Dolphin Technology

Soft IP

RTL Design and Verification

Training Plan

Revision 0.1.1

1 August 2023

Training Plan

Soft IP

RTL Design and Verification

Revision 0.1.1

1 August 2023

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

REVISION HISTORY

Revision	Date	Description of Changes
0.1.0	18 Apr 2023	Original Release
0.1.1	01 Aug 2023	Added 3. Kiểm Thử Thiết Kế Số
0.1.2	08 Apr 2024	Changed 2.2.3.2. Bài tập thực hành số 3 (dti_uart)

1. Quy Trình Làm Việc

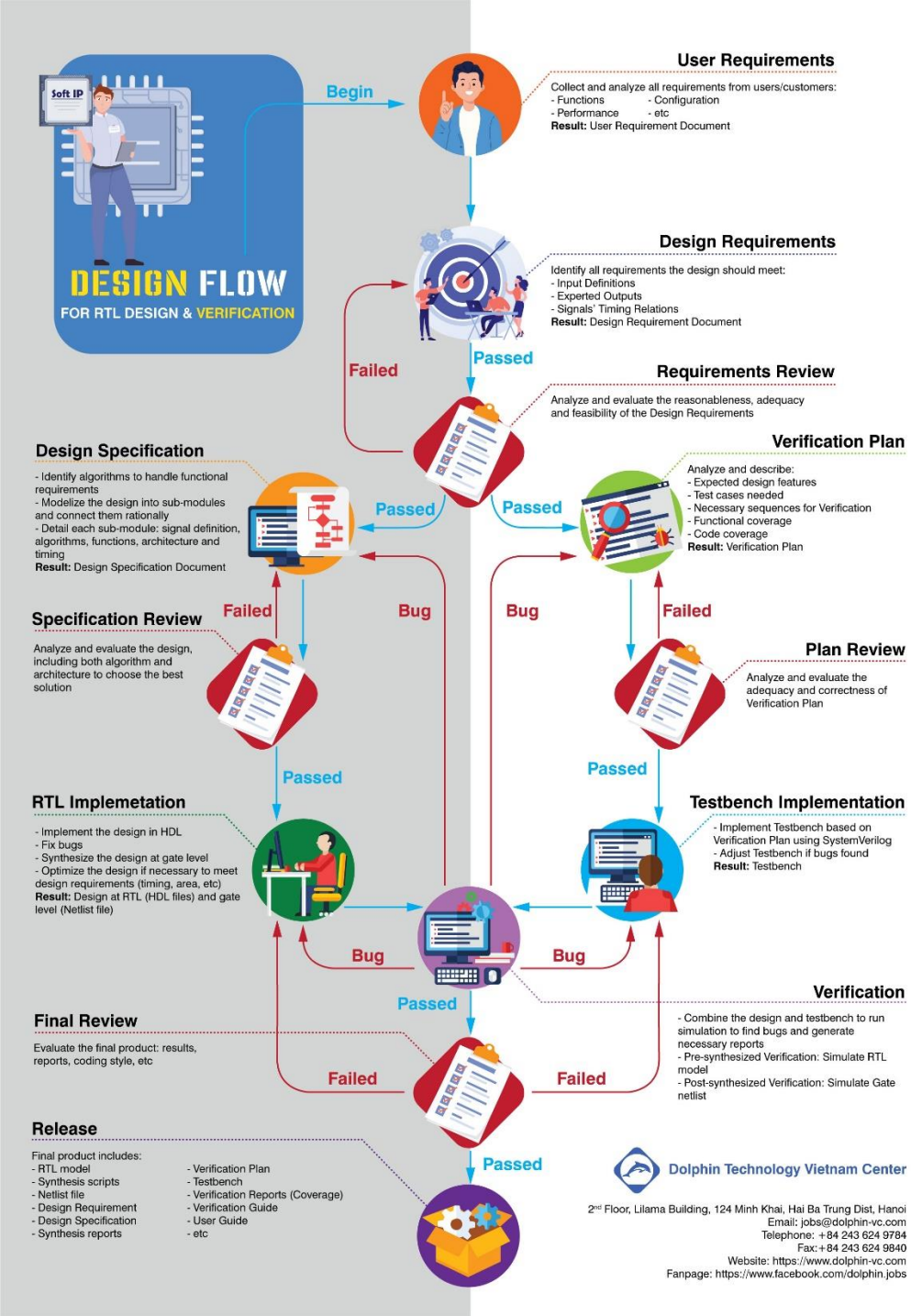

Figure 1. Quy trình thiết kế và kiểm thử thiết kế IC số

Quản lý thư mục Project

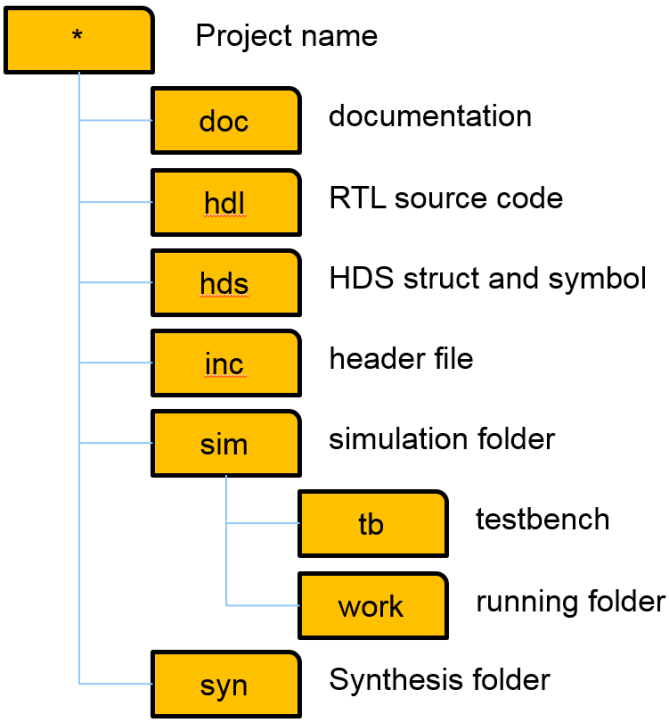

Figure 2. Quản lý thư mục project

2. Thiết Kế Số

2.1. Kỹ Năng

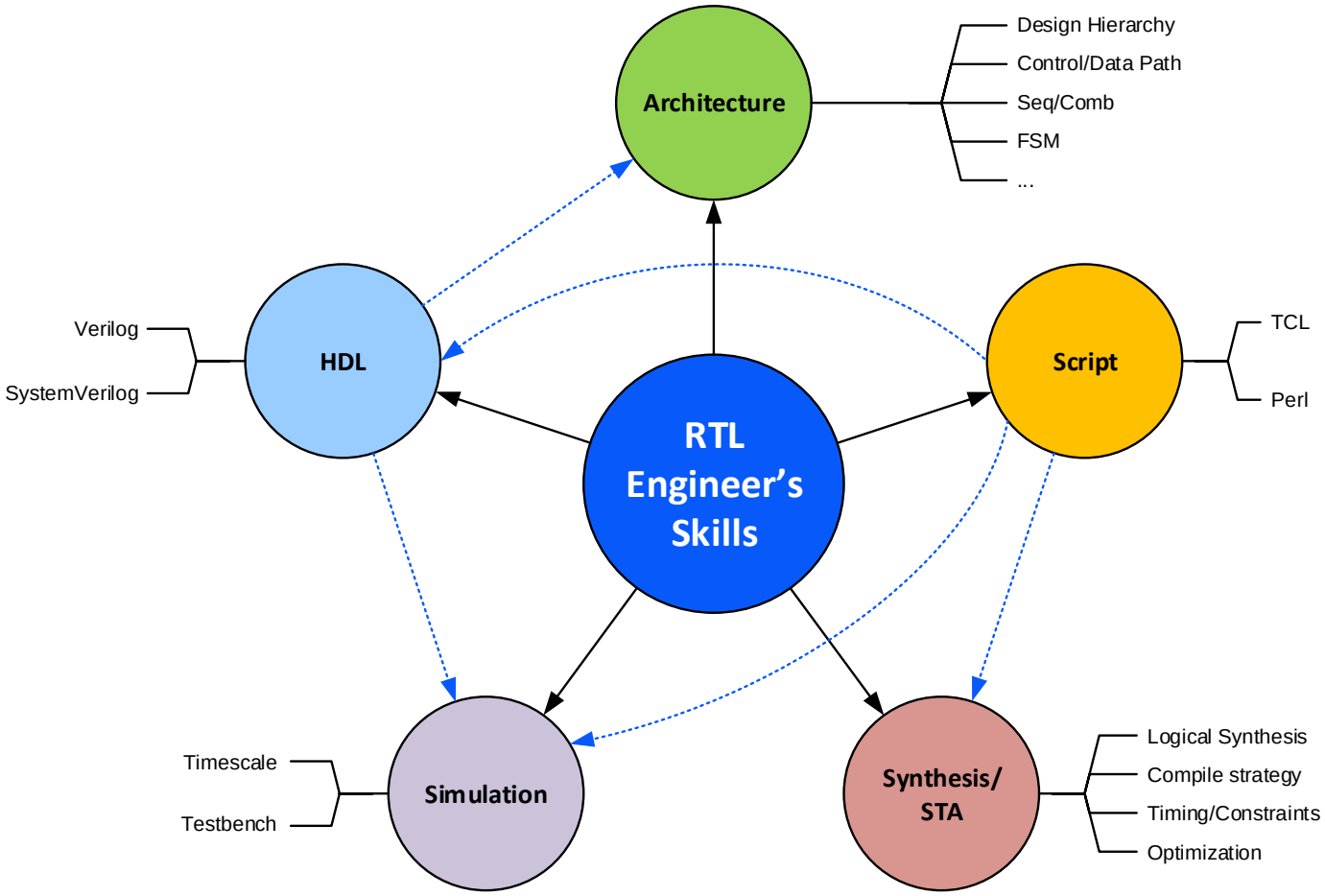

Figure 3. Kỹ năng Thiết kế số

Các kỹ năng cần thiết (được đào tạo) của Kỹ sư thiết kế số:

- Architecture: Thiết kế kiến trúc IP
- HDL: Sử dụng ngôn ngữ mô tả phần cứng để thiết kế IP
- Simulation: Mô phỏng và kiểm thử thiết kế ở mức độ đơn giản
- Synthesis/STA: Tổng hợp thiết kế và đánh giá timing
- Script: Tự động hóa quá trình làm việc bằng việc viết và sử dụng các script

2.2. Nội Dung Đào Tạo

Thời lượng dự kiến:

2.2.1. Tuần 1 + 2

Mục tiêu:

- Ôn lại kiến thức Toán Logic và Điện tử số
- Tìm hiểu về mạch tổ hợp và mạch dãy
- Tìm hiểu và sử dụng ngôn ngữ mô tả phần cứng SystemVerilog
- Làm quen với Design Flow và thiết kế 1 IP đơn giản

2.2.1.1. Buổi 1 – Đào tạo

Nội dung giới thiệu:

- Ôn lại hệ cơ số, số không dấu và số có dấu
- Ôn tập toán logic (Đại số Boolean)
- Mạch logic tổ hợp và Mạch dãy
- Reset Đồng bộ và Reset Không đồng bộ
- Cách mô tả mạch bằng ngôn ngữ SystemVerilog
- Phân biệt Blocking và Non-Blocking

Nội dung tự tìm hiểu:

- Design Hierarchy
    - module declaration
    - module instantiation
    - real circuit
- Signal, parameter, port
    - Chức năng
    - Định dạng
    - type
        - net
        - variable
    - data type
        - 2-state
        - 4-state

