

SystemC-AMS SDF Model Synthesis for Exploration of Heterogeneous Architectures

Popp, Andreas; Herrholz, Andreas; Gruettner, Kim; Le Moullec, Yannick; Koch, Peter; Nebel, Wolfgang

Published in:

Proceedings of the 13th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems

DOI (link to publication from Publisher):

[10.1109/DDECS.2010.5491801](https://doi.org/10.1109/DDECS.2010.5491801)

Publication date:

2010

Document Version

Accepted author manuscript, peer reviewed version

[Link to publication from Aalborg University](#)

Citation for published version (APA):

Popp, A., Herrholz, A., Gruettner, K., Le Moullec, Y., Koch, P., & Nebel, W. (2010). SystemC-AMS SDF Model Synthesis for Exploration of Heterogeneous Architectures. In *Proceedings of the 13th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems* (pp. 133-138). IEEE (Institute of Electrical and Electronics Engineers). <https://doi.org/10.1109/DDECS.2010.5491801>

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal -

Take down policy

If you believe that this document breaches copyright please contact us at vbn@aub.aau.dk providing details, and we will remove access to the work immediately and investigate your claim.

SystemC-AMS SDF Model Synthesis for Exploration of Heterogeneous Architectures

Andreas Popp*, Andreas Herrholz[†], Kim Grüttner[‡],
Yannick Le Moullec*, Peter Koch* and Wolfgang Nebel[‡]

*Center for Software Defined Radio, Department of Electronic Systems, Aalborg University, Denmark
Email: {anp,ylm,pk}@es.aau.dk

[†]OFFIS, R&D Division Transportation, Oldenburg, Germany
Email: {andreas.herrholz,kim.gruettner}@offis.de

[‡]Carl von Ossietzky University Oldenburg,
Faculty II - Department for Computer Science, Division Embedded HW/SW Systems, Oldenburg, Germany
Email: wolfgang.nebel@informatik.uni-oldenburg.de

Abstract—Cost efficient design of embedded HW/SW systems that need to meet certain requirements is a complex task due to the huge number of possible solutions, the "design space". Design space exploration methods depend on the designers' input in terms of application description, target architecture, and cost estimates for implementation alternatives. Obtaining feasible pre-implementation cost estimates causes lots of effort since the designer does not have confident information before implementation on the target architecture, or even different target architectures, has been performed.

In this paper we present a methodology suitable for automatic cost estimation of synchronous data flow (SDF) graphs. We propose to start from an executable SystemC-AMS SDF specification, and demonstrate its automatic transformation and implementation for cost estimation on heterogeneous HW/SW architectures. The presented methodology allows the estimation of both HW and SW implementation alternatives of each SDF node based on a quick synthesis approach. These cost estimates are fed to a mapping framework to obtain a static binding and schedule for the architectures under exploration. With the proposed methodology the designer does not have to perform full synthesis and implementation for design space exploration. This is demonstrated by a case study of a Bluetooth baseband unit considered for implementation on a Xilinx Virtex-5 FPGA.

I. INTRODUCTION

Heterogeneous architectures consisting of different kinds of processing units, like CPUs, DSPs, ASIPs, and custom hardware, provide a vast amount of different implementation alternatives for a given application. In order to obtain the most cost efficient implementation as a System On-a-Chip (SoC), different architecture configurations in terms of area consumption, execution speed etc. need to be evaluated.

However, this is an extensive task due to the many degrees of freedom the designer has in aspects of

- 1) design of the processing and communication architecture, and
- 2) options for selecting where and when to perform which tasks (binding and scheduling).

The design of heterogeneous HW/SW systems is aided by the use of methodologies that output a binding and a schedule.

We propose to build upon our existing mapping framework [1]. The mapping framework handles heterogeneous static HW/SW architectures and takes as input specifications: 1) application model, 2) architecture model, and 3) cost attributes and mapping constraints.

