A Design Assistant for Scheduling of Design Decisions

Reinhard Rauscher
University of Hamburg
Dept. of Computer Science
Hamburg, Germany
e-mail: rauscher@informatik.uni-hamburg.de

Abstract

At the core of contemporary VLSI designs is the task of determing and evaluating the set of decisions relevant for the particular design project. In many designs the set of design decisions is handled intuitively, and also the set of alternatives for each of the design decisions is strongly influenced by experiences from earlier designs. The paper discusses the list of design decisions under consideration, their alternatives, and the ordering of such a set of design decisions. A set of practical designs have been analyzed using the system introduced in the paper.

1. Introduction

In the past, the design of highly integrated (VLSI-) systems has been considered a very time consuming process. The design procedure has to be carried out by specialized designers, who had detailed knowledge of the underlying technology.

In the meantime this has changed. Nowadays - because of widespread use of technology independant models and corresponding tools - the design can be done quite fast. Typical examples of these models are *standard cell design* [MeCo 80] and the *Meet-in-the-middle strategy* [Rab 88]. The latter allows realization of a design even if the designer has little knowledge of the technical details.

On the other hand more and more design decisions arise, which are made intuitively or based on experiences from earlier designs. Leung and Shanblatt define ([LeSh 89]):

> " ... viewing design as a decision making process. That is, system designers must structure the design process so that the solution space is manageable and design alternatives are consciously sought and evaluated."

Figure 1 illustrates the unordered list of design questions to be ordered by the designer. After ordering of the ques-

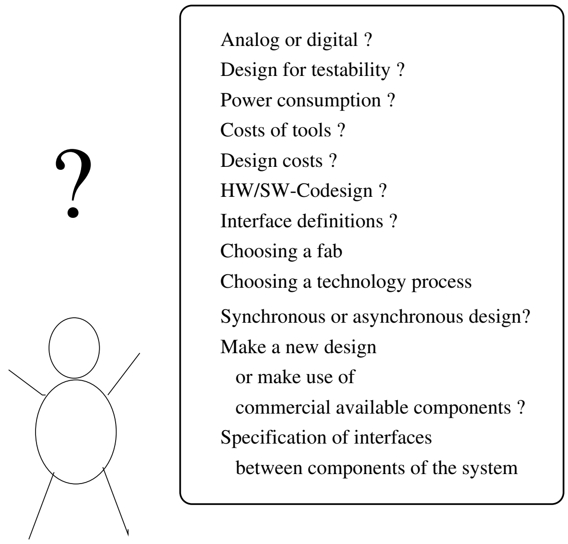


Figure 1. Confusion of the designer

tions he has to choose a solution from a set of alternatives for each of the design decisions. The task of the designer can be roughly sketched by the following quotation ([Rab 88]):

> "Enormous dimensions of the design space have to be explored."

Any a priori sequence of design decisions (valid for all design activities) would be desirable, but it cannot exist. Knapp und Parker ([KnPa 91]):

> "...and also the ordering of the decisions can be permutated as well. This makes an a priori ordering of design decisions of doubtful utility."

In the literature different orders of design decisions have been proposed ([VLS 87, Gil 85]). In addition, the opinions about the relevance of some of the design decisions differ. The same holds for the amount of design decisions, which differ from a few (10) up to 1000 ([KnPa 91]):

"In a design of practical size, there are ten, hundreds or thousands of such decisions. Deciding which decision to make next and predicting the outcome of each individual decision is a combinatoric problem."

Summarized, the number of decisions, the alternatives for each of the decisions, and the sequence of the decisions depends on the application problem. In other words, the designer has the task to determine decisions relevant for his particular design, the alternatives for each of the decisions, and finally the scheduling of the decisions.

Considering Fig. 1, some restrictions - for the actual design task - can be defined.

Some of the decions have to be handled in parallel, others have to be ordered in an a priori defined sequence. An example for decisions one has to deal with in parallel could be *synchronous or asynchronous design ?* and *fixing the maximal power consumption*.

