

# Using Library Modules in Verilog Designs

This tutorial explains how Altera's library modules can be included in Verilog-based designs, which are implemented by using the Quartus II 9.1 software.

## **Contents:**

Example Circuit

Library of Parameterized Modules

Augmented Circuit with an LPM

Results for the Augmented Design

Practical designs often include commonly used circuit blocks such as adders, subtractors, multipliers, decoders, counters, and shifters. Altera provides efficient implementations of such blocks in the form of library modules that can be instantiated in Verilog designs. The compiler may recognize that a standard function specified in Verilog code can be realized using a library module, in which case it may automatically infer this module. However, many library modules provide functionality that is too complex to be recognized automatically by the compiler. These modules have to be instantiated in the design explicitly by the user. Quartus II 9.1 software includes a library of parameterized modules (LPM). The modules are general in structure and they are tailored to a specific application by specifying the values of general parameters.

Doing this tutorial, the reader will learn about:

- Library of parameterized modules (LPMs)
- Configuring an LPM for use in a circuit
- Instantiating an LPM in a designed circuit

The detailed examples in the tutorial were obtained using the Quartus II version 9.1, but other versions of the software can also be used.

## 1 Example Circuit

As an example, we will use the adder/subtractor circuit shown in Figure 1. It can add, subtract, and accumulate n-bit numbers using the 2's complement number representation. The two primary inputs are numbers  $A = a_{n-1} * a_{n-2} \dots a_0$  and  $B = b_{n-1} * b_{n-2} \dots b_0$ , and the primary output is  $Z = z_{n-1} * z_{n-2} \dots z_0$ . Another input is the AddSub control signal which causes  $Z = A + B$  to be performed when AddSub = 0 and  $Z = A - B$  when AddSub = 1. A second control input, Sel, is used to select the accumulator mode of operation. If Sel = 0, the operation  $Z = A \pm B$  is performed, but if Sel = 1, then B is added to or subtracted from the current value of Z. If the addition or subtraction operations result in arithmetic overflow, an output signal, Overflow, is asserted.

To make it easier to deal with asynchronous input signals, they are loaded into flip-flops on a positive edge of the clock. Thus, inputs A and B will be loaded into registers Areg and Breg, while Sel and AddSub will be loaded into flip-flops SelR and AddSubR, respectively. The adder/subtractor circuit places the result into register Zreg.

