Laboratory 11

Finite State Machines and Serial Communication

1. Objectives

Study, design, implement and test

- **Finite State Machines**
- **Serial Communication**

Familiarize the students with

- Xilinx® ISE Webpack
- Digilent Development Boards (DDB)
  - ➢ **Digilent Basys Board – Reference Manual**

2. Theoretical Background

2.1. Finite State Machines

A finite state machine or FSM is a model of behavior composed of a finite number of states, transitions between those states, and actions. A finite state machine is used to describe an abstract model of a control unit. XST proposes a large set of templates to describe FSMs. By default, XST tries to distinguish FSMs from VHDL or Verilog code, and apply several state encoding techniques to obtain better performance or less area.

XST supports the following state encoding techniques:

- Auto – the best suited encoding algorithm for each FSM.
- One-hot – associate one code bit and one flip-flop per state. At a given clock cycle during operation, one and only one bit of the state variable is asserted. Only two bits toggle during a transition between two states. One-hot encoding is appropriate with most FPGA targets where a large number of flip-flops are available. It is also a good alternative when trying to optimize speed or to reduce power dissipation.
- Gray – guarantees that only one bit switches between two consecutive states. It is appropriate for controllers exhibiting long paths without branching. In addition, this coding technique minimizes hazards and glitches. Very good results can be obtained when implementing the state register with T flip-flops.

- Compact – consists of minimizing the number of bits in the state variables and flip flops. Compact encoding is appropriate when trying to optimize area.
- Johnson – like Gray, it shows benefits with state machines containing long paths with no branching.
- Sequential – consists of identifying long paths and applying successive radix two codes to the states on these paths. Next state equations are minimized.
- Speed1 – oriented for speed optimization. The number of bits for a state register depends on the particular FSM, but generally it is greater than the number of FSM states.
- User – original encoding, specified in the HDL file.

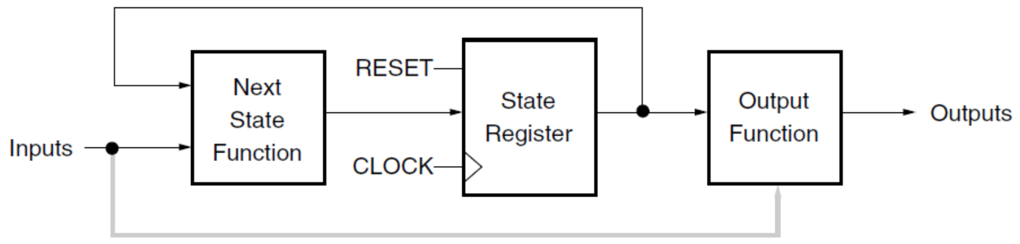


Figure 1: FSM Representation Incorporating Mealy and Moore Machines

When describing a finite state machine in VHDL you may have several processes (1, 2 or 3) depending upon how you consider and decompose the different parts of the preceding model. Appendix 7 describes the VHDL finite state machine implementations.

2.2. Serial Communication – UART

Serial communication is basically the transmission or reception of data one bit at a time. Today's computers generally address data in bytes or some multiple thereof. A serial port is used to convert each byte to a stream of ones and zeroes as well as to convert streams of ones and zeroes to bytes. The serial port contains an electronic chip called a Universal Asynchronous Receiver/Transmitter (UART) that actually does the conversion. Serial transmission of digital information (bits) through a single wire or other medium is much more cost effective than parallel transmission through multiple wires.

When transmitting a byte, the UART first sends a START BIT followed by the data (general 8 bits, but could be 5, 6, 7, or 8 bits), followed by STOP BITs. The sequence is repeated for each byte sent.

