**Prepared By:**   
Ben Earle (100970237)

**Prepared For:**  
Dr. Gabriel Wainer

Firing Squad Synchronization Problem

Assignment 2 – SYSC 5104 – Fall 2018

# Part I – Description:

## Problem Statement

The Firing Squad Synchronization Problem (FSSP) proposes there is line of soldiers of an unknown length with a general at one end. The FSSP asks how we can get all the soldiers to fire simultaneously given that each soldier can only communicate with their neighboring soldiers. The goal of the problem is to find a solution with the least time that also has the least number of states. The neighborhood in the context of this problem is the two adjacent cells in a 1D array or the von Neumann neighborhood in a 2D array. Since it was originally proposed many solutions were found with varying time and state counts, this assignment will implement three solutions proposed in separate papers.

## Model

The first solution to be implemented is described in *Smaller Solutions for the Firing Squad*. The authors take the 1D problem and expands it by allowing the general to be in any position in the array. In this paper they expand on Mazoyer’s 6-state minimum time solution (O(2n-2)) using an additional state to find the edge of the array. To accomplish the minimum time solution, Mazoyer splits the array into sub arrays by propagating different speed signals. The Signals meet in the center and create new generals, and the process repeats, until all the cells are a generals then they fire. The state transition table for this adaptation is shown in figure 1, followed by a graph showing each cell state after each step in figure 2. After implementing the 1D solution I would like to model it in 2D, where each step the simulation moves down as shown in figure 2.

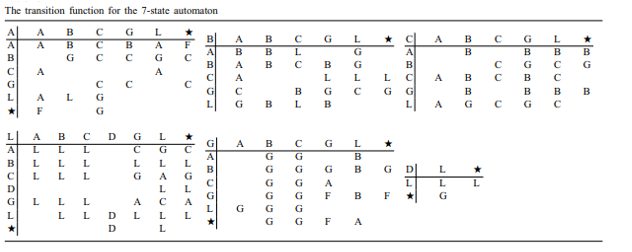


Figure 1: Mazoyer’s original 6-state solution, with an added state D to locate left most corner.

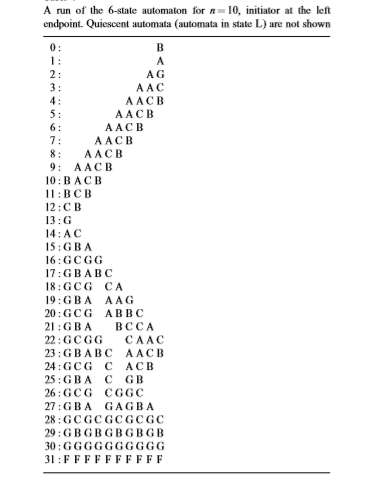


Figure 2: Output of Mazoyer’s 6-state solution

The second solution to be implemented will be an extension of the first model to solve rectangular two-dimensional cell spaces in a non-time optimal manor. I thought of this solution with low state complexity yet time inefficient while researching and would like to try implementing it. The easiest way to synchronize a rectangular 2D cell space is to do an optimal 6-state synchronization of the first columns only at the end, instead of firing turn all the cells into their own row’s general. Although this is very time inefficient (O(2n + 2m - 4)) compared to the optimal time (O(n + m + max(n,m) – 3)), this technique is simple and uses a very similar rule set to solution one.

If time permits, the final solution I would like to implement is I will implement an optimal time solution (O(2n-2)) for square arrays using Umao’s 7-states algorithm proposed in his paper *A Seven-State Time-Optimum Square Synchronizer*. I have included the state diagram in the Appendix. There are many state transitions since Umao listed every combination of transitions given 7 states and a von Neuman diagram. I am working on condensing this list by filtering out all transitions to the same state and other equivalent rules.

