Productivity gains through re–use and quality improvements of hw models(*)

A.Allara†, A. Balboni†, M. Bombana†, M. Mastretti†, P. Plaza‡, J. R. Prieto‡, Joachim Schaaf*

† Italtel SpA, Palazzo Laboratori CLTE, 20019 Settimo Milanese, Milano, Italy
‡ Telefonica I+D, Emilio Vargas, 28043 Madrid, Spain
* Deutsche Telekom AG, TZ Darmstadt, FZ112b, Postfach 100003, 64276 Darmstadt, Germany

Industrial design flows for HW and mixed (HW/SW) applications increasingly exploit re–use of HW models and libraries of complex, quality proven components to increase productivity and decrease time to market. In this paper we analyze the concept of productivity and underline the existing links with various design parameters, focusing on re–use and quality of models. Some theoretical models are introduced to quantify HW model re–use. These are used to provide guidelines to support decision making. Then we analyze heuristic techniques through which designers currently apply HW model re–use in both VHDL and Verilog based design environments. Finally we show how REQUEST Project complies with the industrial requirements and generalizes the experience of designers proposing solutions for quality improvements and design re–use. Some conclusions will provide hints on future activities of the project.

1. Introduction

Today, microelectronics is a basic technology for developing complex telecom systems. This market segment is characterized by a very strong competition, not only among 'domestic' subjects (i.e. European), but also Japanese and American ones. As a consequence, it is crucial for the European microelectronic market to minimize the costs and increase reliability of products. Project REQUEST intends to tackle this aspect, focusing on:

- enhancing reusability of HDL models
- enhancing design quality of HDL models for several design properties [1]

due to their evident impact on the productivity process of hardware devices.

In general, the identification of the dependency of design productivity from reliable design parameters is today perceived as a strategic issue in industry in order to control and to improve the quality of the design process.

Nowadays the attention is specially focused on ASIC design because ASICs are considered the bottle–neck of the implementation process for custom or semicustom ICs. This is a consequence of the technological evolution which moved the logic complexity of new telecom equipments from the board level to the ASIC level. Today a typical PC board for ATM, SDH or GSM systems contains microprocessors and a few complex ASICs (100keg – 300keg).

The evolution in ASIC design follows a trend of a doubling of gate count every 2 years (figure 1). Since design teams dimension didn't increase in the same way it is reasonable to state a similar trend for the design efficiency, expressed as:

D_{eff} = K equivalent gates / man month (1)

Anyway the increase of efficiency was not reflected in a decrease of the total design time. The general perception is that the global design process for sub–micron and deep sub–micron devices became more critical. In particular the predictability of the time needed for specific design steps (synthesis, layout, simulation) seemed to be extremely difficult, causing dramatic errors in the design planning.

A more objective measure of efficiency can be represented by the ratio:

Global Design Efficiency = GD_{eff} = Design Complexity / Global Design Time (2)

GDT is easy to measure, as it is the time from the project starting date to the test of working prototypes. Anyway GDT is affected by the quality of specification. When stable specifications are available,

(*) This research is partially supported by ESPRIT IV Project REQUEST

surprisingly short design times are observed. DC is a function of several parameters.

