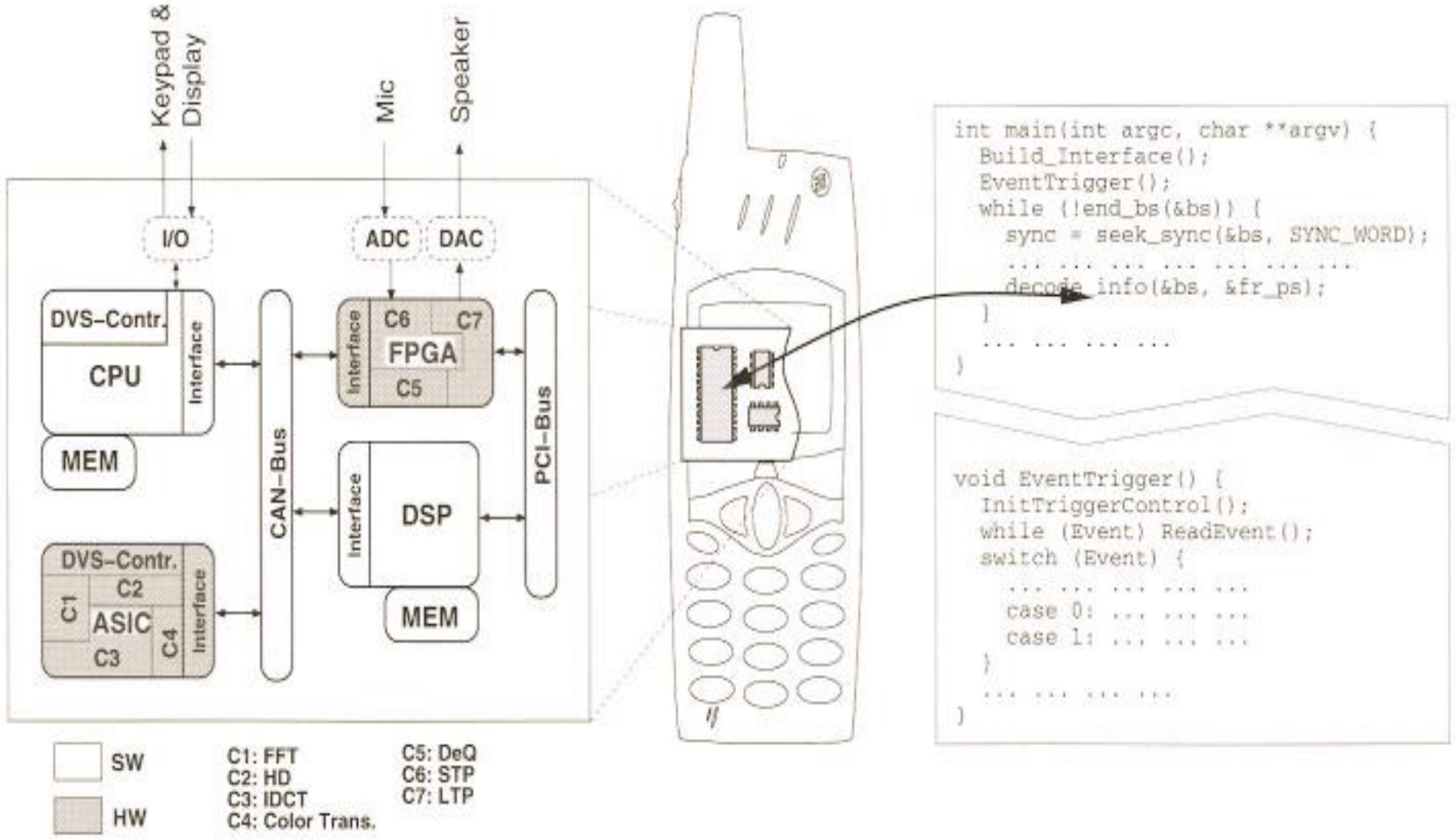
System Level Design For Low Power

Yard. Doç. Dr. Berna Örs Yalçın

References

- Marcus T. Schmitz, Bashir M. Al-Hashimi, Petru Eles, “System-Level Design Techniques for Energy-Efficient Embedded Systems”, 2004, 211 p., Springer.



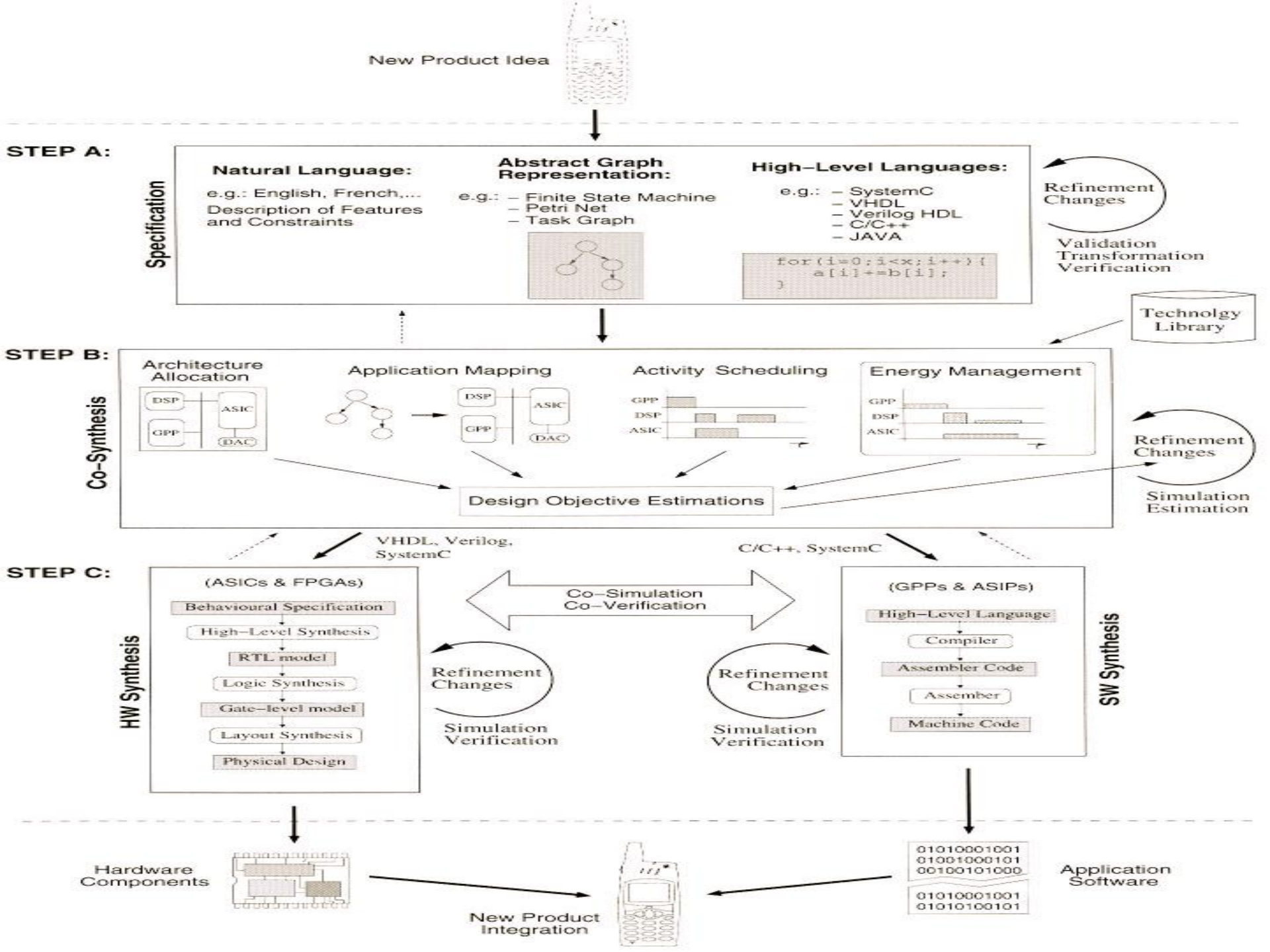
(a) Embedded architecture: A distributed heterogeneous system

(b) Embedded software

Figure 1.1. Example of a typical embedded system (smart-phone)

Design Flow

- Step A: system specification
- Step B: co-synthesis
  - Goal
    - Architecture allocation
    - Application mapping
    - Activity scheduling
    - Energy management
  - Aim : to optimise the design
  - Objectives
    - Power consumption
    - Performance
    - Cost
- Step C: hardware and software synthesis


New Product Idea
STEP A:
Specification
Natural Language:
e.g.: English, French,...
Description of Features and Constraints
Abstract Graph Representation:
e.g.: – Finite State Machine
– Petri Net
– Task Graph
High–Level Languages:
e.g.: – SystemC
– VHDL
– Verilog HDL
– C/C++
– JAVA
for(i=0;i<x;i++){
a[i]+=b[i];
}
Refinement Changes
Validation
Transformation
Verification
Technolgy Library
STEP B:
Co–Synthesis
Architecture Allocation
DSP
GPP
ASIC
DAC
Application Mapping
Activity Scheduling
GPP
DSP
ASIC
Energy Management
Design Objective Estimations
Refinement Changes
Simulation
Estimation
VHDL, Verilog, SystemC
C/C++, SystemC
STEP C:
(ASICs & FPGAs)
Behavioural Specification
High–Level Synthesis
RTL model
Logic Synthesis
Gate–level model
Layout Synthesis
Physical Design
HW Synthesis
Co–Simulation
Co–Verification
Refinement Changes
Simulation Verification
Refinement Changes
Simulation Verification
(GPPs & ASIPs)
High–Level Language
Compiler
Assembler Code
Assembler
Machine Code
SW Synthesis
Hardware Components
New Product Integration
01010001001
01001000101
00100101000
01010001001
01010100101
Application Software

Task Graph Representation

$$G_S = (\mathrm{T}, C)$$

$T = \{\tau_0, \tau_1, \cdots, \tau_n\}$: the set of tasks to be executed

C: the set of directed edges refers to communications between tasks

$\gamma_{ij} \in C$: a communication from task τ_i to task τ_j

- A task can only start its execution after all its ingoing communications have finished.
- Task deadline θ*:* the time by which its execution has to be finished.

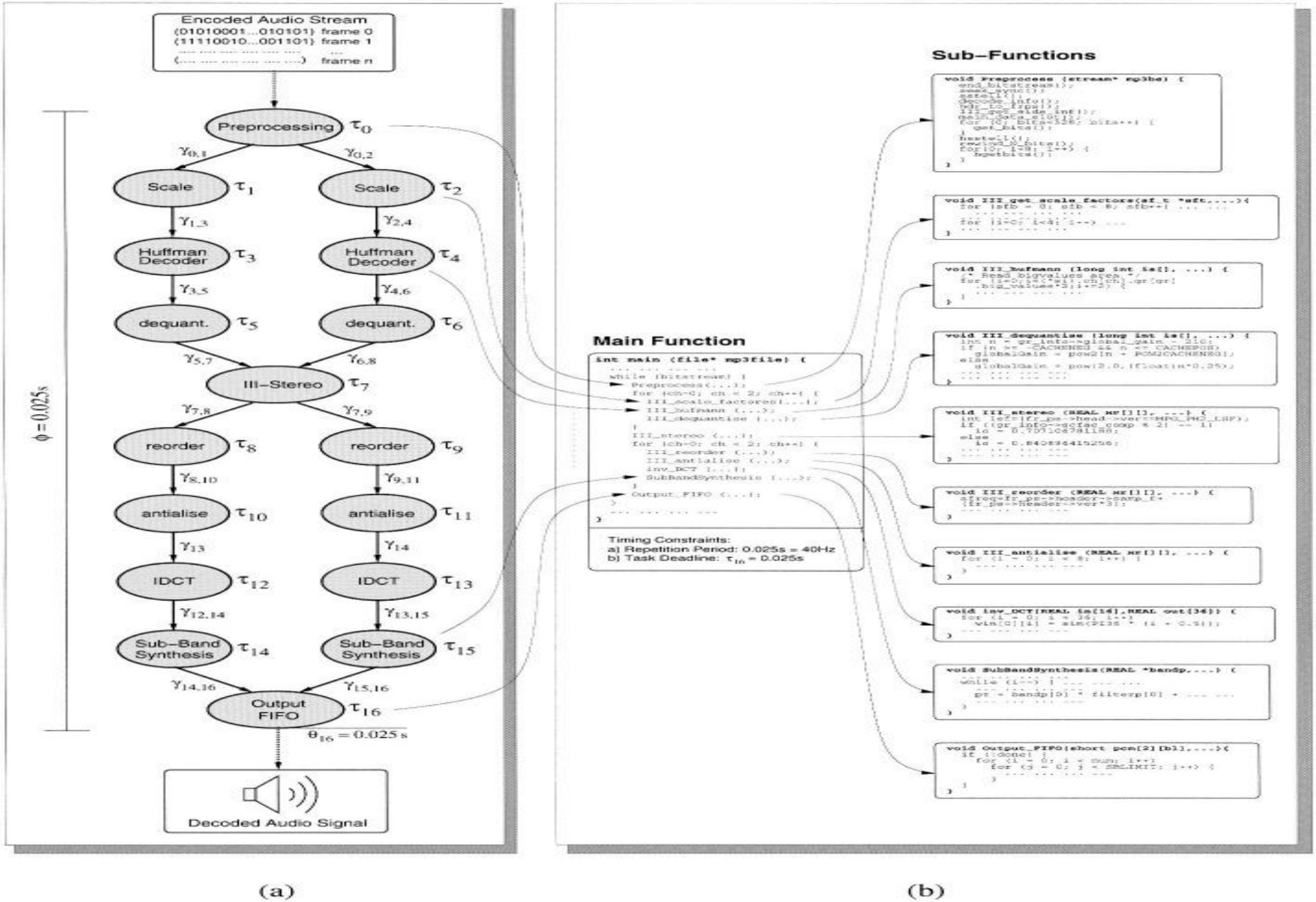
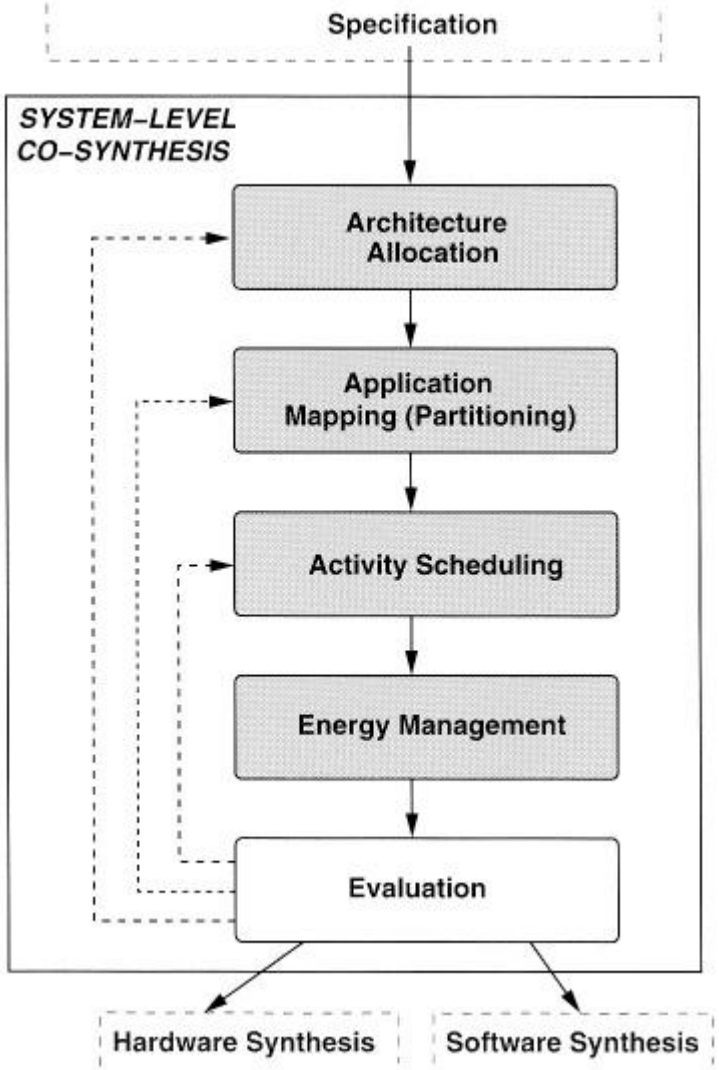


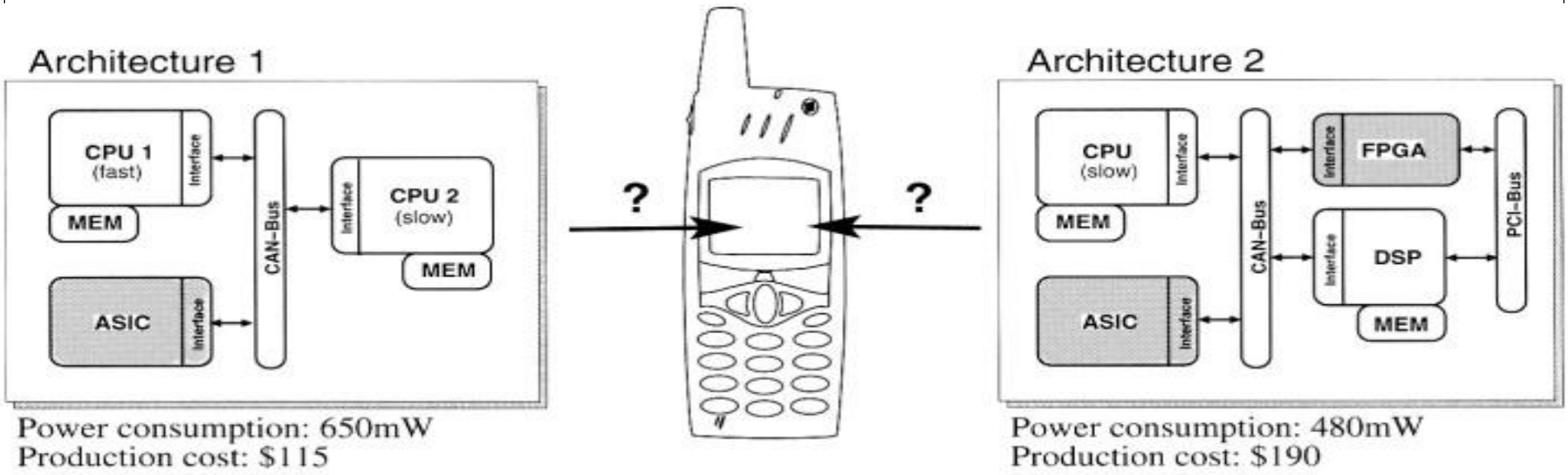
Figure 1.3. MP3 decoder given as (a) task graph specification (17 tasks and 18 communications) and (b) high-level language description in C

System-level co-synthesis flow



Co-synthesis steps are iteratively repeated until all design constraints and objectives are satisfied.

