Реализация обмена данными между ПЛИС и процессорной системой в Cyclone V SoC на базе платы DE1-SoC от Terasic

Алексей ПЕТРОВ
apetrov@icgamma.ru

Введение

Несмотря на то, что системы на кристалле от ALTERA появились больше года назад, многие инженеры-разработчики, имея желание освоить такие системы, сталкиваются с трудностями при изучении технической документации, которая предоставлена на сайте производителя.

Система на кристалле (или по-английски System on Chip, или SoC) от ALTERA представляет собой две самостоятельные функциональные части ПЛИС (или FPGA) и процессорную систему (по-английски Hard Processor System, или HPS), соединенные между собой интерфейсом обмена данными, причем каждая часть имеет свои входы/выходы на корпусе. Процессорная система содержит микропроцессорный блок (Microprocessor unit, или MPU), состоящий из одного или двух ядер процессора ARM Cortex-A9, контроллер DMA, два контроллера Ethernet, два контроллера USB 2.0 OTG, контроллер NAND flash, контроллер Quad SPI, SD/MMC-контроллер, два SPI-контроллера, четыре I^2C-контроллера, память RAM 64 кбайт, два UART- и два CAN-контроллера. Таким образом, данная система на кристалле способна решать различного рода задачи, в том числе цифровой обработки сигналов, реализуемые в FPGA, а результаты обработки, например, передавать в HPS, обрабатывать их процессором и обработанные данные направлять на устройства периферии.

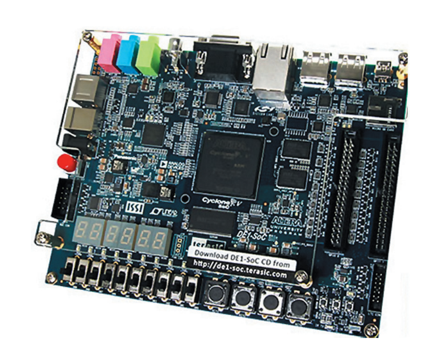

Рис. 1. Плата DE1-SoC

Поскольку система имеет множество возможностей, возникает вопрос, с чего начать, для того чтобы освоить данную систему и использовать ее для реализации инженерных задач. Напрашивается вывод, что простейшей задачей является передача данных из FPGA в HPS и наоборот.

Поэтому цель данной публикации — показать пример реализации обмена данными между FPGA и HPS для того, чтобы инженеры могли использовать его при реализации своей задачи на SoC. Речь пойдет о проектировании оригинального устройства, в котором FPGA формирует посылку из 32 данных с разрядностью 16, такая посылка записывается в первую память внутри FPGA, HPS читает эти данные и записывает в другую память внутри FPGA. Затем FPGA прочитывает эти данные из второй памяти. Именно здесь обрабатывается посылка данных, а не одиночная информация, так как интерфейс обмена данными между FPGA и HPS предусматривает пакетную передачу данных. Внутри FPGA также используется двухпортовая статусная память, в которую по своим адресам HPS и FPGA записывают статусную информацию, содержащую признаки, что определенная задача для FPGA или HPS выполнена. Записывается статусная информация по одному порту одним устройством (FPGA или HPS), по другому порту она может быть прочитана иным устройством (HPS или FPGA).

Реализуется данный проект на плате DE1-SoC от Terasic. Выбор платы основан на двух факторах: относительно недорогой стоимости и широких демонстрационных и отладочных возможностях, таких как наличие достаточно большого числа светодиодов и семисегментных индикаторов для наглядного отображения информации. В данном проекте используются семисегментные индикаторы только лишь для наблюдения за записываемыми и читаемыми данными. Индикаторы подключены к выводам FPGA, управление свечением сегментов осуществляется из FPGA, процессор не имеет готового интерфейса доступа к ним.

На рис. 1 показана плата DE1-SoC (фотография взята с сайта www.terasic.com). На плате установлен кристалл 5CSEMA5F31C6 с 85 000 логических элементов в FPGA, в кристалле HPS содержит двухъядерный процессор Cortex-A9. Конфигурация FPGA возможна через флэш EPCS128, интерфейс JTAG или из HPS. На плате предусмотрена память SDRAM 64 Мбайт со стороны FPGA и 1 Гбайт DDR3 SDRAM со стороны HPS. Есть socket для micro SD-карты, откуда можно загружать Linux. Со стороны HPS есть два порта USB 2.0, USB-to-UART для отладки через консоль приложений Linux, порт Ethernet 1Gb. Также на плате есть два разъема с 40 выводами, разъем на 10 выводов на вход АЦП, разъем LTC, есть выход VGA с 24-битного ЦАП, 24-битный аудиокодек, TV-декодер, 8-канальный 12-разрядный АЦП со скоростью 1 MSPS. Со стороны FPGA имеется четыре пользовательские кнопки, 10 переключателей, шесть семисегментных индикаторов, 10 светодиодов, один светодиод со стороны HPS. Кроме того, со стороны HPS есть две кнопки reset: HPS_RST и HPS_WARM_RST, G-Sensor. Таким образом, мы имеем многофункциональную плату с большими возможностями для отладки различных задач. В данном случае будут использоваться семисегментные индикаторы, кнопка сброса FPGA, порт Ethernet, USB-to-UART, интерфейс JTAG, SD-карта.

Проект, описанный в данной статье, создан в Quartus II версии 14.1, но его можно с небольшими отличиями реализовать и в версиях 12.1, 13.1. В версии 15.0 потребуется дополнительно программным путем в С задать используемое семейство CycloneV/ArriaV, поскольку данная версия подразумевает работу и с Arria 10, это можно сделать в заголовочном файле, который находится в директории *SoC_installed_dir/embedded/ip/altera/hps/altera_hps/hwlib/include/hwlib.h*. В данном файле в самом верху прописать:

```
#define soc_cv_av
```

при использовании Cyclone V или Arria V, или

```
#define soc_a10
```

при использовании Aria 10.

Также в версии 15.0 будет другой путь к некоторым заголовочным файлам, в теле программы эти файлы подключаются следующим образом.

