Bulletin of the *Transilvania* University of Braşov • Vol 7(56), No. 2 - 2014
Series III: Mathematics, Informatics, Physics, 203-210

# BASIC TYPES OF FLIP-FLOPS: SPECIFICATION AND AUTOMATIC VERIFICATION

**Anca VASILESCU**[1]

**Abstract**

The computers hardware components are regarded as very modern real systems, properly to be modeled through formal methods. Specifically, our interests are concerning an algebraic prototype for the entire computer behaviour based on the interconnected hardware components. The authors contributions in this paper are following two directions: presenting the original specification and implementation agents for modelling all the four types of basic flip-flop circuits behaviour and applying automatic verification of the corresponding agents equivalences. These results represent the background of the sequential part of our prototype.

2000 *Mathematics Subject Classification:* 68M07, 68Q60, 68Q85.
*Key words:* automatic verification, bisimulation equivalence, hardware, process algebra.

## 1 Introduction

In the framework of modern research, an important direction is to use an algebra-based calculus for analyzing and modelling the behaviour of a specific class of real systems, both hardware [2] and software [9]. Moreover, the mathematics is ready to value the formal methods support, especially for developing the automated provers and for obtaining fully verified axiomatic proofs of substantial mathematical theorems [3].

Out of these examples, the computers hardware components are also regarded as very modern real systems properly to be modeled through formal methods. A formal-based approach focuses on the communication and synchronization between the involved components and also provides a valuable solution for systematically and exhaustively analysis of the interconnected computer components behaviour in order to prove the correctness and to avoid the bugs before the real circuit assembly.

[1]Faculty of Mathematics and Informatics, *Transilvania* University of Braşov, Romania, e-mail: vasilex@unitbv.ro

Specifically, our interests are concerning an algebraic model for the entire computer behaviour based on the interconnected hardware components. Following this approach, our prototype already consists of many agents which are modelling the internal combinational and/or sequential logic structure of specific hardware components behaviour. Starting from a given hardware component operation, the appropriate modelling agents are defined with respect to the SCCS/CWB-NC syntax. Using the operational semantics of the given SCCS algebra [8], we have evaluated and formal verified if the proposed implementation-based model relates to the intended specification-based definition of the given component behaviour. Further, as an extra mark for our model correctness, the CWB-NC platform [15], as a tool based on the state space method, has to be used for the automatic verification of the model. As final results, we have to collect the TRUE answers from the CWB-NC platform for all of the appropriate verified equivalences between the specification and corresponding implementation agents. These CWB-NC answers authenticate the theoretical result previously proved using the SCCS operational semantics.

Out of these achievements, the authors contributions in this paper consist in presenting the original specification and implementation agents for modelling all the four types of flip-flop circuits behaviour and the CWB-NC automatic verification of these agents equivalences. This represents the background of the sequential part of our prototype. Because SCCS could scale up easily, starting with the model of the flip-flops we were able to continue with specifying the behaviour of many flip-flop-based component, namely memories [11], registers [12] or data-transfer components. For the benefit of our research direction, these results are acknowledged by the recent paper [7].

## 2 Main results

The final outcome of this paper consists in developing an algebraic model of communicating and synchronized hardware components given by flip-flops represented at digital logic level. An algebraic approach is used here not only for studying the general concurrent communicating processes, but for applying it in a practical area, namely the computer architecture and organization. We benefit from the Milner's process algebra SCCS - the synchronous calculus derived from CCS (*Calculus of Communicating Processes*) [8] and we shall combine this process algebra and the automata theory by using the CWB-NC (*Concurrency WorkBench*) platform [15] for automatic verification of the targeted models. Using together SCCS and CWB-NC we have many advantages, such as: CWB-NC recognizes the SCCS specification files, CWB-NC can simulate the behaviour of the system specified in SCCS and, moreover, the CWB platform can automatically verify many types of equivalences between models, including bisimilarity as the most appropriate equivalence between SCCS specifications of the targeted system behaviour.

Following the structural point of view, the targeted models are based on the behaviour of four types of flip-flops [7], as follows: the $D$ flip-flops, based on internal $SR$ flip-flops, the $T$ flip-flops, based on internal $JK$ flip-flops, which are consequently based on the $SR$ flip-flops. For each of these hardware components we define the appropriate SCCS agents for two different specifications: a high-level one, based on the definition of the specific type of circuit, and a lower-level one, based on the structure of the internal communicating logic gates combinations. Explicit CWB-NC automated verifications of the bisimulation equivalence between the corresponding specifications are proving our model correctness.

A *flip-flop* is a sequential circuit, a binary cell capable of storing one bit of information. Its number of inputs varies from one flip-flop type to another, but it always 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 comes from the manner in which the inputs, both data inputs and the clock signal, affect the current 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 [7].