Architectural selection problem

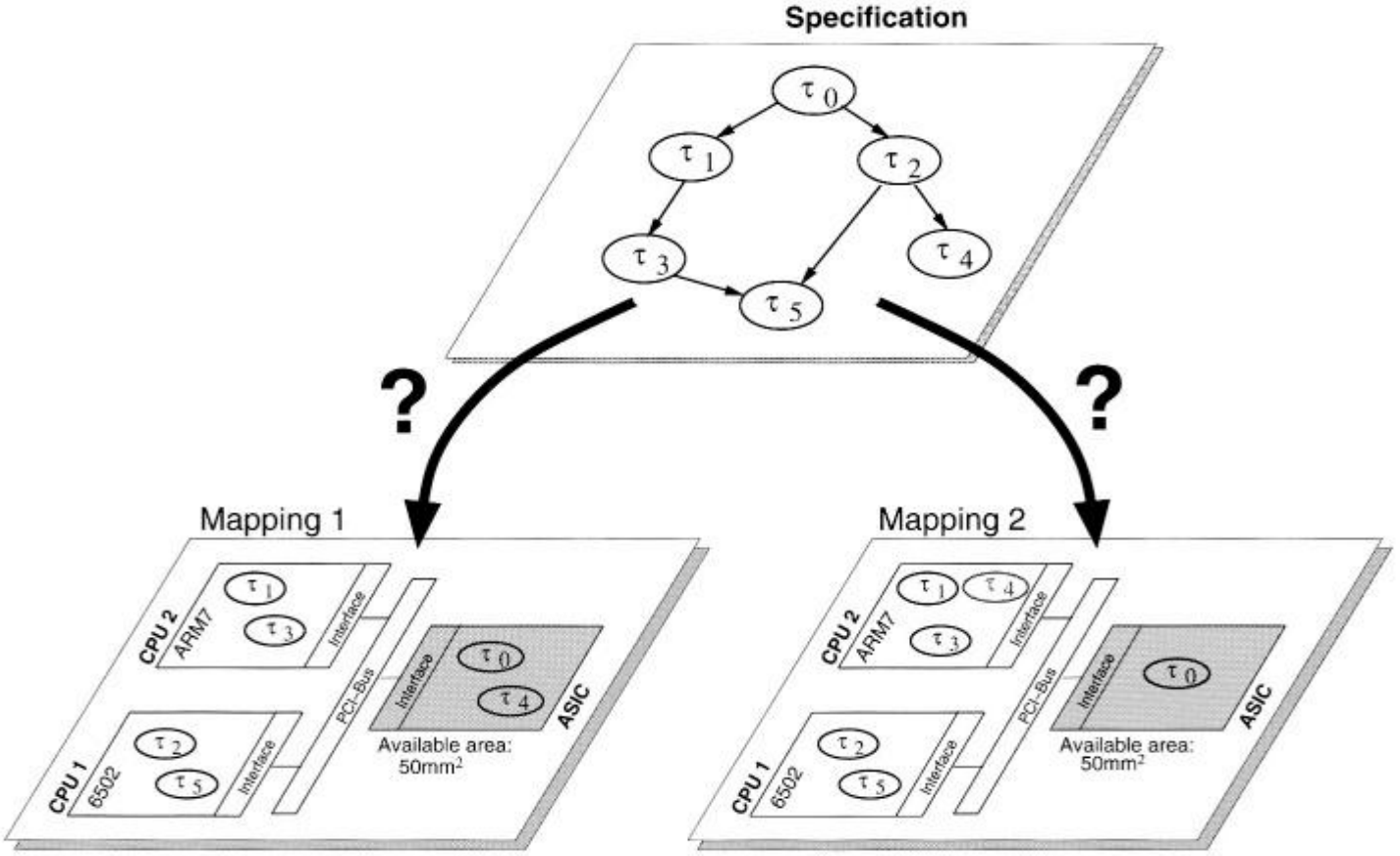


	GPPs	ASIPs	FPGAs	ASICs
Cost	+ +	+ +	o	- -
Flexibility	+ +	+	o	- -
Performance	-	o	+	+ +
Energy-Efficiency	-	o	+	+ +

Table 1.1. Trade-offs between serveral heterogeneous components
(+ + highly advantageous, + advantageous, o moderate,
disadvantageous, - - highly disadvantageous)

- GPP: general-purpose processor
- ASIC: application
- specific integrated circuit
- ASIP: application specific instruction set processor
- FPGA: field-programmable
- gate arrays

Application mapping onto hardware and software components

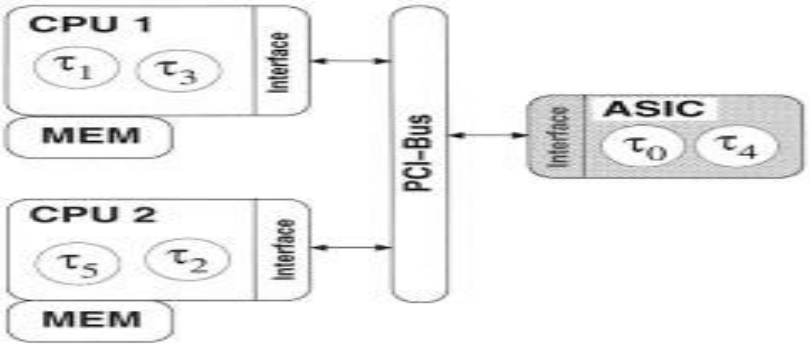


- Mapping explicitly determines if a task is implemented in hardware or software
- *hardware/software partitioning* is often mentioned in this context

Task execution properties (time and power) on different processing elements

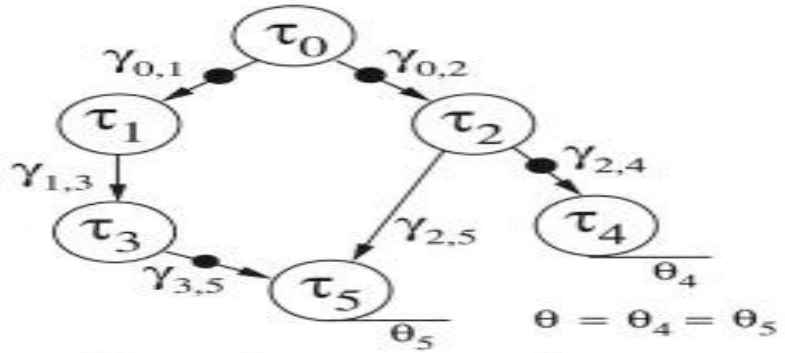
Task	**CPU 1** (6502 @$10MHz$)		**CPU 2** (ARM7 @$20MHz$)		**ASIC** ($50mm^2$, Technology: $0.6\mu m$)		
	t_{exe} (ms)	P_{dyn} (mW)	t_{exe} (ms)	P_{dyn} (mW)	t_{exe} (ms)	P_{dyn} (mW)	A (mm^2)
τ_0	89.3	3.6	12.1	23	1.8	0.13	7.76
τ_1	25.2	3.9	3.0	26	0.3	0.05	5.82
τ_2	19.7	4.4	2.8	28	0.2	0.07	9.71
τ_3	31.1	3.8	4.7	28	0.4	0.02	12.52
τ_4	172.2	3.9	22.3	27	2.7	0.12	8.05
τ_5	27.2	4.2	3.5	24	0.6	0.02	3.74

Architecture and Mapping

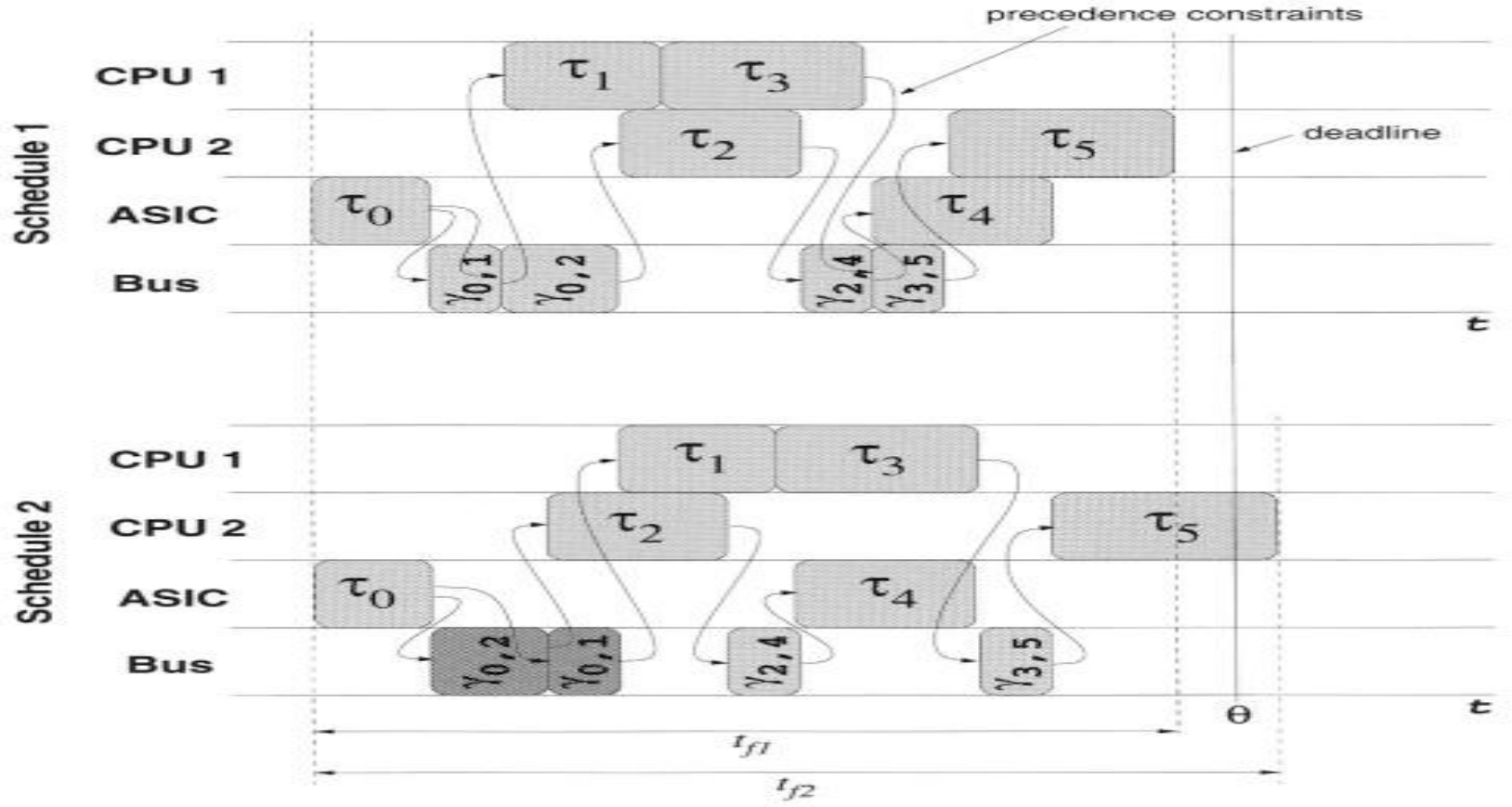


Spatial place of execution

Task Graph

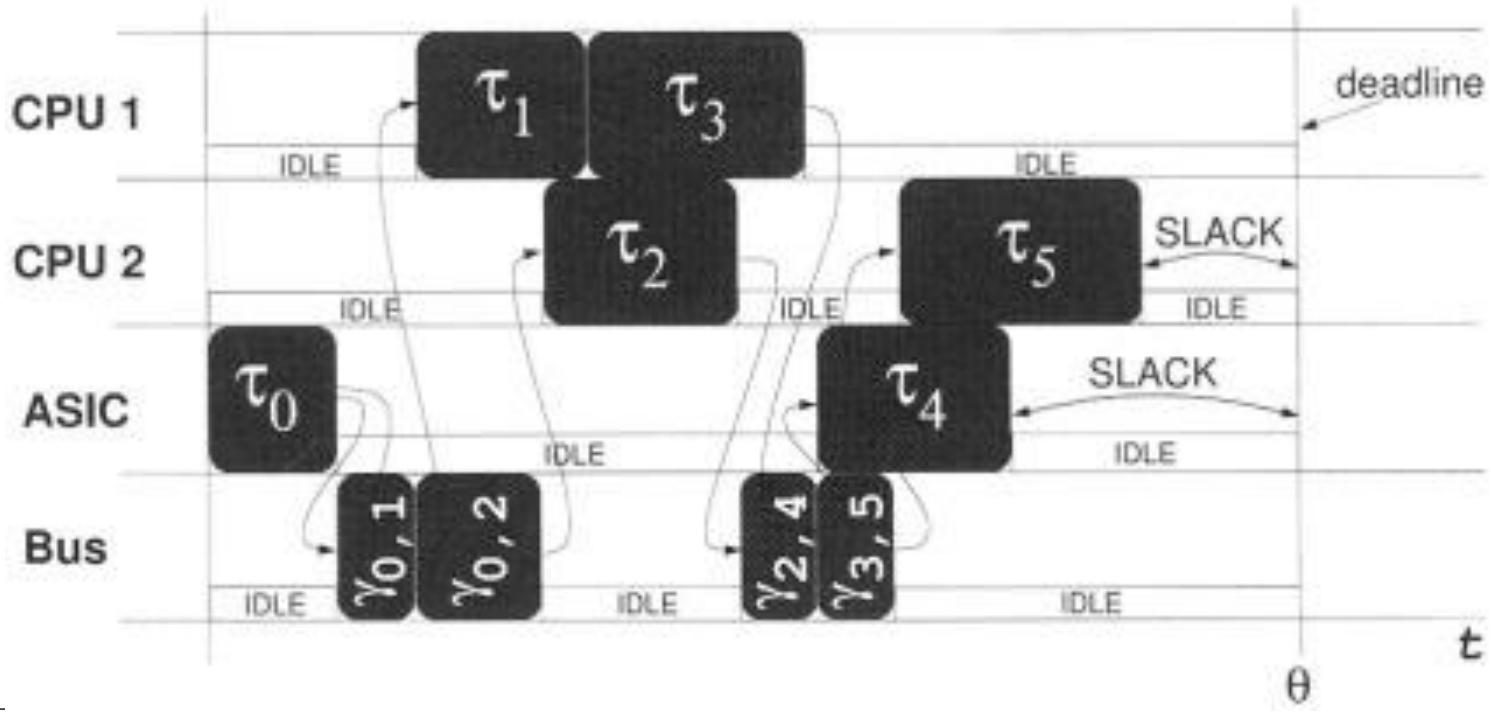


Precedence constraints
(● =Communication via bus)



Energy Management

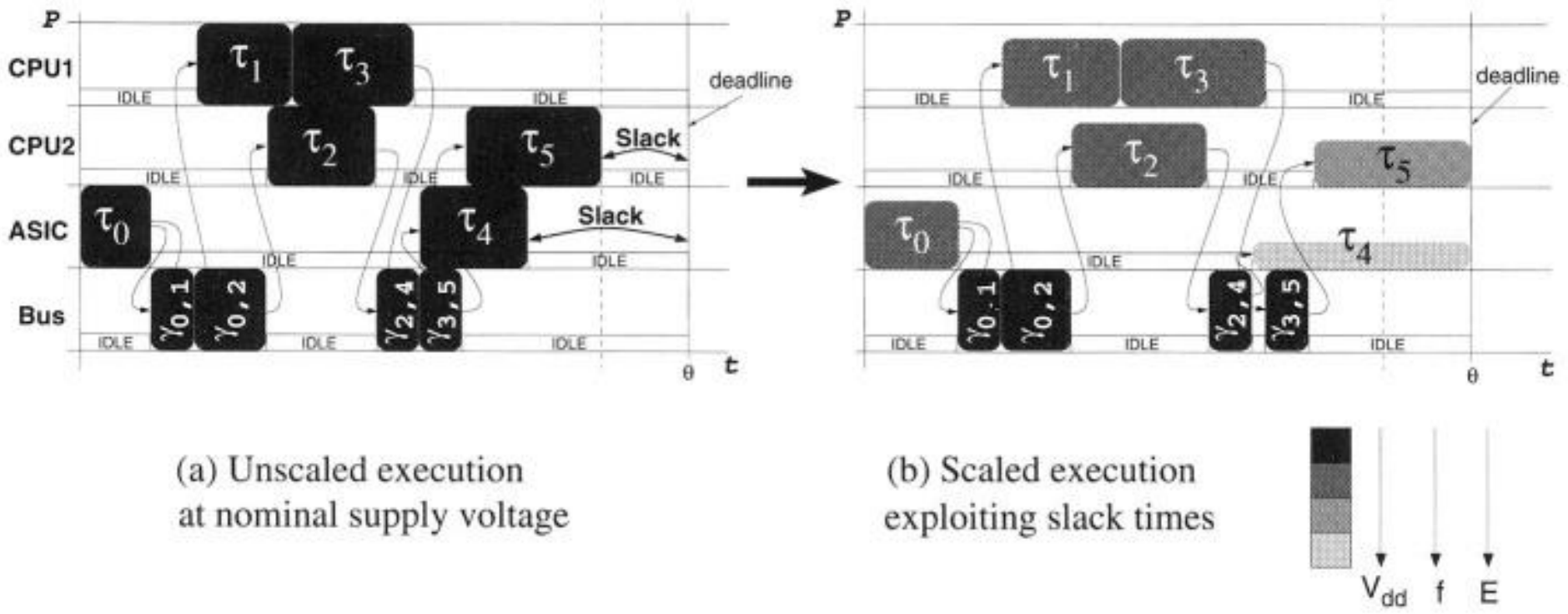
- exploit idle times and slack times within the system schedule by shutting down PEs or by reducing the performance of individual PEs
- **Idle times** refer to periods in the schedule when PEs and CLs do not experience any workload.
- **Slack times** is the difference between task deadline and task finishing time of sink tasks (tasks with no outgoing edges)



Energy Management Techniques

- $P = \alpha C_L F V^2$
- Dynamic Power Management (DPM)
  - Puts processing elements and communication links into standby or sleeping modes whenever they are idle.
  - The reactivation of components takes finite time and energy.
  - Components should only be switched off or set into a standby mode if the idle periods are long enough to avoid deadline violations or increased power consumptions .
- Dynamic Voltage Scaling (DVS)
  - Exploits slack time by reducing simultaneously clock frequency and supply voltage of PEs.
  - Adapts the component performance to the actual requirement of the system.

The concept of dynamic voltage scaling



Such an optimisation requires the iterative execution of the co-synthesis steps (allocation, mapping, scheduling), until the “most” suitable implementation of the system has been found.

Hardware Synthesis

1. *high-level synthesis tool* (or behavioural synthesis): behavioural specification → structural description at the register transfer level (RTL).
2. *logic synthesis:* RTL description → gate level representation
3. *layout synthesis:* gate level representation → final layout

- Power reduction can be addressed at all three synthesis stages
  - high-level: e.g. clock-gating
  - gate-level: e.g. logic optimisation
  - mask-level: e.g. technology choice

- The higher the level of abstraction at which the energy minimisation is addressed, the higher are the achievable energy saving.

