Meeting Time-to-market Challenges
In Large-scale VLSI Design Projects

TATA CONSULTANCY SERVICES

Authored by : Hrishikesh Sharma
Prashanna Venkatesh B
Date : 21 December 2004

Reviewed by : Subramanian P S
Sreedhar D N
Rajarama Nayak
Sherlekar S D
Date : 21 December 2004

Approved by : _____ Date : _____

Meeting Time-to-market Challenges

Printing the document

To print this document optimally, take colour print outs of pages 12, 13 15, 17, 18, 19, 22, 23, 27, 28, 32, 33, 35, 37, 40, 42, 44 and 45. All other pages can be printed in black and white.

Viewing the document

Pages 18 to 44 are best read on screen with a magnification of 125% or higher. Other pages can be read at normal zoom.

About The Document

In design and development projects for IC design, there exists a huge opportunity to exploit the 'activity pipeline' in the design processes. The document highlights such pipelining in the VLSI design process flow for the Centre of Excellence in Embedded Systems, at Tata Consultancy Services Ltd. It also proposes a list of "cuts-across-the-flow" in *any generic VLSI design process flow*, which can potentially be used to **expedite** the design and development efforts for any semiconductor division. *Time-to-market* being the driving business requirement for semiconductor industry now-a-days, such approaches to boost throughput are indeed the need of the hour. Whenever the nature of VLSI *design methodologies* is similar, this approach can also be exploited to co-ordinate throughput enhancement in the joint VLSI design and development efforts of two divisions/organizations, each of which has its own design process.

Target Audience

The main aim of this document is to elaborate on *one way* to enhance productivity; that is by exploiting the Zonal Time Differential. When it comes to productivity, managers will by default have a go at the document. Next come the architects, who have a definite say in decision of design methodology for particular project. Choice of design methodology is an important aspect of work-sharing, as we bring out later. Finally, process and quality analysts will like to see, how productivity can be increased, while working under a tight quality control framework. However, certain sections contain details, which will interest only peoples such as architects, and hence they will be able to appreciate this work better.

Information provided to Audience

We have provided a running example of a design project, only to make our proposed approach clear. At each stage in a *generalized process flow*, there are potential cuts-in-the-flow, where work-sharing can happen. We provide a novel scheme to decide the sharing-points(or cuts). To **help the decision-making process** for work-sharing, we enumerate such points for all stages, and also provide the details of information and its exchange process details at each such cut.

Document Organization

The document is organized as follows. A section of introduction brings out the purpose of our work. It is followed by a proposing a generalized approach to productivity enhancement. We introduce at this point, a complex design project example, which we will be referring throughout the document. Here, the *generic* VLSI design process flow relating to the Centre of Excellence in Embedded Systems Division is introduced. New notions such as cuts, flows and protocols are introduced at this point. Now, the required setup to make such distributed process operational at the identified cut-points is elaborated. This can give a clear idea of the kind of coupling required between such engaged parties. Before it concludes, a section suggests another partitioning approach to VLSI design flow, where the partitions are somewhat tight-coupled.

Contents

List of Figures

List of Tables

List of acronyms

ALU	Arithmetic and Logic Unit
ASIC	Application-specific Integrated Circuit
ATPG	Automatic Test Pattern Generation
DFT	Design for Testability
GMT	Greenwich Mean Time
HLD	High-level Design
IC	Integrated Circuit
LLD	Low-level Design
MST	Mountain Standard Time
PPA	Partially Parallel Activity
R&D	Research and Development
RF	Radio Frequency
R&D	Research and Development
RTL	Register-transfer Level
STA	Static Timing Analysis
VLSI	Very-large Scale Integration

1 Motivation

Driven by the rapid speed of deployment of various electronic products in consumer market, the semiconductor design industry is facing high pressure on *delivering newer solutions at equally high speed*. This pressure is compounded by the fact that the products continue to become more complex and heterogeneous, thus adding more challenge to VLSI design projects due to strict design, electrical & performance requirements. This challenge has fuelled the interest of architects and process experts alike.

Centre of Excellence in Embedded Systems, a division of TCS, Bangalore, offers state of the art Products and Services / Solutions in the areas of ASIC/VLSI [Analog Mixed Signal, RF and Digital IC Design], Multimedia, Wireless, Adaptive Embedded Systems and other high end emerging technology products to the global market. The division has got a unique VLSI design process flow, enriched over years by the experience of its various architects. Owing to the challenge mentioned above, the division has decided to investigate and deploy potential innovative solutions to the same. The role of the various customers in the design process being equally important, one benchmark for such solution has been decided to be the **generality** of such solution. In that case, a "push" rippling through the value chain of the electronic industry will benefit all the parties at various locations in the value chain. Further, there will be times, when the division participates in activities at the same location in the value chain. Here again, given that the processes for that activity may differ for the participating parties, the generality of solution becomes a necessity.

2 A Proposed Approach

There are many ways of dealing with such time pressures. For white-elephant ICs, typical way is to restrict the number of features of the IC. For ambitious IC designs, where the standard of some performance criterion/criteria have been pushed up, a lesser ambitious standard is set. It sometimes reduces the cost of R&D involved in designing new algorithms and architectures. At other times, it works by reducing the cost of quality assurance(such as less than 90% test coverage). There are times, however, when one cannot try to resize the original specifications. One can try to cut down the efforts by using better "automation tools"; this forms part of trying to use a different design methodology. Usage of these cutting-edge tools only partially alleviates the problem; much more time is required to complete the other activities in a design project, than the ones which require tool usage. Once the design methodology is fixed(say, new tools cannot be acquired because of budgetary issues), then the effort estimates hardly vary. Different component vendors may still give differing effort estimates. But they do that by cutting the cost at various nooks and corners, and hence the difference is not substantial. It is at this point that one can think of *enhancing the throughput, or the productivity*.

2.1 Shrinking Time by Throughput Enhancement

As the term 'throughput enhancement' suggests, it implies increase in the output during work-flow on a particular project, visible at the end of a time unit, such as a day or week. Needless to say, that if enhancing throughput is found to be practical, then the number of time units taken(say, months) to deploy a design solution will come down.

2.2 Activity-level Shrinking

Following [1], we quote here few of the relevant terms used in processes for design and development projects, in general. For example of these terms, please refer to figures in section 4.

Process The proposed way of running and finishing a project

Phase Different *similarly-structured segments* of the project, i.e. designing a solution. Each phase is marked with a distinct point of start, and distinct deliverables to end with.

Activity Part of work in a phase, which entails a non-trivial duration-cost. It is also known as *work-package* in certain industries.

Procedure A particular method of performing an activity

Task The *atomic* piece of work, and hence a part of an activity

To enhance the throughput, one can cut down on idle times inherent in the processes. This can be efficiently done by looking at the flow of the design process, and investigating those points, where work can either be parallelized, or can be assigned in a shift manner to various parties. While the first point suggests 'pipelining' the activities, the latter point works by ensuring round-the-clock production. Scheduling work in this manner at task-level can provide the true picture of time-compression. But doing this is inefficient, because of the sheer amount of management (synchronization) overheads. Hence, we propose a method to compress time usage at activity-level.

2.3 Design Methodologies and Pipelining

While dealing with the *first* suggested way of increasing throughput(in previous section), we use the term 'pipelining' in a slightly modified way here. Traditionally, pipelining implies that a future activity, which is not dependent on the outcome of the activity being done presently, is done a-priori by assigning it an idle resource. In this way, the *relative independence* of the individual activities is moulded into *concurrence*, or *parallelism*. We *broaden* the definition of pipelining to involve additional-resource assignment to an activity next in sequence, while the resource dealing with current activity **mandatorily** shuts off. Because different resources are employed to carry out these *concurrent* or *alternate* activities, synchronization at various points is required. It can only happen if these *threads of activities* adhere to some common framework. This choice of design methodology provides such common framework.

Following [2], the design methodology can be broadly defined as evolution of architecture, choice of models of computation(such as gate- and RTL-level for VLSI, and dataflow for signal processing), the break-up of design problem into sub-flows utilizing these models of computations. These sub-flows involve synthesis and refinement across the chosen models of computation, and hence the choice of tools is also part of the design methodology.

The importance of having design methodology as the common framework becomes more conspicuous, when we look at distributed operations. When two parties(e.g., two divisions) are jointly undertaking a project, activity-level pipelining can be implemented. However, if the design methodologies adopted by these parties differ, then there will be times, when few problems arise. It can be that either the activities which they carry out are simply incompatible, or that they do not synchronize *on time*(one party has some old simulation tool, and the results surface after long time, than required by the other party). Hence while trying to identify parallelism, all involved parties should always be looking *within* a single design methodology.

2.4 IC Design Methodologies

The IC design industry has been around now for four decades, with multiple established players trying various options to optimize their in-house design methodologies. New techniques such as virtual prototyping, and newer tools such as RTL-level timing analyzers keep on coming up. But because they establish themselves over existing firm ground of design methodologies area, they act more as innovations in design methodologies, rather than inventions. It is therefore, highly possible, for multiple parties, to suitably adapt their design methodology, whenever they enter into doing joint work. Adopting a common design methodology is the first step towards productivity enhancement in such scenarios.

2.5 Example Manifestation of Pipelining: The Zonal Time Differential

A very relevant case-study, or curious example, is the way IC design teams across geographies have managed to reduce the idle times. We take example of two time-zones: India(5.5 hours past GMT), and MST/USA(7 hours before GMT). Thus, approximately, the day period in Oregon or Arizona is complementary to the day period in India.

Once the design methodology is fixed, the (detailed) design process can then be worked out, part-by-part. This leads to evolution of the (overall) project schedule and management plan. The task(activity)-dependency graph model is one of the initial parts of the schedule. It is directly derivable from the choice of design methodology, and tailored by the manager's/organization's experience of conducting various activities related to it.

One can immediately see that the study of pipelining in these circumstances boils down to *resource-constrained scheduling analysis* over the dependency model. This analysis is akin to that done in high-level synthesis, with further constraints, and hence it has been independently studied. Following [4], which addresses this problem in its most general form, the following forms the elements of this analysis problem.

i. The activity-dependency model. It can also be task-dependency model, if it is not too detailed.

ii. Multiple teams representing 'm' resources; with each resource having distinct, partially overlapping, or similar competencies. This is equivalent of saying that the scheduling is constrained by having ten multipliers, four ALUs and seven adders. In our example, m assumes the value of two. The functionality of these resources is a function of the *intersection* of group/division's competencies with the design methodology[1] chosen for the current project.

iii. Scheduling is further constrained by the fact that the teams will work in their day-times. The Indian team works in day shift convenient to itself, while the Oregon team works in the its day shift. If one looks at timing relative to GMT, then one can approximately claim that the two resources are shut-down in complementary time periods.

One can immediately sense that such setup has significant chances of increasing the throughput. It is not just from the point that complementary shutdown removes certain idle periods, but also that a competent resource(team) reduces the efforts involved in a particular set of activities. Doing this primitive analysis is not just a confidence-building measure, but also stepping stone, over which work distribution amongst various teams can be evolved. The information synchronization at shift boundaries itself takes time, and this can lead to an aggregate increase in overall duration. However, compared to the time taken by the activities themselves, such synchronization time is far less. It comes from the fact that it is mainly required to absorb the information transferred rather than physically transferring, and a closed-loop approach can reduce time taken in absorption.

2.6 Implementing Time-differential Activity Parallelism

The pipelining analysis over activity-dependency model requires evaluation of the following:

- The competencies of various teams. In our example, we assume that the parties entering into work co-ordination(such as two divisions) are *equally competent*.

- The duration of each work-package, or activity. In our example, where we use the complementary zonal time difference, one can choose work do-able in 12-hr units as candidates for overnight scheduling at other location. Having similar competencies is an extremely advantageous situation for both sides in this case, because it is always possible to split any activity which requires more than 12-hr unit into multiple, time-complementary units of work at different locations. It is not just these activities, which can be chosen as candidates. Many activities are complete enough to be scheduled for around a week's duration at other place. In that case, *concurrency* is exploited to expedite the work. We **differentiate** between such 12-hr work-packages with those packages, which can be done in parallel at other place, but over, for example, a week's duration.

[1] This simplified assumption does not account for on-the-project training and learning. Such simplification is almost never the case: most of the while project deals with new learning of at least few things such as tools and technologies.

- The nature of each work-package. For example, if synthesis fails, the best person to analyze the reasons is the RTL programmer. Hence, giving a synthesis run might not be shifted to another location.

- Cost of information handover between two locations. An implicit parameter during information handover is the communication efficiency of coordinating parties.

We take a small example, and explain our approach of deciding these. It is based on the colouring and the maximum flow problems of graph theory.

