User guide for EDA tools

Mentor HDL-Designer and Modelsim

Inhalt

1 Introduction

Welcome to the comprehensive user guide for EDA development using HDL Designer and ModelSim. This guide is designed to assist you in navigating through the functionalities of these powerful tools for Hardware Description Language (HDL) development. HDL Designer facilitates the creation and management of projects, libraries, and schematics, while ModelSim specializes in simulation tasks to verify the functionality of your designs.

In this guide, we will walk you through essential aspects such as launching HDL Designer, managing libraries, creating schematics, and utilizing various features to streamline your EDA development process. The integration of ModelSim for simulation purposes will also be explored, ensuring a thorough understanding of the end-to-end development cycle.

Let's dive into the intricacies of HDL Designer and ModelSim, empowering you to harness their capabilities efficiently.

2 HDL-Designer

2.1 General

2.1.1 Launch HDL Designer

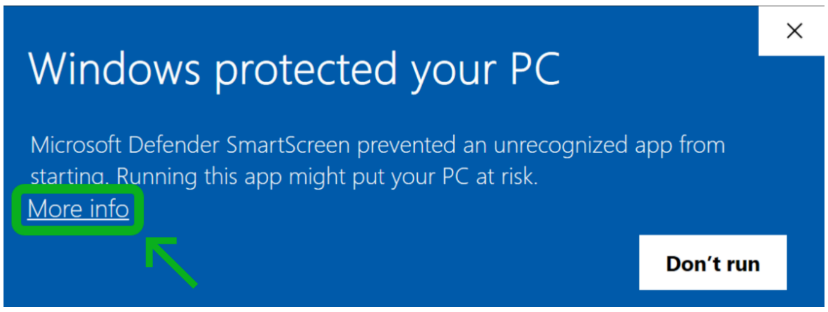
Within each HEVS HDL Designer project, both a **`{projectName}.bat`** for Windows and a **`{projectName}.bash`** for Linux file can be found at the root.

> ⚠ The project path name should not contain spaces or special characters such as accents or other symbols

Windows

Under Windows, HDL can be launched by executing a double-click on the **`*.bat`** file.

If Window Smart Screen protection pops-up (normally only during the first launch), click on **More Info** then **Run anyway**:


Windows protected your PC

Microsoft Defender SmartScreen prevented an unrecognized app from starting. Running this app might put your PC at risk.

Abbildung 1: Windows Smart Screen

Linux

Under Linux, HDL can be launched by executing the **`*.bash`** file. This can be achieved by entering the following commands in a terminal:

```
chmod +x *.bash # Gives ourselves permission to run it
./*.bash        # Run it
```

2.1.2 Libraries

Press on the **`project`** tab to display the list of the libraries available.

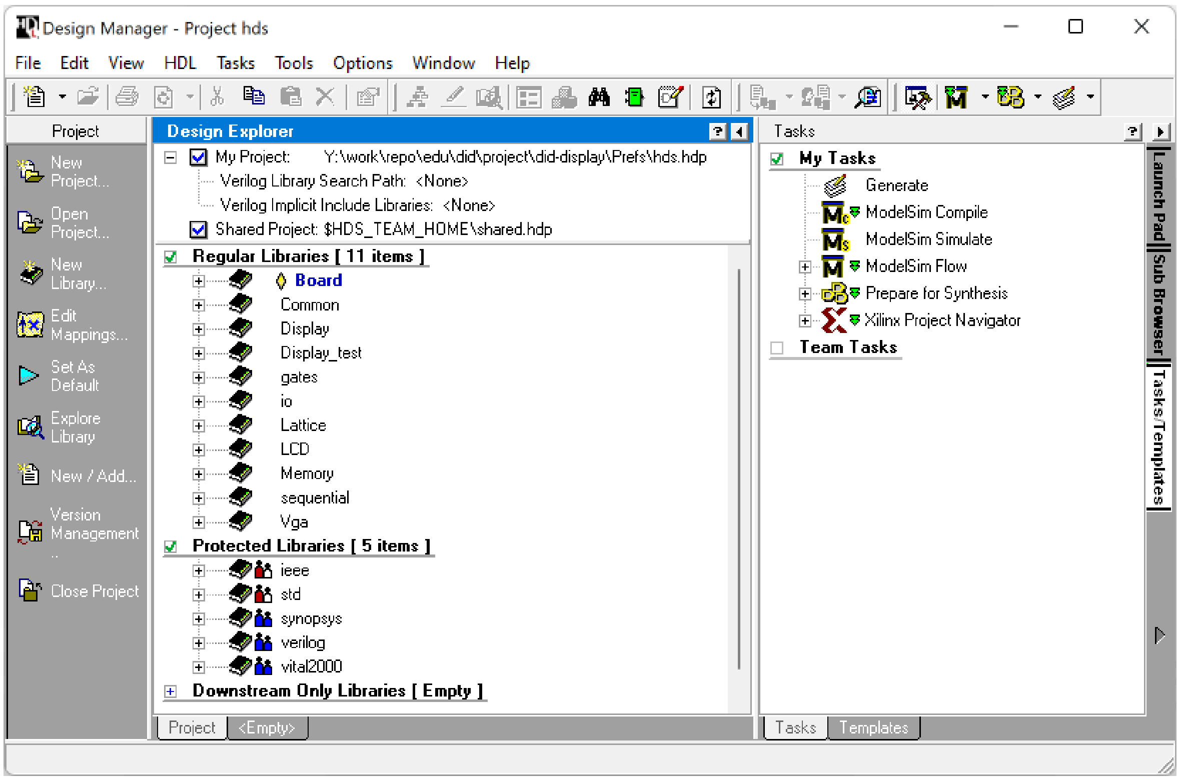


Abbildung 2: Library list

Open library

To open a library, **`double click`** on it.

To refresh its content, once open, simply press **`F5`**.

Library and Test library

Each project has its own library where all the project's blocks are stored. In this library, you can find and access the different blocks developed for the project.

Additionally, each project is equipped with a test library. This test library provides test benches for testing and simulating the developed components.

In the example of Abbildung 2 the library **`Display`** is the project library and **`Display_test`** is the test library.

Board library

The Board library contains the board-level componenets adapted for the used Hardware board. It oversees the management of the system's inputs and outputs, synchronizing them with the system clock and wiring them to known names.

The board-level I/O names are later used to wire the signals onto actual physical pins of the FPGA.

In the example of Abbildung 2 the library **`Board`** is the library containing the board-level abstraction.

2.2 Create schematic

2.2.1 Add signals and buses

To add a bus or a signal in the schematic, click on the highlighted buttons (see Abbildung 3), with the `green button` for signals (1 bit) and the `blue button` for buses (2 or more bits).

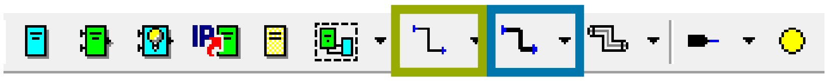


Abbildung 3: Add signal/bus buttons

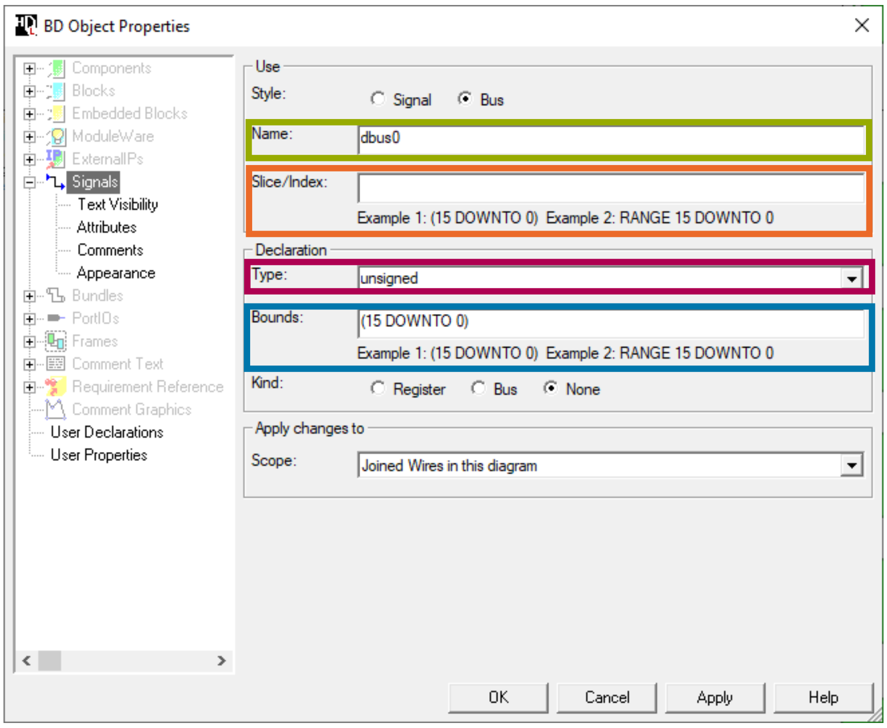
After clicking the button, a new signal/bus can be drawn by clicking on the schematic.

To confirm the new signal/bus, either wire blocks inputs and outputs together, or double-click anywhere on the schematic to finish the signal creation.

With **`ESC`** you can cancel the creation of the signal/bus.

Change types and bus sizes

Double click on the signal or bus and the pop up window (see Abbildung 4) will open.


BD Object Properties

Use
Style: Signal [✓ Bus]
Name: dbus0
Slice/Index:
Example 1: (15 DOWNTO 0) Example 2: RANGE 15 DOWNTO 0
Declaration
Type: unsigned
Bounds: (15 DOWNTO 0)
Example 1: (15 DOWNTO 0) Example 2: RANGE 15 DOWNTO 0
Kind: Register Bus [✓ None]
Apply changes to
Scope: Joined Wires in this diagram

Abbildung 4: Edit signal&bus properties

Parameter	Description
`Name`	Change the signal/bus name
`Slice/Index`	For multi-bits signals, select which bit(s) to use (if empty, uses the whole bus)
`Type`	Change signal/bus type. See Abschnitt 2.2.1.1.1
`Bounds`	Change bus size

The **Style** checkbox is only for cosmetic purpose. **Signal** draws a thin line, **Bus** a thicker one.

NEVER change the **Kind** attribute. Always leave it to **None**.

