Self-Timed FPGA Systems

Rob Payne
rep@dcs.ed.ac.uk

Dept. of Computer Science, University of Edinburgh.

Abstract

Recently, there has been a renewal of interest in self-timed systems, due to their modularity, robustness, low-power consumption and average-case performance. Additionally, this paper argues that there are specific benefits to adopting self-timed design for FPGAs. The mapping problems of placement, routing and partitioning are simplified by not having a global clock constraint to meet, so more mappings are available for mapping algorithms to choose from. Hence, there is greater potential for algorithms to improve utilisation and performance of a design, or instead, to increase design turn-around by taking less time to produce a mapping. Furthermore, the ability to perform mappings quickly enables new FPGA applications where the mapping to the FPGA is done on-the-fly. Currently available FPGAs provide no support for self-timed design. The latter half of the paper describes the STACC architecture, an FPGA architecture targeted at the implementation of self-timed bundled-data systems.

1 Introduction

All current commercial FPGAs are designed for implementing systems synchronously, so have dedicated clock signals. However, the alternative approach of building systems in an asynchronous or self-timed fashion has many potential benefits. This paper examines the case for self-timed FPGA systems and introduces STACC, a dedicated self-timed FPGA architecture.

Section 2 introduces self-timed systems and examines the general advantages of building systems asynchronously. In section 3, the specific benefits of building self-timed FPGA-based systems are considered. Section 4 reviews the current work on self-timed FPGAs and argues that current FPGA architectures do not support self-timed designs. In section 5, the STACC architecture is introduced, concentrating on the development of a timing cell for the architecture, from an unreconfigurable structures to a fully reconfigurable timing cell. Finally, section 6 summarises the paper and looks at future directions for the work.

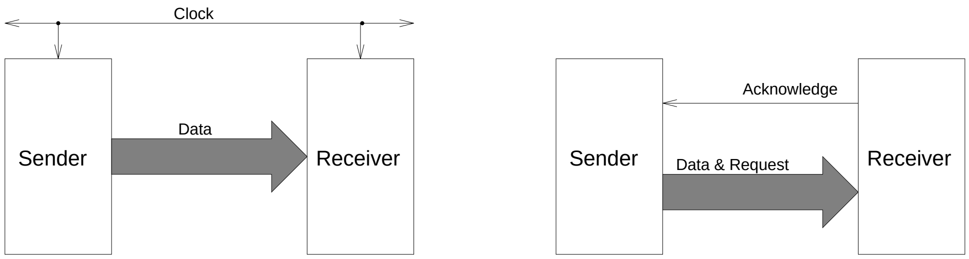

Figure 1: Synchronous and Self-Timed Protocols
(Left) Synchronous Communication Protocol. (Right) Self-Timed Protocol.

2 Self-Timed Systems

In *synchronous systems* (figure 1), all modules are synchronised through a global clock signal. The clock places a global constraint on the system: all the modules within the system must have their data ready by the next tick of the clock. In *asynchronous systems* there is no global synchronisation of the modules within the system, but modules do synchronise locally through their communication protocols. The set of asynchronous communication protocols that use some form of handshake between sender and receiver are known as *self-timed* (for a formal definition see [1]).

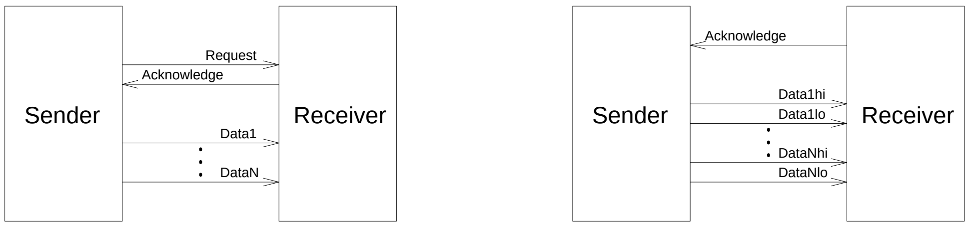

Figure 2: Self-Timed Protocols.
(Left) Bundled Data Protocol. (Right) Dual Rail Encoded Protocol.

Figure 2 shows two common self-timed communication protocols. Both protocols come in *two-phase* and *four-phase* variants that attach different significance to transitions within the request/acknowledge handshake. In two-phase or *event-based* signalling, all transitions are significant. In four-phase signalling, only positive-going transitions are significant, so an idle *return-to-zero* transition is required.

