# Design and Implementation of Simple and Extendable Microprocessor Based on FPGA Technology

**Hassan. J. Hassan**

**University of Technology**

**Waleed. F. Shareef**

**University of Technology**

## Abstract :

**Field programmable gate array (FPGA) are integrated circuits which can be user customized to implement arbitrary digital functions. In this paper, simple and extendable microprocessor is designed and implemented, connected with small extendable read only memory (ROM) which can be used to store the instruction codes of the programs.**

**The design and implementation of the microprocessor system have been done by using very high speed descriptive language (VHDL), and its implementation on field programmable gate array (FPGA) by using the synthesis tools of (Xilinx foundation series 4.1i program), while simulation carried out in ModelSim5.4 environment.**

**The flexibility, low cost, and real time operation are the main features taken into consideration for the proposed design. Several programs are implemented and simulated, which will be demonstrated through several examples.**

# 1. Introduction:

**Field-Programmable Gate Array (FPGA) is a programmable logic device capable of implementing complex digital system [1]. It is a uniform structured VLSI chip that can be configured and reconfigured for different designs [2, 3].**

**FPGA is a digital device that, thanks to the possibility of implementing logic circuits by programming the required functions, offers the benefits of low costs, short manufacturing turnaround time and easy design changes. As a result, most prototypes and many production designs are now implemented on FPGAs, making hardware implementation economically feasible even for those applications which were previously restricted to software implementation [4]. One of the most suitable languages for the FPGA devices is the Hardware Description Language [2, 3, 5].**

**VHDL is an acronym for VHSIC hardware description language. VHSIC in turn stand for very high-speed integrated circuits. The original standard for VHDL was adopted in 1987; a revised standard was adopted in 1993 and called IEEE 1164 [6, 2].**

**VHDL is designed to fill a number of needs in the design process. Firstly, it allows description of the structure of a design. Secondly, it allows the specification of the functions of design using familiar programming language forms. Thirdly, as a result, it allows the design to be simulated before being manufactured [7].**

**VHDL is a powerful hardware description language that can be used to model a digital system at many levels of abstraction, ranging from the algorithmic level to the gate level [5, 8].**

**In traditional FPGA design flow, a user enters a description of the desired circuit by using a hardware description language (HDL) such as VHDL. A CAD tool is used to synthesize a netlist from the HDL description. Another CAD tool used to decompose the netlist to fit the logic resources of the FPGA, and then place and route (P&R) tool is used. Following this, the logic and interconnection of the FPGA is defined, and the CAD tool outputs a bitstream, which can be downloaded to the FPGA's configuration RAM [9].**

**In this paper, the hardware description of Extendable Microprocessor Based on FPGA Technology chip through the use of VHDL language is addressed.**

## 2. Design layout:

**This work presents the design of central process unit (CPU) in VHDL, in such away to ensure the ability of developing and extending system parts easily.**

**The design demonstrates a simple CPU which is provided with 22 instructions, including the main essential operations available in the most of CPU's. The communication between CPU and external modules takes place via 16-bit data bus. The address bus consists of 12 lines. As a result, a memory unit of 4096 word is needed to be connected to the proposed processor. The memory was modelled using *non-synthesisable* VHDL code for simulation only. Since it doesn't represent a part of the proposed microprocessor design. It used only to store program during simulation. So, the final design specification viewed later, doesn't consider the memory size.**

**Assistant MATLAB program which is necessary for simulation process has been developed. This program acts like a simple assembler that receive the text files (written in mnemonics), and then convert them to text files which are written in machine code, in such away that can be entered directly to the processor during simulation process.**

**The instructions set approved by the microprocessor module are explained in the next section, while internal units (and theirs algorithms) are reviewed in section four. Reflection of design aspects on expandability is addressed in section five. Simulation and synthesis results are viewed in section six, while final conclusions are drawn in section seven.**

## 3. Instruction Set

**The proposed design represents a microprocessor for *Complex Instruction Set Computer* (CISC). Its instruction set (see Table 1) consists of three main categories:**

### 3.1. Memory Reference Instructions

**This category involves instructions that deals with operand stored at memory. These are seven instructions as follow:**

- **Arithmetic and logic group: ADD and AND instructions that take accumulator and specific memory location as input operands and the result is stored back to accumulator.**
- **Data transfer group: STA and LDA instructions that used to store/load accumulator to/from specific memory location.**
- **Loop handling: BUN instruction which used, alone, as *unconditional jump* and with ISZ instruction as *conditional jump*.**
- **Subroutine handling: BSA instruction used to implement *call* operation by save address return at the first location of the target subroutine.**

