Проектирование учебного процессора с фиксированной запятой в системе MATLAB/Simulink

Андрей СТРОГОНОВ,
д. т. н.
andreis@hotmail.ru

В работах [1, 2] с использованием системы команд из работ [3, 4] показаны примеры проектирования микропроцессорных ядер для реализации в базисе ПЛИС фирмы Altera с использованием как мегафункций асинхронного ОЗУ/ПЗУ САПР Quartus II, так и функциональных блоков на языке VHDL, сгенерированных с помощью Simulink HDL Coder системы MATLAB/Simulink. Общим недостатком работ [1, 2] является отсутствие управляющего автомата. В данной статье предлагается на основе системы команд из работы [4] спроектировать в системе MATLAB/Simulink процессор с управляющим автоматом, позволяющим проводить вычисления с фиксированной зяпятой. Выполнение арифметических операций над операндами, представленными в формате с фиксированной запятой, дает возможность получать высокую скорость вычислений, но возможно переполнение разрядной сетки либо появление значительной погрешности из-за округления [5].

На рис. 1 показан процессор с управляющим автоматом на шесть состояний и его отладка в системе MATLAB/Simulink с использованием Simulink Debugger. Перед отладкой необходимо в меню Simulation/Configuration Parameters выбрать диалог Solver («Решатели», методы численного решения дифференциальных и дифференциально-алгебраических уравнений). В Solver options выбрать **Type**: Fixed-step; **Solver**: discrete (no continuous state); **Fixed step size (fundamental sample time)** — 1.0. Осуществляется тестирование команд MOV A,12; MOV B,23; ADD A,B (рис. 1).

Проектируемый процессор состоит из следующих блоков: управляющий автомат (блок CPU_Controller, пример 1); память программ — ПЗУ процессора (блок Memory, пример 4); АЛУ процессора (блок alu, пример 7); два регистра общего назначения (РОН, блоки RegisterA, пример 6, и RegisterB); регистр специального назначения (РСН, блок PC_Inc, пример 2), который необходим для обеспечения «прыжковых» команд, таких как JMP, JMPZ, CALL и RET; счетчик команд (блок PC, пример 3); регистр инструкций (блок Instruction_Reg, пример 5).

Процессор реализован в формате с фиксированной запятой, с использованием fi-объектов системы MATLAB. Используем следующий формат, для представления десятичных чисел:

a = fi(v, s, w, f),

где v — десятичное число, s — знак (0 (false) — для чисел без знака и 1 (true) — для чисел со знаком), w — размер слова в битах (целая часть числа), f — дробная часть числа в битах. Все используемые десятичные числа в процессоре — беззнаковые (положительные) и целые. В системе MATLAB пользователь имеет возможность определить беззнаковые (например, uint8, uint16) и знаковые целые числа (sint) с помощью внутренних форматов.

При проектировании процессоров с фиксированной запятой необходимо учитывать следующие факторы: диапазон для результатов вычислений; требуемую погрешность результата; ошибки, связанные с квантованием; алгоритм реализации вычислений и др. Это связано с тем, что десятичное число v представляется с использованием формулы [5]:

$$v \cong 2^{-m} \times Q,$$

где m — длина дробной части числа; для беззнаковых чисел

$$Q = \sum_{i=0}^{n=1} W_i \times 2^i,$$

W_i — весовые коэффициенты, 2^i — веса двоичных разрядов машинного слова, n — длина двоичного слова в битах. Диапазон целого беззнакового числа определяется выражением: $0 \le v \le 2^n - 1$.

Это можно осуществить при помощи формата a = fi(v, s, w, f, fimath):

```
% HDL specific fimath
hdl_fm = fimath(...
'RoundMode', 'floor',...
'OverflowMode', 'wrap',...
'ProductMode', 'FullPrecision', 'ProductWordLength', 32,...
'SumMode', 'FullPrecision', 'SumWordLength', 32,...
'CastBeforeSum', true);
```

Данные настройки вычислений в формате с фиксированной запятой приняты в системе Simulink по умолчанию. Можно задать режим округления (Roundmode) — 'floor' — округление вниз; реакцию на переполнение (OverflowMode) — 'wrap' — перенос, при выходе значения v из допустимого диапазона «лишние» старшие разряды игнорируются. При выполнении операций умножения ('ProductMode') и сложения (SumMode) для повышения точности вычислений (precision) используется машинное слово шириной в 32 бита.

Для блоков РОН, в качестве примера, используем формат a = fi(v, s, w, f, fimath). Можно также добавить учет приведенных факторов и в другие M-файлы функций блоков процессора. Это позволит «управлять» встроенным генератором кода языка HDL (Simulink HDL Coder). Если этого не сделано, то необходимо с использованием проводника модели осуществить настройки блоков процессора для вычислений в формате с фиксированной запятой (рис. 2).

