

Final Report for ECE 6770 Project

# Design of Asynchronous Interconnect Network for SoC

Hosuk Han<sup>1</sup>  
han@ece.utah.edu

Junbok You  
jyou@ece.utah.edu

May 12, 2007



<sup>1</sup>Team leader

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Project Overview</b>	<b>2</b>
2.1	Asynchronous Network Fabric . . . . .	3
2.2	Interface circuit: FIFO and Synchronizer . . . . .	3
2.3	Processing Element . . . . .	3
<b>3</b>	<b>Asynchronous Network Fabric Design</b>	<b>4</b>
3.1	Switch Module . . . . .	4
3.1.1	<i>Petrify</i> Version . . . . .	4
3.1.2	<i>3D</i> Version . . . . .	6
3.2	Join Module . . . . .	7
3.3	Design Issues . . . . .	9
3.4	Circuit and Layout . . . . .	9
3.4.1	Switch Module . . . . .	10
3.4.2	Join Module . . . . .	11
3.4.3	Asynchronous Router . . . . .	13
<b>4</b>	<b>Asynchronous FIFOs</b>	<b>14</b>
4.1	Asynchronous Circuit Modules . . . . .	14
4.1.1	Linear Module . . . . .	14
4.1.2	Toggle Module . . . . .	15
4.1.3	Merge Module . . . . .	16
4.2	Different Structural FIFOs . . . . .	18
4.2.1	Linear FIFO . . . . .	18
4.2.2	Parallel FIFO . . . . .	18
4.2.3	Tree FIFO . . . . .	19
4.2.4	Square FIFO . . . . .	19
4.3	Performance Comparison . . . . .	20
4.4	Conclusion . . . . .	21
<b>5</b>	<b>Behavioral Validation &amp; Results</b>	<b>22</b>
5.1	Golden Model & Testbench . . . . .	22
<b>6</b>	<b>Conclusion &amp; Further Researches</b>	<b>22</b>

<b>A</b>	<b>Petrify Input Files</b>	<b>25</b>
A.1	Asynchronous Network Fabric Modules . . . . .	25
A.1.1	Switch.g . . . . .	25
A.1.2	Join.g . . . . .	26

# List of Figures

1	System Level Block Diagram . . . . .	2
2	Chip Level Block Diagram . . . . .	2
3	Router Block Diagram . . . . .	3
4	Interface of Switch Module, Petrify Version . . . . .	4
5	Petri-net for Switch . . . . .	5
6	Interface of Switch Module, 3D Version . . . . .	6
7	Extended Burst Mode FSM for Switch . . . . .	6
8	Interface of Join Module . . . . .	7
9	Petri-net for Join . . . . .	8
10	Circuit for Switch . . . . .	9
11	Circuit for Switch Controller . . . . .	9
12	Circuit for Switch Module Test . . . . .	10
13	Simulation of Switch Module . . . . .	10
14	Layout of Switch Module . . . . .	11
15	Circuit for Join . . . . .	11
16	Circuit for Join controller . . . . .	12
17	Simulation of Join Module . . . . .	12
18	Layout of Join Module . . . . .	13
19	Layout of Async Router . . . . .	13
20	Linear Module . . . . .	14
21	Linear Module's Petri-net and STG . . . . .	15
22	Toggle Module . . . . .	15
23	Toggle Petri-net and STG . . . . .	16
24	Merge Module . . . . .	17
25	Merge Module's Petri-net and STG . . . . .	17
26	Linear Asynchronous FIFO . . . . .	18
27	Parallel Asynchronous FIFO . . . . .	18
28	Tree Asynchronous FIFO . . . . .	19
29	Square Asynchronous FIFO . . . . .	19
30	Various FIFO's waveform for latency . . . . .	20
31	Testbench Model . . . . .	22

# 1 Introduction

As System-on-Chip (SoC) design is getting complicated, communication between many components in SoC is becoming more difficult and important. Furthermore, the wire delay compared to gate delay becomes more significant. The performance of SoC therefore highly depends on that of the interconnect architecture. Besides high performance and low power operation, a new interconnect architecture should have ability to connect synchronous blocks which have different operation frequencies since each IP block in current SoC may operate its own frequency.

Interconnect system for SoC can be designed with either synchronous or asynchronous design technique. Sufficiently enough support of industrial CAD tools in synchronous design process enables fast and reliable design and verification of synchronous system and high coverage of testability of design. However, global clock problems of synchronous system, such as clock distribution, clock skew problem, are getting worse in SoC design for providing one global clock signal in whole SoC system. Furthermore, this global clock issue prevents predesigned IP blocks from operating with their optimized clock frequencies. Every IP block should be modified its operating clock frequency matching with global clock. As an alternative method, a synchronizer can be used between two clock systems, synchronous interconnect system and each synchronous IP block. But, usually a synchronizer between two clock systems is more complicated and causes more synchronizing cost than between an asynchronous system and a synchronous system.

On the other hand, asynchronous design does not have support with well-developed CAD tools as much as synchronous design. It leads that designers need to spend more time to design and verify their circuits or systems. Asynchronous design is also suffering from testability problem. But, generally asynchronous design provides higher performance and lower power operation compared with synchronous design. Asynchronous interconnect system in SoC design promises several advantages that compensate its disadvantages and surpass the advantages of synchronous interconnect system. With asynchronous interconnect system, SoC system does not require one global clock in whole system any more. It leads to elimination or reduction of any global clock distribution and clock skew problem. The more valuable benefit is that predesigned synchronous IP block can be utilized with its operating frequency optimized for its own power and performance.

From the perspective of the characteristics of interconnect system in SoC, Globally Asynchronous Locally Synchronous (GALS) system is viewed as a promising solution for SoC design. In GALS systems, each synchronous IP core operates with local frequency and asynchronous interconnection is used for enabling each clock domain to communicate with each other. It means that GALS system can make use of the advantages of asynchronous interconnect system as aforementioned.

The ultimate goal of this project is to implement a simple GALS system made up of several simple processing elements (PEs), interface circuits and an asynchronous network fabric for evaluating the performance of our new network fabric. PEs generate network traffic in the network fabric. Interface circuit will support to connect between synchronous PEs and asynchronous network fabric. This project can be seen as a revised version of ECE6710 class project in which synchronous network fabric and other elements were designed. We can compare performance and other factors between synchronous interconnect system and asynchronous one after completing this project.

## 2 Project Overview

Our system will consist of 6 PEs, Interface circuits and an asynchronous network fabric and as a network topology, binary tree architecture will be used as shown in figure 1. Each PE has two communication ports, sending port and receiving port. Data generated from one PE are sent from the sending port via the interface circuits and the network fabric to a receiving port of one of other PEs. In previous project in which  $0.5\mu\text{m}$  technology is used, due to the available size limit of fabricated chip, the whole system couldn't be fit in a one chip. Therefore, one PE, two types of interface circuits, a synchronous interface and an asynchronous interface, and one router were integrated in one chip. Even though  $130\text{nm}$  technology is used for this project instead of  $0.5\mu\text{m}$  resulting in giving more design area, we assume the same architecture of chip design with that of previous project. Figure 2 shows the one chip design diagram in which various interfacing architectures and an asynchronous router. We are able to focus on designing and implementing an asynchronous network fabric and new interface architectures since the PE designed in the previous project can be used.

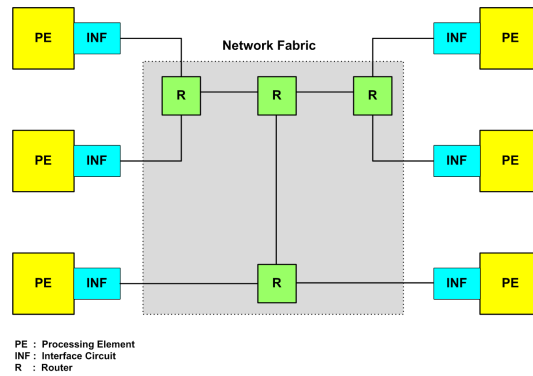


Figure 1: System Level Block Diagram

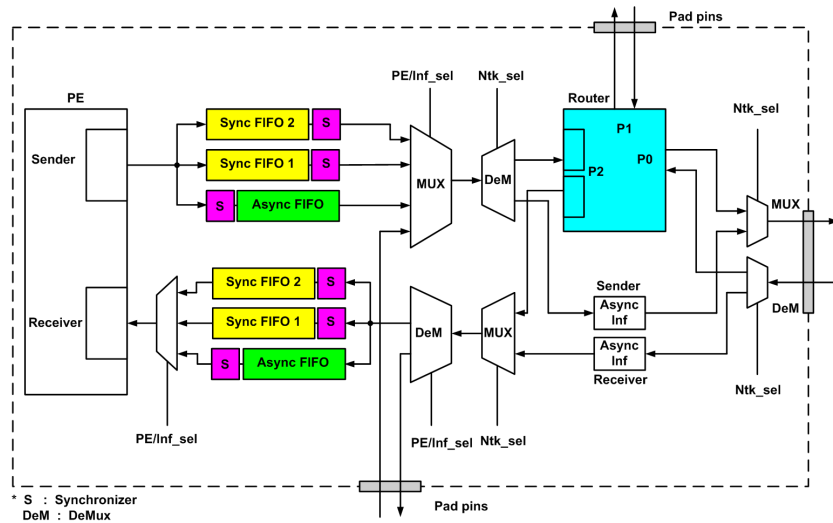


Figure 2: Chip Level Block Diagram

## 2.1 Asynchronous Network Fabric

The main component of network fabric is a router which controls a path for transferring data. Since the binary tree topology is used for our network fabric, the router will be a shape of “T” and handle the communication of between three ports. The communication method of our network fabric is simply to transfer data from a source to a destination . It does not have any data buffering inside of network and does not use packet-based data transfer. This results in a simple design of router which is composed of two types of modules, Switch and Join. Switch module is for selecting a path between possible two other output ports based on the destination address embedded in the data format. Join module arbitrates simultaneous request to transfer data from two other input ports. Each port of three ports in router need to have one switch and one join modules. Consequently, a router consists of three switches modules and three join modules as illustrated in figure 3.

