**FIFO Memory**

DSD first year project

25.05.2023

Balogh Flóra

Project supervisor: Ing. Diana Pop

[**1.Specifications 3**](#_qpk7cyrxoykt)

[**2. Design 4**](#_ii6kmujc092q)

[2.1 Black Box 4](#_k8tgms7363l3)

[2.2 Block Scheme 5](#_5mamb45yvtxk)

[2.2.1 The State Diagram of the C.U. 7](#_oomy0b4hbjba)

[2.3 Resources 8](#_ja9y8pzadzko)

[2.3.1 AND-Gate 8](#_g83tt6y42jxe)

[2.3.2 4-bit Reversible Counter 9](#_wo4tkonxmz20)

[2.3.3 RAM Memory 10](#_ktlc3f126fh9)

[2.3.4 BCD-to-SSD 10](#_4j0y375o3617)

[2.3.5 Frequency divider 11](#_n93v8c78oy)

[2.3.6 Input decoder 11](#_fsn2npgjhjit)

[2.3.7 2:1 Multiplexer 12](#_r2j0yp9xzmgp)

[2.3.8 Debouncer 12](#_siw468osn4j6)

[**3.Internal Scheme 13**](#_wpyr9wnj58i4)

[**4. User Manual 15**](#_v23u9qnysf97)

[**5.Technical justifications for the design 16**](#_y2lbxvybefha)

[**6.Future developments 17**](#_7mfcdsxjfejv)

[**7.References 18**](#_a67042cxs7lk)

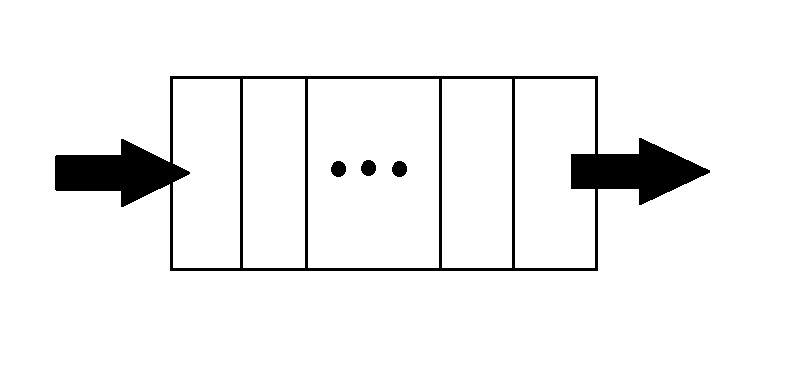
# **1.Specifications**

*“ Implement the* ***FIFO memory*** *according to the existing documentation in the book “Designing digital systems using FPGA technology” by S.Nedevschi, Z. Baruch and O. Cret, on page 238 of the book. The project will be carried by one student. ”*

**FIFO** is an [acronym](https://en.wikipedia.org/wiki/Acronym) for **first in, first out** (the first in is the first out), a method for organising the manipulation of a data structure, where the oldest (first) entry, or "head" of the [queue](https://en.wikipedia.org/wiki/Queue_(data_structure)), is processed first. This data will be stored in a synchronous RAM memory, which makes the implementation much more efficient. This eliminates also the need for registers and makes the project much smaller.

This will be a dual port implementation, which makes simultaneous writing and reading available.

The size of this memory will be 16 x 32.



*Figure 1.1 : Example for FIFO memory*

# **2. Design**

## **2.1 Black Box**



*Figure 2.1 : The black box of the memory*

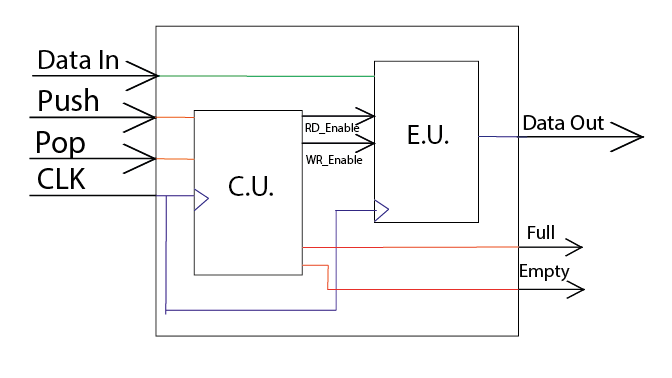
**Inputs:**

* “Data in” : data input, represents the data specified by the user which will be inserted in the memory
* “Push”: control input, mode for inserting the data
* “Pop”: control input, mode for reading from the memory
* “CLK : clock

