EFFICIENT SIGNAL PROCESSING USING SYNTACTIC PATTERN RECOGNITION METHODS

Andrew Koulouris, Theodore Andronikos, Christos Pavlatos, Alexandros Dimopoulos,
Ioannis Panagopoulos and George Papakonstantinou

School of Electrical and Computer Engineering
National Technical University of Athens
Athens, Greece
email:{andreas, tedandro, pavlatos, alexdem, ioannis, papakon}@cslab.ece.ntua.gr

ABSTRACT
This paper presents an optimal architecture for hardware implementation of Context-Free Grammar (CFG) parsers, which can be used to accelerate the performance of applications where response to real time signal processing is a crucial aspect, such as Electrocardiogram (ECG) analysis. Our architecture increases the performance by a factor of approximately two orders of magnitude compared to the pure software implementation, depending on the CFG. This speed up derives mainly from the hardware nature of the implementation, the innovative combinatorial nature of the circuit that implements the fundamental operation of the parsing algorithm and the underlying data representation. We further propose an automated synthesis tool that, given the specification of an arbitrary CFG and using the aforementioned hardware architecture in a template form, generates the HDL (Hardware Design Language) synthesizable source code of the hardware parser for the given grammar. The proposed architecture may be used for real time applications, e.g. natural languages interfaces. The generated source has been simulated for validation, synthesized and tested on a Xilinx FPGA (Field Programmable Gate Array) board.

KEY WORDS
Signal Processing, Syntactic Pattern Recognition, FPGA.

1 Introduction

Many signal processing problems require syntactic analysis. In some applications this comes as a direct result of the application nature itself, e.g., speech recognition systems. In others, like image analysis or biomedical signal analysis, this requirement arises indirectly from the method applied on the problem [1], [2]. In the latter case, it is a common practice to formulate the rules governing the relations between the patterns (e.g., relative position of objects in an image analysis system, or sequence of patterns in a biomedical signal analysis) by a grammar. Thus, the algorithms used for the syntactic analysis can significantly affect the efficiency of the overall signal processing system.
Context-free grammars (CFGs) combine expressive power and simplicity. They are powerful enough to describe the syntax of programming languages [1] (almost all programming languages are defined via context-free grammars) and simple enough to allow the construction of efficient parsing algorithms. Additionally, general context-free methods are exploited today in various application domains, such as speech recognition, natural language processing and image analysis [2], [3]. The large amount of data manipulated by those methods and today's requirement of conforming to very strict real-time constraints, turns the implementation of CFG-parsers that are optimized for performance more useful and imperative than ever. Two key factors that positively influence the CFG-parser's performance are undoubtedly the parsing algorithm itself as well as the implementation approach (software or hardware).
With respect to the first key factor, many modifications [4], [5] and improvements via parallelization [6] have been proposed for the well known classical parsing algorithms by Cocke-Younger-Kasami (CYK) [7], [8] and Earley [9]. The CYK algorithm requires the grammar to be in Chomsky Normal Form (CNF). Every GFG can be put in CNF, so, this requirement does not restrict the generality of the CYK algorithm. However, the transformation of a CFG G into an equivalent one in CNF cannot be carried out without a price: the size of the latter can be $O(|G|^2)$ [10], [11], [12].
With respect to the second key factor, Chiang & Fu [6] and Cheng & Fu [13] have presented designs using VLSI arrays for the hardware implementation of the above parsing algorithms, while Ibbara et al. [14] and Ra et al. [15] presented software implementations running on parallel machines. The hardware oriented approach was reinvigorated by presenting implementations in reconfigurable FPGA boards of the CYK algorithm by Ciressan [16] and Bordim et al. [17] and Earley's algorithm by Pavlatos [18], [19]. The architectures proposed so far do not fully exploit the available parallelization in the parsing algorithms, or they require excessive storage. The software approaches execute a part of the algorithms sequentially, thus they do not achieve the maximum possible speed-up. On the other hand, the hardware approaches must overcome the complexity that the operations involved in the parsing algo-

rithms impose, leading to increased storage needs. In order to relax the hardware complexity most of the proposed architectures implement CYK algorithm, whose basic operations are much simpler than those of Earley's. However, the advantages of CYK algorithm are actually compromised by the fact that a costly transformation of the grammar is required. The first approach to implement in FPGA the Earley's algorithm was given in [18]. Nevertheless, the approach proposed in our paper is based on mapping the fundamental operation of the algorithm to an efficient combinational circuit, achieving a further increase in performance by a factor of an order of magnitude, compared to [18] and two orders of magnitude compared to the software approach.In this paper, we address these problems by presenting a modified algorithm for CFG parsers and an efficient architecture implemented in reconfigurable hardware. The merits of our approach can be summarized in the followings:

- We propose a new parallel parsing architecture, which compared to the so far presented architectures, reduces the required number of processing elements by a factor of $\frac{n+1}{2}$, where n is the length of the input string, while maintaining a simple and elegant communication scheme.

- We propose a combinational circuit $C_{\otimes}$ that implements the fundamental operation $\otimes$ of the parsing algorithm [6] in extremely short time (execution time comparable to propagation delay of a few logic gates). By the use of appropriate number of $C_{\otimes}$ circuits (see next sections) we achieve to compute every cell of the aforementioned architecture in parallel. We distinct then a two-level parallelism, one local (cell-level) which corresponds to the execution of the operation, and one global (architecture level) which corresponds to the tabular form of the algorithm.

