Speed-Up Estimation for HW/SW-Systems

Wolfram Hardt[†], Wolfgang Rosenstiel[††]

email: hardt@uni-paderborn.de

[†]University of Paderborn, Warburger Str. 100

33 095 Paderborn, Germany

[††]University of Tübingen, Sand 13

72 076 Tübingen, Germany

Abstract

HW/SW-codesign has been applied to a wide range of applications. Several partitioning methods have been suggested. Thus the designer selects modules for HW or SW-implementation for the best possible performance within a set of performance and design constraints. This paper describes an estimation method to approximate a priori the entire system performance. The estimation method has been integrated into the codesign tool COD and first results could be generated. The estimated speed-up has been determined for a ciphering algorithm and has been compared to the speed-up of the entire HW/SW-system. The estimation speed-up matches the final speedup.

1. Introduction

During the last decade raising computing power and complex synthesis tools as well as decreasing chip size due to technology improvements lead to a higher level of automated design combining HW- and SW-components to HW/SW-systems. It has been pointed out that the partitioning process is rather complex and different approaches have been suggested [23]. Some approaches use estimation methods to capture the problem complexity. In [16] an estimation of the partitioning costs is presented. The estimated partitioning costs are used as partitioning criteria. Derivations between estimated and real costs may occur due to synthesis, compiler and communication effects. Further estimation approaches consider the SW-part. If constraints form the environment can be met by a SW-implementation additional HW can be avoided. Therefore a detailed estimation of SW-performance is necessary. In [10] an estimation of SW-performance is suggested. Based on a data flow graph a delay estimation is performed. Li and Malik use ILP to determine the bound on the running time of a given program on a given processor [18]. This solution considers all program paths implicitly and the drawbacks of static code evaluation are eliminated and more precise information is obtained.

Beside different estimation and partitioning methods different goals of HW/SW-partitioning are in view. In many

cases time constraints for real time applications can be met if HW/SW-codesign is introduced. The aim of partitioning is the acceleration of the time critical part of the application [25]. Other approaches optimize a HW/SW-implementation. This leads to a design of the same behavior but with a minimized HW-partition [20]. This is necessary if the design environment restrict the size of additional HW. Further approaches are not constraint driven but focus on an overall system optimization. The most important aspect is the computation time of the whole application. HW/SW-partitioning can raise the system performance significantly and a trade-off between flexibility and performance can be determined [13, 22]. This is especially interesting because standard components and standard architecture concepts are used and extended by special purpose components which are application specific. But several effects of HW/SW-partitioning may reduce the system performance, e.g., data transfer between HW- and SW-partitions are very expensive and time consuming. Of course, these costs depend on the chosen architecture as well as on the HW/SW-partitioning and it is difficult to quantify the reached improvements considering system performance a priori because not only the HW-partition must be in view but the
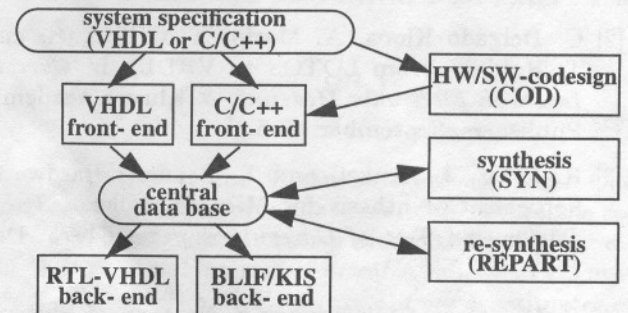

Figure 1. PMOSS - design environment

whole system. Therefore we investigated in this subject and in this paper we present a new estimation method approximating the speed-up of HW/SW-systems. Our estimation method is parameterized by the characteristic of the architecture and can therefore be applied to different codesign architectures. All data needed for speed-up estimation is generated automatically by our design environment PMOSS [6]. This environment (figure 1)

integrates a codesign tool *COD* [11], a high level synthesis tool *SYN* [9] and several front/back ends (C/C++, behavioral VHDL, RTL-VHDL, BLIF, KISS). Also re-synthesis from RT-level is supported [5, 4]. Based on this environment an automated design flow down to RT-level is provided. Starting the design flow with HW/SW-partitioning the COD-tool performs several phases of specification analysis as described later on. The design specification will be partitioned in order to accelerate the overall computation speed. The tasks performed by COD
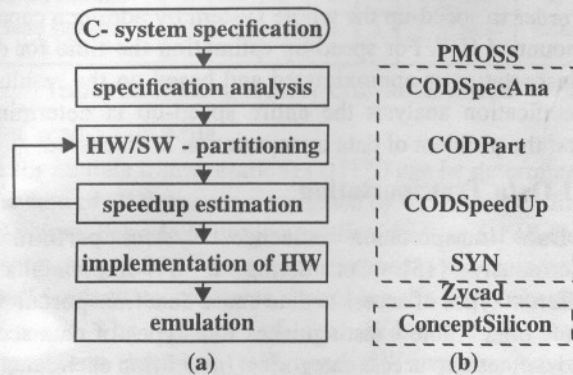

