A Partial Reconfigurable Architecture for Controllers based on Petri Nets

Paulo Sérgio B. Nascimento, Paulo Romero M. Maciel, Manoel E. Lima, Remy E. Sant'ana, Abel Guilhermino S. Filho
Federal University of Pernambuco
Informatics Center
Fone: (+55) 8121268430, Recife-PE, Brazil
{psbn,prmm,mel,res,agsf}@cin.ufpe.br

ABSTRACT

Digital Control System in the industry has been used in most of the applications based on expensive Programmable Logical Controllers (PLC). These Systems are, in general, highly complex and slow, with an operation cycle around 10ms. In this work, a Reconfigurable Logic Controller (RLC) approach is presented, based on a small and low cost Xilinx Virtex-II FPGA architecture, operating as a virtual hardware machine. In this context, the main process is specified in a formal language, based on Petri nets or SFC (Sequential Function Chart). For applications that demand more hardware than that available in the FPGA, a partial reconfiguration mechanism takes place. From the Petri net specification, the main process is split into multiple contexts, which are sequentially executed within the same FPGA, without violating the operation cycle of application.

Categories and Subject Descriptors

J.7 [**Computer Applications**]: Computers in Other Systems – *industrial control, process control, real time.*

General Terms

Performance, Design, Economics, Algorithms, Languages.

Keywords

Petri Nets, Programmable Logic Controller (PLC), FPGAs, Virtual Hardware, Partial Reconfiguration.

1. INTRODUCTION

Digital machine control system has been widely used where exists a set of binary sensors and actuators that only assume one of the two states. Generally, this control is made by specific application equipment PLC (Programmable Logical Controllers), based on a microprocessor. In most industrial applications, the PLC programmers use one the following languages: LAD (*Ladder Diagram*), SFC (*Sequential Function Chart*) or STL (*Statement List*). These languages present different syntax and facilities to represent the process control flow. Particularly, the SFC language [1] is a Petri net based language, similar to GRAFCET specification [3].

In this work, a new Reconfigurable Logic Controller (RLC) architecture for applications of discrete machine control is presented. The RLC uses Temporal Petri Net (TPN) [3][5] as a formal language for functional description of applications. This architecture is based on a Xilinx Virtex-II FPGA (Field Programmable Gate Array) [7][8] device. The use of Petri nets allows a natural description for functional and temporal capture of the machine control system. It happens because Petri nets can easily specify parallelism, concurrence and asynchronous aspects. Besides, a Petri net specification can also be easily derived from LAD and SFC specifications, already used by commercial PLCs [13]. The advantage of a Petri net description is its powerful set of mathematical analysis mechanism that allows verification of properties and system correctness detection. To synthesize the processes in hardware, the Petri net specification is translated to VHDL and further mapped into a Virtex-II FPGA.

FPGA architectures present advantages such as high parallelism and control processing speed-up and a large amount of general purpose I/O-pins, with different voltage standards and high input/output rates when compared with PLC architectures. Current PLC implementations, based on microprocessors, presents difficulties to explore parallelism and meet severe time constraints for high complex applications. This difficulty is result of PLC sequential interpreted-execution of the system specification [4] in software.

Even for applications that need more resources than that available in a such FPGA, a multi-context approach [6][2], based on the Petri net temporal partitioning can be performed without loose of performance in several applications. In this way, this work proposes an architecture in which the main process can be split into small ones and executed in a only FPGA Virtex-II Xilinx device [9], through a temporal and partial reconfiguration process.

The section 2 presents the control application model in Temporal Petri net (TPN); the section 3 exhibits the TPN translation for a synthetizable VHDL; in the section 4 the architecture proposed and dynamic and partial reconfiguration on Virtex-II are discussed in details; section 5 presents a case study based on an elevator control; finally, the section 6 presents some conclusions and future works.

2. TEMPORAL PETRI NET MODEL

The Petri net [3][5] is a formal model that allows modeling of parallel, concurrent and asynchronous systems, in a natural way. At the same time it also allows verification of properties that can be fundamentals for the correct operation and control system safeness. Petri net is based on local place state that allows an easy and direct understanding of the modeled application. Besides, it is possible to translate a LAD and SFC [1] representation directly on a Temporal Petri net.

Others formal models, as Finite State Machines (FSM), also allow the discrete control systems specification, but, they difficult a direct system understanding, because it is based on the concept of global state. Global state hides the local behavior, turning difficult the process modeling.

The Figure 1 depicts the discrete and reactive control system used to model examples in this work.

