Pipeline Optimization using a Cost Extension of Timed Petri Nets

Rémi Parrot, Mikaël Briday, Olivier H Roux

Pipeline Optimization using a Cost Extension of Timed Petri Nets

Rémi Parrot
École Centrale de Nantes
LS2N, UMR CNRS 6004
Nantes, France

Mikaël Briday
École Centrale de Nantes
LS2N, UMR CNRS 6004
Nantes, France

Olivier H. Roux
École Centrale de Nantes
LS2N, UMR CNRS 6004
Nantes, France

***Abstract*—A major step in arithmetic operators design is the placement of pipeline stages, with the goal of drastically increase the data throughput.**

Approaches, such as the *as-soon-as-possible greedy* algorithm, allow pipelining with a frequency target. They can possibly be combined with a *retiming* operation to reduce the number of pipeline registers. This retiming step is based on a weighted directed graph model, from which the pipeline placement is reduced to an optimisation problem (for example ILP). However, this approach produces only a unique solution, and makes it difficult to add additional constraints on the resulting pipeline.

We propose to use a Timed Petri Net extension with cost, where time captures the propagation delay and cost measures the size of pipeline registers. The state space of the model captures exactly the circuit states and the branching points, so its exploration can be guided by comparing the circuit states regarding any feature (number and size of registers, critical path, throughput, etc). The pipeline exploration can be reduced to a weighted branching-time logic model-checking problem, that we prove to be PSPACE-complete on this model.

We have implemented this exploration algorithm in a prototype tool. We apply it on some arithmetic operators provided by FloPoCo showing improvements up to 35% compared to the current implementation.

***Index Terms*—pipeline optimisation, arithmetic operator, synchronous circuit, Timed Petri Net**

I. Introduction

The pipeline is an essential step in the design of arithmetic operators to significantly increase the operating frequency and therefore the throughput, at the cost of an increased latency.

The work presented in this paper takes place just before the synthesis phase, when an unpipelined circuit is available. The circuit is composed of different operators which are combinatorial or sequential entities that are not modified (replacement of a constant multiplier by a *shift-and-add* implementation for example). We focus on the automatic generation of a pipeline, *i.e.* the efficient placement of the flip-flops that guarantee a target frequency, but also minimize the resources consumed by the pipeline. As the target applications are the arithmetic operators, we restrict ourselves on synchronous, dataflow circuits with unfoldable loops.

This work is supported by the Renault-Centrale Nantes chair dedicated to the propulsion performance of electric vehicles.

Automatic pipeline

Automatic pipeline generation was initially formalized by Leiserson and Saxe in [1], using a graph theoric model. This solution is based on *retiming*, *i.e.* moving registers from one place to another to change the timings of the circuit but without altering the functionality. Using *retiming*, it is possible to build a pipeline ensuring a minimal throughput, and minimizing the resources consumed by the registers [1]. However, this solution is complex to implement, and has a level of abstraction that makes it very difficult to add additional constraints on the produced pipeline.

An *as-soon-as-possible greedy algorithm* has been implemented by Istoan and De Dinechin in [2] for the FloPoCo arithmetic core generator framework [3]. This method adds operators as early in the pipeline as possible, until the maximum target period is reached. It then goes to the next pipeline stage and starts the operation again. This approach is very fast to execute, and guarantees a target frequency. However, it does not necessarily compute the optimal pipeline in terms of resource consumption.

This result can be improved by using retiming, but we want to go further, by allowing additional constraints to be added to the pipeline. For example, it can be useful for time-multiplexing to ensure that two signals are never produced simultaneously, or at least at the same pipeline stage. Therefore, we propose a new approach based on a formal model: the Petri Net.

Petri Nets for circuit design

As introduced in [1], a circuit can be abstracted by a weighted directed graph. In fact, the intuition behind this model is a Marked Graph (also called event graph) which is a subclass of Petri Net where each place has one incoming arc, and one outgoing arc. Due to their concurrency nature, Petri Net have been extensively used to analyse and optimise timing properties of both synchronous and asynchronous circuits [4]–[7].

It has proven to be particularly effective in Latency-insensitive systems, since such architectures have been proposed [8]. Bufistov et al. [4] extend the work of Leiserson and Saxe on Latency-insensitive systems, by combining retiming and *recycling*, *i.e.* inserting bubbles (registers without

informative value), in order to reduce the total number of registers while providing a minimum throughput. More recently, Josipovic et al. [9] propose a timing optimization of circuits generated by HLS with *control-flow* structures, by extracting *choice-free* sub-circuit and applying to them the approach of Bufistov et al. [4].

Advances have also been made on asynchronous circuits, through *slack matching*, *i.e.* inserting buffer register to prevent stalls. Najibi et al. [6] focus their work on mode-based conditional asynchronous circuits, where the modes switching have given probabilities. They break the slack matching problem of such systems into Markov chains for the mode switching, and Petri Net for the buffer placement.

All those works share the same method of resolution: deduce the timing constraints from the Petri Net structure, and get back to an Integer Linear Problem. In contrast, we propose to explore the states of the circuit using directly the semantics of the model. The main interest is that the model states capture exactly the circuit states, so the exploration can be guided online by comparing the circuit states regarding any feature (number and size of registers, critical path, throughput, etc).

Formal model of a circuit

We propose to use the formal model of [10] extending Timed Marked Graph, capturing the behaviour of the circuit, the propagation delay and the number of pipeline registers.

The two main time extensions of Petri Nets are Time Petri Nets [11] and Timed Petri Nets [12]. While a transition can be fired within a given interval for Time Petri Nets, deterministic (or constant) "firing times" are assigned to transitions of Timed Petri Nets.

In a Timed Petri Net, transitions are fired according to the maximal-step rule [13], *i.e.* in each marking a maximal set of firable transitions fires at once. This semantics can be ambiguous in the case of conflict between transitions ; for example for 2 simultaneous transitions with the same duration and the same input place such that the firing of one disabled the other one. However, in a Timed Marked Graph, since every place has only one outgoing arc, there can not be any conflict. Hence, Timed Marked Graph, where transitions are fired according to the maximal-step firing rule, is a well-fitting model for logical circuits and their timing properties.

