

Likitha Atluru & Artem Kulakevich
Professor Xiaoyu Song
TA Peng Gao
ECE-527 High-Performance Digital Systems
21 Oct 2020

Project #1: Retiming Process

Introduction:

We have to create a program that will go through the retiming process of a circuit. The circuit is made more abstract by having the combination logic be represented as nodes connected together by edges (registers) with direction for dataflow. We will convert this representation into a matrix to hold the edge connections and an array to hold the propagation delay. Using the matrix format makes it possible to modify the data in C.

Input: Circuit Model G, fixed clock cycle c

Our circuit will be represented using arrays. We represent our circuit using an N by N array. Here is an example of our representation for the circuit presented in class:

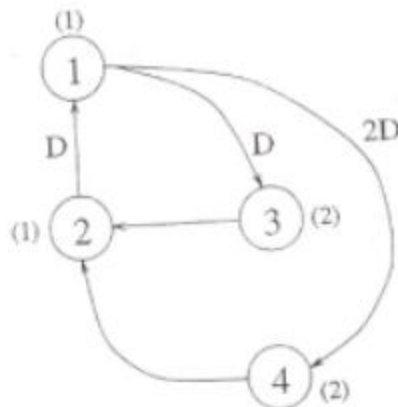


Figure 1: Class circuit example.

We represent each node as a row and column in the matrix. Each D (weight) is placed in the matrix going from the node in the row to the node in the column. For example the transition from 2 to 1. Will result in the value 1 in row 2, column 1. As such we have the following matrix:

```
int init_graph[N][N] = {
    /*1*/ /*2*/ /*3*/ /*4*/
    /*1*/ {inf, inf, 1, 2},
    /*2*/ { 1, inf, inf, inf},
    /*3*/ {inf, 0, inf, inf},
    /*4*/ {inf, 0, inf, inf}};
```

We also store the propagation delay of each node in its own array:

```
int dU[N] = {1,1,2,2}; // propagation delays of each circuit
```

Finally the c value is simply input as an int.

Compute W and D using G' and SP

To find G' we used the equation provided in class:

$$\mathbf{w'}(e) = \mathbf{M} \times \mathbf{w}(e) - \mathbf{d}(U), \text{ where } \mathbf{M} = \mathbf{d_{Max}} * \mathbf{N}$$

We implemented that equation using the following:

```
Gp.a[i][j] = N * DMAX * G.a[i][j] - dU[i];
```

Shortest Path Function

We decided to use the Floyd-Warshall algorithm to compute our shortest path. We implemented that by creating a matrix class and the shortest calculation as the method for that equation:

```
void Matrix::sp() {
int dp[N+1][N][N] = {0};
for (int v = 0; v < N; v++){
    for (int u = 0; u < N; u++){
        dp[0][u][v] = a[u][v];
    }
}
for (int k = 0; k < N; k++){
    for (int v = 0; v < N; v++){
        for (int u = 0; u < N; u++){
            dp[k+1][u][v] = dp[k][u][v];
            if (dp[k+1][u][v] > (dp[k][u][k]+dp[k][k][v])
                && !(dp[k][u][k]+dp[k][k][v]+dp[k+1][u][v] > 19000)) {
                dp[k+1][u][v] = dp[k][u][k] + dp[k][k][v];
            }
        }
    }
}
for (int v = 0; v < N; v++){
    for (int u = 0; u < N; u++){
        a[u][v] = dp[N][u][v];
    }
}
};
```

This is almost a complete copy of the lecture slide equation for Floyd-Washall. It works identically, and it provides us with the following result when using it on G':

Graph Suv =
 1| 12 5 7 15
 2| 7 12 14 22
 3| 5 -2 12 20
 4| 5 -2 12 20

Which is identical to what we see in the slides:

S'_{uv}	1	2	3	4
1	12	5	7	15
2	7	12	14	22
3	5	-2	12	20
4	5	-2	12	20

Figure 2: Class circuit Suv' as a matrix.

