CSE312

Electronic Design Automation

Fall 2023

ATM - based bank system

Name	ID
Omar Alaa	21P0197
Abdelrahman Barakat	21P0012
Maeen Mohamed	2101544
Tsneam Ahmed	21P0284
Salma Mohamed Sayed	21P0279
Farah Ahmed	21P0402
Eslam Ashraf	21P0403

16/12/2023

Contents

1 Design

Design, as a discipline, is a complex and multifaceted field that encompasses a wide range of concepts, methodologies, and applications. It is the process of envisioning, planning, and creating products or solutions that serve specific purposes or meet certain requirements. This document delves into the intricacies of design in the context of a specific system, exploring its various states and transitions, and the mechanisms that govern these changes. The ensuing sections will provide a detailed account of the system's design, including its structure, functionalities, and the principles underlying its operation. This comprehensive exploration aims to elucidate the system's design, thereby providing valuable insights into its workings and potential for optimization.

1.1 Verilog Implementation

1.1.1 Introduction

The Verilog code describes the hardware level of an ATM machine By modularizing the system into two main modules with the benefits of using states to represent the states of the ATM machine such as waiting, menu, balance, transactions and deposit.

1.1.2 Modules

- **Authentication module**

The Authentication module plays a crucial role in ensuring the security and integrity of the ATM system. Its primary responsibility is to verify the identity of users attempting to access the system. This involves validating the entered account number and Personal Identification Number (PIN) against a pre-existing database of account information.

Module Overview:

- **Inputs:**
  - `accNumber`: 12-bit input for the account number.
  - `pin`: 4-bit input for the personal identification number (PIN).
  - `action`: Input signal that specifies the action to be performed (`FIND` or `AUTHENTICATE`).
  - `deAuth`: Input signal to de-authenticate (possibly log out).
- **Outputs:**
  - `wasSuccessful`: Output signal indicating whether the authentication or finding process was successful (`1'bx` when de-authenticating).
  - `accIndex`: Output signal providing the index of the account in the database (relevant for `FIND` action).
- **Internal Database Initialization:** The module initializes an internal database with account numbers and corresponding PINs.
- **De-authentication Handling:** The module monitors the `deAuth` signal and sets `wasSuccessful` to `1'bx` when de-authentication is requested.
- **Authentication and Finding Logic:**

– The module has logic to loop through the internal database to find a match for the given `accNumber` and `pin`.
– If the `action` is set to `FIND`, it searches for a match and sets `wasSuccessful` accordingly, providing the index in `accIndex`.
– If the `action` is set to `AUTHENTICATE`, it checks if the PIN matches the given account and sets `wasSuccessful` accordingly, providing the index in `accIndex`.

```
module authentication(
  input [11:0] accNumber,
  input [3:0] pin,
  input action,
  input deAuth,
  output reg  wasSuccessful,
  output reg [3:0] accIndex
);
  reg [11:0] acc_database [0:9];
  reg [3:0] pin_database [0:9];

  //initializing the database with arbitrary accounts
  initial begin
    acc_database[0] = 12'd4023; pin_database[0] = 4'b0000;
    acc_database[1] = 12'd4000; pin_database[1] = 4'b0001;
    acc_database[2] = 12'd3993; pin_database[2] = 4'b0010;
    acc_database[3] = 12'd3467; pin_database[3] = 4'b0011;
    acc_database[4] = 12'd3100; pin_database[4] = 4'b0100;
    acc_database[5] = 12'd2937; pin_database[5] = 4'b0101;
    acc_database[6] = 12'd2816; pin_database[6] = 4'b0110;
    acc_database[7] = 12'd2429; pin_database[7] = 4'b0111;
    acc_database[8] = 12'd1697; pin_database[8] = 4'b1000;
    acc_database[9] = 12'd1392; pin_database[9] = 4'b1001;
    end

  always @ (deAuth) begin
    if(deAuth == `true)
      wasSuccessful = 1'bx;
  end
  reg [3:0] i;
  always @(accNumber or pin) begin
      wasSuccessful = `false;
      accIndex = 0;

      //loop through the data base
      for(i = 0; i < 10; i = i+1) begin: loop

          //found a match for accNumber
          if(accNumber == acc_database[i]) begin
              if(action == `FIND) begin
                wasSuccessful = `true;
                accIndex = i;
                disable loop;
              end
              else if(action == `AUTHENTICATE) begin
                if(pin == pin_database[i]) begin
                  wasSuccessful = `true;
                  accIndex = i;
                  disable loop;
                end
                else begin
                  wasSuccessful = `false;
                end
              end
          end
      end
  end
endmodule
```

Figure 1: Authentication code

- **ATM module**

This module controls the flow of the system by switching between different states.The module operates in distinct states, including a waiting state for user login and a menu state for navigating transaction options. It interacts with an authentication sub-module to verify user credentials, employing robust security measures. The ATM module facilitates various transactions, such as balance inquiries, cash withdrawals, fund transfers, and deposits. It dynamically manages account balances, ensuring the integrity of financial operations. The inclusion of language support allows for a personalized user experience. Additionally, the module implements timeout mechanisms to enhance security. Through its well-defined states and seamless integration with authentication and transaction sub-modules, the ATM module serves as a comprehensive solution for enabling secure and efficient financial interactions.

- **Inputs:**
  - `clk`: Clock signal.
  - `exit`: Asynchronous exit signal.
  - `lang`: Language selection signal (Arabic or English).
  - `accNumber`: Account number input.
  - `pin`: PIN input.
  - `destinationAcc`: Destination account number input for transactions.
  - `menuOption`: Menu option input.
  - `amount`: Amount input for transactions.
  - `depAmount`: Deposit amount input.
- **Outputs:**
  - `error`: Error signal indicating if an invalid operation occurred.
  - `balance`: Balance output.
- **Internal Variables:**
  - `balance_database`: Database storing initial balance values for different accounts.
  - `currState`: Current state of the FSM.
  - `accIndex`: Index of the authenticated account.
  - `destinationAccIndex`: Index of the destination account for transactions.
  - `isAuthenticated`: Signal indicating successful authentication.
  - `wasFound`: Signal indicating if a destination account was found.
  - `choice`: Signal used for decision-making during deposit.
- **Functionality:**
  - The module initializes the balance database and handles language-based display messages.

- It includes an authentication module (`authentication`) to verify account credentials.
- The module operates as a finite state machine (FSM) with states like `WAITING`, `MENU`, `BALANCE`, `WITHDRAW`, `WITHDRAW_SHOW_BALANCE`, `TRANSACTION`, and `DEPOSIT`.
- The FSM transitions based on user input and performs actions such as balance inquiry, withdrawal, transaction, and deposit.
- Error handling is implemented for cases like insufficient funds, timeout limits, and invalid operations.

1.1.3 ATM states

WAITING:

- **Description:** The initial state where the system waits for a user to authenticate.
- **Transition Conditions:** Moves to `MENU` upon successful authentication.
- **Behavior:**
  - Checks user authentication status.
  - Displays an error message if authentication fails.
  - Waits for user input or asynchronous exit.

```
`WAITING: begin
  if (isAuthenticated == `true && logout == `false) begin
    currState = `MENU;
    // if( lang == `arabic )
    //   $display(" تم تسجيل الدخول");
    // else
    //   $display("Logged In.");
  end
  else if(isAuthenticated == `false || logout == `true) begin
    // if( lang == `arabic ) begin
    //   if (logout == `true)
    //     $display("تم تسجيل الخروج");
    //   else
    //     $display(" رقم الحساب او كلمه المرور خطأ");
    // end
    // else begin
    //   if (logout == `true)
    //     $display("You Have Logged Out");
    //   else
    //    $display("Account number or password was incorrect");
    // end
    counter = 0;
    currState = `WAITING;
  end
end
```

Figure 2: Waiting state code

MENU:

- **Description:** Represents the main menu where users can choose different operations.
- **Transition Conditions:** Transitions to various states based on the selected menu option.
- **Behavior:**
  - Displays the main menu with available options.
  - Waits for user input to select an operation.

```
always @(menuOption) begin
    counter = 0;
    balance = balance_database[accIndex];
    currState = menuOption;
    if(logout == `true)
      currState = `WAITING;
end

always @(isAuthenticated) begin
 if(logout == `true) begin
    //transition to the waiting state
    currState = `WAITING;
    //deathenticate the current user
    deAuth = `true;
 end
end
```

Figure 3: Menu state code

BALANCE:

- **Description:** Displays the account balance to the user.
- **Transition Conditions:** Returns to `MENU` after displaying the balance.
- **Behavior:**
  - Retrieves and displays the account balance.
  - Handles timeout limits for the operation.

```
`WITHDRAW: begin
    if (amount <= balance_database[accIndex]) begin
      balance_database[accIndex] = balance_database[accIndex] - amount;
      balance = balance_database[accIndex];
      currState = `MENU;
    end
    else begin
      currState = `MENU;
    end
end
```

Figure 4: Show Balance state code

WITHDRAW_SHOW_BALANCE:

- **Description:** Combines the withdrawal operation with displaying the updated account balance.
- **Transition Conditions:** Returns to `MENU` after completing the withdrawal and showing the balance.
- **Behavior:** Similar to `WITHDRAW` but also displays the updated account balance.

```
`WITHDRAW_SHOW_BALANCE: begin
    if (amount <= balance_database[accIndex]) begin
      balance_database[accIndex] = balance_database[accIndex] - amount;
      balance = balance_database[accIndex];
      currState = `MENU;
      // if( lang == `arabic )
      //   $display("الحساب رقم %d لديه رصيد %d بعد سحب مبلغ %d", accNumber, balance_database[accIndex], amount);
      // else
      //   $display("Account %d has balance %d after withdrawing %d", accNumber, balance_database[accIndex], amount);
    end
    else begin
      currState = `MENU;
    end
end
```

