International Journal of Software Engineering
and Knowledge Engineering
Vol. 18, No. 6 (2008) 737–758
© World Scientific Publishing Company

World Scientific
www.worldscientific.com

Int. J. Soft. Eng. Knowl. Eng. 2008.18:737-758. Downloaded from www.worldscientific.com
by NATIONAL CHIAO TUNG UNIVERSITY on 04/25/14. For personal use only.

VERIFICATION OF DATAFLOW SCHEDULING

TSUNG-HSI CHIANG* and LAN-RONG DUNG†

Department of Electrical and Control Engineering, National Chiao Tung University,
1001 Ta Hsueh Road, Hsinchu City, Taiwan 300, ROC
**aries.ece89g@nctu.edu.tw*
†lennon@faculty.nctu.edu.tw

Received 4 September 2006
Accepted 23 April 2007

This paper presents the formal verification method for high-level synthesis (HLS) to detect design errors of dataflow algorithms by using Petri Net (PN) and symbolic-model-verifier (SMV) techniques. Formal verification in high-level design means architecture verification, which is different from functional verification in register transfer level (RTL). Generally, dataflow algorithms need algorithmic transformations to achieve optimal goals and also need design scheduling to allocate processor resources before mapping on a silicon. However, algorithmic transformations and design scheduling are error-prone. In order to detect high-level faults, high-level verification is applied to verify the synthesis results in HLS. Instead of applying Boolean algebra in traditional verification, this paper adopts both Petri Net theory and SMV model checker to verify the correctness of the synthesis results of the high-level dataflow designs. In the proposed hybrid verification method, a high-level design or DUV (design-under-verification) is first transformed into a Petri Net model. Then, Petri Net theory is applied to check the correctness of its algorithmic transformations of HLS, and the SMV model checker is used to verify the correctness of the design scheduling. We presented two approaches to realize the proposed verification method and concluded the best one who outperforms the other in terms of processing speed and resource usage.

Keywords: Formal verification; high-level synthesis; dataflow; Petri net; model checking.

1. Introduction

This paper presents a hybrid verification method to verify high-level synthesis (HLS) results of dataflow algorithms. Typically, given a dataflow graph (DFG) or a DSP (digital signal processing) design and a set of design constraints, the HLS aims to generate tasks schedules with processor resources assignment. The HLS performs high-level algorithmic transformations including retiming, scaling, and unfolding techniques on the DFG to meet the architectural constraints and then allocates processor resources accordingly [1–5]. In general, most solutions to the scheduling problem can be found by heuristic and Integer Linear Programming (ILP) [6, 7]. Heuristic method finds good solutions for large problems quickly

Int. J. Soft. Eng. Knowl. Eng. 2008.18:737-758. Downloaded from www.worldscientific.com
by NATIONAL CHIAO TUNG UNIVERSITY on 04/25/14. For personal use only.

but suffers from tightly constrained problems where early pruning decisions exclude candidates, leading to superior solutions. On the other hand, the theoretical framework of ILP based method commonly uses several ILP mapping techniques with cost functions as model of constrained-based schedule. The cost functions may combine several performance measurements such as Iteration Period Bound (IPB), Periodic Delay Bound (PDB) and Processor Bound (PB), which reflect the absolute limits on computation rate, latency and area of hardware implementation [3, 8, 9]. ILP method exactly solves scheduling but has difficulties with time complexity and constraint formulation. Heuristic and ILP scheduling methods produce a single schedule at a time. In order to find an optimal one, scheduling algorithms may be applied iteratively. The overall error-prone refining process and the complexity increasing with more constraints added to problem formulations make scheduling algorithms difficult to solve. Herein, any mistake or incomplete description made in the scheduling procedures may lead to an illegal solution and defeat following synthesis results. In our opinion, introducing high-level verification in system design flow may benefit by speeding up the scheduling procedure by filtering out invalid scheduling and prevent from scheduling faults. Therefore, this paper intends to present a formal verification method to unveil the faults produced in HLS.

The proposed verification method is two-fold, and the verifier utilizes both techniques including Petri Net theory and SMV model checker in it. In the first fold, a high-level dataflow design is converted into a Petri Net model model which can hold data dependence of dataflow design. Therefore any legal high-level algorithmic transformation has to conform to the firing rules of the Petri Net model. In the second fold, SMV model checker is used to check the correctness of a Finite State Machine (FSM) for the data-path scheduling. An admissible FSM schedule must satisfy system specification of the dataflow design. In the proposed verifier, given a DUV (design-under-verification), two inputs including *system description* and *tasks schedule* are required. The system description is basically a fully-specified flow graph (FSFG) [10]. The FSFG represents the behavioral of the dataflow algorithm which is also a design entry of HLS. In order to meet architectural constraints, high-level algorithmic transformations normally reconstruct the original FSFG design and find the optimal tasks schedule. To verify the correctness of the algorithmic transformations, the reconstructed FSFG and its original FSFG design are converted into Petri Net domain. In PN domain, each Petri Net graph can be represented by a PN characteristic matrix. Two PN characteristic matrices including the reconstructed FSFG design and its original FSFG graph must satisfy PN characteristic matrix equation. By using two proposed traverse algorithms, the verifier tries to find the candidate reconstructed FSFGs from PN reachability tree. Each candidate reconstructed FSFG can be seen as a high-level algorithmic transformed design which correspond to its original FSFG design. All the relationships of the data dependence between each operation-pair of the reconstructed FSFG graph can be seen as the system specifications of all the composed operators of the FSFG graph. The system specifications are classified into three classes including the non-preemption, job

completion and precedence properties. Each legal executing sequence of the FSFG graph must satisfy those system specifications. Another input to the verifier is the tasks schedule expressed in the format of processor-time chart (or $P \times T$ chart). The $P \times T$ chart shows equally the executing sequence of all the tasks of dataflow algorithms. In order to verify the tasks schedule, $P \times T$ chart is re-represented by a FSM description. Then, the SMV model checker is used to verify the FSM machine which must satisfy the system specifications.

Int. J. Soft. Eng. Knowl. Eng. 2008.18:737-758. Downloaded from www.worldscientific.com
by NATIONAL CHIAO TUNG UNIVERSITY on 04/25/14. For personal use only.

1.1. *Related work*

While surveying the related work, the existential verification methods utilize technologies like BDD (Binary Decision Diagram) [11, 12], SAT (Satisfiability) solver [13,14], symbolic model checking [15–18] and theorem proving [19]. These technologies are extremely powerful but must be applied in register transfer level (RTL). In order to verify the HLS correctness, most literatures focused on developing a strategy for RTL validation between the synthesized RTL result and its abstract level description. In [20, 21], the verification task is partitioned into two subtasks, verifying the validity of register sharing and verifying correct synthesis of the RTL interconnection and control. Similarly, in [22–25], a high-level design is decomposed into the control part and the datapath part and modeled by using FSMD (Finite State Machine with Data Path) [26]. By applying such decomposition methods, a high-level scheduled design is divided into the control and the datapath. Thus, the equivalent checking [27] can be applied to check the correctness of datapath, and the model checking [28] can be used to verify the validity of control by utilizing the existential verification technologies.

In this paper, the proposed verification method is based on Petri Net and SMV model checker. Instead of using Boolean algebra, a high-level dataflow design is modeled by using proposed Petri Net transformation, thus we can utilize the Petri Net theory to verify the correctness of the HLS result at high level. When compared with the existential method, the proposed verification is to check the validity of high-level design at abstract level rather than synthesized RTL level. Therefore, high-level design faults that violate the non-preemption, job completion and precedence properties can be found, and reworks may be performed in the early stage of the design flow.

