**Additional Materials**

The Construction of Asynchronous Computational Devices with Chemical Reaction Networks

Luca Cardelli, Marta Kwiatkowska and Max Whitby

Contents:

1. Reproducibility of Results …….…………..2
2. Component Code ………………..…………..6
3. Experiments upon Components ……….12
4. Muller Pipeline – Code and Testing…..25
5. Other large systems …………………………34

**Reproducability of Results**

All GEC and PRISM files as well as properties can be found in the following repository:

<https://github.com/max1s/CRNcode>

For all components and systems we provide the code later in the additional materials for reference.

**Microsoft GEC**

Microsoft GEC tool is free to use and can be found online <http://biology.azurewebsites.net/gec/beta/> (21/03/2016).

To use the tool click on the GEC tab and copy and paste one of the CRNs found in the next section. Click “simulate” to simulate the kinetics of the system. To change the inputs to the system change the molecular count of one of the input species. For instance to change the presence of yhi to ylo change the line:

Yhi 10.0

Ylo 0.0

To:

Yhi 0.0

Ylo 10.0

This means that we have 10 molecules of species ylo now instead of 10 molecules of yhi. To change which species are plotted change the parameter “plot”. For instance if we wished to track yhi,xhi then we would add the line:

plot yhi;xhi

To the second line of our program. Similarly we can change the type of simulation by changing the directive “simulation” to deterministic, stochastic and (on some future date) LNA. The top line refers to the number of samples so:

Directive sample 5.0 1000

Means 1000 samples over 5.0 seconds.

For more information on how to use the GEC tool please refer to the operating manual:

<http://research.microsoft.com/en-us/projects/gec/manual.pdf> (21/03/2016)

Note: In a new patch to GEC a user may need to change the rate constant to a parameter. This is done by replacing the line “rate rt = 1.0;” to “parameters [rt = 1.0];”.

**PRISM**

PRISM is a free model checker and can be found at the following website:

<http://www.prismmodelchecker.org/>

To export a GEC file to PRISM first click export -> PRISM in the GEC GUI.

Unfortunately the parser is broken (21/03/2016). We provide a fix to the parser below which is a python file:

**import** sys

**import** re

inputfile = **open**(sys.argv[1])

outputfile = **open**(inputfile.name + "fixed", 'w')

**for** line **in** inputfile:

**if** re.search("\[(.)\*\]\strue", line):

match = re.search("\(.\*\)", line)

newline = line.replace("true", match.group(0) + " > 0", 1)

outputfile.write(newline)

**else**:

outputfile.write(line)

**print** "done"

This file can be downloaded directly from the repository. This file takes in the PRISM export file and produces a new file called \_\_filename\_\_fixed. With this file we can use this in the main PRISM window. Opening a PRISM file in this window might look something like this:

The overall structure of the file is as follows:

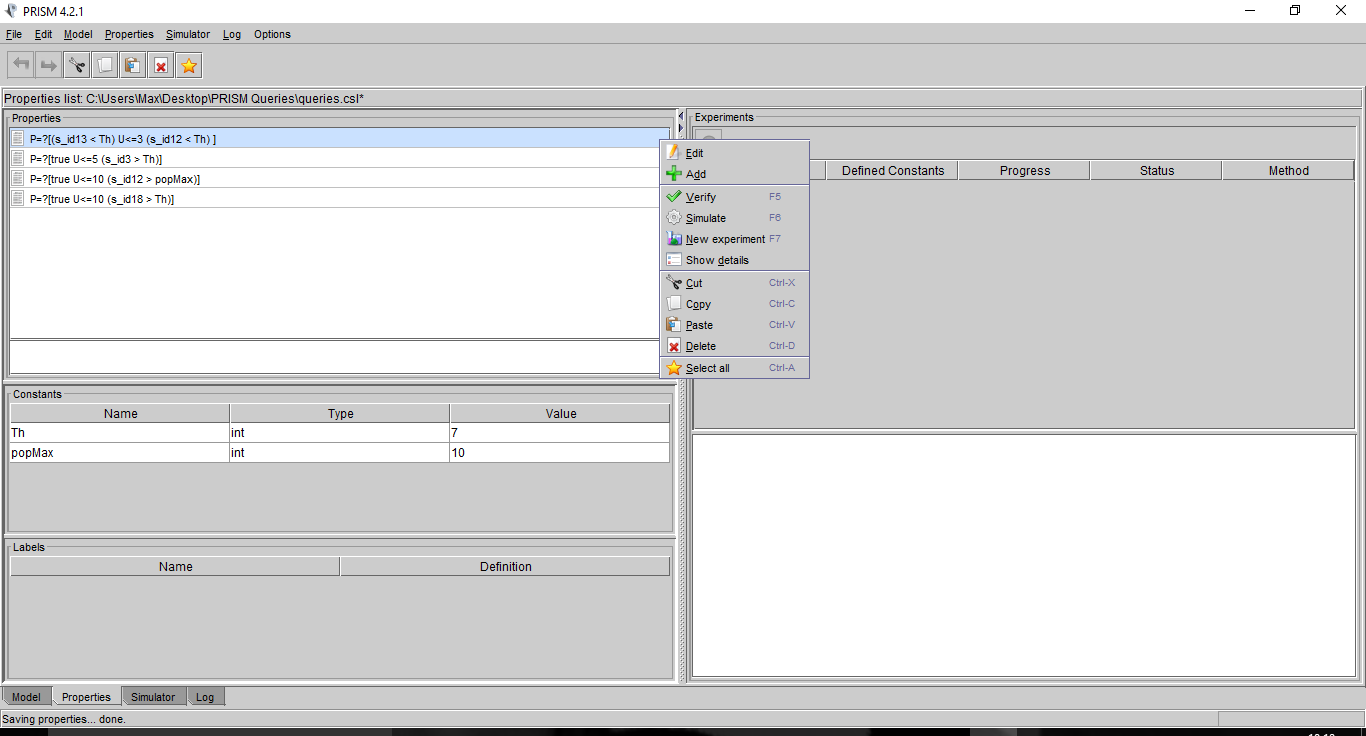
**Rate constants** (e.g. const double k\_r\_id44 = 1.0;) – each reaction has an individual rate constant attached. All of our models operate under the same rate constants.

**Modules –** Each chemical species has a module which determines its interaction with other modules/species. Within these modules we can change the initial molecule count for a species for instance we can change the line: *s\_id4 : [s\_id4\_MIN..s\_id4\_MAX] init 0* to: *s\_id4 : [s\_id4\_MIN..s\_id4\_MAX] init 10* which would mean that species s\_id4 would have a starting condition of 10 molecules as apposed to 0.

Once we have our file in PRISM we can check properties of this file by accessing the properties tab seen in the bottom left of the window. Properties allow us to to query our model.

**Querying the PRISM model**

We can create queries in the properties tab in PRISM seen below:

In this tab we can write properties to query our model in CSL. More information on CSL can be found in the following paper: Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. 2000. Model-checking continuous-time Markov chains. ACM Trans. Comput. Logic 1, 1 (July 2000), 162-170. DOI=http://dx.doi.org/10.1145/343369.343402

A typical query might look like:

P=? [ (true) U<=10 (s\_id3>Th) ] – this means predicate until some other predicate.

Note that we have something called “s\_id3”. This is because the conversion process to PRISM renames species according to their ordering. To use the correct molecular species refer back to the model tab. Find the molecule you want to query in the rewards, for instance ahi:

// Species s\_id12 (ahi)

rewards "s\_id12" true : s\_id12; endrewards

and find the corresponding ID number.

Once you have your query right click and press Simulate to set up a simulation for your query. On the smaller files we can take 1000s of samples although on the larger ones this number is reduced to hundreds. A large amount of querying also depends on initial conditions. This can be changed directly in the model tab by changing the species count for an individual species.

For more information on the operation of PRISM:

<http://www.prismmodelchecker.org/manual/>

**Visual DSD**

Microsoft Visual DSD is a free tool found at: <http://boson.research.microsoft.com/webdna/> (21/03/2016).For more information please refer to: <http://research.microsoft.com/en-us/projects/dna/manual.pdf> (21/03/2016).

We converted our models to DSD by a 2-domain implementation using the catalytic gate described in Cardelli’s paper:

<http://journals.cambridge.org/action/displayAbstract?fromPage=online&aid=8851390&fileId=S0960129512000102>

While this was not a large part of this project we have provided the necessary details below in order for the reader to confirm that this implementation would indeed work in DNA.

Our 2-Domain Catalytic gate translates to the following code in DSD:

directive duration 5000.0 points 1000

directive scale 1000.0

def bind = 0.0003 (\* /nM/s \*)

def unbind = 0.1126 (\* /s \*)

new tx@bind,unbind

new ty@bind,unbind

//Catalytic Gate Cardelli 2-Domain Strand Displacement

def C(N,x,y,z) =

new a new c

( N \* {t^\*}[x t^]:[y t^]:[c]:[a t^]:[a] |

N \* [x]:[t^ z]:[c]:[t^ y]:[t^ a]{t^\*}

| N \* <t^ c a>

| N \* <z v t^>)

Because our motif is two catalytic reactions we can compose two of these gates to form our motif.Here is an example of how to use this code with our and-gate design for instance:

def AND(N, xlo, xhi, ylo, yhi) =

(

    N \* <zTwo t^ zTwo>

  | N \* <zOne t^ zOne>

  | CN, zTwo, zThree, zlo, yhi)

  | C(N, zThree, zhi, zlo, xhi)

  | C(N, zlo, zOne, xlo, zhi)

  | C(N, zlo, zOne, ylo, zhi)

)

(

                  5 \* <yhi t^ yhi>

                | 5 \* <xhi t^ xhi>

                | AND(5, xlo, xhi, ylo, yhi)

)

As we can see the program is primarily a list of of catalytic reactions.

**Asynchronous Component Design**For further information on the designs used within this paper we strongly recommend Furber’s book: Principles of Asynchronous Circuit Design. It provides great insight into the uses of the C-pipeline and control flow elements.

**Components Code**

Here we present code for components listed in Figure 4 in the paper. This code is directly copy and pastable into visual DSD. We comment what the inputs and outputs are to make it easier to replicate experiments. Refer to the plot command to plot specific graphs.

Presented in the following order:

C-element

Latches

Logic gates (AND, OR, XOR and a brief explanation of how to assemble the others from this code).

Control flow elements. – experiments for fork join and biased arbiters.

