Buletinul Ştiinţific al Universităţii "Politehnica" din Timişoara

Seria ELECTRONICĂ şi TELECOMUNICAŢII
TRANSACTIONS on ELECTRONICS and COMMUNICATIONS

Tom 47(61), Fascicola 1-2, 2002

VHDL DESCRIPTION OF PIC14000 MICROCONTROLLER WITH PIPELINE

Ciprian Gavrincea[1], Daniel Mic[2], Stefan Oniga[3], Alin Tisan[4]

Abstract - This work presents a method for VHDL description of a microcontroller with a pipelined datapath. The microcontroller described in this paper follows the specification of PIC14000 microcontroller made by Microchip Technology Inc. The project was designed using Xilinx ISE 4 software and implemented in XC4010XL FPGA circuit.
Keywords: VHDL, PIC14000, pipeline, data hazard, control hazard

I INTRODUCTION

Embedded control products are found in all market segments: consumer, commercial, PC peripherals, telecommunications, automotive, industrial etc. Most often embedded control products must meet special requirements: cost-effectiveness, low power, speed, small footprint and a high level of system integration.

II PIPELINED DATAPATH

Most embedded control systems are designed around a microcontroller that integrates on-chip program memory, data memory and various peripheral functions, such as timers and serial communication. One way of improving the performance of an embedded control system is to improve the performance of its microcontroller. Speed is one of the most important features of a microcontroller. A solution for improving the speed of a microcontroller is to use a pipelined datapath.
A microcontroller depends on an external clock signal to perform its function. The basic clock period is called the clock cycle. The microcontroller machine cycle is made up of several clock cycles. A machine cycle is used to perform internal machine operations or to accomplish external data transfer to memory or I/O ports.

1)Teaching Assistant eng. 2) Teaching Assistant eng.
3)Lecturer eng. 4) Teaching Assistant eng.
Electrotechincal Departament, North University of Baia Mare
Phone +40-95 583324, e-mail: gcg@ubm.ro

An instruction cycle consists of one or more machine cycles. The first part of the cycle is referred to as the fetch phase, during which an instruction is obtained from program memory. The second part of the cycle is called the execution phase, in which the control unit generates the necessary step sequence to perform the required steps. The execution phase can be break up into three parts. First the instruction is decoded and the required operands are been read from the memory or an I/O device. Second the required operation is executed by ALU. Third the result is writing back to memory or to an I/O device.
A conventional microcontroller process the instruction cycle sequentially. This results in an idle ALU during the fetch and a part of the execution phase, and an idle memory during the second part of the execution phase. A microcontroller with a pipelined architecture eliminates the inefficiency of an idle ALU by fetching the next instruction while the current one is executed. To achieve this operation it is necessary to insert a data storage registers between the memory and ALU.
The PIC14000 microcontroller has basic pipeline architecture. We have to make some modification into its architecture to obtain a PIC14000 microcontroller with full pipeline architecture.

Fig.1 presents the simulation results of PIC14000 microcontroller with a typical datapath.

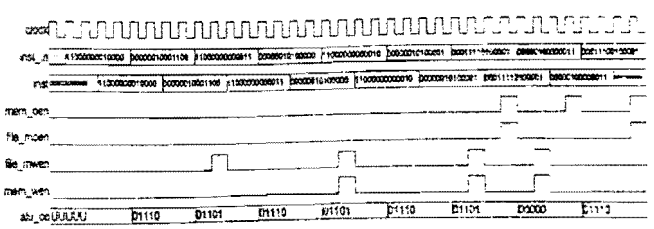


Fig. 1 PIC14000 microcontroller with a typical datapath

An instruction cycle consists of four clock cycles. The instruction fetch (IF) and execute (EX) are pipelined such the fetch takes one instruction cycle while execute takes another instruction cycle. In the

execution cycle, the instruction is decoded on Q1, data memory is read on Q2 and written on Q4, the instruction is executed on Q3.
Fig. 2 presents the simulation results of PIC14000 microcontroller with a pipelined datapath.

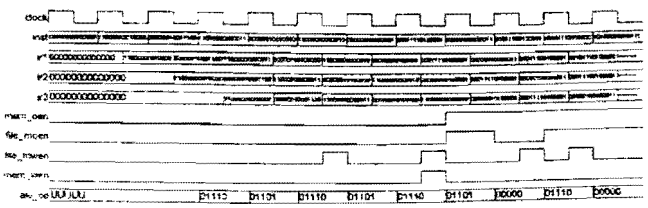


Fig. 2 PIC14000 microcontroller with a pipelined datapath.

It will take four clock cycles for an instruction to be fetched, decoded and executed (2 clock cycles), but on every clock cycle an instruction is fetched, another instruction is decoded and another instruction is executed. The microcontroller with pipelined datapath is almost four times faster then the microcontroller with typical datapath. But in reality is only 3.4 times faster, because it takes some time for filling up the pipeline when the process is started and each time a jump or a conditional branch is executed.