Types

Types tell us „how the data is represented“ and „what value it can take“.

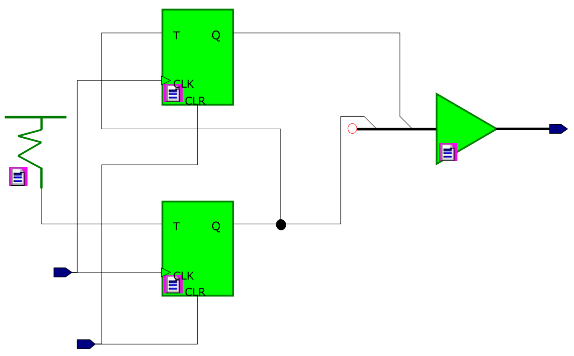
Usage	Signal type	Description	Note
• Design • test-bench	**std_ulogic**	U, X, 0, 1, Z, W, L, H, -	Type used in labs
	std_logic	resolved std_ulogic	
	std_ulogic_vector	array of std_ulogic	Bits together not representing values (e.g. control bits)
	std_logic_vector	array of std_logic	
	unsigned	0 to $2^n - 1$	Bits together representing numbers (e.g. counters)
	signed	2^{n-1} to $2^{n-1} - 1$	
• test-bench	**Boolean**	true, false	
• generics • constant • test-bench	**natural**	0 to $2'147'483'647$	
	integer	$-2'147'483'648$ to $2'147'483'647$	
	positive	1 to $2'147'483'647$	
	real	from $-1.0E38$ to $1.0E38$	
	time	the time primary base of 1 [fs]	E.g.: TIME_DELTA: time:= 100ns;

The difference between **`std_logic`** and **`std_ulogic`** is that when two outputs are wired together, the **`ulogic`** produces an error during compilation, while the **`logic`** allows the simulation to run, outputting an ‚**X**‘ value.

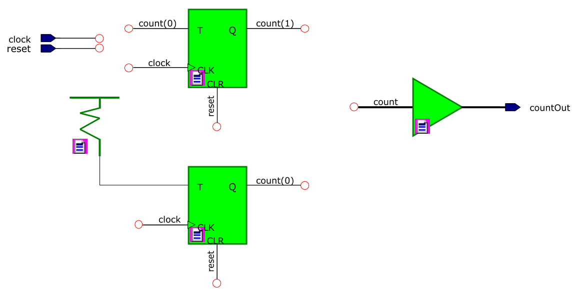
Connect signals and buses

There are two different ways to connect multiple components:

`Wired connection`



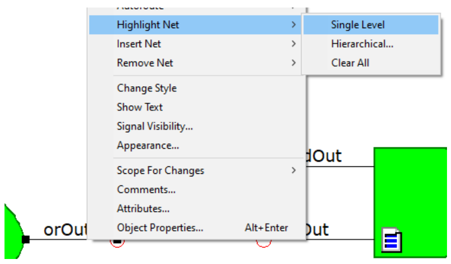
`Wireless connection`



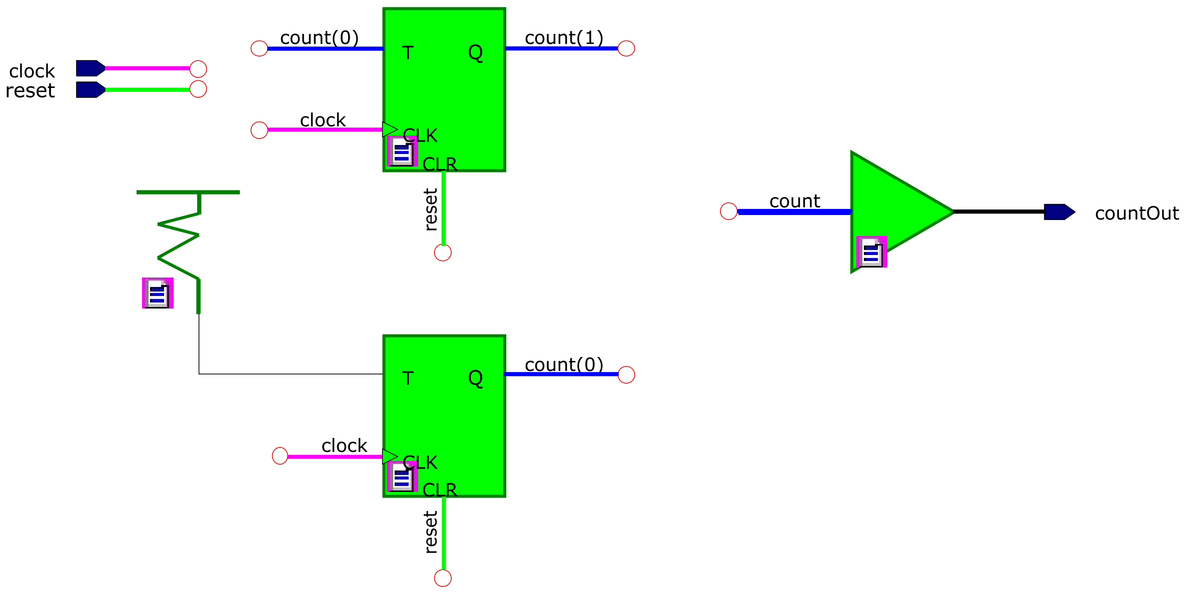
For large designs, the **`wireless connection`** is in most of the cases preferred over a **`wired connection`** for readability reasons.

To verify that a signal is well connected, it can be highlighted :

1. **`Right-Click`** on the signal
2. Select **`Highlight Net`**
3. Click on **`Single level`**

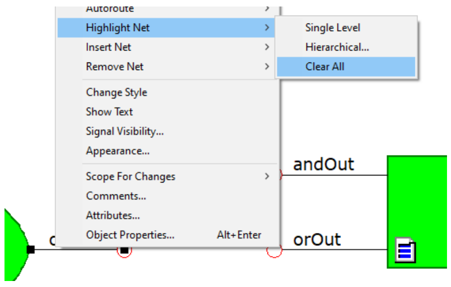


This will produce the result below. Each highlighted signal will appear in a different color.



To clear all the highlighted signals :

1. **`Right-Click`** on the signal
2. Select **`Highlight Net`**
3. Click on **`Clear All`**

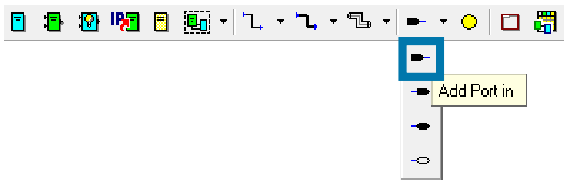


2.2.2 IOs

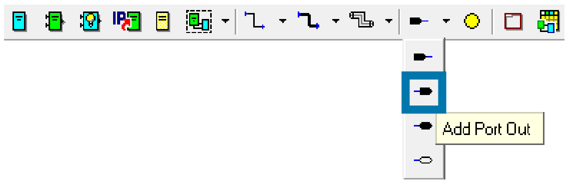
I/Os of a block correspond to the signals exposed to the parent using this block.

Other signals are local and do not allow a direct connection from another block.

To add a new input for the component, click on the **Add Port in** button.



To add a new output for the component, click on the **Add Port out** button.



Then simply wire the targeted signal on this port.

Adding, removing, modifying input/outputs modify the component interface. Therefore the component interface needs to be updated see Abschnitt 2.2.5

An output port can not be used internally. It is necessary to use an intermediary signal to use internally and connect a buffer to forward the signal to the output.

In the example of Abbildung 5 the output **countOut** is not read, a buffer is used and the internal signal **count** is read.

2.2.3 Create a component

To create a new component, a block needs to be added. To add a block, click on the `add component` button (see Abbildung 6).

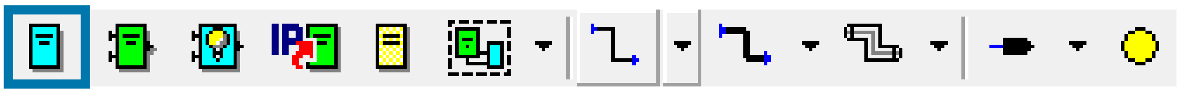


Abbildung 6: Add block

After clicking the button, a new block can be added by clicking on the schematic. Once the block added, the needed IOs can be wired :

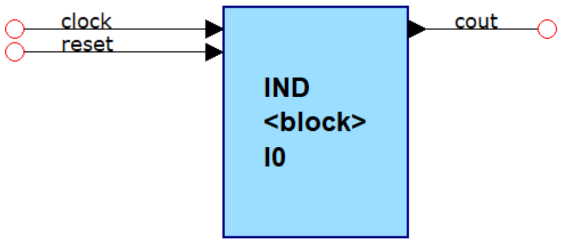
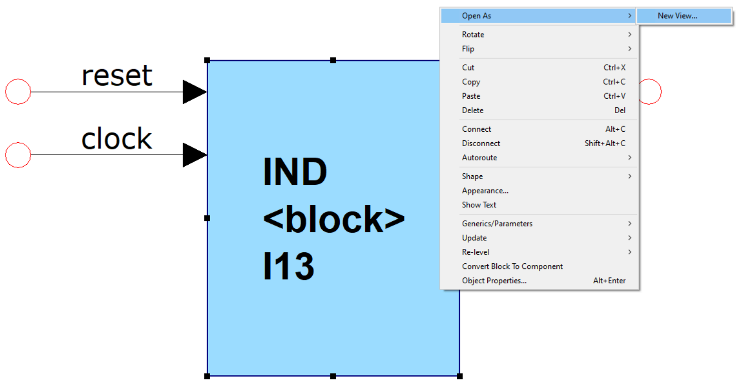


Abbildung 7: New block wired

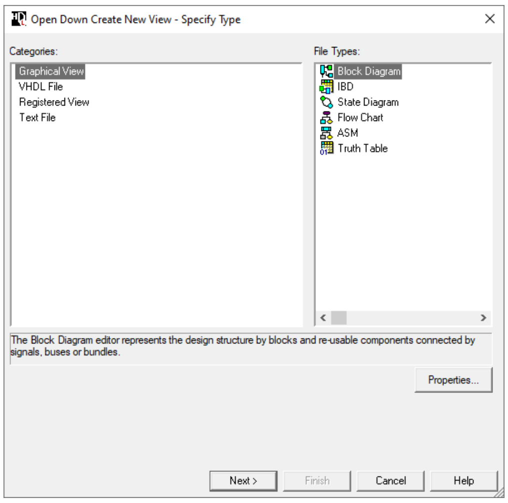
`Blue block` can't be copied is exists only once. Only **`green block`** (components) can be copied. See Abschnitt 2.2.4

Once the IOs wired, the type of the block needs to be selected (**`Block Diagram`** /**`State Diagram`** /**`VHDL file`**).

