HW/SW Partitioning Using High Level Metrics

B. Knerr, M. Holzer, and M. Rupp
Vienna University of Technology
Institute for Communications and RF Engineering
Gusshausstr. 25/389, 1040 Wien, Austria

ABSTRACT

The Kernighan/Lin graph partitioning heuristic, also known as min-cut, has been enhanced over decades and successfully adopted for circuit partitioning. During last years extended versions have been developed to address functional partitioning as it is needed to solve the much investigated co-design problem of partitioning a system's coarse-grained functions among hardware and software components. In combination with advanced code parsing on system level to derive reliable metrics for execution time and hardware size, the heuristic is immediately applicable in an automatic way to achieve a partitioning decision in the early stages of a design process. In this paper we demonstrate a consistent design methodology combining advanced code analysis and an extended Kernighan/Lin heuristic for functional partitioning applied to an industry-designed UMTS receiver component at highest abstraction level.

Keywords: HW/SW Partitioning, Kernighan/Lin Heuristic, High Level Metrics, Single System Description, System on Chip, Design Methodology

1. INTRODUCTION

Hardware/software (hw/sw) co-design is a design paradigm for the joint specification, design and synthesis of mixed hw/sw systems. The interest in automatic co-design techniques is driven by the increasing diversity and complexity of applications employing embedded systems and the need for a decreasing cost in the design and test of hw/sw systems, with the aim of reducing time-to-market. At numerous times during the design process crucial decisions have to be made that dramatically influence the quality and the cost of the final solution. Some design decisions might have an impact of about 90% of the overall cost [1], among these hw/sw partitioning is of prominent relevance.

Partitioning is a well-known NP-complete problem and during the last years many heuristics have been proposed to guide the partitioning of a system into hw and sw components [2–4]. However, the feasibility of these methods depends exclusively on the accuracy of parameters for the values to be optimised, in the main part execution time, hardware size, power consumption, etc. for every component of a design. One way to obtain accurate estimates for optimisation is the synthesis of each single component of the design on the one hand and its simulation on cycle level on different processor cores on the other. Because of the tough time-to-market requirements especially in the wireless domain we cannot afford such exploration methods. This fact entails the need for accurate high-level estimation techniques based on the algorithmic description of the design, i.e. as early as possible in the design cycle. Due to the limited space in this paper we will focus on the estimation and optimisation of execution time and hardware size. In order to obtain reliable metrics for these, in-depth code analysis of each component is obligatory. In this paper we introduce a procedure which combines such code analysis on system level with estimation techniques for execution time and hardware size to which we immediately apply an extended Kernighan/Lin heuristic for functional hw/sw partitioning. As this procedure is embedded in a consistent design methodology, consisting of a design database (DDB) with interfaces to various EDA tools [5], an essential speedup in the design cycle can be achieved.

The rest of the paper is organised as follows. Section 2 points out some contributions in related work in the field of high-level metrics and hw/sw partitioning. Section 3 demonstrates the metrics we selected and how we obtain them by means of static code analysis. In Section 4 these metrics are combined to build a cost function and the chosen Kernighan/Lin heuristic is explained. This is followed by Section 5 describing the integration of the analysis and partitioning process into a consistent design methodology. The following section covers an application example of a UMTS delay profile estimation module and reports on achieved results. In the last section conclusions are made and perspectives to future work are given.

This work has been funded by the Christian Doppler Pilot Laboratory for Design Methodology of Signal Processing Algorithms.

2. RELATED WORK

2.1 High Level Metrics

High level metrics are estimates of the design at an early stage before the synthesis process. Those estimates must satisfy three criteria: accuracy, fidelity, and simplicity [6]. Evaluating the effect of a decision requires a complete run through the design process, is therefore extremely time consuming and cannot be used for high level design decisions. Structural analysis of an algorithm achieves the needed performance in order to be useful for larger projects.

For estimating the gate count of a hardware implementation, first approaches started by simple counting the number of all operations inside a function [7] but neglecting the distribution of the operators over the algorithm or the longest path. Other techniques like [8] analyse the descriptions on register transfer level (RTL) by investigating the graph of boolean networks (BN). For those graphs, structural metrics like number of nodes, edges, fan-in, and fan-out are derived. The application of this approach is limited to pure data flow representations.

