Control and Optimization

sT-Embed Training

Ric Kolk
Altair Engineering
rkolk@altair.com

Topics:

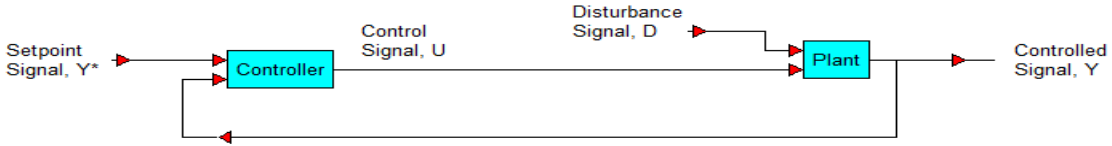
- Basic Classical Controllers
  - Proportional Integral (PI)
  - Proportional Derivative (PD)
  - Proportional Integral Derivative (PID)
- Using sT-Embed Optimize to tune a PID controller
- Controller Function Blocks
  - Merge & Crossdetect Blocks
  - Discrete Reset Integrator
  - Countdown Timer with Underflow Protection
  - Pulse Counter
  - Binary Signal Conditioning

Proportional Integral (PI), Proportional Derivative (PD), and Proportional Integral Derivative (PID) Controllers

Controllers - Basics

Controllers are systems designed to modify and maintain a systems (Plant) performance when subjected to unmeasurable disturbances.

Control System = Controller + Plant



Basic Control System Block Diagram & Signal Terminologies

Performance is generally concerned with:

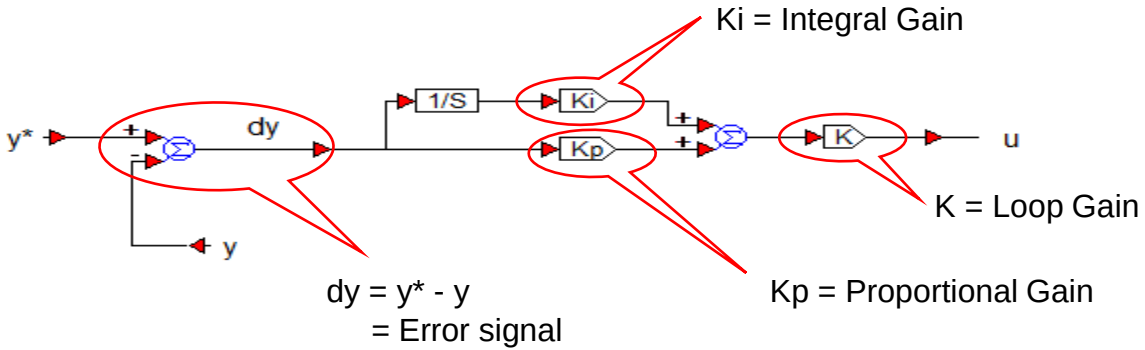
- Accuracy: How well the Controller maintains Y=Y* with D present.
- Transient: How well the Controller shapes the trajectory of Y as changes are made in Y* with D present.

Three Basic type of Controllers:

- Proportional – Integral (PI) Controller is used to achieve Accuracy Performance
- . Proportional – Derivative (PD) Controller is used to achieve Transient Performance
- Proportional – Integral – Derivative (PID) Controller is used to achieve Both

Proportional – Integral (PI) Controller

PI controllers regulate a plant output, “y”, to a constant setpoint value, “y*” with disturbances present.



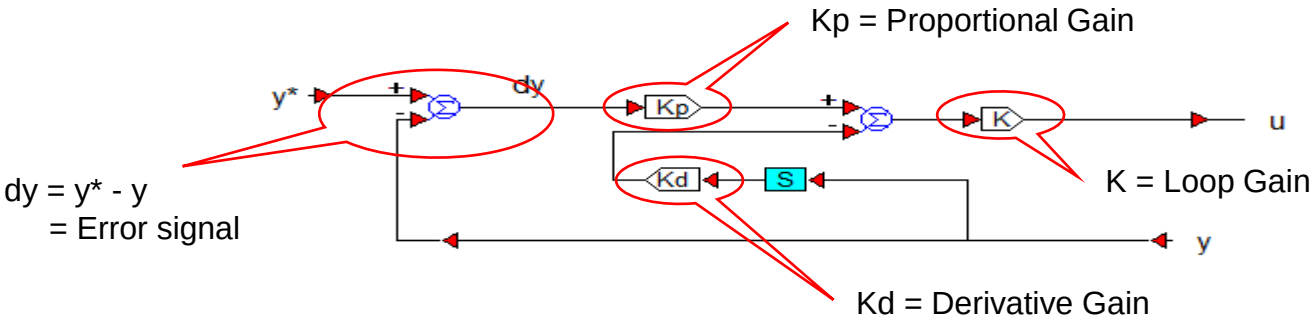
PI Controller Tuning:

Step 1. Ki=0.1; Kp = 1; K = 1
Step 2. Apply PI controller to plant, adjust K for acceptable response

Typical Performance Metric: Maintain “dy” to remain within predefined limits for any disturbance input in steady state, maintain “u” to remain within predefined limits.

Proportional – Derivative (PD) Controller

PD controllers shape the transient behavior of a plant output, “y”, as the setpoint is varied and with disturbances present.



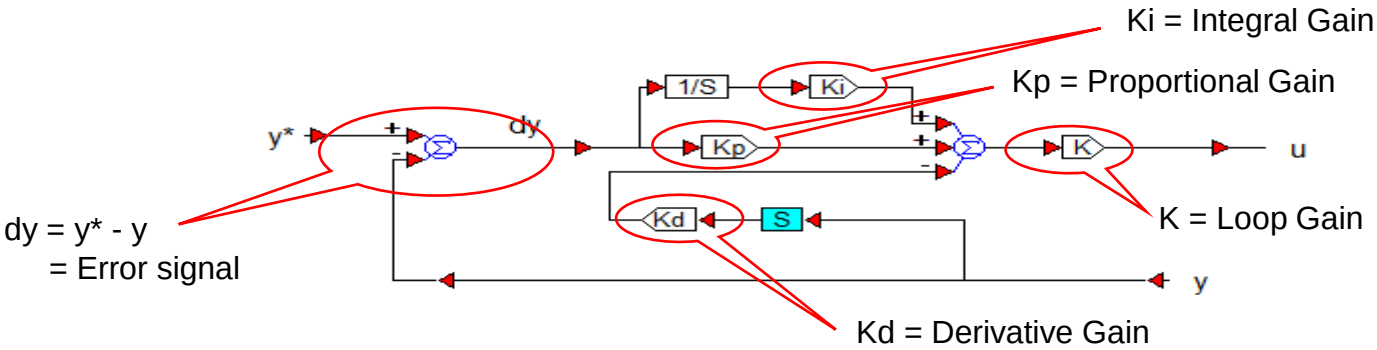
PD Controller Tuning:

Step 1. Kd=a value; Kp = 0; K = 1
Step 2. Apply PD controller to plant, adjust Kd and K for acceptable transient response, then add Kp gain for accuracy

Typical Performance Metric: Control the “Percent Overshoot” and “Time Constant” to remain within predefined ranges, maintain “u” to remain within predefined limits.