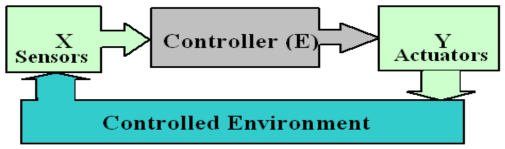

Figure 1. Discrete and reactivate control system.

In this system, the environment is monitored continuously from sensors that produce binary input signals $\boldsymbol{X}$. The controller, as a function of internal state $\boldsymbol{E}$ and the values of the sensors $\boldsymbol{X}$, promotes changes in states and in the output $\boldsymbol{Y}$. $\boldsymbol{Y}$ vector defines the actuators state, that defines actions on the environment, closing the loop control.

2.1 The TPN model overview

The proposed model for the Temporal Petri Net is represented by a seven-tuple ***Ctrl*** that represents the Petri net specification of the control process:

$\boldsymbol{Ctrl} = (\boldsymbol{TPN},\boldsymbol{X},\boldsymbol{Y},\boldsymbol{F},\boldsymbol{G},\boldsymbol{A},\boldsymbol{S})$, where:

$\boldsymbol{TPN}=(\boldsymbol{P},\boldsymbol{T},\boldsymbol{D},\boldsymbol{I},\boldsymbol{O},\boldsymbol{M_0})$

$\boldsymbol{P} = \{p_1,p_2,\ldots,p_N\}$ -- places set,
$\boldsymbol{T} = \{t_1,t_2,\ldots,t_M\}$ -- transitions set,
$\boldsymbol{D} = \{D_{t1},\ldots,D_{tM}\}$ -- transitions delays set,
$\boldsymbol{I} = \{\boldsymbol{I}_t \mid t \in \mathbf{T}\}$ -- input places sets,
$\boldsymbol{O} = \{\boldsymbol{O}_t \mid t \in \mathbf{T}\}$ -- output places sets,
$\boldsymbol{M_0} = (\boldsymbol{M_0}(p_1),\ldots,\boldsymbol{M_0}(p_N))$ -- initial mark places,
$\boldsymbol{X} = \{x_1,x_2,\ldots,x_K\}$ -- inputs set,
$\boldsymbol{Y} = \{y_1,y_2,\ldots,y_L\}$ -- outputs set,
$\boldsymbol{F} = \{f_1,f_2,\ldots,f_R\}$ -- internal flags set,
$\boldsymbol{G} = \{G_{t1},\ldots,G_{tM}\}$ -- transitions guards set,
$\boldsymbol{A} = \{A_{t1},\ldots,A_{tM}\}$ -- transitions actions set,
$\boldsymbol{S} = \{S_{y1},\ldots,S_{yL}\}$ -- output functions set. (1)

The internal state $\boldsymbol{E}$ of the controller is represented by marking $M(\boldsymbol{P})$ of places set $\boldsymbol{P}$ and binary values of flags $f \in \boldsymbol{F}$ inside of controller:

$$\boldsymbol{E} = (M(\boldsymbol{P}), \boldsymbol{F}) \tag{2}$$

The ***TPN*** is *safe* [3][5], meaning that the places markings can only assume the values 0 or 1. The fire of transition $t \in \boldsymbol{T}$ represents a local modification of the state $\boldsymbol{E}$, in other words: the marks modification of local places into the set $\boldsymbol{I}_t \cup \boldsymbol{O}_t$. The fire of each transition $t \in \boldsymbol{T}$ is conditioned by the function of guard G_t, defined as an boolean expression of the input signs, of the places marks $M(\boldsymbol{P})$ and of the values of the flags $\boldsymbol{F}$:

$$G_t = G_t(\boldsymbol{X},M(\boldsymbol{P}),\boldsymbol{F}) \in \{0,1\}. \tag{3}$$

The fire enables H_t of each transition $t \in \boldsymbol{T}$ is given by the boolean expression:

$$H_t = G_t \wedge M(p'_1) \wedge M(p'_2) \wedge \ldots \wedge M(p'_k) \quad \text{where} \quad p'_1,p'_2,\ldots,p'_k \in \boldsymbol{I}_t. \tag{4}$$

The fire of a transition $t \in \boldsymbol{T}$ can represent an action $A_t \in \boldsymbol{A}$ that consists in modifying the state of the output signals $\boldsymbol{Y}$ and of values the internal flags $\boldsymbol{F}$. The action A_t can be a combination of the following actions:

1. *Set*(q): make q = 1,
2. *Reset*(q): make q = 0,
3. *Cpl*(q): make q = 1-q,
4. ϕ : no actions where

$q \in \boldsymbol{Y} \cup \boldsymbol{F}$. (5)

