Last updated 5/15/20

Counters

These slides review the design for several types of memories

Upon completion: You should be able to design ROMs and RAMS of various sizes and register configurations

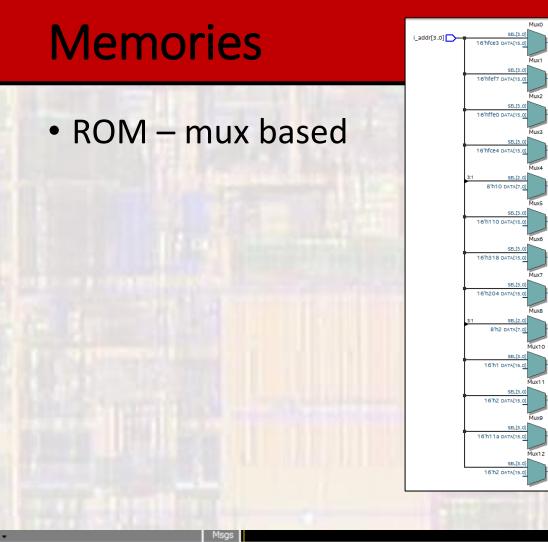
© ti

 ROM – mux based with memory values stored as constants

ROM – mux based



© ti





- 🔁 🗸		Msgs																		
	/rom_muxbased_constants_tb/CLK	0	LUU	h	inn	<u>nnr</u>	L L L		T	LUU	<u>nn</u>	inr	LD.		<u> </u>		inr			
H -4	/rom_muxbased_constants_tb/ADDR	7	0	(1	2	3	4	5	6	7	8	9	<u>)</u> 10	11	12	(13	14	15	(0	
E-	/rom_muxbased_constants_tb/DATA_OUT	F000	C010) C0	4A (5180	<u>(02C0</u>	4640	(F000			2E40	6B00	<u>(</u> F000					ÍT	(<u>C010</u>	
																		AV		-
A 🖬 🖲	Now	000 ps	DS	100	000 ps	2000	oloo ps	dunn	30000	00 ps	4000	000 ps		50000	00 ps	600	000 ps		700000 p	
🔒 🧞 🌔	Cursor 1	0 ps	0 ps																	

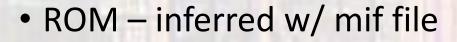
Ţ

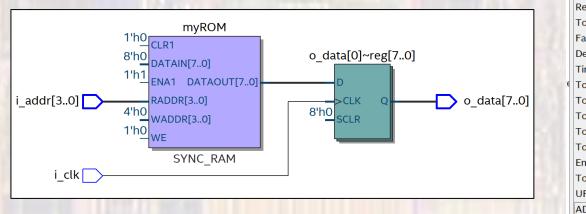
 ROM – inferred with memory values stored in a xx.mif file

• ROM – inferred w/ mif file

```
-- rom_inferred.vhdl
---
-- created 4/25/17
-- ti
--
-- rev 0
-- 16B synchronous ROM loaded from file
-- Inputs: clk, addr
-- Outputs: data
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity rom_inferred is
   port(
               in std_logic;
      i_clk:
      i_addr: in std_logic_vector(3 downto 0);
      o_data: out std_logic_vector(7 downto 0)
   ):
end:
```







Flow Status	Successful - Sun May 17 16:5
Quartus Prime Version	19.1.0 Build 670 09/22/2019
Revision Name	Class_Examples
Top-level Entity Name	rom_inferred
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Einal
Total logic elements	0
Total registers	
Total pins	13
Total virtual pins	<u> </u>
Total memory bits	128
Embedded Multiplier 9-bit elements	
Total PLLs	0
UFM blocks	0
ADC blocks	0

• Cannot be simulated

	/rom_inferred_tb/CLK		າທາທາ	າທາກ	າທາກ	າທາກ	ທ່ານເປ	າທາກ	າກຖາກ	սոու	ທາກ	າທາກ	MMM	າທາກ	າທາກທ	M	າທາກ	າທາກແ	MM	M	MMM	ທາກາ	ທ່ານານ	www	Ŋ	-
	/rom_inferred_tb/A		0 (0							(0)			Ľ				<u>) (o</u>	3	
E -4	/rom_inferred_tb/D	UUUUUUU	UUUUUL	JUU																						-
	Now	2000 ns) ns	20	lı) ns	4	00 ns		l 600 n:	i I s	800) ns	100	l i 0 ns	120	l i Ons	1	400 ns		1600	i l) ns	18	00 ns	20	l i 00 ns	
- 🖻 🥓	Cursor 1	0.00 ns	0.00 ns																							

