Окончание. Начало в КиТ № 11'2006

Цифровой генератор сигнала с перестраиваемой частотой, реализуемый на базе инструментального комплекта Spartan-3E Starter Kit фирмы Xilinx

Валерий ЗОТОВ
walerry@km.ru

Микропрограммное обеспечение цифрового генератора сигнала с перестраиваемой частотой

Исходный текст микропроцессорной программы на языке ассемблера для цифрового генератора сигнала с перестраиваемой частотой включает в себя следующие модули:

- блок определения значений констант, используемых в микропрограмме;
- блок инициализации системы;
- основной модуль микропрограммы;
- подпрограмма анализа состояния кнопочного переключателя, совмещенного с инкрементным энкодером;
- подпрограмма вычисления значений управляющих кодов, подаваемых на соответствующие входы цифрового синтезатора частоты;
- подпрограмма установки новых значений управляющих кодов для цифрового синтезатора частоты;
- подпрограмма отображения данных на экране жидкокристаллического дисплея;
- группа вспомогательных подпрограмм;
- подпрограмма обработки прерывания.

В начале первого модуля приведены директивы определения адресов выходных портов, используемых для управления состояния светодиодных индикаторов, жидкокристаллического дисплея и цифровым синтезатором.

```
;***************************************************************
   ;Port definitions
;***************************************************************
   CONSTANT LED_port, 80            ; 8 simple LEDs
   CONSTANT LED0, 01                ; LED 0 — bit0
   CONSTANT LED1, 02                ; 1 — bit1
   CONSTANT LED2, 04                ; 2 — bit2
   CONSTANT LED3, 08                ; 3 — bit3
   CONSTANT LED4, 10                ; 4 — bit4
   CONSTANT LED5, 20                ; 5 — bit5
   CONSTANT LED6, 40                ; 6 — bit6
   CONSTANT LED7, 80                ; 7 — bit7
   ;
   CONSTANT rotary_port, 00         ; Read status of rotary encoder
   CONSTANT rotary_left, 01         ; Direction of last move
                                    ; Left=1 Right=0 — bit0
   CONSTANT rotary_press, 02        ; Centre press contact
                                    ; (active High) — bit1
   ;
   ;LCD interface ports
   ;
   CONSTANT LCD_output_port, 40     ; LCD character module
                                    ; output data and control
   CONSTANT LCD_E, 01               ; active High Enable E — bit0
   CONSTANT LCD_RW, 02              ; Read=1 Write=0 RW — bit1
   CONSTANT LCD_RS, 04              ; Instruction=0 Data=1
                                    ; RS — bit2
   CONSTANT LCD_drive, 08           ; Master enable (active
                                    ; High) — bit3
   CONSTANT LCD_DB4, 10             ; 4-bit      Data DB4 — bit4
   CONSTANT LCD_DB5, 20             ; interface  Data DB5 — bit5
   CONSTANT LCD_DB6, 40             ;            Data DB6 — bit6
   CONSTANT LCD_DB7, 80             ;            Data DB7 — bit7
   ;
   CONSTANT LCD_input_port, 01      ; LCD character module
                                    ; input data
   CONSTANT LCD_read_DB4, 10        ; 4-bit      Data DB4 — bit4
   CONSTANT LCD_read_DB5, 20        ; interface  Data DB5 — bit5
   CONSTANT LCD_read_DB6, 40        ;            Data DB6 — bit6
   CONSTANT LCD_read_DB7, 80        ;            Data DB7 — bit7
   ;
   ;DDS control ports
   ;
   ;DDS control word is 32-bits
   ;
   CONSTANT DDS_control0_port, 02   ; dds_control_word(7:0)
   CONSTANT DDS_control1_port, 04   ; dds_control_word(15:8)
   CONSTANT DDS_control2_port, 08   ; dds_control_word(23:16)
   CONSTANT DDS_control3_port, 10   ; dds_control_word(31:24)
   ;
   ;Frequency scaling control word is 5-bits
   ;
   CONSTANT DDS_scaling_port, 20    ; dds_scaling_word(4:0)
```

Далее следует группа директив, с помощью которых задаются начальные значения переменных, используемых в процессе вычислений. В эту же группу входят директивы определения адресов ячеек сверхоперативного запоминающего устройства (СОЗУ), выделяемых для хранения различных параметров генератора сигнала, в том числе управляющих кодов для цифрового синтезатора частоты.

```
;***************************************************************
   ;Special Register usage
   ;Scratch Pad Memory Locations
;***************************************************************
   CONSTANT rotary_status, 00       ; Status of rotary encoder
   CONSTANT rotary_event, 80        ; flag set by interrupt in
                                    ; 'rotary_status' — bit7
   ;
   CONSTANT ISR_preserve_s0, 01     ; Preserve s0 contents during ISR
   ;
   CONSTANT LED_pattern, 02         ; LED pattern used in rotation mode
   ;
   ;BCD digits representing selected and displayed frequency
   ;
   CONSTANT BCD_digit0, 03          ; value  1
   CONSTANT BCD_digit1, 04          ; 10
   CONSTANT BCD_digit2, 05          ; 100
   CONSTANT BCD_digit3, 06          ; 1,000
   CONSTANT BCD_digit4, 07          ; 10,000
   CONSTANT BCD_digit5, 08          ; 100,000
   CONSTANT BCD_digit6, 09          ; 1,000,000
   CONSTANT BCD_digit7, 0A          ; 10,000,000
   CONSTANT BCD_digit8, 0B          ; 100,000,000
   ;
   ;Binary integer representation of BCD value
   ;
   CONSTANT frequency0, 0C          ; LS byte
   CONSTANT frequency1, 0D
   CONSTANT frequency2, 0E
   CONSTANT frequency3, 0F          ; MS byte
   ;
   ;Control of frequency selection values
   ;
   CONSTANT cursor_position, 10     ; Pointer to edit position on LCD
   CONSTANT edit_digit_pointer, 11  ; BCD digit to be changed
   ;
   ;80-bit product resulting from 32-bit frequency x 48-bit scaling constant
   ;
   CONSTANT product0, 12            ; LS byte
   CONSTANT product1, 13
   CONSTANT product2, 14
   CONSTANT product3, 15
   CONSTANT product4, 16
   CONSTANT product5, 17
   CONSTANT product6, 18
   CONSTANT product7, 19
   CONSTANT product8, 1A
   CONSTANT product9, 1B            ; MS byte
   ;
   ;Local copies of the DDS control word and DDS scaling word
   ;
   CONSTANT DDS_control0, 1C        ; dds_control_word(7:0)
   CONSTANT DDS_control1, 1D        ; dds_control_word(15:8)
   CONSTANT DDS_control2, 1E        ; dds_control_word(23:16)
   CONSTANT DDS_control3, 1F        ; dds_control_word(31:24)
   CONSTANT DDS_scaling, 20         ; dds_scaling_word(4:0)
   ;
   CONSTANT scale_constant0, 62     ; LS byte
   CONSTANT scale_constant1, 84
   CONSTANT scale_constant2, 11
   CONSTANT scale_constant3, 77
   CONSTANT scale_constant4, CC
   CONSTANT scale_constant5, AB     ; MS byte
```

