Reference DESIGN

Parameterizable Distributed RAM for Virtex (VHDL)

March 15, 1999

Application Note by Daniel Michek

Xilinx, Inc.
2100 Logic Drive
San Jose, CA 95124
Phone: +1 408-559-7778
FAX: +1 408-559-7114
Email: coregen@xilinx.com
URL: http://www.xilinx.com/ipcenter

Features

- Parameterizable VHDL RAMs
- Supports Xilinx Virtex architecture
- Uses SelectRAM™ for more efficient storage
- Supports memory depths from 64 to 8192 in 16 bit increments
- Source files for Single Port / Dual Port versions
- Selectable registered inputs/outputs
- Selectable input/output clock enables
- Selectable multiplexing scheme for the combination of RAMs

Functional Description

This source code implements an N-bit data width by an L-word depth memory array for Virtex devices. VHDL generics are used to allow the customization of the module without making changes to the source file.

The use of generics also allows the user the ability to instantiate multiple memory arrays of different sizes at the user level.

Using VHDL Generics to Specify Inputs

The incoming signal control via generics is shown in Figure 1. When the generic is used to multiplex different signals, the unused logic is trimmed due to the fact that it is loadless.

The data, **DI**, address, **A**, and dual port address, **DPRA** (for dual port RAM only) use two generics to determine how the signals reach the RAM array. The first generic, *input_registered*, selects registered or non-registered versions of **DI**, **A**, and **DPRA** to appear at the memory array. The second generic, *clken_for_input*, assigns either the input clock enable, **ICE**, or a logic level high to appear at the clock enable port for **DI**, **A**, and **DPRA** registers.

The RAM write-enable, **WE**, uses up to three generics (*input_registered*, *clken_for_input*, and *wrten_for_ram*) to determine the write enable for the RAM. The truth table for the memory array write enable is shown in Table 1.

Inputs (generics)			Outputs
Input_ registered	*clken_for_ input*	*wrten_for_ ram*	we_to_ram
0	X	0	1
0	X	1	WE
1	0	0	1 registered
1	0	1	WE registered
1	1	0	ICE registered
1	1	1	(ICE * WE) registered

Table 1: we_to_ram specification using generics

Using VHDL Generics to Specify Outputs

The outgoing signal control via generics is shown in Figure 2. When the generics used to multiplex different signals, the unused logic is trimmed due to the fact that it is loadless. The single port output, **SPO**, and the dual port output, **DPO**, use three generics (output_registered,

