Окончание. Начало в № 7'2003

Микропроцессор своими руками-2
Битовый процессор

Иосиф Каршенбойм

iosifk@narod.ru

Начинаем проект в FPGA

Так же, как и предыдущий проект [1], будем выполнять проект на языке AlteraHDL (AHDL). Этот язык достаточно широко распространен и описан. Кроме того, для начинающих он гораздо более доступен, чем другие языки группы VHDL. Так же более доступны и студенческие версии ПО.

Проект будет выполнен в виде текстовых файлов. Это позволит применить параметрическое выполнение описаний, что даст возможность применить файлы и в других проектах. В пакете ПО MaxPlus фирма Altera дает библиотеку параметризуемых мегафункций, позволяющих легко встраивать в проект счетчики, дешифраторы и пр. Описание этих мегафункций приведено в Help-файлах ПО.

Итак, проектируем:

1. RISC-процессор, потому что все команды выполняются за один такт синхрочастоты.
2. Применим для процессора гарвардскую структуру. Будем считать, что загрузка памяти команд нам не нужна и все команды будут храниться в памяти команд, что и происходит при инициализации микросхемы. Назовем память команд Program Space (PS). Память команд будет иметь разрядность 16 бит. Ограничим область адресов PS 7 битами, так чтобы она помещалась в один блок внутренней памяти FPGA. Для конкретной реализации микропроцессора введем параметр, описывающий разрядность шины адресов PS.
3. Расположим память данных в отдельной области памяти и представим ее как битовое поле. Назовем ее Data Space (DS).

   Ограничим область адресов DS 12 битами, что определяется выбором разрядности соответствующего поля команд микропроцессора, а адресное пространство соответственно будет ограничено значениями 0–FFF. Для конкретной реализации микропроцессора введем параметр, описывающий разрядность шины адресов DS. Представим, что входные переменные доступны по чтению в половине поля с младшими адресами от 0 и до 7FF, а выходные переменные доступны по чтению и проецируются в поле входных переменных в половине поля со старшими адресами, начиная с адреса 800. Представим, что выходы будут читаться в области адресов 800–BFF, а внутренние переменные будут читаться в адресах C00–FFF (рис. 1).
4. Необходимо задать разрядность внешней шины адресов области ввода — ее разрядность равна 11 битам.
5. Необходимо задать разрядность шины адресов внешней области вывода, ее разрядность равна 10 битам.
6. Представим, что микропроцессор по отношению к «внешнему миру» представляет собой активное устройство, имеющее выходную шину адресов, битовый вход и битовый выход, а также сигнал разрешения записи.
7. Будем считать, что системный сигнал СБРОС, действующий внутри кристалла, будет применяться и для нашего микропроцессора.
8. Сигнал РАЗРЕШЕНИЕ может использоваться для согласования режимов работы микропроцессора с дополнительными устройствами, например с блоком таймеров или с блоком DMA.

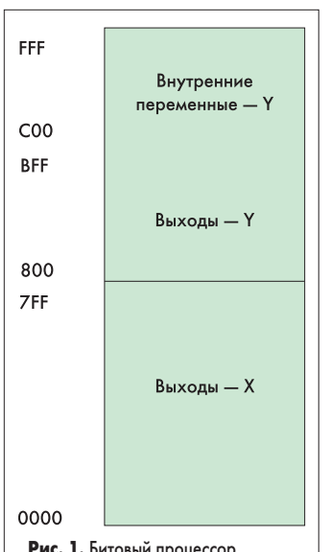

Рис. 1. Битовый процессор. Поле памяти

Построение ядра микропроцессора

Ядро микропроцессора будет состоять из следующих узлов:

а) аккумулятор разрядностью в 1 бит;
б) область DS будет состоять из областей ввода и вывода и части одного блока синхронной памяти, находящейся на кристалле;
в) область ввода будет состоять из входной шины, на которую поступают сигналы данных и части одного блока синхронной памяти, находящейся на кристалле;
г) область вывода будет представлять собой выходную шину и выходные данные будут записываться в часть блока синхронной памяти, находящейся на кристалле;
д) применим самую быструю структуру выборки данных из памяти, то есть синхронную выборку с конвейером команд.

На основании задания и сделанных упрощений на микропроцессор, получим архитектуру, показанную на рис. 2.

Микропроцессор состоит из набора следующих блоков:

- АЛУ, которое содержит узел контроля для выработки управляющих воздействий на все блоки микропроцессора — ALU;
- счетчик адресов памяти программ — PS_CNT;
- блок памяти программ — PS;
- блок памяти данных — DS.

На входы микропроцессора подадим сигнал синхрочастоты CLK, сигнал разрешения работы и системный сигнал СБРОС, так же, как и на все остальные блоки системы (на рис. не показано).

Входами микропроцессора будут сигналы с входной шины данных.

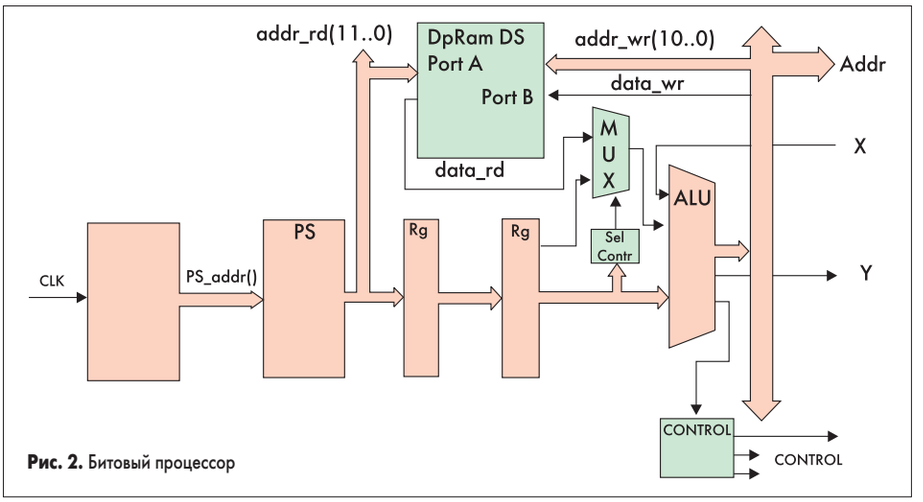

Рис. 2. Битовый процессор

Выходами микропроцессора будут сигналы с выходной шины данных и сигнал разрешения записи.

