Joint Algorithm Developing and System-Level Design: Case Study on Video Encoding

Jiaxing Zhang and Gunar Schirner

Department of Electrical and Computer Engineering
Northeastern University
Boston, MA, 02115
{jxzhang,schirner}@ece.neu.edu

Abstract. System-Level Design Environments (SLDEs) are often utilized for tackling the design complexity of modern embedded systems. SLDEs typically start with a specification capturing core algorithms. Algorithm development itself largely occurs in Algorithm Design Environments (ADE) with little or no hardware concern. Currently, algorithm and system design environments are disjoint; system level specifications are manually implemented which leads to the specification gap.

In this paper, we bridge algorithm and system design environments creating a unified design flow facilitating algorithm and system co-design. It enables algorithm realizations over heterogeneous platforms, while still tuning the algorithm according to platform needs. Our design flow starts with algorithm design in Simulink, out of which a System Level Design Language (SLDL)-based specification is synthesized. This specification then is used for design space exploration across heterogeneous target platforms and abstraction levels, and, after identifying a suitable platform, synthesized to HW/SW implementations. It realizes a unified development cycle across algorithm modeling and system-level design with quick responses to design decisions on algorithm-, specification- and system exploration level. It empowers the designer to combine analysis results across environments, apply cross layer optimizations, which will yield an overall optimized design through rapid design iterations.

We demonstrate the benefits on a MJPEG video encoder case study, showing early computation/communication estimation and rapid prototyping from Simulink models. Results from Virtual Platform performance analysis enable the algorithm designer to improve model structure to better match the heterogeneous platform in an efficient and fast design cycle. Through applying our unified design flow, an improved HW/SW is found yielding 50% performance gain compared to a pure software solution.

1 Introduction

The increasing complexity of Multi-Processor System-On-Chip (MPSoC) designs has become a major challenge. Designers combine components with diverse and distinct architecture characteristics heterogeneously to achieve efficient and flexible platform solutions, which however, dramatically increases design complexity.

G. Schirner et al. (Eds.): IESS 2013, IFIP AICT 403, pp. 26–38, 2013.

In order to tame the complexity, System-Level Design (SLD) has emerged with methodologies, tools and environments for a systematic design at higher levels of abstraction. System-Level Design Environments (SLDE), such as PeaCE [5] and SoC Environment (SCE) [2] operate on an input specification captured in an System-Level Design Language (SLDL), such as SystemC[15] and SpecC[3]. Based on an input specification, SLDEs provide design space exploration through early performance estimation, automated refinement into Transaction Level Models (TLM), and detailed analysis capabilities. Designers can use their synthesis to generate target implementations. A typical SLD environment is shown at the lower half of Fig. 1.

An Algorithm Design Environment (ADE), currently separated from the system-level design, is shown at the top half of Fig. 1. Algorithm development environments mainly concentrate on modeling algorithms. They simplify prototyping of algorithms by providing toolboxes of algorithm components, detailed (graphical) analysis and functional validation tools. Often, at this level of design, little or no hardware knowledge is concerned. ADE examples include TargetLink[13], LabView[7] and Simulink[18]. These ADEs also offer some generation capabilities, however focus on homogeneous solutions and do not offer exploration facilities comparable to SLDEs.

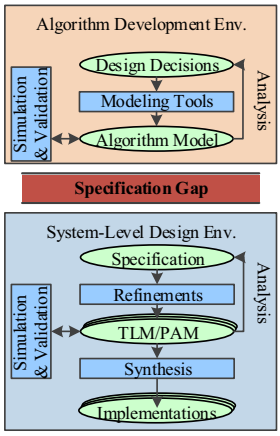

Fig. 1. Specification Gap

However, there is no direct connection between ADEs and SLDEs. Algorithms captured in environments like Simulink are abundantly available through various toolboxes (e.g. computer visions, signal processing), but they need to be converted manually into an SLDL specification for system designed exploration. The manual conversion is time-consuming, tedious and error-prone, which defines the *Specification Gap* (Fig. 1, middle). The specification gap lowers design productivity and hinders efficient co-design of algorithm and architecture.

