2005 University/College IC Design Contest Preliminary

Cell-Based IC Category

A Computational System

1. Problem Description

Please design a computational system whose transfer function is defined in section 2.3 using CIC 0.18um UMC/Artisan cell-based design kit v2.0. Each team should complete the design according to the specifications given in the following section and the constraints as illustrated in Appendix A and Appendix B. You can only claim that your design is successful if it passes the checks by the test bench provided by CIC. You can refer to Appendix A and Appendix B for detailed information of files provided by CIC.

After the end of the contest, CIC will rate the successful designs according to the scoring rule defined in section 3. All the related materials as illustrated in Appendix C should be transferred to CIC. And the procedures for transferring are described in Appendix D. Note that the time for this contest is from 8:30 to 20:30. That means you have to transfer all the related materials to CIC before 20:30.

2. Design Specifications

2.1 Block Overview

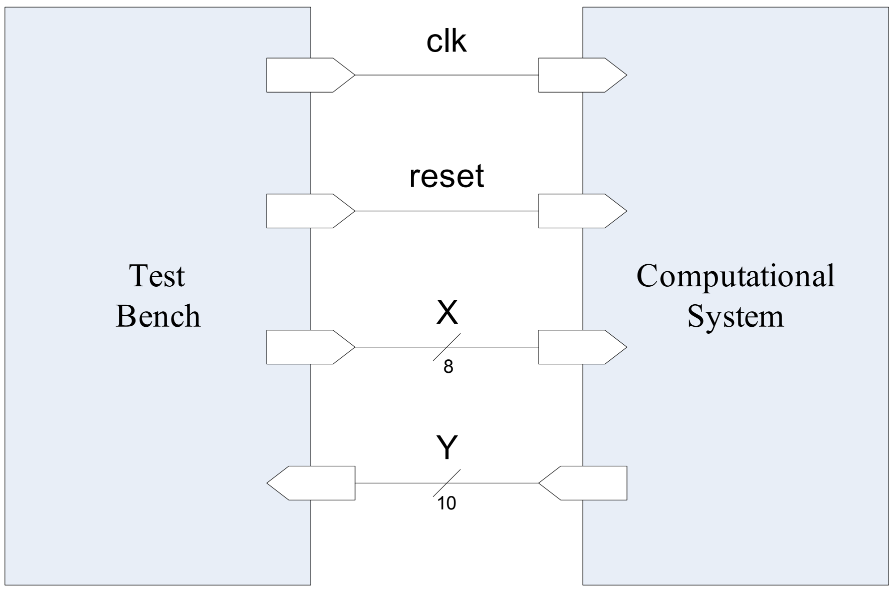

Fig. 1 Block Overview

2.2 I/O Interface

Table I - I/O Interface

Signal Name	I/O	Width	Description
clk	I	1	clock for the computational system
reset	I	1	reset the state of the computational system when it asserts
X	I	8	input data of the computational system
Y	O	10	computed output

2.3 Functional Description

Refer to Fig. 1, a series of 8-bit positive integer is generated as the input of the computational system by the test bench. The output value Y, which is a 10-bit positive integer is calculated according to equations (1), (2), (3) and (4). Note that $\lfloor \; \rfloor$ denotes the quotient operation.

$$Xavg_j = \left\lfloor \frac{\sum_{i=j}^{j+n-1} X_i}{n} \right\rfloor \quad \cdots\cdots (1)$$

where X_i is the value of the ith input data and $j \geq 1$.

$$XS = \{X_j, X_{j+1}, X_{j+2}, \ldots, X_{j+n-1}\} \quad \cdots\cdots (2)$$

$$Xappr_j = \begin{cases} Xappr_j = Xavg_j \\ \qquad \text{if } Xavg_j \in XS \\ X_i \,|\, (X_i \in XS) \; and \; (X_i < Xavg_j) \text{ and } (Xavg_j - X_i \text{ is minimal}) \\ \qquad \text{if } Xavg_j \notin XS \end{cases} \quad \cdots\cdots (3)$$

where $Xappr_j$ is the value of the jth approximate average.

$$Y_j = \left\lfloor \frac{\sum_{i=j}^{j+n-1} (X_i + Xappr_j)}{n-1} \right\rfloor \quad \cdots\cdots (4)$$

where Y_j is the value of the jth output data.

