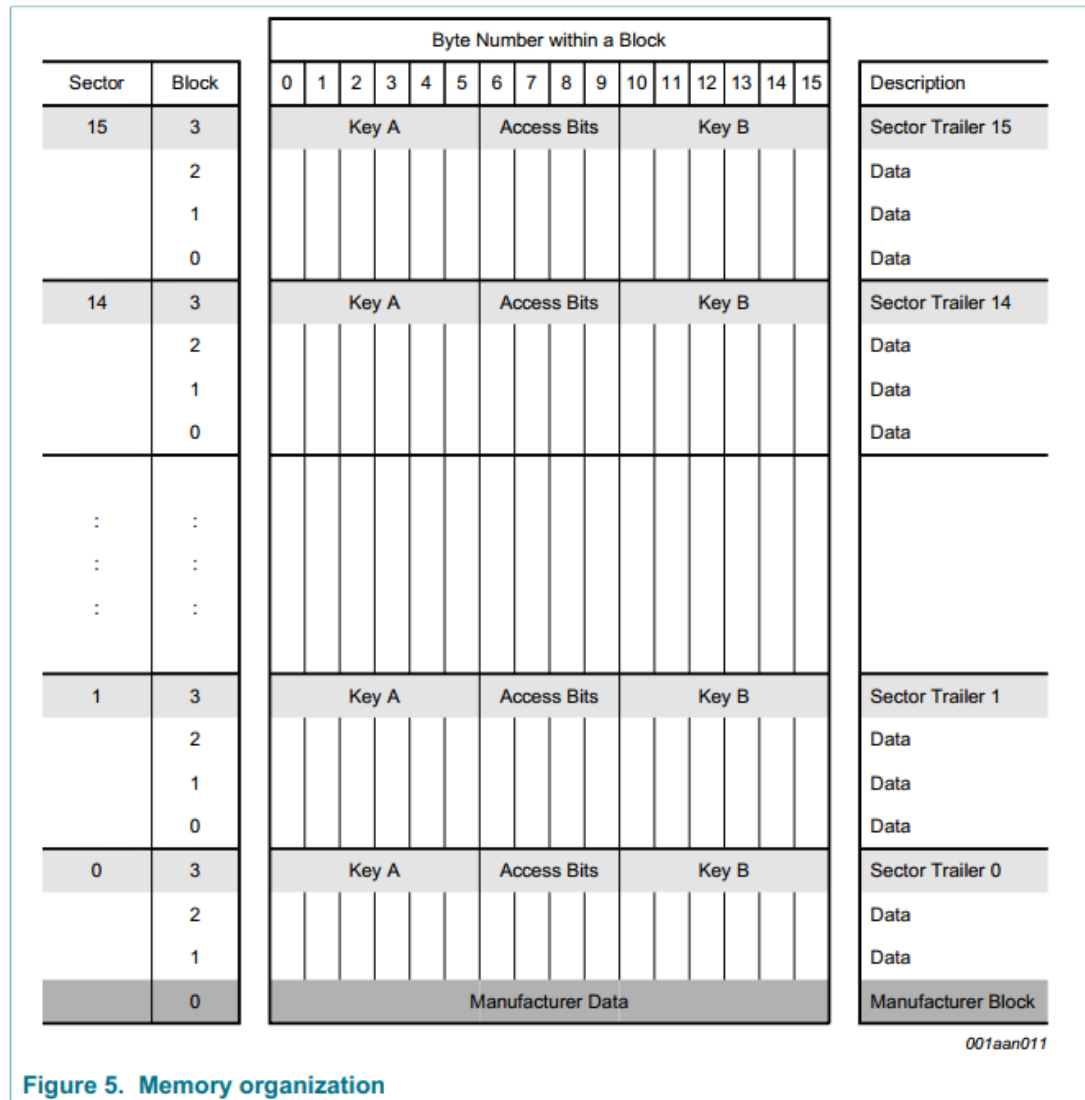
The RF-interface is according to the standard for contactless smart cards ISO/IEC 14443A.

For operation, the carrier field from the reader always needs to be present (with short pauses when transmitting), as it is used for the power supply of the card.

For both directions of data communication there is only one start bit at the beginning of each frame. Each byte is transmitted with a parity bit (odd parity) at the end. The LSB of the byte with the lowest address of the selected block is transmitted first. The maximum frame length is 163 bits (16 data bytes + 2 CRC bytes = $16 \times 9 + 2 \times 9 + 1$ start bit).

2.2.7. Memory organization

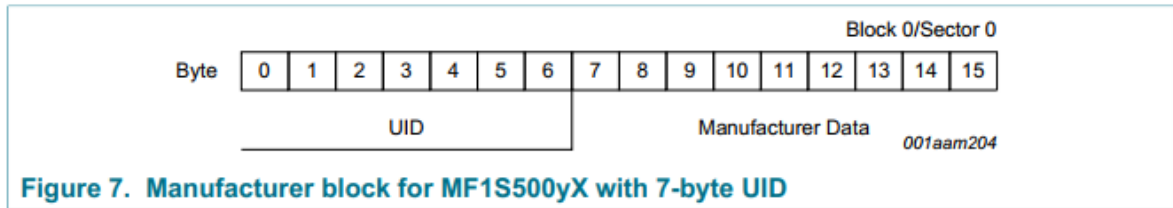
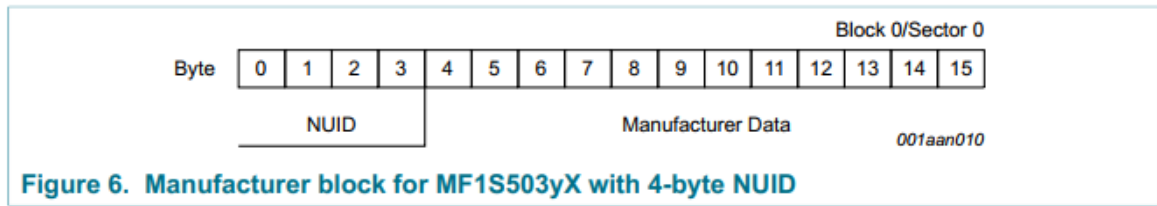
The 1024 × 8 bit EEPROM memory is organized in 16 sectors of 4 blocks. One block contains 16 bytes.



2.2.7.1. Manufacturer block

This is the first data block (block 0) of the first sector (sector 0). It contains the IC manufacturer data. This block is programmed and write protected in the production test.

The manufacturer block is shown in 2 figure below for the 4-byte NUID and 7-byte UID version respectively.



2.2.7.2. Data block

All sectors contain 3 blocks of 16 bytes for storing data (Sector 0 contains only two data blocks and the read-only manufacturer block).

The data blocks can be configured by the access bits as:

- read/write blocks
- value blocks

Value blocks can be used for e.g. electronic purse applications, where additional commands like increment and decrement for direct control of the stored value are provided a successful authentication has to be performed to allow any memory operation.

Remark: The default content of the data blocks at delivery is not defined.

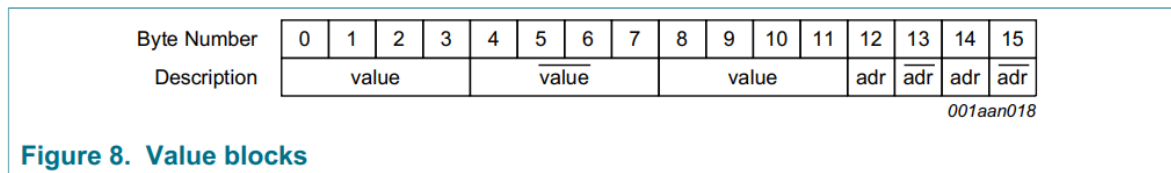
2.2.7.3. Value block

Value blocks allow performing electronic purse functions (valid commands are: read, write, increment, decrement, restore, transfer). Value blocks have a fixed data format which permits error detection and correction and a backup management.

A value block can only be generated through a write operation in value block format:

- Value: Signifies a signed 4-byte value. The lowest significant byte of a value is stored in the lowest address byte. Negative values are stored in standard 2's complement format. For reasons of data integrity and security, a value is stored three times, twice non-inverted and once inverted.

- Adr: Signifies a 1-byte address, which can be used to save the storage address of a block, when implementing a powerful backup management. The address byte is stored four times, twice inverted and non-inverted. During increment, decrement, restore and transfer operations the address remains unchanged. It can only be altered via a write command.



An example of a valid value block format for the decimal value 1234567d and the block address 17d:

Byte Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Description	value				value				value				adr	adr	adr	adr
Values [hex]	87	D6	12	00	78	29	ED	FF	87	D6	12	00	11	EE	11	EE

2.2.7.4. Sector trailer

The sector trailer is the last block (block 3) in one sector. Each sector has a sector trailer containing the

- secret keys A (mandatory) and B (optional), which return logical "0"s when read and
- the access conditions for the blocks of that sector, which are stored in bytes 6...9.

The access bits also specify the type (data or value) of the data blocks.

If key B is not needed, the last 6 bytes of the sector trailer can be used as data bytes. The access bits for the sector trailer have to be configured accordingly.

Byte 9 of the sector trailer is available for user data. For this byte the same access rights as for byte 6, 7 and 8 apply.

When the sector trailer is read, the key bytes are blanked out by returning logical zeros.

If key B is configured to be readable, the data stored in bytes 10 to 15 is returned.

All keys are set to FFFF FFFF FFFFh at chip delivery and the bytes 6, 7 and 8 are set to FF0780h.

Byte Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Description	Key A						Access Bits			Key B (optional)						

001aan013

Figure 9. Sector trailer

2.2.8. Memory access

Before any memory operation can be done, the card has to be selected and authenticated as described in Section 2.2.5. The possible memory operations for an addressed block depend on the key used during authentication and the access conditions stored in the associated sector trailer.

Table 5. Memory operations

Operation	Description	Valid for Block Type
Read	reads one memory block	read/write, value and sector trailer
Write	writes one memory block	read/write, value and sector trailer
Increment	increments the contents of a block and stores the result in the internal Transfer Buffer	value
Decrement	decrements the contents of a block and stores the result in the internal Transfer Buffer	value
Transfer	writes the contents of the internal Transfer Buffer to a block	value and read/write
Restore	reads the contents of a block into the internal Transfer Buffer	value

2.2.8.1. Access conditions

The access conditions for every data block and sector trailer are defined by 3 bits, which are stored non-inverted and inverted in the sector trailer of the specified sector.

The access bits control the rights of memory access using the secret keys A and B. The access conditions may be altered, provided one knows the relevant key and the current access condition allows this operation.

Remark: With each memory access the internal logic verifies the format of the access conditions. If it detects a format violation the whole sector is irreversibly blocked.

Remark: In the following description the access bits are mentioned in the non-inverted mode only.