Завершает первую секцию приведенная ниже последовательность директив, определяющих значение параметра задержки и символов таблицы ASCII.

```
;***************************************************************
   ASCII table
;***************************************************************
   CONSTANT delay_1us_constant, 0B
   ;
   CONSTANT character_a, 61
   CONSTANT character_b, 62
   CONSTANT character_c, 63
   CONSTANT character_d, 64
   CONSTANT character_e, 65
   CONSTANT character_f, 66
```

```
CONSTANT character_g, 67
CONSTANT character_h, 68
CONSTANT character_i, 69
CONSTANT character_j, 6A
CONSTANT character_k, 6B
CONSTANT character_l, 6C
CONSTANT character_m, 6D
CONSTANT character_n, 6E
CONSTANT character_o, 6F
CONSTANT character_p, 70
CONSTANT character_q, 71
CONSTANT character_r, 72
CONSTANT character_s, 73
CONSTANT character_t, 74
CONSTANT character_u, 75
CONSTANT character_v, 76
CONSTANT character_w, 77
CONSTANT character_x, 78
CONSTANT character_y, 79
CONSTANT character_z, 7A
CONSTANT character_A, 41
CONSTANT character_B, 42
CONSTANT character_C, 43
CONSTANT character_D, 44
CONSTANT character_E, 45
CONSTANT character_F, 46
CONSTANT character_G, 47
CONSTANT character_H, 48
CONSTANT character_I, 49
CONSTANT character_J, 4A
CONSTANT character_K, 4B
CONSTANT character_L, 4C
CONSTANT character_M, 4D
CONSTANT character_N, 4E
CONSTANT character_O, 4F
CONSTANT character_P, 50
CONSTANT character_Q, 51
CONSTANT character_R, 52
CONSTANT character_S, 53
CONSTANT character_T, 54
CONSTANT character_U, 55
CONSTANT character_V, 56
CONSTANT character_W, 57
CONSTANT character_X, 58
CONSTANT character_Y, 59
CONSTANT character_Z, 5A
CONSTANT character_0, 30
CONSTANT character_1, 31
CONSTANT character_2, 32
CONSTANT character_3, 33
CONSTANT character_4, 34
CONSTANT character_5, 35
CONSTANT character_6, 36
CONSTANT character_7, 37
CONSTANT character_8, 38
CONSTANT character_9, 39
CONSTANT character_colon, 3A
CONSTANT character_stop, 2E
CONSTANT character_semi_colon, 3B
CONSTANT character_minus, 2D
CONSTANT character_divide, 2F      ; '/'
CONSTANT character_plus, 2B
CONSTANT character_comma, 2C
CONSTANT character_less_than, 3C
CONSTANT character_greater_than, 3E
CONSTANT character_equals, 3D
CONSTANT character_space, 20
CONSTANT character_CR, 0D          ; carriage return
CONSTANT character_question, 3F    ; '?'
CONSTANT character_dollar, 24
CONSTANT character_exclaim, 21     ; '!'
CONSTANT character_BS, 08          ; Back Space command
                                   ; character
```

Блок инициализации системы включает в себя последовательность микропроцессорных команд, выполняющих начальную установку основных параметров генератора сигнала при включении напряжения питания. Данная последовательность инструкций осуществляет операции инициализации жидкокристаллического дисплея, а также определяет значения начальной позиции курсора и отображаемых параметров, которые соответствуют исходному номиналу частоты формируемого сигнала, равному 100 МГц. Кроме того, здесь же присутствует команда установки режима, разрешающего обработку прерываний. Рассматриваемый модуль микропрограммы выглядит следующим образом:

```
;*****************************************************************
      ;Initialise the system
;*****************************************************************
   cold_start: CALL LCD_reset      ; initialise LCD display
      ;
      ;Write 'Frequency Generator' to LCD display and display for 4 seconds
      ;
      LOAD s5, 10                  ; Line 1 position 0
      CALL LCD_cursor
      CALL disp_Frequency
      LOAD s5, 22                  ; Line 2 position 2
      CALL LCD_cursor
      CALL disp_Generator
      CALL delay_1s                ; wait 4 seconds
      CALL delay_1s
      CALL delay_1s
      CALL delay_1s
      CALL LCD_clear               ; clear screen
      ;
      ;Initial frequency of 100MHz
      ;
      LOAD s0, 00
      LOAD s1, 01
      STORE s0, BCD_digit0
      STORE s0, BCD_digit1
      STORE s0, BCD_digit2
      STORE s0, BCD_digit3
      STORE s0, BCD_digit4
      STORE s0, BCD_digit5
      STORE s0, BCD_digit6
      STORE s0, BCD_digit7
      STORE s1, BCD_digit8
      ;
      LOAD s0, 04                  ; Start position for editing
                                   ; frequency is 1MHz digit
      STORE s0, cursor_position
      LOAD s0, BCD_digit6
      STORE s0, edit_digit_pointer
      ;
      ENABLE INTERRUPT             ; interrupts are used to detect
                                   ; rotary controller
      CALL delay_1ms
      LOAD s0, 00                  ; clear the status of any spurious
                                   ; rotary events
      STORE s0, rotary_status      ; as a result of system turning on.
```

Основной модуль микропрограммы построен в виде бесконечного цикла, в котором выполняются следующие операции:

- вычисление значений управляющих кодов для цифрового синтезатора частоты;
- вывод текущего значения частоты выходного сигнала на экран жидкокристаллического дисплея;
- анализ состояния сигналов, формируемых блоком управления;
- переключение между режимами редактирования значения частоты формируемого сигнала;
- установка нового значения частоты формируемого сигнала;
- вычисление значений основных параметров и запись результатов в регистры и СОЗУ.

