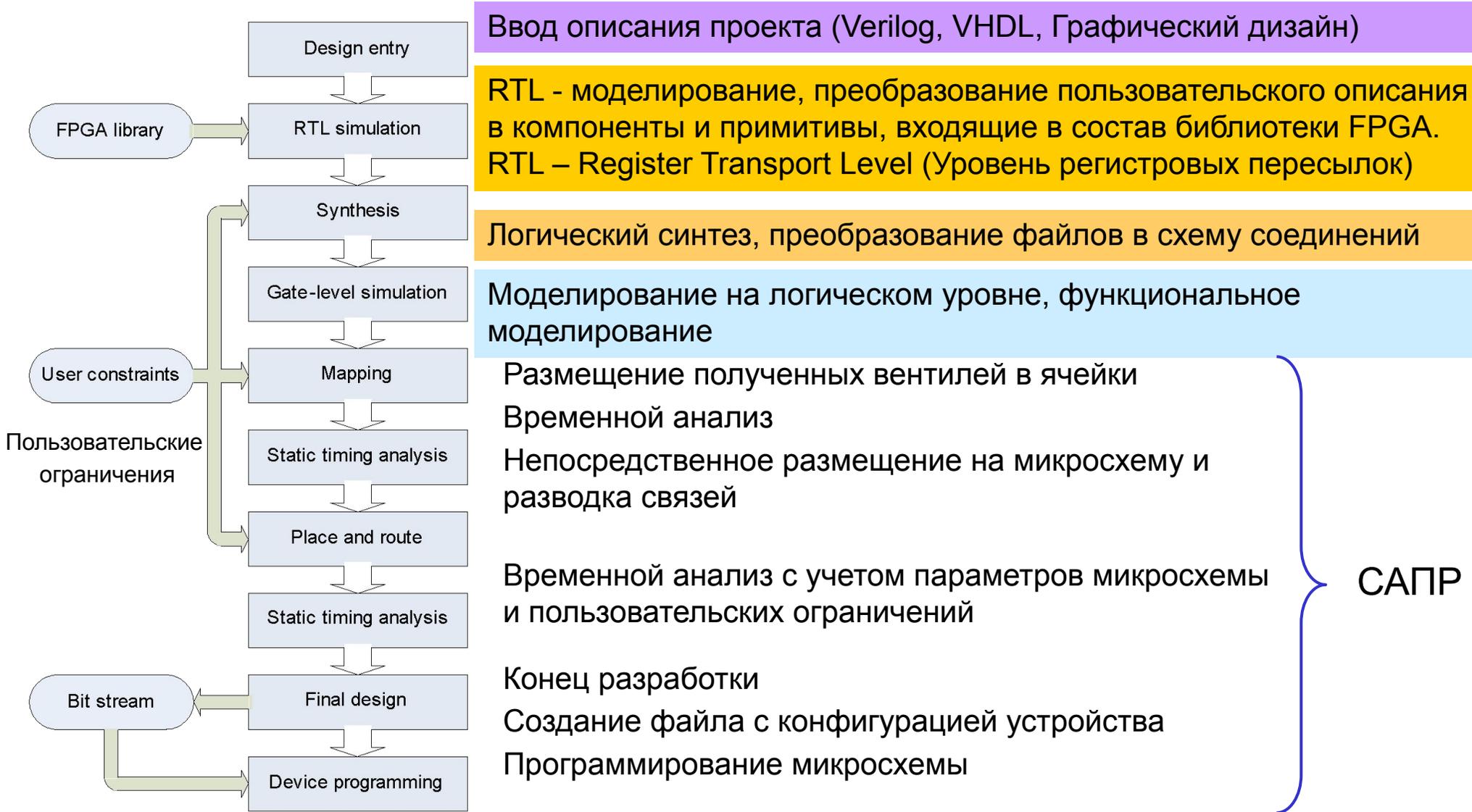


Введение в Verilog

Стандартный поток проектирования устройств на FPGA (Design flow)



Verilog – язык описания аппаратуры

- Описание **схем**, функциональных тестов и **алгоритмов функционирования** аппаратуры

- Любой уровень детализации описания:

Функциональный

- Описание устройств процессор-память

Логический

- Описание узлов на уровне вентилях (**gate level**) на уровне МОП-ключей (**switch level**)

Описание алгоритмов ЭВМ на уровне команд

Описание алгоритмов устройств на уровне межрегистровых передач и булевых функций

Объект проекта – компонент – модуль

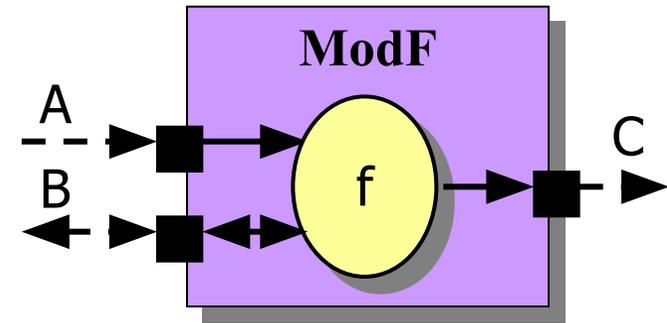
- Модуль – базовая единица проекта
- Модуль должен быть задекларирован (объявлен). □ *подготовка*
- Модуль может быть конкретизирован (создан экземпляр) □ *использование*
- Определение модуля должно вкладывается между ключевыми словами 'module' и 'endmodule'.

Интерфейс

имя модуля,
имена портов,
направленность портов (*in, out, inout*),
разрядность портов (1 бит, *n*-битный вектор)

Тело модуля

- Структурное описание
- Поведенческое описание



```
module module_name (список портов);  
    // in, out, inout объявление портов
```

Интерфейс

```
    // signal/wire/reg объявление сигналов  
    // data variable объявление переменных  
    // sub-module создание экземпляров вложенных  
        // модулей и подключение  
    // initial, always, function, task функциональные блоки,  
        // описывающие логику работы компонента
```

Тело модуля

```
endmodule
```

```
module ModF (A, B, C);  
    input    A;  
    inout   [7:0] B;  
    output  [7:0] C;  
    // описания  
    // описания 'f'  
endmodule
```

Структурное описание

- Функция устройства представляется как структура компонент низшего уровня сложности соединяемых друг с другом связями
- Компонент – объект внутренней структура которого не имеет значения на данном уровне рассмотрения
- Компонент может состоять из более простых компонент
- На самом низшем уровне иерархии проекта функция компонента может быть раскрыта (поведенческое описание)
- Использование библиотечных модулей – структурный синтез

Поведенческое описание

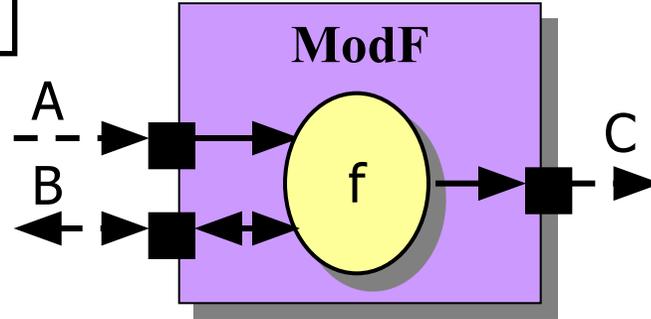
- Раскрывается функция компонента
- Моделируемая система – множество процессов взаимодействующих с помощью сигналов. Изменения сигналов – события. События порождают другие процессы.
- Создание собственных компонент для структурного синтеза. Раскрывается функция компонента.
- Разработчики САПР не раскрывают функцию компонента. Для разработчиков библиотечные модули доступны только в виде готовых компонентов для структурного синтеза

```

module ModF (A, B, C);
  input  A;
  inout [7:0] B;
  output [7:0] C;
  // описания
  // описания 'f'
endmodule

```

Создание экземпляра модуля

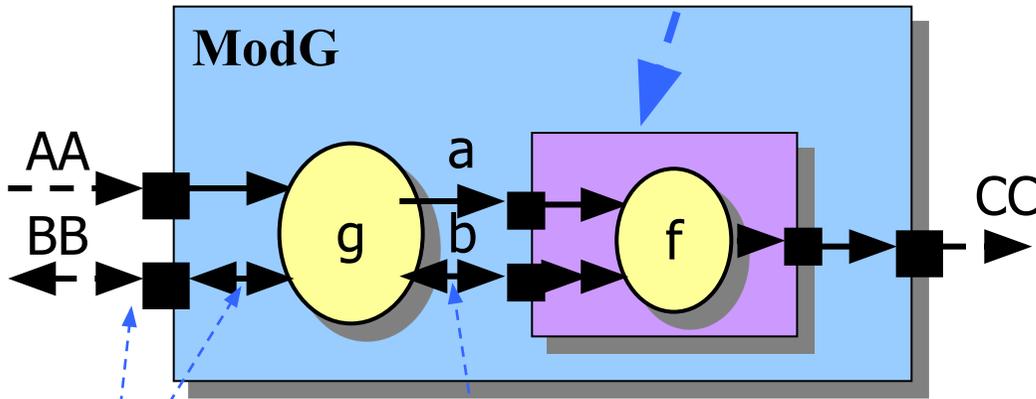


```

module ModG (AA, BB, CC);
  input  AA;
  inout [7:0] BB;
  output [7:0] CC;
  wire  a;
  wire [7:0] b;
  // описание 'g'
  // создание экземпляра 'f'
  ModF Umodf (.A(a), .B(b), .C(CC));
endmodule

```

создание экземпляра



Связь между компонентами - вводятся имена

Соединение портов

Имя экземпляра

Имя модуля

Имена портов и связей совпадают

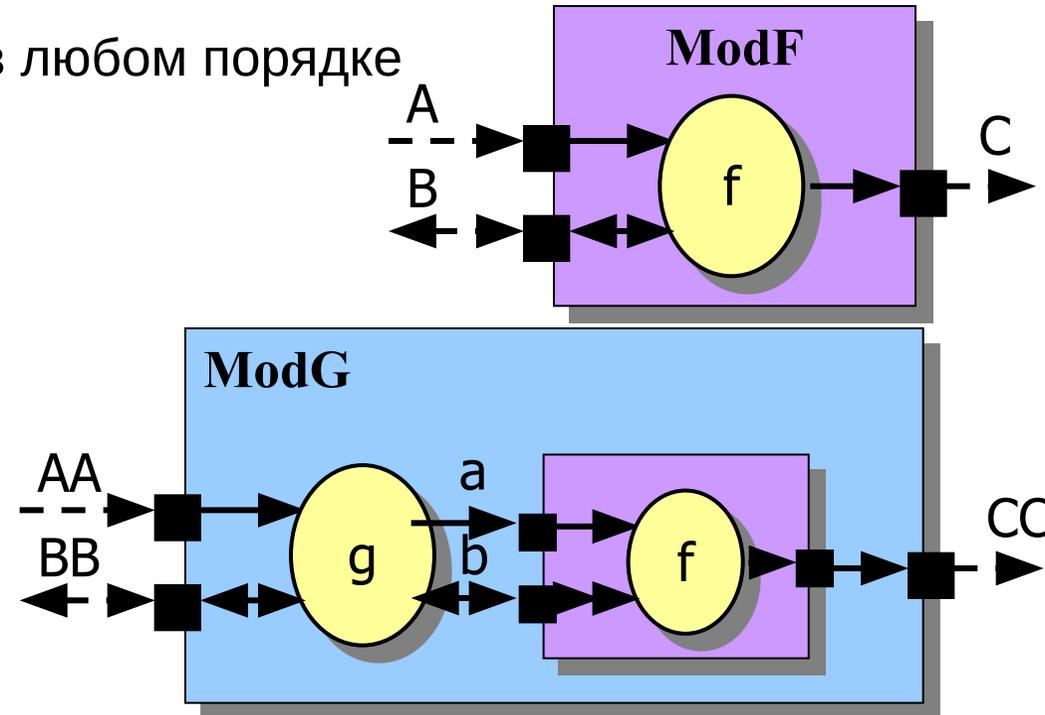
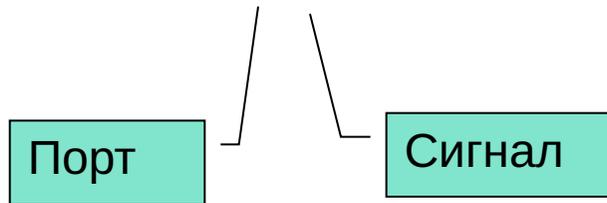
Соединение портов

- Позиционный принцип – сопоставление портов в объявлении модуля и в его экземпляре:

```
module ModF (A, B, C);  
ModF Umodf (a, b, cc);
```

- Ключевой принцип - явное указание, какой сигнал подключить к какому порту модуля

```
module ModF (A, B, C);  
ModF Umodf (.A(a), .B(b), .C(cc)); // в любом порядке
```



Типы данных Verilog

Поведенческое описание схемы - ПРОЦЕСС
`assign` - оператор непрерывного присваивания

- **Net** - соединение (связь, цепь, сигнал)

передача сигналов между объектами, сигнал постоянно удерживается

выходом объекта - источником (логический вентиль, комбинационная схема)

- **wire** – простой провод
- **wand, wor** – монтажное И/ИЛИ
- **tri, tri0, tri1, triand, trior, trireg**
- соединения с третьим состоянием
- **supply0** – постоянный 0 (GND)
- **supply1** – постоянная 1 (VCC/VDD)
- **Variable** - переменная

- **Reg** - регистр (сигнал не требующий драйвера)

- хранение данных при отключении их от источника
- не требует наличия тактирующего сигнала, изменяет значение в любой момент при поступлении новых данных
- средствами синтеза интерпретируется как триггер, группа триггеров, регистр

- **Integer** – специальный тип регистра

- 32-битовый знаковый тип данных

- **Real**– регистры действительного типа

- регистры действительного типа, действительные константы
- Не поддерживаются средствами синтеза, функциональный синтез, отладка проекта, test-bench код

```
// инициализация провода
wire a;
// назначить другой сигнал
wire b;
assign a = b;
```

```
// шина
wire [0:7]c;
wire [15:0]d;
// массив
// (8 32-х разрядных шин)
wire [8:0] g [31:0]
assign a = c[4];
```

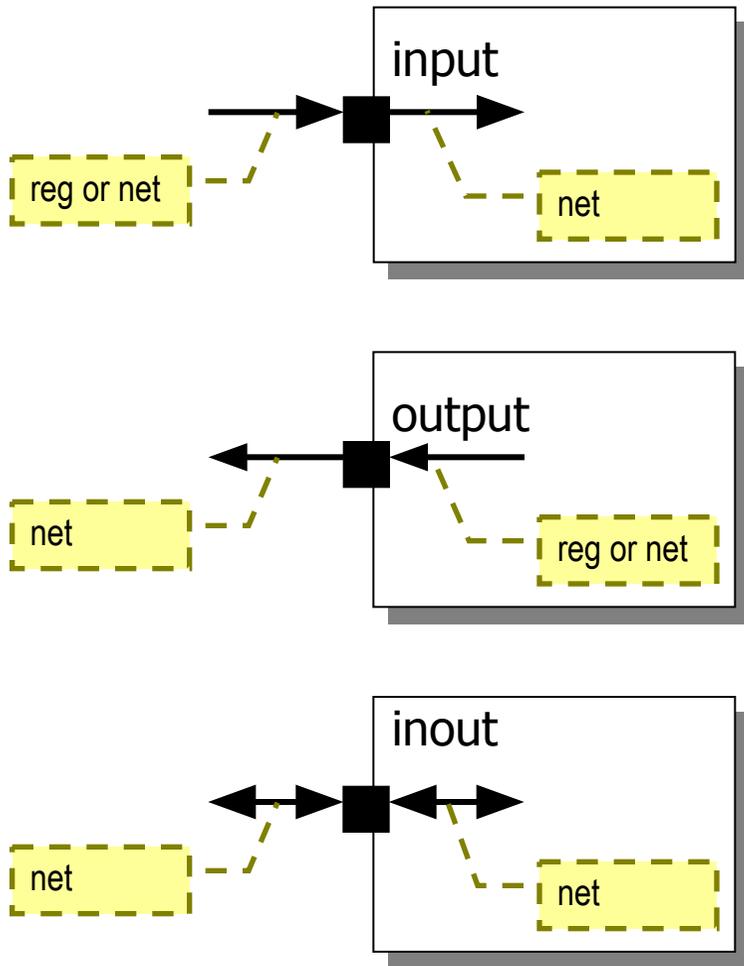
```
//4-х битное двоичное число
wire [3:0] t = 4'b0101;
//8-ми битное шестнадцатеричное
// число A5
wire [3:0] q = 8'hA5;
```

```
// шина
reg m[0:7];
reg n[15:0];
// присвоить проводу значение регистра
wire c[1:0] = s[31:30];
// память 32 слова X 8 бит
reg [7:0] s [31:0];
```

```
// Массив из 11 целых чисел:
integer Data[0:10]
```

```
real A; // регистр действительного типа
real Pi = 3.14; // действительная константа
```

Правила соединения портов



- Входной порт модуля должен иметь тип соединения (связь)
- Выходной порт модуля может иметь тип или соединение или регистр (переменная)
- Двухнаправленные порты должны иметь только тип соединения

```
input wire port_a,  
output wire [6:0] port_b,  
output reg [3:0] z  
input wire [0:4] w,  
inout wire y,
```

Четырехзначный алфавит Verylog

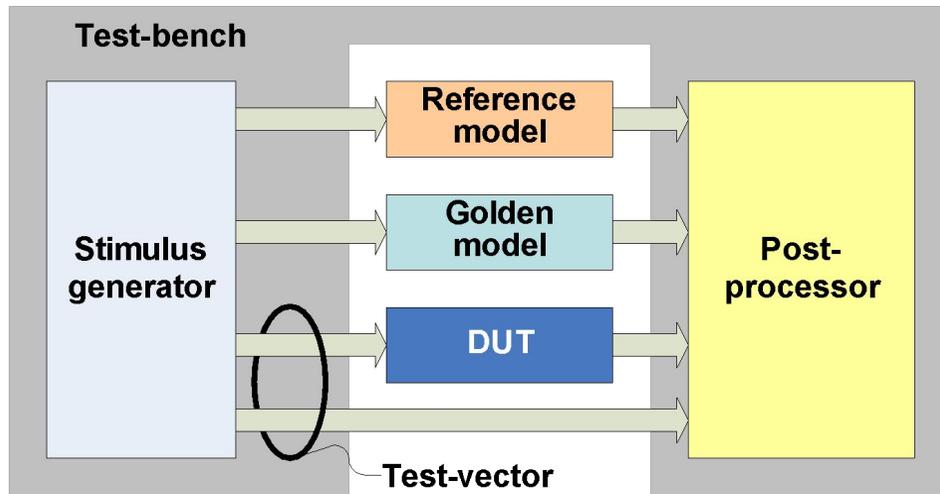
Значение	
0	<u>Низкий логический уровень</u> или <u>ложно</u>
1	<u>Высокий логический уровень</u> или <u>истинно</u>
X, x	Неопределенный логический уровень
Z, z	Высокоимпедансный логический уровень, третье состояние

Символ «?» так же используется для альтернативного представления «Z» состояния

Функциональное моделирование. Общий подход.