The internal logic of the MF1S50yyX/V1 ensures that the commands are executed only after a successful authentication.

Table 6. Access conditions

Access Bits	Valid Commands		Block	Description
C1 ₃ , C2 ₃ , C3 ₃	read, write	→	3	sector trailer
C1 ₂ , C2 ₂ , C3 ₂	read, write, increment, decrement, transfer, restore	→	2	data block
C1 ₁ , C2 ₁ , C3 ₁	read, write, increment, decrement, transfer, restore	→	1	data block
C1 ₀ , C2 ₀ , C3 ₀	read, write, increment, decrement, transfer, restore	→	0	data block

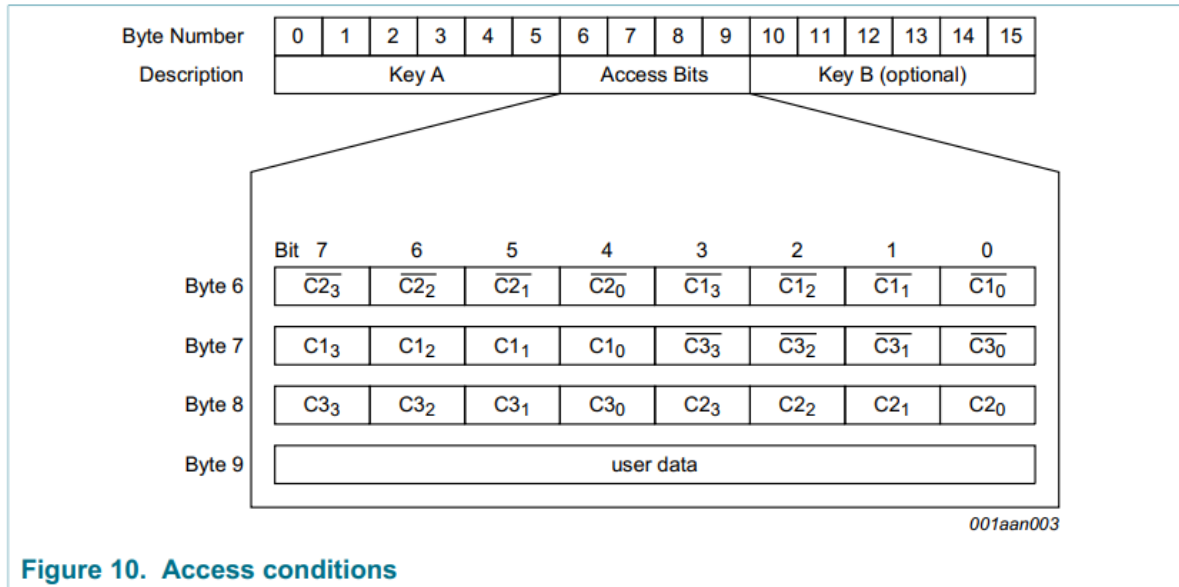


Figure 10. Access conditions

2.2.8.2. Access conditions for the sector trailer

Table 7. Access conditions for the sector trailer

Access bits			Access condition for						Remark
		KEYA			Access bits	KEYB			
C1	C2	C3	read	write	read	write	read	write	
0	0	0	never	key A	key A	never	key A	key A	Key B may be read ^[1]
0	1	0	never	never	key A	never	key A	never	Key B may be read ^[1]
1	0	0	never	key B	key A B	never	never	key B	
1	1	0	never	never	key A B	never	never	never	
0	0	1	never	key A	key A	key A	key A	key A	Key B may be read, transport configuration ^[1]
0	1	1	never	key B	key A B	key B	never	key B	
1	0	1	never	never	key A B	key B	never	never	
1	1	1	never	never	key A B	never	never	never	

Origin and no configuration: the value of byte 6 -> 9 of access condition for sector trailer is 001

Table 7. Access conditions for the sector trailer

Access bits			Access condition for						Remark
			KEYA		Access bits		KEYB		
C1	C2	C3	read	write	read	write	read	write	
0	0	0	never	key A	key A	never	key A	key A	Key B may be read ^[1]
0	1	0	never	never	key A	never	key A	never	Key B may be read ^[1]
1	0	0	never	key B	key A B	never	never	key B	
1	1	0	never	never	key A B	never	never	never	
0	0	1	never	key A	key A	key A	key A	key A	Key B may be read, transport configuration ^[1]
0	1	1	never	key B	key A B	key B	never	key B	
1	0	1	never	never	key A B	key B	never	never	
1	1	1	never	never	key A B	never	never	never	

2.2.8.3. Access condition for data block

Access bits			Access condition for				Application
C1	C2	C3	read	write	increment	decrement, transfer, restore	
0	0	0	key A B	key A B	key A B	key A B	transport configuration ^[1]

Access bits			Access condition for				Application
0	1	0	key A B	never	never	never	read/write block ^[1]
1	0	0	key A B	key B	never	never	read/write block ^[1]
1	1	0	key A B	key B	key B	key A B	value block ^[1]
0	0	1	key A B	never	never	key A B	value block ^[1]
0	1	1	key B	key B	never	never	read/write block ^[1]
1	0	1	key B	never	never	never	read/write block ^[1]
1	1	1	never	never	never	never	read/write block

2.2.9. Command overview

The MIFARE Classic card activation follows the ISO/IEC 14443 Type A. After the MIFARE Classic card has been selected, it can either be deactivated using the ISO/IEC 14443 Halt command, or the MIFARE Classic commands can be performed.

All MIFARE Classic commands typically use the MIFARE Classic using Crypto1 and require an authentication.

All available commands for the MIFARE Classic EV1 with 1K memory are shown in table below

Table 9. Command overview

Command	ISO/IEC 14443	Command code (hexadecimal)
Request	REQA	26h (7 bit)
Wake-up	WUPA	52h (7 bit)
Anticollision CL1	Anticollision CL1	93h 20h
Select CL1	Select CL1	93h 70h
Anticollision CL2	Anticollision CL2	95h 20h
Select CL2	Select CL2	95h 70h
Halt	Halt	50h 00h
Authentication with Key A	-	60h
Authentication with Key B	-	61h
Personalize UID Usage	-	40h
SET_MOD_TYPE	-	43h
MIFARE Read	-	30h

Command	ISO/IEC 14443	Command code (hexadecimal)
MIFARE Write	-	A0h
MIFARE Decrement	-	C0h
MIFARE Increment	-	C1h
MIFARE Restore	-	C2h
MIFARE Transfer	-	B0h

2.2.10. ATQA and SAK responses

The MF1S50yyX/V1 answers to a REQA or WUPA command with the ATQA value shown in Table 11 and to a Select CL1 command (CL2 for the 7-byte UID variant) with the SAK value shown in Table 12.

Table 11. ATQA response of the MF1S50yyX/V1

Sales Type	Hex Value	Bit Number															
		16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
MF1S500yX	00 44h	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0
MF1S503yX	00 04h	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
MF1S700yX	00 42 _h	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0
MF1S703yX	00 02 _h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Table 12. SAK response of the MF1S50yyX/V1

Sales Type	Hex Value	Bit Number							
		8	7	6	5	4	3	2	1
MF1S50yyX/V1	08h	0	0	0	0	1	0	0	0

2.3. Interfacing with MFRC522 module and MF1S50

Our group choose chip atmega328p in order to make this chip as the bridge to the communication between mfrc522 and MIFARE tag.

Firstly, we come to the function that initialize the MFRC522 chip:

```
void mfrc522::RFID_init(){
    sbi (PORTB, this->ss_pin);
    reset();

    // timer: 3390*2*48/13.56MHz = 24ms (for each cmd the timer will run for the
    next cmd)
    // prescaler
    write_to_reg(TModeReg, 0x8D);
    write_to_reg(TPrescalerReg, 0x3E);

    // reload_counter
    write_to_reg(TReloadRegL, 0x30);
    write_to_reg(TReloadRegH, 0x00);
}
```

```

// modulation
write_to_reg(TxASKReg, 0x40); // 100% ASK (amplitude shift keying)
write_to_reg(TModeReg, 0x3D); // transmit only when an RF field is generated
setbitmask(TxControlReg, 0x03); // allow RF signal to transmit on Tx1 and Tx2
pins

```

Next, we come to the function that reader can send command to tag:

```

int mfrc522::commandtag(uint8_t cmd, uint8_t *data, int dlen, uint8_t *res, int*
rlen){
    int status = mi_err;
    uint8_t irqen = 0x00;
    uint8_t wait_irq = 0x00;
    uint8_t lastbits, n ;
    int i ;

    switch(cmd){
        case mfrc522_mf_authent:
            irqen = 0x12; // enable idle cmd and error protocol interrupt to connect
irq pin
            wait_irq = 0x10;
            break;
        case mfrc522_transceive:
            irqen = 0x77; // enable idle cmd, error protocol, transmit, receive,
lower alert fifo and timer counter value gets zero interrupt to connect irq pin
            wait_irq = 0x30;
            break;
        default:
            break;
    }

    write_to_reg(CommIEnReg, irqen | 0x80); // interrupt request
    clearbitmask(CommIrqReg, 0x80); // clear all interrupt request bits (if it's
already set from the beginning)
    setbitmask(FIFOLevelReg, 0x80); // clear read and write pointer if FIFO

    write_to_reg(CommandReg, mfrc522_idle); // no action, cancel the current command

```

```

// write to fifo
for (i = 0; i < dlen; i++){
    write_to_reg(FIFODataReg, data[i]);
}

// execute the command
write_to_reg(CommandReg, cmd);

if (cmd == mfrc522_transceive){
    setbitmask(BitFramingReg, 0x80); // start the transmission and read
continuously between tag and reader (data is get from FIFO)
    // the last byte transmit is 8 bit
}

// waiting for the command complete, so that we can receive data

i = 25; // max wait time is 125ms
do {
    _delay_ms(5);
    // CommIRqReg[7..0]
    // Set1 TxIRq RxIRq IdleIRq HiAlerIRq LoAlertIRq ErrIRq TimerIRq
    n = read_from_reg(CommIrqReg);
    i--;
} while ((i != 0) && !(n & 0x01) && !(n & wait_irq)); // wait for timer, idle
interrupt, (and receive flag interrupt for transceive)

clearbitmask(BitFramingReg, 0x80); // stop transceive

if (i != 0) { // request did not timeout
    if (!(read_from_reg(ErrorReg) & 0x1D)) { // BufferOvfl Collerr CRCErr
ProtocolErr
        status = mi_ok;
        if (n & irqen & 0x01) {
            status = mi_notag_err;
        }
    }
}