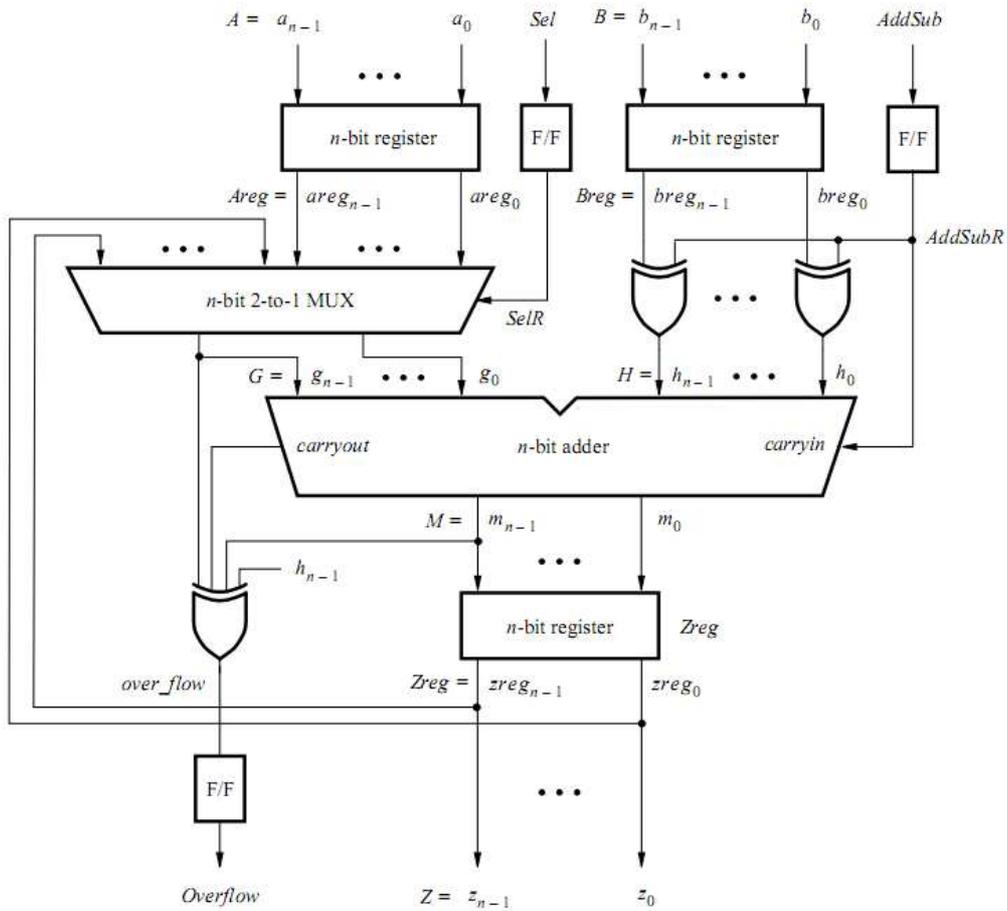


Figure 1. The adder/subtractor circuit.

The required circuit is described by the Verilog code in Figure 2. For our example, we use a 16-bit circuit as specified by  $n = 16$ . Implement this circuit as follows:

- Create a project addersubtractor.
- Include a file addersubtractor.v, which corresponds to Figure 2, in the project. For convenience, this file is provided in the directory DE2-115\_tutorials\design\_files, which is included on the CD-ROM that accompanies the DE2-115 board and can also be found on Altera's DE2-115 web pages.
- Choose the Cyclone IV EP4CE115F29C7 device, which is the FPGA chip on Altera's DE2-115 board.
- Compile the design.
- Simulate the design by applying some typical inputs.

```

// Top-level module
module addersubtractor (A, B, Clock, Reset, Sel, AddSub, Z, Overflow);
parameter n = 16;
input    [n-1:0] A, B;
input    Clock, Reset, Sel, AddSub;
output  [n-1:0] Z;
output  Overflow;
reg SelR, AddSubR, Overflow;
reg [n-1:0] Areg, Breg, Zreg;
wire[n-1:0] G, H, M, Z;
wirecarryout, over_flow;

// Define combinational logic circuit
assign H = Breg ^ {n{AddSubR}};
mux2to1 multiplexer (Areg, Z, SelR, G);
defparam multiplexer.k = n;
adderk nbit_adder (AddSubR, G, H, M, carryout);
defparam nbit_adder.k = n;
assign over_flow = carryout ^ G[n-1] ^ H[n-1] ^ M[n-1];
assign Z = Zreg;

// Define flip-flops and registers
always @(posedge Reset or posedge Clock)
if (Reset == 1)
    begin
        Areg <= 0;
        Breg <= 0;
        Zreg <= 0;
        SelR <= 0;
        AddSubR <= 0;
        Overflow <= 0;
    end
else
    begin
        Areg <= A;
        Breg <= B;
        Zreg <= M;
        SelR <= Sel;
        AddSubR <= AddSub;
        Overflow <= over_flow;
    end
endmodule

```

```

// k-bit 2-to-1 multiplexer
module mux2to1 (V, W, Selm, F);
parameter k = 8;
input    [k-1:0] V, W;
input    Selm;
output   [k-1:0] F;
reg     [k-1:0] F;

always @(V or W or Selm)
if (Selm == 0) F = V;
else F = W;
endmodule

```

... continued in Part b

Figure 2. Verilog code for the circuit in Figure 1 (Part a).

```

// k-bit adder
module adderk (carryin, X, Y, S, carryout);
parameter k = 8;
input    [k-1:0] X, Y;
input    carryin;
output   [k-1:0] S;
output   carryout;
reg [k-1:0] S;
reg carryout;

always @(X or Y or carryin)
{carryout, S} = X + Y + carryin;
endmodule

```

Figure 2. Verilog code for the circuit in Figure 1 (Part b).

