ISEL
Departamento de Engenharia
Eletrónica e Telecomunicações
e de Computadores

Licenciatura em Engenharia Informática e de Computadores

Access Control System

Project of the course
Informatics and Computer Laboratory
2022 / 2023
summer

published: march 3rd, 2023

1 Description

This project consists of implementing an Access Control System that allows controlling access to restrict zones by means of a User Identification Number (UIN) and a Personal Identification Number (PIN). The system shall allow access to the restrict zone after the correct input of a UIN and PIN pair. After a valid access, the system allows the delivery of a text message addressed to the user.

The Access Control System is composed of: a 12-key keypad; a Liquid Cristal Display (LCD) with two lines and 16 columns; a mechanism for opening and closing the door (referred to as *Door Mechanism*); a maintenance key (referred to as *M*) which defines whether the system is in Maintenance Mode; and a Personal Computer (PC) responsible for controlling the other components and managing the system. A block diagram of the Access Control System is presented in Figure 1.

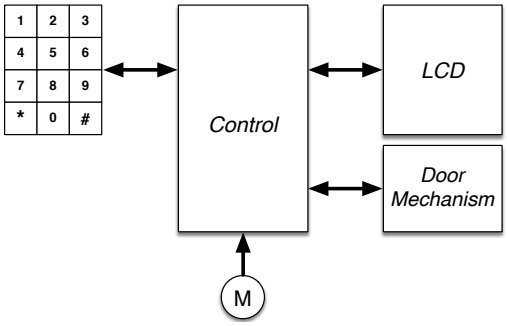


Figure 1 – Access Control System

The system allows the following actions in Access Mode:

- **Access –** To access the restricted zone, the user must input the three digits of its UIN, followed by the four digits of the PIN. If the UIN and PIN pair is correct, the system presents the name of the user and the stored message (if one exists) on the LCD. It then activates the Door Mechanism to open the door. The message must be removed from the system if the '*' key is pressed while it is shown on the LCD. Every access must be logged with in a file called *LogFile* (with one access record per line), including the time/date and UIN.
- **PIN change –** This action is performed if, after the authentication process, the user presses the '#' key. The system requests the user to enter the new PIN twice for confirmation. The new PIN is only registered in the system if the two inputs are identical.

Note: The process of information input from the keypad should use the following criteria: if no key is pressed during five seconds, the current command is aborted; if the '*' key is pressed and the system has already read some digits, all digits read so far are discarded, however, if the system has not read any digits so far, the '*' key aborts the current command.

The system also allows actions in Maintenance Mode. Differently from the Access Mode, the actions in Maintenance Mode are performed using the PC's keyboard and screen. The actions available in this mode are:

- **Adding a new user –** Adds a new user to the system. The system assigns the first available UIN and requests the name and PIN of the user from the system manager. The name of the user must be, at most, 16 characters long.
- **Removing a user –** Removes an existing user from the system. The system requests the system manager to input the UIN, shows the name of the corresponding user, and asks for confirmation.
- **Inserting a message –** Allows associating an information message addressed to a particular user that shall be presented to that user during the process of authenticating to access the restricted zone.
- **Shutdown –** Allows shutting down the Access Control System. The system asks the user to confirm the command and writes the user information to a text file (one user per line). This file is loaded when the software is launched and rewritten during the shutdown. The system must support up to 1000 users, which are added and removed by the system manager using the PC's keyboard.

Note: During the execution of the Maintenance Mode, no actions can be performed on the user keypad, and the LCD must display the message "*Out of Service*".

2 System Architecture

The system shall be implemented as a hybrid hardware-software solution, as shown in the block diagram of Figure 2. The proposed architecture is composed of four main modules: *i*) a module for reading keys from the keypad, designated *Keyboard Reader*; *ii*) a module for interfacing with the LCD, designated *Serial LCD Controller* (*SLCDC*); *iii*) a module for interfacing with the Door Mechanism, designated *Serial Door Controller* (SDC); and *iv*) a control module, designated *Control*. Modules *i*), *ii*) and *iii*) shall be implemented in hardware, while the *Control* module shall be implemented in software and run in a PC.

