Память

Цифровое запоминающее устройство (далее – память) – это устройство, предназначенное для записи, хранения и считывания данных, представленных в цифровом виде. Существует большое количество различных классификаций запоминающих устройств: по типу доступа, энергозависимости, физическому принципу, возможности перезаписи, количеству портов и т.д. Память относится к последовательностным цифровым устройствам. На основе памяти можно создавать также генераторы функций и синтезаторы сигналов. В этой лекции/лабораторной работе нас будет интересовать память применительно к ПЛИС, физический принцип работы памяти рассматривается в другой части курса.

Виды памяти

Основное назначение памяти – хранение цифровых данных. Данные в памяти хранятся в ячейках, у каждой из которых есть свой адрес. Адрес ячейки – это уникальное число (номер), однозначно идентифицирующий данную ячейку (как городской адрес). Обычно цифровые данные хранятся в памяти в двоичном виде словами по несколько разрядов. Таким образом, адрес ячейки однозначно определяет некоторое количество бит, которые хранятся в данной ячейке.

По количеству информационных портов память может быть одно- и двухпортовой (в отдельных ситуациях – с бо́льшим количеством портов). Чем больше информационных портов имеет память, тем больше информационных слов можно записывать/считывать одновременно. Чаще всего используются одно- и двухпортовые модули.

По способу доступа память бывает с произвольным (RAM, random access memory) и с последовательным доступом (SAM, sequential access memory). В памяти с произвольным доступом в любой момент времени можно выполнять операции записи и чтение с любой ячейкой, т.е., другими словами, значение адреса можно менять в произвольном порядке. В памяти с последовательным доступом так делать нельзя. Между очередностью, в которой данные поступили в память, и очередностью, в которой данные считываются из памяти существует строго определенная зависимость. Например, слово, записанное в память раньше, на выход поступает тоже раньше. По такому принципу – “первым пришел, первым ушел” – работает память типа FIFO (first in, first out, принцип очереди). Другой пример памяти с последовательным доступом – память типа LIFO (last in, first out). Память типа LIFO работает по принципу “последним пришел, первым ушел” (стек). Таким образом, в памяти с последовательным доступом доступ к ячейкам происходит в определенном порядке.

Часто возникает необходимость хранить какие-либо данные, без возможности их обновления. Например, коэффициенты фильтров, таблицы перемежителей и др. В этом случае можно использовать память типа ROM (read only memory), не поддерживающую запись. Такой тип памяти относится к типу RAM и имеет следующую особенность: память не имеет входного информационного порта и входа разрешения записи (рис. 1 и 2).

Основными параметрами памяти являются информационная емкость, разрядность, а также потребляемая мощность и быстродействие.

Информационная емкость определяется максимальным количеством одновременно хранимых единиц информации. Разрядность задается количеством разрядов в запоминаемом слове данных. Быстродействие определяется разными параметрами, например, минимальным возможным интервалом времени между поступлением на вход нового адреса и появлением на выходе соответствующего слова данных.

Принцип работы памяти

Однопортовый модуль памяти RAM имеет один адресный вход ***A*** с разрядностью N_A, один одноразрядный вход разрешения записи *WE* (write enable), один информационный вход ***DI*** с разрядностью N_D, и один информационный выход ***DO*** с разрядностью N_D. В общем случае разрядность входных и выходных данных может не совпадать. Объем памяти составляет $2^{N_A} N_D$ бит.

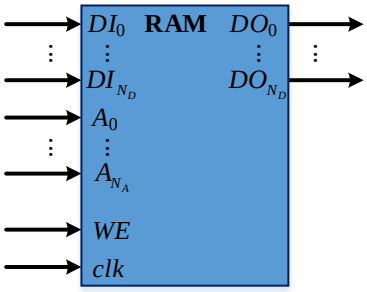

Рисунок 1. Однопортовая память со случайным доступом с возможностью записи

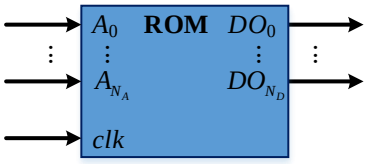

Рисунок 2. Однопортовая память со случайным доступом без возможности записи

При выставлении входа разрешения в единицу разрешается запись в память. Запись будет осуществляться в ячейку памяти, на которую указывает адрес. Также всегда (независимо от значения сигнала на входе разрешения записи) выполняется чтение из ячейки, на которую указывает текущий адрес.