# Part II – Model:

## 1D-FSSP

Conceptual model = < X, Y, I, S, n, {t­1, t­2}, N, >

*X* = *Y*= ;

I = < Px, Py>,

= 3;



Pi,j Y1  Pi,j-1 X1 Pi,j+1 Y1  Pi,j X1

Pi,j Y2  Pi,j+1 X2 Pi,j-1 Y2  Pi,j X2

Pi,j Y3  Pi,j X3 Pi,j Y3  Pi,j X3

S = {L, S, A, B, C, G, F};

n = 2;

{t­1 = Number of nodes +2, t­2 = 1},

N = { (0,-1), (0, 0), (0,1)};

Transitions are easiest to see in Figure 1. The complete ruleset using sudo-CellDevs notation is as follows:

rule : {L} 100 { (0,0) = D and (0,-1) = L}

rule : {G} 100 { (0,0) = D and (0,-1) = S}

rule : {C} 100 { (0,0) = L and (0,-1) = A and ((0,1) = G or (0,1) = S)}

rule : {G} 100 { (0,0) = L and (0,-1) = A and (0,1) = L}

rule : {A} 100 { (0,0) = L and (0,-1) = C and (0,1) = L}

rule : {G} 100 { (0,0) = L and (0,-1) = C and ((0,1) = G or (0,1) = S)}

rule : {C} 100 { (0,0) = L and (0,-1) = G and (0,1) = L}

rule : {A} 100 { (0,0) = L and (0,-1) = G and ((0,1) = G or (0,1) = S)}

rule : {D} 100 { (0,0) = L and (0,-1) = L and (0,1) = D}

rule : {D} 100 { (0,0) = L and (0,-1) = S and (0,1) = D}

rule : {B} 100 { (0,0) = G and ((0,-1) = A or (0,-1) = B or (0,-1) = G) and (0,1) = L}

rule : {A} 100 { (0,0) = G and (0,-1) = C and ((0,1) = G or (0,1) = L or (0,1) = S)}

rule : {A} 100 { (0,0) = G and (0,-1) = S and (0,1) = L}

rule : {F} 100 { (0,0) = G and ((0,-1) = G or (0,-1) = S) and ((0,1) = G or (0,1) = S)}

rule : {B} 100 { (0,0) = A and (0,-1) = A and ((0,1) = B or (0,1) = G)}

rule : {C} 100 { (0,0) = A and (0,-1) = B and ((0,1) = C or (0,1) = G or (0,1) = S)}

rule : {C} 100 { (0,0) = A and (0,-1) = A and (0,1) = C}

rule : {C} 100 { (0,0) = A and (0,-1) = G and ((0,1) = C or (0,1) = G or (0,1) = S)}

rule : {G} 100 { (0,0) = A and (0,-1) = B and ((0,1) = B or (0,1) = L)}

rule : {G} 100 { (0,0) = A and ((0,-1) = S or (0,-1) = L) and (0,1) = C}

rule : {L} 100 { (0,0) = A and (0,-1) = L and (0,1) = B}

rule : {F} 100 { (0,0) = A and (0,-1) = A and (0,1) = S}

rule : {F} 100 { (0,0) = A and (0,-1) = S and (0,1) = A}

rule : {L} 100 { (0,0) = B and ((0,-1) = A or (0,-1) = L) and (0,1) = C}

rule : {L} 100 { (0,0) = B and (0,-1) = C and ((0,1) = G or (0,1) = L or (0,1) = S)}

rule : {A} 100 { (0,0) = B and ((0,-1) = B or (0,-1) = C) and (0,1) = A}

rule : {C} 100 { (0,0) = B and (0,-1) = B and (0,1) = C}

rule : {C} 100 { (0,0) = B and (0,-1) = G and ((0,1) = A or (0,1) = L)}

rule : {G} 100 { (0,0) = B and (0,-1) = L and (0,1) = A}

rule : {G} 100 { (0,0) = B and (0,-1) = G and ((0,1) = G or (0,1) = S)}

rule : {G} 100 { (0,0) = B and ((0,-1) = A or (0,-1) = B) and (0,1) = L}

rule : {A} 100 { (0,0) = C and ((0,-1) = L or (0,-1) = C) and (0,1) = A}

rule : {G} 100 { (0,0) = C and (0,-1) = B and ((0,1) = G or (0,1) = S)}

rule : {G} 100 { (0,0) = C and (0,-1) = L and ((0,1) = B or (0,1) = G)}

rule : {B} 100 { (0,0) = C and (0,-1) = A and ((0,1) = B or (0,1) = G or (0,1) = L or (0,1) = S)}

rule : {B} 100 { (0,0) = C and (0,-1) = G and ((0,1) = B or (0,1) = G or (0,1) = L or (0,1) = S)}

rule : {B} 100 { (0,0) = C and (0,-1) = C and ((0,1) = B or (0,1) = G)}

rule : {(0,0)} 100 { t }

## 1DxTime-FSSP

Conceptual model = < X, Y, I, S, n, {t­1, t­2}, N, >

*X* = *Y*= ;