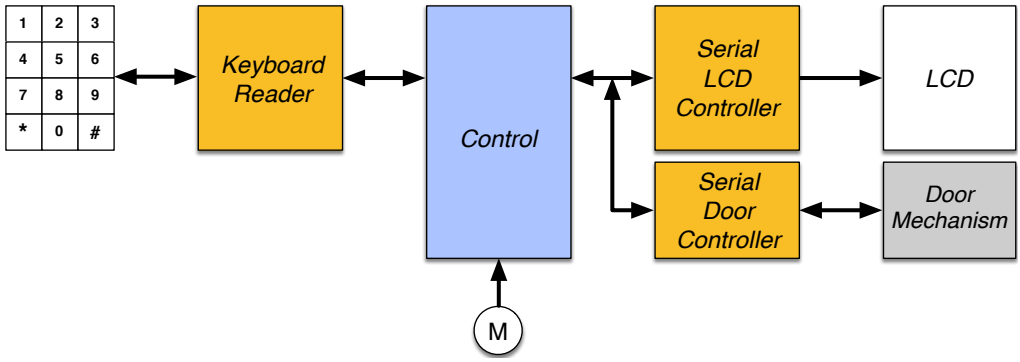


Figure 2 – Architecture for the implementation of the Access Control System

The *Keyboard Reader* module is responsible for decoding the 12-key matrix keypad, determining which key has been pressed and sending its corresponding 4-bit code to the *Control* (if the *Control* is available for reception). If the *Control* is not immediately available for reception, the key code is stored (up to nine stored key codes). The *Control* processes and sends to the *SLCDC* data regarding information to be displayed on the LCD. The information for the Door Mechanism is send through the *SDC*. Due to physical constraints, and to minimize the number of interconnection signals required by the system, the communication between the *Control* and the *SLCDC* and *SDC* modules uses a serial protocol.

2.1 Keyboard Reader

As shown in Figure 3, the *Keyboard Reader* module is composed of three main blocks: *i*) a decoder for the keypad (*Key Decode*); *ii*) a storage block (named *Ring Buffer*); and *iii*) a block for delivering the key code to the consumer (named *Output Buffer*). In this context, the *Control* module, implemented in software, is the consumer entity.

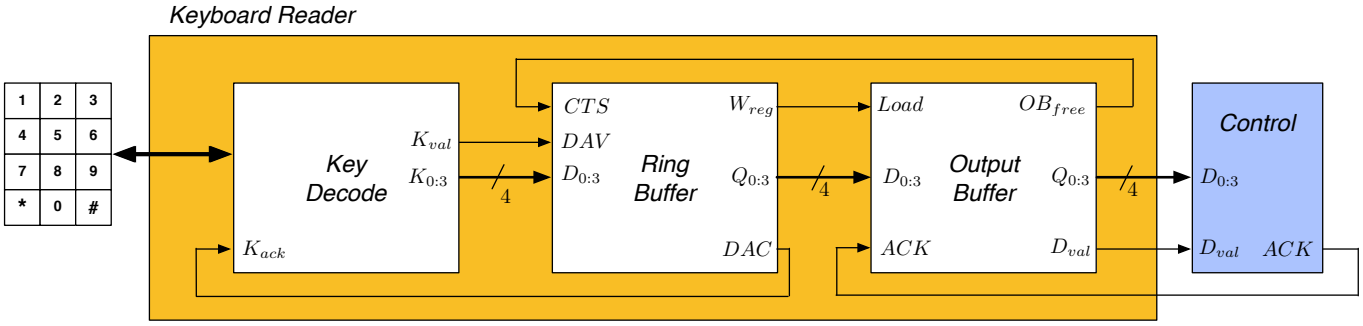


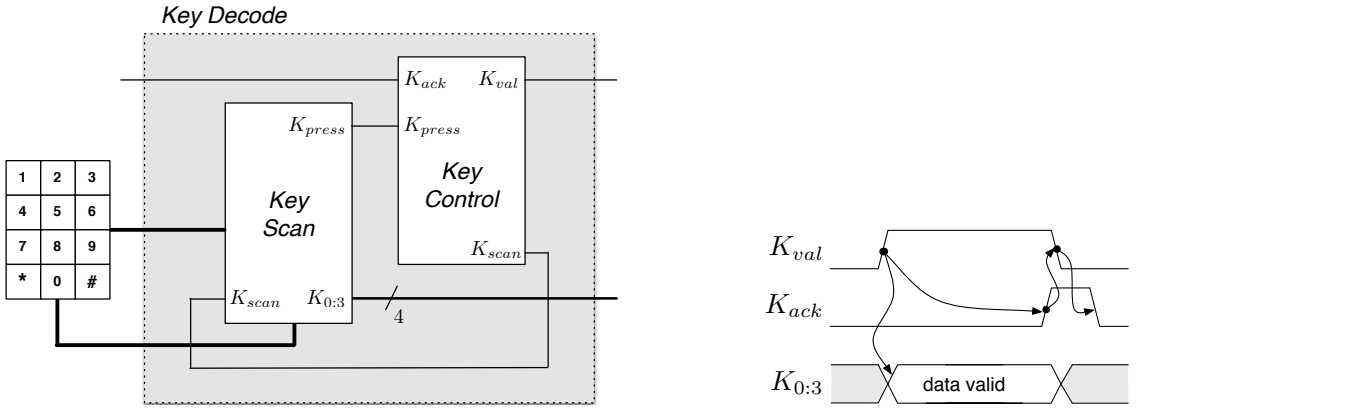
Figure 3 – Block diagram for the *Keyboard Reader*