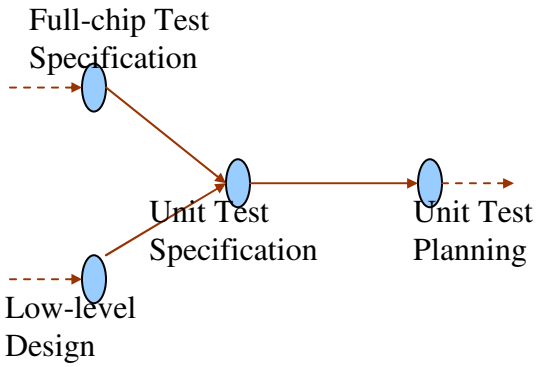


Figure 1: Example Segment of Dependency Model

The idea is that if one activity is chosen to be done by another team, either alternately or concurrently, an information handover has to happen. New, processed information has to be handed back at the finishing of the activity as well. However, it is required only if the next activity is not to be done by the same team, hence one can just focus on one handover at a time.

Initially we assume that there the synchronization overhead is negligible. One can start from the "project start" activity, and consider each arrow one-by-one, while scheduling. In our example, let us assume that Full-chip test specification, and low-level design specification has been decided to be done by a team in Oregon. Now, when one considers then next activity, unit test specification for scheduling, one will consider the capability, efficiency etc. of various teams, before selecting the candidate ones. Suppose that some team, other than Oregon's, makes a very promising candidate, in terms of effort cost. But since start of unit-test specification activity depends on end of two other activities, the small synchronization activity needs to be added to each arrow that ends in unit-test specification activity. If the costs are still suitable, the unit-test specification activity can be re-coloured. The re-colouring signifies that a particular team(having representative colour) has been scheduled and assigned for an activity.

Now let us take the case, where the overhead of information synchronization/handover, though small, is not negligible. At each handover, there is a little effort involved to transfer the information(e.g. problem reports), as well as a duration. If the handover is carefully planned, the duration can also be brought down. Initially, the whole set of arrows in the dependency diagram is annotated with these two properties. It is assumed that from the experience of managers, values of these properties at various points are determinable. Again, one can start from the "project start" activity, and consider each arrow one-by-one, while scheduling. Yet again, let us assume that Full-chip test specification, and low-level design specification has been decided to be done by a team in Oregon. Now, when one considers then next activity, unit test specification for scheduling, one will choose the candidate ones in the similar fashion, as described before. Suppose that some team, other than Oregon's, makes a very promising candidate, in terms of effort cost. In that case, one

additionally needs to check the impact of information handover. For that purpose, a secondary activity for synchronization is added to the dependency diagram. The cost of this activity is already present in the corresponding annotated arrow. But since start of unit-test specification activity depends on end of two other activities, the synchronization activity needs to be added to each arrow that ends in unit-test specification activity. After this modification, one needs to re-check the costs. If the costs are still suitable, the unit-test specification activity can be re-coloured. The re-colouring signifies that a particular team(having representative colour) has been scheduled and assigned for an activity. If the next activity is scheduled and assigned to the same team, the secondary activity node need not be added.

The scheduled diagram can, for example, look like the following.

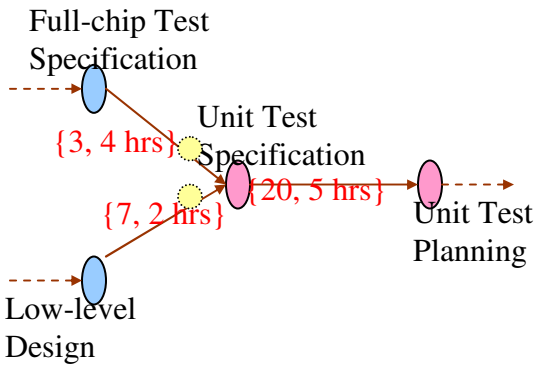


Figure 2: Scheduled Segment of Activity Diagram

In the scheduled diagram, if the two ends of any arrow have different colours, we will say that the (control-)*flow* of process has been *cut*. The synchronization activity inserted in the *flow* represents some kind of an *interface transducer*: details such as the type of data, protocol of data exchange need to be worked out across any such *cut*. These details will provide effort estimates to the synchronization activity.

In the subsequent chapters, we will take a similar cut-exercise the process flow diagram for Centre of Excellence in Embedded Systems.

3 Example VLSI Design Project

A complex design, such as that of a system-on-chip, involves following components

- processing elements, one or more
- storage elements
- peripheral elements
- bus structure
- specific functional elements, which are commonly called as IPs

Let us take an example of project intended to design the receiver for a 3rd generation mobile phone. The block-level architecture of such a diagram is provided in Figure 3. It involves

- A general-purpose RISC processor, and a special-purpose processor such as DSP
- Different storage elements such as Flash memory, SRAM
- Interface to different peripherals such as USB, UART, IrDA, I^2C, Synchronous Serial Ports, General-purpose I/O, colour LCD etc.
- Bus controllers for AMBA bus and AHP/APB bridge
- Functional(IP) blocks, typically implemented as independent ASIC cores, such as digital down-conversion(DDC) block, Matched Filter block, Rake Processing block, Max ratio combination block, Tracking block(involves loop-filter, direct digital synthesizer, estimation and correction, error data generation functions, etc.), Forward-error correction(involves Viterbi decoder, de-interleaver) and CRC block, MAC block etc.

It is evident that such architecture is pretty heterogeneous in nature. Also, each IP block itself has multiple modules. For example, the DDC block involves a numerically-controlled oscillator, root-raised cosine function filter and decimation function. If one adds the gate-count of such a system-on-chip, it can be classified as a **multi-million** gate design. The scale compounding with the heterogeneity of the system in question, makes the whole design process complex.

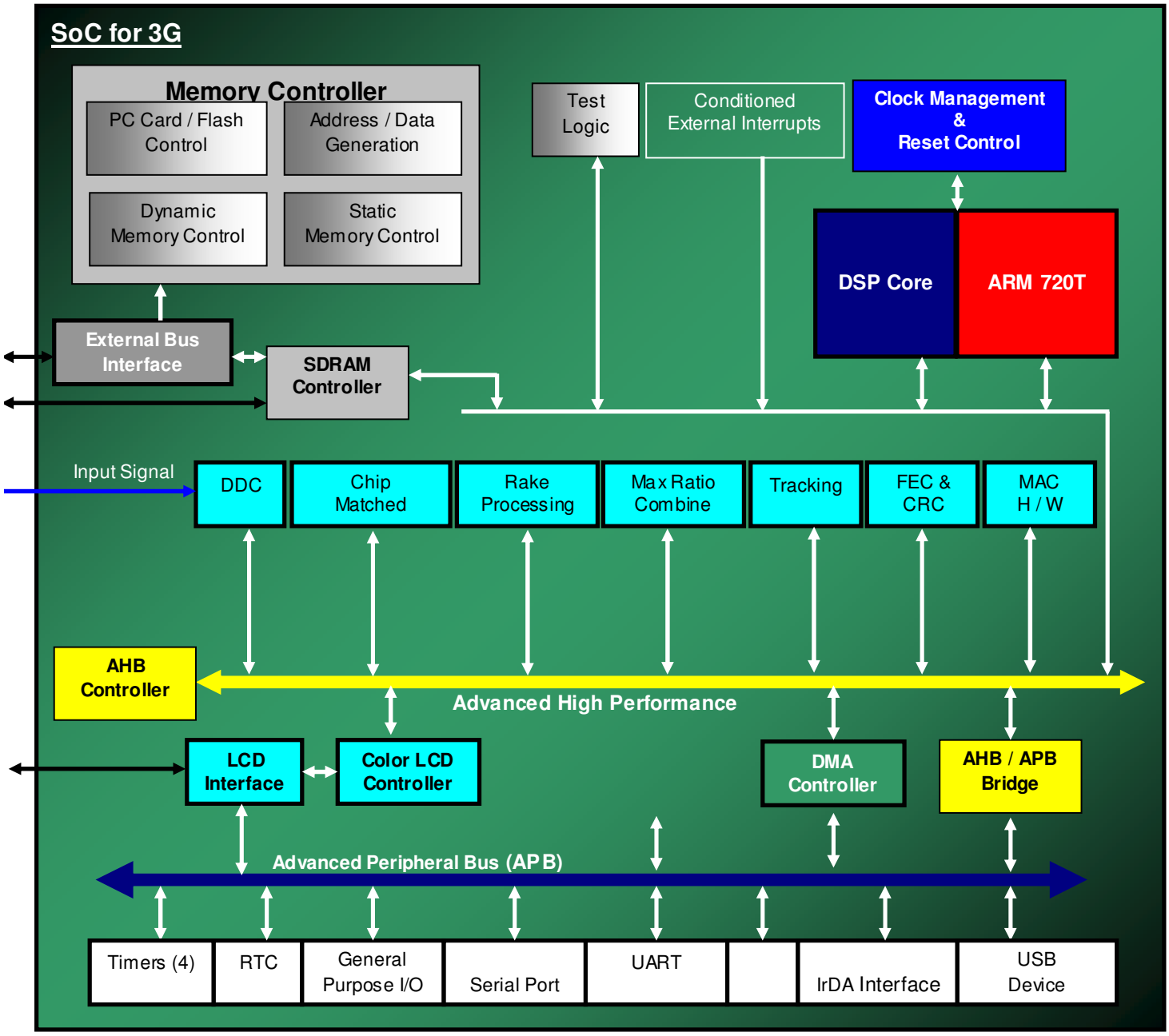


Figure 3: Complex Example

4 Cutting the In-house VLSI Design Process Flow

Centre of Excellence in Embedded Systems, TCS, Bangalore has a well-established, **very general** VLSI Design Process flow. It is described at length in [1]. In each general stage, there are potential points, where piping is present, and where two parties can enter into work-sharing. Such a piping can be evaluated for potential complementary-time scheduling. To locate such cut-points, the process flow diagram in section 2.2 of [1] has been redrawn using appropriate graphical notation, for each stage. From the set of these points, we detail out the operationalization aspects for them.

In business value chain, there can be multiple relationships between various parties. For our example, we take the case where two (VLSI-design) divisions of an organization, which are having time-complementary working hours, enter into such relationship. As pointed out in section 2.6, we have assumed that both parties are equally competent. For business reasons(such as to shorten the time-to-market), they enter into work co-ordination. Also one should note that the case can be generalized into more than two parties entering into work-coordination.

Before moving forward, it will be useful to define the phrase 'operationalization of flow'.

4.1 Operationalization of Flow

In the previous chapter, sample cut-points, and the information-flow across them was identified. Each instance of such flow requires identification of the medium of flow (such as email, shared database, …), the format, the periodicity, to name a few. Specifically, following elements are needed to make the distributed process operational:

- Format of information
- If the information is a set of items(such as a set of documents) to be exchanged during one handshake, then enumeration of the set
- Process for interactions. This list of options includes
    - Email
    - Web-based Tools
    - Distributed Configuration Management
    - Video-conferences
    - Tele-conferences
    - Exchange using magnetic mediums such as disks and tapes

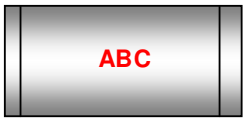
  For each such option, a detailed process, or a protocol and corresponding checklist can be evolved.
- Infrastructure setup. Few of the things to be covered here include
    - Server setup on either side
    - Choice of client computers to expedite activities such as simulation-runs
    - Installation of various tools

- Configuration Management
- Security-related setup

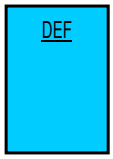
- Billing process.

4.2 Flowchart Legend

Most of the symbols used are self-explanatory. However, following symbols(blocks and arrows) convey special meaning.



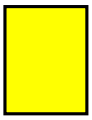
A module of activity; contains a set of activities, when enumerated.



Artefact provided by second party

3^{+}

Means that 3 or more artefacts are related at this end of the arrow.

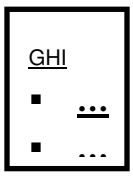


Artefact provided by first party

Feedback-flow(reverse direction); for example, review-based updating.

Partially parallel activity(PPA): implies that it is though parallel to some source activity, it **cannot start** before the activity at the other end of arrow is complete.

