Counter register. Algebraic model and applications

ANCA VASILESCU
Transilvania University of Braşov
Department of Theoretical Computer Science
Str. Iuliu Maniu nr.50
ROMANIA
vasilex@unitbv.ro

Abstract: Hardware system consists of interconnected components, meaning communicating and synchronized components. Since the number of interconnected components in a computer system is continuously increasing, it follows that it is useful to have an alternative solution for verifying the computer operation instead of a simulation-based verification. In this paper we consider a specific component of the modern computer systems, namely a counter register, and we propose an algebraic approach as a solution for modelling and verifying the specification agents. As a final part of the paper, we mention some practical applications of the counter register, both in the everyday life and for the internal structure of the computer systems.

Key–Words: communication, counting operation, hardware system, modelling, SCCS process algebra, synchronization, verification

1 Introduction

A modern approach for analyzing the computer organization is based on the study of internal structure of interconnected components [1]. Hardware system consists of interconnected components. In general, computer systems are synchronous hardware structures so that internal components are synchronized to the pulses of the CPU global clock. Since the number of interconnected components in a computer system is continuously increasing and the distributed systems are widely used, it follows that the sub-systems level of independence is also increasing while the level of synchronization is decreasing. Hence, for this kind of modern systems, the communication protocols become very important for coordinating and unifying the internal cooperative sub-systems. In order to model these protocols, the hardware components are represented as agents involved in specific communications. That is why the SCCS-based agents represent a quite appropriate approach for analyzing the specific relations between the hardware components.

The final outcome of this paper consists in developing an algebraic model of communicating and synchronized hardware components represented at the digital logic level. The content of the paper is based on the following approach: starting from the specific hardware components operation, we have obtained the appropriate theoretical modelling expressions, the SCCS description files and the CWB-NC automated verification.

In our work (see e.g. [11], [12]) we have selected to be modelled those examples and digital circuits which are representative for the computer architecture structure and which are appropriate for developing a scalable, open hardware components hierarchy. Another researchers are also interested in modelling the hardware components behaviour, especially the registers as main memory structures [2], [8] in order to have an alternative solution for verifying the computer operation instead of a simulation-based verification.

There were at least two kind of problems we successfully passed during the modelling process: in [11], how to model the logical values (re)transmitted over the feed-back wires of the sequential circuits and in [12] and here, how to model the communication between the same type of components or between the different type of components.

Sometimes, in order to model the behaviour of a certain sequential circuit it is essential to make the difference between a stable, final state and an intermediate state. For this reason, we have to be able to identify the state changes and, moreover, to follow the circuit operation in time. In this paper we consider as prerequisites our results from [12] and [13]. We start with those models and we modify them in order to focus on the final states. It follows that we use these new agents for modelling the communication between distinct hardware components.

In order to obtain our models, we use the Milner's process algebra SCCS, a synchronous calculus derived from CCS (*Calculus of Communicating Systems*) [7]. We combine the process algebra and the

automata theory by using a *Concurrency WorkBench* platform, namely CWB-NC [17] for automatic verification of the target 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 target system behaviour.

In terms of practical dimension, the main idea of this paper is to use the algebra-based calculus SCCS for a specific class of practical logic circuits. Therefore, this work can be viewed as a result of formal methods applied to the modelling of hardware components of computers. Because SCCS could scale up easily, starting with the model of the flip-flops obtained in [11] we may continue with specifying the behaviour of any memory component, especially registers, and we may finally obtain an algebraic-based model for the computer system behaviour, each hardware component being modelled as an agent. Having an algebraic model like this, we have the main advantage of automatic verification of the specifications of the computer system behaviour for precisely inputs and at certain clock signals.

This paper is structured as follows. Section 2 presents the preliminaries about sequential circuits (flip-flops, registers and counter register) and the algebraic SCCS language. Based on these notions, in Section 3 we could develop the target algebraic model for the counter register two-step operation with final states. The most important part of the paper consists in defining the correct specification agents for the first step, the changing step and the second step of operation, but the core is the model for the changing-step transition. As a practical dimension, we consider in Section 4 some specific applications for the counter register as a basic component of digital computers. We have chosen the appropriate applications both in order to prove the usefulness of the counting operation in the everyday life and to mention the near future direction for our work: to develop an algebraic model for the entire computer operation. The final section is providing some conclusions and open problems.

2 Preliminaries

2.1 Flip-flops. Registers. Counter register

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. The difference among various types of flip-flops is the number of inputs and the manner in which the inputs affect the binary state. The most common types of flip-flops are: SR flip-flop, D flip-flop, JK flip-flop and T flip-flop.

For the interest of this paper we note that, by definition, a T (*Toggle*) flip-flop has one input, namely T and it operates as follows: 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.

A *register* is a group of flip-flops with each flip-flop capable of storing one bit of information. In this paper we refer to a *two-bit register* that has a group of two T flip-flops and consequently it is capable of storing any binary information of two bits. In addition to the flip-flops, a register may have combinational gates that perform certain data-processing tasks. The flip-flops hold the binary information and the gates control when and how new information is transferred into the register.

A register that goes through a predetermined sequence of states upon the application of the clock pulses is called a *counter register*. Counters are found in almost all equipment containing digital logic. They are used for counting the number of occurrences of an event and they are useful for generating timing signals to control the sequence of operations in digital computers. A *two-bit binary counter* follows a sequence of states according to the binary count of two bits, from 0 to 2^2-1. The design of binary counters can be carried out from a direct inspection of the sequence of states that the register must undergo to achieve a straight binary count or a reverse binary count. *Synchronous binary counters* have a regular pattern, namely the clock inputs of all flip-flops receive the common clock.

