MODELLING SELF-ADAPTIVE NETWORKED ENTITIES IN MATLAB/SIMULINK

R. Bartosinski, M. Daněk, P. Honzík, J. Kadlec

ÚTIA AV ČR, v.v.i, Pod Vodárenskou věží 4, Praha 8, 182 08, Czech Republic

Abstract

Recent developments in the digital technology make it cheaper and thus more accessible to an ordinary citizen. Currently almost any everyday device (car, fridge, cell phone) equipped with an electronic intelligence uses a programmable processing element. As the number of such networked devices will likely increase, and given the increased stress on low power consumption due to more strict environmental limitations, it is necessary to research novel techniques to program and use electronic devices more efficiently. This paper presents a framework for building and modeling new-generation self-adaptive systems. The first part of the paper describes an FPGA platform that implements a self-adaptive computing networked entity (SANE) that forms the basic element of the approach. The second part of the article describes a simulation and implementation framework in Matlab/Simulink for developing SANEs in FPGAs.

1 Self-Adaptive Networked Entity

We define self-adaptivity as the ability of a system to adapt to an environment by allowing its components to monitor this environment and change their behaviour in order to preserve or improve the operation of the system according to some defined criteria [2]. The environment of a system is defined by everything that interacts with this system. The adaptation of a system can occur at different levels, from hardware to software. In this article, we study self-adaptivity at the hardware level and especially with reconfigurable architectures.

Figure 1 shows a functional view of the SANE that takes into account the definition given in the previous paragraph and the discussions about implications of self-adaptivity at the hardware level.

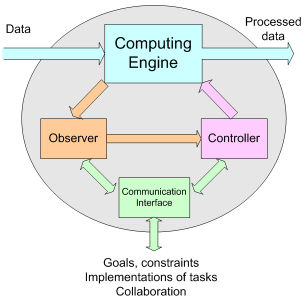


Figure 1: Functional view of the SANE with the four main parts.

Let us describe the different parts shown in Figure 1:

- The computing engine processes data. It must provide the necessary processing power to handle future complex algorithms required by new standards and multimedia applications. It must also be reconfigurable to handle a wide spectrum of applications, and as the user needs may vary when the system is online, this reconfiguration process must be dynamic ("on the fly" reconfiguration). It is the most flexible block of the SANE.

- The observer is responsible for monitoring the computation process and other runtime parameters. It allows the SANE to sense its current environment and optimize its performance, which means both comparing its actual processing parameters with the required constraints given by the application designer, and monitoring communication parameters.
- The controller is in charge of actually taking all decisions regarding the ongoing computation task. Based on the observations of the current running task, the controller is responsible for changing the state of the SANE by loading any locally available task implementation. The computing tasks that are loaded can be pre-synthesized hardware tasks best seen as configurations for an FPGA-like fabric or pre-compiled code best seen as binaries for a CPU soft core, both implemented in the computing engine.
- The communication interface is responsible for the management of SANE assemblies. It enables resources sharing and collaboration between SANE elements as well as providing the SANE with goals to be reached.

These four main functionalities are embedded in a common box which is the SANE. This box has three links with the external world (which might be joined for implementation purposes): the input and the output links are directly related to data processing since the SANE has a dataflow model. The communication link is dedicated to communications among different SANE elements.

