Using FPGAs for Building Reconfigurable Symbolic Sensors

Radu Varbanescu[1], Ana Lucia Varbanescu[2]

POLITEHNICA University of Bucharest, Spl. Independentei 313, sector 6, Bucharest, Romania 060032
Phone/Fax: +40217693079
[1] raduv@isis.pub.ro, [2] analucia@cs.pub.ro

Abstract **-** Symbolic Sensors can answer a query by a qualitative answer, using linguistic symbols instead of the usual numerical representation. Because there are no sensors with a built-in symbolic behaviour, the solution is to develop add-on modules for enhancing common sensors with such a set of features. These add-ons can be built either in software or in hardware. This paper focuses on the hardware solution and it aims to solve the lack of flexibility issue by using reconfigurable devices such as FPGAs. These devices allow dynamic reconfiguration and they provide the required flexibility for the entire system, without having a significant decrease in performance. The experiments conducted during the research prove that such a solution is efficient and reliable, thus being a serious alternative for the older solutions developed in software.

I. Introduction

There is no need in discussing the importance of sensors in the common daily existence – they are almost everywhere: in your car, in your apartment or inside the microwave oven. Basically, as every machine is now embedding a control unit, it has to deal with sensors – several different sensors. Without any doubt, sensors are evolving and increasing in power and performance like never before. Some voices actually say it out loud: the impact of sensors in this decade (started in year 2000) will be as surprising as microprocessors were in the 1980s and lasers in the 1990s.

In this context, various kinds of enhanced sensors are developing – they are generically called “smart sensors”. In fact, any sensor having local memory, having its own local processing power and being able to communicate with other sensors and/or computers can be considered a smart sensor. “Smart” has to deal more with resources than with functionality. The way these resources are actually used and constrained determines the enhanced behaviour a sensor might get, thus defining its specific functionality.

Symbolic sensors are smart sensors with specific communication constraints: they have to answer a query with a qualitative reply, by using symbolic description instead of (or together with) some numeric representation of the requested data.

There are several questions to be asked: Is there any use for symbolic sensors? Is it possible to implement such sensors? Is there a way to transform legacy sensors into smart/symbolic ones? Are there any features that will represent a significant enhancement to such sensors? Can these features be easily implemented?

This paper offers answers to some of these questions. However, it has to be said that the entire research project is based on the strong conviction that symbolic sensors are useful and they can improve communication between machines and human operators. There are some successful experiments involving symbolic sensors, experiments that constitute enough evidence, in the opinion of the authors, for going on into searching solutions to make such sensors affordable, easy-to-implement and easy-to-use.

II. Symbolic Sensors

As stated before, symbolic sensors are devices that will answer a measurement query with symbolic values. For example, consider that you have a temperature sensor for taking the human body temperature, measuring 37.22°C: the symbolic answer can be "SMALL FEVER". At 40°C, the sensor should warn you with "VERY HIGH FEVER". Of course, both these values would mean "TOO COLD" if the temperature is measured in the oven, just before putting a cheese cake inside.

Symbolic sensors were born more than 20 years ago, as a proof of concept for a numerical-symbolic interface [1] developed using fuzzy logic. By that time, the approach was highly theoretical. The other very important aspects – usability and implementation – were completely neglected.

Despite their age, these sensors face the same problems as all the rest of the smart sensors: they are

very interesting for research, but not convincing for industry. Even if the smart sensors are an important trend, there are thousands and thousands of legacy systems already in place. A lot of legacy devices have some very desirable characteristics, like industrial-strength I/O, local functions and operator interface, digital data (frequently RS-232 or RS-485), they are installed and they work. So, unless there is an easy solution for adapting these devices, smart sensors will not be adapted on a large scale.

Symbolic sensors are useful in applications where the numerical precision of the reply is not the key issue – operators do not require a numerical precise representation, but a symbolic hint and/or indication. One applications of such sensors is presented in [4] (real-time object detection in robotics) and several others are involved in research projects by the time. They offer, in the authors' opinion, sufficient reasoning to study symbolic sensors implementation.

A. Implementation

A generic functional diagram of a symbolic smart sensor is presented in Figure 1.

