Self-timed and speed independent latch circuits

A. Bystrov, D. Shang, F. Xia* and A. Yakovlev
Department of Computing Science, University of Newcastle

Indexing terms: Asynchronous systems, latches and flip-flops, speed independent circuits, STGs, Petri nets.

1 Abstract

Several designs of self-timed (some of which speed independent) latch circuits are presented. These are used in the speed independent (SI) implementation of two consecutive binary assignment statements. Issues such as logic reduction, utilisation of well known, simple components, and response speed improvement are dealt with in detail. The techniques employed can be used in designing other asynchronous circuits where regularity and modularity are important issues.

2 Problem

In studying asynchronous communication mechanisms (ACMs) [1], the problem of implementing two consecutive binary assignment statements in a fully SI fashion was encountered. This problem is stated below:

Two binary assignment statements,

$$x{:=}y;$$
$$y{:=}a.$$

where x, y, $a \in \{0, 1\}$, are to be implemented with SI hardware. The "*start*" signal (request) for the first statement is received from the environment of the circuit. When both statements are completed, a "*done*" signal (acknowledgement) is to be issued by the circuit to the environment. This handshake is shown in Figure 1.

It should be noted that the sequence of assignment statements is important. In other words, in the course of execution of these two statements, x is assigned the old value of y and y is assigned the value of a.

---

* Contact: fei.xia@ncl.ac.uk

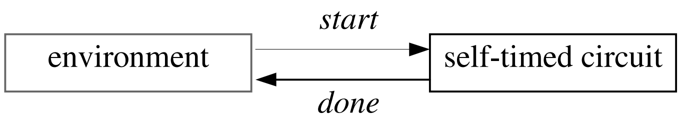

Figure 1 Basic handshake protocol of circuit.

An obvious, non-speed-independent solution exists in the form of "shifting registers" as shown in Figure 2.

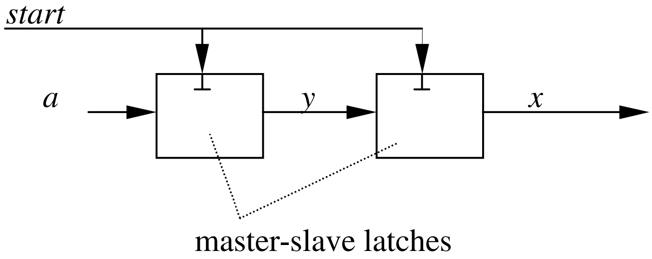

Figure 2 A simple implementation of two consecutive assignments.

If the latches in Figure 2 are of the master-slave (MS) type, the rising edge of the clock pulse can be used to trigger the masters and the falling edge the slaves. This implements the two statements in "parallel" logic.

In reality, of course, the parallelism does not speed up the response as it still requires two sequential latch actions to implement each of the two statements.

In addition, this is not an SI solution and requires the environment to determine the timing of both stages of the two statements. Specifically, the length of the clock pulse is not directly dependent on the actual completion of the master part of the action and the completion of the slave part of the action is not collected for future use.

3 Self-timed considerations

The requirement that the sequential nature of the two statements must be strictly observed means that at least two separate layers of latch action must be used in the implementation. These can be in the form of either master-slave pairs in a parallel solution or single layer latches connected in series in a sequential solution. The interest in simplicity implies that we only seek solutions with two layers of latch action.

Therefore, to further clarify what a self-timed solution must provide, we establish a set of more detailed handshake protocols for the circuit. This is shown in Figure 3. It implies that both layers of latches need to be connected to completion logic, or produce their own completion signals, to serve as the required "$done_1$" and "$done_2$" signals.

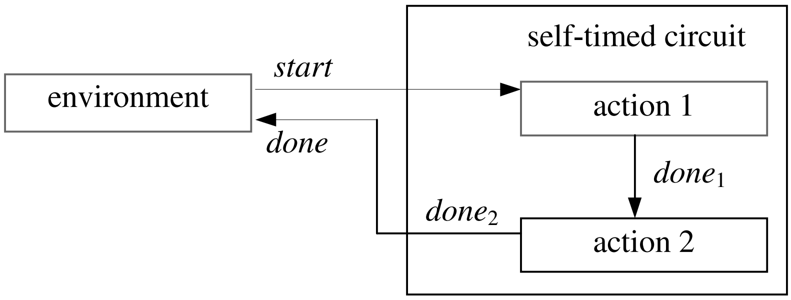

Figure 3 Detailed specification of self-timed solutions.