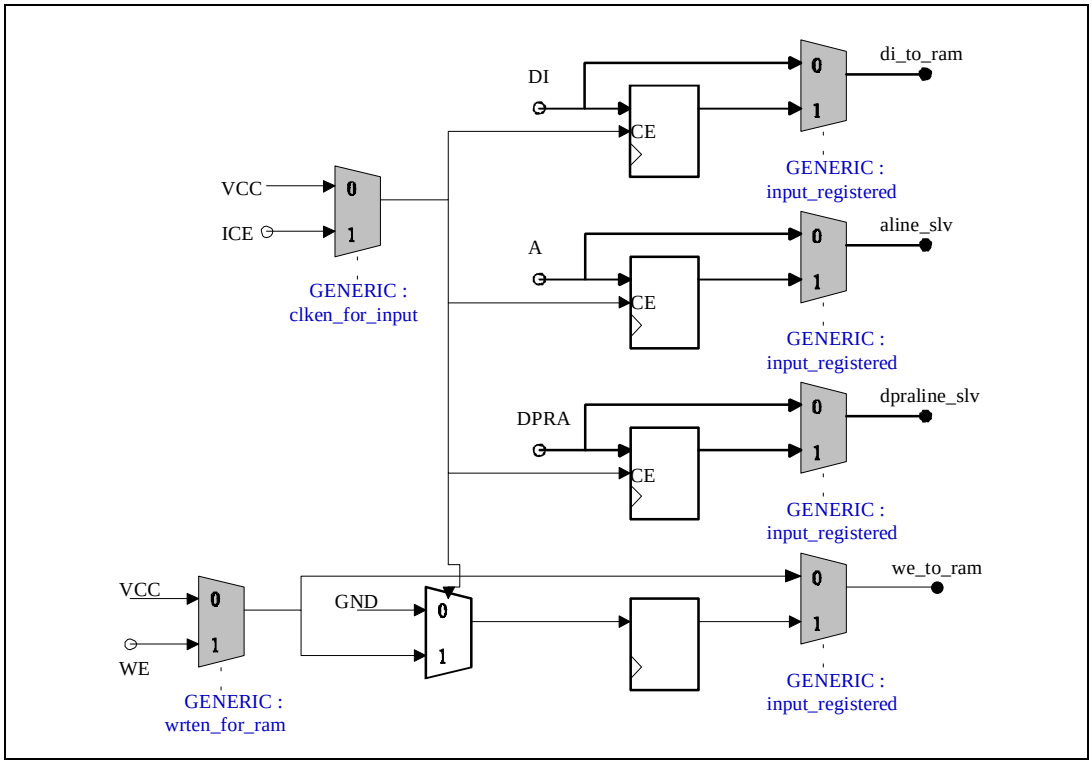

FIGURE 1: Input Generic Multiplexing

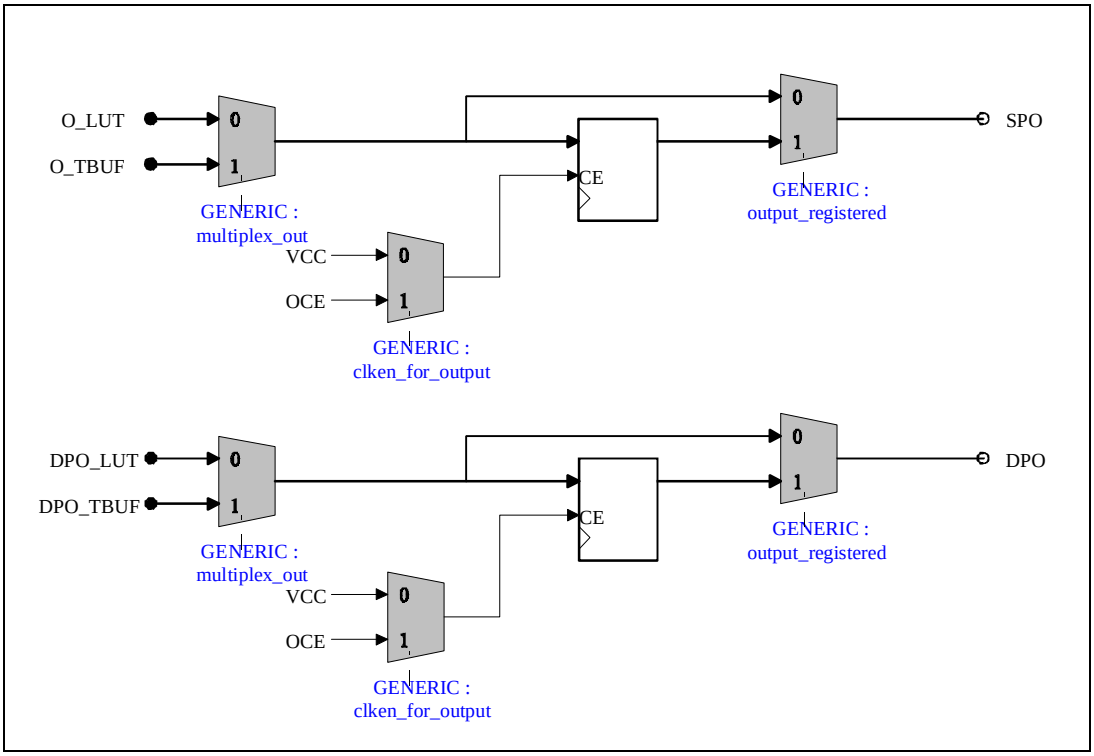

FIGURE 2: Use of VHDL Generics to control Output Multiplexing

clken_for_output, multiplex_out), and the output clock enable, OCE, to determine which output is implemented. The distributed RAM outputs are shown in Table 2.

Address Selection RAM Array

The single port RAM module utilizes the Xilinx primitive RAM32X1S. For arrays that are not multiples of 32 but are multiples of 16, the fifth address line of the most significant RAM32X1S is tied to a logic level low.

The dual port RAM module utilizes the Xilinx primitive RAM16X1D.

For an L-word deep single port RAM, num_alines address lines are required, where

$$num\_alines = RoundUp(Log_2(L-1)) = (M+1)$$

The lowest 5 bits, A (4 downto 0), are connected to every RAM32X1S in the array.

The upper bits, A (X downto 5), are used as follows: for each column of RAM generated (each column is 32 bits deep, generic *memory_depth* / 32 = number of columns) a write-enable to the RAM column is also generated. This is accomplished by bit-wise XORing the column number (constant arline_slv) with A (X downto 5) , followed by logic AND reduction. There are two Lookup tables required to implement the ANDs. The outputs of the two LUTs are used as the select for carry logic multiplexers (MUXCY). By driving the inputs to the MUXCYs with known logic values and the signal we_to_ram, an efficient decode is accomplished in 1 Virtex Slice. The resultant bit (1 if equal, 0 if not equal) is generated as per Table 1. When location constraints and FMAPs are used, a single Virtex slice can implement the write-enable control for a single column (see Figure 3) when the memory depth is less than or equal to 8192 bits. Larger depths of memory should not be used without modifying the VHDL code.