|  |  |
| --- | --- |
| Code Reference | Code |
| Muller C-Element  Inputs:  Xlo, xhi  Ylo,yhi  Outputs:  Zlo, zhi | directive sample 5.0 100  directive simulation deterministic  directive plot zhi;zmid;zlo    rate rt = 1.0;    //internals  init zup 0.0  | init znt 10.0  | init zdn 0.0  //OUTPUTS  | init zhi 0.0  | init zmid 10.0  | init zlo 0.0    //INPUTS  | init xhi 10.0  | init xlo 0.0    | init yhi 0.0  | init ylo 10.0    | zdn + zup ->{rt} znt + znt  | znt + zdn ->{rt} zdn + zdn  | znt + zup ->{rt} zup + zup    | zdn + xhi ->{rt} znt + xhi  | znt + yhi ->{rt} zup + yhi  | zup + xlo ->{rt} znt + xlo  | znt + ylo ->{rt} zdn + ylo    | zlo + zhi ->{rt} zmid + zmid  | zmid + zlo ->{rt} zlo + zlo  | zmid + zhi ->{rt} zhi + zhi    | zdn + zhi ->{rt} zdn + zmid  | zdn + zmid ->{rt} zdn + zlo  | zup + zlo ->{rt} zup + zmid  | zup + zmid ->{rt} zup + zhi |
| Change in inputs –if we want to change the input values during the simulation we can add lines such as this one. | | xhi ->{rt} xlo (etc.)  … |
| AND  Input: ylo, yhi, xlo, xhi  Outputs: zhi,zlo | AND    directive sample 10.0 100    rate rt = 1.0;    //inputs  init xlo 10.0  | init xhi 0.0  | init ylo 1.0  | init yhi 0.0    //internal chemical species  | init ze 0.0  | init zw 10.0  | init zq 10.0    //outputs  | init zlo 0.0  | init zhi 0.0    //reactions    |xhi + ze ->{rt} xhi + zhi  |yhi + zw ->{rt} yhi + ze    |xlo + zq ->{rt} xlo + zlo  |ylo + zq ->{rt} ylo + zlo    //for persistance    |zhi + xlo ->{rt} ze + xlo  |zhi + ylo ->{rt} zw + ylo    |zlo + zhi ->{rt} zhi + zq |
| OR  Input: ylo, yhi, xlo, xhi  Outputs: zhi,zlo | directive sample 10.0 100  directive simulation deterministic      rate rt = 1.0;    //inputs  init xlo 10.0  | init xhi 0.0  | init ylo 0.0  | init yhi 10.0    //internal chemical species  | init ze 10.0  | init zw 10.0  | init zq 0.0    //outputs  | init zlo 0.0  | init zhi 0.0    //reactions  | xhi + ze ->{rt} xhi + zhi  | yhi + ze ->{rt} yhi + zhi    | xlo + zw ->{rt} xlo + zq  | ylo + zq ->{rt} ylo + zlo      //for persistance    |zlo + xhi ->{rt} zw + xhi  |zlo + yhi ->{rt} zq + yhi    |zlo + zhi ->{rt} zlo + ze |
| XOR  Input: ylo, yhi, xlo, xhi  Outputs: zhi,zlo | directive sample 10.0 100  directive simulation deterministic    rate rt = 1.0;    //inputs  init xlo 10.0  | init xhi 0.0  | init ylo 0.0  | init yhi 10.0    //internal chemical species  | init zq 0.0  | init zw 10.0  | init ze 10.0  | init zr 0.0  //outputs  | init zlo 0.0  | init zhi 0.0    //Reactions  | ylo + zw ->{rt} ylo + zq  | xlo + zw ->{rt} xlo + zq  | xhi + zq ->{rt} xhi + zhi  | yhi + zq ->{rt} yhi + zhi    |xhi + ze ->{rt} xhi + zr  |ylo + ze ->{rt} ylo + zr  |yhi + zr ->{rt} yhi + zlo  |xlo + zr ->{rt} xlo + zlo    //for persistance    |zhi + zlo ->{rt} zhi + zr  |zhi + zr ->{rt} zhi +ze  | zlo + zhi ->{rt} zlo + zq  | zlo + zq ->{rt} zlo + zw |
| 2-State latch  Input: xlo, xhi  Outputs: ylo, yhi | directive sample 2.0 100  directive simulation deterministic    rate rt = 1.0;     //outputs  init ylo 10.0  | init yhi 0.0    //INPUTS  | init xhi 10.0  | init xlo 0.0      | yhi + ylo ->{rt} yhi + yhi  | ylo + yhi ->{rt} ylo + ylo    | xhi + ylo ->{rt} xhi + yhi  | xlo + yhi ->{rt} xlo + ylo |
| 3-State latch  Input: xlo, xhi, rhi  Outputs: ylo, yhi | directive sample 10.0 100  directive simulation deterministic  directive plot yhi;ylo;ymid;xhi    rate rt = 1.0;    //internals  init ymid 10.0  //outputs  | init ylo 0.0  | init yhi 0.0    //INPUTS  | init xhi 10.0  | init xlo 0.0  | init rhi 0.0  //just for testing  | init rlo 10.0    | rlo ->{rt} rhi  | xhi ->{rt}    | yhi + ymid ->{rt} yhi + yhi  | ylo + ymid ->{rt} ylo + ylo    | xhi + ylo ->{rt} xhi + ymid  | xhi + ymid ->{rt} xhi + yhi  | xlo + yhi ->{rt} xlo + ymid  | xlo + ymid ->{rt} xlo + ylo    | rhi + yhi ->{rt} rhi + ymid  | rhi + ylo ->{rt} rhi + ymid |
| 2.7 - 3-state latch with additional control input  Used in the queue. Input output the same with addition of control chemical as input | directive sample 10.0 100  directive simulation deterministic    rate rt = 1.0;     //internals  init ymid 10.0  | init ylo 0.0  | init yhi 0.0  | init sOne 10.0  | init sTwo 0.0  | init sThree 10.0  | init sFour 0.0  | init slo 0.0    //INPUTS  | init xhi 0.0  | init xlo 0.0  | init rhi 0.0  | init rlo 0.0    | init chi 0.0  | init clo 0.0    //OUTPUTS  | init shi 0.0  | init slo 0.0    | yhi + ymid ->{rt} yhi + yhi  | ylo + ymid ->{rt} ylo + ylo    | xhi + ylo ->{rt} xhi + ymid  | xhi + ymid ->{rt} xhi + yhi  | xlo + yhi ->{rt} xlo + ymid  | xlo + ymid ->{rt} xlo + ylo    | rhi + yhi ->{rt} rlo + ymid  | rhi + ylo ->{rt} rlo + ymid    | yhi + sOne ->{rt} yhi + sTwo  | chi + sTwo ->{rt} chi + shi  | clo + shi ->{rt} clo + sTwo  | ymid + sTwo ->{rt} ymid + sOne  | ylo + sTwo ->{rt} ylo + sOne  | ymid + shi ->{rt} ymid + sOne  | ylo + shi ->{rt} ylo + sOne    | ylo + sThree ->{rt} ylo + sFour  | chi + sFour ->{rt} chi + slo  | clo + slo ->{rt} clo + sFour  | ymid + slo ->{rt} ymid + sFour  | yhi + slo ->{rt} yhi + sFour  | ymid + sFour ->{rt} ymid + sThree  | yhi + sFour ->{rt} ymid + sThree |
| 2.8 - Biased Arbiter  Comment: Section written EITHER OR, uncomment one line to bias the arbiter either way. | directive sample 10.0 100    rate rt = 1.0;    //INPUTS  init xlo 0.0  | init xhi 5.0    //OUTPUTS  | init yhi 0.0  | init ylo 5.0    //REACTIONS    | xhi + ylo ->{rt} xhi + yhi  | xlo + yhi ->{rt} xlo + ylo  //EITHER OR  //| yhi + ylo ->{rt} yhi + yhi  //| ylo + yhi ->{rt} ylo + ylo |
| 2.9 - Fork then Join  An experiment to show the fork and join at the same time. | directive sample 10.0 100    rate rt = 1.0;    //INPUTS  init xstart 10.0  |init pathOne 0.0  |init pathTwo 0.0  |init lambdaOne 5.0  |init lambdaTwo 0.0  |init xfinish 0.0    //REACTIONS    | xhi ->{rt} pathOne  | xhi ->{rt} pathTwo  | pathOne + lambdaOne ->{rt} pathOne + lambdaTwo  | pathTwo + lambdaTwo ->{rt} pathTwo + xfinish |

**Example Testing of Components**

We show simulations, using visual-GEC, of our components under mass-action kinetics. Any deviation from this (LNA plot) is explicitly specified. The ordering of this testing is as before:

C-element

Latches

Logic gates (AND, OR, XOR and a brief explanation of how to assemble the others from this code).

Control flow elements. – experiments for fork join and biased arbiters.

A brief note on starting condition notation:

A change in input is noted through an arrow. For instance: xhi (yhi -> ylo) means that we have the initial condition xhi, yhi but throughout the runtime of the experiment the species yhi is converted to ylo.

Our first experimentation is with the C-Element. We test over most inputs and display the output signals with a change in input. The C-element, like a latch, it is a gate which retains a state. A C-element has two inputs and one output. When both inputs are low the output is low. Similarly, when both inputs are high the output is high. The variation from a normal gate however, is, if the inputs are high, or low, and one of them changes, it `remembers' the last high, or low, state. In other words it retains the last 0 or 1 state. We demonstrate this property through the following simulations on our code stated previously.

|  |  |  |
| --- | --- | --- |
| **No.** | **Muller C- Element** | |
| 1 | Input: Xhi yhi  output signal: zhi observed from low to high in red responding to the double “high” input. |  |
| **2** | Input: Xlo Ylo  output signal: zlo observed in blue responding to both x and y input being low. | Machine generated alternative text: |
| **3** | Input: xhi (yhi -> ylo)  output signal: zhi observed in red remains high despite y input changing |  |
| **4** | Input: yhi (xhi -> xlo)  output signal: zhi observed in red remains high despite x input changing. |  |
| **5** | Input: xlo (ylo -> yhi)  output signal: zlo observed in blue. Unchanging despite change in y input. | Machine generated alternative text: |
| **6** | Input: (ylo -> yhi) (xlo -> xhi)  When both inputs change our output changes seen in red. |  |
| **7** | LNA plot  Input: xhi (ylo -> yhi)  With a change in input y seen in purple we still have output signal zhi seen in red. This plot shows variance with the shaded regions. | C:\Users\Max\AppData\Local\Temp\msohtmlclip1\04\clip_image002.png |

A latch is a device, used in electronics to store a logical 0 or 1. A latch needs to have at least two stable states which are cycled between. In this first experiment we show the latch switching between the two values based upon input.In the second we change the input and the latch responds accordingly.

|  |  |  |
| --- | --- | --- |
| **Latch Simple** | | |
| **1** | Input: xhi  Output: yhi seen in green responding to xhi input signal. | Machine generated alternative text: |
| **2** | xhi -> xlo  output signal ylo seen in red. Notice how at first yhi is rising in molecular count (seen in green) and then falls away. | Machine generated alternative text: |

In our complex latch design we have an intermediary state ymid which the latch can be reset to. In the second experiment we see this latch reset to this intermediary state.

|  |  |  |
| --- | --- | --- |
| **Latch Complex** | | |
| **1** | Input: xhi  Output: yhi seen in red. As we have an input of x as high the latch responds and outputs a value of yhi. | Machine generated alternative text: |
| **2** | Input:  rlo -> rhi  xhi -> 0  As the reset species rhi is introduced our latch resets to a value of ymid seen in blue. | Machine generated alternative text: |

The next experiments are with our latch with an additional control input which means the value of the latch is not displayed until that control species is present. In the first experiment we have an input value but without the control signal. The output signal isn’t present due to the absence of control species. In the second we introduce the presence of the control species and we note that the latch species shi is present representing a ‘high’ signal.

|  |  |  |
| --- | --- | --- |
| **Latch With Control Signal** | | |
|  | In this first experiment we show that even with latch input xhi we do not exhibit an output zhi without the control signal. Output species is flat line seen in purple. | Machine generated alternative text: |
| 2 | In this second we show that with the introduction of a control species the output species shi is now present seen in purple. | Machine generated alternative text: |

|  |  |  |
| --- | --- | --- |
| AND-gate | | |
| 1 | Input: Yhi (xhi -> xlo)  We originally see an increase in the molecular count of zhi seen in green only for this to diminish as one of the inputs goes to zero. The output zlo is seen in red. | Machine generated alternative text: |
| 2 | Inputs:  1/2 yhi  1/2 xhi  We have both of our inputs at half values. The gate still produces a binary result! In this case we see a rise in the species zlo. | Machine generated alternative text: |
| 3. | Input:  Xhi  Ylo  We see that the gate responds to this input by showing a rise in the molecular count of zlo, as would be expected from an AND gate. | Machine generated alternative text: |

The other logic gates can be tested, by simulation, in a similar manner. The only gate that is slightly different from the rest is the XOR gate so we demonstrate this below.

|  |  |  |
| --- | --- | --- |
| XOR | | |
| 1 | Input: yhi xlo  With an or Input we would expect a value of 1 produced from this gate. That value is represented by the presence of the species zhi seen in green. | Machine generated alternative text: |
| 2. | Input: yhi xhi  With both inputs high we would expect to see a value of 0 produced. This value is represented by the species zlo seen in red. | Machine generated alternative text: |
| 3. | Input: ylo xlo  With both inputs low we see a value of 0 produced. The species zlo is seen in red. | Machine generated alternative text: |
| 4. | yhi (xhi -> xlo)  With a change in input we see the initial signal zlo (as would be expected from the input yhi xhi seen in red). We then can see as we change to an input yhi xlo we see the presence of zhi seen in green. | Machine generated alternative text: ••iiiiiił•i• |

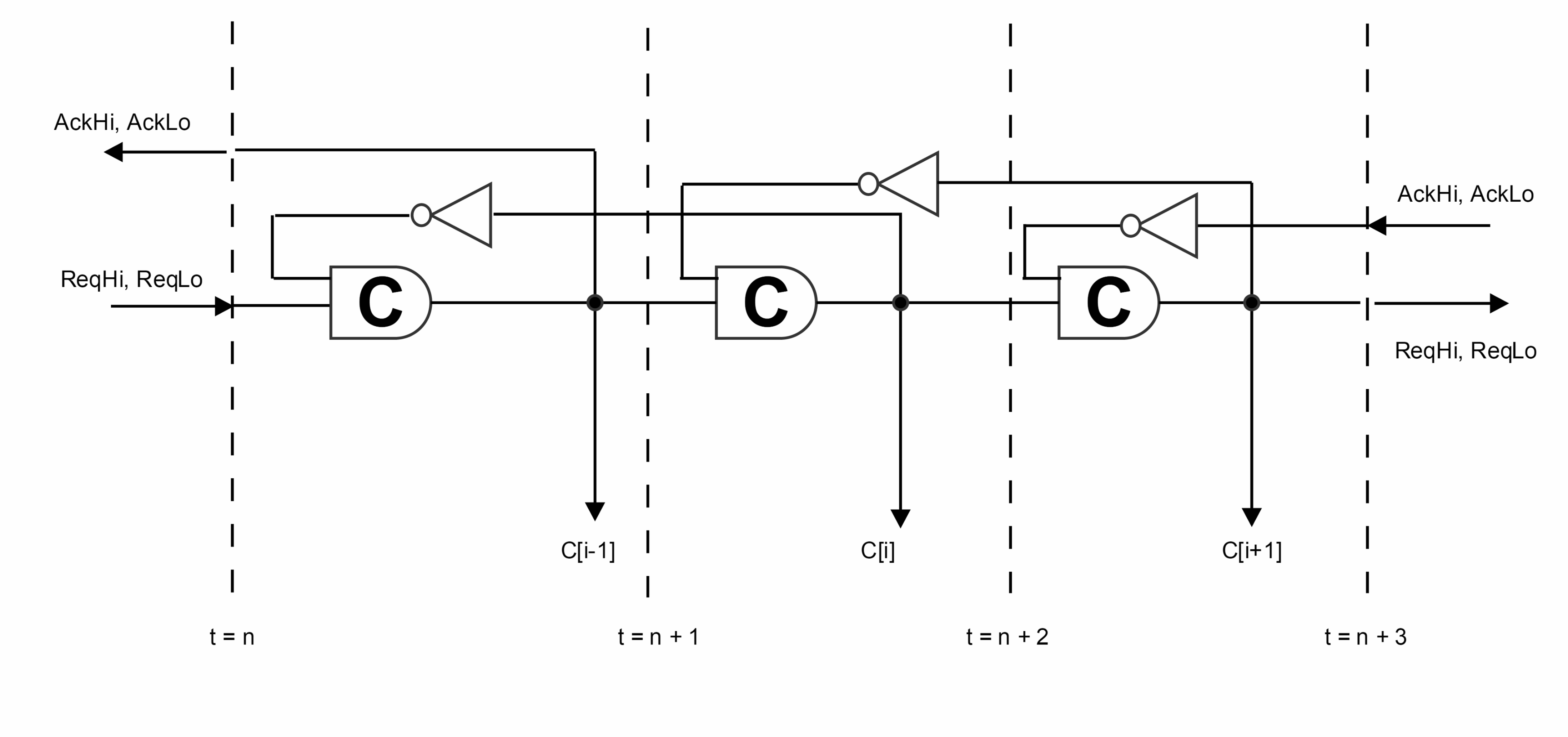
As well as these logic and control flow elements we can also introduce a biased arbiter which is a control flow element used to ‘break ties’ if both input species are present. We can see in our experiments that we can bias the arbiter one of two ways: either towards yhi or ylo.