In control applications, specification of temporizations is important because when the controller reaches a state, it waits for a delay time ($D_t \in \boldsymbol{D}$) to starts a transition action ($A_t \in \boldsymbol{A}$). The three fire time semantics are defined for the transitions as follows:

1. The transition t fire begins removing marks from the input places $p \in \boldsymbol{I}_t$.
2. The transition t waits the delay time D_t.
3. The transition t finishes and placing marks in the output places $p \in \boldsymbol{O}_t$ and executing the action A_t. (6)

Finally, each output $y \in \boldsymbol{Y}$ is given by a boolean function Sy, function of: the input signals $\boldsymbol{X}$, the place marks $M(\boldsymbol{P})$ and the internal flags signals $\boldsymbol{F}$. The output y can also be one internal flag $\in \boldsymbol{F}$, in this case called external flag:

$$y = \begin{cases} S_y(\boldsymbol{X},M(\boldsymbol{P}),\boldsymbol{F}) \text{ or} \\ \text{external } flag \in \boldsymbol{F} \end{cases} \tag{7}$$

3 TPN TO VHDL TRANSLATION

A VHDL code representing a controller description is shown below. The translation of the Petri net to a VHDL is done in a very intuitive way. The automatic parsing process of the controller is based on the expressions described in session 2.1:

```
ENTITY Ctrl  IS
PORT(RST     : IN STD_LOGIC;   -- reset
     CLK     : IN STD_LOGIC;   -- clock
     x1,...,xK : IN STD_LOGIC;  --inputs  X
     y1,...,yL : OUT STD_LOGIC –outputs Y ); END Ctrl;
ARCHITECTURE PetriNet OF Ctrl IS
SIGNAL p1,...,pN    : STD_LOGIC; --places P
SIGNAL f1,...,fR     : STD_LOGIC; --flags  F
-- For each transition t' such that  Dt'≠ 0:
   SIGNAL t's : STD_LOGIC; --start fire.
   SIGNAL t'd: STD_LOGIC; --finish fire.
SIGNAL Gt1,...,GtM : STD_LOGIC;--guards G.
BEGIN
Gti <= Gti(X,M(P),F); -- for each Gt ∈ G
PROCESS(RST,CLK)
--Time count variable for each delayed transition t':
   VARIABLE t'c : INTEGER RANGE 0 TO Dt';
BEGIN
  IF (RST='1') THEN --*****************************
            M(P) <= M0; -- initial place marks.
            t's<='0'; t'd<='0'; -- delayed transitions reset.
            f1<='?';...;fR<='?'; -- ? flags initial values.
  ELSIF((CLK'EVENT)AND(CLK='1')) THEN
-- Fire of each no delayed transition t:
          IF (Gt='1')AND(p='1', ∀ p∈ It) THEN
            p <='0'; --∀ p∈ It1, removes places marks.
            p' <='1'; --∀ p'∈ Ot1 , put places marks.
            $A_t$ ; -- action of transition  t.
         END IF;
--Fire of each delayed  transition t':
        IF ($G_{t'}$='1')AND(p='1', ∀ p∈ $\mathbf{I}_{t'}$) THEN
           p <='0'; --∀ p∈ $\mathbf{I}_{t'}$, remove places marks.
           t's <='1'; -- Start transition t' fire.
           t'c :=$D_{t'}$; -- Initial time count variable value.
        END IF;
        IF (t's='1') THEN
           IF (t'c=0) THEN
              t's <='0';
              t'd <='1';-- Finish transition t' fire.
           ELSE
              t'c:=t'c-1;-- wait delay time count.
           END IF;
        END IF;
        IF (t'd='1') THEN:
           t'd <='0'; -- complete transition  t' fire.
           p' <='1'; --∀ p'∈ $\mathbf{O}_{t'}$, put places marks.
           $A_{t'}$ ; -- action of  transition t'
         END IF;
  END IF;--***************************************
END PROCESS;
  $y_i$ <= $S_{yi}(\mathbf{X},M(\mathbf{P}),\mathbf{F})$; -- output function map $\mathbf{S}$.
END PetriNet;
```

The basic rules for Petri net to VHDL translation can be described as follows:

- In the VHDL code, the guard expressions $G_t \in \boldsymbol{G}$ and the output expression $y = S_y \in \boldsymbol{S}$ are implemented as concurrent statements.
- A process represents the transitions fire rules.
- The transitions are fired by each positive clock edge.
- "IF THEN ELSE" statements defines the control flow for each transition.
- For delayed transitions t' one variable t'c is used to count the delay time.

