Inferring Microchip SmartFusion2 RAM Blocks

Synopsys® Application Note, April 2021

The Synplify Pro® synthesis tool now supports Microchip® SmartFusion2 devices and automatically infers the RAM1K18 and RAM64X18 RAM macros that these architectures include. This application note starts with a general description of the SmartFusion2 RAM block components, and then describes how to infer them using the Synplify Pro software.

See the following topics for details:

- SmartFusion2 RAM Blocks, on page 2
- Inferring SmartFusion2 RAM Blocks, on page 3
- Controlling Inference with the syn_ramstyle Attribute, on page 4
- Using BLK Signal to Reduce Power Consumption, on page 6
- Read/Write Address Collision Checks, on page 6
- RAMINDEX Property Switch, on page 8
- Single-Port RAM Coding Style Examples, on page 8
- Dual-Port RAM Coding Style Examples, on page 33
- True Dual-Port RAM Coding Style Examples, on page 52
- Multiport RAM Examples, on page 65
- Current Limitations, on page 69

SmartFusion2 RAM Blocks

SmartFusion2 devices support two types of RAM macros: RAM1K18 and RAM64X18.

RAM1K18

Here are the main characteristics of the RAM1K18 memory block:

- Contains 18,432 memory bits.
- Has two independent data ports, A and B.
- For dual-port mode, both ports of the RAM1K18 block have word widths less than or equal to 18 bits.
- Can infer a single-port, simple dual-port, and true dual-port RAM.
- For two-port mode, port A is the read port and port B is the write port.
- Has as optional pipelined register at the read data port.
- Has synchronous read and write operations.
- Does not allow read and write operations on the same location at the same time. It has no collision prevention and detection.
- Has a feedthrough write mode available to enable immediate access to the write data.
- Allows read from both ports at same location.

RAM64X18

Here are the main characteristics of the RAM64X18 memory block:

- Has two independent read data ports A and B, and one write data port C.
- Has write operations that are always synchronous.
- For both read data ports, the address can be set as synchronous or asynchronous.
- Has two read data ports that have output registers for pipelined mode operation.
- Allows read from both ports A and B at the same location.
- Does not allow read and write on the same location at the same time. It has no collision prevention or detection.
- Allows read from both ports at same location.

Inferring SmartFusion2 RAM Blocks

The Synplify Pro software identifies the RAM structure from the RTL and implements a SmartFusion2 RAM1K18 or RAM64X18 block.

RAM Mapping

The tool implements the RAM based on the following criteria:

1. For true dual-port synchronous read memory, the Synplify Pro software maps to the RAM1K18 block, regardless of the memory size.
2. For simple dual-port, single-port, or three-port synchronous memory, the software uses memory size to determine how RAM is mapped:

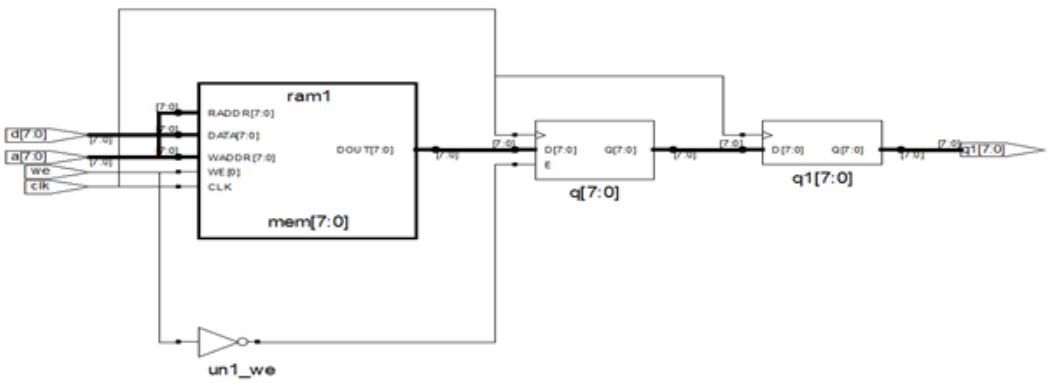
Memory Size	RAM Mapping
4608 bits or more	RAM1K18
More than 12 bits and less than 4608 bits	RAM64X18
Less than or equal to 12 bits	Registers

3. For simple dual-port, single-port, or three-port asynchronous memory, if the size of memory is 12 bits or more, the software maps to RAM64X18. Otherwise, the synthesis tool maps to registers.

You can override this default behavior using the syn_ramstyle attribute. See *Controlling Inference with the syn_ramstyle Attribute* , on page 4.

Pipelined Register Packing

The Synplify Pro tool can extract a pipelined register at the output of the block RAM and pack it in the RAM1K18. The RTL view below shows the pipelined register.



The pipelined register q1 [7:0] optionally can have asynchronous/synchronous reset or clock enable. The software does not pack pipelined register q1 [7:0] if the register q [7:0] has an asynchronous/synchronous reset.

Controlling Inference with the syn_ramstyle Attribute

Use the syn_ramstyle attribute to manually control how the mapping of SmartFusion2 RAM blocks:

To Map to ...	syn_ramstyle Value
RAM1K18	lsram
RAM64x18	uram
Registers	registers

You can apply the attribute globally, or to an individual RAM instance or module.

Here are some additional considerations when using the syn_ramstyle attribute:

- If your RTL code includes a true dual-port synchronous read memory, you cannot use syn_ramstyle = "uram" to infer RAM64X18, because true dual-port mode is not supported. In this case, the Synplify Pro software ignores the attribute setting and infers RAM1K18.
- If your RTL code includes asynchronous memory, you cannot use syn_ramstyle = "lsram" to infer RAM1K18, because asynchronous memory is not supported. In this case, the Synplify Pro software ignores the attribute and infers RAM64X18.
- When you do not want to use RAM resources, use registers as the attribute value, and apply it on the RAM instance name or the signal driven by the RAM.
- The syn_ramstyle attribute also has these valid values: no_rw_check and rw_check. By default, the synthesis tool does not generate glue logic for read/write address collisions. Use syn_ramstyle="rw_check" to insert glue logic for read write address collision. Use syn_ramstyle="no_rw_check" to prevent glue logic insertion. See *Read/Write Address Collision Checks*, on page 6 for more information.

Constraint File Syntax and Example

This is the syntax to add the attribute in a constraint file,. You can apply it on individual modules or globally:

define_attribute {*signalName***[***bitRange***]} syn_ramstyle {***string***}**

define_global_attribute syn_ramstyle {*string***}**

When editing a constraint file to apply the syn_ramstyle attribute, be sure to include the range of the signal with the signal name. For example:

```
define_attribute {mem1[7:0]} syn_ramstyle {registers};
define_attribute {mem2[7:0]} syn_ramstyle {lsram};
define_attribute {mem3[7:0]} syn_ramstyle {uram};
```

Verilog Syntax and Example

object **/* synthesis syn_ramstyle** = **"string" */ ;**

object is a register definition signal, and he data type is string.

```
module ram4 (datain,dataout,clk);
output [31:0] dataout;
input clk;
input [31:0] datain;
reg [7:0] dataout[31:0] /* synthesis syn_ramstyle="uram" */;
// Other code
```

VHDL Syntax and Example

attribute syn_ramstyle of object : *objectType* **is "string" ;**

object is a signal that defines a RAM or a label for a component instance. Data type is string.

```
library ieee;
use ieee.std_logic_1164.all;
library synplify;

entity ram4 is
port (d : in std_logic_vector(7 downto 0);
addr : in std_logic_vector(2 downto 0);
we : in std_logic;
clk : in std_logic;
ram_out : out std_logic_vector(7 downto 0) );
end ram4;

architecture rtl of ram4 is
type mem_type is array (127 downto 0) of std_logic_vector
   (7 downto 0);
signal mem : mem_type;
```

```
-- mem is the signal that defines the RAM

attribute syn_ramstyle : string;
attribute syn_ramstyle of mem : signal is "lsram";
-- Other code
```

Using BLK Signal to Reduce Power Consumption

By default, the tool fractures wide RAMs by splitting the data width to improve timing.

From Synplify® Pro 2014.09M-SP2 onwards, the tool is enhanced to use BLK pin of the RAM for reducing power consumption, by fracturing wide RAMs on the address width.

To enable this feature, set global option low_power_ram_decomp 1 in the project file (*.prj). The tool uses the global option to fracture wide RAMs on the address width, to infer RAM in the low power mode. The tool uses the BLK pin to select RAM for a particular address and OR gates at the output to select an output from RAM blocks.

The control of the individual RAM inference to turn on or turn off low power mode is supported through the synthesis attribute syn_ramstyle.

Add syn_ramstyle = "low_power" to turn on low power inference if the global option is set to off.
Add syn_ramstyle = "no_low_power" to turn off low power inference if the global option is on.

Read/Write Address Collision Checks

The Synplify Pro synthesis software does not check for read/write address collisions when it infers RAM. The tool does not insert glue logic around RAM to prevent read/write address collisions during write operations.

If read and write to the same address occurs simultaneously in your design, explicitly specify checking by setting the syn_ramstyle="rw_check" attribute or enabling the Read Write Check on RAM option on the Implementation Options Device panel. The tool then inserts glue logic around the RAM to prevent read/write address collision during the write operation, while retaining the RTL behavior.

The RAM has different operating modes, which determine behavior when collisions occur:

Write-First mode	Write operations precede read when a collision occurs. Data is first written into memory and then the same data is read.
Read-First mode	Read operations precede write when a collision occurs. Old data is read first and then new data is written into memory.
No change mode	The output of the RAM does not change when a collision occurs.

If the Read Write Check on RAM option is enabled, the tool behaves as follows:

- The SmartFusion2 RAM1K18 architecture supports Write-First mode (active feedthrough mode) and No Change mode (disabled feedthrough mode), when used as single-port RAM. When a single-port RAM with Write-First mode is mapped to RAM1K18, there is no glue logic around the RAM.
- RAM written in Read-First mode is mapped to RAM64X18, because it supports asynchronous read mode. No glue logic is created.
- When you try to map Read-First mode RAM to RAM1K18 using the syn_ramstyle="lsram" attribute, the attribute value is ignored and the RAM is mapped to registers.
- For No Change mode, no glue logic is created because the RAM output does not change when a collision occurs.
- For simple dual-port and true dual-port RAM in Write-First mode, glue logic is created. Glue logic is also created for single-port RAM when it is mapped to RAM64K18.
- If a read/write check creates glue logic, the pipelined register cannot be packed into the block RAM.

RAMINDEX Property Switch

To disable generation of the RAMINDEX property in RAM1K18_RT and RAM64X18_RT RAM blocks, add the `disable_ramindex` switch to the project file (prj):

```
Usage (disable RAMINDEX): set_option  -disable_ramindex 1
```

Single-Port RAM Coding Style Examples

Here are examples of coding styles that infer RAM1K18 and RAM64X18 RAM blocks for single-port (SP) RAM.

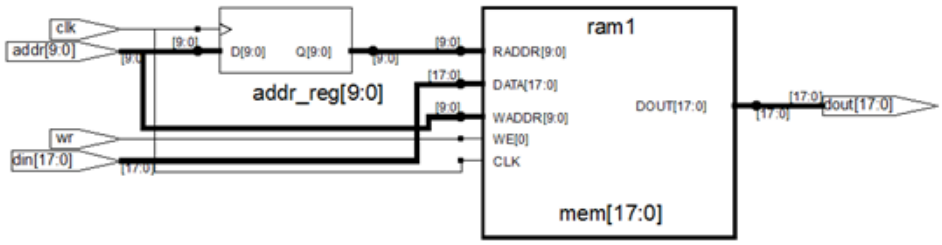
- Example 1: Single-Port RAM, RAM1K18 Write-First Mode, on page 9
- Example 2: Single-Port RAM, RAM64X18 Write-First Mode, on page 10
- Example 3: Single-Port RAM with Pipelined Register, RAM1K18 Write-First Mode, on page 11
- Example 4: Single-Port RAM with Pipelined Register, RAM64X18 Write-First Mode, on page 12
- Example 5: SP RAM with One Pipelined Register on Read Port, No Change Mode, on page 13
- Example 6: SP RAM with One Pipelined Register on the Read Port, Write-First Mode, on page 14
- Example 7: Single-Port RAM with One Pipelined Register on the Read Port, on page 16
- Example 8: Synchronous Read Without Pipelined Register, No Change Mode, on page 17
- Example 9: SP RAM with Asynchronous Read, RAM64X18 Read-First Mode, on page 18
- Example 10: Asynchronous Read with Pipelined Register, RAM64X18 Read-First Mode, on page 19
- Example 11: Single-Port RAM, RAM1K18 No Change Mode, on page 20
- Example 12: Single-Port RAM with Output Registers (VHDL), on page 21
- Example 13: Single Port RAM with Asynchronous Read (VHDL), on page 23
- Example 14: SP RAM with Synchronous Reset for Pipelined Register, RAM64x18, on page 24
- Example 15: Synchronous Reset for Pipelined Register and R/W Check, RAM64x18, on page 25
- Example 16: Single Port RAM1K18 (VHDL), on page 26

- Example 17: Single Port RAM with syn_ramstyle = "low_power" and Global Power Option Off, on page 29
- Example 18: Single Port RAM with syn_ramstyle = "no_low_power" and Global Power Option On, on page 31

Example 1: Single-Port RAM, RAM1K18 Write-First Mode

The following design is a single-port RAM with synchronous read/write. The same address is used for read and write operations in Write-First mode. The FPGA synthesis tool infers SmartFusion2 RAM1K18.

```
module ram_singleport_addreg (clk,wr,addr,din,dout);

input clk;
input [17:0] din;
input wr;
input [9:0] addr;
output [17:0] dout;

reg [9:0] addr_reg;
reg [17:0] mem [0:1023] ;

assign dout = mem[addr_reg];

always@(posedge clk)
begin
   addr_reg <= addr;
      if(wr)
         mem[addr]  <= din;

end
endmodule
```



Resource Usage Report for ram_singleport_addreg

This section of the log file (srr) shows resource usage details.

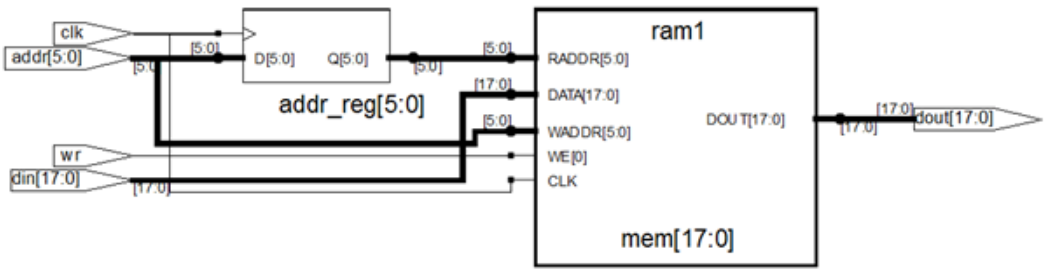
Mapping to part: m2s050tfbga896std
Cell usage:
RAM1K181 use

Sequential Cells:
SLE 0uses

Example 2: Single-Port RAM, RAM64X18 Write-First Mode

The following design is a single-port RAM with synchronous read/write. The same address is used for read and writes operation in Write-First mode. The FPGA synthesis tool infers SmartFusion2 RAM64X18.

```
module ram_singleport_addreg (clk,wr,addr,din,dout);

input clk;
input [17:0] din;
input wr;
input [5:0] addr;

output [17:0] dout;

reg [5:0] addr_reg;
reg [17:0] mem [0:64] ;

assign dout = mem[addr_reg];

always@(posedge clk)
begin
   addr_reg <= addr;
      if(wr)
         mem[addr]  <= din;

end
endmodule
```



Resource Usage Report for ram_singleport_addreg

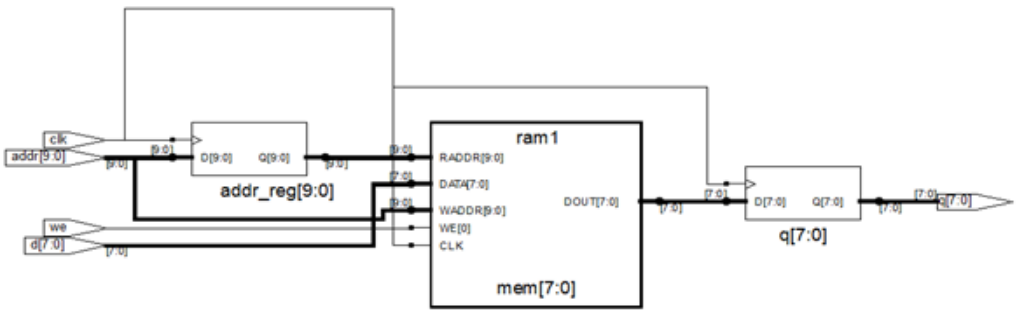
Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT1 use
RAM64x181 use

Sequential Cells:
SLE 0 uses

Example 3: Single-Port RAM with Pipelined Register, RAM1K18 Write-First Mode

The following design is a single-port RAM with one pipelined register on the read port in Write-First mode. The FPGA synthesis tool infers SmartFusion2 RAM1K18.

```
module ram_singleport_pipereg (clk,we,addr,d,q);

input [7:0] d;
input [9:0] addr;
input clk, we;

reg [9:0] addr_reg;
output reg [7:0] q;

reg [7:0] mem [1023:0] ;

always @(posedge clk) begin
addr_reg <= addr;
if(we)
mem[addr] <= d;
end
always @ (posedge clk )
begin

q <= mem[addr_reg];
end
endmodule
```



