A Performance-Oriented Hardware/Software Partitioning for Datapath Applications

Laura Frigerio, Fabio Salice
Dip. di Elettronica e Informazione, Politecnico di Milano
Piazza Leonardo da Vinci, Milano, Italy
{lfrigerio,salice}@elet.polimi.it

ABSTRACT

This article proposes a hardware/software partitioning method targeted to performance-constrained systems for datapath applications. Exploiting a platform based design, a Timed Petri Net formalism is proposed to represent the mapping of the application onto the platform, allowing to statically extract performance estimations in early phases of the design process and without the need of expensive simulations. The mapping process is generalized in order to allow an automatic exploration of the solution space, that identifies the best performance/area configurations among several application-architecture combinations. The method is evaluated implementing a typical datapath performance constrained system, i.e. a packet processing application.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-purpose and Application based Systems,Real-time and embedded systems

General Terms

Design, Performance

1. INTRODUCTION

Datapath applications, where dataflow elaboration dominates over the control-flow constructs, are gaining increasingly popularity in embedded system design (examples are DSP or packet processing applications). One of the main concern when designing this type of applications is to meet strict timing constraints without sacrificing too much the flexibility and at a reasonable cost. Throughput requirements have become particularly critical in last years; for example, in packet processing applications typical requirements have evolved from 1 Gbps in 1990 up to the actual 10 or more Gbps.

The increasing complexity of System-on-Chip designs and the need to cope with conflicting requirements are pushing the development of methodologies and tools, that allow to easily define, analyze and modify a system. In order to manage the design space exploration the Y-chart scheme [1] has been proposed (Figure 1). In this approach, the application and the architecture are defined separately. The application is then mapped onto the architecture and the performance of the system is evaluated. Based on the obtained results, the designer may decide to choose or change the architecture, to modify the mapping or to restructure the application.

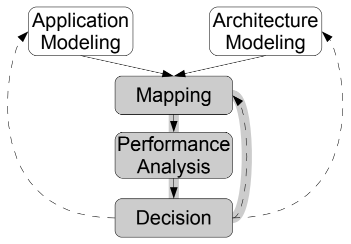

Figure 1: The Y-chart scheme

In this paper we propose a method to automate the architecture mapping, performance evaluation and decision making in the Y-chart approach (these phases are shaded in Figure 1) for datapath oriented application with stringent throughput requirements. Different methods can be used to evaluate the performance of a system and can generally be classified in (1) simulation techniques and (2) formal models. Simulation techniques provide information on the system behaviour by tracing the results obtained when applying stimuli to a system model. Pure simulative approaches using for example the SystemC Library have been applied in [2] and [3]. However, simulation approaches are time consuming and can not provide information on system properties like the absence of deadlocks or system bottlenecks.

Formal models describe the system in a mathematical form and can provide accurate information on its behaviour, even if their effectiveness decreases with the increase of the system dimensions. Examples of formal models used for performance evaluation are Markov processes, Queuing Networks and Timed Petri Nets. In the field of performance evaluation for SoC design, Network Calculus [5] and Stochastic Automata Networks [6] have also been recently proposed.

In this paper, we consider Timed Petri Nets since this formalism is especially suited for describing HW/SW systems. Petri Nets are an intuitive and powerful way to define concurrent and asynchronous processing, resource sharing and events synchronization [8]. The graphical representation of a Petri Net allows to easily describe and interpret system models, and the mathematical framework provides a way to formally describe and analyze them. Moreover, with respect to other methods that consider Stochastic timing models only, Timed Petri Nets allow to consider both Deterministic and Stochastic timing models [4]. This is particularly useful in SoC design, where IP (Intellectual Property) blocks are often used and the exact timing required to process input data is often available (e.g. number of clock cycles of an hardware module) and can be exploited to build accurate performance models. Finally, since Petri Nets are a well established modeling formalism, several tools are provided to support both the extraction of analytical properties and the simulation of Petri Nets models.

