

Testing of Safety Critical Control Systems

YOGANANDA JEPPU



Copyright Notice

Testing of Safety Critical Control Systems by Yogananda Jeppu is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

You are free:

to Share — to copy, distribute and transmit the work

to Remix — to adapt the work

Under the following conditions:

Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Noncommercial — You may not use this work for commercial purposes.

Share Alike — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

For details please visit the website.



Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by-nc-sa/3.0/>

Disclaimer

The views, methods are my experiences working in the field for some time. What I have experienced may not be applicable to your application. Use this presentation as a knowledge gain but USE YOUR JUDGMENT!

I have used pictures and materials available in Google. I have not collected the references every time. In case anyone feels that I have used their published material and not referenced it here please feel free to mail me. I will put in the reference.

The firm/company I work for does not endorse these views which are mine.

Acknowledgements

I would like to thank George Romanski for his critical review comments on the earlier presentation. I have updated this version based on his review comments.

I thank Gorur Sridhar whose comments I have incorporated in this version.

Chethan C U has generated a set of models and Matlab code based on this material and it is available for download on the MathWorks website.
<http://www.mathworks.com/matlabcentral/fileexchange/39720-safety-critical-control-elements-examples>)

I have added a few slides for coverage based on the presentation in MathWorks conference by Chethan CU. I have indicated these with a
(c) chethan.cu@gmail.com

I would like to acknowledge Natasha Jeppu's help in the formal methods section.

Clarification

Many of you have appreciated this presentation over the months. I am happy that this presentation has been useful to you.

This presentation is made so that people in this field use it. I have seen the same mistakes being made again and again. This presentation will help you avoid these mistakes. Please go ahead and make new ones and we as a community can learn from this.

I get mails asking if they can use it in their organizations. Please feel free to use it.

Drop me a mail with suggestions to improve it yvjeppu@gmail.com. I will always appreciate it.

Key Takeaway

An insight into the fascinating field of Model Based Testing of Safety Critical Control Systems

An insight into the mistakes we make – again and again

A set of best practices in this field gleaned from the use of this type of testing on aircraft programs in India



The Major Driver for Me



Unintended Acceleration and Other Embedded Software Bugs

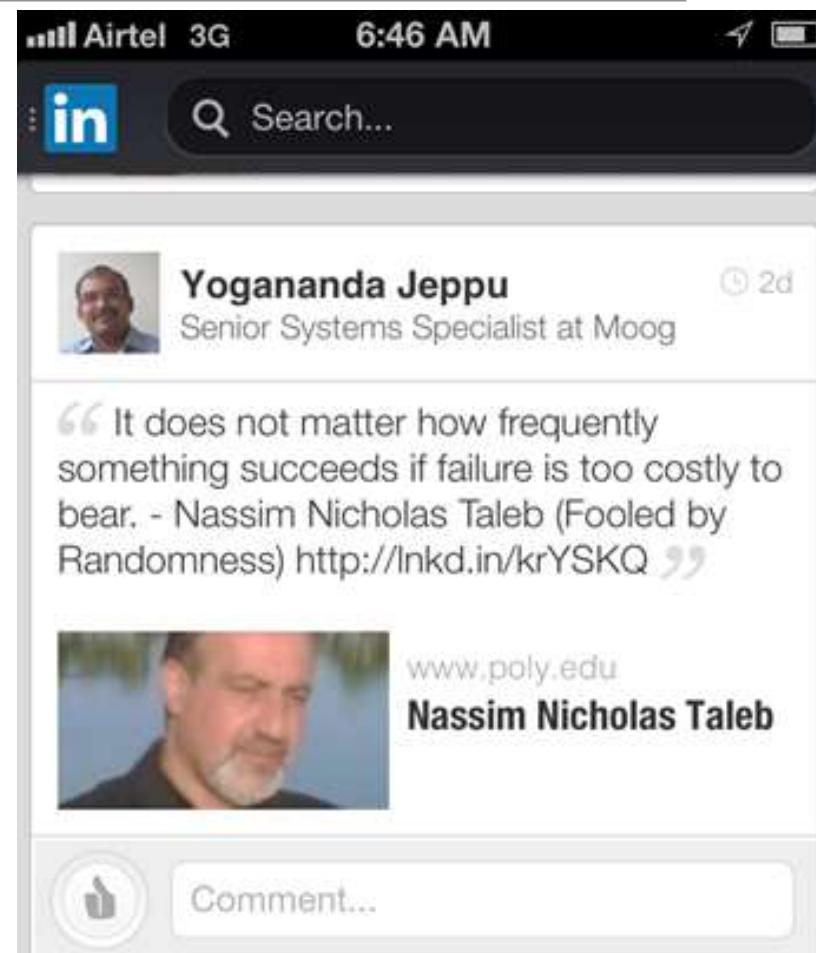
Tuesday, March 1st, 2011 by Michael Barr

Last month, [NHTSA](#) and the [NASA Engineering and Safety Center \(NESC\)](#) published reports of their joint investigation into the causes of unintended acceleration in Toyota vehicles. NASA's multi-disciplinary NESC technical team was asked, by Congress, to assist

Something I posted on LinkedIn

**“It does not matter
how frequently
something succeeds if
failure is too costly to
bear”**

- Nassim Nicholas Taleb



Presenter



Background

I am Yogananda Jeppu. I have a PhD in model based safety critical control system testing. I have 28 years experience in control system design, 6DOF simulation, Model Based Verification and Validation, and system testing.

I have worked on the Indian Light Combat Aircraft (LCA) control system and the Indian SARAS aircraft. I have worked on model based commercial aircraft flight control law programs of Boeing, Airbus, Gulfstream and Comac. I like teaching and interacting with students trying to provide an industrial view of systems development to them.

Currently I am working at Honeywell Technology Solutions, looking at Model Based Development and Formal Methods across various domains. I am responsible for training in this subject and incorporating these techniques in the ongoing projects.

Topics

Safety Critical Control Systems – A brief overview

What are the mistakes we normally make? – a look at the errors made in the various programs since 1988

DO178B, DO178C and DO331 standards overview. How are other standards related?

What are these models? – a look at how they function

- Algorithms for implementing them

How do we test these blocks? – a block by block approach

What are control theoretic coverage metrics?

Formal Methods in Flight Controls – An experimental approach

Some modeling guidelines

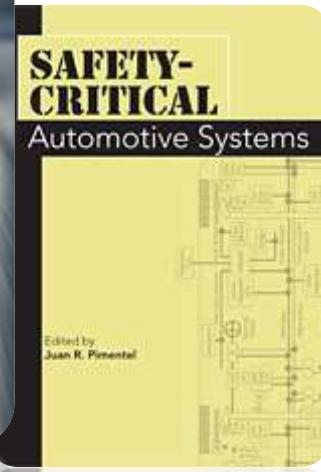
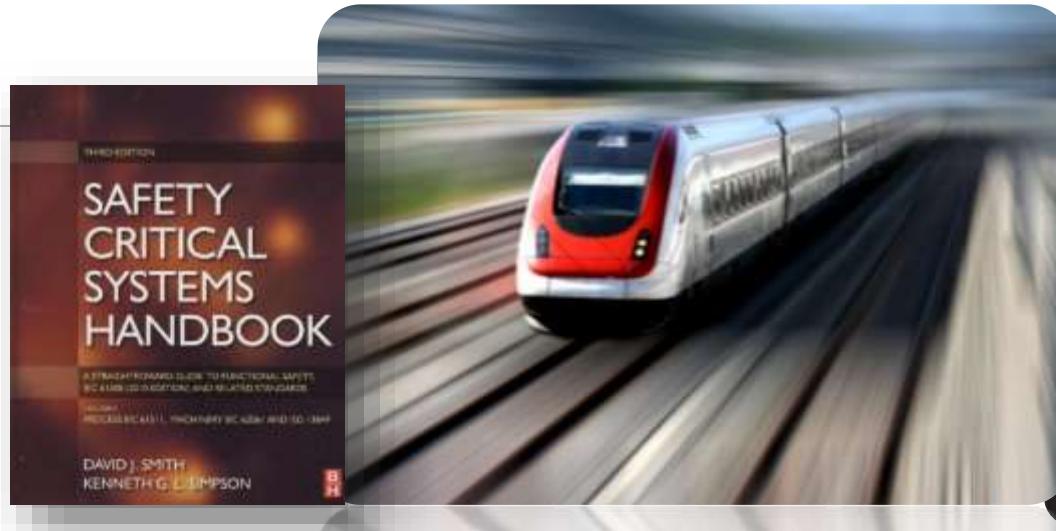
Best practices

Tips

I am providing tips like these as we go along and hope that it will be useful to you.

Tips are my experiences and will help you to get a head start. I hope these will be useful.

Safety Critical Applications



Safety Critical Control Systems

Safety Critical Application:

- An application where human safety is dependent upon the correct operation of the system

Examples

- Railway signaling systems
- Medical devices
- Nuclear controllers
- Aircraft fly-by-wire system
- And now - the automotive domain

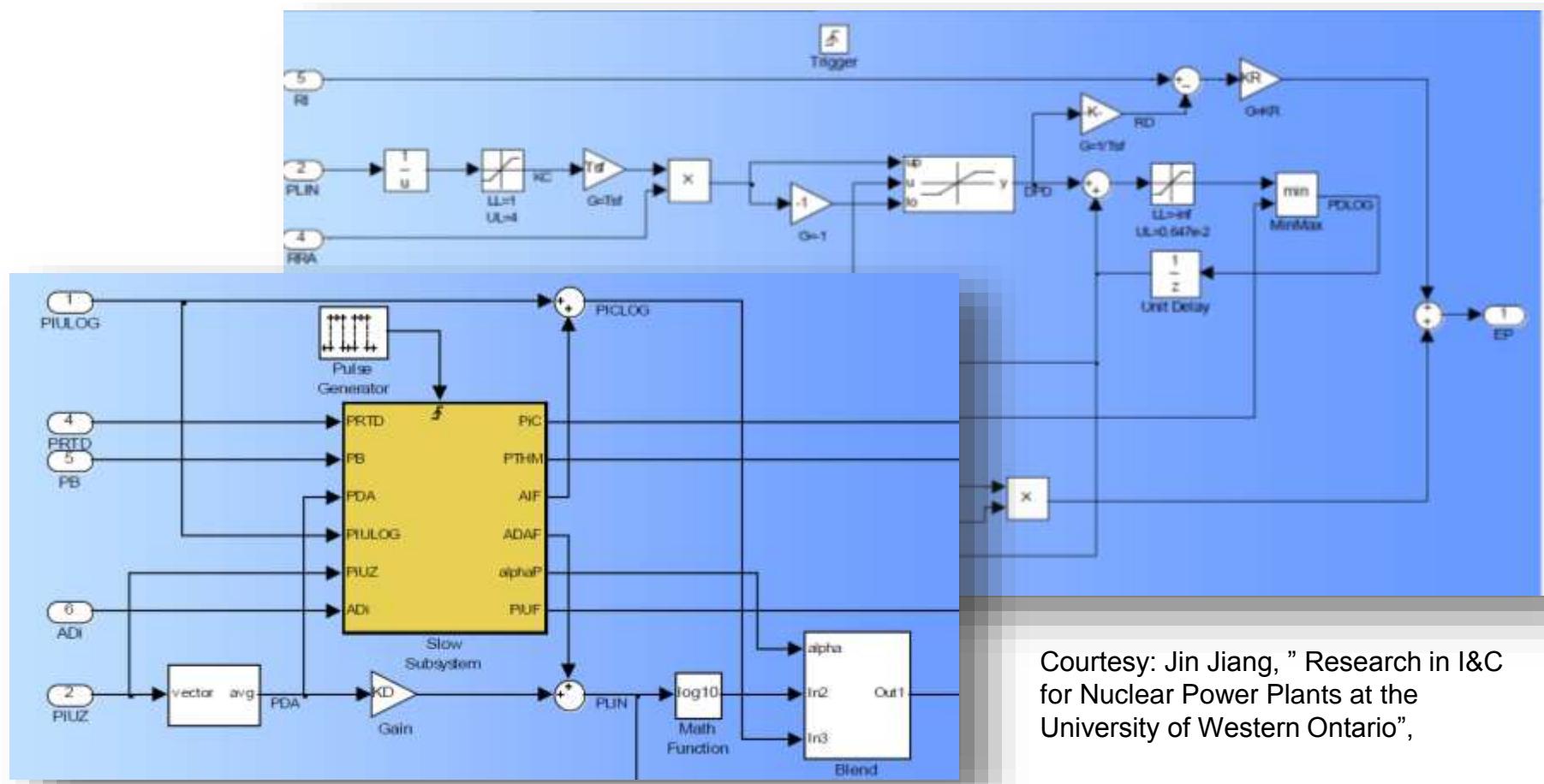
Railway Signaling Systems



Andreas Gerstinger, "Safety Critical Computer Systems - Open Questions and Approaches",
Institute for Computer Technology
February 16, 2007

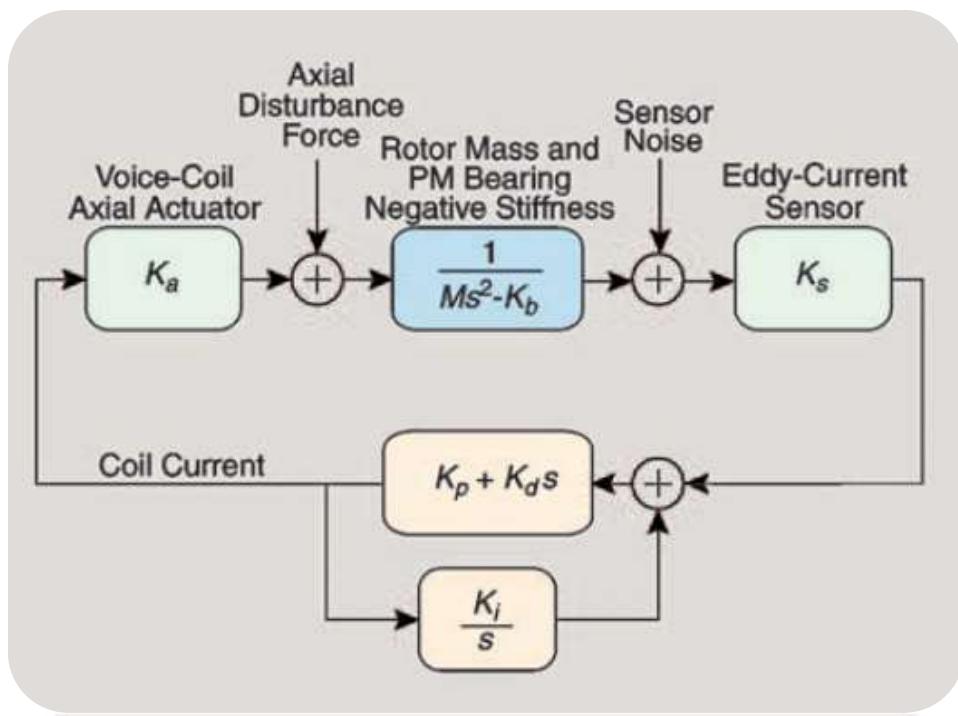


Reactor Core Modeling



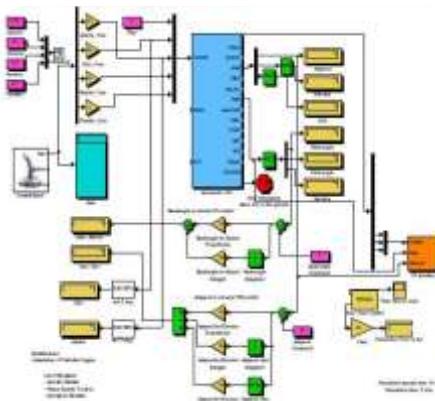
Courtesy: Jin Jiang, "Research in I&C for Nuclear Power Plants at the University of Western Ontario",

Streamliner Artificial Heart



James Antaki, Brad E. Paden, Michael J. Piovoso, and Siva S. Banda, "Award Winning Control Applications", IEEE Control Systems Magazine December 2002

Fly-by-Wire

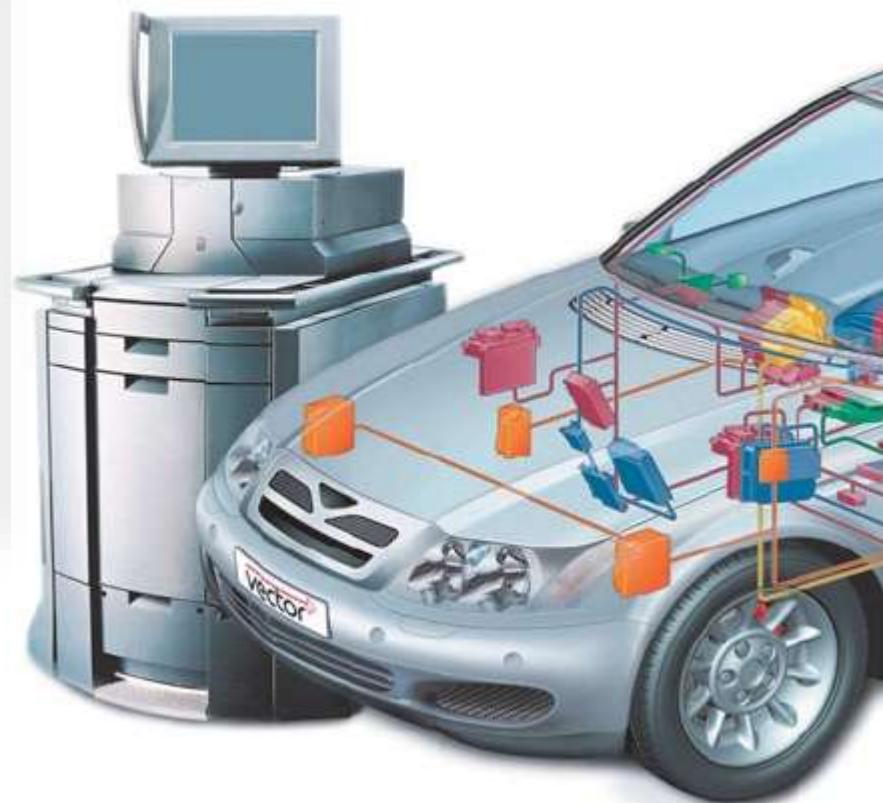


The F-8 Digital Fly-By-Wire (DFBW) flight research project validated the principal concepts of all-electric flight control systems.



<http://www.dfrc.nasa.gov/Gallery/Photo/F-8DFBW/HTML/E-24741.html>

Programmable ECUs



Peter Liebscher, "Trends in Embedded Development", http://www.vector-worldwide.com/portal/medien/cmc/press/PSC/TrendsEmbedded_AutomobilElektronik_200602_PressArticle_EN.pdf

Safety Standards

ISO9001 – Recommended minimum standard of quality

IEC1508 – General standard

EN50128 – Railway industry

IEC880 – Nuclear industry

RTCA/DO178B – Avionics and Airborne Systems

- Updated to DO178C in 2011

MISRA, ISO 26262 – Motor industry

Defense standard 00-55/00-56

Accidents Still Happen

Clinical investigation: normal tissues
Clinical effects in a cohort of cancer patients overexposed during external beam pelvic radiotherapy

Cari Borrás, D.Sc.[‡], Juan Pablo Barés, M.D.[†], Damian Rudder, M.Sc.^{*†}, Ali Amer, M.Sc.[‡],
Fernando Millán, M.D.[†], Oscar Abuchaibe, M.D.[†]

^{*} Pan American Health Organization, Washington, DC, USA
[†] Instituto Oncológico Nacional, Panama City, Panama

[‡] North West Medical Physics, Christie Hospital NHS Trust, Manchester, United Kingdom



Automobile

"The complaints received via our dealers center around when drivers are on a bumpy road or frozen surface," said Paul Nolasco, a Toyota Motor Corp. spokesman in Japan.
"The driver steps on the brake, and they do not get as full of a braking feel as expected." - February 04, 2010



Automobile

'There's no brakes... hold on and pray': Last words of man before he and his family died in Toyota Lexus crash

By MAIL FOREIGN SERVICE
UPDATED: 15:07 EST, 3 February 2010

<http://www.dailymail.co.uk/news/article-1248177/Toyota-recall-Last-words-father-family-died-Lexus-crash.html>



Automobile

Japanese carmaker Honda has recalled more than 25 lakh cars across the world to rectify a software glitch

62,369 vehicles in 2007: the antilock brake system (ABS) control module software caused the rear brakes to lock up during certain braking conditions. This error resulted in a loss of vehicle control causing a crash without warning.

5,902 vehicles in 2006: under low battery voltage condition the air bag control unit improperly sets a fault code and deactivates the passenger side frontal air bag. The airbag subsequently would not deploy in the event of a collision.

<http://www.techtimes.com/articles/39149/20150312/toyota-recalls-112-500-vehicles-due-to-power-steering-and-software-issues.htm>

Toyota Recalls 112,500 Vehicles Due To Power Steering And Software Issues

By [Aaron Mamiit](#), Tech Times | March 12, 10:10 AM

Nuclear

Iran's first nuclear power plant has suffered a serious cyber-intrusion from a sophisticated worm that infected workers' computers, and potentially plant systems. Virus designed to target only Siemens supervisory control and data acquisition (SCADA) systems that are configured to control and monitor specific industrial processes (Wiki) - September 27, 2010



Nuclear

On March 7, 2008 there was a complete shutdown (Scram) of the nuclear core at Unit 2 of the Hatch nuclear power plant near Baxley after a Southern company engineer installed a software update. The software reset caused the system to detect a zero in coolant level of the radioactive nuclear fuel rods starting this unscheduled scram. **Loss \$ 5 million.**



Space

The Accident Investigation Board concluded the root cause of the Titan IV B-32 mission mishap was due to the failure of the software development, testing, and quality/mission assurance process used to detect and correct a human error in the manual entry of a constant for a filter. The entire mission failed because of this, and the cost was about **\$1.23 billion**.



Aircraft

A preliminary investigation found that the crash was caused primarily by the aircraft's automated reaction which was triggered by a faulty radio altimeter, which had failed twice in the previous 25 hours. This caused the autothrottle to decrease the engine power to idle during approach. - 25 February 2009



9 Fatalities, 117 Injured

Railway

The June 2009 Washington Metro train collision was a subway train-on-train collision. A preliminary investigation found that, signals had not been reliably reporting when that stretch of track was occupied by a train.

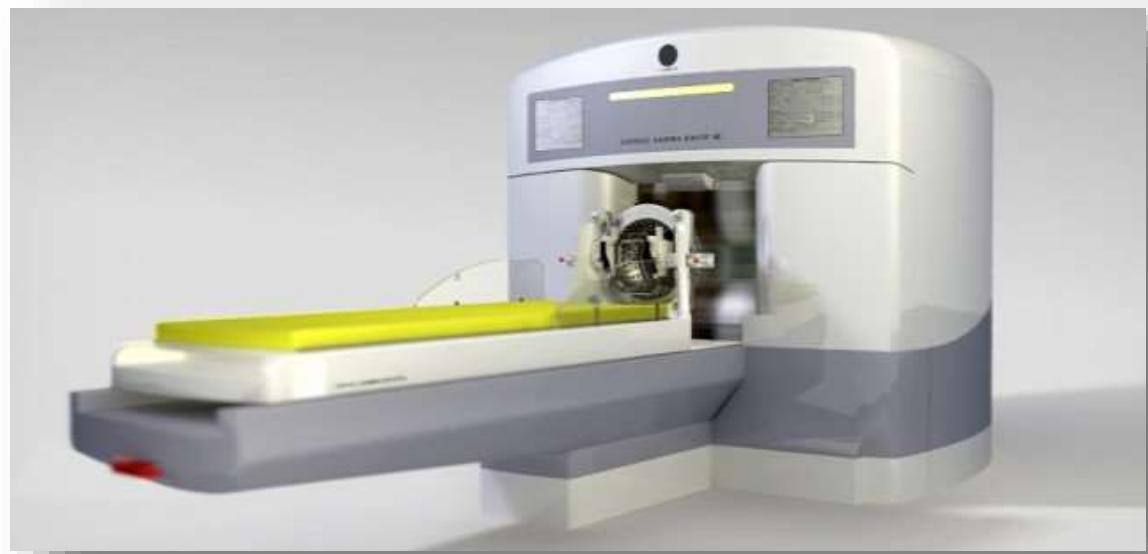


9 Fatalities, 52 Injured

Medical

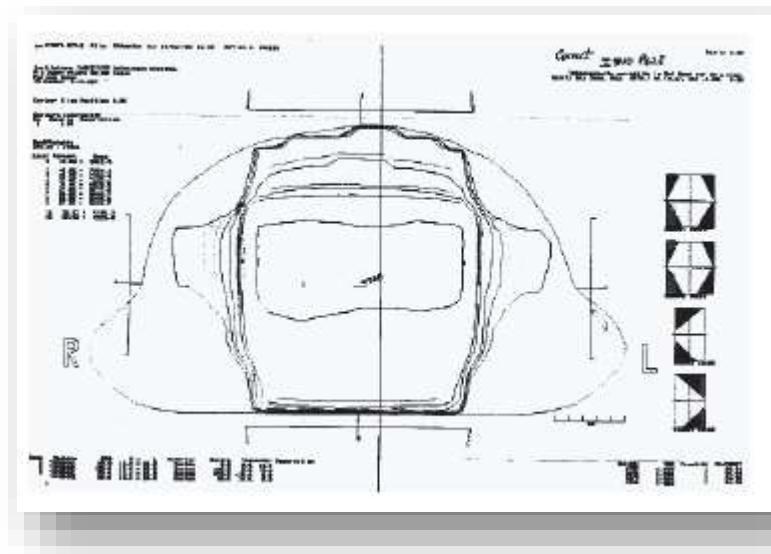
The maker of a life-saving radiation therapy device has patched a software bug that could cause the system's emergency stop button to fail to stop, following an incident at a Cleveland hospital in which medical staff had to physically pull a patient from the maw of the machine.

The bug affected the Gamma Knife, that focuses radiation on a patient's brain tumor while leaving surrounding tissue untouched. - October 16, 2009



Medical

28 radiation therapy patients were over exposed to radiation at the National Oncology Institute (Instituto Oncológico Nacional, ION) in late 2000 and early 2001. 23 of 28 at risk patients died of this due to rectal complications.



A software used to compute the dosage could not detect the erroneous inputs and gave 105% more dosage values

Medical Again

Doctors 'Shocked' by Radiation Overexposure at Cedars-Sinai

abc WORLD NEWS
WITH DIANE SAWYER

By RADHA CHITALE
ABC News Medical Unit
Oct. 13, 2009

[Share](#) 26

38

0

Doctors have expressed outrage and concern for the unsuspecting patients who received eight times the normal dose of radiation during a specific type of brain scan at Cedars-Sinai Medical Center in Los Angeles.

"There was a misunderstanding about an embedded default setting applied by the machine," according a written statement issued by the hospital. "As a result, the use of this protocol resulted in a higher than expected amount of radiation." Eight times higher, to be precise (2009)



Medical recalls (2014)

Spacelabs Healthcare Ltd., ARKON Anesthesia Delivery System with Version 2.0 Software - Software Defect May Cause System to Stop Working

Recall Class: Class I

Date Recall Initiated: March 10, 2014

Product: ARKON Anesthesia Delivery System with Version 2.0 Software

Reason for Recall: Spacelabs Healthcare is recalling the ARKON Anesthesia System with Version 2.0 Software due to a software defect. This software issue may cause the System to stop working and require manual ventilation of patients. In addition, if a cell phone or other USB device is plugged into one of the four USB ports for charging, this may also cause the System to stop working.

This defect may cause serious adverse health consequences, including hypoxemia and death. Spacelabs

<http://www.fda.gov/medicaldevices/safety/ucm393536.htm>

2015 Errors - Boeing

US aviation authority: Boeing 787 bug could cause 'loss of control'

More trouble for Dreamliner as Federal Aviation Administration warns glitch in control unit causes generators to shut down if left powered on for 248 days



The Boeing 787 has four generator-control units that, if powered on at the same, could fail simultaneously and cause a complete electrical shutdown. Photograph: Elaine Thompson/AP

SUMMARY: We are adopting a new airworthiness directive (AD) for all The Boeing Company Model 787 airplanes. This AD requires a repetitive maintenance task for electrical power deactivation on Model 787 airplanes. This AD was prompted by the determination that a Model 787 airplane that has been powered continuously for 248 days can lose all alternating current (AC) electrical power due to the generator control units (GCUs) simultaneously going into failsafe mode. This condition is caused by a software counter internal to the GCUs that will overflow after 248 days of continuous power. We are issuing this AD to prevent loss of all AC electrical power, which could result in loss of control of the airplane.

Aircraft Certification Service.

[FR Doc. 2015-10066 Filed: 4/30/2015 08:45 am; Publication Date: 5/1/2015]

2015 Errors - Airbus

Glitch found in engine software requires immediate checks after issue-plagued fleet is grounded



The Airbus A400M has been plagued by technical faults and now software glitches that reportedly caused a crash. Photograph: Julio Munoz/EPA

Airbus has issued a critical alert calling for immediate checks on all its A400M aircraft after a report identified a software bug as having caused a fatal crash in Spain earlier this month.

In a statement Airbus said it was "devastated to confirm" the loss of four crew members, adding that another two are in hospital in a serious condition.



The plane crashed in a rural area near Seville airport. (Pic: Airline.net)

Four crew dead!!

2016 – F35 Bugs

Radar glitch requires F-35 fighter jet pilots to turn it off and on again

Troubled warplane that has yet to see any cyber security testing hit with yet another bug affecting flight performance requiring software update



<https://www.theguardian.com/technology/2016/mar/08/radar-glitch-requires-f-35-fighter-jet-pilots-to-turn-it-off-and-on-again>

<http://www.defenseone.com/technology/2016/02/f-35s-terrifying-bug-list/125638/>

The screenshot shows a news article from Defense One. At the top, there's a navigation bar with categories: NEWS, THREATS, POLITICS, MANAGEMENT, and TECH. Below the navigation bar is a large image of an F-35 fighter jet parked on a runway with mountains in the background. To the right of the image are social media sharing icons for Facebook, Twitter, Google+, LinkedIn, and Email. Below the image, the headline reads "The F-35's Terrifying Bug List". Underneath the headline, the author is listed as "FEBRUARY 2, 2016 BY PATRICK TUCKER". The main text of the article begins with: "The Pentagon's top testing official has weighed and measured the F-35 and found it wanting." The text continues to discuss the jet's shortcomings, mentioning its inability to tell old parts from new ones and its random prevention of user logins into logistics information systems.

Tips

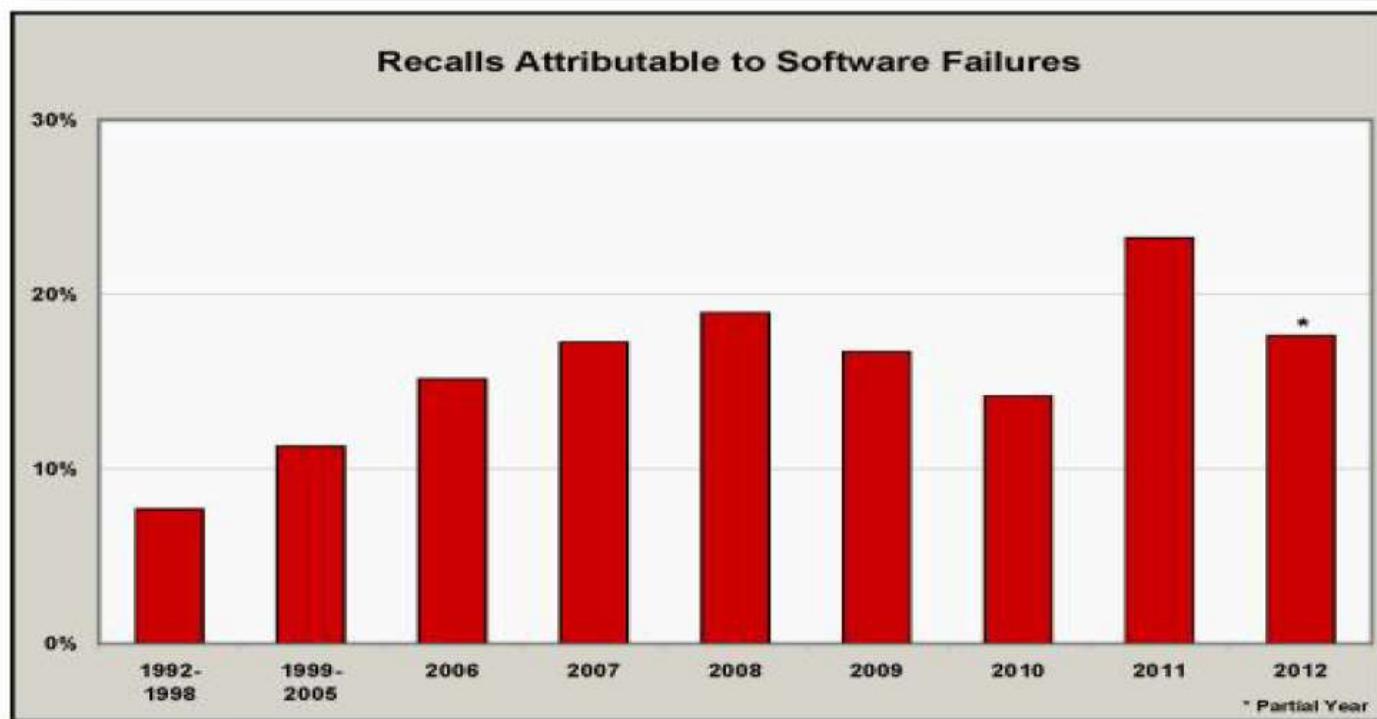
The accidents reported are very recent. These show that we have still not mastered the engineering of software. We require all our wits and ingenuity to make a safety critical system work. Out thinking Nature is an overwhelming task.

Tips

As a hindsight all these accidents could have been avoided with extensive testing. There are no shortcuts in a safety critical system development process.

FDA – Software recalls data

Office of Science and Engineering Laboratories Annual Reports



Additional information <http://www.fda.gov/MedicalDevices/Safety>ListofRecalls/default.htm>

Where does the industry stand?

Fault densities of 0.1 per KLoC are exceptional and seen in space shuttle software

UK DoD study indicates 1.4 safety critical fault per KLoC in a DO178B certified software

18 million flight per annum and a loss due to software being 1.4 per million flights amounts to 0.3×10^{-6} per hour.

Nuclear industry claims 1.14×10^{-6} per hour

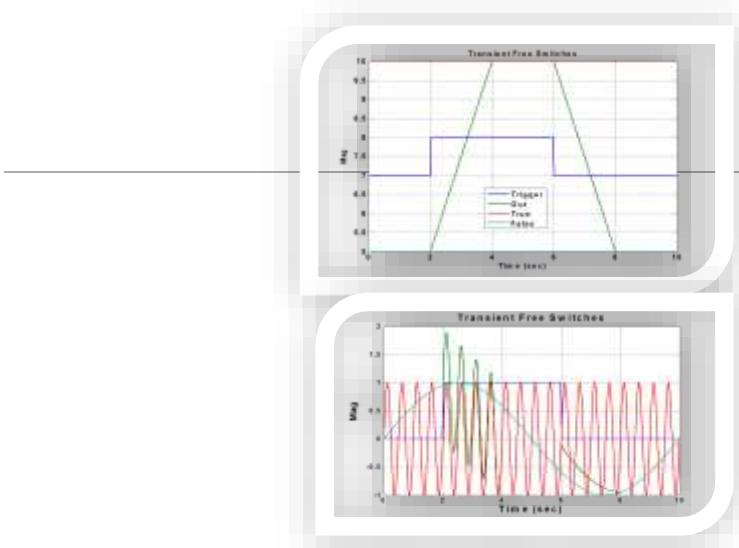
These are approximations

Bottom line we are about

10^{-6}
failures per hour

<http://www-users.cs.york.ac.uk/tpk/nucfuture.pdf>

Mistakes We Made



FOCUS: SAFETY-CRITICAL SOFTWARE

Flight Control Software: Mistakes Made and Lessons Learned

Focus
Safety-Critical Software

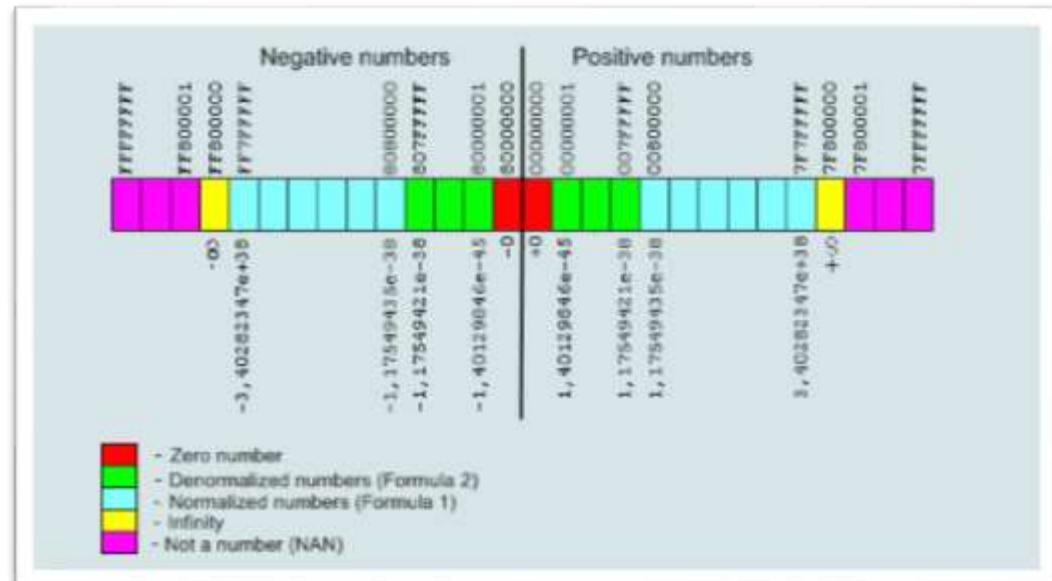
A Dormant Error

Recently an error came up in an aerospace program

This error was existing in a flight control system for the last **12 years**

A very specific sequence of operations carried out by the pilot triggered this error causing a channel failure in flight

- The cause – A very small number equal to 10^{-37}



A Dormant Error

The requirement was for an integrator output to fade to zero in a specific time duration say 3 seconds when a system reset was given

The algorithm computed this by finding the current output and dividing it into the small delta decrement in one sample

Each iteration this small delta is subtracted from the output and the reset mode ends when the output changes sign

In this case the compiler optimizer made this small error to zero and the integrator remained in a reset mode

A Dormant Error

Why did this not happen in the simulation? Why did this not happen in other channels?

This is due to fact that the optimizing compiler for the processor sets floating point values less than $1.1754945E-38$
($00000000100000000000000000000001$) to 0.0. **

In the simulation it was possible to go to $1.4E-45$
($00000000000000000000000000000001$)

This is due to denormal numbers
http://en.wikipedia.org/wiki/Denormal_numbers

** <http://www.h-schmidt.net/FloatConverter/IEEE754.html>

A Dormant Error

In one channel the floating point number was around 10^{-38} and this became 0.0 but in the other channels it was 10^{-37} so it was not set to 0.0.

Why did we fail to find it during testing? This is mainly a project decision to do these tests on the system platform where the correct operation was proved time and again. But it is difficult to test numeric values so low as 10^{-38} at such a platform.

Now that we know the cause recreating this is easy but time consuming. It does not happen always. **It did not happen for 12 years in actual flight tests!**

MOD Issue

Your example of the Host computer and Target compiler treating tiny numbers differently illustrates the importance of target based testing.

Our surprise was the Floating point MOD operator in a standard published mathematical library.

We discovered that the control algorithm could (under some circumstances) attempt a floating point MOD operation when one of the operands was TINY and the other one was HUGE. The algorithm would loop and subtract the divisor from the numerator until it turned negative. This would take up to four seconds and suspend the program for that time. Clearly we had to change the algorithm to prevent this ‘lockout’. It did decrease performance for the general case, but the time determinism was more important to us.



Contributed by
George Romanski

Tips

An error can lie dormant like a volcano. This will surface only when a right trigger exercises this. In Therac-25 the error was dormant till the technicians because experts and started inputting faster leading to a race condition!

Integrator with Limits

Integrators with limits are used very commonly in PID control laws. These are used extensively in safety critical fly by wire system

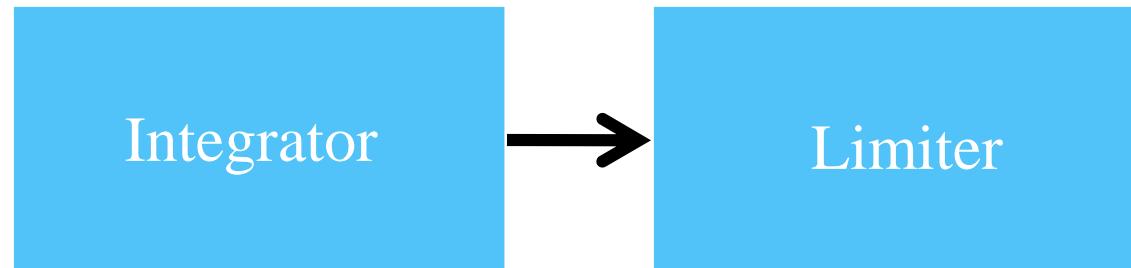
The integrators are called anti-windup integrators

They have a property that the output shall be saturated at a specific value on the positive and negative outputs

They have a very subtle requirement that the outputs shall come out of saturation immediately on the input reversing the sign. This is the anti-windup action.

Integrator with Limits

Is this a correct implementation?



Integrator with Limits

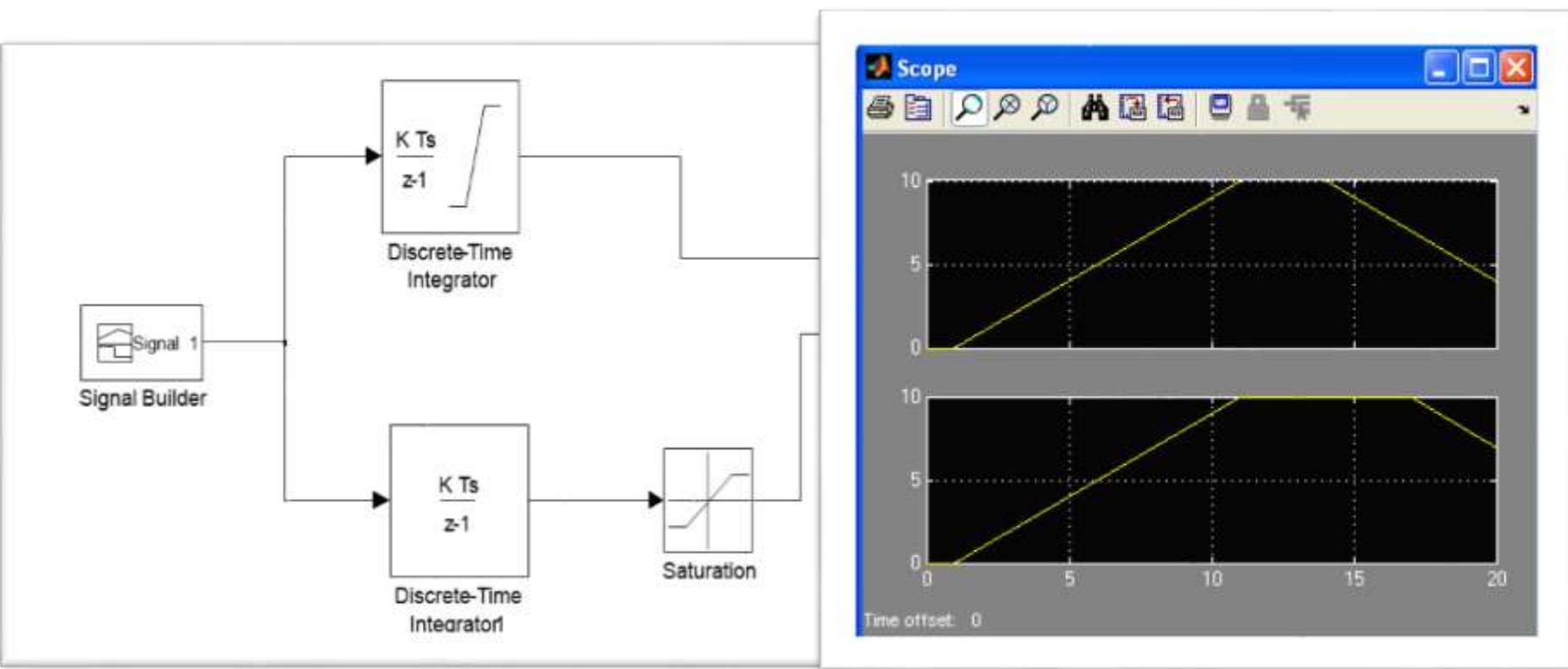
A correct implementation is that the state (output) of the integrator is limited and used in the next frame of computation on a continuous basis every computational cycle.

I have found instances of the incorrect implementation in many of the control system implementation **again and again**.

Integrator

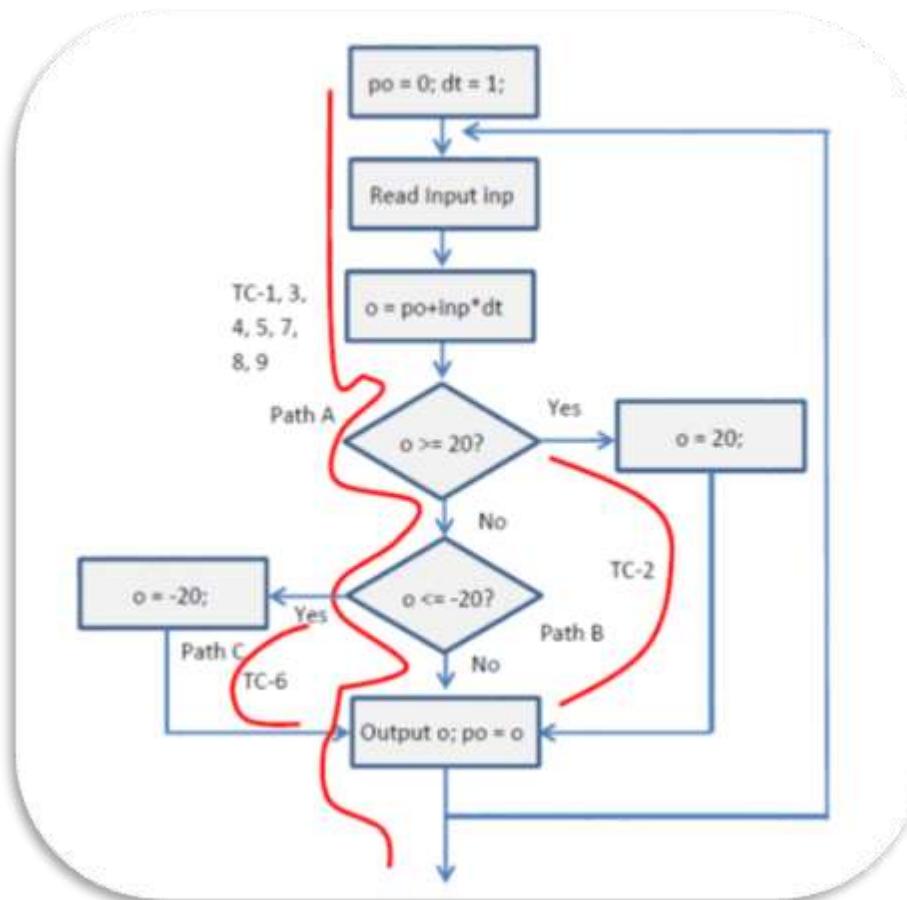
Limit the
States
(output)

Integrator Differences



The output comes out of saturation immediately in Case 1, the correct implementation, (above) and takes time in the Case 2 (below)

Code Coverage



The point of concern is that both implementations provide the same 100% code coverage for test cases that do not bring out the error. We can easily say that I have done testing looking at coverage metrics but still have this error resident in the code.

Tips

A requirement based testing is very important. We have to understand the block and generate test cases such that the requirement that the integrator comes out of saturation immediately is tested.
(More on this on test design later)

Tips

We require “dynamic”, “control theoretic”, requirement coverage metric for control system elements. This has been done for Boeing control system program and is reported in literature.

Cu, C.; Jeppu, Y.; Hariram, S.; Murthy, N.N.; Apte, P.R., "A new input-output based model coverage paradigm for control blocks," Aerospace Conference, 2011 IEEE , vol., no., pp.1,12, 5-12 March 2011, doi: 10.1109/AERO.2011.5747530

Filter with Limits

A first order digital filter was to be implemented and its output signal limited to a specific value in a missile autopilot application

Is this implementation correct?

Filter

Limit the
States (output)



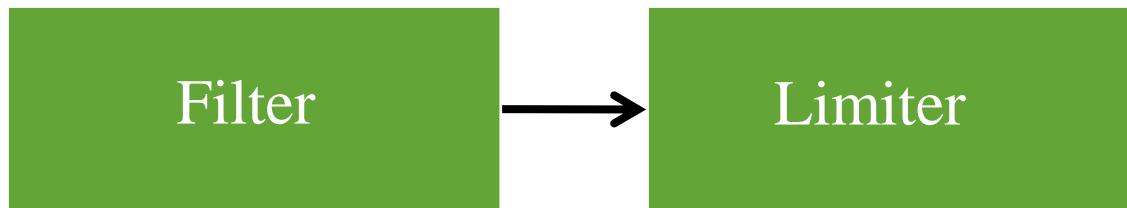
Filter with Limits

In this case the correct implementation is as shown below.

This limiter was wrongly implemented and led to a limit cycling oscillation which destroyed the missile.

This was proved and shown during the post flight analysis.

The next missile had a similar error somewhere else! We love making the same mistakes in life!!



Erratic Fader Logic

Fader Logic or Transient Free Switches are used in aircraft control systems extensively

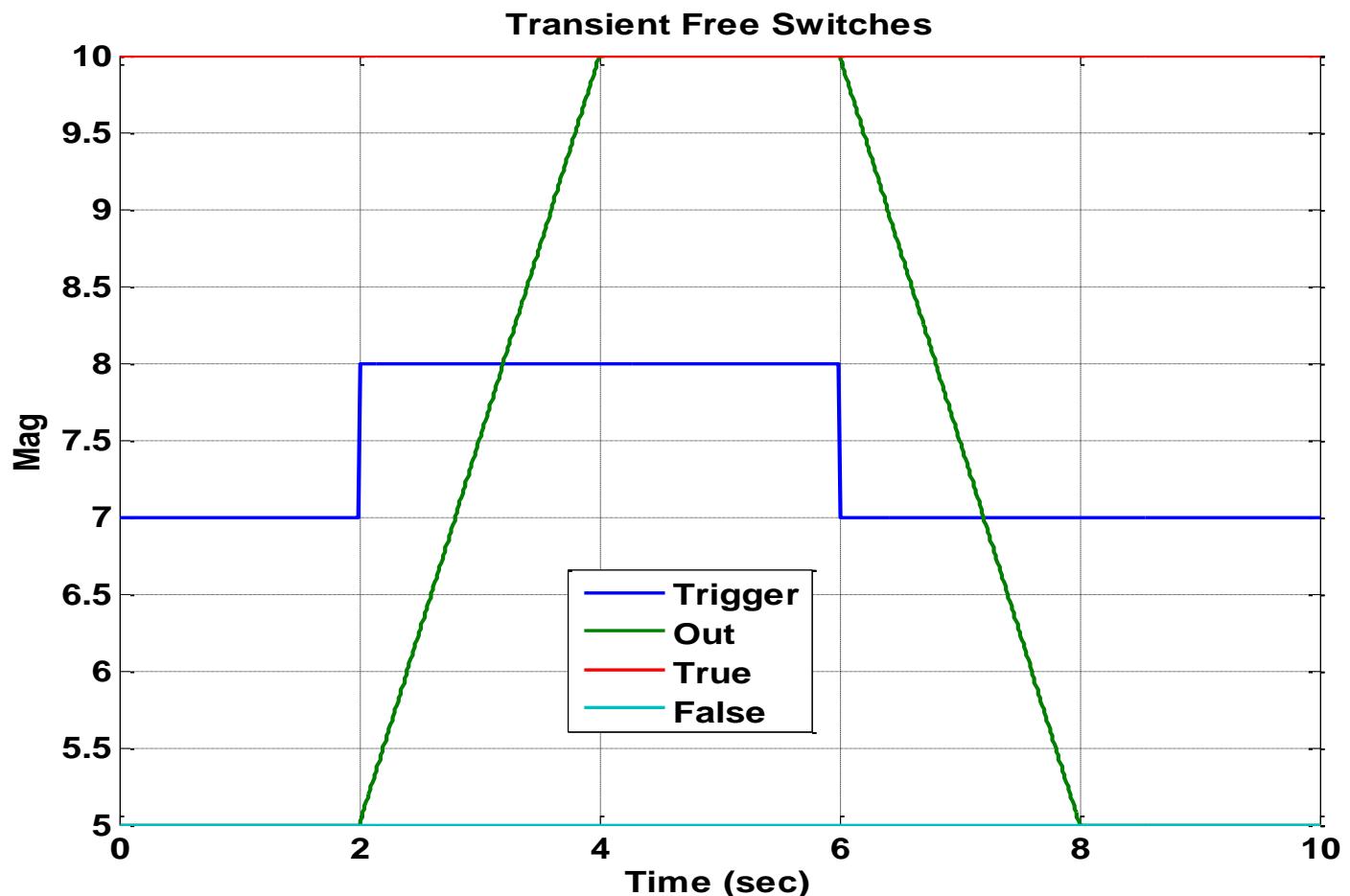
In an Indian program a linear fader logic was implemented to fade from one signal to the other linearly in a specified time (say 2 seconds).

During stress testing it was found that the logic implementation worked very well for two constant signals.

The behavior was very different for a time varying signal. There were instances where the output signal of the fader logic was greater than either of the inputs and in some cases had a negative value even though the inputs were positive.

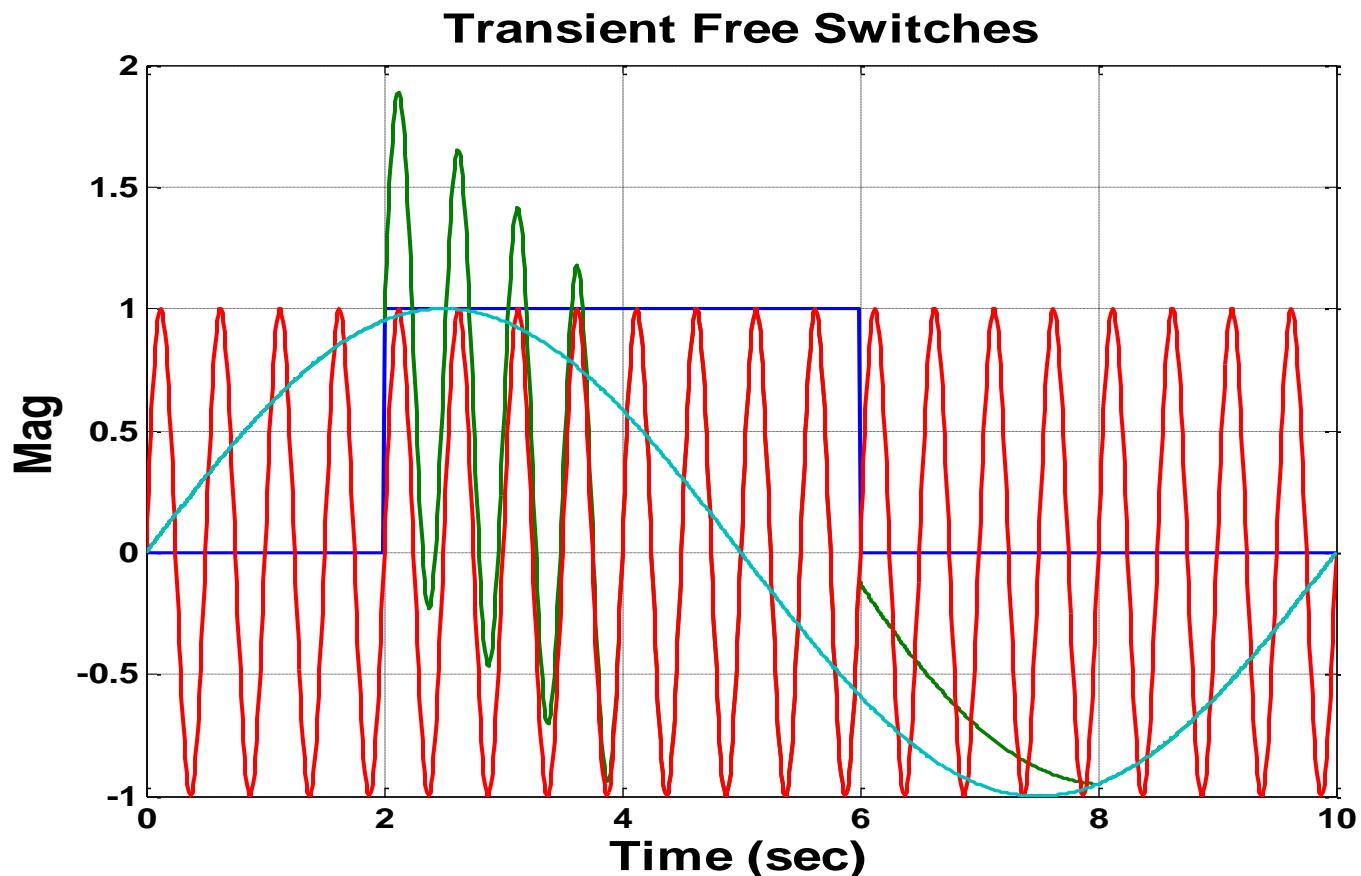
Erratic Fader Logic

The normal behavior of the fader logic. Output fades from 5 to 10 and back from 10 to 5 in 2 seconds based on trigger.



Erratic Fader Logic

The output signal (green) is greater (nearly double) than the inputs (amplitude 1.0)



Erratic Fader Logic

This behavior was ignored by the design team stating that the testing was very vigorous and in flight this could not happen.

A test flight was aborted with a failure in a secondary control system. This was attributed to the erratic fader logic.

In another flight test the pilot had to forcibly bring the aircraft nose down due to this behavior.

The fader logic was rearranged to rectify the problem.

After 15 years I find the same logic in another aircraft control law. This behavior was rectified by changing the logic. *We repeat the same mistakes in life!*

I recently (2016) happened to see the algorithm for this block in the BEACON tools. Its still the same. Waiting for you to use it!!

Tips

This is also brings out an interesting fact. Inputs to control systems are normally limited. But it is very likely that the signals modified by the control system elements may have values very much larger than the limited inputs. When we test individual elements we have to ensure that we have tested it to these larger values.

Tips

One way of finding the bounds of internal variables is to run many random cases and monitor the internal variables and collect the maximum and minimum values. I have done this for the Indian LCA program and the Boeing programs. Simulink monitors these limits automatically. Be aware that in big control systems these can run to thousands of variables! Weekends and nights are the best time to run these tests.

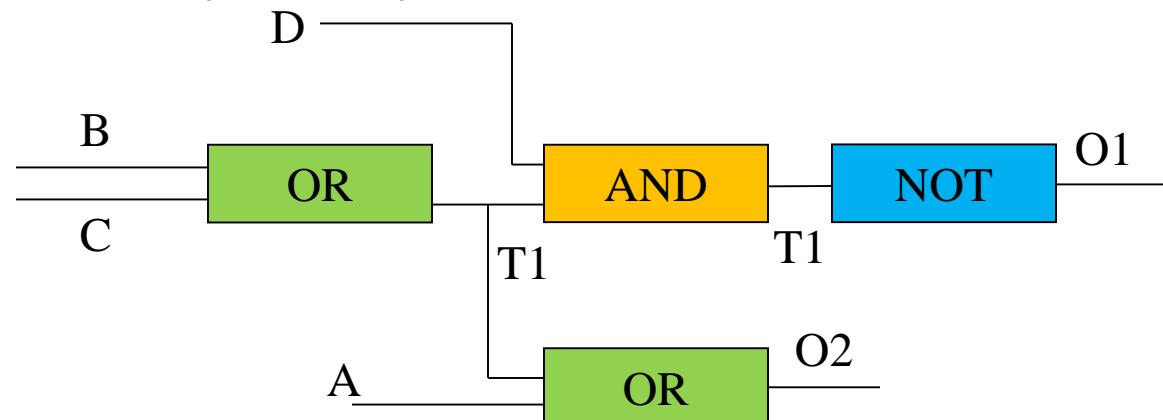
Variable reuse

Handwritten code from models have created problems

It is a good practice in coding to use the same variable again if possible. This saves memory space.

We have seen that errors occur very often by this reuse of variables.

```
T1 = C .OR. B  
T1 = D .AND. T1  
O1 = NOT(T1)  
O2 = T1 .OR. A
```



Is this correct?

Note: Depending on optimization settings, the internal Variable T1 may disappear - George Romanski

Tips

We have a tendency to make the same mistakes again and again. Often designers do not take the Verification team's observation stating the testing is too rigorous and such conditions do not occur in real life.

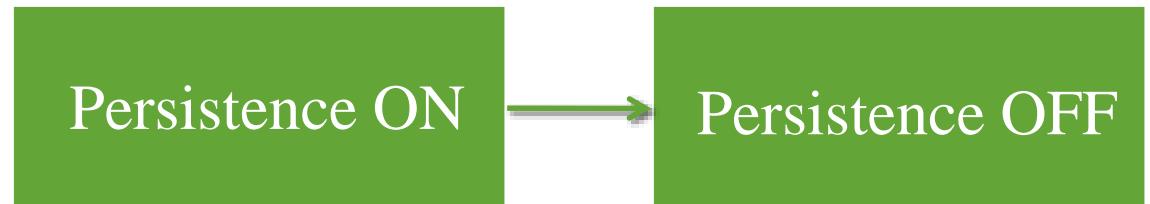
... THIS IS NOT TRUE!

Persistence Blocks Anomaly

Persistence blocks are used in control systems to vote out faulty signals. They are also known as delay On/Off/On-Off blocks.

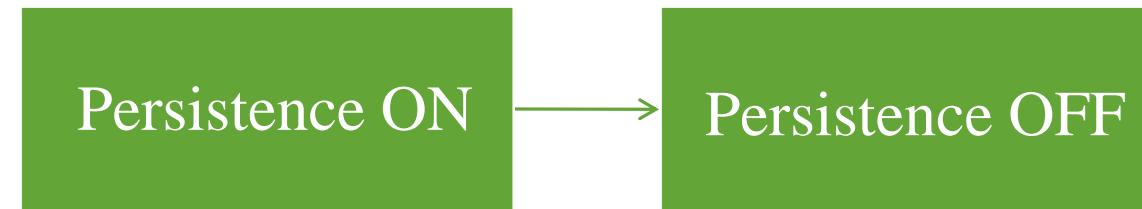
A persistence on block looks for an input signal to be True for a specified amount of time before setting the output True. A persistence off blocks does the same looking for a False input signal. A persistence on/off block looks for either a True or False signal for a specified On (True) or Off (False) time before setting the output to True or False.

Is this ON/OFF?