2.2 Process algebra SCCS

The process algebra SCCS, namely *Synchronous Calculus of Communicating Systems* is derived from CCS [7] especially for achieving the synchronous interaction in the framework of modelling the concurrent communicating processes. Both in CCS and in SCCS, processes are built from a set of atomic actions A. Denoting the set of labels for these actions by Λ, a CCS 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 $\breve{a}$ or (3) an internal on $a \in \Lambda$ denoted by τ. 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 $a^{-n} = \overline{a}^n$ and that the action **1** in SCCS is the action τ from CCS and it is identified in SCCS with the empty product. 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$\backslash L$	restriction, $L \subseteq A \cup \overline{A}$
	| P[f]	relabelling with the morphism $f : A \cup \overline{A} \to A \cup \overline{A}$

Model 1: Syntax of a SCCS process

In this grammar, the restriction is inherited from CCS. There is also an SCCS specific restriction denoted by the $\restriction$ operator and structural related with the CCS operator by

$$\mathrm{P}\backslash L = \mathrm{P} \restriction E$$

where $E = (A\text{-}L)^*$ is the submonoid of A generated by the set difference A-L. By definition, the P $\restriction$ E agent is forced to execute only the actions from the set E as the external actions and the agent P$\backslash L$ is forced to not execute the actions from the set L, except as the internal 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.

3 Modelling two steps of operation with final states

In [12] we have considered a specific structure of a hardware component, namely a counter register based on a combination of interconnected digital components represented in Figure 1.

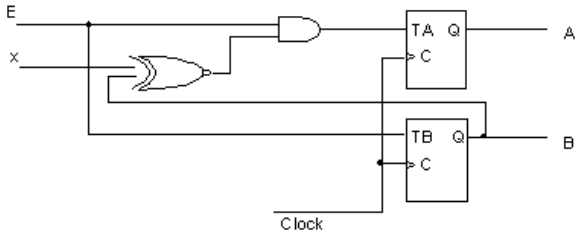

Figure 1: Two-bit synchronous binary counter

This circuit consists of an input combinational level and a sequential level, the latter based on two synchronized T flip-flops.

Remark 1 *The above counter register operates as follows. If the count enable E is 0, both the TA and TB inputs are maintained at 0 and the output of the counter does not change. When the counter is enabled and the clock goes through a positive transition, the operation of the counter depends on the value of the input x, as follows: while $x = 0$ the register operates like a counting-down counter and while $x = 1$ the register operates like a counting-up counter.*

In this model, the bit B is the less significant bit and the bit A is the most significant bit. Hence, the predeterminated sequence of states for the content AB of the register is 00 $\to$ 11 $\to$ 10 $\to$ 01 $\to$ 00, and so on, for the counting-down operation and it is 00 $\to$ 01 $\to$ 10 $\to$ 11 $\to$ 00, and so on, for the counting-up operation.

In this paper we are concerned about the modelling of two steps of operation, namely two state changes for the counter register starting from the initial state 00. For representing this sequential behaviour, the involved agents have to simulate one step of operation starting from the initial state 00 followed by another one step of operation with initial state identical with the final state of the first step. For this reason, it is important to assure that the agents are emphasizing the final state of the flip-flops, so that the rendez-vous (data exchange and synchronization) compulsory takes place between the first and the second step, meaning after the register has achieved the final (stable) state of the first step.

We consider as prerequisites of this work our results from [12] concerning the agent SpecCR(A, B, c) for specifying one step of operation of the counter register. For the interest of this paper, we modify the expression of the agent SpecCR(A, B, c) so that the resulted one-step agent SpecCRfs(A, B, c) has to be able to execute a special, specific action, denoted by $\overline{endCountStep}$ only if the register has achieved the final state of the first step. Further, we use the agents SpecCRfs(A, B, c) for defining two factor agents, namely Step1(A, B) and Step2(A, B), and the final product agent

$$\begin{aligned}\text{SpecCRTwoSteps}(0,0,A^*,B^*) = \\ = (\text{Step1}(0,0) \times \text{Step2}(A^*,B^*))\backslash \text{ComMeet}_{AB}\end{aligned} \quad (1)$$

for specific, correct binary combinations of values A^*B^* according to the possible changes of the counter register content starting from the initial state $(0, 0)$ and for a specific set ComMeet$_{AB}$ of external communicating actions.

3.1 Modelling the first step

In order to model the first step of the counter register operation with final states, we shall define the appropriate agents for specifying both the entrance combinational level (consisting of basic logic gates) and the sequential level (consisting of two synchronized T flip-flops).

Let DELTA be a delaying agent defined by

$$\text{DELTA} = \mathbf{1} \ : \ \text{DELTA} \tag{2}$$

As we have seen in the previous section 2.2 the action **1** is defined by the SCCS language as the delaying action. It follows that the previous agent DELTA can only delay (it is an *idle* agent).

Let also define the specification for the combinational entrance level by the agents

$$\begin{aligned}&\text{SpecIn}(A,B) = \\ &\quad = \sum_{E,x\in\{0,1\}} \epsilon_E \zeta_x \alpha_A \beta_B \overline{andout}_p \overline{Ed}_q : \text{DELTA}\end{aligned} \tag{3}$$

where p = E AND (B NXOR x) and q = E.