Resource Usage Report for ram_singleport_pipereg

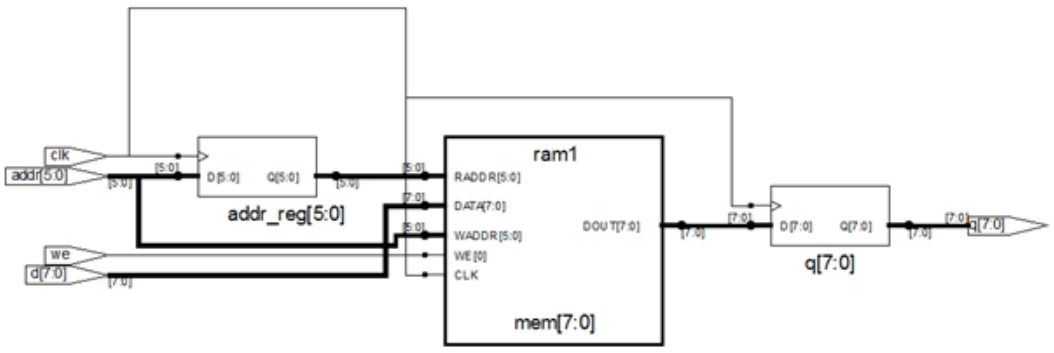
Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT1 use
RAM1K181 use

Sequential Cells:
SLE0 uses

Example 4: Single-Port RAM with Pipelined Register, RAM64X18 Write-First Mode

The following design is a single-port RAM with one pipelined register on the read port in Write-First mode. The FPGA synthesis tool infers SmartFusion2 RAM64X18.

```
module ram_singleport_pipereg(clk,we,addr,d, q);

input [7:0] d;
input [5:0] addr;
input clk, we;

reg [5:0] addr_reg;
output reg [7:0] q;

reg [7:0] mem [63:0] ;

always @(posedge clk) begin
addr_reg <= addr;
if(we)
mem[addr] <= d;
end

always @ (posedge clk )
begin

q <= mem[addr_reg];
end
endmodule
```



Resource Usage Report for ram_singleport_pipereg

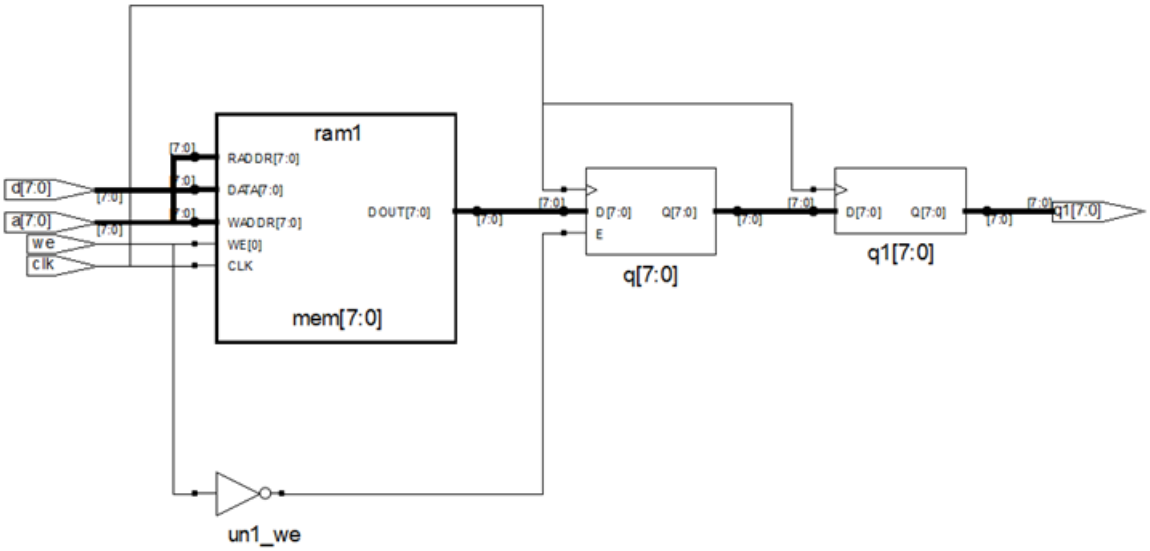
Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT 1 use
RAM64x18 1 use

Sequential Cells:
SLE0 uses

Example 5: SP RAM with One Pipelined Register on Read Port, No Change Mode

The following design is a single-port RAM with one pipelined register on the read port (sync-sync), with No Change mode. The FPGA synthesis tool infers SmartFusion2 RAM64X18. The pipelined register is mapped outside the RAM along with logic for No Change mode.

```
module ram_singleport_pipereg(clk,we,a,d,q1);

input [7:0] d;
input [7:0] a;
input clk, we;

reg [7:0] q;
output [7:0] q1;

reg [7:0] q1;
reg [7:0] mem [255:0];

always @(posedge clk)
   begin
      if(we)
         mem[a] <= d;
      else
         q <= mem[a];
   end
always @ (posedge clk)
   begin
      q1 <= q;
   end
endmodule
```



Resource Usage Summary for ram_singleport_pipereg

Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT1 use
RAM64x182 uses
CFG11 use
CFG38 uses

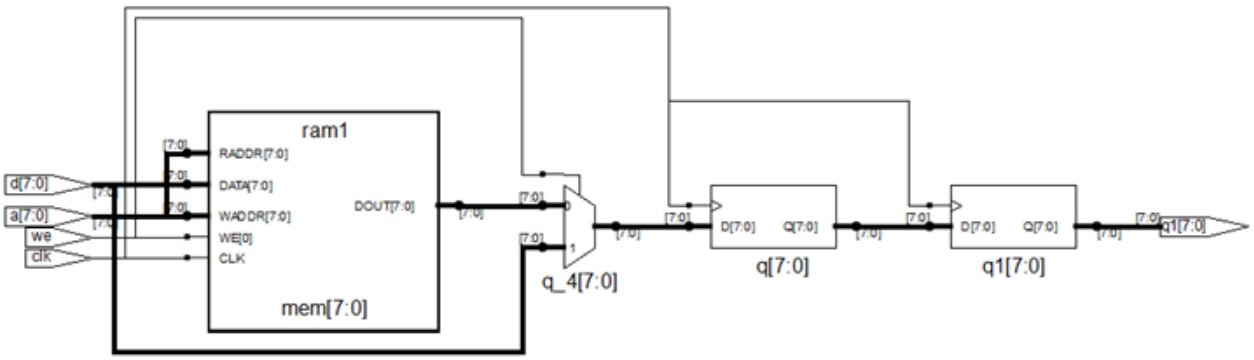
Sequential Cells:
SLE18 uses

Example 6: SP RAM with One Pipelined Register on the Read Port, Write-First Mode

The following design is a single-port RAM with one pipelined register on the read port (sync-sync) with Write-First mode. The FPGA synthesis tool implements two SmartFusion2 RAM64X18 blocks.

```
module ram_singleport_pipereg(clk,we,a,d,q1);
input [7:0] d;
input [7:0] a;
input clk, we;

reg [7:0] q;
output [7:0] q1;

reg [7:0] q1;
reg [7:0] mem [255:0];
```

```
always @(posedge clk)
   begin
      if(we)
         mem[a] <= d;
   end

always @ (posedge clk)
   begin
      if(we)
         q <= d;
      else
         q <= mem[a];
         q1 <= q;
   end
endmodule
```



Resource Usage Summary for ram_singleport_pipereg

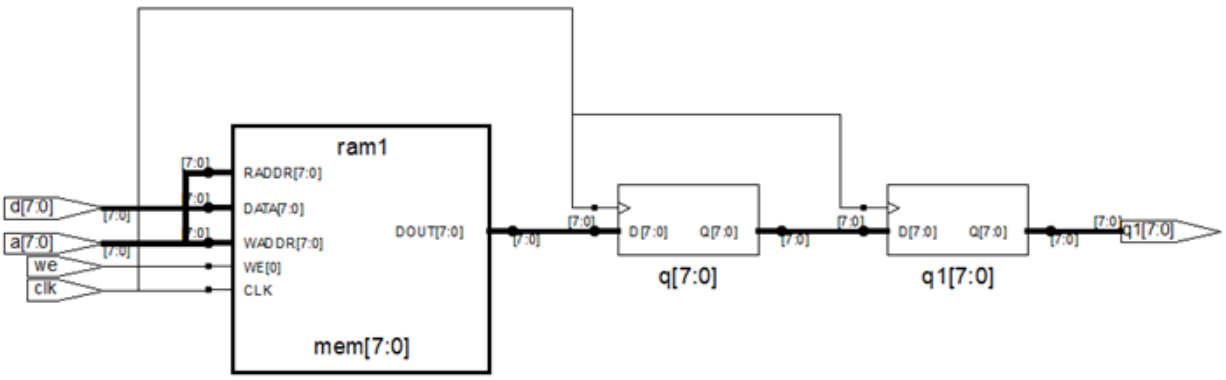
Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT1 use
RAM64x182 uses

Sequential Cells:
SLE0 uses

Example 7: Single-Port RAM with One Pipelined Register on the Read Port

The following design is a single-port RAM with one pipelined register on the read port (sync-sync). The FPGA synthesis tool implements two SmartFusion2 RAM64X18 blocks.

```
module ram_singleport_pipereg(clk,we,a,d,q1);

input [7:0] d;
input [7:0] a;
input clk, we;

reg [7:0] q;
output [7:0] q1;

reg [7:0] q1;
reg [7:0] mem [255:0]

always @(posedge clk)
   begin
      if(we)
         mem[a] <= d;
   end

always @ (posedge clk)
   begin
      q <= mem[a];
      q1 <= q;
endmodule
```



Resource Usage Summary for ram_singleport_pipereg

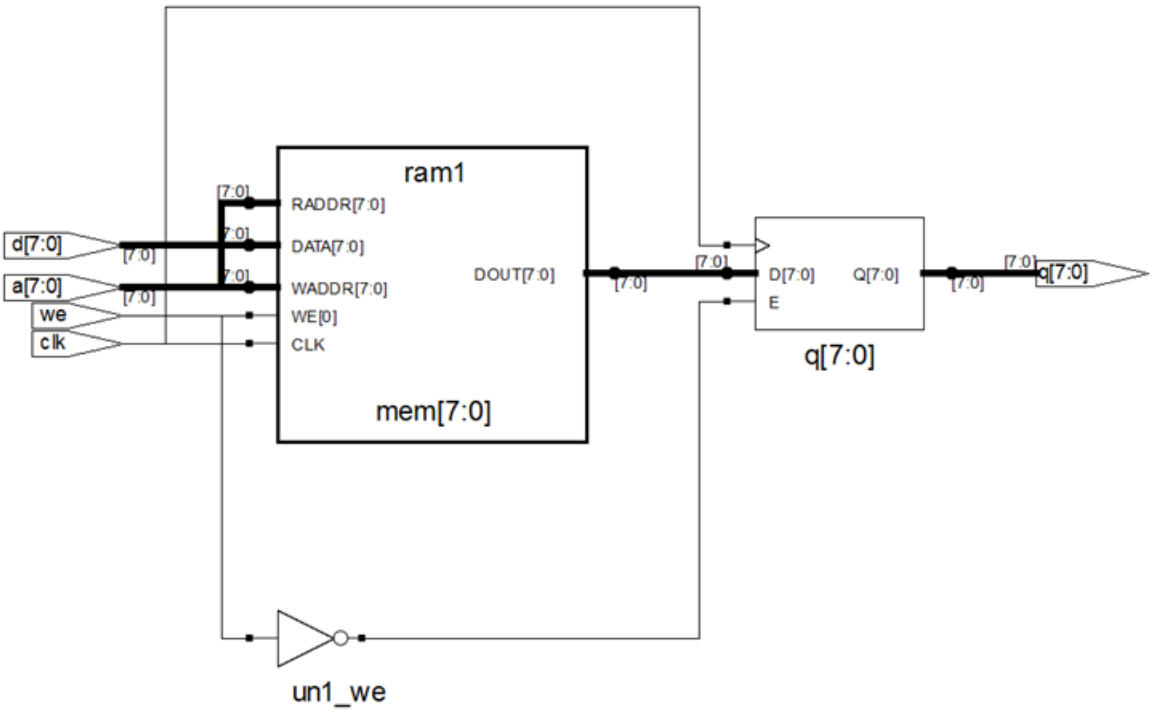
Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT1 use
RAM64x182 uses

Sequential Cells:
SLE0 uses

Example 8: Synchronous Read Without Pipelined Register, No Change Mode

The following design is a single-port RAM with synchronous read without pipelined register (sync-async) using No Change mode. The FPGA synthesis tool implements two SmartFusion2 RAM64X18 blocks, along with logic for No Change mode.

```
module ram_singleport_pipereg(clk,we,a,d,q);
input [7:0] d;
input [7:0] a;
input clk, we;
output [7:0] q;
reg [7:0] q;
reg [7:0] mem [255:0];

always @(posedge clk)
   begin
      if(we)
         mem[a] <= d;
      else
         q <= mem[a];
   end
endmodule
```



Resource Usage Summary for ram_singleport_pipereg

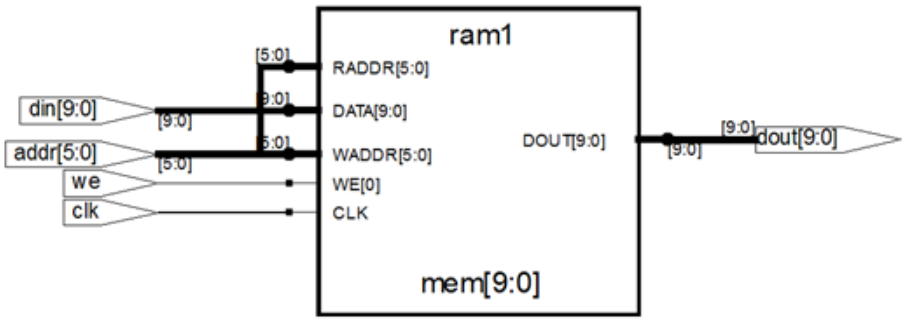
Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT1 use
RAM64x182 uses
CFG11 use
CFG38 uses

Sequential Cells:
SLE10 uses

Example 9: SP RAM with Asynchronous Read, RAM64X18 Read-First Mode

The following design is a single-port RAM with asynchronous read in Read-First mode. The FPGA synthesis tool infers SmartFusion2 RAM64X18.

```
module test_RTL (dout, addr, din, we, clk);

parameter data_width = 10;
parameter address_width = 6;
parameter ram_size = 64;

output [data_width-1:0] dout;
input [data_width-1:0] din;
input [address_width-1:0] addr;
input we, clk;

reg [data_width-1:0] mem [ram_size-1:0];

always @(posedge clk) begin
   //register the address for read operation
   if(we) addr] <= din;
   end

assign dout = mem[addr];

endmodule
```



Resource Usage Summary for test_RTL

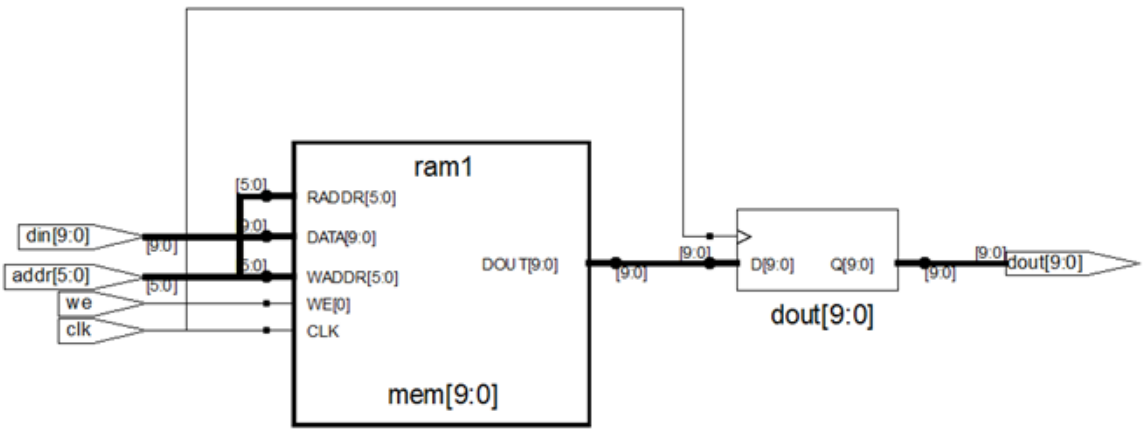
Mapping to part: m2s050tfbga896std
Cell usage:
RAM64x181 use

Sequential Cells:
SLE0 uses

Example 10: Asynchronous Read with Pipelined Register, RAM64X18 Read-First Mode

The following design is a single-port RAM with asynchronous read and pipelined register at its output in Read-First mode. The FPGA synthesis tool infers SmartFusion2 RAM64X18.

```
module test (dout, addr, din, we, clk);
parameter data_width = 10;
parameter address_width = 6;
parameter ram_size = 64;

output reg [data_width-1:0] dout;
input [data_width-1:0] din;
input [address_width-1:0] addr;
input clk;
input we;
wire[data_width-1:0] dout_net;
reg[data_width-1:0] mem [ram_size-1:0];
always @(posedge clk) begin
   if (we)
      mem[addr] <= din;
end
assign dout_net = mem[addr];

always @(posedge clk) begin
   dout <= dout_net;
end

endmodule
```