**Each one of these instructions can implement *direct* and *indirect addressing mode* (depending on left most bit of instruction code). Using *indirect mode* with BSA instruction offers a mechanism for returning from subroutines.**

### 3.2. Register Reference Instructions

**The microprocessor offers 12 *register reference instructions* which classified into two groups:**

- **Accumulator group: These instructions affect the values of accumulator and carry flag. It includes: CLA, CLE, CMA, CME, CIR, CIL, and INC instructions.**
- **Conditional group: These instructions are used with BUN instruction to implement *conditional jump* also. It includes: SPA, SNA, SZA, and SZE instructions.**

**This category involves another useful instruction, which is NOP. Execution of this instruction lead to idle processor cycle which is useful in implementing delay loops.**

### 3.3. Interrupt handling and Processor Control Instructions

**After reset, the microprocessor starts execution from third location in the memory, leaving the first two locations as *Interrupt Vector Table*. Whenever high logic signal is applied on port INTR for one clock cycle, the microprocessor does the following procedure:**

1. **Check the interrupt flag R, if it is zero skip, else continue procedure.**
2. **Stop execution in current program.**
3. **Store next instruction address at location 0.**

4. **Execute instruction at location 1, which must contain address of an *Interrupt Service Routine* (ISR).**
   **Returning to the main program achieved by use indirect branch to address 0.**

**To handle interrupt, the design provides two instructions to switch on/off the R flag. These are ION and IOF.**

**Finally, one processor control instruction is available, HLT that is used to bring microprocessor to shutdown state.**

## 4. Microprocessor architecture

**The microprocessor has 16-bit data bus, 12-bit address bus, and four line control/status signals. These signals involve: two 1-bit to interface memory read/write, *Reset* signal, and 1-bit signal to indicate processor state halt/ready. The proposed microprocessor is designed in structural style that involves four internal components: *Arithmetic and Logic Unit* (ALU), *Registers Unit* (RU), *Sharedbus Unit* (SU) and *Control Unit* (CU). Figure 1 shows how these components are connected together to form the proposed microprocessor.**

### 4.1. Register Unit (RU)

**RU represents internal data repository of the microprocessor. It involves: *Address Register* (AR), *Program Counter* (PC), *Accumulator* (AC), *Data Register* (DR), *Instruction register* (IR), and *Temporary Register* (TR).**

**AR and PC are 12-bit each. The PC register is responsible to hold the address that points the location of the next instruction, while AR is used to hold the address of the data being accessed by the current instruction.**

**AC register is the only general purpose register in the microprocessor. It is 16-bit register used in all instruction that involve data store/load and data manipulation.**

**DR and TR registers used as temporary holders for intermediate data. In contrast to DR register which is used in all instruction to hold the data being accessed in memory,**

**TR is used only in instructions that involve multi micro-operations. At last, each instruction read from memory is stored in IR register.**

**The RU contains also two flag associated with the registers: ZAC, ZDR which indicate *zero accumulators* and *zero data register* respectively. (Also the design offers other flag that discussed later in section 4.4). Figure 2 summarizes names and sizes of these flags and registers.**

**The RU is modelled using behavioural modelling style. A piece of code that represents the *Accumulator* is listed below:**

```
ac_reg:process
begin
wait until ck='1' and ck'event;
if clr_ac='1' then          acs<=(others=>'0');
elsif ld_ac='1' then        acs<=from_ALU;
elsif inc_ac='1' then acs<=acs+1;
else  null;
end if;
end process;
```

## 4.2. Arithmetic and Logic Unit (ALU)

**Since the proposed microprocessor implements basic samples of arithmetic and logical instruction, its ALU design is very simple. The ALU is modelled as combinational circuit using dataflow modelling style, and it takes AC and DR as inputs while the result goes to AC and Carry flag.**

**With six operations only, ALU take another 3-bit input to determine the current operation. The next piece of code shows *with-select* statement used to realize the output that goes to AC:**

```
with sel select
result <=
ac and dr                   when sel="000",
ac +   dr                   when sel="001",
```

```
dr                          when sel="010",
not ac                      when sel="011",
ac(14 downto 0) & ci        when sel="100",
ci & ac(15 downto 1)        when sel="101",
x"0000    when others;
```

**A similar *with-select* statement is also needed in the ALU to realize the output that goes to Carry flag.**

## 4.3. Sharedbus Unit (SU)

**Sharedbus unit represent a multiplexer that arbiter which register must transfer its content to bus. Given seven sources (memory and six registers in RU), selection line of 3-bit must be applied by the control unit to transfer one of them to the bus.**

