John Tramel

CECS460

UART Transmit Engine

Design Review

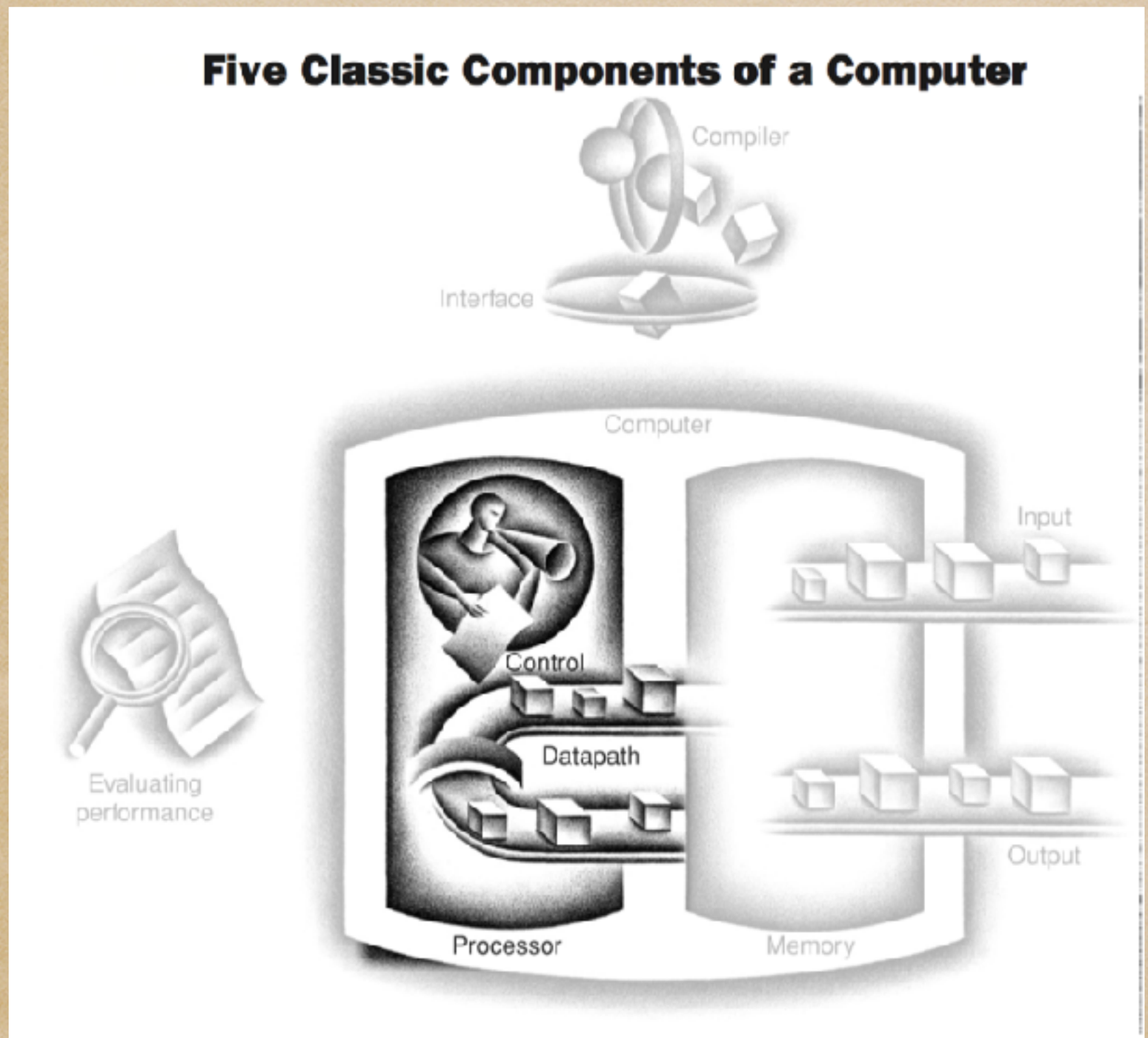
Design Methodology

- Hennesy/Patterson book does an excellent job of presenting digital designs with two main stages: 1) The data path and 2) The control logic
- The first stage is the data that must flow through *your* machine
  - All registers should be defined
  - All counters should be defined
  - All data selection should be defined
  - All functional clouds should be defined