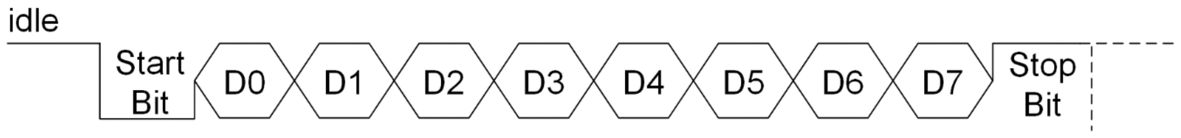
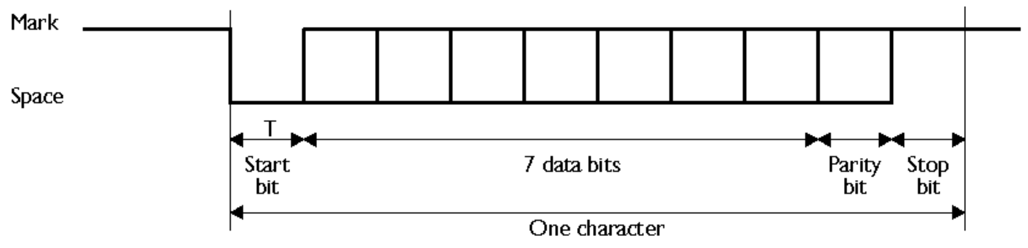


Figure 2: Serial Transmission Timing Diagram

Serial transmission does not involve a clock signal. The information is included in the baud rate (**number of bits per second**). Common baud rates are 2400, 4800, 9600 and 19200. This means that a bit transmitted through the serial line is valid for a given time period (the inverse of the baud rate).

The start bit is always 0, the data bits are transmitted with the LSB (least significant bit) first and MSB (most significant bit) last and the stop bit is always 1. In serial communication the stop bit duration can have multiple values: 1, 1.5 or 2 bit periods in length. Besides the synchronization provided by the use of start and stop bits, an additional bit called a parity bit may optionally be transmitted along with the data. A parity bit affords a small amount of error checking, to help detect data corruption that might occur during transmission. One can choose even parity, odd parity, mark parity, space parity or none at all. When even or odd parity is being used, the number of marks (logical 1 bits) in each data byte is counted, and a single bit is transmitted following the data bits, to indicate whether the number of 1 bits just sent is even or odd.

The data sent through serial communication is encoded using ASCII codes (Appendix 8). Assume we want to send the letter 'A' over the serial communication channel. The binary representation of the letter 'A' is 01000001 (41h). Remembering that bits are transmitted from least significant bit (LSB) to most significant bit (MSB), the bit stream transmitted would be as follows for the line characteristics 8 bits, no parity, 1 stop bit, 9600 baud: **LSB (0 1 0 0 0 0 0 1 0 1) MSB**. This represents (Start Bit) (Data Bits) (Stop Bit). For a binary two-level signal, a data rate of one bit per second is equivalent to one Baud. To calculate the actual byte transfer rate simply divide the baud rate by the number of bits that must be transferred for each byte of data. In the case of the above example, each character requires 10 bits to be transmitted for each character. As such, at 9600 baud, up to 960 bytes can be transferred in one second.



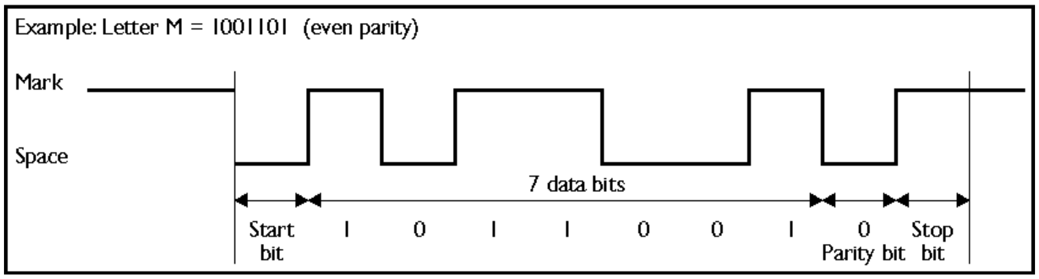


Figure 3: Serial Transmission Example (7 data bits)

For accurate serial communication, on the receiving end, an oversampling scheme is commonly used to locate the middle position of the transmitted bits, i.e., where the actual sample is taken. The most common oversampling rate is 16 times the baud rate. Therefore, each serial bit is sampled 16 times but only one sample is saved.

