Hardware Natural Language Interface

C. Pavlatos, A. C. Dimopoulos, G. Papakonstantinou
National Technical University of Athens
Department of Electrical and Computer Engineering
Zografou 15773, Athens
Greece

Abstract. In this paper an efficient architecture for natural language processing is presented, implemented in hardware using FPGAs (Field Programmable Gate Arrays). The system can receive sentences belonging to a subset of Natural Languages (NL) from the internet or as SMS (Short Message Service). The recognition task of the input string uses Earley's parallel parsing algorithm and produces intermediate code according to the semantics of the grammar. The intermediate code can be transmitted to a computer, for further processing. The high computational cost of the parsing task in conjunction with a possible large amount of input sentences, to be processed simultaneously, justify the hardware implementation of the grammar (syntax and semantics). An extensive illustrative example is given from the area of question answering, in order to show the feasibility of the proposed system.

1 Introduction

Natural Language (NL) processing is a very attractive method of human-computer interaction and may be applied to a considerable number of fields such as intelligent embedded systems, intelligent interfaces, learning systems, etc [2], [3]. It is clear that automatically extracting linguistic information from a text can be an extremely powerful method for NL processing systems.

In this paper a hardware natural language interface is presented, using FPGAs (Field Programmable Gate Array). The system can receive sentences belonging to a subset of Natural Languages (NL) from the internet or as SMS (Short Message Service). In Fig. 4 a possible application is shown. Clients of a firm are asking questions which have to be answered very fast. The recognition task of the input string uses Earley's [1] parallel parsing algorithm and produces intermediate code according to the semantics of the grammar. The intermediate code can be transmitted to a computer, for further processing. The high computational cost of the parsing task, in conjunction with a possible large amount of input sentences to be processed concurrently, can dramatically speed-up the processing, due to the hardware

implementation of the grammar (syntax and semantics). An extensive illustrative example is given from the area of question answering [4], in order to show the feasibility of the proposed system.
In the example given, the well known parallel parsing algorithm of Early [1], [5] has been used, based on the implementation proposed in [10]. When the syntactic recognition of the input sentence is completed, using the created parse tree, the semantics are evaluated and the FPGA sends the intermediate code generated to an abstract data-management machine that has access to a data-base, in order to produce the final result (answer). The intermediate code consists of commands and their parameters. The FPGA may receive the questions either via internet or via SMS receiver, since both interfaces may be implemented on the FPGA. In the second case an extra device (SMS receiver) is necessary.
The proposed architecture has been implemented in synthesizable Verilog in the XILINX ISE 8.2 [6] environment while the generated source has been simulated for validation, synthesized and tested on a Xilinx SPARTAN 3E FPGA.

2 The Hardware Parser

2.1 Theoretical Background

A Context Free Grammar [7] (CFG) is a quadruple G = (N, T, R, S), where N is the set of non-terminal symbols, T is the set of terminal symbols, R is the set of grammar rules (a subset of N x(N ∪ T)* written in the form A→α, where A ∈ N and α ∈ (N ∪ T)*) and S (S ∈ N) is the start symbol (the root of the grammar). We use capital letters A, B, C… to denote non terminal symbols, lowercases a, b, c… to denote terminal symbols and Greek lowercases α, β, γ... for (N ∪ T)* strings, λ is the null string and V = N ∪ T is called vocabulary. A→a means that a can derive from A after the application of one or more rules.
Let S → α, (α∈T*) be a derivation in G. The corresponding derivation (parsing) tree is an ordered tree with root S, leaves the terminal symbols in α, and nodes the rules that are used for the derivation process.
The process of analyzing a string for syntactic correctness is known as parsing. A parser is an algorithm that decides whether or not a string $a_1a_2a_3…a_n$ (of length n) can be generated from a grammar G and simultaneously constructs the derivation (or parse) tree.
An Attribute Grammar [8] (AG) is based upon a CFG. An AG is a quadruple AG = {G, A, SR, d} where G is a CFG, A = ∪A(X) where A(X) is a finite set of attributes associated with each symbol X ∈ V. Each attribute represents a specific context-sensitive property of the corresponding symbol. The notation X.a is used to indicate that attribute a is an element of A(X). A(X) is partitioned into two disjoint sets; the set of synthesized attributes AS(X) and the set of inherited attributes AI(X). Synthesized attributes X.s are those whose values are defined in terms of attributes at descendant nodes of node X of the corresponding semantic tree. Inherited attributes X.i are those whose values are defined in terms of attributes at the parent and