The mapping framework is a combination of multiprocessor scheduling and pre-scheduling of HW/SW partitions. Following the specification, the application's tasks (described by a directed acyclic data flow graph) are partitioned between HW and SW units based on execution time. The HW flow utilises temporal partitioning to create sequentially executable clusters of HW configurations, represented by super-nodes. The execution time of the super-node is the maximum execution time of the tasks it contains. For static architectures, a HW super-node is composed based on which tasks are available first. The rest of the tasks are repartitioned to SW. The super-nodes are fed back to the original SW multiprocessor scheduling flow via an updated task graph. The scheduler uses the Extended DLS scheduling algorithm [13] for heterogeneous multiprocessor architectures to schedule SW and HW execution. The scheduling algorithm takes into account interprocessor communication between SW units as well as the HW/SW communication.

The result of the mapping framework mainly depends on the input of cost estimates for given architecture elements and the binding of tasks. Thus it is crucial to have feasible estimates for the mapping to the examined execution architectures in order to obtain realistic exploration results.

In our previous work the experiments were based on abstract application and costs estimates. Therefore, we propose a methodology to provide and obtain such information based on a specification of the application via a Synchronous Data Flow (SDF) model [3]. The methodology presented here serves as a preprocessor for the mapping framework. Thus, our contribution is an approach for HW/SW implementation cost estimation from an SDF application model combined with a heterogeneous multiprocessor system scheduling framework.

The paper is organised as follows: First, we take a look at related work and describe the proposed methodology for

automatic cost estimation of SystemC-AMS models for heterogeneous HW/SW architectures. The methodology will be presented along a Bluetooth baseband processing unit from the domain of Software Defined Radio (SDR). This is followed by the application of the methodology to the Bluetooth case study and an exploration of different target architecture mappings is presented. The paper closes with a conclusion.

II. RELATED WORK

Initial work on finding schedules for parallel and sequential execution of SDF models has been presented in [3]. In [4] this work has been extended towards buffer minimisation for sequential execution on a single processor. More recently, a buffer-minimising method for mapping SDF models on heterogeneous HW/SW architectures is presented in [2]; however, it is based on a formal non-functional model and does not consider implementation costs.

There is a lot of previous and ongoing work in the field of hardware and software synthesis from SDF and other data flow models. Most works on software synthesis are based on the work on static scheduling of SDF models but typically they do not take into account costs like area and system performance. In [6] a method for generating hardware from SDF models is proposed based on existing work on software synthesis. While it includes proposals for different hardware architectures, it does not explicitly consider any modelling language or tool-based design automation. Other approaches for hardware synthesis, as in [7] and [8], are based on predefined building blocks restricting the set of available computation primitives. A complete and automated design flow for SDF based hardware synthesis based on the actor language CAL is presented in [9].

A design space pruning tool for FPGA design is presented in [10] where the application is specified in C. Each operation corresponds to one or more basic Register Transfer Level (RTL) architecture elements, and the cost of different RTL datapaths is based on scheduling onto a combination of those basic RTL elements. A quite similar approach to ours has been presented in [11]. The flow is based on a tool called *SystemCoDesigner* enabling automated exploration and system-level synthesis of HW/SW systems for data flow applications. Initial specification is done in terms of a dynamic data flow model called *SystemMoC* using a set of predefined SystemC modelling elements. This is different to our work, as we use SystemC-AMS as an initial specification of the application. To the best of our knowledge, there is no existing work on hardware synthesis of SDF models using SystemC-AMS for initial specification and SystemC for final hardware implementation.

III. METHODOLOGY

The methodology presented in this work is outlined in Fig. 1. The user application is specified as a SystemC-AMS [14] SDF model which describes the tasks to be performed, their interdependency, and their activation based on a user-defined transaction container, also called “token”. The

SDF model of computation has been selected as it allows the calculation of a static schedule [3], and thus allows its automatic transformation into an acyclic task graph required by the mapping framework. We have chosen SystemC-AMS, because it adds an SDF-layer to the SystemC discrete event simulation kernel enabling C/C++ based specification of executable SDF models and their integration into system level models. Furthermore it is non-proprietary, freely available and has recently become an Open SystemC Initiative standard.

The architecture model in our methodology is composed of architecture templates: Software processing elements with local memory, dedicated hardware processing elements, and communication infrastructure for the interconnection of these processing elements.