Счетчик команд при загрузке микросхемы или при инициализации системы по сигналу СБРОС устанавливается в состояние 0, после чего производит счет адресов памяти программ. Адреса поступают на вход блока PS. Данные, хранящиеся в памяти программ, выбираются в соответствии с поступившим адресом. Затем данные с выхода PS поступают как код адреса в двухпортовую память данных для того, чтобы можно было загрузить данные из памяти в аккумулятор. Чтобы скомпенсировать задержку, которую имеет синхронная память при чтении, данные с выхода PS поступают на двухкаскадный блок регистров задержки. ALU дешифрирует коды команд микропроцессора. В состав АЛУ входит триггер-аккумулятор, на котором и выполняются все команды микропроцессора. Данные на вход аккумулятора микропроцессора при чтении памяти поступают либо с выхода двухпортовой памяти, либо со входа X в соответствии с адресом шины адресов данных при чтении внешних устройств микропроцессора. Данные с выходов аккумулятора записываются либо только в двухпортовую память, либо одновременно в двухпортовую память и во внешние устройства.

Модель микропроцессора, описанная в данной статье, представляет собой полностью работоспособное устройство. По желанию этот проект может быть легко доработан до практического использования с учетом всех запросов пользователя. Методика разработки позволяет оценить трудоемкость и определить ресурс, необходимый для реализации микроконтроллера в FPGA.

Соглашение о названии сигналов (о наименовании цепей)

Для однообразия примем, что активное состояние сигналов будет равно 1. Если активное состояние сигнала будет равно 0, то в название сигнала на конце будет добавлена буква «*n*».

В файлах описаний применим следующие названия сигналов:

clk — частота, синхрочастота;

reset — сигнал системного сброса;

_ena — сигнал разрешения записи, в большинстве случаев его длительность будет равна длительности 1-го периода синхрочастоты;

_node — название сигнала относится к внутреннему узлу схемы.

Верхний файл проекта, описывающий микропроцессор

Верхний файл проекта one_bit_cpu.tdf — непосредственно микропроцессор. Для описания памяти команд применена библиотечная функция lpm_rom, и для ее инициализации применен файл step1.mif. Для описания двухпортовой памяти данных применена библиотечная функция lpm_ram_dp, для описания счетчика адресов применена библиотечная функция lpm_counter. Два каскада задержки реализованы на регистрах stage_a[PS_WIDTH-1..0] и stage_b[PS_WIDTH-1..0].

В отличие от АЛУ, описанного в предыдущей статье [1], в данном микропроцессоре нет прерываний, поэтому здесь АЛУ представляет собой простой дешифратор 4 в 16. На вход дешифратора подаются биты, соответствующие КОП. АЛУ вырабатывает сигналы управления для всего, что есть в данном проекте, и коммутирует данные на вход аккумулятора микропроцессора. Аккумулятор выполнен на триггере (acc).

Поскольку файл, описывающий данный микропроцессор, достаточно прост и содержит комментарии, то никаких дополнительных описаний здесь не приводится.

