06.02.2011

ХНАДУ

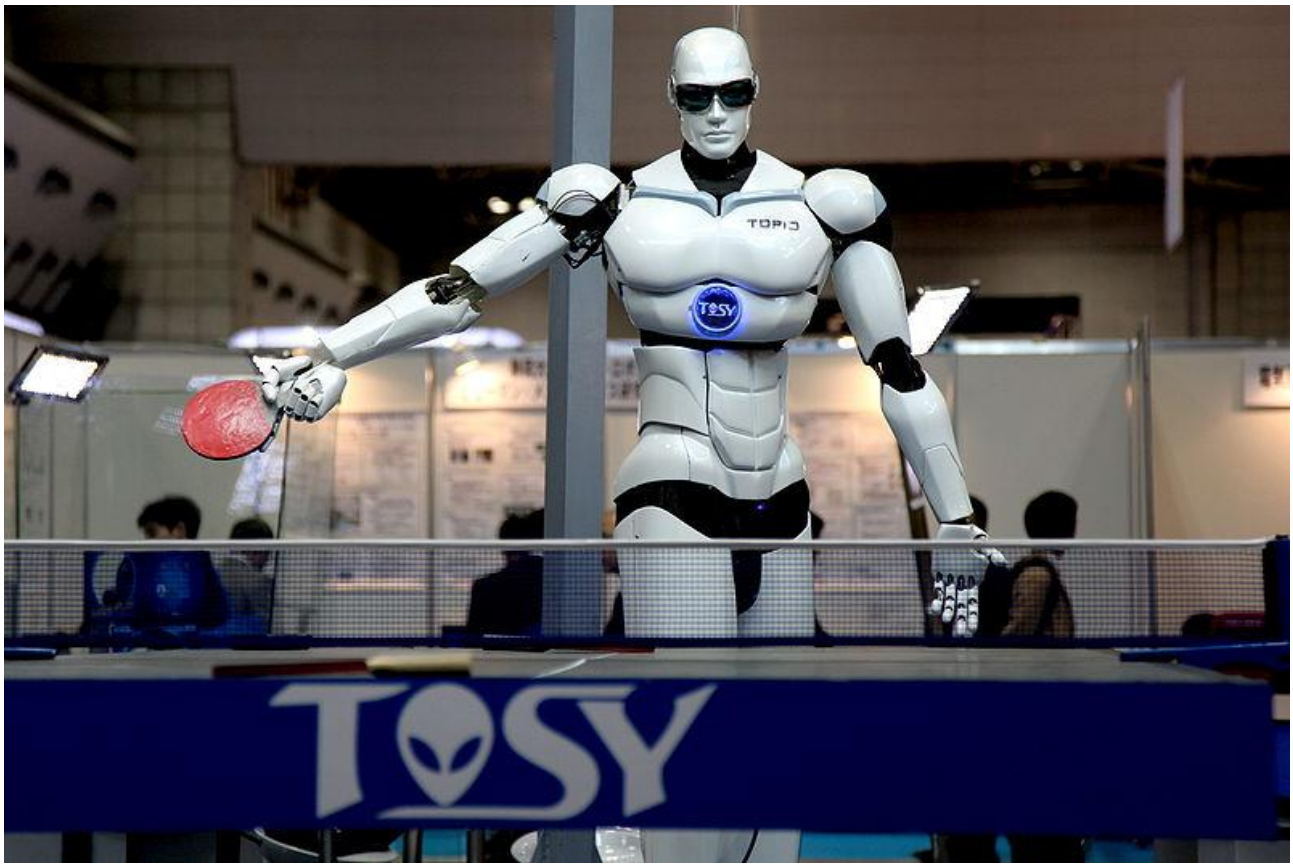
Гнучка автоматизація виробництв та робото-технічні комплекси

Конспект лекцій | Доцент АКІТ Кривенко С.А

Содержание

Введение

Учебная дисциплина изучается на третьем и четвертом курсах студентами, которые готовятся по направлению «Автоматизация и компьютерно-интегрированные технологии».



- Учебная дисциплина изучается в течение двух семестров.
- В весеннем семестре рассматривается 3 темы: 9 лекций и 18 часов лабораторных работ. Отчетность - зачет.
- В осеннем семестре рассматривается еще 6 тем: 18 лекций, 18 часов лабораторных работ и 9 практических занятий. Отчетность курсовая работа, модульный экзамен.
- В первой теме рассматриваются специфика понятий: проекта; стратегии проектирования; алфавита; ключевых слов; идентификаторов; имен; выражений языка. Приведенные неформальные примеры, которые иллюстрируют основные принципы текстового описания проекта.
- Второй тематический модуль посвящен конечным автоматам.
- В третьей теме изложены методы разработки проектов сложных систем.

Тематичний модуль 1.
Проектування цифрових апаратних засобів

Лекція 1. Ввідна

Лекція 2. Реалізація проекту

Лекція 3. Мова опису апаратури

Лекція 4. Вирази

1 Вводная

Agenda

1) Общие сведения о проектировании цифровых аппаратных средств
2) Примеры простых проектов

1.1 Общие сведения

Появление интегральных схем большой логической емкости, развитие средств автоматизации проектирование и уровень сложности цифровых аппаратных средств, которые производятся в современное время, определяет существенные изменения в методологии проектирования цифровых аппаратных средств.

Применение графического описания аппаратных средств, например, с помощью схем электрических принципиальных, которое базируется на ручном и в большинстве случаев на не формальном синтезе, остается в прошлом.

На смену приходит другая методология, в основе которой лежат два момента:

- текстовое описание;
- и автоматический синтез.

1.2 Примеры простых проектов

Раздел описания интерфейса модуля[1] позволяет задать имя модуля и имена его выводов.

Имя модуля должно совпадать с именем логического файла, в котором сохраняется текстовое описание.

Если модуль находится на верхнем уровне иерархии, то его имя должно быть именем проекта. Вспомогательные файлы проекта будут иметь это имя.

Правила применения раздела

Имя файла, в котором должно сохраняться текстовое описание данного модуля - First_project.tdf.

Раздел начинается со слова SUBDESIGN, за которым идет имя модуля.

Максимальная длина имени 32 символа.

Имя модуля должно совпадать с именем файла, в котором сохраняется его текстовое описание.

[1] Subdesign Section

Список выводов модуля.

После имени модуля в круглых скобках приведен список его выводов:

INPUT - вход

OUTPUT - выход

BIDIR - 2-направленный вывод

MACHINE INPUT - вход импортированных состояний автомата

MACHINE OUTPUT - выход экспортированных состояний автомата.

Выводы MACHINE INPUT и MACHINE OUTPUT не могут быть использованы в текстовом описании верхнего уровня иерархии, то есть при описании модуля, выводы которого являются выводами микросхемы программируемой логики PLD.

Выводы перечисляются через запятую в одну или несколько строк. В конце перечня однотипных выводов становиться двоеточие, дальше ключевое слово, которое указывает на тип вывода, еще дальше - точка с запятой.

Выводы перечисляются через запятую в одну или несколько строк. В конце перечня однотипных выводов становиться двоеточие, дальше ключевое слово, которое указывает на тип вывода, еще дальше - точка с запятой.

После слова INPUT может быть указано базовое значение (GND - логический нуль, VCC - логическая единица) входного сигнала.

Базовое значение будет подаваться на вход в том случае, если он окажется неподключенным при использовании данного модуля как компонента при описании модуля высшего уровня иерархии.

Если неподключенным окажется вход, для которого не указано базовое значение, компилятор пакета MAX+plusII выдаст сообщение об ошибке.

В файле с текстовым описанием данный раздел может использоваться только один раз.

