Технология разработки алгоритмически сложных цифровых систем с помощью автоматического синтеза микропрограммных автоматов

**Михаил Долинский,
Игорь Коршунов,
Алексей Толкачев,
Игорь Ермолаев,
Вячеслав Литвинов**

Введение

Возможности современных ПЛИС и СБИС существенно расширили круг устройств, которые могут быть изготовлены на их базе. Сегодня буквально на одном кристалле разрабатываются алгоритмически чрезвычайно сложные устройства, например, реализующие шифрование и дешифрование, компрессию и декомпрессию информации, либо разнообразную обработку потоковой аудио- и видеоинформации. Используемые отечественными, а в большинстве случаев и зарубежными разработчиками средства проектирования уже неадекватны стоящим задачам, что приводит к существенным временным и стоимостным затратам на проектирование и верификацию. Одним из способов решения проблем является введение в систему на кристалле процессора, и, соответственно, программная реализация многих функций устройств. Ранее мы уже описывали разработанные нами средства [1–3], поддерживающие подобные подходы. Однако такие подходы в ряде случаев могут обладать существенными недостатками, такими, например, как:

- избыточные аппаратные затраты;
- существенное снижение системной тактовой частоты, а следовательно, и производительности.

Избыточные аппаратные затраты возникают от того, что при решении конкретной прикладной задачи могут использоваться далеко не все возможности стандартного процессорного ядра. С другой стороны, реальная прикладная задача может требовать специфических инструкций.

Снижение системной тактовой частоты связано с необходимостью обеспечения исполнения программ, когда для выполнения каждой инструкции выполняются циклы модификации счетчика инструкций, загрузки и кодирования кода инструкции, загрузки операндов, исполнения инструкции и выгрузки результата.

Вышеописанное вынуждает разработчиков к ручной разработке микропрограммных автоматов [7, 8].

В СНИЛ «Новые информационные технологии» (Гомельский государственный университет) разработаны метод и средства разработки алгоритмически сложных устройств, которые обеспечивают программную разработку и аппаратную реализацию алгоритмов, основанные на автоматической генерации схем микропрограммных автоматов с управляющими автоматами с жесткой логикой и последующим автоматическим конвертированием этих схем в синтезируемые VHDL-описания.

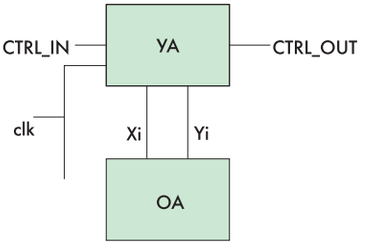
Для того чтобы представить наш подход, поясним (чрезвычайно конспективно), какой смысл обычно вкладывается в понятия «микропрограммный автомат» (МПА), «операционный автомат» (ОА) и «управляющий автомат» (УА). За более подробной информацией можно обратиться к специальной литературе [4–6].

В простейшем виде МПА делится на две взаимодействующие части — управляющий автомат (УА) и операционный автомат (ОА), которые, взаимодействуя друг с другом, реализуют определенный алгоритм (рис. 1).

Управляющий автомат представляет собой схему из двух комбинационных устройств и регистра состояний (рис. 2).

В зависимости от полученных микроусловий Xi, текущего состояния и управляющих сигналов CTRL_IN управляющий автомат принимает свое следующее состояние, в котором он выдает операционному автомату сигналы Yi — предписания исполнения микроопераций для текущего состояния управляющего автомата.

Операционный автомат — это совокупность комбинационных схем, исполняющих микрооперации, регистров/ОЗУ для хранения данных и внешних линий данных.



Xi — микроусловия;
Yi — микрооперации;
DATA_IN — входные данные;
DATA_OUT — выходные данные;
CTRL_IN — входные управляющие сигналы;
CTRL_OUT — выходные управляющие сигналы;
clk — тактовый импульс.