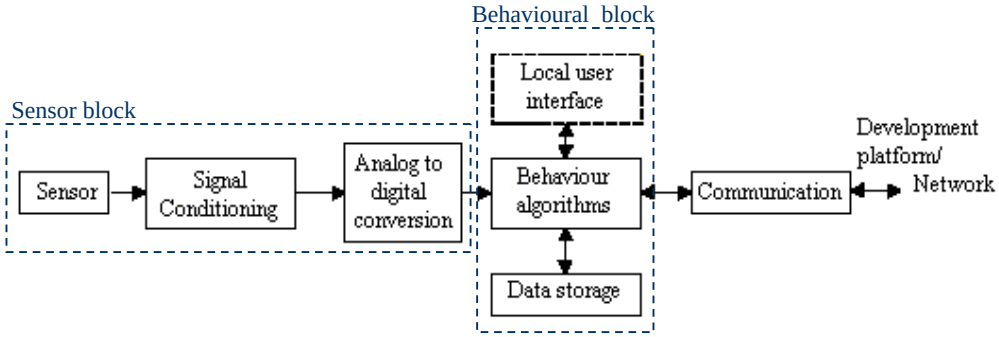

Figure 1. A generic functional diagram of a symbolic smart sensor

For the purpose of this presentation, the sensors to become symbolic have been spited into two very different categories, each of them having specific requirements and needs for obtaining symbolic behaviour.

1. Smart sensors

As they are sensors that have local resources, like processing power and memory, the symbolic behaviour can be locally added. The add-on in this case is a software application, introduced in [2]. Such an application will program the local processor and get built-in symbolic features. Getting back to Figure 1 for one second, the modifications should be made only at the “Behaviour algorithms” block, as they have to be implemented according to the application requirements.

2. Legacy sensors

This is the case of sensors that only reply to measurement queries by producing a measurable corresponding electrical signal. Comparing to the diagram in Figure 1, a legacy sensor will only provide the “Sensor block” functionality, having no other local resources. Adding symbolic behaviour to these older devices is a two-step process: (1) the addition of hardware modules that will simulate the smart sensor and (2) the development of the software application for implementing a specific symbolic behaviour. Such sensors will be called *slave symbolic sensors*.

No matter what types of sensors are used, the behaviour algorithms have to be implemented in order for the symbolic reply to be generated. These algorithms based on the numeric-symbolic interface presented in [1], but the implementation techniques may vary significantly, depending on the processors and additional resources involved.

B. Configurability of the symbolic behaviour

The temperature sensor example provided at the beginning of this section proves an important issue: the symbolic behaviour has to be reconfigurable, depending on the sensor family, on the measured entity and on the measurement scenario (application).

1. Entity-related configurability – Family reconfigurability

   The user has to specify what the measured entity is – temperature, pressure level, smoke level and so on. Depending on the entity, the symbolic behaviour algorithm has to choose the specific family of symbols for expressing the measured value.

2. Application-related configurability

Actually, this is the algorithmic modelling of the linguistic concepts [3]. In theory, each numeric value can be associated to a linguistic symbol. In fact, as expected, several adjacent numeric values will be represented by the same linguistic symbol. This solution partitions the measurement range into sub-ranges, each of those having an associated *meaning*.

From point of view of the implemented system, the *meaning* is represented as a function, defined on the measurement range, having values in the specified symbols set. For getting the correct symbol associated to a certain value, the function has to be applied on the value received from the sensor. Thus, the set of meaning functions used to represent the associations between values and symbols is the only specification required for configuring an application.

The theory behind such functions is not the subject of this paper, so they will be used without further demonstrations or argumentations. However, in this stage, the application is limited to constant and linear functions.

3. Sensor-related configurability – Range reconfigurability
   The symbols scale has to be adapted correctly for the specific sensor and for the specific application. (the temperature sensor for the human body will be different from the one installed in the oven)

C. State-of-the-art in the symbolic sensors implementation

There are no sensors having such a complex behaviour embedded by the producer and this is mainly because of the reconfigurability needs. So far, there are solutions to implement symbolic sensors in software, directly using the development workstation as a data centralization point [2]; although they are providing high flexibility, they do require the presence of PCs in the vicinity of the sensors.

The focus is moved now to another possible solution: a hardware add-on. Sensors that gain symbolic behaviour by the means of this hardware add-on would be faster and easier to plug in other systems. There are several research groups working on proving a viable solution for a hardware add-on, based on microcontrollers. In this case, the biggest disadvantage is the increased reconfiguration cost, both in time and money.

