Chapter 5
Evaluation and Validation

During the design procedure, we have to check repeatedly whether or not the system under design is likely to perform its function and to satisfy all relevant design objectives. This is the purpose of validations and evaluations which must be performed during the design process. This chapter starts with a presentation of techniques for the evaluation of (partial) designs with respect to objectives. In particular, we consider (worst case) execution time, quality of results, thermal behavior, and dependability as objectives. We provide an introduction into fundamental techniques for computing the worst case execution time. Examples of energy models will be presented in order to demonstrate the need for an adjustment of the level of model details to the particular application at hand. Thermal modeling is reduced to the problem of equivalent electrical modeling. With respect to dependability, an introduction to statistical models of reliability as well as an introduction to fault trees are included. As a means for relating results for the different objectives against each other, we introduce the concept of Pareto optimality. This chapter closes with hints regarding validation techniques, including simulation, rapid prototyping, and formal verification.

5.1 Introduction

5.1.1 Scope

Specification, hardware platforms, and system software provide us with the basic ingredients which we need for designing embedded systems. During the design process, we must validate and evaluate designs rather frequently. These activities can be defined as follows:

The original version of this chapter was revised: Caption for the second part of Fig. 5.25 has been updated. A correction to this chapter is available at https://doi.org/10.1007/978-3-030-60910-8_9

P. Marwedel, *Embedded System Design*, Embedded Systems,
https://doi.org/10.1007/978-3-030-60910-8_5

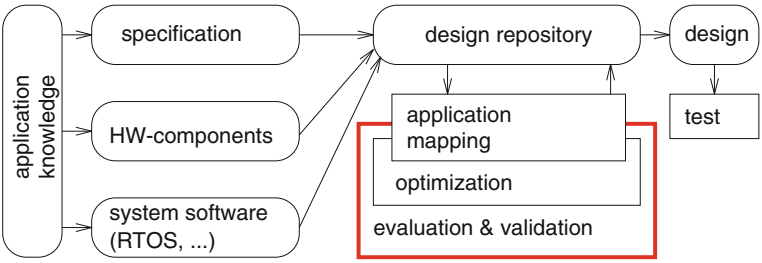

Fig. 5.1 Context of the current chapter

Definition 5.1 Evaluation is the process of computing quantitative information of some key characteristics (or "objectives") of a certain (possibly partial) design.

Definition 5.2 Validation is the process of checking whether or not a certain (possibly partial) design is appropriate for its purpose, meets all constraints, and will perform as expected.

Definition 5.3 Validation with mathematical rigor is called **(formal) verification**.

Validation and evaluation are required at various phases during the design procedure (see Fig. 5.1). Validation and design should be intertwined and not be considered as two completely independent activities. Validation and evaluation, even though different from each other, are very much linked. Due to their impact, we will describe validation and evaluation before we talk about design steps.

5.1.2 Multi-Objective Optimization

Design evaluations will, in general, lead to a characterization of the design by several criteria, such as execution time, energy consumption, quality of results, thermal behavior, and dependability. Merging all these criteria into a single objective function (e.g., by using a weighted average) is usually not advisable, as this would hide some of the essential characteristics of designs. Rather, it is recommended to return to the designer a set of designs among which the designer can then select an appropriate design. Such a set should, however, only contain "reasonable" designs. Finding such sets of designs is the purpose of **multi-objective optimization techniques**.

In order to perform multi-objective optimization, we do consider an m-dimensional space X of possible solutions of the optimization problem. These dimensions could, for example, reflect the number of processors, the sizes of memories, as well as the number and types of buses. For this space X, we define an n-dimensional

function

$$f(x) = (f_1(x), \ldots, f_n(x)) \text{ where } x \in X$$

which evaluates designs with respect to several criteria or objectives (e.g., cost and performance). Let F be the n-dimensional space of values of these objectives (the so-called objective space). Suppose that, for each of the objectives, some total order $<$ and the corresponding $\leq$ order are defined. In the following, we assume that the goal is to **minimize** our objectives.

Definition 5.4 Vector $u = (u_1, \ldots, u_n) \in F$ **dominates** vector $v = (v_1, \ldots, v_n) \in F$ iff u is "better" than v with respect to at least one objective and not worse than v with respect to all other objectives:

$$\forall\, i \in \{1, \ldots n\} : u_i \leq v_i \quad \wedge \tag{5.1}$$

$$\exists\, j \in \{1, .., n\} : u_j < v_j \tag{5.2}$$

Definition 5.5 Vector $u \in F$ is called **indifferent** with respect to vector $v \in F$ iff neither u dominates v nor v dominates u.

Definition 5.6 A design $x \in X$ is called **Pareto optimal** with respect to X iff there is no design $y \in X$ such that $u = f(x)$ is dominated by $v = f(y)$.

The previous definition defines Pareto optimality in the solution space. The next definition serves the same purpose in the objective space.

Definition 5.7 Let $S \subseteq F$ be a subset of vectors in the objective space. $v \in F$ is called a **non-dominated solution** with respect to S iff v is not dominated by any element $\in S$. v is called Pareto optimal iff v is non-dominated with respect to all solutions F.

Figure 5.2 highlights the different areas in an objective space with objectives $O1$ and $O2$, relative to design point (1).

The upper right area corresponds to designs that would be dominated by design (1), since they would be "worse" with respect to both objectives. Designs in the lower left rectangle would dominate design (1), since they would be "better" with respect to both objectives. Designs in the upper left and the lower right area are indifferent: they are "better" with respect to one objective and "worse" with respect to the other. Figure 5.2 (right) shows a set of Pareto points, i.e., the so-called Pareto front.

Definition 5.8 Design space exploration (DSE) based on Pareto points is the process of finding and returning a set of Pareto optimal solutions to the designer, enabling the designer to select the most appropriate implementation.

In order to visualize objectives in multiple dimensions, so-called radar charts, spider charts, or Kiviat diagrams can be used [579]. They are extensions of the type of diagram which we have used in Fig. 2.74 to multiple dimensions.

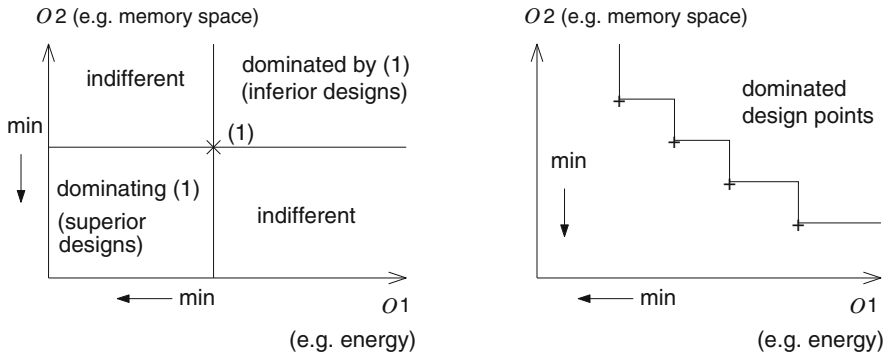

Fig. 5.2 Pareto optimality: **left**, Pareto point; **right**, Pareto front

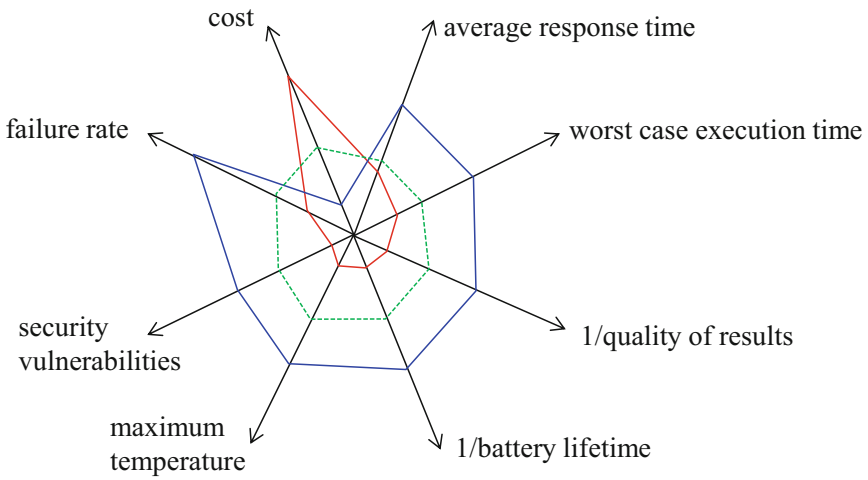

Fig. 5.3 Kiviat diagram: top (red), mid-range (green, dashed), and entry-level (blue) models

Example 5.1 As shown in Fig. 5.3, we can compare several designs (e.g., of mobile phones) according to objectives similar to the ones presented in the next subsection.

Minimization of all objectives is assumed. The top model minimizes most objectives, except for costs. For the entry level model, it is the other way around. ∇

5.1.3 Relevant Objectives

For servers and PCs, the average performance plays a dominating role. For embedded and cyber-physical systems, multiple objectives need to be considered. The following list explains if and where this objective is discussed in this book:

1. **Average performance:** Some comments on this objective will be made in Sect. 5.2. This objective is frequently computed from simulations, which will be introduced in Sect. 5.7.
2. **Worst case performance/real-time behavior:** Some fundamental techniques for computing the worst case execution time (WCET) will be presented in Sect. 5.2.2. This will be complemented by an introduction to real-time calculus in Sect. 5.2.3.
3. **Quality metrics:** Quality metrics will be presented in Sect. 5.3. In addition, transformations between number systems are discussed in Sect. 7.1.5.
4. **Energy/power consumption:** A brief overview of techniques for evaluating this objective will be presented in Sect. 5.4.
5. **Thermal models:** An introduction to this topic will be presented in Sect. 5.5.
6. **Dependability:** Dependability is the topic of Sect. 5.6, with subsections on safety, security, and reliability.
7. **Electromagnetic compatibility:** This objective will not be considered here.
8. **Testability:** Costs for testing systems can be very large, sometimes larger even than production costs. Hence, testability should be considered as well, preferably already during the design. Testability will be discussed in Chap. 8.
9. **Cost:** Cost in terms of silicon area or real money will not be considered here.
10. **Weight, robustness, usability, extendability, and environmental friendliness:** These objectives will also not be considered.

There are more objectives than the ones listed above. For example, we could use standards for the evaluation of software quality, like standards ISO/IEC 25022 [258], ISO/IEC 25023 [259], and ISO/EIC 25024 [257]. The next section presents some approaches for performance evaluation, focusing on the worst case performance.

5.2 Performance Evaluation

Performance evaluation aims at predicting the performance of systems. This is a major challenge (especially for cyber-physical systems) since we might need worst case information, rather than just average case information. Such information is necessary in order to guarantee real-time constraints.

5.2.1 Early Phases

Two different classes of techniques have been proposed for obtaining performance information already during early design phases:

- **Estimated cost and performance values:** Quite a number of estimators have been developed for this purpose. Examples include the work by Jha and Dutt

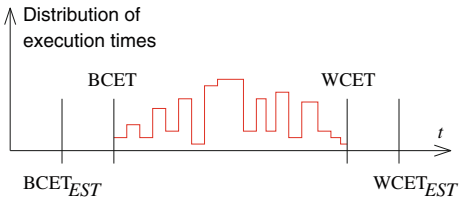

Fig. 5.4 WCET-related terms

[274] for hardware and Jain et al. [266] and Franke [167] for software. Generating sufficiently precise estimates requires considerable efforts.

- **Accurate cost and performance values:** We can also use the real binary software code on a close-to-real hardware platform. This is only possible if interfaces to compilers exist. This method can be more precise than the previous one but may be significantly (and sometimes prohibitively) more time-consuming.

In order to obtain sufficiently precise information, communication needs to be considered as well. Unfortunately, it is typically difficult to compute communication cost already during early design phases.

Formal performance evaluation techniques have been proposed by many researchers. For embedded systems, the work of Thiele et al., Henia and Ernst et al., and Wilhelm et al. is particularly relevant (see, e.g., [210, 536] and [587]). These techniques require some knowledge of architectures. They are less appropriate for early design phases, but some of them can be used without knowing all details about target architectures. These approaches model real, physical time.

5.2.2 WCET Estimation

Scheduling of tasks requires knowledge about the duration of task executions, especially if meeting time constraints has to be guaranteed, as in real-time (RT) systems. The **worst case execution time** (WCET) is the basis for most scheduling algorithms. Some definitions related to the WCET are shown in Fig. 5.4.

Definition 5.9 The **worst case execution time** (WCET) is the largest execution time of a program for any input and any initial execution state.

Unfortunately, the WCET is extremely difficult to compute. In general, it is undecidable whether or not the WCET is finite. This is obvious from the fact that it is undecidable whether or not a program terminates. Hence, the WCET can only be computed for certain programs/tasks. For example, for programs without recursion, without while loops, and with loops having statically known iteration counts, decidability is not an issue. But even with such restrictions, it is usually practically impossible to compute the WCET exactly. The effect of modern processor architectures' pipelines with their different kinds of hazards and memory hierarchies with limited predictability of hit rates is difficult to precisely predict

at design time. Computing the WCET for systems containing interrupts, virtual memory, and multiple processors is an even greater challenge. As a result, we must be happy if we are able to compute good **upper bounds** on the WCET.

Such upper bounds are usually called **estimated worst case execution time**s, or WCET_{EST}. Such bounds should have at least two properties:

1. The bounds should be safe ($\text{WCET}_{EST} \geq \text{WCET}$).
2. The bounds should be tight (WCET_{EST}-WCET $\ll$ WCET).

Note that the term "estimated" does not mean that the resulting times are unsafe.

Sometimes, architectural features which reduce the average execution time but cannot guarantee to reduce WCET_{EST} are completely omitted from the real-time designs (see p. 154). Computing tight upper bounds on the execution time may still be difficult. The architectural features described above also present problems for the computation of WCET_{EST}. The computation of such bounds is extremely difficult for multi-cores. In fact, potential conflicts might even cause multi-cores to have larger worst case bounds than the corresponding single cores.

Definition 5.10 The **best-case execution time** (BCET) is the smallest execution time of a program, considering all feasible inputs and initial states. The BCET_{EST} is a safe and tight lower bound on the execution time.

Computing tight bounds from a program written in a high-level language such as C without any knowledge of the generated assembly code and the underlying architectural platform is impossible. Therefore, a safe analysis must start from real machine code. Any other approach would lead to unsafe results.

We will study WCET estimation more closely, using a description of the tool aiT by R. Wilhelm [587]. The architecture of aiT is shown in Fig. 5.5.

Consistent with our remark about the problems with high-level code, aiT starts from an executable object file comprising the code to be analyzed. From this code, a control flow graph (CFG) is extracted. Next, loop transformations are applied. These include transformations between loops and recursive function calls as well as virtual loop unrolling. This unrolling is called "virtual" since it is performed internally, without actually modifying the code to be executed. Results are represented in the CRL (control flow representation language) format. The next phase employs different static analyses. Static analyses read the AIP-file comprising designer's annotations. These annotations contain information which is difficult or impossible to extract automatically from the program (e.g., bounds of complex loops). Static analyses include value analysis, cache analysis, and pipeline analyses.

A **value analysis** computes enclosing intervals for possible values in registers and local variables. The resulting information can be used for control flow analysis and for data cache analysis. Frequently, values such as addresses are precisely known (especially for "clean" code), and this helps in predicting accesses to memories.