Раздел описания логики[2] позволяет задать алгоритм работы проектируемого модуля.

Правила использования раздела следующие.

Раздел начинается с ключевого слова BEGIN и заканчивается ключевым словом END, за которым идет точка с запятой.

В разделе могут быть использованы следующие конструкции:

оператор задания начальных значений по умолчанию[3];

логическое уравнение[4];

логическое уравнение для управляющих сигналов[5];

оператор выбора вариантов[6];

оператор выбора условий[7];

оператор выбора текста[8];

непосредственное обращение к модулям[9];

[2] Logic Section
[3] Defaults Statement
[4] Boolean Equations
[5] Boolean Control Equations
[6] Case Statement
[7] If Then Statement
[8] If Generate Statement
[9] In line Logic Function reference

оператор цикла[10];

таблица логической функции[11];

оператор контроля[12].

Порядок использования логических уравнений может быть произвольным, потому что в отличие от языков программирования, компилятор осуществляет параллельное их выполнение.

Таблица истинности логической функции задается следующим образом:

Таблица истинности логической функции задается следующим образом:

```
TABLE
 _node _ name, _node _name => _node _name,_node _name;
 _input _ value, _input _value => _output _value, _output _value;
 _input _value, _input _value => _output _value, _output _value;
 _input _value, _input _value =>_output _value, _output _value;
END TABLE;
```

Правила для таблицы истинности

Открывает таблицу слово TABLE, а закрывают слова END TABLE, за которыми идет точка с запятой.

Первая после слова TABLE строка определяет форму таблицы. В ней через запятую перечисляются аргументы (внутренние переменные, входы или выходы модуля) и имена логических функций (внутренние переменные или выходы модуля).

Аргументы и функции разделяет символ стрелка (=>).

В конце строки ставится точка с запятой.

В следующих строках в соответствии с заданной формой называются наборы аргументов и значения логических функций.

Оператор выбора условий

На примере описания приоритетного шифратора, который задан в модуле TRUTH_TABLE таблицей, рассмотрим использование оператора IF THEN.

Он, в общем случае, позволяет последовательно оценить истинность нескольких логических выражений и в соответствии с полученными результатами выполнить то или другое действие.

Оператор IF THEN может использоваться в одной из следующих форм комбинаций операторов.

Три формы оператора IF THEN

```
IF _expression          IF _expression          IF _expression
THEN                      THEN                    THEN
_statement;                _statement;             _statement;
_statement;                _statement;             _statement;
ELSIF  _expression        ELSE                   END IF;
THEN                     _statement;
_statement;              _statement;
_statement;             END IF;
```

[10] For Generate Statement
[11] Truth Table Statement
[12] Assert Statement

```
ELSE
_statement;
_statement;
END IF;
```

1.2 Оператор выбора вариантов

Оператор CASE может использоваться в одной из следующих форм:

```
CASE _expression IS
WHEN _constant_value =>
_statement;
_statement;
WHEN _constant_value =>
_statement;
_statement;
WHEN OTHERS =>
_statement;
_statement;
END CASE;
```

```
CASE _expression IS
 WHEN _constant_value =>
_statement;
 _statement;
  WHEN _constant_value =>
   _statement;
   _statement;
    END CASE;
```

2 Реализация проекта

План лекции:
1) состав проектной документации;
2) процедура реализации проекта .

2.1 Состав проектной документации

Появление интегральных схем большой логической емкости, развитие средств автоматизации проектирование и уровень сложности цифровых аппаратных средств, которые производятся в современное время, определяет существенные изменения в методологии проектирования цифровых аппаратных средств.

Применение графического описания аппаратных средств, например, с помощью схем электрических принципиальных, которое базируется на ручном и в большинстве случаев на не формальном синтезе, остается в прошлом.

На смену приходит другая методология, в основе которой лежат два момента:

текстовое описание;

и автоматический синтез.

Новая методология

Текстовое описание - это применение языковых конструкций высокого уровня для задания алгоритмов работы устройств. Он отвечает применению языков описания аппаратуры.

Автоматический синтез - это процедура формального перевода текстового описания в схемное описание на заданном элементном базисе, которая выполняется системой автоматизации проектирования.

Начиная изучать новую для вас методологию реализации проекта необходимо определить специфику применения следующих понятий: проект, стратегия проектирования и инструментальные средства проектирования.

Дальше следует приступить к разработке проектов, углубляя в ходе проектирования знания инструментальных средств, в том числе, языка.

Под термином проектная документация в рамках пакета MAX+plusII (QuartusII) понимается набор файлов, которые связаны с модулем, который проектируется.

Выделяют две группы файлов: логические файлы, которые описывают алгоритм работы устройства[13], и вспомогательные файлы.

[13] Design Files

Проектная документация может иметь один логический файл или несколько логических файлов, которые образуют иерархическое описание проектируемого модуля.

При иерархическом описании среди множества логических файлов различают: файл верхнего уровня в иерархии описания[14], файлы нижних (одного или нескольких) уровней иерархии[15].

Логический файл - это файл одного из следующих типов:

Graphic Design File (стандартное расширение - .gdf). В файле находится схема, образованная в рамках пакета MAX+plus II;

AHDL Text Design File (стандартное расширение - .tdf). В файле находится текстовое описание модуля на языке AlteraHDL;

Waveform Design File (стандартное расширение - .wdf). В файле находятся временные диаграммы входных и выходных сигналов, которые образованы в рамках пакета MAX+plus II;

VHDL Design File (стандартное расширение - .vdf). В файле находится текстовое описание модуля на языке VHDL;

Verilog Design File (стандартное расширение - .v). В файле находится текстовое описание модуля на языке Verilog HDL;

Orcad Schematic Files (стандартное расширение - .sch). В файле находится схема, образованная в рамках пакета ORCAD;

EDIF Input Files (стандартное расширение - .edf). В файле находится описание в формате EDIF 200 или 300;

Xilinx Netlist Format File (стандартное расширение - .xnf). Файл содержит описание модуля, которое получено в рамках пакета фирмы Xilinx.

Вспомогательные файлы хранят дополнительную информацию о проекте. Их имена совпадают с именем проекта.

2.2 Процедура реализации проекта

Пакет MAX+plusII позволяет реализовывать стратегию нисходящего (сверху вниз) и восходящего (снизу вверх) проектирования.

Восходящее проектирование можно применять в том случае, когда для описания цифрового устройства существует детальное структурное описание, выполненное в элементном базисе, отличающемся от уже существующего в распоряжении разработчика микросхемы. Например, схема электрическая принципиальная. При этом разработчик решает следующие задачи:

создание функциональных аналогов элементов, которые используются в заданном описании;

доработка созданных элементов;

сборку образованных компонентов в единственном модуле;

моделирование и налаживание цифрового модуля в целом.

[14] Top-level Design File
[15] Low-level Design files

Таким образом, в процессе проектирования разработчик сначала создает модули нижнего уровня в иерархии описания, а дальше - модуль верхнего уровня. Отсюда и название стратегии проектирования.

Упрощенно, ориентируясь на возможности пакета MAX+plusII, процедура ниспадающего проектирования выглядит следующим образом:

разработка архитектуры микросхемы PLD. Начальное алгоритмическое описание поведения превращается в структурное описание, элементами которого являются архитектурные модули;

архитектурные блоки либо описываются на функциональном уровне с помощью языка AHDL, либо осуществляется их структурное описание, элементами которого являются функциональные модули;

итерационная процедура повторяется до тех пор, пока все функциональные модули не будут описаны на функциональном уровне;

после этого осуществляется функциональное моделирование модулей, которые имеют описание поведения.

моделирование и налаживание устройства в целом.