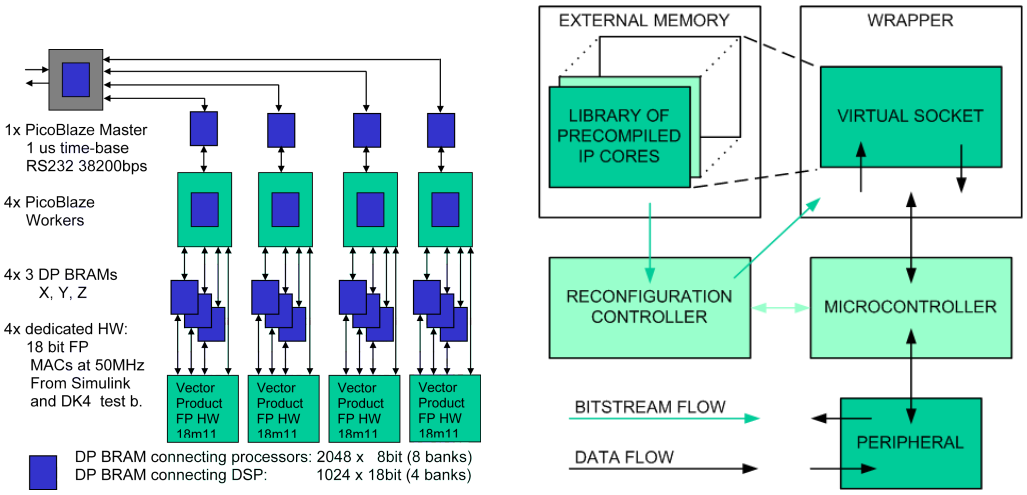
The SANE is basically a tightly coupled hardware/software entity with its computing engine seen as a hardware-based reconfigurable computing unit (e.g. an FPGA fabric), and the observer and the controller implemented as more software programmable CPU cores.

2 FPGA Background

The platform used to model a network of SANEs consists of a hierarchical connection of simple elements. Each element is formed by a simple CPU connected to special-purpose hardware, both implemented as FPGA macros. Two things deserve attention that can be used to implement self-adaptation required by the SANE elements - the possible dynamic reconfiguration of the HW accelerator in FPGAs that support it (such as Atmel or Xilinx), and the possible reprogramming of the CPU program memory. The CPU reprogrammability itself increases reusability of the fixed hardware accelerator (meaning its one specific HW configuration), since by using for example different data indexing orders different computations can be performed with fixed operations in HW (e.g. floating-point addition or floating-point multiplication); it can also be used to emulate dynamic reconfiguration for FPGA platforms that currently do not support it (Altera, Actel).

The simple CPU is meant to execute small programs up to several hundreds of machine-level instructions, such as computation of memory addresses for the HW accelerator, or adaptation to different characteristics of the processed data. Another important issue is that the use of the CPU to control the flow of computation reduces the complexity of datapaths and thus makes better use of FPGA resources.

Since in future applications we foresee a possibility to instantiate or remove SANEs as demanded by varying user requirements (QoS), it does not make sense to consider hard-core CPUs for the SANE elements, since these do not offer the required flexibility to instantiate or remove SANEs during computation.



(a) Master-Worker PicoBlaze network.

(b) Infrastructure for dynamic reconfiguration of FPGAs.

Figure 2: Master-Worker platform. Infrastructure for dynamic reconfiguration.

2.1 The Master/Worker Platform

In general, floating point DSP algorithms in FPGAs require single cycle parallel memory accesses and effective use of pipelined arithmetic operators. We take advantage of the fact that many common DSP vector and matrix operations can be split into batch calculations that fulfil these requirements. We consider architectures that consist of simple PicoBlaze worker processors, each with a dedicated DSP-HW accelerator (see Figure 2a). Each worker can do preparatory tasks for the next batch in parallel with its HW accelerator. Once the DSP-HW accelerator finishes the computation it issues an interrupt to the worker. The job of the workers is to combine the accelerated parts of the computation into a complete DSP algorithm.