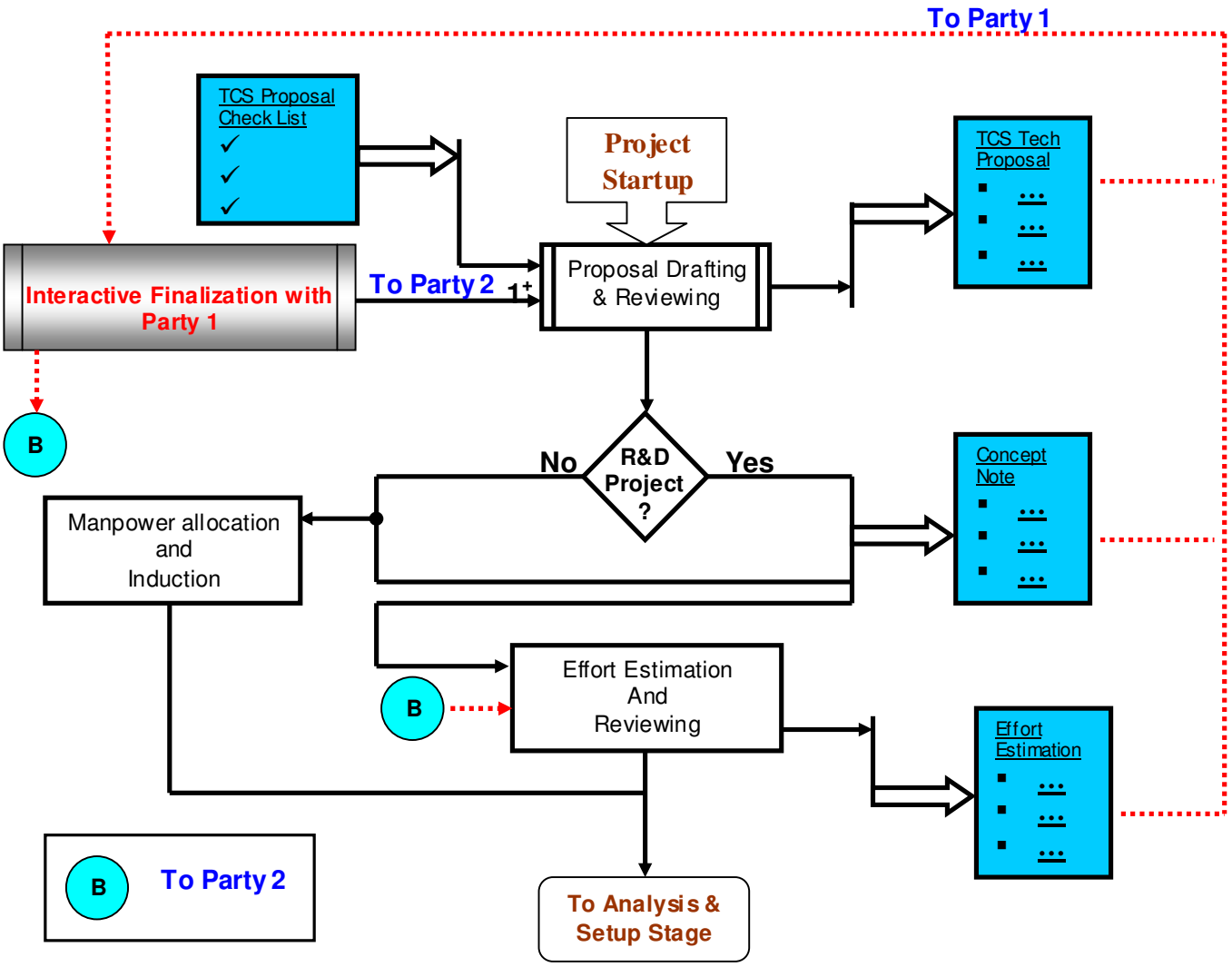
The ticked-list implies a process checklist.

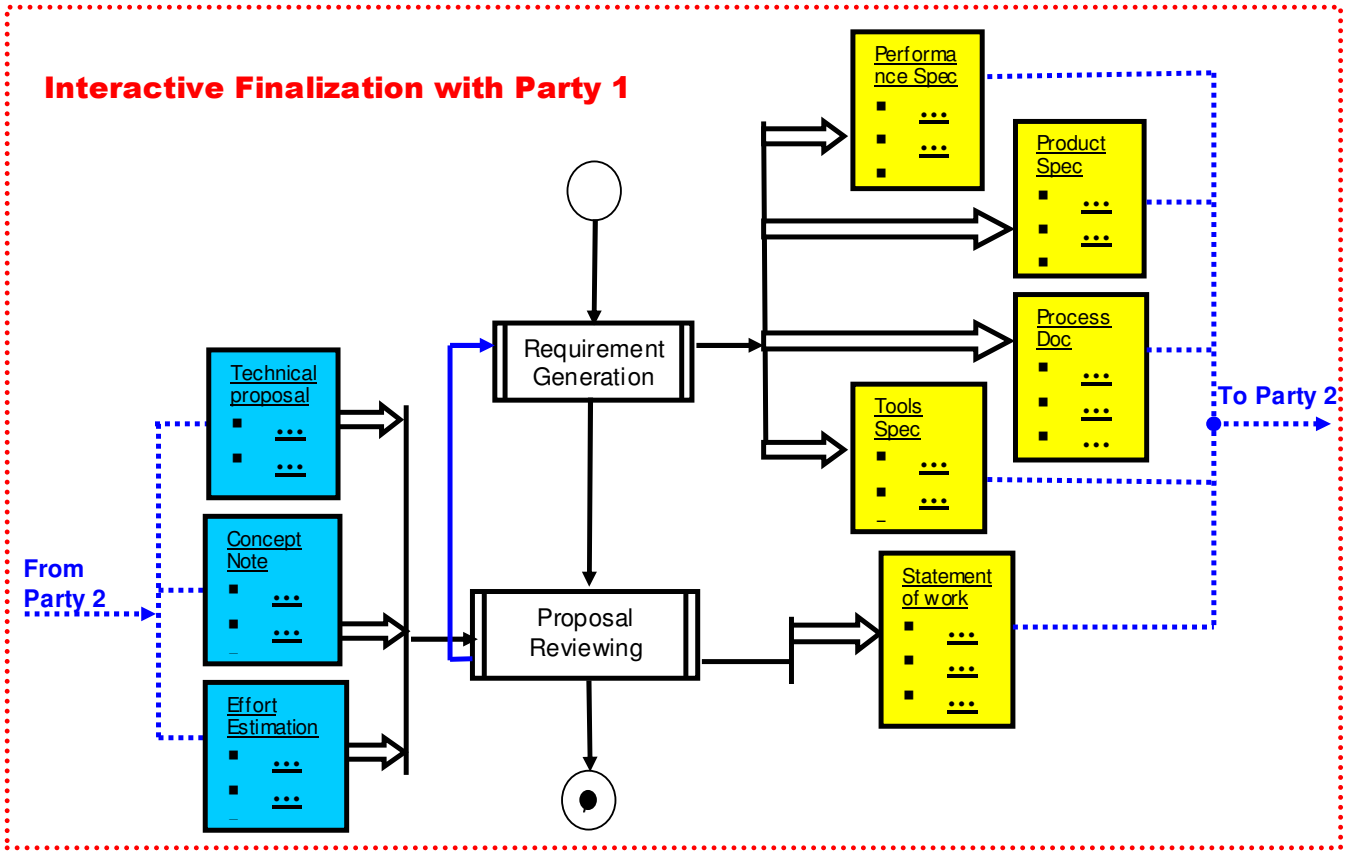


Implies a document; typically input or output of certain activity.

4.3 Stage 1: Proposal Drafting Stage

4.3.1 General Process Flow





4.3.2 Process Explanation

Because of the consumer-provider relationship, the role of one party is to specify what all needs to be done (overall work specification), while the role of other parties is to propose the feasibility of parts of this work, from their behalf.

In this regard, the first party is responsible for working out the overall project requirements. This leads to provision of product specification, performance specification (for the product), tools specification (to be used during design process), process specification (special choice of processes proposed to be used during the course of the project), to name a few. The process specifications further include choice of design methodology, which involves specification of tools and verification[2]/synthesis strategies, etc. Based on these documents, the other party (parties) comes up with a detailed proposal from their behalf.

Time spent in searching for the information can take a sizable amount of design engineers' time on any given project. At times, engineers are not able to find the necessary information needed to make intelligent and efficient architecture. Also there are times, when new IP blocks need to be developed. Both of these jobs involve R&D work. Hence, the other party can optionally also provide a set of concept notes. Since these jobs involve R&D work, it can lead to an independent sub-flow of activities, as explained in section 4.3.5.1. The other party also provide effort and people

[2] Functional or real-time verification

estimates, for sharing.

This process is iterative; all the parties review the corresponding relevant proposals, and suggest modifications to them. By doing this, they come to a mutual point(s) of agreement. At such point(s), the consumer party issues a *statement of work* to the provider parties.

4.3.3 Example Process Flow

For the example of 3G receiver design, first party will provide summary of core functions of the product, and also feature requirements. For example, the usage of DDC for down-conversion is a feature requirement. The performance specification can include throughput, chip rate, bit-error rate requirements etc. Further, the usage of certain tools such as Synopys DC Primetime for timing analysis, Cadence NCSim for simulations, etc. may be mandated by some party due to pragmatic considerations. Also, the choice of various compatible processes to be followed by the involved parties (such as quality assurance/audit process) are settled during proposal stage. Based on this data, other parties can *propose* component provisions, especially for IP blocks, such as MAC Hardware block. The proposal of doing an IP block can lead to detailed work/sub-flow, which can end as late as after construction of a fully-verified IP core. This is an example of *concurrent* sub-flow, which involves many 12-hr units.

After all these proposals are reviewed and agreed, statement of work for various portions of 3G receiver design project are issued to the concerned parties.

4.3.4 Locating Cut-points

During the "proposal" stage, there are times when the second party[3] comes across new components. Also, at times, one has to work on coming up with a better algorithm or architecture for a component, so that it can meet more stringent performance requirements. All these call for R&D, which can vary from small-scale to large-scale. The first step in doing so is the preparation of concept notes. Though not a one-day job for anyone, it also does not hold up any *intermediate* activity during the proposal stage. Hence, it can be classified as a *concurrent activity*, and can be done by (scheduled for) the other party. Depending on the competencies of the other party, domain of this concurrent activity can be enlarged into an R&D sub-flow for the component. Such experiments done in recent past have got success world-wide. For the *proposal stage,* we propose this as an initial cut-point.

An example of this could be a request by first party about details on Rake Receiver structure, DDC algorithms and conceptual architecture for the same. The first party may also require details of the hard DSP macros, which meet the functional and performance specifications.

In the proposal stage, there can be scheduling of *alternate activities* as well. For example, while deciding the performance requirements for a DDC block in 3G receiver, one party can work for one or two 12-hr units, and provide a nice summary document. This will save time for the other party, which is drafting the overall specifications. The exact number and nature of such cut-points for alternating activities depends on the exact product being pursued.

[3] As pointed out earlier, the second party holds the responsibility of drafting the proposal.

4.3.5 Flow across Cut-points

4.3.5.1 Component R&D

The initial cut for the concurrent activity is made, once it is decided that R&D for some component has to be done by the other party. Further cuts are made here (in a sequence), as the preparation activity is detailed. The flow involved here, if restricted to just developing concept notes, involves the following sequence:

I. Providing I/O specifications, or required function using suitable representation by one party. For example, it will provide the information on Rake Receiver, DDC Architecture and DSP macro details that covers parameters such as the device family/device name, CPU frequency, bus-widths, instruction widths, operating voltages, power-down modes, arithmetic support details, caching, memory, memory controller, MMU, timer details, i/o types, interrupt architecture etc.

II. Alternatively, providing newer performance specification for a component, whose basic function is known, by one party.

III. Optionally, few technical articles are suggested, such as papers and books, by one party.

IV. A prepared concept notes document is provided based on these, by the other party.

V. Optionally, few technical articles are provided, which are references to the proposed concept, by the other party.

Thus, one can identify at least three cuts, and corresponding flows in a sequence (I or II and III, IV and V) here. There can be times, when activity such as (III) require more than one 12-hr unit. In that case, because both the teams are equally competent, the *remaining* work can be carried over by the other party. Thus, it is possible to foster alternating the work, within a largely concurrent activity.

4.3.5.2 Sundry Alternate Activities

A large chunk of such activities fall in the category of doing first-level analysis and summarizing some study. Here the involved parties share technical articles/papers, (book) references and summary, etc. An example could be providing competitive (new) performance specifications for a component, whose basic function is known by one party. The second-level analysis of this summary involves reading this summary, and making decision with respect to overall project requirements. *Generally*, the flow involved here involves the following in sequence:

I. A detailed request of what information is required, from one party

II. A completed/partial summary of information, by the other party. If the summary is not complete, the first party can continue working on that during its next shift. Thus, one can observe that the alternating activities can cascade.

4.3.6 Operationalization of flow across Cut-points

4.3.6.1 Operationalization of Component R&D

The sequential order of flow has already been explained in previous section. Few other details about the protocol for interfacing here are as following.