1.2. *Outline*

The remainder of this paper is organized as follows. Section 2 describes background of the high-level transformation techniques in HLS. The proposed high-level verification flow is presented in Sec. 3. The proposed two-stage verification algorithm is discussed in Sec. 4. In Sec. 5, we discuss the complexity analysis of the verification algorithms. In Sec. 6 some experimental results are given. Section 7 gives the conclusions of this paper.

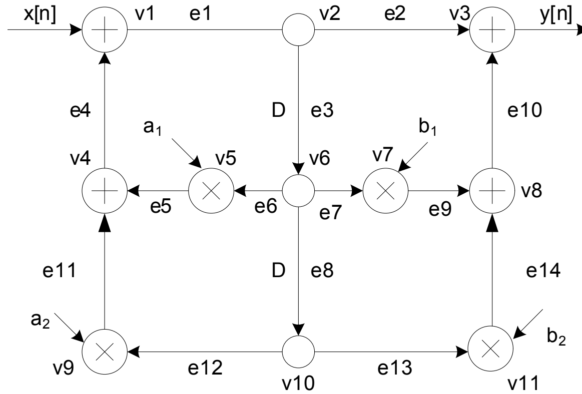

Fig. 1. A second order IIR filter in the form of FSFG.

Int. J. Soft. Eng. Knowl. Eng. 2008.18:737-758. Downloaded from www.worldscientific.com
by NATIONAL CHIAO TUNG UNIVERSITY on 04/25/14. For personal use only.

2. Background

2.1. *Fully-specified flow graph (FSFG)*

Fully-Specified Flow Graph (FSFG) [10] or DFG is a natural paradigm for describing DSP algorithms. An FSFG $G_{FSFG}(V, E, D)$, where $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$, is a three-tuple directed and edge-weighted graph. Vertex set V represents atomic operation of functional units. A vertex may have a zero execution delay, such as the signal duplicator; or may be assumed to take non-zero unit time, such as adder or multiplier. Directed edge set E describes the direction of flow of data between functional units. Inter data dependencies between functional units are denoted by weighted edges. Figure 1, for instance, shows a second order IIR filter in the form of FSFG.

The performance bound of the FSFG can be measured by the IPB (Iteration Period Bound), which is determined by loops of the graph [8, 10, 29]. The Iteration Period (IP) for a loop is defined as the total computational latency in the loop divided by the total number of delays. The IPB is the maximum value of IPs and represents the lower bound of MASP (Minimum Achieved Sample Period). In order to achieve MASP, designer may apply high-level transformations on their design in HLS, such as unfolding and retiming. We will discuss these techniques in following subsections.

2.2. *Retiming and unfolding*

Usually, the optimal IPB does not guarantee the optimal rate. Retiming is a process that may help make MASP equal to IPB. With the delay transfer or nodal transfer, it is possible to make MASP optimized. Unfortunately, the retiming technique might not guarantee the optimal MASP. Figure 2(a), for example, the MASP cannot be achieved by retiming since node v_1 requires $d_1 = 20$ time units to execute. To achieve the optimal rate, [8] presents the unfolding technique. Instead of describing

Int. J. Soft. Eng. Knowl. Eng. 2008.18:737-758. Downloaded from www.worldscientific.com
by NATIONAL CHIAO TUNG UNIVERSITY on 04/25/14. For personal use only.

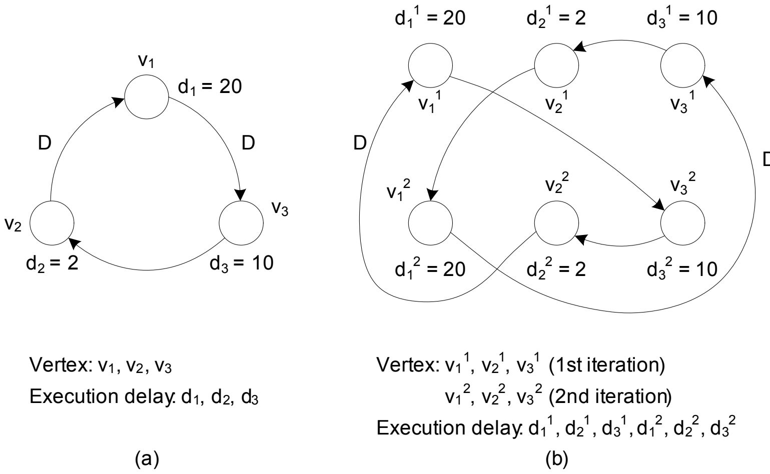

Fig. 2. Unfolding result. (a) An example of FSFG that cannot achieve IPB. (b) A rate-optimal FSFG using unfolding.

one iteration of the computation in the form of a recursive loop, unfolding by a factor f implies f consecutive iterations. If the original FSFG has N tasks, the f-unfolded FSFG has $f \times N$ tasks, and the IPB is f times larger than that of the original FSFG. Figure 2(b) illustrates the result of 2-unfolded FSFG in Fig. 2(a). In Fig. 2(b), the total number of delays, however, remains unchanged and precedence constraints are also not violated. The unfolding tchnique can obtain the rate-optimal static schedules.

2.3. *Scheduling of FSFG*

Before mapping an FSFG design into a hardware, the execution start time of each task must be determined. A *static* schedule of a cycle FSFG is a repeated pattern of an execution of the corresponding loop. And a static schedule must obey the precedence relations of the directed acyclic graph (DAG) portion of a FSFG design that is obtained by removing all edges with delays from that FSFG. A *sequencing graph* is a DAG $G_s(V, E)$, where vertex set $V = \{v_i \mid i = 1, 2, \ldots, n\}$ is in one-to-one correspondence with the set of the FSFG design, and edge set $E = \{(v_i, v_j) \mid i, j = 1, 2, \ldots, n; i \neq j\}$ is representing their dependencies. Different scheduling algorithms have been proposed in [5,8,30] addressing different constrained problems to find the desired schedule. The desired schedule have to satisfy the precedence constraints specified by the sequencing graph. A schedule S to the FSFG design is represented in space-time $(P \times T)$ domain. The *abscissa* denotes time axis, $[1, le(S)]$, where $le(S)$ is the length of the schedule. The *ordinate* denotes the processor space, $[1, n_{res}]$, where n_{res} is the total number of processors that implement each task. During the period of the ith iteration, the schedule determines the start times of all nodes in FSFG. Let op_j^i be a task which correspond to each vertex $v_j \in V$ of a given FSFG in

the ith iteration. A schedule of the given FSFG, $G_{FSFG} = (V, E, D)$, is a function $\varphi : V \to \mathbb{Z}^+$, which arranges each task node op_j^i to begin its execution at the time step $\varphi(op_j^i)$, where $\mathbb{Z}^+ = \{1, 2, \ldots\}$ is the positive integer.

Assuming d_j is the execution delay for each task node op_j^i, the length $le(S)$ of a schedule S is the latest finish time of all the operations scheduled, that is $le(S) = \max\{\varphi(op_j^i) + d_j - 1 | \forall op_j^i \in V\}$. For each task node $op_j^i \in V$, a schedule of the given FSFG is as follow:

- Start time: $t_j^i = \varphi(op_j^i),\ \varphi : V \to \mathbb{Z}^+ = \{1, 2, \ldots\}$
- Execution delay: $d_j \in \mathbb{Z} = \{0, 1, 2, \ldots\}$
- Finish time: $\varepsilon_j = \varphi(op_j^i) + d_j - 1$
- Task assignment:
  $pe_j^i = \tau(op_j^i), \tau : V \to \{1, 2, \ldots, n_{res}\}$
- Length of the schedule:
  $le(S) = \max\{\varphi(op_j^i) + d_j - 1 | \forall op_j^i \in V\}$
- The earliest task-finished step:
  $t_{etf} = \min\{\varphi(op_j^i) + d_j - 1 | \forall op_j^i \in V\}$

An example schedule of the second order IIR filter, for instance, is shown in Fig. 3.