Another example coming from Fig. 1 could be the parallelism of the decisions *analog or digital design?* and *Specification of interfaces between components of the system*. Both questions have to be answered in parallel, if the designer has the opportunity to choose between well suited analogue or digital interfaces.

An example for sequential decision making is the following: *Make a completely new design or make use of commercial available components ?*. This is a decision, which must be handled before the decision *Choosing a fab and/or choosing the production process* can be done. Obviously, the second decision is relevant for the designers task only if the first decision is made "make a new design".

In the following a concept and a corresponding tool is presented which allows a structured procedure in ordering the design decisions. The basic idea is, that a designer can use his own knowledge concerning the set of design decisions and his experiences concerning their interdependencies, but also he might use a standard sequence, which must be extendable.

Such a tool has to meet the following demands:

- The list of design decisions to be considered must be incrementally enlargable by the user (designer).
- The list of mutual dependencies between the design decisions (parallel or sequential) must be incrementally enlargable by the user (designer).

One goal can be defined as follows: Find a sequence of the unordered list of design decisions, which fulfills all specified mutual dependencies between design decisions.

If not all of the specified dependencies can be fulfilled, the violations of the dependencies must be *weighted*. Based on this, a second goal can be described as follows:

Find a sequence of the unordered list of design decisions, which minimizes the weighted sum of violated dependencies.

2. Demands to be fulfilled

Because of the fact, that there can not exist an a priori sequence of the decisions, the user must be able to choose his own set of design decisions and additionally his own *weighted* set of dependencies. These dependencies consist of two classes, first those which describe sequential dependencies between two specified decisons, second, those which describe parallel decisions. The parallel decisions can be interpreted in the following two ways:

- The user wishes to handle both decisions really in parallel. So, they can be considered as one compound decision.
- The decisions are influencing each other. So, they are not necessarily to be executed in parallel, but in the sequence of the decisions they should be close to each other.

The demands for such a tool are:

- Option for an iterative modification. This includes extension, modification and deletion of decisions and dependencies.
- Consistency check. This is necessary because of the interactive capabilities.
- Illustration of intuitively made decisions. This helps designers in widening their solution search space.
- Documentation of the design methodology. The design process is not straight-forward, but progresses more in a Jo-Jo-manner [Ram 85]. Arising from this, the online-documentation of design steps should be possible, including also the explanation for choosing the desired sequence of decisions.
- Determination of a schedule with a set of fixed decisions. This means, that the user can specify a fixed order for a given subset of the decisions. In this case only the remaining decisions have to be included in an initial schedule.
- Determination of milestones. Milestones are those decisions, which allow the interpretation of the handled decisions as a preliminary result. Another description of milestones would be, that there exist none or only weak dependencies between the set of decisions already handled and those to be dealt with next.
- Consideration of practical constraints. E.g. at most k decisions could be handled in parallel. Such a demand could be necessary, because of the fact, that a designer is only able to deal with a limited set of decisions in parallel.
- Partitioning of the problem into subproblems. Especially when more than one person is involved, there is the need for partitioning.

3. Theoretical background

In this section a mathematical model is explained, which is used in the concept and the tool. For illustration, an example is introduced which will be referenced throughout the rest of the paper.

Example 1:
Given is the set of design decisions: $\{$E1, E2, E3, E4, E5$\}$. It should be mentioned that this set of decisions has been extracted from a practical example, in which the following correspandance holds:

- E1 corresponds to ”Design for testability”.
- E2 corresponds to ”Synchronous or asynchronous design?”.
- E3 corresponds to ”Power consumption”.
- E4 corresponds to ”Costs of the tool chain”.
- E5 corresponds to ”Choose a fab, choose the production process”.

The following dependencies between the design decisions should be fulfilled:

E1 before E2 weighted 1.
E3 before E5 weighted 10.
E2 before E5 weighted 4.
E2 parallel to E4 weighted 3.
E2 parallel to E3 weighted 6.
E4 parallel to E5 weighted 2.

