Task Scheduling for Power Optimisation of Multi Frequency Synchronous Data Flow Graphs *

Bastian Knerr
Institute for Communications and RF Engineering
University of Technology
Vienna, Austria
bknerr@nt.tuwien.ac.at

Martin Holzer
Institute for Communications and RF Engineering
University of Technology
Vienna, Austria
mholzer@nt.tuwien.ac.at

Markus Rupp
Institute for Communications and RF Engineering
University of Technology
Vienna, Austria
mrupp@nt.tuwien.ac.at

ABSTRACT

During recent years power optimisation has become one of the most challenging design goals in modern communication systems, particularly in the wireless domain. Many different approaches for task scheduling on single or multi-core systems exist, mostly addressing the minimisation of execution time or the number of processors used. The minimisation of the processor's clock frequency by adjusting the supply voltage or directly by frequency scaling according to the chosen task scheduling has shown good results in the reduction of power consumption. Most of the known approaches base their core algorithms on graph representations for multirate systems or synchronous data flow (SDF) graphs, in a single frequency domain. In many cases a signal processing system comprises several frequency domains, in which processes have to be fired according to their in- and output data rates as well as to their frequency domain. In this work the superposition of frequency domains and data dependencies is incorporated into the optimisation process and used as a another degree of freedom. Several algorithms have been implemented and evaluated to minimise the required processor's clock frequency, including a greedy, a simulated annealing, as well as a tabu search approach.

Categories and Subject Descriptors

I.6 [**Simulation and Modeling**]: Applications

General Terms

Algorithms, Design

Keywords

Task Scheduling, Power Optimisation, Frequency Scaling, Multi Frequency Systems, Synchronous Data Flow Graphs

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

SBCCI'05, September 4–7, 2005, Florianópolis, Brazil.

1. INTRODUCTION

The exponential growth in the class of portable systems, such as mobile phones, handhelds and personal computing devices, demands both high computational performance and low power consumption. Due to the fact that the acquisitions of battery technology have not kept up with the semiconductor industry, minimising power consumption became the primary design goal. Such systems also require high design flexibility, which results in the need for implementation on programmable processor platforms. In fact, embedded software running on RISC or digital signal processor (DSP) cores has emerged as a leading implementation methodology for such applications as speech coding, modem functionality, video compression, and communication protocol processing [6].

The average power consumption of CMOS circuits is given by $P \propto CV_{DD}{}^2 f$, where C is the collective switching capacitance, V_{DD} is the supply voltage and f is the common switching frequency [8]. Dynamic voltage or frequency scaling techniques are effective ways to reduce the CPU power. With both methods a trade-off between performance and power has to be considered. Lowering the frequency immediately affects the computation speed, whereas reducing the supply voltage increases the circuit delay and is often coupled with lower clock frequencies. Hence both ideas have to deal with longer execution times of their respective tasks and its potential violation of timing constraints.

The proposed power optimisation engine addresses a customary SoC, consisting of a DSP capable either of variable voltage supply or of clock frequency scaling, a number of HW accelerators, a system RAM and one or more system busses. The power dissipation of the components is in general very much affected by the applications running on the system. But since a predominant contingent of a SoC's overall power consumption is always spent for the DSP core [10, 5], we concentrate our endeavours on the minimisation of the core's power consumption.

To enlighten the location of the discussed optimisation method within the design flow, a brief description of the general development stages is given: the design process of embedded systems usually starts at the so-called algorithmic level where the initial concept of the system is turned into an executable model and high-level specifications of the system are verified [21]. Development at this stage of the design process is aided by such tools as MathWorks Matlab, CoWare SPW, or Synopsys CoCentric System Studio. The design space exploration, i.e. architecture selection in-

