CORTEXM0 Methodology Guide
IC Compiler

Version 2008.09

March 2009

Confidential

SYNOPSYS®

Copyright Notice and Proprietary Information

Right to Copy Documentation

Destination Control Statement

Disclaimer

Registered Trademarks (®)

Trademarks (™)

Service Marks (sm)

Table of Contents

1 Introduction

Through the results of close technical collaboration between ARM and Synopsys, the process used by ARM Licensees to port synthesizable ARM microprocessor cores to their chosen technologies has been significantly streamlined. The ARM-Synopsys Reference Methodology represents the set of best practices for implementing synthesizable ARM IP with Synopsys tools and provides a proven, low risk solution to providing application specific ARM IP. The ARM-Synopsys Reference Methodology is a fully integrated implementation solution based around Synopsys' Galaxy SI Design Platform. Galaxy is an open, integrated design implementation platform built on best-in-class EDA tools.

The ARM-Synopsys Reference Methodology (from here on referred to as the 'Methodology') provides complete support for the hardening and modeling of synthesizable ARM processors using a physically aware based flow with integrated SI capability for design closure when targeting leading edge CMOS technologies.

This document describes Version 2008.09 of the Methodology based upon IC Compiler tools, and its use in implementing the CORTEXM0 microprocessor from ARM.

The start point for the Methodology is a fully configured RTL representation of the CORTEXM0 that has been validated (using ARM's validation environment) as being compliant with the ARM architecture. This RTL representation is implemented in a target technology to yield an optimal cell level representation of the processor. Highly accurate models or abstractions of this implementation are created that allow the hardened ARM processor to be deployed as a library level component in a system chip. These models are of a sign-off quality and are created automatically as part of the Methodology.

This Methodology has been created as a **Reference** and it is recommended that it be used as a Reference. The Methodology encapsulates a set of best practices for implementing ARM IP with Synopsys tools and will give the User an excellent starting point for their specific implementation projects.

The Methodology has been developed as a means to transfer detailed knowledge of the best practice implementation and modeling of synthesizable ARM IP, to ARM's Licensee community using TSMC 90nm technology with ARM-Artisan libraries. It is expected that this Methodology is fully portable across the majority of other leading edge 90nm CMOS processes.

Note that the Methodology supports two levels of hierarchy for CORTEXM0 – CORTEXM0 and CORTEXM0INTEGRATION – but the implementation is sufficiently similar that only the CORTEXM0 level of hierarchy is described in this document.

In order to allow configuration of the CORTEXM0, or addition of user I/O to the top level, implementation wrappers are supplied for both CORTEXM0 and CORTEXM0INTEGRATION levels of hierarchy. These are respectively named CORTEXM0IMP and CORTEXM0INTEGRATIONIMP and are the names of the top levels that are implemented in the iRM. Please see the Configuration and Sign-off Guide for detailed information on configuration of the CORTEXM0.

1.1 Tool Version Support

The Methodology is based primarily on the 2008.09 release of the Galaxy Implementation Platform from Synopsys. The supported versions for each of the tools that comprise Galaxy are defined in the following table. Use of the Methodology with alternative tool versions is not supported and may yield unpredictable results.

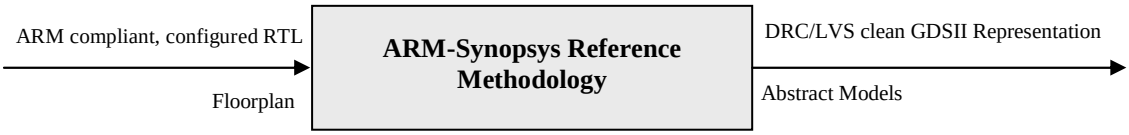
Synopsys Product	Supported Version
Design Compiler	2008.09-SP2
IC Compiler	2008.09-SP2
Formality	2008.09-SP2
PrimeTime SI	2008.06-SP3
PrimeTime PX	2008.06-SP3
TetraMAX	2008.09-SP2
Star-RCXT	2008.12
PrimeRail	2008.12

1.2 Technical Support

Technical support for the Methodology is available from ARM's support team (support@arm.com) and from your local Synopsys support organization.

2 Reference Methodology Overview

The Methodology takes a configured RTL representation of an ARM core and performs all implementation steps through to a DRC/LVS clean GDSII representation together with an accompanying set of models representing specific characteristics (timing, test, physical, etc.) of the final implementation. Consequently, the Methodology can be viewed conceptually as being very similar to that of a memory compiler. The primary goal of the Methodology is to perform this implementation process as quickly as possible without sacrificing performance, power or area, and, hence, enabling the User to rapidly port synthesizable ARM IP to a number of target technologies in an optimal, application specific way.



2.1 Overview

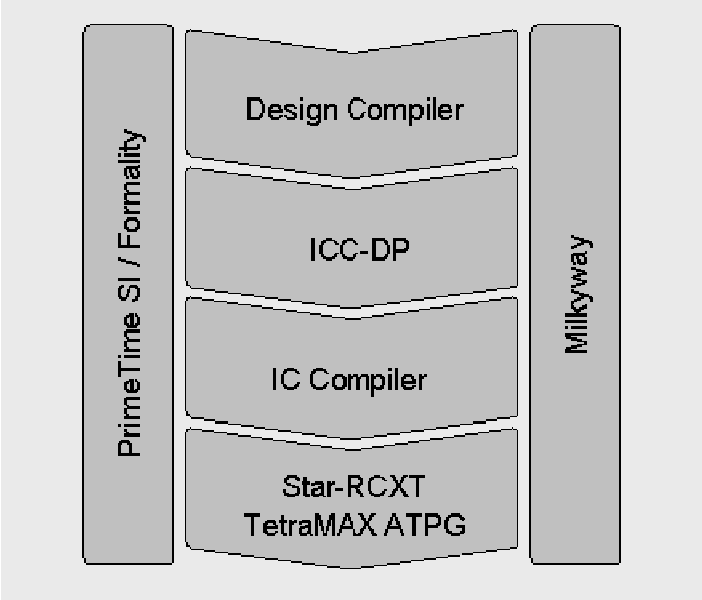
Synthesizable ARM processor IP is licensed in a soft form (RTL). The ARM Licensee configures and implements this IP in a target technology. In order for the ARM Licensee to deploy the hardened core to its customer base for use in ARM based system chips the implementation details and detailed logic design of the IP must remain hidden and so the hardened ARM IP must be abstracted into a set of models or views. These models or views must be of a sign-off quality and allow the processor to be deployed as a library level component in the ARM based system chip.

Specifically, the following models must be created:

- Timing Model for synthesis, static timing analysis and timing driven place and route
- Test Model to capture the testability of a core and to support chip-level test
- Physical Model to support physical synthesis and layout at the system chip level
- Power Grid model for use in voltage drop analysis at the system level

A functional model is also required. However, ARM provides a Design Simulation Model (DSM) with the IP for use in functional simulation.

