Linking Codesign and Reuse in Embedded Systems Design

M. Meerwein, C. Baumgartner
Robert Bosch Gmbh
Automotive Equipment Division 8
72703 Reutlingen, Germany
Matthias.Meerwein@de.bosch.com

W. Glauert
Institute for Computer Aided Circuit Design
University of Erlangen-Nuremberg
91058 Erlangen, Germany
whg@lrs.e-technik.uni-erlangen.de

Abstract

This paper presents a complete codesign environment for embedded systems which combines automatic partitioning with reuse from a module database. Special emphasis has been put on satisfying the requirements of industrial design practice and on the technical and economic constraints associated with automotive control applications. The object-oriented database architecture allows efficient management of a large number of modules. Experimental results from a real-world example demonstrate the viability and advantages of the presented methodology.

1. Introduction

Automotive applications are highly cost-critical and yet they have to satisfy stringent real-time constraints. An example are small, detached systems for sensor signal processing or actuator control taking computational load off the main control unit. While they are essentially of control-dominated nature, they also comprise digital signal processing to a small extent. Maximizing the software-implemented share of the system preserves a high degree of flexibility later in the product's life cycle. On the other hand, a small number of time-critical functional units (often signal processing) would demand a powerful and expensive microprocessor in a pure software solution. This is solved by adding application specific hardware modules (coprocessors) making a less powerful standard microcontroller sufficient to implement the remaining functionality.

The need for continuous design time reduction has led to a new design paradigm: Reuse of existing, validated modules ("Intellectual Property" or IP) from previous projects or third party vendors. While IP reuse is not a panacea for all sort of engineering problems, experience has shown that it can be applied successfully if certain industrial ancillary conditions are met.

In automotive applications, serious effort is spent on software and hardware optimization in order to minimize product cost – an investment that should be exploited at the best by reuse. Therefore, our goal is to control partitioning by the availability of IP. We believe that combining automated HW/SW partitioning and code reuse bears a large potential for a further cut in product cost and design time.

2. Related Work

Current codesign approaches like [1], [2] and [3] focus on interactive partitioning, performance estimation, cosimulation, communication synthesis and code generation. Automated partitioning has been under intense research (e.g. [2], [3], [5], [7] and [8]). However, manual partitioning is still predominant in industry with its quality depending largely on the skills and experience of the designer.

While code reuse is traditionally widespread in software engineering, the viability of this technique in the field of hardware design has also been demonstrated. Use of configurable modules and associated design flows[9] together with version and configuration management [10] allow to implement a new design as a composition of library components with only minor adaptation effort. Integration with existing simulation and synthesis tools into a heterogeneous design environment enables the application of HDL code reuse in industrial-scale projects.

In [11] a database-oriented reuse management system is presented. Focussing on the aspects "design for reuse", "IP repository management" and "design by reuse", this work addresses many important topics of industrial application such as usability, protection against unauthorized access and design flow integration. Proposing a taxonomy-based classification system, it allows to efficiently locate the database-stored modules. Furthermore, that paper provides evidence that design reuse pays off economically.

3. Linking Codesign and Reuse

In our context, the method of reusing existing modules promises advantages over code synthesis:

• The real-time behavior and resource usage of existing modules is well characterized by extensive profiling done in the previous testing process. This allows for a substantially more exact prediction of the real-time reactivity of the sys-

This work was supported in part by the MEDEA project T508.

tem during the partitioning process than estimation performed before or during code synthesis. As our target applications are usually running under strict real-time constraints, this is very important.

• Second, the software synthesis of common codesign systems generates C or C++ code. In the case of 8-bit processors with low register count however, assembly language programming has proved to be advantageous over compiling C code. Due to cost constraints, these CPUs constitute a significant share of industrial and automotive embedded systems. Lacking support for assembly language generation in most codesign frameworks, reusing hand-written, heavily optimized assembler modules is currently the only way to overcome this problem short of writing code from scratch. The high coding and optimization effort for these modules makes their reuse especially worthwhile.