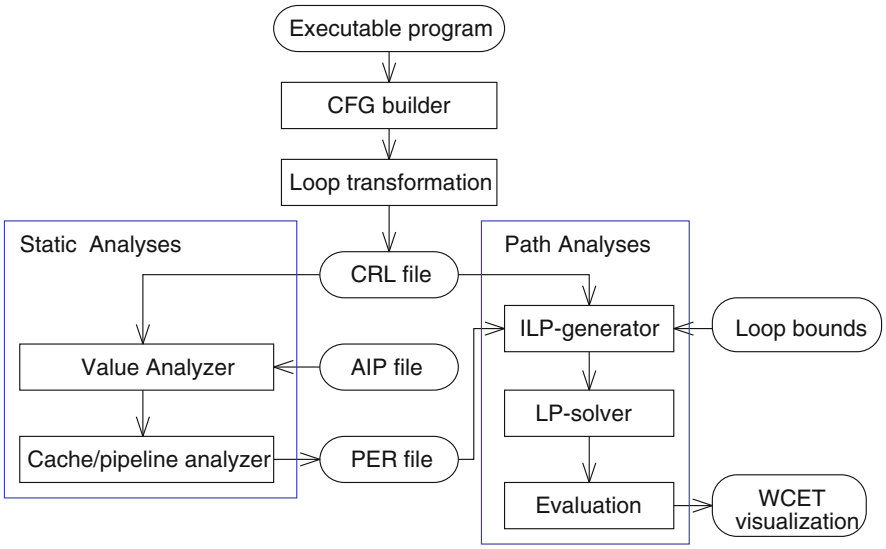

Fig. 5.5 Architecture of the aiT timing analysis tool

The next step is **cache** and **pipeline analysis**. We will present a few details about the cache analysis. Suppose using an n-way set associative cache (see Fig. 5.6).[1] We consider that part of the cache (the **row**) corresponding to a certain index (shown in bold and blue in Fig. 5.6). We assume that eviction from the row is controlled by the least recently used (LRU) strategy.[2] This means that among all references for a particular index, the last n referenced memory blocks are stored in the row. We assume that the necessary LRU management hardware is available for each index and that each index is handled independently of other indexes. Under this assumption, all evictions for a particular index are completely independent of decisions for other indexes. This independence is extremely important, since it allows us to consider each of the indexes independently.

Let us now consider a row and a particular index. Suppose that we have information about potential entries for each of the cache ways (columns). What will happen in case of an access to a particular index? First of all, let us consider the case of an access to a variable e known to be in the cache. After that access, that variable is known to be the youngest (see Fig. 5.7). Entries on the left are assumed to be younger than the ones on the right.

Now, assume that we have an access to some variable (say c) which is not yet in the cache. This access will remove the oldest entry from the cache (see Fig. 5.8).

[1] We assume that students are familiar with concepts of caches.

[2] Unfortunately, this strategy is typically not available for processors.

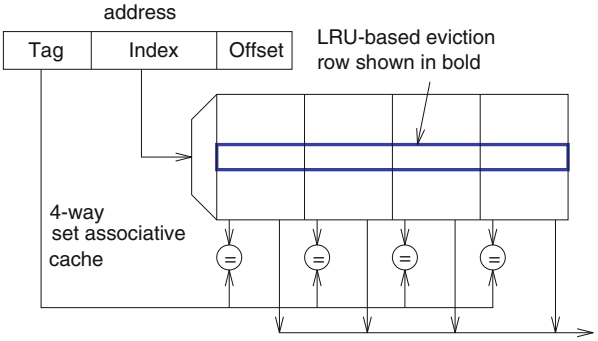

Fig. 5.6 Set associative cache (for $n = 4$)

Fig. 5.7 Access to variable e makes it the youngest

Fig. 5.8 Access to variable c causes eviction of f

Furthermore, consider control flow joins. What do we know about the content of the partial cache after the join?

We must distinguish between *may*- and *must*-information and the corresponding analysis. Must-analysis reveals the entries which **must** be in the cache. This information is useful for computing the WCET. May-analysis identifies the entries which **may** be in the cache. This information is typically used to conclude that certain information will definitely not be in the cache. This knowledge is then exploited during the computation of the BCET.

As an example of must- and may-analysis, we consider must information at control flow joins. Figure 5.9 shows the corresponding situation. In Fig. 5.9, memory object c is assumed to be the youngest object for one path to the join and a is assumed to be the youngest object for the other path. The age of the other entries is defined accordingly. What do we know about the "worst" case after the join? A certain entry is guaranteed to be in the cache only if it is guaranteed to be in the cache for both paths. This means that the **intersection** of the memory objects defines the result of the must-analysis after the join. As a worst case, we must assume the **maximum of the ages** along the two paths. Figure 5.9 shows the result. This analysis uses sets of entries for each cache way.

Now, consider may-analysis for control flow joins. Figure 5.10 depicts the situation. Some object being in the cache on either of the two paths to the join *may* be in the cache after the join. Hence, the set of objects which may be in the cache

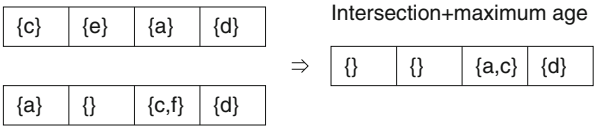

Fig. 5.9 Must-analysis at program joins for LRU caches

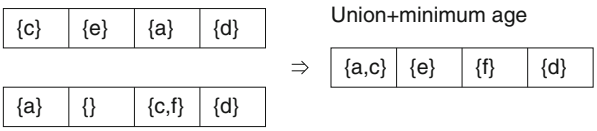

Fig. 5.10 May-analysis at program joins for LRU caches

after the join consists of the **union** of the objects that were in the cache before the join. As a best case, we use the **minimum of the ages** before the join. Figure 5.10 shows the result.

Static analyses also comprise pipeline analysis. Pipeline analysis has to compute safe bounds on the number of cycles required to execute code in the machine pipeline. Details of pipeline analysis are explained by Hahn et al. [196] and Thesing [534]. The result of static analyses consists of bounds on the execution times for each of the basic blocks of a program. Results are written to the PER-file shown in Fig. 5.5.

aiT's next phase exploits these bounds to derive WCET_{EST} values for the entire program, using an **integer linear programming** (ILP) model (see p. 393), comprising two types of information:

- **The objective function**: In our application of ILP modeling, this function represents the overall execution time. This time is calculated as

$$WCET_{EST} = \sum_{\text{basic blocks}} e_i * f_i \tag{5.3}$$

  where e_i is the worst case execution time of basic block i (as computed during static analysis) and f_i its worst case execution count. Only some of these counts can be determined automatically, and additional designer-provided information, e.g., about loop bounds, may be required.
- **Linear constraints**: These reflect the structure of the control flow graph.

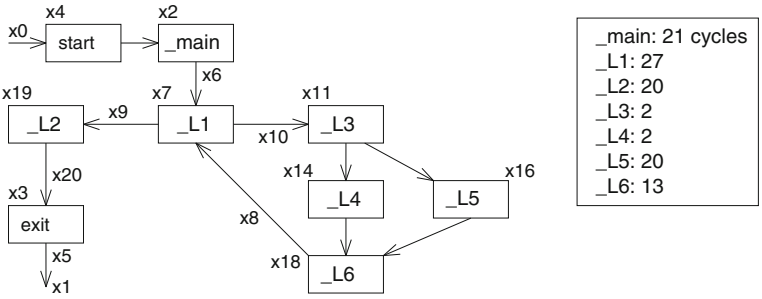

Fig. 5.11 Sample program: **left**: extended control flow graph; **right**: WCET_{EST} of basic blocks

Example 5.2 Let us consider the simple code shown next:

```
int main() { int i,j=0;
    _Pragma("loopbound min 100 max 100")    /* hint for bound analysis */
   for (i=0; i <100; i++) {
     if (i<50) j+=i;
     else j+=(i*13) % 42;
   }
   return j;
}
```

Figure 5.11 (left) shows the control flow graph (CFG) corresponding to this small program. This graph is extended by additional start and exit nodes. Node _L1 reflects the **for**-testing, _L3 the **if**-testing, _L4 and _L5 the two cases of the **if**-statement, and _L6 its join operation. Variables x0 to x20 denote the number of executions of the blocks and the number of transitions between blocks. For example, we are transitioning from node main into node _L1 x6 times and are executing the target node x7 times. We assume that the analysis of the WCET for each of the basic blocks has resulted in the list shown on the right of Fig. 5.11. The following is a partial list of the ILP constraints:

```
01: 21 x2 + 27 x7 + 2 x11 + 2 x14 + 20 x16 + 13 x18 + 20 x19; /*objective*/
02: x7 - x8 - x6 = 0;       /* Constraint for flow entering CFG node _L1 */
03: x7 - x9 - x10 = 0;      /* Constraint for flow leaving CFG node _L1 */
04: x7 - 101 x9 >= 0;         /* Constraint for lower loop bound of _L1 */
05: x7 - 101 x9 <= 0; ...     /* Constraint for upper loop bound of _L1 */
06: x0 - x4 = 0;                                /* CFG Start Constraint */
07: x2 - x4 = 0;          /* Constraint for flow entering function _main */
08: x2 - x6 = 0;          /* Constraint for flow leaving CFG node _main */
09: ...
```

Line 01 contains the cost function. All other lines model constraints reflecting the structure of the graph. Consider, for example, node _L1. Constraints for this node are shown in lines 02 and 03. The number of times that we are branching into the node (x6+x8) is equal to its number of executions (x7). The number of times that we are leaving from the node (x9+x10) is also equal to its number of executions. Lines 04 and 05 reflect the number of loop iterations. This number is taken from the pragma in the code. Line 06 describes the fact that node start is executed exactly as many times as we are branching into the code. The other lines are reflecting the structure in a similar way. ∇

The ILP problem can be solved with some standard ILP solver. Maximizing the objective function yields a safe upper bound on the WCET.

This technique for modeling execution time is called **implicit path enumeration** (IPET) [343], since the problem of enumerating the potentially large number of execution paths is avoided.

aiT visualizes the results as annotated control flow graphs. The designer could optimize the system under design by exploiting these graphs. Due to the presented approach, aiT has limitations: preemption by other processes, interrupts, input/output, and direct memory transfers (DMA) are not supported.

Only few approaches exist for the WCET analysis of multi-cores [264, 265, 286]. New probabilistic approaches [2] aim at complementing available methods. They are usually based on extreme value theory [123].

5.2.3 Real-Time Calculus

WCET estimates allow us to predict the execution of some algorithm for a single input event. However, the overall goal is more comprehensive. Overall, we should make sure that our hardware platform is capable of processing streams of events in a timely manner (which may be important for some parts of the Internet of Things).

This can be checked with Thiele's **real-time calculus** (**RTC**). This calculus (**RTC**) is based on the description of the rate of incoming events.[3] This description also includes fluctuations of this rate. Toward this end, the timing characteristics of a sequence (or stream) of events are represented by a tuple of *arrival curves*:

$$\overline{\alpha}^{\,u}(\Delta), \overline{\alpha}^{\,l}(\Delta) \in \mathbb{R} \geq 0, \Delta \in \mathbb{R} \geq 0$$

These curves represent the maximal resp. the minimal number of events arriving within a time interval of length Δ. There are at most $\overline{\alpha}^{\,u}(\Delta)$ and at least $\overline{\alpha}^{\,l}(\Delta)$

[3] Our presentation of the real-time calculus is based on Thiele's presentation in the book edited by Zurawski [536]. Resulting considerations at the system level have been called *modular performance analysis* (MPA).

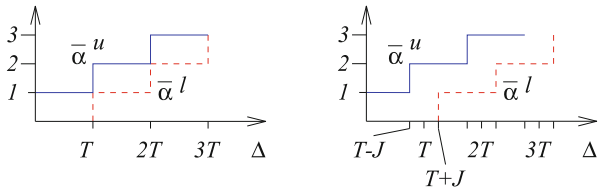

Fig. 5.12 Arrival curves: **left**: periodic stream; **right**: periodic stream with jitter J

events arriving within the time interval $(t, t+\Delta)$ for all $t \geq 0$. Figure 5.12 shows the number of possibly arriving events for some possible models of arriving events. For example, in the case of periodic event streams with period T, there is a maximum of a single event happening in time interval $(0, T)$.[4] Similarly, there is an upper bound of two events within time interval $(T, 2T)$. Now, let us consider the lower bound for time interval $(0, T)$. There is possibly not a single event in this interval. Hence, the bound is zero. For time interval $(T, 2T)$, there has to be at least one event. Therefore, the bound is one. So, for $\Delta = 0.5T$, there will be at least zero and at most one incoming event (see Fig. 5.12 (left)). In the case of periodic event streams with jitter J, these curves are shifted by this amount (see Fig. 5.12 (right)). The upper bound is shifted to the left; the lower bound is shifted to the right. The jitter is assumed not to be accumulating.

We are using bars on top of symbols (like $\overline{\alpha}$) for all entities referring to incoming events.

Available computational and communication service capacity can be described by *service functions*:

$$\beta^u(\Delta), \beta^l(\Delta) \in \mathbb{R} \geq 0, \Delta \in \mathbb{R} \geq 0$$

These functions allow us to model situations in which the available service capacity is fluctuating. Figure 5.13 shows the communication capacity of some *time division multiple access* (TDMA) bus (see p. 176). Allocation is done periodically with a period of T. Bus arbitration allocates this bus during a time window s time units long. During this window, the bus achieves a bandwidth of b. The upper bound is obtained if the bus is allocated exactly at the time we are starting our observation. The transferred amount is then increasing linearly. The lower bound is obtained if the bus was just deallocated when we started our observation of length Δ. Then we must wait $T - s$ time units until the bus gets allocated again.

Separate methods are required to determine $\overline{\alpha}$ and β for streams of ("external") events arriving at the system to be modeled. Their computation is not part of RTC. In contrast, bounds for events generated within the system are derived by the calculus (see below).

[4] We leave out the subtle discussion of discontinuities at $\Delta = n * T$.

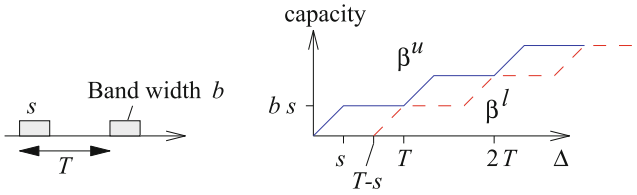

Fig. 5.13 Service functions for a TDMA bus

Up till now, there is no information about the **workload** required by each of the incoming events. This workload is represented by additional functions $\gamma^u(e), \gamma^l(e) \in \mathbb{R} \geq 0$ for each sequence e of incoming events. This information can be derived from bounds on the execution time of code required for each of the events. Figure 5.14 shows an example of such functions. This example is based on the assumption that between three and four time units are required for processing a single event. Accordingly, the workload for a single event varies between three and four time units, the work load for two events varies between six and eight time units, etc. The dashed lines are not part of the function, since it is defined only for an integer number of events. The work load resulting from an incoming stream of events can now be easily computed. Upper and lower bounds are characterized by the functions

$$\alpha^u(\Delta) = \gamma^u(\overline{\alpha}^u(\Delta)) \text{ and} \tag{5.4}$$

$$\alpha^l(\Delta) = \gamma^l(\overline{\alpha}^l(\Delta)) \tag{5.5}$$

There should be enough computational or communication capacity to handle this workload. The number of events which can be processed with the available computational capacity can be computed as

$$\overline{\beta}^u(\Delta) = (\gamma^l)^{-1}(\beta^u(\Delta)) \text{ and} \tag{5.6}$$

$$\overline{\beta}^l(\Delta) = (\gamma^u)^{-1}(\beta^l(\Delta)) \tag{5.7}$$

Equations (5.6) and (5.7) use the inverse of functions γ^u and γ^l to convert bounds on the available capacity (measured in real time units) into bounds measured in terms of the number of events that can be processed.