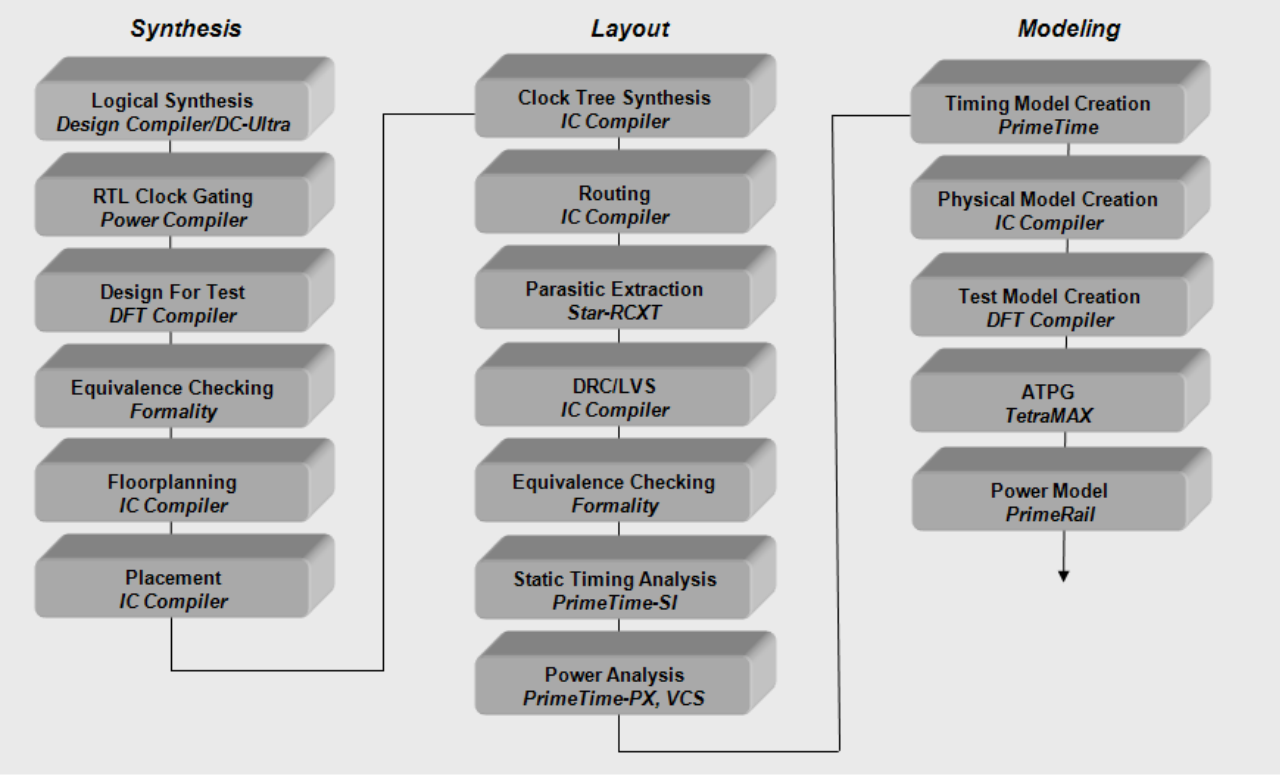
The Methodology is based on Synopsys' Galaxy Design Platform. Galaxy is an open, integrated design implementation platform anchored by the open Milkyway design database. Physical synthesis coupled with an optimized layout environment form the basis of the Methodology, all within the same database. PrimeTime-SI and Formality are used for timing and formal verification sign-off.



In addition, there is fully integrated support for Design-For-Test, Low Power, Datapath and Clock structures as well as a highly efficient approach to static verification to accelerate time to market.

Comprehensive synthesis support provided by Design Compiler is coupled with a complete layout solution in ICC. Additional capability is integrated within these components of the Galaxy platform including:

- RTL Clock Gating with Power Compiler
- High performance synthesis with DC-Ultra
- Scan and test wrapper synthesis with DFT Compiler
- Pin optimization and Power Network Synthesis with IC Compiler
- Crosstalk prevention within IC Compiler
- Power Network Analysis with PrimeRail



All of this capability is supported within the Methodology and can be enabled through the use of configuration parameters described later. The result is a complete flow that supports synthesis, layout and modeling of CORTEXM0.

The high level of integration within IC Compiler provides a rapid synthesis flow that creates an optimized placed gates netlist with excellent correlation to post layout timing. The compile strategies employed within the Methodology are a result of numerous investigations on a number of competitive CMOS technologies at 90nm. While results will vary as the processor is ported to different technologies, the fundamental principles employed in the Methodology hold true from one technology to the next.

The layout phase of the Methodology provides a similar level of integrated support from within ICC. Support for clock tree synthesis, global and detailed routing, crosstalk prevention and power network analysis as well as verification through the use of DRC and LVS are integrated seamlessly within the ICC environment.

Static verification methods are employed to speed verification at various stages in the implementation flow. All verification of the design's functionality is performed with Formality, while the timing and signal integrity is verified statically with PrimeTime-SI after multi-corner extraction using Star-RCXT.

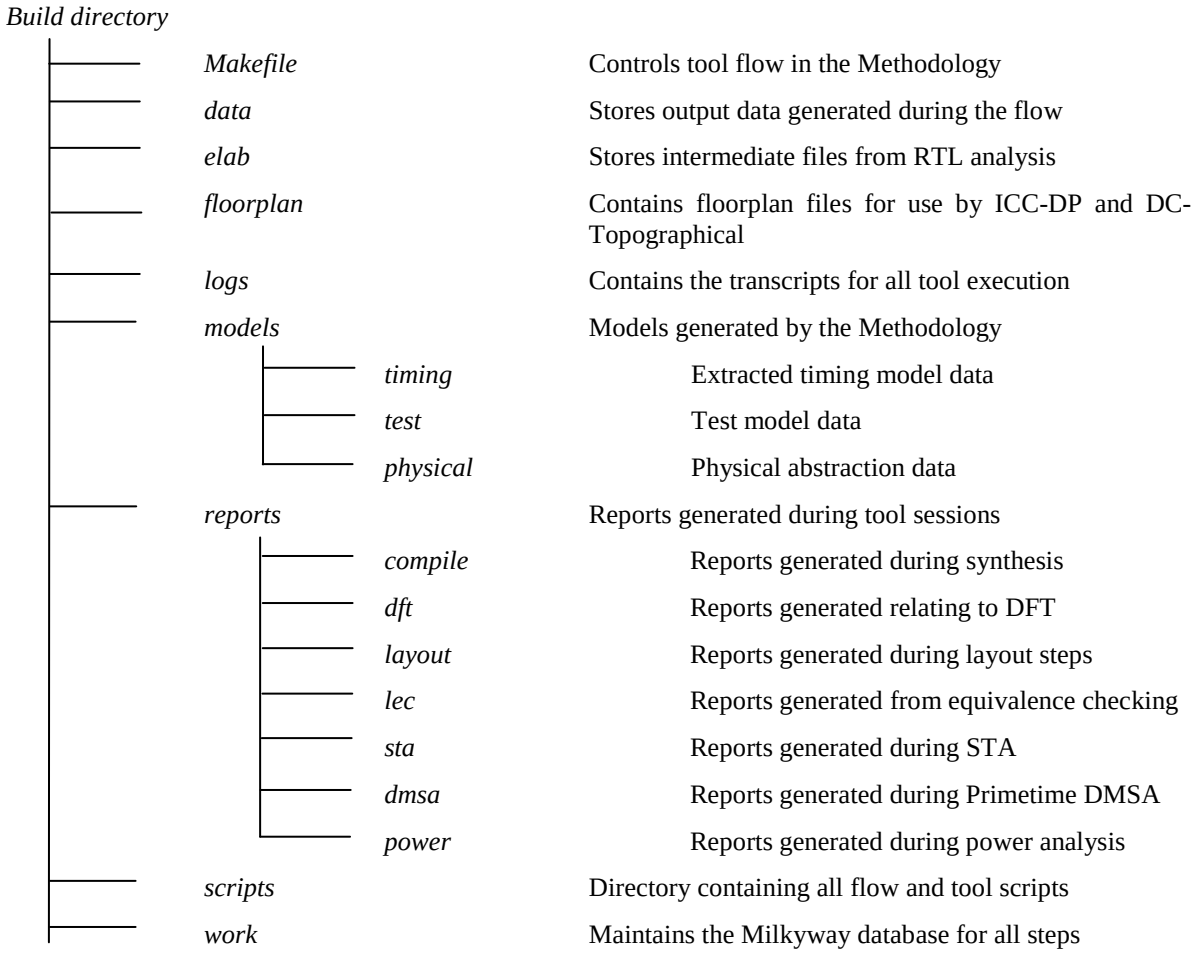
All of this capability is combined within the Methodology to provide the User with a complete, proven, low risk implementation solution for synthesizable ARM IP using Synopsys tools.