Для Cyclone v:

```
#include "soc_cv_av/socal/socal.h"
#include "soc_cv_av/socal/hps.h"
#include "soc_cv_av/socal/alt_gpio.h"
```

Для Aria 10:

```
#include "soc_a10/socal/socal.h"
#include "soc_a10/socal/hps.h"
#include "soc_a10/socal/alt_gpio.h"
```

Для этого примера следует выставить настройки, соответствующие Cyclone V.

Часть 1. Описание проекта

Текущая задача — передать данные из ПЛИС в процессор и обратно. Поскольку мост обмена данными — это всего лишь мост, то должны быть конечные устройства, куда по этому мосту необходимо обращаться.

Так как у ALTERA есть интерфейс Avalon, который в QSys совмещается с AXI, то проще использовать память с Avalon, куда ПЛИС записывала бы данные по одному порту, а затем по второму порту процессор их читал.

Вторая причина использования памяти в том, что передача по AXI происходит транзакциями с handshake и пакетами, поэтому целесообразно в ПЛИС сформировать посылку данных и отправить ее на процессор, и наоборот — принять посылку данных из процессора.

Существует еще два понятия — master и slave. Master инициирует запрос на передачу данных, slave — этот запрос обрабатывает.

Поскольку цель работы — визуальная демонстрация передаваемых данных на отладочной плате, то в качестве данных, записываемых в память, формируются сигналы, управляющие свечением семисегментных индикаторов. Можно было бы генерировать и другие данные, но для демонстрации и визуального наблюдения выбраны именно эти сигналы. Отсчитываемый временной интервал при формировании и записи данных соответствует примерно 1 с для визуального наблюдения. Посылка данных состоит из 32 слов.

Сначала опишем сигналы управления (рис. 2). Счетчик count задает временной интервал, 26-й разряд примерно соответствует интервалу 1 с, вся схема работает на частоте clock — 50 МГц. Следующая схема — select_enable — формирует из 26-го разряда сигнал, соответствующий длительности 1 такт сигнала clock, этот сигнал называется glb_enable. Он задает временной интервал работы части схемы, которая отвечает за формирование данных и запись этих данных в память.

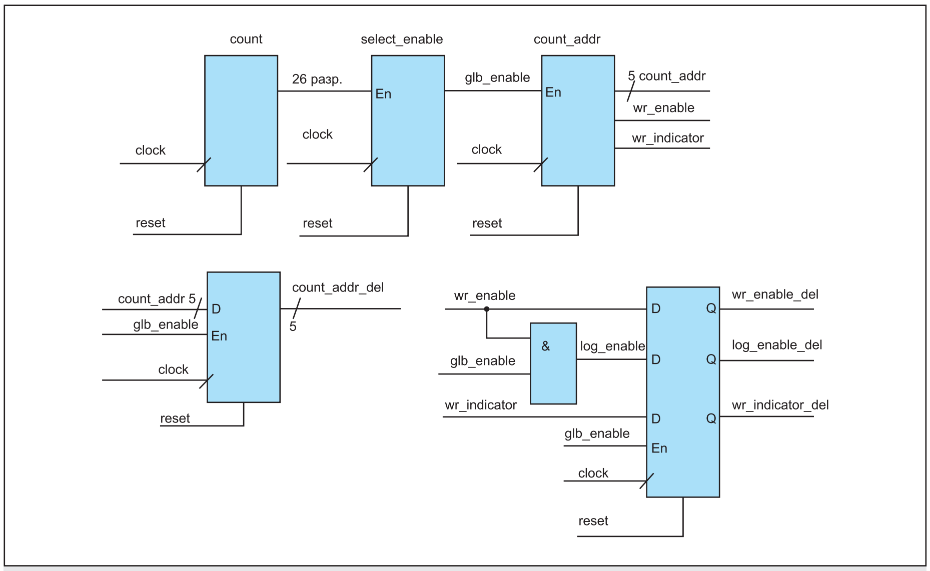

Рис. 2. Схема формирования сигналов управления

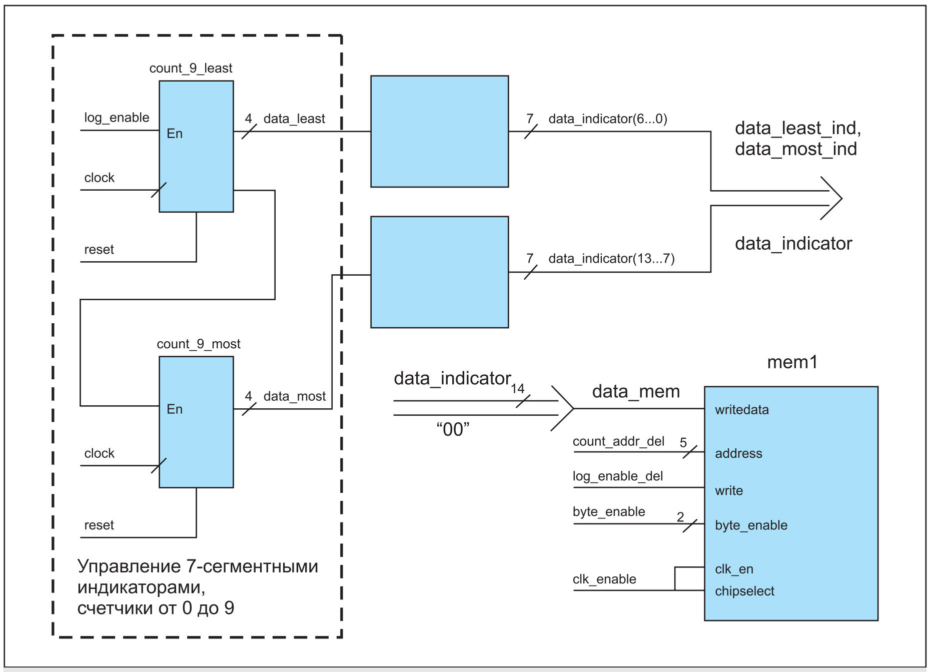

Рис. 3. Схема формирования сигналов управления семисегментными индикаторами

Такой сигнал подается на разрешающий вход счетчика count_addr, который тоже с интервалом 1 с формирует адрес для записи в память данных. Счетчик считает от 0 до 31 и останавливается, при этом формируются сигналы: wr_enable и wr_indicator. Когда происходит сброс счетчика, то wr_enable='1', wr_indicator='0', когда счетчик считает, то wr_enable='1', wr_indicator='1', когда счетчик закончил счет, то wr_enable='0', wr_indicator='0'.

