Term Project

Due: Deadline 06/01/2021, 17:00 (**SHARP**)
Demo Sessions 06-08/01/2021

A Simple Telephone Conversation

You will design a sequential circuit for a simple two-sided telephone conversation and implement it using Verilog HDL. Then, you will simulate and show its functional correctness using Xilinx ISE tool. Basically, the caller will initiate a telephone conversation with the callee and the caller will send characters to the callee (and vice versa). Your circuit will calculate the cost of the call and send the cost value and the characters (sent from the caller to callee and vice versa) as outputs.

Inputs

There will be 8 inputs in your circuitry:

- *rst* will set your circuitry to its initial state.
- *startCall* (1-bit) will be used by the caller and it will represent that the caller pressing a button to start a call.
- *answerCall* (1-bit) will be used by the callee and it will represent that the callee pressing a button to answer an incoming call.
- *callerEndsCall* (1-bit) will be used by the caller and it will represent that the caller pressing a button to end an ongoing/outgoing call.
- *calleeEndsCall* (1-bit) will be used by the callee and it will represent that the callee pressing a button to end an ongoing/incoming call.
- *charSent* (8-bit) will be used to define 8-bit printable ASCII character to be sent from the caller to the callee (and vice versa) according to printable ASCII table shown at Fig. 1.
- *callerSendsChar* (1-bit) will be used by the caller and it will represent that the caller pressing a button to send an ASCII character (set by *charSent* input) to the callee.
- *calleeSendsChar* (1-bit) will be used by the callee and it will represent that the callee pressing a button to send an ASCII character (set by *charSent* input) to the callee.

Outputs

There will be 2 outputs in your circuitry:

- *statusMsg* (64-bit) will be used to display the status of telephone using 8 printable ASCII character. For example, *statusMsg* output should be "`IDLE  `" in its initial state. More details will be provided below.
- *sentMsg* (64-bit) will be used to display the last 8 printable ASCII characters sent by caller or callee. More details will be provided below.

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
32	20	[SPACE]	64	40	@	96	60	`
33	21	!	65	41	A	97	61	a
34	22	"	66	42	B	98	62	b
35	23	#	67	43	C	99	63	c
36	24	$	68	44	D	100	64	d
37	25	%	69	45	E	101	65	e
38	26	&	70	46	F	102	66	f
39	27	'	71	47	G	103	67	g
40	28	(	72	48	H	104	68	h
41	29	)	73	49	I	105	69	i
42	2A	*	74	4A	J	106	6A	j
43	2B	+	75	4B	K	107	6B	k
44	2C	,	76	4C	L	108	6C	l
45	2D	-	77	4D	M	109	6D	m
46	2E	.	78	4E	N	110	6E	n
47	2F	/	79	4F	O	111	6F	o
48	30	0	80	50	P	112	70	p
49	31	1	81	51	Q	113	71	q
50	32	2	82	52	R	114	72	r
51	33	3	83	53	S	115	73	s
52	34	4	84	54	T	116	74	t
53	35	5	85	55	U	117	75	u
54	36	6	86	56	V	118	76	v
55	37	7	87	57	W	119	77	w
56	38	8	88	58	X	120	78	x
57	39	9	89	59	Y	121	79	y
58	3A	:	90	5A	Z	122	7A	z
59	3B	;	91	5B	[	123	7B	{
60	3C	<	92	5C	\	124	7C	|
61	3D	=	93	5D	]	125	7D	}
62	3E	>	94	5E	^	126	7E	~
63	3F	?	95	5F	_	127	7F	[DEL]

Fig.1. Printable ASCII Characters

Operation Steps

The circuitry will start in the IDLE state, in which it should output '`IDLE    `' to *statusMsg* output (last 4 characters are space). You should use *rst* input as an asynchronous reset input to put the sequential circuit into IDLE state. In IDLE state, the caller can initiate a call by pressing *startCall* (by bringing *startCall* to 1 for one clock cycle – see **HINT#1** for how to model this in testbench) when the circuit is in IDLE state. Otherwise, the circuit will stay in IDLE state. If the caller initiates a call by pressing *startCall*, the telephone will start ringing. When the telephone is ringing, the 8-bit ASCII output "`RINGING  `" should be at output *statusMsg*. When the telephone is ringing, there are four possibilities:

1) If the caller hangs up by pressing *callerEndCall*, your circuit should go back to the IDLE state.
2) If the callee rejects the call by pressing *calleeEndCall*, your circuit should output '`REJECTED`' to *statusMsg* output for 10 clock cycles (see **HINT#2**) and then your circuit should go back to the IDLE state.
3) If the callee neither answers or rejects the call and the caller does not hang up for 10 clock cycles, your circuit should go to IDLE state.
4) If the callee answers the call and the conversation starts, there are four possibilities during the conversations:
    a. The caller sends character to the callee by setting *charSent* and pressing *callerSendsChar*. If the caller does not press *callerSendsChar*, telephone takes no input. During the caller sending character to the callee, *statusMsg* output should be "`CALLER  `". Also, *sentMsg* output should be the last 8 ASCII characters sent by the caller. For example, if caller sends characters "C", "S",

