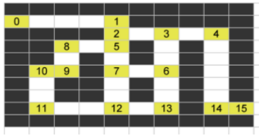
CSEE 4840

Embedded Systems



LABYRINTH

Dijkstra's implementation on FPGA

Ariel Faria
Michelle Valente
Utkarsh Gupta
Veton Saliu

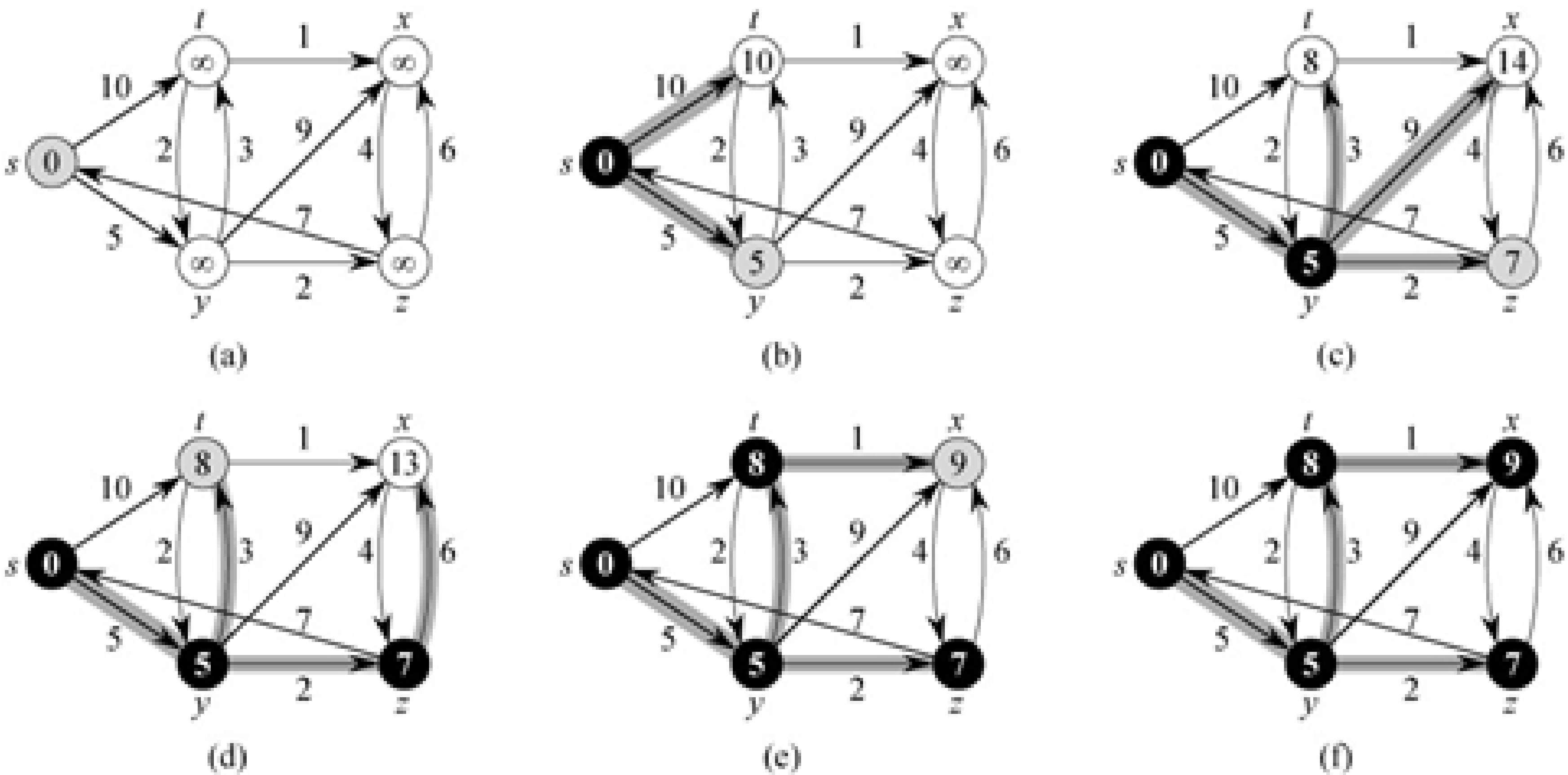
Under the guidance of – Prof. Stephen Edwards

