электронный научно-технический журнал
ИНЖЕНЕРНЫЙ ВЕСТНИК
издатель ФГБОУ ВПО «Московский государственный технический университет им. Н.Э. Баумана»

Проектирование и моделирование цифровых модулей в системах на кристалле

77-48211/618623

09, сентябрь 2013

Хабаров А. П., Хартов В. Я.

УДК 004.9

Россия, МГТУ им. Н.Э. Баумана
mig-foxhound@yandex.ru
khartovbmstu@gmail.com

В процессе проектирования системы на кристалле нередко возникает задача проектирования нестандартных цифровых модулей для включения их в библиотеку компонентов используемой системы. Удобным и эффективным инструментом для решения подобных задач является язык Verilog.

Как правило, при отсутствии в библиотеке нужных компонентов и создании новых компонентов необходимо выполнить общую для всех последовательность действий. Например, в среде проектирования Creator, используемой при разработке систем на базе PSOC 3, PSOC 5 (Cypress), нужно выполнить следующие операции:

- открыть вкладку Components в окне Creator;
- щелкнув правой кнопкой мыши по имени проекта, выбрать Add Component Item;
- ввести имя компонента в поле Component Name и выбрать тип Symbol Wizard для создания условного графического обозначения (УГО) компонента;
- в появившемся окне описать все входы и выходы компонента, указывая их тип (аналоговый или цифровой) и направление (ввод-вывод);
- щелкнув правой кнопкой мыши на свободном поле окна редактора УГО выбрать Generate Verilog, задать имя файла с реализацией (по умолчанию совпадает с именем компонента);
- отредактировать полученный файл;
- сохранить все изменения;
- нажать кнопку обновления Refresh в окне библиотеки. Новый компонент появится на вкладке Default.

Рассмотрим ряд примеров по созданию новых компонентов цифровых модулей и их применению в проектах.

1. Модуль комбинационного типа

Матричный умножитель представляет собой комбинационную схему, реализующую функцию умножения двух двоичных чисел. Основным достоинством комбинационного умножителя является быстрота выполнения умножения. Однако его применение ограничено тем, что для его реализации необходимо использовать большое количество логических элементов.

Пример реализации модуля комбинационного умножителя 4-разрядных двоичных чисел приведен на рис. 1. Данный блок имеет два 4-разрядных входа a[3:0] и b[3:0], на которые поступают входные операнды множимого и множителя, и 8-разрядный выход c [7:0] произведения.

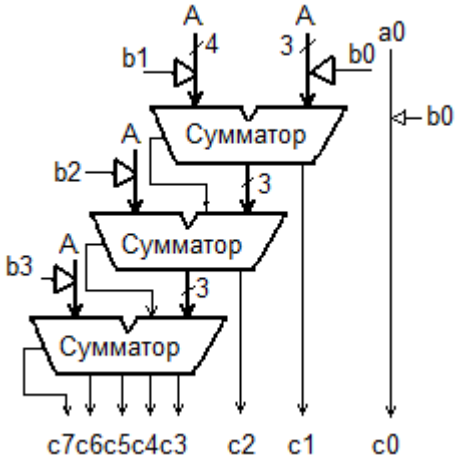


Рис. 1. Матричный умножитель

Продемонстрируем на этом примере возможности создания и повторного использования пользовательских модулей в среде PSoC Creator.

Логика работы модуля, представленная на языке Verilog:

```
`include "cypress.v"
// Компонент: mul_comb
module mul_comb (c,a,b);
     output [7:0] c;
     input  [3:0] a;
     input  [3:0] b;
     wire [7:0]c1;
     wire [7:0]c2;
     wire [7:0]c3;
     wire [7:0]c4;
     wire [7:0]a0;
```

```
//{0,a[3]&b[0],a[2]&b[0],a[1]&b[0],a[0]&b[0]}+
assign a0[3:0]= a;
assign a0[7:4]=4'h0;
assign c1=b[0]?a0:8'h00;
assign c2=b[1]?(a0<<1):8'h00;
assign c3=b[2]?(a0<<2):8'h00;
assign c4=b[3]?(a0<<3):8'h00;
assign c = c1+c2+c3+c4;
 endmodule
```

Схема проекта представлена на рис. 2. Для программного задания входных операндов A, B используем два регистра Control Reg, результат умножения mul_comb_1 выведем на линейку светодиодов. Для этого на вкладке .cydwr/Pins конфигурируются портовые выводы микросхемы: LSB [3:0] – Px[3:0], MSB {3:0] –Px[3:0] с учетом схемы включения светодиодов на плате.