Процессор имеет распределенное управление. В блоках alu, RegisterA, RegisterB, PC_Inc и PC имеется свой локальный управляющий

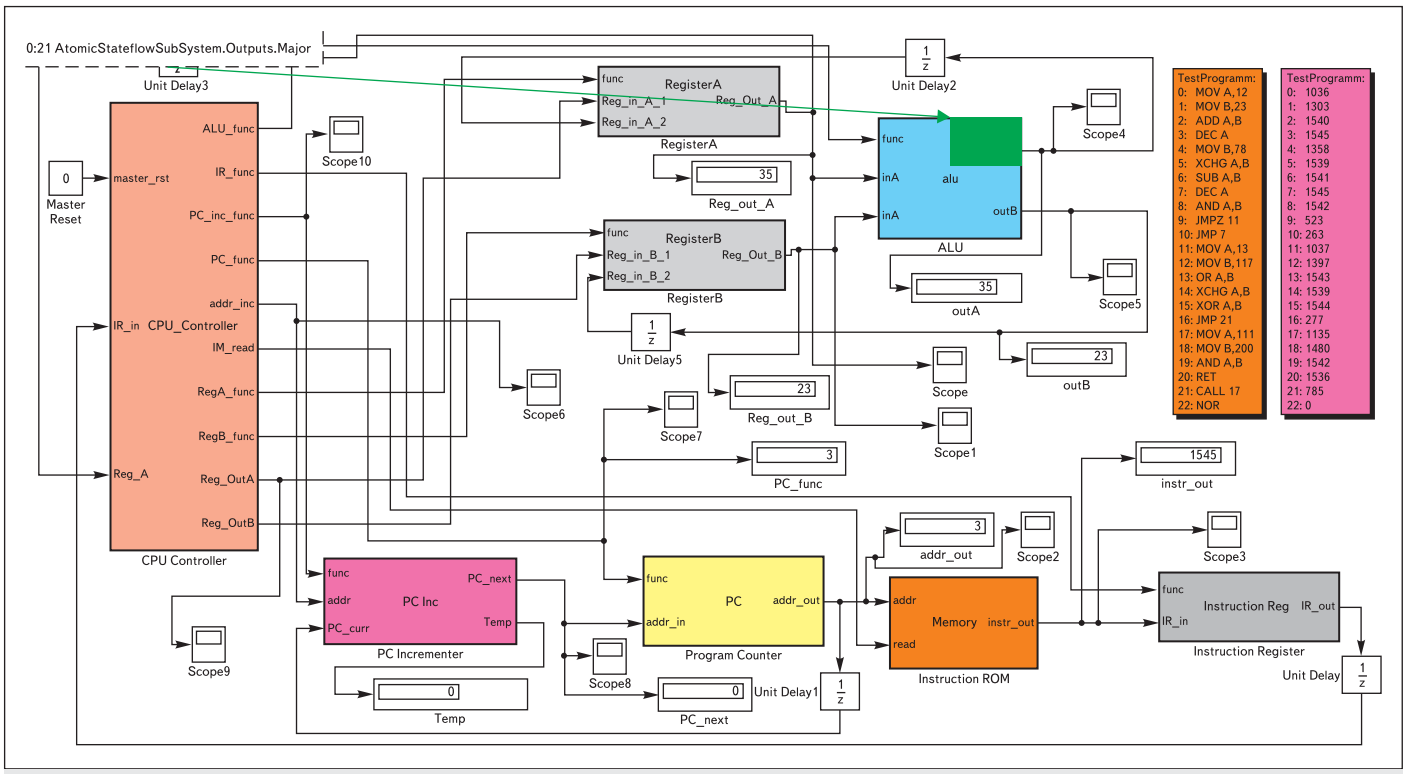

Рис. 1. Процессор с управляющим автоматом в системе MATLAB/Simulink (тестирование команд MOV A,12; MOV B,23; ADD A,B)

сигнал func, дешифрация которого внутри блоков будет приводить к выполнению некоторых операций, например, к изменению внутреннего содержимого блока или, наоборот, к его сохранению. Например, в блоке АЛУ локальный сигнал func 4-разрядный, десятичные числа с 0 по 8 кодируют логико-арифметические операции процессора, такие как ADD A,B; SUB A,B; AND A,B; OR A,B; XOR A,B и DEC, и команды пересылки, такие как MOV A,B; MOV B,A; XCHG A,B. В блоках PC_Inc, PC и Instruction_Reg сигнал func 2-разрядный, а в блоках RegisterA и RegisterB — 3-разрядный.

В примере 1 показан M-файл функции управляющего автомата микропроцессора в системе MATLAB/Simulink (блок CPU_Controller). Управляющий автомат может принимать 6 состояний. Состояния кодируются сигналом CPU_state в формате uint8 (целое десятичное число без знака с размером слова 8 бит). По сигналу master_rst (логическая 1) происходит установка автомата в нулевое состояние CPU_state = uint8(0). Далее происходит настройка блоков процессора с помощью локальных управляющих сигналов func:

```
PC_inc_func = fi(0, 0, 2, 0);
IR_func = fi(3, 0, 2, 0);
PC_func = fi(3, 0, 2, 0);
IM_read = fi(0, 0, 1, 0);
addr_inc = fi(0, 0, 8, 0);
Reg_OutA = fi(0, 0, 8, 0);
Reg_OutB = fi(0, 0, 8, 0);
RegA_func = fi(4, 0, 3, 0);
RegB_func = fi(4, 0, 3, 0);
ALU_func = fi(9, 0, 4, 0);
```

