(If you read it then send comments)

Asynchronous Early Output and Early Acknowledge Dual-Rail Protocols

A thesis submitted to the University of Manchester
for the degree of Master of Philosophy in the
Faculty of Science & Engineering

October 2002

Charlie Brej

Department of Computer Science

Contents

List of Figures

List of Tables

Abstract

Declaration

Copyright

The Author

After being abandoned at birth Charlie was adopted by a herd of wild mountain postmen. Growing up on the rough streets of Nottingham he learned the vital survival skills of circuit design and croquet. After joining the circus for a number of years as a wolf child act Charlie decided that the best way to conquer the world would be to do research for large international conglomerate. After earning his first million he decided to suddenly spend it all on cheap lap dancers and feeding his growing gummy bear addiction. Spending the next 3 years on the streets and on one night consuming several gallons of Dandelion and Burdock, Charlie found himself a shadow of his former self and decided to return to hardware design. For weeks Charlie could be seen in Canal Street offering to sell his body to hardware designers wanting a gimp to optimise their schematics for food and a few precious cans of D&B. His big break came when opitomizing a credit card for several undisclosed multi-national banks. Charlie seized this opportunity and infected all cards with the deadly anti-capitalist virus. Unfortunately due to a missing semicolon the virus instead of stopping the users from expenditure rendered any card holder uncontrollably shopping until they run out of money. Capitalism was safe for another day but a secret group of communist countries paid Charlie in Aldi vouchers and all their latest 486 computers he wanted. These computers would be delivered to a skip outside the Computer Science department where he was pretending to work on asynchronous technology while secretly working of the anti virus. The anti virus had to be written in a way that would not arouse suspicions from the capitalist mind-controlled co-workers so a programming systems that on the outside appears to be a set of web pages was developed. Using this programming system Charlie is able to write 1000 lines of code per day but make it look as if he was innocently browsing the web. Later he was joined by Chairman Miau to star in a low budget cop TV series.

Acknowledgements

I would like to thank Vladimir Ilyich, Joseph Vissarionovich, Lev Davidovich Bronstein, Ernesto Guevara, Nguyen Tat Thanh and Chairman Miau for inspiration.

Andrew Bardsley and Jim Garside for being excellent rebound boards for the many ideas I bothered them with but especially for teaching me all I know about async and somehow convincing me that this is an interesting subject after all.

Thank you to the whole Amulet group who have somehow managed to put up with my silly ways.

My partners Matthew (Fat Boy) Evans and Shuet-Ying Cheung (Sooty) for their support, love and understanding.

Bill Gates for his invention of the Intraweb without which I would be very bored indeed.

Mark Josephs in who's lecture I was "inspired" to invent this latch.

Chapter 1: Introduction

"If I had a million pounds I would be a student" Paul Capewell

1.1 Overview

1.1.1 Justification

For a number of years the VLSI community have been looking towards asynchronous logic to solve some of the problems that appear when using global clocks on very large circuits[1]. There are some adv antages inherent in asynchronous circuits above their synchronous counterparts: lower emissions of electromagnetic noise, no clock distribution, no clock skew, robustness to environmental veriations (e.g. temperature and power supply) or fabrication faults, better modularity and better security are just some of the properties where asynchronous designs have a definite advantage in. These properties are now widely accepted amongst the asynchronous community. Low power consumption [2], low latency and high throughput [3] are three properties which have been claimed but need to be specifically targeted in order to exploit them. It is important to distinguish the difference between throughput and latency rather than just calling them speed. The Amulet group has in the past created three low power microprocessors using low power asynchronous techniques [2][4]. Others have used fine grain pipelining to achieve high throughput at the cost of latency and power consumption[3]. By trying to exploit all three properties the final design will hold little if any advantage over the synchronous implementation. Alternatively by trying to exploit just one of these properties it is possible to gain it at cost of the others. Low latency can be achieved by exploiting the average case performance present in some asynchronous circuits. High throughput is present is asynchronous circuits with very high density pipelining, which is made difficult with global clock skew. The power consumption of synchronous circuits is often higher as the full global clock network has to be driven at a very high rate and many pipeline stages are executed where the result is not desired.

1.2 Synchronous logic

1.2.1 Synchronous logic construction

Synchronous circuits rely on external timing to determine the completion of each pipeline stage and D-type flip-flops to stop data from one stage overwriting the data in the next stage. In the figure of a synchronous pipeline (Figure 1.1) the clock net is connected to every flip-flop. As the clock ‘ticks’ the data changes from being the results of one stage to the inputs of the next. Figure 1.2 shows how data moves from one stage to the next by shifting all data to the next stage at the rise of the clock. Using a global clock ensures that the result will be correct by the time it is accepted by the next stage and a stage holds only one data entry.

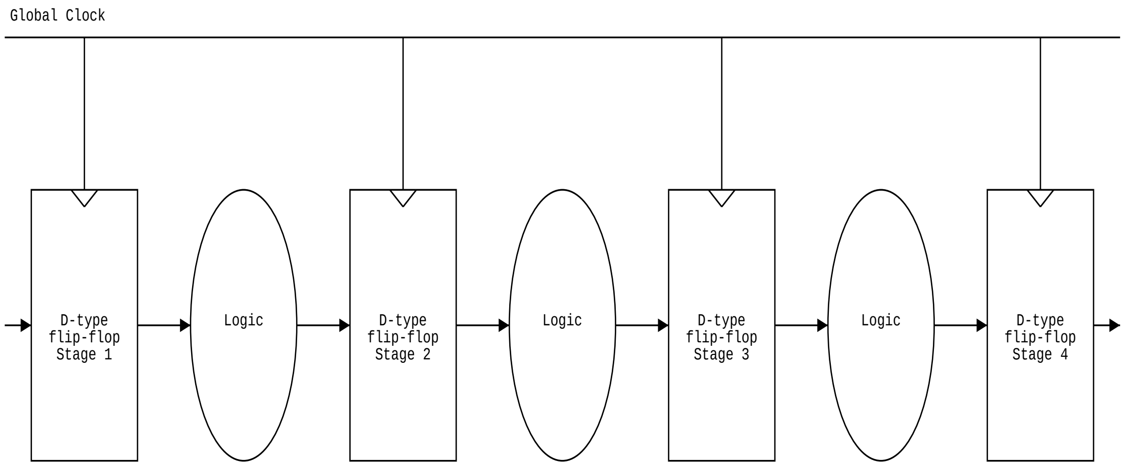

Figure 1.1: Synchronous pipeline

1.2.2 Synchronous pipeline properties

In figure 1.2 the shaded areas of each stage represent the stage having completed its logical operation, the result being valid but waiting for the clock before moving to the next stage. When ‘D0’ passes through Stage 1 its result is ready 0.25 of a clock cycle before the next clock edge arrives. During this time the data is unable to progress to the next stage. When ‘D0’ passes through Stage 3 it requires the entire clock cycle to perform its operation. This operation is known as the worst case pipeline stage as if the clock frequency was increased then the operation would fail because the result of the logical operation would not be ready in time to be accepted into the next latch. These operations may occur very rarely but they still force the clock to be slower to guarantee correct operation in this event. Additional to this performance hit the chip will run slower at a higher temperature or a lower voltage so these parameters have to be considered when choosing a clock speed. This approach gives worst case performance regardless of external conditions and operations executed.

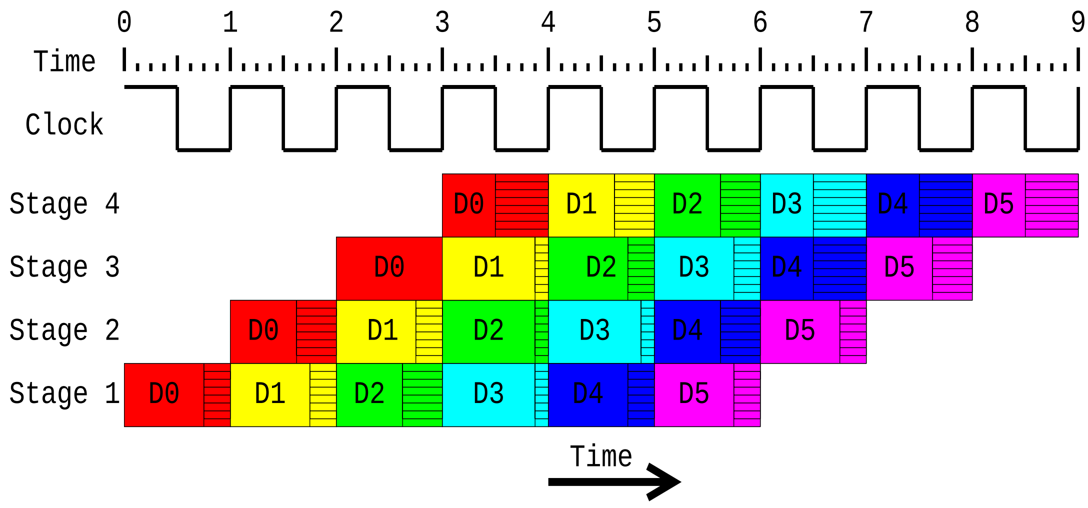

Figure 1.2: Synchronous pipeline occupancy diagram

1.3 Asynchronous circuits

1.3.1 What is asynchronous logic?

Asynchronous logic is a very broad term which can be used to describe any circuit which has the ability to keep and change state without the use of a global clock. This means that even if a chip has an internally generated clock signal it would still not be asynchronous.

1.3.2 Requirements of asynchronous circuits

As stated above the synchronous approach gives a timing which estimates the completion of a stage and the fact that the clock is global keeps data in separate stages. If these two properties can be reproduced without using a global clock it will allow the pipeline to execute faster than worst case performance. The stage completion can be derived in many ways. The easiest method is a matched delay; this is a line of gates that runs along with the data logic and matches the logic depth. When external variables sutch as temperature or voltage slow down the circuit this delay increases to allow the logic extra time to resolve the result. A more complex method is to use a data dependent matched delay. This method uses a several matched delay lines of which one is chosen depending on the data. For example if an ALU stage executed a fast, logical rather than a slow, arithmetic operation then a shorter delay would be chosen.

The most precise method of completion detection is not to use matched delays but use the logic to create a completion signal. The last two methods allow the data dependent speed improvements. Figure 1.3 shows an example of an asynchronous pipeline. The global clock is replaced with a set of asynchronous pipeline control elements. Once new data enters a stage the request signal is generated and on the wire labelled Req1 in figure 1.3. This signal goes through a matched delay or is combined with a completion detection signal and when the logic function has evalusted the request signal is emitted on wire Req2. The data is now ready to be accepted for use in the next stage.

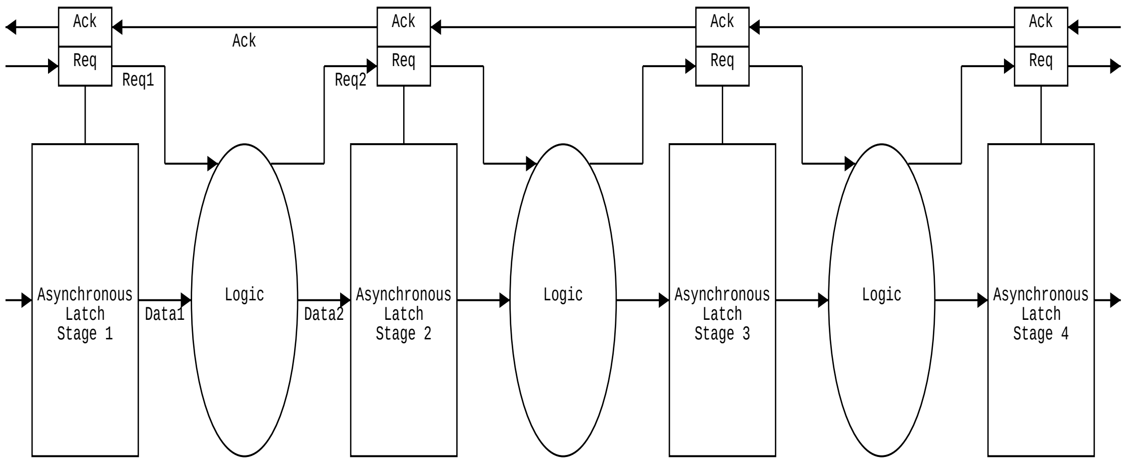
Ack
Req
Ack
Req1
Req2
Asynchronous Latch Stage 1
Data1
Logic
Data2
Asynchronous Latch Stage 2
Asynchronous Latch Stage 3
Asynchronous Latch Stage 4

Figure 1.3: Asynchronous pipeline

This approach solves the completion detection problem but there is still the problem of one piece of data over writing another piece of data in the next pipeline stage. To solve this problem an acknowledge signal (Ack) is sent back to the requesting control unit to inform it that it has accepted the data and the stage can be used for the next set of data. In turn the data that has been accepted is used in the next stage by emitting its request and the cycle then repeats in the next stage.

1.3.3 Properties of asynchronous pipelines

Figure 1.4 shows an asynchronous pipeline executing the same computation as the synchronous pipeline in figure 1.2. There are noticeable differences between the two diagrams. Firstly the asynchronous pipeline is faster as the optimizations described above are implemented. Unlike the synchronous pipeline there are two different types of stalls in the asynchronous pipeline both of which were dealt with simply by using a clock in the synchronous version. The first is demonstrated in stage 2 after D0 has moved to stage 3. Here the hardware is ready to accept new data but D1 has not completed ints function in stage 1. This is a 'hardware stall' as the hardware has to wait for the data to become available. In the figure this is demonstrated with the dashed lines across the stalling area. The second type of stall is shown where D2 is trying to move from stage 1 to stage 2 but the stage is not ready to accept new data as it is still processing D1. This causes a 'data stall' as the data is ready but has to wait for the hardware to become available. In the figure it is shown with dashed lines across the stalling area with the data shading still present. When the pipeline is too empty then hardware stalls are common and the throughput is low. When the pipeline is too full then data stalls appear more often and causes high latency. A balanced pipeline would have low latency and high throughput and so avoiding these stalls is important.

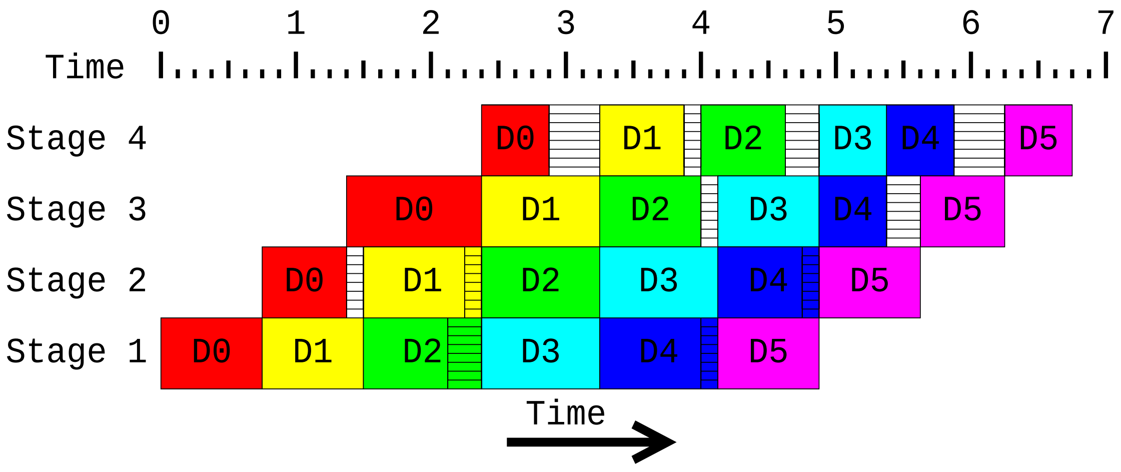

Figure 1.4: Asynchronous pipeline occupancy diagram

1.3.4 Asynchronous protocols

There are many asynchronous protocols but this thesis will concentrate on the four-phase early protocol [5] as shown in figure 1.5. In this protocol the sender asserts the request signal once the data is placed on the data bus. When the data is latched by the target it responds by rasing the acknowledge line. The sender drops its request and can now place new data on the bus. The target then drops its acknowledge when it is ready to accept new data.

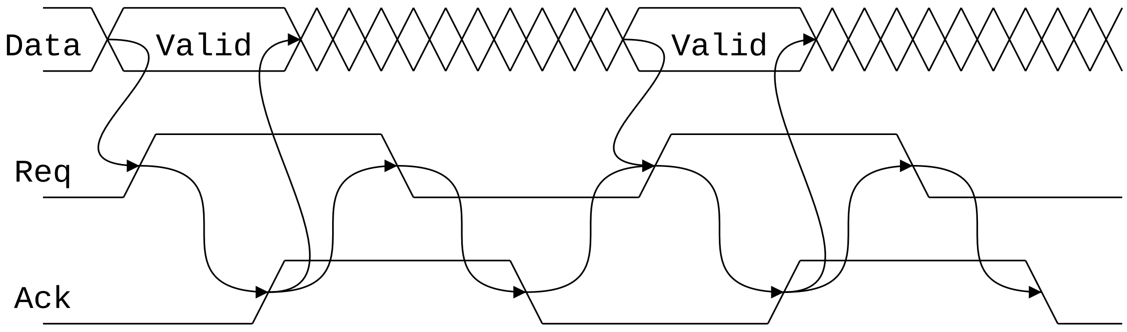

Figure 1.5: Four phase early protocol

Figure 1.5 shows the protocol without a matched delay on the request signal. Such a system is only useful for making a FIFO and cannot complete any computation. The request signal may be delayed to allow time for the computation. Figure 1.6 shows an asynchronous protocol with a delay on the request line[1]. Req1 reaches Req2 with a delay D. This delay should be equal or greater than the maximum delay of data through the logic function (Data1 to Data2). This delay can be asymmetric witch means Req1 low to Req2 low delay can be much shorter.

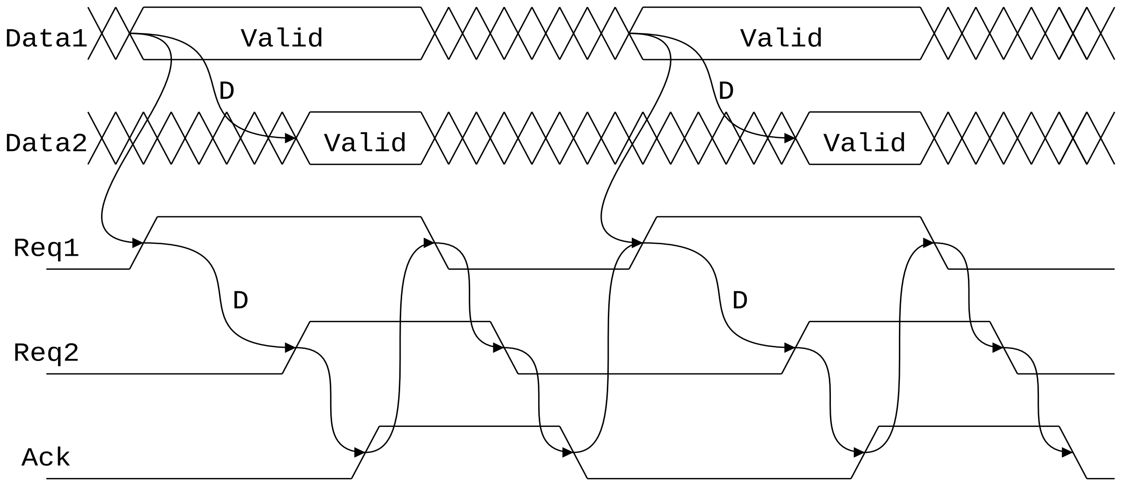

Figure 1.6: Asynchronous protocol with a delay

This is a bundled data protocol as the data is bundled with some control signals. The creation of asymmetric matched delays requires the designer to ensure that the delay line has a longer delay than the logic which could change at the implementation level. Trying to keep the request delay the same size as the data requires a lot of experience and very accurate back annotated timing models.