Рис. 1. Схема микропрограммного автомата

Алгоритм работы операционного автомата:

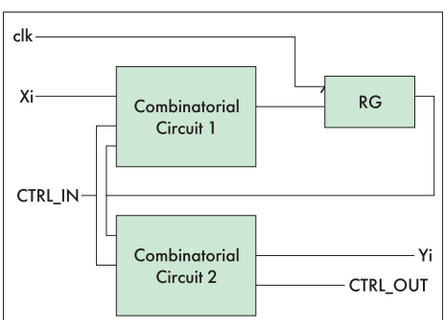
- в зависимости от полученных от управляющего автомата сигналов Yi, операционный автомат совершает микрооперации — различные действия над регистрами, ввод и вывод на внешние контакты;
- после совершения микроопераций в каждом состоянии операционный автомат формирует микроусловия Xi, по которым управляющий автомат принимает свое следующее состояние.

Работа схемы микропрограммного автомата происходит синхронно — по переднему фронту тактового импульса срабатывает управляющий автомат, а по заднему — операционный.

Продолжительность тактового импульса определяется таким образом, чтобы успела сработать самая длинная по времени микрооперация в операционном автомате или самый длинный переход в управляющем автомате плюс время, необходимое для установки новых условий над новым значением регистров в операционном автомате.

Мы предлагаем следующий цикл разработки алгоритмически сложных цифровых систем:

1. На специализированном языке MPDL (MicroProgram Description Language) разработчики (программисты) описывают алгоритм.
2. В системе WInter [2] MPDL-программы доводятся до состояния корректного функционирования с помощью итерационного применения мощных современных технологий отладки программ: редактирование — симуляция — визуализация — анализ — отладка.
3. По корректной отлаженной микропрограмме специальным генератором GenMPA генерируется схема микропрограммного автомата, реализующая отлаженный алгоритм. При этом в операционный автомат включаются только те инструкции, которые были реально задействованы в микропрограмме. Это обеспечивает сокращение аппаратных затрат по сравнению с использованием стандартного процессорного ядра. В то же время управляющий автомат реализуется в виде жесткой логики, что обеспечивает повышение производительности по сравнению с использованием стандартного процессорного ядра.



RG — регистр состояний;
Combinatorial Circuit 1
формирует следующее состояние УА;
Combinatorial Circuit 2
формирует Yi — разрешения микроопераций
Рис. 2. Схема управляющего автомата

4. В системах HLCCAD/IEESD [1] обеспечивается визуализация, симуляция, анализ и, в случае необходимости, оптимизация сгенерированной схемы, а также ее конвертирование к синтезируемому VHDL-описанию.

Заметим, что ручная разработка сопоставимых по аппаратным затратам и производительности операционного автомата и управляющего автомата с жесткой логикой существенно затратнее по срокам и стоимости.

Далее материал скомпонован следующим образом. Раздел 1 содержит описание созданного нами языка MPDL. Раздел 2 кратко знакомит с организацией системы отладки микропрограмм. В разделе 3 описываются генерируемые операционный и управляющие автоматы. Раздел 4 посвящен примерам апробации предлагаемого подхода к разработке цифровых устройств. Раздел 5 иллюстрирует возможности отладки микропрограмм с настройкой на исполнительные устройства. В разделе 6 представляются направления использования разработок в учебном процессе. В разделе 7 освещаются направления дальнейшего развития описанных метода и средств.

1. Язык описания микропрограммных автоматов

Язык описания микропрограммных автоматов MPDL (MicroProgram Description Language) разработан в СНИЛ «Новые информационные технологии» Гомельского госуниверситета (Белоруссия) с учетом возможности простоты и удобства написания и отладки программ, снабжен минимально необходимым набором инструкций, позволяющим эффективно описывать алгоритмы. Язык открыт для дальнейшего расширения, и набор инструкций может быть дополнен по необходимости.

Структура, принцип написания программ, мнемоники и формат инструкций намеренно делались подобными языку ассемблера наиболее распространенного семейства процессоров Intel 80×86.

Текст микропрограммы включает в себя последовательность обязательных и факультативных (отмеченных знаками []) разделов:

<Определение внешних контактов>
[<Определение элементов памяти>]
[<Автоматизированное тестирование>]
<Описание алгоритма>

В обязательном разделе <Определение внешних контактов> описываются входные, выходные и двунаправленные контакты разрабатываемого устройства с помощью соответствующих команд ContIn, ContOut, ContBi.

Например:

```
A    ContIn    1
B    ContOut   4
C    ContBi    8
```

Здесь A определен как входной контакт размерностью 1 бит, B — выходной 4-битный контакт, C — двунаправленный 8-битный контакт.