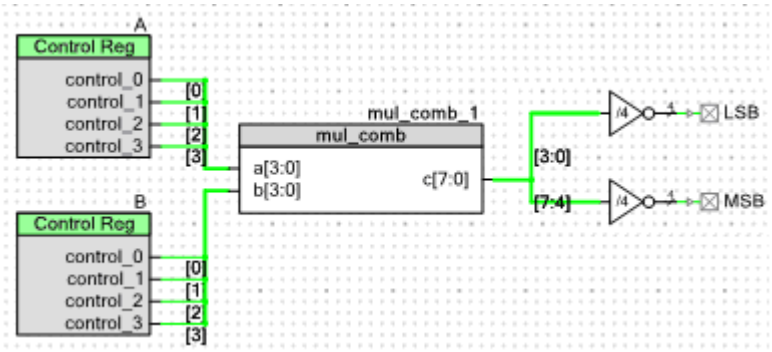


Рис. 2. Схема проекта с комбинационным умножителем

Код программы в файле main.c, осуществляющий взаимодействие процессора системы на кристалле с цифровыми модулями, реализованными на элементах кристалла UDB:

```
#include <device.h>
#include <A.h>
#include <B.h>
void main()
{    A_Write(0xC);
     B_Write(0xD);
     for(;;){  }   }
```

После выполнения компиляции из отчета по результатам синтеза следует, что для реализации проекта на кристалле PSOC 5 потребовалось кроме двух стандартных регистров управления, 11 линий ввода-вывода (15.28%), 48 макроячеек (Macrocells-25%) ресурса кристалла), 57 термов (Unique Pterms–14.84%).

2. Цифровой автомат Мили

Синтез цифрового автомата Мили рассмотрим на примере накапливающего умножителя, который осуществляет накопление (суммирование) частичных произведений. При этом один из операндов (множитель) может быть задан в последовательном коде. Для перемножения *n*-разрядных чисел требуется *n* тактов. На каждом такте осуществляется сложение результата, полученного на предыдущем шаге, с множимым, умноженным на очередной разряд множителя.

В отличие от матричного умножителя комбинационного типа количество элементов, требуемых для реализации последовательного умножителя, мало зависит от разрядности операндов.

Условное графическое обозначение модуля последовательного 8-разрядного умножителя представлено на рис. 3, назначение выводов в табл.1.

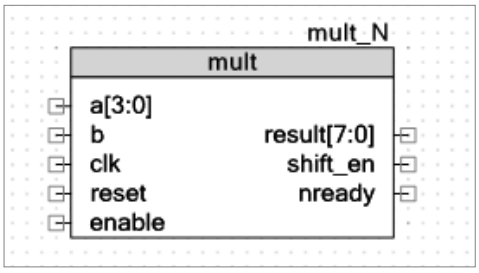


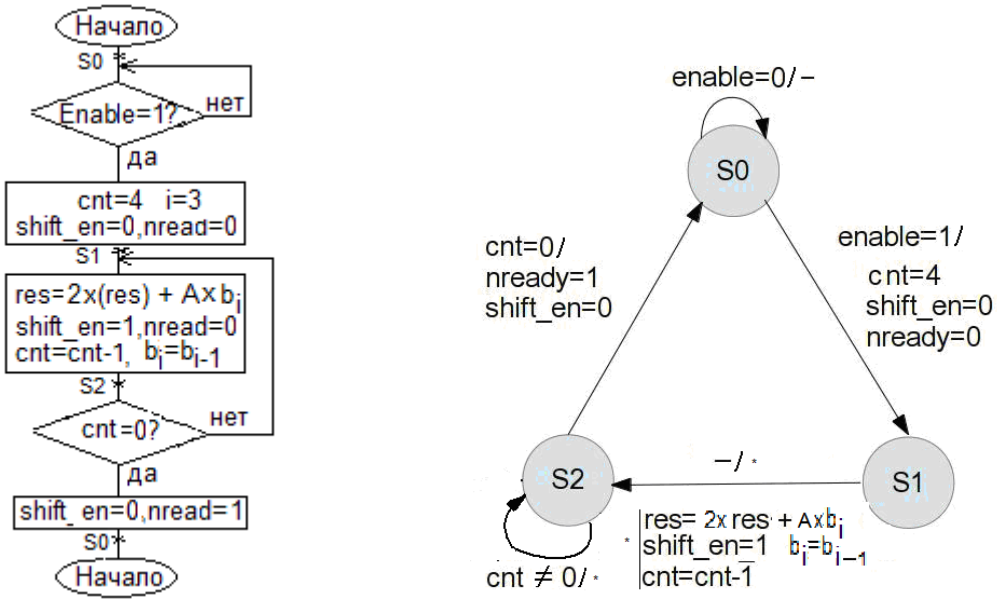
Рис. 3. Графическое обозначение последовательного умножителя