Resource Usage Summary for test

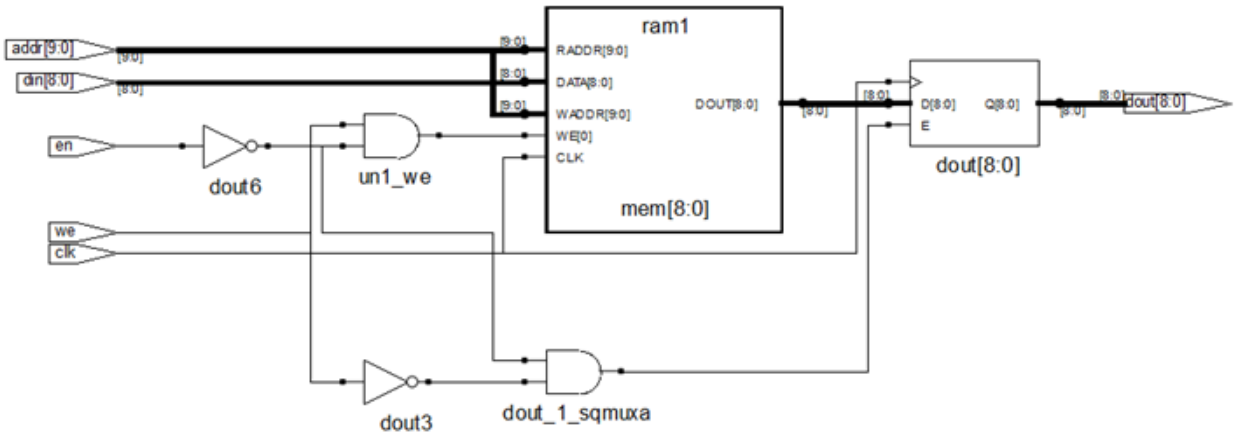
Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT1 use
RAM64x181 use

Sequential Cells:
SLE0 uses

Example 11: Single-Port RAM, RAM1K18 No Change Mode

The following design implements a single-port RAM with No Change mode. The FPGA synthesis tool infers SmartFusion2 RAM1K18 without glue logic for read/write collision check.

```
module test_RTL (dout, addr, din, we, clk, en);

parameter data_width = 9;
parameter address_width = 10;
parameter ram_size = 1024;

output reg [data_width-1:0] dout;
input [data_width-1:0] din;
input [address_width-1:0] addr;
input we, clk, en;

reg [data_width-1:0] mem [ram_size-1:0];

always @(posedge clk)
   begin
      if (!en)
      begin
         if(we)
         mem[addr] <= din;
         else
            dout <= mem[addr];
      end
   end
endmodule
```



Resource Usage Summary for test_RTL

Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT1 use
RAM1K181 use
CFG11 use
CFG21 use
CFG39 uses

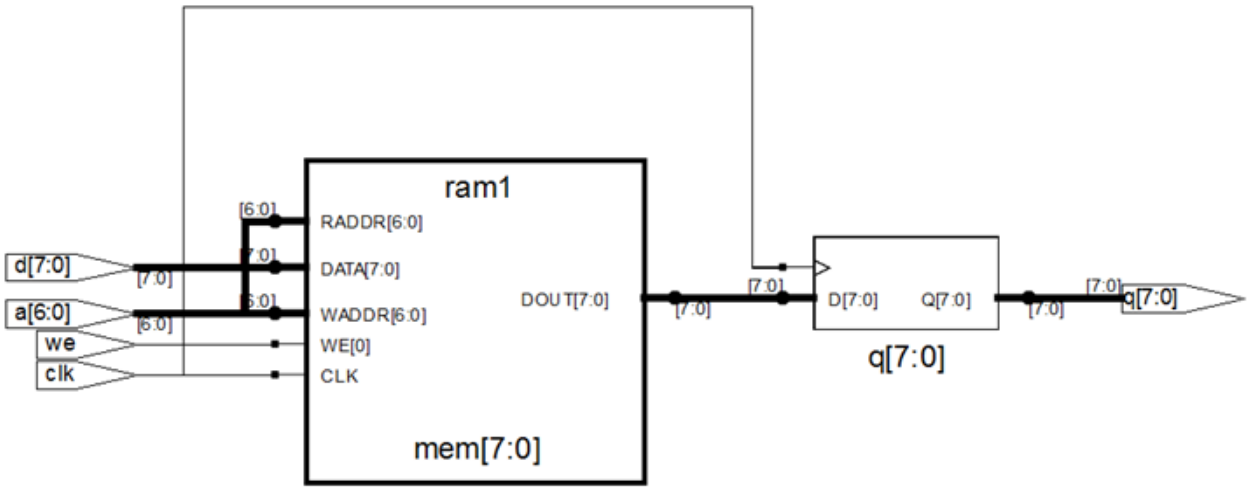
Sequential Cells:
SLE10 uses

Example 12: Single-Port RAM with Output Registers (VHDL)

The following design is a single-port RAM with output registers. The FPGA synthesis tool infers SmartFusion2 RAM64X18.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity ram_singleport_outreg is
port (d: in std_logic_vector(7 downto 0);
   a: in integer range 127 downto 0;
   we: in std_logic;
   clk: in std_logic;
   q: out std_logic_vector(7 downto 0) );
end ram_singleport_outreg;
```

```
architecture rtl of ram_singleport_outreg is
type mem_type is array (127 downto 0) of
std_logic_vector (7 downto 0);
signal mem: mem_type;
begin
   process(clk)
   begin
      if (clk'event and clk='1') then
            q <= mem(a);
            if (we='1') then
             mem(a) <= d;
            end if;
       end if;
   end process;
end rtl;
```



Resource Usage Summary for ram_singleport_outreg

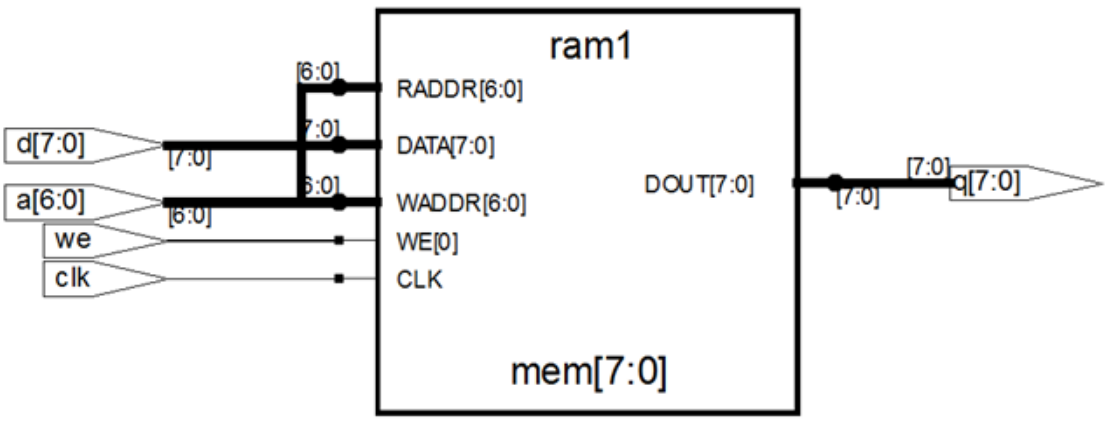
Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT1 use
RAM64x181 use

Sequential Cells:
SLE0 uses

Example 13: Single Port RAM with Asynchronous Read (VHDL)

The following design is a single-port RAM with asynchronous read. The FPGA synthesis tool infers SmartFusion2 RAM64X18.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity ram_singleport_noreg is
   port (d : in std_logic_vector(7 downto 0);
   a : in std_logic_vector(6 downto 0);
   we : in std_logic;
   clk : in std_logic;
   q : out std_logic_vector(7 downto 0) );
end ram_singleport_noreg;


architecture rtl of ram_singleport_noreg is
type mem_type is array (127 downto 0) of
   std_logic_vector (7 downto 0);
signal mem: mem_type;
begin
process (clk)
   begin
      if rising_edge(clk) then
         if (we = '1') then
             mem(conv_integer (a)) <= d;
         end if;
      end if;
end process;
q <= mem(conv_integer (a));
end rtl;
```



Resource Usage Summary for ram_singleport_outreg

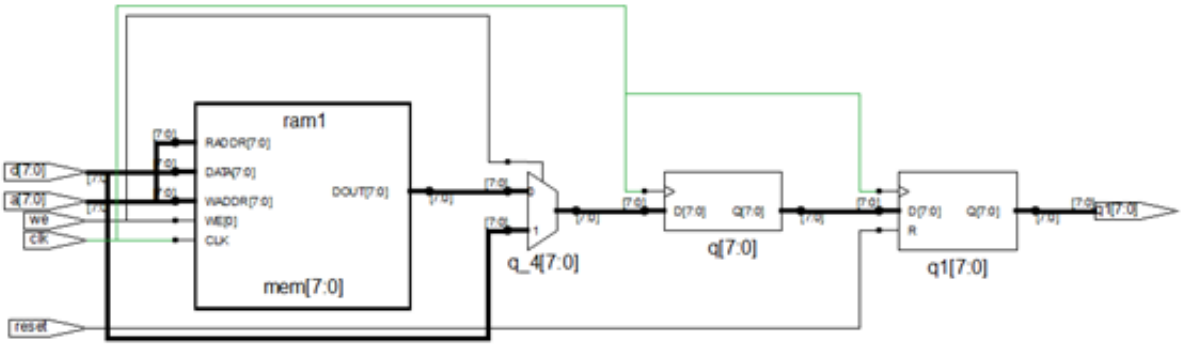
Mapping to part: m2s050tfbga896std
Cell usage:
RAM64x181 use

Sequential Cells:
SLE0 uses

Example 14: SP RAM with Synchronous Reset for Pipelined Register, RAM64x18

The following design is a single-port RAM with synchronous reset for pipelined register. The FPGA synthesis tool infers SmartFusion2 RAM64x18.

```
module ram_singleport_writefirst_pipe_areset(clk,we,a,d,q1,reset);
input [7:0] d;
input [7:0] a;
input clk, we,reset;
reg [7:0] q;
output [7:0] q1;
reg [7:0] q1;
reg [7:0] mem [255:0];

always @(posedge clk) begin
if(we)
mem[a] <= d;
end

always @ (posedge clk)
begin
if(we)
q <= d;
else
q <= mem[a];
end
always @ (posedge clk )
begin
if (reset)
  q1 <= 0;
else
   q1 <= q;
end
endmodule
```



Resource Usage Report for ram_singleport_writefirst_pipe_areset

Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT 1 use
RAM64x18 2 uses
CFG1 1 use

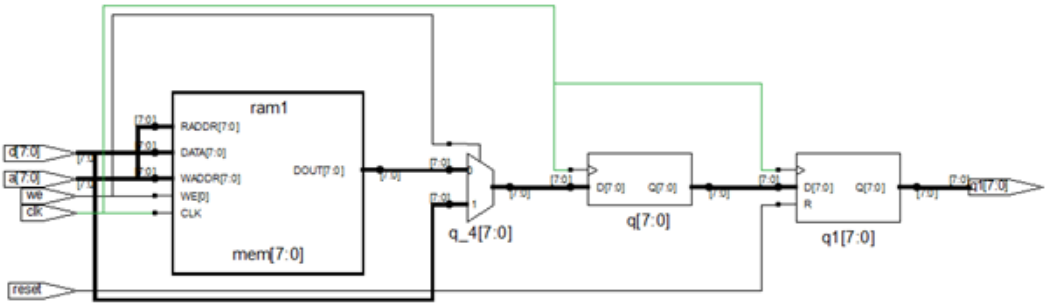
Sequential Cells:
SLE 0 uses

Example 15: Synchronous Reset for Pipelined Register and R/W Check, RAM64x18

The following design is a single-port RAM with synchronous reset for pipelined register. The syn_ramstyle attribute is set to rw_check. The FPGA synthesis tool infers SmartFusion2 RAM64X18 with glue logic.

```
module ram_singleport_writefirst_pipe_areset(clk,we,a,d,q1,reset);
input [7:0] d;
input [7:0] a;
input clk, we,reset;
reg [7:0] q;
output [7:0] q1;
reg [7:0] q1;
reg [7:0] mem [255:0] /* synthesis syn_ramstyle="rw_check" */;
always @(posedge clk) begin
if(we)
mem[a] <= d;
end
always @ (posedge clk)
begin
if(we)
q <= d;
else
q <= mem[a];
end
always @ (posedge clk )
```

```
begin
if (reset)
q1 <= 0;
else
q1 <= q;
end
endmodule
```



Resource Usage Report for ram_singleport_writefirst_pipe_areset

Mapping to part: m2s050tvf400std
Cell usage:
CLKINT 1 use
RAM64x18 2 uses
CFG1 1 use
CFG3 8 uses

Sequential Cells:
SLE 17 uses

Example 16: Single Port RAM1K18 (VHDL)

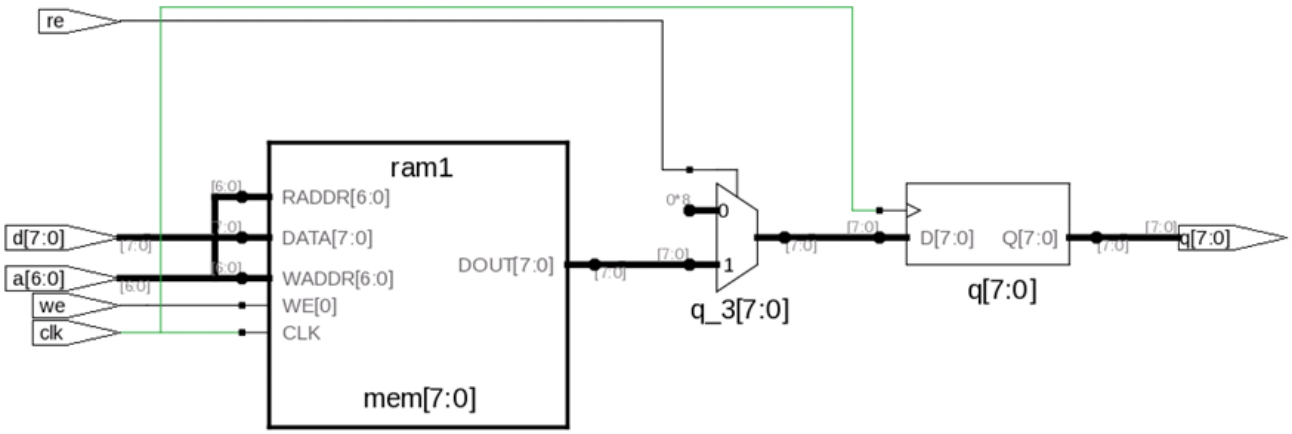
The following design is a VHDL example for RTG4 RAM, with Read-Enable set to read from RAM, and output of RAM set to 0 when Read-Enable is de-asserted.

RTL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity ram_test is
port (d: in std_logic_vector(7 downto 0);
   a: in integer range 127 downto 0;
   we: in std_logic;
   re: in std_logic;
   clk: in std_logic;
   q: out std_logic_vector(7 downto 0) );
end ram_test;
```

```
architecture rtl of ram_test is
type mem_type is array (127 downto 0) of std_logic_vector (7 downto 0);
signal mem: mem_type;

attribute syn_ramstyle : string;
attribute syn_ramstyle of mem : signal is "lsram";
begin
   process(clk)
   begin
   if (clk'event and clk='1') then
   --q <= mem(a);
   if (we='1') then
   mem(a) <= d;
   end if;

    if (re='1') then
      q <= mem(a);
   else
      q <= "00000000";
   end if;

end if;
end process;