Test-bench уровень (Уровень тестирующих программ)

Замкнутая среда моделирования для моделируемой системы, содержит стимулы для моделирования, осуществляет автоматическую проверку и выдает сообщения об успехе/неуспехе. Используется для функционального моделирования (ModelSim)



Golden model – идеальная модель, на выходе выдает идеальные ожидаемые результаты

Reference model – базовая модель, взятая за основу, работает идентично, но может быть создана с помощью других функций, например математических или аппаратная реализация, созданная сторонними разработчиками.

- «Test-bench» это специальный уровень кода (уровень тестирующих программ), который создает пользовательские входные параметры (stimulus) для тестируемого проекта (DUT, design under test) и определяет, производит ли DUT ожидаемые (golden) выходные сигналы
- «Test-vector» установка значений для всех входных портов DUT (stimuli) и ожидаемых значений (образцов) выходных портов тестируемого модуля
- «Test-bench», который создает пользовательские входные сигналы, образцы выходных сигналов DUT и сравнивает выходные сигналы с ожидаемыми (golden) результатами называется однородно-проверяющим.

Разработка структурной схемы сумматора

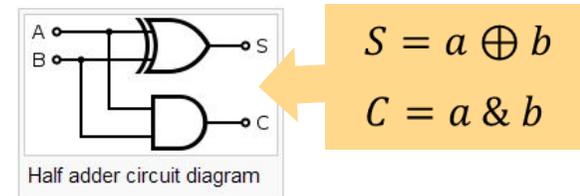
- Сумматор цифровая схема которая выполняет сложение чисел

- Одноразрядный полусумматор складывает два однобитных двоичных числа (A и B). На выходе формируется значение суммы (S) и переноса (C).
- Полный одноразрядный сумматор складывает три однобитных значения (C, A and B). На выходе формируется значение суммы (S) и переноса (C).

Cin	a	b	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

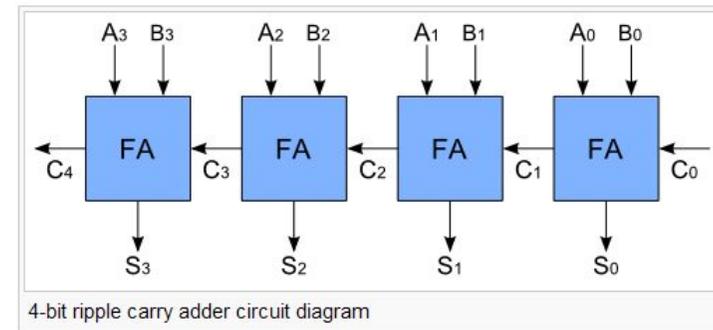
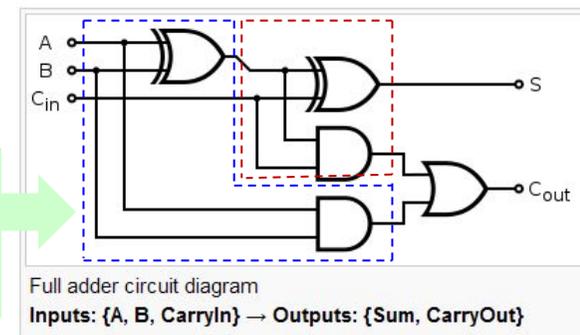
$$S = (a \oplus b) \oplus c_{in}$$

$$C_{out} = ab \vee c_{in}(a \oplus b)$$



$$S = a \oplus b$$

$$C = a \& b$$



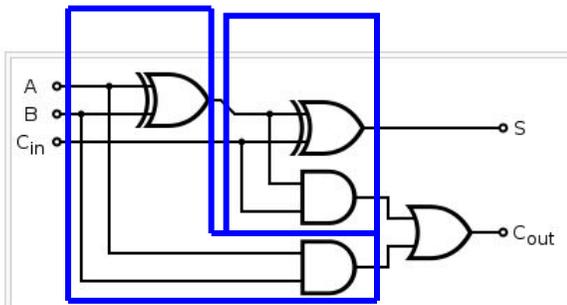
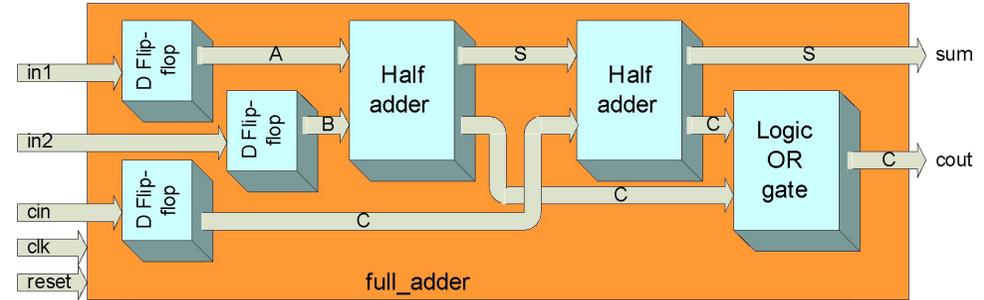
- Многоразрядный сумматор

- Суммирование с распространением переносов
- Перенос из младшего полусумматора учитывается старшим полусумматором

Проект “adder” - сумматор

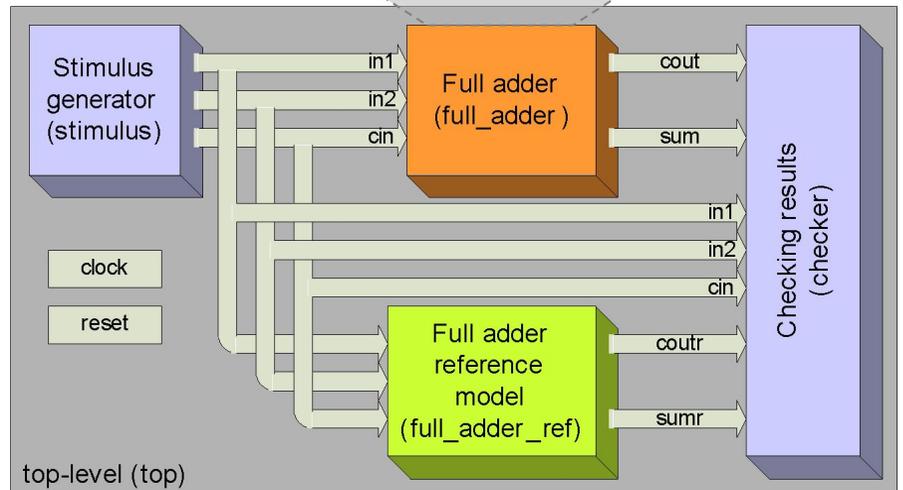
- Содержимое папки ‘design’

- top.v
- full_adder.v
- half_adder_gate.v
- half_adder_rtl.v
- stimulus.v
- full_adder_ref.v
- checker.v

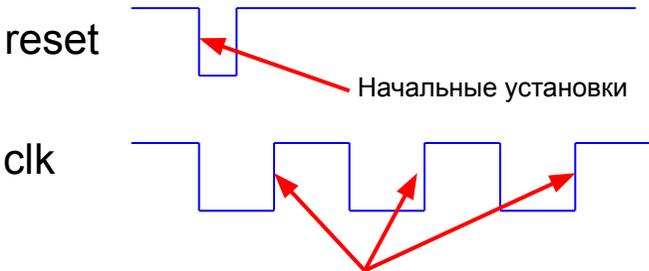
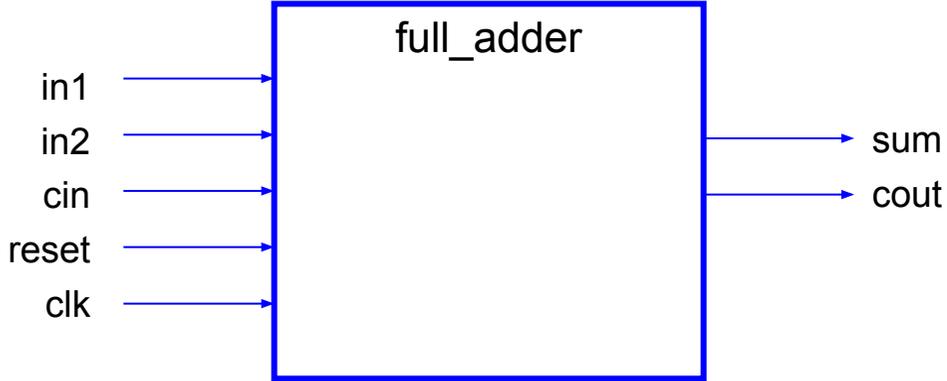


Full adder circuit diagram

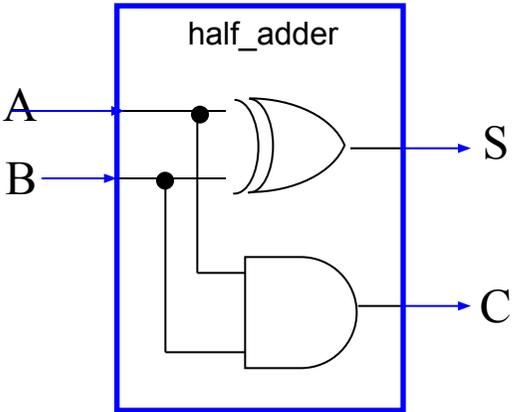
Inputs: {A, B, CarryIn} → Outputs: {Sum, CarryOut}



Модуль full_adder

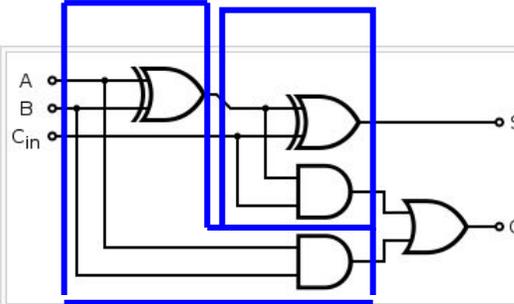


Запись входных значений во входные регистры, формирование результата

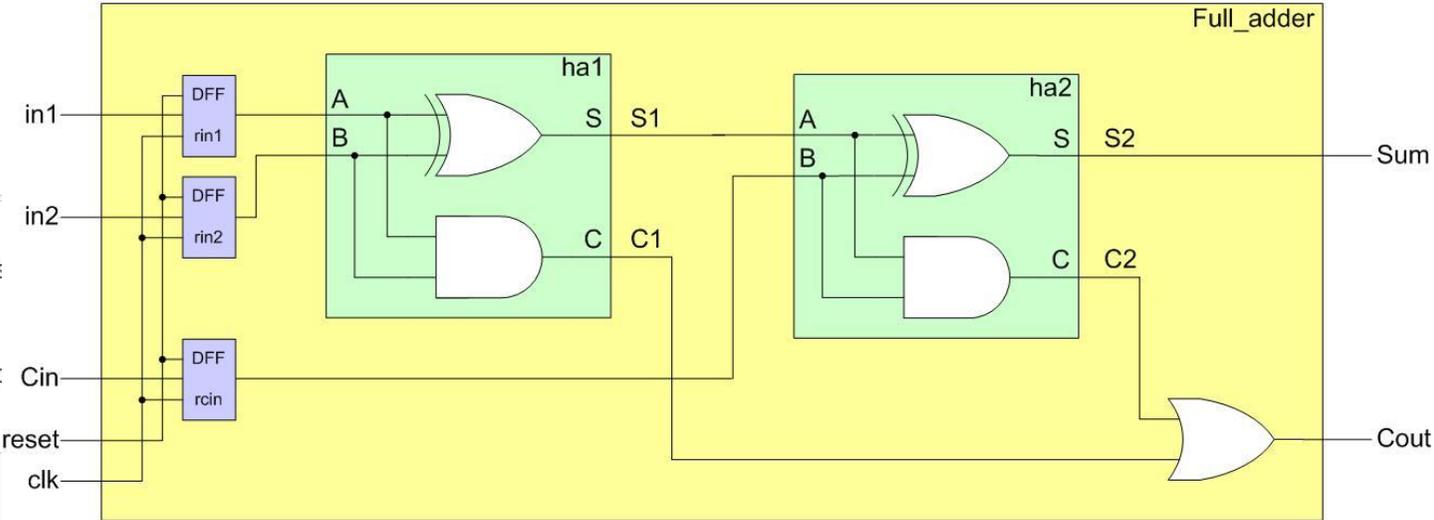


```

module half_adder (S, C, A, B);
    output S, C;
    input  A, B;
    // описание модуля
endmodule
    
```



Full adder circuit diagram
 Inputs: {A, B, CarryIn} → Outputs: {Sum, CarryOut}



Операторы поведенческого описания схем

Поведенческие операторы (процессы)

initial

begin

//Последовательность поведенческих операторов;

end;

- запускается один раз в начале процесса моделирования (модельное время 0);
- несколько блоков запускаются параллельно;
- игнорируется средствами синтеза логических цепей;
- Назначение: формирование начальных значений и тестовых последовательностей, мониторинг значений сигналов, вывод на внешние носители информации

assign - оператор непрерывного присваивания

```
always [@(СПИСОК ЧУВСТВИТЕЛЬНОСТИ) ]
```

```
begin
```

```
// Последовательность поведенческих операторов;
```

```
end
```

Оператор ожидания события

```
@ (A or D) / событие изменения сигналов A или B
```

```
@ posedge clk / событие фронта clk
```

```
@ negedge clk / событие среза clk
```

- здесь следует размещаются поведенческие операторы, реализующие основной алгоритм функционирования электронного устройства.
- работа процесса **always** – бесконечное формирование выходного сигнала электронным устройством по заданному алгоритму.

Операторы присваивания

```
module test1 (input wire date, input wire clk, output reg reg_a, output reg reg_b);
```

```
always @(posedge clk)
```

Неблокирующее присваивание (параллельно)

```
begin
```

```
reg_a<=date;
```

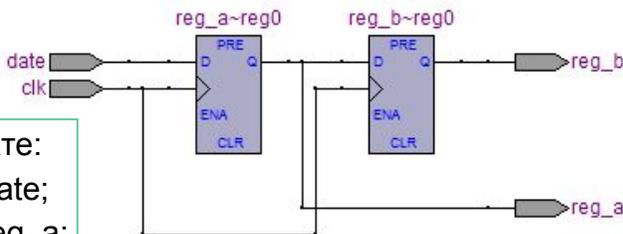
```
reg_b<=reg_a;
```

```
end
```

```
endmodule
```

В результате:

```
// reg_a=date;  
// reg_b=reg_a;
```



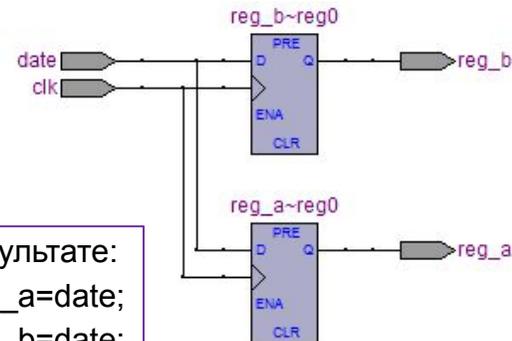
Блокирующее присваивание

```
reg_a=date;
```

```
reg_b=reg_a;
```

В результате:

```
// reg_a=date;  
// reg_b=date;
```

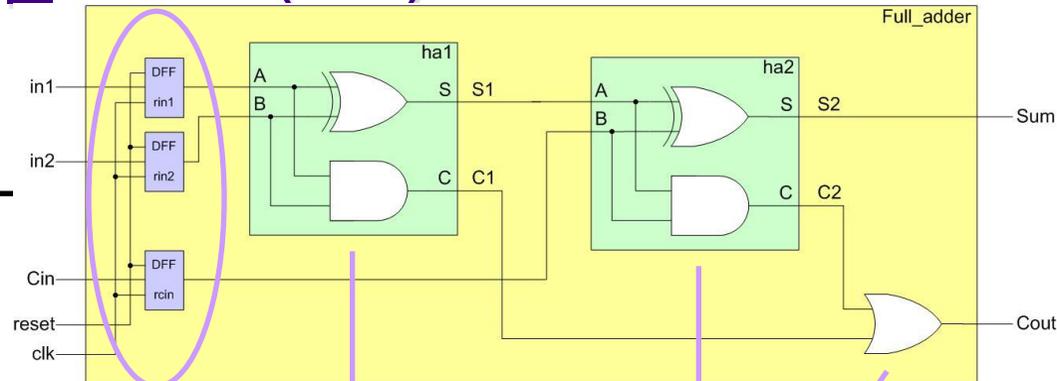


Модуль full_adder (1/3)

```

timescale 1ns/1ps
module full_adder(sum,cout,in1,in2,cin,clk,resetb);
    output sum, cout; /выходные сигналы
    input in1, in2, cin; /входные сигналы
    input clk, resetb; /входные сигналы
    /объявление типов связей, сигналов и соединений
    wire sum, cout; /соединение (проводник)
    reg rin1, rin2, rcin; /переменная (регистр)
    wire s1, c1, s2, c2; /соединение (проводник)
    /процедурный блок - процесс
    always @ (posedge clk or negedge resetb) begin
        if (resetb==1'b0) begin
            rin1 <= 1'b0; rin2 <= 1'b0; rcin <= 1'b0; end
        else begin
            rin1 <= in1; rin2 <= in2; rcin <= cin;
        end
    end
    /создание экземпляров модулей
    half_adder_gate ha1 (.S(s1), .C(c1), .A(rin1), .B(rin2));
    half_adder_rtl ha2 (.S(s2), .C(c2), .A(s1), .B(rcin));
    /процедурные присваивания
    assign sum = s2;
    assign cout = c1|c2;