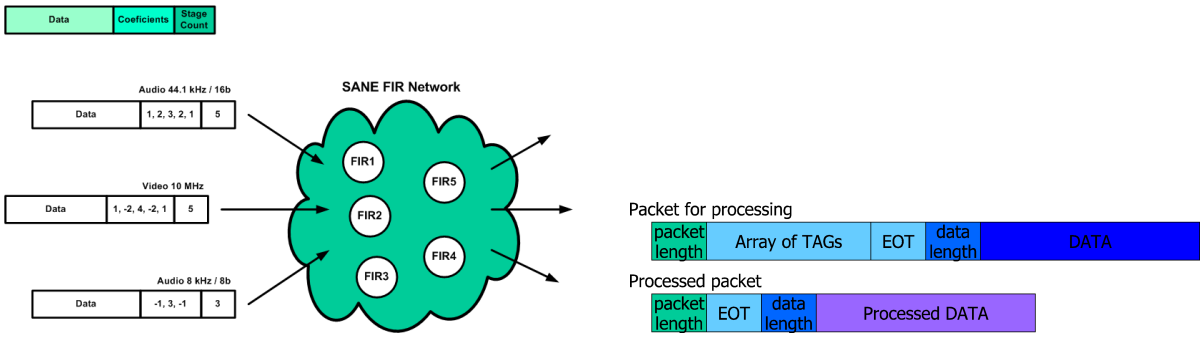
It is ideal if we limit implementations to batch operations of each worker that read data from an input block RAM (BRAM), perform a relatively simple sequence of pipelined operations at the maximum clock speed, and return the result(s) back to another (output) BRAM. These primitives can be mapped effectively to hardware, including a complete autonomous data-flow control in hardware. The related dedicated generators of address counters and control signals can be effectively coded in an m-file and processed in the Xilinx System Generator. Simulink can be used for fast derivation of bit-exact models of the batch calculations in the DSP-HW accelerators.

For more details on the Master/Worker platform see [5].

2.2 Partial Runtime Reconfiguration of FPGAs

In designs with partial reconfiguration the FPGA is divided into a static part and a dynamic part. The static part contains the datapaths and interfaces that implement dynamic reconfiguration. The dynamic part is formed by a so called virtual socket that can contain one or more pre-compiled IP cores. The reconfiguration controller manages data transfers (data=configuration bitstreams) between an external bitstream memory and the FPGA configuration memory. The microcontroller provides an application interface to the IP core, i.e. it enables to access its function through standard C programming.

The infrastructure depicted in Figure 2b provides support for loading and clearing a pre-compiled (i.e. pre placed and routed) IP core in an FPGA, a configuration controller in a



(a) The FIR SANE network. (b) Encoding of data packets.

Figure 3: The FIR SANE network. Encoding of data packets.

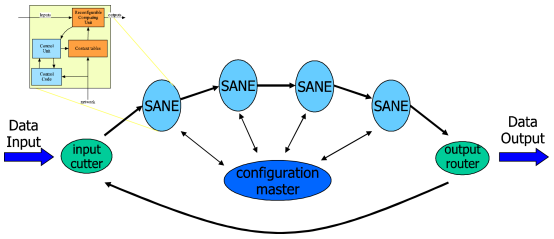


Figure 4: General view of the SANE network.

microcontroller, and a SANE control program programmed in the C language.

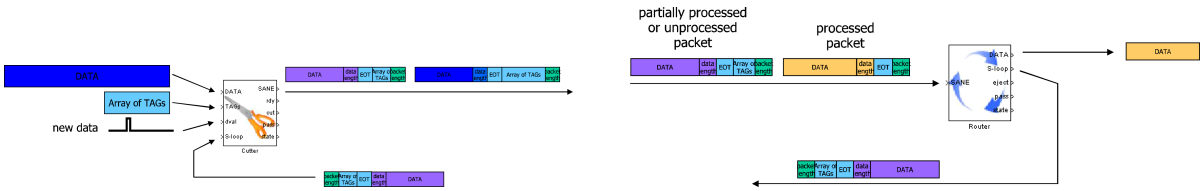
For more details on dynamic reconfiguration of FPGAs see [4], [1], [3].

3 Modelling and Implementing a SANE Network

The SANE approach is tested on a task where the SANE network implements an arbitrary number of FIR filters with a varying number of taps. The task of one SANE is to read an input data packet, process it and generate an output data packet. The initial idea of the task is shown in Figure 3a. The network operates on tagged data packets, a concept often used in data-flow processing. Each packet is formed by a header and data part. The header specifies in some way operations needed to process the data part. In the example given in the figure, the first number in the header specifies the order of FIR required to process the data with the next numbers specifying the FIR weights. The header can be viewed in a more general way as a sequence of tags that specify a sequence of operations required for the attached data (see Figure 3b).