Software Synthesis

1. *Compile:* initial specification → assembly code
   - The goal of the optimisation is the effective assignment of variables to registers such that operations can be performed without "time consuming" memory accesses.
2. *processor specific assemblers:* assembler code → executable machine code

- Power optimisation by
  - instruction reordering
  - reduction of memory accesses
  - careful algorithmic design at the source code level

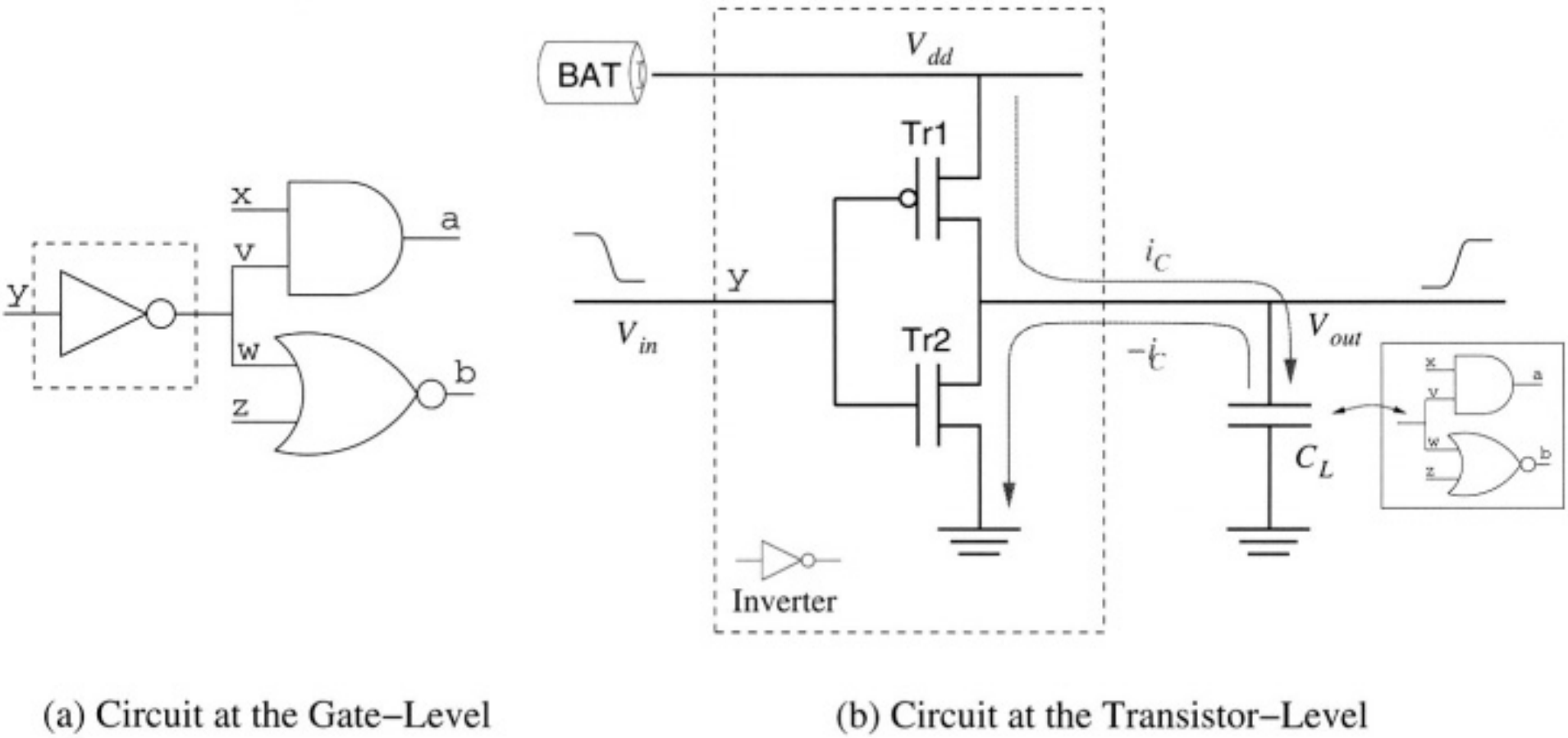
Energy Dissipation of Processing Elements

1. *Static*
   - occur whenever the processing element is switched on, even when no computations are carried out on this unit.
2. *Dynamic*
   - active computations cause switching activity within the circuitry whenever computations are performed.

- $P_E = P_{static} + P_{dynamic} = \underbrace{P_{leak} + P_{bias}}_{P_{static}} + \underbrace{P_{SC} + P_{SW}}_{P_{dynamic}}$

- Out of these four source of power dissipation, switching power P_{SW} is currently the dominant one which accounts for approximately 90% of the total PE power consumption.
- With shrinking feature size ($< 0.07\mu m$) and reduced threshold voltage levels, the leakage currents become additionally an important issue.

Dynamic power dissipation



(a) Circuit at the Gate–Level

(b) Circuit at the Transistor–Level

Switching power is P_{SW} dissipated due to the charging and discharging of the effective circuit load capacitance C_L (parasitic capacitors of the circuit gates).

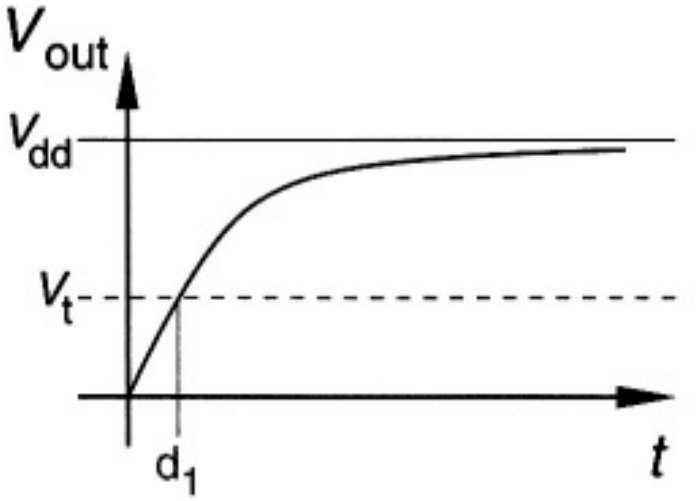
Dissipated Switching Energy of One Clock Cycle

- $P_{SW} = V_{dd} i_{C_L}$
- $i_{C_L} = C_L \frac{\partial V_{out}}{\partial t}$
- $T = \frac{1}{f}$ (period of on clock cycle) f: operational frequency
- ${E_{SW}}^{0\to1} = \int_0^T P_{SW} dt = \int_0^T V_{dd} i_{C_L} dt = V_{dd} \int_0^T C_L dV_{out} = C_L {V_{dd}}^2$
- The total energy ${E_{SW}}^{\tau}$ drawn from the batteries by a PE performing a computational task τ depends additionally on the number of clock cycles N_C needed to execute this task and the switching activity α.
- ${E_{SW}}^{\tau} = N_C \alpha C_L {V_{dd}}^2$
- ${P_{SW}}^{\tau} = \frac{N_C \alpha C_L {V_{dd}}^2}{T N_C} = \alpha C_L {V_{dd}}^2 f$

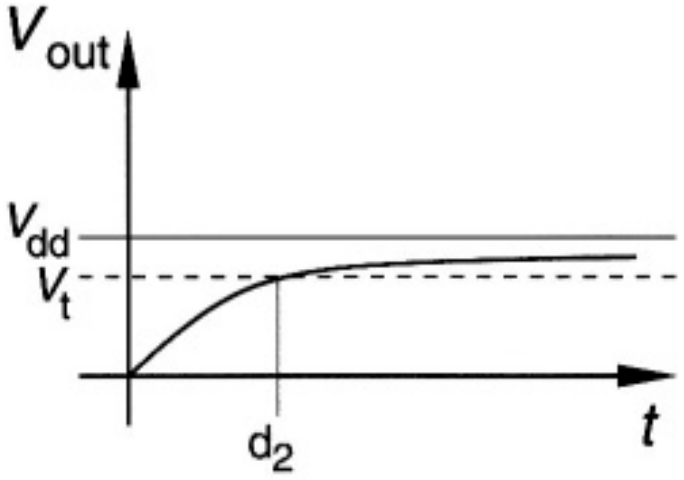
Conclusion from the Equations

- If we assume that the load capacitance C_L is a constant given by the complexity of the design and the circuit technology, and further the switching activity α is a constant depending on the computational task then:
  - Although decreasing the operational frequency f leads to a reduction in power dissipation (${P_{SW}}^{\tau} = \alpha C_L {V_{dd}}^2 f$), it does not reduce the energy dissipation (${E_{SW}}^{\tau} = N_C \alpha C_L {V_{dd}}^2$), which is important for battery-lifetime.
  - The only possibility to reduce the energy consumption is to reduce the circuit supply voltage V_{dd}.
  - Reducing the supply voltage necessitates the reduction of the frequency in order to ensure correct operation.

Supply voltage dependent circuit delay



(a) Low delay with $V_{dd}=V_{max}$



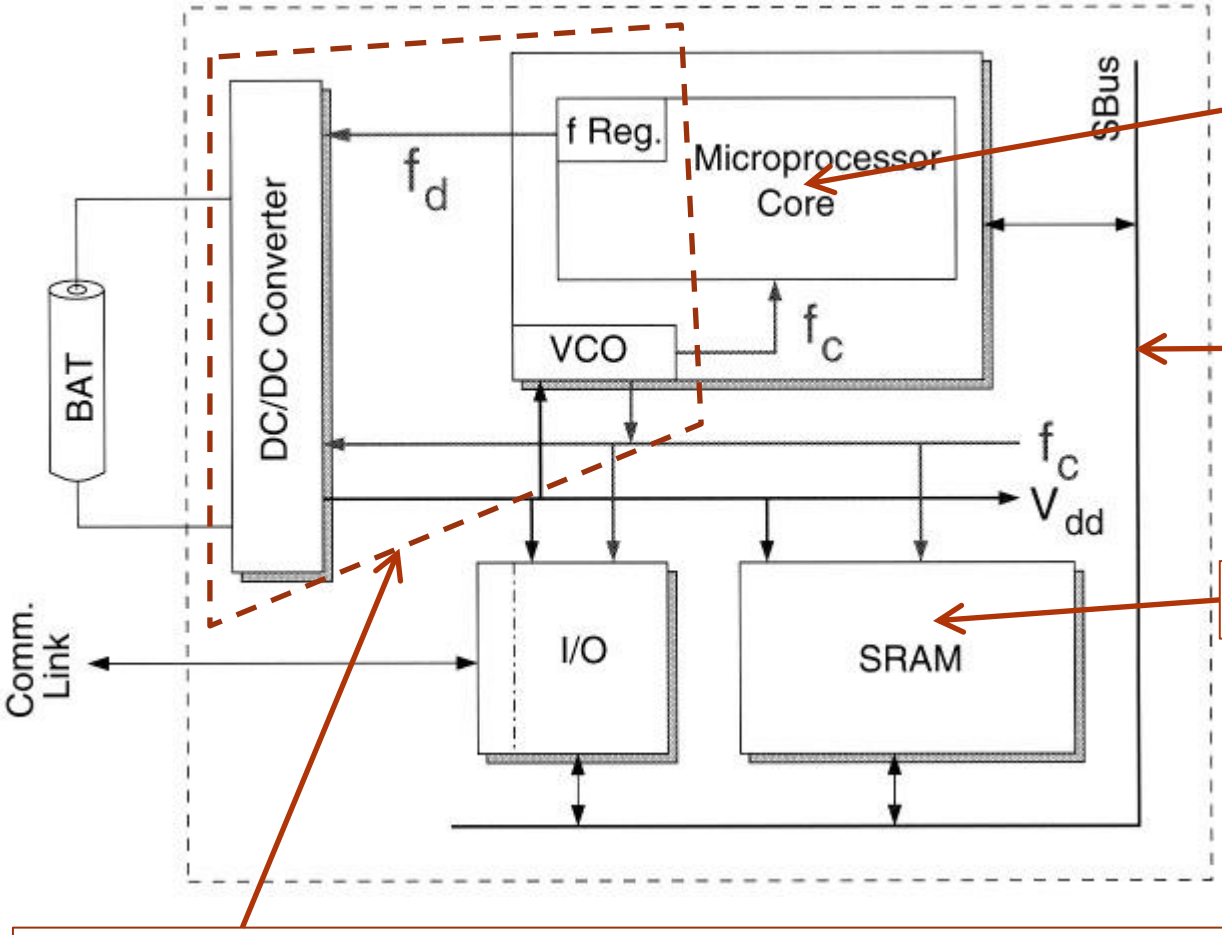
(b) Increased delay with $V_{dd}=V_{max}/2$

- The reduction of the supply voltage leads to longer charging times until the threshold is reached.
- Reducing the supply voltage also reduces the leakage power consumption.
- $P_{leak} = V_{dd} K_1 e^{K_2 V_{dd}} e^{K_3 V_{bs}} + |V_{bs}| I_{Ju}$

Dynamic Voltage Scaling

- DVS-enabled processors have the ability to dynamically change their supply voltage and operational frequency settings during run-time of the application.
- Temporal performance requirements of applications can exploit the energy/delay trade-off to reduce energy dissipation.

Block diagram of DVS-enabled processor



carries out the required computations.

system bus

static memory (cache unit)

The heart of this system, which enables a dynamic voltage selection, consists of a DC/DC voltage converter, a specialised frequency register, and a voltage controlled oscillator (VCO).

Dynamic Voltage Scaling

- Supply voltage and operational frequency are changed by writing the desired frequency f_d into the frequency register.
- Upon writing the desired frequency into the register, the DC/DC converter compares this frequency with the current frequency f_c and either increases or decreases the supply voltage V_{dd}.
- According to the changed voltage, the VCO adapts the system clock to a higher or lower frequency f_c.
- Certainly, the whole voltage scaling process requires a finite time. Typical transition times are in the range of tenths of microseconds.

Dynamic Power Management

- The main strategy is the shutdown of idle system components.
- An advantage is usage not only for digital circuitry, but also for other system components such as displays, hard drives, and analogue circuits.
- Two approaches
  - components are switched off immediately when they become idle
  - the timeout-based policy, which switches off components after a fixed idle interval.
    - well known from the advanced power management (APM) widely used in today's notebook computers.
- Since the restart of a component involves a time and power overhead to restore its fully functional state, such greedy policies might not result in power savings or may even increase the dissipated power.
- The quality of a power management policy depends on the accuracy with which the future behaviour of the system can be predicted, in order to start-up currently inactive components or to shut down currently active components at the right moment.

Dynamic Power Management versus Dynamic Voltage Scaling

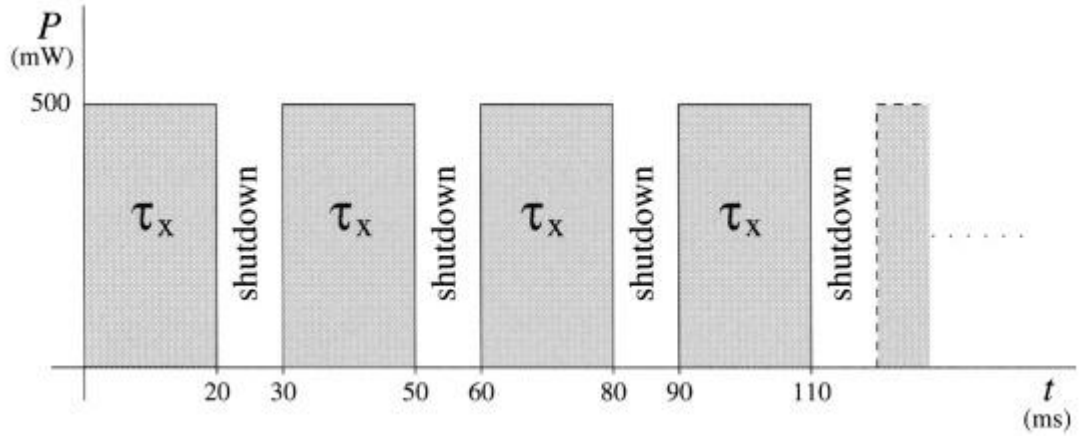
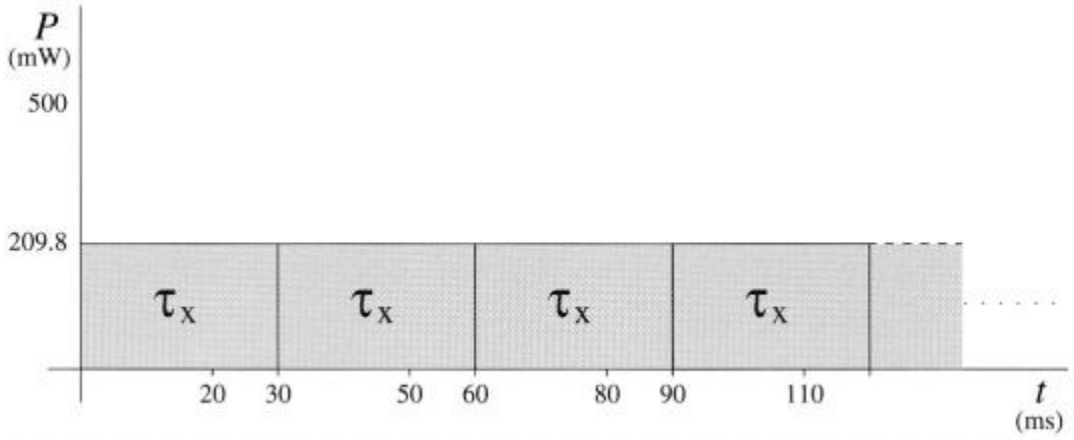


Figure 2.5. Shutdown during idle times (DPM)

- A PE performs a certain task τ_x in 20 ms and dissipates a power of 500 mW when running with 33 MHz at a nominal supply voltage of 3.3 V and a threshold voltage of 0.8 V.
- To meet the performance requirements of the application, the task needs to be repeated every 30 ms.
- Between each consecutive execution of the task there are 10 ms of idleness during which the processor can be deactivated (DPM).
- Under these circumstances the execution of task τ_x results in an energy consumption of 500mW·20ms=10mJ.

Voltage scaling to exploit the slack time (DVS)



- The repetition time of 30*ms* allow to prolong the execution time of task from 20*ms* to 30*ms*.
- Instead of shutting down the processor during the 10*ms* of idleness, the processor's performance is reduced from 33*MHz* to 22*MHz*, since $30ms/20ms = 1.5$ and $33MHz/1.5 = 22MHz$.
- *T*he lower frequency allows the reduction of the supply voltage from 3.3*V* to 2.61*V* and the tolerable increase in circuit delay of $33MHz/22MHz = 1.5$.
- The exact voltage can be obtained by considering the ratio between the delays at 33*MHz* and 22*MHz*.
- At this voltage and frequency values the processing element dissipates a power of 209.8*mW*.
- The energy consumption is given by $209.8\text{mW} \cdot 30\text{ms} = 6.29\text{mJ}$, a reduction of 32.1% compared to the 10mJ dissipated when using DPM.
- This simple example has shown that the energy efficiency of DVS is superior when compared to DPM.
- In fact, DVS always performs better than DPM, whenever both techniques are applicable.

