7th CSNDSP - 21-23 July 2010, University of Northumbria at Newcastle, U.K.

"Driving the future of telecommunications"

CSNDSP 20 10

Proceeding of the

Seventh IEEE, IET International Symposium on

Communication Systems, Networks and Digital Signal Processing

northumbria UNIVERSITY

Edited by: Z. Ghassemlooy, W. P. Ng

List of Papers

Nets4Cars: Communication Technologies for Vehicles

Designing a Reconfigurable Architecture for Ultra-Low Power Wireless Sensors

Johann Glaser, Jan Haase, Christoph Grimm
Vienna University of Technology
Institute of Computer Technology
EMail: {glaser,haase,grimm}@ict.tuwien.ac.at

***Abstract*— In wireless sensor network nodes all tasks are controlled and scheduled by the CPU of the node. It is activated repeatedly from its low-power sleep mode for e.g., measurements and communications tasks. The periodic wake-ups cause a high overhead in power consumption. This problem can be solved by supplementing the CPU with additional modules which autonomously execute selected tasks to enable the CPU to stay in its inactive low-power mode. It is only activated if more complex processing is required, e.g. for generating a network packet to transmit a changed sensor value. We introduce logic modules with a reconfigurable architecture which allow to flexibly adopt the functionality but still perform relatively complex tasks. This work presents the methodology for the design of these reconfigurable logic modules.**[1]

I. Introduction

Wireless sensor network (WSN) nodes comprise a central processing unit (CPU), memory (both volatile and non-volatile), a timing unit like a real time clock (RTC), sensors for environmental measurements (e.g., temperature), communication modules like RF transceivers and a power supply [1]. The applications of WSNs are manifold, including building automation, automotive systems, container tracking and environmental monitoring.

Every application poses different requirements on the WSN and its nodes. Hence, a large amount of different architectures, network protocols, sensor and power supply techniques are available. The optimization goal for a specific WSN node is determined by its field of application. So, there is no WSN node architecture which is suitable for all applications, let alone the optimum for all applications [1].

Besides this there is one requirement which is common to most applications: low energy consumption of the WSN node due to limited power scavenging potential and a limited battery capacity. The main approach for reducing the power consumption is to reduce the activity of the WSN node by periodically entering an inactive low-power mode. On the other hand, every task of the node has to be performed by the CPU, which controls the whole node.

We propose to insert specific reconfigurable peripheral control modules into the WSN node as depicted in Fig. 1 (shaded boxes). These independently perform certain subtasks and only activate the CPU if further, more complex processing is required. With this approach the CPU can remain in an inactive low-power mode for extended periods and thus reduce the total power consumption [2]. While the power consumption of a peripheral module is independent whether it is controlled by the fully-featured CPU or by a dedicated peripheral control module, the peripheral control module must have considerably less power consumption than the CPU. This is ensured due to its specialized design and therefore small count of logic gates.

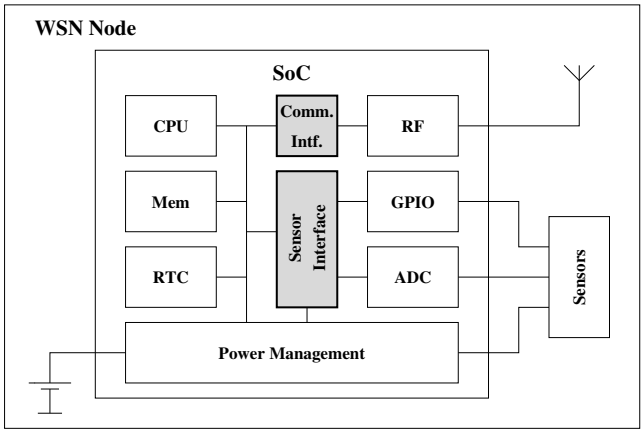

Fig. 1. WSN node with reconfigurable hardware blocks (CPU: central processing unit, Mem: memory, RTC: real rime clock, Comm.Intf.: communication interface, RF: radio frequency transceiver, GPIO: general purpose IO, ADC: analog-to-digital converter).

For example, for the periodic measurements of the sensor values the CPU is activated from its inactive low-power mode. Then the power supply of an external sensor is turned on. After some settling time the sensor value is ascertained by an analog/digital conversion (ADC). The resulting value is finally compared to the previous value and only if the difference is greater than a certain threshold, further processing is conducted, like sending the value via the wireless network. Otherwise the CPU immediately switches back to an inactive low-power mode.