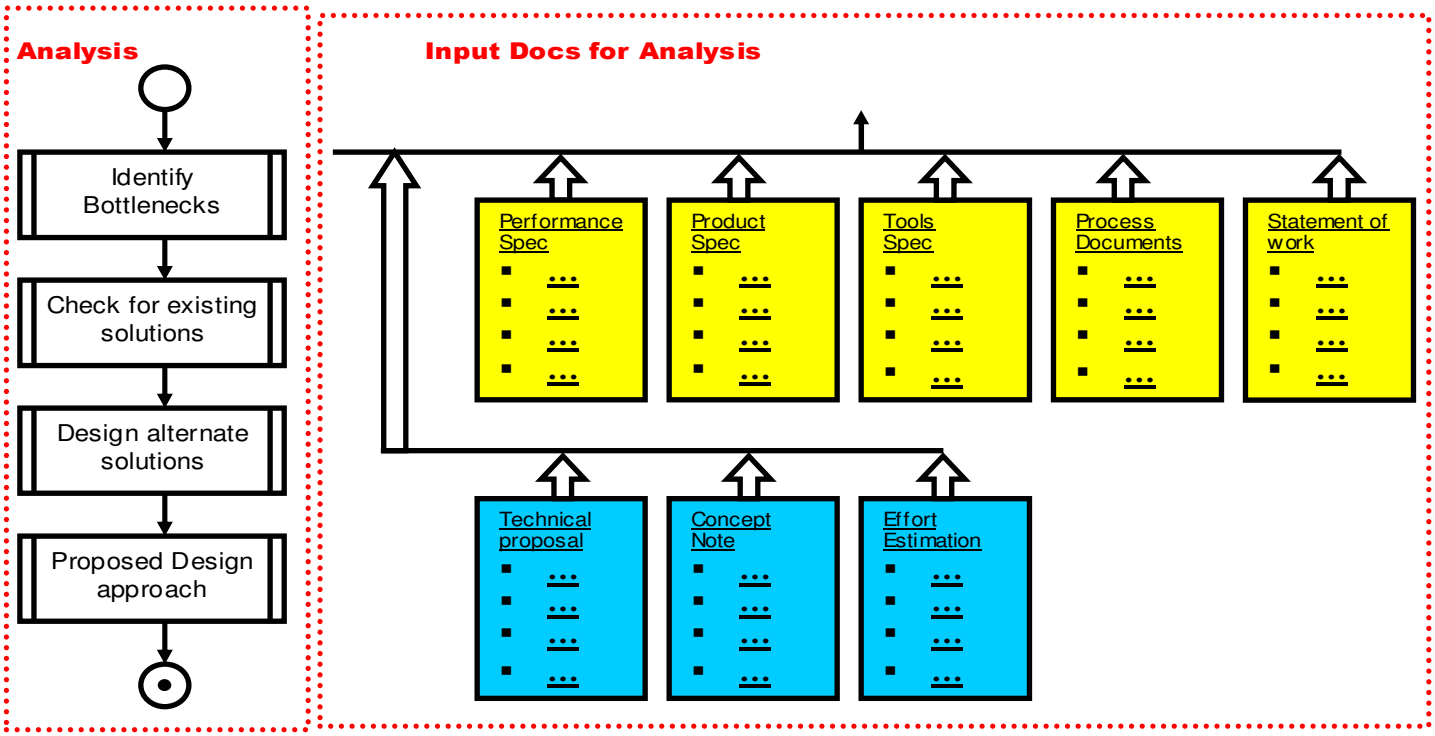
- Whenever one party provides a set of documents, the other party reviews it, and provides feedback on that.
- Various versions of all these documents are managed in a shared repository (database). Usage of such common medium allows both the parties to get the documents in time.
- Preparation for meetings and presentations need to be done, for real-time review, or managerial decision-making process.
- Security aspects such as confidentiality of proposal (NDA) need to be worked out.

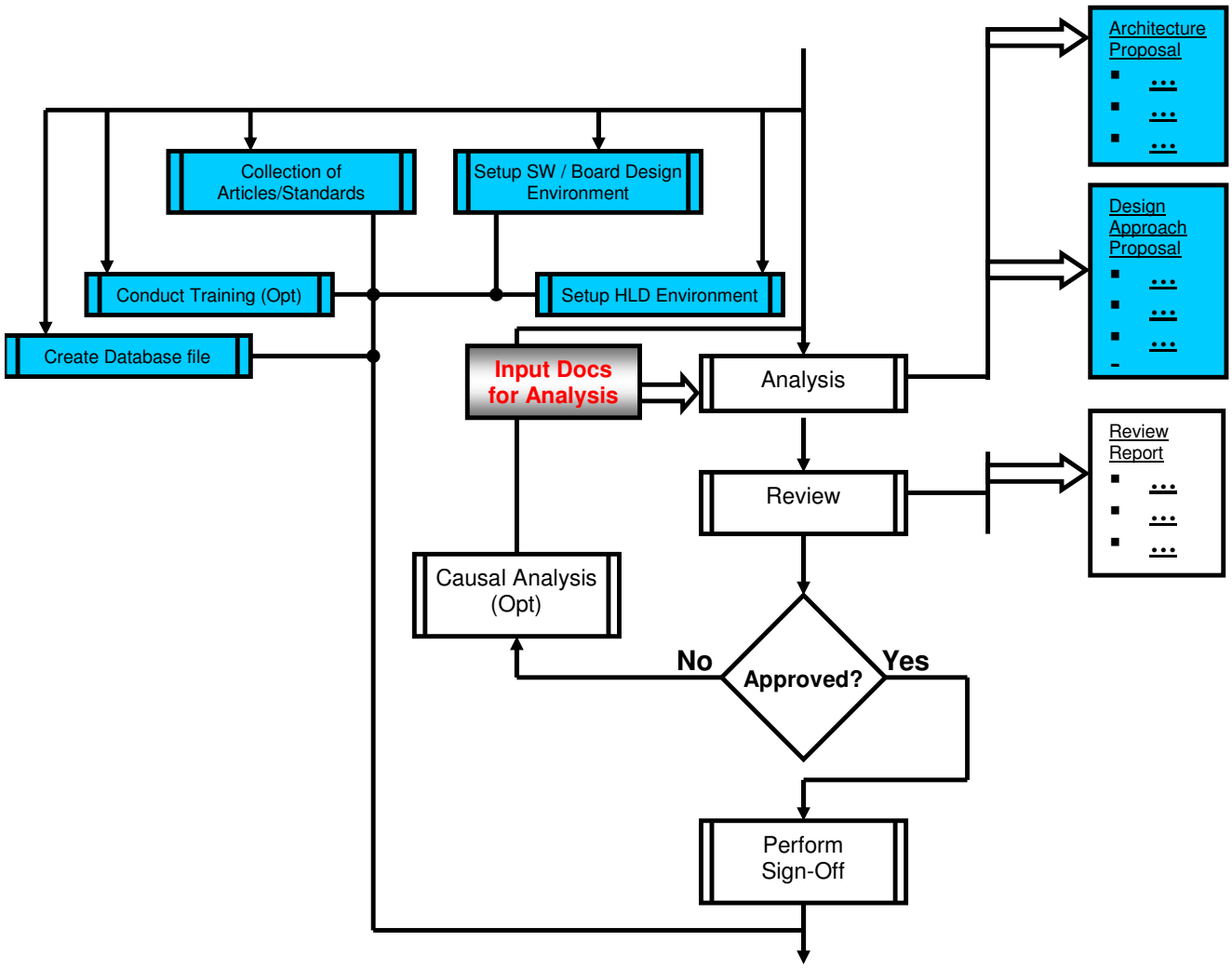
4.3.6.2 Operationalization of Sundry Alternate Activities

- The detailed request can be sent by either an email, or shared by a web tool.
- The summary can be updated best by putting it in repository, and sending the status by email, or via a web tool. The summary documents are the *support documents* for some document which can be classified as an *configuration item*, and hence tying the support documents together to mark a particular version of the primary document is very helpful.

4.4 Stage 2: Analysis and Setup Stage

4.4.1 General Process Flow





4.4.2 Process Explanation

This is the bridging stage which allows us to move further through the design flow. Primary work in this stage involves defining some amount of architecture of the product, and remaining part of design methodology. Given the **final** functional and performance specifications of the product, and also the scope of work as decided by the *Statement of Work*, the individual parties analyze the documents, and propose a first-cut architecture. There can be architectural components, which can just be off-the-shelf. Or there can be those, for which expertise is there, and they just need to be constructed. There can also be those components, which are absolutely new, and for which concept notes have been provided in previous stage. All such considerations also lead to evolution of the proposed design approach hereafter, which get documented in the *design approach proposal document*.

The other aspect of work in this stage is related to setting up the required environment. This includes:

- Collecting the relevant standards for subsequent high-level design activity

- Conducting training for the new staff of project, as well as to cater to the additional training requirement of the existing project staff
- Procurement and installation of required software (licenses) for the project(such as tool licenses), as well as any lab/hardware/board setup
- Planning the overall configuration management for the project, and creation of the repository/database

4.4.3 Example Process Flow

For the 3G receiver example, both the parties will analyze and identify the design bottlenecks in deciding the architecture of Rake Receiver, DDC; or in deciding the filter algorithms. For these bottlenecks, the available resources will be checked, and sometimes alternate design methods and approach will be considered. The top-level block architecture for the receiver will be done in this stage. Specific trainings on 3G technology, especially baseband processing, might be imparted. Rest of the activities are self-explanatory.

4.4.4 Locating Cut-points

As pointed out in section 4.4.2, different projects necessitate learning of different, new types of technologies and tools. Hence, during the "analysis and setup" stage, an activity for training is invariably present. Most of the while, one party has the required know-how of these things. If all of them put together do not have most of the knowledge, then it hardly makes sense for them to choose one-another as strategic partners for collaboration. Also there are times, when new staff is drafted in for a particular project, and they have needs to learn things, which they are lacking with respect to what the rest of their team knows. In either case, given its strength in particular areas, a particular team can always impart training to rest of the teams. Again, performing this activity in a time-complementary manner does not hold up any *intermediate* activities in the "analysis and setup" stage. Hence, it can be classified as a *concurrent activity*, and can be done by(scheduled for) the other party. For the *"analysis/setup" stage,* we propose this as an initial cut-point.

For rest of the activities, there can be scheduling of *alternate activities*. Since it is assumed that the architecture and design approach proposals are being primarily worked out by the first party, hence the second party can play a sound role in reviewing these, at various points, in one or two 12-hr units. This provides another cut point. Doing remote software setup and system administration provides another cut point: it is quite prevalent practice now-a-days. Finally, doing remote configuration management setup, or even drafting parts of the configuration management plan provides the final two cut-points.

4.4.5 Flow across Cut-points

4.4.5.1 Training

The flow here involves the following in sequence:

I. Optionally, certain preparatory material provided by one or more parties

II. Optionally, various handouts or home assignments distributed by the trainer party

III. A set of video-conference based trainings by trainer party

IV. Alternatively, a set of training-room discourses by trainer party

V. Previous point is accompanied by a travel task for the trainer party(a contributor to the effort involved in the synchronization activity)

VI. Optionally, Set of post-assignments by the trainer party

VII. Optionally, training feedback for improvement of further trainings to the trainer party

4.4.5.2 Review Activities

The flow here involves the following in sequence:

I. Provision of the artefact under review(such as a flowchart) by one party

II. Provision of the review comments by the other party

III. Optionally, provision of suggested changes in the artefact under review

4.4.5.3 Configuration Management Planning

The items here involve pieces of plan document, which differ based on what aspect of planning is being done. There is hardly any analysis involved; mostly it requires proposing strategy for various configuration management operations. It can also involve identifying branching needs, configuration and non-configuration items, to name a few. The flow in general involves the following in sequence:

I. An intimation of what part of the plan has to be proposed by one party

II. A proposal, or partial summary

4.4.5.4 Environment Setup

Only for the setup of softwares and tools, remote access methods can be used. Typically, it involves development of certain scripts as well. Also, once the configuration management plan is finalized, doing the required setup can also be expedited. The flow in general involves the following in sequence:

I. Optionally, development and subsequent provision of scripts by one party

II. Provision of system administration services by one party. The concrete output here is the part of environment, which has been set up.

4.4.6 Operationalization of flow across Cut-points

4.4.6.1 Training

- Whether preparatory material is made accessible via some website, or via shared file set.
- Deciding the timeframe for doing the assignments
- Doing setup as well as inviting people for the video-conference, or the training-room discourse.

- Finalizing the travel plans for the trainer's troupe
- Drafting and checking the post-assignment. Also providing the checked assignments back in stipulated time.

4.4.6.2 Review Activities

- The review notification method. Typically done by using some tool; at times, done informally by using email.
- The format/framework in which review comments will be logged.
- Whether any review meeting is required. In such a case, a teleconference might be required. In the worst case, where discussion has to happen over some major changes proposed, video conferencing might be required. If that does not work out, then travel plans for the person need to be fixed.

4.4.6.3 Configuration Management Planning

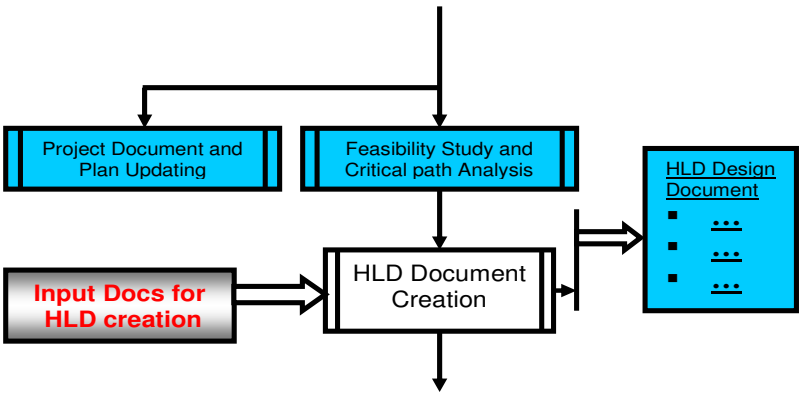
- Whether internal meeting needs to be called to brainstorm over policy design matters
- Evaluation of potential vendors, or follow-up
- Acquiring the details of computers and network at various places, e.g. hardware profile, OS loaded, network file system used, etc.