The cost estimation, as shown in Fig. 1, performs a characterisation of each SDF module for each processing element of the architecture template library. Our proposed cost estimation approach is based on automatic code transformation which allows the synthesis of the behaviour of each SDF module to either dedicated hardware or software. For hardware cost estimation we perform SystemC to VHDL synthesis with our synthesis tool [15]. Logic synthesis for the chosen target technology (e.g. Xilinx Virtex-5 FPGA) allows accurate cost estimates in terms of area, critical path length in terms of f_{\max} , and number of clock cycles per activation of each SDF module. For software cost estimation we propose the use of a lightweight SystemC data type library which can be compiled along with the behavioural code of the SDF block. Therefore, our methodology allows the direct cross compilation and profiling of the SDF module on the chosen target processor. For obtaining the number of clock cycles per SDF module activation we use a generic test bench for HW and SW modules. It generates input stimuli for profiling, based on recorded traces of the SystemC-AMS simulation model. This allows to profile the minimum, average, and maximum number of clock cycles per activation.

To guarantee the correct communication and activation according to the SDF model of computation, our methodology provides certain HW Module and SW Module wrappers, which can be connected via FIFOs. The communication controllers for the FIFOs are part of the architecture template library and can be customised in terms of packet format (i.e. payload or token), communication width, and FIFO depth (number of payloads per FIFO).

The mapping framework takes an architecture configuration, which is an instantiation of processing and communication elements of the architecture template library. The second input of the mapping framework is an acyclic task graph which is generated from the SDF graph through static scheduling.

In the following, we describe the methodology and illustrate it by a case study.

A. Case Study

The performed case study is based on a baseband processing unit for a Bluetooth (IEEE 802.15.1) transmitter [12]. The baseband unit processes data and packs it into packets that are

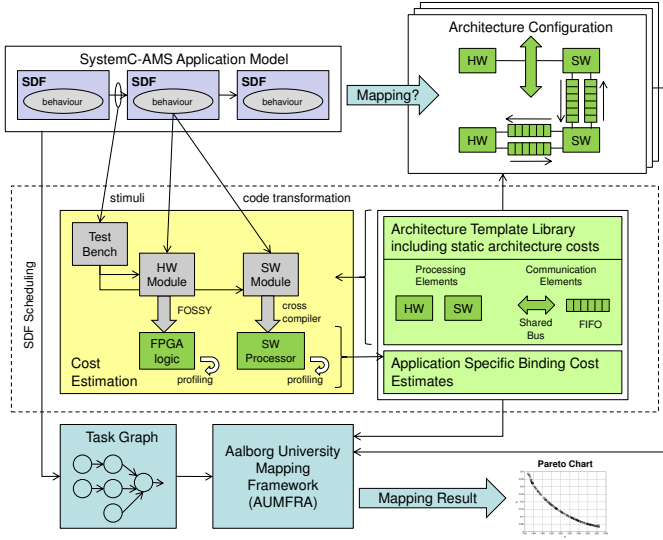


Fig. 1. Overview of the proposed methodology.

transmitted by the modulator and RF front-end in time-slots of 625 μ s. The SDF representation of the baseband unit is shown in Fig. 2a. It composes a packet based on three parts; synchronisation word, packet header, and payload. We have chosen to focus on the mode of operation with the most processing requirements, the Frequency Hop Synchronisation (FHS) or Data-Medium Rate (DM1) payload types including Cyclic Redundancy Check (CRC) checksums to payload, whitening, and (10,15) Hamming Forward Error Correction [12].

The first step of the methodology is the transformation of the SDF graph into an acyclic task graph which can be processed by the mapping framework.

B. Application Model (Task Graph)

The application input model of the mapping framework is a directed acyclic data flow graph, $G = (V, E)$, where the nodes, $\{v_0, \dots, v_{n_v-1}\} \in V$, represent tasks. During this work we call it Task Graph. The edges, $\{e_0, \dots, e_{n_e-1}\} \in E$, represent data dependencies. Each edge is assigned a width, w_0, \dots, w_{n_e-1} , describing how much data is generated and consumed by the nodes.