2.1.1 Key Decode

The *Key Decode* block shall implement a decoder for a 4x3 matrix keypad in hardware. It is composed of three sub-blocks: *i)* a 4x3 matrix keypad; *ii)* the *Key Scan* block, responsible for scanning the keypad; and *iii)* the *Key Control* block, which controls the scanning and data flow, as illustrated in the block diagram of Figure 4a.

The flow control for the output of the *Key Decode* block (towards the *Ring Buffer* block) defines that the signal K_{val} is enabled when a key press is detected, while the code of the key is made available at the $K_{0:3}$ bus. A new keypad scanning cycle is only

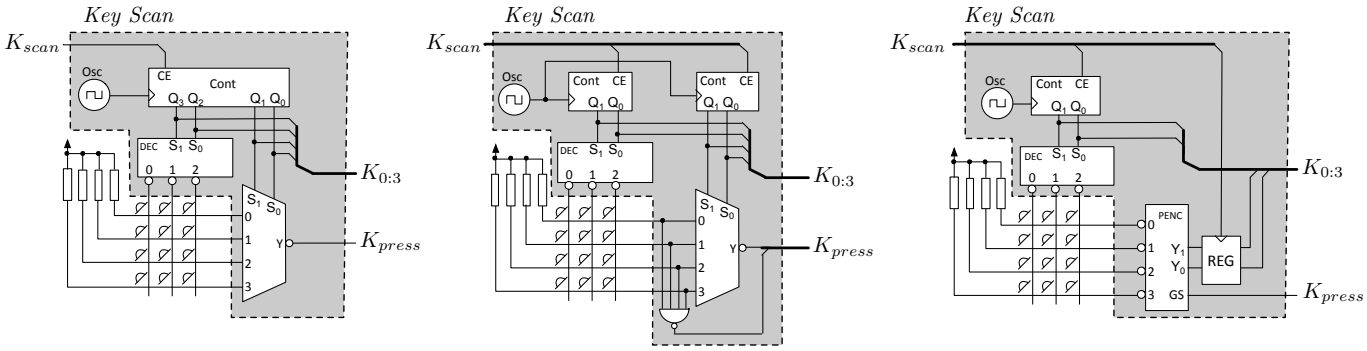
started when the K_{ack} signal is enabled, and the previous key is released by the user. The temporal diagram for this flow control is illustrated in Figure 4b.



a) Block diagram b) Temporal diagram

Figure 4 –*Key Decode* block

The *Key Scan* block shall be implemented according to one of the block diagrams presented in Figure 5, while the development and implementation of the *Key Control* block is left for analysis and study, with the proposal of its architecture being a responsibility of the students.



a) version I b) version II c) version III

Figure 5 – Block diagrams for the *Key Scan* block

2.1.2 Ring Buffer

The *Ring Buffer* block that must be developed as part of this project shall be a data structure for storing key codes based on a FIFO (*First In First Out*) policy, with capacity to store up to eight 4-bit words.

The process of writing data to the *Ring Buffer* starts with the activation of the DAV (*Data Available*) signal by the producer system – in this context, the *Key Decode* block –, indicating that it has data to be stored. As soon as it is available for storing the information, the *Ring Buffer* writes the data $D_{0:3}$ to the memory. Once the writing operation is concluded, it enables the DAC (*Data Accepted*) signal, informing the producer system that the data has been accepted. The producer system keeps the DAV signal enabled until DAC is disabled. The *Ring Buffer* only disables the DAC after the DAV is disabled.

The *Ring Buffer* implementation shall be based on a RAM (Random Access Memory). The write/read address, selected by means of the $put\overline{get}$ signal, shall be defined by the *Memory Address Control* (MAC) block. This block is composed of two registers which store the write and read addresses, called *putIndex* e *getIndex,* respectively. The MAC, thus, supports the *incPut* and *incGet* actions, and outputs flags to indicate if the buffer is full (*Full*) or empty (*Empty*). The *Ring Buffer* proceeds to deliver data to the consumer entity whenever it indicates that it is available for reception, which is done through the *Clear To Send* (CTS) signal. Figure 6 presents a block diagram of the structure of the *Ring Buffer* block.