## 2 Library of Parameterized Modules

The LPMs in the library of parameterized modules are general in structure and they can be configured to suit a specific application by specifying the values of various parameters. Select **Help > Megafunctions/LPM** to see a listing of the available LPMs. One of them is an adder/subtractor module called `lpm_add_sub` megafunction. Select this module to see its description. The module has a number of inputs and outputs, some of which may be omitted in a given application. Several parameters can be defined to specify a particular mode of operation.

For example, the number of bits in the operands is specified in the parameter `LPM_WIDTH`. The `LPM_REPRESENTATION` parameter specifies whether the operands are to be interpreted as signed or unsigned numbers, and so on. Templates on how an LPM can be instantiated in a hardware description language are given in the description of the module. Using these templates is somewhat cumbersome, so Quartus II software provides a wizard that makes the instantiation of LPMs easy.

We will use the `lpm_add_sub` module to simplify our adder/subtractor circuit defined in Figures 1 and 2. The augmented circuit is given in Figure 3. The `lpm_add_sub` module, instantiated under the name `megaddsub`, replaces the adder circuit as well as the **XOR** gates that provide the input `H` to the adder. Since arithmetic overflow is one of the outputs that the LPM provides, it is not necessary to generate this output with a separate **XOR** gate.

To implement this adder/subtractor circuit, create a new directory named `tutorial_lpm`, and then create a project `addersubtractor2`. Choose the same Cyclone IV EP4CE115F29C7 device, to allow a direct comparison of implemented designs.

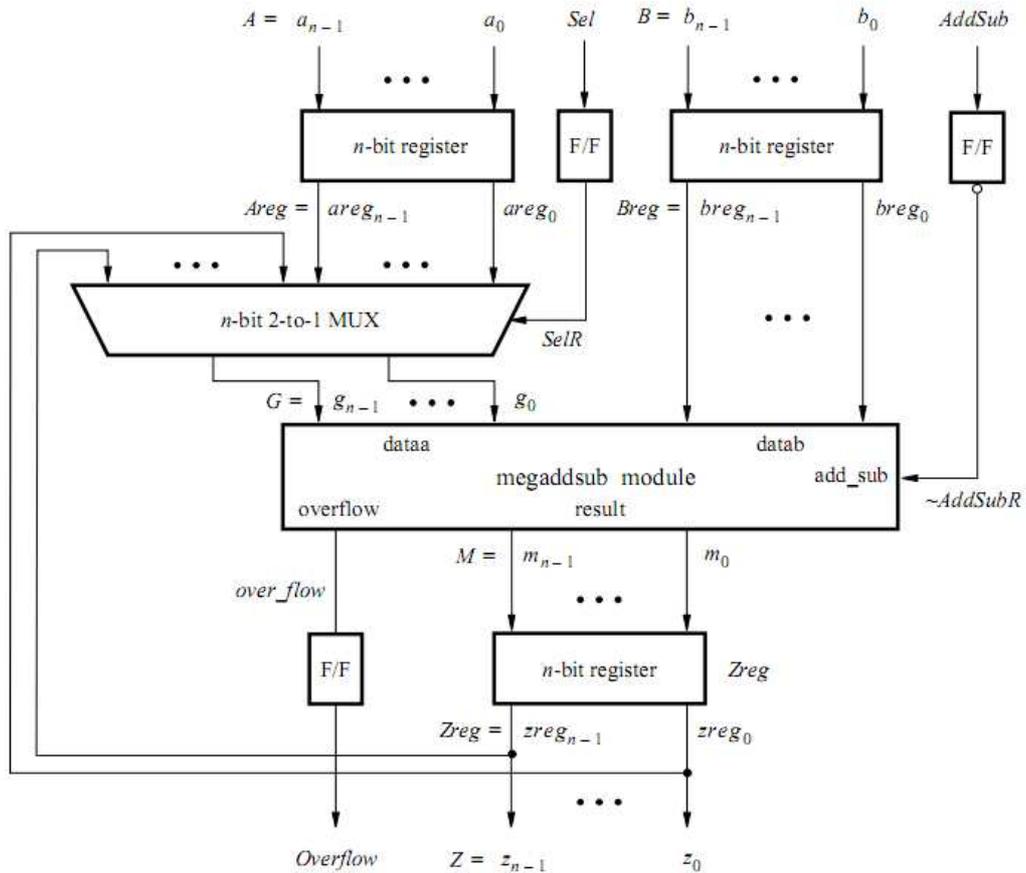


Figure 3. The augmented adder/subtractor circuit.