```

```

        if (cmd == mfrc522_transceive){
            n = read_from_reg(FIFOLevelReg);
            lastbits = read_from_reg(ControlReg) & 0x07;
            if (lastbits){
                *rlen = (n - 1) * 8 + lastbits; // indicate that the number of
bits of the last byte can be different
            } else {
                *rlen = n * 8;
            }
        }

        if (n == 0) {
            n = 1;
        }
        if (n > MAX_LEN) {
            n = MAX_LEN; // 106KB per second : 128 bits (16 bytes) therefore can
communicate less than 16 bytes
        }

        // reading the receive data from FIFO
        for (i = 0; i < n; i++){
            res[i] = read_from_reg(FIFODataReg);
        }
    }

    else status = mi_err;
}
return status;
}

```

Based on the commandtag function above we can write the request tag function:

```

// just only for requesting mf1 tag
int mfrc522::requesttag(uint8_t* data){
    int status;
    int len;

    write_to_reg(BitFramingReg, 0x07); // 7 bits for request cmd

```



```

data[0] = mf1_reqa;
status = commandtag(mfrc522_transceive, data, 1, data, &len);
if ((status != mi_ok) || (len != 0x10)) { // 16-bit ATQA response from tag
    status = mi_err;
}

return status;
}

```

Next, we can get the NUID or identifier of the card by using anticollision function:

```

int mfrc522::anticollision(uint8_t* serial){
    int status, len;
    write_to_reg(BitFramingReg, 0x00);
    serial[0] = mf1_anticol_1;
    serial[1] = 0x20;
    status = commandtag(mfrc522_transceive, serial, 2, serial, &len);
    len = len / 8; //(unit: bytes )
    if ((len == 4)&&(status == mi_ok)) {
        status = mi_ok;
    }
    return status;
}

```

If you want to access to memory of the card and further storage and application (need authentication), you can use selecttag function:

```

uint8_t mfrc522::selecttag(uint8_t* serial){
    int i, status, len;
    uint8_t ack = 0x08 ;
    uint8_t buffer[8];

    write_to_reg(BitFramingReg, 0x00);

    buffer[0] = mf1_select_1;

```

```

    buffer[1] = 0x70;
    for (i = 0; i < 4; i++){
        buffer[i + 2] = serial[i];
    }
    calculateCRC(buffer, 6, &buffer[6]);

    status = commandtag(mfrc522_transceive, buffer, 8, buffer, &len);
    if ((status == mi_ok) && (buffer[0] == ack)) {
        status = mi_ok;
    }
    return status;
}

```

The halt function is final function that is necessary for the end of all the process above:

```

int mfrc522::halttag(){
    int status, len;
    uint8_t buffer[4];

    write_to_reg(BitFramingReg, 0x00);

    buffer[0] = mf1_halt;
    buffer[1] = 0;
    calculateCRC(buffer, 2, &buffer[2]);
    status = commandtag(mfrc522_transceive, buffer, 4, buffer, &len);
    clearbitmask(Status2Reg, 0x08); // turn off encryption
    return status;
}

```

For all the function about MFRC522 and MF1S50:

```

class mfrc522 {
private:
    uint8_t ss_pin;
    uint8_t rst;

```

```

public:
    mfr522          (int ss_pin, int reset);
    void    write_to_reg(uint8_t addr, uint8_t data);
    uint8_t read_from_reg(uint8_t addr);
    void    setbitmask(uint8_t address, unsigned char mask);
    void    clearbitmask(uint8_t address, unsigned char mask);
    void    reset();
    void    RFID_init();
    uint8_t getfirmwareversion();
    int     commandtag(uint8_t cmd, uint8_t *data, int dlen, uint8_t *res,
int* rlen);
    void    calculateCRC(uint8_t *data, int len, uint8_t *result);
    uint8_t selecttag(uint8_t* serial);
    int     requesttag(uint8_t* data);
    int     halttag();
    int     anticollision(uint8_t* serial);

};

```

2.4. Other functions

In this project, additionally we use LCD 16x2 with piggy black board for i2c protocol that can use less PIN then the normal communication (4 bit or 8 bit) and make the connection easier.

We had written the library of I2c_LCD protocol:

- Header file:

```

/*
 * i2c_lcd.h
 *
 * Created: 02/03/2024 9:40:26 SA
 * Author: Vũ Huy Bảo
 */

#ifndef I2C_LCD_H_

```

```

#define I2C_LCD_H_

#include "i2c_moudle.h"

#define slave_address 0x4E // slave address of piggyback board (PCF8574T) is 0x27
+ write(0)

#define lcd_functionset 0x28 // Function Set: 4-bit mode, 2 lines, 5x8 font
#define lcd_displayoff 0x08 // Display Off
#define lcd_cleardisplay 0x01 // Clear Display
#define lcd_entrymodeset 0x06 // Entry Mode Set: Increment cursor, no display
shift
#define lcd_displayon 0x0C // Display On
#define lcd_shiftleft 0x1C
#define lcd_shiftright 0x18

void i2c_lcd_sendcmd(char cmd);
void i2c_lcd_init();
void i2c_lcd_sendchar(char data);
void i2c_lcd_position(int col, int row);
void i2c_lcd_sendstring(char* str);

#endif /* I2C_LCD_H_ */

```

- CPP file:

```

/*
 * i2c_lcd.cpp
 *
 * Created: 02/03/2024 9:39:17 SA
 * Author: Vũ Huy Bảo
 */

#include "i2c_lcd.h"

void i2c_lcd_sendcmd(char cmd){
    unsigned char upper, lower;
    lower = (cmd<<4)&0xF0;

```

```

    upper = cmd & 0xF0;

    i2c_start();
    i2c_transmit(slave_address);
    _delay_ms(2);
    i2c_transmit(upper|0x0C);
    _delay_ms(2);
    i2c_transmit(upper|0x08);
    _delay_ms(2);
    i2c_transmit(lower|0x0C);
    _delay_ms(2);
    i2c_transmit(lower|0x08);
    _delay_ms(2);
    i2c_stop();
}

void i2c_lcd_init() {
    // Initialization sequence
    i2c_start();
    i2c_transmit(slave_address);
    _delay_ms(2);

    i2c_transmit(0x30|0x0C);
    _delay_ms(2);
    i2c_transmit(0x30|0x08);
    _delay_ms(5);

    i2c_transmit(0x30|0x0C);
    _delay_ms(2);
    i2c_transmit(0x30|0x08);
    _delay_ms(5);

    i2c_transmit(0x30|0x0C);
    _delay_ms(2);
    i2c_transmit(0x30|0x08);
    _delay_ms(5);
}

```

```

    i2c_transmit(0x20|0x0C);
    _delay_ms(2);
    i2c_transmit(0x20|0x08);
    _delay_ms(2);
    i2c_stop();

    // Send initialization commands
    i2c_lcd_sendcmd(lcd_functionset); // Function Set: 4-bit mode, 2 lines, 5x8
font
    i2c_lcd_sendcmd(lcd_cleardisplay); // Clear Display
    i2c_lcd_sendcmd(lcd_entrymodeset); // Entry Mode Set: Increment cursor, no
display shift
    i2c_lcd_sendcmd(lcd_displayon); // display on
}

void i2c_lcd_sendchar(char data){
    unsigned char upper, lower;
    lower = (data<<4)&0xF0;
    upper = data & 0xF0;

    i2c_start();
    i2c_transmit(slave_address);
    _delay_ms(2);
    i2c_transmit(upper|0x0D);
    _delay_ms(2);
    i2c_transmit(upper|0x09);
    _delay_ms(2);
    i2c_transmit(lower|0x0D);
    _delay_ms(2);
    i2c_transmit(lower|0x09);
    _delay_ms(2);
    i2c_stop();
}

void i2c_lcd_sendstring(char* str){
    while (*str != '\0'){

```

```

        i2c_lcd_sendchar(*(str++));
    }
}

void i2c_lcd_position(int col, int row) {
    switch(row) {
        case 1:
            i2c_lcd_sendcmd(0x80 | (0x00 + col - 1));
            break;
        case 2:
            i2c_lcd_sendcmd(0x80 | (0x40 + col - 1));
            break;
        default:
            break;
    }
}
}

```

Main function:

```

/*
 * RFID_project.cpp
 *
 * Created: 14/02/2024 3:28:33 CH
 * Author : Vũ Huy Bảo
 */

#include "RFID.h"
#include "spi_module.h"
#include "uart_module.h"
#include "i2c_moudle.h"
#include "i2c_lcd.h"

mfrc522 x(ss,0);

void main_init(){
    DDRC = (1<<0)|(1<<1)|(1<<2);
}

```

```

    i2c_init();
    i2c_lcd_init();
    spi_init();
    uart_init();
    x.RFID_init();
}

void print_lcd(uint8_t x){
    int temp;
    int hex[2];
    for (int i = 0; i<2; i++){
        temp = x % 16;
        x/=16;
        switch (temp){
            case 10:
                hex [i] = 'A';
                break;

                case 11:
                    hex [i] = 'B';
                    break;

                    case 12:
                        hex [i] = 'C';
                        break;

            case 13:

                hex [i] = 'D';

                break;

                case 14:

                    hex [i] = 'E';

                    break;

                    case 15:

```



```

        hex[i] = 'F';

        break;

    default:
        hex[i] = temp + 0x30;
    }
}

for (int i = 1 ;i >= 0; i--){
    i2c_lcd_sendchar((char)(hex[i]));
}
}

int main(void)
{
    main_init();
    int status = mi_err;
    uint8_t serial[MAX_LEN];
    uint8_t buffer[4];

    buffer[0] = 0x46;
    buffer[1] = 0xE9;
    buffer[2] = 0xEC;
    buffer[3] = 0xAD;

    int cnt = 0,same = 0;

    i2c_lcd_position(1,1);
    i2c_lcd_sendstring("let start");
    _delay_ms(3000);
    i2c_lcd_sendcmd(lcd_cleardisplay);
    uint8_t a = x.getfirmwareversion();
    if (a == 0x91) i2c_lcd_sendstring("version 1.0");
    else if (a == 0x92) i2c_lcd_sendstring("version 2.0");
    else i2c_lcd_sendstring("no receive cmd");
}

```

```

/* Replace with your application code */
while (1)
{
    same = 0;
    here:
    i2c_lcd_position(1,2);
    cnt++;
    if (cnt == 1) i2c_lcd_sendstring("                "); // clear
display just only 1 time
    status=x.requesttag(serial);
    PORTC ^= (1<<2);
    _delay_ms(500);
    if(status == mi_ok) {
        status = x.anticollision (serial);
        if (status == mi_ok) {
            cnt = 0;
            i2c_lcd_position(1,2);
            i2c_lcd_sendstring("NUID: ");
            //    uart_transmit_string("NUID: ");
            for (int i = 0;i < 4; i++){
                print_lcd(serial[i]);
            //    uart_transmit_char(serial[i]);
            }

            for(int i = 0 ;i<4;i++){
                if (serial[i] == buffer[i]) same++;
            }

            if (same==4) {
                PORTC |= (1<<1);
                _delay_ms(5000);
                PORTC &= ~(1<<1);
            }
            else{
                PORTC |= (1<<0);