Figure 2. HW/SW-codesign flow (a) and involved tools from our design environment PMOSS (b)

are shown in figure 2. The HW-partition is passed to HW-synthesis. Finally, the synthesized HW-partition is integrated into the target system. Now, the performance of the implemented HW/SW-system can be determined after implementation, e.g., if appropriate benchmarks are executed. But this is a time consuming process and runs out of stream rather quickly if the partitioning process suggests several capable partitions. This can be avoided by a priori performance estimation. But performance estimation must regard not only the HW-partition but the whole HW/SW-system. At least, there are three problems which must be solved:

1. Beside the runtime comparison of a module's HW-implementation with the corresponding SW-implementation **data transportation** from and to the HW-partition must be taken into account.

2. An accurate **performance model** for HW/SW-systems is necessary. This model must be simple enough to handle large systems of e.g., several thousand lines of code and it must contain all details necessary to cover the most important effects of the runtime behavior of a HW/SW-system.

3. Performance model evaluation is rather important. All necessary data must be **extracted automatically** form the HW/SW-system's specification.

In this paper we suggest a solution for these problems

and present a speed-up estimation method. For demonstration, we apply the developed speed-up estimation method to a ciphering algorithm [8]. The estimated speed-up fit the speed-up determined experimentally.

The rest of this paper is organized as follows. First the codesign architecture is described (chapter 2). In chapter 3 data transportation costs are classified (chapter 3.1), an overview over the specification analysis task of the COD-tool is given (chapter 3.2) and the performance model for HW/SW-systems is defined (chapter 3.3). In chapter 3.4 the estimation method itself is presented. Our approach is demonstrated by a case study (chapter 4). Finally, a conclusion and some remarks to future work are stated.

2. Codesign Architecture

This subsection gives a short overview over the intended HW/SW-architecture. As suggested in [12] our codesign approach extends a powerful standard architecture by some specific HW (figure 3). This provides the computation
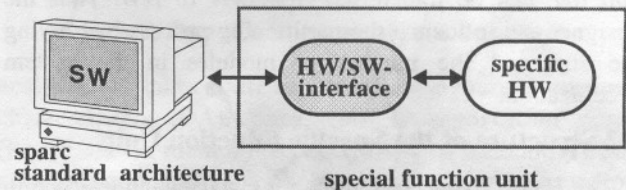

Figure 3. Target system architecture

power of the host processor (e.g., a pipelined RISC processors) as well as fast specific HW for special applications. But it is important, that the features of the standard architecture can be obtained also for HW/SW-implementations. E.g., many architectures provide a very fast cache which accelerates the data handling enormously. If a HW/SW-implementation disables the cache usage, a system speed-up can hardly be reached. This leads to a very tight coupling of the standard processor with the specific HW and requires a processor specific interface between both parts. But in this way a fast data transfer between the HW- and the SW-partition is reached.

2.1 The chosen Interface Concept

The specific HW and the HW/SW-interface (figure 3) build the special function unit (SFU). The SFU is linked to the standard CPU by a special coprocessor port. If this port is not supported by some processors the interface must be modified and a DMA unit must be integrated. But e.g., the SPARC processor provides the coprocessor interface and a standard configuration of the basic components (each available as single chip) and the integration of the SFU is depicted in figure 4. This interface concept allows the host (standard cpu) to control the SFU. Both units can be activated together and all data transfers are pipelined. Furthermore, his concept implies a bipartitioning of the

system specification into one HW- and one SW-partition because there is one communication interface. Both partitions may contain several modules. A module is
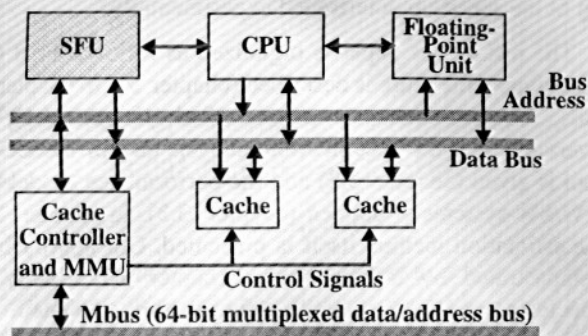

Figure 4. Standard configuration of a sparc architecture [3]

defined by a procedure (or function in 'C' syntax) of the system specification. A module is understood as smallest unit that can be transferred from SW to HW. Thus the designer can influence the partitioning process by varying the size and the number of modules in the system specification.