Based on this information, it is possible to derive the properties of outgoing streams of events from incoming streams of events. Suppose the incoming stream is characterized by bounds $[\overline{\alpha}^l, \overline{\alpha}^u]$. We can then compute characteristics of the outgoing streams such as the corresponding bounds $[\overline{\alpha}^{l'}, \overline{\alpha}^{u'}]$ of the outgoing stream of events and the remaining service capacity, available for other tasks. This remaining capacity is derived by transforming *service curves* $[\overline{\beta}^l, \overline{\beta}^u]$ into *service curves* $[\overline{\beta}^{l'}, \overline{\beta}^{u'}]$ (see Fig. 5.15). This remaining service capacity can be employed for lower-priority tasks to be executed on the same processor.

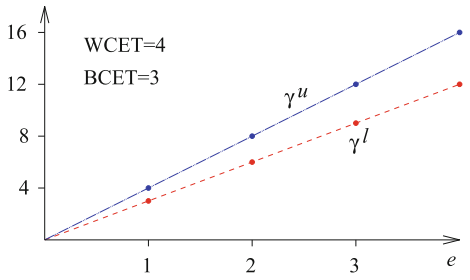

Fig. 5.14 Workload characterization ($WCET_{EST}$ may be used instead of WCET)

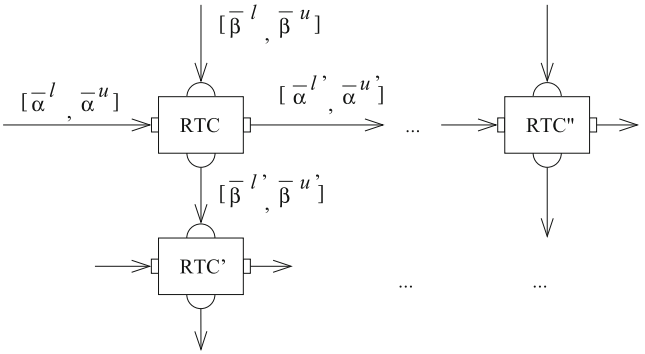

Fig. 5.15 Transformation of event stream and service capacities by real-time components

According to Thiele et al., outgoing streams and remaining service capacities are bounded by the following functions [536]:

$$\overline{\alpha}^{u'} = [(\overline{\alpha}^u \underline{\otimes} \overline{\beta}^u) \overline{\oslash} \overline{\beta}^l] \wedge \overline{\beta}^u \tag{5.8}$$

$$\overline{\alpha}^{l'} = [(\overline{\alpha}^l \overline{\oslash} \overline{\beta}^u) \underline{\otimes} \overline{\beta}^l] \wedge \overline{\beta}^l \tag{5.9}$$

$$\overline{\beta}^{u'} = (\overline{\beta}^u - \overline{\alpha}^l) \underline{\oslash} 0 \tag{5.10}$$

$$\overline{\beta}^{l'} = (\overline{\beta}^l - \overline{\alpha}^u) \overline{\otimes} 0 \tag{5.11}$$

Operators used in these equations are defined as follows:

$$(f \underline{\otimes} g)(t) = inf_{0 \le u \le t}\{f(t-u) + g(u)\} \tag{5.12}$$

$$(f \overline{\otimes} g)(t) = sup_{0 \le u \le t}\{f(t-u) + g(u)\} \tag{5.13}$$

$$(f \overline{\oslash} g)(t) = sup_{u \ge 0}\{f(t+u) - g(u)\} \tag{5.14}$$

$$(f \underline{\oslash} g)(t) = inf_{u \ge 0}\{f(t+u) - g(u)\} \tag{5.15}$$

$\wedge$ denotes the minimum operator.

In essence, these equations characterize outgoing streams and capacities. These equations have been adopted from communications theory. Proofs regarding these equations are provided by Network Calculus [327]. The easiest way of using these equations is to download a MATLAB® toolbox [561].

The same theory also allows to compute the delay caused by the real-time components as well as the size of the buffer required to temporarily store incoming/outgoing events. This way, performance and other characteristics of the system can be computed from information about the components.

A second performance analysis method has been proposed by Henia and Ernst et al. In this so-called SymTA/S approach [210], the different curves in Thiele's approach are replaced by standard models of event streams such as periodic event streams, periodic event streams with random jitter, and periodic event streams with bursts. SymTA/S explicitly supports the combination and integration of different kinds of analysis techniques known from real-time research.

5.3 Quality Metrics

5.3.1 Approximate Computing

Sometimes, computing the best possible output of some algorithm requires a significant amount of resources (in terms of computing time, energy, thermal headroom, etc.). For some applications, the best possible output is not actually needed, since minor degradations will possibly not even be recognized by users. This can be exploited in a resource-constrained environment in order to trade off the quality of the output against needed resources. A certain deviation of the actual output from the best possible output is accepted, for example, for lossy audio, video, and image encoding. This leads us to consider **approximate computing**.

Definition 5.11 Computing which tolerates a certain deviation of generated output of some algorithm from the best possible result is called **approximate computing** [397].

With approximate computing, it is necessary to consider the quality of the generated output as one of the objectives. Unfortunately, it is not easy to evaluate the quality of some generated result, and several metrics can be used.

5.3.2 Simple Criteria of Quality

Some simple metrics can be applied whenever the true (or the best possible) output is known. Suppose that $x_1, \ldots, x_n$ are n samples of some signal x in discrete time.

Furthermore, suppose that instead of the real (or the best possible) values $x_1, \ldots, x_n$ we measure or compute approximate values $y_1, \ldots, y_n$.

Then, our first metric, the mean-squared error (MSE), is defined as follows:

Definition 5.12 The **mean-squared error** (MSE) is defined as

$$MSE(x, y) = \frac{1}{n} \sum_{i=1}^{n} (x_i - y_i)^2 \tag{5.16}$$

The second metric is the root-mean-squared error.

Definition 5.13 The **root-mean-squared error** (RMSE) is defined as

$$RMSE(x, y) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - y_i)^2} \tag{5.17}$$

RMSE has the same dimension as the difference between the actual and the real value, but it should not be confused with the "average error" which is defined next:

Definition 5.14 The **mean absolute error** (MAE) is defined as

$$MAE(x, y) = \frac{1}{n} \sum_{i=1}^{n} |x_i - y_i| \tag{5.18}$$

For identical deviations of the measured signal y from real values x, the MAE is equal to the RMSE. However, the RMSE emphasizes large deviations between real and measured values (so-called outliers).

The signal-to-noise ratio (SNR) was already defined on p. 142. Next, we define the peak signal-to-noise ratio, which is similar to the SNR. Let x be a signal, x_{max} its maximum, and y its noisy approximation.

Definition 5.15 The **peak signal-to-noise ratio** (PSNR) is defined as

$$PSNR(x, y) = 10 \log_{10} \left(\frac{x_{max}^2}{MSE(x, y)} \right) \tag{5.19}$$

$$= 20 \log_{10} \left(\frac{x_{max}}{RMSE(x, y)} \right) \tag{5.20}$$

The PSNR, just like the SNR, is measured in decibels (dB).

The above values are easy to compute, but they are agnostic of the impression which humans might have of certain errors [315]. It is known that certain deviations between real and computed signal values are hardly noticed by humans. This is the foundation of lossy coding techniques such as MP3, JPEG, or digital TV standards. None of the metrics presented so far reflects the impression of deviations by humans.

Next, we will present the **universal image quality index** (UIQI) [562]. This index tries to capture changes in the structure of images, since the human eye is very sensitive to it. We will present the computation of this index for gray-scale images. Several values need to be computed [315]:

$$\mu_x = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{5.21}$$

$$\mu_y = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{5.22}$$

$$\ell(x, y) = \frac{2\mu_x\mu_y}{\mu_x^2 + \mu_y^2} \tag{5.23}$$

Equations (5.21) and (5.22) compute the average brightness of each of the images, and these averages are used to compute $\ell(x, y)$. For images of the same average brightness, $\ell(x, y)$ will be equal to 1. Otherwise, this value will be less than 1.

Furthermore, we consider variances. Equations (5.24) and (5.25) compute the contrast of each of the images, and these averages are used to compute $c(x, y)$:

$$\sigma_x = \sqrt{\frac{1}{(n-1)}\sum_{i=1}^{n}(x_i - \mu_x)^2} \tag{5.24}$$

$$\sigma_y = \sqrt{\frac{1}{(n-1)}\sum_{i=1}^{n}(y_i - \mu_y)^2} \tag{5.25}$$

$$c(x, y) = \frac{2\sigma_x\sigma_y}{\sigma_x^2 + \sigma_y^2} \tag{5.26}$$

For images of the same average contrast, $c(x, y)$ will be equal to 1. Otherwise, this value will be less than 1. Equation (5.27) computes the cross-correlation of the two images:

$$\sigma_{x,y} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y) \tag{5.27}$$

$$s(x, y) = \frac{\sigma_{x,y}}{\sigma_x\sigma_y} \tag{5.28}$$

Positive values of $s(x, y)$ as computed from Eq. (5.28) correspond to a good correlation of the two images; negative values correspond to an inverse correlation.

An overall quality index is then computed by Eq. (5.29):

$$Q(x, y) = \frac{2\mu_x\mu_y}{\mu_x^2 + \mu_y^2} * \frac{2\sigma_x\sigma_y}{\sigma_x^2 + \sigma_y^2} * \frac{\sigma_{x,y}}{\sigma_x\sigma_y} \tag{5.29}$$

$Q = 1$ for identical images, and Q will be negative for inversely correlated images.

It does not make sense to consider the correlation of images globally, since some inverse correlation in a particular block will already provide a negative impression about the image. Hence, Eq. (5.29) is computed only for blocks of pixels. The global UIQI value takes the values of Q for the different blocks into account.

The structural similarity index measure (SSIM) [563] is an extension of the UIQI objective.

Kühn compared the different metrics and found that none of these is really superior to others [315]. He recommends that several of these metrics should be computed and a careful comparison should be performed in practice. An overview over some useful objectives is also provided by Mittal [397].

In digital communications, the bit error ratio (BER) is an important metric.

Definition 5.16 The **bit error ratio (BER)** is ratio of the number of bit errors divided by total number of communicated bits.

5.3.3 Criteria for Data Analysis

Sensors are typically not ideal in sense that some readouts deviate from the real values. Furthermore, it may be necessary to fuse data generated by various sensors. Hence, it is necessary to use data analysis techniques, e.g., machine learning (see p. 15). Generated results will not always be correct as well, either because sensor readouts were already compromised or due to imperfect data analysis techniques. In a way, we are dealing with approximate computing even though this term was not used in this context.

For data analysis, classification of objects is a very frequent goal. Let X be a set of objects which we would like to classify. Suppose that we restrict ourselves to binary classification.

Example 5.3 For example, consider the case of searching for amber at a beach. Unfortunately, white phosphorus as a leftover from bombs found, e.g., at the Baltic ocean, looks very much like amber but starts to suddenly burn at 1300 °C when it dries. Classifying some found objects as either amber or phosphorus is thus a very delicate task (and hence, inexperienced people should not touch such objects anyway). ∇

In this context, four cases are possible:

- **True positives** (TP): we classify some object as amber, and it is actually valuable amber.

- **False positive** (FP): we classify some object as amber, and it is actually dangerous.
- **True negative** (TN): we classify some object as dangerous and it is actually dangerous.
- **False negative** (FN): we classify some object as dangerous, and it is actually valuable amber.

Absolute numbers have to be related to each other. Hence, the following metrics have been defined:

Definition 5.17 The **precision** p is defined as the fraction

$$p = \frac{TP}{TP + FP} \tag{5.30}$$

In the case of searching for amber, we aim at a precision of 1, since we do not want to get burnt.

Definition 5.18 The **recall** r (or **sensitivity**) is defined as the fraction

$$r = \frac{TP}{TP + FN} \tag{5.31}$$

In order to obtain a good precision, we will have to accept some false negatives (e.g., amber classified as phosphorus).

Definition 5.19 The **accuracy** acc is defined as the fraction

$$acc = \frac{TP + TN}{TP + FP + TN + FN} \tag{5.32}$$

In the case of searching for amber, we might tolerate a non-optimal accuracy, due to the importance of keeping false positives as close to zero as possible, and, hence, we might have several false negatives.

Definition 5.20 The ***specificity*** is defined as the fraction

$$specificity = \frac{TN}{TN + FP} \tag{5.33}$$

Definition 5.21 The ***F*1 score** or ***F-measure*** is defined as the harmonic mean of precision and recall:

$$F1 = 2\,\frac{p * r}{p + r} \tag{5.34}$$

In a more general context, the **quality of service** (QoS) is another well-known metric. Frequently, it is related to the quality of communication channels, where bit error rates, latency, and bandwidth are indicators of quality.

In an even wider sense, we may also consider not just those technical parameters but also the overall experience for the user. This is captured in the **quality of experience** (QoE) metric, which refers to the overall user experience including all aspects which might be considered by a user. There is a number of metrics which can be used to estimate the overall quality of experience [400].

5.4 Energy and Power Models

5.4.1 General Properties

Energy models and **power models** are essential for evaluating the corresponding objectives. Such models are needed for optimizations aiming at a reduction of power and energy consumptions. They are also required for optimizations trying to reduce operating temperatures and to improve reliability. Power estimation is used in **power management** algorithms (see p. 373).

Energy and power models are closely related, as can be seen from Eq. (3.13). Energy can be computed as the integral of power over time. Once the energy consumption is known, we can compute the average power consumption. In general, we can use:

1. **Measurements on real hardware**: measurements can be very precise, but they apply only to the hardware at hands. Measuring voltages is typically rather easy and does not require complex procedures.

   Measuring currents can be done with a current clamp or a shunt resistor.

   - **Current clamps** have to enclose one of the wires of the power supply cable. They measure the magnetic field caused by the current flowing through the cable. The advantage of this approach is that no power wires have to be broken and power will remain connected unchanged to the device being analyzed. The disadvantage is that current clamps do not allow precise measurements.
   - Using an **ammeter** typically results in a better precision. However, an insertion of the ammeter directly into the power line has some disadvantages. For example, the system is unpowered if we remove the ammeter. Also, long cables might add noise. Therefore, it is typically preferable if we include a shunt resistor. A typical circuit containing a **shunt** is shown in Fig. 5.16 (left).