Estimating the run time of a hardware implementation on an FPGA is achieved in [9] by simulating the algorithm and using a performance model of the FPGA.

Reliable estimation of the execution time of an algorithm implemented in software running on a processor can also only be performed by simulation which requires compilation and appropriate test vectors. High level estimation without compiling for a target architecture requires the development of a virtual processor as shown in [10]. Usually such a processor model is not available and rather time consuming to develop.

2.2 HW/SW Partitioning

There are many approaches to hw/sw partitioning. One of these is the COSYMA system [11], where hw/sw partitioning is based on simulated annealing using estimated costs. The partitioning algorithm is software-oriented, because it starts

with a first non-feasible solution consisting only of software components. In an inner loop partitioning (ILP), software parts of the system are iteratively realised in hardware until all timing constraints are fulfilled. To handle discrepancies between estimated and real execution time, an outer loop partitioning (OLP) restarts the ILP with adapted costs [12]. The OLP is repeated until all performance constraints are fulfilled.

Another hw/sw partitioning approach is realised in the VULCAN system [13]. This approach is hardware-oriented. It starts with a complete hardware solution and moves stepwise parts of the system to the software as long as the performance constraints are fulfilled. In this approach performance satisfiability is not part of the cost function. For this reason, the algorithm can easily be trapped in a local minimum.

The approach of Vahid [14] uses a relaxed cost function to satisfy performance in an inner partitioning loop and to handle hardware minimisation in an outer loop. The cost function consists of a very heavily weighted term for performance and a second term for minimising hardware. The authors present a binary-constraint search algorithm which determines the smallest size constraint (by binary search) for which a performance satisfying solution can be found. The partitioning algorithm minimises hardware, but not execution time.

Kalavade and Lee [15] presented an algorithm (GCLP) that determines for each node iteratively the mapping to hardware or software. The GCLP algorithm does not use a hardwired objective function but it selects an appropriate objective according a global time-criticality measure and another measure for local optimum. The results are close to optimal but the runtime grows quadratically to the number of nodes. This approach has been extended to solve the extended partitioning problem [2] including the implementation selection problem.

3. METRICS

Partitioning into hw and sw of a system is performed by evaluating a cost function assembled from competing metrics. These metrics are obtained by static code analysis on the highest level of abstraction - the algorithmic description of the system. High level metrics for the hw size, the gate count (GC), and the execution time, the cycle count (CC), for the two implementation alternatives, hw or sw, are computed.

In general an algorithm in form of sequential code can be decomposed into its control flow graph (CFG), built up of interconnected basic blocks (BB). Each basic block contains a sequence of data operations ended by a control flow statement as last instruction. This sequence of data operations forms itself a data flow graph (DFG). Figure 1 shows an example of a function and its graphical descriptions.

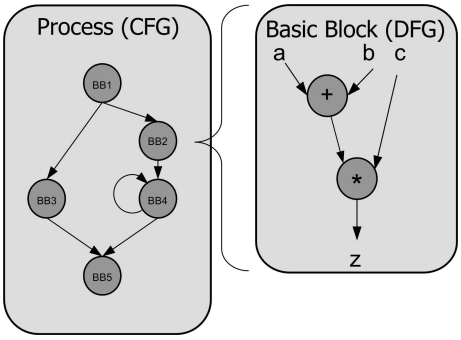

Figure 1: *Example of a CFG with internal basic blocks composed of a DFG.*

It is assumed that the maximum loop counts in the algorithm are known, what is usually the case for signal processing algorithms. All N basic blocks can be annotated with a set of different weights $W(BB_i) = \{w_1, w_2, .., w_m\}$, $i = 1 .. N$ that describe all its internals (e.g. w_1 = number of ADD operations, w_2 = register accesses, w_3 = cycle count). We define the longest path dependent on the jth distinct weight, LP_j, as that sequence $(BB_k\, BB_l\, ..\, BB_m)^T$ of BBs through the CFG, which yields a maximum path weight PW_j by summing up all the weights w_j of the basic blocks that belong to this path, see Eq. (1).

$$PW_j = \sum_{\substack{i=1 \\ BB_i \in LP_j}}^{N} w_j(BB_i) \qquad (1)$$

By focusing on this longest path the sequence of data flow graphs inside of every BB in the path can be combined to a single huge DFG_{LP_j}.

