First International Conference
Modelling and Development of Intelligent Systems
Sibiu - Romania, 22-25 October, 2009

Algebraic model for the synchronous D flip-flop behaviour

Anca Vasilescu

Abstract

The complex systems like modern computers are designed using multiagent concepts. Agent interactions, namely communication and synchronization, fit very well the corresponding hardware components interactions. The agent-based approach is applied here to cover the digital logic circuits design and verification.

In order to achieve the computer components behaviour modelling and the formally verification of the involved agents properties, we use in this paper the SCCS as the appropriate process algebra supporting our interest of developing an algebraic model for the computer system behaviour.

In this paper we consider the specific structure for the synchronous D flip-flop as example for modern computer elementary memory cells. We define the appropriate specification and implementation agents for algebraic modelling of the given circuits behaviour and we formally prove the corresponding bisimilarities between the target agents. As application, we modify the obtained model in order to prepare an appropriate solution for algebraic modelling of a general synchronous logic circuit based on a D flip-flops matrix arrangement.

1 Introduction

The complex systems like modern computer systems are designed using multi-agent concepts. Agent interactions, namely communication and synchronization, fit very well the corresponding hardware components interactions. A component-based style allows components to be specified and verified individually. Larger combinations of trusted components can then be verified more easily. In this paper we use a correctness meaning based on the bisimilarity relation between agents.

There are two modern approaches for system verification, namely model checking and theorem proving. In this paper we apply a theorem proving approach based on the SCCS process algebra for modelling and verifying the appropriate agents provided by the D flip-flops bahaviour. Process algebra are well known as mathematical tools for describing and analysing the concurrent and communication systems like the result of interconnecting hardware components, each hardware component being modelled as an agent. Considering the digital logic level of the computer architecture description, the agent-based approach is applied here to cover both the digital logic circuits design and verification.

Having an algebraic-based model for a multi-agent system has the main advantage of ensuring the reliability and correctness of the core processes of the computer operation following the accurate results provided by a formal methods based verification approach.

The present results follow the contributions of the author already obtained in [4], [6], [7] for modelling the behaviour of different hardware components involved in a modern computer system architecture.

The final outcomes of this paper consist in defining the specification and implementation agents for modelling the synchronous D flip-flop behaviour, proving the bisimulation equivalence between the corresponding agents and preparing the appropriate agents for later modelling the behaviour of a complex memory component based on a matrix organization of D flip-flops.

2 Preliminaries

2.1 Flip-flops. Computer memory organization

Modern computer systems are based on digital components dealing with binary variables and with operations that assume a logical meaning. From the structural point of view, the manipulation of binary information is done by *logic circuits*, combinational or sequential. For a *combinational circuit* or CLC the outputs at any given time are entirely dependent on the inputs that are present at that time. Although every digital system is likely to have a combinational circuit, most systems encountered in practice also include storage, memory elements, which require that the system has to be described in terms of *sequential logic circuits* or SLC.

A *flip-flop* is a sequential circuit, a binary cell capable of storing one bit of information. It has two outputs, one for the normal value and one for the complement value of the bit stored in it. A flip-flop maintains a binary state until it is directed by a clock pulse to change that state. At the different levels of detailing, the theoretical flip-flop might be asynchronous, but the synchronous models are widely used in practice. The difference among various types of flip-flops is the number of inputs and the manner in which the inputs, both data inputs and the clock signal, affect the binary state. Depending on the number of data inputs, the most common types of flip-flops are: SR flip-flop, D flip-flop, JK flip-flop and T flip-flop [3]. In this paper we will consider the case of the synchronous D flip-flop structurally based on the synchronous SR flip-flop model.

Basically, a flip-flop is a one-bit memory. In order to store more data, we need to build a larger memory by linking several one-bit memories to form a cell, and then join many cells together such that one cell is on top of another and so on. In order to assure a straightforward modelling process for such a complex memory component, we prepare in this paper the detailed agents for modelling the behaviour of a specific SLC made up by an appropriate matrix of D flip-flops.