Another example is the MAC (media access control) layer of the network stack. For a network protocol with periodic listening intervals the receiver is turned on for a certain time waiting to receive a packet. If a packet is received, its destination address is inspected to decide upon further processing. After these relatively simple tasks the more complex tasks of the higher protocol layers follow.

[1]This work was funded by the Austrian government via FIT-IT (grant number 815069/13511) within the European ITEA2 project GEODES (grant number 07013).

These cycles consist of several simple tasks (set and read digital IOs, compare two integers) where delays occur (sensor settling time, network listening). These are usually performed by the CPU, which is overqualified and oversized for these tasks. It consumes power during the whole active state as well as for the wakeup procedure. The proposed reconfigurable peripheral control modules are intended to avoid this waste of energy. Here we will discuss the methodology for the design of these modules.

Shifting the border in a software/hardware partitioning process towards hardware reduces the flexibility of the final application. While software can be modified by reprogramming the code memory, synthesized logic cores require a redesign of the chip. Still, there are three important reasons for flexibility:

1) Covering multiple different applications for a higher market potential.
2) Adopting to different external components across PCB design cycles.
3) Fixing bugs without a chip redesign.

Therefore we propose to make the introduced dedicated hardware blocks *reconfigurable*.

The next section describes alternative approaches as well as the basis of our methodology. This is followed by a description of the design flow for ultra-low power reconfigurable logic modules and a section on the design tools. In the following section an example of this design flow is given. This work is finished with conclusions and an outlook to future work.

II. Related Work

The above mentioned sensor interface task was investigated in [3] for a conventional implementation where the CPU programmed with firmware performs all control tasks as well as for two different hardware implementations, one using the proposed reconfigurable architecture and the other using a hard-coded finite state machine (FSM). The latter reduced the power consumption by a factor of more than 90 while the reconfigurable hardware implementation still achieved a reduction by a factor of nearly 34, both compared to the firmware implementation. This number is still largely underestimated, because the authors used an FPGA as underlying architecture instead of a custom chip. From this result we conclude, that our goal to reduce the power consumption of a WSN node is achievable by relieving the CPU from such simple tasks.

An embedded FPGA was presented in [4] for the control of IO ports and protocols. The RISC CPU core of the high-performance SoC can be extended by the "Pipelined Configurable Gate Array (PiCoGA)" where application-specific functions can be mapped ([4, p. 87]). While this approach is very flexible, it is clearly designed for high-performance embedded systems and does not concentrate on low-power design.

The high overhead of an FPGA was tackled by the soft embedded programmable logic cores presented in [5]. They used a non-regular architecture, but synthesis tools get into difficulties then. In later work the overhead of standard cells was reduced by the introduction of

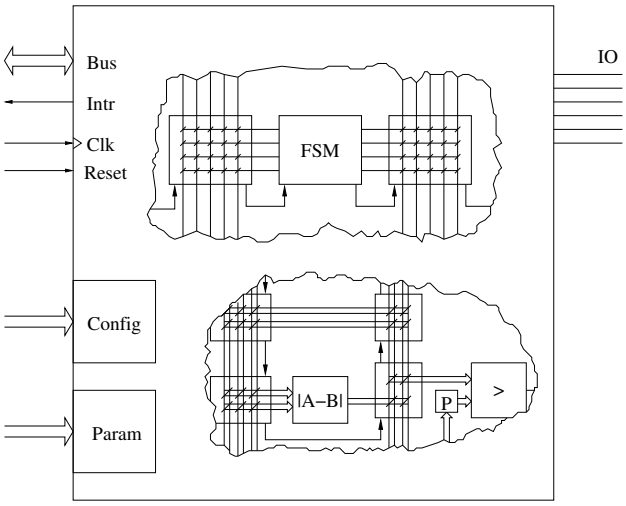

Fig. 2. Internals of a reconfigurable module. Intr: Interrupt, FSM: Finite State Machine, P: Parameterization Cell, Config: Configuration Interface, Param: Parameterization Interface.

architecture specific tactical cells [6]. This work is a good starting point for a logic core inserted between a sensor interface and the CPU, but still lacks the optimization potential spanned by the anticipated knowledge of the application.