Def. 1: Let G=(V,E) be a directed, weighted graph. Each vertex $v \in V$ corresponds to one of the design decisions.
G is called a *sequence graph (S)*, iff for each edge $e = (x, y) \in E \subset VxV$ weighted with $g(e) \in \mathbb{N}$ the following condition holds:
In the list of dependencies there exists an item
”x before y weighted $g(e)$”.

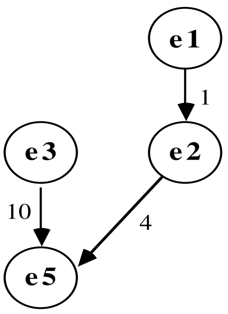


Figure 2. The sequence graph of example 1

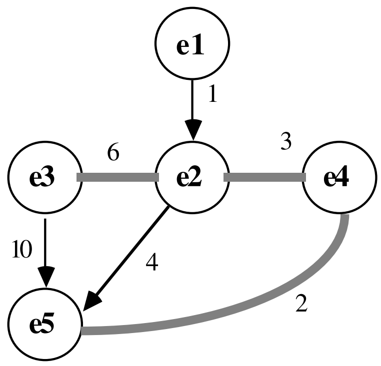
Def. 2: Let $G = (V, E)$ be a directed, weighted graph. Each vertex $v \in V$ corresponds to one of the design decisions. G is called a *parallel graph (P)*, iff for each edge $e = (x, y) \in E \subset VxV$ weighted with $g(e) \in \mathbb{N}$ the following condition holds:
In the list of dependencies there exists an item ”x parallel to y weighted $g(e)$ ”.

Figure 3. The parallel graph of example 1

Def. 3: Let $(S) = (V, E')$ be a sequence graph and $(P) = (V, E'')$ a parallel graph corresponding to a set V of design decisions. The *dependency graph (T) = (V,ES, EP)* is defined as follows: $\forall (x, y) \in VxV$:

$$(x, y) \in ES \Leftrightarrow (x, y) \in E'$$
$$(x, y) \in EP \Leftrightarrow (x, y) \in E''.$$

Additionally, for each edge $e \in (ES \cup EP)$ the weight $g(e)$ is the same as in the original graph. To distinguish between weights in ES and such in EP we use $g_{ES}(e)$ and $g_{EP}(e)$.



Legend: Edges concerning parallelism are marked by grey colored lines, edges concerning sequential execution are marked by black lines.

Figure 4. The dependency graph for example 1

Based on the demands for incremental extension it is obvious, that inconsistencies within the dependency graph may occur. As an example consider the demands for parallelism and sequence as described in Fig. 3. To avoid such inconsistencies the transitive closure can be used.

Def. 4: Let $G = (V, E)$ be a directed weighted graph. The *transitive hull H=(V,E')* is defined as follows:

$$\forall (x, y) \in E \quad \Rightarrow \quad (x, y) \in E' \qquad (1)$$
$$\forall (x, y) \in E' \wedge (y, z) \in E' \quad \Rightarrow \quad (x, z) \in E'. \qquad (2)$$

Remark: In the case of unweighted graphs the definition of the transitive closure is unique (for parallel and sequence graphs). In the case of weighted graphs the definition has to be improved. Let be $(x, y) \in E'$ and $(y, z) \in E'$. The new weight for e.g. $(x, z) \in E'$ could be defined in two alternative ways:

- The new weight of (x,z) is given by the minimum weight of (x,y) and (y,z).
- The new weight of (x,z) is given by the minimum weight of (x,y) and (y,z).

More general:
Let $w_1, ..., w_j$ be all paths of length 2 from x to z,
The new weight $(x, z) \in E'$ is defined by the *maximal* or *minimal* value of the weights corresponding to $w_1, ..., w_j$.

Both alternatives make sense for parallel graphs and for sequence graphs.

