UNIVERSITY OF SOUTHAMPTON

Faculty of Engineering, Science and Mathematics

School of Electronics and Computer Science

Investigation into Energy-Efficient Co-Synthesis of Distributed Embedded Systems

by

Dong Wu

Thesis for the degree of Doctor of Philosophy

December 2004

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

INVESTIGATION INTO ENERGY-EFFICIENT CO-SYNTHESIS OF DISTRIBUTED EMBEDDED SYSTEMS

by Dong Wu

Energy dissipation is an important performance parameter for portable embedded systems. This thesis focuses on minimising power consumption (dynamic and leakage) at the system level of the design flow. Special emphasis is placed upon developing co-synthesis techniques (mapping, scheduling, and energy management) for systems that contain processing elements which can trade off between performance and power consumption during run-time by employing dynamic voltage scaling (DVS) and adaptive body biasing (ABB).

The first part of the thesis addresses dynamic power minimisation for data and control dominated embedded systems. A novel conditional behaviour-aware DVS (CBADVS) targeting embedded systems expressed as conditional task graphs has been proposed. The CBADVS exploits the slack time taking into account the conditional behaviour of the application, so that energy dissipation is reduced and, at the same time, timing feasibility is guaranteed for all possible condition values. Furthermore, a genetic algorithm based mapping is introduced to optimise the system implementation towards effective exploitation of the proposed CBADVS, hence, leading to further energy reduction. The technique has been validated extensively including a real-life example of vehicle cruise controller, and it has been shown that up to 42% energy saving is achieved with 5 seconds computational time, and with no penalty in meeting real-time constraints.

The second part of the thesis addresses the impact of communications on dynamic power minimisation in data and control dominated systems design. It is shown how the concept of enhanced system model, which captures the time and power costs of communications, allows the design of energy-efficient embedded systems by integrating communications with the above CBADVS based co-synthesis technique. The computational time of the communication-integrated co-synthesis technique has been reduced by analysing the deficiency of concurrent task and communication mapping, and decoupling communication from task mapping. A large number of experiments have been used to investigate the effect of alternative communication architectures on system quality in terms of energy efficiency.

The co-synthesis techniques of the first and second parts have focused on dynamic power reduction. The final part of the thesis presents a power-composition profile aware co-synthesis technique to reduce dynamic power as well as leakage power. In particular, the proposed technique performs a power management selection at the architectural level, with the aim of achieving high energy saving at a reduced implementation cost. Furthermore, the proposed technique performs mapping, scheduling and voltage scaling (DVS and/or ABB) for applications specified as task graphs with timing constraints. The technique has been validated using extensive experiments including a real-life GSM voice CODEC example.

Table of Contents

List of Figures

List of Tables

Acknowledgements

First of all I would like to express my gratitude to my supervisor, Professor Bashir M. Al-Hashimi, who has provided me constant guidance and support needed to complete the work presented in this thesis. I am very thankful to him for teaching me how to conduct research and how to clearly organise my ideas and write them into technical papers.

I would like to thank the members of the Electronics Systems Design Group at the School of Electronics and Computer Science of the University of Southampton. I wish to acknowledge Dr Tom J Kazmierski for valuable feedback during my MPhil to PhD transfer examination, as well as Dr Paul H Chappell for his kind support during the early stage of my PhD study. I am thankful to Marcus Schmitz, Theo Gonciari, Paul Rosinger, Mauricio Varea, Reuben Wilcock, Manoj Gaur and Yan Xie for their friendship and providing appreciated help on several occasions.

I would like to acknowledge Professor Petru Eles from Linkoping University (Sweden) for excellent comments and providing benchmark sets. I am also very grateful to the School of Electronics and Computer Science of the University of Southampton for the financial support that made this investigation possible.

My parents and my sister have been giving unconditional love and unlimited support. No words would be enough to present my gratitude to them. Thus I dedicate this thesis to them.

Abbreviations

ABB	Adaptive body biasing
ABS	Anti blocking system
ADC	Analog-to-digital converter
AET	Actual execution time
ALU	Arithmetic-logic unit
ASIC	Application specific integrated circuit
CAN	Controller area network
CBADVS	Conditional behaviour-aware DVS
CEM	Central electronic module
CL	Communication link
CMOS	Complementary metal-oxide semiconductor
CPU	Central processing unit
CTG	Conditional task graph
DAC	Digital-to-analog converter
DPM	Dynamic power management
DSP	Digital signal processor
DVS	Dynamic voltage scaling
ECM	Engine control module
ETM	Electronic throttle module
FPGA	Field programmable gate array
FSM	Finite state machine
GA	Genetic algorithm
GCC	GNU compiler collection
GPP	General-purpose processor
GSM	Global system for mobile telecommunication
HDL	Hardware description language
IC	Integrated circuit
I/O	Input/output
LET	Latest end time

LTP	Long term prediction
MOSFET	Metal-oxide-semiconductor field effect transistor
NMOS	N-type metal-oxide-semiconductor
PDA	Personal digital assistant
PCP	Power-composition profile
PE	Processing element
PMOS	P-type metal-oxide-semiconductor
PM	Power management
PMS	Power management selection
PSM	Program-state machine
RPE	Regular pulse excited
RTL	Register-transfer level
SRAM	Static random access memory
TCM	Transmission control module
TG	Task graph
VCC	Vehicle cruise controller
VCO	Voltage controlled oscillator
VHDL	VHSIC (Very high speed integrated circuit) hardware description language
VLSI	Very large scale integration
WCET	Worst case execution time

Chapter 1

Introduction

Over the last several years, energy efficiency has become an essential issue in the design of embedded systems and there are two major reasons. Firstly, the use of portable systems, such as mobile phones, personal digital assistants (PDAs), and digital cameras have significantly increased. These products are battery-powered and their battery operation time is one of the most critical performance measures. For such embedded systems energy efficiency is a primary design goal. Secondly, the continually growing density and operational frequency of integrated circuits causes higher power consumption. Power consumption implies heat dissipation, which directly influences the reliability, packaging and cooling costs of the systems. Thus energy efficiency benefits all types of digital systems in terms of reliability, packaging and cooling costs.

This thesis investigates design techniques for energy-efficient embedded systems, which have the potential to overcome traditional design techniques that neglect important energy management issues. In this context, special attention is placed upon the employment of the following two techniques: dynamic voltage scaling (DVS) and adaptive body biasing (ABB). The basic idea of DVS and ABB techniques is to dynamically scale the operational frequency of digital circuits during run-time in accordance to the temporal performance requirements of the application, with the aim of reducing energy dissipation whilst meeting the performance requirements at the same time. This is based on the knowledge that there is a trade-off between operational frequency and energy dissipation of digital circuits. The investigated design techniques target the co-ordinated synthesis (co-synthesis) of mixed hardware and software applications towards the effective exploitation of DVS and ABB techniques, aiming to improvement in energy efficiency.

The main aim of this chapter is to place the problem of hardware and software co-synthesis within the general context of embedded system design flow, and illustrate the impact of hardware and software co-synthesis on design objectives including timing feasibility, cost, and especially the focus of this research, energy efficiency. The remaining of this chapter is organised as follows. Section 1.1 is an introduction to embedded systems and Section 1.2 outlines a typical embedded system design flow. Section 1.3 introduces the system specification models used in this thesis. Section 1.4 shows the impact of co-synthesis on the energy dissipation and the traditional design objectives of embedded systems. Finally, Section 1.5 gives an overview of the thesis and highlights the main contribution of this research.

1.1 Embedded Systems

Embedded systems are a class of computing systems that use programmable processors to implement part of their functionality [1]. But unlike general-purpose systems such as personal computers, embedded systems are designed for dedicated application. Figure 1.1 shows a typical embedded system architecture that can be found in a modern mobile phone [2]. It consists of two software programmable processors (CPU, DSP) and a dedicated hardware component (ASIC), which are interconnected through communication links. The CPU/DSP/ASIC interact with environment through input/output ports (I/O), analog-to-digital converters (ADC), and digital-to-analog converters (DAC). A complete embedded system, however, consists of not only an architecture but also software that is executed on the architecture. The architecture provides the raw computing power for the system. Many features of the system, however, are not directly implemented in the architecture but are instead designed into the software. The architecture and software work together to provide the functionality of the application.

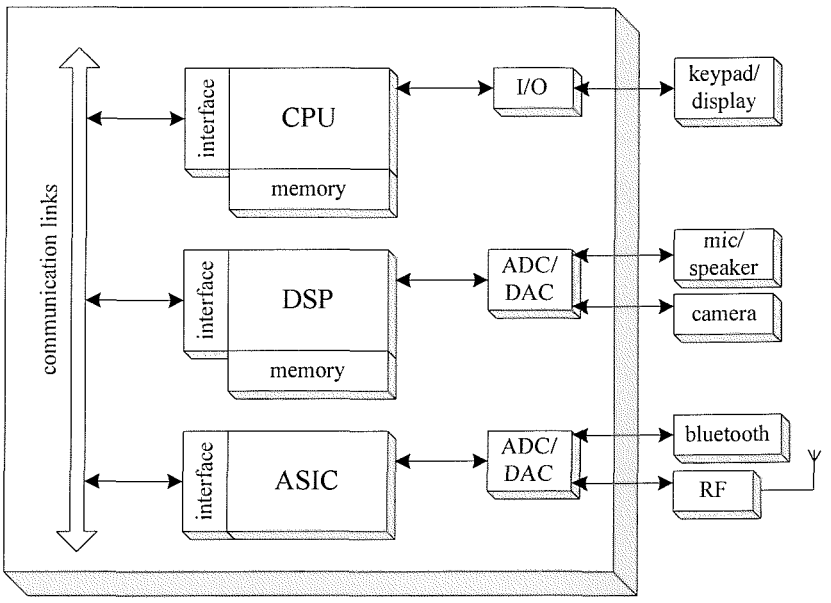


Figure 1.1: Embedded system architecture [2]

Comparing with general-purpose computing systems, embedded systems have three key characteristics described as follows [3, 4].

(1) Embedded systems are application-specific. Therefore, the functionality characteristics are defined before the system is designed. This allows static design schemes and deterministic performance analysis which helps fine tune the design decisions.

(2) Many embedded systems are hard real-time systems, for which the correctness of the system functionality depends not only on the logical results of computation, but also on the time at which the results are produced. A failure to complete a computation by a given deadline causes system failure. A clear example of hard real-time systems is the anti-lock brake controller in automobiles, which computes the brake pressure depending on the spin speed of the wheel and the brake pedal position actuated by the driver. A failure of computing the brake pressure in a certain interval may cause the automobile to go out of control.

(3) Many embedded systems are distributed systems. In order to provide design flexibility and cost efficient design, the embedded system architecture very

often consists of multiple heterogeneous processing elements (PEs) such as software programmable processors and dedicated hardware components. Software programmable processors can be general purpose processors (GPPs), microcontrollers, or digital signal processors (DSPs). The dedicated hardware components on the other side can be application specific integrated circuits (ASICs) or field programmable gate arrays (FPGAs). The multiple PEs are interconnected with communication links (CLs), such as processor bus, CAN (Controller Area Network) [5], I^2C-bus [6], RapidIO [7]. For example, a dual mode mobile communication handset [8] consists of an ARM processor core, a DSP core, and several dedicated-purpose hardware blocks, which are interconnected with a 32-bit system bus. These heterogeneous PEs are combined together to implement the signal processing, the mobile communication protocol stacks, the man-machine interface, and the control of the overall operation of the handset. There are several reasons to employ distributed heterogeneous architecture for embedded systems:

- Time-critical tasks may be placed on different processing elements (PEs) to ensure that all their hard deadlines are met. For example, a microprocessor has a good overall performance, but when it is busy working on a task, it may not be able to give fast response to other tasks at the same time.
- Combination of several small but cheap processors very often leads to a more cost-efficient design than the usage of a large and costly processor.
- The system may be physically distributed, or the imposed performance requirement cannot be met even on the most powerful processor available in the market.

1.2 Embedded Systems Design Flow

Embedded systems have been used in a wide diversity of application domains including automotive electronics, avionics, medical equipment, telecommunication devices, industrial equipment, household appliance, handheld devices, etc. Recently, the popularity of mobile applications has significantly increased. To be commercially

successful in the market, embedded systems, for example mobile applications, should have high performance, low cost, and energy efficiency in order to extend the battery lifetime. Designing such embedded systems is a challenging task, which is the main focus of this thesis.

As illustrated in Section 1.1 (Figure 1.1), an embedded system contains both hardware and software. The hardware and software work together to provide the functionality of the application. Clearly, embedded system design is a hardware-software co-design problem [1, 4, 9-13], which is a process of simultaneous designing the hardware and software portions of an embedded system while considering dependencies between the two, so that the implementation not only functions properly but also meets performance, cost and power goals. A possible design flow of energy-efficient embedded systems is shown in Figure 1.2. There are mainly three steps in the design flow: system specification (Step 1), hardware-software co-synthesis (Step 2), hardware synthesis and software synthesis (Step 3). A brief description of the steps is given here, while the details of system specification and hardware-software co-synthesis are given in Sections 1.3 and 1.4 respectively. The first step of the design flow is system specification. System specification gives the functionality, i.e., the operations to be performed by the system. System specification can be expressed using different representations such as abstract graphic representations, or high-level languages. Having defined the system specification, the next step of the design flow is hardware-software co-synthesis. One possible definition of hardware-software co-synthesis is the automated method for concurrently determining the hardware and software portions of an embedded system based on the system specification [14]. It is an iterative process involving four sub-steps: architecture allocation, application mapping, activity scheduling, and energy management (Figure 1.2). After these four sub-steps, the implementation is evaluated to decide whether it meets the design objectives in terms of timing feasibility, cost, and energy dissipation. If the implementation is not satisfied, the evaluation result is fed back to the four previous sub-steps to modify the implementation towards the desired design objectives. A brief introduction of these sub-steps are given here, while the detailed description and

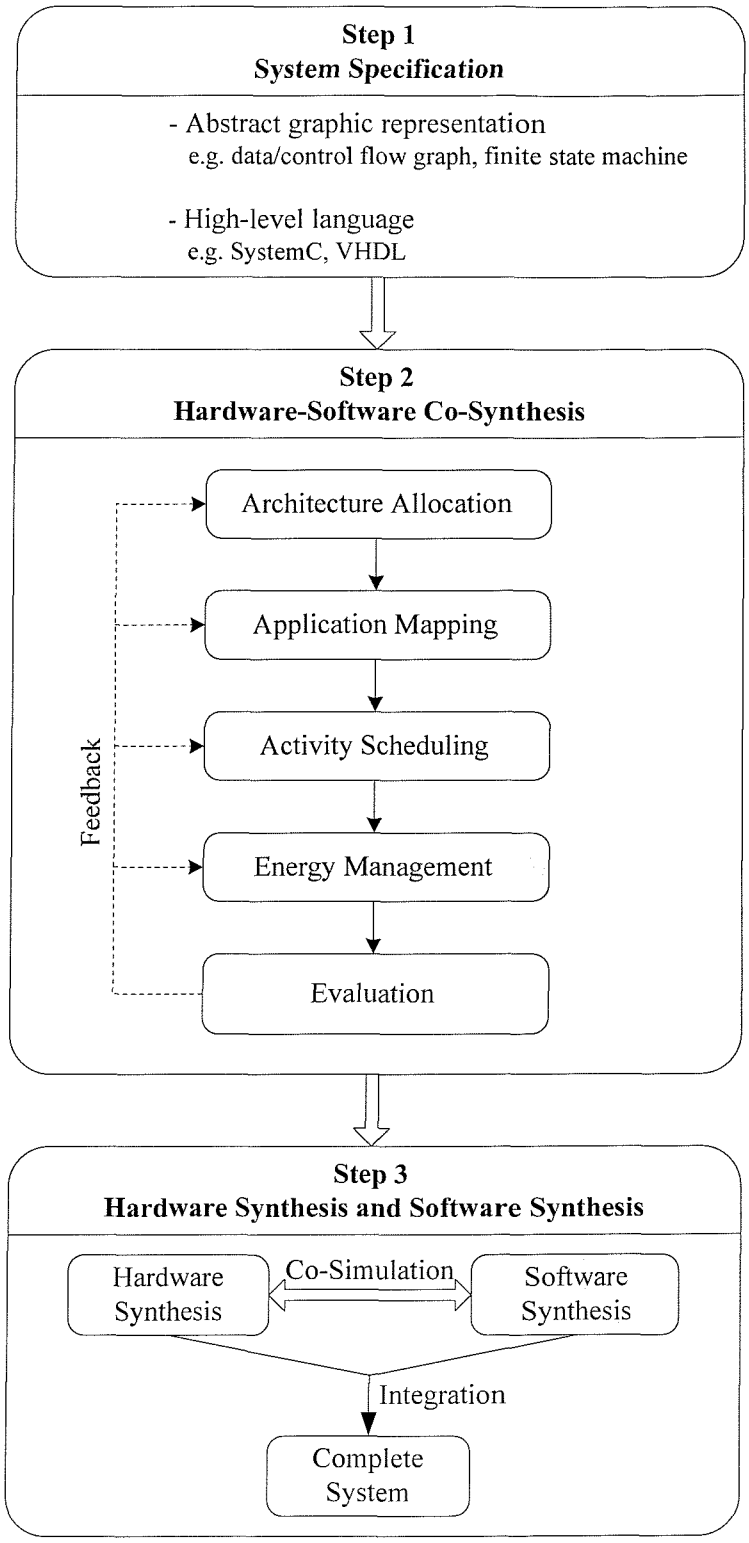


Figure 1.2: Typical design flow of energy-efficient embedded systems

impact of these sub-steps on the design objectives, including timing feasibility, cost and energy dissipation are given in Section 1.4.

Architecture Allocation – This involves the identification of an appropriate architecture, i.e., determining the type and number of the processing elements (PEs, i.e., software programmable processor or dedicated hardware component) and the communication links (CLs) interconnecting the PEs. In this thesis, this step is user-driven and thereby based on the knowledge and experience of the designer. It is assumed that the designer has predefined an architecture based on previous design experience, then application mapping, activity scheduling, and energy management techniques help the designer to evaluate the quality of the allocation in terms of energy dissipation, cost, and timing feasibility.

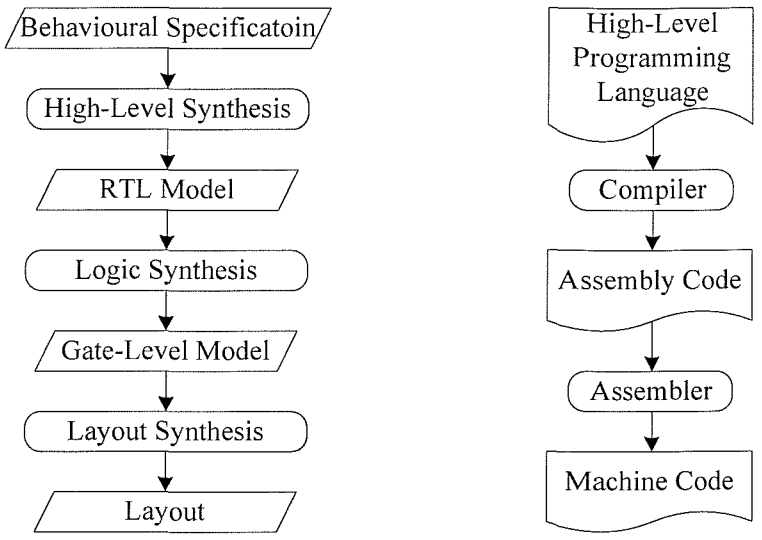
Application Mapping – This involves the mapping of the system functionality to the allocated system architecture, i.e., mapping the computation tasks and the communications among the tasks, which together consist of the system functionality, to the PEs and CLs respectively. Some tasks are to be mapped onto dedicated hardware component (i.e., implemented as hardware), while some tasks are to be mapped to software programmable processor (i.e., implemented as software).

Activity Scheduling – A correct execution order of the computation tasks and communication is determined, such that the timing constraints are satisfied. The data and control dependencies among the activities have to be taken into account in order to effectively exploit the parallelism in the application, i.e., to make tasks executed on multiple PEs in parallel.

The aim of these three sub-steps is to optimise the design according to the design objectives, which traditionally include performance and cost. However, due to the rising importance of the power issue, energy efficiency has become another essential design objective in recent years. In order to optimise energy efficiency, emerging co-synthesis techniques [15] tightly integrate the consideration of energy management techniques within the co-synthesis process.

Energy Management – Energy management techniques utilise the idle and slack times (the duration when the PE are not carrying out useful activities, Section 1.4.4) to reduce the power consumption by either shutting down the idle PEs or reducing the performance of the PEs by means of DVS and/or ABB (Chapter 2, Section 2.1.2).

After carrying out Step 2 (Hardware-Software Co-Synthesis), the next step in the design flow is hardware and software synthesis. In this step, the system functionality, which has been transformed into an architectural description of mixed hardware and software, is transformed into physical implementation. This is achieved by two separate processes: hardware synthesis and software synthesis [16]. **Hardware synthesis** creates hardware components, such as ASICs and FPGAs, for the tasks that have been mapped on hardware components in the co-synthesis step. The hardware synthesis is generally performed using the existing very large scale integration (VLSI) synthesis tools. A possible hardware synthesis flow [13, 17, 18] consists of three steps: high-level synthesis, logic synthesis, and layout synthesis, as shown in Figure 1.3(a). Firstly, the high-level synthesis [19, 20] transforms a behavioural specification, which may be specified as algorithms written in VHDL [21], Verilog HDL [22], or SystemC [23], into a structure of register-transfer level (RTL) components, such as registers, multiplexers, and arithmetic-logic units (ALUs). Secondly, the logic synthesis [24] transforms the RTL description into a gate-level representation. Finally the layout synthesis [25-27] generates the layout of the chip from the gate-level description. **Software synthesis** [28, 29] creates the software codes for the tasks that have been mapped to software programmable components. Figure 1.3(b) shows a possible software synthesis flow. Firstly, the initial specification in a high-level programming language, such as C/C++ [30] or Java [31], is compiled into assembly code using standard compilers or compilers specialised for specific application and processor type [32]. Then the generated assembler code is transformed into executable machine code using processor specific assemblers. The hardware and software parts can be co-simulated [33-35] to verify the design, and help designer find errors at the early design stage to avoid expensive re-designs. After the hardware and software synthesis are completed, the produced hardware and software are integrated together as a complete system.



(a) Hardware synthesis (b) Software synthesis

Figure 1.3: Flow of hardware synthesis and software synthesis

1.3 System Specification

Embedded systems design is the process of implementing a desired functionality using a set of hardware components and software codes [4]. The complete process begins with specifying the desired functionality, which can be captured using a variety of models. To be useful, a model should possess certain qualities [9]. Firstly, the model should be formal so that it contains no ambiguity, and complete so that it can describe the entire system. There are numerous models for system specification, for example, high-level languages such as SystemC, VHDL, and C/C++, as well as more abstract models such as finite state machines (FSM) [9], program-state machine (PSM) [9], Petri Net [36], and data/control flow graph [37-39]. The literatures show that no model is ideal for all classes of systems. The designer should choose a model that most closely matches the characteristics of the target system. Typical applications targeted by this thesis are in the domain of real-time applications with conditional behaviours. Such applications fall into the category of data and data/control flow dominated systems. The appropriate representations for these systems are the task graph model (Section 1.3.1) and conditional task graph model (Section 1.3.2). All the

results given in this thesis are obtained using task graphs and conditional task graphs, which are introduced next.

1.3.1 Task Graphs

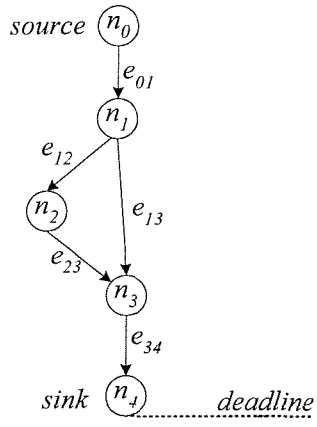
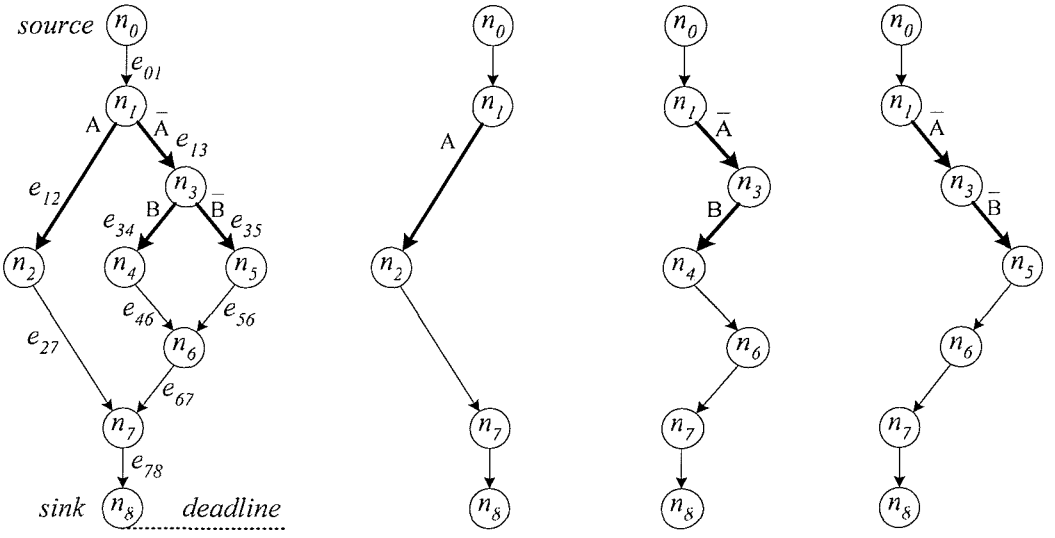


Figure 1.4: Example of a task graph

The functionality of a system containing intensive data flow can be specified as a task graph [9, 40]. Structurally, task graphs are similar to the data-flow graphs [9] that are commonly used in high-level synthesis [19, 20]. However, while nodes in data flow graphs represent single operations, such as multiplication and addition, the nodes in task graphs are associated with larger fragments of functionality, such as whole functions and processes. Each node in the task graph is a task (i.e., a single thread of execution). Tasks may be implemented as either software running on software programmable processors or hardware on dedicated hardware components. Figure 1.4 shows an example of a task graph. Task graph is a directed, acyclic graph $G = (N, E)$. The set of nodes $N = \{n_1, n_2, ..., n_n\}$ represents the set of tasks to be executed without being pre-empted. The set of directed edges $E = \{e_{ij} \mid n_i \in N, n_j \in N\}$ represents the communications between the tasks n_i and n_j: the output of n_i is the input of n_j. All tasks issue their outputs when they terminate. A task can only start its execution after all its inputs have arrived. For example, e_{13} denotes a communication between tasks n_1 and n_3, and e_{23} denotes a communication between tasks n_2 and n_3. Task n_3 can only start execution after the inputs from n_1 and n_2 have arrived. There are two special

nodes in the task graph: source and sink, which conventionally represent the first task and the last task, so that all other nodes in the graph are successors of the source and predecessors of the sink respectively. If you consider the activation of the source as a reference, the termination time of the sink is the delay of the task graph at a certain execution. This delay has to be smaller than a certain imposed deadline. The source task and sink task are introduced as dummy tasks with zero execution time. A dummy task is a task with zero execution time, or a task with non-zero execution time but not allocated to any physical processing element (PE). Detailed definition of dummy task and the extra modelling capability the dummy task provides are introduced in Appendix C. The concept of dummy node is used in Chapters 3, 4 and 5.

1.3.2 Conditional Task Graphs



(a) conditional task graph (b) track 1 (c) track 2 (d) track 3

Figure 1.5: Example of a conditional task graph and its tracks

An embedded system functionality often contains not only data flows but also control flows, which cannot be captured using task graph. This aspect has been recognised by the research community and conditional task graph (CTG) [41, 42] has been proposed to capture both the data and control flows at task level. A conditional task graph is similar to a traditional task graph (Figure 1.4), except that there are some conditions in

the graph to capture the control flow. Figure 1.5(a) shows an example of a conditional task graph. In a conditional task graph $G(N, E_S, E_C)$, N is the set of node, E_S and E_C are the sets of simple and conditional edges respectively, $E_S \cap E_C = \phi$ and $E_S \cup E_C = E$, where E is the set of all edges.

An edge $e_{ij} \in E_C$ is a *conditional edge* (represented with thick lines in Figure 1.5(a)) and has an associated condition value. Transmission on such an edge takes place only if the associated condition value is true and not, like on simple edges, for each activation of the input task n_i. For example, in Figure 1.5(a) edges e_{12}, e_{13}, e_{34}, and e_{35} are all conditional edges, their associated condition values are A, $\overline{A}$, B, and $\overline{B}$ respectively. A node with conditional edges at its output is called a *disjunction node* (and the corresponding task a *disjunction task*). A disjunction node has one associated condition, a value of which it computes. Alternative paths starting from a disjunction node, which correspond to complementary values of the condition, are disjoint and they meet in a *conjunction node* (with the corresponding task called *conjunction task*). For example, in Figure 1.5(a) n_1 and n_3 are disjunction nodes, n_6 and n_7 are conjunction nodes. Consider disjunction node n_3, there are two alternative paths starting from n_3. The first path is $n_3 \rightarrow n_4 \rightarrow n_6$ and the second path is $n_3 \rightarrow n_5 \rightarrow n_6$. These two paths meet at conjunction node n_6. At a given execution, which path to follow is dependent on the value that computed by n_3. If the computation result of n_3 is B, then the first path is followed, otherwise, the second path is followed. A task, that is not a conjunction task, can be activated only after all its inputs have arrived, whilst a conjunction task can be activated after message coming on one of the alternative paths has arrived. Therefore, at a given execution of the system, depending on the condition values that are produced by the disjunction tasks, only a certain subset of the total tasks (track) is executed, and this subset differs from one execution to the other. Consider CTG of Figure 1.5(a), there are three possible tracks that may be followed at a certain execution, which are shown in Figure 1.5(b) – (d) respectively. For example, if in a certain execution, the condition values produced by n_1 and n_3 are $\overline{A}$ and B, track 2 of Figure 1.5(c) will be followed. In Chapter 3, Sections 3.5.1 and Chapter 4, Section 4.4.1, the task graphs of real-life applications of vehicle cruise controller and GSM voice CODEC are derived.

1.4 Co-Synthesis of Embedded Systems

Section 1.3 has briefly outlined the main steps in co-synthesis of embedded systems. In this section, it will show how the steps of architecture allocation, application mapping, activity scheduling and energy management affect the quality of co-synthesis, in terms of cost, performance and energy efficiency.

1.4.1 Example 1: Architecture Selection

Given a library of system components, i.e. processing elements (PEs) and communication links (CLs), the task of architecture allocation is defined as the selection of the type and number of the PEs as well as the determination of their interconnection to meet the design objectives. There are many different possible architectures that can be used to implement a given system functionality. The overall goal of architecture allocation is to identify a suitable architecture, which can provide sufficient computing power to meet the performance requirement, while, at the same time, minimise the cost, power consumption, and design time. In embedded systems, there are two classes of processing elements (PEs): software programmable processors (e.g. CPU, DSP) and dedicated hardware components (e.g. ASIC, FPGA). CPU/DSP are referred to as software programmable processors because their functionality is controlled by application software. While software executing on off-the-shelf CPU/DSP is more flexible and cheaper to implement than hardware components, the ASIC offers higher performance and 1 – 2 orders of magnitude more energy efficiency [2]. Clearly, selecting the appropriate system components, in order to balance between these trade-offs, is essentially important for high quality designs. For example, Figure 1.6 shows two potential architectures for a fictitious embedded system. Assuming the cost and power consumption of the components are as shown in Table 1.1, since architecture 1 implements more functionality in hardware than Architecture 2, Architecture 1 has higher cost but consumes less power than Architecture 2.

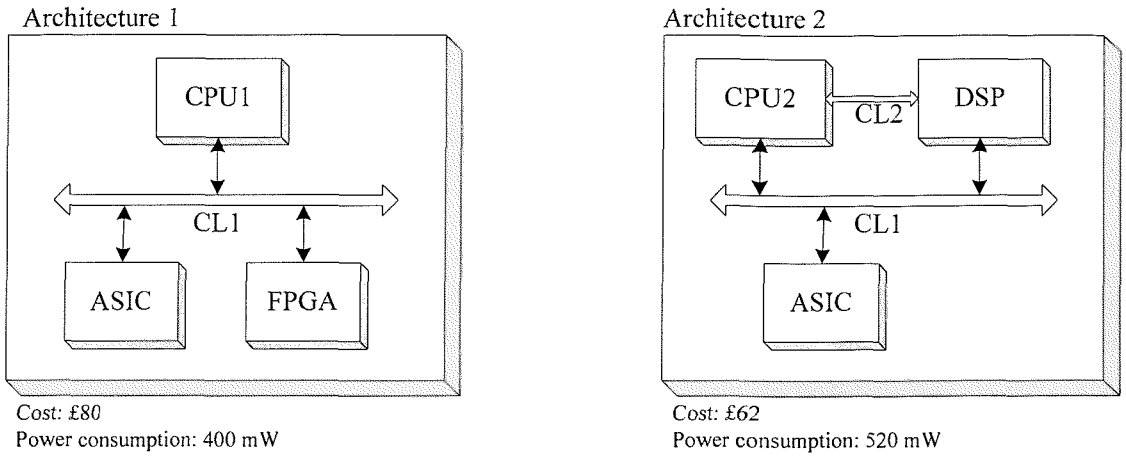


Figure 1.6: Architecture allocation example

Component	CPU1	CPU2	DSP	ASIC	FPGA	CL1	CL2
Cost (£)	20	10	15	30	25	5	2
Power (mW)	140	200	160	100	120	40	20

Table 1.1: System components library

1.4.2 Example 2: Mapping

Figure 1.6 highlights the impact of an appropriate architecture allocation on design objectives. However, how good performance the allocated architecture can present significantly depends on the application mapping. In application mapping, the tasks and communications are mapped onto the allocated processing elements (PEs) and communication links (CLs) respectively. Figure 1.7 illustrates two different mappings of a task graph onto an allocated architecture. These two mappings differ in that task n_1 and n_4 swap their assignments. Note that the source task n_0 and the sink task n_6 are not mapped to any PEs because they are dummy tasks (Appendix C).

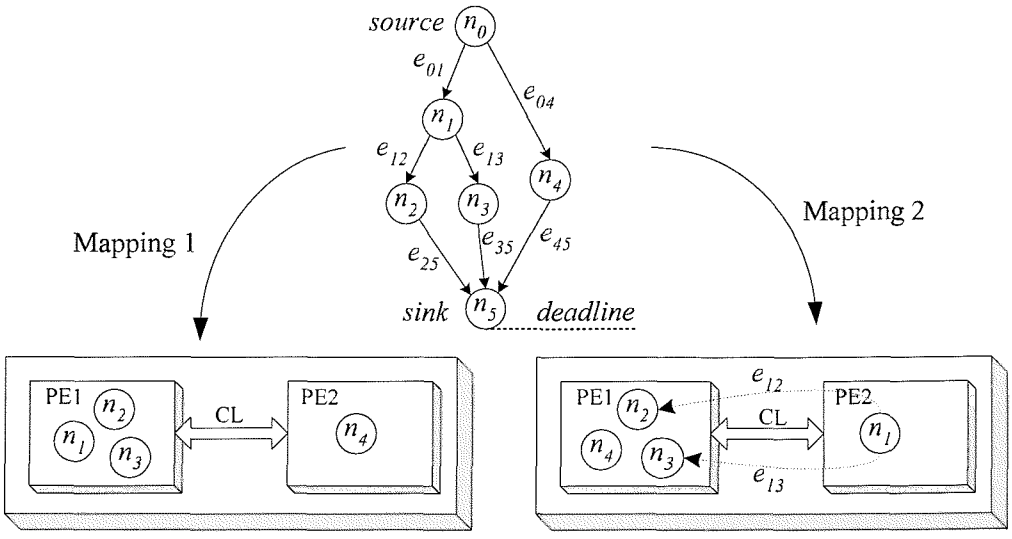


Figure 1.7: Application mapping example

Mapping affects system performance in three ways:

(1) Mapping impacts the execution characteristics of the tasks and communication. Due to the heterogeneity of system components, the execution characteristics (e.g., execution time, power consumption, and, for tasks implemented as hardware, silicon area) of the tasks varies in corresponding to their mappings. For example, assume the execution characteristics of task n_1 of Figure 1.7 are as shown in Table 1.2, where T, P and A denote execution time, power consumption and silicon area respectively. As it can be seen, if n_1 is mapped to PE1 (software processor), its execution time is 56.2ms and it consumes 26.1mW power, while if it is mapped to PE2 (hardware component), its execution time is 4.9ms, it consumes 1.8mW power and 8.7 mm^2 silicon area.

(2) Mapping impacts the communication overheads. There is a normal assumption in hardware-software co-synthesis: the data exchange between two tasks causes communication overhead only if the two tasks are mapped to different PEs, while data exchange between tasks mapped to an identical PE doesn't have communication overhead. Therefore, the mapping has an important effect on the communications. For example, consider the two mappings of Figure 1.7. In Mapping 1, three tightly coupled tasks n_1, n_2, n_3

are mapped to PE1, in such way the data exchanges between them do not cause communication overhead. While in Mapping 2, n_1 is mapped to PE2, thus imposes two communications overheads (e_{12} and e_{13}) on CL.

(3) Mapping influences the utilisation of the architecture. An improper mapping may result in poor utilisation of the architecture, leading to timing infeasibility. For example, assigning tasks, which can be executed in parallel, to a same software PE may result in undesirable delay, since only one task can be executed on a software PE at a given time. This delay could have been avoided if the tasks are distributed to several different PEs.

Task	PE1 (SW)		PE2 (HW)		
	T (ms)	P (mW)	T (ms)	P (mW)	A (mm^2)
n_1	56.2	26.1	4.9	1.8	8.7

Table 1.2: Task implementation

1.4.3 Example 3: Scheduling

Activity scheduling establishes a suitable execution order of the tasks and communications, such that the timing constraints are satisfied. A good schedule efficiently exploits the parallelism between activities to improve the system performance. For example, given a task graph, an architecture and a possible mapping, Figure 1.8 shows two possible schedules (Schedule 1 and Schedule 2). Note that the dummy tasks (n_0 and n_7) are not mapped and scheduled, also the communications are ignored in this simple example in order to give a clear illustration. Consider the following scheduling scenario given in Schedule 1 and Schedule 2. After task n_1 has finished its execution, tasks n_3 and n_4 are both ready to execute. However, because n_3 and n_4 are mapped to the same PE1, they cannot execute at the same time. Thus, a scheduling decision has to be taken at this point – which task to execute first. Schedule 1 (Figure 1.8) corresponds to a schedule in which n_4 executes before n_3. As it can be seen from Schedule 1, the delay of the task graph is d_1, before the imposed *deadline*. On the other hand, if n_3 executes before n_4, as shown in Schedule 2, the late execution of n_4 further delays the execution of n_6. The delay of the task graph using Schedule 2 is d_2, violating the *deadline*. Therefore, Schedule 1 is a better solution

since it meets the *deadline*. This example shows that activity scheduling has a significant impact on the delay of the application, which, in turn, influences the energy dissipation of the application. This is because the application delay explicitly decides available slack time, which can be exploited by energy management to reduce energy dissipation, as discussed next.

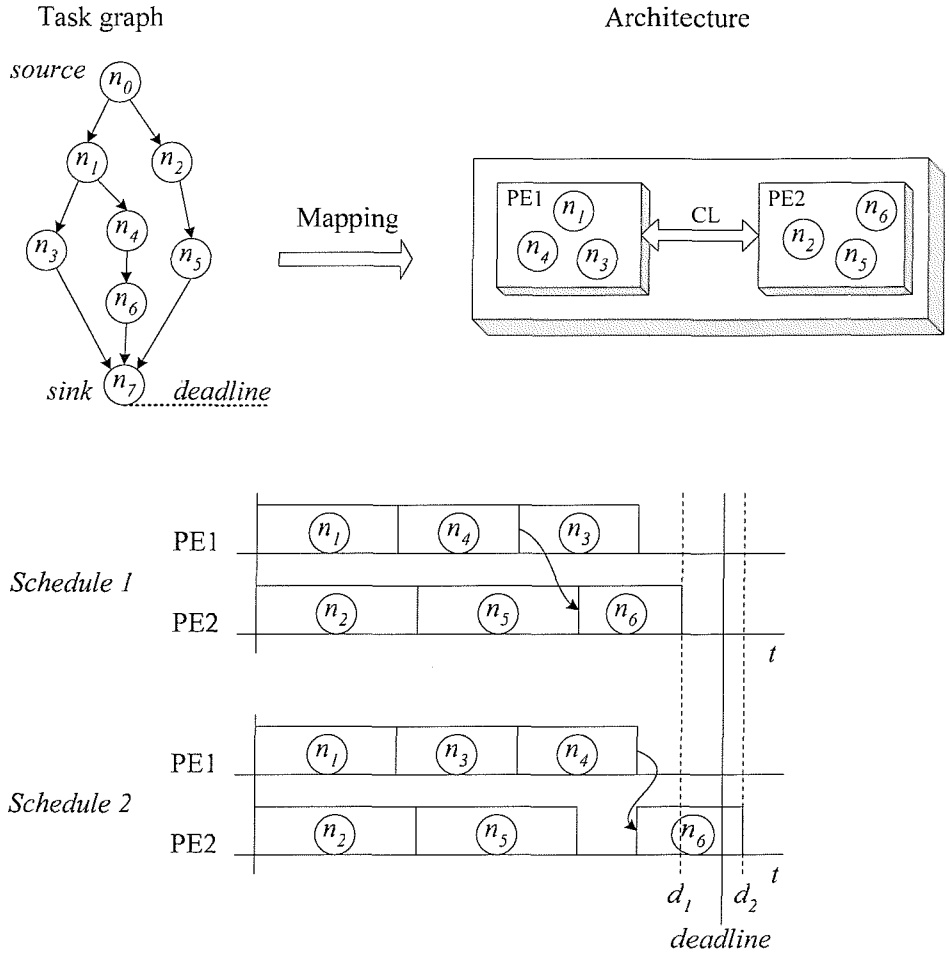


Figure 1.8: Activity scheduling example

1.4.4 Example 4: Energy Management

Early embedded systems design research has focused on traditional design objectives including cost and performance. However, recently energy efficiency has become another important design objective due to the rising requirement of low power. In order to improve energy efficiency of embedded systems, energy management has

become an essential step in the co-synthesis. Energy management techniques exploit the possibility of energy saving after the scheduling step has been determined. Based on the fact that there are periods when applications do not require the maximum performance provided by the allocated architecture, energy management techniques identify these periods, and lower the system performance during these periods to reduce energy dissipation. There are two types of periods when the trade-off between performance and energy is possible. The periods are idle time and slack time, where:

- Idle Time refers to the period in the schedule when PEs and CLs do not experience any workload, i.e., during these intervals the components are redundant.
- Slack Time is the difference between the scheduled finish time and the latest end time (LET) of a task, i.e., slack times are a result of over-performance. Slack time is a special case of idle time.

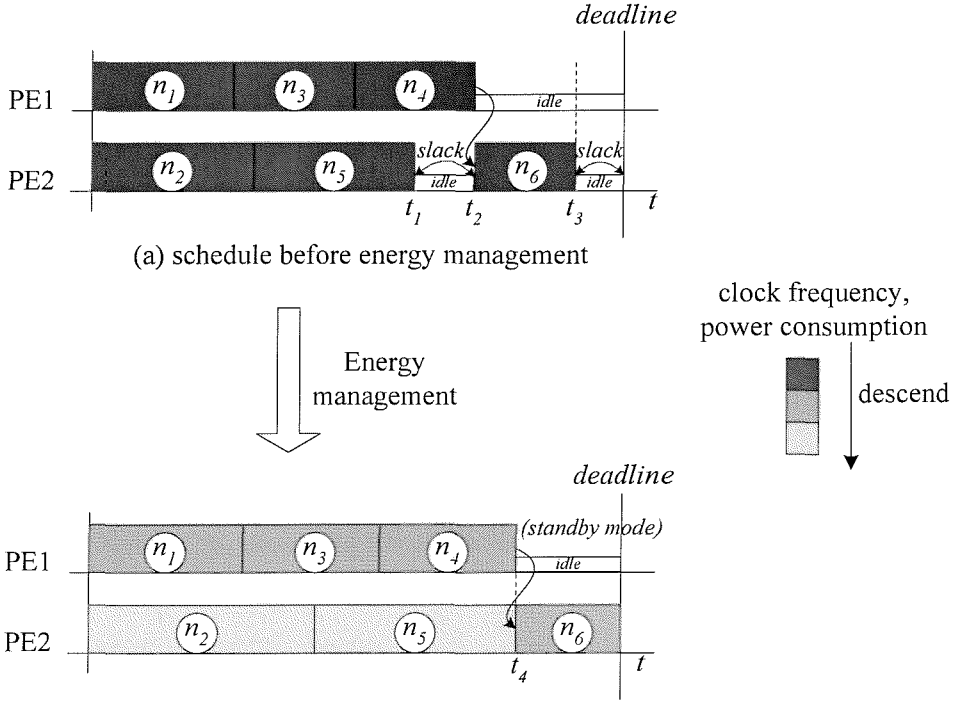


Figure 1.9: Energy management example

For example, Figure 1.9(a) shows a possible schedule for the task graph, architecture, and mapping of Figure 1.8. As indicated in the figure, PE1 has idle time between t2 and the deadline, PE2 has idle time between t1 and t2, and between t3 and the deadline. Since n6 finishes at t3 while the latest end time of n6 is the deadline, PE2 has slack time between t3 and the deadline. Similarly, PE2 also has slack time between t1 and t2.

There are two main energy management techniques: dynamic power management (DPM) [43-51] and dynamic voltage scaling (DVS) (Chapter 2, Section 2.1.2.1) [52-59]. DPM puts the processing elements (PEs) or communication links (CLs) into standby mode whenever they are idle to save energy. In DVS, on the other hand, different tasks and communications run at different supply voltage and clock frequency according to the temporal performance requirement, in order to fill up the slack times in the schedule while still providing an adequate level of performance. By reducing the supply voltage (V_{dd}) of PEs, DVS achieves significant reduction in dynamic energy ($E_{dynamic} \propto V_{dd}^2$). The basic concepts of DPM and DVS are demonstrated in Figure 1.9(b), which is the schedule after these two energy management techniques are applied. As it can be seen, PE1 is put into standby mode between t4 and the *deadline* using DPM. Also, the execution of all tasks is prolonged. This is achieved by scaling down the clock frequency and supply voltage of the PEs using DVS, until all slack times are fully exploited.

As technology feature size continues to scale, leakage power (Chapter 2, Section 2.1.1.2) is becoming important in deep sub-micron technology (Chapter 2, Figure 2.3). Recently adaptive body biasing (ABB, Chapter 2, Section 2.1.2.2) is reported as an effective technique to reduce leakage power [60-64]. Similar to DVS, ABB trades off performance against energy efficiency. ABB simultaneously reduces the clock frequency and increases the threshold voltage (V_{th}) of the PEs through body bias control. In this way, it achieves significant reduction in leakage power ($P_{leakage} \propto e^{-Vth}$). The application of ABB is similar to Figure 1.9(b), i.e., prolong the execution of the tasks to exploit the slack times.

Clearly, the effectiveness of energy management techniques depends to a large extent on the available idle and slack times. The proposed co-synthesis techniques of this thesis produce application mapping and activity scheduling taking into account the optimisation of idle and slack time, allowing an effective exploitation of the energy management techniques.

1.5 Contributions and Thesis Overview

This thesis presents new design techniques for the co-synthesis of energy-efficient distributed embedded systems. In particular, the energy reduction capabilities of dynamic voltage scaling (DVS) and adaptive body biasing (ABB) are investigated and analysed in the context of embedded systems with strict real-time constraints. The remainder of this thesis is organised as follows. Motivation for co-synthesis of energy-efficiency embedded systems, the necessary background information, as well as a comprehensive review of the most relevant research for minimising energy dissipation during co-synthesis are given in Chapter 2.

Chapter 3 presents a novel DVS algorithm for embedded systems expressed as conditional task graphs (CTGs), which capture both control and data flow within the application functionality. The proposed conditional behaviour aware DVS produces a static voltage schedule so that the energy dissipation is minimised and performance constraints are satisfied simultaneously. Furthermore, a genetic algorithm (GA) based mapping is introduced to optimise the system implementation to efficiently exploit the conditional behaviour aware DVS technique, hence, leading to further energy saving. Combining the proposed DVS and mapping algorithm, a co-synthesis technique is developed for embedded systems expressed as CTGs.

Chapter 4 extends the co-synthesis technique to address the impact of communications. The extended co-synthesis technique is applied to a large number of examples, including a real-life GSM CODEC example, which shows the effectiveness of the proposed technique. Also, the extended co-synthesis technique has been used to investigate the effect of alternative communication architecture on system quality in

terms of energy efficiency. As the technology continues to scale, it is expected that leakage power will become comparable to dynamic power in the future. Chapter 5 presents a co-synthesis technique for distributed embedded systems for both dynamic and leakage energy minimisation. The technique performs a power management selection, and maps, schedules, and scales supply and bias voltage. A key feature of the proposed technique is a power-composition aware mapping and the employment of processing elements (PEs) with separate DVS or ABB capability. This has the benefit of reduced cost whilst achieves comparable energy reduction to that achieved using PEs with combined DVS and ABB capability.

The work presented in this thesis has resulted in the following publications:

- "Scheduling and mapping of conditional task graphs for the synthesis of low power embedded systems", D. Wu, B. M. Al-Hashimi and P. Eles, in Proceedings of Design, Automation and Test in Europe, pp. 90-95, March 2003, Munich, Germany [65].
- "Scheduling and mapping of conditional task graph for the synthesis of low power embedded systems", D. Wu, B. M. Al-Hashimi and P. Eles, IEE Proceedings Computers and Digital Techniques, vol. 150, no. 5, pp. 262-273, September 2003 [66]. An extended version of a previous paper [65].
- "Dynamic voltage scaling for control flow-intensive applications", D. Wu, B. M. Al-Hashimi and P. Eles, in M. T. Schmitz, B. M. Al-Hashimi and P. Eles, System-level design techniques for energy-efficient embedded systems, Chapter 6, Kluwer Academic Publishers, 2004 [2].
- "Dynamic and leakage power-composition profile driven co-synthesis for energy and cost reduction", D. Wu, B. M. Al-Hashimi and P. Eles, in Proceedings of IEE/ACM Postgraduate Seminar on SoC Design, Test and Technology, 2004, Loughborough, UK.
- "Energy-efficient co-synthesis of data/control dominated embedded systems", D. Wu and B. M. Al-Hashimi, submitted to PhD Forum at the Design, Automation and Test in Europe 2005.

Chapter 2

Background and Previous Work

Low power design techniques for digital components have received significant attention and research effort over the last decade [67-72]. These techniques focus mainly on the circuit and RTL level. However, it is well known that the higher the level of the design hierarchy where power problem is addressed, the higher the power reduction is possible [15]. The research presented in this thesis attempts to improve the energy efficiency of embedded systems in the context of system level co-synthesis. The aim of this chapter is to introduce the background regarding the power consumption within embedded systems and energy management techniques applicable in embedded system co-synthesis (Section 2.1), provide an overview of the previous research reported in the literature (Section 2.2), and outline the motivations of the proposed work in this thesis (Section 2.3).

2.1 Background

A review of the power consumption of embedded systems is outlined in Section 2.1.1. Two main energy management techniques employed in this thesis, dynamic voltage scaling (DVS) and adaptive body biasing (ABB), are introduced in Section 2.1.2.

2.1.1 Power Consumption of Embedded Systems

In order to design energy-efficient embedded systems, it is important to understand the sources of power consumption. The power consumed by processing elements (PEs) of embedded systems consists of two parts [2]:

(1) static power, which occurs whenever the PE is switched on, even when no computation activity is carried out by the PE;

(2) dynamic power, which is caused by switching activity within the circuitry that only occurs when computation activity is being performed.

The total power consumption of a PE is given by:

$$P_{PE} = P_{static} + P_{dynamic} \tag{2.1}$$

Static power and dynamic power both can be further divided into two components [67, 73]:

$$P_{static} = P_{leakage} + P_{bias} \tag{2.2}$$

$$P_{dynamic} = P_{switching} + P_{short-circuit} \approx P_{switching} \tag{2.3}$$

$P_{leakage}$, leakage power, is due to the leakage current, $I_{leakage}$, which can arise from reverse-bias diode currents and sub-threshold effects. P_{bias}, bias power, arises from circuits that have a constant source of current between the supply and ground (e.g., bias circuitry). $P_{switching}$, switching power, represents the switching component of power, arises when energy is drawn from the power supply to charge load capacitors during switching activity. $P_{short-circuit}$, short-circuit power, is due to the direct-path short circuit current, I_{SC}, which arises when both the NMOS and PMOS transistors are simultaneously active, conducting current directly from supply to ground. This direct-path short circuit current exists for a short period of time during switching activity. Short-circuit power is often trivial in comparison with switching power [67], thus switching power is often referred to as dynamic power.

