2023

Elevator

Digital System Design project

Sofia Dobra & Laurenţiu-Călin Grad
DRD. ENG. DIANA IRENA POP

Table of Contents

Elevator

1 Specifications

Design an automaton that commands an elevator for a 13-storey building. The elevator should work at two selectable speeds: (approximately) 1 or 3 seconds per floor. It should be able to accomplish requests given at any time from outside (up/down) or from inside following a given algorithm. The elevator must operate as follows:

- In the initial state the elevator waits at the ground floor with the doors open.
- The priority of the requests is computed by taking into account the floor number, the direction the elevator is moving (up/down) together with the direction of the call.
- The doors stay open for 10 seconds (if the elevator is not in the idle state at the ground floor).
- The elevator does not start moving if its doors are open or if the weight of the people inside is greater than maximum admissible weight. Also, if somebody is between the doors of the machine, the doors don’t close.

The mass and door sensors are provided. We assume that a clock signal of 100 MHz exists.

2 Design

2.1 Black Box

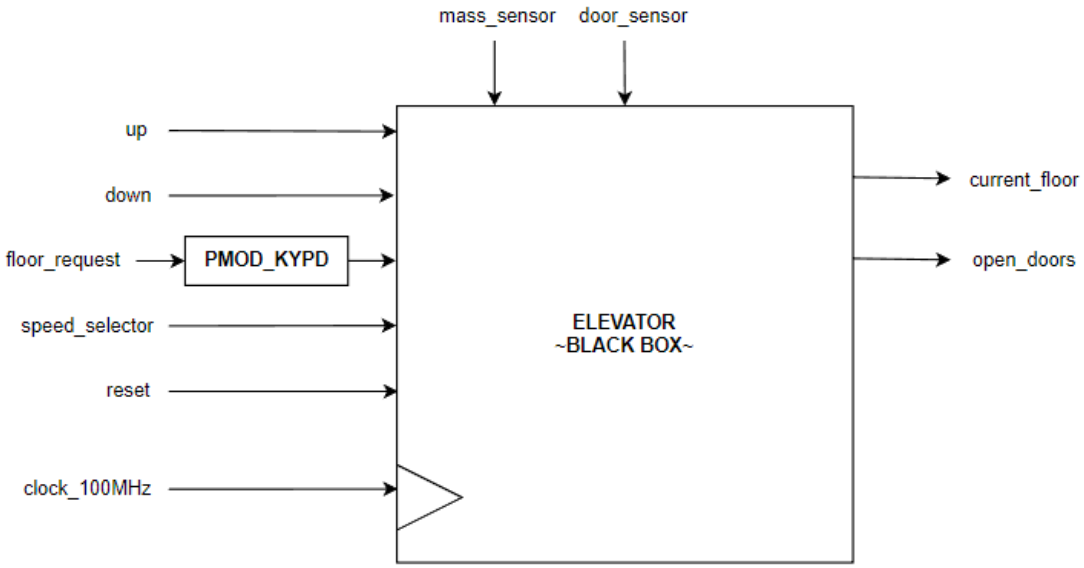


Figure 1. Black Box of the system

2.2 Control and Execution Unit

The system's black box must be further broken down in order to find implementable components. We will do a top-down breakdown of the problem until we get to known circuits, and then we will implement bottom-up.