Persistence Blocks Anomaly

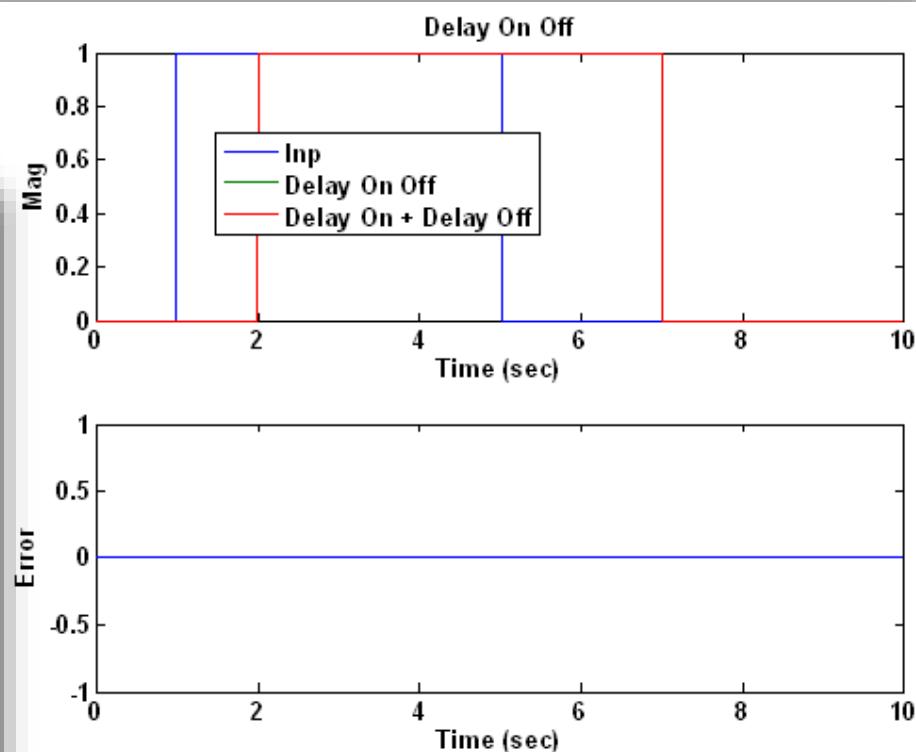
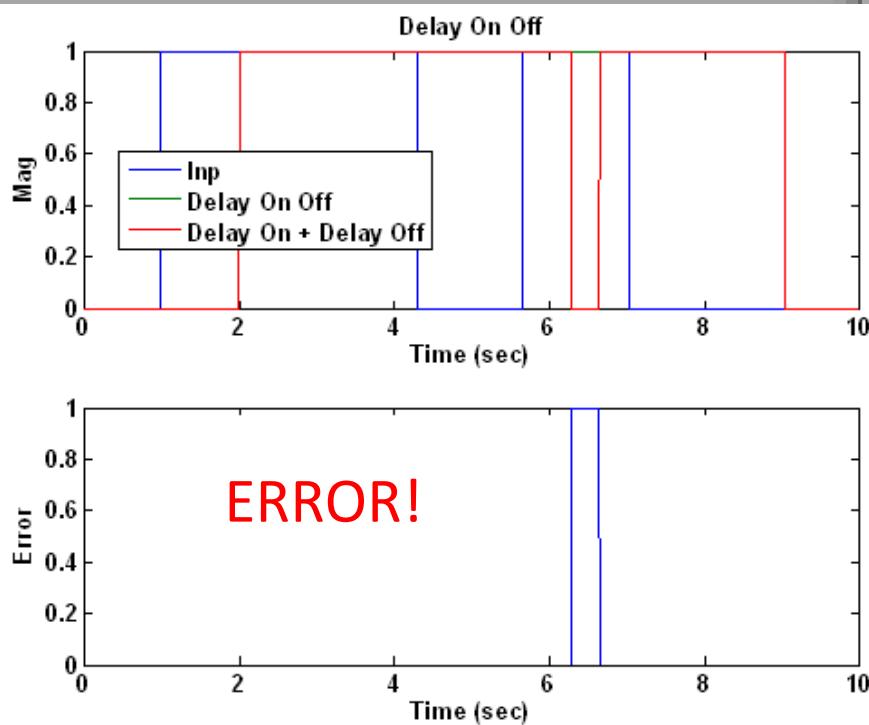
Extensive testing in a Fly-by-Wire system brought out the fact that Persistence Off function called after a Persistence On function in C Code



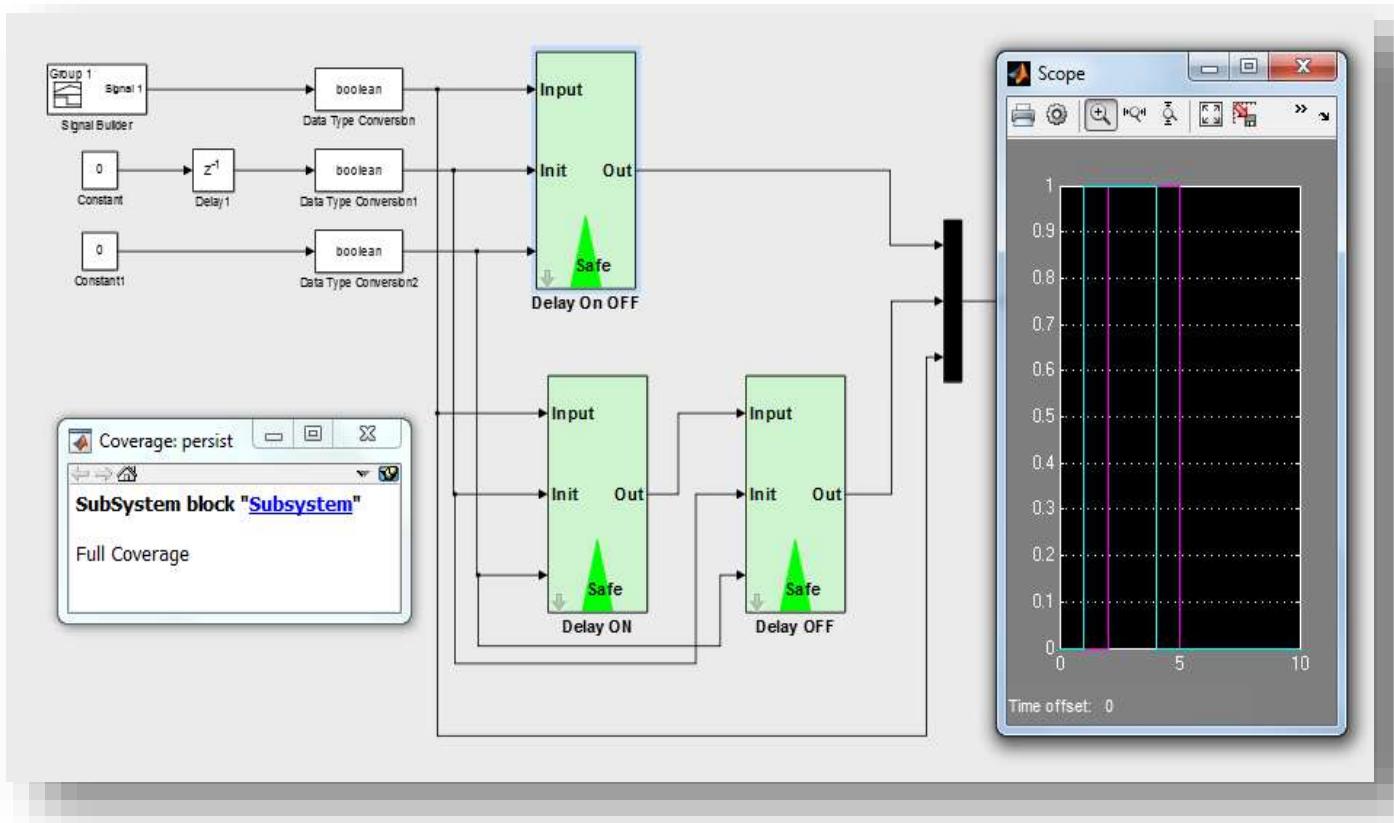
IS ^{Not}



Persistence Blocks Anomaly

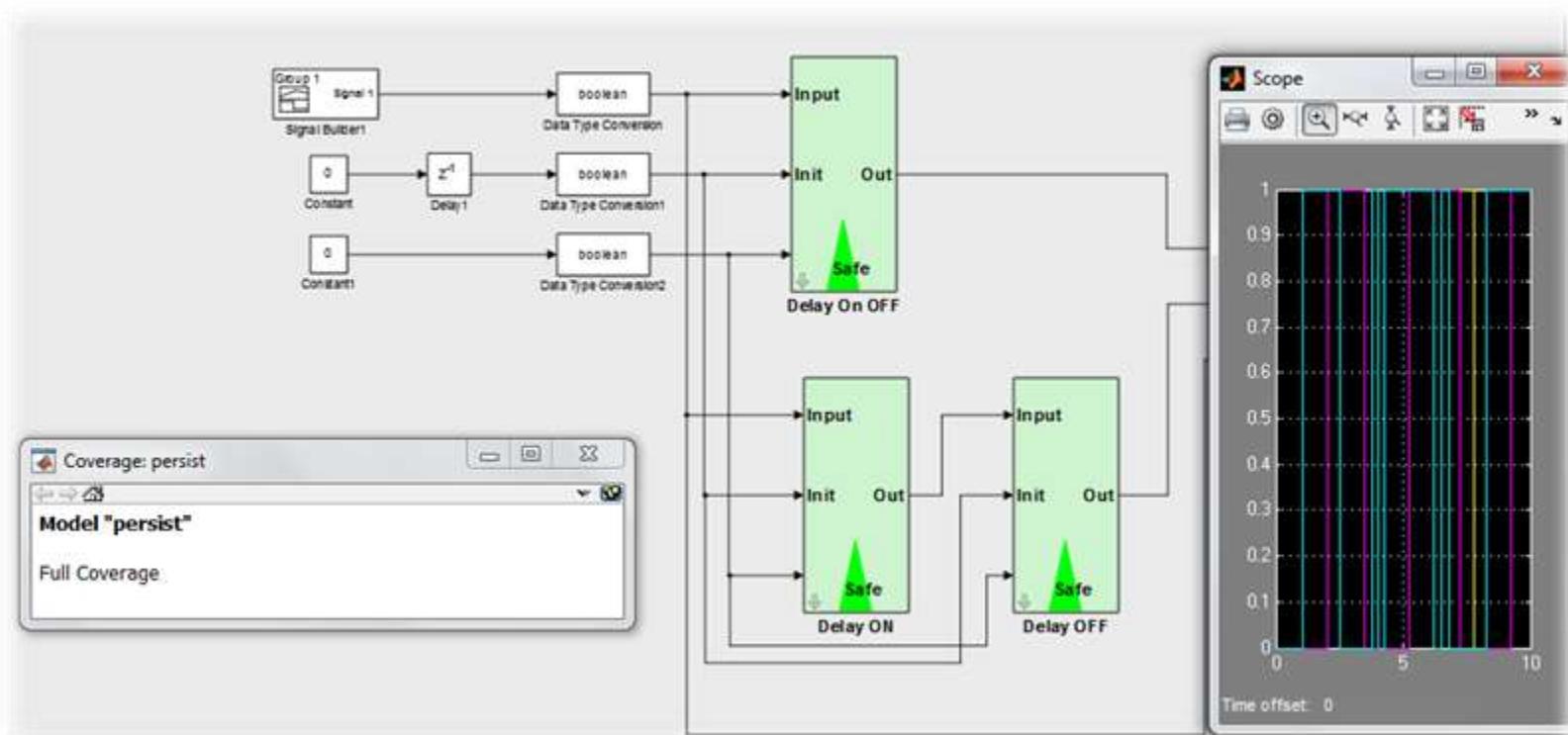


Block coverage



100% block coverage but the error is not found.

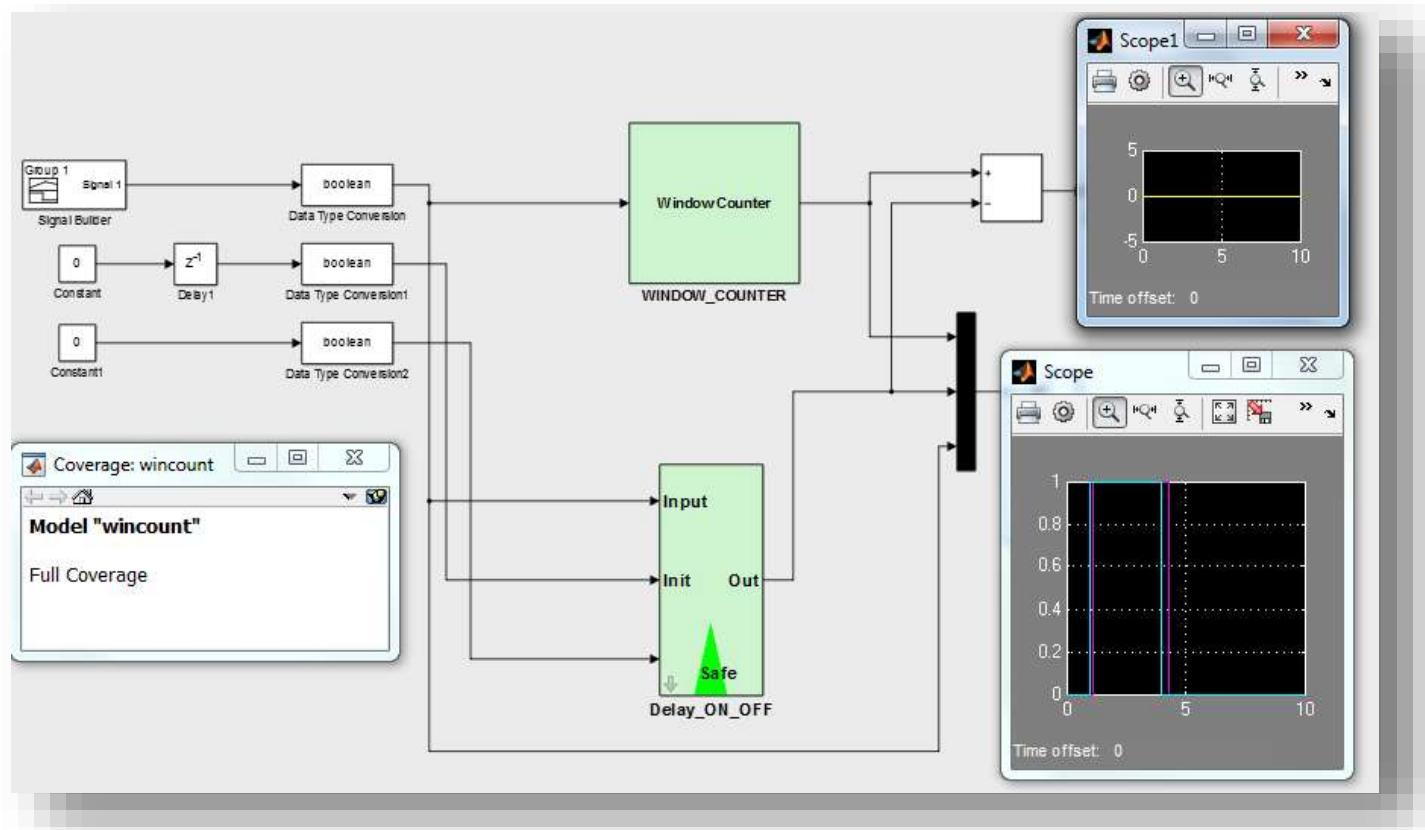
Error Detected



A proper test case design has brought out the error (yellow).

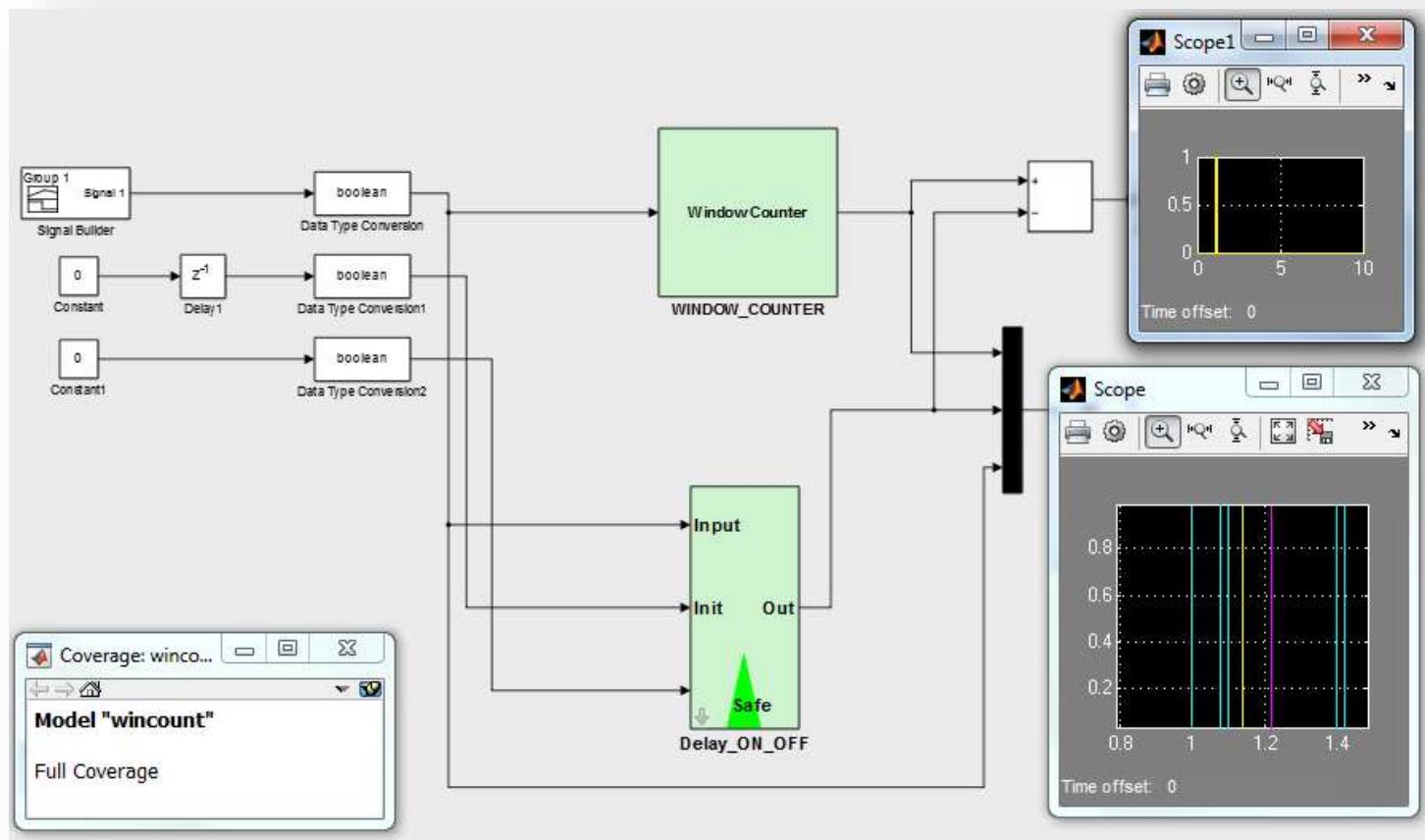
Window Counter Vs Persistence On/Off

100% Model Coverage but Error = 0



Window Counter Vs Persistence On

A 100% Coverage need not necessarily find system error. A Delay On/Off could easily replace a window counter. Proper test case design is very important!



Tips

It is possible to substitute a delay on off block for a window counter and no one will know the difference if the test cases are not good. In control systems there are many such blocks which have similar behavior in specific conditions. Be aware of these.

Tips

While looking at code coverage and block coverage metrics please keep the salt shaker handy. DO NOT take these two metrics for “100% system tested – job done” metric. They will misguide you.

Tips

NEVER, Never generate test cases just to achieve code and block coverage. These two metrics aid in finding locations where you have not tested. If they indicate deficiencies then go ahead add more high and low level **“Requirements Based Tests”**

Filter Coefficient Inaccuracies

Filter coefficients have to be coded with sufficient accuracies and as asked by the designer.

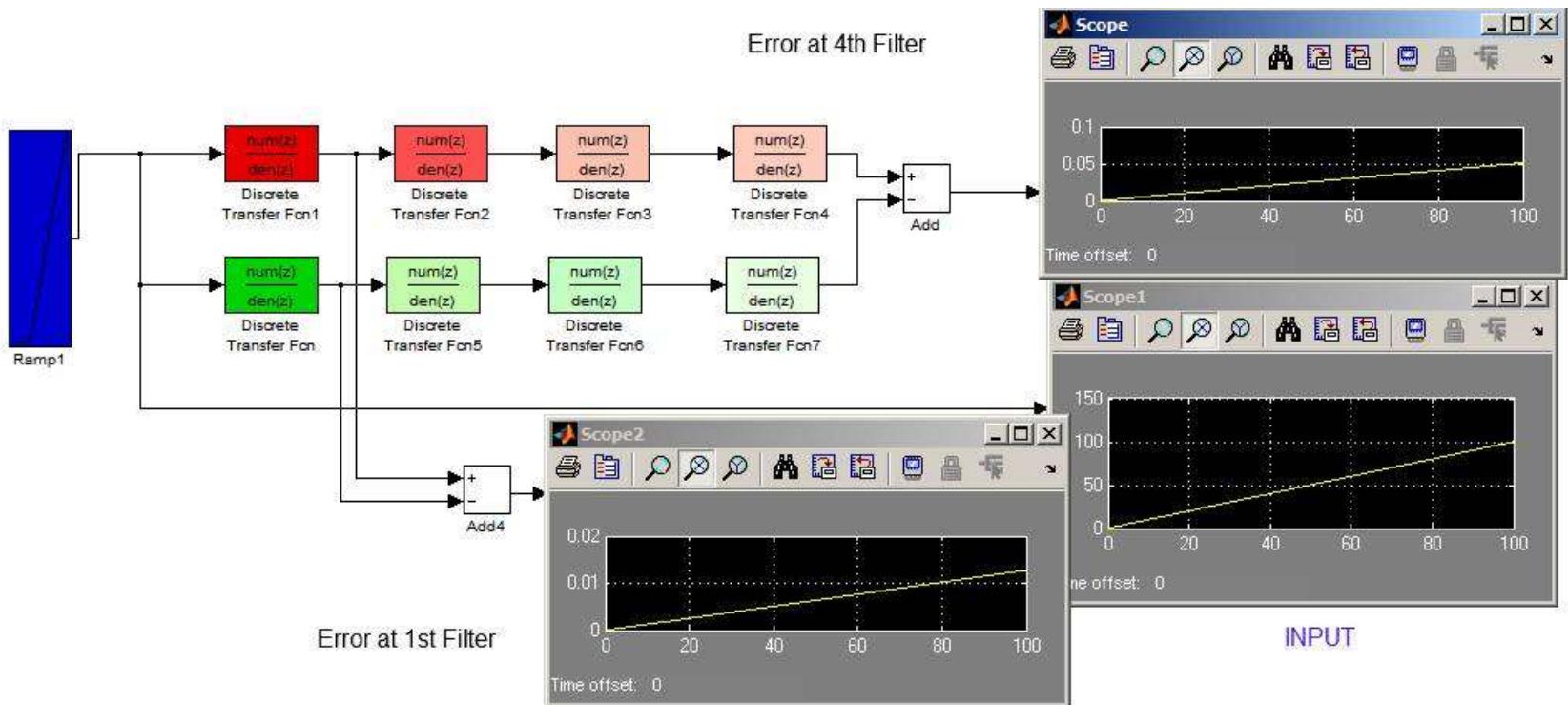
Filter coefficient errors have led to the loss of spacecraft to a tune of billion dollars.

- (*sunnyday.mit.edu/accidents/titan_1999_rpt.doc*)

In a recent test activity in one of our projects a fourth decimal place error in filter coefficient was caught during testing.

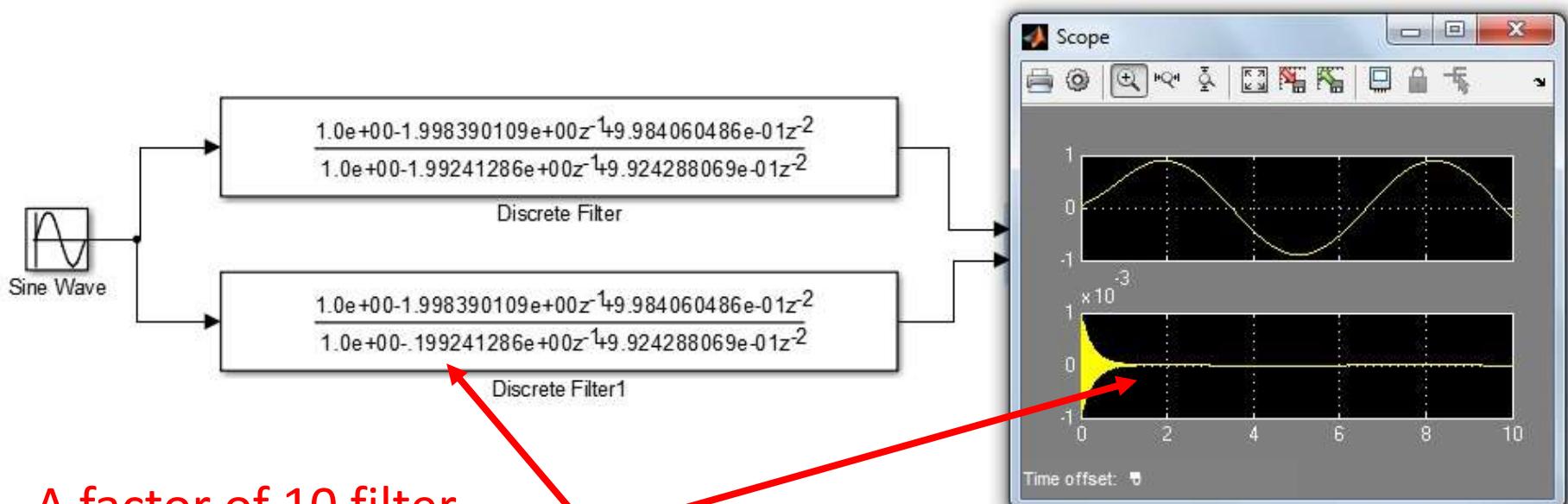
Most often engineers may quip stating this is a small error. But if this error was systemic and the coding team had rounded off all filter coefficients to the 4th decimal place! It could lead to large error terminally.

Filter Coefficient Inaccuracies



A fourth decimal place error could cumulatively pile up after 100 seconds of run.

Titan IV B-32 Filter Problem



A factor of 10 filter coefficient error made the output to zero

Tips

Nothing is small or insignificant in a safety critical system. Patriot missiles, airport communications have failed after continuous runs for long duration. Small drifts can accumulate over time changing direction of travel, dosages to patients ... We have had fun seeing the LCA flight actuators slowly drift from a steady state condition while we were talking!

Filter Error (2014)

We recently found a new error in a washout filter implementation

A washout filter $63 \text{ s } /(\text{s} + 63)$ was implemented as a Simulink block for verification and a manual C code was done to implement the functionality for the on board controller.

```
OUT=[];  
oi=0;ow=0;pow=ow;pi=0;  
for i = 1:length(inp)  
    oi = pow*0.01+oi; % Integrator  
    ow = (inp(i)-oi)*63;  
    pow=ow;  
    OUT=[OUT;ow];  
end
```

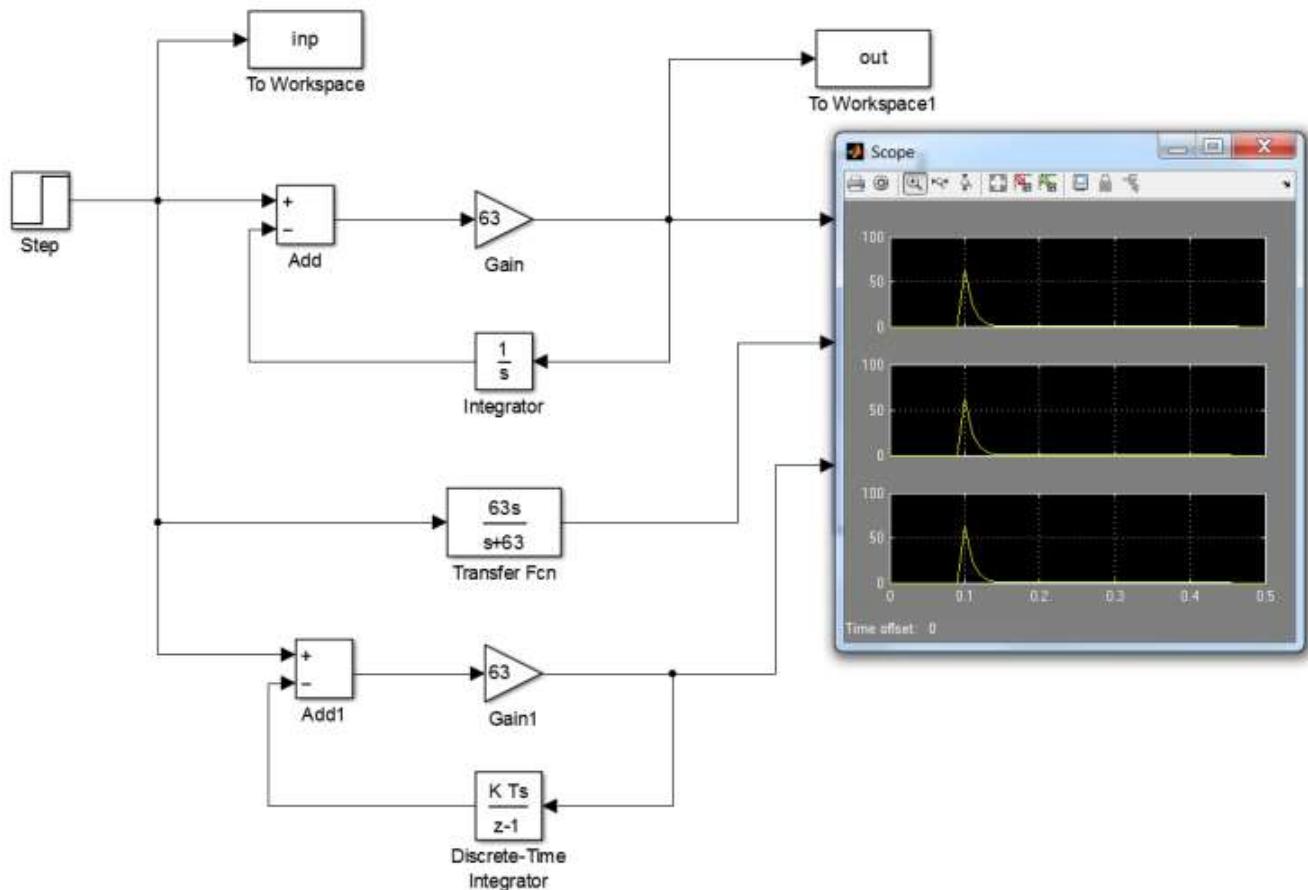


```
OUT=[];  
oi=0;ow=0;pow=ow;pi=0;  
for i = 1:length(inp)  
    ow = (inp(i)-oi)*63;  
    oi = pow*0.01+oi; % Integrator  
    pow=ow; % called later  
    OUT=[OUT;ow];  
end
```



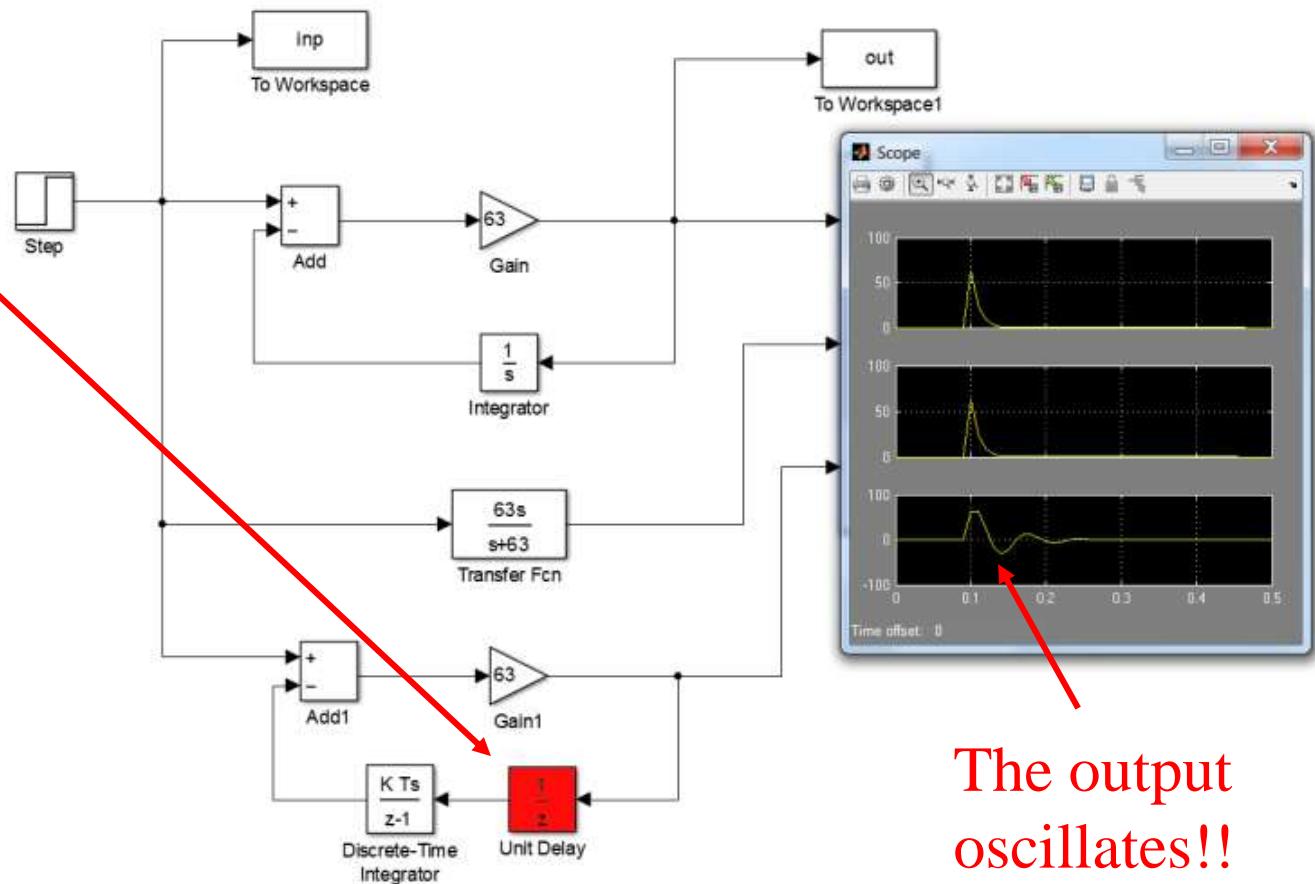
Filter Error (2014)

The washout filter behaves as expected for a step response as seen by the analog and digital implementations



Filter Error (2014)

The code implementation added a one sample delay in the filter due to the way the integrator was called. The integrator used forward Euler method which had another sample delay



The output oscillates!!

Tips

Often enough we may neglect a very simple thing like a calling sequence which can introduce a one sample delay. This is more so in code where a global variable will have some initial value and therefore it will not throw out an error in its usage. ***But the dynamic behavior could be very different!! An oscillation could be catastrophic!!***

Tips

Calling sequence errors can be observed at the Ironbird level also when we do a frequency response analysis. We had found an issue in the LCA slat control calling sequence by looking at the anomaly in a frequency response. The smallest error during testing is worthwhile exploring further.

Up/Down Counter

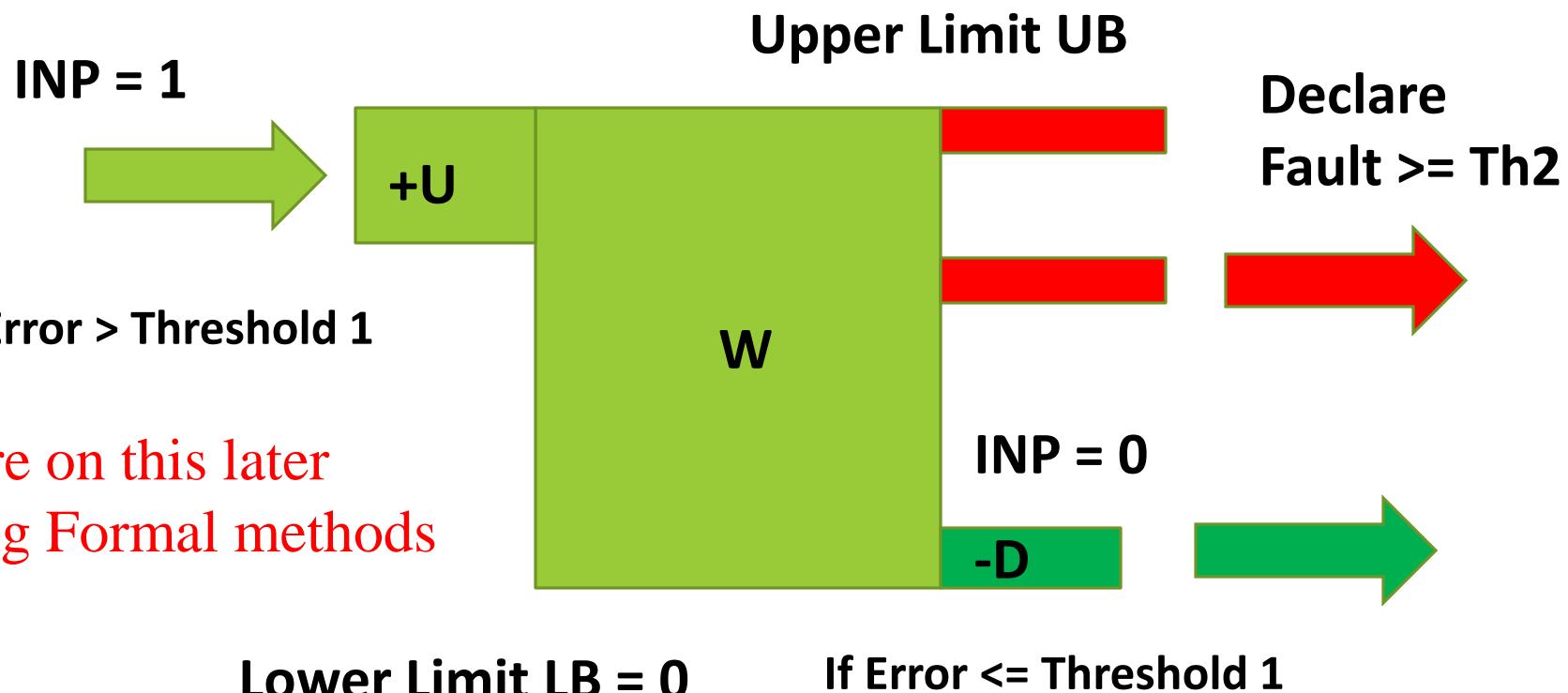
The aim is to increase the fault detection probability while keeping the false alarm rate constant.

An important benefit of up-down counters is that it can reject large, single event upsets. One such example is aerospace applications where radiation effects can corrupt memory and cause a wildly spurious sensor data.

If Inp is TRUE then increase the count W by up count U, If Inp is FALSE then decrease the count by the down count D. if the count is greater than threshold Th2 set output of the Up/Down counter to True. The max count value and the min count value is limited.

In the C code implementation the count was defined as **unsigned integer**. A specific value of down count D caused the value to become negative and there was a **wrap around triggering a fault in a healthy situation**.

Up/Down Counter



If $U > (UB - LB)$ and $-D = -U$ then the up-down counter will declare a fault if the residual exceeds Threshold 1 for one time step.

Tips

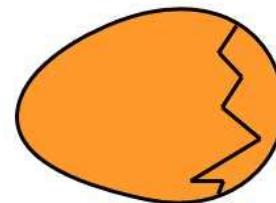
The data type of variables is very important in control system implementation. The control designers normally use double precision numbers in their design. The final implementation that happens for the on board computer will have specific data types. **Take care here.**

Endianness

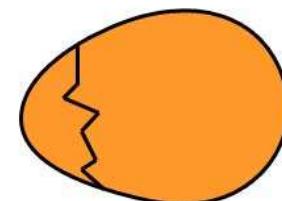
“Endianness is important as a low-level attribute of a particular data format. Failure to account for varying endianness across architectures when writing software code for mixed platforms and when exchanging certain types of data might lead to failures and bugs, though these issues have been understood and properly handled for many decades”.*

- We still find errors due to this in our test activity.

<http://flickeringtubelight.net/blog/wp-content/uploads/2004/05/eggs.jpg>



BIG ENDIAN - The way people always broke their eggs in the Lilliput land



LITTLE ENDIAN - The way the king then ordered the people to break their eggs

* <http://en.wikipedia.org/wiki/Endianness>

Control Block Initialization

All control system blocks are initialized to ensure proper behavior.

Filters are initialized to ensure that there is no transient at start if there is no change in input. The output will hold the input value in a steady state.

Integrators are initialized to ensure that there is no output change if the input is set to zero.

Rate limiters are initialized to ensure that the output is not rate limited at start and does not change its value if the input does not change.

All Persistence blocks, failure latches are initialized to ensure a safe start of system.

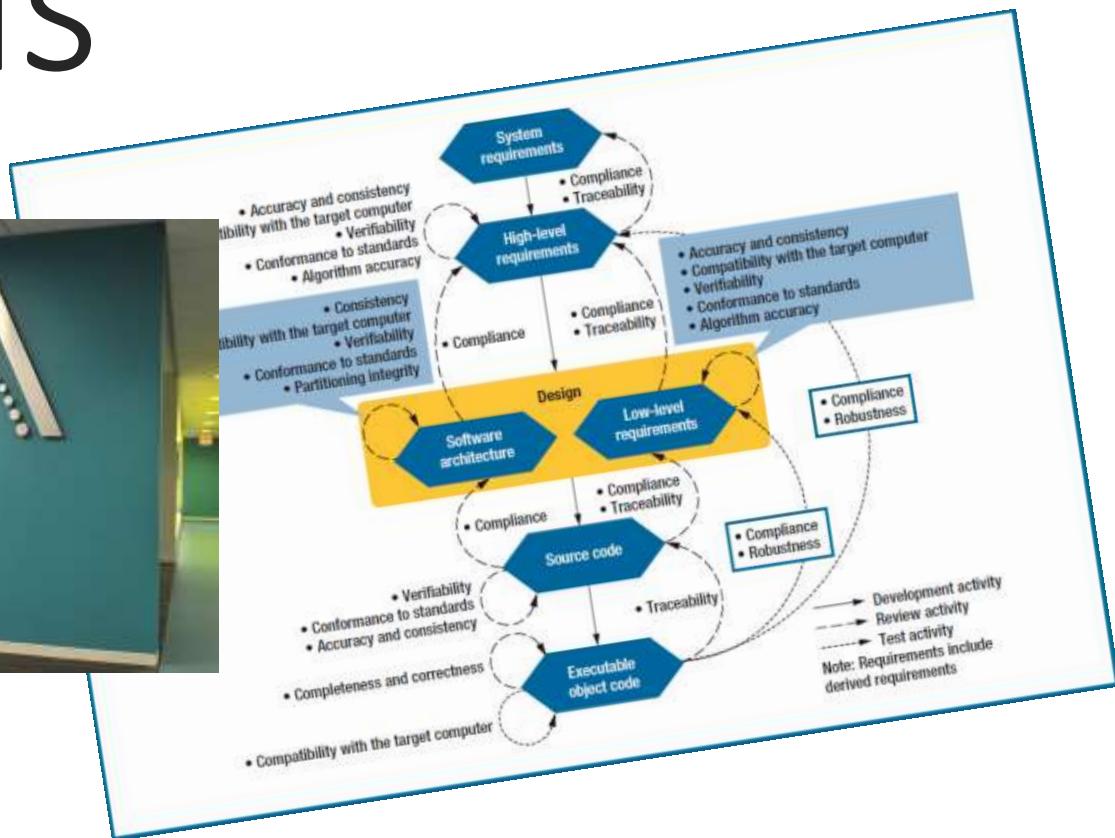
Tips

We always, always make mistakes in initialization. This is very true. Check the initialization with different constant input values to ensure proper initialization! This requires a set of test cases and a system restart.

Tips

Testing safety critical control systems requires the tester to understand the control system, the algorithm, the standards, the test methodologies like model based testing, block coverage metrics and block functional coverage metrics.

Standards



Aerospace Standard – DO-178B

Called “Software Considerations in Airborne Systems and Equipment Certification”

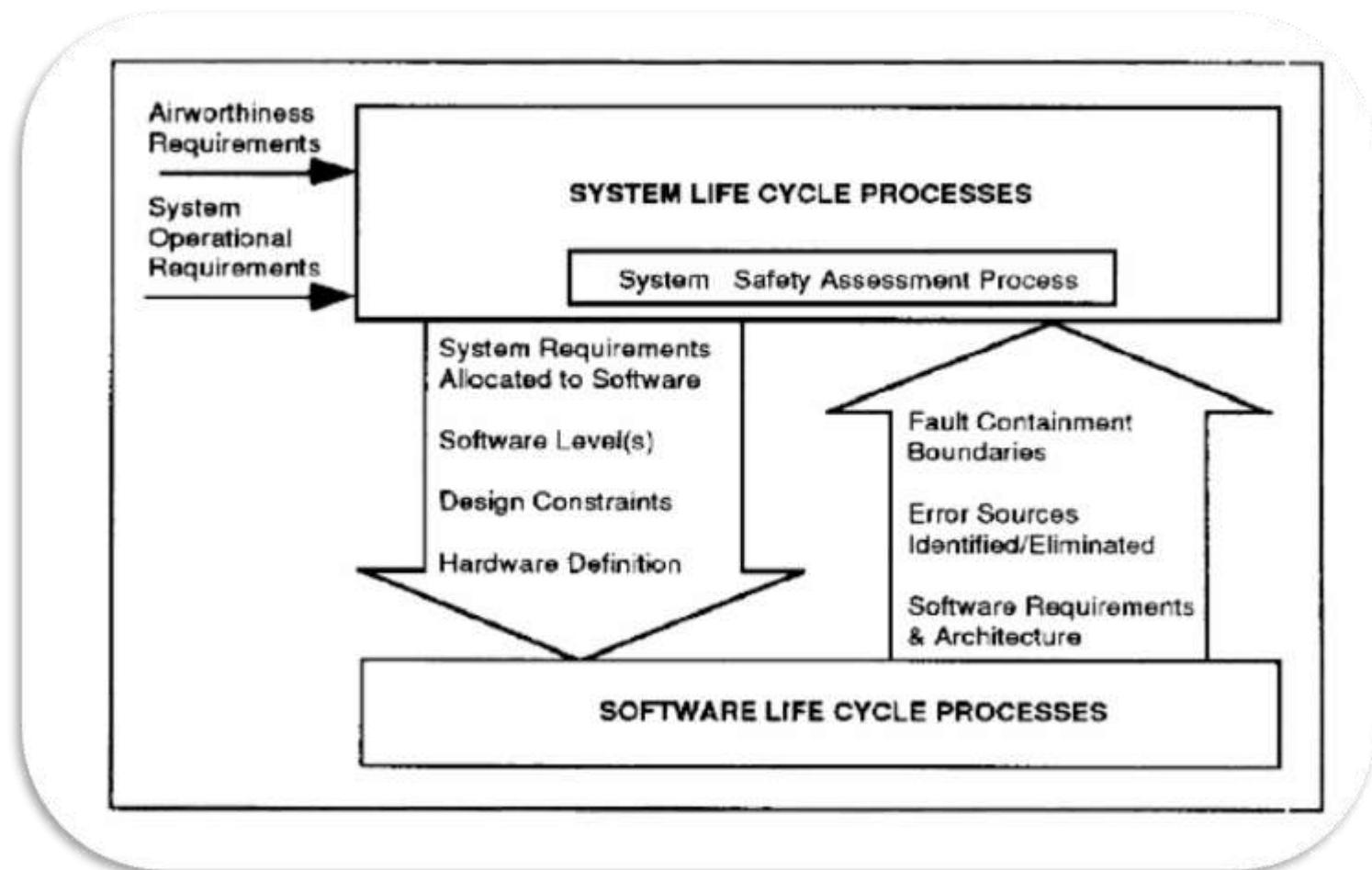
Published by RTCA Inc (This stood for Radio Technical Commission for Aeronautics)

It is a document that addresses the life cycle process of developing embedded software in aircraft systems.

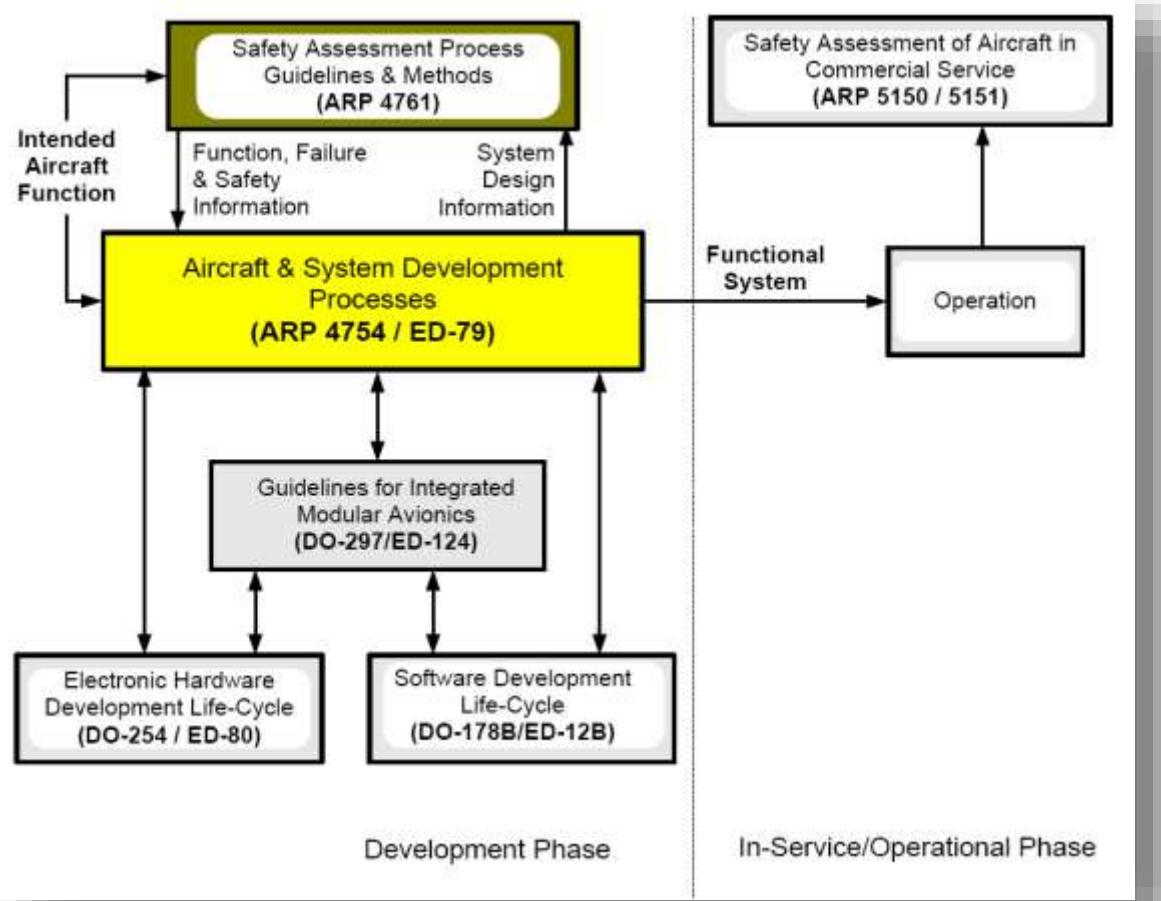
It is only a guidance document and does not specify what tools and how to comply with the objectives

It is a commonly accepted standard worldwide for regulating safety in the integration of software in aircraft systems and insisted by the certifying authorities like FAA

Information Flow System-Software



System Development Process

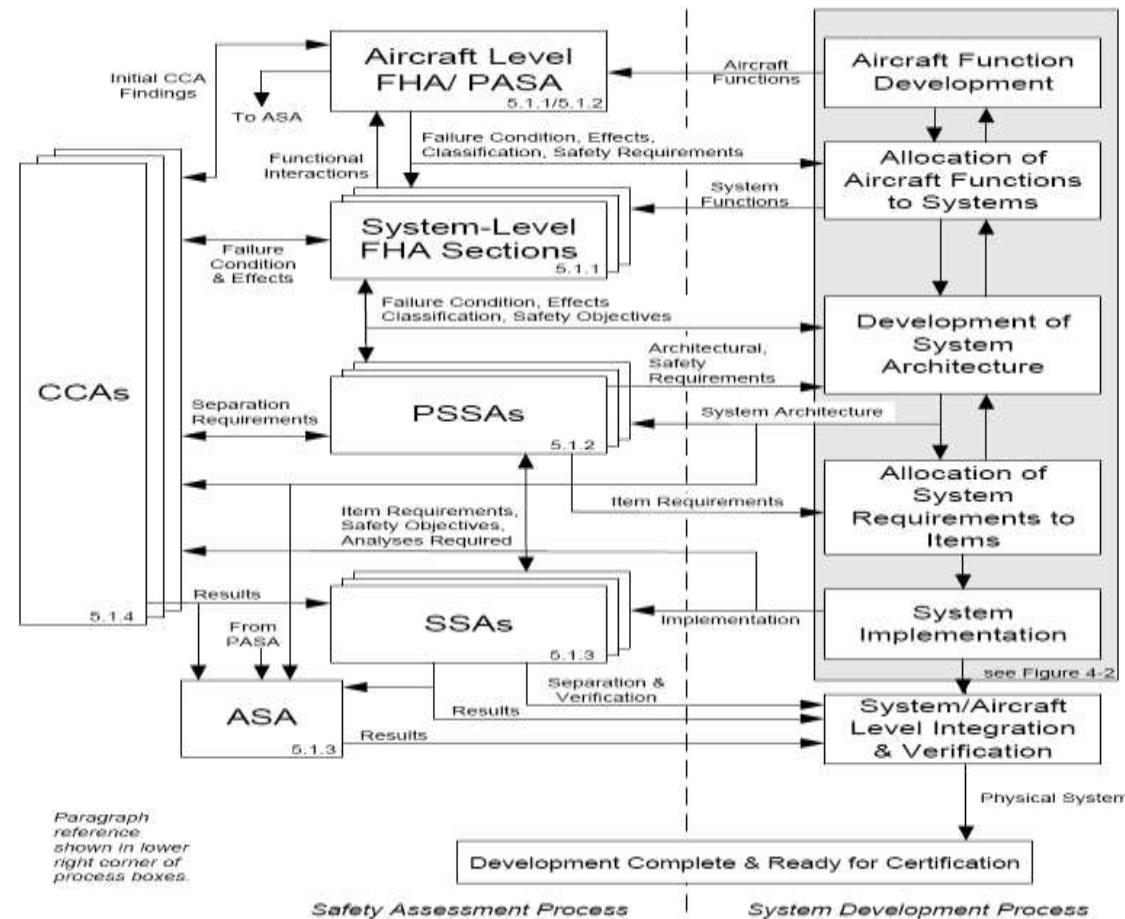


System Safety

The system safety analysis is carried out based on SAE (Society of Automotive Engineers) ARP (Aerospace Recommended Practice) 4761

- Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment
- describes techniques for safety engineering of aviation systems
- Used in conjunction with SAE ARP 4754 "Certification Considerations for Highly-Integrated or Complex Aircraft Systems"
- This refers to DO178B

Safety Assessment Process Model

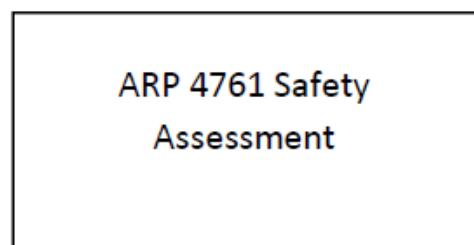


SAE ARP 4761

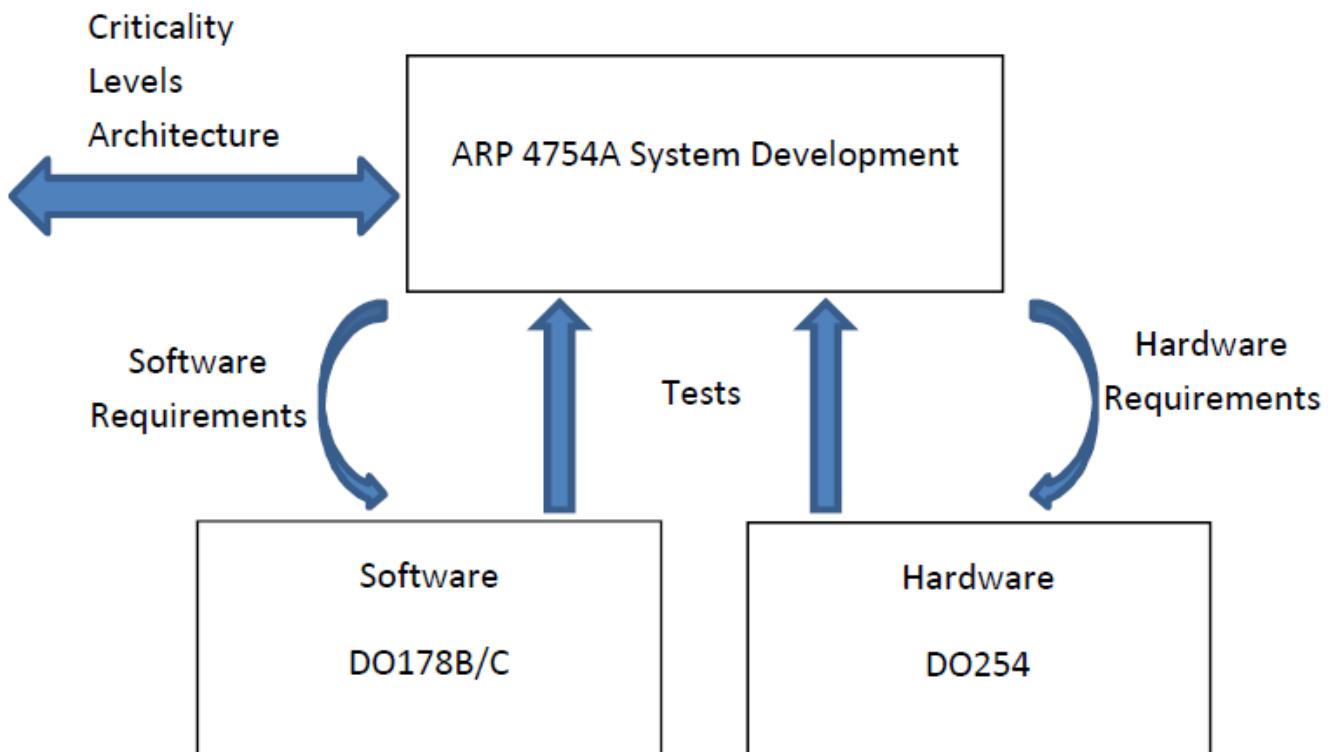
- Functional Hazard Assessment (FHA) (addresses hazard identification and preliminary risk analysis)
- Preliminary System Safety Assessment (PSSA) (analyses the contribution and interaction of the subsystems to system hazards)
- System Safety Assessment (SSA) (assess the results of design and implementation, ensuring that all safety requirements are met)
- Techniques used in one or more of the above phases include Fault Tree Analysis (FTA), Dependency Diagrams (DD), Markov Analysis (MA), Failure Modes and Effects Analysis (FMEA), Failure Modes and Effects Summary (FMES) and Common Cause Analysis (CCA) (consisting of Zonal Safety Analysis (ZSA), Particular Risks Analysis (PRA) and Common Mode Analysis (CMA)).

Relation Between the Standards

Guidelines and Methods for
Conducting the Safety
Assessment Process on Civil
Airborne Systems



AEROSPACE RECOMMENDED PRACTICE revised 2010-12
Guidelines for Development of Civil Aircraft and Systems



There is a
tight coupling
with Systems
Safety

AC 25.1309-1A - System Design and Analysis

This document defines improbable failures and extremely improbable failures

In any system or subsystem, the failure of any single element, component, or connection during any one flight should be assumed, regardless of its probability. Such single failures should not prevent continued safe flight and landing, or significantly reduce the capability of the airplane or the ability of the crew to cope with the resulting failure conditions.

- Subsequent failures during the same flight, whether detected or latent, and combinations thereof, should also be assumed, unless their joint probability with the first failure is shown to be extremely improbable.

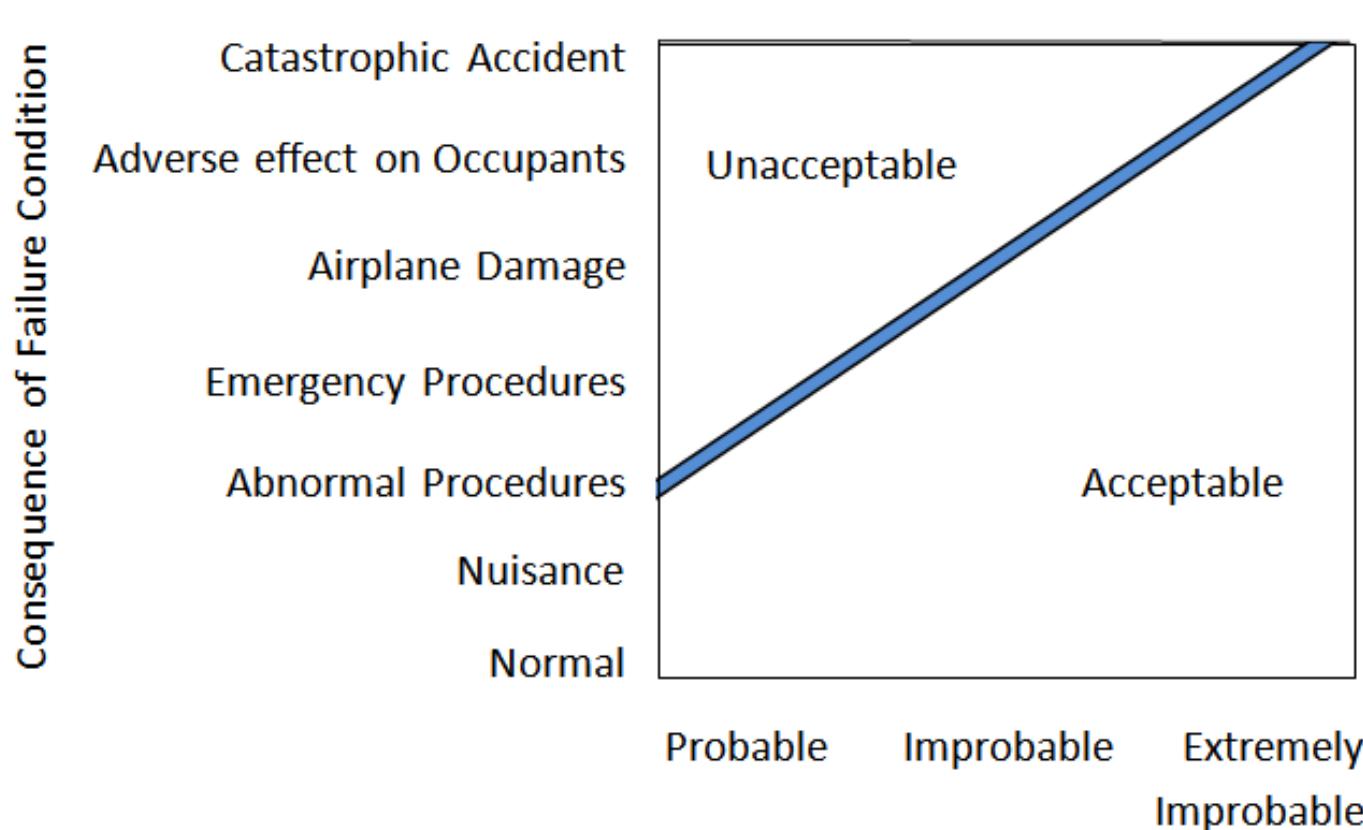
Failure severity

Effects-on the airplane, such as reductions in safety margins, degradations in performance, loss of capability to conduct certain flight operations, or potential or consequential effects on structural integrity.

Effects on the crewmembers, such as increases above, their normal workload that would affect their ability to cope with adverse operational or environmental conditions or subsequent failures.

Effects on the occupants; i.e., passengers and crewmembers.

Probability Vs Consequence



Aerospace Standard – DO-178B

Five levels of software have been defined

Software Criticality	Level	Probability FAR/JAR	Remarks
Catastrophic	A	$< 10^{-9}$	Failure may cause a crash. Error or loss of critical function required to safely fly and land aircraft.
Hazardous	B	$< 10^{-7}$	Failure has a large negative impact on safety or performance. Passenger injury.
Major	C	$< 10^{-5}$	Failure is significant, but has a lesser impact than a Hazardous failure (leads to passenger discomfort rather than injuries)
Minor	D	$< 10^{-3}$	Failure is noticeable, but has a lesser impact than a Major failure (causes passenger inconvenience)
No Effect	E	Any	Failure has no impact on safety, aircraft operation, or crew workload.

Federal Aviation Administration AC 25-1309-1A and/or the Joint Aviation Authorities AMJ 25-1309

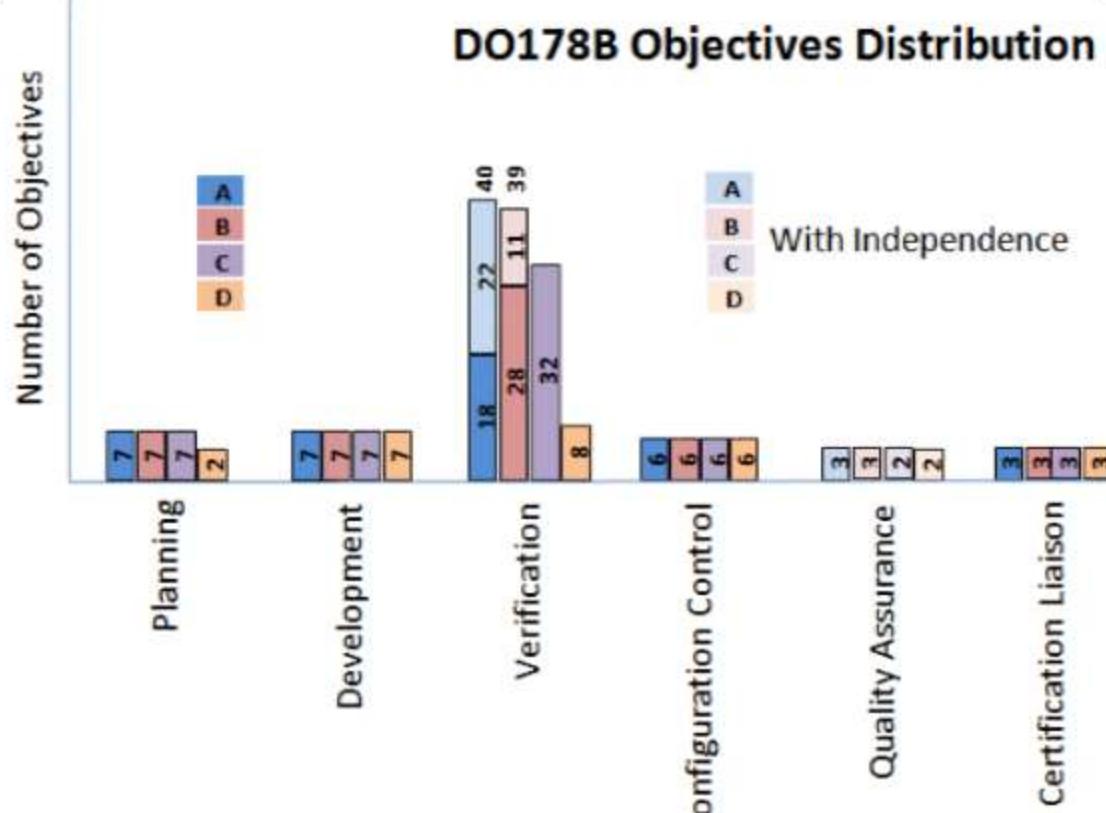
Aerospace Standard – DO-178B

Defines a list of objectives with and without independence for the various design assurance levels of software

Software Levels	Number of Objectives		
	With	Without	Total
A	25	41	66
B	14	51	65
C	2	55	57
D	2	26	28

Process	Planning	Development	Verification	Config . Control	Quality Assurance	Certification Liaison	Total
Objectives	7	7	40	6	3	3	66

DO178B Final Words



Determining the Levels

The impact of failure, both loss of function and malfunction, is addressed when making this determination

The most severe case of failure is considered to determine the level

The levels may change based on the system architecture

- If the system safety assessment process determines that the system architecture precludes anomalous behavior of the software from contributing to the most severe failure condition of a system, then the software level is determined by the most severe category of the remaining failure conditions to which the anomalous behavior of the software can contribute.