2.2 Directory Structure

Implementation of CORTEXM0 using the Methodology operates within a pre-defined directory structure, called the *build directory* (*build_dir*). This build directory resides in the *synopsys* directory in the distribution from ARM.

The User can have as many parallel build directories within the *synopsys* directory as they wish, supporting concurrent implementations of a given configuration of CORTEXM0 with varying implementation approaches. The naming of these build directories is at the discretion of the User.

The structure of the build directory is shown in the diagram below. This directory structure is fixed and cannot be changed by the User without significant modification to the scripts. All tool execution steps in the Methodology are performed in the work *(work/)* directory. All data generated by any step in the flow is stored in the data directory (*data/*). At the top level of the build directory are directories to hold intermediate files (*elab/*), the log files (*logs/*), the scripts that comprise the Methodology (*scripts/*) and a directory that is used to hold the models that are automatically generated as part of the flow (*models/*). All reports are written into sub directories within *reports/*. A simplistic Makefile is also included to define and exercise the various steps in the flow and to illustrate the way in which the tools should be invoked. The User can add as much control and dependency to this Makefile as is desired.



3 Flow Setup

This section discusses the synthesis of CORTEXM0 with Design Compiler and IC Compiler. The goal of the synthesis step is to create a placed gates implementation of CORTEXM0 that is functionally equivalent to the configured compliant RTL image and meets the required performance, area and power goals. In order to satisfy tight time to market requirements synthesis with DC-Topographical enabled Design Compiler followed by detailed placement with IC Compiler is used to enable rapid timing closure.

3.1 Initial Setup

The configuration scripts in the *scripts* directory should be used to control the overall flow through the tool steps. There are two main configuration files containing:

- Flow control parameters
  - The *flow configuration* file (*cortexm0imp_config.tcl*) contains a number of TCL variables that are used throughout the scripts to control the tool flow and to define specific features of the flow.
- Technology-specific Parameters and Library Setup
  - These parameters are defined in a separate file (*cortexm0imp_tech.tcl*) which is usually sourced after *cortexm0imp_config.tcl.*

3.1.1 Flow control parameters

There are a minimal number of flow control options with the Methodology since as a Reference, the level of configurability has been limited in order to ensure a predictable, deterministic flow for the User. The following tables identify the flow control options that are available to the User, together with their default values.

Power Analysis Flow Parameters

This section of the *flow configuration* script defines variables used to control power analysis features of the flow.

Option	Default Value	Purpose
rm_use_vcd_for_pp	0	When set, PrimeTime-PX will use VCD (Value Change Dump) information to compute power consumption. (Note: VCD files are not supplied as part of the Methodology)
rm_pp_activity	0.25	Switching activity applied on inputs in vector-free PrimeTime-PX analysis.

As part of the methodology the design is scan inserted and, optionally, wrapped with an IEEE P1500 compliant test wrapper. The number of scan chains, wrapper chains and the names of the scan control signals are also set in this file. Scan compression is also an option.

Design Configuration Parameters

This section of the *flow configuration* script defines variables used to specify the design configuration.

Option	Default Value	Purpose
rm_include_dbg	1	Specifies whether debug logic is included in the RTL with the parameter 'DBG' (See configuration and signoff guide for details)
rm_include_ahbslv	1	Specifies AHB-Lite SLV port compliance, as defined in the RTL with the parameter 'AHBSLV' (See configuration and signoff guide for details)
rm_include_wic	1	Specifies whether a Wakeup Interrupt Controller is included in the RTL with the parameter 'WIC' (See configuration and signoff guide for details)

DFT Flow Configuration Parameters

This section of the *flow configuration* script defines variables used to control the DFT setup for the design.

Option	Default Value	Purpose
rm_num_scan_chains	4	Defines the number of scan chains to be inserted into the design.
rm_create_test_wrapper	0	When set, directs DFT-Compiler to invoke Test Wrapper Insertion capability to synthesize an IEEE P1500 compliant test wrapper around CORTEXM0.
rm_num_wrap_chains	2	Defines the number of test wrapper chains to be synthesized if the rm_create_test_wrapper variable is set. This value includes both input and output-side scan chains.
rm_use_scan_comp	0	When set, directs DFT-Compiler to perform Scan Data Compression
rm_comp_ratio	10	Default value to use for pattern compression ratio
rm_comp_xtolerance	default	Defines the level of tolerance to X's into the compressor
rm_comp_min_power	false	When set, disables activity in the compressor logic whenever chains are not shifting
rm_low_power_atpg	false	When set, enables low-power shift features: minimizes dynamic power during shift
rm_atpg_power_budget	25	Switching activity threshold, expressed as a percentage. TetraMAX uses this variable to limit switching activity, and thus power consumption, during generation of test vectors.

Layout Flow Configuration Parameters

Option	Default Value	Purpose
rm_use_mcmm	0	When set, enables concurrent multi-corner multi-mode (MCMM) optimizations in IC Compiler steps
rm_use_zroute	1	When set, enables the use of ZRoute routing technology within IC Compiler. When reset to 0, the classic router within ICC is employed.

Implementation Configuration Parameters

This section of the *flow configuration* script defines variables used to control implementation options of the flow.

Option	Default Value	Purpose
rm_use_multivt	1	Enable use of multiple threshold voltage standard cells. Set to 0 to limit synthesis and optimisation.to a single voltage threshold standard cell library

3.1.2 Technology-specific parameters

Technology specific parameters are defined within the technology setup file *cortexm0imp_tech.tcl*. These variables typically define the parameters associated with the clocks and IO in the design. In order to allow the User to quickly port CORTEXM0 from one technology to another, all technology specific information has been located in this single script.

