Overall Architecture:

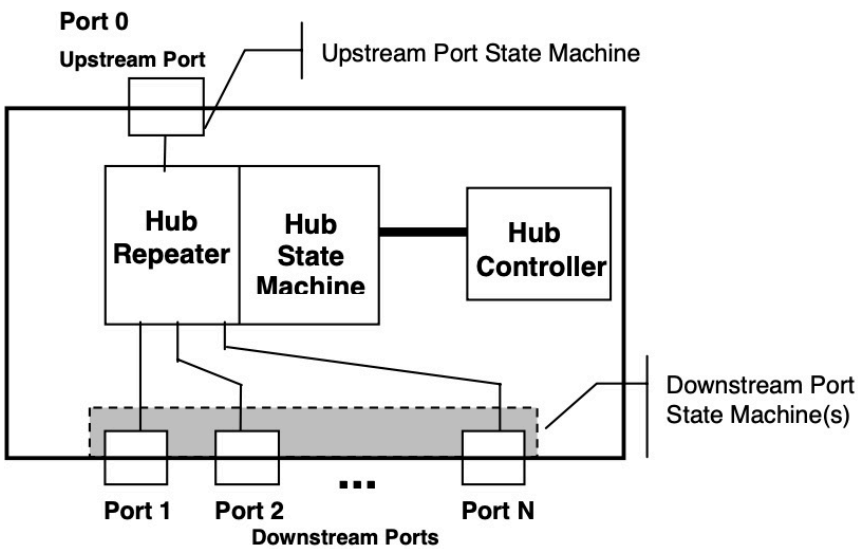
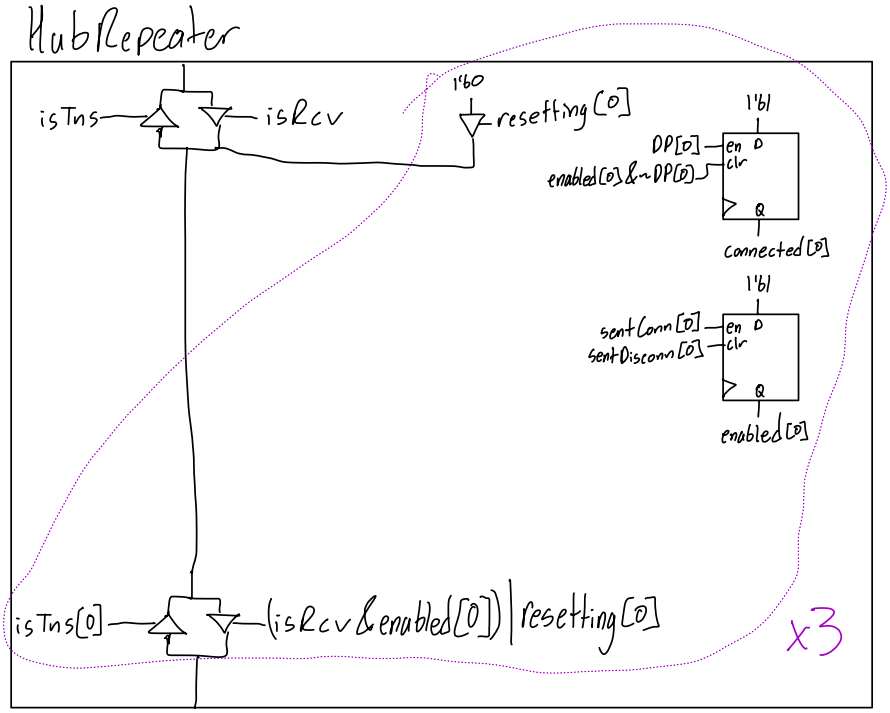


Figure 11-1. Hub Architecture

Note: I did not get to draw the datapath for the Hub Controller in time for this submission. This component is complicated and handles all of the specific packet requests addressed to the Hub. There is a lot more work to be done than what is described in this document.

