**Hardware Design Standards  
  
for the  
  
 ACMEY Co. Avionics Passenger Counter**

**Part no.** S800-1000-01

Document No: 800-HDS-01

Revision: A

|  |  |  |  |
| --- | --- | --- | --- |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | | \_\_\_\_\_\_\_\_\_\_\_ |
| Project Manager |  | Date | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | \_\_\_\_\_\_\_\_\_\_\_ | |
| Technical Project Lead |  | Date | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | \_\_\_\_\_\_\_\_\_\_\_ | |
| Principal Design Engineer |  | Date | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | \_\_\_\_\_\_\_\_\_\_\_ | |
| Principal Quality Engineer |  | Date | |

|  |
| --- |
| **Notice**  This document and the information contained herein are the property of Airworthiness Certification Services LLC. Any reproduction, disclosure or use thereof is prohibited except as authorized in writing by Airworthiness Certification Services LLC. Recipient accepts the responsibility for maintaining the confidentiality of the contents of this document. |

|  |  |  |  |
| --- | --- | --- | --- |
| REVISIONS | | | |
| Rev. | Reason/Description | Requested/ Changed By | Date |
| A | Initial Release | JB | 1/1/20 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

**Section Page**

[1.0 INTRODUCTION 5](#_Toc218781852)

[1.1 Purpose 5](#_Toc218781853)

[1.2 Scope 5](#_Toc218781854)

[1.3 Acronyms and Abbreviations 6](#_Toc218781855)

[1.4 Applicable Documents 7](#_Toc218781856)

[1.4.1 External Documents 7](#_Toc218781857)

[1.4.2 Internal Documents 7](#_Toc218781858)

[2.0 DESIGN GUIDELINES 8](#_Toc218781859)

[2.1 Design Entry Guidelines 8](#_Toc218781860)

[2.1.1 Top-level Design 8](#_Toc218781861)

[2.1.2 Functional Block Design 8](#_Toc218781862)

[2.1.3 Hierarchical Design 8](#_Toc218781863)

[2.1.4 Synchronous/Asynchronous Design 8](#_Toc218781864)

[2.1.5 Gated Clock 8](#_Toc218781865)

[2.1.6 Revision Label 9](#_Toc218781866)

[2.1.7 Source Readability 9](#_Toc218781867)

[2.1.8 VHDL Coding Standards 9](#_Toc218781868)

[2.1.9 Static Timing Analysis 11](#_Toc218781869)

[2.2 Process-Related Guidelines 11](#_Toc218781870)

[2.2.1 Device & Tool Selection 11](#_Toc218781871)

[2.2.2 Traceability & Archive 11](#_Toc218781872)

[2.2.3 Prototype Activity 11](#_Toc218781873)

[2.2.4 Design Description 12](#_Toc218781874)

[2.2.5 Previously Developed Module 12](#_Toc218781875)

[3.0 FUNCTIONAL FAILURE PATH ANALYSIS 13](#_Toc218781876)

[3.1 Introduction 13](#_Toc218781877)

[3.2 Conducting the Functional Failure Path Analysis 13](#_Toc218781878)

[3.2.1 Identify Functions and Design Assurance Levels 13](#_Toc218781879)

[3.2.2 Propose Architecture 14](#_Toc218781880)

[3.2.3 Identify FFP’s 14](#_Toc218781881)

[3.2.4 Define Implementation and Check for Potential Common Mode FFP’s 14](#_Toc218781882)

[3.2.5 Choose Design Assurance Strategy 14](#_Toc218781883)

[3.2.6 Accept Design Assurance Strategy 14](#_Toc218781884)

[3.2.7 Decompose FFP’s 14](#_Toc218781885)

[3.2.8 Modify Implementation 14](#_Toc218781886)

[3.2.9 Revise Architecture 14](#_Toc218781887)

[4.0 Design Process 15](#_Toc218781888)

[4.1 Initial Design 15](#_Toc218781889)

[4.1.1 Inputs 15](#_Toc218781890)

[4.1.2 Outputs 15](#_Toc218781891)

[4.1.3 Process 15](#_Toc218781892)

[4.2 Detail Design with Functional Simulation 16](#_Toc218781893)