Def. 5: Let $T = (V, ES, EP)$ be a dependency graph. The following functions can be defined:
output,input,omi: $V \to \mathbb{Z}$

$$output(v) := \sum_{(v,y)\in ES} g_{ES}((v, y)) \quad (3)$$

$$input(v) := \sum_{(x,v)\in ES} g_{ES}((x, v)) \quad (4)$$

$$omi(v) := output(v) - input(v) \quad (5)$$

Informally, omi (abbreviation for output minus input) denotes the difference between input and output. The definition of omi can be widened for a set of vertices T. *setomi:* $P(V) \to \mathbb{Z}$ is defined as follows:

$$setomi(T) := \sum_{v\in T} omi(v), T \subset V \quad (6)$$

At last *setomiint:* $P(V)xP(V) \to \mathbb{Z}$ is defined as follows: Let be $V1, V2 \subset V$, $V1 \cap V2 = \phi$.

$$temp1 := \sum_{(v,y)\in ES \wedge v\in V2, y\in V1} g(v, y)$$

$$temp2 := \sum_{(v,y)\in ES \wedge v\in V1, y\in V2} g(v, y)$$

$$setomiint(V1, V2) := temp1 - temp2 \quad (7)$$

Informally, $setomiint$ describes the sum of weighted edges between two sets $V1$ and $V2$.

Corresponding to the values in example 1 the following results can be achieved:

$$output(E2) = 4$$
$$input(E2) = 1$$
$$omi(E2) = 3$$
$$setomi(E2, E3) = 14 - 1 = 13$$
$$setomiint : (\{E2, E3, E4\}, \{E5\}) = 0 - 14 = -14$$

Considering especially the parallel graph (P) there is the problem, that the transitive closure induces many parallel dependencies between the design decisions. Additionally, a subset of these decisions could be sequentialized. A reason for this is e.g. that this subset has to be executed in serial to other decisions within the same set of parallel decisions.

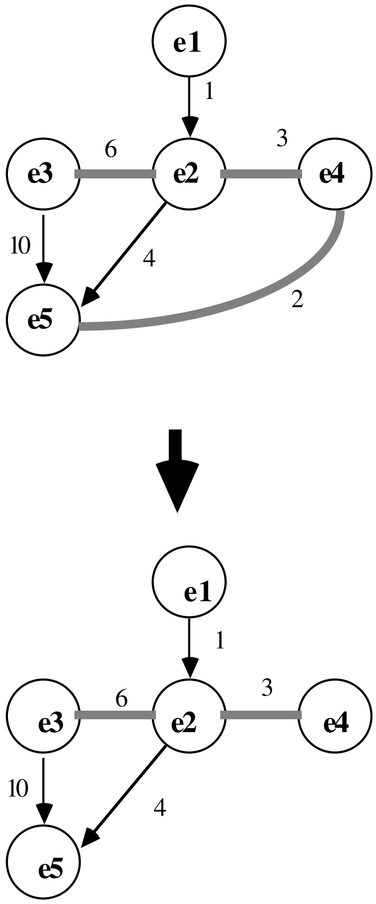


Figure 5. Example for a sequentialization of a set of design decisions specified as parallel

In Fig. 3 such a sequentialization is illustrated. The figure shows the sequentialization of {E2, E3, E4, E5} into {E2, E3, E4} and {E5}.

The basic idea for such a sequentialization is the following: If there exists a partitioning $P1, P2$ of the set $\{E1, ..., Ek\}$ for which the condition $|setomi(P1) - setomi(P2)| > |setomiint(P1, P2)|$ holds, a sequentialization could be either performed automatically or interactively.

Def. 6: Definition of the objective function. Let $(T) = (V, ES, EP)$ be a dependency graph.
The *goal* is a partial ordering ($\gg$) of the set V with

- $Ei \gg Ej$ and $Ej \gg Ei \Leftrightarrow (Ei, Ej) \in EP$.

- $\sum_{(x,y) \in ES \; and \; (x \not\gg y)} g(x,y)$ has the minimum value.

This graph can be generated by using a transitive closure procedure and/or by using a sequentialization procedure as described above.

