Using Library Modules in Verilog Designs

For Quartus Prime 15.1

1 Introduction

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

Contents:

2 Background

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® Prime 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 Prime version 15.1, but other versions of the software can also be used. When selecting a device within Quartus Prime, use the device names associated with FPGA chip on the DE-series board by referring to Table 1.

Board	Device Name
DE0-CV	Cyclone V 5CEBA4F23C7
DE0-Nano	Cyclone IVE EP4CE22F17C6
DE0-Nano-SoC	Cyclone V SoC 5CSEMA4U23C6
DE1-SoC	Cyclone V SoC 5CSEMA5F31C6
DE2-115	Cyclone IVE EP4CE115F29C7

Table 1. DE-series FPGA device names

3 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 $A = a_{n-1}a_{n-2}\cdots a_0$ and $B = b_{n-1}b_{n-2}\cdots b_0$, and the primary output is $Z = z_{n-1}z_{n-2}\cdots z_0$. 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*.

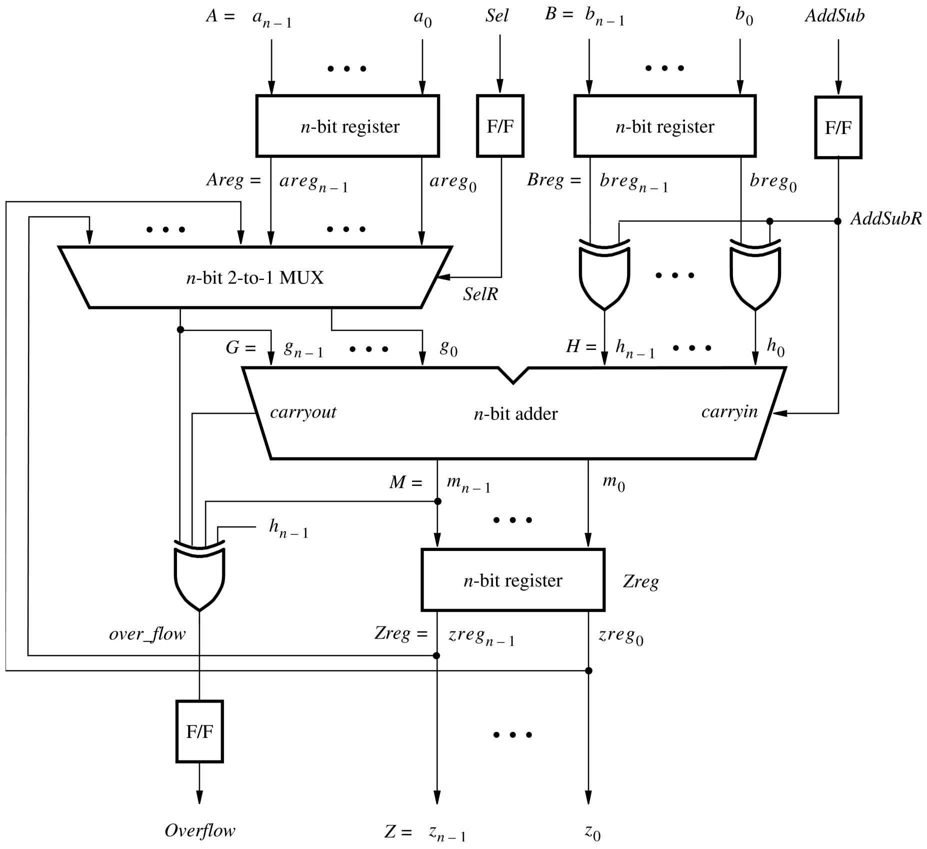


Figure 1. The adder/subtractor circuit.

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.
- Select the FPGA chip that is on the DE-series board. A list of device names on DE-series boards can be found in Table 1.
- 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;
   wire  carry_out, 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, Sel, F);
   parameter k = 8;
```

... continued in Part *b*

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

```
    input  [k-1:0] V, W;
    input  Sel;
    output  [k-1:0] F;
    reg  [k-1:0] F;

    always @(V or W or Sel)
        if (Sel == 0)
            F = V;
        else
            F = W;
endmodule
// 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*).

4 Library of Parameterized Modules

The LPMs in the IP Catalog are general in structure and they can be configured to suit a specific application by specifying the values of various parameters. 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 device as we previously selected (Refer to Table 1) to allow a direct comparison of implemented designs.