- We propose an innovative data representation that not only allows us to implement the operation in parallel, but also permits the storage of the grammar not to be necessary, since it is hardwired. Via the use of a binary equations system, the characteristics of the grammar are incorporated into the combinational circuit.

- Taking into consideration the hardware nature of the implementation, the presented architecture achieves a speed-up factor of two orders of magnitude compared to software approaches.

- Our implementation does not require the grammar to be ε-free, where ε is the empty string. Otherwise, this is a restriction because the number of rules of the equivalent ε-free grammar may be significantly increased. Additionally, by implementing Earley's algorithm no restriction in the form (i.e CNF) of the grammar is posed and, therefore, no costly transformation of the initial grammar is required.

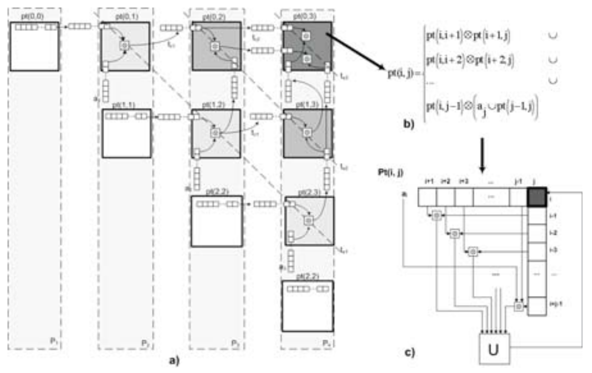

Figure 1. Overview of our architecture

- Finally, we use the proposed architecture in a template form, so as to present an automated tool that given the specification of an arbitrary CFG, generates the HDL (Hardware Design Language) synthesizable source code of the hardware parser for the given grammar.

The proposed architecture was used for the rapid development of an implementation for the analysis of Electrocardiogram(ECG)[20] signal, as well as a considerable number of applications such as the fast translation of natural language or programming language code in real time applications (e.g. natural languages interfaces) or internet applications (e.g. semantic nets) or more general in intelligence embedded systems augmenting the syntax with semantics rules [21], [22].

2 Overview of our approach

The proposed design is a hardware parallel parser for CFGs, which significantly improves the performance of existing CFG parsers by enhancing the architecture, by accelerating the execution time of the fundamental operation $\otimes$ of the parsing algorithm [6] and by proposing an innovative data representation.

Previously presented architecture [6] was based on the construction of a Parsing Table (PT) that can be parallelized with respect to the length of the input string n $(a_1 a_2 a_n)$ by computing at step k the cells pt(i, j) for which $j - i = k \geq 1$. Each PT cell pt(i,j), in order to be computed by the use of operation $\otimes$, requires all the cells that belong to the same row of the table and are left of it ($pt(i, m), (i + 1) < m < (j - 1)$), and all the states of the cells that belong to the same column of the table and are below of it (pt (n, j) ,($i + 1 < n \leq (j - 1)$)). In every execution step, each processing element computes one cell and then during the communication step transmits to other processing elements data that will be needed for next computation steps. Consequently, $\frac{(n+1)n}{2}$ processing elements are needed, since only the cells on and above the PT diagonal are used.

On the contrary, the proposed parallel architecture (Figure 1a) is based on n processing elements ($P_1, P_2, \ldots, P_n$), reducing therefore the FPGA real-estate needs by a factor of $\frac{n+1}{2}$. The processing elements implementation is presented in section 4. Additionally, the architecture communication scheme is improved as well, since no vertical transmissions are required; each processing element gradually computes the cells of a column. Every processing element computes a PT cell in each execution step ($t_{e1}, t_{e2}, \ldots, t_{en}$) and transmits this element, only horizontally, as well as all the cells that belong to the same row of the table and are left of it, during the communication step ($t_{c1}, t_{c2}, \ldots, t_{cn}$), to the processor right of it. Cells are computed by applying the operation $\otimes$, which is implemented by the use of a proposed combinational circuit $C_\otimes$, easily built based on the equations we can automatically create for every CFG, following the methodology analyzed in section 5.2. We noticed that the calculation process of each cell pt(i, j) can also be parallelized due to the nature of the equation in Figure 1b. All $\otimes$ operators can be executed in parallel by the use of multiple $C_\otimes$ circuits, whose output union produce the pt(i, j) (Figure 1c).
We distinct then a two-level parallelism, one local (cell-level) which corresponds to the execution of the operation, and one global (architecture level) which corresponds to the tabular form of the algorithm. The obvious importance of the $C_\otimes$ speedup, so as to increase the overall performance, lead us to propose i) a combinational circuit that implements the $C_\otimes$ in extremely short time (execution time comparable to propagation delay of a few logic gates) and ii) an efficient data representation, suitable for the presented circuit (Section 5.1).
Taking into consideration the hardware nature of the implementation, the presented architecture achieves a speed-up factor of two orders of magnitude compared to software approaches, as shown in section 6.

3 NOTATIONS

In this section the fundamental definitions necessary for presenting Earley's original algorithm are given, as well as the subsequent modifications imposed by various researchers to the latter algorithm and the herewith proposals.

Definition 3.1 *A Context Free Grammar (CFG) G is a quadruple G=(V, N, P, S), where: V is the finite set of symbols of the grammar, N is the set of nonterminal symbols, $T = V - N$ is the set of terminal symbols, $P \subseteq N \times V^\star$ is the finite set of rules and S is the start (nonterminal) symbol of the grammar.*

Rules have the form $A \rightarrow \alpha$, where $A \in N$ and $\alpha \in V^\star$; A is the left hand side symbol ($lhss$) and α is the string of the right hand side symbols ($rhss$) of the rule. A recognizer for G is an algorithm that takes as input a string of terminals $a_1 a_2 \ldots a_n$ and either accepts it or rejects it, depending on whether the string is a sentence of G or not.