2.2 Process algebra SCCS

The process algebra SCCS, namely *Synchronous Calculus of Communicating Systems* is derived from CCS [1], [2] especially for achieving the synchronous interaction in the framework of modelling the concurrent communicating processes. In SCCS processes are built from a set of atomic actions A. Denoting the set of labels for these actions by Λ, an SCCS action is either (1) a *name* or an input on $a \in \Lambda$ denoted by a, (2) a *coname* or an output on $a \in \Lambda$ denoted by $\overline{a}$ or (3) an internal on $a \in \Lambda$ denoted by the action 1, identified with the empty product. In SCCS the *names* together with the *conames* are called the *particulate actions*, while an *action* $\alpha \in \Lambda^*$ can be expressed uniquely (up to order) as a finite product $a_1^{z_1} a_2^{z_2}$... (with $z_i \neq 0$) of powers of names. Note the usual convention that $a^{-n} = \overline{a}^n$.

An SCCS *process* P is defined with the syntax:

P	::=	nil	termination
	|	α:P	prefixing
	|	P+P	external choice
	|	P $\times$ P	product, synchronous composition
	|	P $\upharpoonright E$	restriction, $L \subseteq A \cup \overline{A}$ and $E = (A\text{-}L)^*$
	|	P$[f]$	relabelling with the morphism $f : A \cup \overline{A} \to A \cup \overline{A}$

In the restriction definition $E = (A\text{-}L)^*$ is the submonoid of A generated by the set difference A-L. By definition, the P $\upharpoonright E$ agent is forced to execute only the actions from the set E as the external actions.

The operational semantics for SCCS is given via inference rules that define the transition available to SCSS processes. Combining the product and the restriction, SCCS calculus defines the synchronous interaction as a multi-way synchronization among processes.

A formal approach such as the process algebra SCCS supports a way to relate two different specifications in order to show that those specifications actually describe *equivalent* concurrent systems, for some specific meaning of *equivalence*. In this section we use a concrete relation between two different specifications – a notion of refinement:

Impl *refines* Spec

where a low-level specification, namely Impl, refines a higher-level specification, namely Spec. For each of the next circuits, we construct both of these specifications as follows. The specification Spec is based on the definition of the circuit, while the specification Impl is based on the behaviour of that given circuit. As demonstration technique, we start with the specifications Spec and Impl and then we apply a set of SCCS-based algebraic laws in order to formally prove that the low-level specification, Impl, is correct with respect to the higher-level one, Spec. This correctness proof is based on the *bisimulation congruence*, the appropriate equivalence in the theory of concurrent communicating processes. Implicitly, this result of bisimilarity shows that the behaviour follows the definition of the given system and, on the other hand, it is a guarantee of using that model in other complex circuits.

3 Algebraic model for the synchronous D flip-flop behaviour

In this section we firstly define both specification and implementation agents for asynchronous SR flip-flops, synchronous SR flip-flops and synchronous D flip-flops, respectively. In the second part of this section we formally prove the corresponding agents bisimilarities.

It is helpfully for the next sections to define some generic agents, as follows.

$$\begin{gathered}
\text{NODE} = in\,\overline{out} : \text{NODE} \\
\text{NODE1} = \sum_{i\in\{0,1\}} (in_i\overline{out}_i : \text{NODE1}) \\
\text{NODE2} = \sum_{i\in\{0,1\}} (in_i\overline{up}_i\overline{down}_i : \text{NODE2}) \qquad (1) \\
\text{NOT} = \sum_{i\in\{0,1\}} (in_i\overline{out}_j : \text{NOT}), \text{ where } j = \text{NOT } i \\
\text{AND\_rec} = \sum_{i,j\in\{0,1\}} (andin1_i\; andin2_j\; \overline{andout}_k : \text{AND\_rec}), \text{ where } k = i \text{ AND } j.
\end{gathered}$$

3.1 Algebraic model for the SR flip-flop behaviour

The logic diagram of an SR flip-flop consists in two NOR gates composed like in Figure 1. For the next algebraic models we will recognize the circuit lines as follows: σ for the input value S, ρ for the input value R, $\overline{\delta}$ for the output value Q and $\overline{\gamma}$ for the complemented output value $Q' = \text{NOT } Q$.