(possibly) the sibling nodes of node X of the corresponding semantic tree. The start symbol does not have inherited attributes. Each of the productions $p \in R$ (p: $X_0 \rightarrow X_1 X_2 X_n$) of the CFG is augmented by a set of semantic rules SR(p) that define attributes in terms of other attributes of terminals and on terminals appearing in the same production. The way attributes will be evaluated depends both on their dependencies to other attributes in the tree and also on the way the tree is traversed. Finally d is a function that gives for each attribute a its domain d (a).

In the case of the illustrative example given in this paper (based on the one of ref. [4]) a subset of NL is given in the formalism of AG where the semantics are described using a synthesized attribute called 'output' for each non-terminal symbol. The only operation for the semantic rules needed, between the attributes of the non-terminals, is *conc* (par_1, ..., par_n), which stands for the concatenation of strings par_1, ..., par_n that are values of the attributes of the non-terminal symbols of the syntax rule.

2.2 The Parsing Algorithm

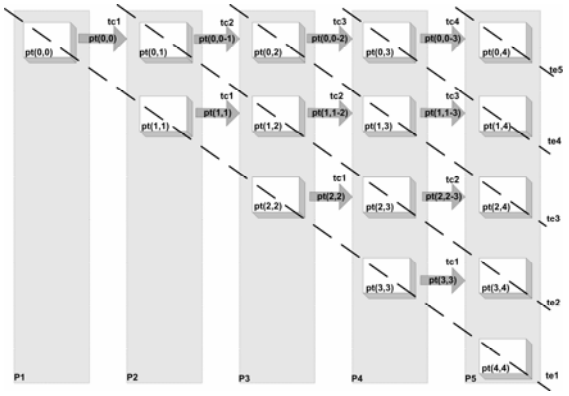

Fig. 1. The parallel architecture for the construction of Parsing Table

The parsing task may be reduced to the procedure of filling a two dimension table (parsing table: pt ()). Chiang & Fu [5] proved that the construction of the parsing table can be parallelized with respect to the length of the input string n, by computing at step k the cells pt(i, j) for which j-i=k≥1. Only the elements on or above the diagonal are used. In [9] a parallel architecture (see Fig. 1) has been presented that uses n+2 elements to compute the parse table in O(n) time where n is the input string length. In each execution step, each processing element P_j is computing one cell pt(i,j) of the column j. At the next execution time P_j is used again to compute the cell that belongs to the same column but is one row higher pt(i-1, j). In addition one processing element is required to control the whole process and one more to handle the attribute evaluation process as shown in Fig. 3. The n elements

that are used for the parallel parsing are following the design presented in [10] (see Fig. 2).

After the end of each execution step k (t_{ek}), the computation of one parsing processing element terminates. At the next execution step this processing element should transmit the cells that it has computed, to the next processing (t_{ek}). Each processing element repeatedly calculates a cell, checks if it should transmit some cells and then if it should receive any.

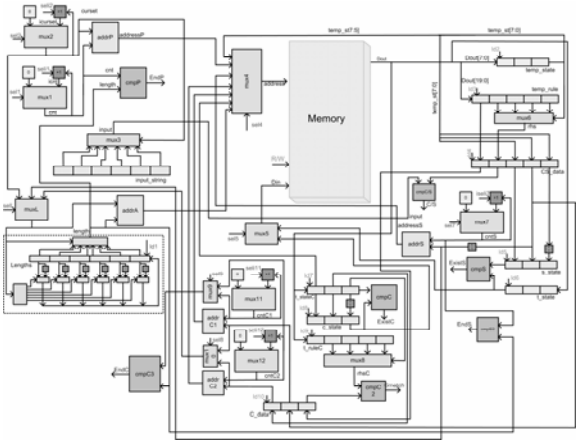

Fig. 2. The architecture of the Processing Element (PE)