[4.2.1 Inputs 16](#_Toc218781894)

[4.2.2 Outputs 16](#_Toc218781895)

[4.2.3 Process 16](#_Toc218781896)

[4.3 Synthesis 17](#_Toc218781897)

[4.3.1 Inputs 17](#_Toc218781898)

[4.3.2 Outputs 17](#_Toc218781899)

[4.3.3 Process 17](#_Toc218781900)

[4.4 Place & Route 18](#_Toc218781901)

[4.4.1 Inputs 18](#_Toc218781902)

[4.4.2 Outputs 18](#_Toc218781903)

[4.4.3 Process 18](#_Toc218781904)

[4.5 Post-Route Simulation 19](#_Toc218781905)

[4.5.1 Inputs 19](#_Toc218781906)

[4.5.2 Outputs 19](#_Toc218781907)

[4.5.3 Process 19](#_Toc218781908)

[4.6 Device Programming and Test 20](#_Toc218781909)

[4.6.1 Inputs 20](#_Toc218781910)

[4.6.2 Outputs 20](#_Toc218781911)

[4.6.3 Process 20](#_Toc218781912)

[4.7 Documentation Release 21](#_Toc218781913)

[5.0 Tools 22](#_Toc218781914)

# INTRODUCTION

## Purpose

This plan describes the software design standards and has been prepared in accordance with the requirements of RTCA/DO-254.

## Scope

This document will be used as the basis for airborne software development. Once approved, it is implemented during the development and product life cycle of the deliverable airborne software. This Hardware Design Standard complies with the documentation requirements of RTCA/DO-254, Section 10.2.2.

## Acronyms and Abbreviations

AIMS Action Item Management System

ALU Arithmetic Logic Unit

ARP Aerospace Recommended Practice

ASIC Application Specific Integrated Circuit

DRMS Document Review Management System

HAS Hardware Accomplishment Summary

HC1 Hardware Control Category 1

HC2 Hardware Control Category 2

COTS Commercial-Off-The-Shelf

EUROCAE European Organization for Civil Aviation Equipment

FAR Federal Aviation Regulations

FFP Functional Failure Path

FFPA Functional Failure Path Analysis

FHA Functional Hazard Assessment

F-FMEA Functional Failure Modes and Effects Analysis

FTA Fault Tree Analysis

HDL Hardware Description Language

JAR Joint Aviation Requirements

LRU Line Replaceable Unit

PHAC Plan for Hardware Aspects of Certification

PLD Programmable Logic Device

PSSA Preliminary System Safety Assessment

RTMS Requirements Traceability Management System

SAE Society of Automotive Engineers

SC Special Committee

SSA System Safety Assessment

WG Working Group

## Applicable Documents

The following documents are listed for reference only. Each document is applicable to this plan only to the extent specified herein.

### External Documents

RTCA/DO-254 Design Assurance for Airborne Electronic Hardware

FAA Order 8110.4C Type Certification

FAA Order 8110.105 FAA, Simple and Complex Electronic Hardware Approval Guidance

AC 20-152 Advisory Circular, RTCA Inc., Document DO-254, Design Assurance for Airborne Electronic Hardware

### Internal Documents

No references

# DESIGN GUIDELINES

## Design Entry Guidelines

### Top-level Design

The top-level of the hardware design should be done in block diagram or schematic, which partitions the design into reasonably sized functional blocks. The top-level diagram should show all interconnects between the functional blocks. Once the top-level block diagram is designed, its HDL can be generated from the HDL tool and the HDL file can be synthesized. The top-level design may be done in HDL and then converted to a block diagram by the tool, but its output is very difficult to read and may require a substantial manual clean-up. The top-level block diagrams should be laid out to provide a good overview of the hardware design.

### Functional Block Design

After a top-level block diagram is completed, the content of each functional block is to be designed. Either a schematic capture tool or HDL may be used. Of the two methods, using a HDL enables a more portable design. It allows for device independent design (i.e. not tied to a gate-level library until logic synthesis) and design portability. Some blocks will be more suited to using a HDL/synthesis tool and others will be more suited to a schematic capture approach. In general, the HDL method should be used for more complex designs.