The translation of control behavior of ***Ctrl*** controller (session 2.1) into a VHDL file is also very easy and intuitive [1][11], increasing the reliability in the equivalence between the both behavior descriptions. The VHDL file encodes the internal states of the net associating one flip-flop to each place ($p \in \boldsymbol{P}$), flag or control elements of the delayed transitions, in similar way to the "*code on-hot*" method used in FSMs [11]. In spite of great use of the flip-flops, this method is well adapted for implementation in FPGA because of the great amount of flip-flops available in these devices. This translation technique also tends to decrease the size of the combinatorial logical during the synthesis for implementation of the transitions fires rules. Thus, in general, less hardware to implement the logic is needed and more speed in the application can be reached.

4 DYNAMICALLY RECONFIGURABLE ARCHITECTURE

Depending on the size of the controller, a suitable FPGA should be chosen. In the literature, however, several methods are proposed to eliminate this restriction by using "Virtual Hardware" technique, similar to the virtual memory used in modern computers [6][2]. This technique takes the advantage of the capacity of partial and dynamic reconfiguration of modern FPGAs, allowing that a great circuit can be partitioned in smaller sub-circuits, called contexts, which can be fitted into the FPGA in the time. A scheduler guarantees that the complete circuit is sequentially mapped and executed as contexts into the FPGA, taking into account all dependences, parallelism and correctness of the net.

In this work, as depicted in Figure 2, the Virtex II architecture from Xilinx [10] is used as the dynamic device for partial reconfiguration.

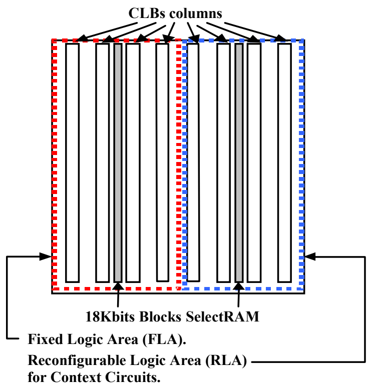

Figure 2. Virtex II FPGA architecture.

In these devices, the logic is organized as a set of *n* columns with *m* Configurable Logic Blocks (CLBs) per column and *k* 18kbits of RAM blocks. Parameters *n*, *m* and *k* are dependents of the chip. Each column in this architecture can be independently configured without any stopping of the logic running in other columns. The partial and dynamic reconfiguration characteristic

provide a powerful resource for those cases where larger circuits can be split into small pieces (contexts) in the time, and then processed sequentially or according to some special scheduling algorithm.

In this direction, we propose an architecture in which some fixed logic and just one reconfigurable area are present into the FPGA:

- The Fixed Logic Area (***FLA***) contains the circuits to generate the output signals ***Y*** (figure 1) and the reconfiguration control (scheduler). Once mapped into the FPGA, these circuits should not change during any partial FPGA reconfiguration.
- The Reconfigurable Logic Area (***RLA***) receives the scheduled context circuits to implement the controller functionality in the time. ***RLA*** can also receive input signals $\in \boldsymbol{X}$ (Figure 1).

The contexts are executed according to their transitions and associated actions.

4.1 The Dynamic Architecture

In the Figure 3 the dynamic architecture for multiple contexts execution is presented. Two main blocks as described before, the ***RLA*** and the ***FLA*** compose this synchronous model. The ***FLA*** block is also composed by two special registers, the ***OAR*** to store the partial context results and the ***Sy*** to store the output signal from the circuits. The execution process is very simple. After a positive edge of CLK_C, the context carried in the ***RLA*** is executed. ***OAR*** stores the arguments of the output functions ***Sy*** $\in \boldsymbol{S}$ generated by context execution. ***OAR*** is synchronized by a negative edge of context clock CLK_C.

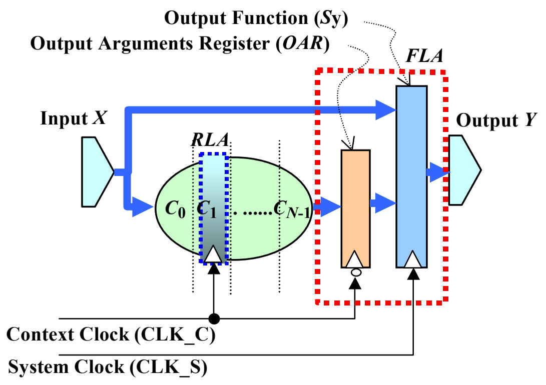

Figure 3. Multiplexed context model.