|  |  |  |
| --- | --- | --- |
| Biased Arbiter | | |
| 1. | Both inputs present in equal concentrations. Bias towards yhi represented in blue. We can see ylo has a lower value seen in yellow. | Machine generated alternative text: |
| 2. | Both inputs present.   Bias towards  ylo seen in yellow. Yhi has a lower molecular count seen in blue. | Machine generated alternative text: |
|  |  |  |

**Pipeline and other Systems**

Now we have explored individual component design we now show these components within typical asynchronous constructions such as the Muller-C Pipeline. We present the code and simulations for these constructions in the following order:

1. Muller Pipeline
2. Queue
3. 4-Phase Pipeline
4. Adder

The muller C- Pipeline is responsible for propagating a control signal through a system and looks as follows in the diagram below. A C-element has two inputs and one output. These inputs can either be present or not present; low or high. When both inputs are low the output is low. Similarly, when both inputs are high the output is high. The variation from a normal gate however, is, if the inputs are high, or low, and one of them changes, it `remembers' the last high, or low, state. In other words it retains the last 0 or 1 state. C-elements allow a circuit to be speed independent by a series of local handshakes. This means that we can wait for longer computational paths to complete before advancing without additional computation occurring, negating the use of a system clock. This leads to the creation of what is called Muller pipelining. C-elements connected in series are referred to as a C-Pipeline which is shown bellow. In this simple example, data, request and acknowledge rails can be set to high or low. The four phase protocol, enacted upon the C-pipeline, is as follows: firstly the sender sends data and sets request to high. The receiver then writes the data to the register and sets acknowledge to high. Then the sender responds by setting request to low and finally the receiver acknowledges this by setting acknowledge to low.

Using the CRN components listed before we compose them in series in order to create the full pipeline given in the code below. The pipeline is used for temporal ordering such that one event can occur strictly before another.

**Notes on code:** The code is lumped into sections a,b,c,d which represent the individual c-elements. For instance the species “ahi” refers to a high signal exhibited by the first C-element. Similarly a species “bhi” is from the second. The inputs (request and acknowledge lines) are under the section //INPUT. Not gates are referred to as bNotUp and bNotDown with them starting in a neutral state bNotNt.

|  |  |
| --- | --- |
| Full Muller Pipeline | directive sample 20.0 100  directive plot ahi;bhi;chi  directive simulation deterministic  rate rt = 1.0;    // C ELEMENTS  //AM SWITCH 1  init aup 0.0  | init ant 0.0  | init adn 10.0    //AM SWITCH 2  | init bup 0.0  | init bnt 0.0  | init bdn 10.0    //AM SWITCH 3  | init cup 0.0  | init cnt 0.0  | init cdn 10.0    //AM SWITCH 4  | init dup 0.0  | init dnt 0.0  | init ddn 10.0    //AMP-AM 1  | init ahi 0.0  | init ab 10.0  | init alo 0.0    //AMP-AM 2  | init bhi 0.0  | init bb 10.0  | init blo 0.0    //AMP-AM 3  | init chi 0.0  | init cb 10.0  | init clo 0.0    //AMP-AM 4  | init dhi 0.0  | init db 10.0  | init dlo 0.0    //INPUTS  | init acchi 10.0  | init acclo 0.0    | init reqhi 10.0  | init reqlo 0.0      //NOT    | init bNotUp 0.0  | init bNotNt 10.0  | init bNotDown 0.0    | init cNotUp 0.0  | init cNotNt 10.0  | init cNotDown 0.0    | init dNotUp 0.0  | init dNotNt 10.0  | init dNotDown 0.0      //AREA 1  | adn + aup ->{rt} ant + ant  | ant + adn ->{rt} adn + adn  | ant + aup ->{rt} aup + aup    | adn + bNotUp ->{rt} ant + bNotUp  | ant + reqhi ->{rt} aup + reqhi  | aup + bNotDown->{rt} ant + bNotDown  | ant + reqlo ->{rt} adn + reqlo    //| alo + aup ->{rt} ahi + aup  //| ahi + adn ->{rt} alo + adn    | alo + ahi ->{rt} ab + ab  | ab + alo ->{rt} alo + alo  | ab + ahi ->{rt} ahi + ahi    | adn + ahi ->{rt} adn + ab  | adn + ab ->{rt} adn + alo  | aup + alo ->{rt} aup + ab  | aup + ab ->{rt} aup + ahi    //AREA 2  | bdn + bup ->{rt} bnt + bnt  | bnt + bdn ->{rt} bdn + bdn  | bnt + bup ->{rt} bup + bup    | bdn + cNotUp ->{rt} bnt + cNotUp  | bnt + ahi ->{rt} bup + ahi  | bup + cNotDown ->{rt} bnt + cNotDown  | bnt + alo ->{rt} bdn + alo    //| blo + bup ->{rt} bhi + bup  //| bhi + bdn ->{rt} blo + bdn    | blo + bhi ->{rt} bb + bb  | bb + blo ->{rt} blo + blo  | bb + bhi ->{rt} bhi + bhi    | bdn + bhi ->{rt} bdn + bb  | bdn + bb ->{rt} bdn + blo  | bup + blo ->{rt} bup + bb  | bup + bb ->{rt} bup + bhi      | bhi + bNotNt ->{rt} bhi + bNotDown  | bhi + bNotUp ->{rt} bhi + bNotDown    | blo + bNotNt ->{rt} blo + bNotUp  | blo + bNotDown ->{rt} blo + bNotUp    //AREA 3  | cdn + cup ->{rt} cnt + cnt  | cnt + cdn ->{rt} cdn + cdn  | cnt + cup ->{rt} cup + cup    | cdn + dNotUp ->{rt} cnt + dNotUp  | cnt + bhi ->{rt} cup + bhi  | cup + dNotDown ->{rt} cnt + dNotDown  | cnt + blo ->{rt} cdn + blo    //| clo + cup ->{rt} chi + cup  //| chi + cdn ->{rt} clo + cdn    | clo + chi ->{rt} cb + cb  | cb + clo ->{rt} clo + clo  | cb + chi ->{rt} chi + chi    | cdn + chi ->{rt} cdn + cb  | cdn + cb ->{rt} cdn + clo  | cup + clo ->{rt} cup + cb  | cup + cb ->{rt} cup + chi    | chi + cNotNt ->{rt} chi + cNotDown  | chi + cNotUp ->{rt} chi + cNotDown    | clo + cNotNt ->{rt} clo + cNotUp  | clo + cNotDown ->{rt} clo + cNotUp    //| chi + reqhi ->{rt} chi + reqlo  //| chi + acchi ->{rt} chi    //AREA 4  | ddn + dup ->{rt} dnt + dnt  | dnt + ddn ->{rt} ddn + ddn  | dnt + dup ->{rt} dup + dup    | ddn + acchi ->{rt} dnt + acchi  | dnt + chi ->{rt} dup + chi  | dup + acclo ->{rt} dnt + acclo  | dnt + clo ->{rt} ddn + clo    //| dlo + dup ->{rt} dhi + dup  //| dhi + ddn ->{rt} dlo + ddn    | dlo + dhi ->{rt} db + db  | db + dlo ->{rt} dlo + dlo  | db + dhi ->{rt} dhi + dhi    | ddn + dhi ->{rt} ddn + db  | ddn + db ->{rt} ddn + dlo  | dup + dlo ->{rt} dup + db  | dup + db ->{rt} dup + dhi    | dhi + dNotNt ->{rt} dhi + dNotDown  | dhi + dNotUp ->{rt} dhi + dNotDown    | dlo + dNotNt ->{rt} dlo + dNotUp  | dlo + dNotDown ->{rt} dlo + dNotUp |
| high to low request line – used in experimentation | …  | dhi + reqhi ->{rt} dhi + reqlo |
| high to low, low to high on cycle to force an oscillatory pattern from the pipeline. | …  | dhi + reqhi ->{rt} dhi + reqlo  | dhi + acchi ->{rt} dhi + acclo  | dlo + reqlo ->{rt} dlo + reqhi  | dlo + acclo ->{rt} dlo + acchi |

**Testing of the Pipeline – Simulation and Prism Queries**We test and observe certain properties of the pipeline confirming it operates in a way we would imagine which is separating the output species at each c-element along the pipeline. These output species are ahi,alo, bhi, blo etc.

|  |  |  |
| --- | --- | --- |
| Muller C Pipeline | | |
| 1 | An initial test where the request line is set to high. The species ahi,bhi,chi indicate  The output signal of the c elements at position a,b,c. We can observe that the signal is transmitted along the pipeline by noting that one after the other the species ahi,bhi,chi become present. This represents a logical '1' at each stage in the pipeline |  |
| 2 | In this experiment we then set the request signal to low at the input to the first C-element prompting the pipeline to respond by transmitting a low signal. These low signals are represented by the species alo, blo,clo. In these diagrams we can see that the signals ahi,bhi,chi diminish and are replaced by these new signals. We also present the LNA of this plot to represent variance. As we can see there is very little variance. | Machine generated alternative text: C:\Users\Max\AppData\Local\Temp\msohtmlclip1\02\clip_image002.png    C:\Users\Max\AppData\Local\Temp\msohtmlclip1\02\clip_image003.png |
| 3. | In this experiment we cycle between a low and high request signal and show that the pipeline responds with signal ahi,bhi,chi being present and then vanishing. We see here why the c-pipeline can be used over an oscillator as we can mimic the high/low cycle produced by an oscillator. | Machine generated alternative text: |

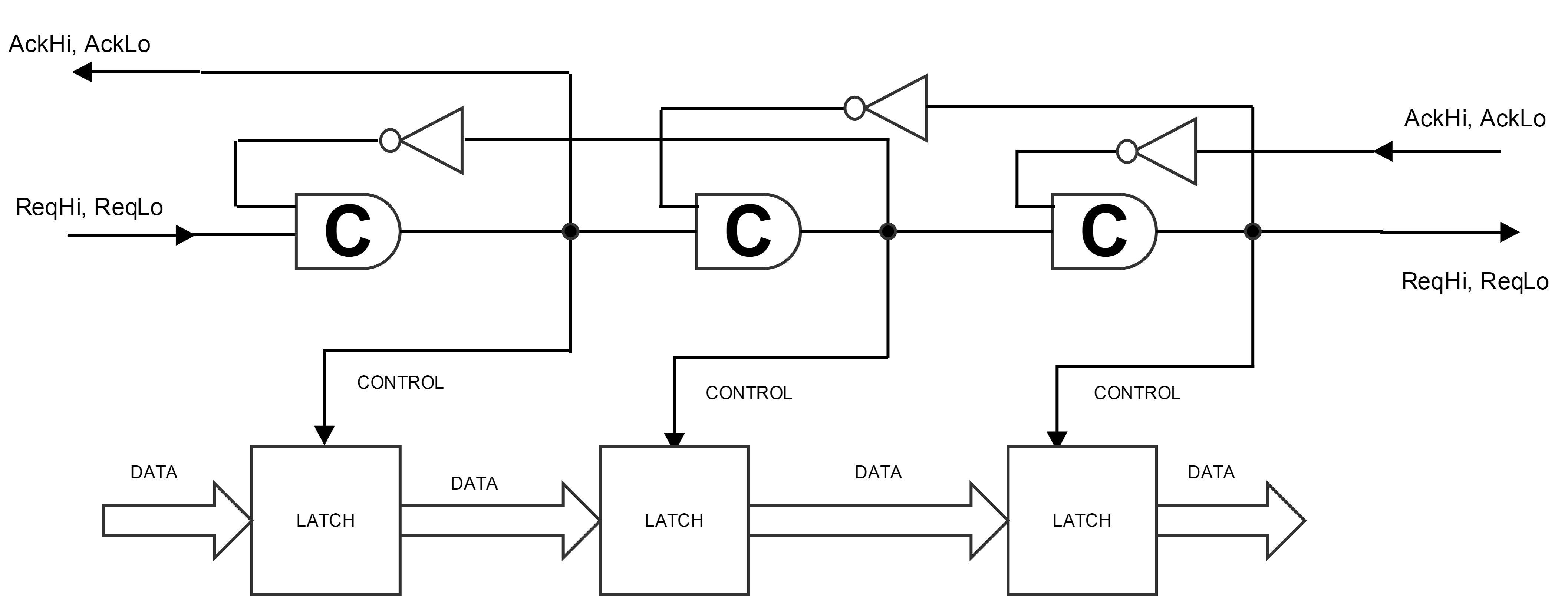
**PRISM QUERIES**

PRISM queries are used to guarantee that we preserve the property of temporal ordering. We explain how to use PRISM at the beginning of the additional materials. We select queries that test the pipelines ability to guarantee one species is present before another. We start with the initial condition that the request line is high (represented by the presence of the species reqhi).

