2003 University/College IC Design Contest

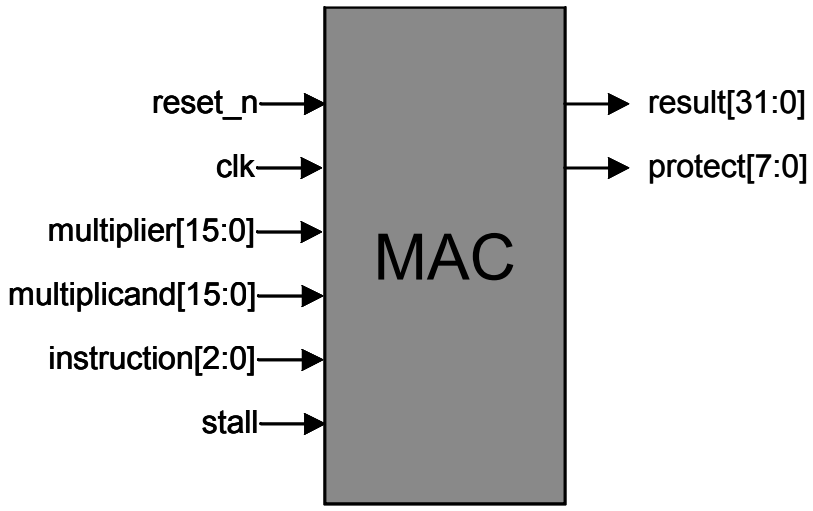
Cell-Based Category for Graduate/Undergraduate Level

MAC Design

I. Problem Description

Please design a bit-slice 16-bit-by-16-bit Multiplier-and-Accumulator (MAC) with saturation capability. The detailed specification, including I/O interface, functional behavior and timing requirement, will be given in the following sections. Each team should finish the design by utilizing the design environment provided by CIC within the given time. Both the front-end and the back-end design should be carried out.

II. Block Diagram and I/O



I/O Interface

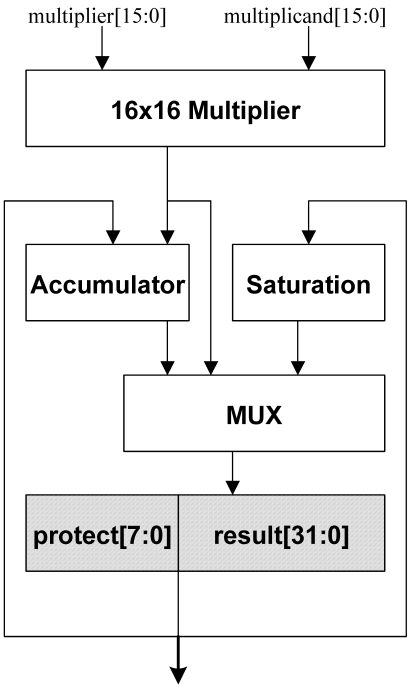
signal name	function	width	description
reset_n	input	1	Active-low asynchronous reset signal. When asserted, result and protect should be cleared to zero.
clk	input	1	Clock source. The MAC is a synchronous design triggered at the positive edge of clk.
multiplier	input	16	When performing 16bx16b, it is a 16-bit multiplier input. When performing 8bx8b, multiplier[15:8] and multiplier[7:0] are two independent 8-bit multiplier inputs, respectively.
multiplicand	input	16	When performing 16bx16b, it is a 16-bit multiplicand input. When performing 8bx8b, multiplicand[15:8] and multiplicand[7:0] are two independent 8-bit multiplicand inputs, respectively.
instruction	input	3	Specify the operation of the MAC. instruction[2:0]: operation 000: result = 0, protect = 0 001: {protect, result} = multiplier * multiplicand 010: {protect, result} += multiplier * multiplicand 011: result = Saturation of {protect, result} 100: result = 0, protect = 0 101: {protect[3:0], result[15:0]} = multiplier[7:0] * multiplicand[7:0] {protect[7:4], result[31:16]} = multiplier[15:8] * multiplicand[15:8] 110: {protect[3:0], result[15:0]} += multiplier[7:0] * multiplicand[7:0] {protect[7:4], result[31:16]} += multiplier[15:8] * multiplicand[15:8] 111: result[15:0] = Saturation of {protect[3:0], result[15:0]} result[31:16] = Saturation of {protect[7:4], result[31:16]}
stall	input	1	Active-high signal. When stall is asserted, the operation is stalled for that cycle. Check out the timing diagrams for detail information.
result	output	32	The operation result.
protect	output	8	Overflow protection bits.