This application model is derived from the SDF specification through graph transformation. It starts with the calculation of a static schedule of the SDF specification which can be represented as an Acyclic Precedence Graph (APG) [5]. The above mentioned application task graph only allows sequential execution along the dependency of tasks. Therefore, the APG needs to be “folded” into a sequential order of tasks. A valid static schedule of the SDF graph of the case study from Fig. 2a is shown as an acyclic task graph model in Fig. 2b.

The SDF scheduling and graph transformation for the case study consist of the following steps:

- 1) Compute a static schedule of the SDF model and map the scheduling sequence into an acyclic task graph.

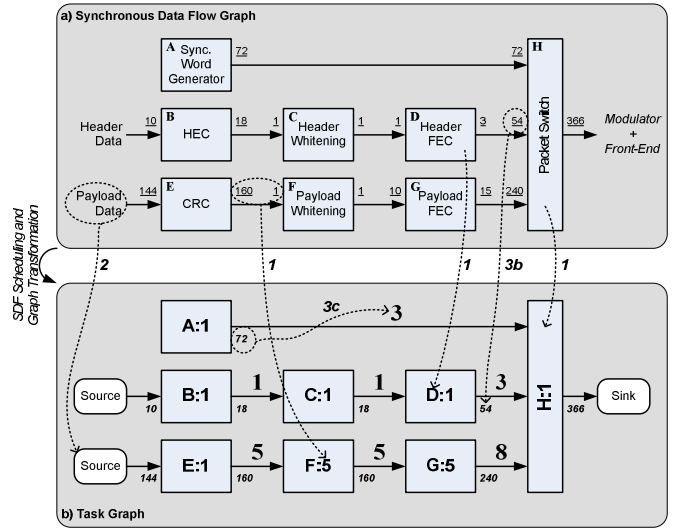


Fig. 2. Illustration of the SDF scheduling and graph transformation procedure. The dashed arrows relate to a graph transformation step, with a number related to the steps in Section III-B. **a)** Synchronous Data Flow graph for Baseband data path of the BlueTooth transmitter. The underlined numbers on the arrows are tokens, describing the relation between quantity of input and output data (in bit). The switch composes the packet from the three parts: synchronisation word, header, and payload. The header and payload data have been appended error checksums, whitened, and forward error correction coded [12]. The static schedule consists of three parallel paths: 1) A, 2) B-C-D, and 3) E-F-G, followed by the join node, H, after all three paths are finished. **b)** Task Graph for case study. The graph is derived from Fig. 2a through our proposed method. The nodes denote operations and the edges denote communication. The letters inside nodes relate to the task in Fig. 2a. The numbers inside nodes determine the number of times the functionality of the task is called before the node is finished. This is determined by the number of activations in the static schedule. The **bold italic** numbers is the total number of input and output bits of a node. The edges are annotated with a number, indicating the number of 32-bit data packets transferred between tasks.

- Allow parallel execution of paths, but sequential execution of all nodes in each path.
- Disregard the number of tokens in the SDF model.
- If an SDF node is executed several times, its functionality is executed several times inside the node of the Task Graph (indicated with a number inside the node in Fig. 2a).

- 2) Add source and sink nodes, as the mapping framework needs these nodes.
- 3) Determine the number of data packets transferred between nodes
 - a) Determine basic data packet size for data transfer between tasks. In the case study, we have selected 32 bits.
 - b) Based on the static schedule, determine the total number of input and output bits of each node.
 - c) Divide the input and output bits by the basic data packet size and round towards the ceiling to normalise the number of data packets.
 - d) No packets are allocated on the edges from source or into sink nodes, as this data is assumed available for processing.