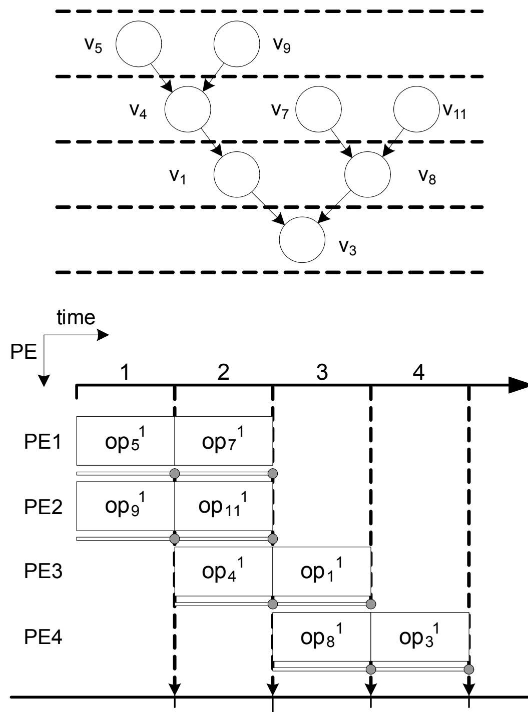

Fig. 3. An example schedule of the second order IIR filter of Fig. 1.

Int. J. Soft. Eng. Knowl. Eng. 2008.18:737-758. Downloaded from www.worldscientific.com by NATIONAL CHIAO TUNG UNIVERSITY on 04/25/14. For personal use only.

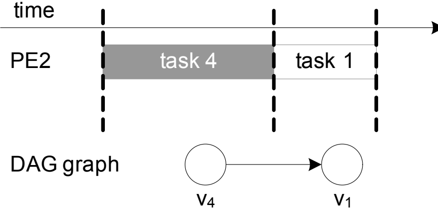

Fig. 4. Precedence property of operation nodes.

Int. J. Soft. Eng. Knowl. Eng. 2008.18:737-758. Downloaded from www.worldscientific.com
by NATIONAL CHIAO TUNG UNIVERSITY on 04/25/14. For personal use only.

2.4. *Admissible conditions for scheduling*

In HLS, designers may apply high-level transformation techniques on their original FSFG design and obtain the desired optimal or suboptimal schedule from its restructured FSFG. For all operation nodes in a FSFG design, the schedule determines the start time of each task. The executing order of all tasks in the schedule must satisfy the system specification, such as job completion, precedence and non-preemptive conditions. In the following sections, we will address these conditions. The false cause and the false detection for the schedule using CTL (computational tree logic) formulas are also discussed.

2.4.1. *Job completion condition*

Let f be the unfolding factor of the design. An admissible schedule must ensure that each operation node in vertex set V of the FSFG is scheduled exactly once. Thus, during the period of the ith-iteration of S, it must ensure that each operation node in vertex set V of the FSFG must be scheduled exactly f times during the length of S, that is:

$$t_j^i > 0, t_j^i = \varphi(op_j^i), 1 \leq i \leq f, \forall op_j^i \in V\,. \tag{1}$$

Job completion fault occurs when one or more operation nodes are not scheduled in S. For example, the start time of operation node is op_j^i is $t_j^i = 0$. Let p be the atomic proposition that op_j^i executes at the cth step of schedule S. The CTL formula of job completion property is represented by

$$EF(p)\,\&\,(p \rightarrow EF(\neg p))\,. \tag{2}$$

2.4.2. *Precedence condition*

Let DAG graph $G_s(V, E)$ be the scheduled sequencing graph of the given schedule S, where node set V represents operation nodes, and edge set E describes dependencies between the nodes. For each edge $e(op_j^i, op_k^i) \in E$, the precedence property ensures that operation op_j^i should be completed before operation op_k^i can start, that is:

$$t_k^i \geq t_j^i + d_j^i\,. \tag{3}$$

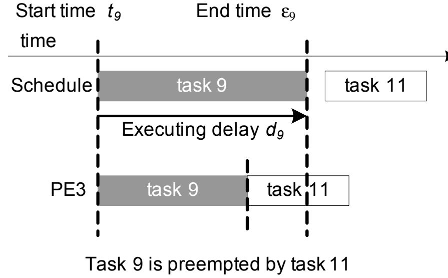

Fig. 5. Task 9 is preempted by operation task 11.

In Fig. 4, for instance, operator v_1 is a successor of operator v_4, thus, the execution order of these two operation nodes must ensure that $op_4 \rightarrow op_1$. Schedule S violates precedence property if these two operation nodes execute in reverse order. Let p and q be the atomic propositions that two operators op_j and op_k execute at some steps of schedule S respectively. And operator op_j is precedent to op_k. The CTL logic of precedence property is represented by

$$AG((p)\&(\neg q) \rightarrow AF((\neg p)\&(q)))\,. \tag{4}$$

2.4.3. *Non-preemption condition*

An admissible schedule must ensure that a computation is not preempted by another that is scheduled on the same processor at the same time. On the other hand, if the deterministic busy time of a single task op_j^i in the ith iteration is d_j^i, then for each time unit during its busy period, the same processor pe_j^i must execute that task, such that:

$$PE_r(u) = pe_j^i,\; pe_j^i = \tau(op_j^i),\; t_j^i \le u < t_j^i + d_j^i\,, \tag{5}$$

where $PE_r(u)$ is the assignment function for resource r, $0 < u \le le(S)$. In Fig. 5, for instance, the executing delay of task 9 is d_9. During executing interval d_9, task 9 is preempted by operation task 11. An admissible schedule must avoid such preemptive execution. Let p be the atomic proposition that task op_j^i executes during the period $t_j^i \le u < t_j^i + d_j^i$ using PE_r and q be the atomic proposition that each of the other task op_k^i uses the same resource PE_r during the period u. The CTL logic of non-preemptive property is represented by

$$AG((p)\,\&\,(\neg q))\,. \tag{6}$$

3. Proposed Verification Flow for HLS

3.1. *Verification flow*

Figure 6 shows the flowchart to illustrate our verification method. The inputs to the flow include a given schedule and the original FSFG graph. Before performing

Int. J. Soft. Eng. Knowl. Eng. 2008.18:737-758. Downloaded from www.worldscientific.com by NATIONAL CHIAO TUNG UNIVERSITY on 04/25/14. For personal use only.

