Контроллер DDR SDRAM для платы DK-START-3C25N

Алексей Гребенников (Актау, Казахстан)

В статье подробно описано построение контроллера памяти DDR SDRAM на языке Verilog для отладочной платы DK-START-3C25N.

Введение

Отладочная плата DK-START-3C25N, краткое описание которой было приведено в [1], содержит ПЛИС Altera EP3C25F324C6 и ряд устройств, которыми эта ПЛИС может управлять, в том числе, динамическую память A2S56D40CTP. Эта память относится к DDR SDRAM и организована по структуре 4 Мбит×16×4. В данной статье рассмотрена реализация контроллера динамической памяти на языке Verilog. Обмен данными между платой и компьютером осуществляется через интерфейс VJTAG, подробно рассмотренный в [1].

Принцип действия динамической памяти

Массив запоминающих элементов в динамической памяти поделен на четыре банка (bank). Каждый банк состоит из набора строк (rows) и столбцов (columns). Управление памятью осуществляется при помощи команд, формируемых определённым набором сигналов CS~ (Chip Select), RAS~ (Row Access Strobe), CAS~ (Column Access Strobe), ~WE (Write Enable), BA[1..0] (Bank Address), A[12..0] (Address), CKE (Clock Enable).

Элементом хранения информации в динамической памяти является конденсатор. В связи с этим возникает необходимость периодической регенерации памяти, т.е. подзарядки конденсаторов. Параметр tREFI определяет средний интервал времени между двумя командами регенерации памяти. Одна команда выполняет регенерацию одной строки всех четырёх банков. Поскольку в данной памяти 8192 строки, полный цикл регенерации занимает время 8192×*tREFI*.

Динамическая память должна быть инициализирована. После включения питания необходимо подождать 200 мкс, а затем подать определённую последовательность команд. После завершения цикла инициализации память переходит в режим ожидания (IDLE) и может выполнять команды контроллера. Каждое состояние памяти в рабочем режиме описывается конечным автоматом, структура которого будет рассмотрена ниже. Более подробно работа динамической памяти описана в спецификации фирмы-производителя [2] и в брошюре фирмы Micron [3].

