Dijkstra Algorithm Implementation using FPGA

Group 12

Sandesh Gharge (00821875)
Yogesh Nagor (00815160)

Aim

The goal of our project is to implement Dijkstra Algorithm on FPGA. Post implementation we will see how FPGA performs in comparison with CPU.

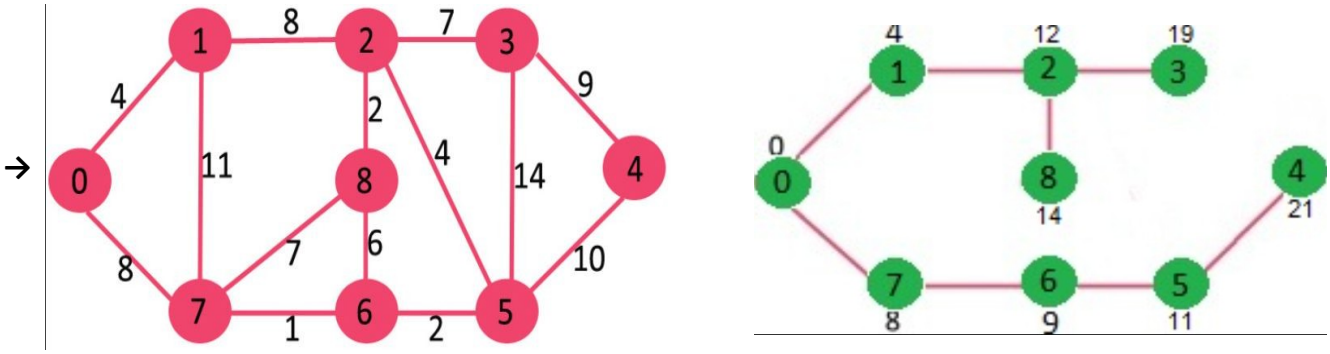
Overview

Dijkstra Algorithm is a pre-defined algorithm conceived by computer scientist Edsger W. Dijkstra. This algorithm helps finding the shortest distance between two points in a given graph.

FPGA (Field Programmable Gate Arrays) are semiconductor devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. To give a brief explanation, FPGA is known for parallel computing unlike normal CPUs.

We are curious to know if Dijkstra Algorithm works better on FPGA as compared to CPU. While running this project we will comparing the time taken by FPGA and CPU on Jupyter Notebook for further conclusions.

Below is the sample example of how Dijkstra Algorithm calculates shortest for a given input -



Tasks:

While implementing the algorithm, following major tasks were supposed to be completed -

1. Creating IP using Vitis HLS
2. Creating Overlay using Vivado

3. Designing Jupyter Notebook
4. Comparing CPU and FPGA results.

We will be looking into each solution in detail.

Solutions:

1. Creating IP on Vitis HLS

   C++ language is used to create an IP using Vitis HLS. Following is the C++ code* implemented as required (only outline is shown) -

```
void Dijkstra(int* a, int* res)
{
#pragma HLS INTERFACE m_axi port = a  offset = slave bundle = gemm1
#pragma HLS INTERFACE m_axi port = res  offset = slave bundle = gemm2
#pragma HLS INTERFACE s_axilite port = return

    // a → Input ( [8 x 8] 2d Array)
    // res → Output ( [8] 1d Array)
    .
    .
    //Algorithm
    .
    .
}
```

   full code is attached in the git link

   m_axi port was used for both input and output in this case since the format type was array.