The new design will include the desired LPM subcircuit specified as a Verilog module that will be instantiated in the top-level Verilog design module. The Verilog module for the LPM subcircuit is generated by using a wizard as follows:

1. Select **Tools > MegaWizard Plug-in Manager**, which leads to a sequence of seven pop-up boxes in which the user can specify the details of the desired LPM.
2. In the box shown in Figure 4 indicate **Create a new custom megafunction variation** and click **Next**.

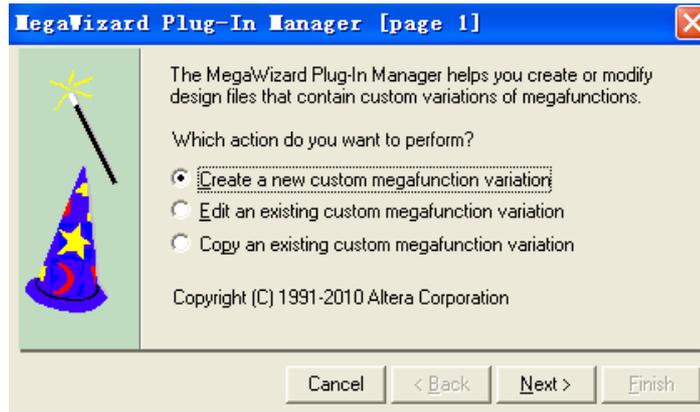


Figure 4. Choose to define an LPM.