  - array
    - packed
    - unpacked
  - port type
    - input
    - output
    - inout
  - types và data types của tín hiệu
    - wire
    - logic
    - reg
  - Cách sử dụng
  - Mạch thật
- Testbench đơn giản, timescale
- Procedural blocks
  - always
  - sensitivity lists
  - always_comb, always_ff, always_latch
  - Cách sử dụng
  - So sánh
  - Mạch thật
- Operation, liên hệ với các phép toán logic
  - statement, condition, expression, operation, operator, operand
  - bitwise vs logical
  - increment and decrement operators
  - assignment operators
  - Cách sử dụng
  - Tổng hợp được hay không?
- Function
  - format
  - type
  - static

  - automatic
  - void
  - function vs task
  - Cách sử dụng
  - Mạch thật
- Race Condition
  - Hardware
  - Software

2.2.1.2. Buổi 2 – Đào tạo

Nội dung giới thiệu:

- Cách viết Specification (Bản đặc tả thiết kế)

2.2.1.3. Bài tập thực hành số 1

Design **dti_updown_count** block: Up/Down Counter.

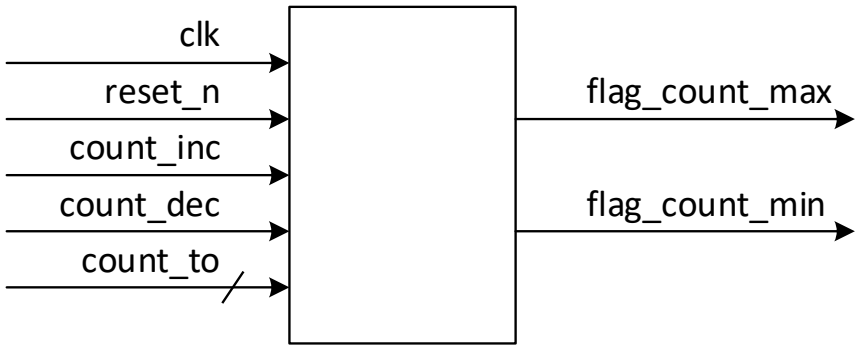

Figure 4. dti_updown_count

Table 1. dti_updown_count Signal Description

Signal Name	Width	I/O	Description
clk	1	Input	DTI Clock Signal
reset_n	1	Input	DTI Asynchronous Reset, active LOW
count_to	COUNT_WIDTH	Input	Indicates the maximum value of the counter
count_inc	1	Input	Increases the counter value
count_dec	1	Input	Decreases the counter value
flag_count_max	1	Output	Indicates the counter reaches maximum value (count_to)
flag_count_min	1	Output	Indicates the counter reaches minimum value (0)

Function descriptions:

- Asynchronous Reset.
- Count-range: 0 ÷ count_to.
- During reset, counter value is 0.
- Value of parameter: COUNT_WIDTH = 2÷128 (Default value: 8)
- Function of count_inc and count_dec

Table 2. Function of count_inc and count_dec

count_inc	count_dec	counter value
0	0	not change
0	1	-1
1	0	+1
1	1	not change

- flag_count_min asserts when the counter value equals to 0.
- flag_count_max asserts when the counter value equals to “count_to”.