III. Functional Description

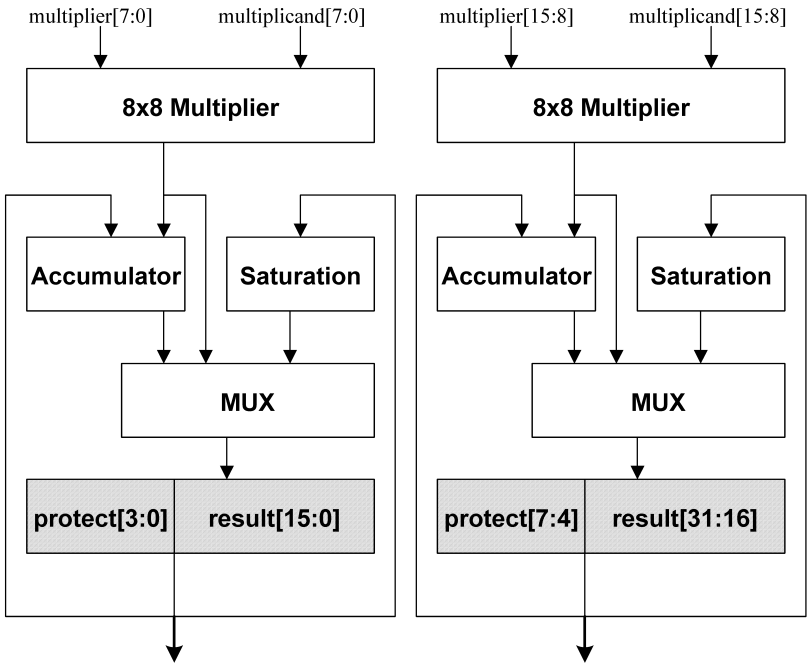
This MAC is capable to perform

- one 16-bit by 16-bit (2's complement) signed multiplication or MAC, and
- two concurrent 8-bit by 8-bit (2's complement) signed multiplications or MACs.

illustrated as follows.



16-bit x 16-bit mode



8-bit x 8-bit mode

The design is asynchronously reset by asserting the signal reset_n. After releasing the signal reset_n, the design begins to operate and is clocked by the **positive** edge of the signal clk. Two input ports, multiplier and multiplicand, provide the operands required for the multiplication and MAC operations in both 16-bit and 8-bit modes. The 3-bit signal instruction indicates what operations should be carried out for the current input cycle. **The design is required to be implemented as a two-stage pipeline architecture.** Check out the timing diagrams at the next section for detail information. The {protect, result} of the following formulate, which is performed as an operand by current instruction, can be obtained by the result of the previous valid instruction. MAC can not have two consecutive instructions with different bit-width MAC type. Instructions with different bit-width shall be separated by "Clear" instruction. For example, instruction=3'b001 and instruction=3'b110 can't not be the two consecutive instructions. However, instruction=3'b001, instruction=3'b100 and instruction=3b'110 can be the consecutive instructions. Each instruction operation is explained below:

- instruction[2:0] == 3'b000 : (clear)
  result[31:0] = 32h'0;
  protect[7:0] = 8'h0;

- instruction[2:0] == 3'b001 : (16-bit signed multiplication)
  {protect[7:0], result[31:0]} = sign_ext(multiplier * multiplicand);

  Since the result of the multiplication is 32-bit wide, the result should be sign extended to 40-bit for proper register transfer.

- instruction[2:0] == 3'b010 : (16-bit signed MAC)
  {protect[7:0], result[31:0]} += sign_ext(multiplier * multiplicand);

- instruction[2:0] == 3'b011 : (32-bit saturation)
  result[31:0] = saturation16({protect[7:0], result[31:0]});
  protect[7:0] not altered;