```

```

        _delay_ms(5000);
        PORTC &= ~(1<<0);
    }

    status = x.selecttag(serial);
    status = x.halttag();
}

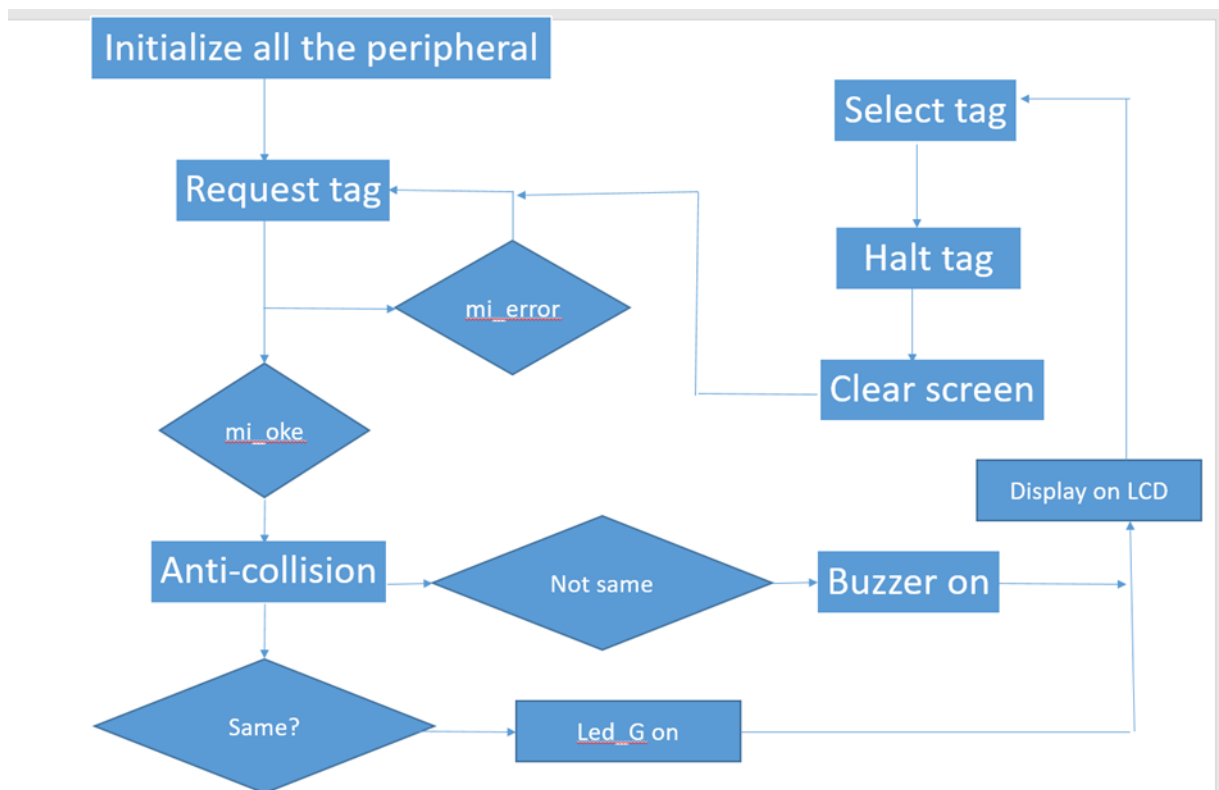
}

}

return 0;
}

```

Flow chart of main function:



III. DESIGN SCHEMATIC

The description of the RFID system using the components:

The RFID system consists of several components connected together. At the center of the system is the ATmega328 microcontroller, which acts as the main control unit. Connected to the microcontroller is the MFRC522 RFID module, which handles the communication with RFID tags.

The MFRC522 module communicates with the microcontroller using SPI (Serial Peripheral Interface) protocol. It provides the necessary functionality to read and write data to RFID tags.

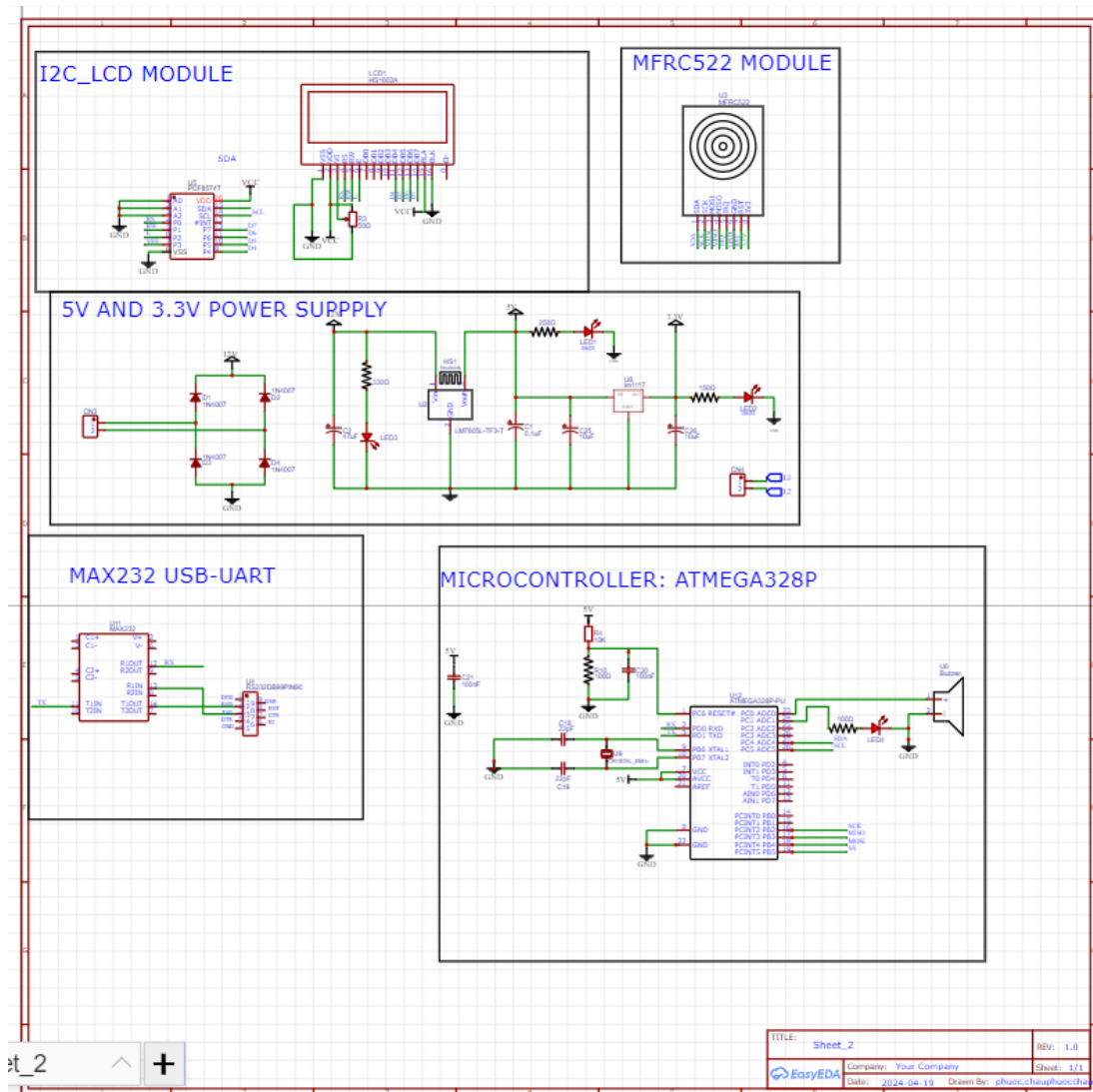
The system also includes an I2C LCD module, which is connected to the microcontroller via the I2C bus. The LCD module allows for displaying relevant information, such as the ID or data stored on the RFID tag.

To enable serial communication with other devices, a MAX232 module is used. The MAX232 module converts the TTL (Transistor-Transistor Logic) level signals from the microcontroller to RS232 voltage levels, which are commonly used for serial communication. This allows the microcontroller to communicate with external devices, such as a computer or another microcontroller, using serial communication protocols.

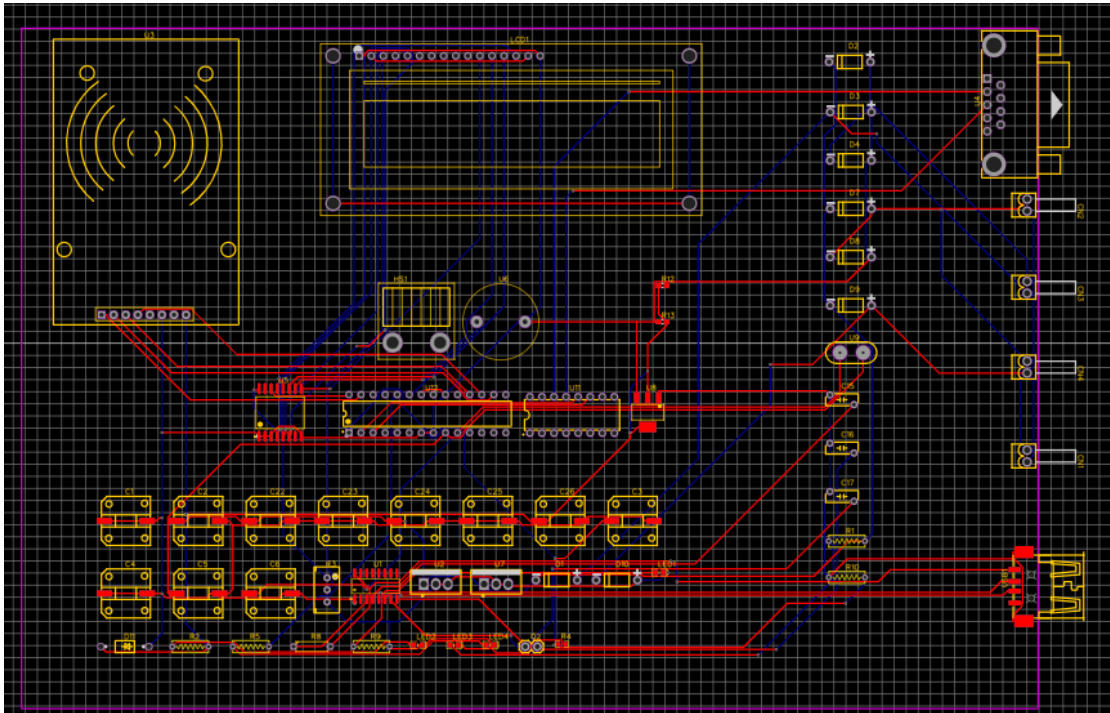
To power the system, a 5V to 3.3V power supply is used. The power supply ensures that all the components receive the appropriate voltage levels for their operation. This is important because the MFRC522 module typically operates at 3.3V, while the ATmega328 microcontroller and other components may require 5V.