1.4 Dual-rail

1.4.1 Dual rail return to zero protocol

To bypass the delay matching problem another method can be used where the data is encoded in the request signal [6]. Dual rail data encoding, when combined with the early four phase protocol, solves this problem but uses two wires to represent one bit of data. If the sender wishes to send a 0 the it will assert the Data_0 line and if it wishes to send a 1 it will assert the Data_1 line. As in the bundled data protocol the target sends an acknowledge. This protocol is return to zero (RTZ) based as after the target has acknowledged the data lines should return to zero. This can be seen in figure 1.7.

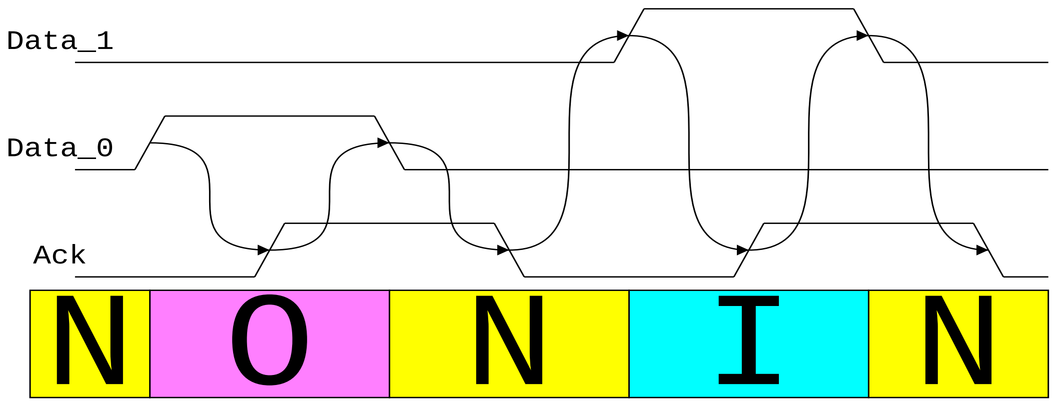

Figure 1.7: Dual-rail protocol

1.4.2 C-elements

The Muller C-element [6] is a very commonly used component when designing asynchronous circuits. Figure 1.8 shows the construction and symbol of a two input C-element. When all of the inputs into a C-element are high then the output will switch high. The output will stay high until all of the inputs are low again.

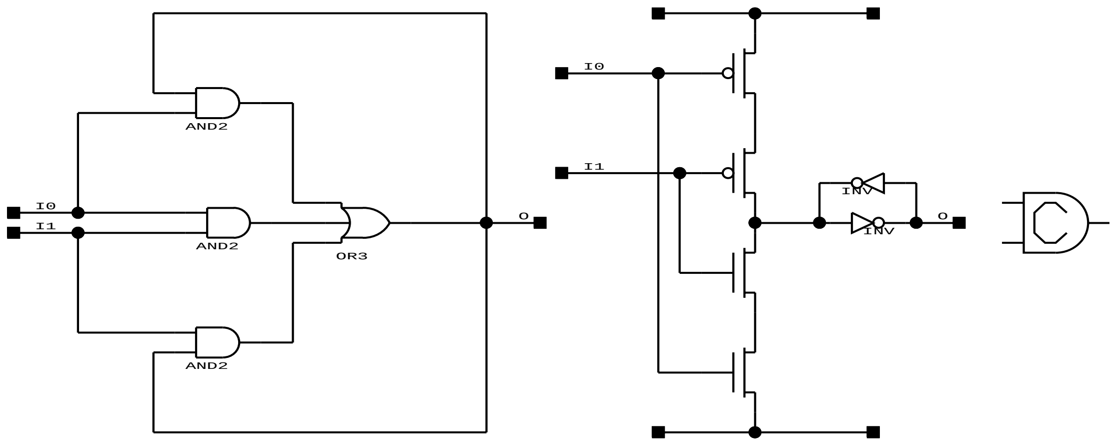

Figure 1.8: Gate and transistor level implementations and symbol of a C-element

1.4.3 Dual rail half latch

To create circuits using this protocol, gates and data storing elements are required. Figure 1.9 shows a dual rail half latch. This latch is used to store the data simmilarly to a master slave flip-flop in a synchronous circuit. If there is no acknowledgement on the output (Q_A is low) any data on the input will progress through the C-elements to the output. Once one of the data outputs is active the latch will acknowledge its input. The data output line will stay high until the target acknowledges and the source has returned to zero. Once the outputs have returned to zero both the acknowledge in and out lines will drop to allow the cycle to repeat. The acknowledge input wire (D_A) stays high while the latch is outputting valid data. Also while acknowledge in is high (Q_A) the latch cannot accept any data. If these latches are placed in a pipeline then the maximum occupancy would be 50% as for each latch that holds data, another separates the data with a null from other data in the pipeline. If arranged into a loop with X data tokens there would need to be more than twice as many half latches to keep the system from deadlocking. The reason for this is explained in secstion 1.4.5.

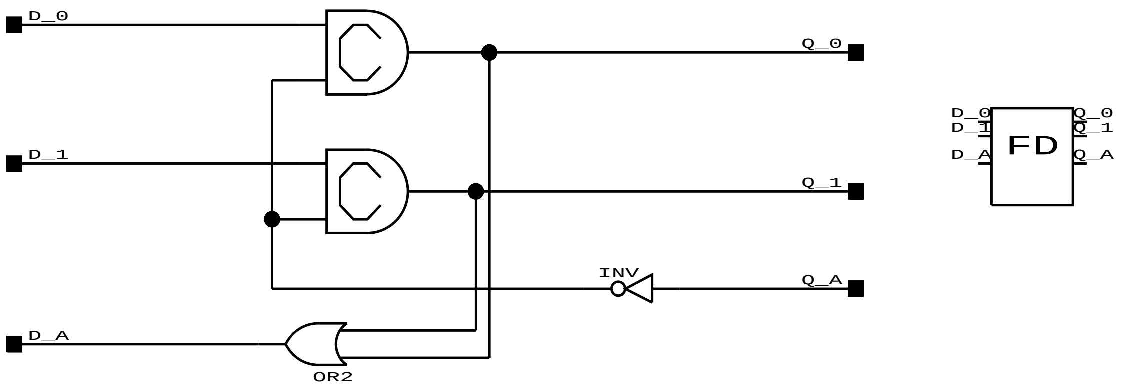

Figure 1.9: Dual rail half latch schematic and symbol

1.4.4 Return To Zero FIFO

Figure 1.10 shows data flowing through an RTZ FIFO. The RTZ protocol forces data to be followed with a return to zero which can be thought of as a null signal. In position 1 the pipeline is all reset to zero. There is a data value (O) entering the pipeline. As the first element contains the same value as the next element (they both contain null) the 'O' is allowed to propagate to this stage. At time 2 the 'O' has propagated into the first latch. It can now carry on propagating as again the next two latches hold the same values. At time 3 the 'O' has propagated to the next latch but left its value in the previous latch. This trail of values can be overwritten by a null entering the stage after the 'O'. The only value that cannot be over written is the leading value. For this reason these stages are shaded to show that they cannot be overwritten. The 'O' will carry on propagating until it meets the last null token. As the next two stages differ (null followed by a one) the zero is

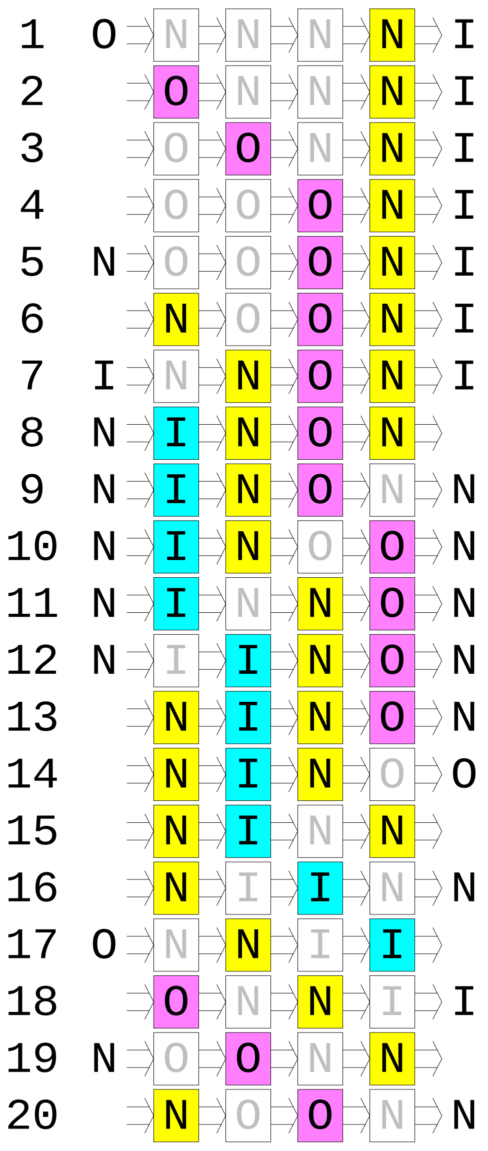

Figure 1.10: FIFO pipeline

blocked from entering the next stage. Any value (in dual rail 'I' or 'O') must be followed by a null. At time 5 the null is about to enter the pipeline. Again as before the same rules apply and it will propagate until it meets a leading value. The 'O' stops the null at time 7. When the 'I' enters the pipeline at time 8 the pipeline is full and cannot accept any more tokens. If one of the values is allowed to leave the pipeline at time 9 then a bubble is formed and each value shifts one forward moving the bubble backwards one space at a time. The bubble eventually allows a token to enter the pipeline at time 12. When the pipeline is very full, throughput becomes very low. The values are allowed to exit out of the pipeline at the maximum rate starting at time 14. The values separate themselves with a bubble when travelling at maximum speed which can be seen from time 15 to time 20. Talk about tokens!!!

1.4.5 Minimum RTZ pipeline loop

A loop containing a number of data items must have two times this number of latches in order to hold all data and null signals in separate latches [1]. The first diagram in figure 1.11 shows a loop of two latches and one data signal. Neither the data nor the null can move as the places they want to move to are occupied by each other. A situation where a circuit cannot move from one state to another irrespective of any inputs is called a deadlock. In the loop of three latches the data is able to move freely. Once the one moves to the next stage it leaves a bubble for the null to move to. The general rule is for a loop with X pieces of data at least 2X+1 latches are required to avoid deadlock.

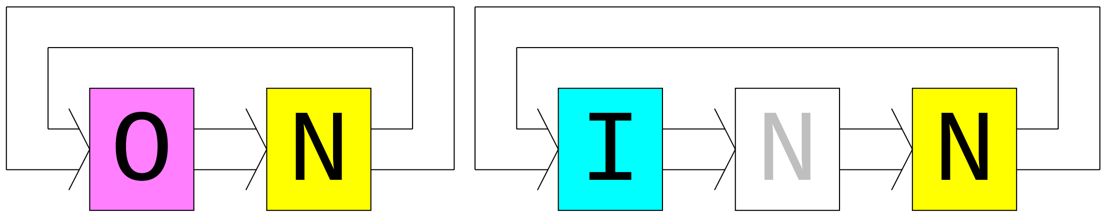

Figure 1.11: Two and three latch RTZ pipeline loops

1.4.6 RTZ pipeline properties

The above examples show many important properties about RTZ pipelines.

- One signal will not be overwritten by another coming in behind it.

- The maximum occupation of a pipeline is one signal per latch. Each piece of data consists of a data signal and a null signal so maximum data occupancy is 50%.
- An over-saturated pipeline has a very slow throughput.
- Highest throughput pipeline will have a bubble between all signals which gives 25% data occupancy.
- For a pipeline loop containing X data elements at least 2X+1 latches are needed for the loop not to deadlock.

1.4.7 Dual rail gates

The RTZ protocol needs to separate its data signals with null signals. When creating logic using this protocol the output should only become valid when all the inputs are valid. Also the output should remain valid until all inputs have returned to zero. Figure 1.12 shows the construction of a dual rail AND gate [6]. The row of C-elements forces both of the inputs to switch before the output switches. The return to zero protocol assumes that once a gate outputs a valid signal then all its inputs are valid. Also the data will not return to zero until all inputs have returned to zero.

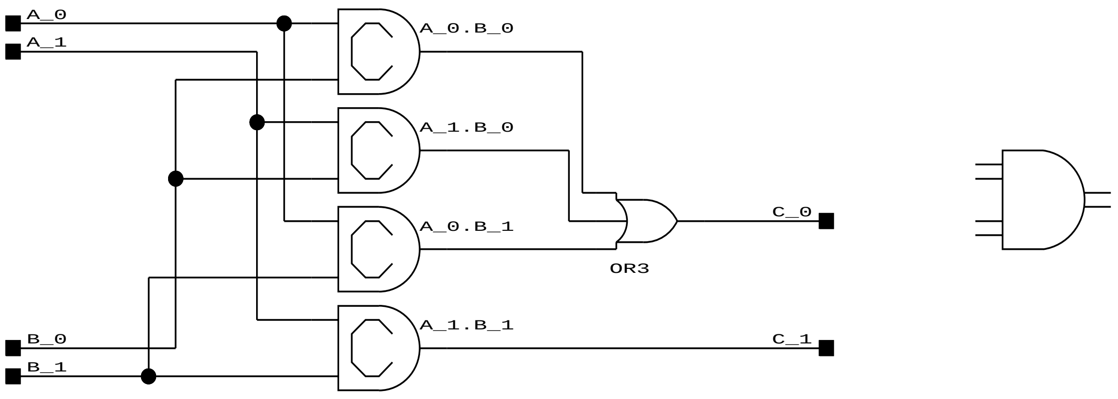

Figure 1.12: Dual rail AND gate schematic and symbol

1.4.8 Acknowledge circuits

Talk about why guarding is required

1.4.9 Circuit reset

At reset time the circuit has to be primed for execution. Before the reset signal is applied the latches can hold random data and gates may have some of their C-elements set. In order to reset the circuit so it contains no tokens a reset line is driven. One approach is to attach a reset line to all C-elements and thus forcing all nets in the circuit to reset. If all inputs to a dual-rail gate are low then the output will switch low. Using this assumption it is possible to reset the whole circuit to just contain nulls by just resetting all latches. There are two types of resettable latches. Firstly the hard reset latches make no assumptions about the inputs and attach a reset line directly to the C-elements. This allows the latch to be reset irrespective of the inputs but is slower than the soft reset approach. A soft reset latch assumes that the inputs are low and the reset line is simply combined with the acknowledge line using an OR gate. The latch will observe an an acknowledge and will remove its data from its output if its input is also low. A hard reset latch is guaranteed to reset with its inputs in any state. Soft reset latches and gates are guaranteed to reset if their all inputs are hard reset latches or reset guaranteed components.

Chapter 2: Standard dual rail circuit construction

“Where do we eat? I am hungry like a wolf!” Tomaz Felicijan

2.1 Direct translation

2.1.1 Overview

To quickly implement dual-rail circuits a tool called direct translation will be used to provide a higher level of abstaraction. This chapter will explain its operation and issues arising when implementing large asynchronous circuits.

2.1.2 Input design

Direct translation works on the principle of using single wires to represent communications chanels. The input design thus looks very similar to a standard synchronous schematic with the assumption that all flip-flops are clocked using a single global clock. To demonstrate direct translation a simple design will be converted. Figure 2.1 shows an abstracted design.

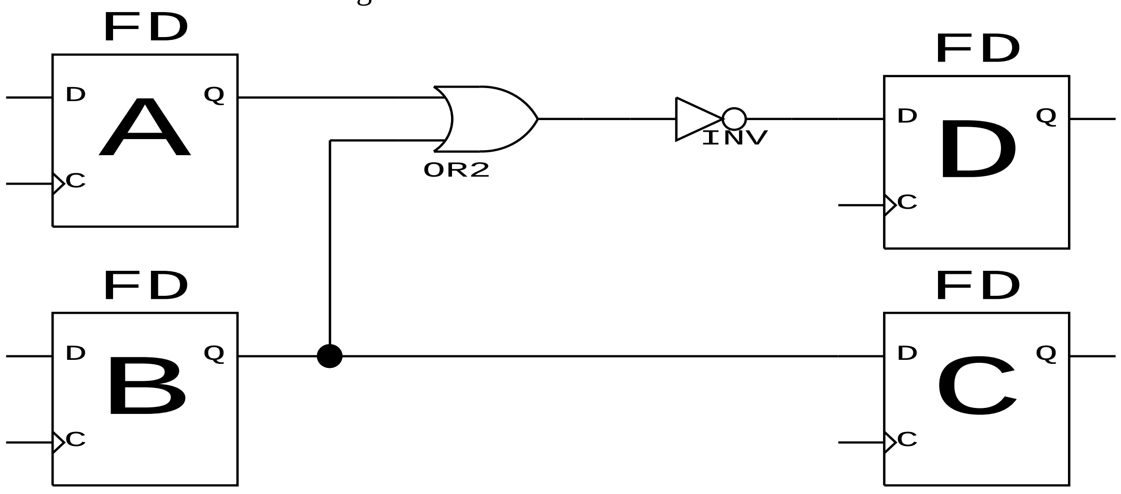

Figure 2.1: Synchronous design

2.1.3 Direct translation rules

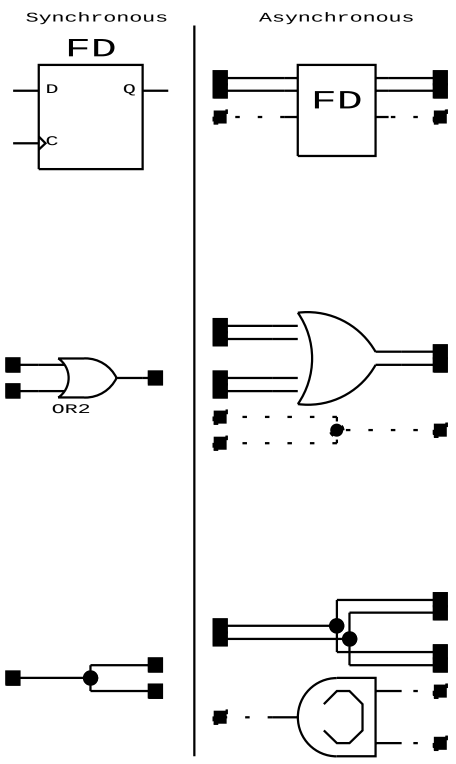

Figure 2.2: Translation table

Direct translation is intrinsically a simple process. The tool simply has to replace all instances of components and certain structures with their asynchronous counterparts. The rules used for the standard direct translation are shown in figure 2.2. Acknowledge paths are distinguished by having dashed lines.

Firstly all gates and latches need to be exchanged for their asynchronous counterparts. Instead of one wire linking components this method needs three. Only one example of a gate is shown but any number input gates can be implemented in this approach simply by distributing the acknowledge signal to all inputs. More difficult to find are instances of forks which have to combine the acknowledge signals using C-elements.

2.1.4 Resultant asynchronous circuit

Figure 2.3 shows the resultant circuit. The four boxes show instances where a rule was applied. Box A shows where a flip-flop was replaced. Box B shows the replacement of the OR gate which is replaced with a dual rail OR gate with the acknowledge signal passed back to both inputs. Box C is a replacement for the inverter which has only one input and so only needs to send an acknowledge to its one input. Box D shows the replacement for the fork and it places a C-element to combine the acknowledge signals.

