
Qspice C++ Discrete PID Controller Implementation

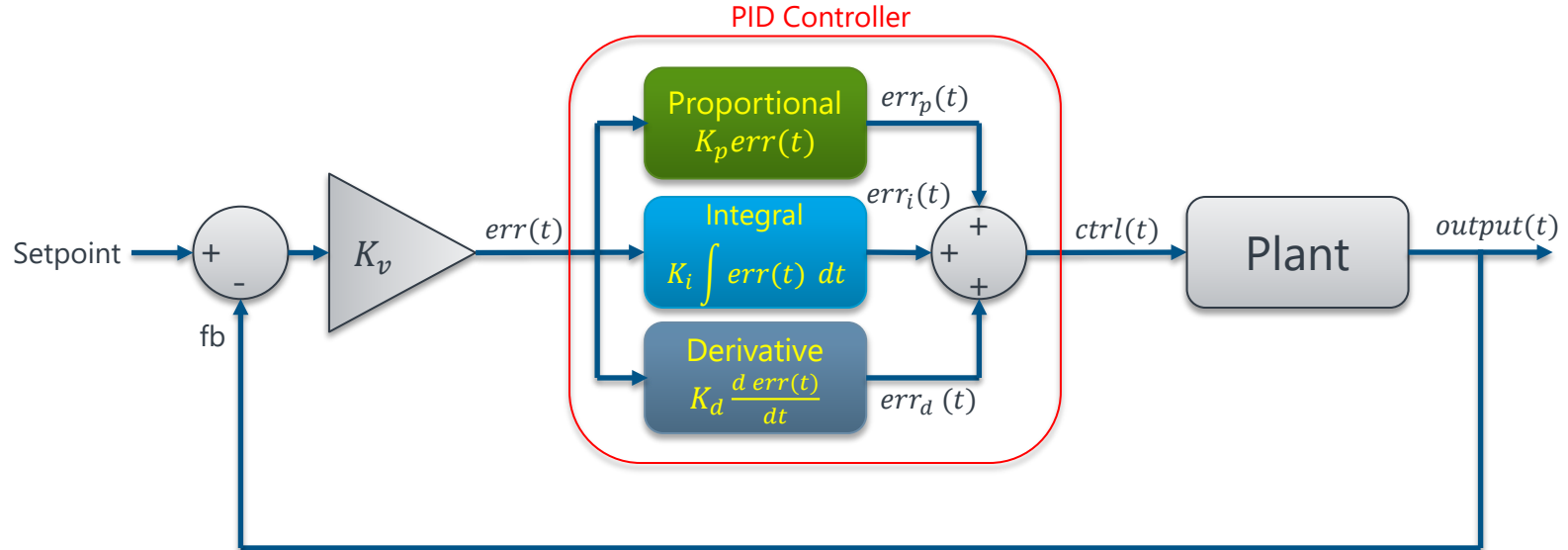
Member : KSKelvin
Member : RDunn
<https://forum.qorvo.com/>

Last Update : 9-2-2023

C++ PID Controller for Qspice

- Information

- Project Title : C++ PID controller for Qspice
- Performed by two Qspice forum members : KSKelvin and RDunn
- Project file location : <https://github.com/robdunn4/QSpice>
- Description : Proportional-Integral-Derivative (PID) control is the most frequently used control algorithm in control system, this project implement a digital PID with a Ø-Device in Qspice



Discrete PID Controller Implementation

- Fundamental of PID Controller

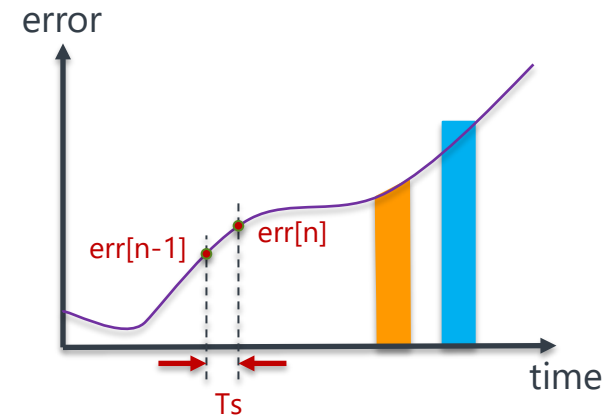
- Time formula

- $ctrl(t) = K_p err(t) + K_i \int err(t) dt + K_d \frac{d err(t)}{dt}$
- $err(t) = K_v \times (setpt(t) - fb(t))$
 - In general, $K_v = 1$ for textbook PID controller
 - K_v is for ease of overall gain adjustment, for example, it is useful to adjust loop gain without affecting phase in bode plot