The schematic provided is a basic representation and include specific pin connections or additional components that may be required for proper functionality. However, it is recommended to consult the datasheets and pinout diagrams of the individual components for accurate connection details and any additional requirements.

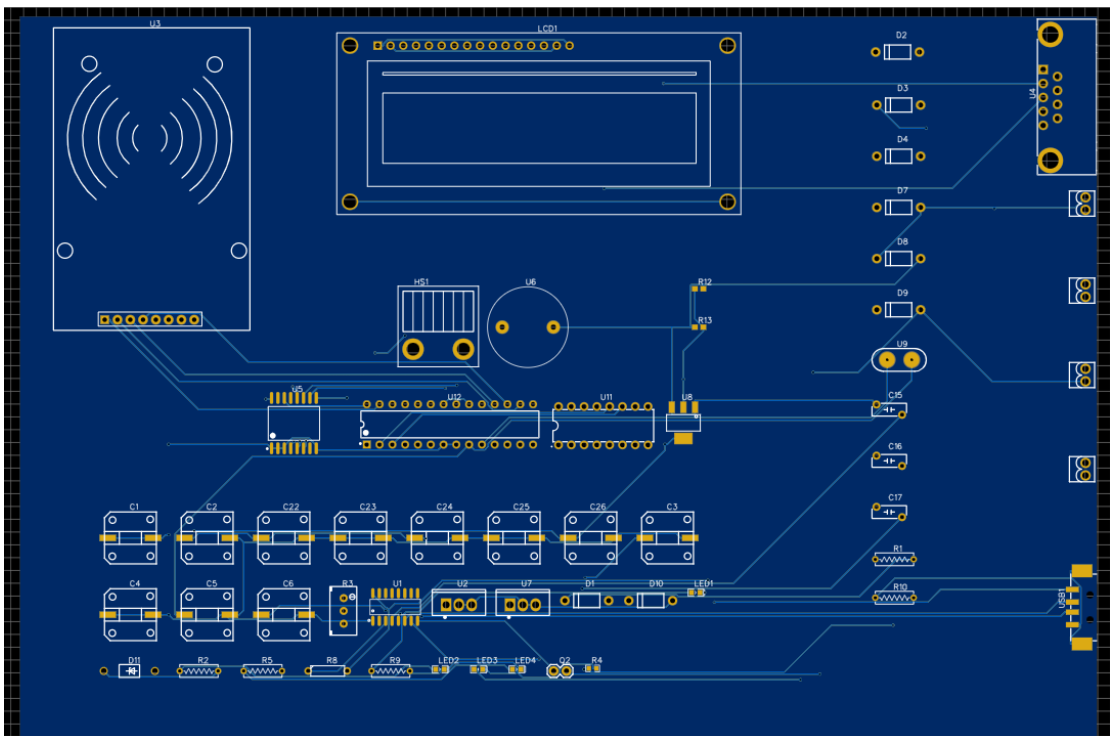
Project schematic:



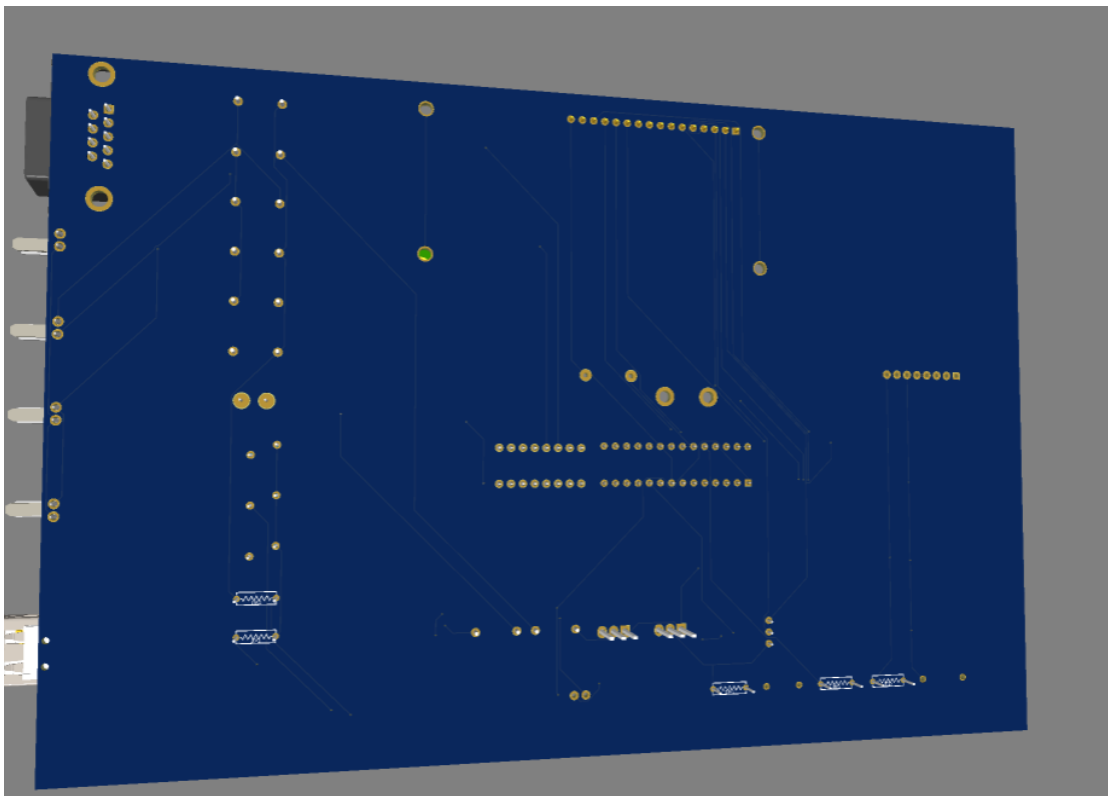
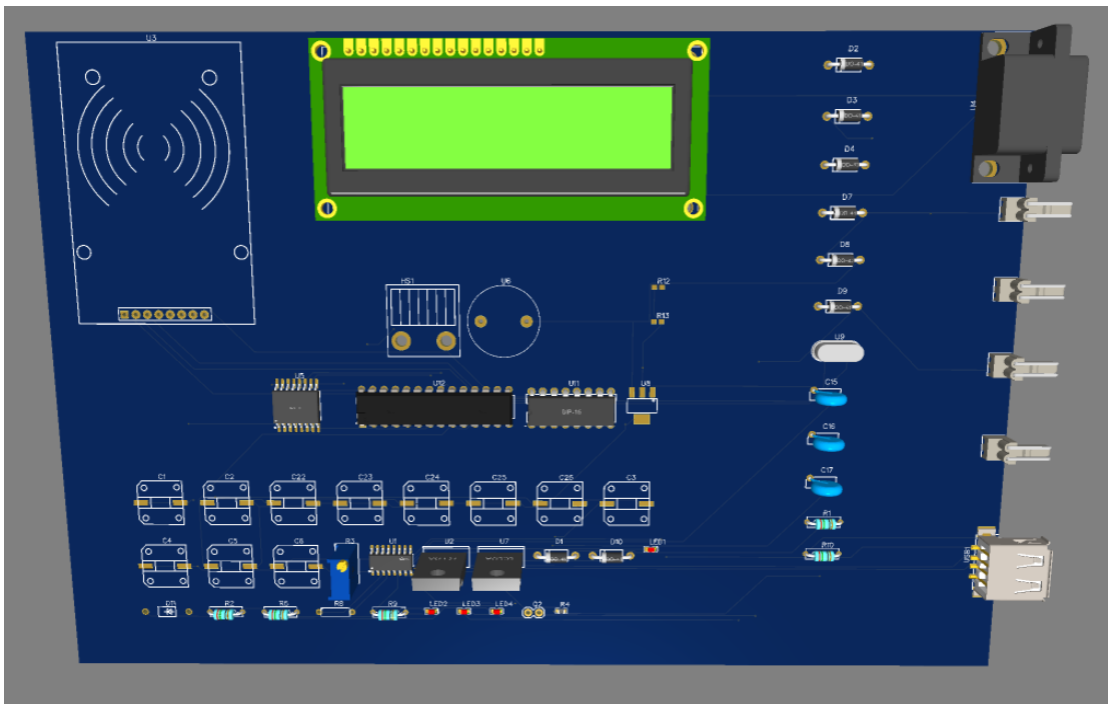
Project PCB layout:



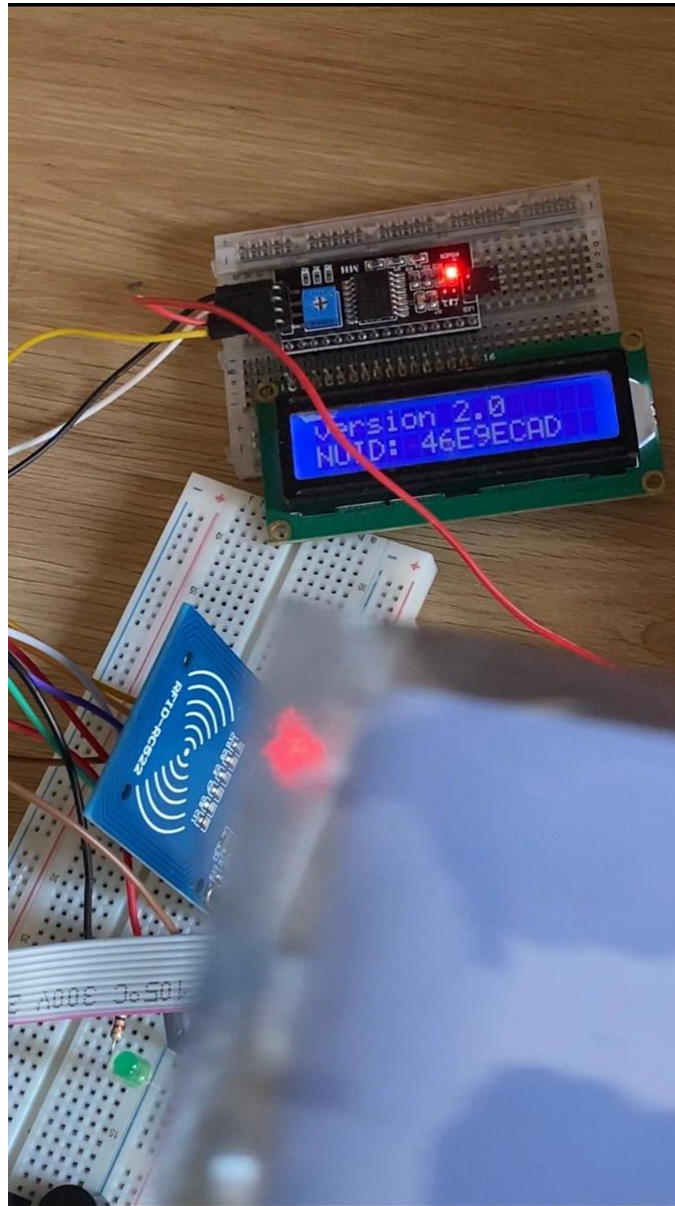
2D image convert from PCB:



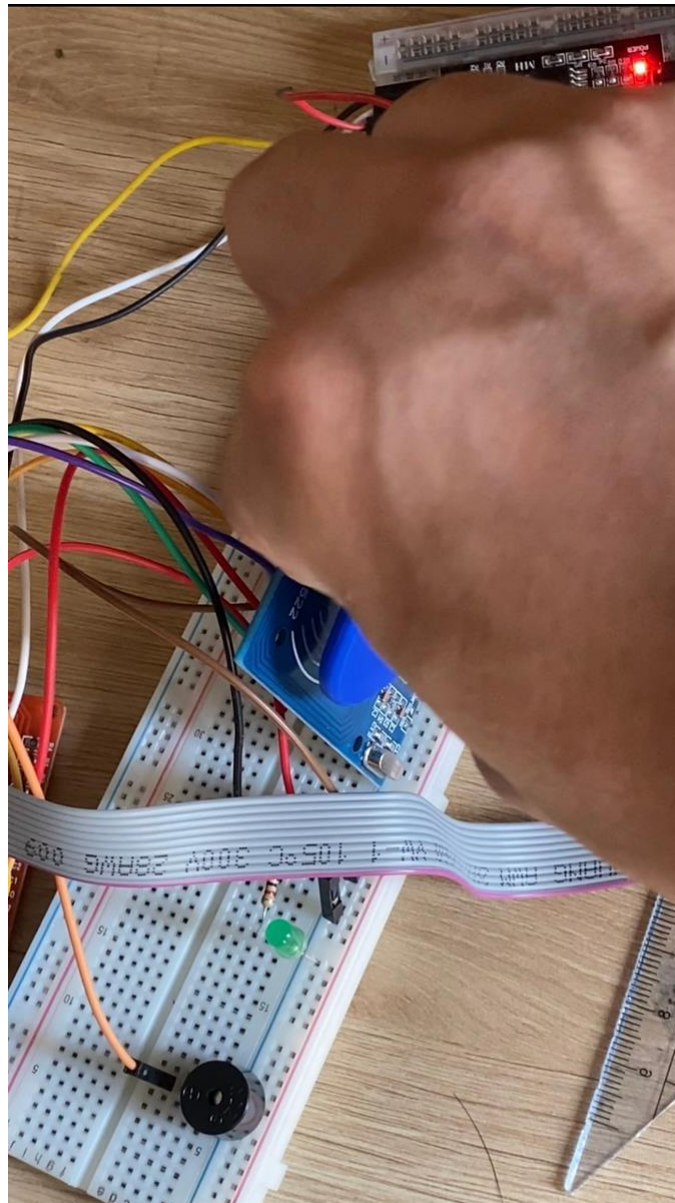
3D image front and back view:

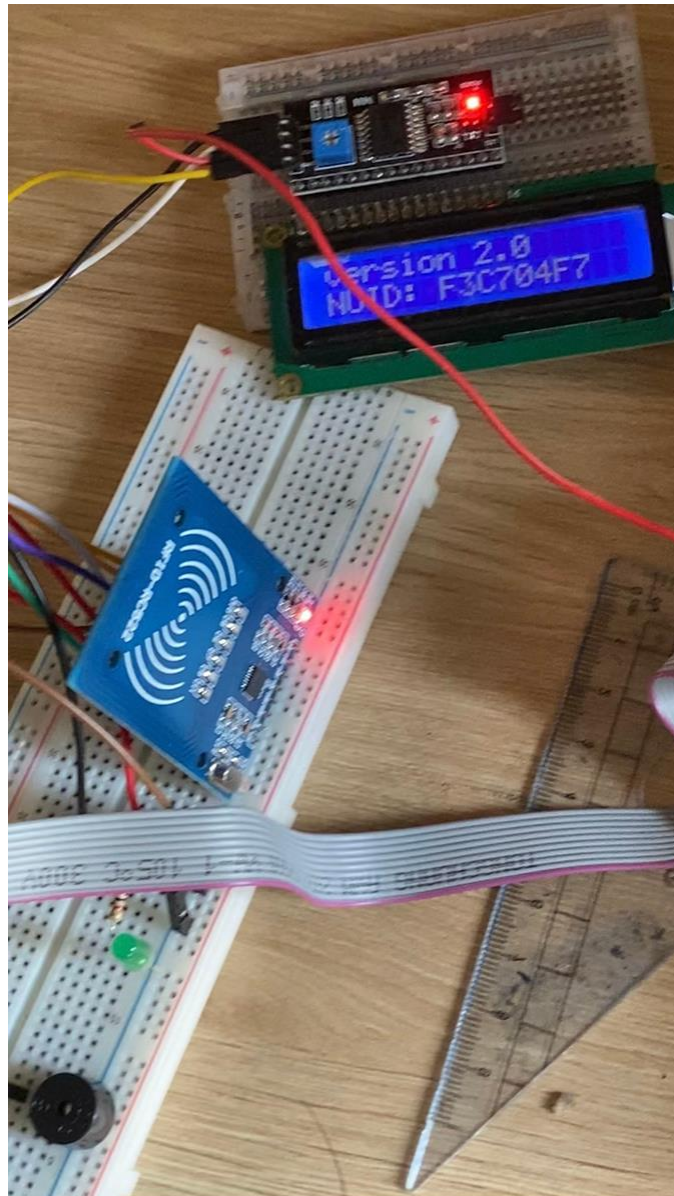


IV. PROJECT RESULT









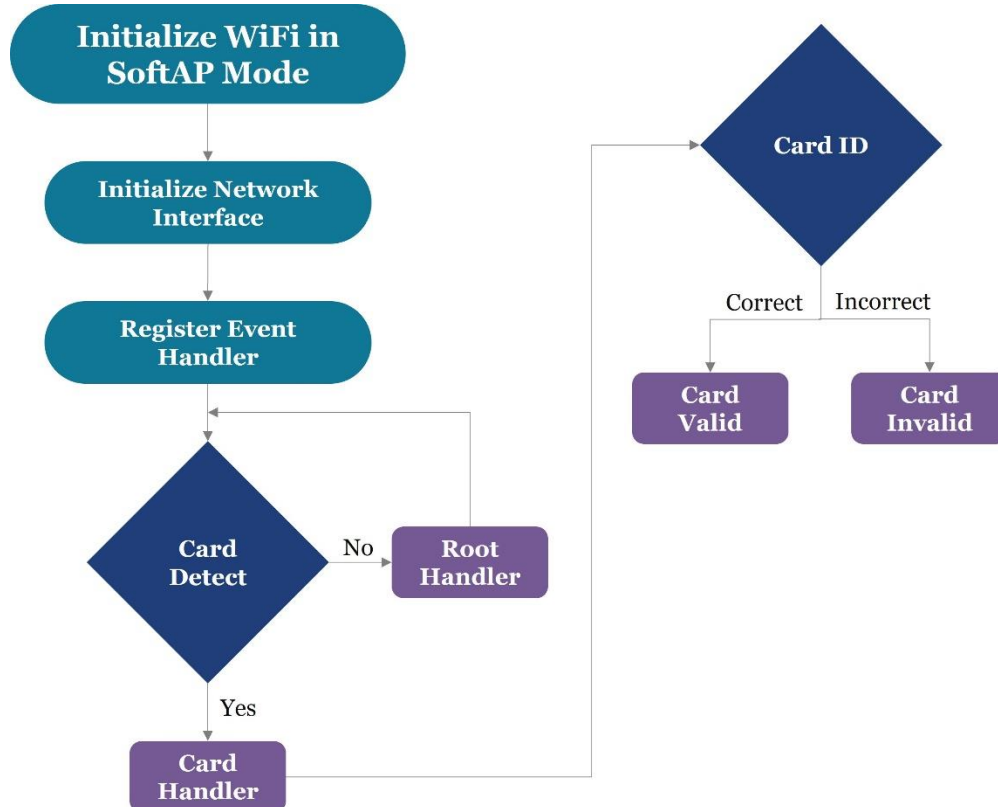
V. FUTURE DEVELOPMENT

In this project, we aim to develop a web interface for the RFID card reader application utilizing the ESP32's WiFi capabilities. By integrating an RFID card scanner with the ESP32 and leveraging its WiFi capabilities, we can create a convenient and accessible solution for managing RFID card data.

The web interface will provide users with a user-friendly platform to interact with the RFID card scanner remotely. Users will be able to perform various tasks such as scanning RFID cards, viewing scanned card data, and possibly configuring the system settings through the web interface.

By combining the power of the ESP32 microcontroller with web technology, we can create a flexible and scalable solution suitable for various applications, including access control systems, attendance tracking, and inventory management.