|  |  |  |
| --- | --- | --- |
| PRISM Query in Written English | Translated Query | Outcome |
| "Guarantee that the first C-Element always outputs a high signal before the second" The high signal is represented by the species ahi, bhi. | P=?[(bhi < Th) U^[0 ,3] (ahi >= Th)]  Where s\_id15 is bhi and s\_id12 is ahi.    Th is a threshold which we have at 7 as this is roughly 2/3rds of the maximum population.  This shows that there is a near to zero probability of the second C-element outputting a 1 before the first. |  |
| "Guarantee that the not gate which inverts the signal produced by the second C-element fires after the first C-element" | P =?[(bNotUp < Th) U<=8 (ahi > Th)]  Where s\_id28 is bNotUp and s\_id12 is ahi.  The result is zero which means that in absolutely no case does the species bNotUp appear before the species ahi. |  |
| "Guarantee that the species ahi reaches   Its maximum population" | P=?[true U^[0,10] ahi > = popMax]    Where ahi is s\_id12.  PopMax is our maximum population any species can have. In this case it is 10.  As we can see we are guaraunteed that the species ahi will reach this maximum before decreasing. This is important as it means the species is not diminishing over time. |  |
| "Guarantee that request signal is propagated  to the end of the pipeline" | P=[F^[0,10] chi > = Th]  Where s\_id18 is chi. In this case we check if we do indeed propagate the signal to the end of the pipeline via the presence of the species ahi, bhi, chi etc.  In this case we are always guaranteed that our species chi will surpass our threshold 7 within 10 seconds. |  |

**Testing of the other Components – Queue, 4-Phase and Adder.**

The schematic for the queue is as follows:



The queue allows us to transfer data from latch to latch along the pipeline. The queue is useful as it represents the C-pipeline controlling a control flow system. By this we mean that anything could replace the latch computationally, and we would still have to wait until an operation completed before moving on. A latch does not transfer a value to the next latch until it has received a control signal from the pipeline. Below is the LBS code for the queue.

**Notes on the code:** Like the pipeline the GEC-LBS code for the queue is split into sections a,b,c etc which is donated by the first letter of each variable. However we add latches denoted by the blocks //MEM1, MEM2 etc. Just as shi represented the output species of our control latch in isolation in this case amshi, bmshi represents a high signal from each latch at each stage in the pipeline. Again the inputs to our pipeline are the request and acknowledge species. However additionally to this we have input to our queue represented by amxhi and amxlo.

|  |  |
| --- | --- |
| 2.0 - Queue | directive sample 20.0 300  directive plot ahi;amshi; bmshi; cmshi; dmshi  directive simulation deterministic  rate rt = 1.0;    // C ELEMENTS  //AM SWITCH 1  init aup 0.0  | init ant 0.0  | init adn 10.0    //AM SWITCH 2  | init bup 0.0  | init bnt 0.0  | init bdn 10.0    //AM SWITCH 3  | init cup 0.0  | init cnt 0.0  | init cdn 10.0    //AM SWITCH 4  | init dup 0.0  | init dnt 0.0  | init ddn 10.0    //AMP 1  | init ahi 0.0  | init ab 10.0  | init alo 0.0    //AMP 2  | init bhi 0.0  | init bb 10.0  | init blo 0.0    //AMP 3  | init chi 0.0  | init cb 10.0  | init clo 0.0    //AMP 4  | init dhi 0.0  | init db 10.0  | init dlo 0.0    //INPUTS  | init acchi 10.0  | init acclo 0.0    | init reqhi 10.0  | init reqlo 0.0      //NOT    | init bNotUp 0.0  | init bNotNt 10.0  | init bNotDown 0.0    | init cNotUp 0.0  | init cNotNt 10.0  | init cNotDown 0.0    | init dNotUp 0.0  | init dNotNt 10.0  | init dNotDown 0.0        //MEM1  | init amymid 10.0  | init amylo 0.0  | init amyhi 0.0  | init amsOne 10.0  | init amsTwo 0.0  | init amsThree 10.0  | init amsFour 0.0  | init amslo 0.0    //INPUTS  | init amxhi 10.0  | init amxlo 0.0  | init amrhi 0.0  //OUTPUTS  | init amshi 0.0  | init amslo 0.0    //MEM 2  | init bmymid 10.0  | init bmylo 0.0  | init bmyhi 0.0  | init bmsOne 10.0  | init bmsTwo 0.0  | init bmsThree 10.0  | init bmsFour 0.0  | init bmslo 0.0    //INPUTS  | init bmrhi 0.0  //OUTPUTS  | init bmshi 0.0  | init bmslo 0.0      //MEM 3  | init cmymid 10.0  | init cmylo 0.0  | init cmyhi 0.0  | init cmsOne 10.0  | init cmsTwo 0.0  | init cmsThree 10.0  | init cmsFour 0.0  | init cmslo 0.0    //INPUTS  | init cmrhi 0.0    //OUTPUTS  | init cmshi 0.0  | init cmslo 0.0      //MEM 4  | init dmymid 10.0  | init dmylo 0.0  | init dmyhi 0.0  | init dmsOne 10.0  | init dmsTwo 0.0  | init dmsThree 10.0  | init dmsFour 0.0  | init dmslo 0.0    //INPUTS  | init dmrhi 0.0    //OUTPUTS  | init dmshi 0.0  | init dmslo 0.0      //AREA 1  | adn + aup ->{rt} ant + ant  | ant + adn ->{rt} adn + adn  | ant + aup ->{rt} aup + aup    | adn + bNotUp ->{rt} ant + bNotUp  | ant + reqhi ->{rt} aup + reqhi  | aup + bNotDown->{rt} ant + bNotDown  | ant + reqlo ->{rt} adn + reqlo    //| alo + aup ->{rt} ahi + aup  //| ahi + adn ->{rt} alo + adn    | alo + ahi ->{rt} ab + ab  | ab + alo ->{rt} alo + alo  | ab + ahi ->{rt} ahi + ahi    | adn + ahi ->{rt} adn + ab  | adn + ab ->{rt} adn + alo  | aup + alo ->{rt} aup + ab  | aup + ab ->{rt} aup + ahi        | amyhi + amymid ->{rt} amyhi + amyhi  | amylo + amymid ->{rt} amylo + amylo    | amxhi + amylo ->{rt} amxhi + amymid  | amxhi + amymid ->{rt} amxhi + amyhi  | amxlo + amyhi ->{rt} amxlo + amymid  | amxlo + amymid ->{rt} amxlo + amylo    | amrhi + amyhi ->{rt} amrlo + amymid  | amrhi + amylo ->{rt} amrlo + amymid    | amyhi + amsOne ->{rt} amyhi + amsTwo  | ahi + amsTwo ->{rt} ahi + amshi  | alo + amshi ->{rt} alo + amsTwo  | amymid + amsTwo ->{rt} amymid + amsOne  | amylo + amsTwo ->{rt} amylo + amsOne  | amymid + amshi ->{rt} amymid + amsOne  | amylo + amshi ->{rt} amylo + amsOne    | amylo + amsThree ->{rt} amylo + amsFour  | ahi + amsFour ->{rt} ahi + amslo  | alo + amslo ->{rt} alo + amsFour  | amymid + amslo ->{rt} amymid + amsFour  | amyhi + amslo ->{rt} amyhi + amsFour  | amymid + amsFour ->{rt} amymid + amsThree  | amyhi + amsFour ->{rt} amymid + amsThree    //AREA 2  | bdn + bup ->{rt} bnt + bnt  | bnt + bdn ->{rt} bdn + bdn  | bnt + bup ->{rt} bup + bup    | bdn + cNotUp ->{rt} bnt + cNotUp  | bnt + ahi ->{rt} bup + ahi  | bup + cNotDown ->{rt} bnt + cNotDown  | bnt + alo ->{rt} bdn + alo    | blo + bhi ->{rt} bb + bb  | bb + blo ->{rt} blo + blo  | bb + bhi ->{rt} bhi + bhi    | bdn + bhi ->{rt} bdn + bb  | bdn + bb ->{rt} bdn + blo  | bup + blo ->{rt} bup + bb  | bup + bb ->{rt} bup + bhi      | bhi + bNotNt ->{rt} bhi + bNotDown  | bhi + bNotUp ->{rt} bhi + bNotDown    | blo + bNotNt ->{rt} blo + bNotUp  | blo + bNotDown ->{rt} blo + bNotUp      | bmyhi + bmymid ->{rt} bmyhi + bmyhi  | bmylo + bmymid ->{rt} bmylo + bmylo    | amshi + bmylo ->{rt} amshi + bmymid  | amshi + bmymid ->{rt} amshi + bmyhi  | amslo + bmyhi ->{rt} amslo + bmymid  | amslo + bmymid ->{rt} amslo + bmylo    | bmrhi + bmyhi ->{rt} bmrlo + bmymid  | bmrhi + bmylo ->{rt} bmrlo + bmymid    | bmyhi + bmsOne ->{rt} bmyhi + bmsTwo  | bhi + bmsTwo ->{rt} bhi + bmshi  | blo + bmshi ->{rt} blo + bmsTwo  | bmymid + bmsTwo ->{rt} bmymid + bmsOne  | bmylo + bmsTwo ->{rt} bmylo + bmsOne  | bmymid + bmshi ->{rt} bmymid + bmsOne  | bmylo + bmshi ->{rt} bmylo + bmsOne    | bmylo + bmsThree ->{rt} bmylo + bmsFour  | bhi + bmsFour ->{rt} bhi + bmslo  | blo + bmslo ->{rt} blo + bmsFour  | bmymid + bmslo ->{rt} bmymid + bmsFour  | bmyhi + bmslo ->{rt} bmyhi + bmsFour  | bmymid + bmsFour ->{rt} bmymid + bmsThree  | bmyhi + bmsFour ->{rt} bmymid + bmsThree    //AREA 3  | cdn + cup ->{rt} cnt + cnt  | cnt + cdn ->{rt} cdn + cdn  | cnt + cup ->{rt} cup + cup    | cdn + dNotUp ->{rt} cnt + dNotUp  | cnt + bhi ->{rt} cup + bhi  | cup + dNotDown ->{rt} cnt + dNotDown  | cnt + blo ->{rt} cdn + blo    | clo + chi ->{rt} cb + cb  | cb + clo ->{rt} clo + clo  | cb + chi ->{rt} chi + chi    | cdn + chi ->{rt} cdn + cb  | cdn + cb ->{rt} cdn + clo  | cup + clo ->{rt} cup + cb  | cup + cb ->{rt} cup + chi    | chi + cNotNt ->{rt} chi + cNotDown  | chi + cNotUp ->{rt} chi + cNotDown    | clo + cNotNt ->{rt} clo + cNotUp  | clo + cNotDown ->{rt} clo + cNotUp    | cmyhi + cmymid ->{rt} cmyhi + cmyhi  | cmylo + cmymid ->{rt} cmylo + cmylo    | bmshi + cmylo ->{rt} bmshi + cmymid  | bmshi + cmymid ->{rt} bmshi + cmyhi  | bmslo + cmyhi ->{rt} bmslo + cmymid  | bmslo + cmymid ->{rt} bmslo + cmylo    | cmrhi + cmyhi ->{rt} cmrlo + cmymid  | cmrhi + cmylo ->{rt} cmrlo + cmymid    | cmyhi + cmsOne ->{rt} cmyhi + cmsTwo  | chi + cmsTwo ->{rt} chi + cmshi  | clo + cmshi ->{rt} clo + cmsTwo  | cmymid + cmsTwo ->{rt} cmymid + cmsOne  | cmylo + cmsTwo ->{rt} cmylo + cmsOne  | cmymid + cmshi ->{rt} cmymid + cmsOne  | cmylo + cmshi ->{rt} cmylo + cmsOne    | cmylo + cmsThree ->{rt} cmylo + cmsFour  | chi + cmsFour ->{rt} chi + cmslo  | clo + cmslo ->{rt} clo + cmsFour  | cmymid + cmslo ->{rt} cmymid + cmsFour  | cmyhi + cmslo ->{rt} cmyhi + cmsFour  | cmymid + cmsFour ->{rt} cmymid + cmsThree  | cmyhi + cmsFour ->{rt} cmymid + cmsThree      //AREA 4  | ddn + dup ->{rt} dnt + dnt  | dnt + ddn ->{rt} ddn + ddn  | dnt + dup ->{rt} dup + dup    | ddn + acchi ->{rt} dnt + acchi  | dnt + chi ->{rt} dup + chi  | dup + acclo ->{rt} dnt + acclo  | dnt + clo ->{rt} ddn + clo    | dlo + dhi ->{rt} db + db  | db + dlo ->{rt} dlo + dlo  | db + dhi ->{rt} dhi + dhi    | ddn + dhi ->{rt} ddn + db  | ddn + db ->{rt} ddn + dlo  | dup + dlo ->{rt} dup + db  | dup + db ->{rt} dup + dhi    | dhi + dNotNt ->{rt} dhi + dNotDown  | dhi + dNotUp ->{rt} dhi + dNotDown    | dlo + dNotNt ->{rt} dlo + dNotUp  | dlo + dNotDown ->{rt} dlo + dNotUp    | dmyhi + dmymid ->{rt} dmyhi + dmyhi  | dmylo + dmymid ->{rt} dmylo + dmylo    | cmshi + dmylo ->{rt} cmshi + dmymid  | cmshi + dmymid ->{rt} cmshi + dmyhi  | cmslo + dmyhi ->{rt} cmslo + dmymid  | cmslo + dmymid ->{rt} cmslo + dmylo    | dmrhi + dmyhi ->{rt} dmrlo + dmymid  | dmrhi + dmylo ->{rt} dmrlo + dmymid    | dmyhi + dmsOne ->{rt} dmyhi + dmsTwo  | dhi + dmsTwo ->{rt} dhi + dmshi  | dlo + dmshi ->{rt} dlo + dmsTwo  | dmymid + dmsTwo ->{rt} dmymid + dmsOne  | dmylo + dmsTwo ->{rt} dmylo + dmsOne  | dmymid + dmshi ->{rt} dmymid + dmsOne  | dmylo + dmshi ->{rt} dmylo + dmsOne    | dmylo + dmsThree ->{rt} dmylo + dmsFour  | dhi + dmsFour ->{rt} dhi + dmslo  | dlo + dmslo ->{rt} dlo + dmsFour  | dmymid + dmslo ->{rt} dmymid + dmsFour  | dmyhi + dmslo ->{rt} dmyhi + dmsFour  | dmymid + dmsFour ->{rt} dmymid + dmsThree  | dmyhi + dmsFour ->{rt} dmymid + dmsThree    | dhi + reqhi ->{rt} dhi + reqlo  | dhi + acchi ->{rt} dhi + acclo  | dlo + reqlo ->{rt} dlo + reqhi  | dlo + acclo ->{rt} dlo + acchi |
| 2.1 Queue Variables | | dhi + reqhi ->{rt} dhi + reqlo  | dhi + acchi ->{rt} dhi + acclo  | dlo + reqlo ->{rt} dlo + reqhi  | dlo + acclo ->{rt} dlo + acchi    | zmOne + dhi ->{rt} zmZero + dhi  | zmZero + dlo ->{rt} zmOne + dlo |

