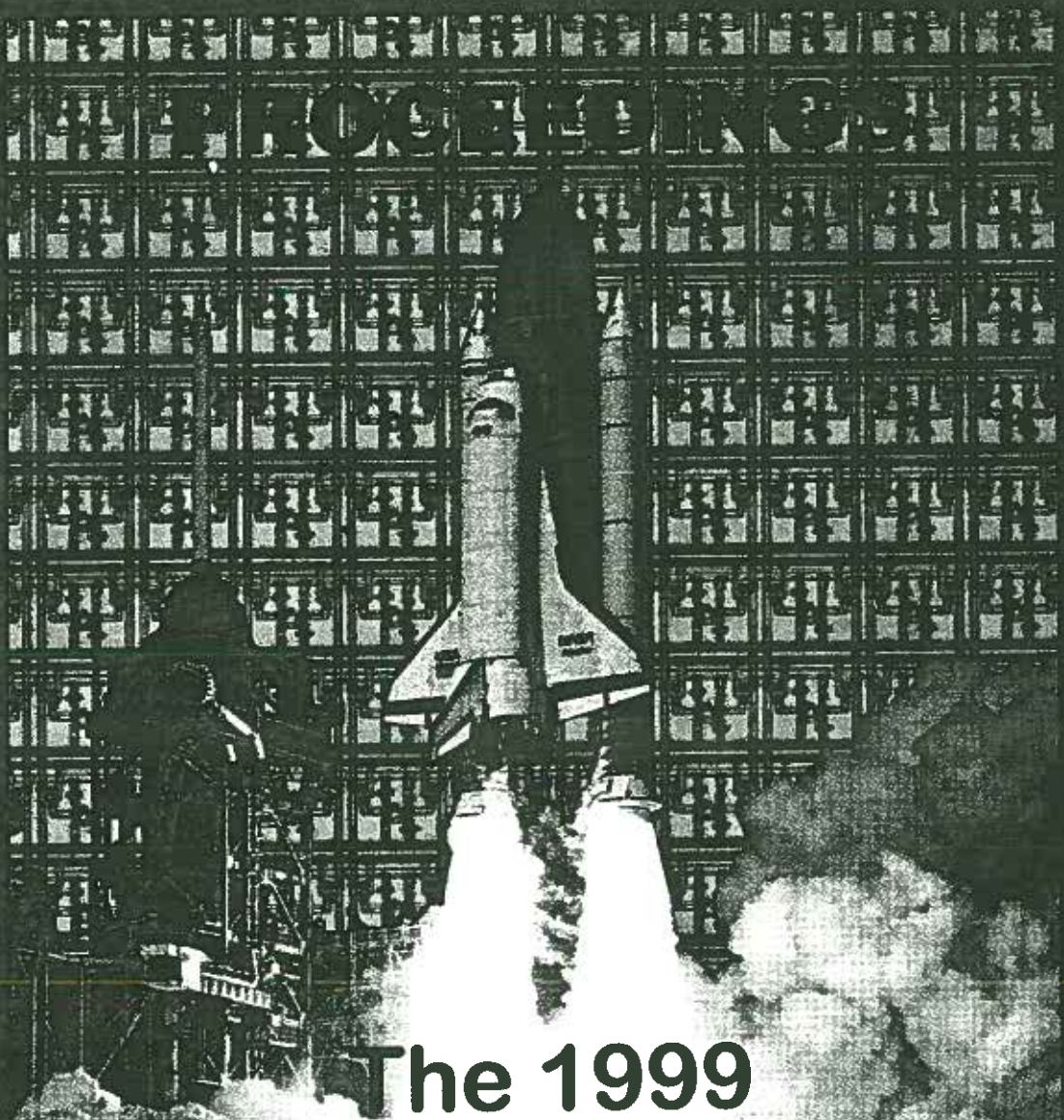
ISCAS '99



The 1999 IEEE International Symposium on Circuits and Systems

May 30 - June 2, 1999

A SYSTEM VERIFICATION STRATEGY BASED ON THE BST INFRASTRUCTURE

Gustavo R. Alves[1,2] and José M. Martins Ferreira[2]

[1]ISEP / DEE
Rua de S. Tome
4200 Porto – PORTUGAL

[2]FEUP / DEEC
Rua dos Bragas
4000 Porto – PORTUGAL

ABSTRACT

A good verification strategy should bring near the simulation and real functioning environments. In this paper we describe a system-level co-verification strategy that uses a common flow for functional simulation, timing simulation and functional debug. This last step requires using a BST infrastructure, now widely available on commercial devices, specially on FPGAs with medium/large pin-counts.