  The saturation16 tries to represent a 40-bit number in a 32-bit format. Obviously, lots of positive (negative) 40-bit numbers are too large (small) to be represented by 32-bit numbers. In these cases, saturation16 uses the largest(smallest) positive(negative) 32-bit numbers for replacement to minimize the truncation errors. Assume N[39:0] is a 40-bit number in 2's complement format, the operation of saturation16 is listed below:

```
if(N > 40'h007FFFFFF)
   saturation16(N) => 32'h7FFFFFFF    // largest positive in 32-bit
else if(N < 40'hFF80000000)
   saturation16(N) => 32'h80000000;    // smallest negative in 32-bit
else
   saturation16(N) => N[31:0];    // truncate the MSB, value not altered
```

- instruction[2:0] == 3'b100 : (clear)
  result[31:0] = 32h'0;
  protect[7:0] = 8'h0;

- instruction[2:0] == 3'b101 : (two concurrent 8-bit signed multiplications)
  {protect[3:0], result[15:0]} = sign_ext(multiplier[7:0] * multiplicand[7:0]);
  {protect[7:4], result[31:16]} = sign_ext(multiplier[15:8] * multiplicand[15:8]);

  Since the result of the multiplication is 16-bit wide, the result should be sign extended to 20-bit for proper register transfer.

- instruction[2:0] == 3'b110 : (two concurrent 8-bit signed MACs)
  {protect[3:0], result[15:0]} += sign_ext(multiplier[7:0] * multiplicand[7:0]);
  {protect[7:4], result[31:16]} += sign_ext(multiplier[15:8] * multiplicand[15:8]);

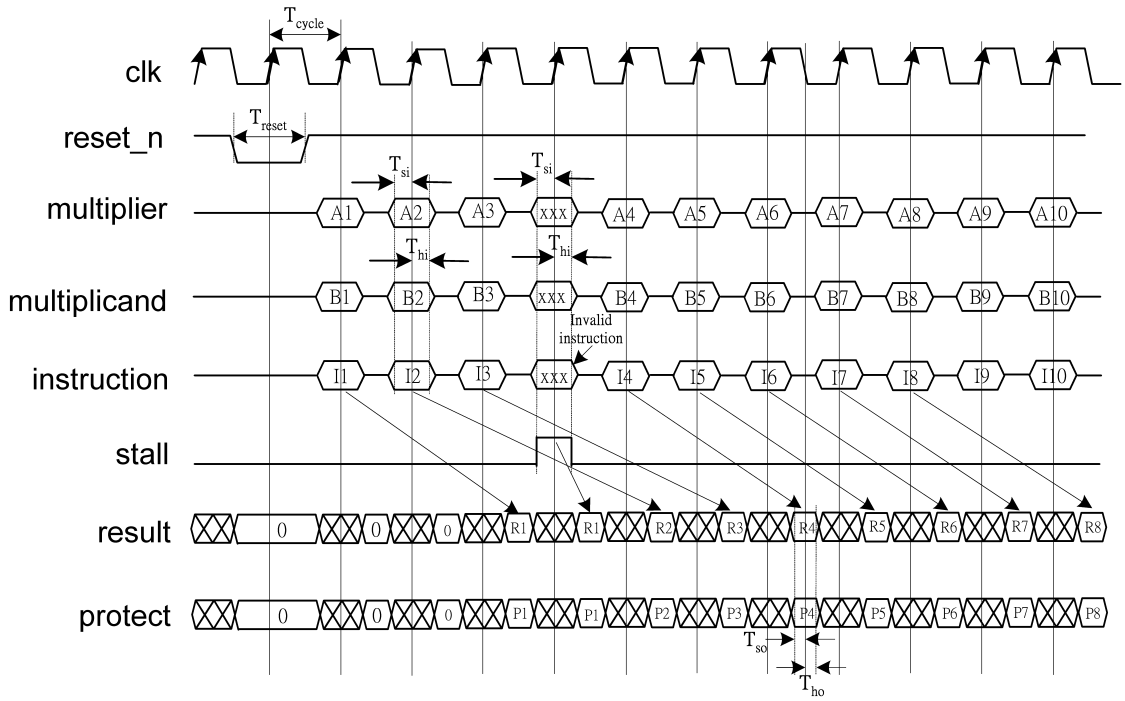
- instruction[2:0] == 3'b111 : (two concurrent 16-bit saturation operations)
  result[15:0] = saturation8({protect[3:0], result[15:0]});
  result[31:16] = saturation8({protect[7:4], result[31:16]});
  protect[7:0] not altered;

  The operation of saturation8 is similar to saturation16. Assume N[19:0] is a 20-bit number in 2's complement format, the operation of saturation8 is listed below:

```
if(N > 20'h07FFF)
   saturation8(N) => 16'h7FFF;    // largest positive in 16-bit
else if(N < 20'hF8000)
   saturation8(N) => 16'h8000;    // smallest negative in 16-bit
else
   saturation8(N) => N[15:0];    // truncate the MSB, value not altered
```

If the signal stall is asserted, the operation is stalled for that cycle. Check out the timing diagrams at the next section for detail information.