Each UART contains a shift register which is the fundamental method of conversion between serial and parallel forms.

3. Laboratory Assignments

3.1. Pmod USB-UART

Read the Pmod USB-UART reference manual. The figure below shows the connection of the USB-UART peripheral module to the FPGA board.

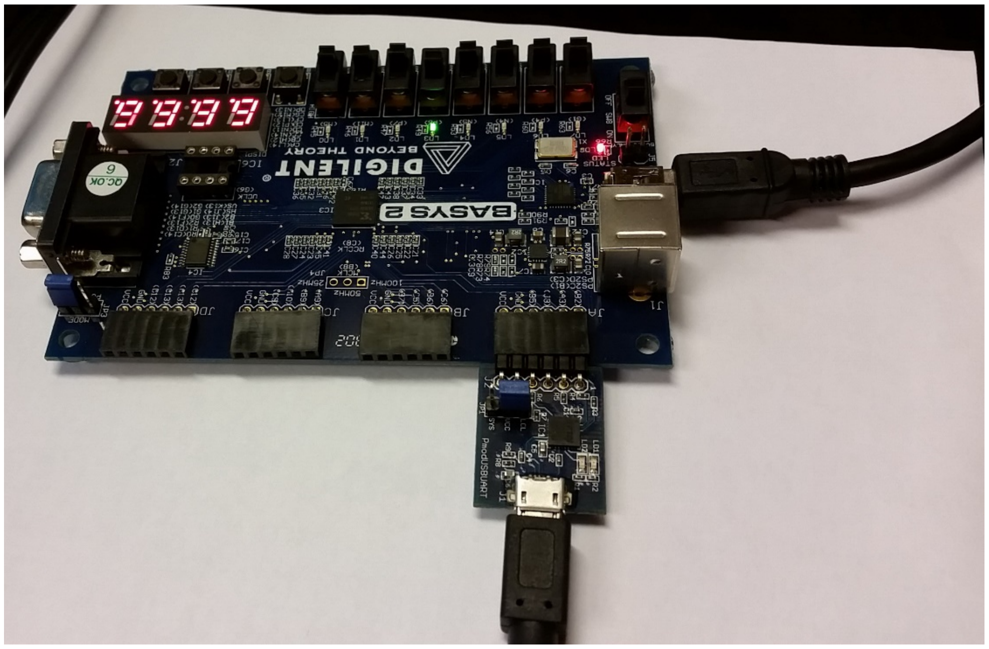


Figure 4: Pmod USB-UART connection to the FPGA board

Use the USB-Mini USB cable to power the board and the USB-Micro USB cable for serial data communication.

Download and open the HTERM terminal program. Alternatively you can use the hyper terminal software available in windows XP.

You need to define the RX (input) and TX (output) ports into your "test_env" project and in the UCF file. Use your board's reference manual to locate the correct pin numbers. Attention: The TX of the FPGA board is the RX of the Pmod USB-UART module and the RX of the FPGA board is the TX of the Pmod USB-UART module.

3.2. Serial Transmit FSM

Design a baud rate generator that would ensure a 9600 baud rate (9600 bits per second) communication over the serial cable. Use a counter to generate the BAUD_ENable signal (generate a '1' every bit time interval).

Baud rate generation:

- For 25 MHz, clock period ~40 ns, input clock must be divided by 2604.
- For 50 MHz, clock period ~20 ns, input clock must be divided by 5208.
- For 100 MHz, clock period ~10 ns, input clock must be divided by 10416.

Define a new entity for the transmission FSM. The next figure presents the ports of this entity.