From this we are able to find W and D using the equations in class:

If $U \neq V$, then

$$W(U, V) = \left\lceil \frac{S'_{UV}}{M} \right\rceil$$

$$D(U, V) = M \times W(U, V) - S'_{UV} + d(V)$$

If $U = V$, then

$$\square W(U, V) = 0 \quad \text{and} \quad D(U, V) = d(U)$$

Implemented like this:

```
/* Finding W(U,V) */
int w[N][N] = {0};
for (int v = 0; v < N; v++){
    for (int u = 0; u < N; u++){
        if (u!=v){
            w[u][v] = ceil(((float) Gp.a[u][v]) / (N * DMAX)); // Gp reps Suv
        }
    }
}
```

And for D(U, V):

```
/* Finding D(U,V) */
int d[N][N] = {0};
for (int v = 0; v < N; v++){
    for (int u = 0; u < N; u++){
        if (u!=v){
            d[u][v] = (N * DMAX) * w[u][v] - Gp.a[u][v] + dU[v];
        } else {
            d[u][v] = dU[u];
        }
    }
}
```

As a result, we get the same outputs as class:

$W(u,v)$	1	2	3	4
1	0	1	1	2
2	1	0	2	3
3	1	0	0	3
4	1	0	2	0

Graph W =

1	0	1	1	2
2	1	0	2	3
3	1	0	0	3
4	1	0	2	0

$D(u,v)$	1	2	3	4
1	1	4	3	3
2	2	1	4	4
3	4	3	2	6
4	4	3	6	2

Graph D =

1	1	4	3	3
2	2	1	4	4
3	4	3	2	6
4	4	3	6	2

Formulate the inequalities from G, W, D, and c

For C = 2 We get the following inequalities:

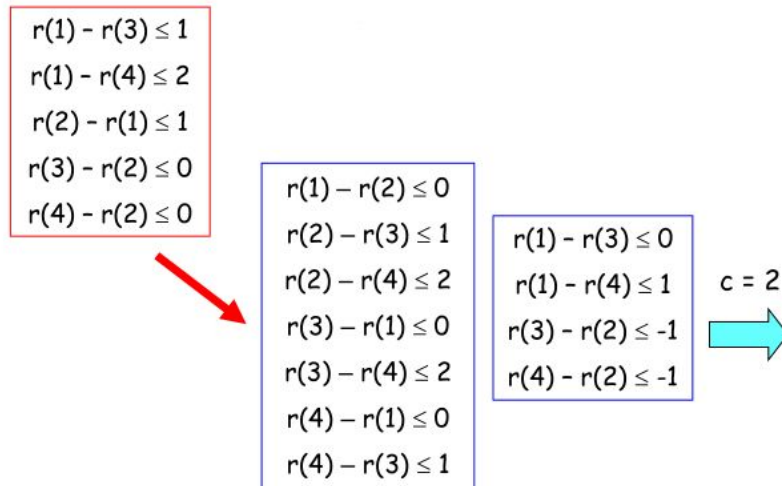


Figure 3: Class circuit inequalities.

Using the equation $W(U,V) - 1$ to find our inequality we get the matrix form of the inequalities as such:

```
Graph Inequalities =  
1| 0  0  0  1  
2| 1  0  1  2  
3| 0 -1  0  2  
4| 0 -1  1  0
```

Solve the inequalities by constraint graph and SP

Now using the shortest path algorithm on our inequalities we get the following results:

```
Graph calculated Inequalities after SP() =  
1| 0 -1  0  1  
2| 1  0  1  2  
3| 0 -1  0  1  
4| 0 -1  0  0
```

We can see that column 2 is the smallest, so we will use that for our solution:

```
R1(-1),  
R2(0),  
R3(-1),  
R4(-1),
```

So we get R1(-1), R2(0), R3(-1), R4(-1) which is the expected function. We can then modify the original graph with these values and get the following result:

Original	Retimed for C = 2
<p>Graph G =</p> <pre> 1 9999 9999 1 2 2 1 9999 9999 9999 3 9999 0 9999 9999 4 9999 0 9999 9999 </pre> <p>(9999 = infinity)</p>	<p>Retimed Graph, Gr:</p> <pre> 1 9999 9999 1 2 2 0 9999 9999 9999 3 9999 1 9999 9999 4 9999 1 9999 9999 </pre>
Figure 4: Class Circuit before retiming	Figure 5: Class Circuit after retiming

Is c feasible?

C=2 is found to be feasible, but C=1 fails:

Inequality expression is FALSE, ending program. Failed c = 1

Retimed?

By examining the result we can see that c is now feasible, there are no paths that are longer than 2.