Исходный текст основного модуля микропрограммы имеет следующий вид.

```
;*****************************************************************
    ; Main program
;*****************************************************************
  move_mode: CALL compute_DDS_words ; compute DDS
                                    ; control values
    CALL display_freq               ; refresh display with
                                    ; cursor position shown
    LOAD s0, LED0                   ; indicate move mode on
                                    ; LEDs
    OUTPUT s0, LED_port
  move_wait: INPUT s0, rotary_port  ; read rotary encoder
    TEST s0, rotary_press           ;test for press of button
                                    ; which changes mode
    JUMP NZ, edit_mode
    FETCH s0, rotary_status         ;check for any rotation
                                    ; of rotary control
    TEST s0, rotary_event
    JUMP Z, move_wait
    ;
    AND s0, 7F                      ; clear flag now that
                                    ; action it is being
                                    ; processed
    STORE s0, rotary_status
    FETCH sA, cursor_position       ; read current position
    FETCH sB, edit_digit_pointer
    TEST s0, rotary_left            ; determine direction
                                    ; to move cursor
    JUMP Z, move_right
    ;
 move_left: COMPARE sB, BCD_digit8  ; can not move left of
                                    ; 100MHz digit
    JUMP Z, move_mode
    ADD sB, 01                      ; move to next higher
                                    ; BCD digit
    SUB sA, 01                      ; move cursor to match
                                    ; digit to be edited
    COMPARE sA, 09                  ; must skip over space
                                    ; separator
    JUMP Z, skip_left
    COMPARE sA, 05                  ; must skip over decimal
                                    ; point
    JUMP NZ, edit_point_update
 skip_left: SUB sA, 01              ; move cursor further left
    JUMP edit_point_update
    ;
 move_right: COMPARE sB, BCD_digit0 ; can not move right of
                                    ; 1Hz digit
    JUMP Z, move_mode
    SUB sB, 01                      ; move to next lower
                                    ; BCD digit
    ADD sA, 01                      ; move cursor to match
                                    ; digit to be edited
    COMPARE sA, 09                  ; must skip over space
                                    ; separator
    JUMP Z, skip_right
    COMPARE sA, 05                  ; must skip over decimal
                                    ; point
    JUMP NZ, edit_point_update
 skip_right: ADD sA, 01             ; move cursor further
                                    ; right
    ;
 edit_point_update: STORE sA, cursor_position ; update edit value
                                              ; in memory
    STORE sB, edit_digit_pointer
    JUMP move_mode
    ;
 edit_mode: CALL wait_switch_release ; wait for switch press to end
 edit_display: CALL compute_DDS_words ; compute DDS control
                                      ; values
    CALL display_freq               ; refresh display with
                                    ; new values
    LOAD s0, LED1                   ; indicate edit mode on
                                    ; LEDs
    OUTPUT s0, LED_port
 edit_wait: INPUT s0, rotary_port   ; read rotary encoder
    TEST s0, rotary_press           ; test for press of button
                                    ; which changes mode
    JUMP NZ, end_edit_mode
    FETCH s0, rotary_status         ; check for any rotation
                                    ; of rotary control
    TEST s0, rotary_event
    JUMP Z, edit_wait
    ;
    AND s0, 7F                      ; clear flag now that
                                    ; action it is being
                                    ; processed
    STORE s0, rotary_status
    FETCH sB, edit_digit_pointer    ; read pointer to BCD
                                    ; digit for initial change
    TEST s0, rotary_left            ; determine direction to
                                    ; increment or decrement
    JUMP Z, inc_digit
    ;
    ; Decrement the value starting at the current position and borrow-
ing from the left.
    ; However the value needs to bottom out at all 0's from the editing
position.
    ;
  dec_digit: FETCH sA, (sB)         ; read digit value at
                                    ; pointer position
    SUB sA, 01                      ; decrement digit
    COMPARE sA, FF                  ; test for borrow from
                                    ; next digit
    JUMP Z, dec_borrow
    STORE sA, (sB)                  ; store decremented
                                    ; digit value
    JUMP edit_display               ; decrement task
                                    ; complete
  dec_borrow: LOAD sA, 09           ; current digit rolls over
                                    ; to nine
    STORE sA, (sB)                  ; store '9' digit value
    COMPARE sB, BCD_digit8          ; check if working
                                    ; on 100MHz digit
    JUMP Z, set_min_value
    ADD sB, 01                      ; increment pointer to
                                    ; next most significant
                                    ; digit
```

```
    JUMP dec_digit                          ; decrement next digit up.
    ;
set_min_value: FETCH sB, edit_digit_pointer ; Must fill digits from
                                            ; insert to MS-Digit
                                            ; with 000...
    LOAD sA, 00
 fill_min: STORE sA, (sB)
    COMPARE sB, BCD_digit8                  ; check if filled to
                                            ; 100MHz digit
    JUMP Z, edit_display
    ADD sB, 01                              ; fill next higher digit
    JUMP fill_min
    ;
    ; Increment the value starting at the current position and carrying
to the left.
    ; However the value needs to saturate to all 9's from the editing po-
sition.
    ;
 inc_digit: FETCH sA, (sB)                  ; read digit value at
                                            ; pointer position
    ADD sA, 01                              ; increment digit
    COMPARE sA, 0A                          ; test for carry to next
                                            ; digit
    JUMP Z, inc_carry
    STORE sA, (sB)                          ; store incremented
                                            ; digit value
    JUMP edit_display                       ; increment task
                                            ; complete
 inc_carry: LOAD sA, 00                     ; current digit rolls over
                                            ; to zero
    STORE sA, (sB)                          ; store zero digit value
    COMPARE sB, BCD_digit8                  ; check if working on
                                            ; 100MHz digit
    JUMP Z, set_max_value
    ADD sB, 01                              ; increment pointer to
                                            ; next most significant
                                            ; digit
    JUMP inc_digit                          ; increment next digit up.
    ;
set_max_value: FETCH sB, edit_digit_pointer ; Must fill digits from
                                            ; insert to MS-Digit
                                            ; with 999...
    LOAD sA, 09
    fill_max: STORE sA, (sB)
    COMPARE sB, BCD_digit8                  ; check if filled to
                                            ; 100MHz digit
    JUMP Z, edit_display
    ADD sB, 01                              ; fill next higher digit
    JUMP fill_max
    ;
 end_edit_mode: CALL wait_switch_release    ; wait for end of switch
                                            ; press
    JUMP move_mode                          ; then go to move cursor
                                            ; mode
```