The partial values stored in ***OAR*** can be feedback to the ***RLA*** control in case of data dependency between the contexts. At the end, after the execution of all the contexts that compose the application, ***OAR*** contains the values of all the arguments of the output functions ***Sy***. The ***Sy*** in then up-to-dated to generate the final output results in ***Y***. ***Sy*** is loaded by the positive edge of system clock CLK_S. Each pulse in the system clock CLK_S corresponds to one controller operation cycle (scan cycle for PLC). If the application has *N* contexts we will have *N* pulses in CLK_C for each pulse in CLK_S.

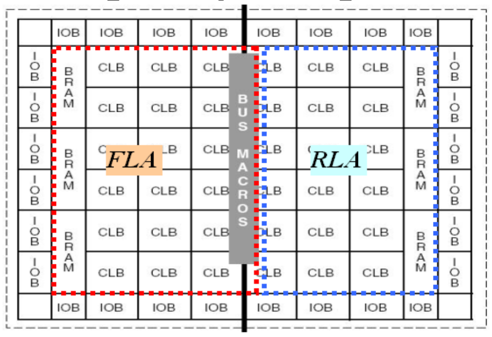

Figure 4. Dynamic architecture layout in Virtex-II.

The communication between ***RLA*** and ***FLA*** is implemented by Xilinx Bus Macros Communication primitives elements [9], placed into the fixed logic area ***FLA***. The Dynamic architecture layout in Virtex device is depicted in the Figure 4.

Partial reconfiguration of contexts requires communication fixed paths between ***FLA*** and ***RLA*** areas. As shown in Figure 4, the bus macro is a fixed routing bridge between fixed area and reconfigurable area, facilitating reliable communication. It is a pre-routed hard macro used to specify the exact routing channels and will not change from compilation to compilation of FPGA configurations [9]. For each of the different context configuration, there is absolutely no variation in the bus macro routing.

For dynamic architecture implementation of the RLC showed in Figures 3 and 4, the Modular Design Methology for Virtex device and ISE Xilinx tools [9] was used.

4.2 Temporal partitioning

The ***RLA*** area has *n* columns, depending on the device, being equivalent to a certain number of gates. If A_C is the necessary area (equivalents gates) to implement a certain controller ***Ctrl*** in the FPGA, then the number of contexts *N* is calculated by:

$$N \approx A_C \,/\, \textit{Number of gates available} \qquad (8)$$

The temporal partitioning ***Part*** of ***Ctrl*** is defined by:
$\boldsymbol{Part} = \{\boldsymbol{C}_0, \boldsymbol{C}_1, \boldsymbol{C}_2, \ldots, \boldsymbol{C}_{N-1}\}$ where $\boldsymbol{C}_i$, i=0,...*N-1*,
$\boldsymbol{C}_i = (\boldsymbol{TPN}_i, \boldsymbol{X}_i, \boldsymbol{Y}_i, \boldsymbol{F}_i, \boldsymbol{G}_i, \boldsymbol{A}_i, \boldsymbol{PI}_i)$,
$\boldsymbol{TPN}_i = (\boldsymbol{P}_i, \boldsymbol{T}_i, \boldsymbol{D}i, \boldsymbol{I}_i, \boldsymbol{O}_i, \boldsymbol{M}_{0i})$,
$\boldsymbol{C}_i$ is a subset of ***Ctrl***,
$\boldsymbol{C}_i$ is defined by grouping of places $\boldsymbol{P}_i \subset \boldsymbol{P}$ such that:
$\boldsymbol{P}_1 \cup \boldsymbol{P}_2 \cup \ldots \cup \boldsymbol{P}_N = \boldsymbol{P}$ and $\boldsymbol{P}_i \cap \boldsymbol{P}_j = \phi \; \forall$ i, j, i ≠ j,
The transition set $\boldsymbol{T}_i$ of $\boldsymbol{C}_i$ is give by:

$$\boldsymbol{T}_i = \{t \in \boldsymbol{T} \mid (\exists\, p \in \boldsymbol{P}_i \mid p \in \boldsymbol{I}_t \cup \boldsymbol{O}_t)\}. \qquad (9)$$

All elements of the sets ***X***, ***F***, ***P*** that affect the fire of $\boldsymbol{C}_i$ transitions, must be contained in $\boldsymbol{C}_i$, so that the places set $\boldsymbol{PI}_i$ is contained in places set of $\boldsymbol{C}_i$:

$$\boldsymbol{PI}_i = \{p \in \boldsymbol{P} \mid (p \notin \boldsymbol{P}_i \wedge \exists\, t \in \boldsymbol{T}_i \mid (p \in \boldsymbol{I}_t \text{ or p appears in } G_t)\} \qquad (10)$$

Observe that $\boldsymbol{C}_i$ uses the values of p' ∈ $\boldsymbol{PI}_i$ but only calculates the new marks of the places p ∈ $\boldsymbol{P}_i$.