Figure 5: Withdraw and Show Balance state code

TRANSACTION:

- **Description:** Manages account-to-account transactions.
- **Transition Conditions:** Returns to `MENU` after completing the transaction.
- **Behavior:**
  - Verifies the availability of funds and the existence of the destination account.
  - Updates account balances for both source and destination accounts.
  - Handles timeout limits for the operation.

```
`TRANSACTION: begin
    if ((amount <= balance_database[accIndex]) & (wasFound == `true) & (balance_database[accIndex] + amount < 2048)) begin
        initial_balance = balance_database[destinationAccIndex];
        currState = `MENU;
        balance_database[destinationAccIndex] = balance_database[destinationAccIndex] + amount;
        balance_database[accIndex] = balance_database[accIndex] - amount;
        balance = balance_database[accIndex];
        // Store the balance after the transaction
        final_balance = balance_database[destinationAccIndex];
        // $display("Destination account %d after transaction has a total balance of %d", destinationAcc, balance_database[destinationAccIndex]);
        // if( lang == `arabic )
        //   $display("رقم الحساب المستلم %d بعد العملية لديه رصيد إجمالي قدره %d", destinationAcc, balance_database[destinationAccIndex]);
        // else
        //   $display("Destination account %d after transaction has a total balance of %d", destinationAcc, balance_database[destinationAccIndex]);
    end
    else begin
        currState = `MENU;
    end
end
```

Figure 6: Transactions code

DEPOSIT:

- **Description:** Handles the deposit operation.
- **Transition Conditions:** Returns to `MENU` after completing the deposit.
- **Behavior:**
  - Prompts the user to confirm the deposit amount.
  - Updates the account balance after a confirmed deposit.
  - Handles timeout limits for the operation.

```
`DEPOSIT:
  begin : Deposit
      // if( lang == `arabic )
      //     $display("هو المبلغ المودع %d", amount);
      // else
      //   $display("The deposited amount is %d", amount);
      balance_database[accIndex] = balance_database[accIndex] + amount;
      balance = balance_database[accIndex];
      // if( lang == `english )
      //   $display("Account %d has balance %d after depositing %d", accNumber, balance_database[accIndex], amount);
      // else
      //   $display("الحساب %d لديه رصيد %d بعد إيداع مبلغ %d", accNumber, balance_database[accIndex], amount);
  end
```

Figure 7: Deposit code

1.2 Finite State Machine Diagram

This diagram represents the sequences of the possible states the system can undergo.

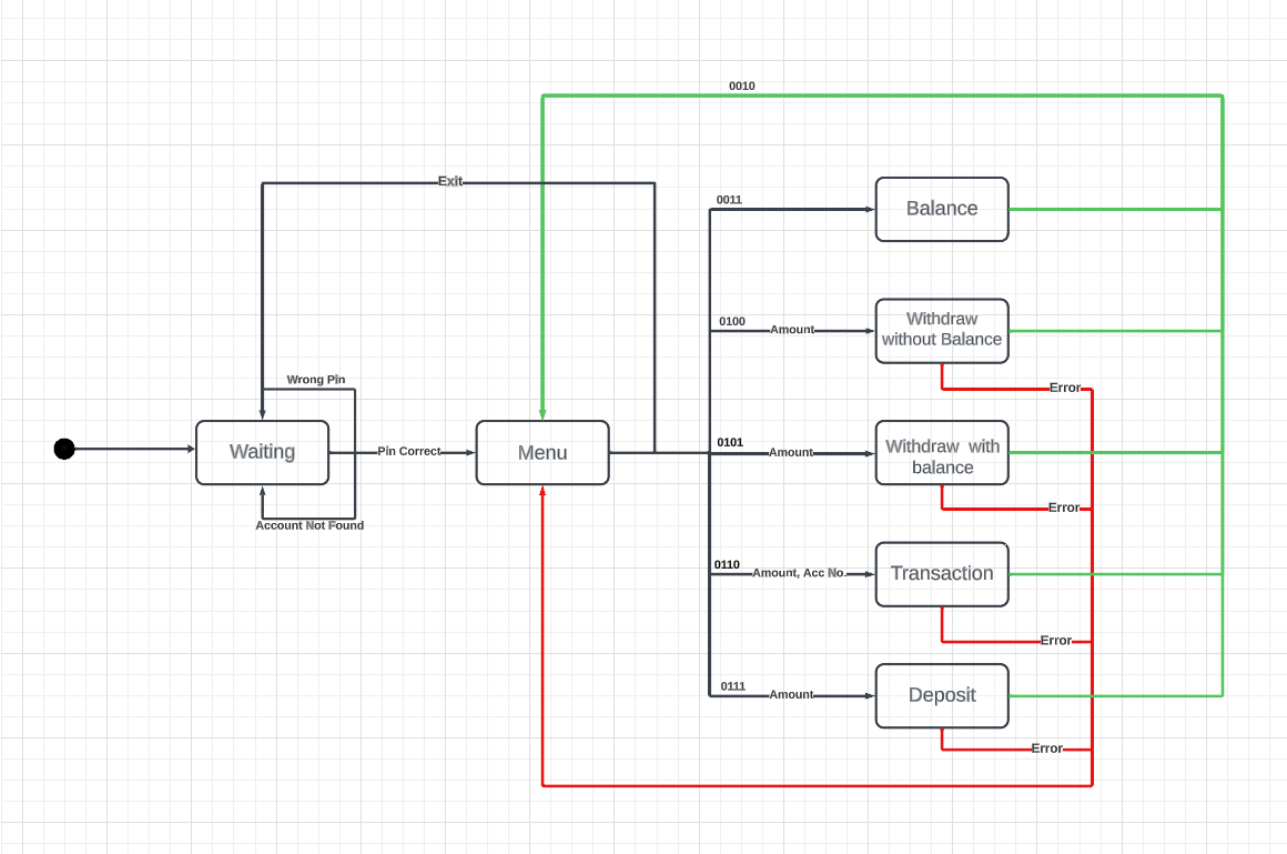

Figure 8: Finite State Machine Diagram

The system initiates in the **Waiting** state. In this state, the system awaits the user to input their account information. Upon receiving the correct account details, the system transitions to the **Menu** state, granting the user access to various services.

However, if the entered account information is incorrect, the system remains in the **Waiting** state. This mechanism ensures that only users with valid credentials can navigate beyond the initial state, thereby enhancing the security of the system.

This process exemplifies a basic yet effective user authentication system, where state transitions are contingent upon the validity of user-provided information. It's a common practice in systems design to safeguard sensitive user data and services.

Upon reaching the **Menu** state, the system presents six potential states for transition: **Balance**, **Withdraw with Balance**, **Withdraw without Balance**, **Transaction**, **Deposit**, and return to **Waiting** State.

Each state represents a distinct functionality offered to the user. The transition from the Menu state to any of these states is contingent upon specific user input, which serves as a command for the desired operation.

For instance, if the user wishes to check their account balance, they would provide the corresponding input(0011), prompting the system to transition to the **Balance** state. Similarly, inputs related to withdrawal, transaction, or deposit would lead the system to their respective states.

In the event the user completes their tasks or opts to exit, the system would receive the command to return to the initial **Waiting** state. This cyclical process ensures a seamless and secure user experience, allowing for efficient navigation through various banking operations.

Upon successful execution of a function, the system transitions back to the **Menu** state, allowing for continuous operation. This design ensures a smooth user experience by providing immediate access to other functionalities after the completion of a task.

In the event of an unsuccessful operation or an error during execution, the system is designed to handle such scenarios gracefully. It will display an error message to inform the user of the issue encountered. Following this, the system will transition back to the **Menu** state. This approach ensures that users are always informed about the status of their requests and can easily navigate back to the main menu, regardless of the outcome of their previous operation.

This robust design contributes to a resilient system that maintains operational continuity even in the face of unexpected events or errors. It prioritizes user experience by providing clear feedback and ensuring seamless navigation through various functionalities.