In the *bundled-data* protocol (figure 2, left), a transition on the request signal indicates that the data is valid. The requirement that the request transition occurs after the data is stable is known as the *bundling constraint*. Sutherland's Micropipelines [2] popularised a 2-phase form of the bundled data protocol. Subsequently, Micropipelines were adopted by the AMULET group [3], at the University of Manchester, who built a self-timed version of the ARM processor.

In the *dual-rail* protocol (figure 2, right), a pair of signals are used to send each data bit. One signal indicates the data bit is high, the other that the data bit is low. There is no explicit request signal, instead, a request is implied when a transition has occured on each pair of data signals. The dual-rail-code has been used by Martin [4] to build an experimental self-timed processor.

An important property of self-timed protocols are that they are *speed independent*: the protocols work regardless of the speed that the modules produce results. In addition, protocols such as the dual-rail code are *delay-insensitive*: arbitrary delays may occur on the wiring and the protocol will still work. The bundled-data protocol is not delay-insensitive due to the bundling constraint.

The removal of the global clock constraint in self-timed systems leads to several advantages over their synchronous counterparts. These are summarised below:

Modularity: In synchronous systems, all parts of the system are implicitly dependent on each other through the global clock signal. In self-timed systems, no global clock constraint has to be met, so any module with the required functionality can be used regardless of performance and the system will still work.

Robustness: Because self-timed systems are speed-independent, they are resilient to delays caused within modules by environmental conditions. Delay-insensitive systems are also resilient to arbitrary delays in wiring as well.

Average-Case Performance: The clock period in a synchronous system must be slower than the worst worst-case delay of all the modules within the system. In contrast, since self-timed systems are not limited by a global clock, they can go at their own speed, so tend to exhibit average-case performance rather than worse-case.

Low Power Consumption: For CMOS, the static power consumption is almost zero. However, in a synchronous system, transitions on the global clock are always causing transitions to be passing through the system causing power dissipation, even when the system is idle. Within a self-timed system, transitions and hence power dissipation only occur when data is passing through the system.

No Clock Distribution Problems: Problems such as clock skew are avoided since there is no global clock in a self-timed system.

The main drawback to self-timed systems is that extra circuitry is required to generate the local timing information. The overhead depends crucially on the protocol chosen. Delay-insensitive protocols, such as dual-rail code, generally need two wires per data bit, which represents a significant overhead. Bundled-data protocols only have the overhead of the acknowledge and request signals so the overhead is smaller and can be negligible for large data bundles. This is at the expense of requiring more careful design to ensure the bundling constraint is met. In both cases, the 4-phase variants of the protocols need less circuitry than the 2-phase equivalents.

3 Self-timed FPGA-based Systems

In the previous section, the general advantages of self-timed systems were considered. However, there are also specific advantages to applying a self-timed approach to FPGA-based systems. By removing the global clock constraint, the mapping problems of partitioning, routing and placement are simplified. Since only the local constraints of the self-timed protocol have to be met, many more working mappings of a circuit are possible. Hence, there is greater scope for mapping algorithms to search for mappings with improved utilisation or performance, or simply to find a working mapping in less time. Below, the main implications of more flexible mapping to the FPGA are outlined:

Ease of Partitioning: The difference between off-chip and on-chip delays causes problems in any design but these problems are particularly acute in FPGAs where partitioning is more frequent as less functionality can fit on one chip. Using self-timed protocols, off-chip delays only limit performance when the signals between partitions are being used. A design can treat a array of FPGAs as a uniform array of cells since the self-timed protocols accommodate for the off-chip delays. Hence partitioning algorithms are only required to improve performance rather than to achieve a mapping that meets the global clock constraint.

Cope with Saturated Routing: In many designs with high utilisation, the interconnect can become saturated, causing many signals to be placed on long snaking paths through the FPGA. This can severely limit clock rate in synchronous designs. In a self-timed design, these paths can be allocated to infrequently used signals, which will only limit performance when the signal is being used.