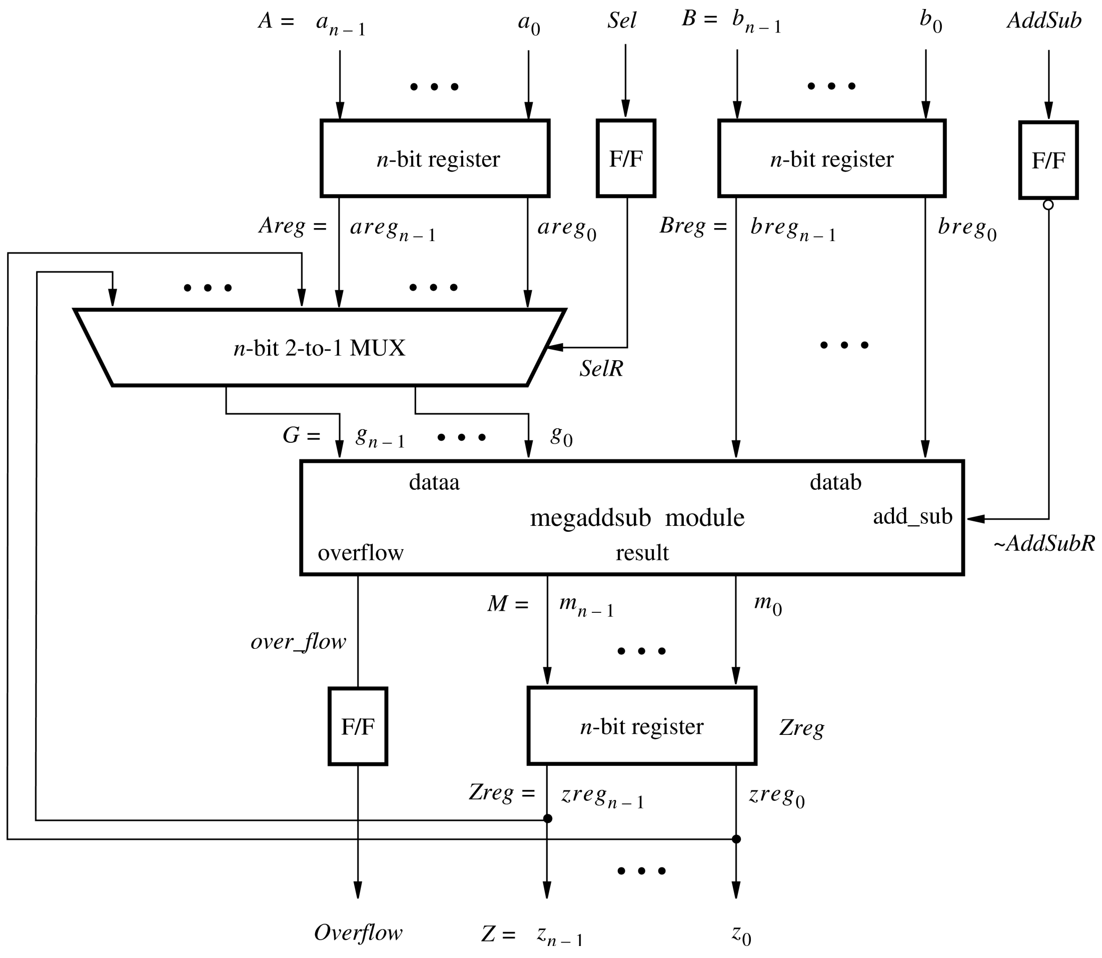


Figure 3. The augmented adder/subtractor circuit.

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 > IP Catalog, which opens the IP Catalog window in Figure 4.
2. In the IP Catalog panel, expand Library > Basic Functions > Arithmetic and double-click on LPM_ADD_SUB

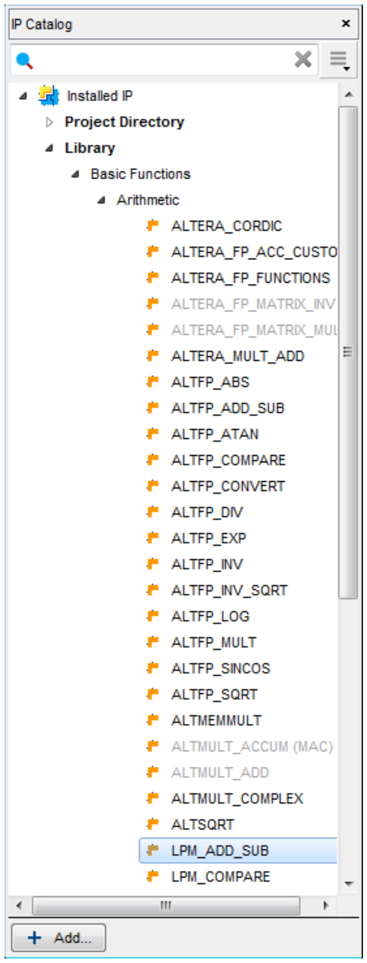
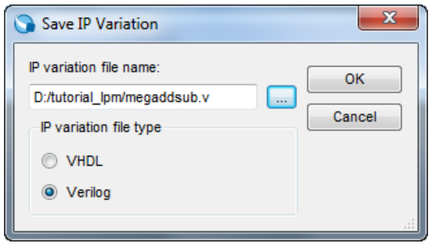


Figure 4. Choose an LPM.