For $\geq$ 0.1μm CMOS technology, switching power $P_{switching}$ is the dominant component, accounting for approximately 90% of the total PE power consumption [67]. As the feature size continues to shrink (< 0.1μm), it is expected that $P_{leakage}$ will increase and become an important component in the total power consumption [72, 74]. The following sub-sections introduce switching power and leakage power in details.

2.1.1.1 Switching Power

Switching power arises when energy is drawn from the power supply to charge load capacitance during switching activity. To examine the energy drawn from the supply for each switching activity, consider the simple inverter gate shown in Figure 2.1, where C_L is the physical load capacitance at the output node and V_{dd} is the supply voltage. This inverter undergoes the following transitions. First, the input signal is set to 1, the transistor T1 is off and the transistor T2 is on. Thus, C_L is discharged since T2 pulls the capacitance to ground. Now consider a 1→0 transition at the input signal. In this case, T1 is on and T2 is off. T1 connects C_L to V_{dd}. V_{dd} charges C_L via T1, until V_{out} reaches V_{dd} at time T, resulting in a 0→1 transition at the output node. The power dissipation of this transition is given by:

$$P_{switching} = V_{dd} \cdot i_C \tag{2.4}$$

where the charging current i_C is given by:

$$i_C = C_L \cdot \frac{\partial V_{out}}{\partial t} \tag{2.5}$$

Therefore, the energy drawn from the power supply for a 0→1 transition at the output node is given by:

$$E_{0\to1} = \int_0^T P_{switching} dt = V_{dd} \cdot \int_0^T i_C dt = V_{dd} \cdot C_L \cdot \int_0^{V_{dd}} dV_{out} = C_L \cdot V_{dd}^2 \tag{2.6}$$

For this transition, the energy stored in C_L is given by:

$$E_{C_L} = \int_0^T P_{C_L} dt = \int_0^T (V_{out} \cdot i_C) dt = C_L \cdot \int_0^{V_{dd}} V_{out} dV_{out} = \frac{1}{2} C_L \cdot V_{dd}^2 \tag{2.7}$$

Therefore, half of the energy drawn the power supply is stored in C_L, and half of the energy is dissipated in T1. It can be observed that for a 0→1 transition at the input

signal, no energy is drawn from the power supply, but instead the energy stored in C_L is dissipated via T2.

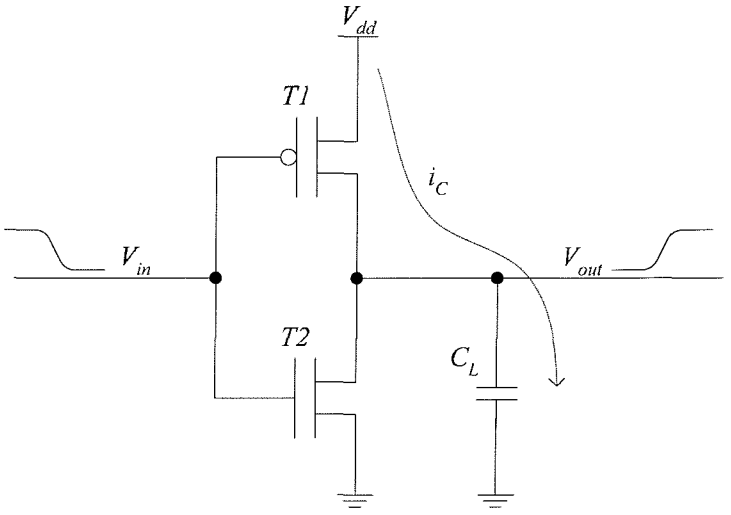


Figure 2.1: Switching power consumption of an inverter [67]

Although the discussion above is for a simple inverter, the validity of Equation 2.6 holds for more complex gates and other logic styles [67, 70, 75]. Therefore, executing a task τ on a PE with N nodes requires switching energy dissipation given by:

$$E_{switching}(\tau) = N_C(\tau) \cdot V_{dd}^2 \cdot \sum_{i=1}^{N} (\alpha_i \cdot C_{Li}) \tag{2.8}$$

where $N_C(\tau)$ is the number of clock cycles needed to execute τ, α is the 0→1 transition activity factor (the number of 0→1 transition per clock cycle averaged over a number of clock cycles, $0<\alpha<1$) averaged over the $N_C(\tau)$ cycles. Equation 2.8 can be simplified to:

$$E_{switching}(\tau) = N_C(\tau) \cdot C_{eff} \cdot V_{dd}^2 \tag{2.9}$$

where C_{eff} is the effective capacitance of the whole PE averaged over the whole duration of execution. Assuming the clock frequency of the PE is f, the average switching power of a PE performing a computational task τ can be derived:

$$P_{switching}(\tau) = \frac{E_{switching}(\tau) \cdot f}{N_C(\tau)} = f \cdot C_{eff} \cdot V_{dd}^2 \quad (2.10)$$

It can be seen from Equations 2.9 and 2.10 that, assuming C_{eff} is a constant determined by the design and the circuit technology, $N_C(\tau)$ is a constant determined by the computational task and the PE, then $E_{switching}(\tau)$ is proportional to the square of V_{dd}, whilst $P_{switching}(\tau)$ is proportional to the product of f and the square of V_{dd}. Although decreasing the clock frequency f reduces the power consumption, it doesn't reduce the switching energy needed to complete the execution of task τ. Consider the following example. A task requires 10M clock cycles to be executed on a PE. The PE works at 100MHz and consumes 100mW switching power. The execution of this task dissipates switching energy of 100mW · (10M / 100MHz) = 10mJ. Alternatively, the PE can work at a reduced clock frequency 50MHz. According to Equation 2.10, the switching power of the PE reduced to 100mW · (50MHz / 100MHz) = 50mW. However, the switching energy is 50mW · (10M / 50MHz) = 10mJ, the same as in the case of 100MHz. Therefore, it is clear that reducing the clock frequency does not reduce switching energy, reducing the supply voltage is the only possibility to reduce the switching energy.

Switching energy is proportional to the square of supply voltage. However, this simple solution to energy-efficient design comes at a cost. When reducing the supply voltage of a digital circuit, the time required for gate signals to settle is prolonged, which increases the circuit delay [67]. The effect of supply voltage reduction on circuit delay for adder and multiplier logic circuits is shown in Figure 2.2. Both the curves in the figure present a trend that the circuit delay increases as V_{dd} reduces, and the delay drastically increases as V_{dd} approaches the threshold voltage (V_{th}) of the circuits. The increase of circuit delay necessitates the reduction of the clock frequency in order to ensure correct operation. The circuit delay d, which is inversely proportional to the clock frequency f, can be approximated (error < 10%) as [67]:

$$d \approx k_d \cdot \frac{V_{dd}}{(V_{dd} - V_{th})^2} \quad (2.11)$$

where V_{th} is the threshold voltage, and k_d is a technology dependent constant given by:

$$k_d = \frac{C_L}{2 \cdot W \cdot v_{sat} \cdot C_{OX}} \tag{2.12}$$

where the constants W, v_{sat}, and C_{OX} denote the width of the sub-micron CMOS device, saturated velocity of the carriers, and gate capacitance respectively, C_L denotes the load capacitance.

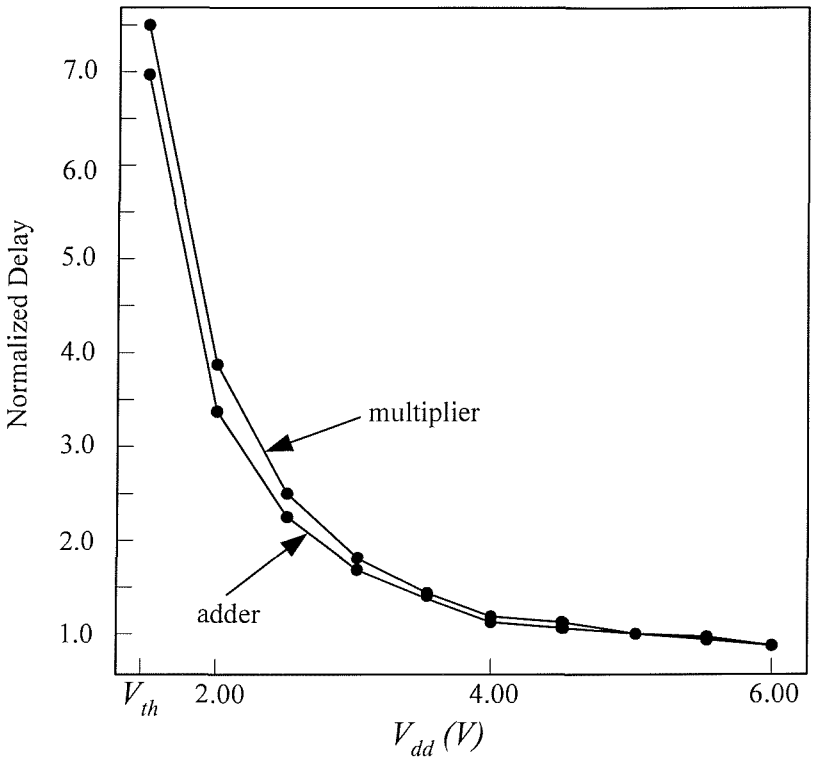


Figure 2.2: Delay of adder and multiplier vs. supply voltage characteristics [67]

2.1.1.2 Leakage Power

Leakage power arises from two types of leakage current: reverse bias diode leakage at the transistor drains, and sub-threshold leakage through the channel of an "off" device. Diode leakage occurs when a transistor is turned off and another active transistor charges up/down the drain with respect to the former's bulk potential. Sub-threshold leakage occurs due to carrier diffusion between the source and the drain when the

gate-source voltage V_{GS} has exceeded the weak inversion point, but is still below the threshold voltage V_{th}, where carrier drift is dominant. The magnitude of diode leakage is negligible comparing with sub-threshold leakage [60, 61]. Therefore, the leakage current can be approximated as sub-threshold leakage, which is given by[67, 76]:

$$I_{leakage} \approx I_{sub-threshold} = I_0 \cdot e^{\frac{V_{GS}-V_{th}}{n \cdot V_T}} \cdot \left(1 - e^{\frac{-V_{DS}}{V_T}}\right) \tag{2.13}$$

$$I_0 = \mu_0 C_{ox} \frac{W}{L}(n-1)V_T^2 \tag{2.14}$$

$$n = 1 + \frac{\varepsilon_{si}}{\varepsilon_{ox}} \cdot \frac{t_{ox}}{W_{dm}} \tag{2.15}$$

where V_T is the thermal voltage (KT/q), V_{th} is the threshold voltage, V_{GS} and V_{DS} are gate-to-source voltage and drain-to-source voltage respectively, μ_0 is the zero bias mobility, C_{ox} is the gate oxide capacitance, W/L is the ratio of channel width to channel length of the device, n is the sub-threshold swing coefficient, t_{ox} is the gate oxide thickness, W_{dm} is the maximum depletion layer width of the device, ε_{si} and ε_{ox} are the dielectric constants of silicon and oxide respectively. For $V_{DS} >> V_T$, Equation 2.13 can be simplified to:

$$I_{leakage} \approx I_0 \cdot e^{\frac{V_{GS}-V_{th}}{n \cdot V_T}} \tag{2.16}$$

therefore, leakage power is given by:

$$P_{leakage} \approx V_{dd} \cdot I_0 \cdot e^{\frac{V_{GS}-V_{th}}{n \cdot V_T}} \tag{2.17}$$

The magnitude of leakage power is set predominantly by the processing technology. For ≥ 0.1μm CMOS technology, leakage power is trivial comparing to

switching power (<10%) [77]. However, as the CMOS feature size continues to shrink, it is expected that leakage power will increase exponentially and become an important component in the total power consumption [74, 77]. Figure 2.3 illustrates the estimated dynamic and leakage power of an IC varying across the technologies [77]. The trend of increasing leakage power raises the need for leakage power reduction techniques.

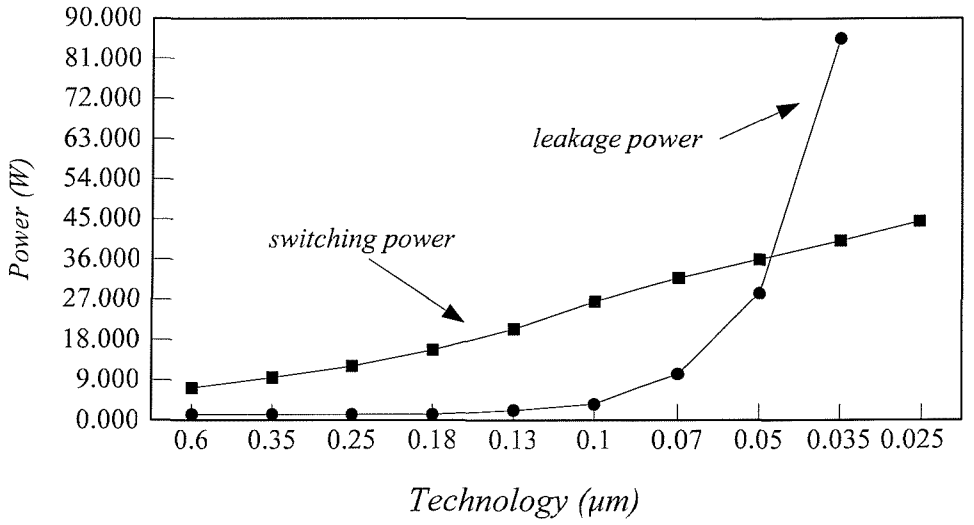


Figure 2.3: Switching and leakage power vs. CMOS technology [77]

From Equation 2.17, it can be seen that, assuming the supply voltage is constant, the leakage power is exponentially proportional to the threshold voltage. The effect of threshold voltage to the leakage power is shown in Figure 2.4. As the threshold voltage increases, the leakage power decreases dramatically. For example, when V_{th} increases from 0.2V to 0.4V, the leakage power decrease from 10e4 to 10e2 (100 times lower). However, according to Equation 2.11, increasing threshold voltage also results in longer circuit delay, which, in turn, necessitates the reduction of the clock frequency in order to ensure correct operation. Figure 2.4 also shows how circuit delay varies with threshold voltage.

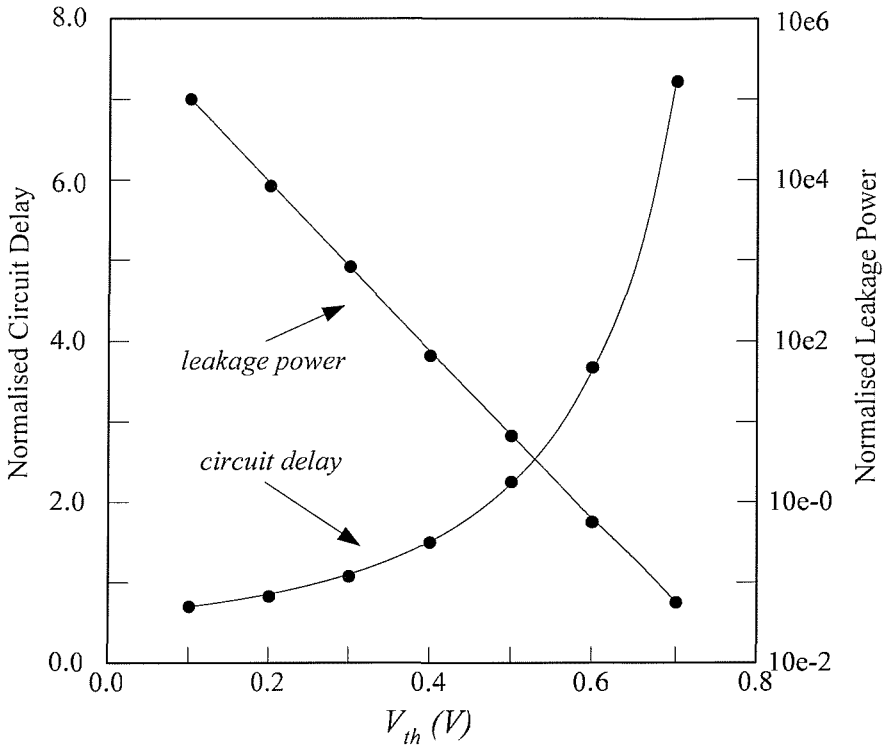


Figure 2.4: Leakage power and circuit delay vs. threshold voltage [61]

2.1.2 Energy Management Techniques

This section introduces two main energy management techniques that have received considerable attention from academia and industry: dynamic voltage scaling (DVS), and adaptive body biasing (ABB), which are used in the co-synthesis techniques presented in this thesis.

2.1.2.1 Dynamic Voltage Scaling

This section introduces dynamic voltage scaling (DVS), an effective technique for dynamic power reduction. DVS is based on the fact that the dynamic energy needed for a processing element (PE) to execute a task is proportional to the square of the supply voltage (Equation 2.9). However, decreasing supply voltage also results in prolonged circuit delay (Equation 2.11), which in turn necessitates the reduction of clock frequency, i.e. the performance of the PE. The idea of DVS is to dynamically vary the supply voltage and clock frequency of the PE in corresponding to the

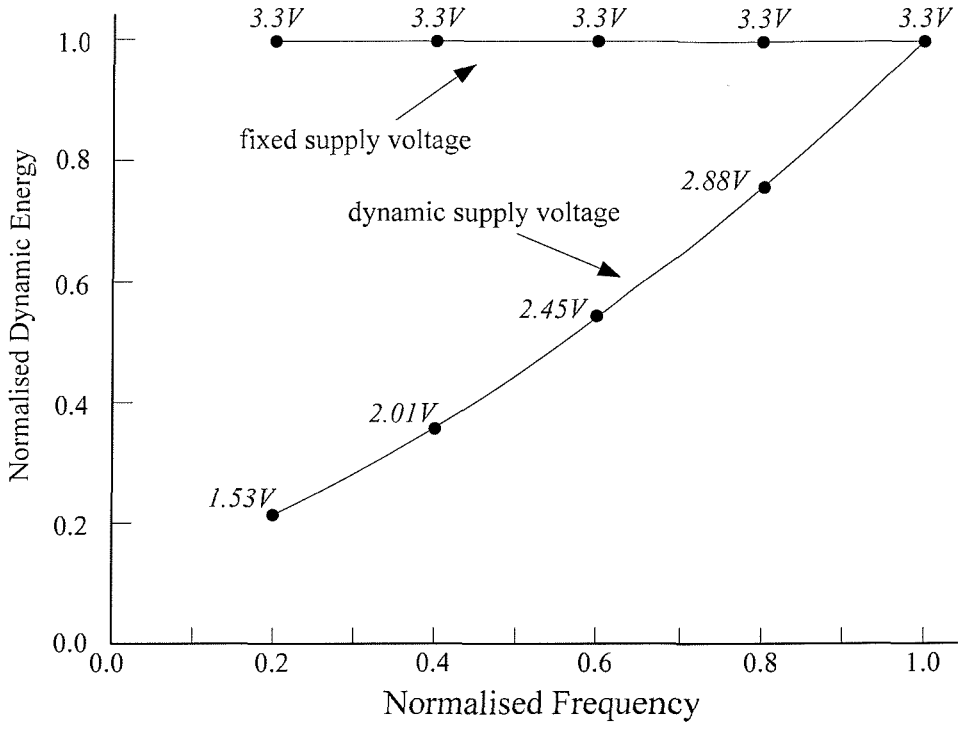


Figure 2.5: Energy vs. Frequency using fixed and dynamic supply voltages [2]

temporal performance requirements of the application. According to Equations 2.9 and 2.11, Figure 2.5 shows the normalised dynamic energy dependent on the normalised clock frequency for two cases: (1) keeping the supply voltage fixed and (2) adjusting the supply voltage dynamically in corresponding to the clock frequency. First consider the curve of fixed V_{dd}, starting at the nominal supply voltage of 3.3V, when clock frequency is reduced, there is no reduction in dynamic energy. Now consider the curve of dynamic V_{dd}, as the clock frequency decreases, there is significant dynamic energy reduction at reduced V_{dd}. For example, when the normalised clock frequency is 0.4, the normalised dynamic energy for a fixed V_{dd} of 3.3V is 1. However, the normalised dynamic energy is reduced to 0.37 if V_{dd} is dynamically adjusted to 2.01V. From the above discussion, it can be seen that dynamic energy can be reduced in a quadratic manner by executing the tasks with lower supply voltage, at the cost of longer computational time. DVS reduces the dynamic energy when the performance requirement is low, while retaining peak throughput when requested. If a majority of the tasks within an application does not

require peak throughput, then the dynamic energy of the application can be significantly reduced.

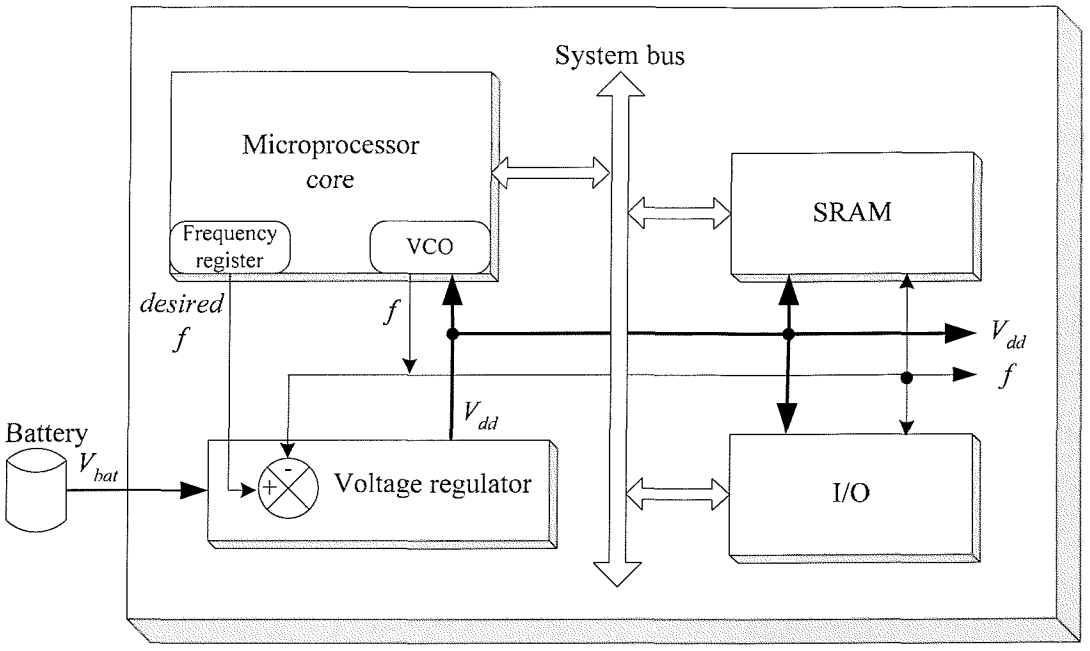


Figure 2.6: Block diagram of DVS-enabled processor [70]

There are three key components for implementing DVS in a processor: an operating system that can intelligently vary the processor operation frequency, a regulator loop that can generate the minimum voltage required for the desired operation frequency, and a processor that can operate over a wide range of voltage [70, 75]. Figure 2.6 shows a block diagram of a typical DVS-enabled processor [70]. It consists of four blocks: microprocessor core, SRAM, I/O and voltage regulator. The microprocessor core, SRAM and I/O are interconnected with a system bus. The clock frequency of the processor system is controlled by the operating system running on the microprocessor. The operation system determines a suitable clock frequency according to temporal performance requirement, and writes the desired frequency into the frequency register, which is then fed to the voltage regulator. Unlike conventional voltage regulator which samples the output voltage and compares it to an input reference voltage within a negative feedback loop, in the voltage regulator within the DVS-enable processor system, the output voltage drives the VCO (voltage controlled oscillator), which generates the clock frequency in corresponding to the voltage. The

voltage regulator compares the clock frequency generated by the VCO with the desired frequency, and regulates the output voltage in corresponding to the frequency error. If the clock frequency is higher than the desired frequency, the regulator reduces the output voltage. Otherwise if the clock frequency is lower than the desired frequency, the regulator increases the output voltage. The voltage regulator and the VCO also supplies supply voltage and clock frequency to SRAM and I/O. This approach allows the operating system to directly set the clock frequency of the processor system, and lets the hardware loop (voltage regulator and VCO) determine the supply voltage to meet this desired frequency.

Several technologies implementing DVS-enabled processor [70, 75, 78] have been reported and showed that energy dissipation can be reduced significantly. Recently, various chip manufactures have also marketed processors with DVS capability. Transmeta introduced the Crusoe Processor integrated with a DVS technique – LongRun [79]. For example, the Crusoe TM5900 can run between (667MHz, 0.8V) – (1GHz, 1.4V), consuming 6.5W – 9.5W power. AMD introduced DVS based processors – PowerNow [80]. Intel also has DVS based processors – XScale [81].

2.1.2.2 Adaptive Body Biasing

As outlined in Figure 2.3, leakage power reduction is becoming an important issue that needs to be addressed. An effective technique recently introduced is adaptive body biasing (ABB) [60-62, 64]. Adaptive body biasing (ABB) is a run-time leakage reduction technique which employs variable threshold CMOS. ABB utilises dynamic adjustment of clock frequency and threshold voltage through body bias control depending on the workload of the processor. When the workload is low, the threshold voltage is adaptively changed to a higher value via changing the body bias voltage. This will reduce leakage current exponentially (Equation 2.16) and at the same time deliver just enough amount of performance required for the current workload. This technique is similar to DVS, which is effective when the dynamic power is dominant.

On the other hand, ABB is effective for deep submicron circuits, where leakage power is an important component of the total power consumption.

Figure 2.7 shows a block diagram of a possible ABB scheme. It is a negative feedback loop consisting of two key blocks: body biasing controller and VCO (voltage controlled oscillator). The operating system running on the processor determines a desired frequency according to temporal performance requirement, and feeds it into the loop. Body biasing controller compares the desired frequency with the actual clock frequency, and regulates the output body bias voltage corresponding to the frequency error. VCO generates the clock frequency in corresponding to the threshold voltage, which is determined by the body bias voltage. If the clock frequency is higher than the desired frequency, the controller decreases the body bias voltage, which in turn increases threshold voltage, thus results in lower clock frequency. Otherwise if the clock frequency is lower than the desired frequency, the regulator increases the body bias voltage. Several technologies implementing ABB [60-62, 64] have been reported and show that leakage power can be reduced significantly. However, there are some overheads of implementing ABB. To make the body bias voltage control available, besides body biasing controller and VCO, the substrate bias lines are required to be interconnected separately to the power lines, thus creating additional wiring.

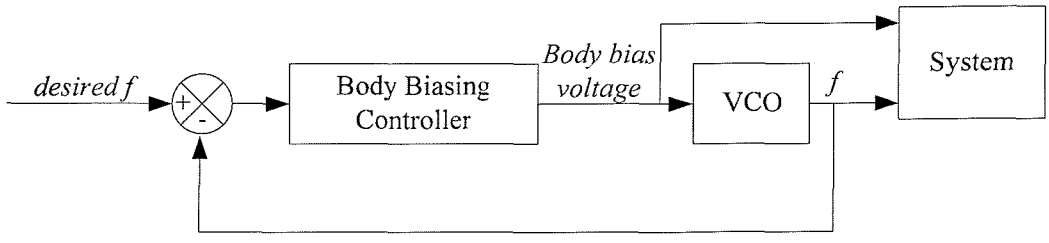


Figure 2.7: Block diagram of adaptive body biasing (ABB) scheme [64]

2.2 Previous Work

Embedded system co-synthesis (Chapter 1, Section 1.4) is a methodology aiming to find a suitable implementation for a given system specification. The degree of suitability is measured in terms of cost, performance, energy efficiency, etc. This

section gives a brief review of previous research in the co-synthesis of energy-efficient embedded systems. There are other literature reviews carried out in Chapters 3, 4 and 5.

The early work in co-synthesis [4, 9, 12] focused on cost and performance. Two important early co-synthesis tools are VULCAN [28, 82, 83] and COSYMA [84-86], which partition the system specification into hardware and software (i.e., hardware-software partitioning). Numerous research has also been carried out in the area of application mapping [84, 87-90] and activity scheduling [3, 14, 91-94]. However, the recent development of the portable application market has intensified the interests in co-synthesis techniques for energy-efficient embedded systems. The pioneer research targeting the reduction of energy dissipation throughout the co-synthesis process was proposed by Dave *et al* [14]. Their algorithm, namely COSYN, targets distributed heterogeneous embedded systems executing a set of acyclic task graphs. The mapping is produced in such a way that not only the timing and cost but also energy dissipation is taken into account. Energy reduction is achieved based on the fact that implementing tasks on hardware execute faster and with less power. A multi-objective genetic algorithm for hardware-software co-synthesis of embedded systems is presented by Dick and Jha [88], where cost and power are optimised while hard real-time constraints are met.

Dynamic Power Reduction

Due to the emergence of DVS and the resulting high dynamic power reduction, embedded system co-synthesis research started to investigate the possibility of integrating DVS. In order to employ DVS, it is necessary to identify appropriate voltages for the task executions so that the available idle and slack time (Chapter 1, Section 1.4.4) can be exploited effectively. Initial research on DVS for co-synthesis was carried out for single processing element (PE) systems. DVS was investigated in the context of non-real-time applications [95, 96], where system traces of workloads are analysed to evaluate energy saving with simple energy/delay models. A foundation was presented by Pering *et al* [97] for the simulation and analysis of DVS algorithms for non-real-time application.

A large number of reported research was carried out for single PE systems executing real-time applications. Ishihara and Yasuura [53] proposed some guideline to solve the voltage scheduling problem, which was formulated as an integer linear programming (ILP) problem to find the optimal solution. To minimise energy dissipation, it shows that, for a PE with continuously variable voltages, a unique voltage should be used for each task to expand its execution to its deadline; while for a PE with a number of discretely variable voltages, at most two voltages need to be used to execute each task. Hong *et al* [52] used a heuristic technique to determine the best scheduling and voltage level. Their synthesis technique also addresses the selection of the PE core and the determination of the instruction and data cache size so as to fully exploit DVS, which results in significantly energy reduction. The fixed priority scheduling, widely used in hard real-time system design, was extended to an energy-efficient version by Shin and Choi [98]. Their method obtains an energy reduction for a PE by exploiting the slack times inherent in the system or arising from variations of execution times of tasks instances. They presented a run-time mechanism to use these slack times for energy reduction, either by transiting the PE into power-down mode or changing the supply voltage and clock frequency dynamically. The problem of determining the optimal voltage scaling for a real-time system with fixed-priority jobs implemented on a variable voltage PE was proposed by Quan and Hu [54]. Two algorithms were presented in that paper. The first one finds the minimum constant operation frequency needed to complete the whole set of jobs; the second one is built on the first one and produces a voltage schedule which results in lower energy consumption. A power optimisation for priority-based pre-emptive scheduling was proposed by Shin *et al* [99]. The method combines off-line and on-line algorithms. The off-line algorithm determines the lowest possible nominal PE operation frequency while guaranteeing deadlines of all tasks. The on-line algorithm dynamically varies the PE operation frequency or brings a PE into a power-down mode according to the status of task set in order to exploit execution time variations and idle intervals. Lee and Sakurai [100] presented an online voltage scaling scheme. It exploits slack time arising from workload variation by partitioning a task into several timeslots and performing online voltage control on timeslot-by-timeslot basis. Their scheme can be easily applied to various targets, by employing software feedback control of supply

voltage and device driver from physical measurement of voltage-frequency relationship.

Numerous embedded systems are multi-PE systems and the previously reported works on DVS techniques for multi-PE systems are discussed next. A hybrid search strategy based on simulated heating [101] was presented by Bambha *et al* [102]. They search the optimal voltage levels for all the tasks executing on a multi-PE systems. Other heuristic techniques have been proposed to get sub-optimal solutions [55, 58, 59]. Luo and Jha [58] presented a power-conscious algorithm for jointly scheduling periodic task graphs and aperiodic tasks. In their technique, periodic task graphs are scheduled statically. Slots are created in this static schedule to accommodate hard aperiodic tasks (aperiodic tasks with hard deadlines). Soft aperiodic tasks (aperiodic tasks with soft deadlines) are scheduled dynamically with an online scheduler. The online scheduler employs a statistical process to predict the next arrival time of soft aperiodic tasks. Using the predicted arrival times and the static schedule, the online scheduler can predict the next idle time for the PE or predict the actual available processing time for a scheduled task, and perform appropriate DVS or DPM. Gruian and Kuchcinski [59] introduced a dynamic list scheduling heuristic, which supports DVS by making the priority function energy-aware. The energy sensitive task priorities are re-calculated at each scheduling attempt. If a scheduling attempt fails by exceeding the hard deadline, the priority function is adjusted and another scheduling attempt is carried out. Schmitz and Al-Hashimi [55] extended the system model by considering the variation of power consumption. They proposed an efficient DVS algorithm, which splits the slack time into slices, and iteratively distributes a slice to a selected task in such a way so that a maximum energy saving is achieved by using the slack time slice. In their recent papers [57, 103], the DVS algorithm [55] was combined with mapping and scheduling to form an energy-efficient co-synthesis. They employed genetic algorithm guided by energy sensitive objective functions to optimise the mapping and scheduling towards energy efficiency.

Leakage Power Reduction

As outlined in Figure 2.3, leakage power will become an important power consumption component for < 0.1μm CMOS technology [74]. The importance of leakage power reduction has been recognised by the research community [74, 76, 77, 104, 105].

Three possible approaches to reduce leakage power [106, 107] in circuit level have been reported: input vector control, supply voltage gating, and increasing the threshold voltage. Input vector control is based on the fact that, the standby leakage power of a circuit unit varies depending on the input vector, which determines the number of transistor stacks in the unit with more than one off-state transistor [104, 108]. The goal of input vector control is to find the input vector that minimise the leakage power [71, 109, 110]. Once this input vector is found, the input to the unit can be switched to this vector when the unit is in standby mode (i.e., the unit is not carrying out useful activities). Supply voltage gating shuts down the power supply so that idle units do not consume leakage power. This can be done using sleep transistors to cut the path from the power supply to the units [110-112]. For increasing the threshold voltage, there are different implementations, all of them involve some process technology support to change the threshold voltage of some (or all) transistors from the default defined for the technology: (1) In dynamic threshold voltage MOSFET (DTMOS) [113], the threshold voltage of the transistor is a function of its gate voltage, i.e., the threshold voltage is high whenever the transistor is off, therefore the leakage power of the transistors in off-state is reduced; (2) Dual Threshold CMOS [72, 114] uses low threshold devices in the critical path while high threshold devices in the non-critical path, therefore the leakage power on the non-critical path is reduced; (3) In Variable threshold CMOS (VTCMOS), the threshold voltage of a circuit unit can be scaled dynamically during runtime by adaptive body biasing (ABB) [60-64].

Khouri and Jha [72] proposed an RTL level leakage power analysis and reduction technique for behavioural synthesis. The technique identifies the frequently idle modules in the data-path, which are targeted for leakage optimisation. The leakage

optimisation is based on the use of dual threshold voltage CMOS technology, i.e., using high threshold voltage devices in the frequently idle modules, while low threshold voltage devices in the other modules. A combined DVS and ABB method was employed in a system level co-synthesis of embedded system [115]. The co-synthesis technique targets single PE system and reduce dynamic and leakage power simultaneously. Analytical models are developed to produce optimal supply voltage and threshold voltage values for energy minimisation.

2.3 Motivations

There is little doubt that the demand of energy-efficient embedded systems will continue to increase in the future. Therefore, new embedded system co-synthesis techniques considering the dynamic and leakage power reduction will be needed.

1. Energy-efficient co-synthesis of data and control dominated embedded systems

Dynamic voltage scaling (DVS, Section 2.1.2.1) technique is an effective technique that can be employed in system level co-synthesis to reduce the dynamic power. A number of recently reported research [52-59] have employed DVS in the co-synthesis of embedded systems to achieve energy efficiency, where purely data-dominated systems are considered. However, embedded systems, particularly real-time applications (e.g., communication or process control systems), often contain both data and control flows [116]. Control flows express the behaviour, where some parts of the system functionality execute only if specific control conditions are met. This aspect has been recognised by the research community and several system level representations have been proposed to capture both the data and control flow at task level [41, 42, 116-118]. Such an abstract system representation, conditional task graph (CTG), has been defined and a scheduling algorithm has been proposed so that the worst case delay is minimised [41, 42]. Xie and Wolf [119] presented a technique performing mapping and scheduling simultaneously for CTGs, which takes advantage of the resource sharing among mutual exclusive tasks. Such accurate system representations also offer potential of more efficient system implementations in terms

of energy efficiency. Nevertheless, the above works [41, 42, 116-118] have been conducted without the consideration of energy. No research has yet addressed the problem of energy minimisation of data and control dominated embedded systems. This forms the first motivation of this research. This thesis investigates energy minimisation techniques for the co-synthesis of embedded systems, whose representations capture both data and control flows (Chapter 3).

2. Influence of communications on energy-efficient co-synthesis

Communications have important impact on the design of multi-PE embedded systems, which have drawn much attention from the research community [42, 120-125]. The time and energy overheads of communication significantly influence the quality of embedded system designs in terms of timing feasibility and energy efficiency. Therefore, the second motivation of this research is to integrate communications with the co-synthesis techniques of Chapter 3 by using enhanced system models (Chapter 4). Furthermore, a performance analysis is carried out to investigate the effect of alternative communication architectures on system quality in terms of energy efficiency.

3. Simultaneous dynamic and leakage power reduction

As outlined in Figure 2.3, leakage power will become an important power consumption component in deep sub-micron designs [74]. ABB has been recently proposed to effectively reduce leakage power [60-64], most of which focus in the circuit level. There is little research reported in the system level, apart the one reported by Martin *et al* [115], where combined DVS and ABB techniques have been proposed for embedded systems to reduce dynamic and leakage power simultaneously. The technique [115] has the disadvantage of increased cost of complexity due to the implementation of combined DVS and ABB technique. Future work is needed in the system level co-synthesis of embedded system for dynamic and leakage power reduction, taking cost into consideration. This forms the third motivation of this research.

2.4 Concluding Remarks

This chapter introduces the sources of power consumptions within embedded systems, and outlines two main energy management techniques applicable in embedded system co-synthesis, namely dynamic voltage scaling (DVS) and adaptive body biasing (ABB), which are effective in reducing dynamic and leakage power respectively. An overview of the previous work in the co-synthesis of energy-efficient embedded systems is given. There is little doubt about the demand for new co-synthesis methods for energy-efficient embedded systems. Embedded systems, particularly real-time applications, often contain both control and data flows, which need special co-synthesis techniques to improve the energy efficiency. Embedded systems are heterogeneous and distributed systems, the influences of communications on the design quality in terms of timing feasibility and energy efficiency need to be considered. Furthermore, most recent research in embedded systems focused on dynamic power reduction. However, leakage power is becoming important in deep sub-micron designs, therefore new cost-effective techniques capable of simultaneous reducing the dynamic and leakage power of embedded systems need to be investigated. All the above form the motivation of the proposed research in this thesis.

Chapter 3

Conditional Behaviour Aware DVS Based Co-Synthesis

Dynamic voltage scaling (DVS) is an effective technique to reduce the dynamic power of processing elements (PEs) within embedded systems. DVS dynamically scales the supply voltage and the clock frequency in response to the temporal performance requirements, therefore, reduces the energy dissipation when the performance requirement is low, while retaining peak performance when requested. Previous DVS techniques for embedded systems have focused on purely data dominated systems [52-59]. However, embedded systems often contain both data and control flows, particularly for real-time applications [116]. Control flows express the behaviour, where some parts of the system functionality execute only if specific control conditions are met. This aspect has been recognised by the research community and several system level representations have been proposed to capture both the data and control flow at task level [41, 42, 116-118]. Using such system representations allows a more accurate modelling for a wide range of applications, which will lead to more efficient system implementations. Based on such consideration, some research has addressed the scheduling and mapping of embedded systems expressed as conditional task graphs (CTGs, Chapter 1, Section 1.3.2) or similar representations [41, 42, 119], with the aim of minimising the worst case delay. Such an accurate system representation also offers the potential of efficient implementations in terms of energy efficiency. Nevertheless, no work has yet addressed the problem of energy minimisation during co-synthesis of system specifications which capture both dataflow and the flow of control. This chapter proposes a new DVS technique for data and control dominated embedded systems, which is capable of improving the energy efficiency taking into account the conditional behaviour of the systems. This chapter also develops a mapping technique for better exploitation of the proposed DVS techniques for further energy reduction.

The remaining of this chapter is organised as follows. Section 3.1 outlines the preliminaries including the concept of the schedule table and the principles of genetic algorithms. Section 3.2 gives an overview of the proposed energy minimisation techniques for data and control dominated embedded systems, including a conditional behaviour aware DVS technique and an energy-efficient mapping technique, which are detailed in Sections 3.3 and 3.4 respectively. Extensive experimental results are presented in Section 3.5. Finally, concluding remarks are given in Section 3.6.

3.1 Preliminaries

This section presents the power and delay models used in the conditional behaviour DVS based co-synthesis for data and control dominated embedded systems. The concept of schedule table is introduced. Finally, the key principles of genetic algorithms are outlined.

3.1.1 Power and Delay Models

In order to apply DVS to the embedded systems, this section derives the power and delay models. The models include equations for the calculation of energy dissipation, execution time, and supply voltage. As outlined in Chapter 2, Section 2.1.2.1, DVS decreases energy dissipation by slowing down the clock frequency of PEs, which in turn results in longer execution time of tasks. Consider a PE executing a task τ. Assume the nominal supply voltage and the threshold voltage of the PE is V_{nom} and V_{th} respectively, the energy dissipation required to execute τ at V_{nom} is $E_\tau(V_{nom})$. The energy dissipation required to execute τ at V_{dd} can be derived from Chapter 2, Equation 2.9:

$$E_\tau(V_{dd}) = \frac{V_{dd}^2}{V_{nom}^2} \cdot E_\tau(V_{nom}) \tag{3.1}$$

Assuming the execution time of τ at V_{nom} is $d_\tau(V_{nom})$, the execution time of τ at V_{dd} can be derived from Chapter 2, Equation 2.11:

$$d_\tau = \frac{(V_{nom} - V_{th})^2}{V_{nom}} \cdot \frac{V_{dd}}{(V_{dd} - V_{th})^2} \cdot d_\tau(V_{nom}) \tag{3.2}$$

According to Equation 3.2, $V_{dd}(d_\tau)$, the supply voltage corresponding to d_τ is given by:

$$V_{dd}(d_t) = V_{th} + \frac{b}{2a} + \sqrt{\left(V_{th} + \frac{b}{2a}\right)^2 - V_{th}^2} \tag{3.3}$$

where

$$a = \frac{d_\tau}{d_\tau(V_{nom})} \tag{3.4}$$

$$b = \frac{(V_{nom} - V_{th})^2}{V_{nom}} \tag{3.5}$$

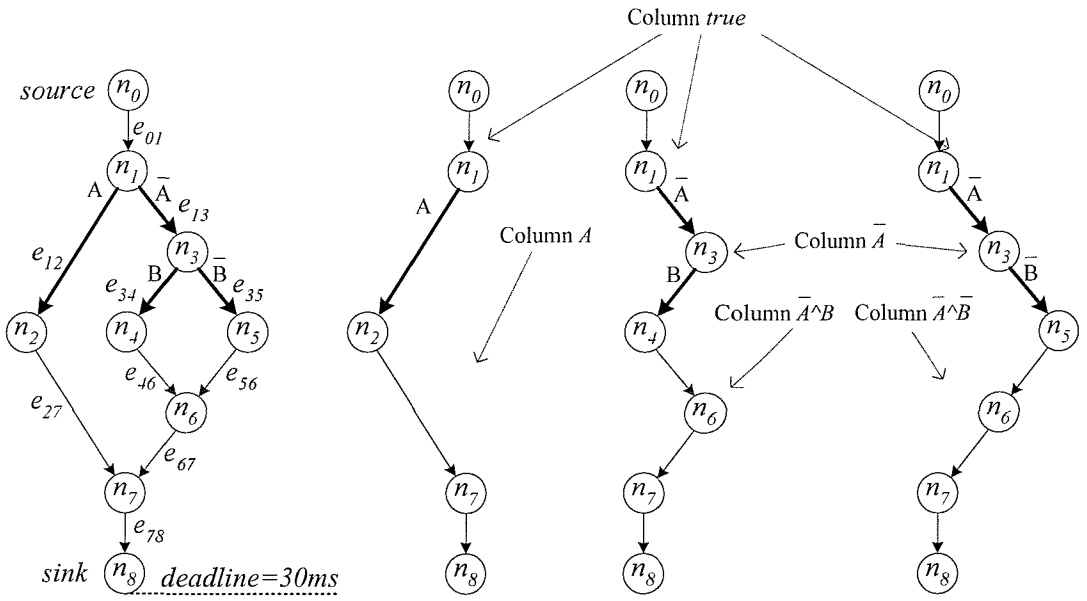
Equations 3.1 – 3.5 will be used in the conditional behaviour aware DVS technique in Section 3.3.

3.1.2 Schedule Table

In Chapter 1, Section 1.3.2, the principles of conditional task graphs (CTGs) were outlined. In this section, additional information in terms of the conditional behaviour of CTGs is given, in order to explain how to present the schedules of different tracks within the CTGs. An example is used to illustrate the concept of schedule table.

As stated earlier, an application is specified as a conditional task graph (CTG). At a given execution of a CTG, depending on the condition values produced by the disjunction tasks, only a certain subset of the total tasks (track) is executed, and this subset differs from one execution to the other. For example, Figure 3.1(a) shows a CTG, where tasks n_1 and n_3 are disjunction tasks. Depending on the condition values produced by tasks n_1 and n_3, there exist three possible tracks as shown respectively in Figure 3.1(b) – (d): (1) track 1 of Figure 3.1(b) will be followed if n_1 produces A; (2)

track 2 of Figure 3.1(c) will be followed if n_1 produces $\overline{A}$ and n_3 produces B; (3) track 3 of Figure 3.1(d) will be followed if n_1 produces $\overline{A}$ and n_3 produces $\overline{B}$.



(a) conditional task graph (b) track 1 (c) track 2 (d) track 3

Figure 3.1: Example of a conditional task graph and its tracks

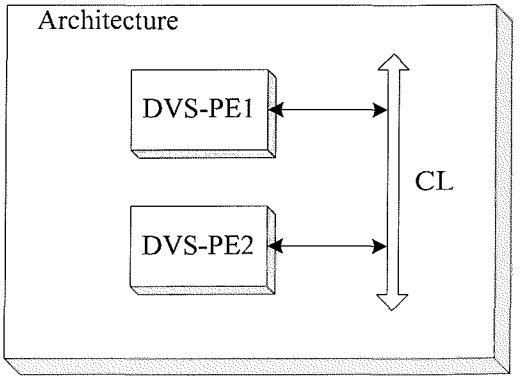


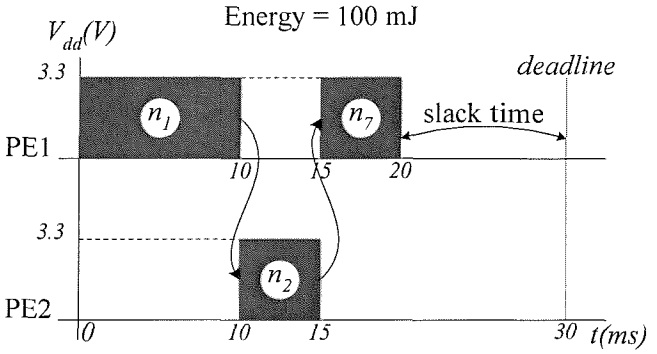
Figure 3.2: Example of architecture

For a mapped CTG (i.e. each task in the CTG has been mapped onto a certain PE of the hardware architecture), there exists an optimal schedule for each individual track which produces a minimal delay. For example, consider the CTG of Figure 3.1(a), assuming this CTG has been mapped to the architecture of Figure 3.2, the

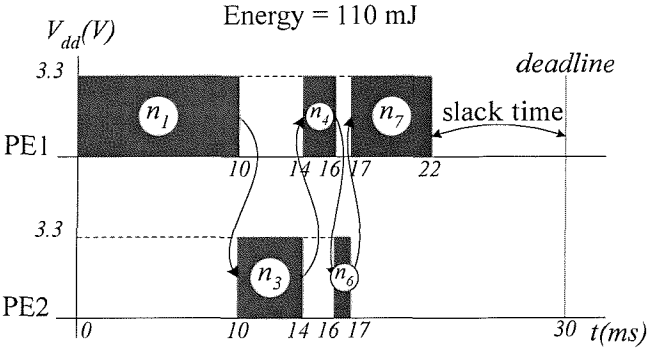
mapping and the corresponding execution times of the tasks are as shown in Table 3.1. Note that n_0 and n_8 are not considered in Table 3.1 and the following discussion, because they are dummy tasks which have zero execution time and are not mapped to any PE (Appendix C). Figure 3.3 shows the schedules of the three tracks of the CTG. As it can be seen, the start times of tasks vary depending on the condition values produced in a specific execution, which determine the track to follow. For example, if track 1 is followed, the schedule starts n_7 at 15ms (Figure 3.3(a)), while if track 2 is followed, the schedule starts n_7 at 17ms (Figure 3.3(b)), finally, if track 3 is followed, the schedule starts n_7 at 21ms (Figure 3.3(c)). Eles *et al* [41, 42] presented the schedule table concept to present the various start times of the tasks under all possible condition values. Table 3.2 is a schedule table corresponding to the schedules of Figure 3.3. Although the original schedule table [41, 42] specifies only the start times of the tasks. In this research, the schedule table is extended to have the end times as well. This is in order to present the clock frequency and, explicitly, the supply voltages (Equations 3.3 – 3.5) at which the tasks are executed. Examining Table 3.2 shows that the table has one row for each task, which contains start and end times for that task corresponding to different condition values. Each column in the table is headed by a logical expression constructed as a conjunction of condition values. Start and end times in a given column under the logical expression represent the activation and termination times of the tasks when the respective expression is true. The start times and end times in the column *true* are for those tasks executed initially.

Task	Mapping	Execution time (ms)
n_1	PE1	10
n_2	PE2	5
n_3	PE2	4
n_4	PE1	2
n_5	PE1	6
n_6	PE2	1
n_7	PE1	5

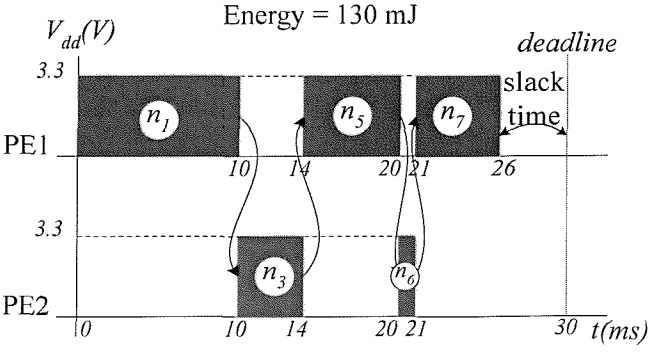
Table 3.1: Task mapping and task execution times



(a) Schedule of track 1



(b) Schedule of track 2



(c) Schedule of track 3

Figure 3.3: Schedules for the CTG of Figure 3.1(a) (V_{th}=0.8V, P=5W)

	true	A	$\overline{A}$	$\overline{A}\wedge B$	$\overline{A}\wedge\overline{B}$
n_1	0→10				
n_2		10→15			
n_3			10→14		
n_4				14→16	
n_5					14→20
n_6				16→17	20→21
n_7		15→20		17→22	21→26

Table 3.2: Schedule table for the CTG of Figure 3.1(a)

The schedule table captures a schedule of the system specified by the CTG considering the given task mapping. This means that all decisions that could be taken off-line have been made by the scheduling algorithm and are written into the schedule table. Based on this information, the run-time kernels running on each PE will take the actual decision on activation of tasks, based on the current values of condition. For example, the CTG of Figure 3.1(a) has three tracks as shown in Figure 3.1(b) – (d). The schedule of track 1 is represented in columns *true* and A; the schedule of track 2 is captured in columns *true*, $\overline{A}$ and $\overline{A}\wedge B$. The schedule of track 3 is given in columns *true*, $\overline{A}$, and $\overline{A}\wedge\overline{B}$. Considering track 2, the actual decisions on the activation of tasks are made as follows: (1) n_1 is executed initially from 0 to 10; (2) if n_1 produces condition value $\overline{A}$, the run-time kernel searches column $\overline{A}$ of the schedule table, and executes n_3 from 10 to 14; (3) if n_3 produces condition value B, the run-time kernel searches column $\overline{A}\wedge B$, and executes n_4 from 14 to 16, n_6 from 16 to 17, n_7 from 17 to 22. It is clear that the actual schedule depends on which track is followed.

3.1.3 Genetic Algorithms

Genetic algorithms (GAs) [126, 127] have been the subject of numerous research in the last decades, and they have been proven to solve different optimisation problems successfully [88, 128]. In this section, the basic concept and principle of genetic algorithms are explained.

Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics. Figure 3.4 shows a general flow of a genetic algorithm.