В процессе проектирования разработчик опускается из верхнего уровня иерархии описания, уровня микросхемы программируемой логики PLD, к нижним уровням. Отсюда и название стратегии проектирования.

3 Язык описания аппаратуры

План лекции:

1) символы

2) ключевые слова, идентификаторы, имена

3) группы, числа.

3.1 Символы

В алфавит языка AHDL входят: заглавные и прописные буквы латинского алфавита (A,B,.Z,a,b,.,z); арабские цифры (0,1,2,3,4,5,6,7,8,9) и специальные символы.

3.2 Ключевые слова, идентификаторы, имена

В языке AHDL определены три типа имен: символическое имя (Symbolic name);имя модуля (Subdesign name) и имя вывода (Port name).

Символическое имя - имя, которое определяется пользователем и используется для задания: переменных, постоянных, состояний и разрядов цифрового автомата; параметров; арифметических выражений; именуемых операторов.

Имя модуля - имя, которое определяется пользователем и используется для именования модуля, а также логического файла, в котором сохраняется его текстовое описание.

Имя вывода модуля - имя, которое определяется пользователем и служит для обозначения: входа модуля, выхода модуля; 2-направленного вывода модуля; входа для импортированного конечного автомата, выхода для экспортированного конечного автомата.

Имя может быть задано в одной из двух форм: строкой символов (от 1 до 32 символов) - Unquoted name; строкой символов (от 1 до 32 символов) в единичных дужках - Quoted name.

3.3 Группы, числа.

В языке AHDL определенно понятие группы - набор однотипных переменных или выводов микросхемы.

Порядковый номер переменной (вывода) в группе - индекс.

Максимальное количество переменных или выводов, которые объединены одной группы, не должна превышать 256.

В языке AHDL определенно три типа групп:

одномерные (Single-range);

двухмерные (Dual-range);

последовательные (Sequential) или, что больше подходит по сути, временные.

Примеры групп

A[5..0] - одномерная группа.

Индексы заданы в последовательности уменьшения слева направо. Элементы группы A5, A4 A3, A2, A1, A0.

CONSTANT UPPER = H «4»; B[UPPER..1] - одномерная группа.

Верхнюю границу диапазона изменения индексов определенно константой. Индексы заданы в убывающей слева направо последовательности . Элементы группы B4, B3, B2, B1.

C[2*2..0][3..2] - двухмерная группа.

Верхняя граница диапазона изменения первой группы индексов задана арифметическим выражением. Индексы заданы в ниспадающей последовательности. Элементы группы C43, C42, C33, C32, C23, C22, C13, C12 C03, C02.

D[1..4] - одномерная группа.

Индексы заданы в растущей слева направо последовательности. Элементы группы: D1, D2, D3, D4.

(A, E[2..1], F) - временная группа.

Элементы группы: A, E2, E1, F.

Числа.

Язык AHDL допускает использование десятичных, двоичных, восьмеричных, шестнадцатеричных чисел.

Десятичное число определяется как совокупность цифр от 0 до 9.

Бинарное число записывается в виде:

B «совокупность символов 0, 1, x»

или b «совокупность символов 0, 1, x».

Восьмеричное число:

O «совокупность цифр от 0 до 7»;

или o«совокупность цифр от 0 до 7»;

или Q «совокупность цифр от 0 до 7»;

или q «совокупность цифр от 0 до 7».

Шестнадцатилетнее число:

H «совокупность цифр от 0 до 9 и букв от A к F»;

или h«совокупность цифр от 0 до 9 и букв от A к F»;

или X «совокупность цифр от 0 до 9 и букв от A к F»;

или x «совокупность цифр от 0 до 9 и букв от A к F».

Примеры:

B «0x01»;

X «A1»;

O «73».

Десятичное число не может быть назначено 1-разрядной переменной или выходу. Вместо числа следует использовать константную единицу (b «1», или VCC) или константной нуль (b «0», или GND).

Примеры:

out = b «1» верное присвоение;

out = GND - верное присвоение;

out = 0 - неверное присвоение.

4 Выражения

План лекции:
1) Арифметические выражения;
2) логические выражения.

4.1 Арифметические выражения

В языке AHDL определены два типа выражений
(Expressions):
арифметические выражения
(Arithmetic Expressions);
логические выражения
(Boolean Expressions).
Арифметические выражения
Арифметические выражения используются для задания:
обозначенного выражения в операторе определения
(Define Statement);
постоянной в операторе задания констант
(Constant Statement);
границ диапазона изменения индексов одномерных и двухмерных групп;
границ диапазона внутренней переменной в операторе
FOR GENERATE;
оцениваемого выражения в операторах
IF GENERATE, ASSERT.
Примеры
DEFINE MAX(a, b)= ((a<b)?a:b);
((a<b)?a:b) -- арифметическое выражение;
CONSTANT Const = 1+2DIV3+LOG2(256);
1+2DIV3+LOG2(256) -- арифметическое выражение;
A[2+4*2..3-2]:INPUT;
2+4*2 и 3-2 -- арифметические выражения.

Значение арифметических выражений оценивается компилятором на этапе проверки синтаксиса проекта, потому для их реализации не нужны логические ресурсы микросхемы программируемой логики PLD.

4.2 Логические выражения

Логические двоичные выражения (Boolean expressions) содержат аргументы, которые соединяются: логическими операторами, арифметическими операторами и операторами сравнения.

Двоичные выражения используются:

в уравнениях (Boolean Equations);

в уравнениях, которые задают управляющий сигнал (Boolean Control Equations);

в операторе CASE;

в операторе IF THEN.

Операторы инверсии

Операторы инверсии (NOT) могут быть применены к одному из трех типов аргументов.

Первый тип - это 1-разрядная переменная (именуемая линия связи), 1-разрядный вывод модуля, а также константная логическая единица и константный логический нуль. При этом осуществляется инвертирование аргумента.

Второй тип - группа переменных (группа именуемых линий связи), группа выводов модуля. В этом случае инвертируется каждый элемент группы, например

!(a,b[2..1]) соответствует !a, !b2, !b1.

Третий тип - число. В этом случае инвертируется каждый разряд бинарного эквивалента данного числа, например

!9 = (!B «1001» = B «0110») = 6

Операторы AND, NAND, OR, NOR, XOR, NXOR

Операторы AND, NAND, OR, NOR, XOR, NXOR допускают использование следующих комбинаций аргументов.

Оба аргумента 1-разрядные (1-разрядная переменная, 1-разрядный вывод модуля, логический нуль, логическая единица).

Оба аргумента - группы (группа переменных, группа выводов модуля). В этом случае логический оператор применяется поразрядно, потому аргументы должны иметь одинаковое количество разрядов, совпадающее с числом результата, например:

R[4..1]=(A,B,C,D)#OP[3..0];

при этом: R4 = A#OP3; R3 = B#OP2; R2 = C#OP1; R1 = D#OP0;

Один аргумент - группа, а другой аргумент 1-разрядный

Один аргумент - группа (например, группа переменных, группа выводов модуля), а другой аргумент 1-разрядный (1-разрядная переменная, 1-разрядный вывод модуля, логический нуль, логическая единица).

В этом случае 1-разрядный аргумент тиражируется и формируется группа, количество разрядов которой равняется количеству разрядов в другом аргументе. Дальше до двух групп разряд к разряду применяется логический оператор, например:

R[3..1]= (a,b[2..1])&D;

при этом: R3 = a&D; R2 = b2&D; R1 = b1&D.

Потому как естественной формой представления чисел для выполнения процедуры синтеза комбинационной схемы является бинарная форма, то в этом случае оба аргумента рассматриваются компилятором как бинарные числа (группы бинарных разрядов).