"3, "0", "3", then *sentMsg* should be " CS303". Another example is shown below.

sentMsg [7]	*sentMsg* [6]	*sentMsg* [5]	*sentMsg* [4]	*sentMsg* [3]	*sentMsg* [2]	*sentMsg* [1]	*sentMsg* [0]	Sent Char.	ASCII (Dec.)
							C	C	67
						C	S	S	83
					C	S	3	3	51
				C	S	3	0	0	48
			C	S	3	0	3	3	51
		C	S	3	0	3		(SPACE)	32
		C	S	3	0	3		No input	
		C	S	3	0	3		No input	
		C	S	3	0	3		Invalid I.	16
	C	S	3	0	3		P	P	80
C	S	3	0	3		P	r	r	114
S	3	0	3		P	r	o	o	111
3	0	3		P	r	o	j	j	106
0	3		P	r	o	j	e	e	101
3		P	r	o	j	e	c	c	99
	P	r	o	j	e	c	t	t	116

Note that only ASCII characters with decimal values between 32 and 126 are considered valid input characters. If the caller enters and sends an invalid character, your circuit should ignore that character (do not include this input in cost calculation). If the caller sends the ASCI character with decimal value 127 (DEL) (include this input in cost calculation), it will be callee's turn to send characters.

b. If caller sends the ASCII character with decimal value 127 (DEL), it is callee's turn and the callee can send character to the callee by setting *charSent* and pressing *calleeSendsChar*. During the caller sending character to the callee, *statusMsg* output should be "CALLEE ". The same rules stated at part (a) applies to the callee as well.
c. The caller ends the call at any time by pressing on *callerEndsCall*.
d. The callee ends the call at any time by pressing on *calleeEndsCall*.

Each sent character costs 2 Krş except for integer digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) which cost 1 Krş. Your circuit should calculate total cost of a call. When the call is ended, total cost of the conversation will be sent as an output to *sentMsg* output in hexadecimal and *statusMsg* output should be "COST " for 5 clock cycles. For example, a call with a cost of 55 Krş should be shown on *sentMsg* as "00000037" for 5 clock cycles. Then, your circuit should go to IDLE state.

Zip your project directory (as much as you've done) and submit your zip file through SUCourse+ with your report which should include:

- An overview of your design with your states
- Simulation results showing that your design is working
- Synthesis results (not implementation)

Appendix A: An Example Template

```
module tel (input clk,
            input rst,
            input startCall, answerCall,
            input endCallCaller, endCallCallee,
            input sendCharCaller, sendCharCallee,
            input [7:0] charSent,
            output reg [63:0] statusMsg,
            output reg [63:0] sentMsg,
           );

     reg [31:0] cost;
     reg [7:0] current_state;
     reg [7:0] next_state;
    …                          // additional registers

    // sequential part – state transitions
    always @ (posedge clk or posedge rst)
    begin
         …          // your code goes here
    end

    // combinational part – next state definitions
    always @ (*)
    begin
         …          // your code goes here
    end

    // sequential part – control registers
    always @ (posedge clk or posedge rst)
    begin
         …          // your code goes here
    end

    // sequential part – outputs
    always @ (posedge clk or posedge rst)
    begin
         …          // your code goes here
    end

    …          // additional always statements

endmodule
```

Please note that this template is only an example. You can add more registers, change the bit-size of the given registers, initialize the values for the given registers, define parameters and extend the number of "always" statements as your design requires.

Appendix B: Hints

- **HINT#1:** Pressing a button (or an input) is modelled as bringing signal to 1 for one period and then lowering to 0. For example, pressing an input *A* can be modelled as (assuming clock period is 10 unit):

```
A = 1;
#10;
A = 0;
#10;
```

- **HINT#2:** Use a counter counting from 0 to 9.

- **HINT#3:** For sending ASCII characters as an input in Verilog testbench, you can either send its decimal value or ASCII character as a Verilog string. For example, ASCII character C can be assigned to a variable as in two different ways:

```
A = 67;
A = "C";
```

For observing ASCII characters in ISim simulator, you can "Radix" of variable as "ASCII" as shown in Fig. 2.