III DATA AND CONTROL HAZARD

Two of the problems that reduce the throughput of a pipelined datapath are: data hazard and control hazard. Data hazards are timing problems that arise because the execution of an operation in a pipelined is delayed by one or more clock cycles from the time at which the instruction containing the operation was fetched. If another instruction tries to use the result of the operation as an operand before the result is available, it uses the old or stale value, which is very likely to give a wrong result. The next example illustrates two data hazards.

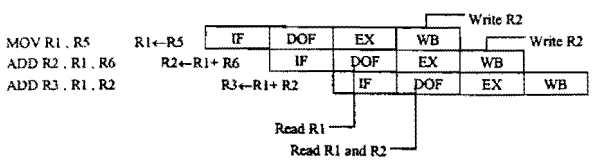


Fig 3 Data hazard

The MOV instruction places the contents of R5 into R1 in the first half of WB(write back), cycle 4. But, as shown by the arrow, the first ADD instruction reads R1 in the last half of DOF(decode and operand fetch), cycle 3, before it is written with the new result. Thus the ADD instruction uses the stale value in R1. The result of this operation is placed in R2 in the first half of WB, cycle 5. The second ADD instruction reads both R1 and R2. In the case of R1 the value read was written in the first half of WB, cycle 4, so it is the write value. But the value of R2 is a stale value. In each of these cases the read of the involved registers occurs one clock cycle too soon with respect to the write of that register. The hardware solution for this kind of problem is to delay the pipeline with one clock cycle when the data hazard condition is detected

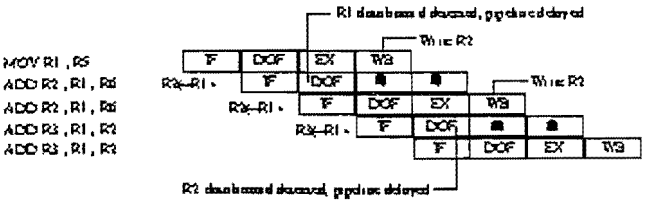


Fig 4 Data hazard solution

Control hazards are associated with branches in the control flow of the program. The following example illustrates a control hazard. If R1 is zero then a branch to the instruction 20 will occurred, skipping the instructions 2 and 3. If R1 is non-zero, then the instruction 2 and 3 will be executed. Assume that the branch is taken to location 20 because R1 is equal to zero. The fact that R1 equals zero is not detected until EX in cycle 3 of the instruction. PC is set to 20 on the next clock edge, at the end of cycle 3. The MOV instructions from locations 2 and 3 are into the EX and DOF stages. Even though the programmer's intention was to skip them these instructions will complete execution.

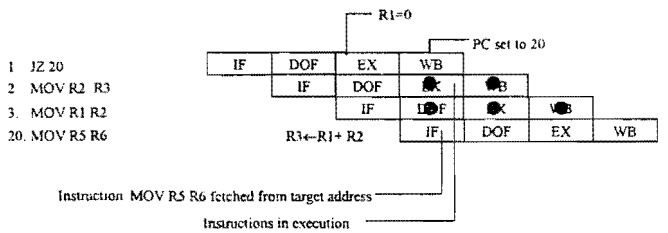
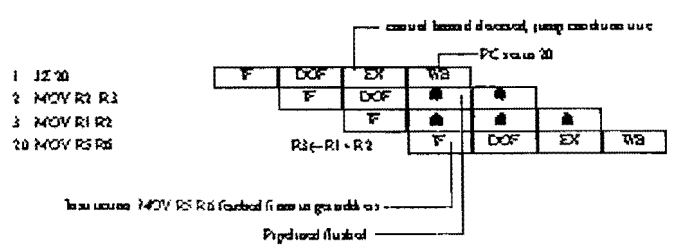


Fig 5 Control hazard

No operation instructions can be used to fix the control hazard problem. When a jump instruction is executed the next two instructions are already loaded in to the pipeline. If the jump condition is false the two instructions will be executed. If the jump condition is true the pipeline must be flushed and loaded with the instruction addressed by the jump. This method is called branch prediction, because first we assumed that the jump condition is false.



Control hazard solution

IV VHDL DESCRIPTION

One of the main differences between a typical and a pipeline microcontroller is the instruction decoder. The VHDL description of the instruction decoder, for the typical PIC14000 microcontroller, includes four case statements while the description for the pipelined microcontroller includes only three case statements.

```
case curentstate is

    when decode =>

        case instruction is

            when MOV =>
              <statements>

            when ADD =>
              <statements>
             etc.

        end case;

    when execute =>

        case instruction is

            when MOV =>
              <statements>

            when ADD =>
              <statements>
             etc.

        end case;

    when write_back =>

        case instruction is

            when MOV =>
              <statements>

            when ADD =>
              <statements>
             etc.

        end case;

end case;
```

```
case instruction1 is

    when MOV =>
      <statements>

    when ADD =>
      <statements>
     etc.

end case;

case instruction2 is

    when MOV =>
      <statements>

    when ADD =>
      <statements>
     etc.

end case;

case instruction3 is

    when MOV =>
      <statements>

    when ADD =>
      <statements>
     etc.

end case;

instruction1<= instruction;
instruction2<= instruction1;
instruction3<= instructio2;
```