Differently from other works, which consider Petri-Nets as a formal intermediate representation [7], this paper proposes the use of a Timed Petri Net for the application-architecture mapping. This provides a description that allows to formally extract performance figures in the early phases of the system design and without the need of expensive simulations. Moreover, the use of a mathematical formalism allows to extract properties of the system that can be used to automatically explore the solution space, identifying the best performance/area tradeoffs among several application-architecture combinations.

The rest of the paper is organized as follows. Section 2 introduces the formal definitions that will be used thorough the paper. Section 3 describes how to model a datapath system with the use of Timed Petri Nets. Section 4 presents an analytical approach and a semantical interpretation for the performance analysis. Section 5 presents the algorithm used to explore the solution space. Section 6 reports some experimental results and Section 7 concludes the work.

2. FORMAL DEFINITIONS

In this section we recall some definitions and properties of Timed Petri Nets that are used in the rest of the paper. A complete description can be found in [8], [4].

A Petri Net (PN) is a 5-tuple, $PN = (P, T, C, W, M_0)$ where $P = \{p_1, p_2, \dots p_m\}$ is a finite set of places, $T = \{t_1, t_2, \dots t_n\}$ is a finite set of transitions, $C \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (connections), $W : C \rightarrow \mathbb{N}^+$ is a weight function, $M_0 : P \rightarrow \mathbb{N}_0$ is the initial marking, with $P \cap T = \emptyset$, $P \cup T \neq \emptyset$.

The dynamic behaviour of a PN is described in terms of two rules: the *enabling rule* and the *firing rule*. A transition is enabled when all its input places contain a number of tokens equal or greater than the weights of the corresponding arcs. An enabled transition can fire or not. When the transition fires it removes a certain amount of tokens from its input places and produces a certain amount of tokens in the output places. The amounts of tokens removed and produced depend on the arcs weights.

The incidence matrix $A = [a_{tp}]$ is a $n \times m$ matrix of integer where $a_{tp} = a^+_{tp} - a^-_{tp}$, $a^+_{tp} = w(t, p)$, $a^-_{tp} = w(p, t)$ and represents the "amount of change" in the number of tokens in p caused by the firing of t. An integer solution y to $Ay = 0$ is called S-invariant. An integer solution of $A^T x = 0$ is called T-invariant. A Petri Net is *consistent* if $\exists x > 0, A^T x = 0$.

In Timed Petri Nets, transition occurrences fire in 'real-time' associated with each occurrence of each transition. In this paper we consider deterministic nets, where the times

F1
F2
F3
F4
F5

Figure 2: Task graph of a datapath application.

are deterministic. Any enabled transition starts its firing in the same instant in which it becomes enabled. Each firing can be considered as a three phase event; first, the tokens are removed from the input places of the firing transition, the second phase is the firing time period, and when it is finished, tokens are deposited to output places of the transition. If a transition occurrence becomes enabled while it is firing a new independent firing cycle begins.

Formally, a Timed Petri Net (TPN) is a pair (PN, h) where PN is a Petri Net, $PN = (P, T, C, W, M^0)$ and h is a firing time function that assigns a positive rational number to each transition of the net: $h : T \rightarrow \mathbb{R}^+$.

3. TIMED PETRI NET MODEL

Datapath applications are dominated by dataflow behavior, with few control-flow constructs. They can be decomposed in distinct tasks at a coarse level of granularity; the tasks are computation intensive and internally strongly interconnected. These applications have iterative nature, since they repeatedly execute over different sets of input data. Example of datapath applications are DSP algorithms or packet processing applications.

A simple way to represent datapath applications is using a data dependent based task graph at coarse granularity. In the following, we will consider applications performing a set of subsequent tasks. Each task is referred as *function* (to avoid confusion with the *software tasks* used later on) and is intended as an operation or set of operations performed on some data. Each independent chunk of input data is referred as *data unit*. An example of task graph is given in Figure 2 where circles identify functions and arcs are used to specify data dependencies.

The architecture is composed of executors that can be processors or hardware modules. There can exist multiple instances of the same executor, in order to satisfy the performance requirements. In the following, we indicate as *resource class* (or in short resource) a set of identical executors, and as *availability* the number of instances of executors in the same resource class. The architecture is therefore specified by the set of *resource classes* and their *availability*.