When tri-state multiplexing of the outputs is selected (multiplex_out = 1), the output of each RAM32X1S primitive drives an internal tri-state buffer and only one buffer is active for each bit of SPO. In this configuration a similar approach is used to control the tri-state buffers in a one-hot scheme. This is accomplished by bit-wise XORing the column number (constant arline_slv) with A (X downto 5), followed by logic AND reduction. There are two Lookup tables required to implement the ANDs. The outputs of the two LUTs are used as the select for carry logic multiplexers (MUXCY). By driving the inputs to the MUXCYs with known logic values, an efficient decode is accomplished in 1 Virtex Slice. The resultant bit (1 if equal, 0 if not equal) is inverted and connected to the T input of the Xilinx primitive BUFT. When location constraints and FMAPs are used, a single Virtex slice can implement the write-enable control for a single column (see Figure 4) when the memory depth is less than or equal to 8192 bits. Larger depths of memory should not be used without modifying the VHDL code.

Inputs				**Outputs**	
output_registered	*clken_for_output*	*multiplex_out*	OCE	SPO	DPO
0	X	0	X	o_lut	dpo_lut
0	X	1	X	o_tbuf	dpo_tbuf
1	0	0	X	o_lut registered	dpo_lut registered
1	0	1	X	o_tbuf registered	dpo_tbuf registered
1	1	X	0	SPO	DPO
1	1	0	1	o_lut registered	dpo_lut registered
1	1	1	1	o_tbuf registered	dpo_tbuf registered

Table 2: RAM outputs specification using generics and OCE

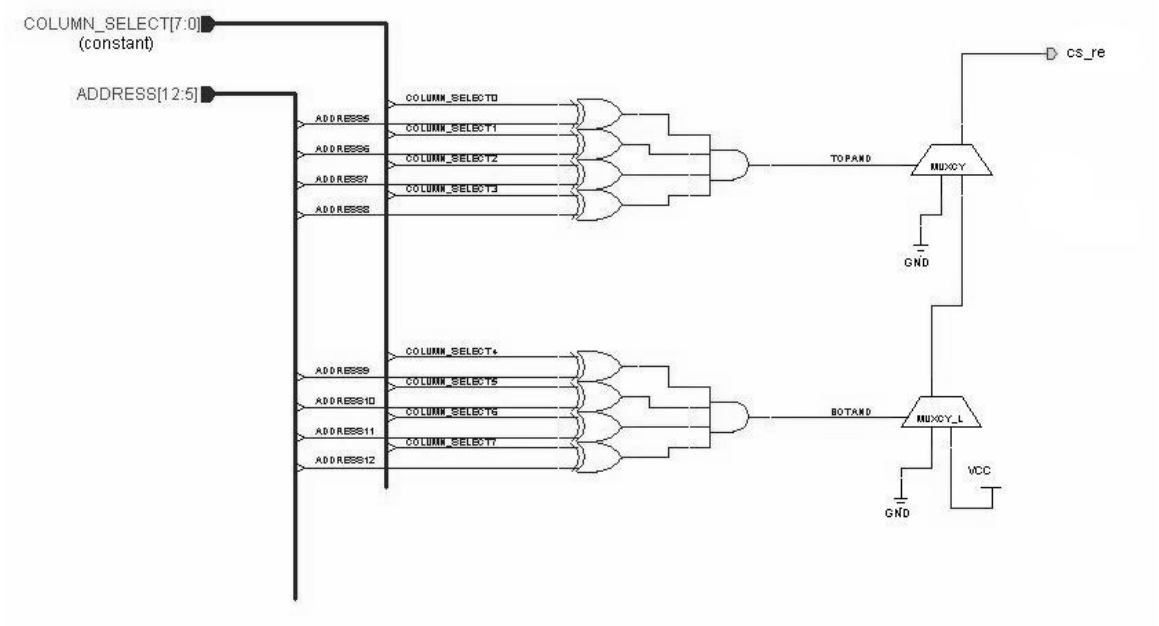

Figure 3: Generation of WE control for RAM column (generation of RAM column select).