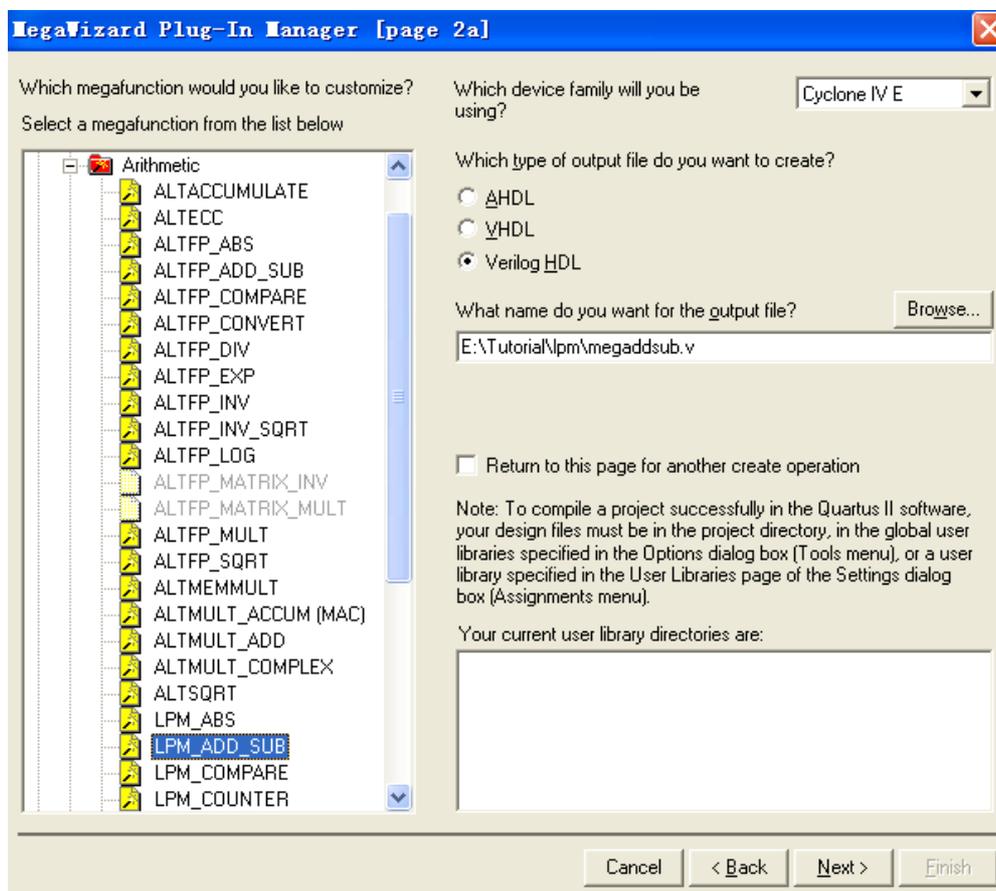


Figure 5. Choose an LPM from the available library.

3. The box in Figure 5 provides a list of the available LPMs. Expand the “arithmetic” sublist and select **LPM\_ADD\_SUB**. Choose **Verilog HDL** as the type of output file that should be created. The output file must be given a name; choose the name `megaddsub.v` and indicate that the file should be placed in the directory `E:\Tutorial\lpm` as shown in the figure. Press **Next**.

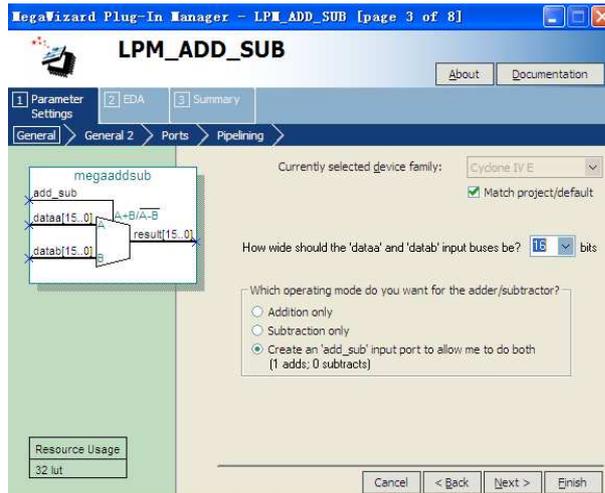


Figure 6. Specify the size of data inputs.

4. In the box in Figure 6 specify that the width of the data inputs is 16 bits. Also, specify the operating mode in which one of the ports allows performing both addition and subtraction of the input operand, under the control of the add\_sub input. A symbol for the resulting LPM is shown in the top left corner. Note that if add\_sub = 1 then result = A + B; otherwise, result = A – B. This interpretation of the control input and the operation performed is different from our original design in Figures 1 and 2, which we have to account for in the modified design. Observe that we have included this change in the circuit in Figure 3. Click **Next**.

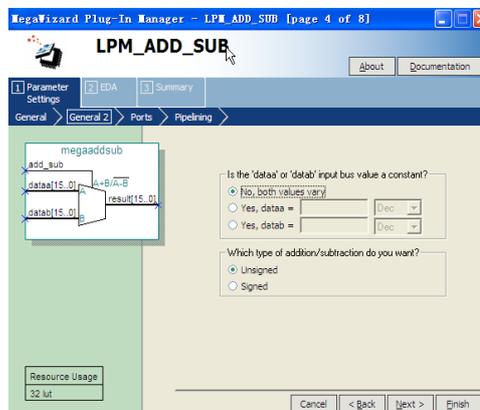


Figure 7. Further specification of inputs.