The computational system produces the output sequence according to the given input sequence. Each input and output data in the respective sequence is indexed. This index, in terms of hardware, is the relative time when the input data is given or the output data is ready. Thinking as a hardware designer, the approximate average is chosen from the last *n* input data which should be stored in the system. The system should be able to calculate the integral part of the real average of the last *n* input data first. And then if the integral part of the real average equals to any one of the last *n* input data, the approximate average is simply the integral part. Else the approximate average is the one which is one of the last *n* input data whose value is smaller than and most close to the integral part of the real average. The above descriptions stated the desired operations as those defined by equations (1), (2), and (3).

After the approximate average is obtained, the output value can be calculated according to equation (4). First, the last *n* input value is added by the corresponding approximate average. And then they are summed up and divided by *n-1*. The output value is the quotient after division.

For example, assume that n=4, X_1=3, X_2=24, X_3=16, X_4=8, and X_5=3. After the first 4 input items are given, the system should store them and calculate the output value. The average of the first 4 input values is 12(only the integral part is left). Since it is not in the set of $\{X_1, X_2, X_3, X_4\}$, the system selects one from $\{X_1, X_2, X_3, X_4\}$ as the approximate average whose value is smaller than 12 and close to 12. In this case, the approximate average is 8. So the first output value is calculated as $\lfloor[(3+8)+(24+8)+(16+8)+(8+8)]/[(4-1)]\rfloor = 27$. Similar to those described above, when the 5th input data item is given, the system should store X_2, X_3, X_4 and X_5 and calculate the corresponding output value. The 2nd output value should be the same as the first one because the values stored in the system is the same.

2.4 Timing Diagrams

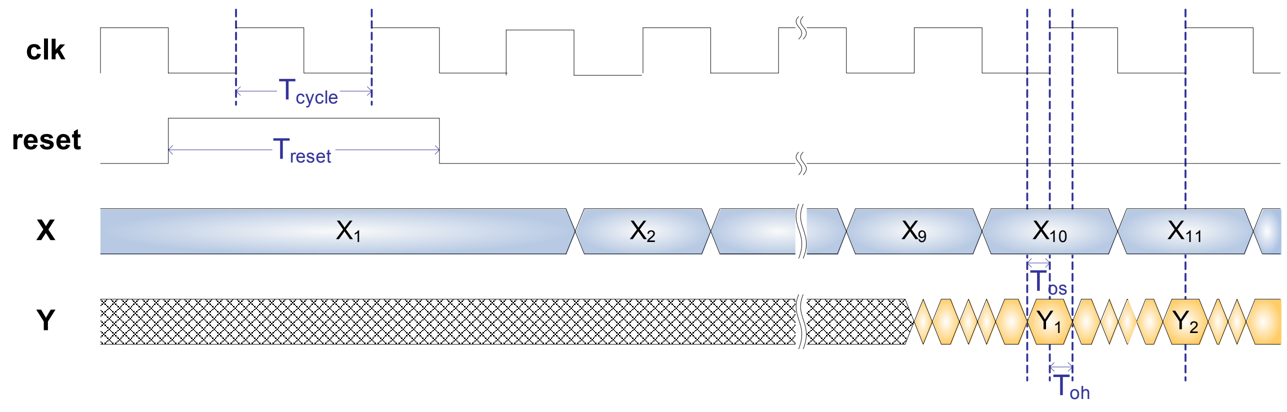

Fig. 2 I/O Timing Diagram

Table II

Symbol	Description	Value
T_{cycle}	clock period	user defined
T_{reset}	reset pulse width	2 T_{cycle}
T_{os}	setup time from valid output to positive edge of *clk*	0.5ns
T_{oh}	hold time from positive edge of *clk* to invalid output	0.5ns

The I/O timing diagram is as shown in Fig. 2. **For this contest, n in equations (1)-(4) are**

fixed to 9. The computational system is reset by asserting reset signal for 2 periods. The input X is changed to the next at the negative edges of the clock while output Y is checked by the test bench at positive edges of the clock. Note that the output should be stable around the positive edges of the clock. The setup and hold time requirements for the output are listed in Table II. The first output data should be valid after the input data changes from 9th one to 10th and before the next positive clock edge. After that, the output should be changed to the next at the next positive clock edge and so on, that is to say, the test bench checks one output value per clock cycle.

3. Scoring

The referees will check the completeness as well as the correctness of your design data. If all materials listed in RTL category and Gate-Level Category of Table V are uploaded to CIC, the design is complete. The correctness means the result of pre-layout gate-level simulation under the user-defined clock rate is the same as the golden result provided by CIC and without any setup/hold time violations.

Once the completeness and correctness are confirmed by the referees, the score is calculated. The scoring rule is quite simple. The earlier you complete your design, the higher score you'll get. And the higher the score, the higher the rank will be. Note that the time you complete your design is determined by the time the design of the latest version is uploaded to CIC's ftp sites.