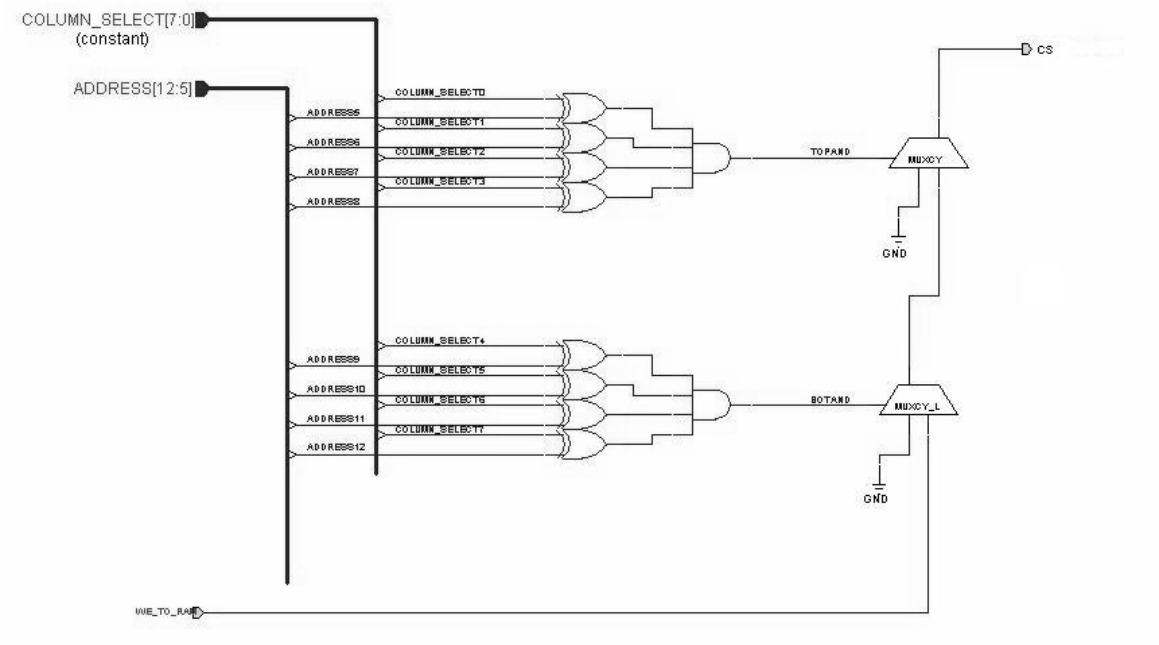

Figure 4: Generation of Tristate column select for *multiplex_out = 1* (tristates used for output muxing).

When lookup table (LUT) based multiplexing is used on the outputs, the upper address lines function as the select lines of a logic multiplexer. Each piece of input data to the multiplexer is an output from a RAM.

Port Name	Description
A [M:0]	Address bus
DI [N:0]	Input data port
ICE	Input clock enable (clock enable for A, DI, and WE)
OCE	Output clock enable
WE	Write enable for RAM array
WR	RAM array clock
SPO [N:0]	Single port output
Generic	**Description**
Data_width	Must be greater than or equal to 2
Memory_depth	Only multiples of 16 are allowed Must be greater than or equal to 48 and less than or equal to 8192
Num_of_alines	Integer greater than or equal to $\log_2$(memory_depth)
Wrten_for_ram	1 or 0 1 uses WE on the RAM 0 does not use WE
Input_registered	1 or 0 1 uses input registers 0 does not use input registers
Clken_for_input	1 or 0 1 uses clock enables for input registers 0 does not use clock enables for input registers
Output_registered	1 or 0 1 used output registers 0 does not use output registers
Clken_for_output	1 or 0 1 uses clock enables for output registers 0 does not use clock enables for output registers
Multiplex_out	1 or 0 1 uses tri-state multiplexing for RAM to outputs 0 uses lookup tables to multiplex RAM to outputs

Table 3: dram_sp.vhd pinout

Port Name	Description
A [M:0]	Address bus
DPRA [M:0]	Dual port Address bus
DI [N:0]	Input data port
ICE	Input clock enable (clock enable for A, DI, and WE)
OCE	Output clock enable
WE	Write enable for RAM array
WR	RAM array clock
SPO [N:0]	Single port output
DPO [N:0]	Dual port output
Generic	**Description**
Data_width	Must be greater than or equal to 2
Memory_depth	Only multiples of 16 are allowed Must be greater than or equal to 48 and less than or equal to 8192
Num_of_alines	Integer greater than or equal to $\log_2$(memory_depth)
Wrten_for_ram	1 or 0 1 uses WE on the RAM 0 does not use WE
Input_registered	1 or 0 1 uses input registers 0 does not use input registers
Clken_for_input	1 or 0 1 uses clock enables for input registers 0 does not use clock enables for input registers
Output_registered	1 or 0 1 used output registers 0 does not use output registers
Clken_for_output	1 or 0 1 uses clock enables for output registers 0 does not use clock enables for output registers
Multiplex_out	1 or 0 1 uses tri-state multiplexing for RAM to outputs 0 uses lookup tables to multiplex RAM to outputs

Table 4: dram_dp.vhd pinout

Pinouts and Instantiation Templates

For the single port RAM (dram_sp.vhd), port names are shown in the symbol in Figure 5. Port names, VHDL Generic names and their descriptions for this module are listed in Table 3.