We present several experiments to justify the validity of our queue implementation focusing mostly on propagating values through our latches. Remember the output species of latches are represented by amshi, amslo etc.

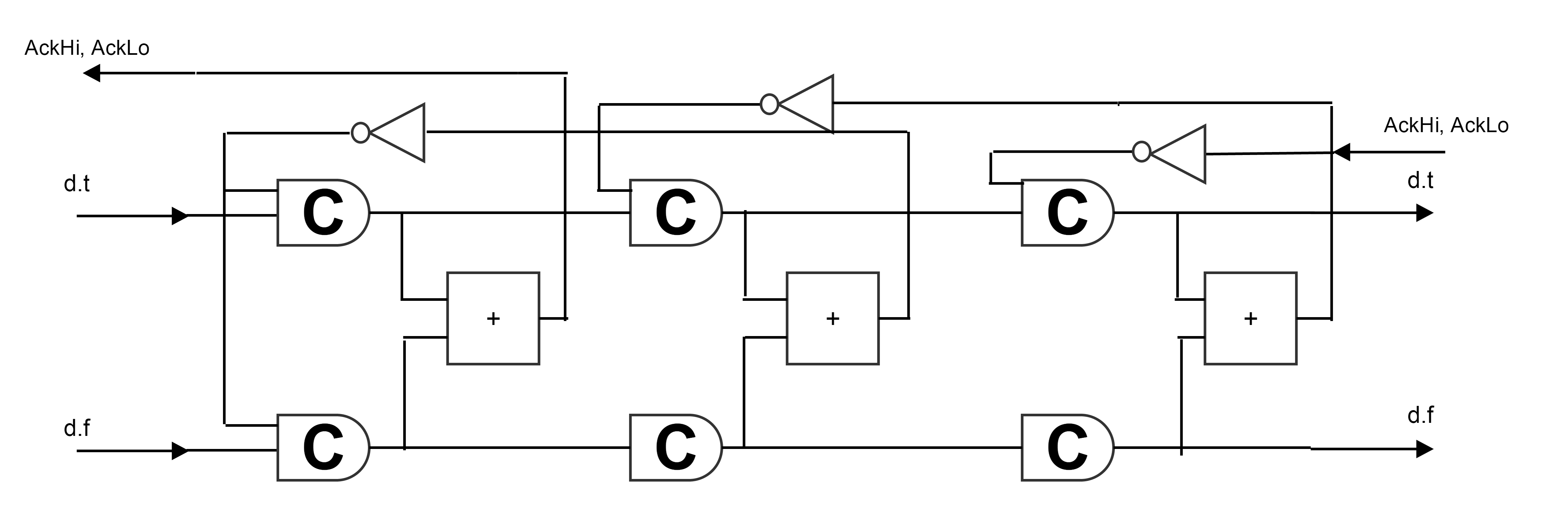
|  |  |  |
| --- | --- | --- |
| Queue Experiments | | |
| 1 | If we refer to the diagram latches are connected to the pipeline and do not 'pass on' their retained value (1/0) until a signal has been given by the corresponding C-Element. In this first experiment the signals refer to the value of logical 1 passed along the pipeline. The species abOne, bbOne, ccOne correspond to the values of logical one. We can see that they are interleaved witht the species from the C-element ahi,bhi,chi | Machine generated alternative text: Simulation  Chart  Inference  Table  Export Characterizatioyy  Populations Reactions Probability Map Probabilities  Resample Legend  Names Edit palette  amshl  bhi  bmshi  chi  cmshl  Time (s)  Microsoft |
| 2 | We propagate a value of 1 and then 0. We see that the value amshi, bmshi, cmshi fall accordingly. | Machine generated alternative text: |
| 3 | We reset the latches, pass a value of 0 and then 1 via the introduction of the rhi species to the pipeline. This time we the species amslo, bmslo representing a 0 value from the a latch, b latch etc. This species decreases as a value of 1 is introduced to the queue. | Machine generated alternative text: |

**PRISM QUERIES FOR QUEUE**

PRISM queries are used to guarantee that we reserve the property of temporal ordering. Here we use PRISM queries to show some interesting properties based on the latches within our queue. We perform these queries on a model with the initial condition that request line is high (species reqhi is present) and that we wish to propagate a value of one and then zero along the latches (using the species amshi, amslo etc.).

|  |  |  |
| --- | --- | --- |
| PRISM Query in Written English | Translated Query | Code Reference |
| "Guarantee that we pass on a value of one in our queue before we pass on a value of zero.” | P=?[(bmshi < Th) U^[0 ,3] (amslo >= Th )]     Where s\_id58 is bmshi and amslo is s\_id48. This means that we guarantee that, despite a value of zero being passed along the queue our latch retains a value of one BEFORE a value of zero. |  |
| "Guarantee that ordering is preserved. A value of One is stored in latch “a” before being stored in latch b” | P=?[(bmshi < Th) U^[0,3] (amshi >= Th)]    Where bmshi is s\_id58 and amshi is s\_id45.  This shows that under no circumstance is a value displayed further along the queue before it is displayed in the current latch. |  |
| "Guarantee that species ahi is present before a value of amshi/amslo” | P=?[(amshi<Th) U<=5 (s\_id12>=Th)]  Where s\_id12 is ahi.  Here we guarantee that the species ahi MUST be present before the species amshi. This is important as it shows that our control latch does NOT pass on a value until a signal has been received from the pipeline. |  |
| "Guarantee that latch value is propagated to the end of the pipeline” | P=[F^[0,10] chi > Th]  We guarantee here that if a value is given to the first latch then it is eventually propagated to the last latch thus confirming the functionality of the queue. |  |

The four phase protocol, enacted upon the C-pipeline, is as follows: firstly the sender sends data and sets request to high. The receiver then writes the data to the register and sets acknowledge to high. Then the sender responds by setting request to low and finally the receiver acknowledges this by setting acknowledge to low. Because there are four points to note here, this protocol is known as the 4-phase transfer protocol. In a 2-phase handshaking protocol we use the same idea however there is now no difference between transitions on the request and acknowledge wires. i.e. the transition from high to low is the same as the transition from low to high on each of these wires. We can see how this now leads to two events: (1) the sender sets data and request, (2) the receiver stores the data and sets acknowledge to high. Here is a diagram describing this process:



For further information please refer to Furber’s book linked to in the introduction. Our CRN implementation of such a pipeline is given below.

**Comments on the implementation:** Like before our pipeline Is split up into sections however unlike before, because there are two C-elements at each section the out signal of these C elements are adthi, adfhi, adtlo, adflo, bdthi, bdflo dependant upon which rail is being used. The species that is present at the and gate at each section is ahi, bhi, chi as before. Only when the signals adthi and adfhi are present will ahi be present by the protocol which governs the pipeline. Other than that the implementation is fairly similar to our 2-Phase standard C-Pipeline Implementation.

