Stratix RAM WYSIWYG User Guide

Version 2.1

July 26, 2006

By

Altera Corporation

Table of Contents:

1. Stratix RAM Block

Note: More information about the RAM in the Stratix™ devices can be found at http://www.altera.com/products/devices/stratix/

Stratix has a heterogeneous memory architecture that includes memory blocks of 3 different sizes. The three memory blocks are named, in order of increasing size: 1) Small Embedded Array Block (M512), 2) Medium Embedded Array Block (M4K), 3) M-RAM Block (mega RAM). Although the features supported by each RAM block type are slightly different, we will use the same wysiwyg to model them. The wysiwyg primitive will be the super set to cover all modes supported by the 3 different blocks.

M512 can support single-port, Apex™ 20KE-style dual-port (1 read and 1 write) with mixed-width and ROM. M4K can support single-port, Apex 20KE-style dual-port with mixed-width, Mercury™ style true dual-port and ROM. The M-RAM can support single-port, Apex 20KE-style dual-port with mixed-width and Mercury-style true dual-port RAM.

Table 1 dictates all different modes supported by each block type:

Operation Mode	M512	M4K	M-RAM [1]
Single-port	512x1 64x9 256x2 32x18 128x4	4Kx1 512x9 2Kx2 256x18 1Kx4 128x36	64Kx9 8Kx72 32Kx18 4Kx144[4] 16Kx36
Simple dual-port (dual_port)	Wx1/RxN WxN/Rx1 Wx2/RxN WxN/Rx2 Wx4/Rx4 Wx9/Rx9 Wx18/Rx18 Wx4/Rx16 Wx16/Rx4 N is (1,2,4,8,16)	WxM/RxN or WxY/RxZ M,N is (1,2,4,8,16,32) Y,Z is (9, 18, 36)	RxM / WxN Rx144/Wx144 M is (9,18,36,72) N is (9,18,36,72)
True dual-port (bidir_dual_port)	N/A	AxM/BxN [3] or AxY/BxZ [3] M,N is (1,2,4,8,16), M ≥ N Y,Z is (9, 18), Y ≥ Z	AxM / BxN M is (9,18,36,72) N is (9,18,36,72)
ROM [2]	Same as single-port	Same as single-port	N/A

Table 1 Stratix RAM Configurations

Note:
1. There is no MIF support in M-RAM. The content can't be pre-initialized.
2. Since all inputs to Stratix RAM need to be registered, the ROM mode really means a pipelined ROM.

3. For user, the RAM is still symmetric because Megafunction will swap the ports when the given B port is wider than A port.
4. The x144 mode is actually supported by emulation through true dual-port mode. The Quartus® II development tool will pack two x72 atoms in true dual-port mode to support this mode. So, indeed each atom can't have more than 72 bits data bus.