Optimization and Constraint Parameters

This section of the *technology* script defines variables used to constrain the design and to set up the environmental parameters. Values suggested are for a typical 90nm implementation, the User is responsible for setting them appropriately for their technology.

Parameter	Default Value	Purpose
rm_load_value	0.10	Specifies the capacitive load value to be placed on all outputs of the design (pF)
rm_driving_cell	BUFX4AD	Defines the type of cell that is to be used as the driving cell for all (non-clocking) inputs to the design
rm_driving_pin	Y	Defines the name of the output pin of the driving cell (above)
rm_clock_driving_cell	CLKBUFX8AD	Defines the type of cell that is to be used as the driving cell for clock ports in the design
rm_clock_driving_pin	Y	Defines the name of the output pin of the driving cell (above)
rm_period_jitter	0.02	Expected period jitter from the PLL (ns).
rm_dcd_jitter	0.05	Expected duty cycle jitter from the PLL (% of period).
rm_setup_margin	0.05	Extra margin for setup checks. Applied in both implementation and signoff.
rm_hold_margin	0.05	Extra margin for hold checks. Applied in both implementation and signoff (ns)
rm_ocv_derate_factor	0.00	Derate to be applied on clock paths; eg. 5%
rm_clock_uncertainty	0.25	Specifies the uncertainty in the arrival time of the clock, pre-CTS (ns)
rm_critical_range	0.10	Specifies the critical range (eg. 10% of the rm_clock_period)
rm_icg_name	TLATNTSCAX8AD	Defines the clock gating cell to be used by Power Compiler
rm_min_icg_fanout	3	Specifies the minimum fanout for ICG insertion.
rm_max_icg_fanout	64	Specifies the maximum fanout for an ICG cell.
rm_max_fanout	32	Specifies the maximum fanout for non-ICG cells.
rm_max_transition	0.80	Maximum transition constraint for nets within

		the design (ns)
rm_max_clock_transition	0.40	Maximum transition constraint for clocks within the design (ns)
rm_clock_period	5.00	Target clock period for the system clock (ns)
rm_clock_latency	0.60	Estimate of post-CTS insertion delay on the main system clock in the design (ns)
rm_icg_latency	0.40	Specifies the insertion delay to each integrated clock gating cell in the design (ns)
rm_cts_latency_WCLK	0.30	Estimated clock latency of wrapper clock after CTS. Used to apply a source delay to model SoC-level balancing against core clock (ns)
rm_max_operating_condition	scadv_tsmc_cln90g_rvt_ss_0p9v_125c	Operating condition to use for min-max settings
rm_min_operating_condition	scadv_tsmc_cln90g_rvt_ff_1p1v_m40c	Operating condition to use for min-max settings
rm_typ_operating_condition	scadv_tsmc_cln90g_rvt_tt_1p0v_25c	Operating condition to use for min-max settings
rm_process_corner	cworst	Default process corner for use in STA
rm_process_corner_power	typical	Default process corner for use in power analysis

DesignWare setup for Formality and Primetime script location for ATPG

Parameter	Default Value	Purpose
rm_hdlin_dwroot	-	Defines the location of the Designware installation. Used by Formality to resolve Designware instances. Usually set to the top of the tool install directory tree.
rm_pt2tmax_path	-	Location of the script *pt2tmax*

TetraMAX library settings

Parameter	Default Value	Purpose
rm_tmax_verilog_libs	-	Defines the locations of TetraMAX model views for standard cells and memory instances

The technology script also sets up the locations of the various libraries and search paths required by the synthesis tools including:

- Path to library installations for standard cells and libraries – *rm_lib_dirs*
- Target technology library name(s) - *rm_target_library*
- Search paths for all technology libraries – *rm_search_path*
- Link paths – formed from *rm_mintypmax_libs*

Due to the variety of ways used to organize libraries and associated technology based files, the Methodology uses a reference pointer *rm_lib_dirs* as a start point for each individual library target. This libraries directory can contain actual library directories or links to the libraries and is used to demonstrate how the reference data can be grouped.

A number of variables are set to point to required cell (*rm_rvt_reflib* and *rm_hvt_reflib*) and compiled macro libraries required for synthesis and design assembly; these are then referenced to create the *rm_mw_reflib* variable. This variable is then used in Design Compiler to create the *search_path* and in creating the Milkyway design library.

The Design Compiler and IC Compiler scripts also require specific variables to be defined to identify the logical (*target_library*) and physical target technology libraries used during synthesis and optimization as well as the link libraries; these are built using the variables *rm_target_library* and *rm_mintypmax_libs* respectively. The logical library names are also used at other steps in the flow, for example, during the reference of library cells to be set as "dont_use" and multi-Vt cell distribution reporting. Library ".db" files are **always** referenced from a Milkyway LM (logic model) view in keeping with the data structure model used in the Methodology.

In addition to Design Compiler and IC Compiler, the technology script setup is used by most other tools in the flow. Some specific setup data is also required for these tools, for example, they may require that the *search_path* variable be set explicitly and so these will reference the *rm_search_path* variable.

The technology script also includes a number of lists of cells to be used at various points throughout the flow. If the User wishes to avoid the inference of specific cells in the design then these cells must be specified in the *rm_dont_use_list* variable.

Other lists are included to define clock tree cells - *rm_clock_cell, rm_clock_size_cell and rm_clock_delay_cell*, filler cells - *rm_fill_cells_wm* and *rm_fill_cells_wom* (those with and without metal, respectively) and tie-cells - *rm_tie_hi_lo_list*; these are referenced by IC Compiler during the clock tree synthesis and routing phases. Many of these lists reference high-Vt as well as regular-Vt cell; for a single Vt flow remove the cells one does not wish to use.

Single and Multi-Threshold Synthesis and Libraries

The Methodology directly supports multi-Vt synthesis and optimization but can also support single-Vt synthesis and optimization; the latter is achieved by setting the variable rm_use_multivt to 0 (see Implementation Configuration Parameters in section 3.1.1). This variable affects the construction of rm_target_library and rm_mintypmax_libs lists, which are used in most implementation and analysis scripts.

4 Flow Overview

This section provides a brief overview of the flow steps employed in the Reference Methodology.

4.1 Makefile

The RM flow is controlled by a Makefile located in *build_dir*. All parts of the flow can be initiated from this directory using a 'make' command together with a Makefile target. To create multiple, parallel, build directories one must copy the entire (clean) structure over to a new area, including the Makefile; all paths referenced within the RM flow are relative and so a parallel copy will always point to the same source data, etc.

To start a new build one must first create all the working and data directories with the command **make dirs**. To clean up a working area that contains a partial or completed build of CORTEXM0 that is no longer needed then the command *make clean* should be used.

Assuming that all required tools are in the Users *$path* and that all appropriate licenses are available then CORTEXM0 can be built either in individual stages – recommended for the first build run, or in parts – the synthesis parts using **make front** and the layout parts with **make back**, etc., or the whole flow together – using **make build**. ARM and Synopsys recommend that the flow is run 'out of the box' to prove that it works and to get familiar with the usage before it is modified to create the Users desired configuration.