2.2 Structure of the Specific Function Unit

The HW/SW-interface provides fast data and instruction transportation. Instructions for data transfer and status information are implemented. Details can be found in [14]. Application execution may be divided into three parts. Input read, result computation including handling of temporary data and writing back the output are different tasks to perform. Generally, these are not sequentially executed. Input registers, a data path computing the results, result registers, and a controller must be provided. We call the data path specific function HW (SFHW). A block build
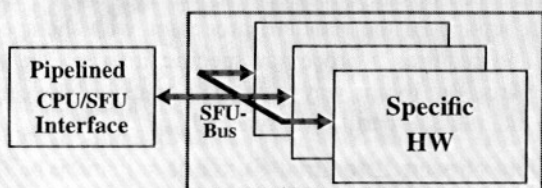

Figure 5. SFU structure

up of this four units is called specific function subunit (SFSU). In figure 5 a high-level block diagram is given. The number of input and result registers can be varied due to the implemented application up to 32. This SFSU is connected to the HW/SW-interface via a 46 bit bus. This bus consists out of 32 data bits, two times 5 bit to address input and result registers and some control and status bits. Such a SFSU is complex enough for a variety of applications. But there are also applications using more input or result registers. Furthermore there may be several modules

mapped to HW. Due to this fact the specific HW may be build up generic in the number of SFSUs. The necessary SFSU selection is realized by an additional input register. The number of SFSU is limited because of technical reasons to 2^{19}. This SFU structure requires no modification of the extern environment.

3. Speed-Up estimation

With respect to this codesign architecture the COD-tool partitions the system specification in a HW- and a SW-part in order to speed-up the whole system by adding a capable amount of HW. For speed-up estimation the time for data transportation is approximated and based on the results of specification analysis the entire speed-up is determined. First the problem of data transportation is discussed.

3.1 Data Transportation

Data transportation influences system performance enormously [15]. Considering a SW-implementation different types of access to data cause data transportation in some kind. Table 1 distinguishes four types of data access and defines two access categories. In addition each category

Types of data access	Access category
access to global data	A_{glob}
parameter transfer	A_{loc}
access to data via pointer	A_{glob}
access to local data	A_{loc}

Table 1: Classification of data access

distinguishes a read and a write class of data access. In practice the execution time of these data accesses vary and depend on the system memory hierarchy. But the two categories *local data access* (A_{loc}) and *global data access* (A_{glob}) can be identified clearly. A more detailed classification distinguishes first and second level caches and main memory as well as very slow secondary storage (disc, tape). The runtime behavior of the system memory hierarchy depend on the operating system. We regard such effects as external influences not to be considered in this context. However, the time characteristics of the defined categories are obviously not the same for HW- and SW-implementations. This is a critical aspect for system performance. Depending on the integration concept access from the HW to main memory can become extremely slow [21,7]. We express these architecture characteristics concerning data transportation by eight constants related to both data access categories and their classes. This constants parameterize the speedup estimation task. We present the characteristics of four different architectures in table 2. This parameter concept allows speed-up estimation for varying sparc based architectures with different HW/SW-interfaces.

Beside the type of the data access the number of data access is important. If a SW-implementation is in view the

Implementation	Software				Hardware			
Category	local		global		local		global	
Architecture	R	W	R	W	R	W	R^a	W^b
cache, pipelined interface.	2	3	2	3	1	1	6	9
without cache	5	6	5	6	1	1	9	12
simple interface	2	3	2	3	1	1	30	30
serial interface (115 KBit/s)	2	3	2	3	1	1	2200	2200