Proportional-Integral-Derivative (PID) Controller

PID controllers regulate the plant output, y, at a setpoint value, y*, and shape the transient behavior of the plant output as the setpoint is varied and with disturbances present.



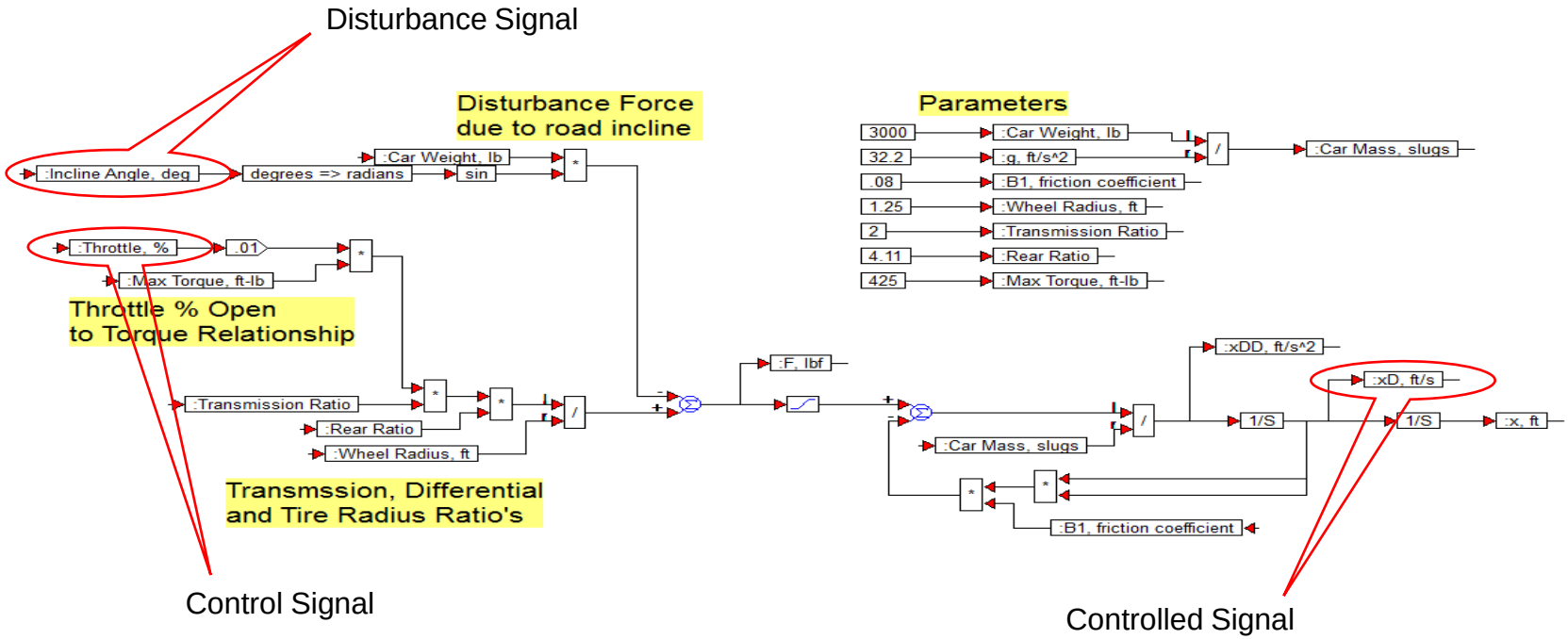
PID Controller Tuning:

Step 1. Kd=0; Kp = 1; Ki = .1; K = 1
Step 2. Apply PID controller to plant, adjust Kd and K for acceptable response

Typical Performance Metrics: Maintain “dy” to remain within predefined limits for any disturbance input in steady state, maintain “u” to remain within predefined limits, control the “Percent Overshoot” and “Time Constant” to remain within predefined ranges.

Automobile PI Speed Control (1/3)

This example presents an automobile model and the design and simulation of a PI speed controller.

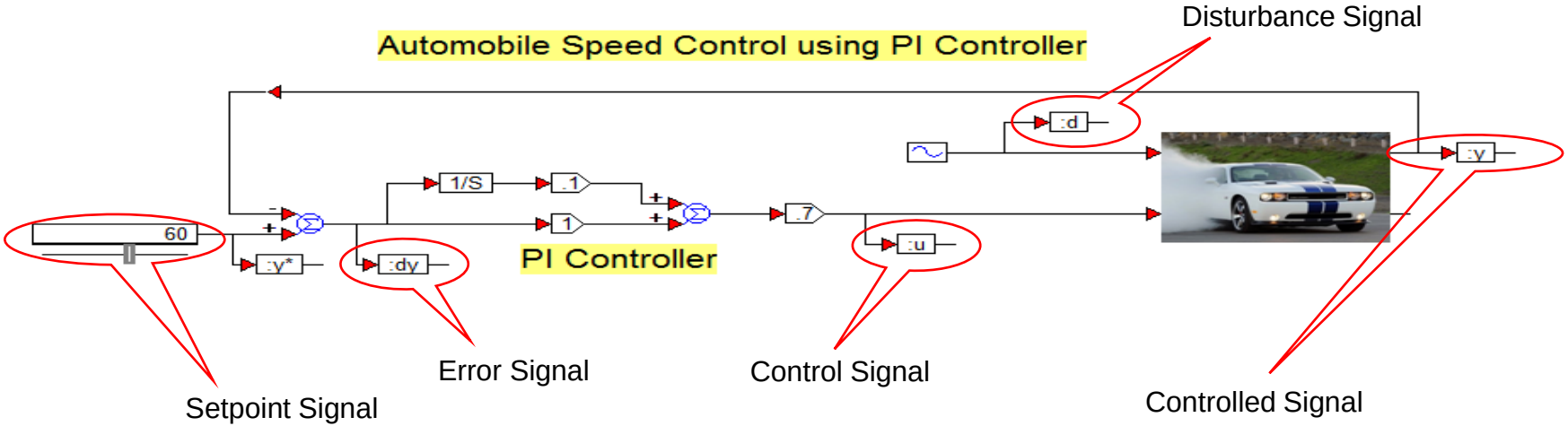


PI Control - Plant Model

Automobile PI Speed Control (2/3)

Performance Requirements: At y* = 60 mph, maintain car speed within +/- 3 mph, limit throttle % open to < 10% variation, Incline disturbance (worst case): +/- 1 degree/20 seconds, maximum 5degrees.

