Pocket calculator

Ilie Berindei

UniveSITATEA TECHNICĂ  ÎNDRUMĂTOR: TIMAR MIHAI

​​

06.06.2024

STUDENT NAME: Berindei Ilie-Mina

PROJECT SUPERVISOR: Timar Mihai

**Table of Contents**

​

1. ​Specifications……………………………………………………………………………...2
2. Design…………………………………………………………………………………......3
   1. Black Box………………………………………………………………………….3
   2. Control and Execution Unit……………………………………………………….4
      1. Resources………………………………………………………………….5
      2. Detailed Diagram of the project……………………………………………7
3. Technical justifications for the design…………………………………………………….8
4. Future developments…………………………………………………………………….10
5. References……………………………………………………………………………….11

​

​

​

​

​

​

​

​

​

​

​

**Pocket Calculator**

1. **Specifications**

Design a pocket computer with basic arithmetic operations (addition, subtraction, multiplication, division). Multiplication and division operations will be implemented using specific algorithms, not language operators. The operands are represented by 8 bits with a sign. The operands and operators will be entered sequentially in decimal form. The 7-segment displays on the FPGA boards will be used. The project will be carried out by 1 student.

The calculator will operate as follows:

* The initial state is a standby state, where the calculator waits for the person to input the first number.
* When the first digit of the number is inputted, the person controlling the calculator has to decide if there is another digit to input, if so then the Clock button will be pressed. Otherwise, the enable operation input will be switched on and the operation will be chosen and the clock signal will be pressed.
* After the enable operation switch is on, the machine waits for the second number to be introduced. Before pressing the last Clock signal, the person using the calculator must turn on the Equal input, so the calculator knows it is time to do the operation that was chosen.
* For the machine to print the result, the person controlling it must enable the start input, so the machine can do the operation. The result will be printed on the Seven-Segment-Display, digit by digit.

1. **Design**
   1. **Black Box**

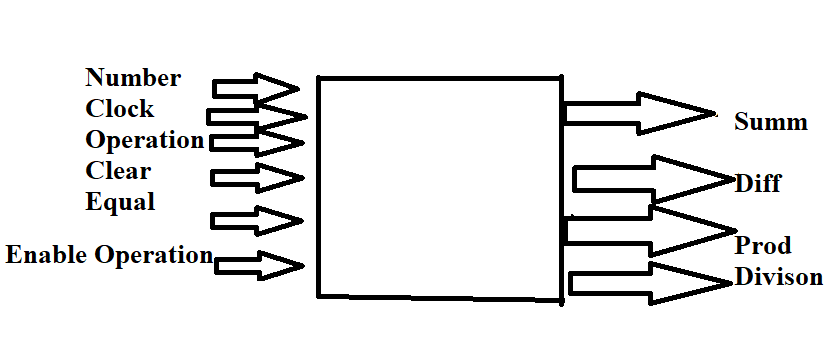


Figure 1. Black Box of the calculator

Using the pocket calculator is straightforward. As described earlier, the user simply inputs the digits of the first number, selects the desired operation, and then inputs the second number. The result is then displayed on the Seven Segment Display (SSD). The calculation process is simple and will be explained below. Numerous signals were integrated into the program to handle the calculation and display of each digit (i.e., each digit of the result must be shown on the SSD). For displaying the output, we control the Cathodes and Anodes of the SSD rather than directly printing the result.

**Input and output signals can be modified or added at any time.**

* 1. **Control and Execution Unit**

To design a pocket calculator, we will first break down the system into its fundamental components: the Control Unit (CU) and the Execution Unit (EU). This approach allows us to distinguish between the control logic and the system resources.