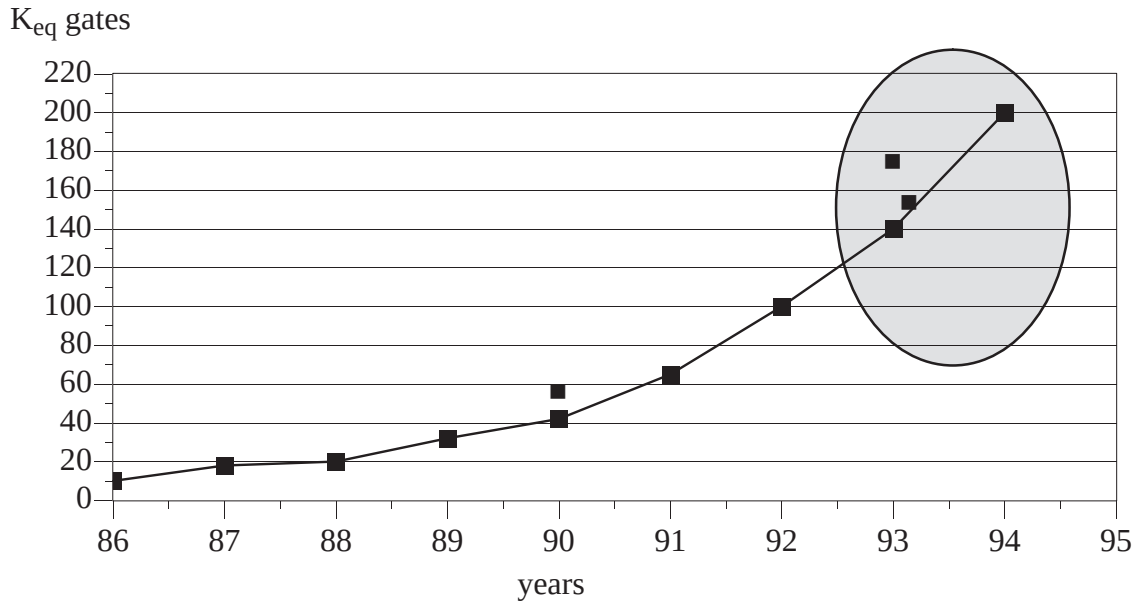

Figure 1. Design Complexity (Source: ITALTEL)

Referring to digital CMOS ASICs we can try to define the complexity as function of gate count and critical design constraints (frequency, die size, power consumption, distance from the intrinsic technology limit)

$$DC = k_0 * \text{Gate count} + k_1 * \text{Frequency} + k_2 * \text{Power} + k_3 * \text{Die Usage} + k_4 * \text{Technology} \quad (3)$$

Gate count only partially contributes as a linear part of DC. In fact gate counts of big regular blocks (e.g. RAMs) usually do not really reflect overall complexity. A block with complex logic with the same gate count gives a much higher contribution to design complexity. A better solution would be to include in this term the gate count for the synthesized logic and the memory bit count, differentiating single ports and dual port RAMs, as the latter are usually more difficult to handle, i.e. more complex. These correction factors can be taken into consideration in the value of k_0. The frequency term is basically valid for the more diffused ASIC technologies, sea of gates or embedded. The technology term represents the level of maturity of the used technology with respect to the project time frame. The linear dependency is acceptable for the average commercial applications. In particular frequency can affect complexity in an exponential way. Moreover these factors that can change quickly from an implementation to another (e.g. the use of a new technology can modify substantially the weights k_1,k_2,k_3,k_4).

In addition to these parameters, also pin count can give a measure of complexity. In fact more complex circuits tend to own more input/output ports. However in this case the dependency with complexity is not linear. In fact let us consider a net consisting of very complex blocks. When interconnecting them the resulting module is more complex than the single elements, but the pin count may not increase, due to the internal connections of the net. So this parameter may show a degree of saturation, that suggests to consider it apart from those listed in the previous equation.

One of the major challenge in the design community is to decrease such complexity and the global design time eliminating this factors of risk by reusing design experience ([2],[3]). The reuse of existing modules can give the designer stable design specifications, real data or useful hints on final implementation allowing an accurate preliminary design evaluation.

In the next three sections we analyze in more details the state of the art in reusability of models and improvements of their quality in compliance with the different design parameters. Finally the conclusions will show the planned future activities.

2. Models for reusability

Reuse of design experience (improving both specification and implementation phase) is the first technically sound proposal to take under control and improve design productivity. When taking into consideration the need (or risk) of re–design the efficiency can change dramatically:

$$GDeff = (\text{Design Complexity} * n) / (\text{Global Design Time} * p) \quad (4)$$

where n is the number of redesigns and p is a factor taking into account the percentage of redesign effort to be applied (p greater than 1).