For the interest of this research, we present here the specific case of the synchronous flip-flops behaviour starting from a specific current state $(m, n) \in \{(0, 1)\}$ and for the clock signal $c = 1$. The values $m$ and $n$ represent the current values on the circuit outputs and the $c$ is for the clock input signal. This combination in addition with its symmetrical $(m, n) \in \{(1, 0)\}$ and $c = 1$ are the most important cases in analyzing the real computer systems operation.

We have obtained successful CWB-NC automatic verification for the corresponding specification agents and for the bisimilarity tests. These CWB-NC answers are guarantees for the correctness of our model and endorsed the involved agents as prerequisites in more complex specifications for scaling up the prototype.

## 2.1 SR flip-flop

From the structural point of view, we are interested here in modelling the synchronous $SR$ flip-flop behaviour. Its structure consists of an asynchronous circuit plus an extra level of AND gates for involving the clock signal. Hence, the definition of the synchronous $SR$ flip-flop is based on the asynchronous $SR$ flip-flop structure. We consider two levels for specifying the synchronous $SR$ flip-flop behaviour, a specification and an implementation, and we conclude with the equivalence result for these models.

The corresponding specification and implementation agents for the $SR$ flip-flop behaviour are:

```
*************** SPECIFICATION AGENTS *******************
set Com_SRs1 = {CLK1, S0, S1, R0, R1, m0, m1, n0, n1}
proc SpecClock1 = 'CLK1.S0.R0.~s0.~r0':SpecClock1 +
                  'CLK1.S0.R1.~s0.~r1':SpecClock1 +
                  'CLK1.S1.R0.~s1.~r0':SpecClock1 +
                 ‘CLK1.S1.R1.~s1.~r1’:SpecClock1

proc SpecSR01s1 = (SpecClock1 # SpecSR01) ! Com_SRs1

****************IMPLEMENTATION AGENTS*******************
proc AND = ‘andin10.andin20.~andout0’:AND +
           ‘andin10.andin21.~andout0’:AND +
           ‘andin11.andin20.~andout0’:AND +
           ‘andin11.andin21.~andout1’:AND
proc ImpClock1 = ‘CLK1.~Cup1.~Cdown1’:ImpClock1
proc AND_S = AND [andin10/‘S0’, andin11/‘S1’, andin20/‘Cup0’,
                  andin21/‘Cup1’, andout0/‘s0’, andout1/‘s1’]
proc AND_R = AND [andin10/‘R0’, andin11/‘R1’, andin20/‘Cdown0’,
                  andin21/‘Cdown1’, andout0/‘r0’, andout1/‘r1’]
set Com_Level1s1 = {CLK1, S0, S1, R0, R1, s0, s1, r0, r1}
proc Level1s1 = (ImpClock1 # AND_S # AND_R) ! Com_Level1s1

proc ImpSR01s1 = (Level1s1 # ImpSR01) ! Com_SRs1
```

For this Spec-Imp pair of agents, we have to verify the appropriate bisimilarity. The corresponding CWB-NC answer for the automated verification is TRUE, like in Figure 1, and the same result is formally proved in [10] using the SCCS operational semantics.

```
cwb-nc> eq SpecSR01s1 ImpSR01s1
Building automaton...
States: 10
Transitions: 40
Done building automaton.
Transforming automaton...
Done transforming automaton.
TRUE
Execution time (user, system, gc, real) : (0.111, 0.000, 0.001, 0.111)
cwb-nc>
```

Figure 1: The CWB-NC answer for SpecSR01s1 $\sim$ ImpSR01s1

## 2.2 D flip-flop

A $D$ flip-flop is derived from an $SR$ flip-flop by replacing the $R$ input with an inverted version of the $S$ input. 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. The next specification and implementation agents model the synchronous $D$ flip-flop operation:

```
*************** SPECIFICATION AGENTS ****************************
set Com_Ds1 = {CLK1, D0, D1, m0, m1, n0, n1}
proc SpecInD = ‘D0.~S0.~R1’:SpecInD + ‘D1.~S1.~R0’:SpecInD

proc SpecD01s1 = (SpecInD # SpecSR01s1) ! Com_Ds1
```

```
**************** IMPLEMENTATION AGENTS ***************************
proc NOT = ‘in0.~out1’:NOT + ‘in1.~out0’:NOT
proc NODE = ‘in0.~up0.~down0’:NODE + ‘in1.~up1.~down1’:NODE
proc Gate1 = NODE [in0/‘D0’,in1/‘D1’,up0/‘S0’,up1/‘S1’]
proc Gate2 = NOT  [in0/‘down0’,in1/‘down1’,out0/‘R0’,out1/‘R1’]
set Com_ImpInD = {D0,D1,S0,S1,R0,R1}
proc ImpInD = (Gate1 # Gate2) ! Com_ImpInD

proc ImpD01s1 = (ImpInD # ImpSR01s1) ! Com_Ds1
```