**Sharedbus unit is so simple that implemented using just one *with-select* (data flow style) statement as follow:**

```
with s select
to_bus<="0000" & arwhen "001",
        "0000"& pc when "010",
                dr when "011",
                ac when "100",
                ir when "101",
                tr when "110",
              memory when "111",
   (others=>'Z' ) when others;
```

## 4.4. Control Unit (CU)

**There are two primary methodologies for designing control units. *Hardwired control* uses sequential and combinatorial logic to generate control signals, whereas *microsequenced* control uses a lookup memory to output the control signals. Each methodology has several design variants [10, 11]. This design uses hardwired control methodology.**

**As all the CISC microprocessor that uses hardwired control, the control unit is complicated [10, 11]. Inside CU there is three subunits (as shown in Figure 3):**

**Opcode decoder: This decoder take bits 12, 13 and 14 of instruction register and it is responsible of generating $D_0$ through $D_6$ that indicate the type of instruction. The output of this subunit is 8-bit mutual signal (one bit active at a time) as shown in the code below:**

```
with ir(14 downto 12) select
Temp_D<=
"00000001" when "000",
"00000010" when "001",
"00000100" when "010",
"00001000" when "011",
"00010000" when "100",
"00100000" when "101",
"01000000" when "110",
"10000000" when "111",
"00000000" when others;
```

**Sequence counter: This subunit points out the sequences of micro-operations needed to perform the fetch, decode, and execute cycles of every instruction.**

**This unit wasn't implemented as counter; instead a 16-bit shift left register (named *T*) is used as shown in the next piece of code:**

```
if clk='1' and clk'event then

if temp_clr_sc ='1' then   T<="0000000000000001";

else T(15 downto 0)<= T(14 downto 0) & T(15);

end if;

end if;
```

**At reset, *T* is set to 8000h then it starts to shift left at each clock cycle. For example, after one clock cycle *T* content becomes 0001h, i.e. all bits are zero except $T_0$ which equal to 1. After second clock, *T* content becomes 0002h, i.e.**

**all bits are zero except $T_1$. Therefore, when $T_4$ is active it indicates the fifth clock cycle.**

**The third subunit is a group of combinational logic that make use of $T_0$ through $T_{15}$ and $D_0$ through $D_6$ in addition to other external signals (*clock*, *INTR*, *Reset* and flags). This group is responsible of generating the selection lines and control signals that manage other components as follow:**

1. **Three bit selection lines to ALU to determine operation.**
2. **Three bit selection lines to sharedbus unit to determine data source.**
3. **Individual signal that control *clearing*, *loading*, and *incrementing* the registers and flags.**
4. **Update carry flag E.**
5. **Update interrupt flag R.**
6. **Update sequence counter.**

**Below are sample of code that implements some of these operations:**

```
mem_write<= (D(3) and T(4)) or (D(5) and T(4)) or (D(6)and T(6)) or ( R and T(1));

clr_ar<=  R and T(0);
inc_ar <= D(5) and T(4);
ld_pc <=(D(4) and T(4)) or          (D(5) and T(5));
```

```
clr_pc<= R and T(1);

clr_dr<= '0';

ld_dr<= (D(0) or D(1) or D(2) or D(6) ) and T(4) ;

inc_dr <= D(6) and T(5);

clr_ac <=r_inst and B(11);
```

## 5. Designs Expanding

**Although the proposed microprocessor instruction set includes a sample of most basic operations, it is still possible to extend it to incorporate more instructions easily. There are 16-bit in each memory word, so the available instruction numbers is $2^{16}$ at maximum. Taking into account that some instructions incorporate memory address, it is still there are much non-used codes.**

**A modification may also be adding general purpose registers to register unit. Since the overall design modeled in structural style, adding control line to these registers implies adding new sections to the *combinational logic* subunit in the control unit.**

## 6. Simulation and Synthesis Results

**Simulation was implemented by running various programs on the proposed design. Figure 4.a shows sample program that used to sum ten numbers and store result back in memory. This sample program was assembled and stored in memory (with the accompanied data). Figures 4.b and 4.c show the memory contents before and after program execution respectively, while its simulation time diagram is shown in Figure 5.**

**Another program (shown in Figure 6) that used to add two double precision (32 bits) numbers was applied. This program uses circulates left instruction to get the carry bit. Figure 7 shows a third sample program that used to set/reset bits of a certain number at memory. Setting and resetting operations are done also through the carry bit. The programs' simulation time diagrams are shown in Figure 8 and Figure 9 respectively.**

