

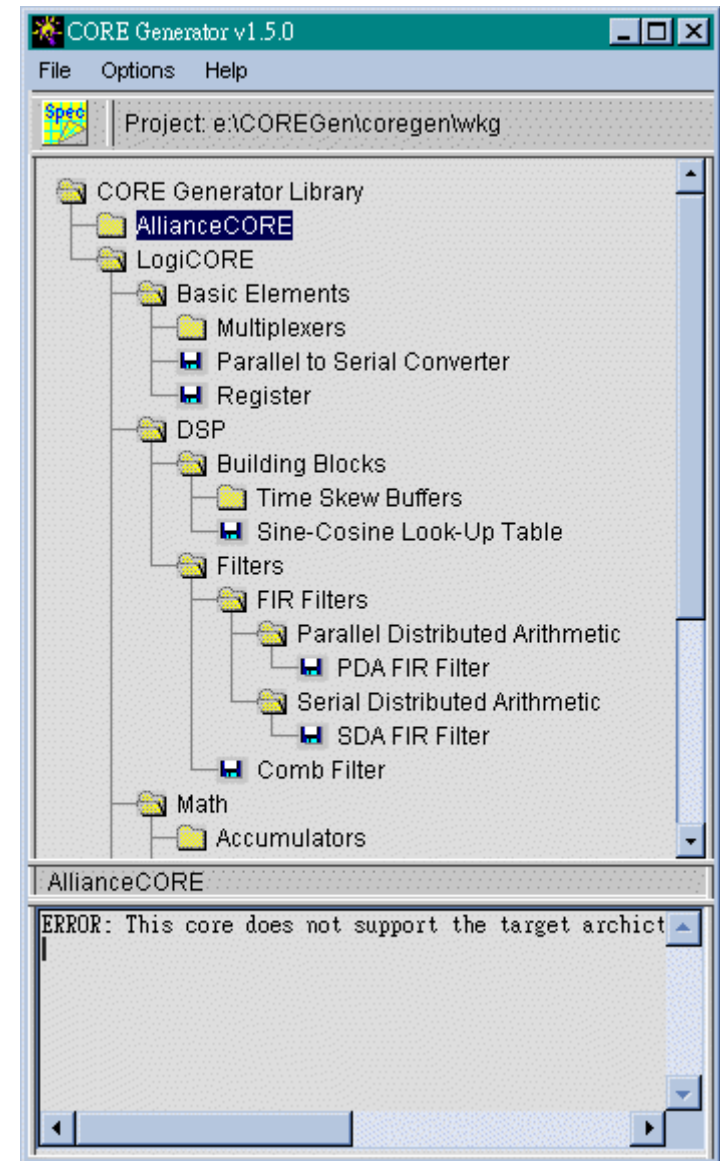
# Xilinx CORE Generator

# Overview I

- ◆ Parameterizable cores optimized for FPGAs.
- ◆ Tree-like presentation
- ◆ Deliver behavioral simulation models, schematic symbols and HDL instantiation templates for fitting design environment.
- ◆ Many functions can be used
  - **Basic Elements**: Multiplexers, Parallel to Serial Converter, Register
  - **DSP**: Time Skew Buffers, Sine-Cosine Lookup Table, PDA FIR Filter, SDA FIR Filter Single-Channel, Comb Filter

# Overview II

- **Math:** Accumulator, Adders/Subtractors, Complements, Constant Coefficient Multipliers, Parallel Multipliers, Integrator, square Root
- **Memories:** Delay Element, Registered DualPort RAM, Registered ROM, Registered SinglePort RAM, Synchronous FIFO



# How to Obtain New Cores and Updates

- ◆ New cores can be downloaded from the Xilinx web site and easily added to the CORE Generator.

- ◆ Bookmark

<http://www.xilinx.com/products/logicore/coregen>

and keep in touch regularly for updates.

# Project Management

- ◆ Setup the Coregen.ini File
- ◆ Set the CORE Generator System Options
- ◆ Set the CORE Generator Output Options

# Setup the Coregen.ini File

## ◆ coregen.ini File

- “coregen.ini” is created during the install of the CORE Generator and can be found in the following directory:  
<CORE\_Generator\_Install\_Path>/coregen/wkg
- Many parameters are set in this file

## ◆ The parameters expressed in the coregen.ini file:

- The following parameters are determined during installation, and usually do not need to be changed:
  - CoreGenPath, FoundationPath, AcrobatPath, and AcrobatName.
- Other settings will usually need to be changed by the user, such as
  - SelectedProducts, ProjectPath, BusFormat, and XilinxFamily.
- Changes to your output options and system options are best done through the CORE Generator *Output format* and *System Options forms*.
  - *Options* -> *Output format* for the Output format form
  - *Options* -> *System Options...* for the System Options form.