Capital letters A, B, C,... denote nonterminals, lower case letters a, b, c, ... denote terminals, lower case letters u, v, w, ... denote strings of terminals, Greek letters α, β, γ, ... denote strings of terminal and nonterminal symbols and ε denotes the empty string. The notation $\beta \xrightarrow{*} \gamma$ denotes that γ can be derived from β by applying zero or more times some rules in P.
A nonterminal A is called nullable if $A \xrightarrow{*} \varepsilon$.

A string w of terminals is a *sentence* of G if $S \xrightarrow{*} w$. A grammar G is *reduced* [11] if for every $A \in N$ the following hold: (i) $S \xrightarrow{*} \alpha A \beta$, and (ii) there exists a sentence w such that $A \xrightarrow{*} w$.
Without loss of generality we assume that the underlying CFG is reduced. In this case, the appropriate measure of the size of G, denoted $|G|$, is $\sum_{r \in P} |r|$, where $|r|$ is the length (the number of right hand symbols) of rule r.
In 1970 Earley[9] presented a top-down parser, whose basic innovation was the introduction of a symbol called dot "$\bullet$" that does not belong to the grammar. The utility of the dot in a rule (now called dotted rule) is to separate the right part of the rule into two subparts. For the subpart on the left of the dot, it has been verified that it can generate the input string examined so far. However, for the subpart on the right of the dot, it still remains to check whether or not it can generate the rest of the input string. For any given rule $A \rightarrow \alpha$ there are exactly $|\alpha| + 1$ dotted rules, as many as the possible positions of the dot. When the dot is at the final position, the dotted rule is called $completed$. Prior to reading any input symbol, the dotted rules are in the form $A \rightarrow_\bullet \alpha$. As we start reading input symbols new dotted rules are created. If after reading the last input symbol a dotted rule of the form $S \rightarrow \alpha_\bullet$ exists, then the input string is a sentence of the grammar. The way in which dotted rules are created during the parsing of the input string can be efficiently formulated by the operator $\otimes$ that was first introduced by Chiang & Fu [6]. This operator takes as input a set of dotted rules and a terminal symbol or another set of dotted rules and produces a new set of dotted rules; its detailed definition is given in definition 5.1, section 5 where the analysis of the proposed circuit is presented.
The number of dotted rules of $P^\bullet$ is bounded above by $|G|$ (there may be less than $|G|$ dotted rules because rules of the form $A \rightarrow \bullet$ are not taken into consideration). $P^\bullet$ denotes the set of dotted rules that arise from P.

4 HARDWARE DESIGN IMPLEMENTATION

The proposed implementation is mainly based on the existence of a basic module, called Processing Element Module (PEM), described in Verilog HDL. Every processing element ($P_1, P_2, \ldots P_n$) shown in Figure 1 is implemented as a PEM instance. Every PEM calculates in each execution step ($t_{e1}, t_{e2}, \ldots t_{en}$) a cell of PT as shown in Figure 1b. In

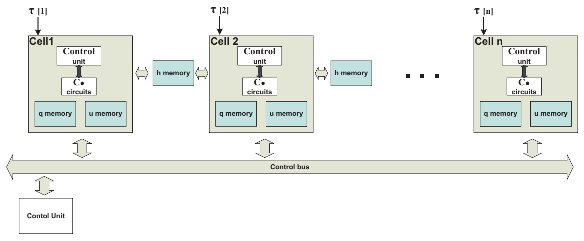

Figure 2. Architecture overview

order PEM to implement the architecture Figure 1b, a necessary number of $C_{\otimes}$ circuits are required, as well as two essential memory units - q and u. The first, q is utilized for the storage of the data that come from the same processing element, but belong to a lower cell of the PT. The second, u is utilized for the storage of data that come from cells on the left of the current PEM. Clearly a PEM Control Unit is needed to guide the operations performed by the PEM such as producing the proper addresses of memories, fetching data e.t.c. In each execution step the necessary data are fetched from memories q and u, processed by $C_{\otimes}$ circuits and their union (U) produces the output data. The output data, computed by a PEM, are stored in a memory - h memory - interconnecting two successive PEMs, as well as in its u memory so as to use it at the next execution steps. Each PEM sequentially executes repeatedly three operations, the execution step, the sending step and the receiving step. All the operations performed by the PEMs are guided through a Control bus, by the usage of a Control Unit. All the above, are shown in Figure 2.

An additional aim of this paper is to present a tool that based on the hardware architecture described above, takes as input the specifications of a CFG and automatically outputs the description of the CFG's parser in HDL synthesizable source code. In order to achieve this goal, we intended, during the design of the aforementioned implementation, to create architecture that is entirely independent from a specific CFG and is composed by general purpose components (memory, multiplexors, logic gates, e.t.c.) that can dynamically be modified to build a parser for any CFG. Therefore, the automated hardware generator may use this design as template, so as to produce a hardware parser for any CFG. The user of this tool provides the specifications of the CFG (in a text file) and the maximum input string length n. It is significant to point out that the architecture (Figure 2) of the design does not change but based on the user's specifications the tool produces the PEMs and adds them to the properly modified template of the architecture. The hardware template has been implemented in synthesizable Verilog in the XILINX ISE 7.1i environment [23] while the automated synthesis tool has been implemented in VISUAL C++ 6.0 [24].