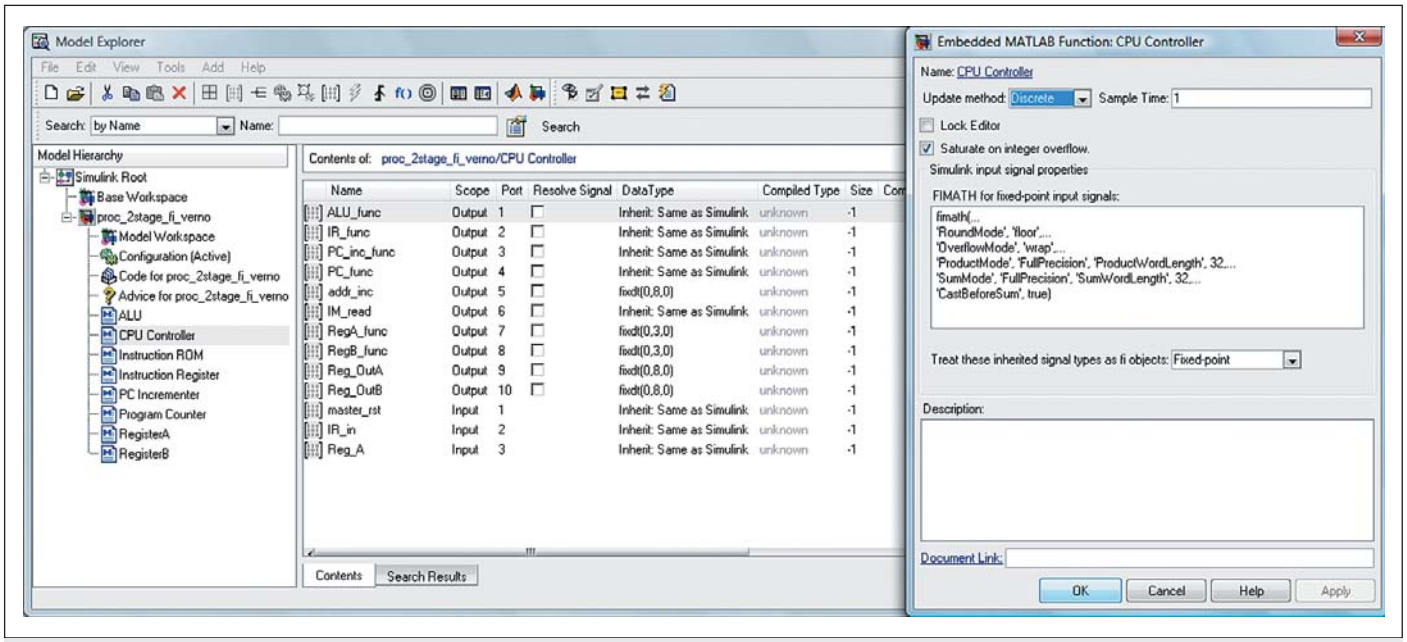

Рис. 2. Настройка блоков процессора с помощью проводника модели для вычислений в формате с фиксированной запятой

Управляющий автомат формирует на выходе PC_inc_func десятичный ноль, по которому внутреннее содержимое блока РСН будет сброшено (распознается блоком как сигнал сброса). На выходах PC_func и IR_func формируется десятичное число 3, по которому текущее содержимое счетчика команд и регистра инструкций остается неизменным. На выходах RegA_func и RegB_func формируется десятичное число 4, по которому текущее содержимое регистров общего назначения РОН А и В также остается неизменным. На выходе ALU_func формируется десятичное число 9, по которому в блоке АЛУ произойдет обход логико-арифметических операций и команд пересылки, а значения сигналов на входах inA и inB будут переданы на выход outA и outB без изменений:

```
outA = fi(inA, 0, 8, 0);
outB = fi(inB, 0, 8, 0);
```

На выходах IM_read, addr_inc, Reg_OutA, Reg_OutB автомат формирует десятичные нули. Ноль на выходе IM_read запрещает чтение из ПЗУ программ. Десятичные нули на выходах Reg_OutA и Reg_OutB означают нулевой операнд РОН для команд пересылки, ноль на выходе addr_inc — нулевой операнд РСН для команд передачи управления JMP, JMPZ и CALL.

В нулевом состоянии (case 0) осуществляется загрузка в РОН (блоки RegisterA, RegisterB), в РСН (блок PC_Inc) и в счетчик команд (блок PC) нуля (десятичный нуль преобразуется в формат с фиксированной запятой с размером слова 8 бит), а в регистр инструкций (Instruction_Reg) также загружается десятичный ноль, но он преобразуется в формат с фиксированной запятой с размером слова 16 бит. Эти операции осуществляются с помощью локальных сигналов управления PC_inc_func, PC_func, IR_func, RegA_func, RegB_func:

```
case 0,
   PC_inc_func = fi(0, 0, 2, 0);
   PC_func = fi(0, 0, 2, 0);
   IR_func = fi(0, 0, 2, 0);
   RegA_func = fi(0, 0, 3, 0);
   RegB_func = fi(0, 0, 3, 0);
   CPU_state = uint8(1);
```

Следующим состоянием автомата будет CPU_state = uint8(1). В этом состоянии и в двух последующих состояниях uint8(2) и uint8(3) происходит выделение полей команды. В состоянии 1 управляющий автомат формирует сигнал разрешения чтения команды из памяти IM_read = fi(1, 0, 1, 0). Поскольку порядковые номера строк в памяти программ начинаются с 1, например, data(1) = fi(1036, 0, 16, 0), то счетчик команд предварительно должен быть обнулен, то есть нулевое значение счетчика указывает на строку в ПЗУ с порядковым номером 1.