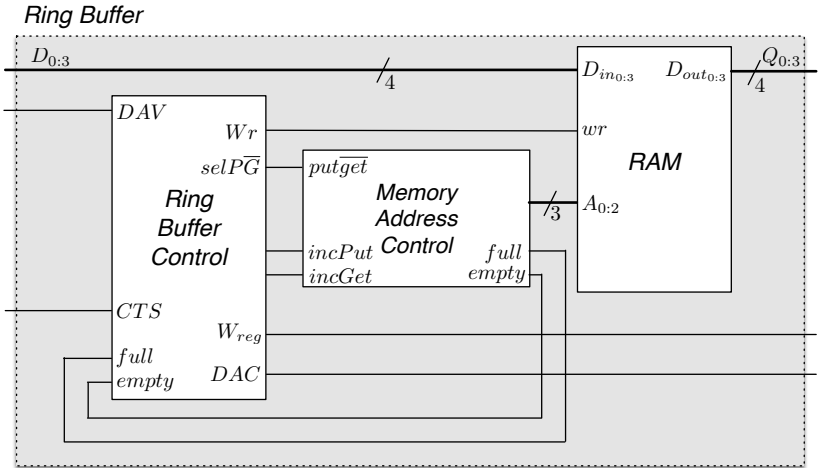


Figure 6 – Block diagram of the *Ring Buffer* block

2.1.3 Output Buffer

The *Output Buffer* block of the *Keyboard Reader* is responsible for the interaction with the consumer system – in this context, the *Control* module.

The *Output Buffer* indicates that it is available to store data through the OB_{free} signal. In that situation, the producer system can activate the *Load* signal to register the data.

When the *Control* wishes to read data from the *Output Buffer*, it waits until the D_{val} is enabled, collects the data and e pulses the *ACK* signal indicating that they have been consumed.

As soon as it receives the *ACK* pulse, the *Output Buffer* must invalidate the data by disabling D_{val} and signal that it is again available to deliver data to the consumer system by enabling the OB_{free} signal. Figure 7 presents the block diagram for the *Output Buffer* block.

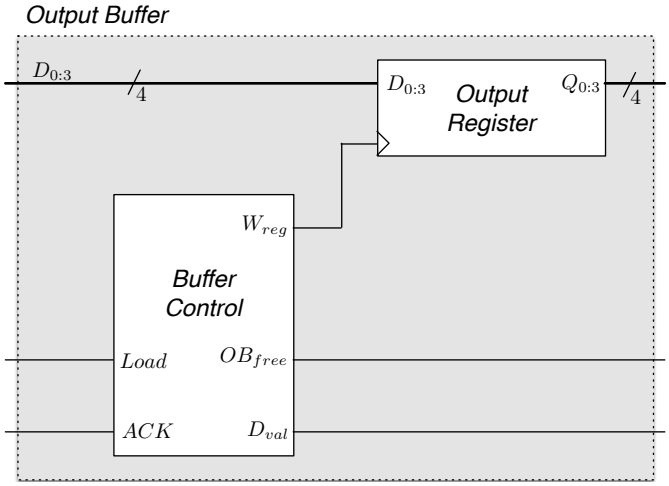


Figure 7 – Block diagram for the *Output Buffer* block

Whenever the producer block *Ring Buffer* has data and the delivery block *Output Buffer* is available (OB_{free} enabled), the *Ring Buffer* performs a memory read and delivers the data to the *Output Buffer* by enabling the W_{reg} signal. The *Output Buffer* indicates it has already registered the data by disabling the OB_{free} signal.

2.2 Serial LCD Controller

The module for interfacing with the LCD (*Serial LCD Interface, SLCDC)* implements the serial reception of the information sent by the *Control* module, delivering it afterwards to the *LCD*, as illustrated in Figure 8.

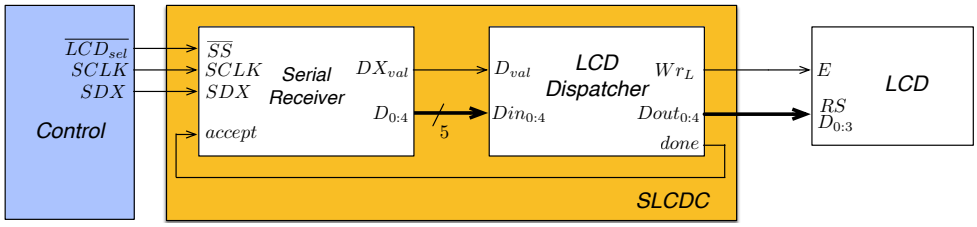


Figure 8 – Block diagram for the *Serial LCD Controller* block

The *SLCDC* receives a message in series composed of five bits of information. The communication with the *SLCDC* follows the protocol illustrated in Figure . The first bit of information, the *RS* bit, indicates if the message is for control or data. The remaining bits have the data to be delivered to the *LCD*.