Genetic algorithm maintains a population, which is a pool of solutions. Each solution in the pool is associated with an objective function value. The objective function is a measure of what the genetic algorithm attempts to maximise or minimise, e.g., cost, execution delay, and energy dissipation. The genetic algorithm (1) initialises a population; (2) evaluates the objective function value for each solution in the population; (3) ranks solutions according to the objective function values; (4) if the optimisation objective is met, terminates the optimisation, otherwise passed the ranked population to evolvement; (5) evolves the population by removing the low-ranked solutions (i.e., the solutions with lower quality) from the population and applying genetic operator to the high-ranked solutions (i.e., the solutions with high quality); (6) repeats steps (2) – (5) to the new generation of population. This procedure repeats until a certain optimisation objective is met.

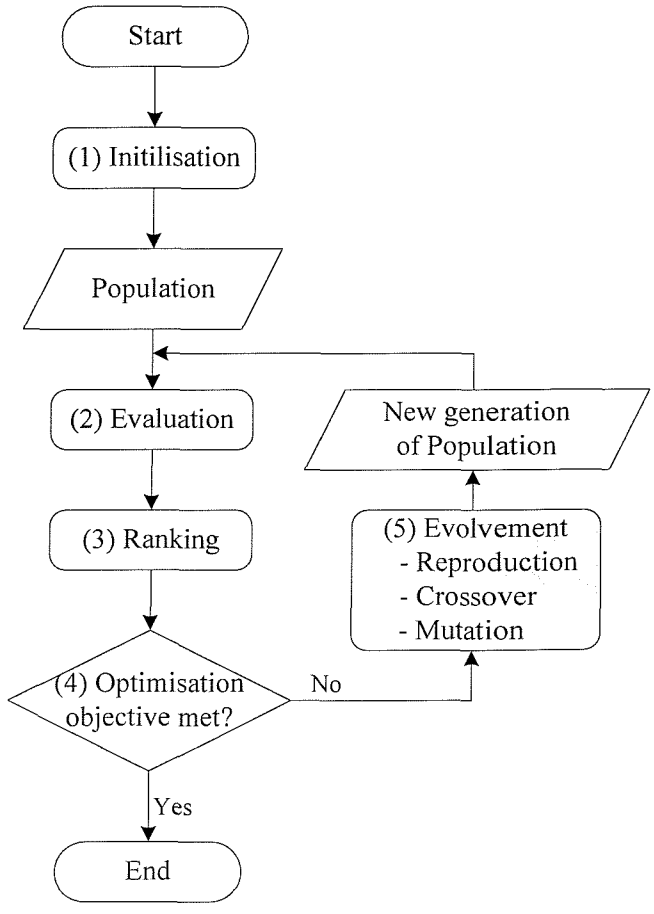


Figure 3.4: General flow of a genetic algorithm

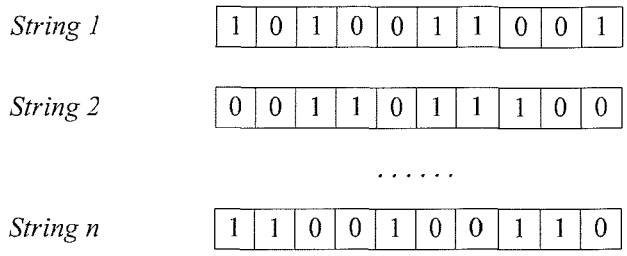


Figure 3.5: Solution pool of genetic algorithm

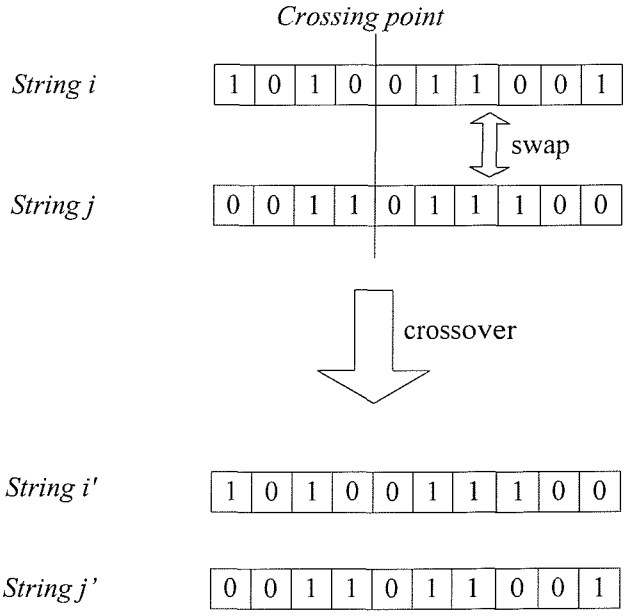


Figure 3.6: Genetic operator: crossover (mating)

In a genetic algorithm, each solution is represented by a string of values, as shown in Figure 3.5. The evolvement of the solutions is brought by three genetic operators: reproduction, crossover (mating), and mutation. Reproduction is a process in which individual strings are copied according to their objective function values. Strings with higher values have a higher probability of contributing one or more offspring in the next generation. After reproduction, crossover swaps portions of different strings at random. Figure 3.6 shows an example of string crossover between *String i* and *String j*. The portions right to the crossing point are swapped, producing *String i'* and *String j'*. The mechanics of reproduction and crossover are simple, involving random number generation, string copies, and partial string exchanges. The combination of reproduction and crossover gives genetic algorithm much of its power

to achieve optimisation objective. Another genetic operator is mutation, which changes one or more digit values in a string at random. Mutation is needed because, even though reproduction and crossover are effective, occasionally they may lose some potentially useful genetic material [126].

3.2 DVS based Co-Synthesis Overview

The application of DVS techniques for task scheduling is based on the assumption that a certain slack time is available and this slack is also predictable, at least to a certain extent, at design time. In the case of system specifications which capture both the data flow and control flow, as in the case with conditional task graphs (CTGs), constructing a schedule with voltage scaling is more difficult than for pure data flow systems, due to the additional problems related to the prediction of slacks. The values of the conditions are unpredictable, so the decision on how much slack time can be distributed to a task is taken without knowing which values the downstream conditions will later get, i.e., the execution path is determined incrementally during run-time. On the other side, at a certain moment during execution, when the values of some conditions are already known (upstream conditions), they have to be used in order to take the best possible decisions.

A new conditional behaviour aware DVS technique for embedded systems expressed as CTGs is presented in Section 3.3, which is capable of exploiting the slack time taking into account the conditional behaviour of the system. The main goal of this new DVS technique is the identification of a voltage scaling such that, under any possible set of condition values, the deadlines are satisfied and, at the same time, energy dissipation is reduced as much as possible. Also, a genetic algorithm based mapping technique is introduced in Section 3.4 to optimise the task mapping to efficiently exploit the proposed DVS technique, hence, leading to further energy saving.

Example 1: Challenge of Conditional Behaviour Aware DVS

To illustrate the challenge of the conditional behaviour aware DVS technique for CTGs, consider the CTG of Figure 3.1(a). Assume that the deadline of the system is 30ms. Figure 3.3(a) – (c) show the schedules of the three possible tracks through the CTG, corresponding to the schedule table Table 3.2. The schedules are produced using the algorithm proposed by Eles *et al* [42], where the aim is to produce a schedule such that the worst case delay is as small as possible. Examining Figure 3.3 shows that the amount of slack time varies with the tracks, ranging from 10ms in the case of track 1 to 4ms in the case of track 3. Figure 3.3(a) – (c) also show the energy dissipation of each track, assuming PE1 and PE2 consume 5W power running at a supply voltage of 3.3V and a threshold voltage of 0.8V. For example, the energy dissipation of track 1 is 5W·(10ms + 5ms) + 5W·5ms = 100mJ. In order to make use of the DVS techniques for energy minimisation, one possible DVS based approach [53, 59] can be employed. This involves uniformly distributing the slack times between the tasks, and scaling the tasks to fit the imposed deadline. The scaling factor is the ratio between the deadline and the total length of the schedule:

$$f_{scaling} = \frac{deadline}{length\ of\ schedule} \tag{3.6}$$

For example, the scaling factor for track 1 is calculated by 30 / 20 = 1.5. Similarly, the scaling factors for track 2 and track 3 are 1.36 and 1.15 respectively. Scaling modifies the start and end times of the tasks, which are given by:

$$t_{start}' = t_{start} \cdot f_{scaling} \tag{3.7}$$

$$t_{end}' = t_{end} \cdot f_{scaling} \tag{3.8}$$

where t_{start} and t_{start}' are the start times before and after the scaling, t_{end} and t_{end}' are the end times before and after the scaling, $f_{scaling}$ is the scaling factor. Using Equations 3.6 – 3.8, scaling the schedules of Figure 3.3(a) – (c) generates the schedules of Figure 3.7(a) – (c). For example, in the scaled schedule of track 1 (Figure 3.7(a)), tasks n_1, n_2,

and n_7 are scaled with a scaling factor 1.5 to meet the deadline. In this case, the supply voltage at which n_1, n_2, and n_7 should be executed is calculated using Equations 3.3 – 3.5:

$$a = \frac{d(V_{dd})}{d(V_{nom})} = \frac{30}{20} = 1.5$$

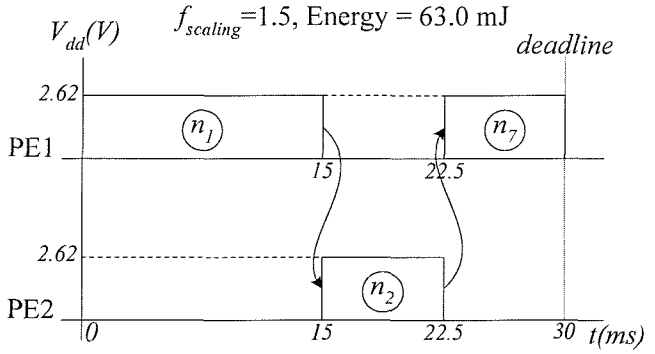
$$b = \frac{(V_{nom} - V_{th})^2}{V_{nom}} = \frac{(3.3 - 0.8)^2}{3.3} \approx 1.894$$

$$V_{dd} = V_{th} + \frac{b}{2a} + \sqrt{\left(V_{th} + \frac{b}{2a}\right)^2 - V_{th}^2} = 0.8 + \frac{1.894}{2 \cdot 1.5} + \sqrt{\left(0.8 + \frac{1.894}{2 \cdot 1.5}\right)^2 - 0.8^2} \approx 2.62V$$

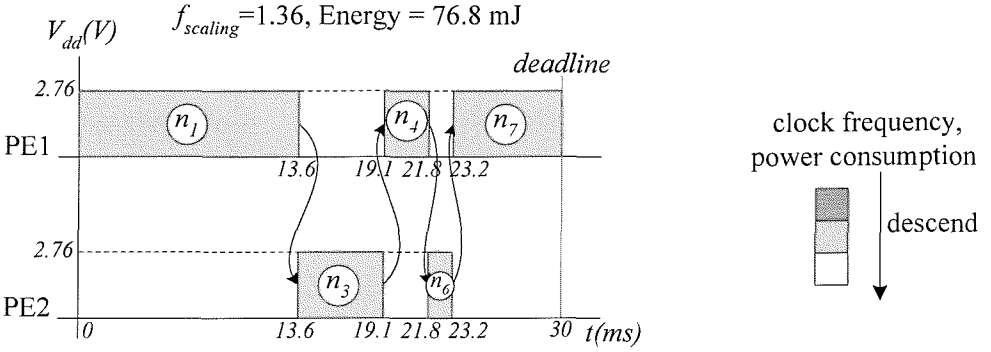
The energy dissipation of track 1 after scaling is calculated using Equation 3.1:

$$E_{track} = \sum_i E_{\tau i}(V_{dd}) = \sum_i \left(\frac{V_{dd}^2}{V_{nom}^2} \cdot E_{\tau i}(V_{nom}) \right) = \frac{2.62^2}{3.3^2} \cdot 100 = 63.0mJ \qquad (3.9)$$

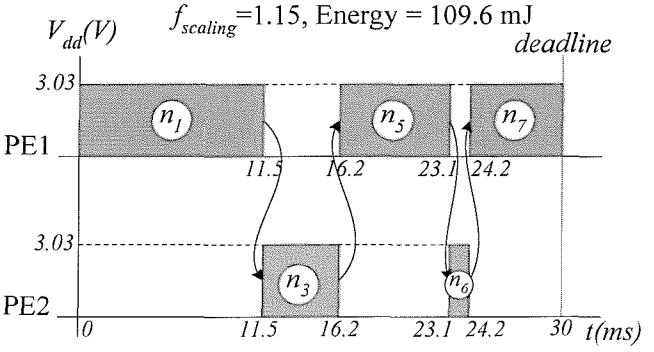
where i is the task number. As it can be seen from Figure 3.7, the scaling factor, and in turn, the supply voltage depends on the track to be followed. For example, the scaling factors for tracks 1, 2 and 3 are 1.5, 1.36 and 1.15 respectively. The scaled schedules of Figure 3.7 are generated assuming the track to be followed is known at advance. During a specific execution, however, the condition values of the CTG, and, in turn, which track will be followed are not known in advance. Therefore, the scaled schedules of Figure 3.7 are impracticable. For example, at the time point of 0, it is unknown that whether track 1, track 2, or track 3 will be followed, hence it is unknown which scaling factor should be employed. If the scaling factor and, implicitly, the supply voltage are decided upon improperly, the time constraints may be conflicted, which cannot be tolerated in systems with hard-real time properties. As shown in Figure 3.8, if n_1 is scaled with the scaling factor of 1.5, i.e., n_1 is executed from time 0 to 15, and the condition values produced by n_1 and n_3 come out to be $\overline{A}$ and $\overline{B}$ later, the deadline will be missed even if the remaining tasks run with the nominal supply voltage (3.3V).



(a) Scaled schedule of track 1



(b) Scaled schedule of track 2



(c) Scaled schedule of track 3

Figure 3.7: Schedule scaled for energy minimisation

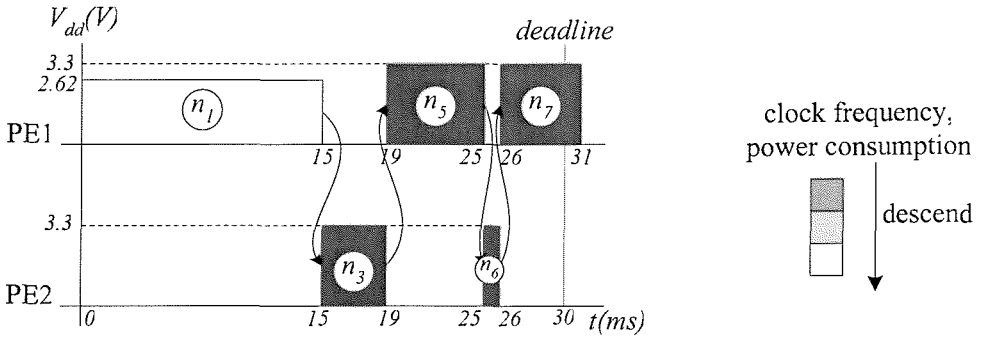


Figure 3.8: Improper scaling

The above example shows that, in order to exploit slack time as much as possible and at the same time meet time constraints, the worst case slack time (the maximum slack time that can be distributed to a task without later conflicting time constraints during upcoming scheduling decisions) should be identified dynamically and used to decide how much slack time a task can exploit.

3.3 Conditional Behaviour Aware DVS

In this section a new conditional behavioural aware DVS technique for conditional task graphs (CTGs) is described. The basic idea of the proposed DVS technique is to identify the available worst case slack time taking into account the conditional behaviour of CTGs. This is achieved by dynamically identifying the worst case track (a track with the longest delay), calculating the scaling factor (i.e. the ratio between the deadline and the total length of the schedule) and modifying the schedule table every time after a disjunction node (a node producing a condition value) has been scheduled.

3.3.1 Problem Formulation

The input of the proposed DVS technique is a conditional task graph (CTG) and a hardware architecture containing a number of DVS-enabled processing elements (PEs). The PEs can run at a range of supply voltages between the threshold voltage and the nominal supply voltage. The supply voltage can be scaled continuously within

the range, but it can be easily adapted to the case with discrete voltages (a number of potential supply voltages) [129]. An assumption is made that the tasks are of sufficiently coarse granularity and that the PEs can continue operation during the voltage scaling, which allows neglecting the scaling overhead in terms of power and time. Furthermore, the PEs might be shut down when they are idle. Each task in a CTG might have multiple implementation alternatives, i.e., it can be potentially mapped onto several PEs which are able to execute this task. For each possible task mapping, certain implementation properties, e.g. execution time and power consumption, are given in a technology library. These values are either based on previous design experiences or on estimation techniques [130-134]. The CTG has been mapped onto the architecture and a schedule table has been generated by the scheduling technique presented by Eles *et al* [42] whose aim is to make the worst case delay as small as possible. The output of the proposed DVS technique is a slack time exploited schedule table indicating activation times and voltage levels such that deadlines are satisfied and at the same time energy dissipation is reduced. In order to concentrate on the specific aspects of importance for this chapter, a simplifying assumption is made that communications between tasks take zero time and consumes zero power. Impact of communications will be discussed in Chapter 4.

3.3.2 Proposed Technique

The strategy of the proposed conditional behaviour aware DVS technique is based on the idea to exploit the information of condition values available when a disjunction node ends, in order to apply the largest possible scaling factor while still guarantee the deadline. The point in time when additional information concerning the future evolution of the system becomes available is the moment when a disjunction node ends. Therefore, at the beginning of the scheduling process, a more conservative scaling factor is applied. Once a disjunction node has been scheduled and, as a result, more available slack time can be identified, a higher scaling factor should be applied. Thus the schedule of a CTG is divided into several scaling regions by the end times of the disjunction nodes. Each scaling region is then scaled with a certain, suitable scaling factor.

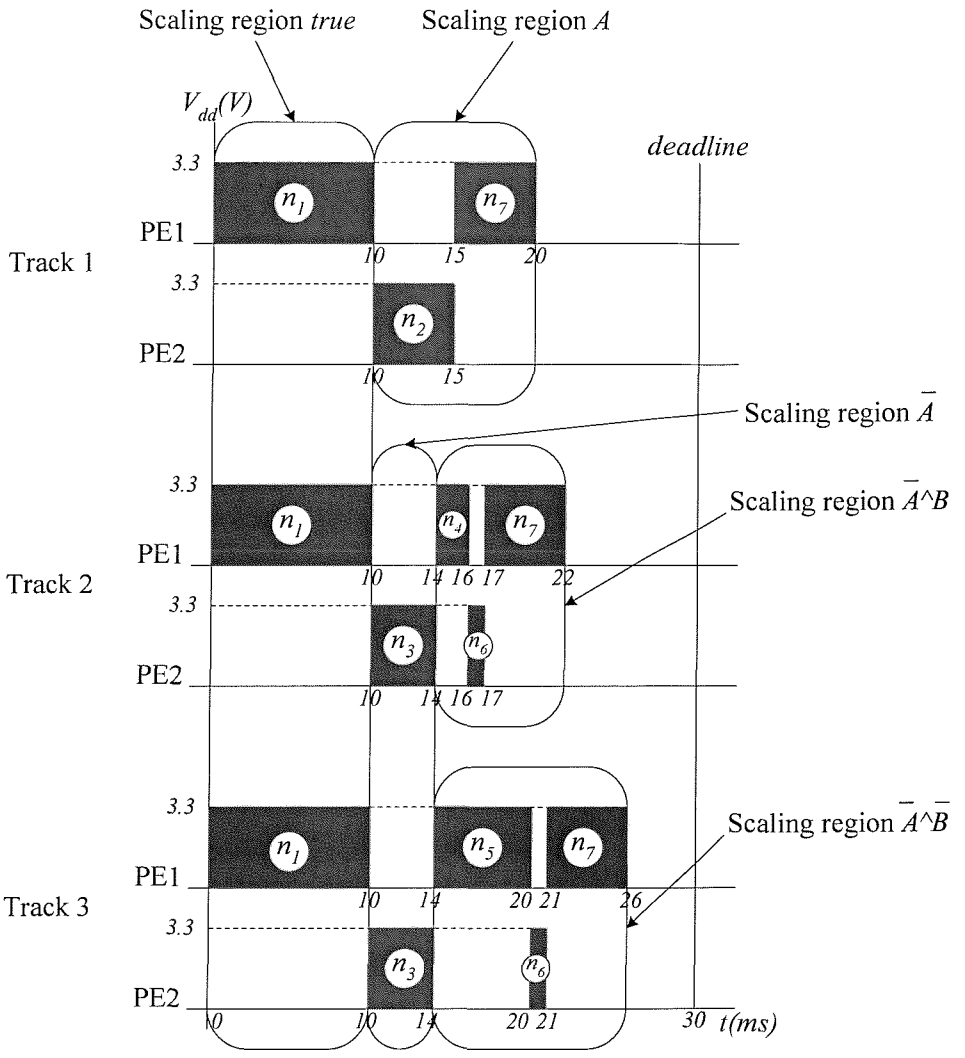


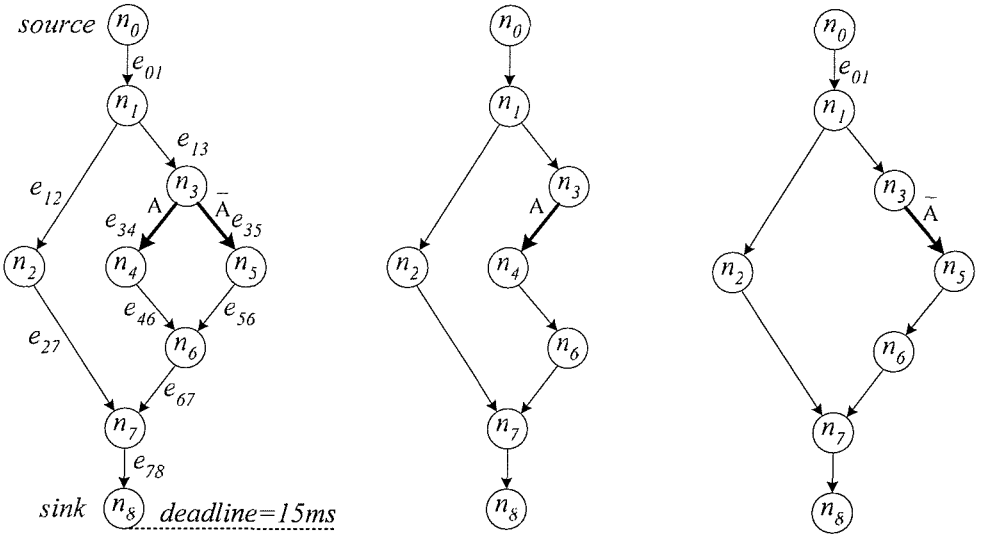
Figure 3.9: Scaling regions vs. columns of schedule table

Figure 3.9 shows the scaling regions for the schedule table Table 3.2. Since the disjunction node n_1 ends at time 10, the schedules of the three tracks are same from time 0 to time 10, which corresponds to scaling region *true*. If n_1 produces condition value A, track 1 is followed. In this case, the schedule of track 1 from time 10 to 20 corresponds to scaling region A. Otherwise if n_1 produces condition value $\overline{A}$, track 2 or track 3 is followed depending on the condition value produced by n_3. In this case, the schedules of track 2 and track 3 are same from time 10 to time 14, which corresponds to scaling region $\overline{A}$. Similarly, scaling regions $\overline{A}\wedge B$ and $\overline{A}\wedge\overline{B}$ can be identified as shown in Figure 3.9. Examining Table 3.2 and Figure 3.9, it can be seen that the schedules of the tasks in each column correspond to a scaling region.

However, a column of the initial schedule table has not necessarily to directly correspond to a scaling region. This will be the case whenever, according to the generated schedule, a task is running in parallel with a disjunction task and is finishing after that one. Such a situation is illustrated next using an example.

Example 2: Problem of Identifying Scaling Regions

Previously, motivational example 1 has been used to show the challenge of the proposed conditional behaviour aware DVS technique. This will be another example showing the problem of identifying scaling regions. Consider the CTG of Figure 3.10, which is mapped to the architecture of Figure 3.2. Assuming the task mappings and execution times are as Table 3.3, Table 3.4 gives its schedule table. Accordingly, Figure 3.11 shows the schedules of the two tracks corresponding to condition value A and condition value $\overline{A}$, where track 2 is the worst case track since it ends later than track 1. Consider the schedule of track 1 (Figure 3.11(a)). Task n_2 is running over the finishing time of disjunction task n_3. To scale the schedule for energy saving and at the same meet the timing constraints in the worst case (track 2), the scaling region from time 0 to 4 should be scaled with a conservative scaling factor, which can be calculated according to Equation 3.10 as 15/12=1.25, as shown in Figure 3.12. However, when task n_3 has finished, the information concerning the track that will be followed, in this example, track 1 corresponding to condition value A, is available. Hence, in order to make the most of the available slack, a larger scaling factor of (15 5)/(10 4)=1.67 will be applied and, consequently, the PE will be run at lower voltage (Figure 3.12). As it can be seen, task n_2 is scaled with two different scaling factors, i.e., it belongs to two different scaling regions. In such cases, the task belonging to more than one scaling regions should be split and distributed over several columns of the schedule table, so that each column corresponds to a scaling region, in order to apply the proposed DVS technique to the schedule table. For this example, n_2 should be split and distributed over column *true*, column A, and column $\overline{A}$, as shown in Table 3.5. It can be seen that, after splitting, the schedules of the tasks in each column directly correspond to a scaling region. Table 3.6 gives the schedule table after voltage scaling for this example.



(a) conditional task graph (b) track 1 (c) track 2

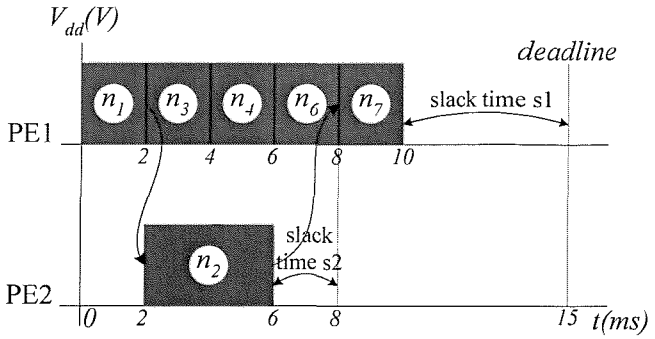
Figure 3.10: A conditional task graph and its tracks

Task	Mapping	Execution time (ms)
n_1	PE1	2
n_2	PE2	4
n_3	PE1	2
n_4	PE1	2
n_5	PE1	4
n_6	PE1	2
n_7	PE1	2

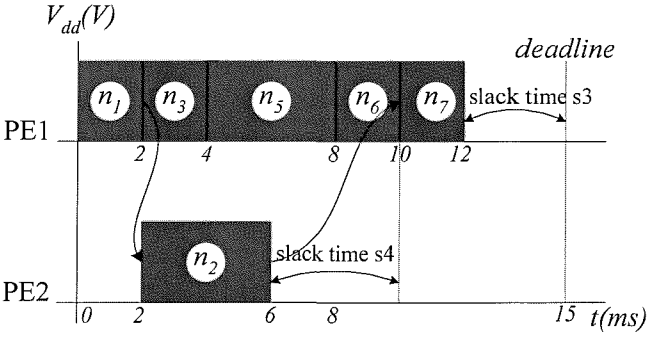
Table 3.3: Task mapping and execution times of the CTG of Figure 3.10

Task	*true*	A	$\bar{A}$
n_1	0→2		
n_2	2→6		
n_3	2→4		
n_4		4→6	
n_5			4→8
n_6		6→8	8→10
n_7		8→10	10→12

Table 3.4: Schedule table of the CTG of Figure 3.10



(a) Schedule of track 1



(b) Schedule of track 2

Figure 3.11: Schedule of the CTG of Figure 3.10

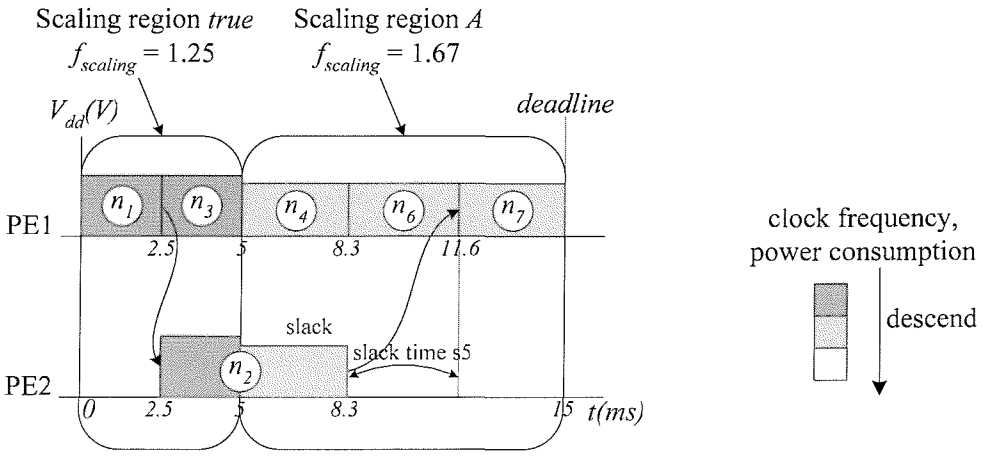


Figure 3.12: Scaling the schedule of Figure 3.11(a)

Task	*true*	A	$\overline{A}$
n_1	0→2		
n_2	2→4	4→6	4→6
n_3	2→4		
n_4		4→6	
n_5			4→8
n_6		6→8	8→10
n_7		8→10	10→12

Table 3.5: Pre-processed schedule table of Table 3.4

Task	*true*	A	$\overline{A}$
n1	0→2.5		
n2	2.5→5	5→8.3	5→7.5
n3	2.5→5		
n4		5→8.3	
n5			7.5→10
n6		8.3→11.6	10→12.5
n7		11.6→15	12.5→15

Table 3.6: Scaled schedule table of Table 3.5

After having identified the scaling regions delimited by the end times of disjunction tasks, each scaling region can be scaled with a scaling factor determined incrementally during run-time. The drawback of this scaling technique is that the tasks on the non-critical paths do not take advantage of the available slack time. For example, as shown in Figure 3.12, after scaling the tasks with corresponding scaling factors, a slack time s5 is still available. This has to be exploited for further energy saving. A technique [55] is adapted to exploit such slack times. The basic idea lies in energy gradient, which is the difference between the energy dissipation of a task with the execution time t and the reduced energy dissipation (due to DVS) of the same task, when extended with a time quantum dt. The technique (1) splits the slack time (difference between the latest finish time and the actual finish time of the tasks) into a number of time quantum dt; (2) calculate the energy gradients for all extensible tasks (tasks which have slack time), achievable by extending them with dt; (3) identifies the task with the biggest energy gradient, which will result in the highest energy reduction; (4) extends the selected task with dt using the voltage scaling capability of the PE to

which the task is mapped; (5) repeats steps (2) – (4) until there is no extensible task left.

DVS Technique for DVS
Input schedule table - *SchTable*
Deadline - T_d

```
01 pre-process SchTable
02 for (each column col in SchTable, from left to right)
03 {
04      identify the worst case track - track_worst
05      calculate the worst case total slack time - slack_worst
06      calculate the slack time distributed to col - slack_col
07      scale col with scaling factor f_scaling calculated using Equation 3.10
08      apply DVS technique in [55] to col
09      update SchTable
10 }
```

Figure 3.13: Conditional behaviour aware DVS technique

The proposed conditional behaviour aware DVS technique for CTG is described in Figure 3.13. Step 01 pre-processes the input schedule table, so that each column corresponds to a scaling region. Steps 02 – 10 apply DVS to all the columns in *SchTable*, in a left-to-right sequence. For each column *col*, step 04 firstly identifies all possible tracks that will be followed after the condition values heading *col* are known; then the track with the latest end time (the end time of the sink node in the track) is identified, which is referred as the worst case track $track_{worst}$. Step 05 calculates the worst case total slack time $slack_{worst}$ which is obtained by subtracting the end time of $track_{worst}$ from the deadline T_d. Step 06 calculates the slack time distributable to *col*, $slack_{col}$, by distributing $slack_{worst}$ to the columns along the $track_{worst}$ in proportion to the columns' duration (i.e. the difference between the latest end time and the earliest start time of the tasks in the column). Step 07 scales *col* with the $f_{scaling}$ given by:

$$f_{scaling} = \frac{duration_{col} + slack_{col}}{duration_{col}} \tag{3.10}$$

where $duration_{col}$ is the duration of *col*. Step 08 exploits the slack times on non-critical path using the DVS technique proposed by Schmitz and Al-Hashimi [55]. Due to the scaling of *col*, Step 09 should update the contents in the columns that are successive to *col* along all the possible tracks.

Example 3: Applying the Proposed DVS Technique

To illustrate the proposed conditional behaviour aware DVS technique, it is applied to the schedule table for the CTG of Figure 3.1(a), Table 3.2. In this case, because the columns have already been corresponding to scaling regions, as shown in Figure 3.9, step 01 can be simply skipped. Then begin to process column *true*. Step 04: taking into account that no condition value is yet known, there are 3 possible tracks: track1, track 2, and track 3 (see Figure 3.9). Track 3 is the worst case track, where the sink node n_7 ends at 26, compared to 20 in track 1 and 22 in track 2. Step 05: since the worst case track finishes at 26 and the deadline is 30, the worst case total slack time is 30ms – 26ms = 4 ms. Step 06: 1.5 ms out of the 4ms slack time is distributed to column *true*, which is given by 4ms · (10ms / 26ms) ≈ 1.5ms, where 10ms is the duration of column *true* and 26 is the time needed to finish the worst case track. Step 07: the task in column *true*, n_1, is scaled with the scaling factor 1.15, which is given by (10ms + 1.5ms) / 10ms = 1.15 using Equation 3.10. According to Equation 3.7 – 3.8, n_1 is then run from 0 to 11.5. Correspondingly, the voltage is scaled to 3.03V using Equation 3.3 – 3.5. Step 08: since there is no non-critical path in column *true*, this step can be skipped. Step 09: column *true* is a part of track 1, track 2, and track3. In track 1, column A is successive to column *true*; in track 2, columns $\overline{A}$ and $\overline{A}\wedge B$ are successive to column *true*; in track 3 columns $\overline{A}$ and $\overline{A}\wedge\overline{B}$ are successive to column *true*. Therefore the schedules of columns A, $\overline{A}$, $\overline{A}\wedge B$, and $\overline{A}\wedge\overline{B}$ are updated due to the scaling of column *true*. Table 3.7 is produced after the end of Step 09, and its corresponding schedule is shown in Figure 3.14.

	true	A	$\overline{A}$	$\overline{A}\wedge B$	$\overline{A}\wedge\overline{B}$
n_1	0→11.5				
n_2		11.5→16.5			
n_3			11.5→15.5		
n_4				15.5→17.5	
n_5					15.5→21.5
n_6				17.5→18.5	21.5→22.5
n_7		16.5→21.5		18.5→23.5	22.5→27.5

Table 3.7: Schedule table after processing column *true*

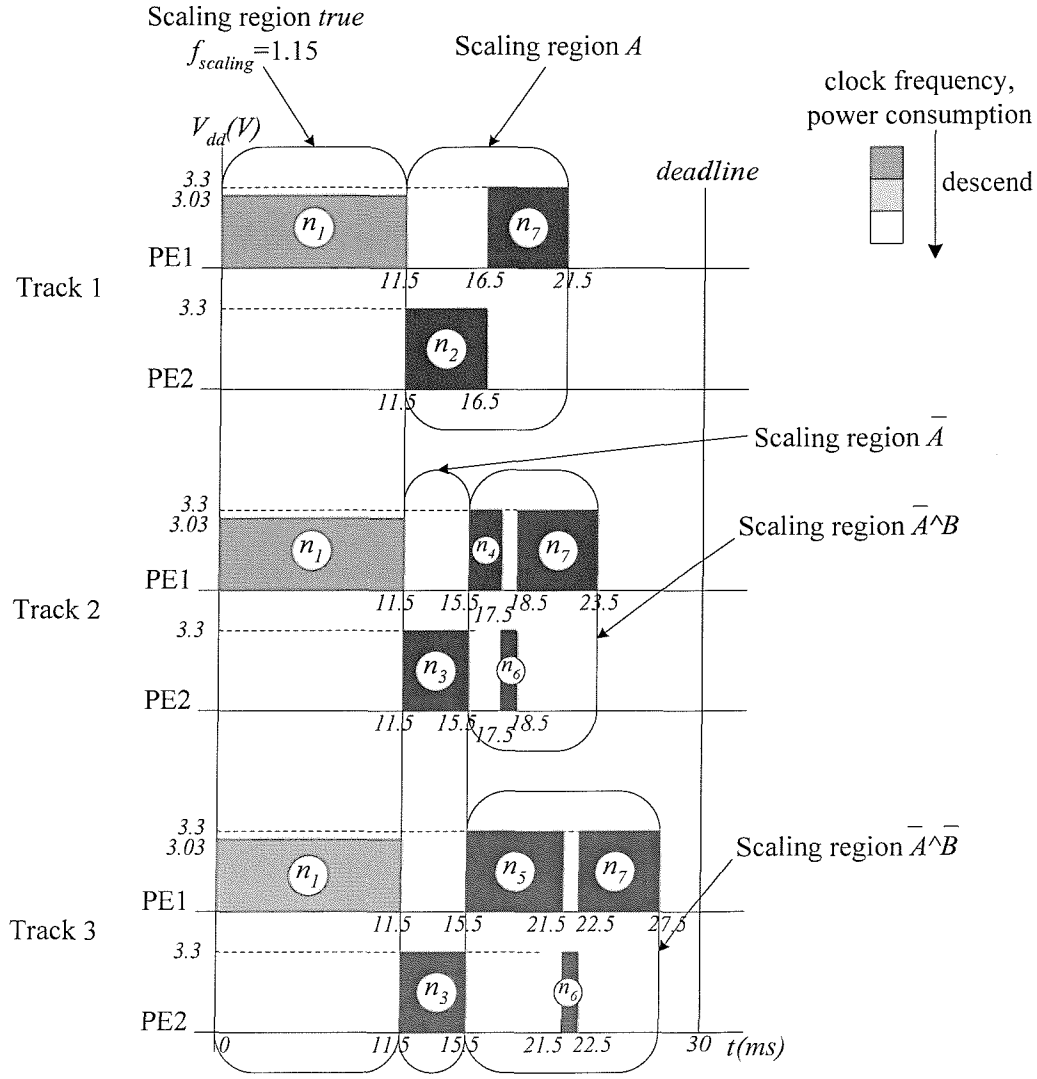


Figure 3.14: Schedule after processing column *true* (Table 3.7)

	true	A	$\overline{A}$	$\overline{A}\wedge B$	$\overline{A}\wedge\overline{B}$
n_1	0→11.5				
n_2		11.5→20.75			
n_3			11.5→15.5		
n_4				15.5→17.5	
n_5					15.5→21.5
n_6				17.5→18.5	21.5→22.5
n_7		20.75→30		18.5→23.5	22.5→27.5

Table 3.8: Schedule table after processing column A

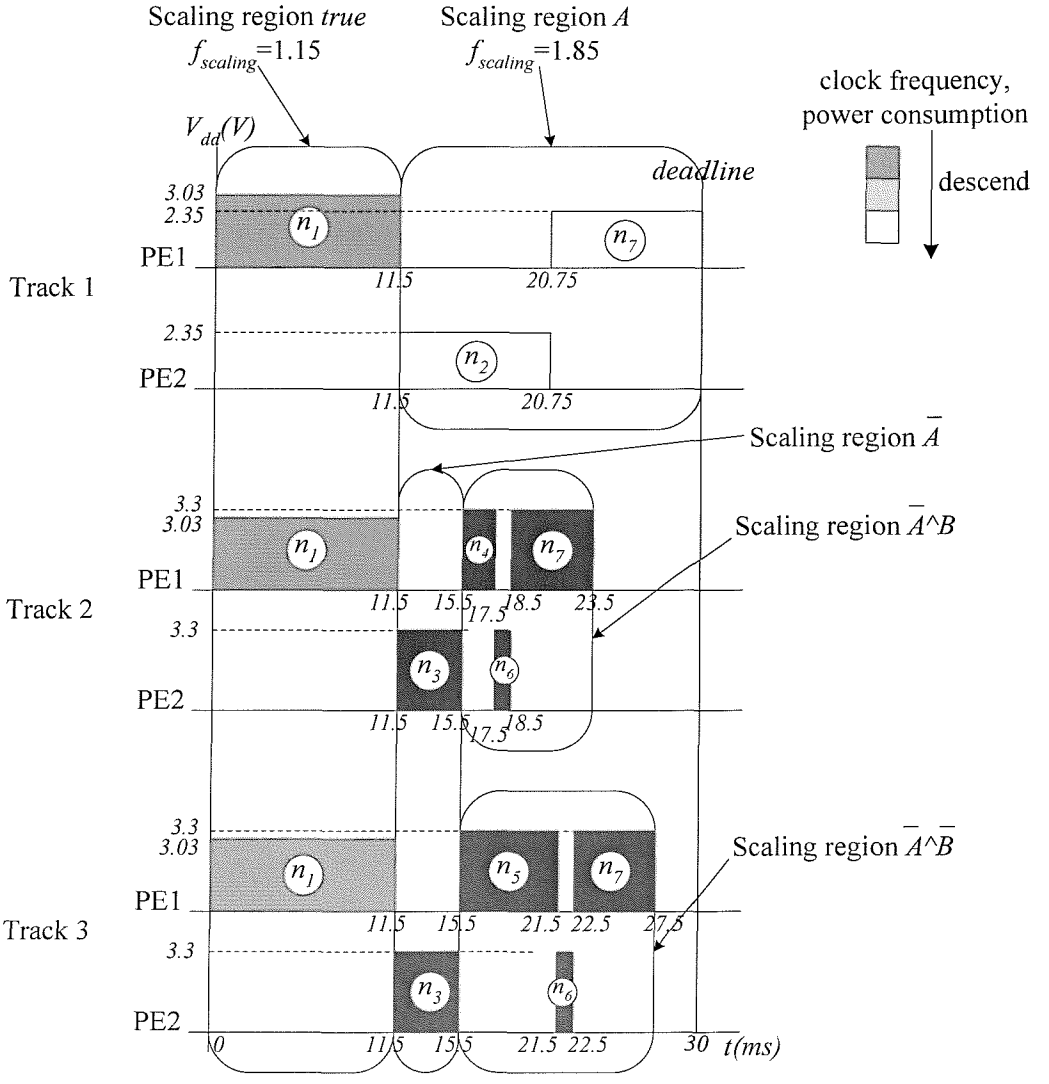


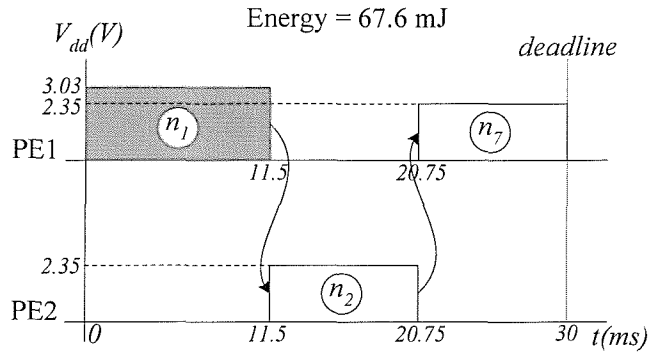
Figure 3.15: Schedule after processing column A (Table 3.8)

Starting with Table 3.7, repeat steps 04 – 09 for column A. Steps 04: having known condition value A, track 1 is the only possible track will be followed. Hence, track 1 is the worst case track (see Figure 3.14). Step 05: since the worst case track finishes at 21.5 and the deadline is 30, the worst case total slack time is 30ms – 21.5ms = 8.5ms. Step 06: since there is no successive columns to column A in track 1, the whole 8.5ms slack time is distributed to the column A. Step 07: the tasks in column A, n_2 and n_7, are scaled with the scaling factor , which is given by (10ms + 8.5ms) / 10ms = 1.85 using Equation 3.10, where 10ms is the duration of column A. After voltage scaling, n_2 is then run from 11.5 to 20.75, and n_7 from 20.75 to 30. Correspondingly, the voltage is scaled to 2.35V using Equation 3.3 – 3.5. Step 08: this step is skipped since there is no non-critical path in column A. Step 09: this step is also skipped since there is no successive column to column A in track 1. Table 3.8 is produced after the end of Step 09, and its corresponding schedule is shown in Figure 3.15.

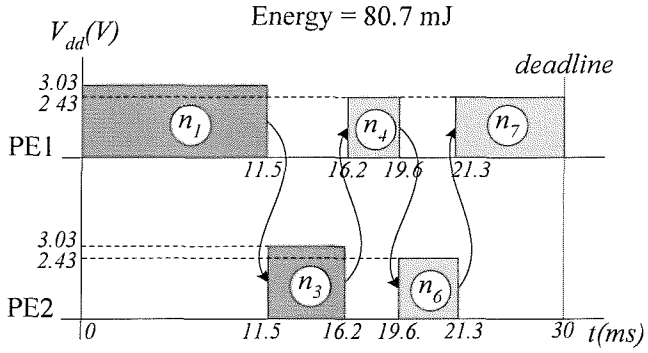
Similarly, considering columns $\overline{A}$, $\overline{A}\wedge B$, and $\overline{A}\wedge\overline{B}$ separately, applying steps 04 – 09 to them, the final schedule table is obtained as shown in Table 3.9. According to Table 3.9, Figure 3.16 show the actual schedules of the three tracks after voltage scaling, which meet the deadline and at the same time produce minimal energy dissipation.

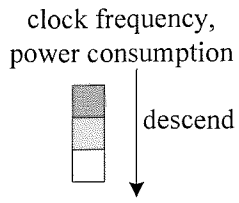
	true	A	$\overline{A}$	$\overline{A}\wedge B$	$\overline{A}\wedge\overline{B}$
n_1	0→11.5				
n_2		11.5→20.75			
n_3			11.5→16.2		
n_4				16.2→19.6	
n_5					16.2→23.1
n_6				19.6→21.3	23.1→24.2
n_7		20.75→30		21.3→30	24.2→30

Table 3.9: Final schedule table after voltage scaling

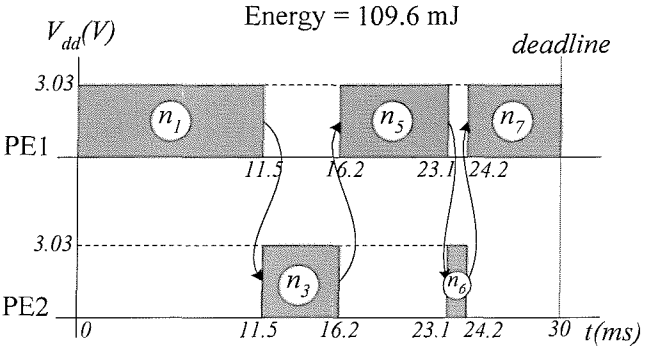


(a) Scaled schedule of track 1





(b) Scaled schedule of track 2



(c) Scaled schedule of track 3

Figure 3.16: Final schedule after voltage scaling (Table 3.9)

Summary of Example 3

By comparing Figure 3.7 and Figure 3.16, it can be observed that the actual schedule is the same as the schedule of Figure 3.7 only in the case of track 3, which is

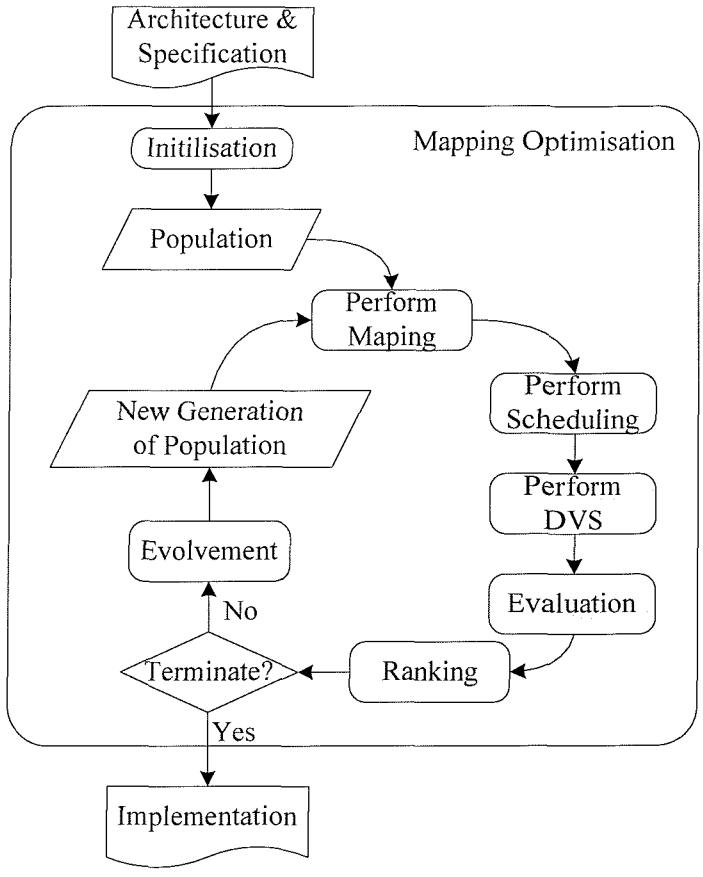
the worst case track. It is important to note that the schedules of the other tracks in Figure 3.7 are impracticable. This is because the schedules in Figure 3.7 are produced upon the assumption that the condition values are known before executing the disjunction nodes, which is not true during the runtime of the application. In practice, the condition values are not known until all the disjunction nodes have finished their execution. Hence, it is not possible for an online voltage scheduler to immediately use this information to achieve feasible and energy-efficient settings.

3.4 Energy-Efficient Mapping for CTGs

In Section 3.3 a conditional behaviour aware DVS technique has been employed with mapped and scheduled CTGs to reduce energy dissipation. In this section, a mapping technique specifically developed for better utilisation of the proposed DVS is described. Combining the proposed mapping technique with the conditional behaviour aware DVS technique can reduce system energy dissipation further, as shown in the experimental results (Section 3.5).

The flow of the mapping technique is shown in Figure 3.17 (a). The input to the mapping technique is the system architecture and specification, the output is an optimised system implementation, including task mapping, task scheduling, and voltage scaling. The mapping optimisation is based on a genetic algorithm (GA, Section 3.1.3). Genetic algorithm maintains a population (a pool of mapping candidates). Each mapping candidate is represented by a mapping string. Figure 3.17 (b) shows a possible mapping string for the CTG of Figure 3.1(a), which means n_1 is mapped to PE1, n_2 is mapped to PE2, and so on. Each mapping candidate is associated with an objective function value (i.e., fitness), which measures the quality of the candidate. In the proposed mapping technique, the objective function is given by:

$$Fitness = -\left(\sum_i E(\tau_i)\right) \cdot \left(\frac{\max(T_d, T_e)}{T_d}\right)^2 \qquad (3.11)$$



(a) Block flow of energy-efficient mapping

Task:	n_1	n_2	n_3	n_4	n_5	n_6	n_7
Mapping string:	1	2	2	1	1	2	1

(b) Mapping string

Figure 3.17: Energy-efficient mapping technique

where i is the task number, $E(\tau_i)$ is the energy dissipation of task τ_i, T_d is the deadline of the CTG, T_e is the real execution time of the CTG. The first part of the fitness function is the total energy dissipation of all tasks, which has to be minimised. The second part of the function introduces a penalty factor due to deadline violations. If the length of the schedule is smaller than the deadline, the value of the second part is 1, hence, no penalty is applied. Otherwise, the squaring introduces a higher penalty to

the fitness. Thus, the optimisation process is driven towards implementations with reduced energy dissipation, and at the same time, deadlines are satisfied.

The process of the mapping technique (Figure 3.17(a)) is described as the following. Firstly, an initial population of mapping candidates is created randomly (Initialisation). For each candidate in the population:

(1) a mapping is generated according to the mapping string (Perform Mapping);
(2) a schedule table is produced for the mapped CTG using the scheduling algorithm [42] (Perform Scheduling);
(3) the schedule table is passed to the conditional behaviour aware DVS technique (Section 3.3) to perform voltage scaling (Perform DVS);
(4) according to the results of voltage scaling, the fitness for the mapping candidate is calculated using Equation 3.11 (Evaluation).

After steps (1) – (4) are applied to all the mapping candidates, the candidates are ranked according to their fitness. Then the algorithm decides whether the optimisation should be terminated: if no improved candidate has been produced for a certain number of iterations, the optimisation is terminated and the best implementation is returned; otherwise, the optimisation continues with generation evolvement (Evolvement). After evolvement, the new generation of mapping candidates is feed back to the loop. This iterative process continues until the termination condition is met. The aim of this iterative optimisation process is to finally produce an implementation that has low energy dissipation, and at the same time meets the deadline.