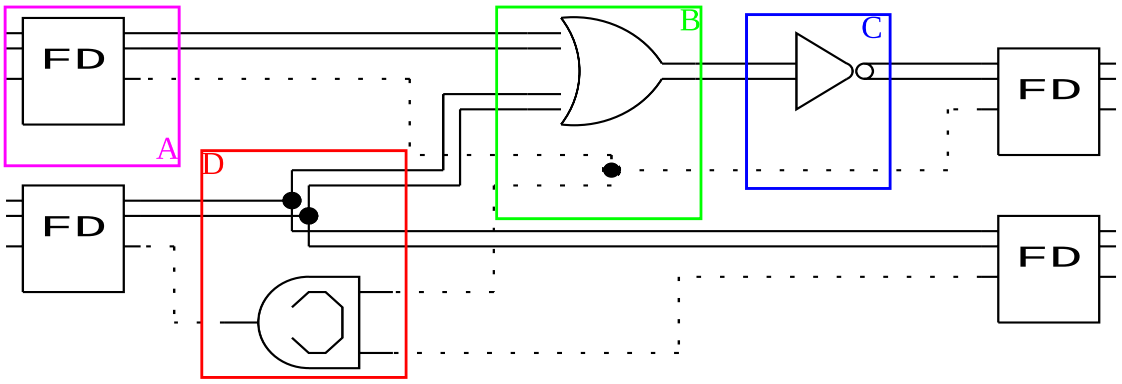

Figure 2.3: Asynchronous circuit implementation

2.1.5 Three stage counter

Consider if in figure 2.1 flip-flop A was the same as C and B was D. Now B outputs to A and A combined with B outputs to B. Figure 2.4 shows the translated asynchronous version of this circuit. It is important to note that the bottom latch outputs through a gate back to itself making a very tight loop. The fact that a this pipeline stage feeds its outputs back to its inputs does not affect the direct translation.

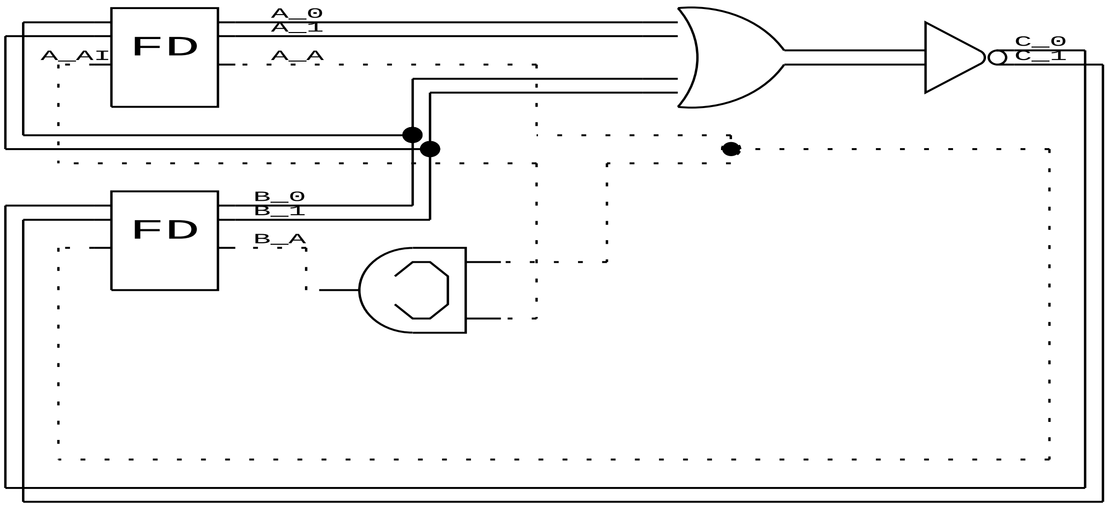

Figure 2.4: Asynchronous three stage counter

2.1.6 Flip-flop replacement

As mentioned in section 1.4.5 the minimum number of latches for a loop with one item of data is three. Because the user could wire the a output of a flip-flop back to its input (as demonstrated in figure 2.4) the replacement must consist of at least three half latches.

Even if this is done there is still no data in the pipeline and so the circuit will not operate. In order for the circuit to start there must be some data tokens placed into the circuit at reset time. A token should be placed in all flip-flop replacements. The flip-flop counterpart is made from three latches. One is a hard reset latch as shown in figure 2.5. This latch makes no assumptions on its inputs. At reset time one of the C-elements is reset while the other is set by the global reset signal labelled “GSR”. To allow any soft reset latches or gates driven by the output of this latch to reset the zero output is gated with the global reset signal. The other two latches in the flip-flop counterpart are standard soft reset latches. This D-type flip-flop counterpart is will at reset hold a zero token but there is another version which will hold a one token at reset.

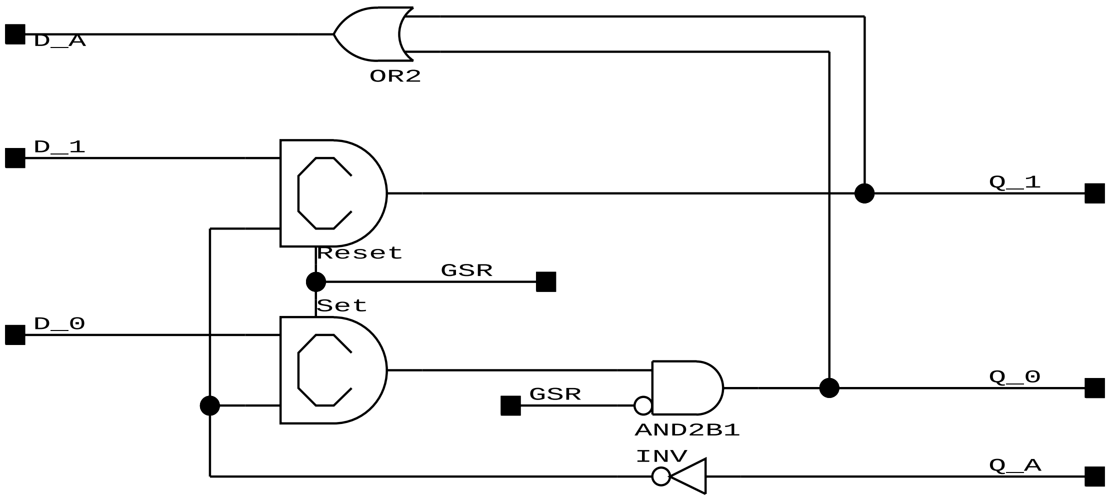

Figure 2.5: Hard reset zero initiating latch

2.2 Results

2.2.1 Simulation

Figure 2.6 shows the simulation of the three stage counter. The nine waves represent the nine labelled wires in figure 2.4. Dual rail net pairs A, B and C go through three stages and each cycle a different pair contains the ‘I’ while the other two contain ‘O’. These values are marked on the graph. The circuit’s functional behaviour is as expected. There are visible differences in the arrival time of the signals. A arrives first as its input is very simple to calculate. B arrives second and lastly C is formed only after A and B have become valid.

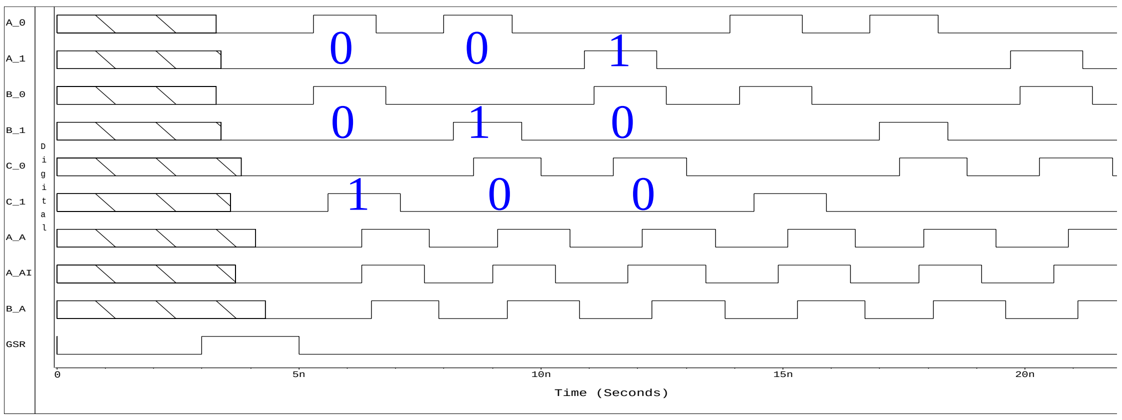

Figure 2.6: Simulation of the three stage counter

2.2.2 Simulation explanation

Figure 2.7 shows a more detailed wave trace. The sequence is started by A and B outputting data values. After both of these transitions have occurred the gates will produce a data value on bus C. Bus B, which outputs directly to a latch, is acknowledged on net A_AI. A data value on bus C causes the latch to acknowledge on net A_A. Both of the latches which bus B feeds have acknowledged so the C-element driving net B_A becomes active. As bus A is only used to generate the value which has just been acknowledged with A_A it will return the bus to null. Bus B is also acknowledged by B_A and returns to null. As both of the inputs to the gate have returned to zero the gate will also drop its output. Both buses feeding the latches (B and C) have returned to zero so the latches drop their acknowledge signals (A_A and A_AI). B_A drops soon after to allow bus B to drive a new data value.

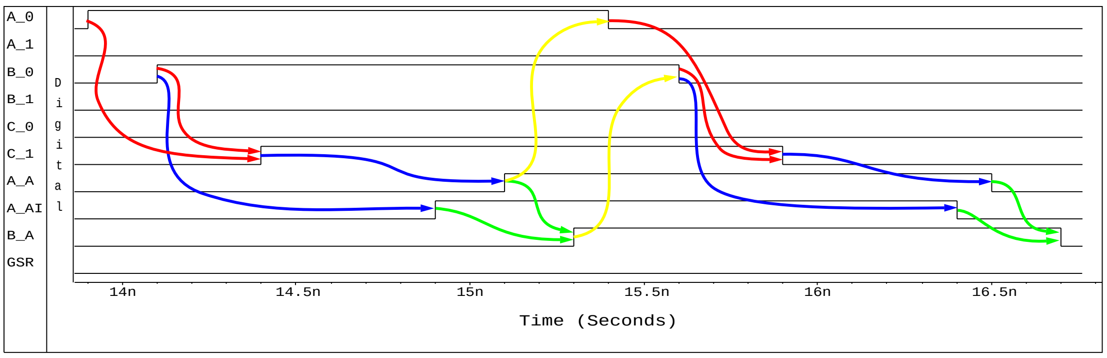

Figure 2.7: Close-up of one data phase

2.3 Summary

2.3.1 Results

In the Virtex library each gate has a delay of 0.1 ns (100 ps). Each cycle takes 4 ns to complete. These times are very slow and will get much slower with introduction of more complex logic as each gate is 3 times slower (delay of a C-element and a gate) than a conventional gate used in the synchronous version. The next chapter will detail a different approach which hopes to alleviate this problem.

Chapter 3: Early output dual rail circuit construction

3.1 Introduction

3.1.1 Motivation

As shown in the previous chapter the direct translation method can be used to convert synchronous circuits to dual rail asynchronous versions. The greatest weakness of the standard dual rail approach is the construction of gates which require a large number of C-elements making them very large and slow. This chapter will show an alternative selection of direct translation rules and elements to bypass these problems.

3.1.2 Overview

As described in section 1.4.8 the C-elements in all multiple input gates are needed to ensure the output does not switch before all inputs have arrived or left. Allowing a gate output to become valid before all inputs have become valid will cause a latch the gate outputs to acknowledge latches which have not output a data signal. Raising the acknowledge line to a latch which has not placed a valid value on its output is not permitted by the protocol. This chapter shows how, with the use of guarding C-elements, gates can output early yet still obay the four phase protocol.

3.2 Early output dual rail circuits

3.2.1 Early output gates

A gate is desired which will output a valid signal once it has enough valid inputs to be sure of the result. In the case of an OR gate, if either of the inputs is one then the output will be one. If one of the inputs is zero then the output cannot be determined until the second input hes become valid. Only when both of the inputs are zero can the output safely switch to zero. Figure 3.1 shows a schematic and symbol of an early output dual rail AND gate. The zero output (C_0) will become valid if either of the inputs is zero (A_0 or B_0). For one to be output both inputs have to be one. Another restriction which is still true, as in the standard dual rail gates, is that the output cannot drop while either of the inputs is still valid. For this reason output one (C_1) has to be kept high even when one of the inputs returns to null. This is done by using a C-element to create the one output.

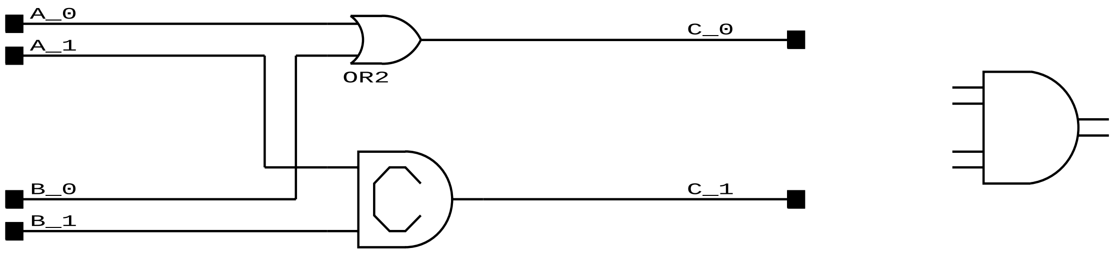

Figure 3.1: Early output gate

3.2.2 Early output latches

In early output circuits the result may be created before all inputs to a logic block are valid. Once the result is valid the latch will acknowledge. This acknowledge signal could be sent to some latches which have still not output valid data. This acknowledge signal might not be seen by the latches or could cause a glitch to enter the logic block. To ensure the acknowledge signal does not reach the input latches before they are valid it is combined with the validity of all input latches. Each latch creates a validity signal by using an OR gate across the two output lines. There is already an OR gate with its inputs attached to the outputs of the latch to create the acknowledge signal. Figure 3.2 shows the construction of an early output latch. The only differences between this latch and the one used in the standard dual rail is an internal acknowledge wire being output for use as a valid signal.

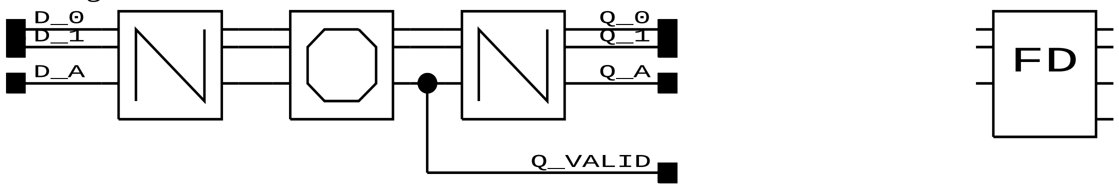

Figure 3.2: Early output latch

3.2.3 Early output translation rules

Figure 3.3 shows the rules for creating early output designs. The difference between these and the standard dual rail rules is the presence of the validity nets. At every instance of a gate the validity lines of all inputs are combined using a C-element to create the validity of the output. All latches now combine their acknowledge with their incoming validity to ensure the valid acknowledge line does not raise until all inputs are valid.

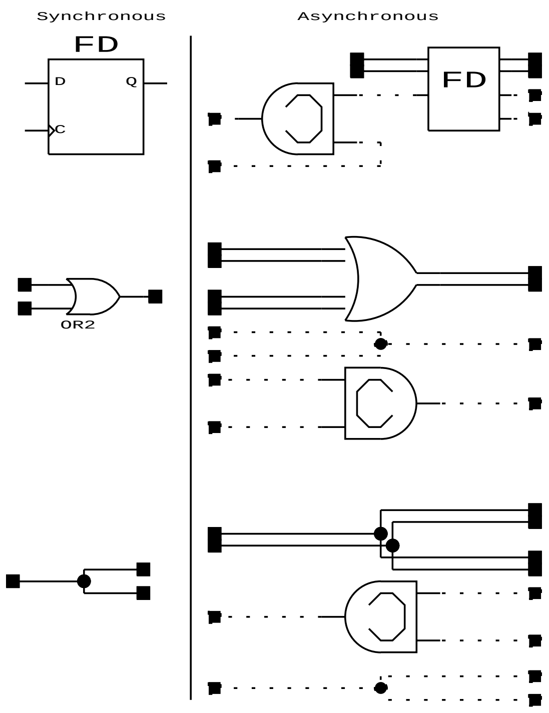

Figure 3.3: Early output translation table

3.2.4 Acknowledge paths

Acknowledge paths in early output systems are similar to standard dual rail ones. The main difference is that the validity of all inputting latches is combined with the acknowledge of the latch using C-elements. Figure 3.4 shows an early output multiplexer. If A outputs one, S outputs zero and B outputs NULL then the logic block will create a valid result (one). Even if latch C acknowledges the data the acknowledge signal will not reach the latches until all have become valid. This validity is checked by placing a C-element combining the validity of all inputs along side every gate.

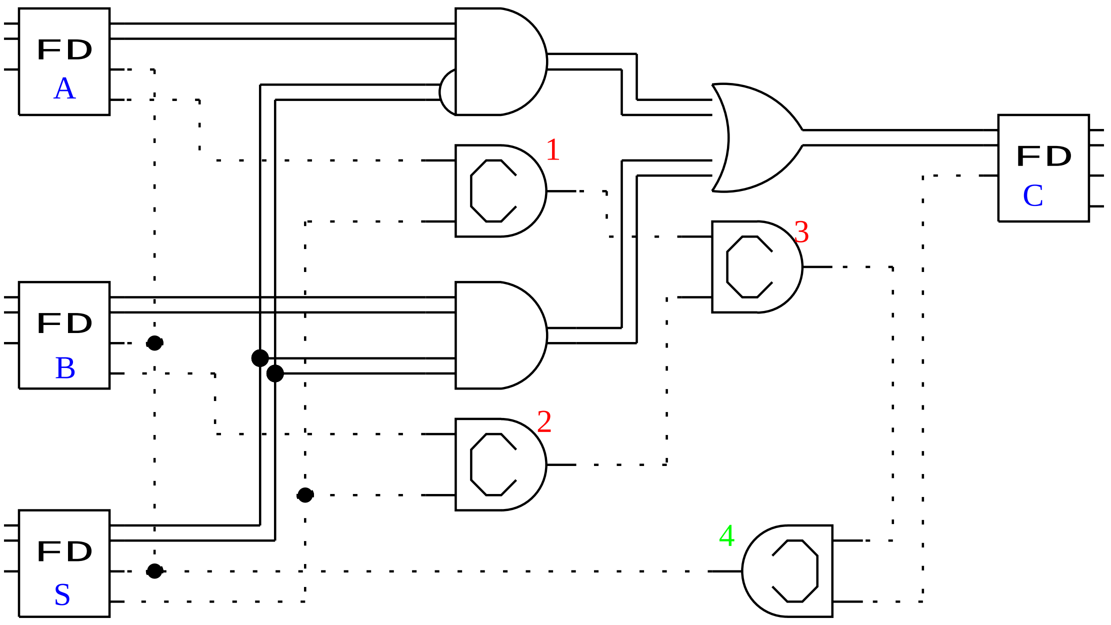

Figure 3.4: Early output multiplexer

3.3 Optimization

3.3.1 Validity C-element flattening

In figure 3.4 C-elements 1 to 3 create the valid signal which is combined with the acknowledge signal in C-element 4 to create the validated acknowledge. Elements 1-4 can be flattened down to a single four input C-element. This increases speed and decreases size of the circuit. Figure 3.5 shows the only the acknowledge circuit of a different logical operation. The solid wires are the validity part of the acknowledge paths while the dotted lines are the acknowledge nets. Unfortunately there is a fork coming out of C-element 2 and now there are two approaches to flatten the acknowledge paths.