teracting with hw/sw partitioning [16] and floating-point to fixed-point conversion [4], has decomposed the algorithmic description into a process graph, analysed interprocess communication and flagged the processes to be implemented preferably in either HW or SW. At the backend of this procedure, when the target architecture with one or even more processor cores is the favoured solution, the processor selection starts. At this stage parts of the design, i.e. subgraphs of the process graph, are selected to get dedicated HW accelerators, others are flagged to run on a DSP or a general purpose processor. Now the selected SW processes have to be mapped onto one or more DSP candidates, while fulfilling all the constraints, such as data rate dependencies and execution time, and simultaneously minimising power consumption. A number of approaches has been published to solve this problem optimally. The following section demonstrates some of these methods and clarifies the contribution of our work in this field. The rest of the paper is organised as follows. Sec. 2 describes the decomposition of a signal processing system into multi-frequency domain graphs and the resulting constraints. It is followed by Sec. 3 exposing a new problem abstraction and the implemented minimisation algorithms. Results of the proposed methods are shown in Sec. 4 with several randomly generated graphs and the application to an industry-designed xDSL device. Finally, conclusions of the paper are drawn in Sec. 5.

1.1 Related Work

Early suggestions to dynamically adapt the supply voltage to temperature and process parameters were made in [19], and the approach in [20] considered data-dependent computation times in self-timed circuits. The idea of data-dependent computation times, as it occurs often in signal processing, was enhanced in [7] towards synchronously clocked circuits. When *average throughput* is the metric of performance, buffering of data and averaging processing rate yields significant power reduction, but increases latency [11]. Because of the averaging these approaches cannot cope with strong task interdependencies and hard timing requirements.

Voltage scaling techniques considering static or dynamic interprocess dependencies could be found in [13, 22]. They act on operating system level via SW interfaces and use either prior knowledge of the application's processor load (static) or produce predictions for the application's demands based on load measurements in the past (dynamic). The dynamic scheduling approaches are very much suited for complex applications handling vast control structures, interrupts, and user interaction, which hinder an a priori scheduling. The static approaches use complete application knowledge, to trigger fixed process schedules according to certain system states, which is appropriate for more data flow oriented signal processing on DSPs.

A very similar approach with respect to the basic idea of the time interval spreading is the work presented in [23]. Processing time is split into small time intervals of 10 to 50 ms and the task scheduler modifies the clock frequency according to the processor load of the preceding time interval. The power saving is reached by spreading busy cycles over a time span, in which the processor would have been in idle mode. This is a dynamic approach as well, whereas we focus on signal processing systems with a set of pre-known execution modes, for all of which we create minimum power schedules, while satisfying all timing constraints.

2. PRELIMINARIES

As stated before this optimisation method is an integral part of the design space exploration process to find a task schedule with the minimum power consumption while satisfying all timing constraints. Thus the first step is to analyse and decompose the system description into an SDF graph and assigning the timing constraints to the processes. The next step is the extension of the graph towards the concept of multi-frequency domains and its formulation into further constraints.

2.1 Digital Signal Processing Systems as SDF Graphs

The concept of SDF graphs for signal processing systems was developed and used extensively by Lee and Messerschmitt [18] and later by Zivojnovic [24]. This representation accomplished the backbone of renowned signal processing work suites, e.g. Ptolemy [17] or SPW [3]. It captures precisely multiple invocations of processes and their data dependencies, thus it is most suitable to serve as a system model, in which process computation time as well as interprocess communication time is considered. In Fig. 1 a

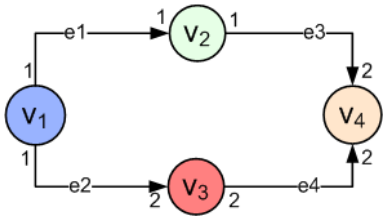

Figure 1: ***Simple example of a synchronous data flow graph.***

simple example of an SDF graph $G = (V, E)$ is depicted, assembled by a set of vertices $V = \{v_1, .., v_4\}$ and a set of edges $E = \{e_1, .., e_4\}$. The numbers on the tail of each edge e_i represent the number of samples produced per invocation of the vertex at the edge's tail. The numbers on the head of each edge indicate the number of samples consumed per invocation of the vertex at the edge's head. The decomposition of the system under investigation into an SDF graph depends heavily on the chosen level of granularity. We assume a vertex to be a sequential code block, also referred to as process, like a distinct matched filter function (e.g. DCT, FIR). As our focus lies on processes completely running on a DSP, the interprocess communication is realised as *read* or *write* instructions via shared memory and hence can be added to the process' computation time.