Def. 7: Let $(T) = (V, ES, EP)$ be a dependency graph and let $\gg$ be a partial ordering of the set V. A vertex $v \in V$ is called *milestone*, if the following conditions hold:
1. $\nexists v_1, v_2 \in V$: $v_1 \gg v \gg v_2$ *and* $(v_2, v_1) \in ES$ or $(v_1, v_2) \in EP$.
2. $\nexists v_2 \in V$: $v \gg v_2$ *and* $(v_2, v) \in ES$ or $(v, v_2) \in EP$.

Def. 8: Let a dependency graph $S = (V, ES, EP)$ be given. An *inconsistency* exists given, if there exists a path $w = (v_1, v_2, ..., v_k)$ with:

- $v_1 = v_k$
- $(v_i, v_{i+1}) \in$ ES $\cup$ EP $\forall i \in \{1, ..., k-1\}$
- $\exists i \in \{1, ..., k-1\} : (v_i, v_{i+1}) \in$ ES

For evaluation of the distance of two different schedules, two metrics have been defined. The use of these metrics will make it possible to analyze, how similiar two schedules of design decisions are.

3.1. Metric 1

Let $A_1 = (p_1, p_2, .., p_k)$ und $A_2 = (p'_1, p'_2, ..., p'_k)$ be two given schedules of the same set of design decisions (PD).

$ord(e, A_i) = i \Leftrightarrow p_i = e$ denotes the ordinal number of the decision e in the schedule A_i.

Now, the distance between two schedules can be defined as follows:

$$D1(A_1, A_2) := \sum_{i=1}^{k} (|ord(e_i, A_1) - ord(e_i, A_2)|)$$

. Considering metric 1 the question about the *maximum distance* (**mdist**) between two schedules and the question about the *average distance* (**adist**) occur The number of permutations can be determined trivially to be: $k!$

mdist is given by: $mdist := 2 \cdot \lfloor \frac{k}{2} \rfloor \cdot \lceil \frac{k}{2} \rceil$

As an example, for $k = 30$, $mdist = 450$ results.

adist has been determined as: $adist := \frac{mdist}{2}$.

As an example, for $k = 30$, $adist = 225$ results.

3.2. Metric 2

Let be given two schedules A_1 and A_2 with $k_1 = |A_1| \geq |A_2| = k_2$. Additionally, the following condition: $\forall E \in A_1 \; \exists E' \in A_2 : \; E \subset E'$ holds.

For any comparison of the schedules, the number of items must be identical. Coming from this, the schedules A_1 and A_2 could be enhanced. The new schedules are named A'_1 and A'_2. Obviously, there exists an enhancement, so that both conditions hold:

1. $|A'_1| = |A'_2|$, $|A'_1|$ is minimal
2. $\forall \; E \in A'_1 \; \exists E' \in A'_2 : E = E'$.

The second metric is defined as follows:

$$D2(A_1, A_2) := \sum_{i:=1}^{k_1} (|ord(e_i, A'_1) - ord(e_i, A'_2)|).$$

4. Implementation

Based on the mathematical model described above, an implementation has been realized.

Input to the system is an ASCII file which describes the set of design decisions and a list of their mutual dependencies.

Following the two different interpretations of parallelism two different heuristics have been implemented.

4.1. Heuristic 1

Due to the first interpretation of parallelism the user (designer) decides interactively, whether the procedures 'transitive closure' or 'sequentialization' should be executed. After this, no further parallelisation or sequentialisation can be done. In the following the term *PD* is used as an abbreviation for a set of parallel design decisions.

For the determination of a suitable schedule of the different PD the following heuristic has been implemented:

1. Set $M := V$
2. repeat
3. Determination of $setomi(A) \subset M$ for all PD A
4. B:= that $PD \subset M$, for which the following condition holds: $setomi(B)$ has the maximal value;
5. B is the set of design decisions to be considered next. B is printed to the output file.
6. $M := M \setminus B$
7. until $M = \phi$
8. STOP

This heuristic can be improved by the following partitioning of the set of design decisions:

1. None of the vertices in PD has a predecessing vertex not in PD:
   $\nexists x \in PD : (\exists y \in (V \setminus PD) : (y, x) \in ES)$

2. There exist vertices in PD, which have a predecessor not in PD and there exist vertices in PD which have a successing vertex not in PD:
   $\exists x \in PD : \exists y \in (V \setminus PD) : (x, y) \in ES$ and
   $\exists x \in PD : \exists y \in (V \setminus PD) : (y, x) \in ES$
3. None of the vertices in PD has a successing vertex not in PD:
   $\nexists x \in PD : \exists y \in (V \setminus PD) : (x, y) \in ES$

In the beginning only such PDs are considered, which belong to the first class. In the second step only such PDs are allocated, which belong to the second class. At last the third class is considered. The generated output is the ordered list of the sets named PD.