The evolvement involves the selection of high ranked candidates and applying genetic operators to them (Section 3.1.3). Crossover (mating) selects a pair of high-ranked mapping strings as parent. Offspring are produced by replacing part of the first parent string with part of the second parent string. Hence, crossover results in two new offspring strings. By selecting high quality mapping strings for crossover, the chances to evolve mapping strings of higher quality are increased. In order to enter an unexplored region of the search space, the genetic algorithm also mutates the mapping strings occasionally with a low probability. The mutation is carried out by randomly

changing a digit of a randomly selected mapping candidate. Note that the flow of Figure 3.17(a), as well as Chapter 4, Figure 4.9, is similar to the one presented by Schmitz and Al-Hashimi [57], because they all use general genetic algorithm flow to perform mapping optimisation. However, the technique presented by Schmitz and Al-Hashimi [57] targets pure data flow systems, while the technique in this research is specifically developed for data and control dominated systems. Conditional behaviour aware DVS technique is employed in the mapping technique of this research to evaluate the quality of mapping candidates.

3.5 Experimental Results

To demonstrate the efficiency and the applicability of the proposed conditional behaviour aware DVS (Section 3.3) and mapping (Section 3.4) technique in reducing the energy dissipation of multi-processor embedded systems expressed as conditional task graphs, experiments and comparisons with previously published approach [119] have been carried out. The DVS and mapping techniques outlined in this chapter have been implemented using C++ on a Pentium-III/866MHz Linux PC (Appendix A). The used examples consist of two sets: (1) a real-life example of vehicle cruise controller from [135]; (2) a number of synthetic examples automatically generated using the tool provided by [42].

3.5.1 Vehicle Cruise Controller Example

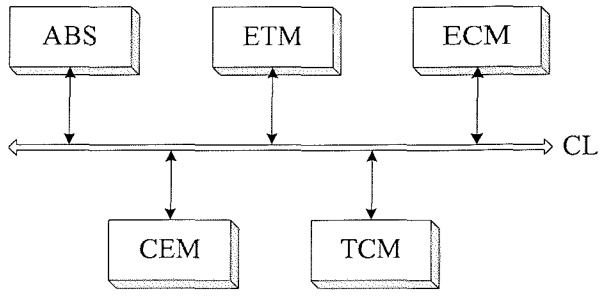


Figure 3.18: Architecture of the vehicle control cruiser (VCC)

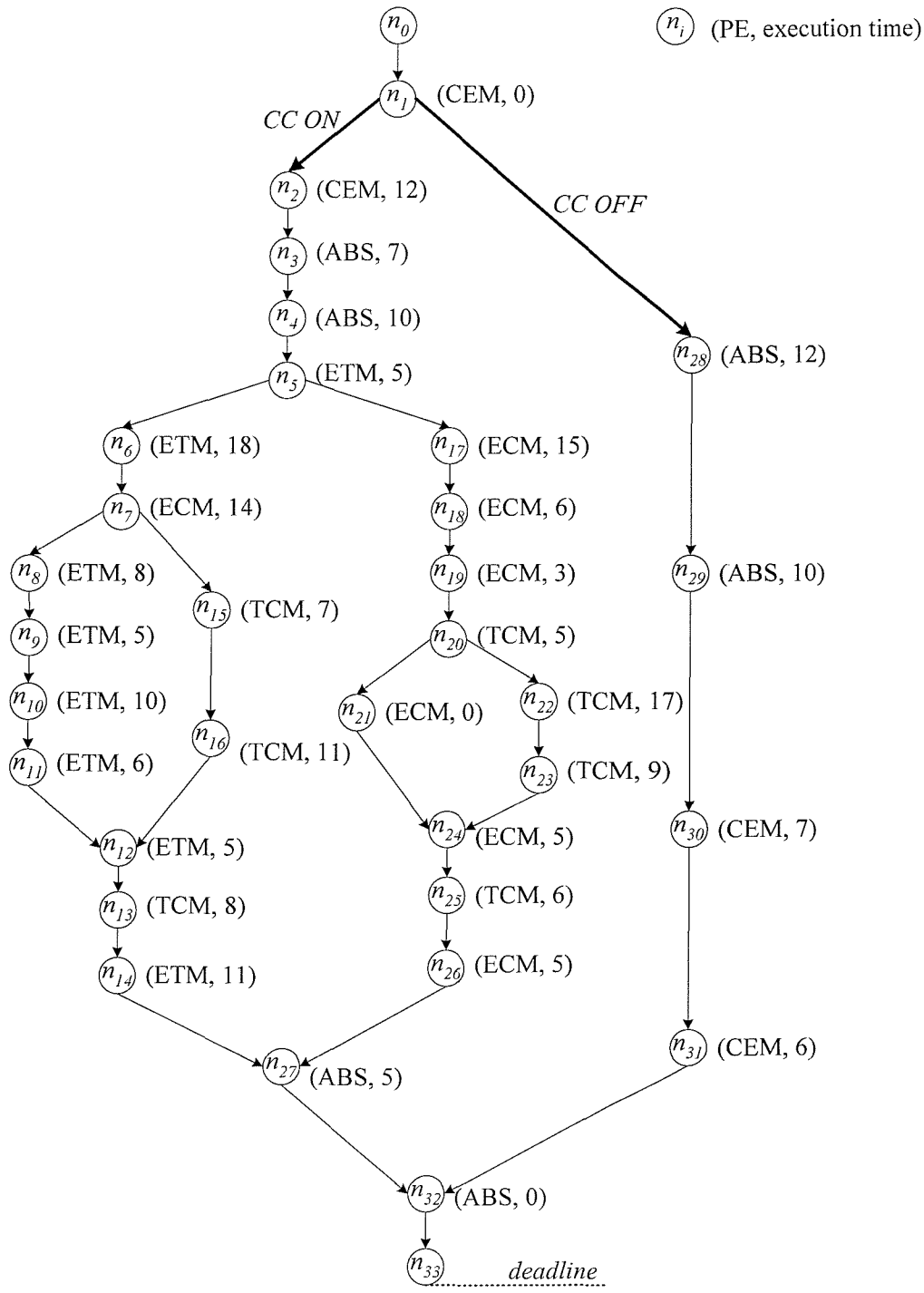


Figure 3.19: The vehicle cruise controller (VCC) functionality expressed as CTG

A real-life example of vehicle cruise controller (VCC) from [135] is used to test the efficiency of the conditional behaviour aware DVS technique. The example is derived from a requirement specification provided by the industry. The functionality of VCC

is described as follows: it maintains a constant speed for speeds over 30m/h and under 80m/h, offers an interface (buttons) for the driver to increase or decrease the reference speed, and is able to resume its operation at the previous reference speed. The VCC suspends its operation when the driver presses the brake pedal or accelerator pedal. The specification assumes that the functionality of the VCC is distributed over an architecture consisting of five interconnected processing elements (PEs): anti blocking system (ABS), transmission control module (TCM), engine control module (ECM), electronic throttle module (ETM), and central electronic module (CEM), as shown in Figure 3.18. The functionality of the VCC has been derived using a CTG that consists of 32 tasks and includes two alternative tracks (CC ON, CC OFF), as shown in Figure 3.19. This figure also shows the task mapping and the corresponding execution time to the right of each task. It is assumed that each PE has DVS capability (V_{nom}=3.3V, V_{th}=0.8V), such that the supply voltage can be scaled continuously within the range of 1.0V – 3.3V, the power consumption of the PEs at the nominal supply voltage (3.3V) is 5W.

To test the effectiveness of the proposed DVS technique for CTG, a scheduling algorithm [42] is used to generate a schedule table for the VCC example, and then apply the proposed DVS technique to the schedule table. Four cases with variant deadlines are examined, i.e., 100%, 105%, 110% and 120% of the schedule length produced by [42]. Table 3.10 gives the experimental results, with one row for each deadline. It can be seen that the proposed DVS technique reduces the energy dissipation significantly for all cases. For example, consider the case of 120% deadline (4^{th} row). The energy dissipation before applying DVS is 440mJ, and it was reduced to 288.87mJ after applying DVS, i.e., a reduction of 34.35%. Also, the energy reduction becomes more significant as the deadline increases, because more slack time is available to be exploited by DVS. For example, the energy reduction is 19.28% in the case of 100% deadline. It is further increased to 34.35% in the case of 120% deadline. For this example, the task mapping (i.e., which PE gets which task) has not been optimised. This is because the mapping is explicitly decided by the actual VCC implementation and changing its mapping will lose its sense of real-life.

	Energy Dissipation before DVS (mJ)	Energy Dissipation after DVS (mJ)	Energy Reduction (%)
100% deadline	440.00	355.15	19.28
105% deadline	440.00	335.61	23.73
110% deadline	440.00	318.28	27.66
120% deadline	440.00	288.87	34.35

Table 3.10: Results for vehicle cruise controller example

3.5.2 Synthetic Examples

Twenty six synthetic mapped CTG examples (ctg1 – ctg26) are generated using the tool provided by [42], with various complexities in terms of the number of node (ranges from 13 to 93), edges (ranges from 16 to 118), conditions (ranges from 2 to 5). These examples use architectures consisting of 2 to 5 DVS-enabled PEs. The DVS-enabled PEs have threshold voltage (V_{th}) of 0.8V and nominal supply voltage (V_{nom}) of 3.3V. Their supply voltage can be scaled between 1V to 3.3V continuously, and their power consumption at nominal supply voltage is 5W. Three experiments have been carried out. Experiment 1 concentrates on the proposed conditional behaviour aware DVS technique (Section 3.3). Experiment 2 deals with the proposed mapping technique (Section 3.4). Experiment 2 also presents a comparison between the proposed mapping technique and a previously proposed technique [119]. Experiment 3 examines the effects of PE number and condition number on the energy dissipation.

Experiment 1

In this experiment, a scheduling algorithm [42] is employed to generate a schedule table for each example. Then the proposed DVS technique is applied to the schedule table. The results of experiment 1 are given in Table 3.11. Column 1 lists the examples used in this experiment; column 2 shows the complexity of the examples in terms of the number of node (task), edge, condition and PE; column 3 gives the schedule length produced by [42]; column 4 shows the deadline, which is 110% of the schedule length; column 5 and column 6 are the energy dissipation before and after applying the DVS technique; finally, column 7 shows the energy reduction achieved by applying DVS, which is calculated as (1 – (column 6 / column 5)). It can be seen

from the table that, for all the examples, the proposed DVS technique significantly reduces the energy dissipation. For example, in the case of ctg1 (1st row), the energy dissipation before DVS is 563.75mJ, and it is reduced to 366.599mJ after DVS, i.e., a reduction of 34.97%; similarly, in the case of ctg26 (26th row), the energy dissipation before DVS is 2690.62mJ, and it is reduced to 1811.29 after DVS, a reduction of 32.68%.

Example	No. of node/edge/ condition/ PE	Schedule length (ms)	Deadline (ms)	Energy dissipation (mJ)		Energy Reduction (%)
				Before DVS	After DVS	
ctg1	13/16/2/2	109	119.9	563.75	366.599	34.97
ctg2	13/16/2/3	73	80.3	547.5	395.35	28.16
ctg3	13/16/2/4	112	123.2	630	421.656	33.07
ctg4	13/16/2/5	97	106.7	573.75	372.45	35.08
ctg5	13/16/3/2	110	121	632.5	511.205	19.18
ctg6	13/16/3/4	112	123.2	717.5	563.569	21.45
ctg7	25/30/2/2	267	293.7	1492.5	1174.4	21.31
ctg8	25/30/2/3	243	267.3	1495	1197.07	19.93
ctg9	25/30/2/4	172	189.2	1090	761.009	30.18
ctg10	25/30/2/5	211	232.1	1302.5	940.649	27.78
ctg11	25/30/3/2	204	224.4	1240	973.057	21.53
ctg12	25/30/3/3	251	276.1	1576.25	1212.75	23.06
ctg13	25/30/3/4	246	270.6	1393.75	1000.47	28.22
ctg14	25/30/3/5	196	215.6	1271.25	1023.15	19.52
ctg15	25/29/4/2	221	243.1	1187.5	881.673	25.75
ctg16	25/29/4/3	183	201.3	1070	789.477	26.22
ctg17	25/29/4/4	222	244.2	1172.5	864.295	26.29
ctg18	25/29/4/5	214	235.4	1060	777.599	26.64
ctg19	35/41/2/2	342	376.2	1371.25	946.793	30.95
ctg20	37/45/2/3	437	480.7	1803.75	1378.73	23.56
ctg21	35/41/2/5	353	388.3	1528.75	1069.83	30.02
ctg22	38/48/2/2	306	336.6	2105	1598.76	24.05
ctg23	48/60/3/3	497	546.7	1921.88	1313.04	31.68
ctg24	46/55/3/5	450	495	1711.88	1163.96	32.00
ctg25	59/71/3/3	613	674.3	3685	2676.84	27.36
ctg26	93/118/5/5	566	622.6	2690.62	1811.29	32.68

Table 3.11: Results of applying DVS to the synthetic examples

Experiment 2

In this experiment, a comparison between two co-synthesis approaches is presented to demonstrate the effectiveness of the proposed mapping technique in terms of energy reduction. Approach 1 consists of the mapping and scheduling algorithm proposed by Xie and Wolf [119], combined with the proposed DVS technique; approach 2 consists of the proposed mapping technique, a scheduling algorithm [42], and the proposed DVS technique. Firstly consider approach 1. The mapping and scheduling algorithm [119] is employed to generate mapping and scheduling for each CTG, then the proposed DVS technique is applied to the mapped and scheduled CTG. The results of approach 1 are given in columns 2 – 6 of Table 3.12. Column 2 lists the deadline, which is 110% of the schedule length produced by [119]; columns 3 and 4 give the energy dissipation before and after applying the DVS technique; column 5 is the energy reduction achieved by DVS, which is calculated as (1 – (column 4 / column 3)); column 6 gives the computational time used to produce the solution. It can be seen that applying the proposed conditional aware DVS reduces the energy dissipation, as expected.

Now consider approach 2. The proposed mapping technique is used to optimise the mapping for each CTG, the schedule technique [42] is employed to generate a schedule table, and the proposed DVS is applied to the schedule table. In order to make a fair comparison with approach 1, the same deadline as in the case of approach 1 is imposed to the CTGs. The results of approach 2 are given in columns 7 – 9. Column 7 gives the energy dissipation of the examples after applying DVS; column 8 is the energy reduction compared to the energy dissipation before DVS in approach 1, calculated as (1 – (column 7 / column 3)); column 9 gives the computational time. Comparing the energy reduction achieved by approach 1 (column 5) and approach 2 (column 8), it can seen that, using the mapping technique specifically developed for better utilisation of the conditional behaviour aware DVS, the energy dissipation is reduced further. For example, consider ctg11 (11^{th} row), the energy reduction achieved by the proposed mapping technique is 31.13%, that is about two times higher than 15.48%, the reduction achieved by using the mapping and schedule technique of

Example	Approach 1: [119]+ proposed DVS					Approach 2: [42] + proposed mapping & DVS		
	Deadline (ms)	Energy dissipation (mJ)		Ene Redu. (%)	CPU time (s)	Energy dissipation after DVS (mJ)	Ene Redu. (%)	CPU time (s)
		Before DVS	After DVS					
ctg1	100.1	510	388.29	23.86	0.030	312.89	38.65	1.29
ctg2	73.7	493.75	382.39	22.55	0.030	282.76	42.73	4.78
ctg3	69.3	562.5	434.04	22.84	0.020	389.32	30.79	4.15
ctg4	82.5	448.75	300.48	33.04	0.040	254.64	43.26	3.14
ctg5	107.8	602.5	493.66	18.06	0.030	404.34	32.88	1.48
ctg6	99	607.5	460.85	24.14	0.050	405.07	33.32	2.90
ctg7	233.2	1330	1142.88	14.07	0.050	967.08	27.29	10.66
ctg8	202.4	1322.5	1088.21	17.72	0.040	932.54	29.49	65.59
ctg9	138.6	970	793.70	18.17	0.090	668.04	31.23	23.68
ctg10	163.9	1105	890.38	19.42	0.070	764.69	30.80	45.63
ctg11	203.5	1157.5	978.28	15.48	0.100	797.14	31.13	25.69
ctg12	181.5	1253.75	1037.29	17.27	0.070	916.32	26.91	57.77
ctg13	198	1141.25	865.13	24.19	0.100	748.03	34.46	33.06
ctg14	166.1	1050	841.29	19.88	0.130	773.01	26.38	56.17
ctg15	183.7	1010	879.49	12.92	0.030	781.45	22.63	6.710
ctg16	130.9	905	773.19	14.57	0.040	687.07	24.08	10.18
ctg17	183.7	948.75	769.84	18.86	0.050	669.76	29.41	10.97
ctg18	140.8	796.25	666.65	16.28	0.040	599.76	24.68	18.44
ctg19	337.7	1256.25	991.20	21.10	0.070	873.23	30.48	3.120
ctg20	396	1618.75	1299.49	19.72	0.080	1160.75	28.29	6.150
ctg21	270.6	1163.75	903.95	22.32	0.070	835.90	28.17	8.57
ctg22	251.9	1982.5	1622.66	16.13	0.120	1509.71	23.85	131.46
ctg23	418	1631.25	1268.92	22.21	0.160	1108.82	32.03	41.530
ctg24	371.8	1316.88	954.67	27.51	0.040	873.68	33.66	12.78
ctg25	504.9	3218.75	2638.23	18.04	0.190	2310.56	28.22	643.46
ctg26	479.6	2156.88	1654.76	23.28	0.450	1513.06	29.85	271.82

Table 3.12: Results of Mapping optimisation of the synthetic examples

[119]. The efficiency of the proposed mapping technique is also shown by comparing the results of approach 2 (columns 7 – 9 of Table 3.12) and the results of step 1 (Table 3.11), where the same scheduling and DVS technique are employed, while approach 2 performs an additional mapping optimisation. Consider ctg1, in Table 3.11, its energy dissipation after DVS is 366.599 (column 6), considering a deadline of 119.9 (column 4); while in Table 3.12, its energy dissipation after DVS is 312.89 (column 7), considering a deadline of 100.1 (column2). By performing mapping optimisation,

further energy reduction is achieved with a tighter deadline. Due to the iterative optimisation nature, the higher energy reduction achieved by the proposed mapping technique is at the cost of increased computational time, as shown in the column 9.

Experiment 3

This experiment investigates the effect of the PE number and condition number on the energy dissipation. The deadline of ctg24 (46 nodes, 55 edges, see Table 3.11 and Table 3.12) is set as 395ms, various PE numbers and condition numbers are assigned to ctg24. The PE number ranges from 3 to 5, whilst the condition number ranges from 1 to 3. The proposed mapping and DVS techniques are applied to ctg24 with various PE numbers and condition numbers. Table 3.13 gives the energy dissipation of ctg24 before and after applying DVS. For example, considering the situation that ctg24 has 1 condition (row 1), when the architecture has 3 PEs, the energy dissipations before and after DVS are 1960mJ and 1547.98mJ respectively; when the architecture has 4 PEs, its energy dissipations before and after DVS are 1875mJ and 1351.61mJ respectively. To show the trend of energy dissipation corresponding to the PE number and condition number, the results in Table 3.13 are represented in Figure 3.20 and Figure 3.21. It can be seen from the figures that, in both cases (before and after DVS), the energy dissipation increase as the PE number reduces. Similarly, the reduction of condition number also increases the energy dissipation. For example, considering Figure 3.21 (after DVS), the energy dissipation is 854.86mJ when the (PE, condition) numbers are (5, 3), the energy dissipation increases to 968.14mJ when the (PE, condition) numbers are (4, 3), the energy

	3 PE		4 PEs		5 PEs	
	Energy (mJ)		Energy (mJ)		Energy (mJ)	
	Before DVS	After DVS	Before DVS	After DVS	Before DVS	After DVS
1 condition	1960	1547.98	1875	1351.61	1810	1235
2 conditions	1558.75	1125.32	1495	1017.3	1442.5	938.31
3 conditions	1488.12	1058.62	1428.75	968.14	1346.25	854.86

Table 3.13: Effects of PE number and condition number

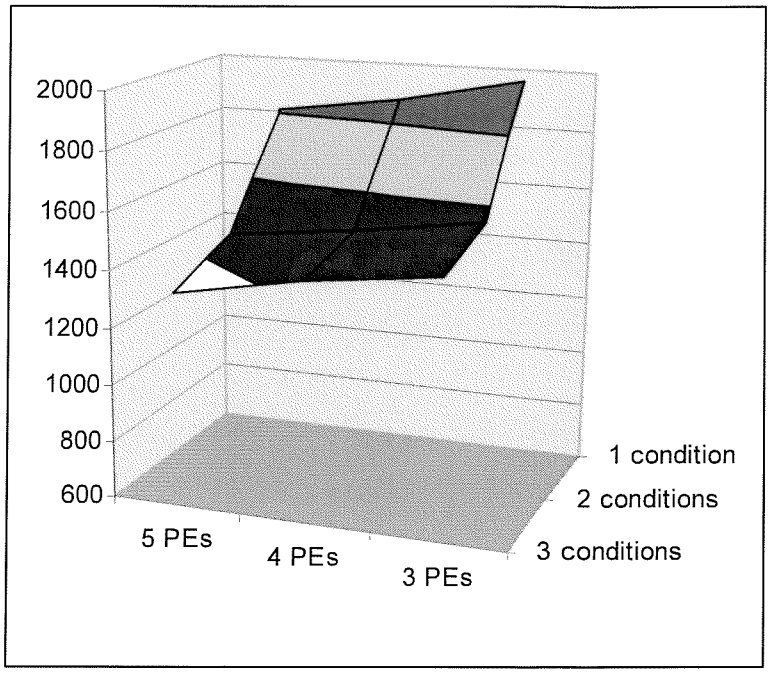


Figure 3.20: Energy dissipation before DVS

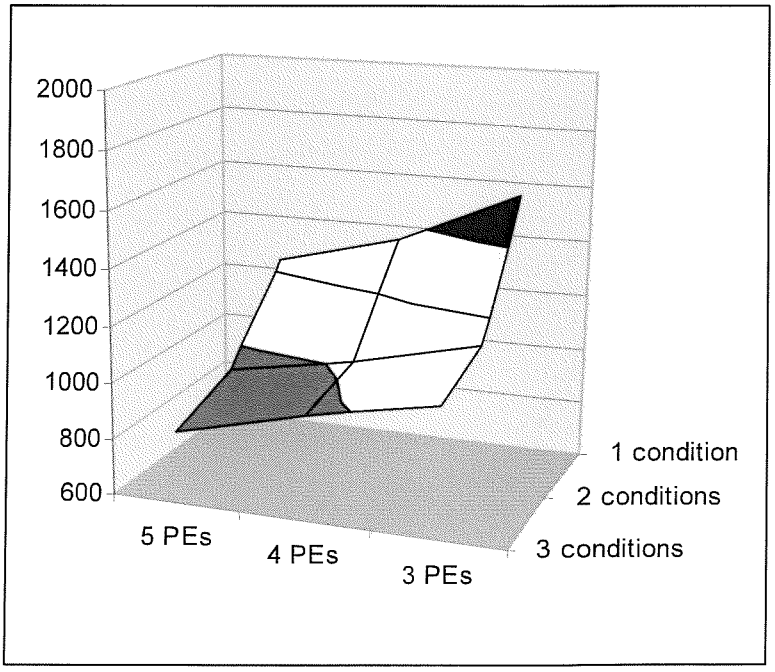


Figure 3.21: Energy dissipation after DVS

dissipation increases further to 1017.3mJ when the (PE, condition) numbers are (4, 2). This is because, as the PE number decreases, the load on each PE increases. The

increased load on each PE results in less slack time available for DVS, therefore higher energy dissipation. On the other hand, as the condition number decreases, a larger part of ctg24 needs to be executed in a certain activation of the application, which leads to higher energy dissipation.

3.6 Concluding Remarks

This chapter presented new DVS-driven energy minimisation techniques for data and control dominated embedded systems expressed as conditional task graphs (CTGs), including a conditional behaviour aware DVS technique and a genetic algorithm (GA) based mapping technique. The proposed DVS technique exploits the slack time taking into account the conditional behaviours of CTGs, such that energy dissipation is reduced and, at the same time, the deadlines are satisfied under any possible conditions. The proposed mapping technique optimises the task mapping towards better utilisation of the proposed DVS technique, thus to achieve further energy reduction. Combining the proposed mapping and DVS techniques with the scheduling algorithm of [42] forms a co-synthesis approach for CTGs, which is able to produce high quality designs in terms of energy efficiency and timing feasibility. The efficiency of the proposed techniques has been validated using a large number of examples, including a real-life vehicle cruise controller (VCC) application. Experimental results show that the proposed techniques achieve significantly better energy reduction, compared to a previously reported approach which neglects the availability of DVS, and this energy reduction can be achieved within a reasonable amount of computational time.

Chapter 4

Communication-Integrated Energy-Efficient Co-Synthesis

A conditional behaviour aware co-synthesis technique has been presented in Chapter 3 to improve the energy efficiency for data and control dominated systems expressed as conditional task graphs (CTGs). In order to concentrate on the analysis of conditional behaviour, a simplified model (referred to as the *basic system model*) is considered where impact of communications is ignored. In this chapter, a more accurate system model (referred to as the *enhanced system model*) is introduced, which takes into account the time and energy cost of communications. A co-synthesis technique integrated with communication considerations is presented to perform mapping, scheduling and voltage scaling for the *enhanced system model*. Using the proposed co-synthesis technique, a performance analysis is carried out to investigate the effect of alternative communication architecture on system quality in terms of energy efficiency.

The remaining of this chapter is organised as follows. Section 4.1 outlines the main motivation of integrating communications with DVS-driven co-synthesis technique for CTGs. The concept of enhanced system model is introduced in Section 4.2. Section 4.3 describes the proposed communication-integrated energy-efficient co-synthesis technique. Experimental results are presented in Section 4.4. Finally concluding remarks are given in Section 4.5.

4.1 Motivation

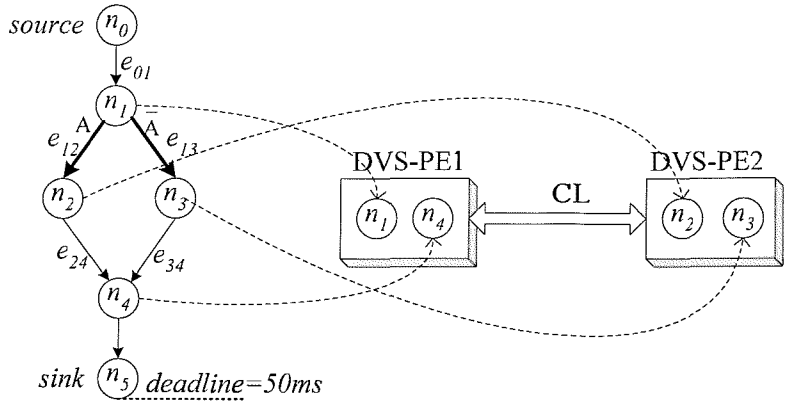
Several early works [52-54, 98-100] in embedded system co-synthesis have considered single processor systems without communication links (CLs). However, numerous embedded systems are implemented as multi-processor systems [55, 58, 59,

101, 102], and the system representations often contain data and control flows [41, 42, 116-118]. For such systems, communications have an important impact on embedded systems design. Communications have drawn much attention from the research community [42, 120-125, 136]. Knudsan and Madsen [122] presented a communication model to estimate the performance of CLs with various parameters, including bus width and clock frequency. The model is used to integrate communication protocol selection with hardware-software partitioning. Eles *et al* [42, 123] investigated the impact of communication protocols on the task scheduling, employing the time-triggered protocol (TTP) [137]. The techniques [42, 122, 123] all try to maximise the system performance without considering the energy dissipation. There has also been some research examining the influence of communications on energy dissipation of embedded systems. Bus encoding techniques were proposed to reduce the communication energy dissipation [124, 125]. Liu *et al* [136] presented a communication speed selection method to exploit the energy/performance trade-offs between communications and computations on DVS-enabled processors, in order to globally optimise the energy dissipation of embedded systems. Joint dynamic voltage scaling for variable-voltage processing elements and communication links are proposed in recent papers [138, 139], to maximise system energy saving, further demonstrating the importance of integrating communication in energy-efficient system design.

Based on the work in Chapter 3, this chapter presents a co-synthesis technique, which integrates the communication considerations into the co-synthesis steps, performs efficient mapping, scheduling and voltage scaling of tasks and communications, aiming to achieve energy-efficient system implementations while meeting the real-time constraints. In particular, the computational time of the communication-integrated co-synthesis technique is significantly reduced by decoupling communication from task mapping, instead of mapping tasks and communications concurrently. Furthermore, using the communication-integrated co-synthesis technique, this chapter investigates the effect of alternative communication architectures on system energy efficiency, providing a guide to communication architecture selection. To illustrate the impact of communication and communication

architecture selection on the system energy efficiency, a motivational example is given next.

Motivational Example



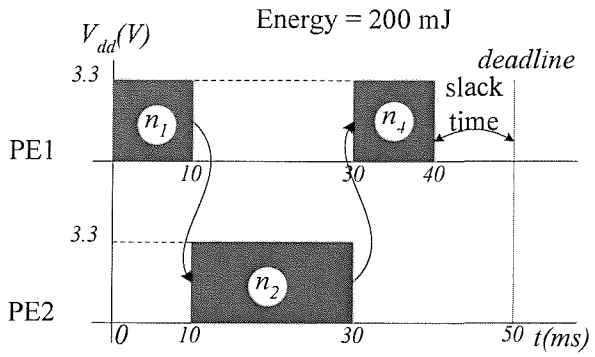
(a) Conditional task graph (b) Architecture

Figure 4.1: Motivational example

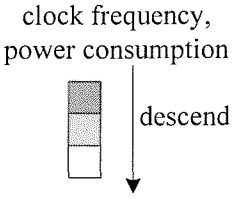
Task	Mapping	Execution time (ms)
n_1	PE1	10
n_2	PE2	20
n_3	PE2	20
n_4	PE1	10

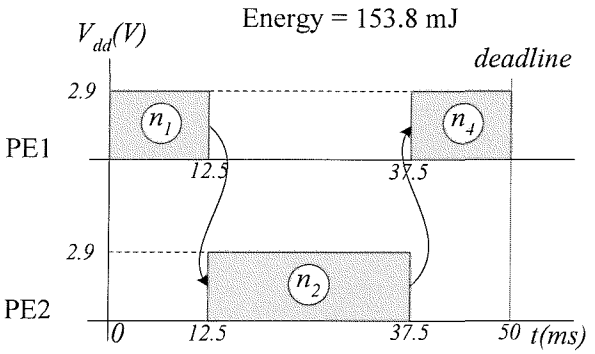
Table 4.1: Task mapping and task execution time

Consider the conditional task graph (CTG) of Figure 4.1(a), assuming this CTG has been mapped to the architecture of Figure 4.1(b), the mapping and the corresponding execution times of the tasks are shown in Table 4.1. The dummy tasks n_0 and n_5 are not considered, because they are assumed to have zero execution time and not mapped to any PE (Appendix C). Assume the power consumption, nominal supply voltage, and threshold voltage of PE1 and PE2 are 5W, 3.3V, and 0.8V respectively. Using the scheduling and voltage scaling techniques outlined in Chapter 3 (i.e. without consideration of communications), Figure 4.2(a)-(b) show the schedules and energy dissipations when the condition value is A, before and after applying DVS. Now, assuming communications have time and energy costs, consider two types of



(a) Schedule before voltage scaling





(b) Schedule after voltage scaling

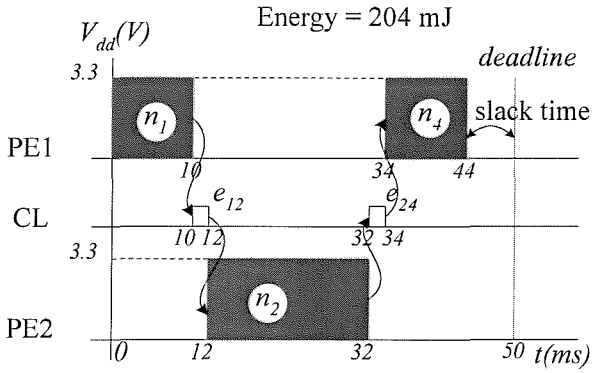
Figure 4.2: Schedule and voltage scaling without communication

CLs: CL1 and CL2. CL2 is faster and more power-consuming compared to CL1. The costs of e_{12} (communication between n_1 and n_2) and e_{24} (communication between n_2 and n_4) are given in Table 4.2. For example, e_{12} takes 2ms and consumes 1W power using CL1, while it takes 0.5ms and consumes 1.5W power using CL2. Assuming PE1 and PE2 are connected with CL1, Figure 4.3 shows the schedules and energy dissipations before and after voltage scaling. Similarly, assuming PE1 and PE2 are connected with CL2, the energy dissipation in this case before and after voltage scaling is 201.5mJ and 158.9mJ respectively. Figure 4.4 summaries the results of this motivational example. It can be seen that taking communications into consideration increases the energy dissipation, as expected. Furthermore, the selection of CL considerably influences the system energy efficiency. Employing CL2 leads to lower energy dissipation than employing CL1. This is because CL2 consumes less energy

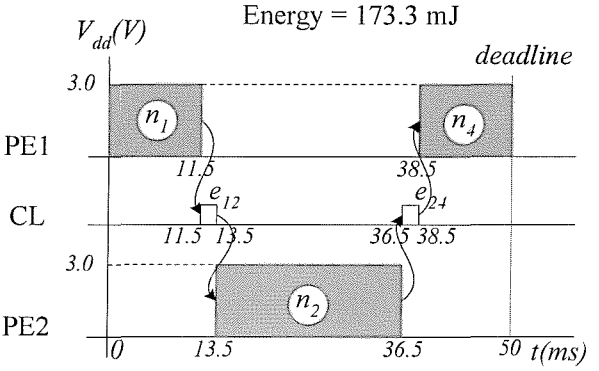
(time × power) for transmitting e_{12} and e_{24} (Table 4.2) than CL1, furthermore, CL2 consumes less time, thus provides more slack time for voltage scaling of the computation tasks.

	CL1		CL2	
	Time (ms)	Power (W)	Time (ms)	Power (W)
e_{12}	2	1	0.5	1.5
e_{24}	2	1	0.5	1.5

Table 4.2: Communication costs



(a) Schedule before voltage scaling



(b) Schedule after voltage scaling

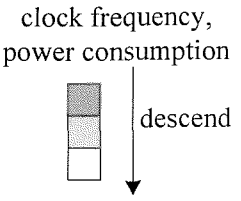


Figure 4.3: Schedule and voltage scaling with communication (using CL1)

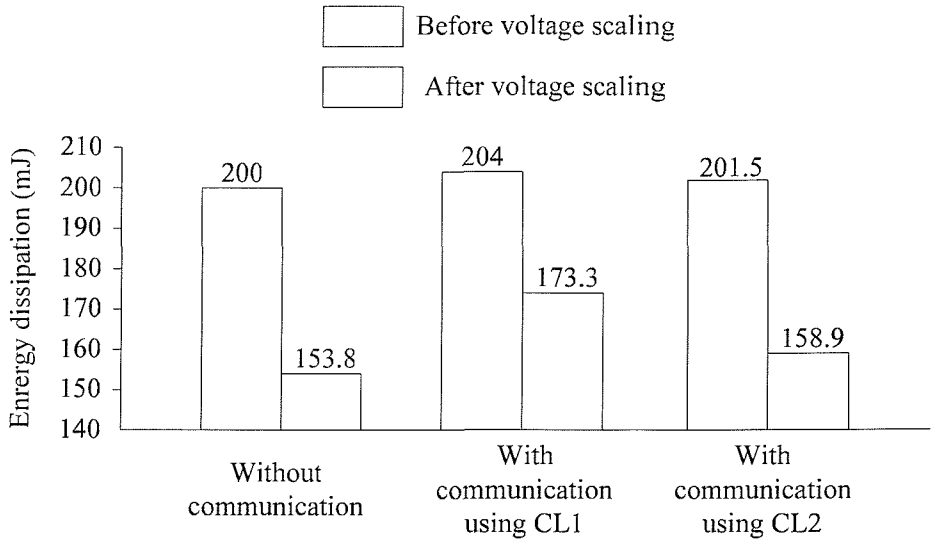


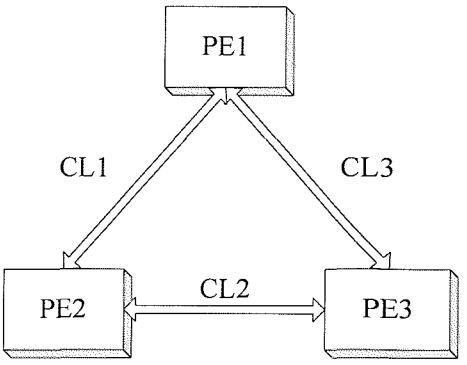
Figure 4.4: Summary of motivational example

4.2 Enhanced System Model

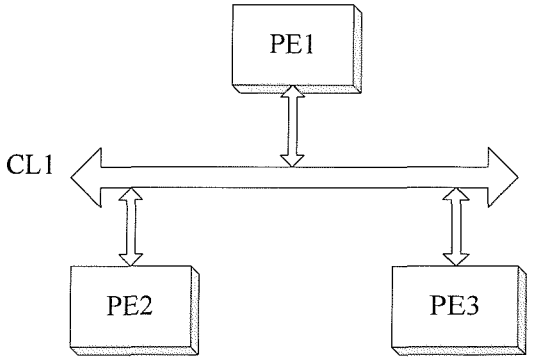
This section describes the enhanced system model considered in this chapter. The communication architecture is introduced. The power and delay models of communications are presented. Finally, the key features of the enhanced system model are outlined.

4.2.1 Communication Architecture

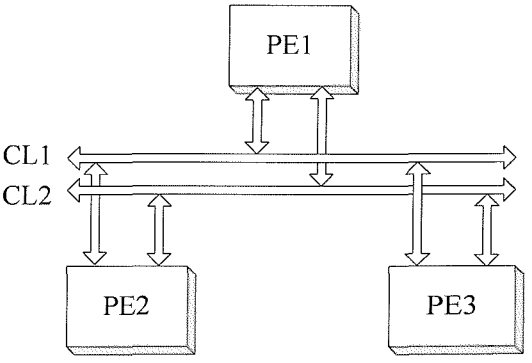
The communication architecture could be as simple as a single shared system bus, or more complex, consisting of a network of shared bus and dedicated communication channels. Shared buses are very commonly used to facilitate communications between PEs. Being shared buses, they require arbitration in order to ensure that only one component has the control of the bus at any given time. Dedicated links are point-to-point connections between PEs. Since such links are not shared, the question of arbitration does not arise and communication on such links can proceed as soon as both the sender and receiver are ready to communicate. Figure 4.5 shows three possible communication architectures for a multi-processor embedded system. Figure 4.5(a) shows a communication architecture where the three PEs are connected via



(a) Dedicated links architecture



(b) Shared bus architecture



(c) Split shared bus architecture

Figure 4.5: Examples of communication architectures

three dedicated links. In Figure 4.5(b), the three PEs are connected by a shared bus. In this shared bus communication architecture, an arbiter resolves conflicts resulting from simultaneous attempts to access the bus. An alternative architecture is shown in Figure 4.5(c), where the shared bus in Figure 4.5(b) is split into two shared buses. The

increase in bus number decreases the chance of bus conflicts. However, each shared bus in Figure 4.5(c) has a reduced bandwidth compared to the one in Figure 4.5(b), which leads to a possible degradation in performance. Hence, there is a trade-off between the higher bandwidth of a shared bus and the lower frequency of bus conflicts in a split bus architecture.

A communication between two tasks, which are mapped to two different PEs, can be carried out on any CL that connects the two PEs. For example, consider the communication architecture of Figure 4.5(c), if there is a communication γ between task τ_1 and τ_2, which are mapped to PE1 and PE2 respectively, γ can be carried on either CL1 or CL2. It is assumed that communications between tasks that mapped to a same PE do not cause time or energy costs, while communications over CLs cost certain amount of time and energy, which can be estimated using power and delay models as discussed next.

4.2.2 Power and Delay Models of Communications

In order to integrate the communications within the co-synthesis techniques, this section drives the power and delay models of communications. The models include equations for the calculation of energy dissipation and execution time needed for a communication over a communication links (CL). The execution time of a communication γ over a CL is dependent on the data size of γ and the performance of the CL [140]:

$$d_\gamma = \frac{B_\gamma}{baud_{CL}} \tag{4.1}$$

where d_γ is the execution time of γ, B_γ is the data size of γ, and $baud_{CL}$ is the baud-rate of the CL. The baud-rate of a CL is given by [140]:

$$baud_{CL} = W_{CL} \cdot f_{CL} \tag{4.2}$$

where W_{CL} and f_{CL} are the bus width and the clock frequency of CL respectively.

With each transmission of data over the CL, the line capacitance is charged and discharged, drawing a current from the I/O pins of the processing elements (PEs). The power consumed by these current is given by [2]:

$$P_{CL}(\gamma) = C_{eff} \cdot f_{CL} \cdot V_{CL}^2 \tag{4.3}$$

where $P_{CL}(\gamma)$ is the power consumption of the CL, C_{eff} is the effective load capacitance of the CL averaged over the whole duration of γ, f_{CL} is the clock frequency of the CL, and V_{CL} is the operational voltage of the CL. The energy dissipation needed for the CL to transmit γ can be derived from Equations 4.1 – 4.3:

$$E_{CL}(r) = P_{CL}(\gamma) \cdot d_{\gamma} = C_{eff} \cdot V_{CL}^2 \cdot \frac{B_{\gamma}}{W_{CL}} \tag{4.4}$$

4.2.3 Heterogeneous Processing Elements

In the enhanced system model, system architecture contains a number of heterogeneous PEs. Each PE is either a software programmable PE (e.g., CPU) or a dedicated hardware PE (e.g., ASIC). There are two aspects needed to be considered in the co-synthesis techniques: (1) software PEs can execute only one task at one time, while hardware PEs allow multiple tasks to be executed in parallel; (2) the area (i.e., functional units) of hardware PEs is a limited resource, which imposes a constraint on the number of tasks that can be mapped to hardware PEs, while software PEs don't have such a constraint because their functional units can be shared by all the tasks mapped to them. The heterogeneous nature of the PEs is also exhibited in their various properties (Table 4.3), including nominal supply voltage, threshold voltage, nominal clock frequency, area limit, whether DVS-enabled or not. Furthermore, it is assumed that the power consumption of the PEs is dependent on the executed tasks, due to the employment of low-level power minimisation techniques such as clock gating in modern IC designs [55, 130]. Each task might be potentially mapped to several PEs, resulting in various task properties for each alternative mapping (Table 4.3), including execution times, power dissipation, area overhead (only applicable for hardware

implementations). Table 4.3 summaries the PE properties and task properties considered in the enhance system model.

PE properties	Task properties
V_{nom}, nominal supply voltage V_{th}, threshold voltage f_{nom}, nominal clock frequency DVS-enabled A, area (only applicable for hardware PE)	N, execution cycle $P(V_{nom})$, power dissipation at V_{nom} a, area overhead (only applicable for hardware implementation)

Table 4.3: PE properties and task properties

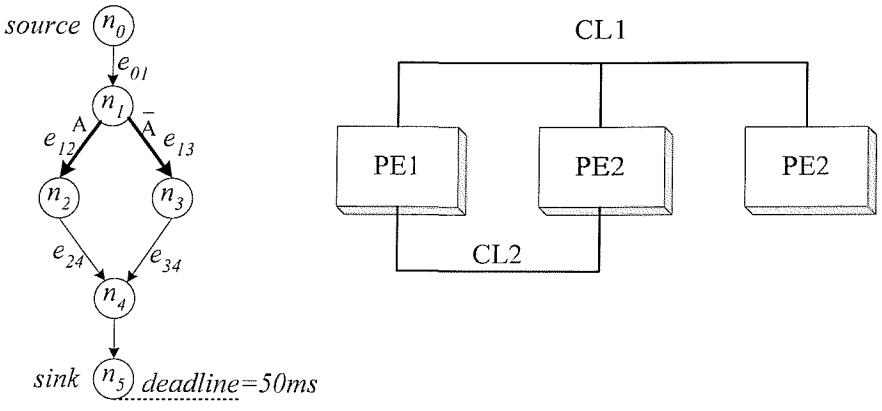
4.3 Integrating Communication with DVS based Co-Synthesis

The co-synthesis techniques of Chapter 3 are extended in this section, in order to address the impact of communications and heterogeneous PEs introduced in the enhanced system model (Section 4.2). The following sub-sections present the modification made to the three co-synthesis steps: mapping, scheduling, and voltage scaling. Section 4.3.1 presents a decoupled task mapping and communication mapping, which has the advantage of reduced computational time comparing with a concurrent task and communication mapping. A scheduling is introduced in Section 4.3.2, which determines the mapping of communications onto communication links (CLs) and the execution order of tasks and communications simultaneously. Section 4.3.3 describes an enhanced DVS, which allows efficient exploitation of slack time taking account of the enhanced system model.

4.3.1 Task Mapping

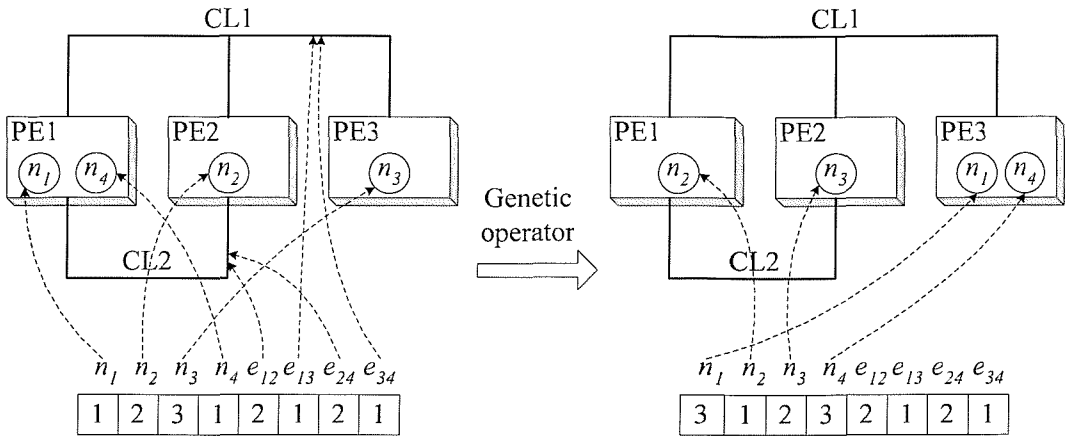
In Chapter 3, a genetic algorithm based mapping is proposed to optimise the task mapping, where a mapping string is used to represent a candidate mapping, each digit in the string represents a task's mapping (Chapter 3, Figure 3.17(b)). However, the mapping technique in Chapter 3 ignores the communication mapping (i.e., assign a CL for each communication between two tasks mapped to different PEs), which is investigated next.

At first sight, communication mapping can be handled together with task mapping using a mapping string, each digit of which represents the mapping of a task or a communication. However, using this way causes two problems. The first problem is illustrated in the following example. Consider the conditional task graph (CTG) of Figure 4.6(a), the tasks and communications need to be mapped to the architecture of Figure 4.6(b). In order to handle the task and communication mapping together, a possible mapping string is shown in Figure 4.7(a). The string combines the task mapping and communication mapping together. This string represents a valid solution, since communication e_{12} between task n_1 and n_2 is mapped to CL2, which connects the PEs that execute task n_1 and n_2. Similarly, communications e_{13}, e_{24} and e_{34} are mapped to CL1, CL2 and CL1 respectively. Now consider a certain genetic operation, e.g., crossover (Chapter 3, Section 3.2.3), transforms the mapping string into the one in Figure 4.7 (b). It is easy to see that the mapping string of Figure 4.7(b) represents an invalid solution. For example, consider the communication e_{12} between tasks n_1 and n_2. Although tasks n_1 and n_2 are respectively mapped to PE3 and PE1, which are solely connected by CL1, the communication e_{12} is mapped to CL2. From this example, it can be seen that combining task mapping and communication mapping



(a) Conditional task graph (b) Architecture

Figure 4.6: A CTG and its target architecture



(a) Valid mapping string (b) Invalid mapping string

Figure 4.7: Genetic operator leading to invalid mapping string

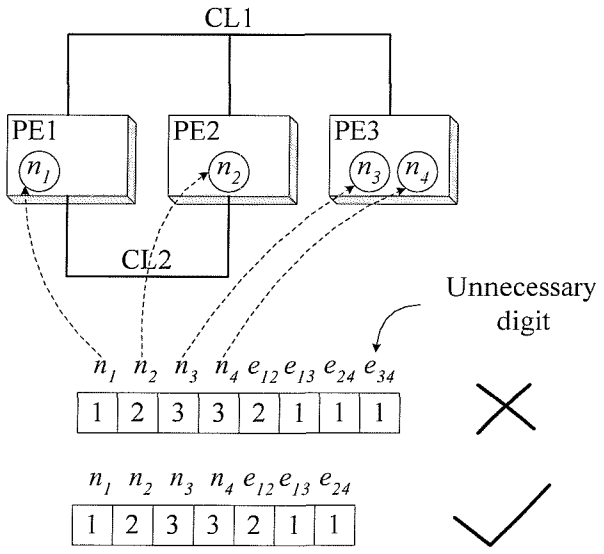


Figure 4.8: Unneeded communication mapping

may produce a large number of invalid solutions during the genetic algorithm based optimisation, which is undesirable for efficient design space exploration. The second problem caused by concurrent task and communication mapping is shown in Figure 4.8, where tasks n_3 and n_4 are mapped to a same PE. Hence there is no need to assign the communication e_{34} to a CL. In this case, ideally, a short mapping string should be employed to speed the mapping optimisation. However, during the combined task and

communication mapping, the length of the mapping string cannot be adapted dynamically. Therefore, the full length of the mapping string is always employed, causing unnecessary computational time.

Based on the above observation, it is clear that handling communication mapping together with task mapping could cause negative effects on the mapping optimisation. Thus the co-synthesis flow of Figure 4.9 is employed in this chapter. As it can be seen, the task mapping and communication mapping are handled separately. After performing task mapping, communication mapping is carried out simultaneously with activity scheduling. In this way, it is possible to avoid invalid solutions, since all possible mappings of communications onto the CLs are statically known for a given task mapping. For example, consider Figure 4.6 again, if the tasks n_1 and n_2 are mapped to PE1 and PE2 respectively, then the communication e_{12} can be mapped onto CL1 or CL2; on the other hand, if the tasks n_1 and n_2 are mapped to PE1 and PE3 respectively, then the communication e_{12} can only be mapped onto CL1. Hence the combined communication mapping and activity scheduling (Section 4.3.2) can takes advantage of this information to ensure that only valid solutions are produced. The flow of Figure 4.9 is similar to the one in Chapter 3, Figure 3.17, except for three enhancements to address the impact of communications and heterogeneous PEs:

(1) after having decided upon the task mapping (Perform Task Mapping), communication mapping is performed simultaneously with activity scheduling (Perform Communication Mapping & Activity Scheduling, Section 4.3.2);
(2) the DVS technique is enhanced to handle communication and heterogeneous PEs (Section 4.3.3);
(3) the fitness function of task mapping is modified to address the area constraints of ASICs (Equation 4.5).

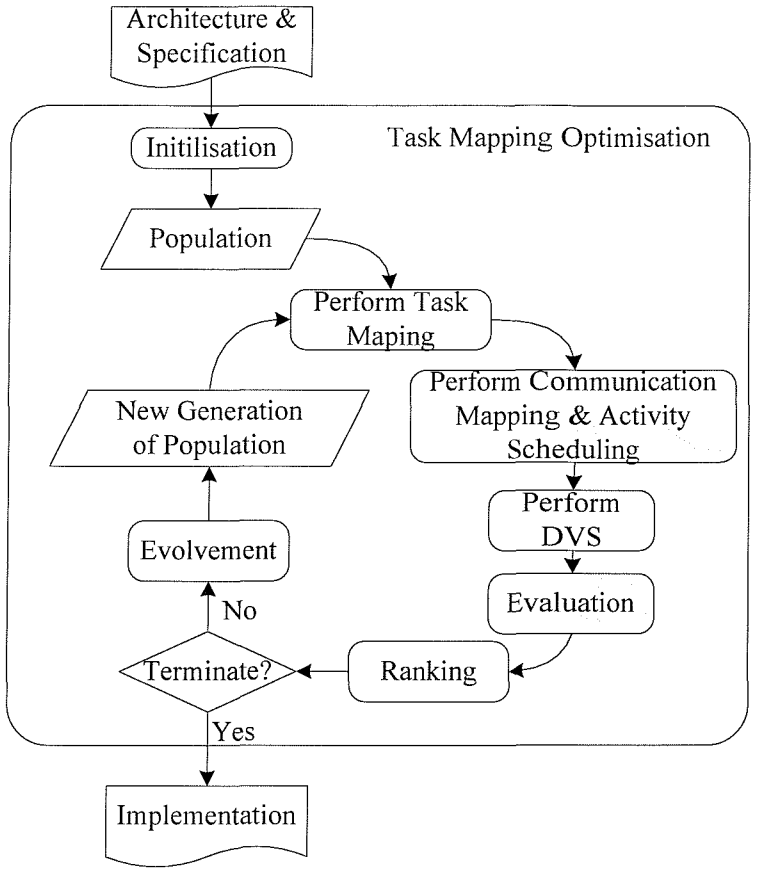


Figure 4.9: Separate task and communication mapping

The fitness function for task mapping is given by:

$$Fitness_{comm\_mapping} = -\left(\sum_i E(\tau_i) + \sum_j E(\gamma_j)\right) \cdot \left(\frac{\max(T_d, T_e)}{T_d}\right)^2 \cdot \prod_k AP_k \qquad (4.5)$$

where i, j and k are respectively the task number, the communication number and the PE number. The first part of the Equation 4.5 is the total energy dissipation of all tasks and communication, which has to be minimised. $E(\tau_i)$ is the energy dissipation of task τ_i, $E(\gamma_j)$ is the energy dissipation of communication γ_j. The second part of Equation 4.5 introduces a penalty factor due to deadline violations. T_d is the deadline of the CTG, T_e is the real execution time of the CTG. If the length of the schedule is smaller than the deadline, the value of the second part is 1, hence, no penalty is applied. In the opposite case, the squaring introduces a higher penalty to the fitness. The third part of

Equation 4.5 introduces a penalty factor due to area constraints violations. AP_k is an area penalty for each ASIC that exceeds its area constrains. AP_k is given by:

$$AP_k = \max\left(1, 1 + K \cdot \left(\frac{UA_k}{AA_k} - 1\right)\right) \tag{4.6}$$

where UA_k is the used area, AA_k is the available area, and K is a constant to control the magnitude of the area penalty. If UA_k is smaller than AA_k, the value of AP_k is 1, hence, no penalty is applied. In the opposite case, a penalty is introduced. The value of K needs to be sufficiently high to avoid infeasible results at the end of the mapping optimisation, and still low enough to allow infeasible solutions to survive sometimes, in order to increase the population diversity and avoid a premature convergence of the genetic algorithm (GA) towards low quality solutions. Extensive experiments are performed, where a number of values (0.01, 0.02, 0.03, 0.04, 0.05 and 0.1) are tried for K. For each value, a set of examples with various complexities are employed to examining the impact of K. For all the examples, a value of 0.02 was found to be a good choice, which guides the genetic algorithm to feasible results quickly. In this way, it is possible to achieve a solution such that the energy dissipation is minimised, and at the time, the timing and area constraints are respected.