The circuit partitioning uses a constructive algorithm derived from the method presented in [6]. This algorithm guarantees that the resultant area of the group of transitions and places, into each context, respects the ***RLA*** area restriction. Each context defined by expressions (9) and (10) is translated into VHDL files in the same way that in the section 3, with small modifications. The new VHDL code implements serial scan paths for input and output of internal state bits and ***X*** inputs in the contexts and the appropriated output signals for supply the ***OAR*** block placed in the fixed logic area ***FLA***.

4.3 Context switching

After timing partitioning, the context switching is performed by four steps. In this stage the bitstream configurations and context states are loaded and saved according to the partitioning schedule. For each new context in the FPGA reconfigurable area (***RLA***) the following control stages takes place:

1. The bitstream for the context circuit configuration $\boldsymbol{C}_i$ is loaded in the reconfigurable area ***RLA*** and the current context internal state is loaded using the serial scan path input. The load is synchronized by CLK_C.
2. $\boldsymbol{C}_i$ is then executed in the positive edge of CLK_C. This procedure up-to-date the current internal state and load into the ***OAR*** the values produced by the current context, in the negative edge of CLK_C.
3. If $\boldsymbol{C}_i$ is the final context of application, one CLK_S pulse is generated for up-to-date output signals ***Y***.
4. The new current internal state of context $\boldsymbol{C}_i$ is saved by serial scan path read, synchronized by CLK_C, and then, the step 1 is repeated for the next context $\boldsymbol{C}_{(i+1)\mathrm{mod}N}$, where mod*N* indicates module *N* operation.

The above cycle starts from context $\boldsymbol{C}_0$ and continues indefinitely along the controller operation. One Finite State Machine, $\boldsymbol{FSM_C}$, placed into the fixed logic area ***FLA*** implements this cycle. This $\boldsymbol{FSM_C}$ is also responsible for the controller power-on initialization and reset sequence. Saved internal context states are implemented by the use of blocks selectRAM components [10]. This resource implements a RAM memory into the ***FLA*** area, big enough for storage of the contexts states. The FPGA reconfiguration is made in the parallel SelectMap mode by one external fast configuration device [10] synchronized by $\boldsymbol{FSM_C}$.

4.4 Response time and reconfiguration time

Consider that T_{rec} is the ***RLA*** reconfiguration time, T_{srs} is the time to restore and save internal contexts state and T_{cycle} is the operation cycle time of an application then, the following relation is true for N contexts application:

$$T_{cycle} \approx N^*(T_{srs} + T_{rec}) \qquad (11)$$

Virtex-II FPGAs are very fast reconfigurable devices, with reconfiguration time between 1µs and 5µs per CLB. Clock signals around 100MHz are possible for these devices and very fast rate (1 bit/clock) of save/restore contexts state is possible. In general, control for industrial machines need operation cycles around 10ms, because of the mechanical nature of their components. This relative slow operation is a very interesting aspect for the proposed architecture, since the process execution cycles of complex controls can be guaranteed in small and cheap reconfigurable Virtex-II devices. Besides, each context can be executed in one clock cycle, because of the high parallelism degree of hardware contexts.

5 A partial reconfiguration design example

In order to demonstrate the architecture, an elevator control unit is used as example. The Figure 5 shows the resulting ***TPN*** controller model. The controller was split into two contexts $\boldsymbol{C}_0$ and $\boldsymbol{C}_1$, schedules in the reconfigurable device according to the architecture described before. The reconfiguration process was totally simulated on the Foundation ISE5.2 environment from Xilinx.

A Virtex-II XC2V80 device with 80,000 equivalent Xilinx gates has been used as the reconfigurable component. This device is organized as a set of 8 columns and 16 CLBs (*Configurable Logic Blocks*) per column, representing a density of 10,000 equivalent gates/column.