Using these definitions, for each pair $(A,B) \in \{(0,0),(0,1),(1,0),(1,1)\}$ the specification of the combinational level behaviour is given by the agents

$$\text{SpecCLC}(A,B) = \text{SpecIn}(A,B)[\Phi CLC] \tag{4}$$

where the morphism ΦCLC is defined by the relabelling pairs: $andout_p \mapsto comTAIn_pTA_p$ and $Ed_q \mapsto comTBIn_qTB_q$.

For the specific case of $A=0$ and $B=0$, the CWB-NC automaton assigned to the agent SpecCLC$(0,0)$ is represented in Model 2. It has two states, namely 0: and 1:, and five transitions.

```
0:'E0.x0.A0.B0.∼comTAIn0.∼TA0.∼comTBIn0.∼TB0' {1}
  'E0.A0.B0.x1.∼comTAIn0.∼TA0.∼comTBIn0.∼TB0' {1}
  'x0.A0.B0.E1.∼comTAIn1.∼TA1.∼comTBIn1.∼TB1' {1}
  'A0.B0.x1.E1.∼comTAIn0.∼TA0.∼comTBIn1.∼TB1' {1}
1:'' {1}
```

Model 2: The SpecCLC(0,0) CWB-NC automaton

As we have already proved in [11] and [13] the T flip-flop operation with final states might be specified by the agents:

$$\text{InTfs}(T) = \xi_T \overline{\alpha}_J \overline{\beta}_K : \text{InTfs}(T) + \overline{endInT}\ \text{InTfs}(T) \tag{5}$$

where the evaluations are J = T and K = T and the agents:

$$\begin{aligned}&\text{SpecTfs}(m,n,c) = \\ &\quad = \sum_{T\in\{0,1\}} (\text{InTfs}(T)[endInT/\overline{final}] \times \\ &\quad \times \text{SpecJKfs}(m,n,c)[endJK_{mn}/final\ endT_{mn}])\backslash \\ &\quad \backslash\{\alpha,\beta,final\}\end{aligned} \tag{6}$$

for $(m,n) \in \{(0,1),(1,0)\}$.

For each $i \in \{0,1\}$ we define the set Restr_TIn$_i$ = $\{CLK, \xi_i, \gamma, \delta, endT\}$ of restricted actions involved in the next specification agents for the two T flip-flops behaviour with final states. Here, the first agent SpecT(A,c) represents the T flip-flop associated with A – the most significant bit of the counter register and the second agent SpecT(B,c) represents the T flip-flop associated with B – the less significant bit of the counter register.

$$\begin{aligned}&\text{SpecT}(A,c) = \\ &\quad = (\sum_{i\in\{0,1\}} comTAIn_i(\text{SpecTfs}(\text{NOT } A, A, c) \upharpoonright \\ &\quad \upharpoonright \text{Restr\_TIn}_i))[\Phi A_c]\end{aligned} \tag{7}$$

and

$$\begin{aligned}&\text{SpecT}(B,c) = \\ &\quad = (\sum_{i\in\{0,1\}} comTBIn_i(\text{SpecTfs}(\text{NOT } B, B, c) \upharpoonright \\ &\quad \upharpoonright \text{Restr\_TIn}_i))[\Phi B_c]\end{aligned} \tag{8}$$

where the morphism ΦA_c is defined by the relabelling pairs $\xi_T \mapsto TA_T$, $CLK_c \mapsto CLKA_c$, $\gamma_m \mapsto nil$, $\delta_n \mapsto \alpha_n$ and $endT_{mn} \mapsto \overline{\alpha}_n New\alpha_n endTA$ and the morphism ΦB_c is defined by the relabelling pairs $\xi_T \mapsto TB_T$, $CLK_c \mapsto CLKB_c$, $\gamma_m \mapsto nil$, $\delta_n \mapsto \alpha_n$ and $endT_{mn} \mapsto \overline{\beta}_n New\beta_n endTB$.

We are now able to give the specification for the behaviour of the counter register sequential level by the agents:

$$\text{SpecCLS\_T}(A,B,c) = \text{SpecT}(A,c) \times \text{SpecT}(B,c) \tag{9}$$

for each current binary content $(A,B) \in \{(0,0),(0,1),(1,0),(1,1)\}$ of the register and the clock signal $c \in \{0,1\}$.

For the specific case of $A=0$, $B=0$ and $c=1$, the CWB-NC automaton assigned to the agent SpecCLCS_T$(0,0,1)$ is represented in Model 3.

In order to obtain the announced one-step agent SpecCRfs(A,B,c) we have to make these two level agents SpecCLC(A,B) and SpecCLS_T(A,B,c) to communicate and synchronize in an appropriate manner following the definition:

$$\begin{aligned}&\text{SpecCRfs}(A,B,c) = \\ &\quad = (\text{SpecCLC}(A,B) \times \text{SpecCLS\_T}(A,B,c)[\Psi])\backslash \\ &\quad \backslash\{comTAIn, comTBIn\}\end{aligned} \tag{10}$$

where the morphism Ψ is defined by the relabelling pairs $endTA \mapsto halfStable$ and $endTB \mapsto \overline{halfStable}\ endCountStep$.

