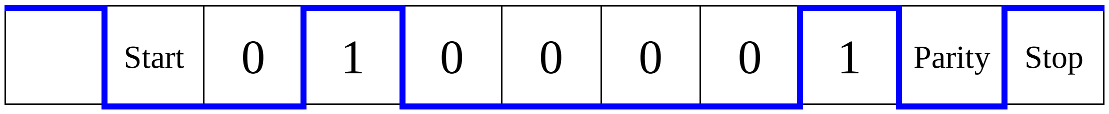
ECEn/CS 224
UART Homework Solutions

1. Draw a serial waveform of the character 'B' (ASCII for 'B') being transmitted over the UART using even parity. Be sure to include the Start bit, 7 data bits, parity bit, and a single stop bit.

   'B' = 22 hex



2. How long will it take (in seconds) to transmit the previous sequence with a baud rate of 9,600? How many characters can be transmitted per second?

   For a UART, baud rate = bits per second. Each character requires 10 bits (start bit, stop bit, parity bit, 7 data bits). Since we can transfer 9,600 bits per second, the time to transfer one character is as follows:

$$\frac{10\text{ bits}}{\text{char}} \cdot \frac{1\text{ s}}{9{,}600\text{ bits}} = 0.001041\overline{6}\text{ s/char} = 1.041\overline{6}\text{ ms/char}$$

   Similarly, the number of characters per second that can be transferred is as follows:

$$\frac{1}{0.001041\overline{6}\text{ s/char}} = 960\text{ char/s}$$

3. Create a state diagram of the receiver UART FSM. The receiver UART will be intended to handle 7 data bits, 1 parity bit, and 1 stop bit.

   You may assume that there is a Baud Rate Generator with two timers. One timer outputs a **HalfBit** signal that gives a single clock-cycle-width pulse every half bit (i.e. twice the frequency of the full bit time). The other timer outputs a **NextBit** signal every full bit time. These timers can be individually cleared using the **ClrHalfBitCntr** and **ClrNextBitCntr** signals. Remember that the receiver wants to sample the input signal in the middle of the bit time. Additionally, you may assume the existence of any other needed circuitry (e.g., counters, shift registers, parity checker, etc.).

   Be sure to consider how you will handle framing errors and parity errors as well as how to inform the host system when new data has been received. Also, consider how you will handle the fact that the serial input signal is asynchronous.

Assumptions for this Solution:

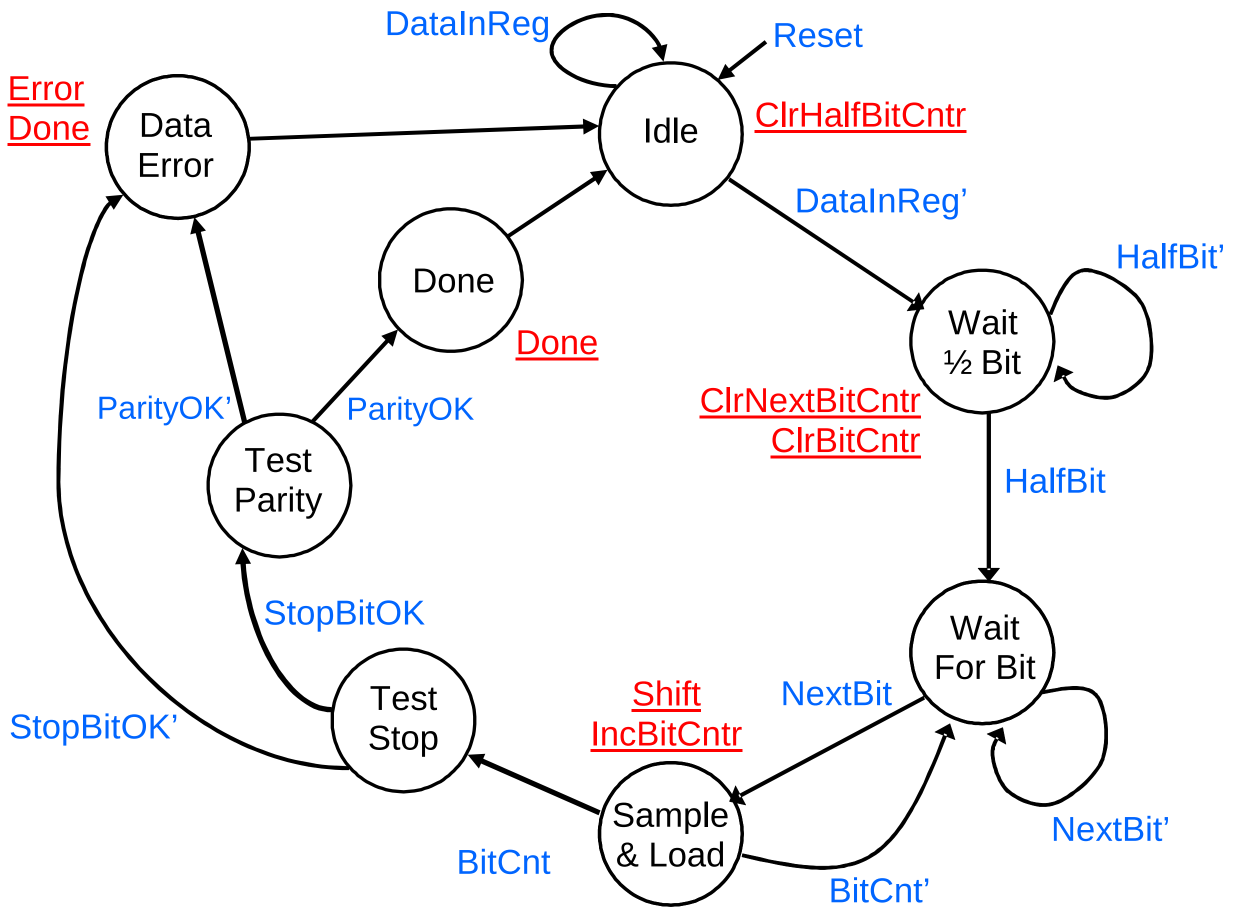
- First we assume the following inputs:
  - **Clk**: Global system clock
  - **Reset**: Global system reset
  - **ParitySelect**: Selects even or odd parity
  - **DataIn**: Serial input line
- We assume the following outputs:
  - **DataOut**[6:0]: Seven bit data output
  - **Done**: Goes high for one clock cycle to indicate when a new byte has been received
  - **Error**: Goes high with **Done** if the data received has errors (i.e., parity or framing)
- We assume that one or more flip flops are used to synchronize the asynchronous serial input with the internal system clock. The output of these synchronization flip flops is called **DataInReg**.
- We assume the existence of a bit counter, with inputs **ClrBitCntr** and **IncBitCntr** and output **BitCnt**. The **BitCnt** output goes high after the counter has been incremented 8 times. The **IncBitCntr** input is used to increment the counter and the **ClrBitCntr** input resets the counter to zero.
- We assume the existence of a 9-bit serial in, parallel out shift register. This register will hold the 7 data bits plus the parity bit and the stop bit. The **Shift** signal is used to cause this register to shift in a new bit. The data shifted in is the **DataInReg** signal.
- We assume that the shift register output is fed into parity checking logic (like that from the lecture slides). This logic computes what the parity bit should be and compares that to the parity bit received, generating the **ParityOK** signal.
- If no error occurs, we assert the **Done** signal for one clock cycle indicating a new byte has been received. If an error occurs (either framing or parity error) we assert the **Done** and **Error** signals for one clock cycle.