5 THE COMBINATIONAL CIRCUIT $C_{\otimes}$

A combinational circuit is a generalized gate of $m\ inputs$ and $n\ outputs$. The circuit is constructed using the binary $equations$, describing the relations between the input and output values. Thus in our case in order to build a combinational circuit for operator $\otimes$, we must first define the inputs and outputs of the circuit, as well as the relative binary equations. Recall from the previous section that operator $\otimes$ operates over a set of dotted rules and a terminal symbol of the grammar $(H = Q \otimes (U \bigcup a))$. Consequently our inputs/outputs should be a proper encoding of these sets, as well as a proper encoding of the input string symbols. With the term $proper$ we mean a binary encoding, which can describe at any given moment which dotted rules and which terminal symbols are contained in the sets respectively. The data representation as well as the binary equations are presented in the following subsections.

5.1 Data Representation

The operator $\otimes$ $(H = Q \otimes (U \bigcup a))$ is applied on two sets of dotted rules (Q, U) and a terminal symbol (a) and produces a set of dotted rules (H). We consider that bit vectors $\vec{q}$ and $\vec{u}$ control the input bit-vectors corresponding to sets Q and U, while the output bit-vector $\vec{h}$ controls the bits corresponding to set H. Furthermore since our construction follows the definition 5.1, shown in the next subsection, we shall use two auxiliary bit vectors $\vec{h}_{12}$ and $\vec{h}_{Y}$. Bit vector $\vec{h}_{12}$ corresponds to the set of dotted rules resulting from the function $h_1 \bigcup h_2$ of definition 5.1, while bit vector $\vec{h}_{Y}$ corresponds to set h_Y. Notice that we use these two bit vectors just as a reference to the internal signals of the circuit. In every bit-vector representation, each bit denotes the presence (1) or absence (0) of a specific dotted rule in the set. Since there are $|r|$ dot positions for each one of the $|P|$ rules, the length of the required bit vector is of $\sum_{r \in P}(|r|) = |G|$ bits. All $\vec{q}$, $\vec{u}$, $\vec{h}_Y$, and $\vec{h}_{12}$ are bit vectors of that size. It must be defined that each rule r belonging to the set P is denoted by the symbolism r_i, where i $(0 \leq i \leq |P| - 1)$ is the enumeration of the rule. In order to denote a dotted rule r, the above symbolism is extended to r_i^j, where j $(0 \leq j \leq |rhss|)$ is the position of the dot in the rule r_i. Each bit b_k in the bit-vector controls the existence of a dotted rule r_i^j. In order to follow the notation shown in figure 3, k should be equal to $\sum_{r_0}^{r_{i-1}}(|rhss|) + j$. Additionally, the set $Predict(N)$ can be represented by a bit-vector of this size. The latter is called $\vec{p}$ and is incorporated in $C_{\otimes}$ since it's essential in the calculation of $\vec{h}_Y$.
Similarly, every terminal symbol is represented in a bit vector $\vec{\tau}$ of size $|T|$, where T is the set of terminal symbols. In this bit-vector each bit denotes the presence (1) or absence (0) of a specific symbol. Clearly, every bit-vector $\vec{\tau}$, has only one bit valued (1) as it represents only one terminal symbol. Consequently, the input string is represented by the use of n bit-vectors of size $|T|$. Figure 3 illustrates our encoding schema.

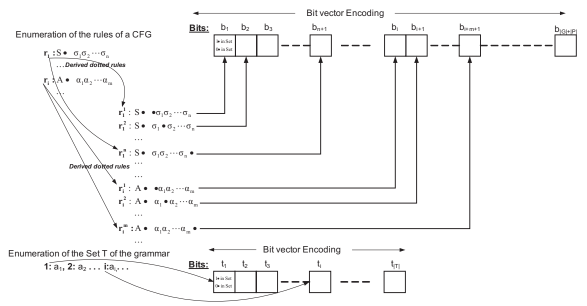
Bit vector Encoding
Bits:
b_1 b_2 b_3 b_{n+1} b_i b_{i+1} b_{i+m+1} $b_{|G|+|P|}$
1• in Set
0• in Set
Enumeration of the rules of a CFG
r_1 : S • $\sigma_1\sigma_2\cdots\sigma_n$
...Derived dotted rules
r_i : A • $\alpha_1\alpha_2\cdots\alpha_m$
r_1^1 : S • $\bullet\sigma_1\sigma_2\cdots\sigma_n$
r_1^2 : S • $\sigma_1\bullet\sigma_2\cdots\sigma_n$
r_1^n : S • $\sigma_1\sigma_2\cdots\sigma_n\bullet$
Derived dotted rules
r_i^1 : A • $\bullet\alpha_1\alpha_2\cdots\alpha_m$
r_i^2 : A • $\alpha_1\bullet\alpha_2\cdots\alpha_m$
r_i^m : A • $\alpha_1\alpha_2\cdots\alpha_m\bullet$
Enumeration of the Set T of the grammar
1: a_1, 2: a_2 . . . i:a_i, . . .
Bit vector Encoding
Bits:
t_1 t_2 t_3 t_i $t_{|T|}$

Figure 3. Data Representation

Adopting this encoding schema, our proposed circuit takes as input 2 bit vectors of size $|G| + |P|$ (Q,U) and one of $|T|$ (a) and outputs a bit vector of size $\big(|G| + |P|\big)$ (H).

5.2 Building the binary equations

In order to explain the methodology followed to built the binary equations, it is necessary to firstly present the definition of the operator $\otimes$.