|  |  |
| --- | --- |
| 3.1 - 4Phase | directive sample 3.0 300  directive simulation deterministic  directive plot ahi; bhi; chi; alo; blo; clo    rate rt = 1.0;    // C ELEMENTS  //AM SWITCH 1  init adtup 0.0  | init adtnt 0.0  | init adtdn 10.0    | init adthi 0.0  | init adtmid 0.0  | init adtlo 10.0    | init adfup 0.0  | init adfnt 0.0  | init adfdn 10.0    | init adfhi 0.0  | init adfmid 0.0  | init adflo 10.0    //pulleys  | init ae 10.0  | init aw 10.0  | init aq 0.0    //outputs  | init alo 0.0  | init ahi 0.0    | init aNotUp 10.0  | init aNotDown 0.0      //AM SWITCH 2  | init bdtup 0.0  | init bdtnt 0.0  | init bdtdn 10.0    | init bdthi 0.0  | init bdtmid 0.0  | init bdtlo 10.0    | init bdfup 0.0  | init bdfnt 0.0  | init bdfdn 10.0    | init bdfhi 0.0  | init bdfmid 0.0  | init bdflo 10.0    //pulleys  | init be 10.0  | init bw 10.0  | init bq 0.0    //outputs  | init blo 0.0  | init bhi 0.0    | init bNotUp 10.0  | init bNotDown 0.0    //AM SWITCH 3  | init cdtup 0.0  | init cdtnt 0.0  | init cdtdn 10.0    | init cdthi 0.0  | init cdtmid 0.0  | init cdtlo 10.0    | init cdfup 0.0  | init cdfnt 0.0  | init cdfdn 10.0    | init cdfhi 0.0  | init bdfmid 0.0  | init cdflo 10.0    //pulleys  | init ce 10.0  | init cw 10.0  | init cq 0.0    //outputs  | init clo 0.0  | init chi 0.0    | init cNotUp 10.0  | init cNotDown 0.0      //INPUTS  | init reqthi 0.0  | init reqtlo 10.0  | init reqfhi 0.0  | init reqflo 10.0        //AREA 1  | adfdn + adfup ->{rt} adfnt + adfnt  | adfnt + adfdn ->{rt} adfdn + adfdn  | adfnt + adfup ->{rt} adfup + adfup    | adfdn + bNotUp ->{rt} adfnt + bNotUp  | adfnt + reqfhi ->{rt} adfup + reqfhi  | adfup + bNotDown->{rt} adfnt + bNotDown  | adfnt + reqflo ->{rt} adfdn + reqflo    | adflo + adfhi ->{rt} adfmid + adfmid  | adfmid + adflo ->{rt} adflo + adflo  | adfmid + adfhi ->{rt} adfhi + adfhi    | adflo + adfup ->{rt} adfmid + adfup  | adfmid + adfup ->{rt} adfhi + adfup  | adfhi + adfdn ->{rt} adfmid + adfdn  | adfmid + adfdn ->{rt} adflo + adfdn      | adtdn + adtup ->{rt} adtnt + adtnt  | adtnt + adtdn ->{rt} adtdn + adtdn  | adtnt + adtup ->{rt} adtup + adtup    | adtdn + bNotUp ->{rt} adtnt + bNotUp  | adtnt + reqthi ->{rt} adtup + reqthi  | adtup + bNotDown->{rt} adtnt + bNotDown  | adtnt + reqtlo ->{rt} adtdn + reqtlo    | adtlo + adthi ->{rt} adtmid + adtmid  | adtmid + adtlo ->{rt} adtlo + adtlo  | adtmid + adthi ->{rt} adthi + adthi    | adtlo + adtup ->{rt} adtmid + adtup  | adtmid + adtup ->{rt} adthi + adtup  | adthi + adtdn ->{rt} adtmid + adtdn  | adtmid + adtdn ->{rt} adtlo + adtdn    //reactions  | adthi + ae ->{rt} adthi + ahi  | adfhi + ae ->{rt} adfhi + ahi    | adtlo + aw ->{rt} adtlo + aq  | adflo + aq ->{rt} adflo + alo      //for persistance    |alo + adthi ->{rt} aw + adthi  |alo + adfhi ->{rt} aq + adfhi    |alo + ahi ->{rt} alo + ae    | aiflo + aNotDown ->{rt} aiflo + aNotUp  | aifhi + aNotUp ->{rt} aifhi + aNotDown      //AREA 2  | bdfdn + bdfup ->{rt} bdfnt + bdfnt  | bdfnt + bdfdn ->{rt} bdfdn + bdfdn  | bdfnt + bdfup ->{rt} bdfup + bdfup    | bdfdn + cNotUp ->{rt} bdfnt + cNotUp  | bdfnt + adfhi ->{rt} bdfup + adfhi  | bdfup + cNotDown->{rt} bdfnt + cNotDown  | bdfnt + adflo ->{rt} bdfdn + adflo    | bdflo + bdfhi ->{rt} bdfmid + bdfmid  | bdfmid + bdflo ->{rt} bdflo + bdflo  | bdfmid + bdfhi ->{rt} bdfhi + bdfhi    | bdflo + bdfup ->{rt} bdfmid + bdfup  | bdfmid + bdfup ->{rt} bdfhi + bdfup  | bdfhi + bdfdn ->{rt} bdfmid + bdfdn  | bdfmid + bdfdn ->{rt} bdflo + bdfdn    | bdtdn + bdtup ->{rt} bdtnt + bdtnt  | bdtnt + bdtdn ->{rt} bdtdn + bdtdn  | bdtnt + bdtup ->{rt} bdtup + bdtup    | bdtdn + cNotUp ->{rt} bdtnt + cNotUp  | bdtnt + adthi ->{rt} bdtup + adthi  | bdtup + cNotDown->{rt} bdtnt + cNotDown  | bdtnt + adtlo ->{rt} bdtdn + adtlo    | bdtlo + bdthi ->{rt} bdtmid + bdtmid  | bdtmid + bdtlo ->{rt} bdtlo + bdtlo  | bdtmid + bdthi ->{rt} bdthi + bdthi    | bdtlo + bdtup ->{rt} bdtmid + bdtup  | bdtmid + bdtup ->{rt} bdthi + bdtup  | bdthi + bdtdn ->{rt} bdtmid + bdtdn  | bdtmid + bdtdn ->{rt} bdtlo + bdtdn    //reactions  | bdthi + be ->{rt} bdthi + bhi  | bdfhi + be ->{rt} bdfhi + bhi    | bdtlo + bw ->{rt} bdtlo + bq  | bdflo + bq ->{rt} bdflo + blo      //for persistance    |blo + bdthi ->{rt} bw + bdthi  |blo + bdfhi ->{rt} bq + bdfhi    |blo + bhi ->{rt} blo + be    | biflo + bNotDown ->{rt} biflo + bNotUp  | bifhi + bNotUp ->{rt} bifhi + bNotDown    //AREA 3  | cdfdn + cdfup ->{rt} cdfnt + cdfnt  | cdfnt + cdfdn ->{rt} cdfdn + cdfdn  | cdfnt + cdfup ->{rt} cdfup + cdfup    | cdfdn + cNotUp ->{rt} cdfnt + cNotUp  | cdfnt + bdfhi ->{rt} cdfup + bdfhi  | cdfup + cNotDown->{rt} cdfnt + cNotDown  | cdfnt + bdflo ->{rt} cdfdn + bdflo    | cdflo + cdfhi ->{rt} cdfmid + cdfmid  | cdfmid + cdflo ->{rt} cdflo + cdflo  | cdfmid + cdfhi ->{rt} cdfhi + cdfhi    | cdflo + cdfup ->{rt} cdfmid + cdfup  | cdfmid + cdfup ->{rt} cdfhi + cdfup  | cdfhi + cdfdn ->{rt} cdfmid + cdfdn  | cdfmid + cdfdn ->{rt} cdflo + cdfdn      | cdtdn + cdtup ->{rt} cdtnt + cdtnt  | cdtnt + cdtdn ->{rt} cdtdn + cdtdn  | cdtnt + cdtup ->{rt} cdtup + cdtup    | cdtdn + cNotUp ->{rt} cdtnt + cNotUp  | cdtnt + bdthi ->{rt} cdtup + bdthi  | cdtup + bNotDown->{rt} cdtnt + cNotDown  | cdtnt + bdtlo ->{rt} cdtdn + bdtlo    | cdtlo + cdthi ->{rt} cdtmid + cdtmid  | cdtmid + cdtlo ->{rt} cdtlo + cdtlo  | cdtmid + cdthi ->{rt} cdthi + cdthi    | cdtlo + cdtup ->{rt} cdtmid + cdtup  | cdtmid + cdtup ->{rt} cdthi + cdtup  | cdthi + cdtdn ->{rt} cdtmid + cdtdn  | cdtmid + cdtdn ->{rt} cdtlo + cdtdn      //reactions  | cdthi + ce ->{rt} cdthi + chi  | cdfhi + ce ->{rt} cdfhi + chi    | cdtlo + cw ->{rt} cdtlo + cq  | cdflo + cq ->{rt} cdflo + clo      //for persistance    |clo + cdthi ->{rt} cw + cdthi  |clo + cdfhi ->{rt} cq + cdfhi    |clo + chi ->{rt} clo + ce    | ciflo + cNotDown ->{rt} ciflo + cNotUp  | cifhi + cNotUp ->{rt} cifhi + cNotDown    | chi + reqhi ->{rt} chi + reqlo  | clo + reqlo ->{rt} clo + reqhi |

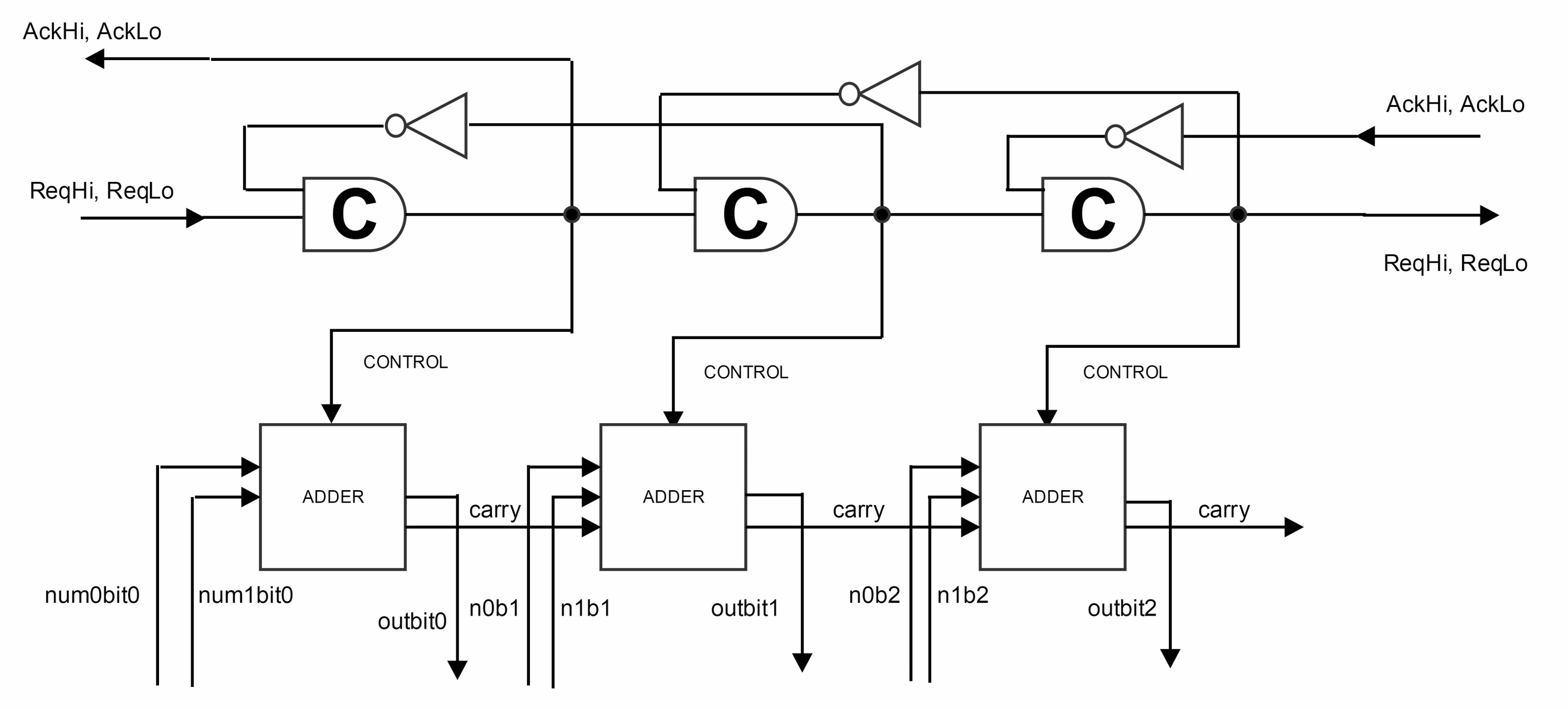
|  |  |  |
| --- | --- | --- |
| Four Phase Pipeline | | |
| 1 | Just as in our original C-pipeline we set the request line to high. The signal is propagated along the pipeline and the species ahi, bhi and chi are observed once again. | Machine generated alternative text: |

**PRISM QUERIES**

PRISM queries are used to guarantee that we reserve the property of temporal ordering. In this case, we guarantee that the outputs from the C-elements at each stage come strictly before the and gates in our 4-phase pipeline. These queries were performed upon a model where the initial conditions were that the request signal (reqhi species) was present. We also verify that a signal reaches the end of the pipeline as before.

|  |  |  |
| --- | --- | --- |
| PRISM Query in Written English | Translated Query | Code Reference |
| "Guarantee that the species adthi is present strictly before ahi” | P=?[(ahi < Th) U^[0 ,5] (adthi >= Th)]   Where ahi is s\_id35 and adthi is s\_id25.  In conjunction with the next query these two queries guarantee that ahi is a result of the AND gate receiving input from both C-elements. |  |
| "Guarantee that the species adfhi is present strictly before ahi” | P=?[(ahi < Th) U^[0 ,5] (adfhi >= Th)]   Where ahi is s\_id35 and adfhi is s\_id254  In conjunction with the query above these two queries guarantee that ahi is a result of the AND gate receiving input from both C-elements. |  |
| "Guarantee that the request signal is propagated to the end of the pipeline” | P=?[true U^[0,10] (chi>=Th)]    Where s\_id52 is chi.    This allows us to see that when the request line is set to high on both rails d.t and d.f we are able to propagate this signal to the end of the pipeline. |  |

An adder is a component which takes two bits as input and outputs the sum of those bits plus a carry signal. These adders can be composed in serial to produce a system which can add larger numbers by passing a carry bit along to the next adder. This is called a Ripple-Carry adder. We can control when this ripple-carry effect occurs with a signal from the pipeline. In this way we can make sure that the adder does not perform computation until a carry bit is present. An implementation of this is shown below.



**Comments on the code for the ripple carry adder:** The file is long but we have listed inputs clearly under //INPUTS. There are two inputs to each adder at each section along the pipeline (4-inputs for 1 or 0 dual rail). These are aaaOneIn, aaaZeroIn, aabOneIn, aabZeroIn, bbaOneIn, bbbOneIn etc. Each carry bit is fed along and is represented by the species aacOneOut etc. Standard non-carry outputs at each adder are bbsOneOut. From this it is fairly easy to track the species along the ripple-carry adder. The rest of the code is comprised of the C-pipeline and the logic gates which comprise the adders.