Overview and objectives

- Single source shortest path
- Dijkstra's and properties
- Sequential queues and growth
- Advantages of Dijkstra's on reconfigurable hardware and applications
- In particular maze router – CAD APR

- Implement the algorithm on FPGA and compute best path on hardware
  - Scale up to accommodate more nodes
  - Display the solved maze on the monitor
  - Benchmarking time

Dijkstra’s algorithm



Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). "Section 24.3: Dijkstra's algorithm". Introduction to Algorithms (Second ed.). MIT Press and McGraw-Hill. pp. 595–601. ISBN 0-262-03293-7.

Project Flow

Software prototype

- To understand the steps and constraints of the algorithms.
- Establish credibility for maze solving.

Hardware implementation

- Designed basic network
- Memory modules
- Comparator blocks
- Hard wire 32 node network
- Implemented Dijkstra's

Software driver

- Software generates maze
- Translates to network
- Communicates the network to FPGA

Scale up and add-ons

- Network display through software
- Implement for a 512 node network

Software Prototypes

- Two steps
  - Sequential, classic implementation
  - Using structures similar to hardware to confirm the correctness of parallel implementation

```
int minDistance(int dist[], bool visited[])
{
   int min_distance = INT_MAX;
   int min_vertex;

   for (int i = 0; i < numV; i++)
     if (visited[i] == false && dist[i] <= min_distance)
     {
        min_vertex = i;
        min_distance = dist[i];
     }
```

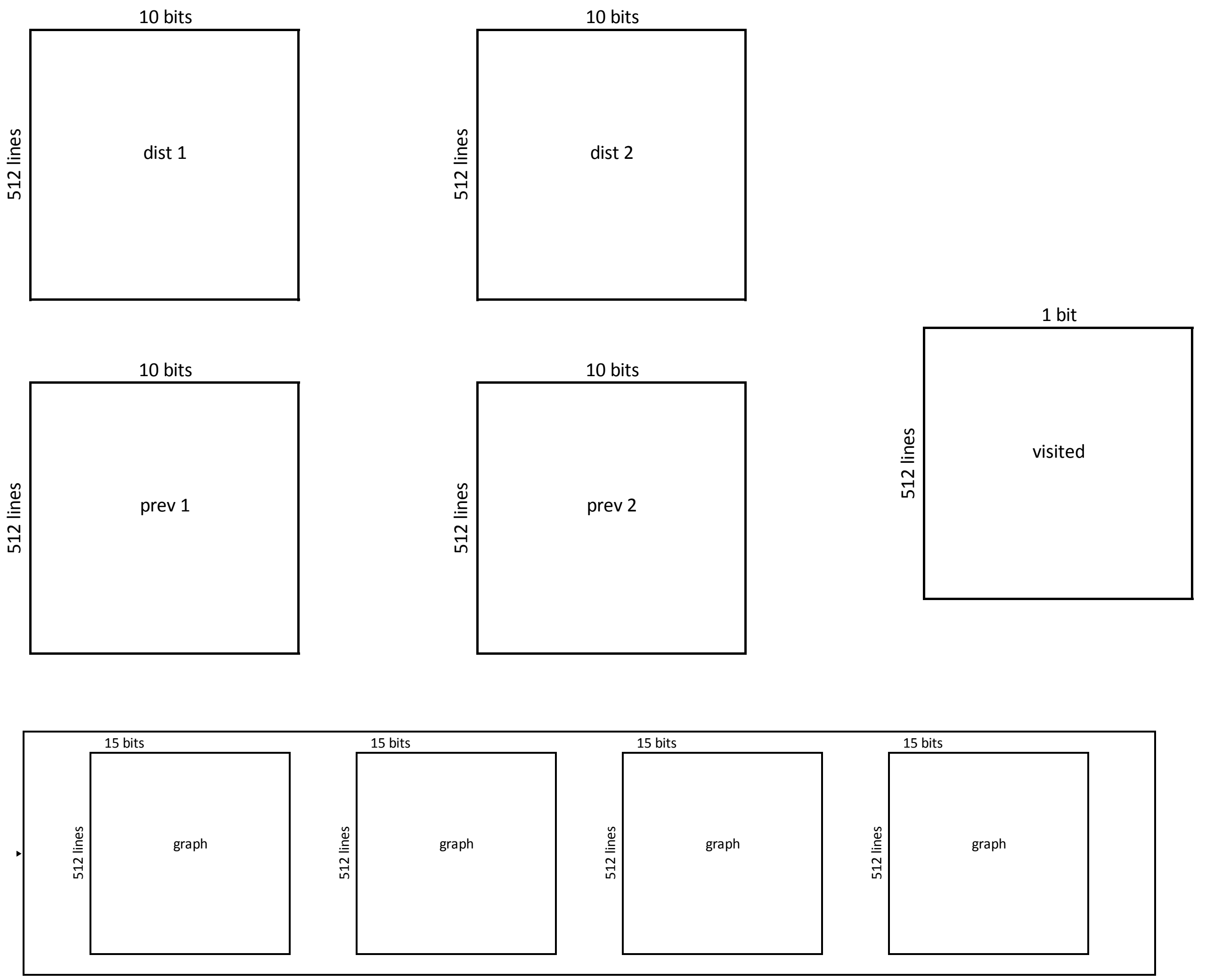
```
//Distance memory -- 32 parallel
    int distance0[32];
    int distance1[32];

//Graph memory -- 4blocks of 32 l
    int graph0[32][2];
    int graph1[32][2];
    int graph2[32][2];
    int graph3[32][2];

//visit memory -- Array of 32 FF'
    int visit[32];
```