• SRAM – synchronous write – generic

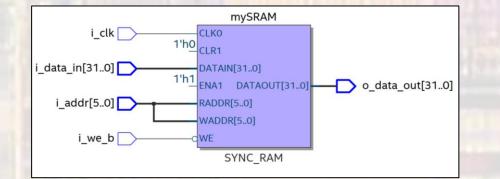
No inferred memory

SRAM – register based

	create type	
 sram_regbased.vhdl	type sram_type is array (0 to (mem_depth - 1)) of std_logic_vector ((mem_width - 1) downtr	<mark>o 0</mark>);
 created 4/25/17	create memory	
tj 	signal mySRAM: sram_type;	
rev 0	begin	
 synchronous RAM built with registers Inputs: clk, addr, we_b, data_in Outputs: data_out	 SRAM write process process(i_clk) begin if (rising_edge(i_clk)) then write logic if(i_we_b = '0') then	
<pre>library ieee; use ieee.std_logic_1164.all; use ieee.numeric_std.all; use ieee.math_real.all; entity sram_regbased is generic(mem_width: positive := 32; mem_depth: positive := 64);</pre>	<pre>mysRAM(to_integer(unsigned(i_addr))) <= i_data_in; end if; end if; end process; SRAM asynchronous read o_data_out <= mySRAM(to_integer(unsigned(i_addr))); end behavioral;</pre>	
<pre>port(i_clk: in std_logic; i_we_b: in std_logic; i_addr: in std_logic_vector((i_data_in: in std_logic_vector((o_data_out: out std_logic_vector(()))))))))))))))))))))))))))))))))))</pre>	<pre>(integer(ceil(log2(real(mem_depth)))) - 1) downto 0); mem_width - 1) downto 0); mem_width - 1) downto 0)</pre>	
); end entity;	32 bits x 64 words	
sealed the local factor of	4 bytes x 64 words	
3921	8x4x64 bits (registers) = 2048	© tj

architecture behavioral of sram_regbased is

SRAM – register based



Flow Summary <<Filter>> Flow Status Successful - Fri May 15 11:1 **Ouartus Prime Version** 19.1.0 Build 670 09/22/207 **Revision Name Class Examples** Top-level Entity Name sram_regbased Family **MAX 10** Device 10M50DAF484C7G Timing Models Final Total logic elements 3,472 Total registers 2048 Total pins Total virtual pins Total memory bits Embedded Multiplier 9-bit element. 0 Total PLLs 0 UFM blocks 0 ADC blocks 0

SRAM – register based

```
-- Run Process
run: Process
                   -- note - no sensitivity list allowed
begin
   -- Initalize values
  ADDR <= (others => '0');
DATA_IN <= (others => '0');
WE_B <= '1';
   -- Read from a few addresses
   for i in 0 to 9 loop
      wait for 2*PER;
      ADDR <= std_logic_vector(to_unsigned(i*250,(integer(ceil(log2(real(mem_depth)))))));
   end loop;
   -- Write to a few addresses
   for i in 0 to 9 loop
      wait for 1*PER;
      ADDR <= std_logic_vector(to_unsigned(i*250,(integer(ceil(log2(real(mem_depth)))))));
      DATA_IN <= std_logic_vector(to_unsigned(i*5, mem_width));</pre>
      WE_B <= '0';
      wait for 1*PER;
      WE_B <= '1':
   end loop;
      -- Read from a few addresses
   for i in 0 to 9 loop
      wait for 2*PER;
      ADDR <= std_logic_vector(to_unsigned(i*250,(integer(ceil(log2(real(mem_depth)))))));</pre>
   end loop;
end process run;
```

Wave - Default														; •							+ e								
\$ 2. .		Msgs																											
♦ /sram_tb/CLK	0		பி	UU		hun	INN	hu	inn	nn	INN	hu	1 I I I	ปาม	านน	ากป	ากก	INN	hn	чл	ЛĻ	ЛЦ	ЛЛ	hu		ปบ	าน	Ъ	in r
	1500		0	(250) (500	750	<u>1000 (</u>	1250 (15	00 (1750	2000	1 10	250	500	750	1000	1250	1500	1750	2000	2250) ()	(250	500 ((750	1000	1250	1500	1750	2000
<pre>/sram_tb/WE_B</pre>	1													ΨГ															
	0		0									<u>(</u> 5				25				45									
🛨 🔶 /sram_tb/DATA_OU	т 30		Х								0	5	10	15	20	25	30	35	40	45) ()	(5	(10	(15	20	25	(30	(35	(40)
r																													
					-			-																					
		- 🔶 /	sram_1	128b_tb	/CLK		0			הן ו							\square	n f	٦Г	ר ר	1 (ПГ				
1.		− → /					30	160	: (00		C	18	<u> </u>	24	13	30) 3C		(48		154	i	50	(6C					
					WE_B		1			Ē				ſ									Г						
							45	0			5	10		15		20	25		(30		35	- F	40	(45					
							20	Ť	10											10									
and the second second			sram_i	1200_00	DATA_	001	_ 20	<u>×</u>	(0	^	(5	(1	<u> </u>	15		(20		25		iU	(3	5	40		45				
EE 3921																													© tj