Формируем также сигнал разрешения записи — log_enable, когда счетчик адреса окончил счет, данные в память не записываются. Все сформированные сигналы задерживаем на величину секундного интервала, чтобы записать данные по 0 адресу, получаем сигналы: count_addr_del, wr_enable_del, log_enable_del, wr_indicator_del.

Сигналы управления семисегментными индикаторами (рис. 3) формируются следующим образом. Два счетчика от 0 до 9 соединены между собой сигналом переноса, который появляется в младшем счетчике count_9_least при переходе счета с 9 на 0. Сигнал, разрешающий счет младшему счетчику, — log_enable. Сигналы с выхода старшего счетчика count_9_most и младшего счетчика count_9_least подаются на две одинаковые комбинационные схемы, работающие с использованием таблицы истинности, которая задает соответствие выхода комбинационной схемы входному сигналу (табл. 1). Таким образом формируются сигналы управления семисегментными

Таблица 1. Таблица соответствия выходных сигналов входным для комбинационной схемы

Data_in	Data_out
0000	1000000
0001	1111001
0010	0100100
0011	0110000
0100	0011001
0101	0010010
0110	0000010
0111	1111000
1000	0000000
1001	0010000

индикаторами — data_indicator, которые будут записаны в память mem1, с добавлением двух нулевых старших разрядов до 16 разрядов, поскольку память можно сгенерировать на 16 разрядов данных. Эти же сигналы поступают на выход FPGA непосредственно на сами семисегментные индикаторы с другими названиями: data_least_ind, data_most_ind. Следует отметить, что на данной плате сегменты индикатора загораются при подаче лог. ‘0’.

Другие сигналы на память mem1 сформированы ранее, необходимо задать дополнительные сигналы по требованию интерфейса Avalon — byte_enable = “11” и clk_enable = ‘1’. Сигнал byte_enable показывает, какой из байтов в записываемом или читаемом слове верный, в данном случае — оба имеют значение. Таким образом, мы описали первый порт двухпортовой памяти mem1, через который ПЛИС записывает данные в память. Второй порт предназначен для чтения данных процессорной системой.

Для того чтобы передавать текущее состояние ПЛИС для использования его процессором и наоборот — передавать текущее состояние процессора для ПЛИС, необходимы статусные регистры (рис. 4). Так как довольно проблематично соединить регистры с мостом обмена «в лоб», то здесь используется память mem_status с интерфейсом Avalon-MM с разрядностью данных — восемь, на четыре слова, поскольку это минимум, который можно сгенерировать для ПЛИС. У данной памяти адрес = “00”— это адрес ПЛИС, адрес “01” — это адрес процессора. Слово stat_word_fpga = “00000001”, записанное по адресу “00”, означает признак, что FPGA записал данные в память mem1. Слово stat_word_hps = “00000010” по адресу “01” означает, что процессор записал данные в память mem2. То есть доступ к статусной памяти имеют и процессор, и ПЛИС на чтение и запись по обоим портам.

Схема select_after формирует сигнал wr_stat_word = ‘1’, длительностью 1 такт, по завершении wr_indicator_del и задерживает его на 1 такт. По сигналу wr_stat_word = ‘1’ в мультиплексоре выбирается адрес для записи “00” и по этому адресу записывается в статическую память слово stat_word_fpga = “00000001”. Далее, когда wr_stat_word = ‘0’,