1.3 High level model

A reference model in Verilog is a high-level representation of the design being tested. It predicts the expected output for given inputs. Also, it operates at a higher level of abstraction, making it simpler and faster. A reference model is also used to verify the correctness of the design by comparing its expected results with the actual results from the design. Any differences can indicate a potential issue in the design.

```
import random
from datetime import datetime
import time

LANGUAGES = {
    'en': {
        'welcome': "Welcome To The ATM",
        'invalid_login': "Invalid username or PIN. Please try again.",
        'log_in': "Logged in successfully.",
        'log_out': "Logged out successfully.",
        'balance': "Your current balance is: ",
        'deposit_done': "Account is Deposited Successfully",
        'deposit_entry': "Please Enter Your Cash: ",
        'insufficient_funds': "Insufficient funds",
        'account_not_found': "Account not found",
        'withdraw': "Please Collect Your Cash",
        'withdraw_with_balance': "Please Collect Your Cash\nYour current balance is: ",
        'invalid_choice': "Invalid choice. Please enter a number between 1 and 9.",
        'thank_you': "Thank you for using the ATM. Goodbye!"
    },
    'ar': {
        'welcome': "مرحبًا بك في جهاز الصراف الآلي",
        'invalid_login': "اسم مستخدم أو رقم تعريف شخصي غير صالح. يرجى المحاولة مرة أخرى.",
        'log_in': "تم تسجيل الدخول بنجاح.",
        'log_out': "تم تسجيل الخروج بنجاح.",
        'balance': "رصيدك الحالي هو: ",
        'deposit_done': "تم إيداع الحساب بنجاح",
        'deposit_entry': "يرجى إدخال مبلغ الإيداع الخاص بك: ",
        'insufficient_funds': "رصيد غير كافٍ",
        'account_not_found': "الحساب غير موجود",
        'withdraw': "يرجى استلام نقودك",
        'withdraw_with_balance': "يرجى استلام نقودك\nرصيدك الحالي هو: ",
        'invalid_choice': "خيار غير صالح. يرجى إدخال رقم بين 1 و 9.",
        'thank_you': "شكرًا لاستخدام جهاز الصراف الآلي. وداعاً!"
    }
}
```

Figure 9: Dictionary in Python that contains messages for the ATM

The provided Python code represents a bilingual dictionary for an Automated Teller Machine (ATM) system. This dictionary supports two languages: English (denoted by 'en') and Arabic (denoted by 'ar'). Each language is associated with a corresponding dictionary that contains key-value pairs. The keys in these dictionaries serve as unique identifiers for specific messages that the ATM system may need to display, while the values are the actual messages in the respective language.
Here's a brief description of each key:

Key	**Message**
welcome	A welcome message displayed when a user interacts with the ATM.
invalid_login	A message displayed when a user enters an invalid username or PIN.
log_in	A message displayed when a user successfully logs in.
log_out	A message displayed when a user successfully logs out.
balance	A message displayed when showing the user's current balance.
deposit_done	A message displayed when a deposit to the account is successful.
deposit_entry	A prompt asking the user to enter the deposit amount.
insufficient_funds	A message displayed when there are insufficient funds in the account for a withdrawal.
withdraw	A message displayed when the user can collect their cash after a withdrawal.
withdraw_with_balance	A message displayed when the user can collect their cash after a withdrawal, along with their current balance.
invalid_choice	A message displayed when the user enters an invalid choice in a menu.
thank_you	A farewell message displayed when the user finishes their session with the ATM.

This bilingual dictionary is a crucial component of the ATM system's internationalization and localization efforts. It enables the system to support multiple languages and provides a scalable solution for adding more languages in the future. By simply looking up the appropriate message based on the user's selected language and the situation, the system can communicate effectively with its users.

Moreover, this approach enhances the maintainability of the code. All the messages are centralized in one place, which means if a message needs to be changed, it can be done in the dictionary without having to search through the entire codebase. This approach also adheres to the DRY (Don't Repeat Yourself) principle, a fundamental concept in software development that reduces repetition and redundancy in the code, making it more efficient and easier to manage.

```python
class ATM:
    def __init__(self, balance=500, lang='en'):
        print(LANGUAGES[lang]['welcome'])
        self.balance = balance
        self.users = {
            '4023': {'pin': 0, 'balance': 500},
            '4000': {'pin': 1, 'balance': 500},
            '3993': {'pin': 2, 'balance': 500},
            '3467': {'pin': 3, 'balance': 500},
            '3100': {'pin': 4, 'balance': 500},
            '2937': {'pin': 5, 'balance': 500},
            '2816': {'pin': 6, 'balance': 500},
            '2429': {'pin': 7, 'balance': 500},
            '1697': {'pin': 8, 'balance': 500},
            '1392': {'pin': 9, 'balance': 500}
        }
        self.current_user = None
        self.lang = lang
        self.idle_timeout = 8  # 8 Seconds
        self.last_activity_time = time.time()

    def is_authentic(self, accountNum, pin):
        if accountNum in self.users and str(self.users[accountNum]['pin']) == pin:
            self.current_user = accountNum
            return True
        else:
            return False

    def login(self, accountNum, pin):
        global LANGUAGES
        if self.is_authentic(accountNum, pin):
            return LANGUAGES[self.lang]['log_in']
        else:
            return LANGUAGES[self.lang]['invalid_login']
```

Figure 10: Class ATM 1

```
74          def logout(self):
75              self.current_user = None
76              return LANGUAGES[self.lang]['log_out']
77
78          def check_balance(self):
79              return str(LANGUAGES[self.lang]['balance']) + str(self.users[self.current_user]['balance'])
80
81          def wasFound(self, dest):
82              return self.users.get(str(dest)) is not None
83
84          def deposit(self, amount):
85              self.users[self.current_user]['balance'] += amount
86              return LANGUAGES[self.lang]['deposit_done']
87
88          def withdraw(self, amount):
89              if amount > self.users[self.current_user]['balance']:
90                  return LANGUAGES[self.lang]['insufficient_funds']
91              else:
92                  self.users[self.current_user]['balance'] -= amount
93                  return LANGUAGES[self.lang]['withdraw']
94
95          def withdraw_with_balance(self, amount):
96              if amount > self.users[self.current_user]['balance']:
97                  return LANGUAGES[self.lang]['insufficient_funds']
98              else:
99                  self.users[self.current_user]['balance'] -= amount
100                 return LANGUAGES[self.lang]['withdraw_with_balance'] + str(self.users[self.current_user]['balance'])
101
102         def transaction(self, amount, dest):
103             if not self.wasFound(str(dest)):
104                 return LANGUAGES[self.lang]['account_not_found']
105             elif amount > self.users[self.current_user]['balance']:
106                 return LANGUAGES[self.lang]['insufficient_funds']
107             else:
108                 self.users[self.current_user]['balance'] -= amount
109                 self.users[str(dest)]['balance'] += amount
110                 return LANGUAGES[self.lang]['balance'] + str(self.users[self.current_user]['balance'])
```

Figure 11: Class ATM 2

```
    def check_idle_timeout(self):
        current_time = time.time()
        idle_time = current_time - self.last_activity_time
        if idle_time >= self.idle_timeout:
            if self.lang == "en":
                choice = str(input("You are Timedout, Do you want to continue? (Y/N)\n"))
                if choice.lower() == "n":
                    if self.current_user is not None:
                        self.logout()
                        print("Logout due to inactivity.")
            else:
                choice = str(input("(نعم/لا) هل ترغب في المتابعة؟ ,تم انتهاء الوقت المحدد\n"))
                if choice == "لا":
                    if self.current_user is not None:
                        self.logout()
                        print("تسجيل الخروج بسبب عدم النشاط.")
```

Figure 12: Class ATM 3

The provided Python code outlines a class, "ATM", which simulates the functionality of an Automated Teller Machine (ATM). The class includes several methods that correspond to typical operations performed on an ATM, such as checking balance, depositing money, and withdrawing money.

The class is initialized with a balance, language preference, a dictionary of users, and a timeout for idle sessions. The dictionary of users contains account numbers as keys, and each account is associated with a PIN and balance.

The "is-authentic" method checks the validity of the account number and PIN entered by the user. The "login" and "logout" methods handle user sessions. The "check-balance", "deposit", and "withdraw" methods perform standard ATM operations. The "check-idle-timeout" method ensures that idle sessions are terminated after a certain period of inactivity.

```python
def main():
    atm = ATM()
    lang = int(input("Please Choose Your Language: (English: 0), (1 :اللغة العربية): "))
    if lang == 0:
        atm.lang = 'en'
        while True:
            print("\n==== ATM Menu ====")
            print("1. Login")
            print("2. Logout")
            print("3. Check Balance")
            print("4. Withdraw")
            print("5. Withdraw With Balance")
            print("6. Transaction")
            print("7. Deposit")
            print("8. Exit")

            choice = input("Enter your choice (1-8): ")

            match choice:
                case "1":
                    if atm.current_user:
                        print("Already logged in. Logout first.")
                    else:
                        accountNum = input("Enter your accountNum: ")
                        pin = input("Enter your PIN: ")
                        print(atm.login(accountNum, pin))

                case "2":
                    print(atm.logout())

                case "3":
                    atm.check_idle_timeout()
                    if atm.current_user:
                        print(atm.check_balance())
                    else:
                        print("Please login first.")

```

Figure 13: Main Function 1

```
167                 case "4":
168                     atm.check_idle_timeout()
169                     if atm.current_user:
170                         amount = float(input("Enter the amount to withdraw: "))
171                         print(atm.withdraw(amount))
172                     else:
173                         print("Please login first.")
174 
175                 case "5":
176                     atm.check_idle_timeout()
177                     if atm.current_user:
178                         amount = float(input("Enter the amount to withdraw: "))
179                         print(atm.withdraw_with_balance(amount))
180                     else:
181                         print("Please login first.")
182 
183                 case "6":
184                     atm.check_idle_timeout()
185                     if atm.current_user:
186                         amount = float(input("Enter transfer amount: "))
187                         dest = int(input("Enter destination account number: "))
188                         print(atm.transaction(amount, dest))
189                     else:
190                         print("Please login first.")
191 
192                 case "7":
193                     atm.check_idle_timeout()
194                     if atm.current_user:
195                         amount = float(input(LANGUAGES[atm.lang]['deposit_entry']))
196                         print(atm.deposit(amount))
197                     else:
198                         print("Please login first.")
199 
200                 case "8":
201                     print("Thank you for using the ATM. Goodbye!")
202                     break
203 
```