```



timescale 1ns/1ps

Длительность
одного шага
моделирования
(единица)

Дискретность
времени
моделирования
(точность)

Full adder (2/3)

```
module full_adder(sum,cout,in1,in2,cin,clk,resetb);  
  output sum, cout;  
  input in1, in2, cin;  
  input clk, resetb;  
  
  wire sum, cout;  
  reg rin1, rin2, rcin;  
  wire s1, c1; wire s2, c2;  
  
  always @ (posedge clk or negedge resetb) begin  
    if (resetb==1'b0) begin  
      rin1 <= 1'b0; rin2 <= 1'b0; rcin <= 1'b0;  
    end else begin  
      rin1 <= in1; rin2 <= in2; rcin <= cin;  
    end  
  end  
  
  half_adder_gate ha1 (.S(s1), .C(c1), .A(rin1), .B(rin2));  
  half_adder_rtl ha2 (.S(s2), .C(c2), .A(s1), .B(rcin));  
  
  assign sum = s2;  
  assign cout = c1|c2;  
endmodule
```

Имя модуля

Объявление портов

Описание портов

Описание сигналов и соединений

Описание логики работы модуля, процесс

Создание экземпляров модулей полусумматора

Описание логики работы модуля, процесс

Full adder (3/3)

```
module full_adder(sum,cout,in1,in2,cin,clk,resetb);
  output sum, cout;
  input in1, in2, cin;
  input clk, resetb;

  wire sum, cout;
  reg rin1, rin2, rcin;
  wire s1, c1; wire s2, c2;

  always @ (posedge clk or negedge resetb) begin
    if (resetb==1'b0) begin
      rin1 <= 1'b0; rin2 <= 1'b0; rcin <= 1'b0;
    end else begin
      rin1 <= in1; rin2 <= in2; rcin <= cin;
    end
  end

  half_adder_gate ha1 (.S(s1), .C(c1), .A(rin1), .B(rin2));
  half_adder_rtl ha2 (.S(s2), .C(c2), .A(s1), .B(rcin));

  assign sum = s2;
  assign cout = c1|c2;
endmodule
```

Verilog типы данных: связь and переменная

□ net : wire

□ variable: reg, integer, real ...

Этот процесс запускается, когда наступает событие размещенное в списке чувствительности.

фронт 'clk'

спец 'resetb'

Блок процедурного присваивания, переменные внутри блока инициализируются и присваиваются многократно, если случается событие

=: оператор блокирующего присваивания

<=: неблокирующее присваивание

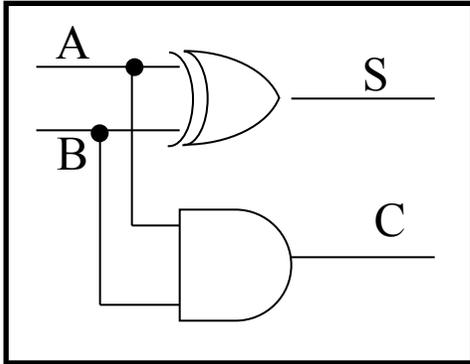
Экземпляры моделей, которые описывают соединение портов в виде позиционного списка

Блоки непрерывного присваивания значений сигналам, которые срабатывают, если изменяется хотя бы один сигнал в правой части

выражения

|: оператор побитового ИЛИ (OR)

Структурная модель полусумматора (описание на уровне базовых элементов (gate level))



использование
библиотечных
модулей and и xor

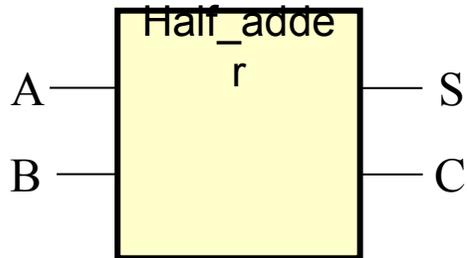
```
module half_adder_gate (S, C, A, B);  
    output S, C;  
    input  A, B;  
    and UAND (C, A, B);  
    xor UXOR (S, A, B);  
endmodule
```

Структурное описание – структура объекта, как композиция компонентов, соединенных между собой и обменивающихся сигналами.

Структурная модель - создание экземпляров примитивов и модулей (использование библиотечных модулей, или создание собственных)

Поведенческое описание объектов

Процессная форма описания поведения (Behavior model)



Объект представлен в виде “черного ящика”, описывают функцию объекта - зависимость выходных сигналов от входных на уровне **одного процесса**.

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

```
module half_adder_beh1 (S, C, A, B);
    output S, C;
    input  A, B;
    reg    S, C;
    always @ (A or B)
    begin
        if ((A==0) or (B==1)) and ((A==0) or (B==1))
            begin S<=1'b1; C<=1'b0; end
        else
            begin
                S<=1'b0;
                if (A==0) and (B==0)
                    C<=1'b0; else C<=1'b1;
            end
        end
    end
endmodule
```

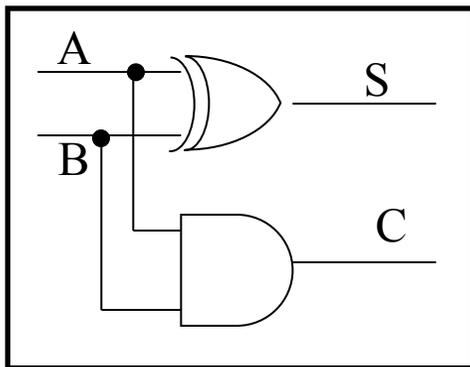
```
module half_adder_beh2 (S, C, A, B);
    output S, C;
    input  A, B;
    reg    S, C;
    always
    begin @ (A or B);
        S<=A^B;    / S = A or B
        C<=A&B;    / S = A xor B
    end
endmodule
```

Поведенческое описание объектов

Полусумматор. Потокное описание архитектуры (Data-flow model (RTL-модель))

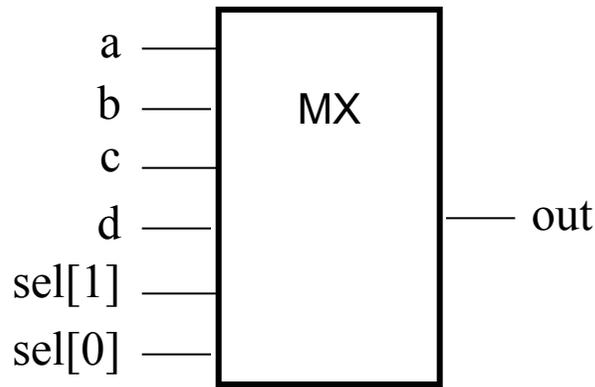
(RTL, Register Transfer Level, Уровень регистровых передач)

- ❖ Объект представлен архитектурным описанием, где минимальная видимая единица примитив RTL уровня - RTL модель
- ❖ Data-flow модель – модель потоков данных
- ❖ Описывает поведение архитектуры объекта, потоки данных функционирующие на уровне архитектуры объекта и их преобразование.
- ❖ Поведение архитектуры описывается с помощью **операторов непрерывных назначений** (присваиваний), которые представляют собой параллельные процессы.



```
module half_adder_rtl (S, C, A, B);  
    output S, C;  
    input  A, B;  
    wire  S, C;  
    assign C = A & B;  
    assign S = A ^ B;  
endmodule
```

Модели мультиплексора (Behavior model)



sel[1]	sel[0]	out
0	0	a
0	1	b
1	0	c
1	1	d

```

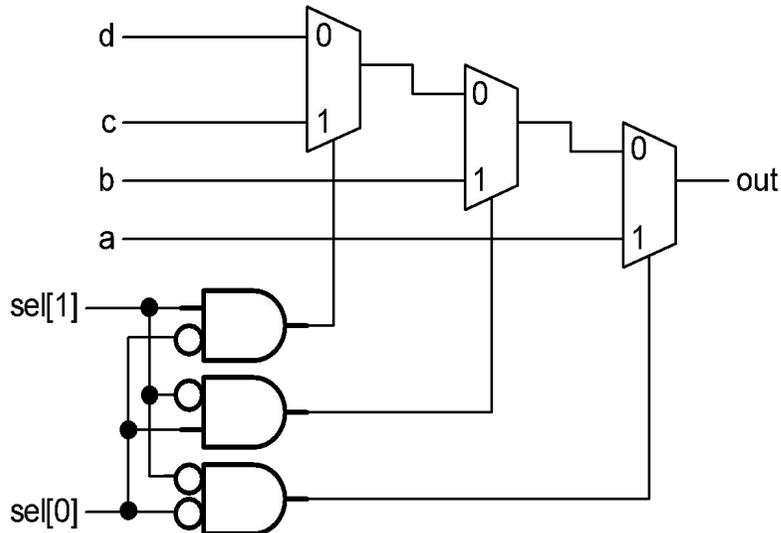
module mx_beh (sel, a, b, c, d, out);
    output out;
    input  sel, a, b, c, d;
    wire  a, b, c, d;
    wire  [1:0]sel;
    / description
endmodule
    
```

```

always @ (sel or a or b or c or d)
    if (sel == 2'b00) out = a;
    else if (sel == 2'b01) out = b;
    else if (sel == 2'b10) out = c;
    else out = d;
    
```

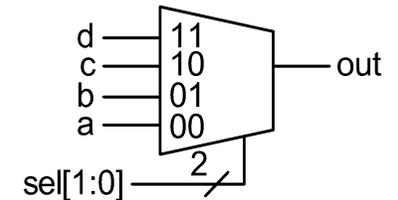
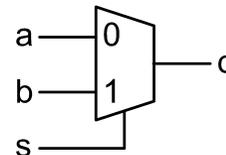
```

always @ (sel or a or b or c or d)
    case (sel)
        2'b00: out = a;
        2'b01: out = b;
        2'b10: out = c;
        default: out = d;
    endcase
    
```

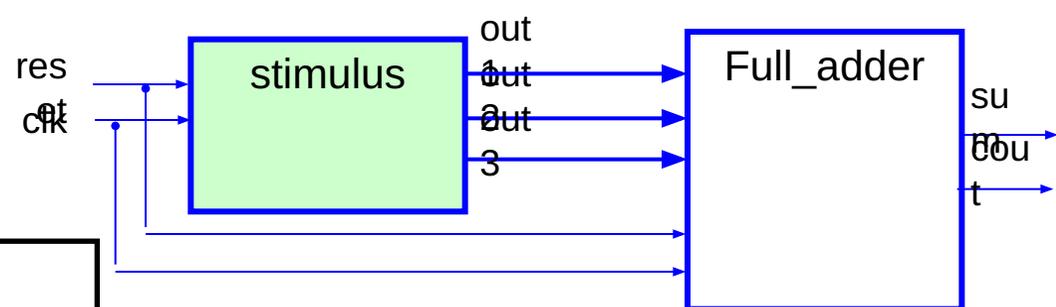


```

assign c = (s) ? b : a;
    
```



Test-bench: stimulus



```

module stimulus(out1,out2,out3,clk,resetb);
    output out1,out2,out3;
    input clk,resetb;

    reg out1,out2,out3;

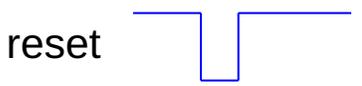
    initial begin
        out1 <=0; out2 <=0; out3 <=0;
        / ожидание сброса триггеров в 0
        wait (resetb==1'b0);
        wait (resetb==1'b1);
        / по фронту clk генерация на выходе трех битов
        @ (posedge clk); out1=1; out2=0; out3=0;
        @ (posedge clk); out1=0; out2=1; out3=0;
        @ (posedge clk); out1=1; out2=1; out3=0;
        @ (posedge clk); out1=0; out2=0; out3=1;
        @ (posedge clk); out1=1; out2=0; out3=1;
        @ (posedge clk); out1=0; out2=1; out3=1;
        @ (posedge clk); out1=1; out2=1; out3=1;
        @ (posedge clk);
        / повторить 3 раза
        repeat (3) @ (posedge clk);

        $finish; /
    end
endmodule
    
```

Initial однократно выполняемая конструкция, выполняется во время старта симуляции.

<=: неблокируемый оператор присваивания

'wait' – оператор ожидания условия (выполняется, когда условие станет true)



'@' оператор ожидания события

=: оператор блокирующего присваивания, присваивания выполняются последовательно

'repeat': Выполняет установку фиксированного количества тактов

\$finish - Системная задача, конец моделирования, выполняет остановку симулятора и передает управление назад операционной системе компьютера

end
endmodule

Test-bench: full_adder_ref

```
module full_adder_ref(sum,cout,in1,in2,cin,clk,resetb);
  output sum, cout;
  input  in1, in2, cin;
  input  clk, resetb;

  wire  sum, cout;
  reg   rin1, rin2, rcin;

  always @ (posedge clk or negedge resetb) begin
    if (resetb==1'b0) begin
      rin1 <= 1'b0;
      rin2 <= 1'b0;
      rcin <= 1'b0;
    end else begin
      rin1 <= in1;
      rin2 <= in2;
      rcin <= cin;
    end
  end
end

assign {cout, sum} = rin1+rin2+rcin;

endmodule
```

Оператор конкатенации ({, }) объединение двух или более значений в последовательность.

Test-bench: checker

```
module checker(in1,in2,cin,sum,cout,sumr,coutr,clk,resetb);
  input in1,in2,cin,sum,cout,sumr,coutr,clk,resetb;
  always @ (clk) begin
    if ({cout,sum}=={coutr,sumr})
      $display($time,,"correct");
    else $display($time,,"error result=%b expect=%b", {cout, sum}, {coutr,sumr});
  end
endmodule
```

Display - системная задача, вывод информации с новой строки на экран.

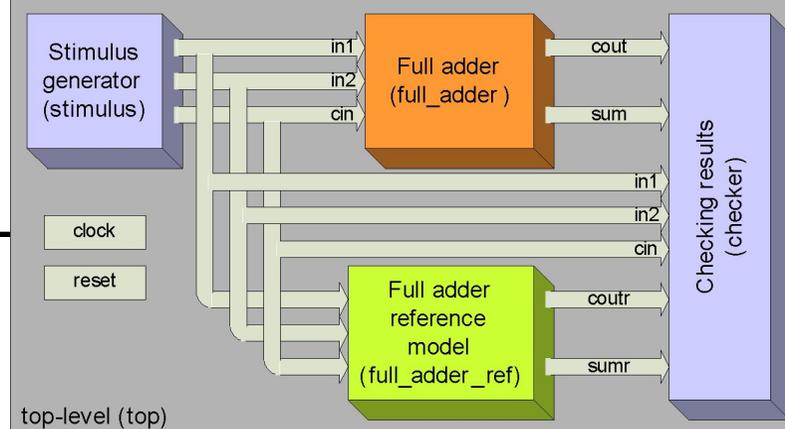
```
$display ( «текст с символами форматирования», list_of_arguments );
```

Time - системная функция возвращает текущее модельное время, целое 64-битное значение времени, масштабируемое соответственно единице временного масштаба модуля (timescale), установленной в нем

```
$time ;
```

Top-level модуль содержит весь проект, и не имеет входных портов.

