THE TECHNICAL UNIVERSITY of CLUJ-NAPOCA

Faculty of Automation and Computer-Science

Automated teller machine

-Digital System Design project-

Keresztes Beáta
Fazakas Borbála
Group 30412

2020

Table of contents

I. Specifications

The device simulates an automated teller machine on an FPGA board which enables customers to perform 4 financial operations, on condition that the card has been successfully validated. The functionalities are the following:

Card validation

After the ATM has started working, the users can choose to log-in at any time. They first need to give the number of their card, and then they are asked to introduce the pin code. If the pin code corresponds to the one set up previously for the given card, then the user gains access to the main menu, where they can choose from any of the 5 possible operations. Otherwise, if the card number and the pin code do not correspond, the user cannot perform any operation but he may try again to introduce the correct pin code and/or the correct card number.

The main menu

After a successful login, the user may choose any of the 4 available operations, or he may choose to log out. Remark that all amounts are considered to be in euro.

The five available operations are:

1. **Reset pin**: the user may change his pin code anytime
2. **Check balance**: the balance currently available on his card will be displayed on the 7-segment displays
3. **Deposit**: the user may deposit a certain amount of money on his card
4. **Extract**: the user may extract a certain amount of money, on condition that the amount does not exceed the balance on his card. The extracted banknotes will be given one by one to the user
5. **Log out:** before leaving, the user must log out from his account such that other users will not have access to his data and money. At this point, the user receives a receipt

Constraints

Because of the limitations of the FPGA board(it can display only 4 characters at once and there are only 16 switches), there are some constraints which need to be taken into account whereas performing an operation.

- For deposits: all the banknotes should have the value of a valid euro banknote(i.e. 5, 10, 20, 50, 100, 200 or 500). The ATM is not supplied with a currency detector. As a result, it’s the user’s responsibility to introduce a valid value, otherwise, the correctness of the operation and of the future operations can’t be guaranteed. However, more banknotes can be deposited one after the other.
- For extractions: the maximum amount that can be extracted at once is 1000 euro. Any integer amount between 0 and 1000 is a valid amount for the extraction operation, but if the given amount can’t be withdrawn using the banknotes existing in the ATM or if the amount is greater than the balance on the card, then the operation won’t be performed, and a message will be displayed instead.
- For the balance: the maximum balance that can be deposited on a card is 9999 euros
- For the pin code: it is a 4 digit number. By default, all the pin codes are set to 0000, but the user can change his pin code to any other 4 digit number later
- The card number: at the moment, the available card numbers are 0, 1, 2 and 3

II. Implementation

Structure

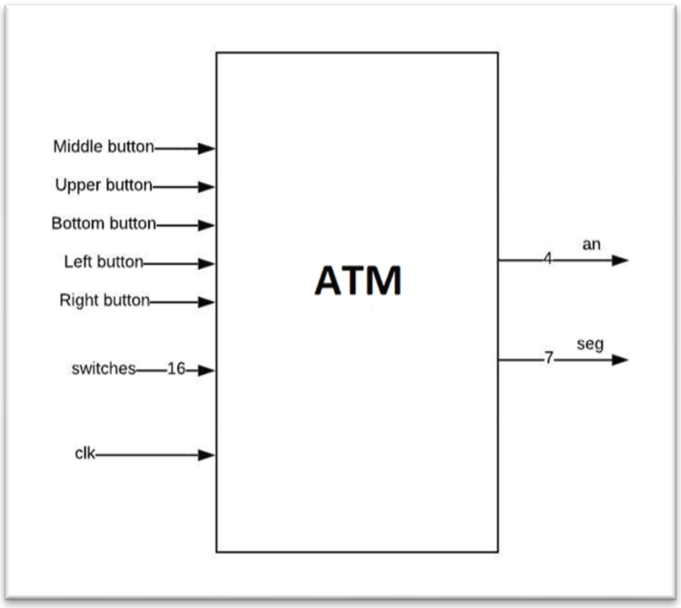
The atm is structured in different units which are controlled by the main control unit. Each unit has a certain function and is connected to the main system through internal signals.

- **Accounts unit**: processes and stores information related to the user's account
- **Banknotes unit**: stores the number of banknotes and performs banknotes related operations
- **Display unit**: controls the 4 in-built 7 segment displays, more precisely which messages should be displayed on it.
- **Frequency divider unit**: slows down the initial clock to reach the optimal frequency for the system's functioning
- **Converter unit**: performs the conversions from BCD to binary format and vice versa
- **Debouncers**: used to debounce the inputs from the buttons

Black box

The black box representation allows visibility only over the input/output signals of the atm.

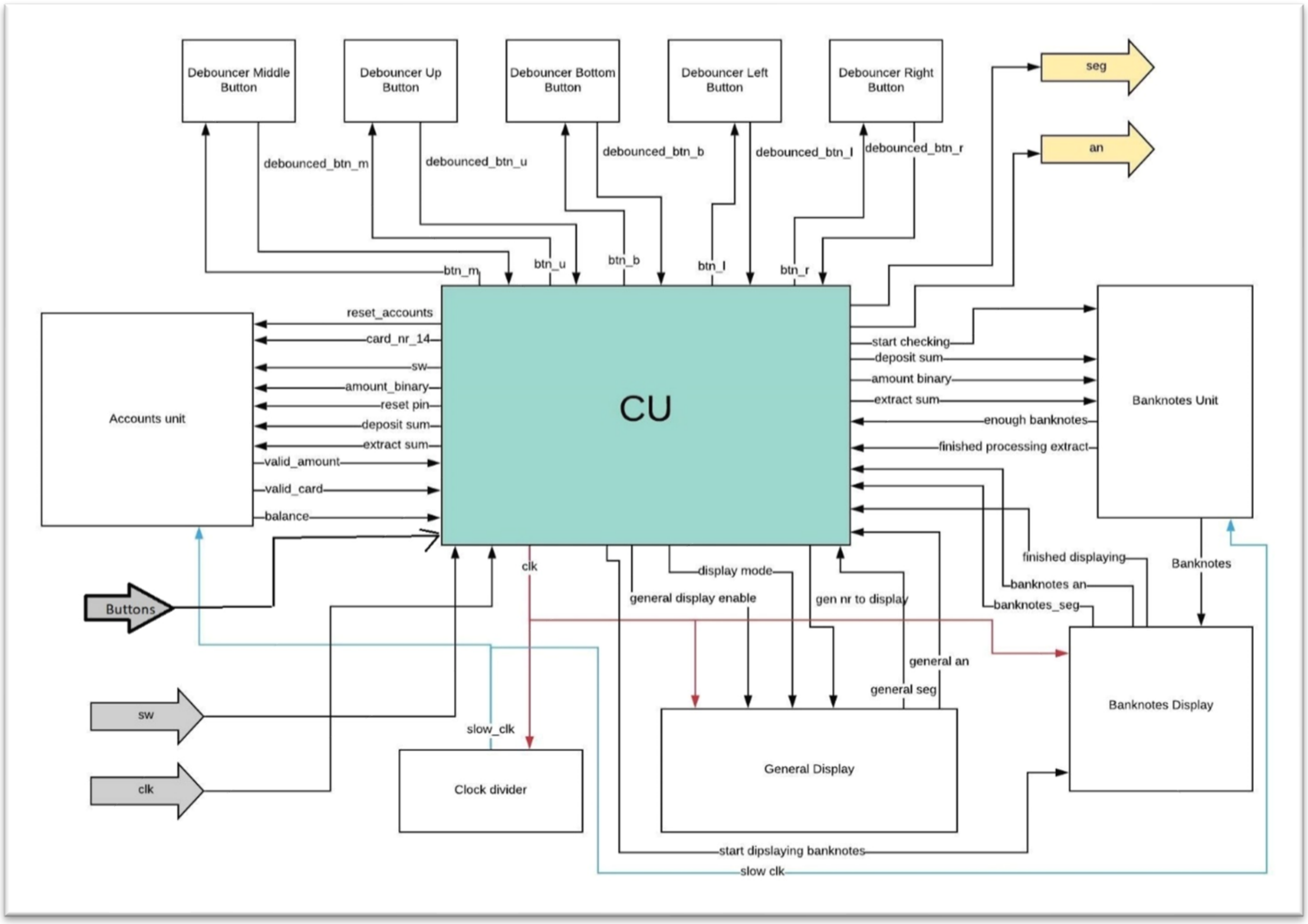
The machine has as inputs the 5 five buttons, which allow the user to select the operations that should be performed and which have different meanings depending on the state of the control system, the 16 switches, on which the user has to write the necessary information, such as the card identification number, the pin code and the amount in case he wants to deposit or withdraw a certain sum from the atm. The FPGA board also includes a single 100 MHz clock, which serves as the input clock for our system.



The outputs of the system are the 4 anodes and 7 segments, which control the display, the mean through which we can communicate with the user by displaying different messages which indicate the current state of the system, which operations are performed and whether the process was finished successfully or it failed.

Block scheme

The atm is structured in two main parts, the control unit, which controls the flow of operations, and the execution unit, including all the components, which executes the commands received from the control unit. The two units communicate with each other through various control signals, as illustrated by the block scheme.



Communication between CU and EU

The Control unit and the Execution unit communicate with each other through internal signals, which control the different components and determine the next state of the control unit.

Components	Input signals for the CU	Outputs signals from the CU
Accounts unit	• valid_amount • valid_card • balance	• slow_clk • reset_accounts • reset_pin • deposit_sum • extract_sum • sw
Banknotes unit	• enough_banknotes • finished_processing_deposit • finished_processing_extract • finished_updating_extract • banknotes_500 • banknotes_200 • banknotes_100 • banknotes_50 • banknotes_20 • banknotes_10 • banknotes_5	• slow_clk • reset_banknotes • deposit sum • extract_sum • start_checking_banknotes
General Display unit	• general_seg • general_an	• clk • general_display_enable • display_mode • general_nr_to_display
Banknotes Display unit	• banknotes_seg • banknotes_an • finished_displaying	• clk • banknotes_display_enable • start_displaying • banknotes_500 • banknotes_200 • banknotes_100 • banknotes_50 • banknotes_20 • banknotes_10 • banknotes_5
Slow clock	• slow_clk	• clk
Slow clock for converter	• slow_clk_converter	• clk
Card_nr_register	• card_nr_14	• slow_clk • load_card_nr • card_nr_14_to_load
Convert_amount_to_binary	• amount_to_load_binary	• Converter_clk • sw
Amount_register	• amount_binary	• slow_clk • load_amount • amount_to_load_binary

Command unit

The command unit is probably the most complex part of the project since it connects all the other units. It does not only take inputs from and displays outputs for the user(inputs from the 5 buttons and from the 16 switches of the board, and it displays outputs on the 7-segment displays), but it receives lots of data from the different execution units and sends meaningful commands to them.