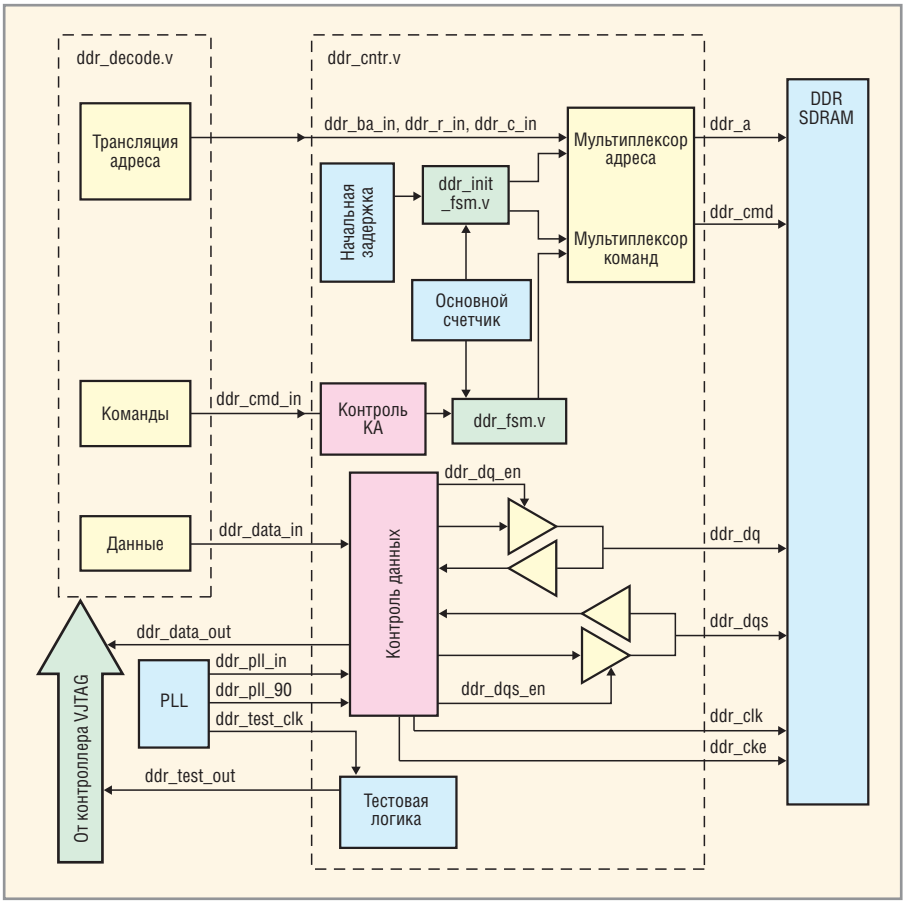

Рис. 1. Блок-схема контроллера памяти DDR SDRAM

Контроллер памяти

Контроллер памяти был построен в среде Quartus II 9.0 Web Edition, симуляция проекта выполнялась с помощью программы ModelSim 6.4a. Полный архив проекта в среде Quartus содержится в файле *vjtag_i_top.qar*. Файлы, необходимые для моделирования, находятся в архиве *ddr_cntr1.zip*, а управляющий сценарий на языке Tcl – в файле *cpanel.tcl*. Этот материал можно найти на сайте журнала в дополнительных материалах к статье.

Блок-схема контроллера приведена на рисунке 1. Команды компьютера принимаются VJTAG-контроллером *vjtag_decoder.v*, подробно описанным в [1]. Затем 72-битный вектор данных поступает на вход блока *ddr_decode.v*, где разделяется на данные для памяти, команду (запись, чтение и т.д.) и адрес. Блок *ddr_decode.v* определяет схему трансляции 32-битного адреса в адрес, необходимый для памяти DDR SDRAM (номера банка, строки и столбца). Из множества схем трансляции выбрана самая простая, называемая flat mapping. При этой схеме два старших бита адреса представляют номер банка, следующие тринадцать битов – номер строки, затем девять битов определяют номер столбца и самый младший бит игнорируется, т.к. минимальной адресу-

емой ячейкой в рассматриваемой памяти является 16-битное слово. На языке Verilog это выглядит следующим образом:

```
assign ddr_dcd_ba[1:0] =
ddr_dcd_vj_cmd_in[24:23];
assign ddr_dcd_r[12:0] =
ddr_dcd_vj_cmd_in[22:10];
assign ddr_dcd_c[8:0] =
ddr_dcd_vj_cmd_in[9:1];
```

Все остальные функции контроллера реализованы в файле *ddr_cntr.v*, причём конечные автоматы инициализации и основного рабочего цикла вынесены в отдельные файлы *ddr_init_fsm.v* и *ddr_fsm.v*.

Блок *PLL* является мегафункцией пакета Quartus и формирует частоты, необходимые для работы; *ddr_pll_in* включает в себя два сигнала частотой 100 МГц, сдвинутых по фазе на 180° друг относительно друга. Тактовая частота платы DR-START-3C25N равна 50 МГц, поэтому коэффициент умножения блока *PLL* для сигналов *ddr_pll_in* равен 2. Обе частоты подаются непосредственно на вход памяти; частота *ddr_pll_in[0]* является основной тактовой частотой блока *ddr_cntr.v*. Сигнал *ddr_pll_90* имеет частоту 100 МГц, сдвинут по фазе на 90° относительно *ddr_pll_in[0]* и используется для записи/чтения данных в блоке *ddr_cntr.v*; *ddr_test_clk* – это сигнал частотой 200 МГц с нулевым фазовым сдвигом относительно *ddr_pll_in[0]*, он является тактовой частотой тестового блока. Функция последнего – запись значений регистров контроллера в определённые моменты времени. Тестовый блок используется только во время отладки работы контроллера.