Входные контакты можно использовать в инструкциях только в качестве операндов-источников, выходные — в качестве операндов-приемников и операндов-источников.

Раздел описания элементов внутренней памяти может содержать команды Reg и Flag для описания регистров и битов соответственно.

Например:

```
X    Reg    8
Y    Flag
```

Здесь X — 8-битный элемент памяти (регистр), Y — 1-битный элемент памяти (триггер).

Необязательный раздел [<Автоматизированное тестирование>] включает теневые инструкции установки начальных значений, тестового вызова исполнения микропрограммы и проверки значений на контактах и элементах памяти после исполнения алгоритма микропрограммы. Например:

```
NOP                ;$S  A=1
TCALL   Device
NOP                ;$T  B=2,X=3,Y=0
```

Здесь перед исполнением алгоритма Device с помощью теневой команды $S на входной контакт A устанавливается значение 1. Инструкция тестового вызова микропрограммы TCALL обеспечивает вызов исполнения алгоритма Device, а теневая команда $T проверяет корректность значений на выходном контакте B и внутренних элементах памяти X и Y после завершения микропрограммы. Количество таких последовательных тестовых вызовов неограничено, поэтому можно иметь и перманентно развивать и пополнять исчерпывающий набор тестов разрабатываемого алгоритма. Заметим, что это не единственный способ тестирования корректности разработанной микропрограммы. Другие способы, непосредственно не связанные с этим языком, были описаны ранее [2].

Раздел <Описание алгоритма> представляет традиционную последовательность инструкций для виртуального процессора микропрограммных автоматов, исполняющих загрузку и выгрузку данных, арифметические, логические и сдвиговые операции, операции условного и безусловного перехода, вызова подпрограмм и возврата из подпрограмм, явного указания параллелизма фрагментов алгоритмов.

Прежде чем описывать семантику и синтаксис инструкций, назовем способы указания операндов.

Операнды (приемники и источники) в командах могут иметь вид:

1. Имя объявленного регистра или контакта, например:

```
not   a              ; инвертировать a
```

2. Имя регистра или контакта с явным указанием номера бита, причем нумерация битов идет справа налево, то есть младший бит имеет номер 0 и находится справа, например:

```
not   a[0]           ; инвертировать младший бит a
```

3. Имя регистра или контакта с явным указанием начального бита и количества битов, участвующих в операции, например:

```
not   a[2]:4        ; в операции участвуют биты: 2, 3, 4, 5
```

Операнды-источники могут также иметь вид констант в одной из четырех систем счисления: двоичной, восьмеричной, десятичной, шестнадцатиричной. Тип системы счисления определяется буквой, которая следует за числом:

- B или b — двоичное;
- O или o — восьмеричное;
- D или d — десятичное;
- H или h — шестнадцатеричное.

Суффикс десятичного числа может быть опущен — все числа по умолчанию считаются десятичными. Если шестнадцатеричное число начинается с буквы, то перед ним ставится 0 (ноль).

Например:

```
AND     X,  11110000b
AND     X,  0F0h
AND     X,  360o
AND     X,  240d
AND     X,  240
```

Полный перечень инструкций, которые можно использовать при разработке микропрограмм, приведен в таблице.

Для большего удобства при разработке микропрограмм реализовано два способа записи операндов микроинструкций: полный и сокращенный. При полном способе записи явно указываются все операнды.

Например:

```
SUB     A,  B,  C       ; C = A – B, A и B не изменяются;
ADD     A,  B,  A       ; A = A + B, B не изменяется;
NEG     A,  B           ; B = –A, A не изменяется;
NEG     A,  A           ; A = –A,
```

Заметим, что при полном способе записи приемником всегда является последний операнд.

В случае, если один из операндов-источников является одновременно и приемником, может быть применен сокращенный формат, например:

```
ADD     A,  B       ; A = A + B ,   B не изменяется;
NEG     A           ; A = –A,
```

Заметим, что при сокращенном способе записи команд первый операнд является и источником, и приемником. Кроме того, в инструкции MOV приемник — всегда первый операнд:

```
MOV     A,  B       ; A = B
```

Языком МПА допускается использование меток.