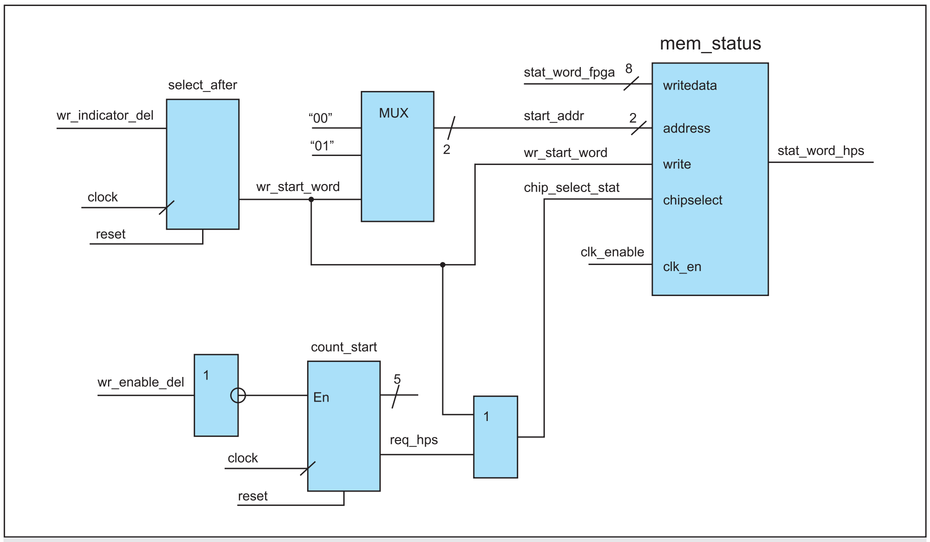

Рис. 4. Схема управления статусной памятью

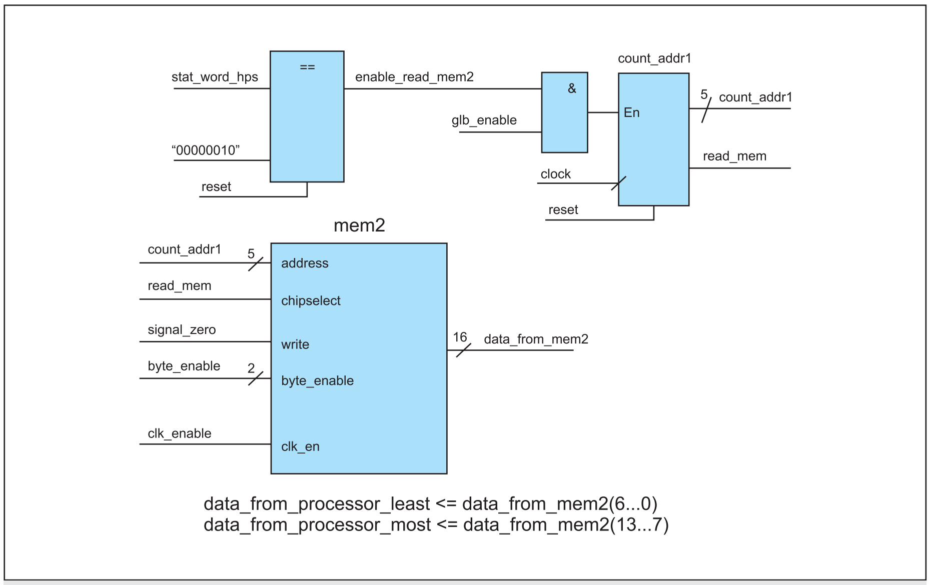

Рис. 5. Схема чтения памяти mem2

устанавливается режим чтения статусной памяти адреса процессора.

Следующая часть схемы начинает вырабатывать периодический запрос на чтение статусной памяти. При переполнении счетчик count_stat вырабатывает сигнал запроса req_hps, который через логическое “ИЛИ” поступает на chip_select. Происходит периодическое чтение адреса “01”. Таким образом, ПЛИС обращается по одному порту к двухпортовой памяти mem_status, второй порт предназначен для доступа процессора.

Далее стоит компаратор с защелкой (рис. 5). Как только слово stat_word_hps, прочитанное по адресу “01” статусной памяти, сравняется с “00000010”, то на выходе схемы сформируется сигнал enable_read_mem2 = ‘1’. Счетчик count_adrr1 начнет перебирать адреса памяти mem2, при этом будет вырабатывать сигнал read_mem = ‘1’, в результате из памяти будут читаться данные data_from_mem2.

Эти данные распределяются на data_from_processor_least и data_from_processor_most, которые выводятся на выводы ПЛИС к семисегментным индикаторам, визуально отображающим процесс чтения данных. Чтобы визуализировать процесс чтения (видеть читаемые данные на семисегментных индикаторах), необходимо сформировать разрешающий сигнал на счетчик, объединив по схеме “И” сигналы enable_read_mem2 и glb_enable. При этом signal_zero = ‘0’, byte_enable = “11”, clk_enable = ‘1’.

В данном проекте исходный код написан на языке VHDL, здесь он не приводится, но по представленному выше описанию любой инженер-разработчик легко напишет его на любом языке — Verilog или VHDL. Сама память подключается в системе QSys. После генерации в системе QSys части проекта, содержащей память и подключение HPS, можно выполнить соединение остального проек-

Таблица 2. Расположение портов ввода/вывода на выводах ПЛИС

Название сигнала порта	Вывод ПЛИС	Тип
data_from_processor_least[6]	PIN_AH28	out
data_from_processor_least[5]	PIN_AG28	out
data_from_processor_least[4]	PIN_AF28	out
data_from_processor_least[3]	PIN_AG27	out
data_from_processor_least[2]	PIN_AE28	out
data_from_processor_least[1]	PIN_AE27	out
data_from_processor_least[0]	PIN_AE26	out
data_from_processor_most[6]	PIN_AD27	out
data_from_processor_most[5]	PIN_AF30	out
data_from_processor_most[4]	PIN_AF29	out
data_from_processor_most[3]	PIN_AG30	out
data_from_processor_most[2]	PIN_AH30	out
data_from_processor_most[1]	PIN_AH29	out
data_from_processor_most[0]	PIN_AJ29	out
data_least_ind[6]	PIN_W25	out
data_least_ind[5]	PIN_V23	out
data_least_ind[4]	PIN_W24	out
data_least_ind[3]	PIN_W22	out
data_least_ind[2]	PIN_Y24	out
data_least_ind[1]	PIN_Y23	out
data_least_ind[0]	PIN_AA24	out
data_most_ind[6]	PIN_AA25	out
data_most_ind[5]	PIN_AA26	out
data_most_ind[4]	PIN_AB26	out
data_most_ind[3]	PIN_AB27	out
data_most_ind[2]	PIN_Y27	out
data_most_ind[1]	PIN_AA28	out
data_most_ind[0]	PIN_V25	out
reset	PIN_Y16	in
clock	PIN_AF14	in

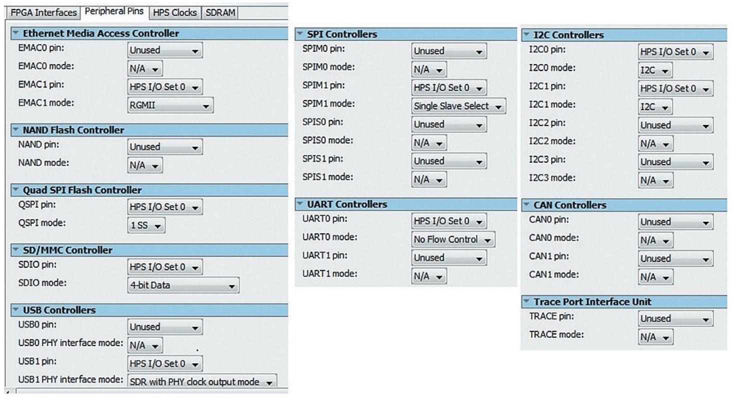

Рис. 6. Настройки вкладки Peripheral Pins

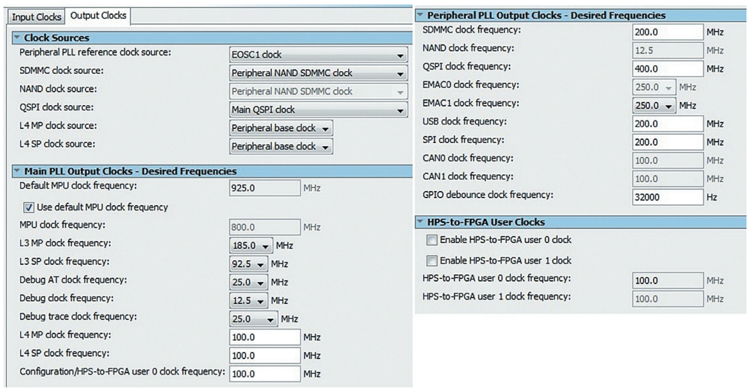

Рис. 7. Настройки выходных сигналов clock для периферии

та к памяти путем подключения сгенерированного модуля в QSys — как иерархического компонента проекта. Расположение портов ввода/вывода устройства на выводах ПЛИС показано в таблице 2.