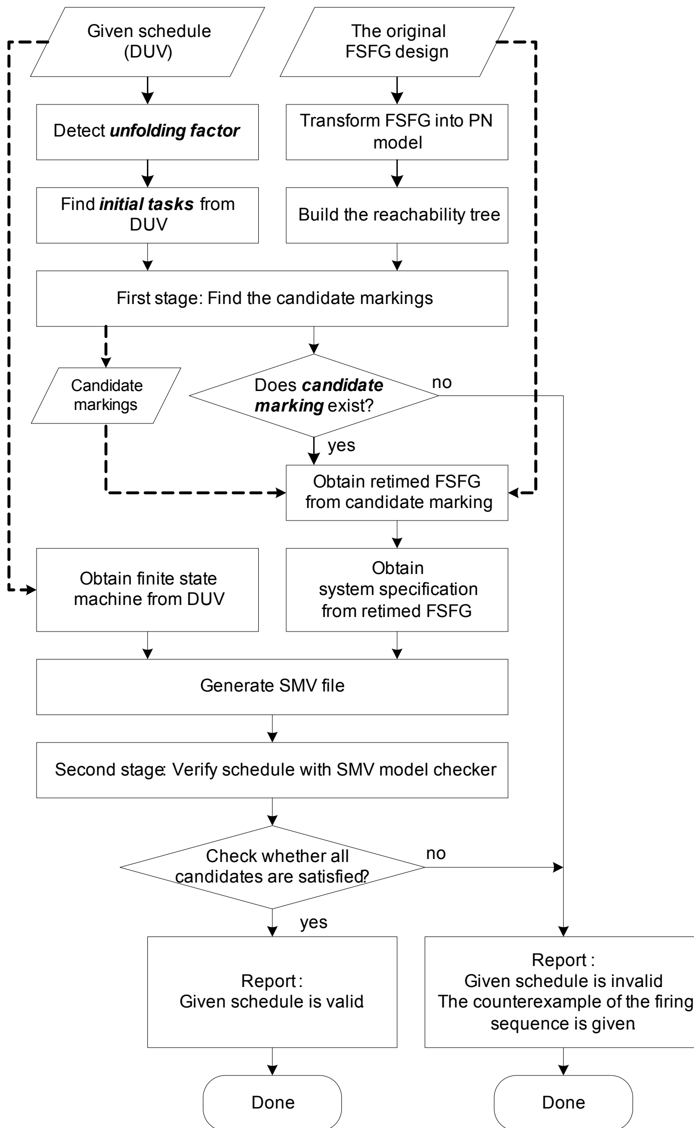

Fig. 6. Flowchart for the proposed high-level verification method.

two-stage verification method, the preprocessing on each input is applied separately. First, the given schedule is the DUV (design under verification) that needs to be verified. In system-level design flow, designers may use unfolding algorithm to pursue perfect FSFG achieving MASP on their original FSFG design. Usually, the FSFG of the DSP algorithm describes one iteration of the computation. By applying unfolding algorithm on the FSFG is to unfold the original FSFG by a factor f which implies f consecutive iterations of the design. In contrast, we perform

Int. J. Soft. Eng. Knowl. Eng. 2008.18:737-758. Downloaded from www.worldscientific.com by NATIONAL CHIAO TUNG UNIVERSITY on 04/25/14. For personal use only.

unfolding checking in our verification flow to detect the *unfolding factor* f from a given schedule. Another input to the flow is the original FSFG graph. It is transformed into a PN model by using proposed transformation. We will discuss the transformation from FSFG graph to PN model in Sec. 3.3.1.

The delay elements of the original FSFG design can be seen as the initial marking state of its transformed PN model. In PN domain, each marking state reached from the initial marking is a retimed version of its original design. Some of these markings, called the candidate markings, may be the correct restructured FSFGs for the given DUV. They can be found by using *initial tasks* obtained from the given schedule. After preprocessing, two-stage verification method is applied continuously.

At the first stage of the flow, we build the reachability tree rooted by initial marking. Let m be one of the markings in the tree. We use vector κ, which includes all *initial tasks* of the given schedule, to be the firing vector of PN. Marking m is said to be a *candidate marking* if and only if its result marking m', which is obtained by taking m and κ into PN matrix equation (9), is valid. A valid marking, m', also means that each element in m' is a non-negative integer. Two Breadth-First traverse algorithms are applied to find all candidate markings from the reachability tree at this stage. If there is no candidate marking found, the erroneous message is reported, since there does not exist any candidate marking leading all *initial tasks* valid. On the other hand, if there exists candidate marking, the flow continues verifying the schedule by using model checker.

At the second stage, we verify the DUV schedule by using SMV model checker. The inputs to the model checker include the behavioral description of FSM and the set of CTL formulas. FSM is directly obtained from a given schedule. CTL formulas which contain job-completion, precedence and non-preemptive properties, are generated according to the retimed FSFG corresponding to each candidate marking. If the FSM model satisfies all the CTL formulas, we say that the candidate marking is satisfied. The given schedule is said to be correct if and only if all the candidate markings are satisfied. If one or more than one of the candidate markings violates its CTL formulas, the erroneous message is reported and the counterexample of the schedule is given by the model checker. On the other hand, if all candidate markings satisfy all the CTL formulas, we say that the given schedule is valid.

3.2. *Relation between marking sets*

Since the nodes of reachability tree are exponential growth with the height of the tree, the policy to shorten the searching space is to find candidate markings to reduce the searching space at the first stage. And then, it verifies the schedule by checking the correctness of the candidate retimed FSFG at the second stage.

Assuming there are n operations in a given FSFG and n transitions in the corresponding PN model. Let f be the *unfolding factor* of a given schedule. At the first stage, the algorithm tries to find the candidate marking set from the reachable

Int. J. Soft. Eng. Knowl. Eng. 2008.18:737-758. Downloaded from www.worldscientific.com
by NATIONAL CHIAO TUNG UNIVERSITY on 04/25/14. For personal use only.

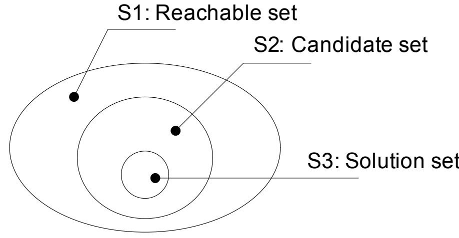

Fig. 7. The relation between reachable, candidate and solution marking sets.

Int. J. Soft. Eng. Knowl. Eng. 2008.18:737-758. Downloaded from www.worldscientific.com
by NATIONAL CHIAO TUNG UNIVERSITY on 04/25/14. For personal use only.

marking set in reachability tree and fires each transition once each time. The height of each node in the reachability tree is the distance from the root node to itself. Since, during one iteration period of the schedule S, $le(S)$, each scheduled task must be fired once, the height can also be seen as the number of transitions that have been fired from the root node. Thus, for an n-tasks schedule, the *upper height-bound* of the reachability tree is bounded by $H_{up} = f \times n$. At the second stage, it continually finds the solution marking set from the candidate marking set. The set relation between three marking sets is shown in Fig. 7, that is $S3 \subseteq S2 \subseteq S1$. The purpose of the first stage is trying to reduce the searching space from reachable marking set $S1$ to candidate marking set $S2$, while the second stage is trying to find solution marking set $S3$ from candidate marking set $S2$.

3.3. *Petri net modeling*

A Petri Net $G_{PN}(P, T, W, M)$ is a four-tuple [31], where $P = \{p_1, \ldots, p_n\}$ and $T = \{t_1, \ldots, t_m\}$ are finite sets of places and transitions, W is the weighted flow relation, and $M : P \to \mathbb{Z}$ is a marking function. If there are k tokens in place p_i, it is represented $M(p_i) = k$ for place p_i. If $W(u, v) > 0$, then there is an arc from u to v with weight $W(u, v)$. For a node u in $P \cup T$, ${}^\bullet u$ (the pre-set of u) is specified by: ${}^\bullet u = \{v \in P \cup T | W(v, u) > 0\}$ and $u^\bullet$ (the post-set of u) is specified by: $u^\bullet = \{v \in P \cup T | W(u, v) > 0\}$. A PN can *execute* by firing *enabled* transitions. A transition t is *enabled* at marking M (denoted by $M[t\rangle$) if $\forall p \in {}^\bullet t : M(p) \geqslant W(p, t)$. Once a transition t is enabled at a marking M, it may fire and then reach a new marking M' (denoted by $M[t\rangle M'$). The occurrence of t leads to a new marking M', defined for each place p by

$$M'(p) = M(p) - W(p, t) + W(t, p)\,. \tag{7}$$

Usually, matrix representation gives a complete characterization of Petri Net. The characteristic matrix of PN is defined by *incidence matrix* A (also called the characteristic matrix), which is a $|P| \times |T|$-matrix with entities

$$A_{ij} = W(t_j, p_i) - W(p_i, t_j)\,. \tag{8}$$

Marking m_0 is an $|P| \times 1$ column vector with entities $m_0(i) = M(p_i)$, $\forall p_i \in P$. We say that m_0 is a *valid marking* if and only if $m_0(p_i) \geqslant 0$, $\forall p_i \in P$. Let $x_j =$