Ниже приведен пример описания с помощью языка МПА алгоритма функционирования стандартного устройства средней степени интеграции «приоритетный шифратор с 8 линий на 3».

Пример:

```
;  Описание контактов и регистров

X    contIn      8                  ;  объявление контактов
Y    contOut     3
G    contOut     1
Rx   reg         8                  ;  объявление регистров
Ry   reg         3

;  раздел тестов

     nop    ;$S x=011000000b        ;  Входное воздействие

     tcall  prcd                    ;  Вызов микропрограммы

     nop    ;$T Y=6                 ;  сравнение выхода с эталоном

prcd:                               ;  начало микропрограммы
     mov  rx,r
     clr  ry
     clr  g
ml:  jnz  rx[0],g1
     shr  rx
     inc  ry
     jnz  ry,ml
     jmp  e
g1:  set  g
e:   mov  y,ry
     end                            ;  окончание микропрограммы
```

Далее приводится полное описание системы инструкций МПА:

Таблица. Система инструкций виртуального процессора МПА

Инструкции установки и пересылки	SET, MOV, CLR, NOP
Арифметические инструкции	ADD, ADC, SUB, SBB, INC, DEC, NEG
Логические инструкции	AND, OR, XOR, NOT
Инструкции сдвигов	SHL, SHR, SAL, SAR, ROL, ROR, RCL, RCR
Инструкции двойных сдвигов	SHLD, SHRD, SALD, SARD, ROLD, RORD, RCLD, RCRD
Инструкции условных переходов	JZ,JNZ,JEQ,JNEQ,JODD,JNODD, JL, JG, JLE, JGE, JA, JB, JAE, JBE
Инструкция безусловного перехода	JMP
Инструкция вызова подпрограммы	CALL

Инструкции установки и пересылки

Мнемоника	Формат	Пояснения
MOV	mov <dst>, <src>	<dst>=<src> Команда также употребляется для загрузки и выгрузки на внешние контакты
CLR	clr <dst>	<dst>=00..0 (по количеству битов в <dst>)
SET	set <dst>	<dst>=11..1 (по количеству битов в <dst>)
NOP	nop	нет операции

Арифметические инструкции

Мнемоника	Формат	Пояснения
ADD	add <src1>,<src2>, <dst> add <dst>, <src>	<dst>=<src1>+<src2> <dst>=<dst> +<src1>
ADC	adc <src1>,<src2>, <dst> adc <dst>, <src> adc <dst>	<dst>=<src1>+<src2>+CF <dst>=<dst> +<src1>+CF <dst>=<dst>+CF
SUB	sub <src1>, <src2>, <dst> sub <dst>, <src>	<dst>=<src1>-<src2> <dst>=<dst> -<src1>
SBB	sbb <src1>, <src2>, <dst> sbb <dst>, <src> sbb <src>	<dst>=<src1>-<src2>-CF <dst>=<dst> -<src1>-CF <dst>=<dst>-CF
INC	inc <src>,<dst> inc <dst>	<dst>=<src>+1 <dst>=<dst>+1
DEC	dec <src>,<dst> dec <dst>	<dst>=<src>-1 <dst>=<dst>-1
NEG	neg <src>, <dst> neg <dst>	<dst>= -<src> <dst>= -<dst>

Все операнды должны быть одинакового размера.

Возможно указание числовой константы в поле источника.

В инструкциях ADD и INC если был перенос, то флаг CF устанавливается, иначе сбрасывается.

Логические инструкции

Мнемоника	Формат	Пояснения
AND	and <src1>,<src2>, <dst> and <dst>, <src>	<dst>=<src1> & <src2> <dst>=<dst> & <src1> Поразрядное «И»
OR	or <src1>,<src2>, <dst> or <dst>, <src>	<dst>=<src1> | <src2> <dst>=<dst> | <src1> Поразрядное «ИЛИ»
XOR	xor <src1>, <src2>, <dst> xor <dst>, <src>	<dst>=<src1> ^ <src2> <dst>=<dst> ^ <src1> Поразрядное «Исключающее ИЛИ» (сложение по модулю 2)
NOT	not <src>,<dst> not <dst>	<dst>=~<src> <dst>=~<dst> Поразрядная инверсия

Все операнды должны быть одинакового размера.

Возможно указание числовой константы в поле источника.