# Coregen.ini Syntax Summary

```
SET CoreGenPath = <some platform-specific path to the
                  $COREGEN\coregen directory>
SET FoundationPath = <some user-specific path to the Foundation tools>
SET ProjectPath = <some user-specific path to the current project>
SET TargetSymbolLibrary = <Viewlogic library name>
SET AcrobatPath = <some user-specific path to Acrobat install>
SET AcrobatName = <Acrobat executable name>
SET SelectedProducts = [ImpNetlist | FoundationSym| VHDLSym |
                       VHDLsim |ViewSym | VerilogSim | VerilogSym]
SET XilinxFamily = [All_XC4000_Families |All_Spartan_Families
                   |All_Virtex_Families]
SET BusFormat = [BusFormatAngleBracket | BusFormatSquareBracket
                |BusFormatParen | BusFormatNoBracket]
SET OverwriteFiles =[true | false]
```

# Sample coregen.ini file

```
SET FoundationPath = C:\FNDTN\ACTIVE  
  
SET ProjectPath = C:\cg95_cur\coregen\wkg  
  
SET TargetSymbolLibrary = primary  
  
SET XilinxFamily = All_XC4000_Families  
  
SET AcrobatPath = C:\Acrobat3\Reader\  
  
SET AcrobatName = AcroRd32.exe  
  
SET SelectedProducts = ImpNetlist FoundationSym
```

# Setup SelectedProducts I

- ◆ **SelectedProducts** - This setting defines the type of output files to be created each time a module is built.

```
SET SelectedProducts = ImpNetlist FoundationSym
```

- **ImpNetlist**: Implementation Netlist.
  - This is the gate level netlist that will be used to implement the logic of the particular core that the COREGenerator System has created.
  - In an HDL synthesis design flow, an HDL instantiation of the core references this netlist as a “black box” in a schematic design flow, a schematic symbol references this netlist.
- **ViewSym**: ViewLogic Schematic Symbol.
  - When specified this option will create a ViewLogic schematic symbol that can be used in your ViewLogic schematic capture tools to instantiate the module netlist.
- **FoundationSym**: Foundation Schematic Symbol.
  - When specified this option will create a Foundation schematic symbol that can be used in your Foundation schematic capture tools to instantiate the module netlist.

# Setup Selected Products II

- **VHDLSym**: VHDL Instantiation Template.
  - When specified this option will create a VHDL instantiation template that can be used in your HDL design capture tools to instantiate the module netlist.
- **VHDLSim**: VHDL Behavioral Simulation Model.
  - When specified this option will create a VHDL simulation model, which can be used to verify the functional simulation of the module netlist.
- **VerilogSym**: Verilog Instantiation Template.
  - When specified this option will create a Verilog instantiation template that can be used in your HDL design capture tools to instantiate the module netlist.
- **VerilogSim**: Verilog Behavioral Simulation Model.
  - When specified this option will create a Verilog simulation model, which can be used to verify the functional simulation of the module netlist.

# Set the CORE Generator System Options

## ◆ Options -> System Options ...

- All default settings are set in "coregen.ini"

## ◆ Project Path

- This setting defines the project working directory

## ◆ Viewlogic Library Alias

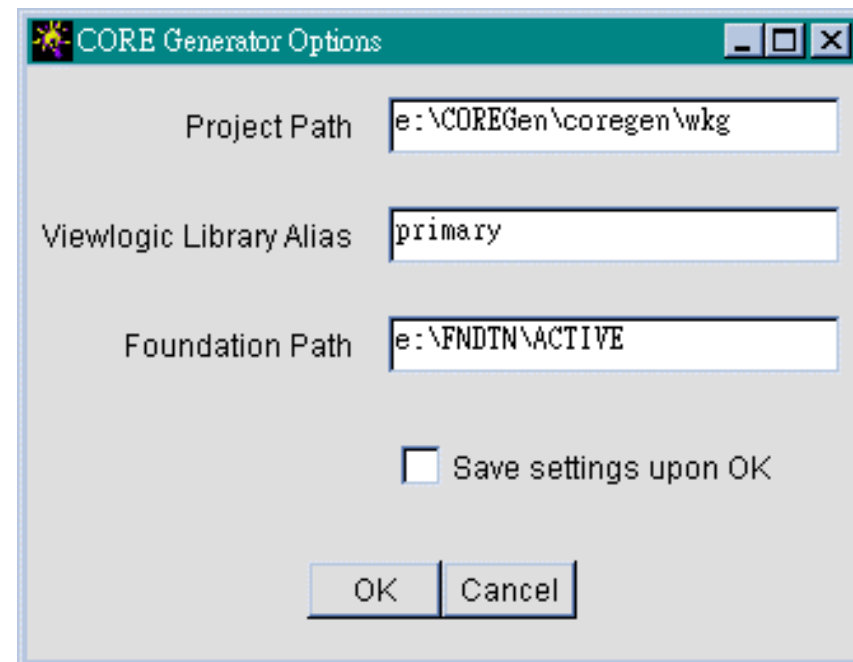
- This setting defines the name of the ViewLogic library alias.

