Hello Rahul,

A few things need attention in your schematic, setup, and simulation.

Setpoint
=======
Please read in some text book on control theory, what the setpoint is.
In your system, the setpoint needs to subtracted from the digitized value of the output voltage to yield the digital error value.

PID-Regulator
===========
I drew and scanned what is implemented in your PID_regulator functional view – see attachment.
Where do you have this from and why didn't you implement my sketch straight ahead?
Nevertheless the implementation is ok.
You got the minus signs in the comment for k2 wrong (line 24), since in the final summation (line 50) it is minus k2 plus all the others

This type of implementation will certainly require larger bit widths for intermediate values. This does not show up explicitly in your code, because you're using integer parameters and 8-bit signed shift registers.
Also, the three coefficients kd, kp, and ki usually take small values, often including fractional values.
A simple way to implement fractional values is a scaling factor at the output.
E.g. you can implement all the data path in the PID regulator in 16 bits, and only output the higher 8 bits. This way all integer values inside the PID regulator take on the 256-fold of their effective value from the outside perspective.

Saturation: To avoid erratic behavior in extreme situations, add a saturation to the output: calculate the final sum with two more bits, which avoids overflow in the first place.
Then clip the result between 0 and the max number according to the chosen bit width of the other parts.
This will make the PID_regulator's output to stay stick at 255 or 0, when the output voltage is constantly too low or high, respectively.

You don't need `celldefine in your Verilog code. It's primary use is for timing annotation to individual standard cells. It has no real meaning in RTL code.

You should extend the PID_controller module such that you can input kd, kp and ki instead of k1, k2, k3.
In order to find good values for the parameters, start with kd=0, kp with a small positive value (one or slightly less in the fixed-point format) and ki with a very small positive value (1/1000 or less in the fixed-point format).
Increase kp and ki together for quicker action. Reduce kp and ki to avoid ringing. Keep ki much smaller than kp.
Probably, you don't need kd, because the controlled section does not have significant dead time or lag.

Clocking
=======
Please include the clock generator for the counter within the PWM_controller functional.

The way you have it right now is a vpulse analog voltage source in the schematic which causes 4 discrete points in time per clock cycle, which makes your complete simulation run very slow.