Example Circuit #2:

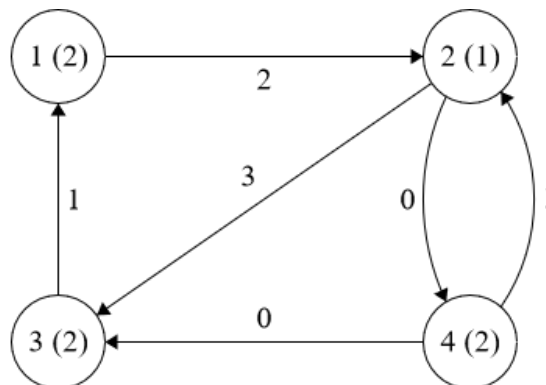
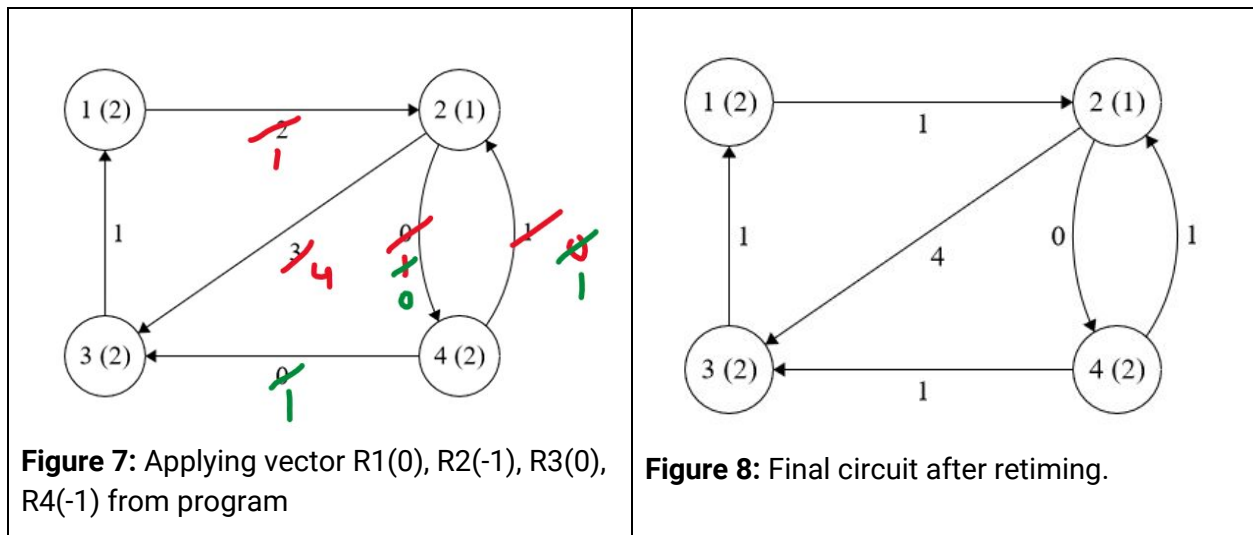


Figure 6: New different circuit for testing

G	<pre>Graph G = 1 99 2 99 99 2 99 99 3 0 3 1 99 99 99 4 99 1 0 99</pre> <p>(99 is infinity)</p>
G'	<pre>Graph G' = 1 99 14 99 99 2 99 99 23 -1 3 6 99 99 99 4 99 6 -2 99</pre>
Suv'	<pre>Graph Suv = 1 17 14 11 13 2 3 5 -3 -1 3 6 20 17 19 4 4 6 -2 5</pre>
W and D	<pre>Graph W = 1 0 2 2 2 2 1 0 0 0 3 1 3 0 3 4 1 1 0 0 Graph D = 1 2 3 7 5 2 7 1 5 3 3 4 5 2 7 4 6 3 4 2</pre>
Inequalities	<pre>Graph Inequalities = 1 0 2 1 1 2 0 0 -1 0 3 0 2 0 2 4 0 1 -1 0</pre>
Retiming Vector	<pre>Graph calculated Inequalities after SP() = 1 0 2 0 1 2 -1 0 -1 0 3 0 2 0 1 4 -1 1 -1 0 R1(0), R2(-1), R3(0), R4(-1),</pre>

	Found retiming vector $R1(0), R2(-1), R3(0), R4(-1)$
Retimed Graph	<pre>Retimed Graph, Gr: 1 99 1 99 99 2 99 99 4 0 3 1 99 99 99 4 99 1 1 99</pre>



So there are no negative cycles, and the delays fill the requirements $c = 3$.

