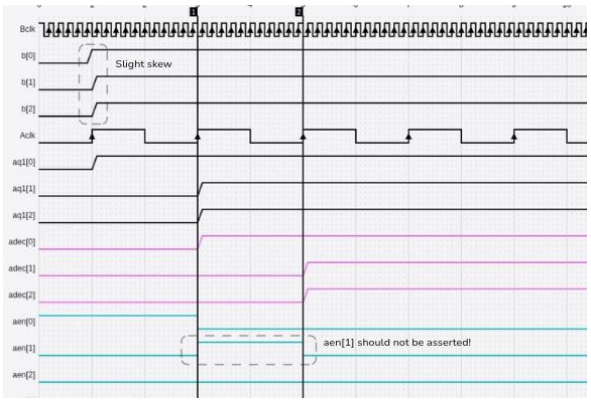
A)1. Explain why synchronizing each bit of an encoded multi-bit control signal independently can lead to incorrect decoding in the receiving clock domain.

- when you send a signal over to a different clock domain with single bits of 1/0s it is compatible to use a multibit toggle synchronizer that too provided the impulses or changes are shifted apart but especially when trying to send data as a bus we to prefer sending the whole thing as 1 package rather than individual bits
- This is primarily due to issues such as data incoherence / metastability or latency/skew(time shift) alongside reconvergence(errors in recombining data once its reached) here mainly we see incoherence or rather latency coming into play as there is a skew in the signal input between a[0] and a[1],a[2] thus they reach the synchronizer at different times
- Decoder simply decodes whatever bit pattern it sees at its input giving an invalid/intermediate value
- When bits of a multi-bit signal are synchronized independently, they are subject to individual wire delays (routing skew) and independent metastability resolution. If the receiving clock edge occurs during the transition of these bits, some bits are updated by the clk while others retain their previous states or get stuck in a metastable state(might push to 1 or 0)
- The signal a[0] reaches before the A-clock/setup time and thus provides required output however bits a[1] and a[2] reach after the clock cycle this might be due to phase/timing error in the inbuilt system a or the "phase" of the sampling clock A-clk or even the wired paths being different lengths causing stalling in their path
- Sometimes devices like ff might have the same specifications and still have different internal parameters like setup time or hold time causing delays in passing out the values

A)2. Using the timing diagram, describe how skew between b[1] and b[0] causes adec[2:0] to momentarily take an invalid intermediate value.

- As observed the input b[1] has a skew due to which the bit toggles after the clock cycle and is thus not recorded by the synchronizer in the same clock cycle
- b[1] lags behind b[0] by one clk cycle hence the n stage synchronizer after n stages gives a value with some bits holding memory while others are updates here b[1] is holding memory 0 while b[0] updates to 1 this gets pushed to decoder which simply decodes this signal to give high at adec[1]



- only at the next Aclk edge b[1] and b[2] are finally captured, allowing adec to reach the correct value of 111

A)3. Identify the fundamental CDC design mistake illustrated in this figure.

- sending bus data as single input signals through seperate multistage synchroniizers makes each one of them individualy vulnerable to skew, metastability further errors are caused by dataincoherence as the data bus might not arrive on time

- the bits in the data bus originate from a single source in the A domain however as they are split and sent they have a skew between them so they face a reconvergence failure and the decoder processes a value that never existed in the A domain thus this system doesn't guarentee that the multibit system remains "coherent" as a bitwise synchronized output

A)4. Propose three different design techniques that can be used to safely transfer this control information across clock domains without generating spurious decoded outputs.

- **Gray Coding**: If the 3-bit signal can be changed to a Gray code sequence, only one bit will change at any given time. This ensures that even if there is skew, the receiver will either see the previous value or the next value, but never an invalid phantom value , hence the old state repeats till the metastability settles and the signal arrives

- **Mux-Recirculation**: Place the 3-bit data on a stable bus and send a single-bit Ready pulse through a 2-stage synchronizer. The receiver only samples the entire 3-bit bus once the Ready signal has been safely synchronized. This is more efficient that the proposed model as it used only a single multistage synchronizer instead of using one for each bit, the databus is normally passed from one domain to another and the handshake mechanism checks for it to have the same value as input has stationary only then it allows next value to enter till then previous value is repeated

- **Asynchronous FIFO**: Use a dual-clock FIFO to pass the data. This is the most robust solution for multi-bit data, as it uses internal Gray-coded pointers to manage the transition safely without the risk of incoherence