This paper will demonstrate a third option: reconfigurable symbolic sensors. They are using FPGAs to obtain the desired behaviour. As the name clearly states, although there is a hardware module, reconfigurability is provided “in-circuit”, so it adds the required flexibility.

III. Reconfigurable Hardware

Reconfigurable devices are integrated circuits that allow their internal configuration to be changed, according to application requirements, by modifying internal cells behaviour and their interconnections. Although the idea is quite old, the further evolution of these devices was rather slow. And this was mainly because of the lack of hardware resources and the lack of professional tools for using them properly. Nowadays, the reconfigurable devices to be used are mainly FPGAs or CPLDs [6].

A. Using a reconfigurable device

The steps required for programming and using an FPGA are briefly presented:

1. *Specify the circuit* – via schematic capture or using HDLs (Hardware Description Languages).
2. *Synthesize* the essence of the design in the form of an ASCII coded circuit description called a *netlist*.
3. *Mapping – maps* (partitions) the set of logic gates found in the netlist into the set of logic blocks available on the target FPGA device.
4. *Placement and routing* – chooses one of the physical logic blocks to implement an allocated piece of the circuit and connects the inputs and outputs of each block to the others
5. *Configuring the FPGA* – the programming data (*bitstream*) is downloaded to the real circuit,

However, in the case of this prototype, all these steps are to be executed only once, by the producer of the symbolic sensor. The deployment of the entire application will only contain the bitstreams for the specified sensors and applications.

B. Choosing the device for the reconfigurable symbolic sensor

The symbolic behaviour hardware add-on will be implemented in a FPGA (Field Programmable Gate Array) from Xilinx – Spartan II. The features that lead the authors to this specific decision were

both objective (the fine-grain cells, the flexible interconnection structure, the amount of memory available in each cell and the ease of reconfiguration) and subjective (a development board is available in the laboratory).

The design was developed using Verilog. It has to be said that this particular choice was purely subjective. Also, for actual implementation of the entire system, we have used the Xilinx tools.

IV. A Prototype of the Symbolic Behaviour Engine

A. Architecture

The architecture consists of the following interconnected blocks: the sensor, the FPGA board, a display and a download/debug parallel connection to the development workstation. In the case of the targeted system, the sensor will be considered a black box that outputs some digital values. The display is used for the human operator to supervise the system reactions – now it is mainly used for debugging. It is also the case of the connection to the main processing unit, which is beyond the purpose of this paper. A block diagram of the entire system is presented in Figure 2.

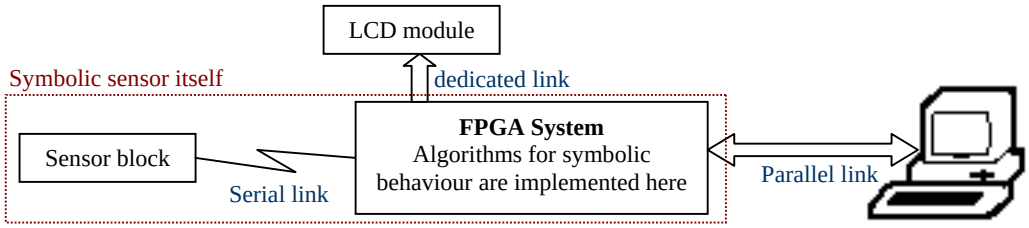

Figure 2. Block diagram of the reconfigurable symbolic sensor

The FPGA board used for the experiments is XESS XSA-50, using the Xilinx Spartan-II FPGA [6]. The LCD unit is a text display, with 2 lines x 20 characters each. It is used for printing the results of the measurement and for debugging. The serial link is RS232 compatible. The parallel link is used for downloading the configuration bitstream into the FPGA, and it is a standard parallel cable. The link to the LCD module is 14 bits wide, but only 8 of them are actually used [8].

A picture of the prototype system – in an early development phase – is presented in Figure 3.