2.2.1.4. Buổi 3 – Review

Review bài tập thực hành và nội dung tự tìm hiểu

2.2.2. Tuần 3

Mục tiêu:

- Thiết kế FSM
- Sử dụng case

2.2.2.1. Buổi 1 – Đào tạo

Nội dung giới thiệu:

- Các loại FSM, cấu tạo FSM, sơ đồ chuyển trạng thái
- Cách mã hóa trạng thái

Nội dung tự tìm hiểu:

- Các loại case
- Cách mô tả FSM bằng SystemVerilog
- Mô phỏng và tổng hợp các loại case

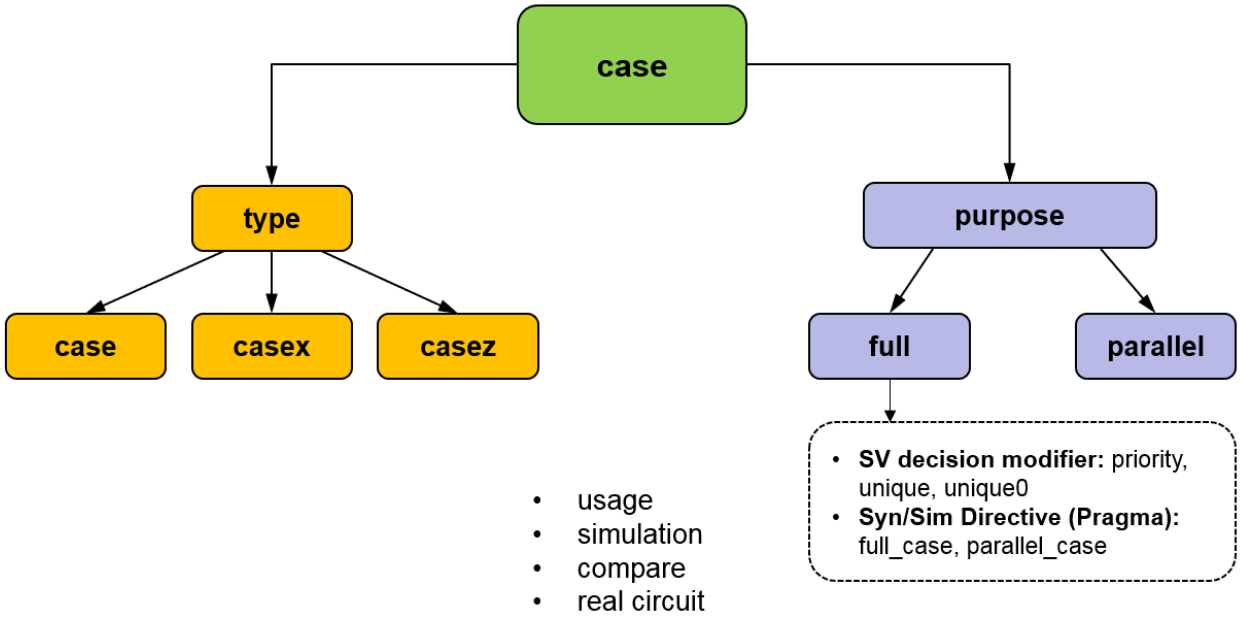

Figure 5. Các loại case

2.2.2.2. Bài tập thực hành số 2

Design **dti_seq_dect_fsm** block: Detects a pattern from a bit-stream

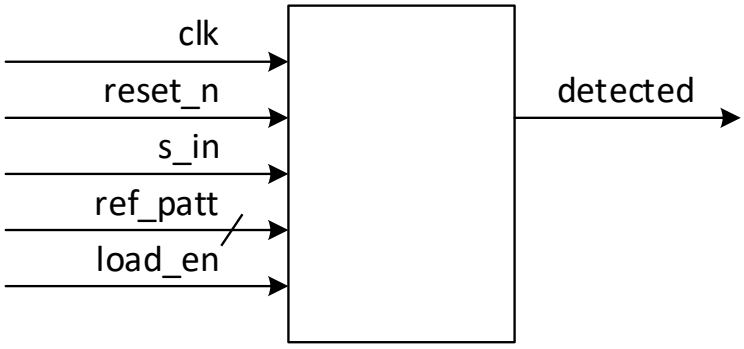

Figure 6. dti_seq_dect_fsm

Table 3. dti_seq_dect_fsm Signal Description

Signal Name	Width	I/O	Description
clk	1	Input	DTI Clock Signal
reset_n	1	Input	DTI Asynchronous Reset, active LOW
s_in	1	Input	Bit-stream input
ref_patt	4	Input	Reference Pattern
load_en	1	Input	Loads new reference pattern into the detector
detected	1	Output	The flag asserting when the pattern is detected from the bit-stream

Functional Descriptions:

- A random bit-stream is sent to s_in of the detector. When a pattern defined by ref_patt is found in the stream, the flag “detected” will assert to HIGH. Otherwise, “detected” is LOW.
- load_en:
  - load_en = 1’b1: Loads new reference pattern into the detector.
  - load_en = 1’b0: Compares the bit-stream with the reference pattern.

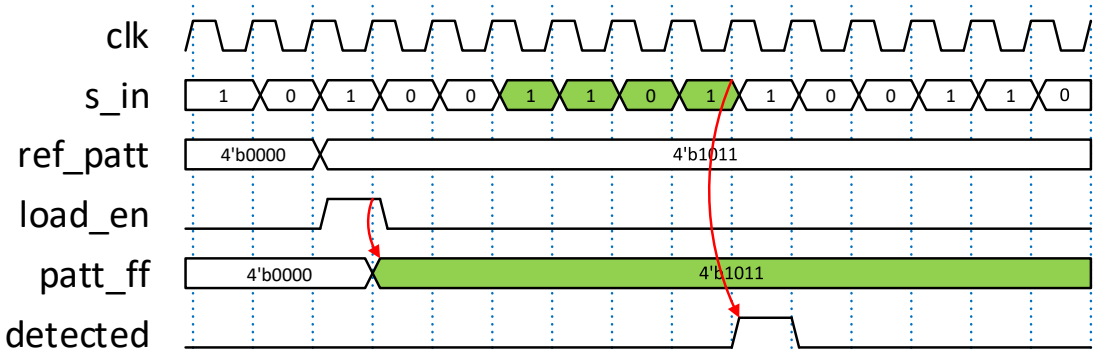

Figure 7. dti_seq_dect_fsm Timing Diagram

2.2.2.3. Buổi 2 – Review

Review bài tập thực hành và nội dung tự tìm hiểu

2.2.3. Tuần 4 + 5

Mục tiêu:

- Static Timing Analysis
- Tổng hợp mạch sử dụng Design Compiler

2.2.3.1. Buổi 1 – Đào tạo

Nội dung giới thiệu:

- Timing
  - Delay
  - Skew
  - Slew
  - Jitter
  - Setup/Hold Time

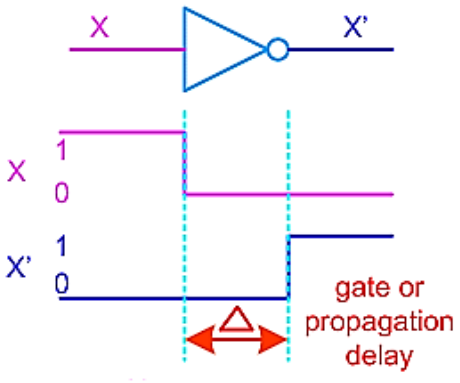

Figure 8. Gate Delay

Figure 9. Wire Delay

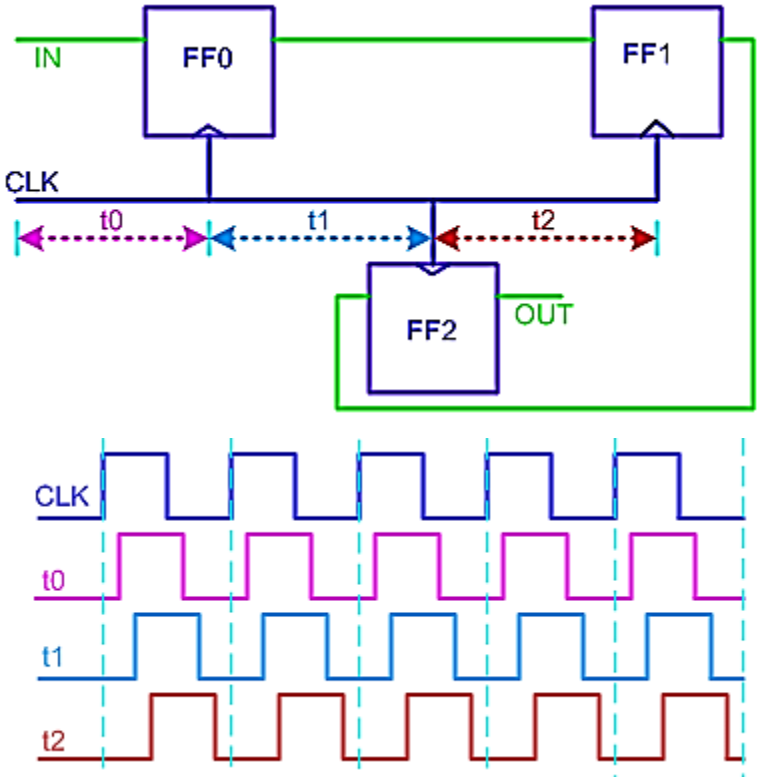

Figure 10. Skew

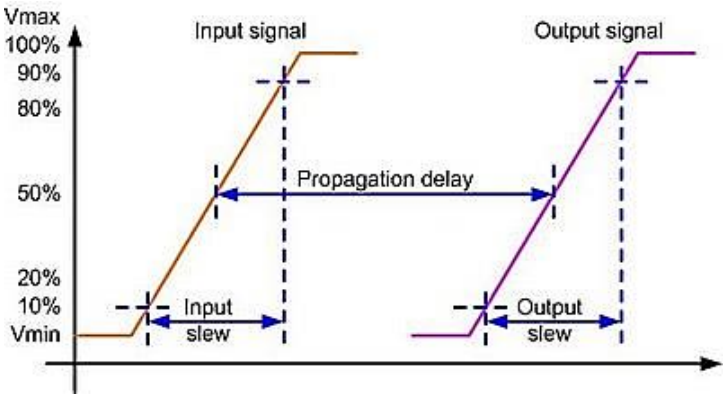

Figure 11. Slew (Transistion Time)