An FPGA holds a large array of universal logic elements with versatile routing resources to combine the elements. This ensures that the underlying reconfigurable FPGA architecture is appropriate for virtually any application. In [2] we presented a different approach on a reconfigurable architecture for peripheral control modules in WSN nodes. Based on the finding that the configuration bit stream of FPGAs consists of only 10 % for the logic configuration and 90 % for the routing configuration, we proposed to reduce this high overhead by designing the underlying hardware architecture tailored to the specific *application class* of each reconfigurable module, (e.g. communication interface and sensor interface in Fig. 1). This defines the range of applications of the reconfigurable module. Referring to the previous sensor interface example, the application class defines how these sensors can be controlled as well as the sensor value post processing algorithms. For the final application the actual sensor and actual post processing are specified. From this the configuration of the reconfigurable module is derived and applied during system startup.

In our approach we limit the range of applications of a single reconfigurable module and therefore are able to place specialized cells and reduce the routing resources. The specialized cells include arithmetic operations like adders or the absolute difference of two integer values (e.g. $|A-B|$ in Fig. 2). [6] report a reduction of area and delay overhead by 58 % and 40 %, respectively with the introduction of these so called "tactical cells".

Reconfigurable logic modules typically also contain control logic like a finite state machine (FSM). The logic which calculates the output signals as well as the next state is usually built from many combinational gates. [7] introduced an architecture which directly treats the

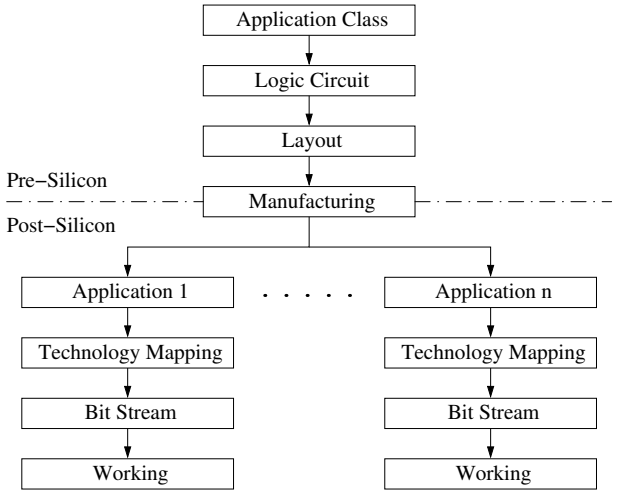

Fig. 3. Design flow diagram for reconfigurable logic modules

state transitions instead of the transition function. With this approach the total count of configuration bits was reduced to only between 7.3 and 12.7 % compared to an implementation with a commercial FPGA for certain sets of predefined automata.

III. Design

The design of a reconfigurable logic module is a two-part process. In the *pre-silicon phase*, the application class is defined and a reconfigurable circuit is developed. This results in chip layout data and finally in a produced chip. In the *post-silicon phase*, the reconfigurable logic block is configured, analogous to an FPGA, to implement the actual application (see Fig. 3).

A. Pre-silicon Phase

In the pre-silicon design phase the desired class of application has to be specified. Several different but similar applications are considered (also referred to as a *set of target applications* by [8]). The requirements for logic blocks and connectivity are specified and used to develop the deployed gates and routing resources which finally result in the chip layout.

Note that for other reconfigurable architectures like FPGAs the pre-silicon phase is usually performed by the chip vendor. The internal structures, gates, connections and switches are designed to meet the requirements of as many applications as possible. The designer will then utilize these structures to implement his application.

In the above sensor interface example, the application class is defined by the desired range of covered sensors and post-processing algorithms. Some external sensors only require a power supply for their activation while others need an additional selection signal. Some sensors provide an output signal to show that the value is stable and others just specify a time interval. Some sensors have an analog output and others have a digital interface (e.g., I^2C, SPI). The post-processing includes a comparison of the current value with the previous one. Only if the difference exceeds a certain threshold, the CPU is notified to take over control and perform more complex tasks.

All combinations of these cases have to be covered by the reconfigurable logic block. It needs interfaces to peripherals (ADC, I^2C and SPI) and the CPU (interrupt), external signals (power supply of the sensor, activation signal, ready signal), a timer (settling time), a comparator for the sensor value, and registers to store the old value and the (configurable) threshold. This threshold and the timing values are setup via special parameterization interfaces to allow adoption of numeric value during operation without applying a different configuration. This is complemented by a flexible state machine to control the order of processing (compare Fig. 3).