**Simulation results were produced using ModelSim4.5 environment. The microprocessor itself was designed in the hardware description language VHDL in the ISE4.1 environment. The selected target device is Xilinx Virtex FPGA-xcv50-6bg256.**

**Finally, *device utilization summary* and *timing summary* for the design (as stated by the *synthesis* and *place and route* reports) are shown in Table 4 and Table 5 respectively.**

## 7. Conclusions

**In this paper, the design of extendable basic microprocessor in VHDL language is presented. The proposed microprocessor characterized by hardware control unit and CISC architecture with 22 instructions that cover samples of essential operations (data manipulation, loops, decision making, etc.).**

**During simulation, a number of programs represent general algorithms were executed to check the design ability. Synthesis results show that the design running at maximum frequency of 43.896MHz, and occupy about 27% of the target device.**

**Its structural architecture and simple design make it a good starting point to develop microprocessor core for various application.**

## 8. References

**[1] T. S. Czajkowski, "*A synthesis oriented Omniscient Manual Editor for FPGA Design",* M.Sc. Thesis, Toronto University, 2004.**

**[2] S. Brown and Z. Vranesic, "*Fundamentals of Digital Logic with VHDL Design",* McGraw-Hill Company, 2000.**

**[3] J. F. Wakerlay," *Digital Design: Principals and Practices",* Prentice Hall, 2000.**

**[4] S. D. Brown, J. Rose, "*FPGA and CPLD Architectures: A Tutorial",* IEEE Design and Test of Computers, Vol. 13, No. 2, pp. 42-57, 1996.**