4 Direct solution

A completely SI master-slave latch can be found in [2] and is used in [3]. This circuit can be immediately adapted for use here by inserting it into the shifting register scheme in Figure 2. A modified form of this latch is used here and is shown in Figure 4.

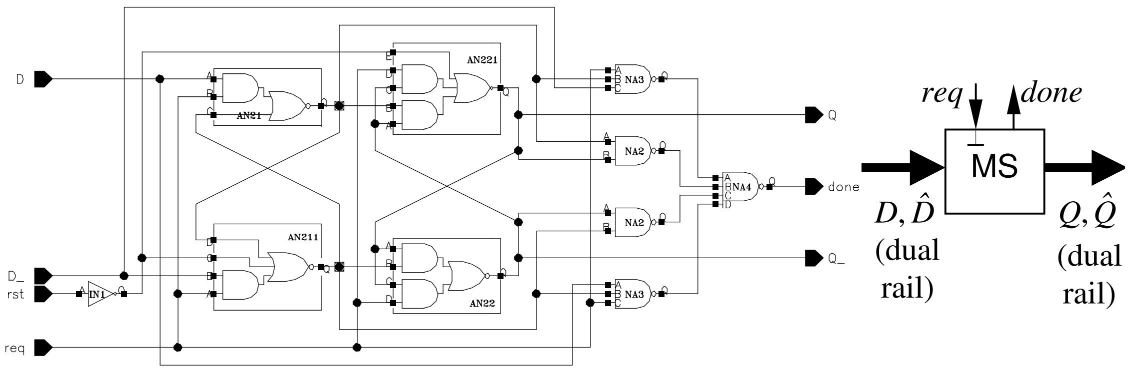

Figure 4 Self-timed master-slave latch circuit its symbol.

As can be seen in [3], this circuit is quite slow in the original form where the completion logic is implemented with a single complex gate, providing speed independence. Here we have used separate gates to increase performance, leaving the question of speed independence open. If the layout is carried out in-house, assumptions can be made so that this is not a problem to worry about. Simulations have shown that the not strictly SI circuit adopted here provides a reasonable response time. However, the logic expenditure is fairly large. As shown in Figure 5, the complete design using this technique also includes a C-element to collect the *done* signals from the two latch pairs and a D-element which manages the *start-done* handshake. Both of these are very quick and simple implementations which do not impede the performance of the overall circuit too much.

The handshake decoupling D-element is independently proposed in [2] and [4]. The implementation adopted here is suggested in [2]. This implementation comprising four two-input NOR gates and an inverter is shown in Figure 6. Being completely SI, this circuit has the STG shown in Figure 7, which is an economic way of managing the four-phase protocol for the overall *start-done* handshake.

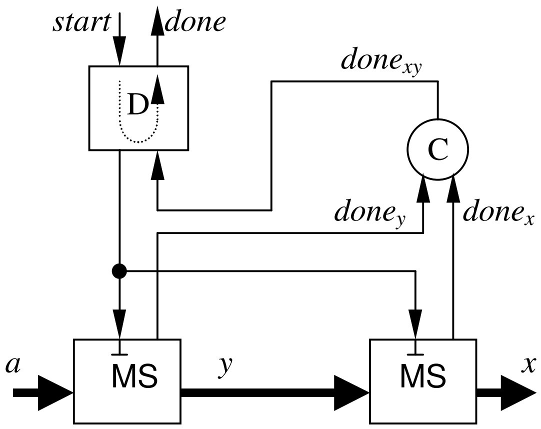

Figure 5 Self-timed shifting register solution.

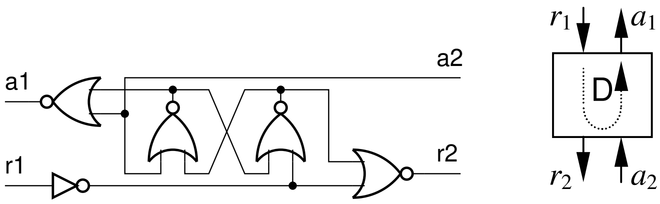

Figure 6 Speed independent D-element circuit and its symbol.