3.3.2 Flatten around the fork

The first approach is to keep the fork in the validity tree and flatten C-elements either side of it. This would flatten C-elements 1 and 2 together and 3 and 4 in separate C-elements. This approach will keep the circuit small but also make the C-element tree more than one level deep and so make the validity path slower than the next approach.

3.3.3 Duplicate common parts

The second approach duplicates the common parts of the trees and flattens then individually. The first validity C-element will be a combination 1,2, 3 and 4. The second will be the combination of 1, 2 and 5. This method will make all validity paths only one deep at the expense of size as now a 5 input and a 4 input C-elements are needed. This is bigger than a 3 input and two 2 input C-elements.

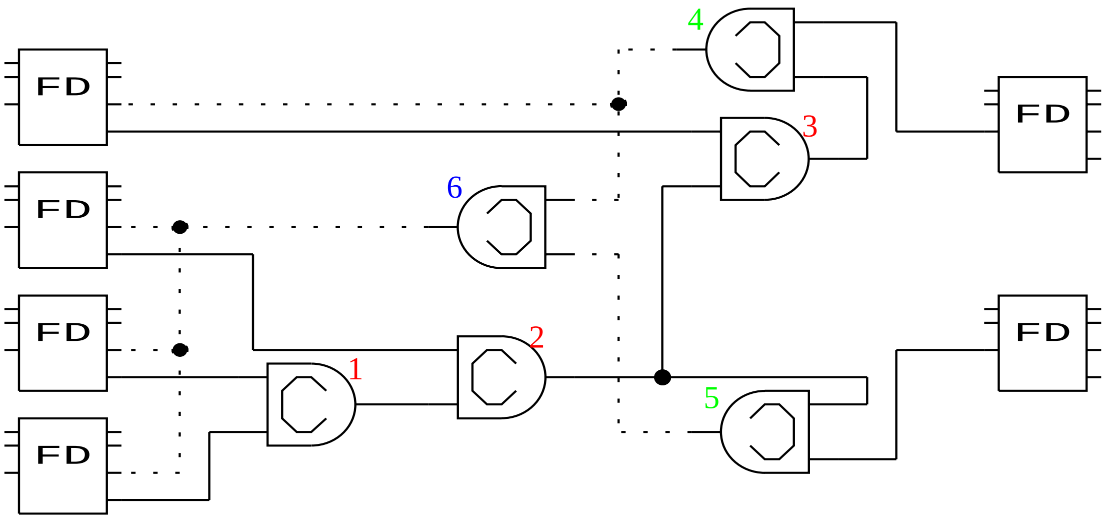

Figure 3.5: Example early output acknowledge paths

3.3.4 Acknowledge tree flattening

The acknowledge tree flattening can be conducted in exactly the same manner in both standard dual rail or early output acknowledge circuits. By either flattening around forks to save space or duplicating common parts to create fast circuits. It is important not to flatten both of the validity and the acknowledge trees together unless certain preconditions are met as this may result in malfunctioning circuits.

3.3.5 Hmmm. What happens if you flatten both trees together?

The acknowledge tree (6 in figure 3.5) should be flattened separately from the validity tree (1-5 in figure 3.5). Figure 3.6 has an example synchronous circuit and its asynchronous counterpart before C-element flattening optimisation. C-elements 1 and 2 are the validating elements and which can be safely combined with 3 and 4 respectively. C-element 5 is part of the acknowledge tree and should not be combined with 3 and 4 by using the "duplicate common parts" strategy.

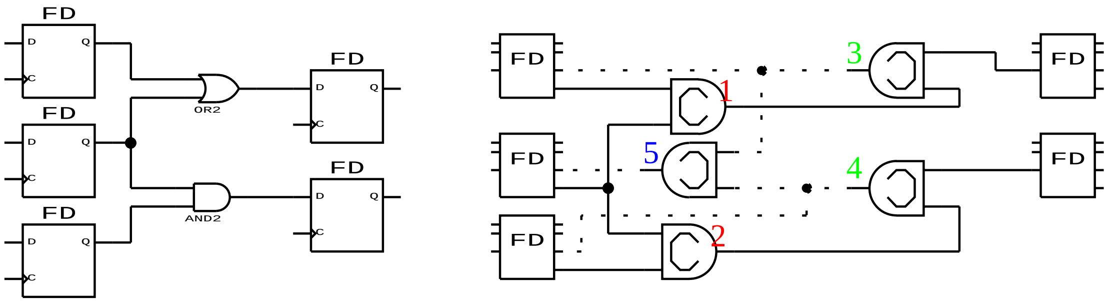

Figure 3.6: Synchronous circuit and its asynchronous acknowledge paths

Figure 3.7 shows a correctly and an incorrectly flattened circuit. The first circuit was flattened but did not replicate C-elements 3 or 4. This circuit will function correctly. The second circuit has duplicated and flattened the C-elements 1-4 and combined them with C-element 5 to create the middle C-element. The fault lies in the fact that the C-elements 3 and 4 have been duplicated. This allows one of these to switch and complete the whole cycle while the other is still waiting for the last input. Considering a situation where data has arrived on latches A and B but not on C. Latch X has received a valid result but latch Y is still waiting for input C. In this situation latch A will be acknowledged as it has all inputs it requires to return to zero (X ack, A valid and B valid). Latch B is still waiting for C valid and Y ack. Latch A will return to zero (A valid drops) and when value C arrives latch B cannot reset as it will be waiting for A valid which will never come. To avoid this error the function completion C-element should never be duplicated. The function completion C-element is one which takes the acknowledge signal from the latch and combines it with the validity signal from the validity tree

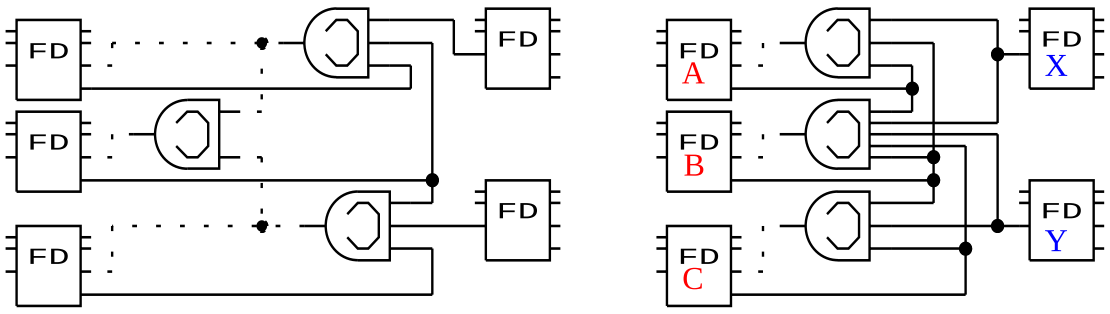

Figure 3.7: Correctly and incorrectly flattened acknowledge circuits

3.3.6 What does direct translation do?

Direct translation aims to make the fastest circuit and always flattens the validity tree along with the function completion C-element into one C-element and the acknowledge tree into a separate C-element.

Chapter 4: Early output protocol extensions

4.1 Early output review

4.1.1 Overview

The previous chapter demonstrated the construction of early output circuits. This chapter will give a more detailed analysis of the protocol. It is imprtant to ensure all components obey their protocol and do not exibit hazardous behaviour. All the protocols described in this chapter are based on the dual-rail, four-phase, return to zero protocol introduced earlier.

4.1.2 Logic

In a standard logic family it is often possible to determine the output of a gate before all its inputs are valid. This a property of all gates with the exception of XOR and XNOR. Early output gates take advantage of this and output as soon as the result is determined. An example of the differences in the behaveour is demonstrated in table 4.1. The table shows the outputs of the standard (ST) and early output (EO) dual rail logic OR gates.

A	B	ST	EO
0	0	0	0
N	0	N	N
1	0	1	1
0	N	N	N
N	N	N	N
1	N	N	1
0	1	1	1
N	1	N	1
1	1	1	1

Table 4.1: Truth table for standard and early output dual rail OR gates

The standard gates will only output once all inputs are valid. This can be seen in the table where if either of the inputs are NULL the output is NULL. The early output OR gate will output as soon as the result can be determined. In the table this can be seen if one of the inputs is NULL and the other is 1. The standard gate will wait for the second input but the early output gate will output a 1.

4.1.3 Reset circuit

As a valid value can be output form a gate without the need of the second input the acknowledge signal must be delayed untill the late input arrives. In early output circuits this is done with the use of the valid signal. Validity circuit ensures that all inputs are low and when combined with the output latch acknowledge it is possible to ensure that outputs are low too. Only when the circuit is fully reset to zero can a new wave of data enter the circuit. The fact that all inputs and outputs are low does not mean that all nets in the circuit are low too. The circuit can exibit some short term hysteresis as the prevous signals have not

4.1.4 Sender protocol

Although still ‘four-phase’, the early output protocol adds a validity net. Figure 4.1 shows the early output protocol from the perspective of the sender. In the figure the striped arrows signify data-dependent causality. This happens when the data being transmitted is not required to create a result from the combinatorial logic. The reason for the validity net is to protect the sending latch from receiving a transition on the acknowledge line before it creates a transition on the data output.

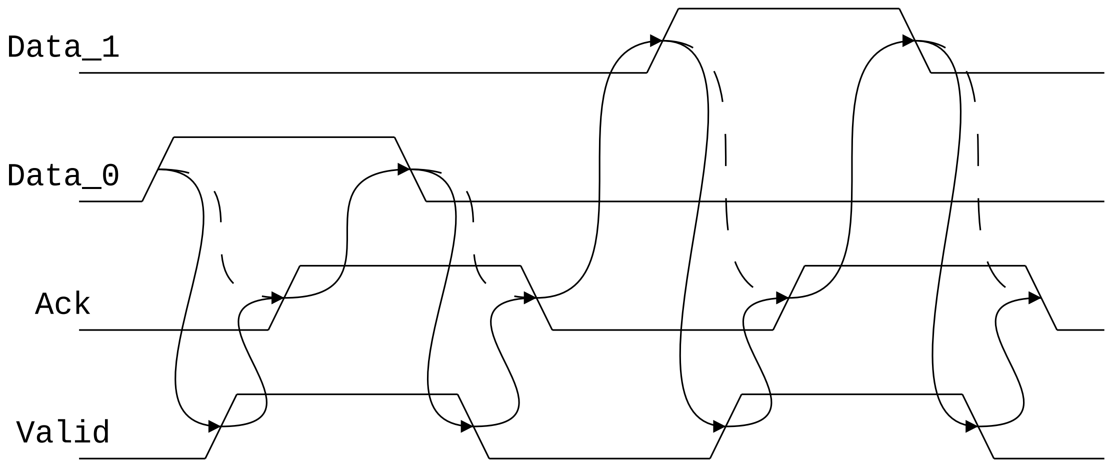

Figure 4.1: Early output sender protocol

4.1.5 Receiver protocol

The reason that the receiver protocol is different than that of the sender is simply because two latches do not talkt to each other directly. Some signals pass through the validity and acknowledge C-elements before reaching the other latch. The receiver protocol is an extension of the standard protocol. The acknowledge from the receiving latch is not passed directly back to the input latches but is combined with the validity net using a C-element. Combining the two signals using a C-element forces a transition to occur on both the validity input and data acknowledge from the latch before a valid acknowledge transition is sent back to the input latches. Unlike in the sender protocol there is no data dependent causality on any wires.

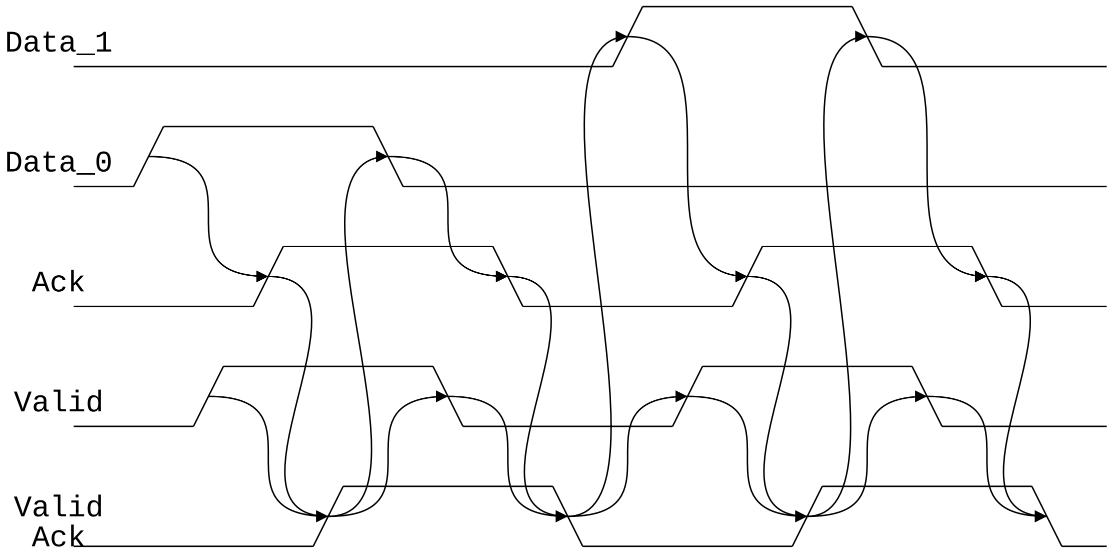

Figure 4.2: Early output receiver protocol

4.2 Comparison of behaviour

To demonstrate the behaviour of the different protocols a simple token based circuit shown in figure 4.3 will be used. The test consists of having a token passed down one of the channels entering an OR gate. Different protocols and latch designs will demonstrate the different points where the circuit stops and unable to proceed without the second token.

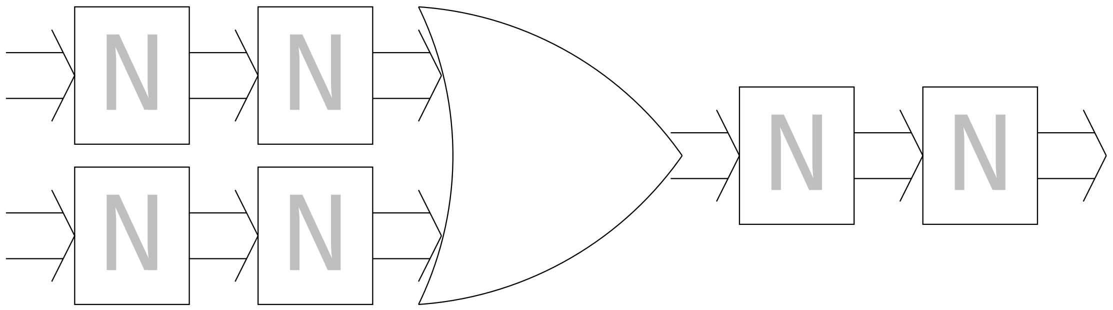

Figure 4.3: Small token circuit example

4.2.1 Standard dual rail

Figure 4.4 shows a logical diferal situation often arising in the standard dual rail design. The value entering the OR gate is sufficient to calculate the result of the logical function but it will not switch until the second value enters it. This is because OR gate is made with guarding C-elements which prevent the output from rising before all inputs are valid. This design method causes many inflexible synchronisations.

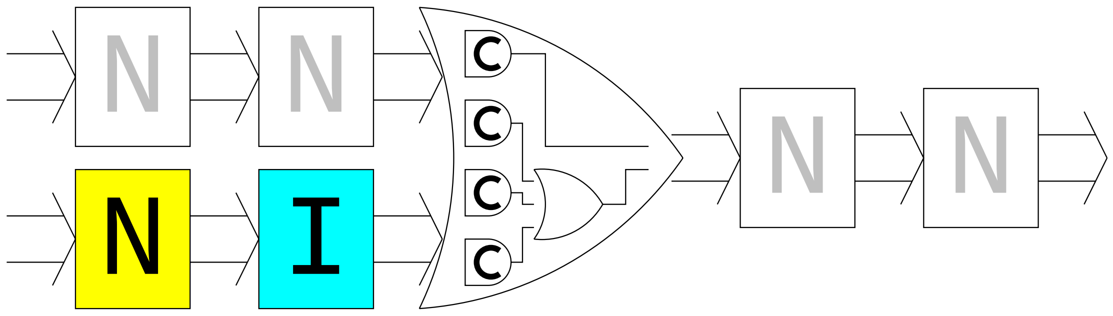

Figure 4.4: Standard dual rail logical diferal

4.2.2 Early output

The early output logic protocol allows the OR gate to switch once it is sure of the result. The data can then flow through the subsequent latches as shown in figure 4.5. The latch accepting the result of the pipeline stage will now acknowledge, but the acknowledge signal will not be passed to the input latches until all the inputs are valid. The stage with the OR gate and any subsequent stages will be stuck in the data phase and will not be able to reset and start working on the next set of inputs. Once the second input enters the stage and the validity signal rises then the stage can complete and enter the reset phase.

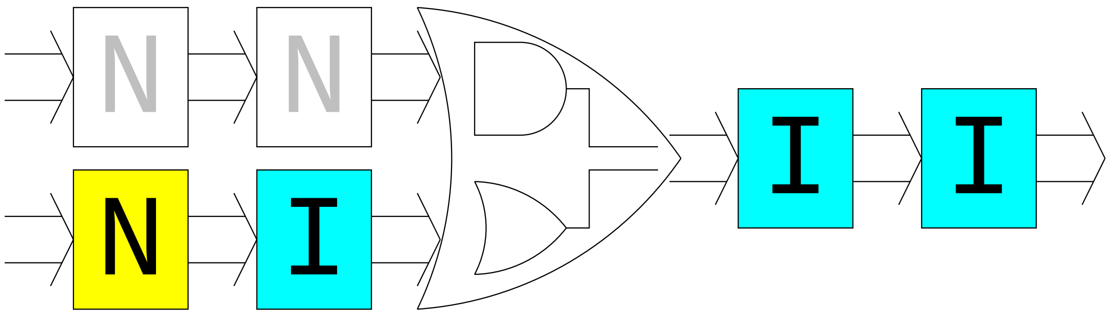

Figure 4.5: Early output circuit stuck in data phase

4.3 Semi-decoupled latch construction

A latch is required which if placed at the output of the OR gate when acknowledged would send a null and remember to wait for the input to drop before propagating another token to the output. This would decouple the output from the input (but not the input from the output thus the name ‘semi-decoupled’). The stages after the semi-decoupled latch will be able to reset and start processing the next set of inputs.

4.3.1 Standard half latch

A semi-decoupled latch can be designed by amending the design for the half-latch described and used in the previous chapters. Figure 4.6 shows a specification of the half latch in an STG graph and a schematic created with the output from Petrify. The Ro and Ri signals have only one instance in the STG but two in the schematic. This is done to keep the STG design process simple. They are simply duplicated in the schematic design. The STG is composed of two interlinked loops which follow the return to zero protocol. The first is the input cycle Ri+ → Ai- → Ri+ → Ai-. The second is the output loop Ro+ → Ao+ → Ro- → Ao-. The loops are linked with several constraints. Firstly Ri+ → Ro+ ensures the output will come only after the input. There are two interlocks to synchronise the input handshake with the output handshake to lower the token storage requirements. These are Ao- → Ai+ and Ao+ → Ai- and in the case of a half latch the token capacity is half. There is one more link which is only used to help with the state coding and to stop a “marking exceeds the capacity for place <Ao-,Ai+>” error. The Ri- → Ro- link also stops the output stage from returning to zero while the input is high.