Далее приведен текст файла описания микропроцессора one_bit_cpu.tdf.

```
TITLE «Тестовый проект битового микропроцессор — one_bit_cpu»;
-- Version 1.0
-- (C) Iosif Karshenboim, 2003
-- You may use or distribute this function freely, provided you do not remove this copyright notice.
-- If you have questions or comments, feel free to contact me by email iosifk@narod.ru
-- ht tp: //w ww .iosifk.na rod.ru

INCLUDE «lpm_counter.inc»;
INCLUDE «lpm_ram_dp.inc»;
INCLUDE «lpm_rom.inc»;

PARAMETERS
(
 PS_WIDTHAD = 7,        -- разрядность шины адресов памяти команд
 PS_WIDTH = 16,         -- разрядность шины памяти команд
 DS_WIDTHAD = 12,       -- разрядность шины адресов памяти данных
 DS_WIDTH = 1,          -- разрядность шины данных
 -- PS_FILE = «step1.mif» -- это файл команд микропроцессора
 PS_FILE = «step2.mif»
);

CONSTANT ADDR_WIDTH = DS_WIDTHAD;

SUBDESIGN one_bit_cpu
(
clk, reset                  : INPUT = GND;  -- системные сигналы
ena                         : INPUT = VCC; -- разрешение работы
xx                          : INPUT; -- входные данные
wr,                         -- сигнал разрешения записи
y,                          -- выходы на управление от uP
addr[ADDR_WIDTH-2..0]       -- шина адресов для чтения и записи от uP
                            : OUTPUT;
)

VARIABLE
skip, data_ram, acc, acc_node, data_ram_wr, data_ram_in,
ps_cnt_node[PS_WIDTHAD-1..0], data,
ps_data[PS_WIDTH-1..0],
stage_a[PS_WIDTH-1..0], stage_b[PS_WIDTH-1..0]
                            : NODE;
BEGIN

-- Счетчик адресов памяти команд «PS»
ps_cnt_node[PS_WIDTHAD-1..0] = lpm_counter(.clock = clk, .clk_en = ena, .cnt_en = ena, aclr = reset, .sclr = skip )
                WITH (LPM_WIDTH = PS_WIDTHAD) RETURNS (.q[PS_WIDTHAD-1..0]);

-- выход счетчика адресов подается на вход памяти команд
ps_data[PS_WIDTH-1..0] = lpm_rom ( .address[PS_WIDTHAD-1..0] = ps_cnt_node[PS_WIDTHAD-1..0], .memenab = VCC )
                WITH (LPM_WIDTH = PS_WIDTH,  -- разрядность шины данных памяти команд
                LPM_WIDTHAD = PS_WIDTHAD,  -- разрядность шины адресов памяти команд
                LPM_FILE = PS_FILE, LPM_ADDRESS_CONTROL = «UNREGISTERED»,
                LPM_OUTDATA = «UNREGISTERED» );

-- два каскада задержки по которым «путешествует» КОП
-- т.е 15..12 разряды и адрес памяти данных, когда есть к ней обращение
FOR i IN 0 to PS_WIDTH-1 GENERATE
                stage_a[i] = DFFE(ps_data[i], clk, !reset, , ena );
                stage_b[i] = DFFE(stage_a[i], clk, !reset, , ena );
END GENERATE;

-- двухпортовая память, чтобы:
-- 1. читать-писать внутренние переменные
-- 2. читать состояние выходов
--    если выходы доступны на чтение, то сегмент адресов 800 — BFF может быть убран
data_ram = lpm_ram_dp (.wren = data_ram_wr & stage_b[ADDR_WIDTH-1], -- начиная с адресов 800
                .data[0] = data_ram_in,
                .wraddress[ADDR_WIDTH-2..0] = stage_b[ADDR_WIDTH-2..0],
                .wrclock = clk,
                .rden = ps_data[ADDR_WIDTH-1],
                .rdaddress[ADDR_WIDTH-2..0] = ps_data[ADDR_WIDTH-2..0],
                .rdclock = clk)
                WITH ( LPM_WIDTH = 1, -- разрядность 1 бит!!!
                LPM_WIDTHAD = ADDR_WIDTH-1, LPM_INDATA = «REGISTERED»,
                LPM_ADDRESS_CONTROL = «REGISTERED», LPM_RDADDRESS_CONTROL = «REGISTERED»,
                LPM_OUTDATA = «REGISTERED», USE_EAB = «ON» ) RETURNS (.q[0]);

-- сигналы записи и чтения входов и выходов
wr = data_ram_wr & ( stage_b[ADDR_WIDTH-1..ADDR_WIDTH-2] == B»10»); -- зона адресов 800 — BFF

-- мультиплексор входных данных от data_ram или из вне — X, но после задержки
IF stage_b[ADDR_WIDTH-1] == 0 THEN
  data = xx;
ELSE
  data = data_ram;
END IF;

-- АЛУ, дешифратор кодов операций

CASE stage_b[PS_WIDTH-1.. PS_WIDTH-4] IS
--=========================================
-- H«1» — MOV Acc, [Mem] — из аккумулятора в память
--=========================================
                WHEN 1 =>
                 skip = GND; acc_node = acc; data_ram_wr = VCC; data_ram_in = acc;
```

```
--=============================================
-- H«2» — MOVI Acc, [Mem] — из аккумулятора в память с инверсией
--=============================================
                WHEN 2 =>
                 skip = GND; acc_node = acc; data_ram_wr = VCC; data_ram_in = !acc;

--=============================================
-- H«3» — MOV Acc, [Mem] — из аккумулятора в память и с очисткой аккумулятора
--=============================================
                WHEN 3 =>
                 skip = GND; acc_node = GND; data_ram_wr = VCC; data_ram_in = acc;

--=============================================
-- H«4» — MOVI Acc, [Mem] — из аккумулятора в память с инверсией и с очисткой аккумулятора
--=============================================
                WHEN 4 =>
                 skip = GND; acc_node = GND; data_ram_wr = VCC; data_ram_in = !acc;

--=============================================
-- H«5» — MOV_AND [Mem], Acc — из памяти в аккумулятор с выполнением операции «И»
--=============================================
                WHEN 5 =>
                 skip = GND; acc_node = data & acc; data_ram_wr = GND; data_ram_in = GND;

--=============================================
-- H«6» — MOV_NAND [Mem], Acc — из памяти в аккумулятор с выполнением операции «НЕ-И»
--=============================================
                WHEN 6 =>
                 skip = GND; acc_node = !data & acc; data_ram_wr = GND; data_ram_in = GND;

--=============================================
-- H«7» — MOV_OR [Mem], Acc — из памяти в аккумулятор с выполнением операции «ИЛИ»
--=============================================
                WHEN 7 =>
                 skip = GND; acc_node = data # acc; data_ram_wr = GND; data_ram_in = GND;

--=============================================
-- H«8» — MOV_NOR [Mem], Acc — из памяти в аккумулятор с выполнением операции «НЕ-ИЛИ»
--=============================================
                WHEN 8 =>
                 skip = GND; acc_node = !data # acc; data_ram_wr = GND; data_ram_in = GND;

--=============================================
-- H«9» — MOV_XOR [Mem], Acc — из памяти в аккумулятор с выполнением операции «исключающее ИЛИ»
--=============================================
                WHEN 9 =>
                 skip = GND; acc_node = data $ acc; data_ram_wr = GND; data_ram_in = GND;

--=============================================
-- H«A» — MOV_XNOR [Mem], Acc — из памяти в аккумулятор с выполнением операции «НЕ-исключающее ИЛИ»
--=============================================
                WHEN H»A» =>
                 skip = GND;
                        acc_node = !data $ acc; data_ram_wr = GND; data_ram_in = GND;

--=============================================
-- H«B» — INV Acc — инверсия аккумулятора
--=============================================
                WHEN H»B» =>
                 skip = GND; acc_node = !acc; data_ram_wr = GND; data_ram_in = GND;

--=============================================
-- H«C» — SET Acc — установить аккумулятор в «1»
--=============================================
                WHEN H»C» =>
                 skip = GND; acc_node = VCC; data_ram_wr = GND; data_ram_in = GND;

--=============================================
-- H«D» — SKIP — пропустить остальное поле памяти
--=============================================
                WHEN H»D» =>
                 skip = VCC; acc_node = acc; data_ram_wr = GND; data_ram_in = GND;

--=============================================
-- H«E» — LDI 0, [Mem] — записать в память «0»
--=============================================
                WHEN H»E» =>
                 skip = GND; acc_node = acc; data_ram_wr = VCC; data_ram_in = GND;

--=============================================
-- H«F» — LDI 1, [Mem] — записать в память «1»
--=============================================
                WHEN H»F» =>
                 skip = GND; acc_node = acc; data_ram_wr = VCC; data_ram_in = VCC;
--=============================================
END CASE;

-- выходные сигналы из микропроцессора
acc = DFFE(acc_node, clk, !reset, , ena );      -- регистр — аккумулятор
y = acc;        -- выходные данные
addr[ADDR_WIDTH-2..0] = stage_b[ADDR_WIDTH-2..0];
-- шина адресов

END ;
```

Пример работы битового микропроцессора

Для примера представим, что у нас есть объект управления, состоящий из 16 входных переменных X0...X15.