Faster Mapping and Quicker Design Turn-around: In synchronous systems, detailed timing analysis of all signal paths is needed to ensure that the result of routing, placement and partitioning of a design meets the global clock constraint. Because of the insensitivity to delays of self-timed designs, any route, place and partition that implements the system net-list will produce a working system, as long as it meets the local constraints of the self-timed protocol. Detailed timing analysis is only required to improve the performance of the mapping rather than to ensure a working system. Initial mappings may be done quickly, enabling faster design turn-around. Detailed timing analysis for improved performance can be reserved for the mapping of the final design.

Mapping on the Fly enables New Applications: Fast mapping enables systems where the mapping to the FPGA is not fixed before run-time as in current FPGA systems. One way to exploit this advantage is to generate custom circuits for the problem in hand, rather than having a very general-purpose circuit. For example, custom pipelines for a specific processing task could be built from a selection of pre-designed generic modules and compiled together quickly to produce a working circuit of the required data width and function.

The ability to map circuits quickly to the FPGA at run-time seems essential for proposed *virtual hardware systems* [5]. Virtual hardware systems are analogous to virtual memory: they try to emulate a larger circuits by swapping sub-circuits to and from configurable hardware. Unless the pattern of swapping is known before run-time, as in current virtual hardware applications (such as the neural-net simulations in [6]), the mapping has to be done on-the-fly, which requires mapping algorithms to the FPGA that are fast and robust.

4 Current Research on Self-Timed FPGAs

Most of the current work on self-timed FPGA systems has concentrated on building such systems using commercially available FPGAs. Brunvand [7] composed a library of self-timed data-bundled elements for the Actel FPGA, and subsequently built a processor with these elements [8]. Oldfield and Kappler [9] compared self-timed FIFOs implemented on CAL chips and on customised silicon. Shaw and Milne [10] implemented asynchronous circuits on the CAL-based SPACE machine. Other researchers have also built Micropipeline libraries [11, 12].

The advantage of using current FPGAs for implementing self-timed systems is that the chips are readily available standard parts, and can implement synchronous systems as well. However, whilst showing the potential of self-timed FPGA systems, these works have highlighted some limitations of current synchronously-oriented FPGAs for implementing self-timed systems. These limitations are listed below:

Arbitration: Arbitration and synchronisation are common functions in self-timed circuits. The arbiter elements used by Brunvand [7] in his self-timed library for Actel Chips have a small chance of failure. Arbiters are the only function that cannot be built using current FPGAs.

Local register clock distribution: In bundled-data protocols, although there is no global clock signal, there are still local clock signals to control registers. For good performance, these local clock signals need to be distributed quickly to registers, but current FPGA architectures have little support for local clocking signals. One solution is to use delay-insensitive protocols: these do not have local clock signals.

Delaying signals: Current FPGAs have no support for delaying signals. Delay elements are essential in bundled-data protocols, to delay the request signals to meet the bundling-constraint. Current architectures have little support for delays; Brunvand [7] has to waste cells by building inverter chains to build delay elements. A related problem is that routing architectures can re-order signals so that the bundling-constraint is not met. Even though delay-insensitive self-timed circuits do not have bundling-constraints to meet, Martin [13] shows that delay-insensitive circuits cannot be built without isochronic forks. Isochronic forks require one-sided delay bounds that can be hard to meet in current FPGA routing architectures.

The only FPGA architecture that has addressed some of these problems has been the MONTAGE architecture [14], which was designed for the implementation of both synchronous and asynchronous systems. MONTAGE solves the arbitration problem by sprinkling a small number of special arbitration cells amongst standard FPGA cells. Also, the routing architecture is designed to facilitate the construction of isochronic forks. However, no specific support is given for local clock distribution or for delaying signals so that they conform to bundling-constraints.

5 STACC: a Self-Timed FPGA architecture

Whilst the MONTAGE architecture includes many features to improve its implementation of self-timed circuits, it still retains a global clock. In contrast, the STACC (Self-Timed Array of Configurable Cells) is a dedicated self-timed FPGA architecture specifically for the implementation of bundled-data systems. The bundled-data protocol was chosen as it has a small area overhead compared to delay-insensitive protocols. Additionally, bundled-data systems use the same data-path as their synchronous counterparts, making it easy for designers to adapt to self-timed design. The four-phase variant of the protocol was chosen since it requires less circuitry to implement basic branching structures.

