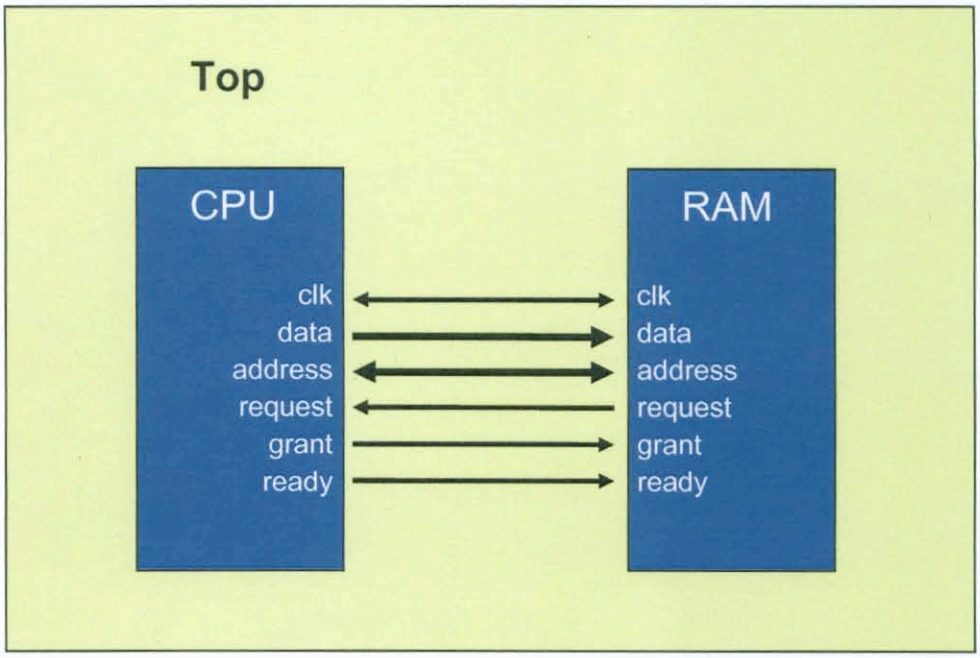
Figure 8 CPU and RAM Interconnection Using Ports



The Verilog code for the design illustrated in Figure 8 is shown in Figure 9.

Figure 9 Verilog Code for CPU - RAM

```
module CPU (clk, data, address, request, grant, ready);
inout [47:0] address;
input request, clk;
output [63:0] data;
output grant, ready
...
endmodule


module RAM (clk, data, address, request, grant, ready);
inout [47:0] address;
input clk, grant, ready;
input [63:0] data;
output request;
...
endmodule


module Top;
wire clk;
wire request, grant, ready;
wire [47:0] address;
wire [63:0] data;

CPU CPU(clk, data, address, request, grant, ready);
RAM RAM(clk, data, address, request, grant, ready);
endmodule
```

The SystemVerilog `interface` construct allows a high level of design abstraction. Unidirectional and bidirectional signals, functions, and gates can be bundled into one unit. As shown in Figure 10 and Figure 11, you can define an interface called `chip_bus` to bundle all of the signals between CPU and RAM.

Figure 10 CPU and RAM Interconnection Using Interface chip_bus

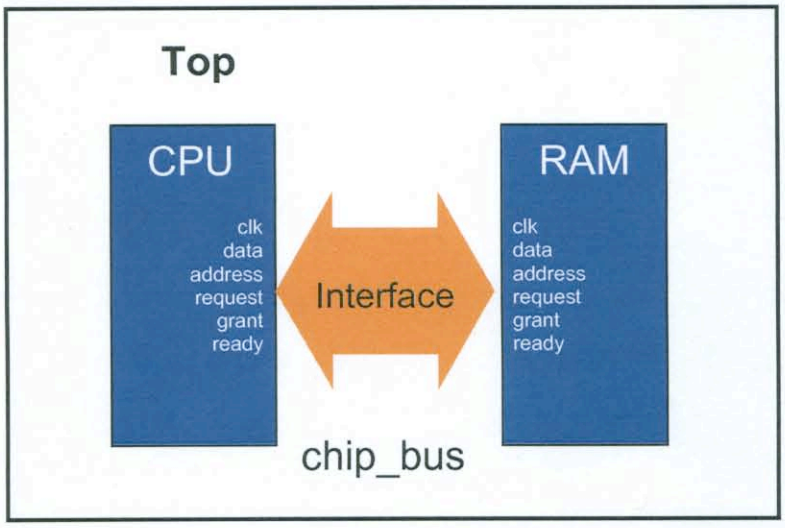


Figure 11 SystemVerilog code for CPU - RAM

```
interface chip_bus ( input wire clk );
logic       request, grant, ready;
wire [47:0] address;
wire [63:0] data;

modport cpu (input request, output grant output ready);
modport ram (output request, input grant, input ready);

endinterface

module CPU (chip_bus.cpu io);
assign cpu.grant  = . . .
endmodule

module RAM (chip_bus.ram pins);
assign pins.request = . . .
endmodule

module Top;
wire clk;
chip_bus a(clk);
CPU CPU(a.cpu);
RAM RAM(a.ram);
endmodule
```