|  |  |
| --- | --- |
| 2.2 Ripple Carry Adder | directive sample 20.0 100  directive plot ahi;bhi;chi  directive simulation deterministic  rate rt = 1.0;    // C ELEMENTS  //AM SWITCH 1  init aup 0.0  | init ant 0.0  | init adn 10.0    //AM SWITCH 2  | init bup 0.0  | init bnt 0.0  | init bdn 10.0    //AM SWITCH 3  | init cup 0.0  | init cnt 0.0  | init cdn 10.0    //AM SWITCH 4  | init dup 0.0  | init dnt 0.0  | init ddn 10.0    //AMP 1  | init ahi 0.0  | init ab 10.0  | init alo 0.0    //AMP 2  | init bhi 0.0  | init bb 10.0  | init blo 0.0    //AMP 3  | init chi 0.0  | init cb 10.0  | init clo 0.0    //AMP 4  | init dhi 0.0  | init db 10.0  | init dlo 0.0    //INPUTS  | init acchi 10.0  | init acclo 0.0    | init reqhi 10.0  | init reqlo 0.0        //inputs  init aaaOneIn 10.0  | init aaaZeroIn 0.0    | init aabOneIn 10.0  | init aabZeroIn 0.0    | init aacOneIn 10.0  | init aacZeroIn 0.0      //outputs  | init aasOneOut 0.0  | init aasZeroOut 0.0    | init aacOneOut 0.0  | init aacZeroOut 0.0    //X0R1  | init aaaxorOneLone 0.0  | init aaxorOnelTwo 10.0  | init aaxorOnelThree 10.0  | init aaxorOnelFour 0.0  | init aaxorOneHi 0.0  | init aaxorOneLo 0.0    //XOR2    | init aaxorTwolOne 0.0  | init aaxorTwolTwo 10.0  | init aaxorTwolThree 10.0  | init aaxorTwolFour 0.0    //AND1  | init aaandOnelq 0.0  | init aaandOnelw 10.0  | init aaandOnele 10.0  | init aaandOneHi 0.0  | init aaandOneLo 0.0    //AND2  | init aaandTwole 0.0  | init aaandTwolw 10.0  | init aaandTwolq 10.0  | init aaandTwoHi 0.0  | init aaandTwoLo 0.0    //OR1  | init aaorle 10.0  | init aaorlw 10.0  | init aaorlq 0.0        //inputs  init bbaOneIn 10.0  | init bbaZeroIn 0.0    | init bbbOneIn 10.0  | init bbbZeroIn 0.0    | init aacOneOut 10.0  | init aacZeroOut 0.0      //outputs  | init bbsOneOut 0.0  | init bbsZeroOut 0.0    | init bbcOneOut 0.0  | init bbcZeroOut 0.0    //X0R1  | init bbaxorOneLone 0.0  | init bbxorOnelTwo 10.0  | init bbxorOnelThree 10.0  | init bbxorOnelFour 0.0  | init bbxorOneHi 0.0  | init bbxorOneLo 0.0    //XOR2    | init bbxorTwolOne 0.0  | init bbxorTwolTwo 10.0  | init bbxorTwolThree 10.0  | init bbxorTwolFour 0.0    //AND1  | init bbandOnelq 0.0  | init bbandOnelw 10.0  | init bbandOnele 10.0  | init bbandOneHi 0.0  | init bbandOneLo 0.0    //AND2  | init bbandTwole 0.0  | init bbandTwolw 10.0  | init bbandTwolq 10.0  | init bbandTwoHi 0.0  | init bbandTwoLo 0.0    //OR1  | init bborle 10.0  | init bborlw 10.0  | init bborlq 0.0        //inputs  init ccaOneIn 10.0  | init ccaZeroIn 0.0    | init ccbOneIn 10.0  | init ccbZeroIn 0.0    | init bbcOneOut 10.0  | init bbcZeroOut 0.0      //outputs  | init ccsOneOut 0.0  | init ccsZeroOut 0.0    | init cccOneOut 0.0  | init cccZeroOut 0.0    //X0R1  | init ccaxorOneLone 0.0  | init ccxorOnelTwo 10.0  | init ccxorOnelThree 10.0  | init ccxorOnelFour 0.0  | init ccxorOneHi 0.0  | init ccxorOneLo 0.0    //XOR2    | init ccxorTwolOne 0.0  | init ccxorTwolTwo 10.0  | init ccxorTwolThree 10.0  | init ccxorTwolFour 0.0    //AND1  | init ccandOnelq 0.0  | init ccandOnelw 10.0  | init ccandOnele 10.0  | init ccandOneHi 0.0  | init ccandOneLo 0.0    //AND2  | init ccandTwole 0.0  | init ccandTwolw 10.0  | init ccandTwolq 10.0  | init ccandTwoHi 0.0  | init ccandTwoLo 0.0    //OR1  | init ccorle 10.0  | init ccorlw 10.0  | init ccorlq 0.0    | init aabridgeOne 10.0  | init aabridgeTwo 0.0  | init aabridgeThree 10.0  | init aabridgeFour 0.0  | init aabridgeOneOut 0.0  | init aabridgeZeroOut 0.0    | init bbbridgeOne 10.0  | init bbbridgeTwo 0.0  | init bbbridgeThree 10.0  | init bbbridgeFour 0.0  | init bbbridgeOneOut 0.0  | init bbbridgeZeroOut 0.0    | init ccbridgeOne 10.0  | init ccbridgeTwo 0.0  | init ccbridgeThree 10.0  | init ccbridgeFour 0.0  | init ccbridgeOneOut 0.0  | init ccbridgeZeroOut 0.0      //NOT    | init bNotUp 0.0  | init bNotNt 10.0  | init bNotDown 0.0    | init cNotUp 0.0  | init cNotNt 10.0  | init cNotDown 0.0    | init dNotUp 0.0  | init dNotNt 10.0  | init dNotDown 0.0      //AREA 1  | adn + aup ->{rt} ant + ant  | ant + adn ->{rt} adn + adn  | ant + aup ->{rt} aup + aup    | adn + bNotUp ->{rt} ant + bNotUp  | ant + reqhi ->{rt} aup + reqhi  | aup + bNotDown->{rt} ant + bNotDown  | ant + reqlo ->{rt} adn + reqlo    //| alo + aup ->{rt} ahi + aup  //| ahi + adn ->{rt} alo + adn    | alo + ahi ->{rt} ab + ab  | ab + alo ->{rt} alo + alo  | ab + ahi ->{rt} ahi + ahi    | adn + ahi ->{rt} adn + ab  | adn + ab ->{rt} adn + alo  | aup + alo ->{rt} aup + ab  | aup + ab ->{rt} aup + ahi    //AREA 2  | bdn + bup ->{rt} bnt + bnt  | bnt + bdn ->{rt} bdn + bdn  | bnt + bup ->{rt} bup + bup    | bdn + cNotUp ->{rt} bnt + cNotUp  | bnt + ahi ->{rt} bup + ahi  | bup + cNotDown ->{rt} bnt + cNotDown  | bnt + alo ->{rt} bdn + alo    //| blo + bup ->{rt} bhi + bup  //| bhi + bdn ->{rt} blo + bdn    | blo + bhi ->{rt} bb + bb  | bb + blo ->{rt} blo + blo  | bb + bhi ->{rt} bhi + bhi    | bdn + bhi ->{rt} bdn + bb  | bdn + bb ->{rt} bdn + blo  | bup + blo ->{rt} bup + bb  | bup + bb ->{rt} bup + bhi      | bhi + bNotNt ->{rt} bhi + bNotDown  | bhi + bNotUp ->{rt} bhi + bNotDown    | blo + bNotNt ->{rt} blo + bNotUp  | blo + bNotDown ->{rt} blo + bNotUp    //AREA 3  | cdn + cup ->{rt} cnt + cnt  | cnt + cdn ->{rt} cdn + cdn  | cnt + cup ->{rt} cup + cup    | cdn + dNotUp ->{rt} cnt + dNotUp  | cnt + bhi ->{rt} cup + bhi  | cup + dNotDown ->{rt} cnt + dNotDown  | cnt + blo ->{rt} cdn + blo    //| clo + cup ->{rt} chi + cup  //| chi + cdn ->{rt} clo + cdn    | clo + chi ->{rt} cb + cb  | cb + clo ->{rt} clo + clo  | cb + chi ->{rt} chi + chi    | cdn + chi ->{rt} cdn + cb  | cdn + cb ->{rt} cdn + clo  | cup + clo ->{rt} cup + cb  | cup + cb ->{rt} cup + chi    | chi + cNotNt ->{rt} chi + cNotDown  | chi + cNotUp ->{rt} chi + cNotDown    | clo + cNotNt ->{rt} clo + cNotUp  | clo + cNotDown ->{rt} clo + cNotUp    //| chi + reqhi ->{rt} chi + reqlo  //| chi + acchi ->{rt} chi    //AREA 4  | ddn + dup ->{rt} dnt + dnt  | dnt + ddn ->{rt} ddn + ddn  | dnt + dup ->{rt} dup + dup    | ddn + acchi ->{rt} dnt + acchi  | dnt + chi ->{rt} dup + chi  | dup + acclo ->{rt} dnt + acclo  | dnt + clo ->{rt} ddn + clo    //| dlo + dup ->{rt} dhi + dup  //| dhi + ddn ->{rt} dlo + ddn    | dlo + dhi ->{rt} db + db  | db + dlo ->{rt} dlo + dlo  | db + dhi ->{rt} dhi + dhi    | ddn + dhi ->{rt} ddn + db  | ddn + db ->{rt} ddn + dlo  | dup + dlo ->{rt} dup + db  | dup + db ->{rt} dup + dhi    | dhi + dNotNt ->{rt} dhi + dNotDown  | dhi + dNotUp ->{rt} dhi + dNotDown    | dlo + dNotNt ->{rt} dlo + dNotUp  | dlo + dNotDown ->{rt} dlo + dNotUp      | aacOneOut + aabridgeOne ->{rt} aacOneOut + aabridgeTwo  | aacZeroOut + aabridgeThree ->{rt} aacZeroOut + aabridgeFour  | ahi + aabridgeTwo ->{rt} ahi + aabridgeOneOut  | ahi + aabridgeFour ->{rt} ahi + aabridgeZeroOut    | aacZeroOut + aabridgeOneOut ->{rt} aacZeroOut + aabridgeTwo  | aacZeroOut + aabridgeTwo ->{rt} aacZeroOut + aabridgeOne  | aacOneOut + aabridgeZeroOut ->{rt} aacOneOut + aabridgeFour  | aacOneOut + aabridgeFour ->{rt} aacOneOut + aabridgeThree    | bbcOneOut + bbbridgeOne ->{rt} bbcOneOut + bbbridgeTwo  | bbcZeroOut + bbbridgeThree ->{rt} bbcZeroOut + bbbridgeFour  | bhi + bbbridgeTwo ->{rt} bhi + bbbridgeOneOut  | bhi + bbbridgeFour ->{rt} bhi + bbbridgeZeroOut    | bbcZeroOut + bbbridgeOneOut ->{rt} bbcZeroOut + bbbridgeTwo  | bbcZeroOut + bbbridgeTwo ->{rt} bbcZeroOut + bbbridgeOne  | bbcOneOut + bbbridgeZeroOut ->{rt} bbcOneOut + bbbridgeFour  | bbcOneOut + bbbridgeFour ->{rt} bbcOneOut + bbbridgeThree    | cccOneOut + ccbridgeOne ->{rt} cccOneOut + ccbridgeTwo  | cccZeroOut + ccbridgeThree ->{rt} cccZeroOut + ccbridgeFour  | chi + ccbridgeTwo ->{rt} chi + ccbridgeOneOut  | chi + ccbridgeFour ->{rt} chi + ccbridgeZeroOut    | cccZeroOut + ccbridgeOneOut ->{rt} cccZeroOut + ccbridgeTwo  | cccZeroOut + ccbridgeTwo ->{rt} cccZeroOut + ccbridgeOne  | cccOneOut + ccbridgeZeroOut ->{rt} cccOneOut + ccbridgeFour  | cccOneOut + ccbridgeFour ->{rt} cccOneOut + ccbridgeThree    //XOR1  //reactions  | aaaOneIn + aaxorOnelFour ->{rt} aaaOneIn + aaxorOneHi  | aabOneIn + aaxorOnelFour ->{rt} aabOneIn + aaxorOneHi    | aaaZeroIn + aaxorOnelThree ->{rt} aaaZeroIn + aaxorOnelFour  | aabZeroIn + aaxorOnelThree ->{rt} aabZeroIn + aaxorOnelFour    | aaaOneIn + aaxorOnelTwo ->{rt} aaaOneIn + aaaxorOneLone  | aabZeroIn + aaxorOnelTwo ->{rt} aabZeroIn + aaaxorOneLone    | aabOneIn + aaaxorOneLone ->{rt} aabOneIn + aaxorOneLo  | aaaZeroIn + aaaxorOneLone ->{rt} aaaZeroIn + aaxorOneLo      //for persistance    | aaxorOneLo + aaxorOnelFour ->{rt} aaxorOneLo + aaxorOnelThree  | aaxorOneLo + aaxorOneHi ->{rt} aaxorOneLo + aaxorOnelFour    | aaxorOneHi + aaaxorOneLone->{rt} aaxorOneHi + aaxorOnelTwo  | aaxorOneHi + aaxorOneLo ->{rt} aaxorOneHi + aaaxorOneLone      //XOR2    | aaxorOneHi + aaxorTwolFour ->{rt} aaxorOneHi + aasOneOut  | aacOneIn + aaxorTwolFour ->{rt} aacOneIn + aasOneOut    | aaxorOneLo + aaxorTwolThree ->{rt} aaxorOneLo + aaxorTwolFour  | aacZeroIn + aaxorTwolThree ->{rt} aacZeroIn + aaxorTwolFour    | aaxorOneHi + aaxorTwolTwo ->{rt} aaxorOneHi + aaxorTwolOne  | aacZeroIn + aaxorTwolTwo ->{rt} aacZeroIn + aaxorTwolOne    | aacOneIn + aaxorTwolOne ->{rt} aacOneIn + aasZeroOut  | aaxorOneLo + aaxorTwolOne ->{rt} aaxorOneLo + aasZeroOut      //for persistance    | aasZeroOut + aaxorTwolFour ->{rt} aasZeroOut + aaxorTwolThree  | aasZeroOut + aasOneOut ->{rt} aasZeroOut + aaxorTwolFour    | aasOneOut + aasZeroOut ->{rt} aasOneOut + aaxorTwolTwo  | aasOneOut + aasZeroOut ->{rt} aasOneOut + aasZeroOut      //AND1  //reactions    |aaaOneIn + aaandOnelq ->{rt} aaaOneIn + aaandOneHi  |aabOneIn + aaandOnelw ->{rt} aabOneIn + aaandOnelq    |aaaZeroIn + aaandOnele ->{rt} aaaZeroIn + aaandOneLo  |aabZeroIn + aaandOnele ->{rt} aabZeroIn + aaandOneLo    //for persistance    | aaandOneHi + aaaZeroIn ->{rt} aaandOnelq + aaaZeroIn  | aaandOneHi + aabZeroIn ->{rt} aaandOnelw + aabZeroIn    | aaandOneLo + aaandOneHi ->{rt} aaandOneHi + aaandOnele    //AND2    //reactions    |aacOneIn + aaandTwole ->{rt} aacOneIn + aaandTwoHi  |aaxorOneHi + aaandTwolw ->{rt} aaxorOneHi + aaandTwole    |aacZeroIn + aaandTwolq ->{rt} aacZeroIn + aaandTwoLo  |aaxorOneLo + aaandTwolq ->{rt} aaxorOneLo + aaandTwoLo    //for persistance    |aaandTwoHi + aacZeroIn ->{rt} aaandTwole + aacZeroIn  |aaandTwoHi + aaxorOneLo ->{rt} aaandTwolw + aaxorOneLo    |aaandTwoLo + aaandTwoHi ->{rt} aaandTwoHi + aaandTwolq      //OR    | aaandOneHi + aaorle ->{rt} aaandOneHi + aacOneOut  | aaandTwoHi + aaorle ->{rt} aaandTwoHi + aacOneOut    | aaandOneLo + aaorlw ->{rt} aaandOneLo + aaorlq  | aaandTwoLo + aaorlq ->{rt} aaandTwoLo + aacZeroOut      //for persistance  | aacZeroOut + aaandOneHi ->{rt} aaorlw + aaandOneHi  | aacZeroOut + aaandTwoHi ->{rt} aaorlq + aaandTwoHi  | aacZeroOut + aaacOneOut ->{rt} aacZeroOut + aaorle      //XOR1  //reactions  | bbaOneIn + bbxorOnelFour ->{rt} bbaOneIn + bbxorOneHi  | bbbOneIn + bbxorOnelFour ->{rt} bbbOneIn + bbxorOneHi    | bbaZeroIn + bbxorOnelThree ->{rt} bbaZeroIn + bbxorOnelFour  | bbbZeroIn + bbxorOnelThree ->{rt} bbbZeroIn + bbxorOnelFour    | bbaOneIn + bbxorOnelTwo ->{rt} bbaOneIn + bbaxorOneLone  | bbbZeroIn + bbxorOnelTwo ->{rt} bbbZeroIn + bbaxorOneLone    | bbbOneIn + bbaxorOneLone ->{rt} bbbOneIn + bbxorOneLo  | bbaZeroIn + bbaxorOneLone ->{rt} bbaZeroIn + bbxorOneLo      //for persistance    | bbxorOneLo + bbxorOnelFour ->{rt} bbxorOneLo + bbxorOnelThree  | bbxorOneLo + bbxorOneHi ->{rt} bbxorOneLo + bbxorOnelFour    | bbxorOneHi + bbaxorOneLone->{rt} bbxorOneHi + bbxorOnelTwo  | bbxorOneHi + bbxorOneLo ->{rt} bbxorOneHi + bbaxorOneLone      //XOR2    | bbxorOneHi + bbxorTwolFour ->{rt} bbxorOneHi + bbsOneOut  | aabridgeOneOut + bbxorTwolFour ->{rt} aabridgeOneOut + bbsOneOut    | bbxorOneLo + bbxorTwolThree ->{rt} bbxorOneLo + bbxorTwolFour  | aabridgeZeroOut + bbxorTwolThree ->{rt} aabridgeZeroOut + bbxorTwolFour    | bbxorOneHi + bbxorTwolTwo ->{rt} bbxorOneHi + bbxorTwolOne  | aabridgeZeroOut + bbxorTwolTwo ->{rt} aabridgeZeroOut + bbxorTwolOne    | aabridgeOneOut + bbxorTwolOne ->{rt} aabridgeOneOut + bbsZeroOut  | bbxorOneLo + bbxorTwolOne ->{rt} bbxorOneLo + bbsZeroOut      //for persistance    | bbsZeroOut + bbxorTwolFour ->{rt} bbsZeroOut + bbxorTwolThree  | bbsZeroOut + bbsOneOut ->{rt} bbsZeroOut + bbxorTwolFour    | bbsOneOut + bbsZeroOut ->{rt} bbsOneOut + bbxorTwolTwo  | bbsOneOut + bbsZeroOut ->{rt} bbsOneOut + bbsZeroOut      //AND1  //reactions    |bbaOneIn + bbandOnelq ->{rt} bbaOneIn + bbandOneHi  |bbbOneIn + bbandOnelw ->{rt} bbbOneIn + bbandOnelq    |bbaZeroIn + bbandOnele ->{rt} bbaZeroIn + bbandOneLo  |bbbZeroIn + bbandOnele ->{rt} bbbZeroIn + bbandOneLo    //for persistance    | bbandOneHi + bbaZeroIn ->{rt} bbandOnelq + bbaZeroIn  | bbandOneHi + bbbZeroIn ->{rt} bbandOnelw + bbbZeroIn    | bbandOneLo + bbandOneHi ->{rt} bbandOneHi + bbandOnele    //AND2    //reactions    |aabridgeOneOut + bbandTwole ->{rt} aabridgeOneOut + bbandTwoHi  |bbxorOneHi + bbandTwolw ->{rt} bbxorOneHi + bbandTwole    |aabridgeZeroOut + bbandTwolq ->{rt} aabridgeZeroOut + bbandTwoLo  |bbxorOneLo + bbandTwolq ->{rt} bbxorOneLo + bbandTwoLo    //for persistance    |bbandTwoHi + aabridgeZeroOut ->{rt} bbandTwole + aabridgeZeroOut  |bbandTwoHi + bbxorOneLo ->{rt} bbandTwolw + bbxorOneLo    |bbandTwoLo + bbandTwoHi ->{rt} bbandTwoHi + bbandTwolq      //OR    | bbandOneHi + bborle ->{rt} bbandOneHi + bbcOneOut  | bbandTwoHi + bborle ->{rt} bbandTwoHi + bbcOneOut    | bbandOneLo + bborlw ->{rt} bbandOneLo + bborlq  | bbandTwoLo + bborlq ->{rt} bbandTwoLo + bbcZeroOut      //for persistance  | bbcZeroOut + bbandOneHi ->{rt} bborlw + bbandOneHi  | bbcZeroOut + bbandTwoHi ->{rt} bborlq + bbandTwoHi  | bbcZeroOut + bbacOneOut ->{rt} bbcZeroOut + bborle      //XOR1  //reactions  | ccaOneIn + ccxorOnelFour ->{rt} ccaOneIn + ccxorOneHi  | ccbOneIn + ccxorOnelFour ->{rt} ccbOneIn + ccxorOneHi    | ccaZeroIn + ccxorOnelThree ->{rt} ccaZeroIn + ccxorOnelFour  | ccbZeroIn + ccxorOnelThree ->{rt} ccbZeroIn + ccxorOnelFour    | ccaOneIn + ccxorOnelTwo ->{rt} ccaOneIn + ccaxorOneLone  | ccbZeroIn + ccxorOnelTwo ->{rt} ccbZeroIn + ccaxorOneLone    | ccbOneIn + ccaxorOneLone ->{rt} ccbOneIn + ccxorOneLo  | ccaZeroIn + ccaxorOneLone ->{rt} ccaZeroIn + ccxorOneLo      //for persistance    | ccxorOneLo + ccxorOnelFour ->{rt} ccxorOneLo + ccxorOnelThree  | ccxorOneLo + ccxorOneHi ->{rt} ccxorOneLo + ccxorOnelFour    | ccxorOneHi + ccaxorOneLone->{rt} ccxorOneHi + ccxorOnelTwo  | ccxorOneHi + ccxorOneLo ->{rt} ccxorOneHi + ccaxorOneLone      //XOR2    | ccxorOneHi + ccxorTwolFour ->{rt} ccxorOneHi + ccsOneOut  | bbbridgeOneOut + ccxorTwolFour ->{rt} bbbridgeOneOut + ccsOneOut    | ccxorOneLo + ccxorTwolThree ->{rt} ccxorOneLo + ccxorTwolFour  | bbbridgeZeroOut + ccxorTwolThree ->{rt} bbbridgeZeroOut + ccxorTwolFour    | ccxorOneHi + ccxorTwolTwo ->{rt} ccxorOneHi + ccxorTwolOne  | bbbridgeZeroOut + ccxorTwolTwo ->{rt} bbbridgeZeroOut + ccxorTwolOne    | bbbridgeOneOut + ccxorTwolOne ->{rt} bbbridgeOneOut + ccsZeroOut  | ccxorOneLo + ccxorTwolOne ->{rt} ccxorOneLo + ccsZeroOut      //for persistance    | ccsZeroOut + ccxorTwolFour ->{rt} ccsZeroOut + ccxorTwolThree  | ccsZeroOut + ccsOneOut ->{rt} ccsZeroOut + ccxorTwolFour    | ccsOneOut + ccsZeroOut ->{rt} ccsOneOut + ccxorTwolTwo  | ccsOneOut + ccsZeroOut ->{rt} ccsOneOut + ccsZeroOut      //AND1  //reactions    |ccaOneIn + ccandOnelq ->{rt} ccaOneIn + ccandOneHi  |ccbOneIn + ccandOnelw ->{rt} ccbOneIn + ccandOnelq    |ccaZeroIn + ccandOnele ->{rt} ccaZeroIn + ccandOneLo  |ccbZeroIn + ccandOnele ->{rt} ccbZeroIn + ccandOneLo    //for persistance    | ccandOneHi + ccaZeroIn ->{rt} ccandOnelq + ccaZeroIn  | ccandOneHi + ccbZeroIn ->{rt} ccandOnelw + ccbZeroIn    | ccandOneLo + ccandOneHi ->{rt} ccandOneHi + ccandOnele    //AND2    //reactions    |bbbridgeOneOut + ccandTwole ->{rt} bbbridgeOneOut + ccandTwoHi  |ccxorOneHi + ccandTwolw ->{rt} ccxorOneHi + ccandTwole    |bbbridgeZeroOut + ccandTwolq ->{rt} bbbridgeZeroOut + ccandTwoLo  |ccxorOneLo + ccandTwolq ->{rt} ccxorOneLo + ccandTwoLo    //for persistance    |ccandTwoHi + bbbridgeZeroOut ->{rt} ccandTwole + bbbridgeZeroOut  |ccandTwoHi + ccxorOneLo ->{rt} ccandTwolw + ccxorOneLo    |ccandTwoLo + ccandTwoHi ->{rt} ccandTwoHi + ccandTwolq      //OR    | ccandOneHi + ccorle ->{rt} ccandOneHi + cccOneOut  | ccandTwoHi + ccorle ->{rt} ccandTwoHi + cccOneOut    | ccandOneLo + ccorlw ->{rt} ccandOneLo + ccorlq  | ccandTwoLo + ccorlq ->{rt} ccandTwoLo + cccZeroOut      //for persistance  | cccZeroOut + ccandOneHi ->{rt} ccorlw + ccandOneHi  | cccZeroOut + ccandTwoHi ->{rt} ccorlq + ccandTwoHi  | cccZeroOut + ccacOneOut ->{rt} cccZeroOut + ccorle |