Test-bench: top



```
module top;  
  wire in1, in2, cin;  
  wire sum, cout, sumr, coutr;  
  reg clk, resetb;  
  full_adder Ufa (.sum(sum), .cout(cout), .in1(in1), .in2(in2), .cin(cin), .clk(clk), .resetb(resetb));  
  full_adder_ref Urf (.sum(sumr), .cout(coutr), .in1(in1), .in2(in2), .cin(cin), .clk(clk), .resetb(resetb));  
  stimulus Ust (.out1(in1), .out2(in2), .out3(cin), .clk(clk), .resetb(resetb));  
  checker Uck (.in1(in1), .in2(in2), .cin(cin), .sum(sum), .cout(cout), .clk(clk), .resetb(resetb));  
/ генератор синхроимпульсов с периодом 5ns  
initial begin  
  clk <= 0;  
  forever #5 clk = ~clk;  
end  
initial begin  
  resetb <= 1'b0;  
  #200 resetb <= 1'b1;  
end  
initial begin  
  $dumpfile("wave.vcd");  
  $dumpvars(1);  
  $dumpvars(1, Ufa);  
end  
endmodule
```

Forever - непрерывное назначение

Просмотр сигналов в графическом виде, в виде *waveform*

VCD файлы – Value Change Dump File, текстовые файлы, описывают сигналы и моменты их изменения в проекте во время симуляции. Специальные программы - отображают сигналы в графическом виде (GTKWave)

Dumpfile задача для установки имени VCD file.

```
$dumpfile( filename );
```

Dumpvars задача для установки переменных для записи в VCD file

```
$dumpvars ( level, module_name );
```

Уровень иерархии проекта относительно некоторого модуля. Без параметров - все сигналы будут выведены в файл

Результаты моделирования

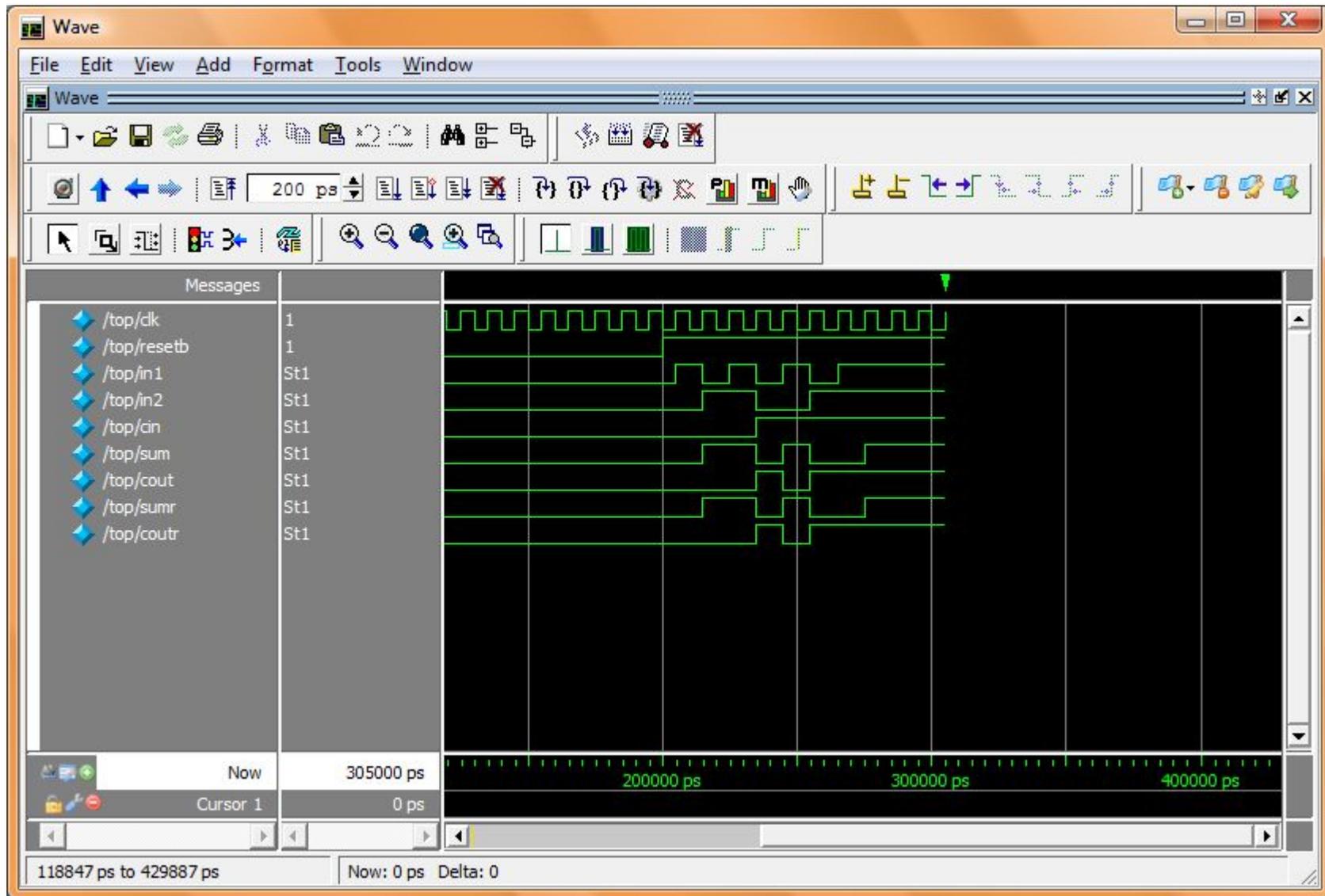
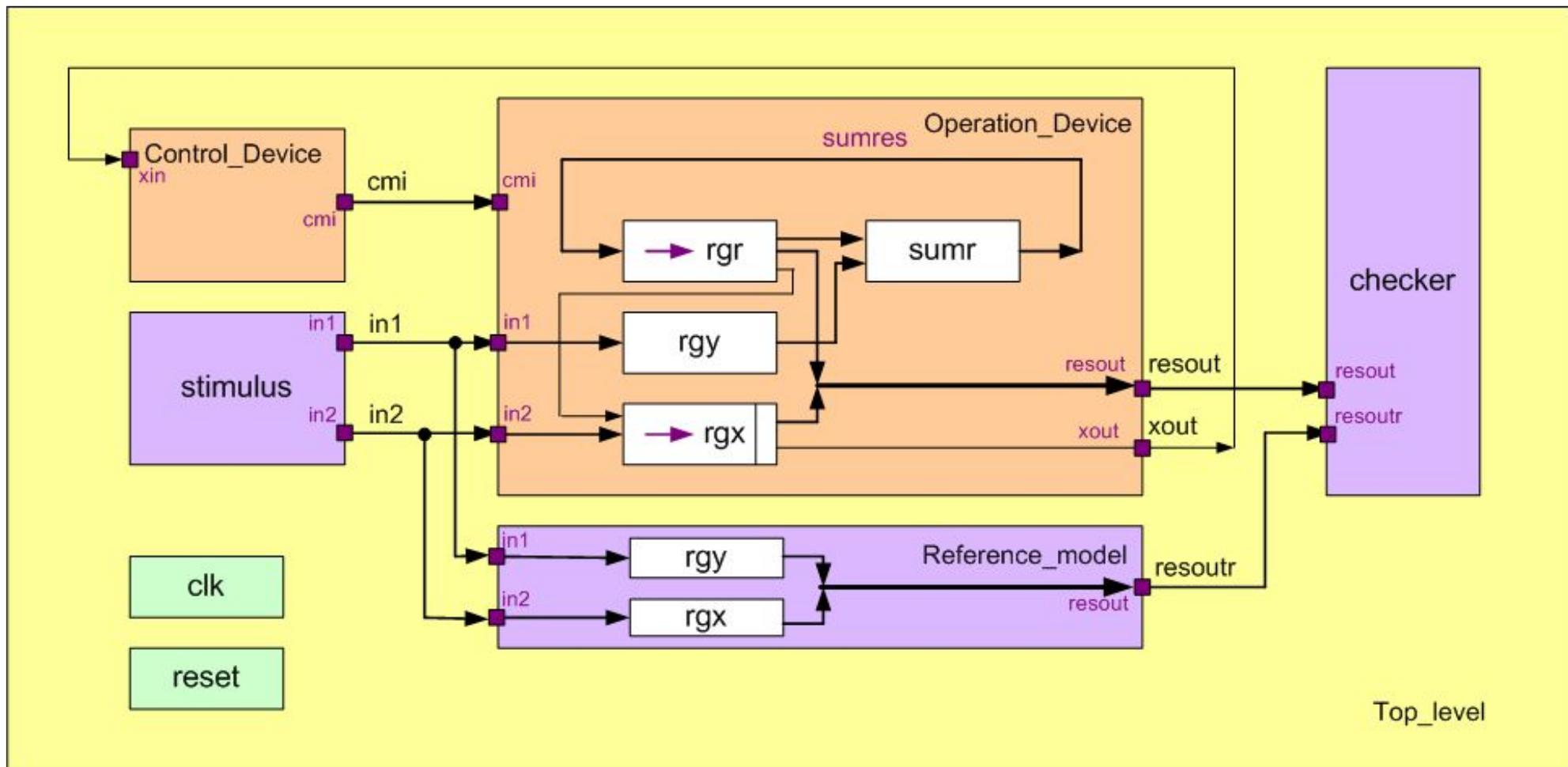


Схема для лабораторной работы №3



The image shows a screenshot of a VSIM Transcript window. The window title is "Transcript". The menu bar includes "File", "Edit", "View", and "Window". The toolbar contains icons for file operations (open, save, print, copy, paste, delete) and simulation controls (run, stop, refresh, help). The main text area displays the following content:

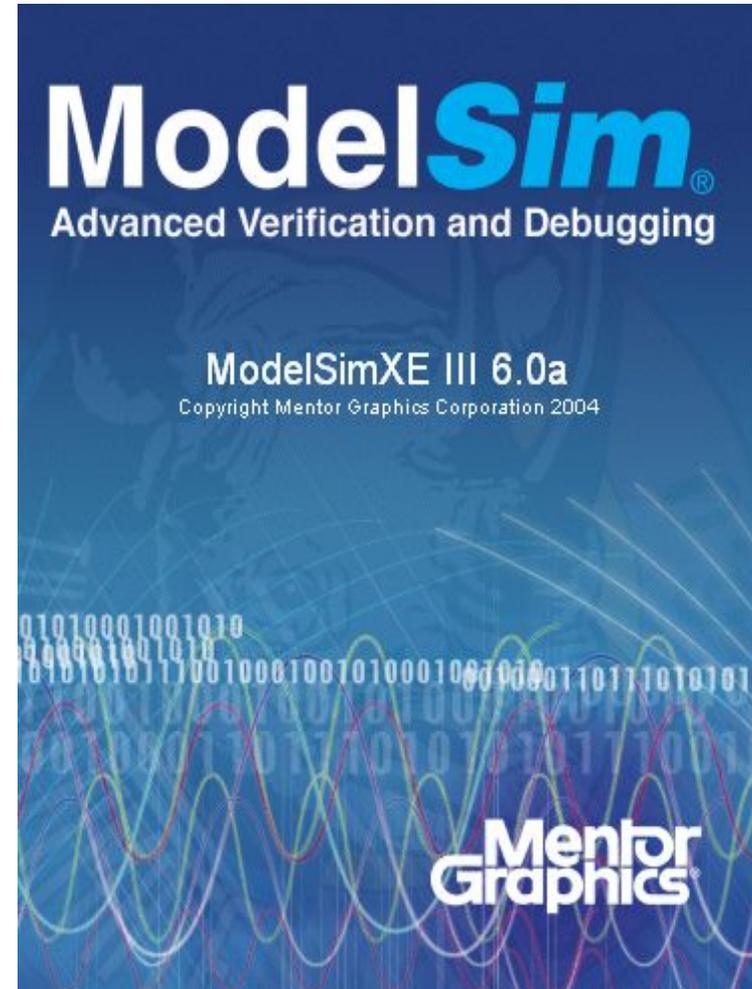
```
# Loading work.full_adder_ref(fast)
# Loading work.checker(fast)
VSIM 2> run -all
#           0 error result=xx expect=xx
#           5 correct
#          10 correct
#          15 correct
#          20 correct
#          25 correct
#          30 correct
#          35 correct
#          40 correct
#          45 correct
#          50 correct
#          55 correct
#          60 correct
#          65 correct
#          70 correct
#          75 correct
#          80 correct
#          85 correct
#          90 correct
#          95 correct
#         100 correct
#         105 correct
#         110 correct
#         115 correct
#         120 correct
#         125 correct
#         130 correct
#         135 correct
#         140 correct
#         145 correct
#         150 correct
#         155 correct
#         160 correct
```

- <http://www.allhdl.ru/verilog.php#primA>
- <http://naliwator.narod.ru/index.html>

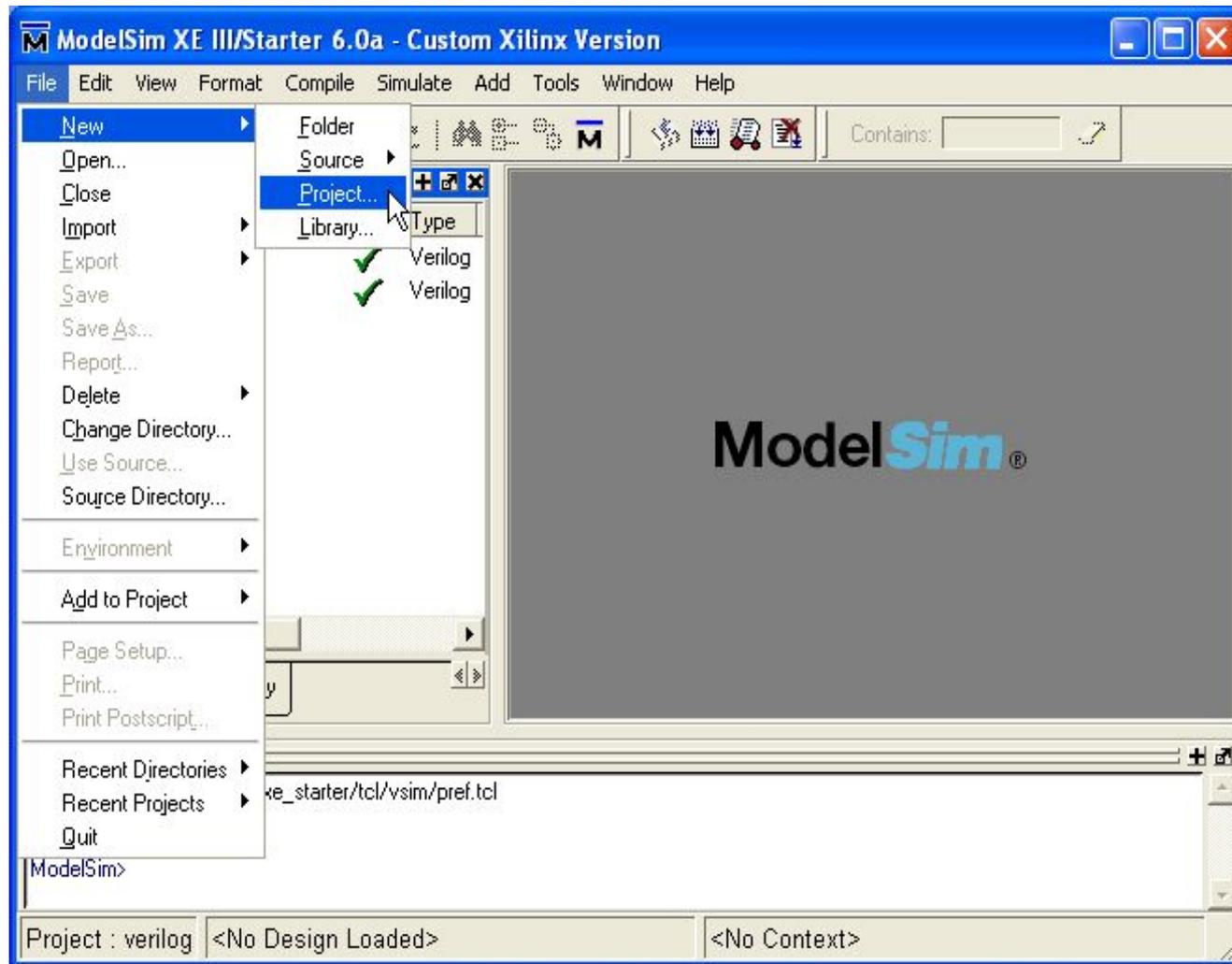
Моделирование с использованием ModelSim GUI

- Invoking ModelSim form start menu
- Create new project
- Add design files
- Compile
- Wave setting
- Simulation
- Invoking ModeSim project