Инструкции сдвигов

Мнемоника	Формат	Пояснения
SHL	shl <src>, <count>, <dst> shl <dst>, <count> shl <dst>	ЛОГИЧЕСКИЙ сдвиг ВЛЕВО на <count> позиций
SHR	shr <src>, <count>, <dst> shr <dst>, <count> shr <dst>	ЛОГИЧЕСКИЙ сдвиг ВПРАВО на <count> позиций
SAL	sal <src>, <count>, <dst> sal <dst>, <count> sal <dst>	АРИФМЕТИЧЕСКИЙ сдвиг ВЛЕВО на <count> позиций
SAR	sar <src>, <count>, <dst> sar <dst>, <count> sar <dst>	АРИФМЕТИЧЕСКИЙ сдвиг ВПРАВО на <count> позиций
ROL	rol <src>, <count>, <dst> rol <dst>, <count> rol <dst>	ЦИКЛИЧЕСКИЙ сдвиг ВЛЕВО на <count> позиций
ROR	ror <src>, <count>, <dst> ror <dst>, <count> ror <dst>	ЦИКЛИЧЕСКИЙ сдвиг ВПРАВО на <count> позиций
RCL	rcl <src>, <count>, <dst> rcl <dst>, <count> rcl <dst>	ЦИКЛИЧЕСКИЙ через CF сдвиг ВЛЕВО на <count> позиций
RCR	rcr <src>, <count>, <dst> rcr <dst>, <count> rcr <dst>	ЦИКЛИЧЕСКИЙ через CF сдвиг ВПРАВО на <count> позиций

При одноопрерандной форме записи операнд сдвигается на один разряд. При двухоперандной — первый операнд сдвигается на количество разрядов, указанное во втором операнде.

Инструкции двойных сдвигов

Мнемоника	Формат	Пояснения
SHLD	shld <dst1>,<dst2>, <count> shld <dst1>, <dst2>	ЛОГИЧЕСКИЙ ДВОЙНОЙ сдвиг ВЛЕВО на <count> позиций
SHRD	shrd <dst1>,<dst2>, <count> shrd <dst1>, <dst2>	ЛОГИЧЕСКИЙ ДВОЙНОЙ сдвиг ВПРАВО на <count> позиций
SALD	sald <dst1>,<dst2>, <count> sald <dst1>, <dst2>	АРИФМЕТИЧЕСКИЙ ДВОЙНОЙ сдвиг ВЛЕВО на <count> позиций
SARD	sard <dst1>,<dst2>, <count> sard <dst1>, <dst2>	АРИФМЕТИЧЕСКИЙ ДВОЙНОЙ сдвиг ВПРАВО на <count> позиций
ROLD	rold <dst1>,<dst2>, <count> rold <dst1>, <dst2>	ЦИКЛИЧЕСКИЙ ДВОЙНОЙ сдвиг ВЛЕВО на <count> позиций
RORD	rord <dst1>,<dst2>, <count> rord <dst1>, <dst2>	ЦИКЛИЧЕСКИЙ ДВОЙНОЙ сдвиг ВПРАВО на <count> позиций
RCLD	rcld <dst1>,<dst2>, <count> rcld <dst1>, <dst2>	ЦИКЛИЧЕСКИЙ через CF ДВОЙНОЙ сдвиг ВЛЕВО на <count> позиций
RCRD	rcrd <dst1>,<dst2>, <count> rcrd <dst1>, <dst2>	ЦИКЛИЧЕСКИЙ через CF ДВОЙНОЙ сдвиг ВПРАВО на <count> позиций

Первые два операнда являются сборным регистром, старшая часть которого образована первым операндом, а младшая — вторым.

Сдвиг выполняется на 1 разряд по умолчанию или на количество разрядов, указанное в третьем операнде.