Control System = PI Controller + Plant Model

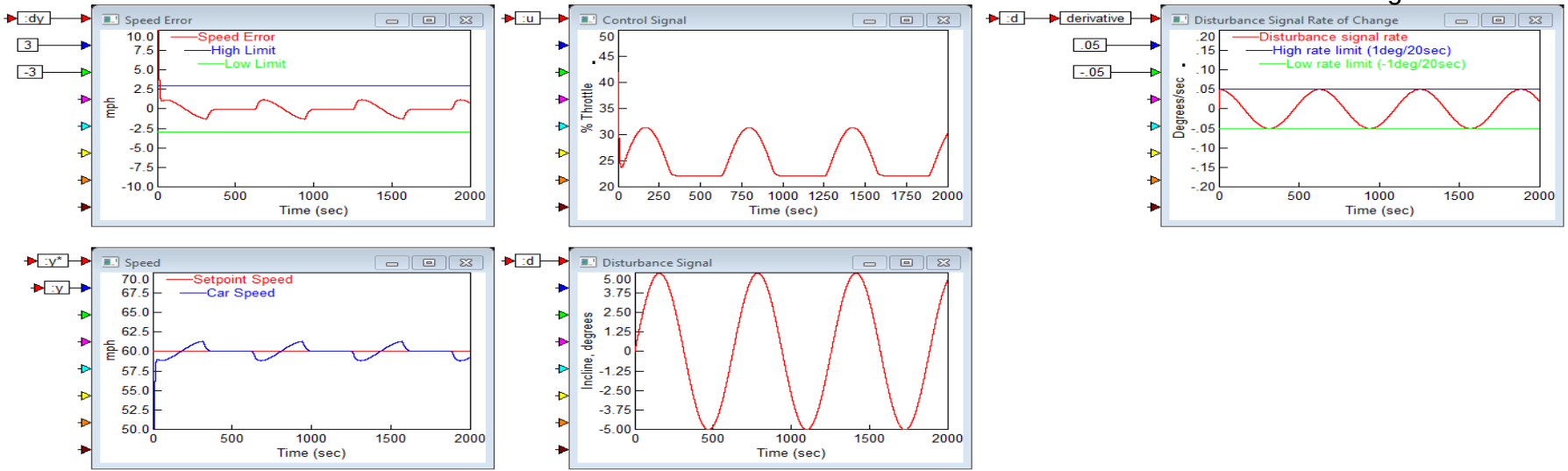


Automobile PI Speed Control (3/3)

Performance Requirements & Responses:

Throttle variations are < 10%

Incline disturbance ranges from + to - .05deg/sec



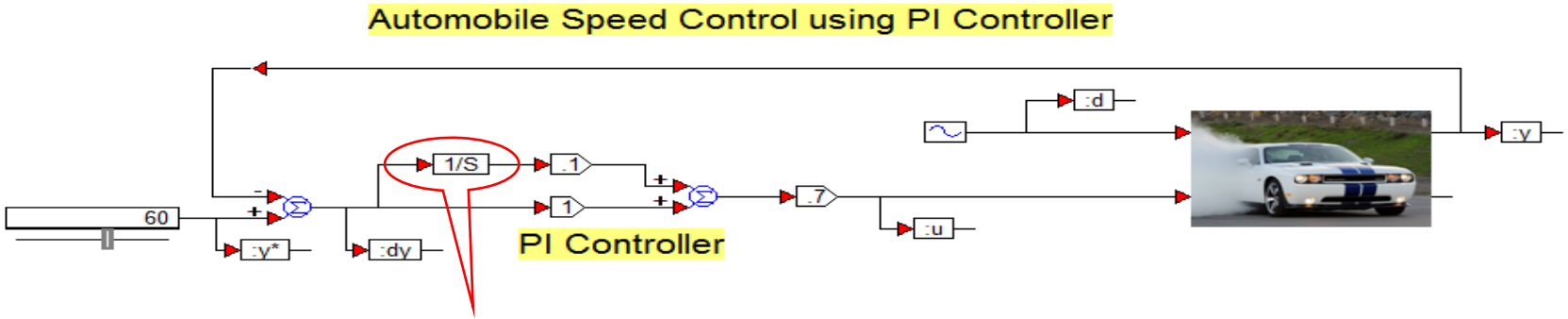
Speed is maintained within +/- 3 mph at 60 mph setpoint

Automobile PI Speed Control System

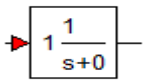
Discrete Automobile PI Speed Control (1/3)

The PI Speed Control developed in the previous example is to be programmed on an embedded processor. Prior to this, a discrete version of the PI controller is developed and its performance is compared with the PI Speed Control. The Discrete PI shall be executed 1000x/sec and shall meet the same performance requirements.

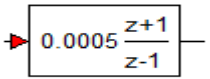
Control System = PI Controller + Plant Model