More detailed models must be introduced to clearly evaluate the dependency of design time from such parameters. In the following we will consider cells of the same complexity, so the parameter DC can be considered constant. An expression more useful than GD_{eff} to express the dependency between design effort and reuse is S, the saving in design time. S is expressed as a percentage of the design time with no reuse (n times GDT) and is measured as the ratio between the design time required for a n–times reusable cell (GDT_{nr}), and the design time required for a set of n 'ad–hoc' cells. In model 1, a simplified model, S is a function of three parameters:

$$S(n,p,p') = 100 * \left(1 - \frac{GDT_{nr}}{n * GDT} \right) = 100 * \left(1 - \frac{(1 + p)}{1 + (n-1) * (1-p')} \right) \quad (5)$$

where the factor 100 is inserted for scaling, and:

- n is the number of different applications where the cell can be applied;
- p is the overhead in design time for the reusable version of the cell, needed for both VHDL code, and Verilog code (as it will be shown in the next section);
- p' is a correction factor measuring the experience when designing similar cells (this parameter is obviously subtracted to the total time needed to design n 'ad–hoc' cells).

The range of parameters p and p' vary considerably according to the type of cells and the application domain. In our experience p is less than 1, i.e. less than the time required to design the cell itself, (but theoretically it can be any number), p' is included between 0 and 1, with average range 0.2–0.5.

S can be any value less than or equal to 100. The limit case S = 100 is when the total design time is saved, and the design time is 0. This is the extrapolated value (infinite efficiency and reuse). When S is negative, overhead overcomes the advantages of potential reuse, i.e. design time increases due to re–use. S equal 0 means a precise balance between the cost of overhead and future savings. In our experience, satisfactory cases see S ranging between 0 and 50.

The proposed model guarantees convergence for the limit case n = 1. In fact, when n = 1 and the cell is not designed for reuse (p = 0), S = 0 (as it should be). If n = 2, then reuse is convenient if (1 – p') > p, i.e. if the design time for the overhead is inferior than the time to design from scratch the second cell. The factors p and p' have an opposite trend in function of the flexibility required for the cell to be reusable. Fixing the value of p' = 0.2 and varying the overhead p from 0 to 1.4 it results that n must be at least greater than 3 to have some advantage in the application of re–use. For a value of n greater than 6 all the curves tend to converge, i.e. the cost of the overhead becomes negligible.

In figure 2 the two families of curves refer to the two extreme values of p (p=0, p=1.4), with p' varying in the range 0.2–0.5. This graph shows the limit of this model, because for n=1, it does not converge to 0, as it should. This may be corrected in a less approximated model.

A more realistic approach (model 2) takes into account the fact that the design overhead is not constant for n, but depends on it. In fact the design overhead to produce a cell reusable two times is less than the design overhead to make it reusable n times. It is clear that there exists a proportional relationships between p and n. In a linear approach it gives:

$$S(n, p, p') = 100 * \left(1 - \frac{1 + \alpha * (n - 1)}{1 + (n - 1) * (1 - p')} \right) \quad (6)$$

where α represents the amount of dependence of the two parameters.