Инструкции переходов

Мнемоника	Формат	Пояснения
JMP	jmp <label>	Безусловный переход на <label>
JC	jc <label>	Перейти на <label>, если установлен (равен 1) флаг переноса CF. Флаг переноса автоматически устанавливается при выполнении некоторых инструкций (inc, add, adc, сдвигов).
JNC	jnc <label>	Перейти на <label>, если сброшен (равен 0) флаг переноса CF.
JZ	jz <src>, <label>	Перейти на<label>, если <src>=0
JNZ	jnz <src>, <label>	Перейти на<label>, если <src>!=0
JODD JNEVEN	jodd <src>, <label> jneven <src>, <label>	Перейти на <label>, если младший бит <src> равен 1 (число в <src> нечетное)
JEVEN JNODD	Jeven <src>, <label> Jnodd <src>, <label>	Перейти на <label>, если младший бит <src> равен 0 (число в <src> четное)
JP	jp <src>, <label>	Перейти на <label>, если число единиц в <src> четно — т. е. сумма всех битов <src> по модулю 2 равна нулю
JNP	jnp <src>, <label>	Перейти на <label>, если число единиц в <src> нечетно — т. е. сумма всех битов <src> по модулю 2 равна единице
JNEG	jneg <src>, <label>	Перейти на <label>, если src<0 (Самый старший бит равен 1)
JEQ JE	jeq <src1>,<src2>, <label> je <src1>,<src2>, <label>	Перейти на <label>, если src1=src2 (равно)
JNEQ JNE	jneq <src1>,<src2>,<label> jne <src1>,<src2>,<label>	Перейти на <label>, если src1!=src2 (не равно)

Инструкции переходов в результате сравнения с учетом знака

JG JNLE	jg <src1>,<src2>,<label> jnle <src1>,<src2>,<label>	Перейти на <label>, если src1>src2 (больше)
JGEJNL	jge <src1>,<src2>,<label> jnl <src1>,<src2>,<label>	Перейти на <label>, если src1>=src2 (больше или равно)
JL JNGE	jl <src1>,<src2>,<label> jnge <src1>,<src2>,<label>	Перейти на <label>, если src1<src2 (меньше)
JLE JNG	jle <src1>,<src2>,<label> jng <src1>,<src2>,<label>	Перейти на <label>, если src1<=src2 (меньше или равно)

Старшие биты в операндах <src1> и <src2> трактуются как знаковые. А сами операнды — как знаковые целые числа, представленные в дополнительном коде.

Инструкции переходов в результате сравнения без учета знака

JUG JUNLE JA JNBE	jug <src1>,<src2>,<label> junle <src1>,<src2>,<label> ja <src1>,<src2>,<label> jnbe <src1>,<src2>,<label>	Перейти на <label>, если src1>src2 (больше)
JUGE JUNL JAE JNB	juge <src1>,<src2>,<label> junl <src1>,<src2>,<label> jae <src1>,<src2>,<label> jnb <src1>,<src2>,<label>	Перейти на <label>, если src1>=src2 (больше или равно)
JUL JUNGE JB JNAE	jul <src1>,<src2>,<label> junge <src1>,<src2>,<label> jb <src1>,<src2>,<label> jnae <src1>,<src2>,<label>	Перейти на <label>, если src1<src2 (меньше)
JULE JUNG JBE JNA	jule <src1>,<src2>,<label> jung <src1>,<src2>,<label> jbe <src1>,<src2>,<label> jna <src1>,<src2>,<label>	Перейти на <label>, если src1<=src2 (меньше или равно)

Старшие биты в операндах <src1> и <src2> трактуются как старшие разряды положительных чисел. А сами операнды — как беззнаковые положительные целые числа.

Инструкции вызова процедур и возврата из процедур

Мнемоника	Формат	Пояснения
CALL	call <proc_name>	Вызов процедуры
RET	Ret	Возврат из процедуры

2. Среда отладки микропрограмм

Отладка микропрограмм обеспечивается средствами систем WInter, HLCCAD и IEESD [1–3]. Напомним, что в этих системах обеспечена интегрированная среда редактирования, компиляции, симуляции и отладки по исходным текстам. Тем самым, существенно сокращается время на создание корректной микропрограммы, реализующей алгоритм функционирования разрабатываемого устройства.

Для реализации идеи автоматического синтеза микропрограммных автоматов было разработано следующее программное обеспечение:

- ModelMPA — модель виртуального процессора микропрограммных автоматов;
- AsmMPA — ассемблер микропрограмм;
- GenMPA — синтезатор цифровых схем микропрограммных автоматов.