Представим некоторую управляющую программу для 4 выходных переменных — Z0, Z1, Z2 и Z3. Пусть каждая из этих переменных описывается логической функцией, в среднем состоящей, к примеру, из 5 членов.

Z1 = (X0 AND X1) OR X6 OR X13 OR X14;
Z2 = ((X14 AND X10) NOR (X12 OR X11)) OR (X8 AND X3 AND X7 NAND X9);
Z3 = (X2 AND X3) XOR (X4 OR X15);
Z4 = (X2 AND X3) XNOR (X4 OR X5);

Введем переменные — Y5, Y6 и далее, соответствующие результатам промежуточных вычислений. Не будем здесь заниматься минимизацией, но учтем, что машина выполняет действия последовательно, поэтому преобразуем предыдущие команды к виду:

Y0 = X0 AND X1 OR X6 OR X13 OR X14;
                -- это соответствует Z0
Y1 = X14 AND X10;
Y2 = X8 AND X3 AND X7 NAND X9;
Y3 = X12 OR X11 NOR Y1 OR Y2;
                -- это соответствует Z1
Y4 = X2 AND X3;
Y5 = X4 OR X15 XOR Y4;
                -- это соответствует Z2
Y6 = X4 OR X5 XNOR Y4;
                -- это соответствует Z3

Представим, что входные переменные расположены по последовательным адресам, начиная с 0. То есть переменная X0 находится в поле памяти по адресу 0, переменная X1 — по адресу 1, и так далее. Представим, что выходы доступны по чтению с адреса 800, а выходные переменные доступны по чтению начиная с адреса C00. Теперь перепишем полученные данные в машинных командах и создадим файл программы, которую будет выполнять микропроцессор.

Для описания памяти команд применим библиотечную функцию lpm_rom, и для ее инициализации применим файл step1.mif. Далее по полученным машинным кодам сформируем файл инициализации памяти программ — файл типа «.mif». Файл step1 будет содержать коды команд микропроцессора, причем команда перехода к началу программы будет выдаваться после всех команд вычислений.

```
-- Version 1.0
-- Copyright Iosif Karshenboim, 2003
-- You may use or distribute this function freely, provided you do not remove this copyright notice.
-- If you have questions or comments, feel free to contact me by email iosifk@narod.ru
-- h ttp: //w ww.iosifk.nar od.ru

WIDTH = 16;
DEPTH = 128;

ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;

CONTENT BEGIN
 0 : 7000;     % MOV_OR [0], Acc   %
 1 : 5001;     % MOV_AND [1], Acc %
 2 : 7006;     % MOV_OR [6], Acc   %
 3 : 700D;     % MOV_OR [13], Acc   %
 4 : 700E;     % MOV_OR [14], Acc   %
 5 : 3800;     % MOVCL Acc, [800] — Y0 calculated   %

 6 : 700E;     % MOV_OR [14], Acc   %
 7 : 500A;     % MOV_AND [10], Acc   %
 8 : 3C00;     % MOVCL Acc, [C00] — Y1 calculated   %

 9 : 7008;     % MOV_OR [8], Acc   %
 A : 5003;     % MOV_AND [3], Acc   %
 B : 5007;     % MOV_AND [7], Acc   %
 C : 6009;     % MOV_NAND [9], Acc   %
 D : 3C01;     % MOVCL Acc, [C01] — Y2 calculated   %

 E : 700C;     % MOV_OR [12], Acc   %
 F : 700B;     % MOV_OR [11], Acc   %
 10: 8C00;     % MOV_NOR [C00], Acc   %
 11: 7C01;     % MOV_OR [C01], Acc   %
 12: 3801;     % MOVCL Acc, [801] — Y3 calculated   %

 13: 7002;     % MOV_OR [2], Acc   %
 14: 5003;     % MOV_AND [3], Acc   %
 15: 3C02;     % MOVCL Acc, [C02] — Y4 calculated   %

 16: 7004;     % MOV_OR [4], Acc   %
 17: 700F;     % MOV_OR [15], Acc   %
 18: 9C02;     % MOV_XOR [C02], Acc   %
 19: 3802;     % MOVCL Acc, [802] — Y5 calculated   %

 1A: 7004;     % MOV_OR [4], Acc   %
 1B: 7005;     % MOV_OR [5], Acc   %
 1C: AC02;     % MOV_XNOR [C02], Acc   %
 1D: 3803;     % MOVCL Acc, [803] — Y6 calculated   %

 1E: D000;     % SKIP          %

 1F: 0031;     % NOP           %
 20: 0032;     % NOP           %
 21: 0033;     % NOP           %
 22: 0034;     % NOP           %

 [23..7f] :          000000;
END;
```

В данной программе после команды SKIP еще 4 ячейки памяти заполнены кодами команды NOP. Это сделано для того, чтобы показать, как выполняется команда перехода SKIP в микропроцессоре с конвейером команд. Подробное описание работы с этой программой будет приведено в следующем разделе.

Создадим еще один файл инициализации памяти программы микропроцессора, файл step2.mif, который также будет содержать коды команд микропроцессора, но команда перехода к началу программы будет выдаваться на 2 такта раньше, чем последняя команда вычислений. На основе данного файла будут показаны последовательности выполнения команд переходов, приведенные в разделе «Время выполнения команды и конвейер команд» (см. «КиТ» № 7'2003).