Таблица 1. Входы-выходы последовательного умножителя и их назначение

Входы-выходы	Направление	Назначение
a[7:0]	вход	Множимое (параллельный 8-разрядный код)
b	вход	Бит множителя, поступающий на вход умножителя
clk	вход	Синхронизация
reset	вход	Сброс
enable	вход	Разрешение операции умножения
result[15:0]	выход	Результат (параллельный код)
shift_en	выход	Разрешение на сдвиг множителя $b_{7...}b_1b_0$ во внешнем регистре для вывода очередного бита bi
nready	выход	Флаг готовности результата: 0 – не готов, 1 – готов

Помимо основной операции формирования 8-разрядного результата умножения (result[7:0]) на модуль возложена управляющая функция по формированию разрешающего сигнала сдвига множителя (shift_en) в регистре множителя и контроля числа выполненных

итераций умножения с помощью счетчика (cnt). Алгоритм выполнения операции умножения, начиная со старшего разряда множителя, и соответствующий ему граф переходов типа Мили с тремя состояниями S0, S1, S2 представлены на рис. 4. Следует иметь в виду, что запись промежуточного результата (res) при вычислении произведения и сдвиг влево содержимого регистра множителя ($B=b_7 \ldots b_1 b_0$) для вывода очередного бита множителя b_i на умножитель происходят синхронно, по фронту входного синхросигнала clk. Одновременно с записью результата на выходе умножителя формируется сигнал разрешения сдвига (shift_en), действие которого сказывается на следующей итерации умножения.



а) б)

Рис. 4. Схема алгоритма умножения (а) и граф переходов умножителя (б)

Исходный код компонента mult на языке Verilog:

```
`include "cypress.v"
// Component: mult
module mult (result, shift_en, nready, a, b, clk, enable, reset);
     output wire [15:0] result;
     output reg shift_en;
     input  [7:0] a;
     input   b;
     input   clk;
     input   enable;
     input   reset;
     output reg nready;
//
     reg [15:0] res;
```

```
     assign result = res;
     reg [7:0] mux_out;
     reg [1:0] state;
     reg [3:0] cnt;
//assign result = {2'b00,shift_en,cnt,state};
 always @(a or b)
begin
     if (b==1'b1)
           mux_out = a;
     else
           mux_out = 8'h00;
end
always @(posedge clk or posedge reset)
begin
     if (reset)
           begin
              state <= 2'b00;
              shift_en <= 1'b0;
              res <= 16'h0000;
              nready <= 1'b0;
           end
     else if ({state,enable} == 3'b001)
           begin
              state <= 2'b01;
              cnt <= 4'h8;
              shift_en <= 1'b0;
              nready <= 1'b0;
           end
     else if ((state == 2'b01))
           begin
              state <= 2'b10;
              cnt <= cnt-1;
              shift_en<=1;
              res <= (res << 1)+{8'h00, mux_out};
              nready <= 1'b0;
           end
     else if (state[1]&(!state[0])&(cnt[3]|cnt[2]|cnt[1]|cnt[0]) == 1'b1)
              //({state == 2'b10) and (cnt != 0'h0))
           begin
              cnt <= cnt-1;
              shift_en <= 1;
              res <= (res << 1)+{8'h00, mux_out};
              nready <= 1'b0;
```

```
            end
        else if({state,cnt} == 6'b100000)
            begin
               state <= 2'b00;
               shift_en <=0;
               nready <= 1'b1;
            end
    end
    endmodule
```

Схема проекта последовательного умножителя для перемножения 8-разрядных операндов приведена на рис. 5.