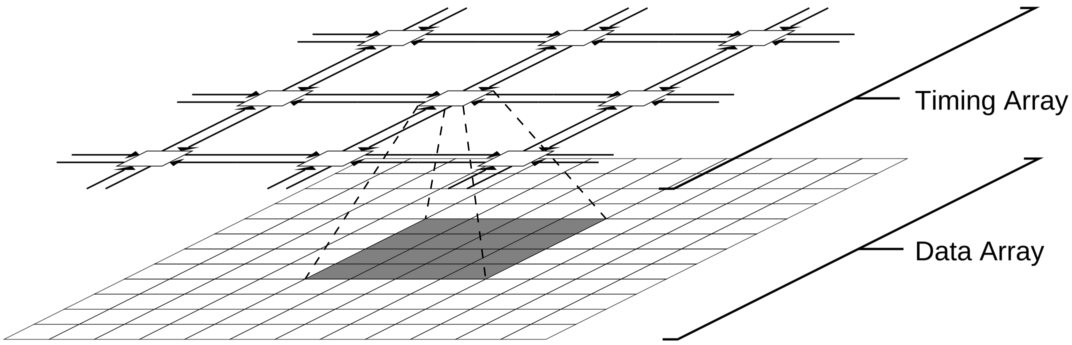

Figure 3: Basic Structure of the STACC architecture

In Figure 3, the basic concept behind the STACC architecture is shown. The *data array* consists of cells similar to current synchronous FPGAs, but the global clock has been replaced by an array of timing cells. Each timing cell provides local timing information to a region of FPGA. A timing cell and the data cells that it provides timing information for, are known as a *timed region* (illustrated by the shaded region in figure 3).

Each timing cell is connected to its neighbours by two wires that are used to initiate request/acknowledge handshakes with neighbouring timing cells. Configuration data determines whether neighbouring timed regions communicate, and the direction of data flow between them. For a configuration to produce a working circuit, the configuration of the data array must reflect the configuration of the timing array. No data should flow between timed regions that are not linked in the timing array.

Since the data array can use any standard style of FPGA cells, the rest of this section concentrates on the novel part of the STACC architecture, which is the timing cell array. The discussion starts by looking at unreconfigurable four-phase bundled-data pipelines, and then develops a timing-cell capable of implementing such pipelines with fan-in and fan-out. Successively more general versions of the timing cell are introduced that allow the implementation of branching and merging within the structures. The final timing cell is flexible enough to implement all the structures mention by Sutherland [2].

5.1 Four-Phase Pipelines

Figure 4 shows a four-phase bundled-data pipeline. As with Sutherland's Micropipelines [2], the basic timing element is the C-muller gate. The C-muller gate's behaviour is to cause a rendezvous between its inputs. In other words, the C-muller gate's output will not change until all of its inputs have changed. This behaviour is summarised in table 1. Bubbled inputs in figure 4 indicate an inverted input. In effect, on initialisation, transitions are already assumed to have occured on bubbled inputs. At initialisation, the output of all the C-muller gates is logic zero (the reset signal has been omitted from the diagram).

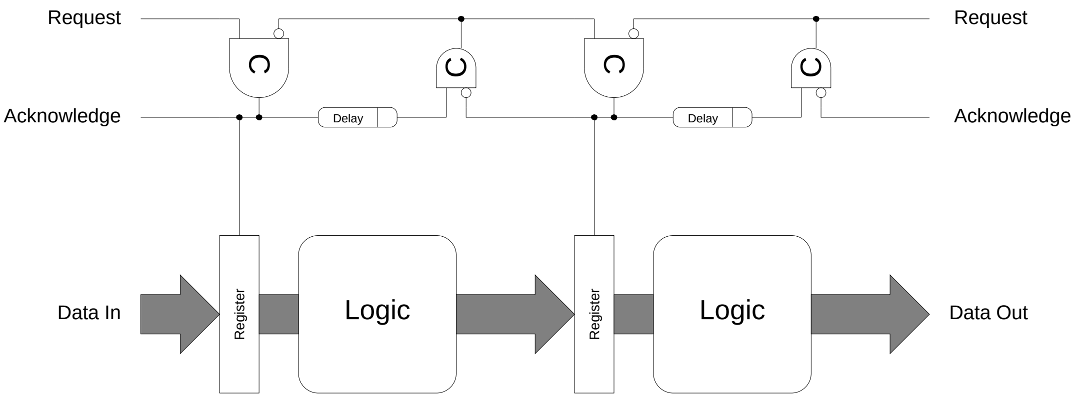

Figure 4: A four-phase pipeline with two stages

Inputs	**Output**
all logic 0	logic 0
dissimilar	previous value
all logic 1	logic 1