4.2 Clock Jitter, Variation and Margins

The Methodology includes support for clock jitter, on-chip variation (OCV) and additional margin on setup & hold checks. These features all provide extra – cumulative – constraint on the design and the User is cautioned to be aware of the implications of changes to these settings.

4.2.1 Clock jitter and duty cycle distortion

Clocks from PLLs are non-ideal and the Methodology is particularly concerned with two types of jitter that can be seen:

- Period jitter, often referred to as "clock jitter": this affects the rising edge of the clock and is of most concern to a design that operates mainly on that clock edge. Because period jitter can shorten the cycle time available for the logic to function this directly affects the final core performance.
- Duty cycle distortion, where the clock duty cycle changes from cycle to cycle: this affects timing of negative edge paths. Similar to the above, this again directly affects final core performance.

Clocks within the design can be declared with both period jitter and duty-cycle distortion, throughout the Methodology. The amount of jitter is controlled by two variables (*rm_period_jitter* and *rm_dcd_jitter*) and Users should ensure that these are set to reflect the properties of their own PLL, as typically found in the relevant datasheet.

4.2.2 On Chip Variation

The Methodology models OCV effects on clock nets by speeding up/slowing down clock paths by a User configurable percentage. Note that it is up to the User to choose an appropriate value for this. Excessive OCV margin may result in significant increase in cell area. The variable that controls the percentage is: *rm_ocv_derate_factor*.

4.2.3 Setup and Hold Margin

Further margin on setup and hold checks is sometimes desired for additional design pessimism. This is supported in the Methodology through the use of the configuration variables *rm_setup_margin* and *rm_hold_margin*. Again the User is cautioned that over-constraining may harm QoR.

4.3 Synthesis – Design Compiler

The synthesis flow is executed entirely within Design Compiler using a number of DC-TCL based scripts. These scripts offer the User sufficient parameterization to accommodate application specific hardening of CORTEXM0, while still providing a deterministic and predictable flow.

A single script executes the flow; this script in turn calls additional scripts where necessary to perform specific tasks:

cortexm0imp_compile.tcl Primary script. Performs the synthesis setup and runs the initial compile steps.

The additional scripts called may also be used in other parts of the Methodology:

cortexm0imp_config.tcl	Configuration script. This script is used to manage configuration of the flow. The synthesis approach and other flow parameters are defined in this script.
cortexm0imp_tech.tcl	Technology definition script. This script is where all the required technology-specific information is defined.
cortexm0imp_verilog.tcl	Identifies all components of the RTL image for a specific version of CORTEXM0. Used both in synthesis and equivalence checking.
cortexm0imp_constraints.tcl	Interface constraints specifying the interface timing for CORTEXM0. Used in both synthesis and static timing analysis.

Implementation of CORTEXM0 takes place in the *work* directory. The compile step manages the initial mapping of the RTL to gates and creation of the Milkyway library and is invoked using the command **make compile** in the build directory. Reports are output from the synthesis phase into the *reports/compile* directory.

4.3.1 Scan Insertion – Design Compiler & DFT MAX

All scan structures are inserted using the SCANDEF flow in Design Compiler. The Methodology supports a number of DFT methodologies, including:

- Insertion of internal scan chains
- Insertion of wrapper chains (optional)
- Scan compression using Synopsys Adaptive Scan technology (optional)

After initial synthesis, the design is prepared for scan insertion using the **`set_scan_configuration`** command. Then the design is checked for Test Design Rule violations, using the **`dft_drc`** command, and the scan chain architecture is previewed using **`preview_dft`**. Scan structures are then inserted in the design using the **`insert_dft`** command.

Subsequent to scan insertion, all the required test views, reports and protocol files are generated. At this stage in the flow all the required scan structures are now in place, though not necessarily in their final ordering. Scan chain definitions are saved by using the Design Compiler **`write_scan_def`** command. This command generates a SCANDEF file, which is an ASCII file that specifies a list of stub chains that can be reordered by IC Compiler without requiring additional test design rule checking.

The Methodology supports two non-exclusive physical DFT methods to optimize scan chains:

- **Placement-Aware Scan Chain Reordering** — The scan chains are repartitioned and reordered, based on scan cell locations.
- **Clock-Aware Scan Reordering** — The scan chains are reordered to minimize the number of clock buffer crossings in the scan chain

For more information on the SCANDEF flow, see the *DFT Compiler User Guide* on Solvnet:

https://solvnet.synopsys.com/dow_retrieve/B-2008.09/dftxg1/dftxg1.html

4.3.2 Test Model Creation

If the User is planning to create a Test Model of CORTEXM0 then it is necessary to wrap CORTEXM0 with an IEEE P1500 compliant test wrapper in addition to synthesizing the internal scan chains. In order to perform this process, the *rm_create_test_wrapper* configuration option must be set to 1.

A test model of CORTEXM0 acts as a substitute for the full netlist information enabling the protection of the core intellectual property but still allowing test related activities in a CORTEXM0 device based design.

4.4 Layout – IC Compiler

The Methodology uses a single pass layout flow. IC Compiler uses the Milkyway unified database for data management. The mapped design is imported into ICC using the DDC database as written by Design Compiler. Subsequently, ICC saves an intermediate version of the design to the Milkyway database after each layout stage. No

conversion to or from other third-party formats is required. The diagram below illustrates the steps performed during layout.

The layout flow is made up of five parts, governed by five scripts; invocation is done within the build directory by issuing the commands: **make design_planning**, **make place_opt**, **make clock_opt** and **make route_opt**, followed by **make export**.