Figure 14: Main Function 2

```
204                     case _:
205                         print("Invalid choice. Please enter a number between 1 and 8.")
206                 atm.last_activity_time = time.time()
207         elif lang == 1:
208             atm.lang = 'ar'
209             while True:
210                 print("\n==== قائمة اختيارات الصرف الالي ====")
211                 print("1. تسجيل الدخول")
212                 print("2. تسجيل خروج")
213                 print("3. التأكد من الرصيد")
214                 print("4. سحب نقدي")
215                 print("5. سحب نقدي مع اظهار الرصيد")
216                 print("6. تحويل رصيد")
217                 print("7. ايداع نقدي")
218                 print("8. الخروج")
219 
220                 choice = input("(8-1) برجاء ادخال رقم من: ")
221 
222                 match choice:
223                     case "1":
224                         if atm.current_user:
225                             print("لقد تم تسجيل الدخول لهذا الحساب من قبل, برجاء تسجيل الخروج اولا.")
226                         else:
227                             accountNum = input("برجاء ادخال رقم الحساب: ")
228                             pin = input("برجاء ادخال الرقم السري: ")
229                             print(atm.login(accountNum, pin))
230                     case "2":
231                         print(atm.logout())
232                     case "3":
233                         atm.check_idle_timeout()
234                         if atm.current_user:
235                             print(atm.check_balance())
236                         else:
237                             print("برجاء تسجيل الدخول اولا.")
238 
```

Figure 15: Main Function 2

```
247                     case "5":
248                         atm.check_idle_timeout()
249                         if atm.current_user:
250                             amount = float(input("ادخل المبلغ المراد سحبه: "))
251                             print(atm.withdraw_with_balance(amount))
252                         else:
253                             print("برجاء تسجيل الدخول اولا.")
254 
255                     case "6":
256                         atm.check_idle_timeout()
257                         if atm.current_user:
258                             amount = float(input("ادخل المبلغ المراد تحويله: "))
259                             dest = int(input("ادخل رقم الحساب المراد التحويل اليه: "))
260                             print(atm.transaction(amount, dest))
261                         else:
262                             print("برجاء تسجيل الدخول اولا.")
263 
264                     case "7":
265                         atm.check_idle_timeout()
266                         if atm.current_user:
267                             amount = float(input("ادخل المبلغ المراد ايداعه: "))
268                             print(atm.deposit(amount))
269                         else:
270                             print("برجاء تسجيل الدخول اولا.")
271 
272                     case "8":
273                         print("شكرا لاستخدامكم خدمات الصرف الالي. مع السلامه!")
274                         break
275                     case _:
276                         print("اختيار خاطئ برجاء اختيار رقم من 1 الى 8.")
277                 atm.last_activity_time = time.time()
278         else:
279             print("Invalid Input")
280 
281 
282     main()
```

Figure 16: Main Function 2

The provided Python code defines the ‘main‘ function for an Automated Teller Machine (ATM) simulation. This function creates an instance of the "ATM" class and provides an interactive menu for users to perform various operations.

Upon execution, the program prompts the user to choose a language. The user is then presented with an ATM menu that includes options to log in, log out, check balance, withdraw with balance, withdraw, deposit, and exit.

The "match" statement is used to handle the user's choice. Depending on the user's input, the program will call the appropriate method from the "ATM" class. For instance, if the user chooses to log in, the program will prompt the user for their account number and PIN, and then call the "login" method of the "ATM" class.

The "check-idle-timeout" method is called before each operation to ensure that the user's session is still active. If the user has been idle for longer than the specified timeout period, they are logged out automatically.

In summary, this "main" function serves as the entry point for the ATM simulation, providing an interactive interface for users to perform various banking operations. It demonstrates the effective use of object-oriented programming concepts, such as classes and methods, to create a user-friendly and functional simulation of an ATM system.

The Console results of the above codes are as follows

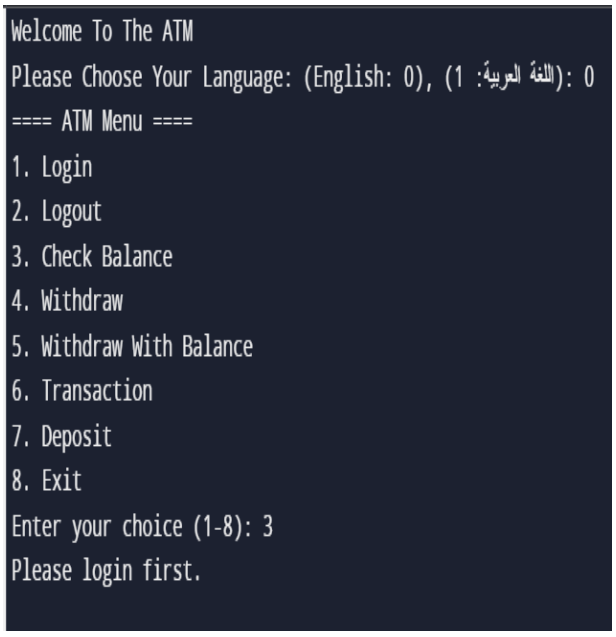

Figure 17: Console result 1

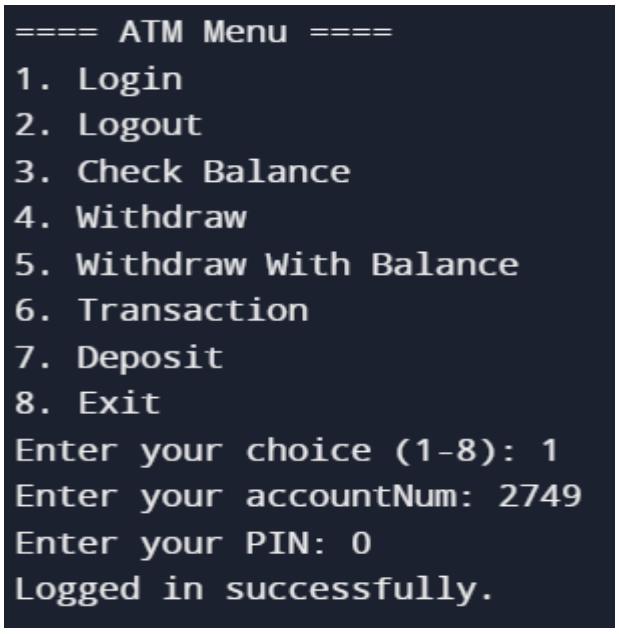

Figure 18: Console result 2

```
==== ATM Menu ====
1. Login
2. Logout
3. Check Balance
4. Withdraw
5. Withdraw With Balance
6. Transaction7. Deposit
8. Exit
Enter your choice (1-8): 3
You are Timedout, Do you want to continue? (Y/N)
Y
Your current balance is: 480.0

==== ATM Menu ====
1. Login
2. Logout
3. Check Balance
4. Withdraw
5. Withdraw With Balance
6. Transaction
7. Deposit
8. Exit
Enter your choice (1-8): 5
Enter the amount to withdraw: 80
Please Collect Your Cash
Your current balance is: 400.0
```

Figure 19: Console result 3

```
==== ATM Menu ====
1. Login
2. Logout
3. Check Balance
4. Withdraw
5. Withdraw With Balance
6. Transaction
7. Deposit
8. Exit
Enter your choice (1-8): 3
Your current balance is: 500

==== ATM Menu ====
1. Login
2. Logout
3. Check Balance
4. Withdraw
5. Withdraw With Balance
6. Transaction
7. Deposit
8. Exit
Enter your choice (1-8): 4
Enter the amount to withdraw: 20
Please Collect Your Cash
```