Модель виртуального процессора ModelMPA работает под управлением систем HLCCAD и WInter, и предназначена для выполнения любой программы, написанной на языке MPDL. Синтезатор цифровых схем может использоваться только системой HLCCAD. Взаимодействие систем ModelMPA, AsmMPA и GenMPA со средами WINTER и HLCCAD показано на рис. 3.

Среда WInter с загруженной моделью виртуального процессора микропрограммных автоматов позволяет редактировать, ассемблировать, симулировать и отлаживать исходные тексты микропрограмм.

В дополнение к вышеописанным возможностям, HLCCAD/IEESD-2000 позволяет вызвать генератор схем микропрограммных автоматов GenMPA, а после получения схемы — визуализировать ее, проводить в случае необходимости ее оптимизацию, симуляцию, отладку, и, наконец, генерировать синтезируемое VHDL-описание.

Возможность симуляции и отладки генерируемых схем микропрограммных автоматов прежде всего предназначена для ручной оптимизации, а кроме того, используется для поиска и исправления ошибок генератора схем GenMPA и является важным средством расширяемости системы инструкций MPDL.

3. Генерация схем по микропрограммам

Для каждой инструкции, включенной в язык MPDL, разработана ее аппаратная реализация в виде схемы из стандартных синтезируемых элементов И, ИЛИ, НЕ, сумматоров и т. д.

Таким образом, ОА всегда представляет совокупность функциональных блоков F_i, каждый из которых реализует определенную микрооперацию f_i, использованную в микропрограмме. Некоторые блоки F_j являются блоками условных переходов. Эти блоки формируют сигналы истинности микроусловий X_i. В ОА также расположен блок памяти, который состоит из регистров и блоков ОЗУ, служащих для хранения промежуточных и итоговых результатов вычислений.

Операционные функциональные блоки реализуют микрооперации, которые не влияют на порядок выполнения инструкций микропрограммы, например, операции пересылки, сложения и т. д. На выходе каждого такого функционального операционного блока устанавливается тристабильный буфер, который пропускает результат микрооперации на блок памяти, только если соответствующий сигнал Y_i установлен. В противном случае на выходе тристабильного буфера устанавливается Z-состояние. То есть результат обработки данных операционным функциональным блоком записывается в память только при установленном Y_i.

Условные функциональные блоки реализуют микрооперации условных переходов. Эти блоки представляют собой комбинационные схемы, которые выдают сигналы истинности микроусловий. Каждый блок выдает два сигнала X_{2n} и X_{2n+1}, где n — это номер микроусловия. Если установлен X_{2n}, то микроусловие истинно. Если установлен X_{2n+1}, то микроусловие ложно. Если УА находится в состоянии, в котором данное микроусло-

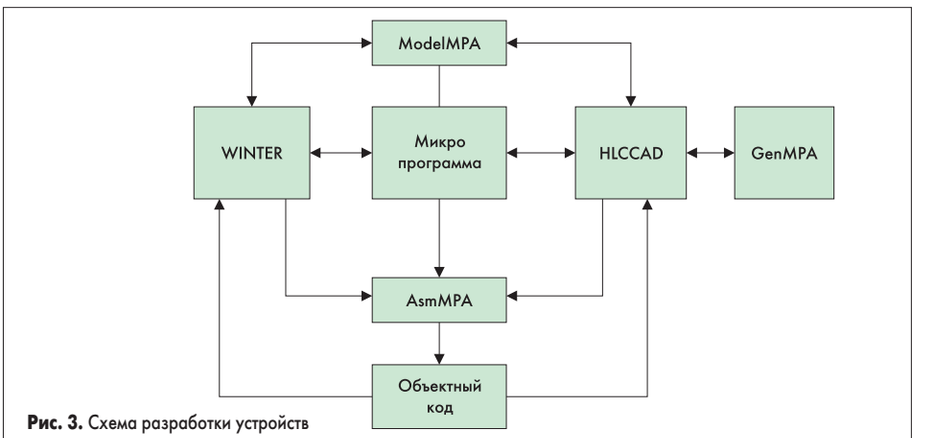


Рис. 3. Схема разработки устройств

вие не анализируется, то оба сигнала сбрасываются. Не существует ситуации, при которой оба сигнала могли бы быть установлены.