A mapping associates application *functions* to architecture *resources*. Formally, given a set of functions F and a set of resources R we define for each function a mapping g on the resource on which it is executed, $g : F \rightarrow R$. The execution of a function F_i on a resource R_j requires a certain execution time e_{ij}. Values e_{ij} are known if the design process is based on IPs (Intellectual Property) or can be estimated on the basis of previous and similar implementations. A Timed Petri Net is used to model the mapping, as represented in Figure 3 for a simple example. For each function and each resource we introduce respectively two places (an F-Place and an R-Place). We also add a Q-place for each function to represent the queue of data waiting to execute the function (the first function does not require this place). The output transition of the F-Place is annotated with the execution time e_{ij} and the initial marking of each R-Place is

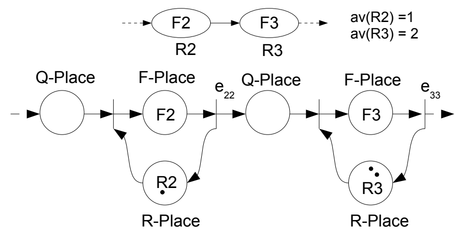

Figure 3: TPN generated from a task graph.

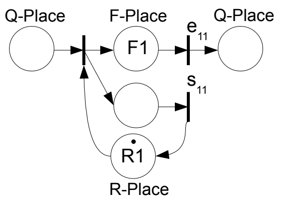

Figure 4: Modeling of a pipelined resource.

defined by its availability. F-Places are connected according to the task graph; if a resource is shared by different functions, a single R-Place is used and appropriate arcs are used to connect different F-Places to the same R-Place.

To model the presence of pipelined hardware resources, we can extend the representation as indicated in Figure 4. The execution on a pipelined resource is characterized by two values: the execution time e_{ij} representing the total time to execute the function and a time s_{ij} representing the stage time defining the rate at which the input data can be accepted (usually equal to one clock cycle).

To complete the system modeling, we introduce a limit on the number of data units that can be processed simultaneously. In some platforms (like the one presented in Section 6.1) this limit is considered explicitly, while in other is implicit and related to the depth of communication queues between the modules. We add a place (P-Place) having as initial marking a number of tokens equal to the maximum number of data units allowed in the system.

In case the communication introduces substantial overheads, it can be modeled using the same framework exploited for the rest of the system (for example, a data transfer becomes a *function* and a *bus* becomes a *resource*).

4. PERFORMANCE EVALUATION

As mentioned before, one of the critical requirements for datapath application is the meeting of stringent throughput constraints. By expressing the mapping between application and architecture as explained in the previous section, we obtain a consistent Timed Petri Net ($\exists x > 0, A^T x = 0$). The minimum cycle time τ_{min} of this net (equivalent to the inverse of the maximum throughput) is computed as [8]:

$$\tau_{\min} = \max_k \{ y_k^T (A^-)^T D x / y_k^T M_0 \} \tag{1}$$

over all the independent minimal-support S-invariants of matrix A, $y_k \geq 0$. M_0 is the initial marking, D is the diagonal matrix of $d_i, i = 1, 2, \ldots, r$ with d_i the time associated to transition i and A^- the matrix of values a^-_{tp} defined in Section 2.

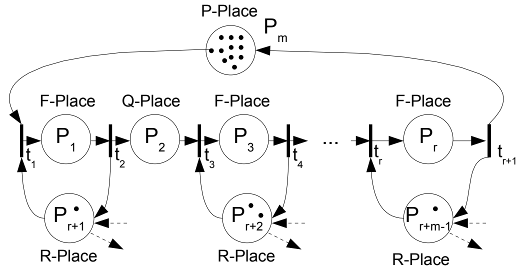

Figure 5: Notation for a general Petri Net model

To find the S-invariants consider the following partition of the incident matrix A of a consistent net with m Places and n transitions:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \tag{2}$$

where A_{11} is a non singular $r \times r$ matrix, with r the rank of A. A set of linearly independent S-invariants y is given by the $(m - r)$ rows of the $(m - r) \times m$ following matrix[8]:

$$B_f = [-A_{12}^T (A_{11}^T)^{-1} : I_{m-r}] \tag{3}$$

where I_{m-r} is the identity matrix of order $m - r$.