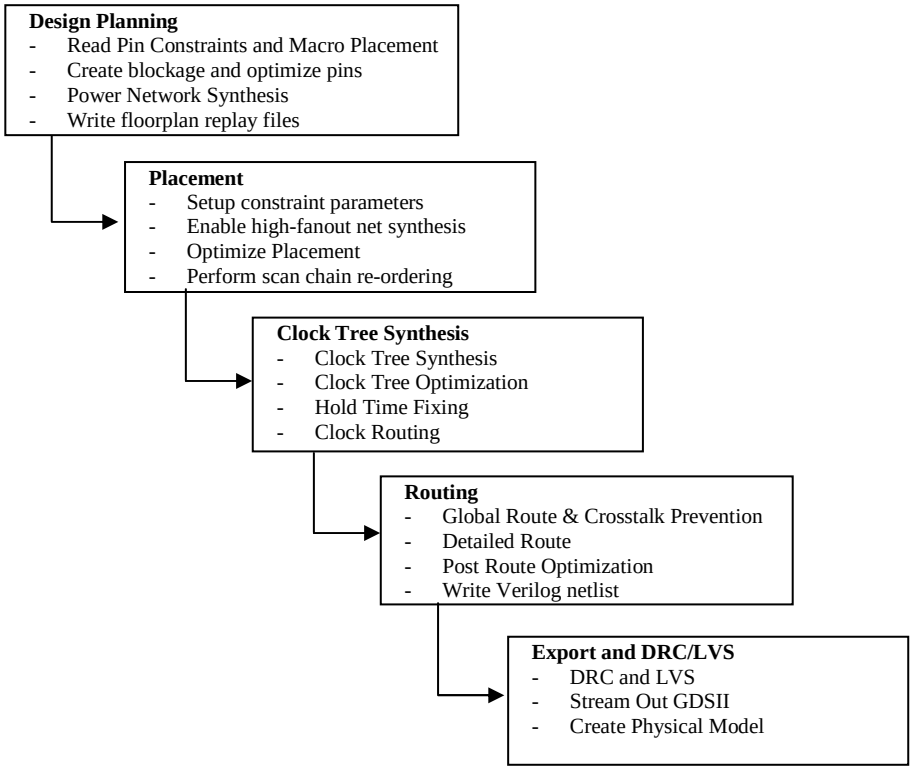


Figure 4-1: Backend flow for creating and analysing CORTEXM0

The Methodology supports concurrent Multi-mode, Multi-corner (MCMM) features throughout place and route for improved timing closure. Analysis and optimization is carried out on all scenarios concurrently, and costing is measured across all scenarios for timing and DRC. This comes at the expense of some runtime and so may be disabled with the *rm_use_mcmm* variable. Currently the flow supports 3 separate scenarios for MCMM optimization:

- Functional mode operation, at worst-case process and operating conditions
- Functional mode, at best-case process
- Test-mode operation, using best-case corner (primarily for hold fixing on scan chain paths)

In addition, a further scenario is defined specifically for use during clock tree synthesis

For more information on MCMM, please see the relevant section of IC Compiler Documentation on Solvnet:https://solvnet.synopsys.com/dow_retrieve/B-2008.09/iccug/iccug_mcmm.html

4.4.1 Design Planning

The goal of this step is to create a floorplan complete with macro placement and power supplies. Using a combination of replay files and Power Network Synthesis (PNS), the turnaround time for floorplanning can be shortened dramatically while assuring a routable design with minimal IR drop.

Two floorplan files are included as part of the Methodology: *cortexm0imp_floorplan.tcl* specifies macro placement and core dimensions, while *cortexm0imp_pins.tdf* constrains pins to the most appropriate edge. By default the floorplanning step will use these files if they exist. The files are based on the ARM-Artisan TSMC90G libraries used during script development. Should the User wish to create their own floorplan, it is recommended that the creation of these files is performed interactively. Sections of the script serve to illustrate the basic steps and inputs required, as well as some of

the assumptions that have been made with respect to the data, eg. macro placement, so that the floorplanning stage will work seamlessly.

This floorplanning flow has been arranged to allow for a certain amount of flexibility. Since floorplan requirements can vary dramatically from design to design, portions of the script will be highlighted as areas where some manual intervention may be required. Any design modifications that are made can easily be saved to the database and/or saved away as replay scripts to be used when the script is run again in batch mode. Note: the design planning stage can also be used to create physical constraints files that are read by Design Compiler. If these exist in the shipped RM bundle then they will need to be moved/deleted if the core configuration changes.

For more information on ICC Design Planning please see *Using Design Planning in the IC Compiler Flow* on SolvNet:

https://solvnet.synopsys.com/dow_retrieve/B-2008.09/iccdp/iccdp.html

4.4.2 Placement Optimization

IC Compiler is used to place and optimize the floorplanned design, based on physical information. The `place_opt` command will optimize the placement of the cells in the design within the constraints provided by the floorplan and the constraints file. Clock routing layer constraints are added to the design in this step as well. By default they will be set to double width and double spacing. If there is a requirement for other width and spacing then the values can be adjusted within the *cortexm0imp_tech.tcl* script in the "Routing Rules" section. The rules that get applied at this point stay persistent through the remaining steps of the Methodology.

Placement-based re-ordering of internal scan chains and test wrapper scan chain(s) is also supported. This process uses the SCANDEF file generated by Design Compiler, specifying the initial ordering of scan chains as created by Design Compiler.

An incremental optimization step is included for the purposes of leakage power optimization. This uses multi-threshold optimization techniques from Power Compiler to minimize the number of higher-speed, lower-threshold cells used, while maintaining the timing performance for the design.

4.4.3 Clock Tree Synthesis

Clock tree synthesis (CTS) is performed after placement optimization. IC Compiler will minimize cell disturbance ensuring that the resulting timing of the design will not change significantly.

The **`set_clock_tree_options`** command is used to direct IC Compiler to insert clock trees into the design with specific requirements. The cells that CTS uses to build and optimize the clock-tree are defined within the *cortexm0imp_tech.tcl* script.

The Methodology invokes the `clock_opt` core command to perform clock tree synthesis, hold fixing and post CTS optimization.

The `clock_opt` command does the following:

- Initial clock tree insertion and optimization without routing
- Back-annotates achieved clock latencies onto virtual clocks
- Enables hold fixing
- Post CTS optimization to include hold fixing and scan re-ordering

After initial CTS has completed the virtual clocks are updated with the internally achieved latency to improve the identification of real critical paths. The **`route_group`** command is used to perform the actual detail routing of the clock-tree nets.

If multiple clocks exist in a design these are **not** balanced during clock tree synthesis because it may be more optimal for the User to do this once the finished design is placed; for instance fewer buffers may be needed to balance a small clock domain with a larger one in the context of a larger SoC. Please see the core Configuration and Signoff Guide for the clock balancing requirements of the core. Note: if a P1500 test wrapper is implemented then the wrapper clock should be balanced with the main core clock, at least.

4.4.4 Routing

Once the clock tree is present, **`route_opt`** is used to route all non-clock wires and to optimize for setup times while also further minimizing hold violations. Some routing options are controlled by the **`set_route_options`** command. The Methodology uses this command to set track assignment to *timing driven* and to prevent any notching in routing from creating DRC's. By default the Methodology also enables crosstalk prevention using the **`set_si_options`** command.

The Methodology supports a choice of two different IC Compiler routers – Zroute and classic. Zroute is state-of-the-art routing technology that delivers a number of advantages compared to the classic router, including runtime and QoR improvements and multi-threading capabilities. Use of Zroute in the flow is optional, and is enabled using the *rm_use_zroute* variable. If Zroute is disabled then the classic router will be used.

Routing is performed in multiple stages in order to achieve the best balance between QoR and time to results.

The `route_opt` steps used are:

- Initial route only
- Optimize the routing for wire length
- Crosstalk reduction
- Hold fixing
- Cell sizing

These steps are run serially and once completed the design is fully placed and routed. A final implementation netlist and a number of reports are generated. The design can now have the parasitics extracted and the timing fully analyzed.

4.4.5 Export

This final IC Compiler step performs LVS and DRC checking on the post-route design, exports the design as GDSII format and produces a physical model (FRAM). Note that the Methodology is shipped with front end library views only and so this GDSII output may be incomplete.