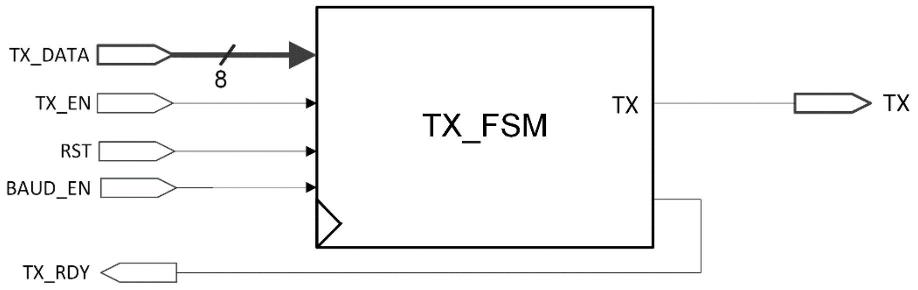


Figure 5: TX_FSM Entity Description

The detailed FSM implementation is presented in the figure below. A state transition is triggered only in the clock cycle when BAUD_ENable is '1'. This ensures that a bit will be valid for the baud rate period. The BIT_CNT is a signal with the functionality of a counter inside the FSM; it holds the current transmitting bit value. It should be incremented in the bit state and should be reset after each serial transfer (you can do that in the idle state, or in all states except the bit state).

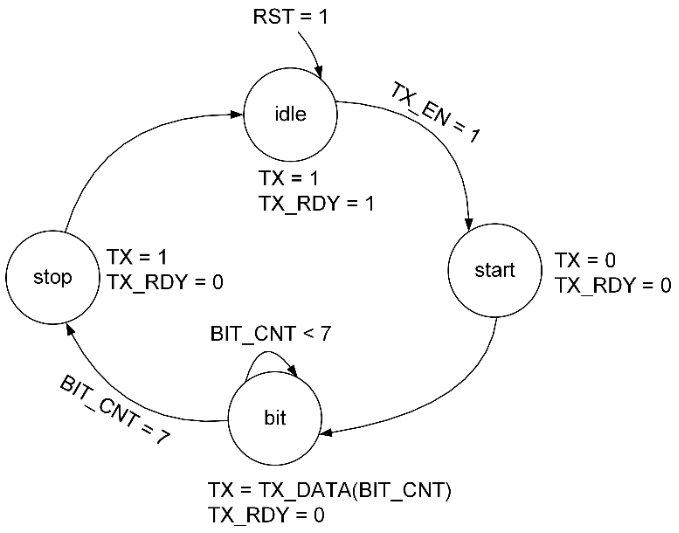


Figure 6: TX_FSM Implementation

Write the VHDL code and implement in the “test_env” project the TX_FSM state machine. Use a FSM with 2 or 3 processes (see appendix 7). Test the communication between the FPGA board and the PC. The parameters of the serial communication are: 1 start bit, 8 data bits, 1 stop bit, no parity bit, 9600 baud rate. Make sure that these settings are also configured in the HTERM / hyper terminal application.

In order to test the serial transmission from the FPGA board to the PC, connect the TX_DATA input to the switches, the TX_EN signal to a MPG enable, RST to ‘0’ or another MPG enable. Make sure that the switches show a valid ASCII code.

Define the correct methodology of asserting the TX_EN signal in order to initiate a single serial data transfer (use a D flip-flop with a set and a reset).

3.3. I/O from the MIPS CPU

Connect the TX_FSM into your own MIPS processor implementation. At this point you are allowed to use your finished and complete processor (single-cycle or pipeline).

You have to send 16-bits of data from your MIPS processor to the PC. Depending on the result of your program define what field you will send (register with the final result, memory location, etc.).

Example:
When your program has finished execution the result is in R7 and the PC is 0x0020. Add a new instruction to your program: addi R7, R7, 0. Define a 16-bit register whose value will be written from the RD1/ALURes signal, write it in this register (write enable with the value of the PC) and initiate the serial transfer.

Remember that when sending over the serial line the 8-bits represent an ASCII character, hence you are required to make 4 transfers in order to send the alphanumerical encoding of the 4 x 4-bit hexadecimal value (use a decoder/ROM to generate the 8-bit ASCII representation for a hexadecimal value).