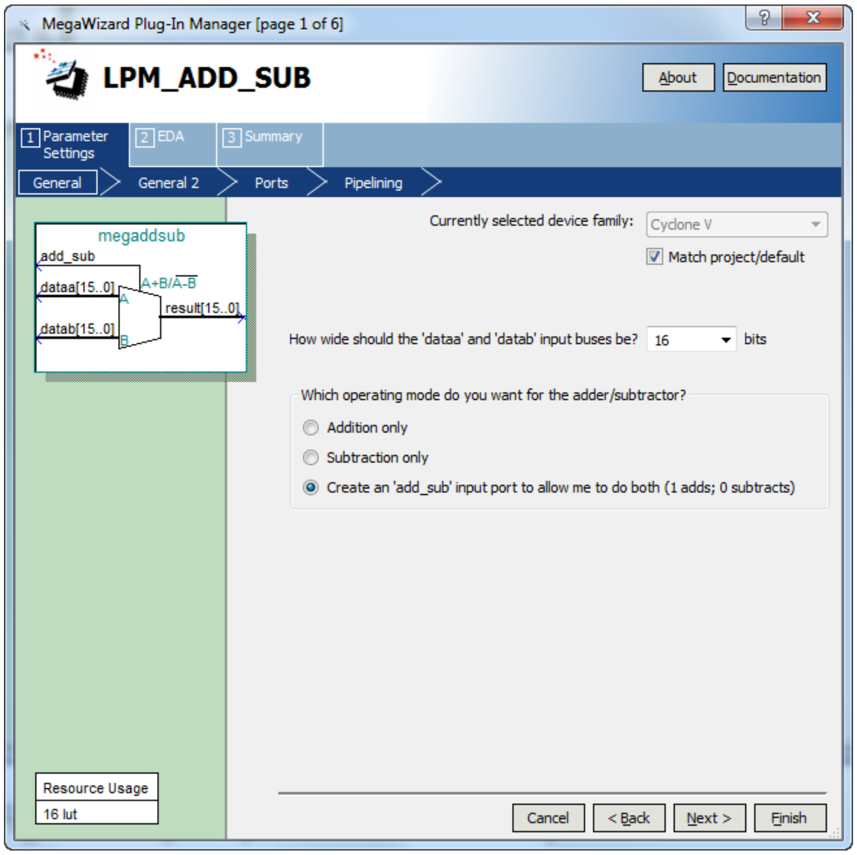


IP variation file name: D:/tutorial_lpm/megaddsub.v

IP variation file type: VHDL [✓ Verilog]

Figure 5. Create an LPM from the available library.

3. In the pop-up box shown in Figure 5, choose Verilog 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 *tutorial_lpm* as shown in the figure. Press Next.



Currently selected device family: Cyclone V

[✓] Match project/default

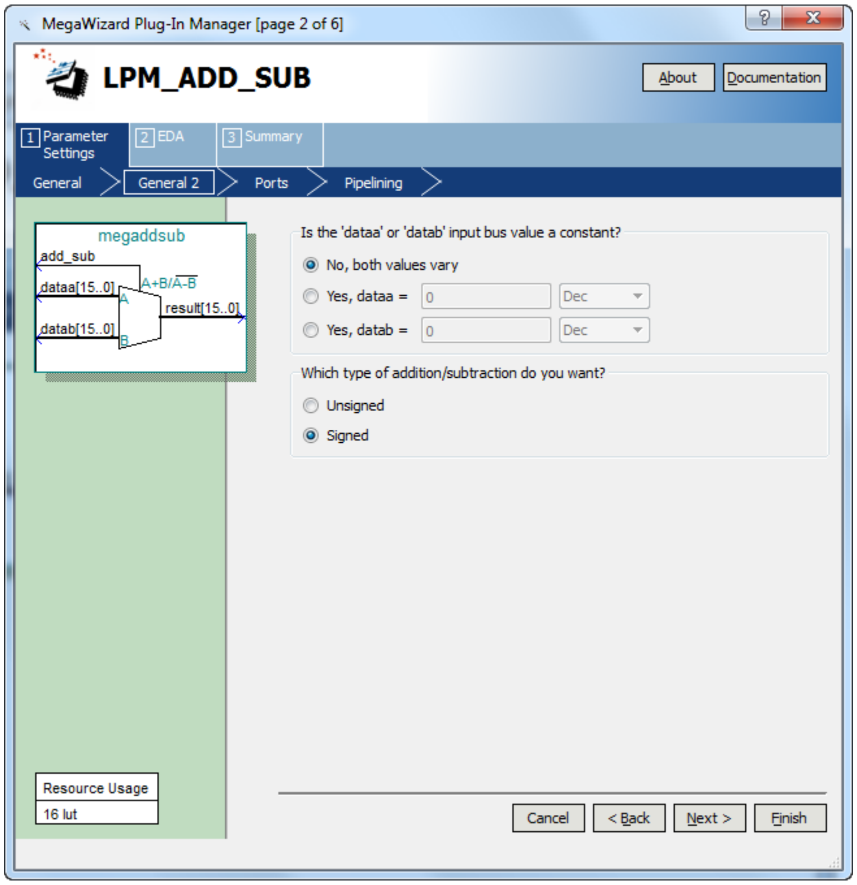
How wide should the 'dataa' and 'datab' input buses be? 16 bits