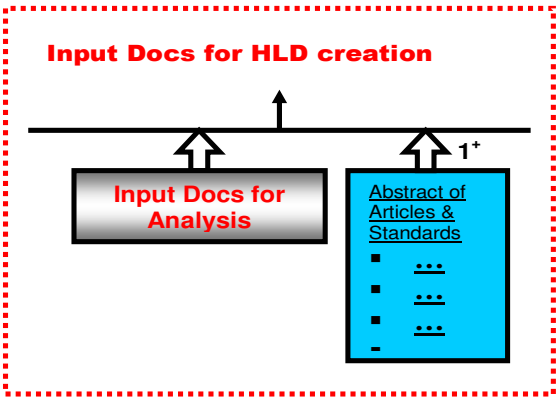
4.4.6.4 Environment Setup

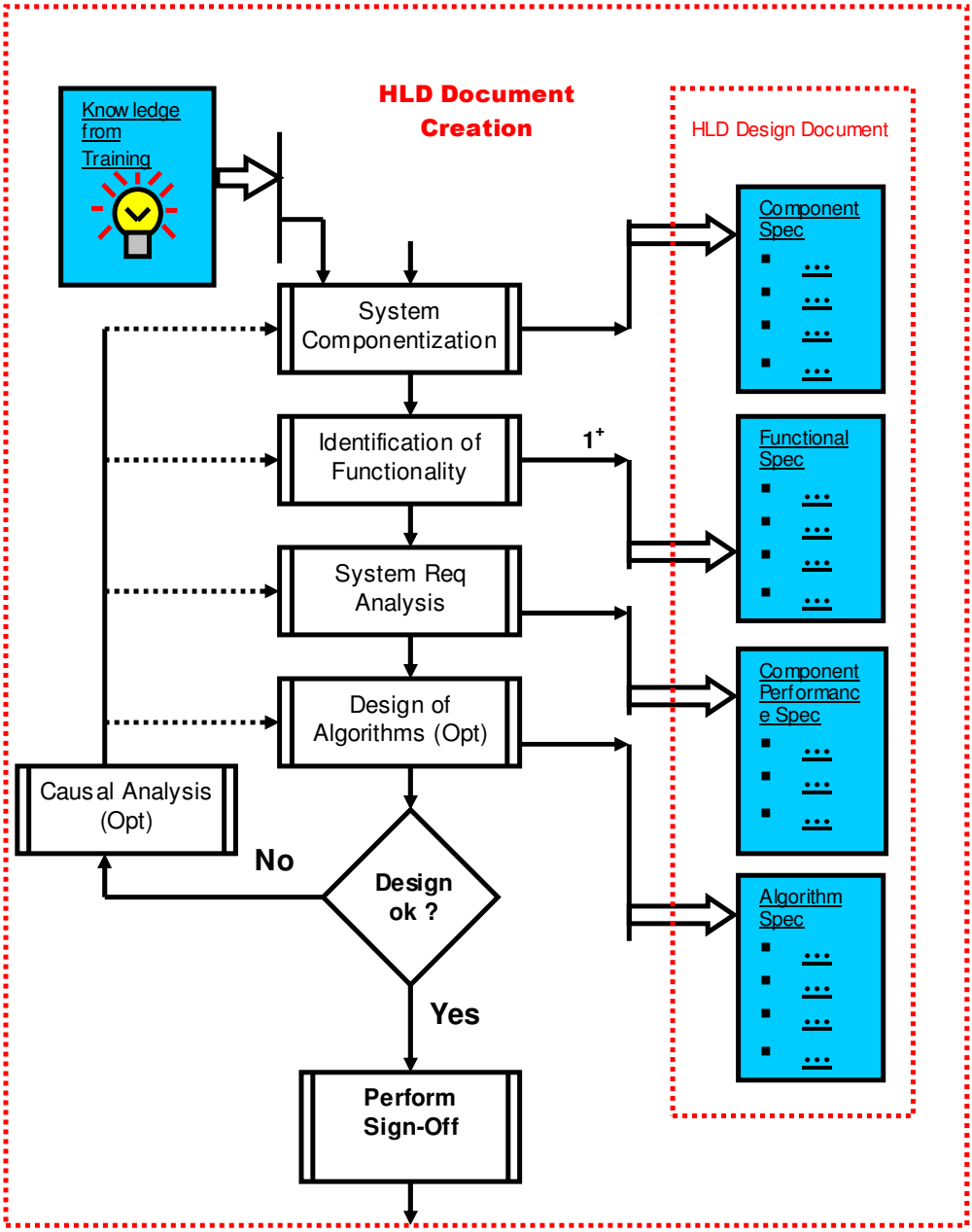
- Deciding the right scripting language to develop scripts
- Checking the compatibility of various computers for the repository setup
- Deciding the preferred time to do the system administration job. Typically, these are scheduled so that user(s) are not performing any important work.
- Deciding the nature of licences used, and whether they are shared or node-locked. If shared, then the policy of its usage.

4.5 Stage 3: High-level Design Stage

4.5.1 General Process Flow







4.5.2 Process Explanation

The stage starts with studying the feasibility of the requirements. There can be few architectural decisions, which are made in the previous stage. Otherwise, the detailed architecture is worked out in this stage. If some parts of requirements are considered challenging for the design purposes, they are highlighted and flagged to the management. The overall architecture design involves defining system components and their interfaces, their functional specification and parameter of operation (performance parameters such as pipeline width, memory type), prototyping to establish feasibility etc. Development for new algorithms for some logic defined in concept notes also happens in this stage. Optionally, mathematical and the Pspice simulations are performed in this stage. A set of specification for components, and a high level document (sometimes split into multiple files for maintainability) are the outputs of this process.

4.5.3 Example Process Flow

For the 3G receiver example, the top-level architecture is constructed, and feasibility study is carried out for the complex blocks. The mathematical and optionally, the Pspice simulation is performed in this stage for some of these blocks.

4.5.4 Locating Cut-points

Rapid prototyping to study feasibility is often carried out in this stage. Depending on the complexity, either the prototyping work, of evaluation of the prototype, or both, can be done in a small number of 12-hr units. This provides another cut-point, for an *alternate activity*.

Further, during the "high-level design" stage, many-a-times new algorithms have to be designed. Again, depending of the expertise of one party, the algorithm design can be assigned to it. This provides the initial cut-point, for a *concurrent activity*.

Also, especially for developing ASICs, one needs to do detailed comparison and decision-making about which macros to use for *individual functions*, whether to purchase or to develop these macros, etc. The required information for each such function is mostly collected in the 'proposal' stage. This work is a major part of HLD stage, and hence can be scheduled for a cut-point of a *concurrent activity*.

Similarly, to meet timing performance, critical path analysis of certain models needs to be carried out. A cut-point can be located here; it can be scheduled for an *alternate activity*.

Finally, assuming that the high-level design is being primarily worked out by the first party, the second party can again play a sound role in reviewing these, at various points, in one or two 12-hr units. Thus, this can be deemed and scheduled for an *alternate activity*. Since the details for this activity is the same as in previous stage, we will omit it henceforth from all the stages. Thus, we conclude that in all stages, review activities in general can be scheduled under *alternate activities*.

4.5.5 Flow across Cut-points

4.5.5.1 Rapid Prototyping

The flow here involves the following in sequence:

I. Providing the objectives for the prototype from one party

II. Provision of the developed prototype from the other party

4.5.5.2 Deciding Hard-macros

The flow here involves the following in sequence:

I. Provision of vendor contacts from one party

II. Detailed performance/feature chart for specific macros from other party

III. Optionally, decisions pertaining to specific macros by first party

IV. Optionally, a partial structural design for a particular functional block from other party

4.5.5.3 Critical Path Analysis

The flow here involves the following in sequence:

I. Provision of circuit model by one party

II. Critical path details from the other party

4.5.5.4 Prototype Evaluation

The flow here involves the following in sequence:

I. Providing detailed report on the output of prototype running by the other party. Critical path analysis report for the prototype can a *specific* evaluation report.

II. Optionally, minor modifications done in prototype, which led to fulfilment of the prototype objectives

III. Optionally, demonstrating a stable prototype to obtain feedback

4.5.5.5 Algorithm Design

The flow here involves the following in sequence:

I. Providing i/o specifications, or required function using suitable representation from one party

II. (Additionally), Providing performance specifications for a component, from one party

III. Optionally, a mathematical model of the algorithm specification as output, from other party

IV. Optionally, a prototype implementation, such as MATLAB code for loop filter design, as output, from other party

V. Optionally, few guidelines for architecture design for that algorithm, from other party

4.5.6 Operationalization of flow across Cut-points

4.5.6.1 Rapid Prototyping

- To decide the demonstration objective of the prototype. It can be:
    - System architecture
    - External interface design
    - Functionality.
    - Re-usability
- Decision on the type of prototype, that is, whether it should be evolutionary, or throwaway.
- Decision on the software/hardware platforms chosen for developing the prototype. They can be

purchased from various vendors.

4.5.6.2 Prototype Evaluation

- Decision about format of reporting
- Whether evaluations such as critical path analysis is done by simulation or by hand
- Doing the required system setup for demonstration, if required

4.5.6.3 Deciding Hard-macros

- Completeness of information. For example, whether gate count information has been included, or not.
- Feasibility of purchase
- Feasibility of in-house development, if required
- Criticality of particular macro from particular vendor
- Amount of details along with the partial structural design, such as functional and hardware requirements of each block

4.5.6.4 Critical Path Analysis

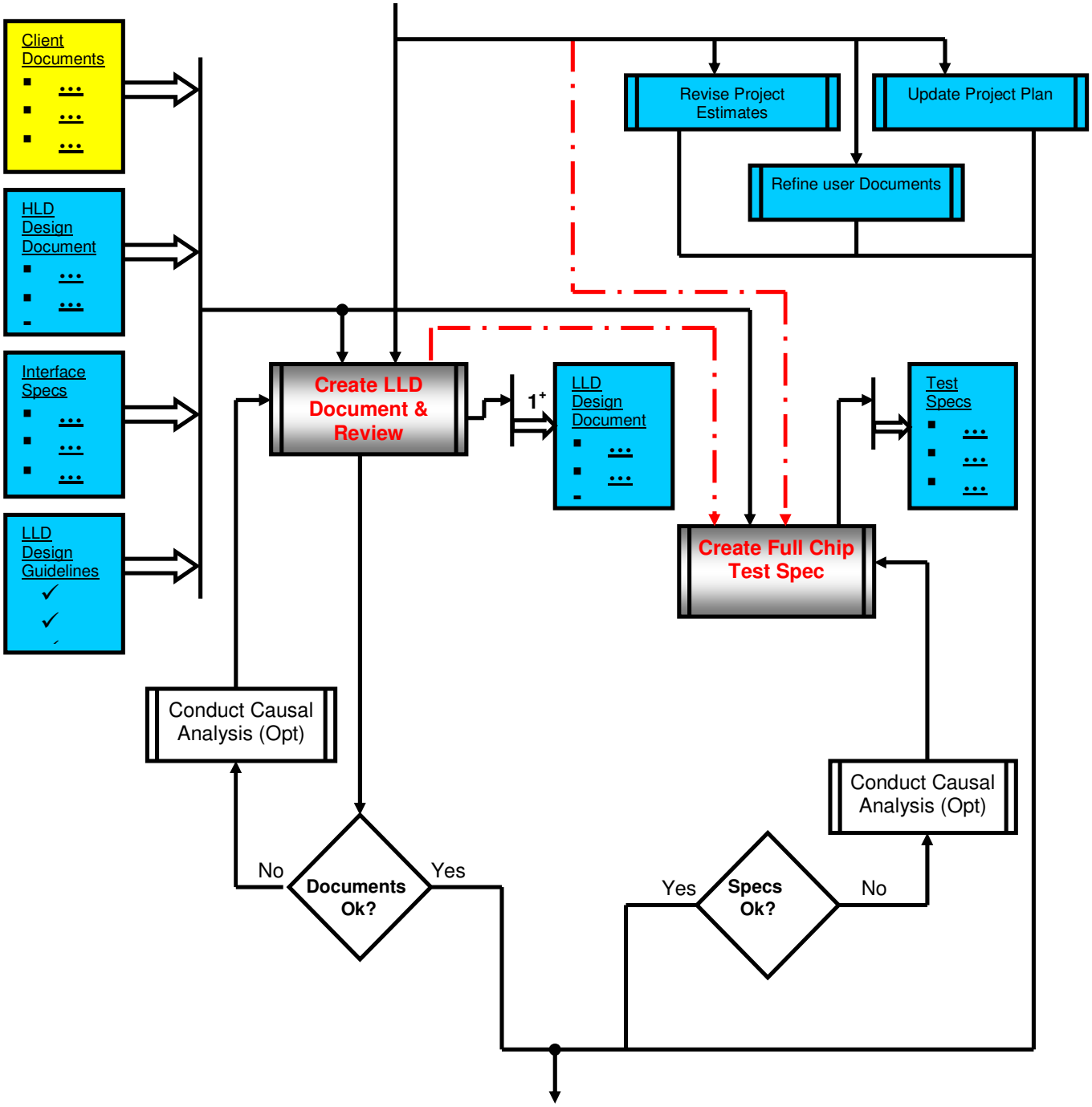
- Decision about whether the analysis is done by simulation or by hand
- Whether marginal timing optimization need to be done using the tool, along with analysis