If necessary, those designs with only files in RTL category uploaded are also ranked. In this case, the correctness means RTL simulation result is the same as the golden result provided by CIC.

Appendix

The related files and notes provided by CIC for designers using Verilog and VHDL are listed and described in the Appendix A and Appendix B respectively. The materials that each team should hand in are listed in Appendix C while the transferring procedures are described in Appendix D. Several inputs as well as their corresponding outputs and intermediums are provided as an example for debugging in Appendix E.

Appendix A (For designers using Verilog)

The files provided by CIC are listed with a simple description in Table III.

Table III - Files Provided by CIC

File Name	**File Description**
`00.README`	contains file descriptions and some important notes
`testfixture.v`	test bench for verifying your design
`CS.v`	I/O declarations of the design RTL code
`synopsys_dc.setup`	environment setup for Synopsys Design Compiler
`CS.sdc`	timing constraints for Synopsys Design Compiler
`in.dat`	input patterns
`out_golden.dat`	golden output patterns
`report.000`	report form
`report.tem`	template of the report

Several rules and tips of using these files are stated below. Please read them carefully before you start to design.

- *When you write your RTL code*

The file name of the RTL code should be "`CS.v`" and the names of the top module I/O ports should be the same as those defined in the file "`CS.v`" CIC provides. Otherwise, your design will not be able to be recognized by the test bench.

- *When you verify your RTL code and gate-level netlist*

Please use "`testfixture.v`" as the test bench to verify your design. In this test bench, 2000 inputs (`in.dat`) are provided and the corresponding golden outputs (`out_golden.dat`) are compared to verify the correctness of your design. **For gate-level simulation, remember to modify the parameter "`CYCLE`" to your target clock period. Also remember to modify the parameter "`SDFFILE`" to the name of your SDF file.** After simulation is finished, if your design meets the specification, a "`PASS`" message will be shown on the display. Else, if there's something wrong in your design, the number and value of the unexpected outputs and their corresponding expected values will be displayed. Note that besides the released test patterns, CIC will also verify your design using other unreleased test patterns to further confirm the correctness.

- *When you synthesize your design*

For the fairness of the contest, the environment setup and design constrains for the synthesizer should be almost the same.

For the environment setup, please use the given file "`synopsys_dc.setup`" as the template. **Modify the "`<Your_Design_Kit_Path>`" keyword in this file to the path of the installed directory of CIC 0.18um UMC/Artisan cell-based design kit and copy this file as "`.synsopsys_dc.setup`" in your home and working directory.** Then Synopsys Design Compiler will set the environment according to it when starting up. Any modifications to this file except mentioned above are illegal. Your design may be treated as not correct due to these illegal conditions. Attempts to change the important settings in this file are also illegal. For example, try to override the link library setting with some related commands. The important settings that are not allowed to change are listed below.

```
Link library:        slow.db fast.db dw_foundation.sldb
Target library:      slow.db fast.db
```

For the timing constraints, please use "`CS.sdc`" as the template. **Modify the parameters, "`cycle`", "`t_in`", and "`t_out`" to the target clock period and reasonable I/O delays.** And then load the constraints. There are two ways to load the constraints into Synopsys Design Compiler. One is GUI method. This is done by clicking "`File/Execute Script`" item in Design Vision menu banner. The other is command method. This is done by executing the command "`source CS.sdc`" in dc shell. Any modifications except mentioned above are illegal. Your design may be treated as not correct due to these illegal conditions. Attempts to change the important settings in this file are also illegal. For example, try to override the output load setting with some related commands. The important settings that are not allowed to change are listed below.

```
Clock uncertainty:              0.1ns
Clock latency:                  0.5ns
Working environment:            slow fast
Wire load model:                umc18_wl10
All input drive (except clk):   1ns/pf
All output load:                1pf
```

- *When you finish your design*

Once you finish your design, you have to fill in the fields in "`report.xxx`". Referees will check the correctness of your design according to the materials you hand in. Refer to "`report.tem`" if you have any difficulties filling in the fields.

Appendix B (For designers using VHDL)

The files provided by CIC are listed with a simple description in Table IV.

Table IV - Files Provided by CIC

File Name	**File Description**
`00.README`	contains file descriptions and some important notes
`testfixture.v`	test bench for verifying your design
`CS.vhd`	I/O declarations of the design RTL code
`synopsys_dc.setup`	environment setup for Synopsys Design Compiler
`CS.sdc`	design constraints for Synopsys Design Compiler
`in.dat`	input patterns
`out_golden.dat`	golden output patterns
`report.000`	report form
`report.tem`	template of the report

Several rules and tips of using these files are stated below. Please read them carefully before you start to design.