Figure 20: Console result 4

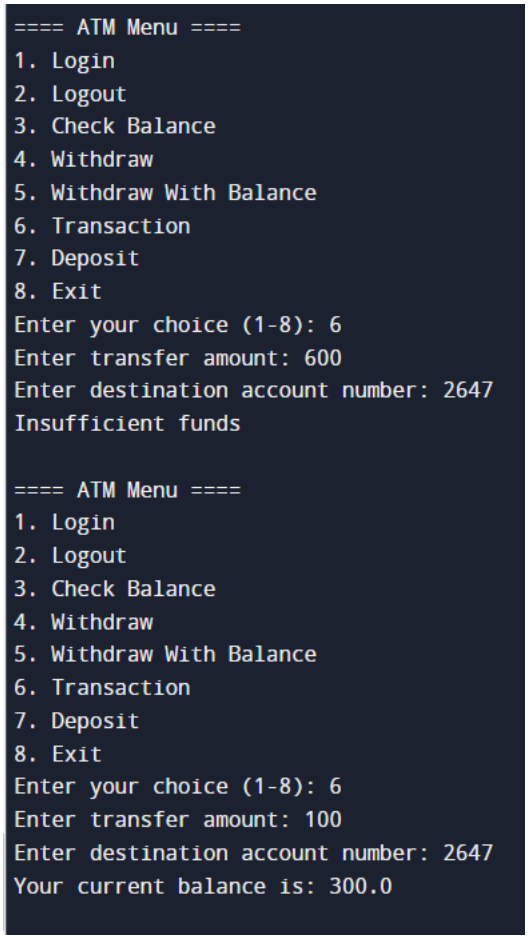

Figure 21: Console result 5

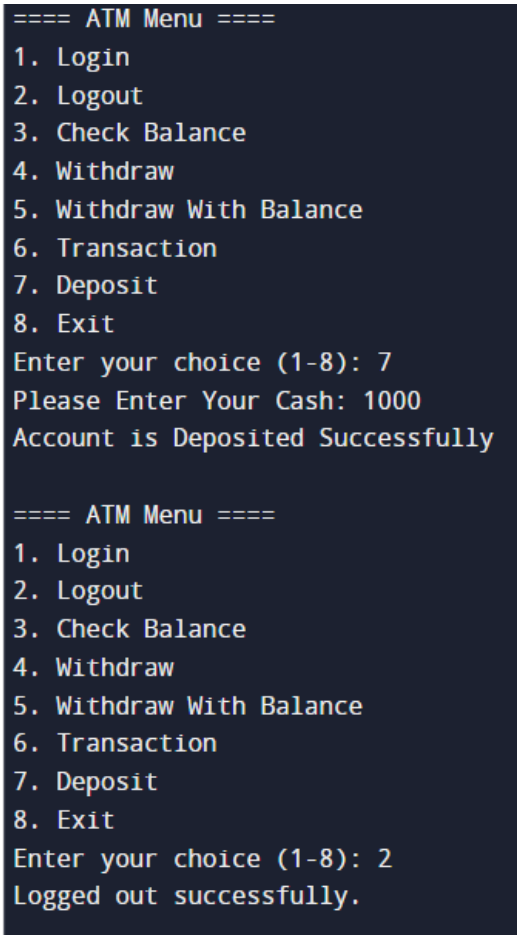

Figure 22: Console result 6

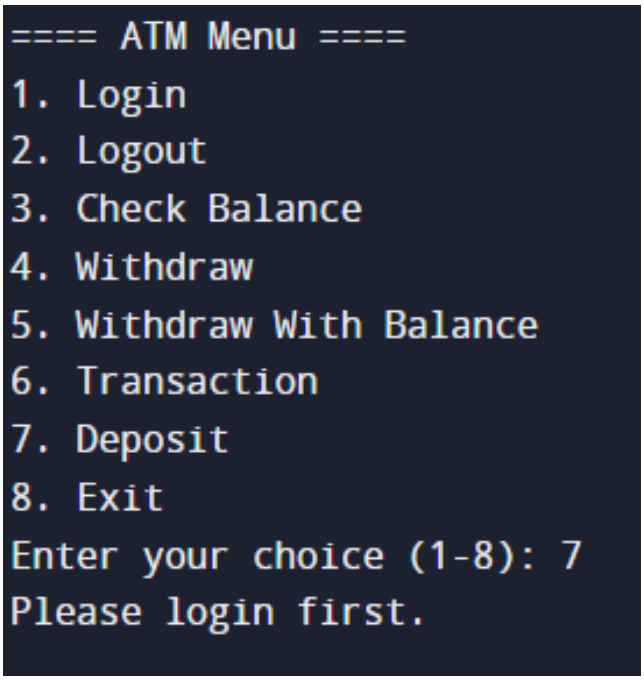

Figure 23: Console result 7

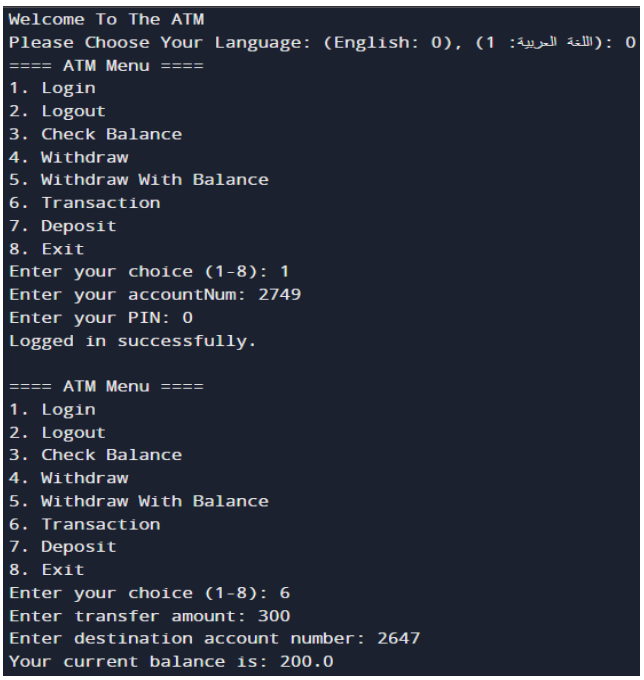

Figure 24: Console result 8

```
==== ATM Menu ====
1. Login
2. Logout
3. Check Balance
4. Withdraw
5. Withdraw With Balance
6. Transaction
7. Deposit
8. Exit
Enter your choice (1-8): 2
Logged out successfully.

==== ATM Menu ====
1. Login
2. Logout
3. Check Balance
4. Withdraw
5. Withdraw With Balance
6. Transaction
7. Deposit
8. Exit
Enter your choice (1-8): 1
Enter your accountNum: 2647
Enter your PIN: 5
Logged in successfully.
```

Figure 25: Console result 9

```
==== ATM Menu ====
1. Login
2. Logout
3. Check Balance
4. Withdraw
5. Withdraw With Balance
6. Transaction
7. Deposit
8. Exit
Enter your choice (1-8): 3
Your current balance is: 800.0

==== ATM Menu ====
1. Login
2. Logout
3. Check Balance
4. Withdraw
5. Withdraw With Balance
6. Transaction
7. Deposit
8. Exit
Enter your choice (1-8): 8
Thank you for using the ATM. Goodbye!
```