Если для бинарного представления аргументов необходимое разное количество разрядов, то автоматически осуществляется наращивание количества разрядов бинарного представления меньшего аргумента. Логический оператор применяется к сформированным группам поразрядно, например:

3$14 = B"0011"$B"1110"=B"1101"=13.

Один аргумент - число, а второй аргумент либо 1-разрядный, либо группа

В этом случае число превращается в бинарное число (группу бинарных разрядов). Логический оператор применяется к этим группам разряд к разряду.

Если количество разрядов в бинарном числе меньше количества разрядов в другом аргументе, то она автоматически расширяется. Если больше, то процессор сообщений (Message Processor) пакета MAX+plusII выдает сообщение об ошибке, например:

R[3..1]= (a,b,c)&3; при этом: R[3]= (a&B"0") = 0; R[2]= (b&B"1") = b; R[1]= (c&B"1") = c;

R[3.. 1]= (a,b,c) & 9 будет сформировано сообщение об ошибке.

Арифметические операторы

Правила использования операторов сложения и вычитания

Аргументами данных операторов могут быть группы и числа. Если оба аргумента - группы, то они должны иметь одинаковое количество разрядов, совпадающее с количеством разрядов результата.

Если оба аргумента - числа, то автоматически осуществляется выравнивание числа разрядов их бинарных представлений путем увеличения числа разрядов в бинарном представлении меньшего аргумента.

Если один аргумент - число, а другой - группа, то число превращается в бинарное число (группу бинарных разрядов). Если число разрядов в этой группе меньше числа разрядов в другом аргументе, она автоматически расширяется. Если больше, процессор сообщений (Message Processor) пакета MAX+plusII выдает сообщение об ошибке.

Операторы сравнения

Результаты операции сравнения

Результаты операции сравнения:

логический нуль (GND), если условия сравнения не выполнены;

логическая единица (VCC), если условия сравнения выполнены.

Выделяют два типа операторов сравнения:

операторы логического сравнения (Logical comparator);

операторы арифметического сравнения (Arithmetic comparator).

Оператор логического сравнения

Аргументами оператора логического сравнения могут быть: 1-разрядные переменные и выводы модуля; группы переменных и группы выводов модуля;

числа (в бинарных числах не должен использоваться символ «x» неопределенного значения разряда).

При логическом сравнении осуществляется побитовое сравнение аргументов, потому они должны иметь одинаковое количество разрядов.

Если один аргумент число, а другой - группа, то число превращается в бинарное число (группу бинарных разрядов). Если количество разрядов в этой группе меньше чем в другом аргументе, то она автоматически расширяется. Если же больше, то процессор сообщений (Message Processor) пакета MAX+plusII выдает сообщение об ошибке.

Операторы арифметического сравнения

Операторы арифметического сравнения позволяют сравнивать числа и группы. Например, группы переменных и выводов модуля

При арифметическом сравнении группа интерпретируется как позитивное бинарное число без знака, количество разрядов которого отвечает количества разрядов в группе. Потому аргументы должны иметь одинаковое количество разрядов.

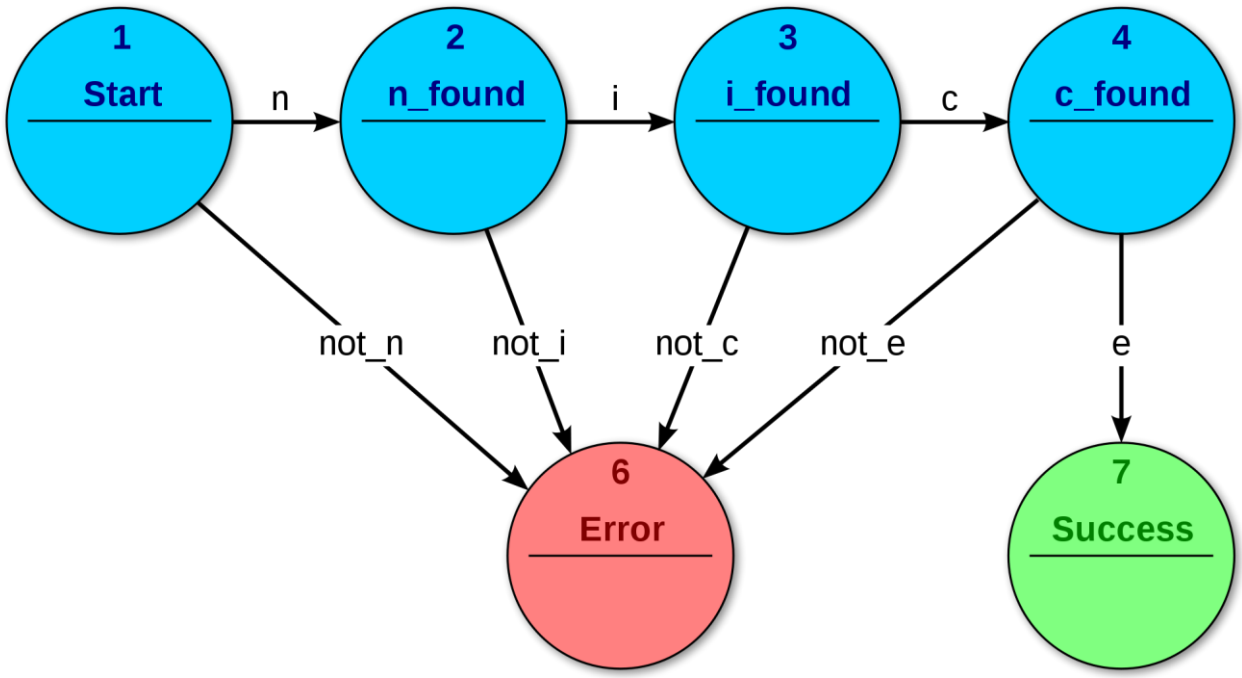
Если один аргумент число, а другой - группа, то число превращается в бинарное число (группу бинарных разрядов). Если количество разрядов в этой группе меньше чем в другом аргументе, то она автоматически расширяется. Если же больше, то процессор сообщений (Message Processor) пакета MAX+plusII выдает сообщение об ошибке.

Тематичний модуль 2.
Автомат з обмеженим числом станів.

Лекція 5. Елементи цифрового автомата.

Лекція 6. Опис синхронного цифрового автомата.

Лекція 7. Приклади реалізації автоматів



5 Элементы конечного автомата

План лекции:
1) примитивы буферов и триггеров;
2) структурная схема цифрового автомата.

5.1 Примитивы буферов и триггеров

Элементная база

Все многообразие цифровых систем автоматизации производства можно разделить на два больших класса: комбинационные схемы; схемы на триггерах.

Схема на триггерах это схема с ячейкой памяти, которая синхронизируется фронтом или уровнем тактового сигнала, синхронным триггером и триггером-защелкой соответственно.

Фирма ALTERA предлагает примитивы наряду с базовыми элементами цифровых схем - комбинационными вентилями (И - НЕ). Примитив - встроенное в пакет MAX+plus II и язык AHDL функциональное описание того или другого внутреннего ресурса микросхемы программируемой логики PLD фирмы ALTERA.

Примитивы буферов и примитивы триггеров

В языке AHDL определены два типа примитивов: примитивы буферов; примитивы триггеров.

Примитивы триггеров объединяются в модуле следующего уровня иерархии - регистры.

Вентили и примитивы буферов объединяются в модуле следующего уровня иерархии - комбинационные схемы.

Регистры и комбинационные схемы объединяются в модуле следующего уровня иерархии - цифровые автоматы, которые позволяют строить очень сложные модули.

Применение определенной таким образом элементной базы цифровых систем гибкой автоматизации производства является предметом данного тематического модуля.

Теоретические возможности вычислительных устройств или цифровых машин определяются отдельно.