We have chosen POLIS as a base for our work because it supports the codesign flow from specification input to synthesized hardware and software output. Targeted at control-dominated real-time systems, it suits the domain of automotive control systems very well.

The goal of our work is to integrate reuse-driven automated partitioning into the POLIS ([1]) codesign methodology as outlined in figure 1. The application specification is entered using Esterel[4] and passed – through POLIS – to the partitioning system where it is transformed into an internal representation. It is enhanced with user-supplied timing and resource usage constraints.

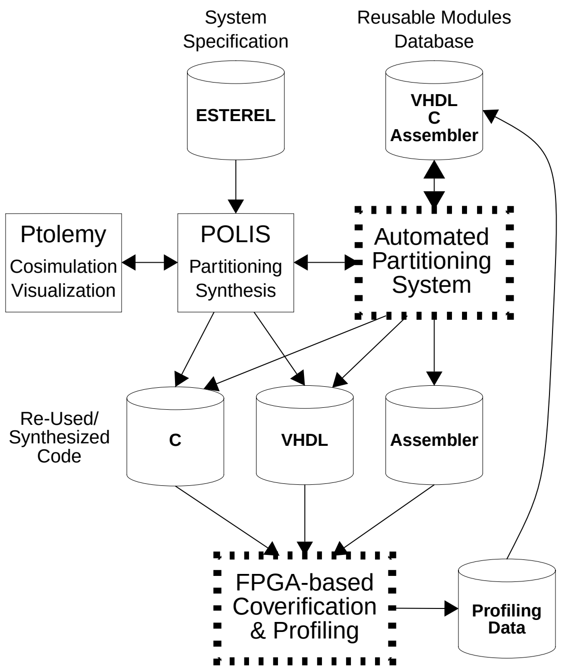

Figure 1: Extending the POLIS system

4. The Module Database

Our module database is running on a commercial relational DBMS (Oracle V7). It is derived from an in-house reuse management system [11] that has proven its effectiveness and stability in everyday design business. We have added the capability to process essential partitioning-relevant information by extending the existing storage architecture. The entity-relationship diagram of the new data model is shown in figure 2. The # sign indicates primary keys, * stands for non-null attribute while + designates optional attributes.

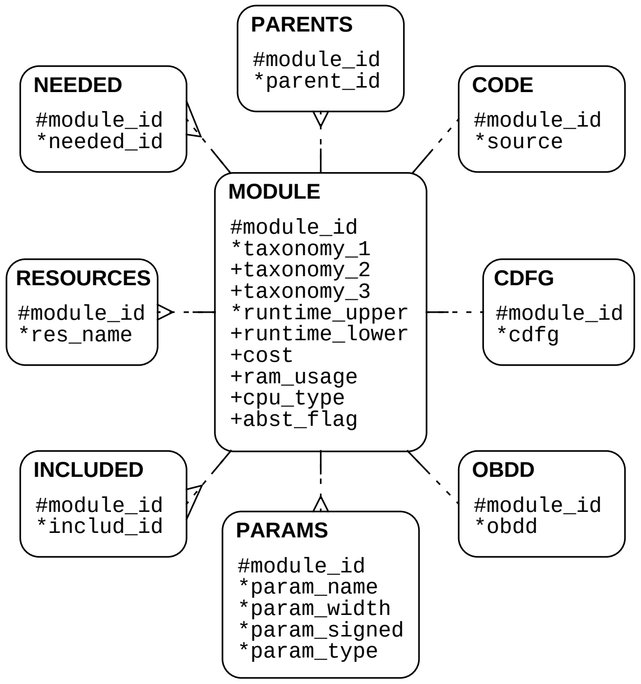

Figure 2: Data Model of the Module Database

Every module is categorized within the reuse management system by a three-level hierarchical taxonomy classification system (table 1). This ordinal information is stored together with the characterization data – real-time behavior, resource requirements (timer, ports etc.) and cost metric – in the MODULE table of the database. The timing characterization data is obtained by the FPGA-based coverification briefly outlined in chapter 5. It is capable of characterizing large numbers of software modules in batch mode. The PARAMS table contains the names and data types of parameters (input/output variables, constants) associated with the module. Inter-module dependencies (a module requiring another one to work, a module functionally including another one) are defined in the INCLUDED and NEEDED tables.