All the following considerations take place in the SDFG representation of the system. Note that the decomposition of a system desciption into an SDF graph representation has been fully automated within our overall design framework for algorithmic descriptions developed in the COSSAP environment [15] and for general SystemC designs [12], which enables its feasibility for Synopsys CCSS as well.

2.2 Multi-Frequency Domain Graphs

Many digital signal processing systems contain more than one frequency domain. For instance a typical digital transceiver is composed of processes running at a high carrier frequency, usually more than one intermediate frequency,

and a baseband processing frequency [9]. Those intertwining domains and the additional constraints brought into the scheduling problem are presented in the remaining section. Fig. 2*a* illustrates a small example of an SDF Graph, running on a DSP, with two frequency domains. Every single process is annotated with its *cycle count*. The two domains are connected by a *Decimation* and an *Hold* block handling the different sample rates. The processes on $256kHz$ have hard real time constraints, i.e. A has to consume one sample **exactly** at a rate of $f_1 = 256kHz$, for instance reading from a register, which the antenna subsystem (peripheral) writes to. The last process C in this domain has to deliver one sample with f_1, as the antenna subsystem expects a new value to appear at this rate. The processes X and Z of the second domain run at a frequency of $f_2 = 128kHz$, but they are allowed to be scheduled freely within their domain. Consequently the SDFG in the $128kHz$ domain has to be executed once within a time window of $t_2 = 1/f_2 = 7.82\mu s$. According to the periodicity of the schedule one can abstract the different sample rates f_1 and f_2 as time *bins*. Note that the length of the processes is measured in *cycle counts*, and the processor's clock frequency cf determines the maximum number of *cycle counts* within such a *bin*. Assume a processor offers two frequencies of $cf_1 = 16MHz$ and $cf_2 = 32MHz$. cf_1 would allow for a *bin* size $cf_1/f_1 = 16MHz/256kHz = 62$ cycles calculated for the $256kHz$ domain; analogously $32Mhz$ would allow for 125 cycles being performed. Now we observe a superposition of timing constraints. It is not ensured that a schedule satisfying the interprocess data dependencies, i.e. the process ordering, is also valid according to the sample rate constraint. Fig. 2*b* depicts a schedules with a processor offering a *perfect* clock frequency, such that the system's overall *cycle count* of 117 cycles could be computed within $7.82\mu s$. In Fig. 2*b*, the second activation of process A violates its hard real time constraint. As A has to be fired at exactly $256kHz$ a simple reordering of the processes doesn't solve the problem. In Fig. 2*c* the next higher clock frequency has to be chosen. In this example at least the second *bin* is completely filled. Note, normally the clock frequency scaling does not enable such a fine grained control.

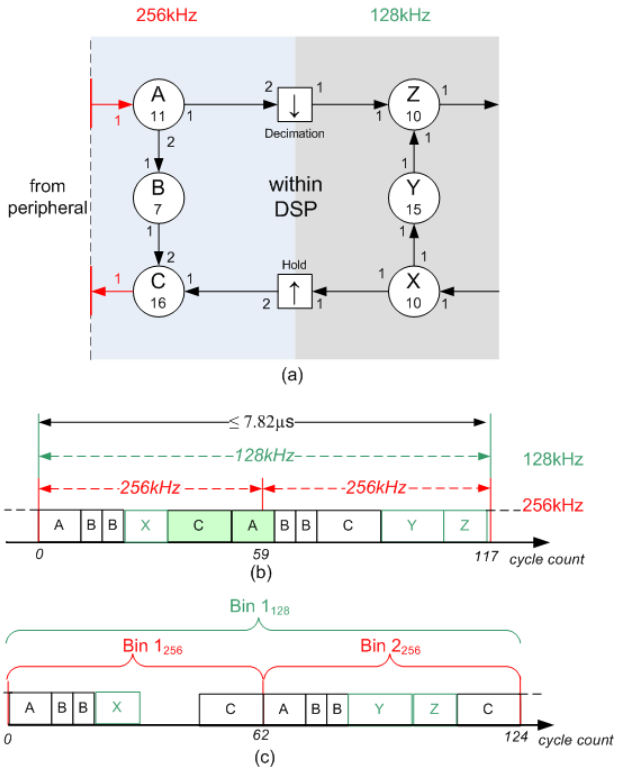

Figure 2: ***Multi-Frequency Domain Graph (a), invalid (b) and valid schedule (c).***

