VHDL Models of Processor Intel Pentium

Valentina S. Kukenska [1], Ivan S. Simeonov [2]

Abstract – **The present papers discuss the opportunity for developing of the models based on most popular Intel processor through behavioral hardware description.**

The paper describes are structural and behavioral VHDL models of the processor Intel Pentium.

Keywords – **VHDL, model, Intel Pentium processor, behavior model**

1. INTRODUCTION

The contemporary electronic systems are complex ones, built up of a lot of components. Within the automated design of such systems methods aiming at reducing both the designing time and the problem's complexity are under search and application. A method of these is concerns the principles of decomposition and hierarchy.

Decomposition is spreading as a common technique for reducing the complexity of all tasks and problems in elaborating the hardware and software parts of the systems.

The dividing of systems is usually carried through in a way, which forms functionally independent units. Digital systems, for instance, are built up of a large number of sub-systems (modules). Each module has a set of ports, which constitute its interface to the outside world. A digital system is usually designed as a hierarchical collection of modules.

The present papers discuss the opportunity for developing of the models based on most popular Intel processor through behavioral hardware description.

The paper describes are structural and behavioral VHDL models of the processor Intel Pentium.

2. ADVANCED VHDL

VHDL is a Hardware Description Language for describing digital electronic system.

VHDL is designed to full a number of needs in the design process. Firstly, it allows description of the structure of a design that is how it is decompressed into sub-designs, and how those sub-designs are interconnected. Secondly, it allows the specification of the function of designs using familiar programming language forms. Thirdly, as a result, it allows a design to be simulated before manufactured, so that designers can quickly compare alternatives and test for correctness without delay and expense of hardware prototyping.

[1]Valentina S. Kukenska is with the Faculty of Electronic Engineering, H. Dimitar 4, 5300 Gabrovo, Bulgaria, E-mail: vally@tugab.bg

[2]Ivan Simeonov Simeonov is with the Faculty of Electronic Engineering, H. Dimitar 4, 5300 Gabrovo, Bulgaria, E-mail: isim@tugab.bg

VHDL contains a number of facilities for modifying the state of objects and controlling the flow of execution of modules.

In VHDL, an entity is such a module which may be used as a component in a design, or which may be the top-level module of the design. The entity declarative part may be used to declare items, which are to be used in the implementation of the entity.

Once an entity has had its interface specified in an entity declaration, one or more implementations of the entity can be described in architecture bodies. Each architecture body can describe a different view of the entity.

The declarations in the architecture body define items that will be used to construct the design description.

Signals are used to connect sub modules in a design. The sub modules in an architecture body can be described as blocks. A block is a unit of module structure, with its own interface, connected to other blocks or ports by signals. A signal assignment schedules one or more transactions to a signal (or port).

The primary unit of behavioral description in VHDL is the process. A process is a sequential body of code, which can be activated in response to changes in state. When more than one process is activated at the same time, they execute concurrently.

A process statement which can be used in an architecture body or block. The declarations define items which can be used locally within the process.

A process may contain a number of signal assignment statements for a given signal, which together form a driver for the signal.

VHDL descriptions write them in a design file. After then invoke a compiler to analyze them and insert them into a design library. A number of VHDL constructs may be separately analyzed for inclusion in a design library. These constructs are called library units. A design file may contain a number of library units.

There are two special libraries which are implicitly available to all design units, and so do not need to be named in a library clause. The first is called work, and refer to the working design library into which the current design units will be placed by the analyzer.

The second special library is called STD, and contains the packages standard. Standard contains all of the predefined types and functions.

3. Architecture of the Pentium Processor

The Pentium is fully compatible with previous Intel processor, but it also different from them in many ways.