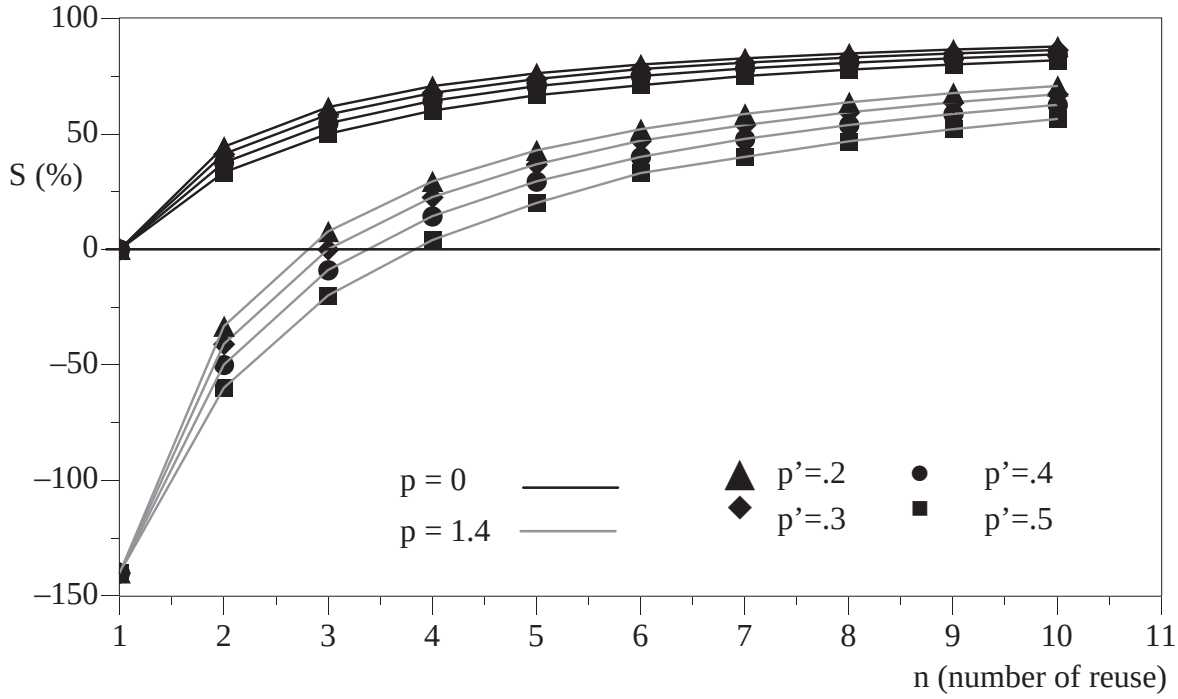

Figure 2. GDT saving S according to model 1

In figure 3 this relationship is plotted for the two extreme values of the range of α in our experience, i.e. 0.3–0.6. The parameter p' is allowed to vary from 0.2–0.6. In this case we see that some amount of gain (> 30%) is obtained when α is lower, that is the dependence of the overhead on n is not so extreme. The curve on the negative side is also interesting: it refers to a value of $\alpha = 0.6$ and a value of $p' = 0.6$. This case is possible, but not realistic because it refers to a case in which both p and p' are elevated, while in our experience they have an opposite trend.

What makes the previous analysis only approximate is the lack of stability in the parameters p and p', due to the lack of a systematic approach (design methodology) to face the task of reuse. This systematization is required because it could drastically reduce p, through the identification, dissemination and stability of the best techniques. Heuristics adopted in the field are considered in the next sections.

3. Heuristics for reusability in design practice

During the last years, many experimental approaches focused at improving design techniques for high–complexity digital integrated circuits. The emphasis was mainly put on the partitioning phase and the identification of sets of blocks (macrocells), able to promote reusability and to reduce design time. Many aspects advocate reusability: it may be adopted because of the requirement of operating in different heterogeneous system contexts (i.e. different network elements such as switches, servers and terminals); or it can be applied when considering different technologies (e.g. ASICs vs. PCBS) or to achieve different operating speeds maintaining the same functionalities (e.g. 155 vs. 622 Mbit/s). As a consequence it is necessary to define a taxonomy of cells to identify the best candidates for implementing reusable units.
Based on design experience, three main levels of macrocells have been identified:

- High level complexity cells, whose design value–added is not particularly high in company applications, so it could be useful to "reuse" them without any significant specialization and parameters definition. Typical examples are represented by microprocessor cores or physical layer modules, which are very constrained by standards. Such macros should be associated to their physical view (layout) and could be considered as hard macros. These cells belong to **level 1**;