Proposition 2 *For each current binary content* $(A,B) \in \{(0,0),(0,1),(1,0),(1,1)\}$ *of the register and*

0: 'comTAIn0.comTBIn0' {1}
'comTAIn0.comTBIn1' {2}
'comTAIn1.comTBIn0' {3}
'comTAIn1.comTBIn1' {4}
1: '∼A0.∼B0.TA0.TB0.CLKA1.CLKB1' {5}
2: '∼A0.∼B0.TA0.TB1.CLKA1.CLKB1' {6}
3: '∼A0.∼B0.TA1.TB0.CLKA1.CLKB1' {7}
4: '∼A0.∼B0.TA1.TB1.CLKA1.CLKB1' {8}
5: '∼endTA.∼NewA0.∼endTB.∼NewB0' {5}
6: '∼B0.TB1.∼endTA.∼NewA0.CLKB1' {9}
7: '∼A0.TA1.∼endTB.∼NewB0.CLKA1' {10}
8: '∼A0.∼B0.TA1.TB1.CLKA1.CLKB1' {11}
9: '∼B1.TB1.∼endTA.∼NewA0.CLKB1' {12}
10: '∼A1.TA1.∼endTB.∼NewB0.CLKA1 {13}
11: '∼B1.∼A1.TA1.TB1.CLKA1.CLKB1' {14}
12: '∼endTA.∼NewA0.∼NewB1.∼endTB' {12}
13: '∼NewA1.∼endTA.∼endTB.∼NewB0' {13}
14: '∼NewA1.∼endTA.∼NewB1.∼endTB' {14}

Model 3: The SpecCLS_T(0,0,1) CWB-NC automaton

the clock signal $c \in \{0,1\}$*, the agent SpecCRfs*(A,B,c) *defined in (10) specifies one step of counter register operation.*

Proof: Considering all the previous defined agents involved in the SpecCRfs(A,B,c) agent expression, we have that the product agent SpecCRfs(A,B,c) assures that the agents SpecCLC(A,B) and SpecCLS_T(A,B,c) meet each other on the *channels* $comTAIn$ or $comTBIn$. It yields that the agent SpecCRfs(A,B,c) specifies one step of counter register operation. □

As any other sequential circuit, this counter register operation is synchronized with the clock signal transmitted on the $Clock$ physical wire. That is why, as a practical dimension of this work, we are interested in the case of $c = 1$ and the natural initial state given by $A = 0$ and $B = 0$. These certain values provides the specific agent SpecCRfs$(0,0,1)$.

Following the previous theoretical expressions (10), we have developed the SCCS models for all of the involved agents and we have used the CWB-NC platform for an automated verification of the proposed models. Out of these results, we present in Model 4 the CWB-NC finite state automaton assigned to the agent SpecCRfs$(0,0,1)$.

Proposition 3 *The finite state automaton represented in Model 4 recognizes one step of counter register operation starting from the initial state 00.*

Proof: The diagrammatic representation of this automaton is illustrated in Figure 2. We mention that: the squared numbers represent the states, the nodes

0: 'E0.x0.A0.B0.∼TA0.∼TB0'{1}
'E0.A0.B0.x1.∼TA0.∼TB0'{1}
'x0.A0.B0.E1.∼TA1.∼TB1'{2}
'A0.B0.x1.E1.∼TA0.∼TB1'{3}
1: '∼A0.∼B0.TA0.TB0.CLKA1.CLKB1' {4}
2: '∼A0.∼B0.TA1.TB1.CLKA1.CLKB1' {5}
3: '∼A0.∼B0.TA0.TB1.CLKA1.CLKB1' {6}
4: '∼NewA0.∼NewB0.∼endCountStep' {4}
5: '∼A0.∼B0.TA1.TB1.CLKA1.CLKB1' {7}
6: '∼B0.TB1.∼NewA0.CLKB1.∼halfStable' {8}
7: '∼B1.∼A1.TA1.TB1.CLKA1.CLKB1' {9}
8: '∼B1.TB1.∼NewA0.CLKB1.∼halfStable' {10}
9: '∼NewA1.∼NewB1.∼endCountStep' {9}
10: '∼NewA0.∼NewB1.∼endCountStep' {10}

Model 4: The SpecCRfs(0, 0, 1) CWB-NC automaton

labels represent the current configuration of the register content AB, the first level transitions labels represent the entrance pairs of (E,x) values and the other arcs labels represent the pairs of entrance values (TA, TB) of the flip-flops. The initial state is 0: and the final states are double-circle marked.

Based on this automaton, if the initial register content AB is 00 then the final content AB might be: 00, 01 or 11. Considering the corresponding transitions labels, one may easily conclude that this automaton represents the counter register operation as we have defined in the previous Remark 1. It means that after one step of operation, the current state of the register is: 00 - if it operated with no state change, 01 - if it counted-up or 11 - if it counted-down. □

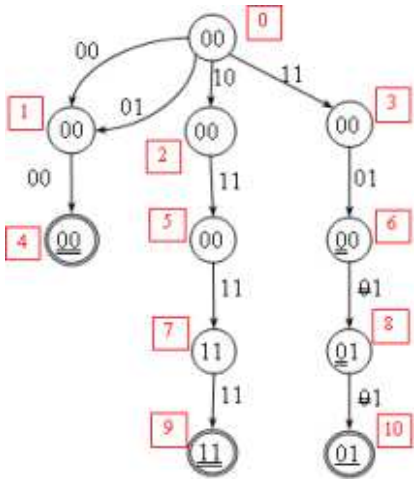

Figure 2: The SpecCRfs(0, 0, 1) diagrammatic representation

(L0) ALGORITHM
(L1) InitialA ← 0; InitialB ← 0
(L2) // Step1
(L3) do SpecCRfs (InitialA, InitialB, 1)
(L4) while NOT FinalState
(L5) NewA ← FinalStateA; NewB ← FinalStateB
(L6) // Step2
(L7) do SpecCRfs (NewA, NewB, 1)
(L8) while NOT FinalState
(L9) write (FinalStateA, FinalStateB)
(L10) END