In this paper, we present a unified design flow that integrates both algorithm and system design environments by bridging the specification gap through *Algo2Spec*. In this paper, we focus on Simulink as an ADE, which offers modeling, simulating, debugging and analyzing multi-domain dynamic systems. We propose *Algo2Spec* to close the specification gap by synthesizing Simulink models into SLDL specifications. This enables rapid heterogeneous design space exploration, e.g. through automatically generated VPs at varying abstraction levels. Simultaneously, by extending an existing system design flow to the algorithm level, algorithm designers in Simulink are provided with rapid, dynamic and accurate feedback from heterogeneous VP analysis. Combining the environments creates a seamless unified design flow reaching from Simulink models, via specification, and VPs to target implementations. The unified flow simplifies model analysis and instant propagation up in the design flow, empowering designers to make strategic decisions for globally optimized designs.

We demonstrate the gained flexibility and efficiency of our unified flow using a video encoder design. We show multiple iterations of design decisions to highlight feedback across the environments. Within three design iterations, models with varying granularity are examined in heterogeneous platform explorations. Through applying the unified flow, a software/hardware co-design solution was identified that yields 50% performance gain compare to pure a software solution.

The remainder of this paper is structured as following: Section 2 introduces relevant related work. Section 3 overviews the design flow. Section 4 shows the cross-layer decisions. Section 5 demonstrates the flow benefits on the video encoder case study. Finally, section 6 concludes the paper.

2 Related Work

Significant research effort has been invested into system-level design usually starting from functional specifications. Conversely, the challenge of how to obtain the specification has not gained much attention and consequently specifications are mostly written manually.

Using UML as part of SoC design methodologies has been intensively studied [8,12,20,17]. Several language conversions to translate UML models into SystemC are proposed [20], [17]. Furthermore, a framework for co-simulation between MATLAB and UML under a co-design flow with application specific UML customizations is provided in [14]. However, these approaches mostly focus on structural conversion and behavior translation (with exception of state machines) is less explored.

A top-down refinement flow from Simulink to SystemC TLM in presented in [6,16]. Refinement and component exploration in their work happens in Simulink, requiring direct annotation, rewriting and modifications of Simulink models. This can be seen as contrary to abstraction separation as the top level of abstraction should not involve hardware concerns [9]. Moreover, no system exploration tools or profiling facilities are instantly applied in their approach to SystemC models. On the implementation side, the Simulink conversion is realized as external tools (ie. yacc/lex), which cannot be integrated to Simulink directly while increasing the overall design tool complexity.

Meanwhile, a re-coding approach [1] is proposed to transform flat C code into a structured SLDL specification. The effort similarly aims to close the specification gap, but directly targets C as an input language. Our approach, on the other hand, starts from a higher abstraction by using Simulink as input models.

Also some generation facilities are present within Simulink. The Simulink Embedded Coder (SEC) [10] directly generates target-specific embedded C code, however does not enable heterogeneous exploration. A co-simulation solution [11] is introduced between Simulink and generated SystemC TLM, however with the focus of being used as testbench rather than the design itself.

3 Unified Algorithm-System Design Flow

Our unified design flow combining both algorithm- and system-level design is shown in Fig. 2. The figure jointly represents three essential aspects (a) the design flow on the left, (b) model examples on the right and (c) decisions at varying levels. *Algorithm Design Environment* (ADE), *Specification Generation Tool* (SGT), namely our proposed *Algo2Spec*, as well as *System-Level Design Environment* (SLDE) compose a top-down design methodology with SGT connecting the other two major design environments. For the work in this paper, we select Simulink [18] as an ADE and the System-on-Chip Environment (SCE) [2] as an SLDE. The latter uses SpecC [3] as an SLDL. The principles and concepts, however, apply to SystemC [15] equally.