Table 2: Data transfer characteristics

a. read   b.write

time for all data transportations (DT^{SW}) can be determined as the sum of all data access weighted by the corresponding access time. For a HW-implementation the time for all data transportations (DT^{HW}) can be determined in the same way. Only the adequate constants describing the HW behavior are used. The effects of the differences in data transportation between HW- and SW-implementations depends on the number of data access. In practice the number of main memory accesses per module varies enormously. Thus the data transportation aspect may decrease or increase the system performance. This can be expressed by an efficiency coefficient η_{DT} derived from data transportation:

$$\eta_{DT} = \frac{DT^{SW}}{DT^{HW}}$$

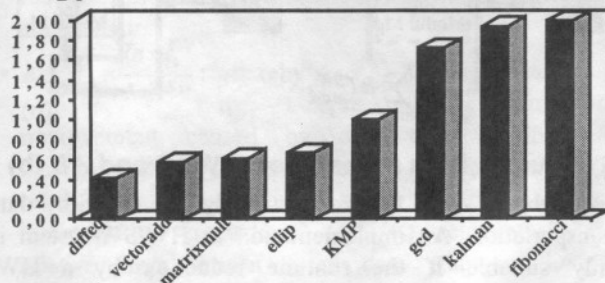If η_{DT} is smaller than 1 data transportation will reduce

Figure 6. η_{DT} **for some benchmarks [1]**

the system performance otherwise a HW-implementation of this module increases system performance because of data transportation. Our codesign architecture (chapter 2) is described by the parameters of the first architecture (table 2) due to tight coupling of processor and specific HW.

Figure 6 shows the detailed results for some benchmarks. Depending on the number of global data accesses a system speed-up can be reached (KMP, gcd, kalman, fibonacci). η_{DT} is limited by 2 due to the architecture parameters. The computation of η_{DT} is subject of one phase of specification

analysis and can be performed automatically. The next paragraph gives a brief overview over the specification analysis task. Details are presented in [11].

3.2 Specification Analysis Task

During specification analysis a design is thought of as a set of interacting modules. The suitability of each modul for HW-implementation is examined during four phases. The
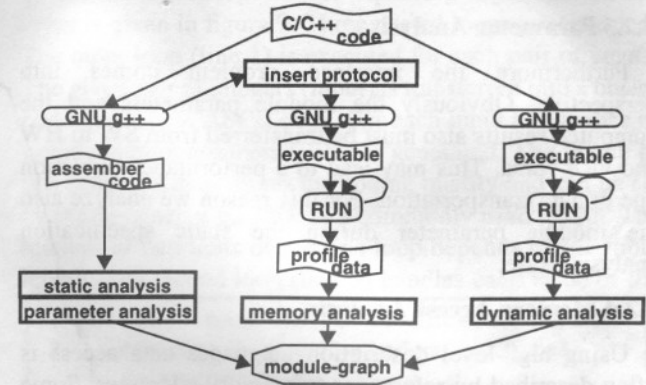

Figure 7. Specification analysis scenario

analysis task takes **static aspects** (SA), **dynamic runtime** characteristics (DA), **parameter transportation costs** (PA) and main **memory access** (MA) into account. These specification analysis phases result in a cost vector $\Psi = (SA, DA, PA, MA) \in \mathbb{R}^4$. In figure 7 the analysis scenario is presented.

3.2.1 Static Specification Analysis

Static specification analysis examines assembler code without execution. A lower bound on the execution time for a SW-implementation is computed. Also the number of jump instructions and bit-level instructions, e.g., AND, EXOR per module are counted because these instructions can be executed in HW much quicker than in SW. E.g., the comparison of two single bits can be done in HW very fast by only one gate. A SW-implementation e.g., for a pipelined architecture needs one cycle at least. However, in order to reach a reasonable speed-up the accelerated module must be of an appropriate size. The approximated runtime of a SW-implementation is interpreted as heuristic definition of the module size. These aspects determine the *SA* component of the cost vector.

3.2.2 Dynamic Specification Analysis

Dynamic specification analysis examines the runtime behavior. Profiling data generated by execution of the SW-implementation of the design is analyzed. Considering a single module three aspects, the absolute, average and relative runtime of a module are taken into account:

- RT^{SW}_{abs} : The time consumed by a module during the whole execution of the SW-implementation of the

design.

- $RT_{average}^{SW}$: The average of all execution times per module found during the whole execution of the SW-implementation of the design.

- RT_{rel}^{SW}: The percentage of a modules absolute runtime of the execution time of the SW-implementation of the design.

3.2.3 Parameter Analysis

Furthermore the interface problem comes into perspective. Obviously the module parameters and the computed results also must be transferred from SW to HW and vice versa. This may lead to a performance reduction due to data transportation. For this reason we analyze also the module parameter during the static specification analysis phase.

3.2.4 Memory Access Analysis

Using high-level description languages data access is often described by references to memory addresses. Some of them are local data values others are of global range and stored in main memory. The number of accesses to main memory can only be evaluated during runtime because of data dependencies. In our approach the behavioral description is read by a parser and automatically modified in order to generate a protocol of all memory accesses during the application execution. The modified specification is compiled to an executable and test pattern are applied. The generated data is examined during the design execution and the number of local and global *read accesses* as well as the number of local and global *write accesses* to main memory are computed and transferred to the central codesign data structure (module graph). Thus an automated procedure is provided to generate all data necessary for the computation of η_{DT}.