В общем случае разрядность информационных входа и выхода может быть разной. Например, запись в память ведется словами по 8 бит, а считывание – по 16 бит. При этом адресный вход должен иметь такое количество разрядов, которого достаточно, чтобы адресовать все слова меньшей разрядности.

Очередность операций чтения и записи в память

В зависимости от того, какая процедура выполняется раньше, запись в ячейку или чтение из нее, различают следующие режимы работы памяти: *“write first”*,*“read first”* и *“no change”* (рис. 3-5). На рис. 3-5 сигналы на входах *we*, *DI* и *A*, смещены относительно фронта сигнала тактирования для того, чтобы подчеркнуть, что к моменту фронта все переходные процессы завершились. Также показана ненулевая задержка на выходе *DO* относительно соответствующих по номеру тактов сигналов на всех входах. Эта задержка обусловлена конечным временем срабатывания логических элементов в составе памяти, а также дополнительными триггерами на выходе. Отметим, что в общем случае указанная задержка может составлять несколько тактов.

В режиме *“write first”* при *we* = 1 новое слово на информационном входе одновременно записывается в память и выводится на выходной порт. Если *we* = 0, то в память ничего не записывается, а на выход выдается содержимое ячейки, адрес которой задан на адресном входе.

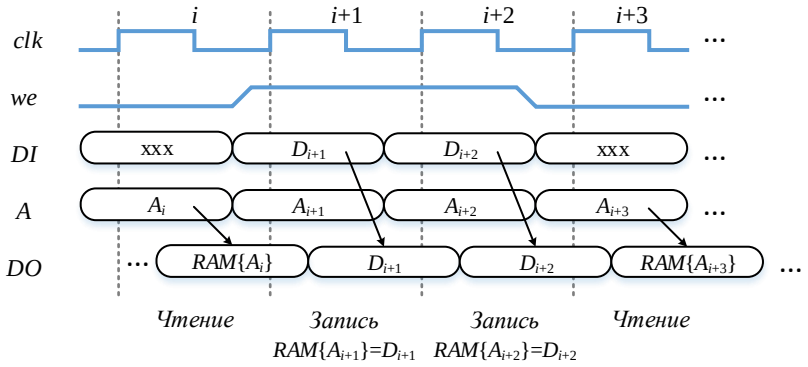

Рисунок 3. Временные диаграммы работы памяти в режиме "write first"

В режиме *"read first"* при *we* = 1 новое слово на информационном входе записывается в память, а на выходной порт выводится старое слово, записанное ранее по этому адресу (на рис. 4 "старое" отмечено символом *). Если *we* = 0, то в память ничего не записывается, а на выход выдается содержимое ячейки, адрес которой задан на адресном входе, как и в режиме *"write first"*.

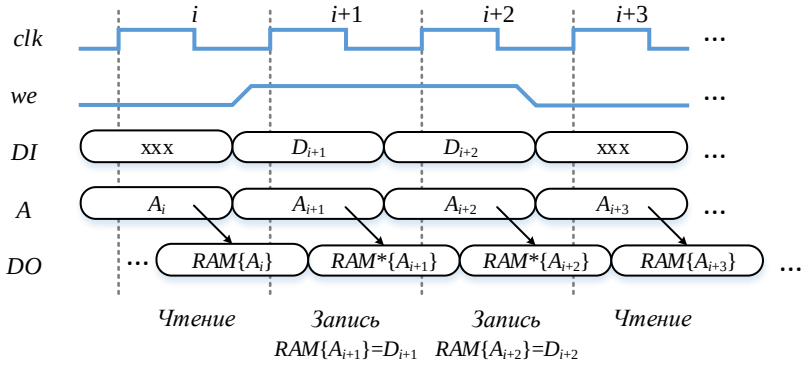

Рисунок 4. Временные диаграммы работы памяти в режиме "read first"

В режиме *"no change"* при записи в память слово на выходе не меняется (по сравнению с предыдущим тактом).