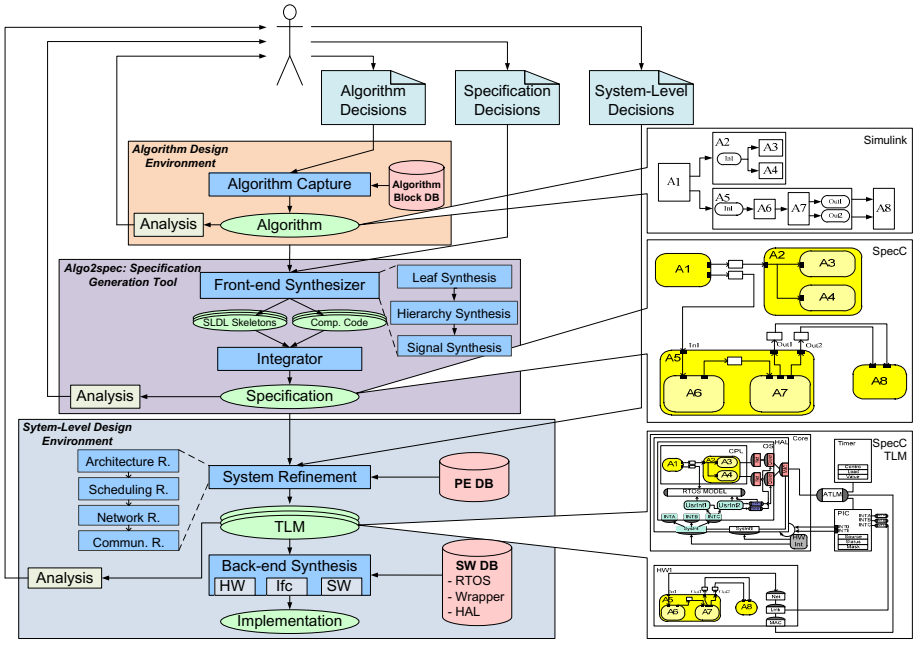

Fig. 2. Unified algorithm-system design flow with feedback/decision loop

The unified design flow, as depicted in Fig. 2, starts with designer capturing in ADE the algorithm with desired functionalities and structure. Available algorithms from the algorithm block database simplify and accelerate the development. The designer tunes and refines the algorithms in ADE as a cycle of modeling, simulation, debugging, validation and evaluation. The result of ADE is the captured algorithm in Simulink as for example shown on top right Fig. 2. The example consists of the top-level blocks A1, A2, A5, and A8, were A2 and A5 execute parallel and contain further sub-blocks.

After finishing the initial algorithm specification, the designer transitions to Specification Generation Tool (SGT), namely *Algo2Spec*, shown in the middle

of the Fig. 2. *Algo2Spec* synthesizes the Simulink model into a specification in SpecC SLDL, following the designer's specification design decision. Different SLDL specification can be generated based on decisions of granularity, scheduling and optimization configurations. An SLDL specification example is shown next to the SGT, were leaf blocks are converted to leaf behaviors and hierarchical blocks are represented through hierarchical behaviors preserving the original hierarchy. Simulink signals are captured as variable in this example.

The generated specification is then the basis for detailed design space exploration in the SLDE. The designer can explore platform decisions: Processing Elements (PE) allocation, behavior to PE mapping, scheduling and communication refinement. In SLDE, depicted in the bottom of Fig. 2, *System Refinement* then realizes these decisions in form of Transaction-level Models (TLM). The TLM of the example, illustrated below the specification, is generated based on a mapping decision of A1, A2 on CPU while A5 and A8 on a custom hardware: HW1. After optimizing algorithms and finalizing the platform decisions, the *Back-end Synthesis* can synthesize the overall design down to the selected hardware/software implementation.

3.1 *Algo2Spec*: Specification Generation Tool

To bridge the gap between ADE and SLDE, we propose *Algo2Spec*. *Algo2Spec* takes the Similink algorithm model as an input and synthesizes it to a SpecC SLDL specification representing the algorithm's functionality and original hierarchy. We construct *Algo2Spec* as a multi-stage process of the front-end synthesizer and the integrator. In the first step, the front-end synthesizer in turn conducts leaf synthesis (generating computation code for each leaf component), hierarchy synthesis (producing the overall SLDL skeleton) and signal synthesis (connecting behaviors). Afterwards, the integrator combines the output of front-end synthesizer to produce a specification. The following paragraphs outline each step.

Leaf Synthesis. As a first step in the generation process, the front-end synthesizer starts with the leaf synthesis. It works on the smallest unit in the model, which is an atomic block in Simulink (e.g computation, algebraic or logical) as the blocks A1, A3, A4, and A6-A8 in the example. Each leaf block in Simulink will be represented as a SLDL behavior, which is an atomic unit capturing computational load. As SpecC SLDL is an extension over ANSI-C, C-code is needed for each behavior. To generate this, leaf synthesis iterates through the Simulink model and invokes Simulink Embedded Coder (SEC) as the back-end C code generator for each leaf. The result is a set of C files for each Simulink block expressing the computation.