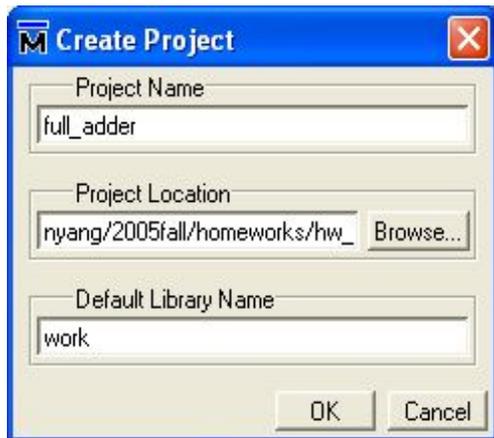
Invoking ModelSim from start menu



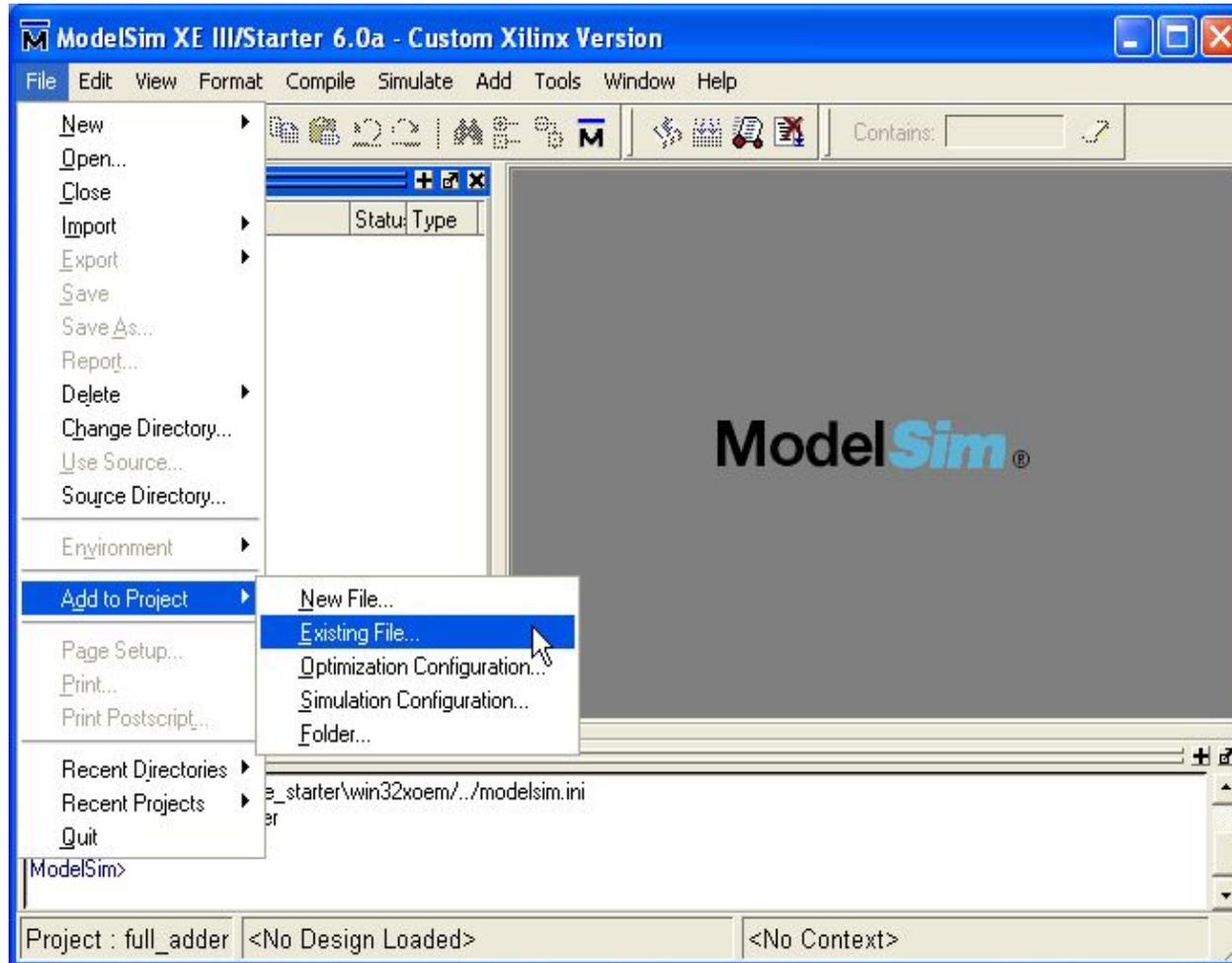
File->New->Project



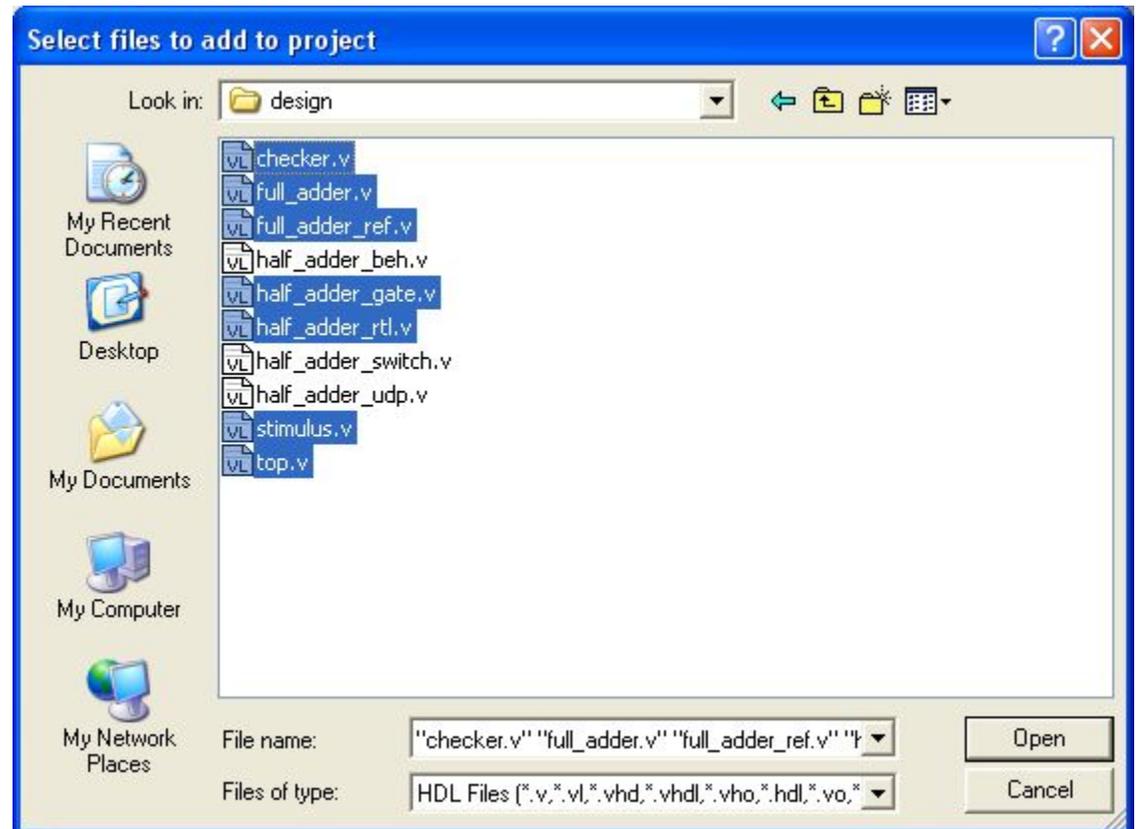
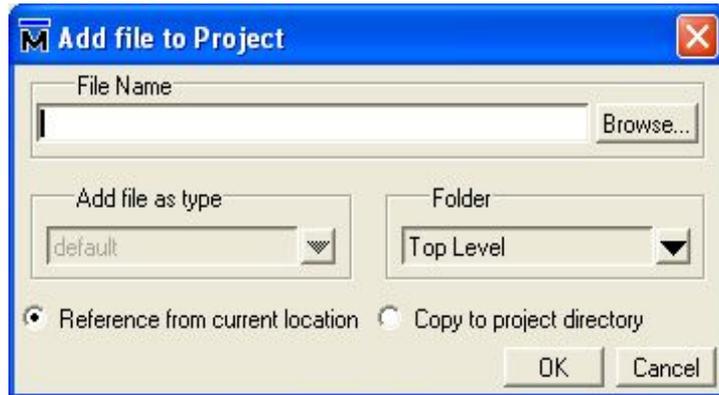
Specify project name and location