Подпрограмма анализа состояния кнопочного переключателя, совмещенного с инкрементным энкодером, осуществляет циклическое чтение информации из соответствующего входного порта ввода/вывода и определяет текущее положение этого переключателя. Работа этой подпрограммы завершается при фиксации момента освобождения кнопочного переключателя. Задержка длительностью 20 мс в начале каждого цикла чтения данных из порта ввода/вывода используется для подавления дребезга контактов переключателя.

```
;*****************************************************************
        ; Routine to poll the press switch
;*****************************************************************
    wait_switch_release: CALL delay_20ms    ; delay to aid switch
        ; de-bounce
        INPUT s0, rotary_port               ; read rotary encoder
        TEST s0, rotary_press               ; test if button is still
        ; being pressed
        JUMP NZ, wait_switch_release
        LOAD s0, 00                         ; clear flag indicating
        ; any  rotary events
        STORE s0, rotary_status
        RETURN
```

В подпрограмме расчета управляющих параметров для цифрового синтезатора частоты производится вычисление новых значений кодов управления, которые соответствуют последнему установленному номиналу частоты выходного сигнала, отображаемому на экране жидкокристаллического дисплея. В начале этой подпрограммы, текст которой приведен далее, выполняется преобразование десятичного значения частоты в 32-разрядный код. Затем выполняются вычисления новых значений управляющих кодов в соответствии с формулами (1) и (2). Полученные результаты вычислений записываются побайтно в соответствующие ячейки СОЗУ. В данной подпрограмме используется ряд вспомогательных подпрограмм (в частности, подпрограмма преобразования формы представления чисел, подпрограмма умножения операндов, представленных в двоичной форме, подпрограмма сдвига данных), текст которых приводится далее при рассмотрении соответствующей группы программных модулей.

```
;*****************************************************************
        ; Compute DDS control words from currently display frequency
value
;*****************************************************************
compute_DDS_words: CALL BCD_to_integer ; convert BCD display
                                       ; value to 32-bit value
        CALL scale_frequency           ; 80-bit product of 32-bit
                                       ; frequency x 48-bit scaling
                                       ; val
        FETCH sA, product9             ; read the upper part of the
                                       ; 80-bit product into
                                       ; [sA,s9,s8,s7,s6,s5,s4]
        FETCH s9, product8             ; The least significant 24-bits
                                       ; of the 80-bit product will
                                       ; never
        FETCH s8, product7             ; be used for frequencies
                                       ; above 1Hz.
        FETCH s7, product6             ; The final 32-bit DDS
                                       ; control word will be
                                       ; formed in
        FETCH s6, product5             ; [sA,s9,s8,s7]
        FETCH s5, product4
        FETCH s4, product3
        CALL shift80_left              ; multiply DDS control word
                                       ; by 4 to achieve default value
        CALL shift80_left
        LOAD sB, 00                    ; default scaling factor is 2
                                       ; (select counter bit0)
normalise_loop: TEST sA, 18            ; Test bits 27 and 28 of
                                       ; 32-bit DDS control word
        JUMP NZ, store_DDS_words       ; DDS control word is
                                       ; normalised to above
                                       ; 50MHz output
        CALL shift80_left              ; multiply DDS control word
                                       ; by 2
        ADD sB, 01                     ; Divide final value by 2
                                       ; to compensate
        COMPARE sB, 1F                 ; Test for maximum division
                                       ; factor
        JUMP NZ, normalise_loop
        LOAD sA, 08                    ; Set for minimum frequency
        LOAD s9, 00                    ; with phase accumulator set
                                       ; to generate 6.25MHz
        LOAD s8, 00
        LOAD s7, 00
store_DDS_words: STORE s7, DDS_control0  ; store local copy of
                                         ; control word
        STORE s8, DDS_control1         ; store local copy of control
                                       ; word
        STORE s9, DDS_control2         ; store local copy of control
                                       ; word
        STORE sA, DDS_control3         ; store local copy of control
                                       ; word
        STORE sB, DDS_scaling
        CALL drive_DDS_words           ; output control words to
                                       ; DDS circuit
        RETURN
```

Подпрограмма установки новых значений управляющих параметров для цифрового синтезатора частоты осуществляет побайтное считывание из СОЗУ последних вычисленных кодов, которые необходимы для формирования выходного сигнала с номиналом частоты, отображаемым на экране жидкокристаллического дисплея, и последующий их вывод в соответствующие выходные порты. Исходный текст этой подпрограммы выглядит следующим образом:

```
;*****************************************************
        ; Set DDS control words
;*****************************************************
    drive_DDS_words: FETCH s7, DDS_control0
        FETCH s8, DDS_control1
        FETCH s9, DDS_control2
        FETCH sA, DDS_control3
        FETCH s6, DDS_scaling
        OUTPUT s7, DDS_control0_port
        OUTPUT s8, DDS_control1_port
        OUTPUT s9, DDS_control2_port
        OUTPUT sA, DDS_control3_port
        OUTPUT s6, DDS_scaling_port
        RETURN
```