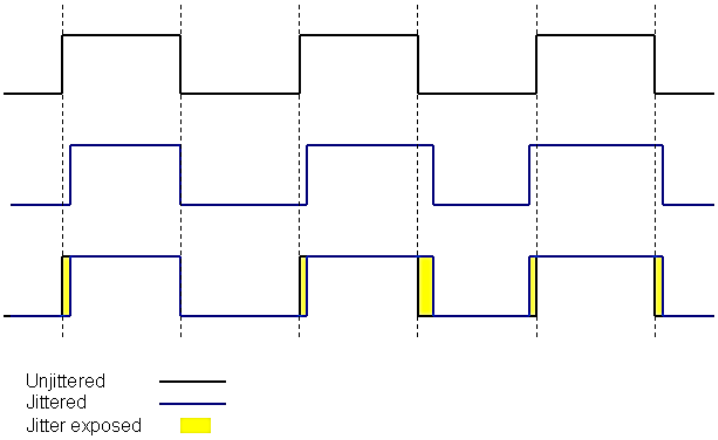

Figure 12. Jitter

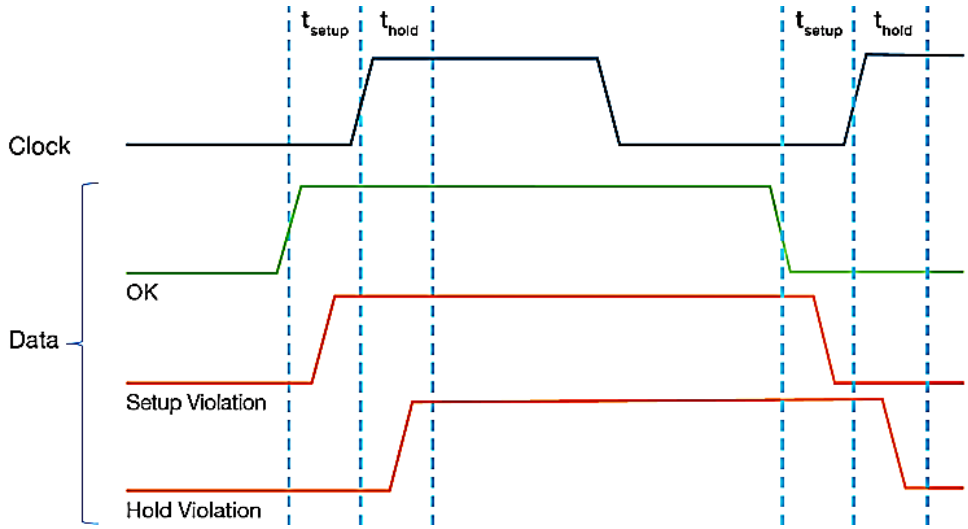

Figure 13. Setup/Hold Time

- Timing Paths

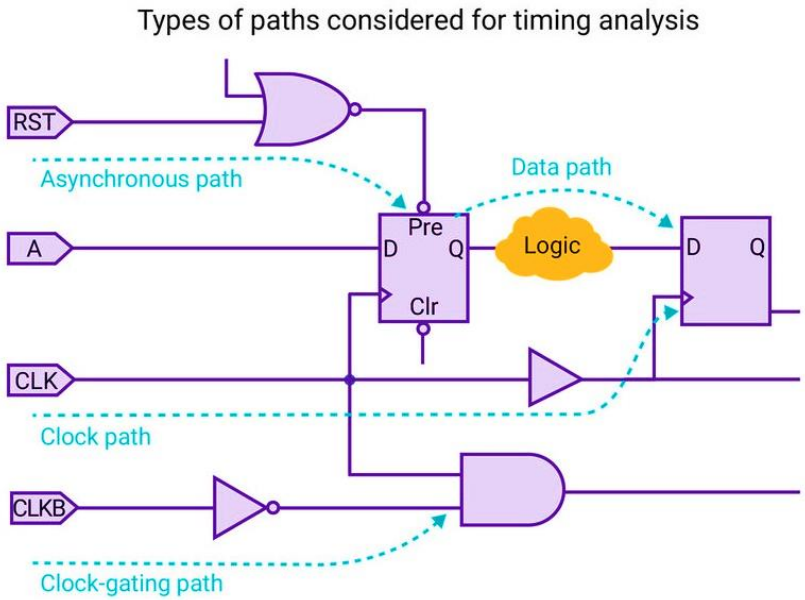

Figure 14. Timing Paths

- Data paths

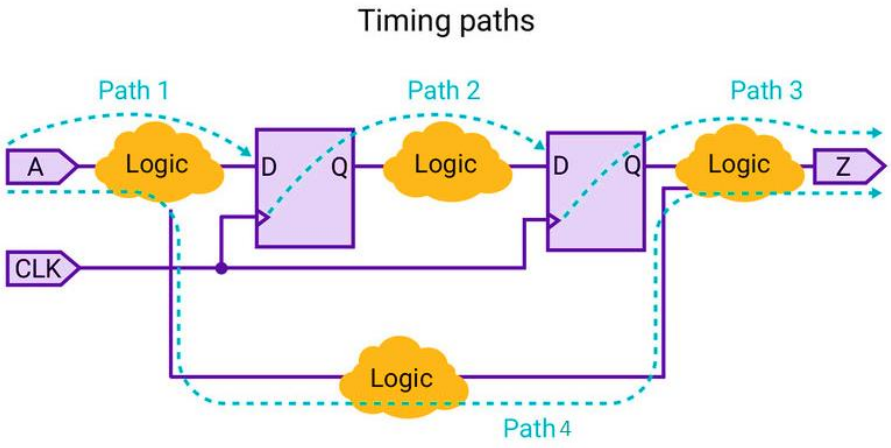

Figure 15. Data Paths

  - Critical path
  - Arrival time, Required time, Slack
  - How to fix timing? Area vs Speed?
- Tổng hợp mạch bằng Design Compiler

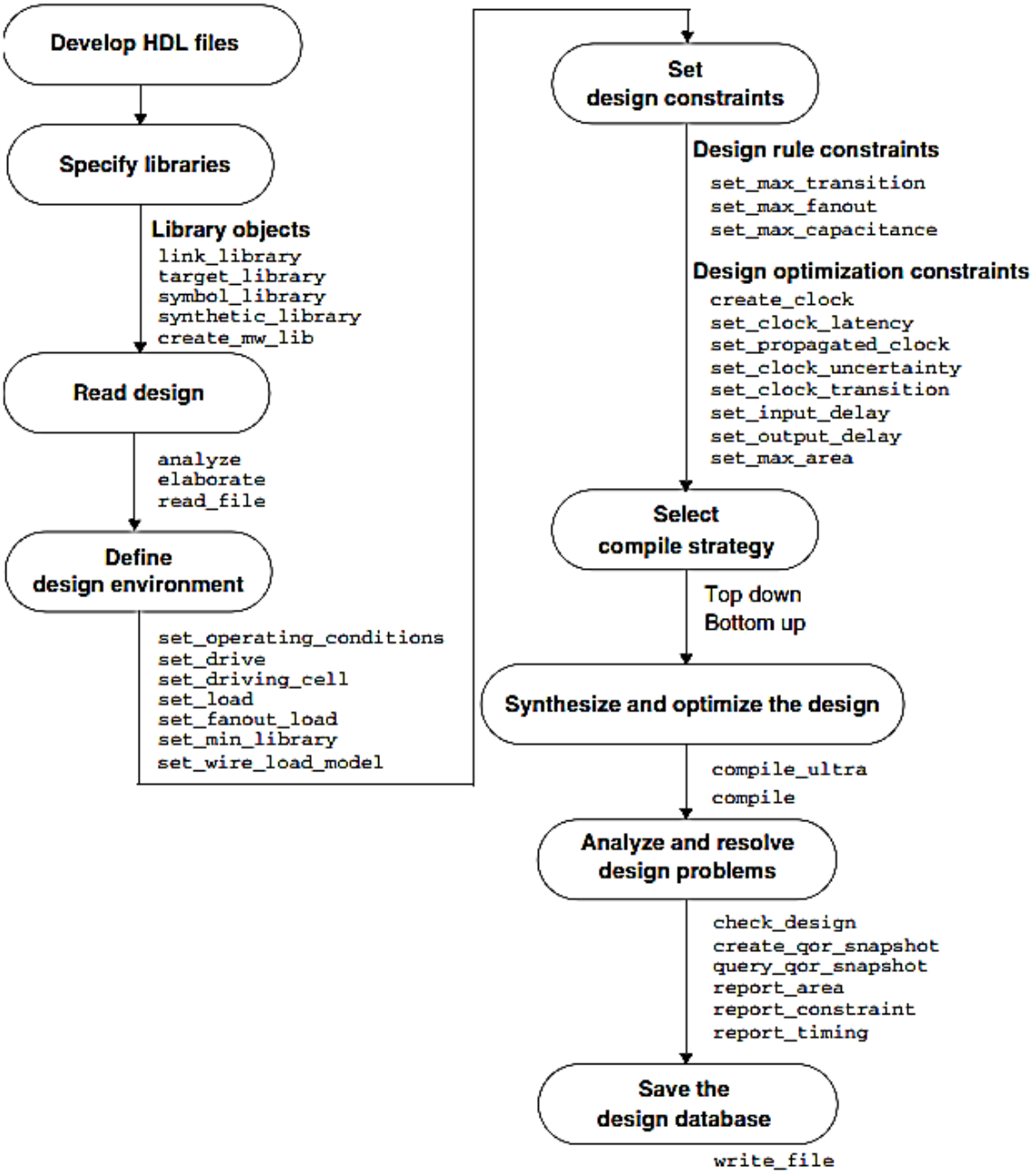

Figure 16. Synthesis Flow

Nội dung tự tìm hiểu:

- Viết TCL Script để chạy tổng hợp mạch
- Tự tổng hợp các IP đã thiết kế

2.2.3.2. Bài tập thực hành số 3

Design **dti_uart** Simplified Universal Asynchronous Receiver/Transmitter (UART) Controller

Description:

dti_uart is a simplified UART APB controller performing two main tasks below:

- Converts serial data received from a peripheral device to parallel data sent to the host (CPU)
- Converts parallel data received from the host to serial data sent to a peripheral device

The host can read the UART operation status at any time.