C. Architecture Model

The architecture model is an abstract architecture model describing the types and numbers of processing elements that are available, as well as how they are interconnected. The processing elements, $p_k \in P$, are assigned an index, $k = \{0, \dots, n_{pu} - 1\}$, and a type, ($sw \in PE_{SW}$, $hw \in PE_{HW}$). PE_{SW} and PE_{HW} are the available sets of SW and HW processing elements, such as the software processor type and the type of reconfigurable logic fabric. Interconnection of processing elements is described in terms of buses, $b_m \in B$, also assigned an index, $m = \{0, \dots, n_b - 1\}$. The buses are furthermore assigned a direction (unidirectional or bidirectional), width, and a speed (as time it takes to transfer one package of data with the size of the bus width). According to its type, the processing and communication elements are assigned their corresponding resource utilisation in terms of Look-Up Tables (LUTs), Flip-Flops (FFs), and Block RAMS (BRAMs).

The cost estimates provide the interrelation between the application and the architecture model. Each task, $v_i \in V$, has at least one cost entry, and there exist cost entries for each binding alternative of a task. The costs are execution times for both HW and SW units, as well as resource utilisation for custom HW units. Thus every cost entry contains at least a task index, a processing unit index, and an execution time.

The architecture investigated for implementation is the Virtex-5 FPGA. The architecture model consists of a number of processing elements that are either a MicroBlaze SW processor or dedicated hardware logic. The processing elements are interconnected by a common bus. The bus is assumed to be implemented by a 32 bit FIFO buffer, the Xilinx Fast Simplex Link (FSL).

The communication model distinguishes between intra and inter processing unit communication. Communication between tasks, executed in the same processing unit, is taken into account by the HW and SW module wrappers. This intra processing unit communication is performed via dedicated and local memories and thus can be considered as side effect free. The inter processing unit communication is implemented over dedicated communication resources with its own timing behaviour. Communication over FSL, as chosen in this work, takes a number of cycles, based on the number of 32 bit data packets that are transmitted. The communication time is estimated by multiplying the number of data packets with the communication delay.

The HW and SW cost estimates were performed using the estimation process as described in Section III-D.

D. Model Transformation & Cost Estimation

To enable the estimation of HW and SW implementation costs the SystemC-AMS SDF model needs to be decomposed and transformed into separate modules suited for either HW or SW implementation. These modules are similar to the task nodes in the application model. One significant advantage of our methodology is that SystemC and its AMS extension are based on C++, and thus enables maximum reuse of the functional parts of each task, minimising the effort to adapt

the tasks to different implementation flows and architectures. Therefore, the behavioural part of each block of the SDF model is implemented as a C++ class, as shown in Listing 1, which can either be used inside an SDF (Listing 2), a SystemC HW (Listing 3), or a SW module (Listing 4).

```
class behaviour_class_type {
protected:
    internal_var_type var_0;
public:
    behaviour_class_type() { this->init(); }
    void init() { var_0 = var_0_init_value; }

    void run(sca_sdf_in_if<token_in_type>* in,
            sca_sdf_out_if<token_out_type>* out) {
        //perform user-defined behaviour here
    }
};
```

Listing 1. The user-defined internal variables and behaviour of either the SystemC-AMS, the SystemC HW, and the SW model is implemented by a C++ class

```
SCA_SDF_MODULE(sdf_module_name) {
    sca_sdf_in<token_in_type> data_in;
    sca_sdf_out<token_out_type> data_out;

    SCA_CTOR(sdf_module_name) { }

    void attributes() {
        data_in.set_rate(input_rate);
        data_out.set_rate(output_rate);
    }
    void init() { beh_inst.init(); }
    void sig_proc() { beh_inst.run(data_in, data_out); }

    behaviour_class_type beh_inst;
};
```

Listing 2. Outline of a SystemC-AMS SDF module to be estimated for HW and SW implementation. The user-defined internal variables and behaviour of the sig_proc method are implemented by a C++ class (Listing 1).

