Arbiter for an Asynchronous Counterflow Pipeline

James Copus

This document attempts to describe a special type of circuit called an arbiter to be used in a larger design called counterflow pipeline. First, a brief introduction of the background of the subject will be described, including use of the arbiter in the system. The design specifications will then follow, along with a suggested implementation for the circuit. Finally, possibilities for addition, revision, and testing structures for an integrated circuit implementation of the arbiter will be proposed.

Counterflow Pipelines

A counterflow pipeline is a radically different way of designing a microprocessor. A good source for a detailed background of counterflow pipeline processors can be found in [1], but in general, microprocessor instructions flow in one direction down the pipeline and the results and register data flow in the other direction down the pipeline. Figure 1 shows a simple counterflow pipeline diagram.

The instructions flowing up the pipeline collect the data it needs to execute from the results pipeline, and after the instructions execute, the results are inserted into the pipeline to go to later instructions that need the data. All of these interactions happen asynchronously, meaning that there is no system clock controlling the data flow and operations. This asynchronous operation requires a different methodology than what is usually taught in digital design. Also, since the counterflow pipeline's instruction data and result data must interact at each stage, there must be some control to ensure that every instruction meets every result without any skipping occurring.

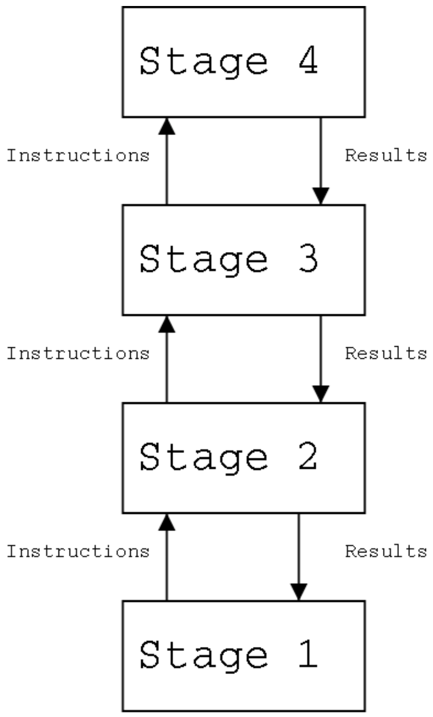


Figure 1: Simple Counterflow Pipeline

Arbiter

To control the flow between the stages of the pipeline, an arbiter is used to prevent the instruction data and the result data from moving at the same time and handling requests to transfer data between the stages. This arbiter works without a clock, and must be able to handle requests occurring according to specifications that will be defined later. Figure 2 shows a simple arbiter and how it fits into the pipeline path between two stages.

Arbiter Specifications

The arbiter that will be built is the same one as shown in Figure 2. Below is the explanation of the signal lines labeled in the arbiter figure.

RI: Request Receive Instruction – This signal is asserted when a pipeline stage is ready to receive a new instruction.

SI: Request Send Instruction – This signal is asserted when a pipeline stage is ready to send its instruction to the next stage.

GI: Grant Instruction Transmission - The arbiter asserts this signal when both RI and SI are asserted AND there is no transmission occurring in the results pipeline at the time.