Model 5: Step-transition algorithm

3.2 Modelling the changing-step transition

The counter register operation in two steps is following the next algorithm:

The translation of this algorithm into the SCCS language was not at all an easy job considering the difficulty of modelling the state change. In order to obtain the SCCS representation of this algorithm, we have defined specific agents for modelling the sequential operation as follows: on the one hand these agents have to assume the final state of the first step and, on the other hand, they have to guide the second step starting from the final state of the first step.

Let Meet(A, B) be some internal agents and let EndStep1 be the agent which is capable of operating in two manners: either it is delaying while the first step is not finished or, when the first step is finished, it outputs the final state ($NewA$, $NewB$) by performing the specific action $\overline{end}$. The definitions of these agents are:

$$\text{Meet}(A,B) = \quad (11)$$
$$= New\alpha_A New\beta_B\, endCountStep : \text{Meet}(A,B)$$

and

$$\text{EndStep1} = \mathbf{1} : \text{EndStep1} + \quad (12)$$
$$+ \sum_{A,B\in\{0,1\}} (New\alpha_A New\beta_B\, endCountStep\, \overline{end}_{AB} :$$
$$: \text{Meet}(A,B))$$

In this way, the first step specification model consists in the product of the agents SpecCRfs and EndStep1, for each initial configuration $(A, B) \in \{(0,0),(0,1),(1,0),(1,1)\}$. The expression is:

$$\text{Step1}(A,B) = \quad (13)$$
$$= (\text{SpecCRfs}(A,B,1) \times \text{EndStep1})\backslash \text{ComStep1}$$

where the set of communicating actions is ComStep1 = $\{New\alpha_1\, , New\beta\, , endCountStep\}$.

We present in Model 6 the CWB-NC finite state automaton assigned to the agent Step1$(0,0)$.

0: ‘E0.x0.A0.B0.∼TA0.∼TB0’ {1}
‘E0.A0.B0.x1.∼TA0.∼TB0’ {1}
‘x0.A0.B0.E1.∼TA1.∼TB1’ {2}
‘A0.B0.x1.E1.∼TA0.∼TB1’ {3}
1: ‘∼A0.∼B0.TA0.TB0.CLKA1.CLKB1’ {4}
2: ‘∼A0.∼B0.TA1.TB1.CLKA1.CLKB1’ {5}
3: ‘∼A0.∼B0.TA0.TB1.CLKA1.CLKB1’ {6}
4: ‘∼end00’ {7}
5: ‘∼A0.∼B0.TA1.TB1.CLKA1.CLKB1’ {8}
6: ‘∼B0.TB1.∼NewA0.CLKB1.∼halfStable’ {9}
7: ‘’ {7}
8: ‘∼B1.∼A1.TA1.TB1.CLKA1.CLKB1’ {10}
9: ‘∼B1.TB1.∼NewA0.CLKB1.∼halfStable’ {11}
10: ‘∼end11’ {12}
11: ‘∼end01’ {13}
12: ‘’ {12}
13: ‘’ {13}

Model 6: The Step1(0,0) CWB-NC automaton

3.3 Modelling the second step

In order to obtain the second step specification, for the binary clock value $c = 1$ and for each current content $(A, B) \in \{(0,0),(0,1),(1,0),(1,1)\}$, we define the agent Step2 as follows: it is indefinitely performing the action end in order to achieve the synchronization with the agent Step1 and, after the *rendez-vous*, it operates exactly like the generic agent SpecCRfs. Its expression is:

$$\text{Step2}(A,B) = \mathbf{1} : \text{Step2}(A,B) + \quad (14)$$
$$+ end_{AB}\overline{endFirstStep}_{AB} : \text{SpecCRfs}(A,B,1)$$
$$[endCountStep/endSecondStep]$$

For example, we present in Model 7 the CWB-NC automaton assigned to the agent Step2$(1,1)$.

0: ‘’ {0}
‘end11. endFirstStepAB11’ {1}
1: ‘E0.x0.B1.A1.∼TA0.∼TB0’ {2}
‘E0.x1.B1.A1.∼TA0.∼TB0’ {2}
‘x0.E1.B1.A1.∼TA0.∼TB1’ {3}
‘x1.E1.B1.A1.∼TA1.∼TB1’ {4}
2: ‘∼B1.∼A1.TA0.TB0.CLKA1.CLKB1’ {5}
3: ‘∼B1.∼A1.TA0.TB1.CLKA1.CLKB1’ {6}
4: ‘∼B1.∼A1.TA1.TB1.CLKA1.CLKB1’ {7}
5: ‘∼NewA1.∼NewB1.∼endSecondStep’ {5}
6: ‘∼B0.TB1.∼NewA1.CLKB1.∼halfStable’ {8}
7: ‘∼A0.∼B0.TA1.TB1.CLKA1.CLKB1’ {9}
8: ‘∼B0.TB1.∼NewA1.CLKB1.∼halfStable’ {10}
9: ‘∼A0.∼B0.TA1.TB1.CLKA1.CLKB1’ {11}
10: ‘∼NewA1.∼NewB0.∼endSecondStep’ {10}
11: ‘∼NewA0.∼NewB0.∼endSecondStep’ {11}