Подпрограмма отображения данных на экране жидкокристаллического дисплея осуществляет вывод текущего значения частоты выходного сигнала, установленного пользователем, и соответствующих этому значению параметров управления для цифрового синтезатора частоты. В этой подпрограмме, текст которой приведен далее, используется ряд вспомогательных подпрограмм для вывода на дисплей каждого типа данных.

```
;*****************************************************************
        ; Display frequency on top line of the LCD and DDS data on the
lower line
;*****************************************************************
    display_freq: CALL display_DDS_data ; display DDS information
                                        ; on lower line
        LOAD s5, 12                     ; Line 1 position 2
        CALL LCD_cursor
        FETCH s5, BCD_digit8            ; read 100MHz digit
        COMPARE s5, 00                  ; test for blanking
        JUMP Z, blank_100M_digit
        CALL display_digit              ; display non zero digit
        FETCH s5, BCD_digit7            ; read 10MHz digit and
                                        ; display
        CALL display_digit
        JUMP disp_1M_digit
        ;
    blank_100M_digit: CALL display_space ; blank 100MHz digit
        FETCH s5, BCD_digit7            ; read 10MHz digit
        COMPARE s5, 00                  ; test for blanking
        JUMP Z, blank_10M_digit
        CALL display_digit              ; display non zero digit
        JUMP disp_1M_digit
        ;
    blank_10M_digit: CALL display_space ; blank 10MHz digit
        ;
    disp_1M_digit: FETCH s5, BCD_digit6 ; read 1MHz digit and
                                        ; display
        CALL display_digit
        LOAD s5, character_stop         ; display decimal point
        CALL LCD_write_data
        ;
        LOAD s2, BCD_digit5             ; set pointer to 100KHz
                                        ; digit
        CALL display_3_digits
        CALL display_space
        LOAD s2, BCD_digit2             ; set pointer to 100Hz digit
        CALL display_3_digits
        LOAD s5, character_M            ; display 'MHz'
        CALL LCD_write_data
        LOAD s5, character_H
        CALL LCD_write_data
        LOAD s5, character_z
        CALL LCD_write_data
        ;
        FETCH s5, cursor_position       ; reposition edit cursor on
                                        ; display
        ADD s5, 10                      ; on line 1
        CALL LCD_cursor
        RETURN
        ;
    display_3_digits: LOAD s3, 03       ; 3 digits to display
        3digit_loop: FETCH s5, (s2)
        CALL display_digit
        SUB s2, 01                      ; decrement digit pointer
        SUB s3, 01                      ; count digits displayed
        JUMP NZ, 3digit_loop
        RETURN
```

```
        ;
display_digit: ADD s5, 30           ; convert BCD to ASCII
                                    ; character
        CALL LCD_write_data
        RETURN
        ;
display_space: LOAD s5, character_space
        CALL LCD_write_data
        RETURN
```

К группе вспомогательных программных модулей относятся:

- подпрограмма сдвига данных;
- подпрограмма преобразования символьного десятичного представления значения частоты в 32-разрядный код;
- подпрограмма умножения операндов, представленных в двоичной форме;
- подпрограмма вывода информации о параметрах управления работой цифрового синтезатора частоты на экран жидкокристаллического дисплея;
- подпрограммы преобразования и отображения шестнадцатиричных значений на экране жидкокристаллического дисплея;
- подпрограммы отображения статического текста на экране жидкокристаллического дисплея;
- подпрограммы формирования временных задержек;
- подпрограммы управления работой жидкокристаллического дисплея.

Подпрограмма сдвига данных выполняет побайтно циклический сдвиг 80-разрядного числа на один разряд влево. В состав данного программного модуля входит следующая последовательность инструкций:

```
shift80_left:  SL0 s4          ; shift (most of the) 80-bit value in
               SLA s5          ; [sA,s9,s8,s7,s6,s5,s4] left 1 place
               SLA s6
               SLA s7
               SLA s8
               SLA s9
               SLA sA
               RETURN
```

Подпрограмма преобразования символов десятичного числа в 32-разрядный код конвертирует последовательность из девяти цифр, которая соответствует десятичному представлению значения частоты выходного сигнала, отображаемого на экране жидкокристаллического дисплея, в целое число, представленное в 32-разрядной двоичной форме. Целочисленное представление значения частоты используется, в частности, при вычислении управляющих кодов для цифрового синтезатора частоты. Исходный текст данной подпрограммы на языке ассемблера имеет следующий вид:

```
;***************************************************************
        ; Convert 9 digit BCD frequency into 32-bit binary integer
;***************************************************************
   BCD_to_integer: LOAD s2, 09      ; 9 digits to convert
        LOAD s0, 00                 ; clear frequency value ready
                                    ; to accumulate result
        STORE s0, frequency0
        STORE s0, frequency1
        STORE s0, frequency2
        STORE s0, frequency3
        LOAD sB, 00                 ; initialise BCD digit
                                    ; weighting [sB,sA,s9,s8] to 1
        LOAD sA, 00
        LOAD s9, 00
        LOAD s8, 01
        LOAD s3, BCD_digit0         ; locate LS-digit
next_BCD_to_int_digit: FETCH s1, (s3)
BCD_digit_convert: COMPARE s1, 00 ; test for zero
        JUMP Z, next_digit_value
        FETCH s0, frequency0        ; add 32-bit digit weighting
                                    ; to memory value
        ADD s0, s8
        STORE s0, frequency0
        FETCH s0, frequency1
        ADDCY s0, s9
        STORE s0, frequency1
        FETCH s0, frequency2
        ADDCY s0, sA
        STORE s0, frequency2
        FETCH s0, frequency3
        ADDCY s0, sB
        STORE s0, frequency3
        SUB s1, 01                  ; decrement digit value
        JUMP BCD_digit_convert
        ;Increase weighting by 10x
next_digit_value: LOAD s7, sB       ; copy existing weighting
        LOAD s6, sA
        LOAD s5, s9
        LOAD s4, s8
        SL0 s8                      ; multiply weight by 4x (shift
                                    ; left 2 places)
        SLA s9
        SLA sA
        SLA sB
        SL0 s8
        SLA s9
        SLA sA
        SLA sB
        ADD s8, s4                  ; add previous weight to
                                    ; form 5x multiplication
        ADDCY s9, s5
        ADDCY sA, s6
        ADDCY sB, s7
        SL0 s8                      ; multiply weight by 2x (shift
                                    ; left 1 places)
        SLA s9
        SLA sA
        SLA sB                      ; weight value is now 10x
                                    ; previous value
        ADD s3, 01                  ; move to next digit for
                                    ; conversion
        SUB s2, 01
        JUMP NZ, next_BCD_to_int_digit
        RETURN
```