Implementation

All in all, we have 18 states, and various outputs which depend on both the states and the inputs. Given the huge amount of inputs which need to be examined to determine the next state, we have chosen to implement the command unit as a **mealy machine**. We have two processes for that:

- One process determines the next state, based on the current state and the inputs
- The other process determines the outputs at each clock pulse, based on the current state and the inputs

For the states, we used an enumerated type, in which each state is identified by a meaningful name.

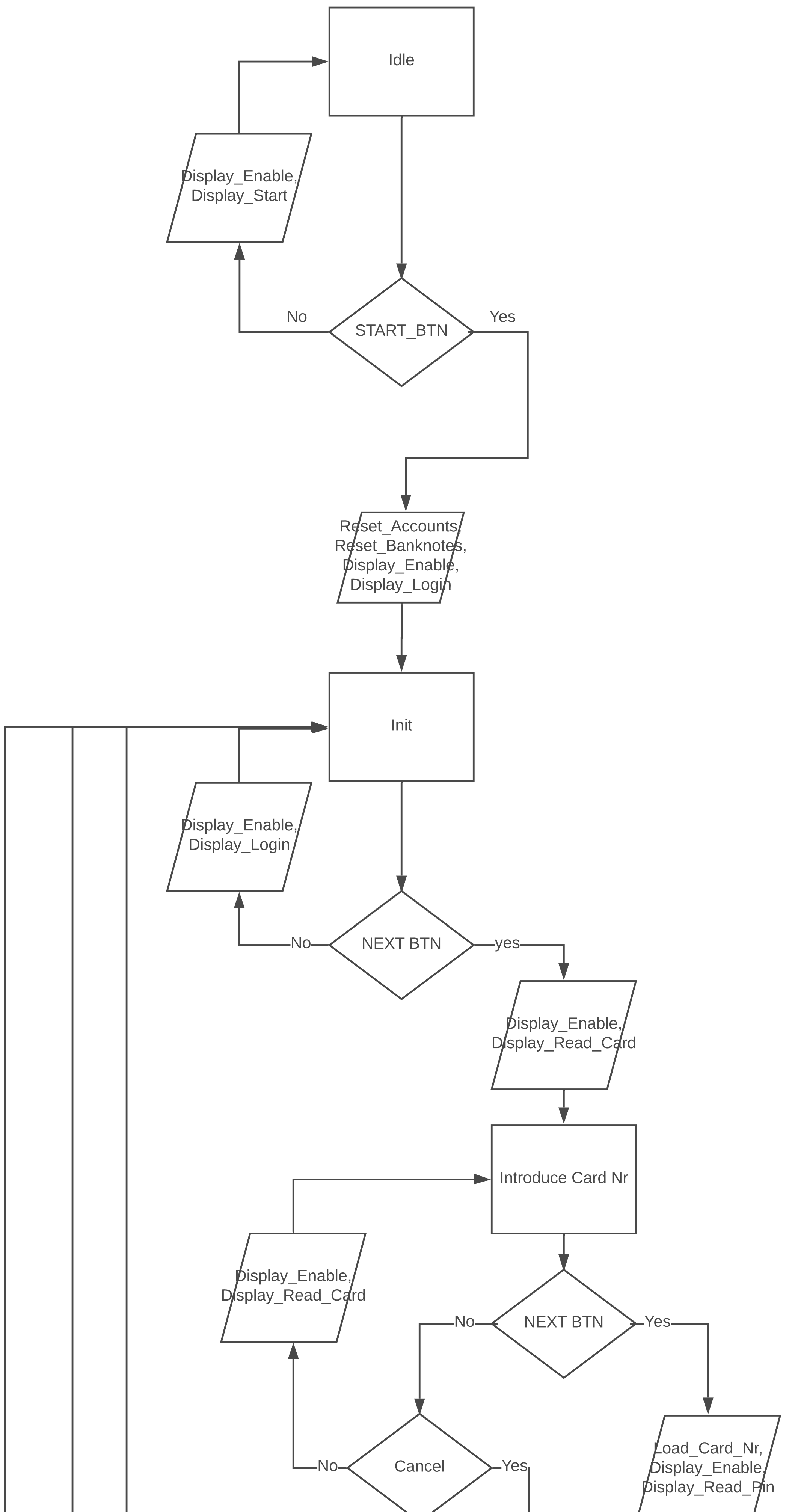
States, State Diagrams

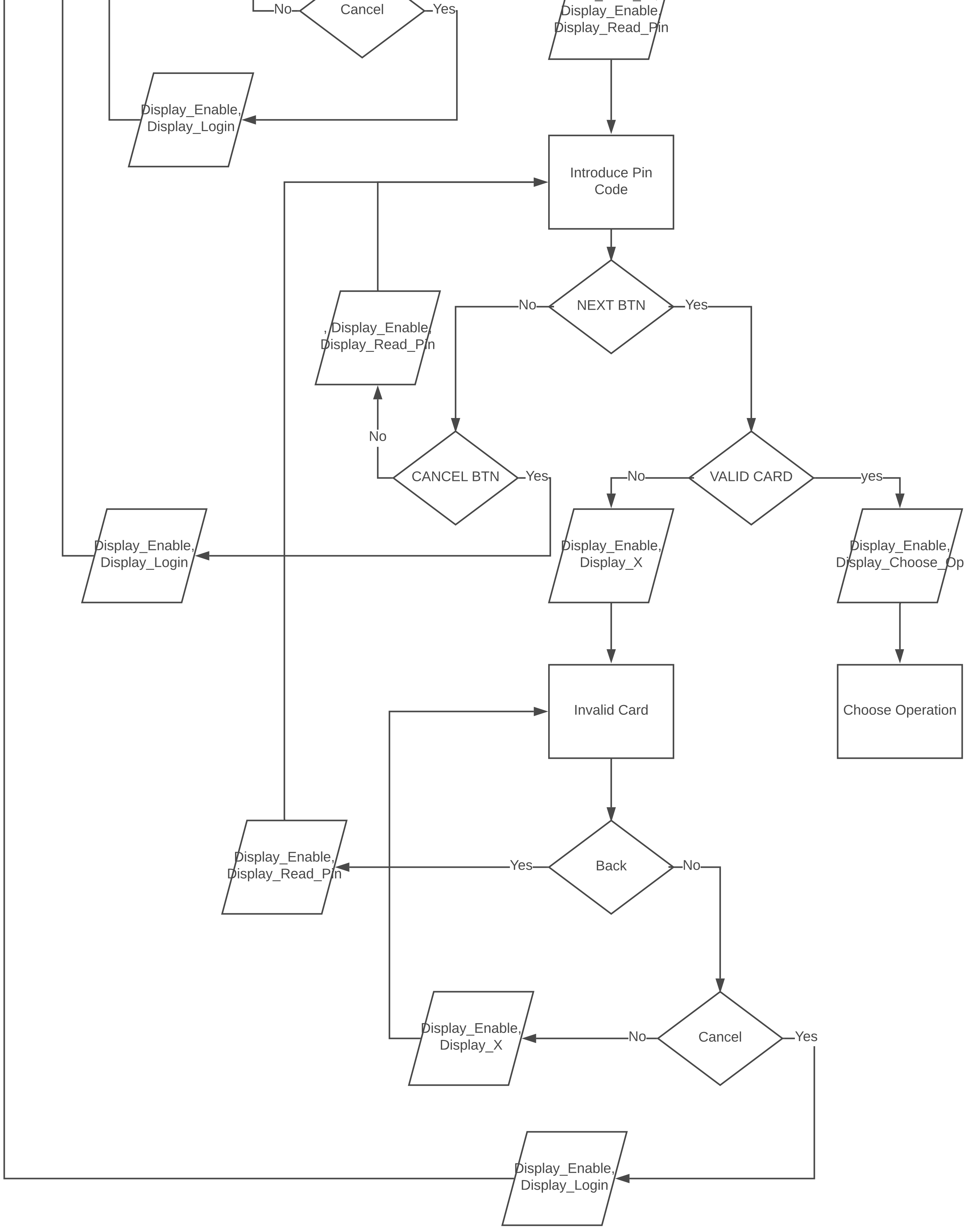
For a better understanding of the command unit, we partitioned the states of the command unit into 6 subcategories, based on the functionality to which they are linked:

- States linked to the card validation
- States linked to choosing an operation
- States linked to resetting the pin
- States linked to displaying the balance
- States linked to depositing a certain sum
- States linked to extracting a certain sum

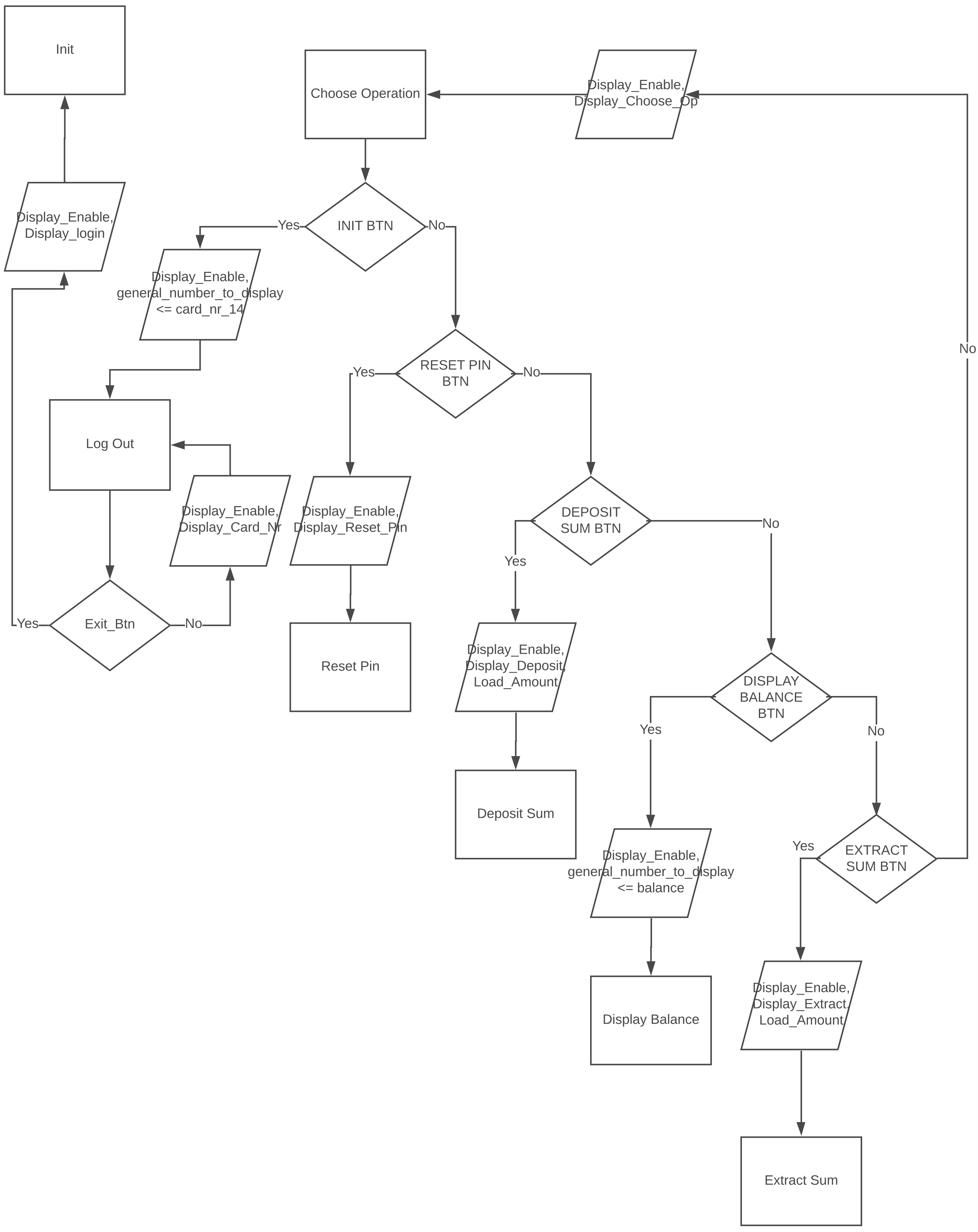
In the following pages, you will find the detailed state diagram of each subcategory.