Примитивы буферов

В языке AHDL используются следующие примитивы буферов:

CARRY - буфер цепного переноса;

CASCADE - буфер каскадного наращивания логической функции;

EXP - буфер логического расширителя;

GLOBAL - буфер глобальной цепи распространения управляющего сигнала;

LCELL - не гибкий буфер размещения логического элемента;

OPNDRN - буфер выхода с открытым коллектором;

SOFT - гибкий буфер размещения логического элемента ;

TRI - буфер выхода с Z состоянием.

Девять примитивов триггеров используются в языке AHDL.

Триггеры синхронные (Flipflop) и асинхронные (Latch).

Триггеры имеют выводы:

D, T, J, K, S, R - информационные входы;

CLK - вход тактового сигнала (активный перепад 0->1);

CLRn - вход асинхронного сброса триггера, активный уровень - логический нуль;

PRn - вход асинхронной установки триггера, активный уровень - логический нуль;

ENA - вход разрешения работы триггера, активный уровень - логическая единица.

5.2 Структурная схема цифрового автомата.

Цифровой автомат имеет следующие сигналы

X[N..1] - входные сигналы;

Y[к..1] - исходные сигналы;

Q[K..1] - разряды памяти, которые определяют состояние автомата;

D[n..1] - данные для записи в память;

M - память автомата (набор триггеров);

КС1 - входная комбинационная схема, которая обеспечивает формирование данных для записи в память;

КС2 - выходная комбинационная схема, формирующая выходные сигналы.

Синхронный цифровой автомат - это автомат, память, которого реализованная на синхронных триггерах. Он может переходить из состояния в состояние только в определенные моменты времени, когда поступает фронт (или спад) тактового сигнала, который синхронизирует триггеры в его блоке памяти.

В зависимости от способа формирования исходных сигналов выделяют два класса автоматов.

Автомат Мура - автомат, выходные сигналы в котором зависят лишь от текущего состояния автомата.

Автомат Мили - автомат, выходные сигналы в котором зависят от текущего состояния автомата и от текущих входных сигналов.

Число триггеров (Nff), которые используются для реализации модуля памяти автомата, определяется числом состояний автомата (Ns) и способом их кодировки.

Бинарная кодировка (Binary coding). При этом

Nff=]log2Ns[,

где].[операция округления к ближайшему большему целому.

Кодировка по принципу: одно состояние - один триггер (One Hot State). При этом

Nff= Ns.

Использование языка AHDL для описания алгоритма работы автомата позволяет автоматизировать процедуру синтеза, включая выбор числа разрядов памяти и кодировки состояний автомата.

В языке AHDL цифровой автомат - это переменная.

Для ее задания в разделе переменных (Variable Section) используется следующая конструкция.

Раздел описания переменных

__machine_name : MACHINE OF BITS (__state_bit, __state_bit)

WITH STATES (

__state_name = __state_value,

__state_name = __state_value,

__state_name = __state_value);

Раздел в общем случае определяет:

имя переменной - символическое имя автомата (machine_name)

число разрядов памяти автомата и символическое имя каждого из разрядов (OF BITS (_state_bit, _state_bit));

символические имена состояний автомата и их коды (WITH STATE (__state_name = __state_value, state_name = __state_value)).

Указывать число разрядов памяти и их символические имена, а также коды состояний автомата необязательно. Потому в простом случае для задания цифрового автомата используется такая конструкция:__machine_name : MACHINE

WITH STATES (__state_name,__state_name,__state_name);

В этом случае компилятор, оптимизируя комбинационные схемы КС1 и КС2, самостоятельно выберет как разрядность блока памяти, так и коды состояний автомата.

В рамках языка AHDL цифровой автомат и переменная которая его обозначает, имеет три входа управления: CLK - вход тактового сигнала (активный - фронт); Reset - вход асинхронного сброса автомата (активный уровень - логическая единица); ENA - вход разрешения работы автомата (активный уровень - логическая единица).

Обязательное использование только входа CLK. Если входы Reset и ENA не используются, то на них автоматически подаются логические уровни, которые не препятствуют нормальной работе автомата.

6 Описание цифрового автомата

План лекции:
1) неформальное описание;
2) формальное описание.

6.1 Неформальное описание.

Исходный алгоритм работы автомата может быть задан в виде схемы или словесного описания.

Для того, чтобы описать автомат на языке AHDL, необходимо формализовать алгоритм его работы, то есть представить его либо графом переходов, либо таблицей переходов и выходов.

Как пример рассмотрим словесное описание алгоритма работы устройства, имеющего входы: START, ABAP, CLK и выходы: WORK END_WORK.

В исходном состоянии(инициализация):

- на выходах WORK END_WORK - логический нуль;
- при появлении на входе START логической единицы устройство переходит в рабочий режим, иначе остается в исходном состоянии.

В рабочем режиме

- на выходе WORK- логическая единица
  на выходе END_WORK - логический нуль;
- при появлении на входе START логического нуля устройство переходит режим ожидания, иначе остается в рабочем режиме.

В режиме ожидания

- на выходах WORK END_WORK - логическая единица;
- при появлении на входе START логической единицы устройство переходит в режим возобновления работы, иначе переходит в режим окончания работы.

В режиме возобновления работы

- на выходах WORK END_WORK - логический нуль;

при появлении на входе START логического нуля устройство переходит в режим ожидания, иначе переходит в рабочий режим.

В режиме окончания работы

- на выходе WORK - логический нуль
  на выходе END_WORK - логическая единица;
- независимо от логического уровня сигнала на входе START устройство переходит в исходное состояние.

При появлении на входе ABAP логической единицы устройство из любого режима асинхронно переходит в исходное состояние.

6.2 Формальное описание.

Граф переходов в форме автомата Мура

Узел содержит и состояние , значения выходных сигналов

Граф переходов состоит из узлов

Граф переходов состоит из узлов, которые помечают состояния автомата, и ветвей, которые отображают синхронные переходы между состояниями. В автомате Мура:

для каждого состояния указан набор выходных сигналов (в данном примере - это сигналы WORK и END_WORK);

у каждой ветви указаны значения входных сигналов (в данном примере сигнала START), которые вызывают соответствующий переход между состояниями.

Не зависимые от значений входных сигналов переходы - это безусловные переходы (например, переход из состояния Ending в состояние INIT).

Отметим, что описанный автомат асинхронно переходит из любого состояния в состояние INIT при появлении на входе ABAP логической единицы.

Граф переходов автомата Мили

У каждой ветви автомата Мили указанны

значение входных сигналов, которые вызывают соответствующий переход (в данном примере - сигнала START);

значения выходных сигналов (в данном примере - сигналов WORK и END_WORK).

7 Примеры реализации автоматов

План лекции:
1) применения таблицы истинности, операторы IF THEN и Case ;
2) Экспорт и импорт состояний автомата.

