Johanna Tuominen | Juha Plosila

Formal Energy Estimation Framework

Turku Centre *for* Computer Science

TUCS Technical Report
No 696, June 2005

Formal Energy Estimation Framework

Johanna Tuominen
Turku Centre for Computer Science
Lemminkäisenkatu 14 A, 20520 Turku, Finland
joeltu@utu.fi
Juha Plosila
University of Turku, Dept. of Information Technology
Lemminkäisenkatu 14-18 A, 20520 Turku, Finland
juplos@utu.fi

TUCS Technical Report
No 696, June 2005

Abstract

Conventionally, the correctness of functional and non-functional properties of hardware components is ensured during the design process by simulation. Furthermore, different description languages are needed during the design process. By adopting the Action Systems, we are able to use the same formalism from specification down to implementation. In this paper, we introduce a formal energy estimation framework for hardware components. Moreover, we demonstrate the formal energy estimation by using example system descriptions.

Keywords: Action Systems, energy consumption, formalism

TUCS Laboratory
Communication Systems

1 Introduction

Formal methods provides an environment to design, analyze, and verify digital hardware with the benefits of rigorous mathematical basis. In this study, the Action Systems formalism is applied [1]. It is a framework for specification and correctness preserving development of concurrent systems and it is based on an extended version of Dijkstra's language of guarded commands [9]. Development of the action system is done in a stepwise manner within the refinement calculus [2]. The specification of a hardware system is transformed into an implementation using correctness preserving transformations. In conventional Action Systems, only the logical correctness of the system is verified, while non-functional properties, like time, power and area, are not validated. The Action Systems formalism has been proved to be suitable for designing both synchronous [14], and asynchronous [13] systems.

In this study, we introduce a framework for energy estimation within the Action Systems context. By adopting this framework, we are able to formally analyze the energy consumption of the hardware system, using precise mathematical calculations, during the development phases from the specification down to implementation. In this paper, we define a formal model for energy estimation, that is applicable for both synchronous and asynchronous systems. The functionality of the model is illustrated using example system specifications.

Overview of the paper. We proceed as follows. In Section 2, we shortly describe the properties of the Action Systems formalism. Section 3 concentrates on the semantics of the energy consumption modeling. The formal energy consumption model is presented in Section 4. Section 5 shows, how the formal model is applied into example systems. Finally, in Section 6, we draw some conclusions.

2 Action Systems

The $Action\ Systems$ formalism is a state-based formalism for concurrent system specification and correctness-preserving development [2, 4]. Based on an extended version of the guarded command language of Dijkstra [9]. It has the $Refinement\ Calculus$ as the mathematical basis for disciplined derivation [3].

2.1 Actions

An action A is defined (for example) by

$$
\begin{array}{ll}
A ::= abort & (abortion, non-termination) \\
\quad | \ skip & (empty\ statement) \\
\quad | \ A_1 [\!] \ ... \ [\!] \ A_n & (non-deterministic\ choice) \\
\quad | \ x := e & ((multiple)\ assignement) \\
\quad | \ g \ \rightarrow \ A & (guarded\ command)
\end{array}
$$

where A_i, $i = 0, ..., n$, are actions; x is a variable or a list of variables; x_0 is a value(s) of the variable(s); e is an expression or a list of expressions; g is a predicate.

Semantics of actions. Action is considered to be atomic, which means that only the initial and final states are observed by the system. Thus, when selected for execution, the action is completed without any interference from other actions. Atomic actions may be represented by simple assignments or by more complex action compositions, such as the atomic sequence.

The actions are defined using weakest precondition for predicate transformers [9]. For instance, the correctness of an action A with respect to predicates P and Q (precondition and postcondition) is denoted by:

$$\{P\}A\{Q\} = P \Rightarrow wp(A, Q)$$

Here $wp(A, Q)$ is the weakest precondition for the action A to establish the postcondition Q. The $guard\ gA$ of an action A is defined by $gA = \neg wp(A, false)$. An action is enabled when its guard evaluates to $true$, otherwise disabled.

Quantified Composition of actions is defined by $[\bullet 1 \leq i \leq n : A_i] \widehat{=} A_1 ... \bullet ... \bullet A_n$, where the bullet $\bullet$ denotes any of the composition operators, and n is a number of actions $(n \in \mathbb{N})$.

2.2 Action System

An action system has a form:

sys *Name* (g) $[par]$
$|[$
type t
const c
var v
actions A
init "$initialization\ of\ the\ variables\ g\ and\ v$"
exec
do "$composition\ of\ actions\ A''$ **od**
$]|$

Three different parts can be identified from the action system description: $interface$, $declarations$, and $iteration$.

The interface part specifies global variables g, that is, variables that are visible outside the action system. In other words, global variables are accessible by other action systems. If an action system does not have any interface variables, it is a $closed$ action system otherwise it is an $open$ action system. The declaration part consists of type (t), variable (v), constant (c), and action (A) declarations. Furthermore, type definitions and initializations are described in the declaration part. Using the items introduced in the interface and declarative parts the operation of the system is described in the iteration section;in the **do** $-$ **od** loop.

The operation of an action system is started by initialization in which the variables are set to predefined values. Actions are selected for execution based on the composition operators and the enabledness of the actions. The operation is continued until there are no actions to enable, which temporarily aborts the system. Thus, the operation continues if some action enables it.

Parallel Action Systems The parallel behavior of the actions is defined by the non-deterministic choice. Thus, this requires that the two actions can be enabled simultaneously and that they do not have any read-write conflicts. In other words, an action does not read variables onto which another action writes. Consider two Action Systems A and B, the parallel composition of the systems, denoted $A \parallel B$ is:

sys $A \parallel B\ (g_{A_n} \cup g_{B_n})$
$|[$
var $l_{A_n} \cup l_{B_n}$
actions A_n, B_n
init $g_{A_n}, g_{B_n}, l_{A_n}, l_{B_n} = g_{A_n}0, g_{B_n}0, l_{A_n}0, l_{B_n}0$
exec
do $A_n \parallel B_n$ **od**
$]|$

Thus, the parallel composition combines the global variables as set $g_{A_n} \cup g_{B_n}$ while keeps the local variables distinct: $l_{A_n} \cap l_{B_n} = \emptyset$.

3 Towards a Formal Energy Consumption Model

The design flow for the formal energy estimation framework is introduced in this section. At first, we describe the energy consumption of the CMOS gate shortly. Then, we concentrate on switching activity estimation by using the Muller C element as an example [7]. Finally, we introduce an overview of the design flow, which lays the foundation for the formal energy consumption model.

