Rolling Average

~Digital System Design project

Bartha Tudor & Duțu Ana Sofia
DRD. ENG. DIANA IRENA POP

Table of Contents

1.Specifications:

The Design Assignment will be to develop a simple signal processing system that will calculate the rolling average of a parallel 8-bit data stream as a systems design exercise.

The task is to develop a VHDL based model for the "Digital Filter / Rolling Average" system combined with a data stream generator. Switches, Buttons and the Seven Segment Display located on the Diligent S3 board will need to be included to demonstrate correct operation.

The system must operate as follows:

The C.U. will generate the data for the E.U. based on the control settings:

1. off-off-off(000)->test mode, will generate 00000000.
2. off-off-on(001)->square wave, will generate 00000000 and 0.25XDataclock(0.25 hz).
3. off-on-off(010)->repeated 6 digit sequence for student nr. 1
4. off-on-on(011)->repeated 6 digit sequence for student nr. 2
5. on-on-off(110)->pseudo random sequence reduced range 0-15.
6. On-on-on(111)->pseudo random sequence full range 0 to 255.

The E.U. will generate the average of the numbers based on the buffer “length” settings:

1. .off-off-off(000)->stop, hold value.
2. on-off-off(100)->2 sample average.
3. on-off-on(101)->4 sample average.
4. On-on-off(110)->8 sample average.
5. On-on-on(111)->16 sample average.