rate constraint. Fig. 2*b* depicts a schedules with a processor offering a *perfect* clock frequency, such that the system's overall *cycle count* of 117 cycles could be computed within $7.82\mu s$. In Fig. 2*b*, the second activation of process A violates its hard real time constraint. As A has to be fired at exactly $256kHz$ a simple reordering of the processes doesn't solve the problem. In Fig. 2*c* the next higher clock frequency has to be chosen. In this example at least the second *bin* is completely filled. Note, normally the clock frequency scaling does not enable such a fine grained control.

In this simple example the selection of a new clock frequency and a refilling into the *bins* is rather trivial. But when we consider the fact that realistic multi-frequency graphs have more than two domains and, more importantly, every domain easily consists of up to twenty processes with several activations, this task becomes extraordinarily complex.

As said before it is realistic to choose a clock, which is higher than the optimal one, since customary DSPs offer a stepwise adjustment of their frequency. Examples are Intel's 80200 XScale [1] or the Transmeta Crusoe SE family [2]. In other words the available number of cycle counts within every time *bin* yields a degree of freedom to move the contained processes as there is more time, i.e. cycles, than needed. A more detailed description will be given in the following section.

3. OPTIMISATION ALGORITHMS

Before the algorithms are examined in detail, the problem abstraction into the *Bin Packing* terminology will be briefly introduced. As shown in Sec. 2.2 the existence of the strict real time constraints of one or more processes leads to a situation, in which the time spectrum is divided into time *bins*. These have to be filled with processes according to their frequency domain as well as to their SDF graph dependency, as it is illustrated in Fig. 3*a*. In this example there are three frequency domains coupled by factor two, thus yielding four time *bins*. The critical constraint is here given by process A, which has to be the first process in every *bin* obeying a sample rate of $256kHz$. The initial schedule is generated as follows (Fig. 3 *b*): the distribution of the processes A,B,C is trivial (step 1). Every *bin* has to contain exactly once the schedule list of the SDFG of the highest frequency domain. For further consideration this part of the schedule is static and therefore drawn patterned. Step 2

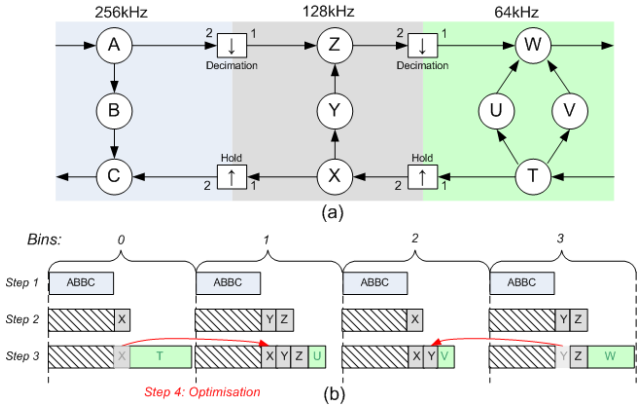

Figure 3: ***Multi frequency SDF Graph (a) and its abstraction into bin filling(b).***

distributes the invocations of the processes within the middle frequency domain over their respective *bin ranges*, the first invocation of *X,Y,Z* in 0 to 1 and the second invocation in 2 to 3. The third step repeats the procedure for the lowest frequency domain over the *bin range* = 0..3. These steps serve as the preparation for the last optimisation step 4, which moves processes from *bin* to *bin* to find a better solution. This solution consists of a *packing* of processes into *bins*, such that the *bin* size of the most filled *bin*, BS_{max}, is minimised. Once this solution BS_{max} has been found, the minimum required clock frequency cf_{min} is computable by $cf_{min} = BS_{max} \times f_{max}$, where f_{max} is the maximum occurring frequency, here $256kHz$. As stated in Sec. 2.2, the processors do not offer a continuous space frequency scaling. Therefore the next higher available clock frequency of the processor will be selected.