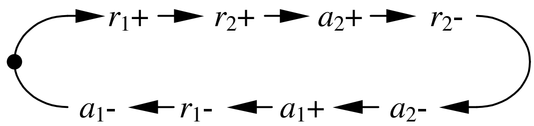

Figure 7 STG of D-element.

5 Decomposition and synthesis

The direct attempt to modify the shift register solution to a self-timed solution is not satisfactory. Further improvements are attempted by decomposing and refining the problem while using automatic synthesis whenever possible. This technique is employed to generated several solutions.

5.1 Simplified data path

The most obvious decomposition is to divide the data handling facilities from anything else.

Since it has been established that any solution of the problem must have two layers of latch action, the parallel solution does not provide any real benefits in response time, especially when there is a need to trigger the second layer of latch action by the completion of the first layer. If a single statement can be implemented with a single layer of latch action, a sequential solution would have the same response time as a parallel one, all other things being equal.

The logical next step is to try and find single layer latches that produce their own *done* signals and are simple in design, involving a minimal number of simple gates in the forward path. One such latch can then be used to implement each of the statements, providing a much simplified data path.

Such a basic latch with completion logic has been found in [2] and is shown in Figure 8. The completion logic is also very simple, compared with that found in Figure 4.

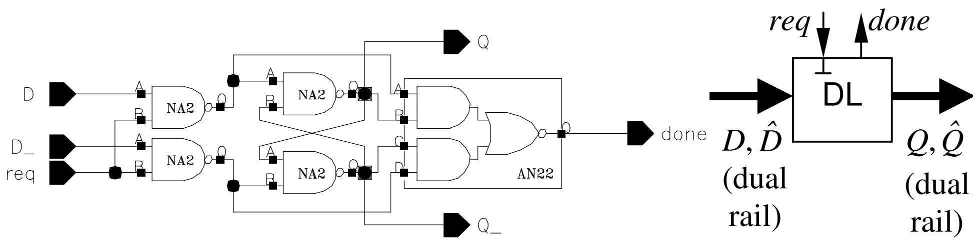

Figure 8 Speed independent latch circuit and its symbol.

By using two such latches in series, the system can be decomposed into three parts as shown in Figure 9. Here we adopt the conventional terminology of calling the input latch the "master" and the output latch the "slave" and index the signals of these latches with the letters *m* and *s*, respectively.

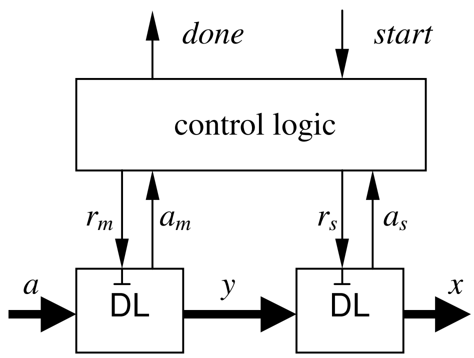

Figure 9 Two simple latches with control logic.

5.2 Sequential control logic design

The control logic is needed to maintain the protocols specified in Figure 3. A more detailed specification in the form of STG, comprising four-phase handshake protocols, is given in Figure 10.

The signal *sm* represents the state "*done*" of the slave latch and the "*start*" of the master latch. In other words, it is where the transition of action from the slave to the master happens. The entirely sequential nature of the STG specifies an SI design which is potentially delay insensitive, if the latches used are required to not issue acknowledgement reset signals (a_s- and a_m-) before their data inputs have been cut off. This is the case for the latch circuit in Figure 8.

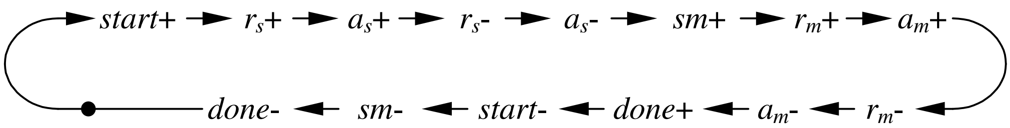

Figure 10 Specification STG of the control part of Figure 9.