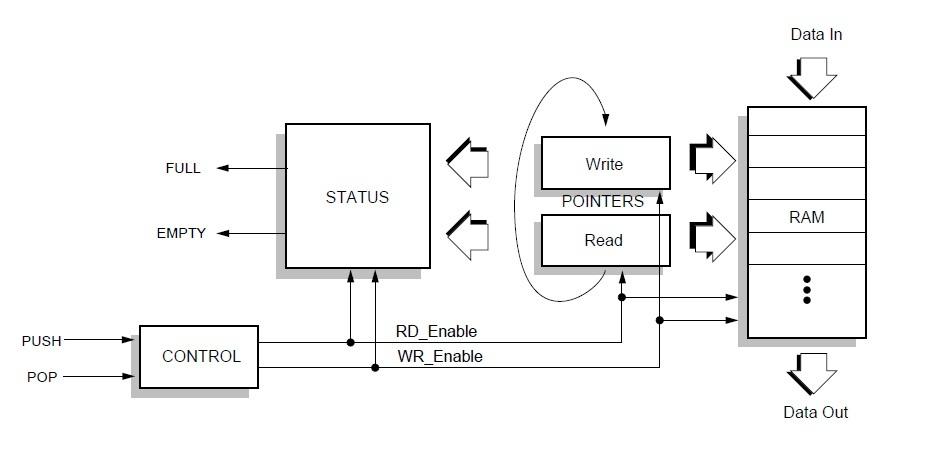
**Outputs**:

* “Data out”: data output, represents the data read from the memory
* “Full” : control output, tells the user if the memory is full
* “Empty” control output, tells the user if the memory is empty
* “Last” control output, tells the user when only one empty space is left in the memory

## **2.2 Block Scheme**



A classic FIFO memory consists of four different logical blocks, as it is shown in Figure *2.2* :



*Figure 2.2 : The block scheme of the FIFO memory*

* The control block:

This block generates validation signals for writing and reading.

* The State block:

Tells the state of the memory, if it is full, empty or has only one empty space left.

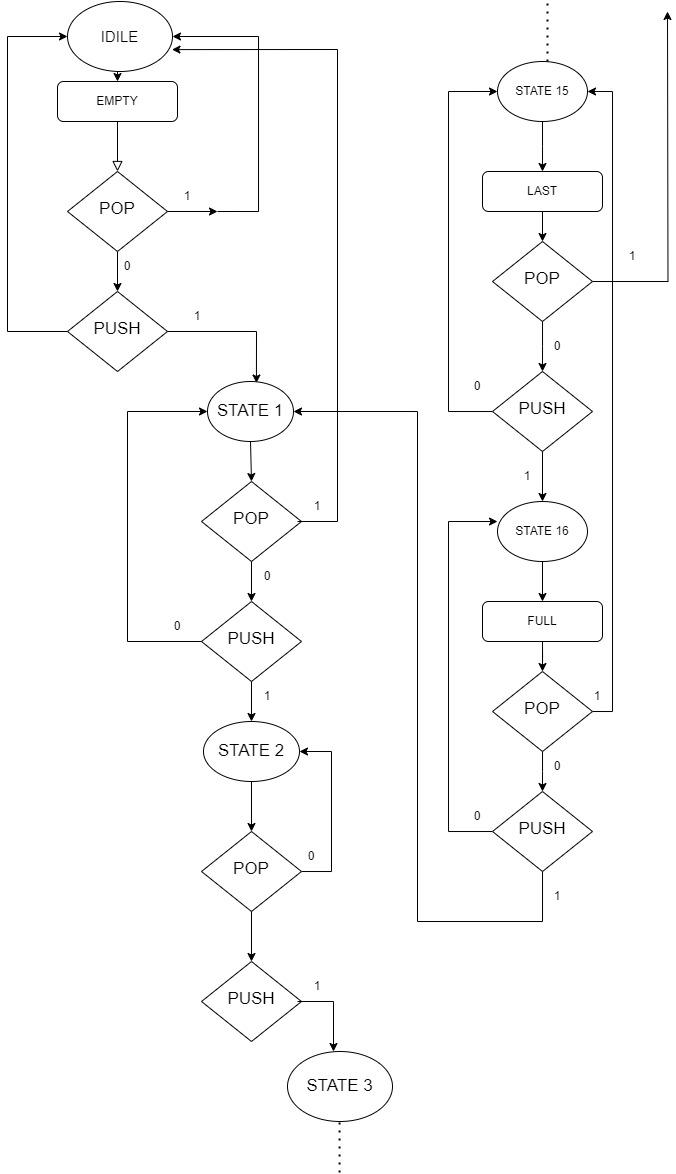
* Write/Read Blocks
* RAM Memory:

The actual memory part where the data is being stored.

### 

### 

### **2.2.1 The State Diagram of the C.U.**

**

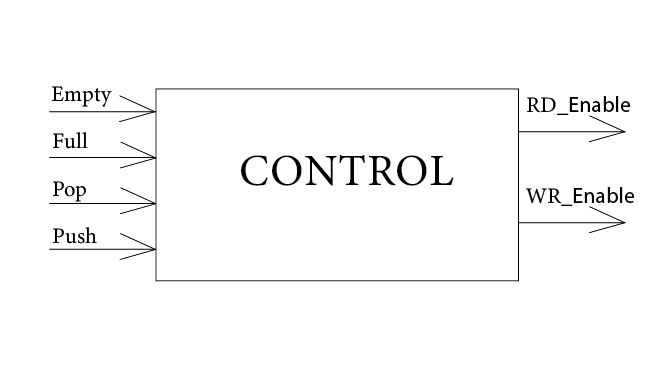
*Figure 2.2.1 : The State Diagram of the Control Unit*

This memory consists of seventeen different states. The first state represents the empty memory, when the output that tells the user that the memory is empty lights up. In State 15 the “LAST” output and in State 16 the “FULL” lights up. This diagram also tells when the states alternate with one another, depending on whether “POP” or “PUSH” has been pushed. When it reaches the last state and “PUSH” is pushed, the circuit goes back to State 1. The Control Unit is built according to this diagram.

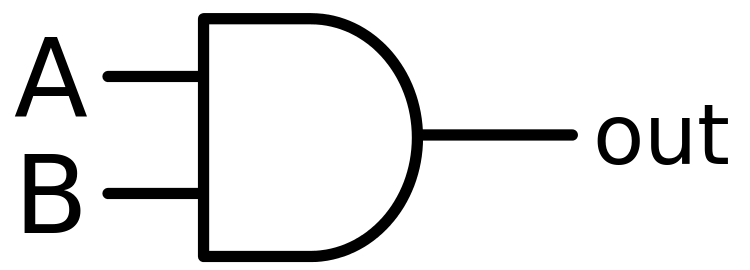
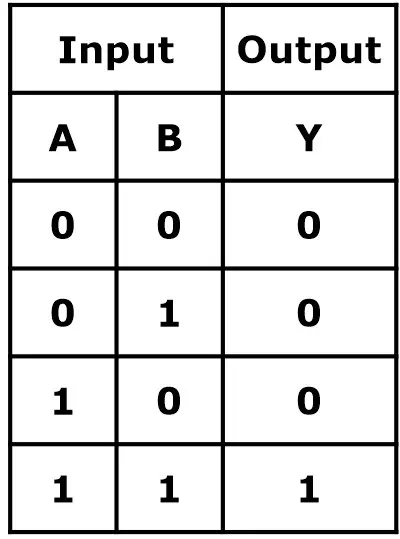
## **2.3 Resources**

### **2.3.1 AND-Gate**

The first component of the circuit is called “Control”, which tells if the user wants to read from the memory or wants to write data in it. For it to be accurate, it has to also take into consideration the state of the signals “Full” and “Empty”. This will be done by using AND-gates: if “Push” is active and “Full” is not (the user wants to write in data and the memory is not full) “WR\_Enable” will be active, if “Pop” is active and “Empty” is not (the user wants to read data from the memory and the memory is not empty) “RD\_Enable” will be active.



This block will contain And-gates:

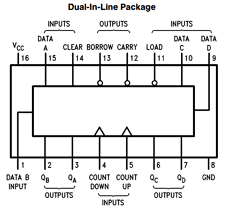
 

### 

### **2.3.2 4-bit Reversible Counter**

This counter is used for identifying the address of the memory. Two of these will be used.