Model 7: The Step2(1,1) CWB-NC automaton

We are now able to specify all the counter register operations in two steps starting from a convenient

initial state and emphasizing the step change on a certain configuration. These specifications use the previous agents Step1 and Step2 for specific combinations of parameters which represent the right chosen states of the register. We select as an appropriate example the elementary two-steps operation starting from the initial state $(0,0)$ and changing the step on the infered state $(A^*,B^*) \in \{(0,0),(0,1),(1,1)\}$ provided by the possible final states of the first step represented in Figure 2. This operation is specified by the agent:

$$\begin{aligned}&\text{SpecCRTwoSteps}(0,0,A^*,B^*) = \\ &\quad = ((\text{Step1}(0,0) \times \text{Step2}(A^*,B^*)) \backslash \{end_{A^*B^*}\}) \\ &\quad [\Phi_{A^*B^*}]\end{aligned} \tag{15}$$

where the morphism $\Phi_{A^*B^*}$ is defined by the relabelling pairs $end_{ij} \mapsto endFirstStep_{ij}$, for each combination $(i,j) \in \{(0,0),(0,1),(1,0),(1,1)\} \backslash \{(A^*,B^*)\}$.

For all of these agents and all of the possible input binary combinations, we have developed the SCCS models and we have used the CWB-NC platform for an automated verification of the proposed models. Particularly, for $A^*=1$ and $B^*=1$, the CWB-NC finite state automaton assigned to the target agent SpecCRTwoSteps$(0,0,1,1)$ is represented in Model 8.

```
0:  'E0.x0.A0.B0.~TA0.~TB0'    {1}
    'E0.A0.B0.x1.~TA0.~TB0'    {1}
    'x0.A0.B0.E1.~TA1.~TB1'    {2}
    'A0.B0.x1.E1.~TA0.~TB1'    {3}
1:  '~A0.~B0.TA0.TB0.CLKA1.CLKB1'      {4}
2:  '~A0.~B0.TA1.TB1.CLKA1.CLKB1'      {5}
3:  '~A0.~B0.TA0.TB1.CLKA1.CLKB1'      {6}
4:  '~endFirstStepAB00'            {7}
5:  '~A0.~B0.TA1.TB1.CLKA1.CLKB1'      {8}
6:  '~B0.TB1.~NewA0.CLKB1.~halfStable' {9}
7:  ''      {7}
8:  '~B1.~A1.TA1.TB1.CLKA1.CLKB1'      {10}
9:  '~B1.TB1.~NewA0.CLKB1.~halfStable' {11}
10: '~endFirstStepAB11'            {12}
11: '~endFirstStepAB01'            {13}
12: 'E0.x0.B1.A1.~TA0.~TB0'    {14}
    'E0.x1.B1.A1.~TA0.~TB0'    {14}
    'x0.E1.B1.A1.~TA0.~TB1'    {15}
    'x1.E1.B1.A1.~TA1.~TB1'    {16}
13: ''      {13}
14: '~B1.~A1.TA0.TB0.CLKA1.CLKB1'      {17}
15: '~B1.~A1.TA0.TB1.CLKA1.CLKB1'      {18}
16: '~B1.~A1.TA1.TB1.CLKA1.CLKB1'      {19}
17: '~NewA1.~NewB1.~endSecondStep'         {17}
18: '~B0.TB1.~NewA1.CLKB1.~halfStable' {20}
19: '~A0.~B0.TA1.TB1.CLKA1.CLKB1'      {21}
20: '~B0.TB1.~NewA1.CLKB1.~halfStable' {22}
21: '~A0.~B0.TA1.TB1.CLKA1.CLKB1'      {23}
22: '~NewA1.~NewB0.~endSecondStep'         {22}
23: '~NewA0.~NewB0.~endSecondStep'         {23}
```

Model 8: The SpecCRTwoSteps(0,0,1,1) CWB-NC automaton

Proposition 4 *The finite state automaton represented in Model 8 recognizes two steps of counter register operation starting from the initial state 00 and changing the step on the state 11.*

Proof: Following the previous expressions (15), the generic agent SpecCRTwoSteps $(0,0,A^*,B^*)$ is defined as a combination of morphism, product and restriction. This combination is successfully used to model the channeling of data, here along the hardware wires.

We can split the automaton definition into three parts: (1) a first part which respects the behaviour of the automaton described in Model 6 for the first step of operation, (2) the changing-step transition from state 10: to state 12: and (3) a third part which respects the behaviour of the automaton from the Model 7 for the second step of operation. Even if the first step of operation is finishing either with the action $\sim endFirstStepAB00$, or $\sim endFirstStepAB11$, or $\sim endFirstStepAB01$, only the action $\sim endFirstStepAB11$ starts a new step of operation (see the changing-step transition) while the other two actions guide the automaton to achieve a final state by indefinitely performing the empty action. In its final states, meaning the states 17:, 22: and 23:, the automaton indefinitely performs the specific actions $NewA$ and $NewB$ for the final configuration of the register content after two steps of operation. □

4 Specific applications

In this section we will consider two different applications of a counter register. The first application arises from a minimal components adding into the internal structure of the previous counter register, with useful implications for the utility of the arised structure. The second application represents an extension of the counter register, namely we consider a new sequential hardware component in which the counter register is integrated as a module.