From this specification a logic circuit is designed which includes the required blocks. All these parts are connected via a routing infrastructure to provide the connections necessary to implement the desired functionality. This is enriched by the configuration and parameterization infrastructure logic. With synthesis, technology mapping and place and route tools the final layout of the reconfigurable logic block is developed. The pre-silicon phase is finalized by the production of the SoC.

After production, the hardware circuits can not be modified (with reasonable costs) so the user has to get by with the available structures to implement the desired application. This means that the pre-silicon design sets restrictions to the post-silicon design space by the available set of blocks, routing resources and configuration and parameterization options.

B. Post-silicon Phase

The post-silicon phase is analogous to the design flow for an FPGA. The actual application is specified and designed. We plan to utilize commercially available tools to synthesize the design to a netlist. For special tactical cells like an FSM a stand-alone description is fed to custom tools which derive the configuration of these cells.

The synthesized netlist is then mapped to the underlying logic module inventory as designed in the pre-silicon phase. Due to the complex non-regular structure we expect difficulties for the tools as already mentioned by [5]. These also have to check timing constraints. The output of this step is a technology specific netlist. The cell instances are then mapped to the locations of the available hardware cells and the connections between these cells are assigned.

From the placement and routing information and the specified functionality of configurable cells (e.g. the FSM) the content of the configuration store is constructed. This is mapped to the configuration chain and finally to the bit stream.

At system start up of the SoC the integrated CPU runs its boot sequence. This includes the setup of peripheral units like timers and ADCs by programming the according registers. Additionally the configuration bit stream is downloaded to the reconfigurable logic modules. We propose to implement the configuration interface as memory mapped registers. After configuring the reconfigurable logic module with the loaded bit stream (termed “configware” by [9], analogous to “software”) the

parameter registers are set to the required values. Finally the reconfigurable logic module is activated and starts to perform its task as specified by the application definition.

IV. Tools

This section gives an overview of the tool chain utilized for the above design flow. This is a topic of ongoing research and therefore preliminary and work in progress.

The design starts with the definition of the application class. Current research is performed on the format of this specification and on how an automated flow can process it. The output of this step are HDL (hardware description language) files with RTL (register-transfer-level) and structural code. These instantiate tactical cells as well as reconfigurable cells (e.g. routing box and configuration registers).

These files are fed to a commercial synthesis tool which outputs a netlist. The synthesis tools uses standard cell libraries together with custom libraries which provide optimized cells for the logic module, e.g. multi-bit cells. These are automatically inferred during the synthesis. For interfacing to the commercial tool we prefer open standard formats, especially the EDIF file format (electronics design interchange format) for the netlist and the Open Liberty Library file format for the custom library.

The placement and routing of the netlist is performed by automated commercial tools. The proposed design flow also includes manual placement and routing on a hierarchically higher level. Therefore the automated tools are used for small blocks. Several of these are then placed and routed by hand. This can provide improved delay and area characteristics by utilizing structural knowledge.

The post-silicon phase depends on the resources provided by the pre-silicon phase. To import the description of the reconfigurable architecture and its inventory and routing resources, the netlist from the pre-silicon phase is loaded. Alternatively the pre-silicon tool could generate a machine readable architecture description besides the previously mentioned HDL files, which is used instead of the netlist.

Two differently complex variants of the post-silicon synthesis are planned. The simple variant provides a graphical representation of the hierarchical reconfigurable architecture and permits to configure every cell including the routing connections to implement the actual application. After the module configuration is completed, the bit stream is generated utilizing the knowledge of the configuration chain. This variant does not require any higher intelligence in the tool and provides the user with full control over the configuration.

A lot more intelligence is required for the complex variant tool. This supports the synthesis of an HDL description of the actual application. The resulting netlist is mapped to the underlying technology provided by the pre-silicon phase. The final automatic place and route results in the same data as the user has to setup by hand with the simple variant tool. After that the configuration bit stream is generated by the same routine as used by the simple variant.

V. Exemplary Application

From the many components used in a reconfigurable module like combinational logic, registers, a finite state machine (FSM), the data path, routing and parameterization values, we pick the FSM and give an example of the above described design flow. For simplicity we only look at a single component instead of a whole reconfigurable module.