Рассмотрим подробнее работу блоков контроллера.

Инициализация

После подачи питания блок начальной задержки начинает отсчёт времени, необходимого памяти до начала инициализации (200 мкс):

```
if (ddr_start_count !== 16'h4E20)
// Start delay counter
begin
ddr_start_count <=
ddr_start_count + 1'b1;
        start_delay_done <= 1'b0;
ddr_cke <= 1'b0;
        end
```

Число 16'h4E20 определяет период времени 200 мкс при частоте 100 МГц, в течение которого тактовая частота в памяти отключена, т.е. сигнал *ddr_cke* равен нулю. По истечении времени задержки включается основной счётчик:

```
ddr_count <= ddr_count + 1'b1;
```

Функцией основного счётчика является отсчёт времени между циклами регенерации (tREFI). Этот параметр для рассматриваемой памяти равен 7,8 мкс, что в периодах основного счёчика будет равняться шестнадцатеричному числу 30Ch.

Одновременно с запуском основного счётчика начинается процесс инициализации памяти, управляемый конечным автоматом Мура *ddr_init_fsm.v*, в котором выходной сигнал зависит только от текущего состояния автомата. Стиль программирования – с двумя блоками *always*; текущее состояние кодируется по схеме с одной единицей (*onehot*). Стиль кодирования *onehot* требует построения регистров больших размеров, однако обеспечивает более высокое быстродействие. Более подробно различные стили программирования конечных автоматов описаны в [4, 5].

Первый блок *always* содержит набор защёлок (latches) и определяет правила перехода автомата из одного состояния в другое, а также выходной сигнал при каждом состоянии конечного автомата. При схеме кодирования *onehot* используется инвертированный оператор условия *case*, т.е. он задаётся как заведомо верный – *case (1'b1)*, и затем перебираются все биты текущего состояния автомата. Только один из битов имеет единичное значение. Второй блок *always* содержит набор триггеров и непосредственно осуществляет переход автомата из одного состояния в другое:

```
fsm_init_state <=
fsm_init_next_state;
```

Для того чтобы автомат в начальный момент времени был в известном состоянии, в блоке триггеров использована конструкция:

```
if (!fsm_init_start)
begin
fsm_init_state <= 6'b000000;
fsm_init_state[NOP_I] <= 1'b1;
end
```

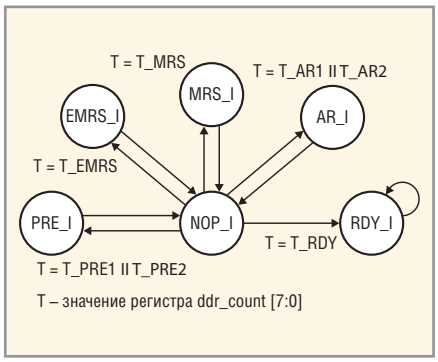

Рис. 2. Граф конечного автомата инициализации памяти

а в блоке защёлок:

```
fsm_init_next_state = 6'b000000;
fsm_init_out = NOP_CMD_I;
fsm_init_done = 1'b0;
```

Заметим, что в блоке защёлок применяются блокирующие операторы присваивания (=), а в блоке триггеров – неблокирующие (<=).

Входным сигналом, определяющим моменты перехода конечного автомата инициализации, является время, т.е. значения основного счётчика *(ddr_count)*. Граф состояний автомата изображён на рисунке 2. В начальный момент времени процесса инициализации автомат устанавливается в состояние *NOP_I*, при этом памяти посылается команда *NOP* (no operations). Затем через промежуток времени T_PRE1 автомат переходит в состояние *PRE_I* на время одного такта и посылает памяти команду *PREA* (precharge all banks) – зарядить ячейки всех четырёх банков. После чего автомат снова переходит в состояние *NOP_I*.