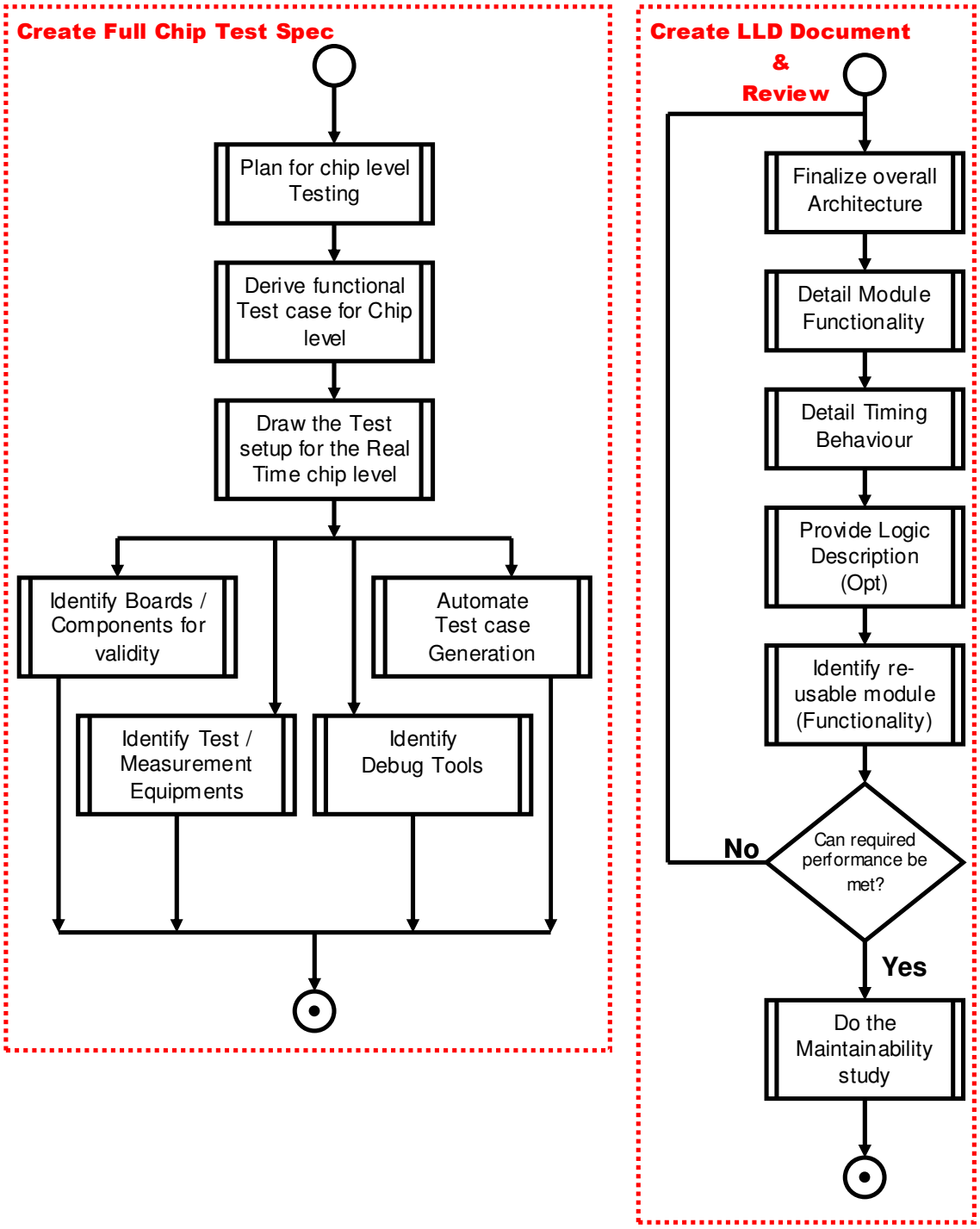
4.5.6.5 Algorithm Design

- Whether design alternates need to be documented as well.
- Whether simulations results be provided as proof of choice. If provided, then it can be exhaustive: a lot of parameters can be tuned, and performance observed. For example, for adaptive algorithms, different set of filter weights, heuristically decided, lead to different real-time convergence performance.
- Limitations of the algorithm, if any, to be provided.

4.6 Stage 4: Low-level Design Stage

4.6.1 General Process Flow





4.6.2 Process Explanation

This stage involves mainly two activities: providing the last level details of the proposed design, and drafting the test strategy for the overall system. The last level details for the proposed design include the detailed functionality, interface details such as timing waveforms for synchronous circuits, the initial shape of programs etc. The overall test strategy for system testing is best done in this stage; for the required details are obtained in a stable state, from previous stage. It not only

includes planning, but also setting up the framework for the detailed simulation, observation of signals, debugging the problem etc.

4.6.3 Example Process Flow

For the 3G receiver example, let us consider the DDC block. High-level design provides the internal module structure for the block, which includes NCO, Digital Mixer, RRC filter, decimation blocks etc., along with their functionality. In LLD stage, modules such as NCO are considered(no further decomposition), and the implementation details for them are specified. The required operators such as Phase Accumulator, Multiply and accumulate, required storages such as the Sin ROM table are included in this specification. Various parameters/data types, such as size of storage, address/data bus width etc., are also brought out.

As shown in the figure, a parallel team can work on the testing and verification strategy for the receiver.

4.6.4 Locating Cut-points

Assuming that the low-level design itself is carried out by one party, there is not much scope of work-sharing here. However, certain activities such as *detailing* the functionality of *multiple* medium or simple modules can be done in one or two 12-hr units, and hence can be scheduled under *alternate activity*. As an example, deciding the filter-width(number of taps), filter coefficients(weights) and filter architecture(implementation detail) for a *simple filter* can be done by the other party. These can be accompanied by providing logic description for modules, such as error detection block in our example. These cut-points are decided, on a case-by-case basis, by taking into consideration the team size, complexity of the project and expertise level of the parties. Also one can multiplex the same team at the same location for working on different modules at different times.

Further, given that test strategy can be mostly decided by looking at the high-level design, one party can always start off doing the test planning. Thus, in terms of a *concurrent* activity, this presents a very potential candidate for scheduling/assigning to other party. This provides the other set of cut-points for this stage.

4.6.5 Flow across Cut-points

4.6.5.1 Functional Detailing

The flow here involves the following in sequence:

I. Specification of module, for which detailing is to be done, by one party

II. The detailed module functionality, by other party

III. Optionally, logic description for the module by the other party

4.6.5.2 Overall Test Planning

The flow here involves the following in sequence:

I. Code coverage requirements provided by the first party

II. Provision of chip-level(integration-level) test plan by the other party.

III. Provision of real-time testing setup, by the other party

IV. Setup of environment for testing, including the required automation

V. Optionally, development of prototypes for virtual prototyping

4.6.6 Operationalization of flow across Cut-points

4.6.6.1 Functional Detailing

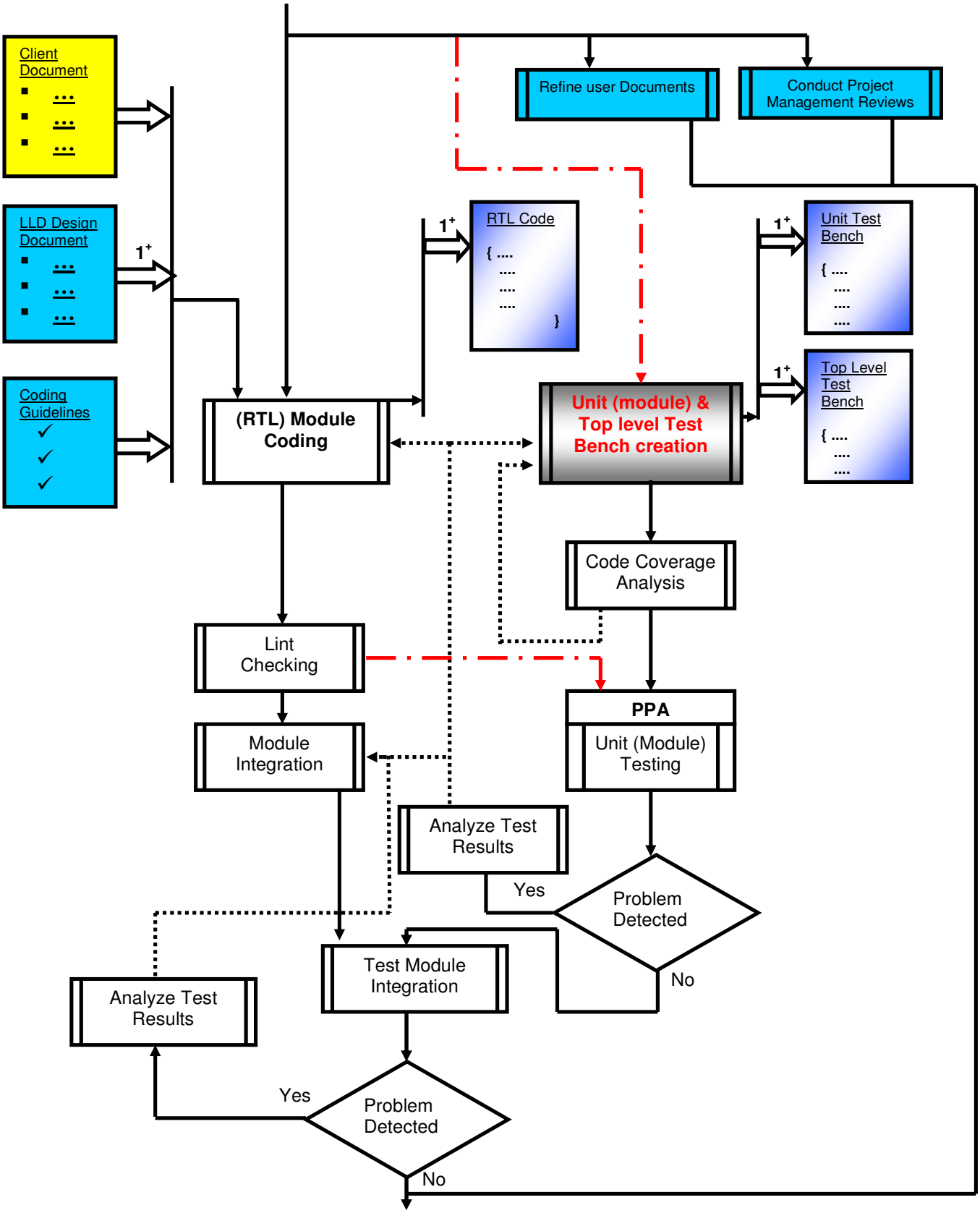
- The relevant standards, which need to be referred, need to be identified.
- The level of details provided (such as flow-chart, or detailed textual level).

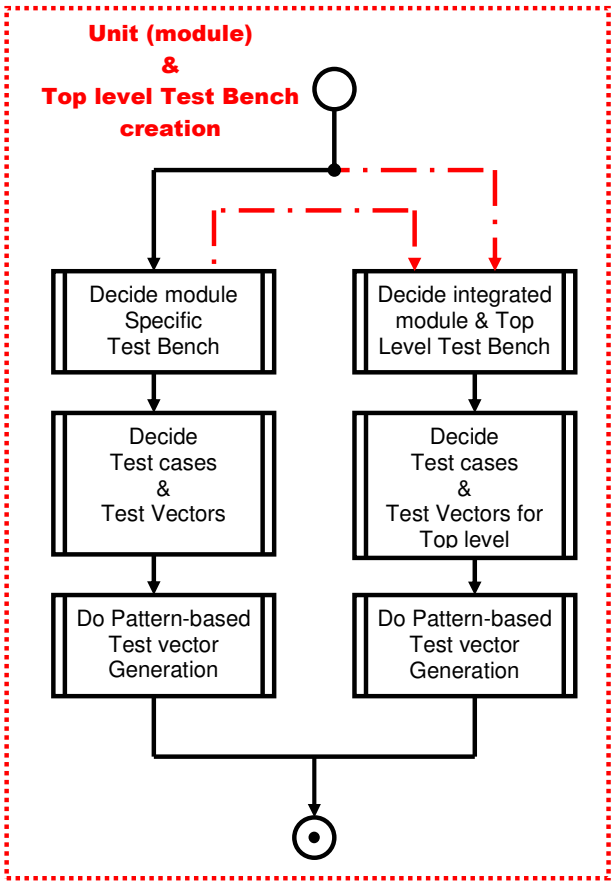
4.6.6.2 Overall Test Planning

- Identification of required configuration of simulation tools
- Working out of test bench generation strategy
- Interface development of simulation tool with some debug environment
- If FPGA is used for the prototyping, then the gate count required for the FPGA board
- Decision of vendor, from which FPGA emulation board and corresponding tools will be purchased.