## ◆ Foundation Path

- This setting defines the path location of the Foundation CAE tools

## ◆ Save settings upon OK

- Make these options permanent



# Set the CORE Generator Output Options I

## ◆ Options -> Output Format...

### ◆ Edif Implementation Netlist

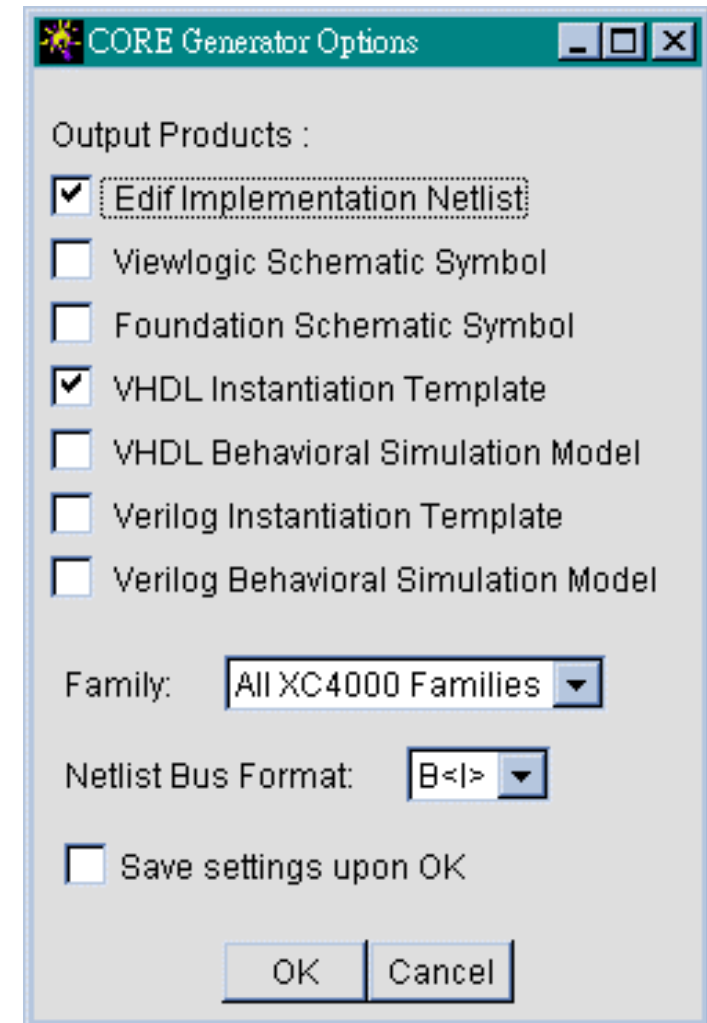
- A gate level netlist
- Output: `<CoreName>.edn`

### ◆ Viewlogic Schematic Symbol

- A Viewlogic schematic symbol and a simulation wire file
- Output: `wir\<CoreName>.1`  
`sym\<CoreName>.1`

### ◆ Foundation Schematic Symbol

- A Foundation schematic symbol and a simulation file
- Output: `<CoreName>.alr`  
`lib\project_name.sym`



# Set the CORE Generator Output Options II

## ◆ VHDL Instantiation Template

- A VHDL instantiation template that can be used in your HDL design capture tools to instantiate the module netlist.
- Output: `<CoreName>.vhi`

## ◆ VHDL Behavioral Simulation Model

- A VHDL simulation model which can be used to verify the functional simulation of the module netlist.
- This file is not intended to be synthesized.
- It is only provided for behavioral simulation
- Output: `<CoreName>.vhd`

## ◆ Verilog Instantiation Template

- A Verilog instantiation template that can be used in your HDL design capture tools to instantiate the module netlist.
- Output: `<CoreName>.vei`

# Set the CORE Generator Output Options III

## ◆ Verilog Behavioral Simulation Model

- A Verilog simulation model which can be used to verify the functional simulation of the module netlist.
- This file is not intended to be synthesized.
- It is only provided for behavioral simulation
- Output: `<CoreName>.v`

## ◆ The Family drop-down box

- Restrict the CORE Generator module browser to show only those modules that may be targeted to the selected family of devices.
- At this time the supported families of devices are
  - All XC4000 Families
  - All Spartan Families
  - All Virtex Families

# Using the CORE Generator

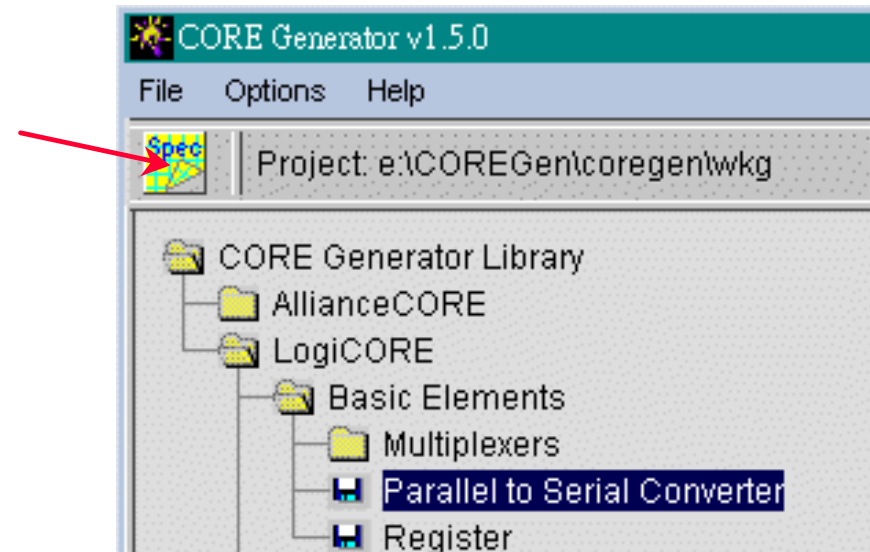
- ◆ Core Browser Tree
- ◆ Getting Module Data Sheets
- ◆ Parameterizing a Module
- ◆ COE Files

# Core Browser Tree

- ◆ The most common view of the CORE Generator is the **Core Browser** window.
- ◆ This window allows you to browse the many cores that are available from the CORE Generator installation.
- ◆ Cores that fall into particular application categories are grouped into folders to assist you in locating the module appropriate for your needs.
- ◆ To expand a folder, double click on the folder icon to the left of the folder name.