3.1 Hardware Gates Count

The high-level metric for the gate count has two constituent parts: one covering the operational units (ADD, MAC, etc.), $GC_{op,all}$, and one covering the registers, which store the intermediate results of the operations, GC_{reg}. For the number and size of the required operational units, we investigate the longest path with respect to every single type of operation. Note that the gate count for an operational unit varies with the bit width of its input variables (n,m). Table 1 lists the estimation of the gate count number for the operations and their dependency on the bit widths of the inputs. Hence, from the longest path analysis we obtain for every type of operation the required gate count $GC_{op,i}$. We assume a stream based

Operation i	$GC(n,m)$
ADD, SUB	$10\,max(n,m)$
MUL	$10\,n\,m$
AND, XOR, OR	$2\,min(n,m)$

Table 1: *Gate count for operation dependent on the bit widths of inputs n and m.*

processing implementation of the algorithm, that means every instantiated arithmetic unit is needed for the final implementation (no reuse of hw units possible during the synthesis process). Therefore the overall hw size for the operational units of this process is the sum of all the $GC_{op,i}$.

In order to get the hw effort for the required registers the longest path regarding its number and sizes is computed. The basis of this estimation is the number of edges between the nodes in the DFG representation (see Figure 2) of every BB. As already mentioned a stream based design is assumed

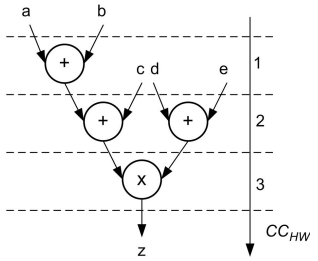

Figure 2: *Deriving high-level metrics from a Data Flow Graph.*

where the result of each computation, which corresponds to an edge in the CFG, is stored in a register. Therefore, the output bit widths of all operations have to be summed up. We approximate each register with seven gates for the implementation. To yield reliable results this computation requires

a fixed-point description of the algorithm. For floating-point descriptions standard bit widths had to be assumed (e.g. 32 bit for all *float* types, 16 bit for all *integer* types), which are misleading in most cases.

3.2 Hardware Cycle Count

For the cycle count of the algorithm implemented in hw, CC_{HW}, the depth of the DFG associated to every BB is measured, as shown in Figure 2. The depth serves as another weight variable, and a longest path considering this weight can be determined. The result is the maximum cycle count for a path in the selected process.

3.3 Software Cycle Count

In [16] for the cycle count of a software implementation, of the algorithm on a processor, CC_{SW}, the MIPS value is applied. As we only aspire toward a relative value for the comparison to the cycle count of the hardware realisation, we consider also cycle counts for the sw realisation instead of real time units. In comparison to the CC_{HW}, we do not consider the depth of the DFGs for every BB, but the total number of instructions within the BB. In other words there is no exploitation of parallelism within a BB. Using this weight a longest path is computed resulting in a metric for the sw cycle count. Scheduling of the processor instructions is not taken into account.

4. HEURISTIC FOR HW/SW PARTITIONING

4.1 Problem Definition

We are given a modular design consisting of interconnected components containing at least one sequential process. Each of these processes can be written in C or some similar procedural language. The processes communicate via the connections, between the components. The design is to be implemented on a target architecture consisting of a standard processor with memory (the software part) and several hardware accelerators possibly with its own memory (the hardware part). A generic version of this is shown in Figure 3. The

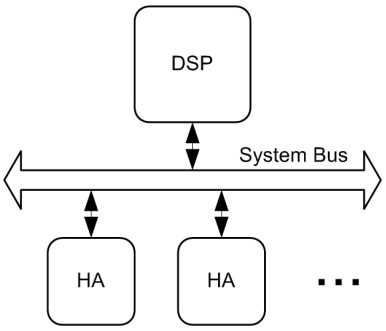

Figure 3: *Standard architecture with a DSP and hardware accelerators.*

standard processor may use a system bus, and/or may have several bidirectional data ports (such as commonly found on microcontrollers). The hardware accelerators can access the software memory, as well as its own local memory. Our problem is to assign every piece of the program to either the software or hardware part, such that we minimise execution time while satisfying any size and I/O constraints. To achieve such a partition, we first decompose the program into functional objects, where each object represents a process. The synchronous DFG (SDFG) represents each process as a node (P_i), and each connection as an edge (e_i), as illustrated in Figure 4. Each node is annotated with estimates of internal process execution times, the cycle count, for hw (CC_{HW}) and