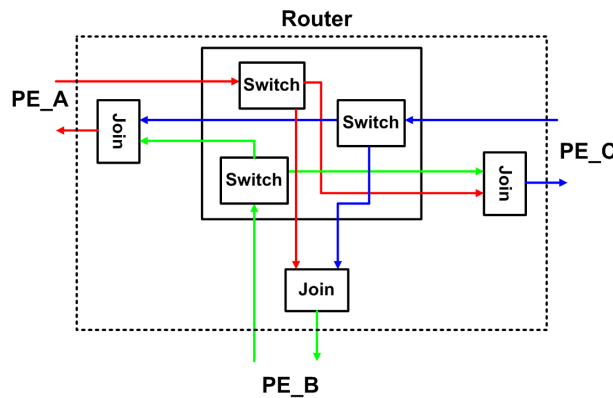


Figure 3: Router Block Diagram

## 2.2 Interface circuit: FIFO and Synchronizer

Besides the network fabric, there are additional two important modules in the interconnect system, FIFO and synchronizer. For the asynchronous network fabric to communicate with any synchronous system without metastability problems, it needs a help of a synchronizer. Two types of synchronizers, a general two-flops synchronizer and a fast synchronizer using MUTEX, designed and utilized in the previous project will be used in this project. FIFOs in interface part between an IP block and the network fabric will improve communication performance. We will investigate various kinds of synchronous and asynchronous FIFOs as well. Thereafter, two or three types of FIFOs will be implemented in our chip for evaluating and comparing performance, power and area.

## 2.3 Processing Element

One more supplemental module is PE which is intended for generating network traffic in the network fabric acting like a synchronous IP block. The design of PE has been almost completed in the previous project and we will use this PE with a little modification that includes reducing the size of PE and adding functionality to adopt FIFO depth variously depending on network traffic burden.

### 3 Asynchronous Network Fabric Design

Among various tools for asynchronous circuit design, we chose to use *Petrify*. After we defined a specification of each functional module in CCS [3], a petri-net was generated based on the specification and it was converted to the input file format of *Petrify*.

Two asynchronous modules, switch module and join module, were designed as components for the router of the network fabric. The same handshake protocol of the asynchronous FIFO controller in [4] were basically used in designing these two modules.

#### 3.1 Switch Module

##### 3.1.1 *Petrify* Version

As depicted in figure 3, a switch module is for selecting a proper path for data transfer between two possible paths based on destination address. It has one left input channel and two right output channel. When a request for data transfer occurs in left channel, the switch module triggers one of two right requests. Figure 4 shows an interface diagram between the switch module and other asynchronous FIFO modules.

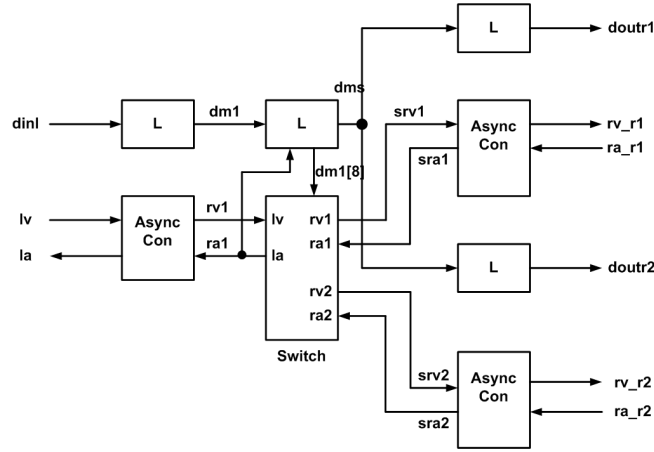


Figure 4: Interface of Switch Module, Petrify Version

The specification of the switch module using CCS is

$$\begin{aligned}
 L &= lv \uparrow . (\bar{c}_1 . \bar{t}_1 . \bar{la} \uparrow . lv \downarrow . \bar{la} \downarrow . L + \bar{c}_2 . \bar{t}_2 . \bar{la} \uparrow . lv \downarrow . \bar{la} \downarrow . L) \\
 R_1 &= t_1 . \bar{rv}_1 \uparrow . ra_1 \uparrow . \bar{rv}_1 \downarrow . ra_1 \downarrow . \bar{r}_1 . R_1 \\
 R_2 &= t_2 . \bar{rv}_2 \uparrow . ra_2 \uparrow . \bar{rv}_2 \downarrow . ra_2 \downarrow . \bar{r}_2 . R_2 \\
 D_0 &= c_2 . r_2 . D_0 + d \uparrow . D_1 \\
 D_1 &= c_1 . r_1 . D_1 + d \downarrow . D_0. \\
 SWITCH &= (L|R_1|R_2|D_0|D_1) \setminus \{t_1, t_2, c_1, c_2, r_1, r_2\}
 \end{aligned}$$



### 3.1.2 3D Version

As seen in previous section, the switch module designed using *Petrify* is relatively complicated. So, we tried to design using another asynchronous tool, *3D*, for the switch module. Another difference than previous design is data latch in the switch module. Since the switch module may be close to the join module in one router, we decided to remove latches for data in switch module. It leads that one data latch stage for one channel exists in one router. This makes the switch design simpler and causes area reduction with removing data latches: in our current design specification, data is 9-bit wide and the size of 9-bit data latches are much bigger than that of the switch module (From the final layout of circuit, 9-bit latch module is 8 times bigger than the switch module). Figure 6 and 7 show the interface of the switch module and FSM of the switch design using *3D*.

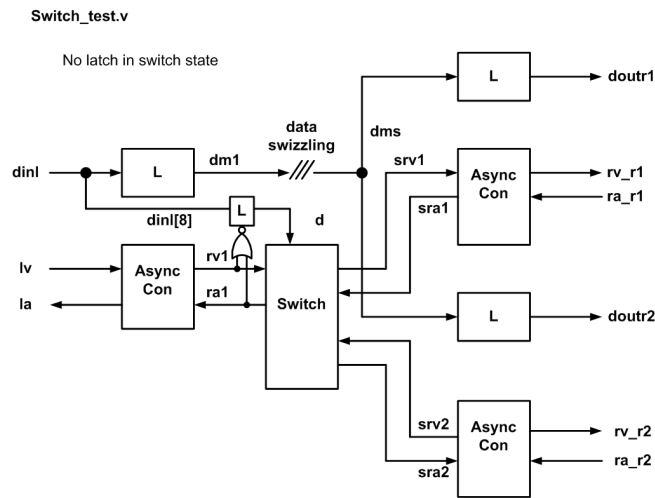


Figure 6: Interface of Switch Module, 3D Version

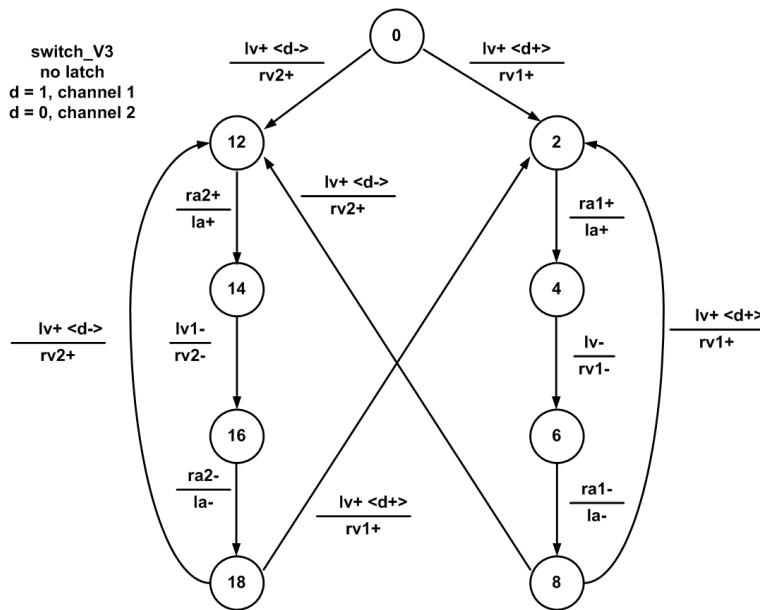


Figure 7: Extended Burst Mode FSM for Switch

The *3D* generates two types of circuits, standard cell and dynamic gate versions, like *Petrify* and the following logic equations are dynamic version of the switch module from *3D* with the specification,

figure 7. As reducing concurrency compared with the *Petrify* version, the circuit of *3D* is much simpler than *Petrify*.