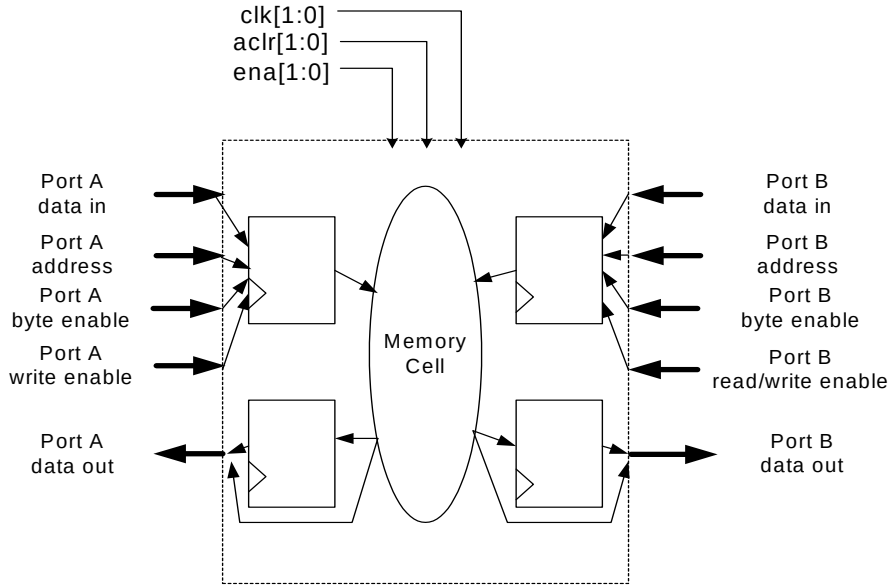

Figure 1: Stratix RAM

1.1 RAM Primitive

```
stratix_ram_block <block_name>
(
      .portadatain(<port A write data source bus>),
      .portaaddr(<port A addresses bus>),
      .portawe(<port A write-enable source>),
      .portbdatain(<port B write data source bus>),
      .portbaddr(<port B addresses bus>),
      .portbrewe(<port B read-enable/write-enable source>),
      .clk0(<clock source 0>),
      .clk1(<clock source 1>),
      .ena0(<clock enable for clock 0>),
      .ena1(<clock enable for clock 1>),
      .clr0(<clear source 0>),
      .clr1(<clear source 1>),
      .portabyteenamasks(<port A byte-enable mask source bus>),
      .portbbyteenamasks(<port B byte-enable mask source bus>),
```

```
        .portadataout(<port A read data output bus>)
        .portbdataout(<port B read data output bus>)
);
defparam <block_name>.operation_mode = <operation mode>;
defparam <block_name>.mixed_port_feed_through_mode = <mixed port
        feed through mode>;
defparam <block_name>.ram_block_type = <ram block type>;
defparam <block_name>.logical_ram_name = <logical RAM's name>;
defparam <block_name>.init_file = <name of the initialization
        file>;
defparam <block_name>.init_file_restructured = <name of the
        restructured initialization file, if necessary>
defparam <block_name>.init_file_layout = <layout of the
        initialization file>;
defparam <block_name>.data_interleave_width_in_bits = <data
        interleave width in bits>;
defparam <block_name>.data_interleave_offset_in_bits = <data
        interleave offset in bits>;
defparam <block_name>.port_a_logical_ram_depth = <port A depth of
        the logical RAM >;
defparam <block_name>.port_a_logical_ram_width = <port A width of
        the logical RAM >;
defparam <block_name>.port_a_data_in_clear = <port A data in
        clear>;
defparam <block_name>.port_a_address_clear = <port A address
        clear>;
defparam <block_name>.port_a_write_enable_clear = <port A write-
        enable clear>;
defparam <block_name>.port_a_byte_enable_clear = <port A byte-
        enable clear>;
defparam <block_name>.port_a_data_out_clock = <port A data out
        clock>;
defparam <block_name>.port_a_data_out_clear = <port A data out
        clear>;
defparam <block_name>.port_a_first_address = <port A starting
        address for this block>;
defparam <block_name>.port_a_last_address = <port A ending
        address for this block>;
defparam <block_name>.port_a_first_bit_number = <port A first
        logical bit position of this block>;
defparam <block_name>.port_a_data_width = <width of the port A
        data bus of this block>;
defparam <block_name>.port_b_logical_ram_depth = <port B depth of
        the logical RAM >;
defparam <block_name>.port_b_logical_ram_width = <port B width of
        the logical RAM >;
defparam <block_name>.port_b_data_in_clock = <port B data in
        clock>;
defparam <block_name>.port_b_data_in_clear = <port B data in
        clear>;
defparam <block_name>.port_b_address_clock = <port B address
        clock>;
```

```
defparam <block_name>.port_b_address_clear = <port B address
      clear>;
defparam <block_name>.port_b_read_enable_write_enable_clock =
      <port B read-enable/write-enable clock>;
defparam <block_name>.port_b_read_enable_write_enable_clear =
      <port B read-enable/write-enable clear>;
defparam <block_name>.port_b_byte_enable_clock = <port B byte-
      enable clock>;
defparam <block_name>.port_b_byte_enable_clear = <port B byte-
      enable clear>;
defparam <block_name>.port_b_data_out_clock = <port B data out
      clock>;
defparam <block_name>.port_b_data_out_clear = <port B data out
      clear>;
defparam <block_name>.port_b_first_address = <port B starting
      address for this block>;
defparam <block_name>.port_b_last_address = <port B ending
      address for this block>;
defparam <block_name>.port_b_first_bit_number = <port B first
      logical bit position of this block>;
defparam <block_name>.port_b_data_width = <width of the port B
      data bus of this block>;
```

