Интерфейс VJTAG для отладочной платы DK-START-3C25N

Алексей Гребенников (г. Актау, Казахстан)

В статье рассмотрено использование виртуального интерфейса JTAG для обмена данными с отладочной платой посредством кабеля USB-blaster.

Введение

Отладочная плата DK-START-3C25N предназначена для изучения архитектуры ПЛИС фирмы Altera серии Cyclone III и особенностей программирования микросхем этой серии. Основными компонентами платы являются ПЛИС типа EP3C25, статическая память SSRAM, флэш-память, динамическая память DDR SDRAM, набор светодиодов и кнопок, разъём для подключения внешних периферийных устройств, а также встроенный контроллер USB-blaster. В комплект также входит скомпилированный проект, содержащий контроллеры всех устройств на плате, и интерфейсная программа для связи с платой.

После прошивки ПЛИС с помощью среды Quartus или интерфейсной программы можно проверить работоспособность всех устройств на плате: считать/записать данные во все виды памяти, включить/выключить светодиоды. Однако как файл прошивки, так и интерфейсная программа поставляются только в бинарной форме, что создаёт определённые трудности. При разработке собственных устройств нельзя использовать интерфейсную программу для связи с компьютером, поскольку неизвестен протокол обмена данными. Встроенных же на плате устройств ввода-вывода явно недостаточно для полноценного обмена информацией. Одним из вариантов решения этой проблемы является использование интерфейса VJTAG (Virtual JTAG – виртуальный JTAG), позволяющего обмениваться данными с платой через кабель USB-blaster.

Структура интерфейса

Схема обмена данными с помощью интерфейса VJTAG показана на рисунке 1. Все микросхемы серии Cyclone III имеют встроенный контроллер JTAG TAP, который позволяет программировать ПЛИС непосредственно в схеме. Сам JTAG-контроллер представляет собой конечный автомат, управляемый сигналом TMS. Данные принимаются контроллером по линии TDI, проходят через определённые сдвиговые регистры, задаваемые командой, и затем передаются на выходную линию TDO. Более подробно работа контроллера описана в [1].

Для получения доступа к основным сигналам конечного автомата контроллера JTAG TAP и обеспечения информационного обмена с внутренними устройствами ПЛИС служит мегафункция VJTAG, создаваемая в среде Quartus. Эта мегафункция также имеет в своём составе конечный автомат, аналогичный применяемому в контроллере JTAG TAP. Разница заключается в том, что сдвиговые регистры данных формируются пользователем внешне по от-