- la  
 $SET(la) = ra1 + ra2$   
 $RESET(la) = ra1' ra2'$
- rv1  
 $SET(rv1) = lv d$   
 $RESET(rv1) = lv'$
- rv2  
 $SET(rv2) = lv d'$   
 $RESET(rv2) = lv'$

### 3.2 Join Module

Join module has two left-side input channel and one right-side output channel. This module arbitrates two requests from input channels and grant a right to access the right channel to only one input channel at a time. Figure 8 illustrates the interface of join modules. In the outside of join module, two requests, *mlv1* and *mlv2*, can occur at the same time or can be overlapped. But, one mutual exclusion element controls these two signals and allows only one request to be asserted inside of join module.

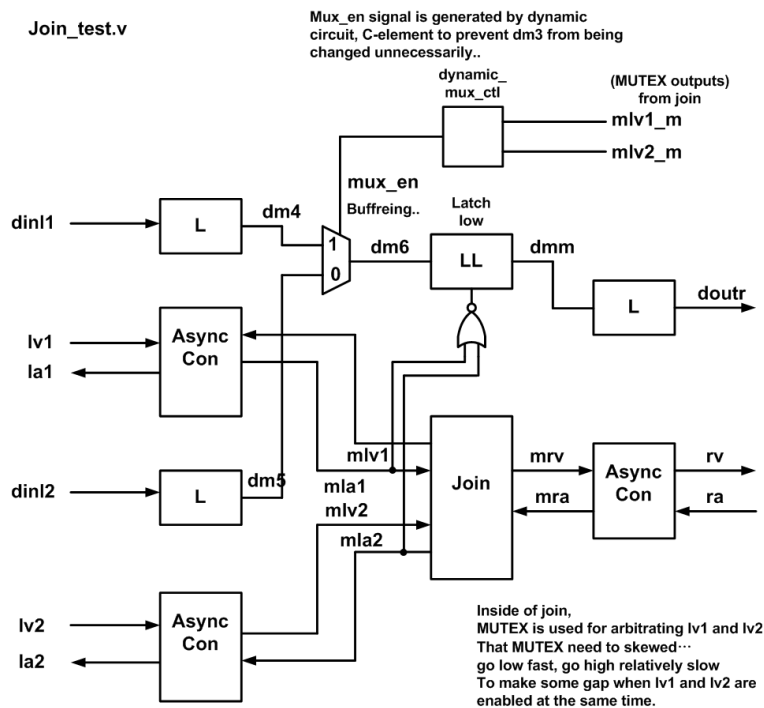


Figure 8: Interface of Join Module

The specification of the join module in CCS is

$$\begin{aligned}
L_1 &= t_1 .lv_1 \uparrow .\overline{c_1} .\overline{la_1} \uparrow .lv_1 \downarrow .\overline{la_1} \downarrow .\overline{x_1} .L_1 \\
L_2 &= t_2 .lv_2 \uparrow .\overline{c_2} .\overline{la_2} \uparrow .lv_2 \downarrow .\overline{la_2} \downarrow .\overline{x_2} .L_2 \\
R &= c_1 .\overline{rv} \uparrow .ra \uparrow .\overline{rv} \downarrow .ra \downarrow .R + c_2 .\overline{rv} \uparrow .ra \uparrow .\overline{rv} \downarrow .ra \downarrow .R \\
ME &= \overline{t_1} .x_1 .ME + \overline{t_2} .x_2 .ME \\
JOIN &= (L_1|L_2|R|ME) \setminus \{t_1, t_2, x_1, x_2\}.
\end{aligned}$$

A left channel is able to request an access of the right channel only with a grant from ME. After the operation of current left channel is completed, the other left channel can start its operation. The petri-net from the specification is shown in figure 9.

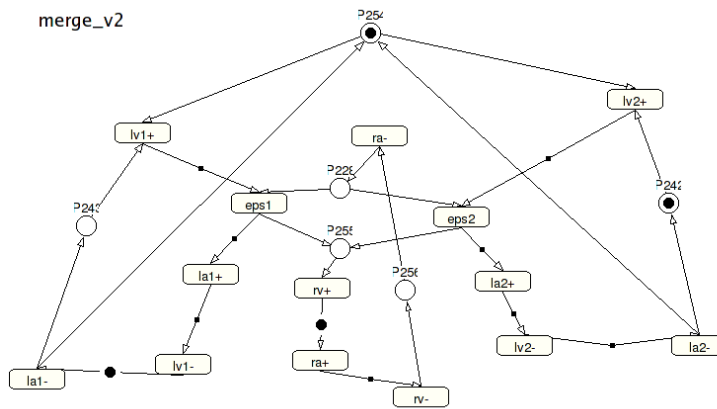


Figure 9: Petri-net for Join

The logic equations for output signals and a internal signal targeting dynamic gates from *Petrify* are as below. Relative timing assumptions were considered when these equations were generated. The input file, join.g, and timing assumptions are included in Appendix A.1.2.

- la1
  - SET(la1') = lv1'
  - RESET(la1') = lv1 csc0
- la2
  - SET(la2') = lv2'
  - RESET(la2') = lv2 csc0
- rv
  - SET(rv') = ra
  - RESET(rv') = ra' csc0 (lv1 + lv2)
- csc0
  - SET(csc0') = lv2' lv1' rv + ra
  - RESET(csc0') = la1' la2' ra' rv'

### 3.3 Design Issues

The switch module and join module were designed using *Petrify* first. But, the generated dynamic gates are still too complex to be built and to analyze operation timing. In some cases, more than 5 transistors need to be connected in series. It might cause an area problem in order to get a nominal FO4 delay. So, another tool, *3D*, designed for burst mode asynchronous circuits is used for getting simpler circuit than *Petrify*. As a result, for the switch module circuit, *3D* version circuit was chosen since it had simpler implementation with modifying specification. Meanwhile, the 3D version for the join module was similar to that of *Petrify* in circuit complexity. Therefore, for the join module, we decided to use *Petrify* version.

### 3.4 Circuit and Layout

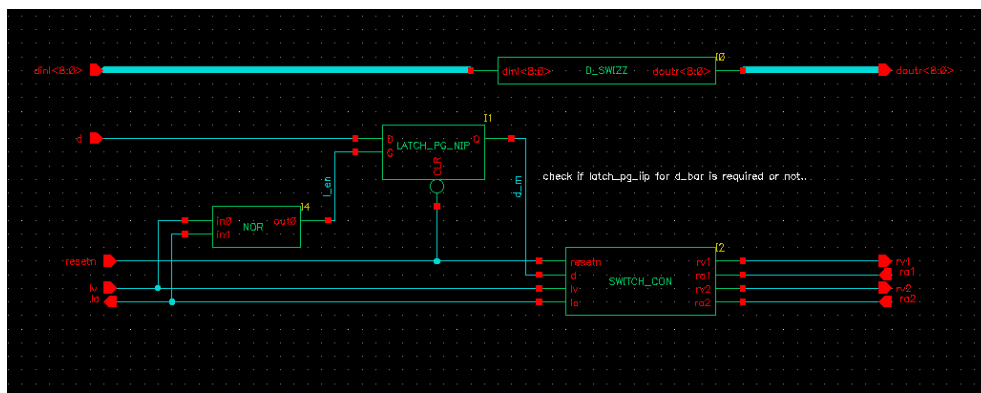


Figure 10: Circuit for Switch

We used 130nm technology for designing our circuit generated from asynchronous tools, *3D* and *Petrify*. For the transistor level circuit, IBM cmrf8sf tech. library was used and the standard cells are from University of Washington standard cell library, cg\_lib13\_se. All the transistor width are determined with basic widths, 560nm for pfet and 280nm for nfet, and logical effort. For the keepers in all dynamic gates, the width of pfet and nfet are the minimum size in this technology, 160nm. The schematics of switch module and join module were extracted and simulated using HSPICE for functional verification as well as timing analysis.

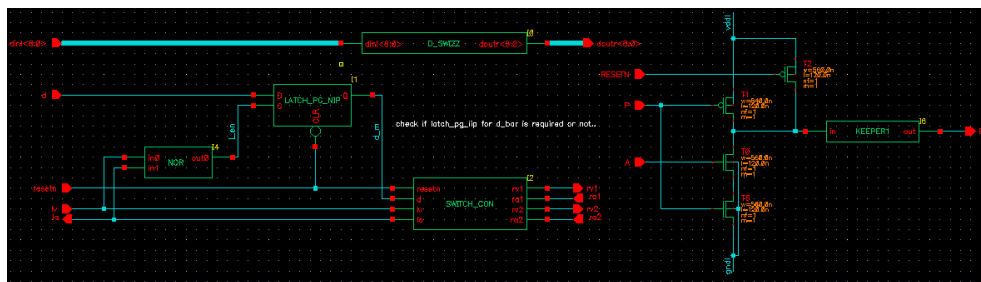


Figure 11: Circuit for Switch Controller

### 3.4.1 Switch Module

The switch module is mainly composed of two part, switch controller and data swizzling part. The data swizzling part is for data swizzling for addressing and it is simply made up of buffers.

The switch controller used two dynamic gates and three static gates and the dynamic gate, DFPA0, is a dynamic footed inverter as shown in figure 11.

The operation and timing of the switch module were verified with a test configuration, figure 12 and the simulation result is shown in figure 14. In the waveform, rv1 and rv2 are asserted according to lv and the MSB bit of input data dinl.

