Random Walkers

EE Lab 230

9 April 2018

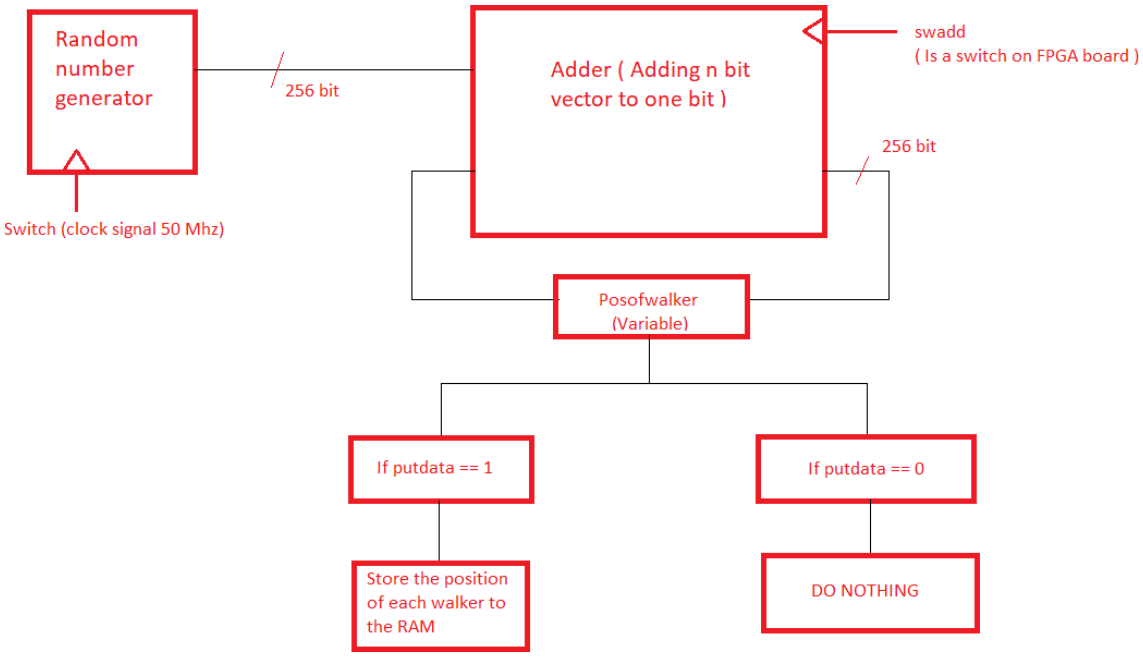
Group Members

1. Kartik Patekar : 160260018
2. Saipriya Satyajit : 160260023

Project Summary

We have simulated an one dimensional random walker with the aim to verify the results of the theory of random walks. We have modelled the problem to a simple adder problem, that is the walker is modelled as a n bit vector and and steps are modelled as random single bits that is zero and one which gets added to the n bit vector. For this we have used a pseudo random number generator using shift registers with a little modification. We have designed an adder which adds the random bit generated to a n bit vector which is the walker. But the point where FPGA's enter the picture is, we instead of having a single walker have used n walkers all of which take steps together randomly, which uses the parallel processing of FPGA. So instead of using a single walker many times to get the statistics we are using many walkers a single time, so this process is considerably faster than the previous one mentioned.

Design block Diagram



Main problems faced and how to overcome

0.1 Random Numbers

- **Problem:** Since we are working with electrical circuits, it is impossible to produce truly random numbers only by using the FPGA. Only seemingly

random numbers can be generated by FPGA. Also, we needed a random bit for each walker(number : n) in our design, and there should be zero correlation between any two random bits.

- **Solution:** To achieve this task, we used a shift register of n bits. The shift register uses feedback mechanism to feed a seemingly random number to the first bit at every clock cycle. This had potential to generate $2^n - 1$ random numbers. However, the random bits between two successive random numbers had high correlation as the bits are shifted. To solve this issue, we used the power of human intervention. The circuit samples the random number on pressing the switch "swadd". Since the time duration between pressing two successive switches has no correlation with number of clock cycles, the numbers we obtained were perfectly random.

0.2 Downloading Data on PC

- **Problem:** The FPGA could store position of all the walkers inside a variable. We could display the position of a single random walker using LEDs, but as we increased n, this way became impossible. Also, we wanted to have a graphical representation of the position of walkers. This made it necessary to transfer the data from FPGA to a PC.