Следующий шаг – состояние *EMRS_I* с выходной командой *EMRS* (extended mode register set, установка дополнительного конфигурационного регистра). Значение регистра выставляется на адресные линии a12 – a0. Только два младших бита являются значимыми, остальные устанавливаются в нулевые значения. Бит a1 – мощность управляющего сигнала (drive strength) – тоже устанавливается в нулевое значение, что соответствует нормальной мощности. Бит a0 – включение/выключение модуля автоподстройки задержки (*DLL – delay locked loop*) – равен нулю (включено).

Через ещё один цикл *NOP_I* конечный автомат переходит в состояние *MRS_I*, в котором командой *MRS* (mode register set) устанавливается

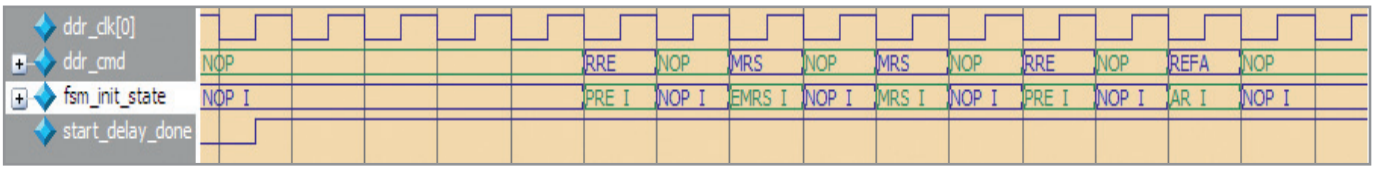

Рис. 3. Инициализация памяти

основной конфигурационный регистр динамической памяти . Значение регистра также выставляется на адресные линии a12 – a0. Биты a6 – a4 определяют латентность памяти – время между подачей команды чтения данных (*READ*) и появлением данных на выходных линиях DQ. При инициализации этот параметр устанавливается в значение 3'b010, что соответствует времени задержки в два периода тактового сигнала. Биты a2 – a0 определяют размер пакета данных при чтении/записи (burst length). Допустимыми значениями являются 2, 4 или 8 шестнадцатибитных слов. При инициализации эти биты устанавливаются равными 3'b001 (два слова).

После установки конфигурационного регистра конечный автомат снова переходит в состояние *PRE_I* и затем выполняет два цикла регенерации памяти в состоянии *AR_I*. Все переходы осуществляются через состояние *NOP_I*. Фрагмент процесса инициализации по результатам моделирования в программе ModelSim показан на рисунке 3.

Состояние *RDY_I* характеризует окончание процесса инициализации – сигнал *fsm_init_done* устанавливается в единичное значение, тем самым управление передаётся основному конечному автомату *ddr_fsm.v*:

```
fsm_init_state[RDY_I]:
begin
fsm_init_done = 1'b1;
fsm_init_next_state[RDY_I] =
1'b1;
end
```

Рассмотрим более подробно работу основного конечного автомата.

Основной конечный автомат

Автомат *ddr_fsm.v* относится к классу конечных автоматов Мили, т.е. выходной сигнал зависит как от состояния автомата, так и от входных сигналов. Стиль программирования – с двумя блоками *always*, бинарным описанием состояний автомата (binary encoded) и регистровым выходом (registered output).

Состояния динамической памяти в рабочем режиме описываются конечным автоматом, полная схема которого приведена в спецификации фирмы-производителя [2]. Автомат *ddr_fsm.v*, граф которого показан на рисунке 4, является упрощённой версией полной схемы и реализует только циклы записи, чтения и регенерации, а также функцию установки конфигурационного регистра.