Hierarchy Synthesis. The hierarchy synthesis is responsible for generating an overall SLDL skeleton for the Simulink model, capturing its structural and behavioral hierarchy. For this it creates an empty behavior for each identified leaf, and also creates hierarchical behaviors to represent the Simulink model structure. In our example A2 and A5 are hierarchical behaviors containing the child behaviors A3, A4 and A6, A7, respectively. During this stage, syntactical correctness and model functional validity are both enforced. Single rate models

appear as a sequential composition. Multiple single-rate islands are represented as parallel behaviors communicating through channels. However, hierarchy flattening is necessary for multi-rate models to partition blocks running at different rates to different rate control behaviors in the SLDL specification. Scheduling is an important part of hierarchy synthesis, please see Sec. 4.

Signal Synthesis. Simulink uses signals for communicating between blocks by defining that each signal has to have at one simulation time interval the same value at every of its point. Signal synthesis recreates the same connectivity in SLDL communication primitives ensuring communication across behaviors. It mainly generates variables mapped through ports and routed through hierarchical behavior to enable communication. Signal synthesis becomes critical in multi-rate models, were buffers are needed. Buffers are generally represented as queues in the SLDL.

Integrator. After front-end synthesis, the integrator combines the computation code generated by leaf synthesis and the SLDL skeleton containing hierarchy, communication and scheduling generated by hierarchy/signal synthesis to output the final specification. The integrator localizes global variables and creates clean and well-defined communication interface adhering to the communication structure in original Simulink models. In addition, the integrator also inserts all initialization and termination routines to maintain proper model execution.

In result, all computation related procedures are merged into SLDL behaviors by the integrator. A specification that captures the model hierarchy, execution semantics, signal properties and valid results is generated. The SLDL specification serves as the fundamental element for detailed design space exploration. The SGT eliminates the need for manually writing the specification, thus avoid coding errors and significantly shortens the time to exploration.

Scope. As Simulink offers a wide range of modeling domains (e.g. continues time and discrete) and in order to achieve a feasible solution. The subset supported by *Algo2Spec* has to be limited. Most importantly, continues time models are out of scope for *Algo2Spec*. Restricting the scope of considered Simulink semantics for synthesis purpose has been shown effective in earlier approaches [19], which demonstrated tremendous benefits while targeting only a safe subset. For this paper, we target the following components: blocks under discrete and fixed step solver and SEC; special blocks in toolboxes (partially supported depending on configurations); multi-rate systems with limited rate-transition schemes.

4 Cross-Layer Decisions

With the tight integration, the unified design flow realizes a feedback/decision loop (see Fig. 2) where feedback by the analysis tools in ADE, SGT and SLDE respectively is propagated to the designer on the left-hand side. In result, on the right side, the designer enters new decisions into the flow for optimization and system-level exploration to identify a suitable algorithm / platform combination. Assisted by the multi-level analysis feedback, the designer can identify and eliminate bottlenecks at an appropriate level. The next paragraphs outline the decisions at each level as well as overall decision opportunities.

Algorithm Decisions. The decisions at the first level mostly stem from within the ADE. Through its simulation analysis, the designer improves the algorithm for optimizing functional performance. Examples include algorithm exploration to find the most suitable algorithm for a given task, the configuration or tuning of an algorithm. Functional performance in this context could refer to the detection accuracy of a vision algorithm.

However, algorithm composition also affects the later design. As a structural example, an S-Function block cannot be further decomposed into smaller blocks. Thus, it hinders finer granularity explorations. Higher dimensions and width of signals used between blocks may cause higher traffic if these blocks are mapped to different PEs. An additional degree of freedom is the exposed amount and type of parallelism. Different types of PEs (such as processor, GPU, custom hardware) efficiently operate on different types of parallelism (task-level, data-level, fine-grained, respectively). If the designer already has a PE type in mind, the algorithms with an appropriate parallelism can be chosen. Overall, algorithm decisions not only alter model functionality, but also impact the system-level explorability of the overall design and preferred PE-type mapping.

Specification Decisions. These decisions affect the generation of the specification in terms of granularity of SLDL blocks, scheduling of blocks and exposed parallelism. The granularity decision determines up to which level the Simulink's structure is exposed in SLDL, and conversely what is treated as a leaf component. At one extreme, the finest granularity exposes each atomic Simulink block as leaf in SLDL. This hinders Simulink to perform block fusion as all blocks are generated separately. Further, it may expose overly simplistic blocks, such as single stage combinatorial logic, to the SLDL. Overall, this may cause unnecessary communication overhead in the SLDL specification without offering impactful mapping alternatives. At the other extreme, the coarsest granularity treats the Simulink top model as one "big" leaf. It gives all cross-block optimization potential to Simulink, however, causes a loss of hierarchy which in turn removes the possibility of heterogeneous exploration. *Algo2Spec* therefore aims at at a granularity with a threshold balancing block fusion potential and exploration flexibility. In the current version, *Algo2Spec* relies on the designer to define the granularity (see Fig. 2). A heuristic and automatic decision making for deciding granularity is part of future work.

