

# UC Berkeley Computer Science

**Subject:** Snippets for a Three-input Adder  
**Date:** November 20, 2005  
**From:** Ivan Sutherland  
**UCIES** #2005-is13

## References:

UCIES# 2005-is02: FLEET – A One-Instruction Computer, Ivan Sutherland, 24 August 2005  
UCIES# 2005-is03: Defining Some SHIPs, Ivan Sutherland, 24 August 2005  
UCIES# 2005-is04: A Dozen Problems, Ivan Sutherland, 6 September 2005  
UCIES# 2005-is05: Notation Questions, Ivan Sutherland, 12 September 2005  
UCIES# 2005-is06: Some Ideas About Notation, Ivan Sutherland, 13 September 2005  
UCIES# 2005-is07: Literals for FLEET, Ivan Sutherland, 20 September 2005  
UCIES# 2005-is08: Function or Addressing, Ivan Sutherland, 20 September 2005  
UCIES# 2005-is10: More About Literals, Ivan Sutherland, 28 October 2005  
UCIES# 2005-is11: Indirection for Memory Read and Write, Ivan Sutherland, 28 October 2005  
UCIES# 2005-is12: Four Views of FLEET, Ivan Sutherland, 2 November 2005

## INTRODUCTION

This memo offers a description of the external behavior of a three-input adder SHIP. The figures represent the behavior in the form of snippets of Petri nets. In order for an event to occur, each and every snippet in which that event appears must permit it. If an event appears more than once in a single snippet, however, it is permitted if any of its appearances is permitted. Properties of the SHIP are drawn in black in the figures; necessary input environment behavior appears in red, and necessary output environment behavior appears in green.

## DISCUSSION

The three-input adder described here acts like a two stage pipeline. The internal stages are represented as four rectangles in the block diagram and their actions carry the subscript “i”. The adder takes three numeric inputs, **A B C**, and produces a numeric sum, **S**. The sum may either be the actual sum of the three inputs or an Out-Of-Band value if any of the inputs are Out-Of-Band. The output interface is said to be “passive” because the output there, **S**, appears only after receipt of a token at **T**. An Out-Of-Band token, **Tx**, at the output interface is ignored.

## BLOCK DIAGRAM – Figure 1

Figure 1 is a block diagram of the three-input adder. The input interface is on the left and the output interface is on the right. There are two internal ranks of latches, **Ai Bi Ci** and **Si**; the designation “i” means internal. The **Si** latch will hold either data representing a sum or an Out-Of-Band value as indicated. The figure omits the adder itself.

The input interface accepts three numeric inputs, **A B C**, and returns an acknowledge token, **M**, or an Out-Of-Band token, **Mx**. Although Out-Of-Band values may arrive on inputs **A B C**, the figures omit them. In addition, the four short arrows with double letter names represent the local acknowledge signals for the switch fabric.

---

This document is a product of a collaboration between Sun Microsystems and the University of California at Berkeley.. The ideas contained herein are freely available for any academic purpose.

The output interface produces output **S** and accepts a token input **T** or an Out-Of-Band token input **Tx**. Again the short arrows with double letter names represent local acknowledge signals for the switch fabric.

The output **S** might be Out-Of-Band under two conditions. First, one of the inputs may be the Out-Of-Band value called LAST, in which case the output will similarly be LAST. Second, the sum may not be representable and thus produce an Out-Of-Band value indication overflow. In either case the value is whatever is latched in the **Si** latch.

## INPUT INTERFACE – Figure 2

Figure 2a describes the minor input interface. Actions of the three-input adder SHIP appear in black; actions of the input environment appear in red.

The leftmost snippet indicates that **A Ai** and **aa** occur in sequence. The dot indicates that the environment gets the first turn. Although **aa** appears later in sequence than **Ai**, delay between them is optional. The snippet requires only that **aa** occur only if **Ai** occurs. In fact **aa** and **Ai** could be the very same event serving not only to latch the data from **A** but also to acknowledge that action to the switch fabric. The snippets for **B** and **C** are identical. Notice that these snippets avoid representing Out-Of-Band data.

The rightmost snippet in Figure 2a describes the behavior of the input interface token output **M**. **M** and **Mx** are mutually exclusive and must alternate with the acknowledge signal **mm**. Here, however, the dot indicates that the adder takes the first turn. The distinction between **M** and **Mx** is probably a matter of the type of token delivered, i.e. it's a data matter. I'm exhibiting it here as a control function to emphasize the presence of Out-Of-Band tokens.