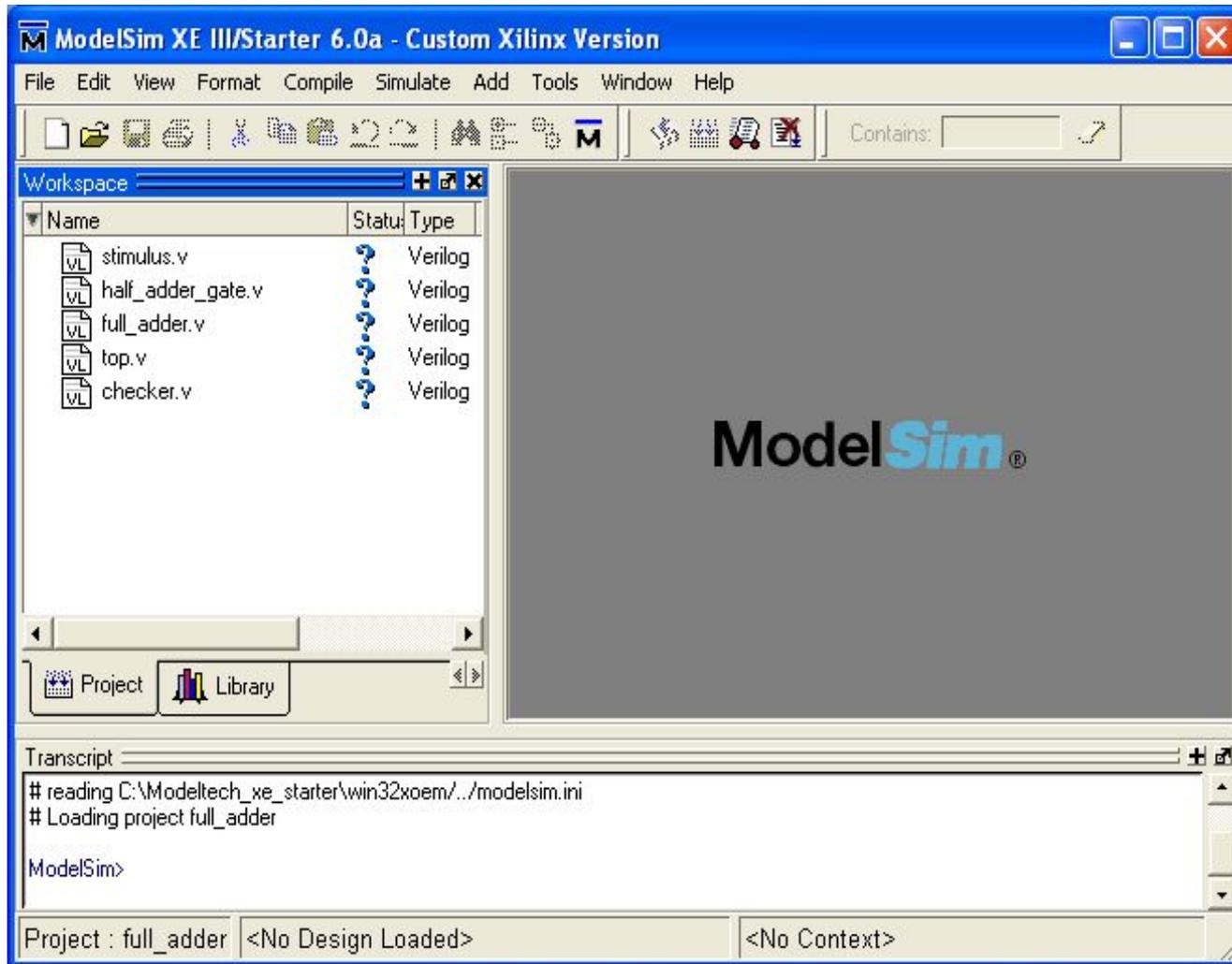
File->Add to Project->Existing File



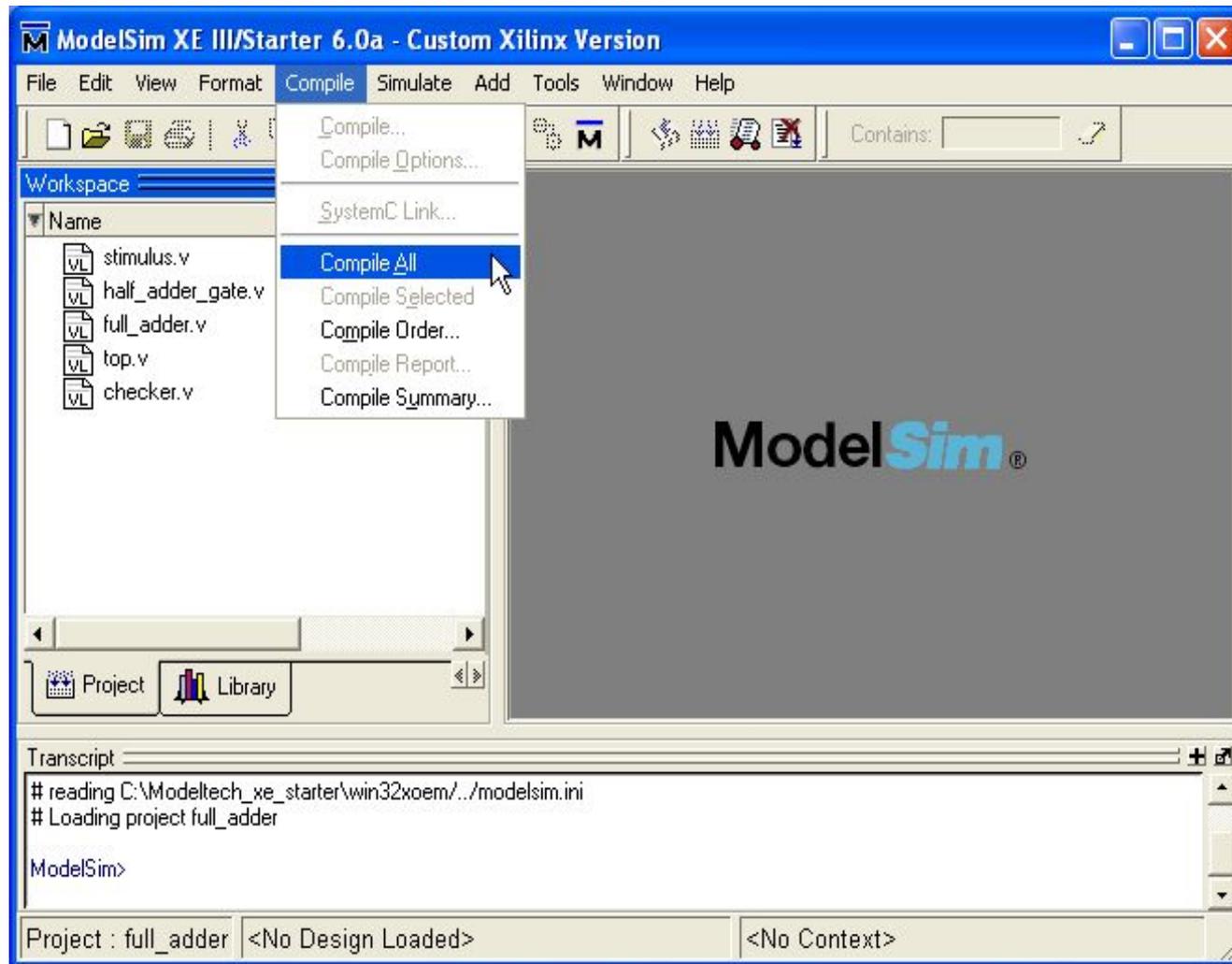
Add files



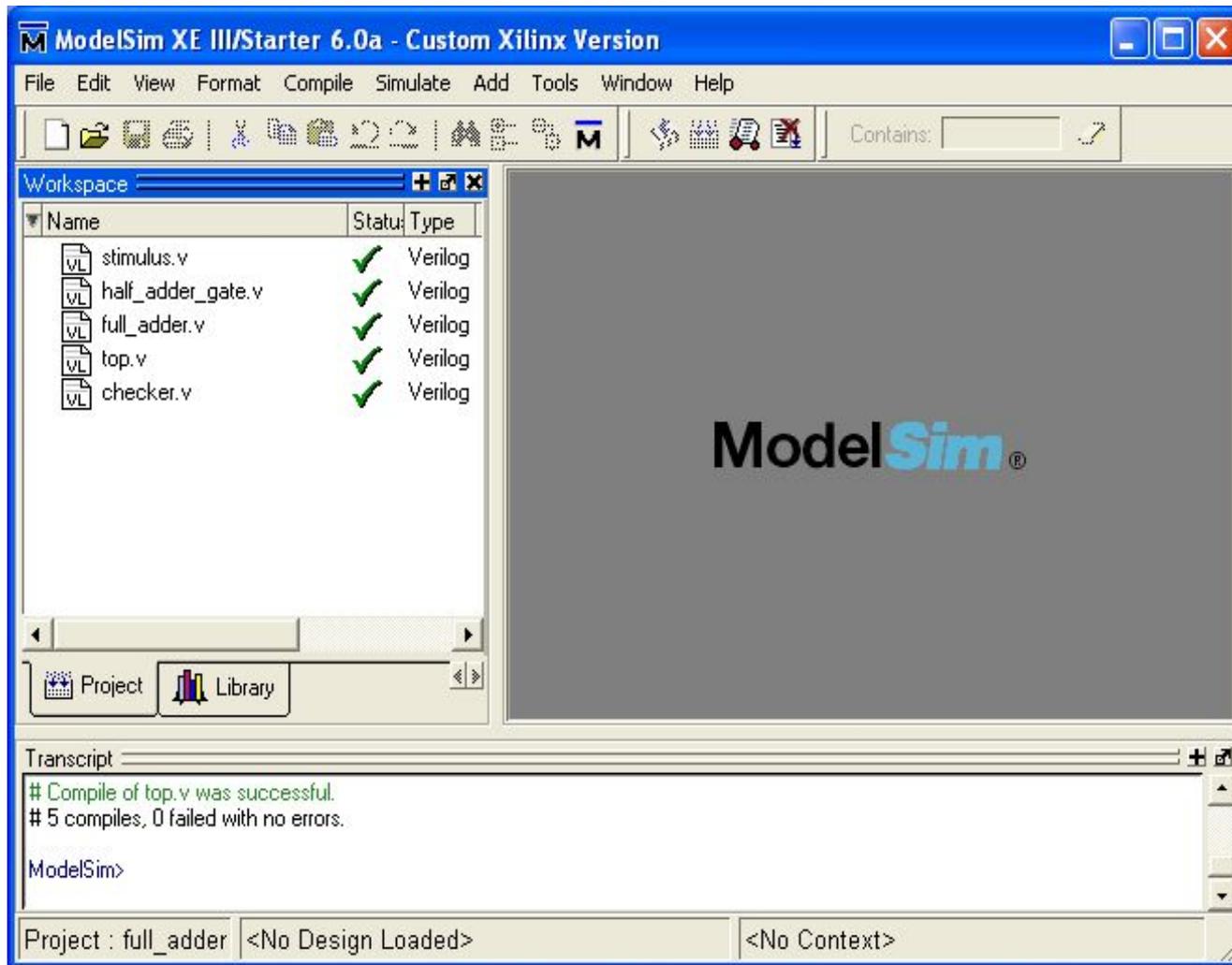
After adding files



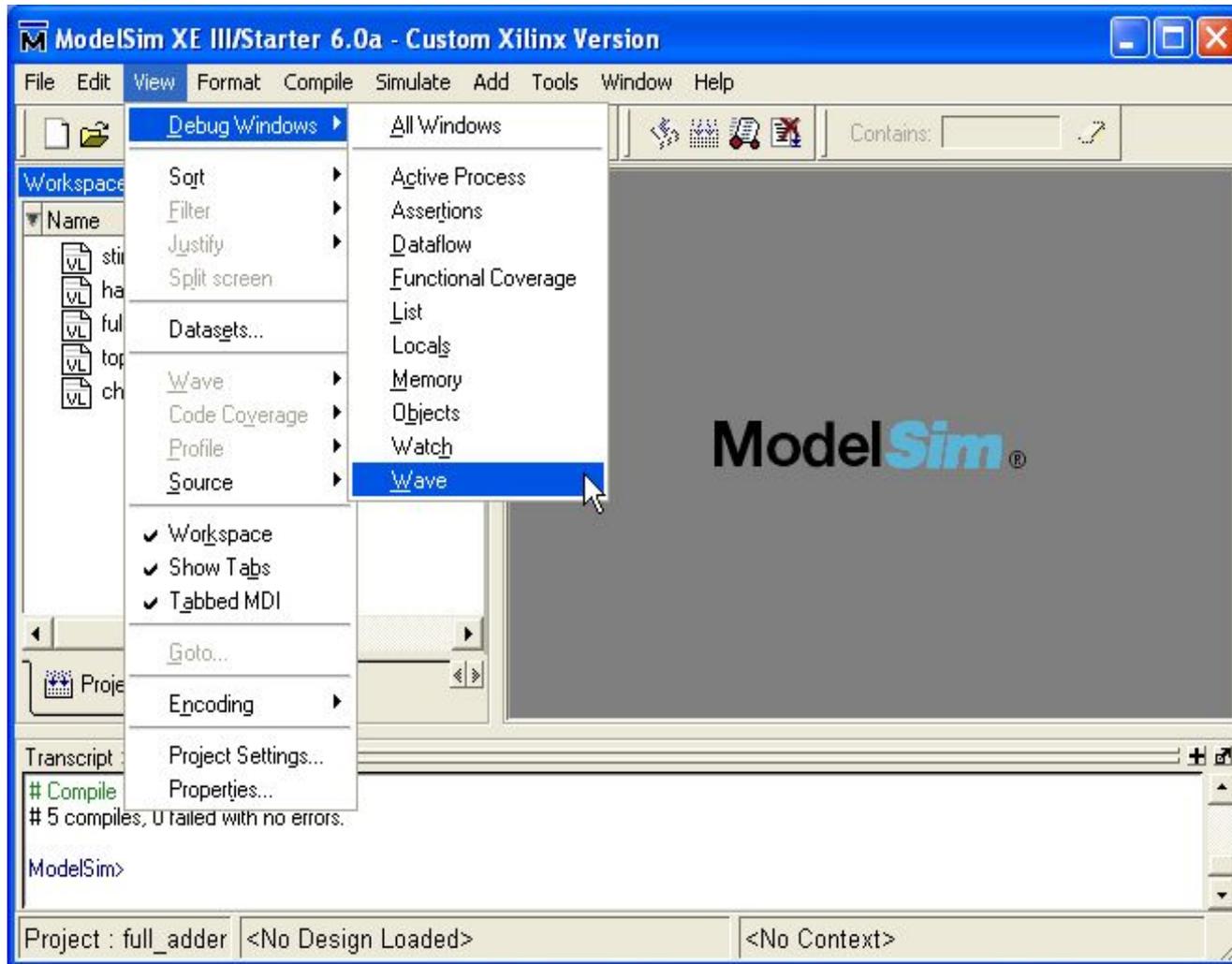
Compile->Compile All



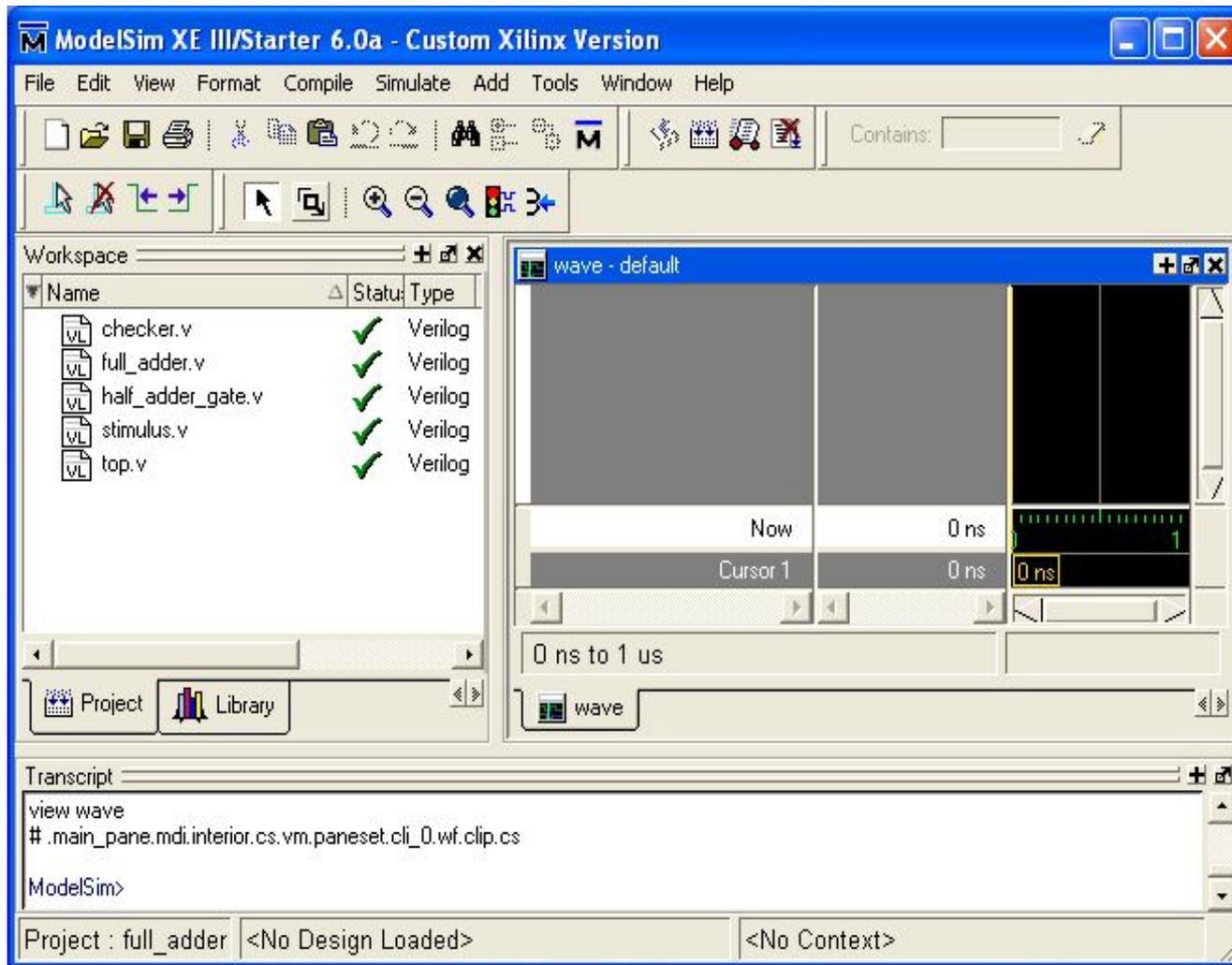
After compilation



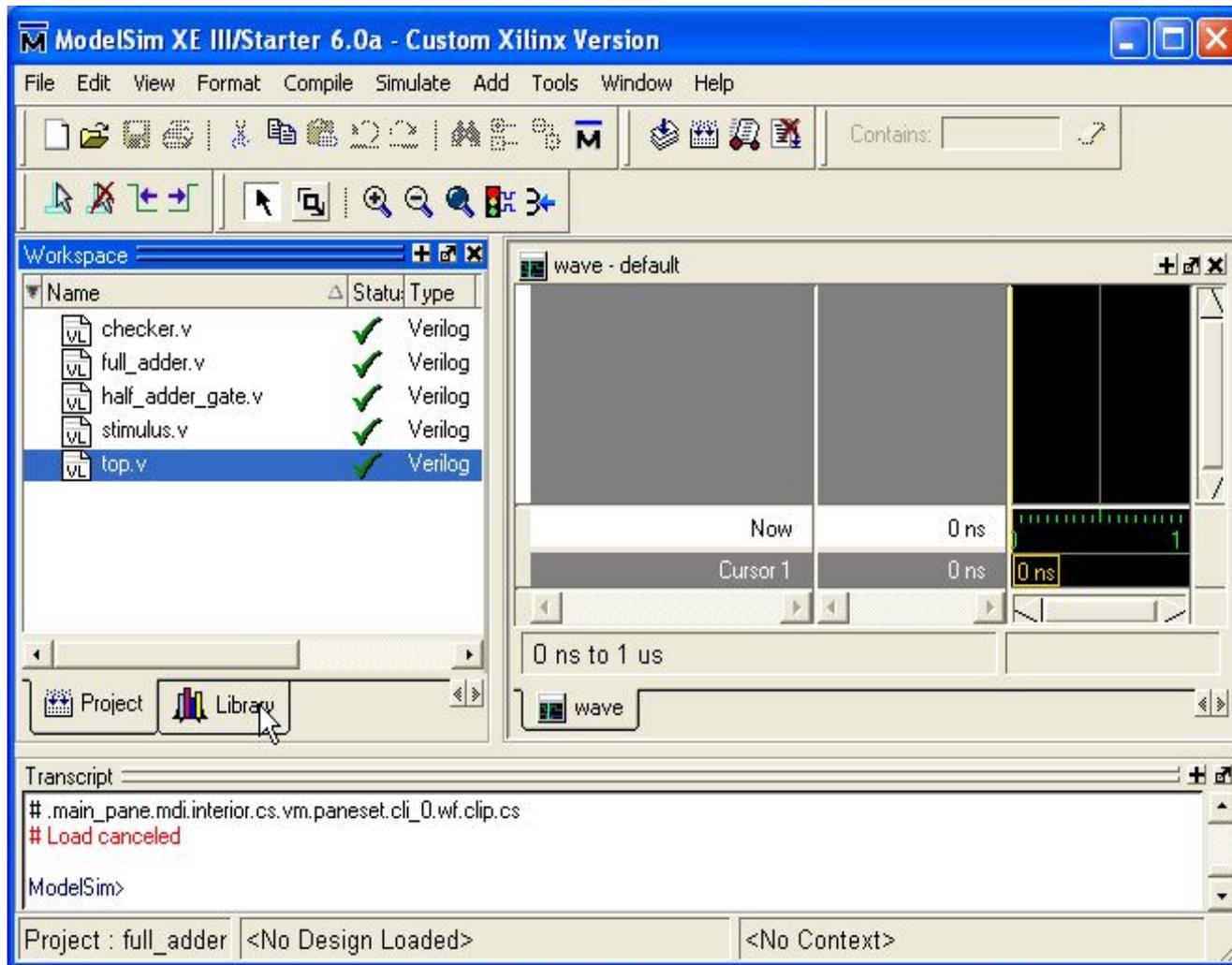
View->Debug Windows->Wave



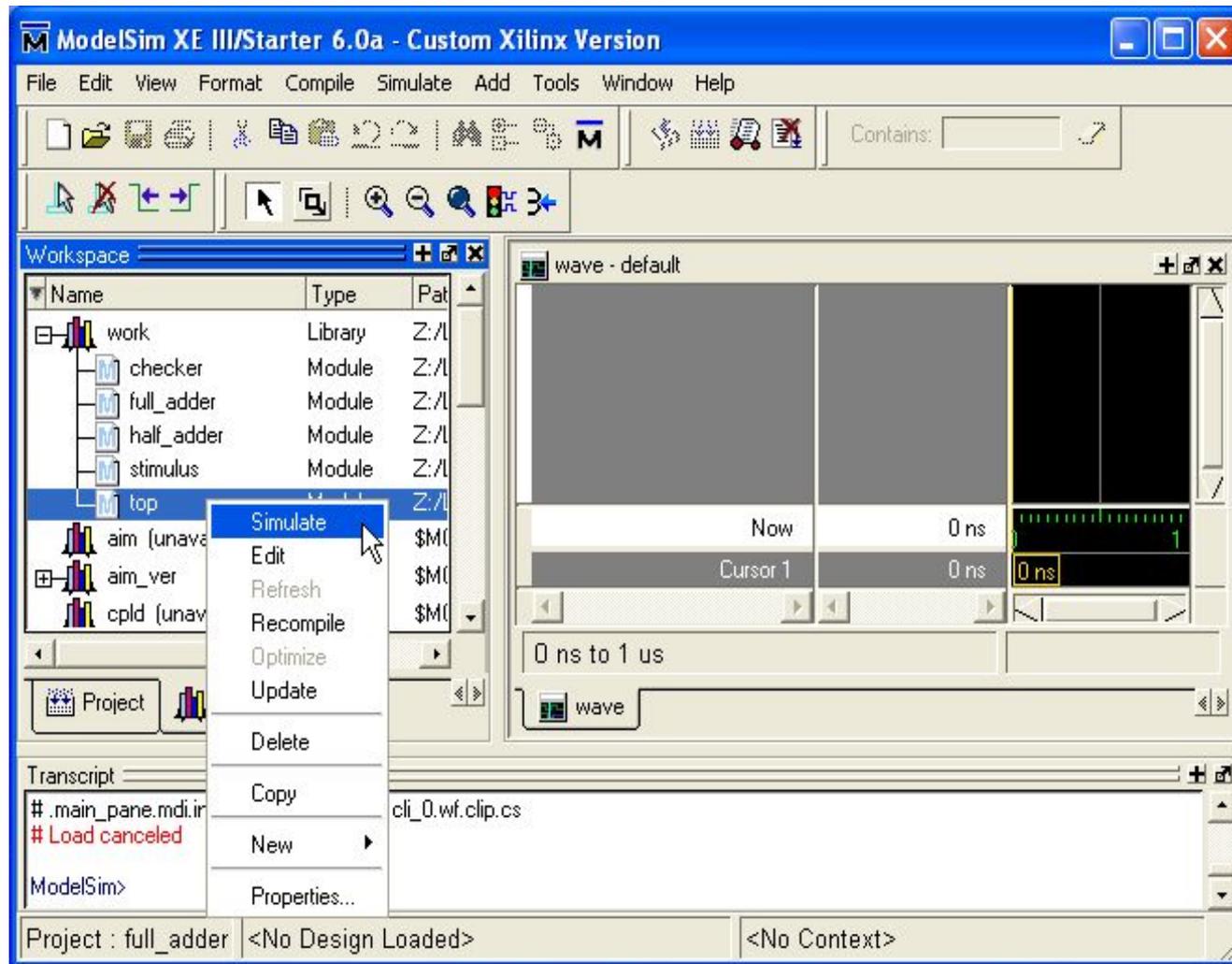
After adding wave window



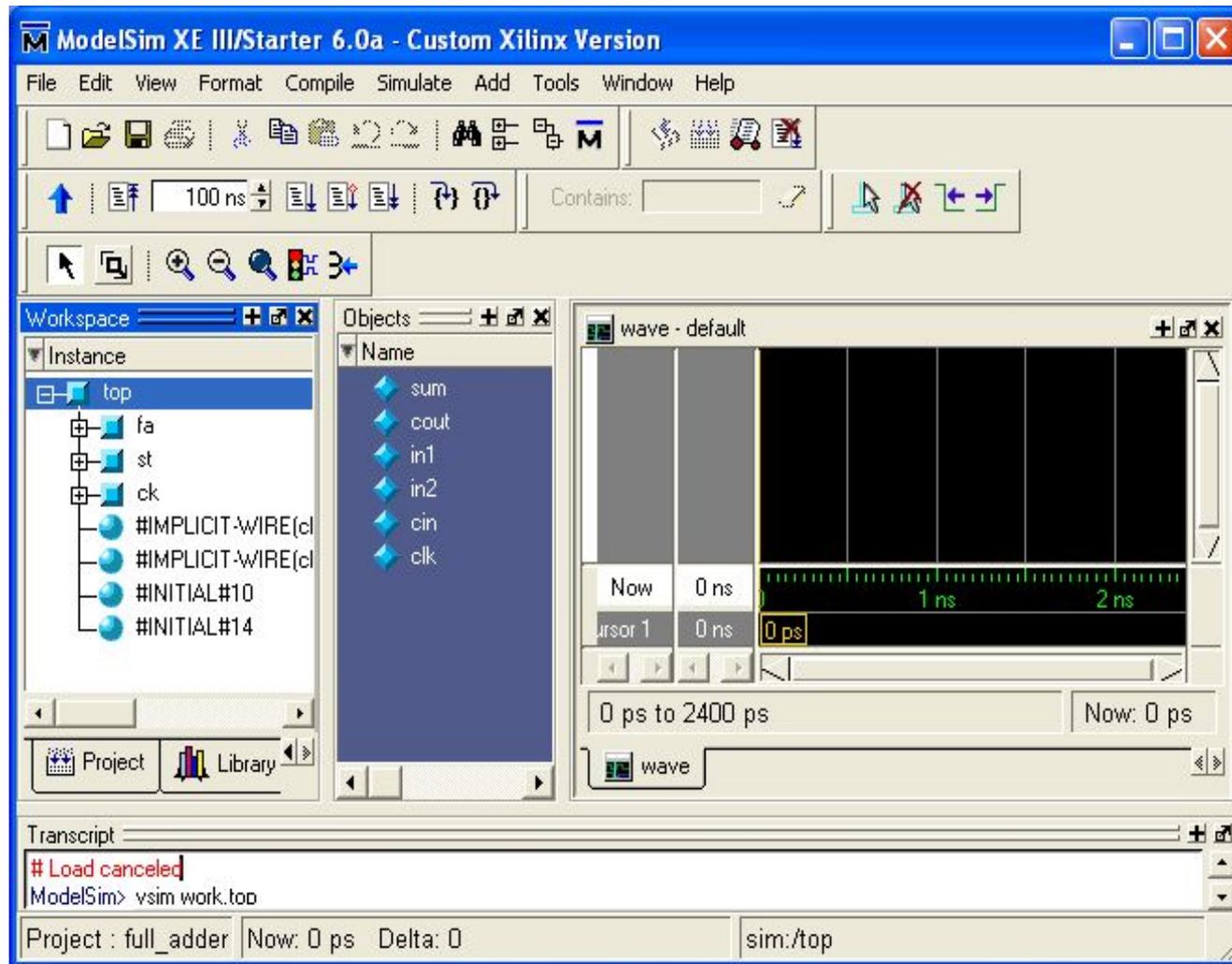
Select Library tab



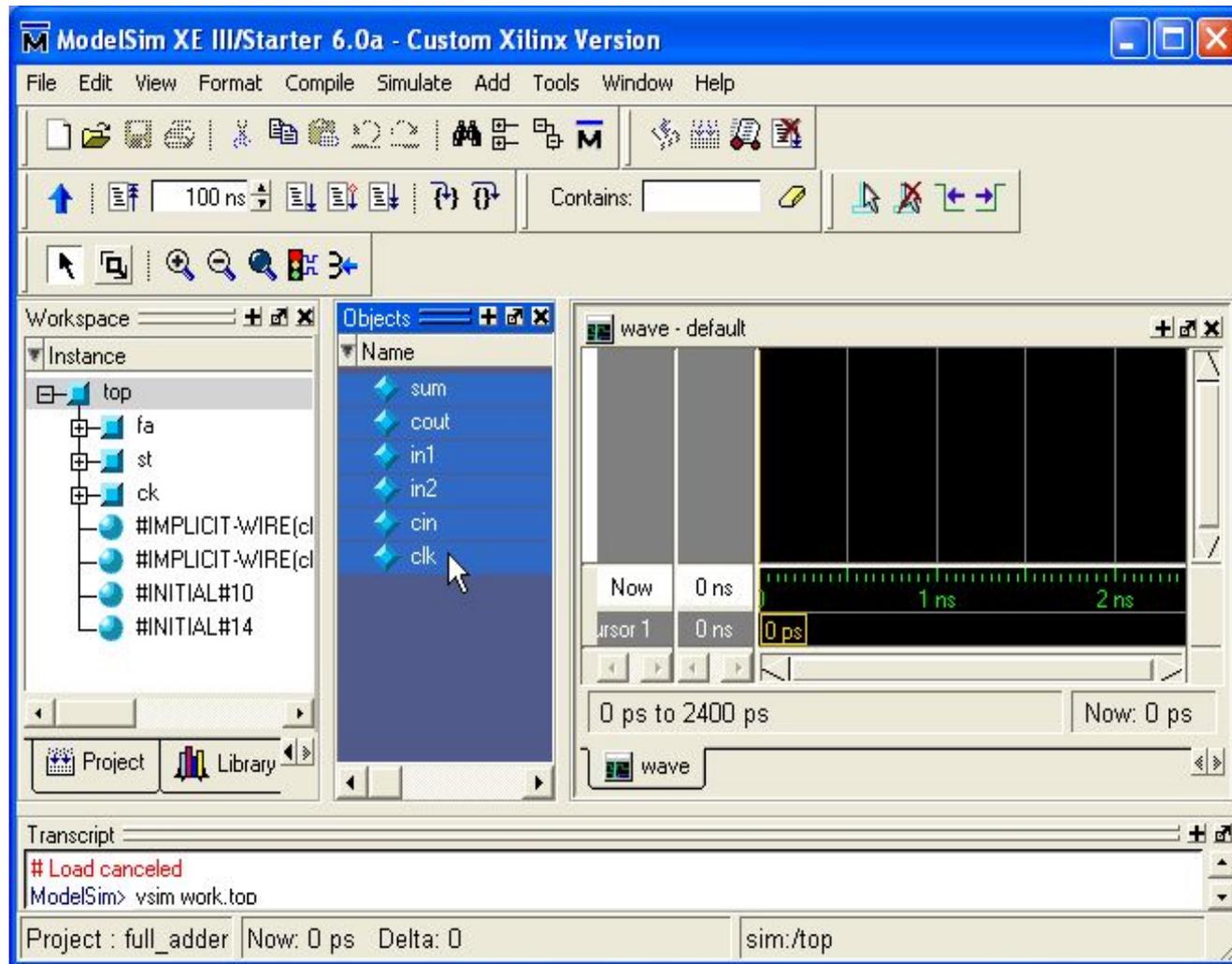
Run simulation with top-level



After simulation



Selecting signals to be view

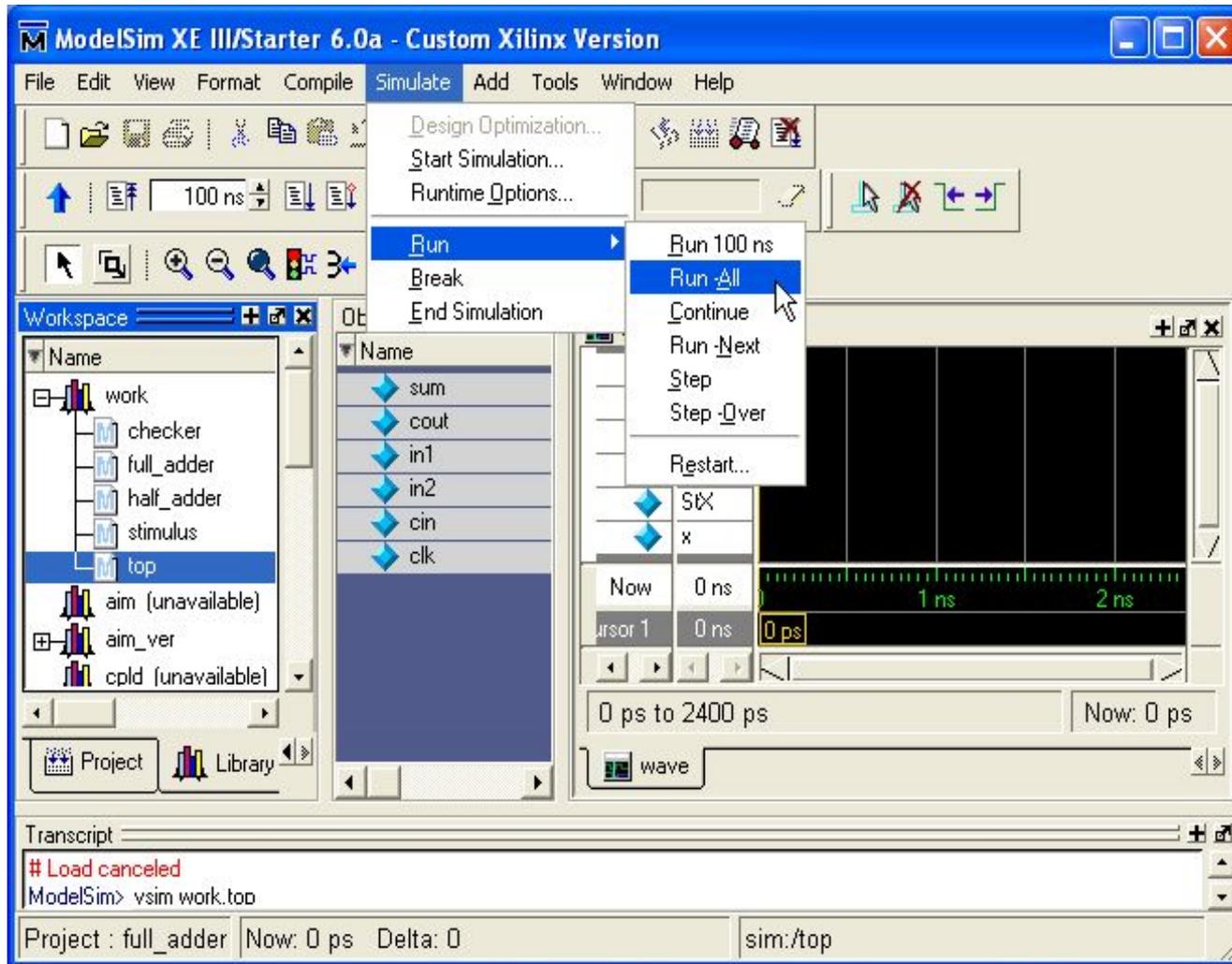


After selection

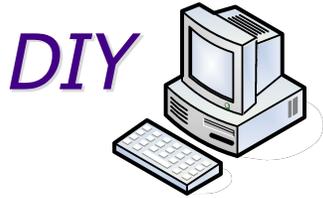
The screenshot displays the ModelSim XE III/Starter 6.0a - Custom Xilinx Version interface. The main workspace is divided into several panes:

- Workspace:** Shows a hierarchical tree structure under 'Instance' with 'top' expanded to show components like 'fa', 'st', 'ck', and several '#IMPLICIT-WIRE(c...)' and '#INITIAL#10'/'#INITIAL#14'.
- Objects:** A list of objects including 'sum', 'cout', 'in1', 'in2', 'cin', and 'clk', all of which are currently selected (highlighted in blue).
- wave - default:** A waveform viewer showing a list of signals: 'StX', 'StX', 'StX', 'StX', 'StX', and 'x'. The time axis is set from '0 ps to 2400 ps' with a scale of '1 ns' and '2 ns'. A cursor is positioned at '0 ps'.