Card validation

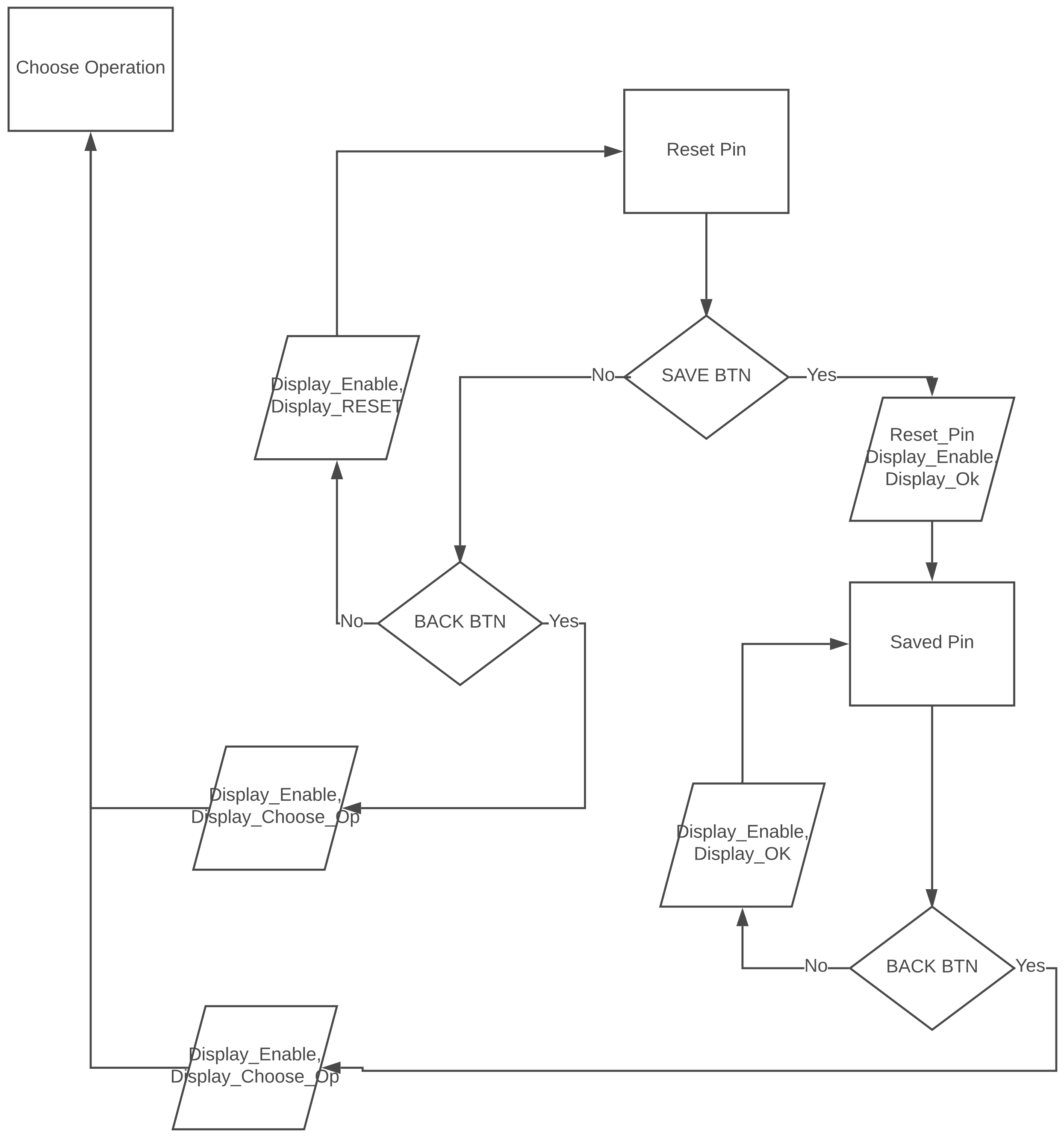



Cancel
No
Yes
Display_Enable,
Display_Read_Pin
Display_Enable,
Display_Login
Introduce Pin Code
NEXT BTN
No
Yes
, Display_Enable,
Display_Read_Pin
No
CANCEL BTN
Yes
VALID CARD
No
yes
Display_Enable,
Display_Login
Display_Enable,
Display_X
Display_Enable,
Display_Choose_Op
Invalid Card
Choose Operation
Display_Enable,
Display_Read_Pin
Yes
Back
No
Display_Enable,
Display_X
No
Cancel
Yes
Display_Enable,
Display_Login

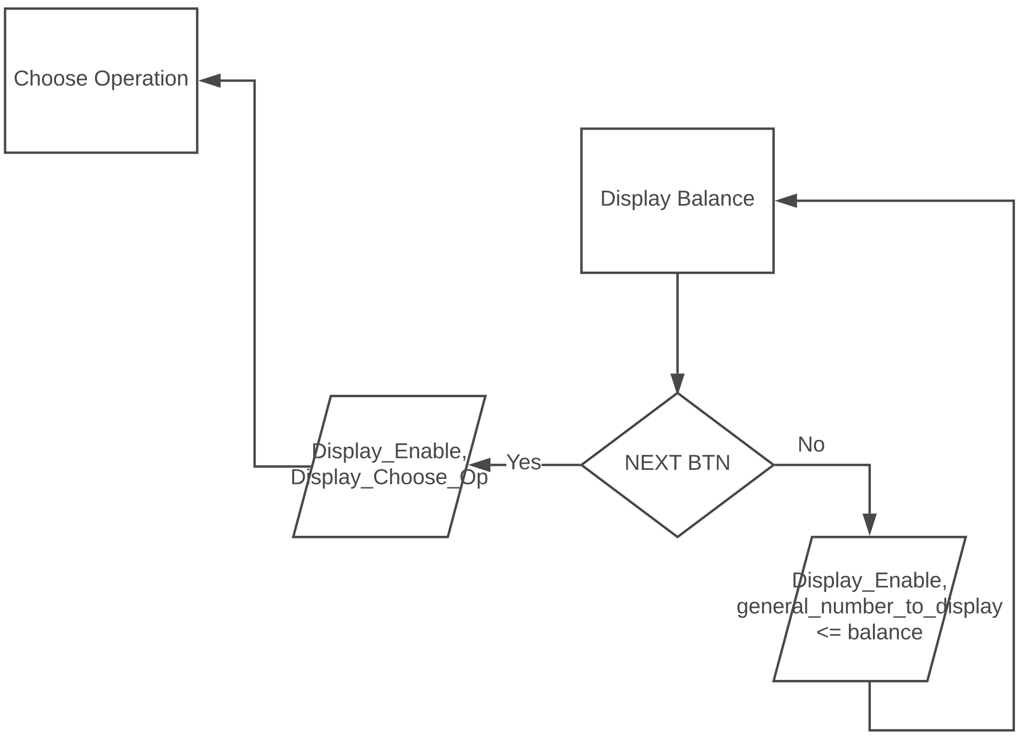
Choose Operation


Init
Choose Operation
Display_Enable, Display_Choose_Op
INIT BTN
Yes
No
Display_Enable, Display_login
Display_Enable, general_number_to_display <= card_nr_14
RESET PIN BTN
Yes
No
Log Out
Display_Enable, Display_Card_Nr
Display_Enable, Display_Reset_Pin
DEPOSIT SUM BTN
Yes
No
Exit_Btn
Yes
No
Reset Pin
Display_Enable, Display_Deposit, Load_Amount
Deposit Sum
DISPLAY BALANCE BTN
Yes
No
Display_Enable, general_number_to_display <= balance
Display Balance
EXTRACT SUM BTN
Yes
No
Display_Enable, Display_Extract, Load_Amount
Extract Sum

