Here is a comprehensive guide as to what the USB RX **SHOULD** do and how the RCU attempts to facilitate it. If the code written does not reflect this, it is **WRONG.** Please let me know if something is off.

All sequences in a packet sent follow one pattern:
A sync byte is sent -> a pid (with its inverse) is sent -> a data phase may or may not be sent, followed by a CRC for validation -> an eop is always sent.

The USB RX's job is to make sure that the packet it receives is correct every step of the way and transmit an error if it finds that it is wrong at any step in the process.
So it must:

- Receive a sync byte
- Check that sync byte
- Receive a pid byte
- Check that pid byte and make a decision on where to go based on it
- Receive data bytes (optional)
- Check the CRC once it is finished receiving those bytes (optional)
- Receive an EOP

Based on the requirements set forth by the protocol module there is one final step; transmitting a done.
So how do we interpret the RCU?

- IDLE - AKA nothing is being received
- RECEIVE - the input has begun to send data so the timer needs to be turned on
- SYNC - Waits until a full sync byte has been received (byte_complete will show this)
- CHECK_SYNC - validates the sync byte before continuing (should be pretty quick)
- PID - Waits until a full pid byte has been received.
- CHECK_PID - validates and determines based on the pid received where to go next (should be pretty quick as well)
- DATA01 - the data state for DATA0 or DATA1, should see a variable number of data bytes, followed by a 16 bit CRC value and then an EOP. EOP is what stops this because there is no other way to determine where a data packet begins or ends…
- DATAINOUT - the data state for an IN or OUT token, this has an address, endpoint number, and then a 5 bit CRC value. This is currently stopped by EOP, but counting the bytes sent through could help.
- CHECK16 - Purely devoted to checking the 16 bit CRC value that will show up during a DATA01 phase. It follows the same pattern as the other check states. Since the data phase **SHOULD** be done at this point (since EOP is what controls the previous phase), the timer needs to be halted at this point to not take extra data beyond what it is supposed to.
- CHECK5 - Exactly the same as 16 bit but instead it checks the 5-bit CRC that follows a IN or OUT token.
- EOP - There are some packets that don't have a data part to them, but an EOP should still be sent. This is to catch the arrival of this EOP for those packets.

- ERROR - any time something goes wrong, this is the state where the RCU goes. Reports all errors.
- DONE - any time something is transmitted, whether there is an error or not, this state is visited to show a transaction is complete.

Challenges:
Timing: The exact moment the timer should be stopped is the clock cycle after an EOP has been received… no sooner or later than that. Determining when this happens is a bit of a struggle based on the operations given.

Since there are various checking states and operations that occur while going through this, it is a hope that each state can capture it’s proper value, without any delay. It then needs to keep it for checking in the rx_data_buffer.

If for any reason an EOP is missed (transmitted but not recognized by the RCU at the right moment), it could throw the whole operation into an infinite loop. This is why the combinational logic to store it until going back to idle is included as an idea to combat this issue.