In the literature, the optimal-cost problem has been addressed for Priced Timed Petri nets in [14] and Cost Time Petri Nets [15] where the rate cost of a place p is the sojourn cost (per time unit) of each token in place p.

Contribution

We first recall the extension of Timed Marked Graph proposed in [10], with delayable transitions and a particular *reset* transition, that allows to model the flip-flop (register) placement. The state space of this model will give most of possible pipelines of the circuit. Moreover, we extend this model with a function giving a cost to each marking, and thus, allowing to compute the cumulated cost of the states after each reset transition, and to compare the different runs. We give the translation rules allowing to build a model from a circuit, and some heuristics for a smart placement of the delayable transitions. From the model, we give a state-space exploration algorithm aiming to minimize this cost function. The approach is fully implemented and uses propagation delay estimations provided by FloPoCo.

An informal definition of Cost Reset Timed Petri Net is provided in Section II. Section III gives the translation rules from a pipelining problem to a Cost Reset Timed Petri Net. We present the state-space exploration algorithm in Section IV and we present the experimental results in Section V. Conclusion and future works are presented in Section VI.

II. A MODEL FOR PIPELINED SYNCHRONOUS CIRCUITS

$\mathbb{N}$ and $\mathbb{R}_{\geq 0}$ are respectively the sets of integer and non-negative real numbers. For vectors of size n, the usual operators $+, -, \times, <, \leq, >, \geq$ and $=$ are used on vectors of $\mathbb{N}^n$ and $\mathbb{R}^n_{\geq 0}$ and are the point-wise extensions of their counterparts in $\mathbb{N}$ and $\mathbb{R}_{\geq 0}$. Let $\bar{\mathbf{0}}$ be the null vector of size n.

A. Timed Petri Net with reset and delayable transitions

An extension of Timed Petri Nets with a *reset* action (RTPN) has been proposed in [10] for pipelined synchronous circuit design. The authors give a formal definition of this model, decidability and complexity results and an efficient state space abstraction algorithm.

Informally, with each transition of the Net is associated a clock and a delay. The clock measures the time since the transition has been enabled and the delay is interpreted as a firing condition: the transition may and must fire if the value of its clock is equal to the delay. Moreover, some transitions are delayable and may fire if the value of its clock is greater or equal to the delay. But, a fired set of transitions must always contain at least one transition whose clock is equal to its delay. Finally, the clocks can be reset (let $reset$ be the corresponding action) and the delay between two successive resets is set within an interval I_{reset}.

Formally:

Definition 1 (RTPN): A Timed Petri Net with reset and delayable transitions is a tuple $(P, T, {}^\bullet(.), (.)^\bullet, \delta, I_{reset}, M_0)$ defined by:

- $P = \{p_1, p_2, \ldots, p_m\}$ is a non-empty set of places;
- $T = \{t_1, t_2, \ldots, t_n\}$ is a non-empty set of transitions;
- $T_D \subseteq T$ is the set of delayable transitions;
- ${}^\bullet(.) : T \to \mathbb{N}^P$ is the backward incidence function;
- $(.)^\bullet : T \to \mathbb{N}^P$ is the forward incidence function;
- $M_0 \in \mathbb{N}^P$ is the initial marking of the Petri Net;
- $\delta : T \to \mathbb{N}$ is the function giving the firing times (delays) of transitions;
- I_{reset} is the reset time interval with lower ($I^{\uparrow}_{reset}$) and upper ($I^{\downarrow}_{reset}$) bounds in $\mathbb{N}$.

A marking M is an element of $\mathbb{N}^P$ such that $\forall p \in P$, $M(p)$ is the number of tokens in place p.

Transitions are fired according to the maximal-step firing rule (among non-delayable firable transitions), and thus must

fire simultaneously. From a marking M, the simultaneous firing of a set $\tau \subseteq T$ of transitions leads to a marking $M' = M + \Sigma_{t\in\tau}(t^\bullet - {}^\bullet t)$.

A marking M enables a transition $t \in T$ if $M \geq {}^\bullet t$. A transition t' is said to be *newly* enabled by the firing of a set of transitions τ, if $M+\Sigma_{t\in\tau}(t^\bullet - {}^\bullet t)$ enables t' and $(M-\Sigma_{t\in\tau}\ {}^\bullet t)$ does not enable t'. If t remains enabled after its firing then t is newly enabled.

A state is a pair (M, v) where M is a marking and $v \in \mathbb{R}_{\geq 0}^{T\cup\{reset\}}$ is a time valuation of the system, *i.e.* the value of the clocks. $v(t)$ is the time elapsed since the transition $t \in T$ has been newly enabled. $v(reset)$ is the time elapsed since the last reset. $\bar{\mathbf{0}}$ is the valuation assigning 0 to every transition and $reset$. The initial state of the RTPN is $q_0 = (M_0, \bar{\mathbf{0}})$.

A non-delayable (resp. delayable) transition is firable if it is enabled and its clock is equal (resp. greater or equal) to its delay. A maximal-step is a set of transitions $\tau \subseteq T$, which contains all the non-delayable transitions that can be fired simultaneously from a given state. In a Marked Graph where every place has one incoming arc, and one outgoing arc, there can not be conflict and the firing of a transition cannot disable another transition. But in the general case, there can be conflicts, and thus there can be several maximal-steps τ from a given state. The firing of a maximal-step $\tau \subseteq T$ from a state (M, v) is denoted $(M, v) \xrightarrow{\tau} (M', v')$. It leads to the new marking $M' = M + \Sigma_{t\in\tau}(t^\bullet - {}^\bullet t)$, and reset the clocks of all newly enabled transitions. τ must contain at least one transition t such that $v(t) = \delta(t)$ (its clock is equal to its delay), therefore we prevent infinite waiting when there is a delayable transition.

A waiting of a delay $d \in \mathbb{R}_{\geq 0}$ from a state (M, v) is denoted $(M, v) \xrightarrow{d} (M, v')$. It leads to the new valuation v' such that for all enabled transitions t, $v'(t) = v(t)+d$. A delay transition is possible only if the clock of all non-delayable transition $t \in T \setminus T_D$ does not exceed its delay $v'(t) \leq \delta(t)$, and the clock of the $reset$ does not exceed the upper bound of I_{reset}, $v'(t) \leq I_{reset}^{\downarrow}$.