Requirements:

- Supports baud rate values: 2400, 4800, 9600, 19200, 38400, 57600, 76800, 115200, 230400 (parameter configuration)
- Supports clear-to-send and request-to-send handshaking signal (autoflow control)
- Programmable UART frame (registers configuration using APB protocol), including:
  - The number of data bits: 5 to 8
  - The number of parity bits: 0 or 1
  - The type of parity (if used): odd or even
  - The number of stop bits: 1 or 2
- Operation control using APB protocol

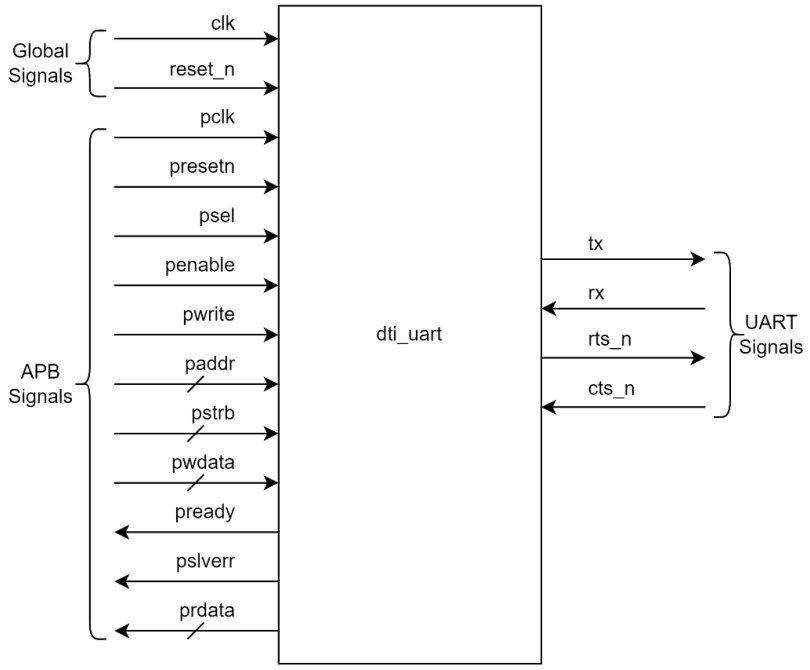

Figure 17. dti_uart block diagram

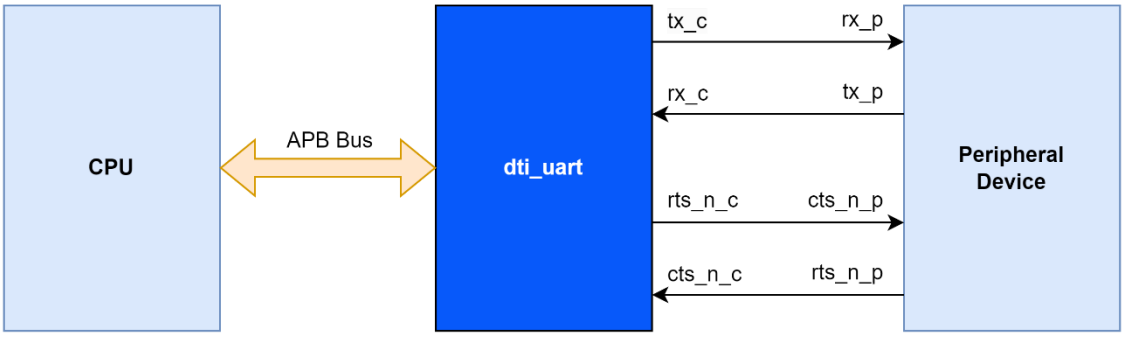

Figure 18. System using dti_uart

Table 4. dti_uart parameter description

Parameter	Value	Description
BAUDRATE	`CFG_BAUDRATE Default: 115200	Baud rate value

Table 5. dti_uart signals description

Signal Name	Width	I/O	Description
Global Signals			
clk	1	Input	System clock signal
reset_n	1	Input	System asynchronous reset, active LOW
APB Completer Signals			
pclk	1	Input	APB clock signal
presetn	1	Input	APB reset signal, active LOW, connected directly to the system reset
psel	1	Input	Select, indicates that the APB completer is selected and that a data transfer is required
penable	1	Input	Enable, indicates the second and subsequent cycles of an APB transfer
pwrite	1	Input	Direction, indicates an APB write access when HIGH and an APB read access when LOW
pstrb	4	Input	Write strobe, indicates which byte lanes to update during write transfer
paddr	12	Input	APB address bus
pwdata	32	Input	APB write data
pready	1	Output	Ready, is used to extend an APB transfer by the completer
pslverr	1	Output	Transfer error
prdata	32	Output	APB read data
UART Signals			
rx	1	Input	Serial data receive
cts_n	1	Input	Clear-to-send handshaking signal
tx	1	Output	Serial data transmit
rts_n	1	Output	Request-to-send handshaking signal

Functional Description:

dti_uart works base on its registers block which provides information about data that need processing, UART frame configuration, control signals and status signals, for more detail, refer to Table 6. The host can control and monitor operation of module by accessing its registers block through APB protocol. **dti_uart** would use the value of registers programmed by the host to perform appropriate data transmission/reception process with peripheral device through UART protocol. Each UART transaction would transmit/receive one character whose bitwidth is equal to the number of data bits configured. Figure 18 depicts the way how **dti_uart**, peripheral device and the host can be connected. Control flows of module are shown in Figure 19 and Figure 20.

Note:

- Because of simplification reason, system clock signal and APB clock signal would use the same frequency.
- In case the number of data bits is smaller than **tx_data**/**rx_data** bitwidth, most significant bits of **tx_data**/**rx_data** would be masked. For example, when the number of data bits is 5, only **tx_data**[4:0]/ **rx_data**[4:0] are valid.

Registers Block

Table 6. dti_uart's register block description

Register Name	Register Description	Register SW Access	Field Name	Field Reset Value	Position	Field Description
tx_data_reg	TX data register	RW	tx_data	0	[7:0]	Parallel data from the host which is converted to serial data sent to peripheral device
			rfu	0	[31:8]	Reserved for future use
rx_data_reg	RX data register	RO	rx_data	0	[7:0]	Parallel data archived after serial-to-parallel conversion on data received on peripheral device
			rfu	0	[31:8]	Reserved for future use
cfg_reg	UART frame configuration register	RW	data_bit_num	0	[1:0]	The number of data bits
			stop_bit_num	0	[2]	The number of stop bits
			parity_en	0	[3]	Parity enable
			parity_type	0	[4]	Parity type select
			rfu	0	[31:5]	Reserved for future use
ctrl_reg	Operation control Register	RW	start_tx	0	[0]	Set to start converting parallel data received from the host to serial data sent to peripheral device
			rfu	0	[31:1]	Reserved for future use
stt_reg	Operation status register	RO	tx_done	1'b1	[0]	Set to indicate that previous parallel-to-serial conversion on data received from the host is completed
			rx_done	0	[1]	Set to indicate that previous serial-to-parallel conversion on data received on peripheral device is completed
			rfu	0	[31:2]	Reserved for future use