2.Design

2.1.Black box

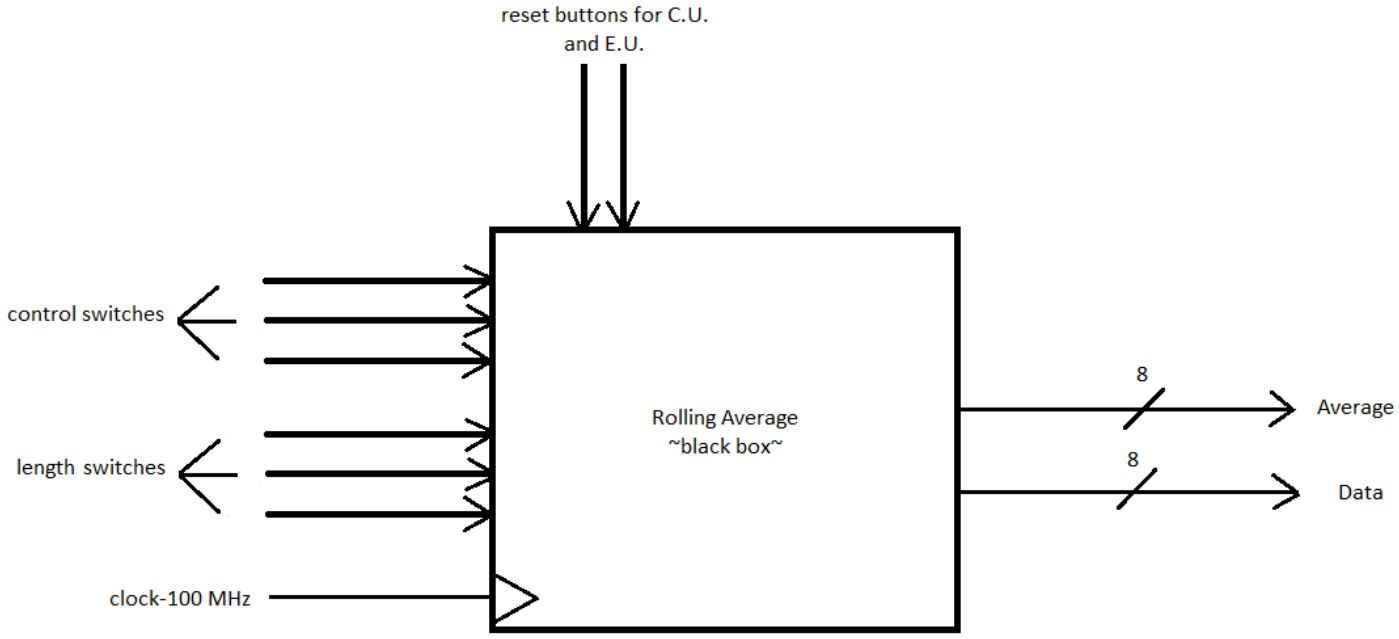


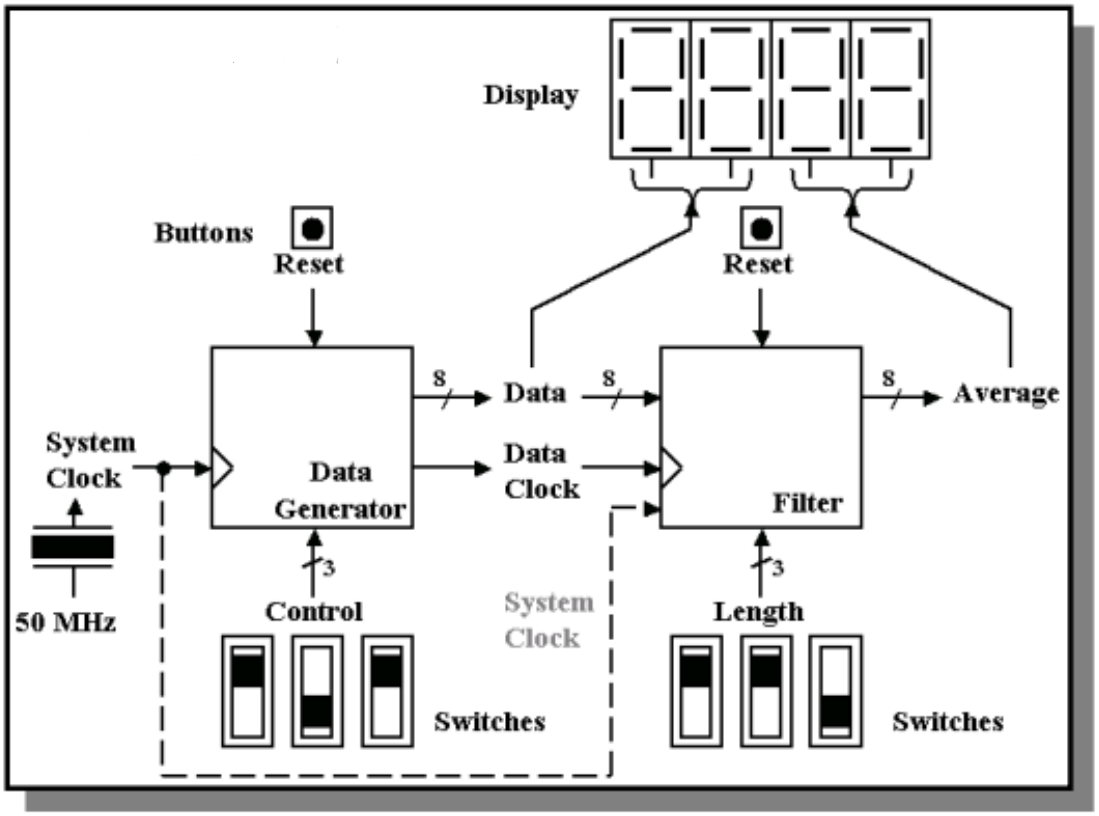
Figure 1 Black Box of the system

2.2 Control and Execution Unit

This black box must be broken down in order to understand the functionality of the whole system, since there usually are inputs and outputs that are not specified in the black box.

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

The first breakdown of any system is one in which we will differentiate between the **control logic** in the system and the **system resources**. The control logic is represented by the Control Unit (CU) and the resources are represented by the Execution Unit (EU).



2.2.2. Resources (breakdown of the Execution Unit)

The Execution Unit (E.U.) performs the operations and calculations forwarded from the instruction unit. It may have its own internal control sequence unit (not to be confused with the control unit), some registers, and other internal units such as an arithmetic logic unit, address generation unit, floating-point unit, load–store unit, branch execution unit or some smaller and more specific components.

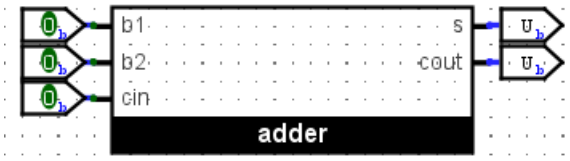
RESOURCES:

1.) 8 bit register:

- This component stores a single multi-bit value. The reset input resets the register's value to 0 (all zeroes) synchronously.

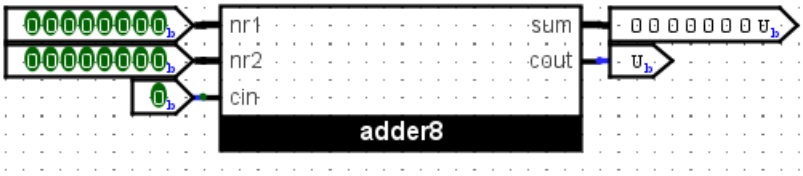
2.) 1 bit adder:

- This component adds two values coming in via the west inputs and outputs the sum on the east output. The component is designed so that it can be cascaded with other adders to provide add more bits than is possible with a single adder: The carry-in input provides a one-bit value to be added into the sum also (if it is specified), and a carry-out output provides a one-bit overflow value that can be fed to another adder.



3.) 8 bit adder:

- This component uses structural architecture using 8 1-bit adders with n signals used for cascading the 1 bit adders.

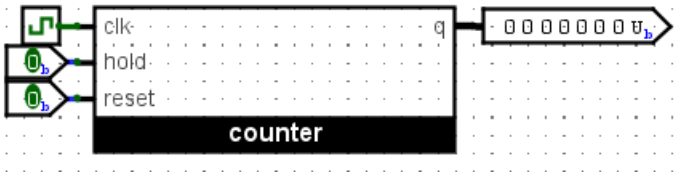


4.) 4:1 mux:

- This component is a 4-1 multiplexer used for choosing the output, average of 2,of 4,of 8 or of 16. The decision is based on the input s (buffer length settings), but only the last two bits because s2 is always 1,unless it's the hold value state.

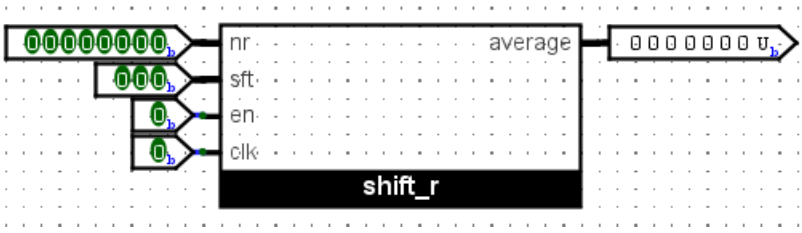
5.) Counter:

- This component is used for determining how many numbers have entered the registers. The output on the average ssd will be 0 as long as the number of numbers required by the length input is not met. Ok was used as a variable, alongside temp, so that the first cycle until reset will be the same as after reset. Without ok, the first cycle work, but after 1 reset the counter will be 1 after the first clock, not after the first 2 clocks. The input hold is generated by another component(length determinate).