Define the methodology to send the 16-bit data over the serial line. Use the TX_RDY signal to control the 4 serial transfers.

4. References

- XST User Guide
- Digilent Basys Board – Reference Manual
- Digilent Pmod USB-UART – Reference Manual
- http://www.asciitable.com/
- http://www.der-hammer.info/terminal/

Appendix 7 – Finite State Machine Implementations

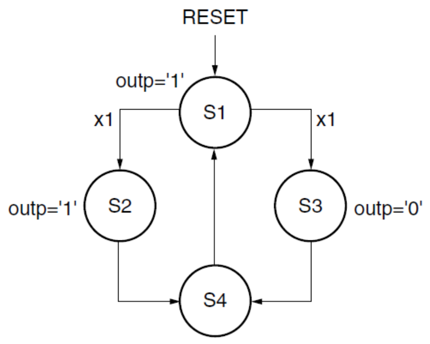


Figure 7: Finite State Machine Example

IO Pins	Description
clk	Positive Edge Clock
reset	Asynchronous Reset (Active High)
x1	FSM Input
outp	FSM Output

Table 1: FSM Pin Descriptions

FSM with One Process VHDL Coding Example

```
entity fsm_1 is
        port (
                clk, reset, x1  : IN std_logic;
                outp            : OUT std_logic
        );
end entity;

architecture beh1 of fsm_1 is
        type state_type is (s1,s2,s3,s4);
        signal state    : state_type ;
begin

        process (clk,reset)
        begin
                if (reset ='1') then
                        state <=s1;
                        outp<='1';
                elsif (clk='1' and clk'event) then
                        case state is
                                when s1 =>  if x1='1' then
                                                        state <= s2;
                                                        outp <= '1';
```

```
                                                else
                                                        state <= s3;
                                                        outp <= '0';
                                                end if;
                                when s2 =>      state <= s4;
                                                outp <= '0';
                                when s3 =>      state <= s4;
                                                outp <= '0';
                                when s4 =>      state <= s1;
                                                outp <= '1';
                        end case;
                end if;
        end process;

end beh1;
```