$\{t_j\} = (\ldots, 0, 1, 0, \ldots)$ be a $|T| \times 1$ column vector, which is zero everywhere except in the jth element. Transition t_j can be represented by the column vector x_j. We say that t_j is *enabled* at a marking m_0 (denoted by $m_0[t_j\rangle$) if $m_0 \geqslant A \cdot x_j$ for every element of m_0 is a non-negative integer. And the result m' of firing enabled transition t_j in a marking m_0 is represented by

$$m' = m_0 + A \cdot x_j \, . \tag{9}$$

3.3.1. *Transformation from FSFG to PN model*

The FSFG is attractive to algorithm developers because it directly models the equations of DSP algorithm. Yet, it does not sufficiently unveil the dynamical behavior and the implementation limits in terms of the degree of parallelism and the memory requirement. Thus, we use Petri Net to model DSP algorithms. It allows us to discover the characteristic of the target architecture and to observe the dynamical behavior of the algorithm also. The FSFG $G_{FSFG}(V, E, D)$ of a DSP algorithm can be modeled as PN $G_{PN}(P, T, W, M_0)$ by applying the following rules:

(1) Functional element set V is transformed into the transition set T, whose elements have computational power.
(2) The edge set E is transformed into the place set P denoting the system states.
(3) Since each place in PN has only one output, the pseudo transition of each fork edge will be added as source duplicators.
(4) The delay element set D in FSFG domain correspond to the number of tokens in place in PN domain. In static analysis, tokens can be represented as delay elements of FSFG, thus moving tokens between places in PN model can be seen as retiming delay elements between edges in FSFG. In dynamical analysis, moving tokens between places in PN model represents the executions of vertex elements in FSFG.

Figure 8, for instance, illustrates the transformation from vertex set V and edge set E to transition set T and place set P. The vertex set $V = \{v_1, v_2\}$ and the edge

Int. J. Soft. Eng. Knowl. Eng. 2008.18:737-758. Downloaded from www.worldscientific.com by NATIONAL CHIAO TUNG UNIVERSITY on 04/25/14. For personal use only.

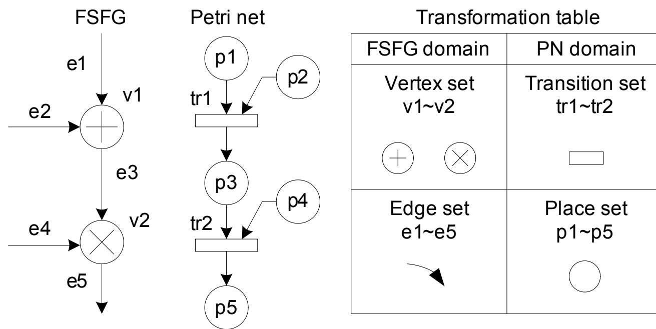

Fig. 8. Transformation from FSFG to Petri Net.

Int. J. Soft. Eng. Knowl. Eng. 2008.18:737-758. Downloaded from www.worldscientific.com
by NATIONAL CHIAO TUNG UNIVERSITY on 04/25/14. For personal use only.

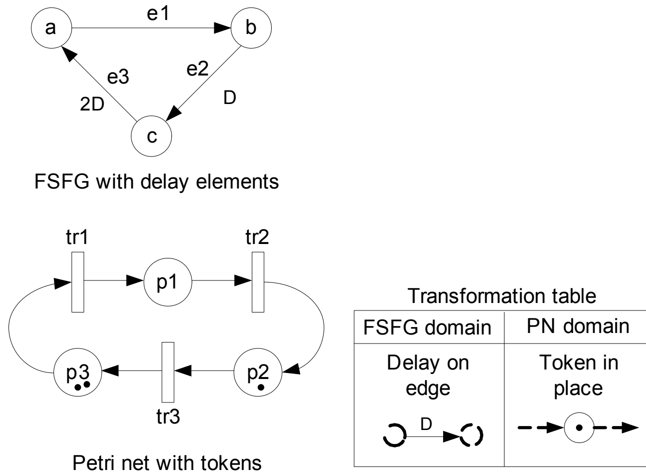

Fig. 9. FSFG with delay elements and Petri Net with tokens.

set $E = \{e_1, e_2, e_3, e_4, e_5\}$ in FSFG domain are transformed into the transition set $T = \{tr_1, tr_2\}$ and the place set $P = \{p_1, p_2, p_3, p_4, p_5\}$ with respect to PN domain. Another example is given in Fig. 9, a FSFG graph with delay elements is transformed into a Petri Net. The vertex set $V = \{a, b, c\}$ and the edge set $E = \{e_1, e_2, e_3\}$ of FSFG are transformed into the transition set $T = \{tr_1, tr_2, tr_3\}$ and the place set $P = \{p_1, p_2, p_3\}$ of PN model. The delay elements in FSFG domain is denoted by the number of tokens, such that $M_0(p_1) = 0, M_0(p_2) = 1$ and $M_0(p_3) = 2$.

In FSFG domain, a computing result of a functional element is one or more other functional elements' inputs. Data source in the prior functional element causes a data fork point. A fork point in FSFG can be modeled as a pseudo-transition in PN model. The pseudo-transition duplicates copies of data source as many of the output nodes in FSFG graph. The equivalence graph of fork point in FSFG domain and pseudo-transition in PN domain is shown in Fig. 10. Another example illustrating the PN model of the second order IIR filter of Fig. 1 by applying the above transformation rules is shown in Fig. 11.

3.4. *The candidate marking*

Candidate marking set is defined as a subset of reachable marking set of a Petri Net. The candidate markings are probably the correct initial markings that lead to the firing sequence of a given schedule being valid. Let S be a schedule of a FSFG. The earliest task-finished set *est_set* of S are the tasks which are finished at the earliest task-finished step t_{est} in S, such that

$$est_set = \{op_i | \varepsilon_i = t_{est},\ \forall op_i \in V\}\,. \tag{10}$$

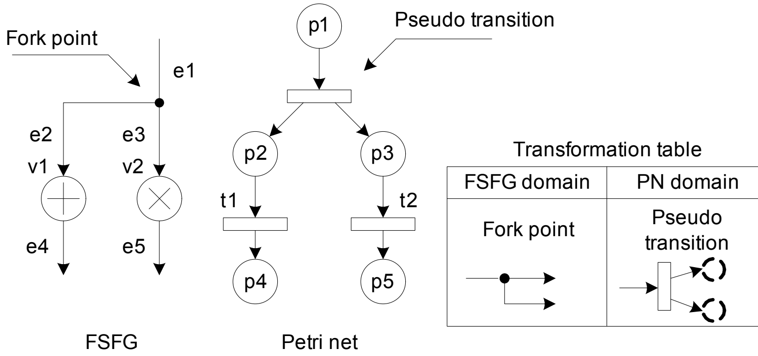

Fig. 10. Fork point in FSFG domain and pseudo-transition in PN domain.

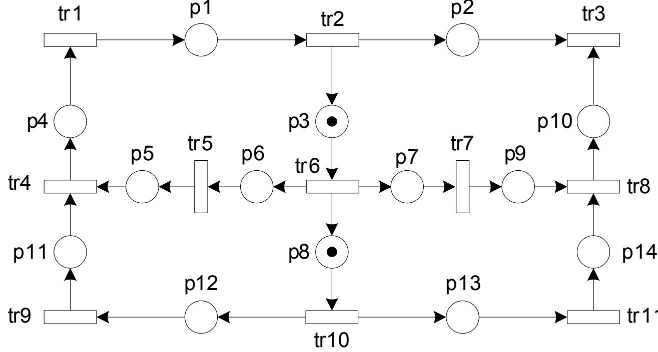

Fig. 11. PN models of the second order IIR filter.