4.3.2 Combined Communication Mapping and Activity Scheduling

This section presents a scheduling technique, which performs communication mapping and activity (task and communication) scheduling simultaneously. The proposed scheduling technique determines the mapping of communications onto communication links (CLs), and produces a schedule table for a conditional task graphs (CTG), aiming to meet the real-time constraints and provide as much as possible slack time for DVS (Section 4.3.3). The proposed scheduling technique is similar to [42], which also produces a schedule table for a CTG, but without the capability of performing communication mapping.

The basic idea is to convert communications into pseudo tasks, and consider CLs as pseudo PEs, thus allowing the unification of tasks and communication. The

mapping of a pseudo task (communication) is determined simultaneously when it is scheduled. In order to perform the combined communication mapping and activity scheduling, after having decided upon the task mapping (Figure 4.9), the CTG should be modified to capture the communication activities. This is achieved by converting communications between different PEs into pseudo tasks, as illustrated in the following example. Consider the CTG of Figure 4.6(a) and its mapping of Figure 4.7(a) (n_1 and n_4 are mapped to PE1, n_2 is mapped to PE2, n_3 is mapped to PE3). Since communications e_{12}, e_{13}, e_{24}, e_{34} are between different PEs, they should be converted to pseudo tasks, as shown in Figure 4.10. Also, all possible mappings of communications onto the CLs are statically decided, this information will be used in the combined communication mapping and activity scheduling technique (Figure 4.11). For example, the possible mappings for communications e_{12}, e_{13}, e_{24}, e_{34} are {CL1, CL2}, {CL1}, {CL1, CL2}, {CL1} respectively.

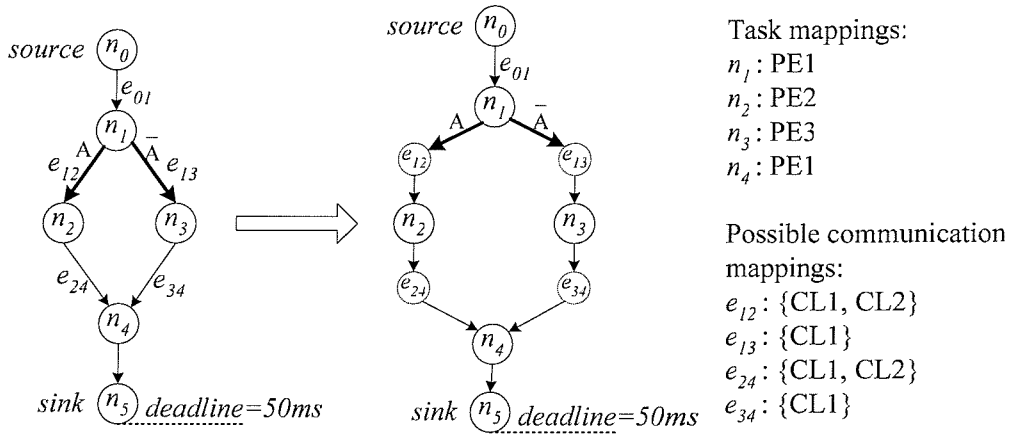


Figure 4.10: Modifying CTG to capture communications

After capturing the communications information using pseudo tasks, the combined communication mapping and activity scheduling is performed using the technique outlined in Figure 4.11. The technique is a list scheduling [42, 141, 142], which maintains an ordered list from which tasks are extracted to be scheduled at certain instances. In the proposed technique, two ordered lists are maintained: *ls_ready_task* and *ls_ready_comm*, in which tasks and communications are respectively placed in increasing order of the time when they become ready for

execution. This time is stored as an attributed t_{ready} for each task n and each communication r, and is the instance when all the predecessors of n or r have terminated. After initialising *ls_ready_task* (step 01), the technique enters two loops, which process communications (step 02 – 11) and tasks (step 12 – 23) respectively. The loops continue until all the communications and tasks have been scheduled. The key feature of the technique is the communication loop (step 02 – 11), which are described next. If there are ready communications in the *ls_ready_comm* (step 02), the communication r at the head of the *ls_ready_comm* is extracted (step 03). For each possible CL, the end time of r is calculates as (steps 04 – 05):

$$r.t_{end} = \max(r.t_{ready}, CL.t_{avaialble}) + r.t_{execution}(CL) \tag{4.7}$$

where $r.t_{ready}$ is the ready time of r, $CL.t_{available}$ is the time when CL becomes available, $r.t_{execution}(CL)$ is the execution time needed by CL to transmit r. Communication r is mapped to the CL which produces the earliest end time (step 06), and r is scheduled accordingly (step 07). Due to the schedule of r, the available time of CL, the *ls_ready_comm* and the *ls_ready_task* should be updated as steps 08 – 10. In step 10, those successor tasks of r, whose inputs have all arrived, are added to the *ls_ready_task*. In the task loop (steps 12 – 23), if task n is mapped to a hardware PE (step 15), n is scheduled, without any restriction, at the instance when n is ready (step 16). This is because hardware PE allows tasks run in parallel. If n is mapped to a software PE (step 17), however, it can be scheduled only after the PE becomes available ($PE.t_{available}$). There can be several tasks mapped to PE, so that at a given instance, $n.t_{ready} \leq PE.t_{available}$. All of these tasks will be ready when the PE becomes available. From these tasks, the one which has the highest priority is selected by function *select_task* (step 18), and is scheduled (step 19). In function *select_task*, the partial critical path (PCP) priority policy is employed due to its proved efficiency in scheduling CTGs [42]. In step 22, the *ls_ready_task* and *ls_ready_comm* are updated, depending on whether the task n that has just been scheduled is a disjunction task (Chapter 1, Section 1.3.2). For how to update the *ls_ready_task* and *ls_ready_comm* for CTGs, readers are referred to [42].

List_schedule(*ls_ready_task, ls_ready_comm*)

```
01 ls_ready_task = {source task}
02 while ls_ready_comm is not empty {
03        r = head (ls_ready_comm)
04        for each possible CL
05                calculate r.t_end
06        r.mapping = the CL with earliest r.t_end
07        r.t_start = max (r.t_ready, CL.t_available)
08        CL.t_available = r.t_start + r.t_execution(CL)
09        remove r from ls_ready_comm
10        update ls_ready_task
11 }
12 while ls_ready_task is not empty {
13        n = head (ls_ready_task)
14        PE = n.mapping
15        if PE is a hardware component
16                n.t_start = n.t_ready
17        else
18                n = select_task(ls_ready_task, PE)
19                n.t_start = max (n.t_ready, PE.t_available)
20                PE.t_available = PE.t_available + n.t_execution(PE)
21        remove n from ls_ready_task
22        update ls_ready_task and ls_ready_comm
23 }
24 goto step 02
```

Figure 4.11: Proposed combined communication mapping and activity scheduling

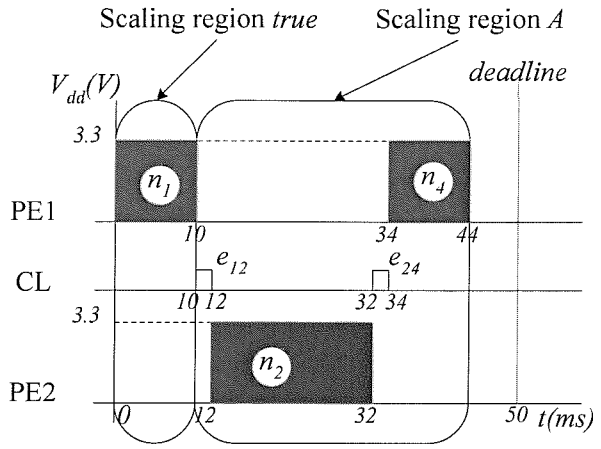
4.3.3 DVS for Enhanced System Model

In Chapter 3, Section 3.3, a conditional behaviour aware DVS technique has been described, but without the consideration of communication. In this section, an enhanced DVS technique is presented to address the impact of communication and the heterogeneous PEs.

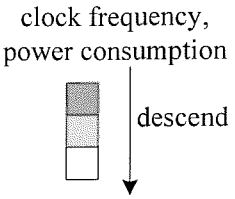
In the DVS technique for CTG of Chapter 3, Section 3.3, the whole schedule table is divided into several scaling regions. After identifying the slack time available for a scaling region, a scaling factor is calculated, and all the tasks in the scaling

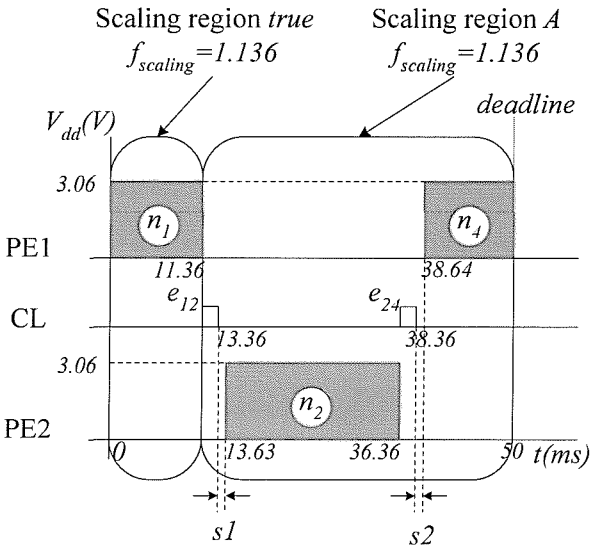
region are scaled with the same scaling factor. This technique works effectively for the basic system model. However, it needs further enhancement in order to achieve energy efficiency for the enhanced system model. For example, consider the motivational example of Figure 4.1(a) and its schedule of Figure 4.3(a) (the track corresponding to condition value *A*). Since n_1 is a disjunction task (Chapter 1, Section 1.3.2), the schedule of Figure 4.3(a) can be divided into two scaling regions as shown in Figure 4.12(a). According to the DVS technique of Chapter 3, Section 3.3, the scaling factors for scaling regions *true* and scaling region *A* both are 1.136. Scaling the two regions with scaling factor 1.136 generates the schedule of Figure 4.12(b). However, as can be seen in the figure, because the CL is not DVS-enabled, the execution time of e_{12} and e_{34} cannot be scaled. Therefore, slack time *s1* and *s2* are wasted, which should have been exploited by other tasks to reduce energy dissipation. This problem also occurs when some PEs are not DVS-enabled. Furthermore, due to the heterogeneous PEs, the tasks within a scaling region may consume different power. In such case, instead of using an identical scaling factor for all tasks within a scaling region, it is beneficial to use variant scaling factors for tasks depending on their power consumption [55]. Highly power-consuming tasks are assigned with more slack time (i.e., larger scaling factor), and low power-consuming tasks are assigned with less slack time (i.e., smaller scaling factor). Based on the above observation, the problems raised by the communication and heterogeneous PEs of the enhanced system model can be solved by controlling the slack time distribution:

(1) avoid distributing slack time to communications or tasks that are mapped to non-DVS-enabled PEs, hence no slack time will be wasted;
(2) distribute slack time to the tasks according to their power consumption (the higher power, the more slack time), in order to achieve maximum energy reduction.



(a) Scaling regions





(b) Improper voltage scaling

Figure 4.12: Improper voltage scaling leading to slack time waste

Figure 4.13 outlines the enhanced DVS technique. The enhancement (highlighted in Figure 4.13) involves the slack times distribution between the tasks within a scaling region, whilst the conditional behaviour aware identification of available slack time for each scaling region is the same as Chapter 3, Figure 3.13, where the technique of [55] is employed to exploit the slack time on non-critical paths within a scaling region (Chapter 3, Section 3.4). The main idea of the enhancement is to use the technique of [55] to control slack time distribution, not only on the non-critical paths, but also

DVS Technique for DVS
Input schedule table - *SchTable*
Deadline - T_d

```
01 pre-process SchTable
02 for (each column col in SchTable, from left to right)
03 {
04      identify the worst case track - track_worst
05      calculate the worst case total slack time - slack_worst
06      calculate the slack time distributed to col - slack_col
07      convert the tasks within col to a task graph TG
08      apply DVS technique in [55] to TG
09      update SchTable
10 }
```

Figure 4.13: DVS technique for enhanced system model

critical paths. In this way, although the computational time increases due to the iterative nature of the technique [55], the slack time is fully exploited for energy efficiency improvement of the enhanced system model. The conditional behaviour aware identification of available slack time and the principle of the technique of [55] have been described in Chapter 3, Section 3.3, the enhancement is explained next. In step 07, after having identified the available slack time for a scaling region (steps 01 – 06), the tasks within the scaling region are converted into a task graph (*TG*), in order to use the technique of [55], which is only applicable to task graphs. In step 08, the technique of [55] is applied to the *TG* to exploit the slack time for energy reduction. The conversion from a scaling region to a task graph is achieved by: (1) from the CTG, extract the part belong to the scaling region; (2) insert a source task and a sink task (Appendix C), and decide the start time of the source task and the deadline of the sink task according to the timing information of the scaling region; (3) for each task n without an inwards edge, insert an edge from the source task to n, for each task n without an outwards edge, insert an edge from n to the sink task. For example, consider the CTG of Figure 4.10, the schedule of its track corresponding to condition value A is given in Figure 4.12(a). Figure 4.14 illustrates the conversion from scaling region A to a task graph. The start time of the source task is 11.36ms. This is because, according to the conditional behaviour aware identification of slack time, the slack

time available for scaling region *true* is 1.36ms, hence the earliest start time of e_{12} is 10ms + 1.36 ms = 11.36ms. Obviously, the deadline of the sink task is 50ms, imposed by the deadline of the original CTG.

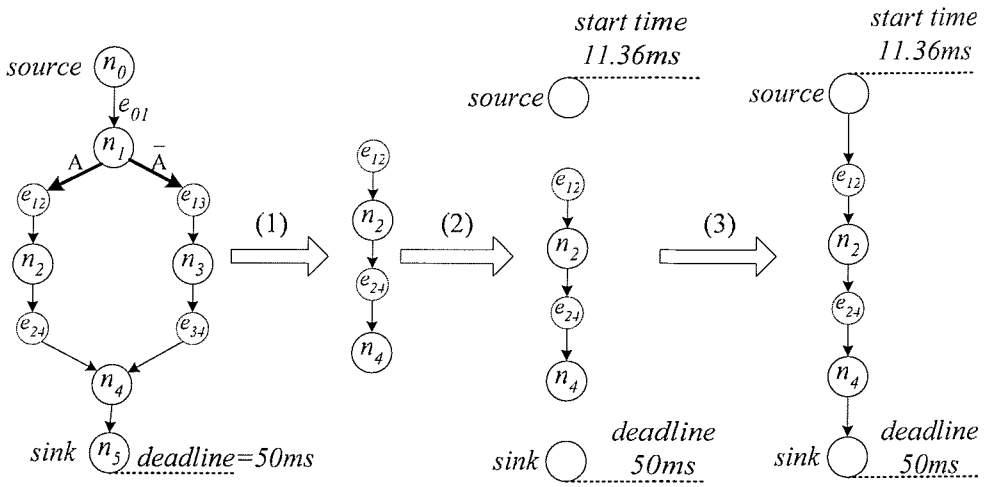


Figure 4.14: Conversion from a scaling region to a task graph (Scaling region A of Figure 4.12)

4.4 Experimental Results

A number of CTG examples have been used to demonstrate the efficiency and the applicability of the proposed communication-integrated co-synthesis technique considering enhanced system models (Section 4.4.1). Also, a performance analysis is carried out to show the effect of alternative communication architectures on the system solution quality, which gives guidance for the design of communication architecture (Section 4.4.2). For details of the simulation set-up, the reader is referred to Appendix A.

4.4.1 Efficiency of the Communication-Integrated Co-Synthesis

Two sets of examples are used to test the efficiency of the communication-integrated co-synthesis technique in terms of energy reduction: (1) a real-life example of GSM voice CODEC; (2) a number of synthetic examples.

GSM Voice CODEC Example

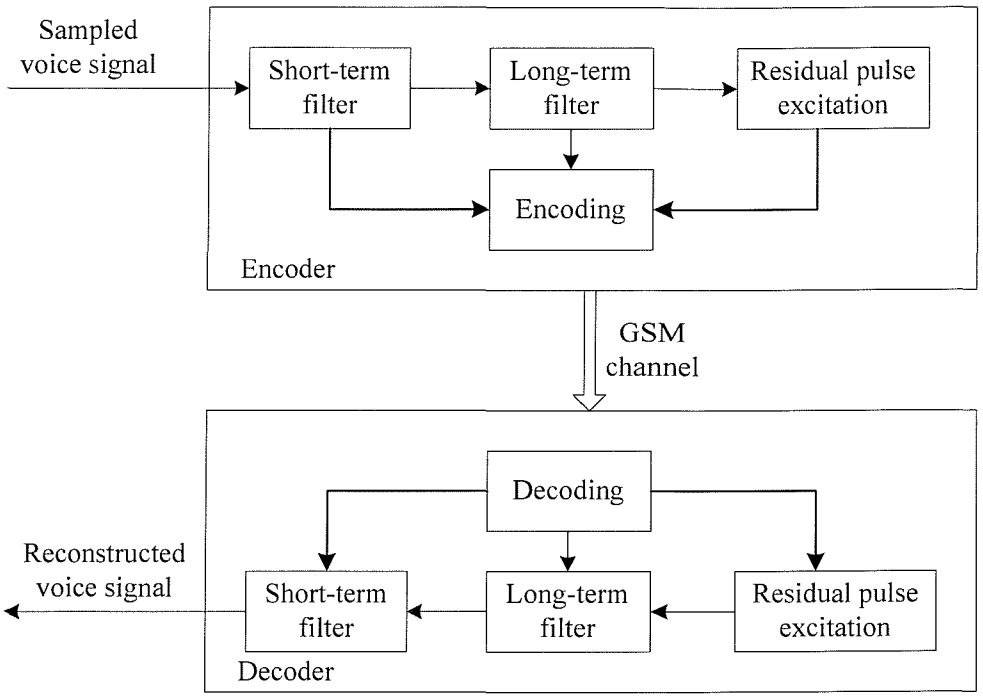


Figure 4.15: Block diagram of the GSM voice CODEC [2]

This experiment is concerned with an energy efficient implementation of a real-life GSM voice CODEC example [2]. GSM voice CODEC is a speech compression/decompression algorithm used in Global System for Mobile Telecommunication (GSM). GSM voice CODEC consists of two parts: encoder and decoder. Figure 4.15 shows the block diagram of the GSM voice CODEC. The encoder divides the incoming voice signal into short-term predictable parts, long-term predictable parts, and the remaining residual pulse. The resulting parameters of the filters and residual pulse are quantised and encoded. Upon receiving the encoded voice stream, the decoder decompressed the parameter settings for the filters and the residual pulse. Using these settings the original voice signal is reconstructed. The task graphs of the encoder and decoder [2] have been derived from the source code of the CODEC [143]. Figure 4.16 and Figure 4.17 respectively show the extracted task graphs of the encoder and decoder. The encoder consists of 53 tasks, whilst the decoder consists of 34 tasks. The task graphs of coder and decoder are combined to form a conditional task graph (CTG). The CTG of CODEC consists of 87 nodes, 139

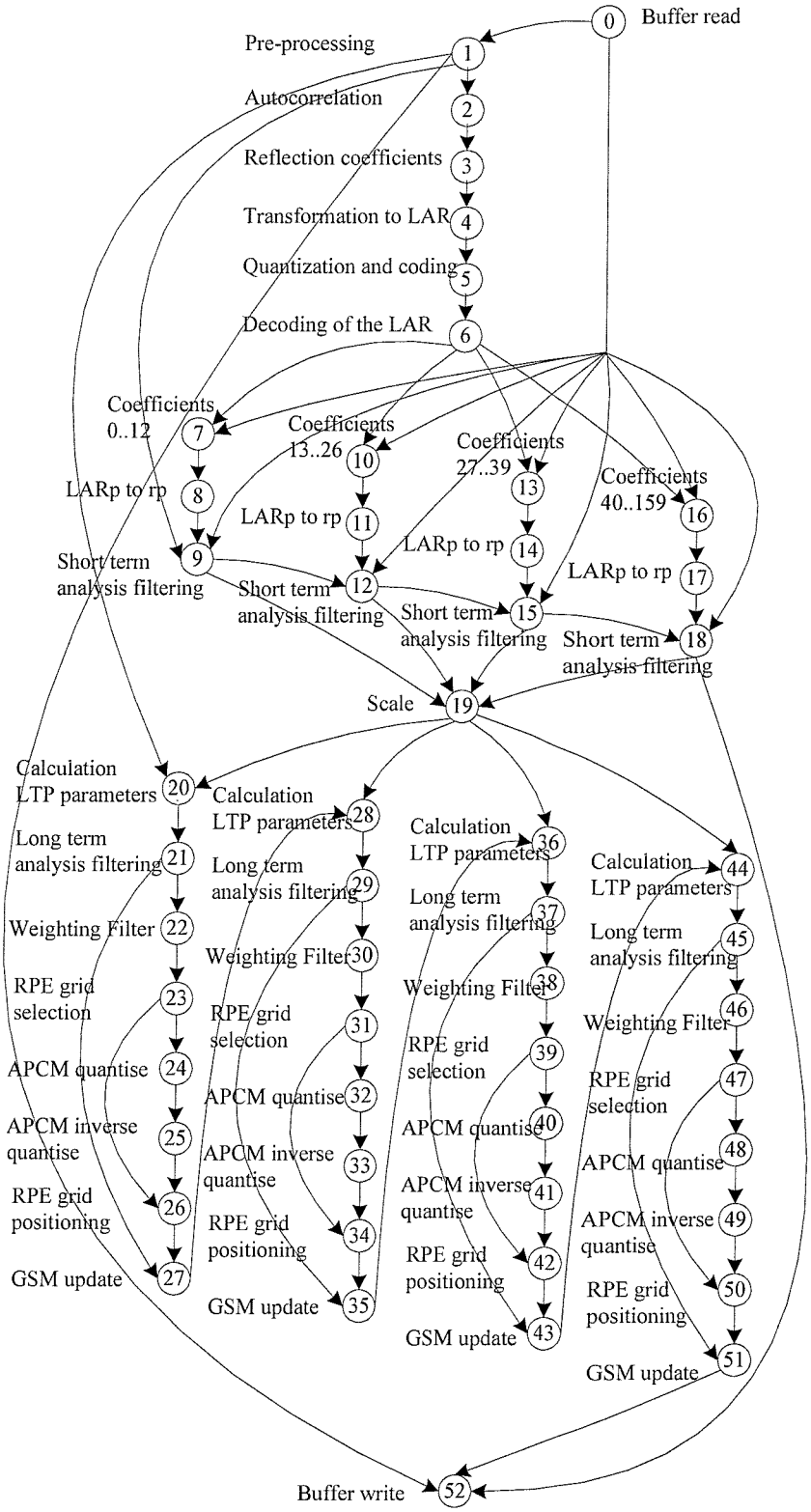


Figure 4.16: Task graph of GSM voice encoder [2]

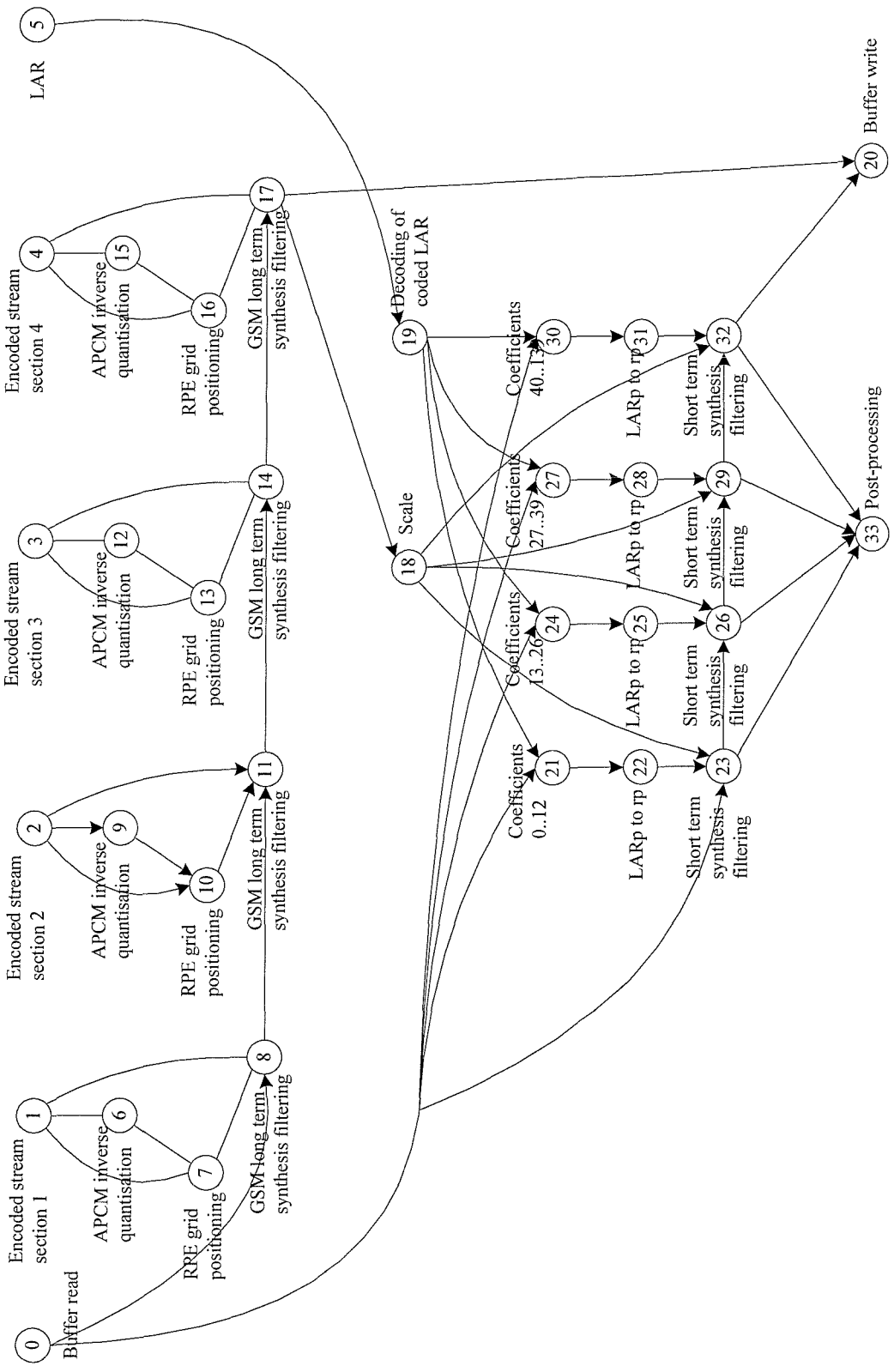


Figure 4.17 Task graph of GSM voice decoder [2]

edges, and two tracks, one track for encoder and another track for decoder. The target architecture of CODEC consists of two processing elements (PEs) connected by a communication link (CL). A technology library gives the properties of the PEs and CL, as well as the implementation properties of the tasks and communications. Details of the technology library are given in Appendix B.

The experiments on the GSM voice CODEC example is used to demonstrate two important aspects. Firstly, it used to show the efficiency of the communication-integrated co-synthesis in terms of energy reduction. Secondly, a comparison is made between the implementations of the CODEC example with and without the consideration of communication, in order to show the impact of communication on energy dissipation. For this purpose three experiments are conducted.

(1) The proposed co-synthesis is used to perform mapping, scheduling and voltage scaling for the CODEC example, without the consideration of communication; (NO COMM 10ms)
(2) The proposed co-synthesis is applied to the CODEC example, taking the communication into consideration; (COMM – 10ms)
(3) Experiment 1 and 2 assume a deadline of 10ms. In experiment 3, a deadline of 20ms is assumed. The proposed co-synthesis is applied to the CODEC example, taking the communication into consideration. (COMM – 20ms)

The results are given in Table 4.4. As it can be seen, in experiment (2), the energy dissipation before DVS is 92.65 mJ, the proposed co-synthesis reduces the energy dissipation to 51.47 mJ by applying DVS, which is a reduction of 44.5%. In experiment (3), the proposed co-synthesis reduces the energy dissipation from 75.26 mJ to 44.66 mJ, a reduction of 40.7%. It is clear that the proposed communication-integrated co-synthesis is efficient in energy reduction. Comparing the results of experiment (1) and (2), it can be seen that the example consumes more energy in experiment (2) than in experiment (1), as expected. This is because the communications consumes energy, and the slack time available for voltage scaling the tasks is reduced due to the execution time of communications.

Experiment	Description	Energy before DVS (mJ)	Energy after DVS (mJ)	Energy Reduction (%)
1	NO COMM – 10ms	90.97	50.39	44.6
2	COMM – 10ms	92.65	51.47	44.5
3	COMM – 20ms	75.26	44.66	40.7

Table 4.4: Results for the GSM voice CODEC example

Synthetic Examples

In order to test the efficiency of the communication-integrated co-synthesis technique in terms of addressing the impact of communications, the examples (ctg1 – ctg26) of Chapter 3, Section 3.5.2 have been extended with communications. Two types of communication links (CLs) with different baud-rate and power consumption are considered, where CL1 has baud-rate and power consumption of 115 Kbits/s and 4mW, CL2 has baud-rate and power consumption of 1 Mbits/s and 35mW. Three experiments are conducted on the examples:

(1) applying the proposed co-synthesis techniques (including mapping, scheduling, and voltage scaling) to the examples, where CL1 is employed;
(2) applying the proposed co-synthesis techniques to the examples, where CL2 is employed;
(3) to indicate the extra energy dissipation caused by communications, experiment is also performed assuming no communication is considered.

Table 4.5 gives the results of the three experiments. Column 2 shows the complexity of the examples in terms of the number of node (task), edge, condition, PE and CL; column 3 gives the deadline of the examples. The results of experiment (1) are given in Columns 4 – 6, where column 4 and column 5 are the energy dissipation before and after applying DVS; column 6 is the energy reduction achieved by applying DVS, which is calculated as (1 – (column 5 / column 4)). Similarly, the results of experiment (2) are given in columns 7 – 9. Column 10 shows the energy dissipation after applying DVS in experiment 3 (no communication). For all results produced by experiments (1) and (2), as expected, the proposed co-synthesis

Example	No. of node/edge/ condition/ PE/CL	dead -line (ms)	Experiment 1 – CL1			Experiment 2 – CL2			Exp.3 – no com. Ene. (mJ)
			Energy dissipation (mJ)		Ene Redu. (%)	Energy dissipation (mJ)		Ene Redu. (%)	
			Before DVS	After DVS		Before DVS	After DVS		
comctg1	13/16/2/2/1	105	495.12	330.31	33.29	500.11	302.13	39.59	297.26
comctg2	13/16/2/3/1	81	471.44	320.92	31.93	441.55	264.16	40.17	256.02
comctg3	13/16/2/4/1	87	562.69	387.51	31.13	551.51	321.65	41.68	305.42
comctg4	13/16/2/5/2	95	471.53	315.62	33.06	449.07	234.22	47.84	226.65
comctg5	13/16/3/2/1	106	575.09	454.63	20.95	565.15	412.68	26.98	407.90
comctg6	13/16/3/4/1	113	581.43	429.04	26.21	560.16	359.49	35.82	352.48
comctg7	25/30/2/2/1	246	1372.90	1061.27	22.70	1353.03	922.38	31.83	912.75
comctg8	25/30/2/3/1	215	1257.86	1012.52	19.50	1250.59	912.83	27.01	892.92
comctg9	25/30/2/4/1	170	980.36	738.80	24.64	950.64	580.97	38.89	534.17
comctg10	25/30/2/5/2	206	1135.40	782.78	31.06	1090.63	667.52	38.79	612.15
comctg11	25/30/3/2/1	211	1125.20	834.49	25.84	1125.34	773.61	31.26	766.21
comctg12	25/30/3/3/1	212	1216.63	938.44	22.87	1191.84	811.74	31.89	764.46
comctg13	25/30/3/4/1	225	1114.23	780.64	29.94	1079.34	677.68	37.21	648.77
comctg14	25/30/3/5/2	194	1120.47	965.69	13.81	1045.56	686.32	34.36	662.52
comctg15	25/29/4/2/1	201	1025.41	814.01	20.62	1007.84	719.28	28.63	708.78
comctg16	25/29/4/3/1	157	880.33	708.55	19.51	845.50	577.54	31.69	560.25
comctg17	25/29/4/4/1	196	970.27	750.03	22.70	969.30	663.86	31.51	623.55
comctg18	25/29/4/5/2	162	877.87	714.09	18.66	806.84	535.88	33.58	513.27
comctg19	35/41/2/2/1	351	1272.91	949.69	25.39	1260.44	850.12	32.55	837.44
comctg20	37/45/2/3/1	420	1638.12	1271.03	22.41	1574.76	1118.97	28.94	1086.21
comctg21	35/41/2/5/2	328	1255.56	978.80	22.04	1175.77	714.03	39.27	678.92
comctg22	38/48/2/2/1	272	1990.69	1557.13	21.78	1955.89	1458.99	25.41	1437.42
comctg23	48/60/3/3/1	465	1711.04	1228.65	28.19	1639.60	1009.10	38.45	972.03
comctg24	46/55/3/5/2	441	1372.63	975.78	28.91	1323.84	750.18	43.33	735.70
comctg25	59/71/3/3/1	567	3266.17	2437.44	25.37	3250.11	2155.51	33.68	2089.33
comctg26	93/118/5/5/2	601	2269.86	1589.81	29.96	2328.41	1326.13	43.05	1269.70

Table 4.5: Results of the co-synthesis techniques considering communications

technique reduced the energy dissipation effectively by applying voltage scaling. Comparing the results of experiments (1) – (3), it can be seen that the examples consumes more energy when communications are considered. Also, the energy dissipation of the examples in experiment (2) (employing CL2) is less than in experiment (1) (employing CL1), for example comctg14 consumes 686.32mJ employing CL2, while consumes 965.69mJ employing CL1, a reduction of 28.93%. This is because CL1 and CL2 consumes similar energy per bit (i.e., energy needed to transmit a bit through the CL), however, using the faster CL (CL2) leads to shorter

execution time for communications, which in turn leaves more slack time for voltage scaling the tasks.

4.4.2 Effects of Alternative Communication Architectures

In this section, the proposed co-synthesis technique is used to examine the effect of alternative communication architectures on the quality of system solutions. Experiments are conducted on a CTG, which has 59 nodes and 71 edges, mapped to an architecture containing three PEs. The PEs are interconnected by three alternative communication architectures as shown in Table 4.6. In case 1 the PEs are connected by three dedicated CLs (Figure 4.5(a)); in case 2, the PEs are connected by a shared bus (Figure 4.5(b)), whose baud-rate and power consumption are the sum of the three dedicated CLs of case 1; in case 3, the shared CL of case 2 is split into two shared CLs (Figure 4.5(c)), each has half of the baud-rate and power consumption.

	Communication Architecture	**Baud-rate (Kbits/s)**	**Power Consumption (mW)**
Case 1: 3 dedicated CLs	PEs are connected by three dedicated CLs, Figure 4.5(a)	150	23.3
Case 2: 1 shared CL	PEs are connected by one shared CL, Figure 4.5(b)	450	70
Case 3: 2 split shared CLs	PEs are connected by two shared CLs, Figure 4.5(c)	225	35

Table 4.6: Comparison of three alternative communication architectures

The proposed co-synthesis technique is employed to produce solutions for the CTG in the three cases. The results are given in Table 4.7. Again, it is clear that the proposed co-synthesis techniques improve the energy efficiency for all the cases. Additionally, Table 4.7 gives a comparison of the system solution quality under three cases. Case 3 (2 split shared CLs) provides better solutions than case 1 (3 dedicated CLs). This is because the split shared CL has higher baud-rate than the dedicated CL, and the two split shared CLs may allow for greater communication parallelisms. Nevertheless, the solution of case 3 is poorer than that of Case 2 (1 shared CL). This is because, for this particular CTG, there are few communications in parallel, thus the

advantage of splitting the CL and thereby increasing parallelism is countered by the lower baud-rate of each CL. As it can be seen, a suitable selection of communication architecture can significantly improve system quality in terms of energy efficiency. The proposed co-synthesis technique selects the appropriate communication architecture to reduce energy dissipation for a particular design.

	Case 1	Case 2	Case 3
Schedule length (ms)	529.6	499.13	499.73
Deadline (ms)	540	540	540
Energy before DVS (mJ)	4352.48	4143.83	4200.58
Energy after DVS (mJ)	3578.89	3291.15	3387.61

Table 4.7: Results for three alternative communication architectures

4.5 Concluding Remarks

This chapter has introduced the concept of enhanced system model, in order to address the impact of communication. Consideration of communications is integrated within the co-synthesis technique of Chapter 3 to perform mapping, scheduling, and voltage scaling for enhanced system models. In the proposed co-synthesis technique, task mapping and communication mapping are performed separately, communication mapping is carried out simultaneously with the activity scheduling. The scheduling and DVS technique are extended to perform scheduling and voltage scaling for enhanced system models. Experimental results show that the proposed co-synthesis techniques achieve significant energy reduction after considering the impacts of communications and heterogeneous PEs. Also, performance analysis is carried out to show the effect of alternative communication architectures on the system solution quality in terms of energy efficiency. It is shown that a suitable selection of communication architecture can significantly improve the system solutions.

Chapter 5

Power-Composition Profile Driven Co-Synthesis

Dynamic voltage scaling (DVS) employed in the co-synthesis techniques of Chapter 3 and Chapter 4 aims to reduce dynamic power, which has traditionally been the primary source of power consumption. However, as the CMOS feature size continues to shrink, leakage power increases exponentially and will become an important source of power consumption in < 0.1μm CMOS technology (Chapter 2, Figure 2.3). This justifies the employment of adaptive body biasing (ABB, Chapter 2, Section 2.1.2.2), which has recently been proposed to reduce leakage power [60-64]. Nevertheless, most of these research [60-64] focus on the circuit level, little research has been reported in the system level. This chapter presents a new system-level co-synthesis technique to reduce dynamic and leakage power simultaneously for embedded systems that contain processing elements (PEs) with DVS and/or ABB capabilities. In particular, the presented techniques perform a power management selection at the architectural level, i.e., it decides upon which PE to be equipped with which power management scheme (DVS, ABB, or combined DVS and ABB [77, 115]) – with the aim to achieve high energy saving at a reduced implementation cost. The techniques also map, schedule, and voltage scale applications specified as task graphs with timing constraints.

The remaining of this chapter is organised as follows. Section 5.1 outlines the motivation of the proposed power-composition profile driven co-synthesis technique. Power and delay models used in the technique are introduced in Section 5.2. Section 5.3 gives two motivational examples to show the main idea of the proposed technique. Sections 5.4 and 5.5 detail the problem formulation and present the proposed co-synthesis technique with power management selection. Section 5.6 gives the extensive experimental results. Finally, concluding remarks are given in Section 5.7.

5.1 Motivation

Dynamic voltage scaling (DVS) and adaptive body biasing (ABB) are two system-level energy management techniques, both allowing trade-off between system performance and energy dissipation during application run-time (Chapter 2, Section 2.1.2). The main difference between them is that DVS scales down the circuit supply voltage to primarily reduce dynamic power, whilst ABB increases the circuit threshold voltage through body biasing to mainly reduce leakage power [60-62, 72]. Certainly, either decreasing the supply voltage or increasing the threshold voltage requires lowering of the clock frequency. The reported techniques [53, 55, 56, 60-62, 72] employ either DVS or ABB to reduce system energy dissipation. However, for foreseeable future systems where dynamic power and leakage power are comparable to each other, neither DVS nor ABB in isolation can achieve the best energy efficiency. Duarte *et al* [77] showed that a combined DVS and ABB will provide the highest energy saving. In such a combined scheme, DVS and ABB compete for the redundant system performance (i.e., slack time) to scale down the system's clock frequency. Therefore, the difficulty in employing combined DVS and ABB is to determine the trade-off between them [63]. Martin *et al* [115] developed analytical models to produce optimal supply voltage and threshold voltage values for energy minimisation. Yan *et al* [144] and Andrei *et al* [145] addressed the supply voltage and body bias voltage scaling problem for task graphs executed on multi-processor systems, unlike the previous work [115] which targets single processor systems. Although the combination of DVS and ABB [63, 115, 144, 145] (referred to as DVS+ABB from now on) provides higher energy saving than DVS or ABB in isolation, it increases system cost and complexity. This is because the implementation of DVS requires an additional voltage converter and impacts processor verification [70]. The implementation of ABB, on the other hand, requires a bias voltage generator and a substrate-bias distribution, i.e., additional wiring [60, 62]. Clearly, combined DVS+ABB implies larger system cost than separate DVS or ABB. Cost and energy are two critical and competing concerns in embedded systems; therefore, it is

important to carefully balance these two aspects as early as possible in the design phase.

As a result, one important problem is the selection of the appropriate power management scheme (DVS, ABB, or DVS+ABB) for each processing element (PE) in the system, in order to find a suitable trade-off between system cost and energy dissipation. The work presented in this chapter addresses this problem and introduces a new methodology that performs the power management selection (PMS) during the co-synthesis. It will be demonstrated that, depending on the ratio of the dynamic power to the leakage power (i.e., power-composition profile), it is possible to employ PEs with separate DVS or ABB capability to achieve comparable energy saving to DVS+ABB PEs while avoiding the extra system cost. In addition to the PMS, the proposed power-composition profile aware co-synthesis methodology maps, schedules and voltage scales applications given as task graphs with timing constraints. The co-synthesis provides the designer with a set of possible design solutions, represented by Pareto-optimal trade-off points in terms of energy and cost, from which the designer can select and decide which PE to be equipped with which power management scheme (DVS, ABB, DVS+ABB).

5.2 Power and Delay Models

Although Chapter 2, Section 2.1.1 has given basic equations for the calculation of dynamic power and leakage power. More elaborate equations are needed to apply DVS and or/ABB to the embedded systems. The power and delay models considering dynamic power only has been given in Chapter 3, Section 3.1.1. This section derives the power and delay models considering both dynamic power and leakage power.

The threshold voltage of MOSFET has a linear dependence on V_{dd} and V_{bs}, as given by [115]:

$$V_{th} = V_{th1} - k_1 \cdot V_{dd} - k_2 \cdot V_{bs} \tag{5.1}$$

where V_{th}, V_{dd} and V_{bs} are the threshold voltage, supply voltage and body bias voltage respectively; V_{th1}, k_1 and k_2 are constants for a given CMOS technology. The circuit delay d is dependent on the supply voltage and threshold voltage, as given by Chapter 2, Equation 2.11. Substituting Equation 5.1 into Equation 2.11 yields the expression of clock frequency f in terms of V_{dd} and V_{bs}:

$$f = \frac{1}{d} = \frac{[(1+k_1)\cdot V_{dd} + k_2 \cdot V_{bs} - V_{th1}]^2}{k_d \cdot V_{dd}} \tag{5.2}$$

where k_d is given by Equation 2.12. Hence the execution time of a task τ is given by:

$$t(\tau) = \frac{N_C(\tau)}{f} \tag{5.3}$$

where $N_C(\tau)$ is the number of cycle needed to execute task τ. There are two major sources of power consumption: dynamic power and leakage power, which are given by Chapter 2, Equations 2.10 and 2.17. Substituting Equation 5.1 into Equation 2.17, the total power consumption consumed by a PE to execute a task can be derived:

$$P_{total} = \underbrace{C_{eff} \cdot f \cdot V_{dd}^{\ 2}}_{P_{dyn}} + \underbrace{L_g \cdot V_{dd} \cdot k_3 \cdot e^{k_4 \cdot V_{dd}} \cdot e^{k_5 \cdot V_{bs}}}_{P_{leak}} \tag{5.4}$$

$$k_3 = I_0 \cdot e^{\frac{V_{GS} - V_{th1}}{n \cdot V_T}} \tag{5.5}$$

$$k_4 = \frac{k_1}{n \cdot V_T} \tag{5.6}$$

$$k_5 = \frac{k_2}{n \cdot V_T} \tag{5.7}$$

where P_{dyn} and P_{leak} denote dynamic power and leakage power respectively, C_{eff} is the effective capacitance of the whole PE averaged over the execution duration of the task,

L_g denotes the number of gates, k_3, k_4 and k_5 are constants for a given CMOS technology. In this chapter, the concept of power-composition profile (PCP) is defined.

Definition *Given a PE and a task τ, the power-composition profile (PCP) is defined as the ratio of the dynamic power to the leakage power consumed by the PE to execute task τ.* Mathematically:

$$PCP(\tau, PE) = \frac{P_{dyn}(\tau, PE)}{P_{leak}(\tau, PE)} \tag{5.8}$$

Since the tasks have different dynamic power and leakage power, the PCP varies from task to task. On the other hand, for a task that can be potentially mapped to several PEs, its dynamic power, leakage power, and therefore PCP varies with the task's mapping. So PCP is a feature of a task and its particular mapping to a PE. The PCPs of the tasks can be estimated using, for example, the technique and component library outlined by Duarte *et al* [106]. The concept of PCP will be used in the proposed co-synthesis techniques to achieve energy efficiency at low system cost (Section 5.5).

Consider a PE executing a task τ. Assume the PE needs $N_C(\tau)$ cycles to execute task τ, and the deadline of task τ is D. The nominal supply voltage and body bias voltage of the PE are $V_{dd}(nom)$ and $V_{bs}(nom)$ respectively. The PE is equipped with a certain type of power management (PM) scheme (DVS, ABB, or DVS+ABB). Equations for the calculation of supply voltage V_{dd}, body bias voltage V_{bs} and clock frequency f, at which the PE can finish executing task τ at the deadline D while at the same time reduce the energy dissipation as much as possible, are derived in three cases: (1) the PE has DVS capability; (2) the PE has ABB capability; (3) the PE has DVS+ABB capability.

(1) If the PE has DVS capability, V_{bs} is kept constant at $V_{bs}(nom)$, while V_{dd} and f can be dynamically scaled according to the performance requirement:

$$\begin{cases} V_{bs} = V_{bs}(nom) \\ f = \dfrac{N_C(\tau)}{D} \\ f = \dfrac{\left[(1+k_1)\cdot V_{dd} + k_2 \cdot V_{bs} - V_{th1}\right]^2}{k_d \cdot V_{dd}} \end{cases} \tag{5.9}$$

(2) If the PE has ABB capability, V_{dd} is kept constant at $V_{dd}(nom)$, while V_{bs} and f are scaled dynamically:

$$\begin{cases} V_{dd} = V_{dd}(nom) \\ f = \dfrac{N_C(\tau)}{D} \\ f = \dfrac{\left[(1+k_1)\cdot V_{dd} + k_2 \cdot V_{bs} - V_{th1}\right]^2}{k_d \cdot V_{dd}} \end{cases} \tag{5.10}$$

(3) If the PE has DVS+ABB capability, V_{dd} and V_{bs} can be scaled simultaneously. In this case, there are infinite numbers of (V_{dd}, V_{bs}) pairs at which the PE can finish executing τ at the deadline D. However, there is only one optimal (V_{dd}, V_{bs}) pair at which the PE consumes the least energy [115]. This optimal (V_{dd}, V_{bs}) pair can be solved using the following nonlinear programming formulation:

Minimise

$$\frac{N_C(\tau)}{f} \cdot \left(C_{eff} \cdot f \cdot V_{dd}^{\ 2} + L_g \cdot V_{dd} \cdot k_3 \cdot e^{k_4 \cdot V_{dd}} \cdot e^{k_5 \cdot V_{bs}}\right) \tag{5.11}$$

subject to

$$f = \frac{N_C(\tau)}{D} = \frac{\left[(1+k_1)\cdot V_{dd} + k_2 \cdot V_{bs} - V_{th1}\right]^2}{k_d \cdot V_{dd}} \tag{5.12}$$

Equations 5.1 – 5.12 will be used in the PCP driven co-synthesis methodology of Section 5.5.

5.3 Motivational Examples

This section illustrates the main idea behind the proposed co-synthesis methodology by means of two motivational examples. The first example demonstrates the impact of power-composition profile (PCP) in the selection of various power management (PM) schemes for the PEs. The second example shows the influence of task mapping on the PCP and why it is important to consider PCP during the task mapping.

5.3.1 Example 1

Consider a voice compression task with an execution time of 10ms at nominal supply voltage and body bias voltage and a deadline of 13ms (i.e., 3ms slack). Figure 5.1 shows the normalised energy dissipations of the task when various PM scheme are employed for two different PCPs. Consider PCP 1, where both dynamic power and leakage power are 4.67mW (i.e., dynamic power / leakage power = 1). The power values are derived using Equation 5.4 assuming the 0.07μm Crusoe processor constants [115]. As it can be seen, the normalised energy dissipation without any voltage scaling is 1.0 (V_{dd} = 1V, V_{bs} = 0V), employing DVS or ABB reduces the energy dissipation to 0.789 (V_{dd} = 0.85V, V_{bs} = 0V) and 0.541 (V_{dd} = 1V, V_{bs} = –0.66V) respectively. ABB produces higher energy reduction than DVS because according to Equation 5.4, dynamic power is quadratically dependent on supply voltage, while leakage power is exponentially dependent on body bias voltage (leakage power is also exponential to V_{dd}, but the constant k_4 for V_{dd} is smaller than k_5 for V_{bs}). Employing DVS+ABB [115] reduces the energy dissipation to 0.537 (V_{dd} = 0.98V, V_{bs} = –0.55V). This shows that ABB produces similar energy reduction as DVS+ABB. In this case, a suitable PM scheme is ABB because of its similar energy reduction to that of DVS+ABB but less system cost.

Now consider PCP 2, where the dynamic power and leakage power are 7.47mW and 1.87mW respectively (i.e., dynamic power / leakage power = 4). The normalised energy dissipation without any voltage scaling is 1.0 (V_{dd} = 1V, V_{bs} = 0V). Employing DVS, ABB, DVS+ABB respectively reduces the energy dissipation to 0.754 (V_{dd} =

0.85V, V_{bs} = 0V), 0.817 (V_{dd} = 1V, V_{bs} = –0.66V), and 0.731 (V_{dd} = 0.89V, V_{bs} = –0.18V). As expected, the DVS+ABB again achieves the highest energy reduction. DVS performs slightly worse than DVS+ABB, but much better than ABB, because the dynamic power is dominating in the total power. There is a trade-off in the selection of the PM scheme. One can choose DVS+ABB for lowest energy consumption at the expense of extra system cost, or choose DVS with slight degradation in energy reduction and reduced system cost.