1. INTRODUCTION

System-level simulation enables a system designer to verify, before any real hardware is produced, the design correctness. If the system contains devices whose functionality is determined by a program stored in memory, then the term co-simulation is better applied for describing this action. CPLDs and FPGAs are currently being used for system rapid prototyping. These devices, specially those with larger pin-counts, are being released with a BST infrastructure [1] for some years now. Development systems for these devices are now extremely powerful and versatile, enabling complex designs to be entered in multi-level forms (from schematic to hard/soft IP cores), simulated, synthesised, fitted, re-simulated (with delays) and finally programmed into one or more devices, either by using appropriated hardware platforms or just a simple cable connected to the PC serial / parallel port. Some devices, may be programmed / configured through the TAP, thus enabling quick and efficient in-system alterations.

In view of all these advantages, we proposed ourselves, in a recent system design to draw a system-level co-verification strategy that could, early in the specification phase, combine and explore the potentialities of the Altera Max+Plus II environment and the presence of a BST infrastructure on all devices belonging to the EPF10K family [2].

The system architecture is described in section 2 and the co-verification strategy is described in section 3. Section 4 presents the conclusions and the current status of our work. Section 5 concludes with the references.

2. THE SYSTEM ARCHITECTURE

Our system comprises two generic devices with an extended BST infrastructure, one emulating an 8-bit non-inverting unidirectional buffer ('244) and the other emulating an 8-bit latch with tri-state outputs ('373), a dual-processor controller, and two memories containing the program executed by each processor. One of the processors controls the extended BST infrastructure included in each generic device. As each one of these devices is to be implemented in an FPGA from the Altera EP10K Family, already containing a BST infrastructure, in the end the device will have two TAPs (the original TAP connected to the pre-implemented BST infrastructure and a second TAP that is part of our design). The second processor controls the system clock. It contains a group of 16-generic inputs and a group of 16-generic outputs that can be used for any generic purposes.

Figure 1 illustrates the system architecture. ROM1 and ROM2 are used for simulation purposes, meaning that they do not correspond to FPGAs. The controller was implemented in an EPF10K30, and the '244 and '373 were implemented in two EPF10K10. Original TAP pins, power pins and other dedicated pins belonging to the device are usually not represented.

Our goal was to specify, develop and verify a system-level debug and test infrastructure based on a built-in controller and an extended BST infrastructure, that could be used for functional and timing debug. The built-in controller would be responsible for controlling the system BS chains and the system clock, thus guaranteeing the synchronism between the system functional and test logic. The extended BS infrastructure would provide support to Breakpoint (BP) and Real-time analysis (RT) operations. For BP operations the BS register is configured to detect a condition corresponding to values present at the input pins or outputs from the component functional logic. For RT operations the BS register is configured to:

- Store a sequence of two contiguous vectors.
- Store a sequence of two contiguous vectors after a certain condition is found.
- Store a sequence of two contiguous vectors until a certain condition is found.

The system's functionality is described in another paper. In this document we will focus on our system verification strategy.

3. THE VERIFICATION STRATEGY

In the system specification its was decided to set up a sound verification strategy that could address the following steps: functional simulation, timing simulation, and functional debug. Structural test was also part of the verification strategy, although it was addressed as an independent task, mainly done through the original BST infrastructure of the FPGAs, that unfortunately is not supported by the model generation tool of the Max+Plus II development system.