All four phases of the specification analysis task are performed automatically and provide detailed system characteristics. Because design execution instead of design simulation is used the runtime consumed by specification analysis is very short.

3.3 Performance Model for HW/SW-Systems

The single aspects of HW/SW-systems presented above must be integrated into a performance model. This model considers:

- the whole HW/SW-system

- all data transports

- and the performance characteristic of the HW-partition.

In this paper we restrict ourselves to a HW/SW-bipartitioning which contains one SW-partition containing several modules and one HW-partition containing only one

module. In principal, the HW-partition is build up by several modules and implemented on one special function processor. But the presented performance model and the estimation method capture this also. We define the HW/SW-system (UUC= <u>U</u>nit <u>U</u>nder <u>C</u>odesign) as set of all modules needed to specify the system. The runtime of a pure SW-system can be build up by the runtime of all modules $\left(RT^{SW}(UUC) = \sum_{mod} RT_{mod}^{SW} \right)$

with $mod \in UUC$. Transferring one (or more) modules (M_i) into HW reduces the SW-partition's complexity and runtime $\left(\sum_{mod \neq M_i} RT_{mod}^{SW} \right)$. Additional runtime is consumed by the HW-partition itself (RT^{HW}) and additional data transportation (DT^{SW}) must be performed. The runtime of the HW/SW-system can be summerized as:

$$RT^{HWSW}(UUC) = \sum_{mod \neq M_i} RT_{mod}^{SW} \\ + RT^{HW}(M_i) + \\ DT^{HW}(M_i) - DT^{SW}(M_i)$$

In figure 8 the characteristics denoted by RT^{HWSW} are pointed out. All data transport to main memory is handled by SW. The HW-partition has no direct access to main memory. Due to our HW/SW-interface this data transport can be executed very efficiently. Also the memory hierarchy of the standard architecture is not restricted. So full cache usage can be provided for the HW/SW-system. The speed-
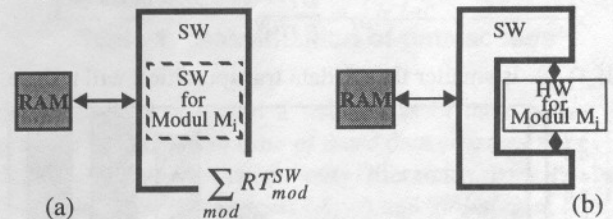
Figure 8. Data access from SW (a) and HW (b)

up of the HW/SW-system is strongly influenced by data transportation. An implementation as HW/SW-system is only suitable if the runtime reduction by a HW-implemented module is not payed with more additional runtime for data transportation between the partitions. In formal terms: $RT^{SW} - RT^{HW}(DT^{HW} - DT^{SW})$

The important influence of data transportation is evident. Based on this performance model our estimation method approximates these effects.

3.4 Estimation method

The abstract performance model is approximated by our estimation method determining the speed-up reached by a HW/SW-implementation of the initial SW-system. Speed-up is understood here as the reduction of system runtime in percent. The computed speed-up reached per module which

is implemented in HW is defined as:
$CODSpeedUp\,(UUC,m) =$

$$RT_{rel}^{SW}(m) \times \left[\frac{RT_{schedule}^{HW}(m)}{RT_{approx}^{SW}(m)} + Acc_{mem}^{SW}(m) \times \frac{Inst_{ls}^{SW}(m)}{Inst^{SW}(m)} \right]$$

This equation is based on three components:

- RT_{rel}^{SW}: The module M_i is transferred to HW. The runtime of the remaining SW-partition is reduced by the runtime of M_i. This builds the basis for speed-up estimation. Theoretically, the reached speed-up cannot exceed RT_{rel}^{SW} (Amdahl's Law). An approximation of the real acceleration is determined by:

- $\frac{RT_{schedule}^{HW}}{RT_{approx}^{SW}}$: This quotients compares two static run-time approximations. $RT_{schedule}^{HW}$ refers to the number of control-steps on the longest acyclic path through the finite state machine of the controller of the HW-implemented module. RT_{approx}^{SW} denotes the static approximation of the execution time of a SW implementation. Here, the longest acyclic path through the data flow graph is regarded. Both approximations are static and consider every loop only once. The quotient of both approximations gives an rough idea of the speed-up reached by HW implementation if no data transfers are taken into account. But the advantages of HW-implementation because of intensive usage of inherent parallelism etc. are taken into account. Because this is only a static relation not regarding any influence of data dependencies this quotient is related to the dynamic runtime of the regarded module M_i. The speed-up reduction of data transportation is captured by the last component:

- $Acc_{mem}^{SW} \times \frac{Inst_{ls}^{SW}}{Inst^{SW}}$: whereby Acc_{mem}^{SW} is determined by $\eta_{DT} - 1$ representing the performance improvement caused by data transportation. As mentioned above this improvement can be negative (in relative terms $Acc_{mem}^{SW} < 1$) but this is only for data transportation instructions ($Inst_{ls}^{SW}$) relevant. So the relative improvement from data transportation is weighted by the relative number of data transportation instructions ($Inst_{ls}^{SW}/Inst^{SW}$) which is obtained from specification analysis phase SA (compare with R3 and R4 in table 4).

4. Speed-up Estimation for a Ciphering Algorithm

For demonstration of our estimation method we present in this paper a case study. In this case study one algorithm has been implemented and examined. The estimation method has been applied and the estimated system speed-up is compared to the experimentally determined system performance. For this case study, we chose a ciphering algorithm [8] because ciphering algorithms need high computation power and are typical candidates. This algorithm belongs to the class of stream ciphering algorithms and is used for encryption of numerical messages.

4.1 The Algorithmic Concept

The basic concept of the ciphering algorithm named *crypt* is given in figure 9. Three nested loops are executed. The outer loop (line 1) is executed for each pair of inputs. The given pair of integers (input) is transferred into a binary coding, e.g., the ASCII code. For each input a sequence of pseudorandom numbers (PRS) is generated. The length of this sequence determines the coding quality and can be set by the designer, e.g., *1000* is a commonly used length. The number of iterations of the first loop depends on the input length. The second loop (line 3) handles each value of the

```
1 for each pair (n,m) of integer inputs
2     compute a pseudorandom integral
      sequences N,M
3     for each value v of sequence N, M
4         converted v into a fibonacci
          binary sequence vN, vM
5         for an appropriate t
6             perform the bitwise logical
              sum of the central portions
              of vN, vM
7         end
8     end
9 end
```

Figure 9. Ciphering algorithm of Filipponi

computed PRSs. Thus the number of iterations of this loop is fixed by the sequence length. For each value a representation as binary fibonacci sequence is computed. This sequence contains only fibonacci numbers and the sum of these numbers are equal to the value in view (v). So, this sequence can be represented by a binary vector indicating which fibonacci number must be summed up. The last step is performed in the inner loop (line 5). The binary fibonacci sequence ob both input values are combined by an logical operator.

That means, for the input pair (h, i) the integer pair (104, 105) is generated. Now the PRS is computed, e.g., PIS(105) = 59922, 1914249, 5398140, 5398269, 5402268, 7924660... Each of this values is transformed into the fibonacci binary sequence (FBS), e.g., FBS(222208) = (1, 0, 1, 0, 1, ..., 0). The last step combines a part of these FBSs by a binary operator. It has been shown in [2] that a fibonacci binary sequence is unique. Based on this sequence a fairly satisfactory ciphering sequence can be generated. See also [[Fi96]].

4.2 HW/SW-Partitioning

We have implemented this algorithm in C++ on an unix workstation. Some general information about the implementation shows table 3. The SW-implementation

Characteristic	Value
lines of C++ code	578
lines of assembler code	7006
number of modules	104

Table 3: Characteristics of benchmark crypt

consists out of 104 modules and ca. 7000 lines of (SPARC) assembler code. This shows that the size of this algorithm is large enough that a HW/SW-implementation can be considered. And a more fine granular analysis, e.g., on data flow graph basis is hardly capable. But our specification analysis can be performed rather quick. The exact execution

SA-Result	Note
R1	number of jump-operations per module
R2	number of bit-level-operations per modul
R3	number of load/store-operations per modul
R4	number of operations per modul
R5	control-dominance = (R1+R2) / R4
R6	approximated maximal SW-runtime

Table 4: Results of static spec. analysis

time is mainly determined by the dynamic analysis phase because the algorithm is executed several times with different inputs. The static specification analysis phase (SA) computes a detailed statistic about the used instructions as derived in table 4. These results were automatically computed for the crypt algorithm and are listed in table 5 for the most important modules. Some of these results are

Module	R1	R2	R3	R4	R5	R6
CompCiphring	53	19	65	272	26	134
CompLvalue	12	9	23	97	21	229
contents	4	0	4	20	20	21
fibonacci_n	4	0	14	31	12	18

Table 5: Results of SA for benchmark _crypt_

relevant for our speed-up estimation method (R3, R4, R6). The partitioning task is based on the results of all four specification analysis phases and classifies the module _fibonacci_n_ as suitable for HW-implementation. This module computes the _n-th_ fibonacci number. The dynamic analysis points out that this module consumes 11% of the system runtime. Figure 6 shows that the memory analysis

phase results with a very good value for η_{DT} (1.94). Thus, the partitioning seems to be good and speed-up estimation was applied in order to determine the speed-up for the HW/W-system.