Design Methodology 2

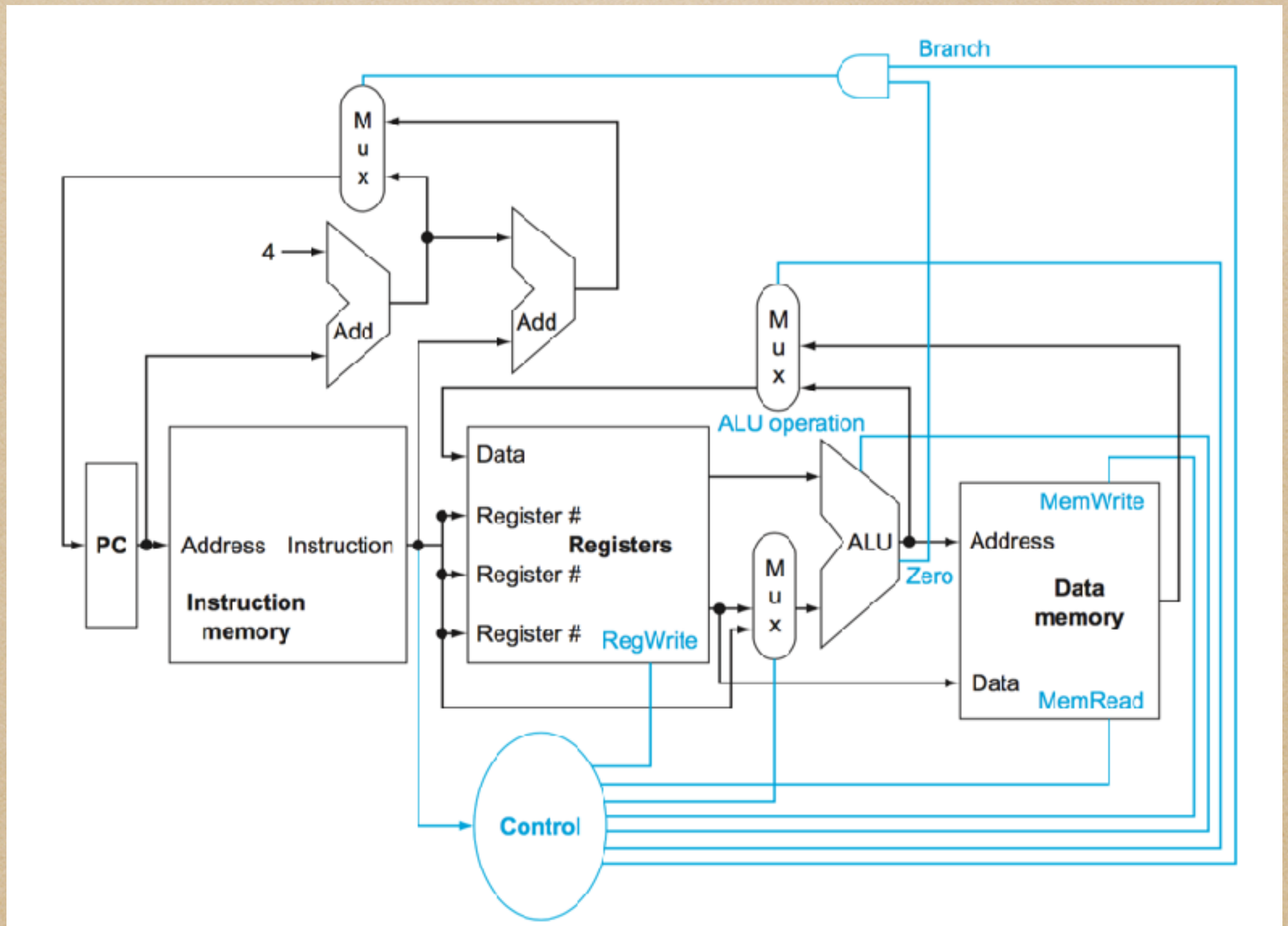
- The second main (and critical) step is to identify all of the control signals that you are going to need (enables, selects, etc.)
- This list of control signals will be the focus of your state machine (control logic) design.
- It is the responsibility of the state machine to control the movement of the data through the data path portion
- In H/P they identify the control signals with a different color (blue)

Hennesy/Patterson Partitioning



- Input
- Output
- Memory
- Processor
  - Datapath
  - Control

Digital Design Architecture: Datapath and Control



Design Methodology 3

- I can not stress enough the need to have a well-thought out plan before you begin your implementation
- Without such a plan you are destined for "design thrashing" and inevitably loose heart in your implementation
- Once you have thought out your design and captured it then the next step is to review your implementation before you commit to coding. Here that review should be with your fellow students in your study groups
- Remember that any skill worth obtaining - must be obtained with a cost: your time plus your concentration