2.2.1 Mapping the inputs and outputs of the black box on the two components

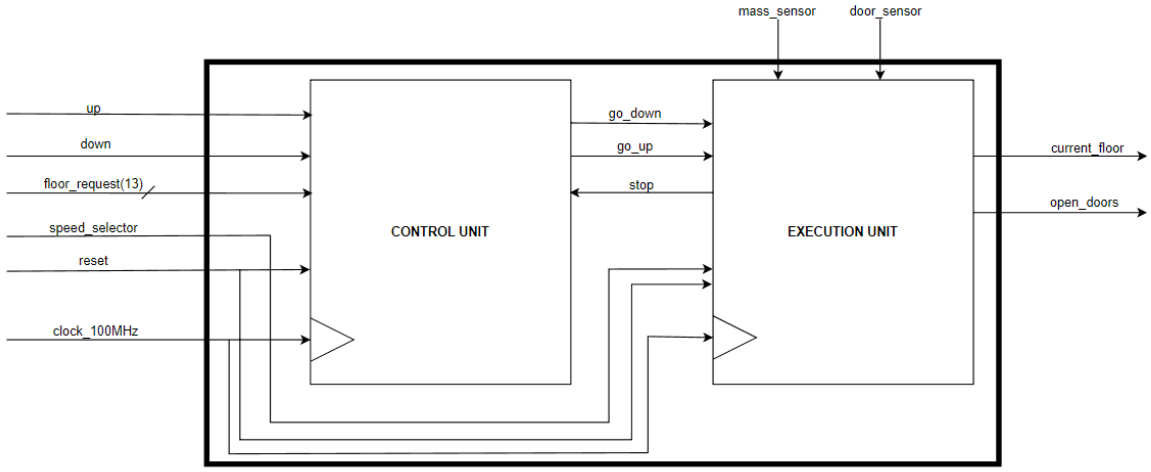


Figure 2. Mapping the inputs and outputs of the black box on the inputs and outputs of the units

2.2.2 Resources (breakdown of the Execution Unit)

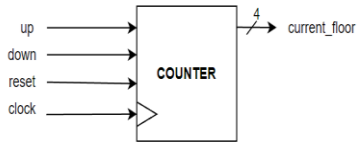
To further establish the links between the CU and the EU, we must first identify the resources on the basis of which we make decisions. These resources can generate signals to the control unit and can be controlled by the CU via Enable or Reset signals.

Any decision-making information must come from a resource that generates that information and passes it on to UC.

Resources can be simple circuits, which can be implemented directly (counter, register, etc.) or complex resources (remainder algorithm, multiplication algorithm, etc.). These complex resources may appear in the first breakdown with black boxes to which we must establish inputs and outputs, but later they must be further broken down (usually also in CU and EU) until we reach known circuits.

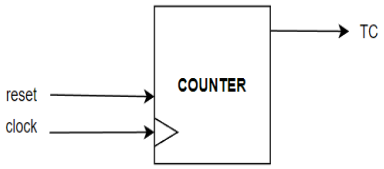
i. Bidirectional counter

This is a 4-bit counter designed in such a way that the other components should never let it reach values greater than 12. This circuit will have as output its state (a 4-bit number) corresponding to the current floor of the elevator. In order for the elevator to move, one of the inputs up/down (provided by the control unit) must be 1, otherwise the counter remains in the same state. Moreover, we provided an asynchronous reset. The clock of this system depends on the user-selectable speed.



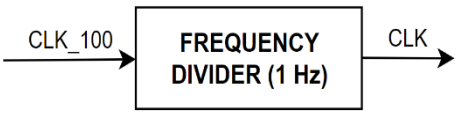
ii. 3-second / 10-second counter

This circuit functions as a frequency divider of the provided 100 MHz clock and is required for the slower speed. The output TC ("terminal carry") signals when approximately 3 / 10 seconds have passed. It is designed with a supplementary asynchronous reset input.



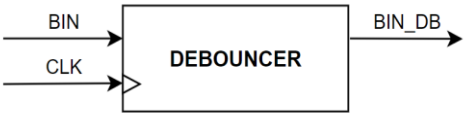
iii. Frequency Divider to 1 / 3 seconds

This resource acts as a frequency divider of the provided clock (100 MHz), bringing the frequency down to 1 Hz. It is used for the faster (1 second/floor) speed.



iv. Debouncer

This resource is connected to all the inputs of type *"button"*. Its functionality is based on cascaded *Delay Flip-Flops* – this resource extends the signal received from a button, deciding if we take it into consideration (it is *"long"* enough) or not.



v. T Flip-Flop

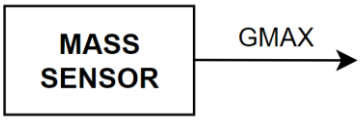
This resource is used in establishing the direction of the elevator. When the input T is 1, the value stored inside the flip-flop is toggled when a rising edge is encountered.

vi. 7-segment Display (SSD)

This resource controls the 7-segment display on the board – it displays the floor number (4-bit number) in decimal format. The outputs – cathodes and anodes – are active low signals. Since we have only 7 cathodes (each one corresponding to a segment from *a* to *g*) and we must display a 2-digit decimal number, we will activate the corresponding anodes successively at a great frequency (more than 70 Hz).

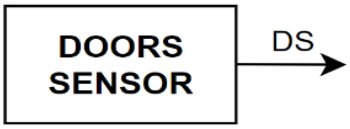
vii. Mass Sensor

This resource has one output *GMAX*, active when the maximum weight is exceeded.