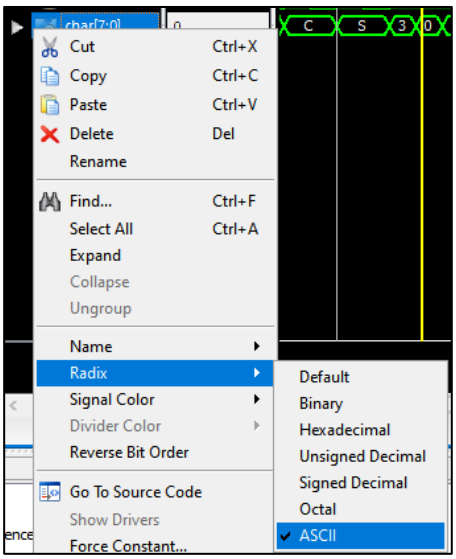

Fig. 2. Changing Radix of a Signal to ASCII

Appendix C: Deadline and Demo

You must follow the project plan given below and demonstrate that you meet a specific deadline by submitting your work **on time**. A demo schedule will be organized before the demo dates and will be announced through SUCourse+. If you want to demonstrate your design before the deadlines, you can do so by scheduling an appointment with your TA or you can use office hours.

1. REQUIREMENTS: Project requirements are given to the students.
   Dec 22, 2020

2. DEADLINE: You should zip your project directory and project report, and submit it to SUCourse+.
   Jan 06, 2021 17:00

3. SIMULATION DEMO: ISim simulation is demonstrated to the lab assistants.
   Jan 06, 2021 – Jan 08, 2021

Note that these deadlines are hard, and there will be no additional time.

Some Tips: Before writing your project on Verilog, you can try to write some small examples to warm up. This will help you to complete your task quicker and painless. Please see recitation notes (Verilog-I/II/III) Also, there some websites that you can find some examples to study Verilog. One of them is:
http://www.asic-world.com/verilog/veritut.html. These sites might help you to understand combinational, sequential circuits and state diagrams.

Note:

- We will provide you simple simulation scenario for you to test your design. However, you should also test your design with your own scenario.

Appendix D: Example Simulation Scenario

During the demonstration, you will be given a simulation scenario and you will be asked to demonstrate the correct transitions between the states and outputs of your circuit. We provide an example simulation scenario shown below for you. This scenario is written as a Verilog testbench (**tel_tb.v**) and provided to you under "**Lab Materials→ Test files**". Please note that you may need to modify this testbench based on your variable/module names.

In order to observe your state transition, you should be able to see your state signals. Please read "**Verilog-Testbench**" document under **Lab_Materials** carefully for learning how to debug (observe internal signals of your design) your Verilog code.

Simulation scenario:

a. Reset the circuit
b. Caller starts the call by pressing *startCall*
c. No answer for 10 clock cycles and go back to the IDLE state
d. Caller starts the call by pressing *startCall*
e. Caller ends the call by pressing *endCallCaller*
f. Caller starts the call by pressing *startCall*
g. Callee rejects the call by pressing *endCallCallee*
h. Caller starts the call by pressing *startCall*
i. Caller sends "T" by setting *charSent* as "T" and pressing *sendCharCaller*
j. Caller sends "E" by setting *charSent* as "E" and pressing *sendCharCaller*
k. Caller sends "R" by setting *charSent* as "R" and pressing *sendCharCaller*
l. Caller sends "M" by setting *charSent* as "M" and pressing *sendCharCaller*
m. Caller sends " " by setting *charSent* as " " and pressing *sendCharCaller*
n. Caller sends invalid character by setting *charSent* as 12 and pressing *sendCharCaller*
o. Caller sends "P" by setting *charSent* as "P" and pressing *sendCharCaller*
p. Caller sends "R" by setting *charSent* as "R" and pressing *sendCharCaller*
q. Caller sends "O" by setting *charSent* as "O" and pressing *sendCharCaller*
r. Caller sends "J" by setting *charSent* as "J" and pressing *sendCharCaller*
s. Caller sends "E" by setting *charSent* as "E" and pressing *sendCharCaller*
t. Caller sends "C" by setting *charSent* as "C" and pressing *sendCharCaller*
u. Caller sends "T" by setting *charSent* as "T" and pressing *sendCharCaller*
v. Caller sends DEL by setting *charSent* as 127 and pressing *sendCharCaller* and callee takes turn for sending character
w. Callee sends "C" by setting *charSent* as "C" and pressing *sendCharCallee*
x. Callee sends "S" by setting *charSent* as "S" and pressing *sendCharCallee*
y. Callee sends "3" by setting *charSent* as "3" and pressing *sendCharCallee*
z. Callee sends "0" by setting *charSent* as "0" and pressing *sendCharCallee*
aa. Callee sends "3" by setting *charSent* as "3" and pressing *sendCharCallee*
bb. Callee ends the call by pressing *endCallCallee*
cc. *sentMsg* output displays "00000021"

Try to simulate your circuit with different scenarios and make sure that your circuit is working correctly.