Фрагмент файла step2.mif.
Файл step2.mif отличается от файла step1.mif только следующим фрагментом:

```
1A: 7004;    % MOV_OR [4], Acc   %
1B: 7005;    % MOV_OR [5], Acc   %
1C: D000;    % SKIP %
1D: AC02;    % MOV_XNOR [C02], Acc %
1E: 3803;    % MOVCL Acc, [803] — Y6 calculated %
```

Создайте рабочую папку, например My_cpu, запишите файл в эту папку, затем выполните компиляцию, чтобы убедиться в отсутствии ошибок. В качестве DEVICE для проекта выберем серию ACEX, а для выбора микросхемы установим режим AUTO. Произведем компиляцию проекта.

Что мы получили

По итогам компиляции проекта из файла рапорта получаем следующий результат:

Тестовый проект микропроцессор — one_bit_cpu
** DEVICE SUMMARY **

Chip/ POF	Device	Input Pins	Output Pins	Bidir Pins	Memory Bits	Memory % Utilized	LCs	LCs % Utilized
one_ bit_cpu	EP1K10 TC100-1 4	13	0	4096	33 %		64	11 %
User Pins:		4	13	0				

Для реализации проекта потребовалось всего 64 ячейки логики и 2 блока внутренней памяти.

Далее, создадим файл симуляции проекта микропроцессора с файлом команд step1 и произведем симуляцию проекта.

Симуляции работы микропроцессора приведены на рис. 3–5.

На диаграммах изображены входы:

- сброс — reset;
- сигнал разрешения работы — ena;
- синхрочастота — clk;
- вход битовых данных — xx.

На диаграммах изображены выходы:

- выход битовых данных — y;
- сигнал разрешения записи выходных данных — wr;
- шина адресов входных и выходных данных — addr[10..0].

Далее представлены внутренние сигналы микропроцессора:

- адрес, поступающий на вход блока памяти команд — lpm_rom;
- данные, поступающие на вход блока памяти данных — lpm_ram_dp;
- адрес, поступающий на вход блока памяти данных — lpm_ram_dp;
- два сигнала, соответствующие stage_a[], stage_b[];
- сигнал на выходе аккумулятора — acc.

Синхрочастота, поступающая на тактовый вход микропроцессора, имеет длительность импульса 30 нс, и каждый положительный фронт синхрочастоты пронумерован. Все описание работы будет опираться на соответствующий фронт синхрочастоты. Будем называть фронты так: фронт1, фронт2 и т. д.

Режим начала вычислений

После снятия сигнала «Сброс» счетчик адреса команд находится в состоянии 0000,

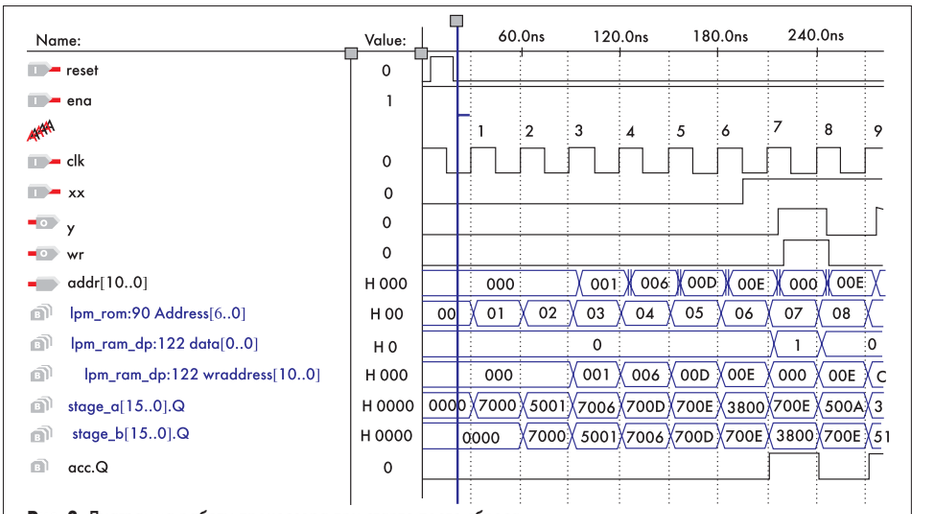

Рис. 3. Диаграмма работы процессора при старте после сброса

регистры конвейера команд обнулены. При действии фронта1 состояние сброса уже снято и в конвейер команд загружается первая команда. Поскольку на выходе конвейера команд эта команда появится только на фронте2, то АЛУ выполняет два первых такта как команды NOP. И только к фронту3 данные из конвейера будут обработаны в АЛУ как первая команда вычислений. Затем под каждый фронт синхрочастоты из конвейера команд будет выдаваться очередная команда, которая будет выполняться в этом же такте синхрочастоты. Далее читатели могут самостоятельно рассмотреть все действия, происходящие при проверке работы микропроцессора с другими командами из файла step1.

Выполнение команды перехода в начальный адрес, выполняемое без учета задержки, вносимой конвейером команд

После того как были вычислены все значения выходных переменных, была выдана команда перехода на адрес 0000. Рассмотрим выполнение микропроцессором данной операции, выполняемой в соответствии с файлом step1. Результат симуляции приведен на рис. 4. На приведенной диаграмме синий вертикальный курсор установлен в том месте, где на вход памяти команд поступает адрес команды 1E, по которому расположен код команды перехода. Этот фронт синхрочастоты пронумерован на данной диаграмме как фронт1. Далее этот код попадает в конвейер команд, по которому и «путешествует» до такта фронт3. При этом счетчик команд продолжает инкрементироваться и из памяти команд извлекаются данные, находящиеся в адресах 1F, 20. Только после выполнения двух лишних тактов происходит переход в начало программы. Такой режим работы характерен для всех микропроцессоров, имеющих конвейер команд. Обычно устройство управления блокирует АЛУ на время перезагрузки конвейера новыми данными. То есть для данного примера это были бы такты 1, 2.

Выполнение команды перехода в начальный адрес, выполняемое с учетом задержки, вносимой конвейером команд

Для проверки микропроцессора в режиме работы с учетом задержки, вносимой конвейером команд, создадим файл инициализации памяти программ микропроцессора step2.mif. Произведем компиляцию проекта с данным файлом инициализации памяти.