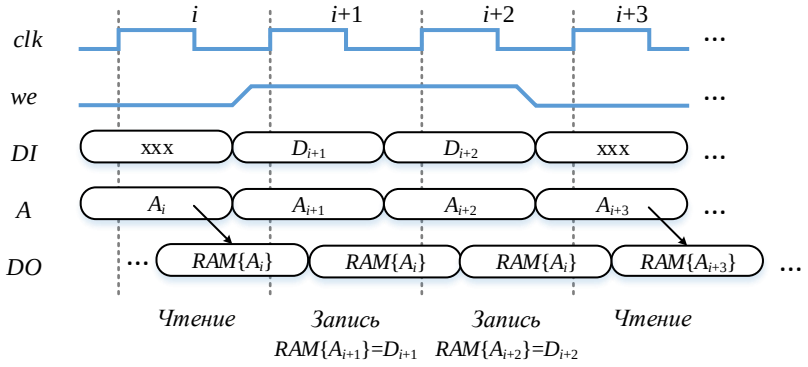

Рисунок 5. Временные диаграммы работы памяти в режиме "no change"

Двухпортовая память

В двухпортовой памяти можно записывать и считывать по два слова по разным адресам одновременно независимо друг от друга. Важно, что при этом адресное пространство общее для всех портов, а тактироваться порты могут разными тактовыми частотами (рис. 6).

В частных случаях оба порта двухпортовой памяти могут тактироваться одной тактовой частотой, иметь общие адресные входы и сигналы разрешения записи.

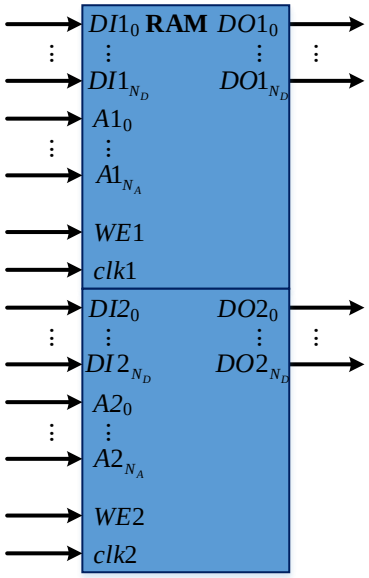

Рисунок 6. Двухпортовая память

Память с последовательным доступом типа FIFO

Память с последовательным доступом, работающая по принципу “первым пришел, первым ушел” очень широко применяется при разработке цифровых устройств на ПЛИС. Одним из основных применений FIFO является согласование тактовых доменов, т.е. перевод данных с одной тактовой частоты на другую. В более широком смысле FIFO применяются для согласования скоростей следования данных, неважно одинаковыми ли или разными тактовыми частотами они тактируются.

В процессе работы данные по мере поступления последовательно записываются в FIFO под управлением контроллера записи, а считываются – под управлением контроллера считывания. Эти контроллеры могут быть реализованы по-разному, но обязательно включают счетчики числа записанных и считанных слов.

Память типа FIFO удобно делать на основе двухпортовой памяти. При этом один порт предназначается только для записи данных в FIFO, а другой – только для чтения из FIFO.