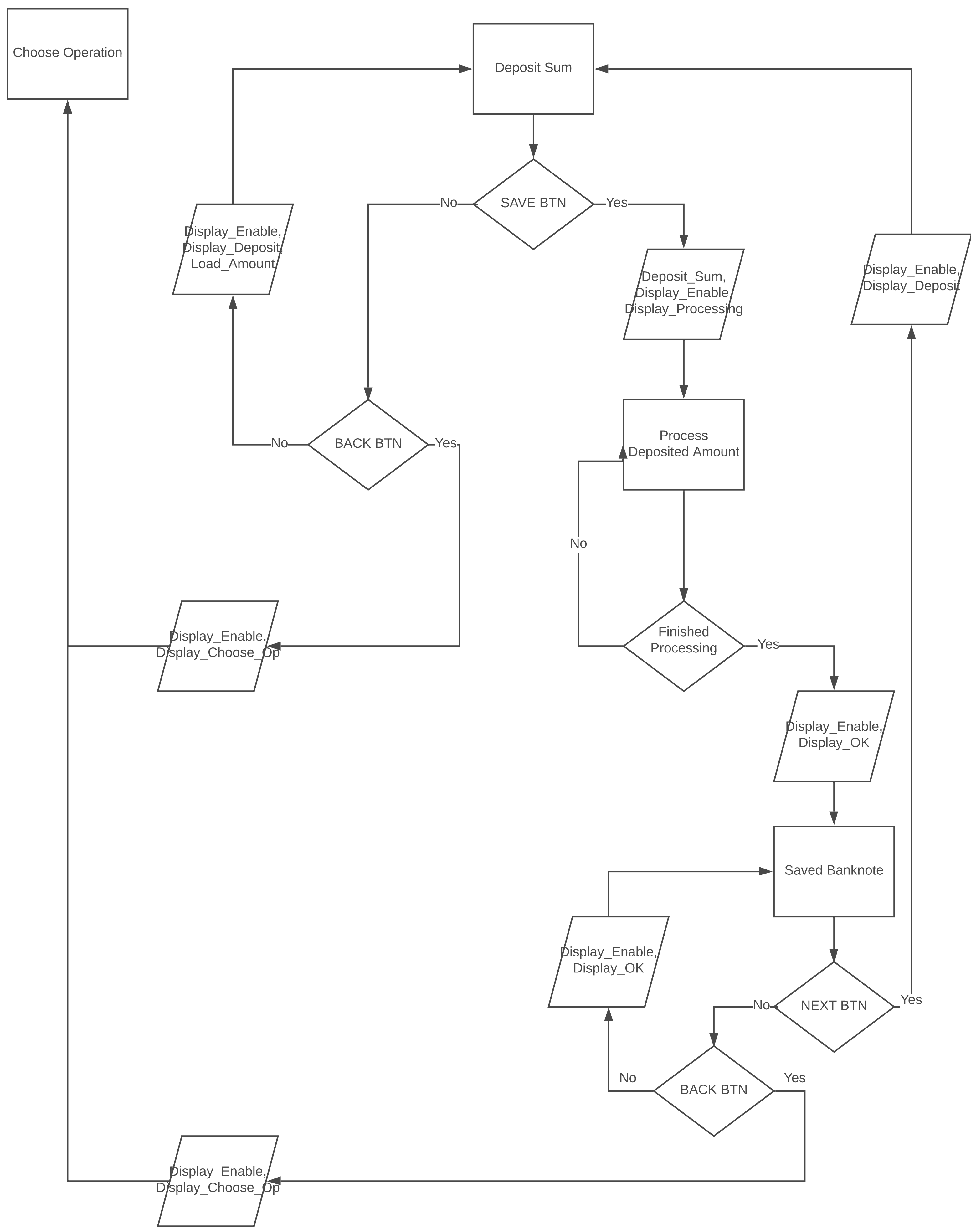
Reset Pin Operation



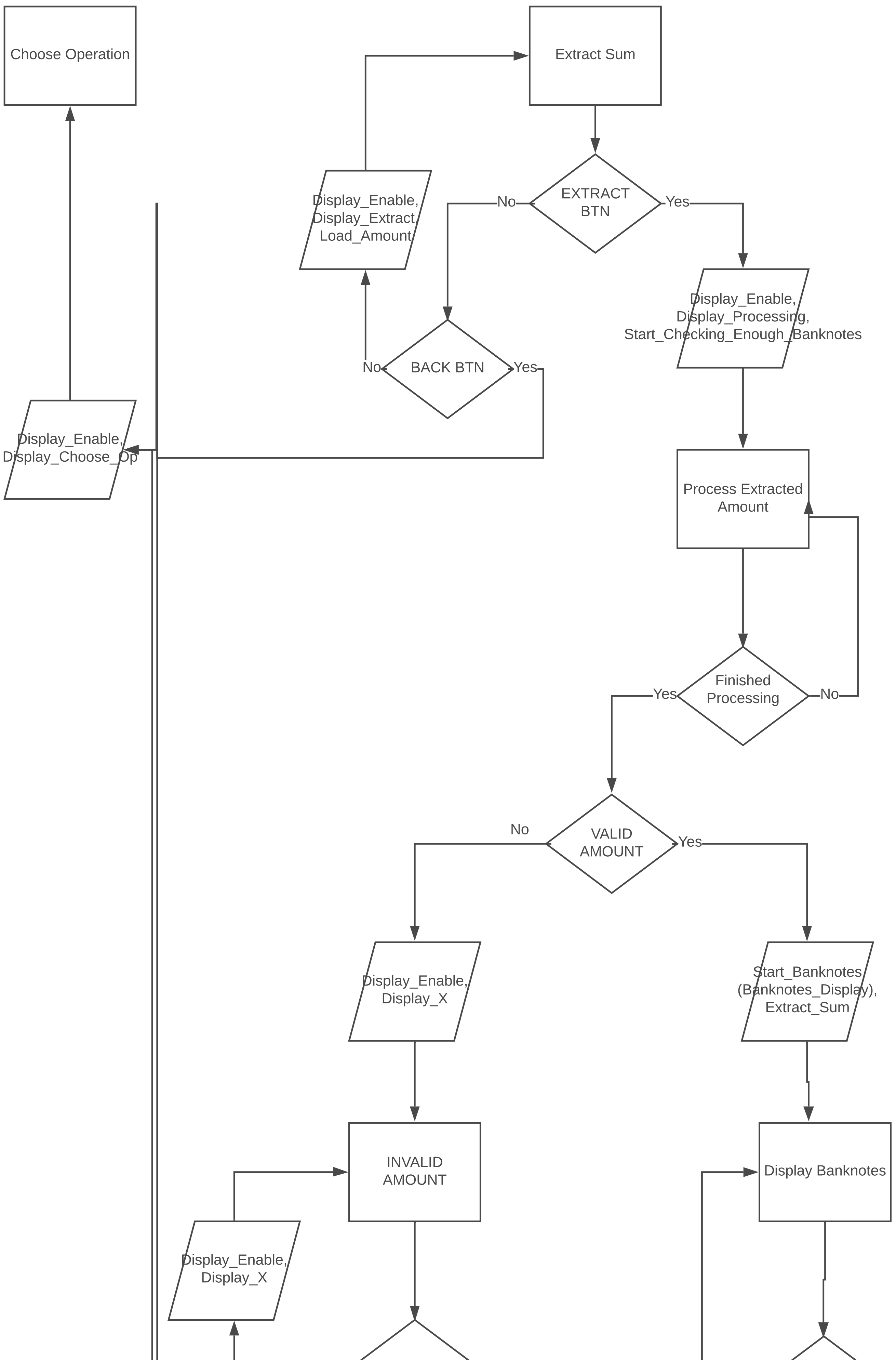
Display Balance operation


Choose Operation
Display Balance
Display_Enable, Display_Choose_Op
Yes
NEXT BTN
No
Display_Enable, general_number_to_display <= balance

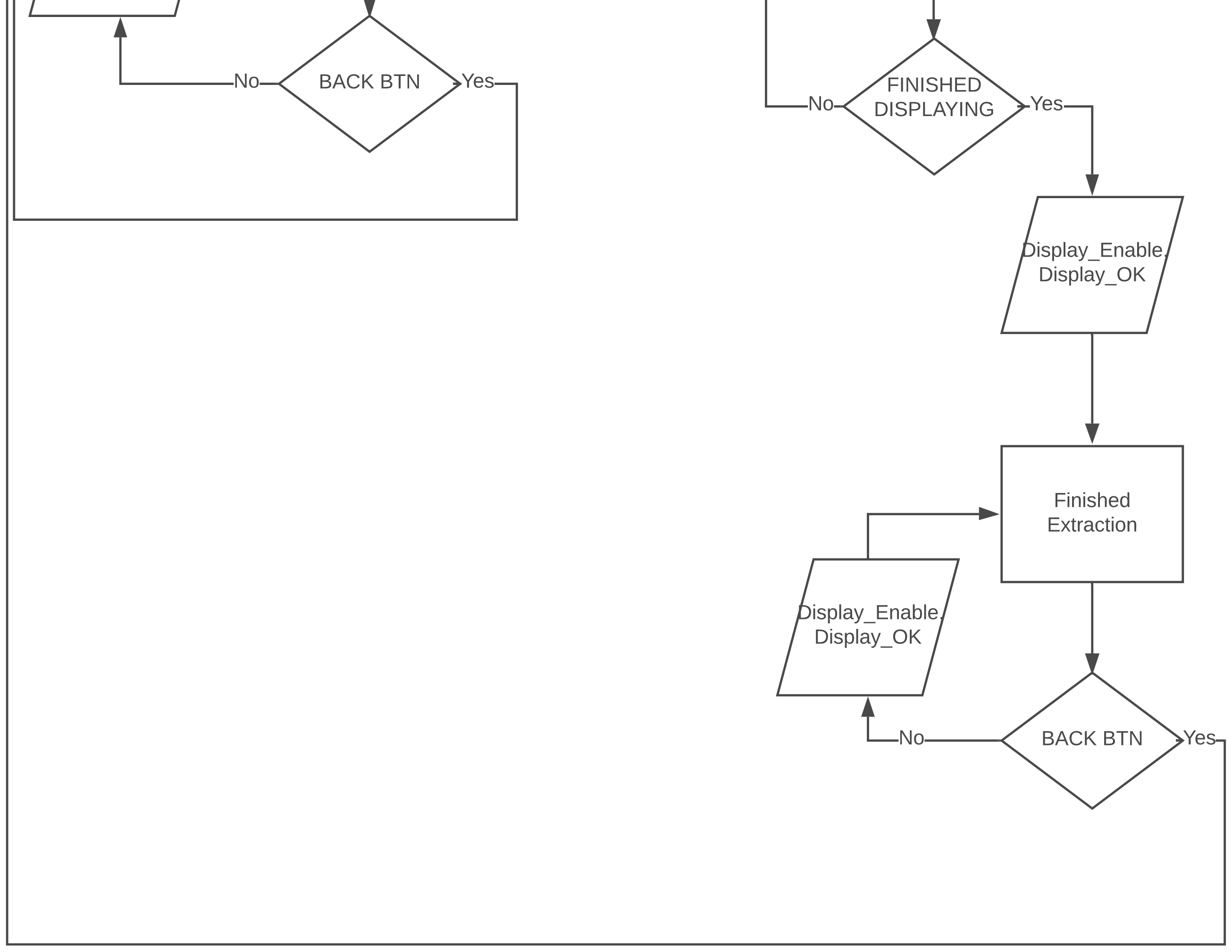
Deposit Sum Operation


Choose Operation
Deposit Sum
SAVE BTN
No
Yes
Display_Enable, Display_Deposit, Load_Amount
Deposit_Sum, Display_Enable, Display_Processing
Display_Enable, Display_Deposit
BACK BTN
No
Yes
Process Deposited Amount
No
Finished Processing
Yes
Display_Enable, Display_Choose_Op
Display_Enable, Display_OK
Saved Banknote
Display_Enable, Display_OK
NEXT BTN
No
Yes
BACK BTN
No
Yes
Display_Enable, Display_Choose_Op

Extract Sum Operation



Choose Operation
Extract Sum
Display_Enable, Display_Extract, Load_Amount
EXTRACT BTN
No
Yes
Display_Enable, Display_Processing, Start_Checking_Enough_Banknotes
BACK BTN
No
Yes
Display_Enable, Display_Choose_Op
Process Extracted Amount
Finished Processing
Yes
No
VALID AMOUNT
No
Yes
Display_Enable, Display_X
Start_Banknotes (Banknotes_Display), Extract_Sum
INVALID AMOUNT
Display Banknotes
Display_Enable, Display_X


BACK BTN
No
Yes
FINISHED DISPLAYING
No
Yes
Display_Enable, Display_OK
Finished Extraction
Display_Enable, Display_OK
BACK BTN
No
Yes

Execution units

Elementary components

While designing our project, we aimed to describe the main components in a structural way, as much as possible. To reach this goal, first of all we have defined some elementary components, with a general functionality, and then we built all the complex units on the top of these elementary components.

