

UC Berkeley computer science

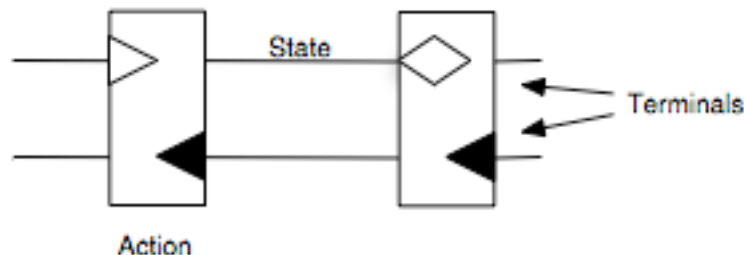
Subject: GASP Exercises
Date: Oct 5, 2005
Authors: Igor Benko
Number: 2005-UCIB05

References:

[1] Jo Ebergen: "Squaring the FIFO in GASP", ASYNC 2001
[2] Ivan Sutherland and Scottt Fairbanks: "GasP: A Minimal FIFO Control", ASYNC 2001

Notation and Semantics

In class, we introduced the following graphical notation:



Here is a brief recap of the notation: Each "box" is called a transition. The "wires" are called states, because this is where the data is stored. Transitions and states are connected via terminals. There are three kinds of terminals: We use no particular notation for output terminals. A triangle (arrows) denotes an input terminal, and a diamond denotes a non-resetting input terminal.

A state can be either active or inactive. When all states that are inputs to a transition are active, the transition can fire. Firing of a transition will cause all output states to become active. Furthermore, and input states connected via self-resetting input terminals will become inactive. Firing has no effect on states connected to non-resetting terminals.

Input terminals connector to states that are initially active are blacked out.

We can use such basic modules to build structures with more complex behaviors. Below are a few exercises that will help you get familiar with the concept and that correspond to some of the structures that we have encountered.

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.

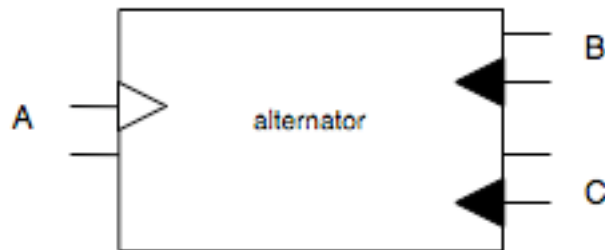
FIFO



A FIFO has a capacity to store N elements. For each activation of a request on the input side (A), we get one activation of the output side request (side B). Moreover, inputs and outputs can overlap, as long as the number of inputs minus the number of outputs is non negative and at most N (FIFO capacity).

Draw a structure for a FIFO with capacity 4.

ALTERNATOR

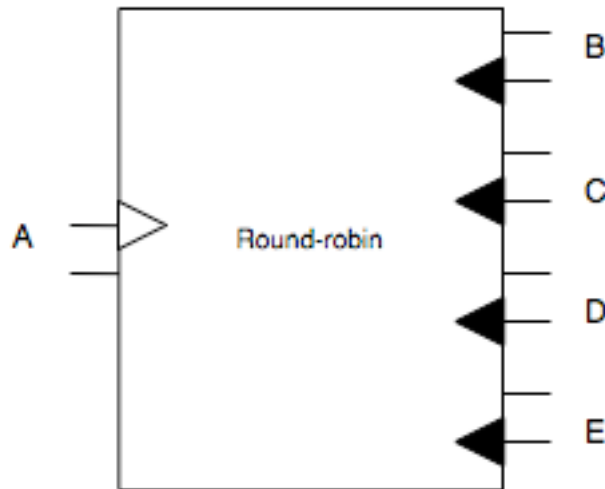


In this document, we will assume that input interfaces to our components consist of pairs that each include a self-resetting input request and an output acknowledgment, and that our output interfaces consist of pairs that each include an output and an self-resetting input. While this is not generally the case, such a convention will simplify our drawings.

We will denote the behavior by listing the sequence of interface pairs. For example, the alternator above has the following behavior: ABACABACABAC... That is, after the request state of interface A becomes active, the request state of interface B becomes active and so on. Assume that acknowledge states become active when their request counterparts go inactive.