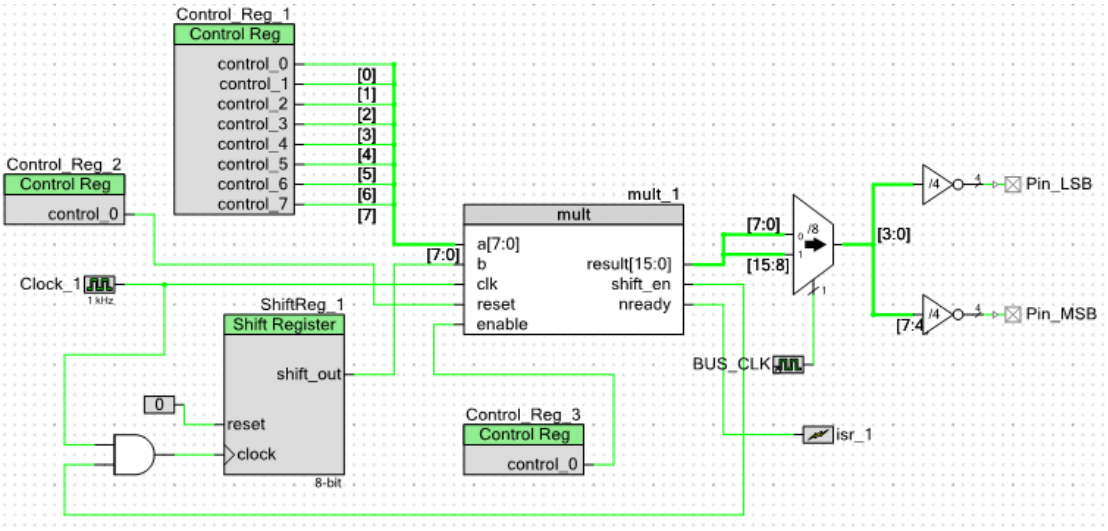


Рис. 5. Модель устройства для последовательного умножения

Для взаимодействия с компонентом, реализующим последовательный умножитель, применяются регистры управления Control_Reg_1, Control_Reg_2 и Control_Reg_3. Для преобразования множителя из параллельного кода в последовательный применяется регистр сдвига ShiftReg_1. Результаты умножения result[15:0] выводятся на линейку светодиодов. По завершении умножения генерируется сигнал прерывания isr_1.

Присоединяем контакты ввода-вывода: Pin_LSB[3:0] – Px[3:0], Pin_MSB[3:0] – Px[3:0].

Исходный код программы, осуществляющий взаимодействие со схемными компонентами, приведен ниже. Множитель и множимое задаются в программном коде. Как видно из текста программы, для взаимодействия с компонентами использованы сгенерированные автоматически библиотечные функции.

```
#include <device.h>
#include <Control_Reg_1.h>
#include <Control_Reg_2.h>
#include <Control_Reg_3.h>
```

```
#include <ShiftReg_1.h>
#include <isr_1.h>
void main()
{  CyGlobalIntEnable;
   isr_1_Start();
   Control_Reg_1_Write(0xff);
   Control_Reg_2_Write(1u);
   Control_Reg_2_Write(0u);
   Control_Reg_3_Write(1u);
   ShiftReg_1_Start();
   ShiftReg_1_WriteRegValue(0xffu);
   for(;;)
   {  uint8 a = ShiftReg_1_ReadRegValue(); }
}
```

Код обработчика прерывания isr_1.c запрещает работу умножителя по окончанию умножения путем записи нуля в регистр управления:

```
CY_ISR(isr_1_Interrupt)
{ Control_Reg_3_Write(0u); }
```

Для реализации проекта на кристалле PSOC 5 потребовалось 11 контактов ввода-вывода, 36 макроячеек из 192 (18%), 55 из 384 термов (14%), одна ячейка DataPath из 24 (4%), одна ячейка состояния (4%), четыре ячейки управления (16%) и одна ячейка синхронизации из 92 (1%), одно прерывание из 32 (3%). После программирования кристалла наблюдаем на линейке светодиодов формируемые суммы частичных произведений и само произведение.