Figure 2b describes the broader behavior expected of the environment. New inputs **A B C** must, as a group, follow receipt of tokens **M** or **Mx**. Although the environment is free to initiate inputs, it should not send a second set of inputs until the adder has returned an **M** or **Mx** token corresponding to the first set of inputs. The programmer is responsible for ensuring this behavior of the environment.

## OUTPUT INTERFACE – Figure 3

Figure 3 represents the behavior of the output interface. Here constraints on the environment appear in green.

The two snippets at the left, Figure 3a, represent the minor behavior of working with the switch fabric. Notice that the environment gets the first turn at the token input, **T** or **Tx**, while the three-input adder gets the first turn at the sum output, **S**. The switch fabric acknowledge actions are **tt** and **ss**.

The snippet at the bottom right of the page, Figure 3b, represents the pipeline behavior of the output interface. The environment gets the first turn to produce **T** or **Tx**, which are mutually exclusive, but if it issues a **T**, it must not produce another **T** or **Tx** until after the adder produces an **S**. The snippet indicates that the adder ignores inputs **Tx**; the environment is free to produce a string of **Tx** but only if after completing any interactions involving **T** and **S**.

## TOKEN OUTPUT **M** or **Mx** – Figure 4

Figure 4 indicates that all three internal data latches, **Ai Bi Ci**, must have captured data before the adder can return **M** or **Mx**. Together with Figure 2 this requires that the input actions **A B C** alternate with the response action **M** or **Mx**. The snippet is silent on the choice between **M** and **Mx** because that is a matter involving data analysis rather than sequence. The snippet requires only that they be mutually exclusive.

## ADDITION REQUIRES THREE INPUTS – Figure 5

This snippet is rather like Figure 4. This snippet insists that all input data be latched before the output latch captures its value. The block diagram indicates that the **Si** latch will capture either the sum data from the adder equipment (not shown) or an Out-Of-Band (OOB) value. This snippet is silent about the choice between sum data and OOB value because that is a matter of data analysis and not sequence.

## LATCH NEW DATA AFTER OUTPUT – Figure 6

Figure 6 relates the three input latches **Ai Bi Ci** to the output latch **Si**. Together with Figure 5 it indicates that the input group **Ai Bi Ci** alternate with the output latch **Si**. The dots in figure 6 indicate that the input latches get the first turn.

## THREE ALTERNATIONS

This is a two stage pipeline. There are three alternations involved, one at the input interface, one at the output interface, and one between the input latches and the output latches. The input interface alternation is clear from Figures 2 and 4. The output interface alternation is clear from Figure 3. The internal alternation appears in Figures 5 and 6. These alternations are characteristic of pipeline systems.

## LONGER PIPELINE – Figure 7

One might wish to have a longer internal pipeline. That could be done as shown in Figure 7 by adding additional internal pipeline latches. The upper figure calls the extra latches at the input **Aj Bj Cj**; the lower figure suggest combing the three input data paths into one register called **Xj**. Both figures add an output register called **Sj**.

The control for these latches follows the pattern we've already seen. With three separate input pipelines of the upper figure, **Ai** and **Aj** must alternate, with **Ai** getting the first turn, and the same for **Bi** and **Ci**. **M** or **Mx**, however, will require only that the first rank of latches all contain data, whereas **Si** will require that the second rank all contain data.

With a combined input pipeline, **Xi** and **M** or **Mx** act more nearly in concert. Here all three input values must have been latched before **Xi** can latch, but from then on there's no control to combine, only the data path to merge through the adding circuits. In effect the lower proposal retains a single layer of input latches and provides three stages of output latch.

There are many possible designs with more or fewer stages before or after the junction of the three channels and with more or fewer stages before or after the addition. What can one tell about the internal structure from the adder's terminal behavior? I assert that one can tell only how many stages there are altogether. How the stages are allocated internally is not visible from outside.

## ONLY ONE INPUT

The design presented here uses three inputs and therefore will require three destination addresses in the switch fabric. An alternate design might use a single destination address, separating the three inputs in time rather than in space. Such a design would be an accumulator that resets itself after each set of three inputs.

The switch fabric must include a mechanism that will resolve the conflict of several data values arriving concurrently at a single destination. A switch fabric with a single trunk resolves such conflicts by sequencing the data flow through the trunk. Richer switch fabrics may resolve such conflicts closer to the destination. Imagine, for example, a switch fabric with multiple routes to the destination that is the adder input. Data following any such route must reach the adder. Thus there must be some sort of "first come first served" mechanism in the switch fabric that will put currently arriving data into a time sequence.