1. Rewrite as a transfer function "Blocks/Linear System/transferFunction"

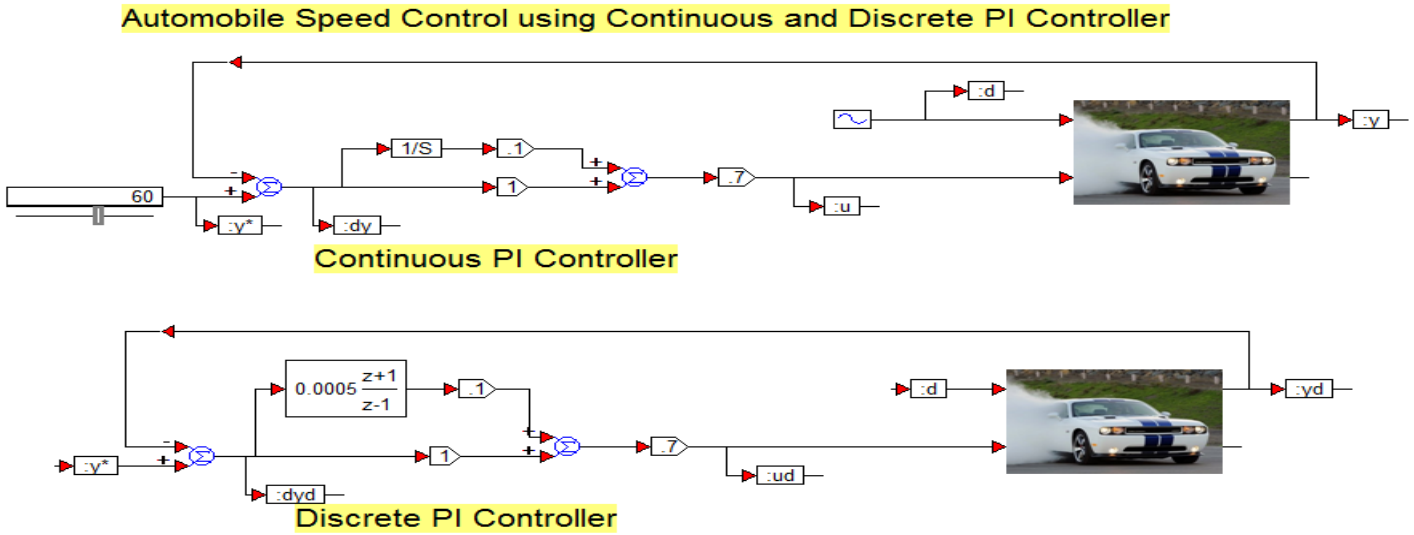


2. Apply "Convert s->z" to get discrete transfer function



Discrete Automobile PI Speed Control (2/3)

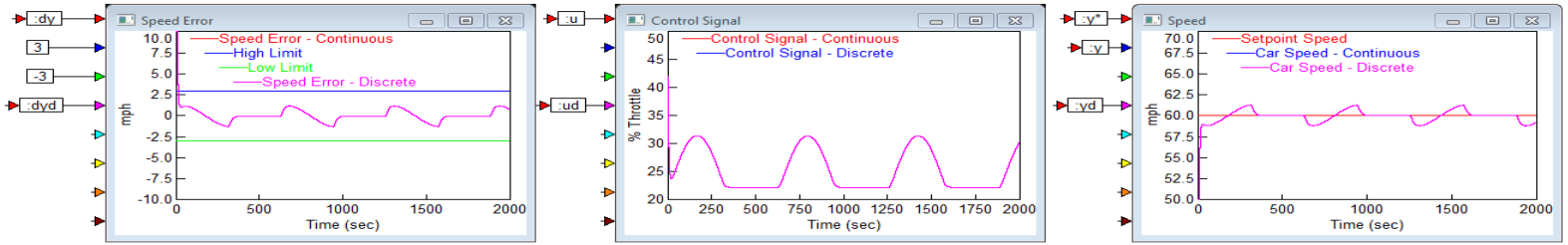
The continuous and discrete PI Speed Control are modeled “side by side”:



Discrete Automobile PI Speed Control System

Discrete Automobile PI Speed Control (3/3)

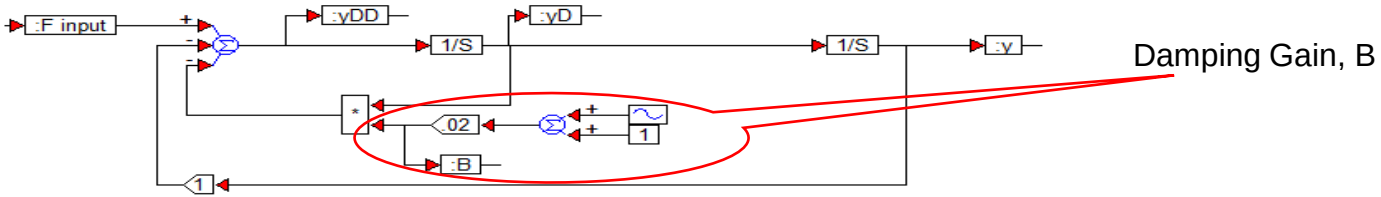
The continuous and discrete PI Speed Control are simulated with “Step Time” = .001 and “End” = 2000.



Structure PD Damping Control (1/3)

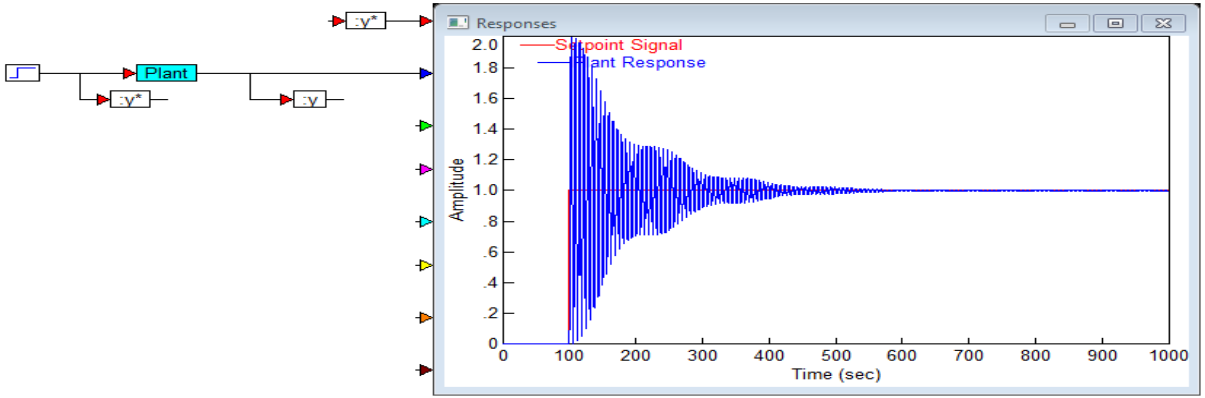
This example Illustrates the design and implementation of a PD controller. The structure has a damping coefficient, B, that varies between 0 - .04 randomly.