### Hierarchical Design

A complex programmable logic design is typically approached in a hierarchical fashion allowing the designer to focus on a manageable portion of the circuit at a time. In general, any hierarchical designs with more than four levels are difficult to manage and review. For the reason, the design hierarchy is limited to no more than four levels from the top function block.

### Synchronous/Asynchronous Design

Synchronous design should be the preferred method over asynchronous design because asynchronous design often depends on element delays which are difficult to predict and verify. Synchronous logic design coupled with static timing analysis provides a reliable method to verify the hardware design timing. It also provides a good overall assurance of robust timing, since simulation methods always face pragmatic limits regarding assurance that all functions are actually tested and that all delay paths have been stimulated. Synchronous logic design also provides a benefit of design reuse issues, including parts, obsolescence issues that require migration of the design to a newer generation device or possibly to a totally different type from the original target device.

### Gated Clock

The gated clocks shall be avoided because they cause more skew between the clock and data and generate glitches. The internally generated clock usage within the hardware design shall be also avoided for the same reason. If the design requires a clock divider within the hardware design, the designer shall use a PLL or a DLL to generate the divided clock.

### Revision Label

If the hardware design has a microprocessor interface, the design should have a register that contains a unique hexadecimal revision label for the build. The format should be MMmm, where MM is a major revision number and mm is a minor revision number. All values are to be read as binary coded decimal (BCD) numbers.

The revision label shall be implemented in a HDL source code file that shall be updated prior to executing synthesis. Therefore, the revision label will be compiled into the binary configuration data and readable through the device interface to the microprocessor. This feature will ease determination of the hardware binary configuration data in the system in the production/maintenance processes.

### Source Readability

HDL source code shall conform to human readability. The review process for the detail design representation should ensure the human readability criteria are met. While the HDL source code shall largely avoid structural notation, the designer may make use of “logic blocks” created by the tools or written separately to render large standard structures. Detail design representations shall be formatted so as to make clear the structures that exist in the design.

HDL source code shall be indented according to structure levels to aid readability. Block diagrams shall be laid out to aid reading the diagram. The detail design representation shall include comments and notations to enhance clarity of the design intent. Excessive comments can hinder clarity by obscuring structure inherent in the representation, and should be avoided.

### VHDL Coding Standards

The following coding standards should be considered for VHDL designs.

1. Every Design/ Test bench should begin with a pre-defined header template. The header contains comments, revision control codes and copyright fields.
2. Everything is lowercase unless otherwise specified. Verilog is case-sensitive, VHDL is NOT.
3. Use 2 spaces for indentation (in place of tab stops)
4. Filename name has to match module/entity name. If the entity name is counter\_unit, filename should be counter\_unit.vhd. Architecture name should obviously be counter\_unit\_ar.
5. Sub module names contain parent module root name as prefix.
6. Designs shall provide sufficient comments to enable self-documenting Code.
7. Use explicit port mapping when instantiating sub modules. Use “Named” association instead of “positional” association.
8. Document all processes, signals, and variables. Enforce documentation of major design elements.
9. Instance names have to be <block\_unit>\_ins. Example. If name of the unit is cnt\_unit, the name of the instance should be cntr\_unit\_ins.
10. Buses are always declared from high to low (e.g. data : std\_logic\_vector [10 downto 0])
11. FSM’s should be coded as one-hot when reasonable.
12. Standard logic used everywhere - NO STD\_ULOGIC!   
    – Recommend std\_logic\_unsigned or std\_numeric   
    – std\_logic\_unsigned looks more like Verilog
13. Use only IEEE packages in general. Some designers re-create functions present in the IEEE version, save it as a separate package and name the package in their own style. Please avoid this.
14. Entity & architecture should be present in one file (industry standard). Some designers have 2 files for a single design, example - count\_ent.vhd and count\_ar.vhd. One of the files contains the entity portion alone, while the other file contains the architecture portion alone, of the same design. This is confusing and people in the industry hate this.
15. Packages should be named <package\_name>\_pkg.vhd. Example: fsm\_package.vhd has to be identified as a package associated with fsm.vhd.
16. If you have a top-level file that consists of 5 sub-units instantiated under it, I want you to include all the 5 component declarations in one single package.   
    Example: If the top-level file called dsp\_unit.vhd has 2 sub-components under it (namely, dsp\_cnt8.vhd and dsp\_cnt16.vhd), you should include the component declarations of these 2 sub-components in a package file called dsp\_comps\_pkg.vhd. Call the compile and call the dsp\_comps\_pkg.vhd file at the top most line (near library declaration area) of the dsp\_unit.vhd file.
17. Configurations should be named <configuration\_name>\_cfg.vhd.   
         Example: xmit\_cfg.vhd
18. Each port declaration should be on a separate line
19. All ports in VHDL are in, out, or inout (no buffer).
20. Name all processes with "\_pr" suffix (e.g. shift\_pr)
21. Naming convention for types– Use [typename]\_t  (e.g. yellow\_t). However, if the type\_name is the name of one of the states in a state-machine, use <typename>\_st. (eg idle\_st).
22. Naming convention for variables– Use [variable]\_v (e.g. variable foobit\_v : std\_logic;)
23. Naming convention for constants– Use [CONSTANT]\_C, ALL UPPER CASE (e.g. REGADDR\_C := "0100" ;) -- REGADDR\_C is type std\_logic\_vector(3 downto 0).
24. Generics defined as all uppercase, no suffix. Only integer types are accepted in Generics. Any other type is not accepted by the synthesizer.
25. Code the register elements as such (library independent). No gate instantiations, infer all cells   
    – Enables design reuse and library portability• Use only IEEE packages in general.
26. Keep levels of hierarchy to a maximum of 2-3   
    – Too many levels of hierarchy become difficult to manage, understand and analyze
27. No clock gating done at the unit level, all clock gating done at a central unit