Simulink and the SLDL differ in execution semantics, which offers opportunities for scheduling exploration. One policy: *faithful scheduling* emulates the simulation semantics of Simulink. All Simulink blocks are statically scheduled in a sorted dependency list and executed sequentially. In result, all SLDL behaviors will be sequentially executed in the same order. A second scheduling policy: *pipelined scheduling* can be observed for example in the Simulink HDL Coder generated code. Blocks are executed in a pipeline fashion offering temporal parallelism. If selected, Simulink blocks are then synthesized into a pipelined SLDL model enabling parallelism exploring. For brevity and simplicity, this paper focuses on *faithful scheduling*.

It has to be noted that granularity and scheduling also affect the exposed parallelism. The desired parallelism depends on the targeted PE class as for example mapping to custom hardware component can benefit from much finer grained parallelism than a processor mapping.

System-Level Decisions. Once the specification is defined, a myriad of system-level decisions are required spanning architecture decisions (PE allocation and behavior mapping), scheduling decisions of behaviors within each PE (static, dynamic-priority, dynamic-FCFS), network decisions (interconnect and communication element allocation), and subsequent communication decisions (selecting interconnect specific communication and synchronization primitives). These decisions are for example discussed in [2]. The SLDE realizes these decisions by model refinement. It generates Transaction-Level Models (TLM) at varying levels of abstraction to implement these decisions. The generated TLMs are then used for performance/power/cost analysis driving new system-level design decisions.

Cross Layer Decisions. In addition to the decisions at each level, cross-layer decisions are needed. For example, we have observed in the past the need to change parallelism granularity and type depending on mapping decisions. Hence in result of system exploration, the algorithm definition needs to be changed. We also have experienced the necessity for platform specific algorithm tuning such as converting from floating to fixed point computation, or reducing memory bandwidth of a vision algorithm through parameter compression. Again, with automating the path from algorithm to specification and instant available performance feedback, the designer can easily try out various decision combinations that will yield better overall designs.

5 Case Study

We demonstrate the flexibility of the unified design flow by exploring a Motion JPEG (M-JPEG) video encoder. We have chosen this application for it is a real-life application that is not too complicated. M-JPEG is a video format that defines each frame of the video feed is compressed as a JPEG image separately, which is like MPEG [4] but without inter-frame predictive coding. It is primarily used in video-capturing equipment, such as digital cameras and webcams. We start with a Simulink model out of the vision toolbox. The model contains 294 blocks, 28 S-functions, 35 subsystems, 86 inports and 69 outputs. The model contains blocks with up to 8 levels in depth (which we will later simply refer to as depth). For our experiments, the design encodes a 910-frame QVGA (320x240) video stream.

In this case study, the designer performs three exploration steps of granularity identification (algorithm decisions/specification decisions), early estimations and platform explorations (system-level decisions). Based on simulation feedback, a new design iteration starts with new higher-level design decisions. Overall, we show three design iterations.

All iterations draw from the same database of processing element (PE) types. For simplicity, we restrict the exploration to one CPU (ARM7TDMI at 100MHz)

that is assisted by up to two custom hardware components (HW1, HW2 at 100MHz). All PEs are connected to one processor bus of type AMBA AHB.

Initial Design Iteration. In the first iteration, the designer selects a coarse granularity which only exposes a minimum decomposition. This may potentially facilitate optimization of the generated computation code, such as block fusion by Simulink Embedded Coder (SEC) and reduce the overall traffic due to fewer components. With the selected granularity, *Algo2Spec* generates the coarse-grained specification as shown in Fig. 3. It only contains three leaf behaviors, `PreProc`, `BlkProc` and `MatCon` in behavior `Encoder_DUT` (Design Under Test) together with `Stimulus` and `Monitor` consisting the testbench.

After specification generation, the SLDE tools for early estimation can be used to identify allocation and mapping candidates. Fig. 5 shows the result of early estimations of both computation and communication demands on each generated leaf behavior in the specification. Computation expresses the number of C-level operations executed in each leaf behavior. Communication demand shows number of bytes are transferred overall in a leaf behavior.