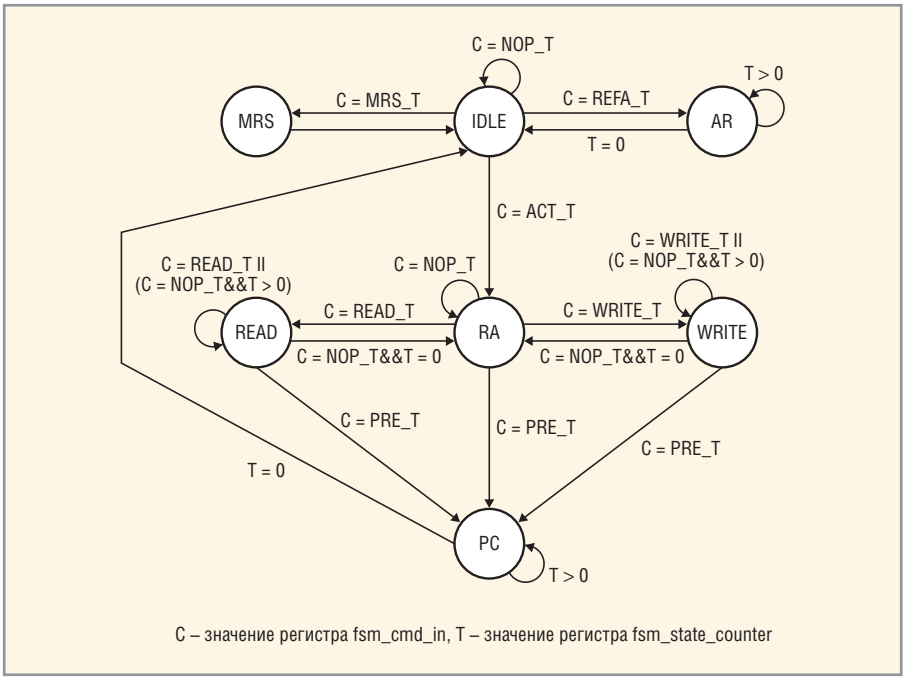

Рис. 4. Граф основного конечного автомата контроллера

Первоначально конечный автомат устанавливается в состояние *IDLE* и затем ожидает дальнейших команд от контролирующего блока. Выходной сигнал, т.е. команды для динамической памяти, зависит от состояния автомата и от значения счётчика времени *fsm_state_counter*. Значение счётчика устанавливается в момент переключения автомата из одного состояния в другое, который характеризуется разными значениями регистров *fsm_state* и *fsm_next_state* (текущее и следующее состояния соответственно):

```
case ({fsm_state,fsm_next_state})
{RA,WRITE}: fsm_state_counter <=
TRCDW;
{RA,READ}: fsm_state_counter <=
TRCDR;
default:
if (fsm_state_counter !==
16'h0000)
fsm_state_counter <=
fsm_state_counter - 1'b1;
endcase
```

Во все остальные моменты времени счётчик декрементируется. На рисунке 5 изображён результат моделирования цикла записи. При поступлении команды *ACT_T* на вход *fsm_cmd_i* (все команды поступают по положительному фронту сигнала *ddr_clk[0]*) регистр *fsm_next_state* устанавливается в значение RA (Row Active). При следующем отрицательном фронте тактового сигнала регистр *fsm_state* принимает значение *fsm_next_state*, то есть *RA*:

```
fsm_state <= fsm_next_state;
```

Одновременно с переходом автомата в состояние *RA* на выходе устанавливается команда *ACT*, которая активирует строку памяти, при этом адрес строки присутствует на адресных линиях a12 – a0.

В следующий положительный такт на вход автомата поступает команда *WRITE_T*, которая приводит к пере-