Plant Model with Damping Gain, B, modeled as a sinusoidal variation: $B = .02(1 + \sin(.05t))$



Damping Gain, B

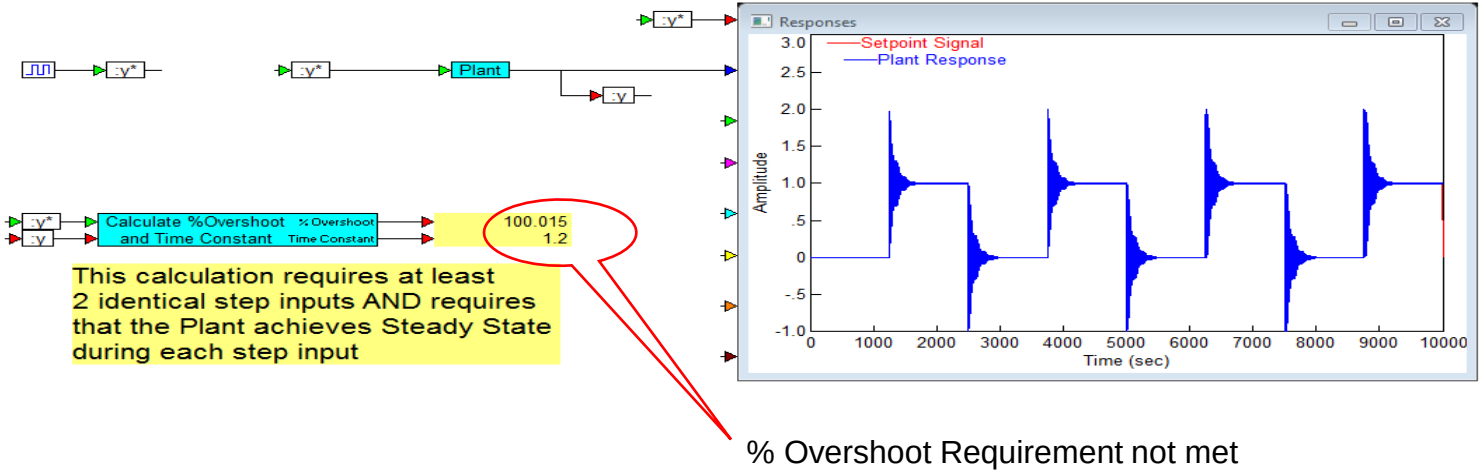
Plant Model Unit Step Response:



Structure Damping - Plant Model

Structure PD Damping Control (2/3)

Performance Requirements: Step response percent overshoot <= 3%, time constant <= 1.5 sec, control signal to remain within +/- 1, Damping disturbance (worst case): B = 0 to 0.04 varying at .05rad/sec.

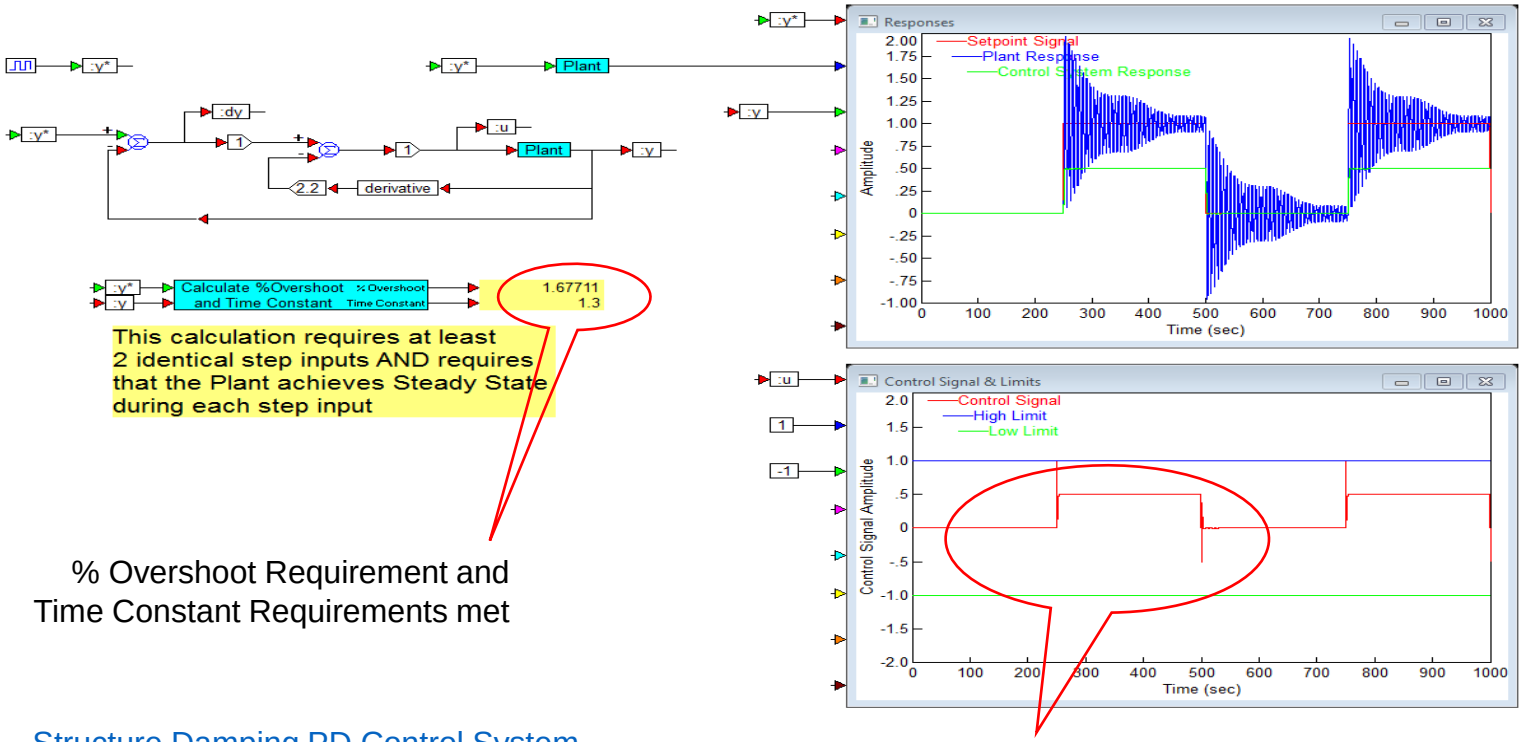


% Overshoot Requirement not met

[Structure Damping - Requirements Modeling]

Structure PD Damping Control (3/3)

Performance Requirements and Responses:



% Overshoot Requirement and Time Constant Requirements met

Control Signal Requirements met

[Structure Damping PD Control System]

Using the sT-Embed Optimizer to tune a PID controller

EMBED™
solidThinking

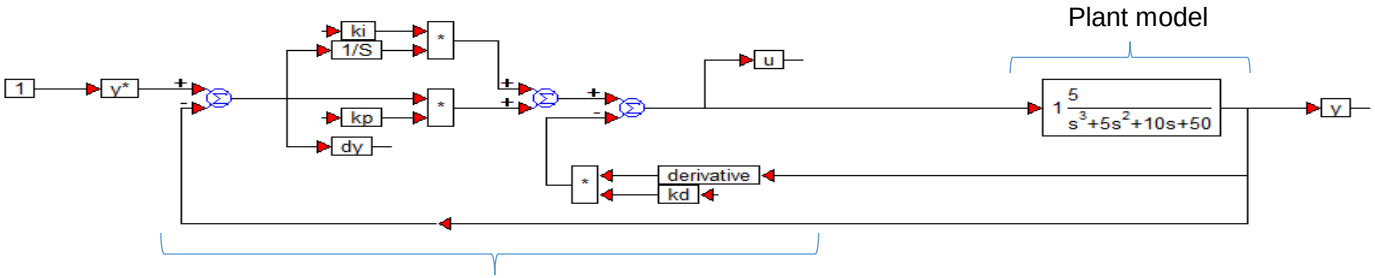
System PID Control Design using Optimization (1/3)

This example Illustrates the design and implementation of a PID controller using the sT-Embed Optimization feature.