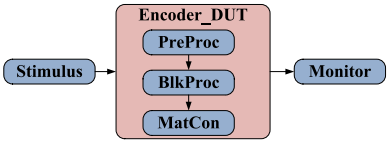

Fig. 3. Initial Specification

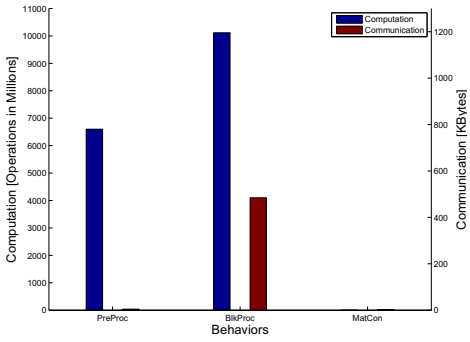

Fig. 4. Initial Early Estimation

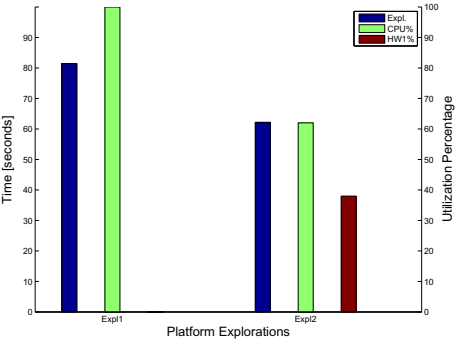

Fig. 5. Initial Exploration Results

Due to the coarse granularity, not many blocks are available for PE mapping as they are all merged together into a few "super" blocks even though it has a fairly light overall traffic. `BlkProc` is the most computational and communicational expensive block. Thus the design is most suitable for SW-only mapping and one with a custom HW component for the most computation intense block `BlkProc`. Tabl. 1 summarize these explorations.

Fig. 5 plots the results of the platform explorations in terms of target execution time (in secs) and PE utilizations (in percentage). The leftmost bar indicates target execution time, while the right bars

Table 1. Initial Explorations

	Expl1	Expl2
CPU	All	All others
HW1		BlkProc

show PE utilizations. Expl1, the SW-only solution takes nearly 80s where Expl2, is faster with 60s. With dedicated HW for executing the most computational intensive behavior, both CPU and HW1 are load-balanced and the co-design solution offers some performance increase. However, the speedup is not too significant due to the coarse-grained model and limited parallelism. Hence, we start a new iteration for a finer-grained specification to increase mapping options.

Intermediate Design Iteration. A finer granularity with depth 3 (hierarchy levels) is chosen to generate a new specification. Fig. 6 illustrates the simplified fine-grained specification. The fine-grained specification decomposes all three levels of `PreProc` to 10 sub-behaviors; `BlkProc` to 4 sub-behaviors; `MatCon` to 1 sub-behavior (see Fig. 6). Two task-level parallelism within, `Spl_Par` and `ImgP_Par` are discovered while decomposing `Pre_proc` and scheduled in parallel in the generated SLDL specification.

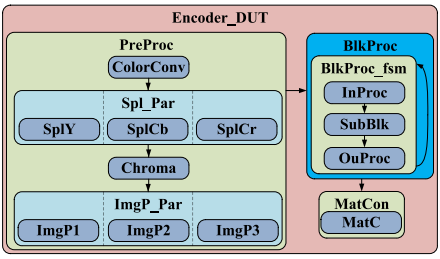

Fig. 6. Intermediate Specification

The estimation results in Fig. 7 omit two behaviors in `ImgP_Par` due to their low computation and communication demands. Also, it is inefficient to explore the task-level parallelism in `Spl_Par`, and `ImgP_Par` as these behaviors have low computation. Splitting `BlkProc` potentially introduces more traffic as `InProc` and `OuProc` have more data exchange with `SubBlk` within the original block. Consequently, Tabl. 2 shows selected explorations: Expl3, mapping `SubBlk` to HW1 and Expl4, mapping `SubBlk` and `Chroma` to HW1.

The results in Fig. 8 (with Expl1 for reference) show that Expl3 yields a similar performance gain as Expl2 in the previous iteration. Unexpectedly, Expl4 downgrades performance by incurring more traffic.