1.2 RAM Input Signals

<*block_name*> is the unique identifier for this particular block. *This field is required.*

In ‘single-port’ or ‘rom’ modes, the B port cannot be used, therefore all the port B inputs are optional.

The ESB only supports two clocks, clock-enables and clears signals. M-RAM does not have clear control on the inputs, so all the clear parameters for the inputs are illegal if the block type is set to ‘M-RAM’.

The M512 doesn’t support byte enable, so the byte enable ports should be always disconnected if the block type is set to ‘M512’. Also the byte enable ports should be disconnected if the write port data width is less than 1 byte. And if the byte enable ports are connected, the corresponding port data width must be a multiple of byte (8 bit or 9 bit).

In simple dual-port mode, the M-RAM doesn’t have the read-enable control on the read port. The read-enable should be tied to VCC if block type is set to ‘M-RAM’.

.portadatain(<*data sources*>**),** is the port A write data input bus to this RAM block.

.portaaddr(<*addresses*>**),** are the address lines for port A. *They should be the same bus for all blocks of the same logical RAM.*

.portawe(<*write-enable source*>**),** is the (active-high) signal which makes the port A active for writing. *It should be the same signal for all blocks of the same logical RAM.*

.portbdatain(<*data sources*>**),** is the port B write data input bus to this RAM block.

.portbaddr(<*addresses*>**),** are the address lines for port B. *They should be the same bus for all blocks of the same logical RAM.*

.portbrewe(<*read-enable/write-enable source*>**),** is the (active-high) signal which makes the port B active for reading. *It should be the same signal for all blocks of the same logical RAM.*

Note: In bidir_dual_port mode, this port will be used as write-enable and it's 'active-high'.

.clk0(<*clock source*>**),** designates one of the clocks for the address, data, output, and enable registers. *This signal should be the same signal for all blocks of the same logical RAM.*

.clk1(<*clock source*>**),** designates one of the clocks for the address, data, output, and enable registers. *This signal should be the same signal for all blocks of the same logical RAM.*

.ena0(<*clock-enable 0*>**),** is the clock enable for .clk0. Only allowed when the .clk0 port on the RAM block is specified.

.ena1(<*clock-enable 1*>**),** is the clock enable for .clk1. Only allowed when the .clk1 port on the RAM block is specified.

.clr0(<*clear source*>**),** is one of the clear signals for the RAM.

.clr1(<*clear source*>**),** is one of the clear signals for the RAM.

.portabyteenamasks(<*byte masks*>**),** are the byte enable masks for the port A write port. *These ports are optional*. This is only allowed when the port A data width is a multiple of byte. If they are not connected, the mask should be considered as 1.

.portbbyteenamasks(<*byte masks*>**),** are the byte enable masks for the port B write port. *These ports are optional*. This is only allowed when the port B data width is a multiple of byte. If they are not connected, the mask should be considered as 1.

1.3 RAM Output Signals

.portadataout(<*data outputs*>**),** is the A port read data output bus of this RAM block.

.portbdataout(<*data outputs*>**),** is the B port read data output bus of this RAM block.