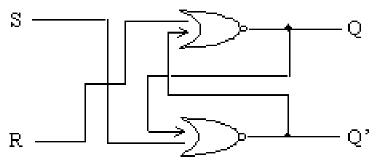

Figure 1: Asynchronous SR flip-flop

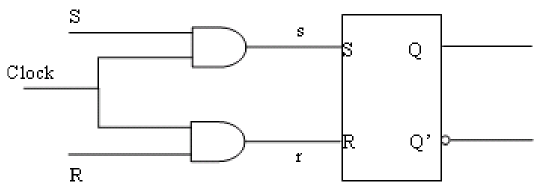

Figure 2: Synchronous SR flip-flop

The current value of a variable like σ_S shows that the circuit line σ is carrying the logic value S. If the current values on the outputs $\overline{\gamma}$ and $\overline{\delta}$ are m and n, respectively, then the specification Spec for the SR flip-flop behaviour might be [1]

$$\text{SpecSR}(m,n) = \sum_{i,j\in\{0,1\}} (\sigma_i\rho_j\overline{\gamma}_m\overline{\delta}_n : \text{SpecSR}(k,l)) \qquad (2)$$

where the values k and l are defined by $k = i$ NOR n and $l = j$ NOR m.

In order to achieve the composition and to assure the fork of the output signal, we have to define two morphisms to make two appropriately relabelled copies of the NOR gate. These two morphisms are: Φ defined by the relabelling pairs $\alpha_i \mapsto \sigma_i$ and $\gamma_i \mapsto \gamma_i\alpha_i$, for $i \in \{0,1\}$ and respectively Ψ defined by the relabelling pairs $\beta_i \mapsto \rho_i$ and $\gamma_i \mapsto \beta_i\delta_i$, for $i \in \{0,1\}$.

We also define the set E of external actions, $E = \{\sigma_i, \rho_i, \gamma_i, \delta_i | i \in \{0,1\}\}$, in order to form the SCCS product between the two communicating NOR gate-agents via the *lines* $(\alpha_i, \overline{\alpha}_i)$ and $(\beta_i, \overline{\beta}_i)$. We usually abbreviate the subsets like $\{\alpha_0, \alpha_1\}$ by α.

Hence, the SCCS low-level specification (or implementation) for the SR flip-flop behaviour might be

$$\text{ImpSR}(m,n) = (\text{NOR}(m)[\Phi] \text{ x } \text{NOR}(n)[\Psi]) \upharpoonright E \tag{3}$$

This example shows how appropriate combinations of morphism, product and restriction may be used to model the channelling of data, here along the hardware wires.

Based on ideas from [1], it is also shown in [5] that the two previous specifications for the SR flip-flop behaviour are bisimulation equivalent (or bisimilar). So, the relation:

$$\text{SpecSR}(m,n) \sim \text{ImpSR}(m,n) \tag{4}$$

is established.

From the structural point of view, in order to obtain the synchronous SR flip-flop we consider the asynchronous circuit and we add an extra level of AND gates for involving the clock signal. The synchronous SR flip-flop is represented in Figure 2. We consider new indexed variables, namely CLK, for representing the clock input signal following the index values $c \in \{0,\ 1\}$.

We define two levels for specifying the synchronous SR flip-flop behaviour, a specification and an implementation, given by the agents, respectively:

$$\text{SpecSRs}(m,n,c) = (\text{SpecInput}(c) \times \text{SpecSR}(m,n)) \upharpoonright E\_SRmn(c) \tag{5}$$

and

$$\text{ImpSRs}(m,n,c) = (\text{ImpInput}(c) \times \text{ImpSR}(m,n)) \upharpoonright E\_SRmn(c) \tag{6}$$

where the set of external actions is $E\_SRmn(c) = \{CLK_c, \sigma, \rho, \gamma, \delta\}$ for a specific value of $c \in \{0,1\}$. Note that the parameters m and n have complemented values, by definition of the flip-flop.

It is important for the next results of this paper that we have just prove in [8] that these two agents are bisimulation equivalent based on the relation

$$\text{SpecSRs}(m,n,c) \sim \text{ImpSRs}(m,n,c) \tag{7}$$

3.2 Algebraic model for the synchronous D flip-flop behaviour

A D flip-flop is derived from an SR flip-flop by replacing the R input with an inverted version of the S input, which thereby becomes D like in Figure 3. For the synchronous D flip-flop it is essential that when the clock is reset the circuit does not operate, meaning it does not change the state, and when the clock is set the D flip-flop loads the D input.