Finally, the $reset$ action from (M, v) is denoted $(M, v) \xrightarrow{reset} (M, v')$. It leads to v' such that for all $t \in T$, $v'(t) = 0$. It is possible when the clock of the reset is in the reset time interval $v(reset) \in I_{reset}$.

The firing of maximal-steps and the reset action are called discrete transitions, whereas the waiting is called delay transition. A run of a RTPN is a possibly infinite sequence $\rho = q_0 \xrightarrow{d_1} q_{d_1} \xrightarrow{\tau_1} q_{\tau_1} \ldots \xrightarrow{d_n} q_{d_n} \xrightarrow{\tau_n} q_{\tau_n}$ of alternating d_i delay (possibly null) and τ_i discrete transition where either $\tau_i \subseteq T$ or $\tau_i = \{reset\}$. Let a run $\rho = q_0 \xrightarrow{\alpha_1} q_1 \ldots \xrightarrow{\alpha_n} q_n$, we write $\rho' = \rho \xrightarrow{\alpha_{n+1}} q_{n+1}$ if $\rho' = q_0 \xrightarrow{\alpha_1} q_1 \ldots \xrightarrow{\alpha_n} q_n \xrightarrow{\alpha_{n+1}} q_{n+1}$ is a run of the RTPN .

B. Graphical representation and example

A Petri net is a directed bipartite graph, in which the transitions are represented by boxes (or bars), places are represented by circles and backward forward incidence functions (pre and post conditions) of transitions are represented by arrows. Moreover, we use the following notations : the delay of a transition is in red and a delayable transition is in blue (the values in green will be used for a further cost extension).

Let us consider the RTPN of Fig. 1 (notice that this particular example is a Marked Graph). A state is a pair (M, v). To simplify the notation we will note a marking as a set of marked places instead of a vector and we will give the valuation v only for the enabled transitions. For example, in the initial state, we have tokens in places p_0 and p_1, and only two enabled transitions t_0 and t_1 then the initial state is noted $q_0 = \begin{array}{l}\{p_0, p_1\}\\ v(t_0) = 0\\ v(t_1) = 0\\ v(reset) = 0\end{array}$

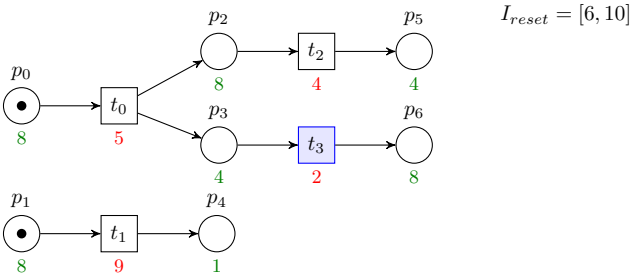


Fig. 1: A Cost Reset Timed Petri Net example

According to the semantics, the runs of the RTPN of Fig. 1 between the initial state q_0 until the occurrence of the reset action are given in Fig. 2. We do not draw all the possible reset actions here, because whatever the delay before the reset, it always leads to the same state. But in the general case, a reset can be done at any time, so there exists an infinite number of runs.

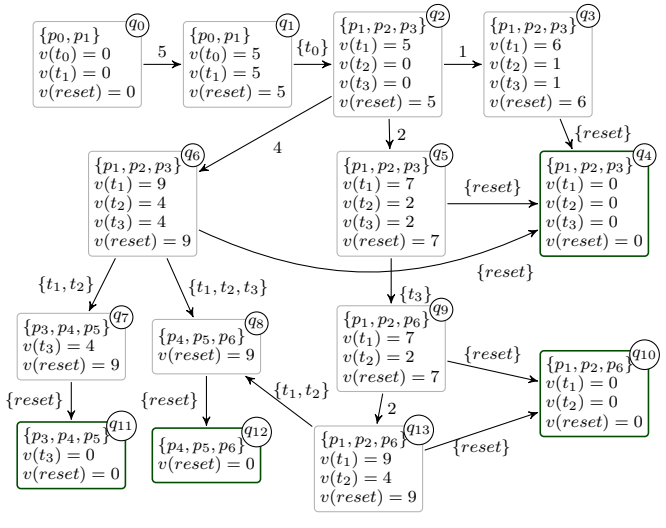


Fig. 2: State graph of RTPN of Fig. 1 from initial state to the first occurrence of reset (green framed states).

Notice that, as the transition t_3 is delayable, it can be fired either alone when $v(t_3) = 2$ (in q_5), or it can wait to be fired with t_1 and t_2 when $v(t_3) = 4$ (from q_6). But when its delay

is missed $v(t_3) > \delta(t_3)$, and no other transition can be fired at its delay, then t_3 can no more be fired (until the next reset). This would have been the case, if we have decided to elapse time from q_7.

C. Cost Reset Timed Petri Net

Cost Reset Timed Petri Net extends RTPN with a cost associated with each place and a marking cost function.

Definition 2 (CRTPN): A Cost Reset Timed Petri Net is a tuple $(\mathcal{N}, \mathcal{C}, \omega)$ where:

- $\mathcal{N} = (P, T, {}^\bullet(.), (.)^\bullet, \delta, I_{reset}, M_0)$ is a RTPN;
- $\mathcal{C}\colon P \to \mathbb{N}$ is the place cost function;
- $\omega\colon \mathbb{N}^P \to \mathbb{N}$ is the marking cost function (recall that a marking $M \in \mathbb{N}^P$).

In the case of one-bounded (safe) Petri Net, the marking of a place $M(p)$ can only take its value in $\{0, 1\}$, which can be interpreted both as a boolean and as an integer value. Therefore, in the following we will use both arithmetical operators (in $\{+, *\}$) and the logical *or* operator $\vee$, in the definition of the marking cost function ω.

For example, let $\omega(M) = (M(p_1) \vee M(p_2)) * 4 + M(p_2) * 10$, and assume that $M_1(p_1) = M_1(p_2) = 1$, then $\omega(M_1) = (1 \vee 1) * 4 + 1 * 10 = 14$.