1. **`Right-click`** on the block
2. Select **`Open As`**
3. Click on **`New View...`**



The following windows will be shown



Open Down Create New View - Specify Type

Categories:
- Graphical View
- VHDL File
- Registered View
- Text File

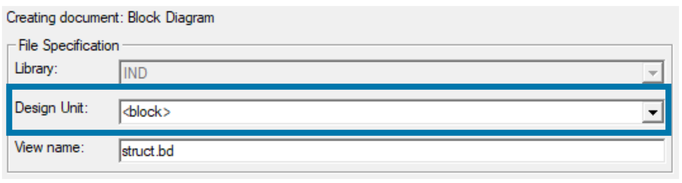
File Types:
- Block Diagram
- IBD
- State Diagram
- Flow Chart
- ASM
- Truth Table

The Block Diagram editor represents the design structure by blocks and re-usable components connected by signals, buses or bundles.

Link	Block type	Icon
`Graphical View/Block Diagram`	block diagram	⊞
`Graphical View/State Diagram`	state machine	⚬
`VHDL File/Architecture`	VHDL code	📄

Block diagram

- Select **`Graphical View/Block Diagram`**
- Press **`Next`**
- Enter the block name under **`Design unit`**
- Fill the I/Os table
  - Ensure the correct type
  - Set the bounds for multi-bits types
- Press „Finish“



Creating document: Block Diagram

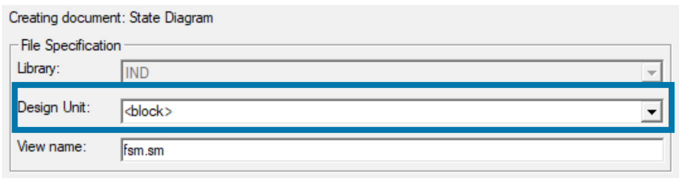
File Specification

Library:	IND
Design Unit:	<block>
View name:	struct.bd

I/Os can still be added, removed and modified when editing the schematic

State diagram

- Select **`Graphical View/State Diagram`**
- Press **`Next`**
- Enter the block name under **`Design unit`**
- Fill the I/Os table
  - Ensure the correct type
  - Set the bounds for multi-bits types
- Press **`Finish`**



Creating document: State Diagram

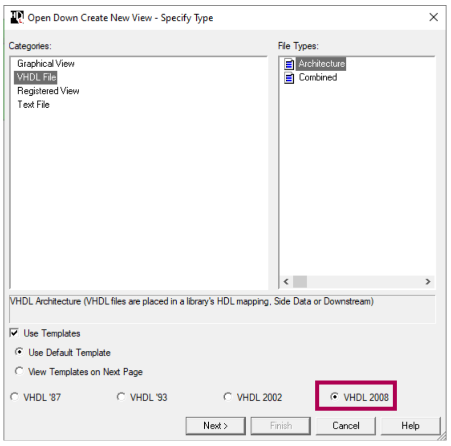
File Specification

Library:	IND
Design Unit:	<block>
View name:	fsm.sm

I/Os can still be added, removed and modified when editing the schematic

VHDL Code

- Select **`VHDL File/Architecture`**
- Select VHDL language version (**`VHDL 2008`**)
- Press **`Next`**
- Enter the block name under **`Entity`**
- Enter architecture name (**`RTL`**)
- Press **`Finish`**



Open Down Create New View - Specify Type

Categories: Graphical View, [✓ VHDL File], Registered View, Text File

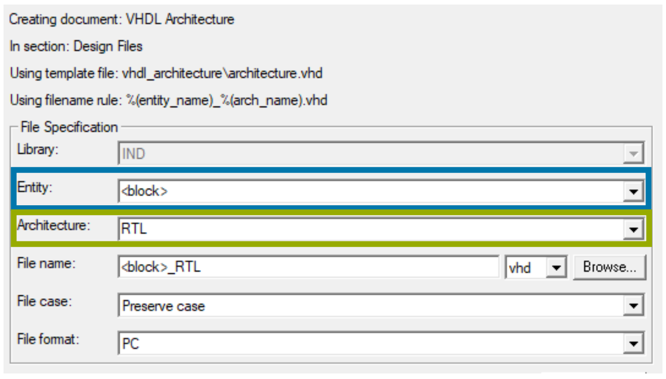
File Types: [✓ Architecture], Combined

VHDL Architecture (VHDL files are placed in a library's HDL mapping, Side Data or Downstream)

[✓ Use Templates]

[✓ Use Default Template] View Templates on Next Page

VHDL '87 VHDL '93 VHDL 2002 [✓ VHDL 2008]



Creating document: VHDL Architecture

In section: Design Files

Using template file: vhdl_architecture\architecture.vhd

Using filename rule: %(entity_name)_%(arch_name).vhd

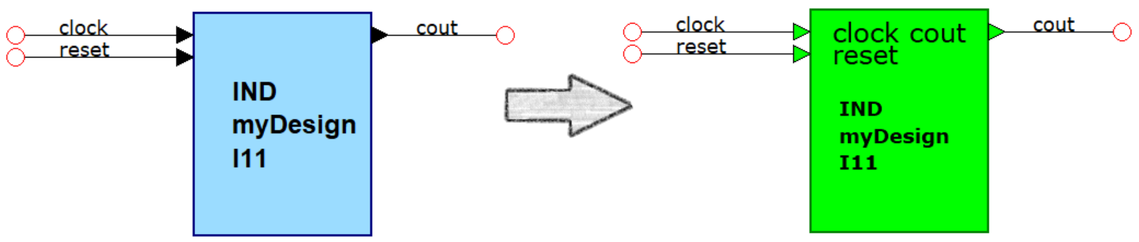
File Specification

Library:	IND
Entity:	<block>
Architecture:	RTL
File name:	<block>_RTL vhd
File case:	Preserve case
File format:	PC

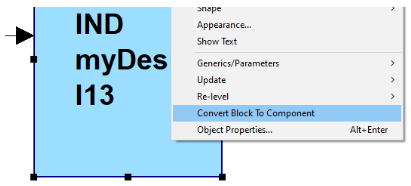
2.2.4 Convert block to component (blue to green)

Converting the block into a component (from blue to green) enables you to copy and paste the block and makes it accessible in the project library.

Blue blocks are great for creating an block interface quickly and green blocks are a must for reusing the block elsewhere in the project.



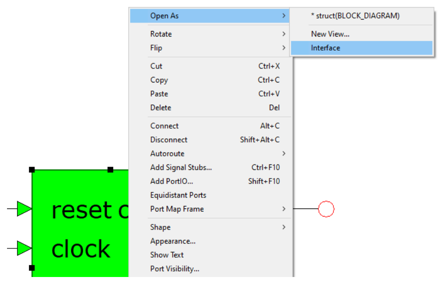
1. **`Right-Click`** on the component (blue block)
2. Click on **`Convert Block to component`**



2.2.5 Update component interface

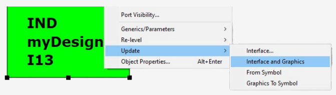
A component interface consists of multiple parts: **`Inputs and Outputs`**, **`Generics`**, and **`Symbol`**. These various parameters can be added, removed and modified.

1. **`Right-Click`** on the component (green block)
2. Select **`Open As`**
3. Click on **`Interface`**

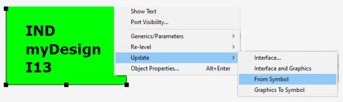


`Once the component interface has been changed`

1. **`Right-Click`** on the component
2. Select **`Update`**
3. Click on **`Interface and Graphics`**



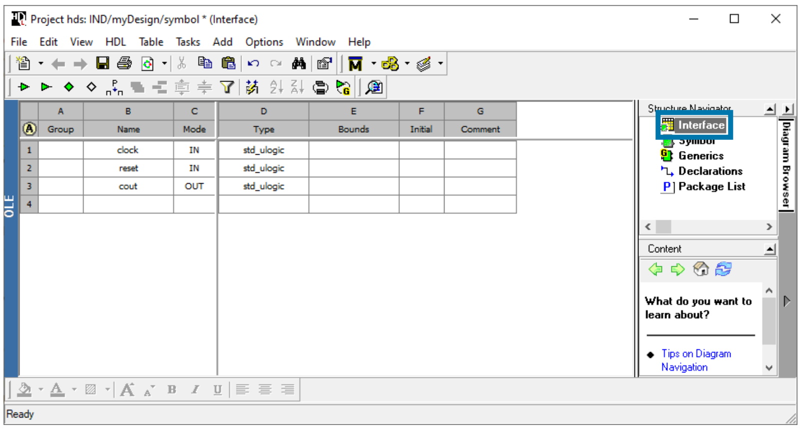
1. **`Right-Click`** on the component
2. Select **`Update`**
3. Click on **`From Symbol`**



Update IOs

I/Os can be added, removed and modified in the following table :

- **`Click`** on **`Interface`**

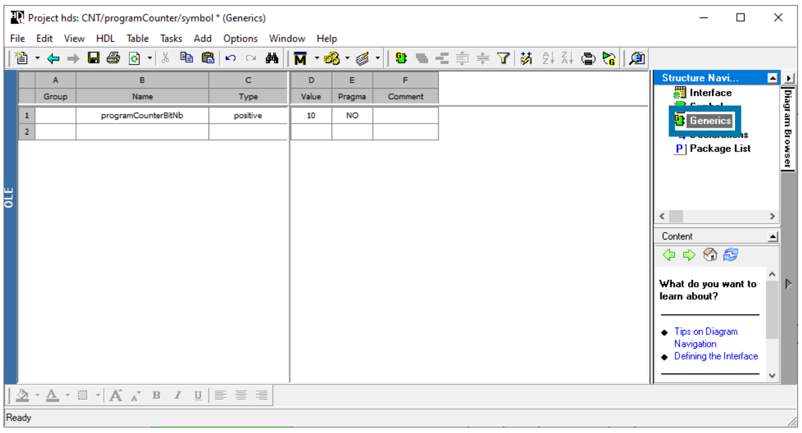

Project hds: IND/myDesign/symbol * (Interface)
File Edit View HDL Table Tasks Add Options Window Help
OLE
Structure Navigator
Interface
Symbol
Generics
Declarations
Package List
Diagram Browser
Content
What do you want to learn about?
Tips on Diagram Navigation
Ready