The flowchart of the soft access point creation on ESP32 with RFID web interface:



Website interface source code:

```

/*
  WiFi softAP Example
*/
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_mac.h"
#include "esp_wifi.h"
#include "esp_event.h"
#include "esp_log.h"
#include "nvs_flash.h"

#include "lwip/err.h"
#include "lwip/sys.h"
#include <stdbool.h>

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/param.h>
#include "esp_netif.h"
#include <esp_http_server.h>
#include "esp_tls.h"

#include "driver/gpio.h"
#include <driver/uart.h>
#include <soc/uart_struct.h>

#if !CONFIG_IDF_TARGET_LINUX
#include <esp_wifi.h>
#include <esp_system.h>
#include "nvs_flash.h"
#endif // !CONFIG_IDF_TARGET_LINUX

#define EXAMPLE_HTTP_QUERY_KEY_MAX_LEN (64)

static const char *TAG = "webserver";
const char *predetermined_values[] = {"12 34 45 43", "56 78 34 56", "AB CD
43 21", "EF GH 34 75"};

#define LED GPIO_NUM_2
#define CARD_UID_LENGTH 12 // Assuming RFID card UID is of 4 bytes length
char card_uid[CARD_UID_LENGTH];

/*****WEBSERVER CODING SECTION*****/

// Function to check if card UID matches any predetermined values

```

```

bool isCardValid(const char *card_uid, const char **predetermined_values,
int num_values) {
    // Iterate through predetermined values array
    for (int i = 0; i < num_values; i++) {
        // Compare card UID with predetermined value
        if (strncmp(card_uid, predetermined_values[i], CARD_UID_LENGTH) ==
0) {
            // Card UID matches, return true
            return true;
        }
    }
    // Card UID does not match any predetermined value, return false
    return false;
}

static esp_err_t card_handler(httpd_req_t *req)
{
    esp_err_t error;

    if (isCardValid(card_uid, predetermined_values, 4))
    {
        ESP_LOGI(TAG, "Card Valid!"); // send log RFID card valid
        gpio_set_level(LED, 1);
    }
    else
    {
        ESP_LOGI(TAG, "Card Invalid!"); // send log RFID card invalid
        gpio_set_level(LED, 0);
    }

    const char *response; //= (const char *) req->user_ctx; // copy user ctx
into response
    if (isCardValid(card_uid, predetermined_values, 4))
    {
        response = "<!DOCTYPE html>\
<html>\
<head>\
<style>\
body {\
    font-family: 'Roboto', sans-serif; /* Google font */\
}\
\
.infographic {\
    border: 1px solid #0077B6; /* Border color */\
    padding: 10px;\
    margin-bottom: 20px;\

```

```

border-radius: 10px; /* Smoother corners */\
background-color: #E5F6F8; /* Background color */\
}\
\
.card-status {\
border: 1px solid #08A045; /* Green border color */\
padding: 10px;\
margin-top: 20px;\
border-radius: 10px; /* Smoother corners */\
background-color: #08A045; /* Green background color */\
text-align: center; /* Center align text */\
color: #FFFFFF; /* White text color */\
display: flex; /* Use flexbox for centering */\
justify-content: center; /* Center horizontally */\
align-items: center; /* Center vertically */\
}\
\
.card-status h3 {\
margin: 0; /* Remove default margin */\
}\
\
.card-uid {\
font-weight: bold; /* Bold font */\
}\
\
h1 {\
text-align: center;\
color: #0077B6;\
}\
\
h3 {\
text-align: left; /* Left align the text */\
}\
</style>\
<link
href=\"https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=
swap\" rel=\"stylesheet\">\
</head>\
<body>\
\
<h1>RFID CARD READER WEB SERVER</h1>\
<div class=\"infographic\">\
<p class=\"card-uid\">Card UID: %s</p>\
<h3>Card state: Read</h3>\
</div>\
\
<div class=\"card-status\">\
<h3>Card Valid</h3>\

```



```

</div>\
\
</body>\
</html>"
;
    }
    else
    {
        response = "<!DOCTYPE html>\
<html>\
<head>\
<style>\
body {\
    font-family: 'Roboto', sans-serif; /* Google font */\
}\
\
.infographic {\
    border: 1px solid #0077B6; /* Border color */\
    padding: 10px;\
    margin-bottom: 20px;\
    border-radius: 10px; /* Smoother corners */\
    background-color: #E5F6F8; /* Background color */\
}\
\
.card-status {\
    border: 1px solid #D21401; /* Green border color */\
    padding: 10px;\
    margin-top: 20px;\
    border-radius: 10px; /* Smoother corners */\
    background-color: #D21401; /* Green background color */\
    text-align: center; /* Center align text */\
    color: #FFFFFF; /* White text color */\
    display: flex; /* Use flexbox for centering */\
    justify-content: center; /* Center horizontally */\
    align-items: center; /* Center vertically */\
}\
\
.card-status h3 {\
    margin: 0; /* Remove default margin */\
}\
\
.card-uid {\
    font-weight: bold; /* Bold font */\
}\
\
h1 {\
    text-align: center;\
    color: #0077B6;\

```



```

}\
\
h3 {\
    text-align: left; /* Left align the text */\
}\
</style>\
<link
href=\"https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=
swap\" rel=\"stylesheet\">\
</head>\
<body>\
\
<h1>RFID CARD READER WEB SERVER</h1>\
<div class=\"infographic\">\
    <p class=\"card-uid\">Card UID: %s</p>\
    <h3>Card state: Read</h3>\
</div>\
\
<div class=\"card-status\">\
    <h3>Card Invalid</h3>\
</div>\
\
</body>\
</html>\"
;
    }

    char formatted_response[2000];
    snprintf(formatted_response, sizeof(formatted_response), response,
card_uid);

    error = httpd_resp_send(req, formatted_response,
strlen(formatted_response)); // server sends response
    if (error != ESP_OK)
    {
        ESP_LOGI(TAG, \"Error %d while sending Response\", error);
    }
    else ESP_LOGI(TAG, \"Response sent Successfully\");
    return error;
}

static const httpd_uri_t card = {
    .uri      = \"/card\",
    .method   = HTTP_GET,
    .handler  = card_handler,
};

```

```
static esp_err_t root_handler(httpd_req_t *req)
{
    esp_err_t error;
    ESP_LOGI(TAG, "No Card!"); // send log RFID card valid
    gpio_set_level(LED, 1);
    const char *response = (const char *) req->user_ctx; // copy user ctx
into response
    error = httpd_resp_send(req, response, strlen(response)); // server
sends response
    if (error != ESP_OK)
    {
        ESP_LOGI(TAG, "Error %d while sending Response", error);
    }
    else ESP_LOGI(TAG, "Response sent Successfully");
    return error;
}
```

```
static const httpd_uri_t root = {
    .uri      = "/",
    .method   = HTTP_GET,
    .handler  = root_handler,
    /* Let's pass response string in user
    * context to demonstrate it's usage */
    .user_ctx = "<!DOCTYPE html>\
<html>\
<head>\
<style>\
body {\
    font-family: 'Roboto', sans-serif; /* Google font */\
}\
\
.infographic {\
    border: 1px solid #0077B6; /* Border color */\
    padding: 10px;\
    margin-bottom: 20px;\
    border-radius: 10px; /* Smoother corners */\
    background-color: #E5F6F8; /* Background color */\
}\
\
.author-box {\
    border: 1px solid #0077B6; /* Border color */\
    padding: 10px;\
    margin-top: 20px;\
    border-radius: 10px; /* Smoother corners */\
    background-color: #E5F6F8; /* Background color */\
}
```

```

}\
\
h1 {\
    text-align: center;\
    color: #0077B6;\
}\
</style>\
<link
href=\"https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=
swap\" rel=\"stylesheet\">\
</head>\
<body>\
\
<h1>RFID CARD READER WEB SERVER</h1>\
<div class=\"infographic\">\
    <h3>Card state: Not read</h3>\
</div>\
\
<div class=\"author-box\">\
    <p><strong>Author: Group 06</strong></p>\
</div>\
\
</body>\
</html>\
"
};

esp_err_t http_404_error_handler(httpd_req_t *req, httpd_err_code_t err)
{
    /* For any other URI send 404 and close socket */
    httpd_resp_send_err(req, HTTPD_404_NOT_FOUND, "Some 404 error message");
    return ESP_FAIL;
}

static httpd_handle_t start_webserver(void)
{
    httpd_handle_t server = NULL;
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();
    #if CONFIG_IDF_TARGET_LINUX
        // Setting port as 8001 when building for Linux. Port 80 can be used
        // only by a privileged user in linux.
        // So when a unprivileged user tries to run the application, it throws
        // bind error and the server is not started.
        // Port 8001 can be used by an unprivileged user as well. So the
        // application will not throw bind error and the
        // server will be started.
        config.server_port = 8001;
    #endif
}

```

```

#endif // !CONFIG_IDF_TARGET_LINUX
    config.lru_purge_enable = true;

    // Start the httpd server
    ESP_LOGI(TAG, "Starting server on port: '%d'", config.server_port);
    if (httpd_start(&server, &config) == ESP_OK) {
        // Set URI handlers
        ESP_LOGI(TAG, "Registering URI handlers");
        httpd_register_uri_handler(server, &root);
        httpd_register_uri_handler(server, &card);
        return server;
    }

    ESP_LOGI(TAG, "Error starting server!");
    return NULL;
}

#if !CONFIG_IDF_TARGET_LINUX
static esp_err_t stop_webserver(httpd_handle_t server)
{
    // Stop the httpd server
    return httpd_stop(server);
}

static void disconnect_handler(void* arg, esp_event_base_t event_base,
                              int32_t event_id, void* event_data)
{
    httpd_handle_t* server = (httpd_handle_t*) arg;
    if (*server) {
        ESP_LOGI(TAG, "Stopping webserver");
        if (stop_webserver(*server) == ESP_OK) {
            *server = NULL;
        } else {
            ESP_LOGE(TAG, "Failed to stop http server");
        }
    }
}

static void connect_handler(void* arg, esp_event_base_t event_base,
                            int32_t event_id, void* event_data)
{
    httpd_handle_t* server = (httpd_handle_t*) arg;
    if (*server == NULL) {
        ESP_LOGI(TAG, "Starting webserver");
        *server = start_webserver();
    }
}
#endif // !CONFIG_IDF_TARGET_LINUX

```

```

/*****WEBSERVER CODING END*****/

static void configure_led(void)
{
    gpio_reset_pin(LED);
    gpio_set_direction(LED, GPIO_MODE_OUTPUT);
}

/* The examples use WiFi configuration that you can set via project
configuration menu.

If you'd rather not, just change the below entries to strings with
the config you want - ie #define EXAMPLE_WIFI_SSID "mywifissid"
*/
#define EXAMPLE_ESP_WIFI_SSID      CONFIG_ESP_WIFI_SSID
#define EXAMPLE_ESP_WIFI_PASS      CONFIG_ESP_WIFI_PASSWORD
#define EXAMPLE_ESP_WIFI_CHANNEL    CONFIG_ESP_WIFI_CHANNEL
#define EXAMPLE_MAX_STA_CONN        CONFIG_ESP_MAX_STA_CONN

static void wifi_event_handler(void* arg, esp_event_base_t event_base,
                               int32_t event_id, void* event_data)
{
    if (event_id == WIFI_EVENT_AP_STACONNECTED) {
        wifi_event_ap_staconnected_t* event =
(wifi_event_ap_staconnected_t*) event_data;
        ESP_LOGI(TAG, "station \"MACSTR\" join, AID=%d",
                  MAC2STR(event->mac), event->aid);
    } else if (event_id == WIFI_EVENT_AP_STADISCONNECTED) {
        wifi_event_ap_stadisconnected_t* event =
(wifi_event_ap_stadisconnected_t*) event_data;
        ESP_LOGI(TAG, "station \"MACSTR\" leave, AID=%d",
                  MAC2STR(event->mac), event->aid);
    }
}

void wifi_init_softap(void)
{
    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());
    esp_netif_create_default_wifi_ap();

    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));

    ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT,