Energy Dissipation of Communication Links

- Fast communication is essential to avoid undesired contention of processing elements.
  - Wide system buses (8 to 128 bit)
  - High speed serial buses (CAN bus, I^2C bus, USB, Gigabit Ethernet, Firewire, etc.) have become commonplace.
- With each transfer of data over the communication links (CLs), the line capacitance is charged and discharged, drawing a current from the I/O pins of the processing elements.
- The power dissipated by these currents is given by:

$$P_{CL} = \beta C_{bus} f_{bus} V_{tr}^2$$

- The drawn energy:

$$E_{CL}^{\gamma} = N_{tr}^{\gamma} \beta C_{bus} f_{bus} V_{tr}^2$$

Energy Dissipation of Communication Links

- One possibility to reduce this energy dissipation is to encode the data before transferring it.
- Such bus-encoding techniques have been investigated with the aim of reducing the switching activity β.
- Unlike the supply voltage of processing elements, the transmission voltage V_{tr} cannot be reduced easily due to reliability issues, that is, environmental noise potentially corrupts data during transfers at low voltages.
- Furthermore, DPM can be equally applied to CLs as to PEs, i.e., during intervals where no data is transfered the CLs can be switched off.

Power Variation Driven DVS

- Many approaches to DVS in distributed systems assume that the power dissipation of processing elements is independent of the executed instructions.
- The voltage selection is carried out considering a constant power dissipation, in the following referred to as *fixed power model*.
- Modern IC designs often make use of low-level power minimisation techniques, such as clock gating to stop the switching activity in unutilised blocks of the circuit.
- The power dissipation can vary considerably during the execution of different functions.
- A voltage scaling technique that takes this power variation effect into account.
  - An energy-gradient driven heuristic
  - The concept of mapped-and-scheduled task graphs is used to account for task and communication dependencies in a fast and effective way.

Motivation

- In the case of a general-purpose processor (GPP), including an integer unit and a floating point unit (FPU), it is not desirable to keep the FPU active if only integer instructions are executed.
  - The clock signal to the FPU can be gated (stopped) during the execution of integer instructions, which will nullify the switching activity in the FPU.
  - Different tasks (different use of instructions) dissipate different amounts of power on the same processing element.
- Several different cores reside together on a single chip. Consider an ASIC accommodating four different cores: a FIR filter, an IDCT algorithm, a DES encrypt/decrypt unit, CORDIC (coordinate rotation digital computing) algorithm.
  - These four cores vary considerably in complexity.
  - This heterogeneity results certainly in different power dissipations, accordingly to which cores are active at a given time.
- Taking this power variations into account during the voltage scaling improves the overall energy efficiency, since the available slack time is distributed more fairly among the tasks.

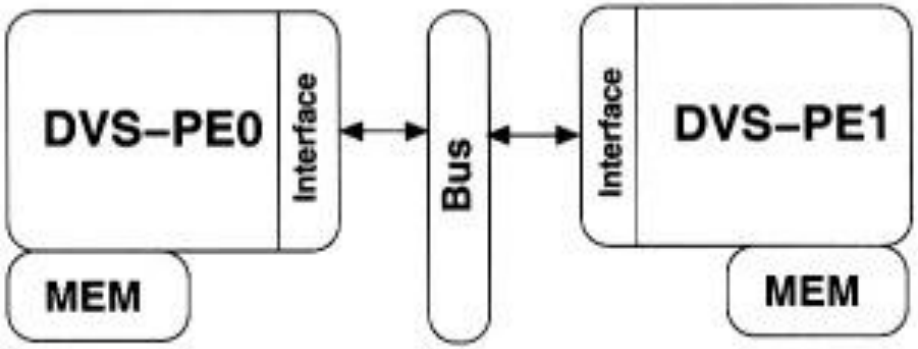
Energy-gradient

- ΔE_τ: Difference between the energy dissipation of task τ with the execution time t_{exe} and the reduced energy dissipation (due to voltage and clock scaling) of the same task when extended by a time quantum Δt.

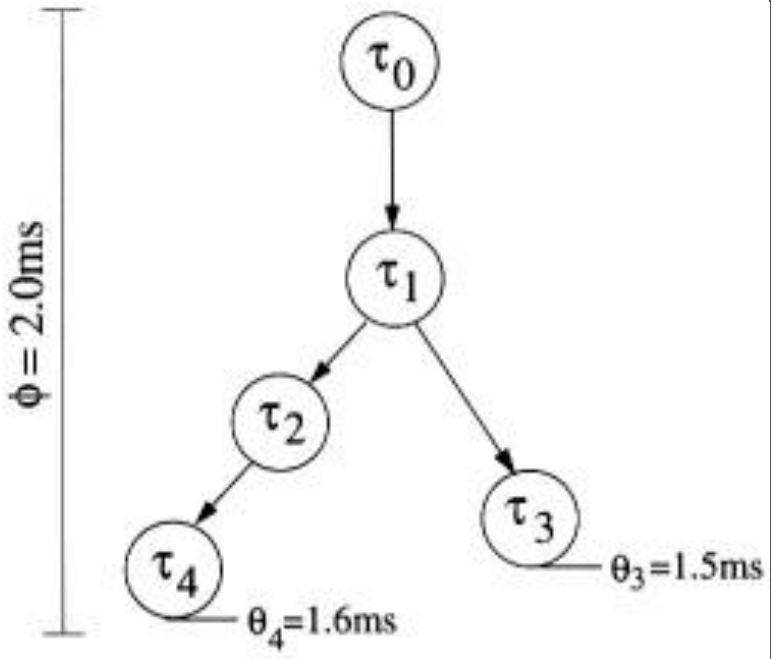
$$\Delta E_\tau = E_\tau(t_{exe}) - E_\tau(t_{exe} + \Delta t).$$

Motivational Example: Considering Power Variations during Voltage Scaling

- The considered architecture is composed of two heterogeneous DVS-PEs
  - a Transmeta Crusoe
  - a StrongARM with Xscale technology



(a) Block diagram of the dynamic voltage scalable target architecture



(b) Task graph specification including period and deadlines

	PE0 ($V_{max} = 5V, V_t = 1.2V$)		PE1 ($V_{max} = 3.3V, V_t = 0.8V$)	
task	*exe. time* (ms)	*power* (mW)	*exe. time* (ms)	*power* (mW)
τ_0	0.15	85	0.70	30
τ_1	0.40	90	0.30	20
τ_2	0.10	75	0.75	15
τ_3	0.10	50	0.15	80
τ_4	0.15	100	0.20	60

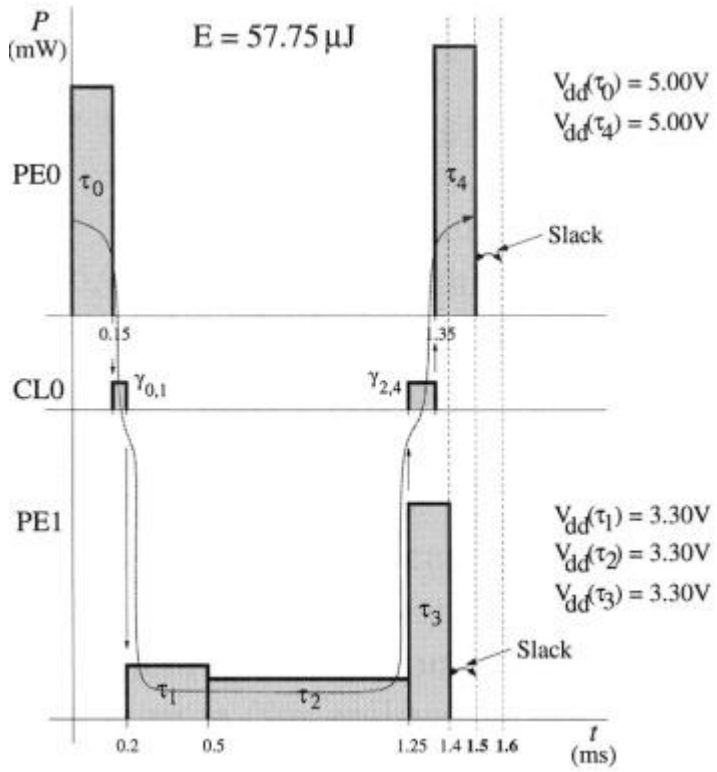
Table 3.1. Nominal task execution times and power dissipations

comm.	*comm. time* (ms)	*power dis.* (mW)
$\gamma_{0\to1}$	0.05	5
$\gamma_{1\to2}$	0.05	5
$\gamma_{1\to3}$	0.15	5
$\gamma_{2\to4}$	0.10	5

Table 3.2. Communication times and power dissipations of communication activities mapped to the bus

Communications between tasks on the same PE (intra communications) are assumed to be instantaneous, and their power dissipation is neglected.

Power profile of a possible mapping and schedule at nominal supply voltage (no DVS is applied)



- The dynamic system energy dissipation of this configuration at nominal supply voltage can be calculated using the dynamic power values and execution times given in Tables 3.1 and 3.2.
- PE0 (τ_0 and τ_4) $\rightarrow 0.15ms \cdot 85mW + 0.15ms \cdot 100mW = 27.75\mu J$
- PE1 (τ_1 , τ_2 , τ_3) $\rightarrow$ $0.3ms \cdot 20mW + 0.75ms \cdot 15mW + 0.15ms.80mW = 29.25\mu J$
- CL0 $\rightarrow 0.05ms \cdot 5mW + 0.10ms \cdot 5mW = 0.75\mu J$
- overall energy dissipation $\rightarrow$ $27.75\mu J + 29.25\mu J + 0.75\mu J = 57.75\mu J$.
- τ_3 finishes at $1.4ms$, deadline : $1.5ms$, slack : $0.1ms$.
- τ_4 finishes at $1.5ms$, deadline : $1.6ms$, slack : $0.1ms$.

A fixed power model : power variations are neglected

- Distribute the slack time evenly among the tasks, that is, each task is "stretched" using the same extension factor.
- Each task execution is extended using a factor of e=1.074
- The execution of task τ_0 was extended to $0.161ms$ ($0.15ms \cdot 1.074$).
- The extension factor can be calculated considering the longest path to the task with the smallest slack time.
- Both deadline tasks (τ_3 and τ_4) have a slack of $0.1ms$.
- Consider the path indicated, which involves the tasks τ_0, τ_1, τ_2 and τ_4.
- The extension factor is given by:

$$e = \frac{\left(\sum_{\tau \in T_P} t_{nom}(\tau)\right) + t_S}{\sum_{\tau \in T_P} t_{nom}(\tau)}$$

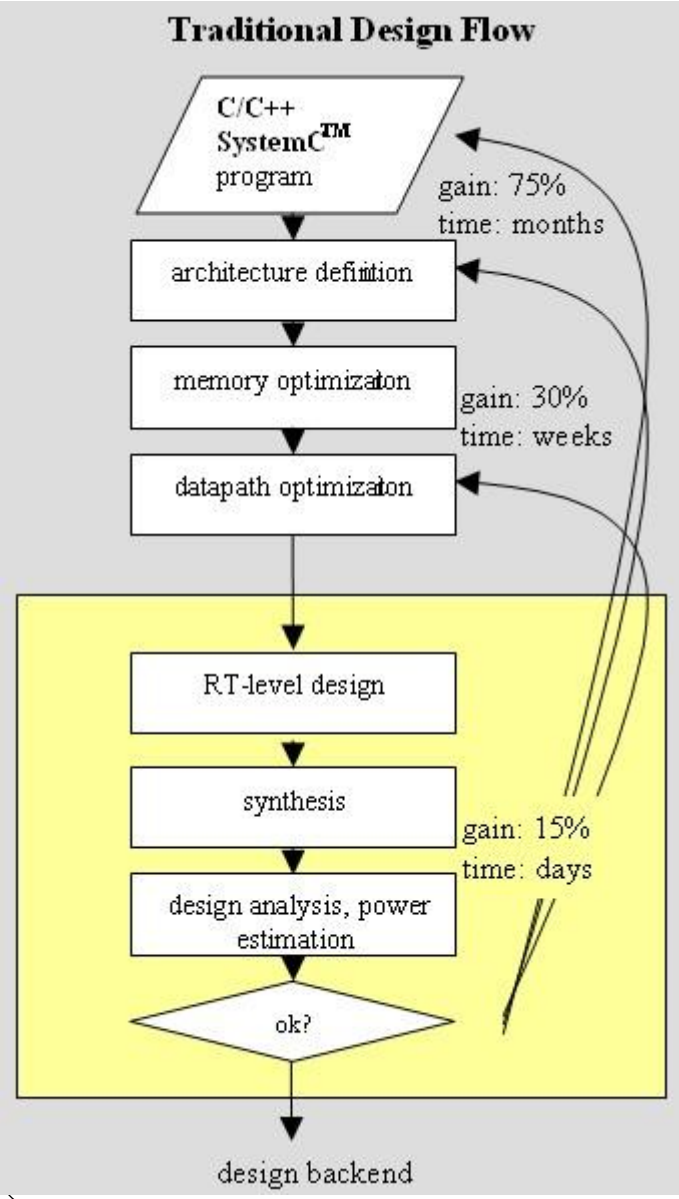
- t_{nom} : nominal execution time, t_S : slack time, T_P : all tasks on the path
- Communications are neglected in this equation since they are not subject to scaling.

$$e = \frac{(0.15 + 0.3 + 0.75 + 0.15) + 0.1}{0.15 + 0.3 + 0.75 + 0.15} \approx 1.074$$

Power Reduction Mechanisms in CMOS Circuits

- Voltage reduction
  - Expense : slower gate speeds
- Power grid sizing and analysis
- Power efficient design of the clock distribution network and flip-flops
- Datapath width adjustment
- Effective circuit structures
- Clock gating
- Low power non volatile memory
- Dynamically varying supply voltages and threshold voltages
- With multiple supply voltages and multiple threshold voltages

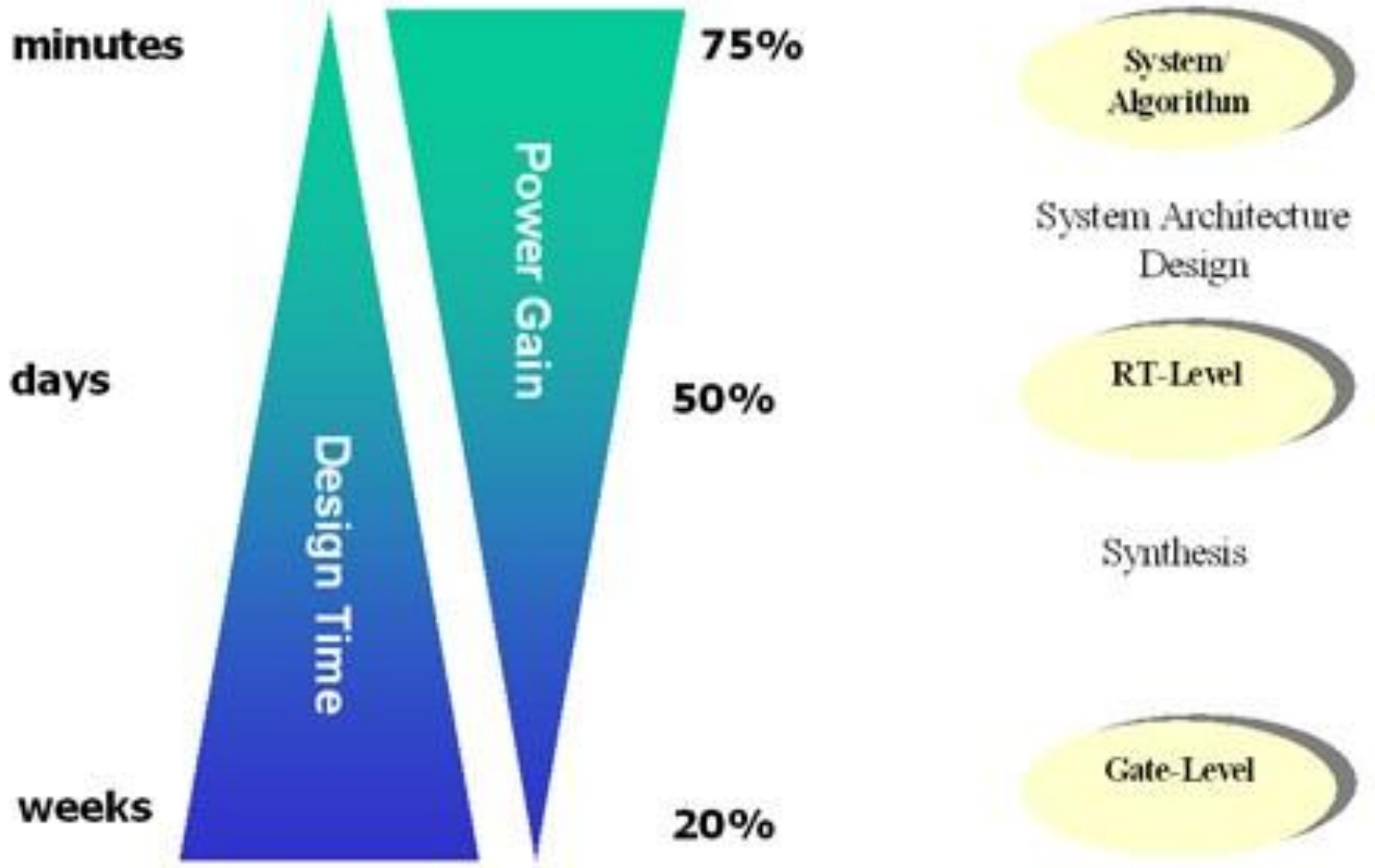
The traditional front-end approach



- Estimate and analyze power consumption at
  - the register transfer level (RTL)
  - the gate level
- Modify the design accordingly.
  - only the RTL within given functional blocks is modified
  - the blocks re-synthesized

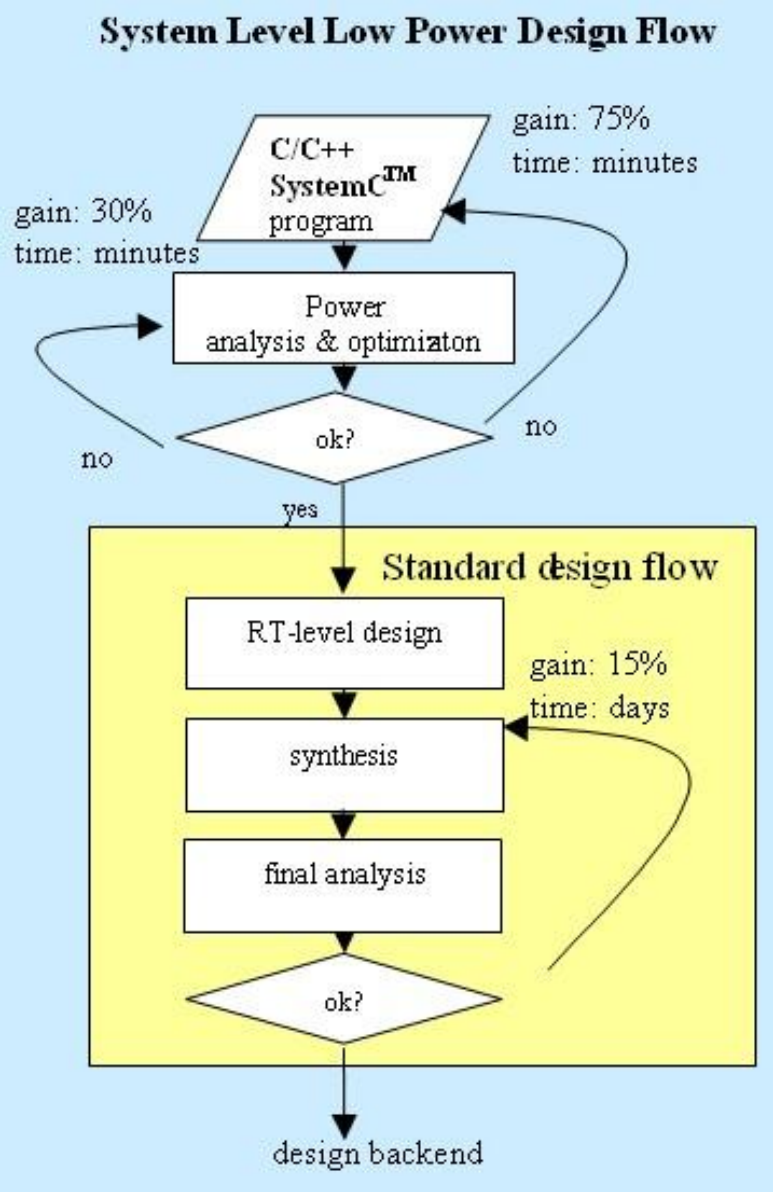
Modifying the design for low power

- The desired power consumption reductions may be achieved by:
  - modifying the architecture
  - modifying the algorithms
- Modifications at system level effect other performance metrics
- Such modifications require re-evaluation and re-verification of the entire design, and re-synthesis of the design.
- datapath optimization
  - may achieve power reductions of the order of 30%
  - a delay of several weeks.
- architectural and algorithmic re-design
  - reductions up to 75%
  - may take months.



