BRAM Parameterized

Verilog code:

```
module bram #(
parameter  data_width =32,
parameter addr_width=10,
parameter output_reg=0
)(
input clk,
input we,
input [addr_width-1:0] addr,
input [data_width-1:0] din,
output [data_width-1:0] dout
);

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

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

Schematics:

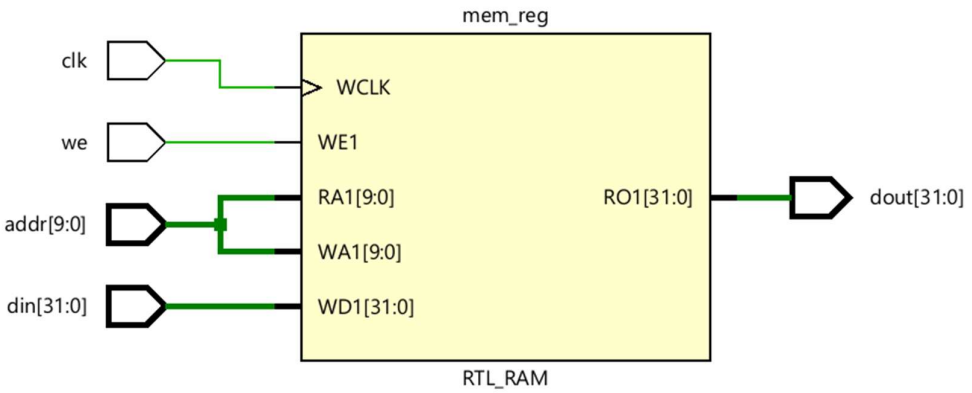

Figure 1: RTL Schematic

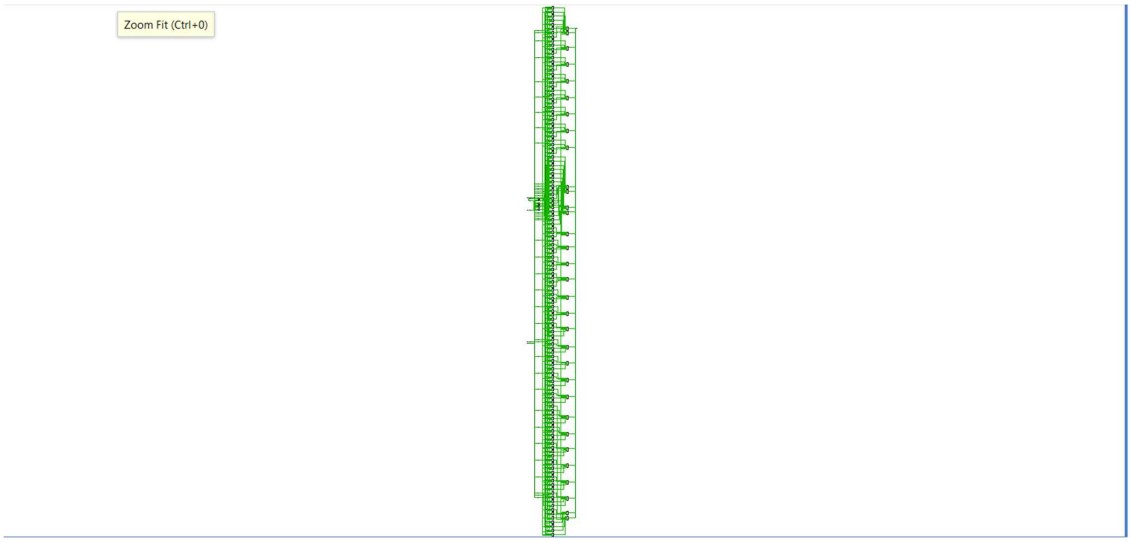

Figure 2: Synthesized schematic

Slice logic table:

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	548	0	0	53200	1.03
LUT as Logic	36	0	0	53200	0.07
LUT as Memory	512	0	0	17400	2.94
LUT as Distributed RAM	512	0			
LUT as Shift Register	0	0			
Slice Registers	0	0	0	106400	0.00
Register as Flip Flop	0	0	0	106400	0.00
Register as Latch	0	0	0	106400	0.00
F7 Muxes	256	0	0	26600	0.96
F8 Muxes	128	0	0	13300	0.96

Figure 3: Slice logic table

Question: Change the design to 50Kb (any depth and width) and check what resources it's consuming?

Ans: To change the design to 50Kb, also we have 32 bit i/o lines. Therefore 50000/32 = 1562.5 bit of data ~ 1600.

Address line width $\log_2(1600) = 10.643 \sim 11$.

Verilog code updated:

```
module bram #(
parameter  data_width =32,
parameter addr_width=11,
parameter output_reg=0
)(
input clk,
input we,
input [addr_width-1:0] addr,
input [data_width-1:0] din,
output [data_width-1:0] dout
);

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

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

Schematics:

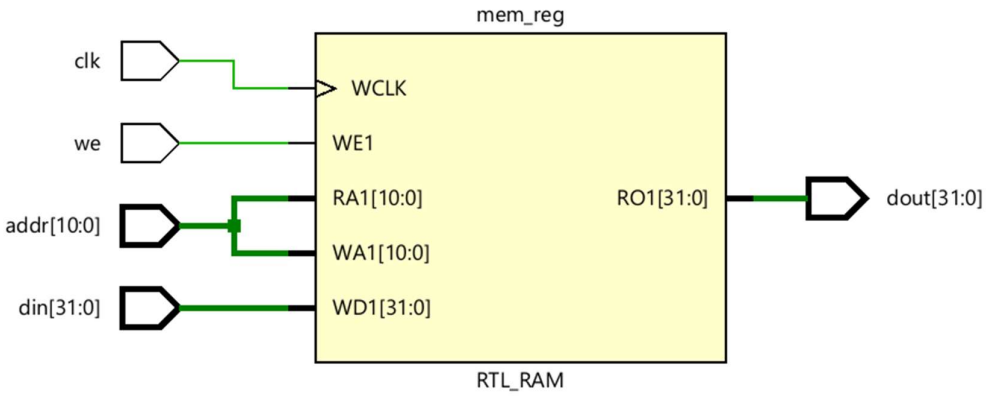

Figure 4: updated for 50Kb

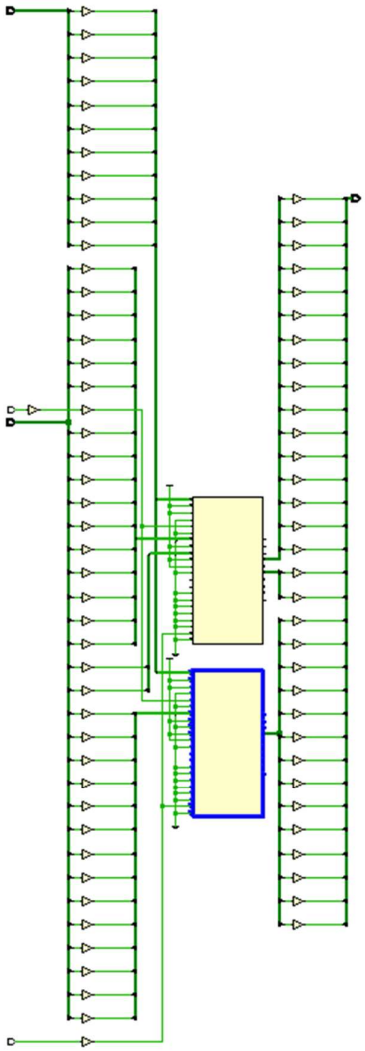

Figure 5: updated for 50Kb

In FPGA for large data it uses BRAM instead LUT's, even considering the fact it LUT is faster than BRAM.

This design uses RAMB36E1. **RAMB36E1** is a 36 Kb block RAM primitive that can be cascaded for larger memories. It supports various configurations, from 1-bit × 32K to 36-bit × 1K true dual-port RAM, and up to 72-bit × 512 simple dual-port RAM. Read and write ports are fully synchronous but can operate independently. Features include byte-enable writes, optional output registers for faster timing, and built-in error detection and correction.

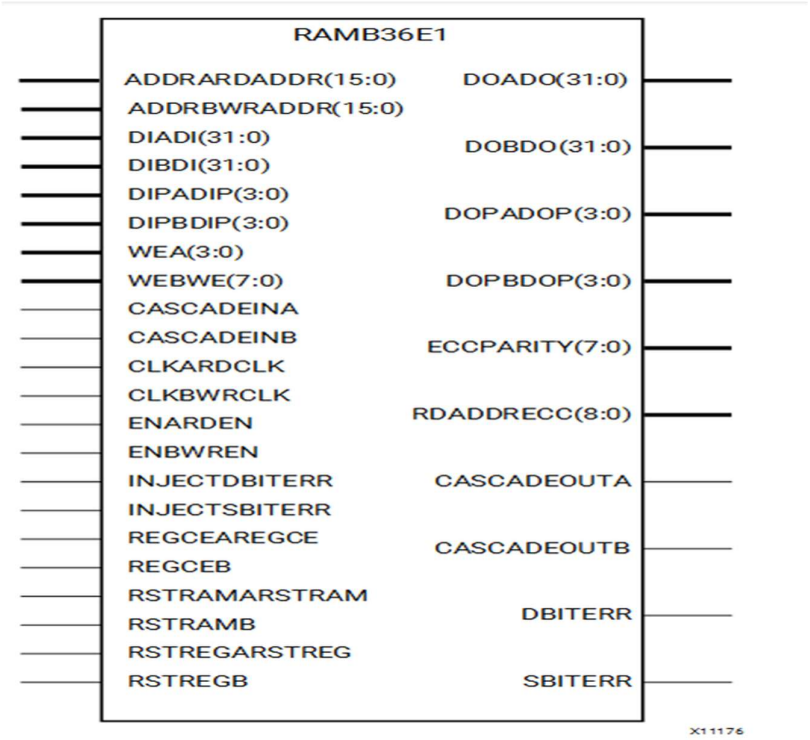

Figure 6: Block diagram for RAMB36E1