I = < Px, Py>,

= 4;



Pi,j Y1  Pi-1,j-1 X1 Pi+1,j+1 Y1  Pi,j X1

Pi,j Y2  Pi-1,j+1 X2 Pi+1,j-1 Y2  Pi,j X2

Pi,j Y3  Pi-1,j X3 Pi+1,j Y3  Pi,j X3

Pi,j Y4  Pi,j X4 Pi,j Y4  Pi,j X4

S = {L, S, A, B, C, G, F};

n = 2;

{t­1 = Number of nodes +2, t­2 = Number of nodes \*3 - 1},

N = { (1,-1), (1, 0), (1,1), (0,0)};

 Transitions are easiest to see in Figure 1. The only change is the new state is taken on by a row below instead of existing row. Also two additional rules are required: If the state above is L or F do not take on the new state. The complete ruleset using sudo-CellDevs notation is as follows:

rule : {L} 100 { (-1,0) = D and (-1,-1) = L}

rule : {G} 100 { (-1,0) = D and (-1,-1) = S}

rule : {C} 100 { (-1,0) = L and (-1,-1) = A and ((-1,1) = G or (-1,1) = S)}

rule : {G} 100 { (-1,0) = L and (-1,-1) = A and (-1,1) = L}

rule : {A} 100 { (-1,0) = L and (-1,-1) = C and (-1,1) = L}

rule : {G} 100 { (-1,0) = L and (-1,-1) = C and ((-1,1) = G or (-1,1) = S)}

rule : {C} 100 { (-1,0) = L and (-1,-1) = G and (-1,1) = L}

rule : {A} 100 { (-1,0) = L and (-1,-1) = G and ((-1,1) = G or (-1,1) = S)}

rule : {D} 100 { (-1,0) = L and (-1,-1) = L and (-1,1) = D}

rule : {D} 100 { (-1,0) = L and (-1,-1) = S and (-1,1) = D}

rule : {B} 100 { (-1,0) = G and ((-1,-1) = A or (-1,-1) = B or (-1,-1) = G) and (-1,1) = L}

rule : {A} 100 { (-1,0) = G and (-1,-1) = C and ((-1,1) = G or (-1,1) = L or (-1,1) = S)}

rule : {A} 100 { (-1,0) = G and (-1,-1) = S and (-1,1) = L}

rule : {F} 100 { (-1,0) = G and ((-1,-1) = G or (-1,-1) = S) and ((-1,1) = G or (-1,1) = S)}

rule : {B} 100 { (-1,0) = A and (-1,-1) = A and ((-1,1) = B or (-1,1) = G)}

rule : {C} 100 { (-1,0) = A and (-1,-1) = B and ((-1,1) = C or (-1,1) = G or (-1,1) = S)}

rule : {C} 100 { (-1,0) = A and (-1,-1) = A and (-1,1) = C}

rule : {C} 100 { (-1,0) = A and (-1,-1) = G and ((-1,1) = C or (-1,1) = G or (-1,1) = S)}

rule : {G} 100 { (-1,0) = A and (-1,-1) = B and ((-1,1) = B or (-1,1) = L)}

rule : {G} 100 { (-1,0) = A and ((-1,-1) = S or (-1,-1) = L) and (-1,1) = C}

rule : {L} 100 { (-1,0) = A and (-1,-1) = L and (-1,1) = B}

rule : {F} 100 { (-1,0) = A and (-1,-1) = A and (-1,1) = S}

rule : {F} 100 { (-1,0) = A and (-1,-1) = S and (-1,1) = A}

rule : {L} 100 { (-1,0) = B and ((-1,-1) = A or (-1,-1) = L) and (-1,1) = C}

rule : {L} 100 { (-1,0) = B and (-1,-1) = C and ((-1,1) = G or (-1,1) = L or (-1,1) = S)}