end rtl;
```

SRS View (RTL View)



Resource Usage

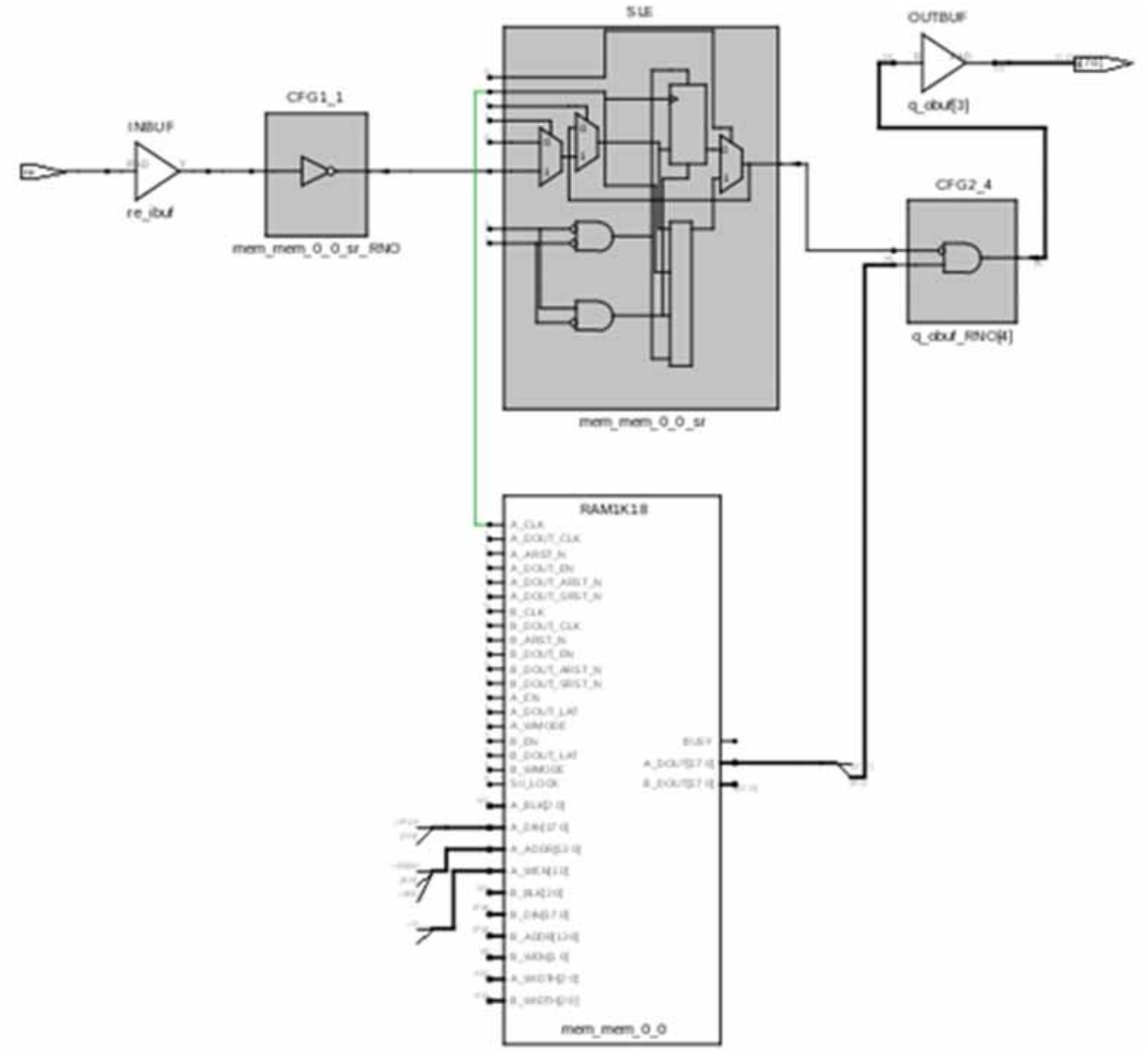
Cell usage:

CLKINT 1 use
CFG1 1 uses
CFG2 8 uses

SLE 1 uses

Block Rams (RAM1K18): 1

SRM (Technology) View

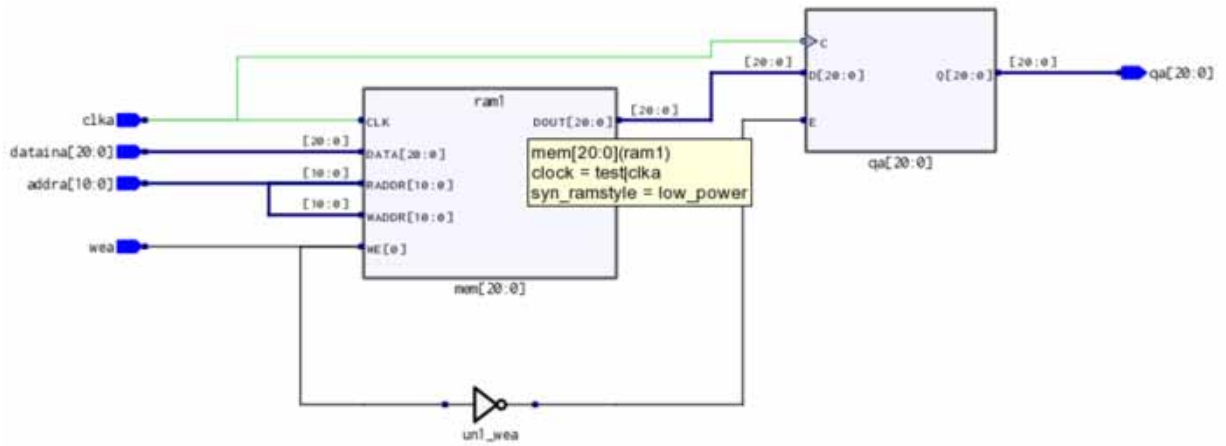


Example 17: Single Port RAM with syn_ramstyle = "low_power" and Global Power Option Off

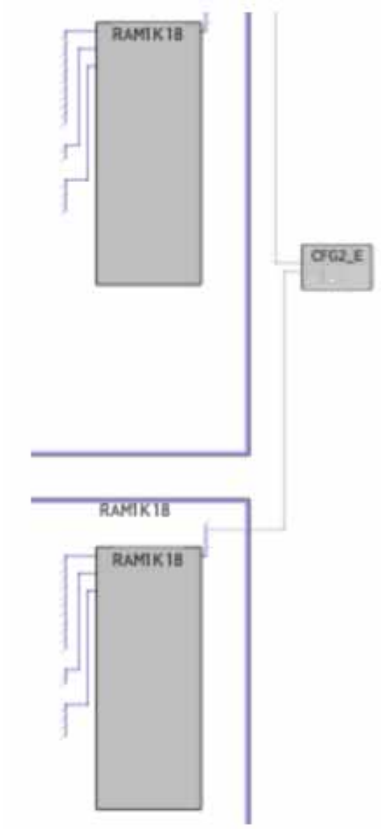
RTL

```
module test (clka,wea,addra,dataina,qa);
parameter addr_width = 11;
parameter data_width = 21;
input clka,wea;
input [data_width - 1 : 0] dataina;
input [addr_width - 1 : 0] addra;

output reg [data_width - 1 : 0] qa;

reg [data_width - 1 : 0] mem [(2**addr_width) - 1 : 0]/* synthesis syn_ramstyle =
"low_power" */;

always @ (posedge clka)
begin
      if(wea) mem[addra] <= dataina;
      else
            qa <= mem[addra];
end
endmodule
```

SRS (RTL) View



SRM (Technology) View



Resource Usage

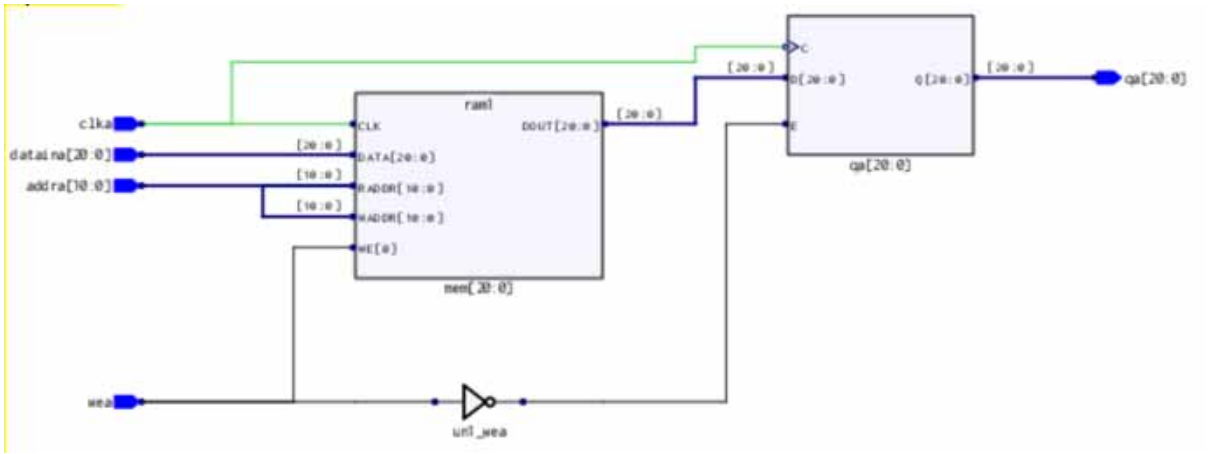
Cell usage:
CLKINT 1 use
CFG1 2 uses
CFG2 23 uses
CFG4 21 uses
SLE 23 uses
Block RAMs (RAM1K18) : 4 - RAMs inferred in low-power mode
Total LUTs: 46

Example 18: Single Port RAM with syn_ramstyle = "no_low_power" and Global Power Option On

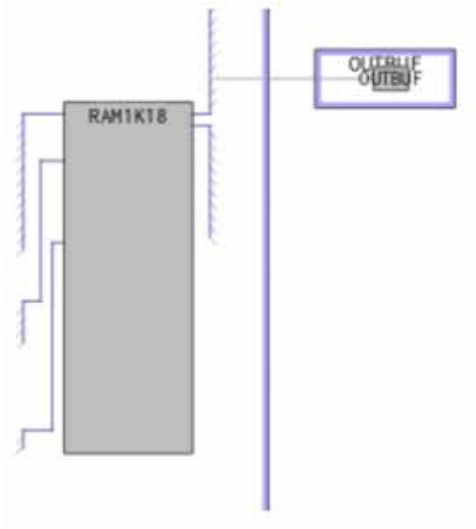
RTL

```
module test (clka,wea,addra,dataina,qa);
parameter addr_width = 11;
parameter data_width = 21;
input clka,wea;
input [data_width - 1 : 0] dataina;
input [addr_width - 1 : 0] addra;
output reg [data_width - 1 : 0] qa;

reg [data_width - 1 : 0] mem [(2**addr_width) - 1 : 0]/* synthesis syn_ramstyle =
"no_low_power" */;

always @ (posedge clka)
begin
        if(wea) mem[addra] <= dataina;
        else
              qa <= mem[addra];
end
endmodule
```

SRS (RTL) View



SRM (Technology) View



Resource Usage

Cell usage:
CLKINT 1 use
SLE 0 uses
RAM/ROM usage summary
Total Block RAMs (RAM1K18) : 3 of 952 (0%)
Total LUTs: 0

Dual-Port RAM Coding Style Examples

Here are examples of inferring RAM1K18 and RAM64X18 RAM blocks for dual-port (DP) RAM:

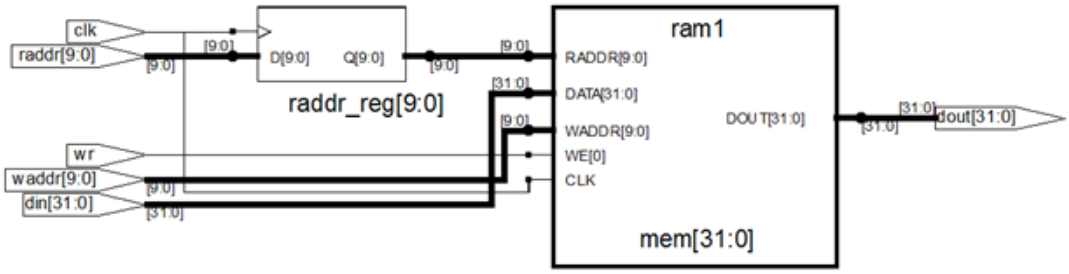
- Example 19: Simple Dual-Port RAM, RAM1K18 Write-First Mode, on page 33
- Example 20: Simple Dual-Port RAM, RAM64X18 Write-First Mode, on page 34
- Example 21: Simple Dual-Port RAM1K18 RAM with Pipelined Register, Write-First Mode, on page 35
- Example 22: Simple Dual-Port RAM64X18 with Pipelined Register, Write-First Mode, on page 37
- Example 23: Simple Dual-Port RAM Asynchronous Read, RAM64X18 Read-First Mode, on page 38
- Example 24: Asynchronous Read and Pipelined Register with Clock Enable RAM64X18, on page 39
- Example 25: Simple Dual-Port RAM with Output Register, on page 40
- Example 26: DP RAM with Output Register and Read Address Register (VHDL), on page 41
- Example 27: DP RAM with Read Address Register (512 x 36 Configurations), on page 43
- Example 28: DP RAM with Asynchronous Reset for Pipelined Register, on page 44
- Example 29: Simple Dual-Port RAM with Output Register Using R/W Check, on page 45
- Example 30: Simple Dual-Port RAM with Enable on Output Register, on page 46
- Example 31: Simple Dual-Port RAM1K18 in Low Power Mode, on page 47
- Example 32: Simple Dual-Port RAM64x18 in Low Power Mode, on page 50

Example 19: Simple Dual-Port RAM, RAM1K18 Write-First Mode

The following design is a simple dual-port (two-port) RAM with synchronous read/write operations. Different read and write address are used in Write-First mode. The FPGA synthesis tool infers SmartFusion2 RAM1K18.

```
module ram_2port_raddreg(clk, wr, raddr, din, waddr, dout);

input clk;
input [31:0] din;
input wr;
input [9:0] waddr, raddr;

output [31:0] dout;
```

```
reg [9:0] raddr_reg;
reg [31:0] mem [0:1023];

assign dout = mem[raddr_reg]  ;

always@ (posedge clk)
begin
   raddr_reg <= raddr;
      if (wr)
         mem[waddr]  <= din;

end
endmodule
```



Resource Usage Report for ram_2port_raddreg

Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT1 use
RAM1K182 uses

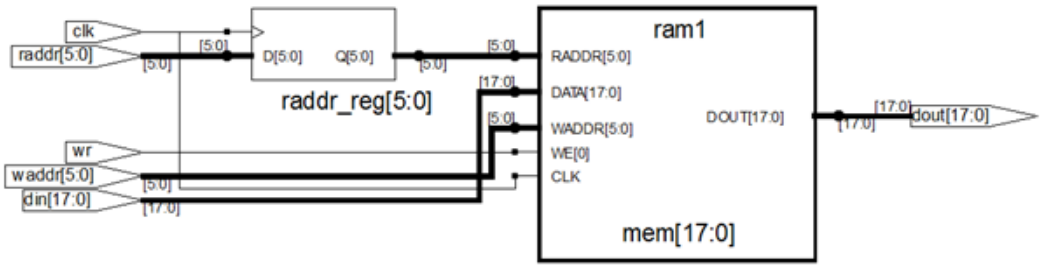
Sequential Cells:
SLE0 uses

Example 20: Simple Dual-Port RAM, RAM64X18 Write-First Mode

The following design is a simple dual-port RAM with synchronous read/write operations. Different read and write addresses are used in Write-First mode. The FPGA synthesis tool infers SmartFusion2 RAM64X18.

```
module ram_2port_raddreg(clk, wr, raddr, din, waddr, dout);

input clk;
input [17:0] din;
input wr;
input [5:0] waddr, raddr;

output [17:0] dout;
```

```
reg [5:0] raddr_reg;
reg [17:0] mem [0:63];

assign dout = mem[raddr_reg]  ;

always@(posedge clk)
begin
   raddr_reg <= raddr;
      if(wr)
         mem[waddr]  <= din;

end
endmodule
```



Resource Usage Summary for ram_2port_raddreg

Mapping to part: m2s050tfbga896std

Cell usage:
CLKINT 1 use
RAM64x18 1 use

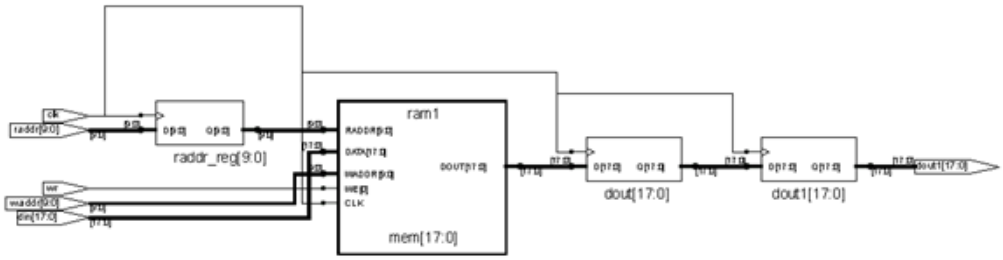
Sequential Cells:
SLE 0 uses

Example 21: Simple Dual-Port RAM1K18 RAM with Pipelined Register, Write-First Mode

The following design is a simple dual-port (two-port) RAM with synchronous read/write operations and pipelined register, in Write-First mode. The FPGA synthesis tool infers SmartFusion2 RAM1K18. The output pipelined register dout1 is not packed into the RAM; only register dout is packed in the RAM.

```
module ram_2port_pipe(clk,wr,raddr,din,waddr,dout1);


input clk;
input [17:0] din;
input wr;
input [9:0] waddr,raddr;
```

```
output [17:0] dout1 ;
reg [9:0] raddr_reg;
reg [17:0] mem [0:1023];
reg [17:0] dout, dout1;


always@(posedge clk)
begin
   raddr_reg <= raddr;
   dout <= mem[raddr_reg];
      if(wr)
         mem[waddr]  <= din;

end

always @(posedge clk)
begin

   dout1 <= dout;
end

endmodule
```



Resource Usage Report for ram_2port_pipe

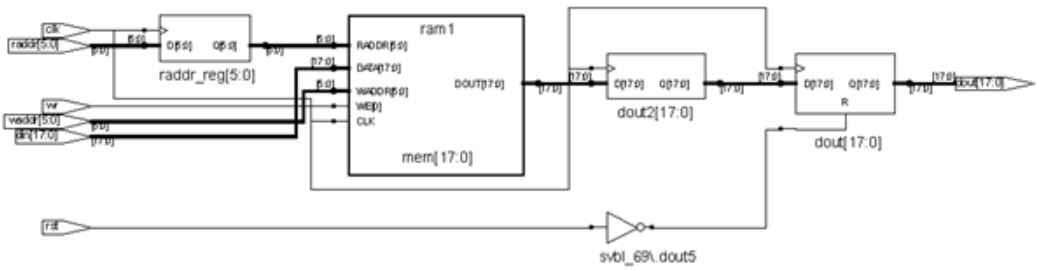
Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT1 use
RAM1K181 use

Sequential Cells:
SLE18 uses

Example 22: Simple Dual-Port RAM64X18 with Pipelined Register, Write-First Mode

The following design is a simple dual-port (two port) RAM with synchronous read/write operation with pipelined register in Write-First mode. The FPGA synthesis tool infers SmartFusion2 RAM64X18. The output pipelined register dout is not packed in the RAM.