Level	Content	Example
1	Functional	Add
2	Implementation	Hardware
3	Algorithm	Ripple

Table 1: Hierarchical taxonomy classification

The object-orientation of the data model (property inheritance from other modules or abstract classes) makes it specifically well-suited for efficient storage of a large number of modules differing only in minor details. Abstract classes do not refer to instantiable code but define common properties for a whole class of modules. Since we want to provide multiple inheritance (m:n-relation), the inheritance information is stored in the intermediate table PARENTS.

As an example, we look at the transformation of rectangular coordinates (Re,Im) into their polar representation (φ,Mag). First, an abstract class (Rect2Pol_meta, as shown in table 2) defines characteristics such as the functional and algorithm classification (`taxonomy_1`, `taxonomy_3`) or the input and output variables (`param_...`) that are common for all implementations of this operation. It is identified by the `abst_flag` being TRUE.

Attribute	Value			
`module_id`	Rect2Pol_meta			
`taxonomy_1`	Rect2Pol			
`taxonomy_3`	CORDIC			
`param_name`	Re	Im	Phi	Mag
`param_width`	16	16	16	16
`param_signed`	TRUE	TRUE	FALSE	FALSE
`param_type`	Input	Input	Output	Output
`cpu_type`	HC08			
`abst_flag`	TRUE			

Table 2: Abstract class for coordinate transformation

The records for the individual implementations listed in table 3 all inherit (`parent_id` = Rect2Pol_meta) the basic properties from the abstract class defined above. Only those attributes like the implementation domain (`taxonomy_2`), the run time bounds and cost parameters that are unique to each implementation are listed in the associated records. Since those modules refer to actual code and may be instantiated, their `abst_flag` is FALSE.

Attribute	Value		
	Pure SW	HWSW (ACP)	HWSW (CCP)
`parent_id`	Rect2Pol_meta	Rect2Pol_meta	Rect2Pol_meta
`taxonomy_2`	SW	HWSW	HWSW
`runtime_upper`	1325	785	52
`runtime_lower`	1305	770	52
`cost (ROM)`	795	522	18
`cost (kgates)`	0	8	5
`ram_usage`	10	6	0
`abst_flag`	FALSE	FALSE	FALSE
`needed_id`	—	ACP16	CCP16

Table 3: Coordinate transformation implementations

The OBDD and CDFG tables provide storage for a machine-readable representations of the module's functional specification. They will be used in future partitioning algorithms: By comparing them against the system specification the user is relieved from the need to manually assign the modules a taxonomy classification.

The database management software we use imposes the limitation of no more than one column of a data type suited to store binary data per table. Hence, the OBDD and CDFG specifications and the module source code must be stored in separate tables even though they have a 1:1 relation with the MODULE table.

5. Automating the Partitioning Process

The goal of our partitioning algorithm is to pre-select solutions meeting the real-time constraints and assign them a cost metric to select the most preferable solution. With product cost and flexibility in mind, we found software-biased timing-driven allocation to be preferable. Thus, as much of the system as possible is implemented in software. Only in cases where timing constraints are violated, blocks are mapped to hardware. However, some other factors are restricting the freedom of the partitioning choice:

• For standardization in an industrial environment, the CPU model is usually predetermined for a whole application family and is not modified to suit the problem.

• Standard microcontroller peripherals or reusable hardware or software modules implemented earlier are preferred over newly synthesized components.

• Coprocessors can provide a speedup in multiple places while consuming chip area only once.