A. Transition-based Reconfigurable FSM

The Transition-based Reconfigurable FSM (TR-FSM) was presented in [7]. It is a reconfigurable cell which is then integrated into the whole reconfigurable logic module. Based on the fact, that FSMs in actual applications have considerably less transitions than the theoretical maximum for a given number of states and input signals, the TR-FSM focuses on the *transitions* rather than on the state transition function. It provides several smaller reconfigurable blocks for implementing each transition (see Fig. 4).

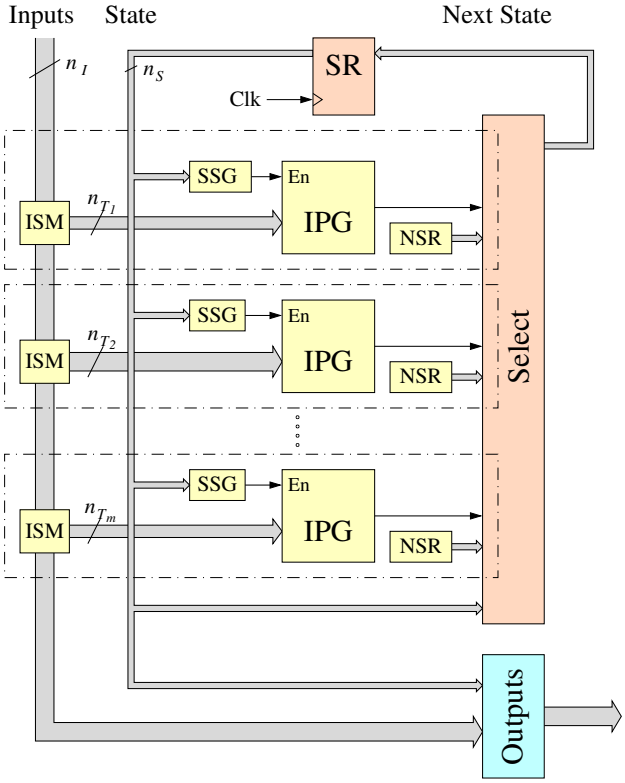

Fig. 4. The overall TR-FSM architecture (SR: state register, ISM: input switching matrix, SSG: state selection gate, IPG: input pattern gate, NSR: next state ID register).

The basis of the proposed architecture is the so called *transition row*. A state selection gate (SSG) compares the current state to its configured value and enables the transition row. A reconfigurable input switching matrix (ISM) selects a subset of the input signals which then serve as the inputs for the input pattern gate (IPG), which is a reconfigurable combinational logic block, e.g. a look-up table (LUT). If activated, the IPG outputs a "1" iff the input pattern matches a certain transition from the

recognized state. The reconfigurable next state ID register (NSR) of the corresponding transition row is then selected by a multiplexer ("Select") and fed back to the state register (SR) as the new state of the FSM.

The overall architecture contains many such transition rows with varying input width. The output signals are computed in a separate configurable logic block ("Outputs" in Figure 4), which for example can be a LUT or an embedded FPGA IP. The output pattern can also be included in every transition row, but this will increase the number of required transition rows if the output pattern depends on input patterns for common next states.

B. Logic Design

In the *pre-silicon phase* all parameters for the chip circuitry have to be specified. Regarding the TR-FSM these are the number of input and output signals, the width of the state vector, the number of transition rows and the number of inputs for each transition row. These parameters are derived from the specification of the application class. This is either given by a set of predefined automata which have to be implemented by the TR-FSM or by a list of common properties of the implemented automata.

The first case is investigated in more detail here. Besides the input and output signals of these automata, the state transitions are investigated to find the total number of transitions and the number of inputs from which each of these depends. The TR-FSM hardware is then specified by the maximum of transitions and the maximum of the numbers of the transition rows. Note that a transition row with n_{T_i} inputs can also implement a transition with $n_{T_j} < n_{T_i}$ because the IPG is able to ignore the state of inputs. For example, one FSM requires 7 states with 3 inputs each plus 3 states with 2 inputs each. Another FSM requires 3 states with 3 inputs each but 7 states with 2 inputs each. In this case, we need 7 transition rows with 3 inputs plus 3 transition rows with 2 inputs. Four of the 7 states with 2 inputs each of the second FSM can be mapped to transition rows with 3 inputs. Additionally the TR-FSM offers the flexibility to implement either an FSM with a high number of states with very few transitions each, as well as an FSM with a low number of states with many transitions each or a combination of both.