The elementary components used to implement the execution units are the following:

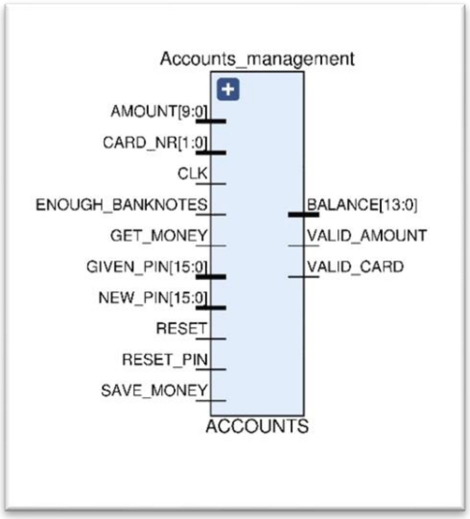
- **RAM memory**: used to store the accounts information (the pin code and the balance) and the different types of banknotes.
- **Register**: mainly used to store the data that is currently being processed
    1. General register
    2. Shift register
    3. Register with a 2 to 1 multiplexer to select which data to load
- **Adder-Subtractors**: it has two modes, the adding mode is represented by '0', while the subtracting mode is represented by '1'
- **Comparators**: used to validate the pin codes and the extracted amounts
- **Priority encoder:** used to get the address in memory of the given banknote
- **Counters:**
    1. With holding functionality
    2. With loading functionality
- **Frequency divider**: generates a slower clock frequency
- **MUX**: to select data. Versions: mux_2:1, mux_4 :1, mux_7:1, mux_16:1, and dmux_7:1
- **DMUX**: for connecting a signal to the corresponding input of other units
- **JK-Flip-Flop**
- **D-Flip-Flop**

Background logic

Accounts unit

This unit is responsible for storing, updating and verifying all the data related to the accounts. Each account is identified by the card number, and to each account there is associated a pin code and a balance.

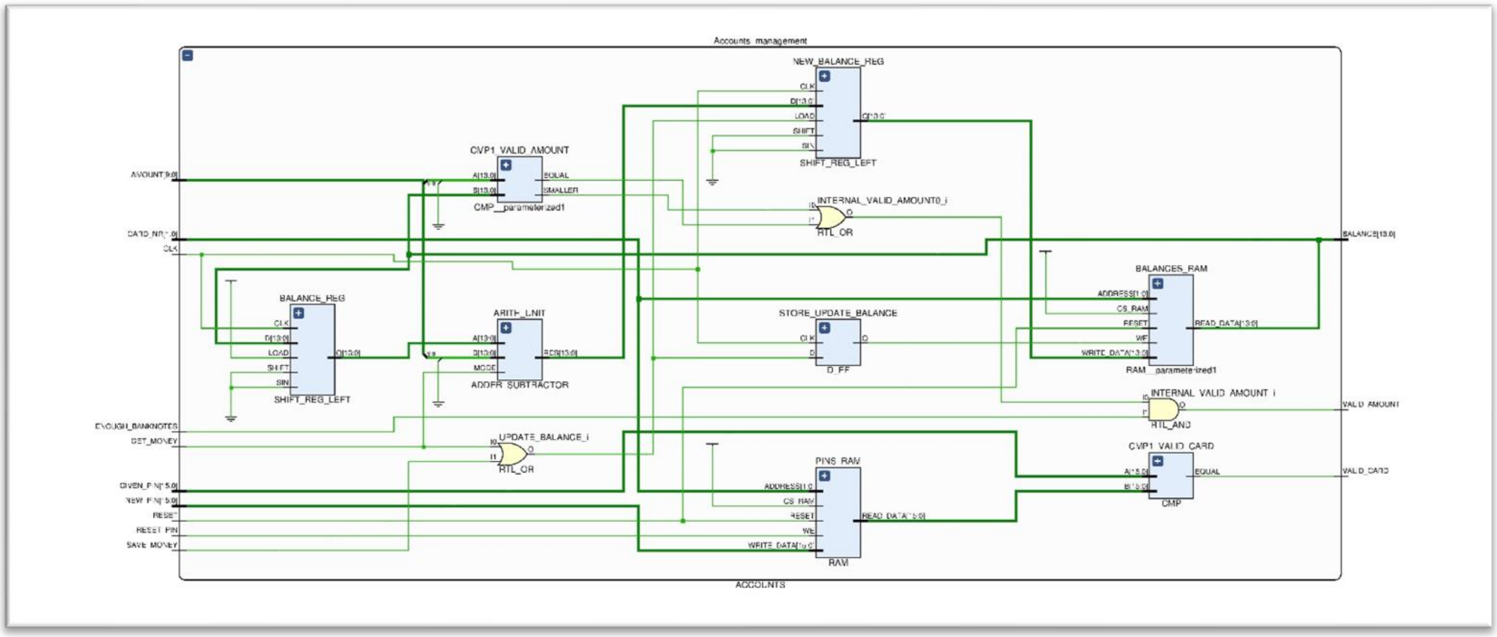
Functionalities



- When the user wants to log-in, the accounts unit decides whether the pin code provided for a certain card number corresponds to the previously set pin code(see the valid_card signal)
- When the user wants to extract a certain amount of money from his card, the accounts unit decides whether the amounts is a valid one, or the operation cannot be performed(see the valid_amount signal)
- When the user extracts or deposits an amount of money, the accounts unit updates the balance on the corresponding card
- When the user wants to reset the pin code associated to their card, the accounts unit updates the pin code
- When the user wants to check the balance available on his card, the accounts unit provides this information

Implementation

- For storing the pins and the balances, we used two separate RAM memories, with the card number used as the address. When an update is needed, we enable writing to these RAMs



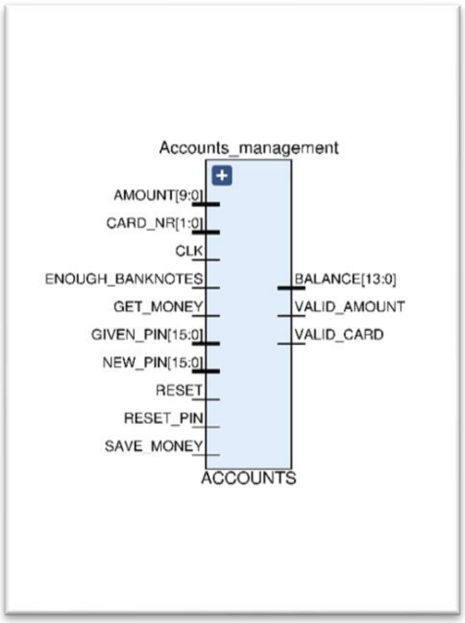
- The card validation and the amount validation(in case of extraction) rely on comparators(and it's also taken into account, whether there are enough banknotes to give out the wanted amount of money[1])
- The new balance is calculated using an adder-subtractor. To avoid infinite loops at the addition/subtraction, this part is a sequential one: the new balance is loaded to a register, and from the register to the RAM
- The balance on an account is read constantly from the RAM memory

Banknotes Unit

This unit performs the banknotes related operations, such as adding or withdrawing banknotes to and from the ATM. It also examines the number of banknotes in the teller machine and decides whether an extraction would be possible with the currently available banknotes. There are in total seven types of banknotes considered, with values 500, 200, 100, 50, 20, 10 and the smallest 5.

Functionalities

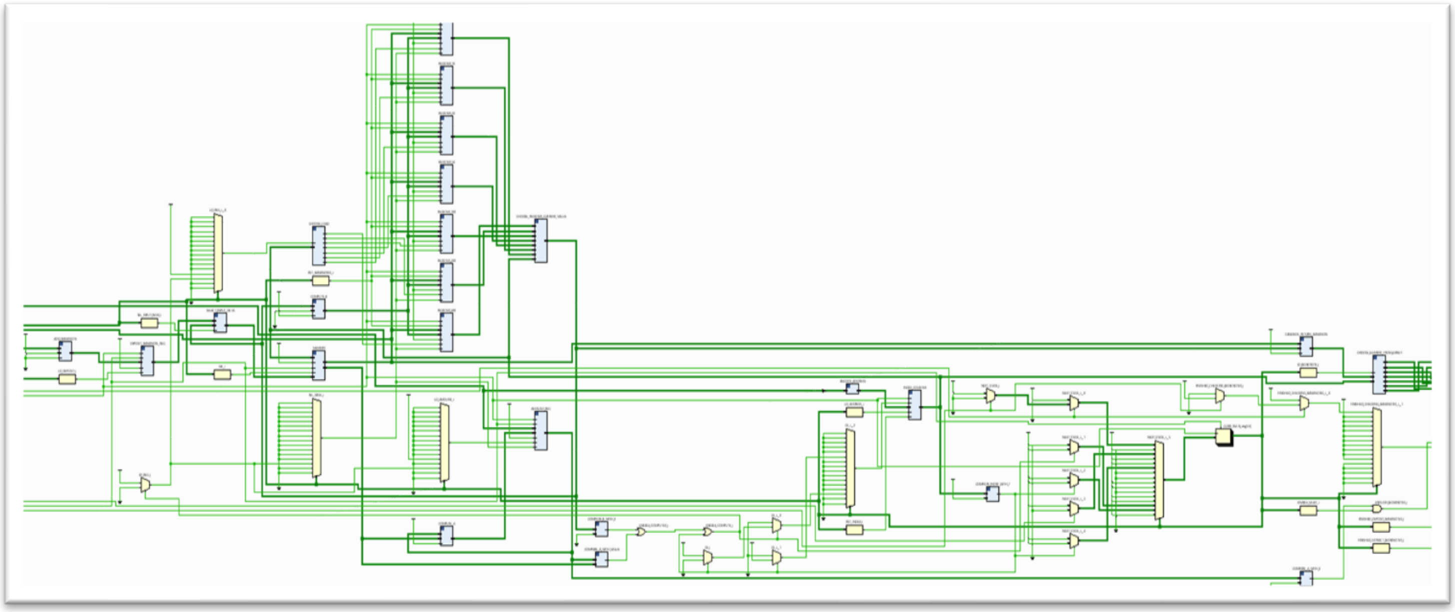
- When the user wants to deposit money, he has to introduce the banknotes one by one. This unit recognizes the type of the introduced banknote and updates the memory, used for storing the banknotes, with the current, incremented, number of the given banknote.
- When the user chooses to withdraw money, the unit receives the amount introduced by the user, and it checks whether the amount is valid, meaning that it can be output with the available banknotes, in which case it transmits a signal to the accounts unit that the withdrawal is possible. In the same time, it generates the banknotes used for the withdrawal and transmits them to the display unit. When it receives access from the control unit, it performs the extraction on the level of the banknotes, such that it updates the memory with the decremented number of each type withdrawn banknote.