1.4 RAM Modes

There are two types of information fields: fields, which give Logical RAM-wide information and fields which are block-specific. An important note here is although the B port inputs can choose between clock0/clear0 and clock1/clear1, the inputs on the same port do not allow to use different clocks/clears. So if port B data in uses clock0, port B addresses can't use clock1. Similarly, if port B data in uses clear0, port B addresses can't use clear1 (but it's ok that B addresses do not use clear at all).

RAM-block wide information fields are as follows:

<*operation mode*> is one of *{single_port, dual_port, bidir_dual_port, rom}*. *This field is required. It should be the same for all blocks of the same logical RAM.*

<*mixed port feed through mode*> is one of *{ dont_care, old }*. *This field is optional.* It only makes sense in dual_port or bidir_dual_port modes. The field is used to dictate the behavior of 'read-during-write on different ports' at the same location with the same clock. When it's 'dont_care', it means the read output is 'unknown'. When it's 'old', it means the read output is the old data in the address before the write occurs. The default value is 'dont_care'.

Note: The M-RAM can only support 'dont_care' mode.

<*ram block type*>, is one of *{M512, M4K, M-RAM(a.k.a mega RAM) or auto}. This field is optional.* The default value is auto.

<*logical RAM's name*>*,* is the unique identifier for the corresponding logical RAM. *This field is required. It should be the same name for all blocks of the same logical RAM.*

<*name of the initialization file*>, is an identifier for the memory initialization file (.mif or .hex). *This field is optional (memory is not initialized). It should be the same file for all blocks of the same logical RAM.*

Note: M-RAM doesn't support MIF. If block type is 'M-RAM' (a.k.a mega RAM), the field should not be used.

<*layout of the initialization file*>*,* is one of *{Port_A, Port_B}. This field is optional.* The default is Port_A. It indicates how the memory initialization file is organized(.mif or .hex). If it's Port_A, the memory initialization file stores the memory as *port_a_logical_ram_depth* words with word size *port_a_logical_ram_width.*

<*data interleave width in bits*>, <*data interleave offset in bits*>, these two parameters, combined with the first bit parameter and the data width parameter, specifies what logical bits are contained in a RAM block. They are common for all possible views (portA read, portA write, portB read, portB write) and default to 1.

RAM-block wide port information fields are as follows:

<*port A depth of the logical RAM* >, represents the logical depth of the A port of the corresponding logical RAM. *This field is required. It should be the same value for all blocks of the same logical RAM.*

<*port A width of the logical RAM*>, represents the logical width of the A port of the corresponding logical RAM. *This field is required. It should be the same value for all blocks of the same logical RAM.*

Note: The input registers (data-in, write-enable, address) are always clocked by clk0 for port A. So there is no need to specify the clock parameter here for port A. Also, since the inputs for port A are always registered, it's illegal to have clk0 unconnected.

<*port A data in clear*> is one of *{clear0, none}*. Designates the asynchronous clear for the port A data input registers. *This field is optional*, and should be the same value for all blocks of the same logical RAM. Defaults to *none.*

<*port A address clear*> is one of *{clear0, none}*. Designates the asynchronous clear for the *port A* address registers. *This field is optional*, and should be the same value for all blocks of the same logical RAM. Defaults to *none.*

<*port A write-enable clear*> is one of *{clear0, none}*. Designates the asynchronous clear for the *port A* write-enable register. *This field is optional*, and should be the same value for all blocks of the same logical RAM. Defaults to *none.*

<*port A byte-enable clear*> is one of *{clear0, none}*. Designates the asynchronous clear for the *port A* byte-enable register. *This field is optional*, and should be the same value for all blocks of the same logical RAM. Defaults to *none.* This field should not be used if port A byte-enables are not connected.

<*port A data out clock*> is one of *{clock0, clock1, none}*. Designates the clock for the *port A* data output registers. *This field is optional*, and should be the same value for all blocks of the same logical RAM. Defaults to *none*.