A classical marking cost function is $\omega(M) = \sum_{p \in P} M(p) * \mathcal{C}(p)$ which is the sum of marked places weighted by their cost.

Definition 3 (Cost of a run): The cost $\Omega(\rho)$ of a run ρ is the cumulated marking cost of the states after each $reset$ transition over the run, starting with the cost of the initial marking. It is inductively defined on a run $\rho_n = \rho_{n-1} \xrightarrow{\alpha_n} q_n$, with $q_n = (M_n, v_n)$ by:

- $\Omega(q_0) = \omega(M_0)$
- $\Omega(\rho_n) = \begin{cases} \Omega(\rho_{n-1}) + \omega(M_n) & \text{if } \alpha_n = \{reset\} \\ \Omega(\rho_{n-1}) & \text{otherwise} \end{cases}$

Let now suppose that the example of Fig. 1 is a CRTPN, where the cost associated to each place is labelled in green. If we consider the marking cost function $\omega(M) = \sum_{p \in P} M(p) * \mathcal{C}(p)$, then the minimal cost following a reset is for the marking $M_1 = \{p_3, p_4, p_5\}$ since $\omega(M_1) = 4+1+4 = 9$. This marking is obtained in state q_{11} of Fig. 2, by firing the transitions t_0, t_1 and t_2. The cost of the marking $M_2 = \{p_4, p_5, p_6\}$ (state q_{12}) where all the transitions are fired is $\omega(M_2) = 1 + 4 + 8 = 13$.

D. Complexity of model-checking problem

Temporal logics were introduced by Pnueli [16] as specification languages to express the behaviours of sequential and concurrent programs. Since the early 1990s, classical temporal logics have been extended with timing constraints. While temporal logics only express constraints on the order of events, their timed extensions can add quantitative constraints on delays between those events. TCTL, introduced in [17], is a real-time extension of the branching-time temporal logic CTL and Weighted CTL [18] extends CTL with cost-constrained modalities.

In the sequel we only consider bounded Nets.

Theorem 1: Model checking of Weighted CTL for bounded CRTPN is PSPACE-complete.

Proof: PSPACE-hardness: Reachability for bounded timeless Petri Nets with the interleaving semantics is PSPACE-hard [19], and so for the maximal-step semantics (we can simulate the interleaving semantics by adding a self loop from all places of the net). The class of Timeless Petri Nets with maximal-step semantics is simulated by CRTPN with $T_D = \emptyset$, $\forall t \in T'$, $\delta(t) = 0$ and $I^{\uparrow}_{reset} > 0$ then reachability is PSPACE-hard for CRTPN, and so is the Weighted CTL model checking.

PSPACE membership: For RTPN (as for CRTPN), the differences between all valuations $v(t)$ of transition t can be break into simple diagonal constraints $v(reset) - v(t) = c$ where $c \in \mathbb{N}$ and $0 \leq c \leq I^{\downarrow}_{reset}$ (as illustrated in section II-E), meaning that the transition t has been enabled c time units after the last $reset$. In other words, there exists a fixed offset between $v(reset)$ and $v(t)$, and therefore we can define a timed abstraction fully defined by the value of $v(reset)$. Hence, we can apply the PSPACE algorithm proposed in [18] for model-checking Weighted CTL on one clock Priced Timed Automata showing that Weighted CTL is PSPACE under "single clock" assumption. ■

The WCTL allows us to verify properties such that, the reachability of a given marking constraint $Goal$, with a cost lower or equal than k, which is written $EF_{\leq k}(Goal)$. As an illustration in our specific pipeline problem, consider, in the Fig. 1, the property $EF_{\leq 9}(M(p_3) = M(p_4) = M(p_5) = 1)$. This reachability property is verified by the run $\rho = q_0 \to q_1 \to q_2 \to q_6 \to q_7 \to q_{11}$ of Fig. 2. The full grammar of WCTL is even more expressive, and is defined in [18].

Although the WCTL is very powerful, it is not well suited for the search of optimal. For this reason, in section IV we define a proper algorithm to directly look for the minimal cost.

E. Symbolic state space

A timed abstraction based on simple diagonal constraints $v(reset) - v(t) = c$ allows to construct an abstract state graph of the model. The elapsing of time is captured and included in the symbolic states and the edges of the graph correspond to the discrete firings of transitions. For example, the states q_2, q_3, q_5 and q_6 of the state graph of Figure 2 belong to the same following symbolic state: $S_2 = \begin{cases} \{p_1, p_2, p_3\} \\ v(reset) - v(t_1) = 0 \\ v(reset) - v(t_2) = 5 \\ v(reset) - v(t_3) = 5 \\ 5 \leq v(reset) \leq 9 \end{cases}$

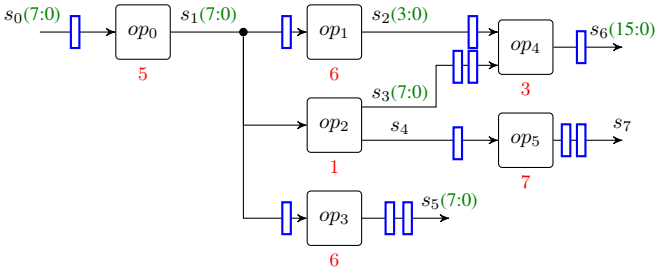
In practice, this symbolic abstraction is much more efficient than the regions one and our algorithm is based on it. One can notice that a firable set of transitions τ contains at least one transition $t \in \tau$, such that $v(t) = \delta(t)$, with $\delta(t) \in \mathbb{N}$. Then, the firing of transitions is done from an integer point, *i.e.* a valuation that assign an integer value to each transition. In this paper, for the sake of clarity, instead of symbolic state graph, we represent the elapsing of time explicitly as in Figure 2.

III. From a circuit to a Cost Reset Timed Petri Net

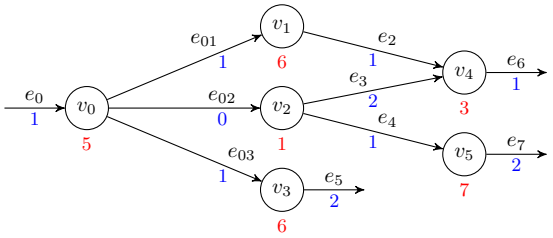
This section presents the translation rules required to model a circuit with a Cost Reset Timed Petri Net. The historical

model of Leiserson and Saxe [1] is recalled as an element of comparison.