Note that for the following optimisation engines a cost function has been used that does not only consider a lower BS_{max} as better solution, but considers the size of all other *bins* provided that BS_{max} is unaltered. Indeed there exist more promising *bin packing* solutions than others, although featured by the same BS_{max}. Assume a solution S_1 with the maximum *bin* size BS_{max} and the smaller *bin* sizes of **all** other bins are similar on a fair fill level. Contrarily another solution S_2 of the same problem instance contains **some** other *bin* sizes of smaller but rather similar size as BS_{max} and **some** others with much smaller *bin* sizes. Intuitively we observe that solution S_2 is more promising for further optimisation steps, since it is more likely that the processes in the maximum filled *bin* can fit into one of those very little filled. This finer difference between solutions S_i can be formulated into the following target function, which then has to be maximised:

$$t(S) = \frac{1}{N} \sum_{\substack{i=0 \\ BS_i^S \neq BS_{max}}}^{N-1} \left(\frac{BS_i^S}{BS_{max}} \right)^2 \qquad (1)$$

Where S is the solution to be evaluated, N is the number of *bins*, and BS_i^S is the *bin size* of the ith *bin* in solution S. As soon as BS_{max} is diminished by the optimisation, it is set to the new value. In the following sections three different optimisation methods are introduced.

3.1 Greedy Algorithm

The most intuitive approach to improve a given schedule solution is certainly to design an algorithm, which takes every process consecutively into consideration. The algorithm tries to move it to another *bin* and accepts the move, if it yields an improvement in the target function. Fig. 4 depicts a simple but typical situation. On the left an SDFG is shown with four processes and all sample rates equal to one. For these the *bins* $i..i+3$ are potential candidates. Busy cycles occupied by processes of other frequency domains are considered static in this iteration step and therefore greyed. Starting with process A, the algorithm checks if *bin* $i+1$ is valid, i.e. firstly is it a member of A's *bin* range, secondly does *bin* i not contain a successor B,C,D of A? Is $i+1$ valid, the cost function is evaluated. In this case, there is no improvement and *bin* $i+2$ is not valid for A. The next process is B, which will be moved to *bin* i, because i is then better filled than $i+1$ was, while not exceeding BS_{max}. The same will be true for process C, which is moved to i as well. Until now there wasn't any decrease in BS_{max}, but cost improvements regarding the target function. When trying to move process D, all preceding *bins* $i..i+2$ are valid and *bin* $i+1$ is chosen, now yielding a lower BS_{max}. This greedy approach is repeated for all processes in all frequency domains (except the highest one, whose processes are static), until no improvement could be found.

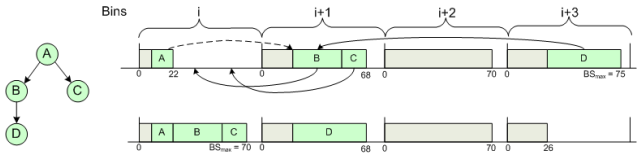

Figure 4: ***Process movement between bins improving target function in equation (1).***

3.2 Tabu Search (*TS*)

TS is a mathematical optimisation method, belonging to the class of local search techniques. In general it enhances the performance of a local search method by using memory structures. A *TS* approach is immediately derivable from the most appalling weakness of the greedy algorithm, which gets easily caught in local optima. Every process activation in the schedule list handles a data structure containing a list of all the *bins*, in which it had been placed during the last iterations - the tabu list. The included *bins* are forbidden for a next move selection. The rather simple mechanism forces processes to leave their last *bin* at least a few times according the parameter adjustment. Thus a control structure is established, which overcomes many local optima. The performance of this approach is very much dependent on the parameter set, such as the initial *tabu list length (tll)* for every process and the adaptive decreasing of the *tll* according to the progress of the algorithm. In the small example in Fig. 4 it is evident that a $tll \geq 3$ doesn't make much sense, as the potential *bin range* is limited to 4. Since the number of potential *bin* candidates for a certain process increases with multiples of the coupling factors between frequency domains, it is reasonable to couple the *tll* to the *bin* number that is available for the currently processed frequency domain. Different sets of parameters have been tested, showing best results regarding to convergence speed in a *tll* being the integral part of one fourth of the available *bin number*. The stepsize to decrement the *tll* for every process has been chosen to $tll_{i+1} = tll_i - 1/2$ per iteration i. By this means the *TS* degenerates to a greedy search. Since the best seen schedule so far has been stored, of course, the decrement mechanism is only applied while the algorithm generates better results, thus leaving enough freedom to return to better solutions and avoiding to converge to minor quality schedules.