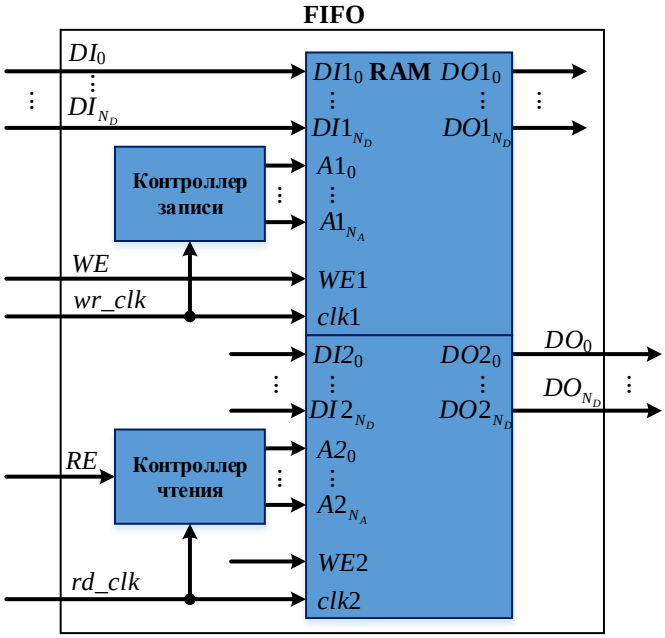

Рисунок 7. FIFO на основе двухпортовой памяти

На рис. 7 изображена память типа FIFO, построенная на основе двухпортовой памяти. Видно, что добавились контроллеры записи и чтения и не все входы и выходы двухпортовой памяти используются. Также добавился сигнал разрешения считывания – *RE*. Запись в FIFO осуществляется по сигналу тактирования *wr_clk* при *WE* = 1, а чтение – по *rd_clk* при *RE* = 1.

Отметим, что каждый контроллер работает в своем тактовом домене: контроллер записи по *wr_clk*, а контроллер чтения – по *rd_clk*.

Пример 1. Табличная реализация сложных функций

При реализации в FPGA сложных, не поддерживаемых на аппаратном уровне функций типа sin(*x*), sqrt(*x*) и др., удобно пользоваться тем, что количество возможных значений числа *x* с фиксированной точкой ограничено и может быть относительно невелико (сотни, единицы тысяч значений). В этом случае, можно заранее с заданной точностью рассчитать значения функций для каждого возможного значения *x* и занести его в таблицу. При реализации в FPGA эта таблица должна быть записана в память типа ROM, а число *x* будет выступать в роли адреса для этой памяти.

В качестве примера, рассмотрим табличную реализацию синусоидального колебания с частотой, равной 1/16 от частоты дискретизации.

Колебание в непрерывном виде выглядит так:

$$s(t) = \sin(2\pi f t). \quad (7)$$

Отсчеты сигнала (7) с частотой дискретизации F_s и интервалом дискретизации $\Delta t = 1/F_s$:

$$s_n = \sin(2\pi f n \Delta t) = \sin(2\pi f n / F_s) = |f = F_s/16| = \sin(2\pi n/16),\ n = 0, 1, 2, \ldots \quad (8)$$

Первые 16 отсчетов этого колебания составляют его период и приведены на рис. 3.

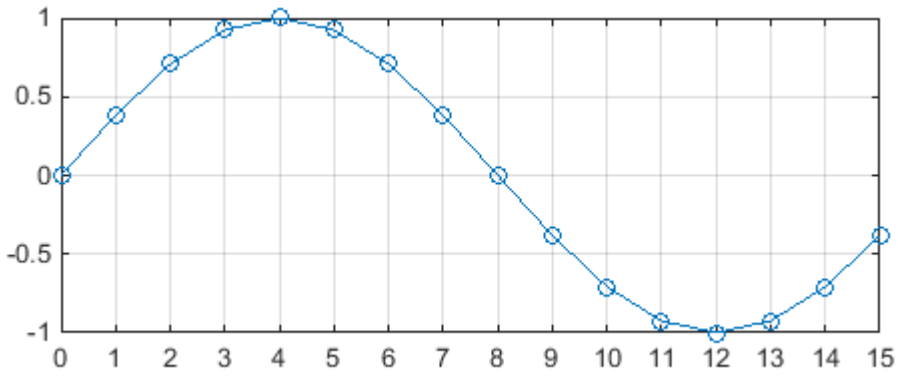

Рис. 8. Отсчеты одного периода синусоидального колебания с частотой $F_s/16$

Значения отсчетов одного периода синусоидального колебания с частотой $F_s/16$ приведены в табл. 1. Для того, чтобы получить двоичное представление значений отсчетов, исходные значения умножены на 127, округлены отбрасыванием знаков после запятой и переведены в двоичный 8-разрядный вид.