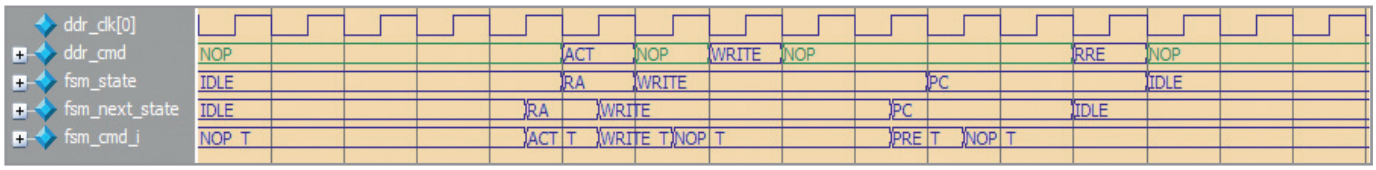

Рис. 5. Цикл записи

ключению регистра *fsm_next_state* и затем *fsm_state* в значение *WRITE*. Заметим, что при переходе из состояния *IDLE* в состояние *RA* счётчик *fsm_state_counter* сохраняет нулевое значение, поскольку в состоянии *RA* не требуются дополнительные такты ожидания до подачи команды активации строки. В то же время при переходе автомата в состояние *WRITE* необходима минимальная задержка *tRCD* (Row to Column Delay), поэтому при этом переходе счётчик *fsm_state_counter* устанавливается в значение *TRCDW*. Перейдя в состояние *WRITE*, конечный автомат выдаёт команду *NOP* до истечения времени задержки *tRCD*. После этого подаётся команда *WRITE* на один цикл, и затем снова вставляются команды *NOP* до окончания цикла записи. Команда *PRE_T* на входе *fsm_cmd_i* заканчивает цикл записи, после которого автомат снова переходит в состояние ожидания (*IDLE*).

Аналогичным образом происходит цикл чтения данных. Рассмотрим теперь логику работы блоков, посылающих команды конечному автомату и коммутирующих входные/выходные линии контроллера.

Блок управления конечным автоматом

Функцией этого блока является создание последовательности команд для конечного автомата на основании одной команды, полученной от декодера команд. Этот блок входит в состав файла *ddr_cntr.v*. Факт получения команды фиксируется установкой соответствующего флага:

```
case (ddr_cmd_in)      //Set flags
based on VJTAG commands
WR_1: vj_cmd_flag <= WR_1;
RD_1: vj_cmd_flag <= RD_1;
MRS_SETUP: vj_cmd_flag <=
MRS_SETUP;
endcase
```

Это делается для того, чтобы команда не была потеряна, если в момент передачи команды контроллер будет занят выполнением других действий, например регенерацией памяти. Затем на основании принятой команды и состояния автомата, текущего и последующего, принимается решение о подаче управляющих команд на вход конечного автомата:

```
case ({vj_cmd_flag, fsm_stat,
fsm_next_stat})//Decode VJTAG
commands
{WR_1, IDLE, IDLE}: fsm_cmd_i <=
ACT_T;  //Write command
       {WR_1, RA, RA}: fsm_cmd_i
<= WRITE_T;
```

После выполнения задачи флаг команды устанавливается в значение *NOP_VJ* – режим ожидания.

```
Vj_cmd_flag <= NOP_VJ; //Write
cycle is done
```

В функции этого блока также входит регенерация памяти, когда на входе блока нет никаких команд и значение основного счётчика превышает 30Ch, т.е. величину *tREFI*:

```
Default:
if (ddr_count > 32'h0000030C)
//Need to do refresh
case ({fsm_stat, fsm_next_stat})
{IDLE,IDLE}:    fsm_cmd_i <=
REFA_T;
default:        fsm_cmd_i <=
NOP_T;
endcase
```

Данный код посылает последовательность команд конечному автомату для выполнения цикла регенерации. Когда цикл регенерации практически завершился – текущее состояние автомата равно *AR*, а следующее состояние равно *IDLE*, – происходит сброс основного счётчика. Всё остальное время счётчик инкрементируется:

```
If ({fsm_stat, fsm_next_stat} !=
{AR,IDLE})
ddr_count <= ddr_count + 1'b1;
else
ddr_count <= ddr_count -
32'h0000030C;  // 30Ch is tREFI
time
```

Коммутация линий DQ и DQS

Линии DQS – это стробы данных. В памяти с 16-битными словами таких линий две – для младшего и старшего байтов. При записи данных они являются выходными для контроллера, при чтении – входными. В спецификации на память подробно указаны временные требования для этих сигналов.