For the dual port RAM (dram_dp.vhd), port names are shown in Figure 6, and port names and VHDL Generic names are listed in Table 4 with their descriptions.

The instantiation templates for dram_sp.vhd and dram_dp.vhd are shown in Tables 5 and 6, respectively.

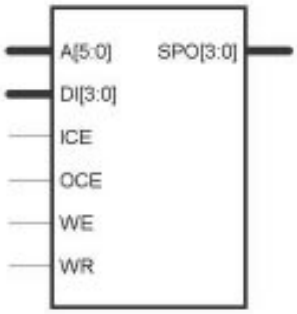

Figure 5: dram_sp.vhd symbol

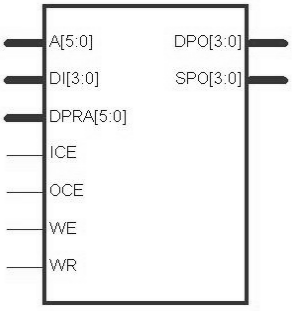

Figure 6: dram_dp.vhd symbol

```
component dram_sp
   generic (
    data_width    : integer := 16;
    memory_depth  : integer := 64;
    num_of_alines : integer := 6;
    wrten_for_ram     : integer := 0;
    clken_for_input   : integer := 0;
    clken_for_output  : integer := 0;
    );

   port (
    DI : in std_logic_vector
(data_width-1 downto 0);
    A  : in std_logic_vector (num_of_alines-1 downto
0);
    WR : in std_logic;
    SPO : out std_logic_vector (data_width-1 downto
0);
    WE  : in std_logic;
    ICE : in std_logic;
    OCE : in std_logic
    );
end component;

dut: dram_sp generic map (
   data_width=>data_width,
   memory_depth=>memory_depth,
   num_of_alines=>num_of_alines,
   wrten_for_ram=>wrten_for_ram,
   input_registered=>input_registered,
   clken_for_input=>clken_for_input,
   output_registered=>output_registered,
   clken_for_output=>clken_for_output,
   multiplex_out=>multiplex_out)
      port map (
   DI=>DI,
   A=>A,
   WR=>WR,
   SPO=>SPO,
   WE=>WE,
   ICE=>ICE,
   OCE=>OCE);
```

Table 5: dram_sp.vhi instantiation template

```
component dram_dp
   generic (
    data_width    : integer := 16;
    memory_depth  : integer := 64;
    num_of_alines : integer := 6;
    wrten_for_ram     : integer := 0;
    clken_for_input   : integer := 0;
    clken_for_output  : integer := 0;
    );

   port (
    DI : in std_logic_vector (data_width-1 downto 0);
    A  : in std_logic_vector
(num_of_alines-1 downto 0);
   DPRA  : in std_logic_vector (num_of_alines-1
downto 0);
    WR : in std_logic;
    SPO : out std_logic_vector (data_width-1 downto
0);
    DPO : out std_logic_vector (data_width-1 downto
0);
    WE  : in std_logic;
    ICE : in std_logic;
    OCE : in std_logic
    );
end component;

dut: dram_dp generic map (
   data_width=>data_width,
   memory_depth=>memory_depth,
   num_of_alines=>num_of_alines,
   wrten_for_ram=>wrten_for_ram,
   input_registered=>input_registered,
   clken_for_input=>clken_for_input,
   output_registered=>output_registered,
   clken_for_output=>clken_for_output,
   multiplex_out=>multiplex_out)
      port map (
   DI=>DI,
   A=>A,
   DPRA=>DPRA,
   WR=>WR,
   SPO=>SPO,
   DPO=>DPO,
   WE=>WE,
   ICE=>ICE,
  OCE=>OCE);
```

Table 6: dram_dp.vhi instantiation template

Example: How to specify a RAM using Generics

Example 1 shows the VHDL generic value assignments required to create a component instantiation for a 32 bit wide, 128 deep single port RAM array with output registers, output register clock enables, and an output-multiplexing scheme employing LUTs.

```
dut: dram_sp generic map (
   data_width=>         32,
   memory_depth=> 128,
   num_of_alines=>     7,
   wrten_for_ram=>     1,
   input_registered=>   0,
   clken_for_input=>    0,
   output_registered=> 1,
   clken_for_output=>  1,
   multiplex_out=>       0)
      port map (
   DI=>DI,
   A=>A,
   WR=>WR,
   SPO=>SPO,
   WE=>WE,
   ICE=>ICE,
  OCE=>OCE);
```

Example 1: Generics to instantiate a 128x32 single port RAM with output registers, output register clock enables, and LUT-based output multiplexing schem.