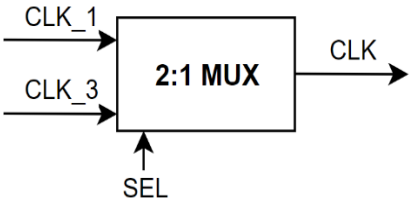
viii. Doors Sensor

This resource has one output *DS*, which signals if somebody is between the doors of the elevator.



ix. Speed Selector

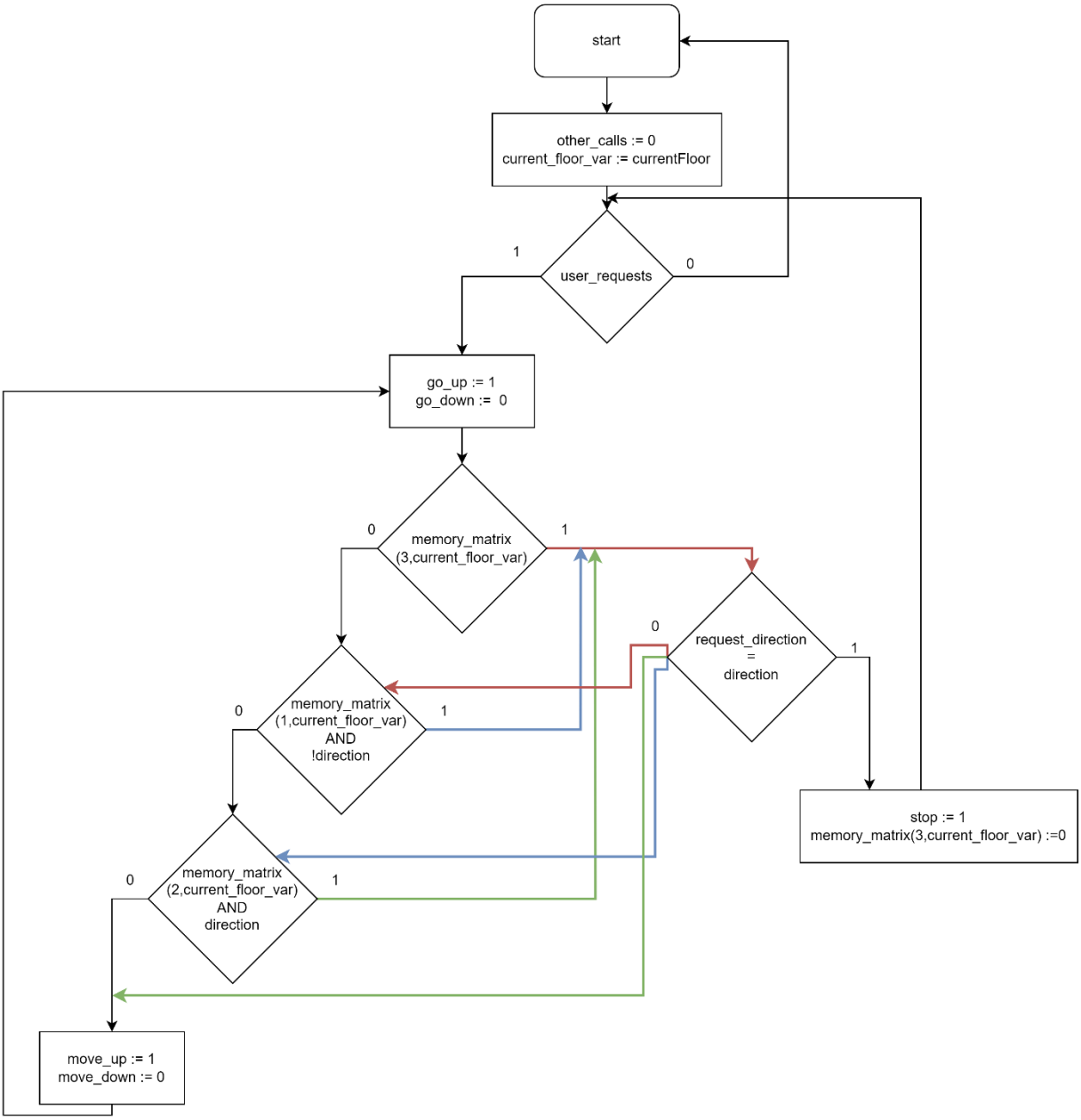
This resource is a *2:1 Multiplexer*, selecting between 2 signals – one with the frequency of 1 Hz, the other one with the frequency of 1/3 Hz.