Для того чтобы счетчик команд содержал адрес следующей команды, управляющий автомат должен сформировать локальный сигнал управления счетчиком PC_func = = fi(2, 0, 2, 0), то есть на выходе PC_func должно присутствовать десятичное число 2, по которому текущее значение счетчика увеличится на 1. Поэтому данная строка стоит второй в операторе case 1. Извлеченную команду (в первоначальный момент и в последующие в регистре инструкций сохраняются текущие команды, а не следующие, загруженные в счетчик по команде PC_func = fi(2, 0, 2, 0)) из памяти программ в этом состоянии необходимо сохранить в регистре инструкций (16-битный регистр). Поэтому автомат сформирует локальный сигнал управления IR_func = fi(1, 0, 2, 0), разрешающий запись команды в регистр. Следующим состоянием, которое примет автомат, будет состояние CPU_state = uint8(2):

```
case 1,
   % Read from IM (ROM)
   IM_read = fi(1, 0, 1, 0);
   % PC increment PC+1
   PC_func = fi(2, 0, 2, 0);
   % store into IR
   IR_func = fi(1, 0, 2, 0);
   CPU_state = uint8(2);
```

Рассмотрим состояние 3 (case 3) управляющего автомата проектируемого процессора. Для того чтобы понять, как работает формат с фиксированной запятой, необходимо последовательно копировать ниже приведенные строки фрагмента М-файла и вставлять их в командную строку системы MATLAB.

Например, рассмотрим, как обрабатывается команда 1536 (RET). Из регистра инструкций целое беззнаковое десятичное число 1536 (размер слова 16 бит) поступает на вход IR_in управляющего автомата CPU Controller и присваивается переменной main_opcode, которая представляет 16-битную инструкцию. Из этой инструкции выделяется переменная major_opcode путем сдвига 16-битного вектора вправо на 8 позиций, с размером слова в 4 бита, таким образом мы выделяем биты с 9-го по 12-й из 16-разрядной инструкции. В системе команд [4] разряды с 13-го по 16-й нулевые, поэтому выделение переменной minor_opcode путем побитного И переменной major_opcode (4 разряда) и маски (переменная mask4, 4 разряда) в принципе не обязательно, но необходимо в случае последующей модификации системы команд процессора. Для выделения операнда (переменная address_data) из инструкции потребуется маска в 16 разрядов. Побитное И с переменной IR_in и с маской mask8 (0000000011111111) позволяет выделить переменную address_data с размером слова 8 бит. Для команды 1536 переменная address_data — это 8 нулей. Следующим состоянием, которое примет автомат, будет состояние CPU_state = uint8(4):

```
IR_in=fi(1536,0,16,0);
main_opcode = fi(IR_in, 0, 16, 0);
disp(bin(main_opcode))
%0000011000000000
% Сдвиг вектора вправо на 8 позиций
major_opcode= fi(bitsrl(main_opcode, 8), 0, 4, 0);
disp(bin(major_opcode))
%0110
mask4 = fi(15, 0, 4, 0);
disp(bin(mask4))
%1111
% Выделение команды, ширина поля 4 бита
minor_opcode = fi(bitand(major_opcode, mask4), 0, 4, 0);
disp(bin(minor_opcode))
%0110
% Выделение из команды операнда
mask8 = fi(255, 0, 16, 0);
disp(bin(mask8))
% 0000000011111111
address_data = fi(bitand(main_opcode, mask8), 0, 8, 0);
```

В состоянии 4 (case 4) происходит декодирование и выполнение инструкции (case 4). Декодирование происходит по сигналу minor_opcode (фактически 9-й, 10-й и 11-й биты сигнала IR_in, 12-й бит не используется, так как он нулевой). Далее декодируются 6 команд: NOP, JMP, JMPZ, CAL, MOV A,XX, MOV B,XX. Рассмотрим команду JMP. Выделенный операнд address_data из инструкции содержит адрес команды в ПЗУ, на который необходимо перейти. Операнд присваивается переменной addr_inc. Автомат формирует локальные сигналы управления РСН — PC_inc_func (десятичное число 1) и счетчика команд — PC_func (десятичное число 1). Далее выделенный операнд (он содержит адрес команды, на который необходимо перейти) будет загружен в РСН и в счетчик команд. При загрузке операнда в РСН содержимое счетчика команд сохраняется во внутренней переменной PC_Temp данного регистра (пример 2):

```
case 4,
      switch(uint8(minor_opcode))
         case 0,
            % NOP
            CPU_state = uint8(1);
         case 1,
            % JMP
            temp_addr_data = fi(address_data, 0, 8, 0);
            addr_inc = fi(temp_addr_data, 0, 8, 0);
            PC_inc_func = fi(1, 0, 2, 0);
            PC_func = fi(1, 0, 2, 0);
            CPU_state = uint8(1);
……….
……….
         case 6,
            switch(uint8(address_data))
            case 0,
               %RET
               PC_inc_func = fi(2, 0, 2, 0);
               PC_func = fi(2, 0, 2, 0);
               CPU_state = uint8(5);
……….
```