Table 2. Intermediate Explorations

	Expl3	Expl4
CPU	All others	All others
HW1	SubBlk	SubBlk, Chroma

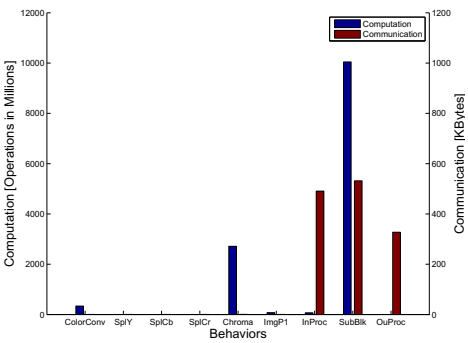

Fig. 7. Interm. Early Estimation

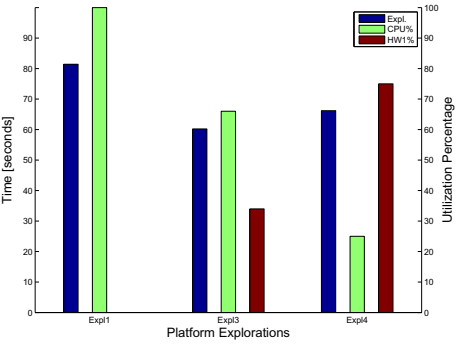

Fig. 8. Interm. Exploration Results

Even though this specification offers more mapping flexibility, it introduces too many oversimplified behaviors such as `SplY`, `SplCb`, `SplCr` and `ImgP1/2/3`. Hierarchy depth is imbalanced: `BlkProc` in Fig. 3 has depth of 8 while on the same level, `PreProc` and `MatCon` only have depths of 3 and 2. It is undesired to decompose computationally non-intense blocks unnecessarily while the computation heavy `BlkProc` is insufficiently split. Therefore, in the next iteration a custom granularity selectively splits `BlkProc` to an even finer granularity to expose potential parallelism without affecting other blocks with overheads.

Final Design Iteration. Fig. 9 shows the custom generated specification which has `BlkProc` with depths of 5, while `PreProc` and `MatCon` remains at level 1. `BlkProc` is decomposed now through 5 levels to `Mot`, along with three parallel behaviors: `Tran1`, `Tran2` and `Tran2`.

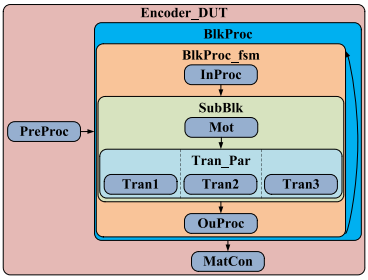

Fig. 9. Final Specification

Fig. 10 presents early estimation for behaviors with meaningful computation or communication. `Tran1` has significant computation and low communication, making it a candidate for concurrent execution. This iteration now uses two custom hardware: HW1 and HW2. Tabl. 3 show the explorations. All four explorations have `Tran1` mapped on a separated HW1, running concurrently with `Tran2`. The designer explores with `Tran1`, `Tran2` and `Tran3` to run in parallel by adding HW2 (Expl6, Expl7 and Expl8).

Fig. 11 shows that, mapping `Tran2` does not boost the performance meaningfully as its low computation to communication ratio. Furthermore, Expl8, even though HW2 is load-balance, the overall performance still does not increase much due to additional traffic.

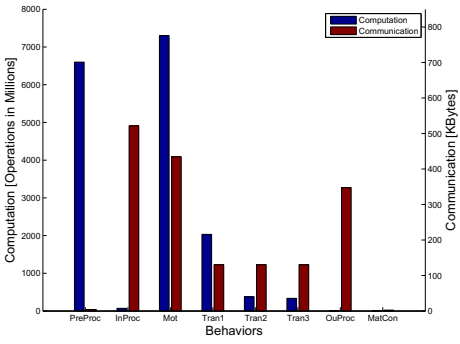

Fig. 10. Final Early Estimation

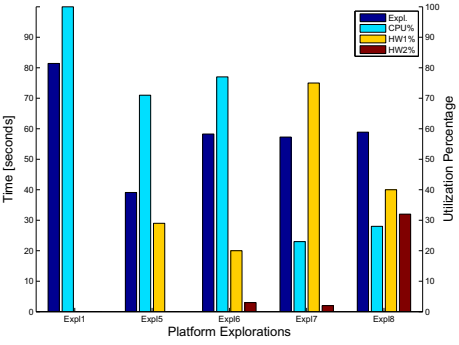

Fig. 11. Final Exploration Results

Expl5 achieves best performance splitting the two computational heavy blocks `PreProc` and `Mot` onto CPU and HW1 without much traffic. Meanwhile, it keeps `Tran1` running in parallel with `Tran2` and `Tran3`. Subsequently, Expl5 is selected for final implementation, which is generated by the back-end synthesis. Due to limitation of space, the details of the implementation are not further examined.