Architectural Considerations

Partitioning is a technique for providing isolation between functionally independent software components

Multiple-version dissimilar software is a system design technique that involves producing two or more components of software that provide the same function in a way that may avoid common mode failures.

Safety monitoring is a means of protecting against specific failure conditions by directly monitoring a function for failures

Redundancy

User-modifiable/Field Loadable software

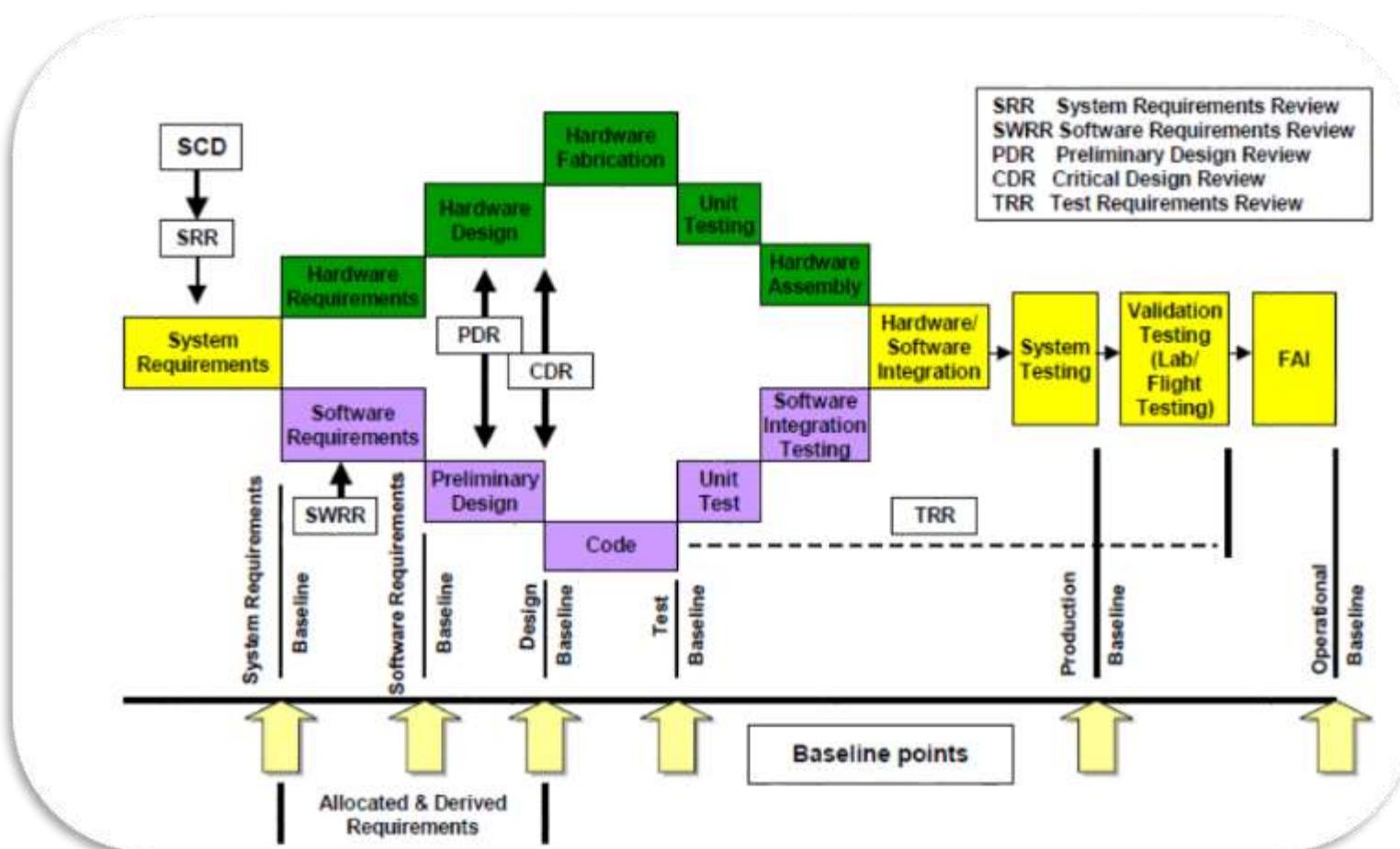
Users may modify software within the modification constraints

The software which provides the protection for user modification should be at the same software level as the function it is protecting

If the inadvertent enabling of the software data loading function could induce a system failure condition, a safety-related requirement for the software data loading function should be specified in the system requirements

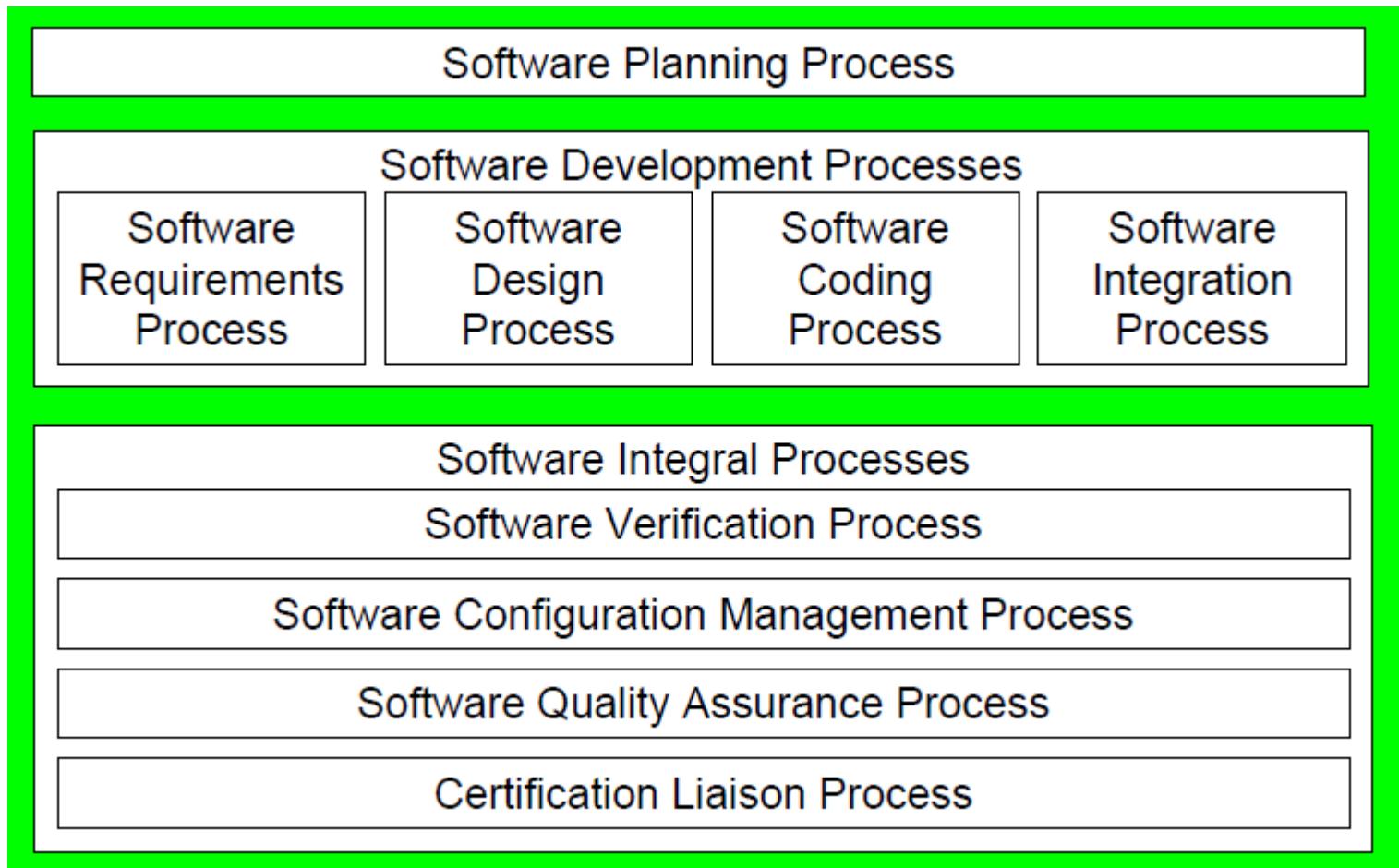
DO-178B – Development Process Model

Software
Development
Under DO-
178B - John
Joseph
Chilenski



DO-178B – Software Life Cycle Processes

Software Development Under DO-178B - John Joseph Chilenski



DO178B Document Structure

System Aspects Relating To Software Development - Section 2	Overview of Aircraft and Engine Certification - Section 10
SW Life Cycle Processes	Integral Processes
SW Life Cycle - Section 3	SW Verification - Section 6
SW Planning - Section 4	SW Configuration Mgmt. - Section 7
SW Development - Section 5	SW Quality Assurance - Section 8
	Certification Liaison - Section 9
SW Life Cycle Data - Section 11	Annex A & B
Additional Considerations - Section 12	Appendices A, B, C, & D

DO178B Objectives

	Objective	Applicability by SW Level	Output				Control Category by SW Level							
			Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Low-level Requirements comply with High-level Requirements.	6.3.2a	● ● ○						Software Verification Results	11.14	②	②	②	②



Indicates with independence

DO-178B – Processes and Outputs

DO-178B is divided into five main processes:

- Software Planning
- Software Development
- Software Verification
- Software Configuration Management
- Software Quality Assurance

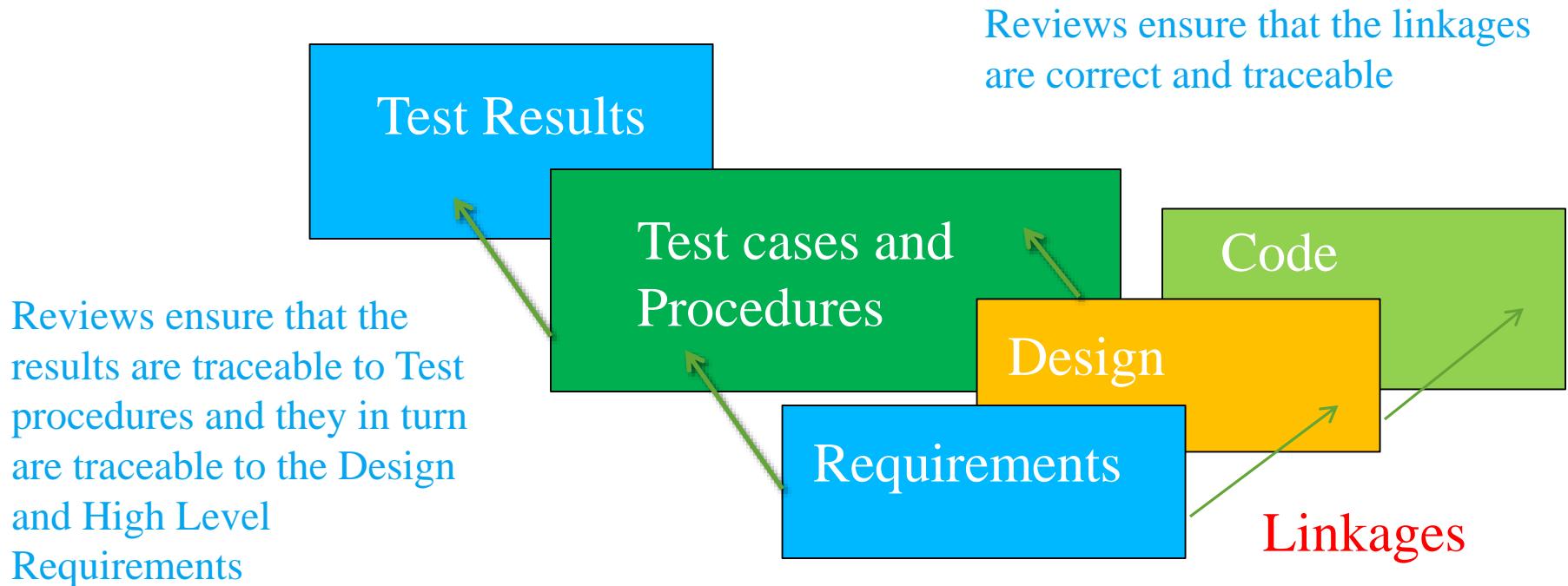
Each process has a set of expected documented outputs.

DO-178B – Documentation

Abr	Name	Type	DO-178B Section
PSAC	Plan for Software Aspects of Certification	Document	11.1
SDP	Software Development Plan	Document	11.2
SVP	Software Verification Plan	Document	11.3
SCMP	Software Configuration Management Plan	Document	11.4
SQAP	Software Quality Assurance Plan	Document	11.5
SRS	Software Requirements Standards	Document	11.6
SDS	Software Design Standards	Document	11.7
SCS	Software Code Standards	Document	11.8
SRD	Software Requirements Data	Document	11.9
SDD	Software Design Description	Document	11.1
	Source Code	Software	11.11
	Executable Object Code	Software	11.12
SVCP	Software Verification Cases and Procedures	Document	11.13
SVR	Software Verification Results	Records	11.14
SECI	Software Life Cycle Environment Configuration Index	Document	11.15
SCI	Software Configuration Index	Document	11.16
PRs	Problem Reports	Records	11.17
	Software Configuration Management Records	Records	11.18
	Software Quality Assurance Records	Records	11.19
SAS	Software Accomplishment Summary	Document	11.2

DO-178B – Traceability

All the software lifecycle processes are linked in any given application i.e. the lifecycle activities must be traceable



DO178B Final Words

This is not a methodology

The project does not operate to DO178B – This is important

- The project makes a Plan for Software Aspects of certification (PSAC)
- This is approved by the certifying authority
- This document shows how the project plans to comply with DO178B by having software development lifecycle, the data and the processes
- The certification is then a showcase and demonstration of how this compliance to the plans and standards was achieved

DO178B alone is not sufficient! You need to see more

DO178C – Updates from DO178B

Errors and Inconsistencies – addressed the known errors and inconsistencies.

Consistent Terminology – addressed issues regarding the use of specific terms such as “guidance”, “guidelines”, “purpose”, “goal”, “objective”, and “activity” by changing the text so that the use of those terms is consistent throughout the document.

Objectives and Activities – section 1.4, titled “How to Use This Document” reinforces the point that activities are a major part of the overall guidance. Annex A now includes references to each activity as well.

DO178C – Updates from DO178B

Supplements – Rather than expanding text to account for all the current Software development techniques DO-178C recognizes the use of supplements like the “Model-Based Development and Verification Supplement to DO-178C and DO-278A”.

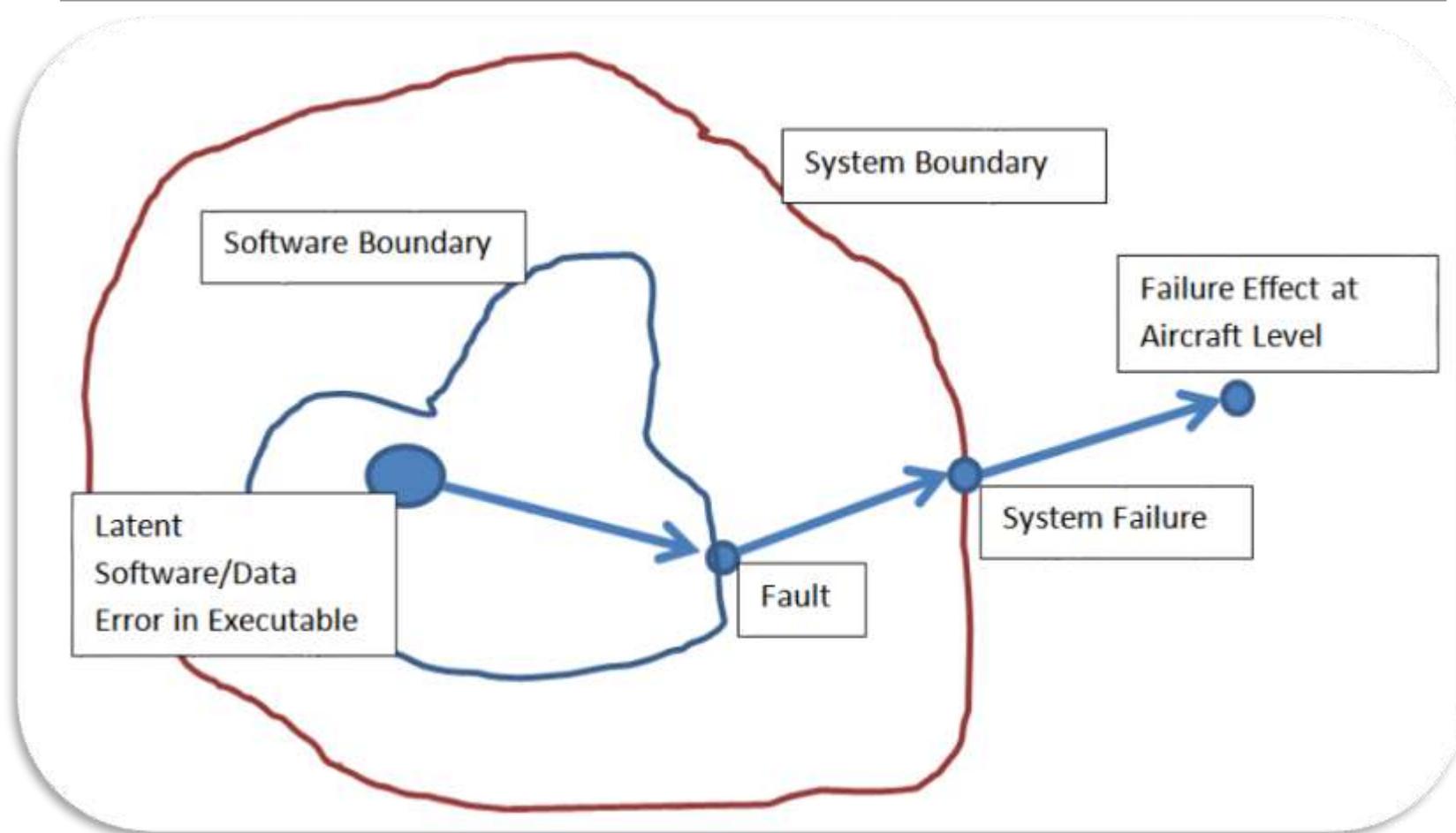
Tool Qualification (Section 12.2) – This is a major change. The terms "development tool" and "verification tool" are replaced by three tool qualification criteria that determine the applicable tool qualification level (TQL) vis-à-vis the software level. The guidance to qualify a tool is removed in DO-178C, but provided in “Software Tool Qualification Considerations”, a separate document.

DO178C – Updates from DO178B

Parameter Data Item – Software consists of Executable Object Code and/or data, and can comprise one or more configuration items. A data set that influences the behavior of the software without modifying the Executable Object Code and is managed as a separate configuration item is called a parameter data item.

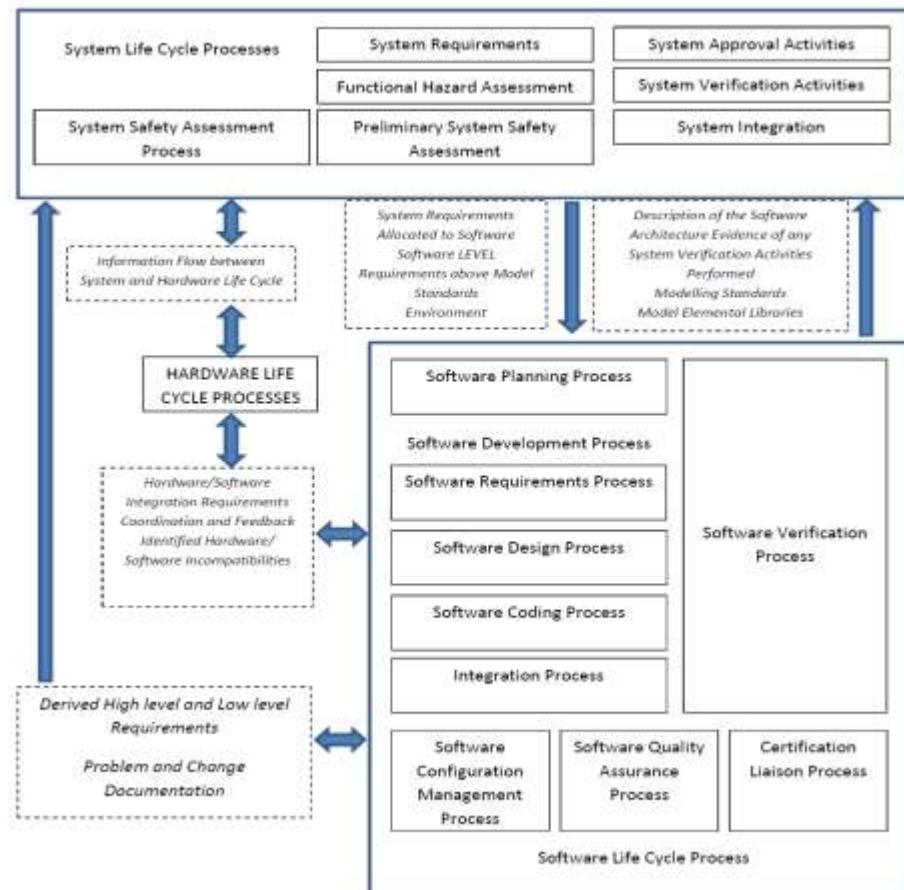
These are taken from DO178C document

System Failure

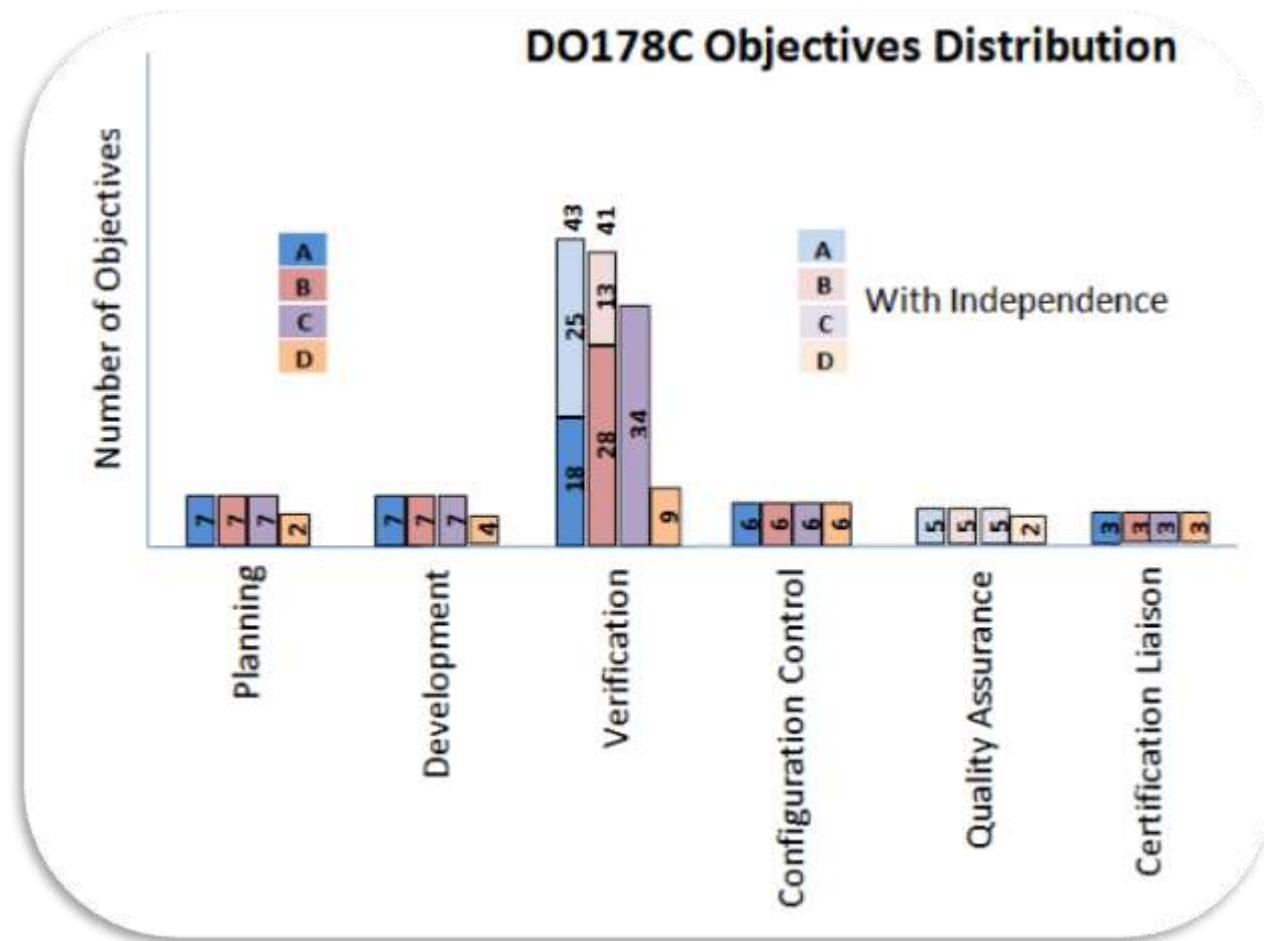


DO178C Information Flow System-Software

This diagram is not complete but just highlights a few points from the bigger picture shown in DO178C



DO178C Objectives



DO331

Model-Based Development and Verification Supplement to DO-178C and DO-278A" released on December 13, 2011

Provides guidelines for the use of models in aviation software projects

The document structure is the same as DO178C. Many of the sections have the same contents with minor changes. They have the text reproduced in italics. Changes and additions made from the DO178C are in non-italicized test.

DO331 - Definitions

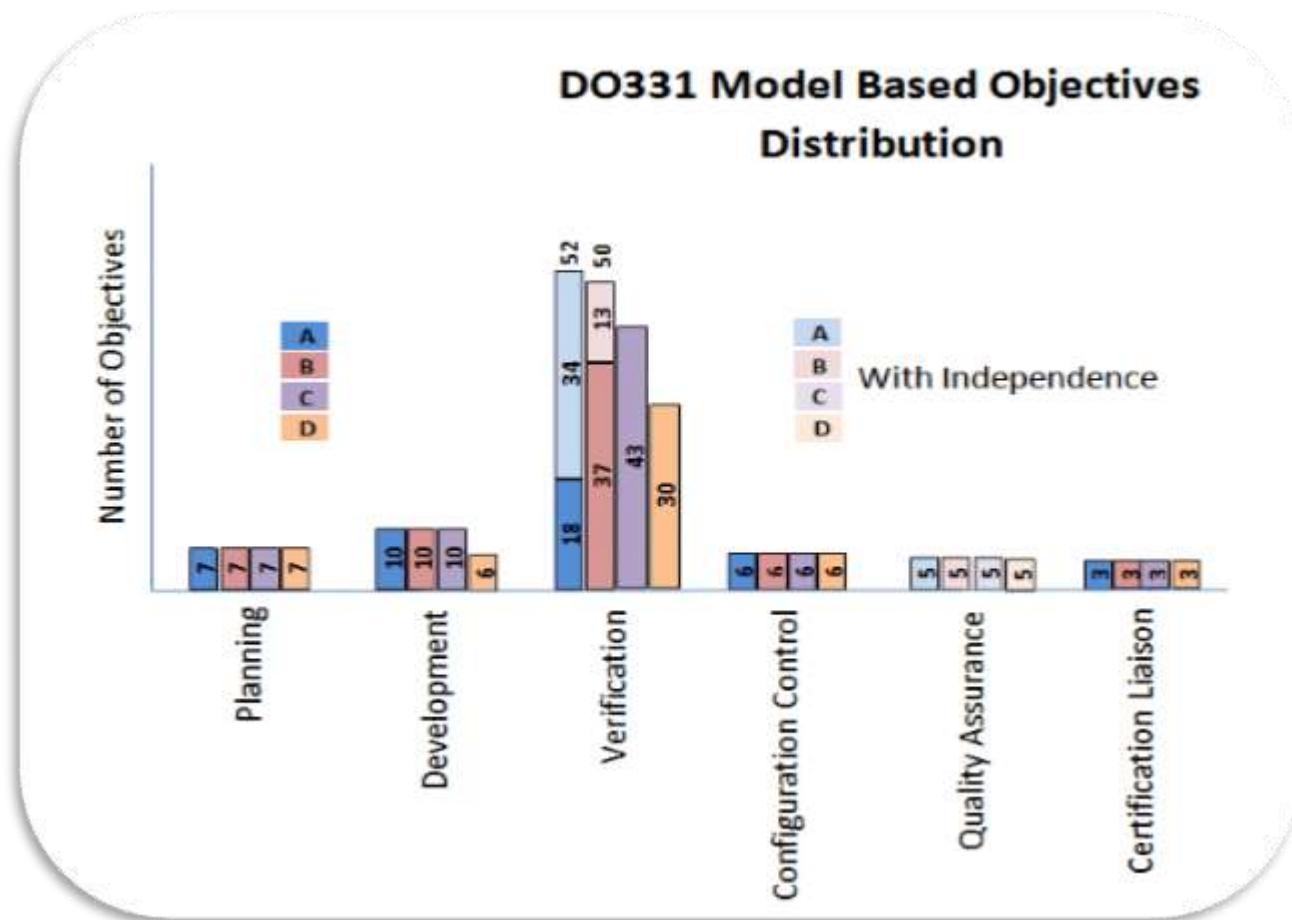
This standard defines model as

- “*An abstract representation of a given set of aspects of a system that is used for analysis, verification, simulation, code generation, or any combination thereof. A model should be unambiguous, regardless of its level of abstraction.*”

It defines Model-Based Development and Verification as

- “*a technology in which models represent software requirements and/or software design descriptions to support the software development and verification processes.*”

DO331 Objectives



DO331 – Highlights

A model cannot be classified as both a Specification Model and a Design Model.

- *The supplement defines two types of models – a specification model and a design model. The specification model is a high level of abstraction and defines the higher level requirements. This model can be simulated and its results used in the verification process. The design model is a low level model which has detailed data flow, algorithm, and can be used for autocode generation. The standard mandates a different specification for the design model with a different modeling standard.*

For Design Models, simulation may be used in combination with testing and appropriate analysis to achieve objectives related to the verification of the Executable Object Code.

DO331 – Highlights

Capabilities and limitations of the model simulator with regard to its intended use and their effects on the ability to detect errors and verify functionality should be addressed.

- *The Software Verification Plan has to highlight any such limitations and provide alternate methods to verify the functionality to completely satisfy the objective.*

Model element library – A collection of model elements used as a baseline to construct a model. A model may or may not be developed using model element libraries.

Model Libraries

The supplement recognizes the building blocks used in making bigger models. These libraries are required to be controlled in the configuration management system using baselining techniques.

A modeling standard has to be established indicating the set of model libraries and elements that are admissible for the model generation.

The element functionality should be clearly defined in the documents.

Symbols should be uniquely identified, they should not be misleading and they should be well documented.

DO331 – More work to do

When simulation is used as part of the verification activities, the means of developing the code and the means of verifying the code (for example, automatic generation of test cases) should be independent.

Software Verification Plan should address model traceability analysis, model coverage criteria, and model coverage analysis should be addressed.

Model coverage analysis should use the outputs (cases, procedures, and/or results) from one or more verification techniques: simulation, testing, and/or other appropriate techniques.

Errors found due to Model Based Tests

Inadequate end-to-end numerical resolution.

Incorrect sequencing of events and operations.

Failure of an algorithm to satisfy a software requirement.

Incorrect loop operations.

Incorrect logic decisions.

Failure to process correctly legitimate combinations of input conditions.

Incorrect responses to missing or corrupted input data.

Incorrect computation sequence.

Inadequate algorithm precision, accuracy, or performance.

Incorrect state transitions.

DO333 – Formal methods

DO 333 is Formal Methods Supplement to DO-178C and DO-278A. This document enables us to use Formal Methods in the safety critical design and development process.

Formal methods are mathematically based techniques for the specification, development, and verification of software aspects of digital systems.

The mathematical basis of formal methods consists of formal logic, discrete mathematics, and computer-readable languages.

The applicability of formal methods to any particular software development activity is bounded by the ability to construct an appropriate formal model

Formal analysis can provide guarantees or proofs of software properties and compliance with requirements.

DO333 – Formal methods

An analysis method can only be regarded as formal analysis if its determination of a property is sound.

Sound analysis means that the method **never asserts a property to be true when it is not true**.

The converse case, the assertion that a property is false when it may be true, colloquially “the raising of false alarms”, is a usability issue but not a soundness issue.

It is acceptable for a method to return “don’t know” or not to return an answer when trying to establish whether a property holds, in which case additional verification is necessary.

DO333 – Formal methods

Formal methods, because of their mathematical basis, are capable of

- Unambiguously describing requirements of software systems.
- Enabling precise communication between engineers.
- Providing verification evidence such as consistency and accuracy of a formally specified representation of software.
- Providing verification evidence of the compliance of one formally specified representation with another.

Verbatim from DO 333

DO333 – Formal methods

Formal methods are also capable of demonstrating properties of software systems such as

- Freedom from exceptions.
- Freedom from deadlock.
- Non-interference between different levels of criticality.
- Worst case execution time.
- Bounds on stack size during execution.
- Freedom from unintended function.
- Correct synchronous or asynchronous behavior.

Formal methods

I am providing examples of usage of formal methods later in this presentation. The standard is mentioned here with very minimal details. A good read to understand formal methods and their use in aerospace is

NASA/CR-2014-218244



Formal Methods Case Studies for DO-333

*Darren Cofer and Steven P. Miller
Rockwell Collins, Inc., Cedar Rapids, Iowa*

Tips

The source of all development is requirements. Get it right, review it, revisit it after every design change. This is the Bible. Preach it to all the groups involved in the development, verification and certification.

Tips

“it is difficult for engineers to change human nature ... we should accept people as we find them and try to remove opportunities for error by changing the work situation ... the method of working.

-Trevor Kletz, An engineer's view of human error

DO-178B– Certification

Certification - legal recognition by the certification authority that a software product complies with the requirements

Certification is done on the individual application of the product

Coding practices must be certified to ensure things like "dead code" are not allowed.

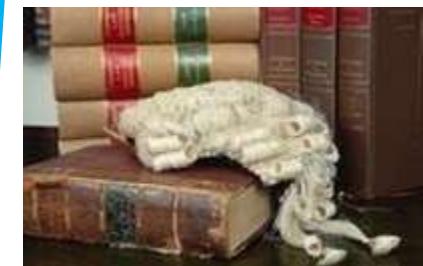
Certification requires that 'full testing' of the system and all of its components (including firmware) be done on the target platform in the target environment.

Certification requires code testing at the MC/DC level. Coverage proof is to be provided by the Requirement based tests.

Tips



Certification is a legal artifact. Once all the technical aspects are correctly carried out there is a lot of documentation effort involved to present the data to the authorities. Remove your engineering hat and wear a lawyer's wig!



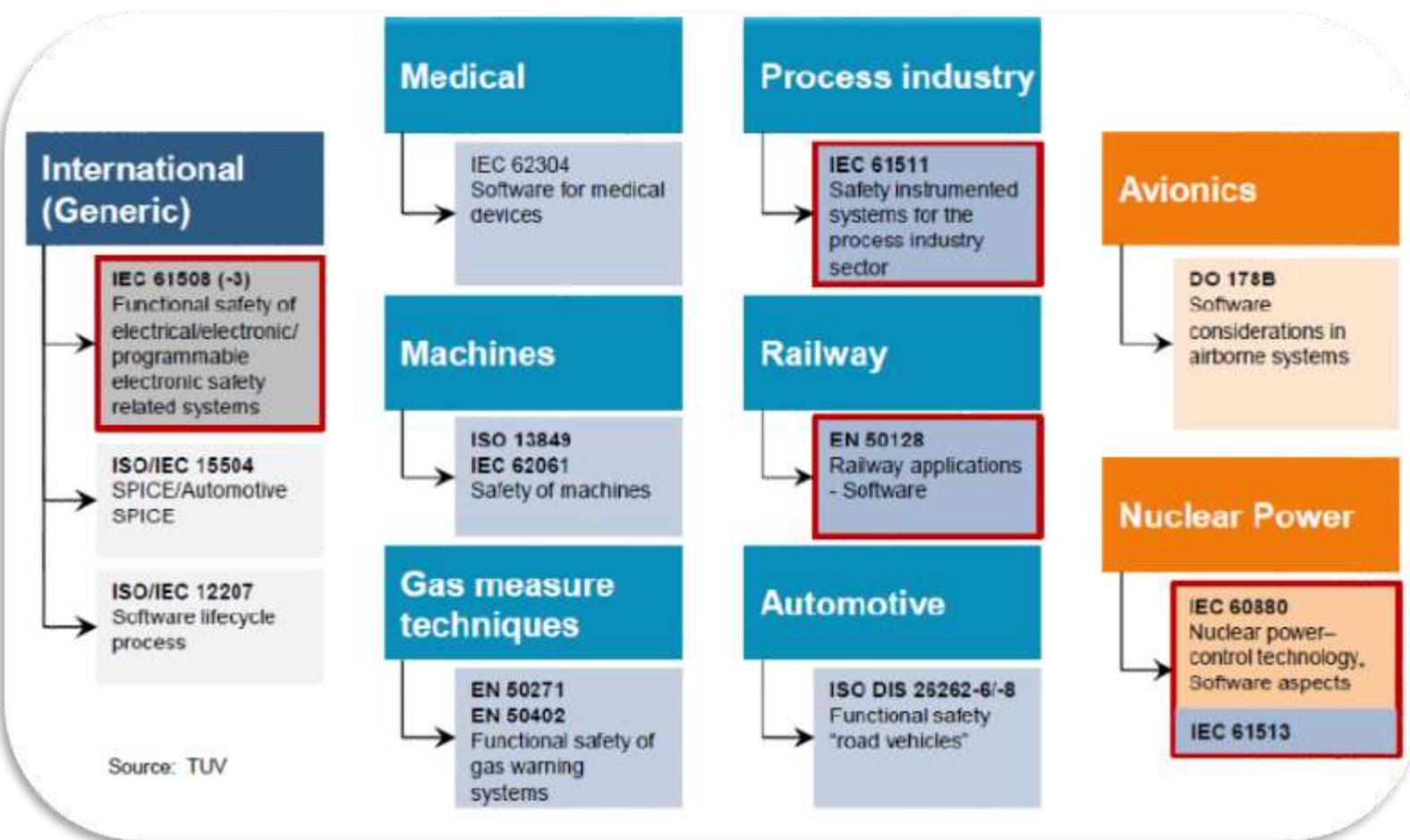
Tips

The aircraft data has a long life (as long as the aircraft is in service). Ensure that the data, requirements, test cases, test results and reviews have enough “simple English” text to be able to be understood and tasks repeated by anyone. People may not last that long (in the project)!

Tips

The essence of DO-178B is that the compiled code be directly traceable to a requirement and a test routine, and that no extraneous code outside of this process is included in the build. The proof of this should be generated on the actual system that will fly. The traceability should be bidirectional.

Other Standards



Automotive Standard

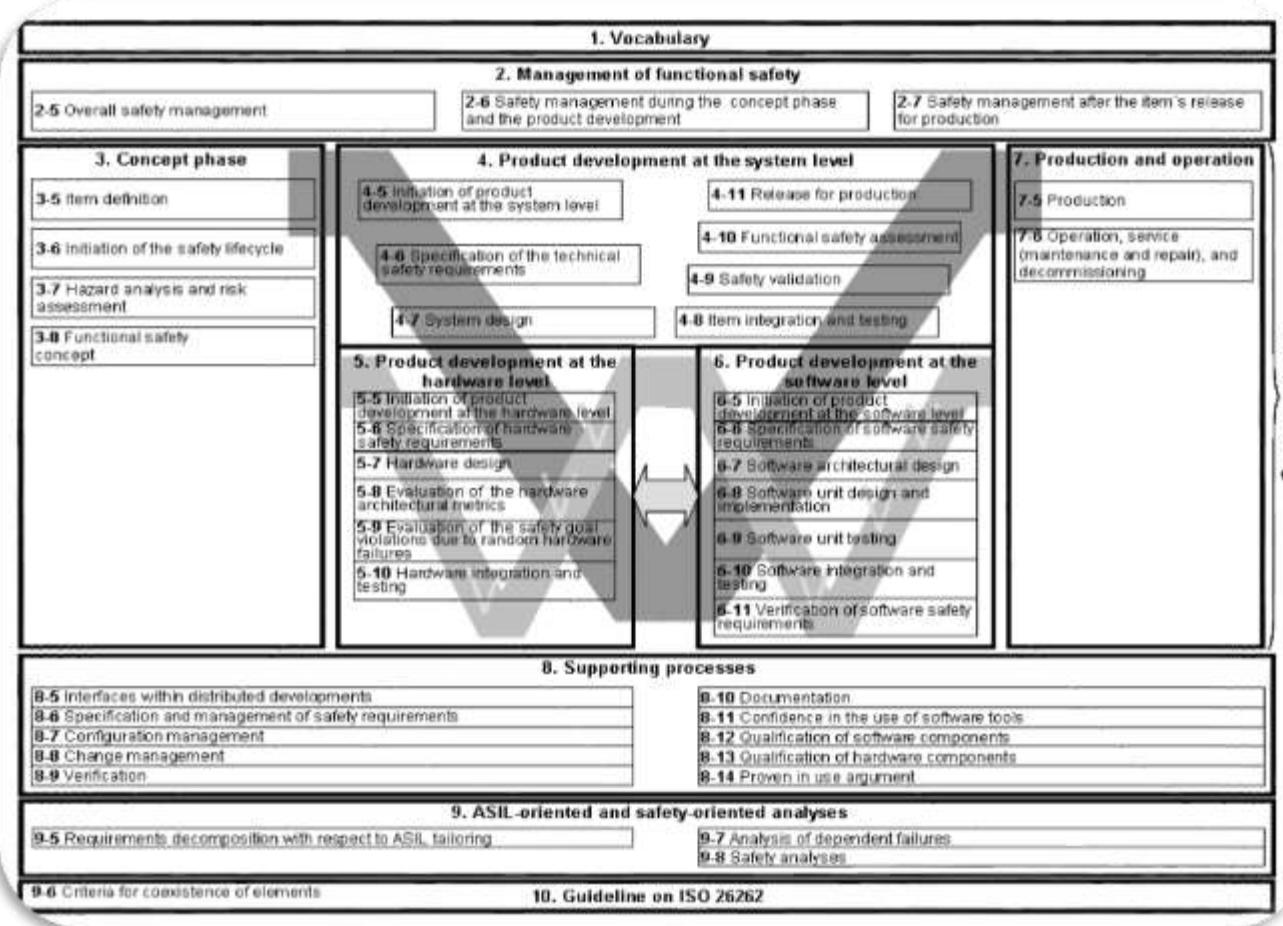
ISO 26262 is the Automotive safety standard for mass production of passenger cars

An excellent comparison of DO178B and ISO 26262 is available in Stephan Weiledner, Robert Hilbrich, Matthias Gerlach - Can Cars Fly? From Avionics to Automotive: Comparability of Domain Specific Safety Standard

Yes They
Can!!



ISO 26262 Document



Source

[ISO/FDIS 26262-3:2010\(E\)](#)

ASIL Determination

Classes of Severity

Class	S0	S1	S2	S3
Description	No injuries	Light and moderate injuries	Severe and life-threatening injuries (survival probable)	Life-threatening injuries (survival uncertain), fatal injuries

Classes of Probability

Class	E0	E1	E2	E3	E4
Description	Incredible	Very low probability	Low probability	Medium probability	High probability

Classes of Controllability

Source: ISO/FDIS 26262-3:2010(E)

Class	C0	C1	C2	C3
Description	Controllable in general	Simply controllable	Normally controllable	Difficult to control or uncontrollable

ASIL Determination

		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

ISO 26262 Recommendation

Methods		ASIL			
		A	B	C	D
1a	Requirements-based test ^a	++	++	++	++
1b	Fault injection test ^b	+	++	++	++
1c	Back-to-back test ^c	+	+	++	++

^a A requirements-based test denotes a test against functional and non-functional requirements.

^b A fault injection test uses special means to introduce faults into the test object during runtime. This can be done within the software via a special test interface or specially prepared hardware. The method is often used to improve the test coverage of the safety requirements, because during normal operation safety mechanisms are not invoked.

^c A back-to-back test compares the responses of the test object with the responses of a simulation model to the same stimuli, to detect differences between the behaviour of the model and its implementation.

- “++” The method is highly recommended for the identified ASIL.
- “+” The method is recommended for the identified ASIL.
- “o” The method has no recommendation for or against its usage for the identified ASIL.

Comparison D0178C and ISO26262

Coverage

A B C D



	explained.												
3	Test coverage of high-level requirements is achieved.	6.4.4.a	6.4.4.1	●	○	○	○	Software Verification Results	11.14	②	②	②	②
4	Test coverage of low-level requirements is achieved.	6.4.4.b	6.4.4.1	●	○	○		Software Verification Results	11.14	②	②	②	
5	Test coverage of software structure (modified condition/decision coverage) is achieved.	6.4.4.c	6.4.4.2.a 6.4.4.2.b 6.4.4.2.d 6.4.4.3	●				Software Verification Results	11.14	②			
6	Test coverage of software structure (decision coverage) is achieved.	6.4.4.c	6.4.4.2.a 6.4.4.2.b 6.4.4.2.d 6.4.4.3	●	●			Software Verification Results	11.14	②	②		

Methods

ASIL

	ASIL	A	B	C	D
1a Statement coverage		++	++	+	+
1b Branch coverage		+	++	++	++
1c MC/DC (Modified Condition/Decision Coverage)		+	+	+	++



Primary Similarities

Both standards focus on integrated safety measures

Both standards define a work flow consisting of several processes

They both have five levels of criticality. Level E which is the least critical in DO178 is QM is ISO26262. Levels A to D (highest to least in DO178) is ASIL D to A in equivalence.

Planning the development process is common to both.

Both standards require a specific set of artifacts to be produced during the development process

Both mention MC/DC coverage at the highest criticality

Primary Differences

ISO26262 defines recommended procedures to ensure safe software. DO178 specifies objectives to be satisfied. Normally we make checklist out of these objectives so that we can ensure that they are fulfilled.

DO178B specifies explicitly its relevance to the avionics software certification. ISO26262 does not foresee official certification.

DO178B is primarily focused on software development. ISO26262 target the complete item development. Avionic system developers have to look at other standards along with DO178B for a complete product development.

IEC 61508 – Automation in Industry

Risk acceptability depends on the **frequency** of the event that causes the degradation and the **severity** of the degraded state.

IEC 61508 Risk Matrix			Severity			
			Negligible	Marginal	Critical	Catastrophic
			Minor injuries at worst	Major injuries to one or more persons	Loss of a single life	Multiple loss of life
Frequency	Frequent	$> 10^{-3}$	Undesirable	Unacceptable	Unacceptable	Unacceptable
	Probable	10^{-3} to 10^{-4}	Tolerable	Undesirable	Unacceptable	Unacceptable
	Occasional	10^{-4} to 10^{-5}	Tolerable	Tolerable	Undesirable	Unacceptable
	Remote	10^{-5} to 10^{-6}	Acceptable	Tolerable	Tolerable	Undesirable
	Improbable	10^{-6} to 10^{-7}	Acceptable	Acceptable	Tolerable	Tolerable
	Incredible	$\leq 10^{-7}$	Acceptable	Acceptable	Acceptable	Acceptable

Source: Antoine Rauzy “Safety Integrity Levels”

IEC 61508 Safety Integrity Levels

	Low Demand Mode: Probability of Failure on Demand (PFD _{average})	High Demand Mode: Probability of Failure per Hour (PFH)
SIL 4	10^{-5} to 10^{-4} (RFF > 10000)	10^{-9} to 10^{-8}
SIL 3	10^{-4} to 10^{-3} ($1000 \leq$ RFF < 10000)	10^{-8} to 10^{-7}
SIL 2	10^{-3} to 10^{-2} ($100 \leq$ RFF < 1000)	10^{-7} to 10^{-6}
SIL 1	10^{-2} to 10^{-1} ($10 \leq$ RFF < 100)	10^{-6} to 10^{-5}

Source: Antoine Rauzy “Safety Integrity Levels”

IEC 62304 - Medical

Medical device developers follow

IEC 61508 with an emphasis on IEC 61508-3:2010,
Functional safety of electrical/electronic/programmable
electronic safety-related systems – Part 3: Software
requirements

IEC 62304:2006 Medical device software – Software life
cycle processes.

Here processes consist of activities, activities consist of
tasks.

Three classes of safety criticality A to C

Source: Vera Pantelic "Systems and Software Engineering Standards for the Medical Domain"

Medical Classes

These are based on SIL levels

This classification is based on the potential to create a hazard that could result in an injury to the user, the patient or other people

Class	Failure Impact
A	No injury or damage to health is possible
B	Non serious injury is possible
C	Death or serious injury is possible

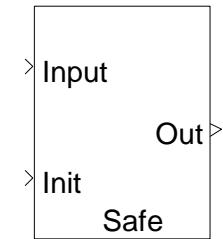
Tips

There are many religions and doctrines in the world today. They all talk about the same thing in different ways!!

All standards talk about safety levels and **what you better do about it !!**

Control Algorithms

```
ns=[A1 A2];
ds=[1 B2];
[Nz,Dz]=c2dm(ns,ds,DT,'tustin');
sim('digital1order');
INP = inp(1);
out = inp(1);
po=out;
Pi=INP;
B=[];
for i = 1:length(o)
    INP=inp(i);
    if init(i) > 0
        INP = inp(i);
        out = inp(i);
        po=out;
        Pi=INP;
    else
        out=Nz(1)*INP+Nz(2)*Pi-Dz(2)*po;
        po=out;
        Pi=INP;
    end
    B=[B;[INP out]];
end
o=B(:,2);
err=abs(o-o1);
iie = find(abs(o > 100));
err(iie)=abs(err(iie)./o(iie));
```



DTF-I-1S1
Num Coeff A 0 = Nz(1)
Num Coeff A 1 = Nz(2)
Den Coeff B 1 = Dz(2)
Sample Time = DT
Discrete Transfer Function
I order 1 State

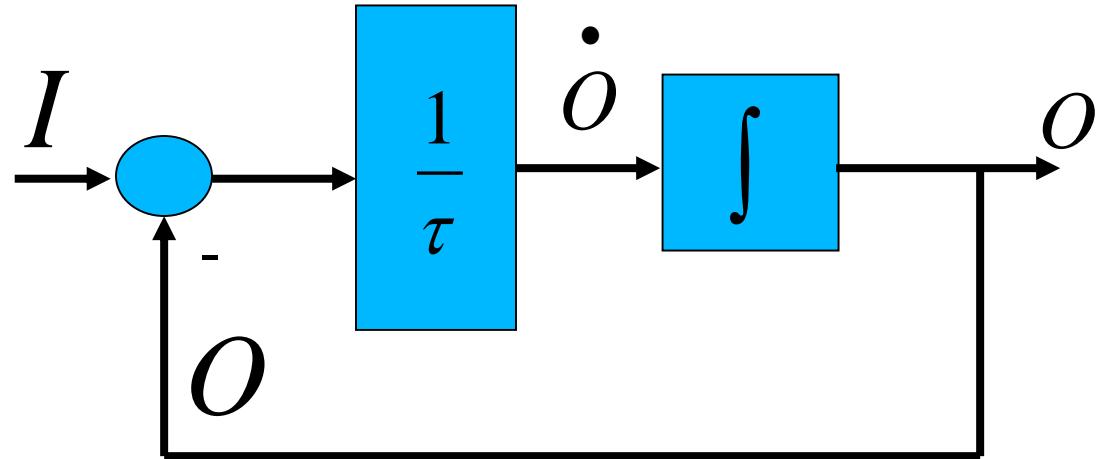
First Order Filter

$$\frac{O}{I} = \frac{1}{1 + s\tau}$$

$$O + s\tau \times O = I$$

$$O + \tau \times \dot{O} = I$$

$$\dot{O} = \frac{I - O}{\tau}$$



First Order Filter

The first order filter is represented by the following transfer function

$$\frac{O}{I} = \frac{Nz(1) + Nz(2)z^{-1}}{Dz(1) + Dz(2)z^{-1}}.$$

Nz and Dz are computed using the Tustin Transform

$$s = \frac{(2/T) + (z - 1)}{z + 1}.$$

The term z^{-1} denotes the previous value. S is laplace denoting differentiation

First Order Filter

If init > 0

Set the previous values of output and input, to input

Set output equal to input

Else

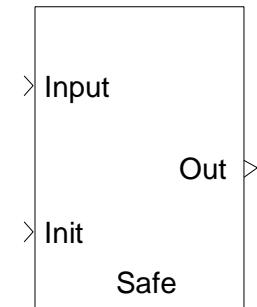
Compute using the following equation

$$\text{out} = \text{Nz}(1) * \text{inp} + \text{Nz}(2) * \text{pri} - \text{Dz}(2) * \text{pro};$$

End

$\text{pro} = \text{out}$

$\text{pri} = \text{inp}$



DTF -I-1 S1

Num Coeff A 0 = Nz (1)

Num Coeff A 1 = Nz (2)

Den Coeff B 1 = Dz (2)

Sample Time = DT

Discrete Transfer Function

I order 1 State

Importance of Initialization

Initial transients are avoided

A constant input will give a constant output. The filter acts as gain. Note: This is also sometime specified as output derivative is zero for constant input

The system comes up very fast and this is very important in a safety critical system

Bank of filters can be used with switching between them based on conditions

Second Order Filter

The Second order filter is represented by the following transfer function

$$\frac{O}{I} = \frac{Nz(1) + Nz(2)z^{-1} + Nz(3)z^{-2}}{Dz(1) + Dz(2)z^{-1} + Dz(3)z^{-2}}.$$

Nz and Dz are computed using the Tustin Transform

The term z^{-1} denotes the previous value and z^{-2} denotes previous to the previous value

Second Order Filter

If init > 0

Set the all previous values of output and input to input

Set output equal to input

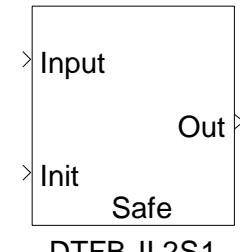
Else

Compute using the following equation

out=Nz(1)*inp+Nz(2)*pri+Nz(3)*ppri

-Dz(2)*pro-Dz(3)*ppro;

End



Num Coeff A 0 = a1
Num Coeff A 1 = a2
Num Coeff A 2 = a3
Den Coeff B 1 = b2
Den Coeff B 2 = b3
Sample Time = DT
Discrete Transfer Function
Bilinear II Order 2 State

Set the previous values like in the case of first order filter

Tustin with Prewarping

- Transfer the S Domain to Z Domain

- Tustin approximation
(Bilinear)

$$s' = \frac{2}{T_s} \frac{z - 1}{z + 1}$$

- Better (freq. resp. matches)
ill-defined at and close to $z = -1$

- Tustin with
Prewarping

- Ensures frequency response
matches at critical frequency
w. T_s is the sampling time

$$s' = \frac{\omega}{\tan(\omega T_s/2)} \frac{z - 1}{z + 1}$$

Use of Filters in Control Systems

Normally used to reduce noise

Filter out high frequency components of a system so that it behaves in a slower manner. i.e. It does not respond very fast to the changing input

To modify the response of the output to transients

It could be a lead/lag filter or a washout filter. This is used to increase stability of control system.

Second order filters are normally used as notch filters to cut out unwanted frequencies.

The second order filters introduce additional phase lag in the system and can cause erosion of margins. They have to be used with care

Tips

Filter coefficients may be specified as time constants leaving the conversion to discrete to the implementer. It is quite likely (it has happened). This has led to problems. Filters have been implemented with 4th decimal place accuracy. Designers should specify the exact digital coefficients.

Tips

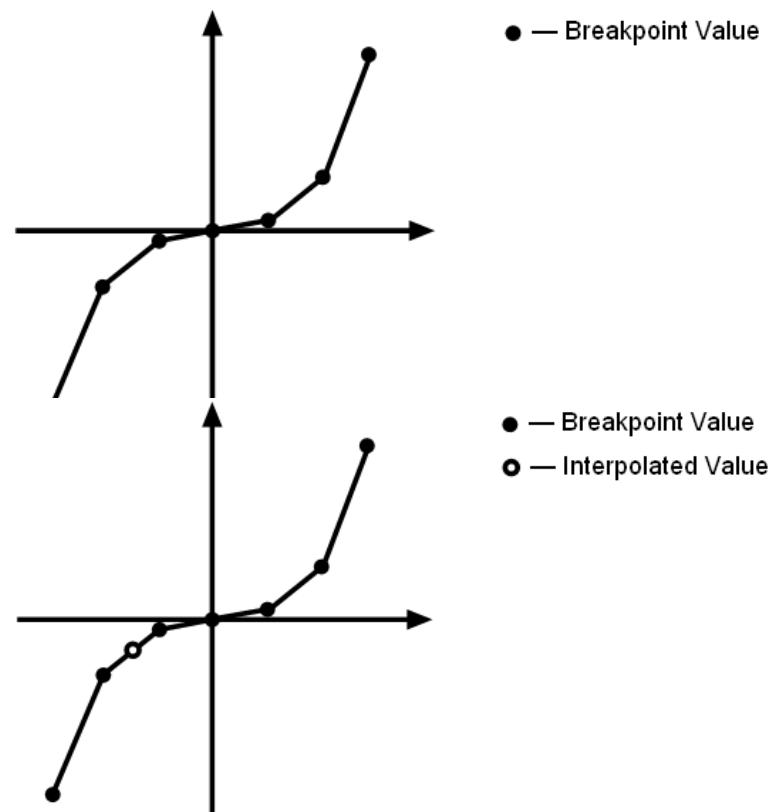
Structural filters need to be very accurate to notch out the structural frequency. They usually use prewarped Tustin to get a better digital frequency precision. During coding with a single precision one can specify 6 decimal place accuracy. Designer need to take into account this during their optimal design. The design tools normally work in double precision. Don't blame the implementer if the frequency response does not match with design.

1-D Interpolation

x	-3	-2	-1	0	1	2	3
y	-27	-8	-1	0	1	8	27

Linearly interpolated value

x	-3	-2	-1.5	-1	0	1	2	3
y	-27	-8	-4.5	-1	0	1	8	27



1-D Interpolation

Given a table of X and Y values and a value of x for which y is required

Find the two values of X between which x lies

This gives index i and index i+1

Find the slope $s=Y(i+1)-Y(i)/((X(i+1)-X(i))$

$$y = (x-X(i))*s + Y(i)$$

- › Index
- › Fraction Inter
- › Size Safe

1-D Table
Y Axis Data = YT
1-D Look Up

Normally extrapolation is not used in the safety critical control systems. One can always extrapolate offline and use them as additional values in the table

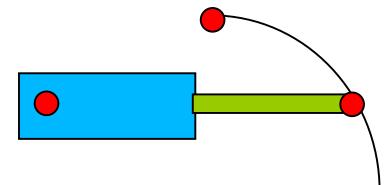
Tips

In case of the Indian LCA program the aircraft behaved abnormally while passing through 1 mach. This was due to the airdata lookup table which had extrapolation. The extrapolated value was wrong and this caused the problem. It is a good practice to extrapolate before hand during design, test out the system and use only interpolation in the control code. Better be safe!

Uses of 1-D Interpolation

Normally 1-D Interpolation is called table lookup and is used to modify the input/output relation

- A linear actuator moves forward and backward measured in inches. This is connected to the aircraft surface which move in degrees. But there is a non linear relation from inches to degrees then we use a 1-D lookup
- A control gain has to change on how fast the vehicle is moving then we will use a 1-D lookup
- The pilot should move the surface very fast when he is close to zero but he should move it slowly when he is greater than 10 degrees. Use 1-D to modify pilot command



2-D Interpolation

		Altitude			
		1 Km	2 km	5 km	10 km
	200 kmph	1.42	1.56	1.8	1.92
	400 kmph	2.45	2.56	2.79	3.1
	800 kmph	3.67	3.81	3.91	4.12
	1000 kmph	4.78	4.90	5.2	5.2

2-D Interpolation

Given a table of X and Y values, a matrix Z of values. Given a value of x and y compute z from the table lookup.

Find the two values of X between which x lies

This gives index i and index i+1

Find the two values of Y between which y lies

This gives index j and index j+1

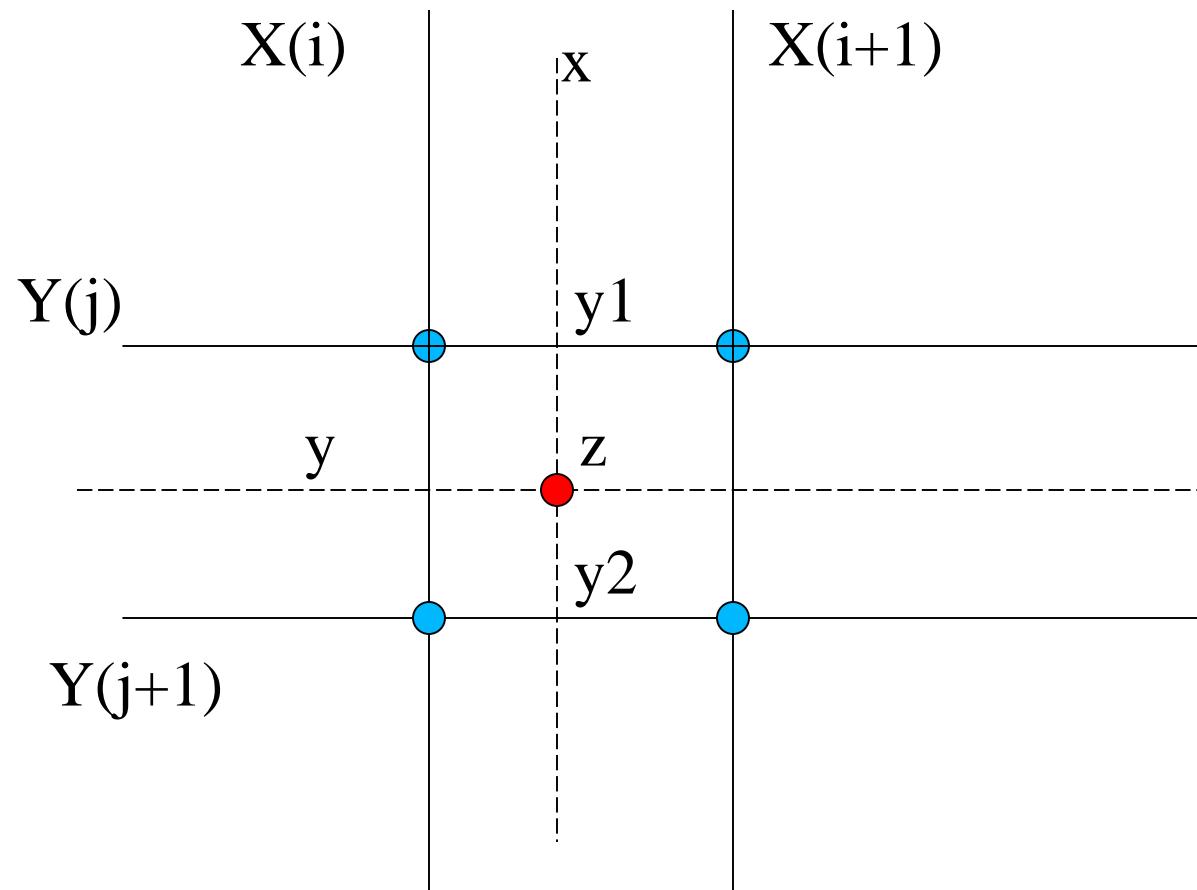
Compute y_1 at x by using $Y(i,j)$ and $Y(i+1,j)$

Compute y_2 at x by using $Y(i,j+1)$ and $Y(i+1,j+1)$

Compute z by using y_1 and y_2

Use 1-D interpolations for the computation

2-D Interpolation



Rate Limiter

All physical systems have a rate limit. A car can go at 100 kmph when the accelerator is pressed fully down. That is the velocity or rate limit.

In aerospace the aircraft surfaces can move at a finite rate for a specific command. This is the system limit which cannot be crossed.

It is dangerous to hit the surface rate limits. In case the rate limits are hit the surface does not respond as required by the control system and the aircraft can **and has** crashed.

Rate limiter blocks are introduced in control systems to avoid the commands causing a rate limit of surfaces.

Rate Limit Accidents

Proceeding of the 2004 American Control Conference
Boston, Massachusetts June 30 - July 2, 2004

FrM04.5

Phase compensation design for prevention of PIO due to actuator rate saturation

I. Alcalá, F. Gordillo and J. Aracil
Dpto. Ingeniería de Sistemas y Automática
University of Seville (Spain)
Email: ismael@cartuja.us.es

Abstract—This paper presents a simple and effective solution for type 2 pilot-induced oscillations due to rate limit in the control surface. The proposed method uses a nonlinear filter that compensates the phase of the control signal before feeding the actuator. The structure of this filter has advantages over previous realizations that allow tuning simplicity considering limit cycle prevention as control specification. Simulation results demonstrate the good performance of the proposed compensation.