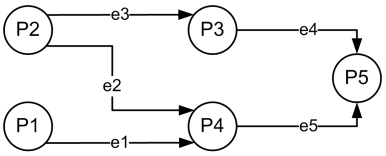

Figure 4: *SDFG with intercommunicating processes.*

sw (CC_{SW}) and potential hardware size (gate count GC); the cycle count represents a node's execution time on a given part, hw or sw, ignoring any communication time with external nodes. Each edge is annotated with the number of bits

Process P_1		Communication e_1	
GC	100	freq	24
CC_{HW}	30	bits	1024
CC_{SW}	95		

Table 2: *Example annotations for P_1 and e_1.*

sent during each transfer, and the frequency of the transfer. This information is derived from the data rates, from the number of samples demanded at the output - both available from the algorithmic description of the processes - and the chosen bus width, which has to be specified by the designer. The actual time to send the bits during a transfer over the edge depends on the bus delay to which the edge is assigned. We need to multiply this delay by the number of transfers required to transfer the edge's bits over the bus width. For example, transfering 32 bits over an 8-bit bus with a bus delay of 6 cycles will require $6 * 32/8 = 24$ cycles. When two adjacent nodes are opted to be sw, they communicate via shared memory thus not suffering from a bus delay. Note that all annotations are determined via the metrics described in Section 3.

4.2 Cost function

It is assumed that for the purpose of hw/sw partitioning an optimisation algorithm is used which optimises by iterative improvement. For each proposed partitioning the cost function, Eq. (2), calculates the cost in terms of the design goals: meeting real-time constraints and minimising hw effort. Since time and area have different physical units, normalisation of the components is required:

$$Cost = K_{GC}\frac{GC_{sys}}{GC_{\max}} + K_{CC}\frac{CC_{sys} - CC_{\min}}{CC_{\max} - CC_{\min}}. \qquad (2)$$

As for the hardware effort, we calculate the sum of the GCs of all those components flagged to be hw ($= GC_{sys}$) and divide by the sum of the GCs of all components ($= GC_{\max}$). Hence we obtain a value between 0 and 1, 0 is representing an all sw solution whereas 1 is representing an all hw solution. As for the execution time ($= CC_{sys}$), the normalisation is a bit more complicated. Since the real attainable minimum and maximum values constitute NP-hard problems by themselves, we calculate lower and upper bounds for $CC_{\min}$ and $CC_{\max}$ that might not be attainable as such, but are sufficient to normalise the exact value for CC_{sys}. For $CC_{\min}$ all components are flagged such to be $min(CC_{SW}, CC_{HW})$. In this graph the longest path is computed, thus considering any parallelisation gains, and at last supposing the complete communication via shared memory instead of a bus. Similar simplifications apply for the $CC_{\max}$: all components are flagged to be $max(CC_{SW}, CC_{HW})$, these values are accumulated and additionally we assume the complete communication via a bus. To obtain a cost value between 0 and 1 it is necessary to subtract the minimum cycle count from both the system cycle count and the maximum cycle count before computing the quotient.

As mentioned in Section 4.1 the system's current cycle count consists of two parts, the communication time and the process execution time. The calculation of the system's process execution time splits up into two different terms, see Eq. (3), for the total number of N processes, as the goal is a bipartite graph. I.e. a process is executed either on a DSP or on its own dedicated hw accelerator: the cycle count of the sw partition, which is the subgraph of all nodes flagged to be sw (SG_{SW}), is simply the sum of the CC_{SW} of its nodes; whereas the cycle count of the hw partition is defined by the hw cycle count of the longest path in the hw subgraph (SG_{HW}) using Eq. (1) only substituting the BB_i with processes P_i. Have a look at the following example: the bipartite graph in Figure 5 is transformed into a graph only consisting of hw processes.