For the alternative implementation of the tasks in HW or SW we have created code stubs, already containing required control processes and interfaces for communication and data transfer. The stubs can be adapted to the specific task very easily by setting the class type of the behaviour object and by adapting the calls to the object's interface methods if necessary. For HW, an SC_MODULE container is used containing the required FSL interfaces and methods and a control process fetching input data via FSL, performing the required computation and writing the resulting data back via FSL. To obtain cost estimates for the HW implementation of a task, we use our SystemC/C++ synthesis tool [15] translating the transformed SystemC model to synthesisable RT-level VHDL. The generated VHDL code has been used with the RT-level synthesis tool Xilinx XST to obtain a first estimate of the hardware costs (logic blocks, maximum frequency). In our case, execution times in terms of clock cycles have been determined by profiling an early HW prototype on the FPGA platform, but they could also be estimated by simulating or statically analysing the generated VHDL model.

```
SC_MODULE(module_name) {
    sc_in<bool> clock, reset;
    sdf_fsl_in<token_in_type, input_rate> data_in;
    sdf_fsl_out<token_out_type, output_rate> data_out;

    SC_CTOR(module_name) {
        SC_CTHREAD(proc, clock.pos());
        reset_signal_is(reset, true);
    }
};
```

```

}

protected:
void proc() {
    beh_inst.init();
    wait();
    while(true) {
        if(data_in->ready()) {
            for(int i=0; i<num_cycles; ++i)
                beh_inst.run(&data_in, &data_out);
            data_out->flush();
            wait();
        } else wait();
    }
}

behaviour_class_type beh_inst;
};

```

Listing 3. Outline of a SystemC HW module template that contains the connection to the FSL FIFOs with annotated input and output rates and a clocked thread that waits until the input data is available and calls the user-defined behaviour `num_cycles` times, as obtained from the static scheduling. The `wait` statements define the clock boundaries. After all cycles have been completed it updates the output FIFOs.

For SW, the behaviour class is used inside the `main` function also defining input and output of data. The function is compiled to the embedded target platform and can either be profiled by use of either an emulator or on the platform itself. The outcome is a number of cycles necessary for execution. To enable the comparison of the execution times for HW and SW the estimates are normalised in terms of clock cycles. To enable the usage of bit-true SystemC data types in HW as in SW, we have created a lightweight C++ library providing the same types and semantics as the SystemC data types without the additional overhead of the SystemC simulation kernel, making it usable for embedded SW targets.

```

sdf_fsl_in<token_in_type, input_rate> data_in;
sdf_fsl_out<token_out_type, output_rate> data_out;

int main() {
    behaviour_class_type beh_name;
    while(true) {
        if(data_in->ready()) {
            for(int i=0; i<num_cycles; ++i)
                beh_name.run(&data_in, &data_out);
            data_out->flush();
        }
    }
}

```

Listing 4. Outline of a SW module template. It has the same structure as the HW module template from Listing 3, but does not contain any explicit timing information in terms of clock boundaries.

IV. EXPERIMENTS

In Section III-A we have presented a case study as an illustration of the methodology described throughout Section III. This section describes how the architecture exploration is performed based on giving the derived models and cost estimates as input to the mapping framework. The study is composed of three parts: 1) the characterisation of architecture models, 2) obtaining cost estimates for the tasks of the application for the elements of the architecture, and 3) utilisation of the mapping framework to obtain a binding and schedule. The outcome is a Pareto Chart describing the costs of the architecture models, paired with the execution times of the obtained schedules. Section III-B described how to obtain an acyclic task graph as

TABLE I
ARCHITECTURE CONFIGURATION MODELS: THE TWO BOTTOM ROWS INDICATE THE COST OF THE BASIC ARCHITECTURE ELEMENTS

Description	# μ Blaze Units	# FSLs	HW Area Size [LUT]	Total HW Cost [LUT]
Max. HW Area	1	1	62800	69033
10% HW Area	1	1	6900	13133
5% HW Area	1	1	3450	9683
2% HW Area	1	1	1375	7608
1% HW Area	1	1	650	6883
1 μ Blaze only	1	0	0	5698
Max. HW Area	2	2	56000	68466
10% HW Area	2	2	6900	19366
5% HW Area	2	2	3450	15916
2% HW Area	2	2	1375	13841
1% HW Area	2	2	650	13116
2 μ Blazes only	2	1	0	11931
3 μ Blazes only	3	2	0	18164
μ Blaze architecture template element				5698
FSL architecture template element				535

TABLE II
RESULTS OF COST ESTIMATION. THE EXECUTION TIMES ARE NORMALISED TO 100 MHZ CLOCK CYCLES.