Если ни одна их этих команд не выполняется, то далее дешифрируются и отрабатываются команда RET, логико-арифметические команды (ADD A,B, OR A,B, XOR A,B, DEC A) и команды пересылки (MOV A,B, MOV B,A, XCHG A,B,). Последним состоянием является состояние case 5. В этом состоянии обновляются регистры РОН А и В, затем будет осуществлен переход в состояние 1. И весь описанный выше процесс обработки команды повторится вновь и до тех пор, пока не будет отработана последняя команда в программе.

```
function [ALU_func, IR_func, PC_inc_func, PC_func, ...
   addr_inc, IM_read, RegA_func, RegB_func, ...
   Reg_OutA, Reg_OutB] = CPU_Controller(master_rst, IR_in, Reg_A)

% CPU Controller
% 16-bit Instruction Encoding:
% -------------minor_opcode---------
% NOP:          00000 000 <00000000>
% JMP:          00000 001 <8-bit>
% JMPZ:         00000 010 <8-bit>
% CALL:         00000 011 <8-bit>
% MOV A,xx:     00000 100 <8-bit>
% MOV B,xx:     00000 101 <8-bit>
% ---------------------------------
% RET:          00000 110 <00000000>
% MOV A,B:      00000 110 <00000001>
% MOV B,A:      00000 110 <00000010>
% XCHG A,B:     00000 110 <00000011>
% ADD A,B:      00000 110 <00000100>
% SUB A,B:      00000 110 <00000101>
% AND A,B:      00000 110 <00000110>
% OR A,B:       00000 110 <00000111>
% XOR A,B:      00000 110 <00001000>
% DEC A:        00000 110 <00001001>

persistent CPU_state;
if(isempty(CPU_state))
   CPU_state = uint8(0);
end

if(master_rst)
   CPU_state = uint8(0);
end

PC_inc_func = fi(0, 0, 2, 0);
IR_func = fi(3, 0, 2, 0); % NOP
PC_func = fi(3, 0, 2, 0); % NOP
IM_read = fi(0, 0, 1, 0);
addr_inc = fi(0, 0, 8, 0);
Reg_OutA = fi(0, 0, 8, 0);
Reg_OutB = fi(0, 0, 8, 0);
RegA_func = fi(4, 0, 3, 0); % NOP
RegB_func = fi(4, 0, 3, 0); % NOP
ALU_func = fi(9, 0, 4, 0); % NOP

% main_code: <16..1>
% major_opcode: <16..9>
% minor_opcode: <12..9>
% address_data: <8..1>

persistent main_opcode;
persistent major_opcode;
persistent minor_opcode;
persistent address_data;

if(isempty(major_opcode))

   main_opcode = fi(0, 0, 16, 0);
   major_opcode = fi(0, 0, 4, 0);
   minor_opcode = fi(0, 0, 4, 0);
   address_data = fi(0, 0, 8, 0);
end

switch(CPU_state)
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % RESETTING OUTPUTS
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   case 0,
      PC_inc_func = fi(0, 0, 2, 0);
      PC_func = fi(0, 0, 2, 0);
      IR_func = fi(0, 0, 2, 0);
      RegA_func = fi(0, 0, 3, 0);
      RegB_func = fi(0, 0, 3, 0);
      CPU_state = uint8(1);

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % FETCH
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   case 1,
      % Read from IM (ROM)
      IM_read = fi(1, 0, 1, 0);
      % PC increment PC+1
      PC_func = fi(2, 0, 2, 0);
      % store into IR
      IR_func = fi(1, 0, 2, 0);
      CPU_state = uint8(2);

   case 2,
      % Read from IR
      IR_func = fi(2, 0, 2, 0);
      % Accommodating for the 'unit delay' from IR_out to IR_in
      CPU_state = uint8(3);

   case 3,
      % IR_in <16..1>
      main_opcode = fi(IR_in, 0, 16, 0);
      % IR_in <16..9>
      major_opcode = fi(bitsrl(main_opcode, 8), 0, 4, 0);
      % for instructions NOP,JMP,JMPZ,CALL,MOV A,xx MOV B,xx,RET
      % IR_in <12..9>
      mask4 = fi(15, 0, 4, 0);
      minor_opcode = fi(bitand(major_opcode, mask4), 0, 4, 0);
      % IR_in <8..1>
      mask8 = fi(255, 0, 16, 0);
      address_data = fi(bitand(main_opcode, mask8), 0, 8, 0);

      % Go to the decode stage
      CPU_state = uint8(4);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DECODE AND EXECUTE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
case 4,
      switch(uint8(minor_opcode))
         case 0,
            % NOP
            CPU_state = uint8(1);
         case 1,
            % JMP
            emp_addr_data = fi(address_data, 0, 8, 0);
            addr_inc = fi(temp_addr_data, 0, 8, 0);
            PC_inc_func = fi(1, 0, 2, 0);
            PC_func = fi(1, 0, 2, 0);
            CPU_state = uint8(1);
         case 2,
            %JMPZ
            temp_addr_data = fi(address_data, 0, 8, 0);
            if fi(Reg_A,0,8,0) == fi(0,0,8,0)
               addr_inc = fi(temp_addr_data, 0, 8, 0);
               PC_inc_func = fi(1, 0, 2, 0);
               PC_func = fi(1, 0, 2, 0);
            end
            CPU_state = uint8(1);
         case 3,
            % CALL
            temp_addr_data = fi(address_data, 0, 8, 0);
            addr_inc = fi(temp_addr_data, 0, 8, 0);
            PC_inc_func = fi(1, 0, 2, 0);
            PC_func = fi(1, 0, 2, 0);
            CPU_state = uint8(1);
         case 4,
            %MOV A,xx
            temp_addr_data = fi(address_data, 0, 8, 0);
            Reg_OutA = fi(temp_addr_data , 0, 8, 0);
            RegA_func = fi(1, 0, 3, 0);
            CPU_state = uint8(1);
         case 5,
            %MOV B,xx
            temp_addr_data = fi(address_data, 0, 8, 0);
            Reg_OutB = fi(temp_addr_data , 0, 8, 0);
            RegB_func = fi(1, 0, 3, 0);
            CPU_state = uint8(1);
         case 6,
               switch(uint8(address_data))
                  case 0,
                     %RET
                     PC_inc_func = fi(2, 0, 2, 0);
                     PC_func = fi(2, 0, 2, 0);
                     CPU_state = uint8(5);
                  case 1,
                     %MOV A,B
                     ALU_func = fi(0, 0, 4, 0);
                     RegA_func = fi(2, 0, 3, 0);
                     RegB_func = fi(2, 0, 3, 0);
                     CPU_state = uint8(5);
                  case 2,
                     %MOV B,A
                     ALU_func = fi(1, 0, 4, 0);
                     RegA_func = fi(2, 0, 3, 0);
                     RegB_func = fi(2, 0, 3, 0);
                     CPU_state = uint8(5);
                  case 3,
                     %XCHG A,B
                     ALU_func = fi(2, 0, 4, 0);
                     RegA_func = fi(2, 0, 3, 0);
                     RegB_func = fi(2, 0, 3, 0);
                     CPU_state = uint8(5);
                  case 4,
                     %ADD A,B
                     ALU_func = fi(3, 0, 4, 0);
                     RegA_func = fi(2, 0, 3, 0);
                     RegB_func = fi(2, 0, 3, 0);
                     CPU_state = uint8(5);
                  case 5,
                     %SUB A,B
                     ALU_func = fi(4, 0, 4, 0);
                     RegA_func = fi(2, 0, 3, 0);
                     RegB_func = fi(2, 0, 3, 0);
                     CPU_state = uint8(5);
                  case 6,
                     %AND A,B
                     ALU_func = fi(5, 0, 4, 0);
                     RegA_func = fi(2, 0, 3, 0);
                     RegB_func = fi(2, 0, 3, 0);
                     CPU_state = uint8(5);
                  case 7,
                     %OR A,B
                     ALU_func = fi(6, 0, 4, 0);
                     RegA_func = fi(2, 0, 3, 0);
                     RegB_func = fi(2, 0, 3, 0);
                     CPU_state = uint8(5);
                  case 8,
                     %XOR A,B
                     ALU_func = fi(7, 0, 4, 0);
                     RegA_func = fi(2, 0, 3, 0);
                     RegB_func = fi(2, 0, 3, 0);
                     CPU_state = uint8(5);
                  case 9,
                     %DEC A
                     ALU_func = fi(8, 0, 4, 0);
                     RegA_func = fi(2, 0, 3, 0);
                     CPU_state = uint8(5);
               end
         end
      case 5,
         RegA_func = fi(2, 0, 3, 0);
         RegB_func = fi(2, 0, 3, 0);
         CPU_state = uint8(1);
end
```