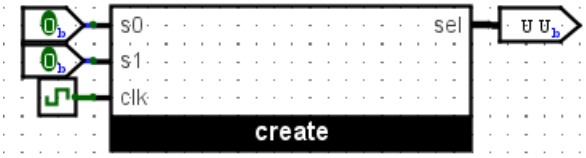
6.) Shifter:

- This component is the shift register which divides the sum of numbers by moving all the data bits to the right with one position on each clock pulse and placing 0 on the bits that are left empty in the left.



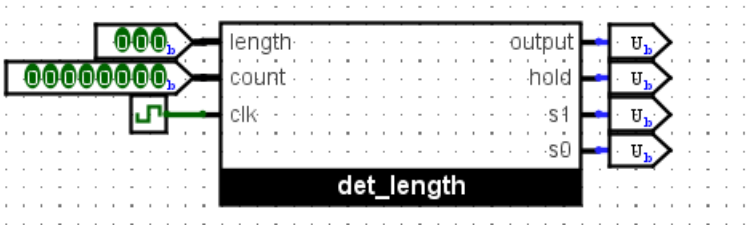
7.) Create:

- This component is used to create the select signal for the multiplexer from 2 signals on 1 bit to create one signal on 2 bits.



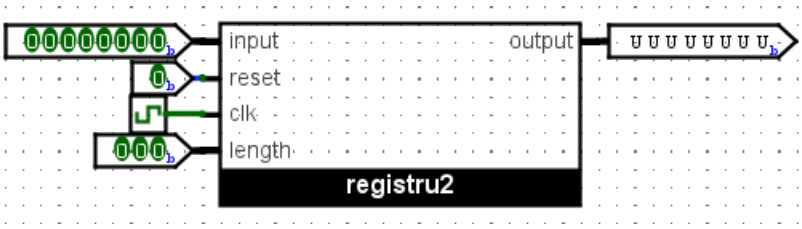
8.) Length determinate:

- This component works in a system with the other components such that it receives the length of the sample average, and it receives the input from the counter to see how many numbers are currently in the registers. If the number of samples is met, the component activates the shift register through the output and sends the s0 and s1 into the create component, otherwise it holds.



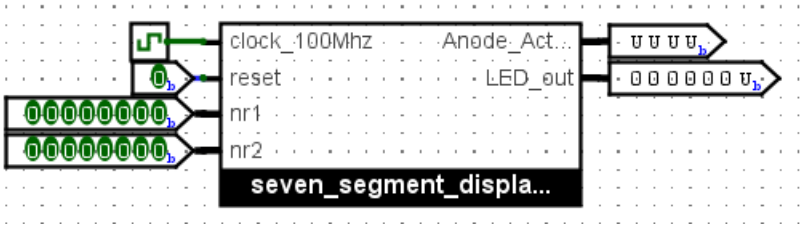
9.) Register for the seven-segment-display:

- This component is a register used to store the average. This register is needed so that when the input on the length is an unspecified bit combination it holds the value until a valid input is given.

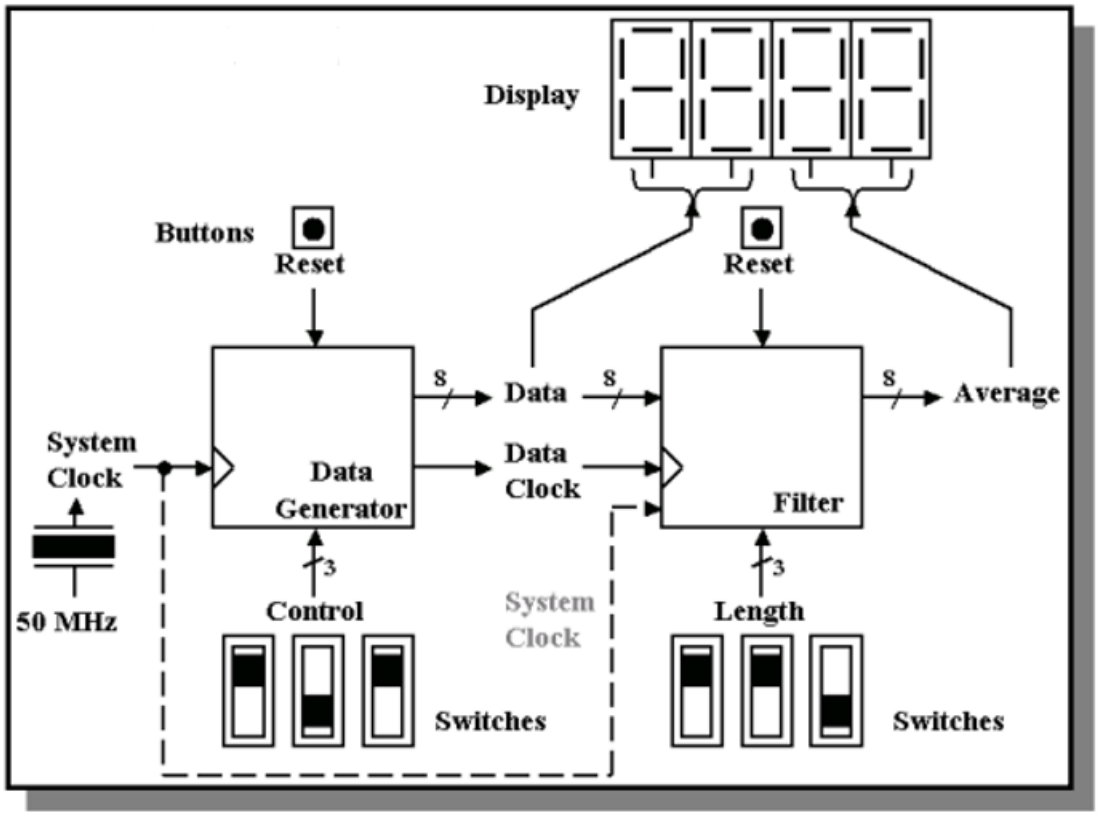


10.) Seven Segment Display:

- This component converts the output data (final average) and the current data into hexadecimal and it displays it on the SSD, correctly using the attributes of the anodes and the cathodes. The component has a 10.5ms refresh period.