<*port A data out clear*> is one of *{clear0, clear1,none}*. Designates the asynchronous clear for the *port A* data output registers. *This field is optional*, and should be the same value for all blocks of the same logical RAM. Defaults to *none.*

<*port B depth of the logical RAM* >, represents the logical depth of the B port of the corresponding logical RAM. *This field is optional. It should be the same value for all blocks of the same logical RAM.*

<*port B width of the logical RAM*>, represents the logical width of the B port of the corresponding logical RAM. *This field is optional. It should be the same value for all blocks of the same logical RAM.*

Note: The physical memory used by a logical RAM must add up the same from all the ports of the RAM. So if the port A mode is 4K x 1. The port B mode can be 128 x 32, but cannot be 128 x 16 since this is not an exact match.

<*port B data in clock*> is one of *{clock0, clock1}*. Designates the clock for the port B data input registers. *This field is optional*, and should be the same value for all blocks of the same logical RAM. If port B is used, this field is required.

<*port B data in clear*> is one of *{clear0, clear1, none}*. Designates the asynchronous clear for the port B data input registers. *This field is optional*, and should be the same value for all blocks of the same logical RAM. Defaults to *none.*

<*port B address clock*> is one of *{clock0, clock1}*. Designates the clock for the port B address registers. *This field is optional*, and should be the same value for all blocks of the same logical RAM. If port B is used, this field is required.

<*port B address clear*> is one of *{clear0, clear1, none}*. Designates the asynchronous clear for the *port B* address registers. *This field is optional*, and should be the same value for all blocks of the same logical RAM. Defaults to *none.*

<*port B read-enable/write-enable clock*> is one of *{clock0, clock1}*. Designates the clock for the *port B read-enable/write-enable* register. *This field is optional*, and should be the same value for all blocks of the same logical RAM. This field is required if port B is used.

<*port B read-enable/write-enable clear*> is one of *{clear0, clear1, none}*. Designates the asynchronous clear for the *port B read-enable/write-enable* register. *This field is optional*, and should be the same value for all blocks of the same logical RAM. Defaults to *none.*

<*port B byte-enable clock*> is one of *{clock0, clock1}*. Designates the clock for the *port B byte-enable* register. *This field is optional*, and should be the same value for all blocks of the same logical RAM. If port B byte-enables are connected, this field is required. Otherwise, this field should not be used.

<*port B byte-enable clear*> is one of *{clear0, clear1, none}*. Designates the asynchronous clear for the *port B* byte-enable register. *This field is optional*, and should be the same value for all blocks of the same logical RAM. Defaults to *none.* This field should not be used if port B byte-enables are not connected.

<*port B data out clock*> is one of *{clock0, clock1, none}*. Designates the clock for the *port A* data output registers. *This field is optional*, and should be the same value for all blocks of the same logical RAM. Defaults to *none*.

<*port B data out clear*> is one of *{clear0, clear1, none}*. Designates the asynchronous clear for the *port B* data output registers. *This field is optional*, and should be the same value for all blocks of the same logical RAM. Defaults to *none.*

The following information fields are block-specific:

<*port A starting address for this block*>, represents the port A starting address of this particular block. *This field is required.*

<*port A ending address for this block*>, represents the port A ending address of this particular block. *This field is required.*

<*port A first logical bit position of this block*> gives the first writing bit position of the port A data in bus of this particular block within the corresponding logical RAM. *This field is required.*

<*width of the port A data bus of this block*> gives the width of the port A data in bus of this particular block within the corresponding logical RAM. *This field is required.*

<*port B starting address for this block*>, represents the port B starting address of this particular block. *This field is required if port B is used.*

<*port B ending address for this block*>, represents the port B ending address of this particular block. *This field is required if port B is used.*

<*port B first logical bit position of this block*> gives the first writing bit position of the port B data in bus of this particular block within the corresponding logical RAM. *This field is required if port B is used.*