В процессе вычислений новых значений параметров управления для цифрового синтезатора частоты используется подпрограмма умножения двоичных чисел, исходный текст которой выглядит следующим образом:

```
;***************************************************************
        ; 32-bit x 48-bit multiply to scale the integer frequency
;***************************************************************
   scale_frequency: LOAD s0, 00     ; clear accumulator section
                                    ; of 'product'
        STORE s0, product9
        STORE s0, product8
        STORE s0, product7
        STORE s0, product6
        STORE s0, product5
        STORE s0, product4
        FETCH sB, frequency3        ; read frequency integer value
        FETCH sA, frequency2
        FETCH s9, frequency1
        FETCH s8, frequency0
        LOAD s1, 20                 ; 32-bit multiply
scale_mult_bit: SR0 sB
        ; shift right frequency integer
        SRA sA
        SRA s9
        SRA s8
        JUMP NC, product_shift      ; no add if bit is zero (note
                                    ; carry is zero)
        FETCH s0, product4          ; addition of scaling factor
                                    ; to most significant bits of
                                    ; product
        ADD s0, scale_constant0
        STORE s0, product4
        FETCH s0, product5
        ADDCY s0, scale_constant1
        STORE s0, product5
        FETCH s0, product6
        ADDCY s0, scale_constant2
        STORE s0, product6
        FETCH s0, product7
        ADDCY s0, scale_constant3
        STORE s0, product7
        FETCH s0, product8
        ADDCY s0, scale_constant4
        STORE s0, product8
        FETCH s0, product9
        ADDCY s0, scale_constant5
        STORE s0, product9          ; carry holds any overflow of
                                    ; addition
product_shift: FETCH s0, product9   ; Divide product by 2
                                    ; (shift right by 1)
        SRA s0                      ; overflow of addition
                                    ; included in shift
        STORE s0, product9
        FETCH s0, product8
        SRA s0
        STORE s0, product8
        FETCH s0, product7
        SRA s0
        STORE s0, product7
        FETCH s0, product6
        SRA s0
        STORE s0, product6
        FETCH s0, product5
        SRA s0
        STORE s0, product5
        FETCH s0, product4
        SRA s0
        STORE s0, product4
        FETCH s0, product3
        SRA s0
        STORE s0, product3
        FETCH s0, product2
        SRA s0
        STORE s0, product2
        FETCH s0, product1
        SRA s0
        STORE s0, product1
        FETCH s0, product0
        SRA s0
        STORE s0, product0
        SUB s1, 01                  ; move to next bit
        JUMP NZ, scale_mult_bit
        RETURN
```

Подпрограмма отображения информации об исходных параметрах синтеза, соответствующих выбранному значению частоты выходного сигнала, предназначена для вывода текущих значений управляющих кодов цифрового синтезатора частоты в нижней строке жидкокристаллического дисплея.

```
;***************************************************************
        ; Display DDS control information on the lower line of the LCD
display.
;***************************************************************
display_DDS_data: LOAD s5, 20       ; Line 2 position 0
        CALL LCD_cursor
        LOAD s5, character_N
        CALL LCD_write_data
        LOAD s5, character_equals
        CALL LCD_write_data
        LOAD s7, DDS_control3       ; pointer to most significant
                                    ; byte in memory
        CALL display_hex_32_bit
        CALL display_space
        LOAD s5, character_D
        CALL LCD_write_data
        LOAD s5, character_equals
        CALL LCD_write_data
        FETCH s0, DDS_scaling
        CALL display_hex_byte
        RETURN
```

В представленном программном модуле используются подпрограммы преобразования и отображения шестнадцатиричных значений на экране жидкокристаллического дисплея, исходный текст которых приведен далее:

```
;***************************************************************
        ; Routines to display hexadecimal values on LCD display
;***************************************************************
   hex_byte_to_ASCII: LOAD s2, s0   ; remember value supplied
        SR0 s0                      ; isolate upper nibble
        SR0 s0
        SR0 s0
```

```
        SR0 s0
        CALL hex_to_ASCII               ; convert
        LOAD s3, s0                     ; upper nibble value in s3
        LOAD s0, s2                     ; restore complete value
        AND s0, 0F                      ; isolate lower nibble
        CALL hex_to_ASCII               ; convert
        LOAD s2, s0                     ; lower nibble value in s2
        RETURN
        ;
        ; Convert hexadecimal value provided in register s0 into ASCII
character
        ;
        ;Register used s0
        ;
    hex_to_ASCII: SUB s0, 0A            ; test if value is in range 0 to 9
        JUMP C, number_char
        ADD s0, 07                      ; ASCII char A to F in range
                                        ; 41 to 46
        number_char: ADD s0, 3A         ; ASCII char 0 to 9 in range
                                        ; 30 to 40
        RETURN
        ;
        ;
        ; Display the two character HEX value of the register contents
's0' on the LCD
        ; at the current cursor position.
        ;
        ; Registers used s0, s1, s2, s3, s4, s5
        ;
    display_hex_byte: CALL hex_byte_to_ASCII
        LOAD s5, s3
        CALL LCD_write_data
        LOAD s5, s2
        CALL LCD_write_data
        RETURN
        ;
        ; Display the 32-bit value stored in 4 ascending memory loca-
tions as an 8 character
        ; HEX value at the current cursor position. Register s7 must con-
tain the memory
        ; location of the most significant byte (which is also the highest
address).
        ;
        ; Registers used s0, s1, s2, s3, s4, s5, s6, s7
        ;
    display_hex_32_bit: LOAD s6, 04     ; 4 bytes to display
    disp32_loop: FETCH s0, (s7)         ; read byte
        CALL display_hex_byte           ; display byte
        SUB s7, 01                      ; decrement pointer
        SUB s6, 01                      ; count bytes displayed
        RETURN Z
        JUMP disp32_loop
```

Подпрограммы отображения статического текста на экране жидкокристаллического дисплея используются для вывода текстовой информации, содержание которой не изменяется в процессе функционирования генератора. Данные программные модули содержат последовательности инструкций вывода соответствующих символов, которые выглядят следующим образом:

```
;*****************************************************
        ;LCD text messages
;*****************************************************
    disp_Frequency: LOAD s5, character_F
        CALL LCD_write_data
        LOAD s5, character_r
        CALL LCD_write_data
        LOAD s5, character_e
        CALL LCD_write_data
        LOAD s5, character_q
        CALL LCD_write_data
        LOAD s5, character_u
        CALL LCD_write_data
        LOAD s5, character_e
        CALL LCD_write_data
        LOAD s5, character_n
        CALL LCD_write_data
        LOAD s5, character_c
        CALL LCD_write_data
        LOAD s5, character_y
        CALL LCD_write_data
        RETURN
        ;
        ;Display 'Generator' on LCD at current cursor position
        ;
disp_Generator: LOAD s5, character_G
        CALL LCD_write_data
        LOAD s5, character_e
        CALL LCD_write_data
        LOAD s5, character_n
        CALL LCD_write_data
        LOAD s5, character_e
        CALL LCD_write_data
        LOAD s5, character_r
        CALL LCD_write_data
        LOAD s5, character_a
        CALL LCD_write_data
        LOAD s5, character_t
        CALL LCD_write_data
        LOAD s5, character_o
        CALL LCD_write_data
        LOAD s5, character_r
        CALL LCD_write_data
        CALL display_space
        LOAD s5, character_v
        CALL LCD_write_data
        LOAD s5, character_1
        CALL LCD_write_data
        LOAD s5, character_stop
        CALL LCD_write_data
        LOAD s5, character_2
        CALL LCD_write_data
        RETURN
```