К настоящему моменту реализовано два вида управляющих автоматов: наиболее распространенный УА в виде схемы Уилкса-Стринжера [9] и разработанный нами УА типа «сдвиговый регистр». Каждый из этих вариантов обладает рядом преимуществ и недостатков. Зачастую предпочтение тому или иному варианту можно отдать только после синтеза соответствующего автомата. Как показала практика, схема Уилкса-Стринжера наиболее употребительна при синтезе малых микропрограмм при отсутствии жестких требований к быстродействию УА. При синтезе устройств по большим микропрограммам оказалось выгоднее использовать схему «сдвиговый регистр», так как она в этом случае обеспечивает большую экономию резервируемых под устройство ячеек ПЛИС. Если же сравнивать эти варианты с точки зрения быстродействия, то в большинстве случаев схема «сдвиговый регистр» позволяет работать УА на более высокой частоте. Тем не менее, выбор схемы УА рекомендуется производить после синтеза конкретного микропрограммного автомата и сравнения его реальных характеристик: количества ячеек, занимаемых в ПЛИС, и частоты, на которой может работать схема.

Схема Уилкса-Стринжера состоит из счетчика, дешифратора и комбинационной схемы переходов. Текущее состояние УА сопоставляется со значением, хранимым в счетчике. Дешифратор служит для преобразования состояния УА в выходные сигналы управляющего автомата Y_i.

Поясним механизм работы схемы Уилкса-Стринжера. В случае линейного исполнения микропрограммы просто инкрементируется счетчик. В случае же условного или безусловного переходов в счетчик записывается новое состояние УА, полученное от схемы переходов. Оно вычисляется по текущему состоянию УА и микроусловию X_i, соответствующему микрооперации условного перехода исполняемой в данный момент, полученному от ОА. В случае безусловного перехода, следующее состояние УА однозначно определяется по текущему.

Схема «сдвиговый регистр» названа так из-за подобия схеме сдвигового регистра. Она состоит из совокупности триггеров, каждый из которых соответствует какому-либо одному состоянию УА. В каждый момент времени только один триггер хранит единицу, остальные — нули. Номер триггера с единицей и соответствует состоянию УА. Выход каждого триггера связан с соответствующей линией Y_i.

Механизм работы схемы «сдвигового регистра» заключается в следующем. Для поддержки линейного исполнения микропрограммы триггеры соединяются последовательно, то есть вход триггера T_{i+1} соединяется с выходом триггера T_i, (аналогично тому, как это делается в сдвиговом регистре). Для поддержки условных и безусловных переходов вход триггера, соответствующий состоянию, в которое должен перейти УА, соединяется с линиями Y_i (i — номер состояния, после которого должен следовать безусловный переход в данное состояние) или X_i (i — номер логического условия, при истинности которого должен следовать условный переход в данное состояние). Если таких линий несколько, то они объединяются через логический элемент ИЛИ. При такой схеме быстродействие УА довольно высоко, так как оно определяется лишь временем срабатывания логического элемента ИЛИ и триггера.

(Окончание следует)

Литература

1. Долинский М., Литвинов В., Галатин А., Ермолаев И. HLCCAD — среда редактирования, симуляции и отладки аппаратного обеспечения // Компоненты и технологии. 2003. № 1.
2. Долинский М., Ермолаев И., Толкачев А., Гончаренко И. WInter — среда отладки программного обеспечения мультипроцессорных систем // Компоненты и технологии. 2003. № 2.
3. ht tp:/ /NewIT.gsu.unibel.by
4. Баранов С. И. Синтез микропрограммных автоматов. Л.: Машиностроение. 1979.
5. Майоров С. А., Новиков Г. И. Структура электронных вычислительных машин. Л.: Машиностроение. 1979.
6. Каган Б. М. Электронные вычислительные машины и системы. М.: Энергия. 1979.
7. Семенов Н., Каршенбойм И. Микропрограммные автоматы на базе специализированных ИС. Chip News. 2000. № 7. ht tp:/ / w ww .chipinfo.ru/literature/chipnews/ 200007/51.ht ml.
8. В. Лобанов. Технический минимум пользователя САПР MAX+PLUS II. ht tp:// ww w .chipinfo.r u/literature/chipnews/ 200101/56.ht ml.
9. Лазарев В. Г., Пийль Е. И. Синтез управляющих автоматов. М.: Энергоатомиздат. 1989.