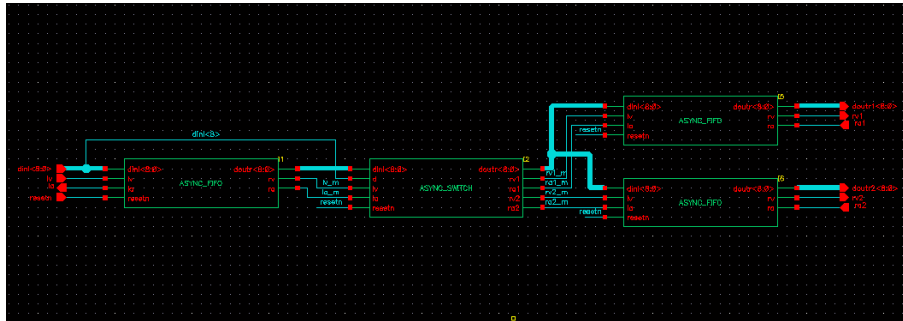


Figure 12: Circuit for Switch Module Test

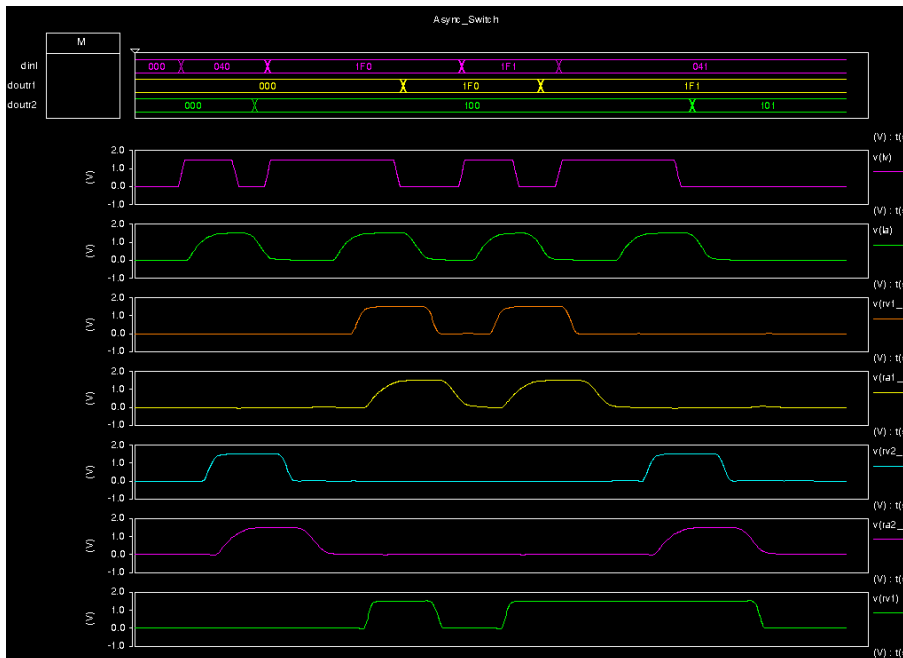


Figure 13: Simulation of Switch Module

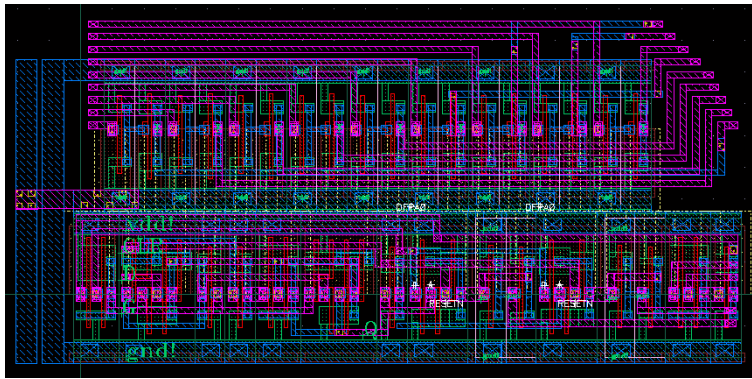


Figure 14: Layout of Switch Module

### 3.4.2 Join Module

The join module consists of join controller, join data part and latch enable circuit. 9-bit mux and latches compose the join data part to select one of input data and latch them into join data latch.

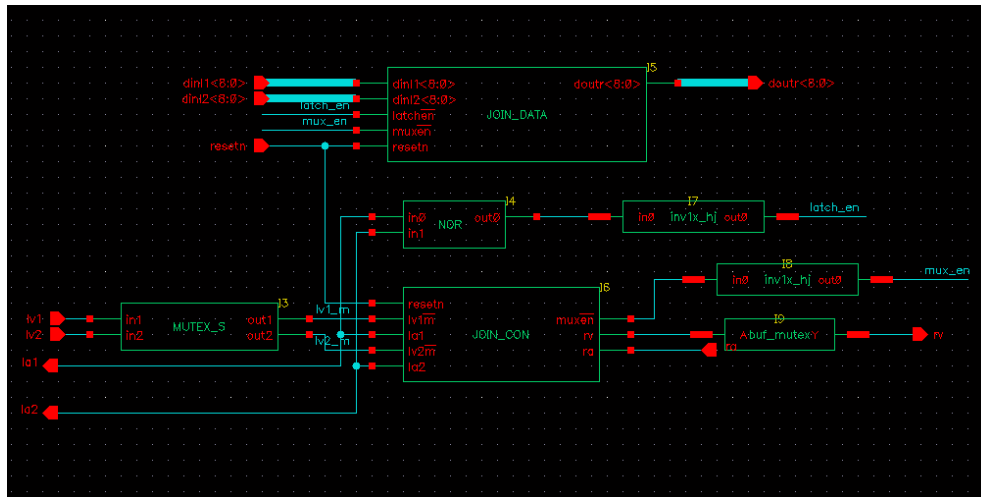


Figure 15: Circuit for Join

Five dynamic gates are used in the join controller and the circuits of two of them, JOIN\_RV and JOIN\_CSC0 are depicted in figure 16.

The operation and timing of the join module were verified and the simulation waveform is shown in figure 17.

