Efficient Multi-Channel SERIAL to PARALLEL Converter

A significant size reduction is accomplished by taking advantage of the dual port mode of the distributed SelectRAM available in Spartan,™ XC4000, and Virtex devices.

by Paul Gigliotti, Field Applications Engineer, Xilinx, paul.gigliotti@xilinx.com

Often, data from remote sensors is sent back to a central location in serial format, to reduce the cost of cabling. The data streams then need to be converted into a parallel format for processing. This is typically accomplished using a shift register, followed by a holding register, thus requiring two flip-flops per bit.

For example, eight 16-bit channels of data would require eight 16-bit shift registers and eight 16-bit registers, for a total of 256 flip-flops. This requires 128 CLBs in a Spartan or XC4000 family device, or 64 CLBs in a Virtex device. The following implementation requires only 30.5 CLBs in a Spartan or XC4000, and only 15.25 CLBs in a Virtex device, and requires 128 (8x16) clock cycles for one complete scan.

The ability of Spartan, XC4000, and Virtex FPGAs to support distributed blocks of RAM allows for significant design efficiencies.

Double Buffering for "free"

The dual port RAM is used to both convert the data from serial to parallel format, as well as provide for double buffering. The data is written into one half of the memory, and read out from the other half. The double buffering is accomplished by "ping-ponging" between the memory halves.

Memory Map

The 16 dual port 16x1 RAMs are structured such that each DPRAM contains a specific bit of data from all channels. For example, DPRAM 0 contains all the bit zeros, DPRAM 1 contains all the bit ones, and so on. As discussed above, the upper/lower half contains the data currently being converted, while the lower/upper half contains the data previously converted.

Double Buffering

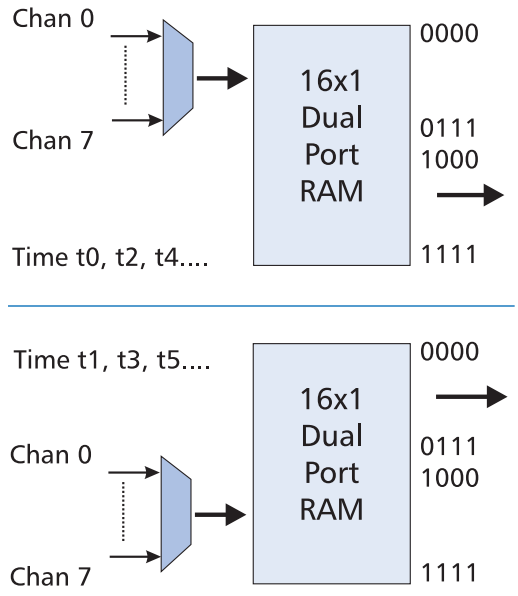

Figure 1

Memory Map

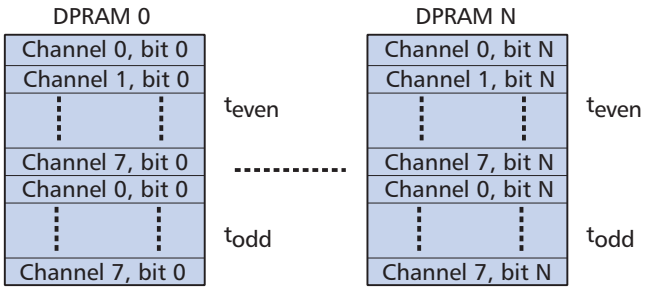

Figure 2

Control Logic

The control logic consists of an eight-bit counter, an 8-to-1 multiplexer, and a 4-to-16 decoder. In a Spartan or XC4000 device, the control logic uses four CLBs for the counter, 2.5 CLBs for the multiplexer, and eight CLBs for the decoder, for a total overhead of 14.5 CLBs. The three LSBs of the eight-bit counter drive the select lines of the 8-to-1 mux, as well as the write port's three LSB address lines. The MSB of the write port is driven by the inversion of the counter's MSB. The four MSBs of the counter drive the read port's addresses. Finally, bits 3, 4, 5, and 6 of the counter source the 4-to-16 decoder, with the outputs of the decoder driving the write enables of the dual port RAMs.

Conclusion

The ability of Spartan, XC4000, and Virtex FPGAs to support distributed blocks of RAM allows for significant design efficiencies. The above design can be extended to 16 channels by adding another bank of sixteen dual port RAMs, increasing the CLB count by 16, rather than an increase of 128 CLBs using a more traditional approach.

The Code

```
entity ser2par is
port (
CLK: in STD_LOGIC;
CHANNEL: in STD_LOGIC_VECTOR (7 downto 0);
DATA: out STD_LOGIC_VECTOR (15 downto 0)
);
end ser2par;

architecture ser2par_arch of ser2par is

signal DATA_BIT:STD_LOGIC;
signal WRITE_ADDRESS:STD_LOGIC_VECTOR(3 downto 0);
signal COUNTER:STD_LOGIC_VECTOR(7 downto 0);
signal WRITE_ENABLE:STD_LOGIC_VECTOR(15 downto 0);

component RAM16X1D
PORT(
A: IN std_logic_vector(3 DOWNTO 0);
DI: IN std_logic;
WR_EN: IN std_logic;
WR_CLK: IN std_logic;
DPO: OUT std_logic;
DPRA: IN std_logic_vector(3 DOWNTO 0));
end component;

begin

process (CLK)
begin
 if CLK='1' and CLK'event then
   COUNTER <= COUNTER + 1;
 end if;
end process;

DECODER:
process (COUNTER)
begin
 for I in 0 to 15 loop
   if (COUNTER(7 downto 4) = I) then
       WRITE_ENABLE(I) <= '1';
   else
       WRITE_ENABLE(I) <= '0';
   end if;
 end loop;
end process;

MUX:
process (COUNTER)
begin
 case COUNTER(2 downto 0) is
   when "000" => DATA_BIT <= CHANNEL(0);
   when "001" => DATA_BIT <= CHANNEL(1);
   when "010" => DATA_BIT <= CHANNEL(2);
   when "011" => DATA_BIT <= CHANNEL(3);
   when "100" => DATA_BIT <= CHANNEL(4);
   when "101" => DATA_BIT <= CHANNEL(5);

   when "110" => DATA_BIT <= CHANNEL(6);
   when others => DATA_BIT <= CHANNEL(7);
 end case;
end process;

WRITE_ADDRESS <= (not(COUNTER(7)) & COUNTER);

RAM:
for I in 0 to 15 generate
RAMBANK:
 RAM16X1D port map (
   A => WRITE_ADDRESS,
   DI => DATA_BIT,
   WR_EN => WRITE_ENABLE(I),
   WR_CLK => CLK,
   DPO => DATA(I),
   DPRA => COUNTER(7 downto 4)
   );
end generate RAM;
end ser2par_arch;
```