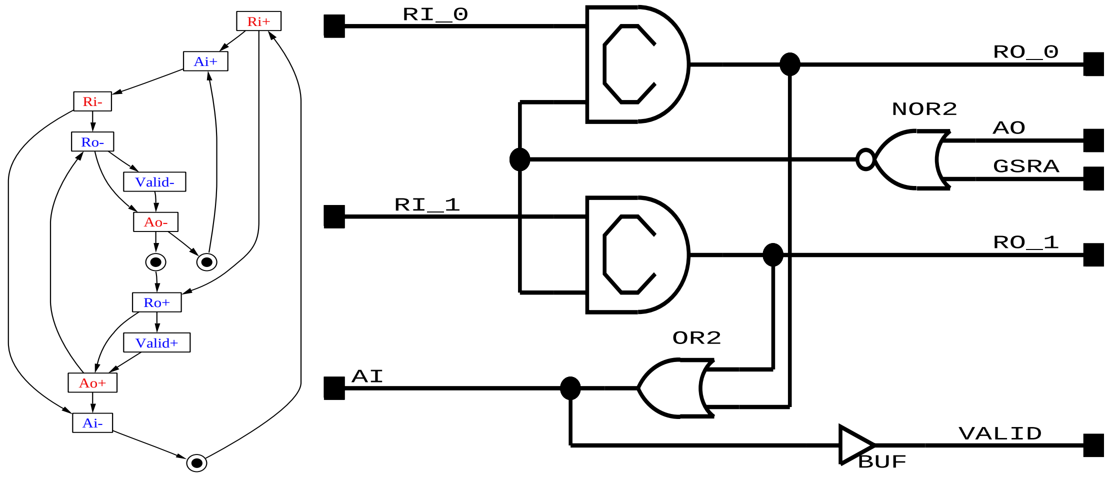

Figure 4.6: Standard half buffer STG and schematic

4.3.2 Standard semi-decoupled latch

Creating a semi-decoupled latch is not as simple as removing the Ri- → Ro- link in the STG description. As mentioned above, removing the link will cause a “marking exceeds capacity error”. To solve the problem an internal state holding signal was created. The ‘csc’ signal was placed in the output loop to separate the Ao and Ro transitions to form a Ro+ → Ao+ → csc+ → Ro- → Ao- → csc- loop. The Ri+ → Ro+ link still exists but the

Ri- ⟶ Ro- link can now be removed. Again some extra links were placed to keep the two sides synchronised. Figure 4.7 shows the final STG specification and the schematic created when the design was passed through Petrify and hand optimised.

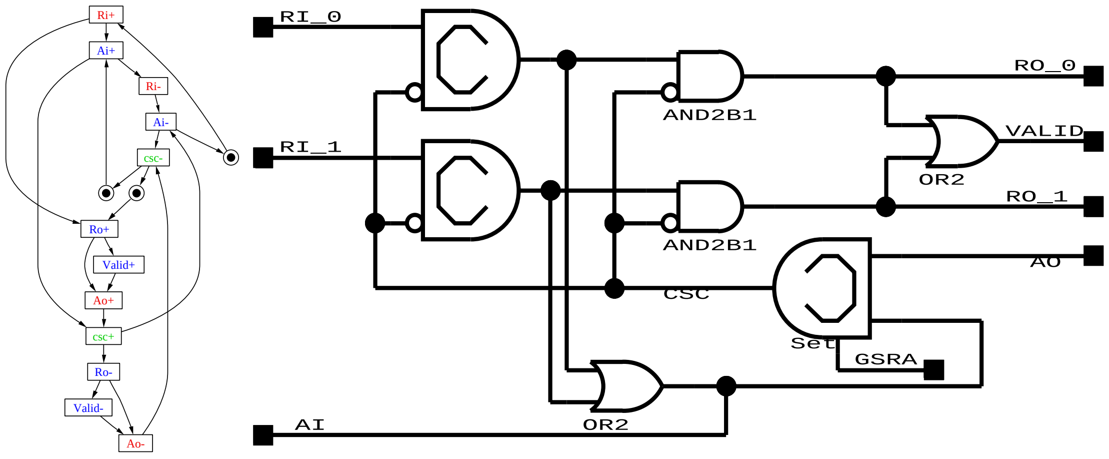

Figure 4.7: Standard semi-decoupled latch STG and schematic.

4.3.3 Semi-decoupled latch behaviour

As before, because the OR gate is implemented using early output logic, the output is valid before all inputs have arrived. If the latch the OR gate outputs to is semi-decoupled, then when the latch's output is acknowledged, it will return to NULL. In figure 4.8 this can be seen where the latch inputs one value and outputs another. This allows the following stages to complete their reset phase and start working on the next set of inputs. The latch remembers not to allow the input request to propagate to the output by setting the csc signal. The csc signal will reset once the input requests have dropped.

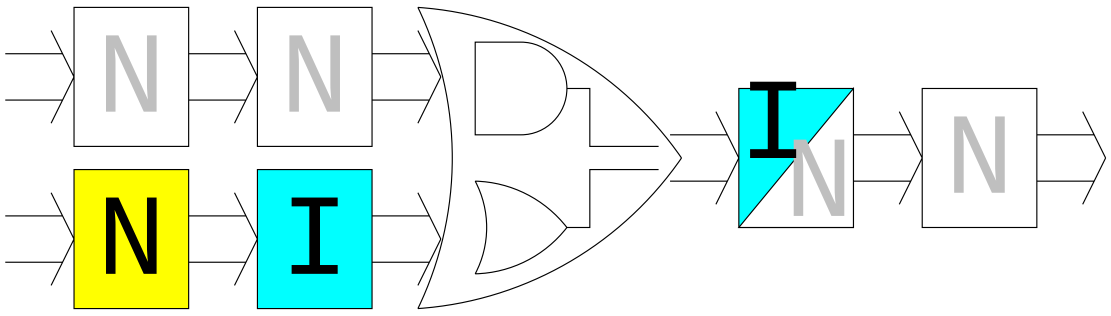

Figure 4.8: Semi-decoupled latch allowing following stages to return to zero

4.3.4 Optimised semi-decoupled latch

Instead of using a csc signal to keep state, knowledge of the circuits surroundings can be used. The valid output goes to a C-element whose output returns as the Ao input. For this reason all the STG graphs have the Valid -> Ao links. In all previous designs the valid signal was controlled by the Ro signal (Ro -> Valid). By attaching the valid signal to the transitions directly linking to the Ro signal it is possible to decouple the two signals. The Ri- -> Ro- link can now be broken, but by keeping the Ri- -> Valid- link the Ao- transition is dithered until the Ri- transition happens. This gives full state encoding and doesn't have any "marking exceeds the capacity" errors when passed through petrify. Figure 4.9 shows the STG graph described above along with the schematic created from the petrify output. This latch design replaces a large and slow settable C-element with a relatively cheap NOR gate which a future chapter will show can be optimised away. The Valid signal is now simply an identity of the Ai signal. The buffer in the schematic is only there to allow the net to have two names and will be optimised away. Compared with the standard half latch the optimised semi-decoupled latch adds only two AND gates which can be incorporated into the C-element design.

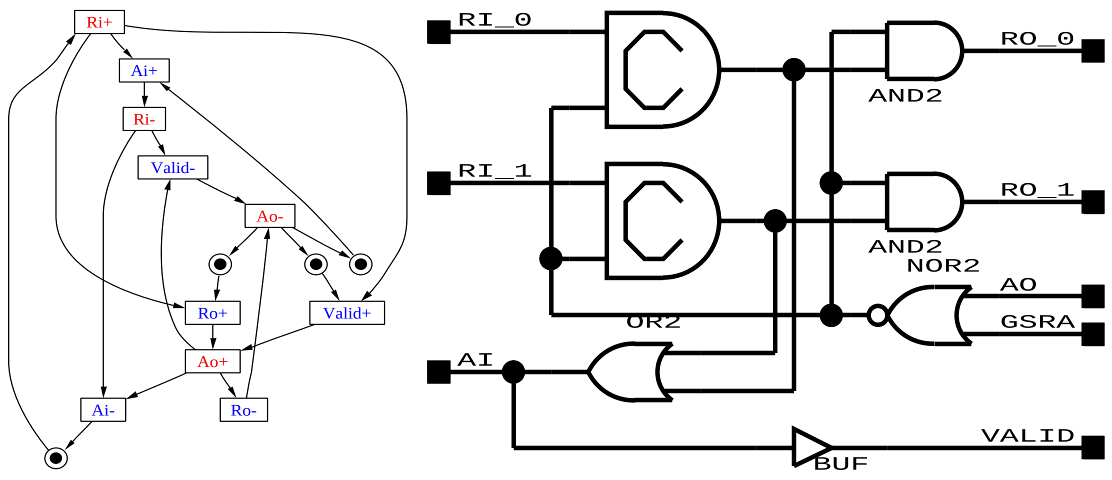

Figure 4.9: Early output optimised semi-decoupled latch

4.3.5 Optimised semi-decoupled latch behaviour

By keeping the valid signal high the stage which the latch outputs to cannot fully reset and start processing new data. But as the latch outputs a NULL which flows through the following latch it does allow subsequent stages to complete their reset stage and accept new data tokens.

4.4 Semi-decoupled latch detailed overview

In section 4.1 the sender and receiver protocol were proven to be fully delay insensitive. In the sender protocol where the link between data and acknowledge can be data dependant and but the validity signal guards and stops the acknowledge signal from arriving early. To make hazard free circuits it is important to check the operation of the new latches.

4.4.1 Semi-decoupled protocol

Figure 4.10 shows the operation of a semi-decoupled latch. As before the dashed lines represent the data dependant causality. As before if the Ao transition is not dependant on the Ro transition then there is still a Valid transition dependency to ensure Ao comes after Ro. The second cycle in the diagram the Ri signal remains high for a long period of time but the Ro is decoupled and returns low. The Ro+ is caused by Ri+ but Ro- is caused by csc-. This allows Ro to return to NULL during long Ri high periods. The csc signal also keeps state to enforce that Ri and Ai drop before a new cycle can begin. Because in the STG the csc and the Valid signals were placed in sequence within output loop the cycle time increases considerably compared to the standard half latch. The results chapter will give more details.

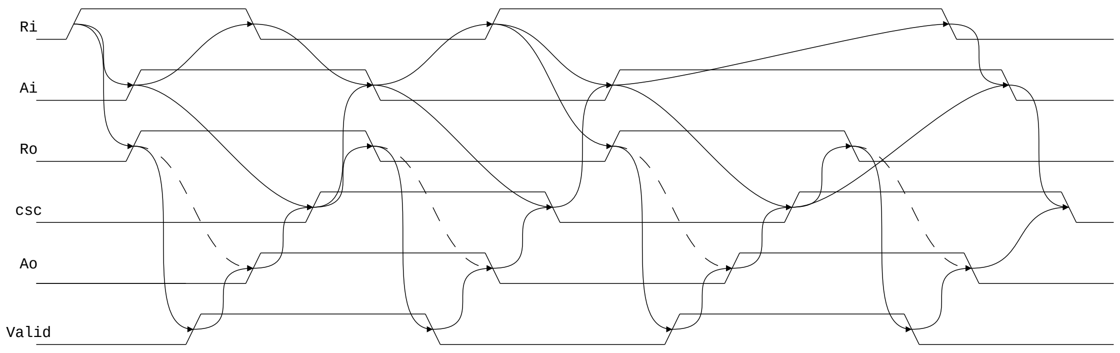

Figure 4.10: Semi-decoupled latch protocol

4.4.2 Optimised semi-decoupled protocol

The operation of the early output optimised version of the semi-decoupled latch is shown in figure 4.11. This time the cycle time is visibly shorter due to more parallelisms in the STG specification. As before Ro+ is caused by Ri+ but this time Ro- is triggered by Ao+. Valid signal is now totally decoupled from Ro. Ro is only observed by the output latch.

Because early output logic is used the target under certain conditions may not be able to observe the Ro transitioning. This is not a problem on the rising edge as if the target latch is unable to observe one of the inputs rising then the data supplied to the other inputs was sufficient to already produce an output. But on the falling edge the input to the target latch can fall without Ro falling. A different hazard can arise if the acknowledge signal comes directly after Ro has gone high. This would cause a glitch which can reach target latch after its input has returned to zero.

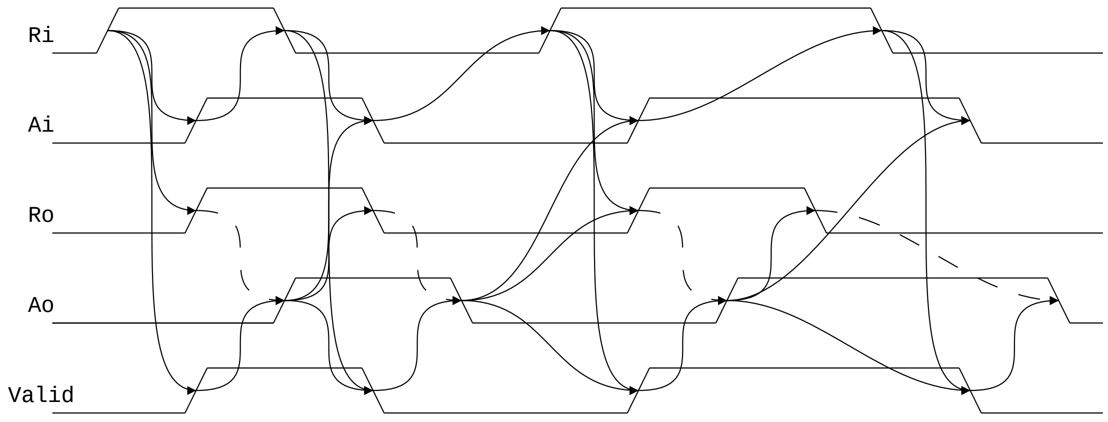

Figure 4.11: Optimised semi-decoupled latch protocol

4.5 Dual-rail gate overview

4.5.1 Standard dual rail gates

The gates used in standard dual rail have safe transitioning. To make a upwards output transition all inputs have to be valid. Even if a combination of inputs has concluded the result without all inputs being valid the output may not be driven high. Simmilarly in the reset phase the gate may not drop its output untill both its inputs have returned to zero. With the use of these two assumptions it is possible to draw that a pipeline stage will not go to a reset phase while any gate has not transitioned high. Also it will not enter a data phase without its output returning low. Even if one of the inputs is very slow the circuit will still operate correctly. Figure 4.12 shows the transition sequence a gate with both the input and output dual-rail wire pairs being abstracted to a single signal.

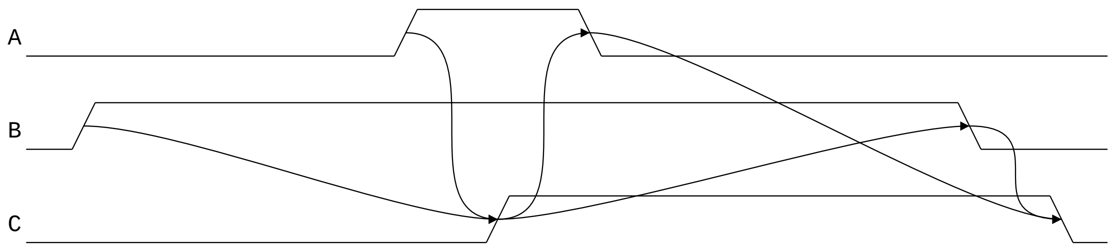

Figure 4.12: Standard dual-rail gate operation

4.5.2 Early output gates

As shown above, the standard dual rail gates have a are fully delay insensitive. Any of the signals can take an abitrary ammount of time and the operation will still be correct. The early output gates will only operate correctly if several timing assumptions are met. Dashed lines in figure 4.13 examplify these assumptions. The output can become valid when only one of the inputs is valid. In this situation the late signal might not reach the gate before the first signal drops again. The output can drop when only one of the inputs has dropped. The validity circuit was built to ensure all inputs and outputs drop before moving to the data state. Also all inputs and outputs have to become valid before entering the reset phase.

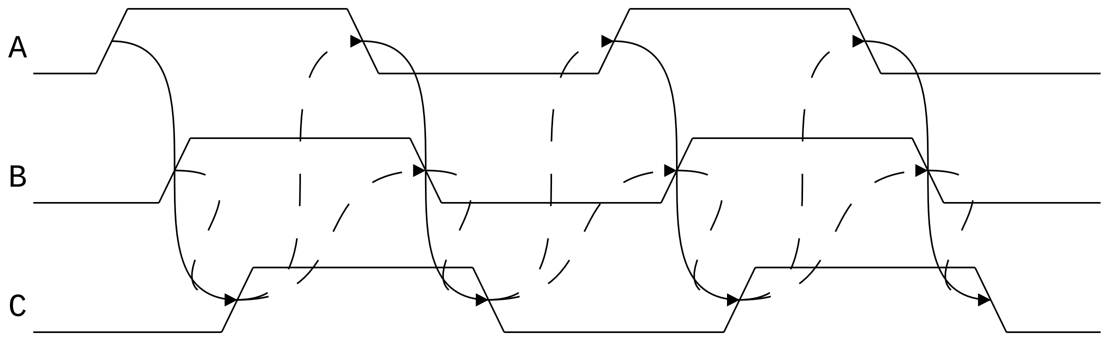

Figure 4.13: Early output dual-rail gate operation

4.6 Summary

A more detailed look was taken of the early output logic methodology. The logic style allows the system to bypass some of the synchronisations enforced by the standard dual-rail system. With the use of semi-decoupled latches the system can complete their reset phase and advance to next set of inputs while some input stages are stuck in the data phase of the previous operation. With the use of optimised early-output latches this can be done with the minimal space and latency overhead. Unfortunately this latch is not fully delay insensitive.

Chapter 5: Anti-tokens

“Choice without arbitration is like perpetual motion machines and anti gravity devices. It can’t be done!” James Garside

5.1 Overview

5.1.1 Introduction

The previous chapter described methods to reduce synchronisations with the use of early-output logic and various semi-decoupled latch designs. Even if these approaches are used, pipeline stages will be forced to wait for the slow tokens to enter the stage before being acknowledged. Ideally once the result of the stage is known without all inputs being present, the stage should be able to continue with the next set of inputs. The late inputs will have to be acknowledged when they arrive and only the second set of these inputs should be allowed to pass to the logic.

5.2 Anti-token latch specifications

5.2.1 Hazards of decoupling the valid signal

The only thing stopping the a stage which has a valid result but is still waiting for some late inputs to arrive from acknowledging is the validity of these late inputs not being high. All validity lines need to be high to allow the validity C-element to switch to the reset phase. The validity created by a half latch or the standard semi-decoupled latch is created by placing an OR gate across the two output signals. This ensures that the acknowledge signal does not come before the output is valid. On the optimised semi-decoupled latch the validity signal was set when the data out was set but was allowed to remain high while the data out dropped. This allows the latch to reset the stage and prime it for the next input while it waits for it’s input to drop. The following stages can fully complete their reset cycle. Decoupling the validity from the data out signal is allowed in the early output protocol, as long as validity only drops when data out is already low. Dropping validity while data out is still high can cause a logical hazard. This means that the reset cycle can complete while the data signal remains high. This data signal now looks like it has arrived early for the set phase while in fact it is withdrawn late from the reset phase. This hazard will be discussed in more detail in the next chapter but until then it is important to note that the only transition dependency between the valid and data out is for valid to drop data must be low.

5.2.2 Decoupling the valid signal

The valid signal controls the acknowledge out signal. The Acknowledge out signal may not transition without Valid transitioning. This allows the latch to control the output stage set and reset phases without the use of data out. This is necessary as data out may be unrequired in the logical function and its rise or fall is unseen. The in the semi-decoupled latch the valid signal was dropped late in order to stop the stage the latch outputs to from fully resetting. By delaying the validity signal the latch stops acknowledge from continuing. Only when the latch is ready to accept another acknowledge transition will it change the state of validity. When decoupled from the data signal the validity signal states if the latch is ready to receive an acknowledge signal.