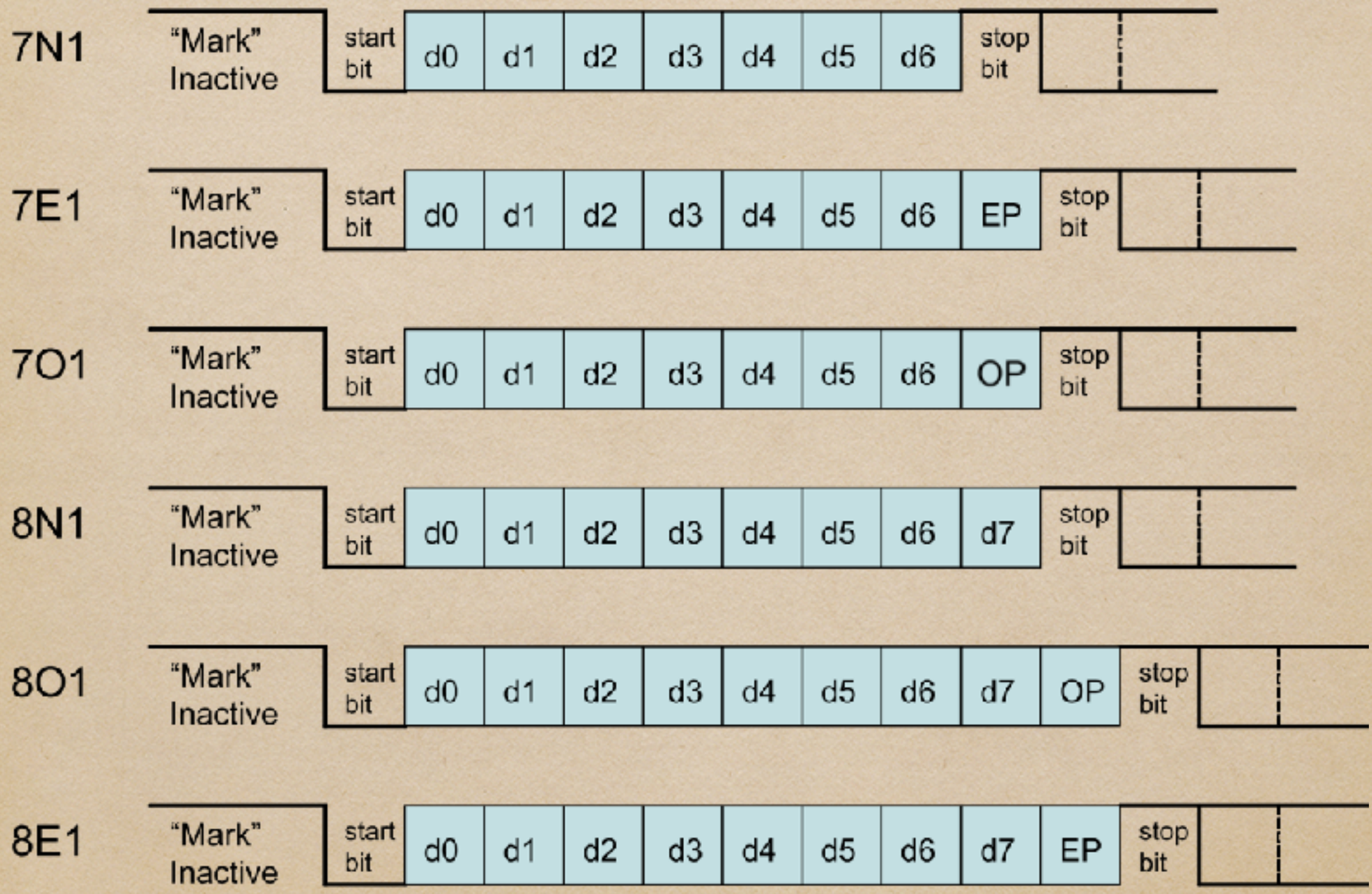
Design Implementation - What?

- This assignment is to design the transmit engine for a UART that has selectable baud rate, bits transmitted, parity enable/disable, odd/even parity when enabled.
- The specification for our design is the CoreUART specification from Actel that gives the signal names and describes the functionality
- Another main functionality is UART protocol discussed in class and reviewed in the specification
- This discussion assumes that you understand both of these

UART Configuration

- Universal Asynchronous Receiver Transmitter (UART) protocol allows communication between two electronic devices without including a clock across the interface
- What is required is that both sides of the communication "know" what the configuration is: baud rate, #bits, parity O/E/none
- The configuration will not change dynamically. It will be configured then reset will be applied. So all decisions regarding the configuration may be made assuming that they will not change