4.2. Heuristic 2

This heuristic is based on to the second interpretation of parallelism. The conceptional difference to heuristic 1 is, that all decisions are considered to be executed in serial. The underlying algorithm can be described as follows:

1. Set $M := V$
2. repeat
3. determination of $omi(a) \in M$ for all decisions a
4. b:= that decision $\in M$, with *omi(b)* has maximal value;
   $aspvalue := omi(b)$;
   Baisc idea: All par-dependencies are ignored in the first step.
5. In case that the last decision y contains par - dependencies, determination of

$$temp1 \quad := \quad g_{EP}(x, y) - g_{ES}(b, x)$$

$$aspnewvalue \quad := \quad omi(x) + temp1 \qquad (8)$$

   for all nodes x connected to y.
6. In the case that the last decision y of the current schedule has no par-dependencies, determination of:

$$temp1 \quad := \quad max(0, \sum_{z \in V : (x,y) \in EP} omi(z))$$
$$temp2 \quad := \quad min(g_{EP}(x, y), temp1)$$

$$aspnewvalue \quad := \quad omi(x) - g_{EP}(x, y) + temp2$$

   The basic idea is to avoid, that a decision corresponding to other decisions using par-dependencies is scheduled at a too early position.
7. That decision b, for which the maximal value $aspnewvalue, aspvalue$ occurs, is the next decision within the current schedule.
8. $M := M \setminus \{b\}$
9. until $M = \phi$
10. Try a parallelisation of the decisions.
11. STOP

5. System overview

In Figure 5 the system overview is illustrated. The inputs are an unordered set of design decisions, a set of dependencies between these decisions, and optionally an initial sequence of the decisions.

In a first step a consistency check is performed. After this the new (near optimal) schedules are determined using two different heuristics.

On the right side of Figure 5 there is a system component, which determines the optimal solution. Because of the fact, that the optimization problem can be mapped onto the problem of the travelling salesman, this is only possible for small or well suited examples. The (academic) goal of this component is to perform an evaluation of the heuristics implemented.

The "comparison" tool determines the distance $D2$ between the solution and an optionally described second schedule.

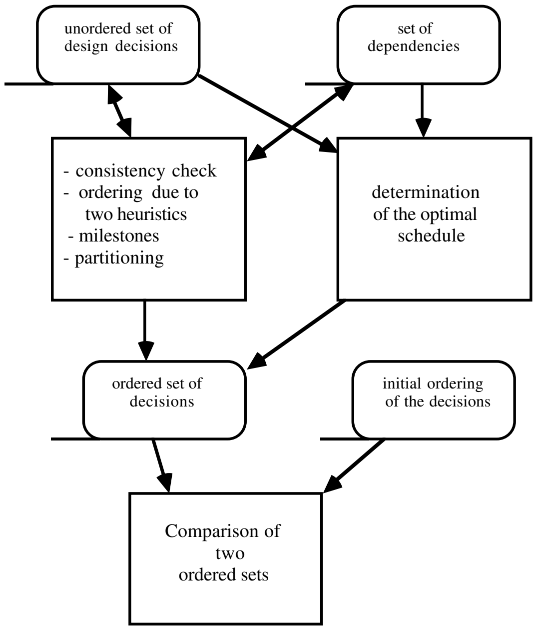


Figure 6. System overview

6. Results for practical ASIC-designs

Based on experiences from several ASIC-designs the system's prototype has been analyzed. An example for such a design is given in Figure 6. Please note, that the graphical illustration is a first result. From Figure 6 it can be seen, that there exist inconsistencies, and that a partioning is possible.