rule : {A} 100 { (-1,0) = B and ((-1,-1) = B or (-1,-1) = C) and (-1,1) = A}

rule : {C} 100 { (-1,0) = B and (-1,-1) = B and (-1,1) = C}

rule : {C} 100 { (-1,0) = B and (-1,-1) = G and ((-1,1) = A or (-1,1) = L)}

rule : {G} 100 { (-1,0) = B and (-1,-1) = L and (-1,1) = A}

rule : {G} 100 { (-1,0) = B and (-1,-1) = G and ((-1,1) = G or (-1,1) = S)}

rule : {G} 100 { (-1,0) = B and ((-1,-1) = A or (-1,-1) = B) and (-1,1) = L}

rule : {A} 100 { (-1,0) = C and ((-1,-1) = L or (-1,-1) = C) and (-1,1) = A}

rule : {G} 100 { (-1,0) = C and (-1,-1) = B and ((-1,1) = G or (-1,1) = S)}

rule : {G} 100 { (-1,0) = C and (-1,-1) = L and ((-1,1) = B or (-1,1) = G)}

rule : {B} 100 { (-1,0) = C and (-1,-1) = A and ((-1,1) = B or (-1,1) = G or (-1,1) = L or (-1,1) = S)}

rule : {B} 100 { (-1,0) = C and (-1,-1) = G and ((-1,1) = B or (-1,1) = G or (-1,1) = L or (-1,1) = S)}

rule : {B} 100 { (-1,0) = C and (-1,-1) = C and ((-1,1) = B or (-1,1) = G)}

rule : {(0,0)} 100 {(-1,0) = L or (-1,0) = F}

rule : {(-1,0)} 100 { t }

## 2D-FSSP

Conceptual model = < X, Y, I, S, n, {t­1, t­2}, N, >

*X* = *Y*= ;

I = < Px, Py>,

= 5;



Pi,j Y1  Pi-1,j X1 Pi+1,j Y1  Pi,j X1

Pi,j Y2  Pi+1,j X2 Pi-1,j Y2  Pi,j X2

Pi,j Y3  Pi,j+1 X3 Pi,j-1 Y3  Pi,j X3

Pi,j Y4  Pi,j-1 X4 Pi,j+1 Y3  Pi,j X3

Pi,j Y5  Pi,j X5 Pi,j Y4  Pi,j X4

S = {L, S, Ax, Bx, Cx, Gx, F, Ay, By, Cy, Gy };

n = 2;

{t­1 = Number of nodes in X +2, t­2 = Number of nodes in Y + 2},

N = { (-1,0), (1, 0), (0,1), (0,-1), (0,0)};

Transitions are inspired by those shown in Figure 1. The major changes are that D now searches for the top left most cell before turning into a Gx cell. The Gx cell will send Ax, Bx, and Cx signals to synchronize the top row. Instead of changing to F state when both neighbours have the Gx state they transition to Gy state. Gy state will use the Ay, By, and Cy signals to synchronize the columns. Once the coloumns are all Gy then the cells will enter the F state. The complete ruleset using sudo-CellDevs notation is as follows:

rule : {L} 100 { (0,0) = D and (0,-1) = L}

rule : {L} 100 { (0,0) = D and (0,-1) = S and (-1,0) = L}

rule : {Gx} 100 { (0,0) = D and (0,-1) = S and (-1,0) = S}

rule : {Cx} 100 { (0,0) = L and (0,-1) = Ax and ((0,1) = Gx or (0,1) = S)}

rule : {Gx} 100 { (0,0) = L and (0,-1) = Ax and (0,1) = L}

rule : {Ax} 100 { (0,0) = L and (0,-1) = Cx and (0,1) = L}

rule : {Gx} 100 { (0,0) = L and (0,-1) = Cx and ((0,1) = Gx or (0,1) = S)}

rule : {Cx} 100 { (0,0) = L and (0,-1) = Gx and (0,1) = L}

rule : {Ax} 100 { (0,0) = L and (0,-1) = Gx and ((0,1) = Gx or (0,1) = S)}

rule : {D} 100 { (0,0) = L and (0,-1) = L and (0,1) = D}

rule : {D} 100 { (0,0) = L and (0,-1) = S and (0,1) = D}

rule : {D} 100 { (0,0) = L and (0,-1) = S and (1,0) = D}

rule : {Bx} 100 { (0,0) = Gx and ((0,-1) = Ax or (0,-1) = Bx or (0,-1) = Gx) and (0,1) = L}