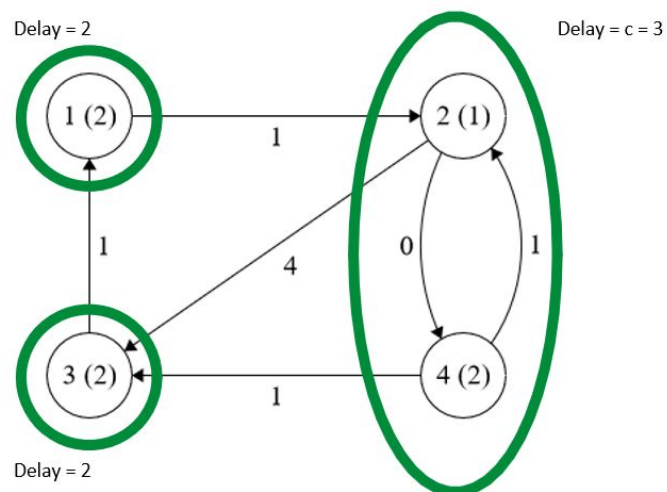


Figure 9: Showing that $c = 3$ is valid.

$C=2$ fails, because it's not possible to meet the requirements by simply moving registers around.

Example Circuit #3:

Original:

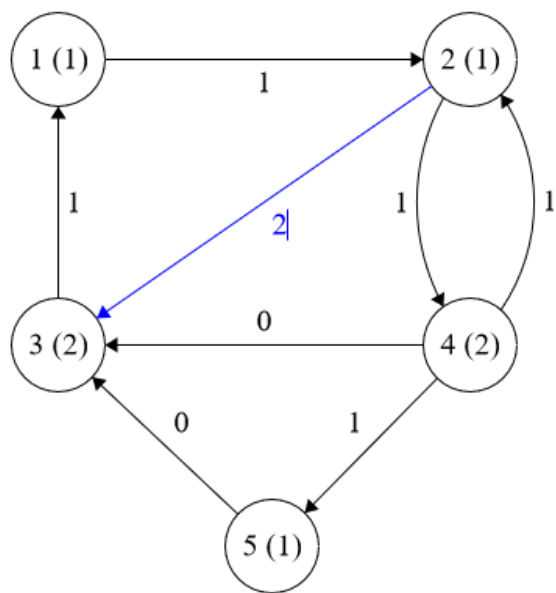
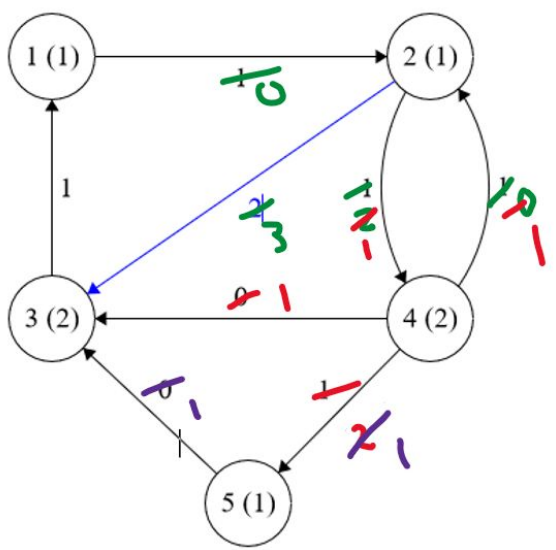
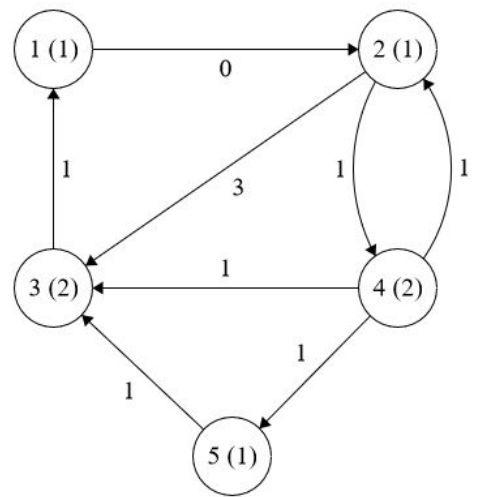


Figure 10: New test circuit.