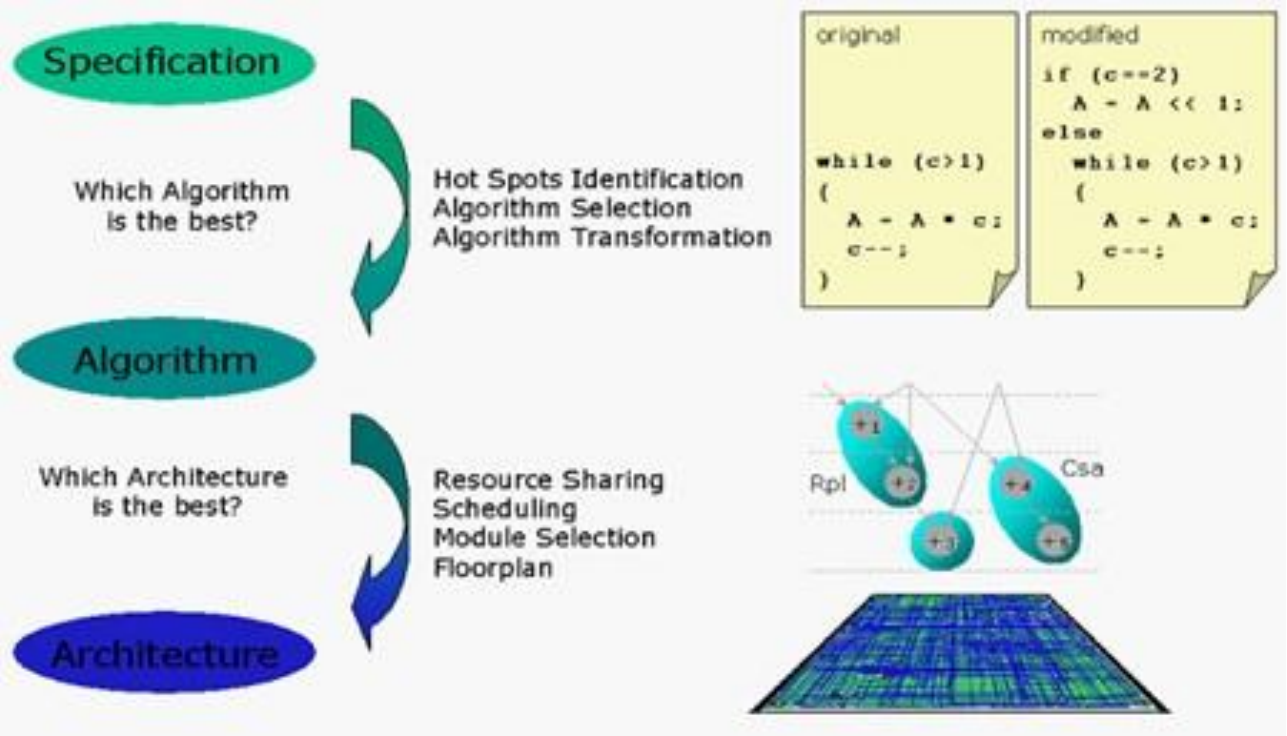
- algorithmic and architectural design decisions have the greatest influence on power consumption
- any new methodology must start at this **system level.**

The system-level solution



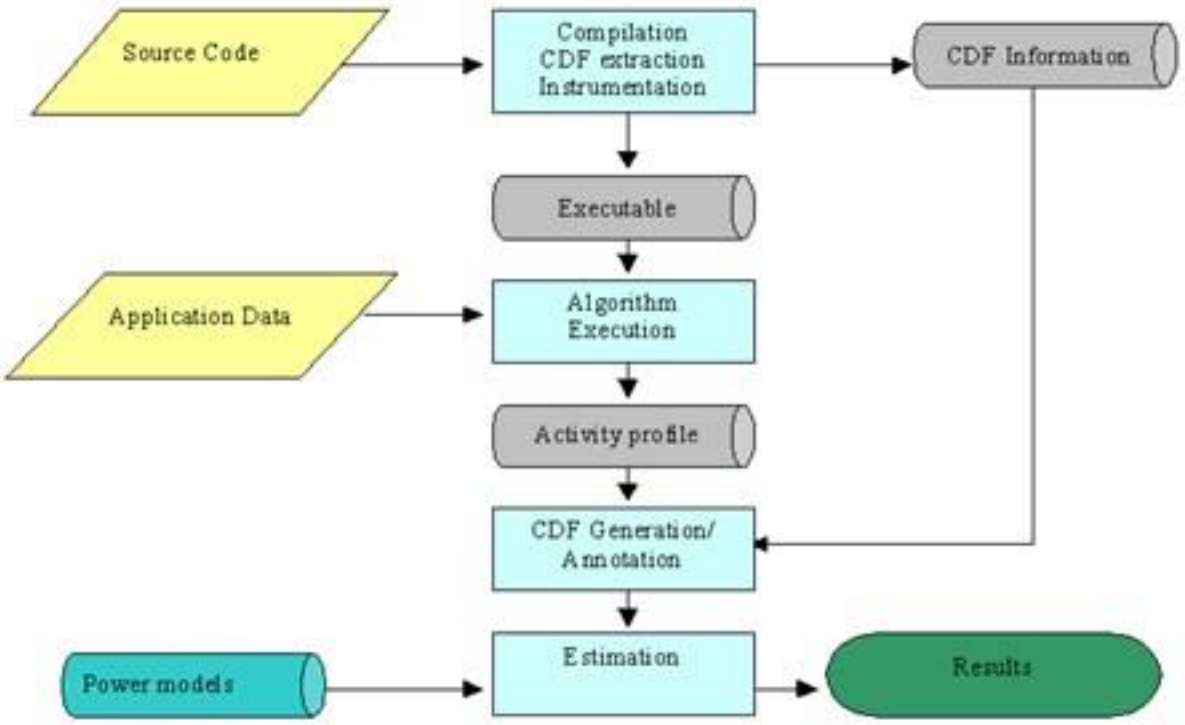
- System-level optimization targets dynamic power consumption.
- Short-circuit power and leakage power are optimized at lower levels of abstraction.

Power-optimal algorithms and architectures



- Candidate algorithms are analyzed
  - for their power characteristics
  - to identify potential function-level hot spots
- The most promising algorithms are then selected and optimized.
- A power optimal system architecture is created.
- The optimal power-consuming functions are transformed into hardware.

Algorithm analysis and optimization (1/2)



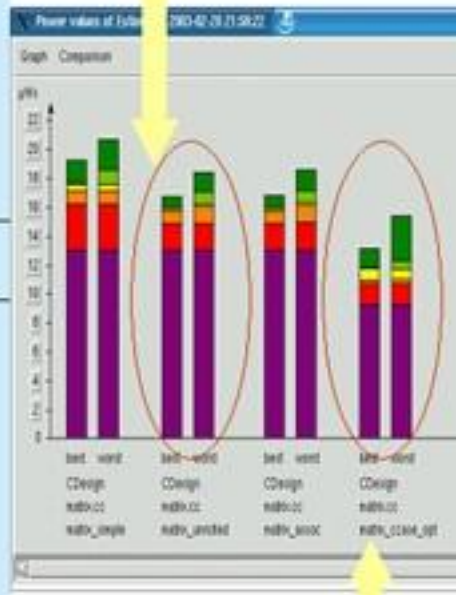
- Power-hungry parts of each algorithm are identified.
- The algorithm is optimized by algorithm transformation.
- The C/C++ or SystemC specification must first undergo compilation and instrumentation.
- Instrumentation: inserting the protocol statements necessary to derive the switching activity at each defined operation in the source code.

Algorithm analysis and optimization (2/2)

- The algorithms are then executed, and the resulting activity profile data is used to annotate a suitable design representation, such as a control data flow (CDF) graph.
- Any number of power estimations may then be performed to determine the power characteristics of any given configuration.
- A power-optimized architecture can be derived from this graph without executing a complete synthesis, using power models created for each RT-level component.
- These models depend on the input data, component characteristics such as bit width and architecture, and the underlying technology or cell library.
- Using the switching activity and the power models, the power consumption of a component can be estimated.
- Algorithm transformation techniques include:
  - HW/SW partitioning,
  - operator substitutions,
  - code transformations. Code transformations include transformations on conditional statements, loop splitting, and loop unrolling.
- Control statement reduction has a significant impact on the power consumption, and such transformations are often best effected via loop transformations.

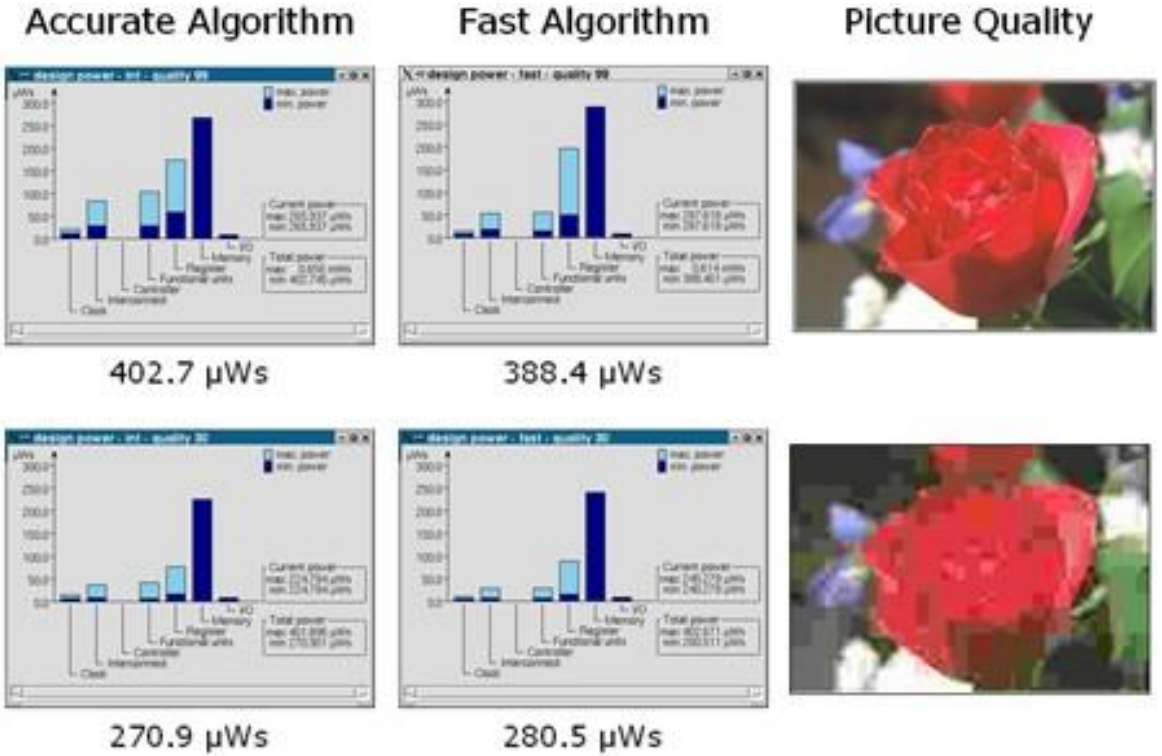
Power reduction effects of loop unrolling and common case optimization techniques.

```
// loop unrolling
void matrix_unrolled( int a[16], int b[16], int c[16] )
{
    // column * row
    int x, y;
    for ( y = 0; y < 4; y++ )
    {
        for ( x = 0; x < 4; x++ )
        {
            int sum = 0;
            sum = sum + me(a,x,0) * me(b,0,y);
            ...
            int idx;
            idx = x+y*4;
            c[idx] = sum;
        }
    }
}

// common case (zero) - optimized
void matrix_ccase_opt( int a[16], int b[16], int c[16]
)
{
    // column * row
    int x, y;
    for ( y = 0; y < 4; y++ )
    {
        for ( x = 0; x < 4; x++ )
        {
            int sum = 0;
            int tmp, tmp2;

            // Common case optimization to save a
            // memory access and a multiplication, if
            // a zero value is encountered.
            //
            tmp = me(b,0,y);
            if ( tmp > 0 )
            {
                sum = sum + me(a,x,0) * tmp;
            }
            tmp = me(b,1,y);
            if ( tmp > 0 )
            {
                sum = sum + me(a,x,1) * tmp;
            }
            ...
```



- The results show the power consumption for the original algorithm (matrix_simple), the algorithm optimized with loop unrolling (matrix_unrolled), and the matrix optimized by common case techniques (matrix_ccase_opt).

Selecting the best algorithm using the initial specification and the input data stream

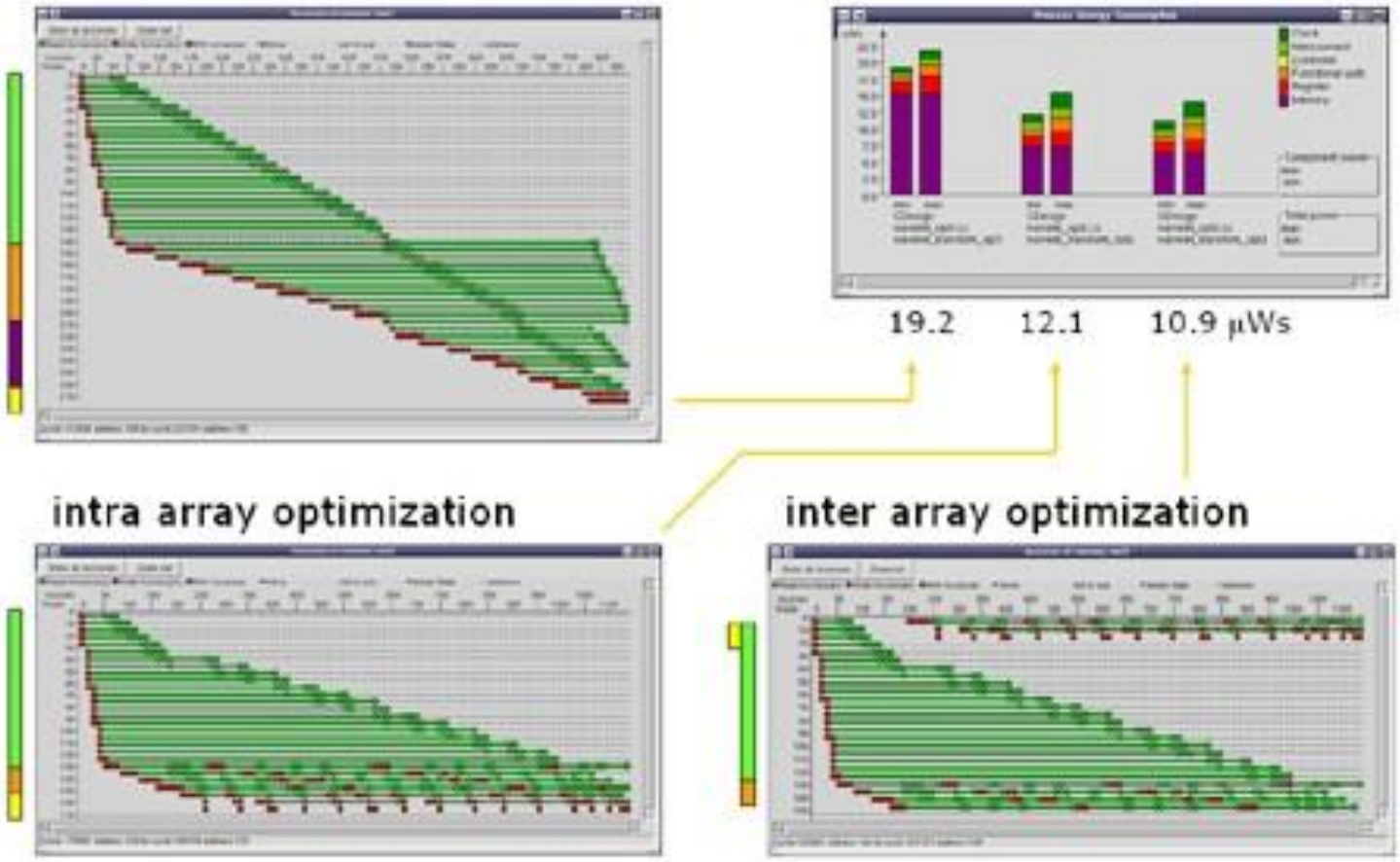


- The benchmark compares the energy consumption of the two algorithms resulting from the processing of two different input streams: a high quality stream — with a low compression ratio — consisting of 99% of the original data, and a fast stream consisting of 30% of the original data.
- It can be seen that the fast algorithm processes the quality stream with less power consumption, while the accurate algorithm consumes less power when processing the fast stream.
- Analysis of such algorithms by the traditional method would have necessitated extensive design work, and introduced a delay of weeks.

Creation of a power optimal architecture

- The optimal power-consuming algorithm is transformed into hardware.
  - Memory architecture
  - Scheduling
  - Number and types of resources
  - How those resources are shared and bound to the algorithm operators
  - Type of data encoding, controller design, floor plan and clock tree design.
- Resources can be distinguished not only by their function, but also by their internal architecture: an adder can be realized as a carry-ripple or a carry-select adder.
- Memories have a significant effect on chip power consumption, and often account for the majority of it — up to 80% in some SOC designs.
- Optimizing memory hierarchy and structure as early as possible is a major step in meeting power consumption constraints.
- Common techniques for optimizing memory access and memory system performance include: basic loop transformations such as loop interchange, loop tiling, loop unrolling, array contraction, scalar replacement, code co-location.