From this specification the semiconductor is produced. In the *post-silicon phase* the actual FSM is specified. This includes the number of states and the input patterns for each transition with their next states. From this FSM description the configuration of the ISMs is generated to select exactly these input signals, which are inspected by the respective states. The configuration of the IPGs is generated so that they output a logic "1" in every case (depending on the selected inputs), where the transition is active. Finally the state mapping is performed by assigning a bit pattern to every state which is then used to generate the configuration of the SSGs and NSRs. All this configuration information is then assembled to a single configuration bit stream to be downloaded into the TR-FSM configurable block.

VI. Conclusion

This paper discusses the methodology for designing reconfigurable logic modules for WSN nodes. These are placed additionally to the CPU into a WSN node SoC and autonomously perform tasks of moderate complexity. Due to the reconfigurability of every module, its functionality can be adapted flexibly. By relieving the CPU from these tasks, it will remain in an inactive low-power mode for extended periods which leads to fundamental power reductions.

Every reconfigurable logic module is specifically tailored to the application class of its operational area. In the pre-silicon phase the reconfigurable resources including the routing are specified. After chip production, i.e. in the post-silicon phase, the actual application is specified. The configuration is derived from the application description and mapped to a configuration bit stream. This is applied to the reconfigurable circuitry and so configures it to implement the application logic. The objective is to reduce power consumption while maintaining a high degree of flexibility for the implemented applications.

Precise measurements of the power reduction will be gained and published after the production of a test chip in a 130 nm CMOS process, which is scheduled for end of May 2010. Future work includes research on reconfigurable routing, development of multi-bit cells and a tool chain to ease the implementation of reconfigurable hardware modules.

References

[1] H. Karl and A. Willing, *Protocols and Architectures for Wireless Sensor Networks*. Wiley, 2005.

[2] J. Glaser, J. Haase, M. Damm, and C. Grimm, "A Novel Reconfigurable Architecture for Wireless Sensor Networks," in *Tagungsband zur Informationstagung Mikroelektronik 10*. Vienna, Austria: OVE, 7.-8. Apr. 2010, pp. 284–288.

[3] ——, "Investigating Power-Reduction for a Reconfigurable Sensor Interface," in *Proceedings of Austrochip 2009*, Graz, Austria, 7. Oct. 2009.

[4] A. Lodi, A. Cappelli, M. Bocchi, C. Mucci, M. Innocenti, C. D. Bartolomeis, L. Ciccarelli, R. Giansante, A. Deledda, F. Campi, M. Toma, and R. Guerrieri, "XiSystem: A XiRisc-Based SoC With Reconfigurable IO Module," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 1, Jan. 2006.

[5] S. J. E. Wilton, N. Kafa, J. C. H. Wu, K. A. Bozman, V. O. Aken'Ova, and R. Saleh, "Design Considerations for Soft Embedded Programmable Logic Cores," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 2, pp. 485–497, Feb. 2005.

[6] V. Aken'Ova, G. Lemieux, and R. Saleh, "An Improved "Soft" eFPGA Design and Implementation Strategy," in *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC)*, 18.-21. Sep. 2005, pp. 179–182.

[7] J. Glaser, M. Damm, J. Haase, and C. Grimm, "A Dedicated Reconfigurable Architecture for Finite State Machines," in *Reconfigurable Computing: Architectures, Tools and Applications, 6th International Symposium, ARC 2010*, ser. Lecture Notes on Computer Science, vol. LNCS 5992. Bangkok, Thailand: Springer, Mar. 2010, pp. 122–133.

[8] J. O. Filho, T. Schweizer, T. Oppold, T. Kuhn, and W. Rosenstiel, "Tuning Coarse-Grained Reconfigurable Architectures towards an Application Domain," in *IEEE International Conference on Reconfigurable Computing and FPGA's, ReConFig 2006*, Sep. 2006, p. 7.

[9] R. Hartenstein and V. Milutinovic, "Introduction to the Configware Minitrack – From Glue Logic Synthesis to Reconfigurable Computing Systems," in *Proceedings of the 32nd Annual Hawaii International Conference on System Sciences, 1999. HICSS-32*, vol. Track 3, 1999, pp. 326–326.