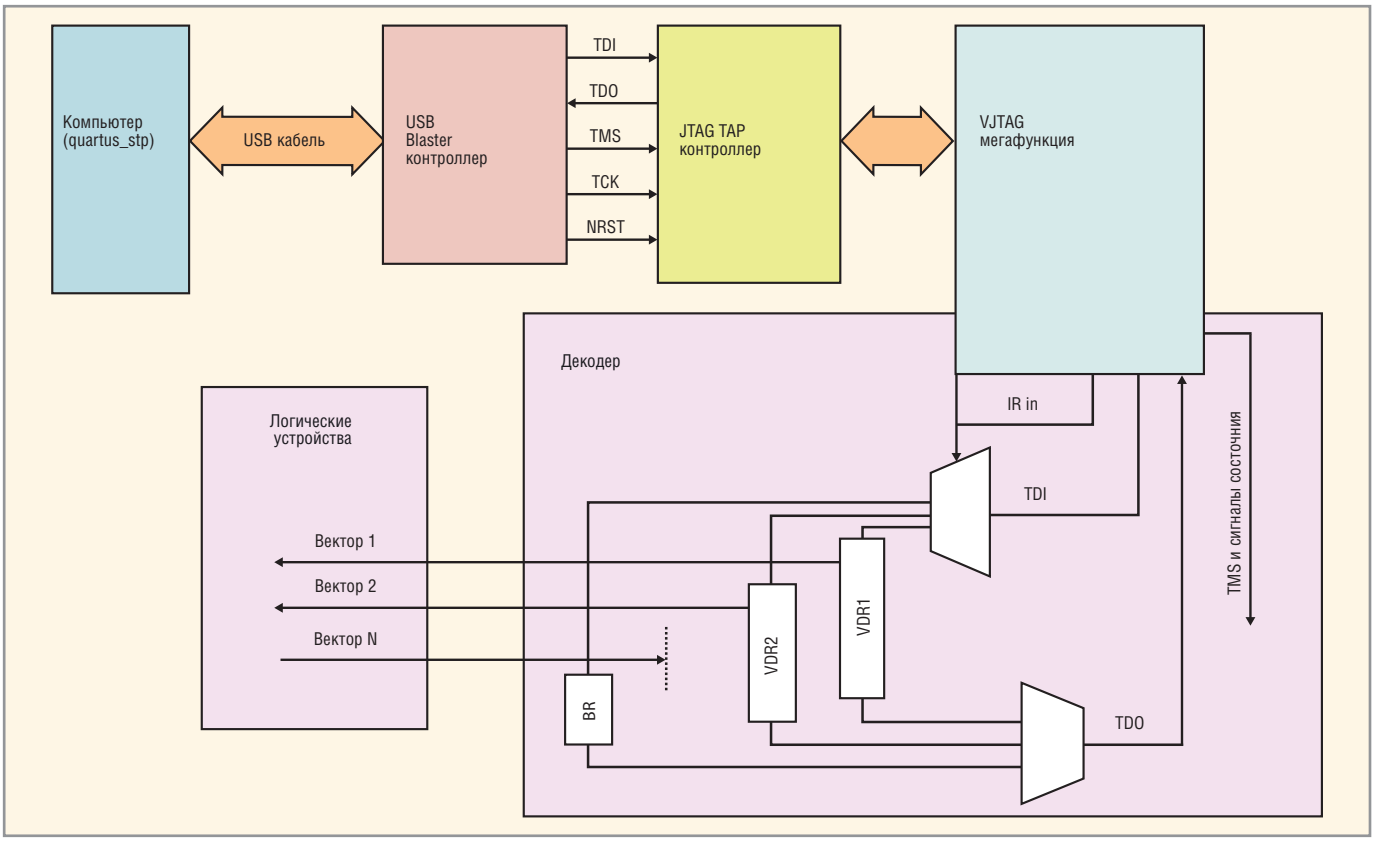

Рис. 1. Структура связи при использовании мегафункции VJTAG

ношению к мегафункции. На рисунке 1 эти регистры обозначены VDR1 (virtual data register), VDR2 и т.д. и находятся в блоке декодера. Последний является вспомогательным блоком, который декодирует команды, управляет сдвиговыми регистрами и обеспечивает коммутацию сигналов TDI и TDO.

Регистр BR (bypass register) служит для непосредственной передачи данных со входа на выход, минуя внутренние регистры данных. Данные, полученные последовательно по линии TDI, затем передаются параллельно конечным логическим устройствам, таким как контроллеры памяти, устройства управления светодиодами и т.д. Приём данных от логических устройств также происходит параллельно в регистры VDR, откуда по линии TDO они последовательно передаются управляющему компьютеру.

Управление интерфейсом VJTAG осуществляется программой *quartus_stp*, входящей в состав программного комплекса Quartus, при помощи сценариев на языке Tcl. Использование расширения Tk языка Tcl позволяет создавать графические приложения.

Таким образом, для создания интерфейса передачи данных через VJTAG требуется:

- создать мегафункцию VJTAG;
- создать логику, декодирующую команды и управляющую сигналами TDI и TDO;
- написать сценарий на языке Tcl/Tk;
- создать конечные логические устройства.

Рассмотрим каждый этап более подробно.

Создание мегафункции VJTAG

Рассматриваемый проект создавался в среде Quartus II 9.0 Web Edition, все дальнейшие ссылки на Quartus относятся именно к этой версии пакета. Архив проекта содержится в файле *vjtag_i_top.qar*, который находится в дополнительных материалах к статье на интернет-странице журнала.