3.1 Energy

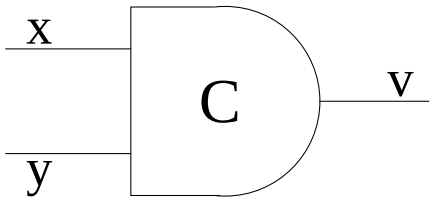
The amount of energy dissipated by the CMOS logic gate each time its output changes is roughly equal to the change in energy stored in the gate's output capacitance. Thus, the well known formula for energy consumption is:

$$E = \sum_{gates} \frac{1}{2} \cdot C_{gateload} \cdot V_{dd}^2 \cdot n \tag{1}$$

where the n stands for a switching activity estimate or the number of gate switches during a discrete time period.

3.2 Switching Activity Estimation - C Element

A C element is a commonly used asynchronous circuit component [7] (originally described in the pioneer work of David Muller). The component produces '1' as an output if all of its inputs are '1', and produces an output of '0' when all of its inputs are '0'. For all other input combinations, the component holds its previous value. The state diagram and graphical symbol of two input C element is illustrated in Figure 1.



x	y	v
0	0	0
0	1	v
1	0	v
1	1	1

Figure 1: Muller C-element

The operation of the C element can be described using Action Systems formalism, for instance, as follows:

sys $C\,(x, y, v : bool)$
$|[$
actions $C1 : x \wedge y \rightarrow v := T$
$C2 : \neg x \wedge \neg y \rightarrow v := F$
$C3 : \neg x \wedge y \vee x \wedge \neg y \rightarrow v := v$
init $x, y, v = F$
exec
do $C1 \,[\!]\, C2 \,[\!]\, C3$ **od**
$]|$

To estimate the energy consumption of the C element, we define that the time period T of which the switching activity is captured, is discrete. Moreover, the time period T can be divided into sub periods: $t, t+1, ..., t+l$. The l is defined as an unit delay between consecutive transitions. The time t is the first time when there is change either from '0' to '1' or '1' to '0', and the $t+1$ describes the second transition, and so on. The time $t+l$ is granted for the last transition under estimation, and $(t+l \in \mathbb{N})$. This timing estimation model is a simplification from the unit delay model presented in [10].

By adopting the timing definition above, we assume that the C element is executed as follows:

$$...x \wedge y, \neg x \wedge \neg y, \neg x \wedge y...$$

Since, the system description is quite straightforward we can define the switching activity n, as shown in Figure 5.

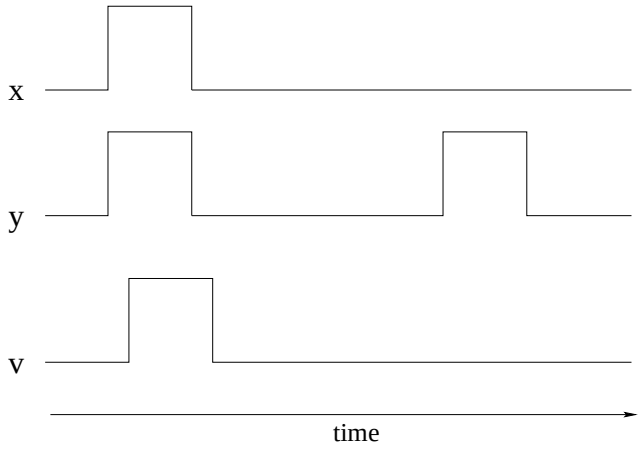


Figure 2: Transitions of C element during time period T

Consider the execution sequence, shown in Figure 5, we can define the order of the transitions by:

$$...(x, y, \neg v, t), (x, y, v, t+1), (\neg x, \neg y, v, t+2), (\neg x, \neg y, \neg v, t+3),$$
$$(\neg x, y, \neg v, t+4), (\neg x, \neg y, \neg v, t+5)...$$

Thus, at time t, the inputs x and y have a transition from '0' to '1'. Next, at time $t+1$ the output v is set to '1' etc.

3.2.1 Signal Transition Graph (STG)

STGs are a particular type of labeled Petri Nets, where the transitions are associated with the changes in the values of binary variables [6]. These variables can for instance be associated with wires, when modeling interfaces between blocks, or with input, output, and internal signals of a control circuit.

A timing diagram, shown for instance in Figure 5, specifies the events of a behavior and their causality. An STG is a formal model for this type of specifications [6]. In its simplest form, an STG can be considered as a causality graph in which each node represents an event and each arc a causality relation. Moreover, an STG can model all possible dynamic behaviors of the system. This is the role of the tokens held by some of the causality arcs. An $event$ is enabled when it has at least one token on each input arc. The enabled event can $fire$, which means that the event occurs. Thus, when the event fires, the token is removed from each input arc of the event, and put on each output arc. In other words, firing of a event produces the enabling of another event. In the specification represent the initial state of the system.

As an example, we re-define the possible transitions of the Muller C element by using STG [6]. The Actions Systems description of the C element is presented

in the previous subsection, the timing diagram is shown in Figure 5, and the corresponding STG presentation is shown in Figure 3. Rising and falling transitions of a signal are represented by the suffixes $+$ and $-$, respectively.

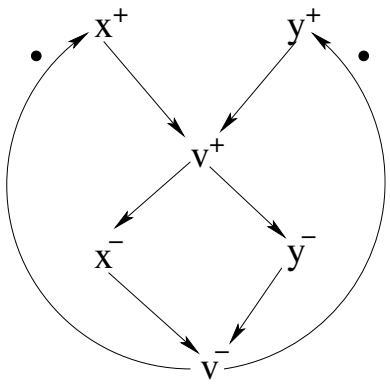


Figure 3: STG specification of the Muller C element

From the STG, not only we can see all possible transitions of the action system C, but also the potential parallel transitions. For instance, from Figure 3, we can notice that the parallel input transitions from '0' to '1' results '1' as an output. This assumption leads to a worst-case energy estimate for the C-element.

3.2.2 Petri Nets

Petri Nets have an abstract view of the events and states in the system, without considering their binary encoding. They are a promising tool for describing and studying systems, that are characterized as being concurrent, asynchronous, distributed, parallel, non-deterministic, and/or stochastic [12]. The dynamic and concurrent behavior of the systems are modeled by using tokens. An example transformation from Action Systems description to the Petri Net description is shown in the case study presented in next section.

3.3 Estimation Flow

In conclusion, we can define an energy estimation flow from the topics discussed in this section, shown in Figure 4. The formal energy model is specified according to the design flow presented.