Int. J. Soft. Eng. Knowl. Eng. 2008.18:737-758. Downloaded from www.worldscientific.com
by NATIONAL CHIAO TUNG UNIVERSITY on 04/25/14. For personal use only.

4. Verification Algorithms

4.1. *First stage: Two approaches for candidate search*

At first stage, two approaches including the early-terminated and the optimal methods are proposed. We will discuss this in the following sections.

4.1.1. *The early-terminated approach*

The first approach is the early-terminated traverse method. Before introducing early-terminated traverse algorithm, we first consider Lemma 1.

Lemma 1. *Let T_{tree} be a reachability tree which is bounded by upper height-bound H_{up} and m_1 be any one of the candidate markings in T_{tree}. For any other candidate marking m_2 in the successor path of marking m_1, m_2 is in the solution marking set $S3$ if and only if m_1 is in $S3$.*

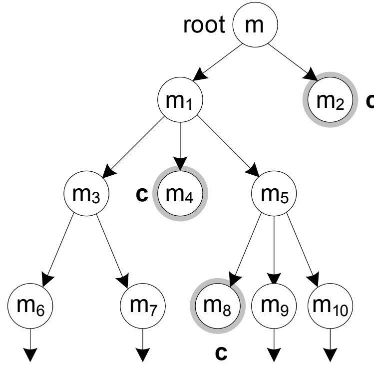

Fig. 12. The traverse order of early-terminated approach.

Int. J. Soft. Eng. Knowl. Eng. 2008.18:737-758. Downloaded from www.worldscientific.com
by NATIONAL CHIAO TUNG UNIVERSITY on 04/25/14. For personal use only.

Proof. Let etf_set be the earliest task-finished set of a given schedule and transition sequence σ_1 be a firing sequence that leads $m_1 \in S2$ from root marking m_r of T_{tree} to be a candidate marking, that is $m_r \xrightarrow{\sigma_1} m_1$. Assuming there exists another candidate marking $m_2 \in S3$, $m_2 \neq m_1$, with firing sequence σ_2 that leads m_2 from root marking m_r of T_{tree} to be a candidate marking, that is $m_r \xrightarrow{\sigma_2} m_2$, and is in the successor path of marking m_1.

As defined in Definition 1, it must be satisfied that $etf_set \subseteq \sigma_1$ and $etf_set \subseteq \sigma_2$ where the elements of σ_1 and σ_2 are all in $\{nop\} \cup \{etf_set\}$. As described in the assumption, m_2 is in the successor path of marking m_1, it is still satisfied that $\sigma_2 = \sigma_1 \cup \{nop\}$. This implies m_2 is in solution marking set $S3$ if and only if m_1 is in $S3$. □

The early-terminated approach tries to minimize the size of candidate set $S2$ from reachable set $S1$. When an enqueued unvisited marking is a candidate, the early-terminated algorithm ignores the candidate marking and marks it as a visited node. Then, it proceeds to other unvisited nodes in queue Q until all the markings have been visited. In Fig. 12, as an example, the traverse order of the early-terminated approach is $m, m_1, m_2, m_3, \ldots, m_{10}$. The pseudo-code of the earliest-terminated traverse method is shown in Fig. 13. In lines 12 to 14, it ignores the candidate marking and proceeds to other unvisited nodes.

4.1.2. *The optimal approach*

The second approach to verify a schedule is the optimal approach which is improved from the early-terminated approach. In order to reduce reachable marking set $S1$, it tries to merge the redundant nodes when it proceeds to Breadth-First traverse.

Let m be an unvisited node to be processed. If m is a candidate marking, it ignores this node by using Lemma 1 and proceeds to other unvisited nodes in the queue. If m is not a candidate marking, it finds enabled set of transitions and creates

```
1: procedure BFS_BUILD_TREE_EARLY_TERMINATED(m_0)
2:     Initialize Queue structure Q
3:     Allocate new node nn                                   ▷ initialize root node
4:     nn.visited ← false
5:     nn.marking ← m_0
6:     nn.height ← 0
7:     nn.candidate ← IS_CANDIDATE(nn)
8:     Q.ENQUEUE(nn)
9:
10:    for all unvisited node n ∈ Q do
11:        n.visit ← true
12:        if n.candidate = true then
13:            continue to the next node                      ▷ Lemma 1
14:        end if
15:        if n.height ⩽ H then
16:            ebl_set ← FIND_ENABLED_TRANS(n.marking)
17:            for all transition tr ∈ ebl_set do
18:                m ← n.marking
19:                κ ← MAKE_FIRING_VECTOR_FROM(tr)
20:                m' ← m + [A] · κ                           ▷ Eq.9
21:                Allocate new node nn
22:                nn.marking ← m'
23:                nn.visited ← false
24:                nn.height ← n.height + 1
25:                nn.candidate ← IS_CANDIDATE(nn)
26:                CREATE_BRANCH(n, nn)
27:                Q.ENQUEUE(nn)
28:            end for
29:        end if
30:    end for
31: end procedure
```

Fig. 13. The early-terminated approach.

Int. J. Soft. Eng. Knowl. Eng. 2008.18:737-758. Downloaded from www.worldscientific.com by NATIONAL CHIAO TUNG UNIVERSITY on 04/25/14. For personal use only.

new node on each enabled transition. For each new node produced with marking m', if there exists another node in the reachability tree, and has the same marking associated with it, then the node with marking m' is a duplicate node. Since, the marking m' has appeared in the tree, this new node produced is redundant. Then, it merges this redundant node to the existential node and creates transition link from marking m to the existential node. As an example in Fig. 14, when it proceeds marking m_5, it finds that the newly created node with marking m_7 is a duplicate node. It merges these nodes and creates transition from m_5 to m_7. Then, the algorithm continually proceeds to other unvisited nodes in the queue.

The pseudo-code of the optimal approach is shown in Fig. 15. In lines 12 to 14, if the node n is a candidate marking, then it ignores this node by using Lemma 1 and proceeds to the other nodes in queue Q. In lines 26 to 32, function $find_dual_node$ checks whether the new created node nn is a duplicate. If there exists a duplicate node, it either returns the dual node to *dual_node* or returns *null*. It continually proceeds to other unvisited nodes until all nodes are visited.

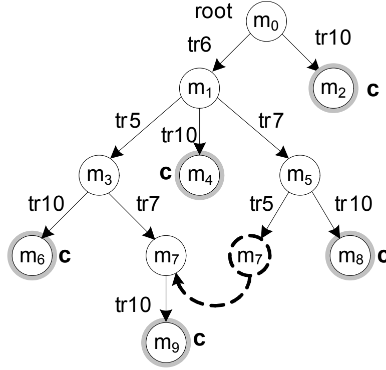

Fig. 14. Merge the redundant nodes in optimal approach.

```
 1: procedure BFS_BUILD_TREE_OPTIMAL(m0)
 2:     Initialize Queue structure Q
 3:     Allocate new node nn                                ▷ initialize root node
 4:     nn.visited ← false
 5:     nn.marking ← m0
 6:     nn.height ← 0
 7:     nn.candidate ← IS_CANDIDATE(nn)
 8:     Q.ENQUEUE(nn)
 9:
10:     for all unvisited node n ∈ Q do
11:         n.visit ← true
12:         if n.candidate = true then
13:             continue to the next node                   ▷ Lemma 1
14:         end if
15:         if n.height ⩽ H then                            ▷ max. height H
16:             ebl_set ← FIND_ENABLED_TRANS(n.marking)
17:             for all transition tr ∈ ebl_set do
18:                 m ← n.marking
19:                 κ ← MAKE_FIRING_VECTOR_FROM(tr)
20:                 m′ ← m + [A] · κ                        ▷ Eq.9
21:                 Allocate new node nn
22:                 nn.marking ← m′
23:                 nn.visited ← false
24:                 nn.height ← n.height + 1
25:                 nn.candidate ← IS_CANDIDATE(nn)
26:                 dual_node ← FIND_DUAL_NODE(nn)
27:                 if dual_node ≠ null then
28:                     CREATE_BRANCH(n, dual_node)
29:                 else
30:                     CREATE_BRANCH(n, nn)
31:                     Q.ENQUEUE(nn)
32:                 end if
33:             end for
34:         end if
35:     end for
36: end procedure
```

Fig. 15. The optimal traverse approach.