rule : {Ax} 100 { (0,0) = Gx and (0,-1) = Cx and ((0,1) = Gx or (0,1) = L or (0,1) = S)}

rule : {Ax} 100 { (0,0) = Gx and (0,-1) = S and (0,1) = L}

rule : {Gy} 100 { (0,0) = Gx and ((0,-1) = Gx or (0,-1) = S) and ((0,1) = Gx or (0,1) = S)}

rule : {Bx} 100 { (0,0) = Ax and (0,-1) = Ax and ((0,1) = Bx or (0,1) = Gx)}

rule : {Cx} 100 { (0,0) = Ax and (0,-1) = Bx and ((0,1) = Cx or (0,1) = Gx or (0,1) = S)}

rule : {Cx} 100 { (0,0) = Ax and (0,-1) = Ax and (0,1) = Cx}

rule : {Cx} 100 { (0,0) = Ax and (0,-1) = Gx and ((0,1) = Cx or (0,1) = Gx or (0,1) = S)}

rule : {Gx} 100 { (0,0) = Ax and (0,-1) = Bx and ((0,1) = Bx or (0,1) = L)}

rule : {Gx} 100 { (0,0) = Ax and ((0,-1) = S or (0,-1) = L) and (0,1) = Cx}

rule : {L} 100 { (0,0) = Ax and (0,-1) = L and (0,1) = Bx}

rule : {F} 100 { (0,0) = Ax and (0,-1) = Ax and (0,1) = S}

rule : {F} 100 { (0,0) = Ax and (0,-1) = S and (0,1) = Ax}

rule : {L} 100 { (0,0) = Bx and ((0,-1) = Ax or (0,-1) = L) and (0,1) = Cx}

rule : {L} 100 { (0,0) = Bx and (0,-1) = Cx and ((0,1) = Gx or (0,1) = L or (0,1) = S)}

rule : {Ax} 100 { (0,0) = Bx and ((0,-1) = Bx or (0,-1) = Cx) and (0,1) = Ax}

rule : {Cx} 100 { (0,0) = Bx and (0,-1) = Bx and (0,1) = Cx}

rule : {Cx} 100 { (0,0) = Bx and (0,-1) = Gx and ((0,1) = Ax or (0,1) = L)}

rule : {Gx} 100 { (0,0) = Bx and (0,-1) = L and (0,1) = Ax}

rule : {Gx} 100 { (0,0) = Bx and (0,-1) = Gx and ((0,1) = Gx or (0,1) = S)}

rule : {Gx} 100 { (0,0) = Bx and ((0,-1) = Ax or (0,-1) = Bx) and (0,1) = L}

rule : {Ax} 100 { (0,0) = Cx and ((0,-1) = L or (0,-1) = Cx) and (0,1) = Ax}

rule : {Gx} 100 { (0,0) = Cx and (0,-1) = Bx and ((0,1) = Gx or (0,1) = S)}

rule : {Gx} 100 { (0,0) = Cx and (0,-1) = L and ((0,1) = Bx or (0,1) = Gx)}

rule : {Bx} 100 { (0,0) = Cx and (0,-1) = Ax and ((0,1) = Bx or (0,1) = Gx or (0,1) = L or (0,1) = S)}

rule : {Bx} 100 { (0,0) = Cx and (0,-1) = Gx and ((0,1) = Bx or (0,1) = Gx or (0,1) = L or (0,1) = S)}

rule : {Bx} 100 { (0,0) = Cx and (0,-1) = Cx and ((0,1) = Bx or (0,1) = Gx)}

rule : {Cy} 100 { (0,0) = L and (-1,0) = Ay and ((1,0) = Gy or (1,0) = S)}

rule : {Gy} 100 { (0,0) = L and (-1,0) = Ay and (1,0) = L}

rule : {Ay} 100 { (0,0) = L and (-1,0) = Cy and (1,0) = L}

rule : {Gy} 100 { (0,0) = L and (-1,0) = Cy and ((1,0) = Gy or (1,0) = S)}

rule : {Cy} 100 { (0,0) = L and (-1,0) = Gy and (1,0) = L}

rule : {Ay} 100 { (0,0) = L and (-1,0) = Gy and ((1,0) = Gy or (1,0) = S)}

rule : {D} 100 { (0,0) = L and (-1,0) = L and (1,0) = D}

rule : {D} 100 { (0,0) = L and (-1,0) = S and (1,0) = D}

rule : {By} 100 { (0,0) = Gy and ((-1,0) = Ay or (-1,0) = By or (-1,0) = Gy) and (1,0) = L}