3. Операционное устройство

Третий пример демонстрирует проект совместного использования умножителя, блока дисплея для отображения результата операции и канала USB для связи с терминалом. Аппаратным аналогом модуля умножителя (C=A×B) может служить схема с накоплением частичных произведений при перемножении двух 8-разрядных сомножителей, начиная с младших разрядов множителя B, приведенная на рис. 6.

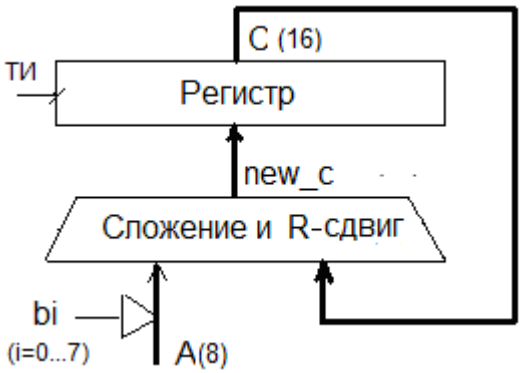


Рис. 6. Схема умножителя

Граф переходов умножителя представлен на рис. 7. При поступлении сигнала start выполняется прием сомножителей, обнуление счетчика итераций и сброс сигнала готовности ready. Выполнение операции начинается после перевода сигнала start в нулевое состояние, при этом на всех последующих итерациях процедуры умножения выполняется получение новой суммы частичных произведений и запись накопленной суммы в регистр результата со сдвигом вправо (состояние S2). После выполнения всех итераций вырабатывается сигнал готовности и устройство возвращается в начальное состояние (S0).

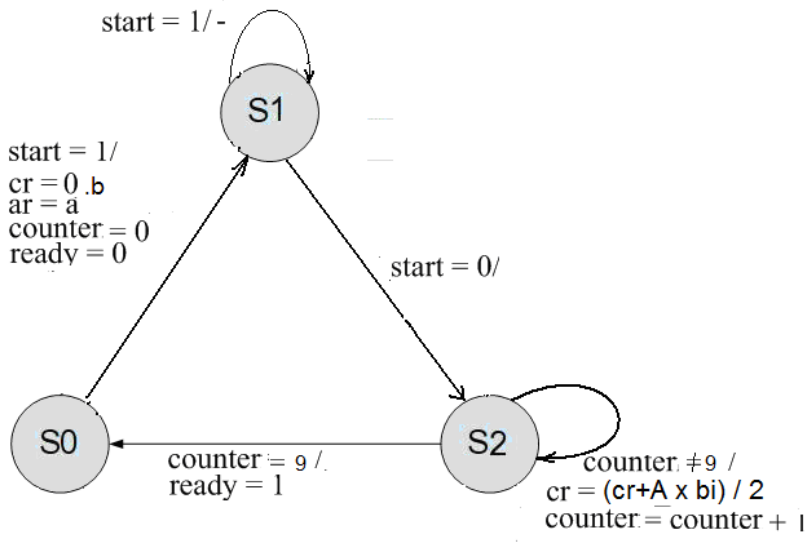


Рис. 7. Граф переходов умножителя

Соответствующее описание модуля умножителя на языке Verilog:

```
`include "cypress.v"
// Component: MultiplierUS
module MultiplierUS
(     output [15:0] c,
      output  ready,
      input  [7:0] a,
      input  [7:0] b,
      input   clk,
      input   start
);
reg [7:0] ar;
reg [16:0] cr;
reg [3:0] counter;
wire [16:0] new_cr;

always @(posedge clk)
begin
```

```
        if (start)
              begin
                 ar = a;
                 cr = {9'b000000000, b};
                 counter = 0;
              end
        else
              begin
                 if (counter != 4'b1001)
                    begin
                       cr = new_cr;
                       counter = counter + 1;
                    end
              end
end

assign new_cr[16:8] = (cr[0])?({1'b0, cr[16:9]} + {1'b0, ar}):{1'b0, cr[16:9]};
assign new_cr[7:0] = cr[8:1];
assign ready = counter[3] & counter[0];
assign c = cr[15:0];
endmodule
```

Для моделирования умножителя собираем схему на рис. 8, предусмотрев вывод результатов операции на ЖК-дисплей и обмен данными с терминалом по каналу USB.