0-7803-5471-0/99/$10.00©1999 IEEE

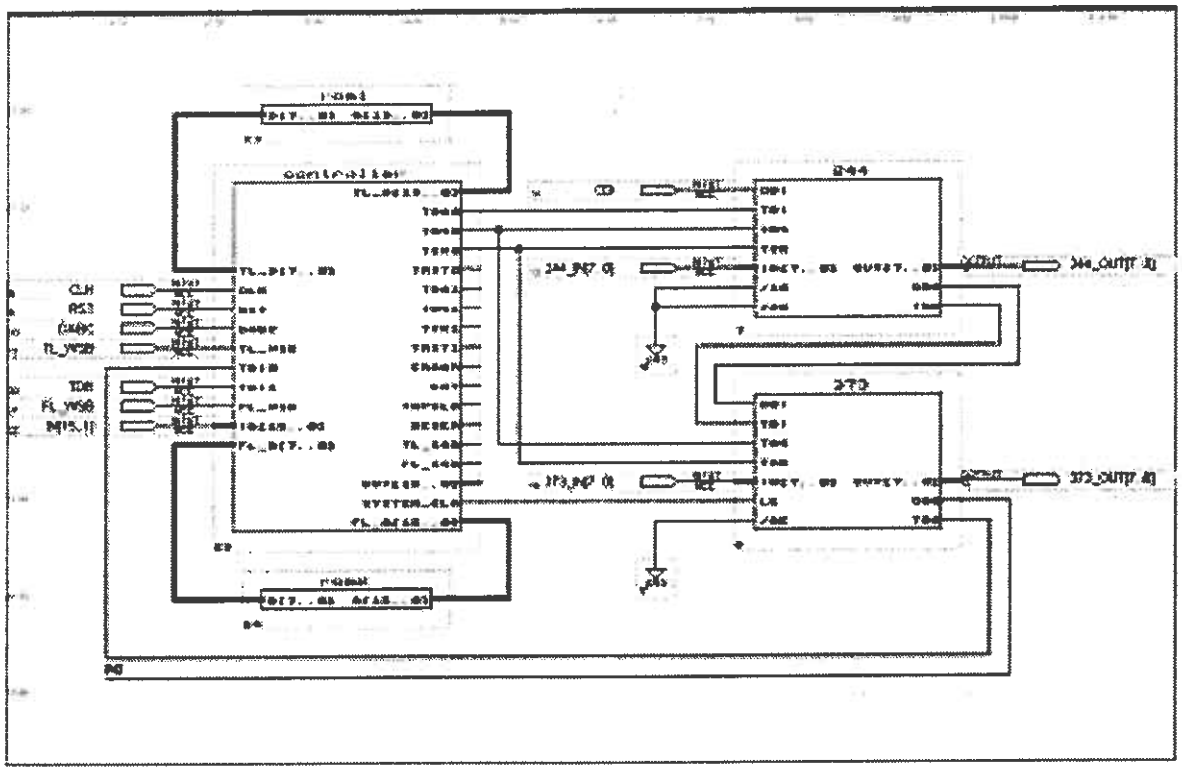


Figure 1: The system-level architecture.

The system specification included the specification of each individual component, and some small debug and test programs to be executed by the dual-processor built-in controller, which was named PRODEP, an acronym that stands for "PROcessors for DEbugging Purposes". The design of each component evolved in a mixed of top-down, bottom-up, block-based design, where some parts corresponded to previously developed blocks. For instance, one of the processors belonging to the controller corresponds to an enhanced version of a board-level BIST processor [3]. The first verification stage corresponded to functionally simulating each component, individually, with a small number of hand-generated test vectors. This first pass enabled some confidence on the component's functionality, and also specific details to be thoroughly covered. Functional simulation of the controller included two stages, one similar to the previous one and another where the controller typically executed very small programs. To achieve this, a test environment comprising one controller and two memories was set up. This consisted of two devices, each containing one Library of Parameterised Module (LPM) corresponding to one ROM, connected to the controller. Functional models of each device were first generated and a "system-level" linked functional model was then created using the capabilities offered by the Max+Plus II compiler.

The debug and test programs were written in assembly, and the object code and list files were then generated by a small freeware application that accepts table-defined instruction sets. A small in-house developed application then took the list file, and using a template file, produced a Memory Initialisation File (MIF) that is read by the simulation tool integrated in the Max+Plus II environment. This process is illustrated in figure 2.

The second verification stage corresponded to a functional simulation of the all system. The functional models of each component were "linked" by the Max+Plus II compiler and larger programs were written, assembled, and converted to MIFs, for performing the system-level co-simulation. After detecting, diagnosing and removing all design errors (at this stage) a golden simulation file with input stimulus and output vectors was produced, for later comparison during the timing simulation phase. A table file in ASCII format, containing the values for all system pins, was also generated. Figure 4 illustrates these first design steps. The design flow then proceeded to the synthesis phase. Figure 5 illustrates the following design steps.