$$CC_{sys} = PW_{CC_{HW}} + \sum_{\substack{i=1 \\ P_i \in SG_{SW}}}^{N} CC_{SW}(P_i) \qquad (3)$$

The new graph then mirrors the dependencies between the

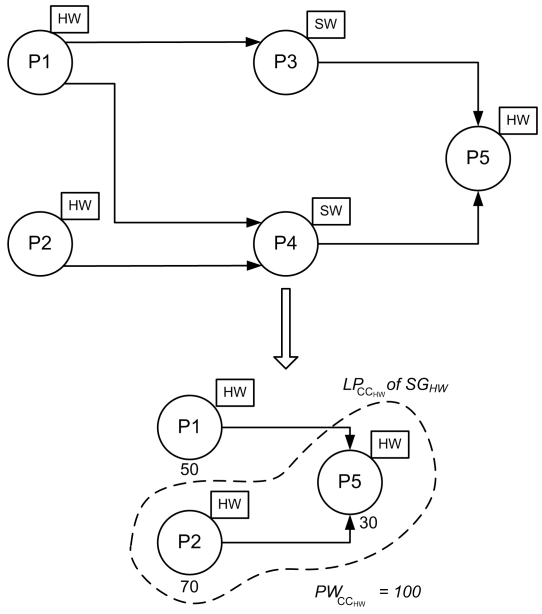

Figure 5: *Transforming bipartite graph into hw graph to calculate maximum CC_{HW} on process level.*

processes. To obtain the system's cycle count for the hw partition the longest path with respect to the CC_{HW} is of interest. In this example the CC_{HW} of each process is located below the process nodes. The longest path in this example is made up of process P_2 followed by P_5. Hence time gains induced by parallelisation effects are taken into account.

By means of the factors $K_{GC}, 0 \leq K_{GC} \leq 1$ and $K_{CC} = (1 - K_{GC})$ the weight of each part can be balanced by the designer. There exist many versions for cost functions, for instance with dynamic weight factors or treshold clipping [3,17], but in order to minimise hardware effort as well as execution time the selected cost function is sufficient.

4.3 Extended Kernighan/Lin Heuristic

The classical KL heuristic [18] seeks to improve a given two-way graph partition by reducing the edges crossing between parts, known as the cut. The essence of the heuristic is its simple yet powerful control strategy, which overcomes many local minima without using excessive moves. The control strategy can be summarised as follows. Starting from an initial partition, the cost of all possible moves of unlocked nodes is measured, then moves the node with the best gain (greatest cost decrease or least cost increase) and flags the node as locked. If all nodes are locked, the heuristic goes back to the lowest- cost partition seen, thus completing one iteration. The heuristic terminates when several consecutive iterations (usually less than three) have not decreased the cost. Three modifications are introduced to enable the heuristic for fast hw/sw partitioning; the first two have been proposed in [19].

- Redefinition of a move as a single object move, rather than a swap, since the goal of the heuristic is not necessarily a balanced partition.
- Application of a powerful data structure, further on referred to as change list that permits selection of the next move in constant time.
- Replacement of the cut metric by the cost function presented above.

A short overview of the implemented algorithm is listed in form of pseudo code in Listing 1. After defining the initial partition in line 1 and evaluating its current cost, the KL heuristic starts in line 2. As long as as the termination criterion is not reached, the next iteration begins. Function *initChangeList*() in line 4 moves all nodes to their opposite partition, calculates the potential change in cost caused by this move, stores it in a single linked list in ascending order and then moves the node back again. Now the preparation of the inner loop is accomplished and the actual moving of nodes can take place (line 9): according to the order of the nodes in the change list, beginning with the one with the best gain, each node is moved to the opposite partition, as long as the change list is not empty (i.e. unlocked nodes exist). In function *moveNextBestNode*() (line 9 and line 22ff) the essential tasks are performed. In line 24 the node is popped from the change list (i.e. the node is 'locked'), then it is moved (line 25) and the current cost is modified by the change value of this move (line 26). Consequently the change list has to be updated in line 28 for all nodes, which are adjacent to the just moved node, since the communication part of the execution time is dependent on the assignment of connected nodes to different partitions, see Section 4.1. After updating the change list entries for adjacent nodes, the function *moveNextBestNode*() returns. In line 11ff the best partition is stored and the termination criterion is modified. When all nodes have been moved once and at least one improvement in cost has been found, the best partition seen is chosen as starting point and the next iteration starts.