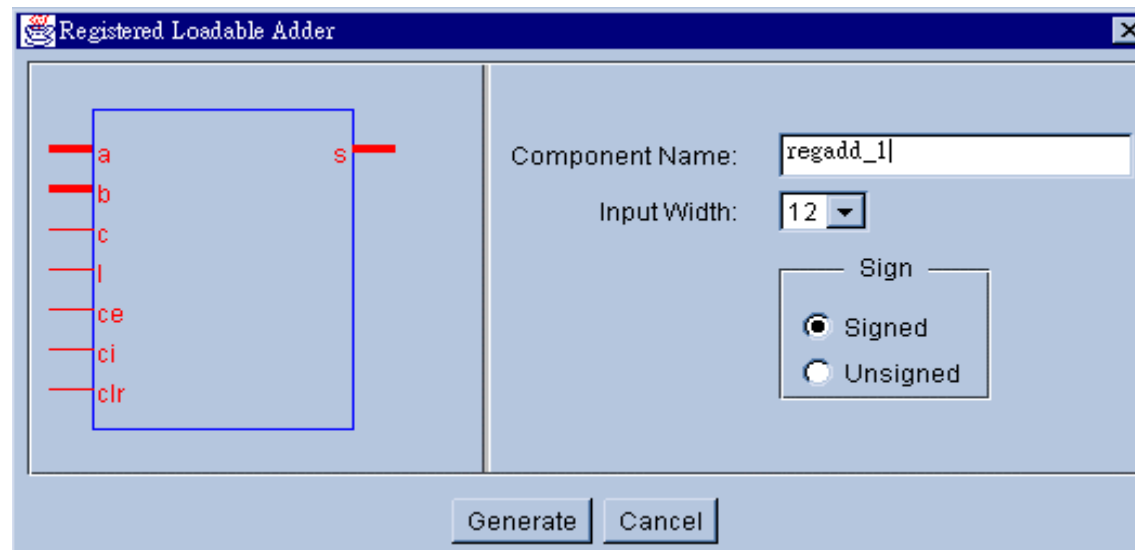
# Accessing Core Data Sheet

- ◆ A data sheet for the selected core can be requested at any time
  - 1. select the core in the core browser
  - 2. click on the **Spec** button on the CORE Generator toolbar.
  - This action will launch the Acrobat Reader application and display the core data sheet.



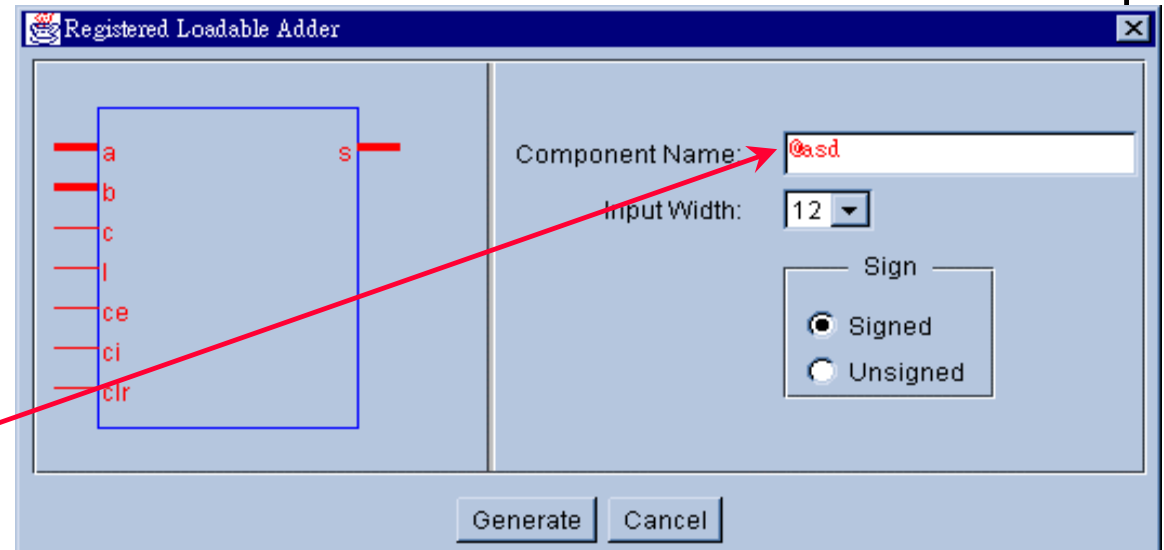
# Parameterizing a Core I

- ◆ Most cores have a parameterization window.
- ◆ Double-clicking on a core's icon or descriptive text will reveal the parameterization window for that module.
- ◆ For example : Registered Loadable Adder



# Parameterizing a Core II

- Component Name
  - allows you to assign a name to a module that you create.
  - Restrictions:
    - Up to 8 characters
    - No extensions
    - Must begin with a alpha character: a-z (No Capital letters)
    - May include (after the first character): 0-9, \_