Таким образом, мы рассмотрели структуру проекта, следующая часть будет описывать построение системы в QSys.

Часть 2. Построение системы в QSys

Далее будет рассказано о построении системы в QSys в Quartus версии 14.1. Если вы предполагаете это делать в других версиях, то возможны небольшие отличия.

Вызываем систему QSys, меню **Tools/Qsys**. Все компоненты находятся в окне слева. Ставим память on_chip_memory2_0 двойным щелчком мыши — Basic Functions/On Chip Memory/(RAM or ROM), в настройках выставляем двухпортовую память (Dual-port access), один clock (Single clock operation), Read During Write Mode — DONT_CARE, Block type — AUTO. Ставим размер данных (Data width) — 16 разрядов, размер памяти (Total memory size) — 64 байт, поля Read latency оставляем равными 1, Reset Request — Disabled, ECC Parameter — Disabled, в настройках Memory initialization не должно быть галочек. Нажимаем кнопку **Finish**.

Подсоединяем reset и clock к источнику (правой кнопкой мыши щелкаем по названию порта, в верхнем пункте Connections всплывающего окна к порту *clk_0.clk_reset* или *clk_0.clk* соответственно). Порт s1 делаем экспортом (для того чтобы ПЛИС обращалась к нему), для этого в столбце Export делаем двойной щелчок мыши напротив порта s1, переименовываем его в mem1_s1, другой порт, s2, будет соединен с мастером HPS. В эту память ПЛИС будет сбрасывать данные, а процессор — их читать; в первой части эта память называется mem1.

Вызываем процессорную систему (двойным щелчком мыши по **Processors and Peripherals/Hard Processor Systems/ Arria V/Cyclone V Hard Processor Systems**) и выставляем настройки, описанные ниже. Настройки FPGA Interfaces сформированы исходя из задачи, остальные взяты из проекта по плате Terasic. Сделано это потому, что надо сохранить работу периферии, то есть загрузить процессор из SD-карты, сохранить работу через кабель USB — последовательный интерфейс и т.д. Периферия тоже окончательно привязана к определенным выводам HPS. Все это свидетельствует о том, что требовалось сохранить данные настройки.

Итак, настройки следующие. Вкладка **FPGA Interfaces**. Пункт General — отсутствуют все галочки. Пункт AXI Bridges — все разрядности должны быть равны 32. FPGA-to-HPS SDRAM Interface — таблица должна быть пустой, если там что-то есть, значит, необходимо убрать с помощью “–”, Resets — все галочки должны отсутствовать, DMA Peripheral Request — в таблице везде должно стоять No. Interrupts — галочка отсутствует. HPS-to-FPFA — все галочки отсутствуют. Вкладка **Peripheral Pins** — настройки показаны на рис. 6. Вкладка HPS Clocks — контролируем значение 25 МГц от EOSC1 и EOSC2, а также отсутствие галочек в FPGA-to-HPS PLL Reference Clocks. Остальные значения установлены по умолчанию. Выходные сигналы clock должны соответствовать сигналам, показанным на рис. 7.

Вкладка SDRAM — здесь все настройки устанавливаются по отношению к внешней памяти, можно ничего не менять, поскольку мы все равно не используем внешнюю память. Нажимаем кнопку **Finish**.

Сигналы: h2f_axi_clock, f2h_axi_clock, h2f_lw_axi_clock подсоединяем к источнику clock, h2f_axi_master подсоединяем к порту s2 памяти on_chip_memory2_0.

Ставим память — on_chip_memory2_1 — Basic Functions/On Chip Memory/(RAM or ROM), выставляем те же самые настройки, что и для памяти on_chip_memory2_0. Соединяем reset и clock с источником, порт s2 делаем экспортом (так же как и для памяти on_chip_memory2_0), переименовываем его

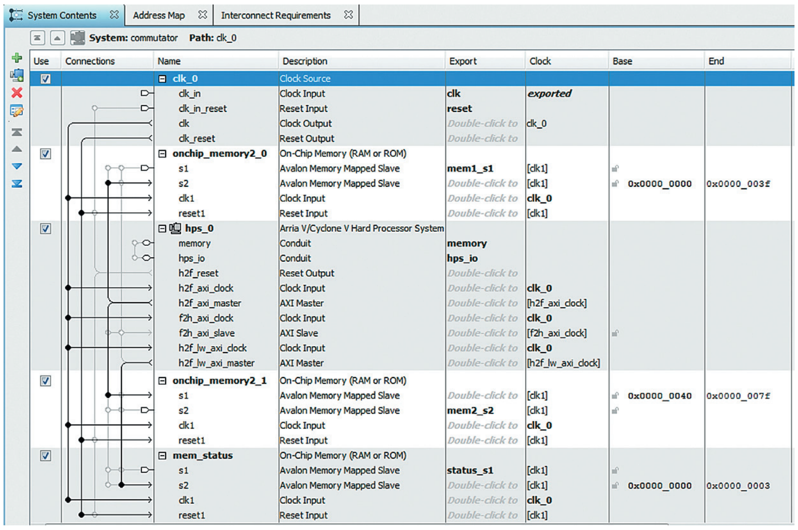

Рис. 8. Вид системы в QSys

на mem2_s2. Другой порт, s1, соединяем с мастером HPS — h2f_axi_master. В эту память процессор будет скидывать данные, а ПЛИС будет их читать, в первой части данная память называется mem2.