<*width of the port B data bus of this block*> gives the width of the port B data in bus of this particular block within the corresponding logical RAM. *This field is required if port B is used.*

Supported ram registering modes are different from Apex 20KE. Stratix RAM is always registered on input sides. This is shown in Table 2:

Table 2 -- Stratix RAM Registering Modes

Operation Mode	Write Logic	Read Logic	Data In	Data Out
Single Port	Reg.	Reg.	Reg.	Comb/Reg.
Simple Dual Port	Reg.	Reg.	Reg.	Comb/Reg.
True Dual Port	Reg.	Reg.	Reg.	Comb/Reg.
ROM	N/A	Reg.	N/A	Comb/Reg.

1.5 Polarities and Default Values

All signals are active high and all secondary inputs have programmable inversions. Upon power-up, all input registers will be 0 except that read-enable and byte-enable registers will be 1.

1.6 Example of RAM WYSIWYG Instantiation

Below is an example of a RAM WYSIWYG instantiation. It instantiates a dual-port RAM that 8 words deep and has 1-bit wide ports for both of the two ports.

```
stratix_ram_block bidir_dp_ram1_8x1_1
(
        .clk0(clk[0]),
        .portawe(a_we),
```

```
        .portaaddr(a_addr[2:0]),
        .portadatain(datain[0]),
        .portadataout(bidir_dp_ram1_8x1_out_a[0]),
        .portbrewe(b_we),
        .portbaddr(b_addr[2:0]),
        .portbdatain(datain[0]),
        .portbdataout(bidir_dp_ram1_8x1_out_b[0])
);
defparam bidir_dp_ram1_8x1_1.operation_mode = "bidir_dual_port";
defparam bidir_dp_ram1_8x1_1.logical_ram_name = "ram1_8x4";
defparam bidir_dp_ram1_8x1_1.port_a_logical_ram_depth = 8;
defparam bidir_dp_ram1_8x1_1.port_a_logical_ram_width = 4;
defparam bidir_dp_ram1_8x1_1.port_a_first_address = 0;
defparam bidir_dp_ram1_8x1_1.port_a_last_address = 7;
defparam bidir_dp_ram1_8x1_1.port_a_first_bit_number = 0;
defparam bidir_dp_ram1_8x1_1.port_a_data_width = 1;

defparam bidir_dp_ram1_8x1_1.port_b_logical_ram_depth = 8;
defparam bidir_dp_ram1_8x1_1.port_b_logical_ram_width = 4;
defparam bidir_dp_ram1_8x1_1.port_b_first_address = 0;
defparam bidir_dp_ram1_8x1_1.port_b_last_address = 7;
defparam bidir_dp_ram1_8x1_1.port_b_first_bit_number = 0;
defparam bidir_dp_ram1_8x1_1.port_b_data_width = 1;
defparam
bidir_dp_ram1_8x1_1.port_b_read_enable_write_enable_clock = \
"clock0";
defparam bidir_dp_ram1_8x1_1.port_b_address_clock = "clock0";
defparam bidir_dp_ram1_8x1_1.port_b_data_in_clock = "clock0";
```

2. Initialization File Format

The RAM WYSIWYG accepts two different initialization file formats. The first format is the well-known “Intel HEX file format”. The second format is the MIF format (Memory Initialization File).

2.1 Intel HEX File Format

The Intel HEX file is an ASCII text file with one HEX record per line. Each record is made up of six fields that are ordered in the following format:

<Record Marker><Record Length><Address><Record Type><Data Bytes><Checksum>

<Record Marker> is the first character of the line. It is always a colon (:) and used to identify the line as an Intel HEX record.

<Record Length> is a two-bit field that represents the number of data bytes in the record.

<Address> is a 4-bit field that represents the starting address for the subsequent data in this record.

<Record Type> is a two-bit field that represents the type of this HEX record. The 4 possible record types are

00 - data record
01 - end-of-file record
02 - extended segment address record
04 - extended linear address record

<Data Bytes> is the field that stores the data bytes for this record. A record can contain a multi-bytes data. The number of data bytes in the record is specified in the <Record Length> field.