Listing 1: *Overview of the implemented Kernighan/Lin algorithm*

```
 0 KL_Heuristic() {
 1   initPartition(); // generates initial partition
 2     while (!KL_Termination) // terminate after T iterations
 3     {                       // without improvement
 4       initChangeList(); // moves all nodes in chosen parti-
 5                         // tion and generates entries in
 6                         // ChangeList
 7       while(ChangeListNotEmpty())
 8       {
 9         moveNextBestNode(); // Core function see line 210
10
11         if (newCost < bestCost) {
12           storeBestPartition();
13           updateKL_Termination(true);
14         else updateKL_Termination(false);
15       }
16     }
17     // start from the very best partition again
18     initPartition(bestPartition());
19   }
20 }
21 /*********************************************************/
22 moveNextBestNode() {
23   // pops the node with the best change value
24   Node = popFirstNodeFromChangeList();
25   moveNode(Node);
26   updateCost(Node->ChangeValue); // modifies newCost by the
27                                  // node's change value
28   updateChangeList(Node); // modifies ChangeList entries
29                           // for all nodes adjacent to
30 }                         // 'Node'
```

5. INTEGRATION INTO SSD

In order to make the design process consistent by providing a unified design environment but at the same time allow for various EDA tools as they are being favoured by the various design teams, a so-called single system description (SSD) has been established in form of a design database [5, 20, 21]. Figure 6 depicts the SSD encircled by a selection of EDA tools and self-developed optimisation libraries. Dedicated interfaces facilitate the seamless communication between the heterogeneous components and the SSD. Automated hw/sw

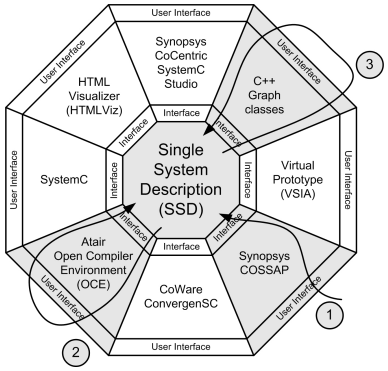

Figure 6: *Single System Description.*

partitioning is performed in the following steps. At first an algorithmic description of a UMTS design written in COSSAP is processed by the interface tools and imported into the SSD. The second stage includes static code analysis by the Open Compiler Environment (OCE) from ATAIR (www.atair.com). This software allows the construction of the CFG and DFG representation of each algorithm in the design. The SSD is enriched with tables of basic blocks and operations for every process. In the third step C++ graph classes are generated that mirror the internal CFG and DFG of every process in the design as well as the DFG on system level, i.e. the communication between the processes. The underlying C++ graph library handles intermediate optimisations, longest path calculation and finally performs the extended Kernighan/Lin heuristic to partition the design. The partitioning decision is fed back into the SSD and further refinement steps can take place, for instance automatic generation of virtual prototypes [21] or bus architecture exploration with ConvergenSC (www.coware.com).

6. PARTITIONING EXAMPLE

To assert the applicability of the proposed procedure an algorithmic description of an industry-designed UMTS receiver component, the Delay Profile Estimator (DPE) has been processed by the presented framework. The DPE consists of

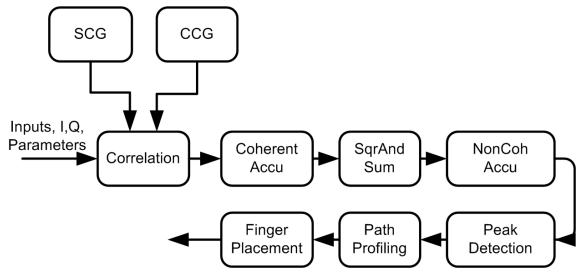

Figure 7: *Block diagram of the Delay Profile Estimator (DPE).*

nine functional blocks as shown in the Figure 7, which look for the arrival time of a signal transmitted over different paths. Two blocks generate the scrambling (SCG) and channelisation codes (CCG). The received discrete signal I/Q is correlated ($Correlation$) with the outputs of the generators and accumulated ($CoherentAccu$, $SqrAndSum$). The $NonCoherentAccu$ block removes spurious peaks out of the data stream. After that peaks are detected ($PeakDetection$) by checking against a threshold. The results are fed into the $PathProfiling$ and the correct placement of the fingers is accomplished ($FingerPlacement$). The original description was available as COSSAP project. After importing of the DPE project into the SSD and static code analysis the characteristics for each of the blocks has been determined. The estimated high-level metrics for the gate counts and the required number of cycles to compute one frame are listed in Table 3.