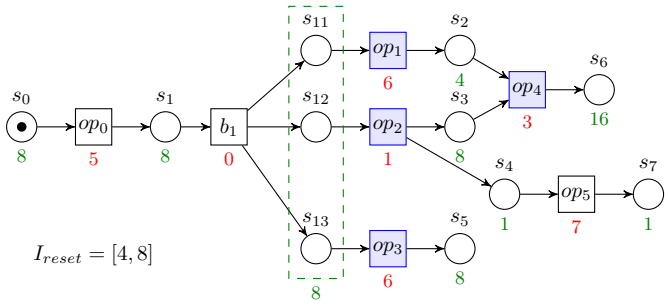
An example of circuit is presented on Fig. 3a, involving some operators op_i and some signals s_j transmitted by connections (data size in bits depicted in green). The pipeline of this circuit is already available and pipeline registers are shown with blue rectangles. The *propagation delay* of each operator is shown in red.



(a) Pipelined circuit (with frequency $f \geq \frac{1}{8}$)



(b) Leiserson and Saxe model



(c) Cost Reset Timed Petri Net

Fig. 3: A pipelined circuit example.

Leiserson and Saxe formalize a circuit abstraction, using a weighted directed graph, where the vertices are basically the *atomic functional elements* (the *operators*) and the edges are the *connections* in between. An illustration of this model is given in Fig. 3b, where each operator has an associated propagation delays (in red), and each connection has an associated *numbers of registers* (in blue). It does not take into account the size of the signals, but a workaround is to add as many edges in parallel as the data size (not drawn in the figure).

As stated in the introduction, this model is in fact a Marked Graph, and many works have shown its utility for solving pipelining relating problems [4]–[7]. We therefore propose to use the extension of Marked Graph presented in section II.

In this model, nodes represent both operators and branching points and the weight of edges represent the size of signals. Then, we first define a circuit as a weighted directed graph $G = (V, E, d, w)$ where the set $V = Op \cup B$ of nodes is the union of the set Op of operators with the set B of branching points and the set $E \subseteq V^2$ of directed edges is the set of signals. For each operator $op \in Op$, $d(op)$ is the propagation delay of op. For each signal $s \in E$, $w(s)$ is the size of the signal in bits (in green). For an edge $s = (v, v') \in E$, we say that s is an outgoing signal of v and an incoming signal of v'. The incoming signal s of a branching point b and all its outgoing signals have the same size $w(s)$.

CRTPN is fundamentally different from a weighted directed graph as it doesn't hold the fully pipelined circuit in its state but only one stage at a time. A complete pipeline is built from a run of the model, *i.e.* a path in the state graph.

Then, such approach allows to build a pipeline from a non-pipelined circuit. Moreover, it has the advantage to permits the comparison between the pipeline stages on-the-fly when they are created.

Translation rules

The CRTPN $((P, T, {}^\bullet(.), (.)^\bullet, \delta, I_{reset}, M_0), \mathcal{C}, \omega)$ produced from the circuit of Fig. 3a, is represented on Fig. 3c and is obtained using 7 translation rules.

The first four rules ensure the preservation of all the elements of the circuit and their interconnections:

rule 1: $\exists \phi_s \colon E \mapsto P$ a bijection, with $\forall s \in E, \mathcal{C}(\phi_s(s)) = w(s)$;
rule 2: $\exists \phi_v \colon V \mapsto T$ a bijection, with $\forall op \in Op$, $\delta(\phi_v(op)) = d(op)$ and $\forall b \in B$, $\delta(\phi_v(b)) = 0$;
$T = T_{Op} \uplus T_B$ with $T_{Op} = \phi_v(Op)$ and $T_B = \phi_v(B)$.
rule 3: If $s \in E$ is an incoming signal of $v \in V$, then ${}^\bullet t(p) = 1$ with $t = \phi_v(v)$ and $p = \phi_s(s)$;
rule 4: If $s \in E$ is an outgoing signal of $v \in V$, then $t^\bullet(p) = 1$ with $t = \phi_v(v)$ and $p = \phi_s(s)$;

A signal and its data size are respectively represented by a place and its associated cost. An operator and its propagation delay are respectively represented by a transition and its firing time. As well, each branching point corresponds to a transition with a zero firing time (b_1 on the figure). Its purpose is to allow one stage of pipeline to be added either prior to branching (s_1), or only on some specific output branches (s_{11}, s_{12} or s_{13}). Rules 3 and 4 define the incidences of the transitions so that the network structure is preserved.

All input signals are considered synchronized, which is equivalent to have them all on the first stage of the pipeline. In the model it conforms with the initial marking M_0 and is related to the 5^{th} rule:

rule 5: If s is an input signal (not outgoing from any operator), then $M_0(p) = 1$ with $p = \phi_s(s)$;

The $reset$ operation corresponds to a transition of a pipeline stage and resets the clocks of the CRTPN for the next stage. Rule 6 defines the reset interval maximum bound:

rule 6: $I^{\downarrow}_{reset} = \frac{1}{f}$;

The time elapsed since the last reset is contained in $v(reset)$. Semantics imposes that a reset can only occur if $v(reset) \in I_{reset}$, and therefore if the maximum bound is set to $\frac{1}{f}$, then

the pipeline produced can run at least at frequency f. Here $\frac{1}{f}$, and in the following $\frac{1}{2f}$, are supposed to be in $\mathbb{N}$, but they can be rational without changing the results of section II.

The cost function gives the total number of flip-flops implemented in the current stage of pipeline:

rule 7: We define $P_{Op} = \{p \in P \mid \exists t \in T_{Op}, t^\bullet(p) = 1\}$ and $P_B(p) = \{p' \in P \mid \exists t \in T_B, {}^\bullet t(p) = 1$ and $t^\bullet(p') = 1\}$. Then $\forall M \in \{0,1\}^P$, $\omega(M) = \sum_{p \in P_{Op}} \mathcal{C}(p) \cdot (M(p) \vee \bigvee_{p' \in P_B(p)} M(p'))$.