At first, we define a time domain $\mathbb{T}$, which is represented using positive real numbers ($\mathbb{T} \in \mathbb{R}_+$). The time domain is assumed to be continuous ($\forall t_1, \exists t_2, (t_1 > t_2)$ [8]. For the energy estimation, we define a discrete time period T_d, which is part of the time domain $\mathbb{T}$, ($T_d \in \mathbb{T}$). Thus, we describe the period T_d by: $< t, t+1, ..., t+l >$, where the l is called as a unit delay between two consecutive transitions, and ($t, l \in \mathbb{R}_+$). The period T_d is therefore defined to be continuous: $]t, t+1, ..., t+l]$. The time values $< t, t+1, ..., t+l >$ are called as time stamps, and the purpose of these stamps is to determine the order of transitions in the

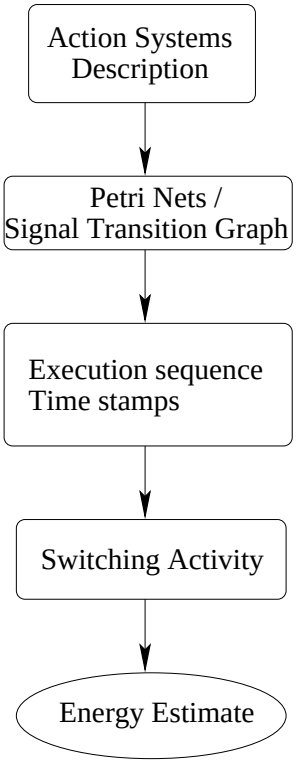


Figure 4: Energy estimation flow for Action Systems

system. Thus, the first transition(s) have time stamp t, the second one(s) have time stamp $t + 1$, etc. The $t + l$ is granted for the last transition under the time period T_d. Notice that, parallel transition share the same time stamp value. This timing model is a simplification from the unit delay model presented in [10].

Consider an arbitrary action A. For the given execution sequence, we define the switching activity per time stamp by using Petri nets [1] [6]. Signal transition graphs (STG) were also considered, but they use is restricted to systems that are modeled by using boolean variables. Of course, we can apply the STG description to model some subsystem or communication channel from a larger entity. According to the time stamp, we generate a transition table, where the rows are indexed by the time stamps and the columns are indexed by the the input / output signals of the system. Thus, the input / output signals means the global variables of the system in this context. The time stamps in transition table are in increasing order ($ts_i \leq ts_i + 1$). The transition table is used as a parameter for the energy estimation action.

After we have defined the switching activity for the action A at time period T_d, we can estimate the energy consumption. At first, we define the energy consumption per time stamp, for instance at time t:

[1]The reader is assumed to be familiar with the basic semantics of the Petri net.

$$E_A(t) = \frac{1}{2} \cdot V_{dd}^2 \cdot C_A \cdot n_A \quad (2)$$

where the C_A is the total node capacitance of the action A, n is the transition count of the action A at time t. In a similar manner we can define the energy consumption at time $t+1$ by taking the transition count n_A at time $t+1$, etc.

The energy values per time stamps are added together, and as a result we have the energy estimate $E_A(T_d)$ for the time period T_d.

$$E_A(T_d) = E_A(t) + E_A(t+1) + ... + E_A(t+l) = \sum_{l=0}^{n} E_A(t+l), (n \in \mathbb{N}) \quad (3)$$

4 Formal Framework for Energy Consumption Estimation

At first we define the functionality of the energy update action by using pseudo language description. This description is used as a guideline to the definition of the formal energy model.

4.1 The Energy Estimation Procedure

The energy estimate is calculated under discrete time period ($T_d \in \mathbb{T}$), which is divided into sub periods, denoted as $time\ stamps$: $((t, t+1, ..., t+l) \in T_d)$. The first transition in input / output signals of the system is granted by the time stamp t, the second transition is granted by the time stamp $t+1$, and so on. The granted time stamps are collected into a time stamp vector ts. Thus, the transition count needed for the energy estimation is calculated by monitoring the switching activity of the input/output signals of the system under investigation. Therefore, we define a transition table, where the rows are indexed by the time stamps $ts[i]$ and the columns are indexed by the input / output signals. For a given execution sequence, the transition table can be read from the systems STG description. The estimation procedure receives two parameters: time stamp vector ts and the transition table trt.

For the switching activity calculation, we define two variables of type $vector$, previous state, $pstate$, and current state, $cstate$. At the initialization phase the previous state variable is set to zero. The initial value of current state vector is read from the first row of the transition table.

Energy Estimation Procedure:

variables

ts[i] (action specific time stamp)

ct (current time stamp under evaluation)
cstate[j] (system specific current state vector)
pstate[j] (system specific previous state vector)
s[i] (the difference between transitions in previous state and current state)
n(i) (the total transition count of the system during discrete time period T)
nos (number of signals in the system)

– 1
init
cs = t;
nos = signal count;
j=0;
k=0;

for j $\leq$ nos *do*
pstate[j] = 0;
s[j] = 0;
j = j + 1 *endfor*

– 2
for every action A(i) *do*
j,k=0,0;
if ts[l] = ct *then*
tr(ts[l], j);
for k $\leq$ nos *do*
s[k] = cstate[k] - pstate[k];
k= k + 1;
endfor
else
– 3
ct = ct + 1;
n[l] = n[l]+($\sum_{t=1}^{nos}$s[t]);
pstate[] = cstate[];
l=l+1;
endif
endfor

– 4
procedure tr(ts[l], j)
for j $\leq$ nos; *do*
cstate[j] = transitionTable[ct, j];
j=j+1;
endfor
endprocedure

The current time stamp variable ct is initialized to t, which is the first time value of the period T_d when transition occurs (1). Thus, the current time stamp is then compared with the system specific time stamps $ts[i]$ (2). If the time stamps are equal, we calculate the transition count by comparing the current state variable $cstate$ to the previous state variable $pstate$. The result is stored into the result vector $s[i]$. Transition activity for the energy estimation is then calculated by adding the states '1' together from the result vector $s[i]$ (3). After that, the current time variable ct is increased by one, and the current state at time t becomes previous state at time $t + 1$. The value of the new current state is read from the transition table using the procedure tr (4). Finally, we can calculate the energy estimate according to the equations 2 and 3.

4.2 Formal model for energy consumption

According to the pseudo language description of the energy estimation procedure, we define a formal model for the energy estimation.

4.2.1 Behavior

Behavior of an action system is a sequence s of states with two components [4]: $s =< (l_1, g_1), (l_2, g_2), ... >$, where l_i and g_i ($i \in \mathbb{N}$) are the states of the local and global variables, respectively. We can define the behavior of the actions with the energy estimation property as a sequence $e : e =< (l_1, g_1, ts_1, tc_1), (l_2, g_2, ts_2, tc_2), ... >$, where the ts_i denotes the time when there was a change in state, $ts_i \leq ts_{i+1}$. The variable tc_i describes the transitions at time ts_i. Thus we introduced two new variables into the sequence s. These variables carry the information related to timing and switching activity.