Consider a Petri Net built as explained in the previous Section, with the notation shown in Figure 5. The correspondent matrix A has the following structure:

$$\text{A=} \begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ \vdots \\ t_{r+1} \end{array} \begin{array}{c} \begin{array}{ccccc|ccccc} P_1 & P_2 & P_3 & \ldots & P_r & P_{r+1} & P_{r+2} & \ldots & P_{m-1} & P_m \end{array} \\ \left[\begin{array}{ccccc|ccccc} 1 & 0 & 0 & \ldots & 0 & -1 & 0 & \ldots & 0 & -1 \\ -1 & 1 & 0 & \ldots & 0 & 1 & 0 & \ldots & 0 & 0 \\ 0 & -1 & 1 & \ldots & 0 & 0 & -1 & \ldots & 0 & 0 \\ 0 & 0 & -1 & \ldots & 0 & 0 & 1 & \ldots & 0 & 0 \\ \vdots & & & & & & & & & \\ \hline 0 & 0 & 0 & \ldots & -1 & 0 & 0 & \ldots & 1 & 1 \end{array}\right] \end{array}$$

Let us consider the partition reported above that separates the sequence of F-Places and Q-Places from R-Places and P-Place and the first r transitions from the $(r+1)$-th transition. It is easy to verify (e.g. with Gauss algorithm) that the matrix has rank r. The partition therefore identifies the matrices A_{11}, A_{12}, A_{21} and A_{22} according to Equation 2. The structure of this matrices is very regular: matrix A_{11} is a Toeplitz matrix with diagonal elements $a_{ii} = 1, \forall i, 0 < i \leq r$, elements $a_{i,i-1} = -1, \forall i, 1 < i \leq r$ and all the other elements equal to zero. Moreover, each of the last $m - r$ columns of matrix A has couples of consecutive $(1, -1)$, corresponding respectively to the transitions that consume and produce tokens in the associated R-Place (or P-Place for the last column), all the other elements are zero. If the resource is shared there is more than one couple of $(1, -1)$ values in that column.

Applying equation 3 we obtain that there are $m - r$ S-invariants:

- $m-r-1$ S-invariants corresponding to the $m-r-1$ R-places in the system. Each vector has elements equal to 1 for the R-Place and for the F-Places using that resource (other elements are equal to 0).
- one S-invariant corresponding to the P-Place. This vector has elements equal to 1 for the P-Place and all the F-Places and Q-Places in the system (other elements are equal to 0).

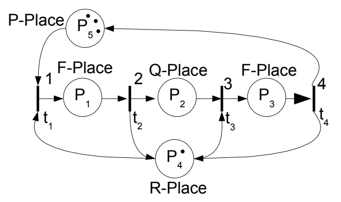

Figure 6: Example of Timed Petri Net model

Equation 1 corresponds in finding the minimum value among the sum of the times on transitions connecting the R-Place (or P-Place) with the corresponding F-Places, divided by the number of tokens in the considered places in the initial marking.

4.1 Example

Let us consider, the net represented in Figure 6, where the execution times are indicated next to the transitions. The corresponding matrix A is:

$$A = \begin{bmatrix} 1 & 0 & 0 & -1 & -1 \\ -1 & 1 & 0 & 1 & 0 \\ 0 & -1 & 1 & -1 & 0 \\ 0 & 0 & -1 & 1 & 1 \end{bmatrix} \tag{4}$$

From $A^T \cdot x = 0$ we obtain $x^T = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$. From equation 3 we obtain two S-invariants: $y_1^T = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \end{bmatrix}$ and $y_2^T = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 \end{bmatrix}$. The minimum performance is therefore given by: $\tau_{\min} = \max\{10/1, 10/3\} = 10$.