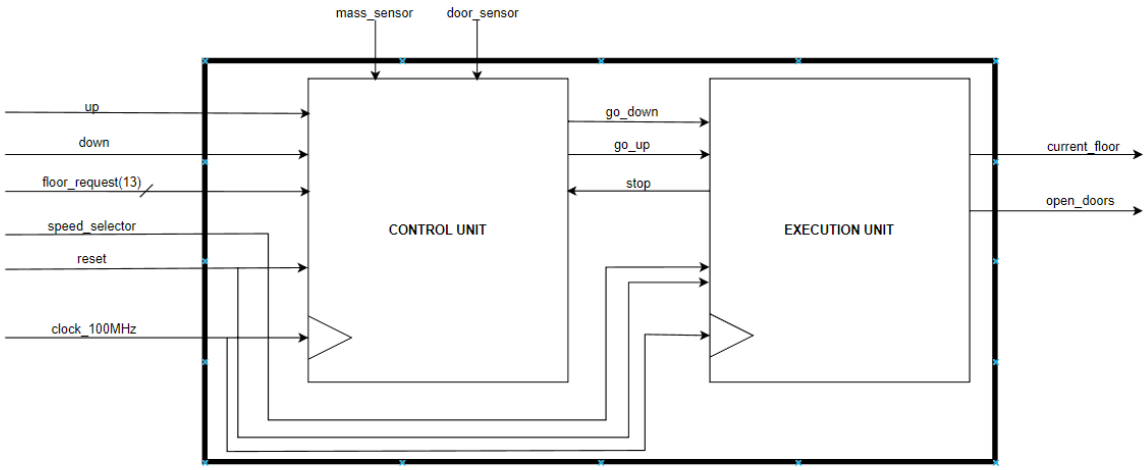
x. PMOD-KYPD Input Controller

This is a provided component that has 16 buttons (floor numbers) used to simplify the data input for the user.

xi. Next floor algorithm flowchart

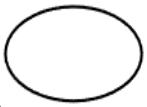


2.2.3 Block Diagram for first breakdown



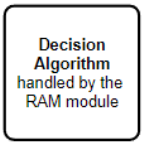
2.2.4 State diagram

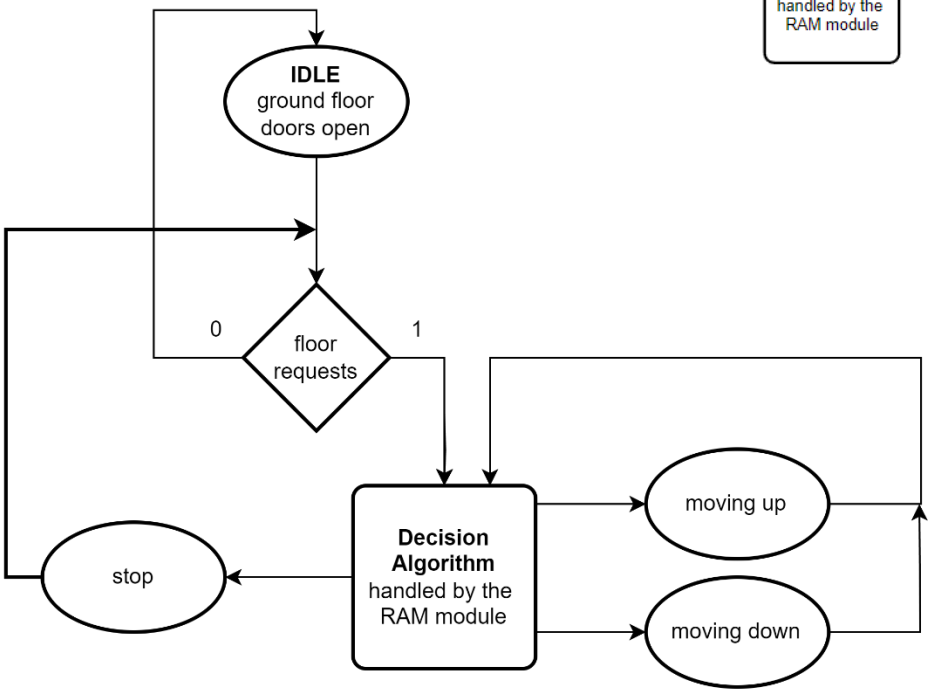
A state represents a moment of time (a period) and is represented by:



The decisions made in each state are represented by:

The decision algorithm is represented by:





2.2.5 Detailed diagram of the project

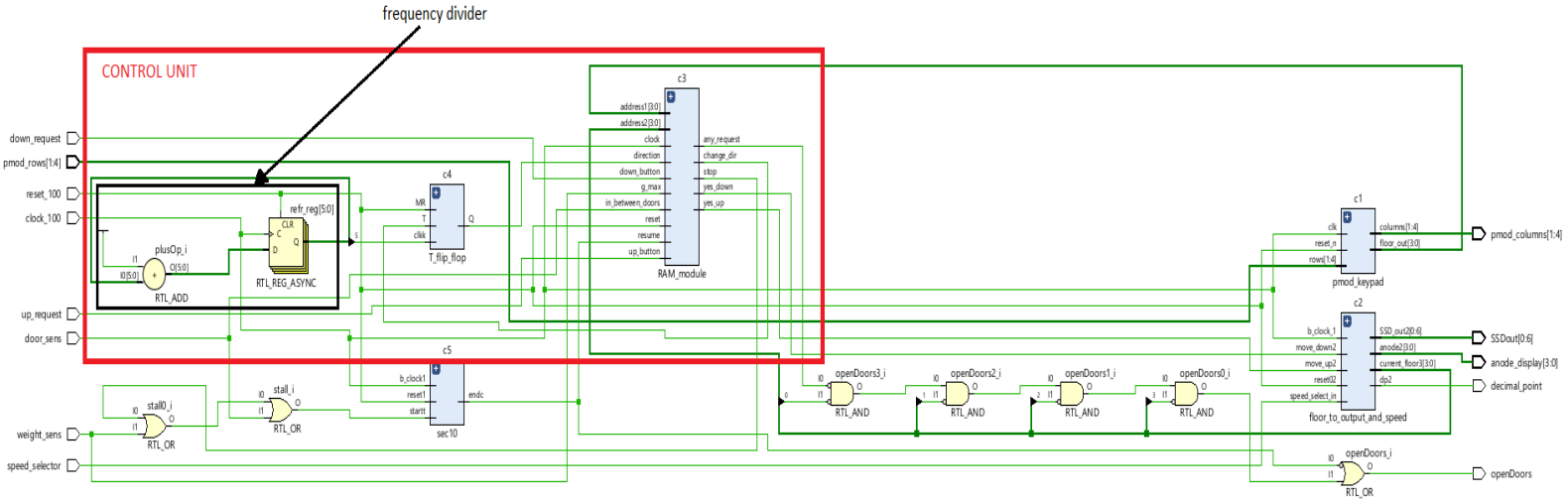


Figure 3. Detailed diagram of the project (VIVADO)

On this diagram the Control Unit does not appear as a single component because it proved to be easier to debug the automaton in this form. However, the control and the execution unit can be represented (structurally) as 2 unique entities.

The control unit consists of 2 sequential components: the T flip-flop and the RAM module. The flip-flop is used to store the direction in which the elevator is going (0 = up, 1 = down), information used by the elevator when deciding the next floor where to stop. The RAM module acts both as a memory and as a processing unit; here the inputs are processed, stored and updated based on the movement of the elevator. In addition to that, this part is responsible for the management of the requests (based on its internal state and the signals sent by the execution unit) and for the update of the direction (flip-flop) when needed. During the debugging process, we observed that if both the flip-flop and the RAM module are connected to the clock of the board (100 MHz) the bit stored inside that flip-flop is not consistent. To solve this problem, we decided to pass the board clock through a frequency divider.

The execution unit is composed of several medium scale integrated circuits grouped in 3 big components: *floor_to_output_and_speed* is responsible for the elevator speed (1 or 3 seconds / floor); it keeps track of the floor where the elevator is and, as a result, it controls what digits are shown of the seven segment displays. The *pmod_keypad* is the part that converts the input from the buttons of a digilent PMOD KYPD into signals easier to manage (bitstreams). This component is provided with button debouncers. The sec_10 is a clock that counts 10 seconds keeping the doors open in the meantime each time the elevator must stop at a requested floor. The logic gates are used to express concurrently (based on logical assumptions) when the doors have to be open and when the door counter has to start.