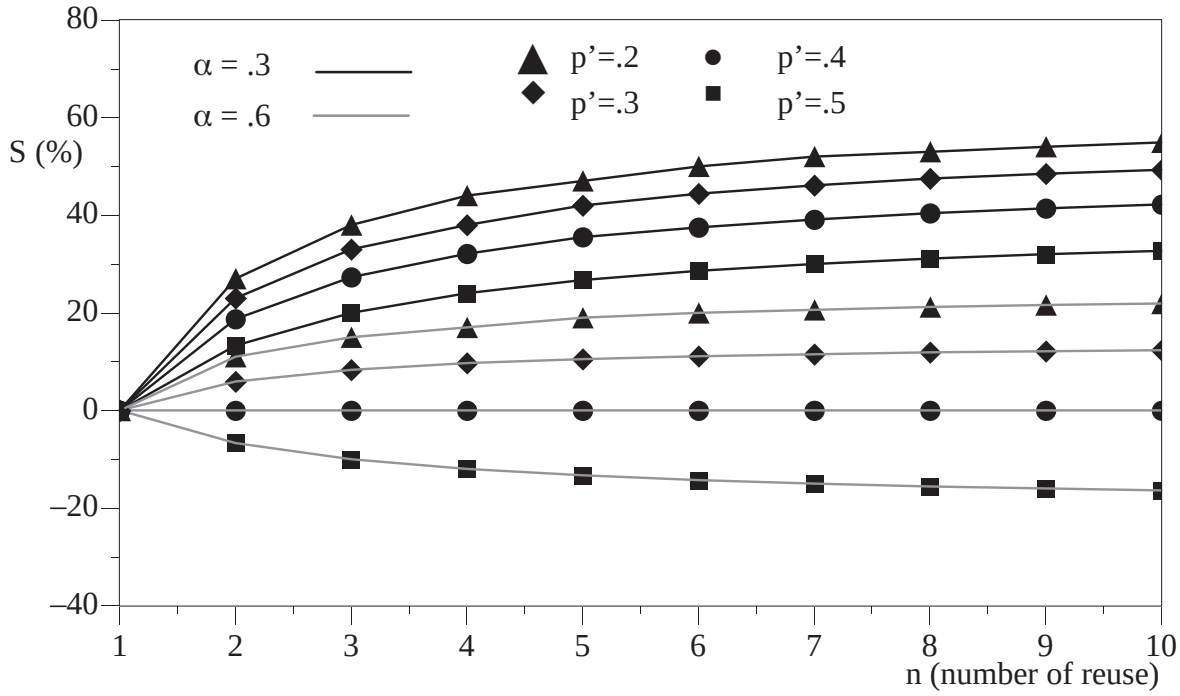

Figure 3. GDT saving S according to model 2

- Medium complexity cells with very high design value–added from a system point of view. They have to be "reused" in different contexts, so that the knowledge about their internal structure as well as the possibility of their specialization (soft macro) are very important (bus interfaces, microprocessor interfaces, filters and so on). These cells belong to **level 2**;
- A third level includes cells with mainly technological usefulness (ex: I/O, PLL, RAMs ...). They should be considered hard macros. These cells belong to **level 3**.

Such taxonomy is the result of the application of heuristic guidelines, formalized by the experience of senior designers. Among these guidelines, the most important to be complied with for the cell identification are:

- the cell should represent a system functionality useful in a wide set of circuits;
- the parameterization should be very flexible and carefully designed;
- even if some technological limitations prevent the use of all requested parameters into the cell implementation, an easy migration towards different applications should be anyway guaranteed by a correct design modularization;
- the cell definition should be abstract with respect to the first original use context: in other terms, the circuit interface and the datapath should be more standardized.

Considering such guidelines and the resulting taxonomy, cells of level 2 are the best candidates for implementation in reusable techniques.

3.1 The VHDL environment

VHDL supports some intrinsic concepts of reusability, and the available mechanisms are are exploited ([4],[5]). Among the applied techniques the following are the most used:

1. through parametrization (i.e. GENERICS and other traditional approaches), enhancing modularity;
2. implementing specialized sub–blocks, which are inserted under the control of flags (macros and similar);
3. exploiting automatic generation of specialized VHDL code to accommodate different cases, usually belonging to a very restricted domain.