     The advantage of using a shunt resistor over using a simple ammeter is that the shunt can be integrated into the power wires. Due to the shunt resistor, currents flowing into the device under test will cause a voltage drop across the shunt, and this voltage can be measured and used to compute the current from Ohm's law. Finding the right resistance of the shunt is an issue. If the resistance is too large, the device under test will be powered with a voltage lower than the original voltage and might even fail to work. If the resistance is too small, the voltage across the shunt will be too small to be reliably

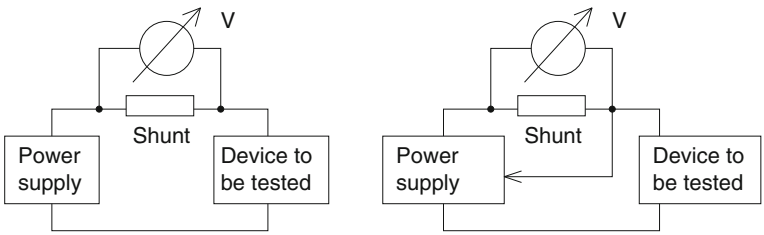

Fig. 5.16 Measuring current: **left**, two-wire connection; **right**, feedback into voltage regulator

measured and will be subject to a substantial amount of noise. Selecting the right resistance depends on the current flowing into the device under test. If this current varies substantially, it may even be necessary to employ several shunt resistors and switch between them, depending on the current actually flowing. The problem regarding the voltage drop can be partially avoided when regulated power supplies are used and the regulator feedback input can be connected to the voltage actually powering the device (see Fig. 5.16 (right)). The power supply would then try to keep the voltage at the device at its nominal level. However, the voltage across the shunt is affected by the current flowing back into the voltage regulator input.

Unfortunately, there will not be a separate power pin or wire for every component within the device and we can compute only a lumped sum of currents drawn by the device. We may have to stimulate the device in a particular way in order to get information about the consumption of the different components.

- **Models** can be used even when real hardware is not available, but they can be very imprecise. Models have to be validated; otherwise they would remain very questionable. Two validation methods can be found for many of the available power and energy models: either models are validated against more detailed models at a lower level of abstraction, or they are compared with measurement for real devices, resulting in a hybrid model. Validation against measurements requires a method for selecting model parameters. Frequently, linear models are selected, and parameters are selected with using the least square method (minimizing the MSE as per Eq. (5.16)). Curve fitting with this method is typically available in mathematical tool boxes such as MATLAB®. More recently, using machine learning for this purpose became more preferable. For example, Falkenberg et al. [161] used machine learning for modeling the power consumption of transmitters in mobile phones.

There is no one-approach-fits-all solution for energy consumption modeling. Instead, the usual approach is to combine ideas for modeling to fit the needs at hand. Therefore, we will present representative examples of power models and hope that the reader will identify the combination of methods which fits his/her constraints best.

5.4.2 Energy Model for Memories

As described in the section on memory hardware (see p. 168), the energy consumption of caches and other memories can be computed with CACTI [408, 589]. CACTI assumes an abstract layout of the memory, extracts capacitances from the layout, and computes access times, cycle times, area, leakage, and dynamic power consumption from this information. CACTI has been validated against models of the same memories at a more detailed level, employing SPICE [519] as the solver at that level. Currently (in 2020), the most recent version of CACTI (version 6.5) is available from http://www.hpl.hp.com/research/cacti/.[5] Recent enhancements include detailed modeling of the interconnect and modeling of non-uniform memory accesses. Models of transmitters and sense amplifiers have been included. Also, used architectural and technological parameters can be specified.

5.4.3 Energy Model for Instructions

One of the first power models was proposed by Tiwari [542]. The model includes so-called base costs and inter-instruction costs. Base costs of an instruction correspond to the energy consumed per instruction execution if an infinite sequence of instances of that instruction is executed. Base costs have been computed by running programs consisting of 120 identical instructions and a branch back to the beginning of this sequence. Programs are designed such that no stall cycles appear. This may require the adding of no-operation instructions and some simple calculations to eliminate their contribution to the energy consumption.

Inter-instruction costs model the additional energy consumed by the processor if instructions change. This additional energy is required, for example, due to switching functional units on and off. Inter-instruction costs reflect the impact of the initial circuit state on the overall energy consumption of an instruction. These costs can be computed by running programs containing an alternating sequence of instructions pairs.

Base costs and inter-instruction costs are computed for a program not generating any cache misses. The effect of cache misses has to be added to these two costs. This requires the knowledge of the cache miss ratio and the memory access energy. The memory energy depends on the addresses accessed. No attempt is made to statically predict memory addresses. Hence, this contribution can only be determined dynamically, during the execution of the program.

The model has been applied to two real systems, an Intel 486 DX2 and a Fujitsu SPARClite 934. Measurements of the currents have been used to calibrate the model.

[5] It is recommended to use this URL, since there are several tools with the same name. Currently, a modifiable C++-version is available. Previously available web interfaces do not exist any longer.

5.4.4 Energy Model for Functional Processor Units

The Wattch power estimation tool [70] estimates the power consumption of microprocessor systems at the architectural level. Wattch uses the SimpleScalar simulator to simulate processors. SimpleScalar can be configured to model the processor at hand as closely as possible. The number of pipeline stages and functional units is typically correctly modeled, whereas some more specialized features are possibly not. Wattch is based on detailed information on the energy consumption of the different components which we could find in a microprocessor. While running, SimpleScalar keeps track of invoked functional units. Wattch exploits this information in order to compute an overall energy consumption.

Wattch requires much more information about the architecture than Tiwari's instruction-set level approach. For example, Wattch includes its own detailed model of the energy consumption in memories. Also, clocking is taken explicitly into account, including conditional clocking if clock gating is used. In the original paper [70], results have been validated for three different processors.

5.4.5 Energy Model for Processor and Memory

The level of details of the model by Steinke et al. [510] lies between that of Tiwari and that of Wattch. For instructions and for data, the model considers the sum of the energies consumed in the CPU and the memory:

$$E_{total} = E_{cpu\_instr} + E_{cpu\_data} + E_{mem\_instr} + E_{mem\_data} \tag{5.35}$$

Each of the four terms is then computed from detailed equations. The following notation is used in these equations: m is the number of instructions considered, $w(b)$ returns the number of ones in its argument (either code or data), $h(b_1, b_2)$ returns the Hamming distance between its two arguments, dir denotes the direction of data transfer, and α_i and β_i ($i \in \{1..10\}$) are constants computed from curve fitting of measured energies. Using this notation, E_{cpu_data} can be computed as follows:

$$\begin{aligned} E_{cpu\_data} = \sum_{i=1}^{m} \big\{ & \alpha_5 * w(DAddr_i) + \beta_5 * h(DAddr_{i-1}, DAddr_i) \\ & + \alpha_{6,dir} * w(Data_i) + \beta_{6,dir} * h(Data_{i-1}, Data_i) \big\} \end{aligned} \tag{5.36}$$

where $Data_i$ is the data value used in instruction i, and $DAddr_i$ is its address.

Furthermore, consider E_{mem_data}, a term which is relevant only when the data is actually loaded from the main memory:

$$
\begin{aligned}
E_{mem\_data} = \sum_{i=1}^{m} \{ & BaseMem(DataMem, dir, Word\_width) \\
& + \alpha_9 * w(DAddr_i) + \beta_9 * h(DAddr_{i-1}, DAddr_i) \\
& + \alpha_{10,dir} * w(Data_i) + \beta_{10,dir} * h(Data_{i-1}, Data_i) \}
\end{aligned} \tag{5.37}
$$

where $BaseMem$ is the base cost for accessing a memory object of a particular width in direction dir.

E_{mem_instr} can be computed as follows:

$$
\begin{aligned}
E_{mem\_instr} = \sum_{i=1}^{m} \{ & BaseMem(InstrMem, Word\_width_i) \\
& + \alpha_7 * w(IAddr_i) + \beta_7 * h(IAddr_{i-1}, IAddr_i) \\
& + \alpha_8 * w(IData_i) + \beta_8 * h(IData_{i-1}, IData_i) \}
\end{aligned} \tag{5.38}
$$

where $BaseMem$ is the base cost for accessing a memory word of a particular width from the instruction memory, $IAddr_i$ is the address of the instruction, and $IData_i$ is instruction i itself.

E_{cpu_instr} can be computed from the following equation:

$$
\begin{aligned}
E_{cpu\_instr} = \sum_{i=1}^{m} \{ & BaseCPU(Opcode_i) + FUChange(Instr_{i-1}, Instr_i) \\
& + \alpha_4 * w(IAddr_i) + \beta_4 * h(IAddr_{i-1}, IAddr_i) \\
& + \sum_{j=1}^{s} \left(\alpha_1 * w(Imm_{i,j}) + \beta_1 * h(Imm_{i-1,j}, Imm_{i,j}) \right) \\
& + \sum_{k=1}^{t} \left(\alpha_2 * w(Reg_{i,k}) + \beta_2 * h(Reg_{i-1,k}, Reg_{i,k}) \right) \\
& + \sum_{k=1}^{t} \left(\alpha_3 * w(RegVal_{i,k}) + \beta_3 * h(RegVal_{i-1,k}, RegVal_{i,k}) \right) \}
\end{aligned} \tag{5.39}
$$

where $BaseCPU$ is the base cost for $Opcode_i$, $FUChange(..)$ reflects the costs caused by the transition from instruction $i-1$ to i, Imm reflects the impact of up to s immediate values per instruction, Reg reflects the register numbers of up to t registers per instruction, and $RegVal$ reflects up to t register values per instruction.

To determine constants, dedicated code sequences have to be designed in order to attribute energy consumption to particular terms of the equations.

Example 5.4 The following code sequence allows measuring the energy required for executing a load word instruction:

```
start: lw R1, address                                    /* load word */
       lw R1, address                                    /* load word */
       ...                        /* lw instruction repeated 50-100 times */
       bra start                                  /* back to the start */
```

The impact of the branch back to the beginning on the energy consumption can be neglected. The impact of different addresses, register numbers, and register content can be studied by varying these values. For example, we can initially set all these values to zero and then incrementally study the impact of additional ones. ∇

In our own experiments, constants were determined by running a linear regression method on the data. A significant impact of the number of ones in the data was found, which would have been unnoticed for Tiwari's model.

5.4.6 *Energy Model for an Application*

The Odroid-XU3 [202] platform (see Fig. 5.17) comprises current sensors. The sensors enable precise measurement of the consumed power during the execution of applications, measuring the consumption of ARM® big cores, little cores, GPU, and DRAM individually. This possibility is exploited by several researchers. For example, Neugebauer et al. [416] have integrated Odroid-XU3 processors into their design space exploration for one application. Hence, design space exploration is based on a realistic analysis of the consumed energy. This approach eliminates the use of models of unknown precision. The overall approach for design space exploration enabled by the XU3 is shown in Fig. 5.18.

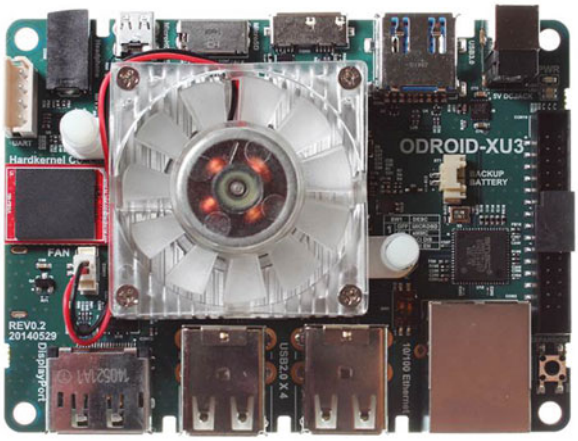

Fig. 5.17 Odroid-XU3

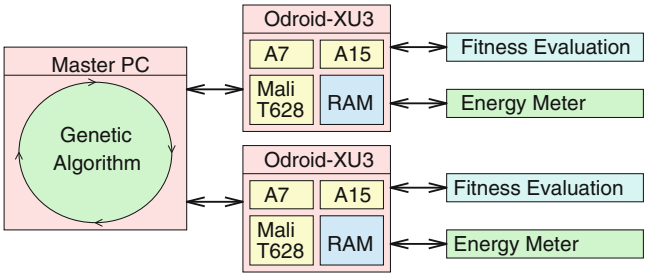

Fig. 5.18 Evolutionary algorithm, fitness estimation based on real measurements

The design space exploration is based on a genetic algorithm. The evaluation of a particular solution is based on real execution of the code on an XU3. The resulting optimized algorithm has been used by Neugebauer et al. [417] within the cyber-physical system PAMONO which is capable of detecting bio-viruses online. It is based on the physical so-called Plasmon effect of visualization of small objects. Unfortunately, the Odroid-XU3 has been discontinued and replaced by the XU4 not including current sensors.

5.4.7 Energy Model for Multiple Applications with Hardware Multithreading

Kerrison and Eder analyzed the energy consumption of the XMOS XS1-L multi-threaded processor design for real-time applications [290]. One of the particular features of that processor is its hardware-supported multithreading: it performs fast context switching between four threads in hardware. One of the research questions was: how much does the hardware context switching between threads cost? Due to the availability of real hardware, this question could be answered with real measurements. The power consumed by the XMOS XS1-L was measured with a shunt resistor inserted into its power cable, and the resistor was connected to an INA219 power measurement chip (see http://www.ti.com/product/ina219). The software running on the processor was controlled from a second processor. It turned out that the best energy efficiency was reached when all four hardware threads are used. However, hardware multithreading leads to many charging/discharging operations and a corresponding energy consumption. The interesting experimental results include an analysis of the impact of executed instructions on the energy consumption, as shown in Fig. 5.19 for the case of 8 bit data.

Figure 5.20 displays the corresponding information for the case of 16 bit data.

The two dimensions of the diagrams encode the applications which are run in the odd and even threads, respectively. In these figures, a change in the number of operands is indicated by dashed lines. Instructions with three or more operands are

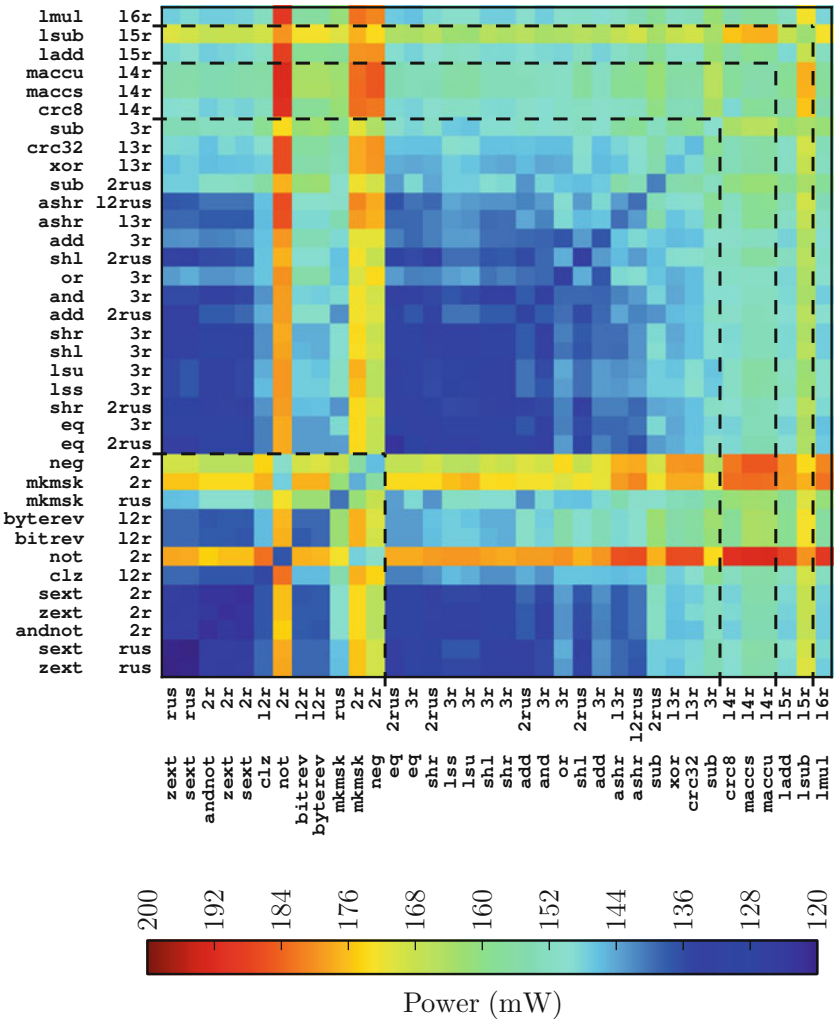

Fig. 5.19 Power analysis for multithreading for 8 bit data, top, power consumption as a function of instructions on 8 bit data executed in the even threads (vertical axis) and in the odd threads (horizontal axis) ©Kerrison, Eder; bottom, color encoding of temperatures

shown at the top and at the right end of each diagram. Obviously, the consumed energy increases with the number of operands. Figure 5.20 demonstrates that processing 16 bit data requires more energy than processing 8 bit data. Kerrison et al. use these results in order to optimize embedded software.