The Plant model is represented by the transfer function: $T(s) = \frac{y(s)}{u(s)} = \frac{5}{s^3 + 5s^2 + 10s + 50}$

Performance Requirements: Unit step response has 0-steady state error for time >= 3 seconds from application of the input signal, the output achieves 80% of its final steady state value at time = 1.5 seconds from application of the input signal, and the Control System is stable.

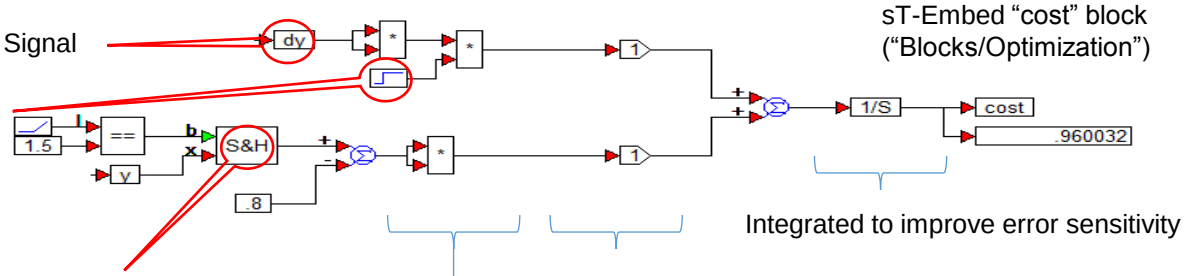
Solution: A PID controller is applied with the gains [ki, kp, kd] evaluated by minimizing a quadratic cost function based on the requirements. The Control System block diagram, with signal labels, is;



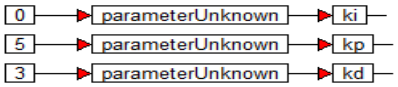
Open Loop Plant Step Response

System PID Control Design using Optimization (2/3)

The cost function to be minimized is defined as a block diagram using signals from the Control System and the requirement information.

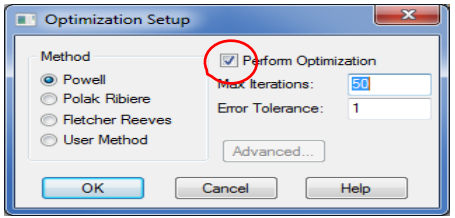


The unknown parameters are the PID gains; [ki, kp, kd]. They are specified using “parameterUnknown” blocks (“Blocks/Optimization”) and with the initial “guess” values (right).

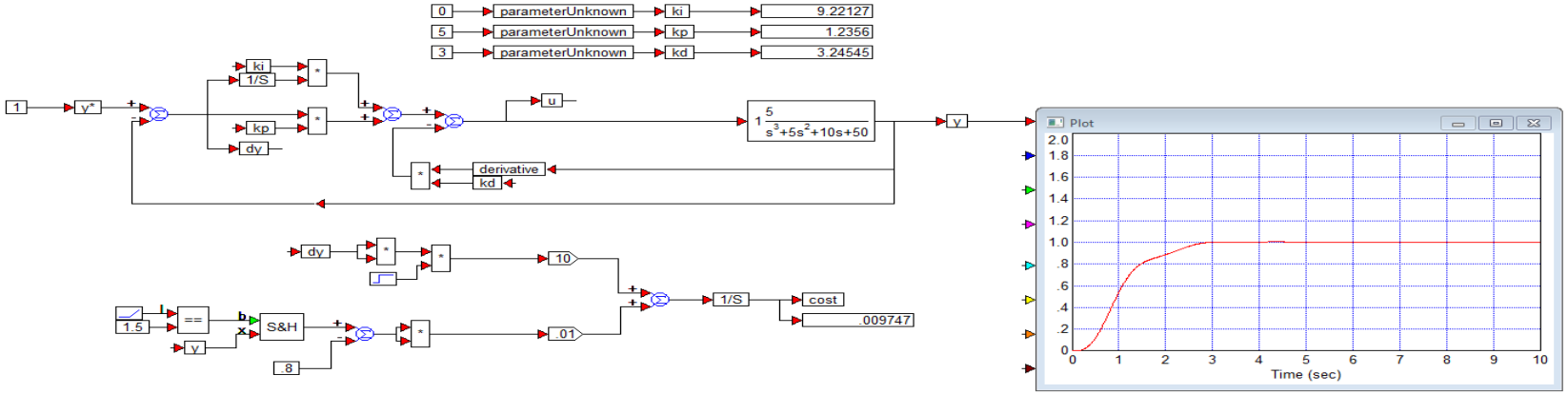


System PID Control Design using Optimization (3/3)

The “Perform Optimization” option is selected in the “Blocks/Optimization” menu. The “Method” and other values are left at their defaults.



Results:



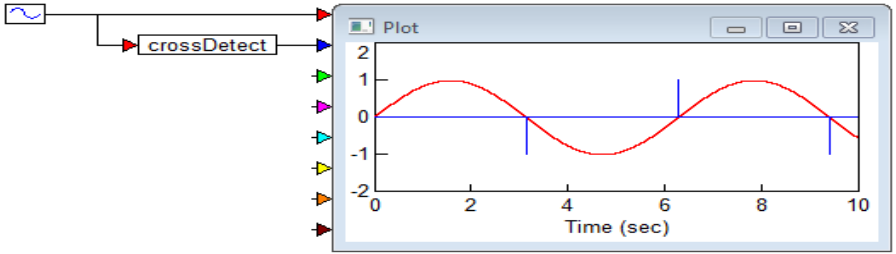
[PID Controller Adjustment using Optimization]

Controller Function Blocks

merge and crossDetect Blocks

The “merge” block (“Blocks/nonlinear”) provides the “if-then-else” function. If the Boolean “b” input is true (1), then the output is set to the “t” input, otherwise the output is set to the “f” input.