4.2 Interpretation

Intuitively, value τ_{min} is related to the processing time required by the resources to process a data unit. Resources that execute functions requiring long processing time are more likely to influence the minimum cycle time. However, a long computational path, even if supported by several resources, can affect the system performance.

Let us consider a simplified version of the system, in which each function is associated to a different resource, the initial marking of each R-Place is equal to one and there is no bound on the maximum number of data units in the system. In this case, the system is equivalent to a pipeline and therefore the minimum cycle time is given by the slowest stage. If the marking of a R-Place is different from one, it means that more data units can be executed simultaneously on that resource, therefore the time for that stage can be divided by the number of executors available (initial marking of the R-Place). When different functions are associated to the same resource, the system is not equivalent to a pipeline anymore, but the "stage" time of a resource is given by the sum of all the times of the functions associated to that resource. Finally, we have to take into account that only a finite number of data units can be processed simultaneously by the system. This means that the system can stall not because one of the resources is slow, but because the maximum number of data units in the system has been reached.

Considering this semantical interpretation, we can rewrite Equation 1 for a system composed of v functions and z resources as follows:

$$\tau_{min} = \max(rl_1, rl_2, \ldots, rl_z, gl) \tag{5}$$

where rl_j is the *Resource Latency* of resource j and gl is the *Global Latency.*

For each resource R_j the *Resource Latency* rl_j is defined as:

$$rl_j = \sum_i e_{ij}/M_0(R_j) \ \texttt{with} \ i \in \{i|g(F_i) = R_j, 1 \le j \le v\}, \tag{6}$$

where $M_0(R_j)$ is equal to the marking of the place associated to R_j and $g(F_i)$ is the mapping of F_i.

The *Global Latency* gl is defined as:

$$gl = \sum_i e_{ij}/M_0(P), 1 \le i \le v \tag{7}$$

where $M_0(P)$ is equal to the marking $M_0(P)$ of the P-Place.

If a resource is pipelined, we must consider in equation 6 the time s_{ij} instead of e_{ij} describing the rate at which data can be accepted by the resource (see Figure 4).

5. PARTITIONING ALGORITHM

The previous equations can be exploited in order to automate the exploration of the solution space given a throughput constraint. Considering v functions and z resources, the solution space is composed of z^v alternatives (number of possible partitionings). However it is very unlikely that every functions can be executed on every resource, since, hardware cores can usually execute very specific functions. The number of possible partitioning is therefore definitely inferior to z^v. Considering the coarse grain at which we are operating, the number of functions considered is relative small (usually in the order on 10-20 functions). Following these observations, the exploration of the solution space has been implemented using a branch and bound algorithm, where equations 5, 6 and 7 are exploited to compute the bounds for pruning the solution tree.

Given a throughput constraint, we search for the solution that minimizes the area, by assigning each function to a resource. The branch operation corresponds to assigning a function to all the resources that can execute it. The bounding operation corresponds to calculate values rl_j and gl considering the functions and the resources composing the partial solution. If one of these values exceed the minimum cycle time specified by the user the branch is discarded.

More in detail, the exploration algorithm works with the following input: (1) the set of functions composing the application; (2) the set of resources composing the architecture (the information related to each resource are: the execution frequency, the availability, the area, the indication if the resource is pipelined or not); (3) values e_{ij} of execution times of function F_i on resource R_j, related to the resource frequency; (4) a maximum throughput (minimum cycle) constraint.

The B&B algorithm is characterized by:

- *Branching rule*: assignment of a function to all the candidate resources.
- *Bounding functions*: rl_j and gl considering the functions and the resources composing the partial solution in a node. If one of the values is greater than the mimimum cycle the solution is discarded.
- Strategy for the selection of the next branch: complex function are selected first (considering the average time of execution on the available resources).