<Checksum> is a 2-bit field that represents the checksum of this record. The checksum value is calculated by summing all the preceding data bytes (hexadecimal digit pair) in this record excluding the checksum byte itself, do the modulo of 256 and then take the two's complement.

Example record:

:04000000000000001FB

: -> Record marker

04 -> Record length

0000 -> Address

00 -> Record type

00000001 -> Data bytes

FB -> Checksum

The following is an example HEX file that describes the initialization content for a RAM of 8x32 (8 words and each word is 32-bit wide).

```
:0400000000000001fb
:0400010000000002f9
:0400020000000003f7
:0400030000000004f5
:0400040000000005f3
:0400050000000006f1
:0400060000000007ef
:0400070000000008ed
:00000001ff
```

Address (in decimal)	Content (in decimal)
0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	8

2.2 MIF File Format (Memory Initialization File)

MIF file is another format that can be used to specify the initial content of a memory block. It's an ASCII text file that contains the initial value for each address. The following is the MIF file format:

```
WIDTH = <Memory width>
DEPTH = <Memory depth>

ADDRESS_RADIX = <BIN/OCT/UNS/HEX>;
DATA_RADIX = <BIN/OCT/DEC/UNS/HEX>;

-- Specify values for addresses, which can be single address or range

CONTENT
BEGIN
        [<start address>..<end address>]    :    <content>;
        <single address>                    :    <content>;
END ;
```

BIN: Binary
OCT: Octal
DEC: Signed Decimal
UNS: Unsigned Decimal
HEX: Hexadecimal

The following is a sample MIF file that basically describes the same initialization file as in previous hex file example but in the MIF format.

```
WIDTH=32;
DEPTH=8;

ADDRESS_RADIX=UNS;
DATA_RADIX=HEX;

CONTENT BEGIN
   0  :  00000001;
   1  :  00000002;
   2  :  00000003;
   3  :  00000004;
   4  :  00000005;
   5  :  00000006;
   6  :  00000007;
   7  :  00000008;
END;
```

3. Advanced RAM partition parameters

It's possible user might want to specify a RAM that's bigger than any individual physical memory block's capacity. In that case, multiple WYSIWYGs can be instantiated and stitched to form the bigger RAM. When the user specified larger RAM is in mixed-width mode, the advanced parameters **data_interlave_width_in_bits** and **data_interleave_offset_in_bits** will need to be specified so Quartus can reconstruct the logical view of the big RAM for both ports. For mixed-width mode, in order to maintain the same logical RAM view from both ports, it is required to interleave the data (input and output) on the wider port. The width of the narrower port in both physical and logical views defines how the interleaving must be done.

The RAM WYSIWYG will have the following 2 parameters:

- **DATA_INTERLEAVE_WIDTH_IN_BITS :** width of narrower port in the wysiwyg view
    - **DATA_INTERLEAVE_OFFSET_IN_BITS :** width of narrowest port in logical user view

The following example shows how to partition the mixed-width big RAM into multiple WYSIWYGs.

3.1 A 8K bit logical RAM example with mixed-width Wx4/Rx16

Read view: 2k x 4

Write view: 512 x 16

The logical RAM consists of 8k bits (8192 bits), labeled L0 through L8191.

Logical read view words (x4)	Consists of logical bits:
Address 0	L3..L0
Address 1	L7..L4
…	

Logical write view words (x16)	Consists of logical bits:
Address 0	L15..L0
Address 1	L31..L16
…	

Since the total bits 8K is larger than a M4K block size, it will require stitching two M4Ks to form this user specified logical RAM. We can partition the RAM into 2 M4K (X and Y) and each is a Read 2Kx4/Write 512x8 block.