(Notes: Generally, gating the clock is a bad idea. However, in certain cases, designers intentionally gate the clock either to alter the clock's skew or to insert delay or to produce an asymmetrical clock. Even in these justified clock-gating situations, do not do it at the sub-block level. Perform all the clock-gatings at the top-most level of the chip)

Reasons for gating the clock at top-level:

Industry standard process   
– Enables skew and insertion delay control by layout tool   
– Clock gating in lower-level modules provides challenges to synthesis & simulation tools   
– Isolating to a central module enables design team to manage the implementation

1. Rams instanced at the top level of unit (or core) hierarchy.
2. Clock / reset blocks instanced at top level of core hierarchy
3. Pads are located at top level of chip
4. When coding VHDL avoid using Verilog keywords and vice versa.
5. All outputs are registered (unless connected directly to pads)   
   – Recommended by Synopsys, improves timing convergence. However, while constructing AHB Masters, I have seen some signals called "hready\_in", "hready\_out" etc that do not traverse till the periphery (pads) of the chip, but still remain unregistered. Such specific situations are to be treated as exceptions.
6. All resets and presets are asynchronously asserted active low, synchronously de-asserted   
   – Industry standard ASIC technique   
   – No timing penalty in D path   
   – No layout penalty

### Static Timing Analysis

Most hardware Place and Route tools have some level of static timing analysis included with them. When hardware place & route is done, the tool generates a report that can be used for static timing analysis. The designer shall check the frequency of the main clock against the maximum operative frequency from the report and ensure that the design has a minimum 10% margin. As explained earlier, synchronous logic design coupled with static timing analysis provides a reliable method to verify robust timings in overall design.

## Process-Related Guidelines

### Device & Tool Selection

Some important considerations in selecting the hardware vendor are design support, device cost over time, vendor’s financial stability, tool compatibility with the Systems, and past performance of the vendor. Types of tools that are used may include:

* Schematic capture software
* Fitter software (for place and route)
* Synthesizer software
* Floor planner software
* Simulator software
* HDL synthesis software.

These tools may be obtained from the hardware vendor, or obtained from third party vendors. The target device should be selected based on size requirements, pin count, maximum gate count, complexity of the required logic, speed requirements, power consumption requirements, capability for “on board” programming, and forecasts of part obsolescence. Whenever feasible, the specific piece part should be chosen from a preferred parts list.