- **Solution:** For this purpose, we used the onboard 1 PORT RAM available on FPGA. The ram can be configured to allow reading and writing the data using the "Insys memory content editor" in Quartus Prime through JTAG interface in real time. On pressing the switch "putdata", the FPGA writes the position on the RAM, which can be viewed on PC.

0.3 Writing Data on RAM

- **Problem:** To write the data on the RAM, we used the variables "ramaddr" and "ramdata" to specify the address and value to be stored. However, since the values in ramdata gets written on ram at positive edge of clock cycle, we were apprehensive about changing their value at the clock edge, and wanted to ensure that values do not change twice in a single clock cycle.

- **Solution:** To solve this issue, we created another clock, using the negative edge of the onboard clock. This clock had frequency 25MHz, and had edges at negative edges of the system clock. We modified the variables "ramaddr" and "ramdata" using this slower clock, to ensure that their values are constant at positive edges of system clock.

0.4 Number of Walkers

- **Problem:** We started with one walker, on it's success we moved on to 2 then to 16 and have finally implemented 256 walkers, we had aimed to

increase the number of walkers furthermore to get better approximation to the theoretical statistics, but when we tried for 1000 walkers only 256 values were stored in the RAM and rest all were zero.

- **Solution:** We haven't yet resolved this issue.

Special hardware parts used

Computer : Used to obtain the statistics of the data output.

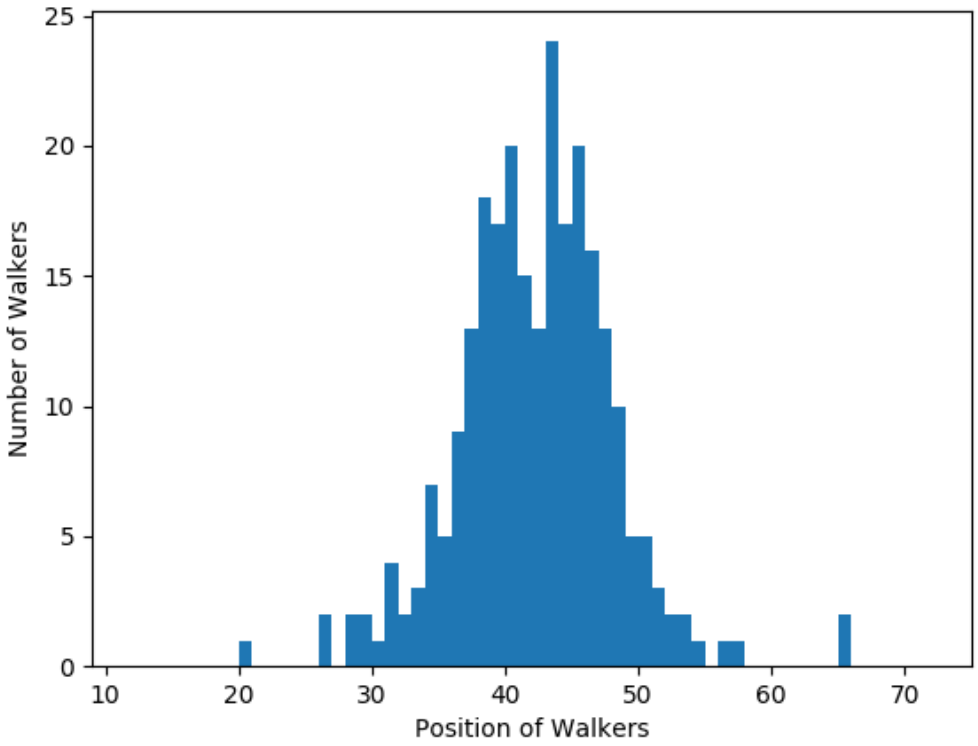
Use of FPGA in implementing this project

The feature of FPGA which is highlighted in our project is its capability to do parallel processing. We have implemented 256 walkers which are taking steps parallely which makes it considerably faster than microcontrollers which would have sequentially done the job that is executing the walk one after the other.

Results

Theoretically, assuming that the walkers take m random steps, then we expect that the average position of walker to be $\frac{m}{2}$ and standard deviation in the position to be $\frac{\sqrt{m}}{2}$.

The following plot was obtained for 256 walkers on 75 random steps.



Parameter	Expected	Observed
Mean	37.5	41.76
Standard Deviation	4.33	5.85