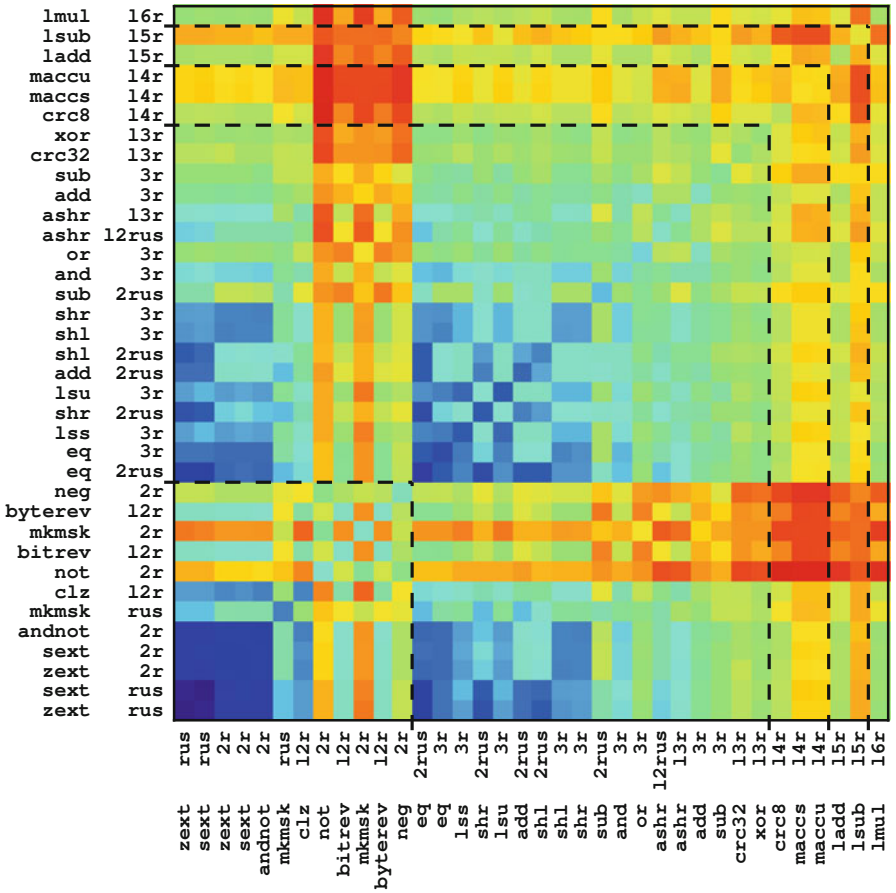

Fig. 5.20 Power analysis for multithreading for 16 bit data, power consumption as a function of instructions on 16 bit data executed in the even threads (vertical axis) and in the odd threads (horizontal axis); temperature encoding as in Fig. 5.19 (bottom) ©Kerrison, Eder

5.4.8 Energy Model for an Android Mobile Phone

Zhang et al. [612] describe a power model construction technique for an HTC Android phone, called PowerBooter. Their technique uses the following equation:

$$\begin{aligned} E = &(\beta_{uh} * freq_h + \beta_{ul} * freq_l) * util + \beta_{CPU} * CPU_{on} \\ &+\beta_{br} * brightness + \beta_{Gon} * GPS\_on + \beta_{Gsl} * GPS\_sl \\ &+\beta_{WiFi\_l} * WiFi_l + \beta_{WiFi\_h} * WiFi_h + \beta_{3G\_idle} * 3G_{idle} \\ &+\beta_{3G\_FACH} * 3G_{FACH} + \beta_{3G\_DCH} * 3G_{DCH} \end{aligned} \tag{5.40}$$

where

$\beta_{..}$: constants to be determined
$freq_i$: CPU frequencies
$util$: CPU utilization
CPU_{on} : refers to processor utilization
$brightness$: takes illumination into account
$GPS_{..}$: relates to GPS usage
$WiFi_l$: amount of time, Wi-Fi is in low-speed mode
$WiFi_h$: amount of time, Wi-Fi is in high-speed mode
$3G_{3G\_idle}$: amount of time, 3G is idle
$3G_{FACH}$: amount of time, a shared 3G channel is used
$3G_{DCH}$: amount of time, a dedicated 3G channel is used

Obviously, PowerBooter is abstracting much more from the details of the hardware implementation. Note that PowerBooter also includes communication, which was not taken into account in our previous models. Parameters are determined, as before, by measuring currents in dedicated setups and using some curve fitting method. Measurements are based on a Monsoon power monitor (see http://www.msoon.com/LabEquipment/PowerMonitor/).

The model construction technique allows, in combination with a battery model, a prediction of battery lifetime. The resulting information is made available to a tool called PowerTutor. PowerTutor is intended to provide some help for adjusting applications to different hardware platforms and as an aid for application developers to exploit power-saving techniques in their application without digging deep into the peculiarities of the available hardware.

Another model for the energy consumption in mobile phones was presented by Dusza et al. [144]. Several commercial tools also provide power and/or energy estimation.

All of the energy consumption models considered so far were designed to model an **average case** power or energy consumption, where term "average case" might still need some clarification. Computed models might apply only for certain inputs or for certain initial states. Average case results are valuable for predicting temperatures and battery lifetime for certain time intervals.

5.4.9 Worst Case Energy Consumption

In certain contexts, the **worst case** power consumption or **worst case** energy consumption is of interest.

Definition 5.22 The **worst case energy consumption (WCEC)** of an embedded system is defined as the largest energy consumption, computed as the maximum of the energy consumption for all inputs and initial states.

Definition 5.23 The **worst case power consumption (WCPC)** of an embedded system is defined as the largest power consumption, computed as the maximum of the power consumption for all inputs and initial states.

The WCPC is relevant in the context of the dimensioning of the interconnect and the power supply. The WCEC is relevant in the context of the design of battery systems. We need to guarantee that the chosen battery system meets the WCEC requirements. A safe upper bound on the WCEC can be computed as follows:

$$\text{WCEC} \leq \int_0^{\text{WCET}} \text{WCPC}\, \mathrm{d}t = \text{WCET} * \text{WCPC}$$

Techniques for tighter WCEC estimation have been proposed, for example, by Jayaseelan et al. [271], by Pallister et al. [443], and by Wägemann et al. [559]. Similar to the computation of worst case execution times, these tighter bounds may still be an overestimation, and the actual worst case power and energy consumption are still unknown.

5.5 Thermal Models

The quest for higher performances of embedded systems increased the chances of components becoming hot. Temperatures of the various components of embedded systems can have a serious impact on their usability, e.g., on sensor readouts. In the worst case, overheated components cause damages to other systems. For example, they may cause fire hazards. Hot components might also have other consequences, even in the absence of immediate failures. For example, the system life might be shortened, sometimes by large factors (see Black's equation on p. 283). Also, it may be necessary to power down parts of silicon chips in order to avoid overheating. This has been called the **dark silicon** problem [153].

The thermal behavior of embedded systems is closely linked to the transformation of electrical energy into heat. Therefore, thermal models are usually linked to energy models. Thermal models are based on the laws of physics.[6]

[6] We will denote temperatures by θ in order to avoid confusion with periods denotes by T.

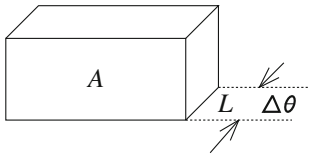

Fig. 5.21 Plate of thickness L

Table 5.1 Approximate thermal characteristics of materials for air, copper, and silicon

Material	κ: thermal conductivity (W/(K m))	c_p: specific heat (J/(K g))	c_v: volumetric heat capacity (J/(K m^3))
Air (25 C)	0.025 [583]	1.012 [578]	1.21 * 10^3 [578]
Copper	401 [583]	0.385 [568, 578]	3.45 * 10^6 [578]
Silicon (≈26 C)	148 [148]	0.705 [148, 568]	1.64 * 10^6 [148][a]

[a]Calculated using Eq. (5.56)

5.5.1 Steady-State Behavior

Consider a homogeneous plate made of a particular material and of area A and thickness L (see Fig. 5.21). Suppose that there is a temperature difference of $\Delta\theta$ between the opposite sides. We assume that heat will be propagating independently of the direction (isotropy), and we assume being in the steady state (no transients). Furthermore, the sides of area are supposed to be much larger than the thickness of the plate, and we can ignore effects at the boundary of the plate. Then, the thermal power which gets transferred across the plate is equal to

$$P_{th} = \kappa \frac{\Delta\theta * A}{L} \quad \text{where:} \tag{5.41}$$

P_{th}: thermal power transferred; κ:thermal conductivity; A: area; $\Delta\theta$: temperature difference; L: thickness

Equation (5.41) is also known as **Fourier's law**.

Definition 5.24 Due to Eq. (5.41), we can define **thermal conductivity** κ as the amount of the thermal power P_{th} transferred through a plate made of some material of unit area and unit thickness when the temperatures at the opposite side differ by one temperature unit (typically Kelvin).

Frequently, λ is used instead of κ. κ depends on the material and environmental conditions. Values for some common materials for common conditions are included in Table 5.1. Refer to the cited sources for more information on the dependency on environmental conditions.

Definition 5.25 **Thermal conductance** [169] is defined as the amount of thermal energy which passes through a plate per unit of time if the temperatures at the two ends differ by one unit of temperature (typically Kelvin).

From Eq. (5.41), we have

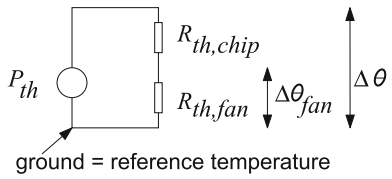

Fig. 5.22 Thermal model of microprocessor with fan

$$\frac{P_{th}}{\Delta\theta} = \kappa * \frac{A}{L} \tag{5.42}$$

The reciprocal of this value is called thermal resistance R_{th}:

$$R_{th} = \frac{\Delta\theta}{P_{th}} = \frac{L}{\kappa * A} \tag{5.43}$$

Lemma 5.1 *Thermal resistances add up like electrical resistances. This allows us to map thermal modeling to electrical modeling.*

Example 5.5 Figure 5.22 shows a microprocessor generating a thermal power P_{th} together with the thermal resistance $R_{th,die}$ of the die (chip) and the thermal resistance $R_{th,fan}$ of the fan.

Adding resistances results in the following equations

$$R_{th} = R_{th,die} + R_{th,fan} \tag{5.44}$$

$$\Delta\theta = R_{th} * P_{th} \tag{5.45}$$

Let us assume the following:

$$R_{th,die} = 0.4\,\text{W/K} \tag{5.46}$$

$$R_{th,fan} = 0.3\,\text{W/K} \tag{5.47}$$

$$P_{th} = 10\,\text{W} \tag{5.48}$$

Then, we compute:

$$\Delta\theta = 7\,\text{K} \tag{5.49}$$

$$\Delta\theta_{fan} = 3\,\text{K} \tag{5.50}$$

Consumed power and thermal resistances are related to the estimation of the thermal design power. ∇

Definition 5.26 ([584]) *"The **thermal design power (TDP)**, sometimes called thermal design point, is the maximum amount of heat generated by a computer chip or component (often the CPU or GPU) that the cooling system in a computer is*

designed to dissipate in typical operation. Rather than specifying CPU's real power dissipation, TDP serves as the nominal value for designing CPU cooling systems."

We could try to derive the TDP from the WCPC. In practice, however, published TDP values are typically smaller. Hence, temperature sensors are required in order to obtain a safe operation.

5.5.2 Transient State Behavior

So far, we have just considered the steady state. In general, transients and thermal capacitance (heat capacity) have to be considered.

Definition 5.27 The **thermal capacitance** (**heat capacity**) of some object is defined as the amount of thermal energy E_{th} which can be stored per difference $\Delta\theta$ in temperatures:

$$C_{th} = \frac{E_{th}}{\Delta\theta} \tag{5.51}$$

Primarily, C_{th} depends on the amount and type of matter contained in the object:

$$C_{th} = c_p * m \tag{5.52}$$

where c_p is the specific heat and m the mass. We can also interpret Eq. (5.52) as the definition of the specific heat:

Definition 5.28 The **specific heat** c_p of some object made of some material of mass m is defined as

$$c_p = \frac{C_{th}}{m} \tag{5.53}$$

c_p depends on the type of matter used. c_p is temperature-dependent, but can be considered constant for small temperature ranges.

In our context, it is frequently more convenient to consider the heat capacity per volume instead of per unit of mass.

Definition 5.29 The **volumetric heat capacity** c_v is defined as

$$c_v = \frac{C_{th}}{\mathcal{V}} \tag{5.54}$$

where $\mathcal{V}$ is the volume of the object.

c_v and c_p are related by the mass density:

Definition 5.30 The **mass density** or **volume density** ρ is defined as

$$\rho = \frac{m}{\mathcal{V}} \tag{5.55}$$

Inserting $\mathcal{V} = m/\rho$ into the definition of c_v, we have

$$c_v = \frac{C_{th}}{\mathcal{V}} = \frac{C_{th} * \rho}{m} = c_p * \rho \tag{5.56}$$

This allows us to convert between tables published for c_p and c_v (see, e.g., Table 5.1). Due to the correspondence to electrical circuits, we can also compute the transient behavior.

Example 5.6 We extend our microprocessor example as shown in Fig. 5.23 (left).

The resulting transient for the temperature across the die and the fan is shown in Fig. 5.23 (right). The system approaches the stable state like a network of resistors and capacitors. ∇

Overall, it is feasible to model thermal behavior by using an equivalent electrical model. Equivalences are shown in Table 5.2.