	A	B	C	D	E	F	G
Ⓐ	Group	Name	Mode	Type	Bounds	Initial	Comment
1		clock	IN	std_ulogic			
2		reset	IN	std_ulogic			
3		cout	OUT	std_ulogic			
4							

Update Generics

A VHDL generic is a constant value used to parameterize a VHDL design description. For example, a generic can be a bus length, a loop max index, etc... The generics can be added, removed and modified in the following table :

- **`Click`** on **`Generics`**


Project hds: CNT/programCounter/symbol * (Generics)

	A	B	C	D	E	F
	Group	Name	Type	Value	Pragma	Comment
1		programCounterBitNb	positive	10	NO	
2						

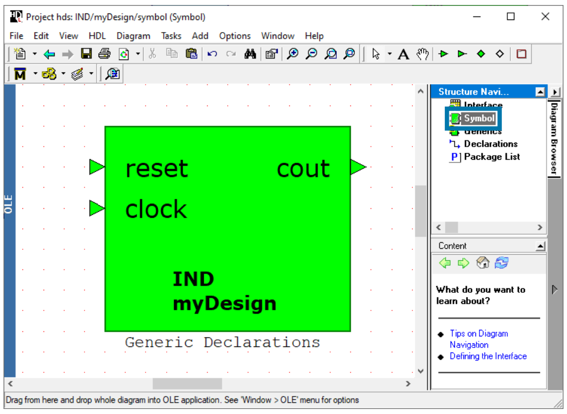
File Edit View HDL Table Tasks Add Options Window Help
OLE
Structure Navi...
Interface
Generics
Package List
Diagram Browser
Content
What do you want to learn about?
Tips on Diagram Navigation
Defining the Interface
Ready

A description of **`VHDL types`** can be found in Abschnitt 2.2.1.1.1

Update symbol

The component symbol can be edited in the following window:

- **`Click`** on **`Symbol`**



Project hds: IND/myDesign/symbol (Symbol)
File Edit View HDL Diagram Tasks Add Options Window Help
OLE
reset
cout
clock
IND
myDesign
Generic Declarations
Structure Navi...
Interface
Symbol
Generics
Declarations
Package List
Diagram Browser
Content
What do you want to learn about?
Tips on Diagram Navigation
Defining the Interface
Drag from here and drop whole diagram into OLE application. See 'Window > OLE' menu for options

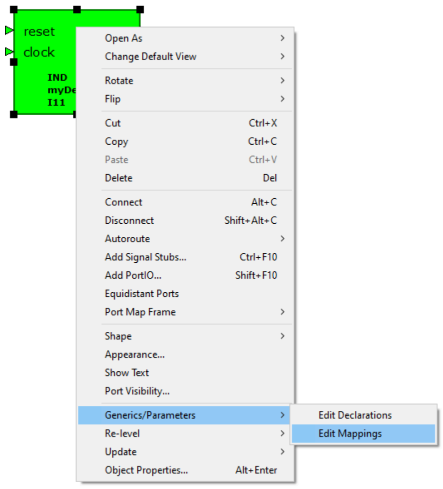
Set where the I/Os are shown, where texts are displayed, the component size ...

2.2.6 Generics mapping

Generic values can be set from outside the component. This enables the creation of a generic component that can be configured based on a specific application.

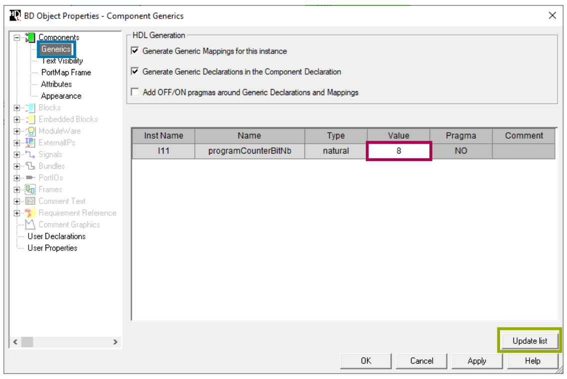
For instance, a generic value is used in counters, allowing the user to configure the number of bits without the need to modify the component itself.

1. **`Right-click`** on the component
2. Select **`Generics/Parameter`**
3. Click on **`Edit Mappings`**



Upon clicking on **`Edit Mappings`**, the following window will be opened:

1. **`Click`** on **`Generics`**
2. **`Fill`** the generic **`value`**. It can either be a value or another generic name available in the current block.
3. **`Click`** on **`Update list`**



HDL Generation

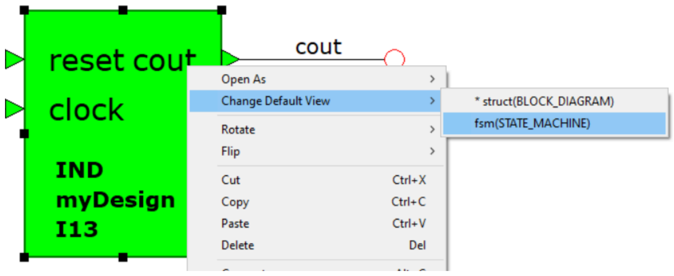
- [✓] Generate Generic Mappings for this instance
- [✓] Generate Generic Declarations in the Component Declaration
- [ ] Add OFF/ON pragmas around Generic Declarations and Mappings

Inst Name	Name	Type	Value	Pragma	Comment
I11	programCounterBitNb	natural	8	NO	

2.2.7 Set default view

A component (green block) can have multiple architectures. To switch between the architectures :

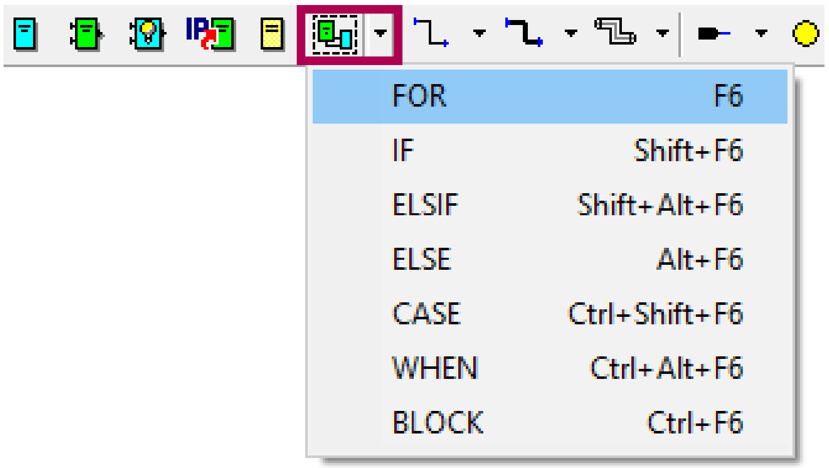
1. **`Right-click`** on the component
2. Select **`Change Default View`**
3. **`Click`** on the new architecture



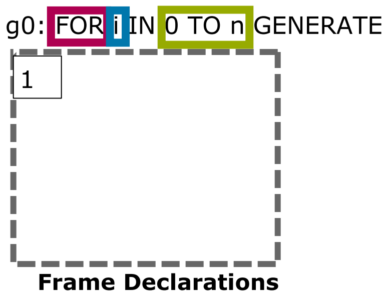
It allows you to have different implementations using the same I/Os to keep track of your progress, various tests you made ...

2.2.8 FOR Generate

`FOR Generate` frames allow to create multiple iterations of the same structure. To add a new frame, **`click`** on the **`add frame`** button and **`click`** on **`FOR`**.



When adding a **`FOR frame`** the following element is drawn on the schematic.



`Part`	**`Description`**
`FOR`	Type of the frame (**`FOR`** loop)
`i`	Index of the loop
`0 TO n`	Range of the for loop

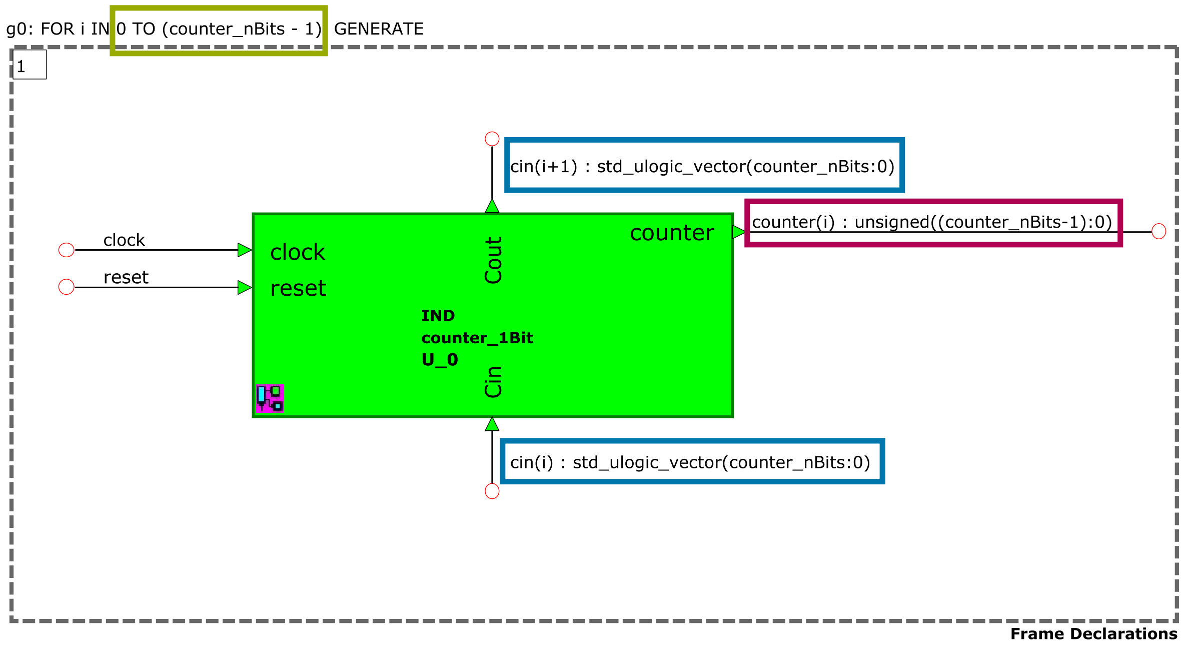
Any block and signal within the frame will be copy-pasted **`n+1`** times. The **`i`** index will go from **`0`** to **`n`**. *Modify n to a corresponding value/generic.*

All signals within the **FOR** frame which should not repeat on all copies must use the **i** index in their **Slice/Index** part. E.g.: clock should be the same everywhere; but a counter must output bit 0, then 1, then 2 ... of the total count.

