CAN Decode Logic

Notes:

* Our controller uses 11 bit IDs for its messages.
* The CAN driver ReceivePacket method requires an address of a buffer of 4 LONGS be passed in for its data. It then stores the packet to that buffer and returns 1 of 3 statuses (NOTHING\_READY, NO\_MORE, STILL\_MORE) which will allow us to decide what to do from there.
* Buffer[0] contains the 11 ID bits in its first 11 bits, thus we will need to shift it right 21 bits to move the ID to its correct position for translation to decimal.
* Buffer[1] can be ignored as it is mostly empty and only contains the message length. We know that all of our packets will be 8 bytes so we don’t need this info.
* Buffer[2] contains Data[0-3]
* Buffer[3] contains Data[4-7]

Logic:

* Variables
  + Long Buffer[4]
  + Long message[8] (there is most likely a cleaner way to use just Bytes for this)
  + Long msgID
  + Long statorT
  + Long inverterT
  + Long torqueActual
  + Long speedActual
  + Long powerstageState
  + Long activeFault
  + Long mechStatCode
  + Long batteryV
  + Long batteryI
  + Long thermLimCause
* If RecievePacket(@Buffer) != NOTHING\_READY
  + msgID := Buffer[0] >> 21 //Shift the message ID over 21 bits to put it in the right place for decimal translation
  + message[0] := ( Buffer[1] AND %11111111000000000000000000000000 ) >> 24
  + message[1] := ( Buffer[1] AND %00000000111111110000000000000000 ) >> 16
  + message[2] := ( Buffer[1] AND %00000000000000001111111100000000 ) >> 8
  + message[3] := ( Buffer[1] AND %00000000000000000000000011111111 ) >> 0
  + message[4] := ( Buffer[2] AND %11111111000000000000000000000000 ) >> 24
  + message[5] := ( Buffer[2] AND %00000000111111110000000000000000 ) >> 16
  + message[6] := ( Buffer[2] AND %00000000000000001111111100000000 ) >> 8
  + message[7] := ( Buffer[2] AND %00000000000000000000000011111111 ) >> 0
  + **Note: There is an elegant way of doing this in a loop. For now this is just a printout off what needs to be done that can/will be reduced to loop form later.**
  + If msgID == EEXCANTempStatusID
    - statorT := (9/5)\*(message[0] – 40)+32
    - inverterT := (9/5)\*(message[1] – 40)+32
    - **Note: This will store all temperatures in Fahrenheit as that is the temperature scale Americans are used to.**
  + If msgID == EEXCANMechStatusID
    - torqueActual := message[0] //Torque data is two bytes long so store the first byte
    - torqueActual << 8 // Shift it right 8 bits to make space for the next
    - torqueActual += message[1] //Add the second one
    - torqueActual -= 3000 //Subtract the offset (in Nm)
    - speedActual := message[2]
    - speedActual << 8
    - speedActual += message[3]
    - speedActual -= 20000 //(in RPM)
    - powerstageState := message[4]
    - activeFault := message[5]
    - mechStatCode := message[6]
  + if msgID == EEXCANElecStatusID
    - batteryV := message[0]
    - batteryV << 8
    - batteryV += message[1]
    - batteryI := message[2]
    - batteryI << 8
    - batteryI += message[3]
    - batteryI -= 500
    - thermLimCause := message[4]