```
module ram_2port_pipe(clk,wr,raddr,din,waddr,dout, rst);

input clk;
input [17:0] din;
input wr, rst;
input [5:0] waddr,raddr;

output [17:0] dout;

reg [5:0] raddr_reg;
reg [17:0] mem [0:63];

reg [17:0] dout2, dout;
wire [17:0] dout1;

assign dout1 = mem[raddr_reg]  ;

always@(posedge clk)
begin
   raddr_reg <= raddr;
      dout2 <= dout1;
      if(wr)
         mem[waddr]  <= din;

end

always @(posedge clk or negedge rst)
begin
   if (~ rst )
      dout <= 8'b0;
   else
      dout <= dout2;
end
endmodule
```



Resource Usage Summary for ram_2port_pipe

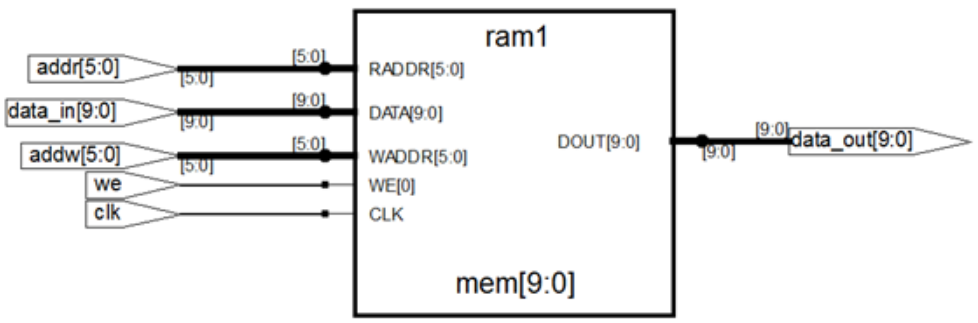
Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT 2 uses
RAM64x18 1 use

Sequential Cells:
SLE 18 uses

Example 23: Simple Dual-Port RAM Asynchronous Read, RAM64X18 Read-First Mode

The following design is a simple dual-port RAM with asynchronous read in Read-First mode. The FPGA synthesis tool infers SmartFusion2 RAM64X18.

```
module test_RTL (addr,addw, we, clk , data_out, data_in);

parameter ADDR_WIDTH = 6;
parameter ADDW_WIDTH = 6;
parameter DATA_WIDTH = 10;
parameter MEM_DEPTH = 64;

input [ADDR_WIDTH-1:0]addr;
input [ADDW_WIDTH-1:0]addw;
input clk, we;
input [DATA_WIDTH-1 : 0]data_in;
output [DATA_WIDTH-1 : 0]data_out;

reg [DATA_WIDTH-1 : 0]mem[MEM_DEPTH-1 :0] ;

assign data_out = mem[addr];
always @(posedge clk)
begin
   if (we)
   mem[addw] <= data_in;
end

endmodule
```



Resource Usage Summary for test_RTL

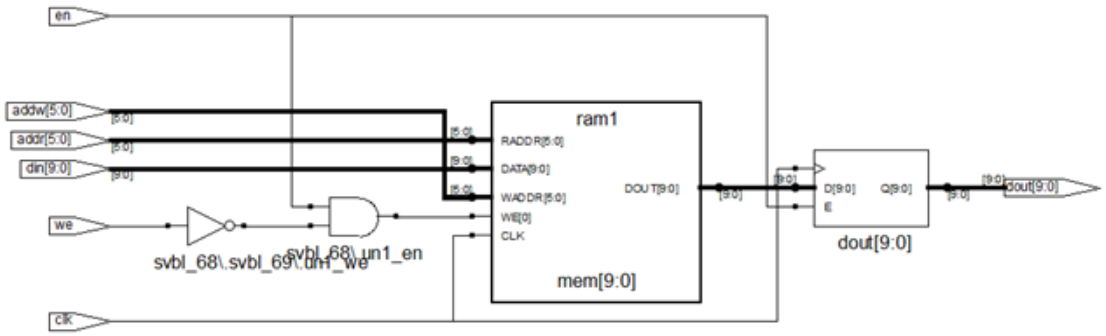
Mapping to part: m2s050tfbga896std
Cell usage:
RAM64x181 use

Sequential Cells:
SLE0 uses

Example 24: Asynchronous Read and Pipelined Register with Clock Enable RAM64X18

The following design is a simple dual-port RAM with asynchronous read and output pipelined register with clock enable. The RAM is in No Change mode. The FPGA synthesis tool infers SmartFusion2 RAM64X18. Logic for the enable is inferred outside the RAM. To pack the enable, use the syn_ramstyle="rw_check" attribute.

```
module test_RTL (dout, addr, din, we, clk, en, addw);

parameter data_width = 10;
parameter address_width = 6;
parameter ram_size = 64;

output reg [data_width-1:0] dout;
input [data_width-1:0] din;
input [address_width-1:0] addr, addw;
input we, clk, en;
reg [data_width-1:0] mem [ram_size-1:0];
wire [data_width-1:0] dout_reg ;

assign dout_reg = mem[addr];
always @(posedge clk) begin
   if (en) begin
      if(!we)
         mem[addw] <= din;
      end
   end

always @(posedge clk) begin

   if (en) begin
      dout <= dout_reg;
   end
end
endmodule
```



Resource Usage Summary for test_RTL

Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT1 use
RAM64x181 use
CFG11 use
CFG3 10 uses

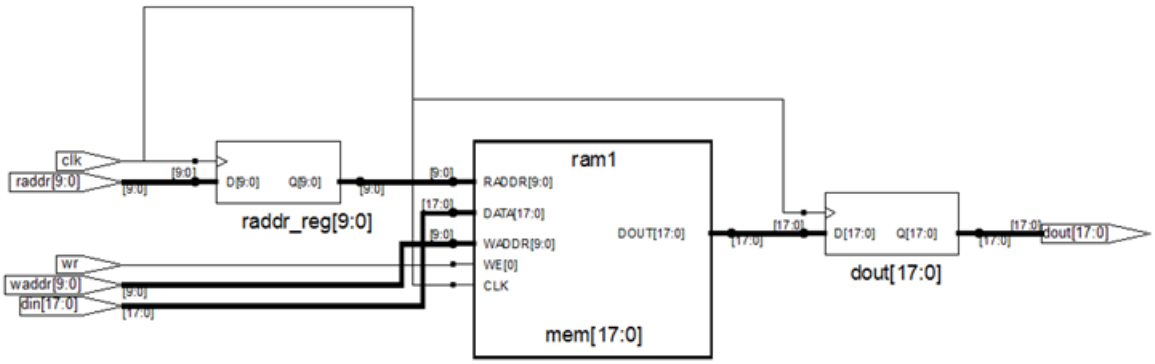
Sequential Cells:
SLE11 uses

Example 25: Simple Dual-Port RAM with Output Register

The following design is a simple dual-port RAM with output register. The FPGA synthesis tool infers SmartFusion2 RAM1K18.

```
module ram_2port_pipe(clk,wr,raddr,din,waddr,dout);

input clk;
input [17:0] din;
input wr;
input [9:0] waddr,raddr;

output [17:0] dout;

reg [9:0] raddr_reg;
reg [17:0] mem [0:1023]
reg [17:0] dout;


always@(posedge clk)
   begin
      raddr_reg <= raddr;
      dout <= mem[raddr_reg];
```

```
      if(wr)
         mem[waddr]  <= din;

   end
endmodule
```



Resource Usage Summary for ram_2port_pipe

Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT1 use
RAM1K181 use

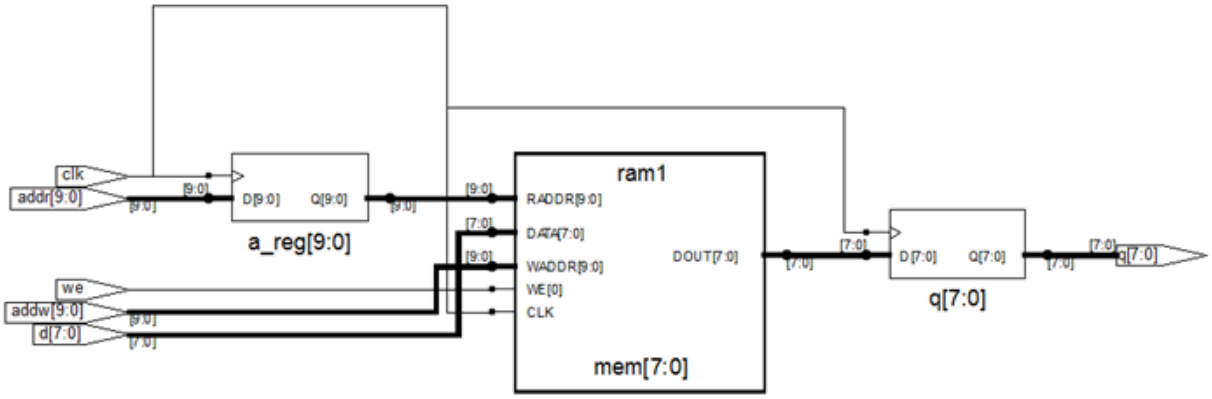
Sequential Cells:
SLE0 uses

Example 26: DP RAM with Output Register and Read Address Register (VHDL)

The following design is a simple dual-port RAM with output and read address registers. The FPGA synthesis tool infers SmartFusion2 RAM1K18.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ram_simpledualport_outreg is
   port (d: in std_logic_vector(7 downto 0);
   addr: in integer range 1023 downto 0;
   addw: in integer range 1023 downto 0;
   we: in std_logic;
   clk: in std_logic;
   q: out std_logic_vector(7 downto 0) );
end ram_simpledualport_outreg;
```

```
architecture rtl of ram_simpledualport_outreg is
type mem_type is array (1023 downto 0) of std_logic_vector (7 downto 0);
signal mem: mem_type;
signal a_reg : integer range 1023 downto 0;
begin
   process (clk)
      begin
      if (clk'event and clk='1' ) then
         a_reg <= addr;
      end if;
   end process;

   process(clk)
      begin
         if (clk'event and clk='1') then
            q <= mem(a_reg);
               if (we='1') then
                mem(addw) <= d;
               end if;
          end if;
   end process;
end rtl;
```



Resource Usage Summary for ram_simpledualport_outreg

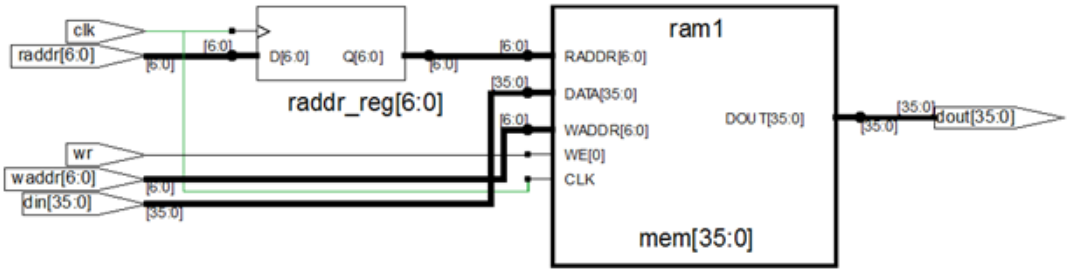
Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT1 use
RAM1K181 use

Sequential Cells:
SLE0 uses

Example 27: DP RAM with Read Address Register (512 x 36 Configurations)

The following design is a simple dual-port RAM with the read address registered. The FPGA synthesis tool infers SmartFusion2 RAM1K18.

```
module ram_2port_addreg_512x36(clk,wr,raddr,din,waddr,dout);

input clk;
input [35:0] din;
input wr;
input [6:0] waddr,raddr;

output [35:0] dout;

reg [6:0] raddr_reg;
reg [35:0] mem [0:511];
wire [35:0] dout;

assign dout = mem[raddr_reg]  ;

always@(posedge clk)
begin
     raddr_reg <= raddr;
     if(wr)
      mem[waddr]  <= din;

end
endmodule
```



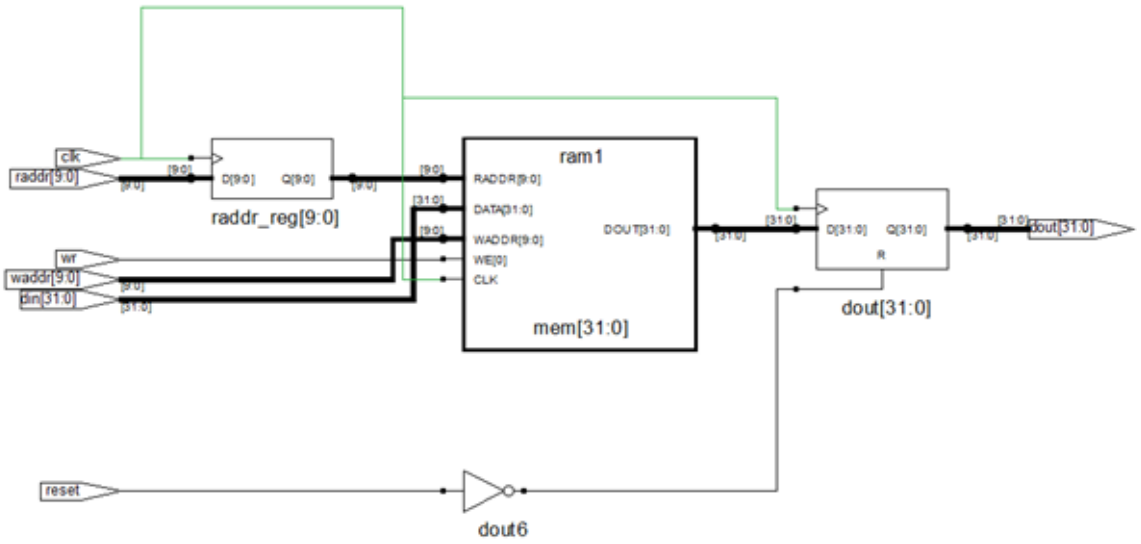
Resource Usage Report for ram_2port_addreg_512x36

Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT 1 use
RAM1K18 1 use
Sequential Cells:
SLE 0 uses

Example 28: DP RAM with Asynchronous Reset for Pipelined Register

The following design is a simple dual-port RAM with asynchronous reset for pipelined register. The FPGA synthesis tool infers SmartFusion2 RAM1K18.

```
module ram_2port_addreg_areset(clk,wr,raddr,din,waddr,dout,reset);
input clk,reset;
input [31:0] din;
input wr;
input [9:0] waddr,raddr;
output [31:0] dout;
reg [31:0] dout;
reg [31:0] mem [0:1023];
reg [9:0] raddr_reg;

always@(posedge clk or negedge reset)
begin
  if (!reset)
    dout <= 0;
  else
     dout <= mem[raddr_reg]  ;
 end

always@(posedge clk )
begin
  if(wr)
    mem[waddr]  <= din;
   raddr_reg <= raddr;
end
endmodule
```



Resource Usage Report for ram_2port_addreg_areset

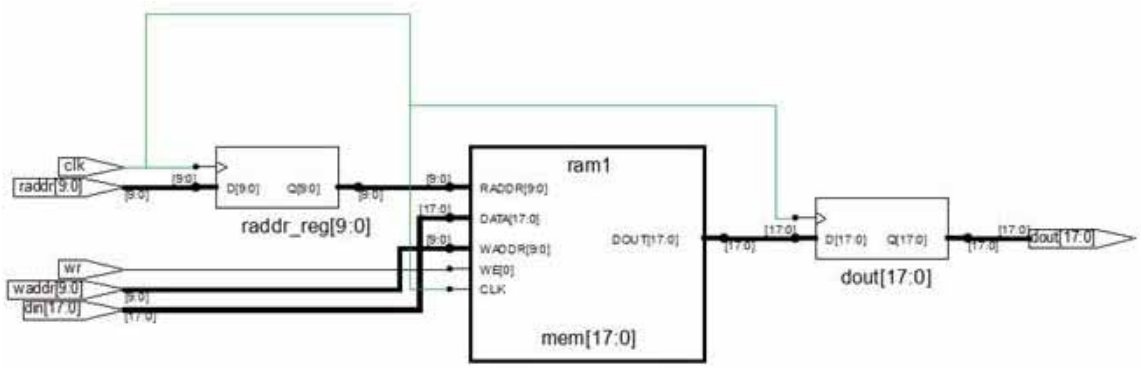
Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT 1 use
RAM1K18 2 uses
CFG2 0 uses

Sequential Cells:
SLE 0 uses

Example 29: Simple Dual-Port RAM with Output Register Using R/W Check

The following design is a single-port RAM with output register and the syn_ramstyle="rw_check" attribute. The FPGA synthesis tool infers SMartFusion2 RAM1K18 along with glue logic for the read/write address check.

```
module ram_2port_pipe(clk,wr,raddr,din,waddr,dout);
input clk;
input [17:0] din;
input wr;
input [9:0] waddr,raddr;
output [17:0] dout;
reg [9:0] raddr_reg;
reg [17:0] mem [0:1023] /* synthesis syn_ramstyle= "rw_check" */;
reg [17:0] dout;
always@(posedge clk)
begin
raddr_reg <= raddr;
dout <= mem[raddr_reg];
if(wr)
mem[waddr] <= din;
end
endmodule
```



Resource Usage Report for ram_2port_pipe

Mapping to part: m2s050tvf400std
Cell usage:
CLKINT 1 use
RAM1K18 1 use
CFG3 2 uses
CFG4 23 uses

Sequential Cells:

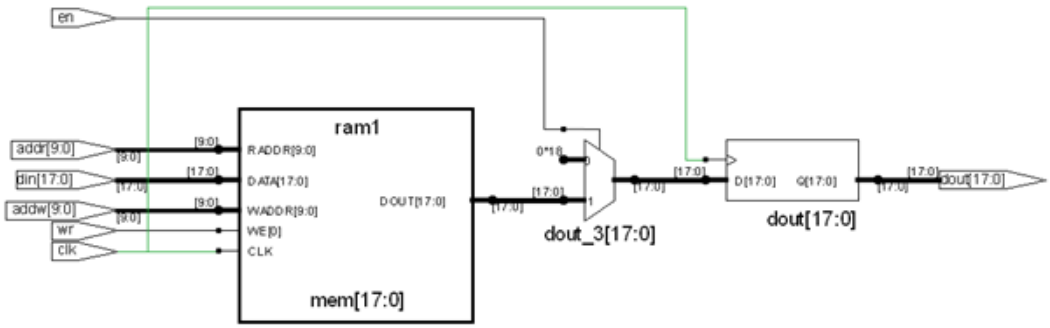
SLE 57 uses

Example 30: Simple Dual-Port RAM with Enable on Output Register

The following design is an example for simple dual-port RAM with enable on output register. When enable is de-asserted, the RAM output is 0.