I. INTRODUCTION

All aircraft control surfaces have restrictions when the actuators are operating at their maximum capacity. One of these limitations is known as rate limit and it relates to the maximum speed at which an actuator can follow changes in the input signal (see figure 1). Furthermore, in fly-by-

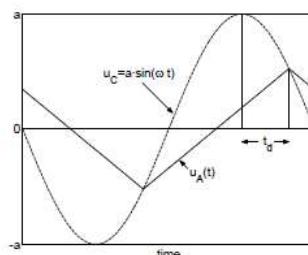


Fig. 1. Input signal to the actuator (dashed), output signal of the actuator with maximum rate denoted as m (solid) and time delay experienced t_d .



https://www.youtube.com/watch?v=k6yVU_yYtEc



http://www.nt.ntnu.no/users/skoge/prost/proceedings/acc04/Papers/0836_FrM04.5.pdf

Rate Limiter

During First frame: $y = IC$

During Normal Operation:

$PosDelta = \text{previous output} + PosRate * T$

$NegDelta = \text{previous output} + NegRate * T$

If ($x > PosDelta$) where x is input

$y = posDelta$

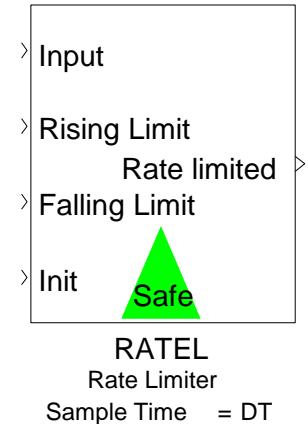
Else if ($x < NegDelta$)

$y = negDelta$

Else

$y=x$

Here $NegRate$ (say -10 in/s) is the negative slew or rate limit and $PosRate$ is the positive rate limit (say 12 in/s) and T is the sampling time



Tips

Rate limiters are difficult to understand for engineers who are just starting their career in control system design and testing. They often confuse it with amplitude limiters. It is worthwhile experimenting with this block to understand. If the output of the block is **differentiated** one should get constants where the limits are hit.

Integrators

Integrators are used in PID controllers

They are used as accumulators. If the pilot wants to fine tune aircraft nose up or down command he uses a trim button. The output of this button is integrated to generate a up/down command. The more time the button is pressed the higher the integrator output.

They are used to keep count of time. If a flag is set for some time the integrator ramps up and if the value is greater than some threshold one can latch a failure.

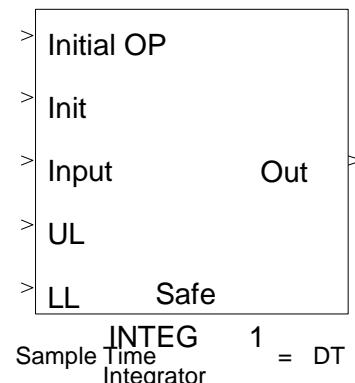
Integrators are used to make filters in the way an analog filter is designed

Anti windup Integrators

Integrators can “run away” if a constant input is given. It is possible for the output variable to have very large values. This is called windup

This is not a very safe situation and integrator have a limit on the state. This is called anti windup.

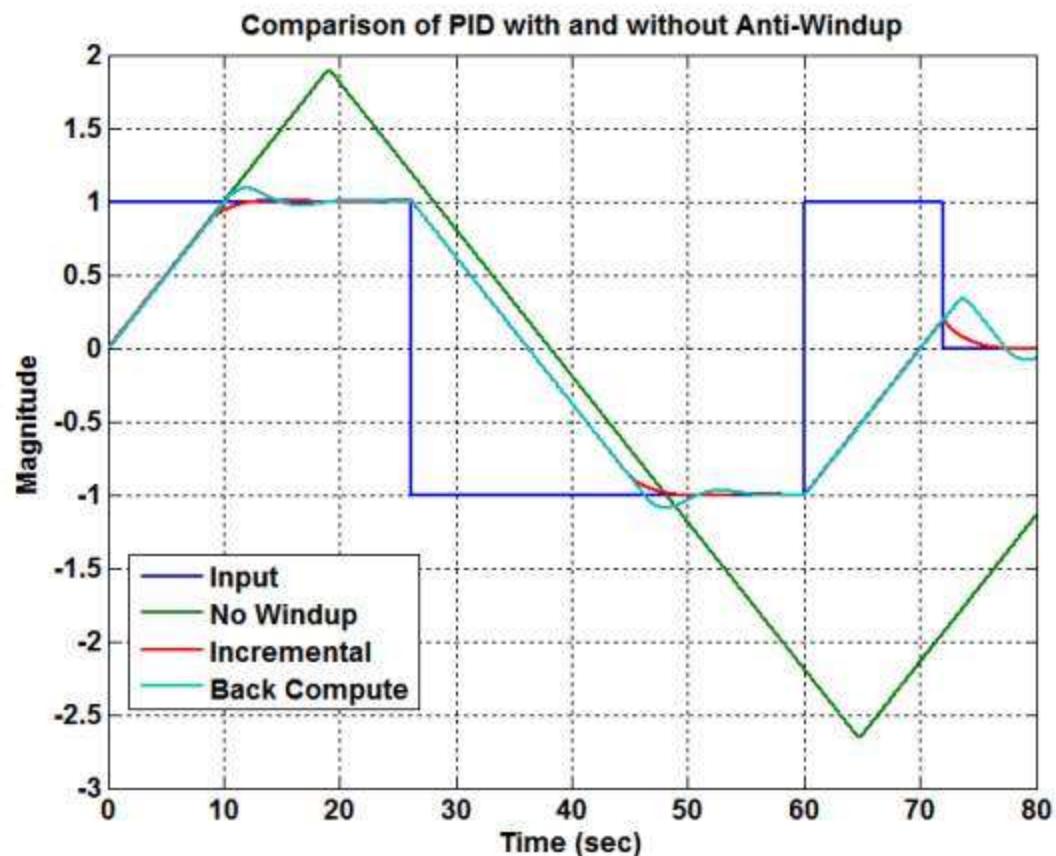
All integrators in a safety critical system have anti windup



Anti windup PID

Integrators without antiwindup can cause such a behavior in PID control systems

<http://safetycriticalmbd.wordpress.com/2014/03/22/be-careful-how-you-windup-your-integrators/>



Integrator – Euler Forward

Inputs: x , IC

Output : y

During first frame : $y = IC$

During normal operation :

- $y(i) = y(i-1) + T*x(i-1)$,

where T = sample time.

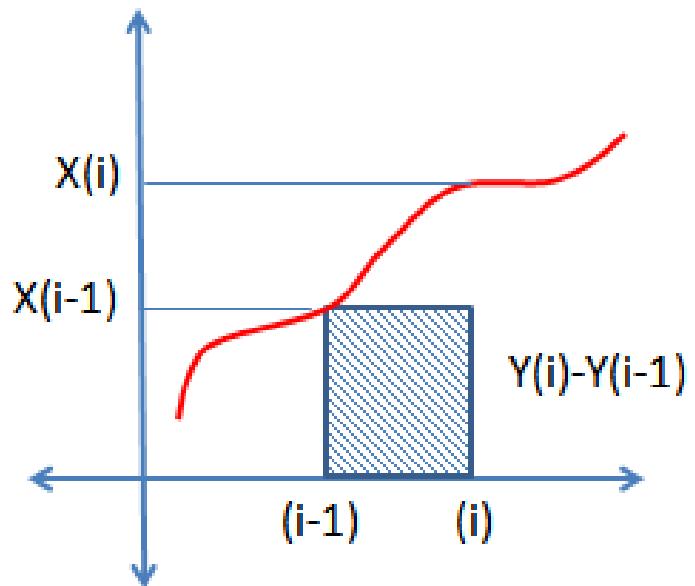
Anti windup

If $y(i) > poslim$

$y(i) = poslim$

Elseif $y(i) < neglim$

$y(i) = neglim$



Integrator – Euler Backward

Inputs: x , IC

Output : y

During first frame : $y = IC$

During normal operation :

- $y(i) = y(i-1) + T*x(i),$

where T = sample time.

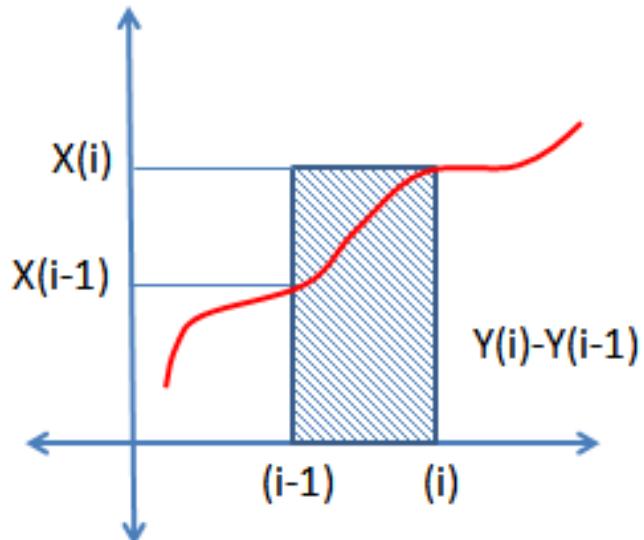
Anti windup

If $y(i) > poslim$

$y(i) = poslim$

Elseif $y(i) < neglim$

$y(i) = neglim$



Integrator - Tustin

Inputs: x , IC

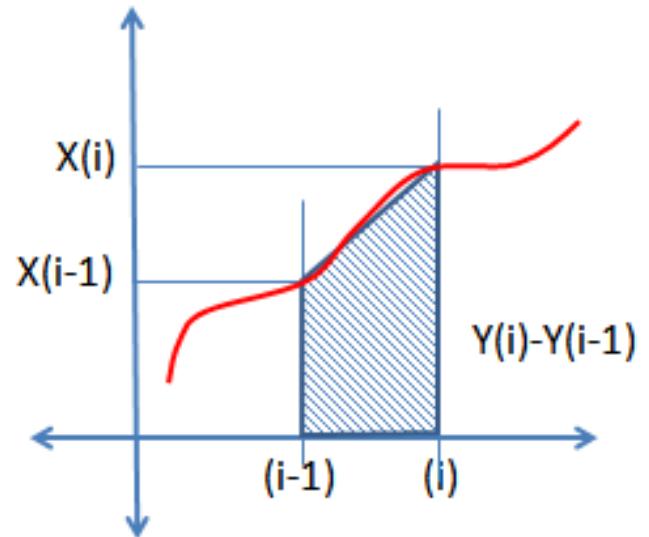
Output : y

During first frame : $y = IC$

During normal operation :

$$y(i) = y(i-1) + T/2 * (x(i-1) + x(i))$$

where T = sample time.



Tips

It is better if designers specify the integrator algorithm they want to implement in code. There is a tradeoff between speed and accuracy in the two algorithms. In Indian programs I have found errors in integrator implementation due to this. The difference is small but it exists.

What integration algorithm?

This is a question that is often asked. I have tried to address this in detail here

- <http://safetycriticalmbd.wordpress.com/2014/02/13/tustin-backward-or-forward-does-it-matter/>

A good algorithm is Backward Euler. This is the simplest to implement and stable.

Forward Euler can become unstable as the sampling time increases. Be very aware of this fact while coding integrators.

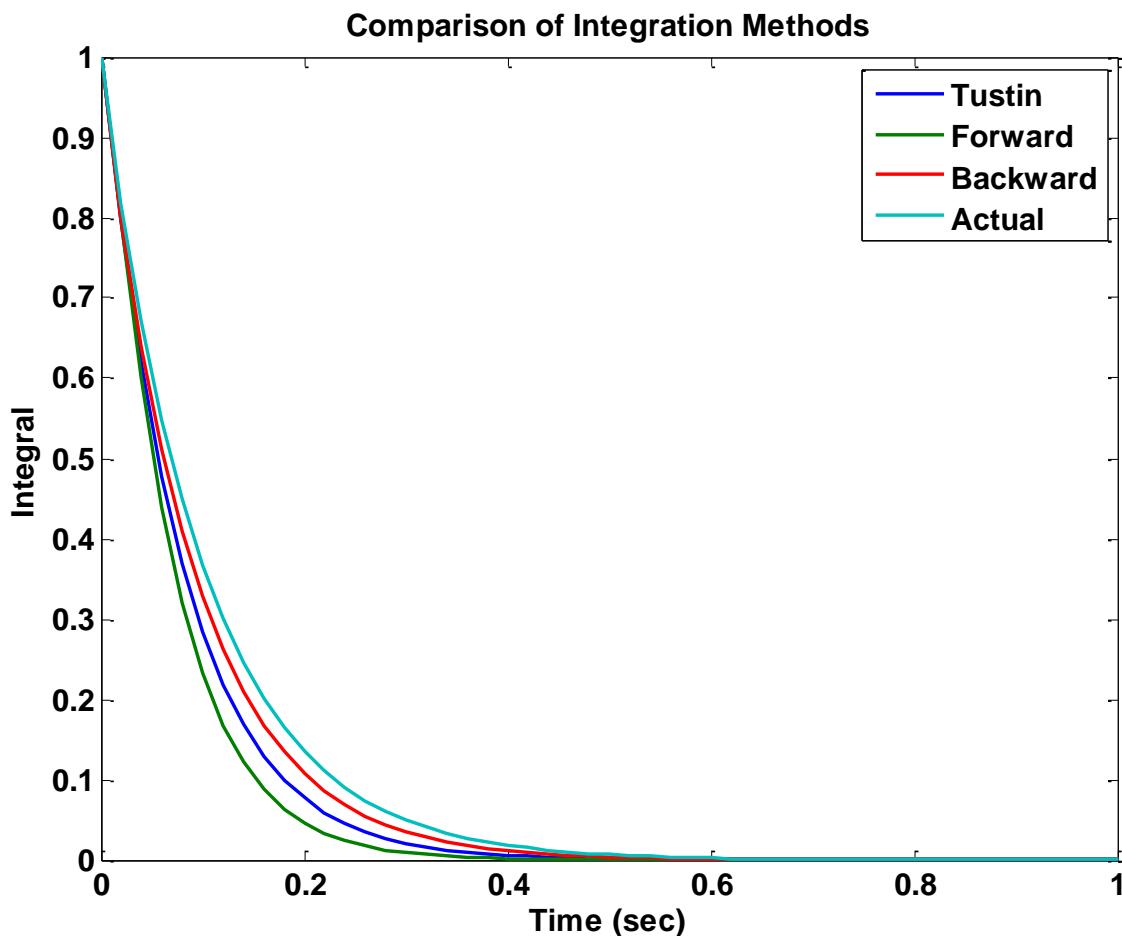
I have put a Simulink model on the Mathworks website to illustrate this.

What integration algorithm?

This gives a comparison of integration algorithms

All work well if the sampling time is small

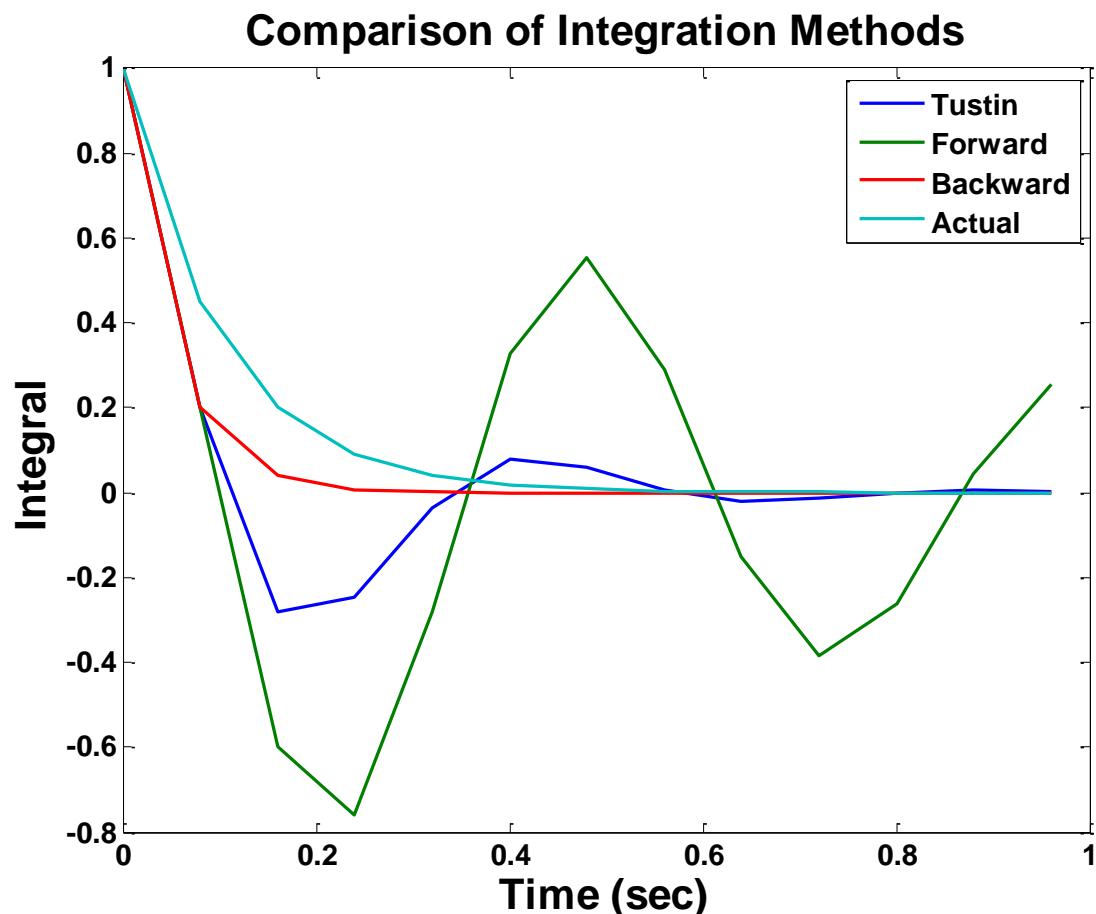
<http://www.mathworks.in/matlabcentral/fileexchange/45537-tustin-backward-or-forward>



What integration algorithm?

As the sampling time increases Tustin and Forward Euler can become unstable

<http://www.mathworks.in/matlabcentral/fileexchange/45537-tustin-backward-or-forward>



Saturation

These blocks are the most important of the blocks in a safety critical control system

They limit the input and output signals of the system. This ensures that the system does not get large values when a sensor fails due to any reason.

Limits can be variable based on flight conditions. A designer would like to prevent large movements very close to the ground but when the aircraft is high above in the skies one has the freedom to move more.

Saturation

if max < min then swap max and min

if input > max

 output = max

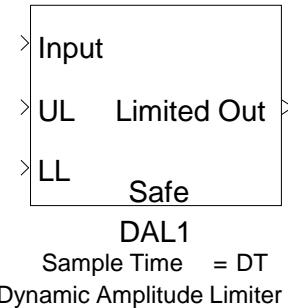
elseif input < min

 output = min

else

 output = input

end



Tips

Saturation blocks can have dynamic inputs as the upper and lower limits. In such cases it is always better to specify what should be the output in case the Lower Limit is greater than the Upper Limit. In the algorithm defined a swap is done on the limits.

Persistence

In safety critical systems it is very important to trap wire cuts, sensor failures etc.

Persistence blocks check for such failures over a finite period of time. If the failure exists for say 2 seconds the output of the block is set to TRUE.

Normally a failure which persists for a long duration causes a latched failure. A latched failure requires a reset to clear

Some of the failures will cause a reset inhibited latch. Such failures in aircraft cannot be cleared when the aircraft is in air. Only after the aircraft lands and the pilot gives an on ground reset is the failure cleared.

Persistence

Inputs: IC, Input, DTOOn , DTOff

Output: Out

If Init True: $y = IC$

During normal operation (i.e. Init = False):

if (input is TRUE and has remained TRUE for DT ON frames)

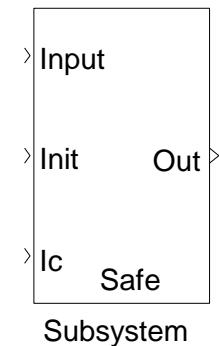
 Out = TRUE

elseif (input is FALSE and has remained FALSE for DT OFF frames)

 Out = FALSE

Else

 Out = Previous frame value of Out



WindowOn/Off

WindowOn/Off is a special type of persistence block

Instead of looking for a continuous failure (on or off state) this block looks for a set of failures in a finite window size

E.g. if a failure occurs 4 times in a window of 20 frames a failure is set.

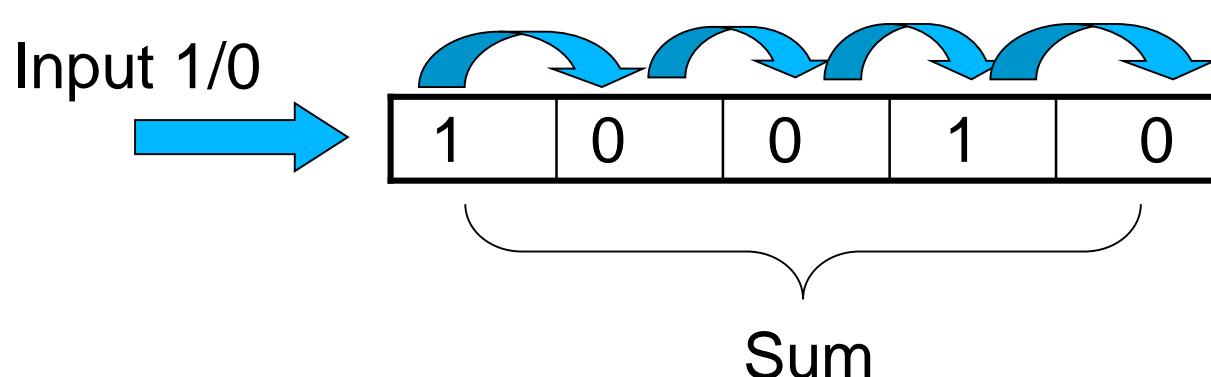
These blocks form a part of the module called redundancy manager. This is a must in all safety critical systems where multiple sensors are continuously monitored and failures and bad sensors are “voted out”

WindowOn

Initially output is False

Open a window (assign a array) of say 20 frames (previous example)

This array represents a moving window



WindowOn

Every frame, the data in each cell is shifted right. The 1st cell has the fresh input data

The sum of all cells in window is computed

If the sum is greater than threshold (4 in previous example) then the output is set to True

Note: 1 indicates On in WindowOn block and a Off in a WindowOff block

Tips

Some algorithms can be complicated and very specific to a program. In these cases it is better if a pseudo code is given as a reference. In some Indian programs we had provided a FORTRAN code with expected results. This helped in implementation and verification.

Latches

These are primarily flip flops used in the digital circuits

In software latches come in basically two flavors – Set Priority and Reset Priority

Latches are used to “latch” a failure in system. It retains its set value and can only be reset by sending a 1 to the reset input

In set priority the set signal is processed first and if it is a ‘1’ the latch is set. In reset priority the reset input is processed first.

In certain cases a count is maintained of the number of resets and then if the count exceeds the limit the reset is disabled making the latch reset inhibited.

Latches

Inputs : S,R

Output =Q

If ($S==1$)

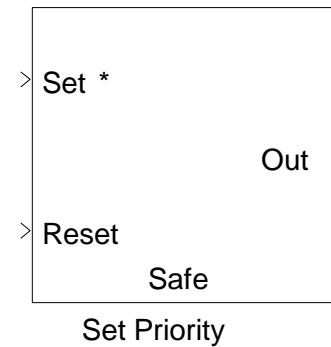
$Q=1$

Else if ($R==1$)

$Q=0$

Else

$Q = \text{prev } Q$



Transient Free Switches

Every control system has a Transient free switch somewhere. It is also called as fader logic.

These are used to fade from one signal to another over time. In aircrafts the lowering of the landing gears cause a change in the system behavior (change in aerodynamics). This causes a change in the control system and the commands to the surface. The smooth transition between the two phases is brought by using the fader logic.

Transient Free Switches

If Event is True output = Sn for 1

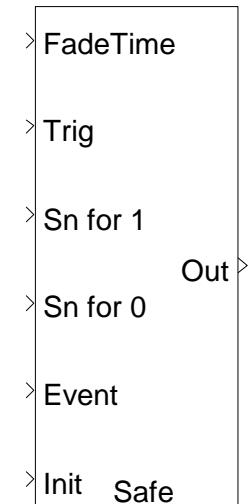
If Event is False output = Sn for 0

If the Event changes state (T-> F or F-> T)

Compute difference between the output and the switched signal

Compute the delta change per frame by dividing this difference by the fade time in frames

Add this delta difference every frame to the output till it reaches the input signal



TFS
Sample Time = DT
Transient Free
Switch

This works well for constants but has problems with continuous signals

Transient Free Switches

If Event is True output = S_n for 1

If Event is False output = S_n for 0

If the Event changes state ($T \rightarrow F$ or $F \rightarrow T$)

Fade a variable A from 1.0 to 0.0 over the fade time

If the fade is from True to False. Multiply the True Signal with A and False signal by $(1-A)$.

This causes the True signal to fade out and the False signal to fade in

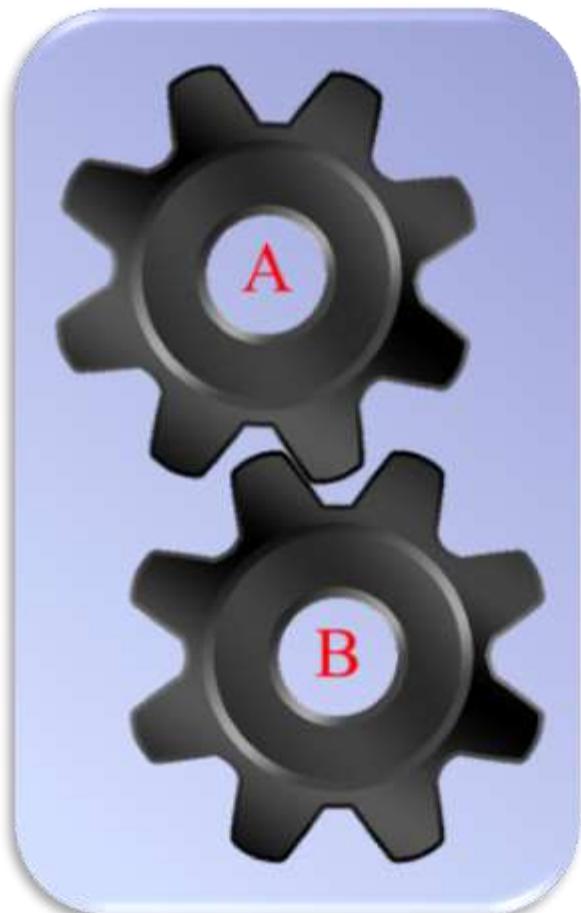
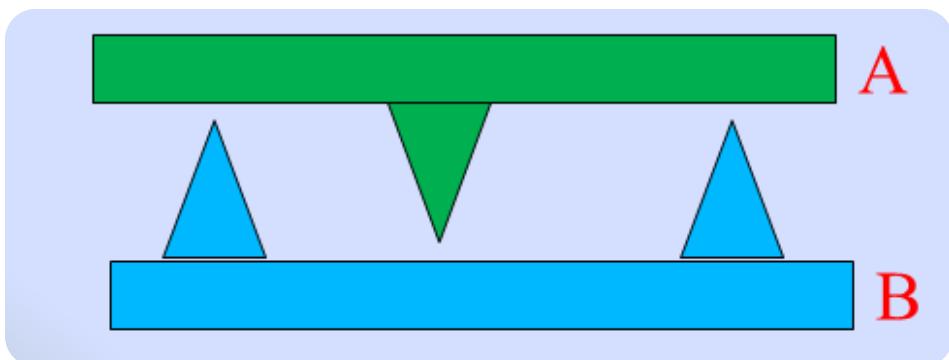
Add these two signals to get the output

This is not a linear fade logic

This is a modified logic used in an Indian program

Backlash

This block represents a gear like operation. Two equal gears rotating together behave like this block. When one of the gear's teeth is between the other two there is no output. The other gear will be stationary. Only when the teeth touches the other and continues further there is an equal output.



Backlash

These blocks are used in control system when we do not want the output to respond to small changes in input.

Disengaged - In this mode, the input does not drive the output and the output remains constant. Input is within the deadband.

Engaged in a positive direction - In this mode, the input is increasing (has a positive slope) and the output is equal to the input minus half the deadband width.

Engaged in a negative direction - In this mode, the input is decreasing (has a negative slope) and the output is equal to the input plus half the deadband width.

Backlash

During Initialization $x_u = I_{np} + band * 0.5$ and $y_0 = I_{np}$

if ($x > x_u$) {input increasing and greater than deadband}

$dx = I_{np} - x_u$ and $x_u = I_{np}$

else

$x_l = x_u - band$ {set the lower band}

if ($x < x_l$) {input is decreasing and beyond DB}

$dx = I_{np} - x_l$, $x_u = x_u + dx$;

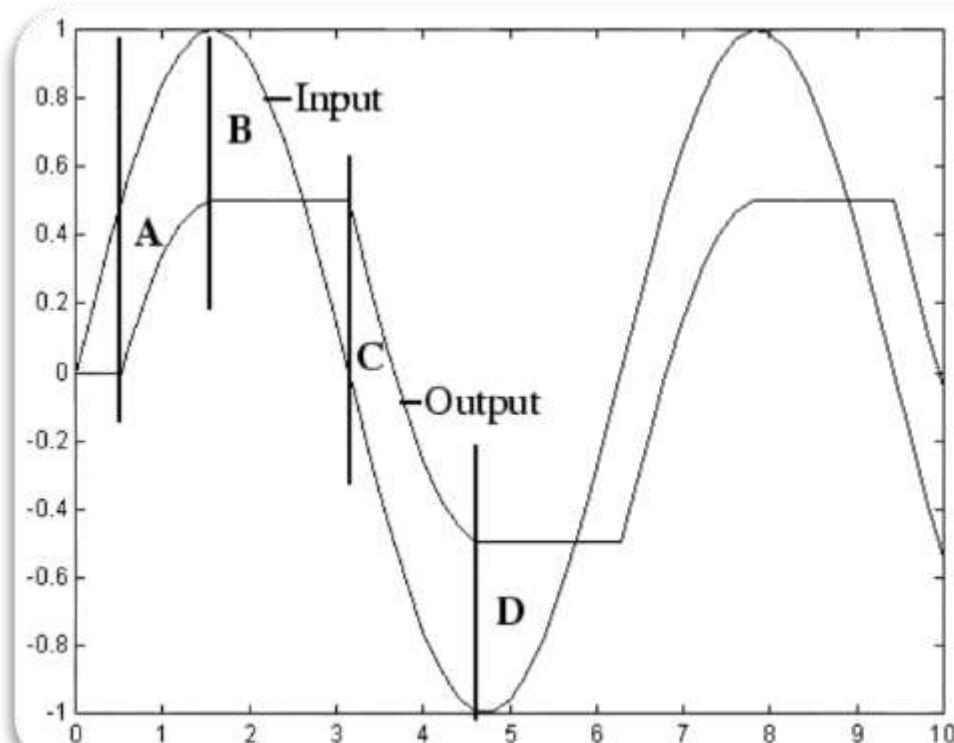
else

$dx = 0.0$ {input is within the dead band}

$y = y_0 + dx$, $y_0 = y$

Backlash

The backlash from Mathworks Simulink help.



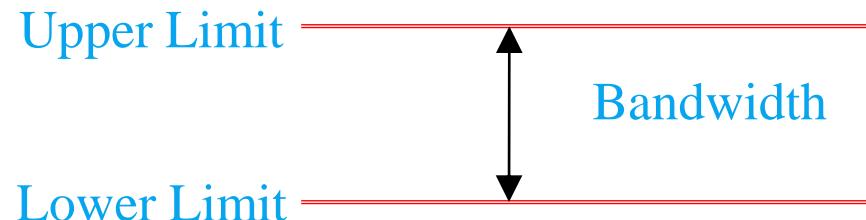
- A Input engages in positive direction. Change in input causes equal change in output.
- B Input disengages. Change in input does not affect output.
- C Input engages in negative direction. Change in input causes equal change in output.
- D Input disengages. Change in input does not affect output.

Logical Hysteresis

This block is similar to the backlash but gives a logical True/False output

These blocks are used to put a finite band on the input signal (normally noisy) to trigger a True if beyond a upper limit.

Once set to True the output is set to False only if the input falls below a lower limit at a distance BW from the upper limit.



Logical Hysteresis

During Initialization Output = False

If Output is True and Input < LL Then Output is False

If Output is False and Input > UL Then Output is True

Else Output does not change

The lower limit LL and Upper Limit UL are defined based on the Bandwidth and the mid point.

Up/Down Counters

Up and Down counters are used monitor failures in signals. They are similar to persistence on/off blocks but behave a little differently.

When the input is TRUE (there is a failure) then the internal counter is incremented up with a count of 3 (say) i.e. $\text{count} = \text{count} + 3$

When the input is FALSE (there if no failure) then the internal counter is decremented down with a count of 1 i.e. $\text{count} = \text{count} - 1$

If the counter has reached a max value then the output is set to TRUE and if the count reaches 0 it is set to FALSE.

Up/Down Counters

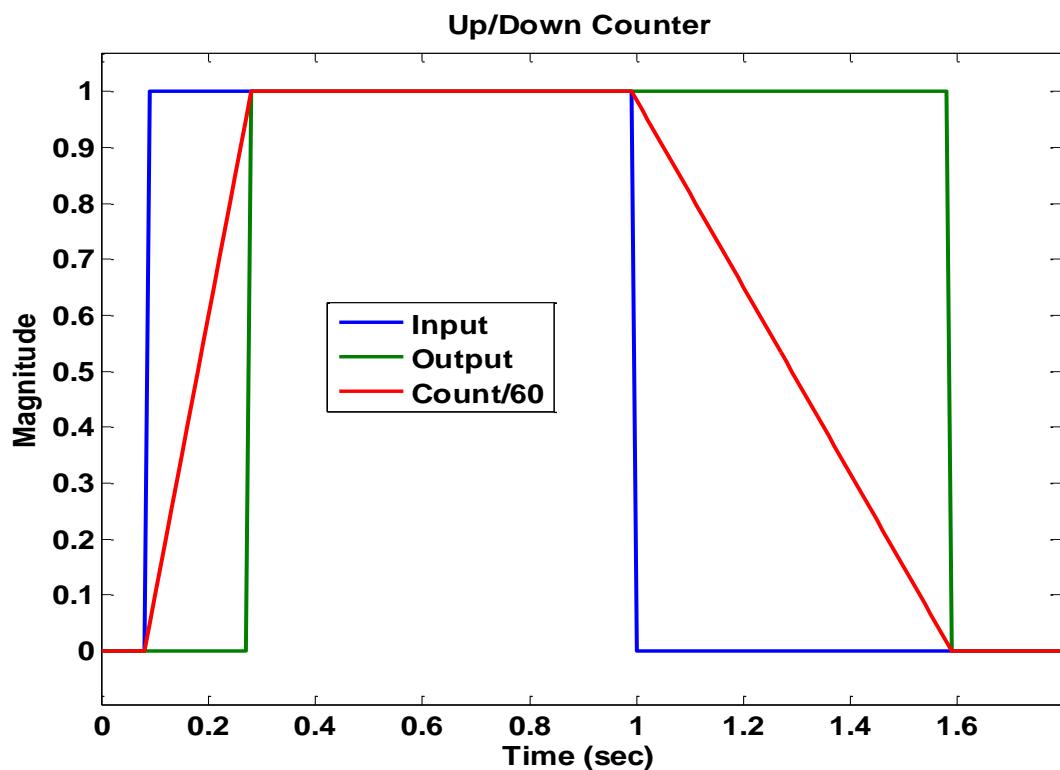
During Initialization Output = False and the internal count is 0

Variations of these counter have a reset inhibit capability. If the output has become TRUE for say 3 counts then a reset does not clear the count and the failure is permanently latched.

Such counters are very widely used in the redundancy management circuits and voter logic in safety critical system.

Up/Down Counters

```
if inp == 1  
    count=count+3;  
else  
    count = count - 1;  
end  
if count >= 60  
    out = 1;  
    count = 60;  
elseif count <= 0  
    count = 0;  
    out=0;  
end
```



Maximum count = 60, Upcount = 3 and Downcount = 1

Are these all?

There are several other blocks but they can be clubbed under one of the type of blocks defined here

For example all filters – structural, washout, complementary filters, digital differentiators, compensators are represented by first or second order filters.

All integrators differ only in the integration algorithm

Voter logic can be represented by the persistence and window type logic

Discontinuities like dead band can be implemented as 1-D lookup tables

Multiport switches are used to select a specific input based on the number. If switch input is 1 select the first input and send it to output. If 2 then the second input I sent to the output and so on.

Are these all?

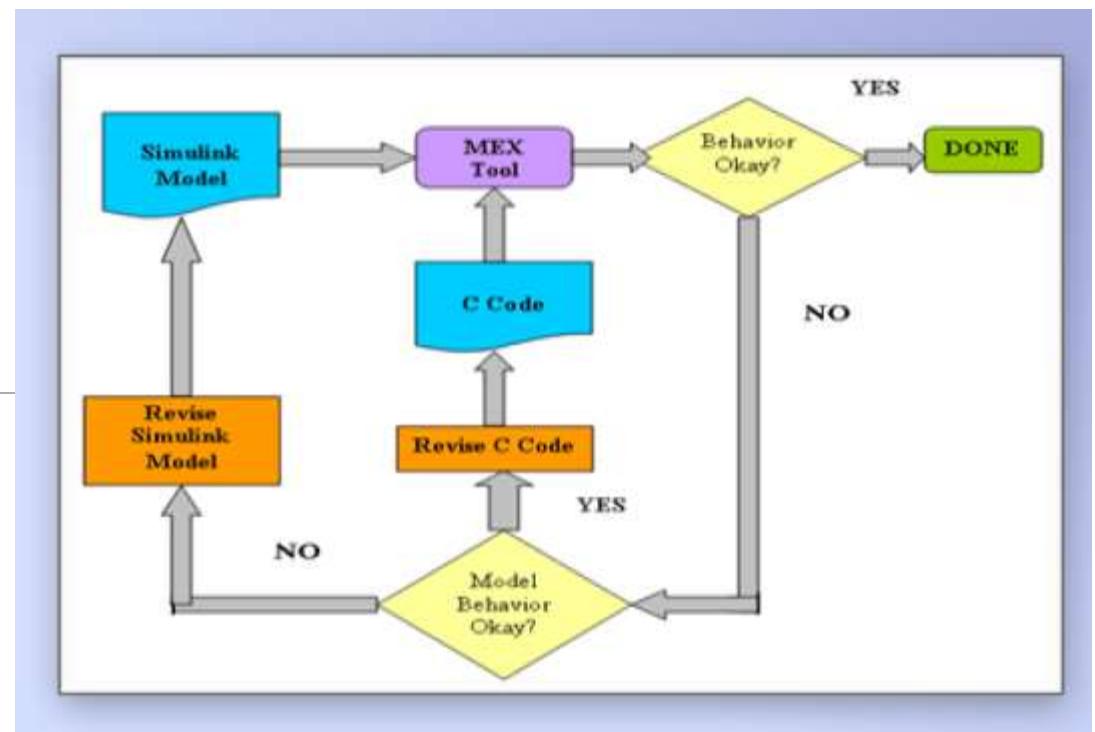
Other blocks that are used are arithmetic operations like adder, subtraction, multiplier and trigonometric blocks. These are fundamental building blocks and normally have an equivalent C or Ada function

Logical blocks like AND, OR, NOR etc are represented as logical statements in C or Ada language. Comparators are used to compare signals against specific values to take a decision.

Switches and multiple selections are done using IF THEN ELSE constructs in the language.

These are enough to implement the most complicated Fly-by-Wire algorithms in Aerospace

Model Based Testing



Model Based Test

An executable requirement of the control system is available as a model

The C/Ada code for this requirement has been developed and runs on a target platform

The idea of model based tests in a nutshell is to generate a set of test cases which will generate a set of input signals time histories. These inputs are injected into the Model and simulated to get the outputs.

The same input signals are injected into the corresponding compiled code inputs and the expected outputs tapped out.

Model Based Test

If both Model and Code outputs match then we infer that the code is as per the requirements.

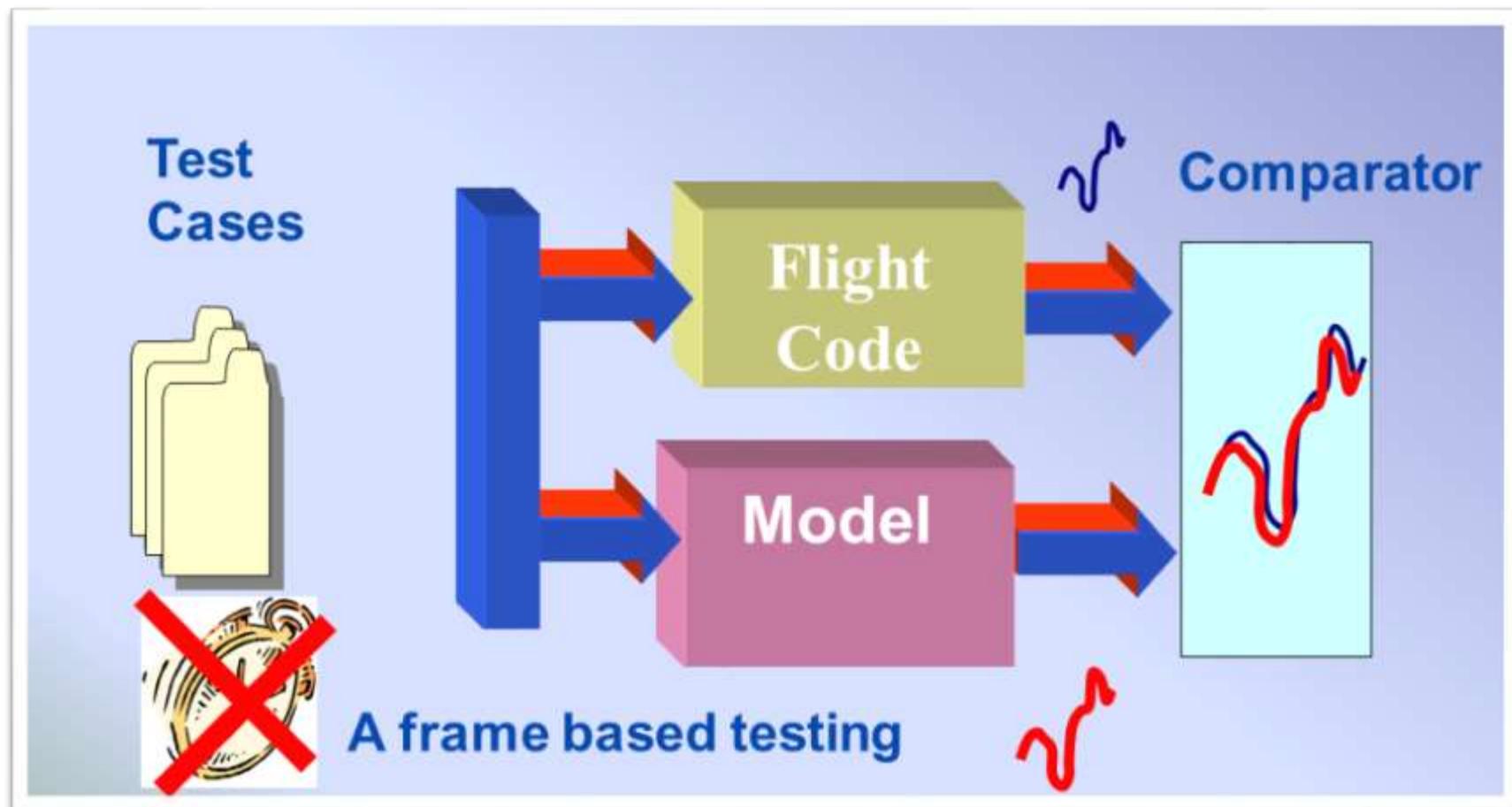
The assumption for a complete test is that we have generated the test cases which cover the Model functionality 100%

The same set of test cases give 100% code coverage on the target on an instrumented code build

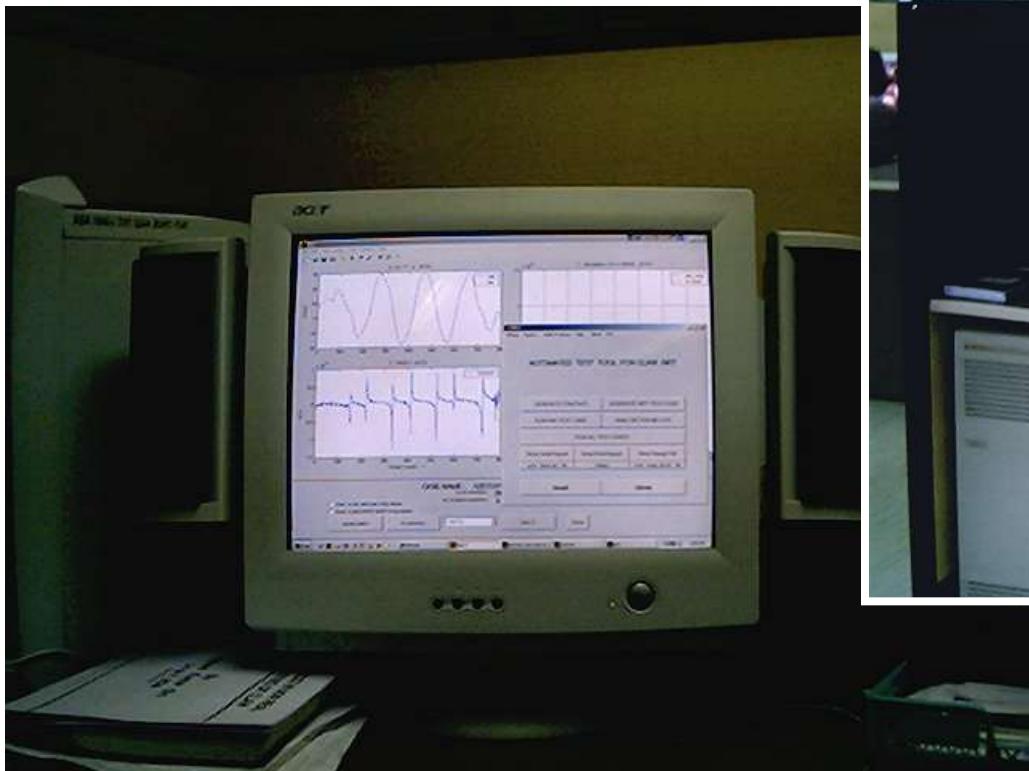
The instrumented code output and non instrumented code output match “very well” with the Model output.

“Very well” is defined beforehand based on the target data, the input output quantization, etc

Schematic



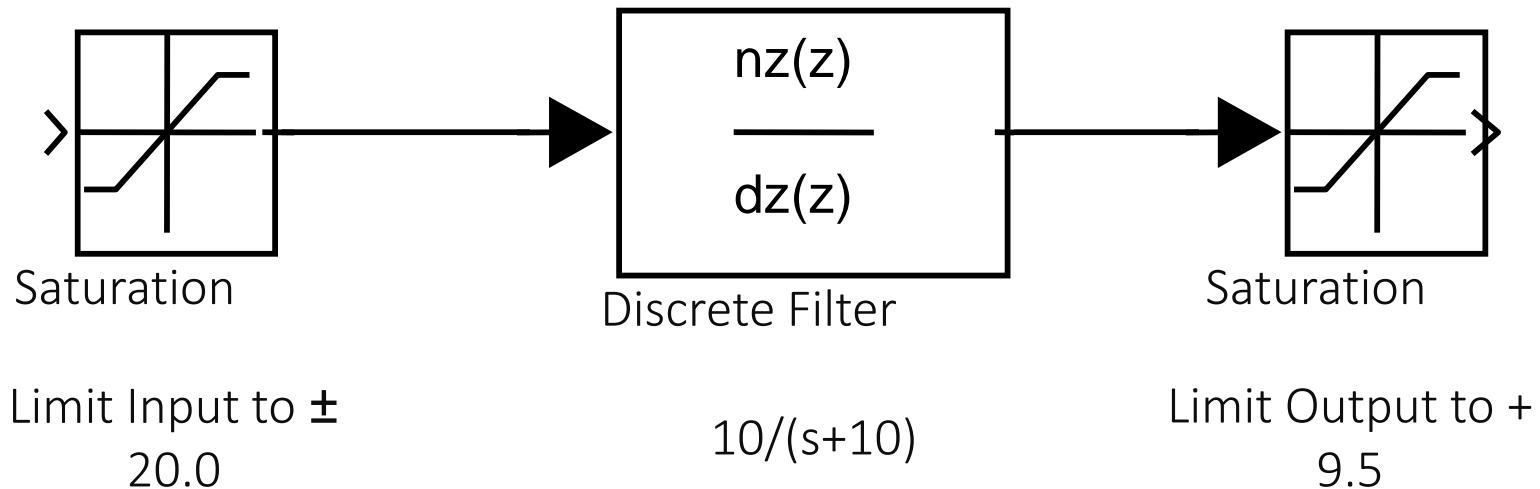
Setup



Year 2000 – That is DEC VAX system in the background

Testing Example

- A small example is shown here. This was a missile implementation which failed. The input is limited between +20 and -20, filtered through a digital filter and the output limited on the positive side.



Limit Input to ± 20.0

$$\frac{nz(z)}{dz(z)}$$

Discrete Filter

$$10/(s+10)$$

Limit Output to + 9.5

Static Test

- A set of constants are used to test the code implementation against the model

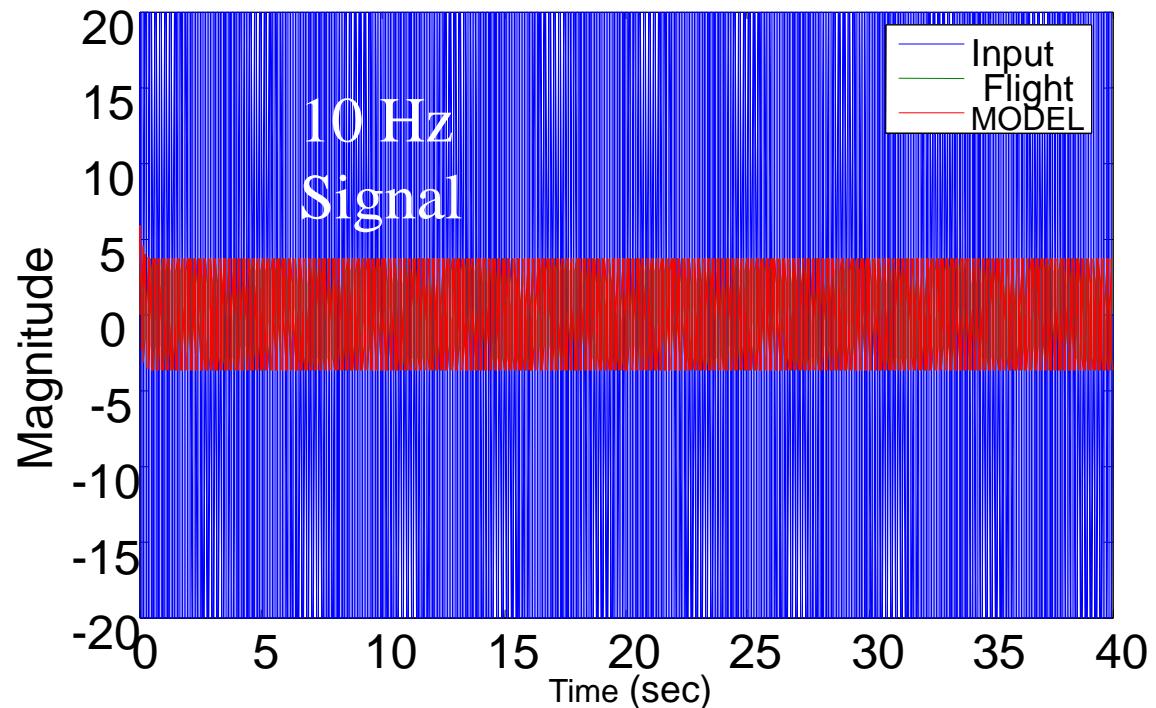
The Flight code
and the Model
outputs match
exactly. Can we
pass a safety
critical system
with these tests?

Input	Model	Flight
0.0	0.0	0.0
-3.0	-3.0	-3.0
-25.0	-20.0	-20.0
3.0	3.0	3.0
25.0	9.5	9.5

Dynamic Test

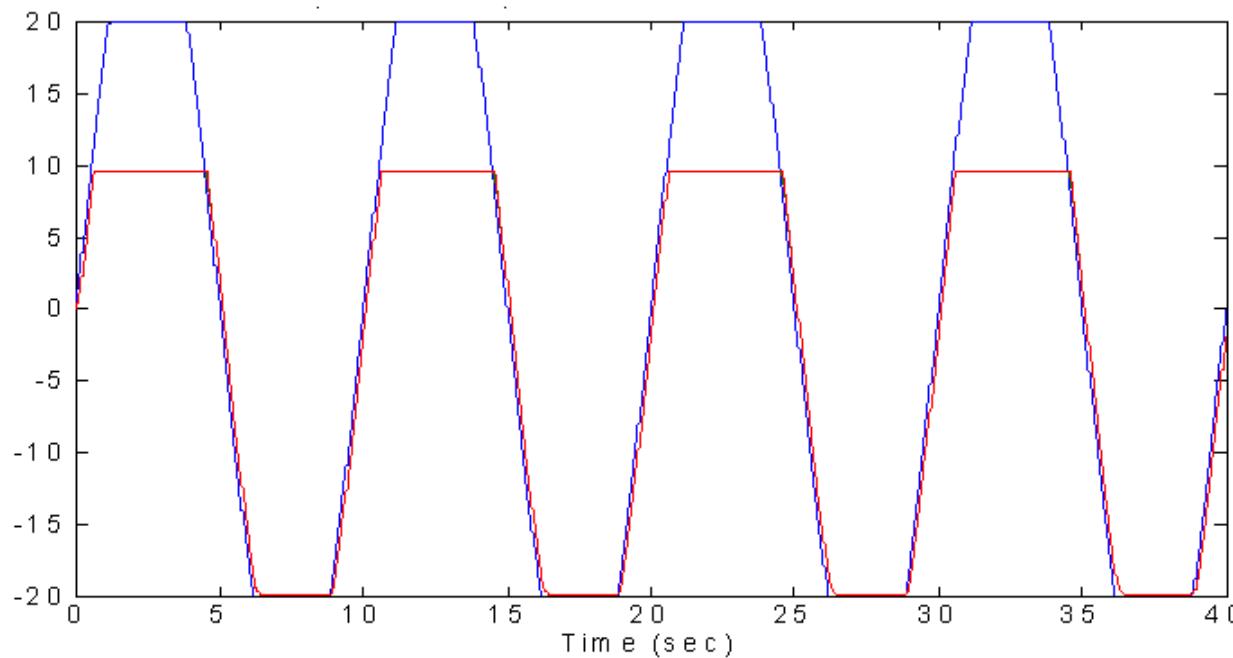
- A 10 Hz signal was injected into the system. The Flight code and the Model match very well.

The Flight code and the Model outputs match exactly. Can we pass a safety critical system with these tests?



Dynamic Test

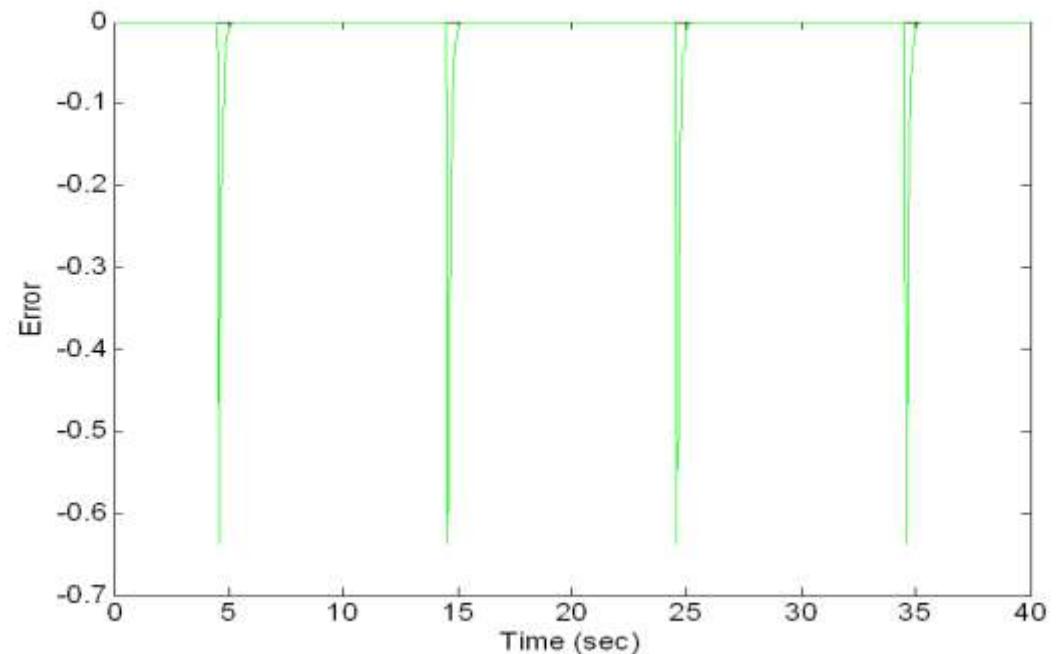
- A 0.1 Hz signal was injected into the system.



Dynamic Test

- There is an error between the Flight code and the Model. This is a significant error.

A high frequency test has not excited all the blocks completely as the filter is reducing the higher frequency signal. The output limiter is not exercised. Taking credit of the static test does not help.



Dynamic Test

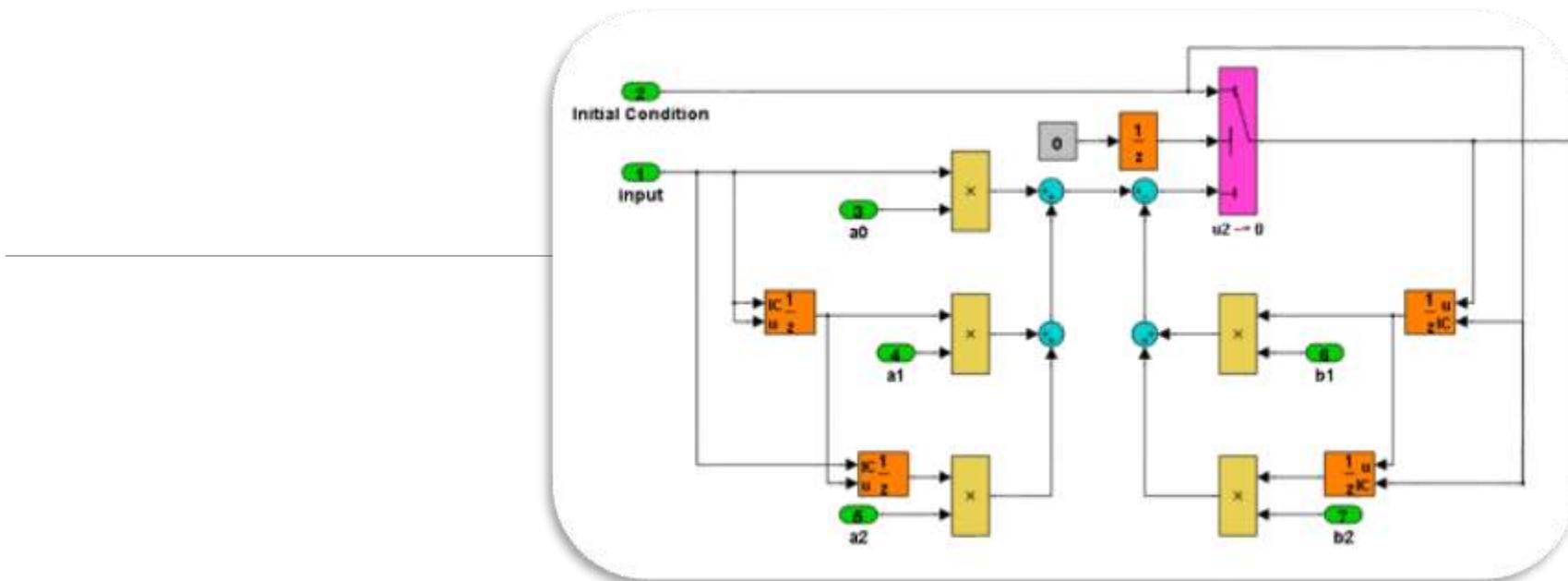
- $\text{nz} = [5.882\text{e-}2 \ 5.882\text{e-}2]; \text{dz} = [1.0 \ -8.823\text{e-}1];$
- Initialisation
 - $O = \text{inp}, \ p_{\text{inp}} = \text{inp}$
- Loop
 - $O = \text{nz}(1) * \text{inp} + \text{nz}(2) * p_{\text{inp}} - \text{dz}(2) * O$
 - if $O > 9.5$
 - $O = 9.5;$
 - end if
- End Loop

The state is limited and used in the computation. This is because the code uses the same variable name “O” for the filter output and the limiter output.

Tips

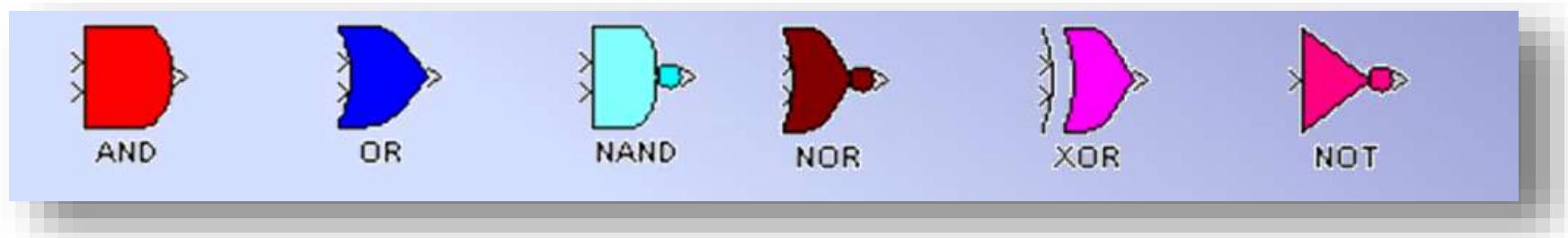
If the C code is generated using the Model it not very meaningful to generate the expected results from the model and compare with the C code output. We require higher level requirements perhaps text based that have to satisfied by the test case design.

Control System Block Tests



Logical Blocks

- IEEE Standard Graphic Symbols for Logic Functions

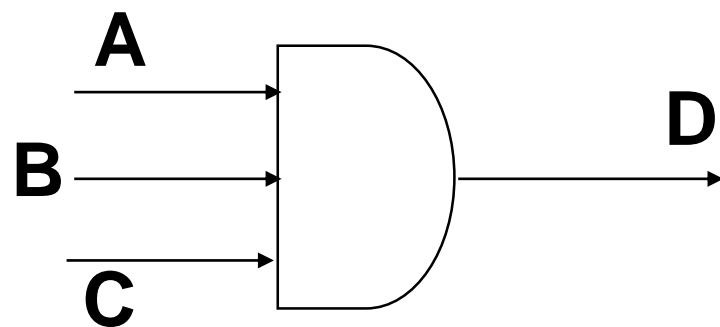


- AND = TRUE if all inputs are TRUE
- OR = TRUE if at least one input is TRUE
- NAND = TRUE if at least one input is FALSE
- NOR = TRUE when no inputs are TRUE
- XOR = TRUE if an odd number of inputs are TRUE
- NOT = TRUE if the input is FALSE

Logical Blocks

- For a Safety Critical Application All Logical Blocks have to be tested to ensure Modified Condition / Decision Coverage (MC/DC)
- The effect of the input signals on the block has to be shown at a output which corresponds to a observable variable in the code (a global variable)
- The logical blocks are normally connected to a switch and both TRUE and FALSE operations of the switch have to demonstrated on the output.