- Laplace function : $G_c(s) = \frac{err}{ctrl} = K_p + \frac{K_i}{s} + K_d s$

- Discrete formula

- $ctrl[n] = K_v \times (setpt[n] - fb[n])$
- $ctrl[n] = K_p \times err_p[n] + K_i \times err_i[n] + K_d \times err_d[n]$
 - proportional error : $err_p[n] = err[n]$
 - integral error (Trapezoidal rule) : $err_i[n] = (err[n] + err[n-1]) \times \frac{T_s}{2} + err_i[n-1]$
 - integral error (Rectangle method) : $err_i[n] = err[n] \times T_s + err_i[n-1]$
 - Backward Euler is used in this project
 - derivative error : $err_d[n] = \frac{err[n] - err[n-1]}{T_s}$
 - T_s is sampling period (i.e. = 1 / clock frequency)



Part 1

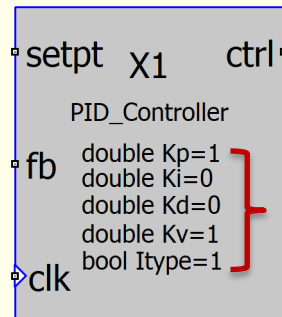
Implementation of PID Controller in C++

PID C++ Implementation

Qspice : pid_controller.qsym ; pid_controller.cpp

- pid_controller.qsym

- I/Os
 - clk : input (↑ trigger)
 - setpt : input
 - fb : input
 - ctrl : output
- Input parameters
 - Kp, Ki and Kd : PID gain
 - Kv : Overall gain
- C++ file
 - pid_controller.cpp



pid_controller symbol
in Qspice

Default Value

- Kp : P Gain=1
- Ki, Kd : No I Gain or D Gain
- Kv : Overall Gain=1
- Itype : Trapezoidal Integration

Itype:
0=rectangular integration
1=trapezoidal integration

Main Code : pid_controller.cpp (written by RDunn)

```
if (clk && !inst->clk_n1)    // rising edge
{
    // time between samples : calculate Tsampling
    double Tsampling =
        t - inst->lastT;    // Tsampling = current time - last rising edge time
    inst->lastT = t;

    // calculate error
    double error = Kv * (setpt - fb);

    // calculate proportional, integral and derivative error
    double errorP = error;

    double errorI;
    if (Itype)    // trapezoidal rule
        errorI = (error + inst->error_n1) * Tsampling / 2 + inst->errorI_n1;
    else    // rectangular method
        errorI = error * Tsampling + inst->errorI_n1;

    double errorD = (error - inst->error_n1) / Tsampling;

    // PID formula
    ctrl = Kp * errorP + Ki * errorI + Kd * errorD;

    // store [n-1] sampling
    inst->error_n1 = error;    // error[n-1] = error[n]
    inst->errorI_n1 = errorI;    // errorI[n-1] = errorI[n]
}

// save clock state
inst->clk_n1 = clk;
```