Definition 5.1 *Given the terminal symbol a and the sets of dotted rules Q and U, operator $\otimes$ is defined as follows:*

- $h_1(Q,a) = \{A \to \alpha a\beta_\bullet\gamma | A \to \alpha_\bullet a\beta\gamma \in Q$ *and* $\beta \xrightarrow{*} \varepsilon\}$
- $h_2(Q,U) = \{A \to \alpha E\beta_\bullet\gamma | A \to \alpha_\bullet E\beta\gamma \in Q, \beta \xrightarrow{*} \varepsilon$ *and* $E \to \zeta_\bullet \in U\}$

Given a set X of dotted rules, the set $h_Y(X)$ is defined as follows:

- $h_Y(X) = \{B \to \delta C\xi_\bullet\eta | D \to \kappa_\bullet \in X,\ C \in Predecessors(D), B \to \delta_\bullet C\xi\eta \in Predict(N)$ *and* $\xi \xrightarrow{*} \varepsilon\}$
- $Q \otimes a = h_1(Q,a) \cup h_Y(h_1(Q,a))$ *and*
- $Q \otimes U = h_2(Q,U) \cup h_Y(h_2(Q,U))$

Definition 5.2 *Given a nonterminal symbol A, the sets* Predecessors(A), Descendants(A) *and* Predict(A) *are defined as follows:*

- Predecessors(A) $= \{B \in N | B \xrightarrow{*} A\}$, *the intuition being that starting from the nonterminal B, there is a sequence of rules that eventually. generate the nonterminal A as the only rhss.*
- Descendants(A) $= \{B \in N | A \xrightarrow{*} B\}$, *this set contains all the nonterminals that can be derived from A.*
- Predict(A) $= \{B \to \gamma \bullet \delta \in P^\bullet | A \xrightarrow{*} B\alpha$ and $\gamma \xrightarrow{*} \varepsilon\}$
- Predict(N) $= \bigcup_{A \in N}$ Predict(A)

It is clearly seen from the above definition that the result of $Q \otimes a$ and $Q \otimes U$ is a set of dotted rules, simply constructed using sets h_1, h_2 and h_Y. Therefore the problem is reduced to that of analyzing the construction of the latter sets, that will be examined separately.
If there is a dotted rule that belongs to Q and right to the dot is the terminal symbol a then this dotted rule is added to set $h_1(Q,a)$ but first the dot is moved one position to the right. If right to a is a string β that can produce the empty string ε then the dot is moved right to β.
Respectively,if there is a dotted rule that belongs to Q and right to the dot is a non terminal symbol E, for which there is a $completed$ dotted rule in U, then the dotted rule belonging to Q is added to set h_2 but first the dot is moved one position to the right. If right to E is a string β that can produce the empty string ε then the dot is moved right to β.
A rule r_i^j of the form $B \to \delta C\xi_\bullet\eta$, belonging to set X, is added to the set $\vec{h}_Y(X)$ if a) rule r_i^{j-1} belongs to set $Predict(N)$ b) the symbol C is a $Predecessor$ of D and c) rule D is completed in set X.If right to C is a string ξ that can produce the empty string ε then the dot is moved right to ξ.
The derived binary equations which connect the inputs to the outputs of the combinational circuit $C_\otimes$, are depended on the given grammar. In the case of $C_\otimes$ the output of the circuit is connected to the necessary inputs through a binary function described as: $\vec{h}(i) = \vec{h}_{12}(i) + \vec{h}_Y(i)$, where $\vec{h}_{12}()$ is a function of $\vec{q}, \vec{u}, \vec{\tau}$ and $\vec{h}_{12}$, while $\vec{h}_Y()$ is a function of $\vec{h}_{12}$, $\vec{h}_Y$ and $\vec{p}$. In the two aforementioned functions, the value of a bit of the output bit-vector depends on previously calculated bit values of the same vector; of course this fact does not interfere the combinational nature of the circuit.
Our tool extracts automatically these binary equations for any given CFG and embeds them in the produced HDL source of our architecture.
Here, the possible forms the binary functions may have, will be outlined. Firstly, function $\vec{h}_{12}()$ will be examined.
In every dotted rule r_i^j, that is not $completed$, the symbol right to the dot might be either a terminal or a nonterminal symbol. These are the two case, that are going to be examined:

- For a dotted rule r_i^j of the form $A \to \alpha_\bullet a\beta$, belonging to $\vec{q}$ in position $s-1$, the dotted rule r_i^{j+1} $A \to \alpha a_\bullet\beta$ will be added to $\vec{h}_{12}$ in position s only if the terminal symbol a is equal to $\vec{\tau}(k)$, where $\vec{\tau}(k)$ is the symbol of the input string to be recognized. Bitwise, the function $\vec{h}_{12}()$ is:
$$\vec{h}_{12}(s) = \vec{q}(s-1) \cdot \vec{\tau}(k) \tag{1}$$
This means that if bit $\vec{q}(s-1)$ is 1 (existence of $A \to \alpha_\bullet a\beta$) and bit $\vec{\tau}(k)$ which refers to a is 1, then

the dotted rule $A \rightarrow \alpha a_\bullet \beta$ should be added to $\vec{h}_{12}$ (set bit $\vec{h}_{12}(s)$).