The first one counts every time a it gets a clock and push, ant the second one counts every time it gets a clock and pop.



*Figure 2.3.2 : 74LS193 Synchronous 4-Bit Up Down Counter*

The data inputs will be set to 0000. The Count Up/ Count Down inputs will be connected to 0.

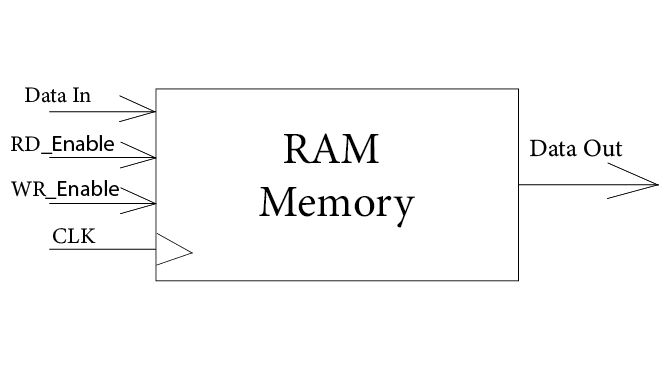
### 

### 

### 

### **2.3.3 RAM Memory**

The Memory component will contain a 16 x 32 RAM memory. This is where the data will be stored. It will have as input “Data In”, “CLK”, “RD\_Enable”, “WR\_Enable”, and its output will be “Data Out”.



*Figure 2.3.3 : RAM Memory*

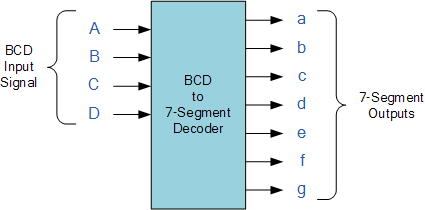
### 

### 

### **2.3.4 BCD-to-SSD**

A 7-segment display (SSD) is a form of electronic display device that consists of seven LEDs arranged in a rectangular fashion. Each LED is called a segment that maps to one of the terminals A through G.

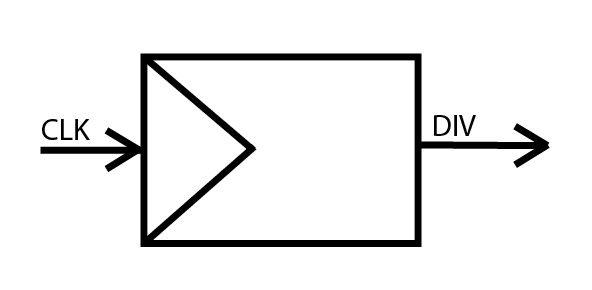
The memory will store the data as a binary number. For decoding the data and putting it on the SSD another circuit will be necessary: a BCD to SSD decoder (Figure 2.3.4.1).



*Figure 2.3.4.1 : BCD to SSD Decoder*

### **2.3.5 Frequency divider**

A frequency divider is needed so it will slow down the clock pulse which is generated. In the case of Nexys4 the clock has a frequency of 100 MHz, so we will make it 40 MHz.

**

*Figure 2.3.5 : Frequency divider black box*

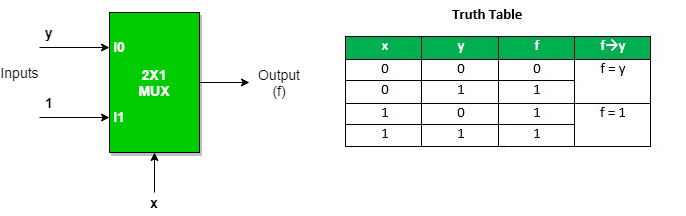
### **2.3.6 Input decoder**

The data will be inserted using the sixteen input buttons from the Nexys4. This will be decoded using a circuit which stores these numbers in a 32-bit number which will be inserted into the memory.

### 

### **2.3.7 2:1 Multiplexer**

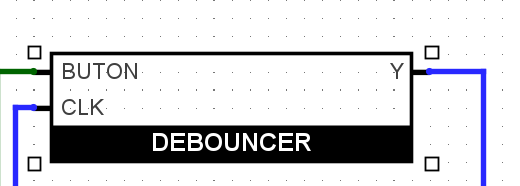
The Multiplexer will tell the memory the address it should be writing on or the address from which the data wants to be read from. It will be a 2:1 multiplexer, with two data inputs and a selection input. It will be on four bits, because the memory has sixteen places (and the numbers from zero to fifteen can only be represented on four bits in binary).