A trace, a sequence of observable states, in the Action Systems is formed by removing all the local variables from the states in the sequence s, and then removing all the consecutive states that are identical, called stuttering states [4]. A trace is formed using the same procedure except that the time and energy variables, which are local variables, are not removed. Therefore, from the time stamps ts_i, we can form a trace that describes the transition activity of the system during during that time. Moreover, we can illustrate the development in the energy consumption per each $time\ stamp$.

4.2.2 Formal model for the energy estimation procedure

Consider an action system A with the following $do - od$-loop:

$$\textbf{do } [\ ]\ 1 \leq i \leq n : A_i]\ //E_u \textbf{ od}$$

where A_i has a form

$$A_i = A_{i,p}\ [\!]\ A_{i,t}\ [\!]\ A_{i,c}$$

where $A_{i,p}$ performs a $procedure$ call, $A_{i,t}$ calculates the $transition$ count, and $A_{i,c}$ is a $count$ action. The actions are defined by:

$$A_{i,p} : \neg b_i \wedge gA \rightarrow tr_i(ts_i[l]); b_i := T;$$
$$A_{i,t} : b_i \wedge gA \rightarrow b_i, n_i, l := F, |(\sum_{k=1}^{nos} cstate_i[k]) - (\sum_{k=1}^{nos} pstate_i[k])|, l + 1$$
$$A_{i,c} : \neg b_i \wedge \neg gA \rightarrow ct, pstate_i, l, := ct + 1, cstate_i, l + 1,$$

where i is the number of actions, $i \in \mathbb{N}$. The guard gA is defined by: $gA \mathrel{\widehat{=}} ct = ts[l] \wedge l \leq n$, which is used, together with the boolean variable b_i, to sequence the operation of the actions into three parts. The guard is set $true$ at the initialization phase, where the current time variable ct is set to t. Moreover, the index variable l is set to 0, which reads the first value from the time stamp vector, and returns the value t. The second condition is also $true$ at the initialization phase, and becomes $false$ when the last cell $n - 1$ time stamp vector ts is read. Thus, it is an end constraint.

The state variables: current state $cstate$, and the previous state $pstate$ are defined as of type vector. The length of the vector depends on the system specific constant nos that describes the number of signals in the system under evaluation, thus the length is nos.

The energy estimation procedure receives two parameters: the transition table trt and the time stamp vector ts. The transition table is defined by: $trt[0..n - 1][0..m - 1]$, where $(n, m \in \mathbb{N})$. The transition table trt is read by the action A_p through a procedure call tr, which returns the new value of the current state variable $cstate$. Thus, the procedure tr is defined by:

$$\textbf{proc} \quad tr(ts[l]) : (cstate := trt[ts[l], 1..nos - 1])$$

We define that the rows of the transition table trt are indexed by the time stamps $ts[l]$, and the columns are indexed from 0 to $nos - 1$. Notice that the first value of the column is the time stamp, and therefore we exclude it from the state variable ($cstate$). The guards in the Action Systems description are embedded in the transition table for the energy estimation model.

At the initialization phase, the guard gA is set to $true$, and the boolean variable is set to $false$. The action A_p performs the procedure call, which returns the new value of the current state variable $cstate$. Notice that, this value is read from the transition table, which is defined according to the given execution sequence. Moreover, it sets the boolean variable b_i to $true$. When both; the guard gA and the variable b_i evaluates to $true$, the transition count n per time stamp $ts[l]$ is calculated (by action A_t). For instance, assuming that the current time stamp is $t + 1$, then the transition count is calculated by comparing the current state at time $t + 1$ to the previous state at time t. The result is then stored into the transition count variable n. Then the $count$ action (A_c) increases the current time variable ct by the amount of unit delay. The operation of these three actions (A_p, A_t, A_c) is

repeated until there is no new time stamp values in the time stamp vector ts, thus the guard gA becomes $false$.

The action E_u is defined as an $update$ action. It counts the energy estimate according to the transition count n.

$$E_u \widehat{=} [[] 1 \leq i \leq n : E_{e,i} = E_{k,i} \cdot n_i]$$

where $E_{k,i} = \frac{1}{2} C_i \cdot V_{dd}^2$ and i is the number of actions, ($n \geq 1 \wedge n \in \mathbb{N}$).

In order to prevent complex descriptions, we define a shorthand notation for the energy update action:

$$[\,[]\, 1 \leq\ i\ \leq\ n : A_i[ts_i, trt_i]] \widehat{=} [\,[]\, 1 \leq\ i\ \leq\ n : (A_{p,i} \,[]\, A_{t,i} \,[]\, A_{c,i}\ //E_u)]$$

where ts_i is the time stamp vector and the trt_i is the transition table of the given Action System A_i.

Composition of Action Systems with energy estimation property. Consider two actions A_n and B_n with the energy estimation property. These two actions have distinct local variables, $l_{A_n} \cap l_{B_n} = \emptyset$, and the global variables are a set, $g_{A_n} \cup g_{B_n}$. We require that the initialization of the global variables $g_{A_n} \cup g_{B_n}$ in the systems A and B are consistent with each other, and therefore the initial values are equivalent in the systems.

The functionality of the two parallel systems is specified by the actions $A_n[ts_{A_n}, trt_{A_n}]$ and $B_n[ts_{B_n}, trt_{B_n}]$, respectively. The parallel composition of A and B is:

sys $A[]B\ (g_{A_n} \cup g_{B_n})$
$|[$ **var** $l_{A_n} \cup l_{B_n} : natural$
init $g_{A_n}, l_{A_n}, g_{B_n}, l_{B_n} := g_{A_n}0, l_{A_n}0, g_{B_n}0, l_{B_n}0$
exec
do $A_n[ts_{A_n}, trt_{A_n}][]B_n[ts_{B_n}, trt_{B_n}]$ **od**
$]|$

where $A_n[ts_{A_n}, trt_{A_n}]$ and $B_n[ts_{B_n}, trt_{B_n}]$ is according to definition: $(A_p \,[]\, A_t[]\, A_c)$ $[](B_p \,[]\, B_t[]\, B_c)//E_u$. Notice that the composition of the actions systems combines the state spaces of the constituent action systems, merges the update actions E_u by unifying the local energy and time variables.

5 Component Modeling

In this section, the functionality of the formal energy estimation model is illustrated. At first, we use the the formal energy model to the Muller C element, which was discussed in Section 3. Then, we consider case when the example system is an asynchronous Action System. Furthermore, the same system is described synchronously. Finally, the results are compared.