Table 1: C-Muller Gate Behaviour

In figure 4, the larger C-muller gates generate the local register clocking signal. The larger timing C-muller gates cause a rendezvous between the request signal from the previous stage and the acknowledge from the next stage in the pipeline. The rendezvous ensures correct operation of the pipeline, since the next processing cycle will not start until the previous stage has data ready (indicated by the request) and the next stage has got this stage's last result (indicated by the acknowledge). After the rendezvous, the C-muller's output latches the inputs into the register and sends an acknowledge back to the previous stage in the pipeline. The delay element delays the request to the next stage so that the logic block's outputs are valid.

Unlike Sutherland's two-phase Micropipelines, the four-phase protocol has an idle return-to-zero phase. Hence the delay elements and registers are only active on positive-going transitions. To allow the idle return-to-zero phase to occur concurrently with the processing phase, the smaller C-muller gates have been added between the stages of the pipeline in figure 4. Their effect is to act as a holder for the return-to-zero transitions whilst the stages are processing.

The other main difference of the four-phase pipeline of Figure 4 from Micropipelines is the choice of register element. Instead of the capture-pass register used in Micropipelines, edge-triggered D-types are used. The advantages of using D-types are that only one control signal is required, and that the pipeline stages can retain state. Hence, by feeding their outputs back, the logic blocks can implement a finite state machine. The pipeline of figure 4 can easily be generalised to deal with fan-in and fan-out in the pipe by having multiple request or acknowledge signals going into the timing C-muller gates.

5.2 A Basic Timing Cell

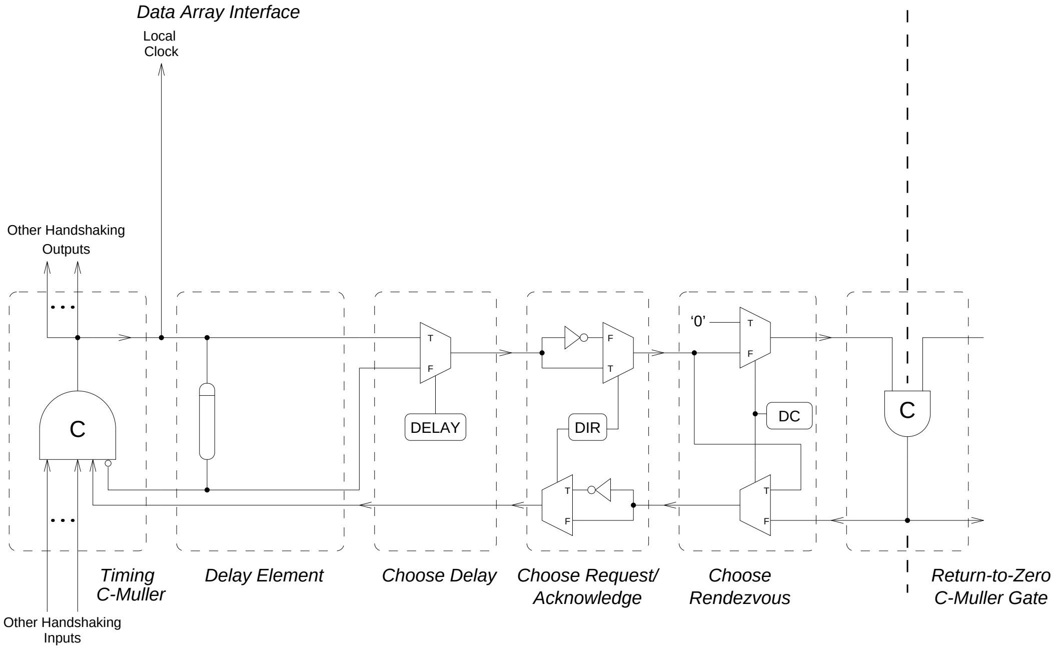

Figure 5: A Basic Four-Phase Timing Cell

This section introduces a basic timing cell that allows the pipelines described above to be implemented. Figure 5 shows half of a reconfigurable connection between two such timing cells. The circuit is mirrored around the thick dotted line for the other side of the connection.

As in figure 4, the timing and return-to-zero C-muller gates are retained. Various configuration bits controlling multiplexors are used to determine the nature of the connection. These configuration bits are described below.