Figure 26: Console result 10

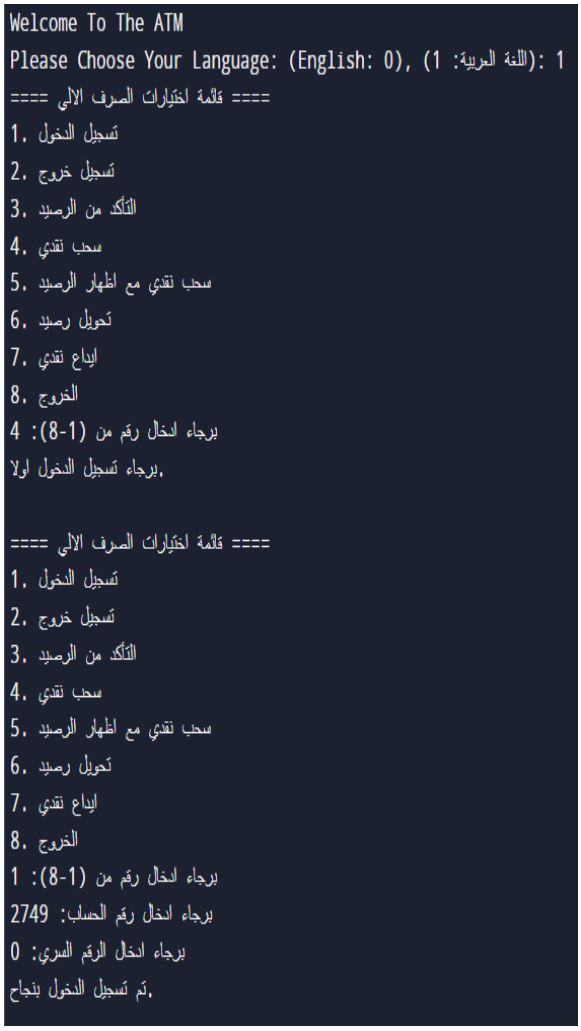

Figure 27: Console result 2 in Arabic

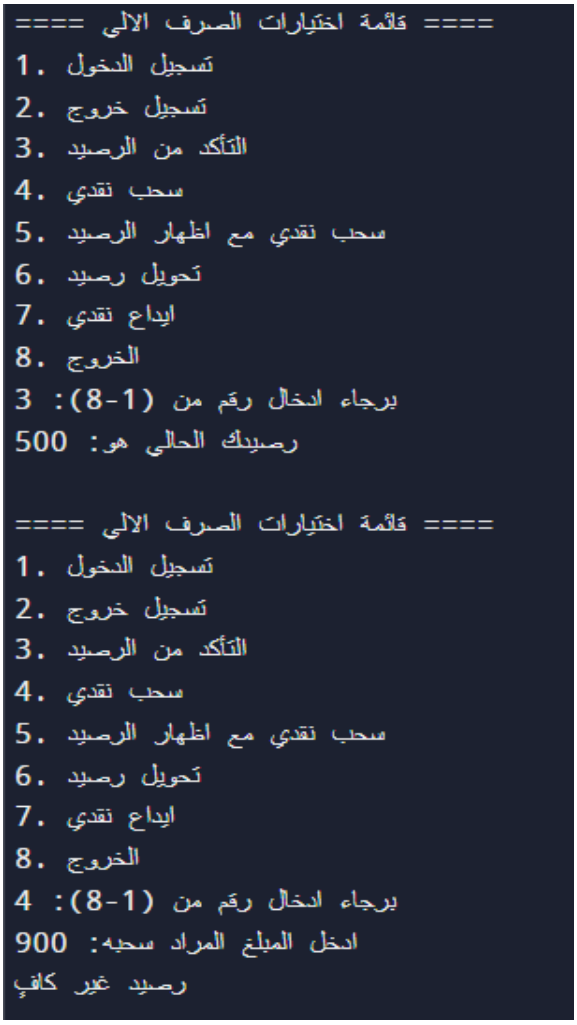

Figure 28: Console result 3 in Arabic

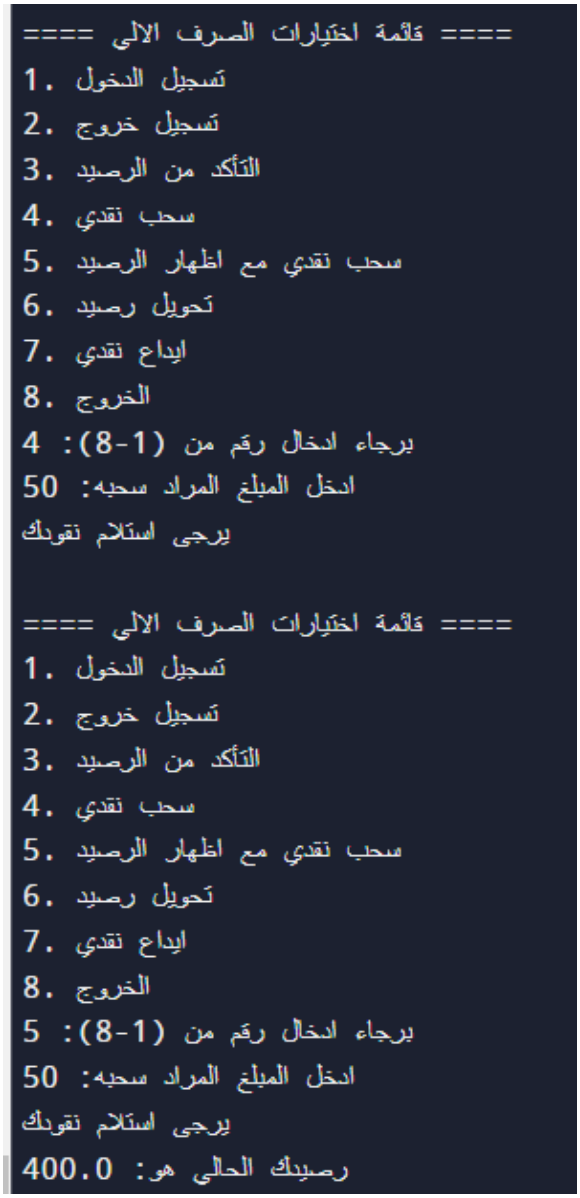

Figure 29: Console result 4 in Arabic

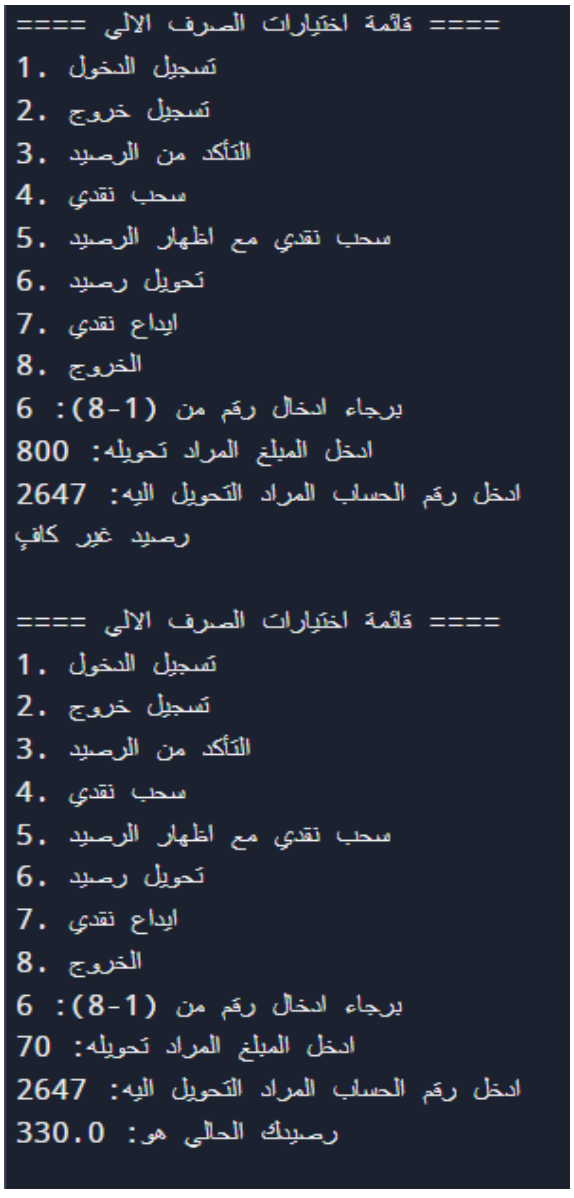

Figure 30: Console result 5 in Arabic

```
==== قائمة اختيارات الصرف الالي ====
1. تسجيل الدخول
2. تسجيل خروج
3. التأكد من الرصيد
4. سحب نقدي
5. سحب نقدي مع اظهار الرصيد
6. تحويل رصيد
7. ايداع نقدي
8. الخروج
7 :(8-1) برجاء ادخال رقم من
4000 :ادخل المبلغ المراد ايداعه
تم إيداع الحساب بنجاح

==== قائمة اختيارات الصرف الالي ====
1. تسجيل الدخول
2. تسجيل خروج
3. التأكد من الرصيد
4. سحب نقدي
5. سحب نقدي مع اظهار الرصيد
6. تحويل رصيد
7. ايداع نقدي
8. الخروج
3 :(8-1) برجاء ادخال رقم من
4330.0 :رصيدك الحالي هو
```

Figure 31: Console result 6 in Arabic

```
==== قائمة اختيارات الصرف الالي ====
1. تسجيل الدخول
2. تسجيل خروج
3. التأكد من الرصيد
4. سحب نقدي
5. سحب نقدي مع اظهار الرصيد
6. تحويل رصيد
7. ايداع نقدي
8. الخروج
2 :(8-1) برجاء ادخال رقم من
تم تسجيل الخروج بنجاح.

==== قائمة اختيارات الصرف الالي ====
1. تسجيل الدخول
2. تسجيل خروج
3. التأكد من الرصيد
4. سحب نقدي
5. سحب نقدي مع اظهار الرصيد
6. تحويل رصيد
7. ايداع نقدي
8. الخروج
4 :(8-1) برجاء ادخال رقم من
برجاء تسجيل الدخول اولا.
```

Figure 32: Console result 7 in Arabic

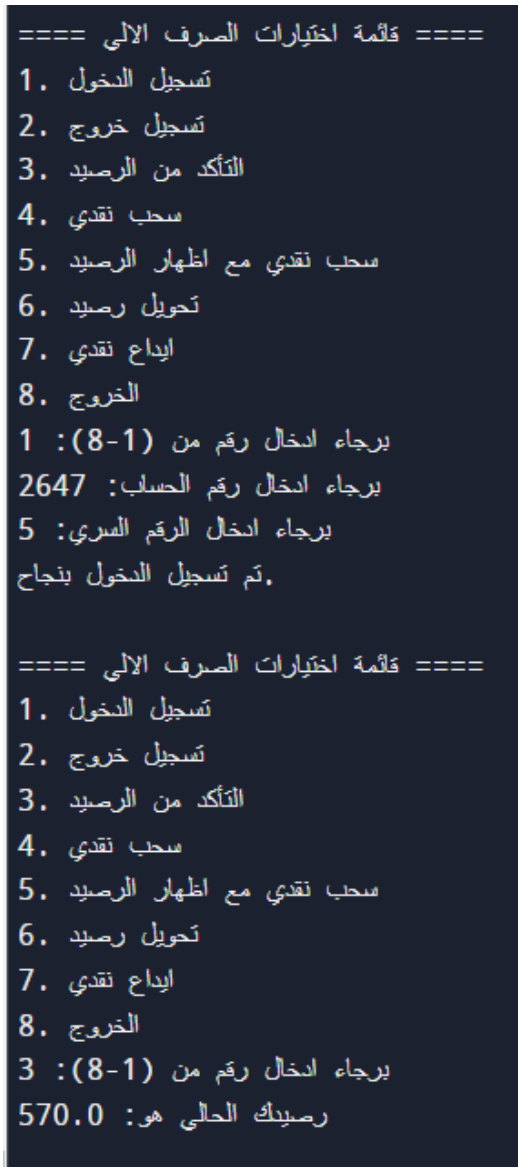

Figure 33: Console result 8 in Arabic

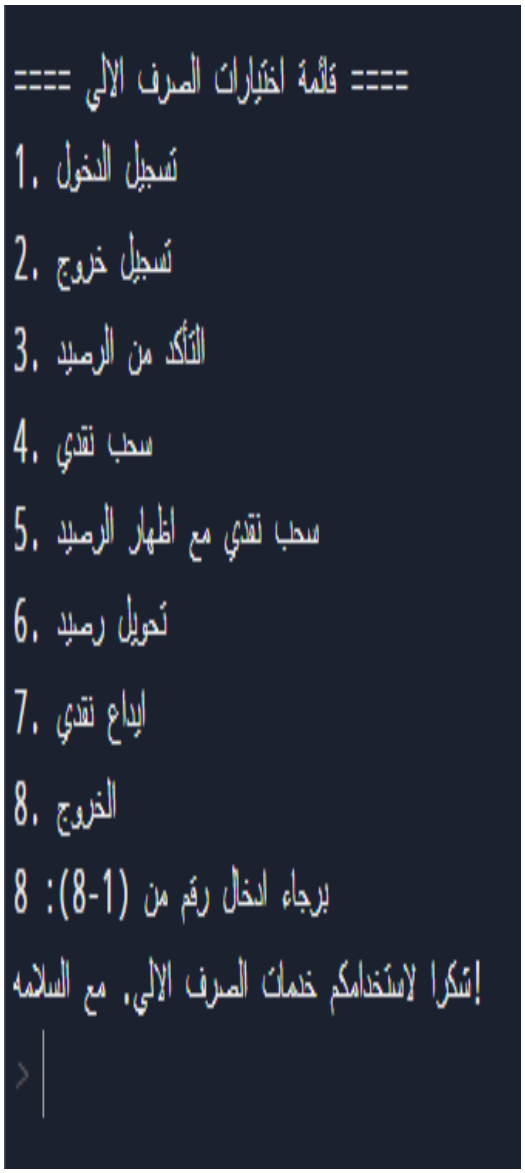

Figure 34: Console result 9 in Arabic

2 Verification

Verification is a critical phase in the development of any system. It serves as a checkpoint to ensure that the system functions as intended, adhering to the specified requirements and design principles. This document delves into the verification process of our system, detailing the methodologies employed, the tests conducted, and the results obtained. The objective is to validate the system's performance, reliability, and robustness under various conditions. The ensuing sections will provide a comprehensive account of the verification process, shedding light on the strategies used to confirm the system's functionality and the measures taken to rectify any identified issues. This exploration aims to underscore the importance of verification in maintaining the integrity and efficiency of the system.