2.3 The proposed modifications for the semantic processing

As it must be clear by now, the proposed implementation follows the architecture shown in Fig.3. The proposed architecture is based on the abovementioned CFG parser. The parser handles the recognition task and constructs the parse tree or parse trees in the case of ambiguous sentences. When the parsing process is over, the attributes may be evaluated. For that purpose, an extra module (Semantic Evaluator) has been created, so as to compute the semantics. This module takes as input the parse tree encoded in bit-vectors and gradually traverses it. In each branch (syntactic rule) of the tree, the semantic evaluator executes the corresponding semantic rule, which is nothing more than a concatenation of alpharithmetic strings. The resulting attribute value of the root symbol is the output string that will be transmitted to the abstract data-management machine that has access to a data-base in order to produce the final result (answer). Both parser and Semantic Evaluator are downloaded into the same FPGA board.

The parser module and semantic evaluator module are initialized by the grammar specifications. The resulting source code is downloaded into the FPGA. The latter, takes as input the input string, recognizes it, evaluates the semantics and responds with an intermediate code. In the example given in the next session the intermediate

code consisting of commands and their parameters, for the abstract data-management machine. Finally, the abstract data-management machine executes the received commands and provides the user with the final result.

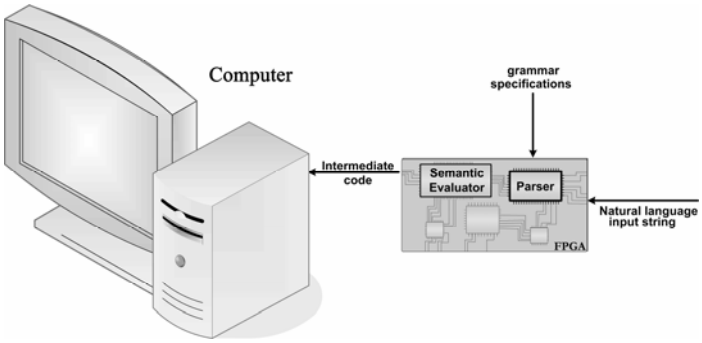

Fig. 3. The proposed architecture

3 An Illustrative Example

In order to show how we can build a natural language interface, using the system proposed, we have chosen a question-answering example [4] from the area of airline flights. In Table 1 an AG is given. The underlying grammar accepts questions concerning airline flights and the semantic rules produce an intermediate code, consisting of commands and their parameters, for an abstract data-management machine that has access to a data-base (Fig. 3), according to the example grammar of Table 1.

The subset of English accepted by the system uses words belonging to classes like: class names, object names, property names e.t.c.

The sentences of the subset of English are questions concerning airline flights and the answer after the processing of the intermediate code by the abstract data-management machine is YES or NO.

In this grammar, the semantics are described using a synthesized attribute called "output" for each non-terminal symbol of the underlying grammar. It contains the generated output string so far. The only operation needed between the attributes of the non-terminals is conc (par1, … parn), which stands for the concatenation of the contains of par1, … parn.

An illustrative simple question is:

A FLIGHT DEPARTS FROM ATHENS?

This question can be syntactically analyzed into a noun phrase consisting of a determiner and a common noun and a verb phrase consisting of a verb, a preposition and a proper noun. The determiner corresponds to a quantifier, the common noun to a class name, the verb and the preposition to a relation and the proper noun to an object. The nouns and the verb will be used as parameters by the commands that will

be generated. In order to show the exact correspondence between the above question and its semantic interpretation (described by the grammar), they are written one underneath the other as follows:

A FLIGHT DEPARTS FROM ATHENS?
(01)INT, (x) UNIV, (01)STO, (y) (z) CON.

The capital letters strings INT, UNIV, STO and CON are the name parts of commands which use as parameters the symbols enclosed in parentheses. Commas are used in the above illustration to separate a command and its parameters from the others. At this point it should be noted that the program generated must always be executed from right to left. For this reason the parameters combined with each call usually lie at the left of the call and this will be the form used in the explanation that follows.

