Программная часть

Вам дана папка pulpino_edu. В папке tmp находится готовый проект Vivado, с которым вы можете начать работать. Вам необходимо добавить в проект ваш модуль-обертку вместе с кузнечиком, в соответствии с инструкцией в разделе “аппаратная часть”.

В папке sw находятся файлы, которые необходимы для сборки вашей программы. В подпапке inc находятся заголовочные файлы, связанные с системой на кристалле, в которую вы интегрируете ваш модуль. Можете игнорировать эти файлы, они подключаются без ваших действий. В этой папке вам будет необходимо разместить заголовочные файлы библиотеки вашего модуля.

Если вы хотите использовать в вашей библиотеке динамическое выделение памяти (использовать malloc), вам будет необходимо подключить заголовочный файл `tlsf_malloc.h`.

В подпапке src есть папка startup и несколько файлов с расширением ".c". Как и с заголовочными файлами, вы можете их игнорировать. Эти файлы содержат код для другой периферии СнК (таймеры, gpio), код обработчика прерываний, и код, предваряющий запуск основной программы.

В корне папки sw находится файл Makefile, который вам будет необходимо дописать после того, как вы добавите в папки inc и src вашу библиотеку, а так же напишите основную программу в отдельном файле папки src.

В основной программе вам необходимо выполнить шифрование посредством кузнечика, добавив в конец программы бесконечный цикл.

После того как вы все сделаете, вам будет необходимо запустить скрипт compile.sh из bash-терминала (на windows для этого используется программа git bash, если ее у вас нет — поставьте себе наконец git, вы же разработчики как-никак). Машина, на которой будет производиться запуск должна находиться в сети МИЭТ (компьютер либо подключен к vpn, либо находится в общажной сети).

Запускать make самостоятельно вам не нужно, поскольку у вас нет кросскомпилятора.

После выполнения скрипта compile.sh, в папке sw появятся два файла:

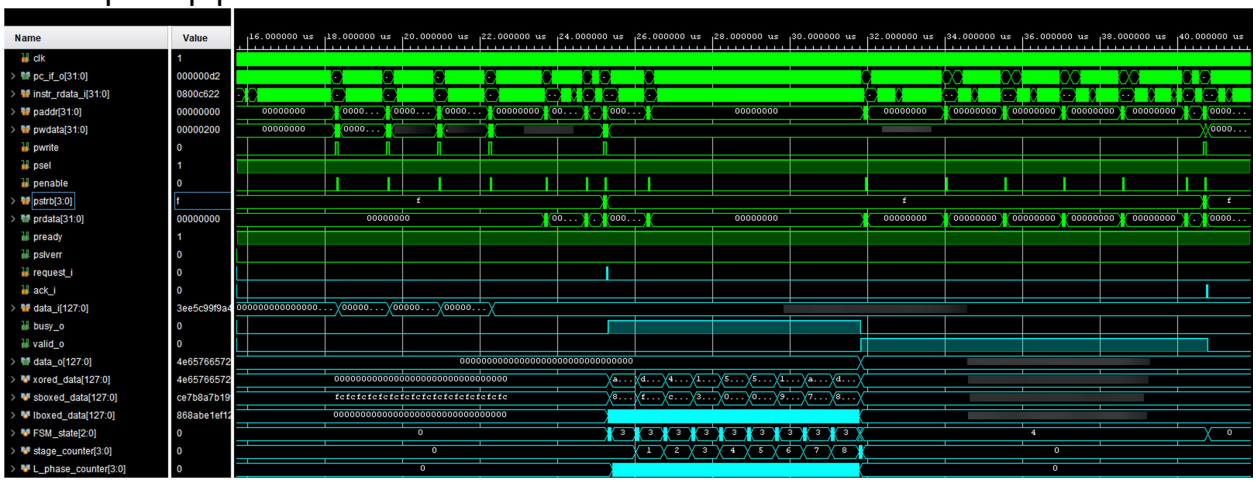
- `pulpino_data.dat`
- `pulpino_text.dat`

С этого момента, вы можете запускать симуляцию — СнК подхватит эти файлы и будет использовать в качестве исполняемой программы.

В проект уже добавлен файл временной диаграммы, который выдаст сигналы программного счетчика, исполняемой инструкции, а также интерфейса apb для вашего периферийного устройства. Вам остается вывести только ваш модуль кузнечика, который можно найти по пути:

```
    tb_pulpino->top_i->peripherals_i->apb_cipher_i->|имя экземпляра кузнечика, которое вы дали в своей обертке|
```

Пример работы СнК:



Зеленым цветом выделены сигналы интерфейса apb, бирюзовым — сигналы кузнечика. По центру видно процесс работы шифратора, по краям от него — работу с модулем из ПО по интерфейсу apb.

Аппаратная часть

Ниже будут перечислены файлы и необходимые изменения.
В листингах кода участки, которые необходимо изменить или дополнить, будут выделены конкретным цветом.