Пример 1. М-файл функции управляющего автомата микропроцессора (CPU_Controller) в системе MATLAB/Simulink

```
function [PC_next,Temp] = PC_Inc(func, addr, PC_curr)

% func = 0 => reset PC_Inc
% func = 1 => store into PC_Inc when JMP, JMPZ, CALL
% func = 2 => load from PC_Inc when RET

persistent PC_Temp;
if(isempty(PC_Temp))
   PC_Temp = fi(0, 0, 8, 0);
end
PC_next = fi(PC_curr, 0, 8, 0);
Temp = fi(0, 0, 8, 0);
switch(uint8(func))
   case 0,
      % reset PC_Inc
      PC_next = fi(0, 0, 8, 0);
   case 1,
      % store into PC_Inc when JMP, JMPZ, CALL
      PC_next = fi(addr, 0, 8, 0);
      PC_Temp = fi(PC_curr, 0, 8, 0);
      Temp = fi(PC_Temp, 0, 8, 0);
   case 2,
      % load from PC_Inc when RET
      PC_next = fi(PC_Temp, 0, 8, 0);
end
```

Пример 2. М-файл функции блока специального назначения (PC_Inc) в системе MATLAB/Simulink

```
function addr_out = PC(func, addr_in)

% Program Counter
% func = 0 => reset PC
% func = 1 => load PC
% func = 2 => increment PC

persistent PC_value;
if(isempty(PC_value))
   PC_value = fi(0, 0, 8, 0);
end

addr_out = fi(PC_value, 0, 8, 0);

switch(uint8(func))
   case 0,
      % reset
      PC_value = fi(0, 0, 8, 0);
   case 1,
      % store into PC
      PC_value = fi(addr_in, 0, 8, 0);
   case 2,
      % increment PC
      PC_value = fi(PC_value + 1, 0, 8, 0);
end
```