Performance and CLB Utilization Data

Depth	Width	Port Type	Mux_Type	Slice Count	Virtex -6 MHz
64	4	Single	tbuf	32	134
64	4	Single	lut4	22	145
64	4	Dual	tbuf		114
64	4	Dual	lut4		114
64	8	Single	tbuf	64	119
64	8	Single	lut4	44	147
64	8	Dual	tbuf		105
64	8	Dual	lut4		105
64	12	Single	tbuf	96	112
64	12	Single	lut4	66	140
64	12	Dual	tbuf		93
64	12	Dual	lut4		102
64	16	Single	tbuf	128	109
64	16	Single	lut4	88	128
64	16	Dual	tbuf		85
64	16	Dual	lut4		95
64	20	Single	tbuf	160	106
64	20	Single	lut4	110	130
64	20	Dual	tbuf		77
64	20	Dual	lut4		99
64	24	Single	tbuf	192	102
64	24	Single	lut4	132	125
64	24	Dual	tbuf		76
64	24	Dual	lut4		87
64	28	Single	tbuf	224	106
64	28	Single	lut4	154	131
64	28	Dual	tbuf		77
64	28	Dual	lut4		88
64	32	Single	tbuf	256	108
64	32	Single	lut4	176	129
64	32	Dual	tbuf		87
64	32	Dual	lut4		70
64	36	Single	tbuf	288	102
64	36	Single	lut4	198	116
64	36	Dual	tbuf		70
64	36	Dual	lut4		82
64	40	Single	tbuf	320	101
64	40	Single	lut4	220	120
64	40	Dual	tbuf		74
64	40	Dual	lut4		82

Depth	Width	Port Type	Mux Type	Slice Count	Virtex -6 MHz
96	4	Single	tbuf	60	114
96	4	Single	lut4	40	134
96	4	Dual	tbuf		101
96	4	Dual	lut4		104
96	8	Single	tbuf	120	112
96	8	Single	lut4	80	129
96	8	Dual	tbuf		96
96	8	Dual	lut4		90
96	12	Single	tbuf	180	109
96	12	Single	lut4	120	118
96	12	Dual	tbuf		83
96	12	Dual	lut4		83
96	16	Single	tbuf	240	99
96	16	Single	lut4	160	123
96	16	Dual	tbuf		67
96	16	Dual	lut4		89
96	20	Single	tbuf	300	91
96	20	Single	lut4	200	123
96	20	Dual	tbuf		86
96	20	Dual	lut4		88
96	24	Single	tbuf	360	96
96	24	Single	lut4	240	112
96	24	Dual	tbuf		84
96	24	Dual	lut4		83
96	28	Single	tbuf	420	99
96	28	Single	lut4	280	116
96	28	Dual	tbuf		87
96	28	Dual	lut4		57
96	32	Single	tbuf	480	96
96	32	Single	lut4	320	96
96	32	Dual	tbuf		68
96	32	Dual	lut4		73
96	36	Single	tbuf	540	95
96	36	Single	lut4	360	109
96	36	Dual	tbuf		66
96	36	Dual	lut4		78
96	40	Single	tbuf	600	96
96	40	Single	lut4	400	108
96	40	Dual	tbuf		67
96	40	Dual	lut4		78

Depth	Width	Port Type	Mux_Type	Slice Count	Virtex -6 MHz
128	4	Single	tbuf	80	122
128	4	Single	lut4	52	132
128	4	Dual	tbuf		95
128	4	Dual	lut4		104
128	8	Single	tbuf	160	109
128	8	Single	lut4	104	133
128	8	Dual	tbuf		90
128	8	Dual	lut4		96
128	12	Single	tbuf	240	99
128	12	Single	lut4	156	120
128	12	Dual	tbuf		76
128	12	Dual	lut4		83
128	16	Single	tbuf	320	97
128	16	Single	lut4	208	118
128	16	Dual	tbuf		78
128	16	Dual	lut4		88
128	20	Single	tbuf	400	101
128	20	Single	lut4	260	117
128	20	Dual	tbuf		76
128	20	Dual	lut4		84
128	24	Single	tbuf	480	96
128	24	Single	lut4	312	111
128	24	Dual	tbuf		71
128	24	Dual	lut4		79
128	28	Single	tbuf	560	89
128	28	Single	lut4	364	92
128	28	Dual	tbuf		61
128	28	Dual	lut4		77
128	32	Single	tbuf	640	88
128	32	Single	lut4	416	104
128	32	Dual	tbuf		78
128	32	Dual	lut4		79
128	36	Single	tbuf	720	94
128	36	Single	lut4	468	109
128	36	Dual	tbuf		63
128	36	Dual	lut4		79
128	40	Single	tbuf	800	93
128	40	Single	lut4	520	106
128	40	Dual	tbuf		
128	40	Dual	lut4		80