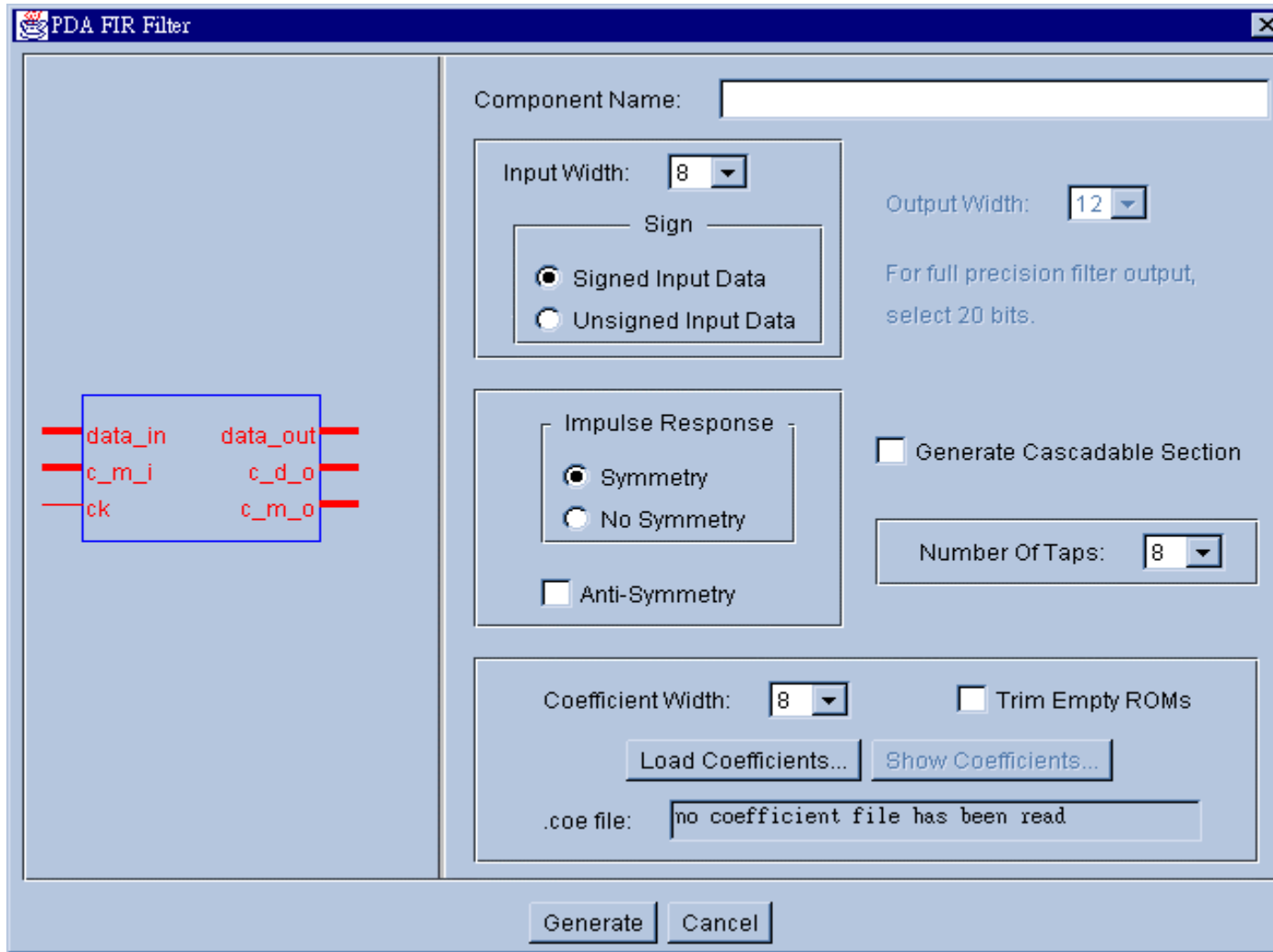
- The illegal or invalid field will be highlighted in red until the problem is corrected.

- ◆ Assuming there are no problems with any of the parameters that have been specified, pressing **Generate** will cause the CORE Generator to create files of the requested types.
- ◆ Pressing **Cancel** will return you to the module browser window without generating any files.

# COE Files

- ◆ Some cores can require a significant amount of information to completely specify their behavior.
- ◆ Cores of this type will have a button on their parameterization windows with which you can load their parameterization information from a file.
- ◆ Additional information about a particular Core's COE file can be found in that Core's datasheet.
  - For examples of PDA FIR, RAM, and ROM COE files, please look in the [coregen/wkg](#) directory.

# Example - PDA FIR Filter



# .COE Format

- ◆ The parameterization window for a FIR filter, which has a Load Coefficients..., button.
- ◆ Files containing this type of information should be ASCII text files and take the extension **.COE**.
- ◆ The format for the .COE file is illustrated below:

```
Keyword = Value ; Optional Comment
```

```
Keyword = Value ; Optional Comment
```

```
...
```

```
CoefData = Data_Value, Data_Value, ...;
```

- **CoefData** or **MemData** keywords must appear at the end of the file as any further keywords will be ignored.
- Any text after the semicolon is treated as a comment and will be ignored.

# .COE Examples - PDA FIR

```
***** EXAMPLE: PDA FIR *****  
component_name=fltr16;  
Number_of_taps=16;  
Input_Width = 8;  
Signed_Input_Data = true;  
Output_Width = 15;  
Coef_Width = 8;  
Symmetry = true;  
Radix = 10;  
Signed_Coefficient = true  
coefdata=1,-3,7,9,78,80,127,-128;
```

# .COE Examples - SDA FIR

```
***** EXAMPLE: SDA FIR *****  
component_name = sdafir;  
number_of_taps = 6;  
radix = 10;  
input_width = 10;  
output_width = 24;  
coef_width = 11;  
symmetry = false;  
coefdata = -1,18,122,418,-40,3;
```