- Transcript:** Shows the command '# Load canceled' and 'ModelSim> vsim work.top'.
- Status Bar:** Displays 'Project : full_adder', 'Now: 0 ps', 'Delta: 0', and 'sim:/top'.

Run-All



Create a new project



- Invoke ModelSim
- File □ New □ Project
- Specify 'Project Name' and 'Project Location'

The screenshot shows the ModelSim SE PLUS 5.8c interface. The 'File' menu is open, and the 'New' submenu is selected, with 'Project...' highlighted. The 'Create Project' dialog box is open, showing the 'Project Name' field with 'hello' and the 'Project Location' field with '_kut/codes/verilog/ex01_hello'. The 'Default Library Name' field contains 'work'. The 'OK' button is highlighted.

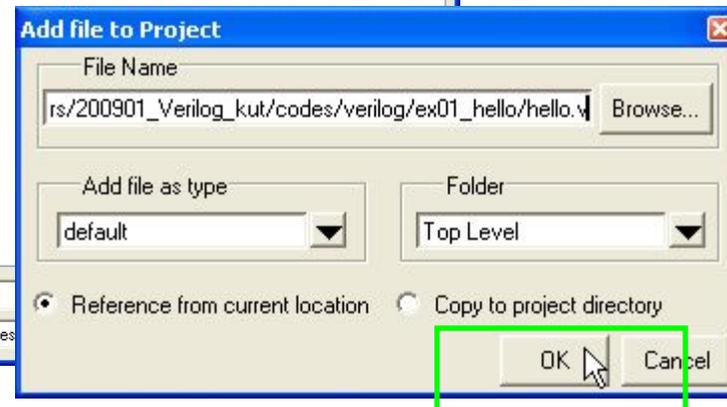
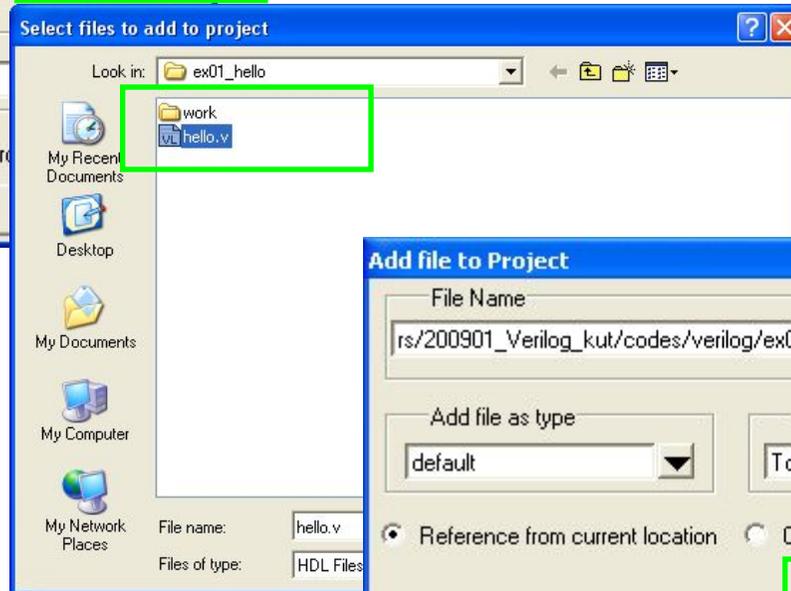
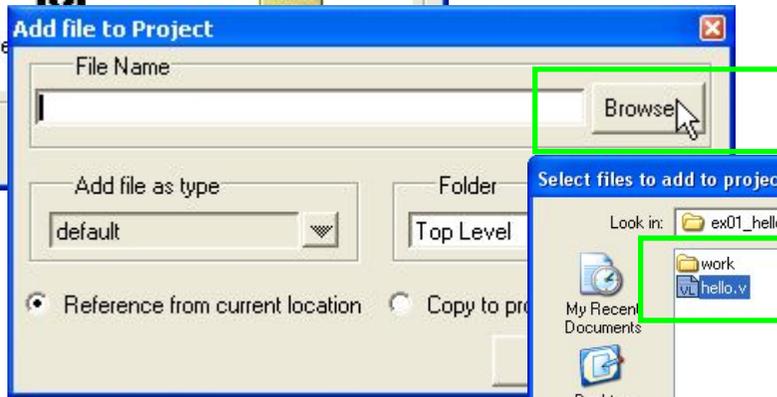
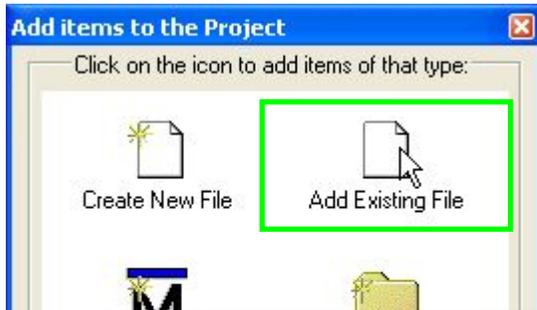
Workspace:

Name	Type	Path
vital2000	Library	\$MODEL_TECH
ieee	Library	\$MODEL_TECH
modelsim_lib	Library	\$MODEL_TECH
std	Library	\$MODEL_TECH
std_developerskit	Library	\$MODEL_TECH
synopsys	Library	\$MODEL_TECH
verilog	Library	\$MODEL_TECH

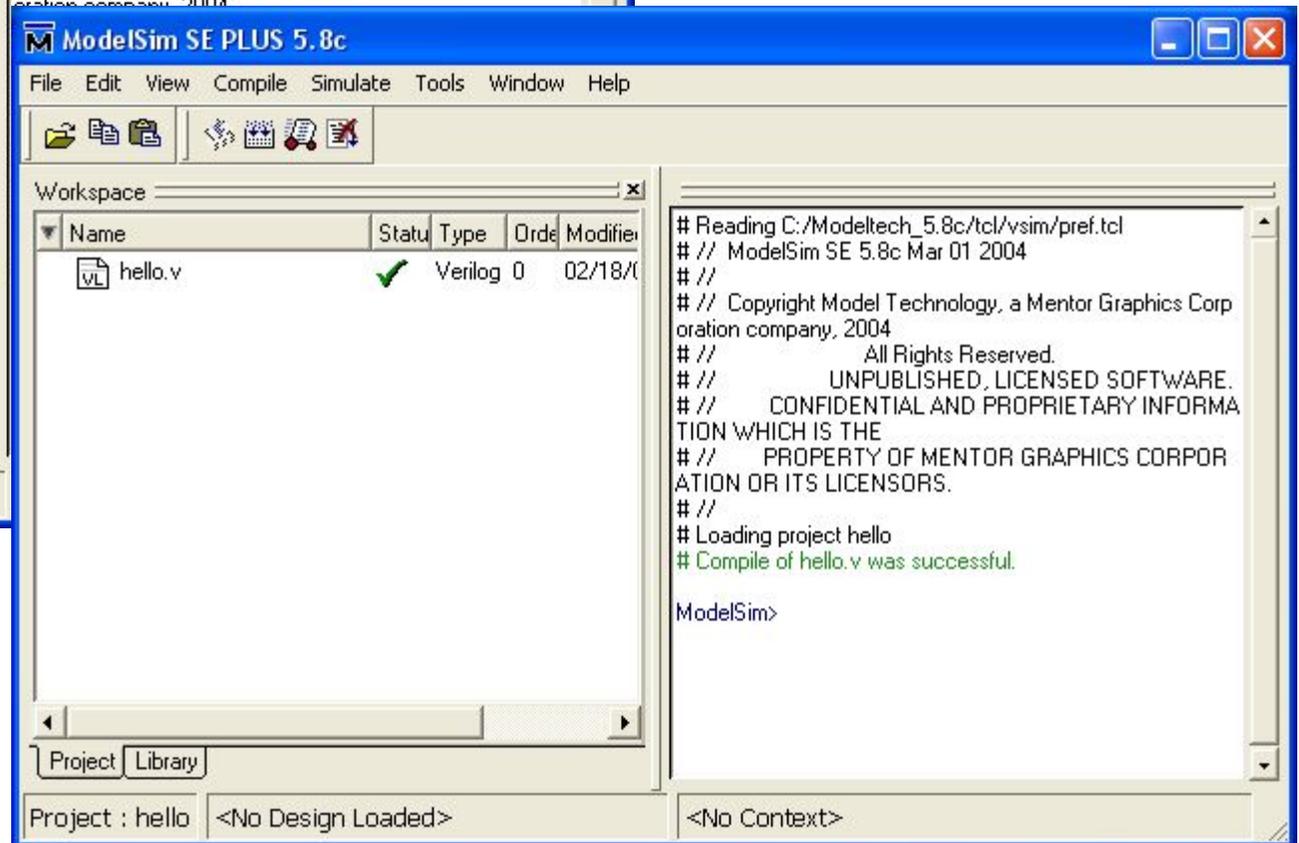
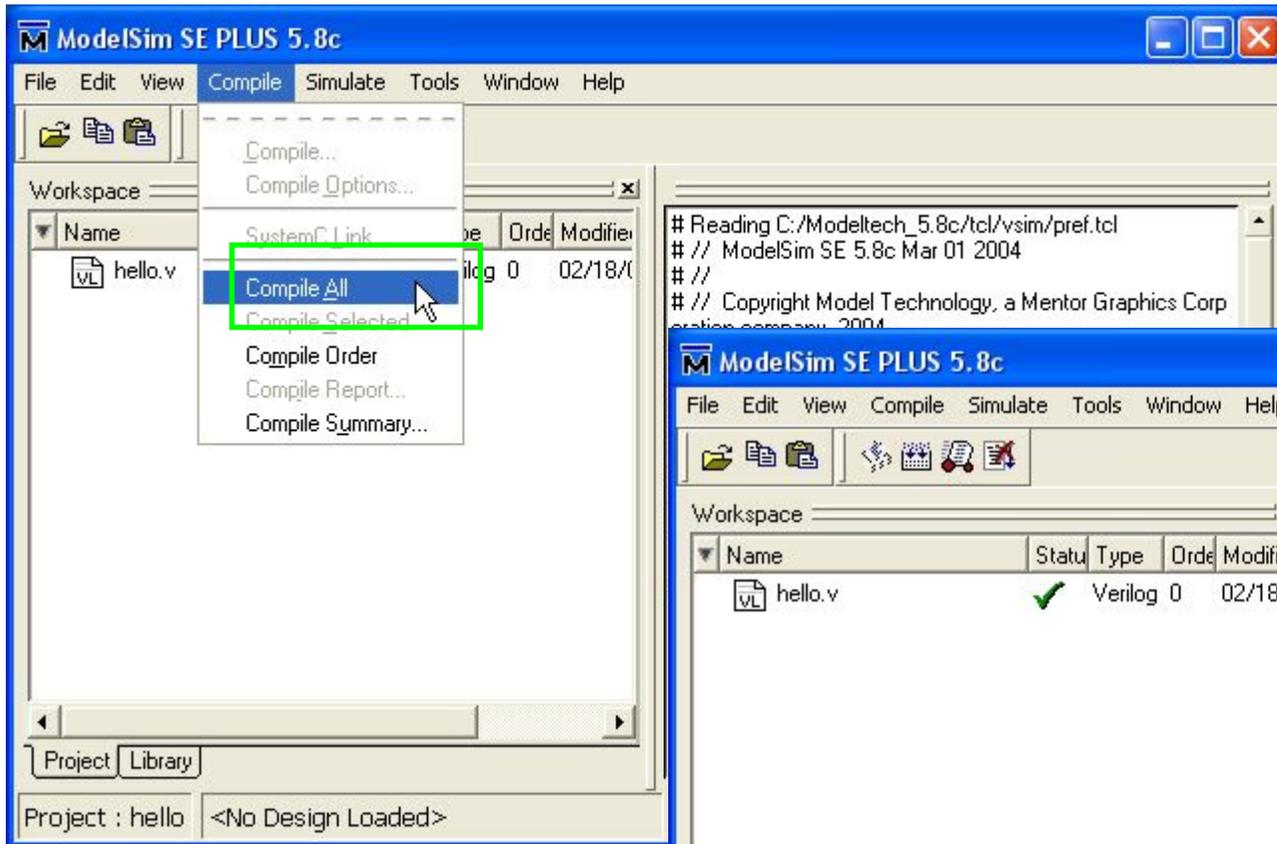
```
# Reading C:/Modeltech_5.8c/tcl/vsim/pref.tcl
# // ModelSim SE 5.8c Mar 01 2004
# //
# // Copyright Model Technology, a Mentor Graphics Corporation company, 2004
# // All Rights Reserved.
# // UNPUBLISHED, LICENSED SOFTWARE.
# // CONFIDENTIAL AND PROPRIETARY INFORMATION WHICH IS THE
# // PROPERTY OF MENTOR GRAPHICS CORPORATION OR ITS LICENSEES.
ModelSim>
```