Which operating mode do you want for the adder/subtractor?

- Addition only
- Subtraction only
- [✓ Create an 'add_sub' input port to allow me to do both (1 adds; 0 subtracts)]

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.


MegaWizard Plug-In Manager [page 2 of 6]

LPM_ADD_SUB

Is the 'dataa' or 'datab' input bus value a constant?

- [✓] No, both values vary
- Yes, dataa = 0 Dec
- Yes, datab = 0 Dec

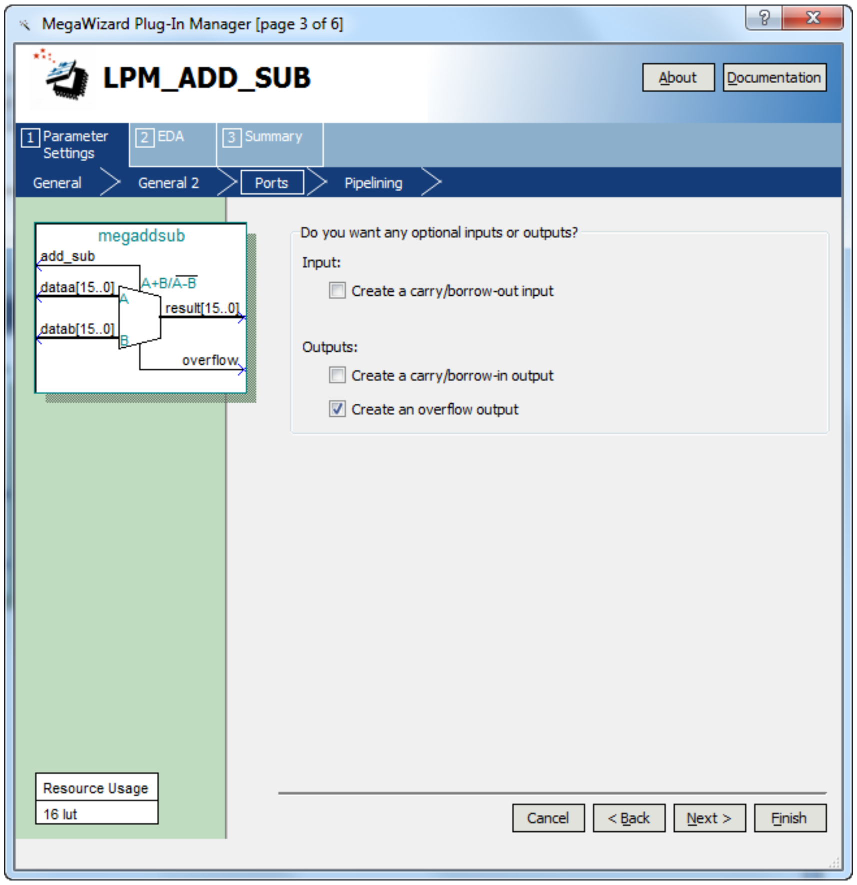
Which type of addition/subtraction do you want?

- Unsigned
- [✓] Signed

Resource Usage: 16 lut

Figure 7. Further specification of inputs.

5. In the box in Figure 7, specify that the values of both inputs may vary and select Signed for the type of addition/subtraction. Click Next.


MegaWizard Plug-In Manager [page 3 of 6]

LPM_ADD_SUB

Do you want any optional inputs or outputs?

Input:

- [ ] Create a carry/borrow-out input

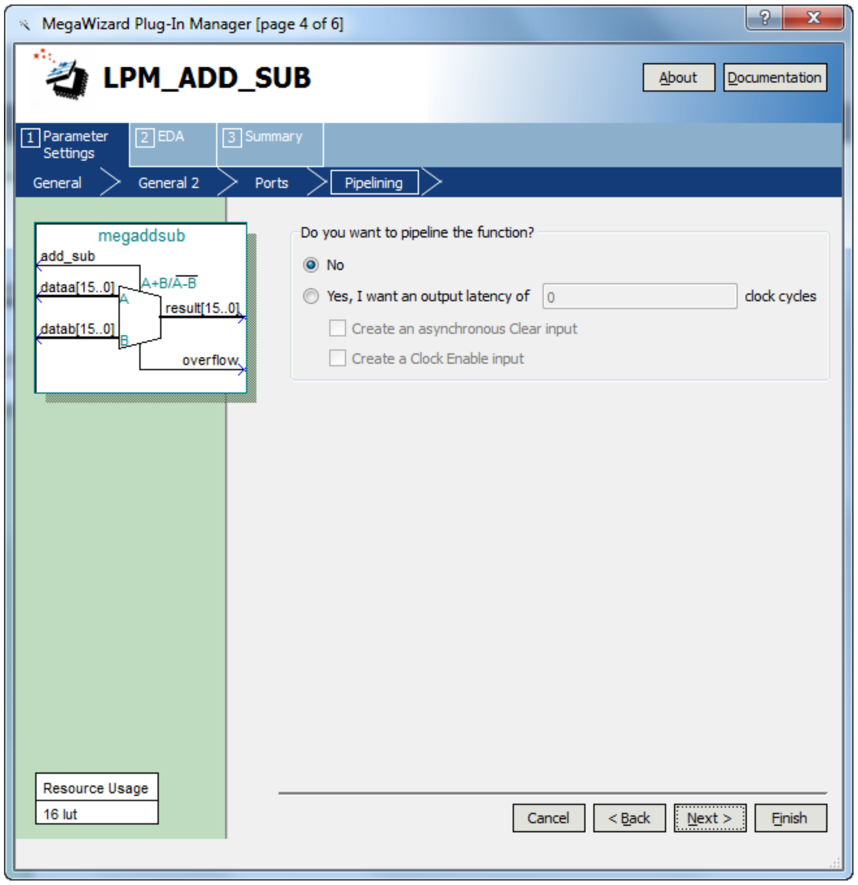
Outputs:

- [ ] Create a carry/borrow-in output
- [✓] Create an overflow output

Resource Usage
16 lut

Figure 8. Specify the Overflow output.

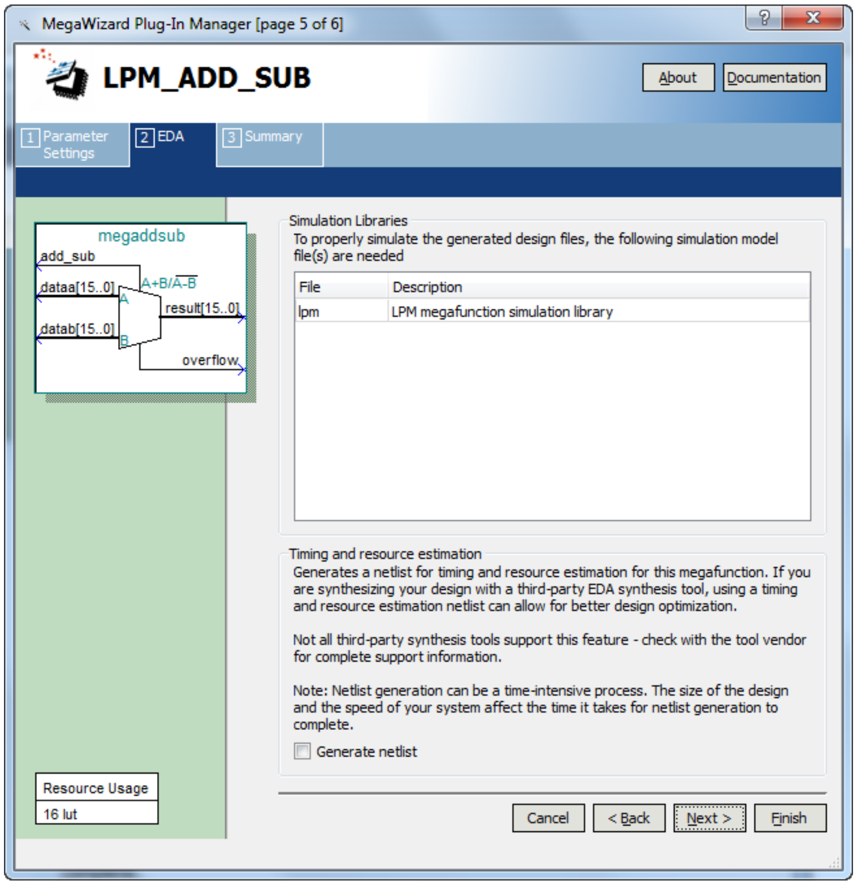
6. The box in Figure 8 allows the designer to indicate optional inputs and outputs that may be specified. Since we need the overflow signal, make the Create an overflow output choice and press Next.



Do you want to pipeline the function?
[✓ No]
Yes, I want an output latency of 0 clock cycles
Create an asynchronous Clear input
Create a Clock Enable input

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 shows the simulation model files needed to simulate the generated design. Press Next to proceed to the final page.