Int. J. Soft. Eng. Knowl. Eng. 2008.18:737-758. Downloaded from www.worldscientific.com
by NATIONAL CHIAO TUNG UNIVERSITY on 04/25/14. For personal use only.

4.2. *Second stage: The model-checking approach*

At the second stage, we verify the given DUV schedule on all candidate markings aided by using SMV model checker. The inputs to the model checker are a FSM and a set of system specification that needs to be verified. FSM of the DUV is automatically generated by the verification flow. Each candidate marking obtained from the first stage can be seen as a restructured FSFG. System specification, CTL formulas, to the DUV is generated from each restructured FSFG.

Schedule S can be converted into a Moore machine. A Moore machine is defined as a 6-tuple $M(W, \Sigma, \Delta, \delta, \lambda, w_0)$, where W, Σ and Δ are the finite set of state, input signal and output signal respectively. Transition function $\delta : W \times \Sigma \to W$ is a mapping function from state and an input to the next state. Output function $\lambda : W \to \Delta$ is a mapping function from each state to the output signal. w_0 is an initial state. Figure 16 is a DUV schedule of a third-order IIR filter. The length of

Int. J. Soft. Eng. Knowl. Eng. 2008.18:737-758. Downloaded from www.worldscientific.com by NATIONAL CHIAO TUNG UNIVERSITY on 04/25/14. For personal use only.

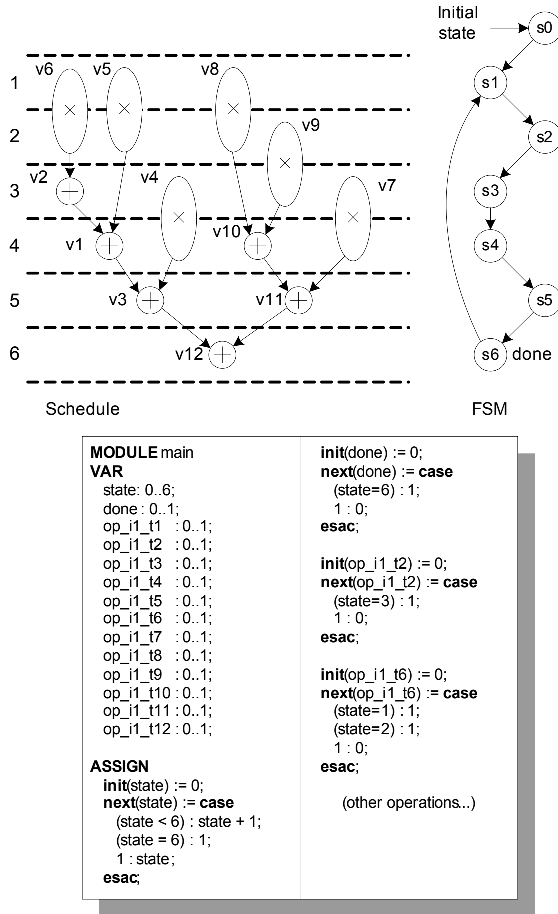

Fig. 16. The behavioral description of FSM for the third-order IIR filter.

Int. J. Soft. Eng. Knowl. Eng. 2008.18:737-758. Downloaded from www.worldscientific.com
by NATIONAL CHIAO TUNG UNIVERSITY on 04/25/14. For personal use only.

schedule S is $le(S) = 6$, and it can be seen as a FSM with $le(S) + 1$ states, where s_0 is the initial state. Each state of the FSM is actually represented as a control step in S. For each task ${op_j}^i$ in S, there is an enabled signal *op_i_j* on it to indicate whether ${op_j}^i$ is enabled in each control step of S. For example, task op_6^1 is scheduled at step 1 to step 2, and let *op_i_j* be its enabled signal. The behavioral description of FSM for task ${op_6}^1$ is described at the bottom of Fig. 16.

5. The Complexity Analysis

Assuming there are n operations in a given FSFG, let f be the *unfolding factor* of a given schedule. Thus, the upper-height of the reachability tree of the corresponding PN model is bounded by $H_{up} = f \times n$.

At the first stage, there are two approaches to perform the *Breadth-First* traverse procedure. Considering exhaustive search, each node of the reachability tree has n enabled transitions in worse case. Then, the total number of nodes is:

$$1 + n + n^2 + \cdots + n^{f \cdot n} = (n^{f \cdot n+1} - 1)/(n-1)\,. \tag{11}$$

In the first approach, i.e. the early-terminated approach, the algorithm stops traversing a node while it is a candidate. Let p, $p \leq (f \cdot n)$, be the deepest level that *Breadth-First* traverse procedure can reach. Thus, the complexity of the first approach is $O(N^p), p \leq f \cdot n$.

In the second approach, i.e. the optimal approach, the algorithm merges duplicate markings in order to reduce the reachable marking set of the reachability tree. Let $x \in \mathbb{Z} = \{1, 2, \ldots\}$ be the merging radio in the reachability tree. The complexity of these two approaches is $O((N/x)^p), p \leq f \cdot n$.

Thus, the relation of the complexity between two approaches is:

$$O(N^p) > O((N/x)^p)\,. \tag{12}$$

At the second stage, the verification flow performs SMV model checking to verify a given schedule by checking the precedence, job completion and non-preemptive properties on given DUV. Let Ω be the size of the FSM converted from DUV, Ψ be the size of the CTL formulas, $le(S)$ be the steps of DUV schedule and c be the number of candidate markings. The complexity of the model checker in the second stage is about $O(\Omega \times \Psi \times le(S) \times c)$, in worse case.

6. Experimental Results

We have implemented these two approaches in our study. Each of these approaches is applied to several dataflow algorithms. Figure 17 shows the statistics of these designs.

In Fig. 17, design *iir2d-sch1* to *iir3d-sch2* [10] are the second and third Infinite Impulse Response filters. Design *p243* [10] is a design with *unfolding factor* 6, the lengths of schedule *p243-sch1* and *p243-sch2* are both 96 steps. Design *ewf-sch1* and *ewf-sch2* are low power schedules for the Elliptic Wave Filter in [2].

Example	#vertices	#edges	Size of PN (Places x Trans.)	Schedule length	Unfolding factor
iir2d-sch1	8	14	(14 x 11)	6	1
iir2d-sch2	8	14	(14 x 11)	4	1
iir2d-sch3	8	14	(14 x 11)	4	1
iir2d-sch4	8	14	(14 x 11)	4	1
iir3d-sch1	12	21	(21 x 16)	90	1
iir3d-sch2	12	21	(21 x 16)	6	1
p243-sch1	5	7	(7 x 5)	96	6
p243-sch2	5	7	(7 x 5)	96	6
ewf-sch1	34	47	(47 x 34)	40	1
ewf-sch2	34	47	(47 x 34)	40	1

Fig. 17. The statistics of test designs.

Test schedule	Early-terminated			Optimal		
	Time (sec)	Res. usage	No. of candidates	Time (sec)	Res. usage	No. of candidates
Iir2d-sch1	93.81	44972	1005	1.01	376	9
Iir2d-sch2	94.27	44972	1005	1.02	376	9
Iir2d-sch3	148.82	114867	635	0.84	319	8
Iir2d-sch4	1610.29	49502	15486	1.85	237	21
Iir3d-sch1	N/A	N/A	N/A	28.58	24676	166
Iir3d-sch2	N/A	N/A	N/A	77.22	22613	335
p243-sch1	0.84	2	1	0.76	2	1
p243-sch2	0.81	2	1	0.8	2	1
ewf-sch1	0.66	2	1	0.64	2	1
ewf-sch2	0.73	2	1	0.62	2	1

Fig. 18. The experimental results.