3.3 Simulated Annealing (*SA*)

SA is a generic probabilistic heuristic approach for the global optimisation problem, namely locating a good approximation to the global optimum of a given function in a large search space [14]. As the *TS* before, the *SA* control structure avoids to be trapped in local optima by accepting worse solutions with a certain probability. This probability decreases as the algorithms proceeds. As the roots of *SA* lie in metallurgy we use in the following the classical terms like energy E and temperature T. Note that energy E is the

inverse of our target function, because *SA* tries to decrease energy, whereas we want to increase this metric. The probability of making the movement to the new *bin*, i.e. generating a new solution S_{new}, is a probability function $P(\delta E, T)$ of the energy difference $\delta E = E(S_{new}) - E(S_{best})$ between the two solutions, and of a global time-varying parameter called the temperature T. As stated before P is defined non-zero, even if δE is positive, meaning that a solution could be accepted even when it is worse. While the optimisation proceeds, T goes to 0, so that $P(\delta E, T)$ tends to 1 for better and to 0 for worse solutions. Obviously the crucial parameter is T, since it defines the *annealing* strategy, i.e. the decreasing probability of accepting worse solutions. In our case the initial temperature $T_0 = T_{init}$ is chosen quite intuitively to be the first BS_{max}, which is then decreased $T_{i+1} = \alpha T_i$ by the factor $\alpha = 0.9$ per iteration i, forcing the algorithm to converge. For the probability the classical formula is used $P_i = e^{\frac{-\delta E_i}{T_i}}$.

4. RESULTS

In order to demonstrate the viability of employing the power optimisation on realistic DSP designs, as well as to investigate the relative performance of its three optimisation techniques, an industrial DSP design as well as several randomly generated multi-frequency graphs were chosen as a test environment.

The realistic DSP design used in this work is an embedded system xDSL transceiver, which implements ADSL and VDSL on one chip. It consists of four frequency domains from 256kHz down to 8kHz and is comprised of 25 processes that are mainly wave digital filters, FIR filters and MAC processes. For one system run, which is periodically repeated, 131 different process activations are required. 67 of these process activations are **not** in the highest frequency domain and are free to be distributed over up to 32 *bins*.

The randomly generated test designs TD_i are described accordingly by their *number of frequency domains*, *number of bins*, *number of movable process activations*, and a lower bound to the size of their search space. The *cycle counts* of the processes lie between 6 (e.g. MAC) and 45 (e.g. 15 tap FIR). Their internal SDFGs are chosen to be sparse, i.e. the number of edges has the same order of magnitude as the number of vertices/processes ($G = (V, E)$ with $O(|E|) = O(|V|)$), which is in general the case for signal processing systems. Table 1 lists the characteristics of the test designs.

Design	Frequency Domains	Bins	Movable process activations	Search space (Lower Bound)
TD_1	2	8	19	$1.37 * 10^5$
TD_2	3	32	50	$1.55 * 10^{22}$
xDSL	4	32	67	$4.61 * 10^{24}$
TD_3	4	64	120	$1.36 * 10^{63}$
TD_4	5	128	240	$3.39 * 10^{145}$

Table 1: *Test design characteristics.*

The next Table 2 gives the results obtained by applying the three optimisation techniques to the five designs. The rows correspond to the *Greedy*, the *TS* and the *SA* approach, the columns are the designs listed. The elements of the table are the maximal *bin* sizes BS_{max}, which the algorithms found. The last row **Optimum** lists the lower bounds for the optimal solution. This lower bound is calculated by the sum over all process invocations of the respective system divided by its number of bins. As expected the worst

Algorithm	TD_1	TD_2	xDSL	TD_3	TD_4
Greedy	66	179	266	251	438
Tabu Search	57	152	219	206	359
SA	57	152	219	197	351
Optimum (Lower bound)	50	141	204	184	339