7.1 Применения таблицы истинности, операторы IF THEN и Case

Применение таблицы истинности для описания автомата Мура

```
SUBDESIGN Moore1 ( Start, ABAP, CLK     : INPUT;  Work, End_work      : OUTPUT; )
VARIABLE  FSM :
 MACHINE WITH STATES (INIT,  WORKING,  WAITING,  RESUMING, ENDING);
BEGIN    FSM.clk= CLK;     FSM.reset = ABAP;
TABLE
-- текущее      входной         выходные                    следующее
-- состояние    сигнал  =>      сигналы                     состояние
FSM,            START   =>      WORK,       END_WORK,       FSM;
INIT,           0       =>      0,          0,              INIT;
INIT,           1       =>      0,          0,              WORKING;
WORKING,  0     =>      1,          0,              WAITING;
WORKING,  1     =>      1,          0,              WORKING;
WAITING,  0     =>      1,          1,              ENDING;
WAITING,  1     =>      1,          1,              RESUMING;
RESUMING, 0     =>      0,          0,              WAITING;
RESUMING,1      =>      0,          0,              WORKING;
ENDING,         B"x"    =>      0,          1       INIT;
END  TABLE;
END;
```

Применение таблицы истинности для описания автомата Мили

```
SUBDESIGN Mealy1 ( Start, ABAP, clk   : INPUT; Work, End_work    : OUTPUT; )
VARIABLE   FSM :
MACHINE    WITH    STATES    (INIT,  WORKING,  WAITING, RESUMING,ENDING);
BEGIN FSM.clk = CLK; FSM.reset = ABAP;
TABLE
-- текущее      входной         выходные                    следующее
```

```
-- состояние      сигнал    =>        сигналы                  состояние
FSM,              START     => WORK, END_WORK,                   FSM;
INIT,               0       =>       0,          0,             INIT;
INIT,               1       =>       1,          0,           WORKING;
WORKING,            0       =>       1,          1,           WAITING;
WORKING,            1       =>       1,          0,           WORKING;
WAITING,            0       =>       0,          1,           ENDING;
WAITING,            1       =>       0,          0,           RESUMING;
RESUMING,           0       =>       1,          1,           WAITING;
RESUMING,           1       =>       1,          0,           WORKING;
ENDING,           B"x"      =>       0,          0,             INIT;
END TABLE;
END;
```

Применение для описания автомата МИЛИ операторов CASE и IF THEN

```
SUBDESIGN  Mealy2 (Start, ABAP, clk  : INPUT; Work, End_work : OUTPUT;)
VARIABLE  FSM :
MACHINE WITH STATES(INIT, WORKING, WAITING, RESUMING, ENDING);
BEGINFSM.clk= CLK; FSM.reset = ABAP;
CASE FSM IS
 WHEN INIT =>
    IF  START==0  THEN FSM = INIT; ELSE FSM = WORKING; WORK=VCC; END IF;
WHEN WORKING =>
    IF START==0 THEN FSM = WAITING; WORK=VCC; END_WORK=VCC;
    ELSE  FSM = WORKING; WORK=VCC;END IF;
WHEN WAITING =>
     IF START==0  THEN  FSM = ENDING; END_WORK=VCC;  ELSE FSM = RESUMING;  END IF;
 WHEN RESUMING=>
     IF START==0THEN FSM = WAITING;WORK=VCC;END_WORK=VCC;
     ELSE FSM = WORKING;WORK=VCC;END IF;
 WHEN ENDING =>
    FSM = INIT;
END CASE;
END;
```

Определение значений выходных сигналов при объявлении автомата Мура

```
SUBDESIGN Moore3 (Start, ABAP, Clk  : INPUT; Work, End_work  : OUTPUT; )
VARIABLE  FSM :
```

```
MACHINE OF BITS (Work, End_work)  WITH STATES
(INIT =B"00“,WORKING= B"10", WAITING = B"11", RESUMING=B"00",ENDING= B"01");
BEGIN   FSM. clk  = CLK; FSM.reset = ABAP;
CASE FSM  IS
    WHEN INIT =>
IF START==0THEN FSM = INIT;ELSE FSM = WORKING;END IF;
    WHEN WORKING=>
IF START==0THEN FSM = WAITING;ELSE FSM = WORKING;END IF;
    WHEN WAITING=>
IF START==0THEN FSM = ENDING;ELSE FSM = RESUMING;END IF;
    WHEN RESUMING=>
IF START==0 THEN FSM = WAITING  ELSE FSM = WORKING;  END IF;
    WHEN ENDING => FSM = INIT;
END CASE;  END;
```

Три варианта отчета компілятора.

```
FSM:MACHINE OF BITS( FSM~5,FSM~4, FSM~3,FSM~2, FSM~1 ) WITH STATES
(INIT          = B"00000",
WORKING  = B"11000",
WAITING    = B"10100",
RESUMING = B"10010",
ENDING      = B"10001");
FSM : MACHINE OF BITS (  FSM~3, FSM~2, FSM~1  )  WITH STATES
(INIT           = B”000”,
WORKING   = B”011”,
WAITING     = B”010”,
RESUMING   = B”100”,
ENDING      = B”001”);
FSM: MACHINE OF BITS ( FSM~I, Work~, End_work~ ) WITH STATES
(INIT           = B"000",
WORKING   = B"010",
WAITING     = B"011",
RESUMING = B”100",
ENDING      = B"001);
```

7.2 Экспорт и импорт состояний автомата

Экспорт состояний автомата.

```
SUBDESIGN   definit_   ( CLK, Reset     : INPUT = GND; START, STOP, _END  : INPUT = GND; FSM_OUT : MACHINE OUTPUT;)
VARIABLE    AA   :  MACHINE  WITH  STATES (Idle, Busy, Wait);
BEGIN
```

```
CASE    AA    IS
WHEN  Idle => IF Start THEN  AA = Busy; ELSE    AA = Idle; END IF;
WHEN  Busy => IF Stop THEN  AA = Wait; ELSE    AA = Busy; END IF;
WHEN  Wait =>
IF Start THEN  AA = Busy;
ELSIF _END THEN AA =Idle; ELSE    AA = Wait;
END IF;
END CASE;
AA.(CLK, Reset) = (CLK, Reset); FSM_OUT = AA;
END;
```

Файл definit_.inc:

```
FUNCTION definit_(CLK, Reset, Start, Stop, _End)
RETURNS (MACHINE  FSM_OUT)
```

Импорт состояний автомата

```
SUBDESIGN       USE_  (FSM_in:  MACHINE  INPUT;  Work,  Waiting:
OUTPUT;)
BEGIN
Work=(FSM_in == Busy); Waiting=(FSM_in == Idle)OR (FSM_in   == Wait);
END;
```

Файл use_.inc:

```
FUNCTION   USE_      (MACHINE  FSM_IN)
RETURNS (Work, Waiting).
 SUBDESIGN     top_level
(CLK, Reset, Start, Stop, _End: INPUT; Work, Waiting; OUTPUT;)
VARIABLE    FSM_       :     MACHINE;
BEGIN
FSM_ =
 definit (.CLK = CLK, .Reset = Reset, .Start = Start, .Stop = Stop, ._End =
_End);
(Work, Waiting) = USE_ (.FSM_in = FSM_);
END;
```