T_s sampling time is calculated in
code, not a user input parameter

PID C++ Implementation

Qspice : pid_controller.qsym ; pid_controller.cpp

- pid_controller.cpp
 - Trunc()
 - Rising edge will trigger timestep reduce to ttol

```
extern "C" __declspec(dllexport) void Trunc(struct sPID_CONTROLLER *inst, double t, union uData *da
{ // limit the timestep to a tolerance if the circuit causes a change in struct sPID_CONTROLLER
  const double ttol = 10e-12; // 10ps default tolerance
  if(*timestep > ttol)
  {
    struct sPID_CONTROLLER tmp = *inst;
    pid_controller(&(&tmp), t, data);

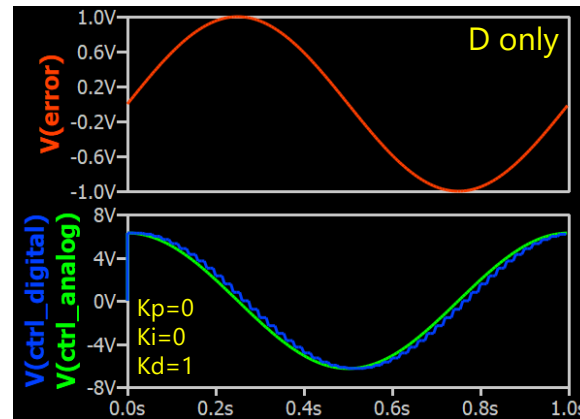
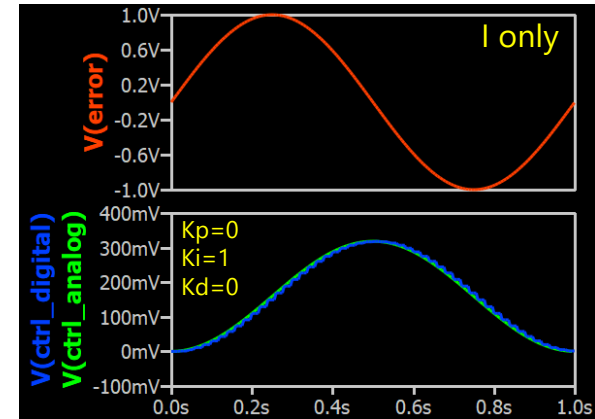
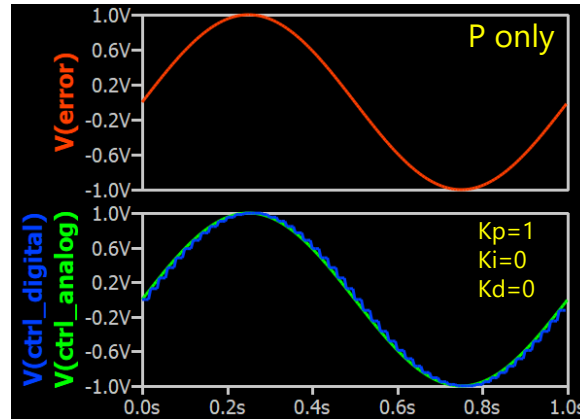
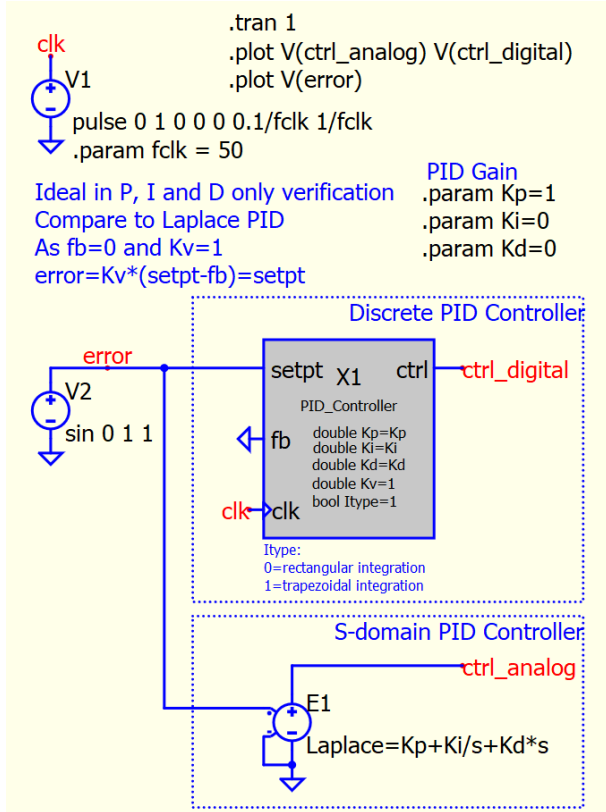
    if(tmp.clk_n1 != inst->clk_n1 & !inst->clk_n1) // clock rising edge reduce timestep to TTOL
      *timestep = ttol;
  }
}
```

Condition AND

- tmp.clk_n1 != inst->clk_n1 : next clock state != current clock state
- !inst->clk_n1 : current clock state is 0
- [therefore, rising edge of clock signal will reduce timestep to TTOL]