Depth	Width	Port Type	Mux_Type	Slice Count	Virtex -6 MHz
160	4	Single	tbuf	100	107
160	4	Single	lut4	66	125
160	4	Dual	tbuf		97
160	4	Dual	lut4		92
160	8	Single	tbuf	200	89
160	8	Single	lut4	132	117
160	8	Dual	tbuf		87
160	8	Dual	lut4		86
160	12	Single	tbuf	300	88
160	12	Single	lut4	198	115
160	12	Dual	tbuf		81
160	12	Dual	lut4		68
160	16	Single	tbuf	400	98
160	16	Single	lut4	264	108
160	16	Dual	tbuf		69
160	16	Dual	lut4		80
160	20	Single	tbuf	500	93
160	20	Single	lut4	330	110
160	20	Dual	tbuf		71
160	20	Dual	lut4		77
160	24	Single	tbuf	600	90
160	24	Single	lut4	396	113
160	24	Dual	tbuf		60
160	24	Dual	lut4		74
160	28	Single	tbuf	700	93
160	28	Single	lut4	462	103
160	28	Dual	tbuf		75
160	28	Dual	lut4		70
160	32	Single	tbuf	800	88
160	32	Single	lut4	528	101
160	32	Dual	tbuf		60
160	32	Dual	lut4		75
160	36	Single	tbuf	900	92
160	36	Single	lut4	594	103
160	36	Dual	tbuf		65
160	36	Dual	lut4		71
160	40	Single	tbuf	1000	87
160	40	Single	lut4	660	103
160	40	Dual	tbuf		59
160	40	Dual	lut4		67

Depth	Width	Port Type	Mux_Type	Slice Count	Virtex -6 MHz
192	4	Single	tbuf	120	103
192	4	Single	lut4	80	126
192	4	Dual	tbuf		88
192	4	Dual	lut4		93
192	8	Single	tbuf	240	102
192	8	Single	lut4	160	124
192	8	Dual	tbuf		70
192	8	Dual	lut4		77
192	12	Single	tbuf	360	99
192	12	Single	lut4	240	109
192	12	Dual	tbuf		82
192	12	Dual	lut4		75
192	16	Single	tbuf	480	97
192	16	Single	lut4	320	110
192	16	Dual	tbuf		65
192	16	Dual	lut4		63
192	20	Single	tbuf	600	90
192	20	Single	lut4	400	104
192	20	Dual	tbuf		56
192	20	Dual	lut4		73
192	24	Single	tbuf	720	92
192	24	Single	lut4	480	105
192	24	Dual	tbuf		71
192	24	Dual	lut4		75
192	28	Single	tbuf	840	87
192	28	Single	lut4	560	104
192	28	Dual	tbuf		72
192	28	Dual	lut4		71
192	32	Single	tbuf	960	90
192	32	Single	lut4	640	101
192	32	Dual	tbuf		57
192	32	Dual	lut4		71
192	36	Single	tbuf	1080	87
192	36	Single	lut4	720	97
192	36	Dual	tbuf		58
192	36	Dual	lut4		72
192	40	Single	tbuf	1200	87
192	40	Single	lut4	800	95
192	40	Dual	tbuf		65
192	40	Dual	lut4		68