These techniques are interleaved with more trivial approaches, like manually introducing modifications and refinements in a previously designed cell (trivially, a 'cut and paste' approach). The first

technique belongs to standard use of the language and needs no further comment. The last two are worth describing in some detail through examples taken from current industrial applications.

VHDL conditional generate statement emulates the conditional compilation capability of programming languages, such as C and Pascal. An example of syntax of conditional generate statement is:

<name_of_generate_block> :
GENERATE
<generate_body>
END GENERATE;

In this example, if <generate_condition> is true the VHDL code (<generate_body>), included between the keyword GENERATE and the keyword END GENERATE, are inserted in the code of the circuit. The <generate_condition> is a boolean condition generally depending on a parameter value. With this capability different replicas of a cell can include different solutions to cope with various needs, so increasing the reuse value of the cell core (invariable part).

In some cases external service routines, typically written in C language, are used to generate in an automatic way the VHDL code according to particular design needs, inside a well defined application domain. The cells are not completely specified; some modules of the VHDL code are generated ad hoc from case to case. The automatic generation of VHDL code is not a simple problem: it applies only to very restricted cases and this is a limiting factor that reduces the scope of this technique. The configuration of the project and simple functions/procedures in VHDL are examples of what can be efficiently generated in this way.

For example, in the ATM domain, a cell is characterized by a 'core' that can be used either in transmission mode to compute the Cyclic Redundancy Checks (CRC) or in reception mode to verify and correct the CRC. Information like the type of CRC or information about which ATM cell bytes are covered by CRC or working modes are given as input parameters to an external service routine able to create a project configuration and the functions to test and correct the CRC. The resulting code is just a sequence of IF .. THEN.. ELSE. It is of very low level, but avoids a tedious work to the designer.

3.2 The Verilog environment

Verilog has only limited built–in support for reusability, so a careful design style, version control and management and bug tracking, are key issues that have to be considered. In fact a Hardware Verilog description can be normally reused very easily without any extra compilation in several designs. If well documented, a 'plug & play' approach can be obtained; if the interface is bit by bit compliant with the new specification, simulation can start right away. So in Verilog, re–use does not depend on too many parameters or language constructions. Verilog blocks are easy to generalize but their structure cannot be modified easily through built in language features.

Nevertheless, Verilog does have some mechanisms for reusability that can be effectively used. Keeping the parallelism to the VHDL case, these are:

- parametrization (through the parameter instruction);
- modularity;
- conditional implementation of hardware inside blocks (under parameters control).

Parameters (and defines) should be thoroughly used instead of any hard–coded constant in the description, even if the design is not intended for reuse. A clever module decomposition is also of a great importance. The existence of common interface standards, as for example Utopia in the ATM field or the VME bus for microprocessors, may be very useful, as the may help in the modularization.

One of the main weakness regarding reuse within Verilog, is the lack of means for generating conditional blocks (like the VHDL equivalent generate statement). To circumvent this problem, the usual approach is to use some kind of preprocessor or language extension, and process the Verilog description before its use (several preprocessors exist in the public domain, based on perl scripts or yacc parsers).

This usually poses an additional problem on configurations, since Verilog support for this issue is very limited (only via parameters). Traditional software configuration control tools, as Make for example, have been used to cope with this problem. A careful organization of the design libraries is also crucial in this respect, as stated previously.

More traditional approaches [6], like the reuse of some parts of the code, or re–tuning of some existing modules (these are manual tasks), are the more common practices in Verilog reuse nowadays. It has been proven, that to help this task, a common coding convention is a must. Also a complete documentation must

exist for each module, this way a data–book of the more usual modules, like all kinds of counters, FIFOs, etc., can be built.