The first command to be executed in the above example would be CON (z,y). The function of the command CON is to retrieve from the data-base the set of all objects which are related by the binary relation y, which here stands for "DEPARTS", to the object (z), which here stands for "ATHENS", and store it in the buffer. The second command to be executed is STO(01). The function of this command is to store the contents of the buffer into a location of the working data structure. This location is specified by the parameter of this call in this case (01). This call is generated when the main verb phrase structure is recognized. The next call is UNIV(x). The function of this command is to store in the buffer all the objects belonging to the class denoted by the parameter x which here stands for "FLIGHT". The next command is INT(01) which forms the intersection of the sets stores it in the buffer and tests whether it is empty or not. The result of this test determines the correct answer to the question and it is held in a flag-register.

A second illustrative question and its procedural semantic interpretation are:

FLT-1 FLIES FROM ATHENS TO NEW-YORK?
(01)(x)MEM, (01)STO, (y) (z) (w) COB.

The new commands generated by the above question are COB and MEM. The function of COB is to retrieve from the data base the set of all objects which are related by the ternary relation y, which stands for "FLIES", to the pair of objects z and w, which stand for "ATHENS" and "NEW YORK" respectively. The function of MEM(x,01) is to test whether the object x is a member of the set stored in location 01. The result of this test again determines the correct answer to the question.

More complex questions, like "EACH FLIGHT WHICH IS CONNECTED TO A FLIGHT WHICH BELONGS TO AIRLINE-1 DEPARTS FROM A CITY WHICH IS LINKED TO EACH CITY WHICH BELONGS TO GREECE", can be handled by the proposed grammar. In this question, nested quantified subordinate clauses are used and new commands will be generated according to the grammar.

It can be seen from the above illustration that semantic interpretation can be achieved by establishing a mapping between syntactic structures and semantic components. This mapping can be described formally with the attribute grammar of Table 1. In this grammar the semantics are described using a synthesized attribute called 'output' for each non-terminal symbol of the underlying grammar. The only operation needed between the attributes of the non-terminals is conc (par_1, …, par_n), which stands for the concatenation of the contains of par_1, …, par_n . In Table 1 the 'output ' attribute is represented by 'o' for simplicity.

Table 1. The AG of the example

Non Terminals	PR, NP, VP, QS, W, OB, A, E, SET, REL, AT, NRL, N, CNJ	
Terminals	Object names, Class Names, Relation Phrases, Property Names, Numerical Relations, Numbers, Relative pronouns, Determiners, Conjunction words or symbols	
Syntax Rules		**Semantic Rules**
PR → NP VP		PR.o=conc(NP.o, "01ST0", VP.o)
NP → QS W VP		NP.o=conc(QS.o, "SEL", W.o, VP.o)
NP → QS		NP.o = conc(QS.o, "UNIV")
NP → OB		NP.o = conc("QS" ,OB.o, "01MEM")
QS → A SET		QS.o=(A.o, "01INT", SET.o)
QS → E SET		QS.o= conc(E.o, "01IMP", SET.o)
VP → VP_1 VP_2 SP		VP.o= conc(VP_1.o, VP_2.o, SP.o)
VP_1 → IP VP_1		VP_1.o = conc(IP.o,VP_1.o)
VP_1 → null		
VP_2 → SP CNJ VP_2		VP_2.o = conc("02INT", SP.o, CNJ.o,"02ST0", VP_2.o)
VP_2 → null		
IP → REL QF W		IP.o = conc(REL.o, QF.o, "SEL", W.o)
SP → REL SB		SP.o = conc(REL.o, SB.o)
SP → AT NRL N		SP.o = conc(AT.o, NRL.o, N.o, "LIM")
SB → QF		SB.o = conc(QF.o,"UNIV")
SB → OB		SB.o = conc(OB.o, "CON")
SB → OB P OB		SB.o = conc(OB.o, P.o, OB.o, "COB")
QF → A SET		QF.o=(A.o, "RAN", SET.o)
QF → E SET		QF.o= conc(E.o, "REA", SET.o)
OB → Object names	*e.g. : Athens*	*OB.o="Athens"*
SET → Class Names	*e.g.: Flights*	SET.*o ="Flights"*
REL → Relation Phrases	*e.g.: Departs from*	*REL.o ="Departs from"*
AT → Property Names	*e.g.: Has population*	*AT.o ="Has population"*
NRL → Numerical Relations	*e.g.: Larger than*	*NRL.o ="Larger than"*
N → Numbers		
W → Relative pronouns	*e.g.: which*	*W.o ="which"*
A → Determiners	*e.g.: A*	*A.o ="A"*
E → Determiners	*e.g.: Each*	*E.o ="Each"*
CNJ → Conjunction words or symbols	*e.g.: And*	*CNJ.o ="And"*