Table 2: *Minimum bin sizes for BS_{max}, found by the optimisation algorithms.*

results are obtained by the *Greedy* algorithm. As stated previously its movement mechanism gets trapped immediately when it has to perform *two successive* process moves to improve the target function. Hence it is very sensitive to the initial schedule. In the two biggest examples this has dramatic influence on the minimal *bin* size and hence on the possible power saving. Remember the power consumption increases linearly with the frequency, thus the possible power savings for the test design TD_3 between *Greedy* and *SA* approach due to the minimum frequency is 22%. *TS* and *SA* achieved very much the same performance regarding the BS_{max} result. In the first three designs both returned the same result. The difference lies in their computation time due to their convergence behaviour. Note that for smaller designs the distance to the lower bound is much higher in percentage (14% for SA applied to TD_1) for all algorithms than for the bigger designs (3.5% for SA applied to TD_4). The cause is that the granularity of the process computation times is very coarse in comparison to the lower bound value for small designs.

Algorithm	TD_1	TD_2	xDSL	TD_3	TD_4
Greedy	0.13	0.25	0.41	3.61	9.82
Tabu Search	0.64	4.97	4.48	31.9	145.2
SA	0.89	3.86	4.35	28.8	131.5

Table 3: *Computation time in sec. of the three optimisation methods for all test designs.*

Table 3 lists the computation times in *sec.* for all designs and algorithms. The computation time examination exposes better convergence for the *SA* algorithm for bigger problem instances compared to the *TS*. The worse computation time performance of the *TS* is due to the fact that the local search space for **one** process to move to another *bin* is already highly limited within the presented scenarios, because of the process's frequency **and** data rate dependency. In Figure 5 the typical convergence behaviour is plotted for an instance of the biggest test design TD_4. Especially in the beginning phase of the *TS* a part of the available *bins* for a process is blocked in any case, even when the improvement gain is tremendous. Whereas the probability function of the *SA* approach gives always the possibility to generate better solutions. Certainly the initial *tabu list length* parameter holds responsible for the worse convergence behaviour. A coupling of the *tll* parameter to the *degree of parallelism* in the contained SDF graphs, which is an indicator for the available *bins* with respect to the data rate dependency, requires further investigation.

5. CONCLUSIONS

Task scheduling to reduce power consumption has been a well studied research topic in recent years. Within this work we extended the SDF graph description for signal processing systems with hard real time constraints to multi-frequency

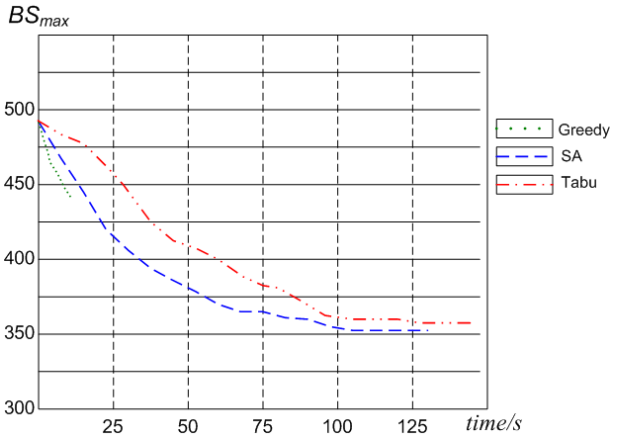

Figure 5: ***Typical convergence behaviour of the algorithm for an instance of the test design*** TD_4.

domain graphs and implemented three different optimisation algorithms to minimise the required clock frequency of the DSP and thus power consumption. The special problem formulation in the *Bin Packing* terminology enables the implementation of algorithms untypical for task scheduling: *Tabu Search* and *Simulated Annealing*.

The best results have been obtained with the *SA* approach, returning minimum schedules with satisfying runtime. *TS* yields comparable results but suffers from impairments regarding its convergence characteristics. The focus of the future work lies on the improvement of the parameter settings of the proposed algorithms, especially for *TS*, as well as on the incorporation of a complex communication model for the interprocess communication of multi-core systems.

6. REFERENCES

[1] Intel 80200 Processor Based on Intel XScale Microarchitecture. http://developer.intel.com/design/iio/manuals/273411.htm.