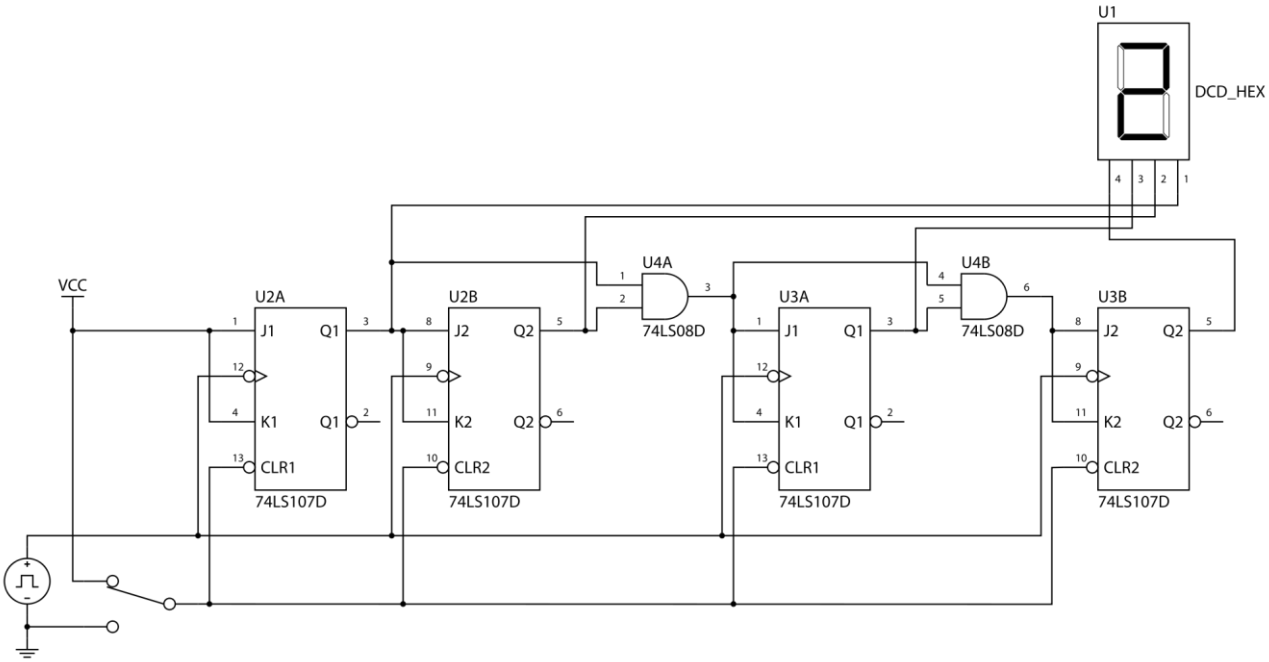
Тематичний модуль 3.
Текстовий опис складних схем.

До складу модуля входять наступні лекції.

Лекція 8. Способи застосування примітивів буферів і тригерів.

Лекція 9. Способи застосування прототипів модулів.

Лекція 10. Повна структура текстового опису



8 Способы применения примитивов буферов и триггеров

Повестка дня:

1) Непосредственное обращение

2) Присвоение примитиву символического имени и обращение к нему как к переменной.

8.1 Непосредственное обращение

Для использования в текстовом описании модуля примитива необходимо обратиться к встроенному в пакет функциональному описанию данного примитива и сопоставить его выводам числа, константы, переменные или выводы модуля.

В языке AHDL определены два способа обращения к примитиву:

- непосредственное обращение (In_line Reference);
- присвоение примитиву символического имени, т. е. объявление его переменной, и обращение к нему как к переменной.

Непосредственное обращение к примитиву осуществляется следующим образом:

указывается вывод (либо внутренняя переменная) модуля, на который передается сигнал с выхода примитива;

далее ставится знак равенства и имя примитива;

за именем примитива в круглых скобках, через запятую перечисляются передаваемые значения: числа, константы, переменные или выводы модуля, сопоставляемые входам примитива;

за круглыми скобками ставится точка с запятой.

Сопоставление входов примитива с передаваемыми значениями может осуществляться:

- позиционно;
- по именам.

При позиционном сопоставлении порядок перечисления передаваемых значений должен соответствовать порядку перечисления входов, использованному в описании прототипа примитива:

Вывод модуля (внутренняя переменная)=

Имя примитива (передаваемое значение, передаваемое значение, ...);

При сопоставлении передаваемых значений и входов примитива по именам в списке передаваемых значений через запятую перечисляются пары: вход, передаваемое значение. Формат записи:

Выход модуля (внутренняя переменная)=

Имя примитива (.имя вывода = передаваемое значение,

.имя вывода = передаваемое значение,...);

Пары могут быть расположены в произвольном порядке, т. е. независимо от того, в какой последовательности перечислены входы в описании прототипа примитива. Неиспользованные входы примитива в списке передаваемых значений не указываются.

8.2 Обращение к прототипу как к переменной

При реализации этого способа примитив, прежде всего, следует объявить переменной. Для этого в разделе переменных (Variable Section) символическому имени или группе символических имен сопоставляется Примитив. Объявленная таким образом переменная, а также каждая переменная из объявленной группы переменных, будет иметь тот же набор выводов, что и примитив.

Обращение к конкретному выводу примитива осуществляется путем указания имени переменной, разделяющей точки и имени вывода примитива.

9 Способы применения прототипов модулей

Повестка дня:

1) Непосредственное обращение

2) Присвоение прототипу символического имени, т. е. объявление его переменной, и обращение к нему как к переменной.

9.1 Непосредственное обращение

Язык AHDL позволяет при описании модуля использовать в качестве его компонентов созданные ранее модули. Для этого текстовое описание модуля верхнего уровня иерархии должно содержать описания прототипов этих модулей. В языке AHDL определены два способа обращения к прототипу модуля: непосредственное обращение (In_Line Reference) и присвоение прототипу символического имени, т. е. объявление его переменной, и обращение к нему как к переменной. Прототип задается с помощью оператора Function Prototype Statement, который может быть расположен либо непосредственно в текстовом описании, либо в файле включения (Include File), содержимое которого подсоединяется к текстовому описанию на этапе компиляции.

Файл включения (Include File) с описанием прототипа модуля создается с помощью команды Create Default Include File (меню File), выполняемой в окне текстового редактора пакета MAX+PLUSII, содержащем описание модуля.

Отметим, что указанные способы обращения к прототипу совпадают с описанными ранее способами обращения к примитиву. В языке AHDL определено два типа модулей: параметризированные (Parameterized) и не параметризированные (Unparameterized).

При этом, как те, так и другие могут быть созданы либо самим разработчиком, либо фирмой Altera.

9.2 Обращение к прототипу как к переменной

В качестве примера можно рассмотреть модуль верхнего уровня, в текстовом описании которого, в разделе переменных объявлены три переменные — символические имена модулей, используемых в данном текстовом описании как компоненты.

10 Полная структура текстового описания

Повестка дня:

1) структура текстового описания на языке AHDL;

2) компоненты структуры текстового описания.

10.1 Структура текстового описания на языке AHDL

Текстовое описание на языке AHDL должно иметь определенную структуру:

- Title Statement ;
- Include Statement;
- Constant Statement;
- Define Statement;
- Parameters Statement;
- Function Prototype Statement;
- Options Statement;
- Assert Statement;

Subdesign Section;

Variable Section

- Instance Declaration
- Node Declaration
- Register Declaration
- State Machine Declaration
- Machine Alias Declaration
- If Generate Statement

Logic Section;

- Defaults Statement;
- Assert Statement;
- Boolean Equations;
- Boolean Control Equations;
- Case Statement;
- For Generate Statement;

- If Then Statement;
- If Generate Statement;
- In_Line Logic Function Reference;
- Truth Table Statement;

Не все перечисленные операторы и разделы являются обязательными. Так, в простейшем случае, текстовое описание может содержать только разделы Subdesign Section и Logic Section.

10.2 Компоненты структуры текстового описания

Оператор заголовка (Title Statement) позволяет задать название для генерируемого компилятором файла отчета (Report File). Оператор начинается с ключевого слова TITLE, за которым следует текстовая строка, заключенная в двойные кавычки («). В конце оператора заголовка ставится точка с запятой. Текстовая строка может содержать до 255 символов включительно. Она не должна содержать символов конца строки или конца страницы. Оператор начинается с ключевого слова TITLE, за которым следует текстовая строка, заключенная в двойные кавычки («). В конце оператора заголовка ставится точка с запятой. Текстовая строка может содержать до 255 символов включительно. Она не должна содержать символов конца строки или конца страницы.