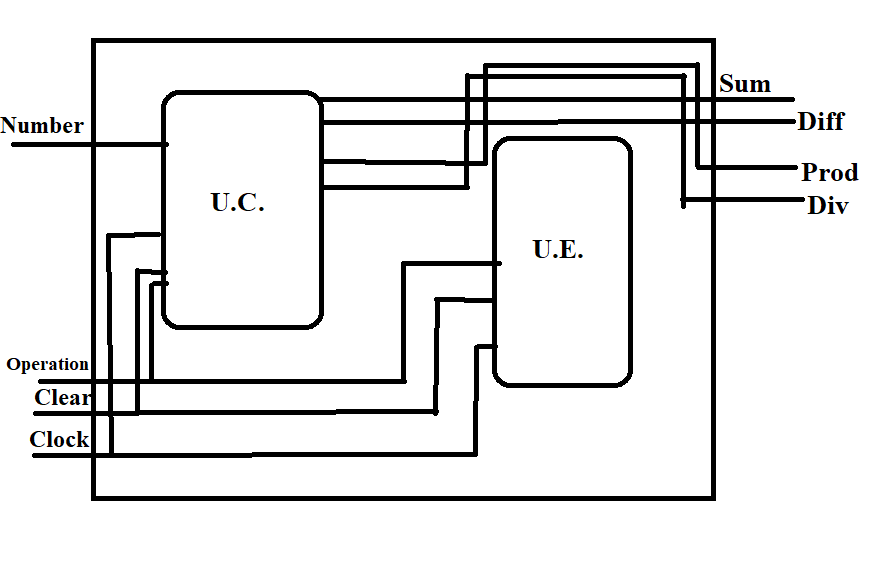


Figure 2 Control and Execution Unit

1. **Control Unit (CU):**

* Input Handling: The CU is responsible for processing user inputs. This includes detecting which keys are pressed (numbers, operators) and in what sequence.
* Operation Control: The CU interprets the input sequence to determine the operations that need to be executed. For example, if the user inputs "5 + 3", the CU recognizes this as an addition operation between the numbers 5 and 3.
* Display Management: The CU updates the display with the current input, and results

1. **Execution Unit (EU):**

* Arithmetic Logic Unit (ALU): The ALU performs the actual calculations. This includes the basic operations like addition, subtraction, multiplication, and division.
* Memory Unit: The EU includes registers or memory cells to store intermediate results, and the current state of the computation (e.g., current total in a running sum).

1. **Resources**

To establish the links between the Control Unit (CU) and the Execution Unit (EU), we must first identify the resources that inform our decision-making. These resources can generate signals for the CU and can be managed by the CU through Enable or Reset signals.

Decision-making information must originate from a resource that generates and passes this information to the CU.

Resources can be simple circuits, such as counters or registers, which can be implemented directly. They can also be complex resources, such as remainder or multiplication algorithms. These complex resources may initially appear as black boxes in the first breakdown, with defined inputs and outputs. However, they must eventually be further broken down (typically into CU and EU components) until they are represented by known circuits.

1. **Counters**

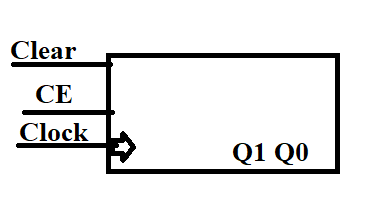


Figure 3 Counter

This is a 2-bit modulo 3-counter. The machine uses 2 of these counters for counting the digits of the first and the second number, and it is used to know which anode to turn-on for the Sevent Segment Display. In total we have 6 states, that is if both the numbers have 3 digits, and the biggest number on 8 bits is 255.

1. **Registers**

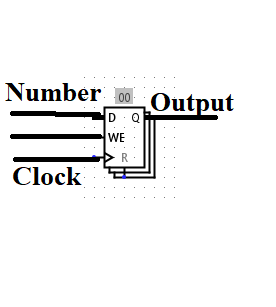
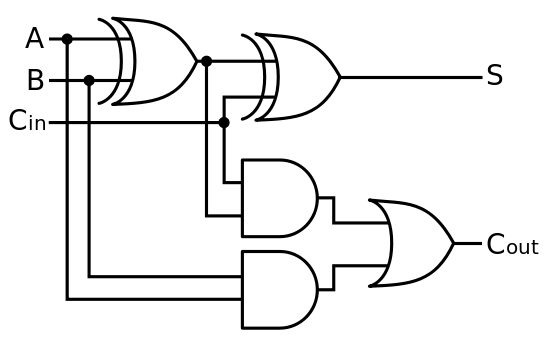


Figure 4 Register

This is a 4-bit register, that contains the digit of each number. After the clock button is pressed, the register keeps the value of the digit, based on the counter. So, if the counter is 1, then the second digit is going into the second register. The program uses a lot of register, even for the full number, the result and maybe for intermediate results, so, the registers have a big factor in this program.