We present a few experiments from our adder. Firstly we show plots of an individual adder in GEC. We then show, using PRISM properties the output of the ripple-carry adder as a whole.

|  |  |  |
| --- | --- | --- |
| Adder | Inputs:   * 1. 0 0 0c   2. 1 0 1c   3. 0 1 1c   4. 1 1 1c   In these four plots we realise the sums listed as input above. In the first plot we expect this to sum to zero with zero carry. As we can see the species asZeroOut (green) and acZeroOut (yellow) are both present signifying the fact we have 0 0c output.Similarly in the second plot we have a 0 1c outcome which you would expect given the input. The zero species is represented in green and the carry species is represented in blue. In the 3rd plot we have the same result which makes sense given the input is basically the same. In the fourth plot we have a 1 1c output which are represented by the species asOneOut and acOneOut represented by the red and blue plots. This set of experiments has hopefully convinced the reader of the functionality of the adder as an isolated unit. | C:\Users\Max\AppData\Local\Temp\msohtmlclip1\02\clip_image001.png    C:\Users\Max\AppData\Local\Temp\msohtmlclip1\02\clip_image002.png    C:\Users\Max\AppData\Local\Temp\msohtmlclip1\02\clip_image003.png        C:\Users\Max\AppData\Local\Temp\msohtmlclip1\02\clip_image004.png |

**PRISM QUERIES**

We now simulate some inputs on our ripple-carry adder and verify them with PRISM. The plots below represent the species ccaOneOut (s\_id96), ccaZeroOut (s\_id97) and cccOneOut (s\_id152). From this we can determine the output bit and carry from the last adder in the chain. Our input is given as the values that were fed to each adder (there are two values as the carry comes from the previous adder). The PRISM results confirm the correct working nature of our adder.

|  |  |
| --- | --- |
| Input | Result |
| Adder A:1 0  Adder B:1 0  Adder C: 0 0 |  |
| Adder A: 1 1  Adder B: 1 1  Adder C: 1 1 |  |
| Adder A: 1 1  Adder B: 1 1  Adder C: 1 0 |  |