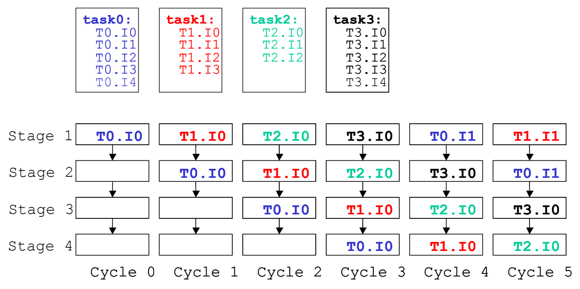
task0: T0.I0 T0.I1 T0.I2 T0.I3 T0.I4
task1: T1.I0 T1.I1 T1.I2 T1.I3
task2: T2.I0 T2.I1 T2.I2
task3: T3.I0 T3.I1 T3.I2 T3.I3 T3.I4
Stage 1
Stage 2
Stage 3
Stage 4
Cycle 0
Cycle 1
Cycle 2
Cycle 3
Cycle 4
Cycle 5

Figure 7: Instruction interleaving

6. EXPERIMENTAL RESULTS

In order to evaluate the approach we implement a typical datapath application, i.e. a packet processing application performing an IP (Internet Protocol) packet forwarding function, using as reference platform the HW/SW architecture for datapath application developed by Altera [10].

The system receives a MAC (Medium Access Control) input packet, verifies that the packet is valid, modifies some packet fields, computes the destination MAC address and issues the packet [9]. The function composing the system are the following: function F_1 receives the packet, function F_2 performs IP header checks and extracts the IP address, function F_3 computes the destination MAC address, function F_4 updates some IP fields, function F_5 computes the $VLAN$ tag, function F_6 updates TTL and $Checksum$ fields, function F_7 formats and outputs the packet and function F_8 concludes the processing of the packet.

In the following we first provide an overview of the Altera platform and we then present the experimental results that are organized in two phases: (1) we verify the suitability of the description of a system with the presented Timed Petri Net approach, (2) we present some results obtained applying the algorithm for the solution space exploration.

6.1 Hardware/Software reference architecture

Altera has developed a hardware/software solution for high performance datapath applications, with particular applicability to packet processing domain, that combines a multithreaded soft processor and hardware accelerators.

The soft processor can execute 8 threads simultaneously by means of a non-conventional multithreading. Instructions corresponding to 8 different threads processing different data units are mixed (interleaved) in the pipeline (Figure 7 represents the situation for an exemplified pipeline with 4 stages). With this approach, the software execution time becomes deterministic given an execution path, since all the sources of indeterminism are avoided: hazards in a given thread instruction are resolved before the next instruction of the same thread is executed and only on-chip memory is used (no cache is required since data and program code are usually limited in size).

A typical processing flow is based on an asynchronous execution paradigm that combines Tasks that are executed in software and Events executed by dedicated hardware blocks, as schematically depicted in Figure 8. The maximum number of data units in the system is a processor parameter that can be configured during the hardware synthesis (typical values: 32-64).

To model the multithreading in the Timed Petri Net, we

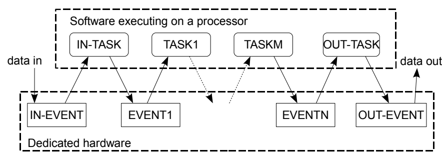

Figure 8: Execution Flow

consider that the execution of eight threads on the same processor at frequency F_{soft}, with the instruction interleaving described, is functionally equivalent to the execution of eight threads on eight identical processors each one running at a frequency $F_{soft}/8$. The multithreaded processor is therefore represented with a resource class having availability equal to eight and frequency equal to $F_{soft}/8$.

6.2 Petri Net model verification

In order to verify the suitability of the Petri Net model we compare the value of the minimum cycle for the system obtained by 1) defining and simulating the Timed Petri Net with a PN simulation tool (CPN tool [11]), 2) implementing and simulating the system through the Altera toolchain that combines an ISS for the processor with software models of hardware blocks [10], 3) applying the static analysis presented in Section 4.

For this test we consider a fixed partitioning in which F_1, F_3, F_5 and F_7 are executed by hardware modules (a shared module is used for F_3 and F_5) and F_2, F_4, F_6 and F_8 are executed on the multithreaded processor (with 8 threads).