The Pentium processor internal architecture is shows on the fig. 1

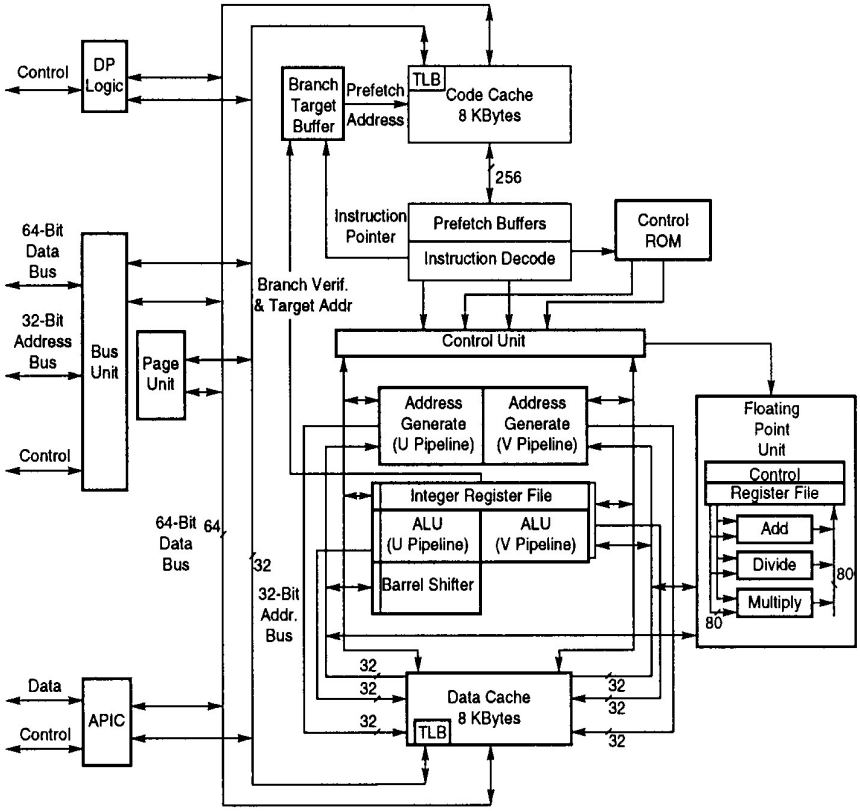

Fig.1 Pentium processor internal architecture

The Pentium features twin data pipelines, which enable it to execute two instructions at a same time. Intel calls the capability to execute two instructions at the same time superscalar technology. This technology provides additional performance compared with the 486.

With superscalar technology, the Pentium can execute many instructions at a rate of two instructions per cycle. Superscalar architecture usually is associated with high-output RISC chips. The Pentium is one of the first CISC chips to be considered superscalar.

The two instruction pipelines within the chip are called the u- and v-pipes. The u-pipe, which is the primary pipe, can execute all integer and floating-point instructions. The v-pipe is a secondary pipe that can execute only simple integer instructions and certain floating –point instructions. The process of operating on two instructions simultaneously in the different pipes is called pairing. Not all sequentially executing instructions can be paired, and when pairing is not possible, only the u-pipe is used. To optimize the Pentium's efficiency, you can recompile software to allow more instructions to be paired.

The Pentium processor has a Branch Target Buffer (BTB), which employs a technique called branch predication. It minimizes stalls in one or more of the pipes caused by delays in fetching instructions that branch to non-linear memory locations. The BTB attempts to predict whether a program branch will be taken, and then fetches the appropriate instructions. The use of branch prediction enables the Pentium to keep both pipelines operating at full speed.

The Pentium has a 32-bit address bus width, given it the same 4GB memory-addressing capabilities. But the Pentium expands the data to 64 bits, which means that it can more twice as much data into or out of the CPU. The 66-bit data bus requires that system memory be accessed 64 bits wide, which means that each bank of memory is 64 bits.

Most Pentium systems use 32-bits wide SIMMs – two of these SIMMs per bank of memory. Most Pentium motherboards have at least four of these 32-bit SIMM sockets, providing for a total of two banks of memory. The newest Pentium systems and most Pentium II systems today use DIMMs, which are 64 bits wide.

The Pentium has only 32-bit internal registers. As instructions are being processed internally, they are broken down into 32-bit instructions and data elements.

The Pentium has two separate internal 8KB caches, compared with a single 8KB or 16KB cache in the 486. The cache-controller circuitry and the cache memory are embedded in the CPU chip. The cache mirrors the information in normal RAM by keeping a copy of the data and code from different memory locations. The Pentium cache also can hold information to be written to memory when the load on the CPU and other system components is less.