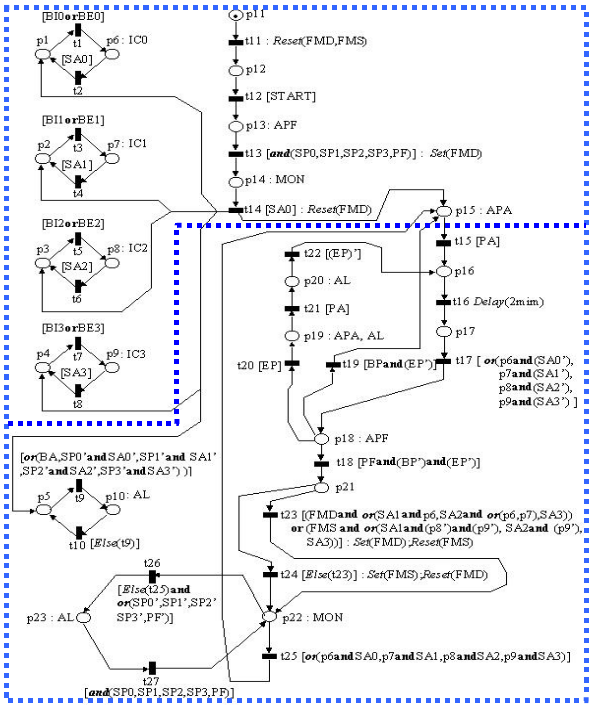
[BI0orBE0]
p1
t1
p6 : IC0
[SA0]
t2
[BI1orBE1]
p2
t3
p7: IC1
[SA1]
t4
[BI2orBE2]
p3
t5
p8: IC2
[SA2]
t6
[BI3orBE3]
p4
t7
p9: IC3
[SA3]
t8
p11
t11 : Reset(FMD,FMS)
p12
t12 [START]
p13: APF
t13 [and(SP0,SP1,SP2,SP3,PF)] : Set(FMD)
p14 : MON
t14 [SA0] : Reset(FMD)
p15 : APA
t15 [PA]
p16
t16 Delay(2mim)
p17
t17 [or(p6and(SA0'), p7and(SA1'), p8and(SA2'), p9and(SA3')]
t22 [(EP)']
p20 : AL
t21 [PA]
p19 : APA, AL
t20 [EP]
t19 [BPand(EP')]
p18 : APF
t18 [PFand(BP')and(EP')]
p21
[or(BA,SP0'andSA0',SP1'and SA1' ,SP2'andSA2',SP3'andSA3'))]
p5
t9
p10: AL
t10 [Else(t9)]
t23 [(FMDand or(SA1andp6,SA2and or(p6,p7),SA3)) or (FMS and or(SA1and(p8')and(p9'), SA2and (p9'), SA3))] : Set(FMD);Reset(FMS)
t24 [Else(t23)] : Set(FMS);Reset(FMD)
t26
[Else(t25)and or(SP0',SP1',SP2' SP3',PF')]
p23 : AL
p22 : MON
t25 [or(p6andSA0,p7andSA1,p8andSA2,p9andSA3)]
t27
[and(SP0,SP1,SP2,SP3,PF)]

Figure 5. TPN Model of elevator controller.

The complete configuration of the XC2V80 device, in the fastest way, takes 0.48 ms [10] or 0.06ms/column. So, it is possible to determine the time $T_{rec}(A)$ for reconfiguration of a given area A (set of columns) in this device, as follows:

$T_{rec}(\mathrm{A}) = \lfloor \mathrm{A}/10{,}000 \rfloor * 0.06\mathrm{ms}$ where A is given in equivalent gates. (12)

In this example, the ***RLA*** block occupies 4 columns, equivalent to 40,000 gates, corresponding to a reconfiguration time of 0,24ms for each context. The maximum number of interns state bits, 512, into each context is the same of the flip-flops in the area ***RLA***. Now, considering a clock frequency of 100 MHz, the operation cycle T_{cycle} given by equation (11) is $T_{cycle}=N^*(2^*512/100\mathrm{MHz} + 0.24\mathrm{ms})$. Thus, the total application time T_{cycle} can be expressed by:

$$T_{cycle} \geq N^*0.251\mathrm{ms} \quad \text{or}$$
$$N \leq T_{cycle}/0.251\mathrm{ms} \qquad (13)$$

The expression (13) defines the maximum number of contexts for high speed reconfiguration selectMap mode for a 100MHz clock. Now, taking into account an industrial operation cycle around of 10 ms, it is possible to implement until 40 contexts into the proposed architecture. The effective area need to support all 40 contexts into the ***RLA*** area of XC2V80 device would be 1,600,000 equivalents gates. In fact, if the circuit is not partitioned, the unique large context would require at least a XC2V1500 device to implement the application. The Figure 6 compares the small prices of Virtex II XC2V80 and of XC2V1500 devices in terms of cost.

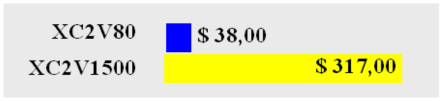

Figure 6. Virtex II devices cost [12].

This demonstrates the high economy in complex controls design into small device using a temporal partitioning for low speed applications. The table 1 shows the metrics extracted from synthesis for the contexts $\boldsymbol{C}_0$ and $\boldsymbol{C}_1$, based on the Foundation ISE 5.2 tool.