Resource Usage
16 lut

Simulation Libraries
To properly simulate the generated design files, the following simulation model file(s) are needed

File	Description
lpm	LPM megafunction simulation library

Timing and resource estimation
Generates a netlist for timing and resource estimation for this megafunction. If you are synthesizing your design with a third-party EDA synthesis tool, using a timing and resource estimation netlist can allow for better design optimization.

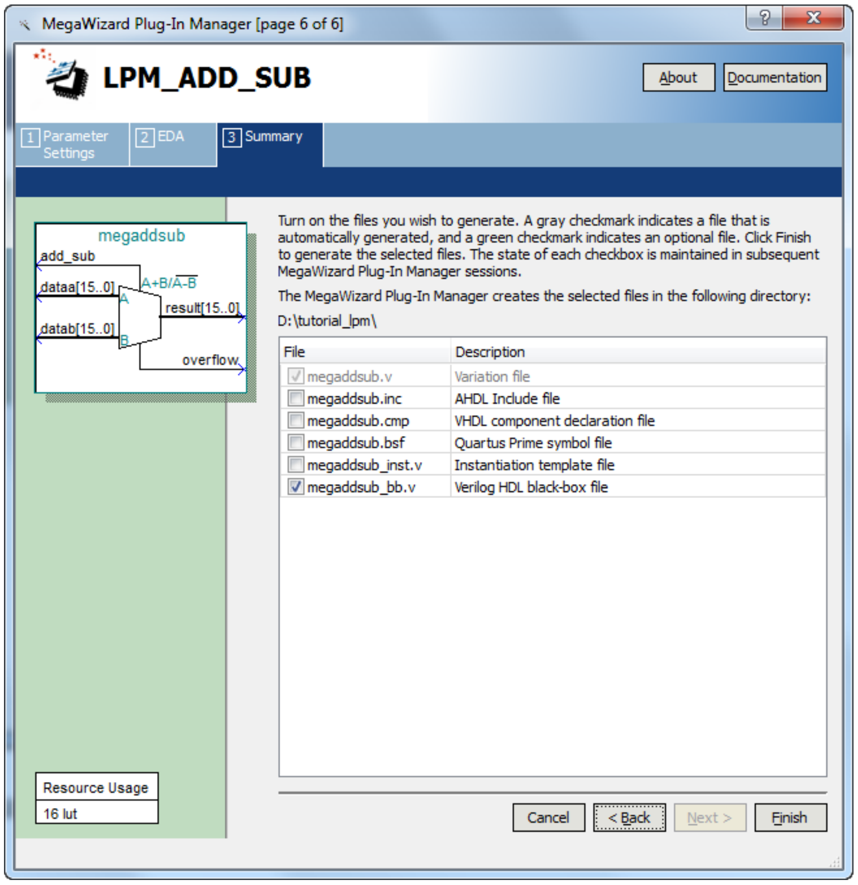
Not all third-party synthesis tools support this feature - check with the tool vendor for complete support information.

Note: Netlist generation can be a time-intensive process. The size of the design and the speed of your system affect the time it takes for netlist generation to complete.

☐ Generate netlist

Figure 10. Simulation model files.

9. Figure 11 gives a summary which shows the files that the wizard will create. Press Finish to complete the process.


MegaWizard Plug-In Manager [page 6 of 6]

LPM_ADD_SUB

Turn on the files you wish to generate. A gray checkmark indicates a file that is automatically generated, and a green checkmark indicates an optional file. Click Finish to generate the selected files. The state of each checkbox is maintained in subsequent MegaWizard Plug-In Manager sessions.

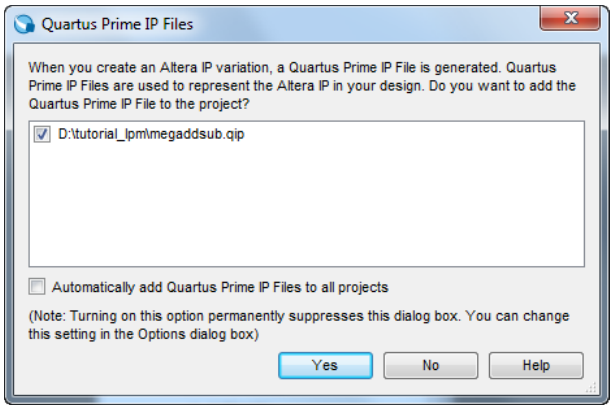
The MegaWizard Plug-In Manager creates the selected files in the following directory:

D:\tutorial_lpm\

File	Description
✓ megaddsub.v	Variation file
megaddsub.inc	AHDL Include file
megaddsub.cmp	VHDL component declaration file
megaddsub.bsf	Quartus Prime symbol file
megaddsub_inst.v	Instantiation template file
✓ megaddsub_bb.v	Verilog HDL black-box file

Figure 11. Files created by the wizard.