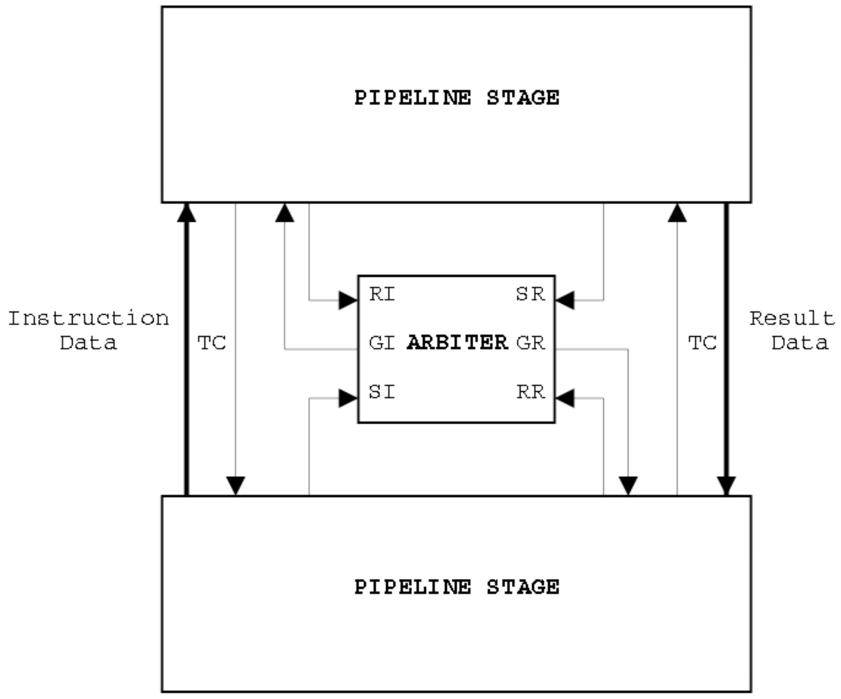


Figure 2: Counterflow Pipeline Arbiter and Signal Lines

RR: Request Receive Result – This signal is asserted when a pipeline stage is ready to receive a new set of results.

SR: Request Send Result – This signal is asserted when a pipeline stage is ready to send its result data to the next stage.

GR: Grant Result Transmission – The arbiter asserts this signal when both RR and SR are asserted AND there is no transmission occurring in the instruction pipeline at the time.

Instruction Data: This line represents a large bus containing all the data representing the instruction data. Note that the arbiter does not directly use this data line and is therefore outside the scope of the specific arbiter design.

Result Data: This line also represents a large bus containing all the results data. This is also outside the scope of the specific arbiter design.

TC: Transfer Complete – These are acknowledge signals used to indicate to the sending stage that the receiving stage has finished receiving and processing the data. Since the arbiter does not directly see these signal lines, it does not need to be accounted for in the arbiter design.

Arbitration Rules

The rules for the arbiter are rather simple. For permission to transmit data to be granted, both stages must request it. Therefore, RI and SI must both be high for GI to be asserted. In the same way, RR and SR must both be high for GR to be asserted. After the arbiter grants the request, the other side of the pipeline cannot send data until the transfer is complete. For the transfer to be complete, both stages must set their request lines low. The granting signal for that side will be set low, and another transmission can take place. If both sides request transmission at the same time, the arbiter must decide which one to grant first. Note that the granting signals, GR and GI cannot be asserted at the same time, which means that they are mutually exclusive. This prevents the two sides of the pipeline from exchanging data at the same time. If this simultaneous exchange were to happen, then the data being transferred would not be able to interact with each other in a pipeline stage. A sort of "state table" for the circuit is included in Table 1 below.

Instruction Transfer			
RI	SI	GI	GR
0	0	0	X
0	1	no change	X
1	0	no change	X
1	1	1	0
Result Transfer			
RR	SR	GR	GI
0	0	0	X
0	1	no change	X
1	0	no change	X
1	1	1	0

Table 1: State table showing the results of some common request signal values. Note that 'X' represents a value that does not depend on the current listed inputs.

In the design proposed, the decision-making is handled so that the instruction and result transmissions are granted randomly. Hopefully, the decision is made so that about half the time GR is asserted, and the other half GI is asserted. Figure 3 shows a transistor and logic gate level schematic of the mutual exclusion circuit that can handle this function. Assuming that R1 and R2 represent requests for different actions (such as the instruction or result transmission requests), G1 and G2 are the "granting" signals for the R1 and R2 requests, respectively. The mutual exclusion circuit is designed so that when one of the requesting signals goes to logic '1', the granting signal corresponding to that signal goes high and the other granting signal is forced low until the original requesting signal goes back to '0'. If both requests occur at the same time, the circuit becomes *metastable*, with the outputs of both NAND gates being at halfway between the supply voltage and ground. While these "undefined" values exist, the circuit will keep the granting signals at logical '0'. Eventually, one of the metastable sides of the circuit will settle to a voltage past the threshold voltage of the transistor, forcing one of the grant signals high and the other grant signal low. Thus a "decision" is made as to which request to grant. A more complete explanation of this circuit can be found in [2].