DC: The DC (Don't Connect) configuration bit determines whether there is a connection between a timing cell and its neighbour. To connect with the neighbouring cells, the handshaking signals are passed to and from the return-to-zero C-muller gate. If there is no connection to the neighbouring cell, the out going handshaking signal is fed back to the timing cell, so that the C-muller gate effectively acknowledges itself.

DIR: The DIR bit determines which handshaking signal is the request and which is the acknowledge. Since the difference between a request and acknowledge is whether the signal is inverted (bubbled) or not, the DIR bit simply chooses which signal to invert.

DELAY: The DELAY configuration bit determines whether the out going handshaking signal is delayed. Normally, requests are delayed as they generally involve a data transfer that must meet the bundling-constraint, whilst acknowledges are not delayed since no data transfer is involved. However, by delaying acknowledges, data can also be passed on the acknowledge signal, allowing a two-way exchange of data. Alternatively, requests need not be delayed if no data transfer is involved with them.

An important part of the timing cell is the delay element. The delay must always be long enough so that the bundling constraint is met. However, for performance, it is important that the delay matches the logic block delay as closely as possible. One easy way to implement the delay element is to allow configuration data to choose from a range of fixed delays. An interesting unproven alternative is to use CSCD (Current Sensing Completion Detection) [15]. CSCD utilises the fact that the static power and hence static current flow of a CMOS circuit is close to zero. By monitoring the current flow between the power rails, a CSCD circuit produces a variable delay that matches the time taken by the current computation.

5.3 A General Timing Cell

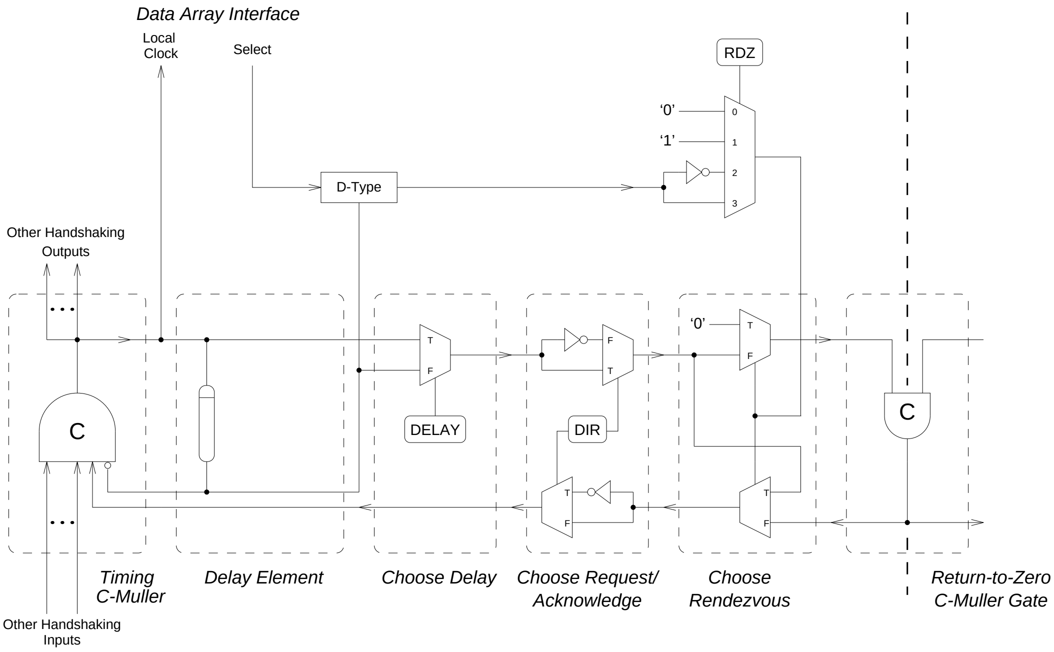

Figure 6: Addition of Select Signals

So far, a cell capable of implementing fan-in and fan-out pipelines has been described. However, when such pipelines branch, all of the fan-out pipes are followed. There is no way to choose that the computation should only continue along one branch. Conversely, there is no way for the cell to pick which of several fan-in pipes it wishes to accept data from. Sutherland's Micropipelines [2] use a variety of elements such as the Select, Toggle and Merge elements to implement branching and merging. In this section, the timing cell is developed to allow such branching and merging in four-phase pipelines.