Depth	Width	Port Type	Mux_Type	Slice Count	Virtex_-6 MHz
224	4	Single	tbuf	140	113
224	4	Single	lut4	94	115
224	4	Dual	tbuf		80
224	4	Dual	lut4		79
224	8	Single	tbuf	280	103
224	8	Single	lut4	188	117
224	8	Dual	tbuf		61
224	8	Dual	lut4		79
224	12	Single	tbuf	420	98
224	12	Single	lut4	282	114
224	12	Dual	tbuf		70
224	12	Dual	lut4		72
224	16	Single	tbuf	560	94
224	16	Single	lut4	376	114
224	16	Dual	tbuf		72
224	16	Dual	lut4		72
224	20	Single	tbuf	700	90
224	20	Single	lut4	470	105
224	20	Dual	tbuf		70
224	20	Dual	lut4		75
224	24	Single	tbuf	840	88
224	24	Single	lut4	564	101
224	24	Dual	tbuf		70
224	24	Dual	lut4		71
224	28	Single	tbuf	980	86
224	28	Single	lut4	658	92
224	28	Dual	tbuf		66
224	28	Dual	lut4		63
224	32	Single	tbuf	1120	88
224	32	Single	lut4	752	94
224	32	Dual	tbuf		55
224	32	Dual	lut4		66
224	36	Single	tbuf	1260	86
224	36	Single	lut4	846	94
224	36	Dual	tbuf		58
224	36	Dual	lut4		72
224	40	Single	tbuf	1400	86
224	40	Single	lut4	940	96
224	40	Dual	tbuf		59
224	40	Dual	lut4		67

Depth	Width	Port Type	Mux_Type	Slice Count	Virtex_-6 MHz

256	4	Single	tbuf	160	106
256	4	Single	lut4	104	127
256	4	Dual	tbuf		91
256	4	Dual	lut4		83
256	8	Single	tbuf	320	101
256	8	Single	lut4	208	115
256	8	Dual	tbuf		66
256	8	Dual	lut4		82
256	12	Single	tbuf	480	95
256	12	Single	lut4	312	106
256	12	Dual	tbuf		60
256	12	Dual	lut4		75
256	16	Single	tbuf	640	92
256	16	Single	lut4	416	104
256	16	Dual	tbuf		69
256	16	Dual	lut4		72
256	20	Single	tbuf	800	84
256	20	Single	lut4	520	99
256	20	Dual	tbuf		70
256	20	Dual	lut4		72
256	24	Single	tbuf	960	86
256	24	Single	lut4	624	98
256	24	Dual	tbuf		72
256	24	Dual	lut4		74
256	28	Single	tbuf	1120	89
256	28	Single	lut4	728	96
256	28	Dual	tbuf		67
256	28	Dual	lut4		61
256	32	Single	tbuf	1280	88
256	32	Single	lut4	832	94
256	32	Dual	tbuf		63
256	32	Dual	lut4		67
256	36	Single	tbuf	1440	84
256	36	Single	lut4	936	97
256	36	Dual	tbuf		61
256	36	Dual	lut4		74
256	40	Single	tbuf	1600	84
256	40	Dual	tbuf		56

Xilinx Reference Design License

By using the accompanying Xilinx, Inc. Reference Designs (the “Designs”), you agree to the following terms and conditions. You may use the Designs solely in support of your use in developing designs for Xilinx programmable logic devices or Xilinx HardWire™ devices. Access to the Designs is provided only to purchasers of Xilinx programmable logic devices or Xilinx HardWire™ devices for the purposes set forth herein.

The Designs are provided by Xilinx solely for your reference, for use as-is or as a template to make your own working designs. The Designs may be incomplete, and Xilinx does not warrant that the Designs are completed, tested, or will work on their own without revisions. The success of any designs you complete using the Designs as a starting point is wholly dependent on your design efforts. As provided, Xilinx does not warrant that the Designs will provide any given functionality, and all verification must be completed by the customer.

Xilinx specifically disclaims any obligations for technical support and bug fixes, as well as any liability with respect to the Designs, and no contractual obligations are formed either directly or indirectly by use of the Designs. XILINX SHALL NOT BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION DIRECT, INDIRECT, INCIDENTAL, SPECIAL, RELIANCE OR CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF THE DESIGNS, EVEN IF XILINX HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Xilinx makes no representation that the Designs will provide the functionality you are looking for, or that they are appropriate for any given use. Xilinx does not warrant that the Designs are error-free, nor does Xilinx make any other representations or warranties, whether express or implied, including without limitation implied warranties of merchantability or fitness for a particular purpose. The Designs are not covered by any other license or agreement you may have with Xilinx.

The Designs are the copyrighted, confidential and proprietary information of Xilinx. You may not disclose, reproduce, transmit or otherwise copy the Designs by any means for any purpose not set forth in this license, without the prior written permission of Xilinx.

You agree that you will comply with all applicable governmental export rules and regulations, and that you will not export or reexport the Designs in any form without the appropriate government licenses.

XILINX, INC., 2100 Logic Drive, San Jose, California 95124

This document is (c) Xilinx, Inc. 1998, 1999. No part of this file may be modified, transmitted to any third party (other than as intended by

```
Xilinx) or used without a Xilinx programmable or hardwire device without
Xilinx's prior written permission.
```