In latch designs presented so far the latch is only ready to be acknowledged when it is outputting data. Only then does it have a piece of data to destroy. By rising the valid signal the latch allows a rise on the acknowledge wire. Even if the data out wire remains low, this is acceptable behaviour in the early output protocol. This can mean that the acknowledge signal arrives before the data signal had been sent. Effectively the latch has now received an order to destroy the next token it receives. This can be thought of as the latch holding an ‘anti-token’. A latch holding an anti-token must wait for a token to arrive and acknowledge it without allowing it to pass through the latch and effect the later pipeline stages. The latch can acknowledge its input even before it has received a data signal. This would appear to the acknowledged latches as the logical function competing without the need of their inputs. If these latches did place data on the outputs then this data would be acknowledged. If they were standard semi-decoupled of half latches then they will stop the acknowledge signal from rising before they have a piece of data to destroy. Anti-token latches will allow the acknowledge line to raise before they have data to destroy so they will accept an anti-token.

5.2.3 Early acknowledge

An ‘early acknowledge’ is defined as an acknowledge before a data input. Although an early acknowledge is allowed in the early-output protocol, it is important to only assert it after the reset phase has completed. If the acknowledge signal is raised before the validity C-element output drops the stage will never leave the reset phase. This is because one of the inputs to the validity C-element is the acknowledge. In normal token passing operation the acknowledge will come only after the new data input has arrived. A new data input can only arrive after the stage has completed its reset phase and entered the data phase. This ensures the acknowledge is not asserted while the stage is still in reset phase. Additionally the reset signal should remain asserted until the stage moves into reset phase. The transition into reset phase is again shown by the state of the data in signal. The data in signal will not drop until the stage has entered the reset phase and it is safe for the latch to drop its acknowledge.

5.2.4 Snoop on valid-acknowledge

The latch holding an anti-token must wait for the stage to enter the data phase before acknowledging it. The easiest way is to observe the data in line. The data in signal can only go high when its in the data phase and drop in reset phase. Unfortunately this would force the latch to keep the anti-token until a token reaches it and only then destroy both.

Another way of finding the phase of the input stage is to snoop on the validity C-element output. The validity C-element output shows the stage phase. When the output is low the stage is in data phase and when the output is high the stage is being reset. Figure 5.1 shows the arrangement of validity C-elements in a simple circuit. The C-element takes the validity of all inputs and combines them with the acknowledge signal from the latch. The result is passed back to all inputs as an acknowledge and to the output latch to allow it to snoop on the phase of the pipeline stage.

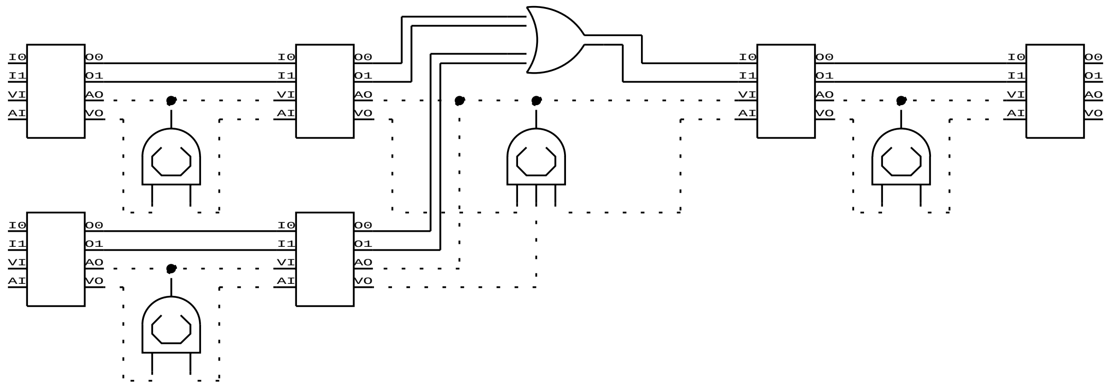

Figure 5.1: Anti-token latch example pipeline

5.2.5 Acknowledge and validity sequencing

In an early acknowledge the acknowledge signal should not rise while the stage is still in reset phase. By snooping on the validity it is possible to enforce this by ensuring that acknowledge does not rise while validity remains high. When acknowledge is raised the latch must wait for the validity C-element to switch back to reset phase to ensure that the signal is acted upon by the C-element. The acknowledge signal places requests for the stage to move between data and reset phases.

Similarly on the output of the latch the validity out signal controls the acknowledge out. Again the signal coming in is created by a C-elements which can be controlled by an output. The output can only control the input by delaying a transition as shown in the optimised semi-decoupled latch. This ability of delaying token or an anti-token enables the latch to stop an overflow of tokens or anti-tokens. A latch will not release the acknowledge in signal until it is ready to accept another token. Similarly the acknowledge out transition can be postponed until the latch is ready to be acknowledged.

5.2.6 Movement of anti-tokens

This organisation of C-elements outside the latches shown in figure 5.1 is very similar to a control pipeline shown in chapter 1. Using the validity C-elements the anti-token latches are able to communicate acknowledge signals without the need of a data portion of latch. This allows the anti-tokens to be passed backwards through the pipeline without affecting the logical part of the circuit. Instead of using gates to ensure all inputs are able to accept an acknowledge the early output circuit uses the validity tree.

An anti-token will cause the latch holding it to raise its acknowledge signal early. If all input latches are anti-token latches then an anti-token will be placed all that have no token to destroy. Once all latches have been acknowledged the input latches themselves will try and acknowledge their inputs. While some input latches are trying to propagate their anti-tokens the stage is kept in reset phase. This is done to stop another anti-token from arriving while they are still trying to propagate the last one. This is done by not releasing the validity out signal until the input stage has entered the reset phase. Anti-tokens will therefore be separated by a NULL stage. This ensures that when anti-tokens are stacked they remain separated and will not over write each other. The same method is used with data tokens. All data tokens are separated by a NULL stage to stop one token over writing another.

5.2.7 Anti-token deadlocks

Tokens will duplicate themselves across forks and combine across gates but ensuring certain rules like the minimum number of latches in a loop the circuit will never deadlock. A circuit ensures that for each fork that duplicates a token there is a gate which combines tokens. Although the number of tokens may rise and fall there is no way to create a circuit which will create an unlimited number of tokens unless an output is blocking their destruction. An output refusing to acknowledge is not a deadlock situation. And similarly a circuit can not unlimitedly destroy its tokens and deadlock it self through the lack of tokens.

Using the fact that in correct circuits for every fork there is a gate (not literally) it is possible to state these circuits will never deadlock. Anti-tokens flow backwards through the system. They duplicate across gates and combine across forks. If it is possible to make the assumption that tokens cannot deadlock the system because their number is limited dues to forks and gates then it is also possible to state that anti-tokens will not deadlock the system as they follow the exact opposite set of rules

So far it has been shown that a token can be propagated backwards through a circuit and duplicate it self across gates to create a set of anti-tokens which is equivalent to the set of tokens when combined across gates would make the token the anti-token was created to destroy.

5.2.8 Token and anti-token collisions

There are several possible ways that tokens and anti-tokens can meet. The destruction of both the token and the anti-token during collisions relies on the latch holding the token to believe that the output is trying to acknowledge its token rather than pass an anti-token. This ability to use a signal for two distinct jobs which have the same effect allows the latch to be built without attributers. MORE HERE!

5.2.9 Anti-token propagation through complex circuits

All anti-token propitiations can be described in two models. A one to many circuit has a latch outputting to many latches. A many to one circuit takes the outputs of many latches and combines them using gates to feed a single latch. In the case of a many to one circuit the output latch holding the anti-token will first assert its acknowledgement signal. Once all input latches are ready to accept an acknowledgement the validity C-element goes high and input latches will either receive an anti-token or clear their token. In a one to many circuit the input latch once ready to receive an anti-token will assert its validity out. Once some output latches receive their anti-token they will rise their acknowledge and switch their validity C-element high. The input latch will only receive an acknowledge once all latches it outputs to have switched the validity C-element on. Only then will the acknowledgement C-element tree have all inputs needed to switch. Individual output latches may drive their validity C-elements and snoop that they have switched on. They may then release their acknowledge signal but the C-elements will remain on until the input latch releases its validity out. This stops the output latches from trying to pass another anti-token into the stage while the previous anti-token is processed.

A many to many model follows rules of both models described above. An input latch will wait until all latches it outputs to have placed an acknowledge and an output latch will wait for all its input latches to accept the anti-token before driving another.

5.2.10 Mixing anti-token latches with other latch designs

So far only communication between anti-token latches was described but one feature of anti-token latches is they can be safely mixed with other early output latches. Non anti-token latches will only receive an acknowledge once they drive the valid signal high. Even if they output to an anti-token latch which has driven its acknowledge signal early, the validity C-element will only switch once all input latches are ready to be acknowledged. A mixture of half, semi-decoupled and anti-token latches can be used as inputs to a stage. If a latch is capable of receiving an anti-token it will but if it is not then the stage is halted until a token has arrived. Even though non anti-token latches can not propagate anti-tokens they can still create them by acknowledging a data token where an input anti-token latch has not contributed to the logical operation.

5.3 Anti-token latch design

The anti-token latch specifications were created using a minimalist description. This was done because the latch exhibits several non delay insensitive properties. These made the process of designing the latch using pertify very difficult. Minimalist was able to accept a very simple non delay insensitive description of the latch and synthesize a hazard free circuit.

5.3.1 Overview

Figure 5.2 shows the interface of an anti-token latch. In order to allow easier design the Ri and Ro signals have been placed outside of the latch. To drive Ro the latch uses signal S which controls the C-element which Ri needs to go through to transition Ro. Again the latch does not control the signal directly rather it can delay the transition until it is ready to be accepted. The state of Ri is not being monitored but the state of Ro is fed back into the latch. As in the other latches the Vo signal passes through the validity C-element and returns as the Ao signal. The latch monitors on the state of the previous stage by snooping on the output of the validity latch. This signal is input into the latch as ‘Vi’.

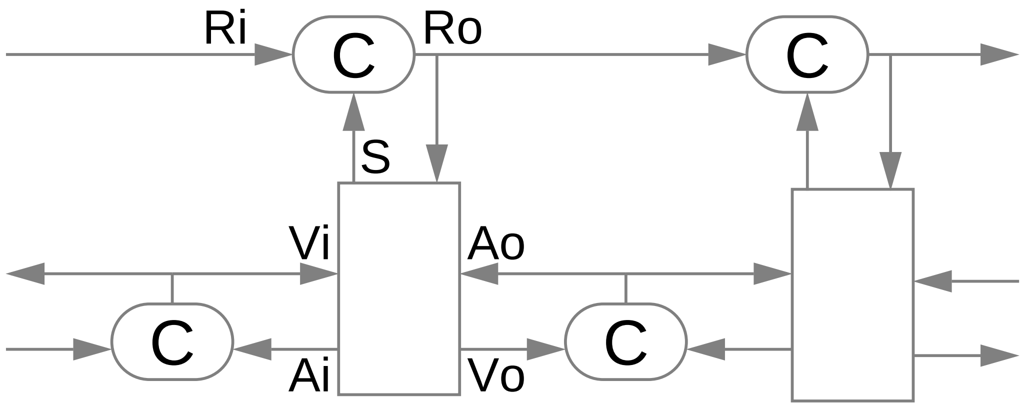

Figure 5.2: Anti-token latch interface

5.3.2 Token passing action

Figure 5.3 shows the minimalist description of the token path in the anti-token latch. The description is only able to handle passing tokens and not anti-tokens. In the initial state 0 the Vo signal is high. This allows the latch to receive an early acknowledge. As the S signal is also high in the initial state the latch may receive the Ro+ or Ao+ signals in any order. The normal token passing operation of the latch is to receive a Ri+ first. This causes the latch to acknowledge its input by raising the Ai line. The stage may be still waiting for a late input so the latch is only allowed to drop Ai when it sees Vi rise. On the output side of the latch, once the Vo signal is raised the Ao may rise. Often this will not happen until the latch has contributed some information into the logical operation by rising Ro. Once Ao has risen the latch may drop Ro. This is done by dropping S. Ro will still remain high while Ri is high so it is important not to drop Vo while Ro remains high. Only once Vi and Ao have risen can the latch drop both Ai and Vo. These signals are not dropped independently as this would require the latch to keep state. Once both validity C-elements have dropped the latch is ready for another token to pass through it. It can once again rise the S and Vo signals.

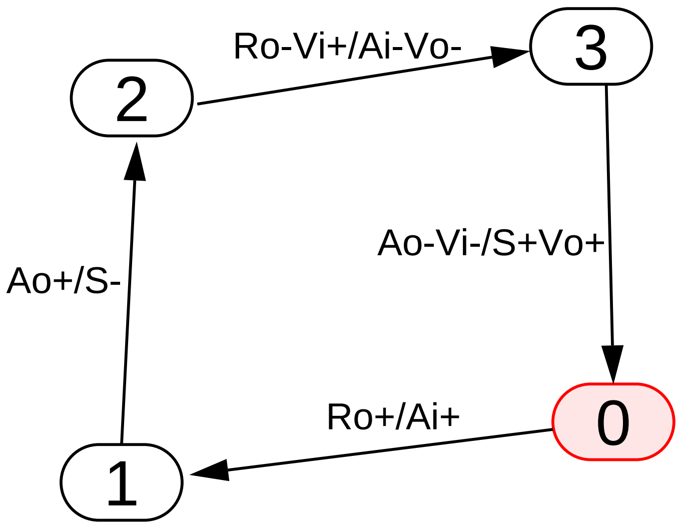

Figure 5.3: Minimalist description of the token path in the anti-token latch

5.3.3 Anti-token passing action

Figure 5.4 shows the minimalist description of the full anti-token latch including the anti-token passing path. From the initial state 0 the circuit now has a choice of going to state 1 or state 4. In the initial state the latch can receive an early acknowledge signal. In the case of Ao rising before Ro does firstly S is withdrawn in order to stop transitions on Ri propagating into the next stage and then moves to execute the anti-token passing path starting with state 4. The latch then rises Ai to attempt an early acknowledge on the input stage. Again it can not drop the Ai signal until the Vi has risen. Only when the Vi line has risen is it safe to drop Ai and Vo. Once the validity C-elements in both stages have dropped the latch can again enter its initial state by rising S and Vo.

Note that this time it is impossible to test for Ri returning to zero by observing Ro. Later in the chapter an explanation will be given of how it is possible to ensure Ri is low before re-enabling S. Raising Ai early may be interpreted as an anti-token or as an acknowledge. Either way a token has been destroyed. The action of withdrawing S is very dangerous and is not allowed in delay insensitive circuits. Thus the latch is not delay insensitive as this S- transition is not acknowledged. It is possible to make very simple timing assumptions that S- will happen and the data C-element will be locked low stopping a Ro+ transition long before Vo drops.

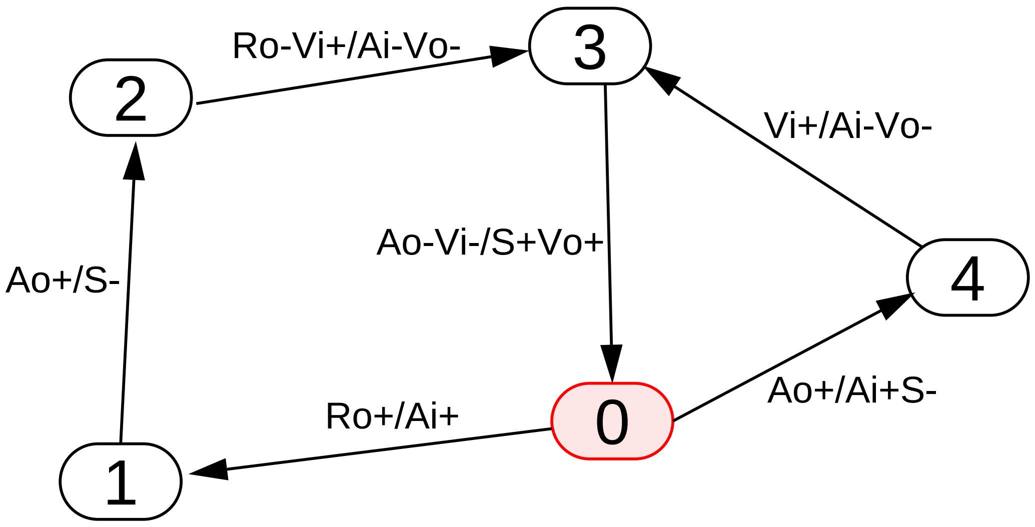

Figure 5.4: Full minimalist description of the anti-token latch

5.3.4 Simultaneous token and anti-token requests

The above examples only describe situations where there is a clear distinction between a request for a token or an anti-token. But as in the initial state both the data and validity out C-elements are primed to fire, both Ro+ and Ao+ transitions can happen simultaneously. In this situation the latch has to be able to choose the correct action. If the wrong action is chosen then either one of the tokens can survive the collision or they can bypass each other.

In situations where there is a choice dependant on one of two actions happening first but where the two actions can happen simultaneously the circuit must use some form of arbitration to make a safe choice between the two requests. It does not matter in which order the requested tasks are completed as long as the fact that both requests happen simultaneously does not perform a wrong action. In the anti-token latch the need for arbitration is avoided by arranging the transitions so that both requests require the same actions to be performed. In the descriptions of the two transition sequences it is possible to see that from the initial state if either the Ao+ or the Ro+ transition happens the required action (Ai+) is equivalent. There are two differences between the two sequences. Firstly from the initial state the first transition decides the choice of next state. If it is possible to synthesize the description so that both paths share one state it will be possible to ensure that there is no need for arbitration. To do this it is necessary to ensure that all internal state and output signals go through the same transition sequences in both paths. The input signal transitions may vary but the operation of the circuit must remain the same so it is possible to create the circuit without the need to record a difference in the state between the two paths in an internal signal.

If the circuit does not distinguish between the two paths using an internal state signal than it is possible to create a circuit with choice but without arbiters and still ensure that it is hazard free. The only other difference between the paths is the fact that the token passing path requires Ro to drop before progressing to state 3. This test for Ro low can be placed in the other path without any errors as Ro never made the upwards transition. It is safe to say it is already low and the extra test will never fail.

5.3.5 Late Ro+ transition

Although S- transition should stop the Ro+ transition the circuit may have already moved to state 4 before it is noticed. As described above in order to ensure the correct operation of the circuit the two paths must have the same state encoding. This means that state 4 must have the same internal state as state 2. So placing a test for Ro- in the transition between state 4 and 3 will allow the state 4 and state 2 to be combined into one. Unfortunately this is not allowed as minimalist needs a consistently encoded specification. This means when taking the anti-token path the Ro signal never raised and so cannot be tested for dropping. This situation can be solved by inserting a conditional state change into the minimalist specification. Shown in figure 5.5 the late Ro+ transition in state 4 causes the circuit to move over to state 2. This way it is possible to input this description into minimalist and ensure a hazard free synthesized circuit.