*Figure 2.3.7: 2:1 Multiplexer*

### **2.3.8 Debouncer**

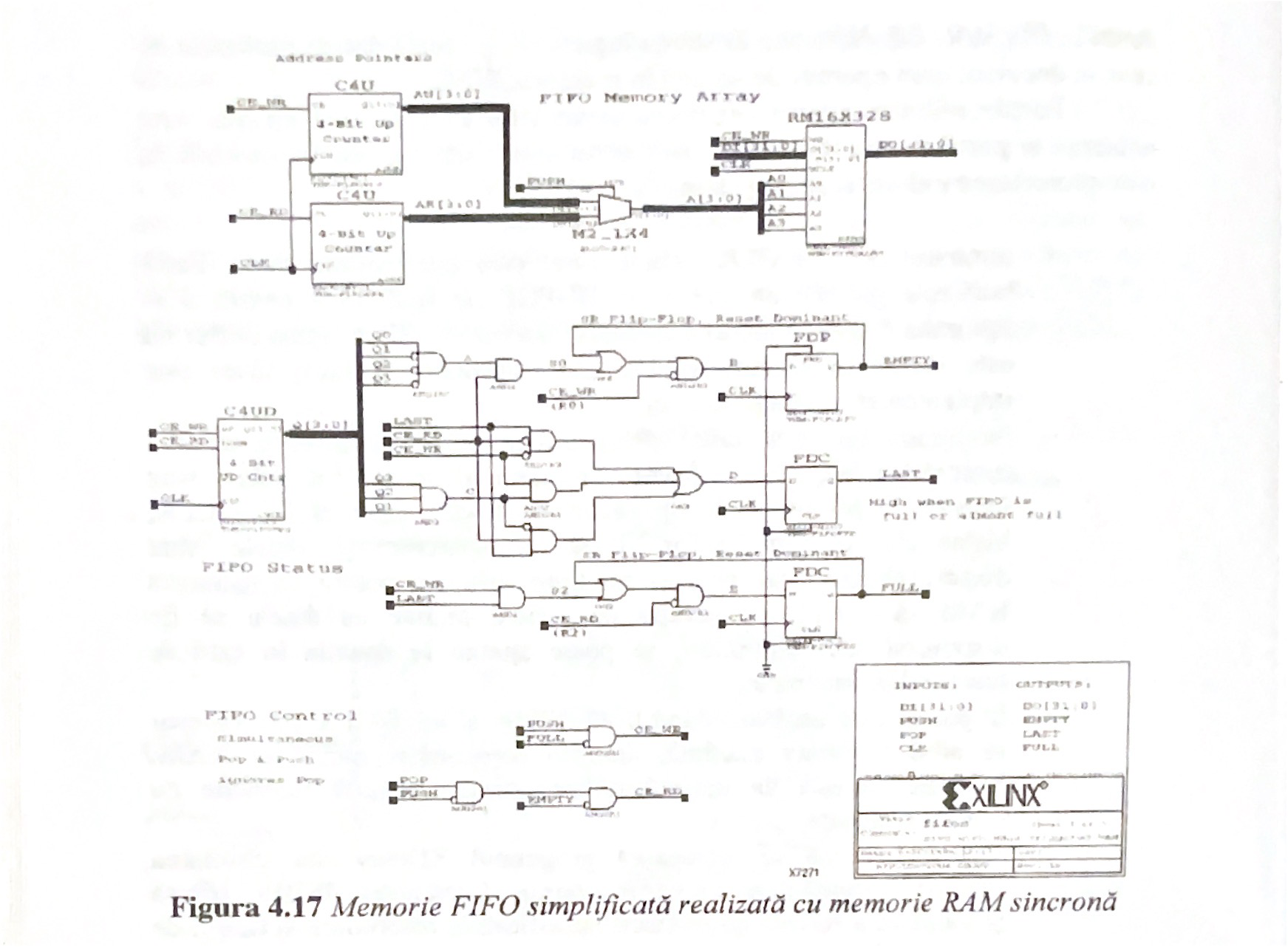
This design is made to be implemented on an FPGA, so for its buttons we will need a debouncer, for example for PUSH, POP. This debouncer will check if the button has been pushed for long enough, in our case for five clock signals and only then it is “turned on”. This is done by cascading five D flip-flops.