Starting with a set of 30 general design decisions, first of all the dependencies have been determined. This has been done using different steps. In the first step, some of the design decisions have been classified as 'irrelevant for the actual problem'. In the second step, parallel dependencies are determined and weighted. Finally, sequential dependencies are evaluated.

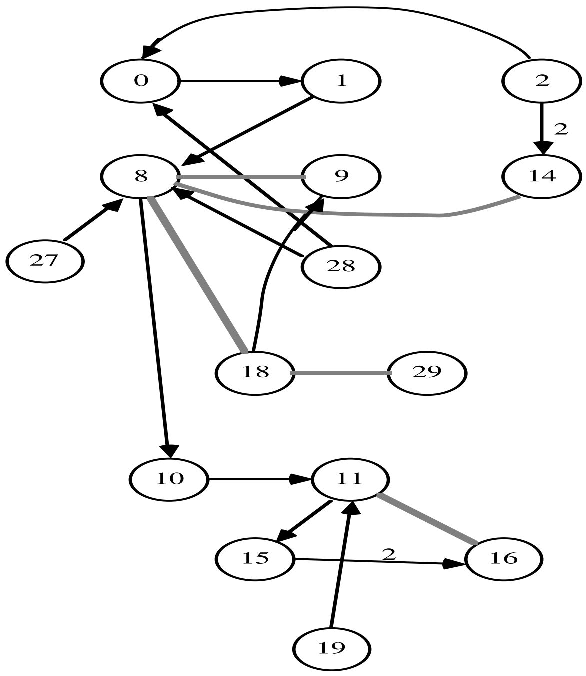


Figure 7. Dependency graph of a practical example (KISS-Chip)

In the following the results for both heuristics implemented are discussed.

6.1. Results with heuristic 1

The heuristic 1 is due to the first interpretation of parallelism.

6.1.1. Generalized results

The results were:

1. For several designs only a subset of the 30 design decisions has been proved to be necessary. Partially, they have been determined by the specification itself or nobody noticed, that a decision had been done intuitively. Furthermore, some of the decisions have not been relevant for the actual design.

2. Partially, the results of the system have been distinct from that sequence of design decisions, which has been choosen while doing the design. The reason for this could be, that most of these designs have been done in a real top-down manner without detailed analysis of dependencies.

3. A comparison between the schedule of design decisions generated using one of the procedures 'transitive closure' or 'sequentialize' with that one without these procedures did result only in quite important changes in the schedule. But, most of the milestones are identical for all these alternatives.

Especially the last item should be discussed in some more detail.

6.2. Results concerning practical designs

For evaluation of the system we analyzed some practical examples designed at our institute. The examples are named:

- small (the small academic example mentioned above)
- general (a set of dependencies generated from a set of designs)
- four real designs named SCEL, SCHEME, DELM, and PWR.

Obviously, recent parameters for heuristic 1 are $sequentialisation$ and $transitiveclosure$. So, the following convention is used: nn is the abbreviation for the case, that no use of tranistive closure and no use of the sequentialization procedure has been choosen. Corresponding to this ny denotes, that now the sequentialization has been done. The cases yn and yy are defined analogue.

The results for each of the examples have been: For each alternative (with or without transitive closure, with or without sequentialization) the following values have been determined:

1. The schedule of the ordered decisions partially to be executed in parallel
2. The number of milestones for yy
3. The number of decisions to be executed sequentially for yy
4. The distance $D2$ corresponding to the schedules for nn and yy.

More detailed results are given in [Rau 95].

Dependent on the choice of $transitive\ closure$ and $sequentialization$ the resulting schedules are as follows:
Schedule for nn: $E1, E2 \mid\mid E3, E4 \mid\mid E5$.
Schedule for ny: $E3, E1, E4 \mid\mid E2, E5$.
Schedule for yn: $E1, E4 \mid\mid E5 \mid\mid E2 \mid\mid E3$.
Schedule for yy: $E3, E1, E4 \mid\mid E2, E5$.