No **Port** should reside within the frame.

Example

The example demonstrate an iterative counter circuit. The generic **`counter_nBits`** allows the selection of the number of bits for the counter.


g0: FOR i IN 0 TO (counter_nBits - 1) GENERATE
1
cin(i+1) : std_ulogic_vector(counter_nBits:0)
counter(i) : unsigned((counter_nBits-1):0)
clock
reset
clock
reset
Cout
counter
IND
counter_1Bit
U_0
Cin
cin(i) : std_ulogic_vector(counter_nBits:0)
Frame Declarations

Part	Description
`0 TO (counter_nBits-1)`	Repeat the loop **`counter_nBits`** times
`counter(i): unsigned...`	Write the counter output. The **`1st block`** writes the **`bit0`** of the counter signal. The **`2nd block`** writes the **`bit1`** of the counter signal, etc..

`cin(i) : std_ulogic_vector ...`
`cin(i+1) : std_ulogic_vector...`
	Transmit information to the next block.
	The **`1st block cout`** is connected to the **`2nd block cin`**, etc..

The size of the **`cin`** bus is **`(counter_nBits DOWNTO 0)`**. Therefore, it's length is **`counter_nBits + 1`** bits.
As we have **`cin(i+1)`**, when **`i = (counter_nBits - 1)`**⇒**`cin(counter_nBits)`**

Do not forget to connect **`cin(0)`**, OUTSIDE of the frame.

2.2.9 Add a component

To **add** a new component, **`click`** on the **`add component`** button.

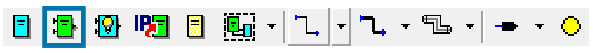
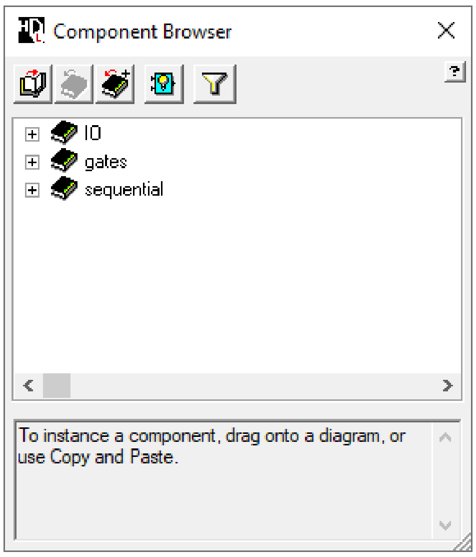


Abbildung 9: Add a component

The following windows will pop-up :



To instance a component, drag onto a diagram, or use Copy and Paste.

Abbildung 10: Add a component

Library name	Description	Content
IO	Input/Output components	Tri-state
gates	combinatorial logic components	Gates, buffers, multiplexer/de-multiplexer, logic level
sequential	Sequential logic components	Flip-flop, counter, register, frequency divider

To instance (use) a component, **`click + drag`** onto a diagram or use **`Copy + Paste`**.

In the libraries **`gates`**, **`sequential`** and **`IO`** component are abbreviated for example :

`gates`

`and2`	**`and`** gate with 2 inputs
`or3inv1`	**`or`** gate with 3 inputs where one is inverted
`nand4`	**`and`** gate with the output inverted and with 4 inputs

`sequential`

`DFF`	**`D`** Flip-Flop
`DFFE`	**`D`** Flip-Flop **`E`**nable
`DFF_PRE`	**`D`** Flip-Flop with **`PRE`**set instead of reset
`TFF`	**`T`** Flip-Flop

To **add** a new library in the component browser, **click** on the **add library** button.

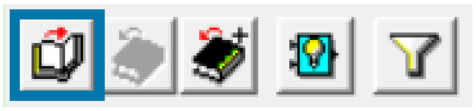
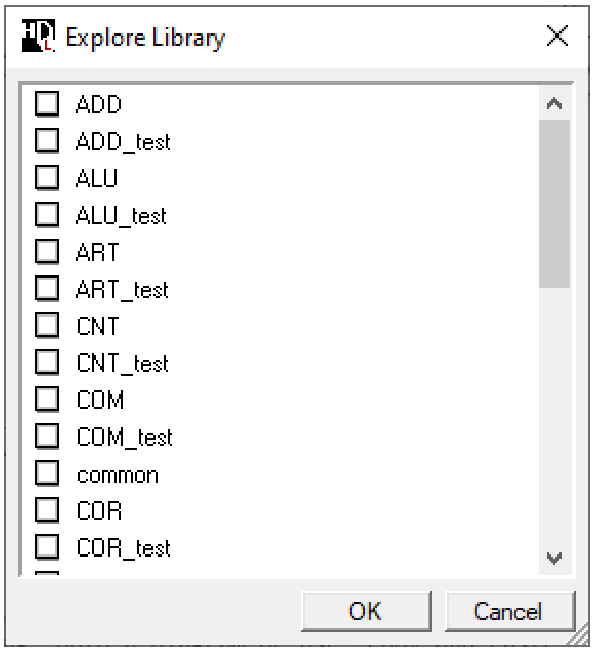


Abbildung 11: Add a component

The following windows will pop-up :



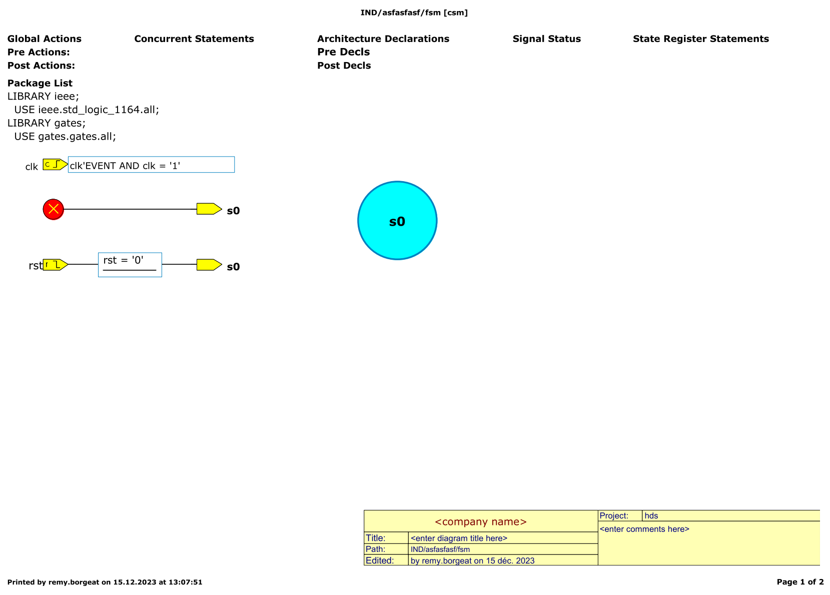
- [ ] ADD
- [ ] ADD_test
- [ ] ALU
- [ ] ALU_test
- [ ] ART
- [ ] ART_test
- [ ] CNT
- [ ] CNT_test
- [ ] COM
- [ ] COM_test
- [ ] common
- [ ] COR
- [ ] COR_test

Abbildung 12: Add a component

Check the library to be added and **click** on the **OK** button.

2.2.10 State machine

Add a **`FSM block`** as explained in Abschnitt 2.2.3. The Abbildung 13 shows the initial content of an **`FSM block`**.



IND/asfasfasf/fsm [csm]

Global Actions **Concurrent Statements** **Architecture Declarations** **Signal Status** **State Register Statements**
Pre Actions:
Post Actions:
Pre Decls
Post Decls