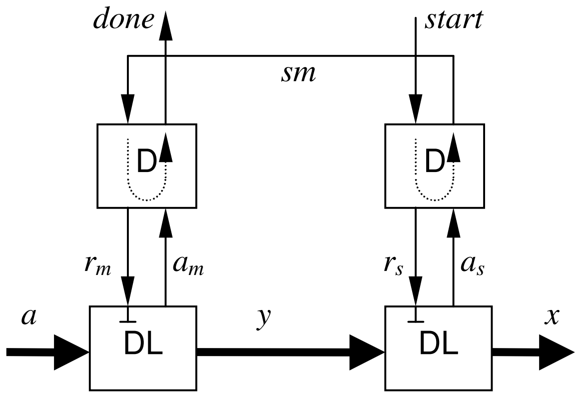

Figure 11 Simple control logic using D-elements.

By observing the STGs in Figure 10 and Figure 7, it is clear that handshake decoupling devices, such as the D-element, can be used to obtain a symmetrical design of the control part. Such a solution with D-elements is shown in Figure 11.

This solution has the advantage of employing simple elements, all of which have a small number of layers of simple gates with a minimum number of inputs in the forward path. It is also completely SI and delay insensitive. The control logic is implemented with two identical circuits (D-elements) which are connected by the intermediate signal *sm*.

The price paid to obtain delay insensitivity is to have a completely sequential STG including the resetting phases. If assumptions can be made about the resetting characteristics of the latches used, faster solutions may be obtained by adding parallelism into the control logic STG.

5.3 Parallel solution with direct `Petrify` synthesis

From Figure 8, it is clear that the input of the latch is cut off when the signal *r*- (r_m- or r_s-, depending on the latch) is received by it. This implies that the environment does not need to wait for the signal *a*- (a_m- or a_s-, depending on the latch) to safely change the data input. Assuming that *r*- affects the latch more quickly than it does the environment, this signal can be safely used to generate the *done* signal for the environment. This assumption is reasonable in the latch circuit we have chosen to use (Figure 8) and the likely environment in which it will be found. Specifically, we are proposing to implement the control logic and the latches on the same chip and arrange them adjacently in the layout. Therefore the wire delay for signal *r* from the control logic to the latch will be negligible and it will take one gate delay for *r*- to shut the data input of the latch off. The shortest time the environment will need to affect a change at the data input of the latch is likely to be many times that. If *r*- is used to generate *done*, the function of signal *a*- then becomes purely the maintenance of the four-phase handshake protocol.

Based on this assumption, a more efficient STG of the control logic is given in Figure 12. The resetting phases for both acknowledgements are run in parallel with the main path.

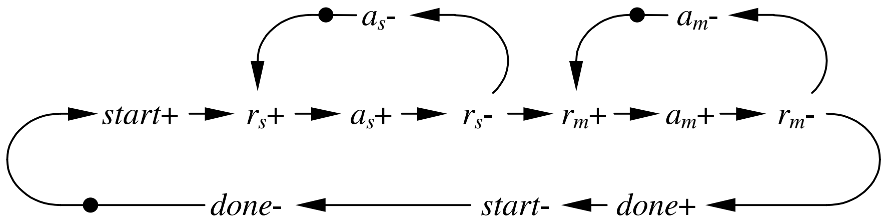

Figure 12 Parallel STG for the control logic in Figure 9.

The control logic can then be synthesised with `Petrify` based on this STG specification. However, although the STG is clearly regular including two essentially identical parts, the result from `Petrify` does not show any apparent regularity. This is given below:

$$done = \overline{csc_2} \cdot \overline{r_m} \,;$$
$$r_m = \overline{csc_1} \cdot \left(\overline{r_s} \cdot \overline{a_m} + r_m\right);$$
$$r_s = csc_1 \cdot \left(\overline{a_s} \cdot start \cdot csc_2 + r_s\right);$$
$$csc_1 = csc_1 \cdot \left(\overline{a_s} + r_s\right) + a_m \cdot \overline{csc_2} \,;$$
$$csc_2 = csc_2 \cdot \overline{r_m} + \overline{start} \,.$$

This circuit uses four latches and is excessively complex. The benefits of the simplified data path are negated by having the large control logic. Obviously, `Petrify` does not provide solutions with regularity.

5.4 Faster speed independent solution based on modified D-elements

A comparison of the STGs in Figure 10 and Figure 12 indicates that a manual decomposition of the control logic into two identical modified D-elements may provide the answer for a faster solution. The STG for such an element would need to be modified from the sequential one to the concurrent one shown in Figure 13.