Such a design might also use a simpler logic circuit to do the addition. Because only a single new input value arrives at a time, a two-input adder would suffice. It would add the existing sum and the new input value to form a new sum.

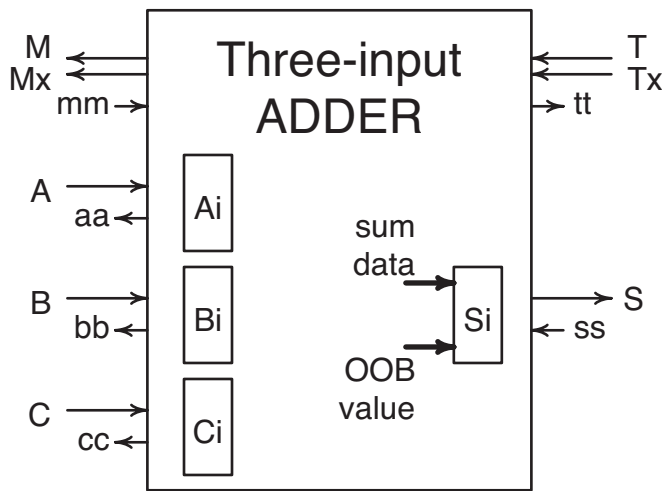


Figure 1: Block diagram

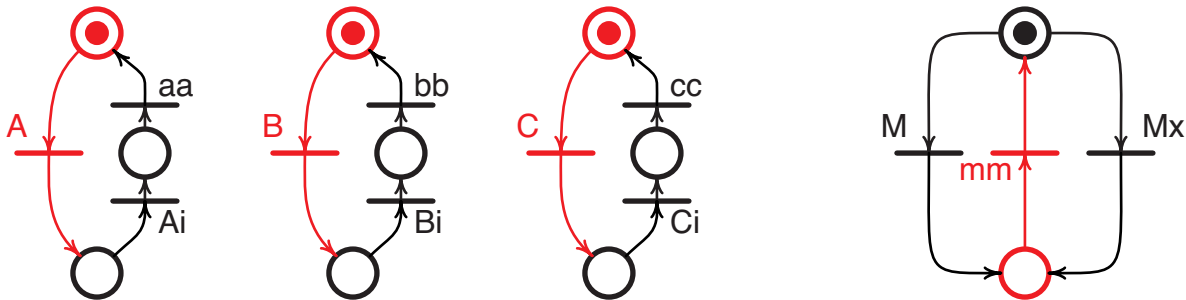


Figure 2a: Minor input interfaces

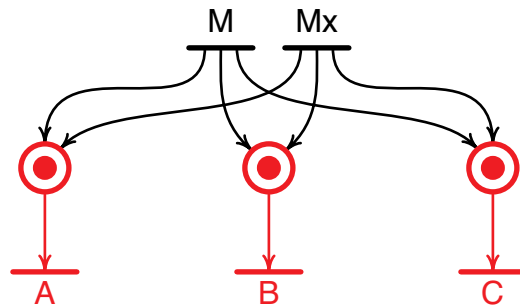


Figure 2b: Major input interface