Package List

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
LIBRARY gates;
  USE gates.gates.all;
```

clk'EVENT AND clk = '1'

rst = '0'

<company name>		Project:	hds
Title:	<enter diagram title here>	<enter comments here>	
Path:	IND/asfasfasf/fsm		
Edited:	by remy.borgeat on 15 déc. 2023		

Printed by remy.borgeat on 15.12.2023 at 13:07:51

Page 1 of 2

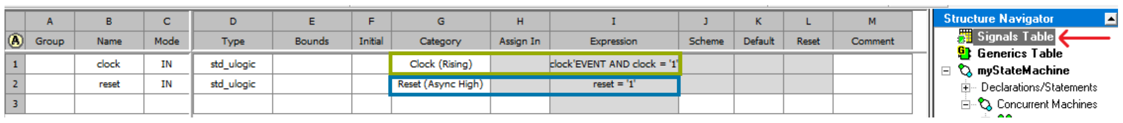
Abbildung 13: Initial content of an FSM block

Clock and reset

The state machine needs at least **`2 signals`** : **`Clock`** and **`reset`**. These 2 signals needs to be configured :

- When do we switch from one state to another ?
- When do we reset our flip-flops ?

All these configurations are done in the Abbildung 14.



	A	B	C	D	E	F	G	H	I	J	K	L	M
Ⓐ	Group	Name	Mode	Type	Bounds	Initial	Category	Assign In	Expression	Scheme	Default	Reset	Comment
1		clock	IN	std_ulogic			Clock (Rising)		clock'EVENT AND clock = '1'				
2		reset	IN	std_ulogic			Reset (Async High)		reset = '1'				
3													

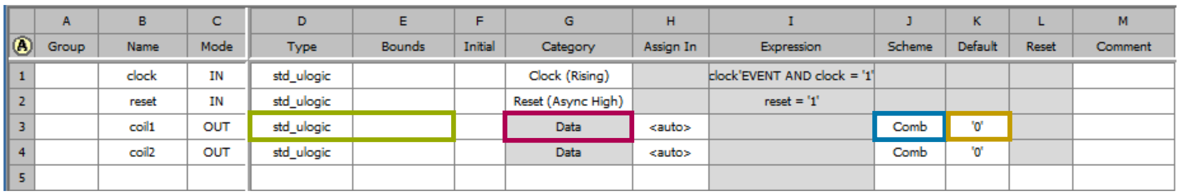
Abbildung 14: State machine signals

`Signal`	**`Configuration`**
`clock`	Rising edge
`reset`	Asynchronous high

By default, reset configuration is **`asynchronous low`**. In the different projects, the reset signal is active high. Therefore, the configuration needs to be changed to **`asynchronous high`**.

I/Os

The state machine I/Os can be added in the signals table as shown in the Abbildung 15.



	A	B	C	D	E	F	G	H	I	J	K	L	M
Ⓐ	Group	Name	Mode	Type	Bounds	Initial	Category	Assign In	Expression	Scheme	Default	Reset	Comment
1		clock	IN	std_ulogic			Clock (Rising)		clock'EVENT AND clock = '1'				
2		reset	IN	std_ulogic			Reset (Async High)		reset = '1'				
3		coil1	OUT	std_ulogic			Data	<auto>		Comb	'0'		
4		coil2	OUT	std_ulogic			Data	<auto>		Comb	'0'		
5													

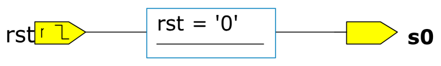
Abbildung 15: State machine signals

The following points are important when configuring one:

Parameter	**Description**
`Type`	Signal type and size
`Category`	The value needs to be **`Data`**
`Scheme`	(For outputs) The value needs to be **`Comb`**
`Default`	(For outputs) It is necessary to assign a default value to the output. IF not specified in the current state, this value is taken.

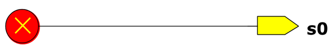
Initial state

Set the initial FSM state after reset (E.g.: enters the **`s0`** state)



Recovery point

Set the recovery state when no valid state is assigned



Draw state machine

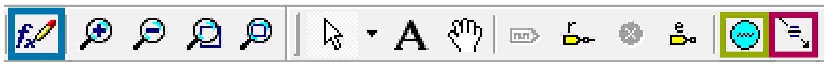
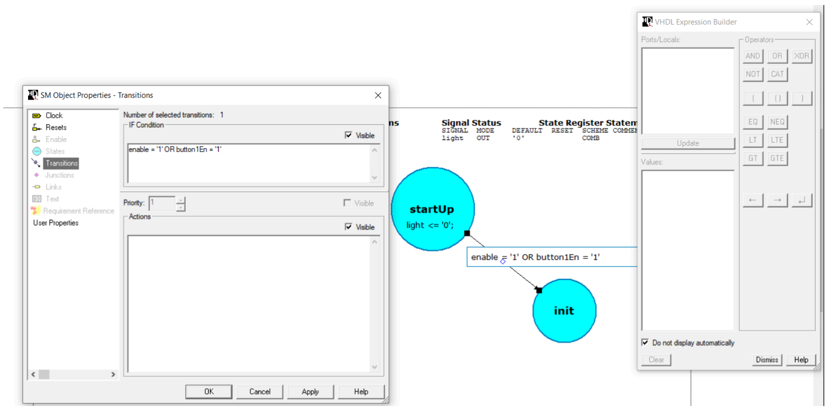


Abbildung 16: Draw state machine buttons

Button	**Goal**
`Expression builder`	Open the expression builder window (See Abschnitt 2.2.10.2.1)
`Add State`	Add a new state
`Add Transition`	Add a transition between states

Expression builder

Upon clicking the **Expression builder** button, the Expression Builder window appears. To modify the state/transition, **double-click** on it and use the expression builder for editing.



Number of selected transitions: 1

IF Condition

enable = '1' OR button1En = '1'

Priority: 1

Actions

Signal Status
```
SIGNAL  MODE  DEFAULT  RESET  SCHEME  COMMEN
light   OUT   '0'             COMB
```
State Register Statem

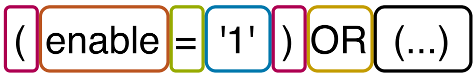
startUp
light <= '0';

enable = '1' OR button1En = '1'

init

☑ Do not display automatically

Transition syntax



Multiples part are forming the transition syntax :

Part	Description
In pink	parentheses to enclose the condition
In orange	input name
In green	sign (=, /=, >, >=, <, <=)
In blue	value; between '...' for ulogic, "..." for vectors and (un)signed
In yellow	logical operator (AND, OR, XOR, NOT, CAT)

Sign

Certain signs are not compatible with all signal types. Refer to the table below

Signal type	Allowed sign
std_(u)logic	=, /=, not
std_(u)logic_vector	=, /=
unsigned / signed	=, /=, >, >=, <, <=

Value

The value syntax in the field varies based on the signal type. Refer to the table below.

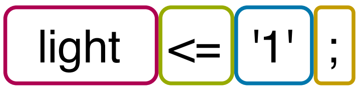
Signal type	Syntax	Example
std_(u)logic	Value between ‚	‚1‘, ‚0‘
std_(u)logic_vector	Value between „	„0011“, „11001100“, 16#FF#
unsigned / signed	Integer Value or bits representation	100, −225, „00100100“

Tabelle 1: VHDL value syntax

Some examples :

```
signal sul  : std_ulogic                     := '0';
signal sl   : std_logic                      := '0';
signal sulv : std_ulogic_vector(15 downto 0) := 16#CAFE#;
signal slv  : std_logic_vector(7 downto 0)   := "01101100";
signal sulv : std_ulogic_vector(7 downto 0)  := (7          => '1',
                                                 5 downto 1 => '1',
                                                 others     => '0');
signal slv  : std_logic_vector(7 downto 0)   := (others => '0');
signal uv   : unsigned(7 downto 0)           := 200;
signal sv   : signed(7 downto 0)             := -100;
```

Output assignment syntax



```
light <= '1' ;
```

Multiples part are forming the output syntax :

Part	Description
In **red**	output name
In **green**	assignment operator
In **blue**	value
In **yellow**	end character

Value

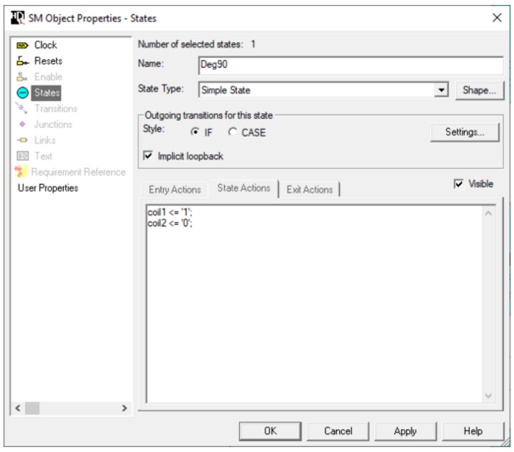
The value syntax in the field varies based on the signal type. Refer to the Tabelle 1.

Some examples :

```
uv   <= to_unsigned(100, y'length);    -- unsigned
uv   <= to_unsigned(16#FF#, y'length); -- unsigned 255
sv   <= to_signed(16#FF#, y'length);   -- signed   -1
sulv <= 16#FF#;                        -- std_ulogic_vector;
sulv <= "01101100";                    -- std_ulogic_vector;
su   <= '0';                           -- std_ulogic
```

Moore

In a Moore state machine, the outputs are configured by double-clicking on a state and filling the „State Actions“ text area (see Abbildung 17).



Number of selected states: 1
Name: Deg90
State Type: Simple State
Outgoing transitions for this state
Style: [✓ IF] CASE
☑ Implicit loopback
☑ Visible

```
coil1 <= '1';
coil2 <= '0';
```

Abbildung 17: Moore output assignment in the state

An example of a Moore state machine is shown in Abbildung 18.

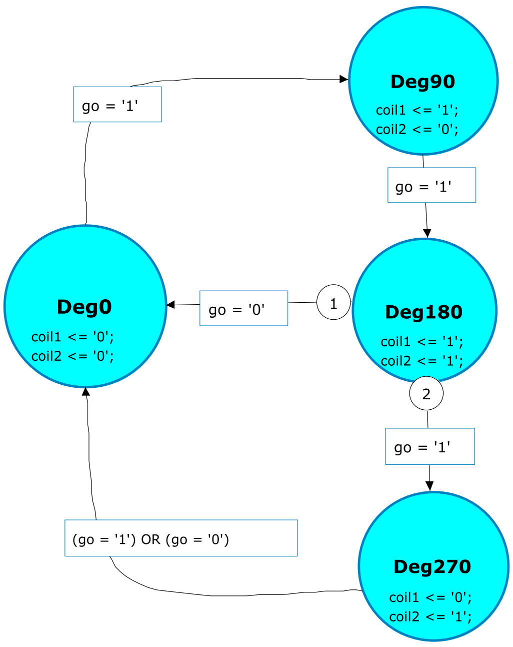
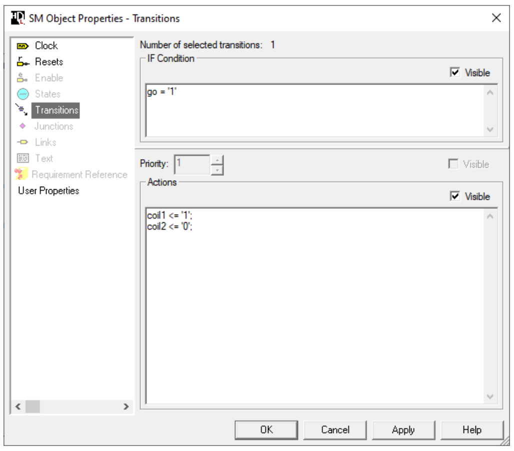


Abbildung 18: Moore state machine

Mealy

In a Mealy SM, the outputs are configured by double-clicking on the transition arrow and filling the „action“ text area (see Abbildung 19).



Number of selected transitions: 1
IF Condition
☑ Visible