Далее нужно устранить конфликт адресов, для чего меняем начальный адрес для памяти on_chip_memory2_1 во вкладке Address Map (вверху) на 0x00000040, щелкнув двойным щелчком мыши напротив порта on_chip_memory2_1.s1 по адресу 0x0000_0000. Система сама поставит орфографические разделители.

Вызываем память RAM or ROM, ставим настройки как и для памяти on_chip_memory2_0, только другими будут: размер слова — 8 бит (минимально возможное), количество байт — 4 шт. (минимально возможное). Сигналы clk1 и reset1 соединяем с clk и reset источника. Порт s1 делаем экспортом, переименовываем в status_s1, порт s2 соединяем с мастером интерфейса Lightweight — h2f_lw_axi_master. Переименовываем память в mem_status. Согласно описанию в первой части — это статусная память.

Сохраняем систему. В меню **File/Save as** пишем имя *commutator.qsys*. Если в процессе создания системы возникла ошибка в настройках, то, щелкнув двойным щелчком мыши по названию компонента, откроем вкладку **Parameters**, где можно отредактировать любые настройки. На рис. 8 изображена система, полученная в QSys.

Небольшой комментарий. В названии портов встречаются сочетания h2f и f2h, которые указывают, что к чему обращается. Если FPGA обращается к HPS, то это будет f2h — сокращение от FPGA — to (или 2) — HPS. И наоборот, если HPS обращается к FPGA, то это будет h2f, или сокращение от HPS — to (цифра 2) — FPGA. Сокращение lw соответствует интерфейсу Lightweight, медленному и предназначенному для обмена статусной информации, передачи констант и т. п.

После сохранения системы надо сгенерировать HDL, Testbench, чтобы использовать систему в файле верхнего уровня и для моделирования. Моделирование рассматривать не будем, это отдельная самостоятельная часть, а как встроить полученную систему в проект — опишем подробно.

Нажимаем внизу кнопку **Generate HDL**. Появится окно **Generation**, в верхней части окна можно выбрать язык для синтеза проекта — например, VHDL. Далее галочка, если требуется синтезировать проект не в Quartus, а в других системах. Эту галочку не ставим. Блок-символ создавать необязательно, он нужен для тех, кто еще пользуется графическим редактором. Затем в окне предлагается выбор языка, в котором будет описана система для моделирования. Выбираем язык, если нужно — ставим галочку «Смешанное моделирование». Далее показана выходная директория, где будет находиться результат. Нажимаем кнопку **Generate**. После генерации системы можно войти в пункт меню **Generate/HDL EXAMPLE** и в окне выбрать язык. После этого в окне появится отображение портов в выбранном языке для того, чтобы это можно было скопировать и вставить в файл верхнего уровня как иерархический компонент. Пункт меню **Generate/Generate Testbench System** предназначен для моделирования с использованием BFMs. Здесь мы не станем рассматривать моделирование, а значит, не будем выполнять данный пункт.

После генерации системы необходимо добавить ее к проекту. Для этого в Quartus в меню **Project** следует выбрать пункт **Add/Remove Files in Project**, найти файл *commutator.qsys* в папке проекта, нажав на **“…”**, затем добавить, нажав кнопку **Add**. Затем нужно нажать кнопки **Apply** и **OK**. Папку commutator и все ее содержимое создает QSys. Там можно найти нетлист в папке synthesis на VHDL или Verilog — это зависит от того, какой язык был выбран для синтеза. Можно использовать название компонента с портами, чтобы скопировать его и вставить в файл верхнего уровня как иерархический компонент. Далее вставляем commutator в файл верхнего уровня по всем правилам VHDL или Verilog, соединяем порты этого компонента с сигналами остальной схемы и выводим порты компонента commutator (их мы не задавали), относящиеся к процессорной системе (HPS), к портам файла верхнего уровня всего проекта. Таким образом, мы подключили к проекту commutator, синтезированный в QSys.

Перед синтезом проекта необходимо запустить *tcl*-скрипт, чтобы указать разводчику (Fitter), где находятся выводы HPS-системы, отвечающие за память. (Вообще, это было обнаружено, когда возникли ошибки на этапе Fitter во время синтеза, и такие ошибки указывали на порты памяти DDR.) Поэтому вы можете сначала синтезировать проект, а затем, проанализировав появившиеся ошибки и выяснив, на какой порт они указывают, вычислить, какой скрипт запускать. Для этого необходимо войти в пункт меню **Tools/Tcl Scripts**, найти в дереве Commutator/synthesis/submodules файл *hps_sdram_p0_pin_assignments.tcl*, однократно щелкнуть по нему мышкой, чтобы выделить его, и нажать на кнопку **Run**. Затем в окне высветится сообщение, что скрипт выполнен, следует закрыть его, нажав **OK**, и закрыть окно запуска скрипта.

Далее нужно задать частоту входного сигнала clock. В версии Quartus 14.1 это делается следующим образом. Открываем меню **Tools/TimeQuest Timing Analyzer**, в появившемся окне создаем новый файл, вписываем в поле строчку:

```
create_clock -name "clock" -period 10.000ns [get_ports {clock}]
```

В текстовом редакторе сохраняем как (меню **File/Save as**) *имя_головного_файла_проекта.sdc*. Закрываем TimeQuest Timing Analyzer. Добавляем этот файл к проекту так же, как и добавляли файл с системой QSys.

У нас задана частота 100 МГц, поскольку для синтеза лучше выбрать частоту с запасом, хотя FPGA будет работать на 50 МГц.

Далее запускаем на синтез весь проект. Заходим в меню **Processing** и выбираем пункт **Start Compilation**, синтез должен пройти без ошибок.

Итак, мы завершили проект в Quartus, и все, что касается FPGA, уже сделано. Следующая часть посвящена запуску программы на ARM-процессоре под Linux и загрузке проекта в кристалл.

Часть 3. Работа с Linux, программирование на C++, загрузка проекта в кристалл и запуск платы

Для работы с SoC ALTERA предлагается использовать операционную систему Linux, которую можно скачать с сайта www.RocketBoards.org. На сайте есть инструкция по применению ОС, а также другие полезные материалы.