2.1 Test Bench

The commencement of our test bench is marked by the definition of the constants and labels that we will be utilizing. This practice significantly enhances the accessibility and manageability of these elements throughout the test bench. It provides a structured approach to handle these constants and labels, thereby promoting efficiency and readability in our code.

```
1   `timescale 1ns/1ns
2   `define true 1'b1
3   `define false 1'b0
4
5   `define FIND 1'b0
6   `define AUTHENTICATE 1'b1
7
8   `define arabic 1'b1
9   `define english 1'b0
10
11  `define WAITING              3'b000
12  `define MENU                 3'b010
13  `define BALANCE              3'b011
14  `define WITHDRAW             3'b100
15  `define WITHDRAW_SHOW_BALANCE 3'b101
16  `define TRANSACTION          3'b110
17  `define DEPOSIT              3'b111
```

Figure 35: Definition of constants

After the definition of constants and labels, we proceed to establish a test bench module. This module encompasses all the wires and registers present in the system. The meticulous listing of these elements facilitates a comprehensive overview of the system's components, thereby streamlining the debugging and testing process. This methodical approach ensures a robust and efficient test bench, primed for rigorous system verification.

```
20    module atm_tb();
21
22      reg clk;
23      reg [11:0] accNumber;
24      reg [3:0] pin;
25      reg [11:0] destinationAccNumber;
26      reg [2:0] menuOption;
27      reg [10:0] amount;
28      wire [10:0] balance;
29      wire [10:0] initial_balance;
30      wire [10:0] final_balance;
31      reg lang;
32      reg [2:0] temp = 0;
33      reg [3:0] w;
34      reg [9:0] i;
35
36      ATM atmModule(clk, lang, accNumber, pin, destinationAccNumber, menuOption, amount, balance, initial_balance, final_balance);
```

Figure 36: Test bench module

The initial begin block is executed at the start of the simulation. Inside this block, two variables, clk and lang, are initialized. clk is set to 1'b0, which means it's a 1-bit binary number with a value of 0. lang is set to English.

initial begin ... end: This block of code will be executed once at the beginning of the simulation. It's often used to initialize variables.

clk=0;: This line initializes the clk variable to 0. clk is often used to represent a clock signal in digital circuits.

forever begin ... end: This is a loop that will run indefinitely. It's often used to model continuous behavior, like a clock signal.

#5 clk= clk;: This line toggles the value of clk every 5 units of simulation time. The #5 is a delay statement, which pauses execution for 5 time units. The operator is a bitwise NOT, which flips the bits of clk. So if clk was 0, it becomes 1, and vice versa.

So, in summary, this code models a clock signal that toggles its state every 5 units of time. The clock starts at 0 and then alternates between 0 and 1 indefinitely. This is a common pattern in digital circuit simulations.

```
38        initial begin
39          clk = 1'b0;
40          lang = `english;
41        end
42
43        initial begin
44          clk=0;
45          forever begin
46            #5 clk=~clk;
47          end
48        end
```

Figure 37: Initial blocks

In this test scenario, we commence by examining the authentication functionality of our system. Initially, we input an invalid PIN to assess the system's ability to reject unauthorized access. Following this, at the negative edge clock, we input the correct account information to verify that the system appropriately grants access to valid users.

This rigorous testing approach helps us ensure the robustness and security of our authentication system, thereby safeguarding user accounts against unauthorized access. It is a crucial part of maintaining the integrity and reliability of our services.

```
49        initial begin
50          // Direct Test Cases Verification
51          amount = 0;
52          accNumber = 12'd6134;
53          pin = 4'b1001;
54          @(negedge clk);
55          accNumber = 12'd2816;
56          pin = 4'b0110;
57          menuOption = `WAITING;
58          lang = `english;
```

Figure 38: Test authentication functionality

In this Verilog code snippet, a `for` loop is employed to iterate over a set of functions, using the variable w ranging from 3 to 8. This loop serves to generate direct test cases for each function.

- When w equals 3, it corresponds to the balance function, which displays the balance.
- For w values of 4 or 5, the code pertains to the withdrawal functionality. It is first tested with a valid amount, followed by an invalid amount.

- If w equals 6, the code relates to the transaction functionality. The language is switched to test the multilingual capabilities of the system. This function requires a destination account number, so a transaction is first attempted with an invalid account, followed by a valid one. It is then tested with both valid and invalid amounts.
- When w equals 7, the code corresponds to the deposit functionality. The language is again switched to continue testing multilingual support. The function is tested first with a valid amount, and then with an invalid one.

This systematic approach allows for thorough testing of each function under various conditions, ensuring the robustness and reliability of the system. It's a professional way to conduct comprehensive testing in digital circuit simulations.

```
60          for(w = 3; w < 8; w = w + 1)begin
61            menuOption = w;
62            if (w == 3)begin
63              @(negedge clk);
64            end
65            else if (w == 4 || w == 5) begin
66              amount = 50;
67              @(negedge clk);
68              amount = 62;
69              @(negedge clk);
70              amount = 505;
71              @(negedge clk);
72            end
73            else if (w == 6) begin
74              lang = ~lang;
75              destinationAccNumber = 12'd4634; amount = 29;
76              @(negedge clk);
77              destinationAccNumber = 12'd3467; amount = 99;
78              @(negedge clk);
79              amount = 73;
80              @(negedge clk);
81              amount = 503;
82              @(negedge clk);
83            end
84            else if (w == 7) begin
85              lang = ~lang;
86              amount = 429;
87              @(negedge clk);
88              amount = 430;
89              @(negedge clk);
90            end
91          end
```

Figure 39: Test the system's functionalities

We also validate the system's ability to maintain accurate account balances across multiple sessions. We do this by attempting to log into the account that we transferred to. This allows us to verify whether the balance has been updated correctly and consistently, ensuring the integrity of our banking system.

```
92          amount = 0;
93          // For Testing Timer
94          #1020;
95          ////////////////////////////
96          accNumber = 12'd3467;
97          pin = 4'b0011;
98          menuOption = 3;
99          @(negedge clk);
```

Figure 40: Re-log in

we here test the system by giving it random cases. This for loop will iterate 1000 times. we randomize several variables (lang, amount, temp, menuOption, destinationAccNumber). The menuOption is set to a random value between 3 and 7 inclusive. The simulation then waits for the next falling edge of the clock. #20 stop();: After the loop, the simulation waits for 20 units of time and then stops.

Here is the waveform that resulted from testing the test bench.

```
102         menuOption = `WAITING;
103         accNumber = 12'd3467;
104         pin = 4'b1000;
105
106         @(negedge clk);
107         for(i = 0; i < 1000;i = i + 1)begin
108           if (i != 0) begin
109             lang = $random();
110             amount = $random();
111             temp = $random();
112             // To Generate Menu Option in Range (3, 7) Inclusive
113             while (!(temp > 2 && temp < 8)) begin
114               temp = $random();
115             end
116             menuOption = temp;
117             destinationAccNumber = 12'd2429;
118           end
119           else
120             menuOption = `BALANCE;
121           @(negedge clk);
122         end
123         #20 $stop();
124       end
```

Figure 41: Constrain Random Verification

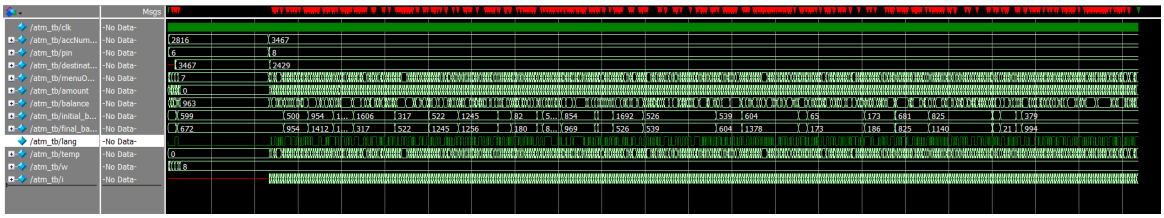

Figure 42: Waveform

2.2 Assertion

In SystemVerilog, an extension of Verilog, assertions are used to validate the behavior of a design. An assertion is a statement about your design that you expect to be always true. They provide a powerful way to write constraints, checkers, and cover points for your design.

There are two types of assertions in SystemVerilog:

- **Immediate Assertions**: These assertions do not depend upon a clock edge or reset. They are equivalent to an if condition within an always comb block.
- **Concurrent Assertions**: These assertions describe more complex expressions that span time and are triggered relative to a clock edge.

The code, we made, uses Property Specification Language (PSL) assertions. PSL is a language for specifying temporal properties, which are primarily used in model checking and formal verification of hardware designs.

These assertions are concurrent assertions as they describe behavior over time and are evaluated continuously throughout the simulation. They are triggered relative to a clock edge, as indicated by the @(posedge clk).