2.2.3. Block Diagram for first breakdown



2.2.4. Resources (breakdown of the Control Unit)

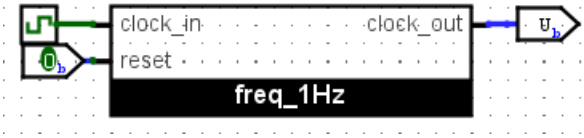
The Control Unit (C.U.) provides all the data for the Execution Unit that will be processed and used. In order to establish the links between the C.U. and E.U., we have to identify and understand all of the resources used to make decisions. These resources can generate signals to

the control unit and can be controlled by the C.U. via Enable or Reset signals. The resources cand be simple circuits(which can be implemented directly), or complex resources(which have to be broken down using black boxes and be built on their own).

RESOURCES:

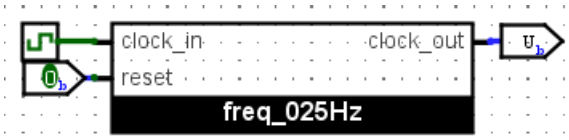
1.) Frequency divider:

- This component takes the input clock with the frequency of 100 MHz and divides it, the output clock having a frequency of 1 Hz.



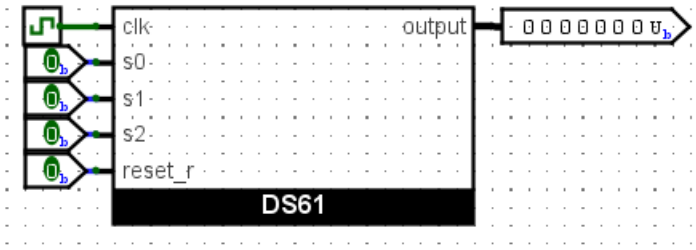
2.) Frequency divider for square wave:

- This component will divide the input clock with 1 Hz frequency and output the 0.25 Hz clock, as required in the control settings by the off-off-on(001) case.



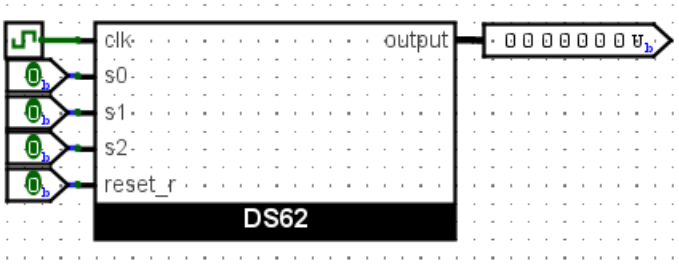
3.) Repeated 6 Digit Sequence for Student nr. 1:

- This component creates a block of memory for the first student, when the control settings are 010, where it keeps a 6 digit sequence of 8 bit numbers chosen by the student in registers and provides them to the execution unit in a repeated order, such that after the last number, the first one will be the output. The component has parallel load such that when the C.U. starts, predefined values enter the registers, and after that they shift.



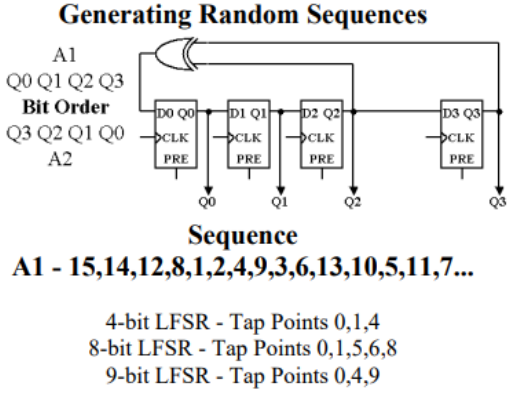
4.) Repeated 6 Digit Sequence for Student nr. 2:

- This component works exactly the same as component nr 3, the only difference being the control settings. For this component to enable the parallel load the switches must be 011.

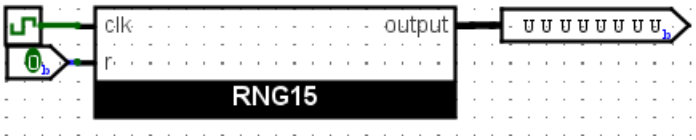


5.) Pseudo Random Sequence reduced range 0 to 15:

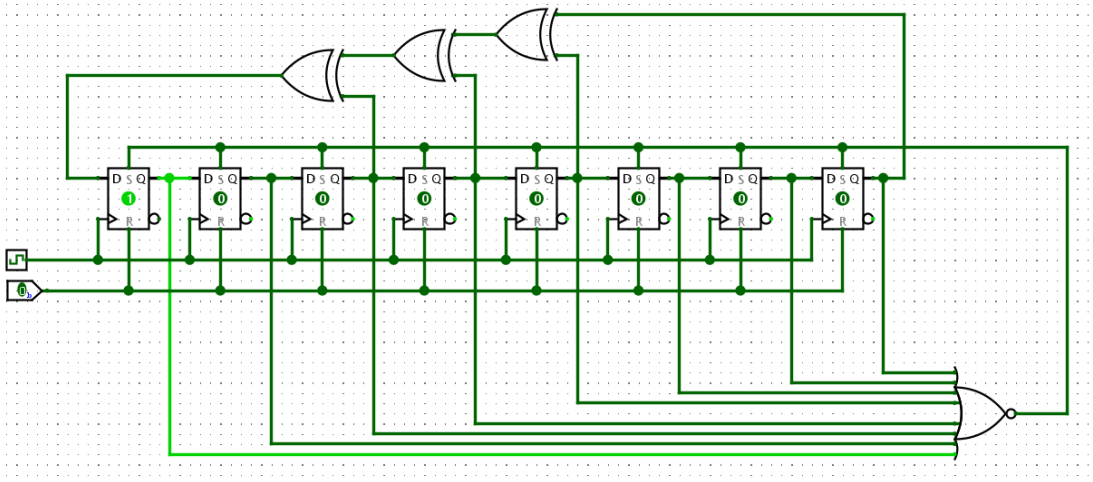
- This component generates random numbers on 4 bits in a range from 0 to 15, using D Flip-Flops.

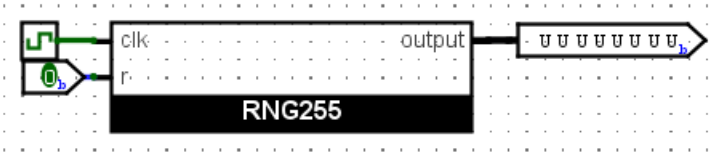


The diagram on the left shows a linear feedback shift register (LFSR). The exclusive-OR gates and shift register act to produce a pseudo-random binary sequence (PRBS) at each of the flip-flop outputs. By correctly choosing the "Tap points" at which we take the feedback from an n -bit shift register, we can produce a PRBS of length 2^n – 1, a maximal-length sequence that includes all possible patterns (or vectors) of n bits, excluding the all-zeros pattern. The diagram shows a 4-bit linear feedback shift register (LFSR) that produces a repeating string of 15 pseudo-random binary numbers. Shown is one of four XOR Flip-flop configurations that exist for PBRS generators. For a greater range of values (eg 255) then the number of bits in the LFSRs may be increased with specific tap points to generate the required maximum length pseudo-random binary sequences.



6.) Pseudo Random Sequence full range 0 to 255:



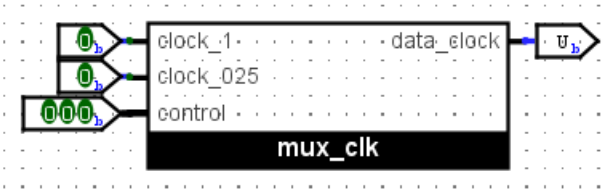


7.) Mux for Data:

- This component takes in all of the outputs from the number generators and transmits only one 8 bit number(or 4 bit number in the case of the pseudo random sequence reduced range 0 to 15) to the E.U. based on the control switches. If the switch case does not trigger any generator, the mux will transmit 00000000.

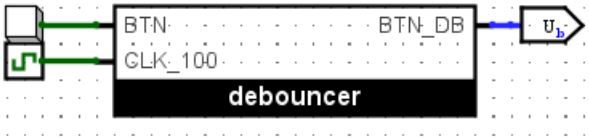
8.) Mux for Data Clock:

- This component transmits the correct data clock to the E.U.: the 1Hz frequency or the square wave frequency of 0.25Hz, based on the control(s0s1s2).