10. The box in Figure 12 may pop up. If it does, press Yes to add the newly generated files to the project.


Quartus Prime IP Files

When you create an Altera IP variation, a Quartus Prime IP File is generated. Quartus Prime IP Files are used to represent the Altera IP in your design. Do you want to add the Quartus Prime IP File to the project?

✓ D:\tutorial_lpm\megaddsub.qip

Automatically add Quartus Prime IP Files to all projects

(Note: Turning on this option permanently suppresses this dialog box. You can change this setting in the Options dialog box)

Figure 12. Add the new files to the project.

5 Augmented Circuit with an LPM

We will use the file *megaddsub.v* in our modified design. Figure 13 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,
        overflow,
        result);

    input  add_sub;
    input  [15:0] dataa;
    input  [15:0] datab;
    output  overflow;
    output  [15:0]  result;
    wire  sub_wire0;
    wire  [15:0] sub_wire1;
    wire  overflow = 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_direction = "UNUSED",
        lpm_add_sub_component.lpm_hint = "ONE_INPUT_IS_CONSTANT=NO,CIN_USED=NO",
        lpm_add_sub_component.lpm_representation = "SIGNED",
        lpm_add_sub_component.lpm_type = "LPM_ADD_SUB",
        lpm_add_sub_component.lpm_width = 16;
endmodule
```

Figure 13. Verilog code for the ADD_SUB LPM.

The modified Verilog code for the adder/subtractor design is given in Figure 14. Put this code into a file *addersubtractor2.v* under the directory *tutorial_lpm*. 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.

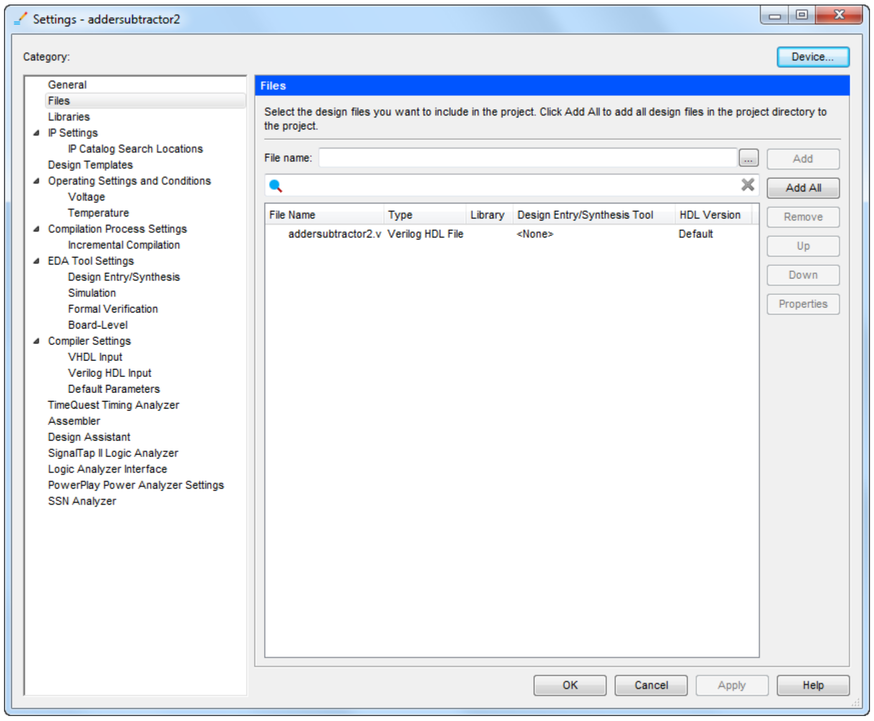
```
// Top-level module
module  addersubtractor2 (A, B, Clock, Reset, Sel, AddSub, Z, Overflow);
   parameter n = 16;
   input  [n-1:0] A, B;
   input  Clock, Reset, Sel, AddSub;
   output  [n-1:0]  Z;
   output  Overflow;
   reg  SelR, AddSubR, Overflow;
   reg  [n-1:0] Areg, Breg, Zreg;
   wire  [n-1:0]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

... continued in Part b
```

Figure 14. Verilog code for the circuit in Figure 3 (Part *a*)

```
            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 14. Verilog code for the circuit in Figure 3 (Part *b*).