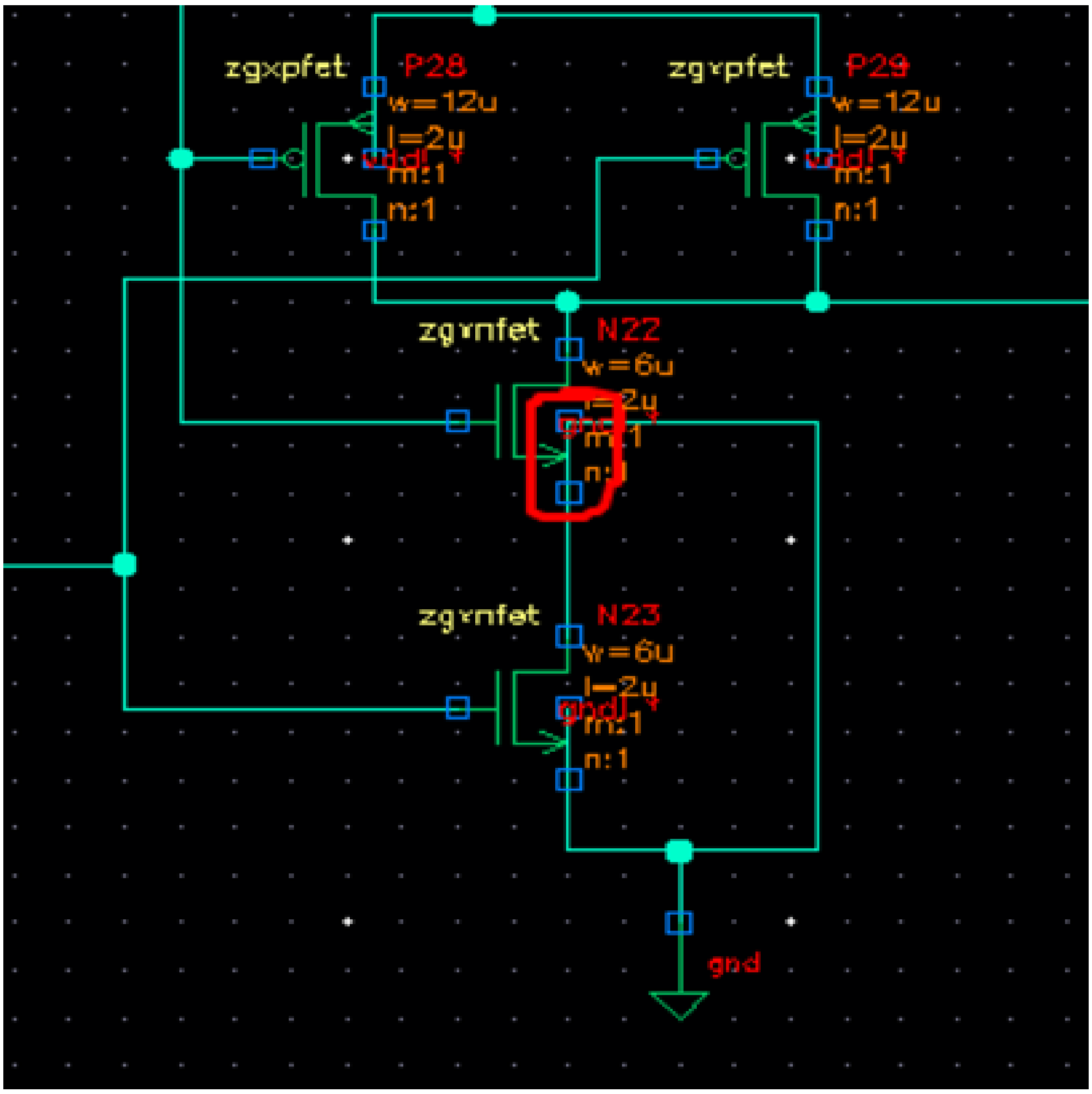
Also, you should try to derive the ADC clock from the PWM_controller as well.
This way ADC and PWM_controller operate synchronously.
A good starting point is one ADC conversion in one full PWM cycle.
As one way of optimization, you can later try to reduce ADC conversion rate.

ADC
===
While an ADC model copied from Cadence examples is a good starting point, it is a good idea to model one yourself.
The first goal should be to avoid unnecessary A/D interface elements (aka Connect Modules), to improve simulation speed.
The current ADC outputs are 8 analog lines, which should be digital lines.

Connect Rules / Interface Elements
=============================
Currently, your simulation setup in the ams_state1 uses ConnRules_inhconn_full_fast.
These use global nets vdd! and vss! for high and low values.
In your schematic, you have set them to the 3.7V supply.
This is fine for the PWM output.
However, it is valid for the PWM_control clock input as well, which is why the clock is not received in the PWM_control and counter.
The clock is generated as a 1V signal. Please do not simply fix the clock voltage source, but remove it and include the PWM base clock in the PWM_control module.

Transistor dimensioning
===================
Good: all devices are now zgx*fet.
You use very large values for transistor length (3u and 4u). While this is good for analog performance, it is not for switching performance.
You should use the minimum allowed channel length for zgx*fet for all the zgx*fet devices. I seem to remember values around 400nm, but you'll find out.
Transistor widths:
Main switching devices (P0 and N0 in your schematic): Make them as wide as appropriate to have a small voltage drop across them, when they are turned on and carry the inductor current. Make sure you check this with a reasonable value for the inductor current.
The current dimensioning (12.8/3 and 9/4) seems quite small.
Transistor widths in the inverter chain: work backwards from the main switching devices: Last inverter stage: approximately chose w_last = w_main/6 for both pfet and nfet. Make the pfet somewhat wider than the nfet.
From stage to stage, backwards through the inverter chain, the width can be reduced by a factor of about 3 to 4.

Schematic error: You shorted the upper nfet in the two NAND gates:



HTH, Kind Regards,
Jörg