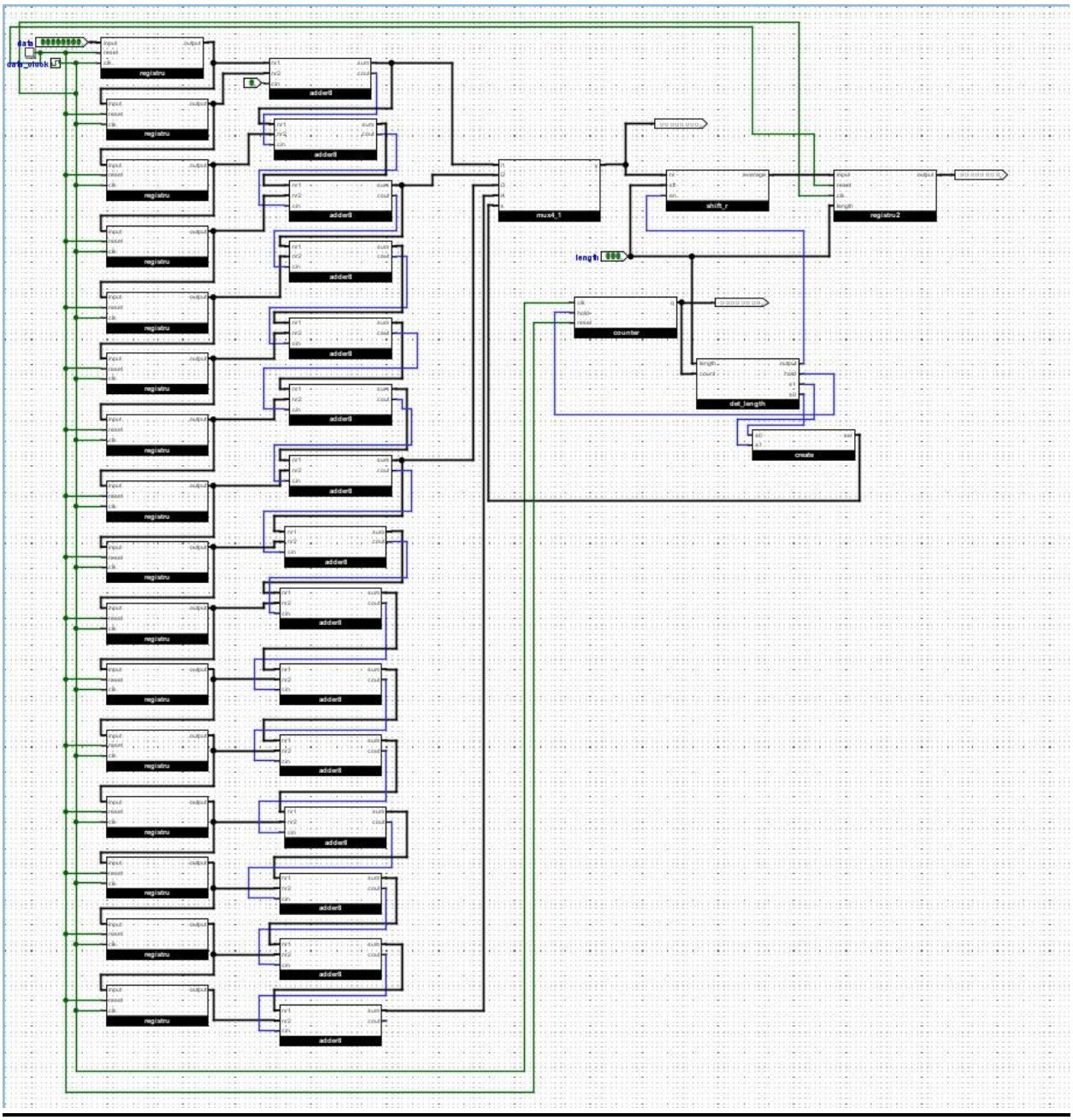


9.) Debouncer:

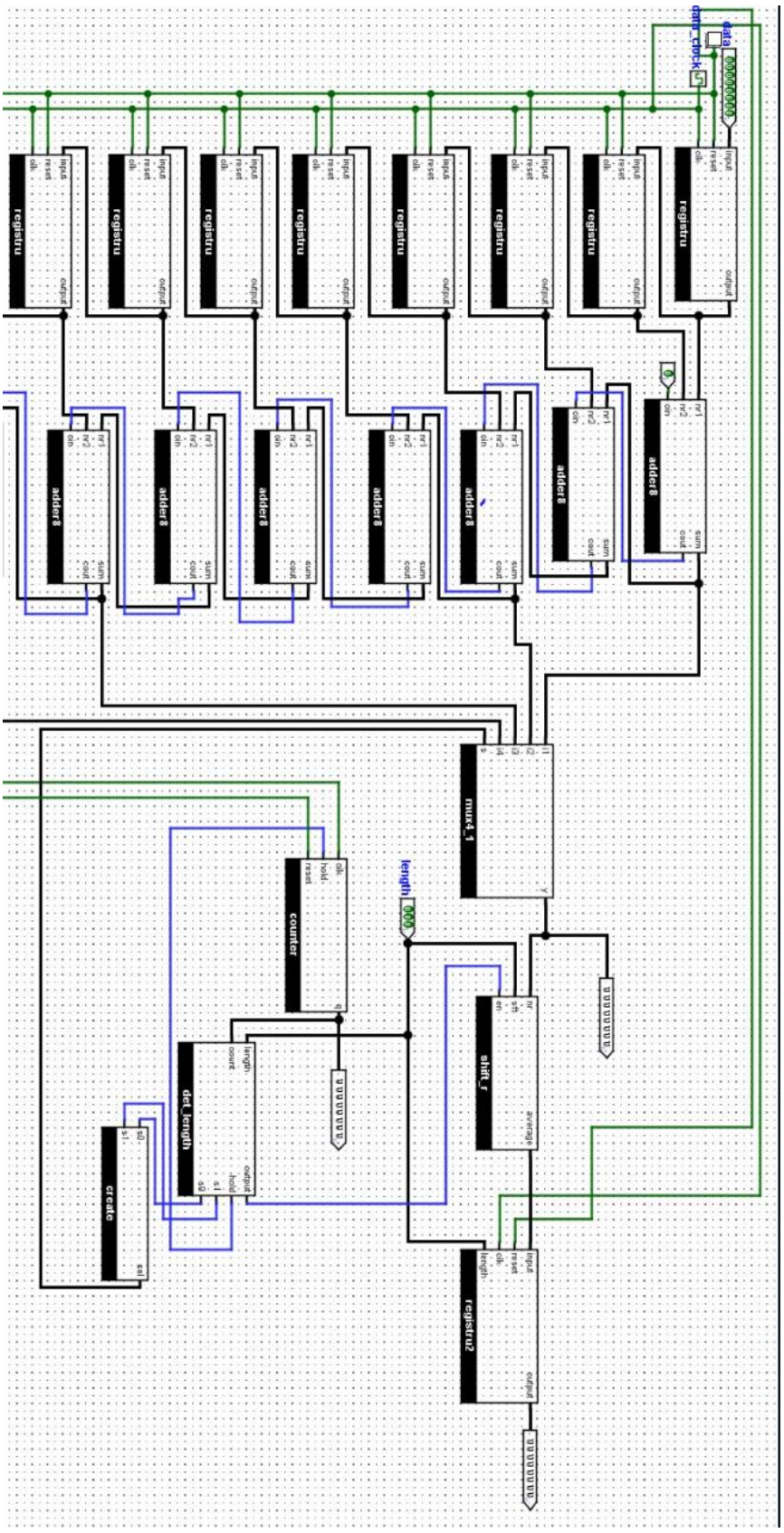
- This component is connected to all the inputs of type “button”. Its functionality is based on cascaded Delay Flip-Flops – this resource extends the signal received from a button, deciding if we take it into consideration (it is “long” enough) or not.