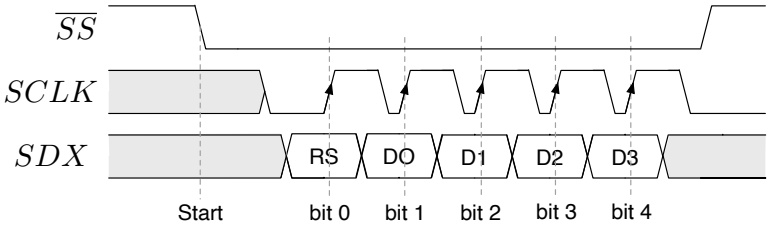


Figure 9 – Protocol for communication with the *Serial LCD Controller*

When the sender, implemented in software, wishes to transmit a frame to the S*LCDC*, it signals the frame start condition (*Start*), corresponding to a transition from high to low of the $\overline{LCD_{sel}}$ line. After the start condition, the S*LCDC* stores the frame bits at every rising edge of the *SCLK* signal.

2.2.1 Serial Receiver

The *Serial Receiver* block within the S*LCDC* is composed of three main blocks: *i*) a control block; *ii*) a counter for the number of received bits; and *iii*) a serial-to-parallel converter. Those are called, respectively, *Serial Control, Counter*, and *Shift Register*. The *Serial Receiver* shall be implemented based on the block diagram of Figure .

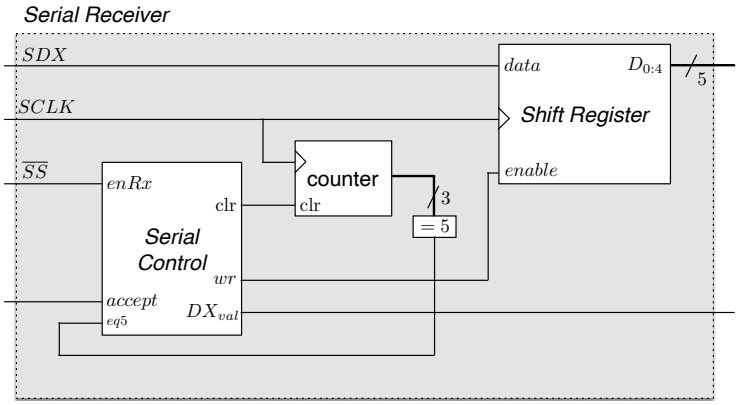


Figure 10 – Block diagram for the *Serial Receiver* block

2.2.2 Dispatcher

When the *Serial Receiver* block has received a valid frame, it signals that to the *Dispatcher* block by enabling the DX_{val} signal. In turn, the *Dispatcher* block delivers the frame the to the *LCD* by enabling the Wr_L signal.

The *LCD* processes the received frames according to the commands defined by the manufacturer, and it is not necessary to wait for the completion of the commands to free the serial communication channel. Thus, the *Dispatcher* may signal to the *Serial Receiver* that the frame was processed by enabling the *done* signal.

2.3 Serial Door Controller

The module that interfaces with the Door Mechanism (*Serial Door Controller, SDC)* implements the serial reception of the information sent by the *Control* module, delivering it afterwards to the Door Mechanism, as illustrated in Figure .