Verilog data types are clearly defined and built in inside the language (no need of packages), and quite well focused to the hardware representation. This, in despite of its limitations in the higher abstraction levels, appears to be an important advantage for reusability, as it simplifies the interfaces at the bit level and avoids data–type problems.

The use of generator programs, is another common approach to help in the generation of Verilog code, both for the functional description and for the test bench generation. However, as said in the VHDL case, its use is limited and represents a problem by itself.

The test issue, is a very important one, not to be forgotten when talking about reusability. Ideally, a test bench should follow each block, both for the functional case, and for the structural (or foundry) testing.

For the former, it has to be carefully considered, that as the configurability and generality of the block increases, so does the complexity of its testing (it may be very hard to test a very general block under all its possible configurations), and it may be worth to divide it in several smaller (less configurable) blocks.

Scan and ATPG techniques can alleviate the structural testing problem, and can also be delayed to the final design stages, when the block is within the whole circuit. When more complex blocks are merged in a design, as for example a circuit with several hard macros (micro–controller, memories, etc), reused blocks and some glue control logic, each of them can have a different test technique (scan, Bist, etc.), so some kind of in circuit boundary scan may be needed.

3.3 General conclusions on reusability

From this analysis it is clear that the degree of flexibility of a parametrized cell is not easy to define and to measure. It depends on the level of abstraction, on the type of applications, and on some imponderable human factors, such as the knowledge of the variants required in future for that type of cell/application. The evaluation of the benefits of the reuse of a cell is restricted to the case of cells belonging to similar applications. In these cases in fact it is more easy to fulfill the success prediction of the models with parameters as required, i.e. n greater than 3 and p less than 0.5.

An increase in the efficiency of the application of re–use can be obtained introducing methods and languages for the specification of application, which intrinsically support re–use in a larger measure than what is done currently by VHDL or Verilog. The object oriented extensions to VHDL are going in this direction, allowing both to increase the abstract level of description, and to reuse hw components in different application environments.

4. Quality improvement and productivity

Quality is defined in terms of relevant attributes for a particular use. These attributes have different weights depending on the considered requirements. Specifically for VHDL models, quality improvement means to increase reliability and suitability of the model in respect to further design steps as well as readability and efficiency in terms in both design and production cycles. Quality checking (human based or automatic) is useful in order to reduce or avoid unnecessary design iterations and consequently reduce design time and provide the products required by the market.

The proposed approach for quality improvements is based on a set of analysis tools that:

- measure the quality of VHDL models in domain such as source code complexity, simulation efficiency and confidence assessment, pre–synthesis estimation and testability;
- check that VHDL models are compliant with user specified VHDL subsets and writing styles.

Such subsets may be imposed by the design methodology or used design tools. In some cases they can be de–facto standards (such as logic synthesis subset), customized by each user to accommodate application domains.

To provide real added value such parameters must be measured as soon as possible in the design flow, i.e. before performing any simulation or synthesis. In this way they will improve models quality (faster simulation speed, better testability) and make them consistent with design environments.

Quality improvements impact mainly on three different aspects:

- efficient implementation parameters (area, timing, power, etc.);
- compliance with market requirements (testability, etc.);
- reduction in the design time (behavioral synthesis, simulation assessments, etc.).