The state diagram is shown below. Here's how it works:

- The FSM stays in the **Idle** state until a falling edge on the synchronized input stream (**DataInReg**) is detected. When that happens, it moves to the **Wait ½ Bit** state. While idle, the ½ bit timer in the Baud Rate Generator is constantly being reset via the **ClrHalfBitCntr** signal.
- From the falling edge (beginning of the start bit), it waits for ½ bit time. While it is waiting, the next bit counter and bit counter are both cleared (via the **ClrNextBitCntr** and **ClrBitcntr** signals).
- Once half a bit time has elapsed the **HalfBit** signal goes high (indicating we are now in the middle of the start bit), causing a transition to the **Wait For Bit** state. From this point, the FSM waits for a full bit time (**NextBit**) instead of a half bit. As a result, the bits will be sampled in the middle of the bit time.
- When a **NextBit** timer signal is received from the Baud Rate Generator (indicating we are now in the middle of the first data bit), it moves on to the **Sample & Load**

state. Here, the current logic value on the serial input line (**DataInReg**) is shifted into the shift register using the **Shift** signal. The Bit Counter is also incremented using the **IncBitCntr** signal.

- If the bit counter indicates that all data bits plus the parity bit and stop bit have been received (i.e., **BitCnt** is asserted) the FSM then proceeds to the **Test Stop** state. If not, we return to the **Wait For Bit** state to wait for the next bit.
- In the **Test Stop** state, we check if the stop bit is indeed a ‘1’, as it should be. If it is, the **StopBitOK** signal will be high and we proceed to the **Test Parity** state. If not, we proceed to the **Data Error** state.
- In the **Test Parity** state, we check the output of the parity checking logic (the **ParityOK** signal). If it is a ‘1’ we proceed to the **Done** state. Otherwise we proceed to the **Data Error** state.
- In the **DataError** state we simply assert the **Error** and **Done** signals, indicating that we have received a byte but that the byte has errors.
- In the **Done** state we simply assert the **Done** signal, while keeping the **Error** signal low, indicating that we successfully received a new byte.
- At this point we have finished and we return to the **Idle** state to wait for the next start bit.


DataInReg
Reset
Error
Done
Data Error
Idle
ClrHalfBitCntr
DataInReg’
HalfBit’
Done
Done
Wait ½ Bit
ParityOK’
ParityOK
ClrNextBitCntr
ClrBitCntr
Test Parity
HalfBit
StopBitOK
Wait For Bit
Shift
IncBitCntr
NextBit
Test Stop
StopBitOK’
Sample & Load
NextBit’
BitCnt
BitCnt’