Рассмотрим выполнение микропроцессором операции перехода на адрес 0000, вы-

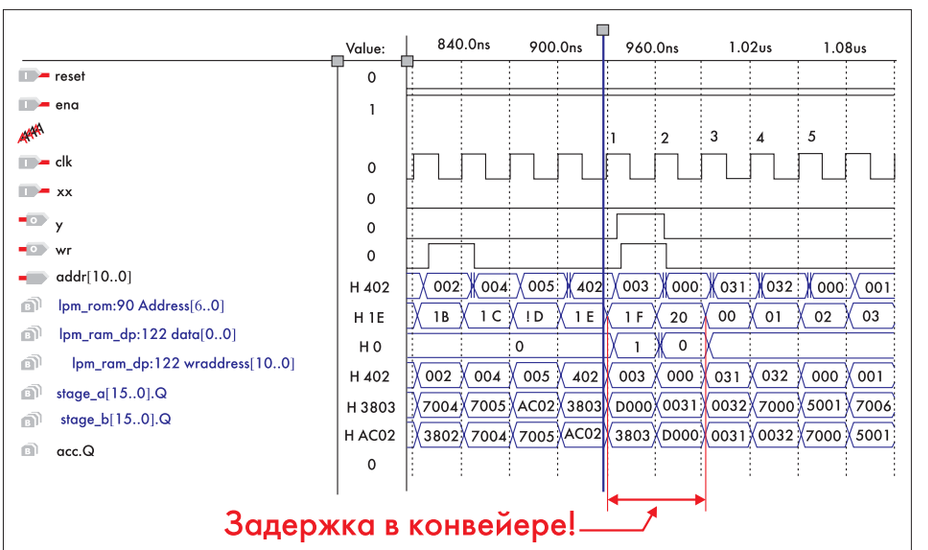

Рис. 4. Диаграмма работы микропроцессора для случая, когда команда перехода не выдается с упреждением

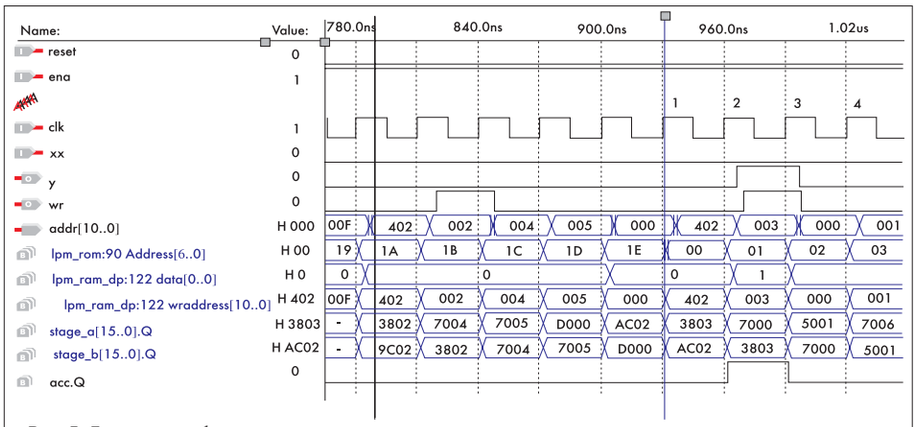

Рис. 5. Диаграмма работы микропроцессора для случая, когда команда перехода выдается с упреждением

полняемой в соответствии с файлом step2. Результат симуляции — временная диаграмма работы микропроцессора с файлом команд step2 — приведена на рис. 5. На приведенной диаграмме синий вертикальный курсор установлен, как и в предыдущем случае, в том месте, где на вход памяти команд поступает адрес команды 1E, по которому расположен код последней команды файла step2. Этот фронт синхрочастоты пронумерован на данной диаграмме как фронт1. Поскольку команда перехода на начальный адрес была выдана по адресу 1C, то к моменту времени, определяемому фронтом1, эта команда успела пройти весь конвейер и поступить в АЛУ. Устройство управления вырабатывает сигнал синхронного сброса счетчика команд. Далее из конвейера извлекаются команды, записанные туда еще до перехода, и эти команды исполняются. А на вход конвейера поступают команды, прочитанные по новому адресу. Таким образом, нет двух лишних тактов при переходе в начало программы.

Итак, можно сделать вывод, что для вычисления значений 4 выходных переменных по 16 входным для приведенного выше примера мы получили 31 машинную команду. А это значит, что за 31 цикл значения всех выходных параметров будут вычислены. При использовании серии микросхем Apex или Acex при тактовой частоте 33 МГц весь цикл вычислений для приведенного выше примера займет 0,93 микросекунды. На микросхемах серии Cyclon или Stratix при тактовой частоте 300 МГц мы получим результат всего за 102 наноосекунды. Если учесть, что за это же время микропроцессор типа AVR при тактовой частоте 8 МГц успеет выполнить примерно 8 команд, то преимущества становятся очевидными.

Еще один пример программирования на битовом процессоре

Описываемый процессор при кажущейся своей «маломерности» способен, тем не менее, выполнять широкий круг задач, возникающих при управлении объектами. Первая из таких задач — счетчик. Приведем пример выполнения программного 2-разрядного счетчика сигналов, приходящих по шине X с одного из входов. Чтобы предотвратить изменение сигнала во время одного цикла вычислений, будем записывать значение входного сигнала в ячейку Y() и далее будем использовать это значение для вычислений. Поскольку сигнал может меняться произвольно, программно определим положительный фронт сигнала, по которому будем вести счет. Далее приведены вычисления, необходимые для выполнения счета.

Y(0) = X(0); — здесь мы зафиксировали значение входа для текущего цикла вычислений.
Y(1) = Y(0) AND (NOT Y(4)); — выделитель фронта.
Y(2) = (Y(0) AND Y(1)) XOR Y(2); — младший разряд счетчика.
Старший разряд счетчика должен переключаться при условии, что младший разряд перешел в состояние 1, тогда получаем:
Y(3) = (Y(2) AND Y(1) AND Y(0)) XOR Y(3);
Y(4) = Y(0); — здесь фиксируется значение Y(0), задержанное на время обработки фронта.