- Let's consider the dotted rule r_i^j of the form $B \rightarrow \gamma_\bullet C\delta$, belonging to $\vec{q}$ in position $s-1$. Since C is a non terminal symbol, it may be the lhss in some rules. Let these rules be $(r_{c_1}) : C \rightarrow \zeta, (r_{c_2}) : C \rightarrow \eta, \ldots, (r_{c_l}) : C \rightarrow \rho$. If dotted rules $C \rightarrow \zeta_\bullet, C \rightarrow \eta_\bullet \ldots, C \rightarrow \rho_\bullet$ are controlled by bits in positions $v_1, v_2, \ldots, v_l$,then rule r_i^{j+1} of the form $B \rightarrow \gamma C_\bullet \delta$ should be added to $\vec{h}_{12}$ in position s only if at least one of the bits $\vec{u}(v_1), \ldots, \vec{u}(v_l)$ is set. Bitwise, the function $\vec{h}_{12}()$ is:

  $$\vec{h}_{12}(s) = \vec{q}(s-1) \cdot (\vec{u}(v_1) + \ldots + \vec{u}(v_l)) \qquad (2)$$

  This means that if bit $\vec{q}(s-1)$ is 1 (existence of $B \rightarrow \gamma_\bullet C\delta$) and at least one of the bits $\vec{u}(v_1), \ldots, \vec{u}(v_l)$ (existence of at least one of $C \rightarrow \zeta_\bullet, C \rightarrow \eta_\bullet \ldots, C \rightarrow \rho_\bullet$) is 1 then the dotted rule $B \rightarrow \gamma C_\bullet \delta$ should be added to $\vec{h}_{12}$ (set bit $\vec{h}_{12}(s)$). Additionally if C produces the ε-string, the term $\vec{h}_{12}(s)$ should be set to $\vec{h}_{12}(s-1)$, which means that merely the existence of $B \rightarrow \gamma_\bullet C\delta$ is sufficient for the addition of $B \rightarrow \gamma C_\bullet \delta$ in $\vec{h}_{12}$.

Secondly, function $\vec{h}_Y()$will be examined. In order a dotted rule r_i^j of the form $B \rightarrow \delta C_\bullet \xi\eta$ to be added in position s of $\vec{h}_Y$, the followings should be satisfied:

1. The dotted rule r_i^{j-1} of the form $B \rightarrow \delta_\bullet C\xi\eta$ should be set in set $\vec{p}$ ($Predict(N)$), i.e. $\vec{p}(s-1) = 1$

2. For at least one of the symbols belonging to set $Descendant(C) = \{D_1, \ldots, D_d\}$ there should be a $completed$ dotted rule with this symbol as $lhss$ in set $\vec{h}_{12}$, i.e. at least one $D_i \rightarrow \kappa_\bullet$ belongs to $\vec{h}_{12}$, where $1 \leq i \leq d$.

Lets assume that symbols $D_1, \ldots, D_d$ are lhss symbols in π rules, which leads to the fact that there are π *completed* dotted rules, controlled by bits $w_1, \ldots, w_\pi$. Then $\vec{h}_Y()$ is given by the following equation:

$$\vec{h}_Y(s) = \vec{p}(s-1) \cdot (\vec{h}_{12}(w_1) + \ldots + \vec{h}_{12}(w_\pi)) \qquad (3)$$

This means that if bit $\vec{p}(s-1)$ is 1 (existence of $B \rightarrow \delta_\bullet C\xi\eta$ in $Predict(N)$) and at least one of the bits $\vec{h}_{12}(w_1)$, $\ldots$, $\vec{h}_{12}(v_\pi)$ (existence of $D_i \rightarrow \kappa_\bullet$) is 1 then the dotted rule $B \rightarrow \gamma C_\bullet \delta$ should be added to $\vec{h}_Y$ (set bit $\vec{h}_Y(s)$). Additionally if C produce ε-string $\vec{h}_Y(s)$ should be set to $\vec{h}_Y(s-1)$ which means that merely the existence of $B \rightarrow \delta_\bullet C\xi\eta$ is sufficient for the addition of $B \rightarrow \delta C_\bullet \xi\eta$ in $\vec{h}_Y$.
Despite the definition 5.1 that checks if symbols produce the ε-string after moving the dot, the proposed equations checks these symbols before moving the dot, without compromising the outcome correctness.

An abstract implementation of the aforementioned architecture is illustrated in figure 4, where only the crucial connections are shown.

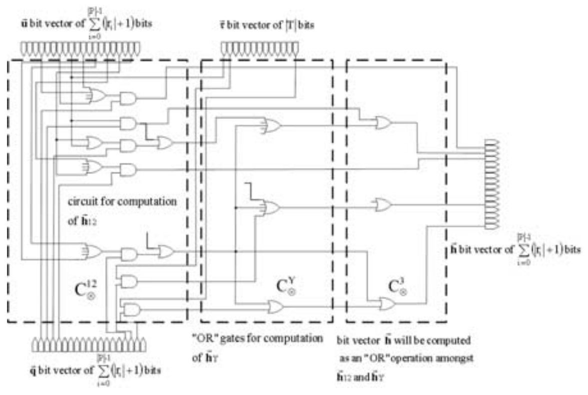
$\vec{u}$ bit vector of $\sum_{i=0}^{|P|-1}(|r_i|+1)$bits
$\vec{r}$ bit vector of $|T|$bits
circuit for computation of $\vec{h}_{12}$
$C^{12}_{\otimes}$
$C^{Y}_{\otimes}$
$C^{3}_{\otimes}$
$\vec{h}$ bit vector of $\sum_{i=0}^{|P|-1}(|r_i|+1)$bits
$\vec{q}$ bit vector of $\sum_{i=0}^{|P|-1}(|r_i|+1)$bits
"OR" gates for computation of $\vec{h}_Y$
bit vector $\vec{h}$ will be computed as an "OR" operation amongst $\vec{h}_{12}$ and $\vec{h}_Y$