- *When you write your RTL code*

The file name of the RTL code should be "`CS.vhd`" and the names of the top module I/O ports should be the same as those defined in the file "`CS.vhd`" CIC provides. Otherwise, your design will not be able to be recognized by the test bench. **Please don't use configuration design unit in the CS.vhd in order to run VHDL and Verilog co-simulation.**

- *When you verify your RTL code and gate-level netlist*

Please use "`testfixture.v`" as the test bench to verify your design. In this test bench, 2000 inputs (`in.dat`) are provided and the corresponding golden outputs (`out_golden.dat`) are compared to verify the correctness of your design. **For gate-level simulation, remember to modify the parameter "CYCLE" to your target clock period. Also remember to modify the parameter "SDFFILE" to the name of your SDF file.** After simulation is finished, if your design meets the specification, a "`PASS`" message will be shown on the display. Else, if there's something wrong in your design, the number and value of the unexpected outputs and their corresponding expected values will be displayed. Note that besides the released test patterns, CIC will also verify your design using other unreleased test patterns to further confirm the correctness. **For VHDL simulation, NC-Sim or ModelSim is recommended since we have to run VHDL and Verilog co-simulation now.**

- *When you synthesize your design*

For the fairness of the contest, the environment setup and design constrains for the synthesizer should be almost the same.

For the environment setup, please use the given file "`synopsys_dc.setup`" as the template. **Modify the "`<Your_Design_Kit_Path>`" keyword in this file to the path of the installed directory of CIC 0.18um UMC/Artisan cell-based design kit and copying this file as "`.synsopsys_dc.setup`" in your home and working directory.** Then Synopsys Design Compiler will set the environment according to it when starting up. Any modifications to this file except mentioned above are illegal. Your design may be treated as not correct due to these illegal conditions. Attempts to change the important settings in this file are also illegal. For example, try to override the link library setting with some related commands. The important settings that are not allowed to change are listed below.

```
Link library:        slow.db fast.db dw_foundation.sldb
Target library:      slow.db fast.db
```

For the timing constraints, please use "`CS.sdc`" as the template. **Modify the parameters, "`cycle`", "`t_in`", and "`t_out`" to the target clock period and reasonable I/O delays.** And then load the constraints. There are two ways to load the constraints into Synopsys Design Compiler. One is GUI method. This is done by clicking "`File/Execute Script`" item in Design Vision menu banner. The other is command method. This is done by executing the command "`source CS.sdc`" in dc shell. Any modifications except mentioned above are illegal. Your design may be treated as not correct due to these illegal conditions. Attempts to change the important settings in this file are also illegal. For example, try to override the output load setting with some related commands. The important settings that are not allowed to change are listed below.

```
Clock uncertainty:             0.1ns
Clock latency:                 0.5ns
Working environment:           slow fast
Wire load model:               umc18_wl10
All input drive (except clk):  1ns/pf
All output load:               1pf
```

- *When you finish your design*

Once you finish your design, you have to fill in the fields in "`report.xxx`". Referees will check the correctness of your design according to the materials you hand in. Refer to "`report.tem`" if you have any difficulties filling in the fields.

Appendix C

The materials that each team should hand in are listed below.

Table V - Necessary Deliverables

RTL category		
Design Stage	*File*	*Description*
N/A	`report.xxx`	design report
RTL Simulation	`*.v or *.vhd`	Verilog (or VHDL) synthesizable RTL code
Gate-Level category		
Design Stage	*File*	*Description*
Pre-layout Gate-level Simulation	`*_syn.vg`	Verilog gate-level netlist generated by Synopsys Design Compiler
	`*_syn.sdf`	SDF timing information generated by Synopsys Design Compiler
	`*_syn.db`	design database generated by Synopsys Design Compiler

Several rules and tips of generating these files are stated below. Please read them carefully before you hand in the materials.

- *For RTL category*

Once you finish your design, you have to fill in the fields as many as possible in "`report.xxx`" of which the extension "`xxx`" indicate the revision number. Referees will check the correctness of your design according to the materials you hand in. Refer to "`report.tem`" if you have any difficulties filling in the fields.

Note that the RTL code should be synthesizable. Otherwise, the design will be treated as not correct.

- *For Gate-Level category*

The gate-level netlist should not contain any behavioral statements, for example, assign statement. Otherwise, your design will be treated as not correct. **About the SDF timing information, please use version 2.1 for pre-layout gate level simulation.**

Appendix D

All files listed in Table V have to be handed in. Before the files are transferred to CIC, all files should be archived and compressed. The following is a step-by-step illustration.