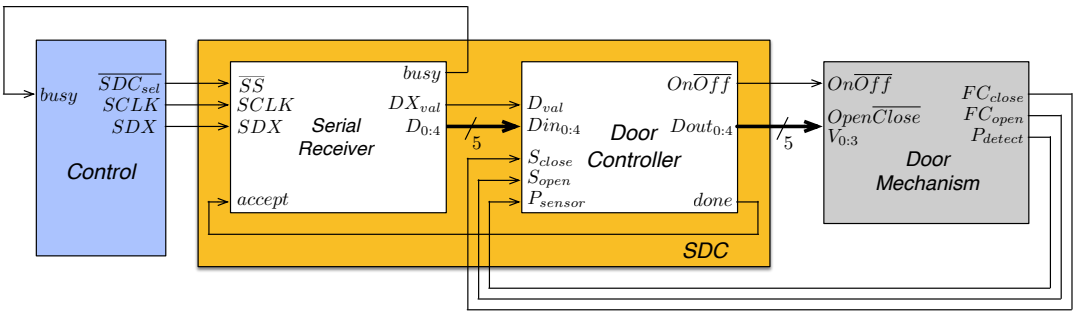


Figure 11 – Block diagram of the *Serial Door Controller* block

The *SDC* receives a message in series composed of five bits of information. The communication with the *SDC* uses the protocol illustrated in Figure 8. The first bit of information, the $Open\overline{Close}$ (*OC*) bit, indicates whether the command is for opening or closing the door. The remaining bits carry the information of the velocity for opening or closing. The *SDC* indicates that it is available for the reception of a new frame after processing the previous one by setting the *busy* signal to the logical level "0".

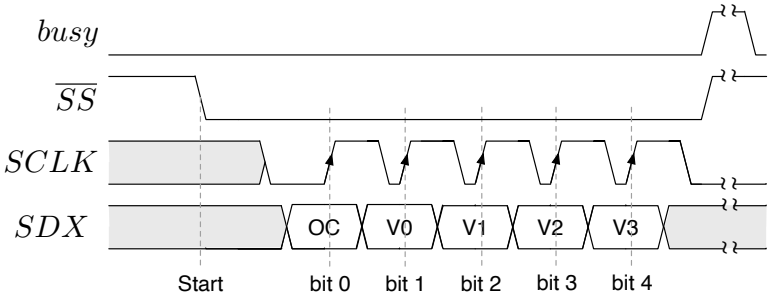


Figure 8 – Protocol for communication with the *Serial Door Controller*

2.3.1 Serial Receiver

The *Serial Receiver* block within the *SDC* shall be implemented with a similar structure to the *Serial Receiver* block developed for the *SLCDC* block.

2.3.2 Door Controller

After receiving a valid frame from the *Serial Receiver*, the *Door Controller* block shall proceed to execute the received command with the Door Mechanism. If the received command is for opening the door, the *Door Controller* shall enable both the $On\overline{Off}$ and $Open\overline{Close}$ signals, until the opened-door sensor (FC_{open}) becomes enabled. However, if the command is for closing the door, the *Door Controller* shall enable the $On\overline{Off}$ signal while the $Open\overline{Close}$ is disabled until the closed-door sensor (FC_{close}) becomes enabled. If, during the process of closing the door, a person is detected in the door path – through the presence sensor (P_{detect}) – the system shall interrupt the closing process, reopening the door. After the interruption, the *Door Controller* block shall automatically (in other words, without the need for sending a new frame) resume closing the door in order to finalize the previous command. Whenever a command is completed, the *Door Controller* signals to the *Serial Receiver* that it is ready to process a new frame by enabling the *done* signal.

2.4 *Control*

The *Control* module shall be implemented in *software*, using the *Kotlin* programming language and following the logical architecture presented in Figure 9.