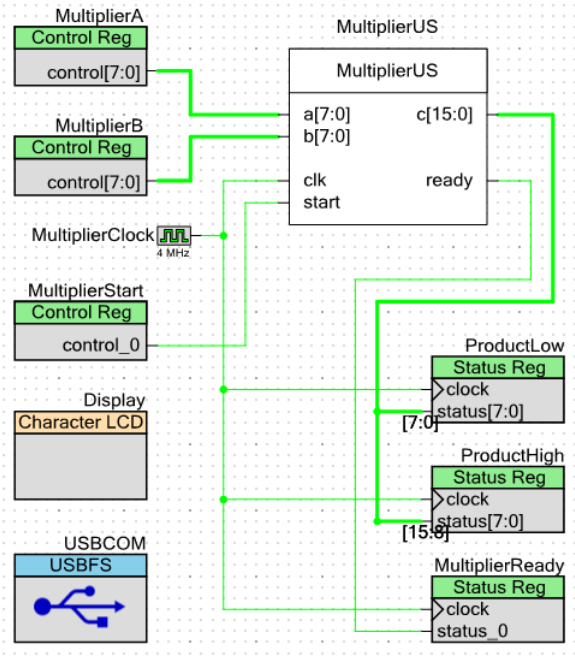


Рис. 8. Моделирование операционного устройства

Далее приведены тексты формируемых программных модулей.

Packets.h:

```
#ifndef _Packets_h_
#define _Packets_h_
#include <cytypes.h>
#define PACKET_TYPE_SERIAL_MUL 0
#define PACKET_TYPE_SERIAL_MUL_RES 1
typedef struct
{     uint8 Type;   // Тип пакета данных (всегда PACKET_TYPE_SERIAL_MUL)
      uint8 A;     // Множимое A
      uint8 B;     // Множитель B
} CPacketSerialMul;     // Пакет данных для пересылки операндов
                        // из компьютера в PSoC
typedef struct
{     uint8 Type;  // Тип пакета данных (всегда PACKET_TYPE_SERIAL_MUL_RES)
      uint16 Product;  // Результат умножения
} CPacketSerialMulRes; // Пакет для пересылки результата в компьютер
#endif
```

PacketProcessing.h:

```
#ifndef _PacketProcessing_h_
#define _PacketProcessing_h_
#include <device.h>
#include "Packets.h"
void ProcessPacket(uint8 *Packet);
void ProcessSerialMul(CPacketSerialMul *Packet);
void SendPacket(uint8 *Packet, uint8 Size);
#endif
```

main.c:

```
#include <device.h>
#include "PacketProcessing.h"
uint8 Packet[64];
void main()
{   CyGlobalIntEnable; // Разрешение прерывания
    Display_Start();   // Инициализация LCD-дисплея
    USBCOM_Start(0, USBCOM_3V_OPERATION); // Инициализация USB
    while(!USBCOM_GetConfiguration());
    USBCOM_CDC_Init();
```

```
    while(1)
    {       if (USBCOM_DataIsReady())     // Если получены данные по USB, чтение
            {     if (USBCOM_GetAll(Packet) > 0)
                  {ProcessPacket(Packet); // и обработка данных
                  }
            }
    }
}
```