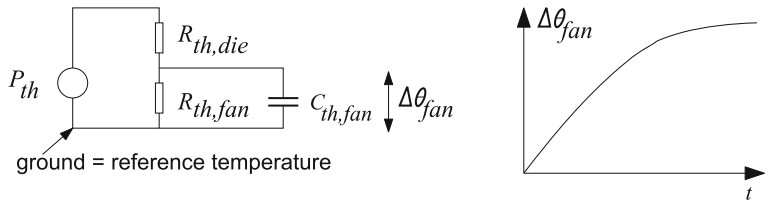

Fig. 5.23 Microprocessor with fan: **left**, thermal model; **right**, transient

Table 5.2 Equivalences between electrical and thermal models

Electrical model		Thermal model	
Current	I	Thermal flow, "power flow"	$P_{th} = \dot{Q}$
Total charge	$Q = \int I \, \mathrm{d}t$	Thermal energy	$E_{th} = \int P_{th} \, \mathrm{d}t$
Potential	ϕ	Temperature	θ
Voltage = potential difference	$V = \Delta\phi$	Temperature difference	$\Delta\theta$
Resistance[a]	$R = \rho_{el} \frac{L}{A}$	Thermal resistance	$R_{th} = \frac{1}{\kappa} \frac{L}{A}$
Ohm's law	$V = R * I$	Δ temperature at R_{th}	$\Delta\theta = R_{th} * P_{th}$
Capacitance	C	Thermal capacitance	C_{th}
Charge on capacitor	$Q = C * V$	Energy at capacitance	$E_{th} = C_{th} * \Delta T$
Capacitance of object[b]	$C = \rho_q \mathcal{V}$	Capacitance of object	$C_{th} = c_v \mathcal{V}$

[a] ρ_{el} is the specific electrical resistance or volume resistivity
[b] ρ_q is the volume charge density

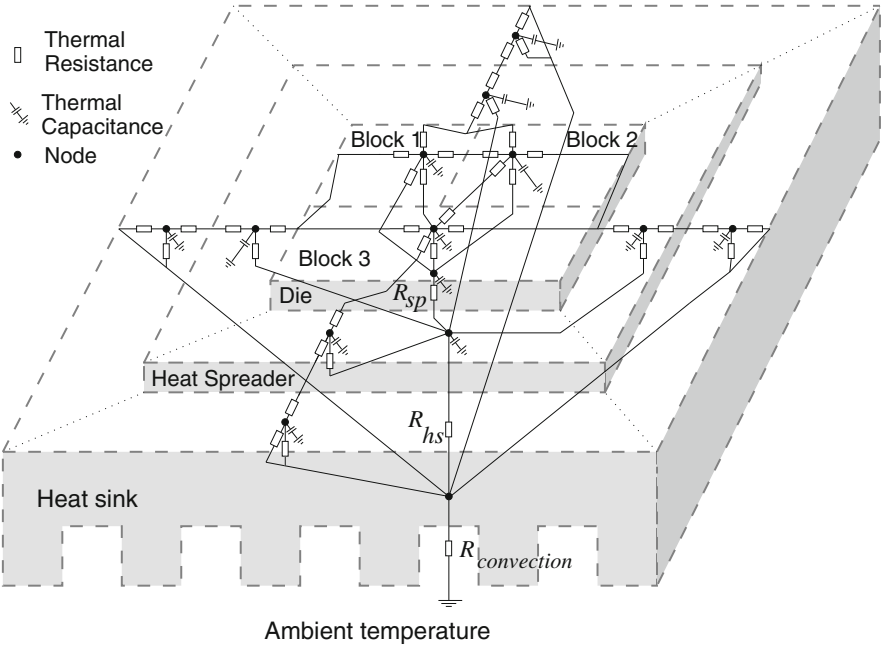

Fig. 5.24 HotSpot model of a chip mounted on a heat spreader and a heat sink

Well-known techniques for solving electrical network equations (see, e.g., Chen et al. [96]) apply. However, there is no component corresponding to inductances on the thermal side. This equivalence between thermal and electrical models is exploited in tools such as HotSpot [500]. Figure 5.24 shows a HotSpot model of a chip mounted on a heat spreader which in turn is mounted on a heat sink [499]. Skadron et al. [499] emphasize the fact that large temperature gradients can exist within a chip, a heat spreader, or a heat sink. Hence, it is important not to assume a uniform temperature for these parts. In Fig. 5.24, the chip is assumed to comprise three micro-architectural components with each component forming one thermal zone.

The heat spreader and the heat sink are modeled as five zones each. One zone of the heat spreader is located beneath the chip, and four zones are located on the sides. Zones on the sides possess a trapezoid-like shape and are indicated by dotted lines. The same partitioning has been done for the heat sink. Zones in the center cannot be shown in Fig. 5.24; they are hidden. Otherwise, each of the zones is shown as a node in the equivalent network in Fig. 5.24. The ambient temperature is assumed to be homogeneous. $R_{convection}$ is the thermal resistance to the environment. It is connected to the five zones of the heat sink. R_{hs} is thermal resistance between the heat spreader and the heat sink. The heat sink is also modeled as five zones. The one in the center is connected to the chip via R_{sp}. The heat source is actually not shown. For each of the zones, there is one thermal capacitance. Each of them models the difference in temperatures if compared to the environment. Accordingly, it is always

considered to be connected to the ground. Furthermore, for each of the zones, there is a pair of thermal resistors connecting adjacent zones.

In their experiments, Skadron at al. have used the Wattch (see p. 262) power simulator as heat source. Microarchitectural simulators such as SimpleScalar can be used to drive Wattch. HotSpot contains mechanisms to create a system of partial differential equations for models such as the one in Fig. 5.24. These equation systems are then solved using a Runge-Kutta equation solver.

Skadron et al. found that it is necessary to consider different thermal zones. Furthermore, they found that power consumption has an impact on the temperature, but in order to really check whether thermal constraints are met, one needs to model temperature explicitly. Several power-saving optimizations had only a small impact on crucial temperatures. For example, register files tend to get hot. Saving power on memory references is of little help in this context and might even have a negative impact.

Example 5.7 As an example of the results of thermal modeling, we consider an MPSoC of STMicroelectronics, comprising 64 P2012 cores [506]. Thermal modeling of this MPSoC has been performed with the 3D-ICE [24] tool. *Relative* temperatures for this MPSoC are shown in Fig. 5.25.[7] High temperatures are shown in red and low temperatures in blue.

The MPSoC contains four clusters, each including 16 cores. Each of the corners of the layout corresponds to one cluster. The 16 processors are located at the center of the clusters. Memories are located below and above the processors. Simulation confirms that the processors are hotter than the memories. The higher utilization of Fig. 5.25 (bottom) leads to higher temperatures. Detailed modeling of the layout avoided temperature overestimation. ∇

Validation of thermal models requires precise temperature measurements [394].

5.6 Dependability and Risk Analysis

Next, we are going to look at dependability and possible risks.

5.6.1 Aspects of Dependability

Embedded and cyber-physical systems (like other products) can cause damages to properties and lives. The fact that such systems are potentially safety-critical was already included in Table 1.2 on p. 18. Hence, in general, we have to take this fact into account. It is not possible to reduce the risk of damages to zero. The best that we

[7] Images are included with permission of David Atienza (EPFL). Images were obtained as part of the cooperation between EPFL and STMicroelectronics in the FP7 EU Project titled: "PRO3D: Programming for Future 3D Architectures with Many Cores".

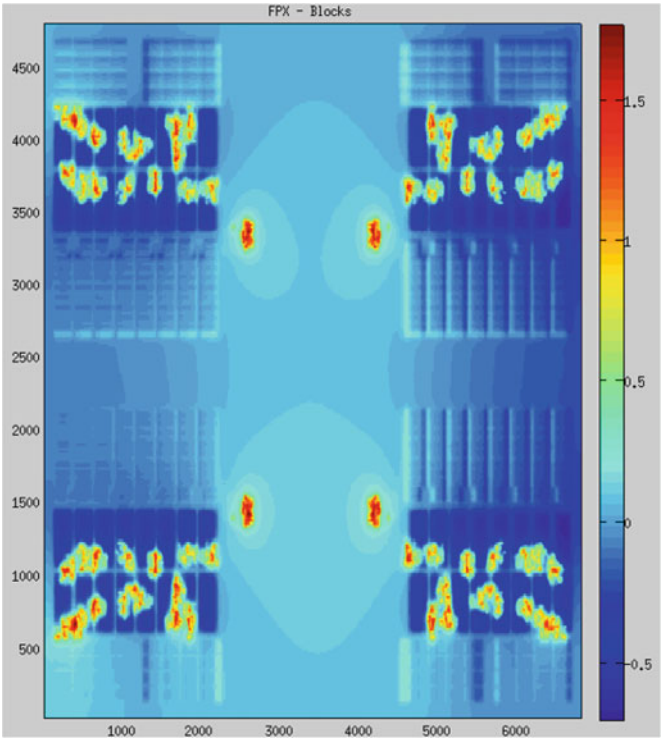

Fig. 5.25 Thermal simulation results for MPSoC: 50% utilization

can do is to make the probability of damages small, hopefully orders of magnitude smaller than other risks. Dependability comprises various aspects, most importantly safety and data security. These, in turn, contain aspects such as reliability and confidentiality. Designs must be evaluated with respect to these aspects.

5.6.2 Security Analysis

Security of embedded and cyber-physical systems was not seen as a serious issue when these systems were not electronically accessible from the outside. This has changed for systems which can be accessed through communication channels, and the two are now much more related, since security holes can cause physical malfunctions resulting in accidents.

Security analysis needs to consider attacker models mentioned already in Sect. 3.8. This analysis needs to find out if attacks are feasible even without having physical access to the embedded system. If the system can be physically accessed, physical attacks must be considered as well.

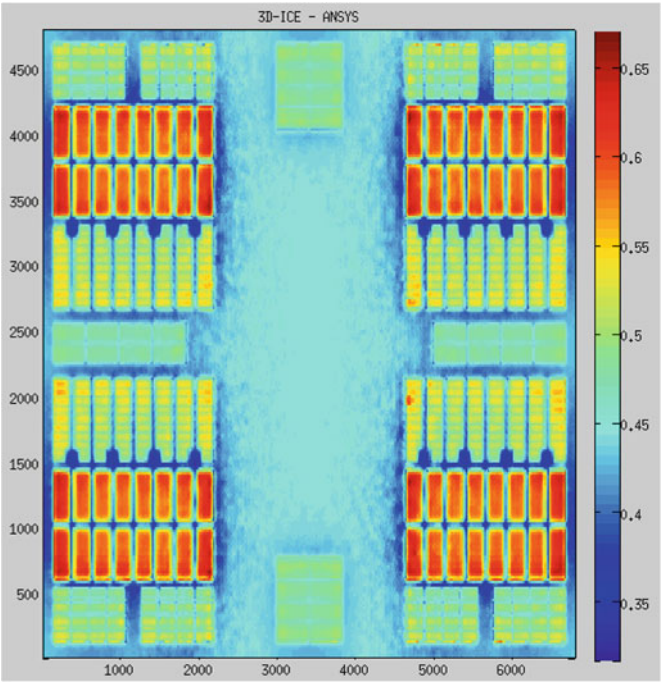

Fig. 5.25 (continued): 100% utilization

Furthermore, relationships between encryption and decryption protocols and achievable data rates must be analyzed, since it could easily happen that resource-constrained embedded devices do not provide the expected encryption and decryption rates.

5.6.3 Safety Analysis

Damages should also be avoided, as much as possible, by designing safe systems. In practice, at best we can expect to design a system such that the probability of damages is orders of magnitude less than the probability of damages from other risks.

Typically, the minimum requirement for manufacturing safety-related products is to be ISO 9001 compliant. This standard defines requirements for quality management systems in general. Requirements as per this standard include the following principles [254]: customer focus, leadership, engagement of people, process approach, improvement, evidence-based decision-making, and relationship management. The first four principles are more or less self-explaining. The improvement principle requires work to proceed in plan, do, check, and act (PDCA) cycles. The goal of planning includes establishing objectives and addressing risks and

opportunities. The goal of the do phase is to implement the plan. This should be followed by checking the results and taking actions to improve if necessary.

For the design of safety-related systems, more specific guidelines have been developed and published as the IEC 61508 international standard [527]. Part 1 [232] of this standard defines standard techniques for technical systems in general. Part 2 [233] specifies *requirements for electrical/electronic/programmable electronic safety-related systems*. Software requirements are listed in part 3 [234]. Parts 4 to 6 contain less formal further recommendations. These standards assume that it is not feasible to design technical systems which always provide the expected service. Emphasis is placed on documented design procedures capable of tracing underlying reasons for incorrect decisions.

In standard IEC 61508, a distinction is made between four different levels of risks, called safety integrity levels (SIL). For continuously operating devices, the standard specifies failure rates per hour of 10^{-5} to 10^{-6} for SIL-1, 10^{-6} to 10^{-7} for SIL-2, 10^{-7} to 10^{-8} for SIL-3, and 10^{-8} to 10^{-9} for SIL-4 [581]. SIL-4 is difficult to achieve and typically requires redundant execution. Problems arise from the current trend toward **mixed-criticality**, which means that subsystems of different SIL-levels are implemented, for example, on the same multi-core processor. Proper shielding of the different levels of criticality is difficult.

Standard IEC 61508 is expected to apply to several industries. There are specific extensions for specific industries. These consider, for example, the amount of time which is available for human interventions, the possibility of transitioning into a fail-safe mode, and the impact of malfunctions. For example, there is very little time to react if something goes wrong in a car. However, cars can usually be stopped and parked in a "fail-safe" mode and a safe place (with the exception of some tunnels, etc.). In contrast, there is usually some more time available in an airplane, but some safety-critical systems in an airplane cannot simply be turned off.

MISRA-C defines rules to be followed when using the C programming language for safety-critical systems [396].

ISO 26262 [252] is a standard more tailored for the automotive industry.

Standards IEC 62279 and CENELEC 50128 take the special situation for rail-based transportation into account [60].

For avionics, systems should comply with the Airworthiness Certification Specifications FAR-CS 25.1309 "Equipment, Systems and Installations" and with AC-AMC 25.1309 "System design and analysis" [549]. This is extended for hardware by standard DO-254 and for software by standard DO-178B ("Software Considerations in Airborne Systems and Equipment Certification") [163, 474], in Europe also called ED-12B. DO-178C is a follow-up standard for DO-178B.

IEC 61511 [236] has been defined for applications in manufacturing, and IEC 61513 [235] is a special standard for nuclear power plants.

Allowed failures may be in the order of 1 failure per 10^9 hours of operation or even significantly less for highly safety-critical systems like nuclear power plants. This may be several orders of magnitude less than the failure rates of chips. Hence, Kopetz [303] stressed that the system as a whole must be more dependable than any of its parts and that safety requirements cannot come in as an afterthought but must be considered right from the beginning. Obviously, fault-

tolerance mechanisms must be used. Due to the low acceptable failure rate, systems are not 100% testable. Instead, safety must be shown by a combination of testing and reasoning. Abstraction must be used to make the system explainable using a hierarchical set of behavioral models. Design faults and human faults must be taken into account.