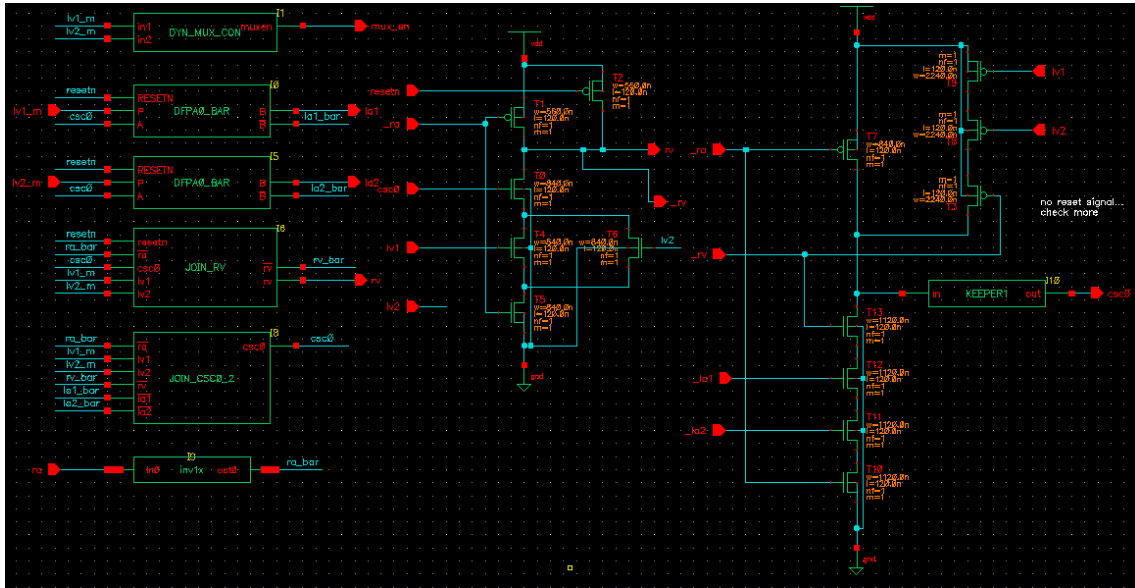


Figure 16: Circuit for Join controller

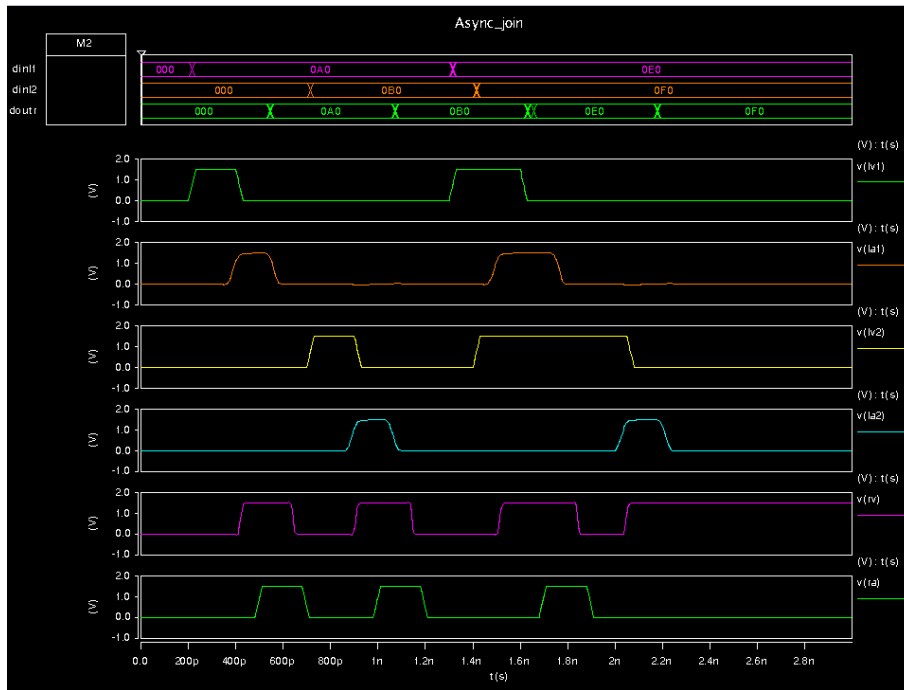


Figure 17: Simulation of Join Module

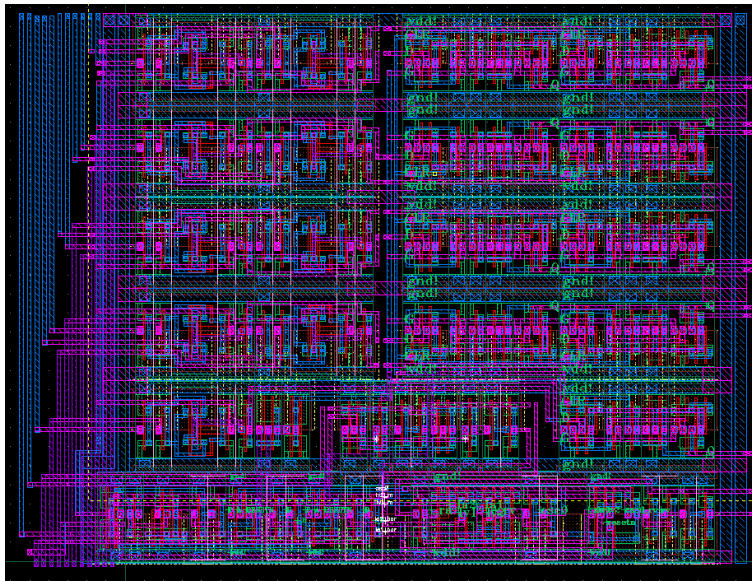


Figure 18: Layout of Join Module

### 3.4.3 Asynchronous Router

The router has three bidirectional ports and each port possesses one switch and one join modules. Thus, three switches and three joins are in one router. The area of one router is  $73 \mu\text{m} \times 62 \mu\text{m}$  ( $4526 \mu\text{m}^2$ ).



Figure 19: Layout of Async Router

## 4 Asynchronous FIFOs

In a previous project, synchronous FIFO which use Elastic Half Buffer and asynchronous FIFO which use asynchronous FIFO controller are used in 0.5 micro technology.

This project describes some alternative approached to building asynchronous flow through FIFOs that reduce the latency and power while retaining high throughput and relative simplicity of a flow through design. Four different designs are presented : a standard *linear FIFO* in which the data pass through every latch in the FIFO, a *parallel FIFO* in which data are delivered in turn to a set of parallel flow-through FIFOs, a *tree FIFO* in which data are fanned out into a tree of simple FIFOs, and a *square FIFO* in which the tree is organized as a square array to achieve better layout packing.

In a first part, some circuit modules are described in an aspect of design strategy. After we defined a specification in CCS, each modules are generated by petri-net and 3D which based on the specification. In a second part, the different structural FIFOs are compare in terms of power, latency and area. For exact comparison with same condition, we designed ten-deep nine bit wide FIFOs.

### 4.1 Asynchronous Circuit Modules

Five asynchronous modules which are linear, toggle 1x and 2x , and merge 1x and 2x modules are designed for different structural FIFOs. These asynchronous modules have a four phase handshake protocol and each module is defined by a specification in CCS and generated by petri-net or 3D.

#### 4.1.1 Linear Module

To build different structural FIFOs, first of all, simple linear FIFO module is required. The figure 20 shows block diagram and layout of linear module. As shown in this figure, linear module is consist of asynchronous controller and data latch and this real layout area is 22x24um in IBM 130 nm technology.

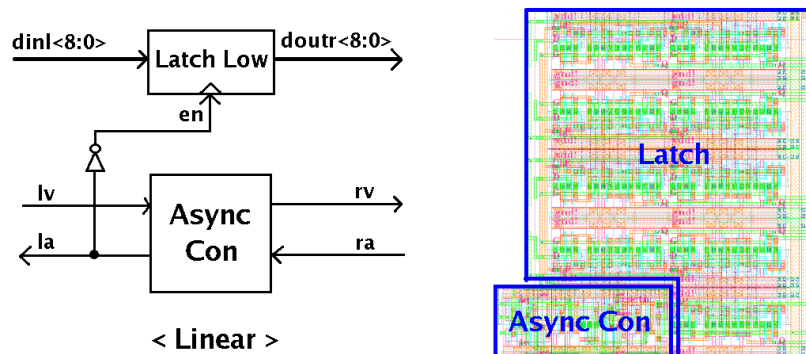


Figure 20: Linear Module

In linear FIFO controller, the input signals are  $lv$  and  $ra$  , and output signals are  $la$  and  $rv$  . When all signals are low, if  $lv$  is rising,  $la$  and  $rv$  are rising simultaneously. After  $la$  goes high  $lv$  can go low and after  $ra$  goes high  $rv$  can go low. In a data latch, we use normally open latch, so when only  $la$  goes high, the latch is closed.

A linear FIFO cell can be specified in CCS as follows:

$$L = lv \uparrow c \bar{la} \uparrow lv \downarrow \bar{la} \downarrow L$$

$$R = \bar{c} \bar{rv} \uparrow ra \uparrow \bar{rv} \downarrow ra \downarrow R$$

$$\text{FIFO} = (L|R)/\{c\}$$

The event  $c$  synchronizes the two processes, so as mentioned before,  $ra$  must go low and  $lv$  must rise before both processes may proceed.

According to above CCS format, we can get petri-net and state transition graph from petrify. Figure 21 shows petri-net and state transition graph

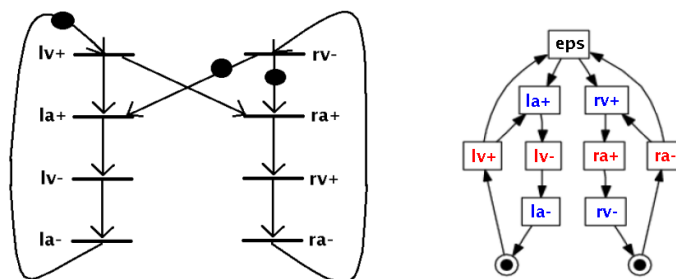


Figure 21: Linear Module's Petri-net and STG

#### 4.1.2 Toggle Module

In our system, two different operation toggle modules are required for different structural asynchronous FIFOs. The first toggle module's operation called toggle-1x is that the data is sent up and down but the second toggle module called toggle-2x operates that the data is sent up, down and down again. The toggle-2x is only used in square FIFO which will be explained the next chapter.

The figure 22 shows block diagram and layout of toggle 1x and 2x module. As shown in this figure, the toggle module is similar with linear module and real layout areas are 20x35um (toggle-1x) and 26x35um (toggle-2x).

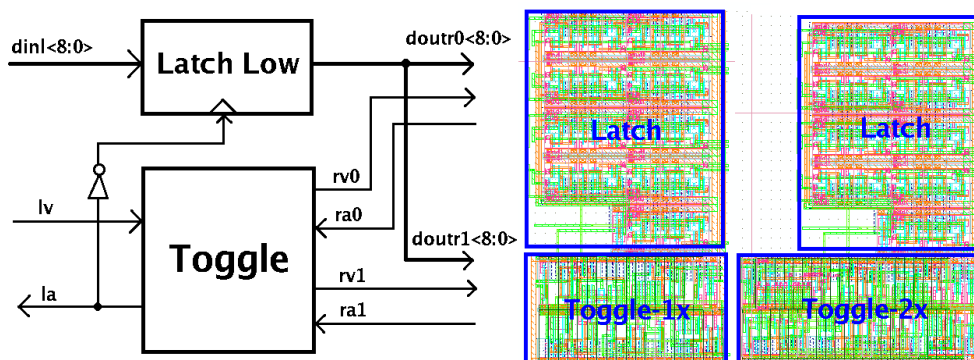


Figure 22: Toggle Module

Toggle module's operation is same with linear module's. The important specification is that after finishing all transition with upper channel, lower channel's transition can start. Since we use this

protocol, the normally open data latch can be used. In other words, if  $la$  or  $rv0$  didn't go low, the next transition couldn't be started, so even though we use normally opened latch, the data can not be overlapped.

A toggle FIFO cell can be specified in CCS as follows:

$$\begin{aligned}
 L &= lv \uparrow c_1 x_1 \bar{la} \uparrow lv \downarrow \bar{la} \downarrow lv \uparrow c_2 x_2 \bar{la} \uparrow lv \downarrow \bar{la} \downarrow L \\
 R1 &= \bar{c}_1 \bar{rv}0 \uparrow ra0 \uparrow \bar{x}_1 \bar{rv}0 \downarrow ra0 \downarrow R1 \\
 R2 &= \bar{c}_2 \bar{rv}1 \uparrow ra1 \uparrow \bar{x}_2 \bar{rv}1 \downarrow ra1 \downarrow R2 \\
 TOGGLE &= (L|R1|R2)/\{c_1, c_2, x_1, x_2\}
 \end{aligned}$$

The toggle's CCS has more synchronizer such as  $c1, x1, c2, x2$  compared with linear FIFO. Typically, if there are more synchronizer, the concurrency is more constrained. That means when there are fewer concurrency, circuit can be expressed simpler than opposite case, but timing constraint is increased. A various trial in petrify and 3D, we found proper petri-net and state transition graph between relationship of concurrency and timing constraint.