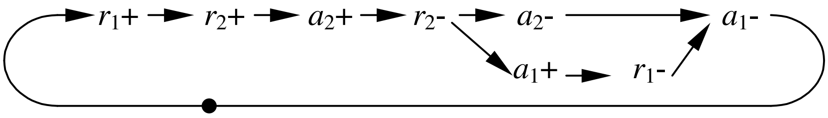

Figure 13 Modified STG of the control element for the first latch.

`Petrify` was used to synthesise a circuit with this modified STG. After a small manual optimisation, the modified D-element shown in Figure 14 is obtained.

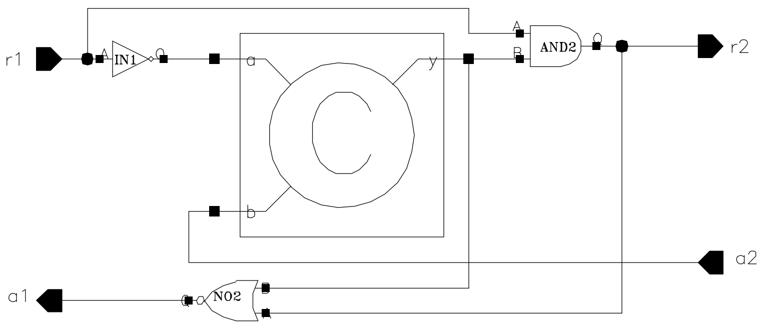

Figure 14 Modified D-element with parallel resetting of a_2.

Instead of the all-simple gate solution in Figure 6, this D-element contains a C-element. However, the C-element can be implemented as a symmetrical circuit with pass transistors, which is fast [5] and helps to realise the potential savings on time as prescribed by the modified STG of Figure 13.

A faster SI solution is obtained by using this version of D-element to control both latches in Figure 11.

5.5 Fast non speed independent solution

The performance of the circuit in Figure 11 can be further improved by modifying the handshake decoupling elements so that events r_s- and r_m+ happen "almost" simultaneously. "Almost" in this sense means that these are enabled at the same time and "fired" within a predictable small delay. Timing assumptions accepted in this example result in a non-SI circuit.

An STG of the modified decoupling element, called a B-element, is shown in Figure 15. Such a circuit provides the nesting of the setting handshake phases ($r_1+ \rightarrow r_2+ \rightarrow a_2+$) and the concurrent execution of the resetting phases ($a_1+ \rightarrow r_1-$ and $r_2- \rightarrow a_2-$). The dotted arc shows the actual causal relation in the circuit implementation.

The circuit implementation shown in Figure 16 is completely SI, though non persistent and non semi-modular. It may happen that the transition s- caused by a_1- becomes interrupted by r_1-, thus forming a glitch at s. However, this glitch does not result in a hazard at the output r_2, because signals arrive at the AND gate in the following order: $s- \rightarrow r_1- \rightarrow s+ \rightarrow r_1+$, i.e. monotonously.

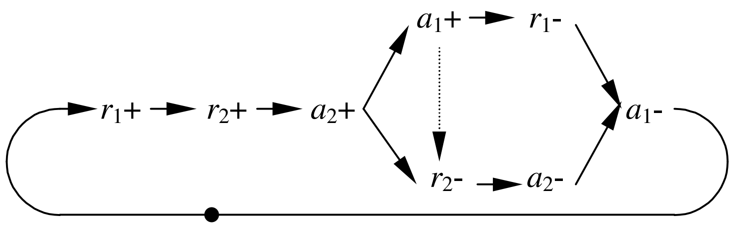

Figure 15 STG of B-element.

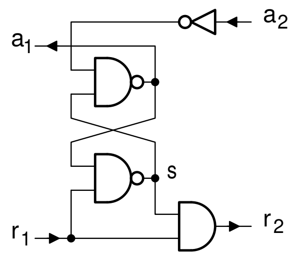

Figure 16 Circuit implementation of B-element.