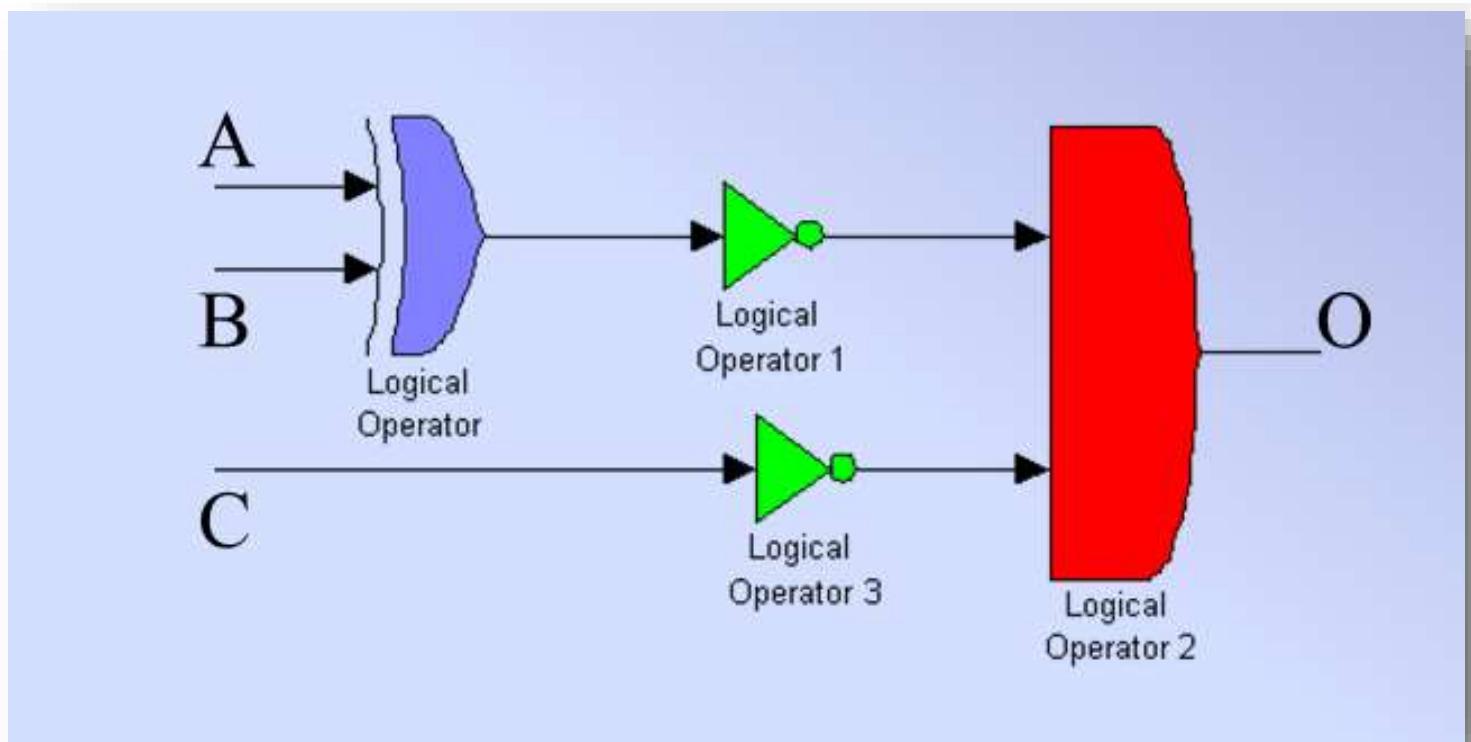
MC/DC Example



A	B	C	D	
F	F	F	F	
F	F	T	F	
F	T	F	F	
F	T	T	F	1
T	F	F	F	
T	F	T	F	2
T	T	F	F	3
T	T	T	T	4

Exercise

- Define the MC/DC Test cases for this Combination Logic



Answer

A	B	C	A xor B	NOT(A xor B)	C'	O	
0	0	0	0	1	1	1	
0	0	1	0	1	0	0	
0	1	0	1	0	1	0	2
0	1	1	1	0	0	0	
1	0	0	1	0	1	0	3
1	0	1	1	0	0	0	
1	1	0	0	1	1	1	1
1	1	1	0	1	0	0	4

Beware of MC/DC

A	B	AND	NOT(XOR)
0	0	0	1
0	1	0	0
1	0	0	0
1	1	1	1

Testing Logic

- I find it easier to understand MC/DC by imagining a light bulb at the observable output and all the inputs as switches. I have to toggle each input to ON/OFF to light the bulb and put it off by keep all other inputs constants in an OFF or ON state
- MC/DC has $1 + \text{number of inputs}$ as an optimal set of test cases.
- A small program can be written to generate a set of test cases which generate $2 \times \text{number of inputs}$. The effect of toggle of a switch (input) is shown between two consecutive tests. This helps a lot!

Testing Logic

- The specific output should be observable. This is very important. Most times the developer uses a set of local temporary variables to define intermediate logic outputs. This is then used in an if-then-else to set a global variable.
- This global variable is the observable output.
- MC/DC has to be shown on this global variable. This is a daunting task for the tester and the cause for delays in the verification and validation process
- Any automation in this will help!

MC/DC Test Case Generator

Say there are N Inputs

Generate a 0,1 sequence randomly

Check the output for this test case

Toggle the first input and observe the output

If the output has toggled when compared to the previous test you have two test cases which check the independent effect for input 1

If the output has not toggled then generate a new 0,1 sequence.

It works most of the times ! Script available on MathWorks site

<http://in.mathworks.com/matlabcentral/fileexchange/37953-mc-dc-test-case-generator>

Tips

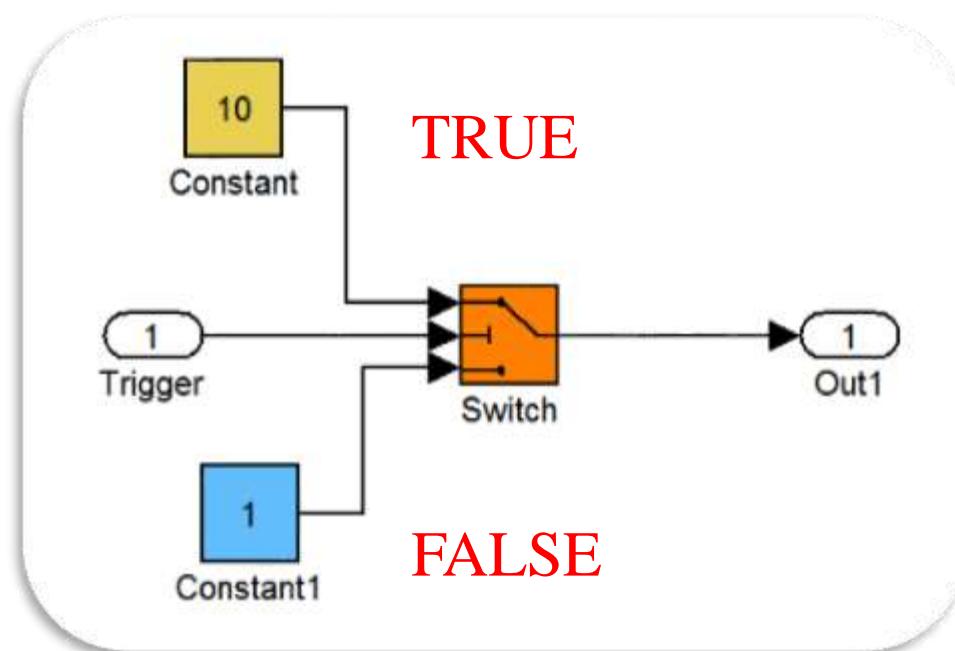
If you are evaluating an automated test tool – ENSURE that the tool generates a test case for the observable output. The normal argument is if you generate an autocode it will work very well.

Tips

A little bit of ingenuity and skillful programming will enable you to come up with simple test tool tailor-made for your specific test requirement. It is worthwhile spending some time on this at the start of the project. It will go along way !

Switch Blocks

- A Switch Block mimics an IF ELSE statement in code
- The Trigger or Event input in the centre causes the output equal to one of the inputs Constant, Constant1



Testing Switches

- In a model based approach it is usually seen that the path till the switch inputs is normally executed. This is not so in the case of C Code. The programmer will normally put a set of instructions inside the if-then-else logic.
- As a result intermediate states may have different values.
- **Solution: Use an If-Then-Else block OR code like the model!**
- Take care while selecting inputs. It is possible that both the inputs to the switch may be equal due to computation in the path above. This will make the test confirmation difficult.

Tips

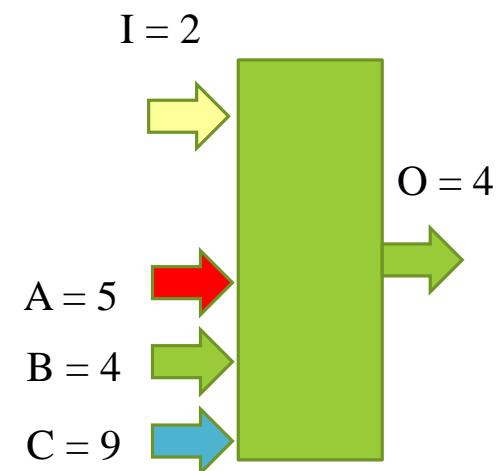
It is a good practice to have the trigger to the switch in the model as a Boolean. Compute this Boolean beforehand in the model.

Testing Multiport Switches

Ensure that you test for the range and beyond (if possible) of the switch input. This will be the integer value.

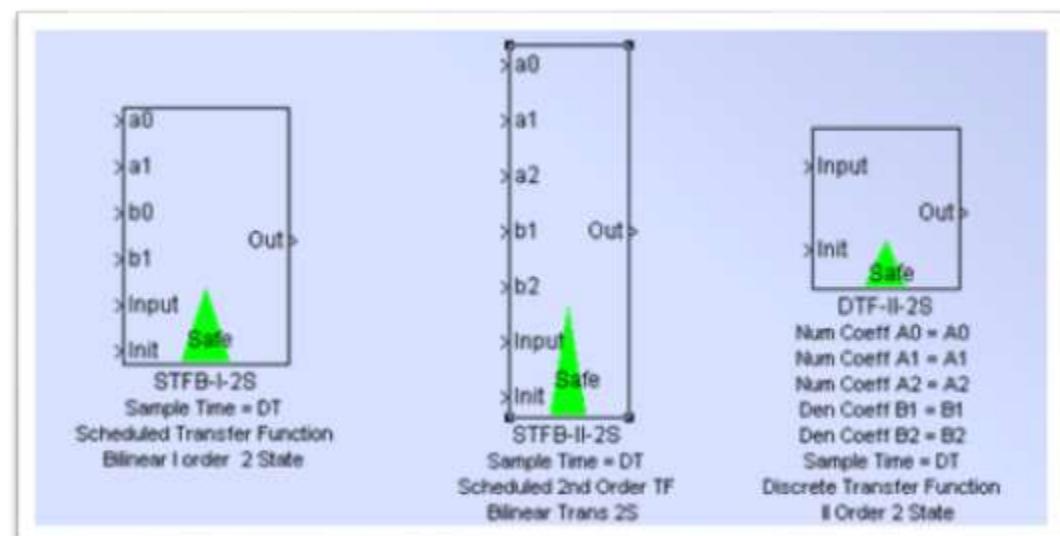
Ensure that all other inputs have different values so that it is possible to ensure correct output just by visualizing the value. Example - if it is a 3 input multiport switch ensure selection input I runs through 1,2,3 (4 and 0 if possible). While this is happening ensure inputs A, B and C have different values.

Same values make your results ambiguous.



Filters

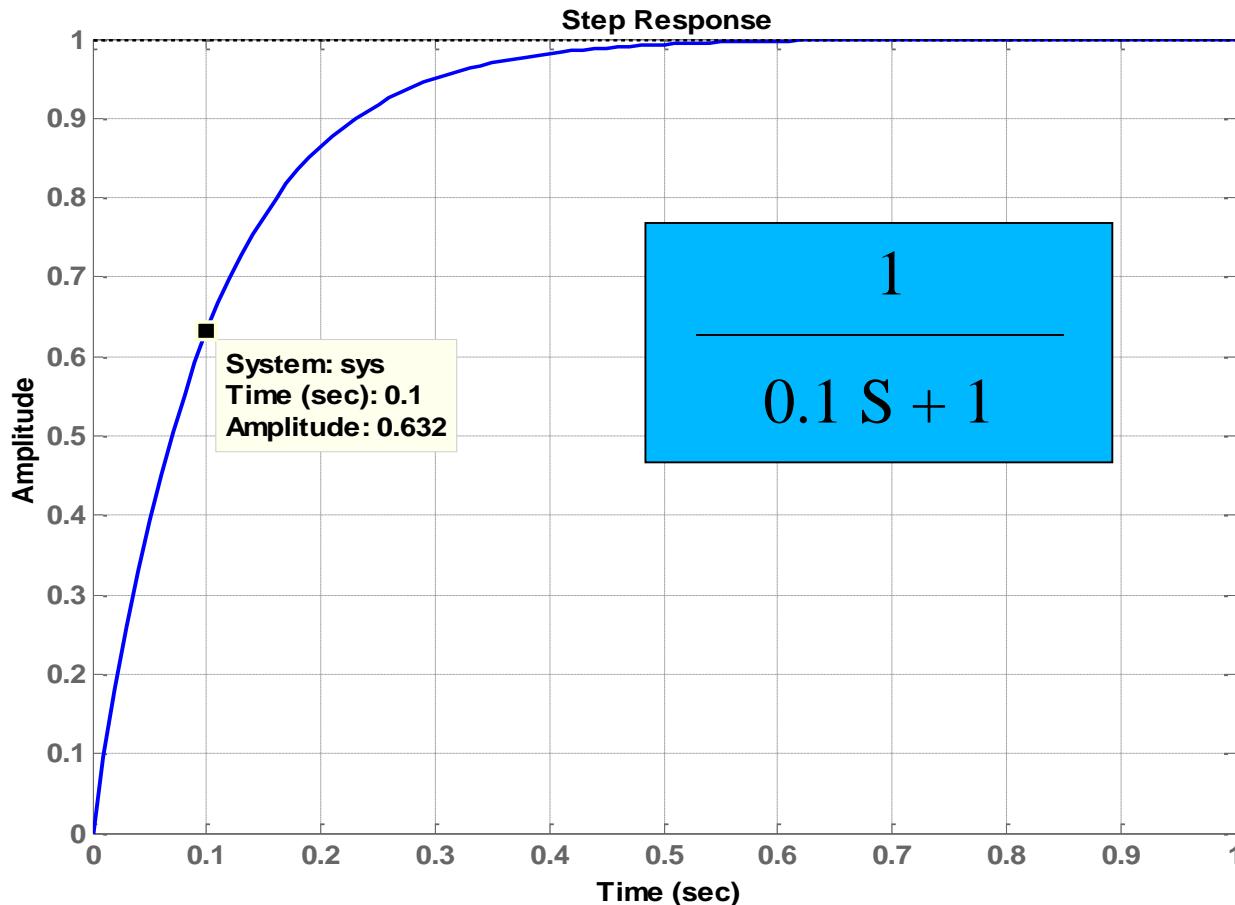
- Filters are dynamic elements of a control system. They have a state and the output changes with time. They are very important to a stability of a system.
- The correct implementation in Code has to be ascertained and demonstrated for Certification.
- Type of filters used in the control system are typically
 - First order
 - Second order
 - Notch Filters
 - Washout



First Order Filters

- First order filters are the simplest of the filters used to cut off noise. In model based testing they can be easily tested by giving a step change at the input of the filter
- The first order lag filters are characterized by a time constant and for a unit step input the value of the output is approximately 0.632 at a time equal to the time constant. This can be used to prove the correctness of the response!
- Normally the filter output and the filter states are initialized to the input. This ensures that the filter output is constant for a constant input.
- **Test initialisation with two separate inputs! Very important!**

First Order Filter Response



It is a good practice to test the filter for settling time. This can be done by giving a step and observing the output for 6 times the time constant. This is not always possible at a high level test but definitely worth a try!

Second Order Filters

- A standard Second Order Filter defined in the S domain will have a constant in the numerator and a second order term in the denominator
- The Second order filter is characterized by Rise Time, Peak Amplitude, Time at Peak Amplitude and the Settling Time to 2% of its Steady State value

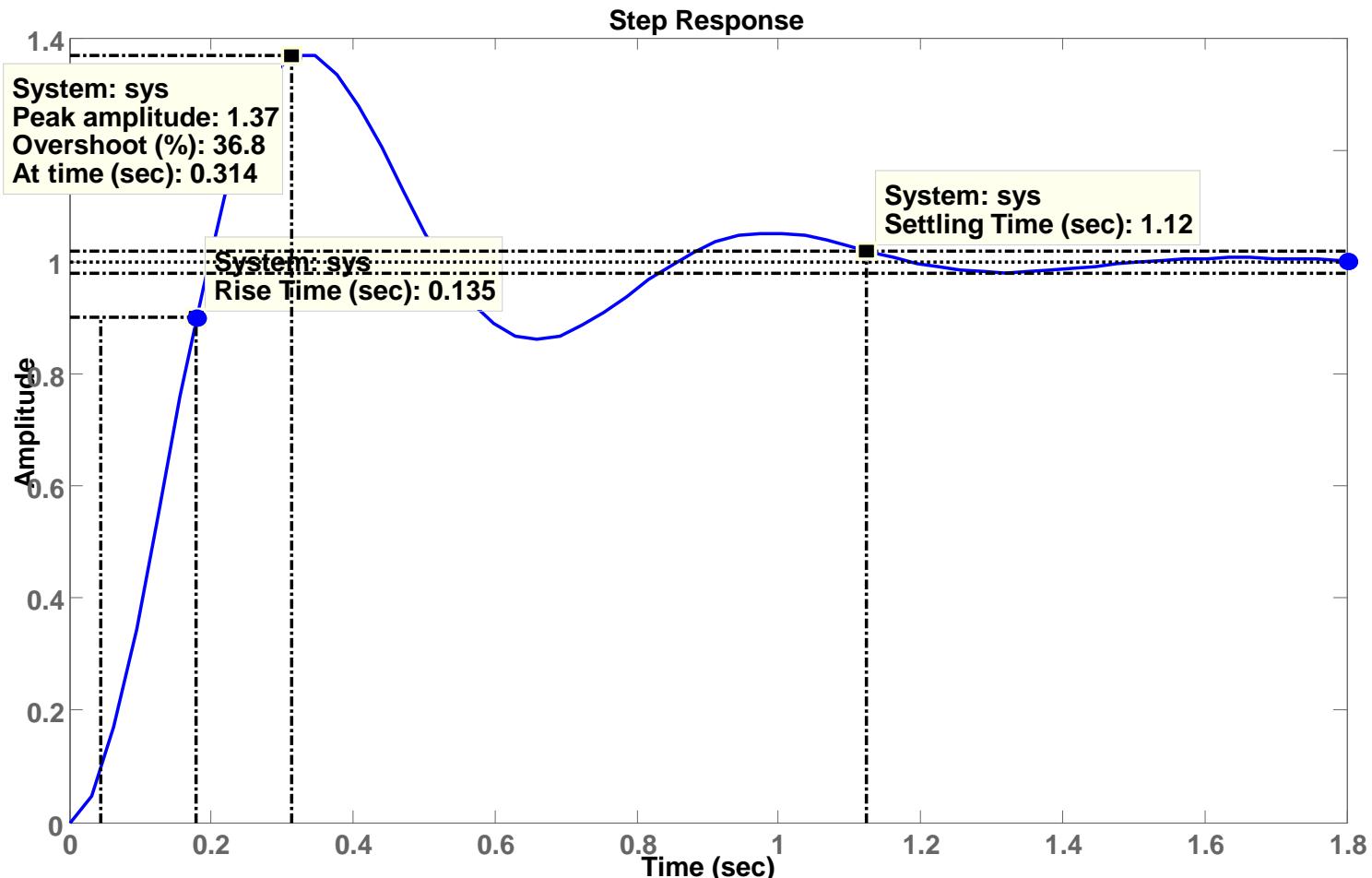
$$\frac{Y}{X} = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n + \omega_n^2}$$

$$Tr = \frac{1}{\omega_n \sqrt{1 - \zeta^2}} \left(\pi - \tan^{-1} \left(\frac{\sqrt{1 - \zeta^2}}{\zeta} \right) \right)$$

$$Tp = \frac{\pi}{\omega_n \sqrt{1 - \zeta^2}}$$

$$Ts = \frac{3.9}{\zeta\omega_n}$$

Second Order Filter Response



Testing 2nd Order Filters

- They are tested the same way as the first order filters with a step response
- The various parameters that characterize the filter are confirmed. It is a good practice here to verify settling.
- Second order filters are sensitive to initialization and the first 3-4 frame values are very important. They can tell if the filter has been implemented correctly
- Normally states are all initialized to the input signal. This in turn ensures that the filter output is constant for a constant initial input. **Test for at least TWO input values!**

Tips

Filters should have an initialization in the model and in code. Filter initialization have to tested very thoroughly. Remember, I repeat, we always find error here.

Tips

Sometimes filters are implemented with an integrator as in Analog days. This has to be tested as a filter rather than an integrator and associated circuits. Integrator initialization plays an important role here.

Testing Notch Filters

- They are special 2nd Order Filters characterized by a different value of numerator and denominator damping ratio
- They have to be prewarped for ensuring correct frequency domain characteristics
- A sine sweep signal will test the filter adequately. Ensure as large an input as possible

$$\frac{Y}{X} = \frac{s^2 + 2\zeta_1\omega_n + \omega_n^2}{s^2 + 2\zeta_2\omega_n + \omega_n^2}$$

Remember how the large frequency test did not test the system completely. A good thumb rule is to have a frequency component close to the notch frequency. A low frequency and a high frequency component.

Tips

I see testing filters as giving inputs with sufficient energy to excite the block. Like shaking a box of marbles. Don't shake the box too much to break the marbles but just right to know how many marbles are there. It is tricky. Life is Great but may not be simple!

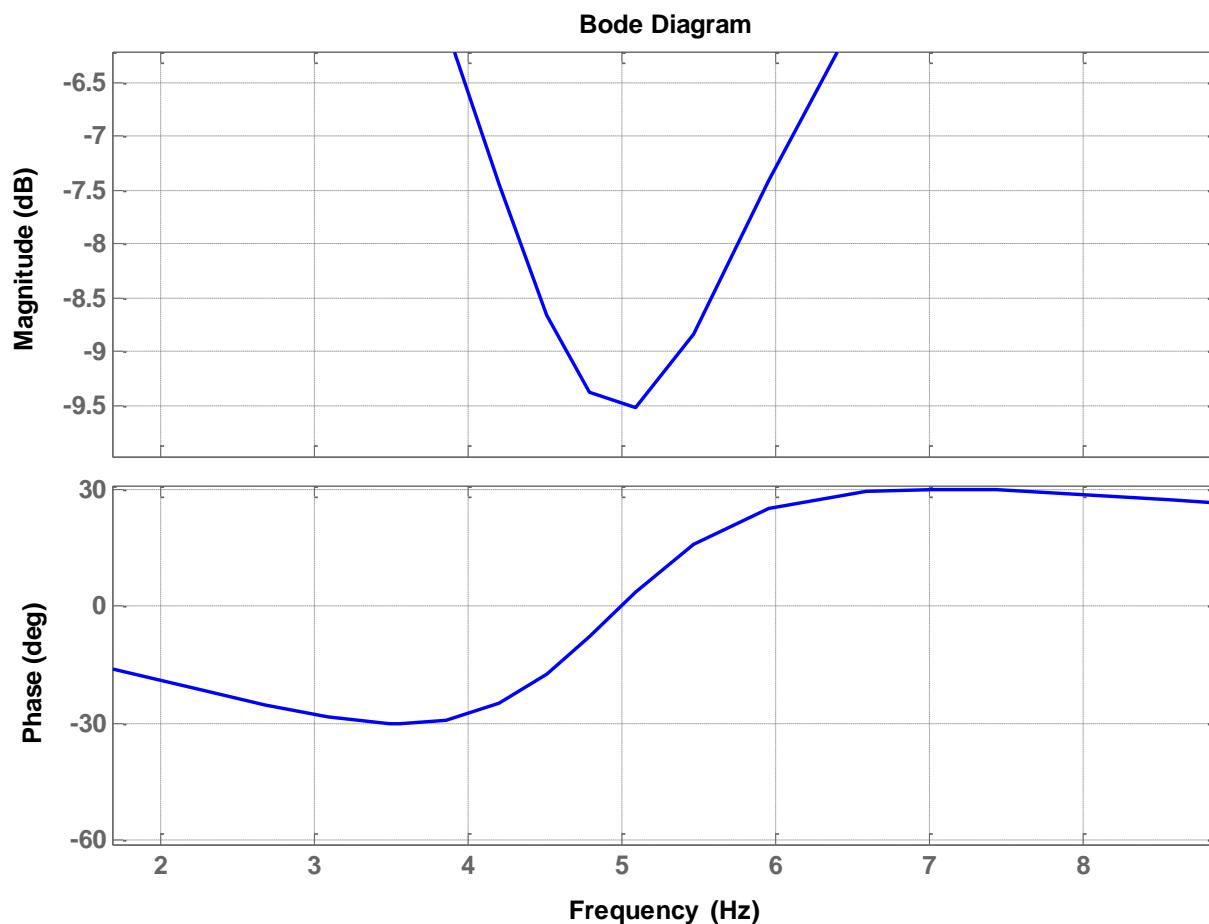
Testing Washout Filters

- Washout filters are differentiating filters
 - The first frame output is normally initialized to 0.0. Why?
 - A static input is not sufficient to test this block. Moreover if there are more blocks downstream of a Washout filter, constant input static tests DO NOT test any of the blocks downstream.
-
-
-
- **The output of a washout filter for a constant input is always ZERO! Be very aware of this fact.**

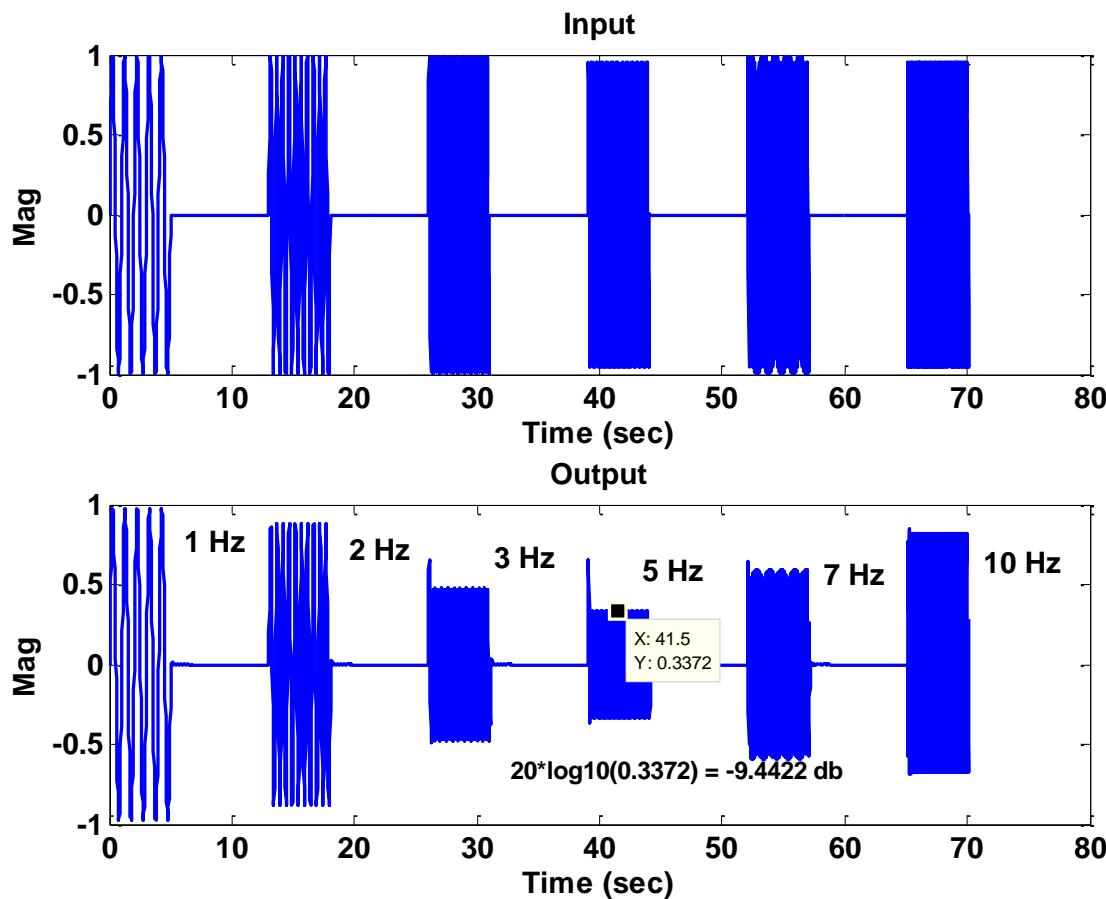
Scheduled Filters

- These are first or second order filters which have time varying coefficients
- It is simpler to specify the filter coefficients in the S Domain for these filters. A first order filter will have the time constant varying with time
- First the filter is tested with constant coefficients. This checks the algorithm
- Then the filter is checked with time varying coefficients
- Sine Sweep signals and sinusoidal waveforms can be used to verify the filter performance

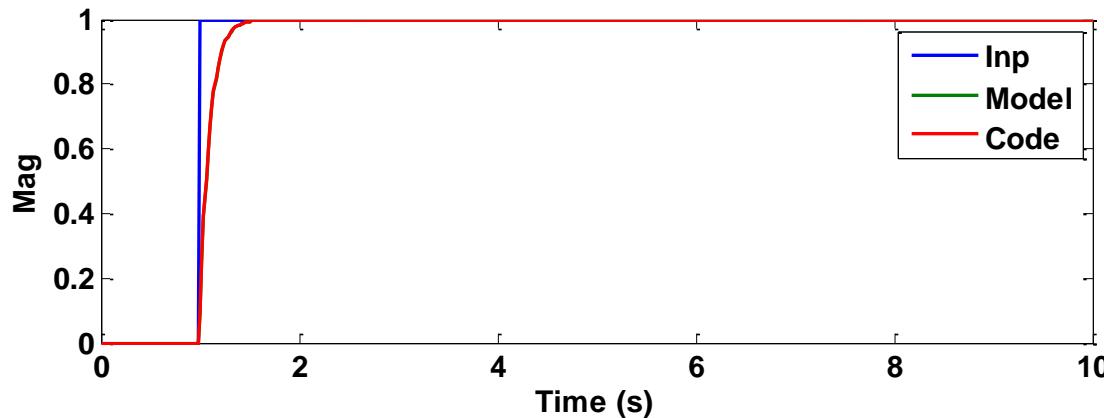
Notch Filter 5 Hz



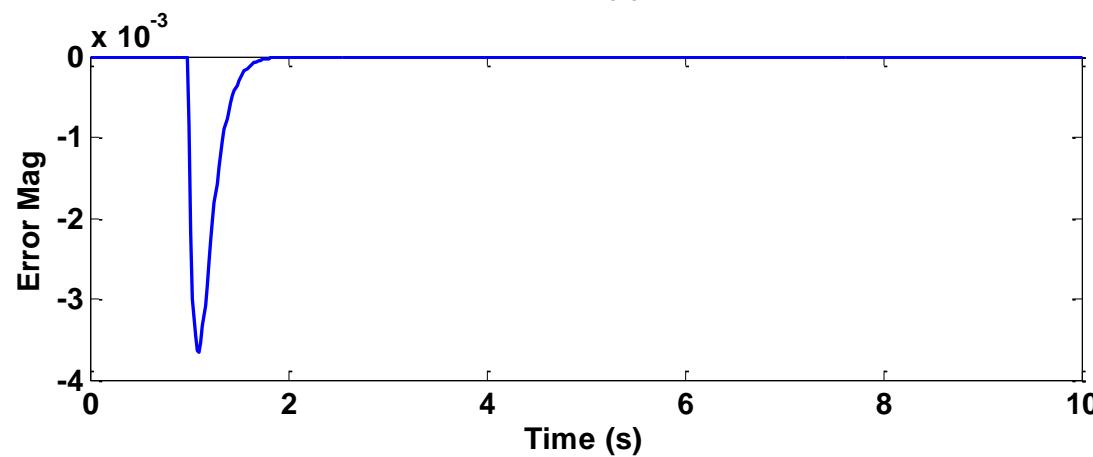
Notch Filter 5 Hz – Test Waveform



First Order Filter with Error



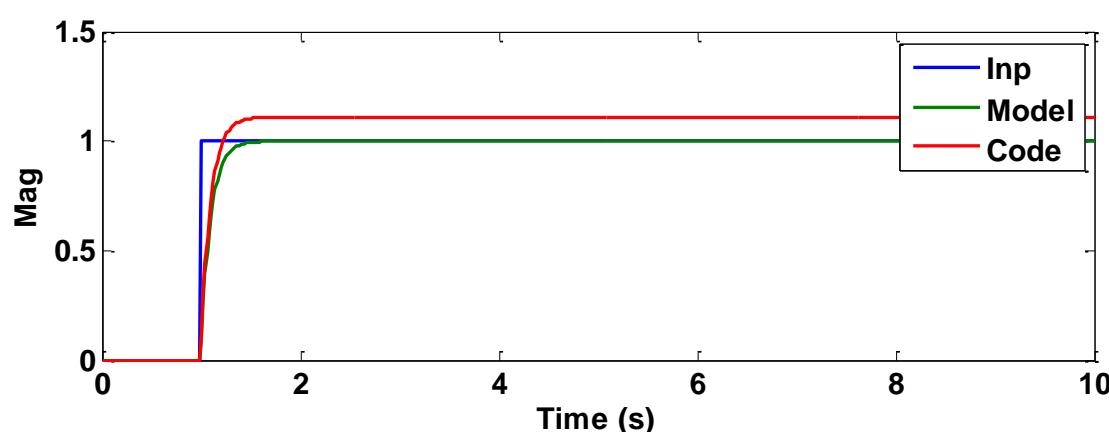
Model
 $10/(s+10)$



Code
 $10.1/(s+10.1)$

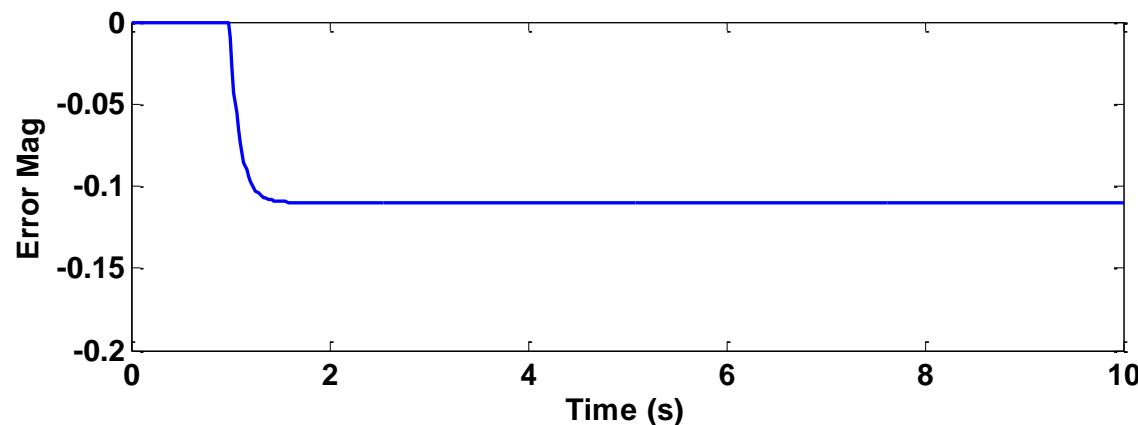
A time constant error is seen in the transient behavior only. Observe the magnitude of error 10^{-3} .

First Order Filter with Error



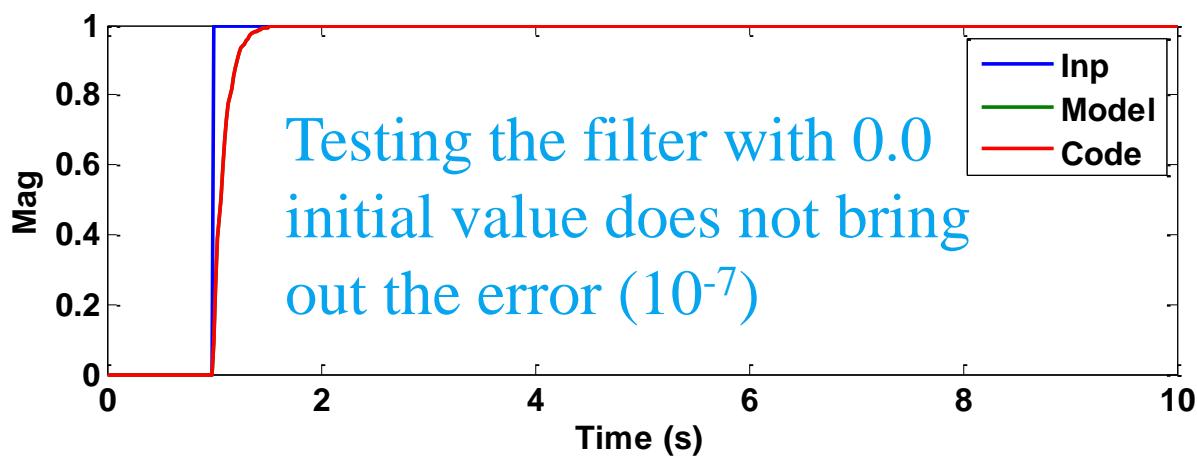
Model
 $10/(s+10)$

Code
 $11.1/(s+10)$



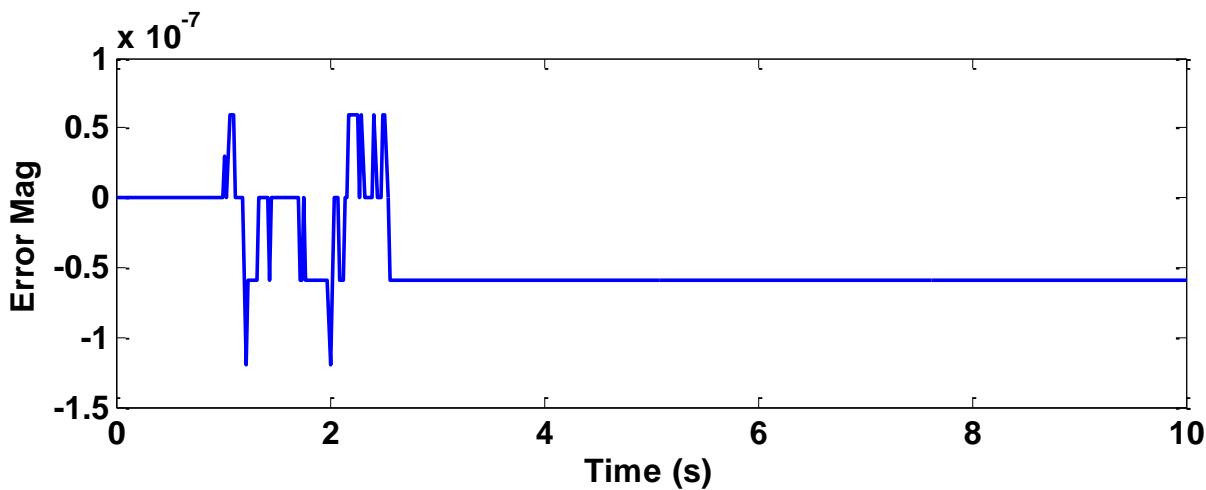
An error in gain is seen in the steady state behavior.
Error is higher and depends directly on the gain.

Filter Initialization



In Model

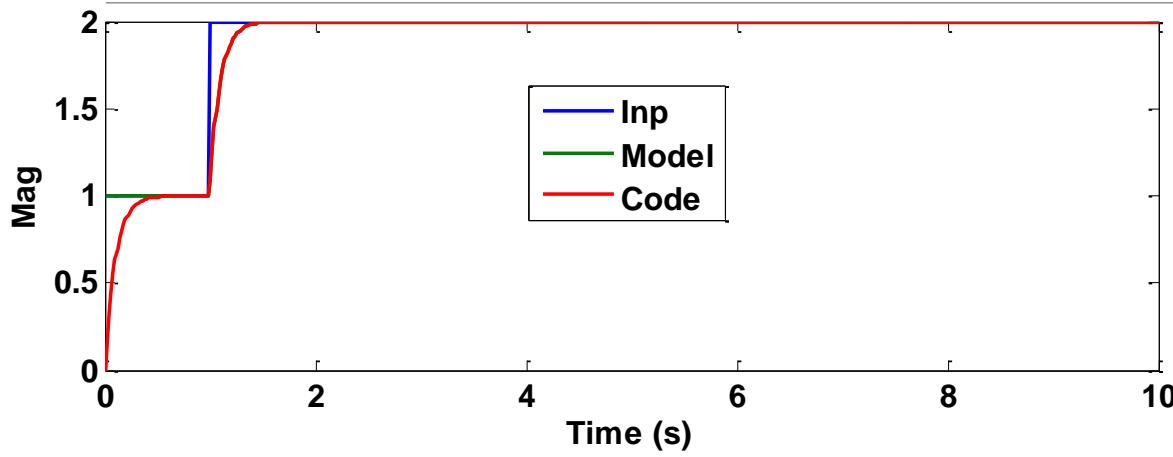
Filter should be initialized so that the output derivatives are 0.0.
IC = Input



In Code

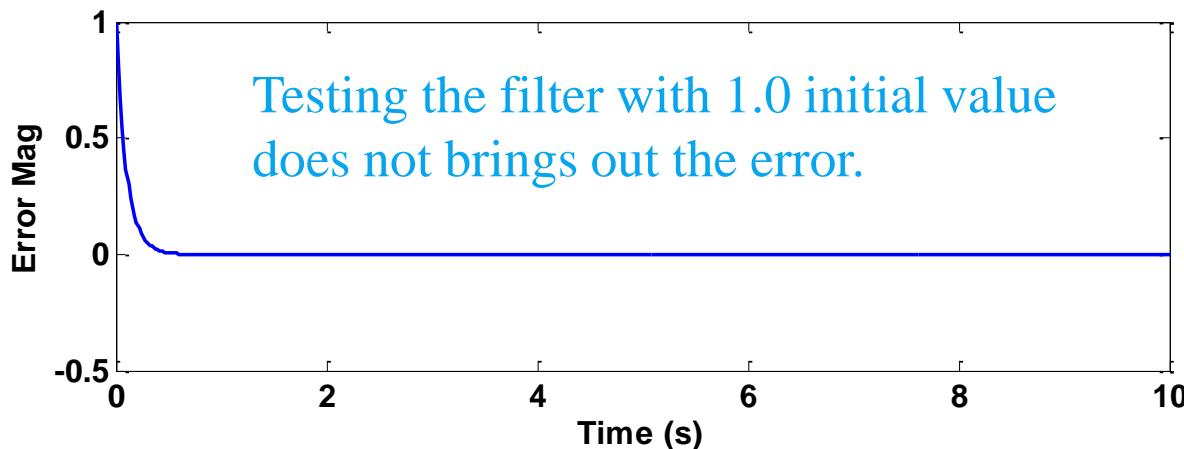
Filter initialized to 0.0

Filter Initialization



In Model

Filter should be initialized so that the output derivatives are 0.0.
IC = Input



In Code

Filter initialized to 0.0

Tips

Scheduled filters are implemented in different ways in Code. This may differ from the Model implementation. The filter behavior will match for constant filter time constant but may give slightly different answers when the time constant is scheduled. Be aware of this fact!

Final Words on Filters

- A large input at the filter input will completely cover the algorithm
- Add a few test cases to check the Initial Conditions. Both True and False conditions of the Initial conditions should be checked.
- It is a good practice to have a non zero value at the filter input in the first frame. This will ensure that in case proper initialization is not happening then the response will not match.
- **Avoid random excitations and very high frequency signals. They may miss out certain aspects of the filter.**

Integrators

- These blocks form a major component of a control system. Some digital filters are implemented using integrators
- Integrators have anti-windup limiters. Care should be taken to see that this is implemented properly in code or in model.
- Integrator output increases for a constant input, hold constant for a zero input and reverses direction if the input sign changes.
- These properties should be used to test an integrator.

Testing Integrators

- Hold a zero input value and ensure that the output holds equal to the initial condition set. Observe this for at least 10 to 15 frames.
- Give a positive constant value and allow the integrator to saturate at the positive limit for a long duration.
- Reverse the input sign and observe the integrator come out of saturation. A long duration in saturation ensures that difference in implementation where a limiter is used instead of the anti windup comes out clearly.
- Repeat the same for negative input.
- Test the reset and Init functionalities if present.

Testing Integrators

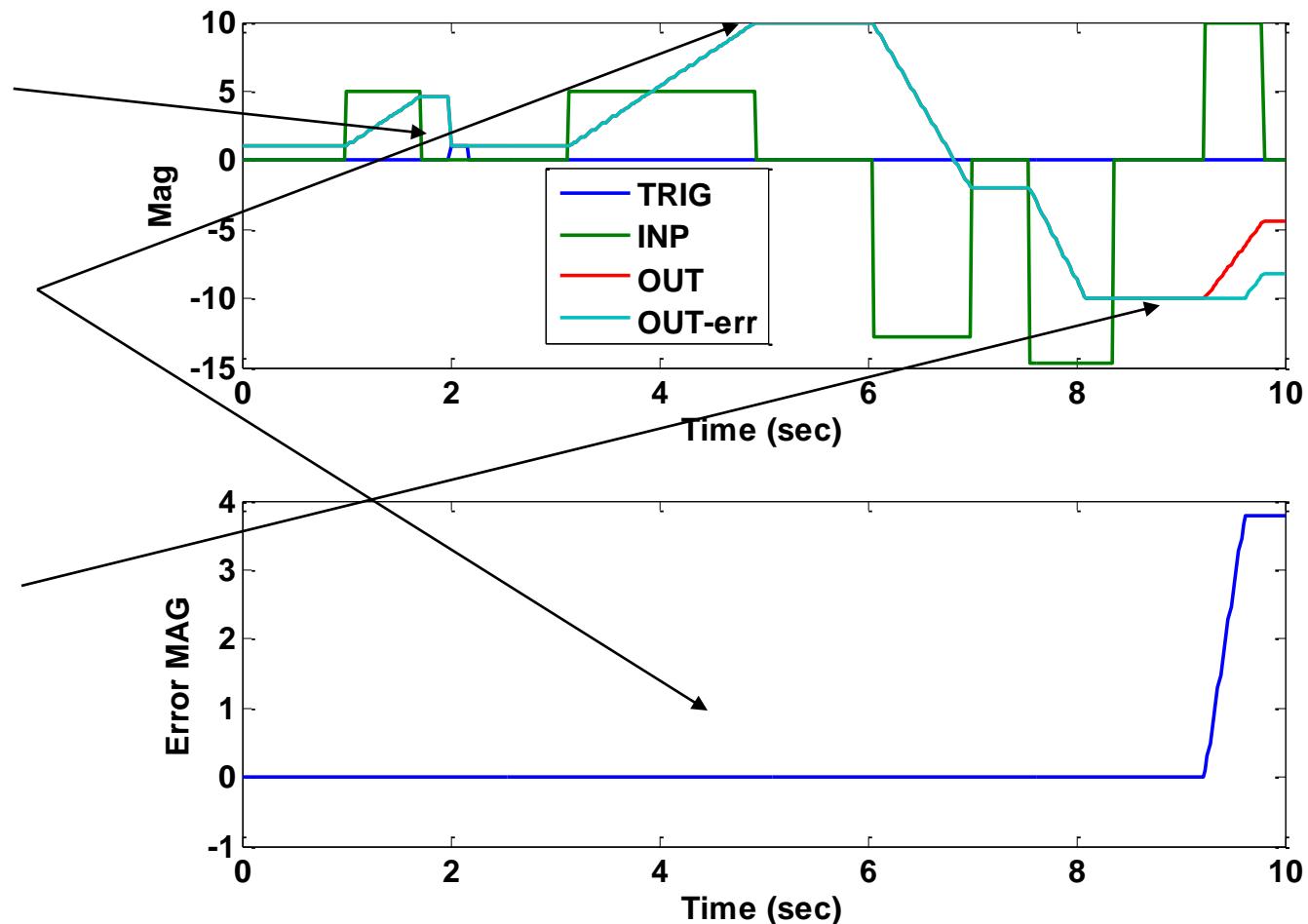
- The initial conditions and reset are checked by giving a reset for at least two different values of the output
- There are instances where the integrator limits are dynamically varying. In these cases the integrator should be checked for at least 2 different values of the limits on both sides.
- Ensure to see that the limits work during initialization. That is if the output is larger than the limit in the first frame does it limit the output. In one implementation the limits were placed in the else part of in the initialization. **It happens!**

Testing Integrators

The Output is set to IC when trig = true.

The input becoming zero just when the output has saturated does not bring out the error.

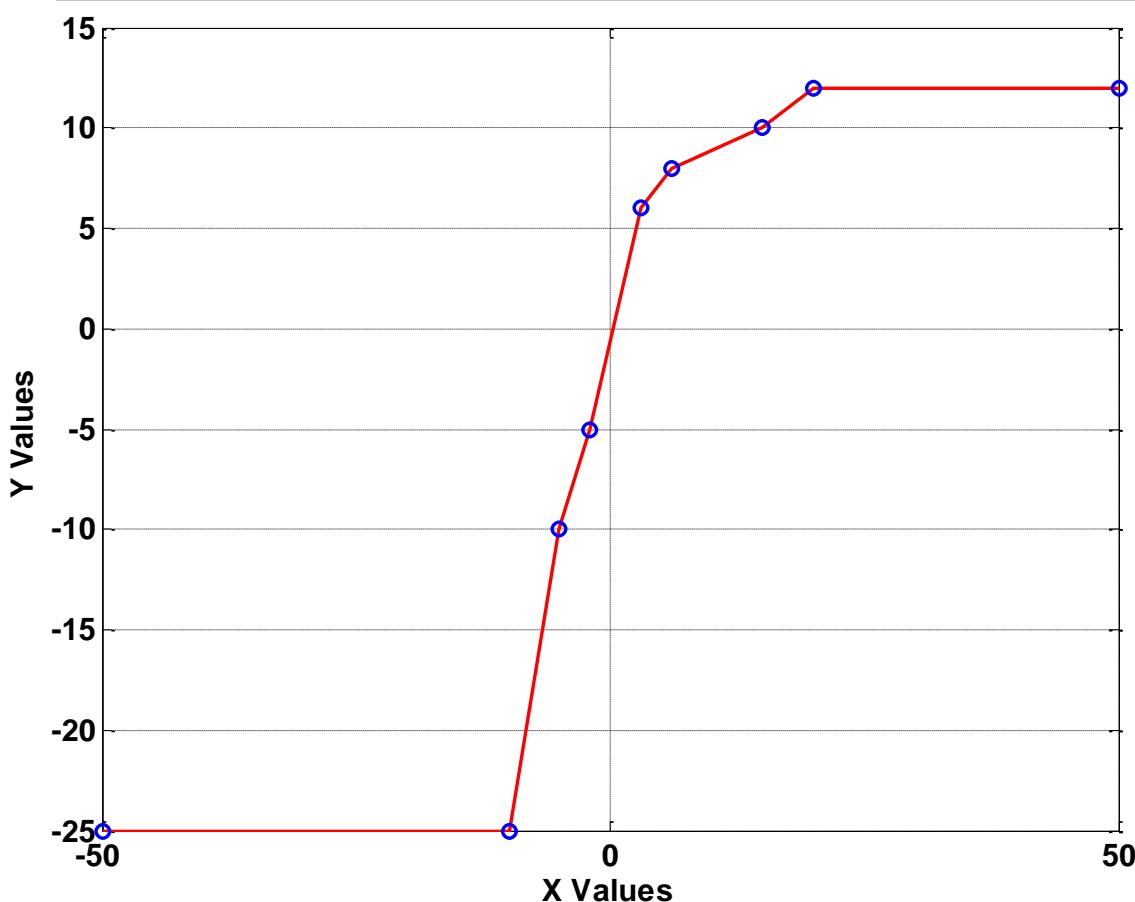
Holding the saturation for a longer duration has caused the error to be observed.



Non Linear – 1D Lookup

- One Dimensional Lookup Table
 - These blocks are used to modify/shape the input in a particular manner.
 - They can be used as variable saturation limits
- 1D tables are characterized by an X-Y relation. The X-Y relation could be continuous or with specified breakpoints
- In control systems a linear interpolation is used to find the values in between breakpoints.
- There are instances when the breakpoints values change based on certain conditions. A switch and two separate tables can be used in such a situation.

1-D Lookup Example



X	Y
-50	-25
-10	-25
-5	-10
-2	-5
3	6
6	8
15	12
20	12
50	12

Testing 1-D Lookup

- A very low frequency sinusoidal waveform with amplitude varying beyond the X values can excite the table completely
- Another alternative is to use a slowly varying ramp signal
- The complete functional coverage can be ensured if there are input signal points
 - Beyond the X extreme values (e.g. -60, 60)
 - At least two points between each breakpoint
 - The two points should be further apart to ensure a linear interpolation and not a cubic or some other.

Tips

Interpolation testing may become difficult if there are other blocks which change the interpolation block's input signal. A single test case may not cover all the points. I have used optimization techniques to come up with a minimized set of tests for these.

Tips

Ensuring a complete table coverage is very important. This data normally changes as flight tests happen and problems are rectified by redesign. It is quite likely that some data may not be changed due to project pressures so late in the program. I have found errors in tables very often.

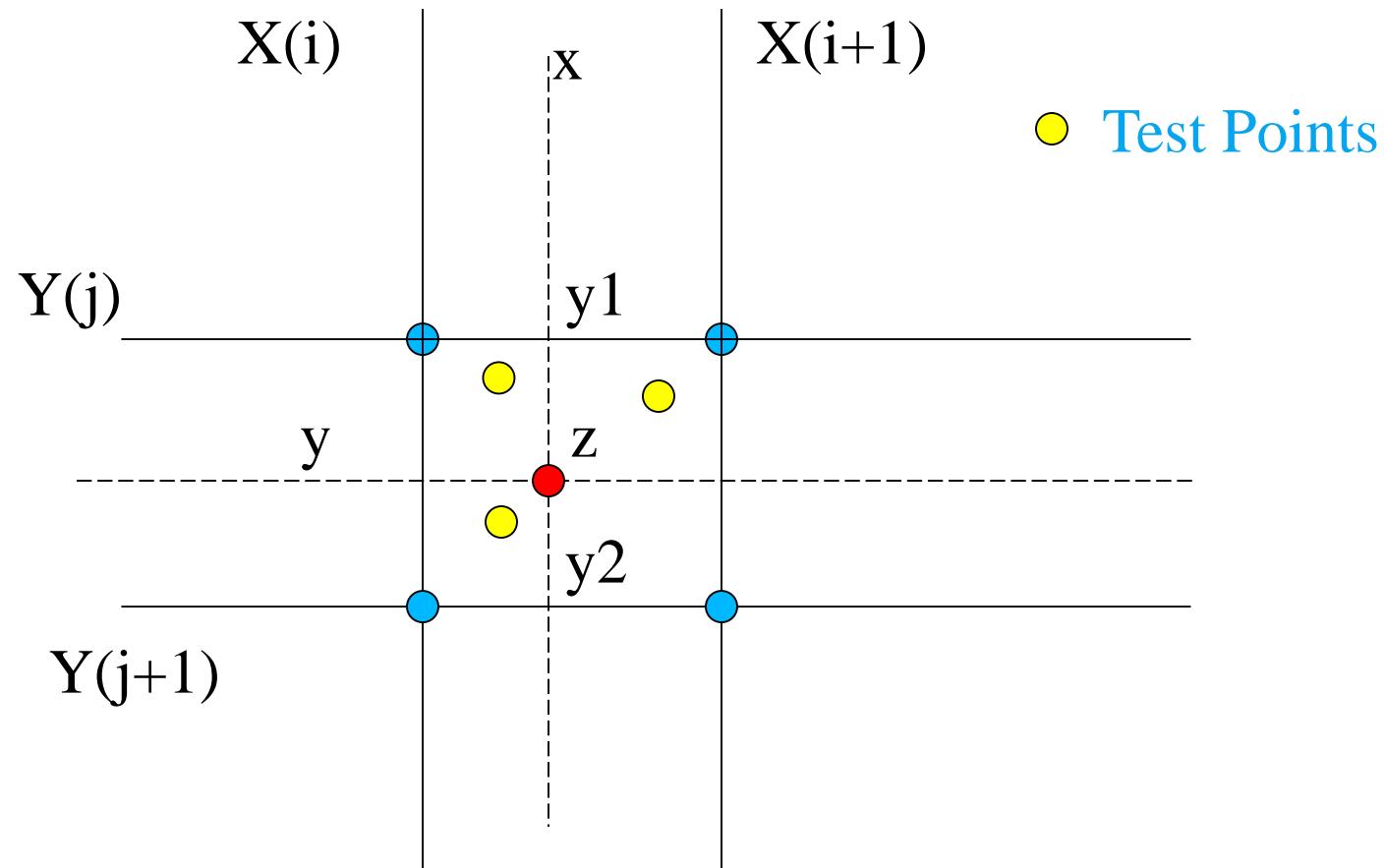
Non Linear – 2D Lookup

- Two Dimensional Lookup Table
 - These are normally used for gain tables in aircraft controllers
 - They can be filter coefficients data also
- The data is provided as a table with Row and Column vectors
- A Linear interpolation is used to find the in between points
- Higher dimension lookup tables are used in simulators and air data systems in aerospace

Testing 2-D Lookup

- The coverage criteria is similar to the 1-D Lookup i.e. two points between break points. In this case both X-Y have to be considered. We require points in each cell.
- One of the axis either X or Y is kept constant and the other input varied as a ramp or sinusoidal signal to scan the values
- Two sinusoidal signals with different frequencies or a step waveform and a sinusoidal waveform can be considered to obtain coverage
- Certain tools like the V&V toolbox of Matlab can provide coverage metrics automatically

Testing 2-D Lookup



Tips

Designers sometimes do not copy paste data from the requirements into the code. Instead they may type this data in. It is also possible that the data is updated in code as it has been used in a system test but this does not get reflected in the updated requirements document. Tables have errors - always!

Tips

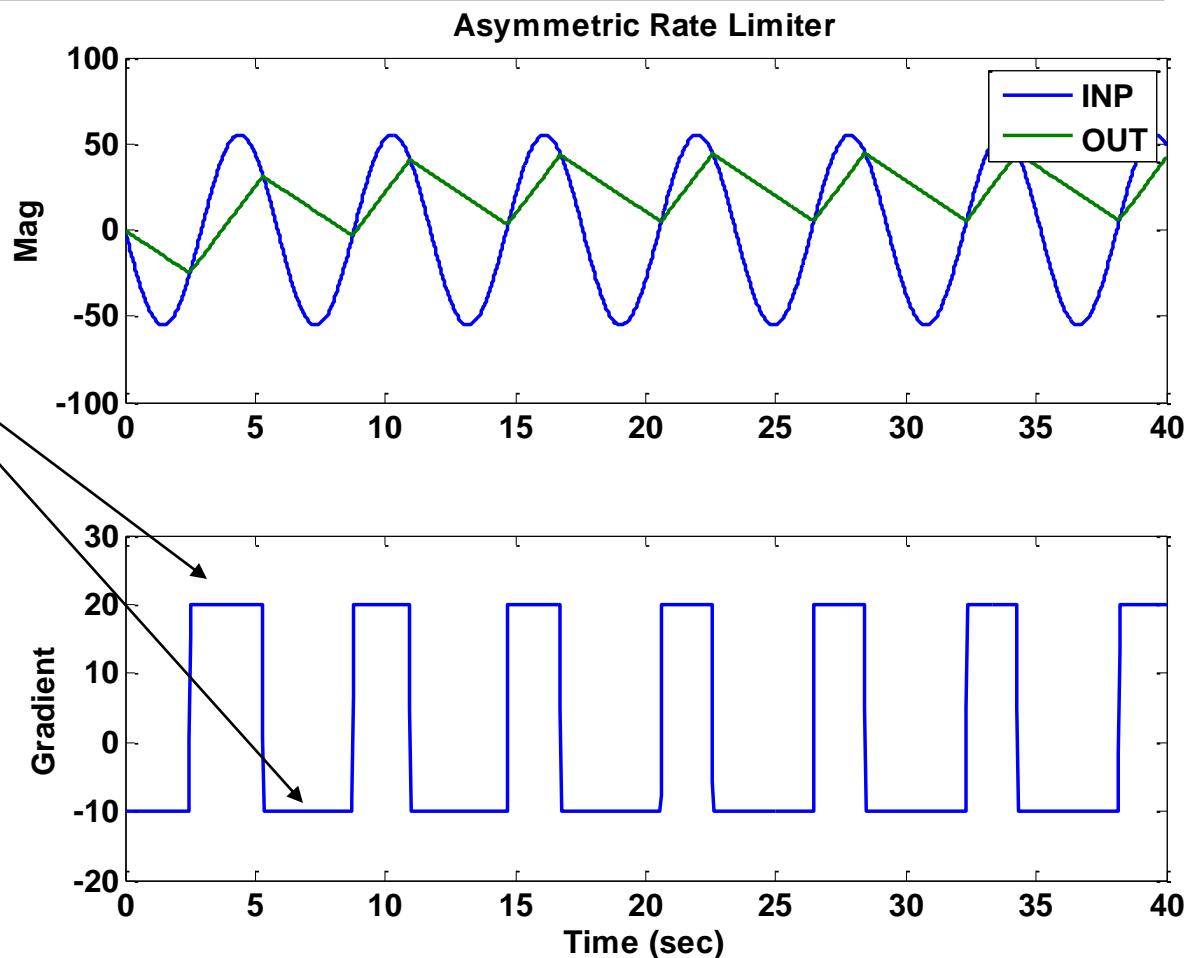
A 3 D visualization of the test data X and Y vs Z shows you the test coverage very clearly. If the testing covers only the edges and extremes you can very clearly see a hole at the center without any data. We need to have a good spread of data to ensure complete table lookup coverage.

Rate Limiters

- Rate limiters limit the rate of the output
- A step input results in a ramp output
- There are variations in the rate limiter implementation
 - Symmetric Rate Limiters
 - Asymmetric Rate Limiters
 - Dynamic Rate Limiters
- The limits are called Max and Min but they are not exactly that – One should specify the Positive Slew Rate and Negative Slew Rate

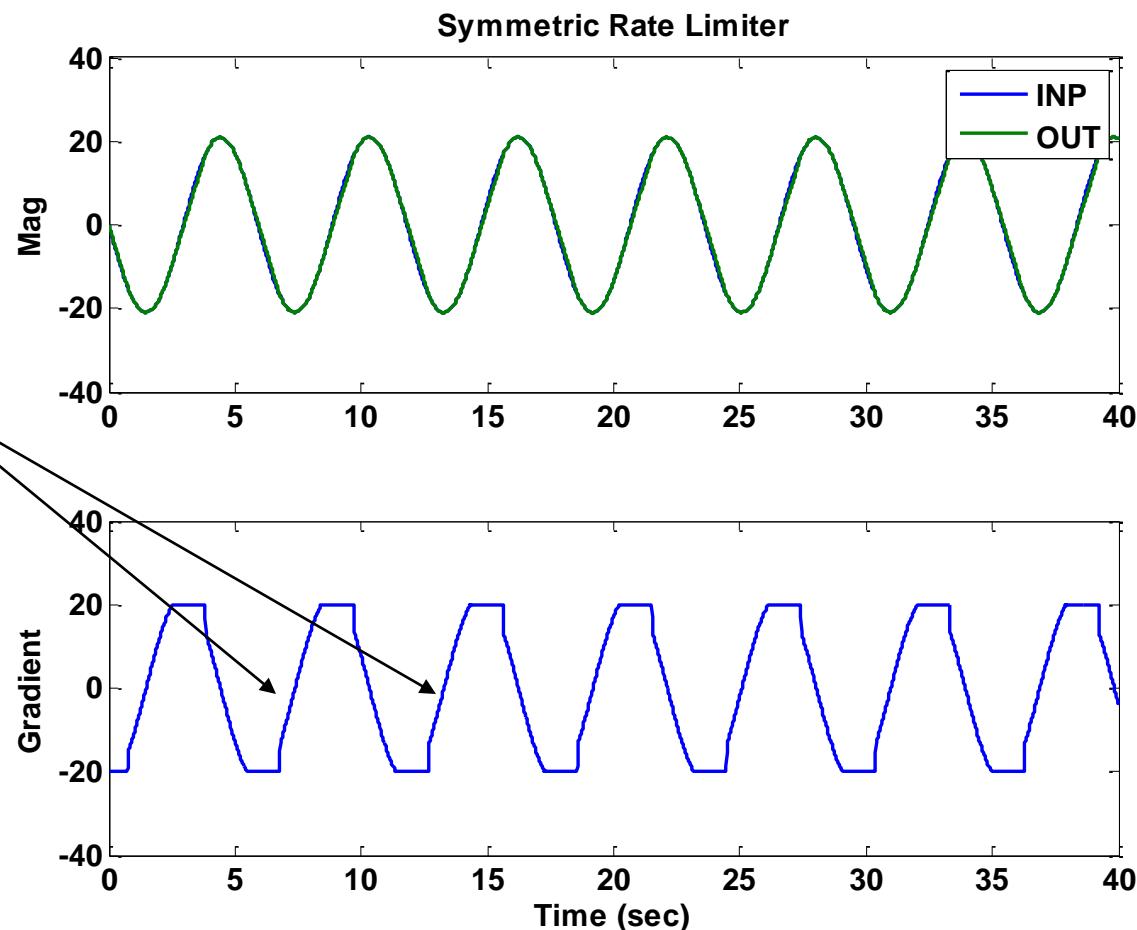
Testing Rate Limiters

The gradient plot shows the two different rates used in the asymmetric rate limiter block (20, -10).