1. **1-bit Full Adder**



*Figure 5 1-bit Full Adder*

The 1-bit Full Adder has a big impact on the program, because he is used on the addition of the two numbers, and also for the subtraction of them. With the help of the for and if generate statements in VHDL, we can build a n-bit adder and subtractor, by only using the 1-bit Adder. Of course, other logical statements have to occur, like the Carry in, for the first addition must be 0.

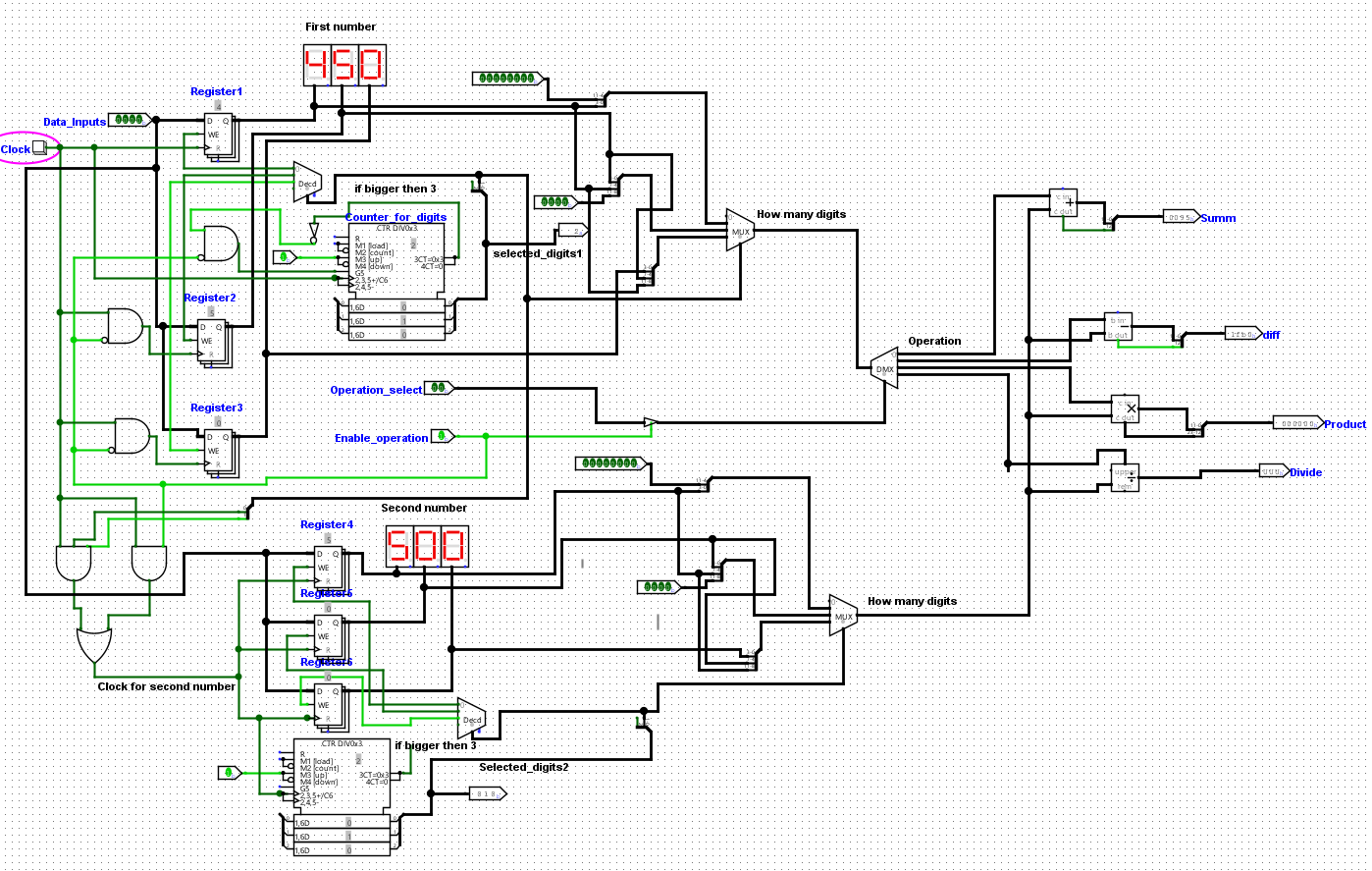
1. **Detailed diagram of the project**

Figure 56 Detailed diagram of the project in Logisim

1. **Technical justifications for the design**
   1. **Modular Design with Reusable Components**