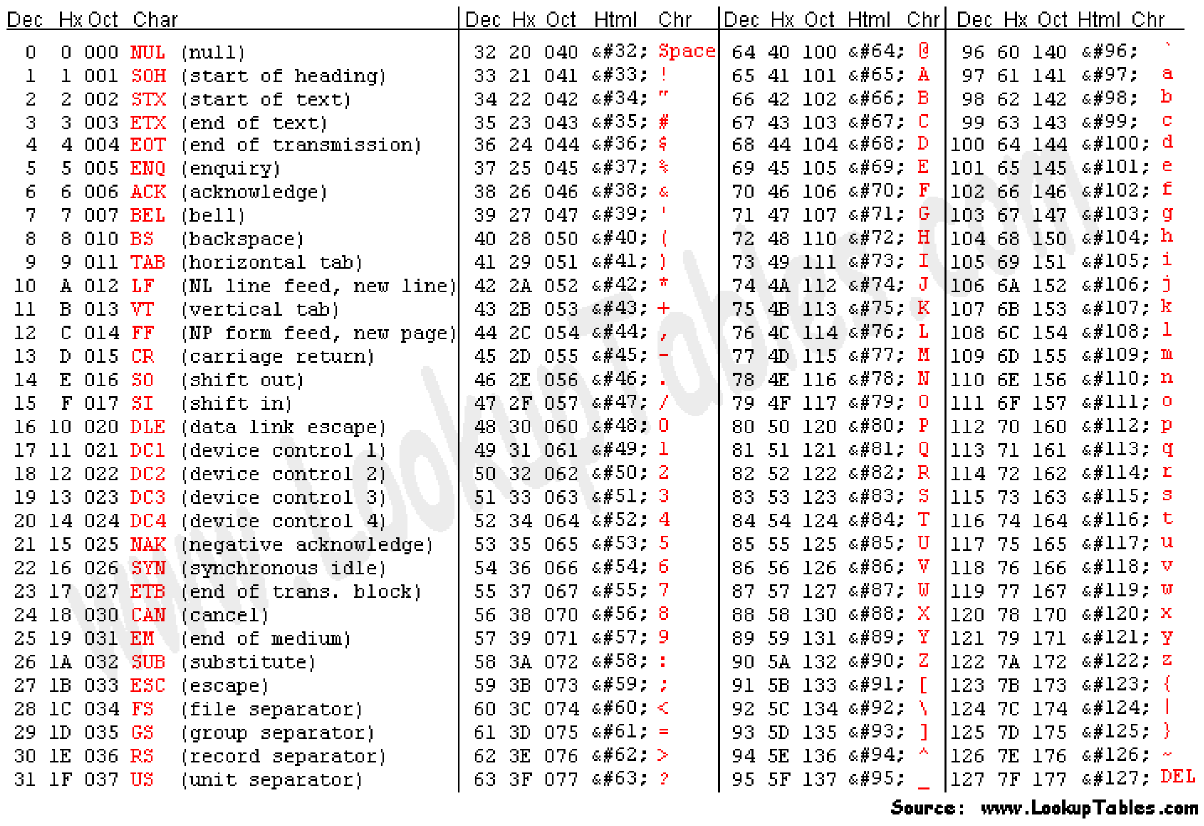
FSM with Two Processes VHDL Coding Example

```
entity fsm_2 is
        port (
                clk, reset, x1  : IN std_logic;
                outp            : OUT std_logic
        );
end entity;

architecture beh1 of fsm_2 is
        type state_type is (s1,s2,s3,s4);
        signal state    : state_type ;
begin

        process1: process (clk,reset)
        begin
                if (reset ='1') then
                        state <=s1;
                elsif (clk='1' and clk'Event) then
                        case state is
                                when s1 =>      if x1='1' then
                                                        state <= s2;
                                                else
                                                        state <= s3;
                                                end if;
                                when s2 =>      state <= s4;
                                when s3 =>      state <= s4;
                                when s4 =>      state <= s1;
                        end case;
                end if;
        end process process1;

        process2: process (state)
        begin
                case state is
                        when s1 => outp <= '1';
                        when s2 => outp <= '1';
                        when s3 => outp <= '0';
                        when s4 => outp <= '0';
                end case;
        end process process2;

end beh1;
```

FSM With Three Processes VHDL Coding Example

```
entity fsm_3 is
        port (
                clk, reset, x1  : IN std_logic;
                outp            : OUT std_logic
        );
end entity;

architecture beh1 of fsm_3 is
        type state_type is (s1,s2,s3,s4);
        signal state, next_state        : state_type ;
begin
        process1: process (clk,reset)
        begin
                if (reset ='1') then
                        state <=s1;
                elsif (clk='1' and clk'Event) then
                        state <= next_state;
                end if;
        end process process1;

        process2 : process (state, x1)
        begin
                case state is
                        when s1 =>      if x1='1' then
                                                next_state <= s2;
                                        else
                                                next_state <= s3;
                                        end if;
                        when s2 =>      next_state <= s4;
                        when s3 =>      next_state <= s4;
                when s4 =>   next_state <= s1;
            end case;
        end process process2;

        process3 : process (state)
        begin
            case state is
                when s1 =>   outp <= '1';
                when s2 =>   outp <= '1';
                when s3 =>   outp <= '0';
                when s4 =>   outp <= '0';
            end case;
        end process process3;

end beh1;
```

Appendix 8 – ASCII Codes Table



Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH	(start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX	(start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX	(end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT	(end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ	(enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK	(acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL	(bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS	(backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB	(horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT	(vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR	(carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO	(shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI	(shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE	(data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1	(device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2	(device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3	(device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4	(device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK	(negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN	(synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB	(end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN	(cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM	(end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB	(substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC	(escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS	(file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS	(group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS	(record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US	(unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com