Создадим файл программы обработки step3.mif, который отличается от предыдущих файлов следующими кодами команд:

```
0 : 7000;    % MOV_OR [0], Acc  %
1 : 1800;    % MOV Acc, [800]  %
2 : 6804;    % MOV_NAND [804], Acc  %
3 : 1801;    % MOV Acc, [801]  %
4 : 5800;    % MOV_AND [800], Acc  %
5 : 9802;    % MOV_XOR [802], Acc  %
6 : 1802;    % MOV Acc, [802]  %
7 : 5800;    % MOV_AND [800], Acc  %
8 : 5801;    % MOV_AND [801], Acc  %
9 : 9803;    % MOV_XOR [803], Acc  %
A : 3803;    % MOVCL Acc, [803]  %
B : 7800;    % MOV_OR [800], Acc  %
C : 3804;    % MOVCL Acc, [804]  %
D : D000;    % SKIP  %
```

Y(803), Y(802) — разряды счетчика, X(0) — входное событие, которое мы хотим сосчитать. Если во входном сигнале может быть дребезг, то выделим одну внутреннюю переменную Y(800) для фиксации приходящего импульса, Y(804) для фиксации задержанного импульса и еще одну переменную Y(801) для выделения фронта импульса. В тестовом проекте этим выходам будут соответствовать выходы: Y(800) ->y0, Y(801) ->y1, Y(802) ->y2, Y(803) ->y3, Y(804) ->y4.

Выполним тестовый проект с применением БПр. На выходную шину подключим четыре триггера, чтобы на диаграммах симуляции удобнее было наблюдать поведение сигналов.

```
TITLE «Тестовый проект битового микропроцессор — uP» ;
-- Version 1.0
-- Copyright Iosif Karshenboim, 2003
-- You may use or distribute this function freely,
-- provided you do not remove this copyright notice.
-- If you have questions or comments, feel free to
-- contact me by email at ik@lmail.loniis.spb.su
-- or iosifk@narod.ru
-- My World Wide WEB: h ttp: //w ww.iosifk.nar od.ru
-- Tabs = 2 Spaces

INCLUDE «one_bit_cpu.inc»;

PARAMETERS
( PS_FILE = «step3.mif» );

SUBDESIGN tst1
(
clk, reset      : INPUT = GND;              -- системные сигналы
xx              : INPUT; -- входные данные
wr,             -- сигнал разрешения записи
y0, y1, y2, y3, y4 : OUTPUT;-- выходы от триггеров, управляемые от uP
)

VARIABLE
cpu : one_bit_cpu WITH ( DS_WIDTHAD = 12, -- разрядность шины адресов памяти данных
                PS_FILE = PS_FILE);
BEGIN
-- подключим микропроцессор
 cpu.clk = clk; cpu.reset = reset; cpu.xx = xx;

-- выходные сигналы
wr = cpu.wr;
-- подключим два триггера на выходную шину микропроцессора
y0 = DFFE(cpu.y, clk, !reset, , cpu.wr & (cpu.addr[10..0] == 0));
                                                    -- регистр;
y1 = DFFE(cpu.y, clk, !reset, , cpu.wr & (cpu.addr[10..0] == 1));
                                                    -- регистр;
y2 = DFFE(cpu.y, clk, !reset, , cpu.wr & (cpu.addr[10..0] == 2));
                                                    -- регистр;
y3 = DFFE(cpu.y, clk, !reset, , cpu.wr & (cpu.addr[10..0] == 3));
                                                    -- регистр;
 y4 = DFFE(cpu.y, clk, !reset, , cpu.wr & (cpu.addr[10..0] == 4));
                                                    -- регистр;
END ;
```

Произведем симуляцию тестового примера. Результат симуляции показан на рис. 6.

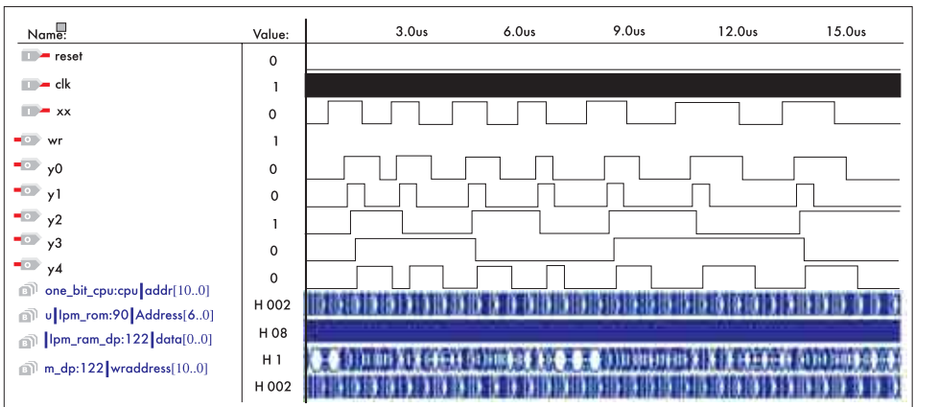

Рис. 6. Диаграмма работы процессора в режиме счета импульсов

Как видно на приведенной диаграмме, микропроцессор считает импульсы, приходящие на вход X, а на выходах y2 и y3 представлено выходное значение счетчика. Конечно, для обработки многоразрядного счетчика требуется затратить соответствующее число команд. Но поскольку мы имеем дело с FPGA, то в том случае, когда необходимо проводить подсчет большого количества переменных, можно ввести в состав периферии для данного микропроцессора массив счетчиков. Ячейки со счетными триггерами могут проецироваться в поле памяти микропроцессора и выполнять счет при обращении к данной ячейке.

Другая задача, очень часто встречающаяся при управлении дискретными объектами — статический автомат. Программирование статического автомата выполняется аналогично программированию счетчика.

Сравнительная характеристика

Для сравнения можно сказать, что широко известные микропроцессоры, такие, как MCS51, выполняют одну команду за 12 тактов, а их более «продвинутые» версии — за 4 такта, и только новейшие Cygnal — за 1 такт. Микропроцессоры, выполненные по структуре PIC16X, выполняют одну команду за 2 такта синхрочастоты. В таблице приведены сравнительные характеристики встроенных микропроцессоров. Сравнивая занимаемые в кристалле ресурсы, можно сделать вывод о том, что применение встроенного битового процессора оправдано, так как при очень малом занимаемом ресурсе он позволяет выполнить достаточно быстродействующие устройства обработки битовых данных.