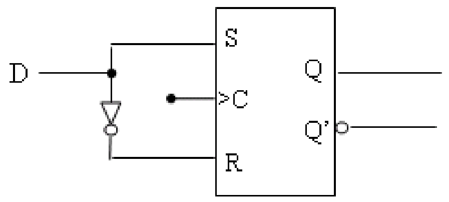

Figure 3: Synchronous D flip-flop

In this section we propose the SCCS algebraic models for both the specification and the implementation of the synchronous D flip-flop behaviour and we conclude by proving the bisimilarity of these two models.

Based on the previous definition of the synchronous D flip-flop, we consider its internal structure consisting of an entrance level of gates and an internal synchronous SR flip-flop. We define the agent SpecInD for specifying the behaviour of the entrance level of the D flip-flop as follows.

$$\text{SpecInD} = \sum_{D \in \{0,1\}} (\zeta_D \overline{\sigma}_S \overline{\rho}_R : \text{SpecInD}) \tag{8}$$

where $S = D$ and $R = \text{NOT } D$.

The appropriate specification agent for the synchronous D flip-flop behaviour is given by the relation

$$\text{SpecDs}(m,n,c) = (\text{SpecInD} \times \text{SpecSRs}(m,n,c)) \upharpoonright E\_Dmn(c) \tag{9}$$

where the set of external actions is $E\_Dmn(c) = \{CLK_c, \zeta, \gamma, \delta\}$ for each value of $c \in \{0,1\}$.

Let the agent ImpInD be the implementation agent for the entrance level of the D flip-flop. As expected, this lower-level specification is based on the operation of the intercommunicating logic gates combination. Using the generic agents already defined in (1), we have

$$\text{NODE2\_D} = \text{NODE2}[in_i \mapsto \zeta_D, up_i \mapsto \sigma_S] \text{ for each } D = i \text{ and } S = i$$

$$\text{NOT\_R} = \text{NOT}[in_i \mapsto down_i, out_j \mapsto \rho_R] \text{ for each } R = j \text{ and } j = \text{NOT } i.$$

We define the ImpInD agent as follows

$$\text{ImpInD} = (\text{NODE2\_D} \times \text{NOT\_R}) \upharpoonright E\_ImpInD \tag{10}$$

where the set of external actions is $E\_ImpInD = \{\zeta, \sigma, \rho\}$.

The appropriate implementation agent for the synchronous D flip-flop behaviour is given by the relation

$$\text{ImpDs}(m,n,c) = (\text{ImpInD} \times \text{ImpSRs}(m,n,c)) \upharpoonright E\_Dmn(c) \tag{11}$$

where the set $E\_Dmn(c)$ of external actions is the same as in the specification case.

Proposition 1 *The previous agents SpecDs(m,n,c) and ImpDs(m,n,c) for $(m,n) \in \{(0,1),(1,0)\}$ and $c \in \{0,1\}$ are bisimulation equivalent.*

Proof: The bisimulation relation '$\sim$' is a congruence over the class $\mathcal{P}$ of agents [1], [2]. Besides, it is already established in the previous relation (7) that SpecSRs$(m,n) \sim$ ImpSRs(m,n). Comparing the definitions (9) and (11) for the target agents SpecDs(m,n,c) and ImpDs(m,n,c) respectively, it follows that we only need to prove that:

$$\text{SpecInD} \sim \text{ImpInD}$$

We first evaluate the lower-level specification ImpInD:

$$\begin{aligned}
\text{ImpInD} &= (\text{NODE2\_D} \times \text{NOT\_R}) \upharpoonright E\_ImpInD = \\
&= (\text{NODE2}[in_i \mapsto \zeta_D, up_i \mapsto \sigma_S] \times \text{NOT\_R}[in_i \mapsto down_i, out_j \mapsto \rho_R]) \upharpoonright \\
\upharpoonright\ &\{\zeta, \sigma, \rho\} = \\
&= (\sum_{i \in \{0,1\}} (\zeta_D \overline{\sigma}_S \overline{down_i} : \text{NODE2\_D}) \times \sum_{i \in \{0,1\}} (down_i \overline{\rho}_R : \text{NOT\_R})) \upharpoonright \\
\upharpoonright\ &\{\zeta, \sigma, \rho\}
\end{aligned}$$