The separate code and data caches are organized in a two-way set associative fashion, with each set split into lines of 32

bytes each. Each cache has a dedicated Translation Lookaside Buffer (TIB) that translates linear addresses to physical addresses. You can configure the data cache as write-back or write-through on a line-by-line basis. When you use the write-back capability, the cache can store write operations and reads, further improving performance over read-only write-through mode. Using write-back mode results in less activity between the CPU and system memory. The code cache is an inherently write-protected cache because it contains only execution instructions and not data, which are updated.

Systems based on the Pentium can benefit greatly from secondary processor caches (L2), which usually consist of up to 512KB or more of extremely fast (15ns or less). Static RAM (SRAM) chips. When the CPU fetches data that is not already available in its internal processor (LI) cache, wait states slow the CPU. If the data already is in the secondary processor cache, however, the CPU can go ahead with its work without pausing for wait states.

The Pentium uses a BiCMOS process and superscalar architecture to achieve the high level of performance expected from the chip.

The Pentium contains an internal math coprocessor or FPU. The FPU in the Pentium has been rewritten and performs significantly better than the FPU in the 486. The Pentium FPU is estimated at two to as much as 10 times faster than the FPU in the 486. In addition, the two standard instruction pipelines in the Pentium provide two units to handle standard integer math.

4. VHDL MODELS

In this section a behavioral and structural VHDL models of the Pentium processor be presented. This model can be used to run test programs in the Pentium instruction set by connecting it to a simulated memory model.

The processor instruction set and bus architectures are first described. Then a behavioural description is given.

The package body or the Pentium_types is listed in fig.2

```
package body or the Pentium_types is
  constant bool_to_bit: bool_to_bit_table:=
    (false => '0', true => '1');
  function resolve_bit_32
    (driver: in bit_32_array) return bit_32 is
    constant float_velue: bit_32:=X"0000_0000";
    variable result: bit_32:= float_velue;
  begin
    for i in driver'range loop
        result:=result or driver(i);
    end loop;
    return result;
  end resolve_bit_32;
```

Fig.2. Package body for Pentium_types

The body or the resolution function for 32-bit buses is defined. The function takes as its parameter an unconstrained array of bit_32 values, and produces as a results the bit-wide logical or of the values. The function cannot assume that the length of the array will be greater than one. If no drivers are active on the bus, an empty array will be passed to the resolution fiction. In this case, the defaults value of all '0' (float_value) is used as the result.

The function bits_to_int converts a bit vector representing a twos-compliment signed integer into an integer type value. The local variable temp is declared to be a bit vector of the same size and index range as the parameter bits. The variable result is initialized to zero when the function is invoked, and subsequently used to accumulate the weighted bit values in the loop.

The entity declaration of the Pentium processor is shown in fig.3.

```
use work.Pentium_types all;
entity Pentium is
    generic (Tpd: Time:=unit_delay);
    port (d-bus: inout bus_bit_64 bus;
          a_bus: out bit_32;
          read, write: out bit;
          fetch:  out bit;
          ready: in bit;
          p1, p2: in bit;
          reset: in bit);
end Pentium;
```

Fig.3. Entity declaration for Pentium processor

The architecture body for the behavioral description is:

```
use work.Pentium types.all;
architecture behaviour of .Pentium is
      subtype reg_addr is natural range 0 to 255;
      type reg_array is (reg_addr) array of bit_32;
begin
process
      variable reg: reg_array;
      variable PC: bit_32;
      variable current_instr: bit_32;
      variable op: bit_8;
      variable r3, r1, r2: reg_addr;
      variable i8: integer;
      alias cm_i: bit is current_instr(19);
      alias cm_V: bit is current_instr(18);
      alias cm_N: bit is current_instr(17);
      alias cm_Z: bit is current_instr(16);
      variable cc_V, cc_N, cc_Z: bit;
      variable temp_V, temp_N, temp_Z: bit;
      variable displacsement, effective_addr: bit_32;
procedure memory_read (addr: in bit_32;
            fetch_cycle: in Boolean; result: out bit_32) is
end memory_read;
procedure memory_write(addr: in bit_32;data: in bit_32) is
end memory_ write;
procedure add (result : inout  bit_32;
                  op1, op2: in integer;  V, N, Z : out bit) is
end add;
procedure multiply (result : inout  bit_32;
      op1, op2: in integer; V, N, Z : out bit) is
end multiply;
end process;
end behavior;
```

Fig.4. Behavioral description for Pentium processor

The declaration section for the architecture body contains the declaration for the DP32 register file type, and array of 32-bit words, indexed by a natural number constrained to be in the range 0 to 255.