Оператор включения (Include Statement) позволяет включить содержимое файла, указанного в операторе, в текущее текстовое описание. Оператор начинается с ключевого слова INCLUDE, за которым в двойных кавычках указывается имя файла включения (Include file). Далее ставится точка с запятой. Если явно не задано расширение файла включения, то компилятор ищет файл, имеющий заданное имя и расширение .INC. Имя файла, указанное в операторе включения, не должно содержать пути к файлу. В файле с текстовым описанием данный оператор может использоваться неограниченное число раз. Последовательность поиска файлов включения при компиляции модуля: в директории проекта; в библиотеках пользователя, заданных командой User Libraries (меню Options); в директориях \maxplus2\max2lib\mega_lpm и \maxplus2\max2inc, созданных в процессе инсталляции пакета. Файл включения может содержать следующие операторы: Function Prototype Statement — оператор описания прототипа;

Define Statement — оператор обозначения; Parameters Statement — оператор объявления параметров; Constant Statement — оператор задания константы.

Оператор задания константы (Constant Statement) позволяет присвоить символическому имени неизменяемое значение либо явно заданное числом, либо являющееся результатом выполнения арифметического выражения. Оператор начинается с ключевого слова CONSTANT, за которым следует символическое имя, символ равно (=), число или арифметическое выражение. Далее ставится точка с запятой (;). Имя константы должно быть уникальным и не должно содержать пробелов. Для улучшения читаемости имени следует применять символ подчеркивания (_). Ссылка на константу допускается только после ее задания. При задании константы могут использоваться заданные ранее константы. Циклическое задание констант недопустимо. В файле с текстовым описанием данный оператор может использоваться неограниченное число раз. Арифметическое выражение, задающее константу, оценивается компилятором и заменяется числом на этапе проверки синтаксиса. Поэтому применение арифметических выражений при задании констант не приводит к использованию дополнительных логических ресурсов микросхемы.

Оператор обозначения (Define Statement) позволяет обозначить арифметическое выражение символическим именем. Оператор начинается с ключевого слова DEFINE, за которым следует символическое имя обозначаемого арифметического выражения со списком аргументов, заключенным в скобки. За списком аргументов ставится знак «равно», далее — само арифметическое выражение и точка с запятой. Список аргументов может содержать один или более аргументов. Аргументы в списке отделяются друг от друга запятой. Символическое имя должно быть уникальным и не должно содержать пробелов. Для улучшения читаемости имени следует применять символ подчеркивания. Ссылка на символическое имя арифметического выражения допустима только после его задания в операторе обозначения. Обозначаемое арифметическое выражение может содержать символическое имя обозначенного ранее арифметического выражения. В файле с текстовым описанием данный оператор обозначения может использоваться неограниченное число раз. Обозначенное арифметическое выражение оценивается компилятором и заменяется числом на этапе проверки синтаксиса, поэтому его применение не приводит к использованию дополнительных логических ресурсов микросхемы.

Параметр в рамках языка AHDL — символическое имя строки символов, заключенной в двойные кавычки, или числа, заданного либо явно, либо являющегося результатом выполнения арифметического выражения. Оператор объявления параметров (Parameters Statement) позволяет объявить параметры, управляющие реализацией параметризированных модулей. Область действия параметра — текстовое описание, в котором он был объявлен. В файле с текстовым описанием оператор объявления параметров может использоваться неограниченное число раз. Компилятор присваивает параметру значение, заданное: при объявлении параметризированного модуля в разделе Variable Section или при непосредственном использовании (In_Line Reference) параметризированного модуля; при объявлении в разделе Variable Section модуля, в состав которого входит параметризированный модуль; как глобальное значение параметра (команда Global Project Parameters меню Assign). Глобальное значение параметра хранится в файле назначений и конфигурации (Assignment & Configuration File) проекта; как исходное (Defaults) значение параметра.

Оператор описания прототипа (Function Prototype Statement) позволяет описать интерфейс (входы, выходы, передаваемые параметры) модулей или примитивов, которые будут использованы в данном текстовом описании. Оператор начинается с ключевого слова FUNCTION, за которым следует имя модуля, а за ним — заключенный в скобки список входов модуля. При описании прототипа параметризированного модуля далее следует ключевое слово WITH и заключенный в скобки список передаваемых в модуль параметров. Затем указывается ключевое слово RETURNS, и далее — заключенный в скобки список выходов модуля. Оператор оканчивается точкой с запятой. Элементы списков (входов, передаваемых параметров, выходов) отделяются друг от друга запятыми. В файле с текстовым описанием данный оператор может использоваться неограниченное число раз.

Оператор задания опции (Options Statement) позволяет определить в группе элемент с меньшим индексом (BITO) как: LSB — младший разряд (Least Significant Bit). При этом индексы в группах должны указываться в убывающей слева направо последовательности; MSB — старший разряд (Most Significant Bit). При этом индексы в группах должны указываться в возрастающей слева направо последовательности; ANY. При этом порядок перечисления индексов в группе произвольный. Нарушение порядка перечисления индексов в группах ведет к появлению предупреждения на этапе синтаксического анализа

текстового описания. Оператор задания опции начинается с ключевого слова OPTIONS, за которым следует ключевое слово BITO, символ «равно» (=) и одно и: трех ключевых слов LSB, MSB, ANY. Далее ставится точка с запятой . Если опция не задана, то по умолчанию предполагается, что BITO = LSB. Значение опции, заданное в файле верхнего уровня иерархии описаний проекта, распространяется и на файлы с описанием модулей нижних уровней иерархии, если в них явно не задано другое значение опции. В файле с текстовым описанием данный оператор может использоваться только один раз.

Оператор контроля (Assert Statement) позволяет проконтролировать истинность арифметического выражения, а для случая, когда оно ложно, указать текст сообщения и определить реакцию компилятора. Оператор начинается с ключевого слова ASSERT, за которым следует арифметическое выражение. Если значение выражения False (ложно), то контролируемое условие считается невыполненным, и процессор сообщений (Message Processor) отображает заключенное в двойные кавычки сообщение, следующее за ключевым словом REPORT. Сообщение может содержать символ процент (%), который заменяется значением переменной, указываемой после закрывающих двойных кавычек. Если используется несколько символов процента (%), то переменные перечисляются через запятую в том порядке, в котором должны подставляться их значения. Необязательное ключевое слово SEVERITY позволяет задать реакцию компилятора («уровень строгости» сообщения): ERROR — ошибка, WARNING — предупреждение, INFO — информация. При отсутствии ключевого слова SEVERITY сообщение, по умолчанию, имеет «уровень строгости» — ERROR. Если ключевое слово REPORT и соответствующее сообщение не были указаны в операторе, то при невыполненных условиях контроля процессор сообщений (Message Processor) отображает следующую строчку:
<"уровень строгости" >Line<номер строки>, File<имя файла>Assertion failed
Оператор контроля оканчивается символом точка с запятой . Допустимо применение оператора в разделе описания логики (Logic Section). В файле с текстовым описанием данный оператор может использоваться неограниченное число раз.

Раздел переменных (Variable Section) позволяет задавать внутренние переменные модуля, предназначенные для использования в разделе описания логики (Logic Section). Переменная — это символическое имя: линии связи; линии связи с тремя состояниями; модуля, используемого в текстовом

описании в качестве компонента (примитива, конечного автомата). Раздел начинается с ключевого слова VARIABLE. Далее указывается: символическое имя переменной, символ двоеточие, тип переменной, точка с запятой. Имена однотипных переменных могут быть перечислены через запятую. Допустимые типы переменных: NODE — линия связи; TRI_STATE_NODE — линия связи с тремя состояниями; модуль более низкого уровня иерархии; примитив; конечный автомат; псевдоним конечного автомата. Раздел переменных может также содержать оператор IF GENERATE, позволяющий, в зависимости от значения, оцениваемого в операторе арифметического выражения, управлять заданием переменных. В файле с текстовым описанием данный раздел может использоваться только один раз.