[2] Transmeta Crusoe SE Processor family. Technical report. http://www.transmeta.com/crusoe/crusoe_se.html.

[3] CoWare SPW 4. CoWare Design Systems, 2004. http://www.coware.com/products/spw4.php.

[4] P. Belanović and M. Rupp. Automated Floating-point to Fixed-point Conversion with the *fixify* Environment. In *International Workshop on Rapid System Prototyping RSP'05*, June 2005.

[5] T. D. Burd and R. W. Brodersen. Processor design for portable systems. *J. VLSI Signal Processing*, 13(2-3):203–221, 1996.

[6] R. Camposano and J. Wilberg. Embedded system design. *Design Automation Embedded Syst.*, 1:5–50, 1996.

[7] A. Chandrakasan, V. Gutnik, and T. Xanthopoulos. Data Driven Signal Processing: An Approach for Energy Efficient Computing. In *Proc. Int. Symp. Low Power Electronics and Design*, pages 344–352, 1996.

[8] A. P. Chandrakasan, S. Sheng, and R. W. Broderson. Low-power CMOS digital design. *IEEE J. Solid-State Circuits*, 27(4):473–484, 1992.

[9] M. Frerking. *Digital Signal Processing in Communication Systems*. Chapman and Hall, 115 Fifth Ave., NY 10003, 1994.

[10] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. *IEEE J. Solid-State Circuit*, 31(9):1277–1284, 1996.

[11] V. Gutnik and A. Chandrakasan. An Efficient Controller for Variable Supply-Voltage Low Power Processing. In *Proc. Symp. VLSI Circuits*, pages 158–159, 1996.

[12] M. Holzer, P. Belanović, D. Mičušík, and M. Rupp. A Consistent Design Methodology to Meet SDR Challenges. In *Wireless World Research Forum WWRF9*, Zurich, July 2003.

[13] T. Ishihara and H. Yasuura. Voltage Scheduling Problem for Dynamically Variable Voltage Processors. In *ISLPED 1998*, pages 197–202, August 1998.

[14] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[15] B. Knerr, M. Holzer, P. Belanović, G. Sauzon, and M. Rupp. Design Flow Improvements for Embedded Wireless Receivers. *XII. European Signal Processing Conference (EUSIPCO 2004)*, September 2004.

[16] B. Knerr, M. Holzer, and M. Rupp. HW/SW Partitioning Using High Level Metrics. *International Conference on Computing, Communications and Control Technologies (CCCT)*, pages 33–38, June 2004.

[17] E. Lee. Overview of the Ptolemy Project. Technical report, University of Berkeley, March 2001.

[18] E. Lee and D. Messerschmitt. Static scheduling of synchronous data-flow programs for digital signal processing. *IEEE Transactions on Computers*, 36:24–35, 1987.

[19] P. Macken, M. Degrauwe, M. V. Paemel, and H. Oguey. A Voltage Reduction Technique for Digital Systems. In *Proc. IEEE Int. Solid-State Circuits Conference*, pages 238–239, 1990.

[20] L. S. Nielsen, C. Niessen, J. Sparso, and K. van Berkel. Low-power Operation Using Self-timed Circuits and Adaptive Scaling of the Supply Voltage. *IEEE Trans. VLSI Syst.*, 2(4):391–397, 1994.

[21] M. Rupp, A. Burg, and E. Beck. Rapid Prototyping for Wireless Designs: the Five-Ones Approach. *Signal Processing Europe 2003*, 83:1427–1444, July 2003.

[22] Y. Shin, K. Choi, , and T. Sakurai. Power Optimization of Realtime Embedded Systems on Variable Speed Processors. In *Intl Conference on Computer Aided Design (ICCAD)*, pages 365–368, November 2000.

[23] M. Weiser, B. Welch, and A. D. S. Shenker. Scheduling for Reduced CPU Energy. In *USENIX Symp. Operating Systems Design and Implementation*, pages 13–23, 1994.

[24] V. Zivojnovic, S. Ritz, and H. Meyr. Optimizing DSP Programs under the Multirate Retiming Transformation. In *7th European Signal Processing Conference*, volume 3, pages 1597–1600, 1994.