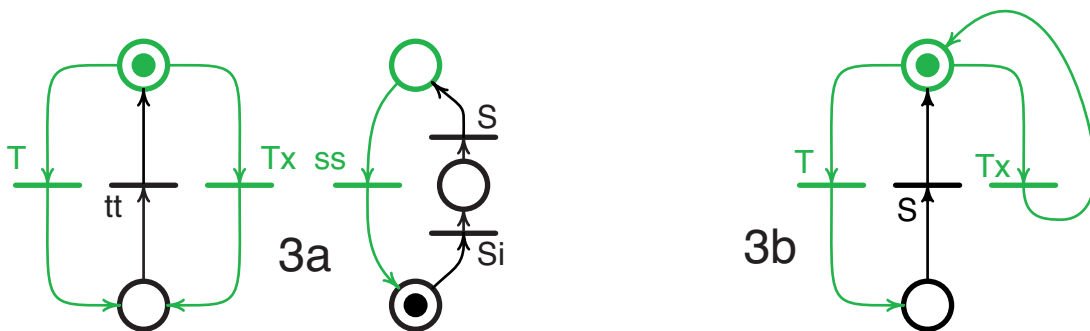


Figure 3: Minor (3a) and major (3b) output interfaces

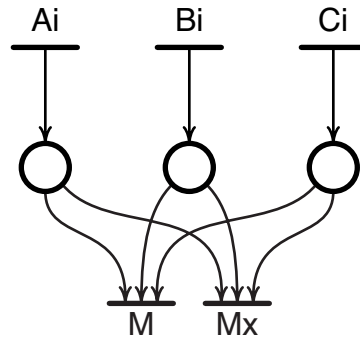


Figure 4: Token output requires all data latched

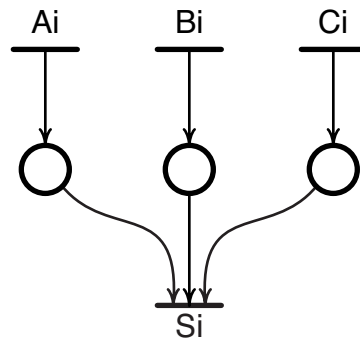


Figure 5: Sum latching requires three values

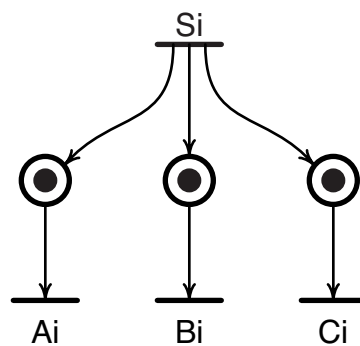


Figure 6: Next data only after sum is latched

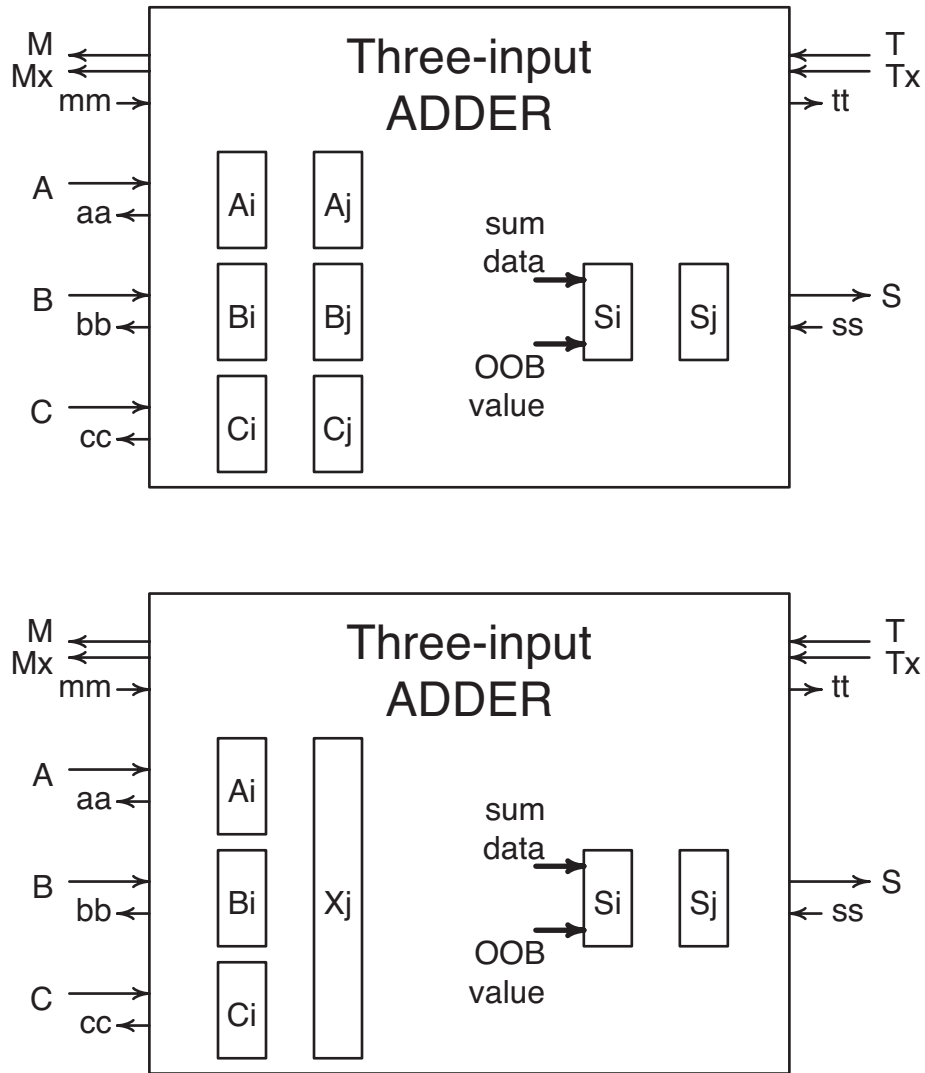


Figure 7: Block diagram with deeper pipeline