Counterflow Networks

Charlie Brej

Dept. of Computer Science,
The University of Manchester,
Oxford Road, Manchester, M13 9PL, UK.

cb@cs.man.ac.uk

Abstract

This paper introduces a concept of counterflow networks. These networks do not have nominal inputs or outputs but rather links along which data tokens can flow on both directions. Nodes, which are connected together with links, are able to execute computational operations on the incoming tokens. As all links are bidirectional, nodes treat all links as both inputs and outputs.This can be used to acknowledge parts of the circuit early or to supply information backwards.

1. Introduction

Asynchronous delay insensitive (DI) logic allows the construction of circuits without making any assumptions on the delay of any gate or wire. Timing information, instead of using a clock, is carried using handshaking signals. Request-Acknowledge signal pairs are used in most asynchronous logic circuits. Sequencing these signals allows asynchronous computation without a need for a global clock.

2. Links and Nodes

Counter-flow[1] networks are constructed with a combination of nodes and links. Links are bidirectional channels[2] along which tokens can move from one node to another. A link can accept a request from either side and transfer it to the other. Nodes are collection points where two or more links meet. A node can fire when all links it is attached to are "ready" and it has met its firing conditions. The firing conditions are based on the requests from the links. An 'OR node' will fire when any of the inputs fire. Effectively tokens are created on all inputs where tokens are not present.

Figure 1 shows an example structure where links are used with 'OR nodes' to create a pipeline. When a token enters the pipeline it will travel in one direction. In the example two tokens travelling in opposite directions have entered the pipeline. When they meet they will cancel each other out. This is because a node will create a token on each input which does not hold a token. The original token is removed and a new token is placed on the other side of the node. When a node has tokens on both sides then both tokens are removed.

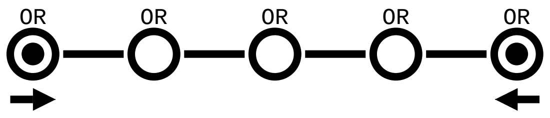

Figure 1: Counterflow pipeline example

A node can have a condition upon which it will fire. In the previous example the OR condition simply required one of the inputs to hold a token for the node to fire. Other conditions are possible involving some or all of the inputs.

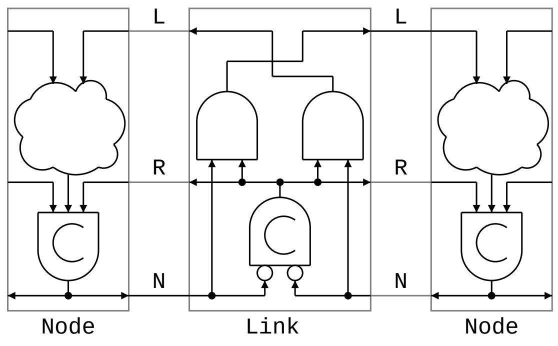

Figure 2: Node-link-node schematic

Figure 2 shows the schematics of two nodes and a link. The communication between them happens across three wires. The R (Ready) signal states if the link is ready to receive a token. The L (Latch request signal) states that the latch wishes to pass a token. N (Node request) informs links that a node has fired.

A link is made from an inverting C-element which fires when both nodes it links to have fired (N from both nodes is high). This drops the ready signal (R) and the latch request (L). Once both nodes have reset the ready signal (R) raises again. Once one of the nodes fires (one of the N signals rises) the latch request signal (L) rises again trying to cause the other node to fire.

Nodes are constructed using one C-element accepting the ready signals from all links and the firing condition signal. The node will only fire once all links are ready. The firing condition (cloud) takes as inputs the requests from all links it is connected to.

2.1. Unidirectional link optimization

Although links allow tokens to flow in both directions the nodes can be made to fire independent of the requests from some inputs. A FIFO is an example of this.

Figure 3 shows the possible optimisations when nodes do not require some link request inputs. The link only passes requests towards the right node and so the other L signal does not need to be generated.