Figure 8 shows in the first column the name of the design. The second and third column describe the number of serial decisions and the amount of milestones in case yy. The fourth column describes the difference due to distance $D2$ between the schedules of nn and yy.

name	#dec.	#ms.	dist.
small	4	2	4
general	22	2	168
SCEL	18	3	186
SCHE	21	2	112
DELM	22	2	128
PWR	21	2	192

Figure 8. Results concerning heuristic 1

6.3. Results concerning heuristic 2

For a set of practical examples the following results concerning heuristic 2 have been determined:

1. Are there inconsistencies ?
2. Amount of relevant decisions
3. Amount of milestones
4. The penalty value. This value describes the sum of the weighted edges, which are violated.
5. Is the solution the best one ?
6. The distance concerning the metric $D2$ between the solution and an initial ordering.

The results are shown in Figure 6.3. The first column describes the name of the design, the second, whether there have been inconsistencies or not. The third and the fourth column depict the amount of relevant decisions, and the amount of milestones, respectively. The column named 'pen.' contains the penalty value for all violated edges. the column named 'opt.' describes, whether there could be guaranteed, that the solution is optimal. In case, that this could not guaranteed, the number of different schedules, which have to be analyzed is depicted. The last column describes the distance due to metric $D2$ between the initial ordering and the solution determined.

name	inc.	#dec.	#ms.	pen.	opt.	dist.
small	yes	5	2	2	yes	-
general	yes	30	16	15	30!	208
SCEL	no	17	9	8	18!	150
SCHE	no	22	12	10	18!	150
DELM	yes	19	10	12	12!	162
PWR	no	17	9	10	16!	126
KISS	yes	15	8	4	9!	78
HIST	yes	26	14	7	18!	272

Figure 9. Results concerning heuristic 2

7. Conclusions

The analysis of results achieved with the system shows, that there has been a huge difference between the schedule of decisions performed within our projects and the best suited one. Additionally, the results show, that the meta-decisions $use\ of\ transitive\ closure$ and $sequentialization$ are of crucial relevance for the schedule of design decisions.

So, using the design assistant could result in a better schedule and reduce the efforts within the design procedure.

References

[ErPo 91] Erenyi, I.; Pongracz, J.: Quality Control in Textile Industry via Machine Version, Proc. EUROMICRO 91, Wien, Sept. 1991, pp 807-813

[Gaj 88] Gajski, D.: Silicon Compilation, Addison-Wesley, 1988

[Gil 85] Giloi, W.K.; Shriver, B.D.: An Overview of the Computer System Design Process; in: Giloi, W.K. (ed.), Methodologies for Computer System Design, North-Holland, Amsterdam, 1985

[LeSh 89] Leung, S.; Shanblatt, M.: ASIC System Design with VHDL: A Paradigm, Kluwer Acad. Press., Boston, 1989

[KnPa 91] Knapp, D.; Parker, A.: The ADAM Design Planning Engine, IEEE Transact. on CAD, Vol. 10, No. 7, July 1991, pp 829-846

[MeCo 80] Mead, C.; Conway, L.: Introduction to VLSI Systems, Addison-Wesley, Reading, Mass., 1980

[Rab 88] Rabaey, J.; De Man, H. et al.: Cathedral-II: A Synthesis System for Multiprocessor DSP Systems, In: Silicon Compilation, D.D. Gajski (ed), Addison-Wesley, 1988, pp 311-360

[Ram 85] Rammig, F.: The Multilevel Cybernetic Model of the Design Process; in Giloi, W.K. (ed.), Methodologies for Computer System Design, North-Holland, Amsterdam, 1985

[Rau 95] Rauscher, R.: Untersuchungen zur (An-) Ordnung von Entwurfsentscheidungen, Proceedings Workshop Mikrosystemtechnik, Chemnitz, 16-17. Okt. 1995, pp206-215

[VLS 87] VLSI DESIGN: Silicon Compilers: Part 2; VLSI DESIGN, Okt. 1987, pp 65-68