Testing Rate Limiters

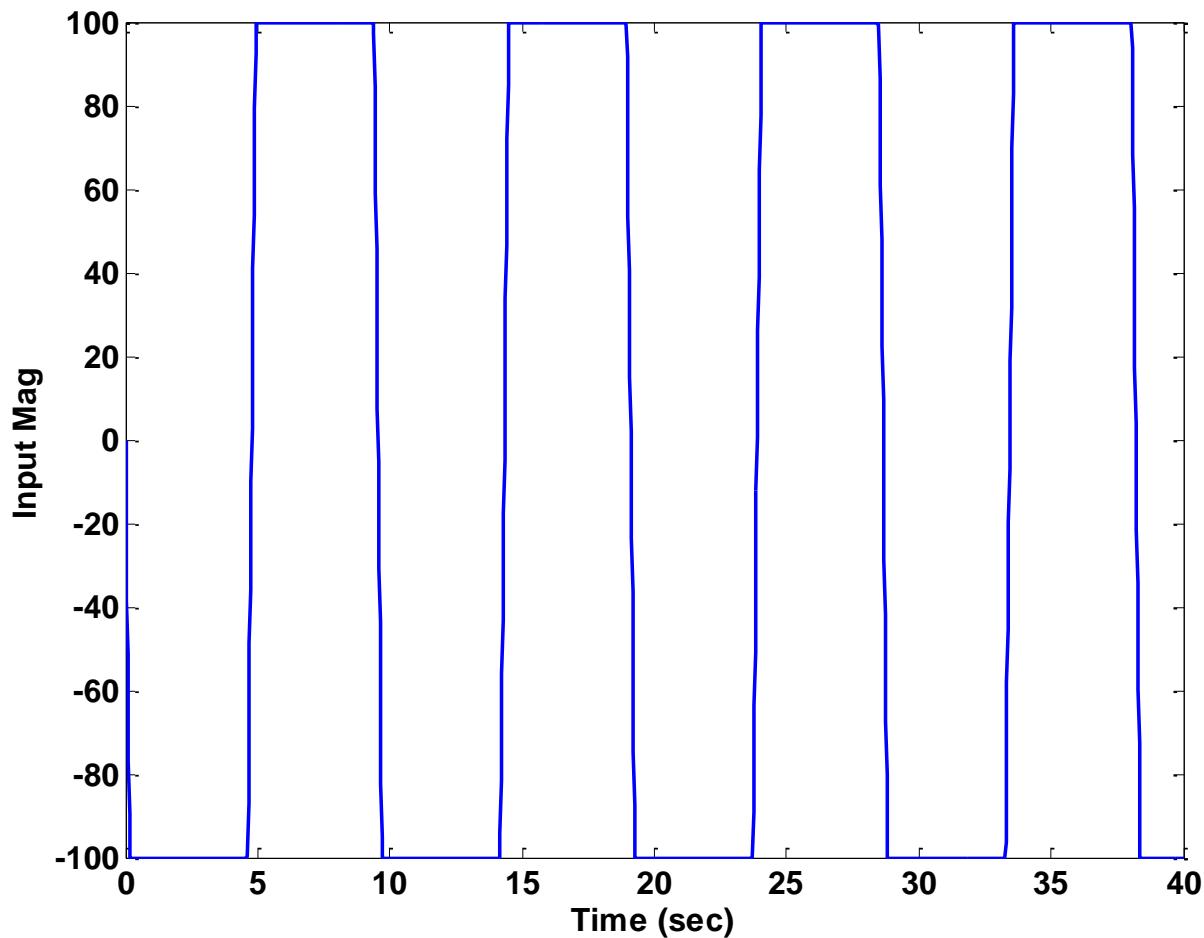
The gradient plot shows the similar rates used in the symmetric rate limiter block (+20, -20). The difference from the previous plot is there are zone where the rate limits have not been hit. This checks for the else condition effectively.



Testing Rate Limiters

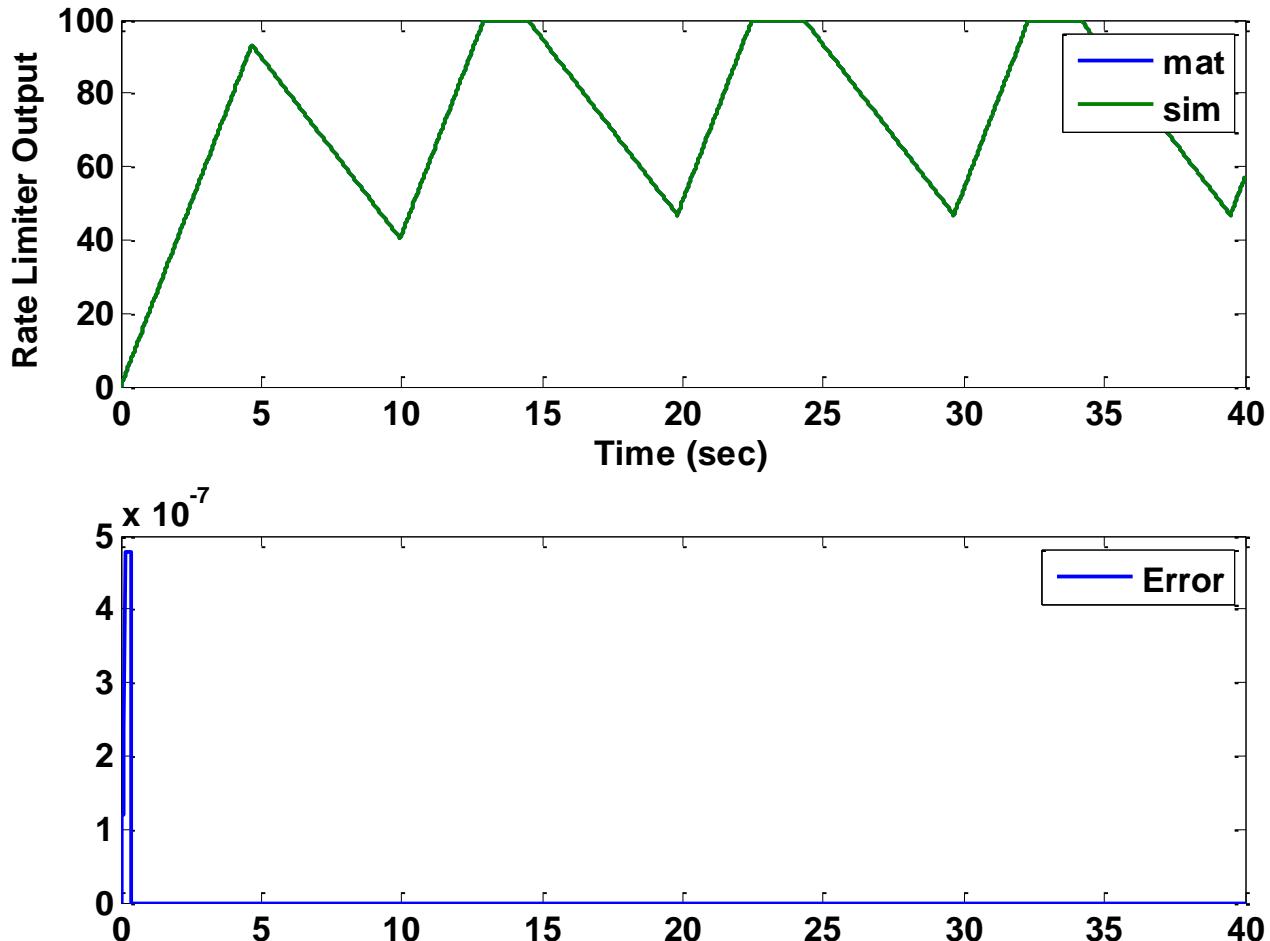
LARGE
PULSING
INPUTS ensure
hitting the Rate
Limit

But ...



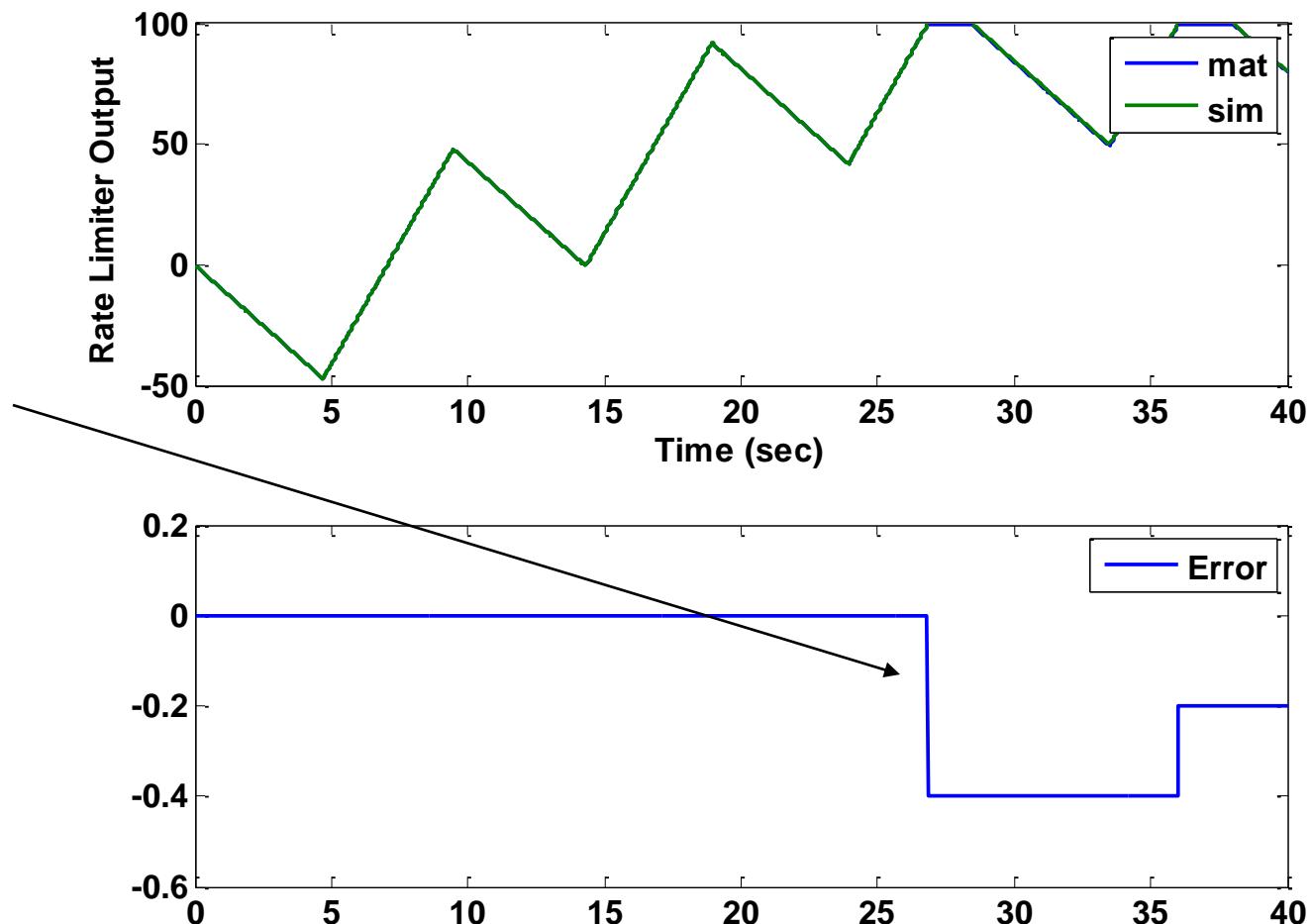
Testing Rate Limiters

They may not be able to capture errors as seen from the plots.
The error is in order of 10^{-7} thus passing the tests.



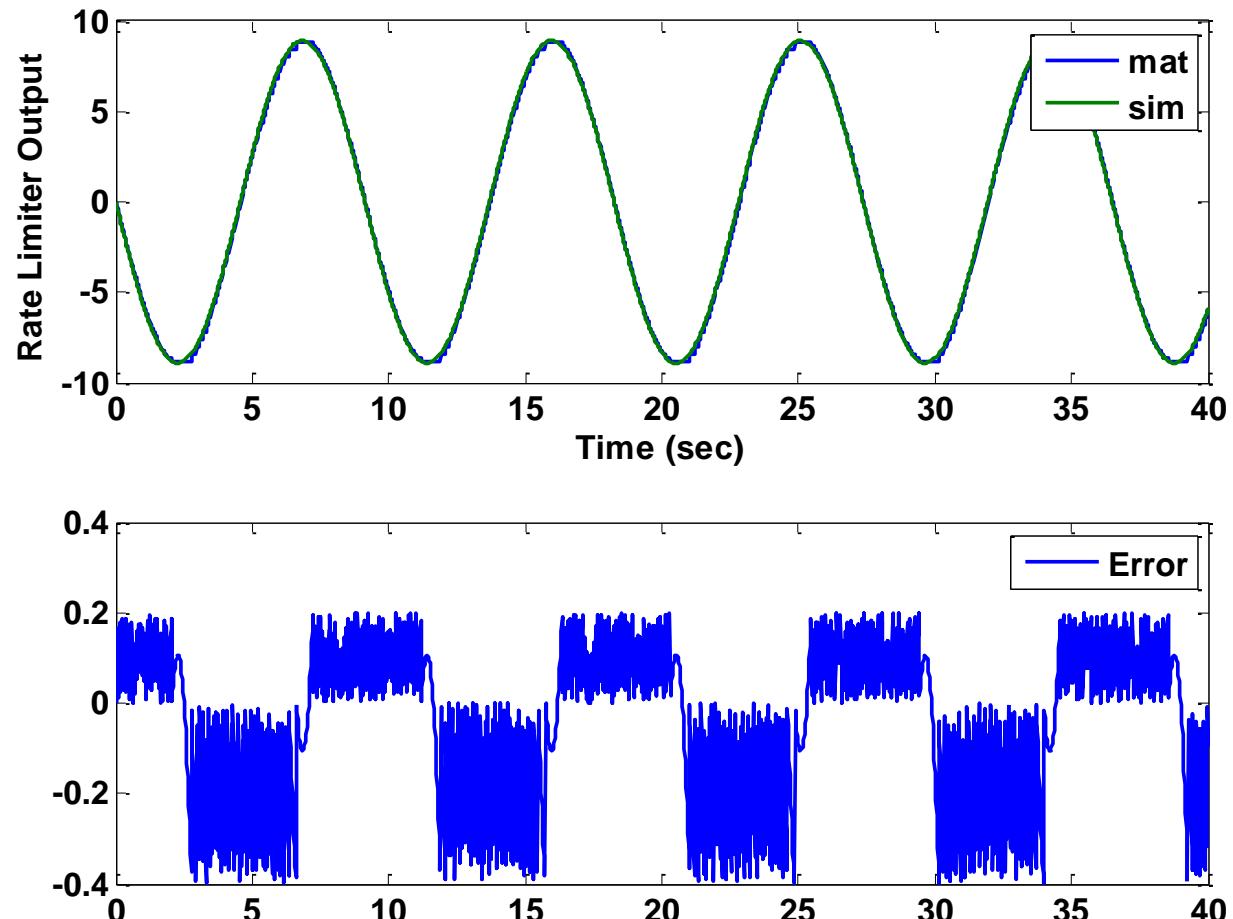
Testing Rate Limiters

Error has been observed after a long run.
Input signal was having a rate limit throughout.
Therefore this error could not be trapped



Testing Rate Limiters

A signal with a rate less than the rate limit has brought out the error earlier.



Testing Rate Limiters

```
if ic == true
    out = Initial_value;
else
    ll = out-abs(LL*dt);
    ul = out+abs(UL*dt);
    if INP < ll
        out = ll;
    elseif INP > ul
        out = ul;
    else
        out = INP
    end
end
```

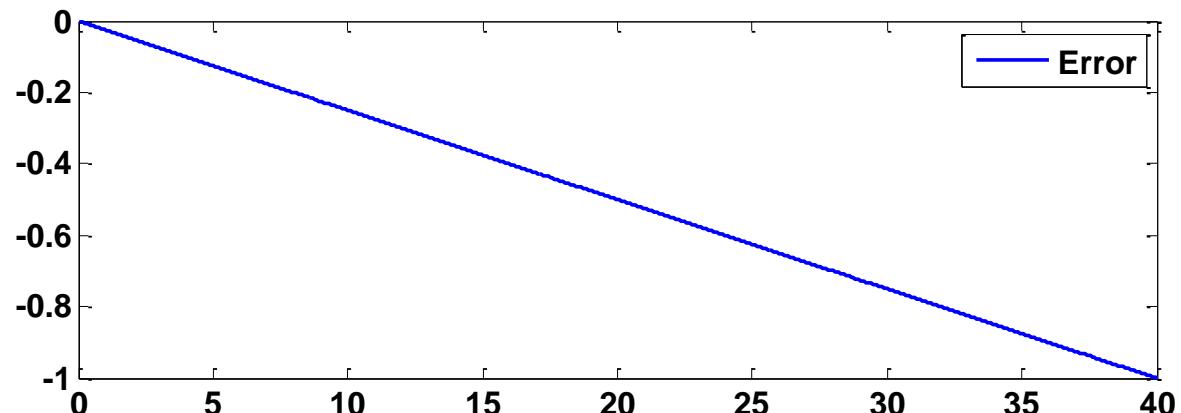
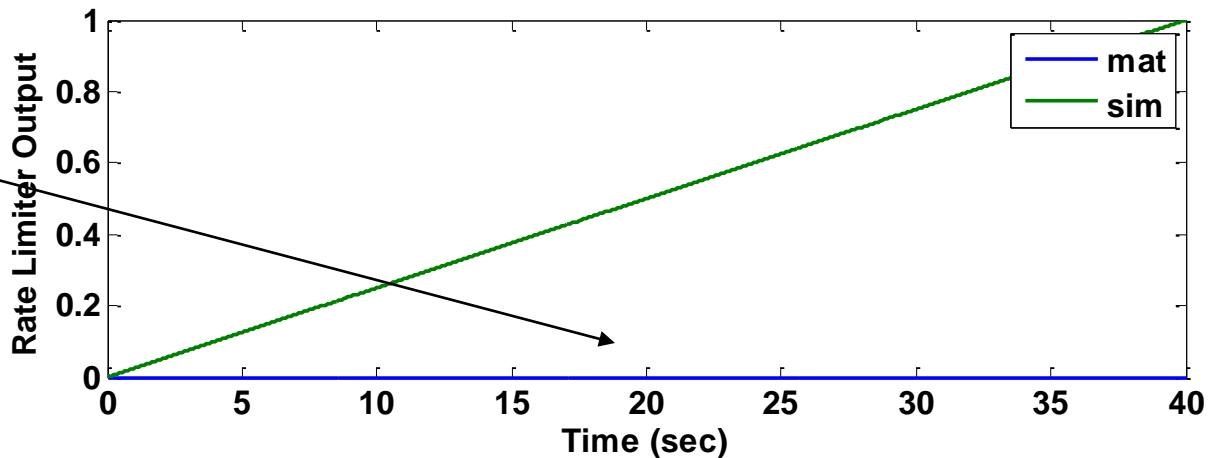
- if ic == true
- out = Initial_value;
- else
- ll = out-abs(LL*dt);
- ul = out+abs(UL*dt);
- if INP < ll
- out = ll;
- elseif INP > ul
- out = ul;
- end
- end

The else condition has been dropped in the code. This would have been trapped with code coverage if algorithm was defined as a flowchart. With model based testing complete functional coverage is required to bring out error – which is a major one. See next plot.

This is an actual scenario!

Testing Rate Limiters

The code
output is 0.0
throughout!



Tips

The rate limiter example proves the necessity of code reviews, functional coverage, code coverage and reviews in a safety critical control system design and testing. My sincere advice is do not dilute the effort in any of the artifacts. The rigor is required and the objectives and guidelines help us maintain this. We make mistakes!

Saturation

- This is a simple amplitude limiter
- There can be problems in an implementation of the simple saturation also

- What happens if the Upper Limit, specified or dynamically arrived at, is Lower than the Lower Limit?

```
a=2;ul=5;ll=10;  
  
if a >= ul  
    a=ul;  
elseif a <= ll  
    a = ll;  
end
```

- Is it protected for a Safety Critical Application?

Tips

A testers job is never done. A little paranoia helps. They say a tester will have to look at both directions when crossing a one way street. It is so very true and required in India.

Persistence

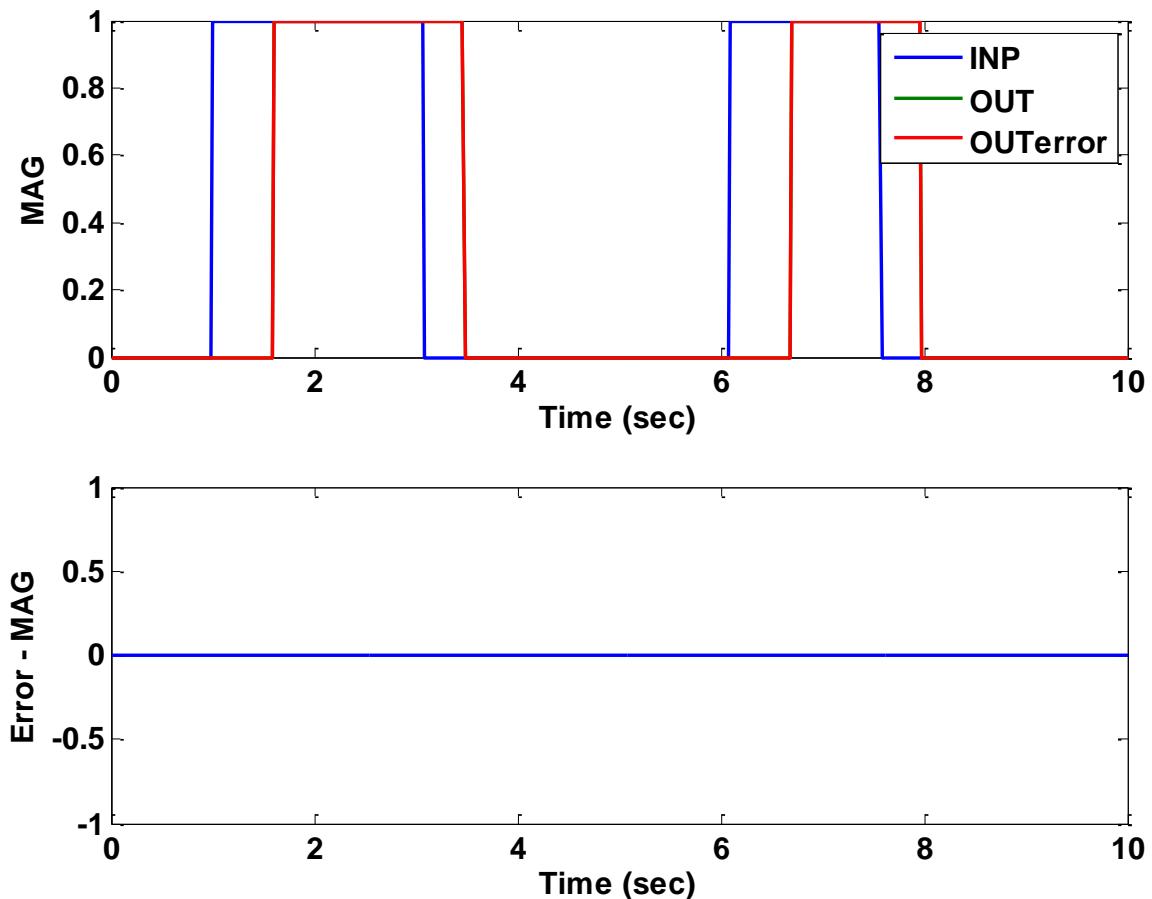
- These blocks are used to check for failures and to observe them over a period of time to see if they “persist”. If they do then a failure is declared
- There are various type of these blocks
 - Persistence On/ Off
 - Persistence OnOff (Together)
 - In Window On/Off/OnOff
- A Persistence On block will become ON (True) if the input is True for a duration greater than ON Time. If it becomes False anytime Output will be False.

Testing Persistence

- The normal operation is checked by setting the required conditions, keeping the input ON/OFF for a duration greater than the Persistence time.
 - There should be sufficient cases to ensure that the input toggles before the persistence time and after it also.
 - Different combination of input toggling have to be used to verify the functionality
-
- This is a good candidate for Random Testing!

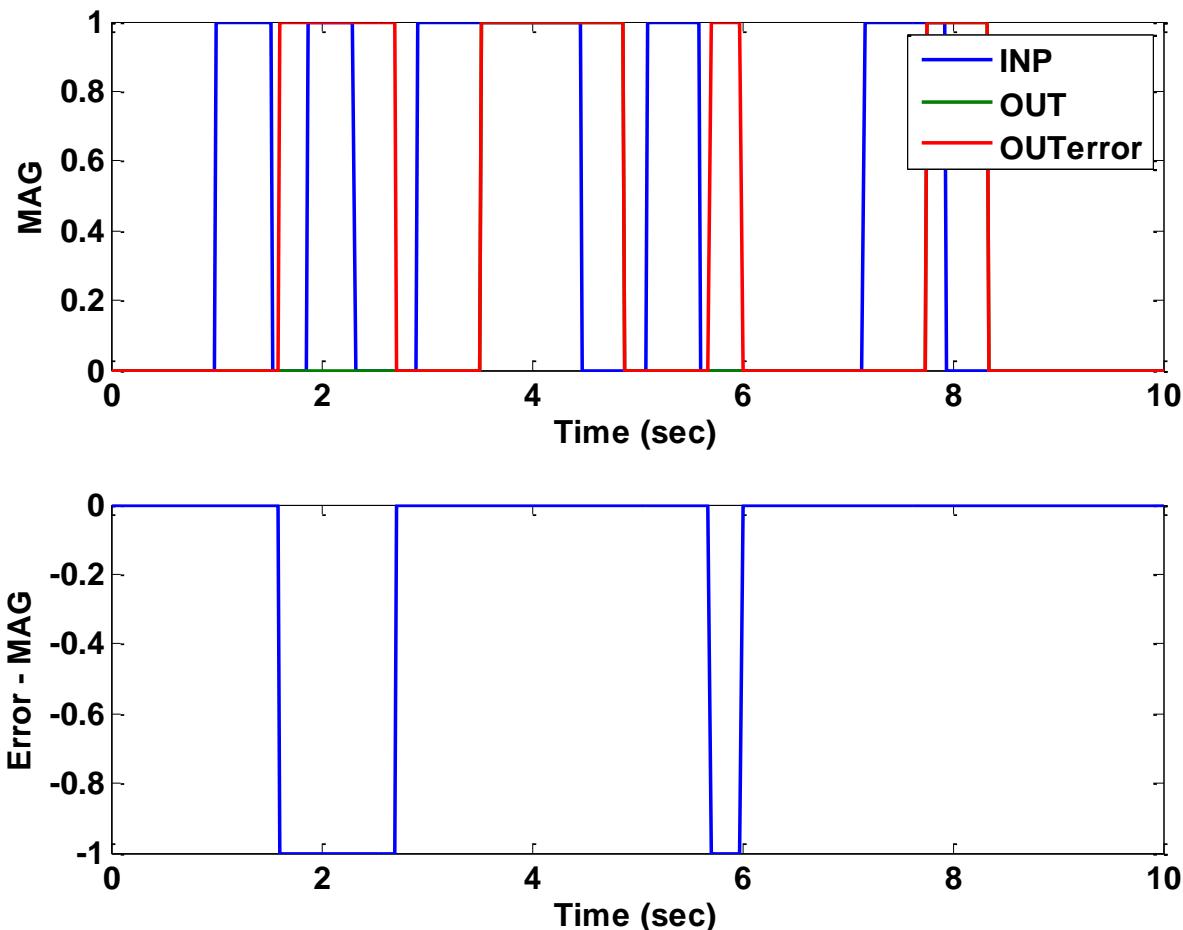
Testing Persistence

A very standard test case with the input changing greater than the DTon/DTOff times has not brought out any error in the Delay On Off behavior

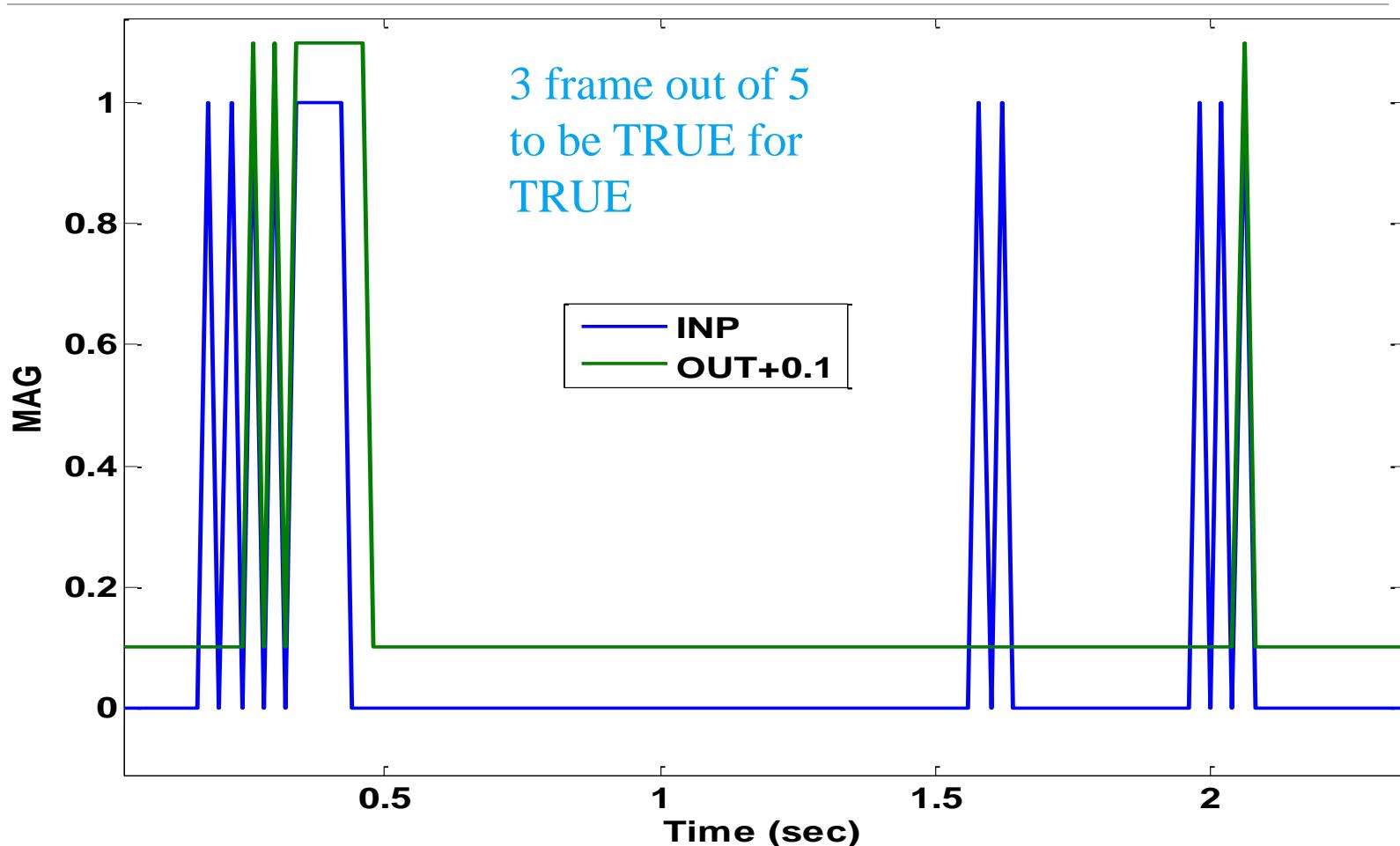


Testing Persistence

A toggle between DT has brought out the error in the behavior.
This is an actual case. Delay OnOff was modeled as Delay On in series with Delay Off. This is not the expected behavior



Window Counter Behavior



Latches

- Latches are used to set a particular failure flag so that it can be cleared only based on the reset
- They would normally be used after the Persistence On/Off blocks to set a failure
- There are two type of latches
- Set and Reset Priority based on what happens when the Set Signal and the Reset Signal both are ‘1’

Testing Latches

- Latches have to be tested for the full truth table for Set and Reset
- Latches are normally incorporated with Persistence blocks for Set and complex logics for Reset. Testing such situations are tricky as the only global available in code would be the Latch output. Complex waveforms to test the Persistence along with the latches have to be designed to test the circuit.
- These test cases should test the full truth table (point 1)

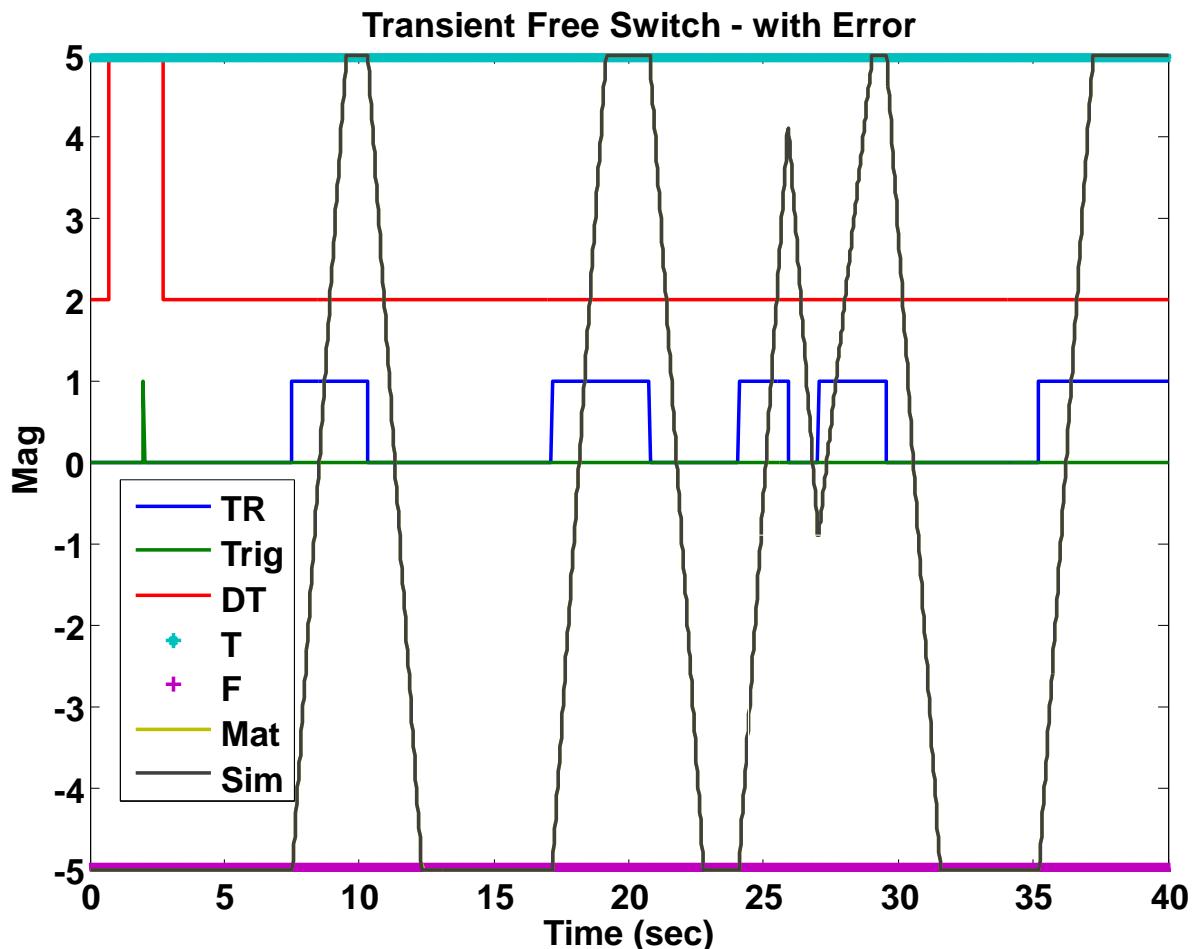
Testing Transient Free Switches

- The testing of Transient Free switches is similar to the Persistence On/Off blocks.
- We have to test the switch toggle for greater than the fade in time and for durations less than the fade in time.
- With the event toggled in this fashion we have to set the True and False signal inputs to constants. This demonstrates the proper functioning of the TFS.
- Keeping a similar toggle profile we have to test the TFS with sinusoidal inputs of different amplitudes and frequencies applied to the True and False inputs. This type of testing brought out the anomaly described earlier.

Testing Transient Free Switches

If DT Toggles
then the fading is
computed. This
was an actual
error in the initial
versions of our
test activity.

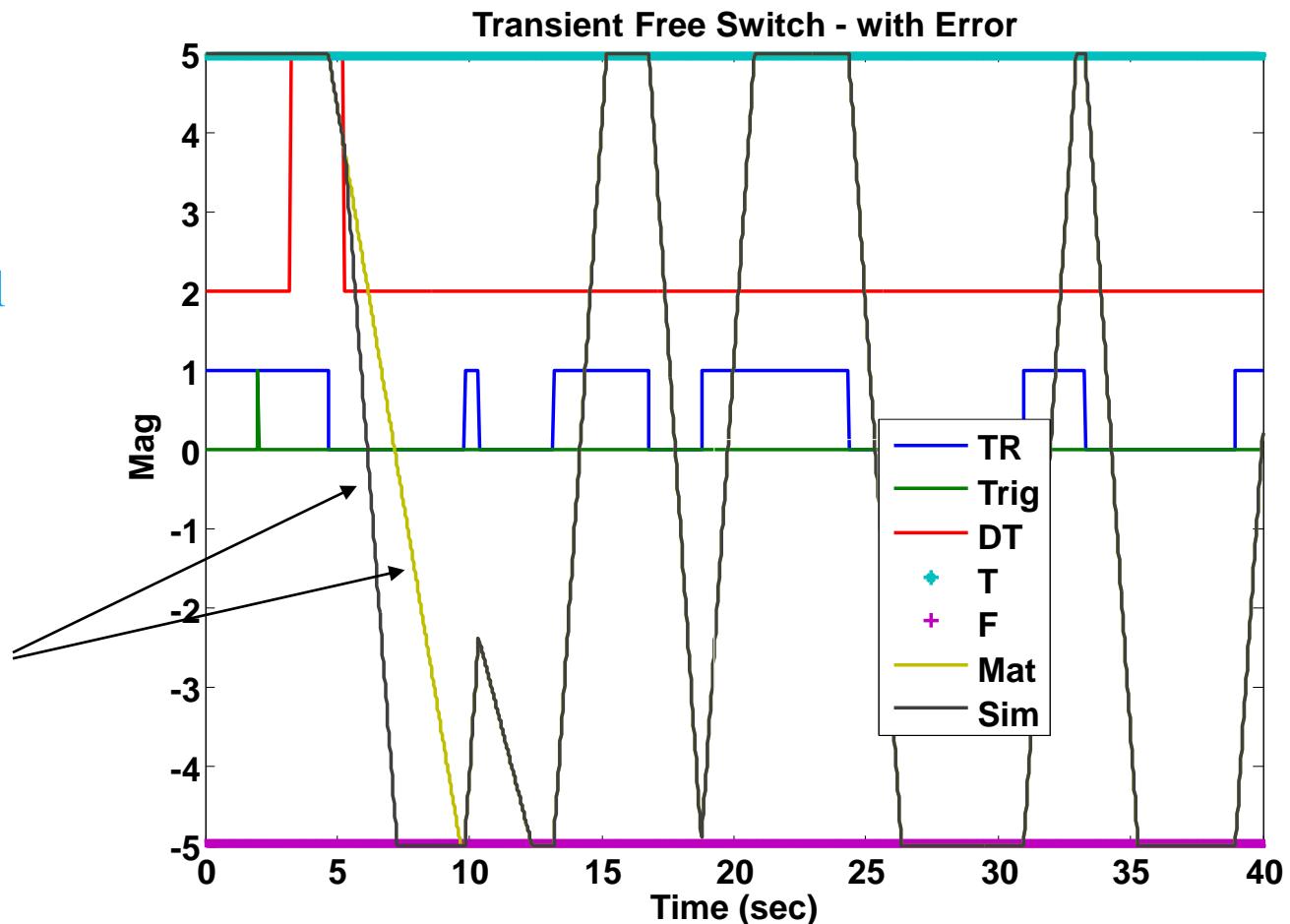
This is not
caught by the
specific test case.



Testing Transient Free Switches

If DT Toggles
then the fading is
computed. This
was an actual
error in the initial
versions of our
test activity.

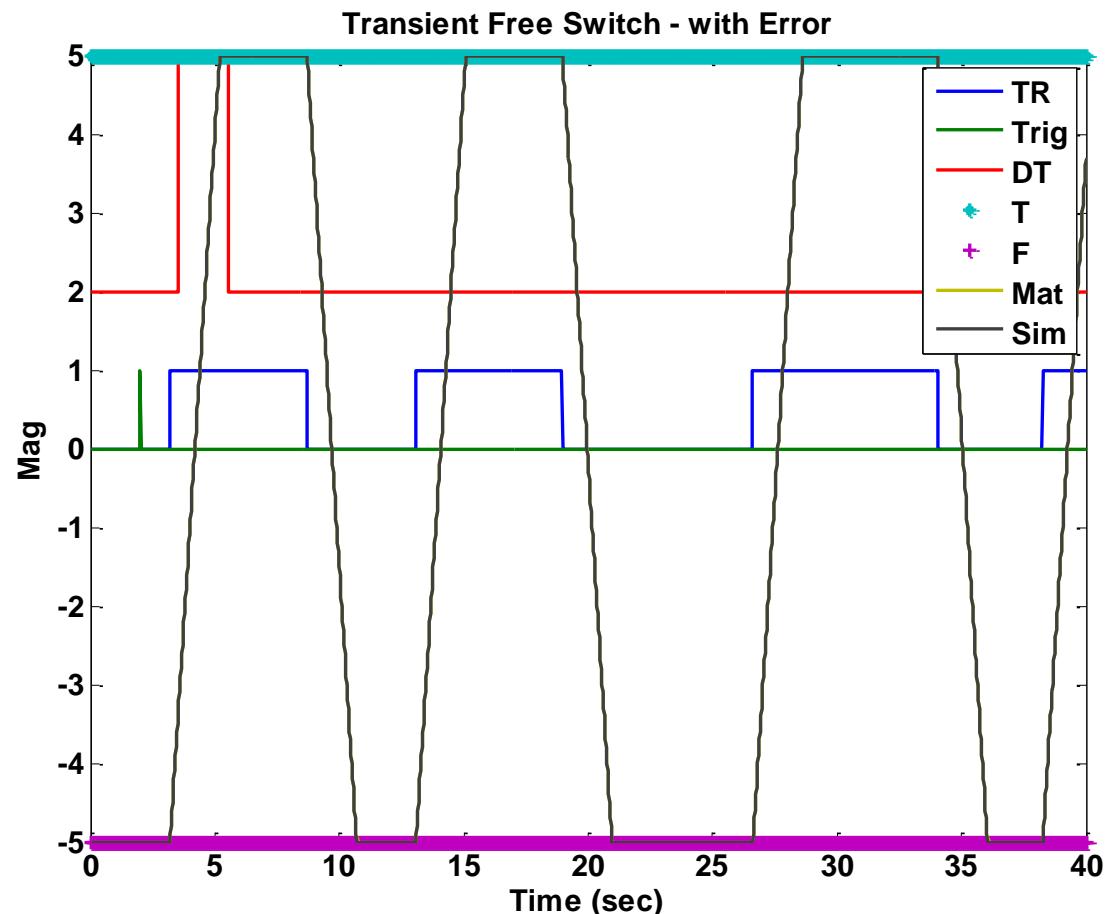
This is caught by
the specific test
case. DT toggled
independent of
other toggles.



Testing Transient Free Switches

A transient Free switch variant for constant is used in the code instead of the TFS. (Actual Error)

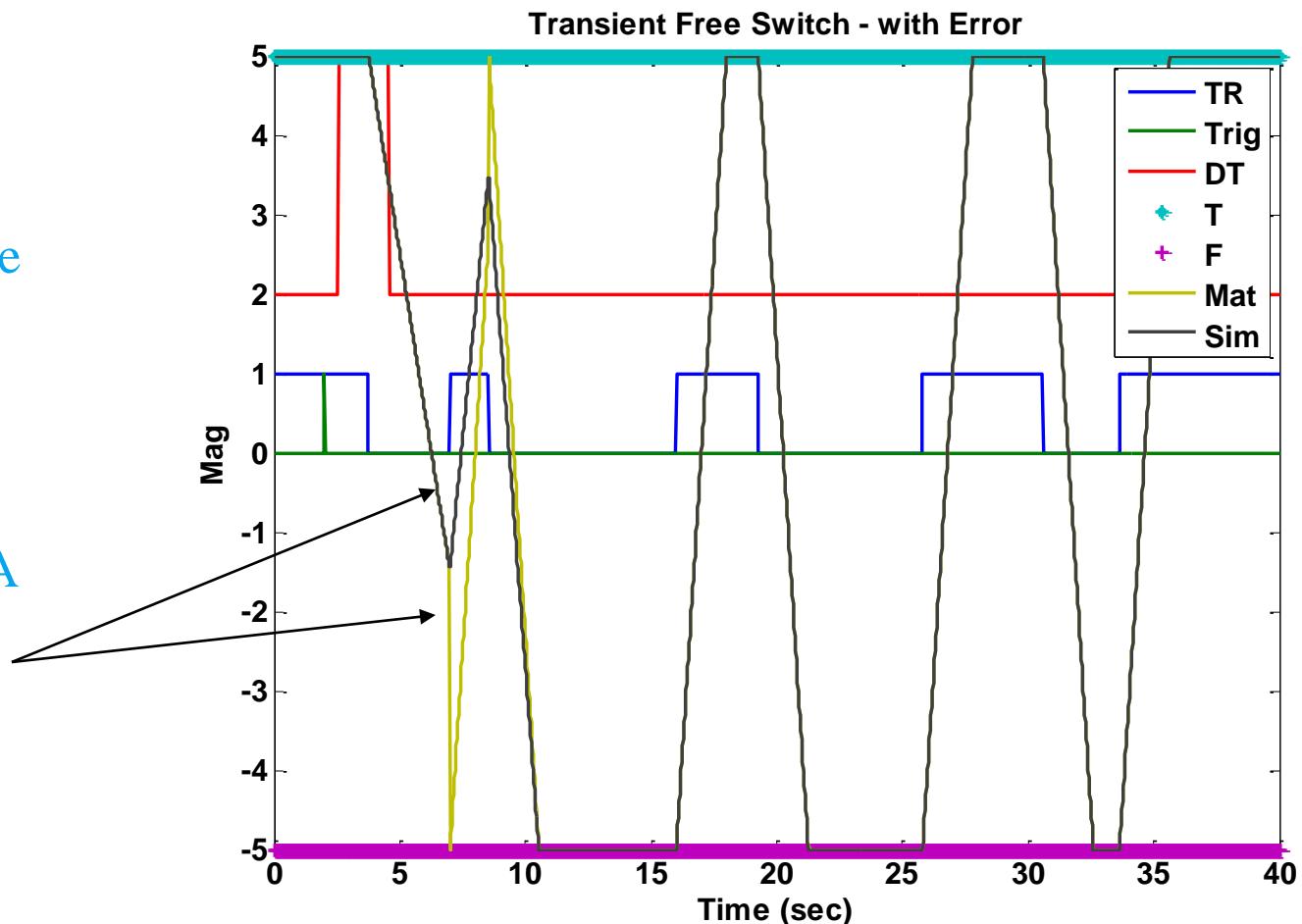
This test case does not bring out the error



Testing Transient Free Switches

A transient Free switch variant for constant is used in the code instead of the TFS. (Actual error)

This test case does brings out the error. A toggle less than fade time has brought out this error.



Tips

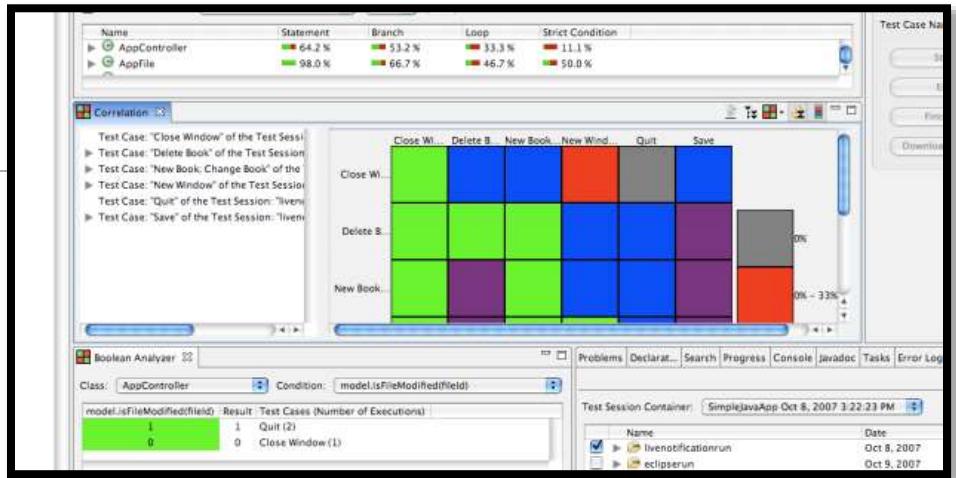
Testing control system blocks is all about understanding how the block works, and a curiosity to break the block. This leads to good test designs which bring out errors in the system before sending it to the aircraft.

Tips

A set of blocks and code is available on the Mathworks website. Anyone starting control system test activity or coding can use these blocks. Check out my MathWorks contribution and Chethan's contributions on file exchange

<http://www.mathworks.in/matlabcentral/fileexchange/authors/75973>
<http://www.mathworks.in/matlabcentral/fileexchange/authors/110838>

Coverage Metrics



http://codecover.org/images/overview_plugin.png

Existing Coverage Metrics

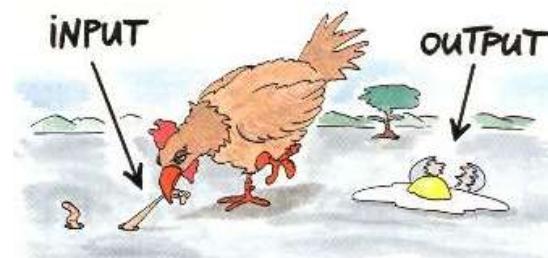
- Code coverage or structural coverage metrics
 - These look at Statement Coverage, Decision Coverage, Condition Coverage, Multiple Condition Coverage, Condition/Decision Coverage, Modified Condition/Decision Coverage
- Block coverage metrics
 - These look at Decision, Condition, MC/DC, Look-up Table, Signal Range (Simulink V&V Toolbox)
- *We have seen in “Mistakes” that these are inadequate. We require better metrics*

(c) chethan.cu@gmail.com

Metric Characteristics

- The metric should be functionality based.
- The metric should be based on the input - output relation of the block under test.
- The metric should be independent of the platform being used.
- The metric should have an capability of test case optimization.

Cu, C.; Jeppu, Y.; Hariram, S.; Murthy, N.N.; Apte, P.R., "A new input-output based model coverage paradigm for control blocks," Aerospace Conference, 2011 IEEE , vol., no., pp.1,12, 5-12 March 2011, doi: 10.1109/AERO.2011.5747530



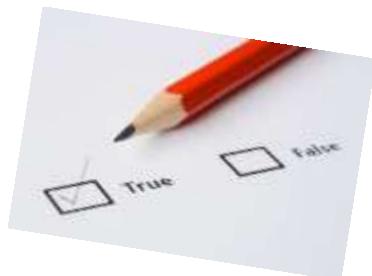
(c) chethan.cu@gmail.com

New Metric

We define a pair of cells for each functional requirements.

The first cell of the pair is discrete (TRUE/FALSE) which tells if a particular functional requirement is exercised or not.

The second cell defines a “distance to coverage”, a continuous metric which can be minimized to ensure coverage



T/F	Distance to coverage
-----	----------------------

(c) chethan.cu@gmail.com



Integrator Requirements

The Integrator ***shall*** be implemented as per the equation

$$\text{Output} = \text{Previous Output} + DT * \text{input}$$

Where, Previous Output is the output obtained at the previous execution frame and DT is the sample time.

The output during the first frame of execution ***shall*** be equal to IC.

If the Output is greater than UL then Output shall be made equal to UL

If the Output is less than LL then the Output shall be made equal to LL

When the Integrator Output has reached a limit, the output will be limited such that any Input sign change will be immediately reflected in the Output.

Integrator Metrics

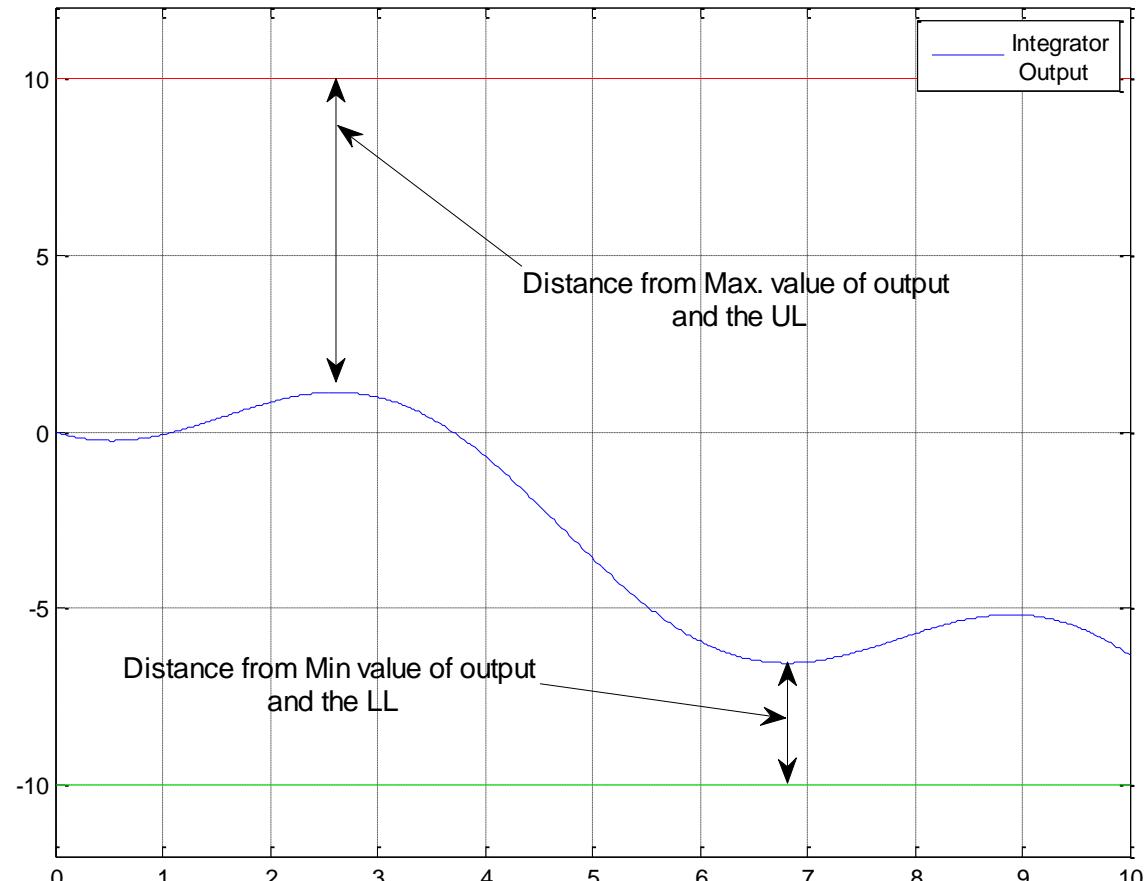
The anti windup integrator has 5 requirements.
The metrics look at these requirements and how we would test them

#	Discrete Metric	Continuous Metric
1	Output of the Integrator has reached the UL	$\text{abs}(\min(\text{Output}-\text{UL}))$
2	Output of the Integrator has reached the LL	$\text{abs}(\min(\text{Output}-\text{LL}))$
3	Output is non-zero and lesser than UL and greater than LL	$\text{abs}(\min(\text{Output}(\text{NonZero}) - (\text{UL}+\text{LL})/2))$
4	Integrator comes out of saturation from UL	When Output == UL, Drive input towards values <0
5	Integrator comes out of saturation from LL	When Output == LL, Drive input towards values >0

(c) chethan.cu@gmail.com

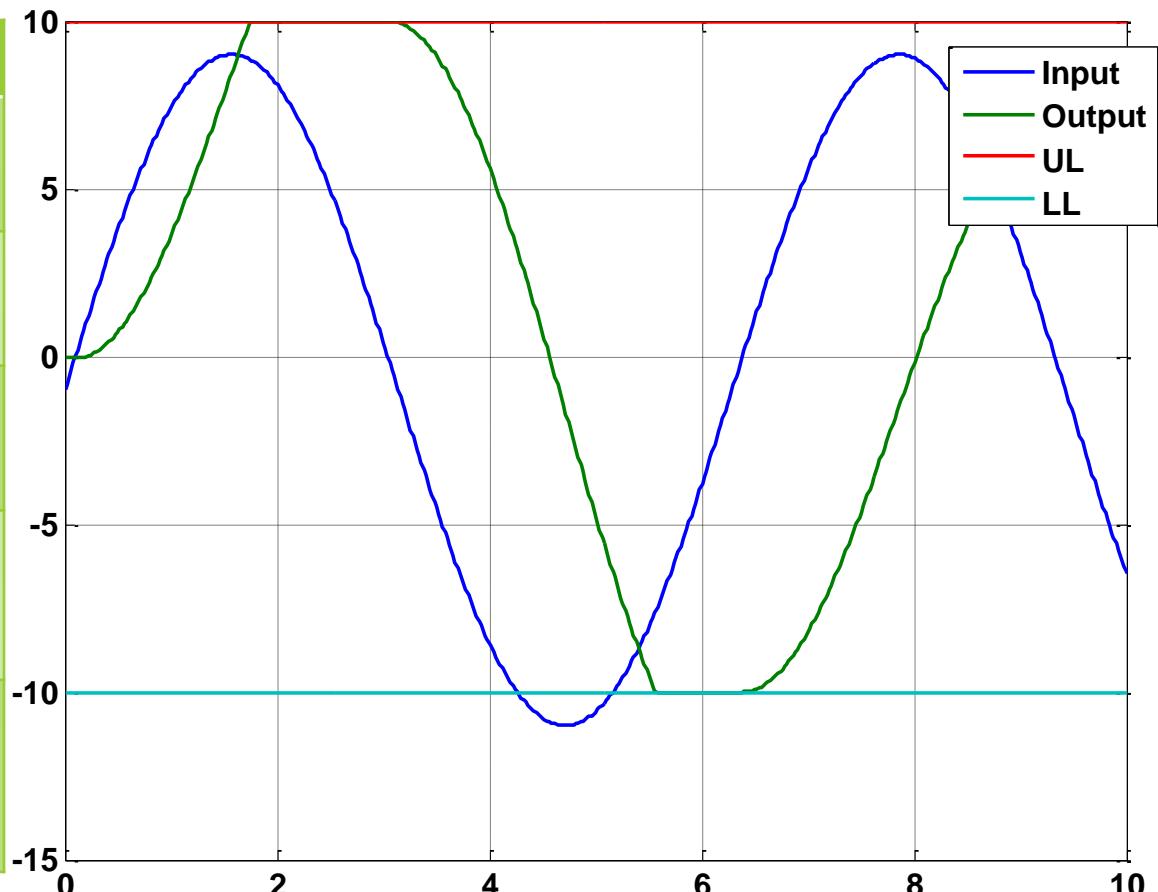
Integrator Metrics

The distance metrics are defined as distances from the output to the required saturation



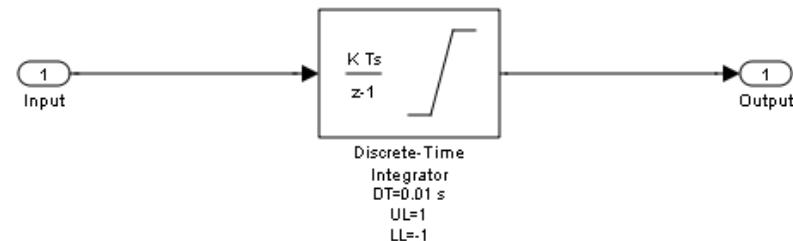
The New Coverage Metrics

Metric	Count
Output \geq UL	65 frames
Output \leq LL	41 frames
Output is non-zero, Output $<$ LL and $>$ UL	395 frames
Coming out of saturation of UL	1 transition
Coming out of saturation of UL	1 transition

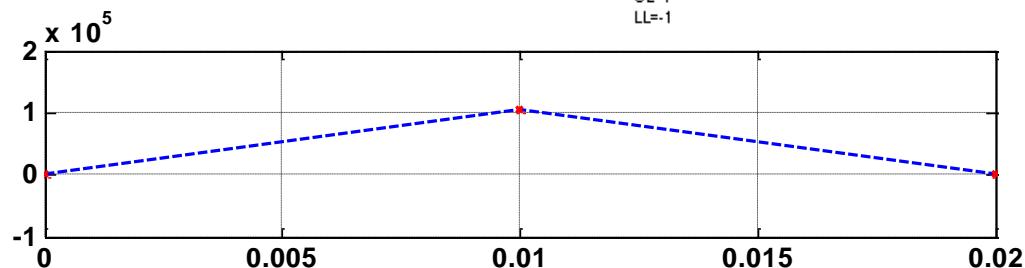


(c) chethan.cu@gmail.com

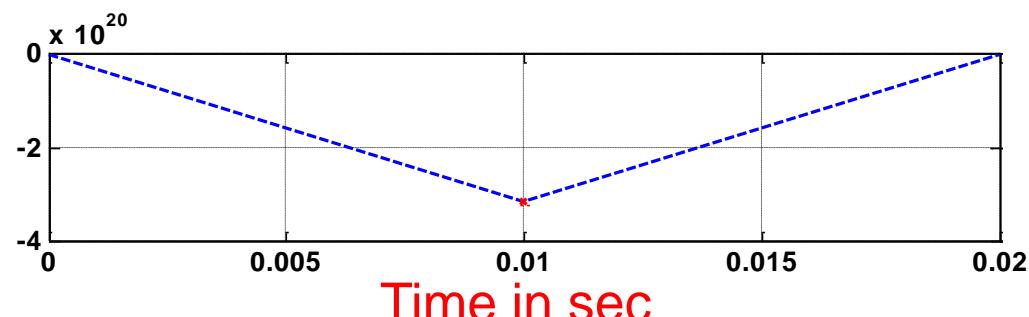
Reactis Test Case



Test Case 1



Test Case 2



Reactis Coverage Report

System "[Integrator](#)"

(c) chethan.cu@gmail.com

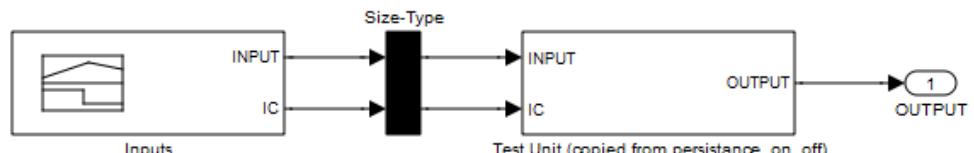
	Local			Cumulative		
	Covered	Unreachable	Uncovered	Covered	Unreachable	Uncovered
Subsystems	0	0	0	--	0	0
Branches	3	0	0	100%	3	0
States	0	0	0	--	0	0
CSEPT	0	0	0	--	0	0
Condition Actions	0	0	0	--	0	0
Transition Actions	0	0	0	--	0	0
Decisions	0	0	0	--	0	0
Conditions	0	0	0	--	0	0
MC/DC	0	0	0	--	0	0
Boundaries	1	0	0	100%	1	0
Lookup Targets	0	0	0	--	0	0
User-Defined Targets	0	0	0	--	0	0
Assertion Violations	0	0	0	--	0	0
Total	4	0	0	100%	4	0
						100%

Target	Type	Status	Test	Step
Discrete-Time Integrator%b1	Branches	covered	2	3
Discrete-Time Integrator%b2	Branches	covered	1	3
Discrete-Time Integrator%b3	Branches	covered	1	1
In1=0.0	Boundaries	covered	1	3

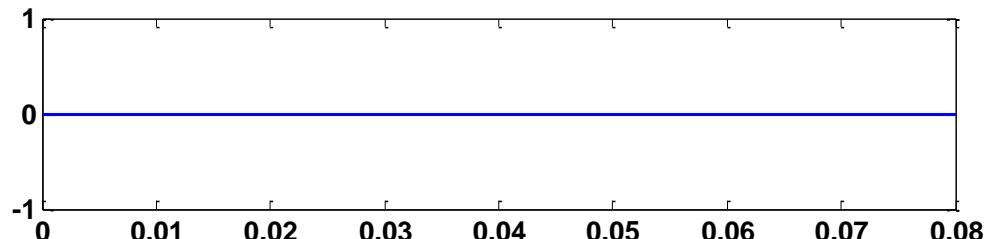
A two frame test gives 100% coverage. We will never catch the error reported earlier!!

Reactis Test Case

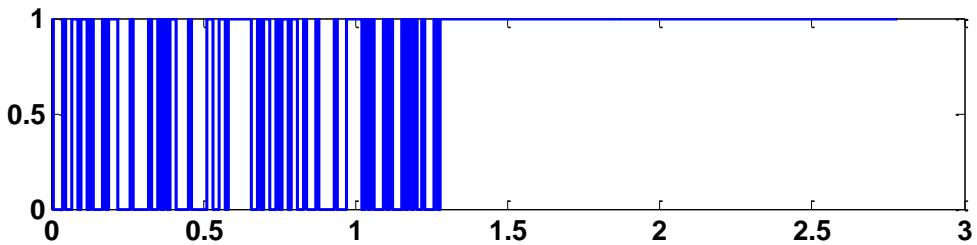
Reactis generates two tests one with 0 and the other with very fast toggles. This provides the coverage. But it will not be able to catch the error defined in “Mistakes” section.



Test Case 1



Test Case 2



(c) chethan.cu@gmail.com

Reactis Report

(c) chethan.cu@gmail.com

System "[PersistenceOnOff](#)"

	Local			Cumulative		
	Covered	Unreachable	Uncovered	Covered	Unreachable	Uncovered
Subsystems	0	0	0	--	0	0
Branches	16	0	0	100%	18	0
States	0	0	0			--
CSEPT	0	0	0			--
Condition Actions	0	0	0			--
Transition Actions	0	0	0	--	0	0
Decisions	2	0	0	100%	2	0
Conditions	2	0	0	100%	2	0
MC/DC	1	0	0	100%	1	0
Boundaries	4	0	0	100%	4	0
Lookup Targets	0	0	0	--	0	0
User-Defined Targets	1	0	0	100%	1	0
Assertion Violations	0	0	0	--	0	0
Total	26	0	0	100%	28	0

We get 100% coverage!!