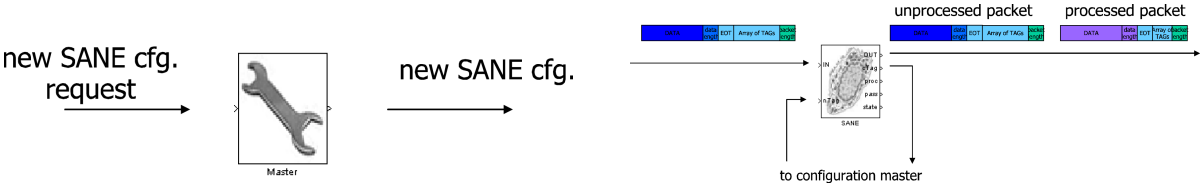
The whole network of FIR SANE elements can be modelled as a linear chain of processing elements with a FIFO memory forming a loop to circulate data not completely processed (see Figure 4). The use of this topology (ring) enables us to concentrate on the SANE computation only, since ring can emulate any given network topology on a logical level. We are aware of the importance of a proper networking scheme, and it will be addressed in more advanced stages of this research.

The SANE network has been modelled in the Matlab/Simulink environment and verified on the Celoxica RC10 boards acting as hardware in the loop. The idea is to use the Simulink environment for modelling, visualization and debugging. A major advantage of this approach is that it allows us to gradually move from the software model to the hardware implementation. Another advantage is the possibility to use any FPGA platform provided it supports a standard simple data exchange protocols; this way we can easily mix implementations on different boards and with FPGAs from different families and manufacturers.



(a) Input cutter. (b) Output router.

Figure 5: SANE model, input cutter and output router.



(a) Configuration master. (b) SANE element.

Figure 6: SANE model, configuration master and SANE element.

The following sections briefly describe the blocks used to model the SANE network in Simulink.

3.1 Input Cutter

The input cutter takes an input data stream and divides it into packets of data. We assume packets with the same length, but this is not necessary in all cases. In addition the cutter prepares the packet headers for the data. In the simplest case the header contains an ordered array of tags that indicate operations that are required to process the data. In more advanced stages of this experiment the array of tags will merely indicate the current state of the data and its desired state, with the proper sequence of operations decided by the SANE network itself.

3.2 Output Router

The output router is responsible for directing processed data out of the network, and to direct partially processed data packets back to the network. The processed data are recognized by an empty array of tags. Partially processed data are then stored in a FIFO memory in the feedback loop (not shown).

3.3 Configuration Master

The configuration master is responsible for managing the database of configuration bitstreams for the reconfigurable FPGA part and binary programs for the CPU part of the SANE elements. The master sends configurations on demand from individual SANE elements. The master by no means introduces central control to the SANE network since this would invalidate the distributed processing character of the experiment.

3.4 SANE Element

The SANE element monitors the character of data that pass through it, it processes data packets with tags that match its functionality. The distributed control of the network is implemented

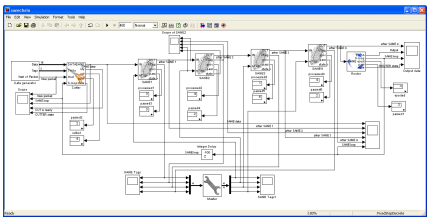


Figure 7: SANE model, overall view in Simulink with signal scopes.

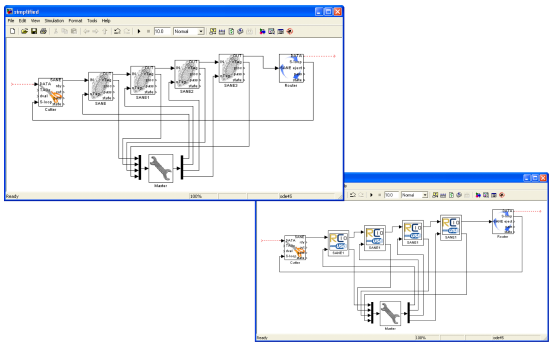


Figure 8: SANE model, simplified view, SW model (top left) and hardware in the loop (bottom right).

as individual local decisions by each SANE to change its internal configuration to match the majority of passing data tags and subsequent requests for new configurations to the configuration master (in future experiments SANE elements will be able to locally copy or exchange their configurations without involving the master).