For this Spec-Imp pair of agents, we have formally proved the bisimulation equivalence of these agents in [11] and that theoretical result is automated verified here. Favorably, the corresponding CWB-NC answer is TRUE, like in Figure 2.

```
cwb-nc> eq SpecD01s1 ImpD01s1
Building automaton...
States: 8
Transitions: 16
Done building automaton.
Transforming automaton...
Done transforming automaton.
TRUE
Execution time (user, system, gc, real) : (0.041, 0.000, 0.000, 0.041)
cwb-nc>
```

Figure 2: The CWB-NC answer for SpecD01s1 $\sim$ ImpD01s1

## 2.3 JK flip-flop

A $JK$ flip-flop is a refinement of the $SR$ flip-flop in that the indeterminate condition of the $SR$ type is defined in the $JK$ type. Inputs $J$ and $K$ behave like inputs $S$ and $R$, respectively, in order to set and to clear the flip-flop current state, respectively. When inputs $J$ and $K$ are both equal to 1, a clock transition switches the outputs of the flip-flop to their complement state. Following these definitions, the specific agents for $JK$ flip-flop behaviour are:

```
******************** SPECIFICATION AGENTS ********************
set Com_JKs1 = {CLK1,J0,J1,K0,K1,m0,m1,n0,n1}
proc InJK0100 = ‘J0.K0.~S0.~R0’:InJK0100
proc InJK0101 = ‘J0.K1.~S0.~R1’:InJK0101
proc InJK0110 = ‘J1.K0.~S0.~R0’:InJK0110
proc InJK0111 = ‘J1.K1.~S0.~R1’:InJK0111

proc SpecJK01s1 = (InJK0100 # SpecSR01s1) ! Com_JKs1 +
                  (InJK0101 # SpecSR01s1) ! Com_JKs1 +
                  (InJK0110 # SpecSR01s1) ! Com_JKs1 +
                  (InJK0111 # SpecSR01s1) ! Com_JKs1

******************* IMPLEMENTATION AGENTS *******************
proc AND00 = ‘in0.~out0’:AND00
proc AND01 = ‘in1.~out0’:AND01
proc AND10 = ‘in0.~out0’:AND10
proc AND11 = 'in1.~out1':AND11
proc ANDJS00 = AND00 [in0/'J0',out0/'S0']
proc ANDJS01 = AND01 [in1/'J1',out0/'S0']
proc ANDJS10 = AND10 [in0/'J0',out0/'S0']
proc ANDJS11 = AND11 [in1/'J1',out1/'S1']
proc ANDKR00 = AND00 [in0/'K0',out0/'R0']
proc ANDKR01 = AND01 [in1/'K1',out0/'R0']
proc ANDKR10 = AND10 [in0/'K0',out0/'R0']
proc ANDKR11 = AND11 [in1/'K1',out1/'R1']
proc InJK01 = ANDJS00 # ANDKR10 + ANDJS00 # ANDKR11 +
              ANDJS01 # ANDKR10 + ANDJS01 # ANDKR11

proc ImpJK01s1 = ( InJK01 # ImpSR01s1 ) ! Com_JKs1
```

We have formally proved the bisimulation equivalence of these specification and implementation agents in [13] and that theoretical result is automated verified here. Favorably, the corresponding CWB-NC answer is TRUE, like in Figure 3.

```
cwb-nc> eq SpecJK01s1 ImpJK01s1
Building automaton...
States: 14
Transitions: 20
Done building automaton.
Transforming automaton...
Done transforming automaton.
TRUE
Execution time (user, system, gc, real) : (0.081, 0.000, 0.000, 0.081)
cwb-nc>
```

Figure 3: The CWB-NC answer for SpecJK01s1 $\sim$ ImpJK01s1

## 2.4 T flip-flop

By definition, a $T$ flip-flop is obtained from a $JK$ type with respect to the next rule: when $T = 0$ a clock transition does not change the state of the flip-flop and when $T = 1$ a clock transition complements the state of the flip-flop. For the current state $(m, n) = (0, 1)$ and the clock signal $c = 1$, the corresponding specification and implementation agents are:

```
******************** SPECIFICATION AGENTS *******************
set Com_Ts1 = {CLK1, T0,T1,m0,m1,n0,n1}
proc InT0 = 'T0.~J0.~K0':InT0
proc InT1 = 'T1.~J1.~K1':InT1

proc SpecT01s1 = (InT0 # SpecJK01s1) ! Com_Ts1 +
                 (InT1 # SpecJK01s1) ! Com_Ts1

******************* IMPLEMENTATION AGENTS *******************
proc NODE = 'in0.~up0.~down0':NODE + 'in1.~up1.~down1':NODE
proc InT = NODE [in0/'T0',in1/'T1',up0/'J0',up1/'J1',down0/'K0',down1/'K1']

proc ImpT01s1 = (InT # ImpJK01s1) ! Com_Ts1
```