The tool infers one RAM1K18 using the A_BLK pin for enable en. The en pin is mapped using the RAM1K18 A_BLK or B_BLK pins only when one port of RAM1K18 is used for reading and the other port is used for writing.

```
module ram_singleport_outreg_areset_en_rtl(clk,wr,addr,addw, din,dout,en);

output [17:0] dout;
input [17:0] din;
input [9:0] addr, addw;
input clk, wr, en;
reg [17:0] dout;
reg [17:0]mem[1023:0] ;

always@(posedge clk)
begin
if(wr)
   mem[addw] <= din;
end

always@(posedge clk)
begin
if(en)
      dout <= mem[addr];
else
      dout <= 0;
end

endmodule
```



Resource Usage Report for ram_singleport_outreg_areset_en

Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT 1 use
RAM1K18 1 use

Sequential Cells:
SLE 0 uses

Example 31: Simple Dual-Port RAM1K18 in Low Power Mode

For 2Kx18 RAM configuration, the tool fractures the data width and infers two RAM1K18 RAM blocks in 2Kx9 mode.

If you set global option low_power_ram_decomp 1 in the project file (*.prj), the tool fractures the address width to infer two RAM1K18 blocks in 1Kx18 mode. The tool connects the MSB bit of address to BLK pin and OR gates at the output, to select the output from two RAM blocks.

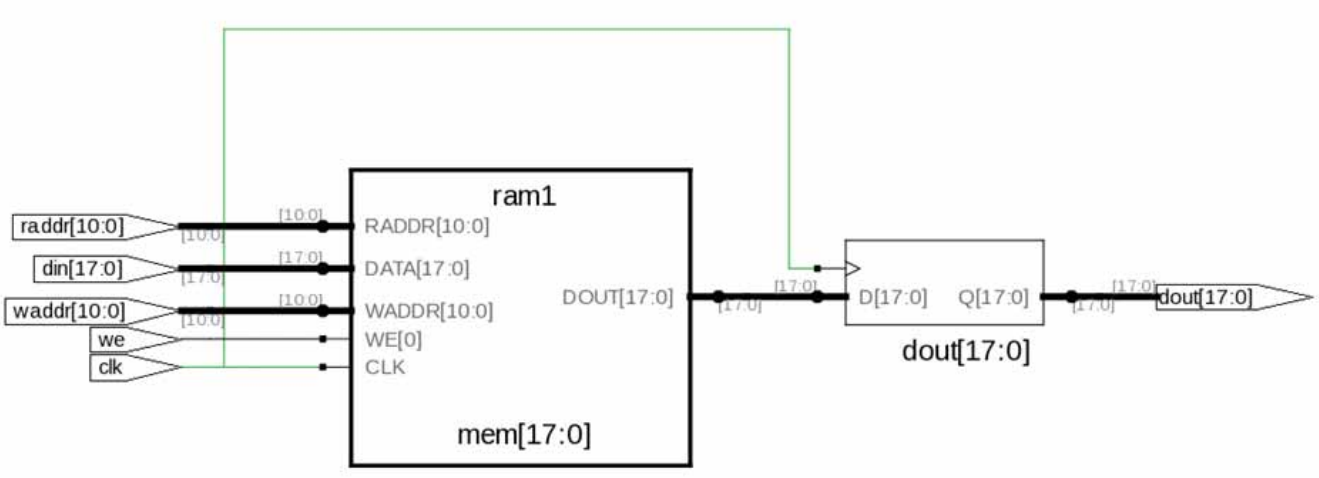
RTL

```
module test (raddr, waddr, clk, we, din, dout);
parameter ADDR_WIDTH = 11;
parameter DATA_WIDTH = 18;
parameter MEM_DEPTH = 2048;
input [ADDR_WIDTH-1:0] raddr;
input [ADDR_WIDTH-1:0] waddr;
input clk, we;
output[DATA_WIDTH-1 : 0]dout;
input [DATA_WIDTH-1 : 0]din;
reg [DATA_WIDTH-1 : 0]dout;
reg [DATA_WIDTH-1 : 0]mem[MEM_DEPTH-1 :0] ;
always@(posedge clk) begin
dout <= mem[raddr];
end
```

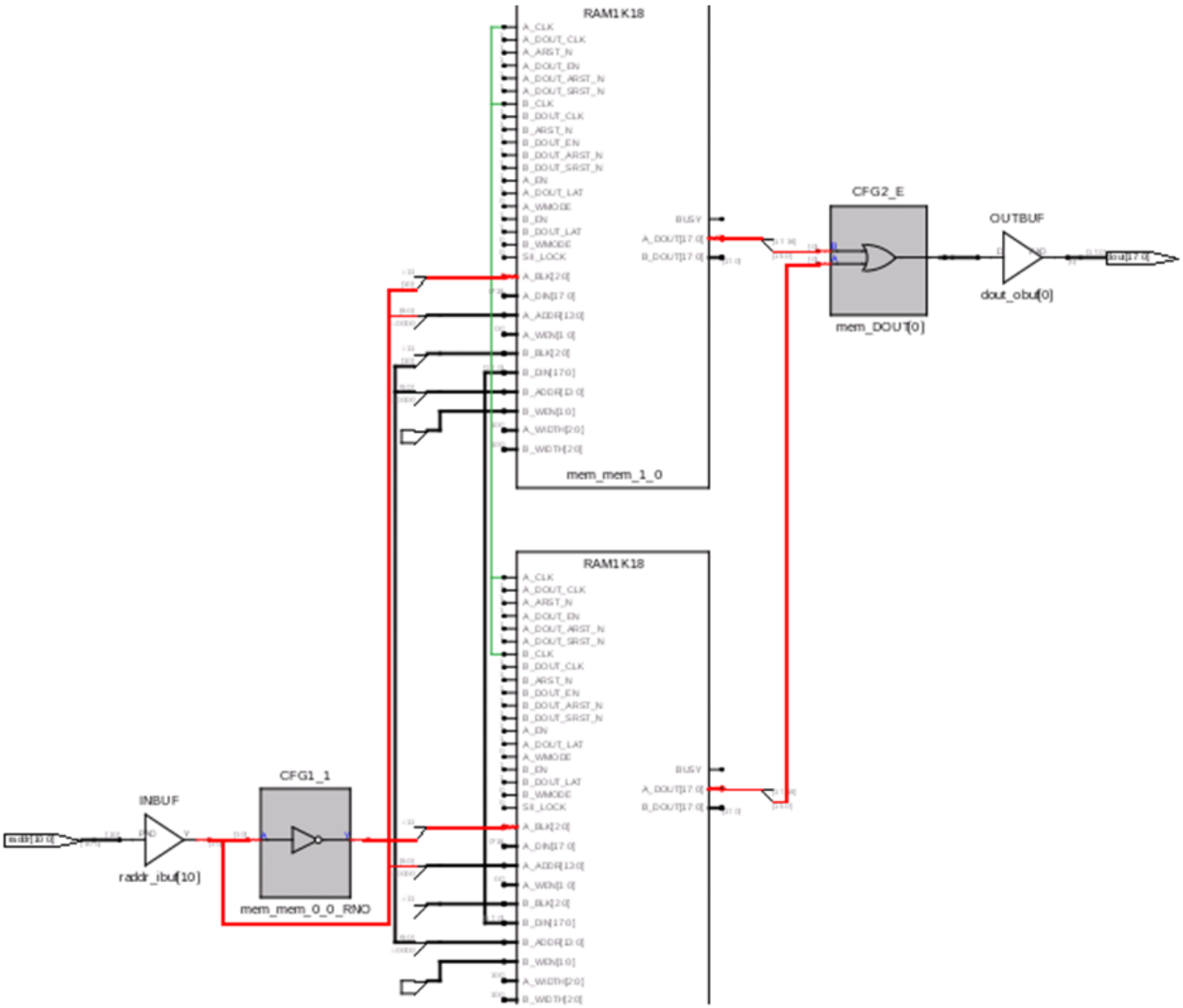
```
always@(posedge clk) begin
   if(we)
     mem[waddr] <= din;
end
endmodule
```

Project File option—set_option -low_power_ram_decomp 1

SRS View (RTL View)



SRM View (Technology View)



Resource Usage

Cell usage:

CLKINT	1 use
CFG1	2 uses
CFG2	20 uses

SLE 0 uses

Block Rams (RAM1K18): 2

Example 32: Simple Dual-Port RAM64x18 in Low Power Mode

For 64Kx32 RAM configuration, the tool fractures the data width and infers two RAM64x32 RAM blocks in 64x18 mode.

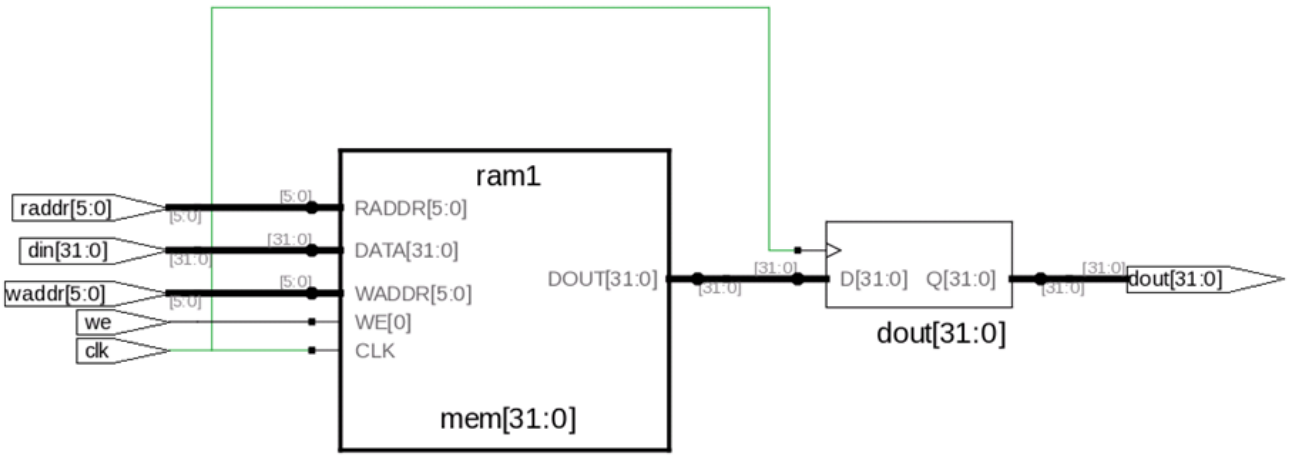
If you set global option low_power_ram_decomp 1 in the project file (*.prj), the tool fractures the address width to infer two RAM64x18 blocks in 512x2 mode. The tool connects the MSB bit of address to BLK pin and OR gates at the output, to select the output from two RAM blocks.

RTL

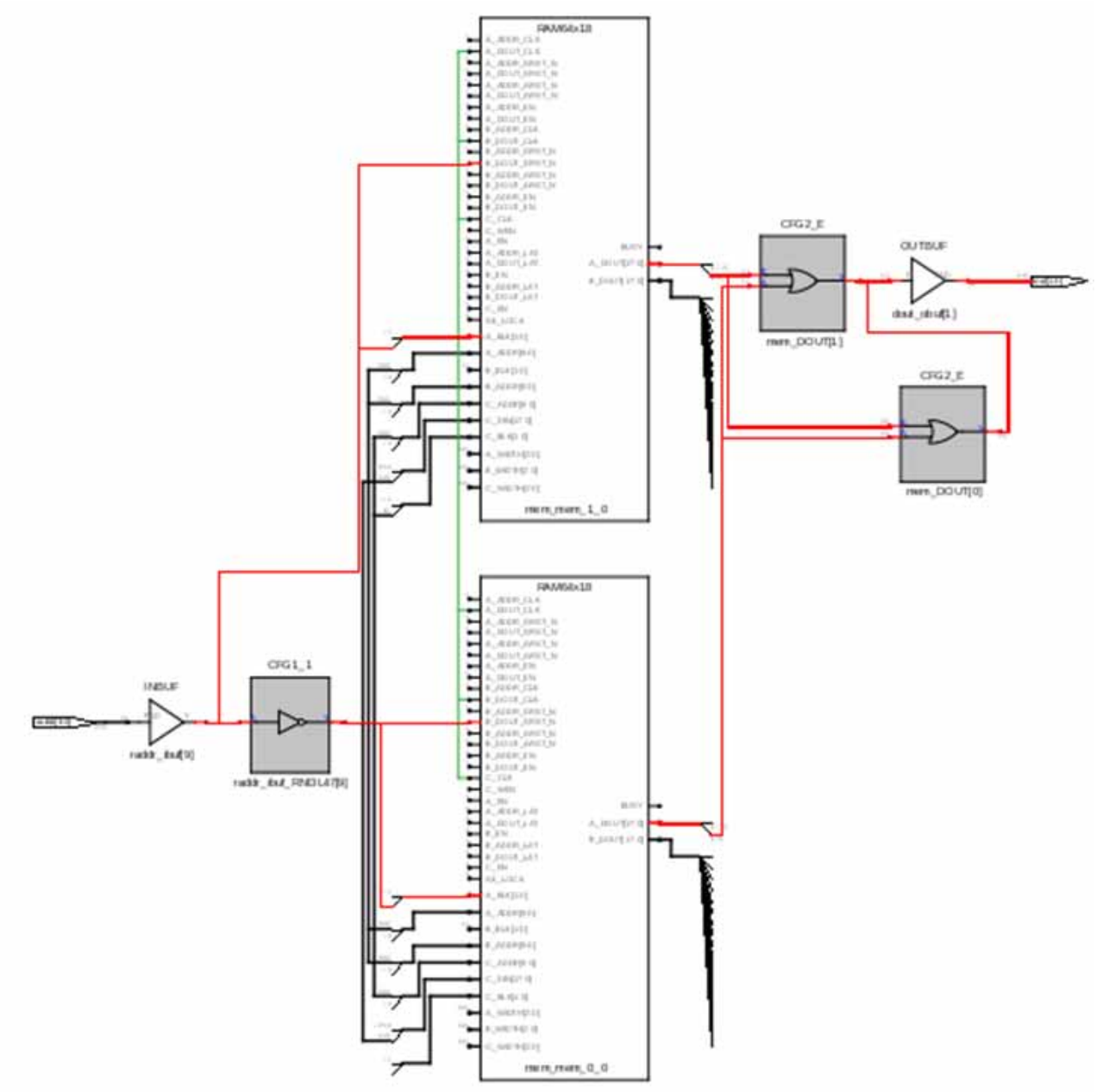
```
module test (raddr, waddr, clk, we, din, dout);
parameter ADDR_WIDTH = 6;
parameter DATA_WIDTH = 32;
parameter MEM_DEPTH = 64;
input [ADDR_WIDTH-1:0] raddr;
input [ADDR_WIDTH-1:0] waddr;
input clk, we;
output[DATA_WIDTH-1 : 0] dout;
input [DATA_WIDTH-1 : 0] din;
reg [DATA_WIDTH-1 : 0] dout;
reg [DATA_WIDTH-1 : 0] mem [MEM_DEPTH-1 :0] ;
always@(posedge clk) begin
dout <= mem[raddr];
end
always@(posedge clk) begin
   if(we)
     mem[waddr] <= din;
end
endmodule
```

Project File option—set_option -low_power_ram_decomp 1

SRS View (RTL View)



SRM View (Technology View)



Resource Usage

Cell usage:
CLKINT 1 use
SLE 0 uses

Block Rams (RAM64x18): 2

True Dual-Port RAM Coding Style Examples

Here are examples where the tool infers RAM1K18 and RAM64X18 RAM blocks for true dual-port (TDP) RAM.