After the synthesis and fitting process, a timing model of each component was generated for individual timing simulation by re-using the input stimulus. In the waveform visualisation tool, integrated in the Max+Plus II environment, the values present on the component outputs were compared against those previously stored in the golden file, produced after the functional simulation stage. Although this was a manual process, if an error due to a long-path occurs, the component behaviour diverges significantly, and the detection is generally easy (also due to the small length of individual simulation files). After removing all errors at this stage, the design proceeded to the system-level timing co-simulation.

A "linked" timing model of the system was first generated by the Max+Plus II compiler, and the small test programs used during system-level functional simulation were now used for system-level timing simulation. At the end of this verification stage, after removing all detected errors, a new table file was generated for automatic comparison with the table file generated during system-level functional simulation. This automatic process, required an

intermediate step where the vectors, not corresponding to moments relative to clock positive edges, were removed from the original table file. Comparing the two files corresponded to comparing the outputs generated during functional and timing simulation. This automatic process enabled larger programs to be written and verified, without recurring to tedious visual inspections on rather extensive waveform files.

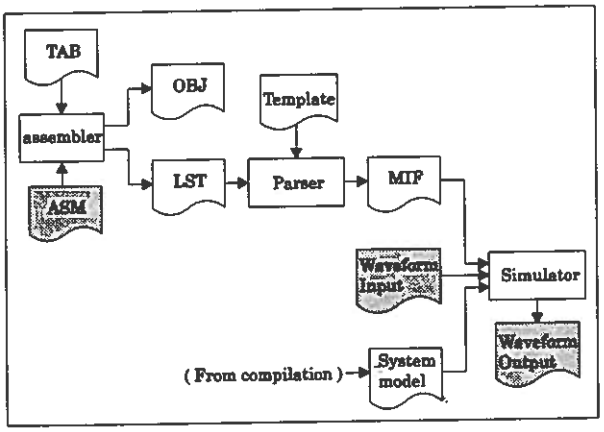


Figure 2: Integration of an assembly file in the simulation process

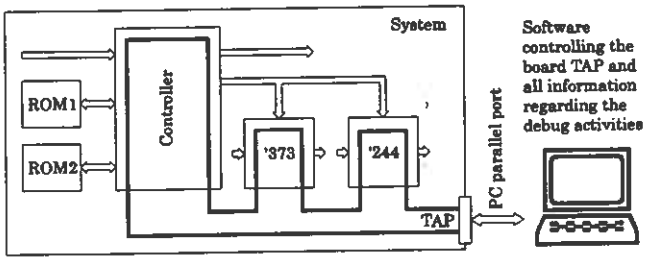


Figure 3: Using TAPPER for system-level functional debug.

The next phase consisted of creating a board with the system, programming the FPGAs, downloading the test programs (executed by the two processors) to the system memories, and performing the system functional debug. This verification stage was carried out using the original BST infrastructure existing in the FPGAs and a PC-based application program called TAPPER, able to control two board BS chains. This program emulates the referred BIST processor, by executing the same instruction set, and controlling/reading the board TAP signals from the PC parallel port. Functional debug corresponds to verifying in-circuit the values obtained during functional simulation [4, 5, 6]. This verification stage is done in the following way:

1. TAPPER shifts the Sample/Preload instruction to all system devices
2. TAPPER places TAP controllers in *Select-DR*
3. The system primary inputs are externally fed with the right stimulus
4. One clock-pulse is applied to the system
5. TAPPER places TAP controllers in *Shift-DR*, through *Capture-DR* (where values appearing at component pins are captured in the BS register), and the first vector is shifted out and compared against the vector stored in one of the table files.
6. Repeat steps 2:5 until all vectors are compared.

The test program executed by TAPPER is generated by an in-house developed application that uses as input information: the BSDL files of the FPGAs, a configuration file, and the table file (for creating the expected values and masks for the comparison). The functional debug environment is illustrated in Figure 3.