```
go = '1'
```

Priority: 1
☐ Visible
Actions
☑ Visible

```
coil1 <= '1';
coil2 <= '0';
```

Abbildung 19: Mealy output assignment in the transition arrow

An example of mealy state machine is shown in Abbildung 20.

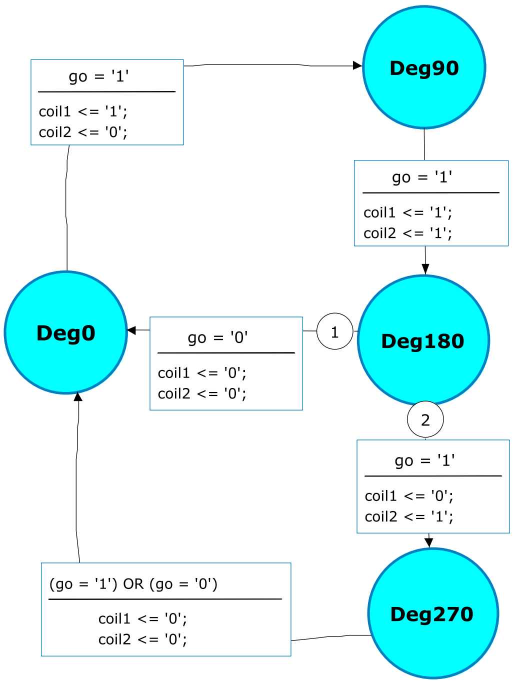


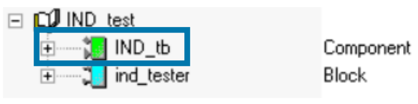
Abbildung 20: Mealy state machine

Do not mix Mealy and Moore representation with HDL-Designer.

3 ModelSim

3.1 Test-bench

- Open the test bench library as explain in Abschnitt 2.1.2.
- **Double-click** on the `test bench component`.



,

The test-bench component contains the following content:

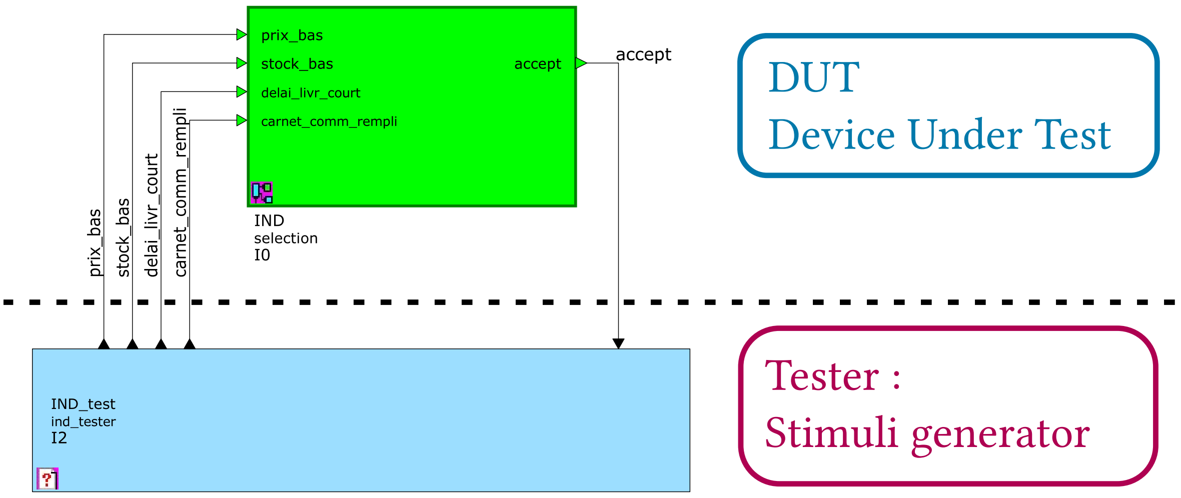


Abbildung 21: Test-bench

It consists of two parts :
- Device under test (DUT)
- Tester

3.1.1 Device under test (DUT)

The DUT refers to the design that has been developed and will be tested by the test-bench.

3.1.2 Tester

The tester is a VHDL block responsible for generating inputs for the DUT and verifying the correctness of its outputs.

3.2 Launch Simulation

Once the test bench is open, **`click`** on the `Performs generation and graphics files (Through Components)` button to perform the generation :

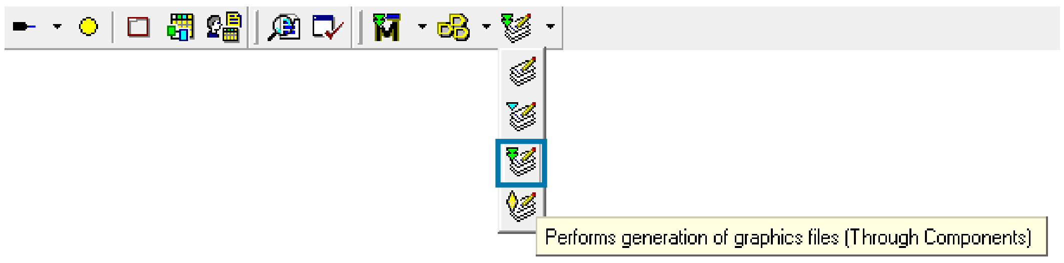


Abbildung 22: Generate VHDL

Then **`click`** on the `Generate and run entire ModelSim flow (Through Components)` button to launch the simulation :

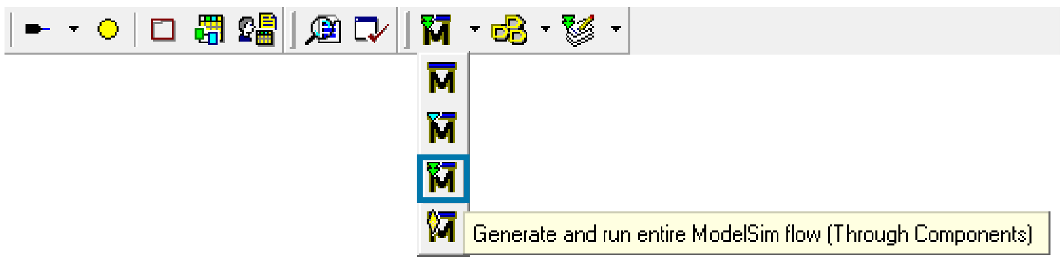
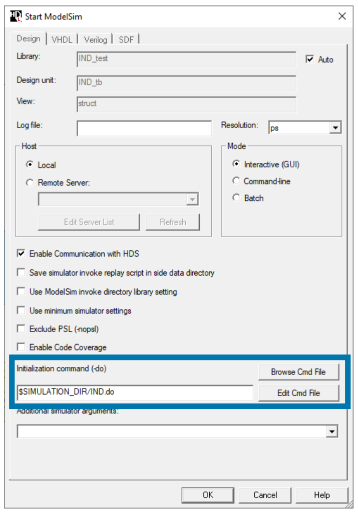


Abbildung 23: Launch ModelSim

It's **`mandatory`** to deselect all the component when generating the VHDL and launching ModelSim.
Always check that the performed operation displays the green arrow, else unroll the task and select its **Through Components** version.

When the generation and compilation are successful the following window pops up:

- **`Enter`** the `Initialization command file` (*.do)
- **`Click`** on **`OK`**



Start ModelSim

Design | VHDL | Verilog | SDF

Library: IND_test [✓ Auto]

Design unit: IND_tb

View: struct

Log file:

Resolution: ps

Host: [✓ Local] Remote Server:

Edit Server List | Refresh

Mode: [✓ Interactive (GUI)] Command-line Batch

[✓] Enable Communication with HDS

[ ] Save simulator invoke replay script in side data directory

[ ] Use ModelSim invoke directory library setting

[ ] Use minimum simulator settings

[ ] Exclude PSL (-nopsl)

[ ] Enable Code Coverage

Initialization command (-do): $SIMULATION_DIR/IND.do

Browse Cmd File | Edit Cmd File

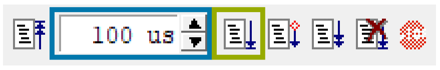
Additional simulator arguments:

OK | Cancel | Help

3.2.1 Restart and Run

Start Simulation

- **Enter** the simulation time in the **text box**
- **Click** on the **start simulation** button



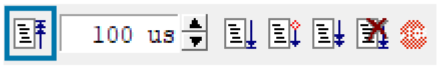
The table below illustrates the available units of time :

time unit	Equivalent in seconds (s)
ps	$10^{-12}s$
ns	$10^{-9}s$
us	$10^{-6}s$
ms	$10^{-3}s$
sec	$1s$

Restart

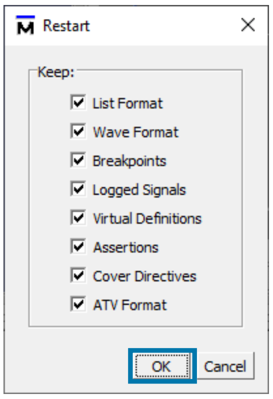
When a new wave is added, the simulation must be restarted.

- **Click** on the **restart simulation** button



The following window will pop-up :

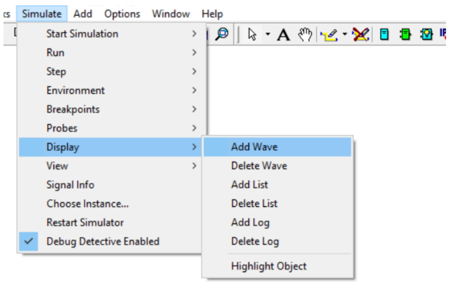
Click on the **OK** button

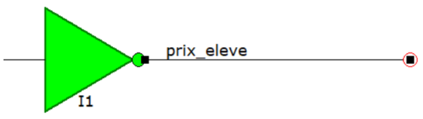


3.3 Signals

3.3.1 Add Signals from HDL Designer

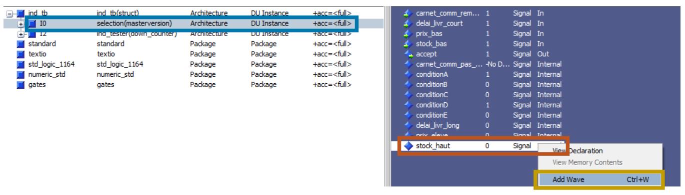
- **`Select`** the signal(s) to be added
- **`Click`** on **`Simulate`**
- Select **`Display`**
- **`Click`** on **`Add wave`**





3.3.2 Add Signals from Modelsim

1. **`Select`** the **`component`** containing the signal(s) to be added
2. **`Select`** the **`signal`** to be added
3. **`Right-click`** on the **`signal`**
4. **`Click`** on **`Add wave`**



3.3.3 Change Signals types and radix

The Abbildung 24 present the simulation of a counter.

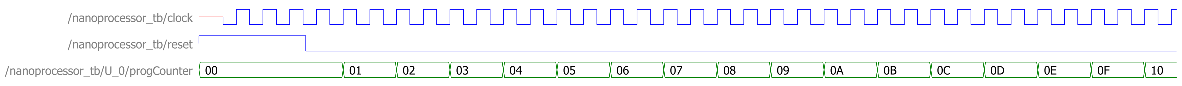
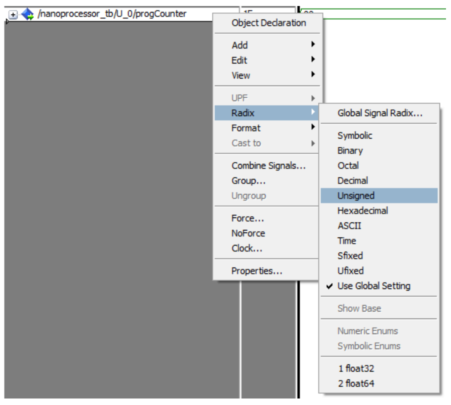


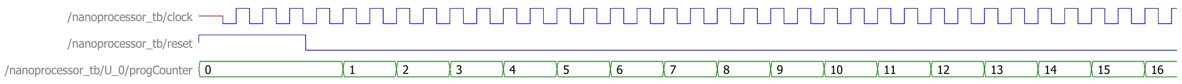
Abbildung 24: Counter simulation

The signal is currently represented in **`hexadecimal`**. This can be changed:

- **`Right-Click`** on the signal name
- **`Select`** Radix
- **`Select`** the new radix.

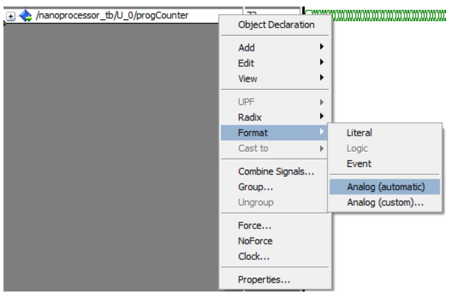


In that example, the radix has been switched to **`unsigned`**:

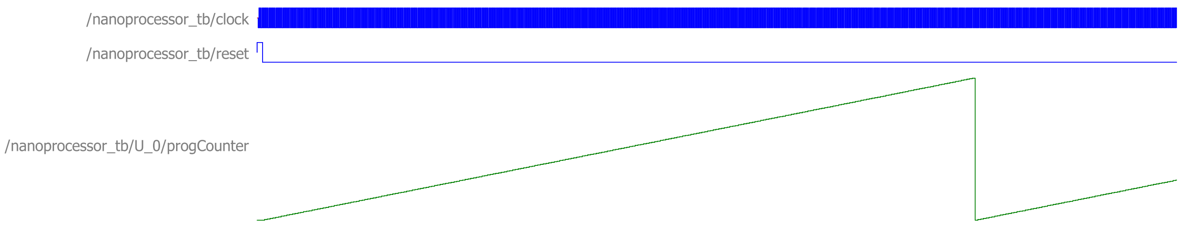


By default, a bus is represented in the **`literal`** format. This format can be changed:

- **`Right-Click`** on the signal name
- **`Select`** Format
- **`Select`** the new format.



In that example, the format has been switched to **`Analog`** :



3.4 Cursors

3.4.1 Add

To **`add`** a new cursor, **`click`** on the **`add cursor`** button.

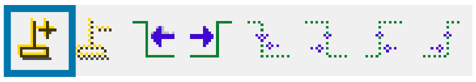


Abbildung 25: Add cursor

3.4.2 Remove

To **`remove`** the selected cursor, **`click`** on the **`remove cursor`** button.

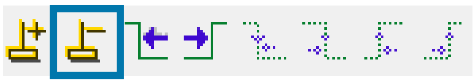


Abbildung 26: Remove cursor

3.4.3 Move

The following **`buttons`** can be used to move the selected cursor. First highlight the signal you want to move on in the simulation window, then:

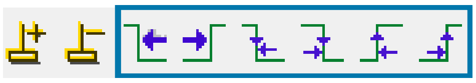


Abbildung 27: Move cursor

- Go to next/previous edge
- Go to next/previous rising edge
- Go to next/previous falling edge

3.4.4 Measure

When multiples cursors are added, the **delta time** between cursors will be displayed :

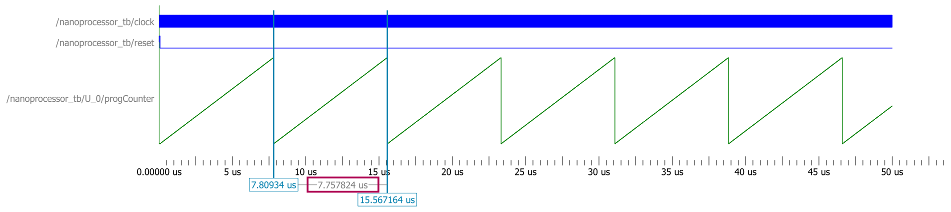
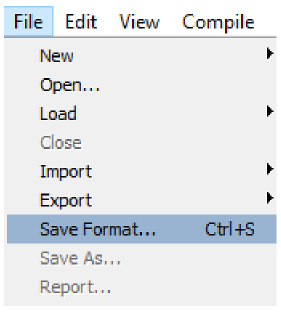


Abbildung 28: Delta time between cursors

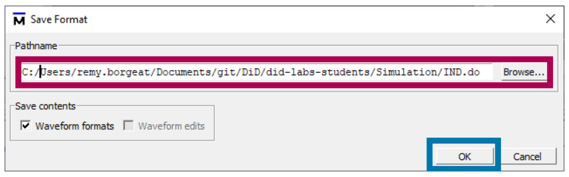
3.5 Save configuration

During debugging in ModelSim, you can add signals, modify the radix, add cursors, etc. This configuration can be saved and can be reloaded later.

- **`Click`** on **`File`** menu
- **`Click`** on **`Save Format...`**



- **`Enter`** the path under **`Path name`**.
- **`Click`** on **`OK`**