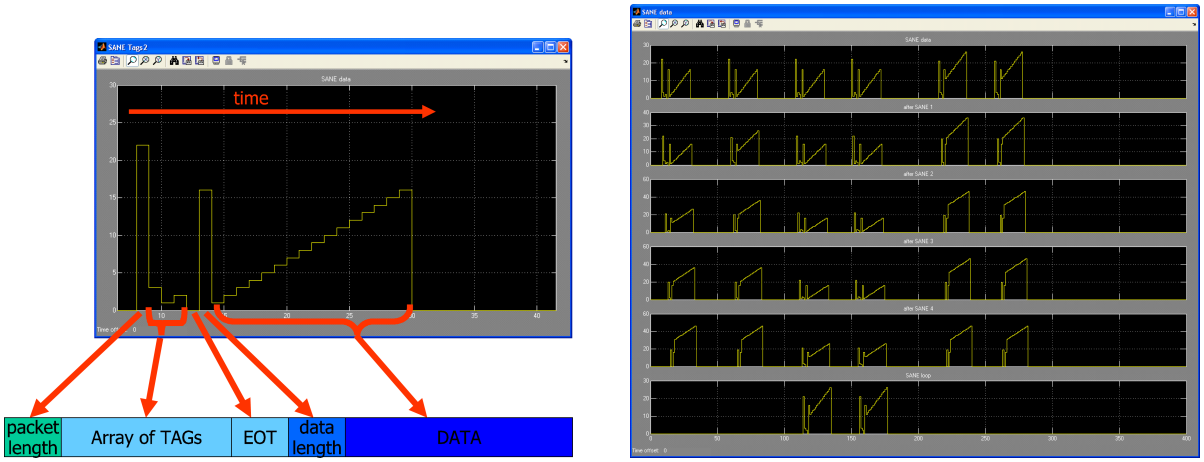
3.5 SANE Model in Simulink

We use the above building blocks to assemble a Simulink schema that models the SANE network. Figure 7 shows a sample network with 4 SANE elements, one input cutter, one output router and a configuration master. The figure shows the complete settings with all signal scopes that are used to log data transfers in the network and the evolution of SANE internal configurations. A clearer view is showed in Figure 8; the top left part shows a situation where the whole network is modelled in Simulink, the bottom right part shows SANEs implemented in individual FPGA development boards (in our case Celoxica RC10) and connected to the Simulink environment as hardware in the loop.

Figure 9a shows how SANE packets are mapped to Simulink data. Packets are formed by consecutive numbers that are visualised as Matlab value plots. One data cycle is shown in Figure 9b, with lines showing data that passed through all connections in the SANE network (i.e. after the input cutter, after SANE1, after SANE2, after SANE3, before the router, in the feedback loop).

Figure 10a shows how SANE configuration evolves in time for all four SANEs in the network. A configuration is represented by a number that corresponds to a specific tag or group of tags that a SANE can process.