4.7 Stage 5: Construction Stage

4.7.1 General Process Flow


Client Document
LLD Design Document
Coding Guidelines
Refine user Documents
Conduct Project Management Reviews
RTL Code
Unit Test Bench
Top Level Test Bench
(RTL) Module Coding
Unit (module) & Top level Test Bench creation
Code Coverage Analysis
Lint Checking
Module Integration
PPA
Unit (Module) Testing
Analyze Test Results
Problem Detected
Yes
No
Test Module Integration
Analyze Test Results
Problem Detected
Yes
No



4.7.2 Process Explanation

In this implementation stage, the project team's goal is to come out with the RTL description for the modules, the unit-level (module-level) test benches/test vectors and the top/chip-level test bench/test vectors'. This is done while keeping the LLD as the base. The tested modules are subsequently integrated, and further verification is carried out. At times, the full-chip verification is done using virtual prototypes targeted for FPGA boards.

Because these jobs are bit parallel, it makes sense for having one party focus on RTL construction, while other party focuses on test bench generation. This makes for the best coordinated scheduling. If the other party starts helping in say, RTL construction, then it will lose time, which it can devote while working independently on test benches.

4.7.3 Example Process Flow

For the 3G receiver example, one party takes LLD and coding guidelines for RTL as input, and start programming these modules. This can include, the DDC block, NCO block(its elements such as phase accumulator, Multiply and accumulate unit, Sin ROM table), Digital Mixer module(pair of digital multipliers), RRC filter (filter TAP and associated modules) and decimation modules.

At the same time, the other party starts working on the test bench for unit testing DDC block, and for integrating testing of baseband modules such as the DDC and MAC units.

4.7.4 Locating Cut-points

There are activities, such as lint-checking, which can be done in one or two 12-hr units, by other party. But the changes involved in the RTL code are best made by the programmer himself; hence a cut cannot be scheduled here.

Since the coding of various component modules will finish in an asynchronous manner, it calls for incremental unit-level and integration-level testing. Also, this is the point in the project, when the maximum number of iterations happen. Every time a problem is located, it has to be fixed, and things have to be re-tested. A particular fix can break down other behavioural aspects (which are observed during testing). Hence it makes the **most sense** to schedule them individually as *alternate activities*. There are activities such as test bench construction, which are independent of finishing of RTL code. Hence, the initial cut-point is for (simulation-based) verification *concurrent activity*. From here, alternate activities of actual simulation runs are embedded, thus providing further cut-points. The ideal situation here is having a full verification team complementing the module construction team. If the programming of certain modules finishes (is expected to finish) concurrently, then *multiple teams* can be assigned to carry out verification activity in the same 12-hr unit. For example, one team might test the error detection module, while the other tests the filter module.

Activities such as code coverage analysis yet again present a potential case of scheduling for *alternate activities*. But like the previous case, it is best carried out by the party, which is doing the verification work. Hence it cannot be scheduled in the different time zone of the other party.

Also, prototype development, if required for the project, can provide a cut point, because it can be scheduled at different location as *concurrent activity*.

Finally, module integration, though asynchronous, provides another cut point, because it can be scheduled at different location as *concurrent activity*.

4.7.5 Flow across Cut-points

4.7.5.1 Module Testing

The flow here involves the following in sequence:

I. Provision of a finished module(say, RTL code)

II. Optionally, provision of the required test-bench. It makes more sense to tie it with the verification team itself. The person doing verification is a better judge of what parts of behaviour, the test-bench covers for the module, and what parts not.

III. Bug Reports. This in turn involves

   a. Nature of problem

   b. Optionally, approximate location of the problem

IV. Optionally, certain output generated during tested, such as timing waveforms

V. Optionally, minor modifications done on-the-fly by the verifier, which have to be reviewed and propagated back in the module

4.7.5.2 Module Integration

As pointed in previous section, during the "construction" phase, coding of various component modules will finish in an asynchronous manner. Hence, it calls for incremental unit-level and integration-level testing.

The flow here involves the following in sequence:

I. Optionally, updated LLD documents. During the coding, there are times when changes are made to the interface (signal and port-level changes). If they are not reflected in the LLD documents, the integrator will connect the modules via a wrong net.

II. The (set of) modules to be integrated

III. The integrated modules as output

IV. Optionally, problems found during integration(such as unconnected port), if any

4.7.5.3 Testing with Prototype boards

The flow here involves the following in sequence:

I. Provision of a general-purpose prototype board by one party, for the evaluation of the modules by one party.

4.7.6 Operationalization of flow across Cut-points

4.7.6.1 Module Testing

- Whether the code has been lint-checked before sharing
- Format of information. The bug-report format can be found as part of the testing plan.
- The set of documents can involve
    - Bug report
    - Changed code, if any
    - Waveform diagrams
- Process for interaction can involve
    - Distributed Configuration Management to share completed modules
    - Web-based Tools such as Bugzilla to share testing outputs and bug reports
    - Optionally, tele-conferences to discuss and narrow down complex problems
    - Optionally, tele-conferences to report critical problems
- Infrastructure. For module testing, it can involve

- Creation/sharing of test-bench
- Sharing of standardized test-cases, if any
- Simulation tools
- Additional Scripts

- Database. Here, it can involve
  - Bug reports
  - Modules
  - Test outputs
  - Coverage reports
  - Scripts and Tools

- Security Aspects. Here, it can involve security of distributed configuration management. It requires configuring firewall, installing secure server, secure protocols such as sftp and https, using encryption during data exchange, and period change of encryption keys.
- Billing. Whether additional billing based on number of problems solved, needs to be done.

4.7.6.2 Module Integration

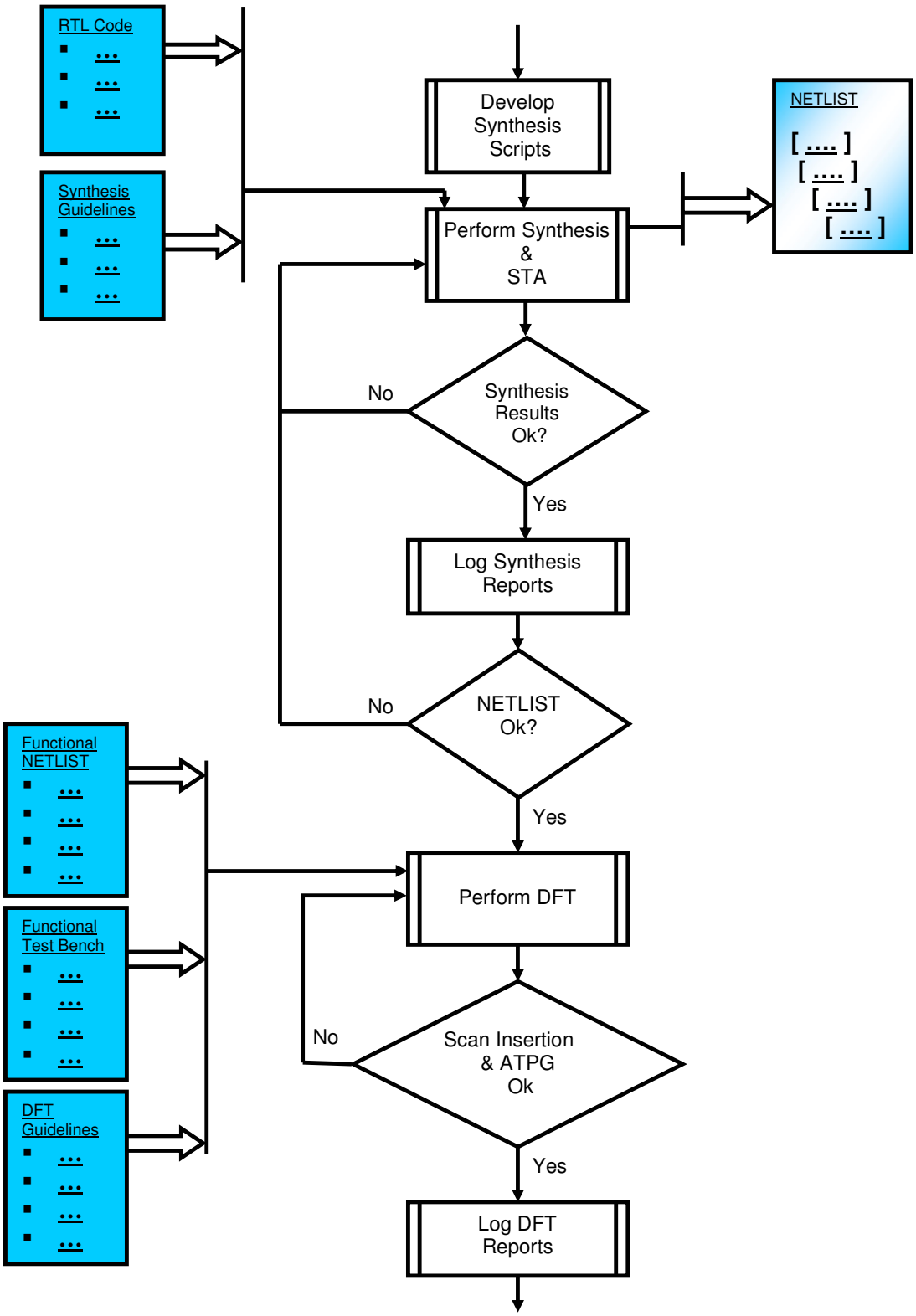
- To check whether synthesis has passed on the modules to be integrated
- Whether new figures for area etc. need to be provided

4.7.6.3 Testing with Prototype boards

- To decide which RTL modules to be chosen, whose functionality can be validated using these boards.

4.8 Stage 6: NETLIST Generation and STA Stage

4.8.1 General Process Flow


RTL Code
Synthesis Guidelines
Develop Synthesis Scripts
Perform Synthesis & STA
NETLIST
[....]
[....]
[....]
[....]
No
Synthesis Results Ok?
Yes
Log Synthesis Reports
No
NETLIST Ok?
Yes
Functional NETLIST
Functional Test Bench
DFT Guidelines
Perform DFT
No
Scan Insertion & ATPG Ok
Yes
Log DFT Reports

4.8.2 Process Explanation

Both Synthesis and Static Timing analysis feed on validated RTL modules. If synthesis fails, in the worst case, the architecture itself may need to be changed. Given this risk, the timing analysis and synthesis run can be done one-after-the-other only.

4.8.3 Example Process Flow

For the 3G receiver example, the functionally-qualified modules are taken for the block level synthesis along with the synthesis guidelines. For example, the DDC block is taken for the synthesis. If it meets the required performance like speed and gate count, then it is checked further using timing analysis. This is carried on for all the modules; and at last the top-block is synthesized.

4.8.4 Locating Cut-points

Synthesis is best carried out by the party, which has done the RTL programming. However, to make things faster, once first party tries to generate the netlist using synthesis, the other party can try to do timing simulations. Thus, a cut point can be located here; hence it can also be scheduled for *alternate activities*.

Given that there will many modules, and that their integration level adds to further rounds of synthesis and timing analysis, the number of actual cuts here depends on the exact product, and its modular structure. For example, in the 3G receiver case, cut can be done after synthesis(so that other team works on timing analysis) for DDC block, filter block, rake receiver block, tracking and error detection block etc.

4.8.5 Flow across Cut-points

4.8.5.1 Timing Analysis

The flow here involves the following in sequence:

I. Provision of netlist for individual modules(or some integrated block) by one party

II. Provision of overall synthesis report by one party

III. A post-map timing report to evaluate the effects of logic delays on timing constraints.

IV. A post-place-and-route timing report[4] that incorporates both block and routing delays as a final analysis of the design's timing constraints.