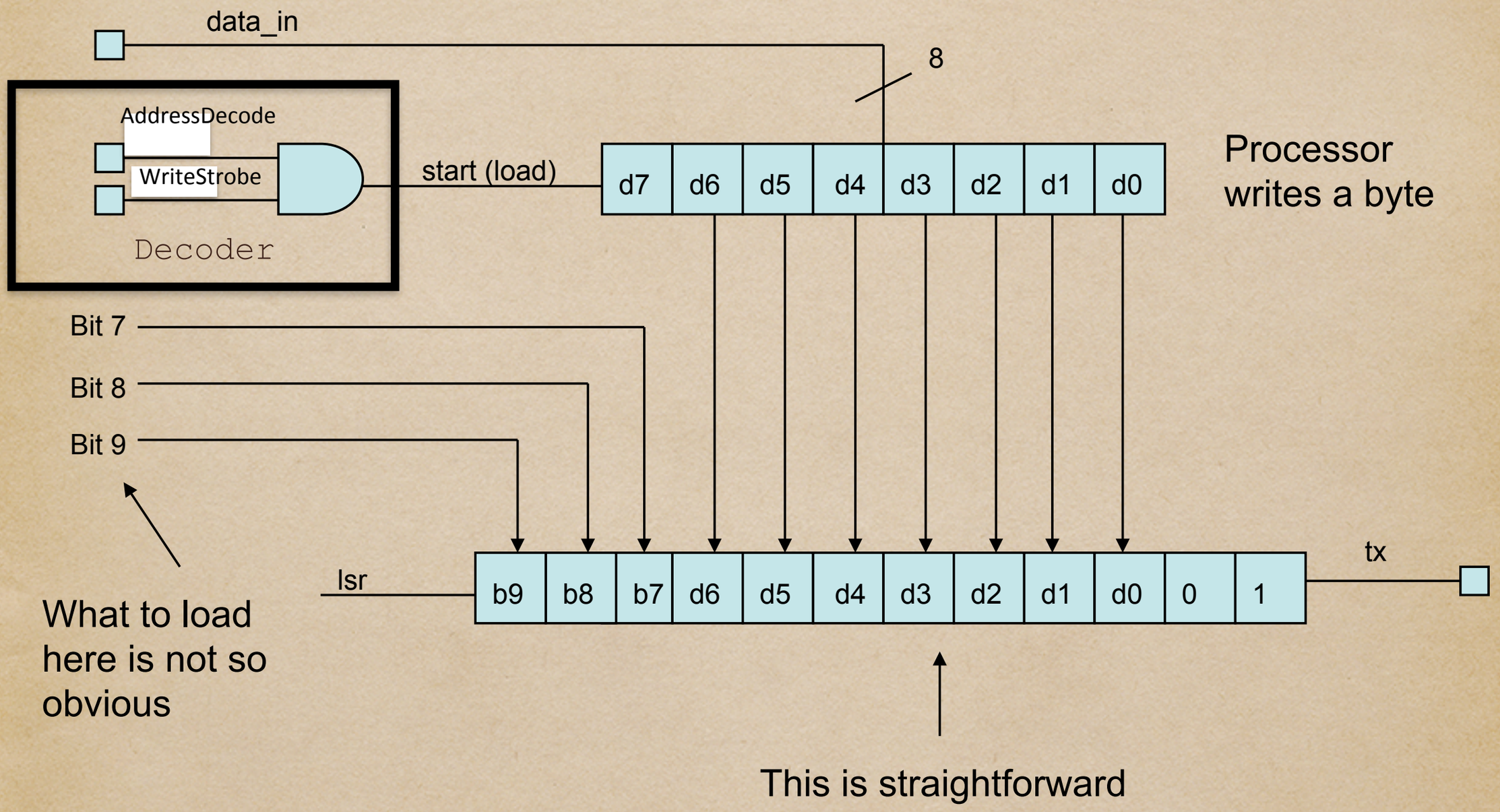
UART Data Protocol



Our Design - Shift Register

- At the heart of our design is a shift register that needs to be loaded with the correct information and then needs to be shifted out the correct number of times
- Knowing that "Mark", or "1" is the inactive state of the tx signal, then reset should assign 1 and we need to make sure that we don't let it switch till we want the "start" bit ("0") to be transmitted.
- How the data is loaded in to the shift register depends on the UART configuration