Alternator is a common part of GasP structures. It's behavior does not correspond to the rules that must hold for a transition (when all inputs are active, the transition fires). An alternator can, however, be constructed from a network of transitions and states. Propose at least one such decomposition.

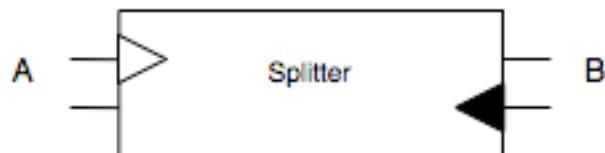
ROUND ROBIN



This component is a generalization of the alternator – it activates outputs in a round-robin fashion: ABACADAEABACADAEAB...

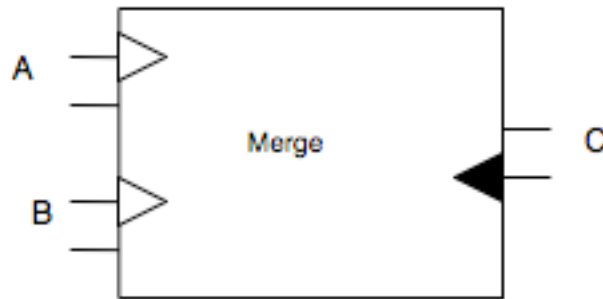
Propose at least one decomposition of the round-robin component into basic states, transitions and terminals.

SPLITTER



We use a Splitter to split a large data items into smaller one. For example, we could use it to split a 64 bid data chunk into four sixteen bit chunks. The behavior of a splitter is as follows: ABBBBABBBBABBBBA... So, for our splitter we assumed a four-way split. Propose how to build a four-way splitter from basic transitions, states, and terminals.

MERGE



The Merge performs the functionality opposite to the alternator; its behavior is ACBCACBCACBCACBC... That is, the Merge takes an input from the two input interfaces in an alternating manner and for each input that it consumes, the Merge produces one output.

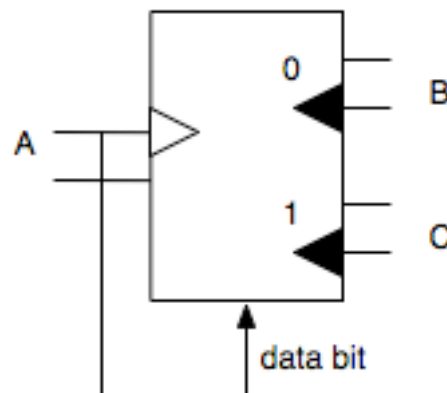
Propose how to build a Merge from basic building blocks. Discuss requirements for the behavior of Merge's environment. Propose extensions of the Merge to handle a wider range of environments.

FIFO with logarithmic latency

Using the building blocks above build a FIFO of capacity N that has latency proportional to $\log N$. The latency is the time required for an item to travel from FIFO's input to FIFO's output, assuming an empty FIFO.

DATA directed output

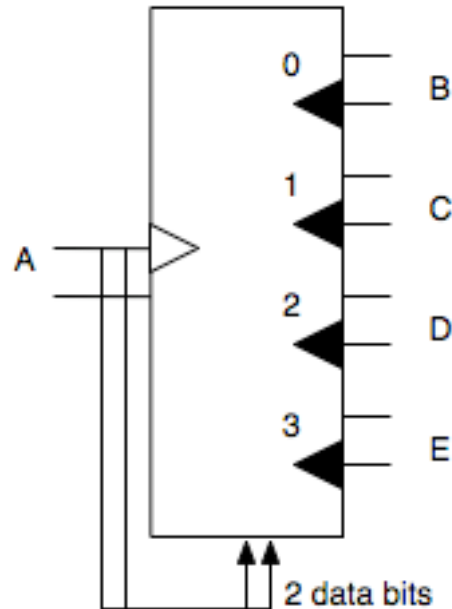
Let us add one more component to the mix.



Assume that the input to the transition carries one bit of data. If the value of that data bit is 0, then the transition fires and the output B becomes active. Output C is not affected.

Similarly, if the data bit is equal to 1, then output C becomes active and output B is not affected. You can imagine this component as a collection of two basic transitions – one with input A and output B, and one with input A and output C. Only one of the two transitions will fire, depending on the value of the data bit.

DISTRIBUTOR



What if we would like to steer data into four (N in general) destinations as opposed to two? Build such a distributor from the basic components presented above.