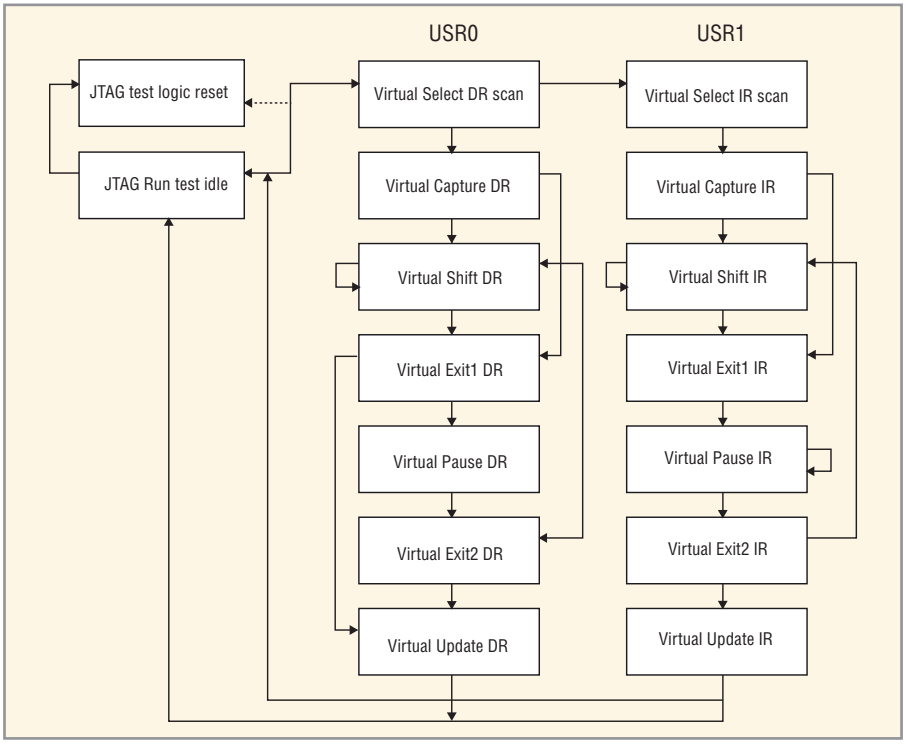

Рис. 2. Конечный автомат мегафункции VJTAG

Как уже упоминалось выше, мегафункция VJTAG содержит конечный автомат, изображённый на рисунке 2. Зелёным цветом обозначены состояния автомата, к которым имеет доступ пользователь. Цепь *USR0* работает с регистрами данных (Data Register), а цепь *USR1* задаёт команды (Instruction Register). Доступ к промежуточным состояниям конечного автомата необходим для того, чтобы оперировать регистрами данных и команд (выполнять операции чтения, сдвига и обновления).

Для включения мегафункции VJTAG в проект необходимо в окне графического построения, в данном случае в окне верхнего уровня, выбрать *Insert* → → *Symbol*. Затем в появившемся окне нажать кнопку *MegaWizard Plug-in Manager*..., выбрать *Installed Plug-Ins* → *JTAG-accessible Extensions* → *Virtual JTAG*. Далее следует ряд окон, задающих параметры мегафункции. В зависимости от числа команд, которые необходимо декодировать, выбирается ширина регистра инструкций (IR). В нашем примере регистр инструкций имеет ширину шесть бит, что позволяет декодировать 64 команды.

Помимо сигналов конечного автомата VJTAG, мегафункция также обеспечивает доступ к сигналам JTAG, однако эти дополнительные сигналы могут быть