5. In the box in Figure 7, specify that the values of both inputs may vary and click **Next**.

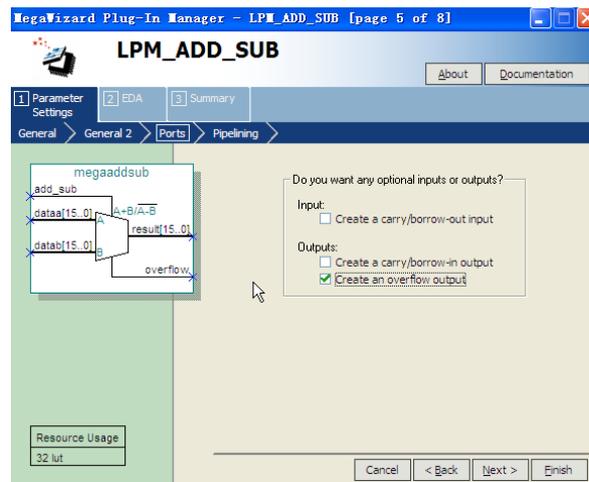


Figure 8. Specify the Overflow output.

6. The box in Figure 8 allows the designer to indicate optional inputs and outputs that may be specified. Since we need the overflow signal, make the Create an overflow output choice and press **Next**.

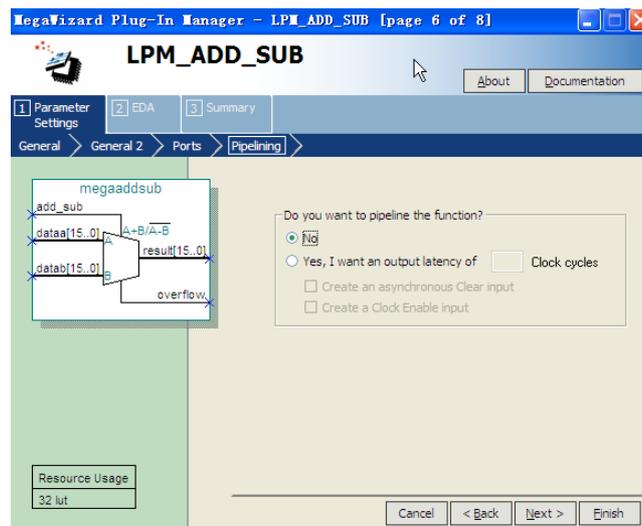


Figure 9. Refuse the pipelining option.

7. In the box in Figure 9 say No to the pipelining option and click **Next**.

8. Figure 10 gives a summary which shows the files that the wizard will create. Press **Finish** to complete the process.

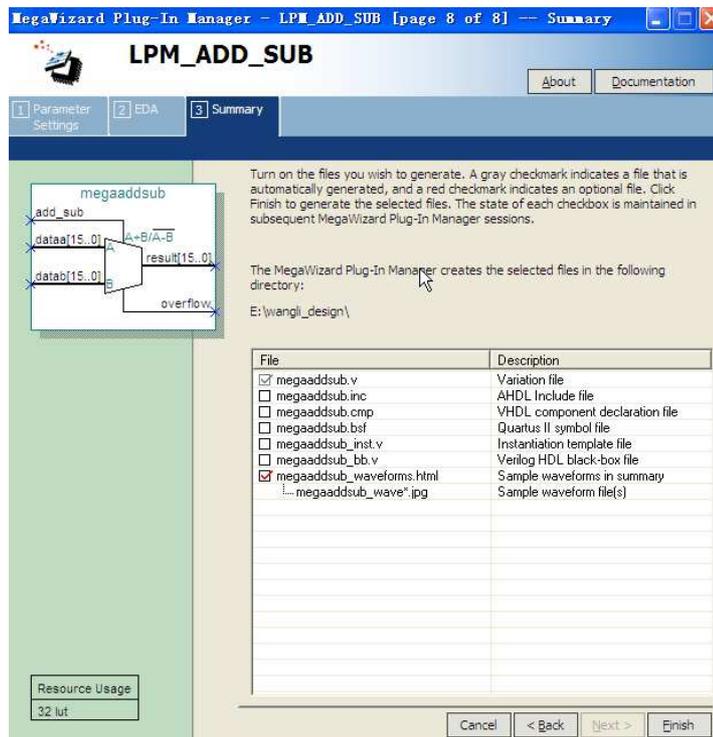


Figure 10. Files created by the wizard.