rule : {Ay} 100 { (0,0) = Gy and (-1,0) = Cy and ((1,0) = Gy or (1,0) = L or (1,0) = S)}

rule : {Ay} 100 { (0,0) = Gy and (-1,0) = S and (1,0) = L}

rule : {F} 100 { (0,0) = Gy and ((-1,0) = Gy or (-1,0) = S) and ((1,0) = Gy or (1,0) = S)}

rule : {By} 100 { (0,0) = Ay and (-1,0) = Ay and ((1,0) = By or (1,0) = Gy)}

rule : {Cy} 100 { (0,0) = Ay and (-1,0) = By and ((1,0) = Cy or (1,0) = Gy or (1,0) = S)}

rule : {Cy} 100 { (0,0) = Ay and (-1,0) = Ay and (1,0) = Cy}

rule : {Cy} 100 { (0,0) = Ay and (-1,0) = Gy and ((1,0) = Cy or (1,0) = Gy or (1,0) = S)}

rule : {Gy} 100 { (0,0) = Ay and (-1,0) = By and ((1,0) = By or (1,0) = L)}

rule : {Gy} 100 { (0,0) = Ay and ((-1,0) = S or (-1,0) = L) and (1,0) = Cy}

rule : {L} 100 { (0,0) = Ay and (-1,0) = L and (1,0) = By}

rule : {F} 100 { (0,0) = Ay and (-1,0) = Ay and (1,0) = S}

rule : {F} 100 { (0,0) = Ay and (-1,0) = S and (1,0) = Ay}

rule : {L} 100 { (0,0) = By and ((-1,0) = Ay or (-1,0) = L) and (1,0) = Cy}

rule : {L} 100 { (0,0) = By and (-1,0) = Cy and ((1,0) = Gy or (1,0) = L or (1,0) = S)}

rule : {Ay} 100 { (0,0) = By and ((-1,0) = By or (-1,0) = Cy) and (1,0) = Ay}

rule : {Cy} 100 { (0,0) = By and (-1,0) = By and (1,0) = Cy}

rule : {Cy} 100 { (0,0) = By and (-1,0) = Gy and ((1,0) = Ay or (1,0) = L)}

rule : {Gy} 100 { (0,0) = By and (-1,0) = L and (1,0) = Ay}

rule : {Gy} 100 { (0,0) = By and (-1,0) = Gy and ((1,0) = Gy or (1,0) = S)}

rule : {Gy} 100 { (0,0) = By and ((-1,0) = Ay or (-1,0) = By) and (1,0) = L}

rule : {Ay} 100 { (0,0) = Cy and ((-1,0) = L or (-1,0) = Cy) and (1,0) = Ay}

rule : {Gy} 100 { (0,0) = Cy and (-1,0) = By and ((1,0) = Gy or (1,0) = S)}

rule : {Gy} 100 { (0,0) = Cy and (-1,0) = L and ((1,0) = By or (1,0) = Gy)}

rule : {By} 100 { (0,0) = Cy and (-1,0) = Ay and ((1,0) = By or (1,0) = Gy or (1,0) = L or (1,0) = S)}

rule : {By} 100 { (0,0) = Cy and (-1,0) = Gy and ((1,0) = By or (1,0) = Gy or (1,0) = L or (1,0) = S)}

rule : {By} 100 { (0,0) = Cy and (-1,0) = Cy and ((1,0) = By or (1,0) = Gy)}

rule : {(0,0)} 100 { t }

# Part III – Implementation:

This section will cover the challenges and additional decisions made during implementation, as well as the test findings for the 1D and 2D solutions. The optimal time NxN solution was not implemented as it requires more then 400 rules and I was satisfied with my simplified 2D solution.

## 1D Implementation

The previously described model was implemented in CD++. There are two missing state transition in the rules layed out in figure one: if the current cell is G and its left neighbor is C and right neighbor is L or S it should become A. These issues were identified and fixed while implementing resulting in a successful 1D FSSP algorithm with any cell length and any starting position. The algorithm requires that each side of the cell space is marked with a special cell (often called S or \*), and one cell starts in the state D. The D cell will shift to the left most cell, then become the first general and use Mazoyer’s 6 state minimum time algorithm to synchronize the cells.