Implementation

- This unit is controlled by a Moore state - machine, because the output doesn't depend on the inputs. There is a process which generates the next state and the outputs, and another process in which the current state is updated. The states are defined by an enumerated type.
- The numbers and values of each type of banknote are stored in a RAM memory with two read ports, one for reading the number, the other for reading the value of the banknote at a given address, and one write port, used for updating the number of banknotes in the atm. In the beginning, when the atm is started, the banknotes memory is loaded with an initial amount of each banknote.

[1] See the banknotes unit for more information



- With a counter, we go through the seven types of banknotes, from greatest to lowest value, and reduce the amount, if possible, by the value of the next banknote that is available. In case the amount becomes 0, meaning that it can be constructed with the current banknotes in the atm, then an enough_banknotes signal is generated that validates the withdrawal.
- If a deposit operation is requested, the unit computes the address of the given banknote in memory, with a priority encoder, and loads that address to the counter. The value of the counter represents the address with which we access the ram memory.
- In case of a deposit operation, after updating the memory of banknotes, a finished_deposit_banknotes signal is generated.
- There are seven registers, which hold each banknote's current number, and they are loaded with their new value after each subtraction, and one register which holds the current banknote's number that we are working with.
- A multiplexer is used to choose, according to the value of the counter, which register should be considered. A demultiplexer selects, based on the value of the counter, which register should be loaded next.
- Another register holds the current value of the amount and it is loaded with its new value each time it is modified.
- The subtractions are performed using a generic adder - subtractor.
- The algorithm to check if there are enough banknotes to perform the extraction consists of repeatedly subtracting the current banknote's value from the amount until the number of banknotes reduces to 0 or the amount is less than the value of the banknote, in which case the counter is enabled and the next banknote is loaded in the register. When the counter reaches seven, a finished_checking_banknotes signal is generated.
- The conditions to perform the computation, that the amount to be greater than or equal with the current value of the banknote, the current number of the banknote to be greater than 0, and the index of the counter to be less than 7, are generated with comparators. The enough_banknotes signal is also computed with a comparator, by comparing the current amount to 0.
- The numbers of the seven types of banknotes which need to be transmitted to the display unit are computed by subtracting from the number stored in the memory the current value of the seven registers.
- If the unit receives an enable signal to withdraw the banknotes, then it goes through each banknote again with the counter, and updates its number in the memory with the value of the registers. This time, when the counter reaches seven, a finished_extract_banknotes signal is generated.

I/O processing

The number systems converters

Functionality

While designing our device, we tried to provide not only an operable system, but also a pleasant user experience. Given that in the background all the arithmetic operations required that the numbers are stored in binary, but at the same time for the user the most comfortable option is to write the input data in BCD format, we had to implement a BCD to binary converter for processing input data and a binary to bcd converter, for processing output data.

Implementation

The implementation relies on the well-known Shift-and-Add-3 or Double Dabble algorithm[2], for the conversion from binary to BCD, and its reverse for BCD to binary.

Given that all of our numbers are integers from the range [0, 9999], the Shift-and-Add-3 algorithm can be easily adjusted to a hardware description language, as follows:

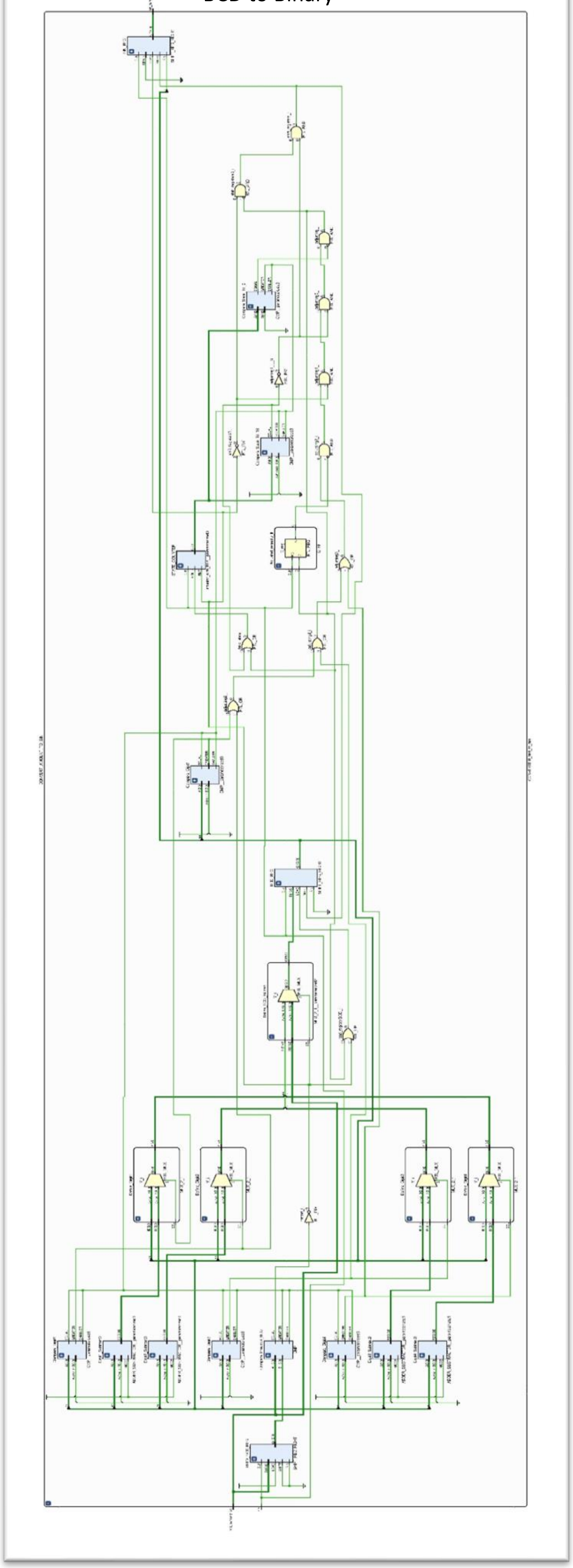
1. When the input binary number changes, load it into the corresponding register(binary-register, 16 bits wide, with shift functionality)
2. Set the ones, tens, hundreds and thousands of the BCD number to 0(in the BCD -register, 16 bits wide, with shift functionality)
3. Set the state counter to 0(5 bits wide counter)
4. If any of the BCD digits is greater than 4, and at the last step a shift operation was performed(this can be decided using the "adjusted recently" flip-flop), than at the next clock pulse, add 3 to those digits(this step requires 1 comparator, 1 full-adder and 1 multiplexer/digit), and hold the value on the state counter
5. Otherwise, at the next clock pulse shift both the binary and the BCD registers to the left by 1 position, such that the most significant bit of the binary number is shifted into the BCD number, and increment the state counter by 1
6. If the state counter is 16, then stop at this point. Otherwise, go to step 4 again

The reverse algorithm is very similar to this one, but the shift-left registers need to be replaced with shift-right registers, and the digits don't need to be compared to 4, but to 7.

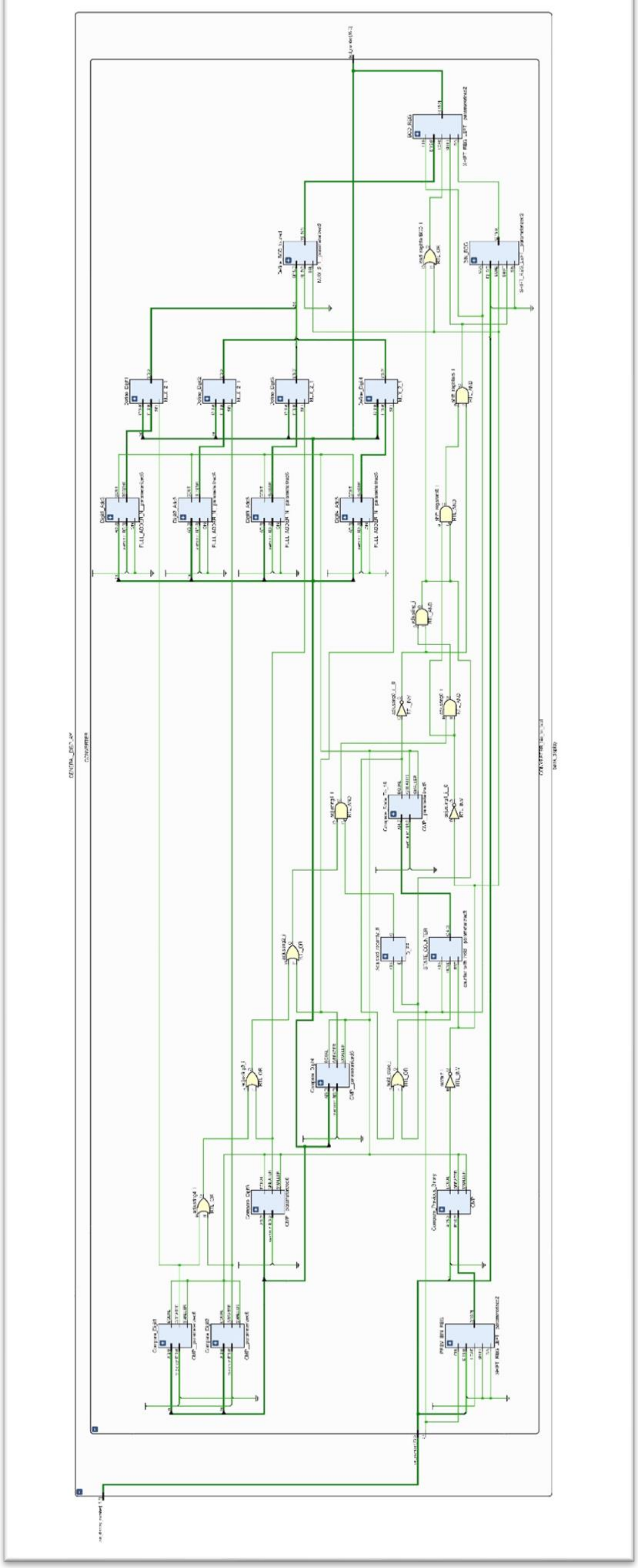
Generally, we can say that a conversion finishes after least 17 and at most 31 clock pulses. This is the explanation for using a faster clock for the converters than for the units that will use the outputs of the converters.