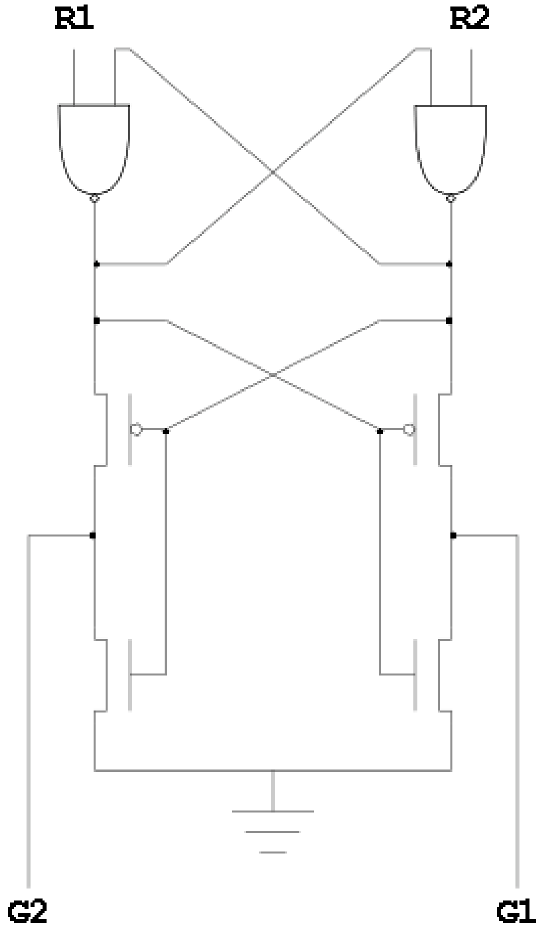


Figure 3: Mutual Exclusion Circuit (MUTEX)

State-Holding Elements

Since both RI and SI or RR and SR must be de-asserted before the grant signals can be de-asserted, it would be helpful to use some sort of state-holding element that can only be reset when two input signals are low. Since the design is asynchronous, normal clocked flip-flops cannot be used. However, a circuit called a Muller C-Element [3] is a standard state-holding element used in asynchronous designs. The symbol for a C-element is shown in Figure 4. A transistor-level schematic and "state table" is shown in Figure 5. Note that there is a feedback inverter that provides the "state-holding" ability of the circuit. This inverter is sized to be overridden when the input signals are

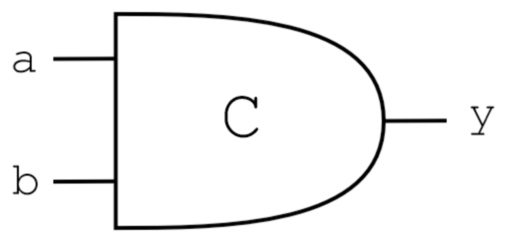
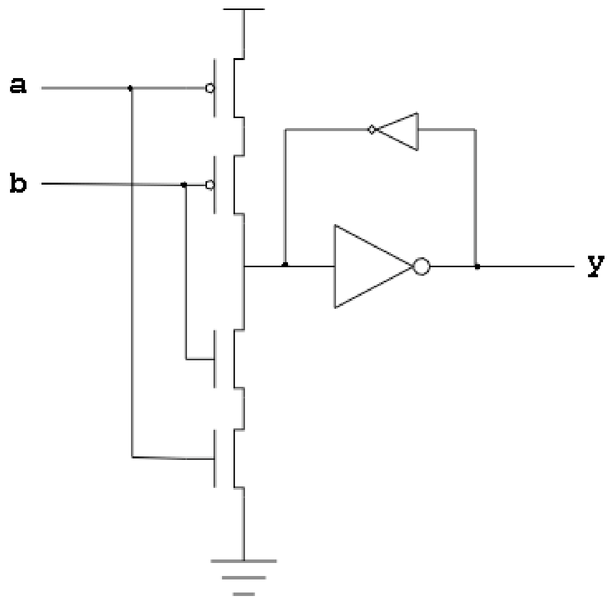


Figure 4: Muller C-element

both high or both low, so care must be taken during the design of a C-element so that the transistors are sized to function properly.



a	b	y
0	0	0
0	1	no change
1	0	no change
1	1	1

Figure 5: Schematic and specification of a C-element.