Пример 3. М-файл функции блока счетчика команд (PC) в системе MATLAB/Simulink

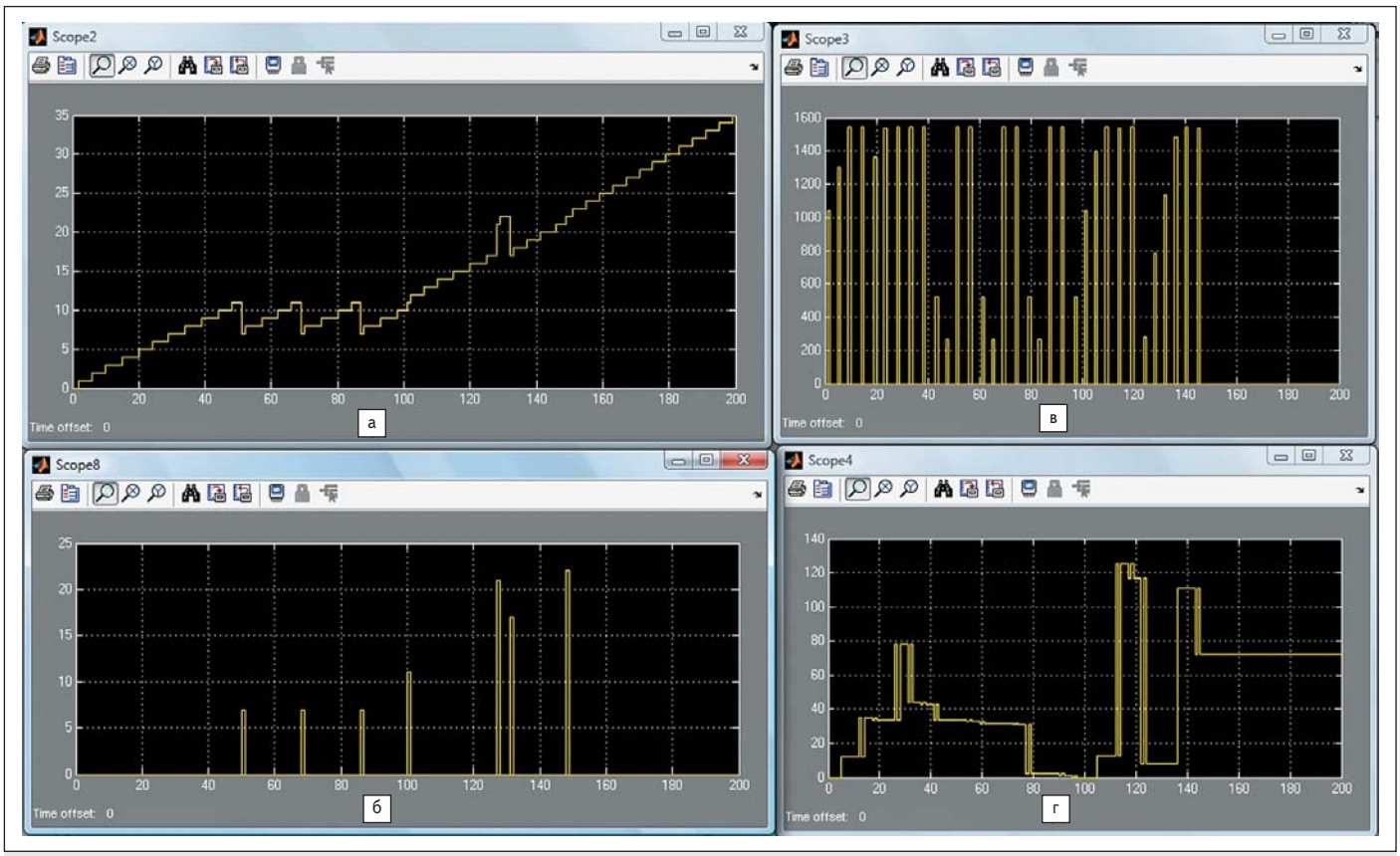

Рис. 3. Временные диаграммы работы процессора с управляющим автоматом в системе MATLAB/Simulink:
а) счетчик команд; б) блок специального назначения; в) память программ; г) РОН A

```
function instr_out = Memory(addr,read)
persistent data;
if(isempty(data))
data = fi(zeros(1, 256), 0, 16, 0);
end

data(1) = fi(1036, 0, 16, 0);
data(2) = fi(1303, 0, 16, 0);
data(3) = fi(1540, 0, 16, 0);
data(4) = fi(1545, 0, 16, 0);
data(5) = fi(1358, 0, 16, 0);
data(6) = fi(1539, 0, 16, 0);
data(7) = fi(1541, 0, 16, 0);
data(8) = fi(1545, 0, 16, 0);
data(9) = fi(1542, 0, 16, 0);
data(10) = fi(523, 0, 16, 0);
data(11) = fi(263, 0, 16, 0);
data(12) = fi(1037, 0, 16, 0);
data(13) = fi(1397, 0, 16, 0);
data(14) = fi(1543, 0, 16, 0);
data(15) = fi(1539, 0, 16, 0);
data(16) = fi(1544, 0, 16, 0);
data(17) = fi(277, 0, 16, 0);
data(18) = fi(1135, 0, 16, 0);
data(19) = fi(1480, 0, 16, 0);
data(20) = fi(1542, 0, 16, 0);
data(21) = fi(1536, 0, 16, 0);
data(22) = fi(785, 0, 16, 0);
data(23) = fi(0, 0, 16, 0);
if(read == 1)
   instr_out = data(addr+1);
else
   instr_out = fi(0, 0, 16, 0);
end
```

Пример 4. М-файл функции блока памяти программ (Memory) в системе MATLAB/Simulink

```
function IR_out = Instruction_Reg(func, IR_in)
% A 16-bit Instruction Register with the following func:
% func == 0 => reset
% func == 1 => store into IR
% func == 2 => read from IR
% otherwise, preserve old value and return 0
persistent IR_value;
if(isempty(IR_value))
   R_value = fi(0, 0, 16, 0);
end
IR_out = fi(0, 0, 16, 0);
switch(uint8(func))
   case 0,
      % reset
      IR_value = fi(0, 0, 16, 0);
   case 1,
      % store into IR
      IR_value = fi(IR_in, 0, 16, 0);
   case 2,
      % read IR
      IR_out = fi(IR_value, 0, 16, 0);
end
```