The theoretical result that these two agents are bisimulation equivalent is formally proved in [12] and it is automated verified here. In Figure 4 we show the corresponding CWB-NC TRUE answer.

```
cwb-nc> eq SpecT01s1 ImpT01s1
Building automaton...
States: 8
Transitions: 10
Done building automaton.
Transforming automaton...
Done transforming automaton.
TRUE
Execution time (user, system, gc, real) : (0.037, 0.000, 0.000, 0.037)
cwb-nc>
```

Figure 4: The CWB-NC answer for SpecT01s1 $\sim$ ImpT01s1

## 3 Conclusions

Using the platforms like CWB-NC is still a reliable approach, following the research interest revealed by the consistent publications like [1] or [14], even in connection with CCS, SCCS and other modelling and verification tools. Despite these references, unfortunately, our experience with bigger models proves that the execution time achieved for some verifications is not convenient. That is why we consider as one of our future work directions the possibility of moving on from this combination based on SCCS/CWB-NC to a more modern opportunity based on functional programming. At this moment, such an interesting solution could follow the alternative of the CHP library [4], namely *Communicating Haskell Processes* - as a set of `Haskell` packages for implementing the concurrency ideas from Hoare's CSP [6] instead of Milner's CCS [8].

Considering `Haskell` as a very active functional programming language, we remark also the research interest for functional-based modelling of hardware components behaviour, especially of the synchronous digital circuits in [5].

Moving forward from an equation-based algebraic modelling approach to a `Haskell`-based functional one means to exploit the most valuable Haskell features like lazy evaluation, pattern matching or manipulating the high level functions. From the scientific point of view, these features are the basic guarantees for a substantial improvement of the previous execution time obtained in our prototype for the automatic verification of the agents bisimilarity equivalences.

## References

[1] Aceto, L., Ingolfsdottir, A., Larsen, K. and Srba, J., *Reactive Systems: Modelling, Specification and Verification*, Cambridge University Press, 2007.

[2] Almeida, A. A., Llanos, C. H., Arias-García, J. and Ayala-Rincón, M., *Verification of Hardware Implementations through Correctness of their Recursive*

*Definitions in PVS*, Proceedings of the 27th Symposium on Integrated Circuits and Systems Design, SBCCI '14, ACM, New York, USA, Article 14, 8 pages, 2014.

[3] Avigad, J. and Harrison, J., *Formally verified mathematics*, Commun. ACM **57, 4** (2014), 66-75.

[4] Brown, N. C. C, *Communicating Haskell Processes*, PhD Thesis, The University of Kent, Computer Science subject, UK, May 2011.

[5] Gammie, G., *Synchronous digital circuits as functional programs*, ACM Comput. Surv. **46**, Article 21 (2013), no. 2, 27 pages, 2013.

[6] Hoare, C. A. R., *Communicating sequential processes*, Prentice-Hall, 1985.

[7] Kumar, V. and Mishra, N., *Flip-flop and its applications*, Intl. J. of Innovative Research in Technology, **1** (2014), no. 5, 621-625.

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

[9] Riccobene, E. and Scandurra, P., *A formal framework for service modeling and prototyping*, Form. Asp. Comput. **26** (2014), no. 6, 1077-1113.

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

[11] Vasilescu, A., *Algebraic model for the behaviour of a D-flip-flops-based memory component*, chapter in book *Mathematical Methods, Computational Techniques, Intelligent Systems*, Proc. of the 12th WSEAS Intl Conf MAMECTIS'10, El Kantaoui, Sousse, Tunisia, May 3-6 2010, 42-47, 2010.

[12] Vasilescu, A. and Georgescu, O., *Algebraic Model for the Counter Register Behaviour*, IJCCC - Supplem. Issue as Proc. of ICCCC2006, Oradea, Romania, Vol. **I**, (2006), 459–464.

[13] Vasilescu, A., *Algebraic model for the JK flip-flop behaviour*, Proc. of SEEFM05 2nd South-East European Workshop on Formal Methods, Ohrid, FYROM, 209-223, 2005.

[14] Zhang, D., Cleaveland, R. and Stark, E. W., *The Integrated CWB-NC/PIOATool for Functional Verification and Performance Analysis of Concurrent Systems. Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science Volume **2619**, (2003), 431-436.

[15] CWB *** The CWB-NC homepage on http://www.cs.sunysb.edu/~cwb.