3 User manual

FPGA implementation – minimal board requirements:

- 1 PMOD connector
- 1 PMOD-KYPD
- 6 switches
- 2 LEDS

We implemented our projects on Digilent Artix-7 family boards: Basys 3 and Nexys 7A. We will use the Basys 3 to explain how the elevator works because it is more common.

Preparing the board for testing:

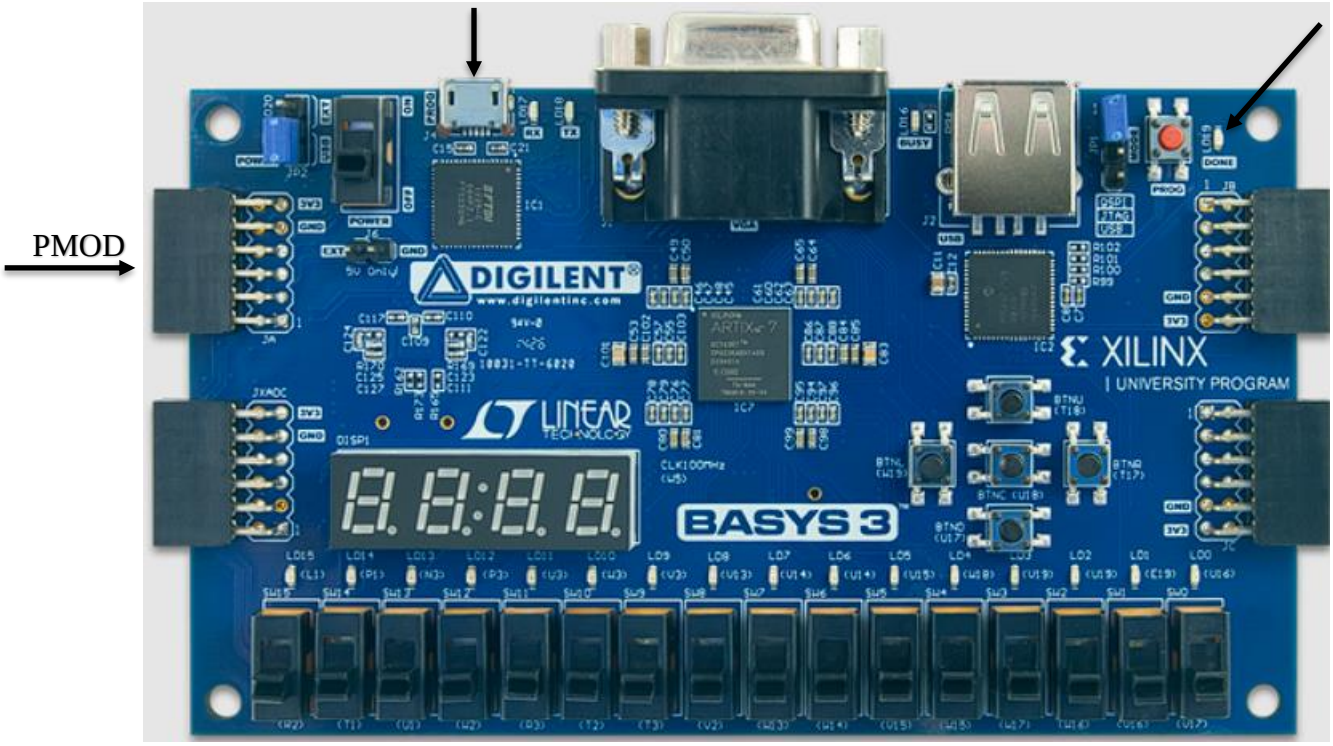
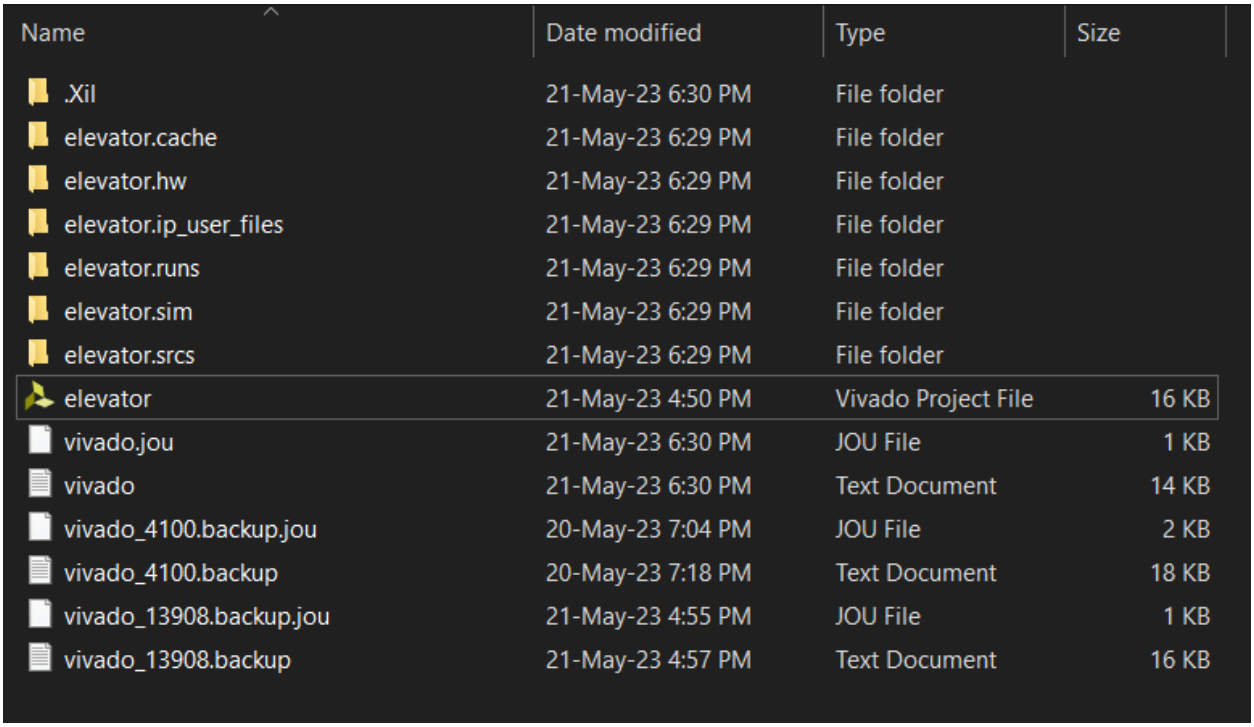