При записи данных слова в памяти фиксируются по фронтам сигнала DQS. После регистрации команды записи *WRITE* динамической памятью, что происходит по положительному фронту синхросигнала (*ddr_clk[0]*), необходимо подождать время tDQSS, обычно равное одному периоду. После этого положительный фронт DQS записывает в память первое слово, а отрицательный фронт – второе. Так как линии DQS обычно находятся в высокоимпедансном состоянии, также задаётся время перехода этих линий в активное состояние при цикле записи и возврата в высокоимпедансное состояние после завершения записи. Эти временные параметры называются соответственно преамбулой *tWPRES* и постамбулой *tWPST*. Включение и выключение линий DQS контролируется сигналом *ddr_dqs_en*, который отвечает вышеперечисленным временным требованиям:

```
assign ddr_dqs = (ddr_dqs_en)?
{ddr_pll_in[0],ddr_pll_in[0]} :
2'bzz;
```

Поскольку номинальное значение tDQSS равно одному периоду и данные записываются по обоим фронтам синхросигнала, необходим дополнительный сигнал, сдвинутый на 90°, который заранее подготавливает данные для записи. С этой целью блок *PLL* вырабатывает сигнал *ddr_pll_90*. Логика подготовки данных для записи выглядит следующим образом:

```
case (ddr_pll_90)
1'b1: ddr_dq_drv[15:0] =
ddr_data_in[15:0];
1'b0: ddr_dq_drv[15:0] =
ddr_data_in[31:16];
endcase
```

Линии данных DQ, так же как и DQS, переводятся в активное и высокоимпедансное состояния сигналом *ddr_dqs_en*:

```
assign ddr_dq = (ddr_dqs_en)?
ddr_dq_drv: 16'hzzzz;
```

При чтении данных важным является параметр латентности, который определяет количество периодов после подачи команды чтения *READ*, через которые данные появляются на шине. При первоначальной конфигурации этот параметр устанавливается равным двум. Заметим, что в данной реализации контроллера в качестве строба чтения используется сигнал *ddr_pll_90*, а не *DQS*.

В контроллере также есть блок тестовой логики, который позволяет при отладке записывать значения основных регистров и затем передавать их пользователю для анализа. Частота записи значений регистров равна удвоенной частоте работы основной логики контроллера, т.е. 200 МГц.

Моделирование работы контроллера

Как уже было отмечено выше, моделирование проекта было выполнено в программе ModelSim, которая входит в состав пакета Quartus. Моделировалось поведения блока *ddr_cntr.v*. В оригиналы файлов для Quartus были внесены некоторые изменения, которые несущественны при моделировании, а именно:

- величина первоначальной задержки в файле *ddr_cntr.v* была изменена с 16'h4E20 на 16'h0010. Этот цикл ожидания необходим только для физического устройства и никак не влияет на результаты моделирования;
- по той же самой причине в файле *ddr_init_fsm.v* время *T_RDY* было изменено с 8'hDC на 8'h20.

Для удобства чтения результатов был создан командный файл *radix1.do*, который после выполнения его командой *do* позволяет заменить числовые значения векторов состояния контроллера на более удобные в восприятии символические, как на рисунках 3 и 5.

Заключение

Построение контроллера памяти DDR SDRAM на языке Verilog в сочетании с отладочной платой, такой как DK-START-3C25N, является хорошим учебным материалом для отработки навыков программирования на языках описания цифровых устройств (HDL) и освоения методов проектирования конечных автоматов.

Литература

1. *Гребенников А.* Интерфейс VJTAG для отладочной платы DK-START-3C25N. Современная электроника. 2010. № 9.
2. 256Mb DDR SDRAM Specification. Powerchip Semiconductor Corp.
3. Double Data Rate (DDR) SDRAM. Micron Technology, Inc.
4. *Clifford E.* Cummings. State Machine Coding Styles for Synthesis. Sunburst Design, Inc.
5. *Clifford E.* Cummings. The Fundamentals of Efficient Synthesizable Finite State Machine Design using NC-Verilog and BuildGates. Sunburst Design, Inc.