Подпрограммы формирования задержек предназначены для реализации необходимых временных задержек различной длительности в процессе выполнения микропрограммы. Далее поочередно представлен исходный текст подпрограмм формирования задержек длительностью 1 мкс, 40 мкс, 1 мс, 20 мс и 1 с.

```
;*********************************************************************
        ; Software delay routines
;*********************************************************************
    delay_1us: LOAD s0, delay_1us_constant
    wait_1us: SUB s0, 01
        JUMP NZ, wait_1us
        RETURN
        ;
        ;Delay of 40us.
        ;
        ;Registers used s0, s1
        ;
    delay_40us: LOAD s1, 28             ; 40 x 1us = 40us
    wait_40us: CALL delay_1us
        SUB s1, 01
        JUMP NZ, wait_40us
        RETURN
        ;
        ;Delay of 1ms.
        ;
        ;Registers used s0, s1, s2
        ;
    delay_1ms: LOAD s2, 19              ; 25 x 40us = 1ms
    wait_1ms: CALL delay_40us
        SUB s2, 01
        JUMP NZ, wait_1ms
        RETURN
        ;
        ;Delay of 20ms.
        ;
        ;Delay of 20ms used during initialisation.
        ;
        ;Registers used s0, s1, s2, s3
        ;
    delay_20ms: LOAD s3, 14             ; 20 x 1ms = 20ms
    wait_20ms: CALL delay_1ms
        SUB s3, 01
        JUMP NZ, wait_20ms
        RETURN
        ;
        ;Delay of approximately 1 second.
        ;
        ;Registers used s0, s1, s2, s3, s4
        ;
    delay_1s: LOAD s4, 32               ; 50 x 20ms = 1000ms
    wait_1s: CALL delay_20ms
        SUB s4, 01
        JUMP NZ, wait_1s
        RETURN
```

В группу подпрограмм управления работой жидкокристаллического дисплея входят программные модули, выполняющие функции, которые необходимы для реализации используемого протокола взаимодействия микропроцессорного блока с интерфейсом этого дисплея. К этой группе, в частности, относятся подпрограммы инициализации дисплея, очистки экрана, управления курсором, записи и чтения данных.

```
;*****************************************************
        ;LCD Character Module Routines
;*****************************************************
LCD_pulse_E: XOR s4, LCD_E              ; E=1
        OUTPUT s4, LCD_output_port
        CALL delay_1us
        XOR s4, LCD_E                   ; E=0
        OUTPUT s4, LCD_output_port
        RETURN
LCD_write_inst4: AND s4, F8             ; Enable=1 RS=0 Instruction,
                                        ; RW=0 Write, E=0
        OUTPUT s4, LCD_output_port      ; set up RS and RW >40ns
                                        ; before enable pulse
        CALL LCD_pulse_E
        RETURN
  LCD_write_inst8: LOAD s4, s5
        AND s4, F0                      ; Enable=0 RS=0 Instruction,
                                        ; RW=0 Write, E=0
        OR s4, LCD_drive                ; Enable=1
        CALL LCD_write_inst4            ; write upper nibble
        CALL delay_1us                  ; wait >1us
        LOAD s4, s5                     ; select lower nibble with
        SL1 s4                          ; Enable=1
        SL0 s4                          ; RS=0 Instruction
        SL0 s4                          ; RW=0 Write
        SL0 s4                          ; E=0
        CALL LCD_write_inst4            ; write lower nibble
        CALL delay_40us                 ; wait >40us
        LOAD s4, F0                     ; Enable=0 RS=0 Instruction,
  RW=0 Write, E=0
        OUTPUT s4, LCD_output_port      ; Release master enable
        RETURN
  LCD_write_data: LOAD s4, s5
        AND s4, F0                      ; Enable=0 RS=0 Instruction,
  RW=0 Write, E=0
        OR s4, 0C                       ; Enable=1 RS=1 Data,
                                        ; RW=0 Write, E=0
        OUTPUT s4, LCD_output_port      ; set up RS and RW >40ns
                                        ; before enable pulse
        CALL LCD_pulse_E                ; write upper nibble
        CALL delay_1us                  ; wait >1us
        LOAD s4, s5                     ; select lower nibble with
        SL1 s4                          ; Enable=1
        SL1 s4                          ; RS=1 Data
        SL0 s4                          ; RW=0 Write
        SL0 s4                          ; E=0
        OUTPUT s4, LCD_output_port      ; set up RS and RW >40ns
                                        ; before enable pulse
        CALL LCD_pulse_E                ; write lower nibble
        CALL delay_40us                 ; wait >40us
        LOAD s4, F0                     ; Enable=0 RS=0 Instruction,
  RW=0 Write, E=0
        OUTPUT s4, LCD_output_port      ; Release master enable
        RETURN
  LCD_read_data8: LOAD s4, 0E           ;Enable=1 RS=1 Data,
                                        ; RW=1 Read, E=0
        OUTPUT s4, LCD_output_port      ; set up RS and RW >40ns
                                        ; before enable pulse
        XOR s4, LCD_E                   ; E=1
        OUTPUT s4, LCD_output_port
        CALL delay_1us                  ; wait >260ns to access data
        INPUT s5, LCD_input_port        ; read upper nibble
        XOR s4, LCD_E                   ; E=0
        OUTPUT s4, LCD_output_port
        CALL delay_1us                  ; wait >1us
        XOR s4, LCD_E                   ; E=1
        OUTPUT s4, LCD_output_port
        CALL delay_1us                  ; wait >260ns to access data
        INPUT s0, LCD_input_port        ; read lower nibble
        XOR s4, LCD_E                   ; E=0
        OUTPUT s4, LCD_output_port
        AND s5, F0                      ; merge upper and lower
                                        ; nibbles
        SR0 s0
        SR0 s0
        SR0 s0
        SR0 s0
        OR s5, s0
        LOAD s4, 04                     ; Enable=0 RS=1 Data,
                                        ; RW=0 Write, E=0
        OUTPUT s4, LCD_output_port      ; Stop reading 5V device and
                                        ; release master enable
        CALL delay_40us                 ; wait >40us
        RETURN
        ;
```

```
LCD_reset: CALL delay_20ms      ; wait more that 15ms for
                                ; display to be ready

    LOAD s4, 30
    CALL LCD_write_inst4        ; send '3'
    CALL delay_20ms             ; wait >4.1ms
    CALL LCD_write_inst4        ; send '3'
    CALL delay_1ms              ; wait >100us
    CALL LCD_write_inst4        ; send '3'
    CALL delay_40us             ; wait >40us
    LOAD s4, 20
    CALL LCD_write_inst4        ; send '2'
    CALL delay_40us             ; wait >40us
    LOAD s5, 28                 ; Function set
    CALL LCD_write_inst8
    LOAD s5, 06                 ; Entry mode
    CALL LCD_write_inst8
    LOAD s5, 0E                 ; Display control
    CALL LCD_write_inst8
    LCD_clear: LOAD s5, 01      ; Display clear
    CALL LCD_write_inst8
    CALL delay_1ms              ; wait >1.64ms for display to
                                ; clear
    CALL delay_1ms
    RETURN
    ;
LCD_cursor: TEST s5, 10         ; test for line 1
    JUMP Z, set_line2
    AND s5, 0F                  ; make address in range 80 to
                                ; 8F for line 1
    OR s5, 80
    CALL LCD_write_inst8        ; instruction write to set cursor
    RETURN
set_line2: AND s5, 0F           ; make address in range C0 to
CF for line 2
    OR s5, C0
    CALL LCD_write_inst8        ; instruction write to set cursor
    RETURN
    ;
LCD_shift_left: LOAD s5, 18     ; shift display left
    CALL LCD_write_inst8
    RETURN
```