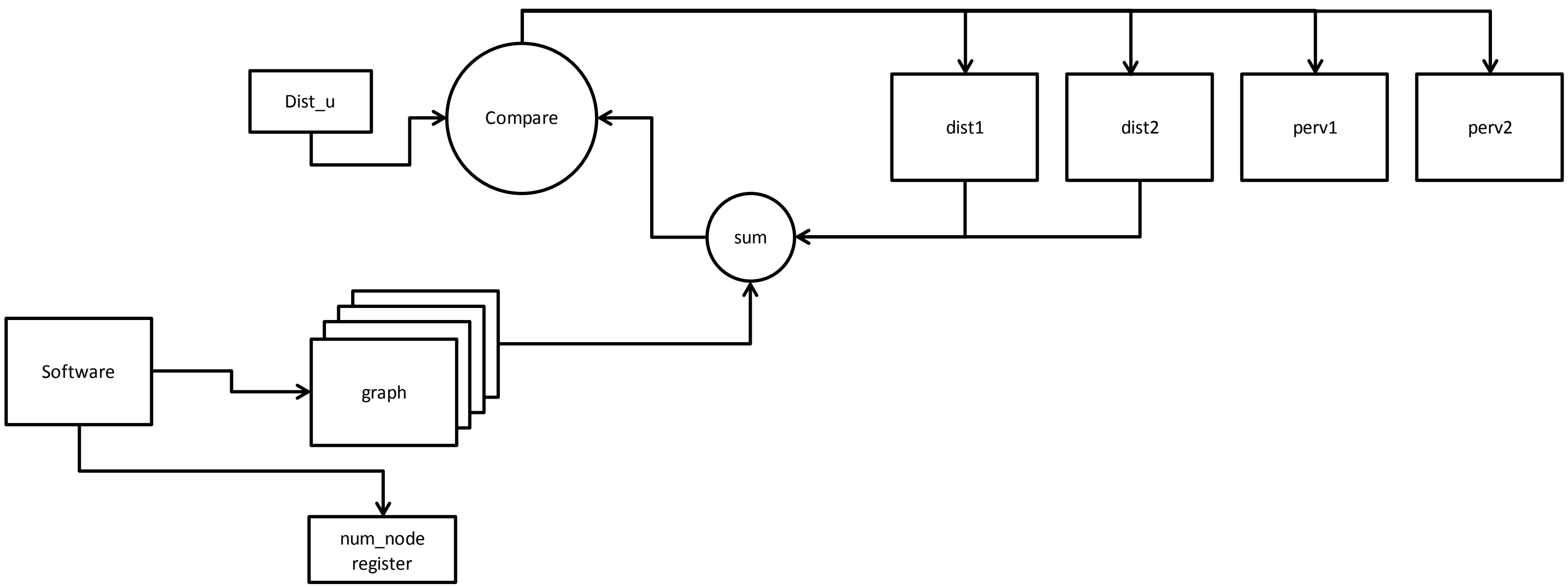
Hardware Implementations

Memory modules