Новости мира News of the World

Датчане обзаведутся домашними ТЭЦ

Производство электрической и тепловой энергии без загрязнения воздуха и с минимальными выбросами CO_2 является очень важной научной и экономической проблемой. Весьма привлекательны с этой точки зрения твердооксидные топливные элементы (SOFC), которые уже 20 лет разрабатывает и испытывает Лаборатория технического университета Дании. И, наконец, сделан важный шаг на пути широкомасштабного внедрения этой технологии – университет подписал соглашение о сотрудничестве с компанией Topsoe Fuel Cell, которая разработала коммерческие ячейки топливных элементов и является партнёром компании Dantherm Power, производящей малые ТЭЦ. Благодаря этому коммерческому и научному союзу, уже в ближайшее время датчане смогут оснастить свои дома компактными и экологически чистыми генераторами тепла и электричества.

Твердооксидные топливные элементы производят электроэнергию и тепло с очень высокой эффективностью и практически без выбросов загрязняющих веществ, таких как оксиды серы и азота. Отдельные ячейки элементов сверхкомпактны, толщиной с лист бумаги, и каждый такой лист вырабатывает напряжение в 1 В. Для достижения желаемого напряжения и мощности достаточно просто собрать пакет из отдельных листов.

Датчане не видят перспективы в огромных центральных ТЭЦ и считают, что будущее за автономными генераторами энергии в каждом доме. Такие микроТЭЦ будут особенно эффективны в будущем, когда всё больше энергии будут поставлять возобновляемые источники энергии, такие как ветер и солнце, а домашняя энергостанция будет производить энергию в безветреную или облачную погоду.

Весной 2010 г. компания Dantherm Power уже выпустила несколько микроТЭЦ на топливных элементах, а в октябре 2010 г. были созданы ещё две системы для «профессиональных» пользователей, в частности, сантехников или электриков. Эти прототипы будут служить опытными стендами для проверки новой технологии. Они работают на природном газе и генерируют 1 кВт электрической мощности и 1 кВт тепла. В настоящее время микроТЭЦ по габаритам сравнима с большим холодильником.

В начале 2011 г. Dantherm Power рассчитывает иметь семь опытных микроТЭЦ, которые будут работать в течение всего отопительного сезона, включая весну 2011 г. В сентябре 2011 г. компания планирует выпустить 15 новых микроТЭЦ, в конструкции которых будет учтён опыт эксплуатации прототипов. Новые приборы будут настолько надёжны и просты в использовании, что их можно будет устанавливать в частных домах. В 2012 г. в продажу поступят микроТЭЦ, позволяющих обычным людям заменить свои старые печи и системы отопления на микроТЭЦ на твердооксидных топливных элементах.

По мнению специалистов компании, настоящая революция в области домашних электростанций произойдет в 2013–2015 гг., когда многие датские семьи будут иметь микроТЭЦ, занимающие места не больше, чем посудомоечная машина. Топливом первоначально будет природный газ, позже его можно будет заменить метанолом, сжиженным нефтяным газом или биотопливом.

http://www.cnews.ru

Пластмассовые температурные датчики

Корпорация TDK-EPC представила лёгкие пластмассовые температурные датчики фирмы Epcos для измерения температуры масла и охлаждающей жидкости в двигателях. Благодаря пластмассовой конструкции датчиков B58101A802A, они на 50% легче, чем изделия в металлическом корпусе, и имеют улучшенную герметизацию. В результате наблюдается улучшение вибрационных характеристик, увеличение срока службы, а также упрощение вторичного использования.

Пластмассовый корпус вокруг головки датчика спроектирован так, что обеспечивается очень хорошая теплопроводность. Это даёт сокращение времени реагирования и увеличение точности измерения температуры рабочих сред в автомобилях в диапазоне температур –40…+200°C.. Датчики типов B58101A802A предлагаются в диапазоне номинальных сопротивлений от 2 кОм до 1,4 МОм.

www.epcos.de