4.3 Speedup Estimation

All data needed for computation of the speed-up has been generated automatically. As mentioned above specification analysis and scheduling are performed before. The obtained data is given in table 6. Now the speed-up estimation can be

Component of estimation method	Obtained data
RT^{SW}_{rel}	11%
RT^{SW}_{approx}	18 cycles
$RT^{SW}_{schedule}$	6 clock steps
Acc^{SW}_{mem}	0.73
$Inst^{SW}_{ls}$	14
$Inst^{SW}$	31

Table 6: Obtained data for estimation method

computed easily: $CODSpeedUp\ (crypt, fibonacci) = 8\ \%$

A reasonable speed-up of the whole system of 8% has been estimated. This indicates also that the generated partitioning will speed-up the system. This could be proved by implementation and emulation of the HW/SW-system.

4.4 Implementation and Emulation

After speed-up estimation the module _fibonacci_n_ has been implemented in HW using our synthesis tool _SYN_. This tool generates a multiplexer based RT-level description of this module. The RTL-description consists out of a controller and a data path. The controller is necessary because the number of iteration which are needed to compute the _n-th_ fibonacci number depends on the parameter _n_. So loop unrolling is not possible. The generated RTL-VHDL description was optimized and mapped to a HW library using the Synopsys design environment [19]. The optimized design was transferred to a EDIF-netlist [17]. This netlist was given to the Concept-Silicon-Software of Zycad [26]. This step is needed to partition the netlist for implementation of the emulator HW. The Zycad emulator is build up by two boards. Each board contains eight daughter boards and each daughter board contains three FPGA components. The designer can chose an automated partitioning process or partition the netlist manually. During the last step the partitioned netlist is mapping to the FPGA components (Xilinx 4010) [24]. The complete circuit used 2500 gates on the emulator and fits into one FPGA.

Once the module is implemented runtime experiments can be performed. For the same input data a speed-up of the

module by HW-implementation of 94% was reached (table 8). But this include no data transfers to the SW-partition. Now, the module is integrated into the codesign architecture and emulation can be performed again. Therefore it is assumed, that the special function unit and the host processor run by the same clock rate. It could be obtained a speed-up for the whole system of 10% (table 8). This experiment include the execution of the SW-partition on a SPARC-based workstation, the data transfers via the HW/SW-interface, the execution of the HW-partition and all data accesses from the HW-partition.

Benchmark	Note	Result
fibonacci_n	SW-Implementation	56 us
fibonacci_n	HW-Implementation	3 us
fibonacci_n	speed-up	94%
crypt	speed-up	10%

Table 7: Experimental determined speed-up

The determined speed-up of the HW/SW-system *crypt* is 10%. This is close to the estimated speed-up of 8%. The same experiment for further benchmarks is under development. The estimation method was implemented and can be performed automatically. Thus the speed-up for the whole HW/SW-system can be approximated very fast and the implementation of inefficient partitions can be avoided.

5. Conclusion and future work

In this paper we presented an automatically performed estimation method to approximate the speed-up of a SW-system implemented as HW/SW-system. Therefore a performance model was established, data-transfer costs classified and an estimation method defined. The estimated speed-up meets the experimentally determined speed-up. Although the design flow is widely automated the experiments are quite time consuming because a lot of technical problems have to be solved. But we will apply our estimation method to more complex benchmarks in near future.

6. References

[1] Benchmarks for the 6th International Workshop on High-Level Synthesis. Available through electronic mail at ics.uci.edu, November 2-4 1992. Proc. of the 6th International Workshop on High-Level Synthesis.

[2] J. L. Jr. Brown. Zeckendorf's Theorem and Some Applications. In *The Fibonacci Quaterly 2*, 1964.

[3] Cypress Semiconductor Ross Technology Subsidiary, 3901 North First Street, San Jose, CA 95134. *Sparc/RISC User's guide*, 1990.

[4] H.-J. Eikerling. Entwurfsdarstellung durch BDDs für die Resynthese. In *GI/ITG-Workshop Anwendungen formaler Methoden im Systementwurf*, Frankfurt, March 21-22 1994.