4.5 Parasitic Extraction – Star-RCXT

Star-RCXT has an accurate parasitic extraction capability that can be invoked from within IC Compiler or in a stand-alone mode. The Methodology calls Star-RCXT stand-alone to generate parasitic views of CORTEXM0. By default the extraction includes cross-coupling capacitances. Scripts are provided to perform extraction at five process corners. These corners are defined as:

- *Cworst*
  - Extraction at this corner is performed using the script at *scripts/cortexm0imp_star_wc.scm*
  - This extraction corner is expected to yield the highest possible capacitance from the backend
- *Cbest*
  - Using script *scripts/cortexm0imp_star_bc.scm*
  - This corner is expected to yield the lowest capacitance from the backend
- *RCworst*
  - Using script *scripts/cortexm0imp_star_wrc.scm*
  - This corner is expected to yield the highest wire resistance, and probable low capacitance
- *RCbest*
  - Using script *scripts/ cortexm0imp_star_wrc.scm*
  - This corner is expected to yield the lowest wire resistance, and probable high capacitance
- *Typical*
  - Using script *scripts/ cortexm0imp_star_typ.scm*

- This corner is expected to yield the process median with respect to parasitics

The extraction flow can be invoked from within *build_dir* using the command: **make extract**.

The parasitic files for the design are stored in SPEF format in the *data* directory.

4.6 Equivalence Checking – Formality

Equivalence Checking is a verification procedure that can be used at many phases in a design flow. The Methodology supports verification of CORTEXM0 in an RTL to gates mode at two possible places in the flow. The reference design used is the configured compliant Verilog RTL as used for synthesis. The implementation design is a gate level netlist generated during the RM flow, and can be either the:

- Gate-level netlist from Design Compiler
- Post layout netlist from IC Compiler with the clock tree inserted

The post-layout netlist will be used as the implementation design if it exists in the *data/* directory, otherwise the post-compile netlist will be used. This script can be invoked from within *build_dir* using the command **make fm**. The Formality run writes all output reports into the *reports/lec* directory.

4.7 Timing Verification – PrimeTime-SI

Static Timing Analysis (STA) can be performed at many points in the design flow, but of particular interest in the Methodology is the use of static timing analysis on a post-layout implementation of CORTEXM0. Here, we are attempting to fully validate the timing of the implementation with respect to our initial goals. Since CORTEXM0 is fully synchronous, and adheres to good design practices, using static methods to verify the timing of the design is relatively straightforward. PrimeTime-SI is used as the delay calculation engine and static timing analysis tool, with support for exhaustive timing analysis of crosstalk and noise effects.

By default timing analysis is performed using parasitics extracted at the *cworst* process corner. This is set as the default by the rm_process_corner variable, as defined in the technology script (*scripts/cortexm0imp_tech.tcl*). To perform STA at other process corners this variable will need to be changed. Allowed values are *cworst* (the default), *rcworst, cbest, rcbest* and *typical*.

The script (*cortexm0imp_sta.tcl*) is used for static timing analysis of CORTEXM0 in functional mode using PrimeTime-SI and is invoked from within *build_dir* using the command: **make pt**.

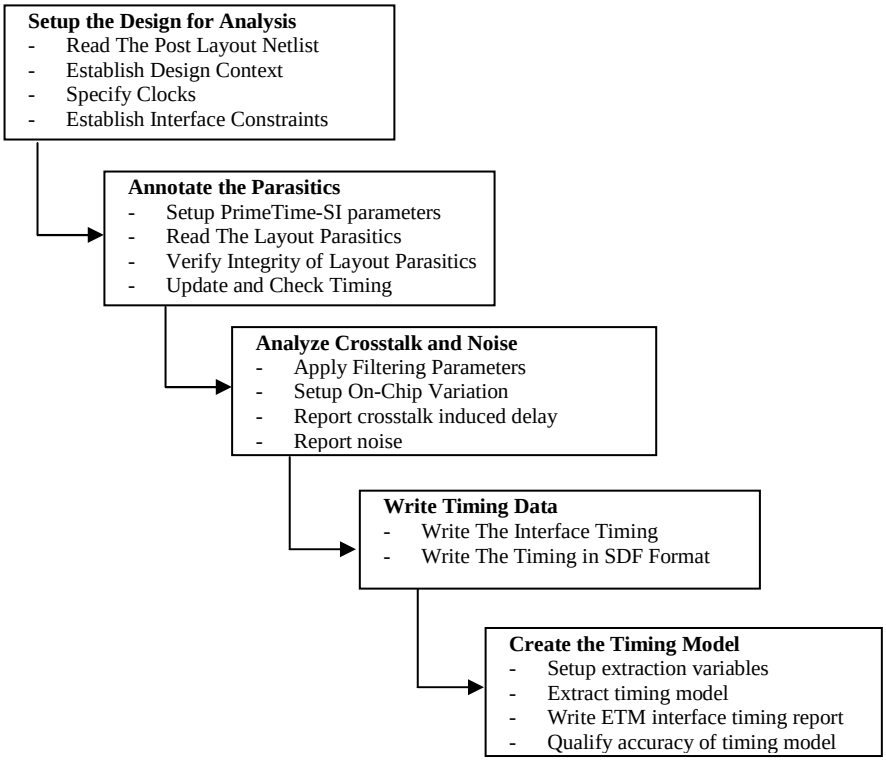


Figure 4-2: Primetime SI flow for analyzing CORTEXM0

4.7.1 Disabled timing paths

The supplied STA scripts will report any disabled timing paths to the file: *reports/sta/CORTEXM0IMP-cworst.disabled*. Paths are expected to be disabled due to the design being constrained to functional mode (SE=0), and due to constant tied logic. Example of constant tied gates include unused set and reset pins on flip-flops and unused top level output pins that will depend on RTL configuration. Examples of disabled paths for a default configured CORTEXM0 are given below:

```
Cell or Port                  From    To      Sense          Flag  Reason
-------------------------------------------------------------------------------
u_cortexm0_u_top_u_dbg_u_ctl_dfsr_dwt_reg CK SI setup_clk_rise c  SE = 0
u_cortexm0_u_top_u_dbg_u_if_slv_write_r_reg SN Q preset_low p     SN = 1
HBURST[2]                                                  p     HBURST[2] = 0
```

HMASTLOCK	**p**	**HMASTLOCK = 0**
HTRANS[0]	**p**	**HTRANS[0] = 0**
HSIZE[2]	**p**	**HSIZE[2] = 0**
HPROT[1]	**p**	**HPROT[1] = 1**
HBURST[1]	**p**	**HBURST[1] = 0**
HBURST[0]	**p**	**HBURST[0] = 0**

Note that for alternative RTL configurations, different paths may be disabled due to constant tied outputs. For more details on which ports are unused under different RTL configurations see the Integration Manual and the Configuration and Sign-off Guide.

4.7.2 Testmode Timing Analysis

Support is provided for timing analysis of the design under tester timing conditions. In this case the clocks and I/O constraints are defined based on the timing waveforms used for clocks and signals on the tester. Analysis is performed for three separate test modes:

- Scan chain shift mode
- Stuck-at faults capture mode
- At-speed capture mode

Input and output timing are relative to virtual clocks with prefixes "forcePI" and "measurePO". These clocks are impulse clocks with 0 percent duty cycles. The forcePI virtual clocks pulse at the beginning of the cycle. The measurePO clocks pulse at the earliest measure PO time. The virtual clocks are used to check activity at the point the clocks are active, to mimic a strobe on the tester.