# .COE Examples - RAM

```
***** EXAMPLE: RAM *****  
component_name=ram16x12;  
Data_Width = 12;  
Address_Width = 4;  
Depth = 16;  
Radix = 16;  
memdata=346,EDA,0D6,F91,079,FC8,053,FE2,03C,FF2,02D,  
FFB,022,002,01A,005;
```

# .COE Examples - ROM

```
***** EXAMPLE: ROM *****  
component_name=rom32x8;  
Data_Width = 8;  
Address_Width = 5;  
Depth = 32;  
Radix = 10;  
memdata=127,127,127,127,127,126,126,126,125,125,  
125,4,3,2,0,-1,-2,-4,-5,-6,-8,-9,-11,-12,-13,-38,  
-39,-41,-42,-44,-45,-128;
```

# .COE Examples - Virtex Block Ram

```
***** EXAMPLE: Virtex Block Ram *****  
component_name=blkram;  
Depth =256;  
Data_Width =32;  
Radix =16;  
Default_Data =FFF;  
MEMORY_INITIALIZATION_VECTOR =FF0,F0F,0FF,FF4,F4F,4FF,FF8,F8F,8FF;
```

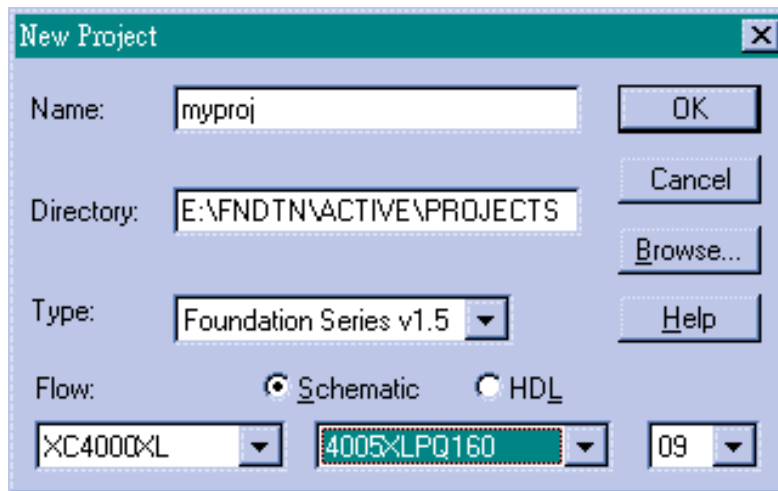
# **CORE Generator Design Flow**

**Foundation Schematic Flow  
and  
Foundation Express Flow**

# Foundation Schematic Flow I

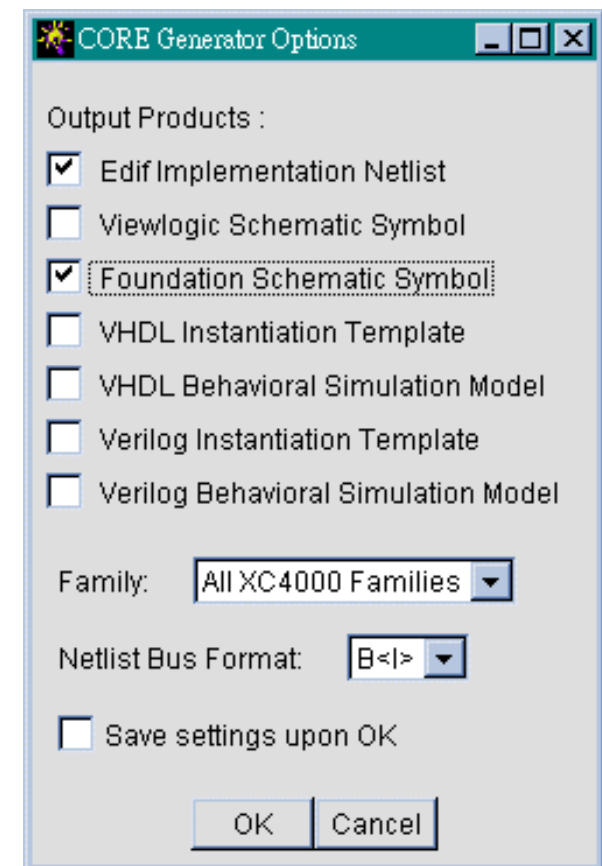
## ◆ 1. Set a Foundation Project

- Create a new project or Select an existing Schematic project from the Foundation Project Manager.



## ◆ 2. Set the CORE Generator Output Format

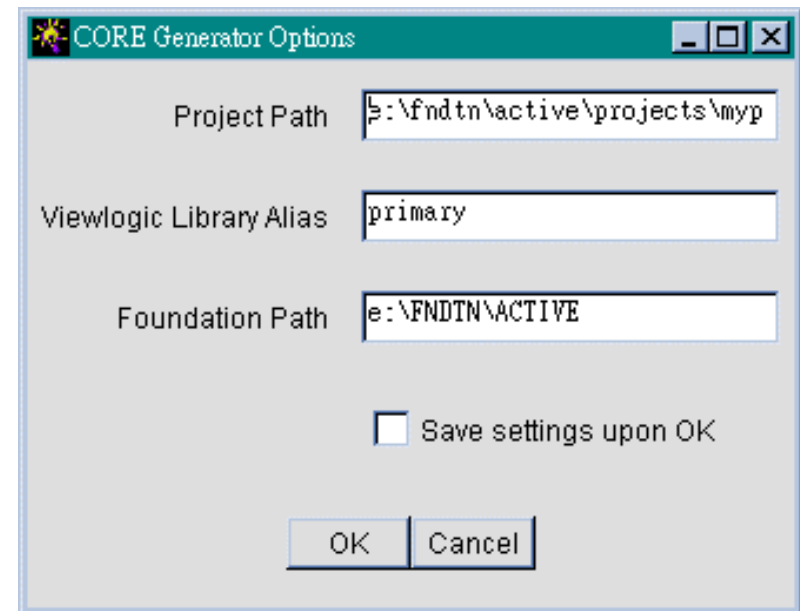
- From the CORE Generator **Options** menu, select **Output Format**, and check the following options:



# Foundation Schematic Flow II

## ◆ 3. Set the System Options

- From the CORE Generator Options menu, select **System Options**.
- Set the **Project Path** to point to your Foundation project directory.
  - The selected Project Path should be a valid Foundation project or an error will occur when generating the core.
  - This path should point to the directory where Foundation has been installed.



# Foundation Schematic Flow III

## ◆ 4. Select the module you want to generate by navigating to the desired module and clicking on it.

- Click on the SPEC button on the CORE Generator toolbar to review the module's datasheet
- Double-click on the selected module to call up its parameterization window.
- When you have entered all the parameterization details required by the module click the **Generate** button.

## ◆ 5. Output Files

- A Foundation symbol, a Netlist File (.**EDN**) and a simulation file (.**ALR**) are created.
- The symbol is automatically copied to the Foundation Project directory.

## ◆ 6. Load the Symbol in the Schematic Editor

- Open the Foundation schematic editor, the new symbols will be found in the symbol list for the selected project.
- The simulation and compilation flow is the same as the flow for a design containing only Unified Library components.

# Foundation HDL Flow I

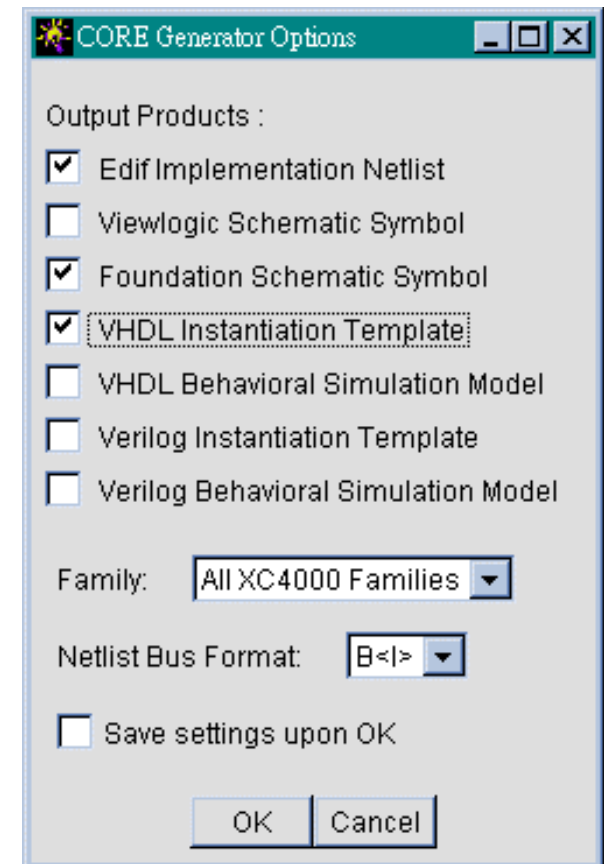
## ◆ 1. Set a Foundation Project

- Create a new project or Select an existing HDL Flow project from the Foundation Project Manager.
- The files generated by the CORE Generator System will automatically be copied into the selected project directory.

## ◆ 2. Set the CORE Generator Output Format

- From the CORE Generator Options menu, select **Output Format**, and check the following options:

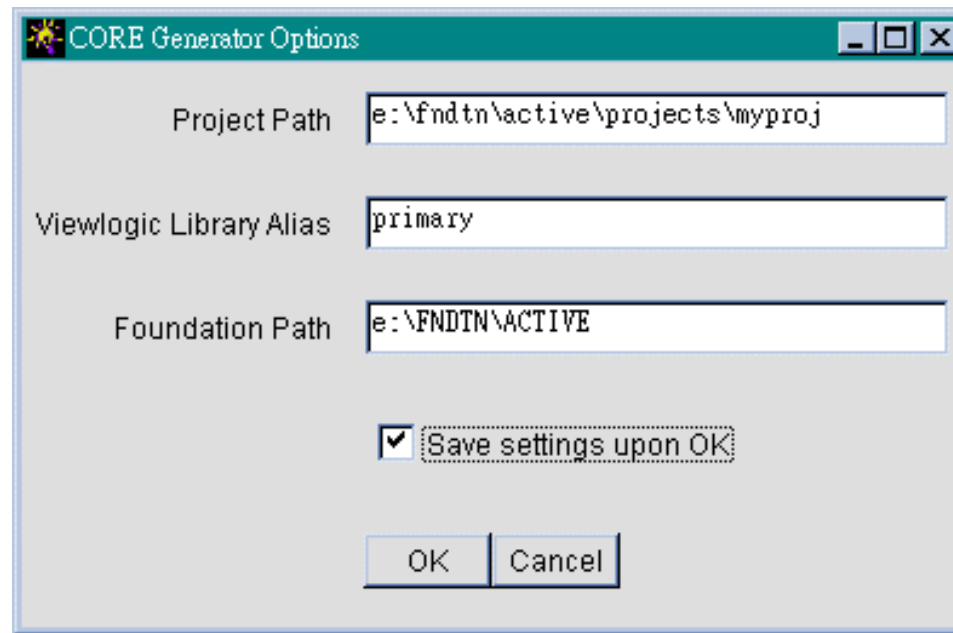
Select either VHDL or Verilog Instantiation template



# Foundation HDL Flow II

## ◆ 3. Set the System Options

- From the CORE Generator Options menu, select **System Options**.
- Set the **Project Path** to point to your Foundation HDL Flow project directory.
  - Enable the **“Save settings upon OK”** checkbox to save these settings for subsequent sessions if desired.