5.1 C element

The action system description of the C element is shown in Section 3. Next, we include the energy estimation action into the the system specification.

sys $C\,(x, y, v : bool)[ts_c, trt_c]$ |[
const $nos : 3$
actions $C1 : x \wedge y \rightarrow v := T$
$C2 : \neg x \wedge \neg y \rightarrow v := F$
$C3 : \neg x \wedge y \vee x \wedge \neg y \rightarrow v := v$
init $x, y, v, ts_c 0 = F, F, F, t$
exec
do $C1[ts_c, trt_c] \,[\!]\, C2[ts_c, trt_c] \,[\!]\, C3[ts_c, trt_c]$ **od**
]|

where the constant nos is the number of signals in the system. Thus, in this Action System, the value of the constant nos is three. Moreover, the system receives the time stamp vector ts and the transition table trt as parameter. The iteration part by definition is:

$$\begin{aligned} C1 \,[\!]\, C2 \,[\!]\, C3 \,\widehat{=}\, & (C1_p \parallel C1_t \parallel C1_c) \\ & (C2_p \parallel C2_t \parallel C2_c) \\ & (C3_p \parallel C3_t \parallel C3_c) \,//E_u \end{aligned}$$

Lets assume the following execution sequence:

$$...(x \wedge y), (\neg x \wedge \neg y), (x \wedge \neg y)...$$

For the given execution sequence, we define the order of the events according to the STG presented in Section 3.

$$...(x, y, \neg v, t), (x, y, v, t+1), (\neg x, \neg y, v, t+2), (\neg x, \neg y, \neg v, t+3), (x, \neg y, \neg v, t+4)...$$

At time t the signals x, and y have transition from '0' to '1', and at time $t+1$ the output signal v of the C element has transition from '0' to '1', and so on. For the energy estimation procedure, we define a transition table for each time stamp, which is shown in Table 1.

Table 1: Transition table trt_c

Time stamp (ts[i])	x	y	v
init.	0	0	0
t	1	1	0
t+1	1	1	1
t+2	0	0	1
t+3	0	0	0
t+4	1	0	0

By re-writing the execution sequence:

$$
\begin{array}{c}
...C1, C1_p, C1_t, C1_c, E_u, \\
C1, C1_p, C1_t, C1_c, E_u, \\
C2, C2_p, C2_t, C2_c, E_u, \\
C2, C2_p, C2_t, C2_c, E_u, \\
C3, C3_p, C3_t, C3_c, E_u ...
\end{array}
$$

At first, the action C1 is enabled followed by the energy estimate actions ($C1_p, C1_t, C1_c, E_u$). Then, at time $t+1$ the output v is set to '1', and therefore the same sequence ($C1_p, C1_t, C1_c, E_u$) is repeated. The energy estimate action compares the previous state vector to the current state vector, $pstate = [1, 1, 0]$ and the $cstate = [1, 1, 1]$. Therefore, it calculates the transition count at time $t+1$ to be one ($n[i] = 1$). Next, the action $C1$ is disabled and the action $C2$ is enabled. Again the transition count calculation is two phase procedure, just as it was with the action $C1$. Finally, the $C2$ is disabled and the action action $C3$ is enabled.

The observable behavior of the C element is illustrated in trace, which is formed by collecting the states of the global variables and the state of the energy variable.

$$
\begin{array}{rl}
tr_C = & (\neg x, \neg y, \neg v, 0) \\
& (x, y, \neg v, 2E_k) \\
& (x, y, v, 2E_k + E_k) \\
& (\neg x, \neg y, v, 3E_k + 2E_k) \\
& (\neg x, \neg y, \neg v, 5E_k + E_k) \\
& (x, \neg y, \neg v, 6E_k + E_k)
\end{array}
$$

The graphical representation of the trace is shown in Figure 5, where the $E_k = \frac{1}{2} \cdot V_{dd}^2 \cdot C_c$. The capacitance C_c is the total node capacitance of the C element.

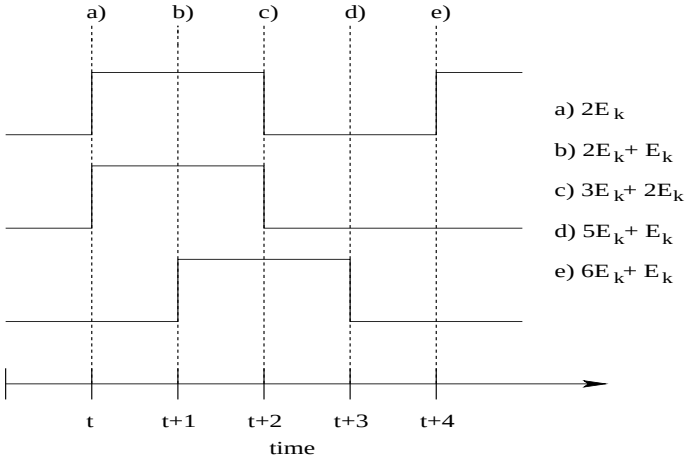


Figure 5: Transition diagram from the trace tr_c

The energy consumption estimate is then calculated according to the equations 2 and 3. For simplicity, we use the E_k notation for the technology dependent parameters, and therefore the total energy consumption estimate for the system C at time period T is:

$$E_c[T] = 2E_k + E_k + 2E_k + E_k + E_k \approx 7E_k$$

We can approximate the energy estimate to be $E_c(T) \approx 7E_k$, because we are estimating the power consumption of an one C element gate. Notice that, if there were larger system description under investigation, The node capacitance per action may vary significantly. Therefore, the capacitances should be counted separately for each action before the addition operation.

5.2 Asynchronous computation

Consider the system S_{Async} below:

sys $S_{Async}\ (dv, rw, din, dout, v : bool, bvec, bvec, bvec)$
|[
type $bvec : (0, ..., n-1)$
init $dv, rw, v, din, dout := F, T, "00", "00", "00"$
actions $S1 : dv \wedge rw \rightarrow v, rw := f(din), F$
$S2 : dv \wedge \neg rw \rightarrow dout, rw, dv := v, T, F$
exec
do $S1 \parallel S2$ **od**
]|

where the action $S1$ is a write action, and the action $S2$ is a read action. Thus, the action $S1$ evaluates the value of the function f according to the value of the input din. The new value of the function f is then stored into the register variable v. The action $S2$ reads the register variable v and transfers the value to the output $dout$.

The data valid signal dv is set $true$ to notify the system that there is new data to process. In other words, it operates as an request signal from the environment. After the read / write cycle is completed the data valid signal is set to $false$, which will notify to the environment that the system is in idle state and ready to process new data.