**[5] J. Bhasker, "*A VHDL Primer",* Prentice Hall PTR, third edition, 1999.**

**[6] M. Zwoliniski,"*Digital System Design with VHDL"*, Prentice Hall Inc., 2000.**

**[7] P. J. Ashenden,"*The VHDL cookbook",* first edition, 1990.**

**[8] F. Scarpino,"*VHDL and AHDL Digital System Implementation",* Prentice Hall, Inc., 1998.**

**[9] A. DeHon,"The density advantage of configurable computing," *Computer,* Vol. 33, No. 4, pp. 41-49, 2000.**

**[10] John D. Carpinelli, "Computer Systems Organization and Architecture", Addison Wesley, 2000.**

**[11] M. Morris Mano, "Computer System Architecture", third edition, 1994.**

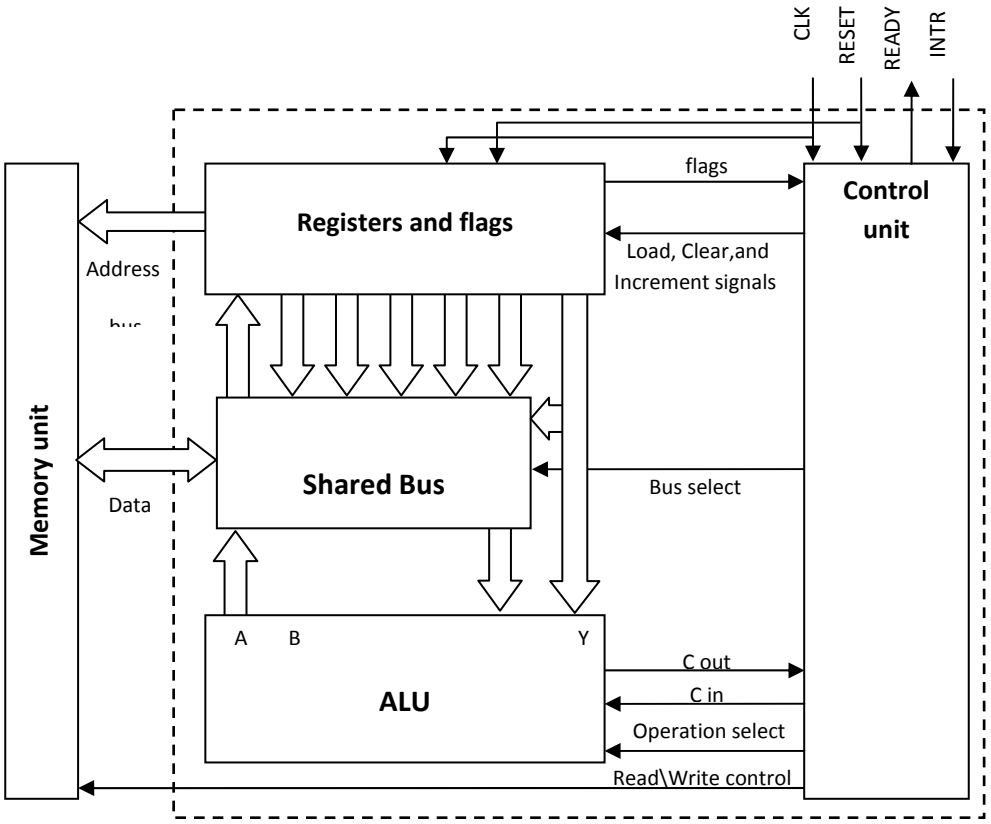


**Figure 1: proposed microprocessor architecture**

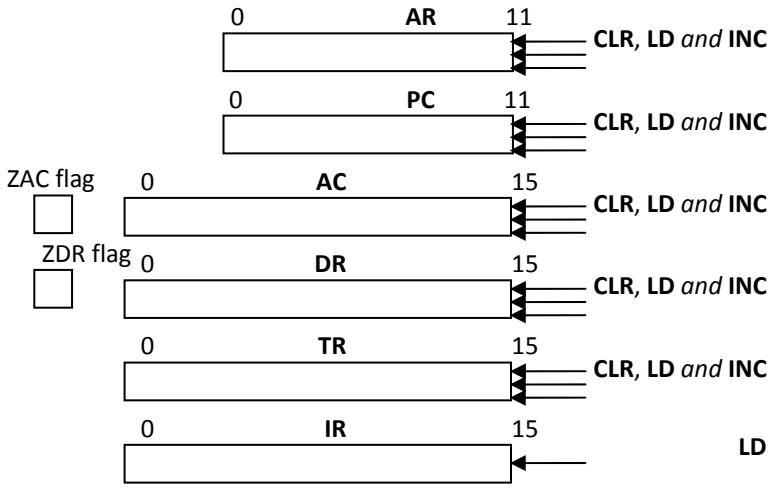


**Figure 2: Register Unit**

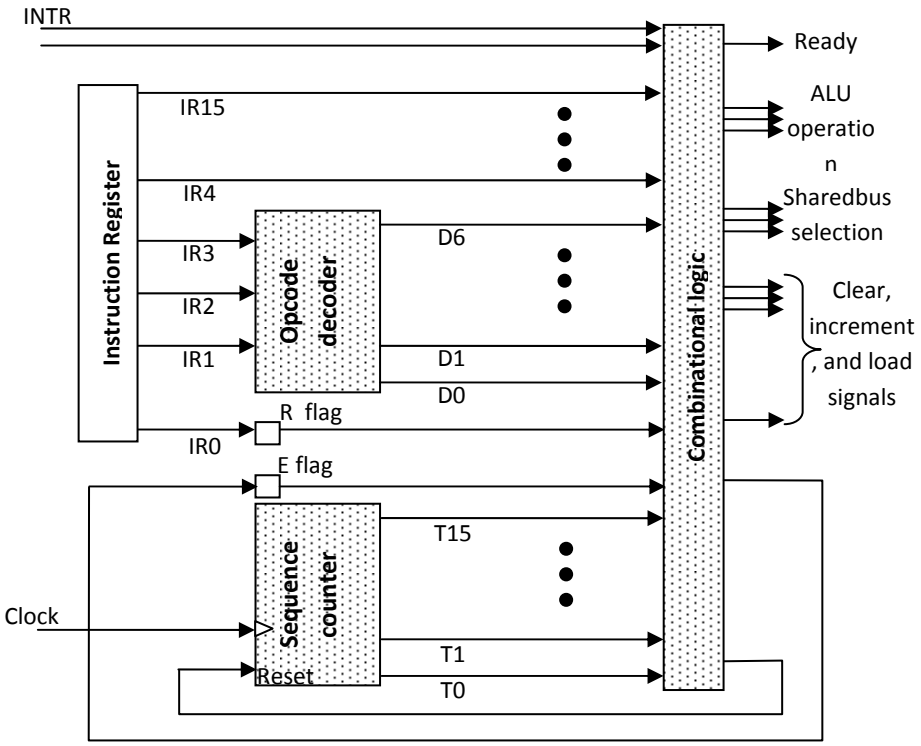


**Figure 3 Control Unit**

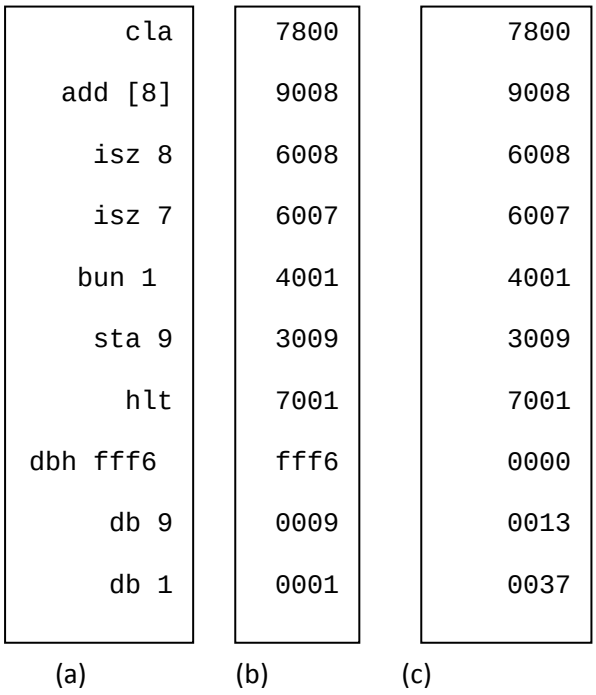


**Figure 4 (a) Program (in assembly) that add ten numbers with its accompanied data.**

**The machine code for the program (b) before and (c) after the execution.**

## Table 1: Instruction set (Mnemonics and codes)

| Mnemonic | Instruction code | Operation |
|---|---|---|
| AND | I000 XXXX XXXX XXXX | AC ←AC + M[*effective address*] |
| ADD | I001 XXXX XXXX XXXX | AC ←AC *and* M[*effective address*] |
| LDA | I010 XXXX XXXX XXXX | AC ←M[*effective address*] |
| STA | I011 XXXX XXXX XXXX | M[*effective address*] ←AC |
| BUN | I100 XXXX XXXX XXXX | PC ←*effective address* |
| BSA | I101 XXXX XXXX XXXX | M[*effective address*] ← PC<br>PC ←*effective address*+1 |