```

```

ESP_EVENT_ANY_ID,
&wifi_event_handler,
NULL,
NULL));

wifi_config_t wifi_config = {
    .ap = {
        .ssid = EXAMPLE_ESP_WIFI_SSID,
        .ssid_len = strlen(EXAMPLE_ESP_WIFI_SSID),
        .channel = EXAMPLE_ESP_WIFI_CHANNEL,
        .password = EXAMPLE_ESP_WIFI_PASS,
        .max_connection = EXAMPLE_MAX_STA_CONN,
#ifdef CONFIG_ESP_WIFI_SOFTAP_SAE_SUPPORT
        .authmode = WIFI_AUTH_WPA3_PSK,
        .sae_pwe_h2e = WPA3_SAE_PWE_BOTH,
#else /* CONFIG_ESP_WIFI_SOFTAP_SAE_SUPPORT */
        .authmode = WIFI_AUTH_WPA2_PSK,
#endif
        .pmf_cfg = {
            .required = true,
        },
    },
};
if (strlen(EXAMPLE_ESP_WIFI_PASS) == 0) {
    wifi_config.ap.authmode = WIFI_AUTH_OPEN;
}

ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_AP));
ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_AP, &wifi_config));
ESP_ERROR_CHECK(esp_wifi_start());

ESP_LOGI(TAG, "wifi_init_softap finished. SSID:%s password:%s
channel:%d",
        EXAMPLE_ESP_WIFI_SSID, EXAMPLE_ESP_WIFI_PASS,
EXAMPLE_ESP_WIFI_CHANNEL);
}

void app_main(void)
{
    strcpy(card_uid, "12 34 01 FA");
    static httpd_handle_t server = NULL;

    configure_led();
    //Initialize NVS
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret ==
ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());

```

```

    ret = nvs_flash_init();
}
ESP_ERROR_CHECK(ret);

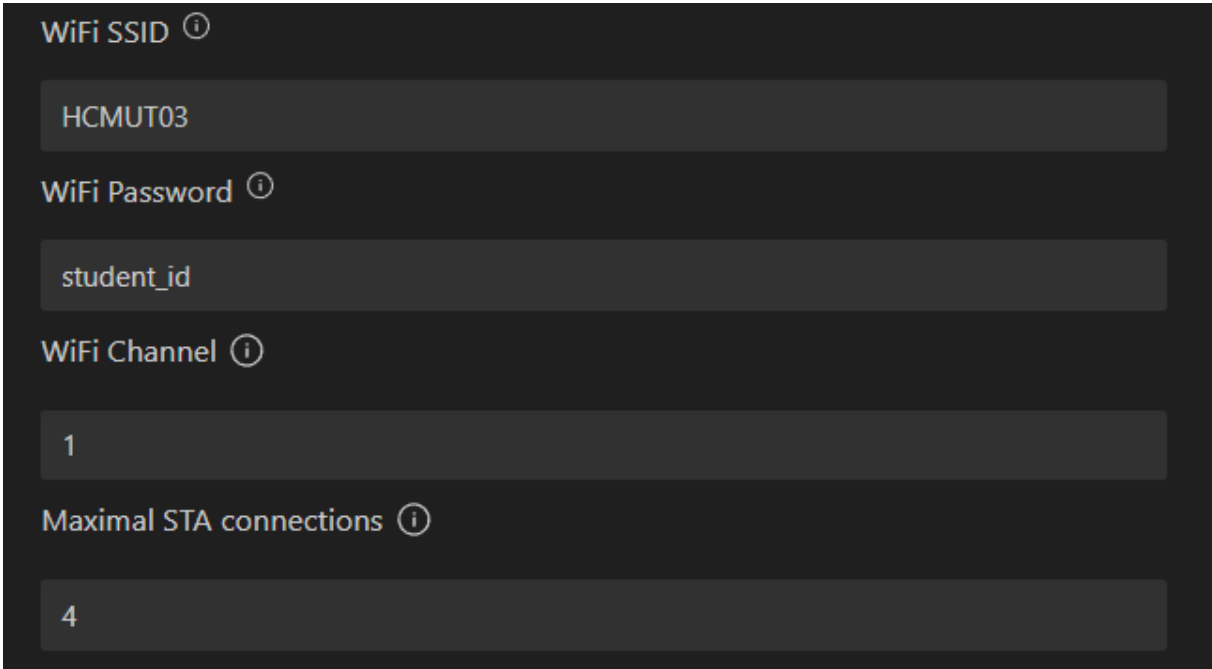
ESP_LOGI(TAG, "ESP_WIFI_MODE_AP");
wifi_init_softap();

ESP_ERROR_CHECK(esp_netif_init());

ESP_ERROR_CHECK(esp_event_handler_register(IP_EVENT,
IP_EVENT_AP_STAIPASSIGNED, &connect_handler, &server));
}

```

Since the website functions on a private WiFi station which is hosted by the ESP32, to connect to the website, we need to connect our device to the corresponding WiFi. You can configure the WiFi SSID and the WiFi password to your choice. In this case, we configure it will Menuconfig in VSCode extension for ESP-IDF.

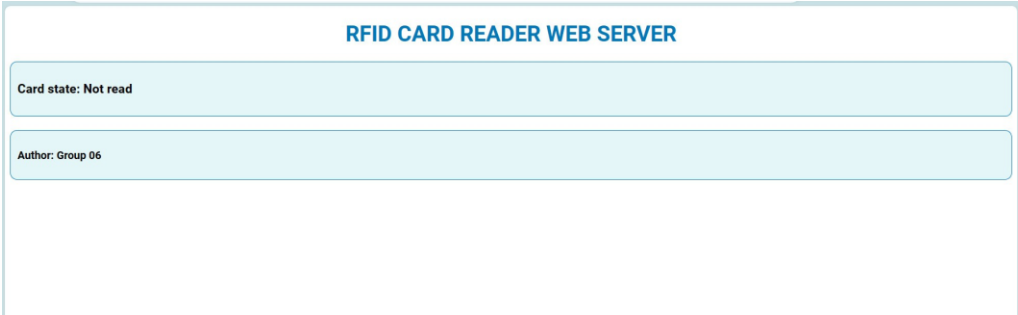


The screenshot shows the Menuconfig interface for configuring WiFi. It includes the following fields:

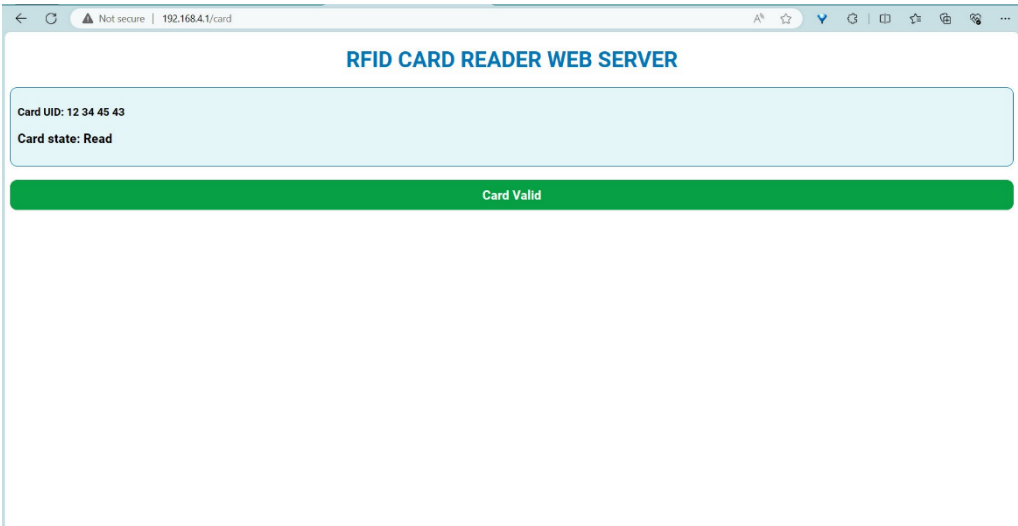
- WiFi SSID** (with an information icon): Set to `HCMUT03`.
- WiFi Password** (with an information icon): Set to `student_id`.
- WiFi Channel** (with an information icon): Set to `1`.
- Maximal STA connections** (with an information icon): Set to `4`.

The web interface will returns the corresponding display depending on the user card validity.

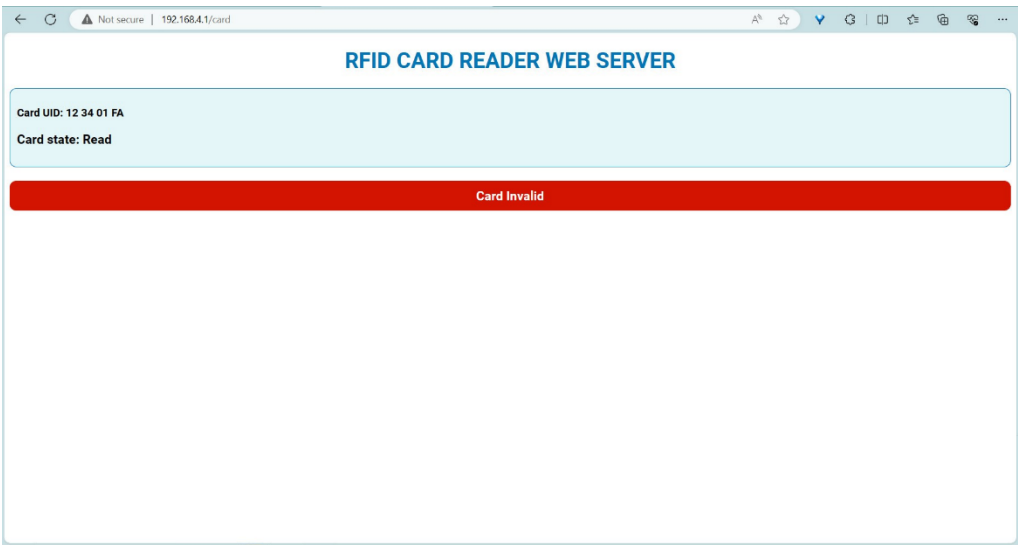
Home case:



Valid card case:



Invalid card case:



VI. REFERENCE

1. MSc. Bùi Quốc Bảo. “Embedded system”, Jan. 2024, Ho Chi Minh University. Lecture.
2. “ESP-IDF Programming Guide - ESP32 - — ESP-IDF Programming Guide v5.2.1 documentation,” docs.espressif.com. <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/> (accessed Jan 20, 2024).
3. NXP, “*MIFARE Classic EV1 1K - Mainstream contactless smart card IC for fast and easy solution development*,” 279232 datasheet, Mar. 2014 [Revised May. 2018].
4. NXP, “*Standard 3V MIFARE reader solution*,” 112138 datasheet, Oct. 2009 [Revised Sep. 2017].