The input and output data signals $din, dout$ are defined as of type binary vector $bvec$. The length of the vector is defined by: $(0, ..., n-1)$, where $(n \in \mathbb{N})$. In this example, the $n = 1$, and the following values are applied to the input signal $din =$ "01", "11". Furthermore, to capture the transition activity of the vector signals properly, we model them as a separate signal, i.e. the input signal din is modeled as din_1, and din_0. Therefore, the number of signals is eight ($nos = 8$). Thus, the action system description of the system S_{Async} with the energy estimation property is:

sys $S_{Async}\ (dv, rw, din, dout, v : bool, bool, bvec, bvec, bvec)[ts_A, trt_A]$
$|[$
const $nos : 8$
type $bvec : (0, ..., n-1)$
init $dv, rw, v, din, dout := F, T, "00", "00", "00"$
actions $S1 : dv \wedge rw \rightarrow v, rw := f(din), F$
$S2 : dv \wedge \neg rw \rightarrow dout, rw, dv := v, T, F$
exec
do $S1[ts_A, trt_A] \parallel S2[ts_A, trt_A]$ **od**
$]|$

The iteration part by definition is:

$$S1 \parallel S2 \mathrel{\widehat{=}} (S1_p \parallel S1_t \parallel S1_c) \\ (S2_p \parallel S2_t \parallel S2_c)\ //E_u$$

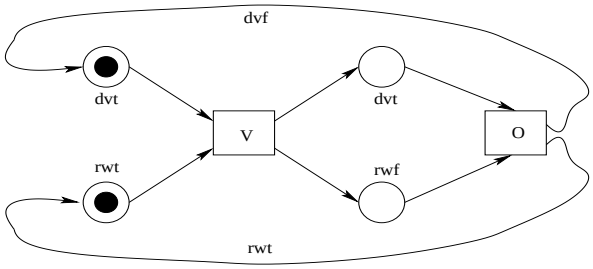


Figure 6: Petri net description from the system t_{Async}

The transition table for the system S_{Async} is generated according to the Petri net description, shown in Figure 6. The read / write signal is set to $true$ (rwt) at the initial state, and when the data valid signal is set to $true$ by the environment (dvt), the event v becomes $true$. In other words, the evaluated input data is written into the register. Similarly, when the read / write (rwf) variable becomes $false$, the output event becomes $true$. Then, the system returns to the initial state, and waits until the request signal (data valid) becomes $true$ again. The transition table is shown in Table 2.

Thus, the execution sequence is:

$$...S1, S1_p, S1_t, S1_c, E_u, \\ S1, S1_p, S1_t, S1_c, E_u, \\ S2, S2_p, S2_t, S2_c, E_u, \\ S2, S2_p, S2_t, S2_c, E_u, \\ S1, S1_p, S1_t, S1_c, E_u, \\ S1, S1_p, S1_t, S1_c, E_u, \\ S2, S2_p, S2_t, S2_c, E_u, \\ S2, S2_p, S2_t, S2_c, E_u...$$

The trace tr_{Async} for the system S_{Async} is then illustrated below:

Table 2: Transition table trt_A for system S_{Async}

Time stamp (ts[i])	dv	rw	v_1	v_0	din_1	din_0	$dout_1$	$dout_0$
init.	0	1	0	0	0	0	0	0
t	1	1	0	0	0	1	0	0
t+1	1	1	0	1	0	1	0	0
t+2	1	0	0	1	0	1	0	1
t+3	0	1	0	1	0	1	0	1
t+4	1	1	0	1	1	1	0	1
t+5	1	1	1	1	1	1	0	1
t+6	1	0	1	1	1	1	1	1
t+7	0	1	1	1	1	1	1	1

$$
\begin{aligned}
tr_{Async} = {} & (\neg dv, rw, \neg v_1, \neg v_0, \neg din_1, \neg din_0, \neg dout_1, \neg dout_0, 0), \\
& (dv, rw, \neg v_1, \neg v_0, \neg din_1, din_0, \neg dout_1, \neg dout_0, 2E_k), \\
& (dv, rw, \neg v_1, v_0, \neg din_1, din_0, \neg dout_1, \neg dout_0, 2E_k + E_k), \\
& (dv, \neg rw, \neg v_1, v_0, \neg din_1, din_0, \neg dout_1, dout_0, 3E_k + 2E_k), \\
& (\neg dv, rw, \neg v_1, v_0, \neg din_1, din_0, \neg dout_1, dout_0, 5E_k + 2E_k), \\
& (dv, rw, \neg v_1, v_0, din_1, din_0, \neg dout_1, dout_0, 7E_k + 2E_k), \\
& (dv, rw, v_1, v_0, din_1, din_0, \neg dout_1, dout_0, 9E_k + E_k), \\
& (dv, \neg rw, v_1, v_0, din_1, din_0, dout_1, dout_0, 10E_k + 2E_k), \\
& (\neg dv, rw, v_1, v_0, din_1, din_0, dout_1, dout_0, 12E_k + 2E_k),
\end{aligned}
$$

The graphical representation of the trace tr_{Async} is shown in Figure 7. For simplicity, we use the shorthand notation $E_k = \frac{1}{2} \cdot V_{dd}^2 \cdot C_i$. The C_i is denoted as the node capacitance of action i.

The energy estimate of the system S_{Async} is:

$$E(T_d) = 2E_k + E_k + 2E_k + 2E_k + 2E_k + E_k + 2E_k + 2E_k \approx 14E_k$$

At this phase we do not apply any capacitance model for the gates, and therefore the approximation shown above roughly estimates the total power consumption of the system S_{Async} during time period T_d. However, when the technology related parameters (C, V_{dd}^2) are adopted, the value of the E_k is calculated separately for each time stamp, and then added together.

5.3 Clocked computation

In a synchronous design, the clock is used to sequence the operation of the system. The clock is defined by the action Clk (for instance):

$$Clk \mathrel{\widehat{=}} \neg clk \rightarrow clk := T \parallel clk \rightarrow clk := F$$