Data Path Design



Generating the Missing Bits

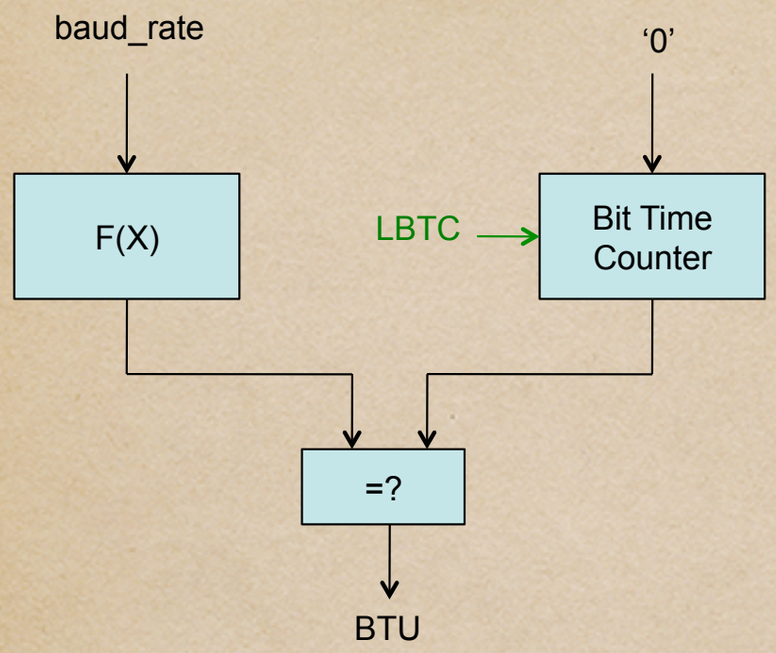
- The functionality of the bits to be loaded, {b9,b8,b7} can be captured in the form of a truth table
- The LHS represents the configuration and the RHS represents the bit values along with the number of bits to transmit

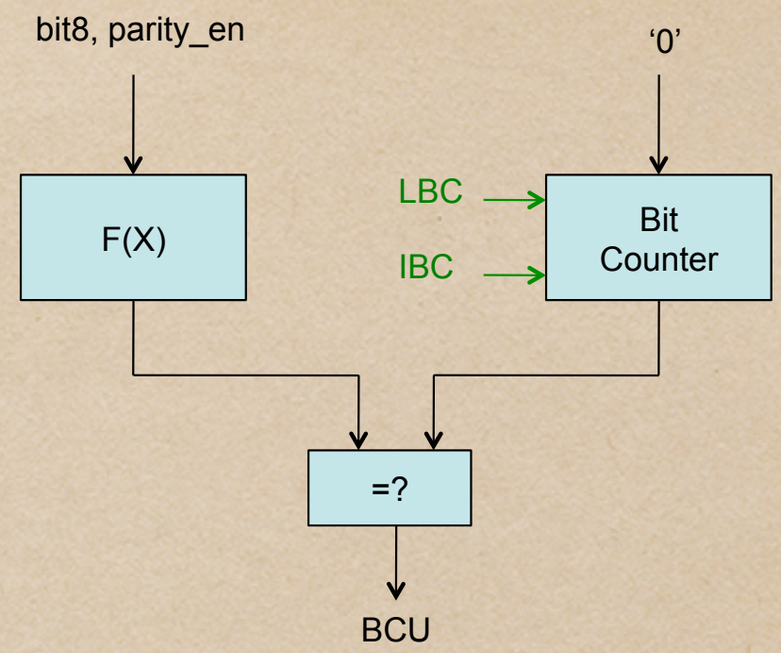
bit8	parity_en	odd_n_even	b8	b7
0	0	0	1	1
0	0	1	1	1
0	1	0	1	EP
0	1	1	1	OP
1	0	0	1	d7
1	0	1	1	d7
1	1	0	EP	d7
1	1	1	OP	d7

Additional Hardware

- We also need two counters: 1) Count clocks to determine the bit time, and 2) Count bits to know when we are done
- We also need to be able to generate the terminal counts for both counters. These are dependent on the configuration bits
- We will use these two to know when we have waited a bit time and to know when all the bits have been transmitted

Diagram of Extra Logic





- The baud_rate determines the number of clocks (50 or 100 MHz) that are necessary to fill one bit time
- The bit8, parity_en will determine the number of bits that need to be transmitted
- BTU = Bit Time Up BCU = Bit Count Up

Conceptual Data Flow

- Transmit machine comes out of reset with TXRDY active - ready to transmit
- TramelBlaze is interrupted telling it that it should write the transmit engine when data is ready - in this project data is always ready
- Writing UART causes TXRDY to go inactive while data is transmitted
- When transmission is complete the UART sets TXRDY to indicate that it is ready to send again
- The transmit engine must keep track of BIT TIME (how long each bit is held on the wire) and NUMBER of BITS sent (for us always eleven)