[5] H.-J. Eikerling and R. Camposano. CP-/DP-Partitionierung und Resynthese. In *6. E.I.S.-Workshop*, Tübingen, November 25-26 1993.

[6] Heinz-Josef Eikerling and Wolfram Hardt. *PMOSS: Paderborner Modular System for Synthesis and HW/SW-Codesign*. University of Paderborn, Warburger Straße 100, 33098 Paderborn, 1995.

[7] R. Ernst and J. Henkel. Hardware-Software Co-design of Embedded Controllers based on Hardware Extraction. In *Proc. of the 2nd ACM Workshop on Hardware/Software Codesign*, October 1992.

[8] P. Filipponi and E. Montolivo. *Application of Fibinacci Numbers*, pages 89–99. Kluwer Academic Publishers, Boston/Dordrecht/London, 1990.

[9] R. Geneviere and A. Hoffmann. PMOSS - A Modular Synthesis and HW/SW-Codesign System. Technical Report SFB - 358 - B2 - 2/94, University of Paderborn, Technical University of Dresden, 1994.

[10] R. K. Gupta and G. De Micheli. Constrained software generation for hardware/software systems. In *Third International Workshop on Hardware /Software Codesign*, pages 56–64, Grenoble, September 1994.

[11] W. Hardt. An Automated Approach to HW/SW-Codesign. In *IEE Colloquium: Partitioning in Hardware-Software Codesigns*, London, Great Britain, February 13 1995.

[12] W. Hardt and R. Camposano. Trade-Offs in HW/SW-Codesign. In *Proc. of the 3rd ACM Workshop on Hardware/Software Codesign*, Cambridge, MA, October 7 - 8 1993.

[13] W. Hardt and R. Camposano. Specification analysis for hw/sw-partitioning. In W. Grass and M. Mutz, editors, *3. GI/ITG Workshop : Anwendung formaler Methoden für den Hardware-Entwurf*, pages 1–10, Passau, March 1995. Shaker-Verlag, Aachen.

[14] W. Hardt, A. Günther, and R. Camposano. Pipelined Interface for HW/SW Codesign. Technical Report SFB - 358 - B2 - 3/94, University of Paderborn, Technical University of Dresden, 1994.

[15] J.L. Hennessy and D.A. Pattersopn. *Computer Architecture : A Quantititive Approach*. Morgan Kaufmann Publishers, San Mateo, CA, 1990.

[16] D. Herrmann, J. Henkel, and R. Ernst. An approach to the adaptation of estimated cost parameters in the cosyma system. In *Third International Workshop on Hardware /Software Codesign*, pages 100–107. IEEE Computer Society Press, September 1994.

[17] Interim standard. *EDIF/EIAL Library of Parameterized Modules*, 1993.

[18] Y. Li and S. Malik. Performance analysis of embedded software using implicit path enumeration. In *DAC_95*, pages 456–461, DACadr_95, June 1995. ACM/IEEE.

[19] Synopsys, Inc., Mountain View, CA. *VHDL Design Analyzer (tm) Manual*, 3.0 edition, 1992.

[20] F. Vahid, J. Gong, and D Gajski. A Binary-Constraint Search Algorithm for Minimizing Hardware during Hardware/Software Partitioning. In *Proc. of the European Design Automation Conference*, pages 214–219, Grenoble, France, September 1994.

[21] M. Wendling and W. Rosenstiel. A Hardware Environment for Prototyping and Partitioning Based on Multiple FPGAs. In *Proc. of the European Design Automation Conference*, pages 77–82, Grenoble, France, September 1994. IEEE.

[22] J. Wilberg, R. Camposano, and W. Rosenstiel. Design flow for hardware/software cosynthesis of a video compression system. In *Third International Workshop on Hardware /Software Codesign*, pages 73–80. IEEE Computer Society Press, September 1994.

[23] W. Wolf. Hardware-Software Codesign of Embedded Systems. *Proceedings of the IEEE*, 83(7):967–989, 1994.

[24] Xilinx Corporation. *Xilinx Programmable Gate Array Data Book*, 1994.

[25] W. Ye, R. Ernst, Th. Benner, and J. Henkel. Fast Timing Analysis for Hardware-Software Co-Synthesis. In *Proc. of the International Conference on Computer-Aided Design*, pages 452–457, Santa Clara, CA, 1993. IEEE.

[26] Zycad Corporation, Inc., USA. *Concept Silicon Software (tm) Manual*, 6.0 edition, 1994.