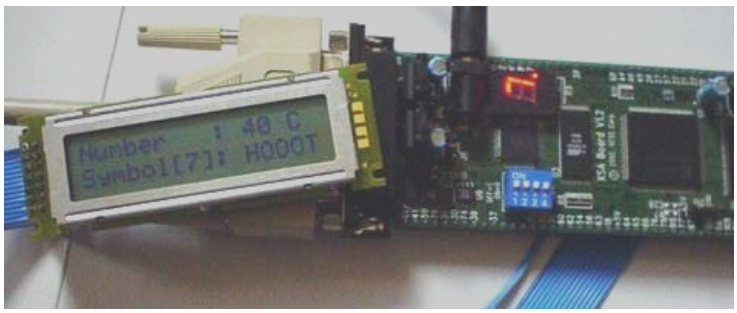

Figure 3. A picture of the prototype system

B. Components in implementation

The system programmed in the FPGA is a query-response system, and it is based on three repositories:

1. Application scenarios – "Apps"
   This repository contains all the applications for a certain family of sensors. In other words, all the scenarios that are suitable for a sensor family will be collected into such a repository, for fast reconfiguration purposes.
2. Functions used for value-symbol evaluation – "Functions"
   This repository stores all the functions used by the system. The function expressions are entered by the user during configuration, and they are directly used for generating the correspondence between the value and the symbol. Practically, the transducer outputs the value and this value becomes a parameter for the evaluation function. Depending on the result of the function, the correct linguistic symbol will be searched for and replied by the symbolic sensor.

3. Linguistic symbols – "Symbols"

This is a collection of words, organized around "*generic concepts*". Depending on the measured entity of each sensors, the user/the application configures a generic concept, which is the concept corresponding to the values in the middle of the measurement range. Although there are several solutions for automatic generation of the secondary concepts, they are not of interest in this case. The system has enough memory too keep track of a huge dictionary, so it is more efficient in terms of access speed to get the value from the repository than to generate it each time it is required.

Besides these three repositories, the entire system contains a few additional modules, required for the integrity of the system:

- Main unit – the control unit for the entire application; it plays the role of the CPU in this system;
- UART module – a simple UART, able to manage the serial communication between the sensor and the main unit;
- LCD Communication – data to be outputted from the main unit to the "user"; in the case of this prototype, the information is send to the LCD module, so that the user sees the sensor answer

A functional diagram of the implementation of the system can be found in Figure 4. The sources for each module are written in Verilog, and they are freeware, under the open source licence agreement. They are available and they can also be requested by e-mail.

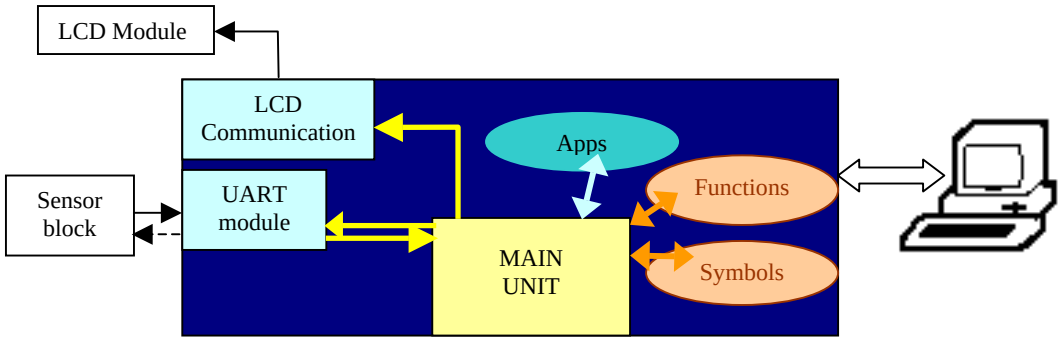

Figure 4. The functional blocks implemented in the FPGA

C. Main Unit Algorithm

The main unit has the following information available:

- It knows the used sensor and all the information required about it – range, entity etc.
- It knows the application for which the sensor is used – in this case, it also knows the generic concept ID for this application and the functions' IDs to be used.

Knowing all these, the M.U. executes the same routine for each query for the sensor:

1. Queries the UART module for the last received value – this value is retrieved from an internal buffer; it will always output the last received value. UART sends the value to the M.U.
2. Searches the Functions Repository for the function with the specified ID; this repository is organized as a content-addressable memory, so the search is quite fast, and it is done in hardware. The function is returned as a lookup memory (called FunctionResult) filled in with the corresponding symbol IDs, according to the range configured by the application specification
3. Indexes the FunctionResult with the value received from the sensor, recovering the symbol ID
4. Searches the Symbols Repository for the symbol (the real word) and it sends it to the "LCD Communication" module.