Function	GC_{op}	GC_{reg}	GC	CC_{HW}	CC_{SW}
SCG	4,800	3,360	8,160	55,297	311,049
CCG	90	63	153	692	865
Correlation	9,760	6,832	16,592	100,645	146,631
CoherentAccu	24,855	17,398	42,253	251,999	353,173
SqrAndSum	2,761	924	3,685	14,752	24,970
NonCohAccu	377	263	640	3,854	4,817
PeakDetection	659	352	1,011	5,637	12,044
PathProfiling	367	257	624	3,676	4,853
FingerPlacement	98	67	166	529	677
Sum	43,770	29,521	73,291	437,083	859,082

Table 3: *Derived high level metrics for the DPE.*

The cost function described in Section 4 has been used with the factors $K_{CC} = 0.5$, $K_{GC} = 0.5$, $GC_{\max} = 73,291$, $CC_{\min} = 436,391$, and $CC_{\max} = 1,367,501$. For the communication a bus system with a bit width of 32 has been assumed, where each transfer takes two cycles. We partitioned this system with the presented Kernighan/Lin heuristic. Results for the derived hw/sw split for the modules of the DPE is given in Table 4 together with the partitioning done by an experienced designer as it is used in an industrial project. With modification of the weight factors $K_{GC} = 0.31$

	Automated		*Manual*	
Function	*HW*	*SW*	*HW*	*SW*
SCG	x		x	
CCG	x			x
Correlation	x		x	
CoherentAccu	x		x	
SqrAndSum	x		x	
NonCoherentAccu	x			x
PeakDetection		x		x
PathProfiling		x		x
FingerPlacement		x		x

Table 4: *Automated and manual partitioning of the DPE.*

and $K_{CC} = 0.69$ it is possible to achieve the same partitioning as it is proposed by the design team. Note that the applied high level metrics may deviate from the real values up to 40%. The prominent advantage of this approach lies in the fact that the suggested procedure to partition a system is full-automated and despite of the complex graph algorithms still applicable with respect to processing time. Table

	Number of blocks	SCA/s	KLH/s
DPE	9	2.56	1.53
ED 1	18	91.99	21.55
ED 2	36	1026.41	393.43

Table 5: *Processing time for the partitioning of different designs.*

5 shows calculation time for static code analysis (SCA) and the Kernighan/Lin heuristic (KLH) for the DPE example and two example designs (ED), which have been generated to test the performance of the chosen approach.

7. CONCLUSIONS

Increasing complexity of systems force the need of automatically derived design decisions at an early stage. One of these important decisions is the partitioning of the system into hw and sw. We showed how hw/sw partitioning can be obtained automatically by means of an advanced framework, the Single System Description, which integrates the extraction of high level metrics by static code analysis and graph partitioning using an extended Kernighan/Lin heuristic. Generation of the metrics and partitioning have been adopted to a Delay Profile Estimator, which is a component of a UMTS baseband processing unit, to prove the concept for the usage in industrial projects.

Future work has to include accuracy improvements of the used high level metrics as well as the consideration of pipelined or block based data processing. An important extension to enhance the applicability of this approach is the integration of various heuristics and different versions of cost functions in order to allow the designer to select the appropriate ones for his design goal.

REFERENCES

[1] N.P. Pitsianis. A Kronecker Compiler for fast transform algorithms. In *8th SIAM Conf. Parallel Proc. For Sci. Comp.*, March, 1997.

[2] A. Kalavade and E.A. Lee. The Extended Partitioning Problem: Hardware/Software Mapping and Implementation-Bin Selection. In *Proc. of Sixth International Workshop on Rapid Systems Prototyping*, pages 12–18, June 1995.