[2] See the textbook from the 1st semester: Logic Design Problems, written by Octavian Creț and Lucia Văcariu, page 22

BCD to Binary



Binary to BCD



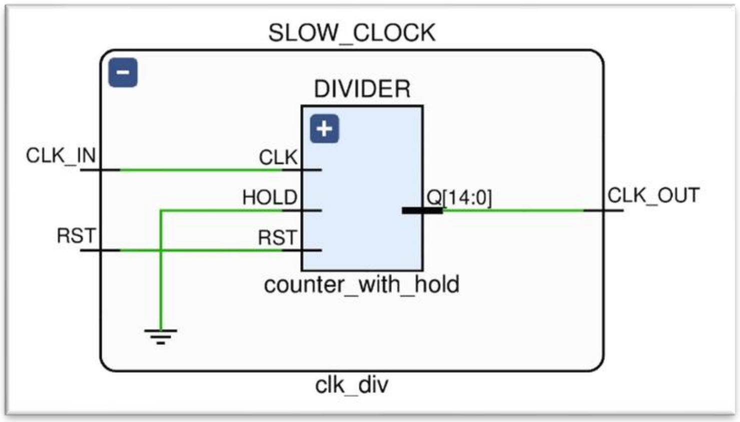
The frequency dividers

Functionality

The default clock signal available on our FPGA board has a frequency of 100mHz, but at this frequency most of the input/output operations didn't work properly. So, we had to create a frequency divider to define some slower clock signals.

Implementation

This frequency divider is a generic component, which takes as an input clk_in signal and divides it by 2^n, where n is a generic parameter, thus generating a new clock signal: clk_out. The implementation is simply based on an n-bit counter, and the MSB(most significant bit) of the counter's output is used as the new clock signal, as the following block scheme shows:



In our project we used several frequency dividers, since different components work on different frequencies.

- The command unit and most of the execution units(the accounts unit, banknotes unit, the debouncers and the display unit) work at a frequency of ~3kHz(which is 2^15 times slower than the original one)
- The numbering system converters work faster than the previously mentioned units, in order to guarantee that they provide a valid result of the conversion within one clock cycle of the other units. Their clocks frequency is 2^8 times slower than the original clock frequency, which means a frequency of ~195kHz
- The display_array_of_banknotes unit is slowed down even more, to a frequency which is 2^25 times smaller than the original one, such that each extracted banknotes value appears for a short, but still reasonable time period on the display, while iterating on the array of extracted banknotes

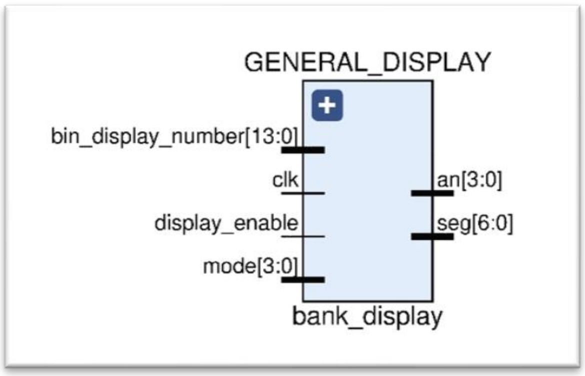
The display

The 7-segment displays on the FPGA board represent the only channel through which we can provide information for the user. In order to obtain a pleasant user experience, we don't only display as a response to the users request to check the balance on a card, to extract a certain amount from the ATM or to get a receipt when leaving, but we also provide useful information about the current state of the operation, required information, errors, etc.

Functionalities

Taking this into account, we designed the display unit to have multiple functionalities

1. Displaying a number, given as an input data in binary format
2. Displaying a message, depending on an external command which defines the required message(see mode signal)



Implementation

Depending on the states(of the command unit and the display_array_of_banknotes unit) and the inputs, there are 12 different messages(and the 13th is displaying a number). In order to optimize the number of input signals, we encoded these messages on 4 bits, as shown in the following table(Remark: for making the state diagrams easier to understand, we didn't mark the commands by the codes there, but we gave to each command a meaningful name, as shown in the 3rd column):

Code	Message to display	Command name
0000	Display the number given as input	-[3]
0001	OK	Display_OK
0010	NOT	Display_X
0011	DEP	Display_Deposit
0100	GET	Display_Extract
0101	RST	Display_Reset_Pin
0110	PIN	Display_Read_Pin
0111	STRT	Display_Start
1000	C OP	Display_Choose_Op
1001	LOG	Display_Login
1010	PROC[4]	Display_Processing
1011	----	Display_Line
1100	C NR	Display_Read_Card
1101	xxxx[5]	-
1110	xxxx	-
1111	xxxx	-

Components

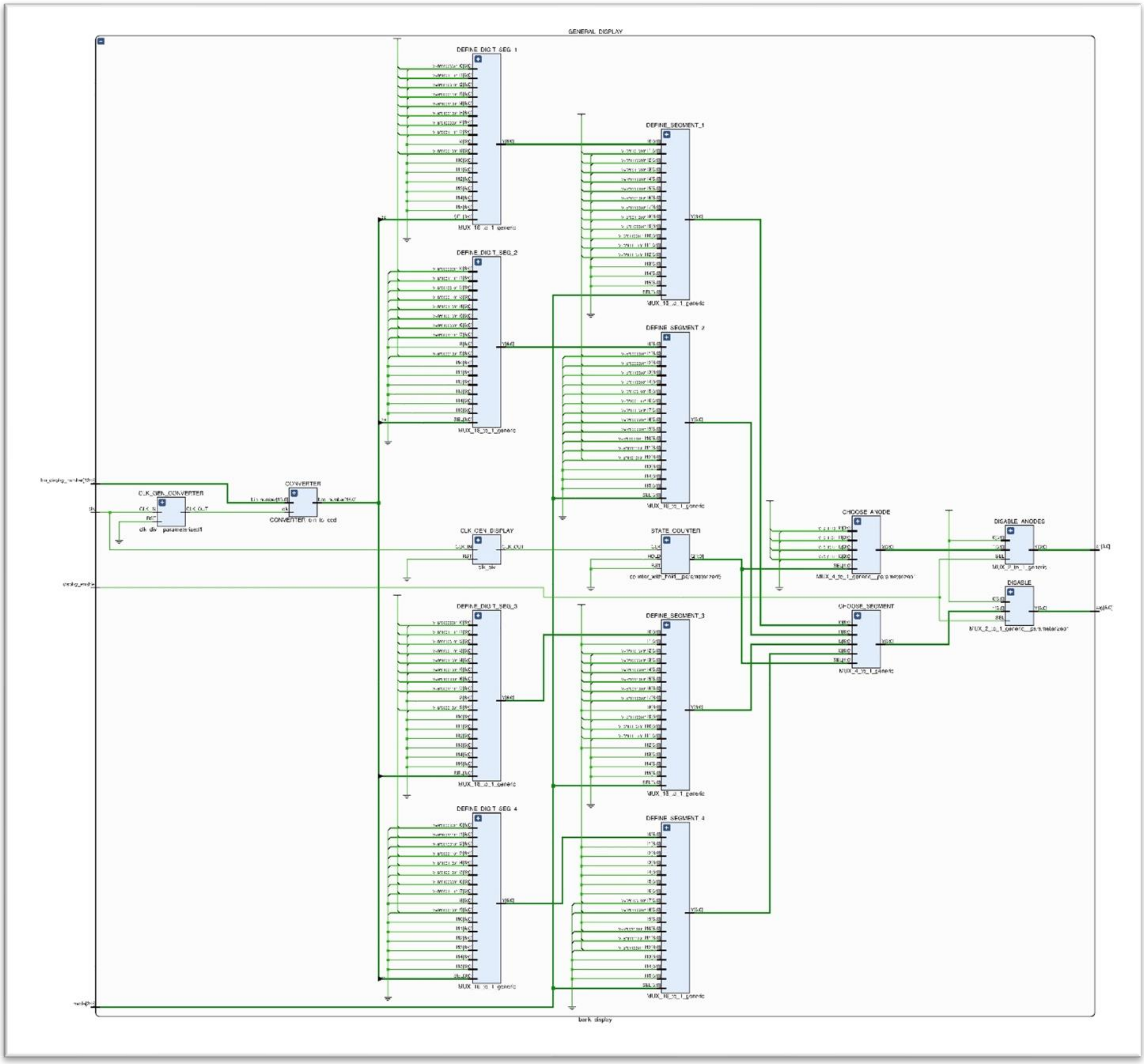
- We used a binary to BCD converter to convert the input number(bin_display_number) to a 4-digit BCD number(number)
- To convert a digits BCD-code to a 7-segment code for the display, we used 16-to-1 multiplexers, one for each digit, with the BCD-code as a select signal.

[3] When no other command is activated, but the display is enabled, then the display number functionality is considered to be active

[4] Remark: this message doesn't appear in the final project, we have used it only for debugging purposes. But given that removing it wouldn't have reduced the number of bits required for the encoding, we didn't re-encode the messages

[5] Don't care

- Similarly, to switch between the messages, we used again 16-to-1 multiplexers, one for each letter, with the command code as a select signal. The letters are stored as constants
- Given the architecture of the 7-segment displays, we had to use a (2 bits wide) counter to switch between the four 7-segment displays
- We defined the anodes and cathodes based on the current output of the counter, this time with a 4-to-1 multiplexer
- In addition to this, we used a frequency divider to obtain an optimal frequency for the display, and one separate frequency divider for the binary to BCD converter[6].



6 See the frequency divider section for more information

Banknotes Display Unit

This unit controls the displaying of the banknotes when the user wants to withdraw money from the atm.