Indeed the cost coefficient of a place is the size of the signal, thus it is the number of flip-flops needed. The calculation of the cost takes into account the particular case of branching points, and the possible mutualisation of registers at the output of a branching point. In Fig. 3c, the register before the operators op_1 and op_3 can easily be shared. This is represented with a dotted green box in which places s_{11} to s_{13} share their cost, as it models signals outgoing from the same branching point. This explains why the cost of the places after a branching point transition t following the place p is $\mathcal{C}(p) \cdot \bigvee_{p' \in P_B(p)} M(p')$.

Construction heuristics

The CRTPN produced with the translation rules previously presented, in which all the transitions are set delayable, is able to find the optimal pipeline, *i.e.* the pipeline minimizing the resources while ensuring the operative frequency f. However in practice, it leads to an explosion of the state space. Thus, we provide heuristics of construction, handling some crucial points to save resources, while limiting the size of the state space.

Delayable transitions are a feature allowing to relax the constraints on the model, and therefore to explore more states. We propose two usages of delayable transitions.

heuristic 1: $\forall t \in T_{Op}$, if $\sum_{p \in P} \mathcal{C}(p) \cdot {}^\bullet t(p) < \sum_{p \in P} \mathcal{C}(p) \cdot t^\bullet(p)$, then $t \in T_D$;

When an operator has a larger bus width at the output than at the input, then we may want to put the pipeline stage before. To explore this possibility, setting its corresponding transition *delayable* allows the exploration of a pipeline keeping the register before. In Fig. 3c, operators op_2 and op_4, are translated into delayable transitions, because of the size of their input/output signals.

heuristic 2: $\forall t \in T_{Op}$, if $\exists t_B \in T_B, p \in P$ such that ${}^\bullet t_B(p) = {}^\bullet t(p) = 1$, then $t \in T_D$;

The operators following a branching point can mutualise their registers (as they use the same signal), in order to save resources. Their corresponding transitions are therefore set delayable to allow keeping the register before. The operators op_1, op_2 and op_3 are translated into delayable transitions, in Fig. 3c, allowing the mutualisation of register.

The reset interval offers flexibility over a fixed value and shorter pipeline stages can be defined to allow exploration of other configurations. However, if the stages are too short, this increases the number of stages (and thus the cost in registers). The lower bound of the reset interval can be settles as follows:

heuristic 3: $I_{reset} = \left[\frac{1}{2f}, \frac{1}{f}\right]$

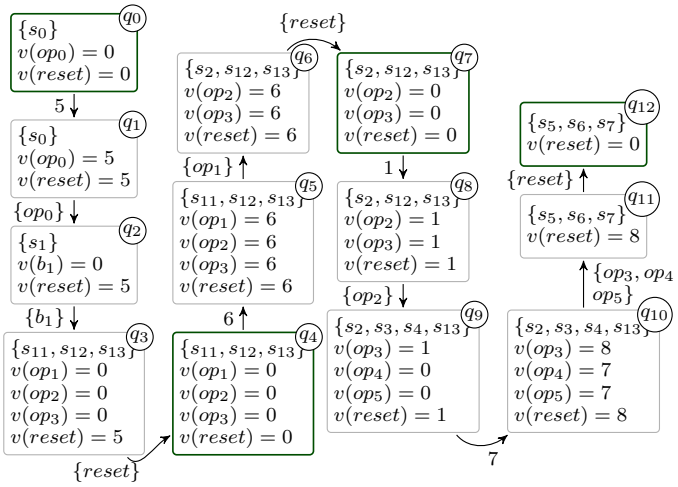
The intuition behind this interval can be seen as the Shannon sampling theorem where the sampling correspond to the reset: if we do a reset before $\frac{1}{2f}$ there will be overlapping of states, *i.e.* we will visit the same state (with one more reset) two times. In any case, this trade-off produces good results while limiting the combinatorial explosion.

IV. Pipeline exploration

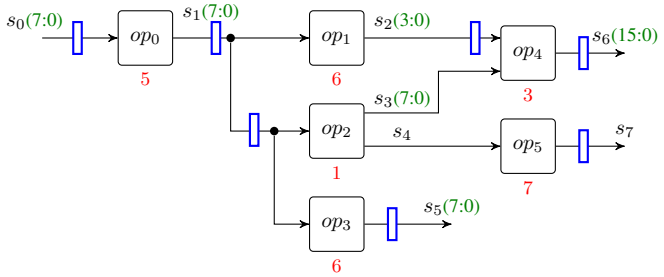
This section shows how to generate pipeline configurations that meet the minimum frequency constraint based on a CRTPN model of the circuit.

The CRTPN's semantics ensures the synchronisation of signals. Indeed, a transition is enabled only when all its predecessor places have a token and an operator propagation delay is counted as soon as all its input signals are available. Thus, each reachable state of the model represents a possible pipeline stage of the real circuit.

A $reset$ operation defines a transition from one pipeline stage to the next one. The full pipeline is retrieved by a walk along a branching of the state graph, collecting $reset$ operations.



(a) State graph part for one run of the Cost Reset Timed Petri Net of Fig. 3c. States after a $reset$ are framed in green (q_0, q_4, q_7 and q_{12})



(b) One possible pipeline of the circuit of Fig. 3a

Fig. 4: Example of the extraction of a pipeline from a run.

One run ρ of the CRTPN of Fig. 3c, is represented on Fig. 4a. It is the best run achievable by the model, *i.e.* the one that minimizes the cost.

The corresponding pipeline on the circuit is presented on Fig. 4b. The marking of every state after a $reset$ (framed in dark green in Fig. 4a) gives the position of the registers in the circuit. Although, if all the signals outgoing from a branching point are marked, then only one register is needed for the unique signal that they represent. For example, at state q_4, the marking is $M_4 = \{s_{11}, s_{12}, s_{13}\}$, but only one register (8 DFFs) is required before the branch.