In Table 1, for each function F_i, the value e_i represents the number of clock cycles if it is executed by a hardware module, or the number of assembly instructions if it is executed by the processor. Both hardware and software run at the same frequency (250MHz). The first configuration corresponds to the real system timing. In order to study the behaviour with different configurations, additional instructions and clock cycles have been considered in the other configurations.

Quantities NP_m and NP_s correspond to the average Number of Packets produced by the system when simulating, respectively, the Petri Net Model and the implemented System for one million time units; σ_{npm} and σ_{nps} are the correspondent standard deviations computed over 20 simulations. τ_m, τ_s, τ_p represent the average system cycle time for respectively the Petri Net Model, the implemented System and the prediction computed using equations in Section 4. ar represents the arrival rate of the packets; when ar is less than the τ_{min} we indicate this situation as working condition, since the system can support the input rate (in this case $\tau_p = ar$). When ar is greater than τ_{min} the system is operating in saturated conditions, processing the maximum number of packets possible (in this case $\tau_p = \tau_{min}$).

As can be noticed, the predictions are quite accurate when compared both to the Petri Net Model results and the implemented System results. In all the tested situations the percentage errors are always less than 8%.

6.3 Partitioning algorithm

To identify the best solution that minimize the area, while maintaining the flexibility and satisfying the throughput constraints, we apply the exploration algorithm to the input

e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	ar	NP_m	σ_{npm}	NP_s	σ_{nps}	τ_m	τ_s	τ_p	Cond.	$\%e_m$	$\%e_s$
1	10	32	10	32	10	50	1	80	12497	0	12196	0	80.02	81.99	80	work.	0	2
1	30	32	20	32	20	50	1	80	12493	0	12196	0	80.04	81.99	80	work.	0	2
1	30	32	20	32	20	50	1	120	8329	0	8197	0	120.06	122.00	120	work.	0	2
1	30	80	20	80	20	50	1	120	8328	0	8197	0	120.08	122.00	120	work.	0	2
1	20	32	20	32	20	50	1	10	16337	8.82	15152	1.43	61.21	66	61	sat.	0	8
1	30	32	30	32	30	50	1	10	10942	12	10417	2.31	91.39	96	91	sat.	0	5
1	50	32	20	32	20	50	1	30	10952	3.43	10103	6.23	91.31	98.98	91	sat.	0	8
1	50	80	20	80	20	50	1	30	10953	6.46	10102	5.15	91.30	98.99	91	sat.	0	8
1	30	5000	20	5000	20	50	1	120	5969	0	6037.29	1.72	167.53	165.64	158.2	sat.	6	4

Table 1: Throughput comparison in working and saturated conditions

	$R_1(processor)$	$R_2(HWmod.)$	$R_3(HWmod.)$	$R_4(HWmod.)$
F_1	20	1	10	-
F_2	30	20	25	10
F_3	50	40	45	32
F_4	20	10	-	10
F_5	30	10	40	32
F_6	20	15	20	-
F_7	80	60	50	70
F_8	20	15	1	20
Area	1(2017)	548	358	233

Table 2: Input data for the exploration algorithm

τ_{min}	Mpk/s	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	N. sol	Time(s.)	Area
280	0.89	R_1	R_1	R_1	R_1	R_1	R_1	R_1	R_1	27648	1	2017
230	1.09	R_1	R_1	R_4	R_1	R_1	R_1	R_1	R_4	27632	1	2250
150	1.67	R_1	R_4	R_4	R_1	R_4	R_1	R_1	R_4	25623	1	2250
110	2.27	R_3	R_1	R_3	R_1	R_1	R_1	R_3	R_3	16671	1	2375
100	2.50	R_2	R_2	R_2	R_2	R_2	R_1	R_1	R_2	13432	1	2565
80	3.13	R_1	R_3	R_4	R_1	R_4	R_1	R_3	R_3	5586	<1	2608
70	3.57	R_3	R_4	R_4	R_1	R_1	R_1	R_3	R_4	2075	<1	2608
60	4.17	R_2	R_2	R_1	R_2	R_2	R_2	R_2	R_3	351	<1	2923
50	5.00	R_2	R_2	R_4	R_1	R_2	R_1	R_3	R_2	43	<1	3156