* Components for Arithmetic Operations: The design uses separate VHDL components (‘Sum’, ‘Subtract’, ‘Product’, and ‘Divide’) for each arithmetic operation. This modular approach allows for reusable and maintainable code.
* Ease of Debugging and Testing: Each component can be individually tested and verified, simplifying the debugging process. Any changes or optimizations to a specific operation can be done in isolation without affecting other parts of the system.

Technical Details:

* Component Instantiation: Components are instantiated and mapped within the architecture, enabling structured and clear connections between different parts of the system.
  1. Efficient Use of State Machines:
* State Management: The design efficiently manages states using multiple signals (‘count\_digit1’, ‘count\_digit2’, ‘count\_digit3’) to keep track of the current state of input, operation selection, and result display.
* Clear State Transitions: By defining clear conditions for state transitions, the design ensures robust handling of input sequences and operation execution.

Technical Details:

* State Transitions: The code uses conditional checks and state transitions based on signal values and input conditions (e.g., EnableOperation, Equal, Start, Clear).
  1. Effective Use of Seven Segment Display (SSD) Control
* Dynamic Display Updates: The design dynamically updates the SSD based on the current state and digit values, ensuring that the correct digit is displayed at the correct time.
* Direct Control Signals: The use of Cat1 and An output signals allows for precise control of the SSD segments, enabling clear and accurate display of results.

Technical Details:

* SSD Control Logic: The code contains multiple processes to handle the SSD updates based on the current state and calculated results.
  1. Flexible Input Handling
* Sequential Input Processing: The design handles input digits sequentially for both numbers involved in the operation. This allows for flexible and dynamic input without needing to predefine the number of digits.
* State-Based Input Assignment: Input digits are assigned to registers based on the current state (count\_digit1, count\_digit2), ensuring that each digit is correctly placed.

Technical Details:

* Sequential Register Assignment: Input digits are stored in registers as they are entered, with different processes handling the assignment based on the current state.
  1. Scalable Design for Multiple Operations
* Operation Handling: The design supports multiple operations (addition, subtraction, multiplication, division) by using a combination of components and state machines. This makes the design easily extendable for additional operations in the future.
* Result Handling: The results of different operations are processed and displayed correctly, ensuring that each type of calculation is handled appropriately.

Technical Details:

* Operation Selection and Result Processing: Based on the operation selected, the appropriate arithmetic component is activated, and the result is processed and formatted for display.

1. **Future Developments**
   1. Fractional Arithmetic Support

* Many real-world calculations involve fractions or decimals. Adding support for fractional arithmetic would significantly enhance the calculator's utility.
  1. Memory Functionality
* Memory functions like memory recall, memory store, and memory clear are standard features in calculators, enhancing user convenience and productivity.
  1. Scientific Functions
* Scientific calculators offer advanced mathematical functions such as trigonometric, logarithmic, and exponential functions, which are valuable for scientific and engineering applications.
  1. Error Handling and Feedback
* Error handling mechanisms improve user confidence and help prevent incorrect calculations.

1. **References**
2. <https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual>
3. <https://www.allaboutcircuits.com/technical-articles/basic-binary-division-the-algorithm-and-the-vhdl-code/>
4. <https://en.wikipedia.org/wiki/VHDL>
5. <https://didatec.sharepoint.com/:w:/r/sites/LogicDesign2023-2024/_layouts/15/Doc.aspx?sourcedoc=%7B4DEC65B9-6D55-4887-AEB9-CC79A57C3A5F%7D&file=Cooking%20Oven%20Example%20project.docx&action=default&mobileredirect=true>
6. <https://www.allaboutcircuits.com/projects/from-vhdl-code-to-real-hardware-designing-8-bit-ALU/>
7. <https://medium.com/@karlrombauts/building-an-8-bit-computer-in-logisim-part-2-arithmetic-ae7861c82e79>