G	<div>Graph G = 1 99 1 99 99 99 2 99 99 2 1 99 3 1 99 99 99 99 4 99 1 0 99 1 5 99 99 0 99 99</div> <div>(99 = infinity)</div>
G'	<div>Graph G' = 1 99 9 99 99 99 2 99 99 19 9 99 3 8 99 99 99 99 4 99 8 -2 99 8 5 99 99 -1 99 99</div>
Suv'	<div>Graph Suv = 1 24 9 16 18 26 2 15 17 7 9 17 3 8 17 24 26 34 4 6 8 -2 17 8 5 7 16 -1 25 33</div>

<p>W and D</p>	<pre> Graph W = 1 0 1 2 2 3 2 2 0 1 1 2 3 1 2 0 3 4 4 1 1 0 0 1 5 1 2 0 3 0 Graph D = 1 1 2 6 4 5 2 6 1 5 3 4 3 3 4 2 6 7 4 5 3 4 2 3 5 4 5 3 7 1 </pre>
<p>Inequalities</p>	<pre> Graph Inequalities = 1 0 1 1 1 2 2 1 0 0 0 1 3 0 1 0 2 3 4 0 0 -1 0 0 5 0 1 -1 2 0 </pre>
<p>Retiming Vector</p>	<pre> Graph calculated Inequalities after SP() = 1 0 1 0 1 1 2 -1 0 -1 0 0 3 0 1 0 1 1 4 -1 0 -1 0 0 5 -1 0 -1 0 0 R1(0), R2(-1), R3(0), R4(-1), R5(-1), </pre> <p>Retiming vector found: R1(0), R2(-1), R3(0), R4(-1), R5(-1)</p>
<p>Retimed Graph</p>	<pre> Retimed Graph, Gr: 1 99 0 99 99 99 2 99 99 3 1 99 3 1 99 99 99 99 4 99 1 1 99 1 5 99 99 1 99 99 </pre>

Original Retiming Process	Final Retimed
 <p>Figure 11: Retimed with $R1(0)$, $R2(-1)$, $R3(0)$, $R4(-1)$, $R5(-1)$</p>	 <p>Figure 12: Result of program retiming</p>

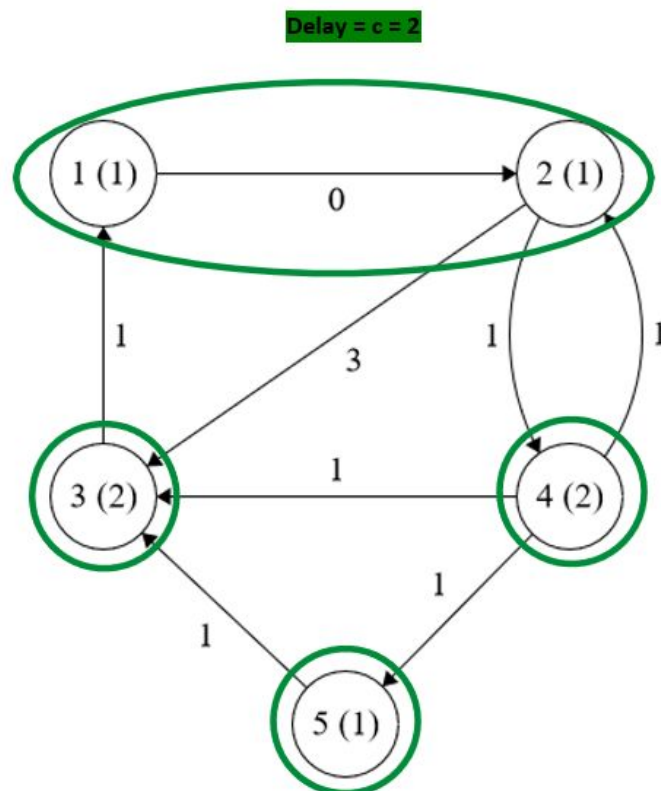


Figure 13: Showing that $c = 2$ requirement is met.

The program then fails to find a solution for $c = 1$, because our smallest delay is 2. But the program successfully found a solution for $c = 2$ as we can see above.

```
Inequality expression is FALSE, ending program. Failed c = 1
```

Conclusion

The program is working for our original class solution, and for 4 different circuits. Including a 5 nodes circuit. The program looks like it is working well, but more extensive testing would be needed to confirm that it is flawless. Also better implementations of infinity would be nice, but were not necessary for now. We were able to find the correct solution for every circuit that we tested the program with, and even some that were not described in this project. It would be interesting we were able to use some kind of visualizing tool to have the program convert the circuit directly into a state machine with nodes and connections, but that would require much more time.