Early output logic and Anti-Tokens

Charlie Brej (cb@cs.man.ac.uk)

Introduction

The synchronous design model has proved extremely successful for designing logic for many decades. One of the main advantages of global synchronisation is that the designer is freed from worries about inter-unit communication; it can be assumed that any units in a device can communicate on any clock edge. Unfortunately synchronisation across a system, even a single chip, is increasingly difficult. One reason for this is the very high clock speeds employed in high performance designs. The effect of this starts to become problematic as the reachability of silicon area within one clock cycle becomes smaller.

Asynchronous logic

To address this some designers have looked at asynchronous communications, both between functional units and within blocks themselves. Such techniques allow elasticity in the system, alleviating or eliminating timing closure problems. In asynchronous designs, instead of all data progressing from one stage to another on the strike of the clock, the data progresses independently, managing their own timing. The data packets flowing through the system can be thought of as tokens.

Units have a number of input and output channels where they communicate tokens with other units. Each unit waits until it has accepted all tokens on all its input channels, processes the data, and generates tokens on its outputs.

Early output logic

In many situations some of the inputs are redundant to the generation of the correct output. These ‘early output’ cases are common in both low level circuits, such as OR gates where one of the inputs is one and high level circuits, such as multiplexers where only one of the data inputs is necessary to generate a result. In a multiplexer the necessity of a particular input is reliant on other input data (in this case the select input). The forced wait for all inputs to arrive before a result is generated is unnecessary and slows down the operation of the system.

Generating the result once sufficient data to complete the operation has arrived removes some of the unnecessary synchronisations and allows a more ‘free’ operation. However, although the result can be generated early, the unit has to wait for the late input to arrive in order to acknowledge it.

Anti-tokens

The late, unnecessary inputs stall the unit which can reduce the system performance. The unit instead of waiting for the late input can send a signal back up the pipeline towards the late token in order to cancel it and then continue. This signal travels in the opposite direction to normal tokens and when a collision happens between a it and token both are eliminated. Because of its behaviour it is called an ‘anti-token’.

Anti-tokens can progress backwards through the pipeline and when entering a unit which generates the late token, will remove all inputs present and generate anti-tokens on further input channels. This removes all tokens which would have combined to create the late token which was to be destroyed.

The action of removing the result closer to its origin both increases the system performance by releasing stages to operate on the next input set and reduces power consumption by stopping speculative operation from being conducted.

Composition

Early output circuits can be constructed using a method called direct translation. Taking a synchronous design as an input, the method decomposes the input circuit down to a set of non inverting AND/OR gates, inverters and clocked flip-flops. The direct translation then replaces each element with its asynchronous counterpart.

Dual-Rail

The dual-rail protocol is used to communicate and process data. Using two wires an upwards transition on either wire indicates different data (either one or zero). An acknowledge wire indicates the targets acceptance of the token. After each transaction the wires ‘return to zero’.

Gates

Asynchronous equivalent gates are constructed using two logic gates, the output of each driving one of the data wires. Each dual-rail gate also distributes the acknowledge of its output back to both of its inputs. In situations where the data signal forks the acknowledge signal must be gathered from all data destinations before being propagated back to its driver.

Inversions

In dual rail designs data inversions come for free. The wires are simply crossed so a zero signal will signify a one and vice-versa. Both buffers and inversions have no effect on the acknowledge signal.

Latches

Flip-Flop equivalents can be constructed using asynchronous latches. A variety of asynchronous latches are available with varying performance in different scenarios. The standard design called the “half latch” is constructed using two C-elements to capture the data and an OR gate to acknowledge the input once the data is latched. C-elements retain the state of their output until all inputs are in the same state. Then the C-element will switch to reflect the state of the inputs. Asymmetric C-elements have inputs which only effect either the rising or the falling transitions.

Validity

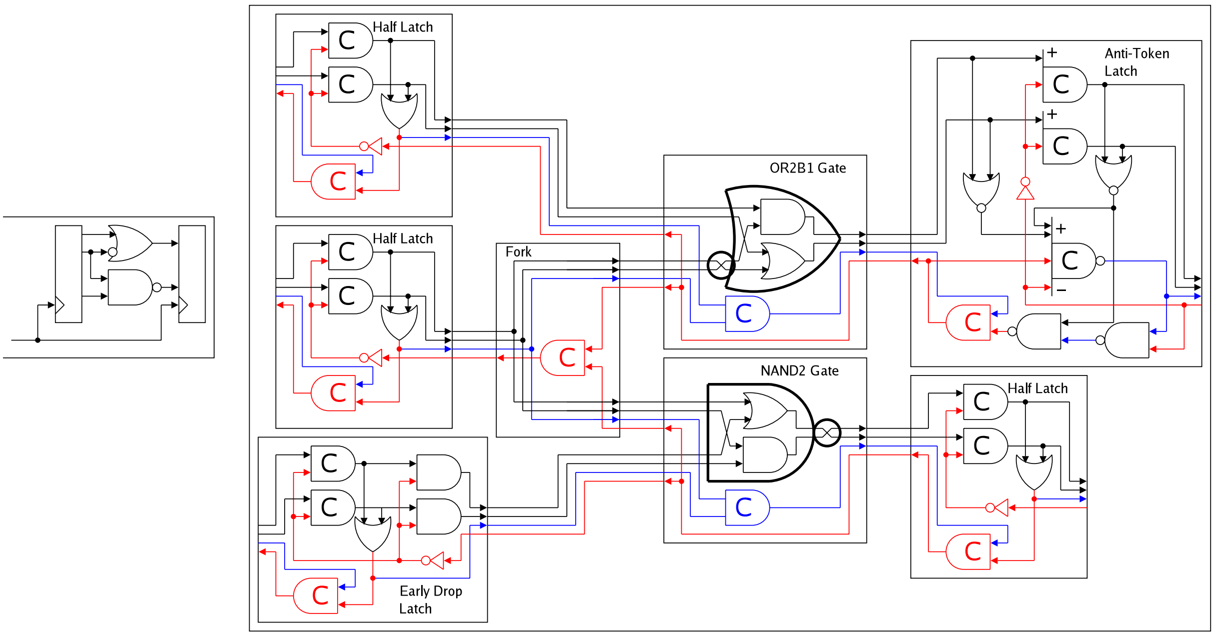
Early output circuits also use a separate signal called ‘Valid’ to ensure all inputs are ready to be acknowledged before the signal is propagated. The validities are gathered in much the same way as the acknowledge signals and combined with the acknowledge before propagating back to the inputs.

Anti-Tokens

Implementation of anti-tokens involves using anti-token latches to replace flip-flops. The anti-token latch will assert its validity before presenting data to the stage. This enables the stage to acknowledge it once the result has been generated.

Example circuit

The circuits shown below demonstrate the direct translation technique. The input circuit no the left is decomposed and then each element is replaced with an asynchronous equivalent. The resultant circuit uses bundles of four wires to replace each wire in the original design. Request zero, request one, valid and acknowledge wire bundles are used to pass data between units.



Example direct translation input circuit and generated asynchronous version