Table 3. Final Explorations

	Expl5	Expl6	Expl7	Expl8
CPU	All oth.	All oth.	All oth.	All oth.
HW1	PreProc Tran1 Mot	PreProc Tran1	PreProc Tran1 InProc	PreProc Tran1
HW2		Tran2	Tran2	Mot, Tran2

6 Conclusion

In this paper, we presented a unified algorithm and system design flow. We introduced *Algo2Spec* which closes the specification gap by generating SLDL specifications out of Simulink algorithm models. Automating specification generation eliminates the error tedious manual implementation. Moreover, it enables algorithm designers to explore the platform implications of the chosen algorithms, and empowers a cross-layer decision process.

We demonstrated the unified flow using an MJPEG video encoder example and highlighted cross-layer design iterations. An improved HW/SW solution was identified yielding 50% performance gain compared to a pure software solution. All explorations are conducted in a few hours which yield a significant improvement on productivity and efficiency.

Acknowledgment. The work presented in this paper is in part supported by the National Science Foundation under Grant No. 1136027.

References

1. Chandraiah, P., Domer, R.: Code and data structure partitioning for parallel and flexible MPSoC specification using designer-controlled recoding. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 27(6), 1078–1090 (2008)
2. Dömer, R., Gerstlauer, A., Peng, J., Shin, D., Cai, L., Yu, H., Abdi, S., Gajski, D.D.: System-on-Chip Environment: A SpecC-based Framework for Heterogeneous MPSoC Design 2008(647953),13 (2008)
3. Gajski, D.D., Zhu, J., Dömer, R., Gerstlauer, A., Zhao, S.: SpecC: Specification Language and Design Methodology. Kluwer Academic Publishers (2000)
4. Gall, D.L.: MPEG: a video compression standard for multimedia applications. Communications of the ACM 34, 46–58 (1991)
5. Ha, S., Kim, S., Lee, C., Yi, Y., Kwon, S., Pyo Joo, Y.: PeaCE: A hardware-software codesign environment for multimedia embedded systems. ACM Transactions on Design Automation of Electronic Systems 12 (2007)
6. Jerraya, A.A., Bouchhima, A., Pétrot, F.: Programming models and HW-SW interfaces abstraction for multi-processor SoC. In: Design Automation Conference, pp. 280–285 (2006)

7. Johnson, G.: LabVIEW Graphical Programming: Practical Applications in Instrumentation and Control, 2nd edn. McGraw-Hill School Education Group (1997)
8. Martin, G., Mueller, W.: UML for SOC Design. Springer, Dordrecht (2005)
9. Marwedel, P.: Embedded Systems Design. Kluwer Academic Publishers (2003)
10. The Mathworks, Inc. Simulink Embedded Coder Reference R2011b (2011)
11. The Mathworks, Inc. Untimed SystemC/TLM Simulation (2012b)
12. Mellor, S.J., Balcer, M.J.: Executable UML: A Foundation for Model-Driven Architecture, 1st edn. Addison-Wesley Professional (May 2002)
13. MISRA. Misra ac tl: Modelling style guidelines for the application of targetlink in the context of automatic code generation (2007)
14. Mueller, W., Rosti, A., Bocchio, S., Riccobene, E., Scandurra, P., Dehaene, W., Vanderperren, Y.: UML for ESL design: basic principles, tools, and applications. In: International Conference on Computer Aided Design, pp. 73–80 (2006)
15. Open SystemC Initiative. Functional Specification for SystemC 2.0 (2000)
16. Popovici, K.M.: Multilevel Programming Envrionment for Heterogeneous MPSoC Architectures. PhD thesis, Grenoble Institute of Technology (2008)
17. Riccobene, E., Scandurra, P., Rosti, A., Bocchio, S.: A SoC design methodology involving a UML 2.0 profile for SystemC. In: Design, Automation, and Test in Europe, pp. 704–709 (2005)
18. The Mathworks Inc. MATLAB and Simulink (1993)
19. Tripakis, S., Sofronis, C., Caspi, P., Curic, A.: Translating discrete-time simulink to lustre. ACM Trans. Embed. Comput. Syst. 4(4), 779–818 (2005)
20. Vanderperren, Y., Dehaene, W.: From UML/SysML to MATLAB/Simulink: current state and future perspectives. In: Design, Automation, and Test in Europe (2006)