### Traceability & Archive

Once the detail design with functional simulation step is completed, all development files shall be checked into PVM prior to executing synthesis. Design traceability and archive shall be done according to the *Hardware Configuration Management Plan*.

### Prototype Activity

The lifecycle for the hardware development allows an iterative approach, beginning first with a prototyping phase. The prototype phase will be used as a means to elicit requirements from our customers and higher design levels. The prototype serves to aid in the generation of requirements for the hardware design. The prototype phase also permits the ability to explore the requirements for the software and hardware. The prototype activity is considered an external process to the development effort. The issues identified in the prototyping phase may be incorporated into the system design description, which allocates requirements for both software and hardware.

After the prototype phase, the iteration of requirements capture can begin. The hardware requirements, conceptual design, and detailed design are cyclic or iterative. This approach allows the hardware design team to design, code, and integrate subsets of requirements at a time, keeping the task at hand small and manageable. The prototyping effort implements many of the functions that may be required for the official development process.

HDL source code brought from the prototyping or other development efforts as suitable shall be subject to hardware Design Standards. After the prototype phase is complete, formal development activities may begin.

### Design Description

The hardware design description should be captured during the detail design with simulation step and may be used for hardware Validation and verification. The design description should consist of the following level of details:

* Assignment of groups of functionality to basic design blocks
* State Diagrams with transitions and I/O signal behavior
* Data flow descriptions describing clock cycle timing
* Memory element descriptions and connections
* Register map with register field descriptions

### Previously Developed Module

The project may utilize source code from previously developed projects. In the event that any design is reused, it will be baselined and used as an entry point to begin development. The project will not take credit for previous certification of the reused design.

# FUNCTIONAL FAILURE PATH ANALYSIS

For projects that are identified as criticality Level B or Level A, Functional Failure Path Analysis (FFPA) is used to provide design assurance confidence for functions, in order to decompose the hardware functions.

## Introduction

FFPA is used to provide design assurance confidence for functions and a process for choosing a design assurance strategy appropriate to the design assurance level. Initially, the design assurance level of a hardware function is determined to identify potential hazards and to allocate associate failure conditions to the function implemented in the hardware. The goal of an FFPA is to identify individual functional failure paths (FFPs) so that hardware implementing functions of different design assurance levels can be addressed by an appropriate design assurance method.

Identification of separate FFPs for functions implemented in different technologies or offering different degrees of design visibility is useful because the total hardware item’s design assurance may be accomplished using multiple methods.

## Conducting the Functional Failure Path Analysis

The FFPA begins by identifying each function allocated to the hardware item, and its design assurance level based on the hardware requirements for that function in consideration of potential design errors***.*** Design errors are emphasized rather than physical faults because they typically present common-modes that can overcome traditional redundancy. Then, the FFPs within the hardware function and the interrelationships among the FFPs are identified. The FFPA begins by identifying each function’s boundaries that may be assemblies, circuits, components, or component elements, as appropriate and as needed. At the bottom level of the decomposition, the FFPs are either simple enough for deterministic verification or verifiable following an acceptable design assurance strategy appropriate for the design assurance level.

For each FFP with a Level A or B design assurance level, the proposed means of implementing the function and the design assurance options are also evaluated. The decomposition boundary must provide sufficient testability to ensure acceptable assurance data will be available for the design assurance strategy or strategies chosen. If there is no acceptable strategy for design assurance or if it appears that acceptable design assurance data cannot be expected to become available, a new conceptual design is proposed and the FFPA is iterated. Either the implementation of the function or the hardware architecture must be modified until an acceptable strategy of design assurance has been determined that can be expected to provide acceptable design data for all FFPs.

Decomposition is performed using conventional top-down assessment techniques and may be complemented by bottom-up analysis methods for each successive level of decomposition. FFP has been identified, a design strategy appropriate to the implementation and the design level has been chosen for each FFP.

The major steps in performing an FFPA are identified in the following paragraphs, in the order in which the steps should be taken.

### Identify Functions and Design Assurance Levels

The FFPA process begins by identifying the system functions allocated to the hardware item and their design assurance levels based on the hardware requirements for that function. This analysis includes the consequences of each function’s potential loss or anomalous behavior individually.