Table 3: Exploration algorithm results.

data presented in Table 2, where values (F_i,R_j) corresponds to values e_{ij} (expressed in number of clock cycles or number of instructions) and the last row is an area measure obtained synthesizing the resource on an Altera Stratix FPGA (each value correspond to a the number of normalized logic elements obtained as: Resource Logic Elements + Resource Memory bits · Total FPGA Logic Elements/Total FPGA Memory bits). In order to select flexible solutions first (with software processor) we force the area of the software processor to a low value during the solution space exploration.

Table 3 reports the results obtained by increasing the throughput requirement. The first and the second columns in the table represent the throughput constraint expressed as minimum cycle time or maximum throughput (millions of packets per second with hardware running at 250MHz). Column $F_1 \ldots F_8$ indicate the best mapping obtained for each case. The last three columns indicate the number of found solutions that meet the constraint, the time to run the exploration algorithm (in seconds on an Intel Xeon 3.4Ghz) and the system area (in equivalent logic elements).

As it can be noticed, as the throughput constraint becomes more demanding, the number of solutions satisfying the constraint diminish and the best solution tend to evolve from a completely software configuration to hardware/software configurations. These solutions become more and more expensive in term of area but allow to satisfy the specified timing constraint. For this explorations the time required by the algorithm is negligible (≤ 1 sec).

7. CONCLUSION

This paper presents a method for the solution space exploration of datapath applications with stringent throughput constraints. We propose the use of Timed Petri Nets to represent the mapping of the application onto the architecture, in a Y-chart approach. We obtain a set of bounds that are exploited by an exploration algorithm, based on a branch and bound approach, to search the solution space for the best performance/area configuration. The approach has been applied to a packet processing application and the experimental results show that (1) the Petri Net Model can accurately represent the behaviour of a real system (2) the exploration algorithm is able to find the best compromise in terms of area/throughput in reasonable times.

8. ACKNOWLEDGMENTS

The authors would like to thank Dr. Argy Krikelis and the European Technology Center of Altera (UK) for providing support and the technology for the methodology evaluation.

9. REFERENCES

[1] B. Kienhuis, E. Deprettere, K. Vissers, P. van der Wolf, *An approach for quantitative analysis of application specific dataflow architectures*, in Proc. ASAP 1997.

[2] C. Haubelt, J. Falk, J. Keinert, T. Schlichter, M. Streubühr, A. Deyhle, A. Hadert, J. Teich, *A SystemC-based design methodology for digital signal processing systems*, EURASIP J. Emb. Syst., N.1, 2007.

[3] K. Ueda, K. Sakanushi, Y. Takeuchi, M. Imai *Architecture-level Performance Estimation for IP-based Embedded Systems*, in proc. DATE 2004.

[4] W. M. Zubereck, *Timed Petri Nets - definitions, properties and applications*; Microelectronic and Reliability, pp 627-644, 1991.

[5] A. Maxiaguine, S. Unzli, S. Chakraborty, L. Thiele, *Rate analysis for streaming applications with on-chip buffer constraints*, in proc. ASP-DAC'2004.

[6] N. Zamora, X. Hu, R. Marculescu, *System-Level Performance/Power Analysis for Platform-Based Design of Multimedia Applications*,ACM Trans. on Design Automation of Electronic Systems, Vol.12, N.1, 2007.

[7] P. Maciel, E. Barros, W. Rosenstiel, *A Petri Net Model for Hardware/Software Codesign*, Journal Design Automation for Embedded Systems, Springer, 1999.

[8] T. Murata, *Petri Nets: Properties, Analysis and Applications*, Proceedings IEEE, Vol.77, N.4, 1989.

[9] *RFC1812: Requirements for IP Version 4 Routers*, RFC Editor, United States, 1995.

[10] Altera Corporation website, `www.altera.com`.

[11] CPN Tools website, `www.daimi.au.dk/CPnets/`.