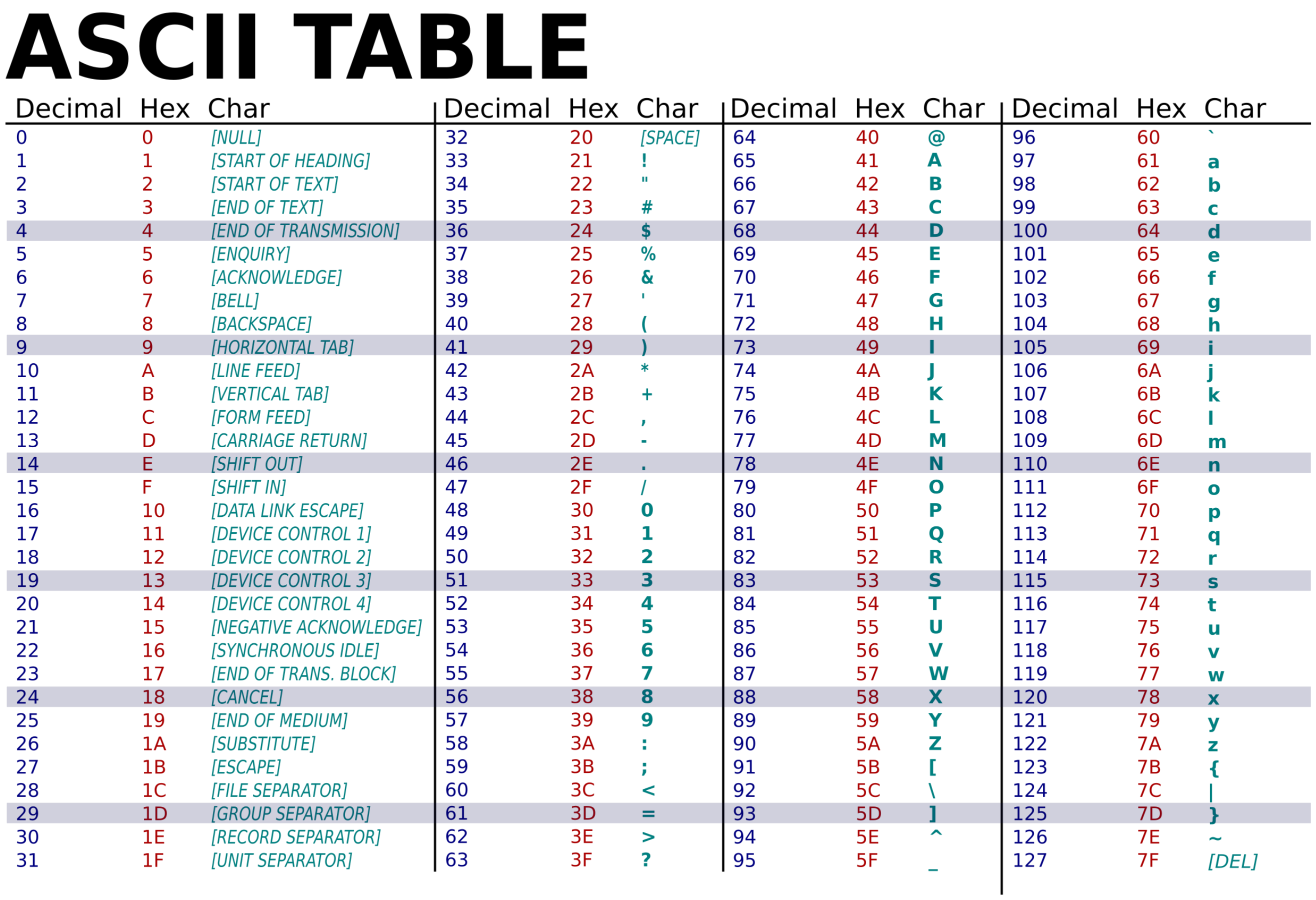
Lab 20: Design an UART Transmitter

In this lab, we will design a UART Transmitter module, and check the results using the a remote serial console on our laptop.

The design has the following port Interface

Port	Direction	Type	Width
CLK	In	std_logic	1
BTN	IN	std_logic_vector	5
UART_TXD	OUT	std_logic	1

Depending on which button is pressed, the design will transmit a different string on the `UART_TXD`. Each character in the string is represented by an 8-bit word, using the ASCII notation.


ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Exercise 1. Design a UART Transmitter

The first step is to design a UART Transmitter module. Go to `~/labs/lab19/` and open `src/UART_TX.vhd` with a text editor.

```
kate src/UART_TX.vhd &
```

The module should have the following ports and generics

Generic	Type	Default
BAUDRATE	integer	96000
CLKRATE	integer	100000000

Port	Direction	Type	Width
CLK	In	std_logic	1
DATA	IN	std_logic_vector	8
DV	IN	std_logic	1
BUSY	OUT	std_logic	1
TX_OUT	OUT	std_logic	1
DONE	OUT	std_logic	1

- The `BAUDRATE` is the number of symbols transferred per seconds.
- `DATA` is the 8-bit word to transmit.
- `DV` is the data valid bit associated to the `DATA`.
- `BUSY` signals whether the module can accept or not input data.
- `TX_OUT` is the output UART serial line.
- `DONE` tells that the module has finished transmitting data.