### Propose Architecture

Propose an architecture for the hardware function. Identify any sub functions and interrelationships between hardware functions.

### Identify FFP’s

For each of the hardware sub functions in the proposed architecture, identify all of the FFPs and their design assurance levels, considering each FFP’s potential loss or anomalous behavior individually*.*

### Define Implementation and Check for Potential Common Mode FFP’s

Develop a conceptual design to implement the FFP. Check for potential common-mode design errors between FFPs that may compromise their assigned design assurance levels. If such common-mode potential exists, the design assurance levels of the applicable common mode FFPs may need to be increased, a different implementation may be needed, or a detailed examination may be needed to justify no changes.

### Choose Design Assurance Strategy

Consider the available design assurance strategies and determine which may be applicable or appropriate. The basic strategy for any FFP is requirements-based verification to show that all requirements have been met. The FFPA is complete for FFPs having design assurance levels of C, D, or E that present no common-mode impact to level A or B FFPs.

### Accept Design Assurance Strategy

The chosen design assurance strategy is acceptable if sufficient useful design assurance data can be expected to be available. For example, the assurance data provided by verification tests or simulations provides sufficient testability to obtain acceptable data.

### Decompose FFP’s

Determine if decomposing the candidate FFP into smaller FFPs is viable. Doing so may provide the added testability needed to provide acceptable design assurance data. If the FFP is decomposed, this process is repeated for the newly defined FFPs. Such decomposition may be considered architectural mitigation, particularly if the resulting derived FFPs specifically mitigate design errors.

### Modify Implementation

Is a different implementation viable? If there is no acceptable strategy for the defined hardware implementation, review alternative implementations and repeat the process described in section 2.2.4.

### Revise Architecture

If there is no hardware implementation of the proposed architecture that can provide acceptable design assurance data, revise the functional architecture and re-enter the FFPA process.

# Design Process

## Initial Design

### Inputs

The following are inputs to the design process:

* Hardware Requirements
* Partition of hardware function to programmable logic.
* Design Entry tool

### Outputs

The following are outputs of the design process:

* Top-level block diagram.
* Description for functional blocks in terms of logic elements or HDL representation.

### Process

The initial step of the design is to create a top-level block diagram, which partitions the design into reasonably sized functional blocks. The top-level diagram should show all interconnects between the functional blocks. After a top-level block diagram is completed, the next step is to define the contents of each functional block. Testability issues should be addressed at this time to ensure the programmable logic design is compatible with the planned test approach.

## Detail Design with Functional Simulation

### Inputs

* Top-level block diagram
* Description for functional blocks in terms of logic elements or HDL representation
* Design Entry tool
* Simulation tool

### Outputs

* Design source files to be checked into PVM
* Simulation files consisting of input vectors created to test the functional blocks of the hardware design. These input vectors, along with the predicted results (bit patterns, waveforms, and/or test bench files) make up the simulation files.
* Hardware Design Description

### Process

For complex devices, each functional block is designed and simulated for its functional operation. Functional vectors are written which thoroughly test the performance of the block by providing significant inputs and predicting the expected outputs.

A test bench technique may be used in addition to or in place of test vectors. The test bench uses more sophisticated simulation techniques that are particularly more useful for more complex designs and for timing checks, and also it can be re-used for the post-route timing simulation.

Other forms of stimulus definition such as waveform-based stimulus files may also be used. It is important that expected results be determined prior to simulation so that discrepancies between the actual outputs and the expected outputs are readily recognized.

## Synthesis

### Inputs

* Design source files in HDL
* Synthesis tool
* Synthesis constraint file

### Outputs

* EDIF file that represents hardware design
* Synthesis log file

### Process

The synthesis tool translates the HDL source code into an aggregation of primitive blocks, such as look up tables, registers, buffers, etc., available in the target hardware device. For complex programmable logic devices, an optimizing process is followed which trades off design size and speed. The synthesis tool creates the implementation according to the target hardware parameters. This optimization is an iterative process which is complete when satisfactory performance is achieved. The synthesis log file should be reviewed to ensure all the warnings are accounted for.