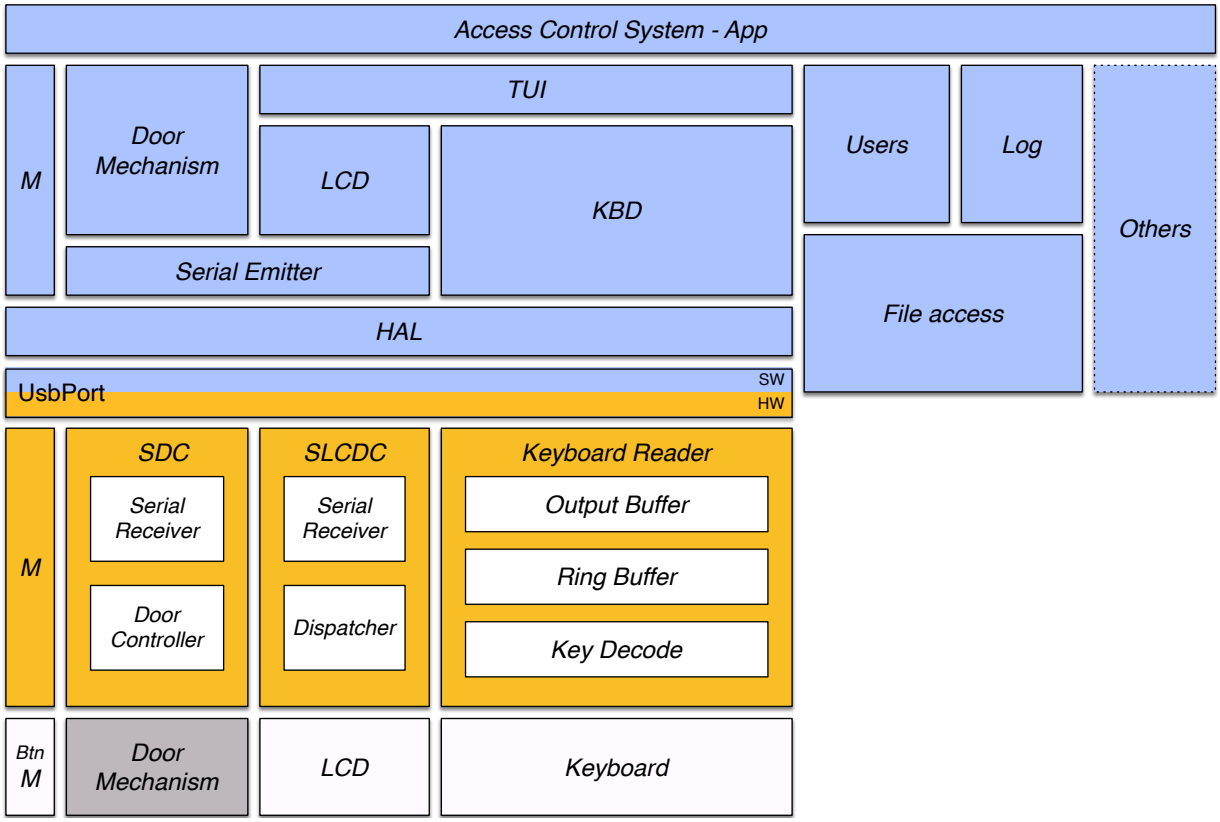


Figure 9 – Logical diagram of the Access Control System

The signatures of the main functions and objects that shall be developed are presented in the next sections. The remaining ones shall be analysed by the group, which is free to choose implementation details.

2.4.1 *HAL*

```
object HAL {   // Virtualizes the access to the UsbPort system
   // Initializes the class
   fun init() ...
   // Returns true if the corresponding bit has logical value ‘1’
   fun isBit(mask: Int): Boolean ...
   // Returns the value of the bits indicated by mask in the UsbPort
   fun readBits(mask: Int): Int ...
   // Writes value to the to the bits indicated by mask
   fun writeBits(mask: Int, value: Int) ...
   // Puts the bits indicated by mask at the logical value ‘1’
   fun setBits(mask: Int) ...
   // Puts the bits indicated by mask at the logical value ‘0’
   fun clrBits(mask: Int) ...
}
```

2.4.2 KBD

```
KBD {  // Read keys. Methods return ‘0’..’9’,’#’,’*’ or NONE.
   const val NONE = 0;
   // Initializes the class
   fun init() ...
   // Immediately returns the pressed key or NONE there is no key available.
   fun getKey(): Char ...
   // Returns the pressed key, if one becomes available before ‘timeout’ (represented in
   milliseconds), or NONE otherwise.
   fun waitKey(timeout: Long): Char ...
}
```

2.4.3 LCD

```
LCD {  // Writes to the LCD using the 4-bit interface.
   private const val LINES = 2, COLS = 16; // Dimensions of the display.
   // Writes a command/data nibble to the LCD in parallel
   private fun writeNibbleParallel(rs: Boolean, data: Int) ...
   // Writes a command/data nibble to the LCD in series
   private fun writeNibbleSerial(rs: Boolean, data: Int) ...
   // Writes a command/data nibble to the LCD
   private fun writeNibble(rs: Boolean, data: Int) ...
   // Writes a command/data byte to the LCD
   private fun writeByte(rs: Boolean, data: Int) ...
   // Writes a command to the LCD
   private fun writeCMD(data: Int) ...
   // Writes data to the LCD
   private fun writeDATA(data: Int) ...
   // Sends the initialization sequence for 4-bit communication.
   fun init() ...
   // Writes a character at the current position.
   fun write(c: Char) ...
   // Writes a string at the current position.
   fun write(text: String) ...
   // Sends a command to position the cursor (‘line’:0..LINES-1 , ‘column’:0..COLS-1)
   fun cursor(line: Int, column: Int) ...
   // Sends a command to clear the screen and position the cursor at (0,0)
   fun clear() ...
}
```

2.4.4 SerialEmitter

```
SerialEmitter {       // Sends frames for the different Serial Receiver modules.
   enum class Destination {LCD, DOOR}
   // Initializes the class
   fun init() ...
   // Sends a frame with ‘data’ to the SerialReceiver identified as destination in ‘addr’
   fun send(addr: Destination, data: Int) ...
   // Returns true if the series communication channel is busy
   fun isBusy(): Boolean ...
}
```