Child	Subsystems	Branches	States	CSEPT	Condition Actions	Transition Actions	Decisions	Conditions	MC/DC	Boundaries	Lookup Targets	User-Defined Targets	Assertion Violations	Total
Unit Delay External IC	--	(2) 100%	--	--	--	--	--	--	--	--	--	--	--	(2) 100%

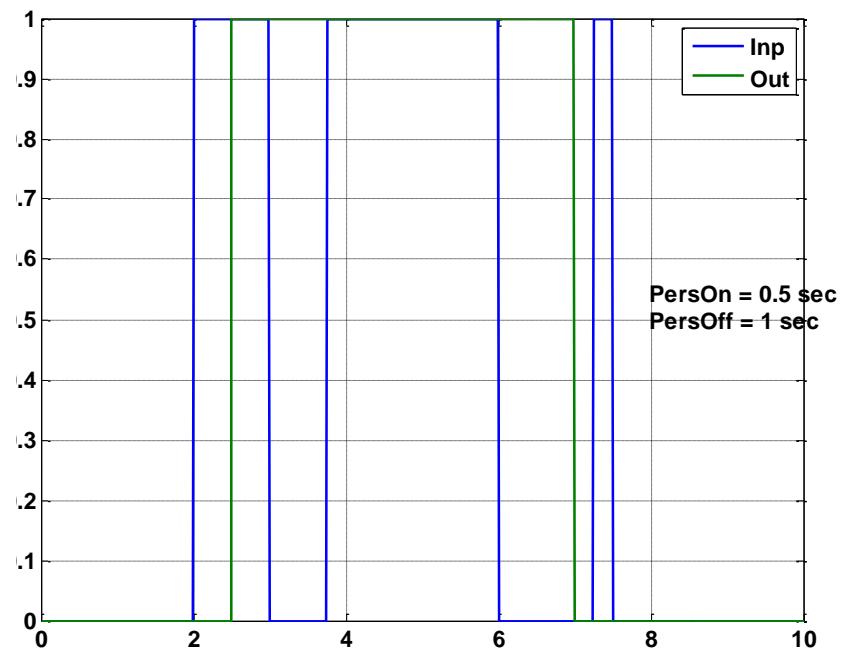
New Metrics for Persistence

#	Discrete Metric	Continuous Metric
1	IC is tested for TRUE value	NA
2	IC is tested for FALSE value	NA
3	Input has a TRUE pulse whose width is less than PersOn	$\text{abs}(\text{PersOn}/2 - \text{min. TRUE pulse Width})$
4	Input has a TRUE pulse whose width is greater than PersOn	$\text{abs}(\text{PersOn} - \text{max. TRUE pulse Width})$
5	Input has a FALSE pulse whose width is less than PersOff	$\text{abs}(\text{PersOff}/2 - \text{min. FALSE pulse Width})$
6	Input has a FALSE pulse whose width is greater than PersOff	$\text{abs}(\text{PersOn}/2 - \text{max. FALSE pulse Width})$

(c) chethan.cu@gmail.com

New Metric - Persistence

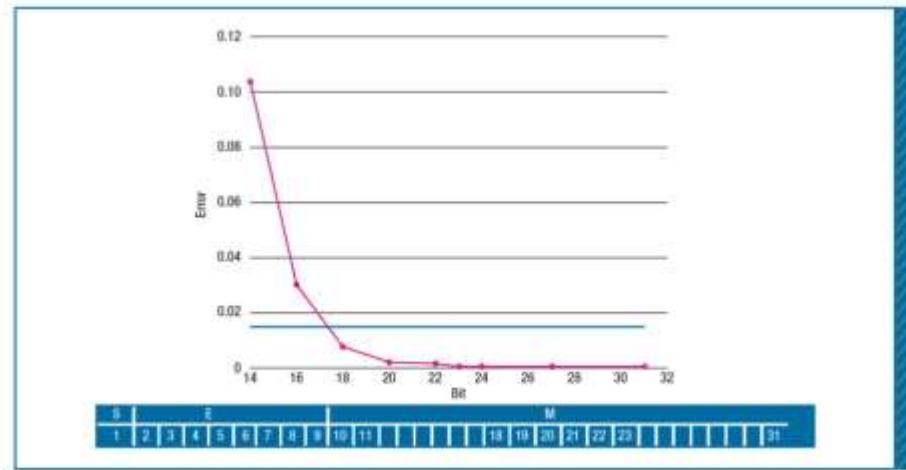
Metric	Count
IC tested for TRUE	0
IC tested for FALSE	1
Input has a TRUE pulse whose width is less than PersOn	1
Input has a TRUE pulse whose width is greater than PersOn	1
Input has a FALSE pulse whose width is lesser than PersOff	1
Input has a FALSE pulse whose width is greater than PersOff	1



(c) chethan.cu@gmail.com

Filter Coverage Metrics

- Filters can have the new coverage metric.
- The test case should be able to find the small errors injected into the mutant filter.
- The input signal is passed through the filter and the mutated filter. If the input is capable of bringing out the error then the test case is good and the filter is covered!!
- Error can be 1 bit toggle in a floating point representation of the filter coefficient

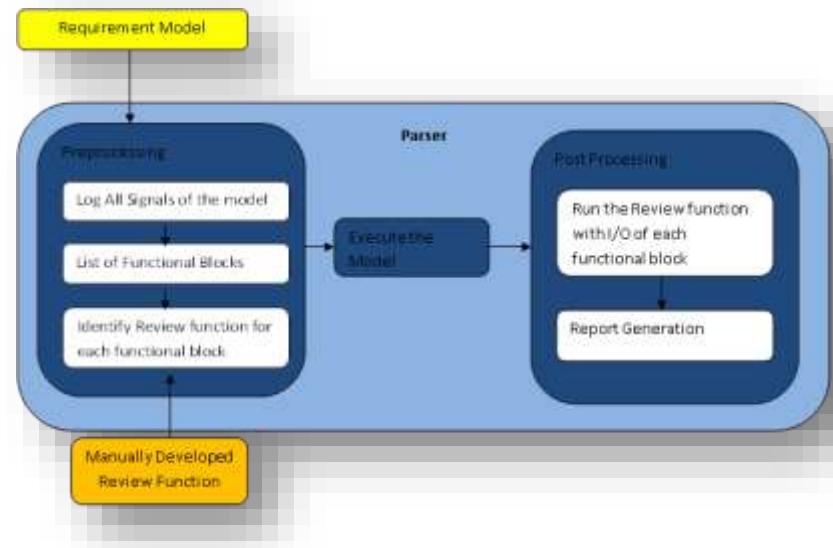


Error in the filter output as the specific bit is changed is seen in figure. Error in bit 18 is good enough to bring out the error

Jeppu, Y., "Flight Control Software: The Mistakes We Made and the Lessons We Learnt," Software, IEEE , vol.PP, no.99, pp.1,1, 0, doi: 10.1109/MS.2013.42

Autoreview Tool

- The coverage metrics are a part of a new tool today which automatically reports the functional coverage based on the metrics defined. We have been able to define metrics for more than 50 odd blocks used in the control system in this manner.
- This is being used for our second project these days.



Atit Mishra, Manjunatha Rao, Chethan CU, Vanishree Rao, Yogananda Jeppu, and Nagaraj Murthy. 2013. An auto-review tool for model-based testing of safety-critical systems. In Proceedings of the 2013 International Workshop on Joining AcadeMiA and Industry Contributions to testing Automation (JAMAICA 2013). ACM, New York, NY, USA, pp 47-52.

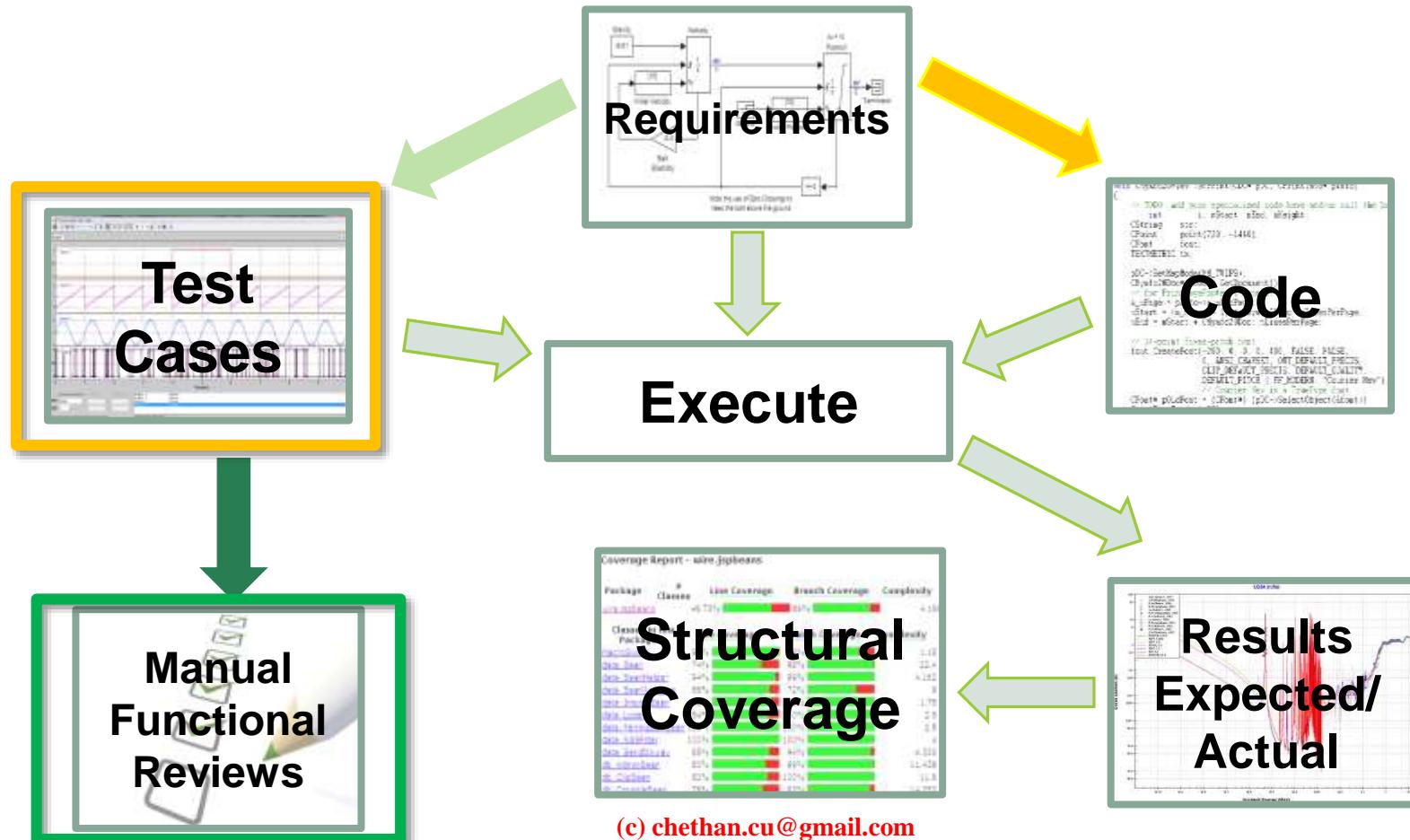
Test Methods

The screenshot shows a logic simulation environment with the following components:

- Circuit Diagram:** A logic circuit with various gates (AND, OR, NOT) and connections. Inputs include `Q1, Q2, Q3, Q4, Q5` and `Q1, Q2, Q3, Q4, Q5`. Outputs include `Q1, Q2, Q3, Q4, Q5` and `Q1, Q2, Q3, Q4, Q5`.
- Waveform Viewer:** A window showing waveforms for the inputs and outputs over time.
- Test Plan Table:** A table listing test cases with columns for Test ID, Description, Input Values, and Result.
- Test Plan Editor:** A separate window showing a hierarchical tree structure of test plans.

Test ID	Description	Input Values	Result
1	What is the effect of each eight-channel pulse corresponding to port 1?	0.42100	Yes
2	Are applications to cover the appropriate parameter ranges of tested range repeat?	0.42100	Yes
3	Are load requirements understandable and acceptable within the time specified?	0.42100	Yes
4	Are there at least two different values of port 1?	0.42100	Yes
5	Are port 1 values in the test range values of port 1?	0.42100	Yes
6	Does each repetition of a sequence take probably effect that function's interests?	0.42100	Yes
7	Is it possible to test more correctly following	0.42100	Yes

Model Based Test Process



Manual Tests

- We require to prove a safety critical system to be correct manually!
- The low level test process calls for a tester to design test case by injecting inputs at the system input point and show its effect at each and every block output
- This expected output has to be shown to be correct by hand calculation or excel computations
- The test artifacts, test cases, test procedures and results are reviewed against a checklist. These have to be kept under Configuration Control to be produced for Certification

Manual Tests

- The expected outputs are also generated using the Simulink Blocks and stored in an Excel Sheet for review
- The Code is injected with these signals using Code test tools. These tools also produce the instrumented output and coverage metrics
- Manual tests have to be requirements based as against code based or block based.
- All the tools, models have to be qualified according to the standards. The standards demand that the tool determinism be proved and documented
- This means lots and lots of work!

Tips

I prefer multiple frame tests instead of 1 frame tests at a low level. It is possible to generate such test cases. Test a filter block in the first 2 frames, then loop for 20 frames without monitoring output, check frames 21 and 22 loop again till steady state reached and check say frame 100.

Tips

It is very important that you test functionality in a requirement provided as a model. Do generate test cases to test the $1/z$ delay blocks and the constant blocks for a filter. Make your test cases test the filter algorithm instead. This is important.

Automated Tests

- A collection of Manual Test cases can be executed on target in a batch mode
- In such cases the pass/fail criteria have to be defined beforehand
- Normally test cases are executed on a simulator on the PC and later cleared for execution on the actual flight computer board in an automated manner
- V&V groups have developed methods to automate the execution which are proprietary to the company
- However, all automated test case results have to be reviewed or should be reviewable for Certification

Generating Automated Tests

- Several tools are available or have been developed in house by the V&V groups to generate test cases automatically
- This saves a lot of effort, but it is very important that “if the test cases and results (outputs) are not verifiable (manually) then the tool has to be qualified”
- A lot of effort and money is spent in these automated tools. Companies feel that it makes a business sense to qualify the tool and use it than to make manual test cases.

Random Test Cases

- One of the methods used by the tools is to generate test case randomly
- The code/block coverage metrics are monitored for each test case
- A selection is done at the end of a set of test cases to optimally select a subset of tests which give maximum coverage
- This has been successfully utilized to test the Mode Transition Logic (MTL) for the Indian SARAS aircraft. A set of 100 test cases generated randomly could cover the complete MTL
- We have used random testing for flight control system elements and monitored coverage functionally. It works very well.

Techniques for Random Tests

- Control Systems cannot be checked by injecting random signals as the filters consider these as noise and reject them. One method is to inject sinusoidal waveforms with their parameters – Frequency, Amplitude, Bias and Phase selected randomly.
- Another method that can be used is to select these parameters with a probability. 90% of the time the aircraft does maneuvers in the frequency band 1-3 Hz. 10% of the time it can do some high frequency large amplitude maneuvers. We can select the input parameters to mimic these realistic situations
- You will find some random text case generators here
 - <http://www.mathworks.com/matlabcentral/fileexchange/44168-random-signal-generation>

Coverage Metrics

- Random Tests rely on coverage metrics for selection
- Block coverage has been discussed earlier. Simulink gives the coverage metrics automatically. It is possible to define coverage metrics for specialized blocks and monitor them during test case generation.
- It is very important to take in the code coverage metrics also when generating test case
- Test cases should give 100% coverage for functionality and code. If not, these have to be justified as unreachable and documented
- We use the new functional coverage metrics for random tests with excellent results.

Orthogonal Arrays

- It is always possible to look at the test cases as parameters to a process and the various amplitude as levels.
 - Instead of looking at changing one parameter and keeping the other constant, it is possible to look at pair wise combinations
 - Orthogonal Arrays can be used successfully to reduce test cases
 - A freeware software called “allpairs” has been used to reduce test cases in the SARAS and LCA programs while maintaining the rigor of testing
-
- Matlab has orthogonal array generation routines. One is the hadamard() function. There are two files contributed by users which can generate OA.
 - <http://www.mathworks.in/matlabcentral/fileexchange/47218-orthogonal-array>
 - <http://www.mathworks.com/matlabcentral/fileexchange/46783-generate-oa-m>

An L8 Array

- An L8 array can be used to test 7 input parameters with two levels each
- The Two levels could be True or False and the 7 inputs to a logic circuit
- Any two cols show all combinations of (1,1), (1,2), (2,1) and (2,2)

Experiment Number	Column						
	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	1	1	2	2	2	2
3	1	2	2	1	1	2	2
4	1	2	2	2	2	1	1
5	2	1	2	1	2	1	2
6	2	1	2	2	1	2	1
7	2	2	1	1	2	2	1
8	2	2	1	2	1	1	2

Orthogonal Cases for SARAS

- In the Indian SARAS program system tests were carried out for Altitude, Speed, Autopilot Up/Down, Autopilot Soft Ride On/Off cases
- 4 Altitude and 4 Speed cases had to be tested
- “Allpairs” software was used to generate 13 test cases for each autopilot mode, <http://www.satisfice.com/tools.shtml>
- These are covering arrays and not orthogonal arrays but they get the job done!
- The Flight Envelope coverage was checked in a dynamic situation and found to be adequate
- The complete set of test cases was automated and executed on the system test rig

Covering Arrays

ACTS from National Institute of Standards and Technology, NIST, U.S. Department of Commerce can generate covering arrays for you. An excellent tool for testing optimally.

csrc.nist.gov/groups/SNS/acts/documents/comparison-report.html

The screenshot shows the ACTS software interface. On the left, the 'System View' pane shows a hierarchical tree structure with nodes such as [Root Node], [SYSTEM-Test], Input1, Input2, Input3, and Relations. Input1 has children true and false. Input2 has children 10, 20, and 30. Input3 has children On, Off, DontCare, and Relations. In the center, the 'Test Result' tab displays a table with 9 rows and 3 columns. The columns are labeled INPUT1, INPUT2, and INPUT3. The data is as follows:

	INPUT1	INPUT2	INPUT3
1	false	10	On
2	true	10	Off
3	false	10	DontCare
4	true	20	On
5	false	20	Off
6	true	20	DontCare
7	false	30	On
8	true	30	Off
9	false	30	DontCare

An 'About ACTS' dialog box is open at the bottom right, containing the following information:

ACTS Beta Release
Version: Beta 2 - Revision 2.7

Developed by
the Automated Combinatorial Testing for Software Group
at NIST and the University of Texas at Arlington

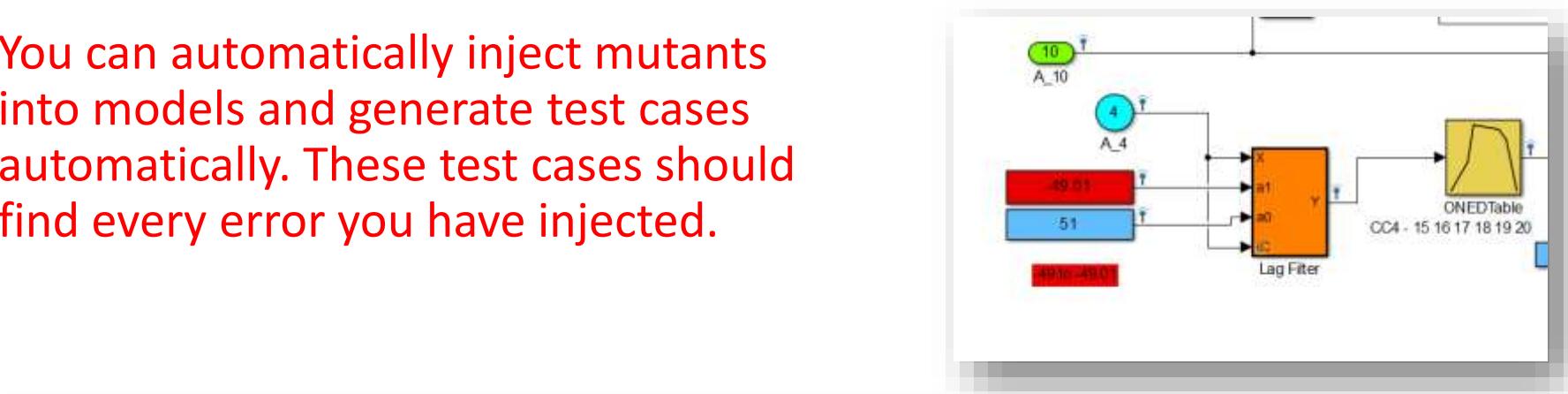
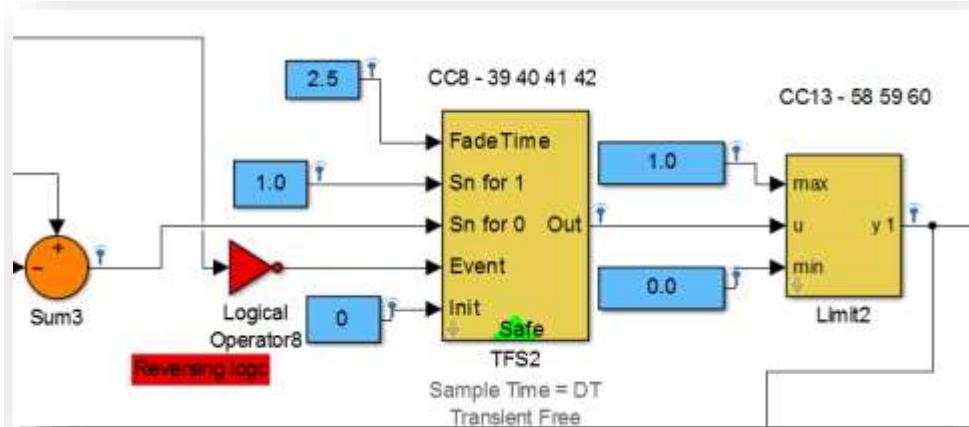
08/16/2012

Error Seeding

- A technique of Error Seeding was used successfully to design test cases for the LCA controller
- The Model for the controller was seeded with errors for the block under test
 - Only 1 error was introduced in a Delta Model
- The efficacy of the test case to bring out this error was determined by ensuring that the output error was very much above the pass/fail threshold
 - A set of 400 odd cases were generated to test each and every block in the Model by verifying on the Delta Model
- LCA flies today without any safety critical CLAW errors!

Mutation testing

You can automatically inject mutants into models and generate test cases automatically. These test cases should find every error you have injected.



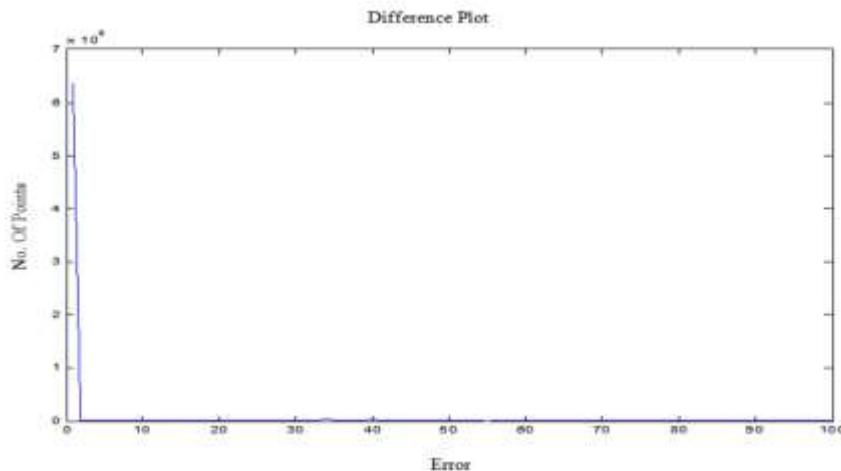
Pass/Fail Threshold - Discussion

- What should be the pass/fail threshold for an automated test?
- Altitude varies from 0 Km to 15 Km, and Mach Number varies from 0 to 2. Can they have the same threshold for pass/fail?
- What is the best way to solve this issue?
- Does the precision of my hardware effect this threshold?
- Can I catch all errors if I keep a very low threshold? Will I get spurious failures?

LCA Example

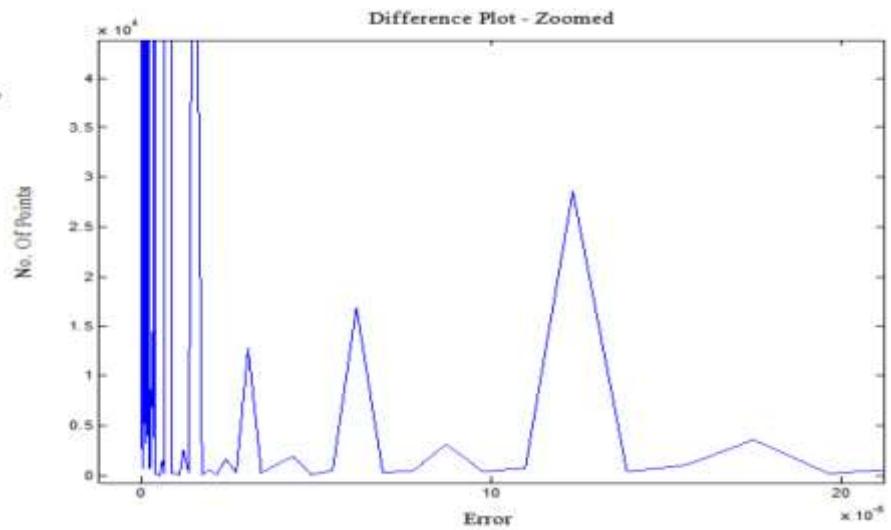
- We have found that a good threshold is to use the formula
- If the $|Output|$ signal is > 1.0 then divide the error by the signal
- If it is ≤ 1.0 then take the computed error itself
- We used a threshold of 0.0002 for the pass/fail and found it to be adequate for our processor and precision used
- This has been reported in open literature so feel free to use it!

Automated Thresholds

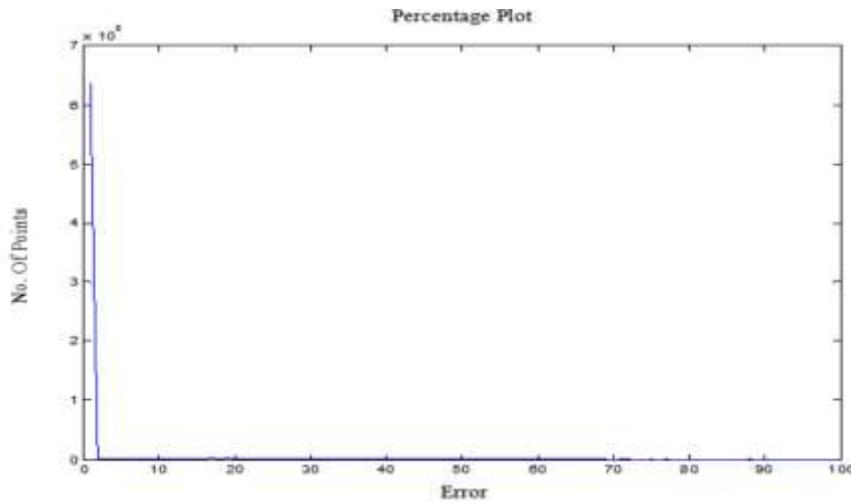


There are many points which lie below 2×10^{-3}

This is the plot of the **difference** between model and code
Total test points 650,000,000

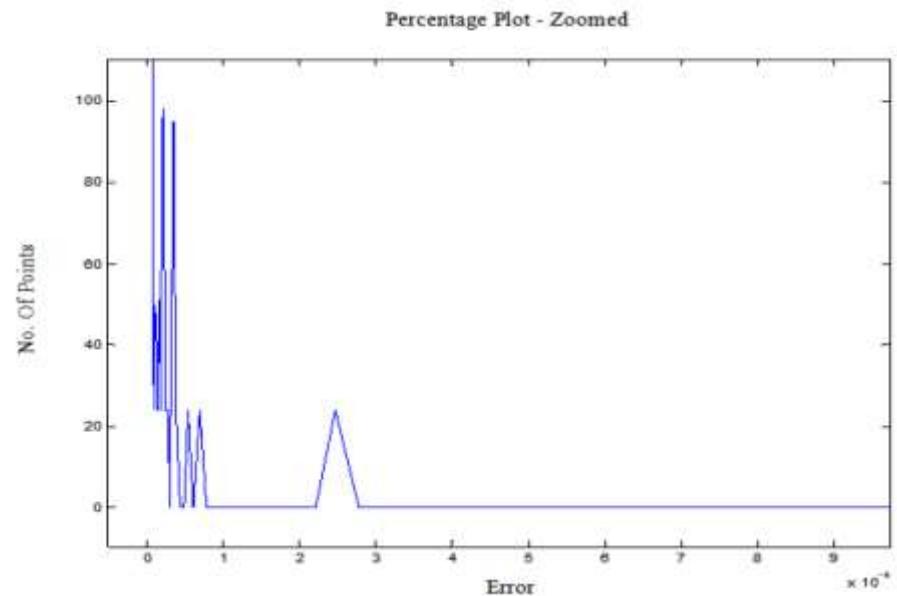


Automated Thresholds



This is the plot of the ***percentage difference*** between model and code for the same tests

There are very few points less than 2×10^{-4} points which lie below 2×10^{-4}



Tips

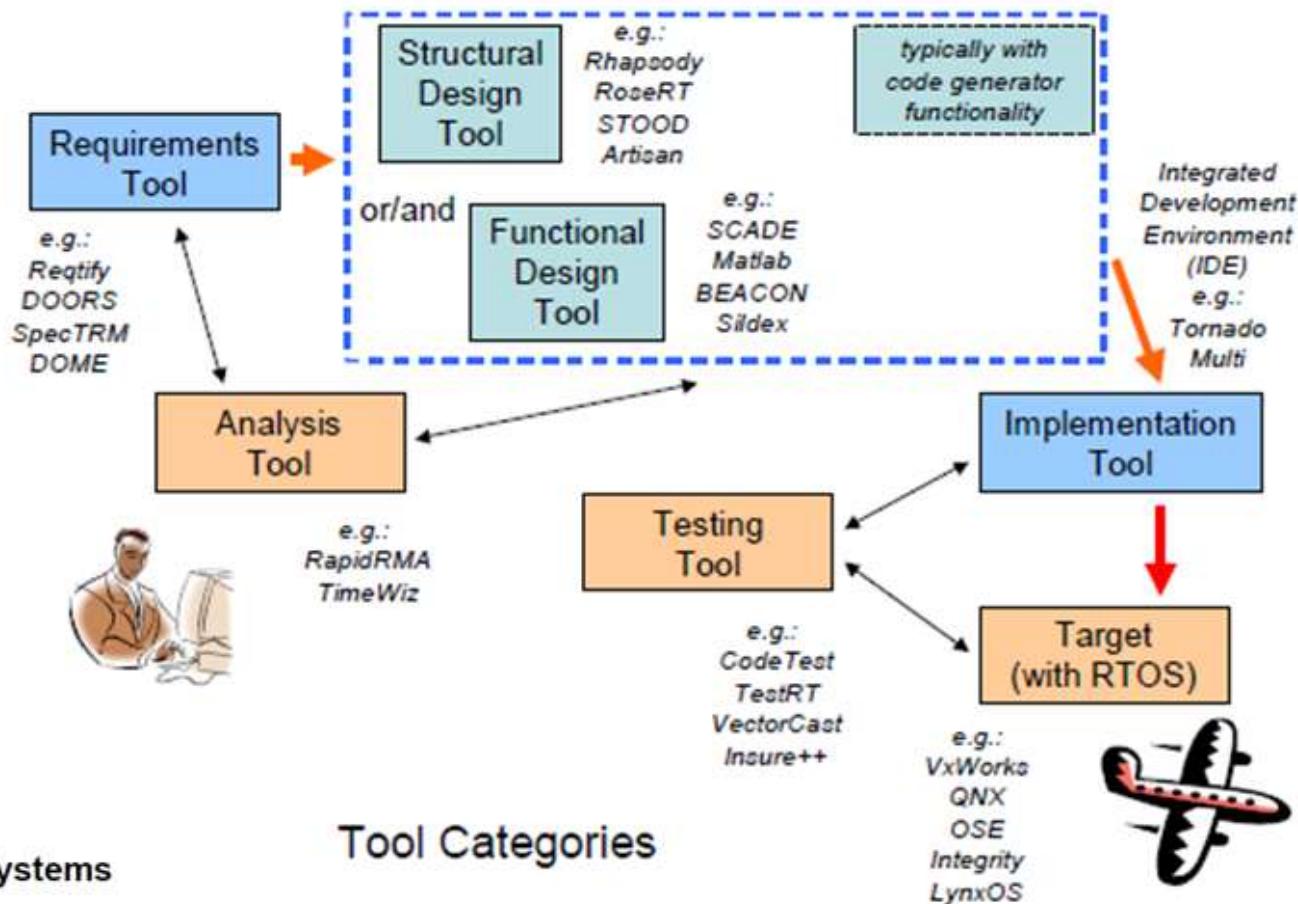
The pass fail threshold is dependent on the hardware, the precision of arithmetic used in computation. This will vary for your application. Select this with care so that it does not pass errors and does not alarm you with many failures that have to be explained each time. Define this as soon as you have your test bench ready. A little experimentation will help.

Tools Of Trade

Traditionally, a sushi knife would be made of an incredibly high-quality carbon steel, the same type used in the forging of katana, traditional Japanese swords



Tool Categories



DOT/FAA/AR-06/36

Assessment of Software
Development Tools for
Safety-Critical, Real-Time Systems

Leading Safety Critical Standards



Avionics

DO-178B (First published 1992) / DO-178C

Industrial

IEC 61508 (First published 1998)

Railway

CENELEC EN 50128 (First published 2001)

Nuclear

IEC 61513 (First published 2001)

Automotive

ISO/DIS 26262 (Draft)

Medical

IEC 62304 (First published 2006)

Process

IEC 61511 (First published 2003)

So, the experience of other sectors is invaluable
to the medical device (and automotive) industries.

MathWorks



Products & Services **Solutions**

Solutions > Automotive > IEC 61508

Automotive

IEC Certification Kit aids certification for IEC 61508 and its derivative standards including ISO 26262. It supports Embedded Coder and Polyspace code verification products.

Key products for developing IEC 61508 applications:

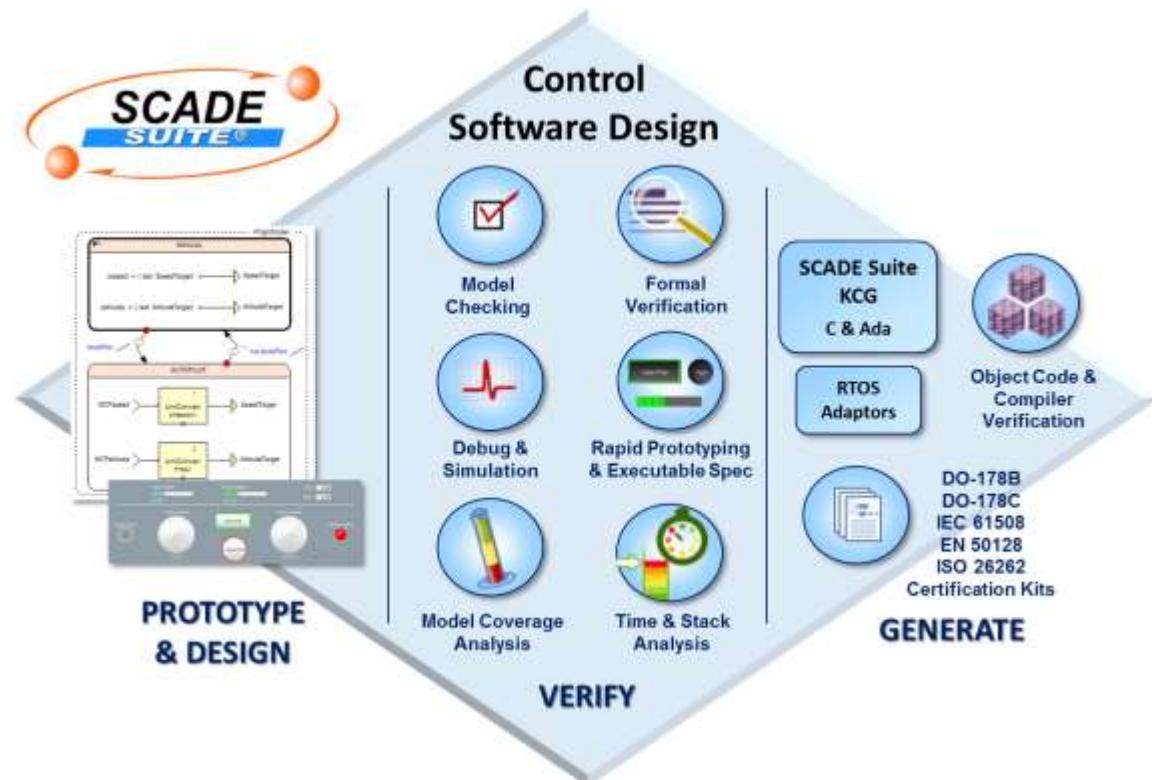
- [IEC Certification Kit](#) (for IEC 61508 and ISO 26262)
- [Simulink](#)
- [Stateflow](#)
- [Embedded Coder](#)
- [Simulink Verification and Validation](#)
- [Embedded Coder](#)
- [Polyspace code verification products](#)

www.mathworks.com

More about MathWorks support for IEC 61508:

- Press Release: [Embedded Coder Certified By TÜV SÜD Automotive GmbH](#)

SCADE



www.esterel-technologies.com/products/scade-suite/

And Many More

I have not brought out the complete list of tools

Please note that these are not my recommendation on the tool. I have known these. Some by usage and some by the vendors asking me to explore these.

All these tools have their uses like the knives. Not all knives can be used to do all the tasks like Carving, Boning, Slicing, Chopping, Dicing, Mincing, Filleting ...

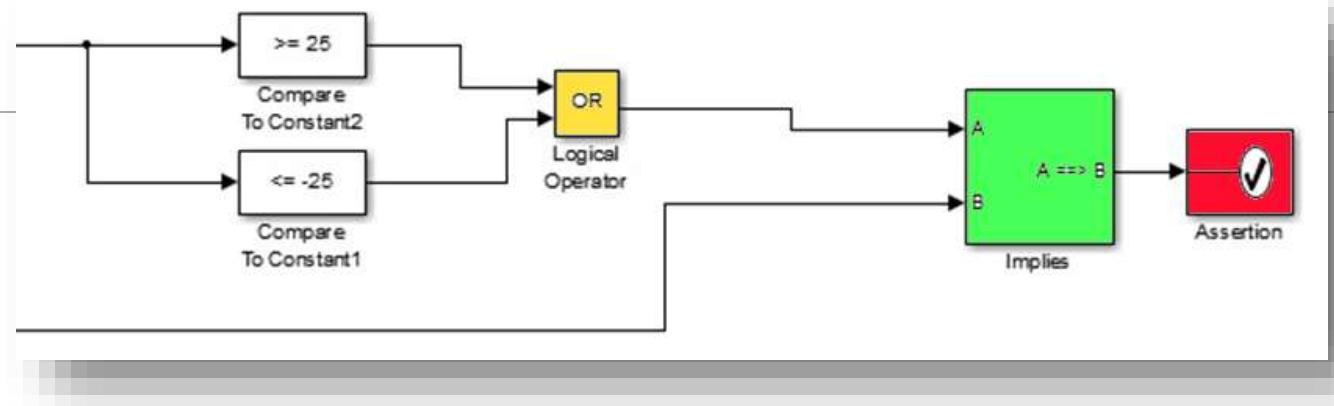
All tools are sharp and can cut you. Know your tools and their effectiveness and limitation.

Tools for safety critical systems have to be qualified. There are standards and procedures for tool qualification.

Tips

Tool qualification is an important activity in the safety critical system development. I have qualified a few tools. The limitations of the tool and their behavior under stress tests are a revelation! We have to be aware of this fact.

Formal Methods in Control



Formal Methods

In a nutshell

- Formal methods are techniques used to model complex systems as mathematical entities. (They are more precise)
- The complex system behavior is broken down into smaller units and each one of these is defined as a mathematical equations. This is the TRUTH.
- Defining systems formally allows the engineer to validate the system (mathematically correct behavior - mostly safety criteria) using other means than testing – like a proof of correctness



Why Formal Methods ?

Program testing can be used to show the presence of bugs, but never to show their absence!

Edsger W. Dijkstra

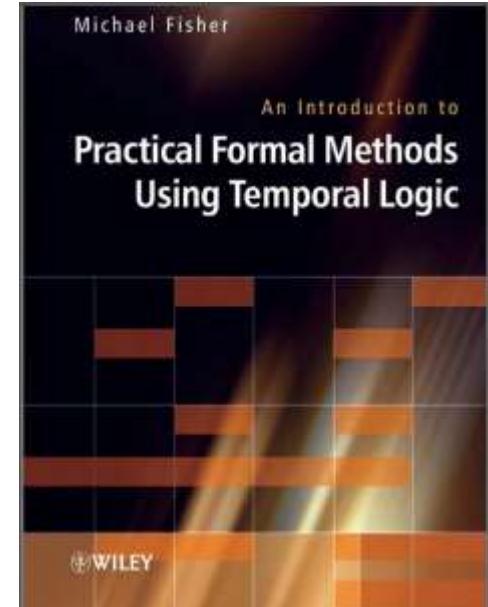
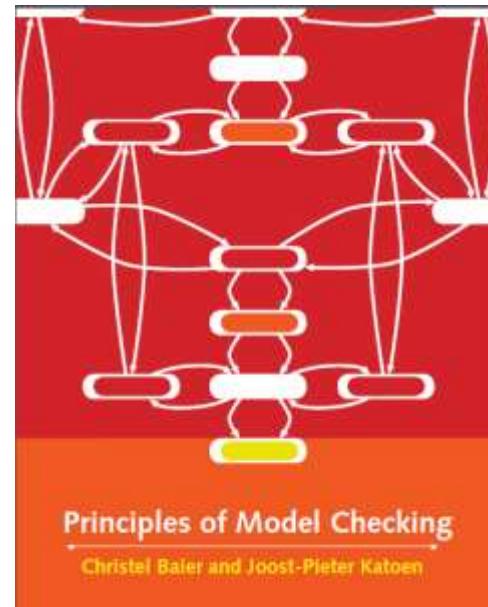
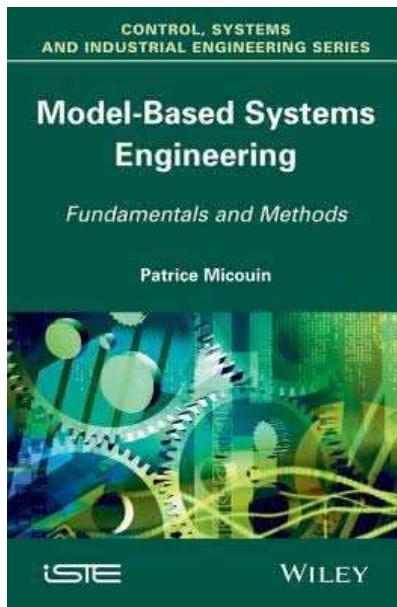
There is a lack of clarity when we use natural language

It is sometimes difficult to use language in a precise and unambiguous way without making the document wordy and difficult to read (I prefer to use a block diagram)

Several different requirements may be expressed together as a single requirement (This happens all the time even with guidelines)

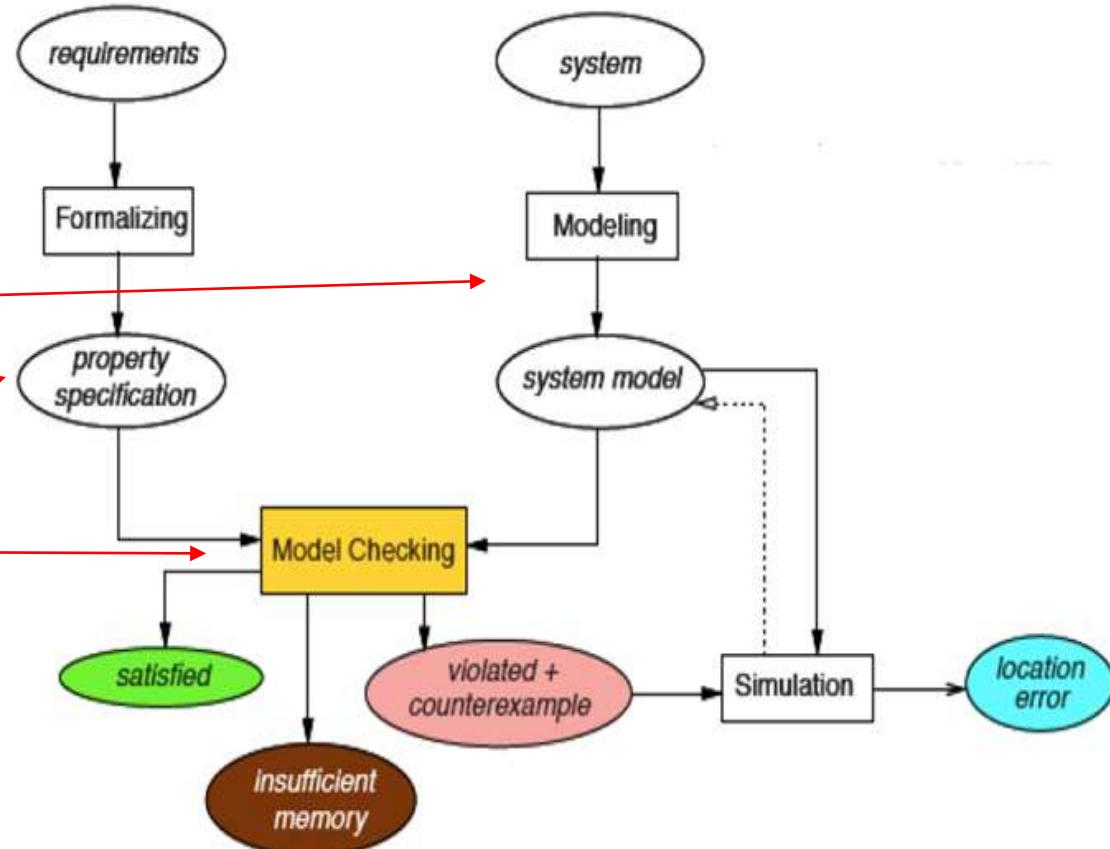
Formal methods

Three good books I have liked on this subject.



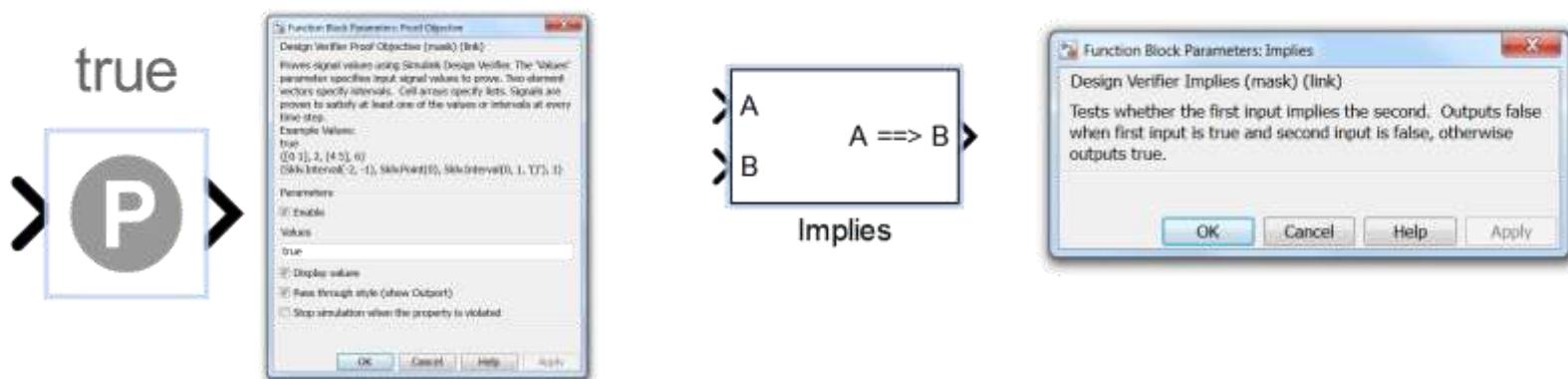
Model Checking

The examples here are using a technique of formal analysis called model checking. This consists of making a system behavior model, making a set of assertions on the system behavior and running a model checker to ensure that the behavior is consistent with the specification.



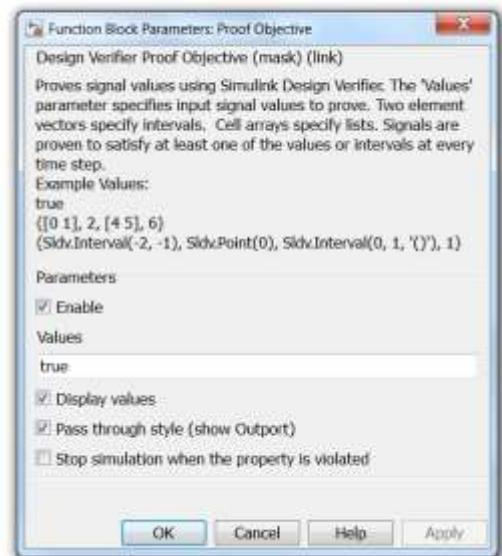
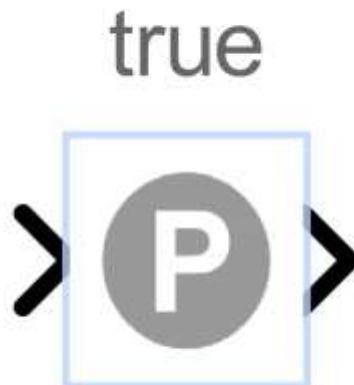
Mathworks Simulink Design Verifier

- Simulink Design Verifier uses formal methods to identify hard-to-find design errors in models without requiring extensive tests or simulation runs.
- It uses the Implies, Assertion or Proof block to indicate that the implication has to be proved.



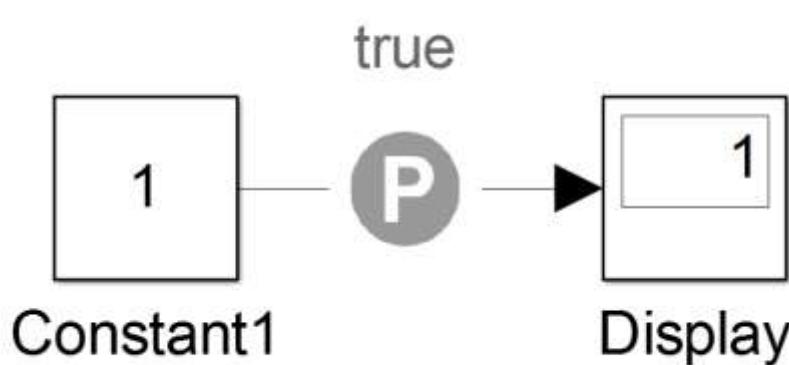
Proof Objective

- Simulink Design Verifier proof objective is used to prove a particular formulated requirement to be True
- This defines objectives that signal input must satisfy when proving model properties
- The values can be range, a list, or a simple True or False



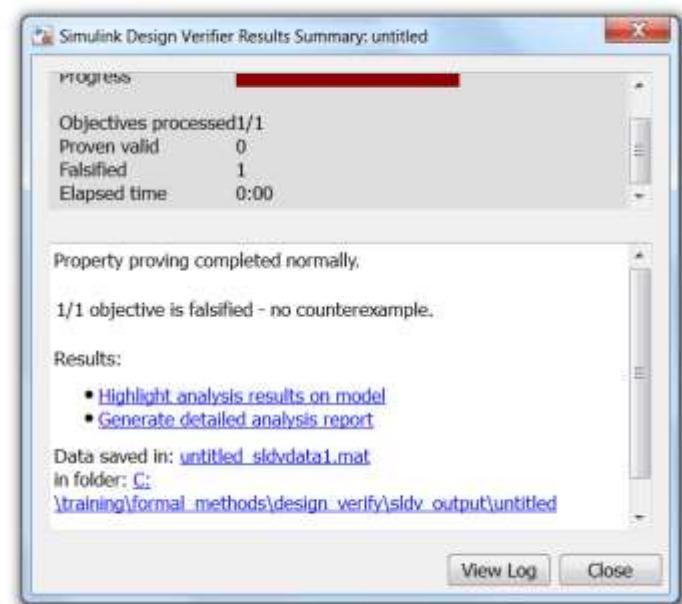
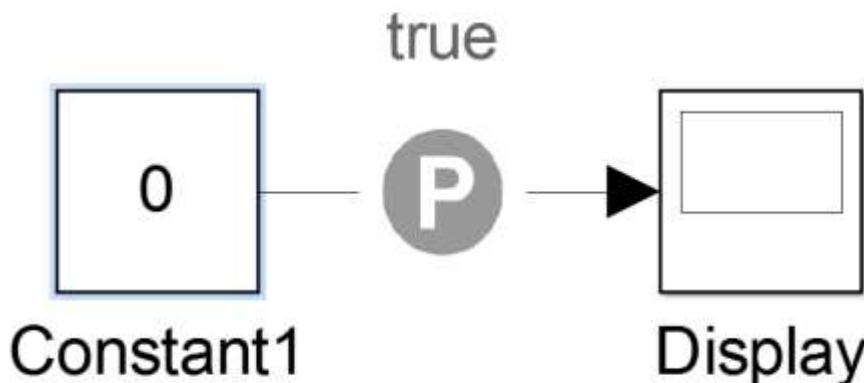
Proof Objective – Proven Valid

- An input to be True is proved correctly by the SLDV. This is simple example that brings out the functionality of the block
- The user has to provide a detailed requirement structure as Simulink block or Matlab function and the output of the requirement SHALL be TRUE always



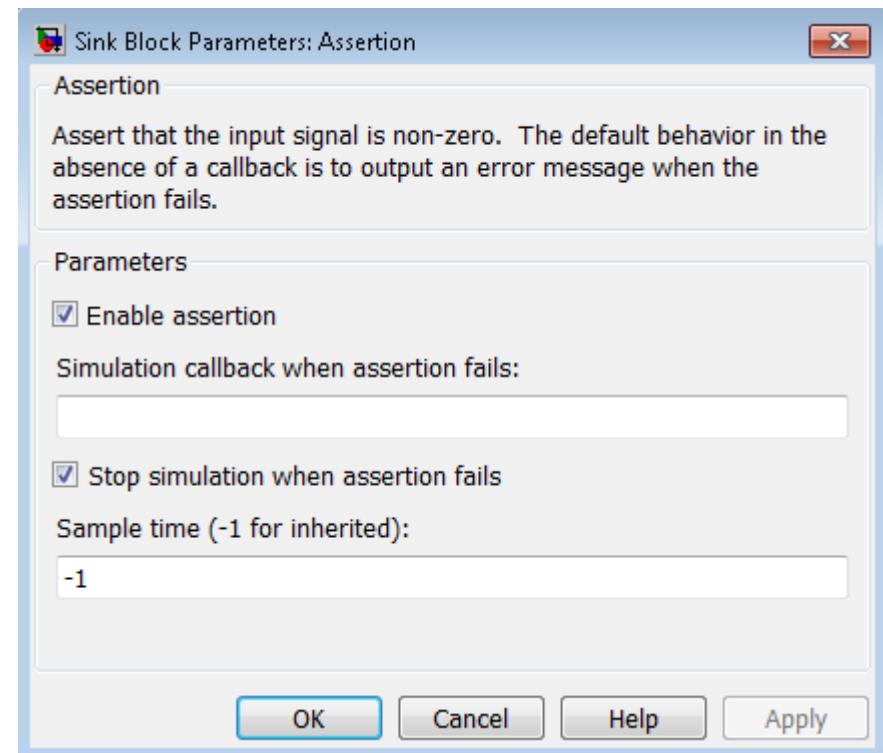
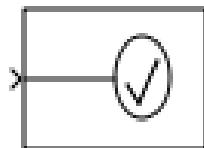
Proof Objective - False

- An input if False is proved as falsified. SLDV generates a counter example that can make the input to the Proof Objective false and indicates “falsified”
- The user need not worry about the methodology in the background



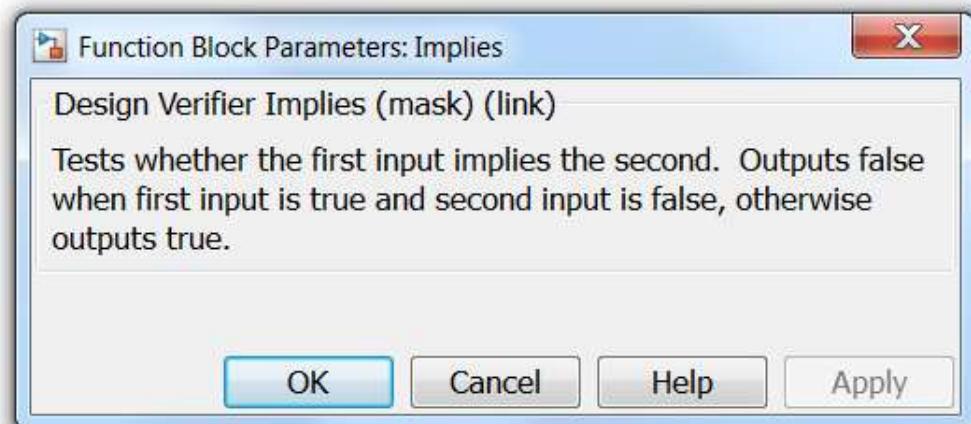
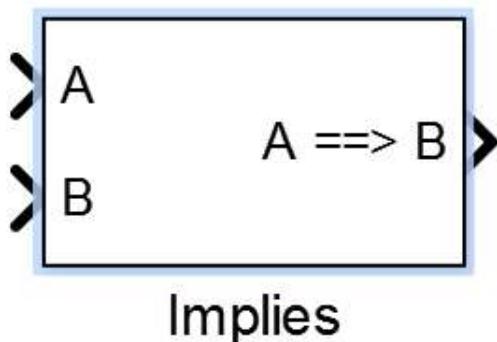
Assertion

- The Assertion block checks whether any of the elements of the input signal is zero. If all elements are nonzero, the block does nothing. If any element is zero, the block halts the simulation, by default, and displays an error message.



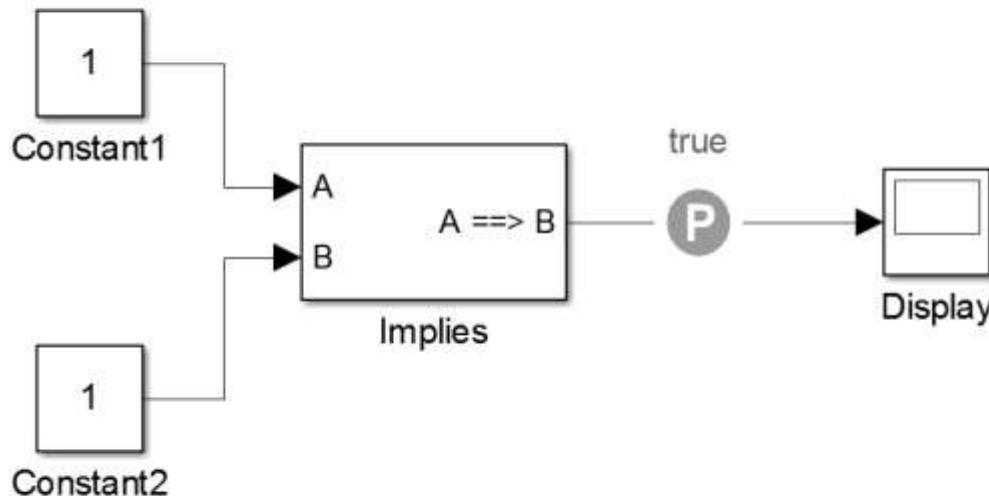
Implies Block

- This block tests whether the first input (A) TRUE implies that the second input (B) is TRUE.
- The output of this block is given to the Proof Objectivities to prove the implication.



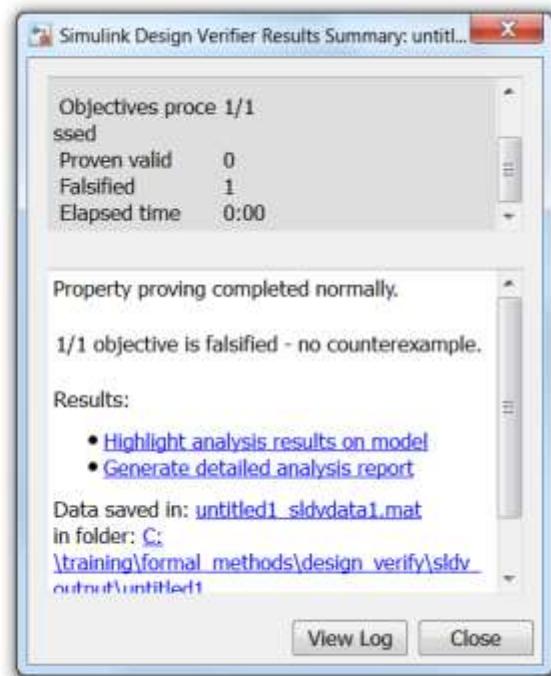
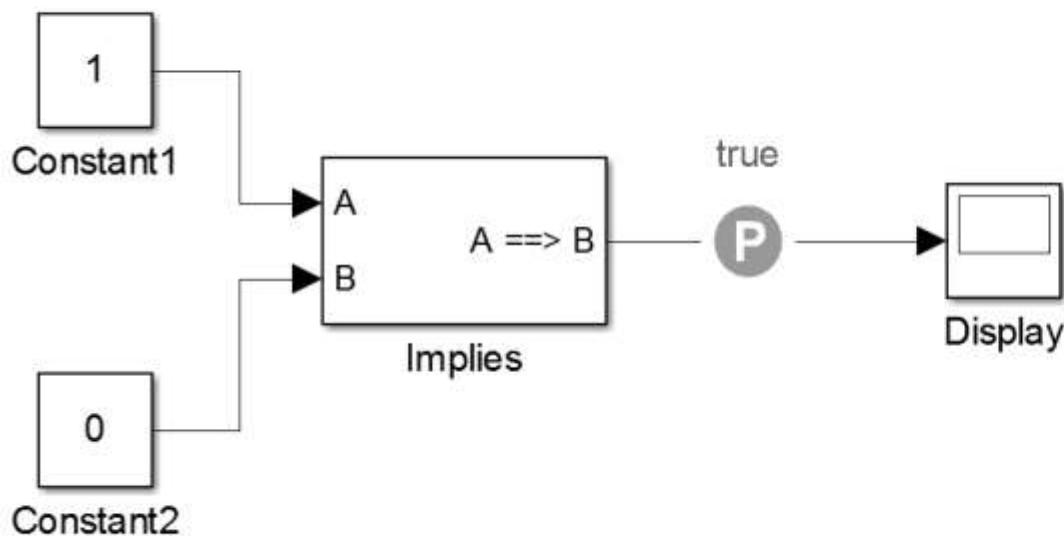
Implies Proven True

- If A is TRUE (1) and B is TRUE (1) then the proof block proves that the requirement is proven Valid



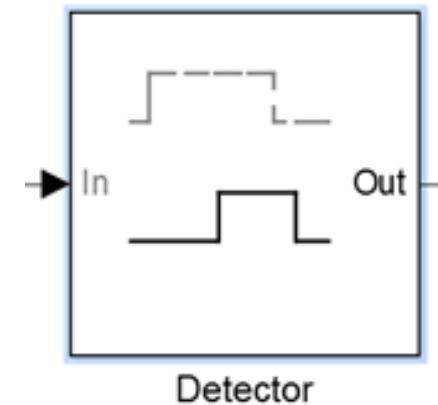
Implies Proven False

- If A is TRUE (1) and B is FALSE (0) then the proof block indicates Falsified.