According to above CCS format, three different type of transition graph is depicted in figure 23.

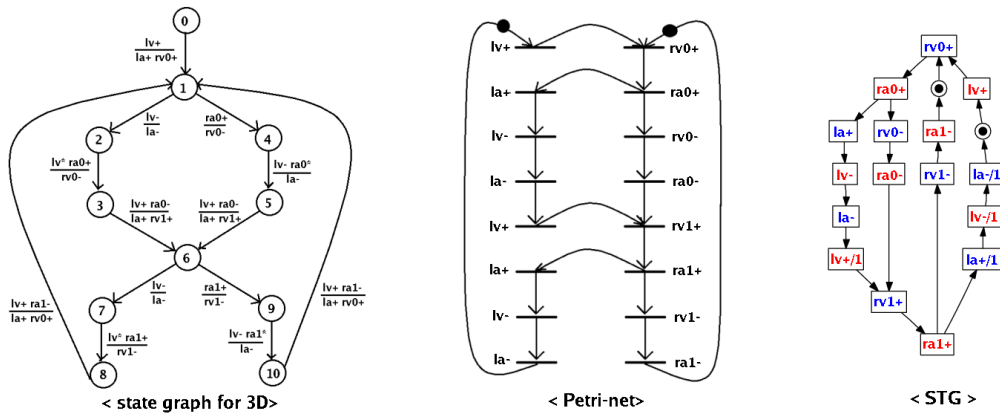


Figure 23: Toggle Petri-net and STG

In this figure, the first graph is specification of 3D, according to this state graph, toggle-1x is synthesized by 3D. The toggle-2x is synthesized by petrify. The reason for using different asynchronous circuit design tool to synthesize the circuit is that these tools generate different circuit in a same specification. In a some condition, 3D can generate simpler circuit than the petrify and vice versa. As a result, a toggle-1x is synthesized by 3D and toggle-2x is synthesized by petrify.

### 4.1.3 Merge Module

Two kinds of merge module is required for our asynchronous FIFOs. Merge module has two left channel and one right channel. This module receives two requests from left channel and send only one input channel at a time to right channel. This module didn't receive the input channel arbitrarily. According to sequence, the merge modules are operated. Merge-1x module receive left up channel and then left bottom channel. However, merge-2x module receive left up, down and down again. This merge-2x module is only required in square FIFO which will be explained in next chapter.

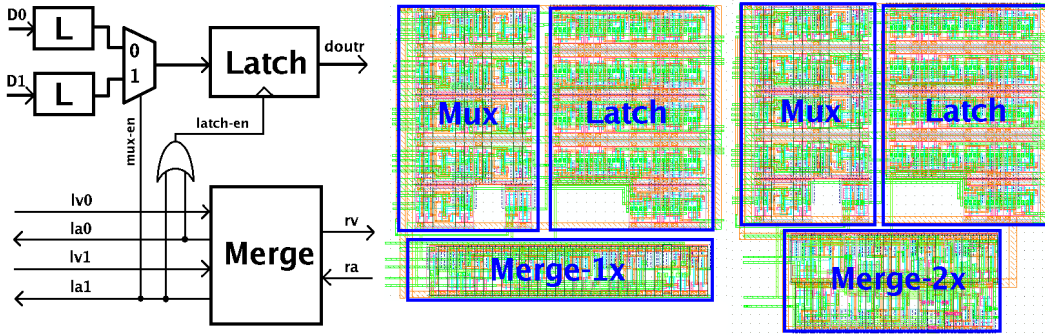


Figure 24: Merge Module

Figure 24 shows block diagram and layout of merge 1x and 2x module. As shown in this figure, merge module is similar with other module but only different thing is that Mux is required for data latch. Since we used normally opened data latch, when the both channel's request high in a left channel, the data can be overlapped. To prevent this circumstance, Mux is required. The merge 1x layout area is 35x31um and merge 2x area is 35x36um. Since this module required Mux, the size is relatively bigger than other modules.

Merge module's operation is similar with toggle module's. After the operation of current left channel is complete, the other left channel can start its operation.

A merge FIFO cell can be specified in CCS as follows :

$$\begin{aligned}
 L1 &= lv0 \uparrow c_1 x_1 \overline{la0} \uparrow lv0 \downarrow x_2 \overline{la0} \downarrow L1 \\
 L2 &= lv1 \uparrow c_2 x_3 \overline{la1} \uparrow lv1 \downarrow x_4 \overline{la1} \downarrow L2 \\
 R &= \overline{c_1} \overline{rv} \uparrow ra \uparrow \overline{x_1} \overline{rv} \downarrow ra \downarrow \overline{x_2} \overline{c_2} \overline{rv} \uparrow ra \uparrow \overline{x_3} \overline{rv} \downarrow ra \downarrow \overline{x_4} R \\
 \text{MERGE} &= (L1|L2|R)/\{c_1, x_1, x_2, c_2, x_3, x_4\}
 \end{aligned}$$

According to above CCS format, we can get petri-net and state transition graph from petrify.

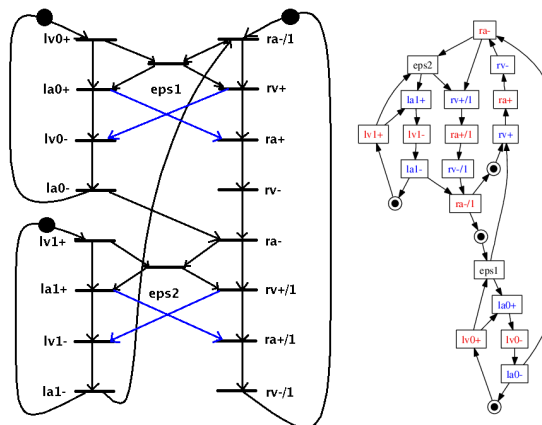


Figure 25: Merge Module's Petri-net and STG

Figure 25 shows the Petri-net and state transition graph. As mentioned before, 3D and Petrify generate different circuit in a same specification. In a case of merge module, the circuit from petrify is simpler than 3D, so the merge 1x and 2x modules are synthesized by petrify.

## 4.2 Different Structural FIFOs

In this project, we will compare different structural FIFOs for low latency and low power while retaining high throughput. In this chapter, four different type FIFOs will be described, linear, parallel, tree and square FIFOs. By combining above asynchronous circuit modules, we will compare a various FIFOs in ten-deep nine bit wide FIFOs in terms of latency, power and area.

### 4.2.1 Linear FIFO

First of all, ten stage simple linear FIFO is used.

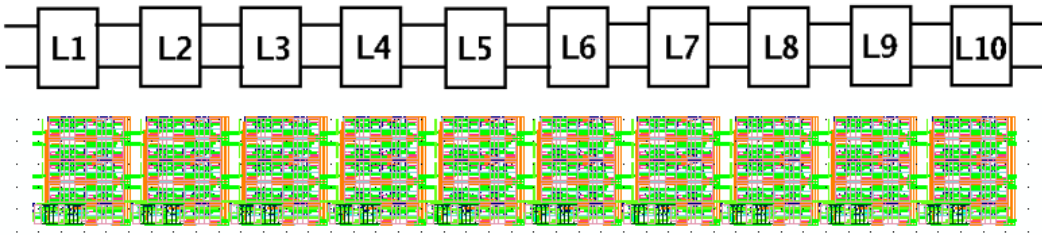


Figure 26: Linear Asynchronous FIFO

Ten stage of linear FIFO are shown in Figure 26. Each latch is a transition latch that captures bundled data in response to a transition  $la$ . Each FIFO stage acts as a concurrent process that will accept new data when the previous stage has data to give, and the next stage is finished with the data currently held. This is the FIFO circuit that is used as the basis for comparing each of the other FIFO designs.

### 4.2.2 Parallel FIFO

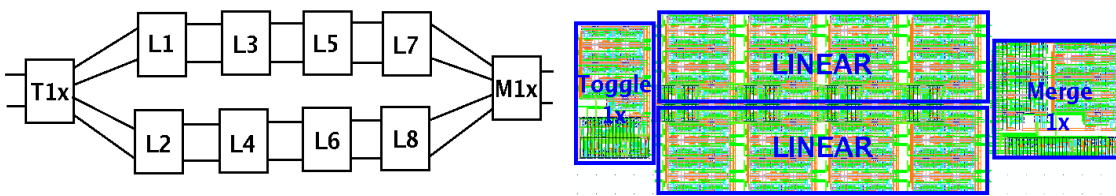


Figure 27: Parallel Asynchronous FIFO

Figure 27 shows the block diagram and layout of ten stage parallel FIFO. As shown in this figure, the parallel FIFO is consist of a toggle-1x, merge-1x and eight linear FIFO module. Since we use 1x of toggle and merge module, the data is delivered up and down sequenced. As we can estimate the result, the parallel FIFO reduces latency by a factor equal to the number of parallel FIFOs. In this FIFO, we use two number of parallel FIFO, the latency would be reduced by almost half of ten stage linear FIFO. However, since the extra circuit will have more latency in toggle and merge module, the whole latency is bigger than a half of linear FIFO's latency.