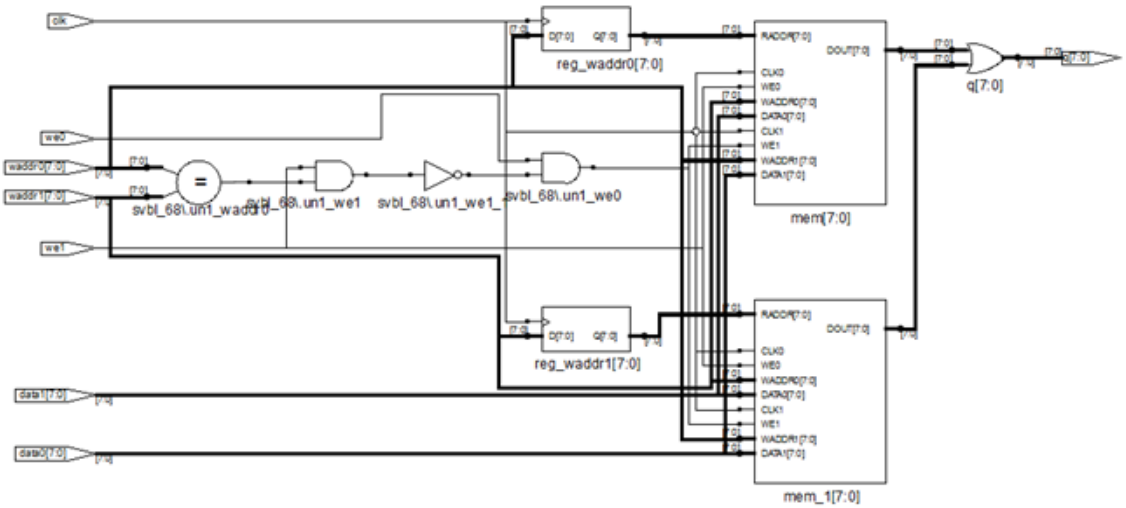
- Example 33: True Dual-Port RAM, Single Clock, on page 52
- Example 34: True Dual-Port RAM, Multiple Clocks, on page 53
- Example 35: True Dual-Port RAM with Pipelined Register, on page 55
- Example 36: True Dual-Port RAM with Read Address Register (VHDL), on page 57
- Example 37: TDP RAM with Output Registered, Pipelined and Non-Pipelined (VHDL), on page 59
- Example 38: TDP RAM with Asynchronous Reset for Pipelined Register, RAM1K18, on page 63

Example 33: True Dual-Port RAM, Single Clock

The following design is a true dual-port RAM with two read and write ports and one clock. The FPGA synthesis tool infers SmartFusion2 RAM1K18.

```
module ram_dport_reg(data0,data1,waddr0, waddr1,we0,we1,clk,q);

parameter d_width = 8;
parameter addr_width = 8;
parameter mem_depth = 256;

input [d_width-1:0] data0, data1;
input [addr_width-1:0] waddr0, waddr1;
input we0, we1, clk;

output [d_width-1:0] q;

reg [d_width-1:0] mem [mem_depth-1:0];
reg [addr_width-1:0] reg_waddr0, reg_waddr1;

wire [d_width-1:0] q0, q1;

assign q = q0 | q1;

assign q0 = mem[reg_waddr0];
assign q1 = mem[reg_waddr1];

always @(posedge clk)
begin
if (we0)
mem[waddr0] <= data0;
if (we1)
mem[waddr1] <= data1;
```

```
reg_waddr0 <= waddr0;
reg_waddr1 <= waddr1;

end
endmodule
```



Resource Usage Summary for ram_dport_reg

Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT1 use
RAM1K181 use
CFG28 uses
CFG31 use
CFG45 uses

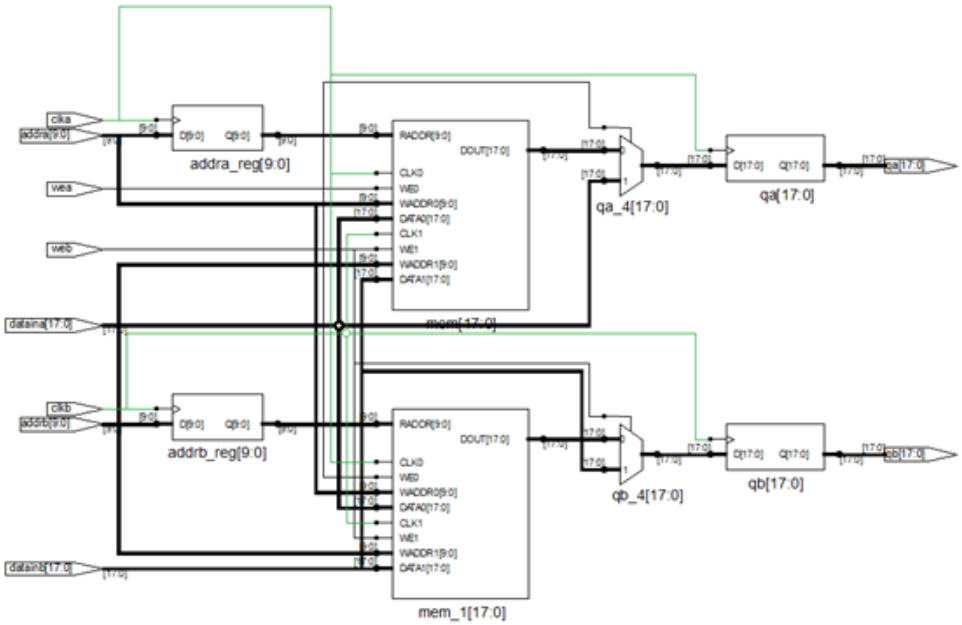
Sequential Cells:
SLE0 uses

Example 34: True Dual-Port RAM, Multiple Clocks

The following design is a true dual-port RAM with two read and write ports and two clocks. The FPGA synthesis tool infers SmartFusion2 RAM1K18 with output registers qa and qb outside the RAM, in SLEs.

```
module test (clka,clkb,wea,addra,dataina,qa,web,addrb,datainb,qb);

parameter addr_width = 10;
parameter data_width = 18;

input clka,clkb,wea,web;
```

```
input [data_width - 1 : 0] dataina,datainb;
input [addr_width - 1 : 0] addra,addrb;

output reg [data_width - 1 : 0] qa,qb;

reg [addr_width - 1 : 0] addra_reg, addrb_reg;
reg [data_width - 1 : 0] mem [(2**addr_width) - 1 : 0] ;

always @ (posedge clka)
begin
   addra_reg <= addra;
   if(wea)
   mem[addra] <= dataina;

end

always @ (posedge clkb)
begin
   addrb_reg <= addrb;
   if(web)
    mem[addrb] <= datainb;
end

always @ (posedge clka)
begin
   if(~wea)
   qa <= mem[addra_reg];
   else qa <= dataina;
end


always @ (posedge clkb)
begin

   if(~web)
   qb <= mem[addrb_reg];
   else qb <= datainb;
end

endmodule
```



Resource Usage Summary for test

Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT2 use
RAM1K181 use
CFG336 uses

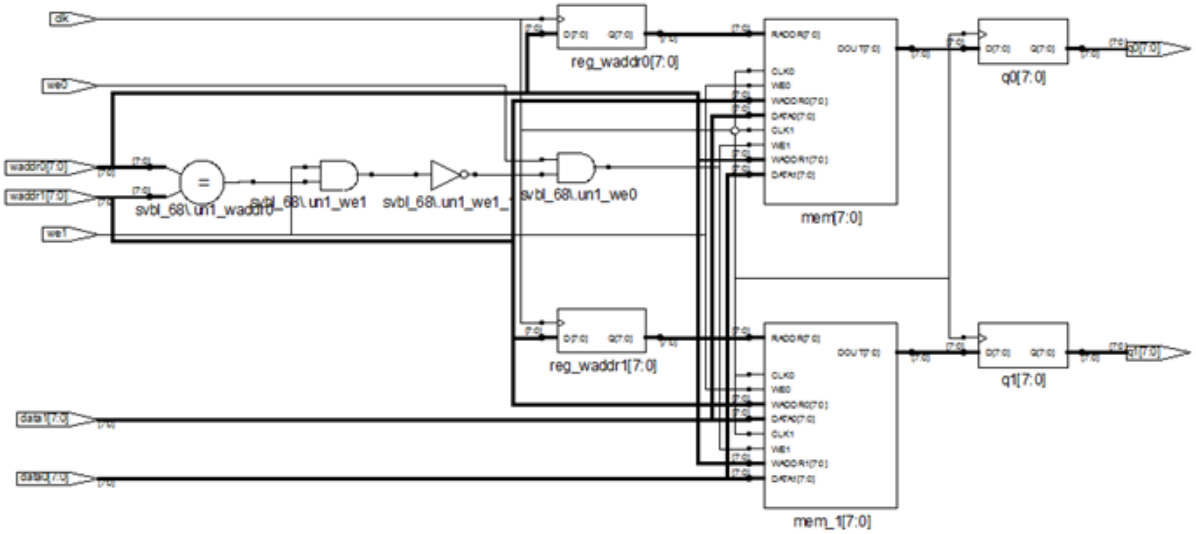
Sequential Cells:
SLE36 uses

Example 35: True Dual-Port RAM with Pipelined Register

The following design is a true dual-port RAM with two read and write ports and one clock with one pipelined register. The FPGA synthesis tool infers SmartFusion2 RAM1K18.

```
module ram_dport_reg(data0,data1,waddr0, waddr1,we0,we1,clk,q0, q1);

parameter d_width = 8;
parameter addr_width = 8;
parameter mem_depth = 256;

input [d_width-1:0] data0, data1;
input [addr_width-1:0] waddr0, waddr1;
input we0, we1, clk;

output [d_width-1:0] q0, q1;

reg [d_width-1:0] mem [mem_depth-1:0] ;
```

```
reg [addr_width-1:0] reg_waddr0, reg_waddr1;
reg [d_width-1:0] q;

reg [d_width-1:0] q0, q1;


always @(posedge clk)
begin
if (we0)
mem[waddr0] <= data0;
if (we1)
mem[waddr1] <= data1;

reg_waddr0 <= waddr0;
reg_waddr1 <= waddr1;


end

always @ (posedge clk)
begin
   q0 <= mem[reg_waddr0];
   q1 <= mem[reg_waddr1];
end
endmodule
```



Resource Usage Summary for ram_dport_reg

Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT1 use
RAM1K181 use
CFG3 1 use
CFG45 uses

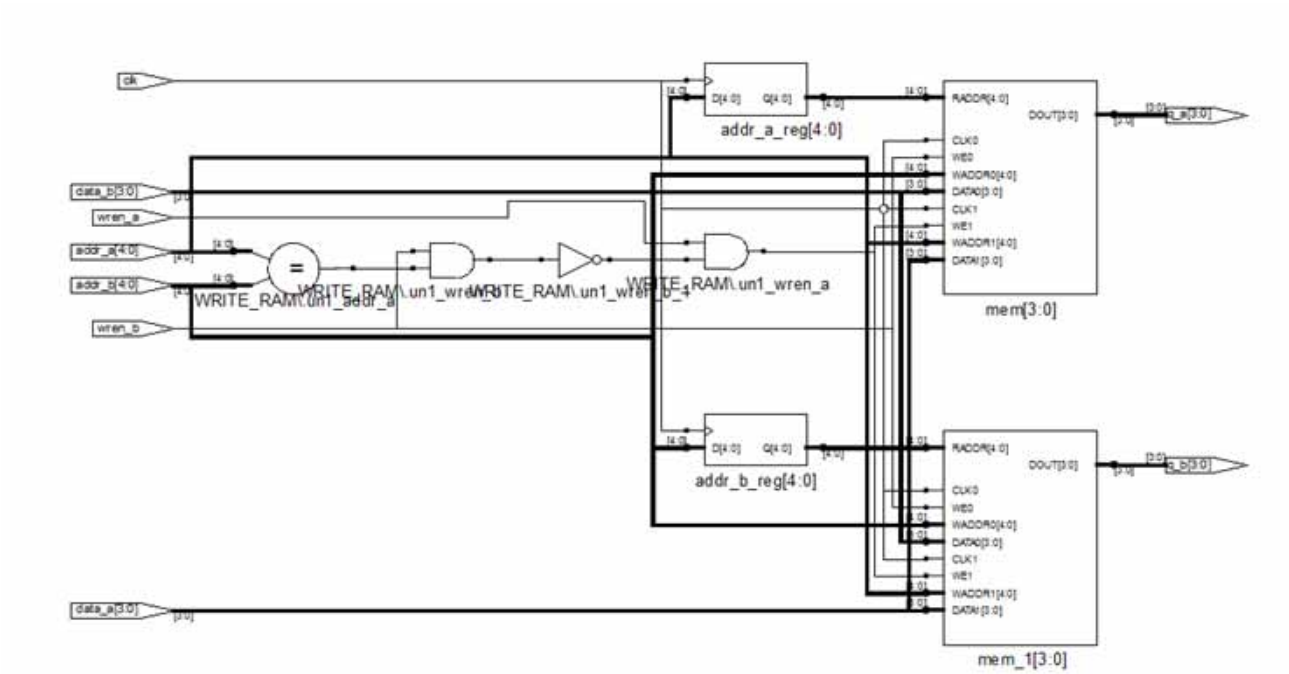
Sequential Cells:
SLE0 uses

Example 36: True Dual-Port RAM with Read Address Register (VHDL)

The following design is a true dual-port RAM with read address register. The FPGA synthesis tool infers SmartFusion2 RAM1K18.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity ram_dp_reg is
generic (data_width : integer := 4;
address_width :integer := 5 );
port (data_a:in std_logic_vector(data_width-1 downto 0);
data_b:in std_logic_vector(data_width-1 downto 0);
addr_a:in std_logic_vector(address_width-1 downto 0);
addr_b:in std_logic_vector(address_width-1 downto 0);
wren_a:in std_logic;
wren_b:in std_logic;
clk:in std_logic;
q_a:out std_logic_vector(data_width-1 downto 0);
q_b:out std_logic_vector(data_width-1 downto 0) );
end ram_dp_reg;


architecture rtl of ram_dp_reg is
type mem_array is array(0 to 2**(address_width) -1) of
std_logic_vector(data_width-1 downto 0);
signal mem : mem_array;


signal addr_a_reg : std_logic_vector(address_width-1 downto 0);
signal addr_b_reg : std_logic_vector(address_width-1 downto 0);
begin
   WRITE_RAM : process (clk)
      begin
         if rising_edge(clk) then
            if (wren_a = '1') then
               mem(to_integer(unsigned(addr_a))) <= data_a;
            end if;
               if (wren_b='1') then
               mem(to_integer(unsigned(addr_b))) <= data_b;
```

```
                end if;
                addr_a_reg <= addr_a;
                addr_b_reg <= addr_b;
            end if;
    end process WRITE_RAM;


    q_a <= mem(to_integer(unsigned(addr_a_reg)));
    q_b <= mem(to_integer(unsigned(addr_b_reg)));

    end rtl;
```



Resource Usage Summary for ram_simpledualport_outreg

Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT1 use
RAM1K181 use
CFG31 use
CFG43 uses

Sequential Cells:
SLE0 uses

Example 37: TDP RAM with Output Registered, Pipelined and Non-Pipelined (VHDL)

The following VHDL design is a true dual-port RAM with output registered. There is a generic version as well as pipelined and non-pipelined versions.

Generic Version

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity RAM_inference_examples is
     generic (data_width    :integer := 32;
             addr_width   :integer := 10;
              depth       :integer := 1024;
              testcase     :integer := 1); --- change to 1,2,3 to use variations in
coding style of Dual port RAM
     port(clk          :in std_logic;
              reset_n   :in std_logic;
              re_n     :in std_logic;
              we_n     :in std logic;
              data_in    :in std_logic_vector(data_width-1 downto 0);
              data_out   :out std_logic_vector(data_width-1 downto 0);
              addr_0    :in std_logic_vector(addr_width-1 downto 0);
              addr_1    :in std_logic_vector(addr_width-1 downto 0);
              r_wen_0   :in std_logic;
              r_wen_1   :in std_logic;

              data_in_0  :in std_logic_vector(data_width-1 downto 0);
              data_out_0 :out std_logic_vector(data_width-1 downto 0);
              data_in_1  : in std_logic_vector(data_width-1 downto 0);
              data_out_1  :out std_logic_vector(data_width-1 downto 0)
         );

end RAM_inference_examples;

architecture DEF_ARCH of RAM_inference_examples is
type mem_type is array (depth-1 downto 0) of std_logic_vector (data_width-1 downto
0);
signal BRAM_store   :mem_type;
signal int_addr_0   :integer range 0 to 4096;
signal int_addr_1   :integer range 0 to 4096;
signal rd_addr   :integer range 0 to 4096;
signal wr_addr   :integer range 0 to 4096;
signal data_out_tmp  :std_logic_vector(data_width-1 downto 0);
signal data_out_0tmp  :std_logic_vector(data_width-1 downto 0);
signal data_out_1tmp  :std_logic_vector(data_width-1 downto 0);

begin
```

Case 1 - Dual Port Without Pipelining and Registered data_out Ports

```
case_num1 : if testcase = 1 generate

int_addr_0  <= CONV_INTEGER(addr_0);
int_addr_1  <= CONV_INTEGER(addr_1);

process(clk)
begin
  if rising_edge(clk) then
  -- port 0
        if (r_wen_0 = '0') then
            BRAM_store(int_addr_0)    <= data_in_0;
       else
           data_out_0      <= BRAM_store(int_addr_0);
       end if;
  -- port 1
       if (r_wen_1 = '0') then
            BRAM_store(int_addr_1)   <= data_in_1;
       else
           data_out_1  <= BRAM_store(int_addr_1);

          end if;
    end if;
   end process;

end generate;
```

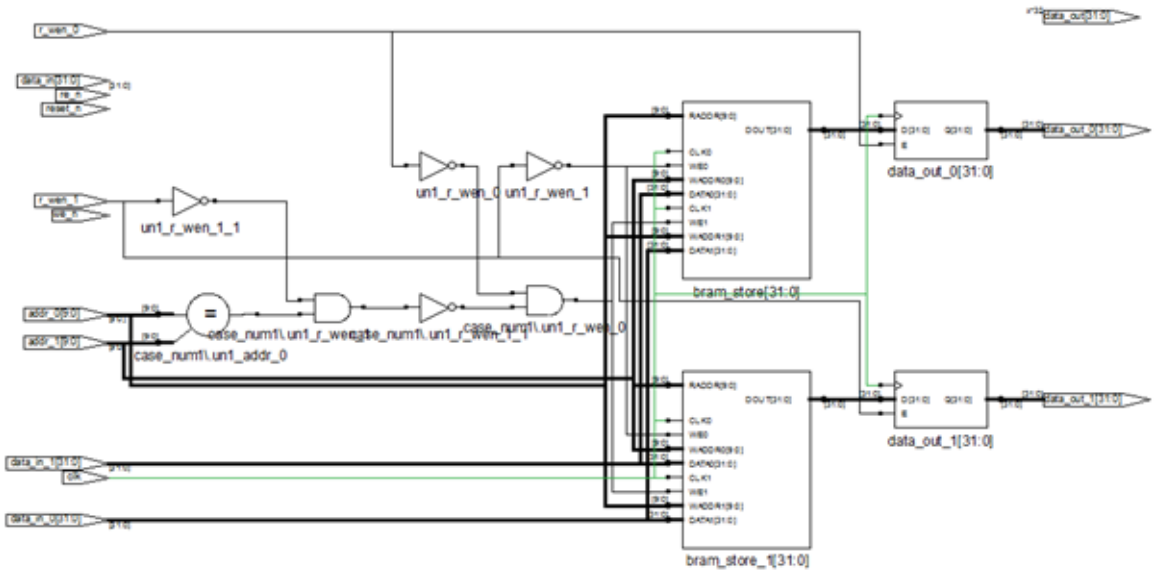
Case 2 - Dual Port with Pipelining and Registered data_out Ports

```
case_num2 : if testcase = 2 generate

  int_addr_0 <= CONV_INTEGER(addr_0);
  int_addr_1 <= CONV_INTEGER(addr_1);

  process(clk)
  begin
          if rising_edge(clk) then
          -- port 0
                 if (r_wen_0 = '0') then
                        BRAM_store(int_addr_0)   <= data_in_0;
                 else
                        data_out_0tmp   <= BRAM_store(int_addr_0);
                 end if;
          -- port 1
                 if (r_wen_1 = '0') then
                          BRAM_store(int_addr_1) <= data_in_1;
                  else
                          data_out_1tmp <= BRAM_store(int_addr_1);
                  end if;

                  data_out_0<= data_out_0tmp;
                  data_out_1<= data_out_1tmp;
   end if;
  end process;
end generate;

end def_arch;
```