These assertions are used to verify that the system behaves as expected when performing deposit, withdrawal, and transaction operations. If any of these assertions fail during simulation, it indicates a bug in the design.

```
//psl Deposit_Check: assert always((menuOption==`DEPOSIT)->next(balance==( prev(balance) + prev(amount) ) ) ) @(posedge clk);
//psl Withdraw_Check: assert always((menuOption==`WITHDRAW)->next(balance==(prev(balance)-prev(amount)))) @(posedge clk);
//psl Withdraw_Show_Balance_Check: assert always((menuOption==`WITHDRAW_SHOW_BALANCE)->next(balance==(prev(balance)-prev(amount)))) @(posedge clk);
//psl Transaction_Check: assert always((menuOption==`TRANSACTION)->next(balance==prev(balance)-prev(amount) && final_balance==initial_balance+prev(amount))) @(posedge clk)
```

Figure 43: assertion code

Here is the waveform that resulted from testing the test bench.

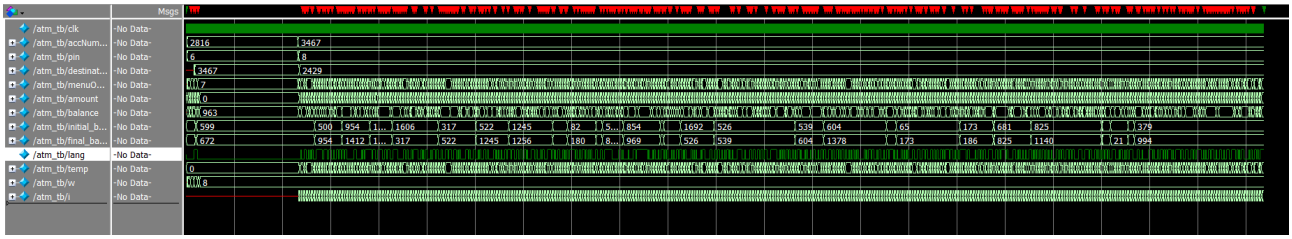

Figure 44: Waveform

These are some of the assertions messages resulted from the direct test cases verification part.

```
** Error: Assertion failed
   Time: 55 ns Started: 45 ns  Scope: /atm_tb/Withdraw_Check File: testbench.v Line: 126
** Error: Assertion failed
   Time: 85 ns Started: 75 ns  Scope: /atm_tb/Withdraw_Show_Balance_Check File: testbench.v Line: 127
** Error: Assertion failed
   Time: 95 ns Started: 85 ns  Scope: /atm_tb/Transaction_Check File: testbench.v Line: 128
** Error: Assertion failed
   Time: 125 ns Started: 115 ns  Scope: /atm_tb/Transaction_Check File: testbench.v Line: 128
```

Figure 45: Assertion results of the direct test cases part

These are some of the assertions messages resulted from the constrained random verification part.

```
** Error: Assertion failed
   Time: 1215 ns Started: 1205 ns  Scope: /atm_tb/Withdraw_Show_Balance_Check File: testbench.v Line: 127
** Error: Assertion failed
   Time: 1225 ns Started: 1215 ns  Scope: /atm_tb/Withdraw_Show_Balance_Check File: testbench.v Line: 127
** Error: Assertion failed
   Time: 1235 ns Started: 1225 ns  Scope: /atm_tb/Transaction_Check File: testbench.v Line: 128
** Error: Assertion failed
   Time: 1245 ns Started: 1235 ns  Scope: /atm_tb/Withdraw_Show_Balance_Check File: testbench.v Line: 127
** Error: Assertion failed
   Time: 1265 ns Started: 1255 ns  Scope: /atm_tb/Withdraw_Show_Balance_Check File: testbench.v Line: 127
** Error: Assertion failed
   Time: 1305 ns Started: 1295 ns  Scope: /atm_tb/Withdraw_Show_Balance_Check File: testbench.v Line: 127
** Error: Assertion failed
   Time: 1315 ns Started: 1305 ns  Scope: /atm_tb/Withdraw_Show_Balance_Check File: testbench.v Line: 127
** Error: Assertion failed
   Time: 1375 ns Started: 1365 ns  Scope: /atm_tb/Transaction_Check File: testbench.v Line: 128
** Error: Assertion failed
   Time: 1385 ns Started: 1375 ns  Scope: /atm_tb/Withdraw_Check File: testbench.v Line: 126
** Error: Assertion failed
   Time: 1425 ns Started: 1415 ns  Scope: /atm_tb/Withdraw_Check File: testbench.v Line: 126
** Error: Assertion failed
   Time: 1445 ns Started: 1435 ns  Scope: /atm_tb/Transaction_Check File: testbench.v Line: 128
** Error: Assertion failed
   Time: 1475 ns Started: 1465 ns  Scope: /atm_tb/Transaction_Check File: testbench.v Line: 128
** Error: Assertion failed
   Time: 1485 ns Started: 1475 ns  Scope: /atm_tb/Transaction_Check File: testbench.v Line: 128
** Error: Assertion failed
   Time: 1515 ns Started: 1505 ns  Scope: /atm_tb/Transaction_Check File: testbench.v Line: 128
** Error: Assertion failed
   Time: 1575 ns Started: 1565 ns  Scope: /atm_tb/Transaction_Check File: testbench.v Line: 128
```

Figure 46: Assertion results of constrained random verification part

2.3 Coverage

Coverage 's like checking if your blueprint covers all the important parts of your gadget. Here's what it means:

- **Code Coverage:** Think of it as a detective checking every nook and cranny of your blueprint. It makes sure that every line of your Verilog code gets tested. If you miss a line, it's like leaving a hidden trapdoor in your gadget!
- **Functional Coverage:** This one is like a checklist. It asks, "Did you test all the cool features your gadget is supposed to have?" If you forget to test something, it's like forgetting to add a secret compartment to your robot.

In our system, we used code coverage to verify our system. We mainly focused on **Statements Coverage**, **Branches Coverage**, and **Toggles Coverage**.

Total Coverage:						**86.6%**
Coverage Type	**Bins**	**Hits**	**Misses**	**Weight**	**% Hit**	**Coverage**
Statements	55	55	0	1	100.00%	**100.00%**
Branches	23	23	0	1	100.00%	**100.00%**
Toggles	198	169	29	1	81.3%	**81.3%**

Figure 47: Coverage Results

2.4 Equivalence Checking

In the process of equivalence checking verification, both the Verilog system and the high-level Python model are provided with identical inputs to evaluate the system's output. The outcomes from both systems were found to be congruent, as demonstrated in the subsequent data. This consistency validates the accuracy and reliability of the system under test.

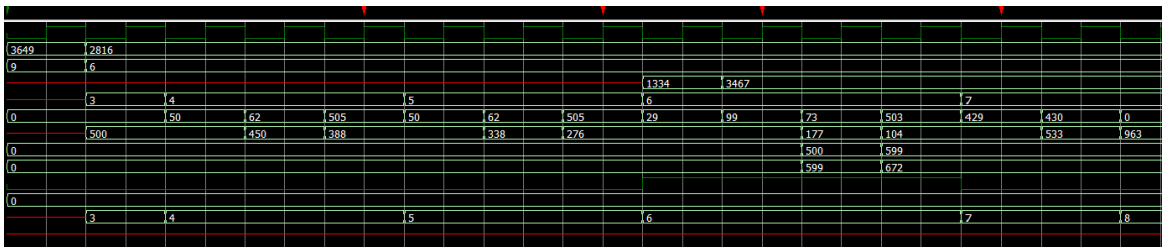

Figure 48: Verilog Waveform

```
Enter your choice (1-8): 1
Enter your accountNum: 3649
Enter your PIN: 9
Invalid username or PIN. Please try again.
```

Figure 49: Console result 1

```
Enter your choice (1-8): 1
Enter your accountNum: 2816
Enter your PIN: 6
Logged in successfully.
```

Figure 50: Console result 2

```
Enter your choice (1-8): 3
Your current balance is: 500
```

Figure 51: Console result 3

```
Enter your choice (1-8): 4
Enter the amount to withdraw: 50
Please Collect Your Cash
```

Figure 52: Console result 4

```
Enter your choice (1-8): 4
Enter the amount to withdraw: 62
Please Collect Your Cash
```

Figure 53: Console result 5

```
Enter your choice (1-8): 4
Enter the amount to withdraw: 505
Insufficient funds
```

Figure 54: Console result 6

```
Enter your choice (1-8): 5
Enter the amount to withdraw: 50
Please Collect Your Cash
Your current balance is: 338.0
```

Figure 55: Console result 7

```
Enter your choice (1-8): 5
Enter the amount to withdraw: 62
Please Collect Your Cash
Your current balance is: 276.0
```

Figure 56: Console result 8

```
Enter your choice (1-8): 5
Enter the amount to withdraw: 505
Insufficient funds
```

Figure 57: Console result 9

```
Enter your choice (1-8): 6
Enter transfer amount: 29
Enter destination account number: 1334
Account not found
```

Figure 58: Console result 10

```
Enter your choice (1-8): 6
Enter transfer amount: 99
Enter destination account number: 3467
Your current balance is: 177.0
```

Figure 59: Console result 11

```
Enter your choice (1-8): 6
Enter transfer amount: 73
Enter destination account number: 3467
Your current balance is: 104.0
```

Figure 60: Console result 12

```
Enter your choice (1-8): 6
Enter transfer amount: 503
Enter destination account number: 3467
Insufficient funds
```

Figure 61: Console result 13

```
Enter your choice (1-8): 7
Please Enter Your Cash: 429
Account is Deposited Successfully
```

Figure 62: Console result 14

```
Enter your choice (1-8): 7
Please Enter Your Cash: 430
Account is Deposited Successfully
```

Figure 63: Console result 15

```
Enter your choice (1-8): 3
Your current balance is: 963.0
```

Figure 64: Console result 16