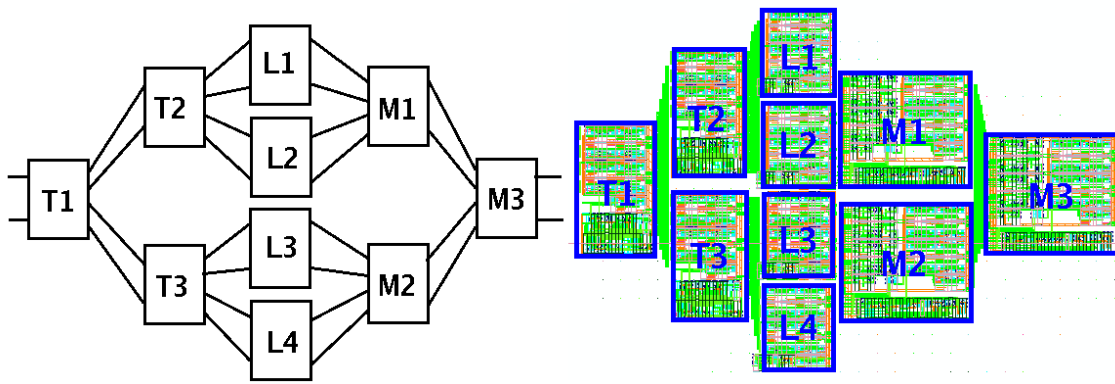


Figure 28: Tree Asynchronous FIFO

### 4.2.3 Tree FIFO

Figure 28 shows the block diagram and layout of ten stage tree FIFO. The tree structural FIFO consists of three 1x toggle and merge modules and four linear modules. A tree FIFO is essentially a parallel FIFO but one in which each of the parallel FIFOs are also parallel. The data are fanned out to a binary tree of FIFO cells, and then collected in another binary tree to a single output cell. The most important things in tree FIFO, the entire FIFO behaves exactly like a ten-deep FIFO, but data pass through only 5 stages. As a result, the latency would be the lowest among our four different structural FIFOs.

### 4.2.4 Square FIFO

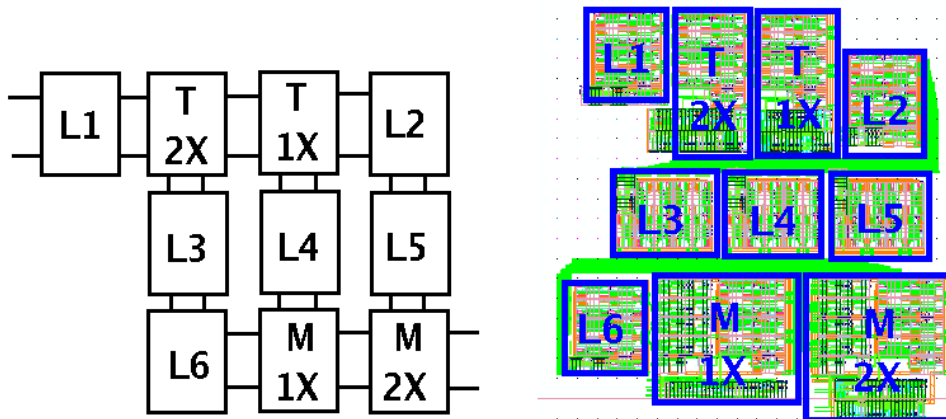


Figure 29: Square Asynchronous FIFO

Figure 29 shows the block diagram and layout of ten stage square FIFO. Square FIFO can compensate the drawback of tree FIFO, which the physical tree structure may not pack well onto the two-dimensional surface of an IC since rectangular shapes might pack better in terms of layout. As shown in this figure, however, each module doesn't have the same size, so it is not an exact rectangular shape.

In a square FIFO, the data pattern is a little bit different with other FIFOs. The first data goes all the way to the right end before dropping into the vertical FIFO in the middle. The second goes to

the second column and the next drops into the first column and the cycle repeats. So, toggle 2x is required in the first column and top row, and merge 2x is required in the third column and bottom row. It can make the bottom row collect data from the vertical FIFOs in the same order using the same idea. Even though tree's latency is more efficient than square's, the square arrangement may be more amenable to planar VLSI layout than the tree.

### 4.3 Performance Comparison

In order to evaluate the performance of the various FIFO organizations presented here, we used Hspice simulation with extracting file from cadence. For precise evaluation, first of all, we compared each module's power and size. Table 1 denotes the result of simulation.

Module	Power ( $\mu\text{W}$ )	% Power Increase	Size ( $\mu\text{m}^2$ )	% Size
Linear	387.1	0 %	2x24 (528)	0 %
Toggle 1x	467.8	20.8 %	20x35 (700)	32.6 %
Toggle 2x	551.4	42.4 %	26x35 (910)	72.3 %
Merge 1x	546.5	41.2 %	35x31 (1085)	105.5 %
Merge 2x	615.8	59.1 %	35x36 (1260)	138.6 %

Table 1: Each Module Power and Size

The percent power and size is a measure of how much the value of each FIFO modules higher than a simple linear FIFO. In this table, we didn't compare the latency because toggle and merge 2x is not symmetric circuit, so the latency of ( $lv0$ ,  $lv1$ ) and ( $rv0$ ,  $rv1$ ) is different. As we can see in this table, merge module's power and size is relatively bigger than others because of Mux. This size includes the data latch and asynchronous FIFO controller. Actually, the data latch's size occupied more than 70% of whole FIFO.

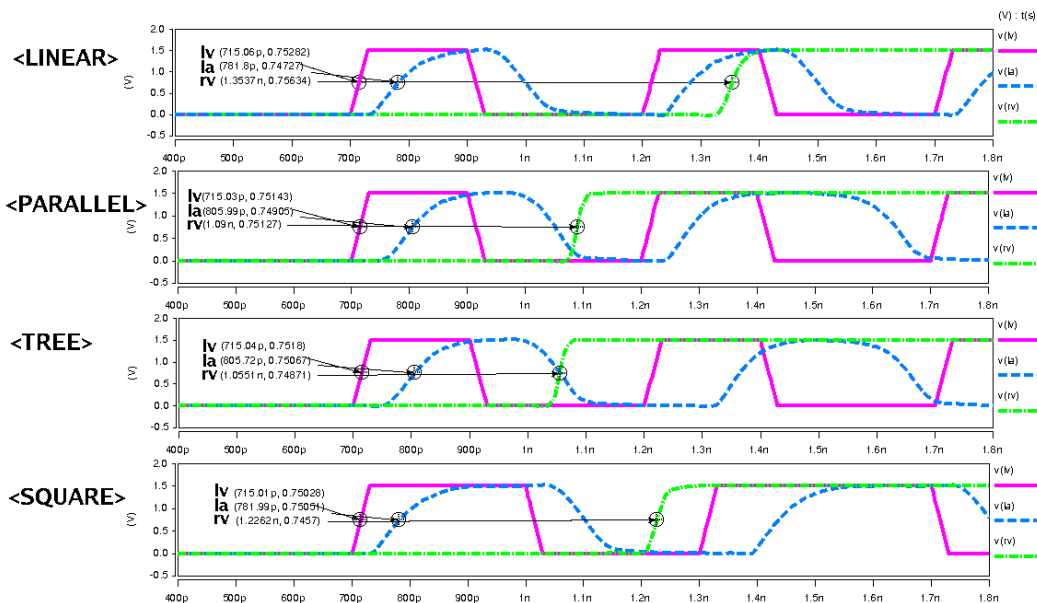


Figure 30: Various FIFO's waveform for latency

By combining of each module, we designed various FIFOs, linear, parallel, tree and square FIFOs. Figure 30 shows the simulation result in Hspice. In this figure, signal  $lv$ ,  $la$  is first stage's input and output, and signal  $rv$  is last stage's output. To measure the latency of various FIFO, we need to know when request signal  $lv$  comes into first stage, how long take the time signal  $rv$  goes up. As shown in this figure, parallel and tree FIFO's latency is very low. Even though the parallel and tree also have ten stage, when the FIFOs are empty, the data is through only 5 or 6 stage.

	Latency (ps)	% Latency	power (mW)	% Power	size ( $\mu m^2$ )	% Size
Linear	640	100 %	2.459	100 %	5280	0 %
Parallel	375	58.6 %	1.679	68.3 %	6010	13.8 %
Tree	330	51.6 %	1.479	60.1 %	7470	41.4 %
Square	510	79.7 %	1.546	62.9 %	7123	34.9 %

Table 2: Latency, Power and Size

The various FIFOs latency, power and area is summarized in Table 2. This table illustrates tree FIFO has the lowest latency and power, but drawback is size. The square FIFO has not low latency. An aspects of shape, however, the square FIFO's shape is almost rectangular. Even though the area is relatively big among a various FIFO, it has minimum length in its longest dimension(vertical and horizontal).

In our test-bench, we simulate these various FIFO to compare the efficiency with synchronous FIFO. However, since asynchronous FIFO is much faster than synchronous FIFO, it is hard to distinguish the latency among various asynchronous FIFO. The whole test-bench is used for measuring throughput and we get a result that each FIFO's throughput is almost same. Therefore, as we expected, we designed low latency asynchronous FIFO while retaining high throughput.