The next paragraph presents a part of the VHDL description for the instruction decoder of PIC14000 pipelined microcontroller.

```
PROCESS (clk, reset)
BEGIN

 if reset='0' then(.....)
 elsif clk1'event and clk1='1' then

        if IR1( 7 downto 0) = IR2(7 downto 0) then
        hazard<=1;  -- data hazard detected
        else

        case IR1(13 DOWNTO 8) is

        when NOP =>
                if skip_trig='1' then
                        inc_pc <= '0';
                else
                        inc_pc1 <= '1';
                end if;

         when SUBWF =>
                w_oen <= '1';
                file_moen <= '1';
                inc_pc <= '1';

         when DECF =>
                file_moen <= '1';
                inc_pc <= '1';

         when GOTO =>
                inc_pc1 <= '0';

end case;

case IR2(13 DOWNTO 8) is

        when NOP =>

        when SUBWF =>
                alu_op <= ALUOP_SUB;
                zero_wen <= '1';
                carry_wen <='1';

        when DECF =>
                alu_op <= ALUOP_DEC;
                zero_wen <= '1';

         when GOTO =>
                inc_pc1 <= '0';
                next_state <= Q3;
end case;

case IR3(13 DOWNTO 8) is

         when NOP =>
                if skip_trig='1' then
                        clr_skip_trig <='1';
                end if;
                ir_en <= '1';

        when SUBWF =>
                zero_wen <= '1';
                carry_wen <='1';
                if dest = '0' then w_wen <= '1';
                else file_mwen <= '1'; end if;

        when DECF =>
                zero_wen <= '1';
                if dest = '0' then w_wen <= '1';
                else file_mwen <= '1'; end if;
```

```
        when GOTO =>
                data_ir<=ir3(10 downto 0);
                 inc_pc2<='0';
                jmp_en <= '1';
                pch_en <= '1';
               inst_skip<= '1'
           set_skip_trig <='1'; ;--control hazard triger
           pch_trig <='1';
end case;

if skip_trig ='1' then
        IR3<=IR2;
        IR2<="0000000000000";
        IR1<="0000000000000";
else
        if hazard ='1' then

                IR3<=IR2;
                IR2<="0000000000000";
        else
                IR3<=IR2;
                IR2<=IR1;
                IR1<=inst;
         end if;
end if;

END PROCESS;



PROCESS (set_skip_trig,clr_skip_trig,reset)
BEGIN

if reset='0' then
        skip_trig<='0';

elsif clr_skip_trig='1' then
        skip_trig<='0';

elsif set_skip_trig'event and set_skip_trig='1' then
        skip_trig<='1';

end if;

END PROCESS;
```

The three case statements correspond to DOF, EX and WB cycles. The IR1, IR2 and IR3 registers contain the instructions that are loaded into pipeline. Data hazard is detected if the destination register for one instruction is the same with the source register for the next instruction. A no operation instruction is inserted in the IR2 register if data hazard is detected. The control hazard routine is triggered by skip_trig signal. If there is a true condition for a jump instruction, skip_trig signal is set to '1' and a no operation instruction is inserted in the IR1 and IR2 registers. When a no operation instruction is process by the microcontroller the only action taken is to increment the program counter. A no operation instruction introduced by the hazard routine has no action at all (the program counter is not incremented). This can be done using a branch for NOP instruction in the DOF stage.

The instruction decoder is the component which has big modifications in its structure. The other components of the PIC14000 microcontroller have small timing modifications. Because of timing reasons it is necessary to insert a register between ALU and data bus. Another observation is that it is necessary to set to '0' the skip_trig signal after the hazard routine had been started.

V CONCLUSION

Using a pipelined datapath it is possible to increase the speed of PIC14000 microcontroller without changing its clock frequency. The slightly modification in its structure does not involve an increase in its power consumption. The performance of the PIC14000 microcontroller with pipeline architecture can be increase if a two edge clock is to be used. In both cases is necessary to pay attention to the timing problems between internal components.

REFERENCES

[1] Daniel Mic, Stefan Oniga: Circuite Logice Programabile, Risoprint, 2002.

[2] Edward Karalis: Digital Design Principles and Computer Architecture, Prentice Hall, 1997.

[3] Mark Zwolinski: Digital System Design with VHDL, Prentice Hall, 2000.

[4] Morris Mano, Charles Kime: Logic and Computer Design Fundamentals, Prentice Hall, 2000.

[5] Stefan Oniga: Circuite Digitale, Risoprint, 2002.

[6] Thomas L. Floyd: Digital Fundamentals, Prentice Hall, 2000.

[7] xxx PIC16/17 Microcontroller Data Book, Microchip, 1996.