| ISZ | I110 XXXX XXXX XXXX | M[*effective address*] ← M[*effective address*] +1<br>*if* M[*effective address*] = 0 *then* PC← PC +1 |
| CLA | 0111 1000 0000 0000 | AC ← 0 |
| CLE | 0111 0100 0000 0000 | E (Carry flag ) =0 |
| CMA | 0111 0010 0000 0000 | AC ← *not* AC |
| CME | 0111 0001 0000 0000 | E ← *not* E |
| CIR | 0111 0000 1000 0000 | AC(15 -0)←E *combined with* AC(15-1)<br>E ←AC(0) |
| CIL | 0111 0000 0100 0000 | AC(15 -0)← AC(14-0) *combined with* E<br>E ←AC(15) |
| INC | 0111 0000 0010 0000 | AC←AC+1 |
| SPA | 0111 0000 0001 0000 | *If* AC(15)=1 *then* PC←PC+1 |
| SNA | 0111 0000 0000 1000 | *If* AC(15)=1 *then* PC←PC+1 |
| SZA | 0111 0000 0000 0100 | *If* AC=0 *then* PC←PC+1 |
| SZE | 0111 0000 0000 0010 | *If* E =0 *then* PC←PC+1 |
| HLT | 0111 0000 0000 0001 | *Processor halt* |
| NOP | 0111 0000 0000 0000 | *No operation* |
| ION | 1111 0000 1000 0000 | *Set* R (Interrupt Flag)=1 |
| IOF | 1111 0000 0100 0000 | *Set* R (Interrupt Flag)=0 |

**Table 2: Device utilization summary**

| | | |
|---|---|---|
| Number of External IOBs | 64 out of 180 | 35% |
| Number of SLICEs | 209 out of 768 | 27% |
| Number of GCLKs | 1 out of 4 | 25% |
| Number of TBUFs | 16 out of 832 | 1% |

**Table 3: Timing Summary**

| | |
|---|---|
| Speed Grade | -6 |
| Minimum period<br>(Maximum Frequency) | 22.781ns<br>(43.896MHz) |
| Minimum input arrival time before clock | 9.366ns |
| Maximum output required time after clock | 23.526ns |
| Maximum combinational path delay | 10.111ns |