*Figure 2.3.8: How the debouncer looks in the implementation*

# **3.Internal Scheme**

The memory will be built after this scheme, which can be found in the book “Designing digital systems using FPGA technology” by S.Nedevschi, Z. Baruch and O. Cret :



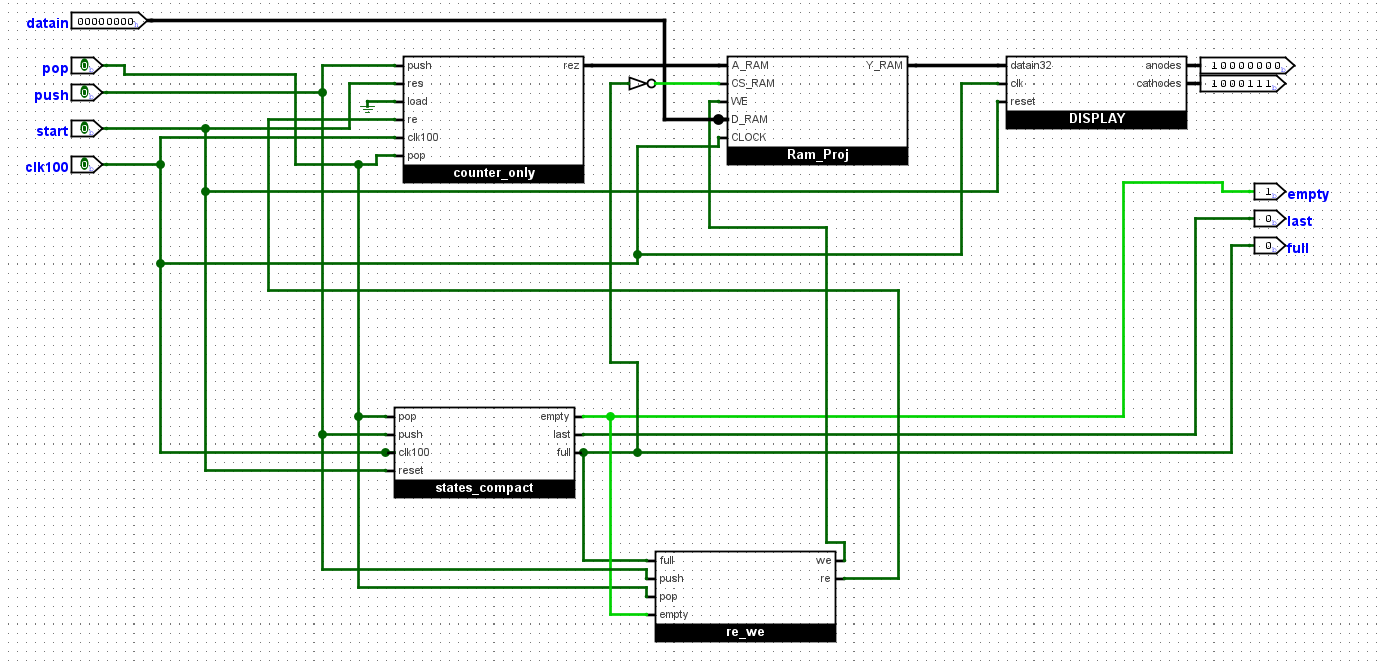
The only change that has been made affects the way the states are defined. The states are defined by the implementation of the state diagram. This has sixteen states, when it is in state 0 it activates the ‘EMPTY’ output, when it is in state 15 it activates the output called ‘LAST’, and in the last state it activates the output called ‘FULL’. Depending on whether ‘PUSH’ or ‘POP’ has been pushed it changes the states and also the outputs. For example, if it is in the last state but ‘POP’ has been pushed it goes back to state 15 and ‘LAST’ will be active again. In means of VHDL code, this is simply done with two case statements.

Also, the Ram memory gets CS\_enable when, according to the state circuit, the ram is not full.

The other parts are built the same way as it is shown on the figure.