### 3 Augmented Circuit with an LPM

We will use the file megaddsub.v in our modified design. Figure 11 depicts the Verilog code in this file; note that we have not shown the comments in order to keep the figure small.

```

// Adder/subtractor module created by the MegaWizard
module megaddsub (
add_sub,
dataa,
datab,
result,
overflow);

input    add_sub;
input    [15:0] dataa;
input    [15:0] datab;
output   [15:0] result;
output   overflow;
wiresub_wire0;
wire[15:0] sub_wire1;
wireoverflow = sub_wire0;
wire[15:0] result = sub_wire1[15:0];

lpm_add_sub lpm_add_sub_component (
.dataa (dataa),
.add_sub (add_sub),
.datab (datab),
.overflow (sub_wire0),
.result (sub_wire1));
defparam
lpm_add_sub_component.lpm_width = 16,
lpm_add_sub_component.lpm_direction = "UNUSED",
lpm_add_sub_component.lpm_type = "LPM_ADD_SUB",
lpm_add_sub_component.lpm_hint = "ONE_INPUT_IS_CONSTANT=NO";
endmodule

```

Figure 11. Verilog code for the ADD\_SUB LPM.

The modified Verilog code for the adder/subtractor design is given in Figure 12. Put this code into a file tutorial\_lpm\For convenience, the required file addersubtractor2.v is provided in the directory DE2-115\_tutorials\design\_files, which is included on the CD-ROM that accompanies the DE2-115 board and can also be found on Altera's DE2-115 web pages. The differences between this code and Figure 2 are:

- The **assign** statements that define the over\_flow signal and the **XOR** gates (along with the signal defined as **wire H**) are no longer needed.

- The adderk instance of the adder circuit is replaced by megaddsub. Note that the dataa and datab inputs shown in Figure 6 are driven by the G and Breg vectors, respectively. Also, the inverted version of the AddSubR signal is specified to conform with the usage of this control signal in the LPM.
- The adderk module is deleted from the code.

```
// Top-level module
module addersubtractor2 (A, B, Clock, Reset, Sel, AddSub, Z, Overflow);
parameter n = 16;
input    [n-1:0] A, B;
input    Clock, Reset, Sel, AddSub;
output   [n-1:0] Z;
output   Overflow;
reg SelR, AddSubR, Overflow;
reg [n-1:0] Areg, Breg, Zreg;
wire[n-1:0] G, M, Z;
wireover_flow;

// Define combinational logic circuit
mux2to1 multiplexer (Areg, Z, SelR, G);
defparam multiplexer.k = n;
megaddsub nbit_adder (~AddSubR, G, Breg, M, over_flow);
assign Z = Zreg;

// Define flip-flops and registers
always @(posedge Reset or posedge Clock)
    if (Reset == 1)
        begin
            Areg <= 0; Breg <= 0; Zreg <= 0;
            SelR <= 0; AddSubR <= 0; Overflow <= 0;
        end
    else
        begin
            Areg <= A; Breg <= B; Zreg <= M;
            SelR <= Sel; AddSubR <= AddSub; Overflow <= over_flow;
        end
    end
endmodule

// k-bit 2-to-1 multiplexer
module mux2to1 (V, W, Selm, F);
parameter k = 8;
input    [k-1:0] V, W;
input    Selm;
output   [k-1:0] F;
```

```
reg [k-1:0] F;
```

```
always @(V or W or Selm)
```

```
    if (Selm == 0) F = V;
```

```
    else F = W;
```

```
endmodule
```

Figure 12. Verilog code for the circuit in Figure 3.

To include the megadbsub.v file in the project, select **Project > Add/Remove Files in Project** to reach the window in Figure 13. The file addersubtractor2.v should already be listed as being included in the project.

Browse for the other files by clicking the button **File name: ...** to reach the window in Figure 14. Select the file megadbsub.v and click **Open**, which returns to the window in Figure 13. Click **Add** to include the file and then click **OK**. Now, the modified design can be compiled and simulated in the usual way.