• Time-consuming hardware-software communication is to be minimized.

• User-supplied implementation rules ("must be hardware", "must be software") have to be obeyed to facilitate migration between product generations.

After reading the specification (POLIS-written SHIFT files) the application is clustered into modules. Currently, the clustering granularity is inherently predetermined by the modularity of the initial Esterel code read by POLIS. Next, the modules are functionally categorized by interactively assigning them to a functional class equivalent to level 1 of the three-level hierarchical taxonomy classification system (cf. table 1). Now, the data types of the interface variables – whose literal names are taken from the shift code – are determined.

Subsequently, the partitioning algorithm as depicted in figure 3 is invoked. For each module of the clustered system specification, a database search for a reusable module matching the functional classification (taxonomy level 1) and the auxiliary constraints (implementation domain, CPU type, resources, data types) is performed. If no implementation constraint has been set, software modules are preferred. If an appropriate entry can be located in the database the module is assigned the attribute "instantiate"; otherwise the module is scheduled for code synthesis.

After this has been accomplished for all modules of the system, the inter-module dependencies ("includes", "requires") held in the reuse database are resolved: Additional modules are scheduled for instantiation and unnecessary modules are removed. Now the cost (gate count, memory usage) and real-time characteristics of the partitioned system will be calculated and checked against the user-supplied constraints. If they are satisfied, the partitioning process is completed. If the real-time requirements (deadlines) are not satisfied, modules are iteratively moved into the hardware partition, again preferring instantiation over synthesis. Our current strategy in this place is to start with the module with the greatest runtime. If resource usage constraints are violated (e.g. the software modules require more timers than exist) the user is prompted to resolve the conflict. He may either change the available resources or manually create a workaround (e.g. two modules sharing one timer using a software scheduler) for the problem.

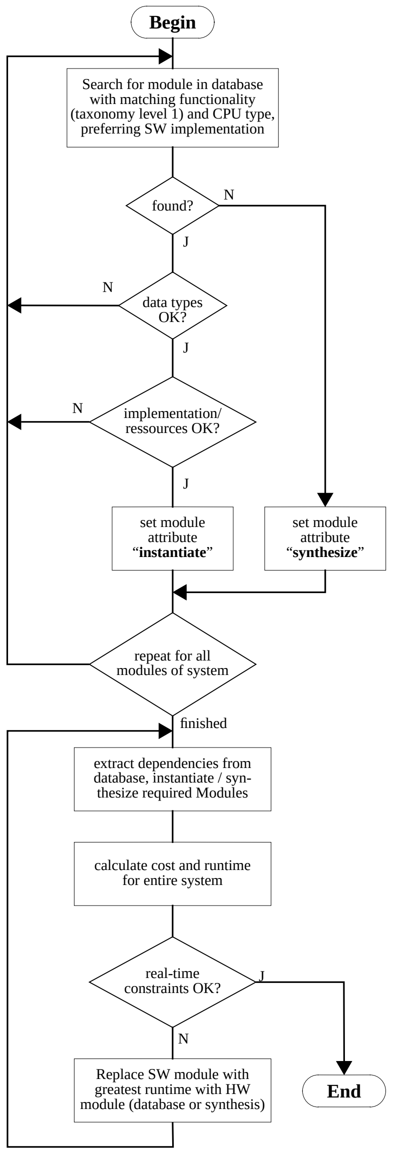
Begin
Search for module in database with matching functionality (taxonomy level 1) and CPU type, preferring SW implementation
found?
N
J
N
data types OK?
J
N
implementation/ ressources OK?
J
set module attribute "**instantiate**"
set module attribute "**synthesize**"
repeat for all modules of system
finished
extract dependencies from database, instantiate / synthesize required Modules
calculate cost and runtime for entire system
real-time constraints OK?
J
N
Replace SW module with greatest runtime with HW module (database or synthesis)
End