4. CONCLUSION AND FUTURE WORK

We described a system-level co-verification strategy based on the potentialities enabled by the Max+Plus II environment and the presence of a BST infrastructure on commercial available FPGAs. Our strategy included four main verification phases: functional simulation, timing simulation, structural test and functional debug. Functional simulation provided initial information for the remaining phases. Vectors obtained during timing simulation were compared, at clock edges, with those generated during functional simulation. After simulating the whole system, the design proceeded to the prototype phase, where functional debug took place on real hardware. The original FPGAs BST infrastructure was used for sampling the pin values, on a vector-by-vector basis. Each vector was compared against the expected vector, extracted from the table files generated at the end of the functional simulation stage.

Although it may be confusing that there were two BST infrastructures, it should be reminded that we were developing a system-level debug and test infrastructure, and because the FPGAs used for prototyping purposes, already contained a BST infrastructure, in the end they co-existed, one implemented at the silicon foundry, the other implemented by us. We used this approach for debugging our own debug and test infrastructure. In a near future, we plan to incorporate the extended BST infrastructure (implemented on each device) and the built-in controller, in complex systems. The debug and test infrastructure will provide the appropriate mechanism for system functional debug and timing debug. This last step was not performed, because TAPPER does not run the test program at the same operating speed of the built-in controller, and because the original FPGAs' BST infrastructure does not match our extended BST infrastructure. Note that we could not use our controller for debugging our present system, because there would be an overlapping, causing potential conflicts on where could be located a possible error.

5. REFERENCES

[1] IEEE Standard Test Access Port and Boundary-Scan Architecture, Oct. 1993, IEEE Std. 1149.1

[2] The Altera web page: http://www.altera.com.

[3] J. M. Ferreira et al., "Automatic Generation of a Single-Chip Solution for Board-Level BIST of Boundary Scan Boards," *EDAC Proc.*, March 1992, pp. 154-158.

[4] M. F. Lefébvre, "Functional Test and Diagnosis: A Proposed JTAG Sample Mode Scan Tester," in *ITC*, pp. 294-303, 1990.

[5] Richard Sedmak, "Boundary-Scan: Beyond Production Test," in *ITC*, pp. 415-420, IEEE Computer Society Press, 1994.

[6] Andy Halliday et al., "Prototype Testing simplified by Scannable Buffers and Latches," in *ITC*, pp. 174-181, 1989.