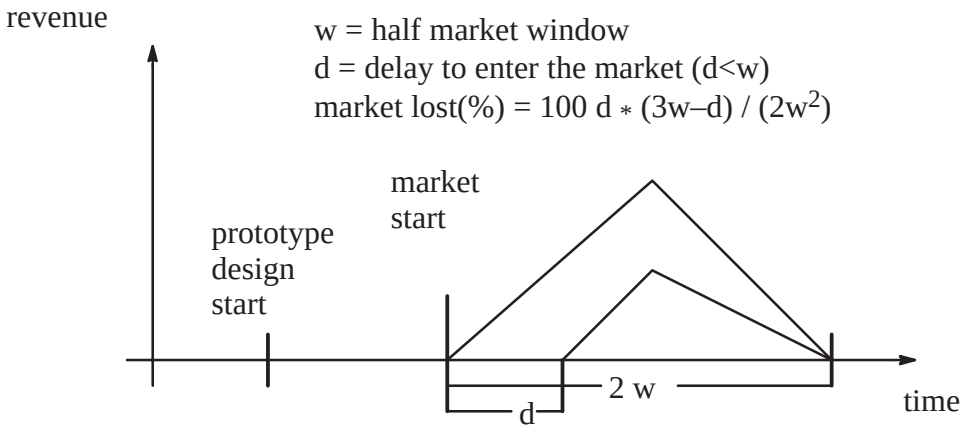

Figure 4. Simplified model of product life cycle.

Measurements of such improvements are obtained making explicit their impact on productivity and market satisfaction. This is described by their impact on time to market, which is a weighted average of the aspects previously considered.

Products life cycle in the telecom domain can be schematically divided in five stages: analysis of the future market requirements and acquisition of new standard developments, formalization of product specification in terms of functional requirements, prototyping, production and delivery, support and maintenance. In two phases of this life cycle the activity of the project are active, i.e. in the second and third. Formalization of specification and prototyping are usually time consuming tasks, and may delay the appearance of the product on the market with following business loss. A simplified marketing model (figure 4) shows that a time delay d to enter the market, due to a delay in the prototype development, impacts heavily on the figure corresponding to the market segment lost [2]. The model relies on some preliminary assumptions: the speed to enter the market is equal in the two cases, the culmination point has the same time coordinate, and the speed to leave the market is different. An interesting conclusion derived from the formula is that, when d is rather small (i.e. when d/w squared can be neglected), the loss is less or equal to 150 d / w, which shows in a clear way the influence of the delay. As a consequence it is extremely important to activate support methodologies to overcome situations when prototyping is shifted arbitrarily into the market window.

Quantitative figures are necessarily dependent on the applications. Expectations are in the range of a decrease of 30% in total design time of the prototyping phase. The gains are expected to be concentrated in the specification and verification phases (where specification/simulation are taking now 37% of global estimated design time) through some reduction in the re–design loops (taking now 21% of global estimated design time if including low level debugging). Other phases of design are also expected to improve (logic synthesis, testing), while low level simulations remain unchanged. Reliability is expected to increase less, because manual interventions will not be completely excluded from the design flow. Time to market may be reduced of 15%–20% due to the reduction in design time, and global sales will improve of different percentages, depending on the maturity of the device in the market (closer to 30% if the design is innovative and competition low, lower for a almost saturated and mature market).

The comparison between expectations and real gains can be accomplished only after an extended use of the tools under development in the project.

5. Conclusions

In this paper we have shown that model quality and re–use are among the major concerns of telecom application developers. Independently from the specification language used in design centers, designers have tried to apply heuristics to increment the re–use of models and to comply them to the quality standards required by the market and by customers. Both these items are perceived as fundamental factors influencing the global revenue and market penetration. Future research will involve the testing of the tools developed in the project and the evaluation of the expected enhancements.

References

[1] M. Mastretti, ”VHDL quality: synthesizability, complexity and efficiency evaluation”, Proc. EURO–DAC’95, pp. , Brighton 1996

[2] M. Bombana, G. Gorla, ”A different approach to IPR sharing”, Proc. EMSYS’96, pp. 206–215, Berlin 1996

[3] A. Barabanov, M. Bombana, N. Fominykh, G. Gorla, A. Terekhov, ”Re–usable objects for optimized DSP design”, Proc. EMSYS’96, pp. 433–442, Berlin 1996

[4] D.L. Perry, ”VHDL”, McGraw Hill, Inc. (1991)

[5] R. Airiau, J.M. Berge’, V. Olive, ” Circuit Synthesis with VHDL”, Kluwer Academic Publishers (1995)

[6] W. Tracz, ”Tutorial. Software Reuse: Emerging Technology”, IEEE Computer Society Press. Washington 1988