Table 1. Contexts elevator controller synthesis

Device: XC2V80 – Virtex II; **Tools**: ISE - 5.2			
	FLA	Context 1	Context 2
Input and Output Pins*	28	18	18
Area (Equivalent gates)	1922	679	1100
Slice FlipFlops** Used	73	29	40
18Kbits Block Select RAM***	2	--	--
Max. Clock. Freq (MHz).	162	234	220

*Total number of I/O Pins: 120; **Total number of slice flip-flops: 1024; ***Total number of 18Kbits Block Select RAM: 8.

The synthesis is successful at a high clock speed around 160MHz. The high parallelism of the FPGA allows the execution of contexts with up to 40K equivalents-gates (bit operation) in 0.251ms, including save/restore state and context reconfiguration time. This represents 159Million gates-operations per second. The most powerful CPU of Siemens PLC, the CPU 416, executes 12,5M bit-operations per second [13]. This metrics show the potential of such architecture even for higher performance application.

6 CONCLUSIONS AND FUTURE WORKS

In this work, RLC architecture of industrial control, based on a partially reconfigurable Virtex II device, has been presented as an alternative for PLC-based control. In this architecture, the application is specified in a Temporal Petri net (***TPN***) that introduces advantages for the formal verification of system properties. Besides, the Petri net have a powerful graphic representation of concurrence and asynchronous control systems and a similar semantics to others already traditional control specification tools in the industry (LAD and SFC). Also, a direct mapping method from Petri net specification into VHDL files bas been presented in order to improve the use of the effective area of a FPGA hardware, by temporal multiplexing of the controller context circuits.

An example, an elevator controller, has been presented and the synthesis results of the dynamic reconfiguration were presented. Although the reconfiguration time (context switching) can represent one bottleneck, it was demonstrated that, for low speed industrial applications, as a machine control, the proposed model can reach good results at a low cost, since that small devices can be used without loose of performance. The execution of splitting processes into contexts in small FPGAs, with high level of parallelism, can also be faster than in commercial PLCs, since PLC architectures are essentially sequential machines.

As future works, we intend to use more complex and real industrial applications; expand the ***TPN*** model to include more complex actions and data process; study architectures that combine microprocessors models and FPGA-based ***TPN*** model to incorporate the advantages of each approach in a based platform design for industrial controllers; develop tools that allow the automatic translation of currents LAD and SFC specification into a ***TPN*** model; contexts properties verification of the application specification and partitioning; automatic controller implementation.

7 REFERENCES

[1] Adamski, Marian; "SFC, Petri Nets and Application Specific Logic Controllers", Proceedings of IEEE, 1998.

[2] Chang, Douglas; Sadowska, Marek; "Partitioning Sequential Circuits on Dynamically Reconfigurable FPGAs", IEEE Transactions on Computer, vol. 48, no. 6, June 1999.

[3] Desrochers, Alan A.; Al-Jaar, Robert Y.; "Application of Petri Nets in Manufacturing Systems", IEEE PRESS, NewYork, 1995.

[4] Koo, Kyeonghoon; "Architectural Design of RISC Processor for Programmable Logic Controllers", Proceedings of the 9th CISL Winter Workshop, Sungwoo Resort, February 1996.

[5] Murata, Tadao; "Petri Nets: Properties, Analysis and Applications", Proceedings of the IEEE, vol. 77, no. 4, April 1989.

[6] Nascimento, Paulo S. B.; Lima, Manoel. E.; Maciel, Paulo R. M.; "Algorithm for Switching Context Temporal Partitioning Based in CDFG-Petri Net Model", Proceedings HPC2003, pp. 254-258.

[7] Xilinx, Virtex II Plataform FPGAs: Introduction and Overview, Datasheet DS031-1 (v 1.9), September 26, 2002.

[8] Xilinx, Virtex II Plataform FPGAs: Detailed Description, Datasheet DS031-2 (v 2.1.1), December 6, 2002.

[9] Xilinx, Two Flows for Partial Reconfiguration: Module Based or Small Bit Manipulations, Application Notes XAPP290 (v 1.0), May 17, 2002.

[10] Xilinx, Virtex II Platform FPGA User Guide, User Guide UG002 (V 1.5) December 2, 2002.

[11] Yakovlev, Alex; Gomes, Luis; Lavagno, Luciano; "Hardware design and Petri Nets", Kluwer Academic Publishers, Boston,2000.

[12] www.marshall.com, site for information of prices in November 2003.

[13] www.siemens.com/simatic-controller.