Let $q_i = (M_i, v_i)$ $(0 \leq i \leq 12)$ be the states of this run ρ. The run cost is calculated for each $reset$ along the run: $\Omega(\rho) = \omega(M_0) + \omega(M_4) + \omega(M_7) + \omega(M_{12}) = (\mathcal{C}(s_0)) + (\mathcal{C}(s_1)) + (\mathcal{C}(s_1) + \mathcal{C}(s_2)) + (\mathcal{C}(s_5) + \mathcal{C}(s_6) + \mathcal{C}(s_7)) = 53$. This cost matches with the number of flip-flops in the pipeline of Fig. 4a. Note that on this example, the greedy algorithm produces the result in Fig. 3a, with a total of 79 flip-flops (49% more registers are required).

The interest of the heuristic which defines the delayable transitions just after a branching point is highlighted in the state q_7 where the signals s_{12} and s_{13} can mutualise the same pipeline registers. If the transition related to the operator op_3 has not been delayable, the register would have been placed after it.

The algorithm used to build the state graph is classical, built around a waiting list and a list of elements already visited. At each step, the algorithm pops one state out of the waiting list, adds it to the visited list, computes all its successors not yet visited, and adds them to the waiting list. It then loops until the waiting list is empty.

To reduce computation time, the algorithm prunes the state graph exploration. If it reaches a state already reached — *i.e.* a state with the same marking and the same valuation already exists in the state graph — then it compares their run cost so far, and it keeps the new state only if it has the lowest cost. By removing runs that reach the same point in the state graph, but at an additional cost, it is ensured that no optimal run is pruned.

V. Implementation and Experiments

FloPoCo is a tool to generate circuits for floating point arithmetic operators. It splits an operator into numerous elementary operators and each operator's delay is estimated by heuristics. For the experiments, we rely on FloPoCo version 5 (non-stable version from git) to generate a pipeline that minimizes the number of flip-flops while ensuring a target frequency for these arithmetic operators.

The pipeline algorithm in FloPoCo is a greedy algorithm (explained in [2]). This algorithm associates (c, τ) to each signal production date, where c is an integer that counts the number of stages and τ is a real number that represents the critical delay since the last pipeline register. When a new signal is handled by the algorithm, the following formula is applied[1]:

$$(c, \tau) + \delta = \left(c + \left\lfloor \frac{\tau + \delta}{\delta_{obj}} \right\rfloor, \delta_{obj} \cdot \left(\frac{\tau + \delta}{\delta_{obj}} - \left\lfloor \frac{\tau + \delta}{\delta_{obj}} \right\rfloor \right) \right)$$

[1]There is a typo in the original paper, section III.D

TABLE I: Results of both greedy and the Cost Reset Timed Petri Net based algorithms on some floating point operators generated by FloPoCo. Operator parameters are input data sizes in bits (exponent, mantissa). *(s,del)* gives the number of states analysed and the number of delayable transitions. When both heuristic 1 and 2 are enabled, the number of delayable transition is limited manually.

		FPAdd(8,23) 500 MHz	FPMult(8,23) 500 MHz	FPDiv(8,23) 500 MHz	FPSqrt(8,23) 500 MHz
circuit size (ops, signals)		(108,165)	(151,237)	(116,197)	(189,316)
Pipeline stages		16	7	30	25
Greedy	Time (s)	0.01	0.01	0.01	0.01
	Nb FF	2080	671	3480	2085
CRTPN	Time (s)	0.06	0.02	0.10	0.06
(without	states analysed	418	110	1711	215
delayable)	Nb FF	1999	671	3182	2082
Improvement (%)		**3.9%**	**0.0%**	**8.6%**	**0.1%**
CRTPN	Time (s)	1.35	0.47	0.44	4.12
(with	s - del	8368 - 8	2001 - 7	4711 - 13	15946 - 51
heuristic 1)	Nb FF	1852	437	3158	1595
Improvement (%)		**11.0%**	**34.9%**	**9.3%**	**23.5%**
CRTPN	Time (s)	270.8	170.48	325.5	238.76
(with	s - del	410513 - 25	223741 - 23	489423 - 55	362375 - 70
heuristic 1+2)	Nb FF	1815	437	2816	1590
Improvement (%)		**12.7%**	**34.9%**	**19.1%**	**23.7%**

δ is the delay of the operator producing the signal, δ_{ff} is the delay of a flip-flop register and $\delta_{obj} = \frac{1}{f} - \delta_{ff}$ is the maximal delay inside a pipeline stage to reach frequency f.

However, this algorithm makes an approximation which allows to insert a pipeline stage inside an elementary operator. This is not possible on the final circuit, and this leads to not being able to ensure a target operating frequency. We have adapted the greedy algorithm that preserves the atomicity of the operators with the following formula:

$$(c, \tau) + \delta = \left(c + \left\lfloor \frac{\tau + \delta}{\delta_{obj}} \right\rfloor, \tau \cdot \left(1 - \left\lfloor \frac{\tau + \delta}{\delta_{obj}} \right\rfloor \right) + \delta \right)$$

Note that this formula only works if $\left\lfloor \frac{\tau+\delta}{\delta_{obj}} \right\rfloor \in \{0, 1\}$, otherwise it means that $\delta > \delta_{obj}$ and the pipeline is not feasible.

This algorithm ensures the operating frequency and therefore allows a comparison with pipeline generation based on CRTPN modelling. The CRTPN has an input frequency $f' = \frac{1}{\delta_{obj}}$, to take into account the propagation delay of a pipeline register.

Our sample is composed of four single-precision float operators (Adder, Multiplier, Divider and Square root). The results are summarised in the table I. These arithmetic operators contain many elementary operators (108 to 189) and internal signals (165 to 316) which define the granularity of the model. This shows the scalability of the approach. The target frequencies chosen are close to the maximum frequency achievable by the circuit, with the FloPoCo's estimation on a Xilinx Virtex 6 target. The purpose is to show the interest of our approach on time-constrained circuits, with a maximum number of pipeline registers.

Both algorithms produce the same number of pipeline stages. The CRTPN approach reduces the number of flip-flops from 12% up to more than 34%, and never gives a worse result

than the greedy algorithm. This was expected as the greedy algorithm is one run of the CRTPN state graph.