The RI/SI and RR/SR signals can be fed into the C-elements and then go into the mutual exclusion circuit described previously, as shown in Figure 6. This circuit should successfully implement the complete asynchronous arbitration circuit.

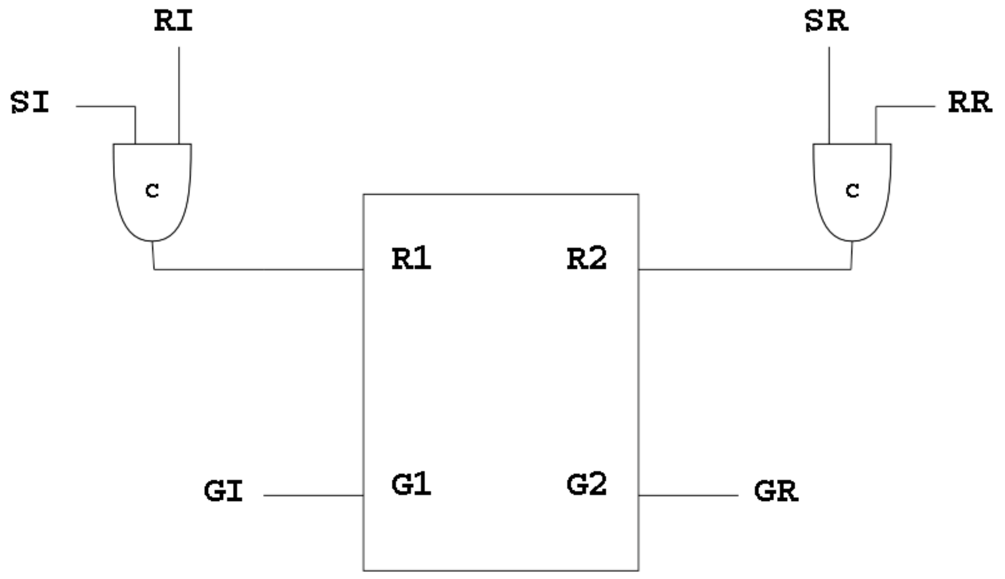


Figure 6: Full design of asynchronous arbiter

Proposal

The arbitration circuit described above should be designed and tested to verify that it functions as specified. To aid in testing of the fabricated circuit, multiple instances of the arbiter could be layed out. Additional test circuitry should be added to allow some instances of the arbiter have some self-test functions. The schematic of the arbiter described above is just one way to implement the arbitration rules; it is not necessarily the only way. Other functionality could be added to the arbiter, such as a reset signal.

To understand more about the background of asynchronous circuits and specifically counterflow pipelines, read the sources provided. I can provide access to the sources, along with additional resources. Any questions or comments are welcome.

SOURCES:

[1] R. F. Sprowll, I. E. Sutherland, and C. E. Molnar, “The Counterflow Pipeline Processor Architecture,” *Design & Test of Computers, IEEE,* vol. 11, pp. 48-59, Autumn/Fall 1994.

[2] Jens Sparso and Steve Furber, *Principles of Asynchronous Circuit Design: A Systems Perspective*, chapter 5.8: Mutual exclusion, arbitration, and metstability, pp. 77-80, European Low-Power Initiative for Electronic System Design, Kluwer Academic Publishers, Boston, 2001.

[3] Jens Sparso and Steve Furber, *Principles of Asynchronous Circuit Design: A Systems Perspective*, chapter 2.2: The Muller C-element and the indication principle, pp. 14-16, European Low-Power Initiative for Electronic System Design, Kluwer Academic Publishers, Boston, 2001.