3.1.1 Read port physical view:

2k x 2 (each word is 2-bit) **M4K 'Y'** Logical read bit 3,2			2k x 2 (each word is 2-bit) **M4K 'X'** Logical read bit 1,0		
Address	Content		Address	Content	
0	Y0	(L3, L2)	0	X0	(L1, L0)
1	Y1	(L7, L6)	1	X1	(L5, L4)
2	Y2	(L11, L10)	2	X2	(L9, L8)
3	Y3	(L15, L14)	3	X3	(L13, L12)

To construct the logical read view from the physical read view; the bits are concatenated in a straightforward manner:

Logical read view words (x4)	Consists of physical bits:
Address 0	Y0, X0 (L3..L0)
Address 1	Y1, X1 (L7..L4)
…	

X<num> is the <num>th word in memory block X.

Y<num> is the <num>th word in memory block Y.

3.1.2 Write port physical view

On the write port, since the ratio between the write port width and read port width is 2, it means every two read words will make up 1 write word. The following table shows the physical view of each WYSIWYG on the write port.

512 x 8 (each word is 8-bit) **M4K 'Y'** Logical write bit 7,6,3,2			512x8 (each word is 8-bit) **M4K 'X'** Logical write bit 5,4,1,0		
Address	Content		Address	Content	
0	Y0[3..0]	(L7, L6, L3, L2)	0	X0[3..0]	(L5, L4, L1, L0)
1	Y1[3..0]	(L15, L14, L11, L10)	1	X1[3..0]	(L13, L12, L9, L8)
2	Y2[3..0]	(L23, L22, L19, L18)	2	X2[3..0]	(L21, L20, L17, L16)

The following table shows the logical view of the write port.

Logical write view words (x16)	Consists of physical bits:
Address 0	Y1[3..2], X1[3..2], Y1[1..0], X1[1..0], Y0[3..2], X0[3..2], Y0[1..0], X0[1..0]
Address 1	Y3[3..2], X3[3..2], Y3[1..0], X3[1..0], Y2[3..2], X2[3..2], Y2[1..0], X2[1..0]
…	

X<num1>[num2] is the <num2>th bit of the <num1>th word in block X

X<num1>[num2..num3] are bits [num2..num3] of the <num1>th word in block X

Y<num1>[num2] is the <num2>th bit of the <num1>th word in block Y

Y<num1>[num2..num3] are bits [num2..num3] of the <num1>th word in block Y

As above, to re-construct the logical write view, the data of each WYSIWYG in write view must be interleaved, as described by these parameters

DATA_INTERLEAVE_WIDTH_IN_BITS = 2, DATA_INTERLEAVE_OFFSET_IN_BITS = 4

The start bit parameter for each WYSIWYG gives the position of the LSB within the logical word, and each adjacent set of 2 (**DATA_INTERLEAVE_WIDTH_IN_BITS**) bit must be offset by 4 (**DATA_INTERLEAVE_OFFSET_IN_BITS**) to construct the logical view.

4. Internal Input registers

Input registers of a RAM can be referred to using the input register name extensions. Users can make assignments to the input registers by padding the name extensions to the RAM block instance name. The following is a list of RAM internal register name extensions.

Input Register	**Name extension**	**Requires index?**
Port A Write Enable	porta_we_reg	N

Port B Read Enable	portb_we_reg	N
Port B Write Enable	portb_re_reg	N
Port A Data In	porta_datain_reg	Y
Port B Data In	portb_datain_reg	Y
Port A Address Register	porta_address_reg	Y
Port B Address Register	portb_address_reg	Y
Port A Byte Enable Register	porta_bytena_reg	Y
Port B Byte Enable Register	portb_bytena_reg	Y

The format of the register name is *<ramblock_instance_ name>~<register_name>[<index>]*.

The “index” is not required for read enable registers and write enable registers.
E.g., *ram~porta_we_reg*
The “index” is required for data-in registers, address registers, and byte enable registers.
E.g., *ram~porta_address_reg9*