- detect transmissions from UFP, relay to all DFPs
- detect transmissions from DFP, relay only to UFP
- detect connections and disconnections, report during GET_PORT_STATUS requests
- Most signals are simply handled by the FSMs
- This datapath indicates the connection status that is used by the HubController

HubRepeater



Necessary FSMs:

- One for each DFP, controls its BusDrivers
- One for UFP receiving, outputs J, K, or SE0 for Hub Controller to interpret data
- One for UFP transmitting (i.e. if a device is transmitting) to prevent a device hanging or throwing everything into an error state
- One for HubRepeater, controls BusDrivers

DFP FSM:

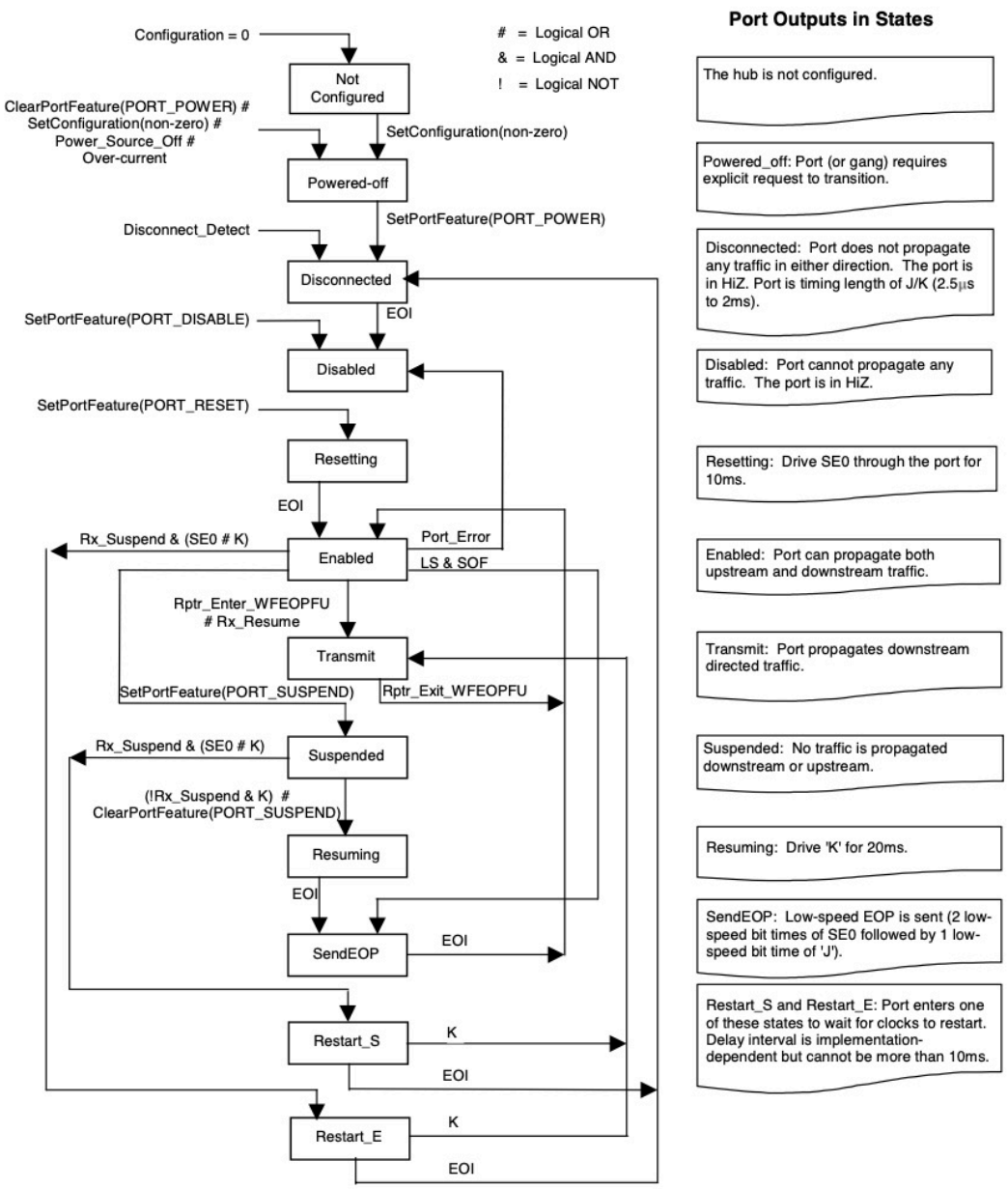


Figure 11-6. Downstream Hub Port State Machine

UFP Receiver FSM:

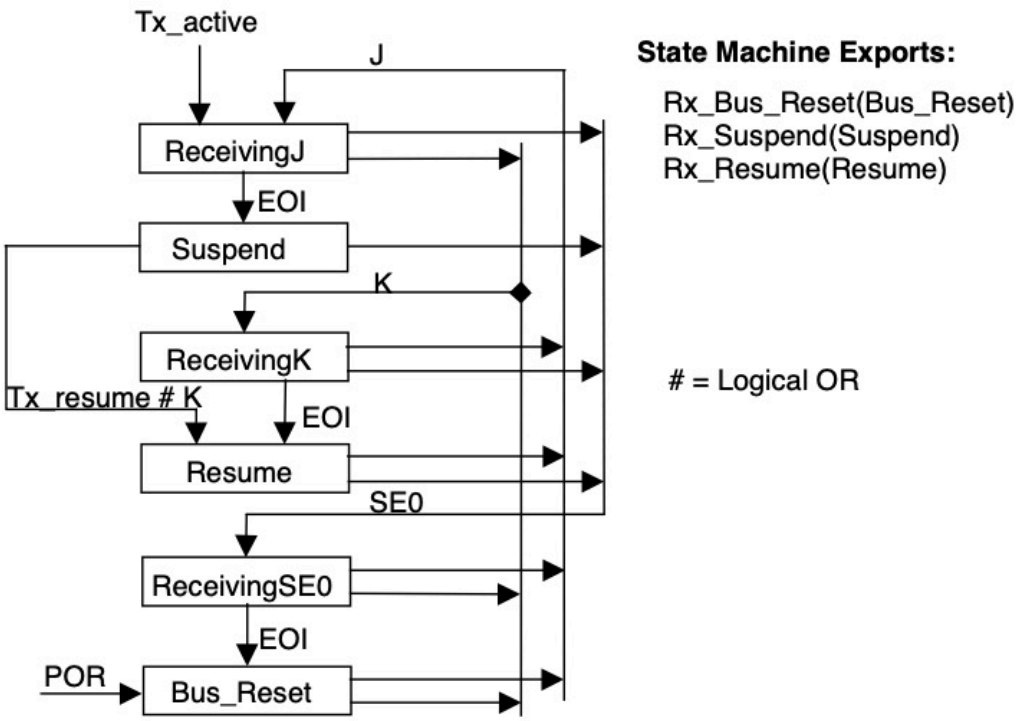


Figure 11-7. Upstream Port Receiver State Machine

UFP Transmitter FSM:

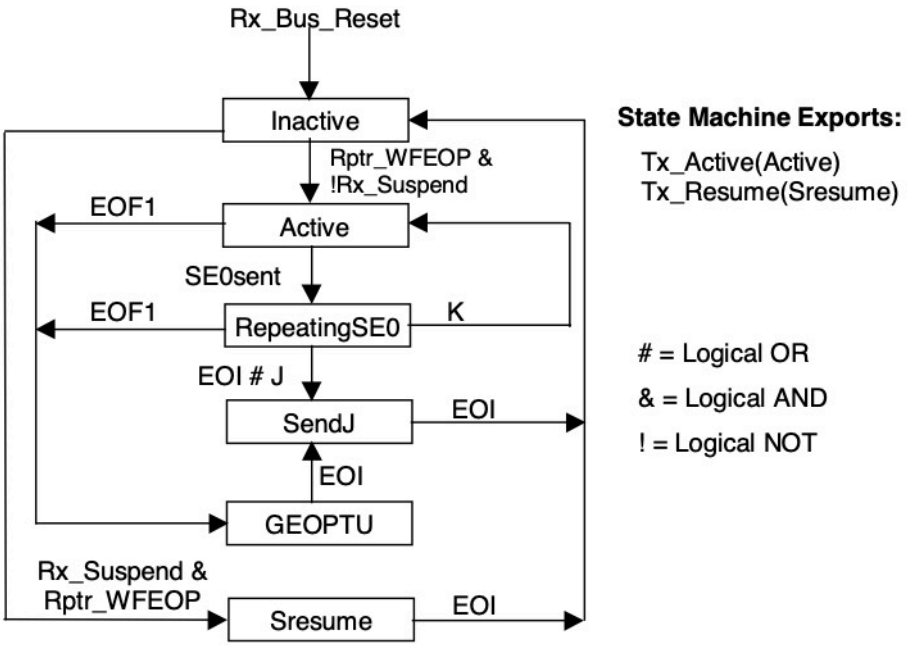


Figure 11-8. Upstream Hub Port Transmitter State Machine

•

Hub Repeater FSM:

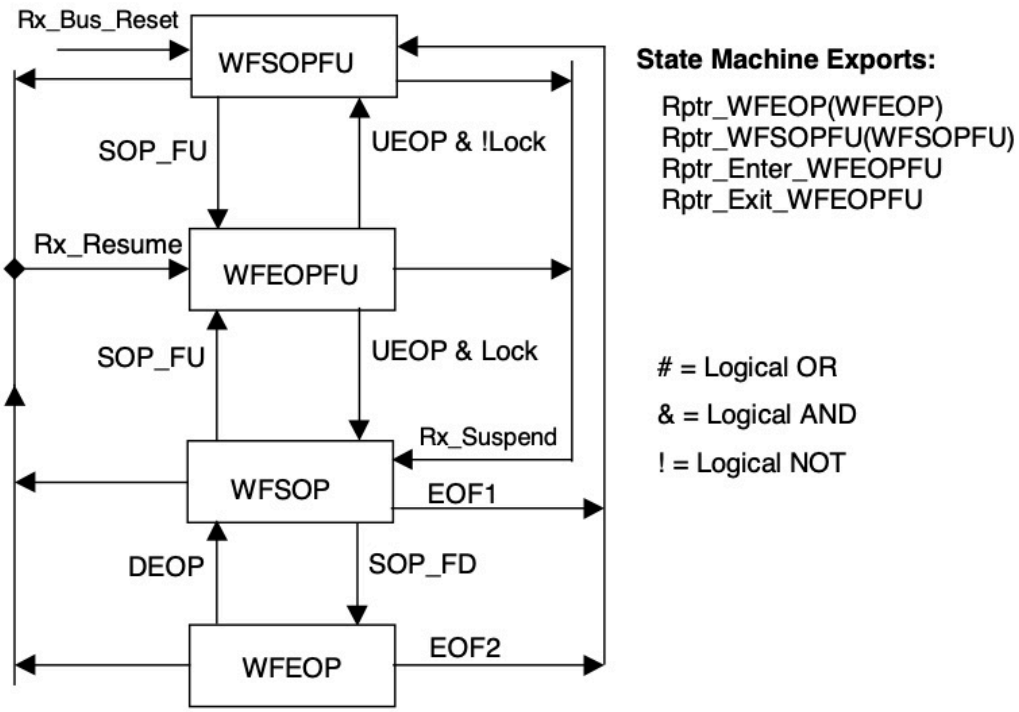


Figure 11-9. Hub Repeater State Machine

Testing Strategies:

For this project, I would have to create sequences of serial signals that emulate the packets a host would send. In SystemVerilog and/or Python, this would not be too difficult to actually create and send the signals at the appropriate bit rates. The most tedious part of this would be having to create the packets themselves, but thankfully I would be able to get away with modifying only certain parts of packets since many parts of them are set in stone (SYNC, PID, etc.). For an initial test, sending a sequence of a RESET and SETUP would suffice I believe. Eventually, I would want to attempt faking a device connection.

Below and on the next page are all of the requests that a Hub must respond to. While it would be best to test all of these, my focus will be upon those related to SETUP, device connection, and PRE packets that are used when sending data to a low-speed device.

Table 11-9. Hub Responses to Standard Device Requests

bRequest	Hub Response
CLEAR_FEATURE	Standard
GET_CONFIGURATION	Standard
GET_DESCRIPTOR	Standard
GET_INTERFACE	Undefined. Hubs are allowed to support only one interface
GET_STATUS	Standard
SET_ADDRESS	Standard
SET_CONFIGURATION	Standard
SET_DESCRIPTOR	Optional
SET_FEATURE	Standard
SET_INTERFACE	Undefined. Hubs are allowed to support only one interface
SYNCH_FRAME	Undefined. Hubs are not allowed to have isochronous endpoints

Table 11-10. Hub Class Requests

Request	bmRequestType	bRequest	wValue	wIndex	wLength	Data
ClearHubFeature	00100000B	CLEAR_ FEATURE	Feature Selector	Zero	Zero	None
ClearPortFeature	00100011B	CLEAR_ FEATURE	Feature Selector	Port	Zero	None
GetBusState	10100011B	GET_ STATE	Zero	Port	One	Per-Port Bus State
GetHubDescriptor	10100000B	GET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
GetHubStatus	10100000B	GET_ STATUS	Zero	Zero	Four	Hub Status and Change Indicators
GetPortStatus	10100011B	GET_ STATUS	Zero	Port	Four	Port Status and Change Indicators
SetHubDescriptor	00100000B	SET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
SetHubFeature	00100000B	SET_ FEATURE	Feature Selector	Zero	Zero	None
SetPortFeature	00100011B	SET_ FEATURE	Feature Selector	Port	Zero	None