2.2.5. Detailed diagram of the project

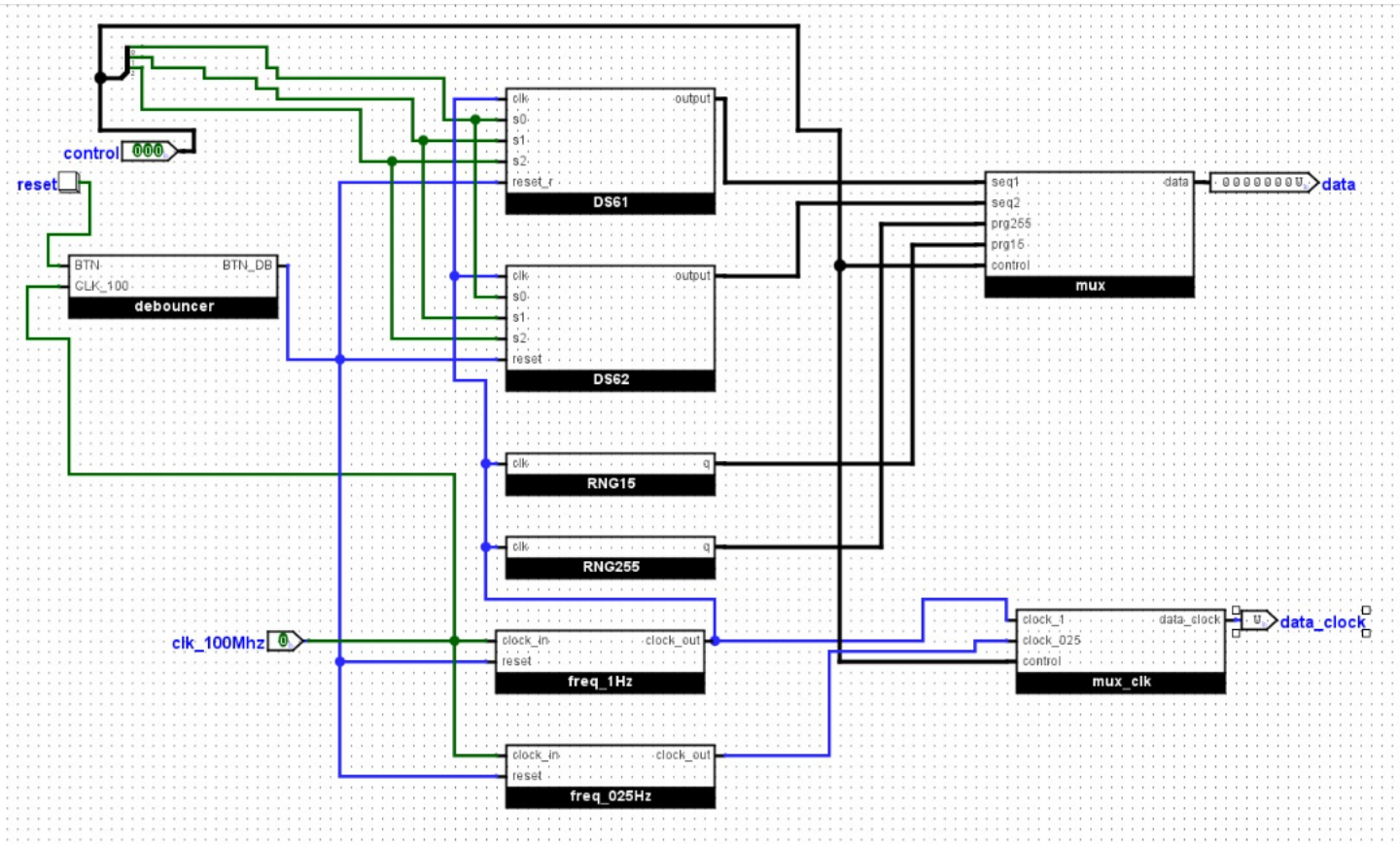


- This is the execution unit, before the final stage of structural encoding.



- A close-up of the execution unit.