Figure 4. Basys 3 board

The PMOD-KYPD is inserted in the JA adapter (top-left corner). We connect the board to the computer via a USB 3.0 to micro-USB (inserted in the upper part of the board) and we power it on by lifting the top-left switch; the done LED (top-right) should become green after a few moments.

We open the project via Xilinx-Vivado environment by using the shortcut provided in the folder.



Name	Date modified	Type	Size
.Xil	21-May-23 6:30 PM	File folder	
elevator.cache	21-May-23 6:29 PM	File folder	
elevator.hw	21-May-23 6:29 PM	File folder	
elevator.ip_user_files	21-May-23 6:29 PM	File folder	
elevator.runs	21-May-23 6:29 PM	File folder	
elevator.sim	21-May-23 6:29 PM	File folder	
elevator.srcs	21-May-23 6:29 PM	File folder	
elevator	21-May-23 4:50 PM	Vivado Project File	16 KB
vivado.jou	21-May-23 6:30 PM	JOU File	1 KB
vivado	21-May-23 6:30 PM	Text Document	14 KB
vivado_4100.backup.jou	20-May-23 7:04 PM	JOU File	2 KB
vivado_4100.backup	20-May-23 7:18 PM	Text Document	18 KB
vivado_13908.backup.jou	21-May-23 4:55 PM	JOU File	1 KB
vivado_13908.backup	21-May-23 4:57 PM	Text Document	16 KB

In Vivado, we search and click on the “Generate Bitstring” option in the left menu. A loading window should appear. This process will take about 30 seconds. We click the background option provided by the previous pop-up window. In the meantime, we open the “Hardware manager” and click on “Auto-connect”. At this moment, Vivado should be able to interact with the board.

When the synthesis and implementation process is finished and the bitstream is generated a pop-up window will appear asking if we want to download the design onto the board. We accept it, the “done” LED should turn off for a couple of seconds. When the LED turns green again, the design is ready to be tested.

NOTE: Do not disconnect the board from the computer at any time as it is powered through the USB cable. If the cable is removed at any time, the board will reset itself.