Figure 18 shows the experimental results of using two approaches. There are two columns on each approach including execution time in seconds and the resource usage in unit number of allocated nodes of the reachability tree. According to experimental results, early-terminated approach suffers from the state explosion problem while it traverses the reachability tree in *iir3d-sch1* and *iir3d-sch2*. On average, optimal approach takes more benefit than early-terminated approach. According to experimental results, the optimal approach outperforms the others in terms of time and resource usage in average.

7. Conclusion

This paper aims to exploit formal verification techniques for high-level synthesis. In the top-down design flow, design errors should be removed as early as possible; otherwise, errors detected at the later stages will result a costly, time-consuming redesign cycles. Although formal verification for logic synthesis has been studied very

Int. J. Soft. Eng. Knowl. Eng. 2008.18:737-758. Downloaded from www.worldscientific.com
by NATIONAL CHIAO TUNG UNIVERSITY on 04/25/14. For personal use only.

extensively, little work has been done for high-level synthesis. The paper presents a verification flow that can efficiently detect the design errors from the results of high-level synthesis. As shown in the experimental results, we can apply the optimal approach for the first phase to efficiently verify complex design cases.

Acknowledgments

This work was supported by the National Science Council, ROC, under Grant No. NSC 94-2220-E-009-039.

Int. J. Soft. Eng. Knowl. Eng. 2008.18:737-758. Downloaded from www.worldscientific.com
by NATIONAL CHIAO TUNG UNIVERSITY on 04/25/14. For personal use only.

References

1. V. K. Madisetti and B. A. Curtis, A quantitative methodology for rapid prototyping and high-level synthesis of signal processing algorithms, *IEEE Trans. Signal Processing* **42**(11) (1994) 3188–3208.
2. L.-R. Dung and H.-C. Yang, On multiple-voltage high-level synthesis using algorithmic transformations, *IEICE Trans. Fundamentals* (2004).
3. K. Ito, L. E. Lucke, and K. K. Parhi, Ilp-based cost-optimal dsp synthesis with module selection and data format conversion, *IEEE Trans. Very Large Integration Systems* **6**(4) (1998) 582–594.
4. K. K. Parhi, High-level algorithm and architecture transformations for dsp synthesis, *J. VLSI Signal Processing* **9** (1995) 121–143.
5. L.-F. Chao and E. H.-M. Sha, Scheduling data-flow graphs via retiming and unfolding, *IEEE Trans. Parallel and Distributed Systems* **8**(12) (1997) 1259–1267.
6. X. Liu, M. C. Papaefthymiou, and E. G. Friedman, Retiming and clock scheduling for digital circuit optimization, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems* **21**(2) (2002) 184–203.
7. G. A. Constantinides, P. Y. K. Cheung, and W. Luk, Optimum and heuristic synthesis of multiple word-length architectures, *IEEE Trans. Very Large Scale Integration Systems* **13**(1) (2005) 39–57.
8. K. Parhi and D. Messerschmitt, Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding, *IEEE Trans. Computers* **40**(2) (1991) 178–195.
9. K. Parhi, Algorithm transformations for concurrent processors, in *Proc. IEEE* **77**(12) (1989) 1879–1895.
10. V. K. Madisetti, *VLSI Digital Signal Processors* (IEEE Press, 1995).
11. R. E. Bryant, Symbolic Boolean manipulation with ordered binary-decision diagrams, *ACM Computing Surveys* **24**(3) (1992) 293–318.
12. K. S. Brace, R. L. Rudell, and R. E. Bryant, Efficient implementation of a BDD package, in *ACM/IEEE Design Automation Conf.*, 1990, pp. 40–45.
13. K. L. McMillan, Applying SAT methods in unbounded symbolic model checking, in *Proc. 14th Conf. on Computer Aided Verification*, 2002, pp. 250–264.
14. G. Parthasarathy, K.-T. Cheng, and C.-Y. Huang, An analysis of ATPG and SAT algorithms for formal verification, in *Proc. Int. High Level Design Validation and Test Workshop*, 2001, pp. 177–182.
15. J. Burch, E. Clarke, and D. Long, Symbolic model checking with partitioned transition relations, in *Int. Conf. Very Large Scale Integration*, 1991, pp. 49–58.
16. J. Burch, E. Clarke, D. Long, K. MacMillan, and D. Dill, Symbolic model checking for sequential circuit verification, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems* **13**(4) (1994) 401–424.

17. H.-J. Kang and I.-C. Park, SAT-based unbounded symbolic model checking, in *Proc. 40th Conf. on Design Automation*, 2003, pp. 840–843.
18. G. Parthasarathy, M. K. Iyer, K.-T. Cheng, and L.-C. Wang, Safety property verification using sequential SAT and bounded model checking, *IEEE Design and Test of Computers* **21**(2) (2004).
19. J. Kljaich, B. T. Smith, and A. S. Wojcik, Formal verification of fault tolerance using theorem-proving techniques, *IEEE Trans. Computers* **38**(3) (1989) 366–376.
20. P. Ashar, S. Bhattacharya, A. Raghunathan, and A. Mukaiyama, Verification of RTL generated from scheduled behavior in a high-level synthesis flow, in *Proc. 1998 IEEE/ACM Int. Conf. Computer-Aided Design*, 1998, pp. 517–524.
21. D. Sarkar, Register transfer operation analysis during data path verification, in *Proc. 2002 Conf. Asia South Pacific Design Automation/VLSI Design*, 2002, p. 172.
22. C. Karfa, C. Mandal, D. Sarkar, S. R. Pentakota, and C. Reade, A formal verification method of scheduling in high-level synthesis, in *Proc. 7th Int. Symp. on Quality Electronic Design*, 2006, pp. 71–78.
23. D. Borrione, J. Dushina, and L. Pierre, A compositional model for the functional verification of high-level synthesis results, *IEEE Trans. VLSI Systems* (2000) 526–530.
24. N. Mansouri and R. Vemuri, Automated correctness condition generation for formal verification of synthesized RTL designs, *J. Formal Methods in System Design* **16**(1) (2000).
25. C. Bolchini, R. Montandon, F. Salice, and D. Sciuto, Design of VHDL-based totally self-checking finite-state machine and data-path descriptions, *IEEE Trans. Very Large Scale Integration* (*VLSI*) *Systems* **8**(1) (2000) 98–103.
26. D. D. Gajski and L. Ramachandran, Introduction to high-level synthesis, *IEEE Design and Test* **11**(4) (1994) 44–54.
27. C. Kern and M. Greenstreet, Formal verification in hardware design: A survey, *ACM Trans. Design Automation of E. Systems* **4** (1999) 123–193.
28. E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking* (MIT Press, 1999).
29. T. Barnwell, C. Hodges, and M. Randolph, Optimum implementation of single time index signal flow graphs on synceronous multiprocessor machines, *IEEE Int. Conf. Acoustics, Speech, and Signal Processing* (*ICASSP '82*), Vol. 7, May 1982, pp. 679–682.
30. C. Hwang, J. Lee, and Y. Hsu, A formal approach to the scheduling problem in high level synthesis, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems* **10** (1991) 464–475.
31. W. Reisig and G. Rozenberg, *Lectures on Petri Nets I: Basic Models* (Springer-Verlag, 1998).

Int. J. Soft. Eng. Knowl. Eng. 2008.18:737-758. Downloaded from www.worldscientific.com by NATIONAL CHIAO TUNG UNIVERSITY on 04/25/14. For personal use only.