2.4.5 DoorMechanism

```
DoorMechanism {       // Controls the state of the Door Mechanism.
   // Initializes the class, establishing initial values.
   fun init() ...
   // Sends a command to open the door with the given velocity parameter
   fun open(velocity: Int) ...
   // Sends a command to close the door with the given velocity parameter
   fun close(velocity: Int) ...
   // Checks if the previous command has finished
   fun finished() : Boolean ...
}
```

3 Project Schedule

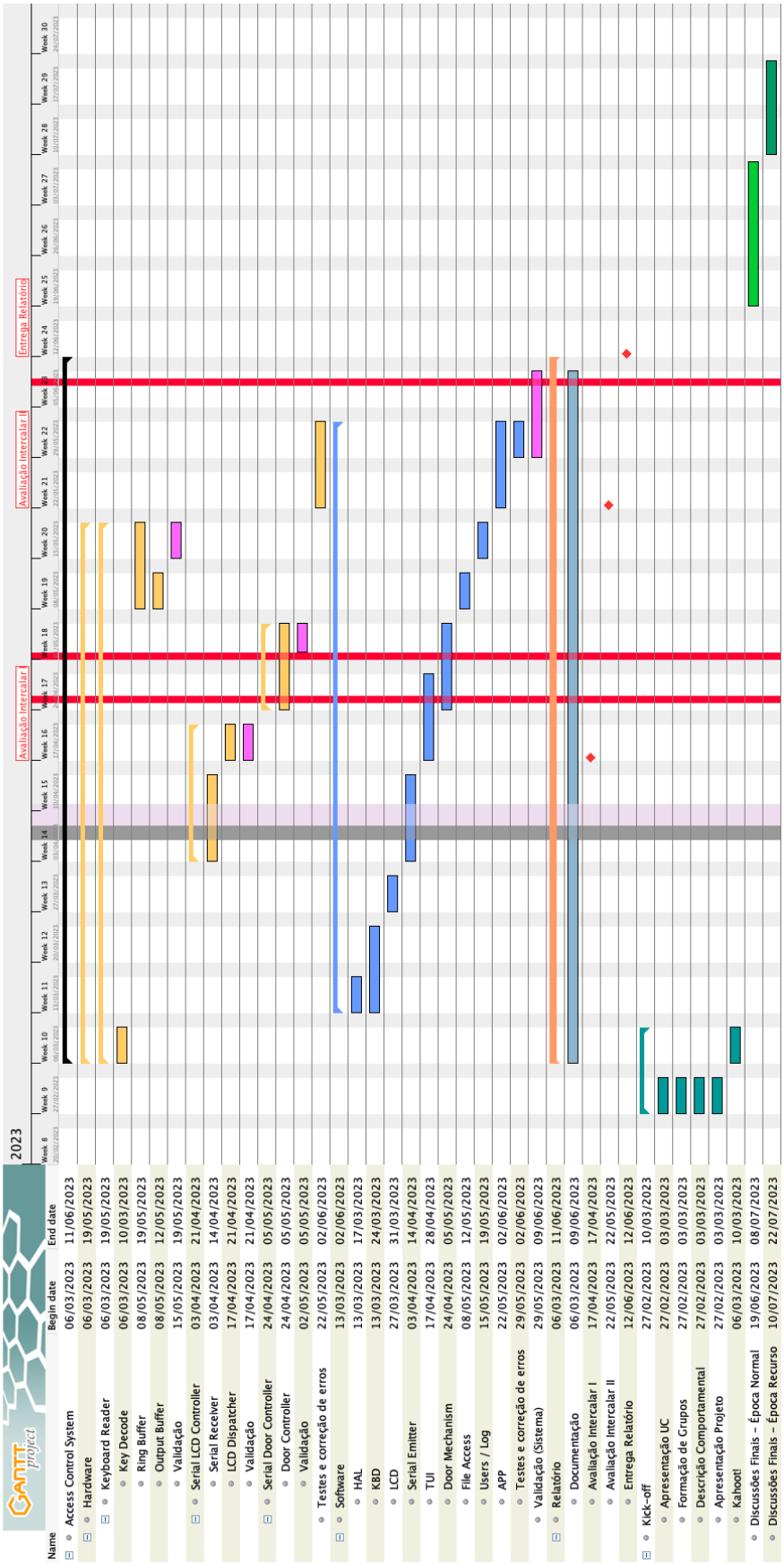


Figure 10 – Gantt diagram of the project schedule