Timing Diagrams



Symbol	Description	Value
T_{cycle}	Clock (clk) period with duty cycle 50%	User defined
T_{reset}	reset_n pulse width, active between negative edge of clk.	= T_{cycle}
T_{si}	Time period from valid signal to positive edge of clk.	=0.2 T_{cycle}.
T_{hi}	Time period from positive edge of clk to invalid signal.	=0.2 T_{cycle}.
T_{so}	Setup time from valid output to negative edge of clk.	>0.5ns
T_{ho}	Hold time from negative edge of clk to valid output.	>0.5ns

Implementation Requirements

1. No gated clock is allowed
2. No latch is allowed

Score will be doubled for each violation. Please see the description in the next section for the details.

IV. Scoring

1. The referees will verify the function with the specifications of the design and check the completeness of the uploaded materials for scoring. A higher weighting will be given as part of function specifications is not met or some materials are lacked. If the front-end design is performed but back-end design (P&R, DRC/ERC, etc.) is not finished yet, the design and materials can still be transferred.
2. The formula of score：

$$\text{Score} = Area \times Time \times A \times B$$

Time: clock period, unit: ns.

Area: area of design boundary. The design boundary is the purple rectangle around the chip in Silicon Ensemble or the yellow dash rectangle in Apollo.

A: weighting for gated clock violation. If there is no gated clock in the design, A is equal to 1. Otherwise, if there is gated clock in the design, A is equal to 2.

B: weighting for latch violation. If there is no latch in the design, B is equal to 1. Otherwise, if there is latch in the design, B is equal to 2.

Rank: the lower the score, the higher the rank. If the score among teams are close, the committee of IC contest will decide the rank by the innovation of the design.

V. Others:

1. The following files are provided by the contest committee:

   00.README: Readme file.

   testfixture.v: Test bench including the definition of clock period

   mac.v: The input and output ports are declared in this file.

   synopsys.dc: The operating condition and boundary conditions file for the Synopsys Design Compiler setup (without the statements of design constraint).

   EXPECT.DAT: The expect result is recorded in this file.

   report.000: This file is used to describe the materials that should be handed in by each team. The design, file name, tool names, specifications and others are described in this file.

2. Please use the module ***mac*** in the file, ***mac.v***, to design Multiplier with Accumulator. The names and port types of IO are declared as follows:

```
module  mac  (instruction, multiplier, multiplicand, stall, clk, reset_n, result, protect);
input [15:0]    multiplier;
input [15:0]    multiplicand;
input           clk;
input           reset_n;
input           stall;
input  [2:0]    instruction;
output [31:0]   result;
output [7:0]    protect;
endmodule
```

3. Please use the test bench (testfixture.v) provided by CIC to verify your design. The referee will verify the correctness by using this test bench. In some cases, referee may use another test pattern to verify the design of each team.

i. In the file testfixture.v: Test bench is included in this file. The signals including clk, reset_n, multiplicand, multiplier, instruction, and stall are generated by test bench for verification. First, the 16-bit MAC instructions are tested, and then 8-bit MAC instructions are tested, the outputs of result and protect are verified by the test bench. The parameter, **HPHASE**, defined in the testfixture.v can be modified by each team to change the length of half CLK period.

ii. After the MAC instructions are verified, test fixture prompts **PASS** as all the results are correct. Otherwise, the cycle count, fault value and expected value of errors are displayed.

iii. synopsys.dc contains the basic setup environment of SYNOPSYS except the design constraints. In the synopsys.dc, you can find:

(1) Working environment: WCCOM.

(2) Except the CLK and reset_n, all the driving strengths of input ports are set as 1ns/1pf.

(3) All the loads of output ports are set as 1pf.