Пример 5. М-файл функции блока регистра инструкций (Instruction_Reg) в системе MATLAB/Simulink

```
function Reg_out_A = RegisterA(func, Reg_in_A_1, Reg_in_A_2)
% func == 0 => reset;
% func == 1 => store into RegisterA from port 1;
% func == 2 => store into RegisterA from port 2;
% func == 3 => read from RegisterA;
% HDL specific fimath
hdl_fm = fimath(...
'RoundMode', 'floor',...
'OverflowMode', 'wrap',...
'ProductMode', 'FullPrecision', 'ProductWordLength', 32,...
'SumMode', 'FullPrecision', 'SumWordLength', 32,...
'CastBeforeSum', true);
persistent Reg_value;
if(isempty(Reg_value))
   Reg_value = fi(0, 0, 8, 0, hdl_fm);
end

Reg_out_A = fi(Reg_value, 0, 8, 0, hdl_fm);

switch(uint8(func))
   case 0,
      % reset
      Reg_value = fi(0, 0, 8, 0, hdl_fm );
   case 1,
      % store into Reg_A from port 1
      Reg_value = Reg_in_A_1;
   case 2,
      % store into Reg_A from port 2
      Reg_value = Reg_in_A_2;
   case 3,
      % read Reg_A
      Reg_out_A = Reg_value;
end
```

Пример 6. М-файл функции блока регистра общего назначения A (RegisterA) в системе MATLAB/Simulink

```
function [outA, outB] = alu(func,inA,inB)
% This 8-bit ALU supports the following operations:
% MOV, XCHG, ADD, SUB, AND, OR, XOR, DEC
% func = 0 => MOV A,B
% func = 1 => MOV B,A
% func = 2 => XCHG A,B
% func = 3 => ADD A,B
% func = 4 => SUB A,B
% func = 5 => AND A,B
% func = 6 => OR A,B
% func = 7 => XOR A,B
% func = 8 => DEC A

% Simply pass the inA, when there is no designated func
outA = fi(inA, 0, 8, 0);
% Simply pass the inB, when there is no designated func
outB = fi(inB, 0, 8, 0);
switch (uint8(func))
        case 0, %MOV A,B
            outA = fi(inB, 0, 8, 0);
        case 1, %MOV B,A
            outB = fi(inA, 0, 8, 0);
        case 2, %XCHG A,B
            X_temp = fi(inB, 0, 8, 0);
            outB = fi(inA, 0, 8, 0);
            outA = fi(X_temp, 0, 8, 0);
        case 3, %ADD A,B
            outA = fi(inA + inB, 0, 8, 0);
        case 4, %SUB A,B
            outA = fi(inA – inB, 0, 8, 0);
        case 5, %AND A,B
            outA = fi(bitand(inA,inB), 0, 8, 0);
        case 6, %OR A,B
            outA = fi(bitor(inA,inB), 0, 8, 0);
        case 7, %XOR A,B
            outA = fi(bitxor(inA,inB), 0, 8, 0);
        case 8, %DEC A
            outA = fi(inA — 1, 0, 8, 0);
end
```

Пример 7. М-файл функции блока АЛУ в системе MATLAB/Simulink

На рис. 3 показаны временные диаграммы работы процессора с управляющим автоматом в системе MATLAB/Simulink. По оси Y откладываются целые беззнаковые десятич-

ные числа (которые преобразуются в процессе вычислений в формат с фиксированной запятой), а по оси X — время моделирования.

На рис. 3а видно, что значения, накопленные счетчиком команд, непрерывно увеличиваются, и только при команде JMP 7 (команда выполняется в программе 3 раза) счетчик изменяет свое значение на значение операнда, содержащееся в команде, то есть на 7. На рис. 3б показано содержимое блока PCH, на рис. 3в — содержимое памяти программ, а на рис. 3г — содержимое РОН A.

В системе MATLAB/Simulink разработан учебный вариант 8-разрядного процессора, позволяющего проводить вычисления в формате с фиксированной запятой, с управляющим автоматом на шесть состояний и системой команд из работы [4]. Преимущество такой архитектуры — ее адаптивность к последующим модификациям, например, если потребуется добавить дополнительные команды. Недостатком является отсутствие памяти данных и конвейера команд, поддержка незначительного числа команд, а также то, что процессор оперирует только с целыми положительными числами. ■

Литература

1. Строгонов А. Проектирование учебного процессора для реализации в базисе ПЛИС // Компоненты и технологии. 2009. № 3.
2. Строгонов А., Буслов А. Проектирование учебного процессора для реализации в базисе ПЛИС с использованием системы MATLAB/Simulink // Компоненты и технологии. 2009. № 5.
3. Тарасов И. Проектирование конфигурируемых процессоров на базе ПЛИС. Часть I // Компоненты и технологии. 2006. № 2.
4. Тарасов И. Проектирование конфигурируемых процессоров на базе ПЛИС. Часть II // Компоненты и технологии. 2006. № 3.
5. Жуков К. Г. Справочное руководство пользователя Fixed-Point Blockset. **www.exponenta.ru**