1. Create a new directory.
2. Copy all the files to be handed in except the "`report.xxx`" file into the created directory. The file extension "`xxx`" means the revision number. For example, for the first revision, the file name is "`report.000`". You may have improvements on your design later and want to re-transmit the design data. Now, the report file name should be "`report.001`" to let referees always get the most updated design data.
3. Execute the following UNIX command to create an archive. After this operation, you will get "`your_username_xxx.tar`". The purpose of using "`xxx`" postfix is the same as that of using "`xxx`" extension for report file.
   ```
   > tar cvf your_username_xxx.tar *
   ```
4. Execute the following UNIX command to compress the archive. After this operation, you will get "`your_username_xxx.tar.Z`".
   ```
   > compress your_username_xxx.tar
   ```
5. Check if all necessary fields of the report form (`report.xxx`) are filled in.
6. Transfer the compressed file "`your_username_xxx.tar.Z`" and the report file "`report.xxx`" via ftp with binary mode to the following ftp sites. The username and password will be given 4 days before of the contest via email. Any problems, please contact CIC.

   FTP site1 (NTU)：iccftp.ee.ntu.edu.tw (140.112.20.85)
   FTP site2 (CIC)：iccftp.cic.org.tw (140.126.24.6)
   FTP site3 (NCKU)：iccftp.ee.ncku.edu.tw (140.116.156.55)
   FTP site 4(CIC South region office): iccftp1.cic.org.tw(140.110.117.20)

7. Repeat the above steps if you have updated design. Remember to change the extension of the report file name as well as the postfix of the compressed file name to indicate the revision number. Also remember that the content of the report should be updated.

Appendix E

An example is shown for debugging in the next pages.

(Hexadecimal number)

index i	X_i	$Xavg_j$	$Xappr_j$	$\sum_{i=j}^{j+n-1}(X_i + Xappr_j)$	Y_j	*index j*
1	8f					
2	0b					
3	5d					
4	20					
5	f3					
6	3e					
7	e5					
8	03					
9	0c	5c	3e	056a	00ad	1
10	74	59	3e	054f	00a9	2
11	79	65	5d	06d4	00da	3
12	01	5b	3e	0561	00ac	4
13	30	5c	3e	0571	00ae	5
14	2e	46	3e	04ac	0095	6
15	a4	52	30	0494	0092	7
16	76	45	30	0425	0084	8
17	84	54	30	04a6	0094	9
18	51	5b	51	0614	00c2	10
19	d6	66	51	0676	00ce	11
20	70	65	51	066d	00cd	12
21	35	6b	51	06a1	00d4	13
22	10	68	51	0681	00d0	14
23	23	66	51	0676	00ce	15
24	e7	6e	51	06b9	00d7	16
25	3b	67	51	067e	00cf	17
26	6d	65	51	0667	00cc	18
27	34	61	3b	0584	00b0	19
28	61	54	3b	050f	00a1	20
29	89	57	3b	0528	00a5	21
30	bf	67	61	0708	00e1	22
31	dc	7d	6d	0840	0108	23
32	d3	91	89	09ec	013d	24
33	9c	88	6d	08a5	0114	25
34	8f	92	8f	0a2b	0145	26

(Decimal number)

index i	X_i	$Xavg_j$	$Xappr_j$	$\sum_{i=j}^{j+n-1}(X_i + Xappr_j)$	Y_j	*index j*
1	143					
2	11					
3	93					
4	32					
5	243					
6	62					
7	229					
8	3					
9	12	92	62	1386	173	1
10	116	89	62	1359	169	2
11	121	101	93	1748	218	3
12	1	91	62	1377	172	4
13	48	92	62	1393	174	5
14	46	70	62	1196	149	6
15	164	82	48	1172	146	7
16	118	69	48	1061	132	8
17	132	84	48	1190	148	9
18	81	91	81	1556	194	10
19	214	102	81	1654	206	11
20	112	101	81	1645	205	12
21	53	107	81	1697	212	13
22	16	104	81	1665	208	14
23	35	102	81	1654	206	15
24	231	110	81	1721	215	16
25	59	103	81	1662	207	17
26	109	101	81	1639	204	18
27	52	97	59	1412	176	19
28	97	84	59	1295	161	20
29	137	87	59	1320	165	21
30	191	103	97	1800	225	22
31	220	125	109	2112	264	23
32	211	145	137	2540	317	24
33	156	136	109	2213	276	25
34	143	146	143	2603	325	26