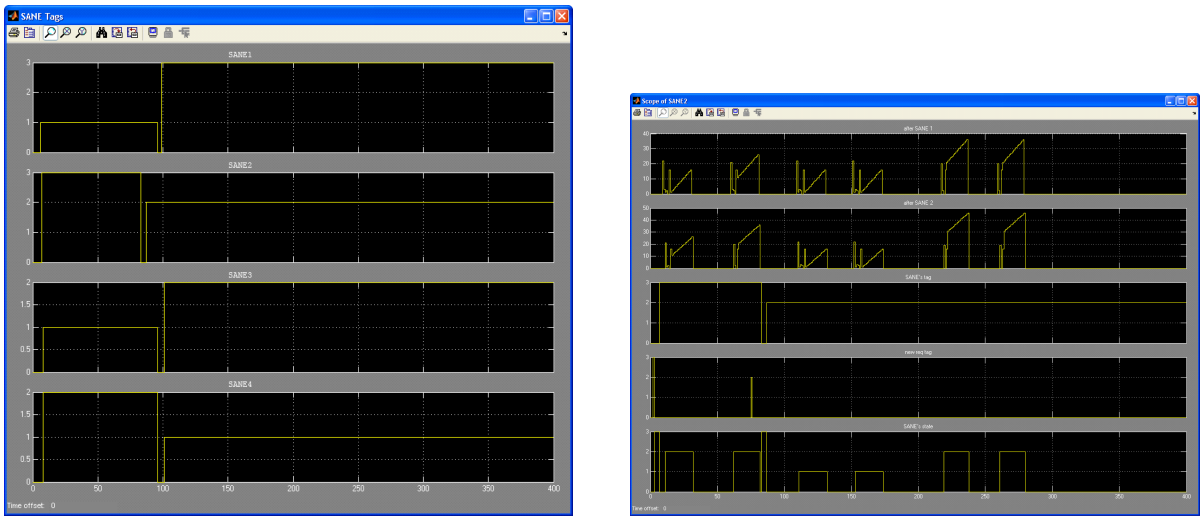
Figure 10b shows an evolution for SANE2. The top line shows input data, the next line output data. The last three lines show the evolution of SANE configuration, requests for reconfiguration, and a SANE internal state.



(a) Interpretation of signal plots.

(b) Data flow in the SANE network.

Figure 9: SANE model, data flow in the SANE network.



(a) SANE configurations in time.

(b) Data view for SANE unit 2.

Figure 10: SANE model, configurations in time.

4 Conclusions

This paper has presented a modelling and implementation platform for building novel systems based on self-adaptive elements called Self-Adaptive Networked Entities (SANEs). The design of such a system has been illustrated on an approach where dynamic reconfiguration can be used together with design decomposition to increase design reuse and decrease power consumption. A Matlab/Simulink block set for modelling SANE-based systems has been introduced, and it has been demonstrated on a model of a SANE network that computes FIR filters with FPGA kits acting as hardware in the loop.

References

[1] R. Bartosinski, M. Daněk, P. Honzík, and R. Matoušek. Dynamic reconfiguration in fpga-based soc designs. In J. Sziray et al., editor, *Proceedings of IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop DDECS2005*, pages 129–136. University of West Hungary, 2005.

[2] The Aether consortium. *The AETHER project web page - http://www.aether-ist.org.* The AETHER consorcium, 2006.

[3] The RECONF2 consortium. *The RECONF2 project web page - http://reconf.org.* The RECONF2 consorcium, 2002.

[4] M. Daněk, P. Honzík, J. Kadlec, R. Matoušek, and Z. Pohl. Reconfigurable system-on-a-programmable-chip platform. In E. Gramatová et al., editor, *Proceedings of IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop DDECS2004*, pages 21–28. Slovak Academy of Sciences, 2004.

[5] J. Kadlec and S. Chappel. Implementing floating-point DSP. *Xilinx Embedded Magazine*, 2(3):12–14, 2006.

Roman Bartosinski
Academy of Sciences of the Czech Republic,
Institute of Information Theory and Automation,
Department of Signal Processing,
Pod vodárenskou věží 4, 18208 Praha 8, Czech Republic
e-mail: bartosr@utia.cas.cz
www: sp.utia.cz

Martin Daněk
Academy of Sciences of the Czech Republic,
Institute of Information Theory and Automation,
Department of Signal Processing,
Pod vodárenskou věží 4, 18208 Praha 8, Czech Republic
e-mail: danek@utia.cas.cz
www: sp.utia.cz

Petr Honzík
Academy of Sciences of the Czech Republic,
Institute of Information Theory and Automation,
Department of Signal Processing,
Pod vodárenskou věží 4, 18208 Praha 8, Czech Republic
e-mail: peters@utia.cas.cz
www: sp.utia.cz

Jiří Kadlec
Academy of Sciences of the Czech Republic,
Institute of Information Theory and Automation,
Department of Signal Processing,
Pod vodárenskou věží 4, 18208 Praha 8, Czech Republic
e-mail: kadlec@utia.cas.cz
www: sp.utia.cz