Для упрощения всей процедуры мы будем использовать Linux с сайта Terasic, поскольку для отработки принципа передачи данных этого достаточно. Сама операционная система и документация на плату расположена по ссылке: http://cd_de1-soc.terasic.com. Инструкция, как портировать данную операционную систему, находится в документе Getting Started Guide внутри DE1-SoC CD-ROM. Для создания загрузочной SD-карты необходимо установить *Win32DiskImager.exe* — для работы с платой UART terminal и PuTTY terminal. Вся процедура установок и настроек подробно изложена в данной инструкции. Отметим лишь, что использовать Linux для SoC можно только с SD-карты. В данной работе применена Linux, скачанная с сайта Terasic по первой ссылке в таблице.

Теперь требуется написать программу на C++, которая будет выполнять чтение и запись данных по определенным адресам, скомпилировать ее, получить исполняемый файл, записать его на SD-карту в директорию root и запустить исполняемый файл. За основу возьмем программу на C из проекта my_first_hps-fpga, предложенную Terasic, и ее проанализируем.

Вначале происходит вызов драйвера памяти устройства dev/mem, что является стандартной процедурой. Далее идет вызов функции mmap, остановимся на ней более подробно. Описание ее работы есть во множестве источников, например, подробная информация о ней предлагается по ссылке: www.man7.org/linux/man-pages/man2/mmap.2.html. Саму функцию с параметрами можно представить в виде:

void *mmap(void **addr*, size_t *length*, int *prot*, int *flags*, int *fd*, off_t *offset*);

Переменная addr определяет стартовый адрес для новой карты памяти в Linux, если ее значение NULL, то kernel выбирает адрес. Для нас это непринципиально, а потому зададим ее равной NULL.

Важным параметром для выделения области памяти является размер length, определяемый в байтах, и стартовый физический адрес offset, от которого отсчитывается область памяти, заданная в количестве байтов.

Возвращаемся к анализу программы. Переменная length задана константой HW_REGS_SPAN = 0x04000000, начальный адрес offset — константой HW_REGS_BASE = ALT_STM_OFST, а значение ALT_STM_OFST в заголовочном файле *socal/hps.h* равно ALT_STM_OFST = 0xfc000000.

Согласно документации ALTERA (по ссылке www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/cyclone-v/cv_5v4.pdf) "Cyclone V Hard Processor System Technical Reference Manual", в таблицах 1–2, базовый адрес FPGA slaves = 0xC0000000, базовый адрес периферии 0xFC000000, базовый адрес моста Lightweight FPGA slaves = 0xFF200000. В этой же документации следует обратить внимание и на рис. 9, где показана карта адресов так, как эти адреса видит процессор.

Отсюда видно, что в данной программе выделена вся область памяти периферии, начиная с адреса 0xFC000000 до 0xFFFFFFFF с размером 0x04000000. Далее, если разобрать математические действия при формировании адреса h2p_lw_led_addr, то мы заметим, что ((unsigned long) (ALT_LWFPGASLVS_OFST + PIO_LED_BASE) & (unsigned long) (HW_REGS_MASK)) — образует смещение 0x03200000 относительно адреса 0xFC000000, равное по величине, чтобы обращаться по начальному адресу моста Lightweight FPGA slaves = 0xFF200000. Здесь само смещение pio_led_base равно 0, задается в QSys.

Итак, основные моменты с адресацией мы разобрали, можно двигаться дальше. Вернемся к рассматриваемой задаче. Листинг программы показан на рис. 9.

```c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "hwlib.h"
#include "socal/socal.h"
#include "socal/hps.h"
#include "socal/alt_gpio.h"


#define MEM_SPAN ( 0x40000000 )

#define BRIDGE_H2F ( 0xC0000000 )

#define OFST_LW ( 0x3F200000 )


#define mem2_base_ofst ( 0x00000040 )        // from QSys
#define mem_status_base_ofst ( 0x00000000 )

int main() {

    void *virtual_base;
    int fd;

    int loop_flag;
    unsigned char data_status_mem;
    int j = 0;
    int i = 0;
    int k = 0;
    unsigned char value_mem1[64];
    unsigned char value_mem2[64];

    void *mem1_base;
    void *mem2_base;
    void *mem_status_base;

    //вызов драйвера

    if( ( fd = open( "/dev/mem", ( O_RDWR | O_SYNC ) ) ) == -1 ) {
        printf( "ERROR: could not open \"/dev/mem\"...\n" );
        return( 1 );
    }

    //завершен вызов драйвера

       //Выделение области памяти

    virtual_base = mmap( NULL, MEM_SPAN, ( PROT_READ | PROT_WRITE ), MAP_SHARED, fd, BRIDGE_H2F );

    if( virtual_base == MAP_FAILED ) {
        printf( "ERROR: mmap() failed...\n" );
        close( fd );
        return( 1 );
    }
       //Завершено выделение памяти.

        mem1_base=virtual_base;

        mem2_base=virtual_base + ( ( unsigned long  )( mem2_base_ofst ) );

        mem_status_base=virtual_base + ( ( unsigned long  )( OFST_LW + mem_status_base_ofst ) );

        printf( "Start program \n");

        for (j = 0; j <= 63; j++)
            {

                value_mem1[j] = 0x00;
                value_mem2[j] = 0x00;

            }

    loop_flag = 0;

    while( loop_flag < 1 ) {

        // ожидание 1000ms
        usleep( 1000*1000 );

        //чтение признака записи из статусной памяти
                data_status_mem = *((uint32_t *) mem_status_base);

        if (data_status_mem == 1)
                {
            loop_flag = 1;
            printf( "Status word= %X \n", data_status_mem);

        }


    } // while

                printf( "loop flag= %d \n", loop_flag);

        //чтение памяти mem1

            for (j = 0; j <= 63; j++)
            {
                value_mem1[j] = *( (uint32_t *) (mem1_base + j) );
            }

            for (j = 0; j <= 63; j++)
            {
                printf( "value_mem1 = %d, %X \n", j, value_mem1[j]);
            }

        //запись памяти mem2
            for (i = 0; i <= 63; i++)
            {
                value_mem2[i] = value_mem1[i];

                printf( "value_mem2 = %d, %X \n", i, value_mem2[i]);
            }

            for (i = 0; i <= 63; i++)
            {
                *( (uint32_t *) (mem2_base + i) ) = value_mem2[i];
                if(i==63)
                   k = 63;
            }

         //запись в status_memory
        if(k == 63)
            {
                *( (uint32_t *) (mem_status_base + 1) ) = 0x02;

                printf( "mem status write \n");

            }

    // освобождение карты памяти и выход

    if( munmap( virtual_base, MEM_SPAN ) != 0 ) {
        printf( "ERROR: munmap() failed...\n" );
        close( fd );
        return( 1 );
    }

    close( fd );

    return( 0 );
}
```