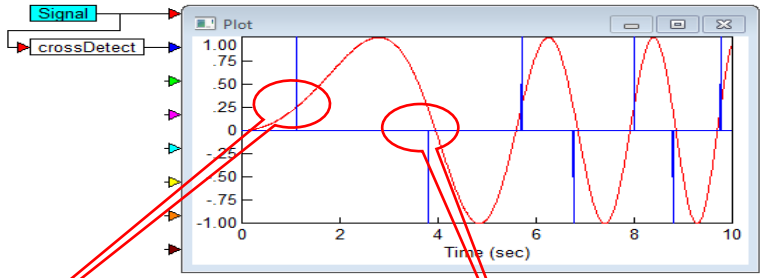
The “crossDetect” block (“Blocks/nonlinear”) produces a +1 pulse when a signal passes through a “cross point” from below to above and a -1 pulse when a signal passes through a “cross point” from above to below. In the example below, the cross point = 0



CrossDetect, Limit, and ABS Blocks

The “pulseTrain” block is used to produce pulses with a constant “Time Between Pulses”.

The “crossDetect” block (“Blocks/Nonlinear”) may be used to produce pulse train signals with variable time between pulses. The behavior of the “crossDetect” block, configured with its “Cross Point” = 0.25, is illustrated in the following block diagram.



“crossDetect” produces a high positive pulse when the signal transitions from **less** than the “Cross Point” to **greater** than the “Cross Point”

“crossDetect” produces a high negative pulse when the signal transitions from **greater** than the “Cross Point” to **less** than the “Cross Point”

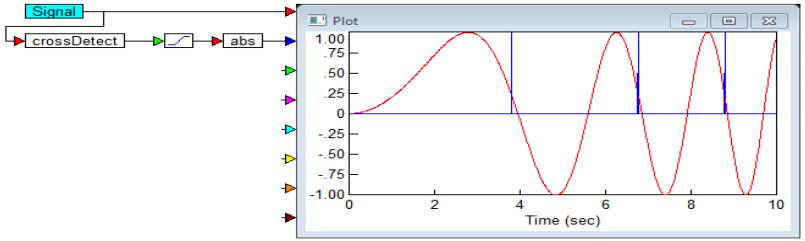
.

[Cross-Detect Example]

.

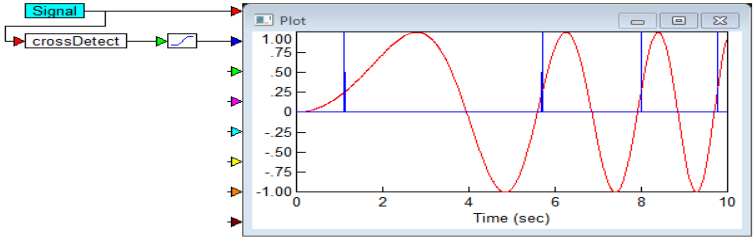
CrossDetect, Limit, and ABS Blocks

To retain only the negative pulse and convert it to a positive pulse, the “**limit**” block (“Blocks/Nonlinear”) configured with “Lower Bound” = -1 and “Upper Bound” = 0 and the “abs” block (“Blocks/Arithmetic”) may be used



.

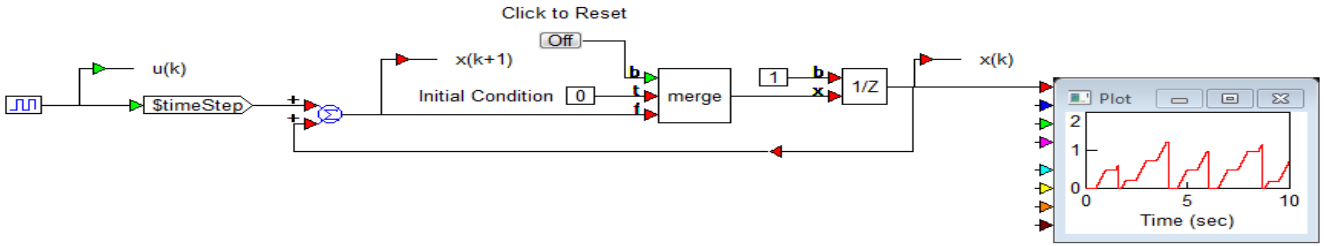
Similarly, to retain only the positive pulse the “**limit**” block configured with “Lower Bound” = 0and “Upper Bound” = 1 may be used



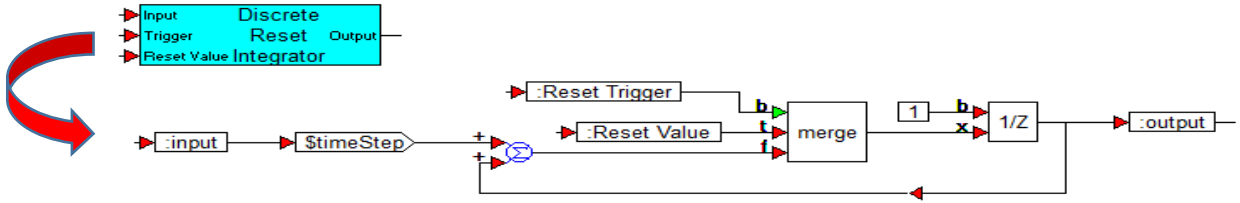
Cross-Detect Example

Discrete Integrator with Reset

A discrete integrator with reset to a specified initial condition can be created by adding the “merge” block to the discrete integrator previously described.