[3] F. Vahid and T. Dm Le. Extending the Kernighan/Lin Heuristic for Hardware and Software Functional Partitioning. *Design Automation for Embedded Systems Journal*, (2):237–261, 1997.

[4] T. Wiangtong, P.Y.K. Cheung, and W. Luk. Comparing Three Heuristic Search Methods for Functional Partitioning in Hardware-Software Codesign. *Design Automation for Embedded Systems*, 6:425–449, 2002.

[5] P. Belanović, M. Holzer, D. Mičušík, and M. Rupp. Design Methodology of Signal Processing Algorithms in Wireless Systems. In *International Conference on Computer, Communication and Control Technologies CCCT'03*, pages 288–291, July 2003.

[6] D. Gajski, N. Dutt, A. Wu, and S. Lin. *High-Level Synthesis: introduction to chip and system design.* Kluwer Academic Publishers, 1992.

[7] J.M. Rabaey and M. Potkonjak. Estimating Implementation Bounds for Real Time DSP Application Specific Circuits. In *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, volume 13 of *6*, pages 669–683, June 1994.

[8] K.M. Büyüksahin and F.N. Najm. High-Level Area Estimation. In *ISLPED'02*, pages 271–274, August 2002.

[9] P. Bjuréus, M. Millberg, and A. Jantsch. FPGA resource and timing estimation from Matlab execution traces. In *Proceedings of the tenth international symposium on Hardware/software codesign*, pages 31–36, 2002.

[10] P. Giusto, G. Martin, and E. Harcourt. Reliable estimation of execution time of embedded software. In *Proceedings of the conference on Design, automation and test in Europe*, pages 580–589, 2001.

[11] R. Ernst, J. Henkel, and T. Benner. Hardware-software Cosynthesis for Microcontrollers. *IEEE Design & Test*, 12:64–75, 1993.

[12] D. Henkel, J. Herrman, and R. Ernst. An Approach to the Adaption of Estimated Cost Parameters in the COSYMA System. In *Third International Workshop on Hardware/Software Codesign*, pages 100–107, 1994.

[13] R.K. Gupta, C. Coelho, and G. De Micheli. Synthesis and Simulation of Digital Systems Containing Interacting Hardware and Software Components. In *29th ACM, IEEE Design Automation Conference*, pages 225–230, 1992.

[14] F. Vahid, J. Gong, and D. Gajski. A Binary-Constraint Search Algorithm for Minimizing Hardware during Hardware/Software Partitioning. In *European Design Automation Conference (EURO-DAC)*, pages 214–219, 1994.

[15] A. Kalavade and E.A. Lee. A Global Critically/Local Phase Driven Algorithm for the Constrained Hardware/-Software Partitioning Problem. In *Third International Workshop on Rapid Systems Prototyping*, pages 42–48, 1994.

[16] J.G. D'Ambrosio and X. (Sharon) Hu. Configuration-level hardware/software partitioning for real-time embedded systems. In *Proceedings of the 3rd international workshop on Hardware/software co-design*, pages 34–41, 1994.

[17] J. Henkel and R. Ernst. High-Level Estimation Techniques for Usage in Hardware/Software Co-Design. In *IEEE/ACM Proc. of Asia and South Pacific Design Automation Conference*, pages 353–360, February 1998.

[18] B. Kernighan and S. Lin. An efficient heuristic procedure in partitioning graphs. *Bell System Technical Journal*, February 1970.

[19] C. Fiduccia and R. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the Design Automation Conference*, pages 175–181, 1982.

[20] M. Holzer, P. Belanović, B. Knerr, and M. Rupp. Design Methodology for Signal Processing in Wireless Systems. In *Beiträge der Informationstagung Mikroelektronik 2003*, volume 33, pages 303–306, October 2003.

[21] B. Knerr, M. Holzer, P. Belanović, G. Sauzon, and M. Rupp. Advanced UMTS Receiver Chip Design Using Virtual Prototyping. In *International Symposium on Signals, Systems, and Electronics (ISSSE)*, August 2004.

[22] R. Enzler, T. Jeger, D. Cottet, and G. Tröstler. High-Level Area and Performance Estimation of Hardware Building Blocks on FPGAs. In *R.W. Hartenstein and H. Grünbacher (Eds.) FPL 2000*, volume 1896, pages 525–534. Springer, 2000.