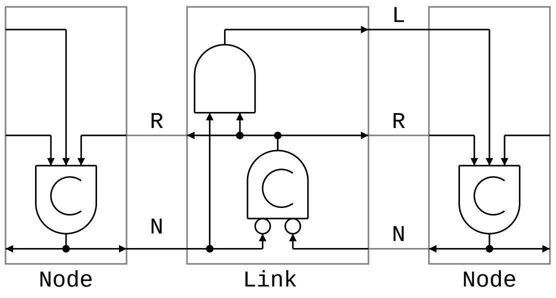

Figure 3: FIFO example

3. Data passing networks

Until now only control (0 bit) circuits have been demonstrated. Although complex networks can be created using this method, the circuits need to be able to pass data to allow actual computation.

Figure 4 shows a dual-rail counterflow network. The data passing wires and components are duplicated to allow the passing of dual-rail data. The link element detects the completion of both sides by passing the node request lines through a NOR gate before entering the C-element.

This construction allows data to flow in both directions. Computation can be done in the firing condition logic (cloud in fig. 4). Nodes can fire with a value. The links will accept values and pass them to the next node. Once both nodes have fired they are reset and primed for the next data inputs.

Dual-rail is just one of many encodings available. Other encodings are possible by simply changing the completion detection gate (in dual rail this is a NOR gate).

Figure 5 shows an example data passing network constructed from counterflow elements. Although in counterflow networks there are no strict inputs or outputs,

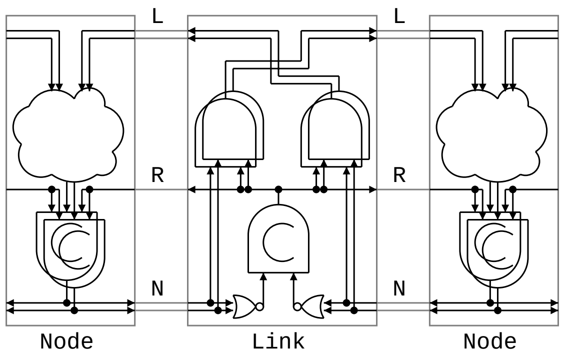

Figure 4: Data passing link and nodes

the example usually receives data on channel ‘I’. This data is duplicated in the OR node and passed to A and B. Circuits A and B process the data and generate interactions with other circuits. Both A and B generate data passed to C through the OR node. Both A and B generate the same result but use different methods to create it (e.g. ripple-carry v’s fast-carry adders). In a situation where A completes before B the result will be passed backwards to B. B can either use it as an acknowledgement and will not spend time generating a result to pass to C, or it can use the result to derive the values of the late inputs.

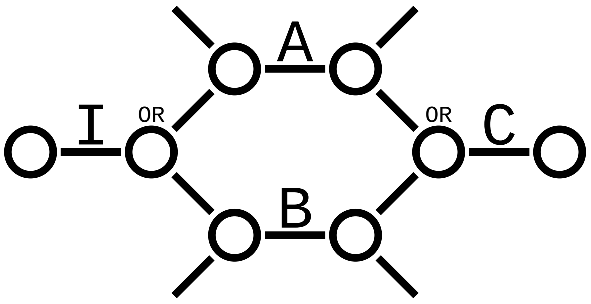

Figure 5: Example network

4. Conclusion

Counterflow networks look interesting and allow many strange computing structures to be created. Unfortunately it is not clear what function they can be used for.

The computation speed is comparable to that of DIMS circuits but the main advantage is creating circuits which have been in the past impossible without expensive arbitration.

5. Refrences

[1] R.F. Sproull, I.E. Sutherland and C.E. Molnar, "Counterflow Pipe-line Processor Architecture", Sun Microsystems Laboratories Technical Report, April 1994.

[2] C.F. Brej, “An automatic synchronous to asynchronous circuit convertor”, 11th UK Asynchronous Forum, 2001.