# Foundation HDL Flow III

## ◆ 4. Select the module you want to generate by navigating to the desired module and clicking on it.

- Click on the SPEC button on the CORE Generator toolbar to review the module's datasheet
- Double-click on the selected module to call up its parameterization window.
- When you have entered all the parameterization details required by the module click the **Generate** button.
- **Note: Do not name your Module with the same name as a Unified Library component as this will cause the Synthesizer to use the Unified Library XNF file instead of the EDIF file generated by the CORE Generator System.**

## ◆ 5. Output Files

- A VHDL or Verilog instantiation template (module\_name.**VHI** or module\_name.**VEI**) and a Netlist File (.**EDN**) will be created and copied into the CORE Generator project directory.
- The instantiation template contains the component declaration as well as the Port Map/Module declaration for the module that has been selected.
- This instantiation template can be copied and pasted into the top-level HDL file.

## VHDL Example - Instantiation template

**\*\* 8 Bit Adder VHDL Instantiation template: adder8.vhi\*\***

```
component adder8 port (  
    a: IN std_logic_VECTOR(7 downto 0);  
    b: IN std_logic_VECTOR(7 downto 0);  
    s: OUT std_logic_VECTOR(8 downto 0);  
    c: IN std_logic;  
    ce: IN std_logic;  
    ci: IN std_logic;  
    clr: IN std_logic);  
end component;
```

```
yourInstance : adder8 port map (  
    a => a,  
    b => b,  
    s => s,  
    c => c,  
    ce => ce,  
    ci => ci,  
    clr => clr);
```

\*\*\*\*\*

## VHDL Example - Top Level VHDL

```
Library IEEE;
use IEEE.std_logic_1164.all;
entity adder8_top is
    port (ina, inb: in STD_LOGIC_VECTOR (7 downto 0);
          clk, enable, carry, clear: in STD_LOGIC;
          qout: out STD_LOGIC_VECTOR (8 downto 0));
end adder8_top;
architecture BEHAV of adder8_top is
    -- Instantiate the adder8.edn file.
    component adder8 port (
        a: IN std_logic_VECTOR(7 downto 0);
        b: IN std_logic_VECTOR(7 downto 0);
        s: OUT std_logic_VECTOR(8 downto 0);
        c: IN std_logic;
        ce: IN std_logic;
        ci: IN std_logic;
        clr: IN std_logic);
    end component;
begin
    u1 : adder8 port map (
        a => ina,
        b => inb,
        s => qout,
        c => clk,
        ce => enable,
        ci => carry,
        clr => clear);
end BEHAV;
*****
```

## Verilog Example - Instantiation template

```
** 8 Bit Adder Verilog Instantiation template: adder8.vei **  
  
module adder8 ( a, b, s, c, ce, ci, clr);  
input [7:0] a;  
input [7:0] b;  
output [8:0] s;  
input c;  
input ce;  
input ci;  
input clr;  
endmodule  
  
// The following is an example of an instantiation :  
adder8 YourInstanceName (  
    .a(a),  
    .b(b),  
    .s(s),  
    .c(c),  
    .ce(ce),  
    .ci(ci),  
    .clr(clr));  
  
*****
```

## Verilog Example -

### Top Level Verilog

### Instantiation Module Declaration

```
***** Top Level Verilog file: adder8_top.v *****
module adder8_top(ina, inb, clk, enable, carry, clear, qout);
input [7:0] ina;
input [7:0] inb;
input clk;
input enable;
input carry;
input clear;
output [8:0] qout;
//instantiate the adder8.xnf file
adder8 U1 (
    .a(ina),
    .b(inb),
    .s(qout),
    .c(clk),
    .ce(enable),
    .ci(carry),
    .clr(clear));
endmodule
*****
***** Instantiation Module Declaration: adder8.v *****
module adder8 ( a, b, s, c, ce, ci, clr);
input [7:0] a;
input [7:0] b;
output [8:0] s;
input c;
input ce;
input ci;
input clr;
endmodule
*****
```

# Foundation HDL Flow IV

## ◆ 6. Compiling the Design in Foundation Express

- 1. Create a new project or open an existing one in Foundation Express.
- 2. Add all HDL files to be synthesized for the project.  
**Note: Do NOT add the EDIF files created by the CORE Generator System to the Express project. Also, do NOT add any HDL simulation files.**
- 3. **Verilog Only: Add a .v module declaration file for each instantiated block.**
- 4. Select the top level entity and select Create Implementation to generate a new implementation.
- 5. Optimize the implementation.
- 6. Write out the EDIF file for this implementation.
- 7. The EDIF file written by Express and the EDIF file(s) created by the CORE Generator System are required as inputs to the XACTstep M1 Implementation Tools, and should all be located in the same directory when 1 the design is input to M1.