4.1 Producer-consumer buffer

We start from the internal structure of the counter register already considered in Figure 1. We add an extra sequential component, namely a D flip-flop synchronized with the two T flip-flops on the same clock signal like in the Figure 3 [5].

Besides the output lines storing the binary content of the register, this circuit provides an extra output value, y which is given by the stored value in the D flip-flop. There is only one clock signal in the circuit and this follows that the three flip-flops are entirely synchronized. Because the clock entrance for the D flip-flop is provided by an AND gate between E and

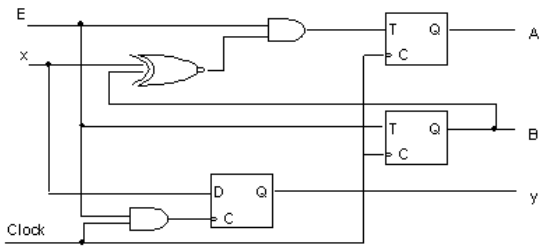

Figure 3: Register as producer-consumer buffer

Clock, it yields that the value of input x is stored on the output y only when E=1, and otherwise the value on the y output line is not modified (practically, the D flip-flop does not operate).

Here, we are interested in a practical interpretation of this circuit operation, namely in connecting the counter variable values stored by the internal counter register with the word Y of succesive symbols sent on the output line y of the circuit.

In [5] this register behaviour is modelled using different approaches, including an algebraic model and a transition model based on a FSM (Finite State Machine). The automatum writes a 0 symbol on the output y as the result of each clock-controled occurrence of the input combination E=1 and x=0 and it writes an 1 symbol as the result of each occurrence of the combination E=1 and x=1. Moreover, these two events have opposing significance: the occurrence of an event 11 corresponding for E=1 and x=1 is increasing the counter variable value and an event 10 corresponding for E=1 and x=0 is decreasing it. It follows that the length of the word Y is depending on the numbers of occurrences of the events 10 and 11. Formally, the language $\mathcal{L}$ = L(FSM) generated by the FSM machine is

$$\mathcal{L} = \{Y \in \{0,1,\lambda\}^* | N_0(Y) + N_1(Y) = |Y|\}.$$

where λ represents the empty string. The output symbol λ is corresponding to E=0. The length condition for the words Y has to assure that each occurrence of a counted event adds a symbol on the output word Y.

We consider now the classical example of the producer-consumer buffer. Let n be the number of items in the buffer at the given initial time t_0=0. Consider also the recursive function $f_t : \mathcal{L} \rightarrow$ Z with $f_0(\lambda)$ = n, $f_1(0)$ = n-1, $f_1(1)$ = n+1 and for each succesor x of the current word Y the function definition is

$$f_{t+1}(Yx) = \begin{cases} f_t(Y) - 1, & \text{if } x = 0 \\ f_t(Y), & \text{if } x = \lambda \\ f_t(Y) + 1, & \text{if } x = 1. \end{cases} \quad (16)$$

Proposition 5 *The number of items in the buffer at a given time t is depending on the output word Y with respect to the relation $f_t(Y)$ = n + $N_1(Y)$ - $N_0(Y)$.*

Proof: We shall prove this relation by induction in t, the counting variable.
For t = 0 we have $\mathcal{L} = \{\lambda\}$ and $f_0(\lambda)$ = n by definition. The right side of the required relation is n + $N_1(\lambda)$ - $N_0(\lambda)$ = n + 0 - 0 = n = $f_0(\lambda)$.
Suppose f_t(Y) = n + $N_1(Y)$ - $N_0(Y)$ and we prove that $f_{t+1}(Yx)$ = n + $N_1(Yx)$ - $N_0(Yx)$ for each $x \in \{0,1\}$.
For x = 0 we have $f_{t+1}(Y0)$ = $f_t(Y)$ - 1 = n + $N_1(Y)$ - $N_0(Y)$ - 1 = n + $N_1(Y)$ - ($N_0(Y)$ + 1) = n + $N_1(Yx)$ - $N_0(Yx)$, as required.
For x = 1 we have $f_{t+1}(Y1)$ = $f_t(Y)$ + 1 = n + $N_1(Y)$ - $N_0(Y)$ +1 = n + ($N_1(Y)$ + 1) - $N_0(Y)$ = n + $N_1(Yx)$ - $N_0(Yx)$, as required. □

In accordance with this relation, the occurrence of the event E=1 and x=1 might be recognized as a *producer* event and the occurrence of the event E=1 and x=0 might be recognized as a *consumer* event.

4.2 Serial-parallel converter

In this section we consider a specific sequential circuit in order to integrate the two-bit counter register structure. Figure 4 represents this circuit.

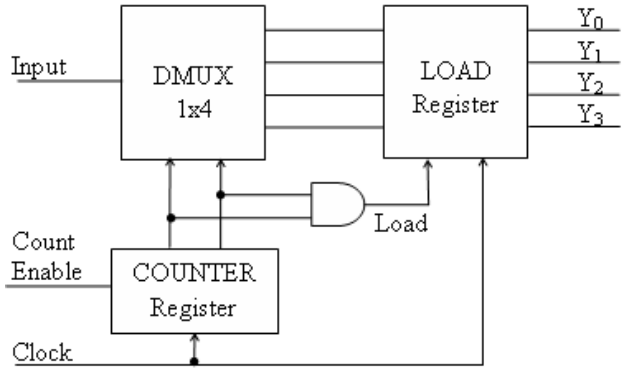

Figure 4: Serial-parallel Converter

This circuit consists of one demultiplexer 1x4 DMUX, one four-bit LOAD Register and the announced two-bit COUNTER Register. The circuit is entirely synchronized because all the sequential components are using the same clock signal.