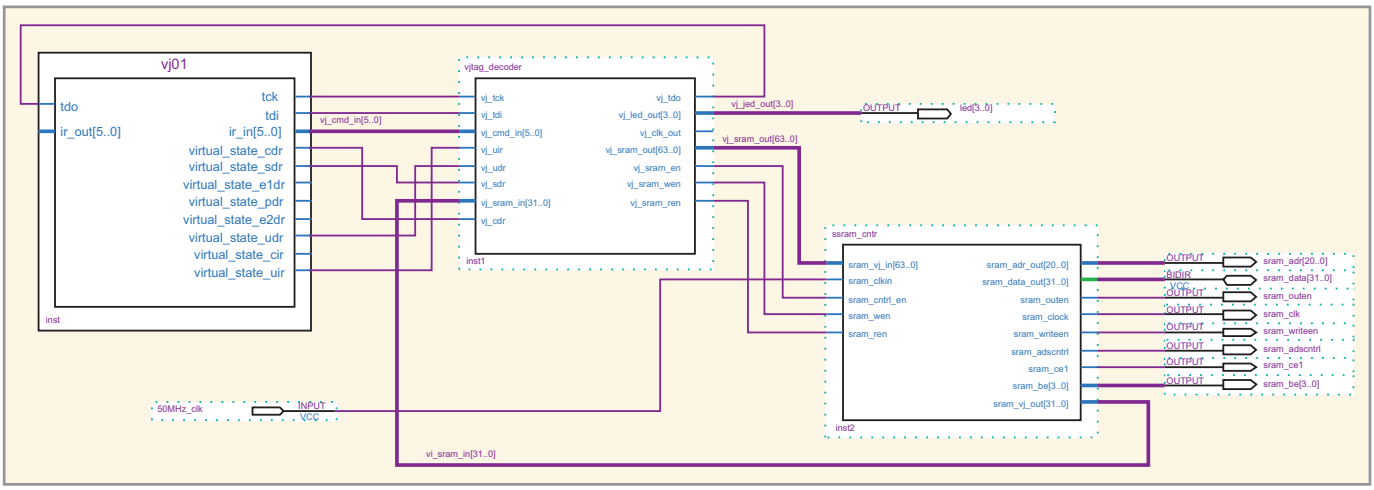

Рис. 3. Графический файл верхнего уровня проекта

использованы только для информационных целей. Вместе с графическим символом создаётся шаблон на языках Verilog или VHDL, позволяющий включать мегафункцию в файлы на этих языках. На рисунке 3 показан графический файл верхнего уровня с включенным интерфейсом VJTAG под названием *vj01*. Более подробно процесс создания мегафункции VJTAG описан в [2].

Создание декодирующей логики

Следующий шаг – создание декодирующей логики. На рисунке 3 блок декодера обозначен символом *vjtag_decoder*. Этот блок написан на языке Verilog и на обобщённой схеме (см. рис. 1) также называется декодером.

Полностью исходный код блока декодера приведён в дополнительных материалах к статье. Рассмотрим основные моменты функционирования блока. Все команды декодируются выражениями типа:

```
// Set leds 000001
wire vj_led_set = vj_cmd_in[0] &&
~vj_cmd_in[1] && ~vj_cmd_in[2] &&
        ~vj_cmd_in[3] &&
~vj_cmd_in[4] && ~vj_cmd_in[5];
```

В данном случае для активации блока контроля светодиодов необходимо установить регистр инструкций IR равным 000001b. Для загрузки регистров данных используется конструкция:

```
always @(posedge vj_tck)
begin
if (vj_sdr && vj_led_set)
vj_led_reg <= {vj_tdi,
vj_led_reg[3:1]};
.............................
end
```

Все значения фиксируются по переднему фронту тактового сигнала. Частота *vj_tck* равна тактовой частоте интерфейса JTAG. Регистр *vj_led_reg* имеет размерность четыре бита – по числу светодиодов на плате; *vj_sdr* (virtual JTAG shift data register) – это сигнал конечного автомата функции VJTAG. При установке этого сигнала в лог. 1 можно производить сдвиг регистра. Поскольку число регистров данных может быть произвольным, необходима дополнительная логика для выбора регистра. Конструкция *vj_sdr && vj_led_set* выполняет именно эту функцию. В блоке *always* используются и другие сигналы конечного автомата:

```
if (vj_cdr && vj_sram_read_start)
vj_sram_in_buf[31:0] <=
vj_sram_in[31:0];
```

Сигнал *vj_sram_read_start* – это декодированная команда, задающая начало цикла чтения памяти SSRAM. Данные из памяти должны быть параллельно загружены в сдвиговый регистр для дальнейшей последовательной передачи. Соответственно, сигнал *vj_cdr* (Capture Data Register) используется для параллельной загрузки данных, а затем по сигналу *vj_sdr* (Shift Data Register) полученные данные последовательно передаются по цепочке:

```
if (vj_sdr && vj_sram_read_start)
vj_sram_in_buf <=
{vj_tdi,vj_sram_in_buf[31:1]};
```

Заметим, что в данном случае данные на входе *vj_tdi* могут быть любыми; нас интересуют данные, подаваемые на выход *vj_tdo*. Соответственно, этот выход должен быть правильно сконфигурирован, т.е. подключен к нужному регистру. Эту задачу выполняет следующая конструкция:

```
always @(vj_led_reg[0] or
vj_sram_out[0] or vj_bypass or
vj_sram_in_buf[0])
begin
        if (vj_sram_read_start)
        vj_tdo <=
vj_sram_in_buf[0];
.....................................
end
```

Команда *vj_bypass* служит для прямой передачи данных с входа на выход, минуя внутренние регистры.

Написание сценария на языке Tcl/Tk

Следующий этап – написание сценария для компьютера, который будет управлять функцией VJTAG. Полный исходный текст сценария приведён в дополнительных материалах к статье в файле *cpanel.tcl*. Для выполнения этого сценария требуется утилита *quartus_stp.exe*, которая находится в директории *bin* пакета Quartus и работает только в режиме командной строки. Запускаем утилиту:

```
> quartus_stp -s
> source cpanel.tcl
```

В данном случае файл *cpanel.tcl* должен находиться в той же директории, что и сама утилита. После выполнения второй команды открывается окно, показанное на рисунке 4.