Besides this, I also used four D flip-flops, to delay by one clock the address when it is passed on to the memory. It needs to be delayed because the counter starts counting when it first gets a signal and immediately makes the address “0001”, so “0000” will remain empty. If it is delayed before passing it to the multiplexer it uses “0000” for the first input.

The most “compact” way to represent this circuit is shown on the figure below:



From this scheme one component is missing, which transfers the data to the Ram memory.

# 4. User Manual

In practical implementation this circuit has more inputs: 16 buttons for data input, START, New\_Data, PUSH, POP. The output of the memory is shown on 8 SSD-s, and on three leds the user can see the outputs “LAST”, “FULL” and “EMPTY”. Also, it is useful to know that this circuit uses a hexadecimal numbering system for input and output.

There are a few steps to followed when using this memory:

1st step:

Push “START”.

2nd step:

Push “New\_Data”

3rd step:

Start to input the data into the memory. You can do this by turning on and then off the switches. There are sixteen switches and each of them represents a digit from the hexadecimal numbering system. For example, if you want to input the number “12AD”

You need to press for “D” the 14th switch, then for “A” the 10th switch and so on. Because the memory is able to store 32-bit numbers, the maximum number of hexadecimal digits that can be stored at a time is eight. Do not turn on and off more than 8 switches.

4th step:

Push the button “PUSH”.

5th step:

If you want to continue adding data to the memory go back to step 2. If not, go on to step 6.

Important: You can only introduce data into the memory until the light “LAST” is off, when it turns on, there is no more space in it. There is also a “LAST” light that gives you a warning when you're left with only one space.

6th step:

Push “POP” for making the data appear on the display. You will notice that the data appears in the same order that you’ve introduced it.

Important: Pop can be pressed until the light “EMPTY” is off, when it has turned on, it means that there is no data left in the memory.

7th Step:

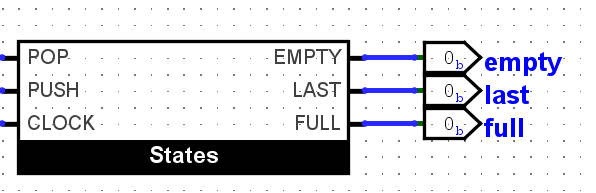
For starting over press “START”.

# 5.Technical justifications for the design

There are a few design choices that I’ve made.

The first one, and the main design choice was the introduction of the data. I decided to use the switches of the FPGA. In my project the component “store\_data” hase sixteen inputs, each of them represents a digit, as it has been explained in the user manual. For new data input the user has to press a button, “New\_data”. This seemed the most user friendly and easily implementable solution for my problem. The main question was how to make the user able to introduce data into the memory. The result of this component is a 32-bit number, which will be passed to the memory.

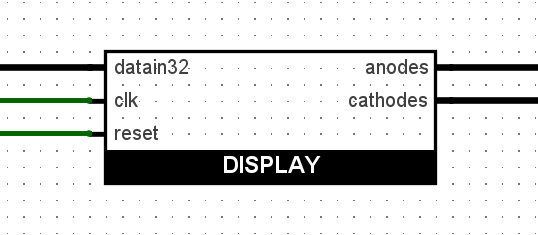
Another design choice that differs from the one that is presented in the book is the implementation of the Control Unit. As I’ve explained it in the chapter “Internal Scheme”, I implemented my state diagram.



*Figure 5.1: The black box of the C.U.*

The data from the memory is seen by the user on the Seven Segment Displays of the FPGA.

Nexys4 has 8 of these, so on this FPGA the whole 32-bit number can be seen. This is done by the “Display” component:



# 6.Future developments

The main future development for this project is the usage of a Keypad for introducing the data. This would make it all much easier to use.

The Keypad, which is technically a keyboard with the digits from 0 to F (15H) , passes to the circuit the code of the button that has been pressed, this needs to be decoded and passed further to the memory.

Another development for the project could be implemented without using the button “New\_Data”. This button makes the implementation easier but has no other usage, it simply resets the data to 0 so the user can introduce a new set of digits. This could be done in another way, such that this component for example, detects if “push” has been pushed and it resets its data to 0.

# 7.References

For my project the main reference was the book “Designing digital systems using FPGA technology” by S.Nedevschi, Z. Baruch and O. Cret. The main idea was from the book and also it helped a lot in the process of the implementation.

I also used the Laboratory Works from the Digital System Design class for correctly implementing the circuit.