The circuit operates as follows. We constantly keep 1 logic value on the Count Enable input line so that the COUNTER Register operates continuously. The current value stored by the counter register represents the selecting input for the demultiplexer so that the demultiplexer successively outputs the value from the Input line to the LOAD register. An important part

of the circuit operation consists in the role of the AND gate. This gate transfers 1 logic value to the Load enable input of the LOAD Register in only one configuration, namely the value 11 stored by the counter register and it transfers 0 logic value for all the other configurations. It follows that after each counting-up cycle (from 00 to 11) and only in this *final* configuration 11, the LOAD Register effectively loads on the output lines of the circuit its input values, namely the input values successively sent by the demultiplexer. For all the others configurations, the AND gate sends 0 logic value and the LOAD Register doesn't operates, only gathers the values on its input lines.

Hence, while the COUNTER Register counts, we successively charge specific input values on the entrance Input line of the circuit and the circuit outputs on the Y_0, Y_1, Y_2, Y_3 lines the entrance values, not successively but all together. That means that this circuits operates as a serial-parallel converter.

From the algebraic modelling point of view, we already have the algebraic models for the demultiplexer [10] and the counter register and it follows that one possible direction of our future work is to model the load register behaviour and after that the complete circuit considered here. Having these algebraic models for the hardware components behaviour we may put all of them together in order to obtain a global algebraic model for the entire computer operation.

5 Conclusions

From the theoretical point of view, this paper uses the algebraic theory of processes as a formal method applied to model the behaviour of hardware components. Many other researchers are interested in this subject, for example: in [3] and [6] the authors use a computer-based approach for designing a specific class of circuits, in [9] the authors interest is for modelling even the asynchronous circuits, in [4] the interest is for modelling the general circuits behaviour in CCS, in [16] there is a model of the flip-flops behaviour in VHDL. Besides, [15] is a very recent book concerned about constructing a mathematical theory of modelling the asynchronous circuits behaviour. Beyond these natural models, in this paper we had to adjust a pure algebraic language like SCCS to model a specific type of hardware component, namely the counter register.

From the practical point of view, a counter register is used for counting the number of occurrences of a certain event. Based on an algebraic approach, this paper results refer to a counter register for two steps of operation, meaning the possibility of counting two occurrences of the event. It follows that our work will progress in the direction of modelling the global counter register operation in many steps. And, moreover, the next level of our modelling interest is to develop the corresponding implementation agents and to integrate these counter register models in more complex computer architecture structures.

The original contributions of the author in this paper consists both in modelling the counter register behaviour with final states and in modelling two steps of operation based on communication and synchronization.

References:

[1] E. Alaer, A. Tangel and M. Yakut, "MIB-16" FPGA Based Design and Implementation of a 16-Bit Microprocessor for Educational Use, *WSEAS Transactions on Advances in Engineering Education*, 5(5), May 2008, pp: 326–330.

[2] N.–G. Bardis, A.–P. Markovskyy and D.–V. Andrikou, Method for Designing Pseudorandom Binary Sequences Generators on Nonlinear Feedback Shift Register (NFSR), *WSEAS Transactions on Communications*, 3(2), April 2004, pp: 758–763.

[3] D. Batas and H. Fiedler, Computer-Based Design of Analog Integrated CMOS-Circuits, *Proc. of the 11th WSEAS International Conference on CIRCUITS, Ag. Nikolaos, Greece*, vol.1, 2007, pp: 31–36.

[4] G. Clark and G. Taylor, The Verification of Asynchronous Circuits using CCS, *Technical Report as ECS-LFCS-97-369*, 1997

[5] O. Georgescu, Problem Solving with Different Models, *Proc. of SEEFM05 2nd South-East European Workshop on Formal Methods, Ohrid, FYROM*, 2005, pp. 247–255.

[6] M. Kamran and S. Feng, Digital Circuit Design and Implementation with Efficient Task Partitioning Algorithm, *WSEAS Transactions on Circuits and Systems*, 5(4), April 2006, pp: 511–517.

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

[8] C.–M. Ou, W.–J. Hwang and M.–K. Chen, A Novel VLSI Architecture For Block Matching Using Shift-Register Based Memory Modules, *WSEAS Transactions on Circuits and Systems*, 3(9), November 2004, pp: 1876–1882.

[9] A. Razafindraibe, M. Robert and P. Maurine, Compact and secured primitives for the design of asynchronous circuits, *Journal of Low Power Electronics* Vol.1, 2005, pp: 20–26.

[10] A. Vasilescu, Recursive Rules For Demultiplexers Expanding, *Studia Universitatis Babeş-Bolyai, Informatica*, XLVI(1), 2001, pp. :67–74.

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

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

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

[14] A. Vasilescu, Algebraic Model for the Intercommunicating Hardware Components, *Proceedings of the* 12^{th} *WSEAS International Conference on Computers, Crete, Greece*, 2008, pp. 241–246.

[15] S.–E. Vlad, *Asynchronous Systems Theory*, WSEAS Press 2007

[16] R.–D. Wittig, OneChip: An FPGA Processor With Reconfigurable Logic, *IEEE Symposium on FPGAs for Custom Computing Machines*, 1995

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