Для создания графического окна и обработки данных используются стандартные команды языка Tcl/Tk. В сети Интернет доступно большое количество информации об этом

языке, в частности, на интернет-странице http://www.tcl.tk. Однако для управления интерфейсом VJTAG используются специальные команды, информация о которых приведена в [2, 3].

Прежде чем отправлять данные, следует узнать, какие устройства подключены к порту USB:

```
set usb [lindex
[get_hardware_names] 0]
set device [lindex
[get_device_names -hardware_name
$usb] 0]
```

Первая команда присваивает переменной *usb* имя устройства, которое присутствует на шине USB. Функция *lindex* выбирает первое устройство (из найденных) командами *get_hardware_names* и *get_device_names*.

После обнаружения кабеля USB-blaster и ПЛИС Altera EP3C25 на шине USB можно посылать данные и команды для VJTAG, но предварительно необходимо загрузить прошивку с мегафункцией VJTAG в ПЛИС с помощью пакета Quartus.

Блок команд обмена данными с VJTAG в общем случае содержит следующие команды:

```
open_device -hardware_name $usb -
device_name $device
device_lock -timeout 10000
```

Это – команды открытия устройства и его блокировки. Если эти две команды были выполнены успешно, можно производить обмен данными при помощи двух основных команд.

```
device_virtual_ir_shift -in-
stance_index 0 -ir_value 2
```

Эта команда программирует регистр инструкций *IR* мегафункции VJTAG (*Instance_index 0* обозначает номер мегафункции). В данном примере используется всего один VJTAG, поэтому параметр *instance_index* всегда равен нулю; *ir_value* – непосредственно команда в десятичной системе счисления, которая будет присутствовать на выводах *ir_in*[5..0] (см. рис. 3). Команда записи данных выглядит следующим образом:

```
device_virtual_dr_shift -in-
stance_index 0 \
-dr_value
$data_read$ssram_cmd$seq_ssram_ad
dr -length 64 -value_in_hex
```

В результате выполнения этой команды в сдвиговый регистр декодера загружается 64-битное слово, задаваемое параметром *dr_value*. Сдвиговый регистр, в который будет загружено это слово, определяется предыдущей командой установки регистра инструкций. В данном случае это будет регистр для записи памяти, так как значению *ir_value 2* соответствует команда записи памяти.

Для чтения данных используется та же самая команда, но в данном случае задаётся только длина вектора (сами данные не задаются) и назначается переменная для чтения:

```
set ssram_r
[device_virtual_dr_shift -in-
stance_index 0 -length 32 -
value_in_hex]
```

После выполнения этой команды переменная *ssram_r* будет содержать 32-битное слово, считанное из сдвигового регистра декодера.

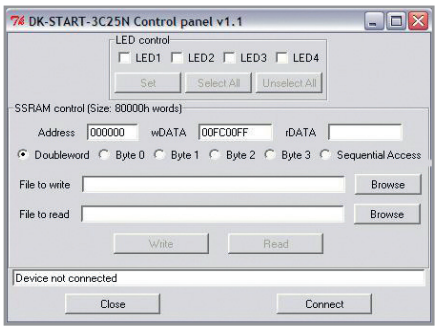

Рис. 4. Управляющий сценарий Tcl/Tk

Следующий шаг – создание конечных логических устройств. В файле проекта содержатся контроллеры статической памяти и светодиодов. Функция управления светодиодами достаточно проста, поэтому она включена в блок *vjtag_decoder*. Контроллер статической памяти вынесен в отдельный блок *ssram_cntr*.

При создании новых устройств модифицируются все блоки, т.е. добавляется новое устройство в ПЛИС и, соответственно, модифицируется декодер и управляющий сценарий.

Заключение

Использование мегафункции VJTAG является одним из вариантов организации обмена данными с отладочной платой DK-START-3C25N. Применение языка сценариев Tcl/Tk позволяет создавать полнофункциональные графические приложения для контроля за работой платы.

Литература

1. www.altera.com: Application note 39. IEEE 1149.1 JTAG Boundary-Scan Testing in Altera Devices www.altera.com.
2. www.altera.com: Virtual JTAG (sld_virtual_jtag) Megafunction User Guide.
3. www.altera.com: Quartus II Scripting Reference Manual.