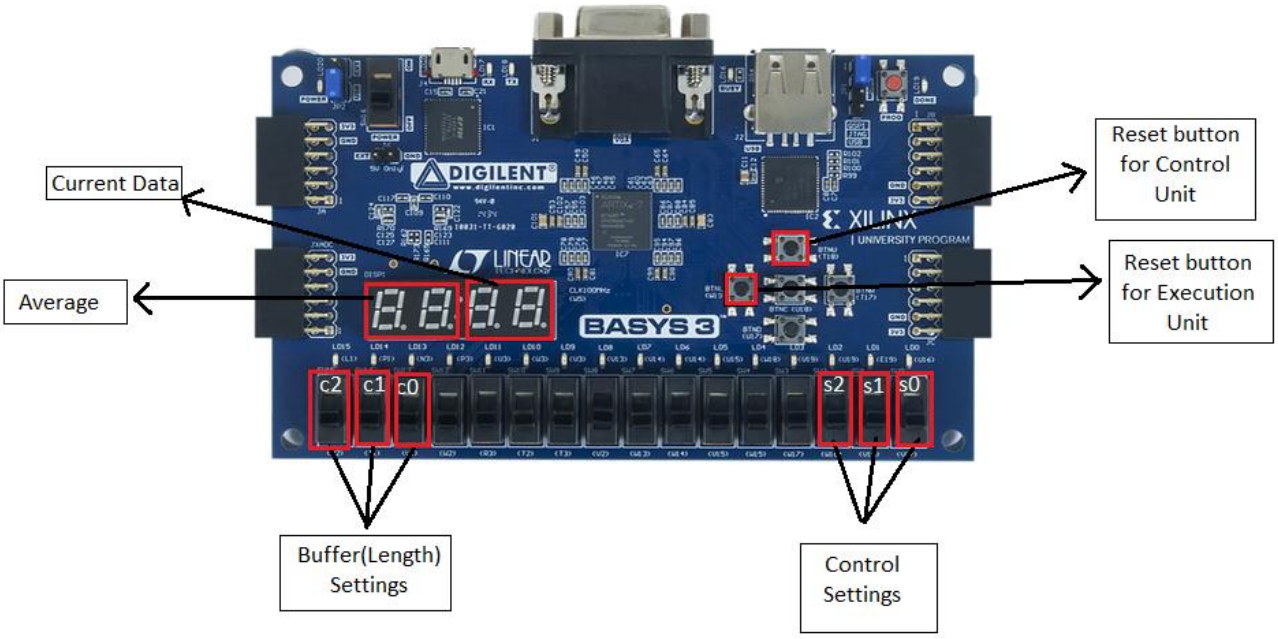
- This is the Control Unit before the final stage of structural encoding:



3.User Manual

- The design will be implemented on a self contained Xilinx/Digilent Spartan 3 XC3S200 FPGA board to allow demonstration of a working system
- Minimal board requirements :
  - 2 buttons
  - 6 switches
  - 4 Seven Segment Display

Preparing the board for testing:



Control Settings:

Off - Off - Off	Test Mode o/p 0 (Zero)
Off - Off - On	Square wave (0.25 x data clock)
Off - On - Off	Repeated 6 digit Sequence for Student Number One
Off - On - On	Repeated 6 digit Sequence for Student Number Two
On - On - Off	Pseudo Random Sequence reduced range 0 to 15
On - On - On	Pseudo Random Sequence full range 0 to 255

Buffer "Length" Settings:

Off - Off - Off	Stop - Hold Value
On - Off - Off	2 Sample Average
On - Off - On	4 Sample Average
On - On - Off	8 Sample Average
On - On - On	16 Sample Average

4. Technical justifications for the design

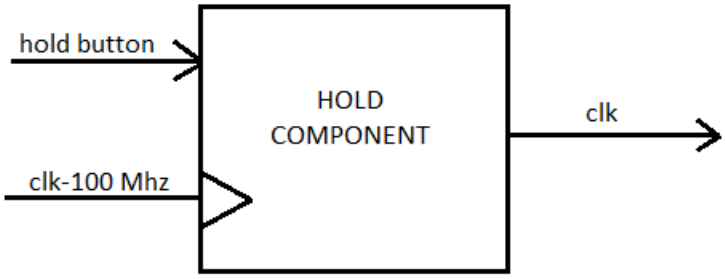
- The project is based on structural implementation, using the bottom-up technique, we divided all of the components needed and kept breaking down until we got to known circuits. The components were tested individually on the board with each step we took, and at the end we just glued all of the components together.

- The Execution Unit works on 15 adders and 16 registers, which calculate all of the possible averages as we go and only displays the required one, giving the illusion that only the average needed is calculated. All of the components work on the 100 MHz frequency data clock, except for the registers who work on the 1 Hz frequency data clock. When the case selected has no action related to it in the user manual, the Seven Segment Display will enter a hold state, the average and the data not being modified until a correct length setting occurs.

- The Control Unit has all of the generating components running the whole time and based on the control settings it generates the numbers as they come from the required component. The reset is tied to all of the components after going through the debouncer, bringing all of the components to their initial state. When the case selected has no action related to it in the user manual, the data will generate 8 bit numers of 0, and the data clock will generate the 1 Hz frequency.

5.Future Developments

1.) The first idea for a future development is a **hold button**, which will send the whole system into a hold state and not letting anything happen on either component (Control Unit & Execution Unit). A new component will be needed which will require a new input button on the board and the clock of frequency 100 MHz. If the button is pressed once, the clock will no longer be transmitted, and if pressed a second time, only then the clock would be transmitted.



2.) The second idea for a future development is to **increase the number** of numbers with which we can calculate the average to 32 numbers, or even 64. The requirements for this implementation would be to double the number of registers and adders. For a 32 sample average there would be needed 32 registers and 31 adders. For a 64 sample average there would be needed 64 registers and 63 adders. The changes to the Execution Unit would be massive, but nothing would change in the Control Unit.

3.) The third idea for a future development is to generate bigger numbers on more bits (eg.16,32…).

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. Stack Overflow - Where Developers Learn, Share, & Build Careers
R5. Intel® FPGA Design Examples | Intel
R6. https://github.com/Digilent/Basys3/blob/master/Projects/XADC_Demo/src/constraints/Basys3_Master.xdc
R7. https://www.researchgate.net/publication/257887573_Hardware_implementation_of_random_number_generators