PacketProcessing.c:

```
#include <PacketProcessing.h>
void ProcessPacket(uint8 *Packet)
{     switch (*Packet)// Определение типа пакета данных
      {     case PACKET_TYPE_SERIAL_MUL:
            // Если тип пакета PACKET_TYPE_SERIAL_MUL,
            // то передача пакета процедуре умножения ProcessSerialMul
            ProcessSerialMul((CPacketSerialMul *) Packet);
            break;
      }
}
void ProcessSerialMul(CPacketSerialMul *Packet)
{     CPacketSerialMulRes MulRes;
      uint16 Product;
      // Запись исходных операндов в регистры
      MultiplierA_Write(Packet->A);
      MultiplierB_Write(Packet->B);
      MultiplierStart_Write(1);           // Передача операндов в PSoC
      while(MultiplierReady_Read());      // Ожидание завершения передачи
      MultiplierStart_Write(0);           // Запуск умножения
      while(!MultiplierReady_Read());     // Ожидание завершения умножение
      //Формирование пакета для передачи в компьютер
      MulRes.Type = PACKET_TYPE_SERIAL_MUL_RES;
      // Чтение младшего байта произведения
      ((uint8 *) &MulRes.Product)[0] = ProductLow_Read();
      // Чтение старшего байта произведения
      ((uint8 *) &MulRes.Product)[1] = ProductHigh_Read();
      // Запись в Product результата умножения, изменив порядок байтов
      ((uint8 *) &Product)[1] = ((uint8 *) &MulRes.Product)[0];
      ((uint8 *) &Product)[0] = ((uint8 *) &MulRes.Product)[1];
      Display_ClearDisplay();         // Очистка дисплея
      Display_PrintNumber(Packet->A); // Вывод на дисплей множимого
      Display_PrintString(" * ");
```

```
        Display_PrintNumber(Packet->B); // Вывод на дисплей множителя
        Display_PrintString(" =");
        Display_Position(1, 0);         // Переход на новую строку
        Display_PrintNumber(Product);   // Вывод произведения
        // Передача результата операции в компьютер
        SendPacket((uint8 *) &MulRes, sizeof(CPacketSerialMulRes));
}
void SendPacket(uint8 *Packet, uint8 Size)
{       // Ожидание готовности интерфейса USB
        while (USBCOM_CDCIsReady() == 0);
        // Передача данных в компьютер
        USBCOM_PutData(Packet, Size);
}
```

Для реализации проекта на кристалле PSOC 3 потребовалось 12 контактов ввода-вывода из 72 (16%), 39 макроячеек из 192 (20%), 129 термов из 384 (33%), три ячейки регистра состояния из 24 (12%), три ячейки регистра управления из 24 (12%), 8 прерываний, фиксированный блок USB. Общий объем использованных ресурсов матрицы PLD составил 22 блока из 48 (46%).

Резюмируя все вышесказанное можно отметить, что наличие программируемой матрицы в составе микроконтроллеров позволяет расширить процессорные функции за счет дополнительно синтезируемых аппаратных модулей, взаимодействующих с процессором. Эти модули создаются достаточно удобными языковыми средствами Verilog путем описания их функций и последующей компиляции составленной схемы взаимодействия разработанного модуля (проекта) совместно с основным файлом программы. Описанная методика проверена на примере проектирования цифровых модулей комбинационного и последовательностного типа (автоматов Мили). Моделирование результатов синтеза проведенное на базе современных перспективных систем на кристалле PSoC 3, PSoC 5, имеющих поддержку в виде интегрированной среды проектирования и высокоуровневого программирования с помощью API-функций, показало высокую технологичность элементной базы и возможность ее применения для разнообразных приложений.

Список использованных источников

1. PSoC3: CY8C38 Family datasheet. Режим доступа: http://www.cypresss.com (дата обращения 01.06.2012).

2. PSoC5: CY8C55 Family datasheet. Режим доступа: http://www.cypresss.com (дата обращения 01.06.2012).

3.PSoC3, PSoC5 Architecture TRM (Technical Reference Manual). Режим доступа: http://www.cypresss.com (дата обращения 01.06.2012).

4. PSoC5 Firtst Touch Starter Kit Guide. Режим доступа: http://www.cypresss.com (дата обращения 01.06.2012).

5. By Robert Ashby. My First Five PSoC 3 Desings. Режим доступа: http://www.cypress.com. (дата обращения 01.06.2012).

6. PSoC®. Development Kit Guide. Режим доступа: http://www.cypresss.com (дата обращения 01.06.2012).

7. Ермохина Н.И., Ибрагимов С.В., Хартов В.Я. Конфигурирование, настройка и программирование модулей системы на кристалле PSoC // Электронный научно-технический журнал «Инженерный вестник». 2012. № 6. Режим доступа: http://engbul.bmstu.ru/doc/457900.html (дата обращения 07.06.2013).

8. Поляков А.К. Языки VHDL и VERILOG в проектировании цифровой аппаратуры на ПЛИС. М.: Издательский дом МЭИ, 2012. 221 с.

9. Килочек Д. Проектирование устройств с технологией Cypress Capsense // Компоненты и технологии. 2009. № 3 (92). С. 139-143.