The variable mapping and color palet are shown in the following table:

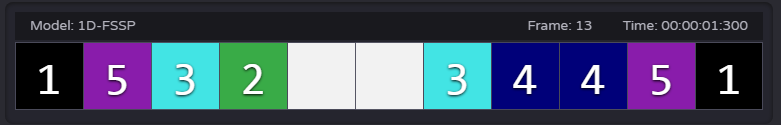
|  |  |  |  |
| --- | --- | --- | --- |
| State | Numeric Value | RGB Value | Coresponding color |
| L | 0 | 255 255 255 |  |
| S | 1 | 0 0 0 |  |
| A | 2 | 260 180 75 |  |
| B | 3 | 70 240 240 |  |
| C | 4 | 0 0 128 |  |
| G | 5 | 145 30 180 |  |
| F | 6 | 230 25 75 |  |
| D | 7 | 170 110 40 |  |

## 1D Testing

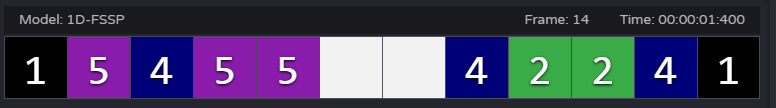
Without the amendemnts mentioned in the introduction to this section the algorithm would only work if the first state taken on by the right most cell was a C. For example, 12 cell width would work but both 11 and 13 would cases fail. I included a video file of the original algorithm failing titeled “1D-FSSP-WithoutTwoAdditionalRules”.

There is sliglty different behavior depending on which state the first signal is in when it arrives at the right most barrier. The following set of figures shows the frame where the first signal meets the right most side for various cell spaces. Notice three different cases, either the first state the right most cell takes on is G, C, or A. I included one more test to show that it cycles back to the first case.

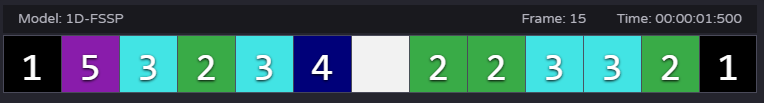
11 Cell Width:



12 Cell Width:



13 Cell Width:



14 Cell Width:

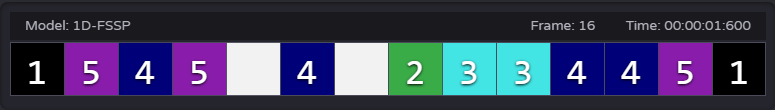


Figure 3: The frame where the first signal hits the end of varying cell widths.

I included a complete video of these test cases titled “1D-FSSP-SizeX” I also tested the lower bounds to ensure it worked correctly in small cell spaces. To better visualize the algorithm many papers displayed the 1D solution’s progression over time. Each time advance was shown on a new row. This was implemented in in 1DxTime-FSSP and the output is shown in Figure 4, below.

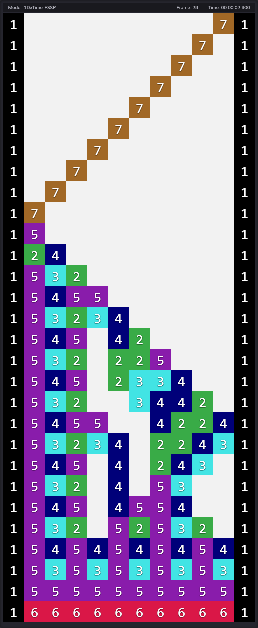


Figure 4: 1D-FSSP shown across time.

## 2D Implementation

Next, I implemented the proposed 2D solution, with a slight difference: I had specific states for signals that were traveling in the x direction and in the y direction. These extra four states were required since the problem limits you to the von Neuman neighbor hood of the cells. If the cells could look at the diagonals, then they could identify when the entire top row is generals and start the vertical synchronization. The solution I implemented is not minimum time nor minimum state; however, it has the simplest rule set out of any 2D FSSP solution I could find online. This was optimal for me as I was limited by the time available for assignment 2. Also, worth noting many of the solutions I found online had limitations on the starting cell and the shape of the matrix (either they only work for NxN or NxM where N >M); where as, this solution will work for any M x N matrix, with any starting cell.