2. Creating Overlay using Vivado

Below block diagram is created by using “Run Auto Connection” in Vivado -

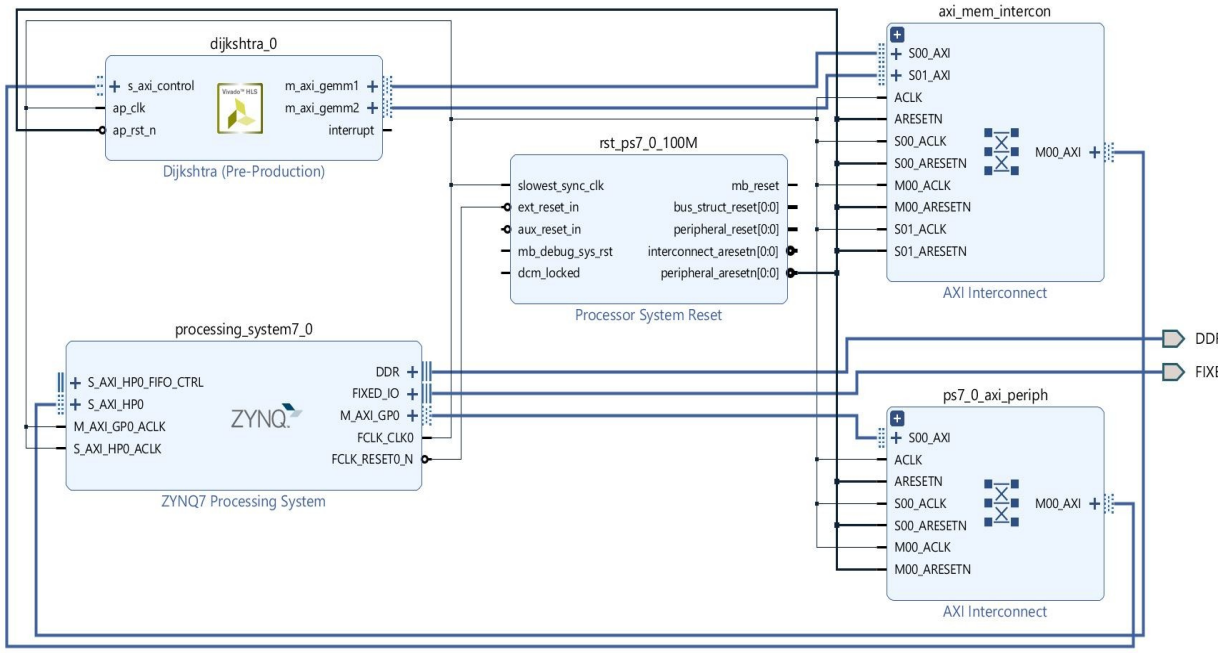


FIGURE 1.1. Block Diagram of Overlay

3. Designing Jupyter Notebook

Jupyter Notebook was designed using python code whose link is as follows - https://mygit.th-deg.de/group12-embeddedsecurity/Dijkstraalgorithmfpga

Python forms a great platform for presenting results in required graphical formats. We have attempted to show results in the Bar Graph format

4. Comparing CPU and FPGA results

As checked, FPGA perform better than CPU most of the time. Also note that same input is processed multiple times to get results, i.e. we are using a for loop for running the algorithm on both FPGA as well as CPU. Below is the graph for few results -

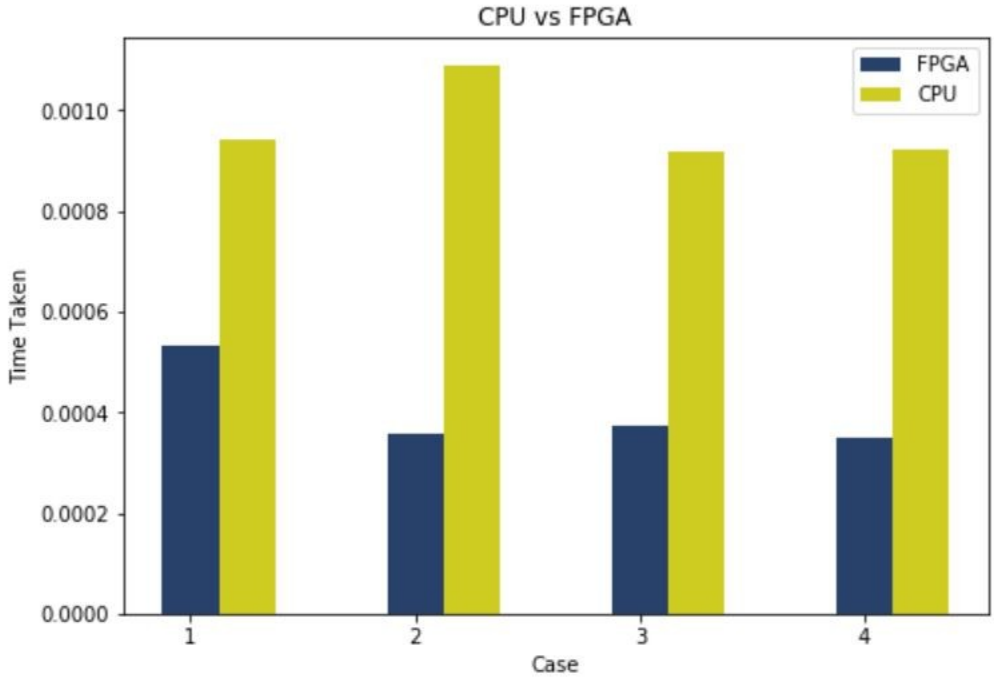


FIGURE 1.1. An example of Dijkstra's algorithm: Used as an Input for FPGA

***Jupyter Notebook is configured to generate the graph, in the last block.

Improvement Ideas:

This project has implemented Dijkstra algorithm with fixed input size i.e. array of size 8 x 8 , along with fixed reference point (0^{th} position) i.e. vertex in the graph from where the shortest distance needs to be calculated to all other vertices. It will of great advantage if this project is upgraded for dynamic inputs size and dynamic reference point.

References:

1. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_2/ug902-vivado-high-level-synthesis.pdf
2. https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
3. https://www.programmersought.com/article/90206197757/
4. https://ilearn.th-deg.de/mod/url/view.php?id=348864

Errors Faced with Solutions:

1. DMA Receive channel Issue
   Initially DMA were used where we did not get the output. Below command was running forever in jupyter while fetching the results-
   dma.recvchannel.wait()
   As an alternative the Overlay was redesigned without DMA, with registers of ZYNQ processsor.

Results and solution :

– The results showed that FPGA performed 50-80% better than CPU on average.
– In several instances, the FPGA was able to compute results 150-200 percent faster than the CPU.
– As a result, we can conclude that the FPGA outperformed the CPU.