Please use the GUI command, ***Setup/Execute Script***, in Design Analyzer or use the command, "***include synopsys.dc***", in dc_shell to load the setup file: synopsys.dc. After you complete the synthesis procedure, the gate-level netlist and sdf file are created and can be used in the simulation tool.

iv. In the gate-level simulation, the cb35os142.v and the sdf file which is created by SYNOPSYS are needed. Even if the gate level netlist is not created by SYNOPSYS, this gate level netlist should be loaded by SYNOPSYS with the setup environment, sysnopsys.dc, and then created the sdf file for simulator. (Please refer to the description of 00.Readme.Verilog in CIC 0.35um Cell Based Design Kit.)

v. The block layout is adopted. That is to say, design is created by only core cells without using the IO pads:

(1) The width of power ring and power pin, marked as vdd! and gnd!, are fixed as **8um.**

(2) The positions of signal pins are decided by each team.

(3) Cadence SE user can refer to the following website for the details about the block layout:

http://www.cic.edu.tw/~nschang/seflow/hardmacro/hardmacro.htm

(4) Avant! Apollo user can refer the following website:

http://www.cic.edu.tw/~cschen/design_without_pad.html

vi. Layout verification (DRC and LVS) :

- ✧ The design created by Cadence SE: Please use Dracula to perform the DRC and LVS verification. For LVS, text labels are created on the layout and details are shown in the above web sites. All the pin names of the inputs and outputs should be the same as the assignment in the Section V.2.
- ✧ The design created by Avant! Apollo: Please use the built-in DRC and LVS verification tool in Apollo. Then, the Dracula DRC is adopted to recheck the DRC.

Appendix:

The materials that each team should hand in are listed in Appendix A. The upload steps are described in Appendix B.

Appendix A:

1 . Design database:

i. Design using Cadence SE: The completed design database directory of Cadence library (This directory is created by icfb) is required. Please use UNIX command, ***tar***, to add this directory into a .tar file. If your library name is **your_lib**:

tar cvf *yourname*.tar **your_lib**

You will get *yourname*.tar file.

If the design is performed by the Cadence SE, the **.def** file generated by SE should also be handed in.

ii. Design using Avant! Apollo：Please use UNIX command, ***tar***, to add the directory of Apollo library into a .tar file. If your library directory is **your_lib**:

tar cvf *yourname*.tar **your_lib**

You will get *yourname*.tar file.

2. Verilog synthesizable RTL file.
3. Verilog gate-level netlist file.
4. GDSII layout file.
5. Result files after layout verification:
   i. For Cadence SE: DRC output (*.sum) and LVS output (*.lvs).
   ii. For Avant! Apollo: Text file of the built-in DRC/LVS output in Apollo and DRC output file (*.sum) in Dracula.
6. The design architecture and special innovation can be described with the text or figures in another file. (by using the text editor or MS-word)

Appendix B:

1. Please create a new directory and then copy all the materials which have to be

handed in into the created directory. Then using the following commands to create an archive and to compress the archive:

> tar cvf **xxxxxxx.tar** *
↑
File name decided by you.
> compress **xxxxxxx.tar**

After using these two commands, you can find a file name with **xxxxxxx.tar.Z**.

2. Report file name (**report.xxx**): This file describes all the information that shall be handed in by each team. In this file, the file name, tools used in this design, specifications, and documentations are listed. The committee of IC contest will provide this file. Each team shall fill in the necessary information in the file **report.000**.

3. Finally, the compressed file (**xxxxxxx.tar.Z**) and the report file (**report.xxx**) have to be transferred with binary mode (ftp>binary) to one of the following ftp site. The username and password will be given by e-mail.（Please transfer the file, xxxxxxx.tar.Z, firstly, and then the file report.xxx.）

National Taiwan University：iccftp3.ee.ntu.edu.tw (140.112.17.202)
Chip Implementation Center：iccftp1.cic.edu.tw(140.126.24.8)
National Yunlin University of Science and Technology：
iccftp4.el.yuntech.edu.tw (140.125.35.19)
National Cheng Kung University：iccftp2.ee.ncku.edu.tw (140.116.156.55)

4. Please repeat the above steps when the design and materials are modified. The transferred file name and the file name described in the report file should be modified and updated

5. The report file should be prefixed with **report** and a number as the extended file name. For example, **report.000** is used as the first report file, **report.001** is used as the second version of report file, and so on.

6. Other notes please refer the manual of this contest.