```
C:/Users/remy.borgeat/Documents/git/DiD/did-labs-students/Simulation/IND.do
```

Configuration files are usually saved in the **`simulation`** directory.

3.6 Print or Generate a PDF of the Waveforms

To achieve a well-formatted printout of the waves, begin by **`undocking`** the waves window by cli-cking on the **`undock`** button.

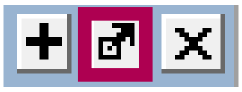


Abbildung 29: Undock button

Once the waves window unlocked, **`click`** on the **`print`** button.

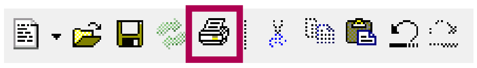
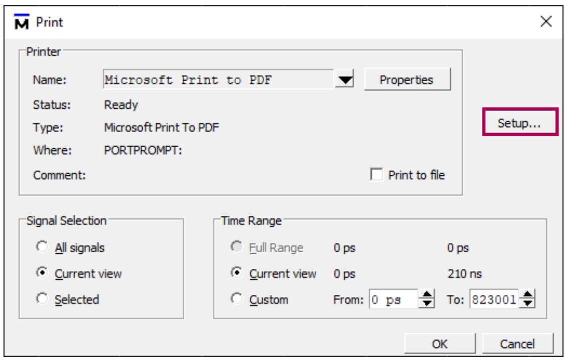


Abbildung 30: Undock button

The following window will pop-up:

To modify the printing setup, **`click`** on the **`Setup...`** button.

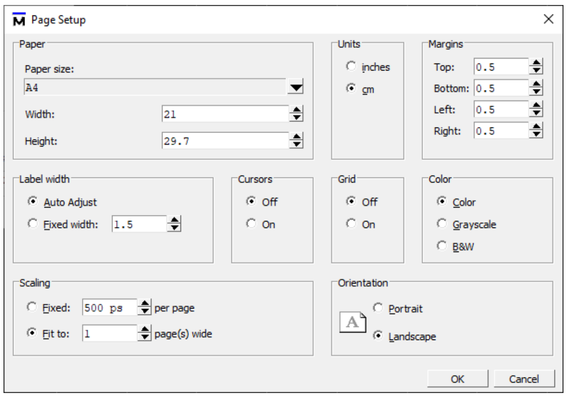

Print

Printer
Name: Microsoft Print to PDF
Status: Ready
Type: Microsoft Print To PDF
Where: PORTPROMPT:
Comment:
[ ] Print to file

Signal Selection: All signals [✓ Current view] Selected

Time Range:
Full Range 0 ps 0 ps
[✓ Current view] 0 ps 210 ns
Custom From: 0 ps To: 823001

Abbildung 31: Undock button


Page Setup

Paper
Paper size: A4
Width: 21
Height: 29.7

Units: inches [✓ cm]

Margins
Top: 0.5
Bottom: 0.5
Left: 0.5
Right: 0.5

Label width: [✓ Auto Adjust] Fixed width: 1.5

Cursors: [✓ Off] On

Grid: [✓ Off] On

Color: [✓ Color] Grayscale B&W

Scaling: Fixed: 500 ps per page [✓ Fit to: 1 page(s) wide]

Orientation: Portrait [✓ Landscape]

Abbildung 32: Undock button

The following options are recommended :

Parameter	**Value**
Label width	Auto Adjust
Cursors	Off
Grid	Off
Color	Color
Orientation	Landscape

Once the setup done, **`click`** on **`OK`** button. Then the page can be printed.

4 Conclusion

Congratulations on completing this guide to HDL Designer and ModelSim for EDA development. We hope this comprehensive resource has equipped you with the knowledge and skills necessary to navigate through the intricacies of these powerful tools seamlessly.

As you embark on your EDA journey, remember that HDL Designer and ModelSim serve as indispensable companions, offering a robust environment for designing, testing, and validating your hardware projects. Whether you are a novice or an experienced developer, mastering these tools will undoubtedly enhance your efficiency in FPGA and ASIC design. Feel free to revisiting this guide as you continue to work on your projects.

As technology continues to evolve, staying adept with cutting-edge EDA tools is crucial. Keep exploring, experimenting, and innovating to push the boundaries of what's possible in the exciting realm of hardware development.

Happy designing!