Figure 6 shows an adapted timing cell that allows a connection to be chosen by an input from the data array. The select signal is latched after the delay, and determines whether communication takes place with the neighbouring cell in the current cycle. The RDZ configuration bits allows the choice of selected or inverted select signals so that the initial value of the select signal can be defined (assuming that the D-type is reset to a pre-defined value). The RDZ configuration bits can also choose the constants '0' or '1' so that the common functions of "never communicate" and "always communicate" can be implemented without using resources in the data array.

Another function of the select signal, not shown on the diagram, is to act as an enable signal for memory elements in the data array. Since neighbouring timed regions that are not selected for communication are not synchronised to the timing cell, sampling their data inputs can cause a meta-stable state in the data-array's memory elements. Hence, the select signals are also used to disenable memory elements not involved in the current communication.

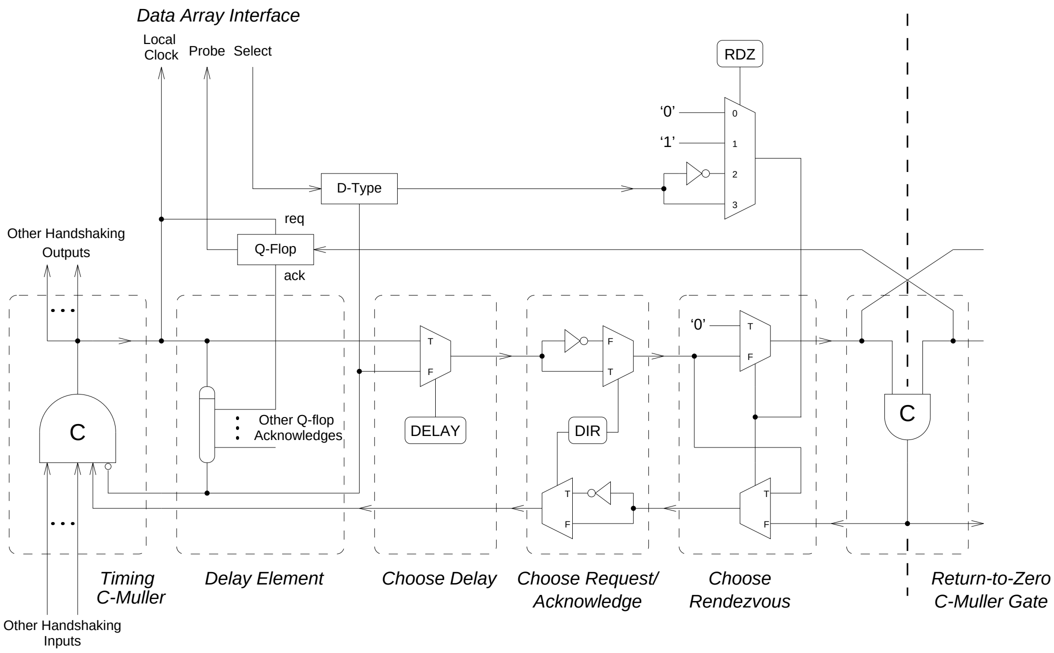

Figure 7: Addition of Probe Signals

There are still certain behaviours that the timing cell of figure 6 cannot implement. Although, it can make decisions based on its own internal state, it cannot choose which connections to select on the basis of which neighbouring timed-regions are waiting to communicate. Figure 7 shows the development of figure 6 that allows the state of neighbouring cells to be probed. The probe is fed in to the data array which can then make a decision on what connections to select. Since the probe signals are asynchronous to the cell, some form of synchroniser element must be used to sample the inputs. In this case, a Q-flop [16] is used. The Q-flop samples its input after a transition on its request signal. After any meta-stable state has been resolved, the Q-flop generates an acknowledge. This is fed to the delay element which delays the logic accordingly. Since Q-flops are relatively complicated elements to implement, alternative variants of the architecture are possible that just use a single arbiter element per timing cell.

6 Conclusions and Future Work

The first half of this paper set out to show the potential benefits of building a dedicated self-timed FPGA architecture. Many FPGA-specific benefits arise from the general robustness, modularity and average-case performance properties of self-timed systems. The mapping problems of routing, placement and partitioning are given a wider range of solutions to choose from, allowing a greater variety of trade-offs between performance, utilisation and time to find a solution.