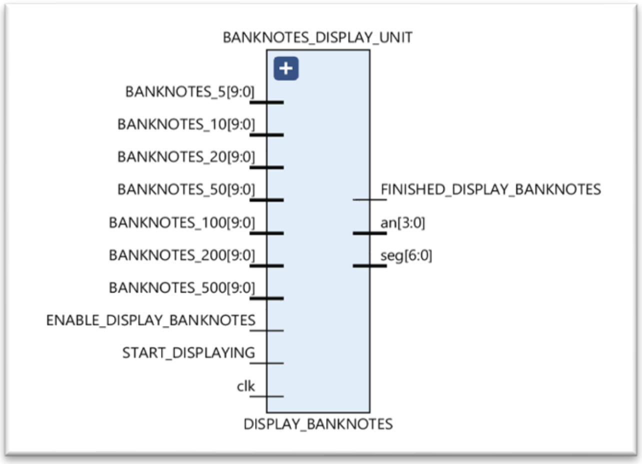
Functionalities

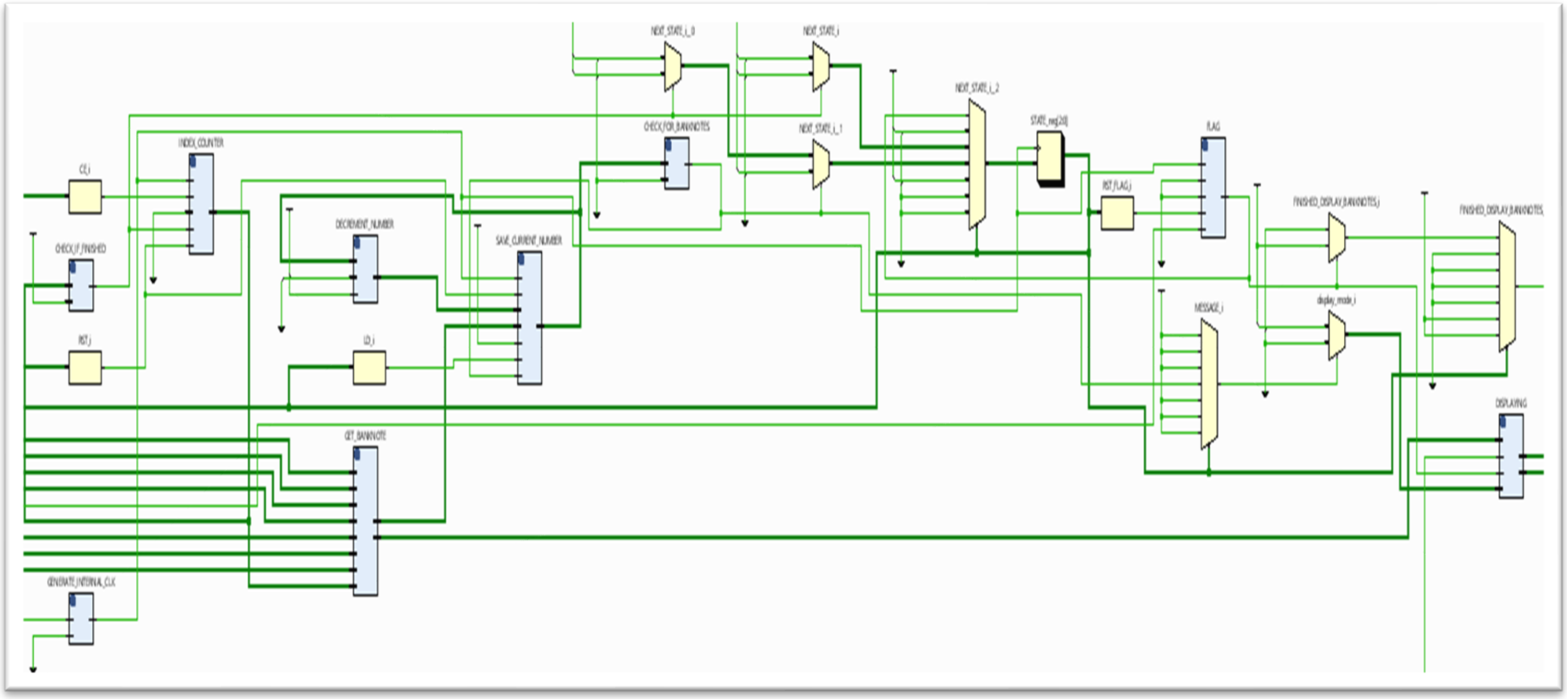
- This unit is linked to the banknotes unit by the signals which indicate which and how many of the banknotes were used to extract the given amount.
- When it receives a start_displaying_signal, the unit displays each type of banknote, one after the other, as many times as its number indicates, thus representing the total amount that the user wanted to withdraw.

Implementation

This unit controls a separate bank display unit, different from the general display module form the command unit, which only works in the extract sum state, if the amount is validated.

The unit is controlled by a Moore state machine, providing that the outputs only depend on the current state, and the states are defined as an enumerated type.





Components

- Separate bank display unit, as mentioned above, is used to display the banknotes.
- In order to show the banknotes for a longer time on the displays, we used a frequency divider inside this unit, which slows down the clock pulse received from the command unit by 2^{25} times. However, the bank display unit still receives the original clock signal form the control unit.
- A counter is used to go through each type of banknote and display it. Its counting function is enabled when the number of the current banknote is 0, which indicates to go to the next available banknote.
- A multiplexer is used to select the current banknote according to the value of the counter.
- The number of the currently processed banknote is stored in a register, which is loaded with the new value of the banknote whenever a subtraction is performed or the counter goes to the next banknote.
- The number of the current banknote gets reduced after each time it was displayed with an adder-subtractor.
- We check with a comparator whether the current number of the banknote is 0 or not. Another comparator indicates us when the index of the counter reached seven and generates the ouput finished_displaying signal.

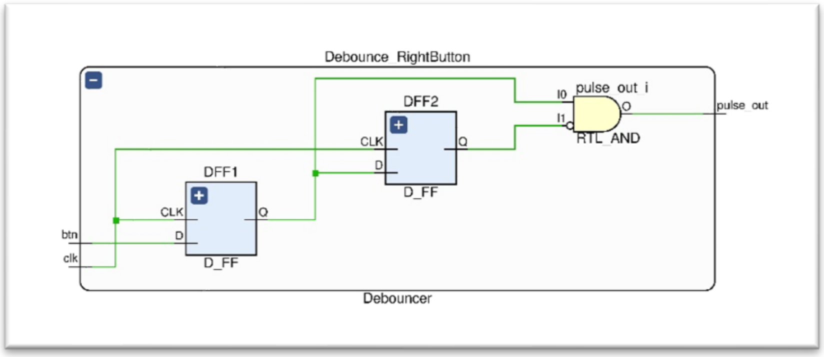
Debouncers

An important element of the user interface are the buttons on the FPGA board. However, without some additional debouncers, these signals can't be used in the command unit for defining the next state, for two reasons:

- Several glitches may appear
- The signal may be active high for a much longer period than the clock pulse, thus making the system react to the same button push several times

Functionality

Given that we had already slowed down the frequency of our clock, we didn't have to worry about the glitches, but we had to generate a debounced signal for each button which is active high only for one clock pulse, even if the user holds the button pushed down for a longer period of time



Implementation

We solved this problem using two D-Flip-Flops, as it can be seen on the block scheme.

III. Instructions for use and maintenance

Instructions

Turn-in

To “turn in” the ATM after the FPGA has been programmed(at this point, the “STRT” message appears on the display), we must push the middle button(=start button). After the device being turned-in, the “LOG” message appears, and the user may freely log in.

Log-In

The log-in is a two-step operation. After deciding to log-in, the user needs to push the right button (=next button). Now, the “C NR” message appears on the display, and he has to introduce the card, by providing its number in BCD format on the switches. When he is finished, he needs to push the right button (=next button) again. If he changes his mind about logging in, he may push the left button (=cancel button), and this way he can return to the initial state.

If he has chosen to go to the next step, then he will see the “PIN” message on the display, and as this message suggests, he should introduce his pin code on the switches, in BCD format. When finished, he has to push the right button (= next button) again. Otherwise, he may choose to return to the initial state by pushing the left button (=cancel button).

If he has pushed the right button, then there are two possible cases:

1. If the data that he provided is correct, then he gains access to the main menu, and the “C OP” message appears
2. Otherwise, the “NOT” message appears. He may go back and reintroduce the pin code by pushing the right button, or he may cancel the log-in by pushing the left button

Main Menu

After the Log - in was successfully completed, the user arrives to the main menu, where he can choose which operation to perform. At this moment, the “C OP” message appears on the display, and the user can advance by pushing one of the buttons. The 5 operations available for the user and the buttons corresponding to them are:

- Reset the Pin code (upper button)
- Display the Balance (bottom button)
- Deposit a sum of money (right button)
- Extract a sum of money (left button)
- Log-out (middle button)

Reset Pin

The user can reset his pin by pushing the upper button (= reset pin button), and the “RST” message appears on the display. Next, he writes his new pin on the switches in BCD format. If he does not want to continue with the operation, he can go back to the main menu by pushing the left button (= back button). If he wants to save the new pin he needs to push the right button (= save button). Once the pin was changed an “OK” message appears on the display and the user can return to the main menu by pushing the left button (= back button).

Display Balance

From the main menu, by pushing the bottom button he may check the balance currently available on his card. The balance will be displayed on the 7-segment displays. He may return to the main menu by pushing the right button(=next button).

Deposit Sum

Another option from the main menu is to deposit a certain amount of money. To do that, the user has to push the right button (=deposit sum button), and then the “DEP” message appears on the display. Similarly to the previous operations, he may always go back to the menu by pushing the left button. Otherwise, the user has to introduce the banknotes one by one, by writing each banknote’s value on the switches, in BCD format. When the value of the current banknote is written on the switches, the user has to push the right button (=save button) again. The message “OK” should appear, with the meaning that the banknote was processed and accepted by the ATM. After that, the user can choose to deposit the next banknote, by pushing the right button (=next button), or to finish this transaction and go back to the main menu (by pushing the left button).

Remark that the ATM is configured for valid EURO banknotes only. Thus, the possible values for a banknote are: 5, 10, 20, 50, 100, 200, 500. It’s the user’s responsibility to check that the value written on the switches is correct.

Extract Sum

To withdraw money from the atm, the user needs to push the left button (= extract sum button), after which the message “GET” appears on the display. If he wants to go back to the main menu, he needs to push the left button once again (= back button). If he wants to continue with the extraction, he needs to introduce the amount on the switches in BCD format. The maximum amount that can be withdrawn is 1000 euro. Next, he pushes the right button (= extract button), after which the amount is processed. If the amount is valid, meaning that there is enough money on his account and there are enough banknotes to give out the sum, the output banknotes will be shown one by one on the display. In the end, an “OK” message will appear, at which point the user can go back to the main menu by pushing the right button (= back button). In case the amount is invalid, the extraction is not performed and a “NOT” message will appear on the display, after which the user can go back to the main menu by pushing the left button (= back button).

Maintenance

The atm device will work properly as long as it is connected to a power source, the temperature and humidity of the air are optimal, and no chemical or physical harm is caused to the circuit. When the atm is first plugged in, it enters in an idle state. To reset the device, we need to push and then release the start button (= middle button), which also sets it to functioning.

IV. Further development

A straightforward improvement would be to allow more cards for the ATM. Due to the fact that we have used a generic RAM memory for storing the data, this improvement can be added just by changing the address width.

Other than that, there are several features which could be included in the project to improve the security of the ATM. Some ideas for security measures:

- We could block the card after 3 failed attempts to log-in. This can be easily solved by adding one more RAM memory to the accounts unit, in which we count for each card the number of failed attempts.
- We could add a currency detector to the ATM: at each deposit operation, the currency detector should check if the given value of the banknote is the value of a valid euro banknote