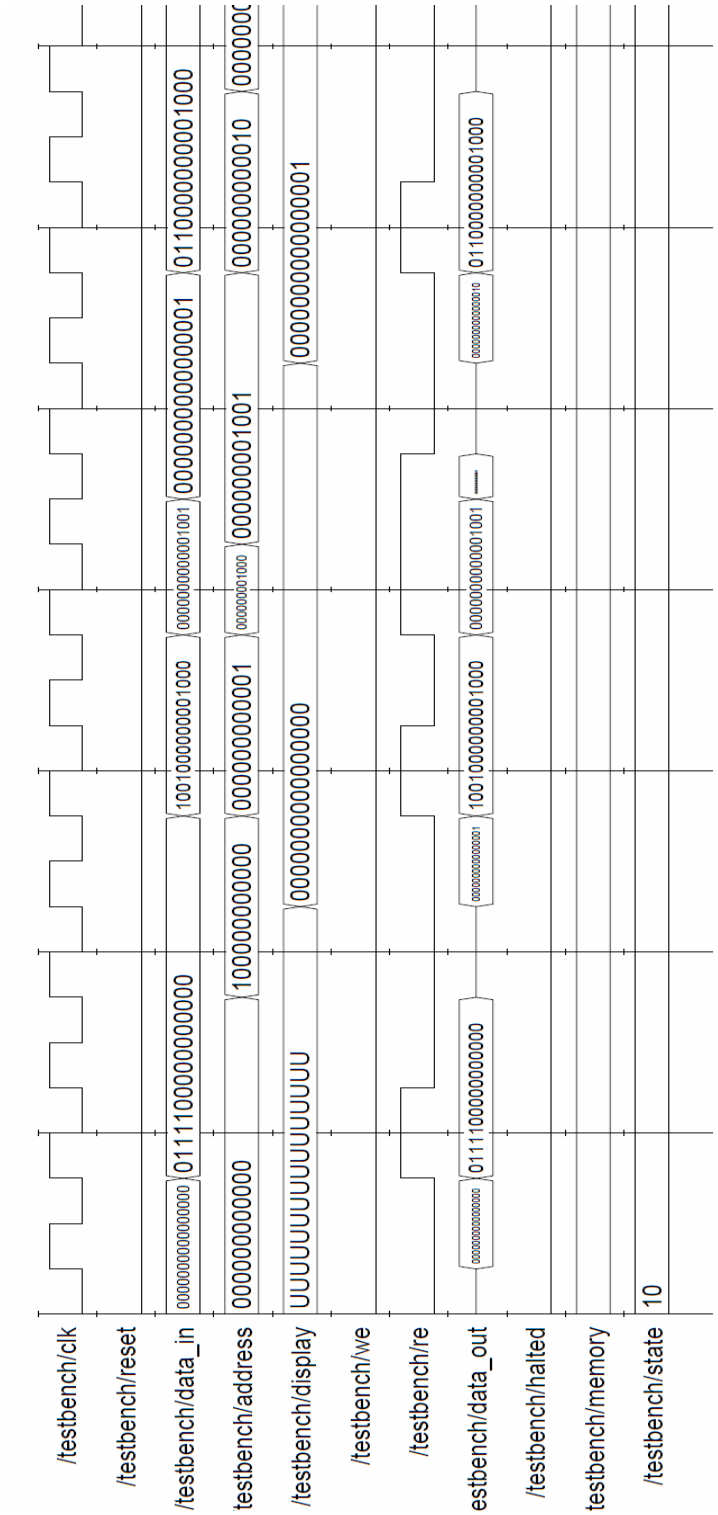


**Figure 5 Simulation Time Diagram (for the first fifteen clock cycles of the program in Figure 4)**

| Mnemonic | Code (HEX) |
|---|---|
| CLE | 7400 |
| LDA 40 | 2040 |
| ADD 50 | 1050 |
| STA 60 | 3060 |
| CLA | 7800 |
| CIL | 7040 |
| ADD 41 | 1041 |
| ADD 51 | 1051 |
| STA 61 | 3061 |
| HLT | 7001 |

**Figure 6: Program (in assembly and hexadecimal) that add two double precision (32 bits ) numbers**

| Mnemonic | Code (HEX) |
|---|---|
| CLE | 7400 |
| LDA 20 | 2020 |
| CIL | 7040 |
| CLE | 7400 |
| CIR | 7080 |
| CIR | 7080 |
| CLE | 7400 |
| CME | 7100 |
| CIL | 7040 |
| HLT | 7001 |