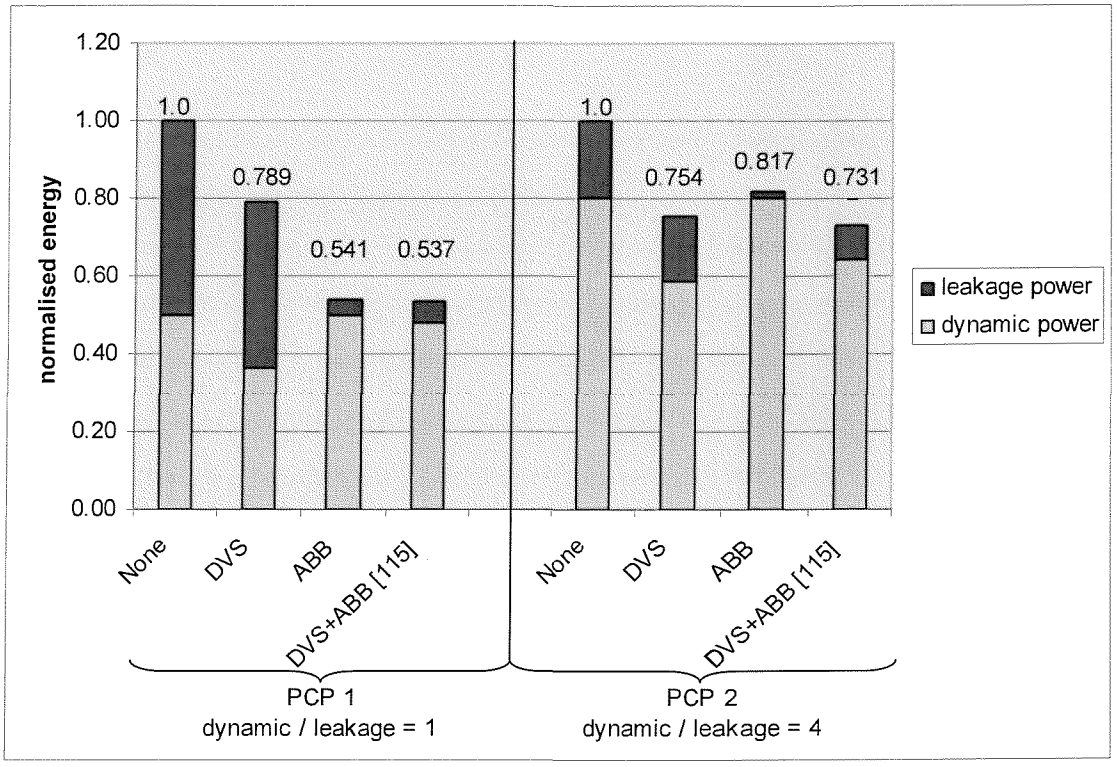


Figure 5.1: Energy dissipation using different PM schemes for two PCPs

Figure 5.2 shows the normalised energy dissipation of the same task using various PM schemes for power-composition profiles (PCP) in the range of 1 to 4, considering current and future CMOS technologies (<0.1μm) [106]. First, consider the energy reduction of ABB and DVS+ABB. It can be seen that both schemes have comparable energy reduction in the PCP range of 1 to 2.25 (Zone I), but DVS+ABB has much higher energy reduction than ABB in Zone II and III. Now consider the energy reduction of DVS and DVS+ABB. As it can be seen, the two schemes have

comparable energy reduction in Zone III, but DVS+ABB has much higher energy reduction than DVS in Zone I and II. Based on these comparisons, it can be observed that, for a single task to be executed on a PE, if the PCP is in Zone I, then ABB can be selected with little degradation in energy reduction but less system cost when compared with DVS+ABB. Similarly, if the PCP is in Zone III, then DVS can be selected. Finally, if the PCP is in Zone II, DVS+ABB should be selected because it performs much better than either DVS or ABB. Note that the zone ranges of Figure 5.2 are for the case of 0.07μm Crusoe processor [115] as an example. This motivational example shows the importance of taking into account the PCP when identifying PM schemes for the PEs in order to produce energy and cost efficient designs.

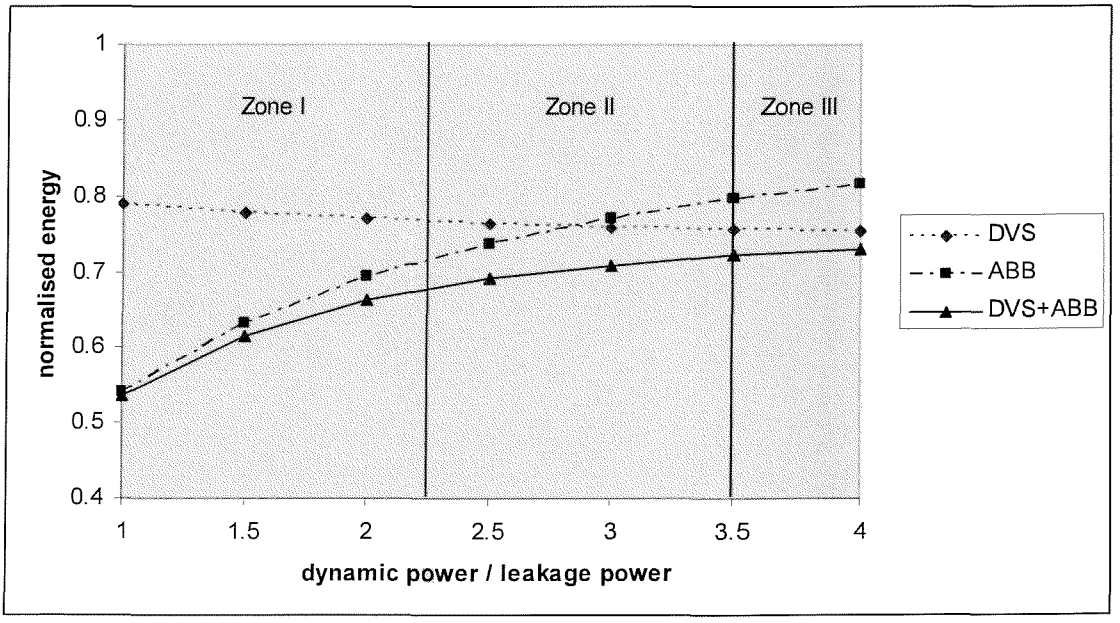


Figure 5.2: Energy dissipation using different PM schemes for a range of PCPs

5.3.2 Example 2

As mentioned in Section 5.2, a task's PCP depends on its mapping. To examine this influence, consider the same task as in the Example 1, but this time it can be possibly mapped to two different PEs, which have already been equipped with certain types of power management scheme: PE_1 with DVS capability and PE_2 with ABB capability. It

is assumed that the execution properties of the task on the two PEs are as shown in Table 5.1. It can be seen that, the PCP is 3 when the task is mapped to PE_1, and the PCP is 2 when the task is mapped to PE_2. Hence the two mappings generate different PCPs. Next we decide to which PE the task should be mapped with the aim of high energy saving. Based on the knowledge that the total power consumption of the task is lower on PE_1 (13.44mW) than on PE_2 (14.01mW), one may possibly map the task to PE_1. In this case, the energy dissipation without any voltage scaling is 134.4μJ (V_{dd} = 1V, V_{bs} = 0V). Applying DVS reduces the energy dissipation to 102.2μJ (V_{dd} = 0.85V, V_{bs} = 0V). However, if one maps the task to PE_2, although the energy dissipation without any voltage scaling (VS) is worse (140.1μJ), applying ABB reduces the energy dissipation to 97.3μJ (V_{dd} = 1V, V_{bs} = –0.66V), which is better than being mapped to PE_1. This is because PE_2 has ABB capability, which is able to efficiently reduce the energy of the task with a PCP of 2, as shown in Figure 5.2. On the other hand, the DVS capability available on PE_1 is less efficient in reducing energy for the task with a PCP of 3. This motivational example shows the influence of the mapping on the PCP and the importance to consider the PCP during the mapping to achieve good energy saving.

	execution time (ms)	Pdyn (mW)	Pleak (mW)	Ptotal (mW)	PCP	energy before VS (μJ)	energy after VS (μJ)
PE1 (DVS)	10	10.08	3.36	13.44	3	134.4	102.2
PE2 (ABB)	10	9.34	4.67	14.01	2	140.1	97.3

Table 5.1: Execution properties for different mappings

Motivational examples have demonstrated that it is necessary to select power management (PM) scheme for PEs appropriately depending on the PCP. Clearly, in the single PE/single task case the decision upon which PM scheme to employ can be taken directly based on the above given observations. Once the execution property of the task (including execution time, deadline, dynamic power, leakage power, and, in turn, PCP) is known, the selection can be made simply by comparing the PCPs in a diagram as shown in Figure 5.2. However, it is important to note that in the case of multiple PEs that execute a set of tasks, this problem becomes non-trivial. This has the following reasons: (1) The PCPs of tasks cannot be fixed before the actual tasks

mapping have been determined, i.e., the PCPs of tasks change depending on which PE the tasks are assigned to, and (2) the tasks can have different dynamic power, leakage power and PCP, and the distribution of slack time between the tasks is non-uniform in order to achieve the best energy saving [55]. In general, the search space of the PMS problem is $M^{|P|}$, where M is the number of PM schemes (for instance, 4 for None, DVS, ABB, DVS+ABB) and |P| is the number of PEs in the system. Furthermore, the interrelation between task mapping and PCP hampers the usage of effective constructive heuristics. Before introducing our co-synthesis technique with PMS, a detailed problem formulation is given next.

5.4 Problem formulation

The input to the proposed co-synthesis methodology includes a task graph $G(V,E)$, an architecture, and a technology library. In the architecture, the number and type of PEs have been decided, but the type of power management scheme (DVS, ABB, or DVS/ABB) of each PE is not fixed. Each power management (PM) scheme is associated with a cost factor reflecting its hardware overhead, which is given in the technology library. Each task $\tau \in V$ in the task graph can be potentially mapped to several PEs able to execute it. For each possible mapping, the number of clock cycle N_c required to execute the task, the dynamic power P_{dyn} and the leakage power P_{leak} at the nominal supply voltage and body bias voltage are also given in the technology library. These values are either based on previous design experience or on estimation techniques [133, 134]. The goal of the co-synthesis is to select a suitable PM scheme for each PE, and to find a mapping, scheduling, supply voltage and/or body bias voltage assignment for the tasks (depending on the PM scheme selected for the PE), such that the imposed deadlines are met, and at the same time, the energy dissipation and system cost are reduced. The proposed methodology provides the designer with trade-offs between two optimisation objectives: energy dissipation and system cost. An assumption is made that the tasks are of sufficiently coarse granularity and that the PEs can continue operation during the voltage scaling, which allows neglecting the scaling overhead in terms of power and time [2]. If such overheads cannot be

neglected, techniques such as those presented recently [145] can be used to take the overhead into consideration during the optimisation process.

5.5 PCP Driven Co-Synthesis with Power Management Selection

This section presents co-synthesis methodologies, which, aware of the tasks' power-composition profile (PCP), perform a power management selection (PMS) with the aim to find suitable trade-offs between energy dissipation and system cost. In addition it optimises mapping, scheduling, and voltage assignment towards dynamic and leakage energy saving. Two alternative methodologies are presented: methodology 1 performs an exhaustive design space search to find the optimal power management selection (Section 5.5.1); while methodology 2 employs a heuristic to search for near-optimal PMS (Section 5.5.2), leading to shorter computational time compared with methodology 1.

5.5.1 Methodology 1 - Exhaustive Design Space Search

Methodology 1 is based on two nested optimisation loops, as shown in Figure 5.3. The inner loop is a power-composition profile aware mapping. It is responsible for the optimisation of the mapping, scheduling and supply voltage and body bias voltage assignment for a given power management selection (PMS), which is identified in the outer loop. The outer loop exhaustively identifies all possible PMS for the PEs. It starts from a PMS with the highest system cost (i.e., all PEs are equipped with DVS+ABB scheme) and incrementally moves towards PMS with lower system costs. In each iteration, the outer loop

(1) identifies a possible PMS;
(2) passes the identified PMS to the inner loop to generate the corresponding mapping, scheduling, supply voltage and body bias voltage assignment;
(3) checks whether there exists another PMS with equal or lower system cost;

(4) if such a PMS exists, then an actual PMS is identified, e.g., associate a PE with separate DVS or ABB instead of DVS+ABB, and the newly identified PMS is passed to the inner loop.

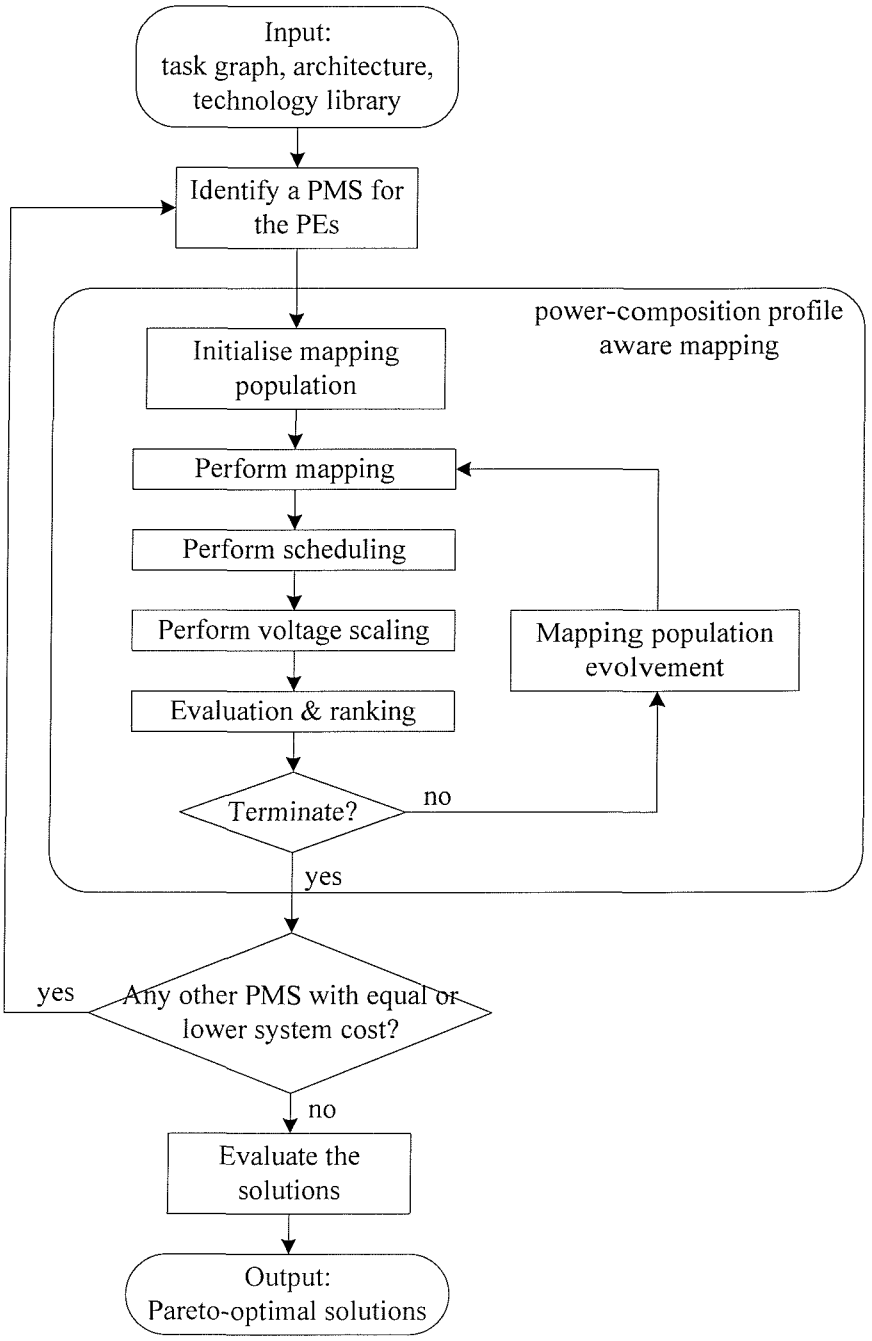


Figure 5.3: Flow of methodology 1 - exhaustive design space search

This procedure repeats until there is no PMS with equal or lower system cost. Finally, all the solutions (including PMS and the corresponding mapping, scheduling and voltage assignment) are evaluated in terms of energy dissipation and system cost. A solution α dominates a solution β, if one of the following conditions is met:

(1) energy(α) $\leq$ energy(β) AND cost(α) < cost(β);
(2) energy(α) < energy(β) AND cost(α) $\leq$ cost(β).

Solutions that are not dominated by any other solutions consist of the Pareto-optimal solutions (Pareto set), which are presented to the designer to give a trade-off between energy dissipation and system cost.

PCP Aware Mapping

The key feature of methodology 1 is the power-composition profile (PCP) aware mapping (the inner loop of Figure 5.3). Taking advantage of the available PCP information, it optimises the mapping, scheduling, supply voltage and body bias voltage assignment in terms of energy dissipation and timing feasibility for a given PMS, which is identified in the outer loop.

The basic idea of the PCP aware mapping is to, for each task, look at the PCPs resulted from all possible mappings, then decide which mapping is preferable. The complexity of this process is O(number of tasks * number of PEs). Denote a specific task as τ_i, a specific PE as PE_j, the PCP of τ_i when it is mapped to PE_j as PCP_{ij}. According to Figure 5.2, there are three cases that τ_i should be preferably mapped to PE_j:

(1) PCP_{ij} is in Zone I (Figure 5.2) and PE_j has ABB capability or DVS+ABB capability;
(2) PCP_{ij} is in Zone II and PE_j has DVS+ABB capability;
(3) PCP_{ij} is in Zone III and PE_j has DVS capability or DVS+ABB capability.

Ideally all tasks need to be mapped according to the PCP. However this is not always possible due to the imposed time constraints, area constraints, or lack of PEs with suitable voltage scaling capabilities. As will be shown later in this section, to meet timing feasibility, some optimised task mappings may not be in accordance with their PCP, although this increases energy dissipation.

```
01 for (each task τ_i) {
02   for (each PE_j able to execute τ_i) {
03     PCP_ij = P_dyn(i,j) / P_leak(i,j)
04     if ((PCP_ij ∈ Zone I) and (PE_j has ABB or DVS+ABB))
05       PE-SET_i = PE-SET_i + PE_j
06     if ((PCP_ij ∈ Zone II) and (PE_j has DVS+ABB))
07       PE-SET_i = PE-SET_i + PE_j
08     if ((PCP_ij ∈ Zone III) and (PE_j has DVS or DVS+ABB))
09       PE-SET_i = PE-SET_i + PE_j
10   }
11   if (PE-SET_i is empty )
12     PE-SET_i = all PEs able to execute τ_i
13 }
14 generate an initial mapping candidate: for each τ_i, choose a PE ∈
     PE-SET_i randomly
15 repeat step 14 to generate a proportion of the initial mapping
     population
16 generate the remaining proportion of the initial mapping
     population randomly
```

Figure 5.4: PCP aware mapping initialisation

The PCP aware mapping is based on a genetic algorithm (GA). For further details concerning the application of genetic algorithm in task mapping the reader is referred to Chapter 3, Section 3.4. As shown in Figure 5.3, the first step of the PCP aware mapping is mapping initialisation. In this step an initial population of mapping genome (each genome represents a mapping candidate) is generated taking account of the PCP. A detailed description of the mapping initialisation is given in Figure 5.4 and explained as follows. According to the previous discussion of this section, there are three cases that τ_i should be preferably mapped to PE_j depending on PCP_{ij} (the PCP of τ_i when it is mapped to PE_j). Correspondingly, steps 01 – 10 identify the PEs

preferable for τ_i to be mapped to, and include them into $PE\text{-}SET_i$. Therefore, $PE\text{-}SET_i$ includes the PEs that have suitable PM scheme to efficiently reduce the energy dissipation of τ_i. If there is no PE preferable for τ_i, then all PEs able to execute τ_i are included in $PE\text{-}SET_i$ (Steps 11 – 12). In steps 14 – 15, $PE\text{-}SET_i$ is used to generate a proportion (50% is found to work practically well) of the initial task mapping population. In step 16, the remaining proportion of population is generated randomly (i.e., non PCP-aware) in order to increase the diversity of the population. This initialisation has been found to improve the optimisation procedure significantly by introducing candidates that are likely to evolve into high quality solutions.

After mapping initialisation, as shown in Figure 5.3, an actual mapping is generated for each genome in the population (Perform mapping). Following this, the execution order of the tasks are scheduled, using a critical path list scheduler similar to [42] (Perform scheduling). After scheduling, supply voltage and/or body bias voltage assignments are generated (Perform voltage scaling), where a technique adopted from [55] is employed. The technique of [55] addresses DVS only. To enable it to address both DVS and ABB, a modification is made, which basically involves the distribution of slack time between DVS and ABB. After performing voltage scaling, the solutions are evaluated and ranked, using the fitness function:

$$Fitness = \left(\sum_i E(n_i)\right) \cdot \left(\frac{\max(T_d, T_e)}{T_d}\right)^2 \tag{5.13}$$

where $E(n_i)$ is the energy dissipation of task n_i, T_d is the deadline of the task graph, and T_e is the actual delay of the task graph. The first part of the fitness function is the total energy dissipation of all tasks. The second part introduces a penalty factor due to deadline violations. Thus the candidates with low energy dissipation and timing feasibility are associated with high fitness and rank. The mapping candidates with high fitness are selected to evolve a new population by mating and mutating them. The produced offsprings are inserted into the population to replace the low ranked ones. In order to improve the optimisation procedure, a new genetic mutation operator is introduced next.

PCP aware mapping improvement: To fully exploit the PM scheme of the PE and decrease the requirement of PEs with DVS+ABB scheme, in each generation, a number of genomes (2% of the population) are picked randomly to be modified as follows. The PCPs of a task resulted from its possible mapping are examined. A task's mapping to a PE is PCP-aware if: (1) the PE has DVS scheme and the task has a high PCP value (3-4); or (2) the PE has ABB scheme and the task has a low PCP value (1-2). The tasks whose mapping is not PCP-aware are randomly remapped to other PEs (if such PE exists) such that the PCP-aware principle is satisfied. Although some of the produced genome by the mutation might be infeasible in terms of timing behaviour, the mutation has been found to improve the optimisation procedure by introducing genes that are likely to evolve into high quality solutions.

This mapping – scheduling – voltage scaling – evaluation and ranking – evolvement procedure continues to optimise the mappings until reaching the termination criteria: no improved individual has been produced for a certain number of generations. In the final optimised mappings, it is possible that some tasks' mappings are not in accordance with the PCP. This is because the optimisation algorithm changes the initial mappings during the evolvement in order to meet timing feasibility, at the expense of increased energy dissipation.

5.5.2 Methodology 2 - Concurrent PMS and Mapping Optimisation

Methodology 1 is effective in finding the best PMS. However, it takes a long computational time due to its exhaustive search nature. The computational time will become unacceptable for systems containing a large number of PEs. To address this problem, methodology 2 is introduced next, which takes shorter computational time, while, at the same, doesn't lose quality of the produced solutions.

Methodology 2 uses a genetic algorithm (GA) to optimise the PMS and mapping concurrently. To explain the basic idea, consider the genome representation shown in Figure 5.5, which encodes both a possible PMS and a mapping. It can be observed that this genome is divided into two sections. The PMS genes determine which PE has which PM scheme (0 – None; 1 – DVS; 2 – ABB; 3 – DVS+ABB), while the mapping

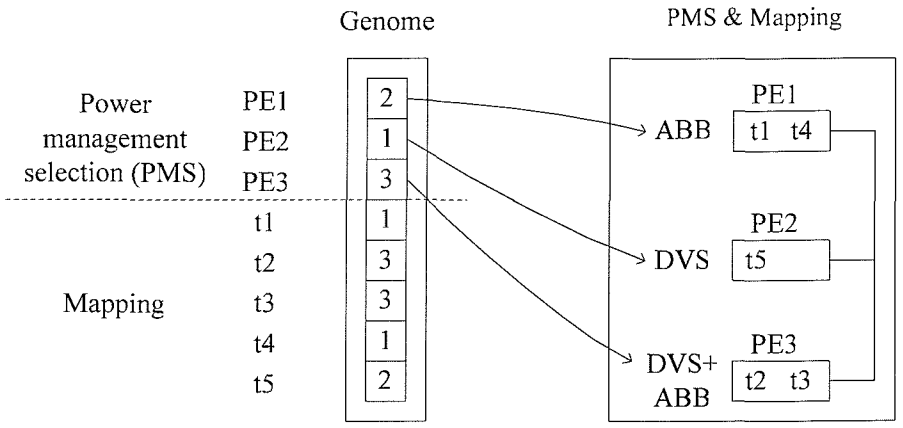


Figure 5.5: Concurrent PMS and mapping genome

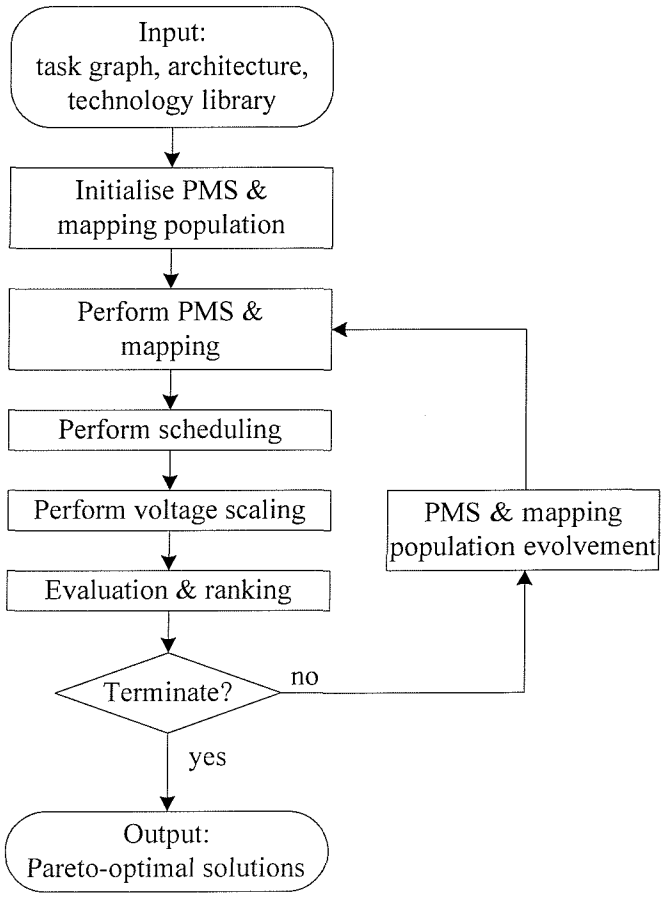


Figure 5.6: Flow of methodology 2 – concurrent PSM and mapping optimisation

genes determine the mapping of tasks onto PEs (1 – PE1; 2 – PE2; 3 – PE3;). The combined genome representation allows the concurrent optimisation of PMS and mapping.

The flow of the methodology 2 is given in Figure 5.6, which is similar to the flow of PCP-aware mapping (inner loop of Figure 5.3), apart from several important differences, which are detailed next.

- **Initialise PMS & mapping population**: the mapping section of the initial genomes are initialised in the same way as methodology 1 (Figure 5.4), while the PMS section of the initial genomes are created randomly (random in the sense that a random choice is taken among the valid possibilities).
- **Perform PMS & mapping**: a PMS and mapping is generated for each genome in the population.
- **Perform scheduling & Perform voltage scaling**: the execution order of tasks, as well as supply voltage and body bias voltage assignment, are decided in the same way as methodology 1.
- **Evaluation & ranking**: the solution resulted from each genome is evaluated and ranked in terms of system cost and fitness, which is calculated using Equation 5.13. A solution α dominates a solution β, if one of the following conditions is met:

  - $energy(\alpha) \leq energy(\beta)$ AND $cost(\alpha) < cost(\beta)$;
  - $energy(\alpha) < energy(\beta)$ AND $cost(\alpha) \leq cost(\beta)$.

  Solutions are ranked according to their Pareto-rank, which is the number of other designs in the population, which does not dominate it [88]. For example, in Figure 5.7, each circle represents a solution. Each solution's cost and energy dissipation are indicated by the position of its circle in the graph. The number in each circle indicates the Pareto-rank of the associated solution.

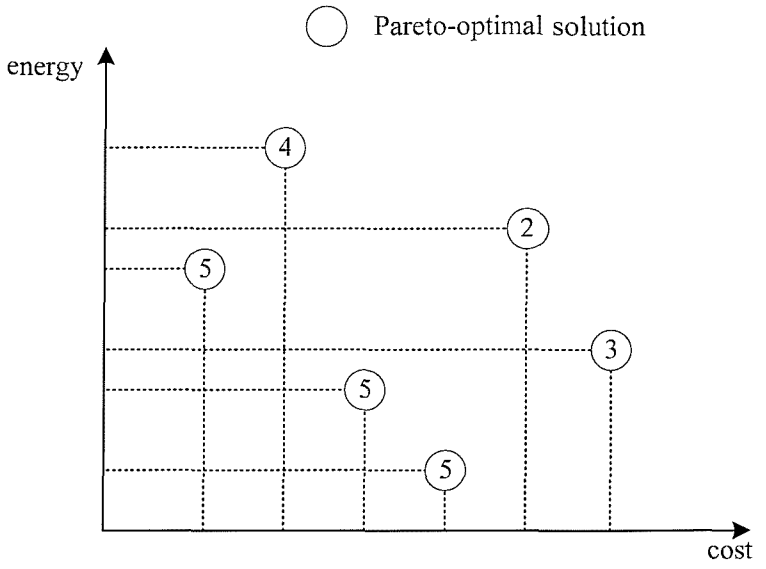


Figure 5.7: Pareto-rank and Pareto-optimal solutions

- **PMS & mapping population evolvement**: the genomes with high Pareto-rank are selected for mating and mutation. The produced offsprings are inserted into the population to replace the genome with low Pareto-rank. In order to improve the optimisation procedure, the PCP aware mapping improvement mutation introduced in Methodology 1 is also employed in Methodology 2.

A record of Pareto-optimal solutions (Pareto set, i.e., designs which are not dominated by any other designs in the population) is kept during the optimisation. For example, in Figure 5.7, the circles with solid filling consist of the Pareto-optimal solutions. The Pareto set are updated after each evolvement, as shown in Figure 5.8. When a new solution α, which is not dominated by the ones already in the Pareto set, is produced in the evolvement, then (1) α is included in the Pareto set; (2) the solutions in Pareto set which are dominated by α are erased. The optimisation procedure continues until the termination condition is met: there has been no Pareto set update for a certain number of generations (10 is found to lead to good results). The evaluation and ranking strategy ensures that the designs are optimised towards timing feasibility and high energy saving at reduced system cost. At the end of the

optimisation, the Pareto set is presented to the designer to give a trade-off between energy saving and system cost.

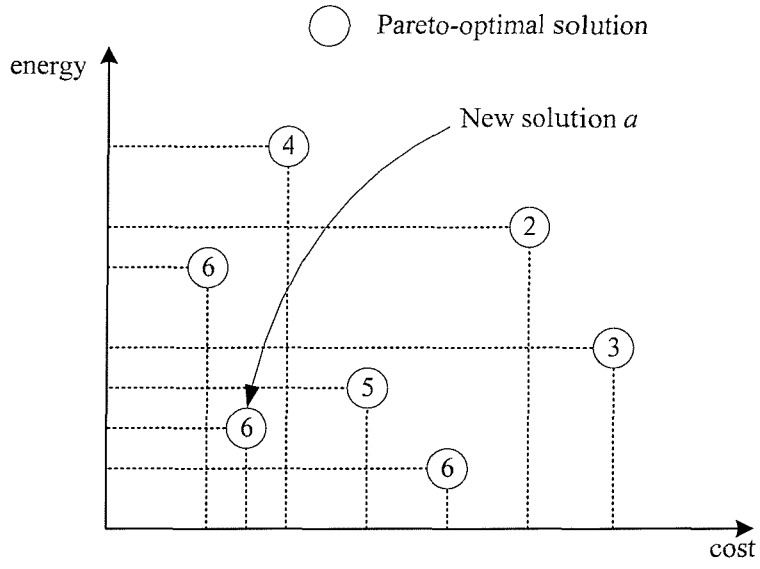


Figure 5.8: Update of Pareto-optimal solutions

5.6 Experimental Results

The proposed power-composition profile (PCP) driven co-synthesis methodologies have been implemented on a Pentium III 866/256MB PC running Linux (Appendix A). In order to evaluate its effectiveness in terms of achieving energy saving at reduced system cost, extensive experiments have been conducted using a number of synthetic examples and a real-life GSM voice CODEC example. The deadline of all the examples is 120% - 125% of the minimum delay. The technology dependent parameters of these PEs are considered to correspond to a 0.07μm CMOS fabrication [115], where the PCPs of the tasks range from 1 to 4. The supply voltage of the DVS and body bias voltage of the ABB can be scaled continuously within 0.5V – 1V and –1.0V – 0V respectively. To indicate the additional cost associated with a PE with power management scheme, it is assumed that PEs with DVS or ABB scheme impose a 10% system cost overhead compared to their non-DVS/non-ABB counterparts. Similarly, the hardware overhead for DVS+ABB PEs is set to 20%. Other overhead cost figures can be chosen without influencing the solution quality. The experiments

are carried out for methodology 1 (exhaustive design space search, Section 5.5.1) and methodology 2 (concurrent PMS and mapping optimisation, Section 5.5.2) respectively.

Experimental results - methodology 1

Two sets of experiments are used to demonstrate that the combination of DVS-only and ABB-only PEs can achieve energy saving comparable to DVS+ABB PEs, but at a reduced system cost. The first set of experiments are conducted on 5 synthetic examples (tg1 – tg5), which contain 25 – 69 nodes, 29 – 84 edges and 2 – 3 PEs; the second set of experiment is conducted on a GSM voice CODEC example.

Using the proposed co-synthesis methodology 1 (Section 5.5.1), the results for a synthetic example tg3 (48 nodes, 60 edges, 3 PEs) are given in Figure 5.9, where the x-axis represents cost factor (relative to the architecture without power management scheme), the y-axis represents the energy dissipation (relative to the energy dissipation at nominal supply voltage and body bias voltage). Each numbered point represents a

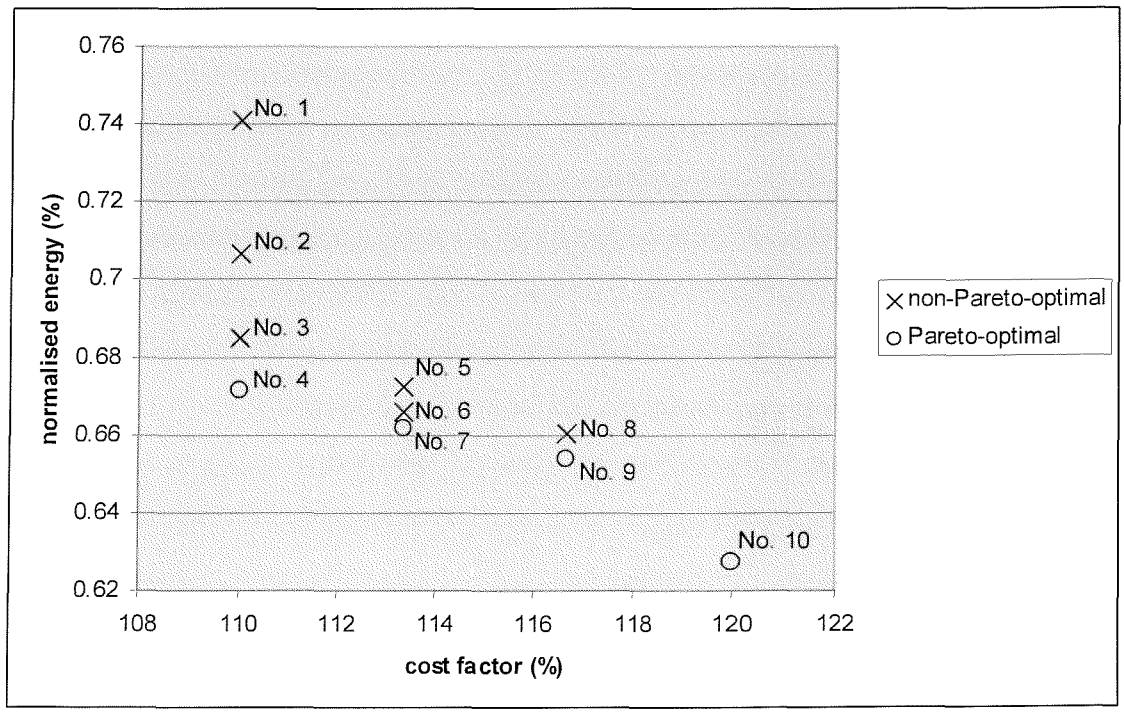


Figure 5.9: Co-synthesis results for the tg3 example

solution, where a 'o' represents a Pareto-optimal solution, and a '×' represent a non-Pareto-optimal solution. For example, consider the solutions with cost factor of 110% (i.e., all PEs have DVS-only or ABB-only capability). Solutions 1 – 3 are dominated by solution 4. Similarly, solutions 5 – 6 are dominated by solution 7; solution 8 is dominated by solution 9. Solutions 4, 7, 9 and 10 are Pareto-optimal solutions. The proposed co-synthesis methodology discards the non-Pareto-optimal solutions and provides the Pareto-optimal solutions to the designer to give a trade-off between energy dissipation and system cost.

Similarly, the co-synthesis methodology is applied to tg1, tg2, tg4 and tg5. The generated Pareto-optimal solutions are given in Table 5.2, where each row presents a Pareto-optimal solution. Columns 2 – 4 show the power management selection (i.e., the numbers of PEs with DVS capability, PEs with ABB capability, and PEs with DVS+ABB capability respectively). Employing the power management selection indicated in columns 2 – 4, columns 5 and 6 give the corresponding cost factor (relative to the architecture without power management scheme) and energy dissipation (relative to the energy dissipation at nominal supply voltage and body bias voltage). As expected, for all the examples, using PEs with DVS+ABB capability achieves the best energy saving. However, by carefully selecting separate DVS or ABB capability for the PEs, comparable energy saving can be achieved with reduced system cost. Consider tg3, in the case when all the three PEs have DVS+ABB capability (1st row), the energy dissipation is 62.7%. The energy dissipation increases to 66.2%, when the three PEs have different voltage scaling capabilities (3rd row). This 3.5% of increase in energy dissipation has been achieved through the reduction of system cost from 120% to 113.3%. Furthermore, as it can be seen, the system cost can be reduced further to 110% (4th row) at the expense of a slight increase in energy dissipation (0.9%). Similarly, the Pareto-optimal solutions for tg1, tg2, tg4 and tg5 also provide trade-offs between energy dissipation and system cost. Based on these results, a designer can choose a suitable selection of voltage scaling capabilities for the PEs according to the design constraints of the application. The execution time of the examples ranges from 43 minutes (tg1) to up to 266 minutes (tg4) due to the exhaustive search nature as outlined in Section 5.5.1.

Example	Pareto-optimal solutions					
	Power management selection			Cost factor (%)	Energy (%)	Computational time (min)
	DVS	ABB	DVS+ABB			
tg1	0	0	2	120	61.5	43
	1	0	1	115	64.3	
	1	1	0	110	69.1	
tg2	0	0	3	120	63.3	92
	0	1	2	116.7	63.5	
	2	0	1	113.3	65.3	
	2	1	0	110	68.2	
tg3	0	0	3	120	62.7	135
	0	1	2	116.7	65.4	
	1	1	1	113.3	66.2	
	2	1	0	110	67.1	
tg4	0	0	3	120	65.0	266
	1	0	2	116.7	66.1	
	1	1	1	113.3	67.8	
	2	1	0	110	69.6	
tg5	0	0	3	120	62.1	153
	1	0	2	116.7	65.4	
	1	1	1	113.3	66.1	
	2	1	0	110	69.4	

Table 5.2: Pareto-optimal solutions of the synthetic examples - Methodology 1

To further validate the proposed co-synthesis methodology 1, it has been applied to a real life GSM voice CODEC [2]. This example has a task graph of 87 nodes and 139 edges, and an architecture containing 3 PEs. Details of the example are given in Section 4.4.1 and Appendix B. Table 5.3 shows the Pareto-optimal solutions and some non-Pareto-optimal solutions generated using the co-synthesis methodology 1. As can be seen, the Pareto-optimal solutions provide a trade-off between energy dissipation and system cost. Using three PEs with DVS+ABB capability (1st row of Pareto-optimal solutions), the system cost is 120% and the energy dissipation is 64.3%, while using three PEs with DVS capability (4th row of Pareto-optimal solutions), the system cost and energy dissipation are 110% and 71.3% respectively. This shows that, in some situations, comparable energy saving can be achieved without using PEs with DVS+ABB capability. However, a careful identification of voltage scaling capability should be proceeded to achieve the desired energy saving. For example, comparing the Pareto-optimal solutions and non-Pareto-optimal solutions with the system cost of

110%, the energy dissipation of using three PEs with ABB capability (3rd row of non-Pareto-optimal solutions) is 78.3%, much higher than that of using three PEs with DVS capability (71.3%). For this example, DVS should be selected for the three PEs. This is because, according to the PCPs, most tasks in this example should preferably be mapped to PEs with DVS capability.

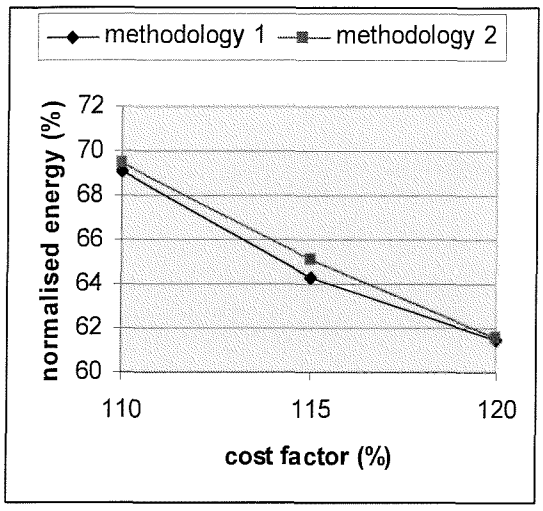
	Power management selection			Cost factor (%)	Energy (%)	Computational time (min)
	DVS	ABB	DVS+ABB			
Pareto-optimal solutions	0	0	3	120	64.3	429
	1	0	2	116.7	67.1	
	1	1	1	113.3	68.2	
	3	0	0	110	71.3	
Non-Pareto-optimal solutions	2	1	0	110	73.9	
	1	2	0	110	75.9	
	0	3	0	110	78.3	

Table 5.3: Co-synthesis results of the CODEC example – methodology 1

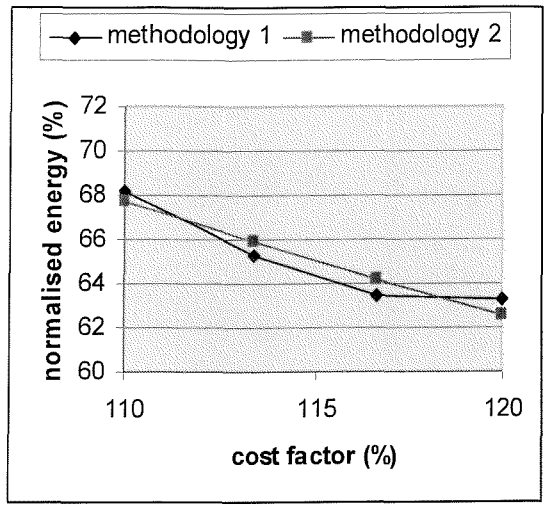
Experimental results - methodology 2

Two sets of experiments are conducted to validate the proposed co-synthesis methodology 2 (Section 5.5.2). Firstly, to show the efficiency of the proposed co-synthesis methodology 2 (concurrent PMS and mapping optimisation) for the power management selection, a comparison is made between the solutions generated by methodology 2 and methodology 1 (exhaustive design space search of all possible power management selection, Section 5.5.1). Figure 5.10 (a) – (f) shows the Pareto-optimal solutions for examples tg1 – tg5, GSM voice CODEC respectively, using methodology 1 and methodology 2. As it can be seen, the Pareto-solutions generated by the two methodologies are almost the same. However, methodology 2 requires significantly less computational time than methodology 1. Methodology 2 requires 22, 37, 115, 127, 72 and 254 minutes to generate the Pareto-optimal solutions for tg1 – tg5, and the GSM voice CODEC example respectively, while methodology 1 requires 43, 92, 135, 266, 153 and 429 respectively (Table 5.2 and Table 5.3). Note that the comparison between methodology 1 and methodology 2 is not given for examples

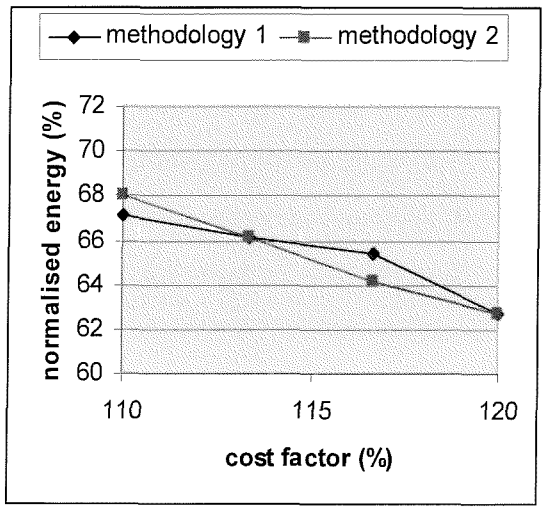
with more PEs, since the increased number of PEs caused unacceptable computational time using the methodology 1.



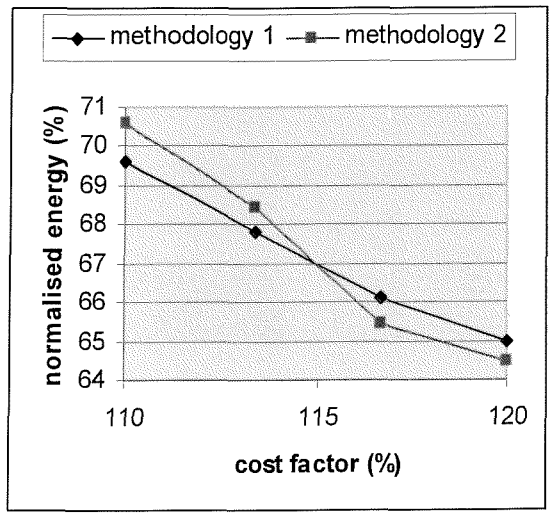
(a) tg1 (35 tasks, 2PEs)



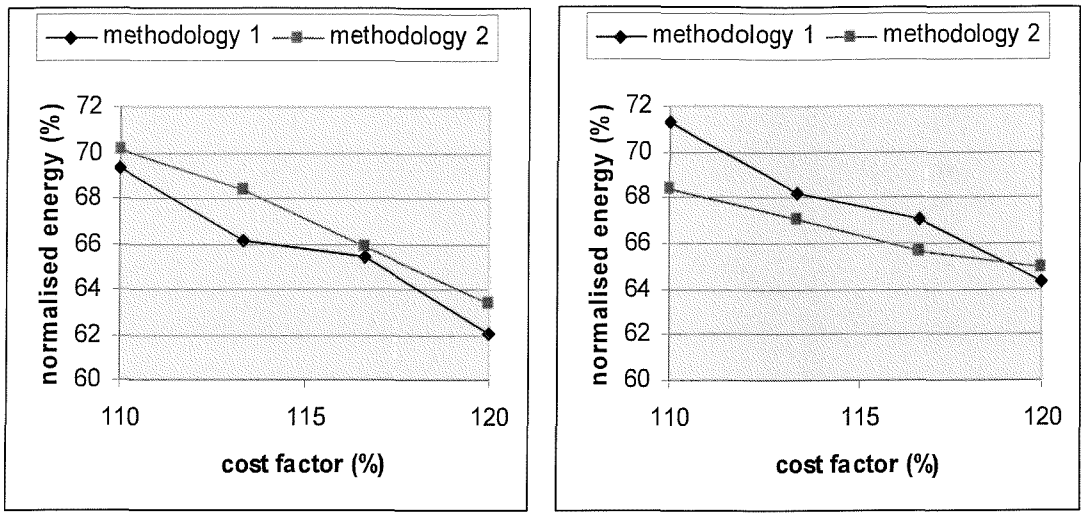
(b) tg2 (35 tasks, 3 PEs)



(c) tg3 (48 tasks, 3 PEs)



(d) tg4 (69 tasks, 3 PEs)

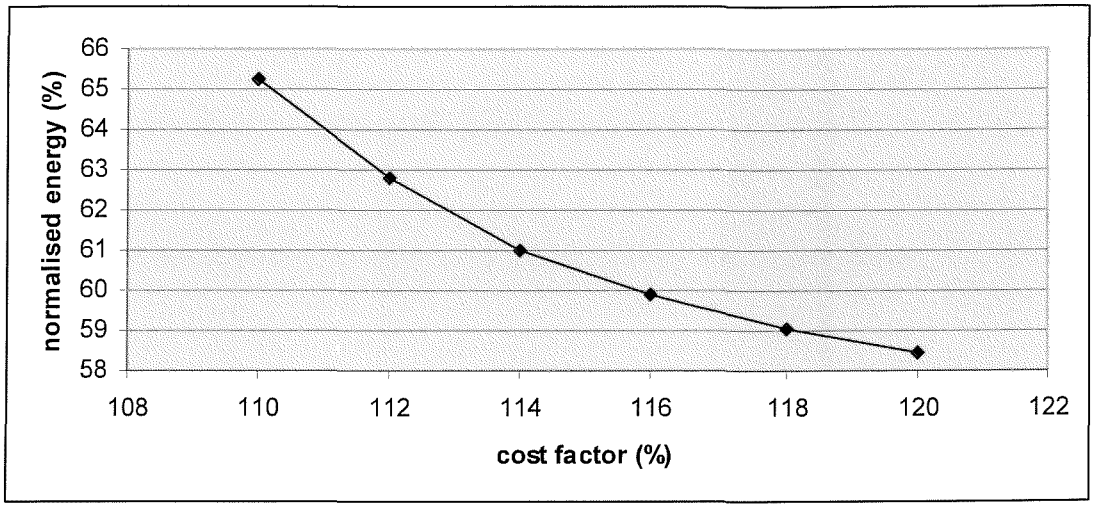


(e) tg5 (59 tasks, 3 PEs) (f) GSM voice CODEC (87 tasks, 3 PEs)

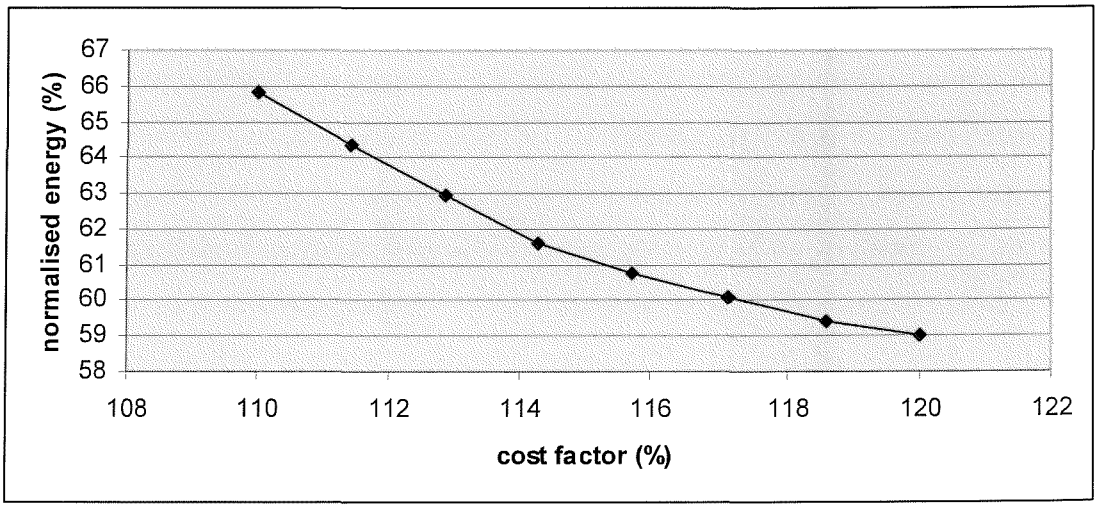
Figure 5.10: Comparison between methodology 1 and methodology 2

In order to show the trade-off between system cost and energy dissipation for systems with more PEs, experiments are carried out on 3 groups (group 1 – 3) of synthetic examples using methodology 2. Each group contains 5 examples, which have the same number of tasks and PEs, but different task execution properties. Figure 5.11 (a) – (c) shows the produced Pareto-optimal solutions for groups 1 – 3, where the x-axis represents cost factor (relative to the an architecture without power management scheme), the y-axis represents the average energy dissipation of the examples within the group (relative to the energy dissipation at nominal supply voltage and body bias voltage). It can be seen that the Pareto-optimal solutions for each group give a trade-off between energy dissipation and system cost. As expected, for all the examples, equipping all PEs with DVS+ABB scheme achieves the best energy saving (rightmost point). However, by carefully selecting separate DVS or ABB capability for the PEs, comparable energy saving can be achieved with reduced system cost. Consider example group 1, in the case when all the 5 PEs have DVS+ABB scheme, the energy dissipation is 58.4%. The energy dissipation is increased to 61.0%, when only 2 PEs have DVS+ABB scheme. This 2.6% of increase in energy has been achieved through the reduction of system cost from 120% to 114%. The system cost can be reduced further to 110% at the expense of a slight increase in

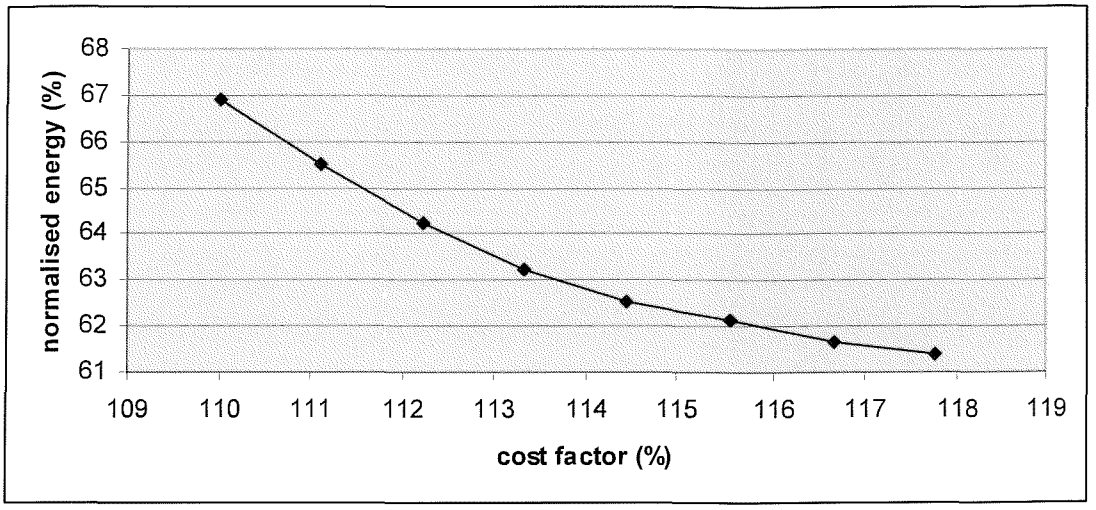
energy (4.3%). The average computational times of the examples in group 1, 2 and 3 are 158, 202 and 259 minutes respectively, while methodology 1 failed to produce a Pareto-optimal solutions for any example in group 1 within 10 hours.