## 4.4 Conclusion

This project has explored some circuit for building various asynchronous FIFO that have lower latency than a simple linear FIFO. To reduce latency and power consumption, fewer signal transitions are needed to pass data through the FIFO.

The linear FIFO uses a distribution to compare with other various FIFO, so we use simple linear FIFO which from Dr. Stevens linear FIFO. The parallel shows that the latency is reduced by a factor equal to the number of parallel FIFOs the data are distributed into. If we designed three way toggle in front of parallel FIFO, the latency might be reduced by one third. Tree FIFO distributes the data into a binary tree of FIFOs. Even though the layout shape is not rectangular, this has the lowest latency and power consumption. Square FIFO is similar with tree FIFO, but there are big difference. Data are stored in the top and bottom FIFOs as those cells are doing the distribution. In addition, the shape is fit well in terms of VLSI layout.

In our SoC design, we don't need many stage FIFO. In a previous project, we only use 2 stage FIFO, but to compare various FIFO in a same condition, we use ten stage FIFOs. According to traffic signal from PE, if we need from 4 to 8 stage FIFO, the parallel FIFO would be efficient due to low latency and low power consumption.

# 5 Behavioral Validation & Results

## 5.1 Golden Model & Testbench

Golden model and testbench were designed targeting the system shown in figure 31. It consists of one router, 6 interface circuits and three ports, A, B and C, each of which has sending and receiving ports.

Golden model written in C language generates random data for three ports. All source and destination of data transfer, stall condition, and data transfer rate are randomly decided in each simulation. These data are stored in text file.

Testbench works like PEs which send or receive data with each other. Testbench read data from the text file that the golden model generates and it makes the three sending ports to send data to the destination receiving port. On the other hand, the testbench reads data from the three receiving ports and store them into appropriate output files. After a simulation is done, the output files from the testbench are compared with the input files from the golden model. Testbench is written in tcl and the simulations were performed using Modelsim.

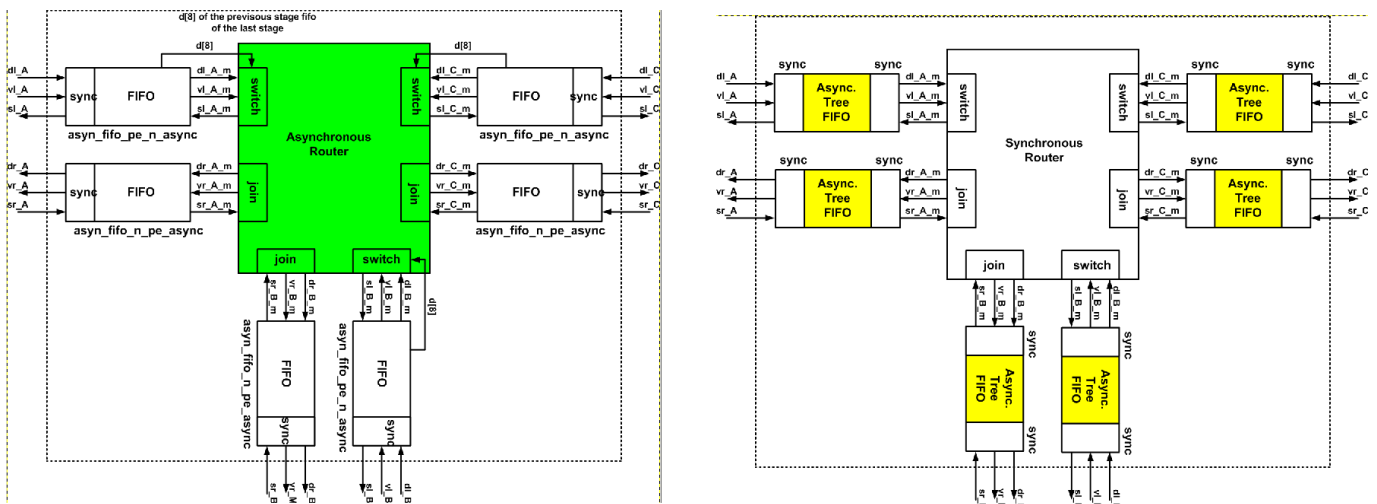


Figure 31: Testbench Model

These golden model and testbench were designed and used in the previous project in which the router is synchronous version and interface circuits used linear asynchronous FIFO. We replaced corresponding modules for the behavioral validation of the asynchronous router and the asynchronous tree-FIFO. Verilog behavioral models for the asynchronous router and the tree FIFO were written from the selected circuits in *Petrify* and *3D* output log files. Test results showed the behavioral models of the router and the tree FIFO work correctly.

## 6 Conclusion & Further Researches

In this project, we designed an asynchronous router and various asynchronous FIFOs. Two asynchronous tools, *3D* and *Petrify* were used to design asynchronous modules and circuits and layout were designed using IBM 130nm technology. The functionalities of the design was verified with

verilog behavioral model and ModelSim simulation. The timing analysis of operation of circuits were performed in HSPICE simulation.

The asynchronous router is a main module composing asynchronous network fabric to interconnect processing elements or IP blocks in SoC system. As one of following researches of this project, the operation and performance of the asynchronous router need to be verified and evaluated with other modules, Processing Elements and interface circuits, which are not currently available in 130nm technology. Comparison of this asynchronous router with the synchronous version from the previous project may be another valuable research.

Various asynchronous FIFOs, such as linear, tree, parallel, and square FIFO were designed and analyzed as well. As an aspects of latency, power consumption, and area, we compared various FIFO with simple linear FIFO. By combining linear, toggle 1x and 2x , and merge 1x and 2x, we designed various FIFOs. As a result, tree FIFO has the lowest latency and power consumption, but it occupied relatively big area. Therefore, in our SoC system, the parallel FIFO is more efficient than others since it has lower latency and power consumption and the size increases only 10% with simple linear FIFO. However, to use the most appropriate FIFO in an interface system, it depends on which part is more weighted than others, such as latency and power consumption. In further research, we need to design more simple circuit by reducing concurrency and investigate more efficient FIFOs for instance three way parallel FIFO or folded FIFO.

Our system will be fabricated in both 130nm and  $0.5\mu\text{m}$  in near future. For fabricating our asynchronous interconnect network system, the rest of modules such as PE and synchronizers are supposed to be designed in 130nm technology which can be possible only after synthesis and APR process are available in 130nm technology. And, only the asynchronous router is required to be redesigned for  $0.5\mu\text{m}$  chip

## References

- [1] T. Bjerregaard and S. Mahadevan. *A survey of research and practices of network-on-chip*. ACM Computing Surveys, 38(1), 2006.
- [2] Amde, M., Felicijan, T., Edwards, A. E. D., and Lavagno, L. *Asynchronous on-chip networks*. IEE Proceedings of Computers and Digital Techniques 152, 273-283, 2005
- [3] R. Milner. *Communication and Concurrency*. Prentice-Hall, London, 1989
- [4] Kenneth S. Stevens, Ran Ginosar, and Shai Rotem *Relative Timing*. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 11(1), Feb. 2003, pp. 129-140.
- [5] Erik Brunvand *Low Latency Self-Timed Flow Through FIFOs*. 16th Conference on Advanced Research in VLSI, Chapel Hill, NC, March 1995.

# A Petrify Input Files

## A.1 Asynchronous Network Fabric Modules

### A.1.1 Switch.g

```
.model switch_v
.inputs ra2 lv ra1 d
.outputs rv2 la rv1
.dummy eps2 t2 t1 eps1
.graph
d+ P174
d- P204
ra1- P174 P227
P174 d- t1
rv1+ ra1+
ra1+ rv1-
rv1- ra1-
lv+ P217
la+ lv-
lv- la-
la- P218
eps1 rv1+ la+
t1 eps1
P204 d+ t2
la+/1 lv-/1
lv-/1 la-/1
la-/1 P218
eps2 la+/1 rv2+
t2 eps2
ra2- P204 P227
rv2+ ra2+
ra2+ rv2-
rv2- ra2-
P217 eps1 eps2
P218 lv+
P227 t1 t2
.marking { P174 P218 P227 }
.slowenv
.time la+<|ra1+ # concurrency reduction (automatic)
.time la+/1<|ra2+ # concurrency reduction (automatic)
.time rv1+<|lv- # concurrency reduction (automatic)
.time rv2+<|lv-/1 # concurrency reduction (automatic)
.end
```

## A.1.2 Join.g

```
.model merge
.inputs lv2 lv1 ra
.outputs la2 la1 rv
.dummy eps1 eps2
.graph
ra- P228
rv+ ra+
ra+ rv-
rv- P256
lv2+ eps2
la2+ lv2-
lv2- la2-
la2- P242 P254
lv1+ eps1
la1+ lv1-
lv1- la1-
la1- P243 P254
P228 eps1 eps2
eps2 la2+ P255
eps1 la1+ P255
P242 lv2+
P243 lv1+
P254 lv1+ lv2+
P255 rv+
P256 ra-
.marking { P242 P243 P254 P256 }
.slowenv
.time rv+<|lv1-          # concurrency reduction (automatic)
.time rv+<|lv2-          # concurrency reduction (automatic)
.time la2+<|ra+          # concurrency reduction (automatic)
.time la1+<|ra+          # concurrency reduction (automatic)
.end
```