Control Flow

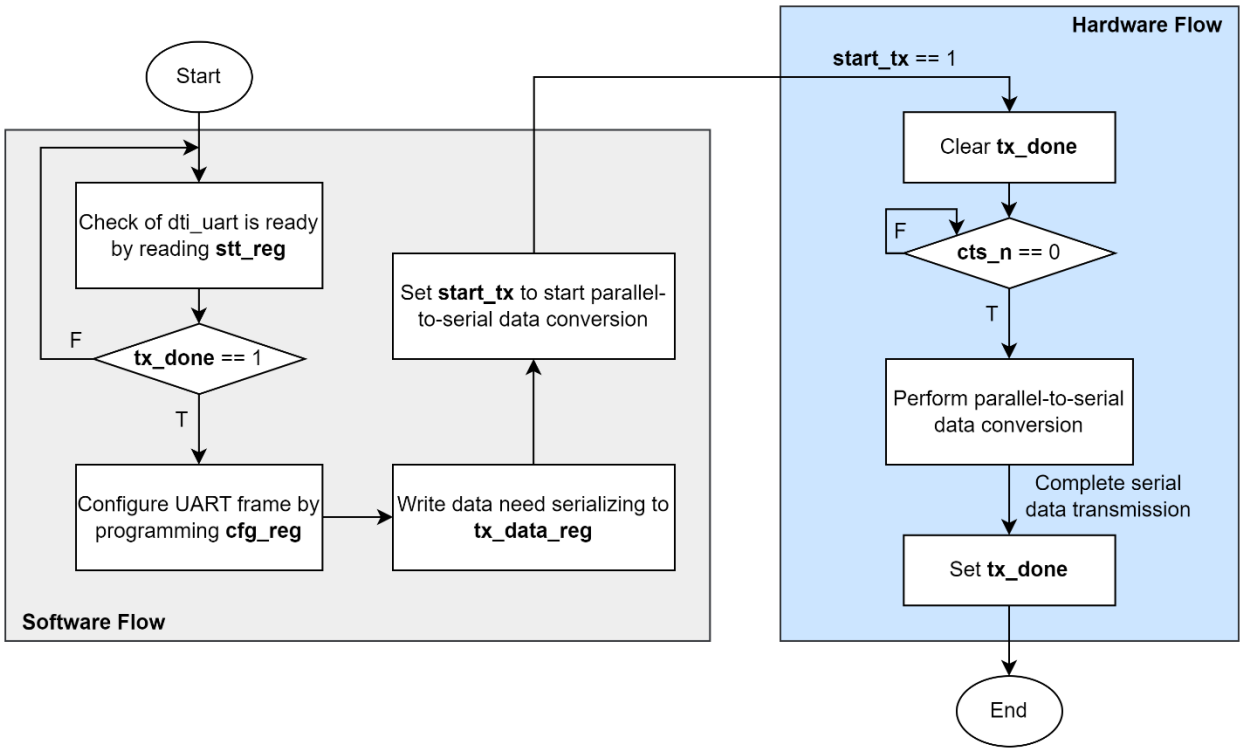

Figure 19. Parallel-to-serial data conversion control flow

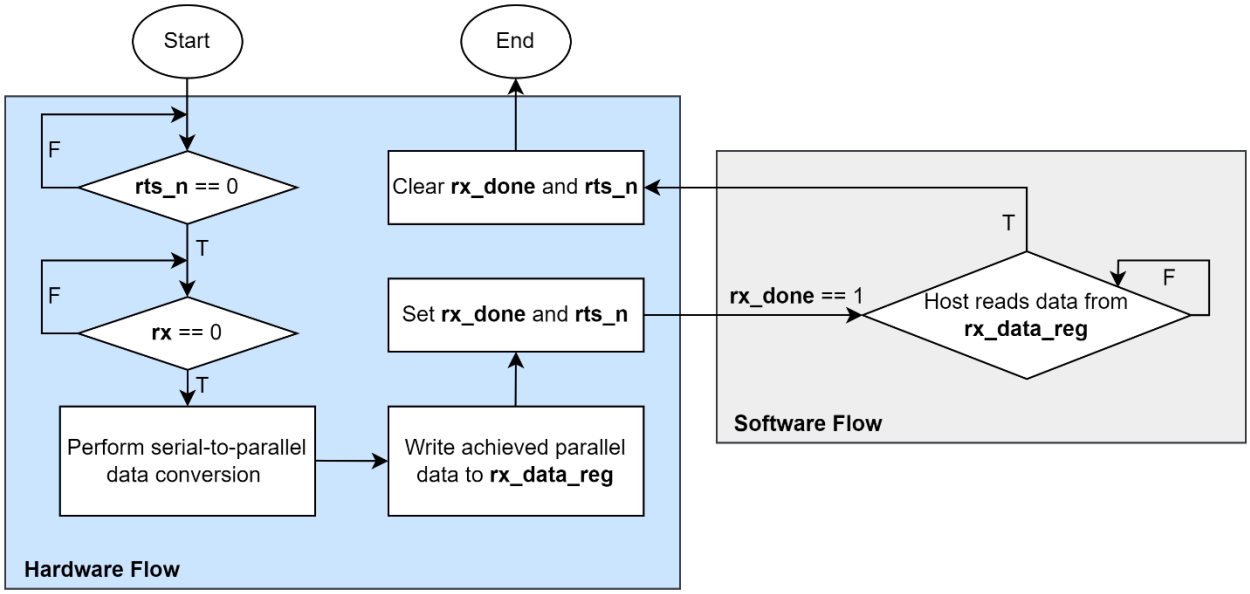

Figure 20. Serial-to-parallel data conversion control flow

2.2.3.3. Buổi 2 – Review

Review bài tập thực hành và nội dung tự tìm hiểu

2.2.3.4. Kiểm tra và đánh gia kết quả đào tạo

3. Kiểm Thử Thiết Kế Số

3.1. Tuần 1

Nội dung giới thiệu:

- What is a Testbench?
- Verification Methodology
  - ASIC design
  - Functional verification need
  - Test bench
  - Linear test bench
  - Linear random test bench
  - Does a design work and how is it done?
  - Code coverage
  - Statement coverage
  - Conditional coverage
  - Brand coverage
  - FSM coverage
  - Functional coverage
  - Coverage driven constraint random verification architecture
  - Verification plan and phases of verification

Nội dung tự tìm hiểu:

- Data Types
  - New Data types: logic, bit
  - Signed integers, byte
  - Strings
  - Enumeration
  - Fixed-size arrays
    - Declaration and initiation
    - Operations: for, foreach, copy and compare
    - Packed and unpacked arrays
  - Dynamic Arrays
  - Associative Arrays
  - Array Manipulation Methods
  - Queues

- Structures
- User-defined Data Types

- Control Flow
  - Loops
    - while/do-while loop
    - foreach loop
    - for loop
    - forever loop
    - repeat loop
  - break, continue
  - if-else-if
  - case
  - Blocking & Non-blocking
- Statements
  - Events
  - Functions and void function
  - Tasks
- Processes
  - SystemVerilog Threads
    - fork join
    - fork join_any
    - fork join_none
  - Disable fork join
  - Wait fork
- Communication
  - Interprocess Communication
  - Semaphores
  - Mailboxes
- Interface
  - Interfaces
  - Introduction
  - Interface bundles

- Modports
- Clocking Blocks
- Clocking Blocks II

Questions:

- What is the main function of a test bench?
- When you can stop functional verification process of a design?
- What data type can be used with bitwise operation?
- Compare the flexibility, size, speed and sorting performance between fixed-size array, dynamic array, queue and associative array.
- Distinguish packed types and unpacked types
- Differentiate between typedef and `define
- Be aware of advantages and disadvantages of implicit net
- Differentiate between task and function
- Distinguish non-blocking assignment and blocking assignment
- Differentiate between static and global variable

References:

- http://testbench.in/TS_00_INDEX.html
- https://www.chipverify.com/systemverilog/systemverilog-tutorial
- Springer – System Verilog for Verification (3nd)
- Professional verification – A guide to advanced functional verification

3.2. Tuần 2

Nội dung giới thiệu:

- Basic OOP and connecting test bench to the DUT
- What are classes?
- How can I access signals within a class?
- How do I create an array of classes?
- Class Handles and Objects
- Constructors

- Randomization
  - What are direct tests?
  - What are randomized tests?
  - Random variables
  - Constraint blocks
  - The pre_randomize and pos_randomize functions

Nội dung tự tìm hiểu:

- Basic OOP
  - this pointer
  - super keyword
  - typedef forward decl.
  - Inheritance
  - Polymorphism
  - Virtual Methods
  - Static Variables/Functions
  - Shallow/Deep Copy
  - Parameterized Classes
  - extern keyword
  - Access Qualifier : local
  - Abstract Class/Pure Methods
  - Packages
- Randomization
  - Array Randomization
  - Common Constraints
  - inside constraint
  - Implication Constraint
  - foreach Constraint
  - solve before Constraint
  - Static Constraints
  - Practical Constraint Examples
  - Bus Protocol Constraints

- Randomization Methods
- In-line Constraints
- Soft Constraints
- Disable Constraints
- Disable Randomization
- Random Weighted Case

Questions:

- Distinguish class type, object, handle and variable
- Differentiate between public and private attributes
- Differentiate between class and module
- What is the main function of random constraint?
- How many ways to randomize an available within a specific range, advantages and disadvantages of each?
- Explain how to solve a random constraint

References:

- https://www.chipverify.com/systemverilog/systemverilog-tutorial
- Springer – System Verilog for Verification (3nd)
- Professional verification – A guide to advanced functional verification

3.3. Tuần 3

Nội dung giới thiệu:

- System Verilog Assertion
  - Introduction
  - Immediate Assertion
  - Concurrent Assertion
  - $rose, $fell, $stable
  - Assertion Time delay ##
- SystemVerilog Coverage
  - Code coverage
  - Function coverage
  - Covergroup & Coverpoint

- Coverpoint bins

Nội dung tự tìm hiểu:

- System Verilog Assertion
  - SVA Building Blocks
  - SVA Sequence
  - Implication Operator
  - Repetition Operator
  - SVA Built-In Methods
  - Ended and Disable iff
  - Variable delay in SVA
- SystemVerilog Coverage
- Function Coverage
  - SAMPLE
  - COVER POINTS
  - COVERPOINT EXPRESSION
  - GENERIC COVERAGE GROUPS
  - COVERAGE BINS
  - EXPLICIT BIN CREATION
  - TRANSITION BINS
  - WILDCARD BINS
  - IGNORE BINS
  - ILLEGAL BINS
  - CROSS COVERAGE
  - COVERAGE OPTIONS
  - COVERAGE METHODS
  - SYSTEM TASKS
  - COVER PROPERTY
- Cross Coverage
- Coverage Options
- Parameters and `define

Questions:

- What is the purpose of an assertion?
- Difference between immediate assertion and concurrent assertion
- Write SVA checkers for timing and protocol of write operation as defined in DDR2 specification.
- Why need to care of coverage information during test?
- Where coverage properties can be declared?
- Compare cover grop and cover property
- Meaning of coverage information

References:

- https://www.chipverify.com/systemverilog/systemverilog-tutorial
- Springer – System Verilog for Verification (3nd)
- Professional verification – A guide to advanced functional verification
- https://verificationguide.com/systemverilog/systemverilog-tutorial/

3.3.1. Bài tập thực hành số 1

Contents:

- Read design requirements and Specs
- Build Basic System Verilog Test Bench to verify the Design
- Write testcases to figure out the RTL bugs
- Write SVA checkers, Functional Coverage

Requirement:

- Reading specification of the design to extract all its functions
- Build and Understanding operation of Testbench
- Test bench structure
- Test cases
- Functional coverage

3.4. Tuần 4

Nội dung giới thiệu:

- UVM testbench architecture
  - UVM Testbench Top
  - UVM Test
  - UVM Environment
  - UVM Driver
  - UVM Sequencer
  - UVM Sequence
  - UVM Monitor
  - UVM Agent
  - UVM Scoreboard
  - UVM Subscriber
  - UVM Virtual sequencer

Nội dung tự tìm hiểu:

- UVM Sequence item
  - Utility & Field Macros
  - Methods with example
  - Create
  - Print
  - Copy
  - Clone
  - Compare
  - Pack
  - UnPack
- UVM Sequence
  - Sequence Methods
  - Sequence Macros
  - Sequence Example codes

  - UVM Sequence control

- UVM Sequencer with Example
  - Set Method
  - Get Method
- UVM Driver with example
- UVM Monitor with example
- UVM Agent with example
- UVM Scoreboard with example
- UVM Environment with example
- UVM Test with example

Questions:

- Main components in architecture of UVM test bench, function of each?

References:

- http://www.testbench.in
- Uvm-cookbook
- https://www.chipverify.com/uvm/uvm-tutorial
- https://verificationguide.com/uvm/uvm-tutorial/

3.5. Tuần 5

Nội dung giới thiệu:

- UVM Phases
- UVM Factory
- UVM Config DB
- Stimulus Generation
- Driver Sequencer Handshake

Nội dung tự tìm hiểu:

- UVM User-defined phase
- UVM Resource database
- UVM Config database

- UVM config_db examples
- Creating/Using sequences
- UVM `uvm_do sequence macros
- UVM sequence - start()
- UVM sequence - do macros
- Macros for pre-existing items
- UVM Virtual Sequence
- UVM sequence library
- UVM Sequence Arbitration
- UVM Driver Sequencer Connection
- Using get_next_item()
- Using get() and put()

Questions:

- Run all example

References:

- http://www.testbench.in
- Uvm-cookbook
- https://www.chipverify.com/uvm/uvm-tutorial
- https://verificationguide.com/uvm/uvm-tutorial/

3.6. Tuần 6

Nội dung giới thiệu:

- Reporting Infrastructure
- UVM TLM
- Register Layer

Nội dung tự tìm hiểu:

- UVM Reporting Functions
- UVM Printer
- UVM TLM Preface

- TLM Blocking Put Port
- TLM Nonblocking Put Port
- TLM Put -> Export -> Imp
- TLM Blocking Get Port
- TLM Nonblocking Get Port
- UVM TLM Fifo
- UVM TLM Example
- UVM TLM Analysis Port
- UVM TLM Sockets
- Using UVM TLM _decl macros
- Introduction UVM Register
- Register Model
- Register Environment
- Connecting register env
- Complete Example
- UVM Backdoor Access

Questions:

- Run all example

References:

- http://www.testbench.in
- Uvm-cookbook
- https://www.chipverify.com/uvm/uvm-tutorial
- https://verificationguide.com/uvm/uvm-tutorial/

3.7. Tuần 7

3.7.1. Bài Tập Thực Hành Số 2

Contents:

- Read design requirements and Specs
- Create verification plan
- Build UVM test bench to verify the design

- Write sequences and testcases to figure out the RTL bugs
- Write SVA checkers, Scoreboard check model, Functional Coverage
- Create reports about coverage information
- Analyzing test results

3.7.1.1. Study specification and write Verification plan

Requirement:

- Reading specification of the design to extract all its functions
- Understanding operation of design
- Create verification plan:
- Functional extraction
- Test bench structure
- Stimulus generation plan
- Creating functional coverage model for the design
- Checker plan
- Test cases
- Functional coverage plan

3.7.1.2. Building test bench, sequences and testcases

Requirement:

- Building test bench based on the verification plan
- Running simulation
- Integrating checkers to the test bench

3.7.1.3. Functional coverage and SVA integration

Requirement:

- Creating coverage model
- Integrating coverage model into the test bench
- Write SVA
- Running all test cases

3.7.1.4. Analyzing the test result and creating the test report

Requirement:

- Get 100% code and functional coverage
- Analyzing the test results
- Creating test reports
- Review and Presentation