Visual display of power analysis results

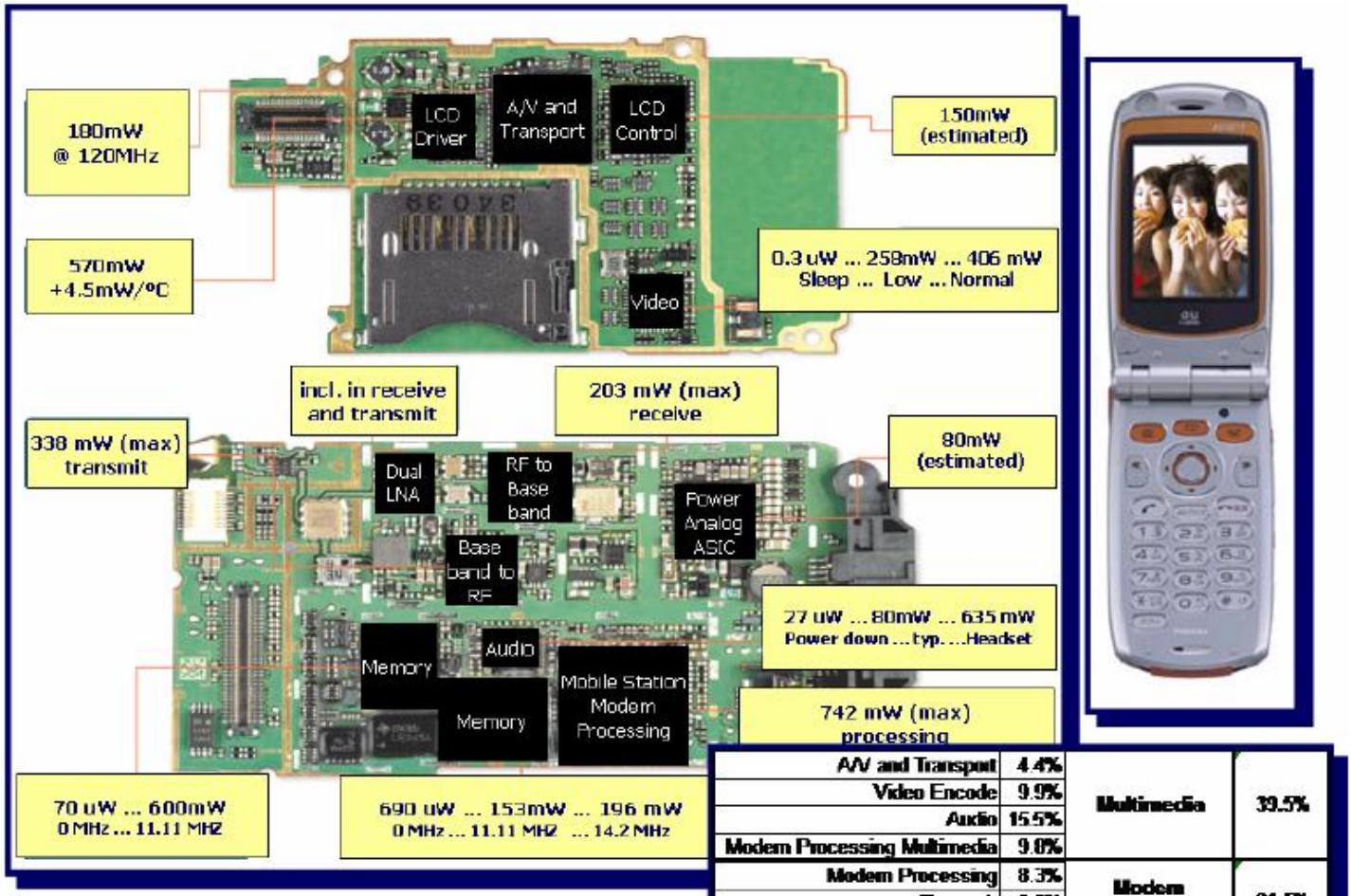


- The graphic shows an analysis of the power consumption of algorithms used in a digital signal processing application — a Wavelet transform.
- The bar graph shows the power consumed, while the memory access traces show memory usage.
- It can be seen that intra-array optimization reduces power consumption from 19.2 µW to 12.1 µW, or 37%.
- Inter-array optimization — memory size reduction by mapping arrays onto the same addresses of another array — reduces the consumption by another 1.2 µW, yielding a total 43% reduction.

Tool requirements

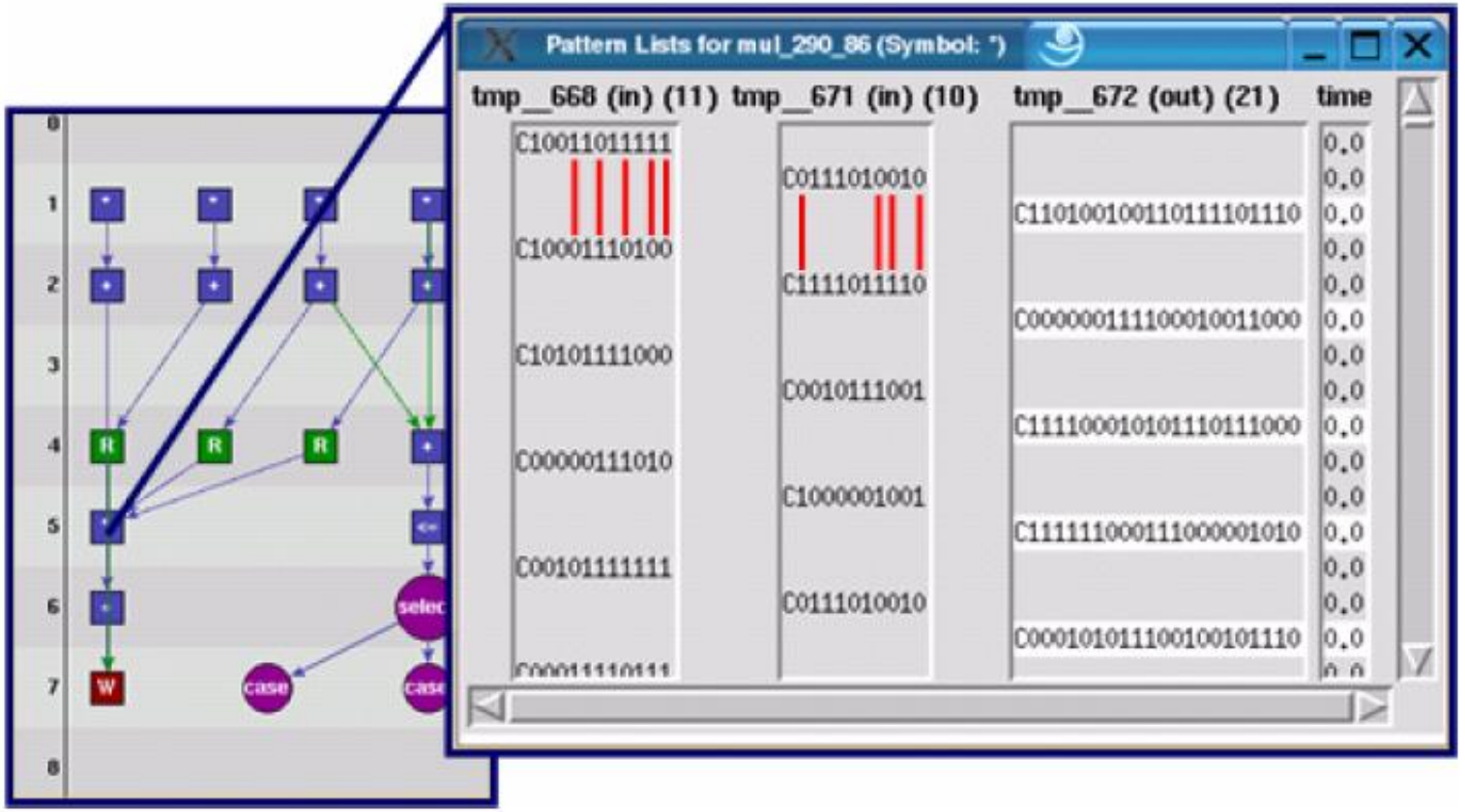
- Thus, the general requirements of such tools are:
  - Model creation tools must be capable of automatically characterizing RT-level macros and memories to create the appropriate power models for use by power estimation tools.
  - Estimation tools must perform comprehensive design space exploration automatically, calculating the maximum and minimum power consumption associated with each and every combination of algorithmic state and architecture.
  - The tools must accept data written in standard high-level languages, such as C/C++ and SystemC.
  - The tools must fit into a standard design flow, and communicate effectively with other tools in that flow.
- More specifically, the estimation tools must:
  - Estimate power using actual application data.
  - Perform instrumentation automatically on the initial specification.
  - Efficiently execute the algorithmic description with application data.
  - Automatically annotate the CDF with activity profile data.
  - Automatically create constraint scripts for high-level synthesis tools.
  - Accept standard data sheet information for I/O pads, registers, and off-chip memories.
  - Have an estimation scheme for clock power and interconnect.
  - Output the analyses in graphic form.

Typical Energy Distribution in a Multimedia Mobile Phone



A/V and Transpot	4.4%	Multimedia	39.5%
Video Encode	9.9%		
Audio	15.5%		
Modem Processing Multimedia	9.0%		
Modem Processing	8.3%	Modem Operation	21.5%
Transmit	8.2%		
Receive	5.0%		
Memory	19.4%	Memory	19.4%
LCD Control	3.7%	LCD	17.6%
LCD Driver	13.9%		
Power Management	2.0%	Control	2.0%

Activity at the inputs of a multiplier



- Whenever a signal changes from 0 to 1 or vice versa capacities have to be switched resulting in charge currents.
- Dynamic energy consumption can be influenced significantly if correlation between subsequent data values can be optimized to minimize the required switching.

Estimation and Optimization of Energy Consumption (1/2)

- ***Layout Level***
  - Sufficient data is available to allow accurate simulation and estimation
  - Depending on the estimates design teams can still do transistor sizing and layout re-arrangements to achieve optimization based on placement and interconnect.
  - The accuracy at this level of abstraction is very high
  - The amount of data to be processed and the simulation speed are so limited that typically only a small number of test vectors can be used for analysis.
  - The leverage on energy consumption is comparatively low as no major design changes like adjustments of the number of arithmetic resources can be made anymore.
- ***Gate Level***
  - The gate level netlist can be simulated with event based models.
  - Dynamic energy consumption can be determined fairly accurately based on the design activity.
  - Design teams can utilize optimizations minimizing the capacities which have to be driven by the most active nodes in the design.
  - Energy consumption can be optimized using balancing of path delays to avoid spikes and spurious transitions and re-timing.
  - The accuracy of the estimation is still quite high but the amount of data to be dealt with is limiting.

Estimation and Optimization of Energy Consumption (2/2)

- ***Register Transfer Level***
  - At the RT Level – prior to logic synthesis – energy consumption can be estimated using event based or probabilistic simulation.
  - Development tools at this level perform a "quick logic synthesis" followed by gate-level estimation.
  - The number of options to reduce the energy consumption is quite high, allowing reasonable overall leverage.
  - Specific areas of the design can be dynamically switched to low-power modes trading performance vs. energy during execution.
  - Popular methods are reduced clocking or full clock gating of areas in the design.
  - Optimized resource sharing, isolation of operands and optimized coding of controller and bus states can contribute to reduce the capacities to be switched.
  - Simulation times are still significant.
- ***Electronic System Level***
  - *Micro Architects,* who take an algorithm defined in C or SystemC and decide how to implement it, are facing various high impact areas for low-power optimization.

Bit Width Selection

- Typical algorithms defined in C or SystemC will initially not contain definitions of the actual bit width for operations and storage elements.
- For algorithm selection the design team often relies on floating point and straight integer calculations.
- Based on the stimulus which is applied to the *Design under Optimization* users can assess the minimum and maximum values on specific operations and then choose the optimal bit width accordingly.
- This allows users to understand the impact of bit width on energy and is a step towards trade offs between *quality*, which may be higher in a video application using higher bit width, vs. *energy* which decreases with lower bit width in the operators.

Trading Performance and Voltage vs. Energy Consumption

- Increasing the frequency at which an algorithm runs often will increase the energy consumption even though the algorithm could run at lower frequency and still would meet application requirements.
- The impact of lowering the frequency of an algorithm has significant impact as more operations can be chained between registers hence reducing the overall storage requirements.
- Alternatively the supply voltage can potentially be reduced resulting in slower performance but significantly reducing energy consumption because voltage contributes to energy consumption in a quadratic way.

Optimizing Design Activity

- If the activity of a design can be determined via simulation, users can observe the activity profile at the inputs of operations.
- This information helps to guide optimizations reducing the activity of operations both prior to and after allocation and binding of resources.
- If the simulation of activity is done at the C/SystemC level it is orders of magnitude faster than signal level simulation at the RT Level.

Trading Area vs. Energy Consumption

- The design space may allow to implement a schedule using different numbers of resources.
- Often additional resources will result in less overall energy consumption because with more resources the correlation of the subsequent data in a stream can be used to minimize switching of the capacities at the inputs.
- After going though this process the *Micro Architect* can clearly define the optimal number of resources trading area against low-power considerations.

Memory Optimization

- Designers will map arrays in the C Code to different types of memories accessible by the data path.
- Using memory access trace graphs users can identify areas of suspiciously high number of memory accesses.
- Users can then consider alternative memory access protocols and their impact on energy consumption.

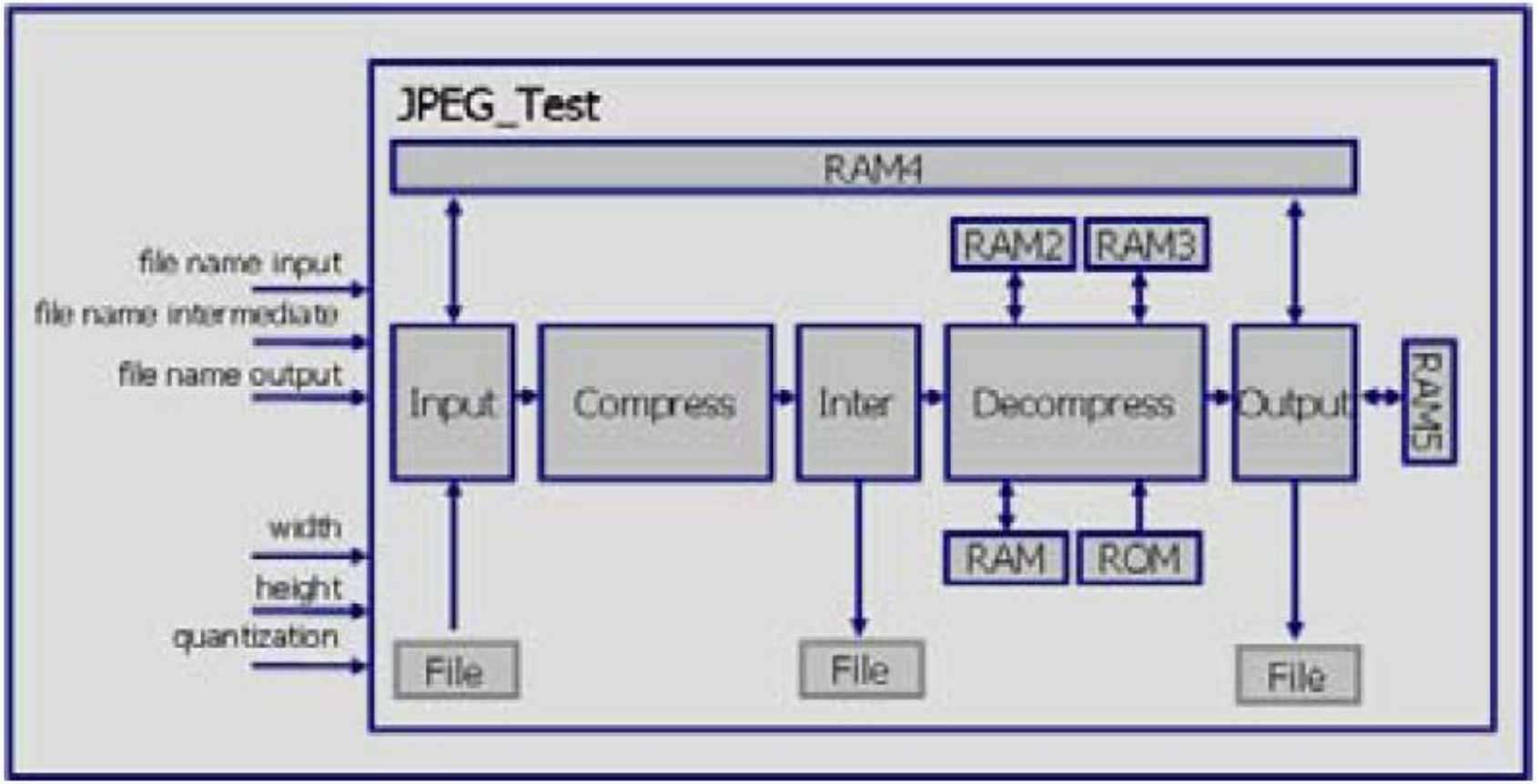
Clock Gating and voltage scaling

- If users understand on a cycle by cycle basis which resources are active and which resources are idle, the modules best suited for clock gating can be identified.
- Detailed knowledge which modules can be run at lower clock frequencies identifies potential for voltage scaling.