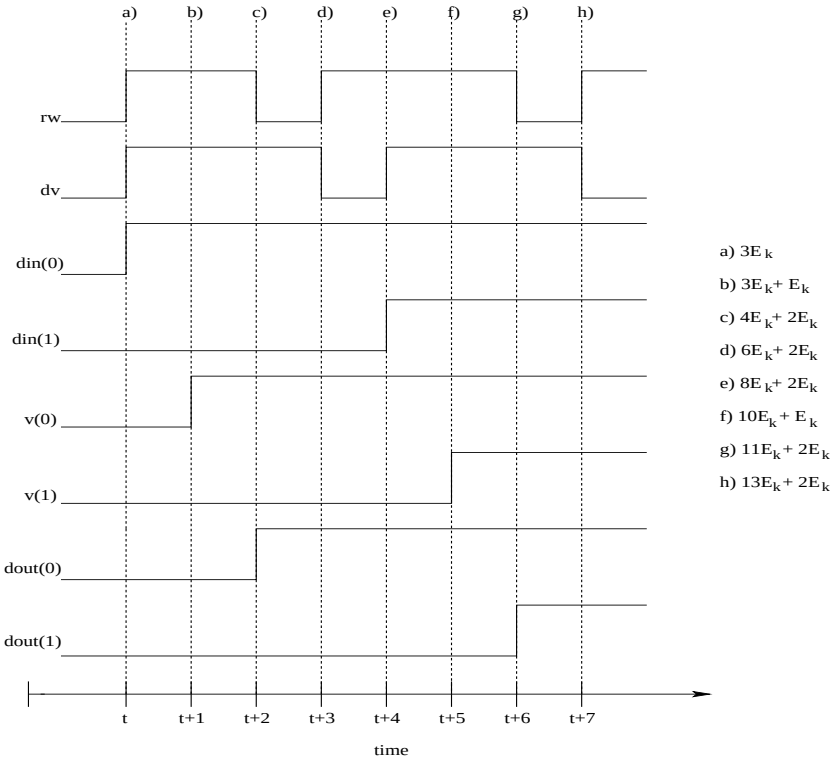


Figure 7: Transition diagram from the trace t_{Async}

The asynchronous composition of the system S_{Async} is re-written as a synchronous system description, denoted by S_{Sync}. It has three actions: $S1$ is a write action, $S2$ is a read and store action, and the last action Clk is the clock definition. In other words, the operation of the actions $S1$ and $S2$ is sequenced by the clock signal. The synchronous system S_{Sync} with the energy estimation property is shown below:

sys $S_{Sync}\,(din, dout, clk, v, : bvec, bvec, bool, bvec)[ts_S, trt_S]$
$|[$
const $nos : 8$
type $bvec : 0, ..., n-1$
init $rw, clk, din, dout, v := F, F, "00", "00", "00"$
actions $S1 : clk \wedge \neg rw \rightarrow v, rw := f(din), T$
$S2 : \neg clk \wedge rw \rightarrow dout, rw : v, F$
$Clk : \neg clk \rightarrow clk := T \,[\!]\, clk \rightarrow clk := F$
exec
do $(S1[ts_S, trt_S] \,[\!]\, S2[ts_S, trt_S])\ \ //Clk[ts_S, trt_S]$ **od**
$]|$

The iteration part by definition is:

$$(S1 \,[\!]\, S2)\ //\ Clk \mathrel{\widehat{=}} ((S1_p \,[\!]\, S1_t \,[\!]\, S1_c),\ (S2_p \,[\!]\, S2_t \,[\!]\, S2_c)),\ (Clk_p \,[\!]\, Clk_t \,[\!]\, Clk_c)//E_u$$

In this case study, the constant nos is eight ($nos = 8$), and the signals of type binary vector $bvec$ are modeled separately as one bit signals. To compare the results with the asynchronous case study, we use the same input data sequence: "01", "11". The operation of the system is quite similar with the asynchronous implementation, except that the data valid signal is replaced by the clock signal. Therefore, we do not present the Petri net description here. The transition table for the synchronous system S_{sync} is illustrated in Table 3.

Table 3: Transition table trt_S for system S_{Sync}

Time stamp (ts[i])	rw	clk	v_1	v_0	din_1	din_0	$dout_1$	$dout_0$
init.	0	0	0	0	0	0	0	0
t	0	1	0	1	0	1	0	0
t+1	1	0	0	1	1	1	0	1
t+2	0	1	1	1	1	1	0	1
t+3	1	0	1	1	1	1	1	1
t+4	0	1	1	1	1	1	1	1
t+5	0	0	1	1	1	1	1	1

The results from the energy estimation action are shown in the trace tr_{Sync}:

$$\begin{aligned} tr_{Sync} = &(\neg rw, \neg clk, \neg v_1, \neg v_0, \neg din_1, \neg din_0, \neg dout_1, \neg dout_0, 0),\\ &(\neg rw, clk, \neg v_1, v_0, \neg din_1, din_0, \neg dout_1, \neg dout_0, 3E_k),\\ &(rw, \neg clk, \neg v_1, v_0, din_1, din_0, \neg dout_1, dout_0, 3E_k + 4E_k),\\ &(\neg rw, clk, v_1, v_0, din_1, din_0, \neg dout_1, dout_0, 7E_k + 3E_k),\\ &(rw, \neg clk, v_1, v_0, din_1, din_0, dout_1, dout_0, 11E_k + 3E_k),\\ &(\neg rw, clk, v_1, v_0, din_1, din_0, dout_1, dout_0, 14E_k + 2E_k),\\ &(\neg rw, \neg clk, v_1, v_0, din_1, din_0, dout_1, dout_0, 16E_k + E_k) \end{aligned}$$

The graphical representation of the trace t_{Sync} is shown in Figure 8.
The total energy consumption estimate at time period T_d is defined:

$$E(T_d) = 3E_k + 4E_k + 3E_k + 3E_k + 2E_k + E_k \approx 16E_k$$

where the $E_k = \frac{1}{2} \cdot V_{dd}^2 \cdot C_i$. The C_i is denoted as the node capacitance of action i. Again, we approximate the total energy consumption, since at this phase, we do apply any technology related parameters nor capacitance models.

5.4 Comparison

The operation of example system is described by using both synchronous and asynchronous design approaches. The example system evaluates the value of the function f according to the input value. Stores the result into the register and outputs it. We estimated the energy consumption during two read / write cycles by using the following input sequence "01", "11".

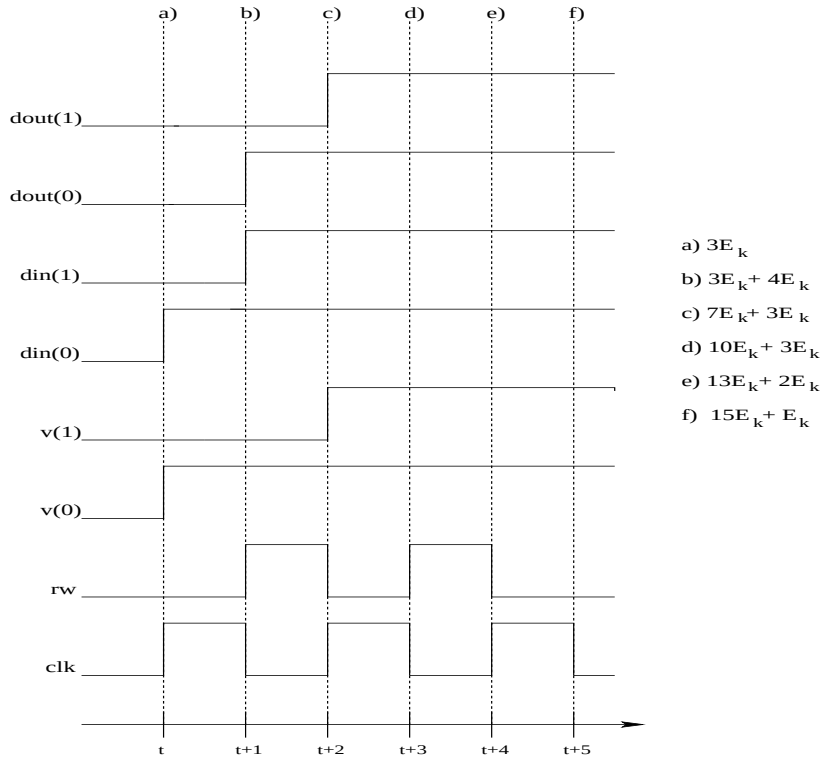


Figure 8: Transition diagram from the trace t_{Sync}

The asynchronous system, denoted by S_{Async}, is estimated at time period T_d. The time period consist of seven time stamps, $T_d = < t, t+1, ..., t+7 >$. Thus, the energy consumption estimate is: $E(T_d) = 2E_k + E_k + 2E_k + 2E_k + 2E_k + E_k + 2E_k + 2E_k \approx 14E_k$, where the E_k is noted by: $\frac{1}{2} \cdot C \cdot V_{dd}^2$.