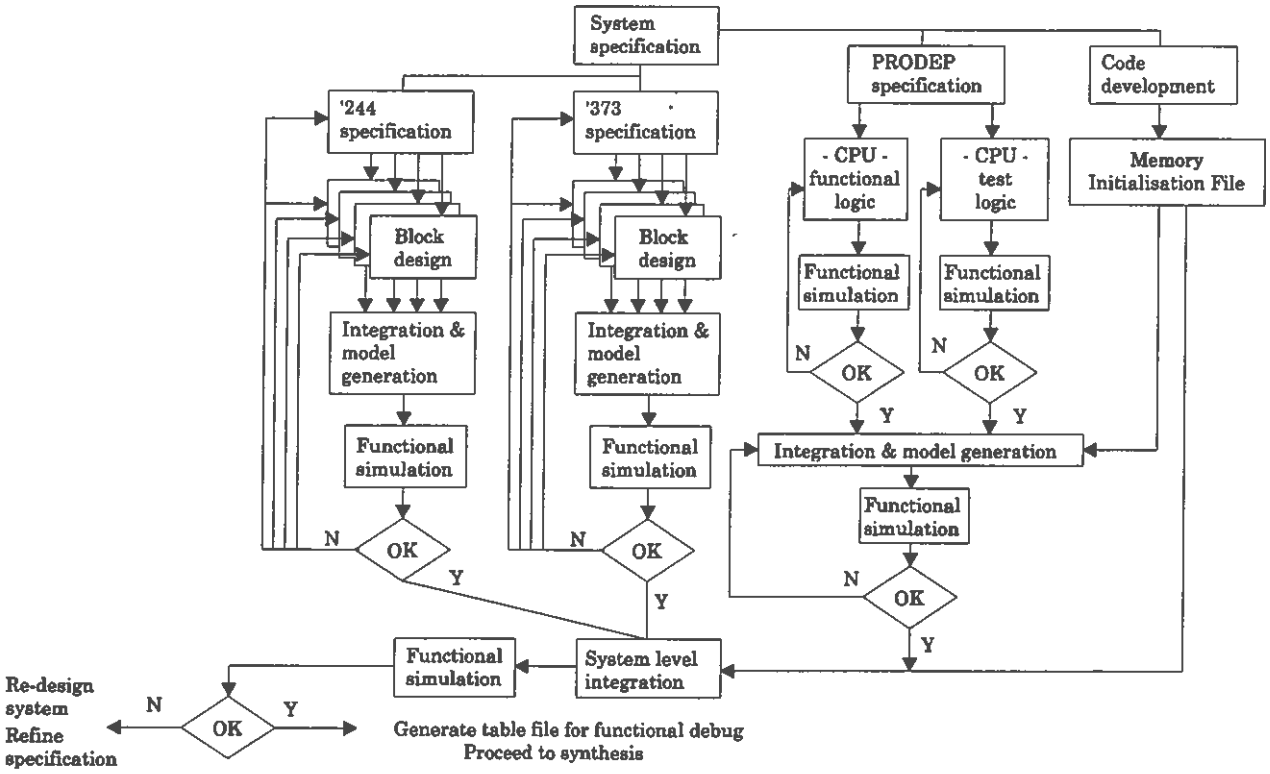


Figure 4: Design steps leading to the system-level functional simulation

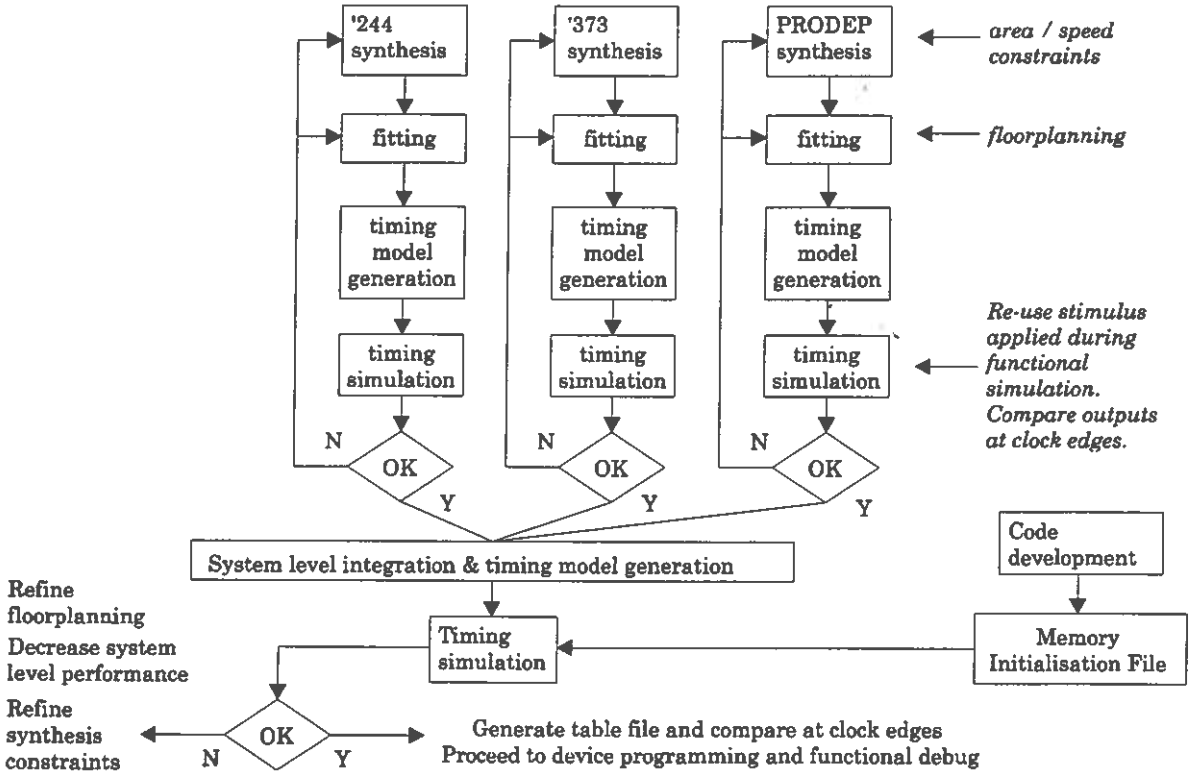


Figure 5: Design steps leading to the system-level timing simulation