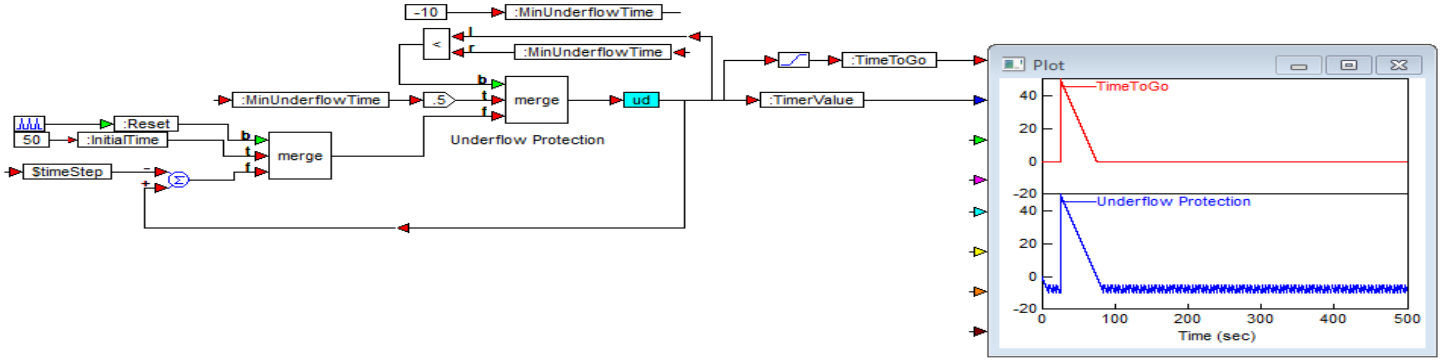
A digital integrator compound block with Pin labels could be created as:



[Discrete Reset Integrator Example]

CountDown Timer with Underflow Protection

The “CountDown” Timer outputs the remaining time to reach 0 seconds from an “InitialTime”. It begins counting down when a positive pulse is applied to the “Reset” input. Although the “CountDown” Timer output, “TimeToGo”, is limited to a lower bound of 0, the raw timer signal, “TimerValue” will continue counting down below 0. If unchecked, this can cause an underflow problem. To prevent this, underflow protection logic is added.



Underflow Protection: When the “TimerValue” decreases to less than "MinUnderflowTime", the timer is reset to 0.5* "MinUnderflowTime". In this example, "MinUnderflowTime" is set to -10, so once the “TimerValue” reaches -10, it will rest to -5 and continually count down from -5 to -10.

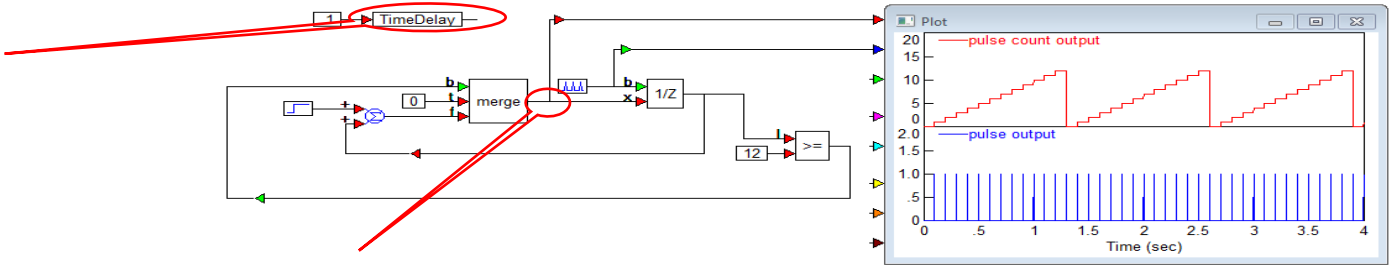
[Count Down Timer Example]

Pulse Counter

The “merge” block (“Blocks/Nonlinear”) is used to model “IF THEN ELSE” logic. The block has three inputs; “b” = boolean, “t” = true, “f” = false, and one output. When the boolean “b” input is 1, the output is set to the true “t” input and when the boolean “b” input is not 1, the output is set to the false “f” input.

In the following example block diagram, the “unitDelay” block is used to count the number of pulses produced by a “pulseTrain” block configured with “Time Between Pulses” set to 0.1 seconds and “Time Delay (sec)” set to the global variable “TimeDelay”. When the count reaches 12, the “merge” block is used to reset the “unitDelay” initial condition to 0.

The “TimeDelay” value is applied to both the “step” and “pulseTrain” blocks for correct initial condition counter response..



The “pulse count output” is taken upstream of the “unitDelay” to avoid a delay in the count value.

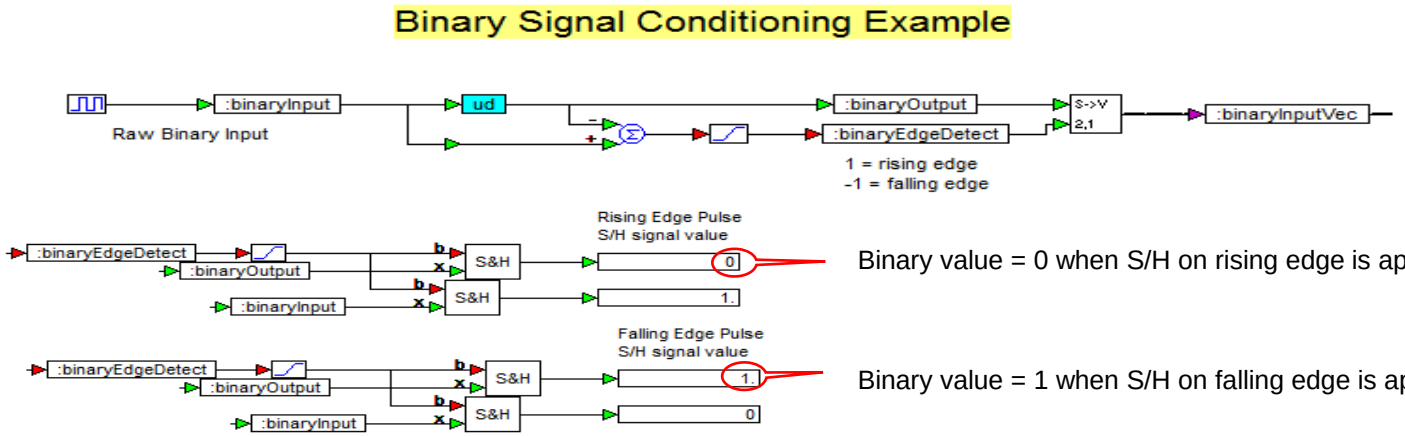
The “pulse count output” is reset to “0” on each 13th pulse and then counts up to 12 pulses.

Pulse Counter Example

Binary Signal Conditioning

When using binary signals in a control algorithm it is sometimes necessary to detect rising and falling edge times. Edge times are calculated using unit delays and backward differencing to produce a positive pulse for a rising edge and a negative pulse for a falling edge.

When using the pulses to Sample/Hold, it is sometimes necessary to ensure the pulse occurs before the binary signal state has changed. The following model accepts a raw binary input signal and outputs a 2x1 vector with element 1 = binary signal value (0 or 1) delayed by 1DT and element 2 = edge pulse (+1 for rising edge, -1 for falling edge).



Binary value = 0 when S/H on rising edge is applied

Binary value = 1 when S/H on falling edge is applied

[Binary Signal Conditioning Example]

End of Section

EMBED™
solidThinking