If both D-elements in Figure 11 are replaced by B-elements, a possibility occurs that r_s- happens after r_m+ has brought a new data item to the input of the slave latch (change of y). This possibility is eliminated by careful consideration of gate delays. In the proposed implementation, a_s+ causes a chain of events, which branches into two parallel paths at a_1 of the right-hand decoupling element. Starting from the bifurcation point, it takes 3 gate delays (the AND gate is represented by a NAND-NOT pair) to propagate the signal to r_s. The number of gate delays in the path which goes through the left-hand decoupling element and the master latch to the data input of the slave latch is 4, which is sufficient [6] to guarantee the order r_s- $\rightarrow$ change of y.

6 Simulation results

All circuits presented here have been put through the VLSI design flow of Cadence, using the AMS-0.6µm toolkit, with some of the figures taken directly from their Cadence schematics. Analogue simulations using Cadence Spectre have been carried out for all circuits based on identical assumptions about the technology and operating environment. Overall response times (from *start*+ to *done*+) have been recorded for all circuits. These are given in Table 1.

SI M/S shifting register (Section 4)	4.61ns
D-element control logic (Section 5.2)	4.15ns
`Petrify` synthesised control logic (Section 5.3)	5.41ns
Modified D-element control logic (Section 5.4)	3.33ns
B-element control logic (Section 5.5)	1.69ns

Table 1 Performance comparison of all solutions.

The timing values are averages from multiple simulation runs in order to eliminate the effects of such factors as data values and initial conditions. The results confirm the intuitive notion that the majority of time is spent in the control logic, when fast data paths are employed. It is also quite clear that using the result of `Petrify` directly without having considered the optimum refinement and decomposition is not in general a good idea. The improvement obtained by using B-elements is significant.

As expected, no logical function violations or hazards were observed during all simulation sessions.

7 Conclusions and discussions

This work started out as the finding of *ad hoc* hardware solutions to the problem of two consecutive binary assignment statements with a requirement of self-timing and speed independence. The solutions themselves turned out to be elegant and systematic enough to be shown in their own right.

By concentrating on well known and simple elements it is possible to achieve better logic efficiency and faster response. The general techniques of these solutions can be applied to a much wider field where two latches need to be “strung up” sequentially in an SI manner. For instance, it can be used to implement the circuits which assemble two half bytes into a whole byte proposed in [7], or by extension, any similar circuits which assemble two data items of the same size into a data item twice the size (two bytes into a 16-bit word, etc.).

These solutions can be applied to sequential statements numbering more than two, in which case a modified shift register, with each master and slave pair connected using the techniques shown here can be employed.

These techniques can also be used to develop general purpose master-slave pairs that are SI internally.

The design methodology of decomposition/refinement combined with automatic synthesis has again been shown to be effective. The current shortcoming of such synthesis tools as `Petrify` in dealing with regular solutions has been demonstrated.

This work is supported by the EPSRC Comfort and HADES projects at Newcastle University.

References

1 F. Xia, D. Shang, A. Yakovlev and A.M. Koelmans, “An asynchronous communication mechanism using self-timed circuits”, elsewhere in this forum.

2 V. Varshavsky et al, Self-Timed Control of Concurrent Processes, Kluwer Academic Publishers, P.O. Box 17,3300 AA Dordrecht, The Netherlands, 1990 (Russian Edition: Nauka, Moscow,1986).

3 D.J. Kinniment, B. Gao, A. Yakovlev and F. Xia, “Towards asynchronous A-D conversion”, Proc. 4th Int. Symp. on Advanced Research in Asynchronous Circuits and Systems, San Diego, CA, pp. 206-215, IEEE Computer Society Press, 1998.

4 A. J. MARTIN, “Collected Papers on Asynchronous VLSI Design”, technical report Caltech-CS-TR-90-09, p. 50, Dept. of Computer Science, California Institute of Technology, 1990.

5 Shams, M., Ebergen, J.C., Elmasry, M.I., "Modelling and comparing CMOS implementations of the C-element", IEEE trans. on VLSI systems, vol. 6, No 4, pp. 563-567, December 1998.

6 Ch. Seitz, “System Timing”, ch. 7 in C. Mead & L. Conway “Introduction to VLSI Systems”, Addison Wesley, London, 1980.

7 A. Yakovlev, V. Varshavsky, V. Marakhovsky and A. Semenov, “Designing an asynchronous pipeline token ring interface”, Proc. of 2nd Working Conference on Asynchronous Design Methdologies, pp. 32-41, London, May 1995, IEEE Comp. Society Press, N.Y., 1995.