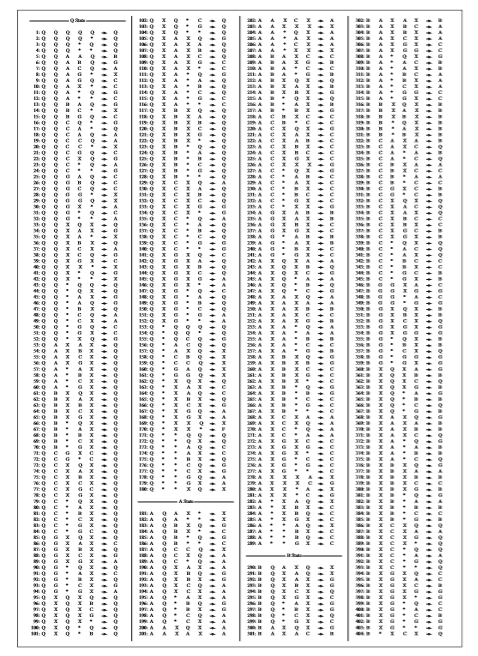
The variable mapping and color palet are shown in the following table:

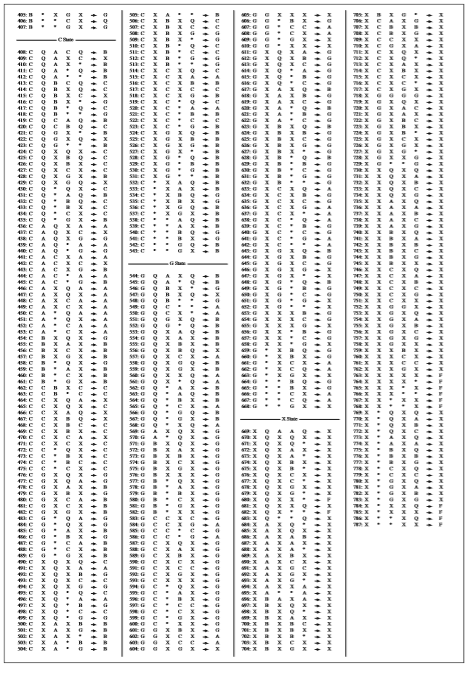
|  |  |  |  |
| --- | --- | --- | --- |
| State | Numeric Value | RGB Value | Coresponding color |
| L | 0 | 255 255 255 |  |
| S | 1 | 0 0 0 |  |
| Ax | 2 | 260 180 75 |  |
| Bx | 3 | 70 240 240 |  |
| Cx | 4 | 0 0 128 |  |
| Gx | 5 | 145 30 180 |  |
| F | 6 | 230 25 75 |  |
| D | 7 | 170 110 40 |  |
| Ay | 8 | 260 180 75 |  |
| By | 9 | 70 240 240 |  |
| Cy | 10 | 0 0 128 |  |
| Gy | 11 | 145 30 180 |  |

## 2D Testing

There are four mp4 examples included for the 2D-FSSP algorithm. The first, ex1, shows the synchronization of a NxM matrix where N > M. The second, ex2, shows the synchronization of a NxM matrix where N < M. The third, ex3, is of a NxN matrix with the starting square as far from the corner as possible. The final, ex4 is of a 1D cell space using the 2D algorithm. The splitting of the D signal while finding the top right corner was not intentional, however I found no case where it would cause any issues. For this reason, I left it as is, as I think it looks more interesting to watch them converge in the top corner. It is less efficient then only having one of the signals and could cause a significant waste in resources if the cell space were large.

# Appendix A – Complete State Table for Optimum Time 2D Square Array





# References

* Mazoyer, J. (1987). A six-state minimal time solution to the firing squad synchronization problem. *Theoretical Computer Science*, 50(2), pp.183-238.
* Settle, A. and Simon, J. (2002). Smaller solutions for the firing squad. *Theoretical Computer Science*, 276(1-2), pp.83-109.
* Umeo H. and Kubo K. (2010) A Seven-State Time-Optimum Square Synchronizer. In: Bandini S., Manzoni S., Umeo H., Vizzari G. (eds) Cellular Automata. ACRI 2010. *Lecture Notes in Computer Science*, vol 6350. Springer, Berlin, Heidelberg