4.8.6 Operationalization of flow across Cut-points

4.8.6.1 Timing Analysis

- To check whether the synthesis has passed the customer-supplied goals, before it is passed on

[4] The Interactive Timing Analyzer tool produces detailed timing constraint, clock, and path analysis for post-map or post-place-and-route implementations.

by one party

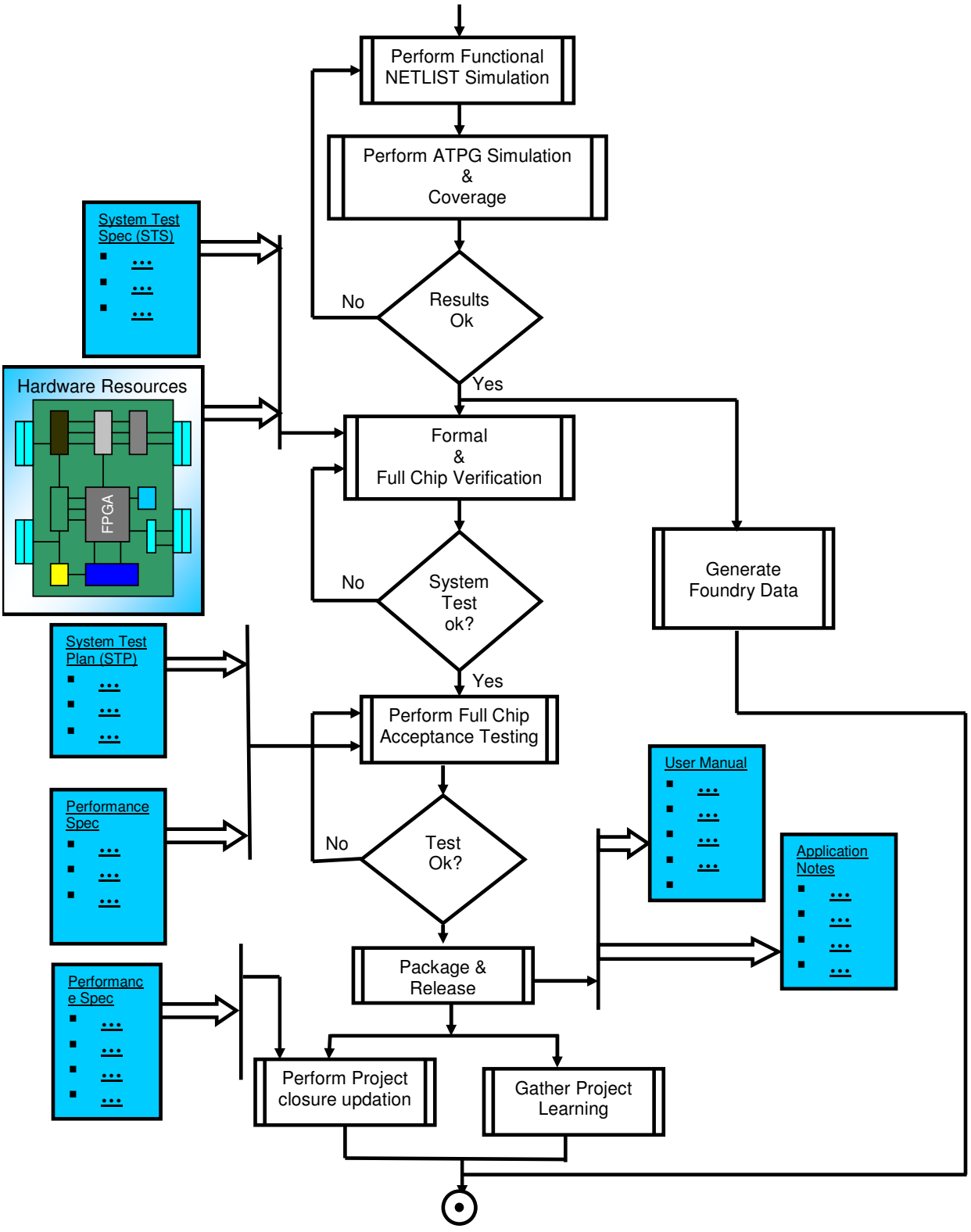
- To check whether synthesis has met local parameter optimization, global parameter optimization, optimization for parameters with topology, and full structural synthesis, before it is passed on

- Agreement on what type of hardware will be generated by special language constructs.

- If the synthesis fails(for example, creation of redundant, reconvergant logic structures, regardless of the designer's intent), then the modified RTL code for the module needs to be passed back to the other party(in previous stage) for verification purposes. Further, if some design change has happened, then LLD needs to be updated and also sent across for reference.

4.9 Stage 7: DFT and ATPG Simulation Stage

4.9.1 General Process Flow



4.9.2 Process Explanation

While developing or purchasing IP core, it is usually unclear, how well it conforms to the DFT rules. Even when designers strive to follow DFT rules for new functions, is sometimes impossible to comply, due to the design constraints. Hence, scan chain insertion needs to be done for these, before pattern generation and simulations are performed.

4.9.3 Example Process Flow

For the 3G receiver example, scan chain insertion and ATPG simulation can be done for various modules such as DDC.

4.9.4 Locating Cut-points

By looking carefully, cut-points can be identified for *alternate activities* of scan chain insertion, ATPG data generation, ATPG simulation and foundry data generation. Especially, the second party can do data generation and ATPG simulations for modules, for which first party has done the scan chain insertion activity. For example, in the 3G receiver case, cut can be done after scan-chain insertion for DDC block, filter block, rake receiver block, tracking and error detection block etc.

4.9.5 Flow across Cut-points

4.9.5.1 ATPG Simulation

The flow here involves the following in sequence:

I. Provision of scan-inserted netlist from one party

II. Provision of netlist, which has passed verification post ATPG, from other party

4.9.6 Operationalization of flow across Cut-points

4.9.6.1 ATPG Simulation

- Whether efficient ATPG algorithms have been proposed for combinational circuits, as well as for synchronous sequential circuits.

4.10 Stage 8: Verification & Floor-Plan Estimation Stage

4.10.1 General Process Flow

Flow chart

4.10.2 Process Explanation

Following the DFT guidelines enables designers to implement deterministic logic test capabilities transparently in their designs without impacting the functional, timing or power requirements of the design. Going further, doing formal verification enables us to do functional simulation and static timing analysis on the scan inserted NETLIST.

4.10.3 Example Process Flow

For the 3G receiver example, both the parties will split the blocks and do formal verification and STA for the scan inserted NETLIST at the block level(DDC block, Filter block, Rake Receiver block, Tracking and Error detection, etc.). Along with this activity, floor-planning is also carried out.

4.10.4 Locating Cut-points

Similar on lines of stage 6, cut-points can be identified for *alternate activities* of formal verification, static timing analysis and floor-planning activities. For each module, these activities happen in sequence; though many such sequences may operate concurrently. Hence, two cut-points per sequence can be identified. At a particular point of time, many cut-points may be operational; it depends on the availability of scan-inserted netlists.

4.10.5 Flow across Cut-points

4.10.5.1 STA

The flow here involves the following in sequence:

I. Provision of netlist from one party, which has passed formal verification

II. Provision of timing analysis report on this netlist by other party

4.10.5.2 Floor-planning

The flow here involves the following in sequence:

I. Provision of netlist by second party, which agrees to timing constraints

II. Provision of high-level floor planning report by the first party

4.10.6 Operationalization of flow across Cut-points

4.10.6.1 STA

- Availability of same persons for problem fixing due to functional or timing violation. The RTL programmers will have the right knowledge and control to do these changes.

4.10.6.2 Floor-planning

- Whether the floor-planning can meet information-generation requirements. High-level floor-planning generates very little physical information, such as number of I/O's, number and size of macro blocks, and some approximate sizes for the core blocks.

4.11 Stage 9: Detailed Floor Planning, Placement & Routing Stage

4.11.1 General Process Flow

Flow chart

4.11.2 Process Explanation

High-level floor-plans defines the relative locations of design blocks, global routing resources including clock domains, I/Os pin locations, power and ground distribution, resolve area and timing budgets, generate models for interconnect estimation, and extract boundary condition information. This activity naturally flows into detailed floor-planning. The estimates from previous stage allow the tools down the design flow, to work with more accurate and reasonable information instead of "guesswork," which can lead to multiple back-end iterations, or worst-case estimations, which can lead to over design.

4.11.3 Example Process Flow

For the 3G receiver example, both the parties will split the blocks and do the Floor-Plan and Routing, IR analysis, signal Integrity analysis, post layout timing analysis and Post layout Formal verification for various blocks(DDC block, Filter block, Rake Receiver block, Tracking and Error detection, etc.). Further, chip-level floor-planning, placement of the blocks and routing is also carried out.

4.11.4 Locating Cut-points

By looking carefully, cut-points can be identified for *alternate activities* of block-level, and chip-level floor-planning, placement and routing. Also, even at block-level, the work can be distributed, thus saving time.

4.11.5 Flow across Cut-points

Since the blocks are independent, no information needs to flow between teams engaged in block-level activity only.

4.11.5.1 Chip-level Floor-planning, placement and routing

The flow here involves the following in sequence:

I. Both parties will provide the floor-plan guidelines, approximate dimension, layer information, interface(pin details) details for individual blocks. It will be part of placed & routed NETLIST, post layout timing analysis reports, post layout formal verification reports, and DRC results.

II. A team in the second party, involved in chip-level floor-planning, placement and routing will pass back similar information at chip-level.

4.11.6 Operationalization of flow across Cut-points

4.11.6.1 Chip-level Floor-planning, placement and routing

- To ensure that block-level DRC is passed before passing it on to the second party

4.12 Stage 10: Verification & Role-out Stage

4.12.1 General Process Flow

Flow chart

4.12.2 Process Explanation

Full-chip functional verification is done during this phase of the design process. Chip assembly is the process of bringing all the gate-level design blocks together for functional and timing verification, design rule check, pattern generation, and either ASIC vendor sign-off or in-house place and route. It is assumed that all design blocks, embedded blocks, and test blocks have been implemented (RTL) and functionally is verified. This part of the design process is very computation-intensive. Therefore, the verification strategy must be well defined, and test-benches written and in place. Verification of the design requires exhaustive simulations at multiple levels of abstraction' and therefore automated regression techniques should be employed as well as network "task brokering" to make use of all available resources.

4.12.3 Example Process Flow

For the 3G receiver example, both parties will simply participate in IR analysis, signal integrity analysis, post-layout timing analysis and post-layout formal verification.

4.12.4 Locating Cut-points

By observing carefully, a cut-point can be scheduled between (IR analysis, signal integrity analysis) and (post-layout timing analysis, post-layout formal verification). While the first party can do IR analysis and signal integrity analysis, the other party does post-layout timing analysis, formal verification. At this point, both parties combine their work and do conformance activities such as ATPG data generation and simulation DRC and GDSII or CIF generation and sign-off. The other party can also perform project closure with updating the library resources independently.

4.12.5 Flow across Cut-points

In this final stage of the design closure, the regression Chip-level, IR analysis, and Signal integrity analysis reports are being passed to the second party for the final conformance testing. If and when there is any change in the chip-level routing modification, the NETLIST files are transferred *back* to the first party for the above analysis.

4.12.6 Operationalization of flow across Cut-points

Here, both the parties do the Chip-level IR analysis, signal Integrity analysis, post layout timing analysis and Post layout Formal verification, and also ensure that the chip-level DRC is passed. Further, they collaborate to do the Chip-level Test conformation; and Tape-out (GDSII) is generated for the foundry and sign-off activities.

To Summarize for this stage, by taking the Golden RTL, NETLIST and SDF both the Parties confirm that the design fits into the layout and ensure the DRC, Timing and Functionality is right, for the entire chip. This ensures that their design GOAL IS MET. Sign-off activity involves the handover of Final Tape-out (GDSII) format, ATPG vectors and relevant documents to foundry. As the last

activity, the parties share and update the libraries for building future technologies.

5 Other Approaches

A better option to leverage the competencies of comparable groups is to co-ordinate at a somewhat lower level than activity-level. It should also be somewhat higher than task-level co-ordination, because that will be far too detailed to be co-ordinated across two teams. The somewhat **tight-coupling** can potentially involve these teams to sit at the same geographic location.

As an example, consider certain standard-experts from one team helping the other team to come up with the overall proposal. Typically, the architects are used to closed-cards formulation: there is hardly any *delegation of work* involved. In such a case, and also given that certain standards-expert belong to another team, a closed-loop approach is required. A time-complementary approach will fail, because there is no *full-scale* delegation of work. Hence, one needs to come down from activity-level parallelism exploitation.

This *tight-coupled* approach to work sharing is highly suggested, because a lot of delay in feedback(wherever activities involve feedback loops) can be brought down. However, further investigations are needed to make a concrete proposal on this approach, and are out of scope of the current document.

6 References

[1] CoE Embedded Systems. *IC Development Process Handbook.* TCS Internal Document, March 2004.

[2] PS Subramanian and Hrishikesh Sharma. *Elements of System Design: Classification and Organization*. TCS Internal Paper, September 2004.

[3] P.S. Subramanian. "*Formal Verification Techniques and System Design*". TCS Internal Report, April 2004.

[4] PS Subramanian and Hrishikesh Sharma. "Optimal Scheduling of CDFGs using resources having mandatory shut-off periods". TCS Internal Paper, November 2004.