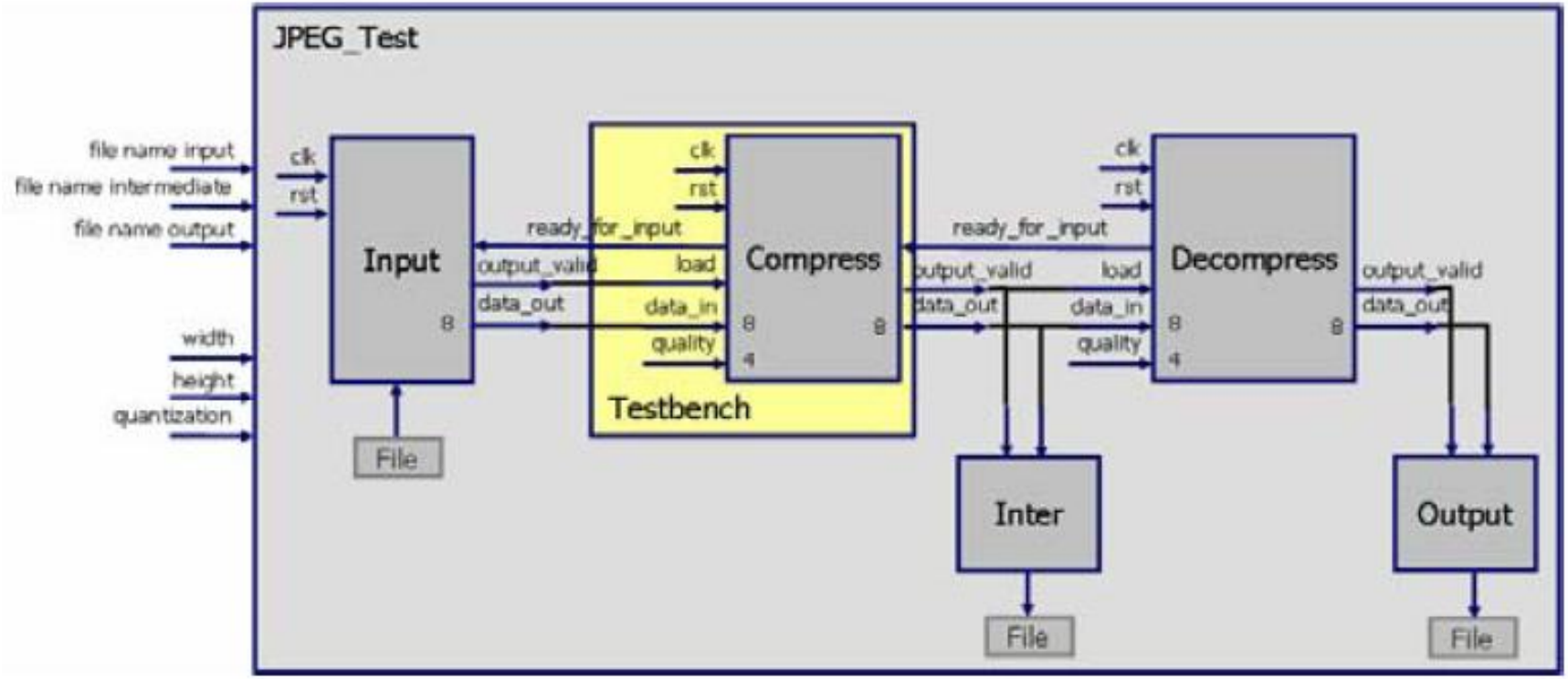
Trade offs between different abstraction levels

	Impact on Energy	***Time to Model***	***Accuracy***	***Amount of Data***	***Simulation Speed***
Pre-RTL	biggest	short (from C or SystemC)	ok	ok	ok
RT Level	ok	long	ok to good	large	slow
Gate Level	small	longer	good	larger	slower
Layout	smallest	longest	best	enormous	slowest

Design Example – A JPEG Decoder

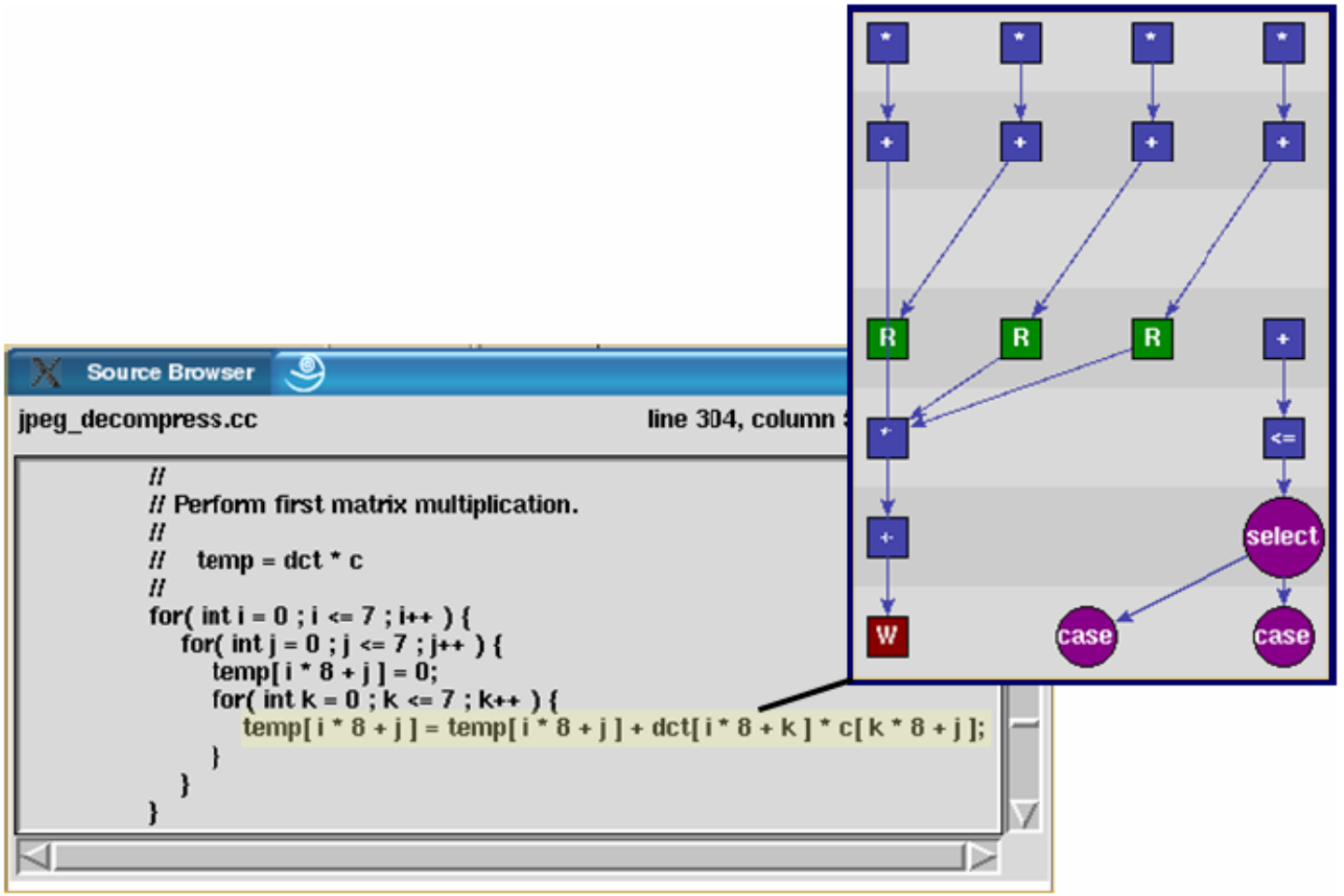


SystemC overview of the example design



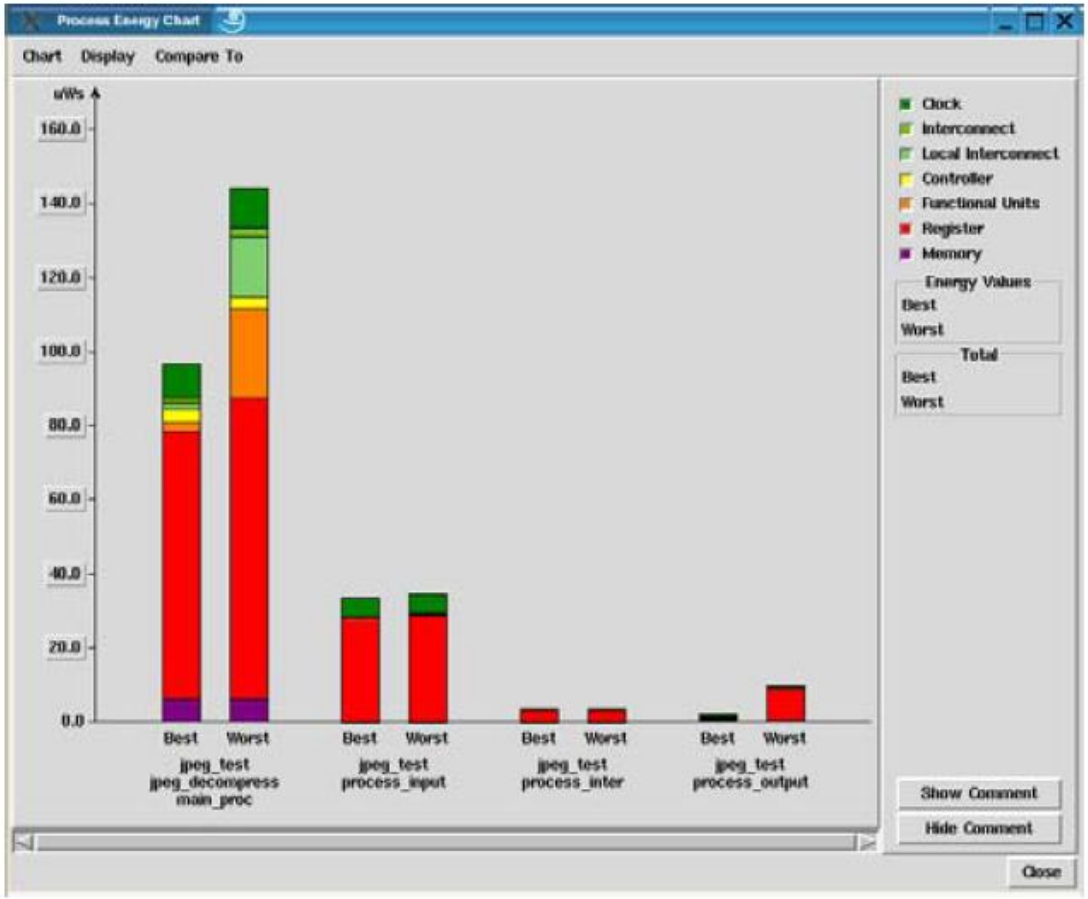
- The input data are read from a file.
- The compression itself has been integrated into the design as test bench.
- The module “Decompress” is the “Design under Development”, together with the module reading in data (“Input”), storing intermediate data (“Inter”) and presenting the data at the output (“Output”).

Behavioral code and Control-Dataflow Graph (CDFG)



•A source code example of a matrix multiplication and the corresponding control-dataflow graph (CDFG).
•This graph shows all arithmetic operations, dependencies and memory accesses.
•The task is to find the right set of resources to map and bind the behavior to in order to achieve a low-power implementation.

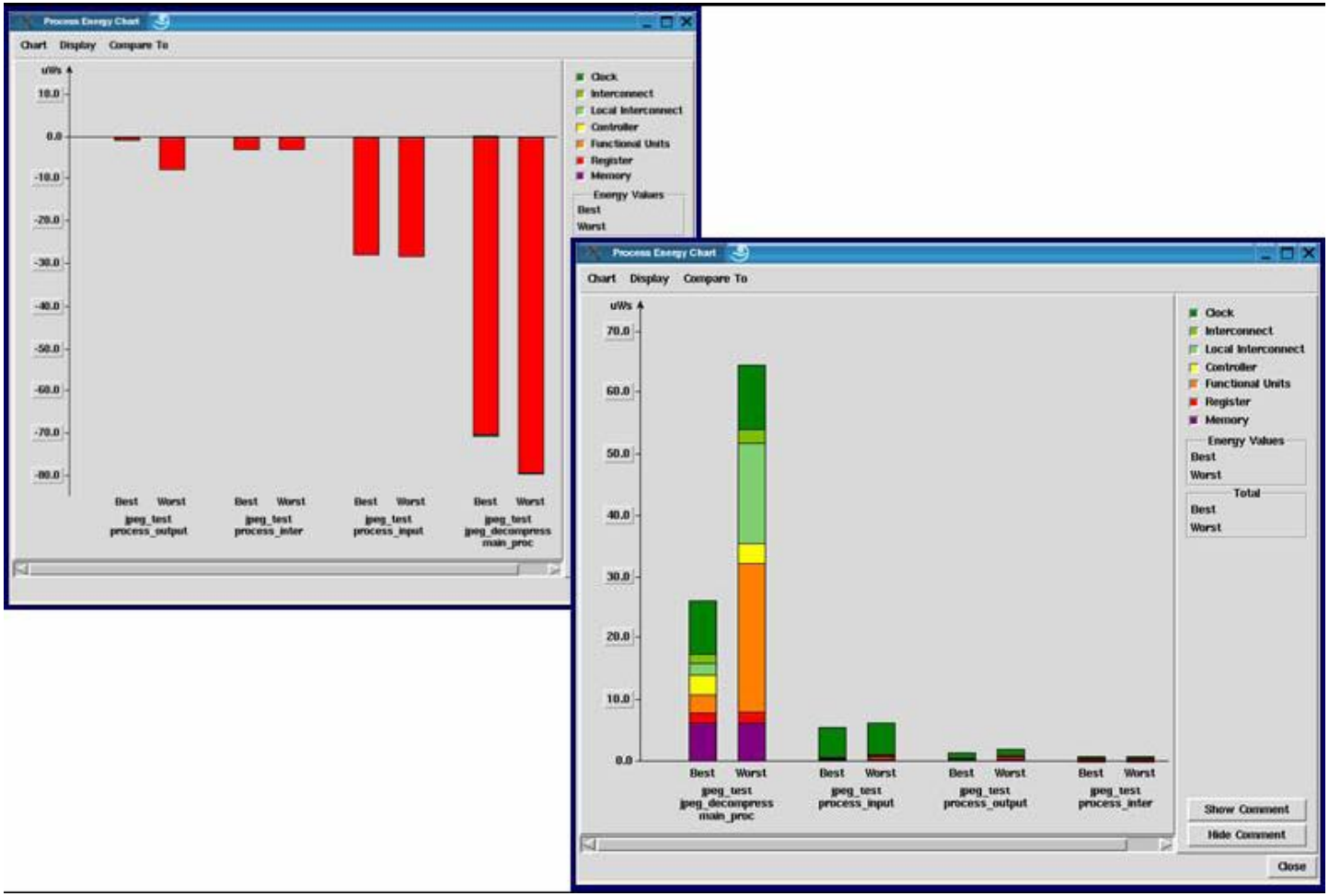
Initial Energy Consumption without Optimization



•Using ChipVision ORINOCO® technology the energy consumption was estimated.
•ORINOCO® assumes a macro based standard cell design flow.
•It bridges the gap between system level design in C or SystemC using a combination of micro architecture prediction, modeling of structural blocks in the design and simulation.

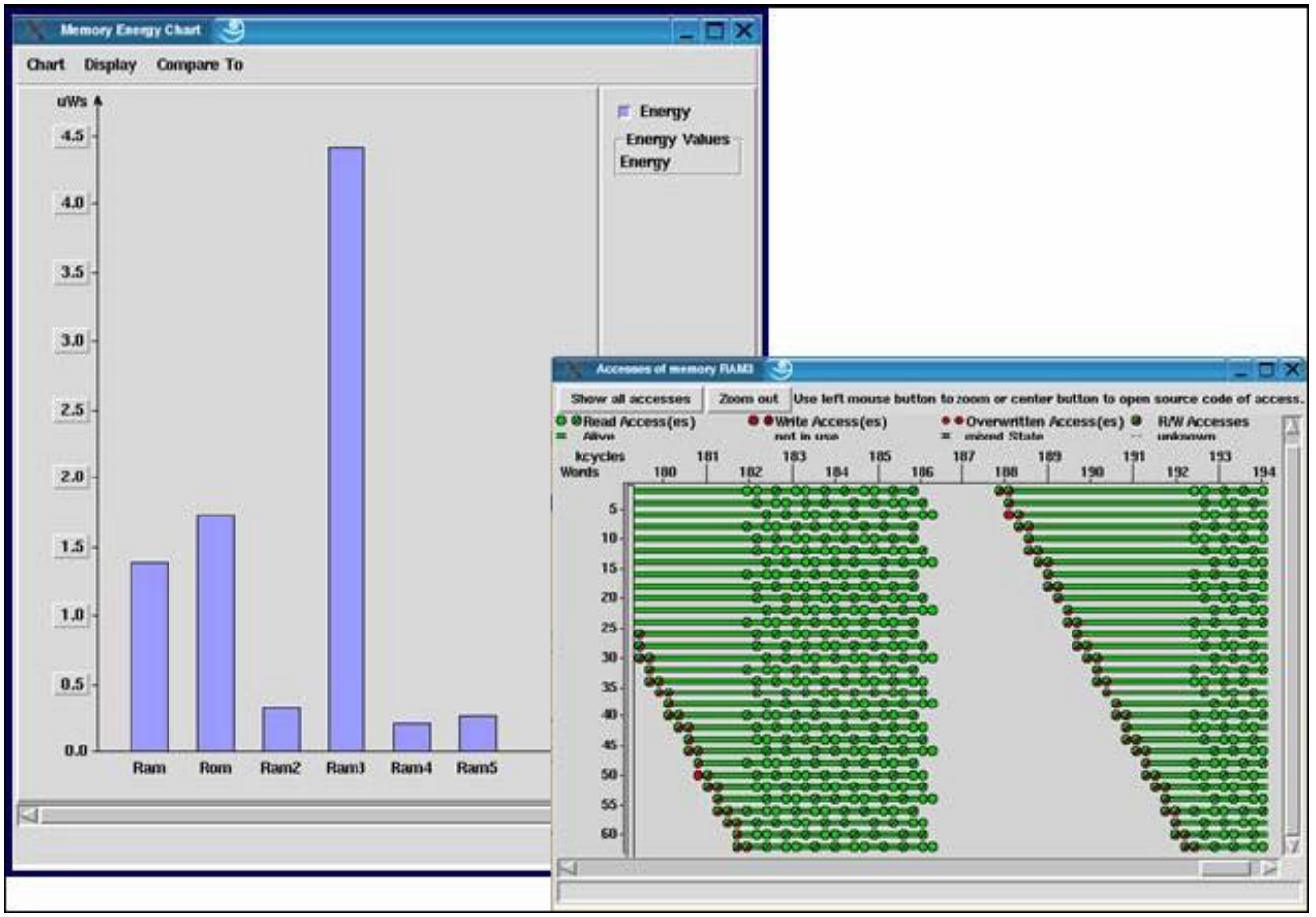
•Energy consumption is split into the different components Clock, Interconnect, Local Interconnect, Controller, Functional Units, Register and Memories.
•The left and right columns per block indicate the best and worst case energy consumptions within the defined constraint space. They represent different target micro-architectures.

Impact of Clock Gating



- From SystemC simulation it is possible to derive in which cycles each register is active to determine when to apply clock gating.
- The saving per module on the left and the new resulting overall energy distribution on the right.