Figure 4. Abstract Implementation of the Proposed Architecture

6 THE SIGNAL PROCESSING EXAMPLE - EXPERIMENTAL RESULTS

The ECG is a biosignal which is generated by the electrical activity of the human heart that is transmitted to the body surface. The ECG is routinely used in clinical practice and due to the large number of ECGs analyzed in daily basis, it is worthwhile to automate and accelerate the process to the maximum extent possible. In Syntactic Pattern Recognition, the task of recognition is essentially reduced to that of parsing a linguistic representation of the patterns to be recognized with a parser that utilizes a certain grammar, called "pattern grammar" [1]. The pattern grammar describes the patterns to be recognized in a formal way, and the formulation and parsing of the grammar are always the crucial subproblems in a pattern recognition application that is to be tackled by the syntactic approach. In the case of ECGs, where we have a large number of different morphologies of the patterns, where added morphologies can be found due to the noise, and where measurements of the various parameters have to be performed, powerful grammars capable of describing syntax as well as well as semantics are needed as a model for the formulation of a pattern grammar. Due to their descriptive power attributes grammars are usually [20] selected and used as the model for the formulation of a pattern grammar for ECGs. We used the automated synthesis tool to generate a hardware parser for the syntactic part of the attribute grammar presented in [20] G_{ECG}.The results of the recognition time of the grammar tested for various lengths of input string and implementations (software - hardware) are shown in Figure 5. In [20] the alphabet of symbols $T = K+, K-, E, \Pi$ has been adopted for encoding the ECG waveforms, where $K+$ denotes positives peaks, $K-$ negative peaks, E straight line segments and Π parabolic segment. Thus an ECG waveform is lin-

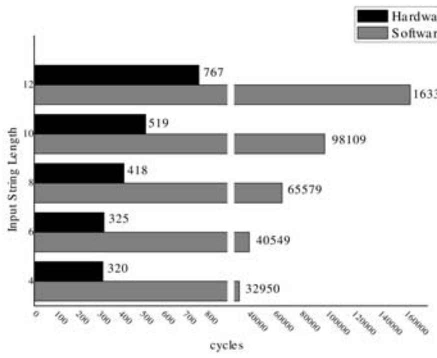

Figure 5. Performance evaluation of ECG grammar for different lengths of input string

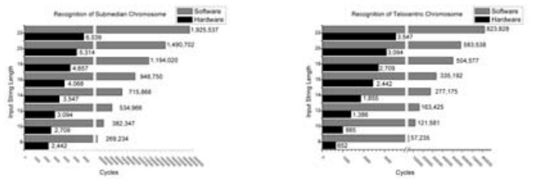

Figure 6. Performance evaluation of G_{chrom} parsers for both types of chromosomes and different lengths of input string

guistically represented as a sting of symbols from the alphabet T. Each symbol is associated with the values of the corresponding attributes. The CFG G_{ECG} consists of 34 syntactic rules with maximum length 7 symbols (terminals or non-terminals).

A parser for the abovementioned grammar was implemented in hardware using the proposed architecture and in software as well. The performance in both cases (hardware/software) is measured in clock cycles. Provided that the technology used for the hardware implementation is the same for both the FPGA and the microprocessor (if we run the software implementation on a conventional microprocessor) we can safely use the number of required clock cycles as measurement of the efficiency of the two approaches (hardware versus software). Additionally, the computational power of microprocessors used in embedded system, is comparable to that of an FPGA. Hence the performance in all implementations is measured using the number of required clock cycles, so as to purely compare the architecture, regardless of the technology used. The clock cycles in the software implementations refer to those needed by the microprocessor to execute the algorithm.

Our implementation was also tested in various grammars of different sizes, used in real life applications [25], [26]. By the use of the grammar G_{chrom} presented in [2], both types of chromosomes can be recognized. The syntactic rules of this grammar are 18 with maximum length 3, the non-terminal symbols are 8, the terminal symbols are 5 and $|G_{chrom}|$=32. Furthermore, the performance evaluation for two CFGs G_1 containing 8 rules where $|G_1|$=18, and G_2 containing 30 rules where $|G_2|$=55 is presented in Figure 7. G_1 and G_2 are subsets of Java Programming Language [27].

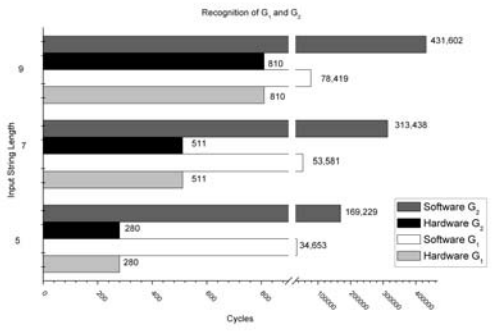

Figure 7. Performance Evaluation of G_1, G_2

Apparently, the performance of our implementation is two orders of magnitude greater than the software approach, for all tested grammars. The reason why the performance in hardware implementation remains the same in cases of G_1 and G_2 is due to the fact that the propagation delay of circuit C for both grammars is less than a clock cycle. In cases where the gate level of circuit $C_{\otimes}$ inputs propagation delay greater than a clock cycle then the overall performance will decrease by a factor of 16 % (for every extra clock cycle needed). The small decrease in the performance is explained by the fact that only the $C_{\otimes}$ part of the whole architecture is delayed.