Файл: rtl/includes/apb_bus.sv

Увеличиваем **NB_MASTER** на 1 (было 9, стало 10).

```
15: `include "config.sv"
16:
17: // SOC PERIPHERALS APB BUS PARAMETRES
18: `define NB_MASTER  10
19:
20: // MASTER PORT TO CVP
21: `define UART_START_ADDR       32'h1A10_0000
```

Определяем **адресное пространство** устройства через **define**.

```
52: // MASTER PORT TO DEBUG
53: `define DEBUG_START_ADDR      32'h1A11_0000
54: `define DEBUG_END_ADDR        32'h1A11_7FFF
55:
56: // MASTER PORT TO CIPHER
57: `define CIPHER_START_ADDR      32'h1A11_8000
58: `define CIPHER_END_ADDR        32'h1A11_8FFF
59:
60: `define APB_ASSIGN_SLAVE(lhs, rhs)     \
61:     assign lhs.paddr    = rhs.paddr;   \
62:     assign lhs.pwdata   = rhs.pwdata;  \
63:     assign lhs.pwrite   = rhs.pwrite;  \
```

Файл: rtl/periph_bus_wrap.sv

Объявляем **APB-шину устройства.**

```
16:     parameter APB_DATA_WIDTH = 32
17:     )
18:    (
19:     input logic        clk_i,
20:     input logic        rst_ni,
21:
22:     APB_BUS.Slave      apb_slave,
23:
24:     APB_BUS.Master     uart_master,
25:     APB_BUS.Master     gpio_master,
26:     APB_BUS.Master     spi_master,
27:     APB_BUS.Master     timer_master,
28:     APB_BUS.Master     event_unit_master,
29:     APB_BUS.Master     i2c_master,
30:     APB_BUS.Master     fll_master,
31:     APB_BUS.Master     soc_ctrl_master,
32:     APB_BUS.Master     debug_master,
33:     APB_BUS.Master     cipher_master
34:
35:     );
36:
37:   localparam NB_MASTER      = `NB_MASTER;
```

Подключаем стартовый и конечный адрес устройства к шине.

```
90:   `APB_ASSIGN_MASTER(s_masters[8], debug_master);
91:   assign s_start_addr[8] = `DEBUG_START_ADDR;
92:   assign s_end_addr[8]   = `DEBUG_END_ADDR;
93:
94:   `APB_ASSIGN_MASTER(s_masters[9], cipher_master);
95:   assign s_start_addr[9] = `CIPHER_START_ADDR;
96:   assign s_end_addr[9]   = `CIPHER_END_ADDR;
```

Файл: rtl/peripherals.sv

Увеличиваем **APB_NUM_SLAVES** на 1 (было 8, стало 9).

```
100:     output logic        [31:0] boot_addr_o
101:   );
102:
103:   localparam APB_ADDR_WIDTH  = 32;
104:   localparam APB_NUM_SLAVES  = 9;
105:
106:   APB_BUS s_apb_bus();
107:
108:   APB_BUS s_uart_bus();
```

Объявляем **APB-шину устройства.**

```
111:   APB_BUS s_timer_bus();
112:   APB_BUS s_event_unit_bus();
113:   APB_BUS s_i2c_bus();
114:   APB_BUS s_fll_bus();
115:   APB_BUS s_soc_ctrl_bus();
116:   APB_BUS s_debug_bus();
117:   APB_BUS s_cipher_bus();
118:
119:   logic [1:0]   s_spim_event;
120:   logic [3:0]   timer_irq;
```

Подключаем шину к экземпляру модуля periph_bus_wrap.

```
226:     .spi_master        ( s_spi_bus        ),
227:     .timer_master      ( s_timer_bus      ),
228:     .event_unit_master ( s_event_unit_bus ),
229:     .i2c_master        ( s_i2c_bus        ),
230:     .fll_master        ( s_fll_bus        ),
231:     .soc_ctrl_master   ( s_soc_ctrl_bus   ),
232:     .debug_master      ( s_debug_bus      ),
233:     .cipher_master     ( s_cipher_bus     )
234:   );
```

Создаем экземпляр модуля.

```
517:     .per_master_add_o     ( debug.addr              ),
518:     .per_master_we_o      ( debug.we                ),
519:     .per_master_wdata_o   ( debug.wdata             ),
520:     .per_master_be_o      (                         ),
521:     .per_master_gnt_i     ( debug.gnt               ),
522:
523:     .per_master_r_valid_i ( debug.rvalid            ),
524:     .per_master_r_opc_i   ( '0                      ),
525:     .per_master_r_rdata_i ( debug.rdata             )
526:   );
527:
528:   kuznechik_cipher_apb_wrapper #(
529:     .APB_ADDR_SPACE_SIZE_BYTES(512)
530:   )
531:     apb_cipher_i
532:   (
533:     .pclk_i     ( clk_int[8]                ),
534:     .presetn_i  ( rst_n                     ),
535:     .paddr_i    ( s_cipher_bus.paddr[31:0] ),
536:     .psel_i     ( s_cipher_bus.psel         ),
537:     .penable_i  ( s_cipher_bus.penable      ),
538:     .pwrite_i   ( s_cipher_bus.pwrite       ),
539:     .pwdata_i   ( s_cipher_bus.pwdata       ),
540:     .pstrb_i    ( s_cipher_bus.pstrb        ),
541:     .pready_o   ( s_cipher_bus.pready       ),
542:     .prdata_o   ( s_cipher_bus.prdata       ),
543:     .pslverr_o  ( s_cipher_bus.pslverr      )
544:   );
545:
546: endmodule
```