Task	HW Cost [LUT]	HW Exec-Time [cycles]	SW Exec-Time [cycles]
SyncWordGen	1165	547	11533
HECGenerator	931	460	141
Whitener (Header)	631	460	1188
EncoderFEC13	856	530	1512
CRCGenerator	981	581	843
Whitener (Payload)	631	577	10560
EncoderFEC23	1134	1162	1696
Packet Switch	-	-	1888
Sum	6329	4317	29361

input specification to the mapping framework. The application model in Fig. 2b is the basis of the experiments.

The architecture exploration is based on a set of architecture models, which all share the properties described in Section III-C. The architecture is based on the Virtex-5 FPGA with two basic processing element types: MicroBlaze soft-core processor and FPGA logic. The processing elements are interconnected via FSLs. The costs of these basic elements are based on IP core synthesis using the Xilinx EDK and XST logic synthesis tools.

The MicroBlaze SW Processor is running at a clock frequency of 100 MHz, and the set of architecture models are shown in Table I. The HW resource costs are measured in terms of Look-Up Tables (LUTs), which is the basis for comparing the cost of the architecture models, HW area size, and the cost of tasks in HW. The total cost of an architecture model is estimated by summing the costs of its elements.

The FSLs are 32 bit wide, and we assume that it is possible to transmit 32 bit for every two clock cycles between processing elements. Internally, within each processing element (FPGA logic or MicroBlaze processor), we assume communication costs to be included in the cost estimates.

The case study is subject to the cost estimation procedure described in Section III-D, which results in the costs shown in Table II. The packet switch task is placed inside a SW processor, as it is mainly communication oriented.

The cost estimates in Table II are provided as input to the mapping framework together with the application model and

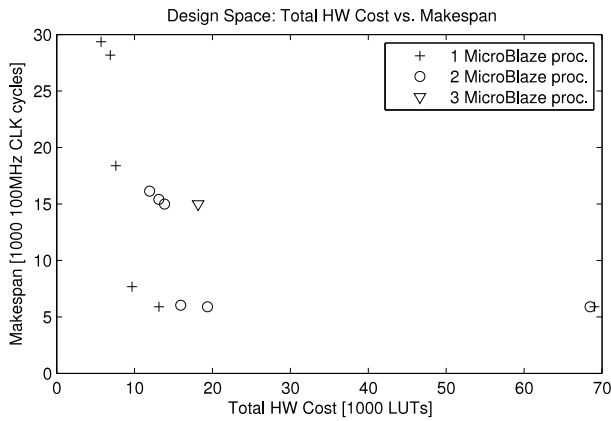


Fig. 3. Pareto Chart showing the mapping result: The x-axis shows the total HW cost in terms of Look-Up Tables (Table I), whereas the y-axis shows the makespan (as determined by the mapping framework) in 100MHz CLK cycles. (Please observe that both axes are in thousands).

the architecture models. The framework is invoked for each of the architecture models, and outputs total execution time of all tasks. The execution time is interpreted as *makespan*, i.e. the span from the start-time of the first task, to the finish time of the last task.

The resulting Pareto Chart for all architecture models is shown in Fig. 3. The graph is composed by pairing the total HW cost (rightmost column in Table I) with the mapping result (makespan) for the corresponding architecture model. Fig. 3 shows that the set of architecture models have different characteristics in terms of area consumption and makespan. Based on the results, it is concluded that the architecture with only 1 MicroBlaze processor is the most feasible as it does not give any speed improvement to increase the number of SW units. The lowest total HW cost (5698 LUTs) is obtained by the pure SW-processor solution with the highest makespan of 29361 cycles. This solution is accepted since the makespan is below the time-slot length of $625 \mu s$. However, the graph shows that by e.g. allocating some HW area (e.g. 3450 LUTs) and thereby increase the area cost by 70% (to 9683 LUTs), the makespan can be reduced to 26% (7676 cycles) of the pure SW implementation.