PID C++ Controller Algorithm Verification

Qspice : Parent - PID_Controller - Discrete and Laplace.qsch



Conclusion

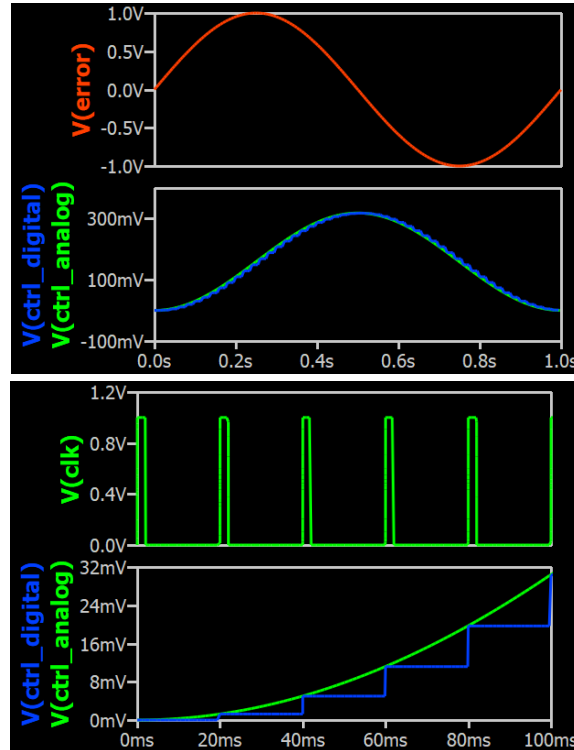
- PID controller C++ block is compared to Laplace E-source in Qspice to verify concept and equation are properly implemented

PID C++ Controller Algorithm Verification

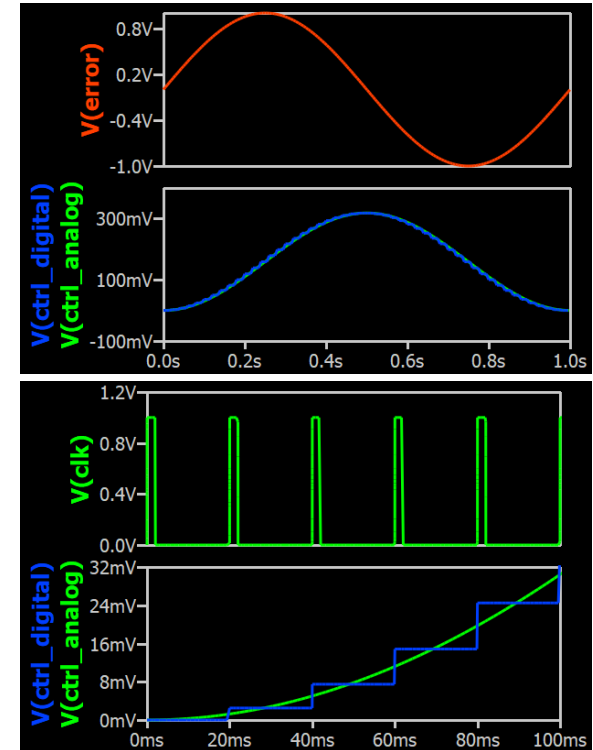
Qspice : Parent - PID_Controller - Discrete and Laplace.qsch