(a) group 1 (38 tasks, 5 PEs)



(b) group 2 (46 tasks, 7 PEs)



(c) group 3 (69 tasks, 9 PEs)

Figure 5.11: Pareto-optimal solutions - Methodology 2

5.7 Concluding Remarks

A new co-synthesis technique aware of the tasks' power-composition profile is presented for dynamic and leakage energy reduction. The presented techniques perform power management selection during the co-synthesis, thus optimise designs not only towards energy reduction but additionally towards system cost reduction. Two co-synthesis methodologies are presented: methodology 1 performs an exhaustive design space search to identify the optimal power management selection; methodology 2 employs a genetic algorithm to optimise the power management selection and task mapping concurrently, leading to shorter computational time compared with methodology 1. The methodologies have been validated using extensive experiments including a real-life GSM voice CODEC example. These experiments have demonstrated that, depending on the power-composition profiles, it is possible to achieve significant energy reduction without the employment of PEs with DVS+ABB scheme, i.e., reduced system cost. The presented co-synthesis methodology is most suitable for designs where the designer has the flexibility to decide which processing element should be equipped with either DVS, ABB, or DVS+ABB scheme.

Chapter 6

Conclusion

According to ITRS'2003 [146] the request for embedded computing systems with low power consumption, as well as their complexity, will continue to increase. Computation intensive applications such as multimedia with enhanced audio and video coding and decoding techniques are needed in hand-held devices. Designing energy-efficient embedded computing systems for such applications is a challenging and difficult task. This is because the limited energy resource available and the increased functionality requirement. Furthermore, short design cycle and low cost constraint are essential for the success of these products. The work presented in this thesis has focused on minimising power consumption (dynamic and leakage) of embedded systems at system level of the design flow. To achieve energy-efficient designs, three novel co-synthesis techniques capable of reducing dynamic and leakage power have been developed to support the design of such embedded systems. In particular three key issues have been addressed:

(1) Dynamic voltage scaling (DVS) was investigated in the context of data and control dominated embedded systems. For this purpose, a new conditional behaviour aware DVS technique has been proposed. A new mapping technique has also been developed specifically for an effective utilisation of the conditional behaviour aware DVS, by increasing available slack time.

(2) Impact of communication on the co-synthesis was investigated. A communication-integrated co-synthesis technique has been presented to perform mapping, scheduling and dynamic voltage scaling for embedded systems taking into account the energy and time cost of communications.

(3) A new power-composition profile (PCP) aware co-synthesis technique capable of reducing both dynamic power and leakage power was proposed. The PCP aware co-synthesis enables an efficient power management scheme

selection so that significant energy saving can be achieved without extra system cost.

The following Section 6.1 summarise the main contributions made by the presented work, and Section 6.2 outlines some relevant areas for future research.

6.1 Summary and Research Contributions

Energy management techniques including dynamic voltage scaling (DVS) and adaptive body biasing (ABB), which are capable of trading off energy against performance at run-time by means of changing the processing element's (PE) operational state in terms of supply voltage (DVS) or threshold voltage (ABB), as well as clock frequency, have recently become viable in practice. The work presented in this thesis has focused on the energy minimisation of distributed embedded systems that contains PEs with DVS and/or ABB capabilities.

Chapter 3 has introduced a new conditional behaviour aware DVS technique for data and control dominated embedded systems expressed as conditional task graphs (CTGs). This technique exploits system slack time taking into account the conditional behaviour of the application, in order to reduce the system energy dissipation. The proposed DVS technique starts with a more conservative scaling factor at the beginning of the scheduling process, and incrementally modifies the scaling factor when more slack time is identified, thus achieves the highest possible energy saving and at the same time guarantees the timing feasibility for all possible condition values. To further reduce the energy dissipation of data and control flow dominated embedded systems, a mapping technique has been developed specifically to effectively exploit the energy reduction capability of the proposed conditional behaviour aware DVS technique. This is achieved through a mapping optimisation based on a genetic algorithm, which carefully balances the design between energy minimisation and timing feasibility. The mapping optimisation is guided by an energy estimation based on the proposed DVS technique. Extensive experiments have been conducted including a real-life example of a vehicle cruise controller. These experiments have

shown that up to 35% energy saving can be achieved by using the proposed conditional behaviour aware DVS technique. Performing the proposed mapping optimisation reduces the energy dissipation further. The experimental results also shows that the proposed mapping optimisation achieves significantly better energy saving, compared to a previously reported mapping algorithm, and this energy saving can be achieved within a reasonable amount of computational time.

In Chapter 4, the capability of the co-synthesis technique proposed in Chapter 3 has been extended, in order to address the impact of the communications on embedded systems design. The concept of enhanced system model, which captures the time and energy cost of the communications, has been introduced for this purpose. In the extended co-synthesis technique, task mapping and communication mapping have been decoupled to avoid infeasible mappings. As a result, the search space of task mapping is restricted to feasible mapping solutions only, reducing the computational time while maintaining the full possible optimisation potential. Furthermore, the conditional behaviour aware DVS technique of Chapter 3 has been extended to address the enhanced system model with no penalty in the utilisation of slack time. Extensive experiments have been conducted including a real-life GSM voice CODEC example. These experiments have shown that the extended co-synthesis technique can achieve significant energy reduction after considering the impact of communication. It is also shown that an appropriate selection of communication architecture is essential in order to achieve energy efficiency.

Chapters 3 and 4 have focused on the reduction of dynamic power, which has traditionally been the primary source of power consumption for $> 0.1\mu m$ CMOS technology. Chapter 5 considers the trend of increasing leakage power in deep sub-micron technology, which justifies the employment of adaptive body biasing (ABB) to reduce leakage power. A new co-synthesis technique has been introduced in Chapter 5 employing DVS and ABB to reduce the dynamic power as well as leakage power. A key feature of the proposed co-synthesis technique is its awareness of the tasks' power composition profile (PCP). By taking advantage of the PCP information, the proposed co-synthesis technique performs a power management selection (PMS) for the PEs,

i.e., it decides upon which PE to be equipped with which power management scheme (DVS, ABB, or combined DVS and ABB). As a result, the design is optimised not only towards energy saving but also towards system cost reduction. In addition, the proposed co-synthesis performs mapping, scheduling and voltage scaling (DVS and/or ABB). Two methodologies have been developed. Methodology 1 performs an exhaustive design space search to identify the optimal power management selection. Methodology 2 employs a genetic algorithm to optimise the power management selection and task mapping concurrently, leading to shorter computational time compared with methodology 1. A new improvement strategy has been developed to aid the genetic algorithm evolvement by carefully pushing the optimisation into promising search space. The proposed PCP aware co-synthesis technique has been validated using a large number of experiments, which show that, depending on the PCP, it is possible to achieve significant energy reduction without the employment of PEs with combined DVS and ABB capability, i.e., reduced system cost. The presented approach is most suitable for system-on-chip (SoC) designs where the designer has the flexibility to decide which processing element should be equipped with either DVS, ABB, or combined DVS and ABB capability.

In conclusion, the main aim of this thesis was the development of co-synthesis techniques for energy-efficient distributed embedded system. A detailed investigation into the co-synthesis of data and control dominated embedded systems has shown that significant energy reduction can be achieved through dynamic voltage scaling taking into account of the conditional behaviour of the application, as well as through an appropriate mapping of the application onto the target architecture. Furthermore, it was shown that the consideration of communication and a suitable selection of communication architecture during the co-synthesis are essential for energy-efficient designs. Finally, considering the trend of increasing leakage power in deep sub-micron technology, it was shown that the power composition profile aware co-synthesis enables an efficient power management scheme selection so that significant dynamic and leakage energy reduction can be achieved without extra system cost. In essence, system-level co-synthesis techniques that consider energy management, as

presented in this thesis, should be given serious consideration when designing low power embedded computing systems.

6.2 Future Research Directions

During the course of this work, a number of relevant challenging research topics have been identified. In the following a short introduction of two interesting topics for future research is given.

6.2.1 Online Voltage Scheduling of CTG

The conditional behaviour aware dynamic voltage scaling (DVS) technique presented in this work (Chapter 3) produces a schedule table for conditional task graphs (CTGs). The DVS technique is offline, i.e., the voltage settings are calculated before application run-time, assuming that the actual execution time (AET) of a task under nominal supply voltage always equals a given worst case execution time (WCET). However, depending on the input data, the AET of a task is usually shorter than the WCET. The difference between WCET and AET results in slack time that can be exploited by DVS. Because the amount of the extra slack time is unknown until run-time, offline DVS is incapable of predicting and utilising the extra slack time. It would be interesting to develop an online DVS, which identifies the extra slack time and accordingly re-calculates the voltage settings at the run-time of applications, aiming to further energy reduction. Since the online DVS is performed at run-time, its algorithm complexity should be low enough to avoid unacceptable time overhead. The benefit of online DVS has been illustrated by the recent work [147]. Useful work need to be conducted further to exploit the online slack time with low time overhead.

6.2.2 Co-Synthesis of Network-on-Chip

The importance of communication in embedded system design has been shown in Chapter 4. A key element in system-on-chip (SoC) design is the on-chip communication that interconnects the SoC cores. SoCs complexity is increasing with

the continuing scaling down of CMOS technology. According to ITRS'2003, an average SoC will contain more than 50 cores in 2008 and 100 cores in 2012. One promising solution to providing scalable, energy-efficient and reliable communication for such SoCs is a new interconnection design methodology – Network-on-Chip (NoC). The NoC model, which is drawn from knowledge developed by the networking and parallel computing communities, is to view the SoC as a micro-network of components. In NoCs, cores communicate with each other by sending packets over the network. This facilitates modular and structured design with increased bandwidth.

NoCs differ from wide-area network because they have less area resources but more wire resources, and because they are more application-specific and less non-deterministic. The design experience in wide-area networks does not necessarily apply to NoCs. New co-synthesis techniques are needed to determine the network architecture, the flow control and the message routing algorithm for NoC-based designs. Another distinctive characteristic of NoCs is the energy constraint. Whereas computation and storage energy greatly benefits from device scaling, the global communication in SoCs will consume increasingly higher energy. Hence, communication energy minimisation is a necessary concern in NoC design. Work is needed to be done in all layers of the communication protocol stacks (physical layer, data link layer, network layer, transport layer, and application layer) to achieve a balanced trade-off between communication energy and network performance. Also, the trend of lower supply voltage, increased interconnect density and faster clock rates exposes on-chip communication to increased transient failures caused by capacitive and inductive crosstalk, power supply noise, etc. Traditional fault tolerant algorithms for wide-area networks are infeasible in the NoC domain due to the limited on chip area resource. New techniques must be developed if fault tolerant NoCs are to become possible. The fault tolerant algorithms for NoCs must take the energy constraint into account.

Appendix A

Simulation Set-up

This appendix describes the simulation set-up used to generate the experimental results in Chapters 3, 4 and 5. The appendix is organised as follows. Section A.1 briefly introduces the development environment of the simulation tools. Sections A.2 and A.3 introduce the implementation of graph and genetic algorithm respectively. Details of the simulation flow are illustrated in Section A.4. Section A.5 explains the organisation of the input files to the simulation flow.

A.1 Development Environment

The simulation tool for the experiments of this work has been developed using the C++ programming language in a Pentium/Linux PC. The development flow is illustrated in Figure A.1, where the input to the GCC C++ compiler is user C++ code (source code), and the output is simulation tool (executable code). Three C++ libraries are employed in the development flow, besides the GNU Standard C++ Library provided by the C++ compiler, the other two libraries are LEDA and GAlib, which are for the implementation of graph and genetic algorithm respectively, as introduced next.

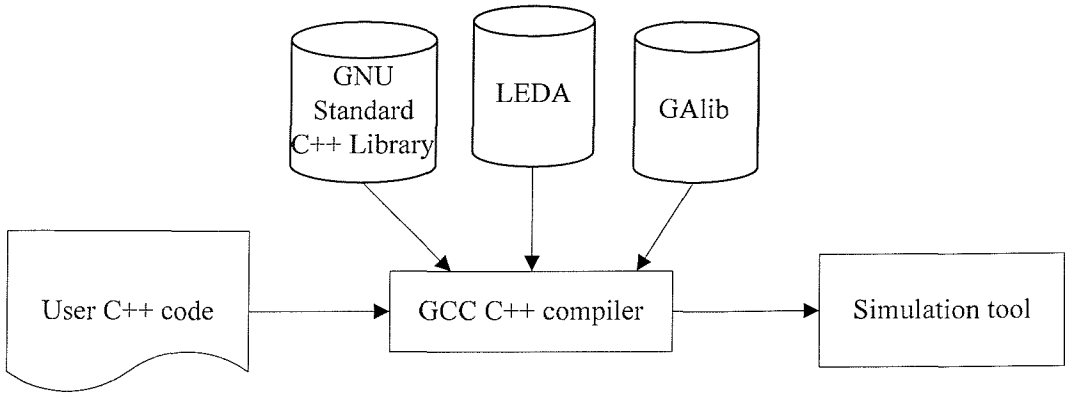


Figure A.1: Simulation tool development flow

A.2 Implementation of Graph

Since the system specification of this work is task graph and conditional task graph, the implementation of graph abstraction and graph algorithm is an important element influencing the efficiency of the simulation tool. In this work, task graph and conditional task graph have been implemented using the data types *graph* and *GRAPH* provided by LEDA [148]. LEDA is a C++ library containing efficient implementations of all graph data types and algorithms, as well as many abstract data types (e.g., sets, queues, lists). An instance G of the data type *graph* consists of a list V of the nodes and a list E of edges. Two lists of edges are associated with every node v: $out\_edges(v) = \{e \in E \mid v=source(e)\}$, i.e., the list of edges starting from v, and $in\_edges(v) = \{e \in E \mid v=target(e)\}$, i.e., the list of edges ending in v. While an instance G of the data type *GRAPH* (parameterised graph) is a graph whose nodes and edges contain additional data. Every node contains an element of a data type *vtype*, and every edge contains an element of a data type *etype*. Data types *vtype* and *etype* are defined by user to store information associated with individual nodes and edges. For example, in this work, the data type *vtype* contains the node id number, node type (disjunction node, conjunction node, or normal node), and the data type *etype* contains the edge id number, edge type (conditional edge or normal edge), conditional value, etc. Figure A.2 illustrates the implementation of a task graph. Note the $in\_edges(n_1)$ is empty because n_1 is the source node. Programming with LEDA has dramatically decreased development time.

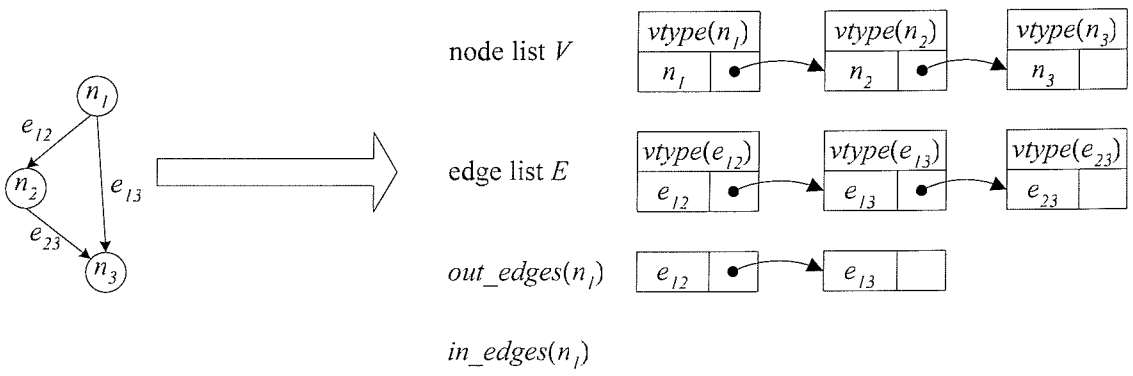


Figure A.2: Implementation of task graph

A.3 Implementation of Genetic Algorithm

The implementation of genetic algorithm (Chapter 3, Section 3.1.3) is another important factor in the efficiency of the simulation tool, which is carried out using GAlib [149]. GAlib is a C++ library of genetic algorithm components. It contains a set of C++ genetic algorithm objects and includes tools for using genetic algorithm to do optimisation. The applying of GAlib mainly involves two classes: a genome and a genetic algorithm. Each genome instance represents a single solution to a specific problem. The genetic algorithm instance uses a user-defined objective function to determine the quality of each genome. It uses the genome operators (mutation and crossover) to generate new individuals. There are three steps need to be done to solve an optimisation problem using GAlib: (1) to define a representation for the problem solution (e.g., Chapter 3, Figure 3.17); (2) to define the genetic operator (e.g., Chapter 5.5, Section 5.5.1); (3) to define the objective functions (e.g., Chapter 3, Equation 3.11). GAlib provides many built-in representations and genetic operators, which, in many cases, can be used with little or no modification. Appropriate representation, genetic algorithm type, genetic operators, as well as the parameters which decide how genetic algorithm behaves, are quite important for achieving an efficient genetic algorithm, which are described as follows. For the representation used in this work, the reader is referred to Chapter 3, Section 3.4 and Chapter 5, section 5.5.2. The genetic algorithm employed in this work is of steady state type, i.e., not all of the individuals in the solution pool are replaced by the new individuals. The generation gap is set to 70%. The crossover is carried out by a random two-point crossover. The genetic algorithm parameters are set as follows: the population size is 25, the crossover probability is 0.7, and the mutation probability is 0.03. Experimental results of Chapters 3, 4 and 5 have proved the efficiency of these parameters for the specific problems of this work.

A.4 Simulation Flow

This section describes the simulation flow employed in the DVS-based co-synthesis (Chapters 3 and 4), and the concurrent power management selection (PMS) and mapping optimisation (Chapter 5).

DVS-Based Co-Synthesis

Using the development environment illustrated in Figure A.1, a simulation tool has been developed to produce the experimental results of Chapters 3 and 4. The flow of the simulation tool is given in Figure A.3. As it can be seen, the input to the simulation tool is three files (Section A.5):

(1) a graph file that gives a description of the conditional task graph;

(2) an architecture file that describes the number and type of processing elements (PEs) in the architecture, as well as the communication links (CLs) interconnecting the PEs;

(3) a technology library file which gives the parameters of the PEs and CLs, and the execution properties of the tasks.

The output is three files giving the co-synthesis solution: a mapping file, a schedule table file, and a statistics file which includes the information of the solution (e.g., energy dissipation of the application before and after applying DVS, energy reduction achieved by applying DVS). As shown in Figure A.3, the simulation tool consists of four processing objects (solid fill) and seven data objects (no fill), which are listed in Table A.1. Firstly, the simulation tool reads the three input files and initiates objects GRAPH, ARCHI and LIBRARY. Secondly, based on the information provided by objects GRAPH, ARCHI and LIBRARY, object GA_MAP performs mapping optimisation based on a genetic algorithm. For each mapping candidate produced in the process of mapping optimisation, object GA_MAP calls object SCHEDULE to perform communication mapping and activity scheduling, and calls object DVS to perform voltage scaling, the produced co-synthesis solution are stored into objects

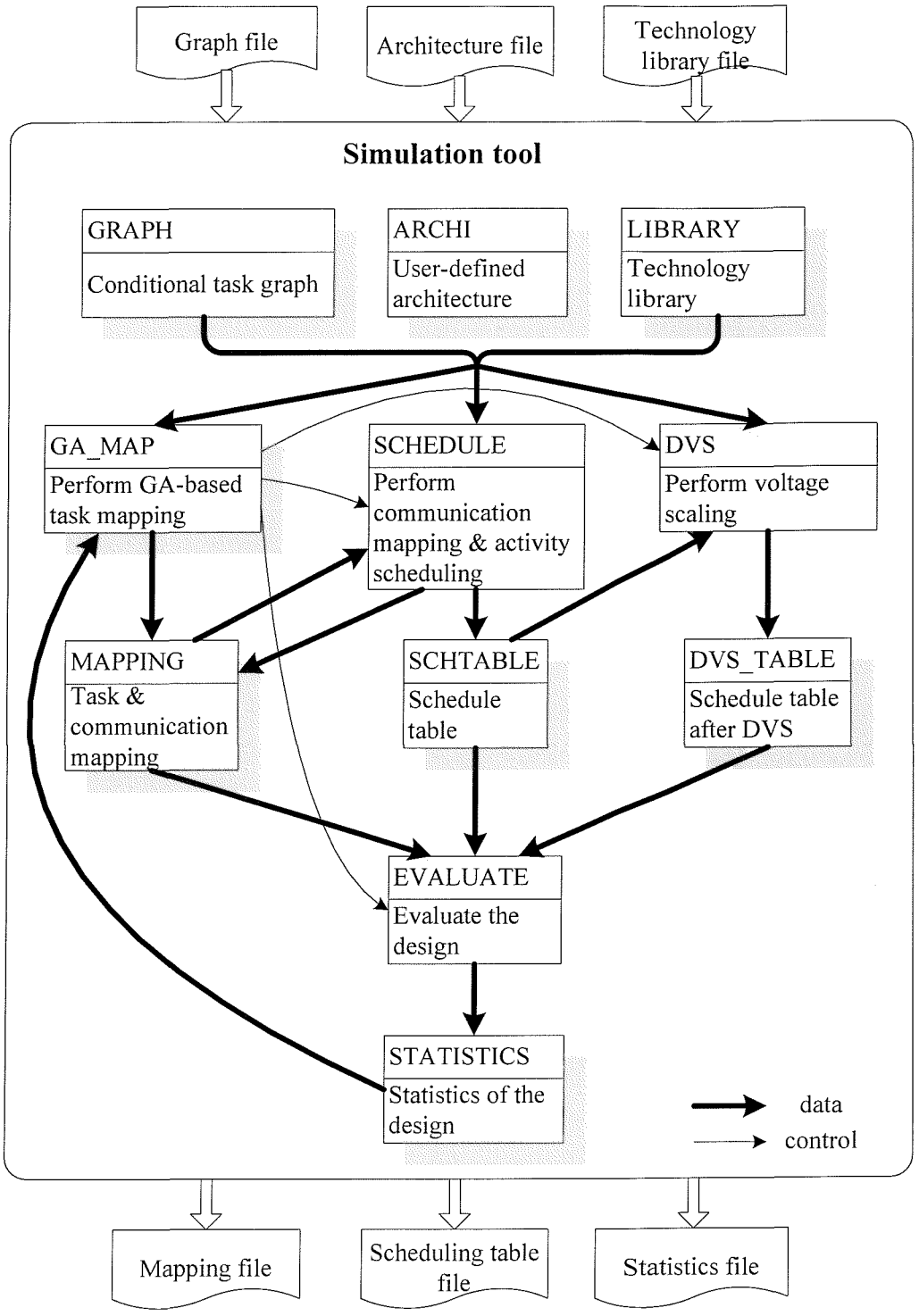


Figure A.3: Simulation flow of DVS-based co-synthesis

MAPPING, SCHTABLE and DVS_TABLE. Then object GA_MAP calls object EVALUATE to evaluate the co-synthesis solution in terms of area, timing and energy

dissipation. Object GA_MAP continues the mapping optimisation until a termination condition is met. Finally, the co-synthesis solution and its statistics are written into the output files. For details of the algorithm used in objects GA_MAP, SCHEDULE, DVS and EVALUATE, the reader is referred to Chapter 3, Sections 3.3 and 3.4, and Chapter 4, Section 4.3. The termination condition of object GA_MAP is: the improvement achieved over ten continuous iterations is no more than 1%. The selection of parameters used in object GA_MAP has a significant impact on the optimisation efficiency, which has been given in Section A.3.

Object	Description
GRAPH	nodes, edges and conditional behaviour of the conditional task graph
ARCHI	number and type of the PEs and CLs, and the interconnection topology
LIBRARY	parameters of the PEs, CLs, tasks and communications
GA_MAP	for a given simulation input (input files), performing genetic algorithm based task mapping
SCHEDULE	for a given simulation input and a task mapping, performing communication mapping & activity scheduling, produce a schedule table
DVS	for a given simulation input and a schedule table, applying DVS to the schedule table
MAPPING	mapping of tasks to PEs, and mapping of communications to CLs
SCHTABLE	schedule table (Chapter 3, Section 3.1.2) before applying DVS
DVS_TABLE	schedule table after applying DVS
EVALUATE	evaluate the co-synthesis solution in terms of area, timing and energy
STATISTICS	information concerning the quality of the co-synthesis solution

Table A.1: Description of the objects within the simulation tool

Concurrent PMS and Task Optimisation

The simulation flow employed in the concurrent power management selection (PMS) and mapping optimisation is similar to Figure A.3, apart from some details due to the consideration of leakage power and PMS, which are given next.

- Object LIBRARY includes additional information concerning leakage power.
- Object LIBRARY does not specify the power management scheme of the PEs, which is to be decided by object GA_MAP.
- Object GA_MAP performs concurrent PMS and task mapping optimisation.

- Object DVS scales supply voltage and body bias voltage simultaneously to reduce both dynamic power and leakage power.
- Object EVALUATE evaluates the co-synthesis solutions according to their Pareto-rank (Chapter 5, Figure 5.7).
- The termination condition of object GA_MAP is: there has been no Pareto set update for 10 iterations.

For details of the algorithms used in the concurrent PMS and mapping optimisation, the reader is referred to Chapter 5, Section 5.5.2.

A.5 Input Files

This section explained the organisation of the three input files to the simulation flow: graph file, architecture file and technology file. Figure A.4 shows parts of an example graph file, where part 1 lists the nodes of the tasks, part 2 describes the edges between the nodes, part 3 lists the conjunction nodes of the conditional task graph, and part 4 describes the conditional value of the edges. For example, the conditional task graph described in Figure A.4 has 35 nodes and 41 edges where edge 0 starts from node 0 and ends at node 1. There are two conjunction nodes: nodes 6 and 12. Edges 4, 5, 17, 18 are conditional edges, which are associated with condition value 2, -2, 1, -1 respectively.

```
=========================================
//part 1: nodes of the graph
   (NODES NODE0)
   (NODES NODE1)
    ......
   (NODES NODE33)
   (NODES NODE34)
=========================================
//part 2: edges of the graph
   (EDGES EDGE0 (Connect NODE0  NODE1))
   (EDGES EDGE1 (Connect NODE3  NODE5))
    ......
   (EDGES EDGE39 (Connect NODE31  NODE33))
   (EDGES EDGE40 (Connect NODE33  NODE34))
=========================================
//part 3: conjunction nodes
   (JOINNODE NODE6 NODE12)
=========================================
```

```
//part 4: conditional value of the edges
   (CONDITION 2 EDGE4)
   (CONDITION -2 EDGE5)
   (CONDITION 1 EDGE17)
   (CONDITION -1 EDGE18)
=========================================
```

Figure A.4: File description of a conditional task graph

Figure A.5 shows an example architecture file, where part 1 and 2 respectively list the PEs and CLs in the architecture and their type, part 3 gives the interconnection topology. For example, the architecture described in Figure A.5 contains three PEs: PE0 of type 0, PE1 of type 0 and PE2 of type 1. There are two CLs, CL0 and CL1, which are of type 0 and type 1 respectively. The "conn" property of CL0 is $7=2^0+2^1+2^2$, which means CL0 links PE0, PE1 and PE2, while the "conn" property of CL1 is $6= 2^1+2^2$, which means CL1 links PE1 and PE2.

```
=========================================
//part 1: PEs in the architecture
  {PE_ALLO
   (PE 0 type 0)
   (PE 1 type 0)
   (PE 2 type 1)
  }
=========================================
//part 2: CLs in the architecture
  {LINK_ALLO
   (CL 0 type 0)
   (CL 1 type 1)
  }
=========================================
//part 3: interconnection topology
  {LINK_CONN
   (CL 0 conn 7)
   (CL 1 conn 6)
  }
=========================================
```

Figure A.5: File description of an architecture

Figure A.6 shows an example technology library file, where part 1 gives the parameters of the PEs and tasks, part 2 gives the parameters of the links, and part 3 gives the data amount of the communication on the edges. A brief description of all parameters is given in Table A.2. The technology file shown in Figure A.6 is for the concurrent PMS and mapping optimisation (Chapter 5). The technology file for the

DVS-based co-synthesis (Chapters 3 and 4) is very similar, except that there is no parameter concerning the leakage power and ABB.

Parameter	Description
pe_type	the type of the PE
dvs	whether the PE is DVS-enabled
abb	whether the PE is ABB-enabled
freq (PE)	the nominal frequency of the PE
vmax	the nominal supply voltage
vmin	the minimum supply voltage
vbsmax	the nominal body bias voltage
vbsmin	the minimum body bias voltage
k1, k2, k3, k4, k5, vth1, ceff, lg	technology constants (Chapter 5, Section5.2)
execyc	cycle number needed for execution
dynpwr	dynamic power
leakpwr	leakage power
stpwr (CL)	static power of CL
freq (CL)	frequency of CL
dynpwr (CL)	dynamic power of CL
amount	data amount of communication

Table A.2: Parameter description of the technology library file

```
=========================================
//part 1: parameters of PEs and tasks
{PE 0
(pe_type GPP)(dvs 1)(abb 1)
(freq 4.21e+09)(vmax 1)(vmin 0.5)(vbsmin -1)
(k1 0.063)(k2 0.153)(k3 1.87407e-07)(k4 1.83)(k5 4.19)
(vth1 0.244)(ceff 1.11e-09)(lg 4e+06)(vbsmax 0)
(task 0)(execyc 1.22e08)(dynpwr 12.46)(leakpwr 4.67)
(task 1)(execyc 7.998e07)(dynpwr 7.78)(leakpwr 4.67)
      ......
(task 33)(execyc 1.85e08)(dynpwr 4.67)(leakpwr 4.67)
(task 34)(execyc 1.51e08)(dynpwr 4.67)(leakpwr 4.67)
}
      ......
=========================================
//part 2: parameters of CLs
{CL 0
(stpwr 0)(freq 3e09)(dynpwr 1)
}
      ......
=========================================
//part 3: communication amount
{COM_AMOUNT
(com 0)(amount 12)
(com 1)(amount 100)
      ......
(com 39)(amount 6)
```

```
(com 40)(amount 30)
}
==========================================
```

Figure A.6: File description of an technology library

Appendix B

GSM Voice CODEC Benchmark

A GSM voice CODEC example [2, 143] was used in the experiments of Chapter 4 and Chapter 5. This appendix details the system specification of the GSM voice CODEC example.

B.1 Conditional Task Graph of the GSM Voice CODEC

GSM voice CODEC is a speech compression/decompression algorithm used in Global System for Mobile Telecommunication (GSM). GSM voice CODEC consists of two parts: encoder and decoder. The task graphs of the encoder and decoder [2] have been derived from the source code of the CODEC [143] (Figure 4.16 and Figure 4.17). The task graphs of encoder and decoder are combined to form a conditional task graph (CTG). The following listing describes the conditional task graph of the GSM voice CODEC, which is used as the input to the experiments of Chapter 4 and Chapter 5. In the listing, PRs 1-53 correspond to the tasks 0-52 in the task graph of Figure 4.16, PRs 54-87 correspond to the tasks 0-33 in the task graph of Figure 4.17.

```
=========================================
// CTG file for GSM voice CODEC
=========================================
//part 1: nodes
//encoder tasks
   (PROCESS PR1)
   (PROCESS PR2)
   (PROCESS PR3)
   (PROCESS PR4)
   (PROCESS PR5)
   (PROCESS PR6)
   (PROCESS PR7)
   (PROCESS PR8)
   (PROCESS PR9)
   (PROCESS PR10)
   (PROCESS PR11)
```

```
    (PROCESS PR12)
    (PROCESS PR13)
   (PROCESS PR14)
    (PROCESS PR15)
    (PROCESS PR16)
    (PROCESS PR17)
    (PROCESS PR18)
    (PROCESS PR19)
    (PROCESS PR20)
    (PROCESS PR21)
    (PROCESS PR22)
    (PROCESS PR23)
    (PROCESS PR24)
    (PROCESS PR25)
    (PROCESS PR26)
    (PROCESS PR27)
    (PROCESS PR28)
    (PROCESS PR29)
    (PROCESS PR30)
    (PROCESS PR31)
    (PROCESS PR32)
    (PROCESS PR33)
    (PROCESS PR34)
    (PROCESS PR35)
    (PROCESS PR36)
    (PROCESS PR37)
    (PROCESS PR38)
    (PROCESS PR39)
    (PROCESS PR40)
    (PROCESS PR41)
    (PROCESS PR42)
    (PROCESS PR43)
    (PROCESS PR44)
    (PROCESS PR45)
    (PROCESS PR46)
    (PROCESS PR47)
    (PROCESS PR48)
    (PROCESS PR49)
    (PROCESS PR50)
    (PROCESS PR51)
    (PROCESS PR52)
    (PROCESS PR53)
//decoder tasks
    (PROCESS PR54)
    (PROCESS PR55)
    (PROCESS PR56)
    (PROCESS PR57)
    (PROCESS PR58)
    (PROCESS PR59)
    (PROCESS PR60)
    (PROCESS PR61)
    (PROCESS PR62)
    (PROCESS PR63)
    (PROCESS PR64)
    (PROCESS PR65)
    (PROCESS PR66)
    (PROCESS PR67)
```

```
    (PROCESS PR68)
    (PROCESS PR69)
    (PROCESS PR70)
    (PROCESS PR71)
    (PROCESS PR72)
    (PROCESS PR73)
    (PROCESS PR74)
    (PROCESS PR75)
    (PROCESS PR76)
    (PROCESS PR77)
    (PROCESS PR78)
    (PROCESS PR79)
    (PROCESS PR80)
    (PROCESS PR81)
    (PROCESS PR82)
    (PROCESS PR83)
    (PROCESS PR84)
    (PROCESS PR85)
    (PROCESS PR86)
    (PROCESS PR87)
=========================================
//part 1: edges
//encoder communications
   (ARCS ARC1 (Connect PR1  PR2 )
   (ARCS ARC2 (Connect PR1  PR8 )
   (ARCS ARC3 (Connect PR1  PR11 )
   (ARCS ARC4 (Connect PR1  PR14 )
   (ARCS ARC5 (Connect PR1  PR17 )
   (ARCS ARC6 (Connect PR1  PR10 )
   (ARCS ARC7 (Connect PR1  PR13 )
   (ARCS ARC8 (Connect PR1  PR16 )
   (ARCS ARC9 (Connect PR1  PR19 )
   (ARCS ARC10 (Connect PR2  PR3 )
   (ARCS ARC11 (Connect PR2  PR10 )
   (ARCS ARC12 (Connect PR2  PR21 )
   (ARCS ARC13 (Connect PR2  PR53 )
   (ARCS ARC14 (Connect PR3  PR4 )
   (ARCS ARC15 (Connect PR4  PR5 )
   (ARCS ARC16 (Connect PR5  PR6 )
   (ARCS ARC17 (Connect PR6  PR7 )
   (ARCS ARC18 (Connect PR7  PR8 )
   (ARCS ARC19 (Connect PR7  PR11 )
   (ARCS ARC20 (Connect PR7  PR14 )
   (ARCS ARC21 (Connect PR7  PR17 )
   (ARCS ARC22 (Connect PR8  PR9 )
   (ARCS ARC23 (Connect PR9  PR10 )
   (ARCS ARC24 (Connect PR10  PR13 )
   (ARCS ARC25 (Connect PR10  PR20 )
   (ARCS ARC26 (Connect PR11  PR12 )
   (ARCS ARC27 (Connect PR12  PR13 )
   (ARCS ARC28 (Connect PR13  PR16 )
   (ARCS ARC29 (Connect PR13  PR20 )
   (ARCS ARC30 (Connect PR14  PR15 )
   (ARCS ARC31 (Connect PR15  PR16 )
   (ARCS ARC32 (Connect PR16  PR19 )
   (ARCS ARC33 (Connect PR16  PR20 )
   (ARCS ARC34 (Connect PR17  PR18 )
```

```
   (ARCS ARC35 (Connect PR18  PR19 )
   (ARCS ARC36 (Connect PR19  PR20 )
   (ARCS ARC37 (Connect PR19  PR53 )
   (ARCS ARC38 (Connect PR20  PR21 )
   (ARCS ARC39 (Connect PR20  PR29 )
   (ARCS ARC40 (Connect PR20  PR37 )
   (ARCS ARC41 (Connect PR20  PR45 )
   (ARCS ARC42 (Connect PR21  PR22 )
   (ARCS ARC43 (Connect PR22  PR23 )
   (ARCS ARC44 (Connect PR22  PR28 )
   (ARCS ARC45 (Connect PR23  PR24 )
   (ARCS ARC46 (Connect PR24  PR25 )
   (ARCS ARC47 (Connect PR24  PR27 )
   (ARCS ARC48 (Connect PR25  PR26 )
   (ARCS ARC49 (Connect PR26  PR27 )
   (ARCS ARC50 (Connect PR27  PR28 )
   (ARCS ARC51 (Connect PR28  PR29 )
   (ARCS ARC52 (Connect PR29  PR30 )
   (ARCS ARC53 (Connect PR30  PR31 )
   (ARCS ARC54 (Connect PR30  PR36 )
   (ARCS ARC55 (Connect PR31  PR32 )
   (ARCS ARC56 (Connect PR32  PR33 )
   (ARCS ARC57 (Connect PR32  PR35 )
   (ARCS ARC58 (Connect PR33  PR34 )
   (ARCS ARC59 (Connect PR34  PR35 )
   (ARCS ARC60 (Connect PR35  PR36 )
   (ARCS ARC61 (Connect PR36  PR37 )
   (ARCS ARC62 (Connect PR37  PR38 )
   (ARCS ARC63 (Connect PR38  PR39 )
   (ARCS ARC64 (Connect PR38  PR44 )
   (ARCS ARC65 (Connect PR39  PR40 )
   (ARCS ARC66 (Connect PR40  PR41 )
   (ARCS ARC67 (Connect PR40  PR43 )
   (ARCS ARC68 (Connect PR41  PR42 )
   (ARCS ARC69 (Connect PR42  PR43 )
   (ARCS ARC70 (Connect PR43  PR44 )
   (ARCS ARC71 (Connect PR44  PR45 )
   (ARCS ARC72 (Connect PR45  PR46 )
   (ARCS ARC73 (Connect PR46  PR47 )
   (ARCS ARC74 (Connect PR46  PR52 )
   (ARCS ARC75 (Connect PR47  PR48 )
   (ARCS ARC76 (Connect PR48  PR49 )
   (ARCS ARC77 (Connect PR48  PR51 )
   (ARCS ARC78 (Connect PR49  PR50 )
   (ARCS ARC79 (Connect PR50  PR51 )
   (ARCS ARC80 (Connect PR51  PR52 )
   (ARCS ARC81 (Connect PR52  PR53 )
//decoder communications
   (ARCS ARC82 (Connect PR54  PR62 )
   (ARCS ARC83 (Connect PR54  PR75 )
   (ARCS ARC84 (Connect PR54  PR78 )
   (ARCS ARC85 (Connect PR54  PR81 )
   (ARCS ARC86 (Connect PR54  PR84 )
   (ARCS ARC87 (Connect PR54  PR77 )
   (ARCS ARC88 (Connect PR55  PR60 )
   (ARCS ARC89 (Connect PR55  PR61 )
   (ARCS ARC90 (Connect PR55  PR62 )
```

```
  (ARCS ARC91 (Connect PR56  PR63 )
  (ARCS ARC92 (Connect PR56  PR64 )
  (ARCS ARC93 (Connect PR56  PR65 )
  (ARCS ARC94 (Connect PR57  PR66 )
  (ARCS ARC95 (Connect PR57  PR67 )
  (ARCS ARC96 (Connect PR57  PR68 )
  (ARCS ARC97 (Connect PR58  PR69 )
  (ARCS ARC98 (Connect PR58  PR70 )
  (ARCS ARC99 (Connect PR58  PR71 )
  (ARCS ARC100 (Connect PR59  PR73 )
  (ARCS ARC101 (Connect PR60  PR61 )
  (ARCS ARC102 (Connect PR61  PR62 )
  (ARCS ARC103 (Connect PR62  PR65 )
  (ARCS ARC104 (Connect PR63  PR64 )
  (ARCS ARC105 (Connect PR64  PR65 )
  (ARCS ARC106 (Connect PR65  PR68 )
  (ARCS ARC107 (Connect PR66  PR67 )
  (ARCS ARC108 (Connect PR67  PR68 )
  (ARCS ARC109 (Connect PR68  PR71 )
  (ARCS ARC110 (Connect PR69  PR70 )
  (ARCS ARC111 (Connect PR70  PR71 )
  (ARCS ARC112 (Connect PR71  PR72 )
  (ARCS ARC113 (Connect PR71  PR74 )
  (ARCS ARC114 (Connect PR72  PR77 )
  (ARCS ARC115 (Connect PR72  PR80 )
  (ARCS ARC116 (Connect PR72  PR83 )
  (ARCS ARC117 (Connect PR72  PR86 )
  (ARCS ARC118 (Connect PR73  PR75 )
  (ARCS ARC119 (Connect PR73  PR78 )
  (ARCS ARC120 (Connect PR73  PR81 )
  (ARCS ARC121 (Connect PR73  PR84 )
  (ARCS ARC122 (Connect PR75  PR76 )
  (ARCS ARC123 (Connect PR76  PR77 )
  (ARCS ARC124 (Connect PR77  PR80 )
  (ARCS ARC125 (Connect PR77  PR87 )
  (ARCS ARC126 (Connect PR78  PR79 )
  (ARCS ARC127 (Connect PR79  PR80 )
  (ARCS ARC128 (Connect PR80  PR83 )
  (ARCS ARC129 (Connect PR80  PR87 )
  (ARCS ARC130 (Connect PR81  PR82 )
  (ARCS ARC131 (Connect PR82  PR83 )
  (ARCS ARC132 (Connect PR83  PR86 )
  (ARCS ARC133 (Connect PR83  PR87 )
  (ARCS ARC134 (Connect PR84  PR85 )
  (ARCS ARC135 (Connect PR85  PR86 )
  (ARCS ARC136 (Connect PR86  PR74 )
  (ARCS ARC137 (Connect PR86  PR87 )
=========================================
```

B.2 Technology Library File

The technology library file gives the parameters of three different processing elements (PEs) used in the GSM voice CODEC example. The PE parameters have been derived