<u>Testing the design:</u>

The rightmost 4 switches are (from left to right): RESET, SPEED_SELECTOR, DOWN_CALL and UP_CALL.

- RESET → resets all components of the design (frequency dividers, flip-flops etc.) except the memory (we wanted to use the pre-existent memory module implemented on the board and not let vivado software create a substitute using flip-flops; to do that we had to give up the reset of the memory).

- SPEED_SELECTOR → decides at which speed the elevator moves. Lifting the switch up (powering on) means the selected speed is 3 seconds from one floor to the next one, in its default state the speed is 1 second/floor.
- DOWN_CALL and UP_CALL → act as replacements for the exterior buttons of the elevator. Suppose when one (and only one) of these switches is up the input from the keypad is regarded as an exterior input from the chosen floor with a direction (the destination is above or below the floor where the request is made).

The leftmost 2 switches control the input from the sensors: the mass sensor and the door sensor (from left to right). We decided to separate them because in a real implementation the sensors should not be accessed by the user but be internal signals of the design provided by external circuits. These switches simulate the behavior of a sensor: powering them on means the sensor triggers a signal. Their default state represents that the elevator can move freely (no one is in between the doors, nor the elevator is overweight). If both switches are off and any PMOD button is pressed, the input is regarded as coming from inside the elevator.

The PMOD is used to simplify the user-machine interaction. The elevator we are trying to simulate is compatible with a 13-storey building so the available keypad buttons (representing floors) are 0 = ground floor, 1, 2, 3, 4, 5, 6, 7, 8, 9, A = 10, B = 11, C = 12, all the others do not influence the design (they do nothing). In addition to that, illegal input (up request form the 12th floor or down request from the ground floor) does not disturb the functionality. Any button can be pressed at any time.

The rightmost LED is used to signify that the doors are open.

Now, the design is ready for testing. Create an input and feed it to the machine via the keypad and check the order the requests are answered.

4 Technical justifications for the design

We tried to create the simplest design we could imagine starting from the “heart” of our automaton – the bidirectional counter. From there we expanded the execution unit until, improving its functionality as it grew larger. Once we were able to show digits on the seven-segment display, control the speed of the elevator and make it go up or down we started to work at the control unit. We agreed to use a RAM memory and a flip-flop to store the direction the elevator was moving. Even though we planned to keep the design as clear and as one-task-focused as possible (we did not have much success in doing that when implementing the execution unit) we decided to transform the RAM component into a module that is able to perform more than storing data. The RAM module has become the “brain” of the machine taking care of the internal control signals as well as the input ones. Some synchronization problems appear along the way, but we managed to solve them.

As far as VHDL goes, we combined structural and behavioral description everywhere throughout the design.

5 Future developments

Some possible future developments we thought about are:

- Adapting the design such that it supports an emergency button that stops the elevator regardless of what it is doing.
- Generalizing the design for a programmable number of floors (using parameters for the number of floors, speeds, the time duration when the doors are open).
- Updating the way floors are shown to the user: an arrow pointing up or down when the elevator is in between floors.
- Optimizing the energy consumption: the current design forces the elevator to go to the ground floor each time there are no requests using energy for this not so necessary operation (although, it may be useful for some particular case of building).
- Optimizing the request algorithm based on particular cases (algorithms for elevators that operate in buildings depend on the number of elevators functioning, the average number of requests per floor, the energy consumption etc.)

6 References

R1. Digital System Design – lectures & laboratories
R2. Logic Design – lectures & laboratories
R3. Octavian Creț, Lucia Văcariu, "Logic Design Problems -for digital systems-", U. T. Press, Cluj-Napoca, 2013
R4. https://docs.xilinx.com/r/en-US/ug912-vivado-properties/PIN
R5. https://forum.digikey.com/t/keypad-pmod-controller-vhdl/13134
R6. https://www.xilinx.com/support/documents/sw_manuals/xilinx2022_1/ug903-vivado-using-constraints.pdf
R7. https://support.xilinx.com/s/topiccatalog?language=en_US
R8. VHDL RAM: VHDL Single-Port RAM Design Example | Intel
R9. Stack Overflow - Where Developers Learn, Share, & Build Careers
R10. Intel® FPGA Design Examples | Intel
R11. Microsoft Word - Digilent Pmod Interface Specification.doc