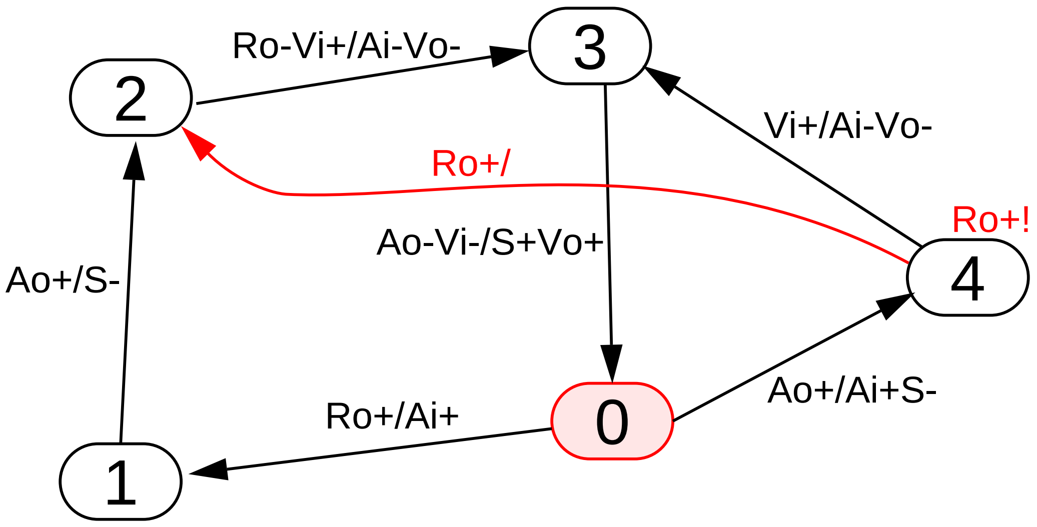

Figure 5.5: Late Ro+ transition example

5.3.6 Withdrawing S hazard

It is possible to say as the two paths have the same state and so the choice can be done without inducing hazards. It has been shown that by withdrawing S as Ri+ is about to switch the data C-element, Ro can go high in state 4. This is protected against but unfortunately the C-element can also go metastable. Although in this stage it does not matter if Ro is high or low as both result in correct operations, the metastability can have two hazardous effects. Firstly the signal can seem to be a certain value and only much later transition to a different value when the circuit has moved to a state where it is unable to deal with this transition. The second possible hazard when a signal becomes metastable is that the signal can be observed as holding different values to different components in the circuit. The problem of metastability can not be ignored and has to be solved in order to create workable circuits. Additionally the inability to observe Ri returning to zero during an anti-token pass will have to be solved. These problems will be solved by altering the circuit rather than by changing the minimalist description. This is because Ri signal can rise and drop while S is low which will make the minimalist description much more complicated. Instead these hazards can be solved by analysing the synthesized circuit.

5.3.7 Minimalist synthesis

The burst-mode specification in figure 5.4 was entered into minimalist. The specification was synthesized with a number of complex gate suites with target optimization including area and speed. Below is the set of equations which achieve the desired effect chosen from the most optimal implementations created by the different target optimization strategies.

```
S = Vi’ Ao’
Ai = Ro + Vi’ Ao Vo
Vo = Ro + Vi’ Ao’ + Vi’ Vo
```

A generalised C-element suite was run on the specification and below is the resultant set and reset portions of the desired C-elements.

```
S_S = Vi’ Ao’
R_S = Ao

S_Ai = Ro + Ao S
R_Ai = Ro’ Vi

S_Vo = Vi’ Ao’
R_Vo = Ro’ Vi
```

Both proposed solutions are valid and even a combination of the two is possible. When creating an implementation it is important to choose the smallest and fastest circuit suggestions available.

S signal

The S signal can be much more easily implemented using a NOR gate suggested in the complex gate implementation than using an asymmetric C-element. It is both smaller and faster than the alternative.

Vo signal

The Vo signal can be implemented using an asymmetric C-element much more easily than using a selection of gates. The C-element has two set inputs and two reset inputs. This makes the element the same size and speed as a standard 2 input C-element.

Ai signal

In the case of the Ai signal both implementations are of similar size. Ignoring inversions the complex gate version requires a two input OR gate and a three input AND gate. The general C-element version requires three set and two reset transistors. One important fact to note is that the set and reset sets are not mutually exclusive. This can lead to both set and reset regions of the C-element being turned on and creating a direct power to ground connection. Additionally the asymmetric C-element requires a custom cell to be built for this rare combination.

5.3.8 Minimalist generated circuit

The schematic shown in figure 5.6 is of the anti-token latch created by using the elements synthesized in minimalist. For the Ai signal the gate implementation was chosen for clarity. The Vo circuit can be created using an asymmetric C-element but unfortunately this requires either an inversion on the output of the C-element or on all inputs.

This circuit is not the optimal implementation as it is impossible to input all information into the burst mode specification. The circuit does prove that a hazard free circuit is possible and the fact that a circuit can be made without any state storing elements proves that the choice can be made safely. Although the circuit is hazard free and behaves correctly it is still too big and slow. Synthesizing the circuit by hand directly from the behavioural description should give better results.

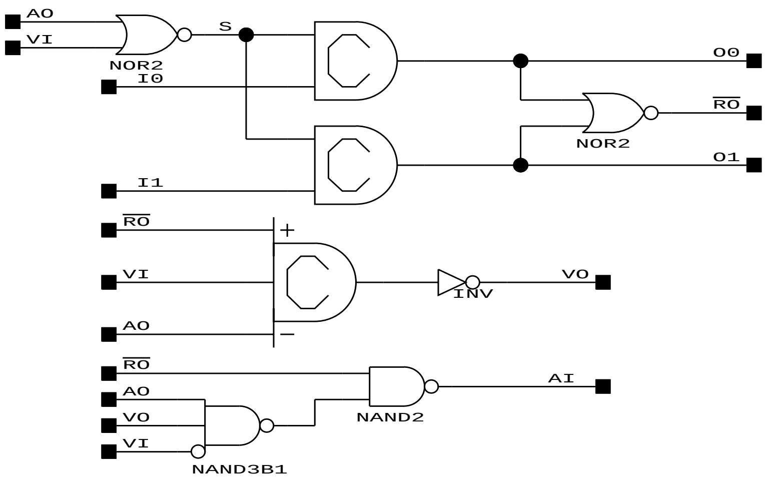

Figure 5.6: Anti-token latch schematic

Chapter 6: Optimised anti-token latch

6.1 Hand synthesis of the anti-token latch

6.1.1 Token passing behaviour

Figure 6.1 shows the behaviour of the anti-token latch while passing a token. In the diagram the Vi signal does not effect any latch signals. This is because the Vi signal is only used when passing an anti-token. Using this behaviour diagram it is possible to hand synthesize a better anti-token latch circuit. This circuit will only be able to cope with passing tokens. Later this will be corrected by analysing the anti-token passing behaviour and combining the circuit created there with the token passing functions. This way a circuit can be made which works in both conditions but with the minimum complexity when designing.

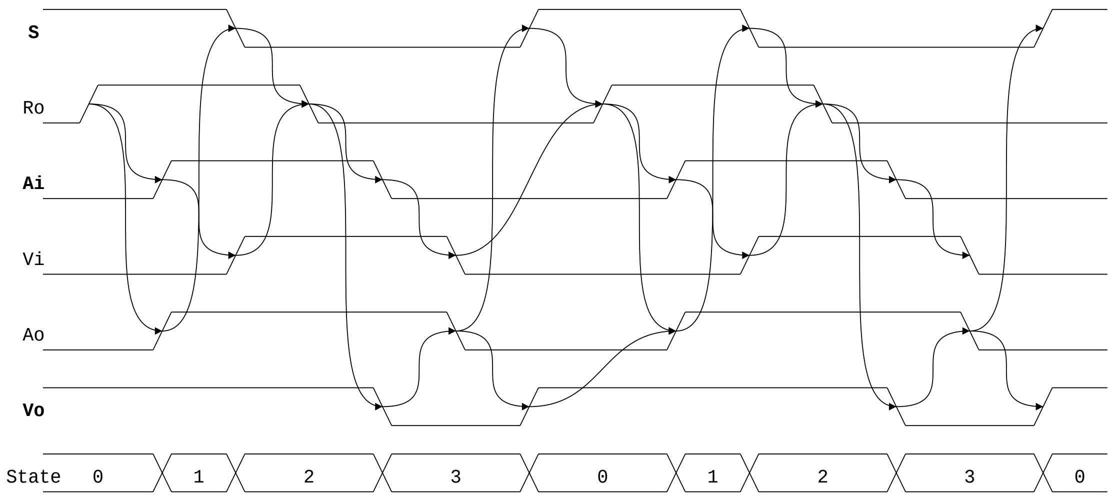

Figure 6.1: Token passing behaviour

S and Ai signals

The S signal transitions are always triggered by transitions on the Ao wire. Creating the S signal can be simply done by inverting the Ao wire. The Ai also reflects the value of another wire. The value on Ro can be used to create the Ai signal.

```
S = Ao’
Ai = Ro
```

Vo signal

Vo is triggered by both the Ro and Ao signals. The transitions of the Ro and Ao signals are dependant on each other. An Ro transition relies on a Ao transition through S. Also Ao transitions rely on Ro transitions (indirectly in the case of Ao- through Vo-). This ensures transitions Ro+ → Ao+ → Ro- → Ao- will happen in sequence. Vo needs to drop on a Ro- and raise on a Ao-. This allows the signal to be generated much more simply. The Vo signal needs to stay high except when Ro is low until Ao becomes low. The equation below satisfies this requirement.

```
Vo = Ro + Ao′
```

6.1.2 Anti-token passing behaviour

Figure 6.2 shows the behaviour of the anti-token latch passing an anti-token. This time the Ro signal is ignored as it is not needed during an anti-token pass. The S signal is driven but will not have any effect as this signal is only used when passing tokens. The signal still has to be driven as the next transaction could be an token. Again similar methods will be used to synthesize the circuits out of the behavioural description.

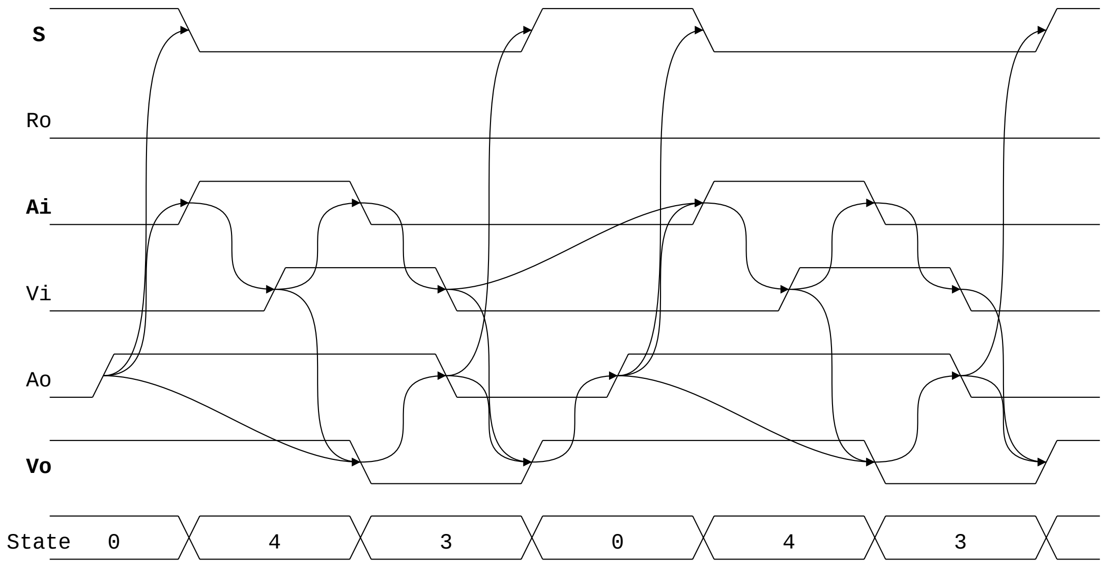

Figure 6.2: Anti-token passing behaviour

S signal

As in the token passing behaviour the S signal is directly represented by inverting the value on Ao.

```
S = Ao’
```

Ai signal

Each Ai transition requires to be proceeded by a transition on Vi. Additionally the Ai+ transition must wait for a Ao+. Ao and Vi transition in sequence although Ao- and Vi- can happen in parallel. The sequence of transitions Ao+ → Vi+ → Ao- || Vi- → Ao+ can not happen out of order. Although the easiest solution to the problem would be to use a gate to combine the two inputs. ‘Ai = Vi’ Ao’ would unfortunately be incorrect as Vi could drop before Ao and cause the gate output to raise for the second time in the cycle. As a simple two input gate is no longer possible a more complex strategy must be used. Because the two wires can enter the Ai switch state (Vi low Ao high) later in the cycle when Ai should not transition, another wire must be used to keep state. The Vo signal drops when Vi raises. This Vi+ transition also causes Ai to drop. Vo remains low until both Vi and Ao are low. The Ai switch state on wires Vi and Ao can only occur during the time that Vo is low. If the Vo signal is added to the gate creating the Ai signal then the Ai high Vi low state will not switch the gate.

```
Ai = Vi’ Ao Vo
```

This implementation is similar to that created by Minimalist. Unfortunately there is a race between Vi and Vo. Vi can switch high, cause Ai to return low and itself switch low before Vo had a chance to switch. This hazard can occur in the minimalist version of the circuit. Minimalist states that a delay between output bursts and input transitions must be kept in order to create hazard free circuits. Luckily the problem can be easily solved. Instead of watching Vi directly the state can be observed by watching Vo instead. Both Ai and Vo need to wait for Vi+ before transitioning to low. The Vo signal is already in the Ai creating gate so using it to also observe Vi will not cost anything. This creates a simpler circuit.

```
Ai = Ao Vo
```

This also removes the direct link between Vi and Ai. Instead Ai waits for and ensures that Vo transitions correctly before Vi is allowed to switch. This can make the circuit slower but more simple.

Vo signal

The Vo+ transition requires Ao and Vi to be become low. Although the Vo- transition requires both Ao and Vi to be high, Vi becomes high as an indirect consequence of Ao going high. Ao remains high until Vi does go high. It is possible to say that when Vi goes high only while Ao is high and so for a Vo- transition only Vi needs to be observed. The fact that for a Vi+ transition only one of the lines signals needs to be tested allows the removal of a gate or a transistor from the C-element in the circuit. The gate version of the circuit simply raises Vi when both Vi and Ao are low and only drops Vo when Vi raises.

```
Vo = Vi’ Ao’ + Vi’ Vo
```

The asymmetric C-element uses two transistors for the set and one for reset sides of the C-element. Unfortunately both inputs are inverted. Either the inputs need to be inverted before entering the C-element or the output of the C-element is inverted.

```
S_Vo = Vi’ Ao’
R_Vo = Vi
```

6.1.3 Combining and comparing implementations

The token and the anti-token implementations can now be combined. It is important that the combined version is able to both pass tokens and anti-tokens.

S signal

In both the token and anti-token passing behaviours the S signal can be generated directly by inverting Ao. Minimalist did not generate this simpler circuit as the transition from state 3 to state 0 in the burst-mode specification requires both Ao- and Vi- to happen before S raises. The real requirement are: Ao drops for S to rise, Ao and Vi drop for Vo to rise. Unfortunately Ao- and Vi- can happen in either order. The easiest method to specify this behaviour is to describe it all in one transition. Unfortunately this implies extra rules minimalist requires in the implementation.

```
S = Ao′
```

Ai signal

The Ai signal is generated by Ro in the token cycle and by Ao Vo in the anti-token cycle. A combined circuit must be created which behaves like Ro during a token pass and as Ao Vo during an anti token pass. Ro remains low during an anti-token pass so should not effect the signal during an anti-token pass. In the token pass Ro rises before Ao and directly causes Vo to drop. The fact that Ao Vo will not raise before Ro goes high and drops directly after Ro drops allows the two functions to be passed to an OR gate and combined. The resultant circuit works but in the token pass the Ai- transition waits for Vo to drop which happens a gate delay later than just waiting for Ro to drop. This removes a an inverted input into the and gate saving some area and power but possibly reducing the speed.

```
Ai = Ro + Ao Vo
```

Another circuit can be used for a possible speed improvement.

```
Ai = Ro + Vi′ Ao Vo
```

Vo signal

In the token pass the Vo+ transition needs Ao- to happen but in the anti-token pass both Ao- and Vi- need to occur. As Vo+ happens when the circuit is resetting back to its initial state it is impossible to tell which mode it is in, so the safe approach of waiting for both Ao- and Vi- to occur must be taken. Vo- occurs when Ro drops in the token pass and when Vi rises in the anti-token pass. In the anti-token pass Ro- does not occur and in the token pass Vi+ occurs before Ro-. This stops the use of just one of the variables to cause the down transition. The second option to use one of the other variables is also not viable as the only other variable which exhibits the desired property (Ai drops when Vi is high and Ro is low) gets its order to transition from Vo. If Ai was used then both signals would wait for each other to drop and the circuit would deadlock. This means that the Vo signal has to wait for both Vi+ and Ro- before dropping.

```
S_Vo = Ao′ Vi′
R_Vo = Ro′ Vi
```

In this case no improvement could be made on the minimalist generated solutions. The gate implementation is also a viable option for this signal.

```
Vo = Ro + Vi′ Ao′ + Vi′ Vo
```

6.1.4 Schematic

Figure 6.3 shows a schematic of the hand synthesized anti-token latch. Data C-elements are now controlled directly by Ao. No improvements could be made in the Vo generating circuit. Ai has one less input which would have needed an inverter. It is still possible to put the Vi input back into the Ai function to gain extra performance. This would decrease the delay between Vi+ and Ai- by one C-element delay but add one inverter and turn a two input NAND gate into a three input gate.

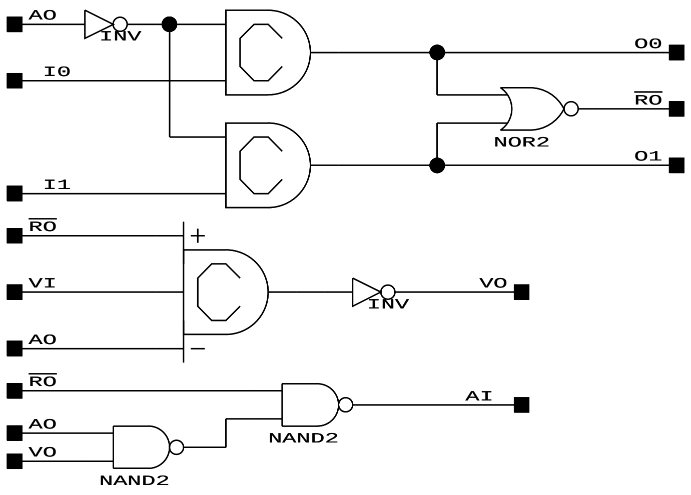

Figure 6.3: Hand synthesized anti-token latch schematic

6.2 Removing hazards

6.2.1 Ri high during anti-token pass hazard

In an anti-token pass the input latch which is being acknowledged early can be holding a token so before the early acknowledge comes it may raise its data lines. As noted in the previous chapter, during an anti-token pass the latch does not observe the Ri wires. Although at the sending latch the data lines are is dropped before the Vo drops the delay through the logic before the signal gets to Ri may keep the lines high even when the validity C-element drops. When the validity C-element drops Vo and Ao will drop too, causing S to be reasserted. This allows the data C-element to be switched on with the data from the previous stage which can cause incorrect circuit behaviour. In the token passing action Vo is kept high until Ro drops. The Ri signal dropping is observed through Ro as Ro cannot drop before Ri. This stops Ao from dropping and re-enabling S. To stop the ‘Ri high during anti-token pass hazard’ the Ao- transition must be delayed until both Ri and Ro have dropped. The Ao- transition is dependant on Vo-, so by delaying Vo- it is possible to defer Ao- until it is ready to be accepted. In the circuit creating the Vo signal the condition that Ri is low before Vo- transition occurs can be easily added. The asymmetric C-element simply needs an extra pin connected to the plus side driven by a NOR gate from with inputs from Ri. Alternatively a four input NOR gate combining both Ri and Ro signals can be used in place of the two input NOR gate driving the positive pin of the asymmetric C-element. Note that the Vo C-element is inverted before driving the Vo signal.