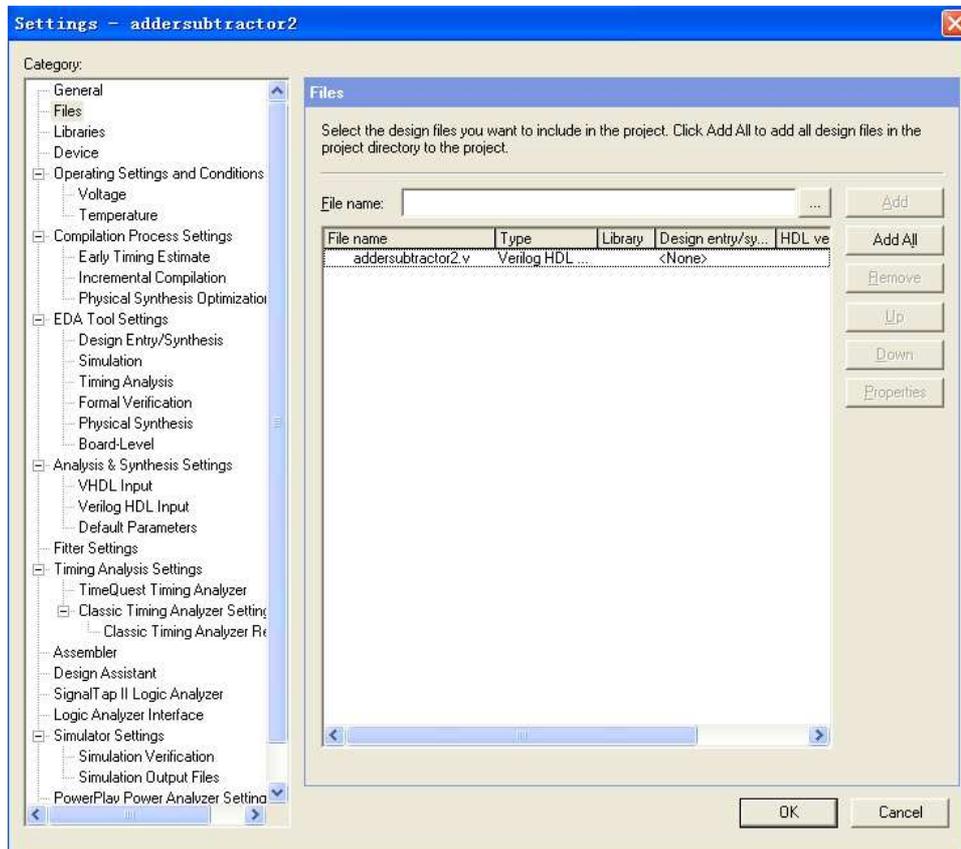


Figure 13. Inclusion of the new file in the project.



Figure 14. Specify the megaddsub.v file.

#### 4 Results for the Augmented Design

Compile the design and look at the summary, which is depicted in Figure 15. Observe that the modified design is implemented in 52 logic elements, which is the same as when using the code in Figure 2. In very small circuits which are the case with our example, it is unlikely that using LPMs will result in a significant advantage.

Flow Status	Successful - Wed Jul 07 21:10:31 2010
Quartus II Version	9.1 Build 350 03/24/2010 SP 2 SJ Full Version
Revision Name	adder_subtractor
Top-level Entity Name	adder_subtractor
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Preliminary
Met timing requirements	N/A
Total logic elements	52 / 114,480 ( < 1 % )
Total combinational functions	51 / 114,480 ( < 1 % )
Dedicated logic registers	51 / 114,480 ( < 1 % )
Total registers	51
Total pins	53 / 529 ( 10 % )
Total virtual pins	0
Total memory bits	0 / 3,981,312 ( 0 % )
Embedded Multiplier 9-bit elements	0 / 532 ( 0 % )
Total PLLs	0 / 4 ( 0 % )

Figure 15. Compilation Results for the Augmented Circuit.

Copyright ©2010 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, mask work rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

This document is being provided on an "as-is" basis and as an accommodation and therefore all warranties, representations or guarantees of any kind (whether express, implied or statutory) including, without limitation, warranties of merchantability, non-infringement, or fitness for a particular purpose, are specifically disclaimed.