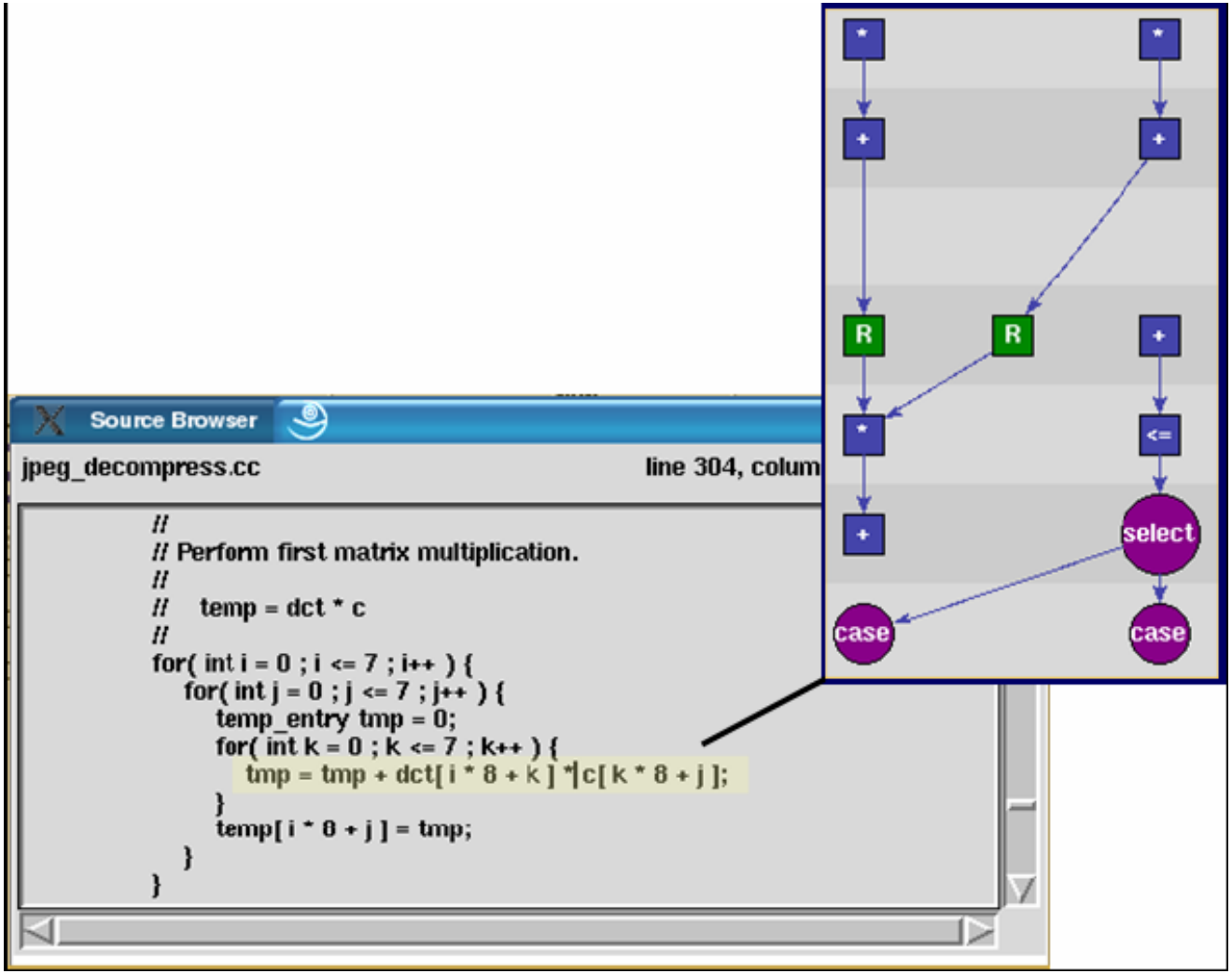
Memory Optimization (1/2)



Memory Optimization (2/2)

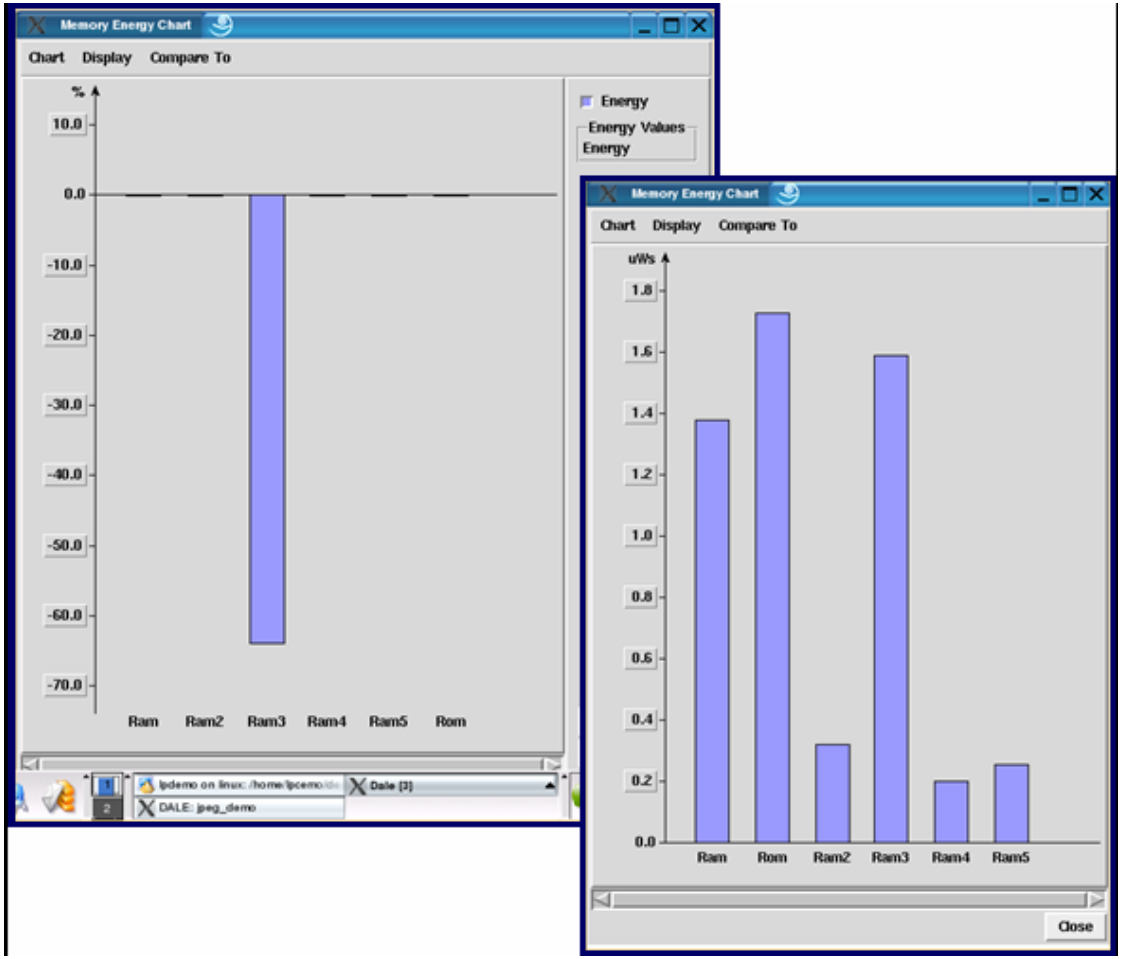
- The left side the actual impact of the different memories used in the design.
- The graph on the right side represents an access trace for one of the key memories, RAM3, which contributes the majority of the energy consumption.
- This access trace is caused by the code for matrix multiplication.
- It shows the timeline on the horizontal axis and the individual memory cells on the vertical axis.
- The experienced user will recognize a suspicious sequence of memory accesses following each other closely, always to the same address.
- A more detailed analysis of the code reveals that the matrix multiplication has been implemented in an n-optimized fashion.
- The access to a temporary storage array at temp[i*8+j] is done 8 times even though the surrounding loop does change neither i nor j.
- This behavior causes unnecessary memory accesses.

Modified source code

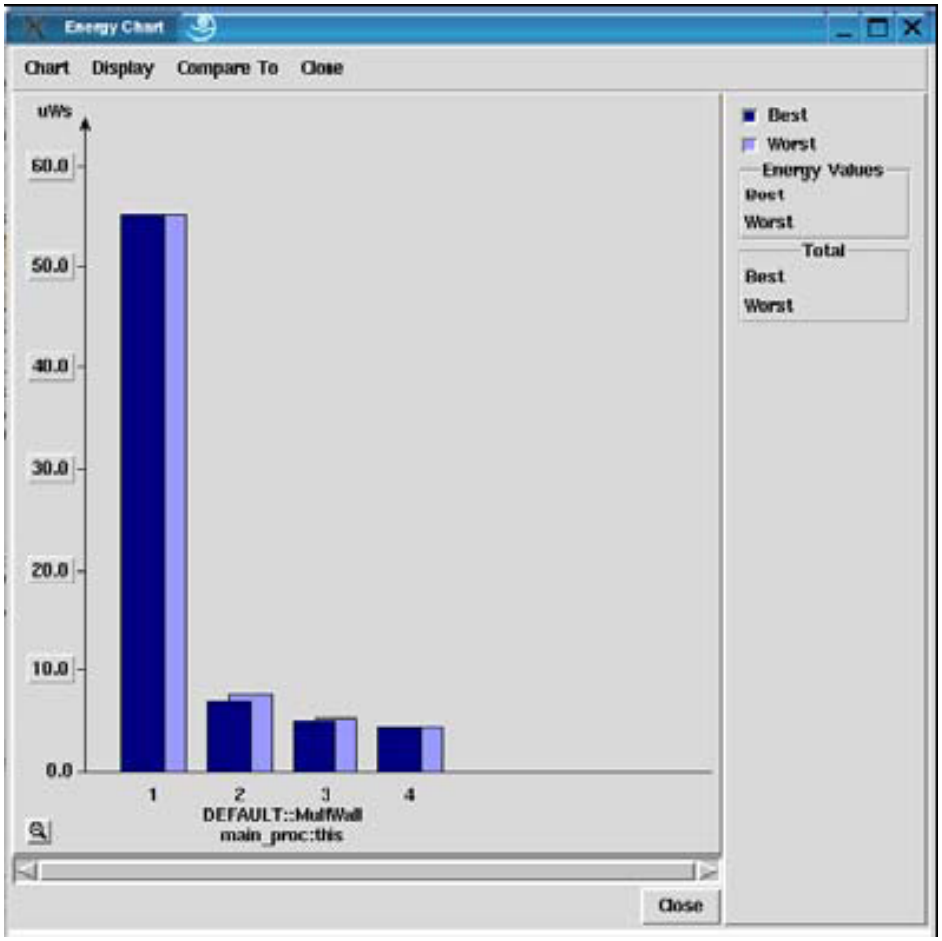


•Avoids this behavior by using a temporary variable instead of writing back to the memory in each cycle of the loop.

•The resulting saving in energy consumption is 60% in this example.


Memory Energy Chart
Chart Display Compare To
%
10.0
0.0
-10.0
-20.0
-30.0
-40.0
-50.0
-60.0
-70.0
Ram Ram2 Ram3 Ram4 Ram5 Rom
Energy
Energy Values
Energy
Memory Energy Chart
Chart Display Compare To
uWs
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Ram Rom Ram2 Ram3 Ram4 Ram5
Close

Data Dependent Optimization of Resources (1/2)



Data Dependent Optimization of Resources (2/2)

- Using SystemC simulation one can create activity traces representing typical use cases for the design under optimization.
- These activity traces represent the switching at the inputs of the arithmetic components – how many bits of an 8x8 multiplier change from 0 to1 or 1 to 0 per cycle.
- The influence of additional multipliers in the JPEG Design on the energy consumption.
- When comparing the options to implement a part of the decoder with 1, 2, 3 or 4 multipliers a reduction in energy consumption of more that 60% can be achieved!
- Even though 3 or 4 multipliers technically reduce the energy consumption even further for a small amount, it is probably not practical to do so as the saving has to be paid with by additional area being used.

Mechanisms for Shutting Down a Resource

- *Disabling registers or by gating the clock:* data propagation through combinational logic is halted.
- *Scale down supply voltage or turn power off:* usually involves a non-negligible time to restore operation.

Shut Down At Different Levels

- A hard-disk drive's operational states: idle, low-power idle, standby, sleep
- **Idle:** disk is spinning, but some of the electronic components of the drive are turned off. The transition from idle to active is extremely fast. 50-70% saved.
- **Standby and sleep:** disk is spun down, 90-95% saved. The transition to the active state is slow. It causes additional power consumption, because of the acceleration of the disk motor.
- The lower the power associated with a system state, the longer the delay in restoring an operational state.

Industrial Design Standards

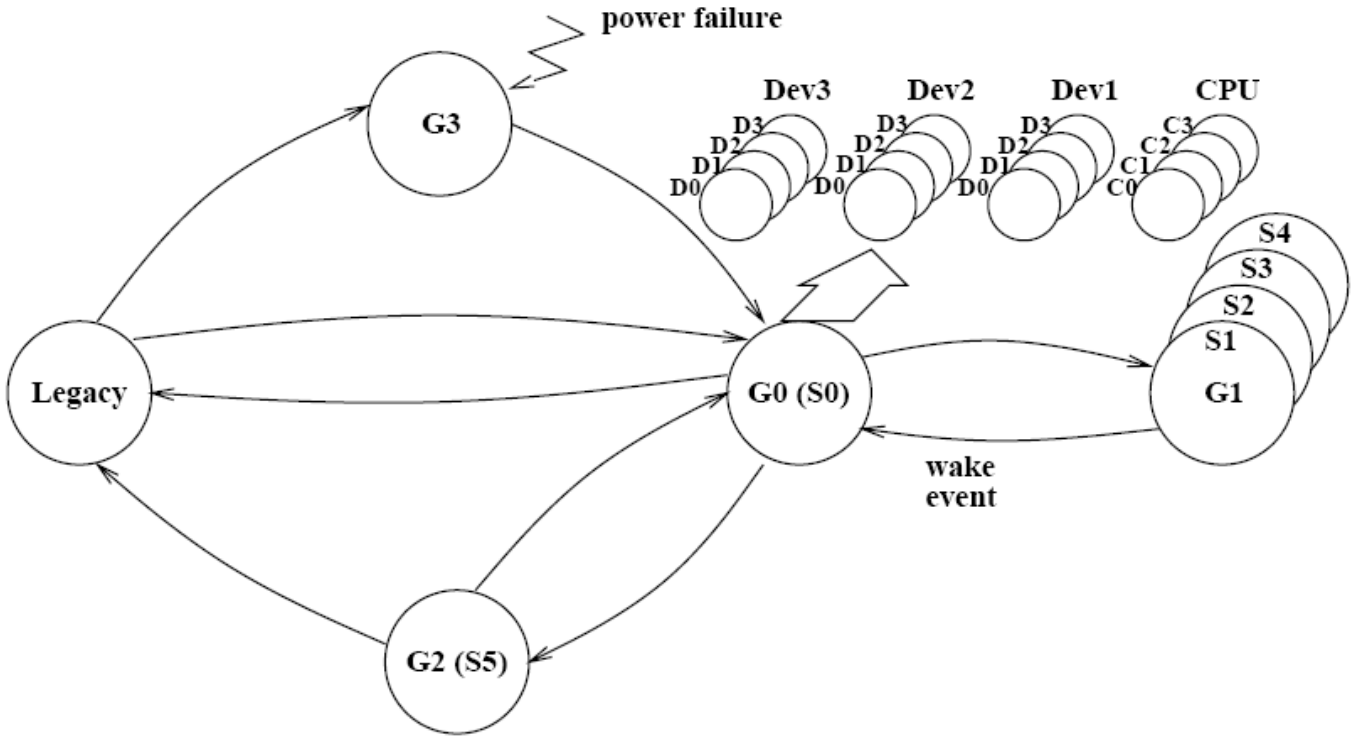
- Facilitate the development of operating system-based power management.
- ***ACPI:*** *Advanced Configuration and Power Interface* standard.
- ACPI recognizes dynamic power management as the key to reducing overall system power consumption, and it focuses on making the implementation of dynamic power management schemes in personal computers as straightforward as possible.

ACPI (1/3)

- Attempts to integrate power management features in the low-level routines that directly interact with hardware devices (firmware and BIOS).
- Open standard: for adoption by hardware vendors and operating system developers.
- Defines interfaces between OS software and hardware.
- Applications interact with the OS kernel through *application programming interfaces* (APIs).
- A module of the OS implements the power management policies.
- The power management module interacts with the hardware through kernel services (system calls).
- The kernel interacts with the hardware through device drivers.

ACPI (2/3)

- Describes the behavior of a PC with an abstract, hierarchical finite-state model.
- Transitions between states are controlled by the OS-based power manager.



ACPI (3/3)

- working (G0) and the sleeping (G1) states
- G0: the system appears fully operational, but the power manager can put idle devices to sleep (states D1 to D4, C0 to C3).
- When the entire system is idle or the user has pressed the power-off button, the OS will drive the computer into one of the global sleep states.
- The sleeping sub-states (S1 to S4) differ in which *wake* events can force a transition into a working state, how long the transition should take and how much power is dissipated in the state.
- User turn-on button: a latency of a few minutes can be tolerated, the OS could save the entire system context into non-volatile storage and transition the hardware into a soft-off state (G2).
- G2: power dissipation is almost null and context is retained in non-volatile memory for an arbitrary period of time.
- G3: Mechanical off state is entered in the case of power failure or mechanical disconnection of power supply. Complete OS boot is required to exit the mechanical off state.
- *legacy* state: entered in case the hardware does not support OSPM.

System-level Power Management

1. The entire chip can be put in one of several sleep states through external signals or software control.
2. Chip units can be shut down by stopping their local clock distribution.

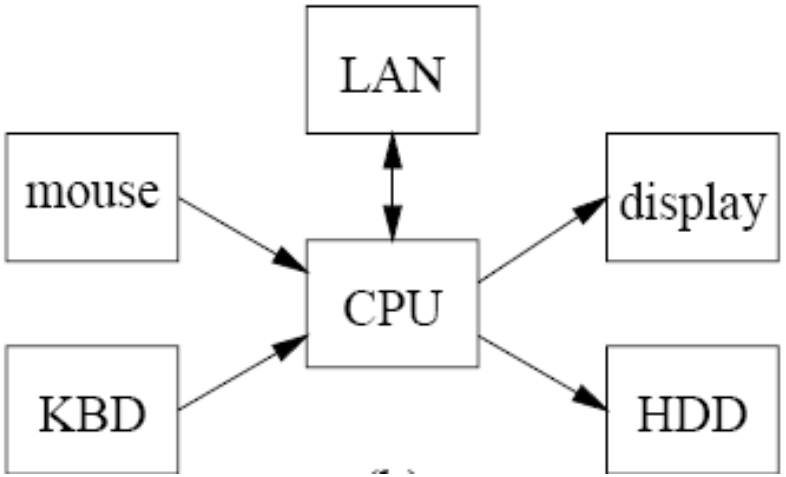
System Modeling

- The hardware part of the system is a set of resources.
- Resources: Units that perform or request specific services and that communicate by requesting and acknowledging such services.
- The hardware behavior: *finite-state system*
- Power and service levels are associated with the different states and transitions among states.
- It is difficult to have precise information about power and performance levels of each resource.
- This uncertainty can be modeled by using random variables for the observable quantities of interest, and by considering average values as well as their statistical distributions.
- Computing optimum dynamic power management policies: a *stochastic optimum control* problem
- The problem solution, and its accuracy in modeling reality, depend highly on the assumptions we use in modeling.

Assumptions

- Resources: providers and requesters of services to other resources.
- *System structure:* resources are vertices of a directed graph and resource interactions are edges.
- The interaction is the request of a service and/or its delivery.
- *Queues:* accumulation of requests waiting for services.

Example Of A CPU Requesting Data



•CPU interacting with a LAN interface, a HDD, a display, a keyboard and a mouse.
•Requests to the CPU can be originated from the keyboard, mouse and LAN interface.
•The CPU can request services to the display, the HDD and LAN.
•The keyboard and mouse models can express also the behavior of the human user who hits their keys and buttons.

Statistical Properties Of The Components Of A System

- *Stationarity* of a stochastic process: its statistical properties are invariant to a shift of the time origin.
- When resources are viewed as providers of services in response to input stimuli, their behavior is stationary.
- Conversely,when resources act asworkload sources,andwhen wemodel users' requests
- as such, the stationarity assumption may not hold in general. For example, patterns
- of human behavior may change with time, especially when considering the fact that
- an electronic system may have different users. On the other hand, observations of
- workload sources over a wide time interval may lead to stationary models that are
- adequately accurate. An advantage of using stationary models is the relative ease of
- solving the corresponding stochastic optimization problems.