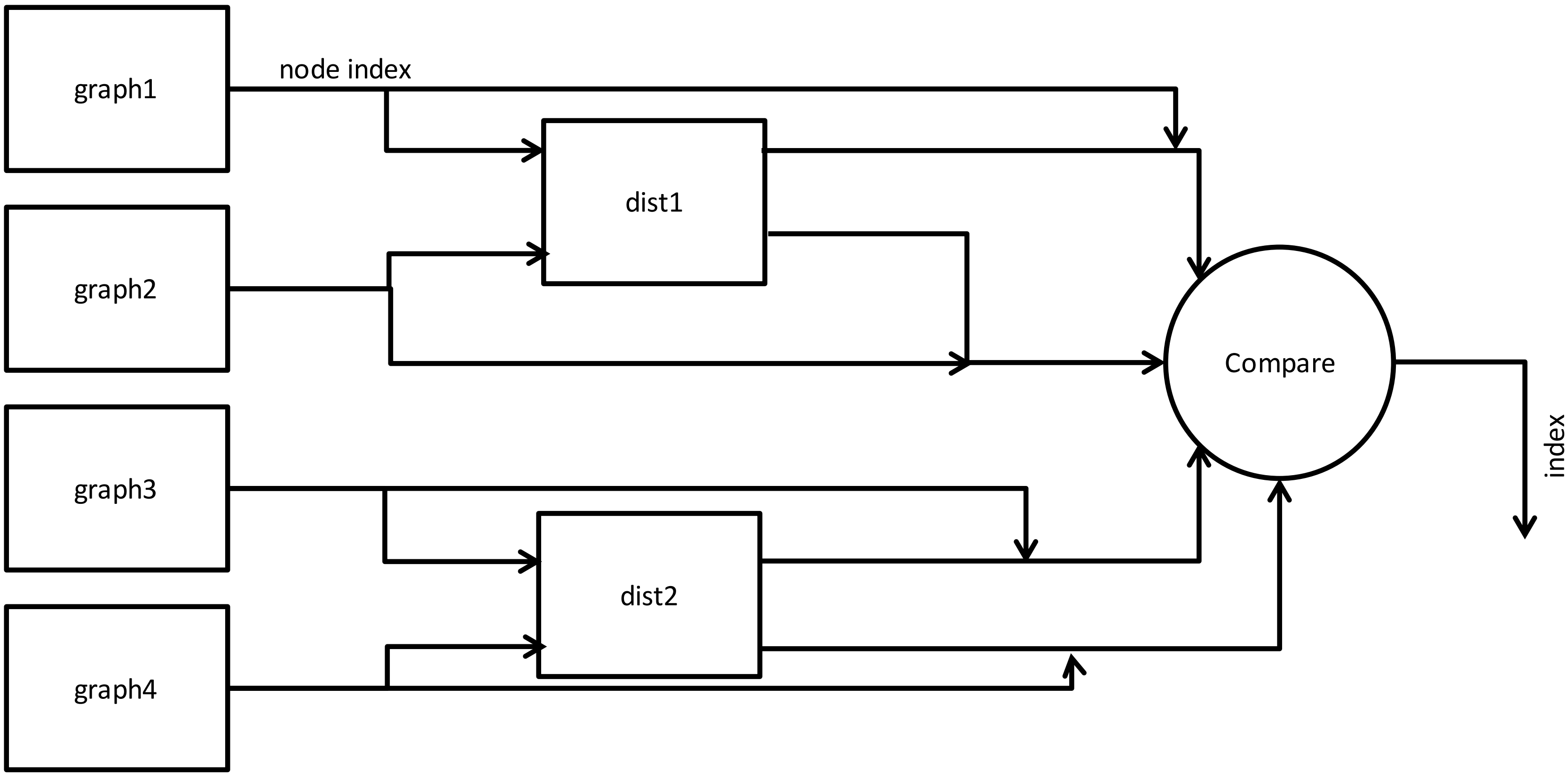


Architecture (datapath)