The parameter values are: $D = i$, $S = i$, $R = j$ and $j = \text{NOT } i$.
We apply the specific SCCS operational laws and the result is

$$\begin{aligned}
\text{ImpInD} &= \sum_{D \in \{0,1\}} (\zeta_D \overline{\sigma}_S \overline{\rho}_R : (\text{NODE2\_D} \times \text{NOT\_R}) \upharpoonright E\_ImpInD)) = \\
&= \sum_{D \in \{0,1\}} (\zeta_D \overline{\sigma}_S \overline{\rho}_R : \text{ImpInD})
\end{aligned} \tag{12}$$

with $S = D$ and $R = \text{NOT } D$.
Comparing the definition (8) of the agent SpecInD with the previously proved relation (12) for the agent ImpInD, it follows that both these agents are solutions for the same equation $X = \sum_{D\in\{0,1\}}(\zeta_D\overline{\sigma}_S\overline{\rho}_R : X)$ up to a permutation of actions in the multi-particulate prefix action. Because here the variable X is guarded, the equation has a unique solution up to bisimilarity [2], so that the agents SpecInD and ImpInD are bisimilar, as required. □

We have just proved that

$$\text{SpecDs}(m, n, c) \sim \text{ImpDs}(m, n, c) \tag{13}$$

3.3 Specific synchronous D flip-flop models for SLC integration

In order to model the sequential circuits we have to apply two modifications to both the specification and the implementation of the synchronous D flip-flop agents as follows. The first modification consists in making obvious the initial state of the D flip-flop and defining separate agents for the D flip-flop operation with each of the input value $D \in \{0, 1\}$. The final agents of this step are modelling the synchronous D flip-flop operation starting from a specific initial state, with specific value on the D input and a certain clock signal. The second modification consists in unifying the previous agents by selecting the D flip-flop operation for a specific input value $D \in \{0, 1\}$, but for all the initial state combinations and all the clock signal combinations.

As the first modification we define the next agents for making obvious the initial state Q_n of the flip-flop and for selecting the D flip-flop operation for a specific Input value $D \in \{0, 1\}$:

$$\text{InDisI}(D, n) = Q_n\zeta_D\overline{\zeta}_D : \text{InDisI}$$

where $D, n \in \{0, 1\}$.

The appropriate specification agents are:

$$\text{SpecDs\_isI}(m, n, c, D) = (\text{InDisI}(D, n) \times \text{SpecDs}(m, n, c)) \upharpoonright E\_DsisI(c, D) \tag{14}$$

where the set of external actions is $E\_DsisI(c, D) = \{Q, \gamma, \delta, CLK_c, \zeta_D\}$ and the parameters are $(m, n) \in \{(0, 1), (1, 0)\}$, $c \in \{0, 1\}$ and $D \in \{0, 1\}$.

The appropriate implementation agents are:

$$\text{ImpDs\_isI}(m, n, c, D) = (\text{InDisI}(D, n) \times \text{ImpDs}(m, n, c)) \upharpoonright E\_DsisI(c, D) \tag{15}$$

with the same set of external actions as in the specification case.

Proposition 2 *For all $(m, n) \in \{(0, 1), (1, 0)\}$, $c \in \{0, 1\}$ and $D \in \{0, 1\}$ the previous agents SpecDs_isI(m, n, c, D) and ImpDs_isI(m, n, c, D) are bisimulation equivalent.*

Proof: The result is obvious considering that the bisimulation relation '$\sim$' is a congruence over the class $\mathcal{P}$ of agents [1], [2] and it is already established in the previous relation (13) that SpecDs$(m, n, c) \sim$ ImpDs(m, n, c). □