To implement the UART Transmitter, you have to do the following.

- Calculate a constant `CLKS_PER_BIT`, which is equal to the ratio between the system clock and the baud rate.
- Create a FSM with the following states: `IDLE`, `START`, `SEND_DATA`, `STOP`
  - When `IDLE`, the module should send `1` on `TX_OUT` (Refer to the lesson for the UART protocol). `BUSY` and `DONE` should be 0. If `DV = 1`, copy the `Data` in a local signal `copy_data` and move to the `START` state.
  - When `START`, `TX_OUT` should go low for `CLKS_PER_BIT`, and then move to the `SEND_DATA` state. `BUSY` should be driven high.
  - When in `SEND_DATA`, a counter should be increased every `CLKS_PER_BIT`. `TX_OUT` should get the bit-value of `local_data` at the current counter. When you have sent all 8 bits, reset the counter and go to `STOP` state.
  - In `STOP`, the `TX_OUT` output should be driven back to 1 for `CLKS_PER_BIT`. After that, `DONE` goes high and move back to `IDLE` state.

N.B. You might need different counters for the UART frames and the bit indexes for the data.

Run the simulation script, to check your module implementation.

```
./run_uart_sim.sh
```

Exercise 2. Design the Top module

a. Create the Vivado project

1. Go to `~/labs/lab20` and start Vivado
2. Create a new Vivado project, called `UART` (RTL Project)
3. In the `Add Sources` window, click on `Add Files` and import `src/UART_TX.vhd`, `src/deb.vhd`, `src/multiple_debouncer.vhd` and `src/top_uart.vhd`
4. In the `Add Constraints` window, click on `Add Files` and import `src/Basys3_Master.xdc`
5. In the `Add Simulation` window, click on `Add Files` and import `sim/tb_top_uart.vhd`
6. In the `Default Part` select the `Basys3` from the Boards tab.
7. Click on Finish

b. Design the top module

Open the `src/top_uart.vhd` and implement the following functionalities, following the comments in the file.

1. Define a new type `t_word`, which is an array of `std_logic_vector(7 downto 0)` with an open range

```
type t_word is array (integer range<> of std_logic_vector(7 downto 0));
```

2. Define five constants of type `t_word`. Each element in the array will correspond to an ASCII character to be transmitted via UART. You can decide which string to print. Here there is an example.

```
constant BASYS3_STR : t_word(0 to 26) := (X"0A",  --\n
                                                   X"0D",  --\r
                                                   X"42",  --B
                                                   X"41",  --A
                                                   X"53",  --S
                                                   X"59",  --Y
                                                   X"53",  --S
                                                   X"33",  --3
                                                   X"20",  --
                                                   X"47",  --G
                                                   X"50",  --P
                                                   X"49",  --I
                                                   X"4F",  --O
                                                   X"2F",  --/
                                                   X"55",  --U
                                                   X"41",  --A
                                                   X"52",  --R
                                                   X"54",  --T
                                                   X"20",  --
                                                   X"44",  --D
                                                   X"45",  --E
                                                   X"4D",  --M
                                                   X"4F",  --O
```

```
                                                            X"21",  --!
                                                            X"0A",  --\n
                                                            X"0A",  --\n
                                                            X"0D"); --\r
```

Bonus: Instead of creating five constants, you could create a new array type `t_words`, which is an array of `t_word` and create a single constant `words` of type `t_words`, where each element is a word that you define. 3. Create a new enumerated type for the top FSM, with states `IDLE` and `SEND_DATA`. 4. Instantiate the `mult_debouncer` module, to debounce the five push buttons. Declare a `debounced_btns` signal that must be connected to the module. 5. Instatiate the `UART_TX` module, you made in the previous exercise. Declare the missing signals accordingly. 6. Write the synchronous process, that defines the behaviour of the state machine.

1. When in `IDLE` state, wait for one the bit in the `debounced_btns` to go high. Copy `debounced_btns` to a `command` signal and move to the `SEND_DATA` state.
2. When in the `SEND_DATA` state, check whether the `UART_TX` module is busy. If busy, set the UART `dv` low. If not, use a case condition on the `command` signal, to send characters of a word, to the `data` port of `UART_TX`, using the constants you defined before. Do not forget to set the UART `dv` signal high. Consider also the case when you push more than one button at the same time (`when others`). E.g.

```
case command is
 when "00001" =>
   uart_data <= WORD0(string_index);
   uart_dv   <= '1';
 when "00010" => ....
 when others => WORD0(string_index);
```

```
If the `string_index` counter is less than then the length of the word
array, increase it every time the UART `done` goes high. Otherwise, reset
the counter and set `dv` low, and go back to the `IDLE` state.
```

When you are done, run the simulation, clicking on `Run Simulation` on the left sidebar, to validate your design.

c. Generate the bitstream

If everything goes well, generate the bitstream and load it to the Basys3 board.

d. Test the design

Open another terminal window on your laptop. Launch a `minicom` serial terminal on port `/dev/ttyUSB1`, using the baudrate you implemented in your design. E.g for `baudrate=9600`.

```
minicom -D /dev/ttyUSB1 -b 9600
```

Try now to push the buttons on the Basys3. If the design is correctly implemented, you should see the words you defined earlier printed on the screen.

You can close the terminal, typing `CTRL-A` and `X`.