Рис. 9. Листинг программы на языке С

Выделяемая область памяти функцией mmap начинается с физического адреса BRIDGE_H2F = 0xC0000000, с размером MEM_SPAN = 0x40000000. Таким образом подключается вся область, начиная с базового адреса FPGA slaves = 0xC0000000 по конечный адрес периферии 0xFFFFFFFF.

Затем мы формируем базовые (или начальные) адреса для памяти mem1, mem2, mem_status. Для памяти mem1 начальный адрес соответствует адресу FPGA slaves = 0xC0000000, для памяти mem2 начальный адрес соответствует адресу FPGA slaves плюс смещение, определенное в системе QSys или стартовый адрес mem2 в QSys. Для памяти mem_status задаем смещение OFST_LW = 0x3F200000 относительно 0xC0000000, таким образом попадая на начальный адрес моста Lightweight FPGA slaves = 0xFF200000, и добавляем смещение mem_status_base_ofst = 0x00000000, взятое из QSys. Поскольку оно нулевое, его можно не добавлять, но это сделано для наглядности.

Далее следует очень важный момент, на который следует обратить внимание. Когда мы задавали настройки для памяти mem1, mem2 и mem_status, QSys просил задать разрядность данных и количество байт. Для памяти mem1 и mem2 была задана разрядность данных, равная 16, а количество байт составляет 64. Таким образом, обе памяти были созданы для записи 32 слов по 16 разрядов. Со стороны FPGA QSys все сделал так, как мы и ожидаем. А вот со стороны HPS обращение должно быть побайтным, в системе QSys адресное пространство для памяти mem1 и mem2 создано на 64 слова (см. address map в QSys). Поэтому переменные в программе value_mem1 и value_mem2 имеют тип unsigned char (8-разрядные), их количество равно 64, и адресное пространство в циклах перебирается начиная с сформированного базового адреса по базовый адрес+63. Циклы записи и чтения обозначены комментариями. В конце программы происходит освобождение области памяти функцией munmap.

После того как программа написана, сохраняем ее как *main.c*. Далее делаем так же, как по алгоритму, описанному в документации к плате от Terasic. Берем тот же самый Makefile из проекта my_first_hps-fpga, в переменной TARGET можно поменять название на obmen_hps_fpga — это будет имя исполняемого файла. Запускаем *Embedded_Command_Shell.bat*, находящийся в директории с установленным программным обеспечением SOCEDS, в командной строке с помощью оператора cd переходим в директорию, где находится программа *main.c* и Makefile. В командной строке даем команду make, получаем исполняемый файл *obmen_hps_fpga*. Далее мы должны перенести этот файл на SD-карту на плате. Для этого вставляем SD-карту с образом Linux на плату, соединяем ее кабелем Ethernet с компьютером.

После запуска *Putty.exe* в окне нужно ввести команду:

```
# ifconfig eth0 inet 192.168.10.15 netmask 255.255.255.0
# ifconfig eth0 up
```

Здесь inet — это адрес, который должен первыми тремя цифрами совпадать с IP-адресом компьютера, последняя цифра любая — от 0 до 255, netmask — должна совпадать с компьютером.

Где взять эти цифры? Открыть Центр управления сетями и общим доступом, открыть «**Подключение по локальной сети**», нажать на «**Сведения**». Адрес ip — это Адрес IPv4, маска — Маска подсети IPv4. Последнюю цифру адреса присвоим равную 10. Например, вводим:

```
Ifconfig eth0 inet 192.169.0.10 netmask 255.255.255.0.
```

Далее с помощью проводника заходим в директорию, где установлен SOCEDS, директорию Embedded, запускаем файл *Embedded_Command_Shell.bat*. При помощи оператора cd заходим в директорию с исполняемым файлом. Пишем команду без кавычек:

```
"scp obmen_hps_fpga root@192.169.0.10:/home/root"
```

Здесь scp — команда копирования файла, my_first_hps — название копируемого файла, 192.169.0.10 — настроенный IP-адрес платы. Далее будет сообщение "Are you sure you want to continued connecting (yes/no)?", ввести "yes" без кавычек. Если менялся пароль на Terasic, ввести Terasic. Файл скопирован.

Теперь загружаем конфигурационный файл по JTAG в FPGA. Затем в окне **putty** надо ввести команду "ls" без кавычек. Ввести команду: "chmod 777 obmen_hps_fpga" без кавычек, далее можно запустить исполняемый файл, при этом ввести команду:

```
./obmen_hps_fpga
```

Нажимаем на кнопку сброса **FPGA**, которую обозначали, когда делали проект на FPGA. Наблюдаем за семисегментными индикаторами. Левая пара показывает счет от 0 до 32. По окончании на правой паре, спустя некоторое время, должны отобразиться те же самые значения.

Заключение

В данной статье был приведен типичный пример, который может использоваться в любой задаче при разработке системы на кристалле. Здесь показан лишь принцип построения такой системы, данные могли вырабатываться любой другой схемой, необязательно именно такой для семисегментных индикаторов, и количество данных может быть иным. Однако показанный принцип может оказаться полезным для инженеров-разработчиков, так как основные моменты, на которые следует обратить внимание, были отражены в статье. Любые отзывы и замечания прошу присылать по электронной почте.

Литература

1. Cyclone V Hard Processor System Technical Reference Manual. **www.altera.com**
2. **www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/cyclone-v/cv_5v4.pdf**