For delay test timing analysis, a single primary clock can have clock waveforms that vary due to different waveform tables. For example, the waveform may change between the last shift cycle and the capture cycle. In this mode, the clock waveforms are defined to reflect the functional operating speed of the design, taking mark-space ratio into account. PrimeTime has some facilities to handle this situation. This involves superimposing two clock cycles on top of each other, offset by the period of the first cycle. Each cycle will have its own set of forcePI and measurePO virtual clocks.

The test mode STA script uses the *pt2tmax* utility to create path data files for use in TetraMAX. This utility script is usually located in the $SYNOPSYS/auxx/syn/tmax directory.

The files generated contain:

- Pin slack data for Small Delay Defect based transition fault testing
- Timing exceptions for use in PathDelay fault testing
- List of critical timing paths for PathDelay vector generation

The flow also generates an SDC file containing testmode timing exceptions for Small Delay Defect based transition fault testing.

A single script (*cortexm0imp_testmode_sta.tcl*) is used for test-mode timing analysis in PrimeTime-SI. The script is invoked from within *build_dir* using the command: **make pt_testmode**.

4.7.3 Distributed Multi-Scenario Analysis

Full timing verification of a design typically requires several PrimeTime analysis runs in order to verify correct operation under different operating conditions (such as extreme temperatures and operating voltages) and in different chip operating modes (mission mode, test mode, and so on). The Methodology provides support for performing multiple runs efficiently using the Distributed Multi Scenario Analysis (DMSA) capabilities available within Primetime-SI.

A specific combination of operating condition and operating mode for a given design is called a *scenario* for that design. Instead of analyzing each scenario separately, the User creates a master PrimeTime-SI process that sets up, executes, and controls multiple slave processes, one for each scenario. The User can distribute the processing of scenarios onto different hosts running in parallel, thus reducing the overall turnaround time to finish analysis of all scenarios. In addition, total runtime is reduced when one shares common data between different scenarios.

For more information, refer to the relevant section of the *Primetime User Guide* on Solvnet:

https://solvnet.synopsys.com/dow_retrieve/B-2008.06/ptms/ptms.html

A DMSA master script (*cortexm0imp_dmsa.tcl*) is used to execute multi-scenario analysis of CORTEXM0 in PrimeTime-SI; some editing of this script may be required to enable correct operation on your compute cluster. The analysis is invoked from within *build_dir* using the command: **make dmsa**.

4.8 Timing Model Generation – PrimeTime

The **`extract_model`** command in PrimeTime is used to extract a timing model of the design. PrimeTime uses a number of variables to control the extraction of the timing model and typically these variables trade off accuracy of the timing model against the speed of creation, but others control the nature of the resulting model and its applicability to other phases of the design flow.

It is important to note that the resulting timing model of CORTEXM0 is an abstraction of the original design. Since the netlist is no longer available, the level of detail available to PrimeTime (or any delay calculator) is reduced. This reduction in detail compromises the accuracy that can be achieved with the model. The simpler the model, the less accurate it will be. The compromise between accuracy and model simplicity is a trade off that the User must make. The accuracy of the generated model is checked during qualification. By default, timing arcs and, input caps and output transitions are checked to be within 20% absolute or +/-0.05 library units of the source netlist. This can be altered to match your signoff requirements by changing the '*–absolute_tolerance*' and '*–percent_tolerance*' options of the command '*compare_interface_timing*' in the qualification script (*cortexm0_qual.tcl*).

Two scripts are used for the timing model creation (*cortexm0imp_etm.tcl*) and qualification (*cortexm0imp_qual.tcl*). These scripts are invoked from within *build_dir* using the command: **make etm**. Reports are output into the *reports/sta* directory and models into *models/timing*.

4.9 Test Pattern Generation – TetraMAX

Automatic test pattern generation is performed using TetraMAX, a high-speed, high-capacity, automatic test pattern generation (ATPG) tool, to generate test patterns that maximize the test coverage of CORTEXM0 using a minimum number of test vectors.

TetraMAX offers three different ATPG modes:

- Basic Scan
- Fast-Sequential ATPG
- Full-Sequential ATPG

TetraMAX supports test pattern generation for five types of fault models, in the following order:

- Path delay faults
- Small Delay Defect based transition delay faults
- Stuck-at faults
- Bridging faults
- IDDQ faults

The Methodology supports all three modes of TetraMAX (where applicable) with all five types of fault model for regular scan, compressed scan and/or wrapped designs. The TetraMAX flow is invoked from within *build_dir* using the command **make tmax**. Reports are output into the directory *reports/dft* and patterns, etc are output to *data*. This step should be run after Testmode timing analysis.

Note: By default, the STIL Protocol File (SPF) generated by DFT Compiler, and used by TetraMAX, is targeted for Stuck-At Fault testing. This file requires edits in order to support At-Speed fault tests (Transition and Path Delay faults). For more information refer to the article about STIL file anatomy on Solvnet:

https://solvnet.synopsys.com/retrieve/012874.html

4.10 Power Analysis – PrimeTime-PX

In addition to analyzing the power dissipation of CORTEXM0 in PrimeRail, the User can use PrimeTime-PX for this. PrimeTime-PX is a dynamic gate-level analysis tool that accurately analyzes power dissipation of cell-based designs. PrimeTime-PX generates a power profile using:

- Design connectivity: Verilog netlist
- Design switching activity: VCD file or toggle-rate estimate (default)
- Pin-to-pin delay information: Synopsys libraries
- Synopsys library power table: Standard cell library characterized for power
- Net capacitance and resistance parasitics from Star-RCXT

Power analysis is performed by issuing the command **make ptpx** in *build_dir*.

4.11 Rail Analysis – PrimeRail

The Methodology includes support for analysis of the power mesh within CORTEXM0 using PrimeRail. PrimeRail is a complete power rail analysis solution, providing early analysis as well as sign-off accuracy. PrimeRail also provides the User with the capability of creating a power grid model (White-Box-Model – WBM) for SoC analysis.

In order for PrimeRail to accurately analyze the power mesh of CORTEXM0, some power information must be provided. Specifically the definitions of the power supplies and the net switching activity must be made available to PrimeRail. Power consumption data must also be provided for hard macros which are missing WBMs and CONN views. This data is provided in three scheme files called during rail analysis. The location of these side files is defined by variables in the technology script:

```
set rm_supplies_file  "../scripts/cortexm0imp_powerSupply.scm"
set rm_activity_file  "../scripts/cortexm0imp_netSwitching.scm"
set rm_mempower_file  "../scripts/cortexm0imp_mempower.scm"
```

The parasitic data coupled with the instance switching activity is used in the calculation of the cell instance power. WBMs are created for both the power and ground nets. WBMs include parasitic data as well as the current distribution of the specific power or ground net. Combined with the FRAM view of a cell, a white box model provides a power grid model for performing rail analysis at the SoC level while maintaining a high level of obfuscation. For more information on creating these views see the Milkyway and PrimeRail User Guides:

https://solvnet.synopsys.com/dow_retrieve/B-2008.09/mwug/mwug.html

https://solvnet.synopsys.com/dow_retrieve/B-2008.09/prug/prug.html

Rail analysis is performed by issuing the command: **make rail** in *build_dir*