Similar analysis is carried out with the synchronous system, noted as S_{Sync}, under the time period T_d. The time period consists of five time stamps, $T_d =< t, t+1, ..., t+5 >$. Thus, as a result we have an energy consumption estimate of $E(T_d) = 3E_k + 4E_k + 3E_k + 3E_k + 2E_k + E_k \approx 16E_k$.

The asynchronous system needed two time stamps more to calculate the two read / write cycles than its synchronous counterpart. However, since we do not use the actual delays between transition nor do we know the clock frequency, we cannot assume that the other one would be significantly faster than the other. Thus, at this level of abstraction, the calculation speed makes no significant difference between the designs in terms of performance.

The energy consumption is smaller in the asynchronous system than it is in the synchronous one. In this case study, we assumed that the two read / write cycles are consecutive. However, we could assume that there is an arbitrary long idle period between the two cycles. Thus, the asynchronous system have natural support for the power down mode, in other words, if there is no data to process, the system is in idle state. On the contrary to the synchronous system, where at least the flip-flops and clock distribution network consumes energy at each clock cycle

whether they are involved in the useful system function or not. Thus, this increases the energy consumption in the synchronous system, while the energy consumption of the asynchronous system remains the same. Another clock related issue is that in the synchronous system the simultaneous switching activity per time stamp is higher. This increases the amount of switching noise in the system [11].

6 Conclusions and Future Work

In this paper, we introduced a formal energy estimation framework for the Action Systems formalism. By adopting this framework, we are able to formally analyze the energy consumption of the hardware system from the specification down to the implementation. The energy estimation is carried out during discrete time period T_d, which is divided into time stamps. These time stamps are used to define the order of the transitions in the system. The transitions per time stamps are defined according to the Petri net (or STG) description of the system, and stored into the transition table. Thus, the energy estimation action calculates the switching activity estimate per time stamp. Finally, the energy consumption per time stamp is calculated. Notice that, the total energy of the time period T_d is the sum of the energy consumption values per time stamp. To illustrate the functionality of the formal energy estimation framework, we implemented a simple action system structure by using both synchronous and asynchronous design approaches. Then, the results of the energy estimation were compared.

Future Work: The experiences of this study revealed the possibilities of the defined Action System property. The next step is to take the existing framework into deeper abstraction level, that is to the gate level. In other words, to include the technology dependent parameters and the node capacitance model into the formal model. Then we are able to compare the accuracy of the formal model with the existing estimation methods. Moreover, the purpose is to upgrade the energy estimation model to the power estimation model by applying the existing timing model for the Action Systems, noted as Timed Actions [15], into the energy estimation framework. Furthermore, it would be interesting to use Timed Actions in the energy estimation model as well.

References

[1] R. J. R. Back. *On the Correctness of Refinement Steps in Program Development*, Ph.D Thesis, university of Helsinki, 1978.

[2] R. J. R. Back and K. Sere. *From Modular Systems to Action Systems*, in Proc. of Formal Methods Europe' 94, Spain, October 1994. Lecture notes on computer science, Springer-Verlag.

[3] R. J. Back and J. von Wrigth. *Refinement Calculus: A Systematic Introduction*, Springer-Verlag, April 1998.

[4] R. J. Back and J. von Wrigth. *Trace Refinement of Action Systems*, in International Conference on Concurrency Theory, pp. 367-384, 1994.

[5] P. A. Beerel, K. Y. Yun, S. M. Nowick and P-C. Yeh. *Estimation and Bounding of Energy Consumption in Burst-Mode Control Circuits*, in Proc. of the 1995 IEEE/ACM international conference on Computer-aided design 2nd edition.

[6] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. *Logic Synthesis of Asynchronous Controllers and Interfaces*, Springer-Verlag, 2002.

[7] A. Davis, and S. M. Nowick. *Asynchronous Circuit Design: Motivation, Backround, & Methods*, in G. Birtwistle, and A. Davis, Editors, Asynchronous Digital Circuit Design, Springer, 1995.

[8] J. Davies, and S. Schneider. *A Brief History of Timed CSP*, Theoretical Computer Science, 138(2):243-271, February 1995.

[9] E. W. Dijkstra. *A Discipline of Programming*, Prentice-Hall International, 1976.

[10] A. Ghosh, S. Devadas, K. Keutzer, and J. White. *Estimation of Average Switching Activity in Combinational and Sequential Circuits*, in Proc. ACM/IEEE 29th Design Automation Conference, June 1992, pp. 253-259.

[11] P. Liljeberg, J Tuominen, S. Tuuna, J Plosila, and J. Isoaho. *Self-Timed Methodology and Techniques for Noise Reduction in NoC Interconnects*, Chapter 11 in Interconnect-Centric Design for Advanced SoC and NoC, Kluwer Academic Publishers, Boston, July 2004, pp. 285-313, ISBN 1-4020-7835-8.

[12] T. Murata. *Petri Nets: Properties, Analysis and Applications*, in Proc. of the IEEE, vol 77. no. 4, April 1989, pp. 541-580.

[13] J. Plosila. *Self-Timed Circuit Design - The Action Systems Approach*, Ph.D Thesis, University of Turku, 1999.

[14] T. Seceleanu. *Systematic Design of Synchronous Digital Circuits*, Ph.D Thesis, Turku Centre of Computer Science, 2001.

[15] T. Westerlund, and J. Plosila. *Formal Timing Model for Hardware Components*, in Proc. 22nd Norchip Conference, November 2004, Oslo, Norway, pp. 293-296.

Turku
Centre *for*
Computer
Science

Lemminkäisenkatu 14 A, 20520 Turku, Finland | www.tucs.fi

University of Turku
- Department of Information Technology
- Department of Mathematics

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

Turku School of Economics and Business Administration
- Institute of Information Systems Sciences

ISBN 952-12-1578-X
ISSN 1239-1891