Add existing file

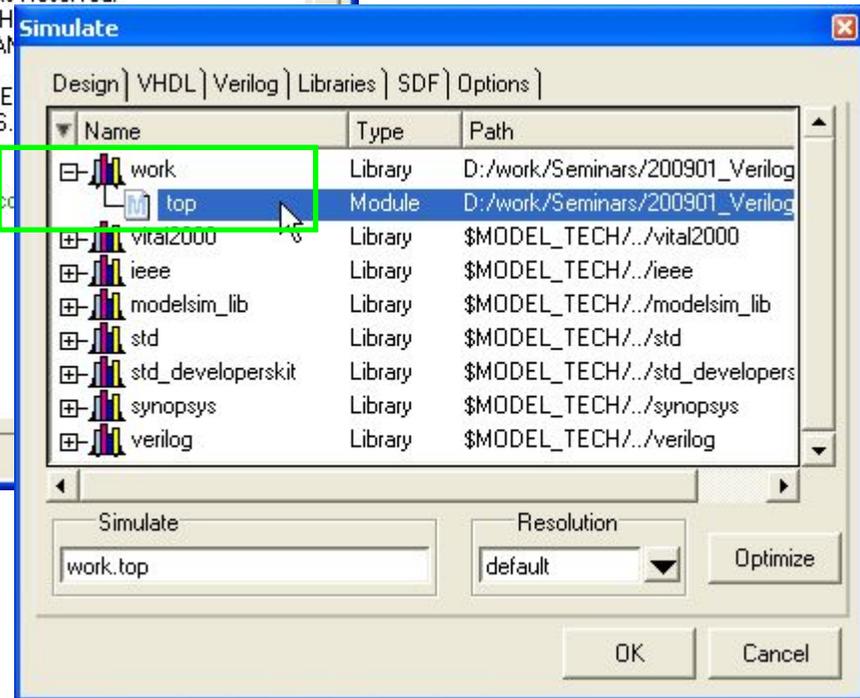
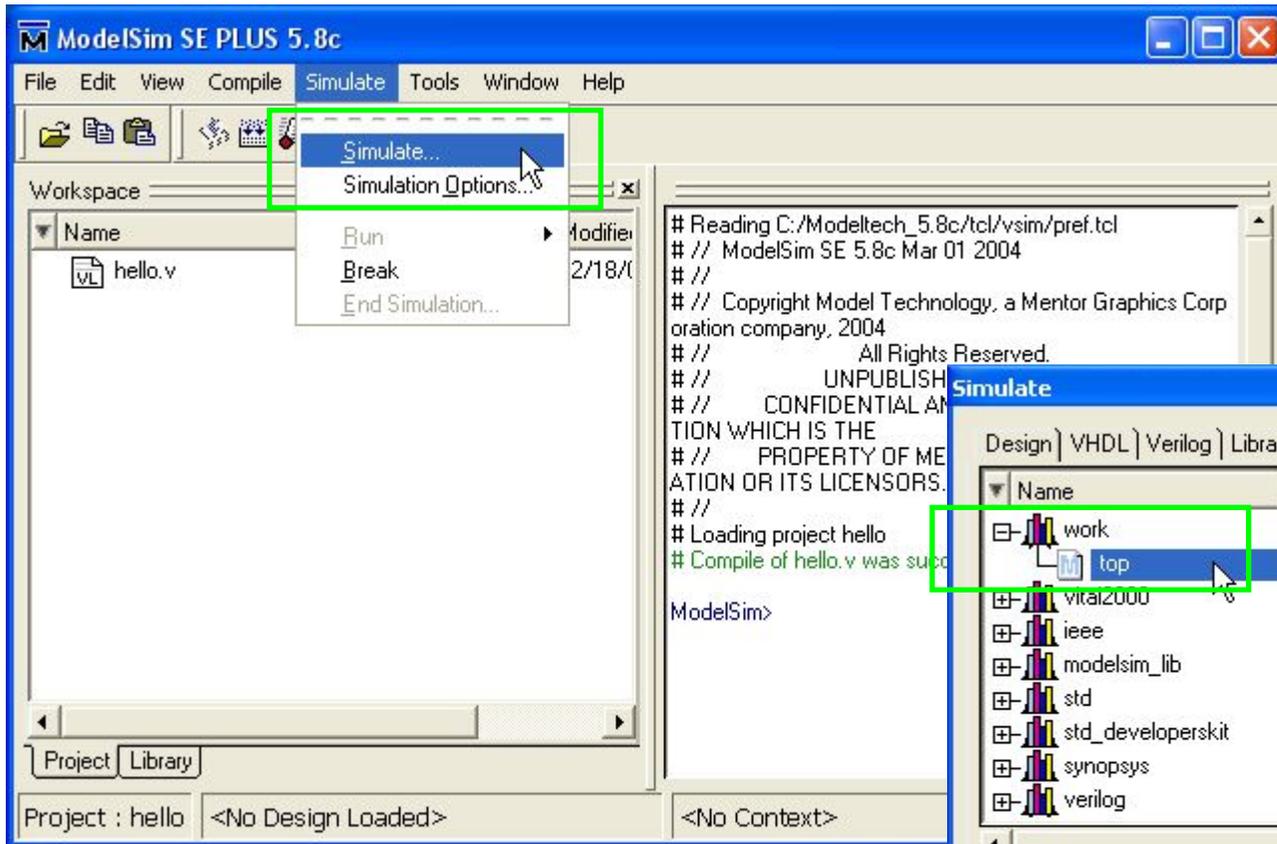
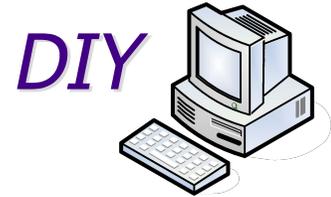
- Add the Verilog design file



Compile



Compile



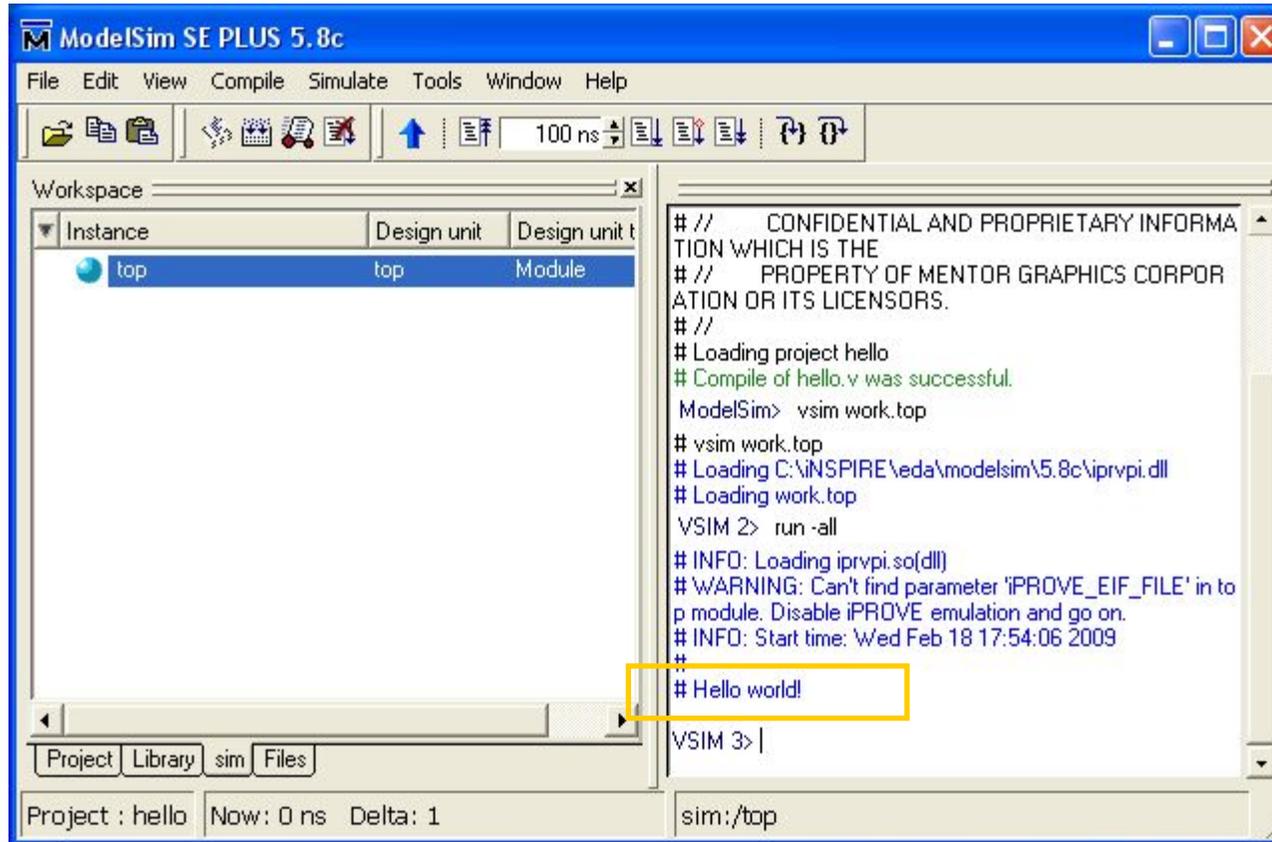
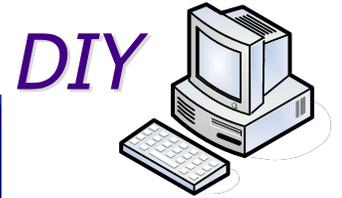
Compile



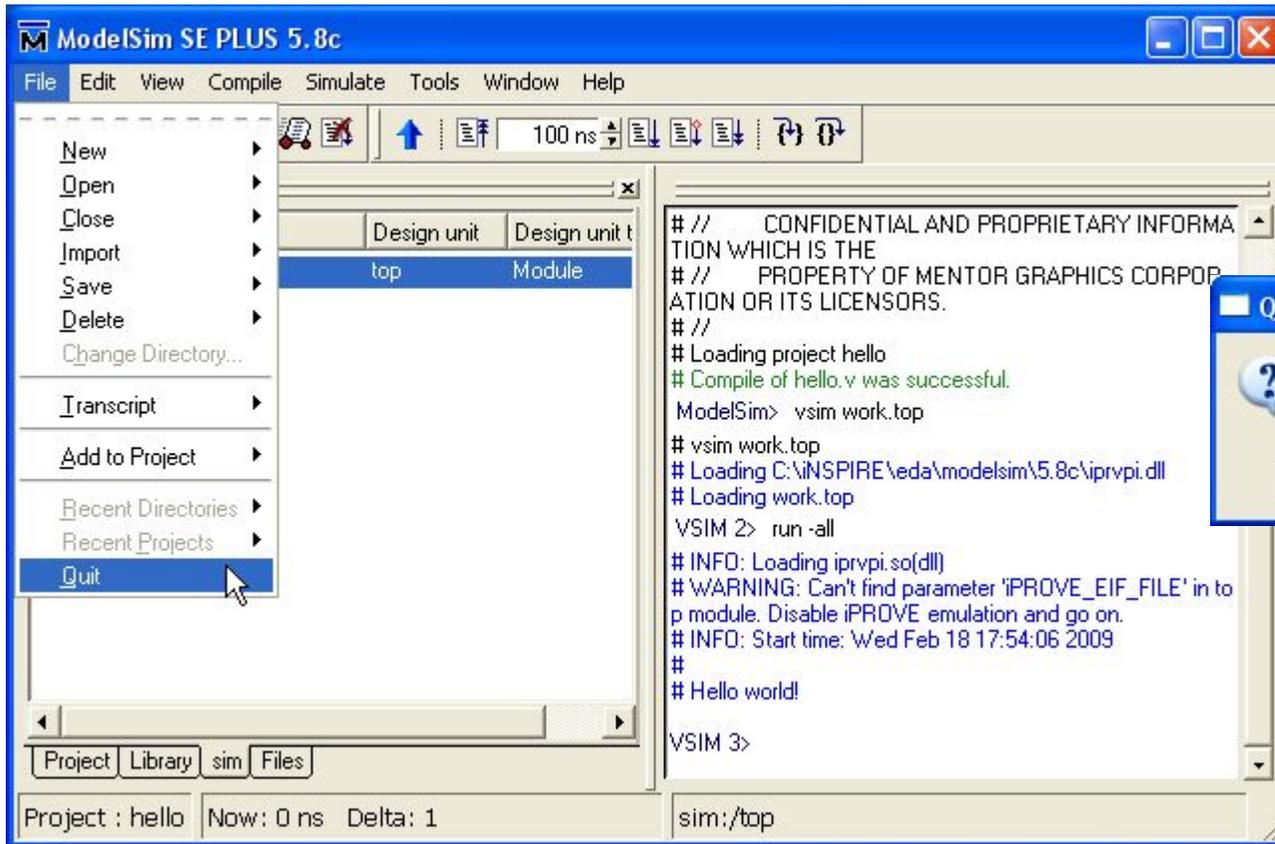
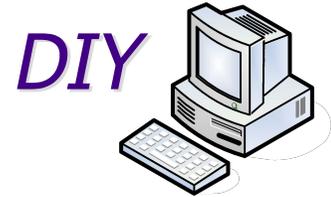
The image displays two overlapping screenshots of the ModelSim SE PLUS 5.8c software interface. The top-left screenshot shows the main workspace with a project named "hello" and a design unit "top". The bottom-right screenshot shows the "Simulate" menu open, with the "Run -All" option highlighted in a green box. The console window in the bottom-right screenshot shows the following output:

```
# Reading C:/Modeltech_5.8c/tcl/vsim/pref.tcl
# // ModelSim SE 5.8c Mar 01 2004
# //
# // Copyright Model Technology, a Mentor Graphics Corp
All Rights Reserved.
UNPUBLISHED, LICENSED SOFTWARE.
CONFIDENTIAL AND PROPRIETARY INFORMATION.
THIS IS THE
PROPERTY OF MENTOR GRAPHICS CORPORATION
OR ITS LICENSORS.
# //
# Loading project hello
# Compile of hello.v was successful.
ModelSim> vsim work.top
# vsim work.top
# Loading C:\INSPIRE\eda\modelsim\5.8c\iprvpi.dll
# Loading work.top
VSIM 2> |
sim:/top
```

Simulation



Quit



- There should be 'hello.mpf', which is ModelSim project file.