V. CONCLUSION

In this paper we have presented a methodology for performing SystemC-AMS based exploration for heterogeneous architectures. Main contribution is the derivation of an acyclic task graph based on a synchronous data flow model in SystemC-AMS, and its cost estimation for HW and SW implementations based on a generic C++ class template representation. Each node of the acyclic task graph is subject to cost-estimation based on an architecture template library. The task graph and its cost characterisation for available architecture template elements are input to our mapping framework. Output of the framework is a Pareto Chart allowing evaluation of implementation costs of various architecture configurations.

The methodology has been demonstrated by an IEEE 802.15.1 Bluetooth transmitter case study and has partly been

performed manually for proof of concept. The methodology can easily be automated, reducing the effort of heterogeneous HW/SW architecture exploration from executable SystemC-AMS SDF specifications. The advantage of the full methodology is that the designer does not have to perform the full synthesis and implementation to obtain cost estimates for performing the exploration. Moreover, our approach is modular and IP-centric, allowing the designer to reuse cost estimates, even when new architecture template elements are added.

As future work we will consider pipelining of tasks in the mapping framework and we plan to investigate the impact of SDF granularity on the exploration process. Furthermore, we plan to extend the cost estimation methodology to reconfigurable architectures.

REFERENCES

- [1] A. Popp, Y. L. Moullec, and P. Koch, "Scheduling temporal partitions in a multiprocessing paradigm for reconfigurable architectures," in *NASA/ESA Conference on Adaptive Hardware and Systems*, 2009.
- [2] J. Zhu, I. Sander, and A. Jantsch, "Buffer Minimization of Real-Time Streaming Applications Scheduling on Hybrid CPU/FPGA Architectures," *Proceedings of Design Automation and Test in Europe (DATE'09)*, Nice, France, April 2009.
- [3] E.A. Lee and D.G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Transactions on Computers*, vol. C-36, no. 1, pp. 24-35, January 1987.
- [4] P.K.M. Shuvra, S. Bhattacharyya, and E.A. Lee, "Software Synthesis from Dataflow Graphs," Norwell, MA, USA: Kluwer Academic Press, 1996.
- [5] S.S. Bhattacharyya and W.S. Levine, "Optimization of Signal Processing Software for Control System Implementation," *Proceedings of the 2006 IEEE Conference on Computer Aided Control Systems Design*, Munich, Germany, October 4-6, 2006.
- [6] M. Edwards and P. Green, "The Implementation of Synchronous Dataflow Graphs Using Reconfigurable Hardware," *Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing*, Springer LNCS, vol. 1896/2000, pp. 739-748, January 2000.
- [7] J. Horstmannshoff and H. Meyr, "Efficient building block based RTL code generation from synchronous data flow graphs," *Proceedings of the 37th Annual Design Automation Conference (DAC'00)*, Los Angeles, USA, pp. 552-555, 2000.
- [8] M.C. Williamson, "Synthesis of Parallel Hardware Implementations from Synchronous Dataflow Graph Specifications," PhD thesis, EECS Department, University of California, Berkeley, 1998.
- [9] J.W. Janneck, I.D. Miller, D.B. Parlour, G. Roquier, M. Wipliez, and M. Raulet, "Synthesizing hardware from dataflow programs: an MPEG-4 Simple Profile decoder case study," *Proceedings of IEEE Workshop on Signal Processing Systems*, 2008. (SiPS 2008), Washington, USA, 2008.
- [10] S. Bilavarn, G. Gogniat, J. Philippe, and L. Bossuet, "Design Space Pruning Through Early Estimations of Area/Delay Tradeoffs for FPGA Implementations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, pp. 1950-1968, 2006.
- [11] J. Keinert et al., "SystemCoDesigner: an automatic ESL synthesis approach by design space exploration and behavioral synthesis for streaming applications," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 14, issue 1, January 2009.
- [12] IEEE Comp. Soc., "802.15.1: Wireless medium access control (mac) and physical layer (phy) specifications for wireless personal area networks (wpans)," May 2005.
- [13] G.C. Sih and E.A. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 2, pp. 175-187, February 1993.
- [14] Open SystemC Initiative, "Standard SystemC AMS extensions Language Reference Manual," March 8, 2010.
- [15] FOSSY - Functional Oldenburg System Synthesiser, <http://www.system-synthesis.org>.