If this synthesis process fails to produce a design that meets all of the performance requirements, then the design in the previous (initial and detail designs) steps may have to be corrected or changed. This iteration process may have to be repeated several times. The specific device piece part choice is confirmed based on the results of the optimization process described above. Sometimes an alternate device piece part choice may be necessary.

## Place & Route

### Inputs

* EDIF file that represents hardware design
* Place & Route constraint file that contains design pin out and timing constraints
* Place & Route tool

### Outputs

* HDL Timing Simulation Model
* Binary Configuration Data to program the target device
* Place & Route reports

### Process

The Place & Route is performed using the Place & Route tool and its associated programs. These programs perform the steps necessary to place primitive blocks for the synthesized EDIF net list into the target device and route the design signals between those blocks. The Flow Engine process generates a file that is used to program the target device. If the target device is a RAM-based FPGA, the file is converted into an image that can be programmed into the configuration PROM for the target device. These files are released as hardware Configuration Data.

The Flow Engine process also generates a timing-annotated HDL model and Place & Route reports. This HDL timing model shall be used for the verification process as the model for simulation. The HDL timing model shall be maintained as a part of the hardware verification results.

## Post-Route Simulation

### Inputs

* HDL Timing Simulation Model
* Actual signal delay data based on synthesized design and device models
* Simulation Tool
* Simulation input vectors, test benches, and/or other stimulus file types created in detail design step

### Outputs

* Design corrections needed based on the results of the simulation

### Process

The delay data is back-annotated into the design database and then the hardware design simulation is rerun to verify that the design still meets its performance requirements and all required timing margins. In post-route simulation, the maximum delay should be used for timing analysis, while simulation with minimum or typical delay is possible. The maximum delay simulations shall include both on-chip (clock speed, propagation delay) timing verification and off-chip (set-up, clock-to-out) timing verification.

If the simulation passes, the process proceeds to the Program and Test step. If the simulation fails, corrections shall be made and the appropriate process steps shall be repeated. The steps that shall be repeated may start at the Initial Design step, Synthesis step, or Place and Route step, depending on the nature and severity of the design corrections.

Varying the programmable logic performance parameters until the programmable logic simulation fails may also be done to perform margin analysis. This would provide data on how much margin exists above the frequency at which the programmable logic device was designed to operate. The simulations performed in this step may be repeated in the formal hardware verification process. This is further detailed in *Hardware Validation and Verification Standards*.

## Device Programming and Test

### Inputs

* Binary Configuration Data to program the target device
* Device programming tool provided by the target device vendor
* All the information necessary to program the target device

### Outputs

* Test data, which are pass/fail results of the tests, defined by hardware Verification procedure

### Process

For the flash-based hardware, the binary configuration data is downloaded to the programmer and the target device is programmed. If the target device is a RAM-based FPGA, the file is downloaded to the programmer and then programmed into the configuration PROM for the target device. Once programmed, the device is marked as an altered item giving a new assembly part number. For the hardware programmed in circuit, testing of the resulting functionality is done at the Circuit Card Assembly level or higher. This is further detailed in *Hardware Validation and Verification Standards*.

## Documentation Release

After the hardware design is fully verified formally, it may be released to production. The followings are the deliverables for the hardware design released to production:

* Binary Configuration Data to program the target device
* EDIF or equivalent file that enables the design to be transported to other tools
* Information necessary to generate the hardware build
* Information necessary to program the target device
* Hardware design description
* The hardware design tool set
* Target hardware data
* Simulation files consisting of input vectors, test benches, or other stimulus file types created to test the functional blocks of the programmable logic design. These stimulus files, along with the predicted results (bit patterns, waveforms, and/or test bench data) make up the simulation files.
* Hardware verification data from formal hardware verification procedure
* Hardware Configuration Index

# Tools

| **Activity** | **Tool** |
| --- | --- |
| Design Entry | Aldec Active HDL,  Mentor Graphics ViewLogic |
| Synthesis | Synplicity Synplify,  Xilinx XST |
| Place & Route | Xilinx ISE,  Actel Libero |
| Simulation | Active HDL |
| Device Programming | Xilinx iMPACT,  Actel Flash Pro |
| Source Revision Control | Vissual Source Safe |