**Figure 7: Program (in assembly and hexadecimal) that set/reset bits of a number at certain memory locations**

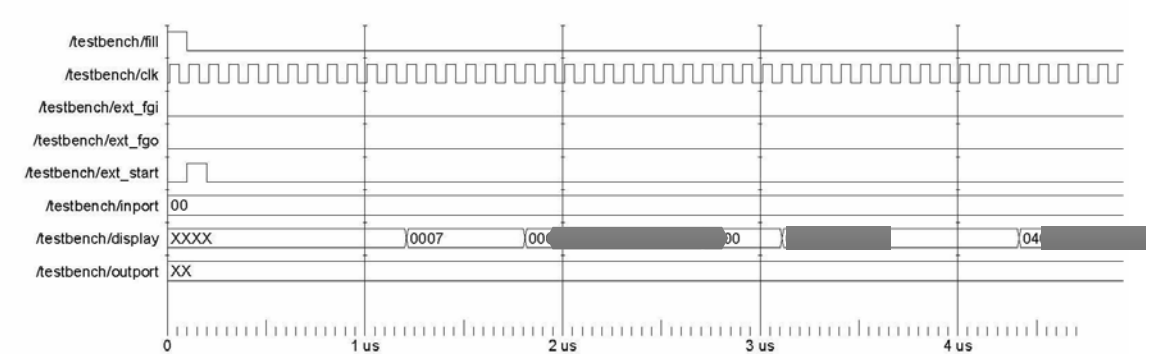


**Figure 8 Simulation Time Diagram for program in Figure 6. (The program used to add the numbers $0000\_00007_H$ with $0400\_FFFF_H$). The result is $0401\_0006_H$.**

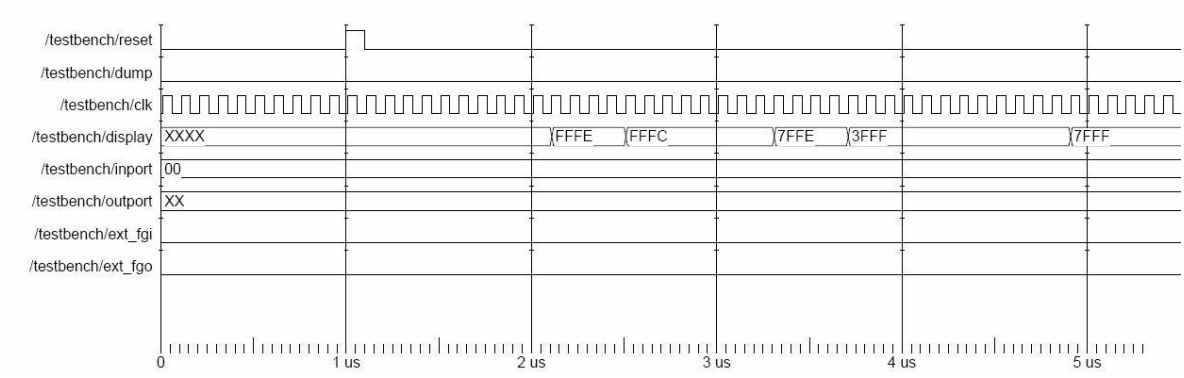


**Figure 9 Simulation Time Diagram for program in Figure 7. (The program used to set bit 0 and reset bit 15 of the numbers $FFFE_H$). The result is $7FFF_H$.**

# تصميم و تنفيذ معالج دقيق اساسي و قابل للتوسع مبني على تقنية البوابات المبرمجة ميدانياً

م. حسن جليل حسن
الجامعة التكنلوجية

م.م. وليد فواز شريف
الجامعة التكنلوجية

**المستخلص :**

ان مصفوفات بوابات المجال المبرمج (FPGA) هي عبارة عن مجموعة دوائر متكاملة لها القدرة على بناء دوال منطقية مختلفة. في هذا البحث تم تصميم و تنفيذ معالج دقيق(Microprocessor) بسيط ويتميز بقابليته على التوسع في اضافة عمليات اضافيه مستقبلا. تم ربط المعالج بذاكرة صغيرة قابلة للتوسع تستخدم لخزن شفرة ايعازات البرنامج.

تم تصميم المعالج الدقيق باستخدام لغة وصف الدوائر الفائقة السرعة (VHDL) وذلك باستخدام برنامج ( Xilinx foundation series 4.1i program) في حين تم تنفيذ المحاكاة في بيئة .ModelSim5.4

سرعة التنفيذ, الكلفة القليلة, والمرونة العالية من الامور المهمة التي تمت مراعاتها في التصميم. مجموعة من البرامج تم تنفيذها ومحاكاتها من خلال بعض الامثلة التي سوف يتم استعراضها.