As the second modification we define the appropriate agents for unifying the previous agents by selecting the D flip-flop operation for a specific input value $D \in \{0, 1\}$, but for all the initial state combinations and all the clock signal combinations. The specification agents are:

$$\text{SpecCBBDisI}(D) = (\sum_{\substack{(m,n)\in\{(0,1),(1,0)\}\\ c\in\{0,1\}}} SpecDs\_isI(m, n, c, D))[\delta_n \mapsto Q_n^*, \gamma_m \mapsto 1] \tag{16}$$

for each value $D \in \{0, 1\}$ and the implementation agents are:

$$\text{ImpCBBDisI}(D) = (\sum_{\substack{(m,n)\in\{(0,1),(1,0)\}\\ c\in\{0,1\}}} ImpDs\_isI(m, n, c, D))[\delta_n \mapsto Q_n^*, \gamma_m \mapsto 1] \tag{17}$$

for each value $D \in \{0, 1\}$.

Proposition 3 *The previous agents SpecCBBDisI(D) and ImpCBBDisI(D) for $D \in \{0,1\}$ are bisimulation equivalent.*

Proof: The result is obvious considering that the bisimulation relation '$\sim$' is a congruence over the class $\mathcal{P}$ of agents [1], [2] and it is already established in the previous Proposition 2 that SpecDs_isI(m,n,c,D) $\sim$ ImpDs_isI(m,n,c,D) for all $(m,n) \in \{(0,1),(1,0)\}$, $c \in \{0,1\}$ and $D \in \{0,1\}$. □

It follows to use these results for modelling varied examples of sequential logic circuits behaviour, especially those involving the D flip-flops.

4 Conclusions

Based on ideas from [1], [5], in this paper we have considered the internal structure of specific memory cells, namely SR flip-flop and D flip-flop. For both the asynchronous and synchronous circuit organization, we have defined specific agents for modelling the concrete flip-flops behaviour.

The authors' scientific contributions in this paper refer mainly two directions. Firstly, we have defined appropriate SCCS agents at two different specification levels for all of the agents involved in the hardware components structure we were interested in. Secondly, we have proved all the corresponding agents bisimilarities.

An immediate extension of this paper will consist in assembling all these results in order to have an appropriate model for a computer memory component. Another direction for this work progress will be the use of these final agents into a more complex digital logic circuit like a standard read/write memory hardware component with an input/output controller unit integrated. For the interest of modelling the behaviour of such a complex memory component, we have prepared here the detailed involved agents, as application of this paper theoretical results concerning the internal D flip-flops bahaviour models.

After the modelling achievements from [4], [6], [7], these modelling attempts represent a further step in the direction of having a global algebraic model of the entire computer system behaviour. Both the properties of the SCCS calculus and the computer organization are supporting our interest of developing a scalable, open algebraic models hierarchy for hardware components.

References

[1] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25, pp. 267–310, 1983.

[2] R. Milner. *Communication and concurrency*, Prentice Hall, 1989.

[3] M.M. Mano. *Digital Logic and Computer Design*, Prentice-hall Of India Pvt Ltd, 2007.

[4] A. Vasilescu. Algebraic Model for the Behaviour of the Computer's Arithmetic-Logic Unit. *ICAM Abstracts*, pp. 50, 2006.

[5] A. Vasilescu. Formal models for non-sequential processes. *PhD Thesis, Babeş-Bolyai University of Cluj-Napoca*, 2006.

[6] A. Vasilescu. Algebraic model for the intercommunicating hardware components behaviour. *New Aspects of Computers, Proc. of the 12th WSEAS International Conference on COMPUTERS*, pp. 241-246, 2008.

[7] A. Vasilescu. Counter register. Algebraic model and applications. *WSEAS TRANSACTIONS on COMPUTERS*, Issue 10, Vol. 7, pp. 1618-1627, Oct. 2008.

[8] A. Vasilescu. Algebraic model for the synchronous SR-flip-flop behaviour. *Proc. of the Intl. Conf. on Knowledge Engineering, Principles and Techniques, KEPT2009, Special Issue of Studia Universitatis Babes-Bolyai Informatica*, Anul LIV, pp. 235-238, 2009.

Anca Vasilescu
Transilvania University of Braşov
Department of Theoretical Computer Science
Str. Iuliu Maniu nr. 50, 500091 Braşov
ROMANIA
E-mail: *vasilex@unitbv.ro*