The architecture body contains only one concurrent statement, namely an anonymous process which implements the behaviour as a sequential algorithm. This process declares a number of variables, which represent the internal state of the processor: the register file, the program counter (PC), and the current instruction register. A number of working variables and aliases are also declared.

The procedure memory read implements the behavioural model of a memory read transaction. The parameters are the memory address to read from, a flag indicating whether the read is an instruction fetch, and a result parameter returning the data read. The procedure refers to the entity port, which are visible because they are declared in the parent of the procedure.

The memory read model firstly drives the address and fetch bit ports, and then waits until the next leading edge of p1, indicating the start of the next clock cycle. (The wait statement is sensitive to a change from '0' to '1' on p1). When that event occurs, the model checks the state of the reset input port, and if it is set, immediately returns without further action. If reset is clear, the model starts a T1 state by asserting the read bit port a propagation delay time after the clock edge. It then waits again until the next p1 leading edge, indicating the start of the next clock cycle. Again, it checks reset and discontinues if reset is set. The model then starts a loop executing T2 states. It waits until p2 changes from '0' to '1' (at the end of cycle), and then checks reset again, returning if it is set. Otherwise it checks the ready bit input port, and if set, accepts the data from the data bus port and exit the loop. If ready is not set, the loop repeats, adding another T2 state to the transaction. After the loop, the model waits for the next clock edge indicating the start of the Ti state at the end of the transaction. After checking reset again, the model clears ready to complete the transaction, and returns to the parent process.

The procedure memory write is similar, implementing the model for a memory write transaction. The parameters are simply the memory address to write to, and the data to write. The model similarly has reset checks after each wait point. One difference is that at the end of the transaction, there is a null signal assignment to the data bus port. This models the behaviour of the processor disconnecting from the data bus, that is, at the point it stops driving the port.

The entity declaration and behaviour architecture of the memory module are show in this section.

```
entity memory is
        genetic (Tpd : Time := unit_delay);
        port (d_bus : inout bus_bit_32 bus;
                a_bus : in bit_32;
                read, write : in bit;
                ready : out bit);
end memory;

architecture behaviour of memory is
begin
  process
        constant low_address : integer :=0;
        constant high_address : integer :=65535;
        type memory_aray is
           array (integer range low_address to high_addres)
                                                of bit_32;
        variable mem : memory_array;
        variable address : integer;
  begin
        d_bus <= null after Tpd;
         ready <= '0' after Tdp;
        wait until (read = '1' ) or (write = '1' );
                        address := bits_to_int (a_bus);
                   if address >= low_address and address <=
                                        high_address then
        if write = '1' then
                ready <= '1' after Tpd;
                wait until write = '0';
                mem (address) := d_bus'delayed(Tpd);
        else  -- read = '1'
                d_bus <= mem (address) after Tpd;
                        ready <=  '1' after Tpd;
                wait until read = '0';
        end if;
    end if;
  end process;
end behaviour.
```

Fig.5. Structural and bbehavioural models for memory module

5. CONCLUSION

In this paper a structural and behavioral models of the processor Intel Pentium will be presented. Descriptions will be given for each of the sub-modules in this architecture (fig.1). They will be used in a structural architecture body of the processor entity.

The declaration section for the architecture body contains the declaration for the Intel processor registry file type and array of 32-bit words.

The architecture body contains only one concurrent statement, namely an anonymous process which implements the behavior as a sequential algorithm. This process declares a number of variables, which represent the internal state of the processor: the register file, the program counter, and the current instruction register. A number of working variables and aliases are also declared.

The present model can use to run test programs in the Intel Pentium processor instruction set by connecting it to a simulated memory model.

REFERENCES

[1] "VHDL Reference Manual", SYNOPSYS, inc., 1997.

[2] "Behavioral Compiler Methodology Guide", SYNOPSYS, inc., 1997.

[3] Mueller S., "Upgrading and Repairing PCs", QUE Corporation, Copyright, 2002.

[4] "Pentium", NiSoft Ltd., 1998.

[5] J.R. Armstrong,"Chip-Level Modeling with VHDL", Prentice-Hall, Inc., 1989.