Несколько слов об отладке программ во встроенных микропроцессорах

В том случае, когда разработчик применяет для отладки «внешние» по отношению к FPGA микропроцессоры, к таким микропроцессорам обычно поставляются и средства отладки. В новейших микропроцессорах часть внутрисхемного эмулятора встроена в кристалл и отладка осуществляется при подключении внешнего отладочного средства. Обычно для целей отладки программ используется интерфейс JTAG. При реализации микропроцессора, встроенного в систему на кристалле, можно также использовать внутрисхемный эмулятор, описав его как файл проекта и подключив к микропроцессору. Однако возможны и другие приемы отладки программ в FPGA, рассмотренные ранее [1]. Позволю себе повторить основные тезисы данного пункта, поскольку именно отладка разработанного проекта микропроцессора и программ, по которым он работает, представляет собой наиболее трудоемкую часть работы, а ресурсы примененного кристалла не всегда позволяют выполнить отладку каким-либо одним способом.

Во-первых, используемые микросхемы FPGA — перепрограммируемые, то есть в случае обнаружения ошибки всегда есть возможность исправить эту ошибку без переделки всего устройства, причем в случае

Таблица. Сравнение характеристик встроенных микроконтроллеров

Название ядра	Число ячеек	Блоки встроенной памяти	Тактовая частота, МГц	Микросхема
Nios 3.0 16 бит	1500		Более 125	Stratix
Nios 2.0 16 бит	1100	7	33	EP20K200E
Alatek 8051-standard[1]	2201	Без встроенной ROM	18	EPF10K100-1
CAST Inc. 8051-standard[2]	2554	3	102.32	Cyclone 1C3
CAST Inc. C165X3	774	–	63	EP20K200E
Битовый процессор	64	2	30–50 (Acex) 300 (Stratix)	Acex Stratix

Примечания:

1. ht tp:/ /w ww .alatek.com/pages/products/ip_industrial.asp.
2. ht tp:/ /w ww .altera.com/products/ip/processors/m-cas-8051-micro.h tml.
3. t tp:/ /w ww .altera.com/products/ip/processors/m-cas-pic165x.h tml.

FPGA, выполненных по технологии SRAM, перепрограммирование сводится к простой замене файла инициализации. Такая замена файла может быть выполнена и в удаленном режиме, когда изделие находится на объекте.

Во-вторых, при отладке программы во встроенных микропроцессорах есть много дополнительных аппаратных возможностей, используя которые, можно достичь быстрого положительного результата.

Поскольку весь проект «самодельного» микропроцессора для нас открыт и мы знаем его устройство, то мы можем облегчить отладку программ при помощи дополнительных аппаратных ресурсов, так как в процессе отладки можно использовать FPGA с большим количеством ячеек, чем нужно для функционирования проекта. Это позволит выделять отладочные ресурсы. К таким ресурсам можно отнести:

а) дополнительные поля памяти для вызова тестовых и мониторных программ;
б) тестовые входы и выходы для микропроцессора, позволяющие снимать состояния сигналов и задавать воздействия;
в) дополнительные поля памяти могут использоваться для включения в систему встроенных логических анализаторов, как входящих в комплект инструментального ПО, так и «самодельных», разработанных пользователем, что является очень мощным средством отладки (см. например, [5]);
г) дополнительные счетчики, тестовые триггеры и т. д.

Далее, ко встроенным микропроцессорам может быть «пристроен» блок, выполняющий функции внутрисхемного эмулятора. С его помощью пользователю становятся доступны внутренние регистры процессора, осуществляется останов по адресу, шаговый режим и пр.

Все эти возможности позволяют проводить успешную отладку программы не только в программном симуляторе, но и на «живом железе».

Заключение

В целом применение управляющего микропроцессора в FPGA позволяет повысить интеграцию устройства и перейти от «системы на плате» к «системе на кристалле».

Мы получили битовый RISC-микропроцессор, работающий на частоте 30–300 МГц и выполняющий все команды за один такт. И, следовательно, спроектированный микропроцессор имеет производительность 30–300 MIPS. Проект требует всего 64 ячейки логики и 2 блока внутренней памяти. При сравнении ресурсов кристалла, требуемых для такого микропроцессора, можно сказать, что они составляют не более десятой части от ресурсов ядра 8-разрядного микропроцессора.

Данный проект может послужить основой для разработки микропроцессора с требуемыми для вычислений ресурсами. На основе такого микропроцессора может быть разработан управляющий микроконтроллер путем добавления к проекту блоков периферийных устройств из имеющихся библиотечных мегафункций, добавления таймеров, счетчиков и других необходимых пользователю узлов. Проект имеет все возможности, чтобы его можно было легко доработать до реальной конструкции.

Конечно, при значительном увеличении числа входов и выходов увеличивается файл команд, требуемых для обработки данных, а следовательно, увеличивается время на обработку. Но в любом случае, это время будет меньше, чем у стандартного микропроцессора.

Следующей статьей в данном цикле будет статья о стековых процессорах — о Forth- и Java-микропроцессорах.

Литература

1. Каршенбойм И. Микропроцессор своими руками // Компоненты и Технологии. 2002. № 6–7.
2. Каршенбойм И. Микропроцессор для встроенного применения Nios. Система команд и команды, определяемые пользователем // Компоненты и Технологии. 2002. № 8–9.
3. Каршенбойм И. Микропроцессор для встроенного применения Nios. Конфигурация шин и периферии // Компоненты и Технологии. 2002. № 2–5.
4. И. Каршенбойм, Н. Семенов. Микропрограммные автоматы на базе специализированных ИС // Chip News. 2000. № 7.
5. И. Каршенбойм, К. Паленов. «Встроенный» логический анализатор — инструмент разработчика «встроенных» систем // Схемотехника. 2001. № 12.