In order to address these challenges, Kopetz proposed the following 12 design principles:

1. Safety considerations may have to be used as **the** important part of the specification, driving the entire design process.
2. Precise specifications of design hypotheses must be made right at the beginning. These include expected failures and their probability.
3. Fault-containment regions (FCRs) must be considered. Faults in one FCR should not affect other FCRs.
4. A consistent notion of time and state must be established. Otherwise, it will be impossible to differentiate between original and follow-up errors.
5. Well-defined interfaces must hide the internals of components.
6. It must be ensured that components fail independently.
7. Components should consider themselves to be correct unless two or more other components pretend the contrary to be true (principle of self-confidence).
8. Fault-tolerance mechanisms must be designed such that they do not create any additional difficulty in explaining the behavior of the system. Fault-tolerance mechanisms should be decoupled from the regular function.
9. The system must be designed for diagnosis. For example, it has to be possible to identify existing (but masked) errors.
10. The man-machine interface must be intuitive and forgiving. Safety should be maintained despite mistakes made by humans.
11. Every anomaly should be recorded. These anomalies may be unobservable at the regular interface level. This recording should involve internal effects, since otherwise they may be masked by fault-tolerance mechanisms.
12. Provide a never-give-up strategy. Embedded systems may have to provide uninterrupted service. The generation of pop-up windows or going off line is unacceptable.

Definition 5.31 As system is **resilient** if internal or external changes of the assumptions made at design time will change the overall user experience only in a limited way.

A system which is **self-repairing** would provide some level of resiliency. Resiliency is beyond the scope of this book.

5.6.4 Reliability Analysis

The design of dependable systems also requires an analysis of the reliability (the likelihood of initially correctly designed systems not to malfunction due to

some internal fault). This task is expected to become more important and more difficult in the future, since **decreasing feature sizes of semiconductors will be resulting in a reduced reliability of semiconductor devices** (see, e.g., http://variability.org). Transient as well as permanent faults are expected to become more frequent. Shrinking feature sizes will also cause an increased variability among device parameters. Therefore, dependability analysis and fault-tolerant designs are becoming extremely important [179, 406]. Faults within semiconductors might lead to failures of the system. The terms **faults**, **failures**, and the related terms **error** and **service** were defined by Laprie et al. [29, 323].

Definition 5.32 "*The **service** delivered by a system (in its role as a **provider**) is its behavior as it is perceived by its user(s); ... The delivered service is a sequence of the provider's external states. ... **Correct service** is delivered when the service implements the system function.*"

Definition 5.33 "*A **service failure**, often abbreviated here to **failure**, is an event that occurs when the delivered service of a system deviates from the correct service. ... A service failure is a transition from correct service to incorrect service.*"

Definition 5.34 An **error** exists if one of the system's states is incorrect and may lead to its subsequent service failure.

Definition 5.35 "*The adjudged or hypothesized cause of an error is called a **fault**. Faults can be internal or external of a system.*"

Some faults will not cause a system failure.

As an example, we might consider a transient *fault* flipping a bit in memory. After this bit flip, the memory cell will be in *error*. A *failure* will occur if the system service is affected by this error.

In line with these definitions, we will talk about *failure* rates when we consider systems that do not provide the expected system function. We will talk about *faults* whenever we consider the underlying **reasons** that might cause failures. There are a large number of possible reasons for faults, some of them resulting from reduced feature sizes of semiconductors. *Errors* will not be considered in the remaining part of this book.

Reaching a level of dependability corresponding to SIL-4 is only feasible if design evaluation also comprises the analysis of the reliability, the expected lifetime, and related objectives. Such an analysis is usually based on the probability of failures.

More precisely, we consider the probability densities of failures. Let x be the time until the first failure. x is a random variable. Let $f(x)$ be the probability density of this random variable.

As an example, we are frequently using the exponential probability density $f(x) = \lambda e^{-\lambda x}$. For this density function, failures are becoming less and less likely over time (after some time, it is likely that the system is not working anymore and a system which is not working cannot fail). This density function is frequently used since it has a constant failure rate and, hence, describes in an appropriate way cases

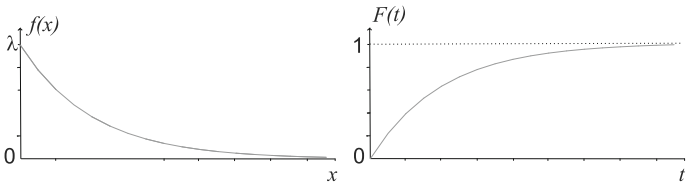

Fig. 5.26 Exponential distribution: **left**, density function; **right**, probability distribution

for which the failure rate is constant. We might even use this density function when the actual failure rate is unknown since a constant failure rate may be a good starting point. Moreover, this density function has nice mathematical properties. Figure 5.26 (left) shows this density function.

The probability distribution is frequently more interesting than the density. This distribution represents the probability of a system not working at time t. It can be obtained by integrating the density function until time t.

$$F(t) = Pr(x \leq t) \tag{5.57}$$

$$F(t) = \int_0^t f(x)\mathrm{d}x \tag{5.58}$$

For example, for the exponential distribution, we obtain:

$$F(t) = \int_0^t \lambda \mathrm{e}^{-\lambda x}\mathrm{d}x = -[\mathrm{e}^{-\lambda x}]_0^t = 1 - \mathrm{e}^{-\lambda t} \tag{5.59}$$

Figure 5.26 (right) contains the corresponding function. As time advances, this probability approaches 1. This means that, as time progresses, it becomes more likely that the system will have failed.

Definition 5.36 The **reliability** $R(t)$ of a system is the probability of the time until the first failure being larger than t:

$$R(t) = Pr(x > t), t \geq 0 \tag{5.60}$$

$$R(t) = \int_t^\infty f(x)\mathrm{d}x \tag{5.61}$$

$$F(t) + R(t) = \int_0^t f(x)\mathrm{d}x + \int_t^\infty f(x)\mathrm{d}x = 1 \tag{5.62}$$

$$R(t) = 1 - F(t) \tag{5.63}$$

$$f(x) = -\frac{\mathrm{d}R(t)}{\mathrm{d}t} \tag{5.64}$$

For the exponential distribution, we have $R(t) = \mathrm{e}^{-\lambda t}$ (see Fig. 5.27).

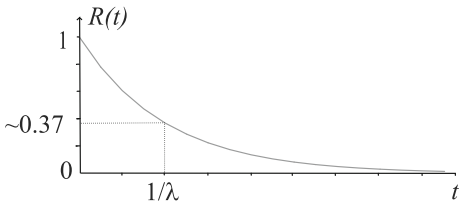

Fig. 5.27 Reliability for exponential distribution

The probability for the system to be functional after time $t = 1/\lambda$ is about 37%.

Definition 5.37 The **failure rate** $\lambda(t)$ is the probability of a system failing between time t and time $t + \Delta t$.

$$\lambda(t) = \lim_{\Delta t \to 0} \frac{Pr(t < x \leq t + \Delta t | x > t)}{\Delta t} \tag{5.65}$$

$Pr(t < x \leq t + \Delta t | x > t)$ is the conditional probability for the system failing within this time interval provided that it was working at time t. For conditional probabilities, there is the general equation $Pr(A|B) = Pr(AB)/Pr(B)$, where $Pr(AB)$ is the probability of A *and* B happening. $Pr(AB)$ is equal to $F(t + \Delta t) - F(t)$ in our case. $Pr(B)$ is the probability of the system working at time t, which is $R(t)$ in our notation. Therefore, Eq. (5.65) leads to:

$$\lambda(t) = \lim_{\Delta t \to 0} \frac{F(t + \Delta t) - F(t)}{\Delta t R(t)} = \frac{f(t)}{R(t)} \tag{5.66}$$

For example, for the exponential distribution, we obtain:[8]

$$\lambda(t) = \frac{f(t)}{R(t)} = \frac{\lambda e^{-\lambda t}}{e^{-\lambda t}} = \lambda \tag{5.67}$$

Failure rates are frequently measured as multiples (or fractions) of 1 FIT, where "FIT" stands for *Failure unIT* and is also known as *Failures In Time*. 1 FIT corresponds to 1 failure per 10^9 hours.

However, failure rates of real systems are frequently not constant. For many systems, we have a "bath tub curve"-like behavior (see Fig. 5.28).

For this behavior, we are starting with an initially larger failure rate. This higher rate is a result of an imperfect production process or "infant mortality." The rate during the normal operating life is then essentially constant. At the end of the useful product life, the rate is then increasing again, due to wear-out.

[8]This result motivates denoting the failure rate and the constant of the exponential distribution with the same symbol.

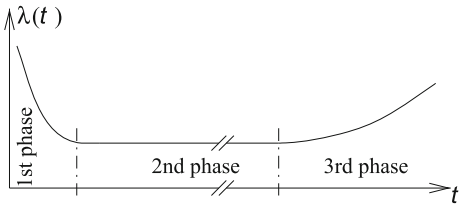

Fig. 5.28 Bath tub curve-like failure rates

Definition 5.38 The **mean time to failure** (**MTTF**) is the average time until the next failure, provided that the system was initially working. This average can be computed as the expected value of random variable x:

$$\text{MTTF} = E\{x\} = \int_0^\infty x f(x) \mathrm{d}x \tag{5.68}$$

For example, for the exponential distribution, we obtain:

$$\text{MTTF} = \int_0^\infty x \lambda \mathrm{e}^{-\lambda x} \mathrm{d}x \tag{5.69}$$

This integral can be computed using the product rule ($\int uv' = uv - \int u'v$ where in our case we have $u = x$ and $v' = \lambda \mathrm{e}^{-\lambda x}$). Therefore, Eq. (5.69) leads to the following equation:

$$\text{MTTF} = -[x\mathrm{e}^{-\lambda x}]_0^\infty + \int_0^\infty \mathrm{e}^{-\lambda x} \mathrm{d}x \tag{5.70}$$

$$= -\frac{1}{\lambda}[\mathrm{e}^{-\lambda x}]_0^\infty = -\frac{1}{\lambda}[0 - 1] = \frac{1}{\lambda} \tag{5.71}$$

This means that, for the exponential distribution, the expected time until the next failure is the reciprocal value of the failure rate.

There is the following empirical relationship between MTTF and operating temperatures:

Lemma 5.2 (Black's equation [49, 55])

$$\text{MTTF} = \frac{A}{j_e^n} \mathrm{e}^{\frac{E_a}{k\vartheta}} \tag{5.72}$$

where

A : constant

j_e : current density

n : constant (1..7), controversial, 2 according to Black

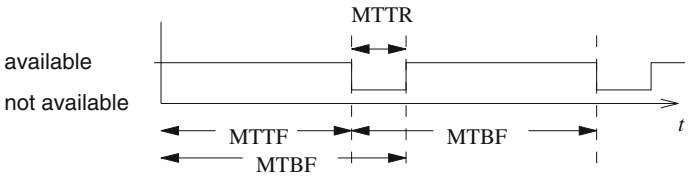

Fig. 5.29 Illustration of MTTF, MTTR, and MTBF

E_a : activation energy (e.g., $\approx 0.6\,\text{eV}$)

k : Boltzmann constant ($\approx 8.617 * 10^{-}5\,\text{eV/K}$)

θ : temperature

Regardless of discussions about the correct value of n, this equation shows that the temperature has an exponential impact on the MTTF. Furthermore, current densities are also important: the larger the current densities, the shorter the lifetime of the product.

Definition 5.39 The **mean time to repair** (**MTTR**) is the average time to repair a system, provided that the system is initially not working. This time is the expected value of the random variable denoting the time to repair.

Definition 5.40 The **mean time between failures** (**MTBF**) is the average time between two failures.

MTBF is the sum of MTTF and MTTR:

$$\text{MTBF} = \text{MTTF} + \text{MTTR} \tag{5.73}$$

Figure 5.29 shows a simplistic view of this equation: it is not reflecting the fact that we are dealing with probabilistic events, and actual MTBF, MTTF, and MTTR values may vary randomly. For many systems, repairs are not considered. Also, if they are considered, the MTTR should be much smaller than the MTTF. Therefore, the terms MTBF and MTTF are frequently mixed up. For example, the lifetime of a hard disk may be quoted as a certain MTBF, even though it will never be repaired. Quoting this number as the MTTF would be more correct. Still, the MTTF provides only very rough information about dependability, especially if there are large variations in the failure rates over time.

Definition 5.41 The **availability** is the probability of a system being in an operational state.

The availability varies over time (just consider the bath tub curve!). Therefore, we can model availability by a time-dependent function $A(t)$. However, we are frequently only considering the availability A for large time intervals. Hence, we define

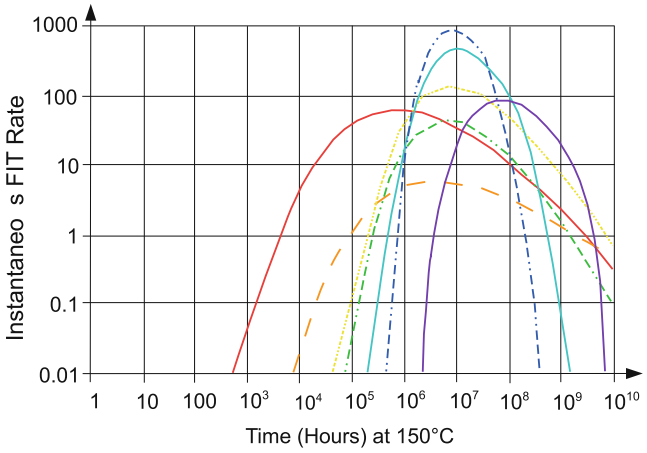

Fig. 5.30 Failure rates of TriQuint's gallium arsenide devices (courtesy of TriQuint, Inc., Hillsboro), ©TriQuint

$$A = \lim_{t \to \infty} A(t) = \frac{\text{MTTF}}{\text{MTBF}} \tag{5.74}$$

For example, assume that we have a system which is repeatedly available for 999 days and then needs 1 day for repair. Such a system would have an availability of $A = 0.999$.

Allowed failure rates can be in the order of 1 FIT. This may be several orders of magnitude less than the failure rates of chips. This means that systems must be more reliable than their components! Obviously, the required level of reliability makes fault-tolerance techniques a must!

Obtaining actual failure rates is difficult. Figure 5.30 shows one of the few published results [546]. This figure contains failure rates for different gallium arsenide (GaAs) devices with the hottest transistor operating at a temperature of 150 °C.

This example is used here to demonstrate that there exist devices for which the assumptions of constant failure rates or a bath tub-like behavior are oversimplifying.[9] As a result, citing a single MTTF number may be misleading. The actual distribution of failures over time should be used instead. In the particular case of this example, failure rates are less than 100 FIT for the first 20 years (175,300 h) of product lifetime, despite the high temperature. FIT numbers are actually very much temperature dependent, and temperatures up to 275 °C and known temperature dependences have been used at TriQuint to compute failure rates for periods larger than the time available for testing. TriQuint claims that their GaAs devices are more

[9] Therefore, the so-called log-normal distribution is sometimes considered.

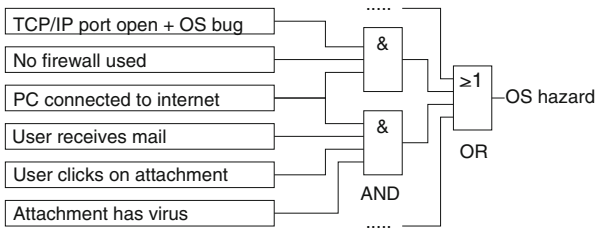

Fig. 5.31 Fault tree

reliable than average silicon devices. Reports on FIT testing are also available for Xilinx FPGAs (see, e.g., [600]).