from the 0.07μm Crusoe processor [115]. Furthermore, the technology file also gives the execution properties of the 87 tasks, where tasks 0-52 correspond to the tasks 0-52 in the task graph of Figure 4.16, tasks 53-86 correspond to the tasks 0-33 in the task graph of Figure 4.17. The technology library file also gives the parameters of two different communication links and the communication amount between tasks. For the meaning of the parameters, the reader is referred to Section A.5 and Table A.2.

```
// Technology file for GSM voice CODEC

=========================================
//part 1: parameters of PEs and tasks

//parameters of PE0
{PE 0
(pe_type GPP)(dvs 1)(abb 1)
(freq 4.21e+09)(vmax 1)(vmin 0.5)(vbsmin -1)
(k1 0.063)(k2 0.153)(k3 1.87407e-07)(k4 1.83)(k5 4.19)
(vth1 0.244)(ceff 1.11e-09)(lg 4e+06)(vbsmax 0)
(task 0)(execyc 3617)(dynpwr 9.42097)(leakpwr 5.53295)
(task 1)(execyc 23628)(dynpwr 9.25273)(leakpwr 4.76656)
(task 2)(execyc 542)(dynpwr 7.43023)(leakpwr 6.58907)
(task 3)(execyc 3460)(dynpwr 8.07511)(leakpwr 6.8788)
(task 4)(execyc 132)(dynpwr 13.2716)(leakpwr 5.42079)
(task 5)(execyc 315)(dynpwr 11.9164)(leakpwr 3.97213)
(task 6)(execyc 160)(dynpwr 13.8324)(leakpwr 4.86002)
(task 7)(execyc 52)(dynpwr 12.4304)(leakpwr 5.32733)
(task 8)(execyc 552)(dynpwr 13.1408)(leakpwr 4.61702)
(task 9)(execyc 527)(dynpwr 7.06572)(leakpwr 4.14971)
(task 10)(execyc 55)(dynpwr 5.60772)(leakpwr 5.60772)
(task 11)(execyc 552)(dynpwr 6.26195)(leakpwr 3.08424)
(task 12)(execyc 323)(dynpwr 8.14054)(leakpwr 4.00952)
(task 13)(execyc 52)(dynpwr 12.5239)(leakpwr 6.16849)
(task 14)(execyc 552)(dynpwr 8.13119)(leakpwr 5.8881)
(task 15)(execyc 527)(dynpwr 12.608)(leakpwr 5.14975)
(task 16)(execyc 471)(dynpwr 10.2808)(leakpwr 8.41158)
(task 17)(execyc 41)(dynpwr 7.77604)(leakpwr 7.17788)
(task 18)(execyc 862)(dynpwr 6.31803)(leakpwr 5.83203)
(task 19)(execyc 3617)(dynpwr 9.05646)(leakpwr 6.83207)
(task 20)(execyc 728)(dynpwr 11.0752)(leakpwr 2.94405)
(task 21)(execyc 5013)(dynpwr 11.5986)(leakpwr 4.2899)
(task 22)(execyc 329)(dynpwr 11.5893)(leakpwr 7.10311)
(task 23)(execyc 1140)(dynpwr 9.86958)(leakpwr 5.08433)
(task 24)(execyc 1966)(dynpwr 6.84142)(leakpwr 4.37402)
(task 25)(execyc 981)(dynpwr 12.2809)(leakpwr 4.54225)
(task 26)(execyc 148)(dynpwr 12.0753)(leakpwr 3.81325)
(task 27)(execyc 200)(dynpwr 12.5239)(leakpwr 6.16849)
(task 28)(execyc 759)(dynpwr 7.17788)(leakpwr 4.03756)
(task 29)(execyc 5013)(dynpwr 5.55164)(leakpwr 4.72918)
(task 30)(execyc 329)(dynpwr 7.40219)(leakpwr 3.81325)
(task 31)(execyc 1140)(dynpwr 7.19657)(leakpwr 2.14963)
(task 32)(execyc 1966)(dynpwr 9.81351)(leakpwr 4.20579)
(task 33)(execyc 981)(dynpwr 9.02843)(leakpwr 4.05625)
```

```
(task 34) (execyc 148) (dynpwr 8.41158) (leakpwr 8.41158)
(task 35) (execyc 200) (dynpwr 7.71996) (leakpwr 5.36472)
(task 36) (execyc 728) (dynpwr 7.06572) (leakpwr 4.14971)
(task 37) (execyc 5013) (dynpwr 8.99104) (leakpwr 3.15901)
(task 38) (execyc 5014) (dynpwr 14.0286) (leakpwr 3.72913)
(task 39) (execyc 544) (dynpwr 9.25273) (leakpwr 7.57042)
(task 40) (execyc 1966) (dynpwr 6.28064) (leakpwr 4.93479)
(task 41) (execyc 981) (dynpwr 11.2154) (leakpwr 2.80386)
(task 42) (execyc 148) (dynpwr 8.18727) (leakpwr 3.02817)
(task 43) (execyc 120) (dynpwr 6.82272) (leakpwr 2.52347)
(task 44) (execyc 759) (dynpwr 8.50504) (leakpwr 4.57964)
(task 45) (execyc 5013) (dynpwr 11.0098) (leakpwr 6.74795)
(task 46) (execyc 561) (dynpwr 6.6358) (leakpwr 2.7104)
(task 47) (execyc 544) (dynpwr 4.86002) (leakpwr 4.48617)
(task 48) (execyc 259) (dynpwr 11.9631) (leakpwr 6.72926)
(task 49) (execyc 228) (dynpwr 14.5801) (leakpwr 4.11233)
(task 50) (execyc 395) (dynpwr 6.16849) (leakpwr 5.04695)
(task 51) (execyc 120) (dynpwr 8.67327) (leakpwr 6.28064)
(task 52) (execyc 3617) (dynpwr 6.07503) (leakpwr 3.27117)
(task 53) (execyc 2882) (dynpwr 10.0191) (leakpwr 4.93479)
(task 54) (execyc 162) (dynpwr 11.0285) (leakpwr 7.66388)
(task 55) (execyc 162) (dynpwr 6.31803) (leakpwr 5.83203)
(task 56) (execyc 200) (dynpwr 7.77604) (leakpwr 4.37402)
(task 57) (execyc 73) (dynpwr 6.47691) (leakpwr 3.8039)
(task 58) (execyc 100) (dynpwr 10.1687) (leakpwr 4.78525)
(task 59) (execyc 228) (dynpwr 9.37423) (leakpwr 6.5143)
(task 60) (execyc 148) (dynpwr 5.24322) (leakpwr 5.0376)
(task 61) (execyc 2952) (dynpwr 9.76677) (leakpwr 7.991)
(task 62) (execyc 228) (dynpwr 8.27138) (leakpwr 5.74791)
(task 63) (execyc 148) (dynpwr 10.2621) (leakpwr 6.56103)
(task 64) (execyc 2952) (dynpwr 8.89758) (leakpwr 4.1871)
(task 65) (execyc 981) (dynpwr 7.29003) (leakpwr 6.72926)
(task 66) (execyc 148) (dynpwr 9.75743) (leakpwr 7.06572)
(task 67) (execyc 2952) (dynpwr 13.8324) (leakpwr 4.86002)
(task 68) (execyc 228) (dynpwr 8.67327) (leakpwr 6.28064)
(task 69) (execyc 395) (dynpwr 10.4677) (leakpwr 8.22465)
(task 70) (execyc 63) (dynpwr 11.5426) (leakpwr 6.21522)
(task 71) (execyc 2000) (dynpwr 10.0939) (leakpwr 3.9254)
(task 72) (execyc 160) (dynpwr 7.77604) (leakpwr 4.37402)
(task 73) (execyc 2882) (dynpwr 10.2621) (leakpwr 6.56103)
(task 74) (execyc 115) (dynpwr 8.55177) (leakpwr 5.46752)
(task 75) (execyc 63) (dynpwr 9.86958) (leakpwr 5.08433)
(task 76) (execyc 15200) (dynpwr 9.81351) (leakpwr 3.27117)
(task 77) (execyc 123) (dynpwr 9.5892) (leakpwr 7.23396)
(task 78) (execyc 552) (dynpwr 11.1033) (leakpwr 5.71987)
(task 79) (execyc 16370) (dynpwr 8.97235) (leakpwr 2.24309)
(task 80) (execyc 115) (dynpwr 8.63589) (leakpwr 4.44879)
(task 81) (execyc 552) (dynpwr 11.5426) (leakpwr 6.21522)
(task 82) (execyc 867) (dynpwr 7.89754) (leakpwr 4.25252)
(task 83) (execyc 1060) (dynpwr 8.82281) (leakpwr 6.1311)
(task 84) (execyc 63) (dynpwr 8.14054) (leakpwr 4.00952)
(task 85) (execyc 1469) (dynpwr 10.3369) (leakpwr 2.74778)
(task 86) (execyc 15596) (dynpwr 9.42097) (leakpwr 3.66371)
}

// parameters of PE1
{PE 1
```

```
(pe_type GPP)(dvs 1)(abb 1)
(freq 4.21e+09)(vmax 1)(vmin 0.5)(vbsmin -1)
(k1 0.063)(k2 0.153)(k3 1.87407e-07)(k4 1.83)(k5 4.19)
(vth1 0.244)(ceff 1.11e-09)(lg 4e+06)(vbsmax 0)
(task 0)(execyc 3617)(dynpwr 7.38349)(leakpwr 1.9627)
(task 1)(execyc 23628)(dynpwr 9.92566)(leakpwr 6.89749)
(task 2)(execyc 542)(dynpwr 5.60772)(leakpwr 3.73848)
(task 3)(execyc 3460)(dynpwr 5.86006)(leakpwr 4.42075)
(task 4)(execyc 132)(dynpwr 7.58911)(leakpwr 5.49556)
(task 5)(execyc 315)(dynpwr 10.3275)(leakpwr 5.56099)
(task 6)(execyc 160)(dynpwr 9.29012)(leakpwr 3.79456)
(task 7)(execyc 52)(dynpwr 8.74804)(leakpwr 2.4674)
(task 8)(execyc 552)(dynpwr 9.5705)(leakpwr 5.38341)
(task 9)(execyc 527)(dynpwr 11.2154)(leakpwr 7.47696)
(task 10)(execyc 55)(dynpwr 10.7668)(leakpwr 6.05634)
(task 11)(execyc 552)(dynpwr 13.6735)(leakpwr 4.08429)
(task 12)(execyc 323)(dynpwr 10.2808)(leakpwr 8.41158)
(task 13)(execyc 52)(dynpwr 5.44883)(leakpwr 4.83198)
(task 14)(execyc 552)(dynpwr 13.8511)(leakpwr 3.90671)
(task 15)(execyc 527)(dynpwr 8.38354)(leakpwr 3.76652)
(task 16)(execyc 471)(dynpwr 10.8322)(leakpwr 6.92553)
(task 17)(execyc 41)(dynpwr 9.15927)(leakpwr 3.9254)
(task 18)(execyc 862)(dynpwr 7.58911)(leakpwr 5.49556)
(task 19)(execyc 3617)(dynpwr 14.0286)(leakpwr 3.72913)
(task 20)(execyc 728)(dynpwr 5.8881)(leakpwr 3.45809)
(task 21)(execyc 5013)(dynpwr 10.0752)(leakpwr 3.00948)
(task 22)(execyc 329)(dynpwr 11.0098)(leakpwr 6.74795)
(task 23)(execyc 1140)(dynpwr 8.57981)(leakpwr 7.30873)
(task 24)(execyc 1966)(dynpwr 6.44888)(leakpwr 2.89732)
(task 25)(execyc 981)(dynpwr 9.76677)(leakpwr 7.991)
(task 26)(execyc 148)(dynpwr 7.29003)(leakpwr 2.05616)
(task 27)(execyc 200)(dynpwr 6.47691)(leakpwr 3.8039)
(task 28)(execyc 759)(dynpwr 8.41158)(leakpwr 5.60772)
(task 29)(execyc 5013)(dynpwr 12.7856)(leakpwr 4.03756)
(task 30)(execyc 329)(dynpwr 11.2154)(leakpwr 7.47696)
(task 31)(execyc 1140)(dynpwr 7.73865)(leakpwr 3.47678)
(task 32)(execyc 1966)(dynpwr 8.07511)(leakpwr 6.8788)
(task 33)(execyc 981)(dynpwr 8.76673)(leakpwr 4.31794)
(task 34)(execyc 148)(dynpwr 8.01904)(leakpwr 2.26178)
(task 35)(execyc 200)(dynpwr 7.09376)(leakpwr 3.18705)
(task 36)(execyc 728)(dynpwr 10.6547)(leakpwr 8.03773)
(task 37)(execyc 5013)(dynpwr 9.68266)(leakpwr 3.40202)
(task 38)(execyc 5014)(dynpwr 7.6265)(leakpwr 7.32742)
(task 39)(execyc 544)(dynpwr 11.2715)(leakpwr 5.55164)
(task 40)(execyc 1966)(dynpwr 13.1408)(leakpwr 4.61702)
(task 41)(execyc 981)(dynpwr 6.16849)(leakpwr 3.17771)
(task 42)(execyc 148)(dynpwr 7.06572)(leakpwr 4.14971)
(task 43)(execyc 120)(dynpwr 9.59854)(leakpwr 2.55151)
(task 44)(execyc 759)(dynpwr 7.09376)(leakpwr 3.18705)
(task 45)(execyc 5013)(dynpwr 7.16853)(leakpwr 4.98152)
(task 46)(execyc 561)(dynpwr 7.40219)(leakpwr 3.81325)
(task 47)(execyc 544)(dynpwr 12.7108)(leakpwr 3.17771)
(task 48)(execyc 259)(dynpwr 7.6078)(leakpwr 2.67301)
(task 49)(execyc 228)(dynpwr 12.3931)(leakpwr 3.49548)
(task 50)(execyc 395)(dynpwr 7.98165)(leakpwr 5.10302)
(task 51)(execyc 120)(dynpwr 11.608)(leakpwr 5.21518)
(task 52)(execyc 3617)(dynpwr 8.67327)(leakpwr 6.28064)
```

```
(task 53)(execyc 2882)(dynpwr 7.29003)(leakpwr 3.9254)
(task 54)(execyc 162)(dynpwr 6.3928)(leakpwr 4.82264)
(task 55)(execyc 162)(dynpwr 7.65453)(leakpwr 4.49552)
(task 56)(execyc 200)(dynpwr 7.41153)(leakpwr 4.73852)
(task 57)(execyc 73)(dynpwr 10.4677)(leakpwr 4.48617)
(task 58)(execyc 100)(dynpwr 10.3743)(leakpwr 3.64502)
(task 59)(execyc 228)(dynpwr 9.76677)(leakpwr 7.991)
(task 60)(execyc 148)(dynpwr 10.4677)(leakpwr 2.61693)
(task 61)(execyc 2952)(dynpwr 6.2713)(leakpwr 4.00952)
(task 62)(execyc 228)(dynpwr 11.0285)(leakpwr 7.66388)
(task 63)(execyc 148)(dynpwr 10.3275)(leakpwr 5.56099)
(task 64)(execyc 2952)(dynpwr 7.991)(leakpwr 6.0283)
(task 65)(execyc 981)(dynpwr 9.59854)(leakpwr 2.55151)
(task 66)(execyc 148)(dynpwr 12.1127)(leakpwr 4.71048)
(task 67)(execyc 2952)(dynpwr 9.3462)(leakpwr 9.3462)
(task 68)(execyc 228)(dynpwr 14.9539)(leakpwr 3.73848)
(task 69)(execyc 395)(dynpwr 7.58911)(leakpwr 5.49556)
(task 70)(execyc 63)(dynpwr 13.1408)(leakpwr 4.61702)
(task 71)(execyc 2000)(dynpwr 9.5705)(leakpwr 5.38341)
(task 72)(execyc 160)(dynpwr 11.9444)(leakpwr 4.87871)
(task 73)(execyc 2882)(dynpwr 9.9537)(leakpwr 4.0656)
(task 74)(execyc 115)(dynpwr 8.74804)(leakpwr 8.07511)
(task 75)(execyc 63)(dynpwr 8.22465)(leakpwr 2.05616)
(task 76)(execyc 15200)(dynpwr 10.2808)(leakpwr 8.41158)
(task 77)(execyc 123)(dynpwr 5.24322)(leakpwr 5.0376)
(task 78)(execyc 552)(dynpwr 13.8511)(leakpwr 3.90671)
(task 79)(execyc 16370)(dynpwr 12.5519)(leakpwr 3.33659)
(task 80)(execyc 115)(dynpwr 11.8136)(leakpwr 3.14032)
(task 81)(execyc 552)(dynpwr 9.75743)(leakpwr 7.06572)
(task 82)(execyc 867)(dynpwr 6.16849)(leakpwr 3.17771)
(task 83)(execyc 1060)(dynpwr 10.4677)(leakpwr 8.22465)
(task 84)(execyc 63)(dynpwr 9.55181)(leakpwr 3.53286)
(task 85)(execyc 1469)(dynpwr 8.86954)(leakpwr 3.28051)
(task 86)(execyc 15596)(dynpwr 9.67331)(leakpwr 4.34598)
}

// parameters of PE2
{PE 2
(pe_type GPP)(dvs 1)(abb 1)
(freq 4.21e+09)(vmax 1)(vmin 0.5)(vbsmin -1)
(k1 0.063)(k2 0.153)(k3 1.87407e-07)(k4 1.83)(k5 4.19)
(vth1 0.244)(ceff 1.11e-09)(lg 4e+06)(vbsmax 0)
(task 0)(execyc 3617)(dynpwr 9.42097)(leakpwr 5.53295)
(task 1)(execyc 23628)(dynpwr 9.25273)(leakpwr 4.76656)
(task 2)(execyc 542)(dynpwr 7.43023)(leakpwr 6.58907)
(task 3)(execyc 3460)(dynpwr 8.07511)(leakpwr 6.8788)
(task 4)(execyc 132)(dynpwr 13.2716)(leakpwr 5.42079)
(task 5)(execyc 315)(dynpwr 11.9164)(leakpwr 3.97213)
(task 6)(execyc 160)(dynpwr 13.8324)(leakpwr 4.86002)
(task 7)(execyc 52)(dynpwr 12.4304)(leakpwr 5.32733)
(task 8)(execyc 552)(dynpwr 13.1408)(leakpwr 4.61702)
(task 9)(execyc 527)(dynpwr 7.06572)(leakpwr 4.14971)
(task 10)(execyc 55)(dynpwr 5.60772)(leakpwr 5.60772)
(task 11)(execyc 552)(dynpwr 6.26195)(leakpwr 3.08424)
(task 12)(execyc 323)(dynpwr 8.14054)(leakpwr 4.00952)
(task 13)(execyc 52)(dynpwr 12.5239)(leakpwr 6.16849)
(task 14)(execyc 552)(dynpwr 8.13119)(leakpwr 5.8881)
```

```
(task 15) (execyc 527) (dynpwr 12.608) (leakpwr 5.14975)
(task 16) (execyc 471) (dynpwr 10.2808) (leakpwr 8.41158)
(task 17) (execyc 41) (dynpwr 7.77604) (leakpwr 7.17788)
(task 18) (execyc 862) (dynpwr 6.31803) (leakpwr 5.83203)
(task 19) (execyc 3617) (dynpwr 9.05646) (leakpwr 6.83207)
(task 20) (execyc 728) (dynpwr 11.0752) (leakpwr 2.94405)
(task 21) (execyc 5013) (dynpwr 11.5986) (leakpwr 4.2899)
(task 22) (execyc 329) (dynpwr 11.5893) (leakpwr 7.10311)
(task 23) (execyc 1140) (dynpwr 9.86958) (leakpwr 5.08433)
(task 24) (execyc 1966) (dynpwr 6.84142) (leakpwr 4.37402)
(task 25) (execyc 981) (dynpwr 12.2809) (leakpwr 4.54225)
(task 26) (execyc 148) (dynpwr 12.0753) (leakpwr 3.81325)
(task 27) (execyc 200) (dynpwr 12.5239) (leakpwr 6.16849)
(task 28) (execyc 759) (dynpwr 7.17788) (leakpwr 4.03756)
(task 29) (execyc 5013) (dynpwr 5.55164) (leakpwr 4.72918)
(task 30) (execyc 329) (dynpwr 7.40219) (leakpwr 3.81325)
(task 31) (execyc 1140) (dynpwr 7.19657) (leakpwr 2.14963)
(task 32) (execyc 1966) (dynpwr 9.81351) (leakpwr 4.20579)
(task 33) (execyc 981) (dynpwr 9.02843) (leakpwr 4.05625)
(task 34) (execyc 148) (dynpwr 8.41158) (leakpwr 8.41158)
(task 35) (execyc 200) (dynpwr 7.71996) (leakpwr 5.36472)
(task 36) (execyc 728) (dynpwr 7.06572) (leakpwr 4.14971)
(task 37) (execyc 5013) (dynpwr 8.99104) (leakpwr 3.15901)
(task 38) (execyc 5014) (dynpwr 14.0286) (leakpwr 3.72913)
(task 39) (execyc 544) (dynpwr 9.25273) (leakpwr 7.57042)
(task 40) (execyc 1966) (dynpwr 6.28064) (leakpwr 4.93479)
(task 41) (execyc 981) (dynpwr 11.2154) (leakpwr 2.80386)
(task 42) (execyc 148) (dynpwr 8.18727) (leakpwr 3.02817)
(task 43) (execyc 120) (dynpwr 6.82272) (leakpwr 2.52347)
(task 44) (execyc 759) (dynpwr 8.50504) (leakpwr 4.57964)
(task 45) (execyc 5013) (dynpwr 11.0098) (leakpwr 6.74795)
(task 46) (execyc 561) (dynpwr 6.6358) (leakpwr 2.7104)
(task 47) (execyc 544) (dynpwr 4.86002) (leakpwr 4.48617)
(task 48) (execyc 259) (dynpwr 11.9631) (leakpwr 6.72926)
(task 49) (execyc 228) (dynpwr 14.5801) (leakpwr 4.11233)
(task 50) (execyc 395) (dynpwr 6.16849) (leakpwr 5.04695)
(task 51) (execyc 120) (dynpwr 8.67327) (leakpwr 6.28064)
(task 52) (execyc 3617) (dynpwr 6.07503) (leakpwr 3.27117)
(task 53) (execyc 2882) (dynpwr 10.0191) (leakpwr 4.93479)
(task 54) (execyc 162) (dynpwr 11.0285) (leakpwr 7.66388)
(task 55) (execyc 162) (dynpwr 6.31803) (leakpwr 5.83203)
(task 56) (execyc 200) (dynpwr 7.77604) (leakpwr 4.37402)
(task 57) (execyc 73) (dynpwr 6.47691) (leakpwr 3.8039)
(task 58) (execyc 100) (dynpwr 10.1687) (leakpwr 4.78525)
(task 59) (execyc 228) (dynpwr 9.37423) (leakpwr 6.5143)
(task 60) (execyc 148) (dynpwr 5.24322) (leakpwr 5.0376)
(task 61) (execyc 2952) (dynpwr 9.76677) (leakpwr 7.991)
(task 62) (execyc 228) (dynpwr 8.27138) (leakpwr 5.74791)
(task 63) (execyc 148) (dynpwr 10.2621) (leakpwr 6.56103)
(task 64) (execyc 2952) (dynpwr 8.89758) (leakpwr 4.1871)
(task 65) (execyc 981) (dynpwr 7.29003) (leakpwr 6.72926)
(task 66) (execyc 148) (dynpwr 9.75743) (leakpwr 7.06572)
(task 67) (execyc 2952) (dynpwr 13.8324) (leakpwr 4.86002)
(task 68) (execyc 228) (dynpwr 8.67327) (leakpwr 6.28064)
(task 69) (execyc 395) (dynpwr 10.4677) (leakpwr 8.22465)
(task 70) (execyc 63) (dynpwr 11.5426) (leakpwr 6.21522)
(task 71) (execyc 2000) (dynpwr 10.0939) (leakpwr 3.9254)
```

```
(task 72)(execyc 160)(dynpwr 7.77604)(leakpwr 4.37402)
(task 73)(execyc 2882)(dynpwr 10.2621)(leakpwr 6.56103)
(task 74)(execyc 115)(dynpwr 8.55177)(leakpwr 5.46752)
(task 75)(execyc 63)(dynpwr 9.86958)(leakpwr 5.08433)
(task 76)(execyc 15200)(dynpwr 9.81351)(leakpwr 3.27117)
(task 77)(execyc 123)(dynpwr 9.5892)(leakpwr 7.23396)
(task 78)(execyc 552)(dynpwr 11.1033)(leakpwr 5.71987)
(task 79)(execyc 16370)(dynpwr 8.97235)(leakpwr 2.24309)
(task 80)(execyc 115)(dynpwr 8.63589)(leakpwr 4.44879)
(task 81)(execyc 552)(dynpwr 11.5426)(leakpwr 6.21522)
(task 82)(execyc 867)(dynpwr 7.89754)(leakpwr 4.25252)
(task 83)(execyc 1060)(dynpwr 8.82281)(leakpwr 6.1311)
(task 84)(execyc 63)(dynpwr 8.14054)(leakpwr 4.00952)
(task 85)(execyc 1469)(dynpwr 10.3369)(leakpwr 2.74778)
(task 86)(execyc 15596)(dynpwr 9.42097)(leakpwr 3.66371)
}

==========================================
//part 2: parameters of CLs

// parameters of CL0
{CL 0
(stpwr 0)(freq 3e+09)(dynpwr 1)
}

// parameters of CL1
{CL 1
(stpwr 0)(freq 6e+09)(dynpwr 2)
}

==========================================
//part 3: communication amount
{COM_AMOUNT
(com 0)(amount 330)
(com 1)(amount 16)
(com 2)(amount 16)
(com 3)(amount 16)
(com 4)(amount 16)
(com 5)(amount 26)
(com 6)(amount 28)
(com 7)(amount 26)
(com 8)(amount 240)
(com 9)(amount 320)
(com 10)(amount 16)
(com 11)(amount 240)
(com 12)(amount 16)
(com 13)(amount 36)
(com 14)(amount 16)
(com 15)(amount 16)
(com 16)(amount 16)
(com 17)(amount 16)
(com 18)(amount 16)
(com 19)(amount 16)
(com 20)(amount 16)
(com 21)(amount 16)
(com 22)(amount 16)
(com 23)(amount 16)
```

```
(com 24)(amount 26)
(com 25)(amount 16)
(com 26)(amount 16)
(com 27)(amount 16)
(com 28)(amount 28)
(com 29)(amount 16)
(com 30)(amount 16)
(com 31)(amount 16)
(com 32)(amount 26)
(com 33)(amount 16)
(com 34)(amount 16)
(com 35)(amount 240)
(com 36)(amount 16)
(com 37)(amount 80)
(com 38)(amount 80)
(com 39)(amount 80)
(com 40)(amount 80)
(com 41)(amount 324)
(com 42)(amount 100)
(com 43)(amount 80)
(com 44)(amount 80)
(com 45)(amount 26)
(com 46)(amount 2)
(com 47)(amount 32)
(com 48)(amount 28)
(com 49)(amount 160)
(com 50)(amount 240)
(com 51)(amount 324)
(com 52)(amount 100)
(com 53)(amount 80)
(com 54)(amount 80)
(com 55)(amount 26)
(com 56)(amount 2)
(com 57)(amount 32)
(com 58)(amount 28)
(com 59)(amount 160)
(com 60)(amount 240)
(com 61)(amount 324)
(com 62)(amount 100)
(com 63)(amount 80)
(com 64)(amount 80)
(com 65)(amount 26)
(com 66)(amount 2)
(com 67)(amount 32)
(com 68)(amount 28)
(com 69)(amount 160)
(com 70)(amount 240)
(com 71)(amount 324)
(com 72)(amount 100)
(com 73)(amount 80)
(com 74)(amount 80)
(com 75)(amount 26)
(com 76)(amount 2)
(com 77)(amount 32)
(com 78)(amount 28)
(com 79)(amount 160)
(com 80)(amount 240)
```

```
(com 81)(amount 240)
(com 82)(amount 16)
(com 83)(amount 16)
(com 84)(amount 16)
(com 85)(amount 16)
(com 86)(amount 16)
(com 87)(amount 7)
(com 88)(amount 2)
(com 89)(amount 2)
(com 90)(amount 7)
(com 91)(amount 2)
(com 92)(amount 2)
(com 93)(amount 7)
(com 94)(amount 2)
(com 95)(amount 2)
(com 96)(amount 7)
(com 97)(amount 2)
(com 98)(amount 2)
(com 99)(amount 16)
(com 100)(amount 24)
(com 101)(amount 80)
(com 102)(amount 240)
(com 103)(amount 24)
(com 104)(amount 80)
(com 105)(amount 240)
(com 106)(amount 24)
(com 107)(amount 80)
(com 108)(amount 240)
(com 109)(amount 24)
(com 110)(amount 80)
(com 111)(amount 320)
(com 112)(amount 240)
(com 113)(amount 26)
(com 114)(amount 28)
(com 115)(amount 26)
(com 116)(amount 240)
(com 117)(amount 16)
(com 118)(amount 16)
(com 119)(amount 16)
(com 120)(amount 16)
(com 121)(amount 16)
(com 122)(amount 16)
(com 123)(amount 16)
(com 124)(amount 26)
(com 125)(amount 16)
(com 126)(amount 16)
(com 127)(amount 16)
(com 128)(amount 28)
(com 129)(amount 16)
(com 130)(amount 16)
(com 131)(amount 16)
(com 132)(amount 26)
(com 133)(amount 16)
(com 134)(amount 16)
(com 135)(amount 16)
(com 136)(amount 240)
}
```

Appendix C

Dummy Task in Task Graphs

This appendix introduces the concept of dummy task, and shows how dummy task provides extra modelling capability to task graphs.

A dummy task is a task with zero execution time, or a task with non-zero execution time but not allocated to any physical processing element (PE). Dummy tasks do not consume any resources but can provide extra modelling capability to task graphs. The source and sink task are examples of dummy task with zero execution time (Chapter 1, Section 1.3.1). The introduction of the source task and the sink task makes it possible to model applications containing several concurrently running but nearly decoupled tasks, as shown in Figure C.1. Some tasks in the task graph may have a release time, i.e., the tasks cannot be initiated until a certain time after the invocation of the task graph. Also, some tasks may have various deadlines. Release times can be modelled by inserting dummy tasks between certain tasks and the source task, as shown in Figure C.2. These dummy tasks are associated with a certain

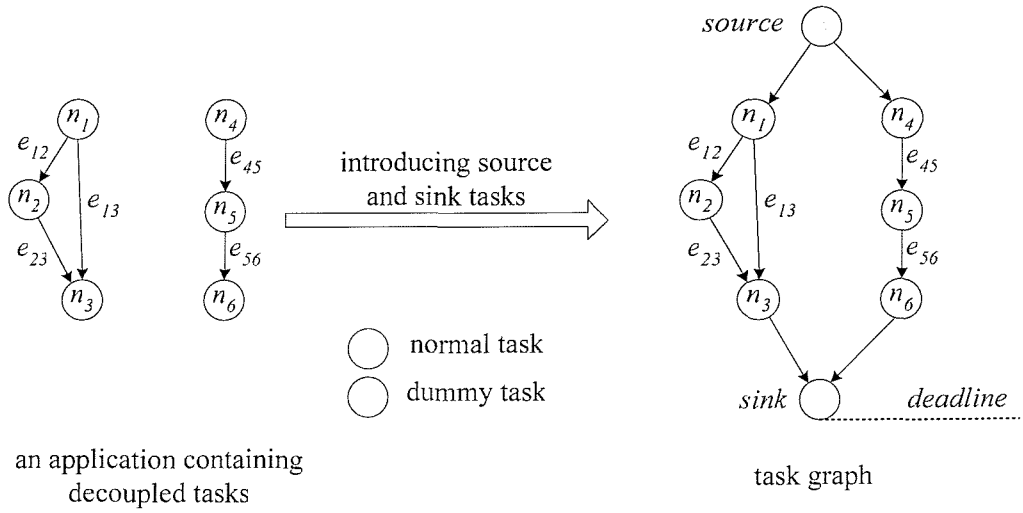


Figure C.1: A task graph modelling an application containing decoupled tasks

execution time but are not mapped to any physical PE. Similarly, multiple deadlines can be modelled by inserting dummy nodes between certain tasks and the sink task, as shown in Figure C.3.

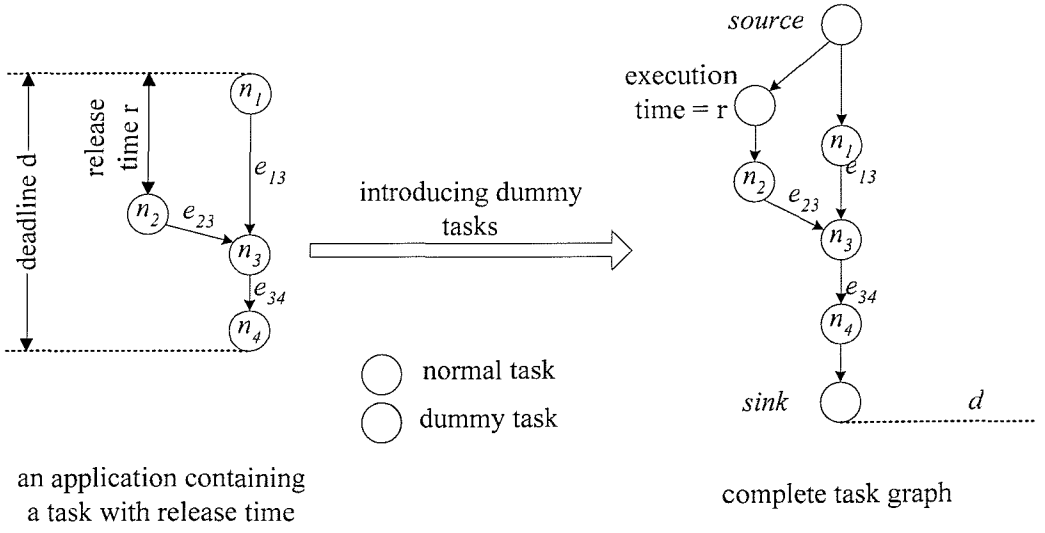


Figure C.2: A task graph modelling tasks with release times

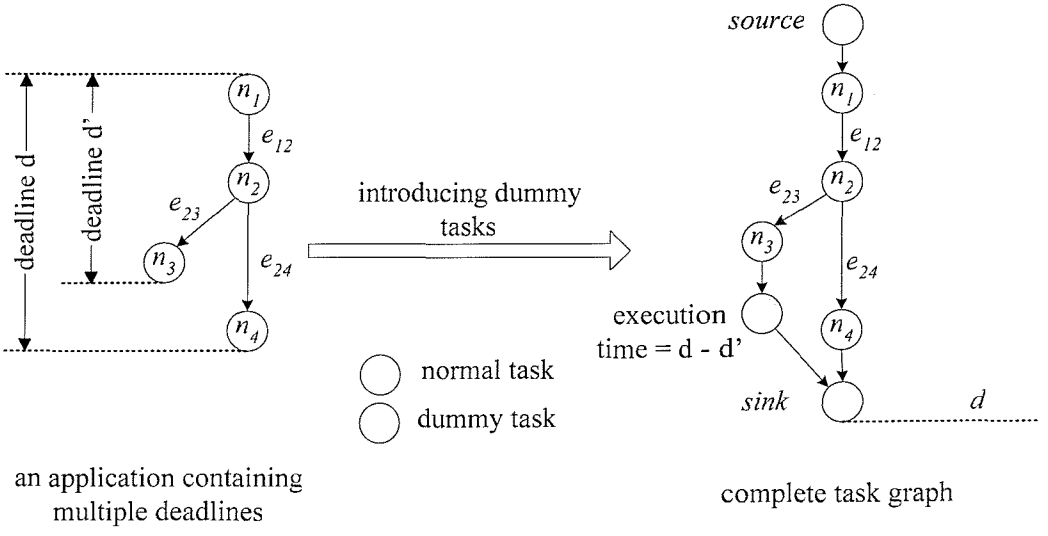


Figure C.3: A task graph modelling multiple deadlines

References

[1] G. D. Micheli, R. Ernst, and W. Wolf, *Readings in hardware/software co-design*, Morgan Kaufmann Publishers, 2001.

[2] M. T. Schmitz, B. M. Al Hashimi, and P. Eles, *System-level design techniques for energy-efficient embedded systems*, Kluwer Academic Publishers, 2004.

[3] T.-Y. Yen and W. Wolf, *Hardware-Software Co-Synthesis of Distributed Embedded Systems*, Kluwer Academic Publishers, 1996.

[4] W. H. Wolf, "Hardware-software co-design of embedded systems," *Proceedings of the IEEE*, vol. 82, no. 7, 1994, pp. 967-89.

[5] "BOSCH's Controller Area Network," http://www.can.bosch.com/, 2004.

[6] "Philips Semiconductors I2C-bus," http://www.semiconductors.philips.com/buses/i2c/index.html, 2004.

[7] "About RapidIO," http://www.rapidio.org/about, 2004.

[8] C. Drosos, M. Zayadine, and D. Metafas, "Real-time communication protocol development using SDL for an embedded system on chip based on ARM microcontroller," in *Proceedings Euromicro Conference on Real-Time Systems*, 2001, pp. 89-94.

[9] J. Staunstrup and W. Wolf, *Hardware/software co-design principles and practice*, Kluwer Academic Publishers, 1997.

[10] W. Wolf, "A decade of hardware/software codesign," *Computer*, vol. 36, no. 4, 2003, pp. 38-41.

[11] G. D. Micheli, "Computer-aided hardware-software codesign," *IEEE Micro*, vol. 14, no. 4, 1994, pp. 10-16.

[12] G. D. Micheli and R. K. Gupta, "Hardware/software co-design," *Proceedings of the IEEE*, vol. 85, no. 3, 1997, pp. 349-65.

[13] D. D. Gajski and F. Vahid, "Specification and design of embedded hardware-software systems," *IEEE Design & Test of Computers*, vol. 12, no. 1, 1995, pp. 53-67.

[14] B. P. Dave, G. Lakshminarayana, and N. K. Jha, "COSYN: Hardware-software co-synthesis of heterogeneous distributed embedded systems," *IEEE*

Transactions on Very Large Scale Integration (VLSI) Systems, vol. 7, no. 1, 1999, pp. 92-104.

[15] N. K. Jha, "Low power system scheduling and synthesis," in *Proceedings IEEE/ACM International Conference on Computer Aided Design*, 2001, pp. 259-63.

[16] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli, "Design of embedded systems: formal models, validation, and synthesis," *Proceedings of the IEEE*, vol. 85, no. 3, 1997, pp. 366-90.

[17] G. D. Micheli, *Synthesis and optimization of digital circuits*, McGraw-Hill Education, 1994.

[18] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI design: a systems perspective*, Second ed, Addison-Wesley Publishing Company, 1993.

[19] D. D. Gajski, N. D. Dutt, A. C.-H. Wu, and S. Y.-L. Lin, *High-level synthesis: introduction to chip and system design*, Kluwer Academic Publishers, 1992.

[20] J. P. Elliott, *Understanding behavioral synthesis: a practical guide to high-level design*, Kluwer Academic Publishers, 1999.

[21] P. Ashenden, *The designer's guide to VHDL*, Second ed, Morgan Kaufmann Publishers, 2001.

[22] S. Palnitkar, *Verilog HDL*, Second ed, Prentice Hall PTR, 2003.

[23] J. Bhasker, *A SystemC primer*, Star Galaxy Publishing, 2002.

[24] S. Devadas, A. Ghosh, and K. Keutzer, *Logic synthesis*, McGraw-Hill, 1994.

[25] N. A. Sherwani, *Algorithms for VLSI physical design automation*, Third ed, Kluwer Academic Publishers, 1995.

[26] S. H. Gerez, *Algorithms for VLSI design automation*, John Wiley & Sons, 1998.

[27] M. Sarrafzadeh and C. K. Wong, *An introduction to VLSI physical design*, McGraw-Hill Science/Engineering/Math, 1996.

[28] R. K. Gupta, *Co-synthesis of hardware and software for digital embeded systems*, PhD Thesis, Stanford University, 1993.

[29] R. K. Gupta, G. D. Micheli, and C. N. Coelho, "Program implementation schemes for hardware-software systems," *Computer*, vol. 27, no. 1, 1994, pp. 48-55.

[30] B. Stroustrup, *The C++ programming language*, Addison Wesley Professional, 1997.

[31] H. M. Deitel and P. J. Deitel, *Java*, Prentice Hall, 2002.

[32] R. Leupers and P. Marwedel, *Retargetable Compiler Technology for Embedded Systems - Tools and Applications*, Kluwer Academic Publishers, 2001.

[33] J. A. Rowson, "Hardware/software co-simulation," in *Proceedings Design Automation Conference*, 1994, pp. 439-440.

[34] A. Hoffmann, T. Kogel, and H. Meyr, "A framework for fast hardware-software co-simulation," in *Proceedings Design, Automation and Test in Europe*, 2001, pp. 760-4.

[35] M. El Shobaki, "Verification of embedded real-time systems using hardware/software co-simulation," in *Proceedings EUROMICRO Conference*, 1998, pp. 46-50.

[36] M. Varea and B. Al-Hashimi, "Dual transitions Petri Net based modelling technique for embedded systems specification," in *Proceedings Design, Automation and Test in Europe*, 2001, pp. 566-71.

[37] A. Dasdan, D. Ramanathan, and R. K. Gupta, "A timing-driven design and validation methodology for embedded real-time systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 3, no. 4, 1998, pp. 533-53.

[38] K. Strehl, L. Thiele, D. Ziegenbein, R. Ernst, and J. Teich, "Scheduling hardware/software systems using symbolic techniques," in *Proceedings Seventh International Workshop on Hardware/Software Codesign*, 1999, pp. 173-7.

[39] D. Ziegenbein, K. Richter, R. Ernst, J. Teich, and L. Thiele, "Representation of process mode correlation for scheduling," in *Proceedings IEEE/ACM International Conference on Computer Aided Design*, 1998, pp. 54-6.

[40] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: task graphs for free," in *Proceedings Sixth International Workshop on Hardware/Software Codesign*, 1998, pp. 97-101.

[41] P. Eles, K. Kuchcinski, Z. Peng, A. Doboli, and P. Pop, "Scheduling of conditional process graphs for the synthesis of embedded systems," in *Proceedings Design, Automation and Test in Europe*, 1998, pp. 132-38.

[42] P. Eles, A. Doboli, P. Pop, and Z. Peng, "Scheduling with bus access optimization for distributed embedded systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 5, 2000, pp. 472-91.

[43] L. Benini, A. Bogliolo, and G. D. Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 3, 2000, pp. 299-316.

[44] M. B. Srivastava, A. P. Chandrakasan, and R. W. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 4, no. 1, 1996, pp. 42-55.

[45] C. H. Hwang and A. H. Wu, "A predictive system shutdown method for energy saving of event-driven computation," *ACM Transactions on Design Automation of Electronic Systems*, vol. 5, no. 2, 2000, pp. 226-41.

[46] P. Krishnan, P. M. Long, and J. S. Vitter, "Adaptive disk spindown via optimal rent-to-buy in probabilistic environments," *Algorithmica*, vol. 23, no. 1, 1999, pp. 31-56.

[47] D. P. Helmbold, D. D. E. Long, T. L. Sconyers, and B. Sherrod, "Adaptive disk spin-down for mobile computers," *Mobile Networks and Applications*, vol. 5, no. 4, 2000, pp. 285-97.

[48] F. Douglis, P. Krishnan, and B. Bershad, "Adaptive disk spin-down policies for mobile computers," in *Proceedings Second USENIX Symposium on Mobile and Location Independent Computing*, 1995, pp. 121-37.

[49] Y. H. Lu and G. De-Micheli, "Adaptive hard disk power management on personal computers," in *Proceedings Ninth Great Lakes Symposium on VLSI*, 1999, pp. 50-3.

[50] L. Benini, A. Bogliolo, G. A. Paleologo, and G. De Micheli, "Policy optimization for dynamic power management," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 18, no. 6, 1999, pp. 813-33.

[51] E. Y. Chung, L. Benini, A. Bogliolo, and G. De-Micheli, "Dynamic power management for nonstationary service requests," in *Proceedings Design, Automation and Test in Europe*, 1999, pp. 77-81.

[52] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava, "Power optimization of variable-voltage core-based systems," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 18, no. 12, 1999, pp. 1702-14.

[53] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *Proceedings International Symposium on Low Power Electronics and Design*, 1998, pp. 197-202.

[54] G. Quan and X. Hu, "Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors," in *Proceedings Design Automation Conference*, 2001, pp. 828-33.

[55] M. T. Schmitz and B. M. Al-Hashimi, "Considering power variations of DVS processing elements for energy minimisation in distributed systems," in *Proceedings International Symposium on System Synthesis*, 2001, pp. 250-5.

[56] Y. Zhang, X. Hu, and D. Z. Chen, "Task scheduling and voltage selection for energy minimization," in *Proceedings Design Automation Conference*, 2002, pp. 183-8.

[57] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles, "Energy-efficient mapping and scheduling for DVS enabled distributed embedded systems," in *Proceedings Design, Automation and Test in Europe*, 2002, pp. 514-21.

[58] J. Luo and N. K. Jha, "Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems," in *Proceedings IEEE/ACM International Conference on Computer Aided Design*, 2000, pp. 357-64.

[59] F. Gruian and K. Kuchcinski, "LEneS: task scheduling for low-energy systems using variable supply voltage processors," in *Proceedings Asia and South Pacific Design Automation Conference*, 2001, pp. 449-55.

[60] T. Kuroda, T. Fujita, S. Mita, T. Nagamatsu, S. Yoshioka, K. Suzuki, F. Sano, M. Norishima, M. Murota, M. Kako, M. Kinugawa, M. Kakumu, and T. Sakurai, "A 0.9-V, 150-MHz, 10-mW, 4 mm^2, 2-D discrete cosine transform core processor with variable threshold-voltage (VT) scheme," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 11, 1996, pp. 1770-9.

[61] K. Nose, M. Hirabayashi, H. Kawaguchi, S. Lee, and T. Sakurai, "V_{TH}-hopping scheme to reduce subthreshold leakage for low-power processors," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 3, 2002, pp. 413-9.

[62] M. Miyazaki, G. Ono, and K. Ishibashi, "A 1.2-GIPS/W microprocessor using speed-adaptive threshold-voltage CMOS with forward bias," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 2, 2002, pp. 210-17.

[63] J. T. Kao, M. Miyazaki, and A. P. Chandrakasan, "A 175-MV multiply-accumulate unit using an adaptive supply voltage and body bias architecture," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 11, 2002, pp. 1545-54.

[64] C. H. Kim and K. Roy, "Dynamic V_{TH} scaling scheme for active leakage power reduction," in *Proceedings Design, Automation and Test in Europe*, 2002, pp. 163-7.

[65] D. Wu, B. M. Al-Hashimi, and P. Eles, "Scheduling and mapping of conditional task graphs for the synthesis of low power embedded systems," in *Proceedings Design Automation and Test in Europe*, 2003, pp. 90-5.

[66] D. Wu, B. M. Al-Hashimi, and P. Eles, "Scheduling and mapping of conditional task graph for the synthesis of low power embedded systems," *IEE Proceedings: Computers and Digital Techniques*, vol. 150, no. 5, 2003, pp. 262-273.

[67] A. P. Chandrakasan and R. W. Brodersen, *Low power digital CMOS design*, Kluwer Academic Publishers, 1995.

[68] M. Pedram, "Power minimization in IC design: principles and applications," *ACM Transactions on Design Automation of Electronic Systems*, vol. 1, no. 1, 1996, pp. 3-56.

[69] T. D. Burd and R. W. Brodersen, "Design issues for Dynamic Voltage Scaling," in *Proceedings International Symposium on Low Power Electronic Design*, 2000, pp. 9-14.

[70] T. D. Burd and R. W. Brodersen, *Energy efficient microprocessor design*, Kluwer academic publishers, 2002.

[71] J. P. Halter and F. N. Najm, "A gate-level leakage power reduction method for ultra-low-power CMOS circuits," in *Proceedings Custom Integrated Circuits Conference*, 1997, pp. 475-8.

[72] K. S. Khouri and N. K. Jha, "Leakage power analysis and reduction during behavioral synthesis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 10, no. 6, 2002, pp. 876-85.

[73] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," *IEEE Journal of Solid State Circuits*, vol. 27, no. 4, 1992, pp. 473-84.

[74] N. S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J. S. Hu, M. Jane Irwin, M. Kandemir, and V. Narayanan, "Leakage Current: Moore's Law Meets Static Power," *Computer*, vol. 36, no. 12, 2003, pp. 68-75.

[75] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen, "A dynamic voltage scaled microprocessor system," *IEEE Journal of Solid State Circuits*, vol. 35, no. 11, 2000, pp. 1571-80.

[76] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits," *Proceedings of the IEEE*, vol. 91, no. 2, 2003, pp. 305-27.

[77] D. Duarte, N. Vijaykrishnan, M. J. Irwin, H.-S. Kim, and G. McFarland, "Impact of scaling on the effectiveness of dynamic power reduction schemes," in *Proceedings IEEE International Conference on Computer Design*, 2002, pp. 382-7.

[78] V. Gutnik and A. P. Chandrakasan, "Embedded power supply for low-power DSP," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 5, no. 4, 1997, pp. 425-35.

[79] "Transmeta Coperation: Crusoe Processor," http://www.transmeta.com/crusoe/, 2004.

[80] "AMD PowerNow! Technology," 2002.

[81] "Intel XScale Technology," http://www.intel.com/design/intelxscale/, 2004.

[82] R. K. Gupta, C. N. Coelho, Jr., and G. De Micheli, "Synthesis and simulation of digital systems containing interacting hardware and software components," in *Proceedings 29th ACM/IEEE Design Automation Conference*, 1992, pp. 225-30.

[83] R. K. Gupta and G. De Micheli, "System-level synthesis using re-programmable components," in *Proceedings European Conference on Design Automation*, 1992, pp. 2-7.

[84] R. Ernst, J. Henkel, and T. Benner, "Hardware-software cosynthesis for microcontrollers," *IEEE Design & Test of Computers*, vol. 10, no. 4, 1993, pp. 64-75.

[85] J. Henkel, T. Benner, R. Ernst, W. Ye, N. Serafimov, and G. Glawe, "COSYMA: a software-oriented approach to hardware/software codesign," *Journal of Computer and Software Engineering*, vol. 2, no. 3, 1994, pp. 293-314.

[86] J. Henkel, R. Ernst, U. Holtmann, and T. Benner, "Adaptation of partitioning and high-level synthesis in hardware/software co-synthesis," in *Proceedings IEEE/ACM International Conference on Computer Aided Design*, 1994, pp. 96-100.

[87] P. Eles, Z. Peng, K. Kuchchinski, and A. Doboli, "System level hardware/software partitioning based on simulated annealing and tabu search," *Design Automation for Embedded Systems*, vol. 2, no. 1, 1997, pp. 5-32.

[88] R. P. Dick and N. K. Jha, "MOGAC: a multiobjective genetic algorithm for hardware-software cosynthesis of distributed embedded systems," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 17, no. 10, 1998, pp. 920-35.

[89] J. G. D'Ambrosio and X. Hu, "Configuration-level hardware/software partitioning for real-time embedded systems," in *Proceedings Third International Workshop on Hardware/Software Codesign*, 1994, pp. 34-41.

[90] P. V. Knudsen and J. Madsen, "PACE: a dynamic programming algorithm for hardware/software partitioning," in *Proceedings Fourth International Workshop on Hardware/Software Co-Design*, 1996, pp. 85-92.

[91] P. B. Jorgensen and J. Madsen, "Critical path driven cosynthesis for heterogeneous target architectures," in *Proceedings Fifth International Workshop on Hardware/Software Codesign*, 1997, pp. 15-9.

[92] Y.-K. Kwok and I. Ahmad, "Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 5, 1996, pp. 506-21.

[93] M.-Y. Wu and D. D. Gajski, "Hypertool: a programming aid for message-passing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 3, 1990, pp. 330-43.

[94] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing & Microprogramming*, vol. 40, no. 2-3, 1994, pp. 117-34.

[95] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," in *Proceedings First USENIX Symposium on Operating Systems Design and Implementation*, 1994, pp. 13-23.

[96] K. Govil, E. Chan, and H. Wasserman, "Comparing algorithms for dynamic speed-setting of a low-power CPU," in *Proceedings International Conference on Mobile Computing and Networking*, 1995, pp. 13-25.

[97] T. Pering, T. Burd, and R. Brodersen, "The simulation and evaluation of dynamic voltage scaling algorithms," in *Proceedings International Symposium on Low Power Electronics and Design*, 1998, pp. 76-81.

[98] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," in *Proceedings Design Automation Conference*, 1999, pp. 134-9.

[99] Y. Shin, K. Choi, and T. Sakurai, "Power optimization of real-time embedded systems on variable speed processors," in *Proceedings IEEE/ACM International Conference on Computer Aided Design*, 2000, pp. 365-8.

[100] S. Lee and T. Sakurai, "Run-time voltage hopping for low-power real-time systems," in *Proceedings Design Automation Conference*, 2000, pp. 806-9.

[101] E. Zitzler, J. Teich, and S. S. Bhattacharyya, "Optimizing the efficiency of parameterized local search within global search: a preliminary study," in *Proceedings Congress on Evolutionary Computation*, 2000, pp. 365-72.

[102] N. K. Bambha, S. S. Bhattacharyya, J. Teich, and E. Zitzler, "Hybrid global/local search strategies for dynamic voltage scaling in embedded multiprocessors," in *Proceedings Ninth International Symposium on Hardware/Software Codesign*, 2001, pp. 243-8.

[103] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles, "Synthesizing energy-efficient embedded systems with LOPOCOS," *Design Automation for Embedded Systems*, vol. 6, no. 4, 2002, pp. 401-24.

[104] M. C. Johnson, D. Somasekhar, and K. Roy, "Models and algorithms for bounds on leakage in CMOS circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 6, 1999, pp. 714-725.

[105] S. Dropsho, V. Kursun, D. H. Albonesi, S. Dwarkadas, and E. G. Friedman, "Managing static leakage energy in microprocessor functional units," in *Proceedings International Symposium on Microarchitecture*, 2002, pp. 321-32.

[106] D. Duarte, Y.-F. Tsai, N. Vijaykrishnan, and M. J. Irwin, "Evaluating run-time techniques for leakage power reduction," in *Proceedings 15th International Conference on VLSI Design*, 2002, pp. 31-8.

[107] Y.-F. Tsai, D. Duarte, N. Vijaykrishnan, and M. J. Irwin, "Implications of technology scaling on leakage reduction techniques," in *Proceedings Design Automation Conference*, 2003, pp. 187-190.

[108] Z. Chen, M. Johnson, L. Wei, and K. Roy, "Estimation of standby leakage power in CMOS circuits considering accurate modeling of transistor stacks," in *Proceedings International Symposium on Low Power Electronics and Design*, 1998, pp. 239-244.

[109] Y. Ye, S. Borkar, and V. De, "A new technique for standby leakage reduction in high-performance circuits," in *Proceedings Symposium on VLSI Circuits*, 1998, pp. 40-1.

[110] W. Zhang, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, D. Duarte, and Y.-E. Tsai, "Exploiting VLIW schedule slacks for dynamic and leakage energy

reduction," in *Proceedings International Symposium on Microarchitecture*, 2001, pp. 102-113.

[111] S. i. Mutoh, T. Douseki, Y. Matsuya, T. Aoki, S. Shigematsu, and J. Yamada, "1-V power supply high-speed digital circuit technology with multithreshold-voltage CMOS," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 8, 1995, pp. 847-854.

[112] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, "Gated-V_{dd}: A circuit technique to reduce leakage in deep-submicron cache memories," in *Proceedings International Symposium on low Power Electronics and Design*, 2000, pp. 90-95.

[113] F. Assaderaghi, D. Sinitsky, S. A. Parke, J. Bokor, P. K. Ko, and C. Hu, "Dynamic threshold-voltage MOSFET (DTMOS) for ultra-low voltage VLSI," *IEEE Transactions on Electron Devices*, vol. 44, no. 3, 1997, pp. 414-422.

[114] L. Wei, Z. Chen, M. Johnson, K. Roy, and V. De, "Design and optimization of low voltage high performance dual threshold CMOS circuits," in *Proceedings Design Automation Conference*, 1998, pp. 489-494.

[115] S. M. Martin, K. Flautner, T. Mudge, and D. Blaauw, "Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads," in *Proceedings International Conference on Computer Aided Design*, 2002, pp. 721-5.

[116] A. Doboli and P. Eles, "Scheduling under data and control dependencies for heterogeneous architectures," in *Proceedings IEEE International Conference on Computer Design*, 1998, pp. 602-608.

[117] K. Kuchcinski, "Embedded system synthesis by timing constraints solving," in *Proceedings Tenth International Symposium on System Synthesis*, 1997, pp. 50-57.

[118] S. Chakraborty, T. Erlebach, S. Kunzli, and L. Thiele, "Schedulability of event-driven code blocks in real-time embedded systems," in *Proceedings Design Automation Conference*, 2002, pp. 616-21.

[119] Y. Xie and W. Wolf, "Allocation and scheduling of conditional task graph in hardware/software co-synthesis," in *Proceedings Design, Automation and Test in Europe*, 2001, pp. 620-5.

[120] R. B. Ortega and G. Borriello, "Communication synthesis for embedded systems with global considerations," in *Proceedings Fifth International Workshop on Hardware/Software Codesign*, 1997, pp. 69-73.

[121] R. B. Ortega and G. Borriello, "Communication synthesis for distributed embedded systems," in *Proceedings IEEE/ACM International Conference on Computer Aided Design*, 1998, pp. 437-44.

[122] P. V. Knudsen and J. Madsen, "Integrating communication protocol selection with hardware/software codesign," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 18, no. 8, 1999, pp. 1077-95.

[123] P. Pop, P. Eles, and Z. Peng, "Scheduling with optimized communication for time-triggered embedded systems," in *Proceedings Seventh International Workshop on Hardware/Software Codesign*, 1999, pp. 178-82.

[124] M. R. Stan and W. P. Burleson, "Bus-invert coding for low-power I/O," *IEEE Transactions on Very Large Scale Integration VLSI Systems*, vol. 3, no. 1, 1995, pp. 49-58.

[125] L. Benini, G. De-Micheli, E. Macii, D. Sciuto, and C. Silvano, "Address bus encoding techniques for system-level power optimization," in *Proceedings Design, Automation and Test in Europe*, 1998, pp. 861-6.

[126] D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*, Addison Wesley Professional, 1989.

[127] T. Baeck, U. Hammel, and H.-P. Schwefel, "Evolutionary computation: Comments on the history and current state," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, 1997, pp. 3-17.

[128] M. Palesi and T. Givargis, "Multi-objective design space exploration using genetic algorithms," in *Proceedings Tenth International Symposium on Hardware/Software Codesign*, 2002, pp. 67-72.

[129] D. Sciuto, F. Salice, L. Pomante, and W. Fornaciari, "Metrics for design space exploration of heterogeneous multiprocessor embedded systems," in *Proceedings Tenth International Symposium on Hardware/Software Codesign*, 2002, pp. 55-60.

[130] C. Brandolese, W. Fornaciari, F. Salice, and D. Sciuto, "Energy estimation for 32-bit microprocessors," in *Proceedings Eighth International Workshop on Hardware/Software Codesign*, 2000, pp. 24-8.

[131] Y.-T. S. Li and S. Malik, "Performance analysis of embedded software using implicit path enumeration," in *Proceedings Design Automation Conference*, 1995, pp. 456-61.

[132] T. Y. Yen and W. Wolf, "Performance estimation for real-time distributed embedded systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 11, 1998, pp. 1125-36.

[133] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, no. 4, 1994, pp. 437-45.

[134] V. Tiwari and M. T.-C. Lee, "Power analysis of a 32-bit embedded microcontroller," in *Proceedings Asia and South Pacific Design Automation Conference*, 1995, pp. 141-8.

[135] P. Pop, *Scheduling and communication synthesis for distributed real-time systems*, Licentiate thesis, Linkopings University, 2000.

[136] J. Liu, P. H. Chou, and N. Bagherzadeh, "Communication speed selection for embedded systems with networked voltage-scalable processors," in *Proceedings Tenth International Symposium on Hardware/Software Codesign*, 2002, pp. 169-174.

[137] H. Kopetz and G. Grunsteidl, "TTP - a protocol for fault-tolerant real-time systems," *Computer*, vol. 27, no. 1, 1994, pp. 14-23.

[138] J. Luo, L.-S. Peh, and N. Jha, "Simultaneous dynamic voltage scaling of processors and communication links in real-time distributed embedded systems," in *Proceedings Design Automation and Test in Europe*, 2003, pp. 1150-1.

[139] A. Andrei, M. Schmitz, P. Eles, Z. Peng, and B. M. Al-Hashimi, "Simultaneous Communication and Processor Voltage Scaling for Dynamic and Leakage Energy Reduction in Time-Constrained Systems," in *Proceedings International Conference on Computer Aided Design*, 2004.

[140] W. Stallings, *Data & Computer Communications*, Prentice-Hall Inc., 2000.

[141] T. L. Adam, K. M. Chandy, and J. R. Dickson, "A comparison of list schedules for parallel processing systems," *Communications of the ACM*, vol. 17, no. 12, 1974, pp. 685-90.

[142] G. C. Sih and E. A. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 2, 1993, pp. 175-87.

[143] "GSM 06.10," http://kbs.cs.tu-berlin.de/~jutta/toast.html.

[144] L. Yan, J. Luo, and N. K. Jha, "Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Heterogeneous Distributed Real-time Embedded Systems," in *Proceedings International Conference on Computer Aided Design*, 2003, pp. 30-37.

[145] A. Andrei, M. T. Schmitz, P. Eles, Z. Peng, and B. M. Al-Hashimi, "Overhead-conscious voltage selection for dynamic and leakage energy rduction of time-constrained systems," in *Proceedings Design, Automation and Test in Europe*, 2004, pp. 518-523.

[146] "International technology roadmap for semiconductors," http://public.itrs.net/, 2003.

[147] J. Ahmed and C. Chakrabarti, "A dynamic task scheduling algorithm for battery powered DVS systems," in *Proceedings International Symposium on Cirquits and Systems*, 2004, pp. 813-816.

[148] "The LEDA user manual, Version 4.1," http://itp.nat.uni-magdeburg.de/docs/MANUAL/MANUAL.html.

[149] "GAlib - a C++ library of genetic algorithm components," http://lancet.mit.edu/ga/.