Figure 3: The partitioning algorithm

When the partitioning run is finished, the allocation information (HW/SW, synthesis/instantiation) is passed back into POLIS, where a first verification can be performed by Ptolemy-based simulation. Subsequently, in places where existing modules have been scheduled for instantiation, the corresponding VHDL or C code pieces are checked out from the database. Otherwise POLIS synthesizes them. Finally, the partitioning choice is in-system-verified. For this purpose, we use a commercially available in-circuit emulator for the CPU core coupled with FPGA boards to map application specific controller peripherals. Modeling the complete ASIC including its physical interfaces and analog portions, we are able to obtain results that are significantly more precise compared with cosimulation.

6. Results

We evaluated the performance of the presented approach using the magneto-resistive steering wheel angle sensor published in [12] as an example. The algorithm employed essentially consists of three components:

- Offset and gain correction (add & multiply operation) of the four raw sensor signals (quadrature components of Ψ and Θ) coming from the ADC
- Transformation (arctan function) of the four quadrature signals into two angles Ψ and Θ
- Evaluation of the vernier function $\Phi = \frac{a \cdot (\Theta - \Psi) + \Theta}{b}$ with mechanical constants a and b into the wheel angle Φ

The operand and result data types of all operations shall be 16 bit two's complement. The total real-time constraint for the entire system is 4000 clock cycles. Since about 1000 cycles are required for communication and fault detection, the constraint for the algorithm outlined above is 3000 cycles. The target microcontroller architecture is the ST7 family (cf. [12]) from STMicroelectronics.

An excerpt from the module database is shown in table 4. ACP stands for an arithmetic coprocessor and CCP denotes a special coprocessor for the CORDIC algorithm. Columns that are common to all rows (like `cpu_type` = ST7) have been omitted.

The partitioning scheme presented in chapter 5 produces the following results:

- A complete software implementation (result of first iteration in figure 3) requires 1271 bytes of ROM space and 4026 cycles of execution time. Hence it misses the real-time requirement stated above.
- Moving the vernier resolution function (which contributes the largest execution time share) into the HWSW implementation domain requires instantiation of the ACP needing additional 8 kgates. As the presence of the ACP enables the transition of the other two operations into the HWSW domain, the execution time is reduced to 1181 cycles and the ROM usage is diminished to 788 bytes.
- Another valid alternative results if the implementation domain of the vernier function is set to SW as a user constraint. Now, the algorithm chooses to use the CCP for the arctan evaluation. It requires an extra 5 kgates and speeds up the entire calculation to 2753 cycles. The ROM usage amounts to 497 bytes.

At this time, the user may select between the second and the third alternative. The latter requires less chip area for the coprocessor and is hence favorable in terms of cost. If ROM space is a prime concern, the third choice is also preferable while the second solution leaves significantly more slack in terms of real-time behavior.

taxon _1	taxon _2	runt_ upper	cost (ROM)	cost (kgate)	needed_ id
Add	SW	15	8	0	—
Add	HWSW	7	4	0	ACP
Add	HW	1	0	0.2	—
Mul	SW	124	60	0	—
Mul	HWSW	7	4	0	ACP
Mul	HW	1	0	1.2	—
Atan	SW	1325	795	0	—
Atan	HWSW	785	522	0	ACP
Atan	HWSW	52	18	0	CCP
Vernier	SW	820	408	0	—
Vernier	HWSW	340	258	0	ACP
ACP	HW	—	—	8	—
CCP	HW	—	—	5	—

Table 4: Module database for experimentation

For comparison, we let POLIS synthesize all three modules and compile the resulting C files. The result takes up 2555 bytes ROM space and needs 12360 cycles for execution. Obviously, this is far beyond the constraints stated above and demonstrates the inappropriateness of this approach. The largest optimization potential for hand-written assembler code – remaining unused with synthesized C code – lies in the fact that *a* and *b* in the vernier function are constants for a given mechanical sensor layout. This greatly simplifies the multiply and divide operations.