5.6.5 Fault Tree Analysis, Failure Mode, and Effect Analysis

It is frequently not possible to experimentally verify failure rates of complete systems. Requested failure rates are too small, and failures may be unacceptable. We cannot fly 10^5 airplanes 10^4 hours each in an attempt to check if we reach a failure rate of less than 10^{-9} (SIL-4)! The only way out of this dilemma is to use a combination of checking failure rates of components and formally deriving from this guarantees for a reliable operation of the system. Design- and user-generated failures also must be taken into account. It is state of the art to use decision diagrams to compute the reliability of a system from that of its components [260].

Damages are resulting from **hazards** (chances for a failure). For each possible damage caused by a failure, there is a severity (the cost) and a probability. Risk can be defined as the product of the two. Information concerning the damages resulting from component failures can be derived with at least two techniques [143, 459]:

- **Fault tree analysis (FTA):** FTA is a top-down method of analyzing risks. The analysis starts with a possible damage and then tries to come up with possible scenarios that lead to that damage. FTA is based on modeling a Boolean function reflecting the operational state of the system (operational or not operational). FTA typically includes symbols for AND- and OR-gates, representing conditions for possible damages. OR-gates are used if a single event could result in a hazard. AND-gates are used when several events or conditions are required for that hazard to exist. Figure 5.31 shows an example.[10] FTA is based on a **structural** model of the system, i.e., it reflects the partitioning of the system into components.

[10]Consistent with the ANSI/IEEE standard 91, we use the symbols &, =1 and ≥1 to denote AND-, XOR-, and OR-gates, respectively.

Table 5.3 FMEA table

Component	Failure	*Consequences*	Probability	Critical?
...	...	...	...	...
Processor	Metal migration	No service	10^{-7}/h	Yes
...	...	...	...	...

The simple AND- and OR-gates cannot model all situations. For example, their modeling power is exceeded if shared resources of some limited amount (like energy or storage locations) exist. Markov models [67] may have to be used to cover such cases. Markov models are based on the notion of **states**, rather than on the structure of the system.

- **Failure mode and effect analysis (FMEA):** FMEA starts at the components and tries to estimate their reliability. Using this information, the reliability of the system is computed from the reliability of its parts (corresponding to a bottom-up analysis). The first step is to create a table containing components, possible failures, probability of failures, and consequences on the system behavior. Risks for the system as a whole are then computed from the table. Table 5.3 shows an example.

Tools supporting both approaches are available. Both approaches may be used in "safety cases". In such cases, an independent authority has to be convinced that certain technical equipment is indeed safe. One of the commonly requested properties of technical systems is that no single failing component should potentially cause a catastrophe.

The design of safe and dependable systems is a topic on its own. This book can only provide a few hints into this direction. There is an abundant amount of recent publications on the impact of reliability issues on system design. Examples include publications by Huang [223], Zhuo [613], and Pan [445]. For more information about dependability, consult books [181, 323, 339, 418, 513] on those areas.

5.7 Simulation

In this chapter, we have so far placed an emphasis on design evaluation. Starting with this section, we are now also considering **validation**. Simulation is a very common technique for evaluating and validating designs. Simulation consists of executing a design model on appropriate computing hardware, typically on general-purpose digital computers. Obviously, this requires models to be executable. All the executable models and languages introduced in Chap. 2 can be used in simulations, and they can be used at various levels as described starting at p. 115. The level at which designs are simulated is always a compromise between simulation speed and accuracy. The faster the simulation, the less accuracy is available.

So far, we have used the term behavior in the sense of the functional behavior of systems (their input/output behavior). There are also simulations of some non-functional behaviors of designs, including the thermal behavior and the electromagnetic compatibility (EMC) with other electronic equipment. Due to the integration with physics, there is a large range of physical effects which may have to be included in the simulation model. As a result, it is impossible to cover all relevant approaches for simulating cyber-physical systems in this book. Law [325] provides an overview of approaches and topics in simulations on digital systems. A large amount of additional information on the simulation of systems (in particular of heterogeneous, cyber-physical systems) is available (see, e.g., [126, 362, 442]). Some simulators specialize on specific application areas. Due to the large number of physical effects, it is impossible to provide a complete list of references.

For cyber-physical systems, simulations have serious limitations:

- Simulations are typically a lot slower than the actual design. Hence, if we interface the simulator with the actual environment, we can have quite a number of **violations of timing constraints**.
- Simulations in the physical environment may even be **dangerous** (who would want to drive a car with unstable control software?).
- For many applications, there may be huge amounts of data, and it may be impossible to simulate enough data in the available time. Multimedia applications are notoriously known for this. For example, simulating the compression of some video stream takes an enormous amount of time.
- Most actual systems are too complex to allow simulating all possible cases (inputs). Hence, simulations can help us to find errors in our designs. They cannot guarantee absence of errors, since simulations cannot exhaustively be done for all possible combinations of inputs and internal states.

Due to these limitations, there is an increased emphasis on validation by formal verification (see p. 290). Nevertheless, sophisticated simulation techniques continue to play a key role for validation (see, e.g., Braun et al. [66]). Academic solutions like gem5 (see http://gem5.org), SimpleScalar, and OpenModelica as well as commercial solutions like the Synopsys® Virtualizer™ (see http://synopsys.com) are available. There are several tools for the simulation of networks (as required for the Internet of Things), including OMNET++ (see https://omnetpp.org/).

5.8 Rapid Prototyping and Emulation

Simulations are based on models, which are approximations of real systems. In general, there will be some difference between the real system and the model. We can reduce the gap by implementing some parts of our system under design (SUD) more precisely than in a simulator (e.g., in a real, physical component).

Definition 5.42 Adopting a definition phrased by M^{c}Gregor [383], we define **emulation** as the process of executing a model of the SUD where at least one component is **not** represented by simulation on some kind of host computer.

According to M^{c}Gregor, "*Bridging the credibility gap is not the only reason for a growing interest in emulation — the above definition of an emulation model remains valid when turned around — an emulation model is one where part of the real system is replaced by a model. Using emulation models to test control systems under realistic conditions, by replacing the ... (real system) ... with a model, is proving to be of considerable interest to those responsible for commissioning, or the installation and start-up of automated systems of many kinds.*"

In order to further improve credibility, we can continue replacing simulated components by real components. These components do not have to be the final components. They can be approximations of the real system itself but should exceed the precision of simulations.

Note that it is now common to discuss the "emulation" of one computer on another computer by means of software. There is a lack of a precise definition of the use of the term in this context. However, it can be considered consistent with our definition, since the emulated computer is not just simulated. Rather, a speed faster than simulation speed is expected.

Definition 5.43 Fast prototyping is the process of executing a model of the SUD where **no** component is represented by simulation on some kind of host computer. Rather, all components are represented by realistic components. Some of these components should not yet be the finally used components (otherwise, this would be the real system).

There are many cases in which the designs should be tried out in realistic environments before final versions are manufactured. Control systems in cars are an excellent example for this. Such systems should be used by drivers in different environments before mass production is started. Accordingly, the automotive industry designs prototypes. These prototypes should essentially behave like the final systems, but they may be larger, have more power consuming, and have other properties which test drivers can accept. The term "prototype" can be associated with the entire system, comprising electrical and mechanical components. However, the distinction between rapid prototyping and emulation is also blurring. Rapid prototyping is by itself a wide area which cannot be comprehensively covered in this book.

Prototypes and emulators can be built, for example, using FPGAs. Racks containing FPGAs can be stored in the trunk while test drivers exercise the car. This approach is not limited to the automotive industry. There are several other fields in which prototypes are built from FPGAs. Commercially available **emulators** consist of a large number of FPGAs. They come with the required mapping tools which map specifications to these emulators. Using these emulators, experiments with systems which behave "almost" like the final systems can be run. However, catching errors

by prototyping and emulation is already a problem for non-distributed systems. For distributed systems, the situation is even more difficult (see, e.g., Tsai [547]).

5.9 Formal Verification

Formal verification[11] is concerned with formally proving a system correct, using the language of mathematics. First of all, a formal model is required to make formal verification applicable. This step can hardly be automated and may require some effort. Once the model is available, we can try to prove certain properties.

Formal verification techniques can be classified by the type of logic employed:

- **Propositional logic:** In this case, models consist of Boolean expressions. Tools are called **Boolean checkers**, **tautology checkers**, or **equivalence checkers**. They can be used to verify that two representations of Boolean functions (or sets of Boolean functions) are equivalent. Since propositional logic is decidable, it is also decidable whether or not the two representations are equivalent (there will be no cases of doubt). For example, one representation might correspond to gates of an actual circuit and the other to its specification. Proving the equivalence then proves the effect of all design transformations (e.g., optimizations for power or delay) to be correct. Boolean checkers can cope with designs which are too large to allow simulation-based exhaustive validation. The key reason for the power of Boolean checkers is the use of binary decision diagrams (BDDs) [571]. The complexity of equivalence checks of Boolean functions represented with BDDs is linear in the number of BDD nodes. The number of BDD nodes can potentially grow exponentially with the number of variables, but, in practice, many relevant functions can be represented with compact BDDs.[12] In contrast, the equivalence check for functions represented by sums of products is NP-hard. BDD-based equivalence checkers have therefore replaced simulators for this application and handle circuits with millions of transistors.
- **First-order logic (FOL):** FOL adds $\exists$ and $\forall$ quantifiers to propositional logic. Some automation for verifying FOL models is feasible. However, since FOL is undecidable, there may be cases of doubt. Popular techniques include the **Hoare calculus**. Typically, operations on integers are also supported.
- **Higher-order logic (HOL):** Higher-order logic is based on lambda calculus and allows functions to be manipulated like other objects [423]. For higher-order logic, proofs can hardly ever be automated and typically must be done manually with some proof support.

[11] This initial text on formal verification was based on a guest lecture given by Tiziana Margaria at TU Dortmund.

[12] Multiplication is a prominent exception [284].

Propositional logic can be used to verify stateless logic networks but cannot directly model finite state machines. For short input sequences, it may be sufficient to cut the feedback loop in FSMs and to effectively deal with several copies of these FSMs, each copy representing the effect of one input pattern. However, this method does not work for longer input sequences. Such sequences can be handled with **model checking**.

For model checking, we have two inputs to the verification tool:

1. The model to be verified
2. Properties to be verified

States can be quantified with ∃ and ∀; numbers cannot. Verification tools can prove or disprove the properties. In the latter case, they can provide a counterexample. Model checking is easier to automate than FOL. It has been implemented for the first time in 1987, using BDDs. It was possible to locate several errors in the specification of the *future bus* protocol [104]. UPPAAL is a very popular tool for model checking.[13]

This technique could be used, for example, to prove properties of the railway model of Fig. 2.52 (see p. 82). It should be possible to convert the Petri net into a state chart and then confirm that the number of trains commuting between Cologne and Paris is indeed constant, confirming our discussion of Petri net place invariants on p. 81.

5.10 Problems

We suggest solving the following problems either at home or during a flipped classroom session:

5.1 Let us consider an example demonstrating the concept of Pareto optimality. In this example, we study the results generated by task concurrency management (TCM) tools designed at the IMEC research center (*Interuniversitair Micro-Electronica Centrum*). TCM tools aim at establishing efficient mappings from applications to processors. Different multiprocessor systems are evaluated and represented as sets of Pareto optimal designs. Wong et al. [595] describe different options for the design of an MPEG-4-player. The authors assume that a combination of StrongARM processors and specialized accelerators should be used. Four designs meet the timing constraint of 30 ms (see Table 5.4). These different designs are shown in Fig. 5.32. For combinations 1 and 4, the authors report that only one mapping of tasks to processors meets the timing constraints. For combinations 2 and 3, different time budgets lead to different task to processor mappings and different energy consumptions.

[13] See http://www.uppaal.org for the academic and http://www.uppaal.com for the commercial version.

Table 5.4 Processor configurations

Processor combination	1	2	3	4
Number of high-speed processors	6	5	4	3
Number of low-speed processors	0	3	5	7
Total number of processors	6	8	9	10

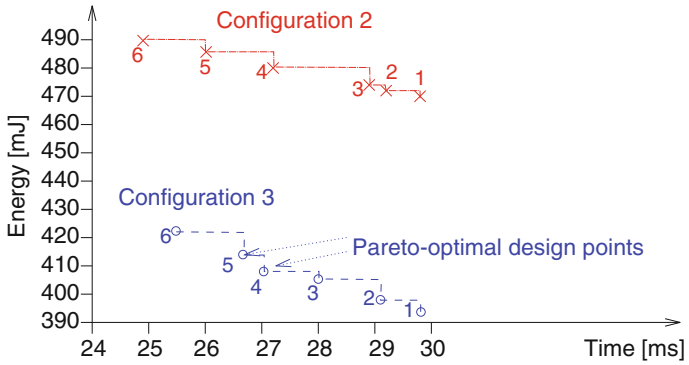

Fig. 5.32 Pareto points for multiprocessor systems 2 and 3

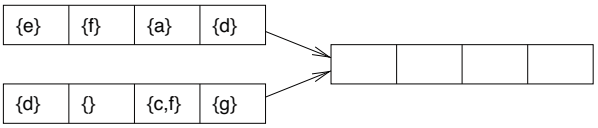

Fig. 5.33 Abstract cache states

Which area in the objective space is dominated by at least one design of configuration 3? Is there any design belonging to configuration 2 which is not dominated by at least one design of configuration 3? Which area in the objective space dominates at least one design of configuration 3?

5.2 Which conditions must be met by computations of WCET_{EST}?

5.3 Let us consider cache states at a control flow join. Figure 5.33 shows abstract cache states before the join.

Now let us look at abstract cache states after the join. Which state would a *must*-analysis derive? Which state would a *may*-analysis derive?

5.4 Consider an incoming "bursty" event stream. The stream is periodic with a period of T. At the beginning of each period, two events arrive with a separation of d time units. Develop *arrival curves* for this stream! Resulting graphs should display times from 0 up to $3*T$.

5.5 Suppose that you are working with a processor having a maximum performance of b.

1. What do the *service curves* look like if the performance can deteriorate to b', due to cache conflicts?
2. How do the *service curves* change if some timer is interrupting the executed program every 100 ms and if servicing the interrupt takes 10 ms? Assume that there are no cache conflicts.
3. How do the *service curves* look like if you consider cache conflicts like in (1.) **and** interrupts like in (2.)?

   Resulting graphs should display times from 0 up to 300 ms.

5.6 Suppose that we try to collect amber. However, there is the risk of also collecting white phosphorus. Suppose that we collect 50 objects. We keep all of them in water to avoid fire hazards. We classify 30 objects as amber and 20 as white phosphorus. However, two of the objects classified as amber are actually pieces of white phosphorus and 8 objects classified as white phosphorus are actually consisting of amber. Compute the precision, recall, accuracy, and specificity for this classification!

5.7 Suppose that you try to compute the power consumption of your mobile phone using a shunt resistor. The following values are relevant for the computation of the power consumption at some time t: resistor, 0.47 Ω; power supply voltage, 5.1 V; and voltage across shunt, 0.23 V. What is the power consumption of your mobile at this time t?

5.8 Consider a copper plate of area A=10 cm^2 and length 5 mm. How much thermal power is transferred if the difference between the temperatures at the two ends of the plate is 10 °C?

5.9 Consider a hard disk drive for which we assume that half of the drives have failed after 5000 h of operation. Let us assume that failures follow an exponential distribution. Compute the corresponding value of λ!