- Integration Method
 - Itype = 0 : Rectangular
 - Itype = 1 : Trapezoidal

1=Trapezoidal Integration



0=Rectangular Integration

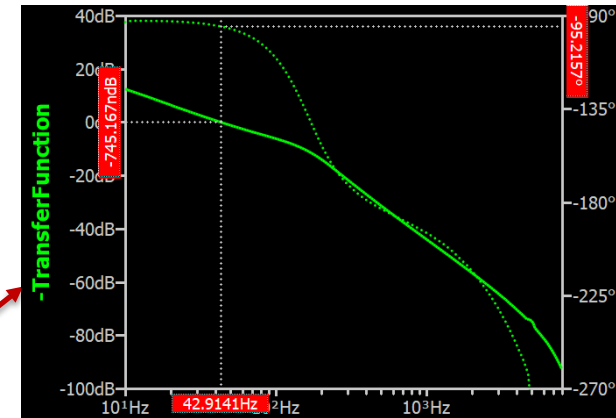
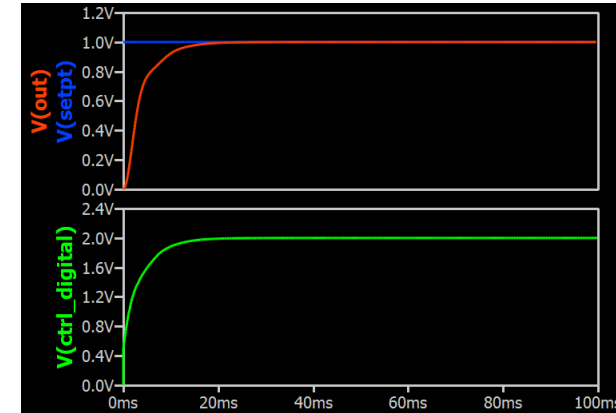
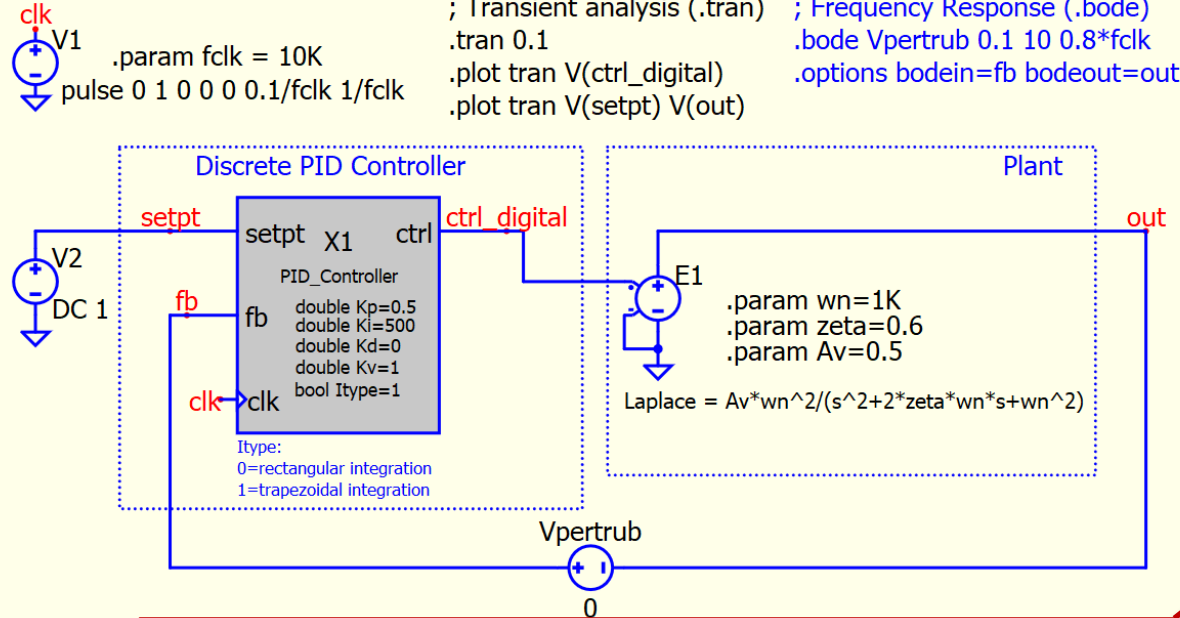


Part 2

Close Loop Examples

Example 1 : Second-Order Laplace System with PID Control

Qspice : Parent - PID_Controller - Second-Order Laplace.qsch



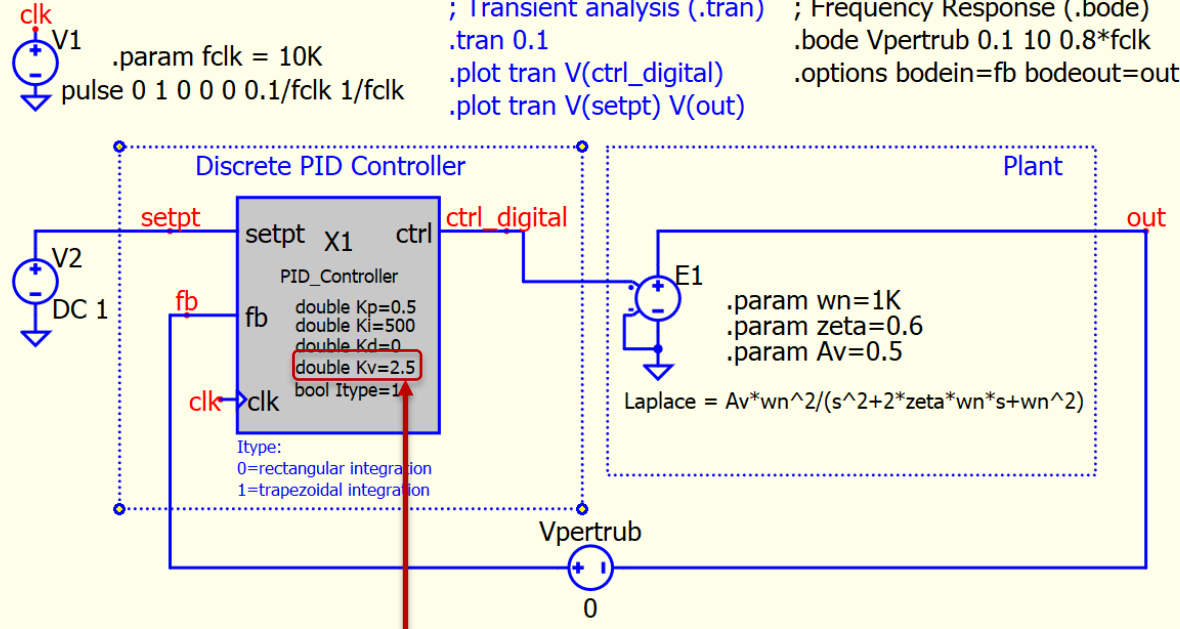
Why -ve TransferFunction is used (use +ve or -ve depends on your practice)