Подпрограмма обработки прерывания осуществляет чтение данных из порта, используемого для сопряжения с устройством управления рассматриваемого генератора, и запись текущего состояния инкрементного энкодера в выделенную для этого ячейку СОЗУ.

```
;*****************************************************************
    ;Interrupt Service Routine (ISR)
;*****************************************************************
ISR: STORE s0, ISR_preserve_s0      ; preserve s0
    INPUT s0, rotary_port           ; read rotary encoder
    OR s0, rotary_event             ; set flag
    STORE s0, rotary_status         ; put result in SCM
    FETCH s0, ISR_preserve_s0       ; restore s0
    RETURNI ENABLE
```

Завершает микропроцессорную программу код определения вектора обработки прерывания, который записывается по адресу 3FF в соответствии с правилами, принятыми для используемой версии микропроцессорного ядра семейства PicoBlaze [2, 7].

```
;*********************************
    ;Interrupt Vector
;*********************************
    ADDRESS 3FF
    JUMP ISR
```

Для реализации рассмотренного генератора в оригинальном виде следует воспользоваться командным файлом install_frequency_generator.bat, который позволяет автоматически в пакетном режиме выполнить загрузку конфигурационной последовательности, содержащейся в файле frequency_generator.bit, в ПЛИС инструментального модуля Xilinx Spartan-3E Starter Board. Процесс загрузки указанной последовательности в кристалл осуществляется с помощью USB-кабеля, входящего в состав инструментального комплекта Spartan-3E Starter Kit [1], и встроенной схемы загрузочного кабеля фирмы Xilinx, предназначенного для конфигурирования ПЛИС и программирования ППЗУ. Наличие исходного текста микропрограммы предоставляет пользователю возможность внесения необходимых изменений в алгоритм функционирования рассмотренного генератора, а также модификации значений параметров в соответствии с условиями решаемой задачи. После коррекции ассемблерного текста микропроцессорной программы необходимо повторить процесс его трансляции. Чтобы воспользоваться загрузчиком JTAG Program Loader, необходимо при трансляции исходного текста использовать модифицированные варианты шаблонов описания содержимого ППЗУ JTAG_Loader_ROM_form.vhd и JTAG_Loader_ROM_form.v [2, 7]. Полученный в результате трансляции код микропрограммы может быть записан непосредственно в программную память ядра PicoBlaze через порт JTAG-интерфейса ПЛИС инструментального модуля с помощью загрузчика JTAG Program Loader и того же кабеля, который использовался для загрузки конфигурационной последовательности, содержащейся в файле frequency_generator.bit.

Литература

1. Новый инструментальный комплект Spartan-3E Starter Kit для практического освоения методов проектирования встраиваемых микропроцессорных систем на основе ПЛИС семейства FPGA фирмы Xilinx // Компоненты и технологии. 2003. № 10.
2. Зотов В. Проектирование встраиваемых микропроцессорных систем на основе ПЛИС фирмы Xilinx. М.: Горячая линия — Телеком, 2006.
3. Зотов В. PicoBlaze — семейство восьмиразрядных микропроцессорных ядер, реализуемых на основе ПЛИС фирмы Xilinx // Компоненты и технологии. 2003. № 4.
4. Зотов В. Система команд микропроцессорного ядра PicoBlaze, реализуемого на основе ПЛИС семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E // Компоненты и технологии. 2003. № 5.
5. Зотов В. Особенности микропроцессорного ядра PicoBlaze, предназначенного для применения в проектах, реализуемых на основе ПЛИС семейства Virtex-II // Компоненты и технологии. 2003. № 6.
6. Зотов В. Особенности микропроцессорного ядра PicoBlaze, предназначенного для применения в проектах, реализуемых на основе ПЛИС семейства CoolRunner-II // Компоненты и технологии. 2003. № 7.
7. Зотов В. Особенности микропроцессорного ядра PicoBlaze, предназначенного для применения в проектах, реализуемых на основе ПЛИС семейств Spartan-3, Virtex-II и Virtex-IIPRO // Компоненты и технологии. 2005. № 5–6.
8. Зотов В. MicroBlaze — семейство 32-разрядных микропроцессорных ядер, реализуемых на основе ПЛИС фирмы Xilinx // Компоненты и технологии. 2003. № 9.
9. Зотов В. Система команд микропроцессорного ядра MicroBlaze // Компоненты и технологии. 2004. № 1–3.
10. Зотов В. Организация памяти микропроцессорного ядра MicroBlaze // Компоненты и технологии. 2004. № 5.
11. Бибило П. Н. Основы языка VHDL. М.: Солон-Р, 2000.
12. Бибило П. Н. Синтез логических схем с использованием языка VHDL. М.: Солон-Р, 2002.
13. Уэйкерли Дж. Ф. Проектирование цифровых устройств. Том 1. М.: Постмаркет, 2002.