Heurisic 1 adds a delayable transition when an operator has a larger signal size at the output than at the input. This heuristic gives very good results for a relatively small number of delayable transitions (except for the `FPSqrt` operator). This leads to a very fast pipeline generation and a significant reduction in the number of flip-flops (between 9.3% and 34.9%) compared to greedy. The `FPSqrt` operator has more delayable transitions, but this doesn't appear to influence much the calculation time. This is due to its very sequential structure.

Heuristic 2 allows mutualising signals at a branching point. Numerous transitions become delayable and this relaxation results in an explosion of the state space, and therefore of the associated computation time. If a delayable transition is placed at the beginning of the circuit, then the different runs generated will propagate through the state space exploration. The best results are obtained when combined with the first heuristic and are those detailed in the table. We deliberately limit the number of delayable transitions in order to find a compromise that allows to add enough delayable transitions while limiting the calculation time to a few minutes. The results are always improved, even if the results are relatively heterogeneous. The `FPDiv` operator seems well adapted to signal pooling by significantly reducing the number of flip-flops (from 3158 to 2816).

Heuristic 3 defines the minimum bound of the reset interval which is set to $\frac{1}{2f}$ for all experiments: lower bounds increased the calculation time, without improving the results.

The current tool is a functionnality-focused single threaded prototype, and more efficient implementations are hoped for in the future. First, the symbolic states as presented in section II can be efficiently represented as DBMs (Difference Bound Matrices) [20]. Moreover, since we have simplified zones (diagonal constraints are equalities), it will probably be possible to further optimise the classical operations over DBMs.

VI. Conclusion

We have proposed a formal approach to generate automatically the pipeline of an arithmetic operator. The Cost Reset Timed Petri Net model guarantees to produce a pipeline with a minimum target operating frequency, and with a minimal resource consumption considering the size of registers and possible mutualisation on signal branching points.

A state space exploration algorithm guided by cost chooses among all the possible pipeline combinations the one that minimizes the resources allocated to the pipeline. The complexity of this algorithm is proven to be PSPACE-complete.

We provide heuristics that reduces the state space size, and so the computation time. Although they do not ensure to obtain the optimal solution, they target some specific parts of the circuit where resources can be efficiently saved : operators with bigger size in input than in output, and operators that can mutualise registers.

A prototype has been developed to automatically generate the pipeline, while optimising hardware resources by up to 34% compared to a classical approach using a greedy algorithm, on real use cases.

This first results are encouraging and there is still room for improvement, for example in our implementation by using optimised DBM library for the computation of the state space. We also believe that the representation of the circuit is tight enough to produce pipeline which handles resource sharing of functional parts, and thus save even more resources.

References

[1] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, no. 1-6, pp. 5–35, Jun. 1991.

[2] M. Istoan and F. de Dinechin, "Automating the pipeline of arithmetic datapaths," in *Design, Automation & Test in Europe Conference & Exhibition (DATE 2017)*, Lausanne, Switzerland, 2017, pp. 704–709.

[3] F. de Dinechin and B. Pasca, "Designing custom arithmetic data paths with FloPoCo," *IEEE Design & Test of Computers*, vol. 28, no. 4, pp. 18–27, Jul. 2011.

[4] D. Bufistov, J. Cortadella, M. Kishinevsky, and S. Sapatnekar, "A general model for performance optimization of sequential systems," in *2007 IEEE/ACM International Conference on Computer-Aided Design*, 2007.

[5] J. Campos, G. Chiola, J. M. Colom, and M. Silva, "Properties and performance bounds for timed marked graphs," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 1992.

[6] M. Najibi and P. A. Beerel, "Slack matching mode-based asynchronous circuits for average-case performance," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '13. IEEE Press, 2013, p. 219–225.

[7] Sangyun Kim and P. A. Beerel, "Pipeline optimization for asynchronous circuits: complexity analysis and an efficient optimal algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 3, pp. 389–402, 2006.

[8] L. P. Carloni, K. L. McMillan, A. Saldanha, and A. L. Sangiovanni-Vincentelli, "A methodology for correct-by-construction latency insensitive design," in *1999 IEEE/ACM International Conference on Computer-Aided Design. Digest of Technical Papers*, 1999, pp. 309–315.

[9] L. Josipović, S. Sheikhha, A. Guerrieri, P. Ienne, and J. Cortadella, "Buffer placement and sizing for high-performance dataflow circuits," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 186–196.

[10] R. Parrot, M. Briday, and O. H. Roux, "Timed Petri Nets with Reset for Pipelined Synchronous Circuit Design," in *The 42th International Conference on Application and Theory of Petri Nets and Concurrency (Petri Nets 2021)*, ser. Lecture Notes in Computer Science, vol. 12734. Springer, Jun. 2021.

[11] P. M. Merlin, "A study of the recoverability of computing systems," Ph.D. dissertation, Department of Information and Computer Science, University of California, Irvine, CA, 1974.

[12] C. Ramchandani, "Analysis of asynchronous concurrent systems by timed Petri nets," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, 1974, project MAC Report MAC-TR-120.

[13] L. Popova-Zeugmann, *Time and Petri Nets*. Springer, 2013.

[14] P. A. Abdulla and R. Mayr, "Priced Timed Petri Nets," *Logical Methods in Computer Science*, vol. 9, no. 4, 2013.

[15] H. Boucheneb, D. Lime, O. H. Roux, and C. Seidner, "Optimal-cost reachability analysis based on time Petri nets," in *18th International Conference on Application of Concurrency to System Design (ACSD'18)*, Bratislava, Slovakia, Jun. 2018.

[16] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA*. IEEE Computer Society, 1977, pp. 46–57.

[17] R. Alur, C. Courcoubetis, and D. Dill, "Model-checking in dense real-time," *Information and Computation*, vol. 104, no. 1, pp. 2–34, 1993.

[18] P. Bouyer, K. G. Larsen, and N. Markey, "Model checking one-clock priced timed automata," *Logical Methods in Computer Science*, vol. 4, no. 2, May 2008.

[19] A. Cheng, J. Esparza, and J. Palsberg, "Complexity results for 1-safe nets," *Theoretical Computer Science*, vol. 147, pp. 117–136, 1995.

[20] R. Bellman, *Dynamic Programming*. Princeton, NJ, USA: Princeton University Press, 1957.