All the errors that might appear during these procedures are signalled to the user – either by displaying an error message on the LCD or by using the onboard 7-segment LED.

D. How is it configured?

On the development workstation, there is a SensorsFamily library. This is a library containing all specifications of the sensor family to be used – sensor manufacturer, model number, serial number, measurement range, calibration info, user info and so on, for each member of the family. All needed information is easily imported from TEDS (Transducer Electronic DataSheet), in conformity with the IEEE 1451.4 standard – the one dedicated to the so-called "plug&play sensors" [8].

The FPGA is configured using the configuration files from the Sensor Family requested by the user. If the FPGA has enough memory for loading more applications for a certain family, the system will be

faster, because application switching will not require reconfiguration from the development workstation. In the case of the presented prototype, only one application can be stored for a specific configuration.

Finally, the step-by-step procedure for configuring the entire system is presented below:

1. The user requests the system to be reconfigured for a specific sensors family – one or several applications for this family will be downloaded
2. The specific application scenario is configured from the FPGA board (using the switches)
3. The M.U. will get the information by querying the selected application in the Apps Repository
4. The M.U. will start executing the algorithm and it will output the results
5. If a new application is required, the M.U. stops and loads a different version of an application from the Apps – then goes back to step 3; if the new application is not available, it goes back to step 2.
6. If a new family of sensors is to be plugged in, the system has to be stopped/paused and it has to be reconfigured. It goes back to step 1.

V. Conclusions: Performance Issues and Further Development

First of all, this system proves that symbolic sensors can be obtained from legacy devices by embedding reconfigurable hardware add-ons. In terms of prototyping and teaching, reconfigurable devices are the best. This first prototype is a proof-of-concept. The main performance issue here is functionality!

However, some performance criteria of this first prototype will target only three issues:

- the response time of the system – useful only as a benchmark for further development
- the percentage of FPGA resources used per family – lack of resources determines significant performance decrease, because of more reconfiguration operations required
- ease of reconfiguration – reconfiguration using already built bitstreams is very fast; if the bitstreams are not available, the entire process of building bitstreams could be complicated.

Further developments include a second build of the entire application. We aim to use a bigger FPGA system, with several improvements – dynamic reconfiguration, multiple contexts, and increased memory space. Thus, further statistics can be added and discussed only after this second build will exist.

Next, we will focus on comparisons to similar MCU-based symbolic sensors. The software approach is practically incompatible with such a comparison, because it belongs to a different category of applications. After this, statistics will become relevant for discussing options for implementing symbolic sensors.

From the usability point of view, on the long term, we intend to group several sensors to a multi-interface based on such architecture. In this case, it is quite interesting if several transducer families can use the same reconfigurable add-on for generating symbolic answers.

Although the authors think that the architecture of the entire system is pretty good, there are several performance issues still to be proven and, even more, there is the issue of generality – we cannot claim, yet, that the system is general enough for being considered as a good solution for building stable and efficient reconfigurable symbolic sensors.

References

[1]. E. Benoit, L. Foulloy, “Symbolic Sensors: One Solution to the Numerical-Symbolic Interface”, *Proc. IMACS DSS*, Toulouse, 1991

[2]. R. Varbanescu, A.L. Varbanescu - “A Software Solution for Developing Software Symbolic Transducers” – *12th IMEKO TC4 International Symposium*, Zagreb, Croatia

[3]. G. Mauris, E. Benoit, L. Foulloy, “Fuzzy Symbolic Sensors - From Concept to Applications”, *Measurement* no. 12/1994, pp. 357-384

[4]. M. Pauly, H. Surmann, M. Finke, N. Liang – “Real-Time Object Detection for Autonomous Robots”

[5]. R.Swan, A.Wyatt, R.Cant, C.Langensiepen – “Re-configurable Computing” - *ACM Crossroads Student Magazine*, January 2001, [http://www.acm.org/crossroads/xrds5-3/ntu.html]

[6]. * * * - “XESS XSA-50 Presentation” - http://www.xess.com/prod027.php3

[7]. C. Peacock, P. Luethi – “The Extended Concise LCD Data Sheet” – *http://www.beyondlogic.org*

[8]. *** – “Sensors Plug&Play, The New Standard for Automated Sensor Measurements” – *National Instruments*