Таблица 4. Значения отсчетов одного периода синусоидального колебания с частотой $F_s/16$

n	≈sin(2πn/16)	≈floor(127*sin(2πn/16))	Двоичное представление отсчетов
0	0	0	0000 0000
1	0,3826	48	0011 0000
2	0,7071	89	0101 1001
3	0,9238	117	0111 0101
4	1	127	0111 1111
5	0,9238	117	0111 0101
6	0,7071	89	0101 1001
7	0,3826	48	0011 0000
8	0	0	0000 0000
9	-0,3826	-49	1100 1111
10	-0,7071	-90	1010 0110
11	-0,9238	-118	1000 1010
12	-1	-127	1000 0001
13	-0,9238	-118	1000 1010
14	-0,7071	-90	1010 0110
15	-0,3826	-49	1100 1111

Отметим, что табличное генерирование синусоидальных колебаний очень часто используется при разработке цифровых смесителей частоты (таких устройств, как DDS (direct digital synthesis), DDC/DUC (digital down/up converter), NCO (numeric controlled oscillator)).

Особенности FPGA. Блочная память

В современных FPGA память может быть реализована двумя способами: на логических ресурсах общего назначения (“распределенная память” – в ПЛИС Xilinx на SLICEM) и с помощью встроенных блоков памяти (“блочная память”). Второй способ позволяет реализовывать блоки памяти типовой конфигурации (однопортовая, двухпортовая RAM/ROM, FIFO), при этом доступный в ПЛИС объем памяти на встроенных блоках относительно большой (может исчисляться единицами мегабайт). Логические ресурсы общего назначения обеспечивают бо́льшую гибкость при создании модулей памяти (например, трехпортовая RAM), однако доступный в ПЛИС объем памяти на логике общего назначения значительно меньше суммарного в данной ПЛИС объема памяти на встроенных блоках. Важно отметить: что при реализации памяти на встроенных блоках все равно потребуются логические ресурсы общего назначения.

Объем блочной памяти в ПЛИС Xilinx серий Artix7, Kintex7 и Virtex7 составляет от 100 Кбайт до 8,5 Мбайт в зависимости от устройства, а распределенной памяти – от 5,5 Кбайт до 2,7 Мбайт.

Также, как и DSP элементы, блочная память организована по столбцам (рис. 9).

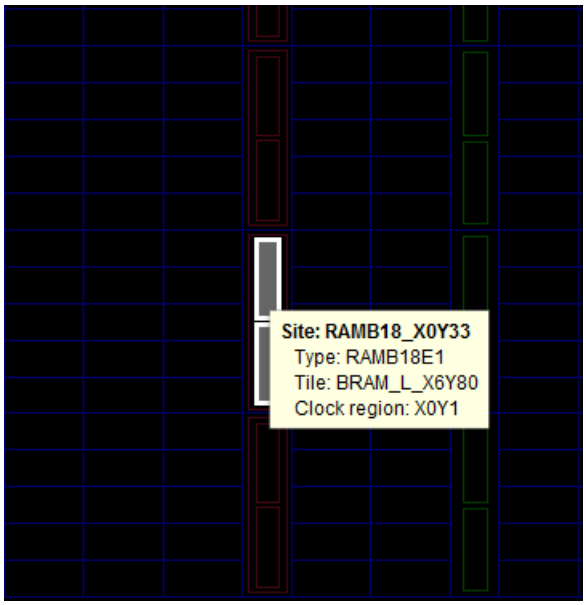

Рис. 9. Столбец блочной памяти (RAMB) в FPGA Xilinx 7 поколения