- TransferFunction = bodein/bodeout = fb/out
- $\text{GH}(s) = -\text{fb}/\text{out} = -\text{TransferFunction}$

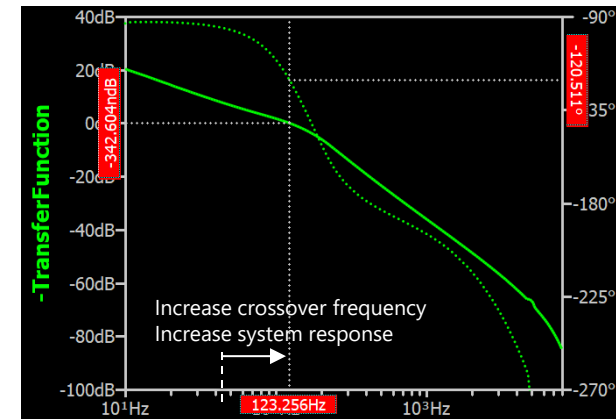
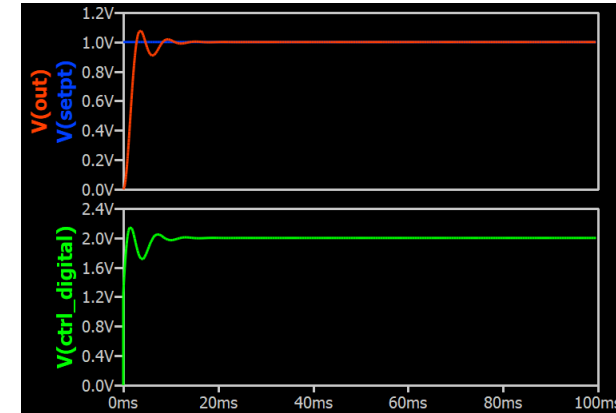
But discussion of .bode is not the scope of this presentation

Example 1 : Second-Order Laplace System with PID Control

Qspice : Parent - PID_Controller - Second-Order Laplace.qsch



Tune system response by adjusting overall gain (Kv) from 1 to 2.5
Therefore, Gain increased by $20 \log_{10}(2.5) = 7.96\text{dB}$



Example 2 : Close Loop control of a Buck Converter

Qspice : Parent - PID_Controller - Buck Converter.qsch

- Close Loop Control of Buck Converter
 - This is a demo to use PID controller for switching mode power supply
 - Sawtooth frequency set to 50kHz and controller sampling frequency set to 1MHz (20 times of switching action)
 - As this is an ideal discrete controller, if Vsetpt reduce Vsetpt rise time, V(ctrl) will shoot up further as derivative error $err_d = \frac{\Delta error}{\Delta t}$, and if rise time tends to 0s, $err_d \rightarrow \infty$ and give a huge V(ctrl)
 - A voltage limiter can be added at PID output to restrict its min and max output voltage

Close Loop Feedback for Buck Converter with PID Controller

```
.tran {100/fsawtooth}
.plot tran V(setpt)*scale V(out)
.plot tran V(gate)
.plot tran V(ctrl)
```

```
.param fsawtooth = 50K
.model IdealSW SW (Ron=1m Roff=100Meg Vt=0.5 Vh=0)
.model IdealD D (Ron=1m Roff=100Meg Vfwd=0)
```