6.2.2 Metastability of Ro during an anti-token pass

During an anti-token pass the Ro C-element may be half way through switching when S is withdrawn. This may move the data C-element into metastability. The above section described how the Ri signal can be observed directly rather than through Ro. This allows Ri to be tested without the need to set Ro and observe it dropping each cycle. As Ri is observed directly Ro can drop with out the need to wait for Ri to drop. If the S signal dropping resets the data C-elements without the need for Ri to drop the metastability can be removed. The original metastability was caused by S dropping and the C-element moving into its ‘keep state’ mode while not being fully switched. Now S dropping signals the C-element to reset rather than keep state. Figure 6.4 shows the updates schematic of the interfacing of the latch to the rest of the circuit. The latch now snoops on Ri. Also the data C-element becomes asymmetric and the Ri signal only drives the plus side. This causes the C-element to reset when S drops, even if Ri remains high.

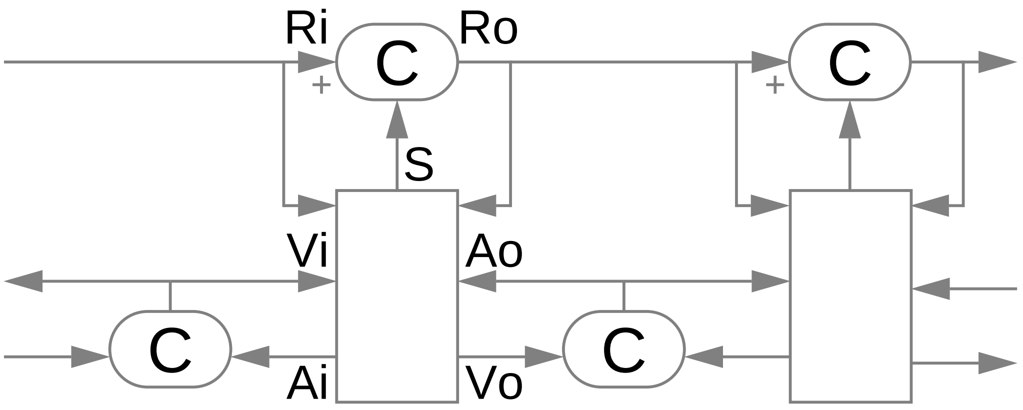

Figure 6.4: Anti-token latch interface

6.2.3 Hazard free anti-token latch schematic

Figure 6.5 shows the implementation of the anti-token latch including the hazard removing features described above. The cost of the hazard removal is very low. There is little or no cost in speed. Five extra transistors are added by the NOR gate and the extra input into the Vo C-element but the data C-elements can now be made without the use of keeper inverters and saving one transistor on each.

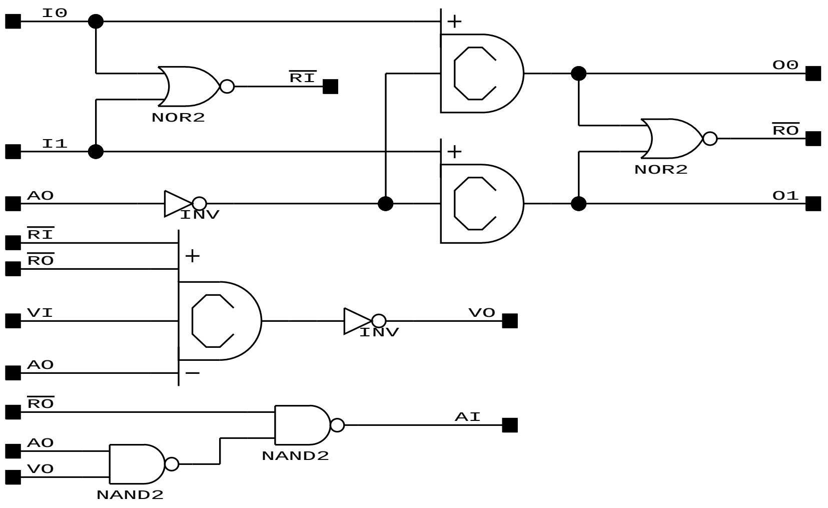

Figure 6.5: Hazard free anti-token latch schematic

6.2.4 Races

The Ro signal can be reset without Ri dropping first. Ai signal which observes Ro to ensure Ri has dropped can no longer do so as this assumption is no longer true. In the anti-token pass Ai rises without Ri ever transitioning as it is controlled by Ao and Vo instead. When either Vo and Ao are high the Ai signal will also be high. When Ro returns low the Vo and Ao signals should take over the control of Ai and only allow it to drop when Ri drops. The drop of Ro is caused by Ao rising. While Ao is high (along with Vo) Ai will remain high. Unfortunately the Ro- transition can reach the Ai producing function before Ao+. The Ao+ signal has to go through four inversions before Ro signal drops (inverter, two inversions in the C-element and a sensing NOR gate). The Ao signal must pass through a NAND gate before Ri drops or Ai may glitch. Figure 6.6 shows the two paths of the race. These race conditions can be easily met. A four to one race is generally safe unless the final design is very badly routed. But if there is a possibility of this hazard occurring a safe circuit can be designed.

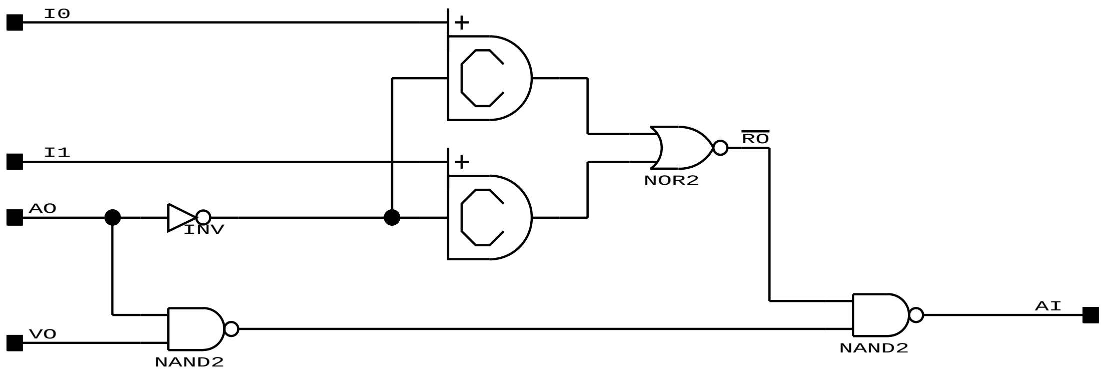

Figure 6.6: Race in the anti-token latch

Figure 6.7 shows an implementation of the anti-token latch which is free of races. The race is easily be removed by not relying on Vo and Ao signals during a token pass. Instead the Ri signal is observed directly and only releasing the Ai signal to drop when it has dropped. This design adds eight extra transistors (one NAND, one inverter and an extra pin one a NAND). It is advised to not use this design as the race condition of four inversions to one can be easily met and the cost of ensuring hazard free using this anti-token latch implementation is costly in both area and power.

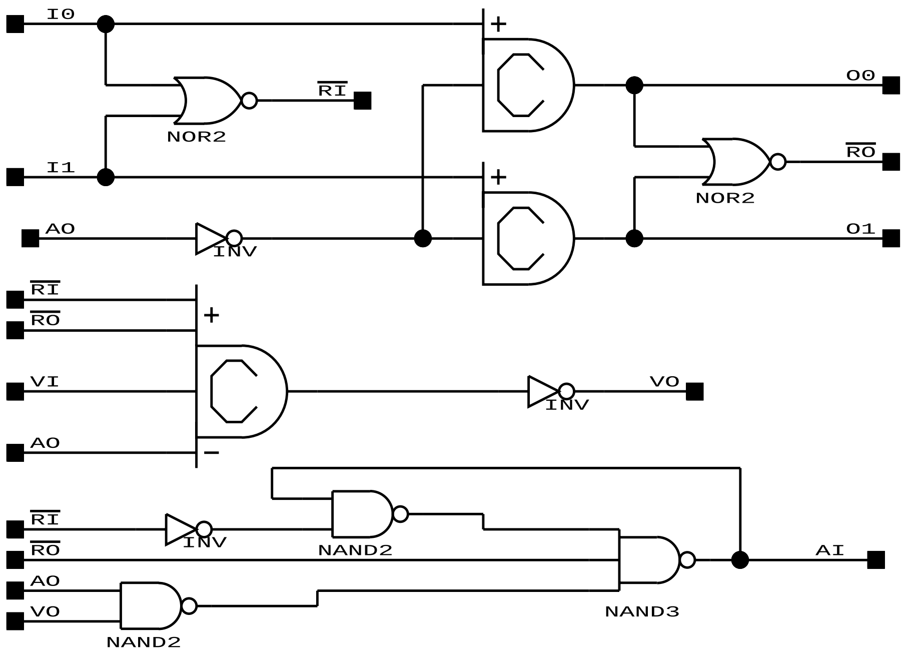

Figure 6.7: Race free anti-token latch

Although the circuit is now fully functional and hazard free it still must initialise into the correct state.

6.2.5 Reset state

When the reset signal is activated the latch should clear its state holding elements and return to its initial state. To allow the external validity and acknowledge collection C-elements to reset both Ai and Vo should be low at reset time. Also the data C-elements should be reset to remove any token inside the latch. The latch can assume that the data in signals will be low but the other signals (Vi and Ao) can be in any state. As Ao may be low at reset time and the data C-elements must be reset the Ao signal is passed through a NOR gate along with the reset signal in place of the inverter to allow a manual reset. This will reset the data C-elements and ensures that Ro is low. In the initial state the Vo signal is high but in order to reset the validity C-elements it should be forced low. The Vo signal can be forced low using the same method used for the Ao signal. The inverter is replaced with a NOR gate which allows the second input (reset signal) to force the output low. The Vo C-element will eventually reset low as Vo and Ai are set low to allow the external C-elements to reset and so will eventually cause the drop of Vi and Ao. These are the two signals required to be low for the Vo C-element to drop. Ai will reset without help as both Ro and Vo are low. Figure 6.8 shows the resettable anti-token latch. The overall cost of inserting the reset signals is four extra transistors. The speed and power penalties are nearly zero.

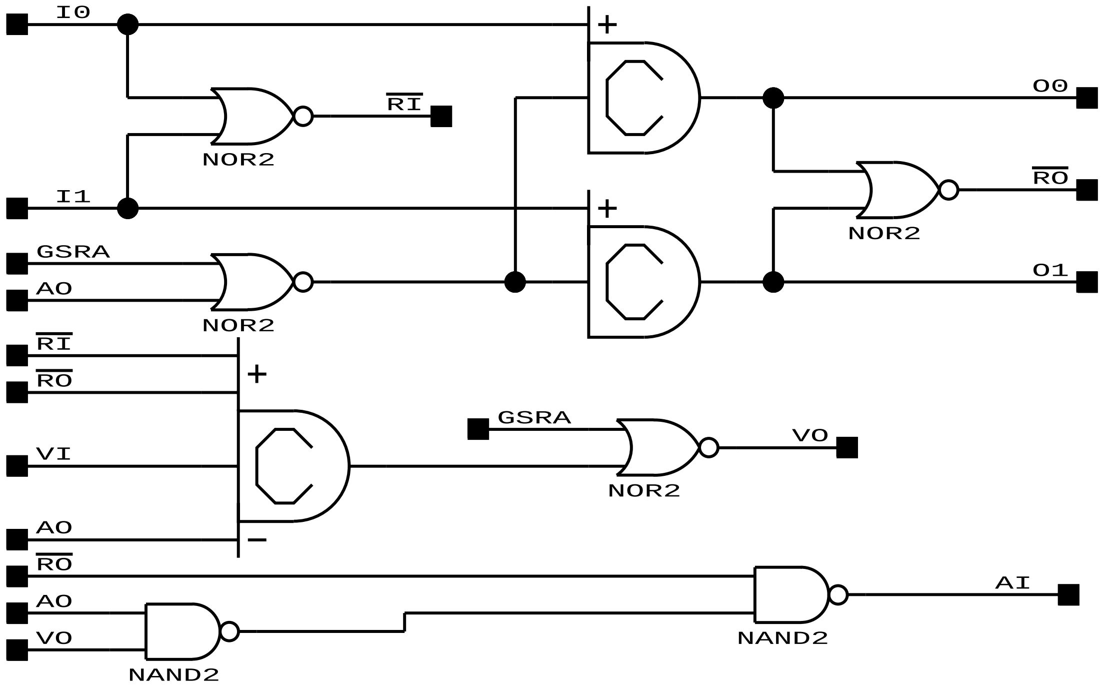

Figure 6.8: Resettable anti-token latch

6.3 Anti-token behaviour

6.3.1 Anti-token creation

Figure 6.9 shows the example from chapter 4 where the OR gate receives a token on one of its two inputs. The token is able to switch the gate to output a valid value. The output latch receives a token so it acknowledges. The as the late inputting latch is an anti-token latch it drives its validity line high before outputting valid data. This allows the validity

C-element to switch and the latch receives an early acknowledge. This causes the latch to hold an anti-token.

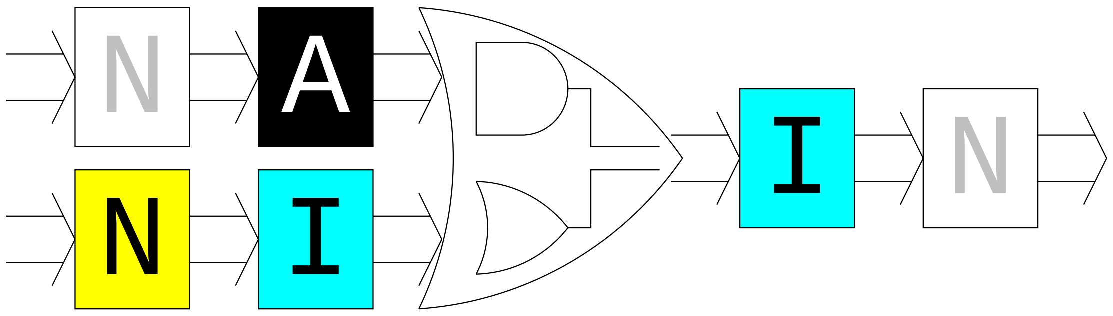

Figure 6.9: Anti-token creation

The acknowledge reaches the bottom latch and removes its data so the bottom pipeline is ready to accept more tokens. The token in the result pipeline will move forward. The anti-token created in the top pipeline will move backwards trying to destroy a token. It will pass through logical stages as well as pipelines.

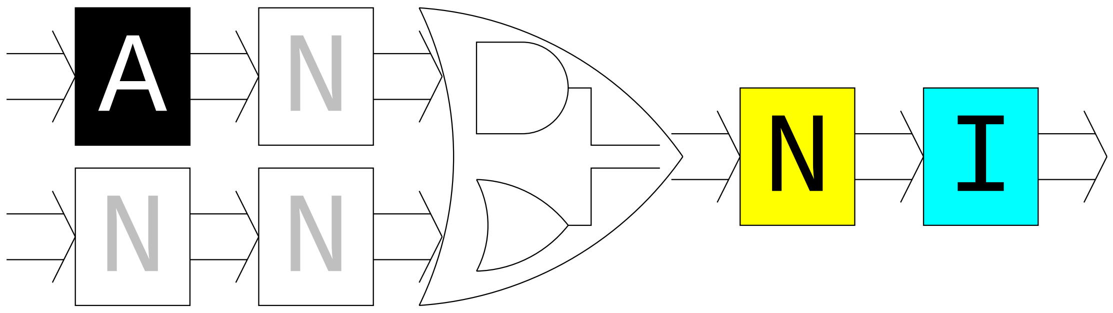

Figure 6.10: Movement of an anti-token

Anti-tokens are created by early output logic stages when an input into the stage is not nessesary for the creation of a valid output. To increase the chance of anti-token creation rather than blocking the stage it is important that the logic outputs early whenever possible.

6.3.2 Anti-token movement

The anti-token while travelling backwards will be able to cross logic stages. In the example in figure 6.11 the anti-token will cause both of the input latches to accept an anti-token. If one of the input latches holds data then the data is removed and no anti-token is inserted. This allows the anti-token to unblock logic stages. Not only is the stage which created the anti-token freed to process a new set of inputs but other stages that the anti-token passes through are able to flush and process new data.

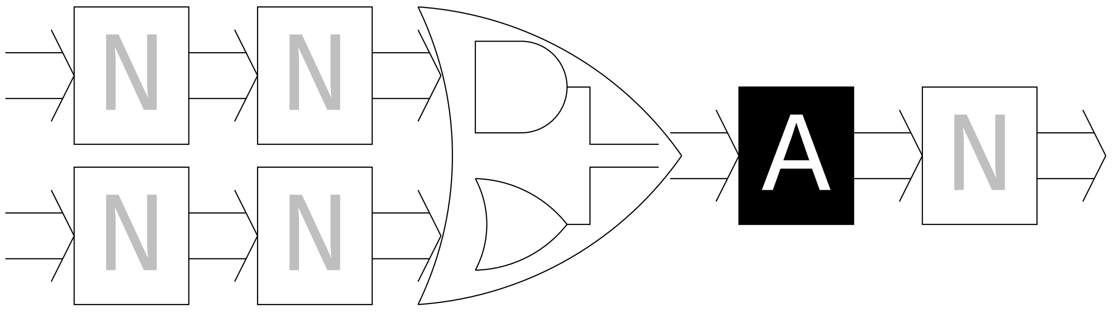

Figure 6.11: Anti-token propagation through logic

Chapter 7: Conclusions

7.1 Overview

This thesis presented three main consepts which when combined can be very benifitial to the asynchronous design community. The direct translation

References

[1] J. Sparsø and S. Furber, “Principles of Asynchronous Circuit Design”, Kluwer Academic Publishers, 2001, (ISBN 0-7923-7613-7)

[2] S. B. Furber, J. D. Garside, S. Temple and J. Liu. “AMULET2e: An Asynchronous Embedded Controller”. Proceedings Async ’97, pp. 290-299, IEEE Computer Society Press, 1997.

[3] Montek Singh, Jose A. Tierno, Alexander Rylyakov, Sergey Rylov, and Steven Nowick. "An Adaptively-Pipelined Mixed Synchronous-Asynchronous Digital FIR Filter Chip Operating at 1.3 Gigahertz". Proceedings of the 8th IEEE International Symposium on Asynchronous Circuits and Systems (“Async-02”), 2002.

[4] J.D. Garside, W.J. Bainbridge, A. Bardsley, D.M. Clark, D.A. Edwards, S.B. Furber, J. Liu, D.W. Lloyd, S. Mohammadi, J.S. Pepper, O. Petlin, S. Temple and J.V. Woods, “AMULET3i - an Asynchronous System-on-Chip”, Proceedings of the 6th IEEE International Symposium on Asynchronous Circuits and Systems (“Async 2000”), 2000.

[5] S.B. Furber and P. Day, “Four-Phase Micropipeline Latch Control Circuits”, IEEE Transactions on VLSI Systems, vol. 4 no. 2, 1996.

[6] D.E. Muller, “Asynchronous logics and application to information processing”, Switching Theory in Space Technology, Stanford, University Press, Stanford, CA, 1963.

[7] G. Kayne and J. Heinrich, “MIPS RISC Architecture”, Prentice Hall, 1992, (ISBN 0-13-590472-2)

[8] C. Brej, “Yellow Star: A MIPS R3000 microprocessor on an FPGA”, 2001.