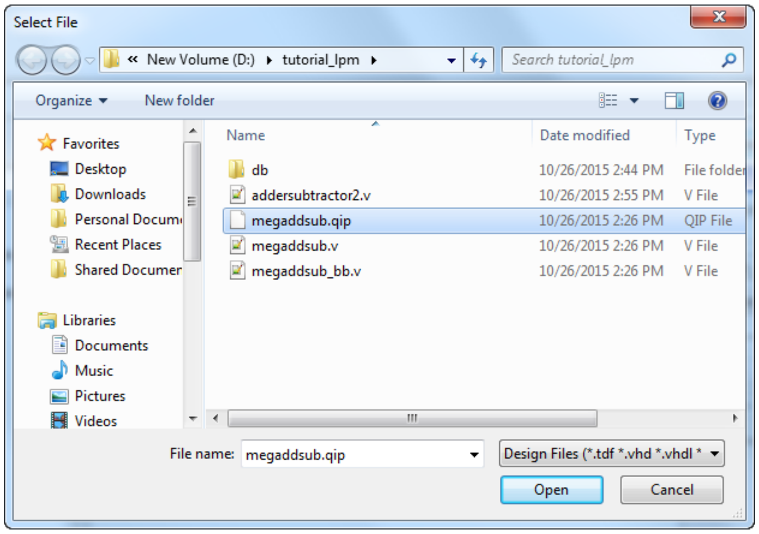
If the *megaddsub.qip* file has not been included in the project (e.g. if you answered No in the box in Figure 12, or possibly if the box did not show up at all), you need to include it manually. To include the *megaddsub.v* file in the project, select Project > Add/Remove Files in Project to reach the window in Figure 15. The file *addersubtractor2.v* should already be listed as being included in the project. Browse for the other files by clicking the button ... to reach the window in Figure 16. Select the file *megaddsub.qip* and click Open, which returns to the window in Figure 15. Click Add to include the file and then click OK. Now, the modified design can be compiled and simulated in the usual way.



Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project.

File Name	Type	Library	Design Entry/Synthesis Tool	HDL Version
addersubtractor2.v	Verilog HDL File		<None>	Default

Figure 15. Inclusion of the new file in the project.



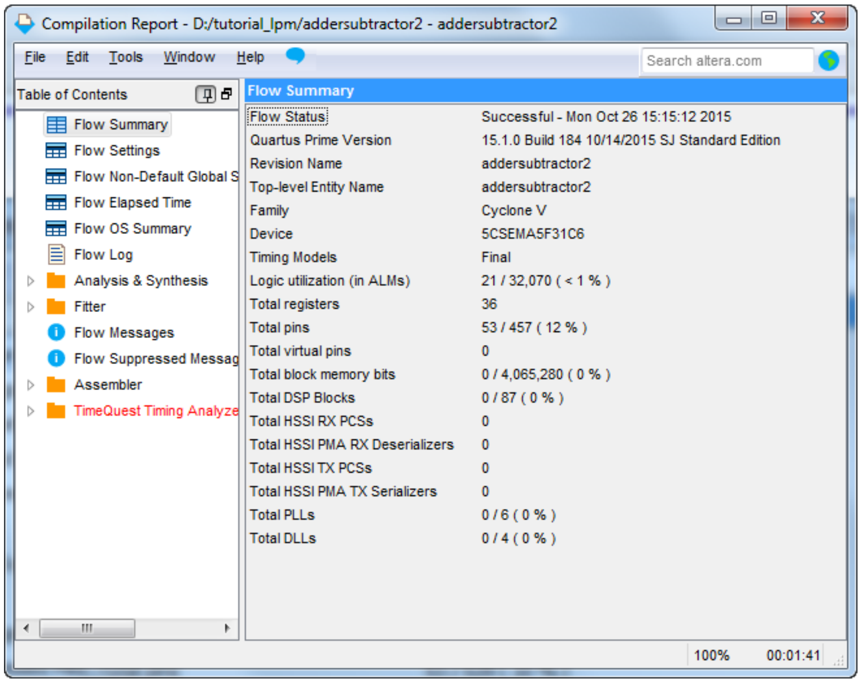
Name	Date modified	Type
db	10/26/2015 2:44 PM	File folder
addersubtractor2.v	10/26/2015 2:55 PM	V File
megaddsub.qip	10/26/2015 2:26 PM	QIP File
megaddsub.v	10/26/2015 2:26 PM	V File
megaddsub_bb.v	10/26/2015 2:26 PM	V File

File name: megaddsub.qip
Design Files (*.tdf *.vhd *.vhdl *

Figure 16. Specify the *megaddsub.qip file.*

6 Results for the Augmented Design

Compile the design and look at the summary, which is depicted in Figure 17. Observe that the modified design is implemented with a similar number of logic elements compared to using the code in Figure 2.



Flow Summary

Flow Status	Successful - Mon Oct 26 15:15:12 2015
Quartus Prime Version	15.1.0 Build 184 10/14/2015 SJ Standard Edition
Revision Name	addersubtractor2
Top-level Entity Name	addersubtractor2
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	21 / 32,070 (< 1 %)
Total registers	36
Total pins	53 / 457 (12 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 17. Compilation Results for the Augmented Circuit.