5 Conclusion and Future Work

This work is a part of a project for developing a platform (based on AGs) in order to automatically generate special purpose embedded systems. In this paper, an efficient architecture for natural language processing is presented, implemented in hardware using FPGA. The system can receive sentences belonging to a subset of NL from the internet or as SMS, as it is shown in Fig. 4. The system has been tested with numerous application examples [10] and the speed-up was an order of magnitude on the average. The main contribution of this paper is the proposed model, for implementing in hardware the complete grammar (syntax rules plus semantic rules), for NL applications. Our future work remains focused in implementing the proposed architecture using a faster parser, e.g. the one proposed in ref. [11]. In applications where more complicated semantics are required instead of a simple module, as in the illustrative example, a processor should also be incorporated in the FPGA e.g. MicroBlaze [6] soft-core microprocessor, as proposed in ref. [12].

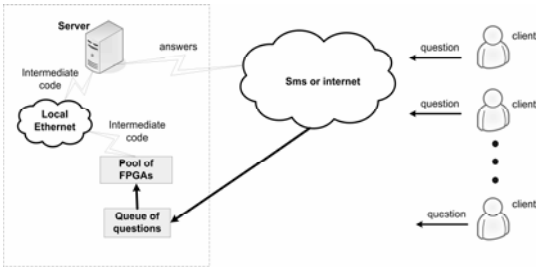

Fig. 4. A real-life application example

Acknowledgements: This work has been funded by the project PENED 2003.This project is part of the OPERATIONAL PROGRAMME ”COMPETITIVENESS” and is co-funded by the European Social Fund (75%) and National Resources (25%).

References

1. J. Earley, “An efficient context–free parsing algorithm”, Com.of ACM, 13, pp. 94-102, 1970.
2. Y. Li, H. Yang and H. V. Jagadish, “NaLIX: an interactive natural language interface for querying XML”, Proceedings of the 2005 ACM SIGMOD international conference on Management of data, pp. 900-902, Baltimore, Maryland
3. A. Yates, O. Etzioni and D. Weld , “A reliable natural language interface to household appliance”, Proceedings of the 8th international conference on Intelligent user interfaces, pp. 189 - 196, 2003
4. J. Kontos and G. Papakonstantinou, “Semantic interpretation of English like questions using procedural components”, NATO ASI on on-line mechanized information retrieval systems, Lyngby, Denmark, 1972.
5. Y. Chiang, K. Fu “Parallel parsing algorithms and VLSI implementation for syntactic pattern recognition”, IEEE Trans. Pattern Anal. and Mach. Intell. PAMI-7, 1984
6. Xilinx Official WebSite,www.xilinx.com
7. A. Aho, R. Sethi and J. Ullman, “Compilers – Principles, Techniques and Tools”, Reading, MA, MADDISON-WESLEY, pp. 293-296. 1986
8. J. Paaki, “Attribute grammar paradigms -a high-level methodology in language implementation” ACM Computing Surveys, 27(2):196–255, 1995
9. C. Pavlatos, A. Dimopoulos and G. Papakonstantinou, “An Intelligent Embedded System for Control Applications”, Workshop on Modeling and Control of Complex Systems, Cyprus, 2005
10. C. Pavlatos, I. Panagopoulos and G. Papakonstantinou, “A programmable Pipelined Coprocessor for Parsing Applications”, Workshop on Application Specific Processors CODES, Stockholm, 2004
11. A. Koulouris, T. Andronikos, C. Pavlatos, A. Dimopoulos, I.Panagopoulos, and G. Papakonstantinou, “Efficient Signal Processing using Syntactic Pattern

RecognitionMethods", International Conference on SIGNAL AND IMAGE PROCESSING ,Honolulu, Hawaii, USA, August 14–16, 2006

12. C. Pavlatos, A. Dimopoulos and G. Papakonstantinou, "An embedded system for the electrocardiogram recognition", EMBEC'05, Prague, Czech Republic, November 2005