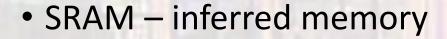
- SRAM synchronous read/write generic
 - Inferred memory

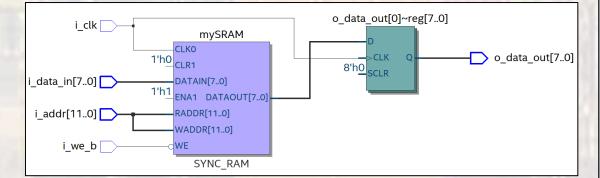
SRAM – inferred memory

```
-- create type
                                                 type sram_type is array (0 to (mem_depth - 1)) of std_logic_vector ((mem_width - 1) downto 0);
-- sram inferred.vhdl
                                                 -- create memory
-- created 4/25/17
                                                 ---
-- ti
                                                 signal mySRAM: sram_type;
-- rev 0
                                                begin
-- synchronous RAM using inferred memories
                                                    -- SRAM write process
                                                   process (i_clk)
                                                   begin
-- Inputs: clk, addr, we_b, data_in
                                                       if (rising_edge(i_clk)) then
-- Outputs: data_out
                                                          -- read logic
                                                          if(i_we_b = '0') then
                                                             mySRAM(to_integer(unsigned(i_addr))) <= i_data_in;</pre>
library ieee;
                                                          end if;
use ieee.std_logic_1164.all;
                                                          --registered output
use ieee.numeric_std.all;
                                                          o_data_out <= mySRAM(to_integer(unsigned(i_addr)));</pre>
use ieee.math_real.all;
                                                      end if;
                                                   end process;
entity sram_inferred is
   generic(
                                             end behavioral;
      mem_width: positive := 8:
      mem_depth: positive := 4096
   ):
   port(
      i_clk:
                        std_logic;
                  in
                  in std_logic;
      i_we_b:
                        std_logic_vector(((integer(ceil(log2(real(mem_depth))))) - 1) downto 0);
      i_addr:
                  in
                        std_logic_vector((mem_width - 1) downto 0);
      i_data_in: in
      o_data_out: out std_logic_vector((mem_width - 1) downto 0)
   );
end:
```

architecture behavioral of sram_inferred is

8 bits x 4096 words = 32,768 bits





Flow Status	Successful - Fri May 15 11:2
Quartus Prime Version	19.1.0 Build 670 09/22/201
Revision Name	Class_Examples
Top-level Entity Name	sram_inferred
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	0
Total registers	0
Total pins	30
Total virtual pins	0
Total memory bits	32,768
Embedded Multiplier 9-bit elements	0
Total PLLs	0
UFM blocks	0
ADC blocks	0

SRAM – inferred memory

₩	Msgs												
/sram_sync_rw_tb/CLK	0	JUUUU	nnn	mm	nnn	mm	M	mm	huuu	mm	M	nnn	nnn
	000	000 (0	<u>(</u> 1F4 <u>(</u> 2EE	<u>(3E8 (4E2 (5.</u>	<u>(6D6 (</u> 7	<u>(000 (0FA</u>	1F4 (2EE) 3E	8 (4E2 (5	<u>(6 (700 (80</u>	:A (000 (0F	<u>A (1F4 (2EE</u>	3E8 (4E2) 5.	<u>(6 (7</u> 00)
/sram_sync_rw_tb/WE_B	1												
Image: stam_sync_rw_tb/DATA_IN	0	0				<u>(</u> 5	10 15 20) (25 (30	35 (40 (49	i			
	40	Х								(45) 0) !	5 (10 (15	<u>) 20 (25)</u>	30 <u>(35 (40</u>

/sram_sync_rw_tb/CLK	0																	
	000	000	(OFA	1F4		2EE		3E8		4E2	ĭ	5DC		6D6		7D0	1	8CA
/sram_sync_rw_tb/WE_B	1																	
	0	0	5	10		15		20		25	X	30		35		40		45
/sram_sync_rw_tb/DATA_OUT	40	Х	(0 X	(5 X	10	X	15	Х	(20	X	25	Х	(30	X	35	X	(40	X