7. Future Work

The partitioning granularity is currently dependent on the Esterel coding style. To become more specification-independent, we will explore the impact of granularity refinement algorithms (broadening the decision space) on the partitioning quality. The current partitioning algorithm and its reacttion to deadline violations is quite primitive and bears considerable potential for improvement.

Assigning the modules to a functional taxonomy classification is currently done manually. This will be automated by using more abstract representations of functionality like CDFGs and OBDDs. Another problem arises from the interface variable names of the database-stored modules not matching those in the specification. Currently, these literals are neglected during partitioning leading to ambiguities if modules have more than one input or output variable.

8. Conclusion

Our codesign system is targeted at industrial applications and covers both partitioning and coverification. We extended an existing cosimulation/cosynthesis environment by integrating an automated partitioning engine. Working on a database of reusable software and hardware modules, it allows to obtain better results on systems built around small resource-limited controller cores.

The object-oriented storage architecture of the module database allows efficient storage and retrieval of a large number of similar modules. It provides storage for information needed in the further development on the part of the partitioning algorithm.

9. References

[1] Balarin, F., Chiodo, M., Giusto, P., Hsieh, H., Jurecska, A., Lavagno, L., Passerone, C., Sangiovanno-Vincentelli, A., Sentovich, E., Suzuki, K., Tabbara, B.: Hardware-Software Co-Design of Embedded Systems – The POLIS Approach, Kluwer, Norwell, USA, 1997

[2] Madsen, J., Grode, J., Knudsen, P., Petersen, M., Haxthausen, A..: LYCOS: The Lyngby Co-Synthesis System, Design Automation for Embedded Systems, Volume 2, Issue 2, Kluwer, Norwell, USA, 1997

[3] Gajski, D., Vahid, F., Narayan, S., Gong, J.: SpecSyn: An Environment Supporting the Specify-Explore-Refine Paradigm for Hardware/ Software System Design, IEEE Trans. on VLSI Systems, Vol. 6, No.1, pp. 84-100, 1998

[4] Berry, G.: Esterel v5 Language Primer, Ecole de Mines and INRIA, Sophia-Antipolis, France, 1998

[5] Ernst, R., Henkel, J., Benner, T.: Hardware-Software Cosynthesis for Microcontrollers, IEEE Design & Test of Computers, December 1993

[6] Eles, P., Peng, Z., Kuchcinski, K., Doboli, A.: System Level Hardware/Software Partitioning Based on Simulated Annealing and Tabu Search, Design Automation for Embedded Systems, Volume 2, Issue 1, January 1997, Kluwer, Norwell, USA

[7] Vahid, F.: Modifying Min-Cut for Hardware and Software Functional Partitioning, 5th International Workshop on Hardware/Software Codesign, March 24-26 1997, Braunschweig

[8] Hollstein, T., Becker, J., Kirschbaum, A., Glesner, M.: HiPART: A New Hierarchical Semi-Interactive HW/SW Partitioning Approach with Fast Debugging for Real-Time Embedded Systems, 6th International Workshop on HW/SW Codesign, March 15-18 1998, Seattle

[9] Koegst, M., Conradi, P., Garte, D., Wahl, M.: A Systematic Analysis of Reuse Strategies for Design of Electronic Circuits, Proc. of DATE Conference 1998, February 23-26, 1998 Paris, France

[10] Olcoz, S., Ayuda, L., Izaguirre, I., Peñalba, O.: VHDL Teamwork, Organization Units and Workspace Management, Proc. of DATE Conference 1998, February 23-26, 1998 Paris, France

[11] Reutter, A., Rosenstiel, W.: An Efficient Reuse System for Digital Circuit Design, Proc. of DATE Conference 1999, March 9-12, 1999, Munich, Germany

[12] Reichmeyer, H.: Erfassung von Winkel und Positionen im KFZ, Elektronik Industrie 7/1999, p. 35ff, Huethig, Munich, Germany