7 CONCLUSION - FUTURE WORK

This paper presents an innovative architectural design for Signal Processing applications based on a Context-Free Grammar (CFG) parser and an innovative automated synthesis tool that exploits its characteristics for the hardware implementation of applications that require parsing to enhance their performance by integrating syntactic knowledge via CFGs. Our implementation increases the performance by a factor of approximately two orders of magnitude compared to the pure software implementation, depending on the CFG. Our future work is focused on extending the proposed architecture so as to handle semantic rules (Attribute Grammar - AG) as well. This work is a part of a project[1] for developing a platform (based on AGs) in order to automatically generate special purpose embedded systems. The application area will be that of signal processing, syntactic pattern recognition, intelligent user interfaces and Artificial Intelligence (AI) applications us-

[1]This work is co - funded by the European Social Fund (75%) and National Resources (25%) - the Program PENED 2003

ing software hardware co-design techniques. The parallel parser (FPGA) will work in coordination with a RISC microprocessor that will handle the attribute evaluation process following the approach presented in [21]. Alternatively, the whole implementation, both microprocessor and parser may be mapped on a single FPGA, by the use of a Soft Processor Core, such as "$Microblaze$" microprocessor that can be embedded in FPGA boards.

References

[1] Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation, and Compiling*, volume I: Parsing of *Series in Automatic Computation*. Prentice Hall, Englewood Cliffs, New Jersey, 1972.

[2] K. S. Fu. *Syntactic Pattern Recognition and Applications*. Prentice-Hall, Englewoods Cliffs, 1982.

[3] Elisabeth Andr. The generation of multimedia presentations. In H. Moisl R. Dale and H. Somers, editors, *A Handbook of Natural Language Processing: techniques and applications for the processing of language as text*, pages 305–327. Marcel Dekker Inc., 2000.

[4] Susan L. Graham, Michael A. Harrison, and Walter L. Ruzzo. An improved context-free recognizer. *ACM Trans. Program. Lang. Syst.*, 2(3):415–462, 1980.

[5] Walter Lawrence Ruzzo. *General context-free language recognition*. PhD thesis, Univ. of California, Berkeley, 1978.

[6] Y.T. Chiang and K.S. Fu. Parallel parsing algorithms and vlsi implementations for syntactic pattern recognition. *T-PAMI*, 6:302–314, 1984.

[7] T. Kasami. An efficient recognition and syntax analysis for context-free languages. *Science Report AF CRL-65-758*, Air Force Cambridge Research Laboratory Bedford,Mass., 1965.

[8] D. H. Younger. Recognition and parsing of context-free languages in n^3. *Information and Control*, 10:189–208, 1967.

[9] Jay Earley. An efficient context-free parsing algorithm. *Commun. ACM*, 13(2):94–102, 1970.

[10] M. A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, Reading, Mass., 1978.

[11] Alica Kelemenová. Complexity of normal form grammars. *Theoretical Computer Science*, 28(3):299–314, February 1984.

[12] Norbert Blum and Robert Koch. Greibach normal form transformation revisited. *Inf. Comput*, 150(1):112–118, 1999.

[13] H. D. Cheng and K.S. Fu. Algorithm partition and parallel recognition of general context-free languages using fixed-size vlsi architecture. *Pattern Recognition*, 19(5):361–372, 1986.

[14] Oscar H. Ibarra, Ting-Chuen Pong, and Stephen M. Sohn. Parallel recognition and parsing on the hypercube. *IEEE Trans. Comput.*, 40(6):764–770, 1991.

[15] Dong-Yul Ra and Jong-Hyun Kim. A parallel parsing algorithm for arbitrary context-free grammars. *Inf. Process. Lett.*, 58(2):87–96, 1996.

[16] C. Ciressan et al. An FPGA-based coprocessor for the parsing of context-free grammars. In *IEEE Symp. on FCCM*, pages 236–245. Computer Society Press., 2000.

[17] J. Bordim, Y. Ito, and K. Nakano. *IEICE Transactions Information & Systems*, E-86D(5):803–810, May 2003.

[18] C. Pavlatos, I. Panagopoulos, and Papakonstantinou G. A programmable pipelined coprocessor for parsing applications. In *Workshop on Application Specific Processors (WASP) CODES, Stockholm*, September 2004.

[19] C. Pavlatos, A. Koulouris, and Papakonstantinou G. Hardware implementation of syntactic parsing recorgnition algorithms. In *IASTED SPPRA, Rhodes,Greece*, 2003.

[20] P. Trahanias and E.Skordalakis. Syntactic pattern recognition of the ECG. *IEEE Transactions on PAMI*, 12, 1990.

[21] I. Panagopoulos, C. Pavlatos, and G. Papakonstantinou. An embedded system for artificial intelligence applications. *International Journal of Computational Intelligence*, 2004.

[22] Ioannis Panagopoulos, Christos Pavlatos, and George Papakonstantinou. An embedded microprocessor for intelligent control. *Journal of Intelligent and Robotic Systems*, 42(2):179–211, February 2005.

[23] Xilinx official website. www.xilinx.com.

[24] Microsoft official website. www.microsoft.com.

[25] G. Belforte, R. De Mori, and F. Ferraris. A contribution to the automatic processing of electrocardiograms using syntactic methods. *IEEE Transcactions on Biomedical Engineering*, 26(3), March 1979.

[26] J. Udupa and I. Murthy. *IEEE Transcactions on Biomedical Engineering*, 27(7), July 1980.

[27] Java technology. java.sun.com.