The second half of the paper concentrated on the design of the STACC architecture, a dedicated self-timed FPGA using a four-phase bundled-data protocol. The discussion focussed on the design of the timing cells since these constitute the main novel part of the architecture. The final version of the timing cell could implement more than just a timing function since it was flexible enough to implement a wide range of control functions using a relatively small number of configuration bits. As presented, the timing cell only required four bits per connection. Furthermore, some configuration bits such as the DIR bit could be shared with neighbouring cells.

Many other aspects of the architecture were left undiscussed, and are part of ongoing simulation studies. Such aspects include the style of configuration interface, non-local and off-chip communication links and the nature of routing, placement and partitioning algorithms. The results of the simulation work are intended to select an architecture variant for chip layout, so that comparison can be made with synchronous FPGA architectures.

Acknowledgements

My thanks to Gordon Brebner and Iain Lindsay for their advice and guidance. Also to Vinod Rebello and Rob Mullins for helpful discussions concerning self-timed systems and CAD tools.

References

[1] C.L.Seitz. *System Timing*, chapter 7. Addison-Wesley, Mead and Conway Introduction to VLSI Systems edition, 1980.

[2] I.E.Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–38, 1989.

[3] S.B.Furber, P.Day, J.D.Garside, N.C.Paver, and J.V.Woods. A Micropipelined ARM. In T.Yanagawa and P.A.Ivey, editors, *Proceedings of VLSI 93*, pages 5.4.1–5.4.10, September 1993.

[4] A.J.Martin, S.M.Burns, T.K.Lee, D.Borkovic, and P.J.Hazewindus. The Design of an Asynchronous Microprocessor. In C.LSeitz, editor, *Advanced Research in VLSI: Proceedings of the Decennial Caltech Conference on VLSI*, pages 351–373. MIT Press, 1989.

[5] X.Ling and H.Amano. WASMII: a data driven computer on a virtual hardware. In *FCCM93: Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, 1993.

[6] P.Lysaght, J.Stockwood, J.Law, and D.Girma. Artificial Neural Network Implementation on a Fine-Grained FPGA. In *4th International Workshop on Field Programmable Logic and Applications*, 1994.

[7] E.Brunvand. Using FPGAs to Implement Self-Timed Systems. *Journal of VLSI Signal Processing*, 6(2):173–190, August 1993.

[8] E.Brunvand. Using FPGAs to Prototype a Self-Timed Computer. In *Workshop on Field Programmable Logic and Applications*, pages 192–198, 1992.

[9] J.Oldfield and C.Kappler. Implementing Self-timed Systems: Comparision of Configurable Logic Arrays with Full Custom Circuits. In *FPGAs: International Workshop on Field Programmable Logic and Applications*, chapter 6.3. Abingdon EE&CS Books, 1991.

[10] P.Shaw and G.Milne. A Highly Parallel FPGA-Based Machine and its Formal Verification. Technical Report HDV-28-93, U. of Strathclyde, 1993.

[11] M.Gamble, B.Rahardjo, and R.D.Mcleod. Reconfigurable FPGA Micropipelines. Technical report, U. of Manitoba, 1994.

[12] K. Maheswaran and V. Akella. Hazard-free Implementation of the Self-Timed Cell set for the Xilinx 4000 Series FPGA. Technical report, U.C.Davis, 1994.

[13] A.J.Martin. The Limitations to Delay-Insensitivity in Asynchronous Circuits. In W.J.Dally, editor, *Sixth MIT Conference on Advanced Research in VLSI*, pages 263–278. MIT Press, 1990.

[14] S.Hauck, G.Borriello, S.Burns, and C.Ebeling. MONTAGE: An FPGA for Synchronous and Asynchronous Circuits. In *Workshop on Field Programmable Logic and Applications*, 1992.

[15] M.E.Dean, D.L.Dill, and M.Horowitz. Self-Timed Logic Using Current-Sensing Completion Detection (CSCD). In *Proc. International Conf. Computer Design (ICCD)*, pages 187–191. IEEE Computer Society Press, October 1991.

[16] F.U.Rosenberger, C.E.Molnar, T.J.Chaney, and T.Fang. Q-Modules: Internally Clocked Delay-Insensitive Modules. *IEEE Transactions on Computers*, C-37(9):1005–1018, September 1988.