Case 3 - Dual Port with Pipelining and Registered Read Address

```
    case_num3 : if testcase = 3 generate

       process(clk)
       begin
             if rising_edge(clk) then
             -- port 1
                   if (r_wen_0 = '0') then
                         BRAM_store(int_addr_0) <= data_in_0;
                   else
                         int_addr_0 <= CONV_INTEGER(addr_0);
                   end if;
             -- port 1
                   if (r_wen_1 = '0') then
                         BRAM_store(int_addr_1) <= data_in_1;
                   else
                         int_addr_1 <= CONV_INTEGER(addr_1);
                   end if;

                   data_out_0 <= BRAM_store(int_addr_0);
                   data_out_1 <= BRAM_store(int_addr_1);

          end if;
       end process;
    end generate;

    end def_arch;
```

Results of Generic Case Set to 1



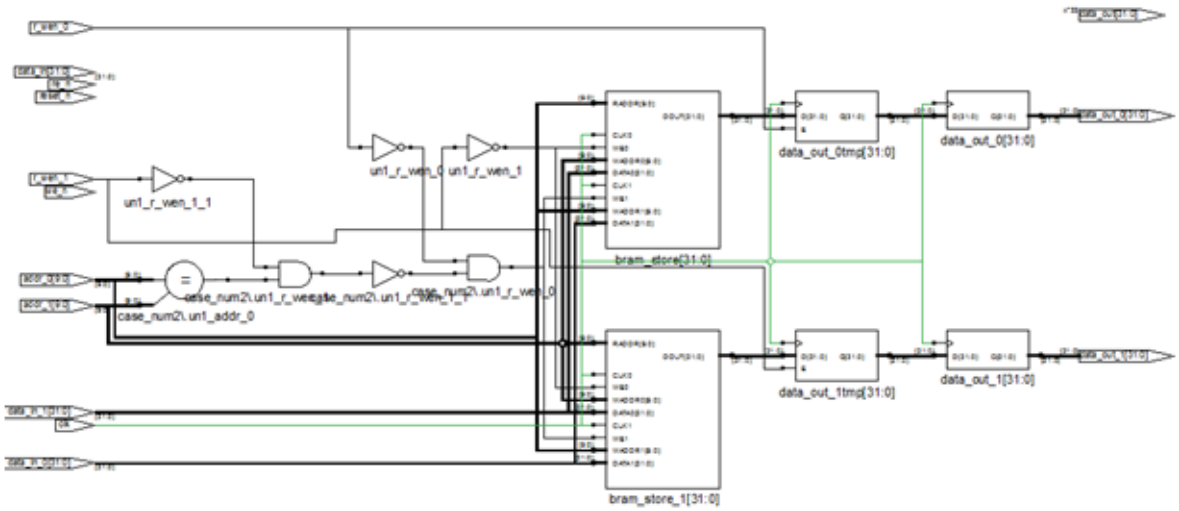
The FPGA synthesis tool infers SmartFusion2 RAM1K18.

Resource Usage Report for RAM_inference_examples, Case 1

Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT 1 use
RAM1K18 2 uses
CFG1 2 uses
CFG4 7 uses

Sequential Cells:
SLE 0 uses

Results of Generic Case Set to 2



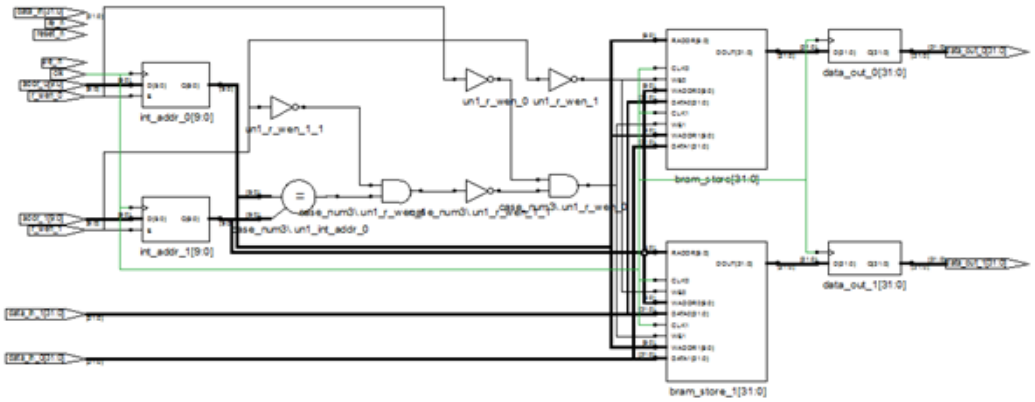
The FPGA synthesis tool infers SmartFusion2 RAM1K18.

Resource Usage Report for RAM_inference_examples, Case 2

Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT 1 use
RAM1K18 2 uses
CFG1 2 uses
CFG4 7 uses

Sequential Cells:
SLE 0 uses

Results of Generic Case Set to 3



The FPGA synthesis tool infers SmartFusion2 RAM1K18.

Resource Usage Report for RAM_inference_examples, Case 3

Mapping to part: m2s050tfbga896std
Cell usage:

CLKINT	1 use
RAM1K18	2 uses
CFG1	2 uses
CFG2	1 use
CFG3	1 use
CFG4	6 uses

Sequential Cells:

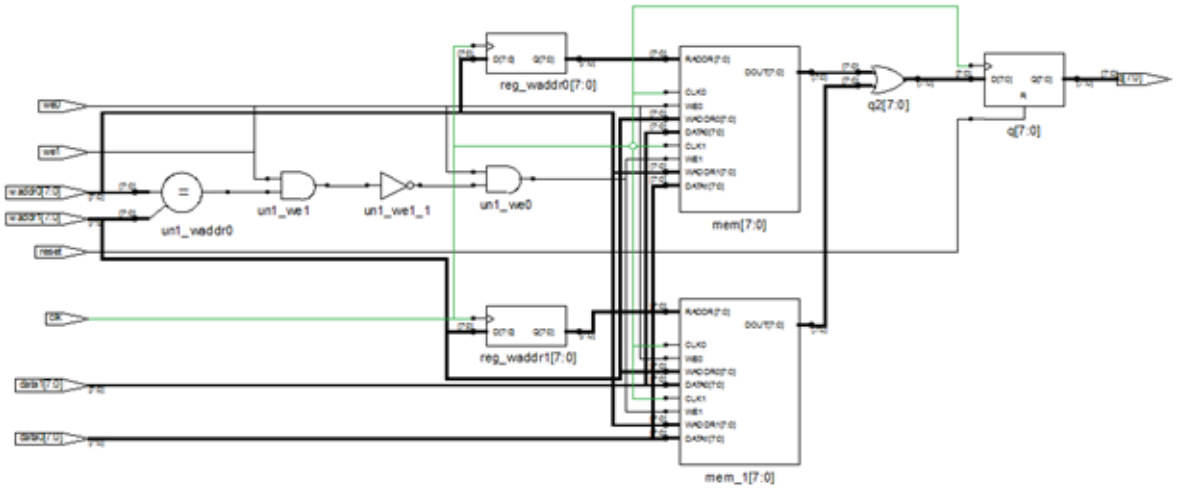
SLE	20 uses

Example 38: TDP RAM with Asynchronous Reset for Pipelined Register, RAM1K18

The following design is a true dual-port RAM with asynchronous reset for pipelined register. The FPGA synthesis tool infers SmartFusion2 RAM1K18.

```
module ram_dport_addreg_pipe_areset(data0,data1,waddr0,
waddr1,we0,we1,clk,q,reset); parameter d_width = 8;
parameter addr_width = 8;
parameter mem_depth = 256;
input [d_width-1:0] data0, data1;
input [addr_width-1:0] waddr0, waddr1;
input we0, we1, clk,reset;
output [d_width-1:0] q;
reg [d_width-1:0] mem [mem_depth-1:0];
reg [addr_width-1:0] reg_waddr0, reg_waddr1;
reg [d_width-1:0] q;

wire [d_width-1:0] q0, q1;
wire [d_width-1:0] q2;
```

```
assign q2 = q0 | q1;
assign q0 = mem[reg_waddr0];
assign q1 = mem[reg_waddr1];

always @(posedge clk)
begin
if (we0)
mem[waddr0] <= data0;
if (we1)
mem[waddr1] <= data1;
reg_waddr0 <= waddr0;
reg_waddr1 <= waddr1;
end

always @(posedge clk or posedge reset)
begin
  if(reset)
     q <= 0;
   else
     q <= q2;
 end
 endmodule
```



Resource Usage Report for ram_dport_addreg_pipe_areset

Mapping to part: m2s050tfbga896std
Cell usage:
CLKINT 1 use
RAM1K18 1 use
CFG1 1 use
CFG2 8 uses
CFG3 1 use
CFG4 5 uses

Sequential Cells:
SLE 8 uses

Multiport RAM Examples

The following are examples of coding to infer multiport RAM:

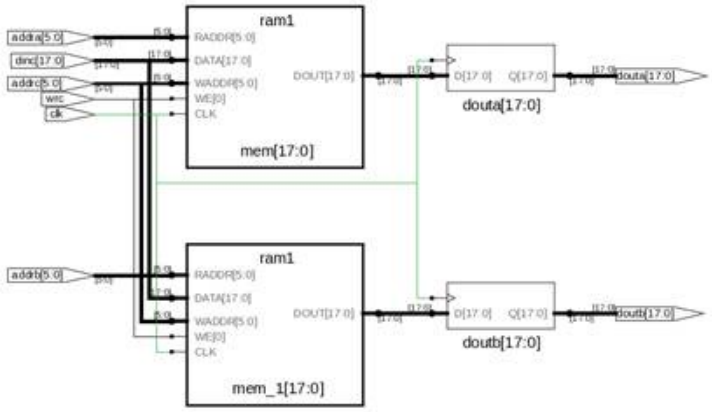
- Example 39: Three-Port RAM with Synchronous Read, on page 65
- Example 40: Three-Port RAM with Asynchronous Read, on page 66
- Example 41: Three-Port RAM with Read Address and Pipelined Register, on page 68

Example 39: Three-Port RAM with Synchronous Read

The following design is a Verilog example for three-port RAM with synchronous read. The tool infers one RAM64X18.

```
module ram_infer15_rtl(clk,dinc,douta,doutb,wrc,addra,addrb,addrc);
input clk;
input [17:0] dinc;
input wrc;
input [5:0] addra,addrb,addrc;
output [17:0] douta,doutb;
reg [17:0] douta,doutb;
reg [17:0] mem [0:63];
always@(posedge clk)
begin
if(wrc)
mem[addrc] <= dinc;
end

always@(posedge clk)
begin
douta <= mem[addra];
end
```

```
always@(posedge clk)
begin
doutb <= mem[addrb] ;
end
endmodule
```



Resource Usage Summary for ram_infer15_rtl

Mapping to part: m2s050tfbga896std

Cell usage:
CLKINT 1 use
RAM64x18 1 use

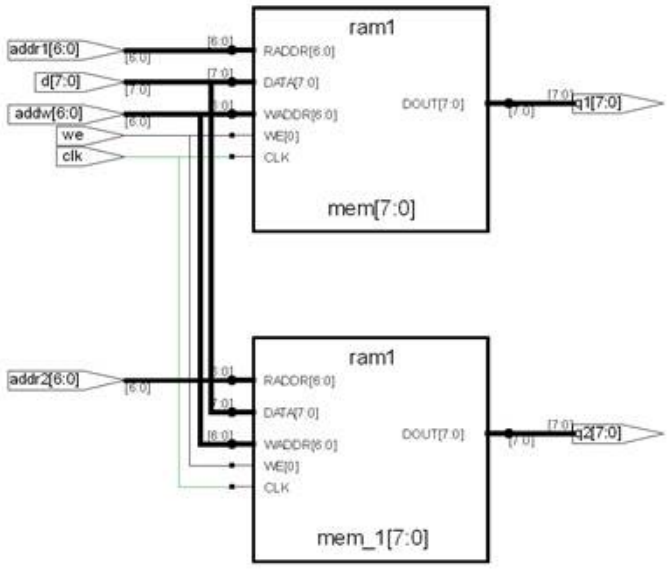
Sequential Cells:
SLE 0 uses

Example 40: Three-Port RAM with Asynchronous Read

The following design is a VHDL example for three-port RAM with asynchronous read. The tool infers one RAM64X18.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity ram_singleport_noreg is
port (d : in std_logic_vector(7 downto 0);
addw : in std_logic_vector(6 downto 0);
addr1 : in std_logic_vector(6 downto 0);
addr2 : in std_logic_vector(6 downto 0);
we : in std_logic;
clk : in std_logic;
q1 : out std_logic_vector(7 downto 0);
q2 : out std_logic_vector(7 downto 0) );
end ram_singleport_noreg;
architecture rtl of ram_singleport_noreg is
```

```
type mem_type is array (127 downto 0) of
std_logic_vector (7 downto 0);
signal mem: mem_type;
begin
process (clk)
begin
if rising_edge(clk) then
if (we = '1') then
mem(conv_integer (addw)) <= d;
end if;
end if;
end process;
q1<= mem(conv_integer (addr1));
q2<= mem(conv_integer (addr2));
end rtl;
```



Resource Usage Report for ram_singleport_noreg

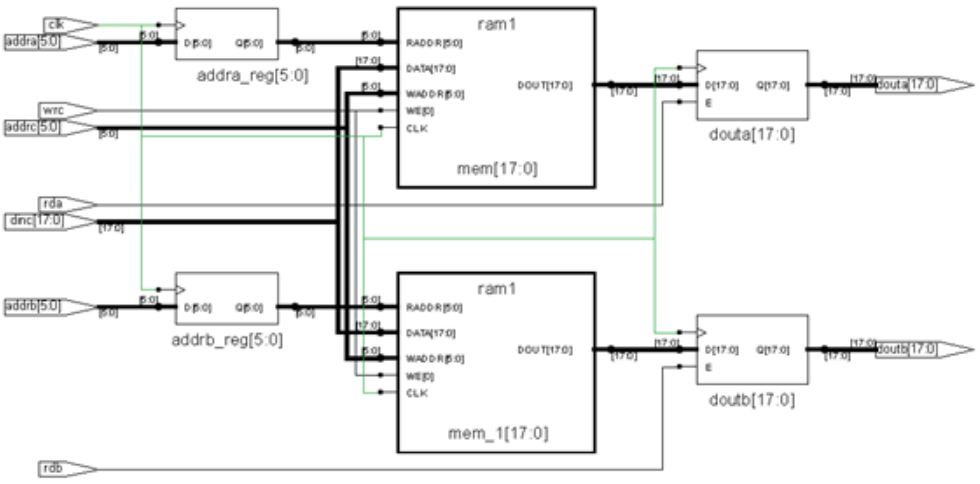
Mapping to part: m2s050tvf400std
Cell usage:
RAM64x18 1 use

Sequential Cells:
SLE 0 uses

Example 41: Three-Port RAM with Read Address and Pipelined Register

The following design is an example for three-port RAM with read address and pipelined register. The tool implements one RAM64X18.

```
module ram_infer(clk,dinc,douta,doutb,wrc,rda,rdb,addra,addrb,addrc);
input clk;
input [17:0] dinc;
input wrc,rda,rdb;
input [5:0] addra,addrb,addrc;
output [17:0] douta,doutb;
reg [17:0] douta,doutb;
reg [5:0] addra_reg, addrb_reg;

reg [17:0] mem [0:63];
always@(posedge clk)
begin
addra_reg <= addra;
addrb_reg <= addrb;

if(wrc)
mem[addrc] <= dinc;
end

always@(posedge clk)
begin
if(rda)
douta <= mem[addra_reg];
end

always@(posedge clk)
begin
if(rdb)
doutb <= mem[addrb_reg] ;
end
endmodule
```



Resource Usage Report for ram_infer

Mapping to part: m2s050tvf400std
Cell usage:
CLKINT 1 use
RAM64x18 1 use

Sequential Cells:
SLE 0 uses

Current Limitations

For successful SmartFusion2 RAM inference with the Synplify Pro software, it is important that you use a supported coding structure, because there are some limitations to what the synthesis tool infers. Currently, the tool does not support the following:

- ROM inference is not supported for RAM1K18 and RAM64X18.
- Large RAMs are broken down into multiple RAM64X18 or RAM1K18 blocks; only one type of RAM block can be used.
- RAMs which can be mapped into a single RAM primitive, are not fractured on the address to infer multiple RAM blocks.