- Comparing
- Updating

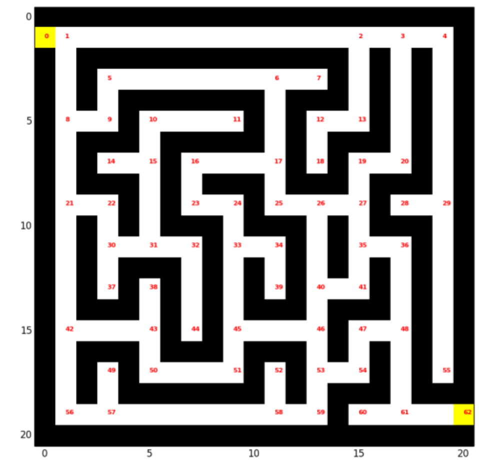


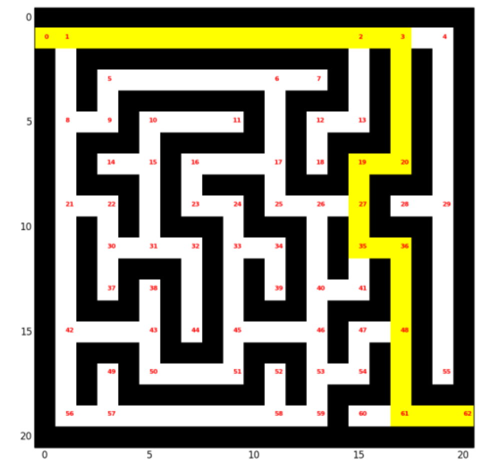
Minimum Distance Node Finder



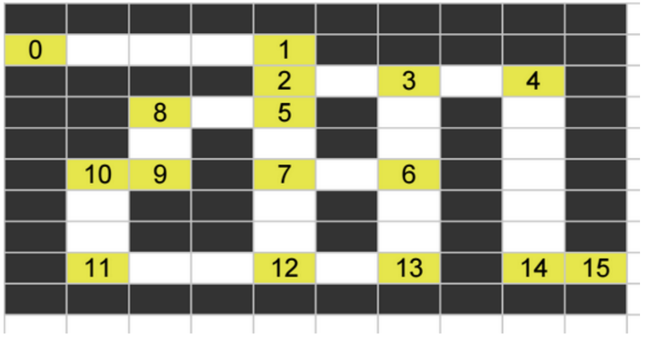
Software and Driver

- Software spits out a random network
- Sends this information in 32 bits to the FPGA
- FPGA computes the minimum distance and displays on the monitor
- Software sends the solved maze to the user monitor





Experiences and Issues



- Monitor first, wrong approach
  SOLN: algorithm implementation
- Maze size too big too ambitious
  SOLN: 32 node smaller network
- Optimal structures for the memory modules for scaling up and parallel reads and stores
- Algorithm
  - Comparing the neighbors but ended in dead end
    SOLN: Compare all nodes
- Memory corruption
  SOLN: explicitly set values to reg in each state
- Debugging and high compile time

Summary

- Lessons learned
  - Not to violate setup or hold times by trying to fit heavy computation within a clock cycle; either make computations more efficient/ fast or allocate multiple clock cycles for the computation.
  - Allocating two dual port memory blocks to both the previous and distance data as opposed to allocating a separate module per node
  - There are two modules for scalability and efficient use of memory resources
  - Test the hardware after adding extra cycles of computation, makes it easier to debug and therefore reduces development time
  - We initially planned to compare all the distances but we found that that would be too costly in terms of the hardware we generated for a minor improvement in performance instead we decided to perform the comparison stage of the algorithm 4 nodes at a time on each clock cycle