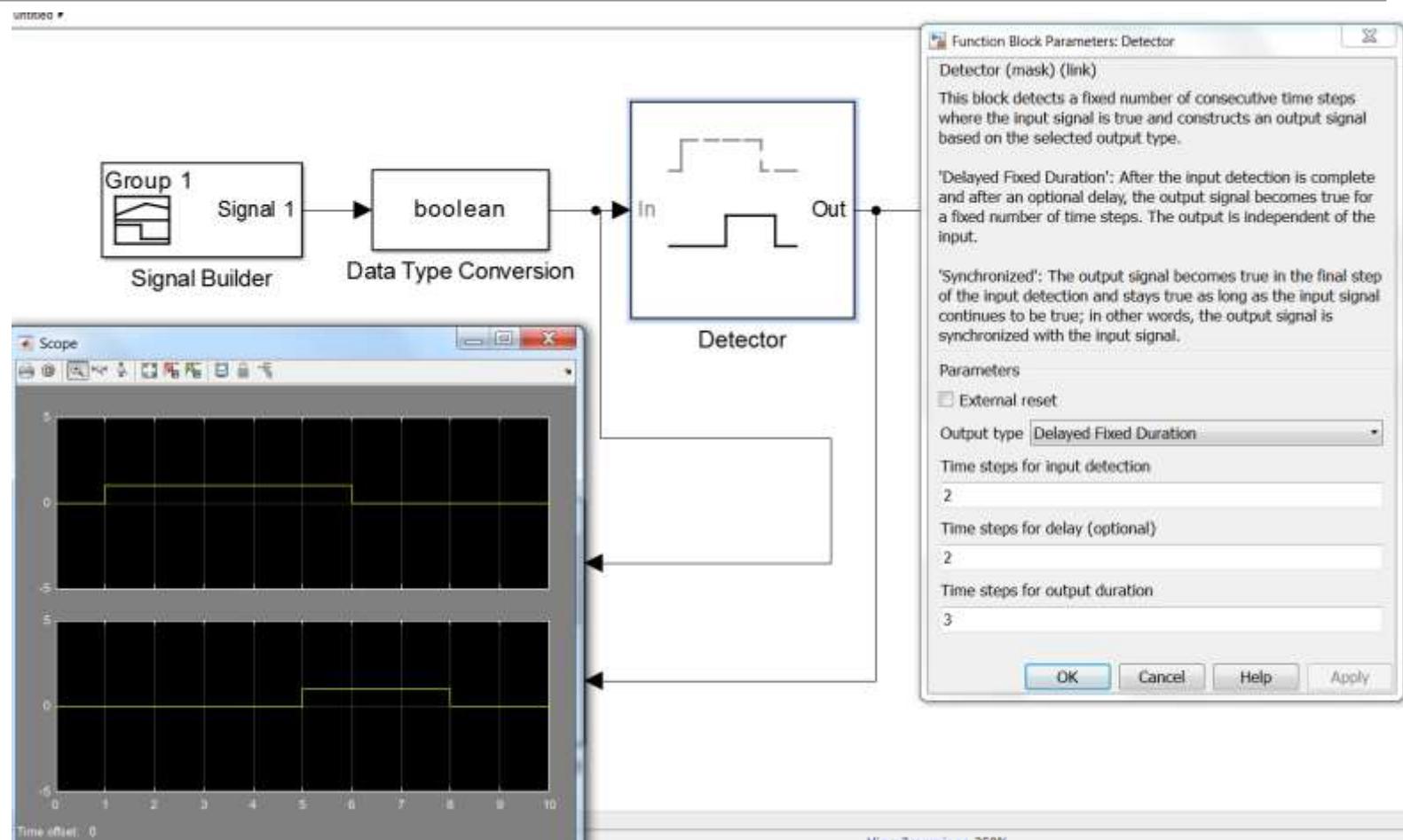


Detector

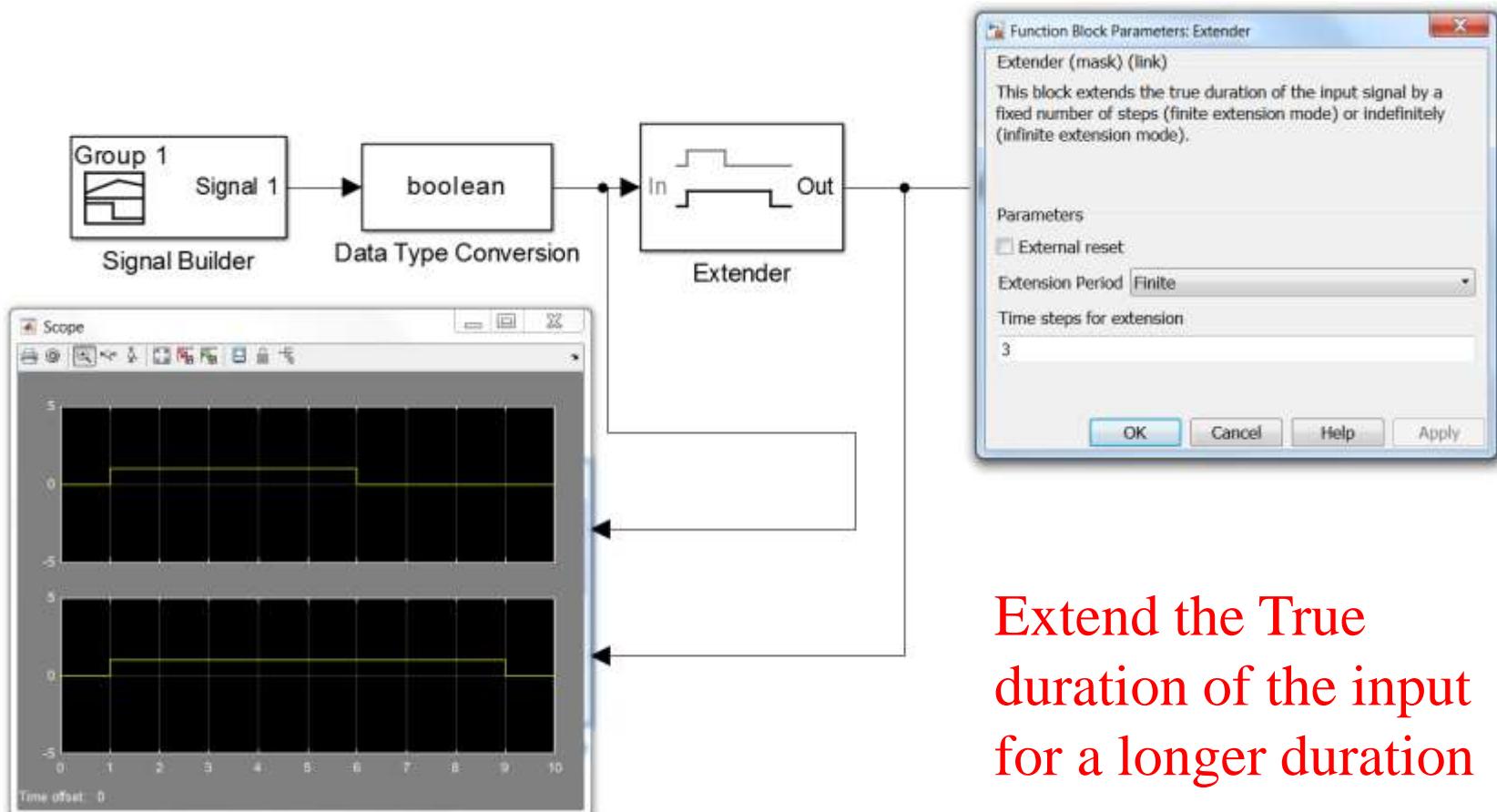
- This block detects true duration on input and constructs output true duration based on output type specified for the block
- The True duration of the input signal is the consecutive time steps or frames for which the input is True
- The input change can be detected and after a specific duration and delay the output can be set to true for a specific duration.
- The output can be synchronized to the input if required.



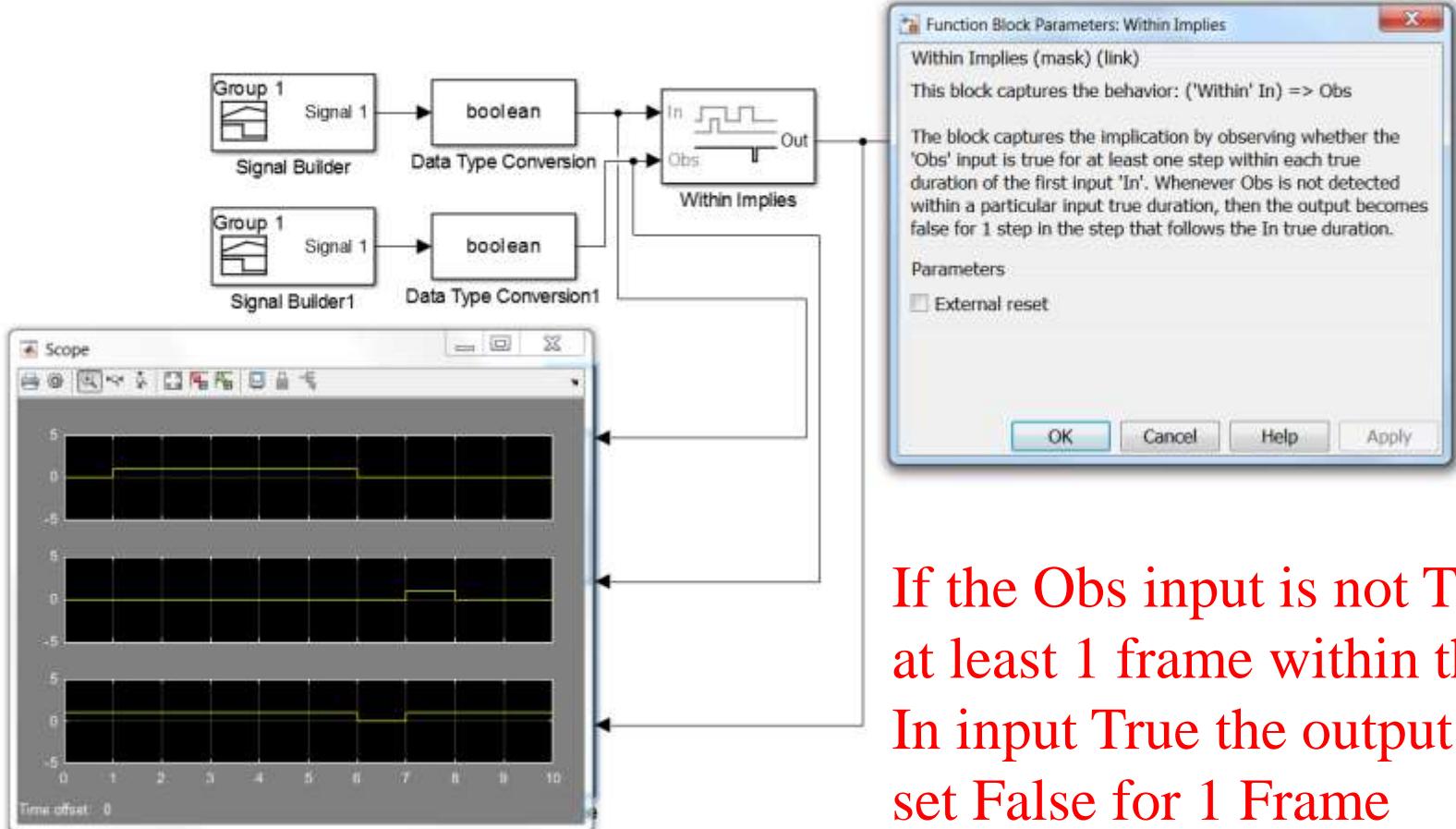
Detector



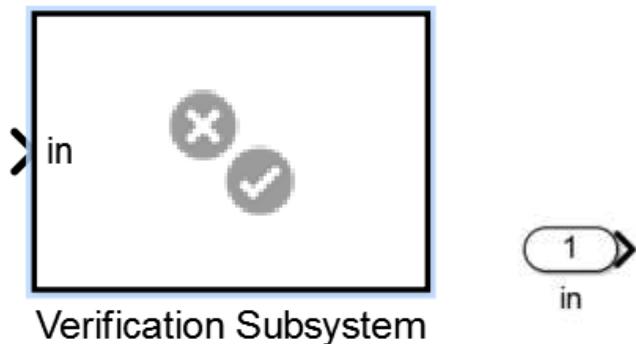
Extender



Within Implies



Verification Subsystem



Many inputs are permitted. This block can have the formal assertions for the system.

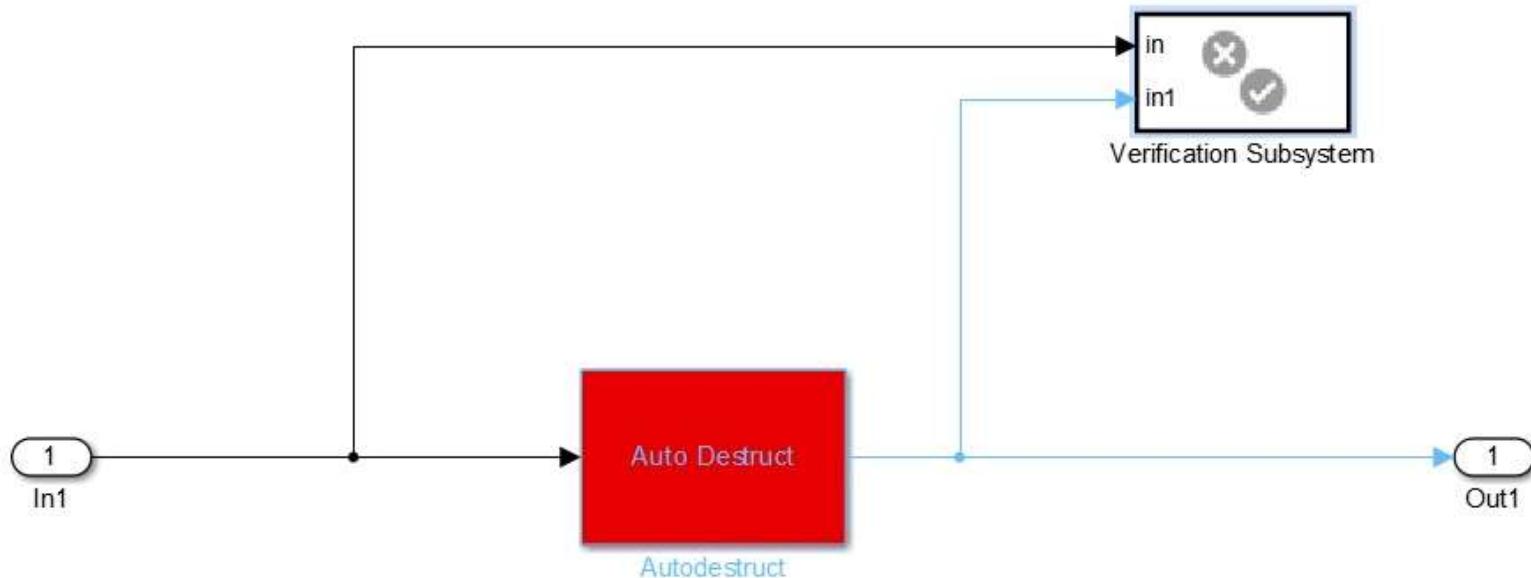
The contents of this subsystem are excluded from the code Simulink Coder generates.

To work correctly this system must continue to

- * Contain no output ports
- * Enable its 'Treat as Atomic Unit' parameter
- * Specify its 'Mask type' parameter as "VerificationSubsystem"

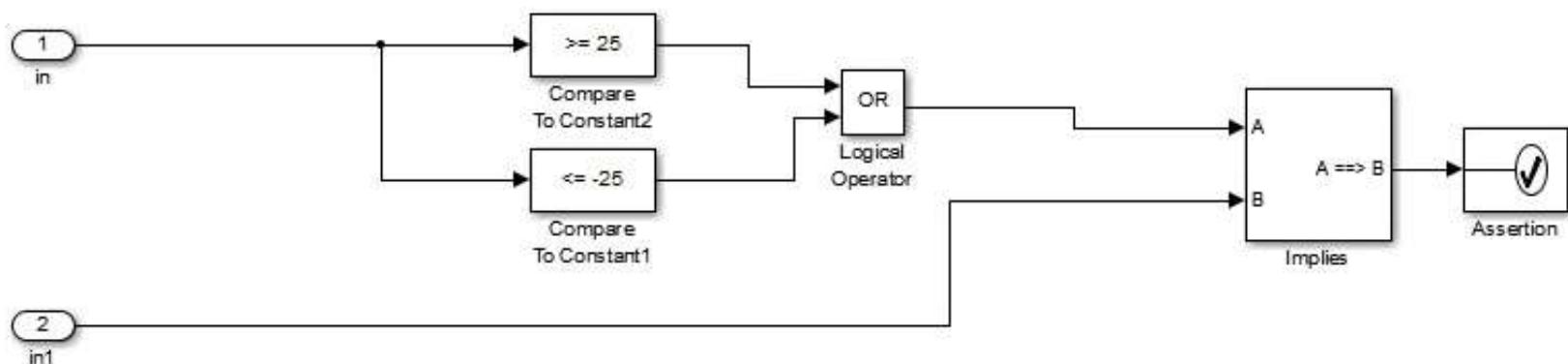
My first bug

In a missile system the autodestruct was set to True if the pitch attitude angle Theta was greater than or equal to 25 degrees. I found this bug (my first one) by doing a code review.

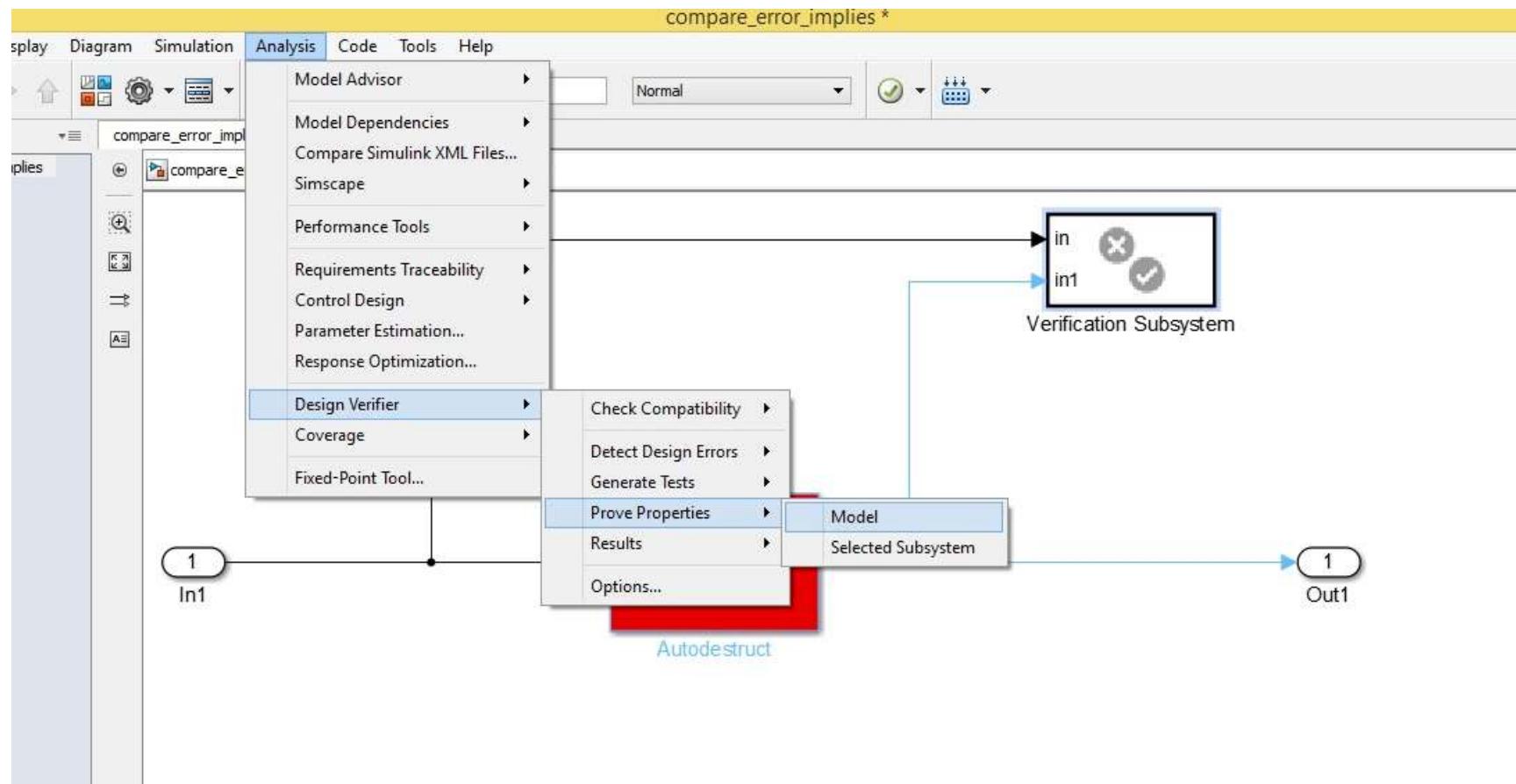


System Requirement

If the input (the pitch angle) $\geq \pm 25$ deg then autodestruct.
This is modelled in the verification subsystem as follows. Input In is theta and input In1 is autodestruct flag. If In $\geq +25$ OR In ≤ -25 implies that the autodestruct signal In1 SHALL be True!



Design Verifier Prove Properties



A Counter Example

Objectives Falsified with Counterexamples

#	Type	Model Item	Description	Analysis Time (sec)	Counterexample
1	Assert	Verification Subsystem/Assertion	Assert	1	1

Chapter 4. Properties

Table of Contents

[Verification Subsystem/Assertion](#)

Verification Subsystem/Assertion

Summary

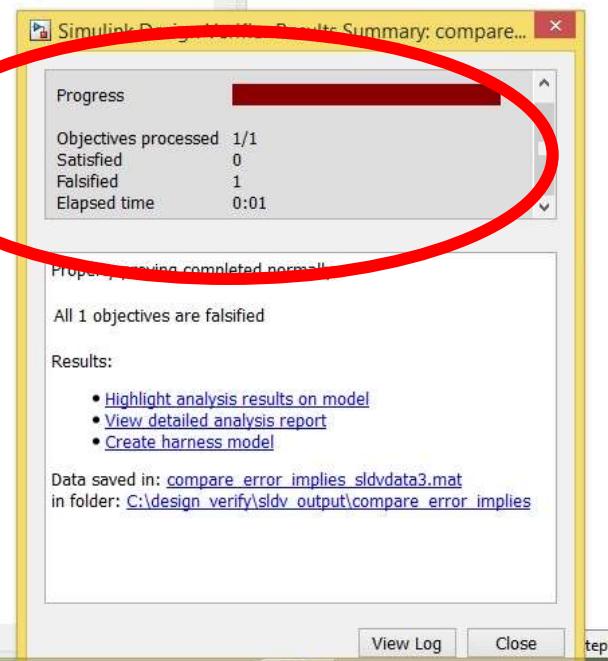
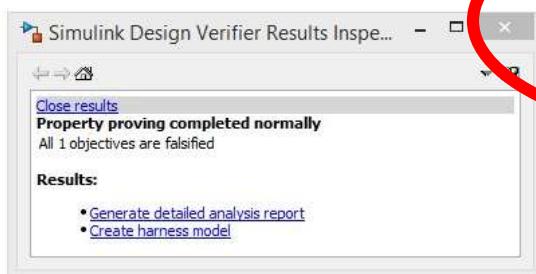
Model Item: [Verification Subsystem/Assertion](#)

Property: Assert
Status: Falsified

Counterexample

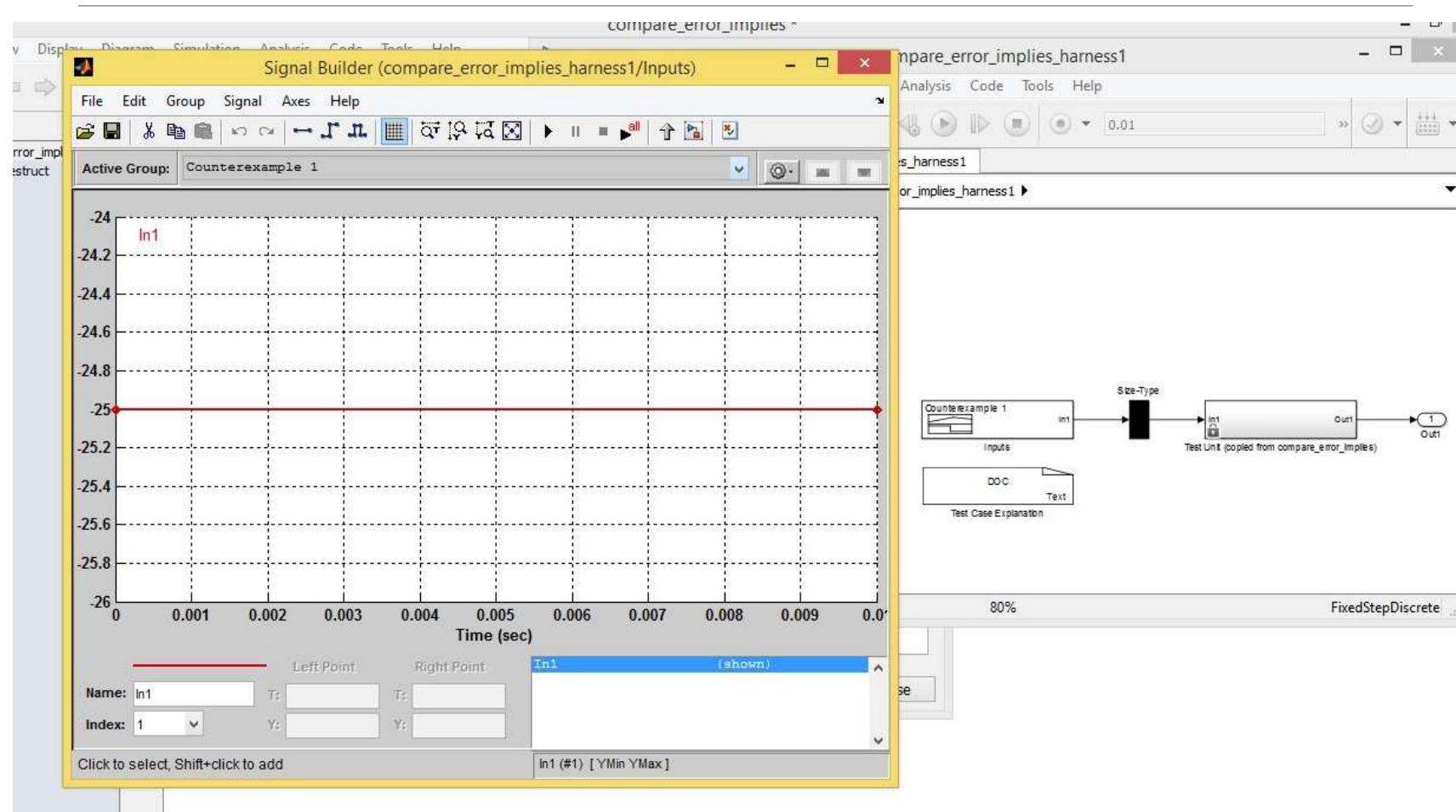
Time	0
Step	1
In1	-25

Done



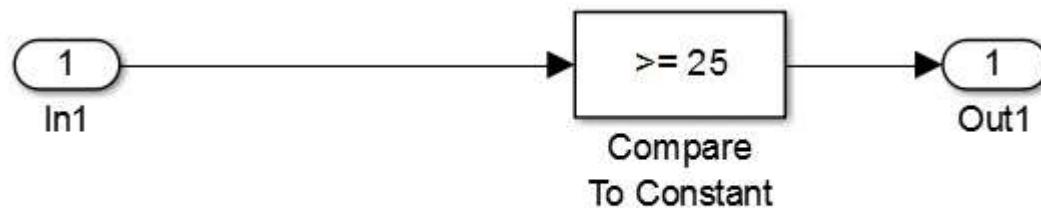
The verification is Falsified indicating modelling error!!

Generated Test Case

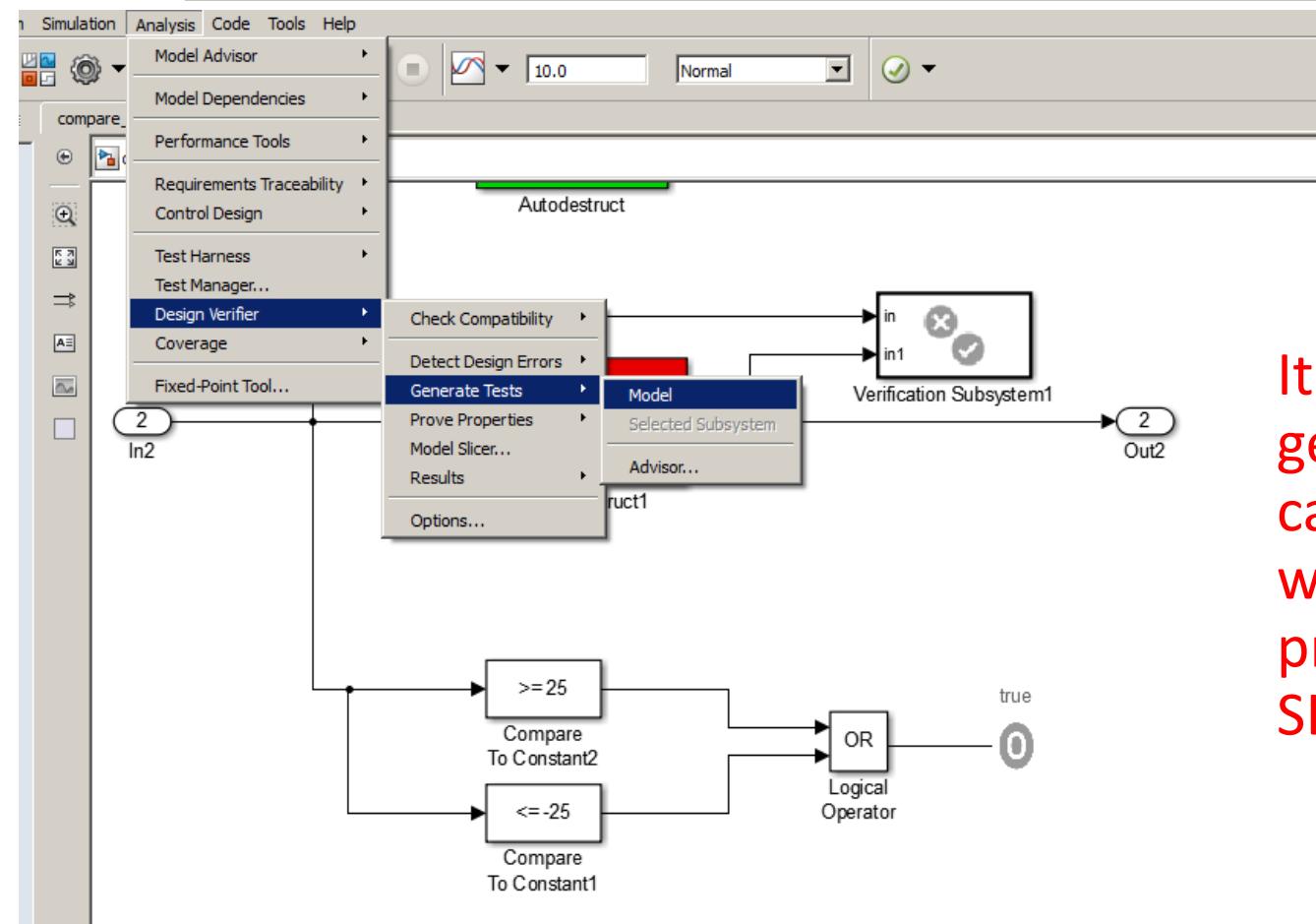


The Error

The ABS is missing in the design block. This was an actual error which caused a delay in the program. The on board computer was sent back to the laboratory for inclusion of the abs sign in the code, re-tested and the missile fired.

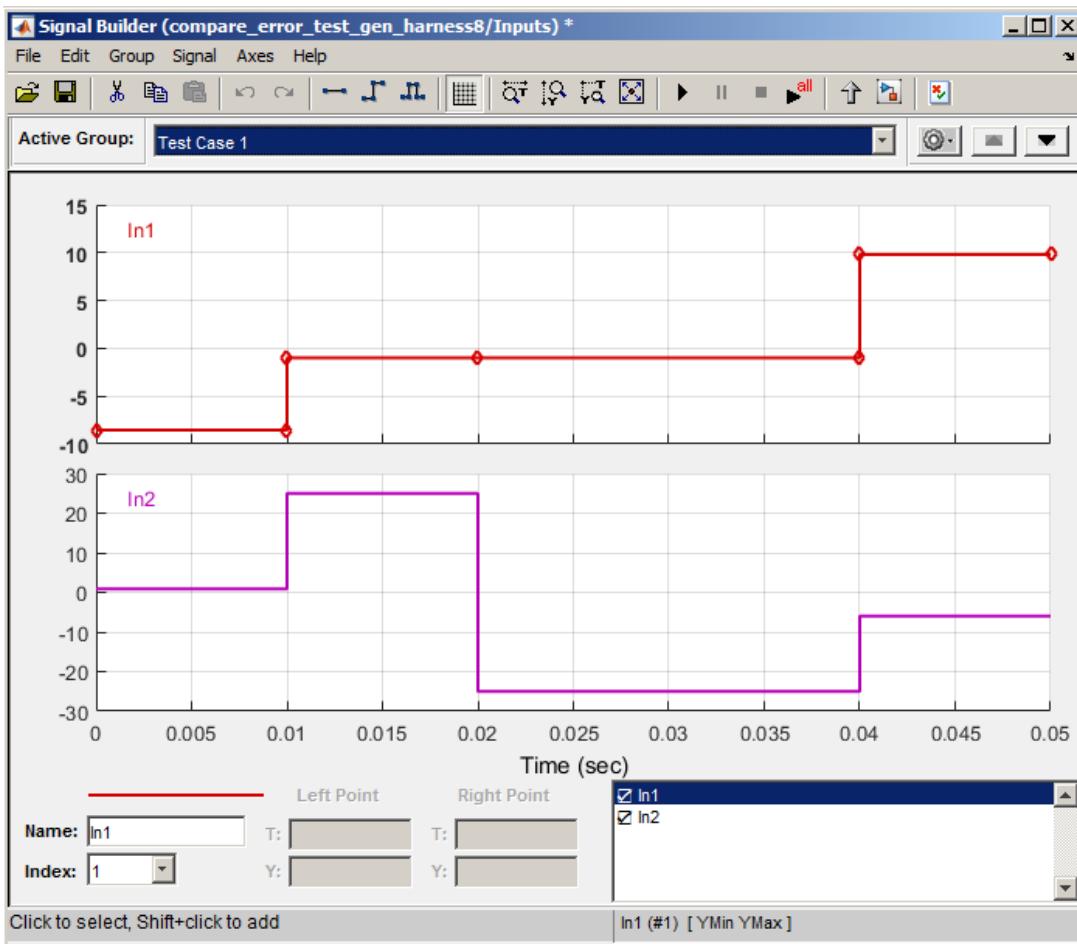


Generation of test cases



It is possible to generate test cases alone without proving a property using SLDV.

Test case using SLDV



The test case with the objective enabled tests the just $> +25$ and just <-25 case. It is possible to define our test objects for each block based on requirement and tests are generated automatically. Notice the time dependent waveform that is created to test the system.

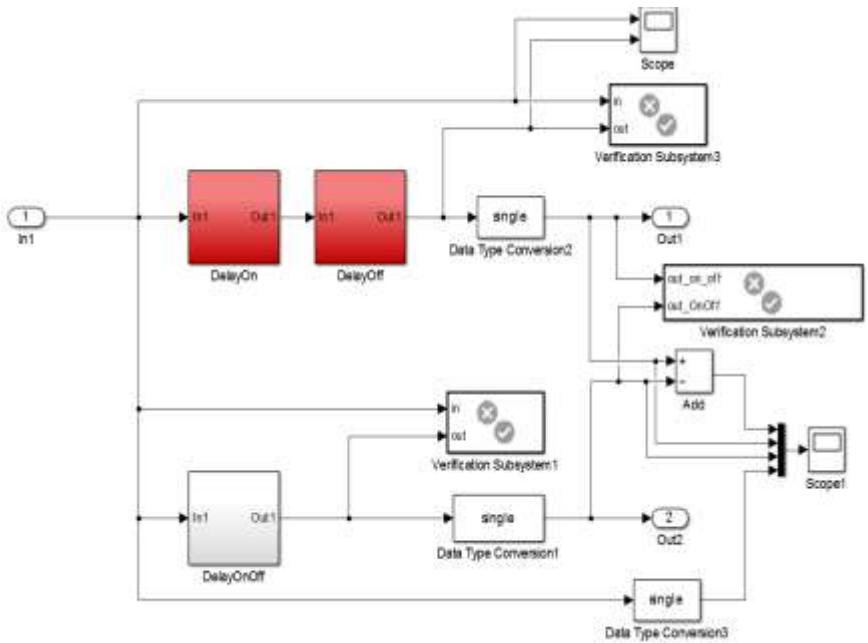
Delay On Off – Revisit as SLDV

The 3 Requirements are

- A) If the input is TRUE and holds TRUE for a duration of TON (20) Frames then output = TRUE
- B) If the input is FALSE and holds FALSE for a duration of TOFF (40) Frames then the output = FALSE

ELSE

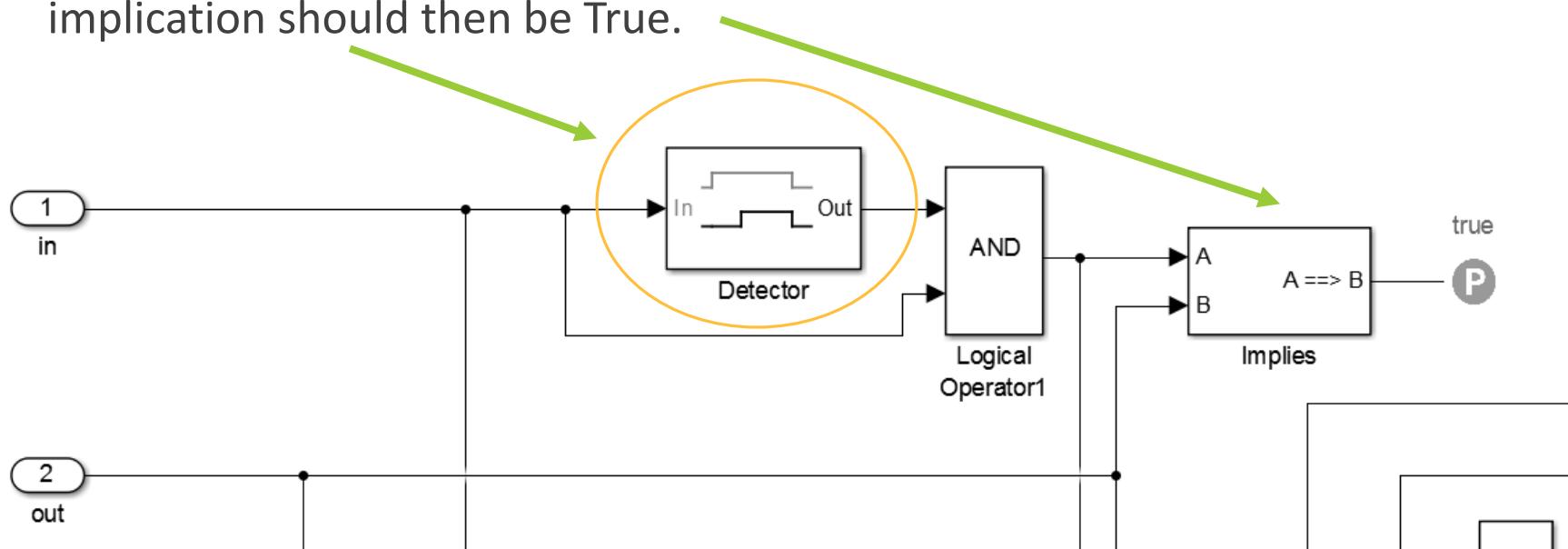
- C) The output shall be equal to previous value (NO CHANGE)



Two representation are modelled for property proving. Is DelayOn followed by DelayOff a correct representation? – We revisit this as SLDV

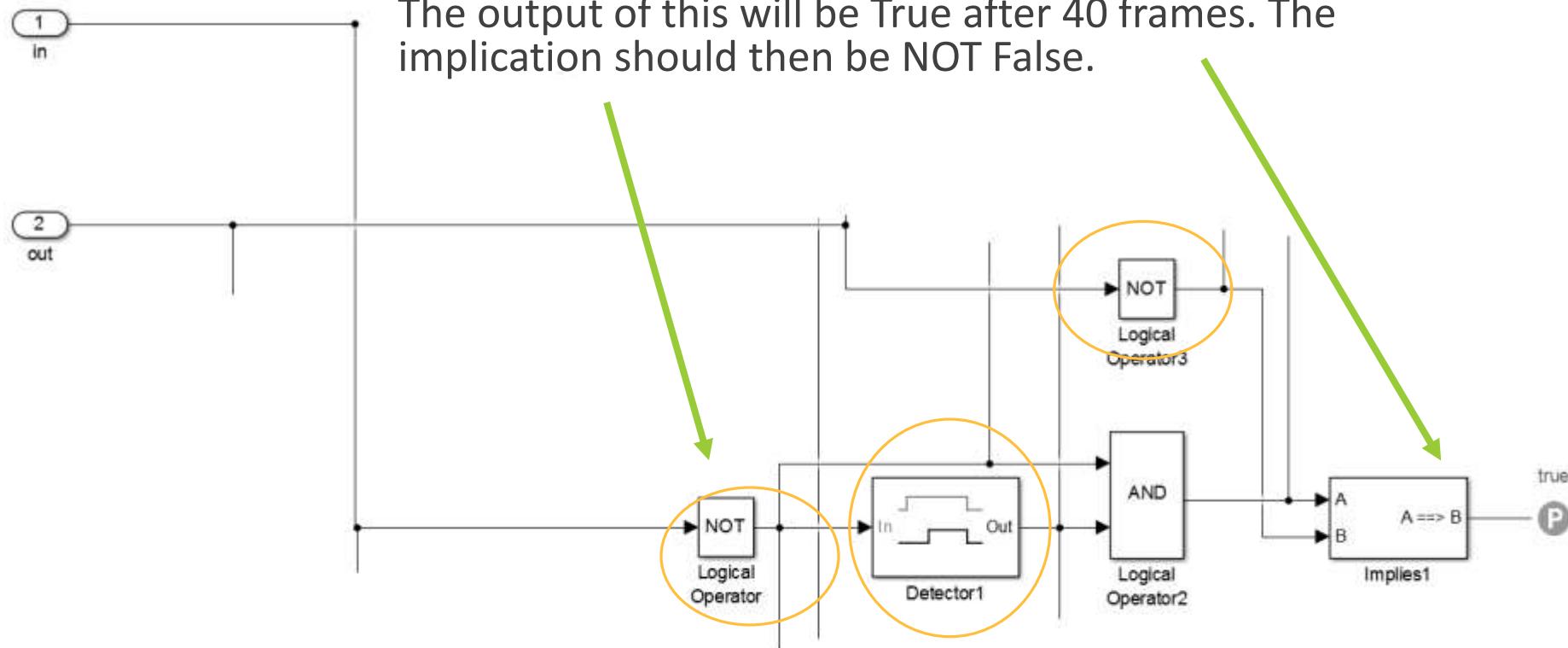
Verification Model

If the input is True and holds True for 20 Frames is modeled using the Detector Block. The output of this will be True after 20 frames. The implication should then be True.

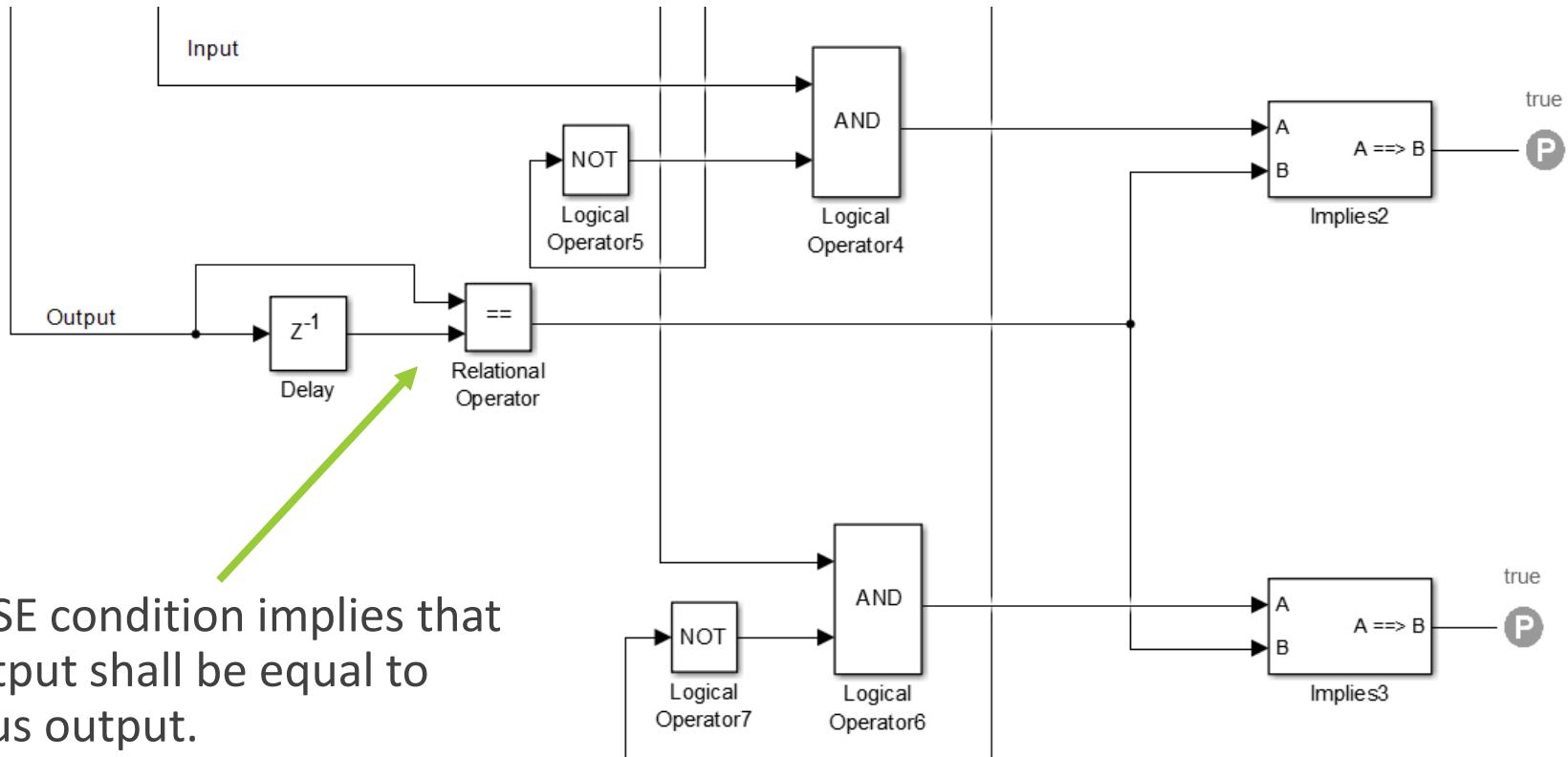


Verification Model

If the input is False and holds False for 40 Frames is modeled using the Detector Block fed by a NOT of Input. The output of this will be True after 40 frames. The implication should then be NOT False.

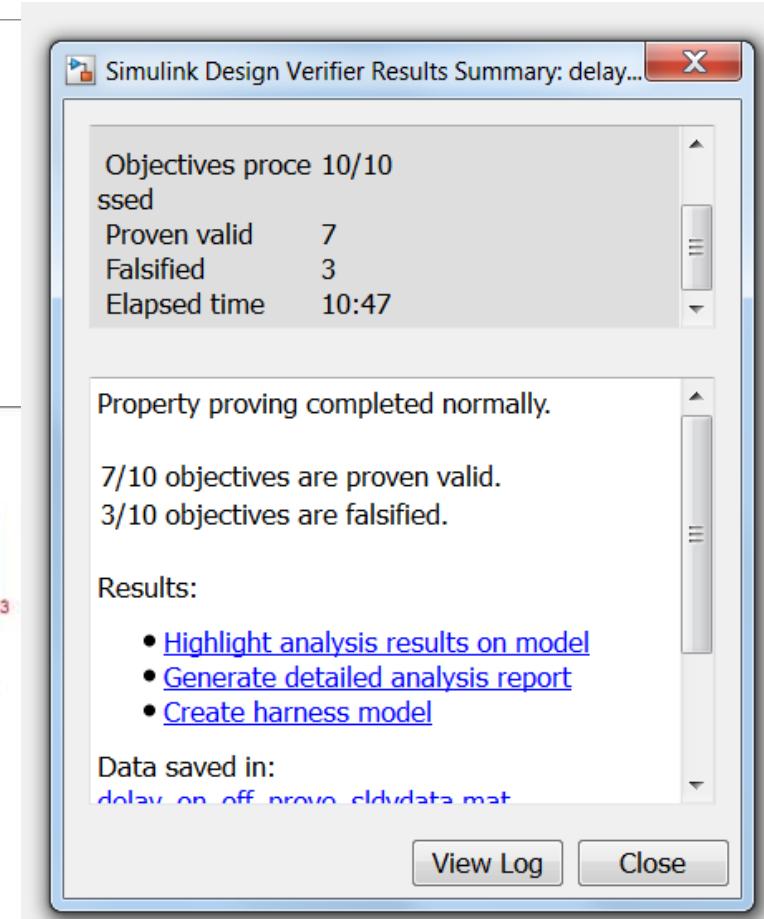
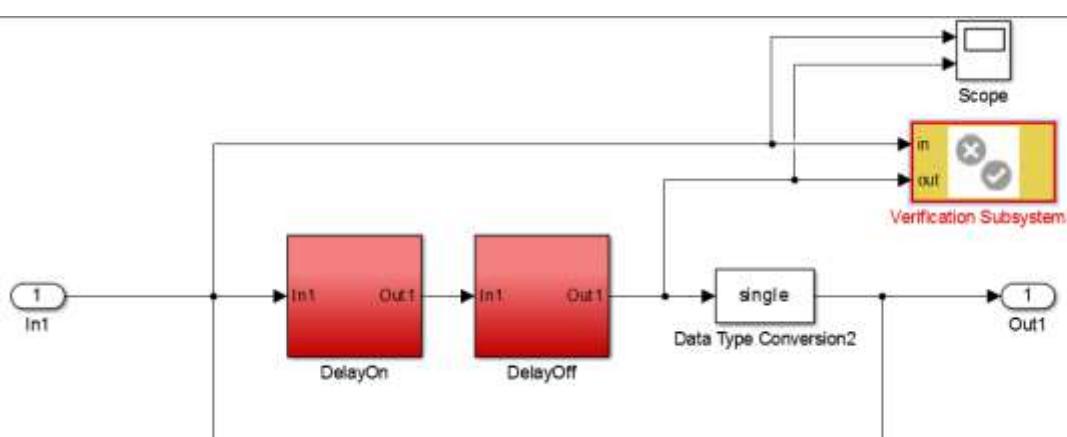


Verification Model



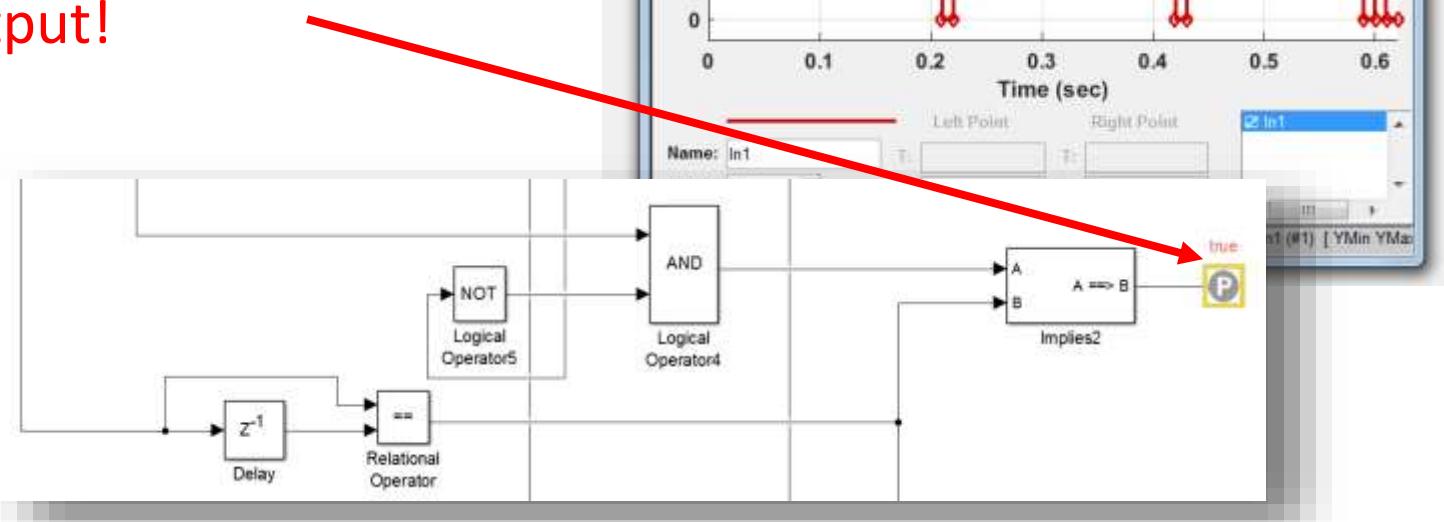
Assertions Falsified

Three test cases prove that the Delay On Followed by DelayOff is an incorrect implementation of the Delay On Off Block



Test Cases

The error between the two implementations is that in the else phase the combination of blocks does not hold the previous value of output!



Antiwindup Integrator - Revisit

Integrator Requirements

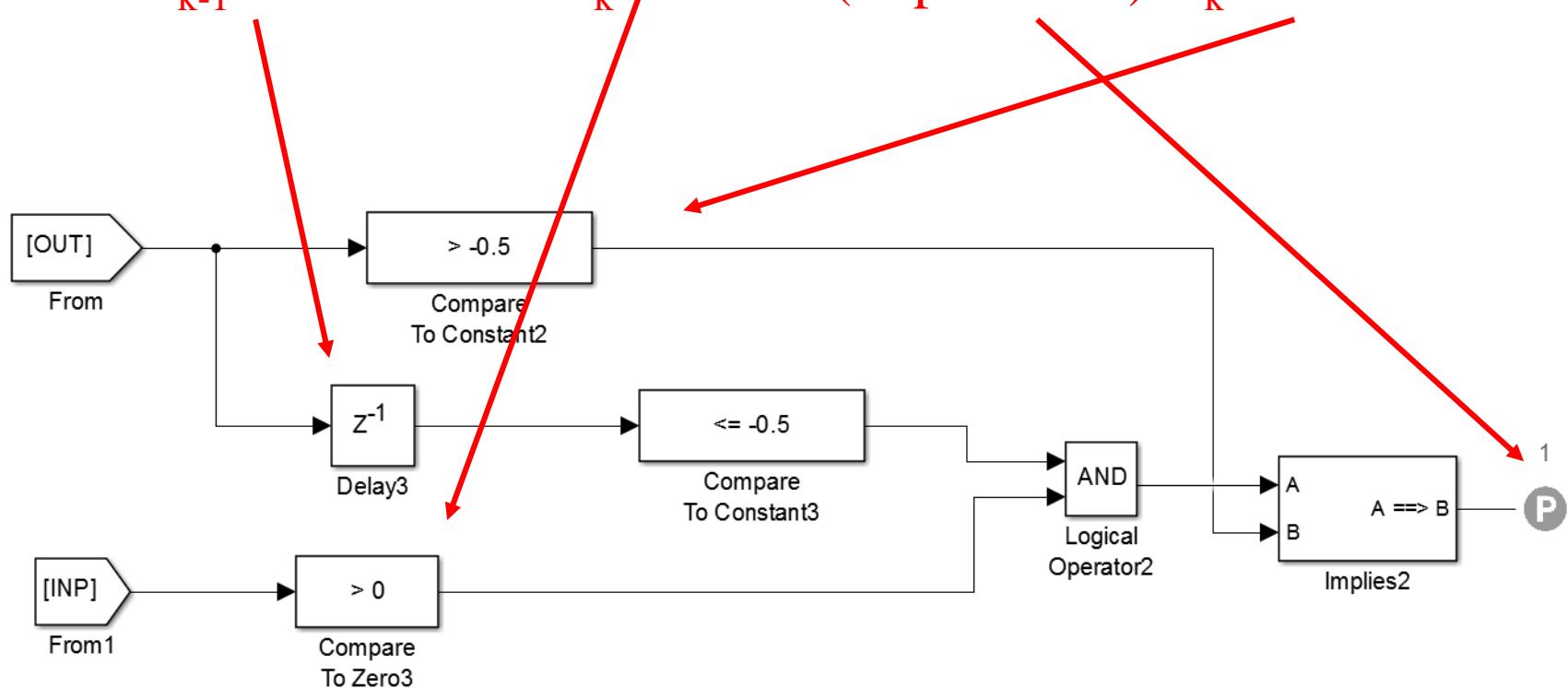
The output shall be computed as Input * DT (0.01 sec) + Previous Output

- $O_k = O_{k-1} + I_k * (0.01)$
- If $I_k = I_{k-1} = 0$ then $O_k = O_{k-1}$
- If $O_{k-1} \geq 0.5$ AND $I_k < 0$ then $O_k < 0.5$
- If $O_{k-1} \leq -0.5$ AND $I_k > 0$ then $O_k > -0.5$

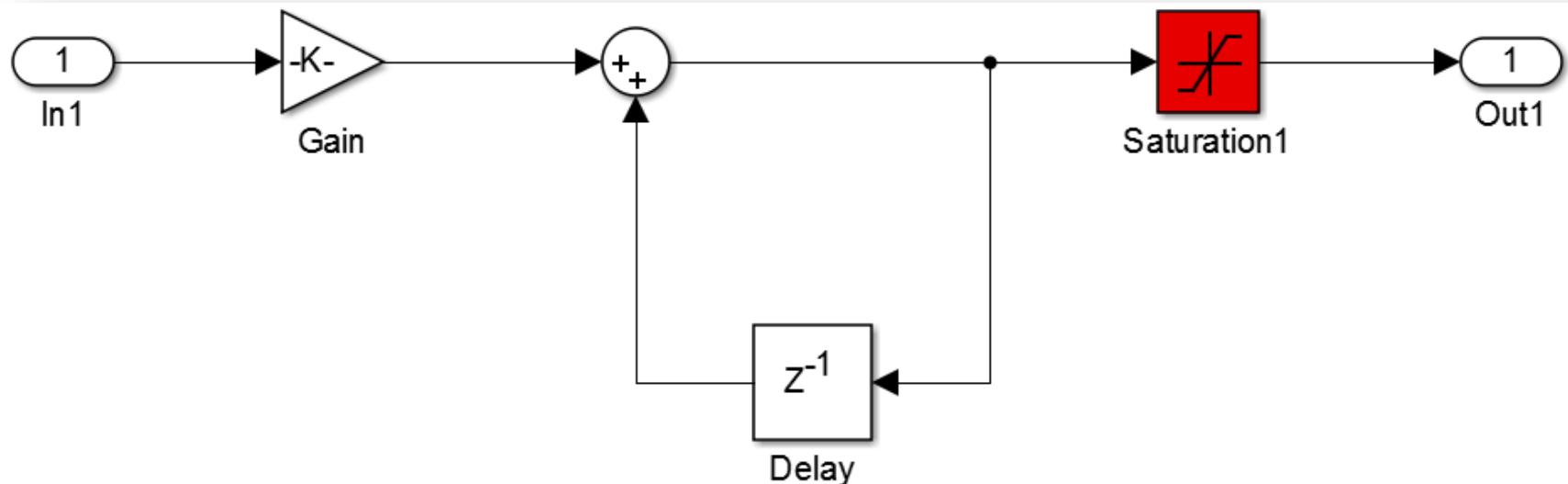
Where k is the current frame. O is the output and I the input.

Requirement Model

If $O_{k-1} \leq -0.5$ AND $I_k > 0$ then (implies that) $O_k > -0.5$

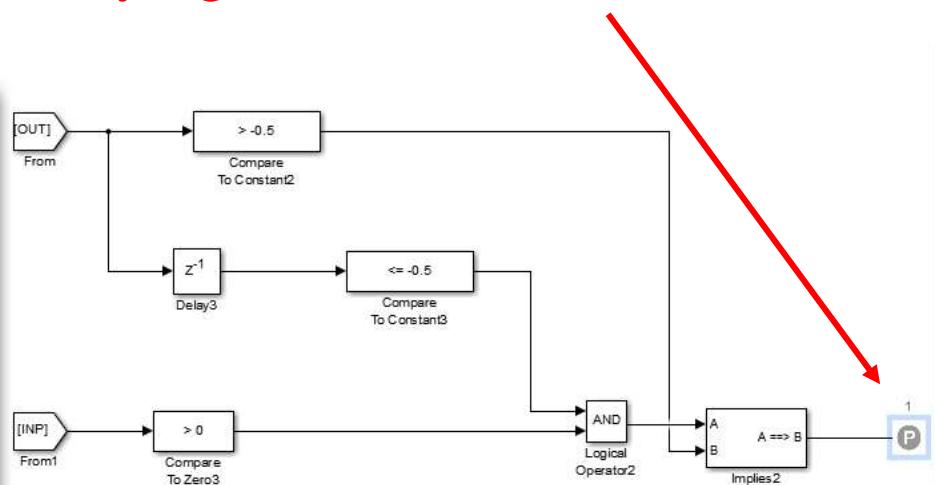
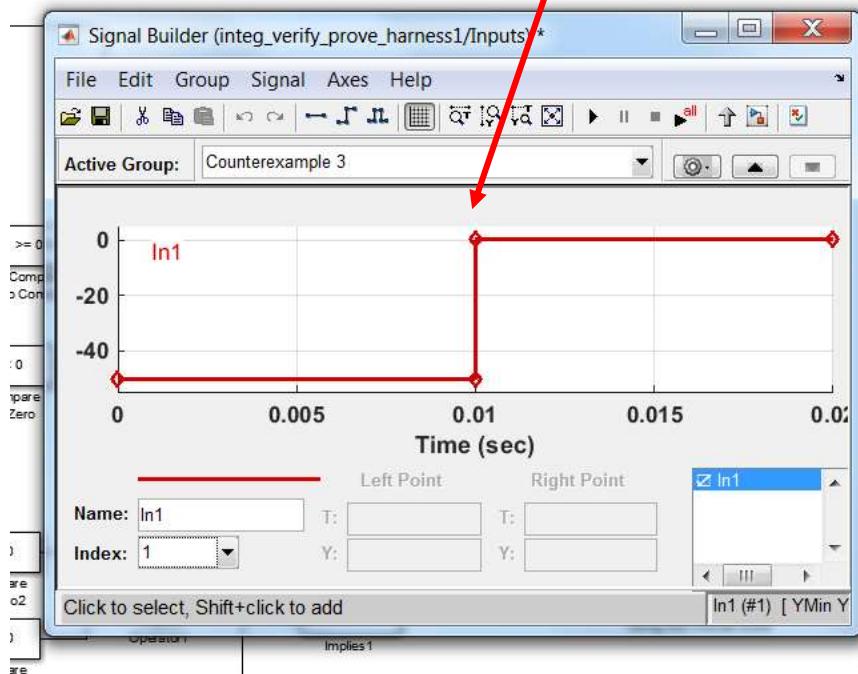


Incorrect Implementation



Test Case

The test case generated causes the integrator to saturate and then the input changes sign falsifying the assertion.



Here the integrator is not an antiwindup implementation

Rate Limiter

Requirements

- $O_1 = I_1$

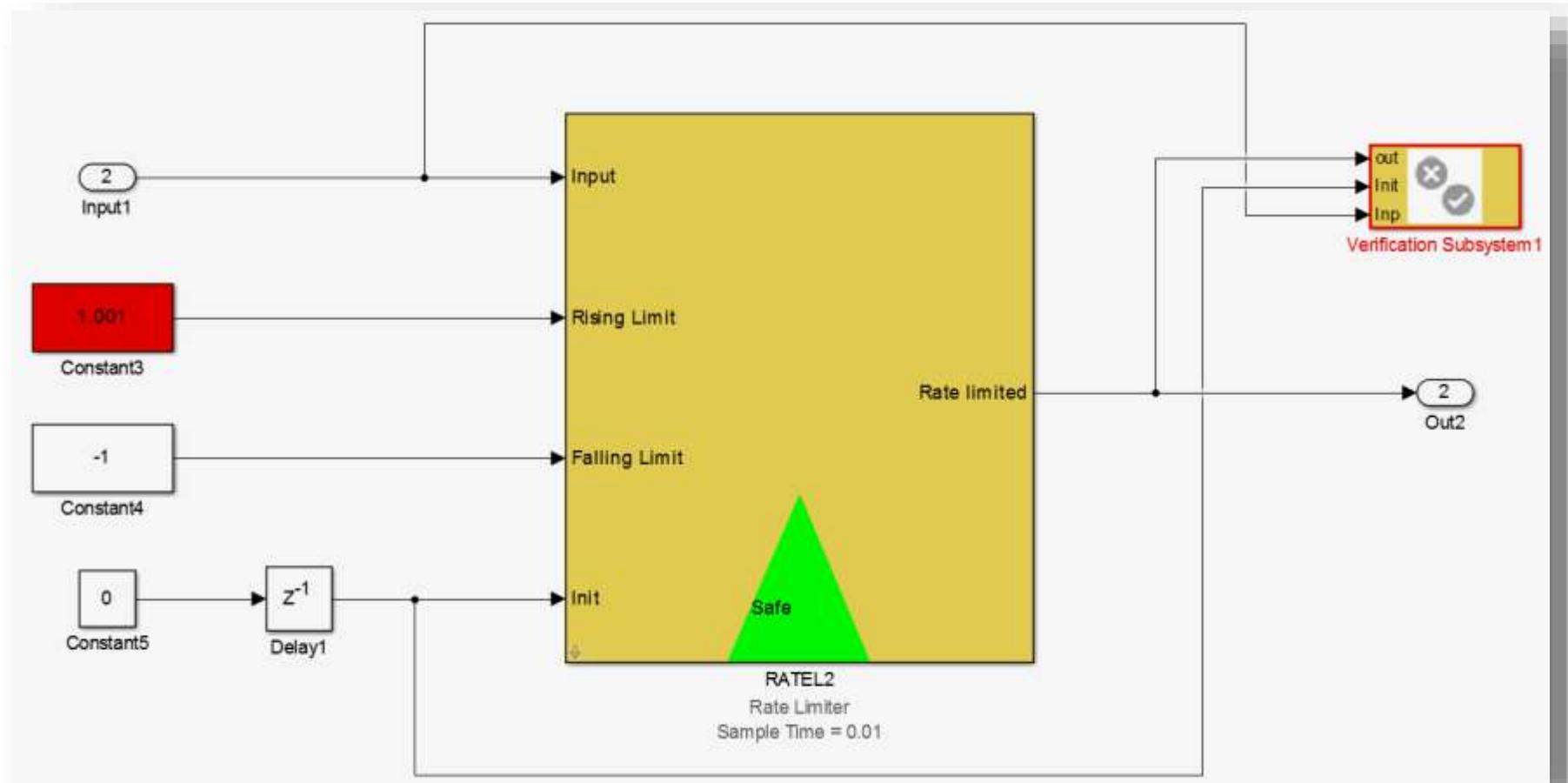
Where k is the current frame, I is the input and O is the output. DT is the sampling time.

For all $k > 1$

- If $\text{abs}\{(O_k - O_{k-1})/DT\} < 1$ then $O_k = I_k$
- $\text{abs}\{(O_k - O_{k-1})/DT\} \leq 1$ for all $k > 1$

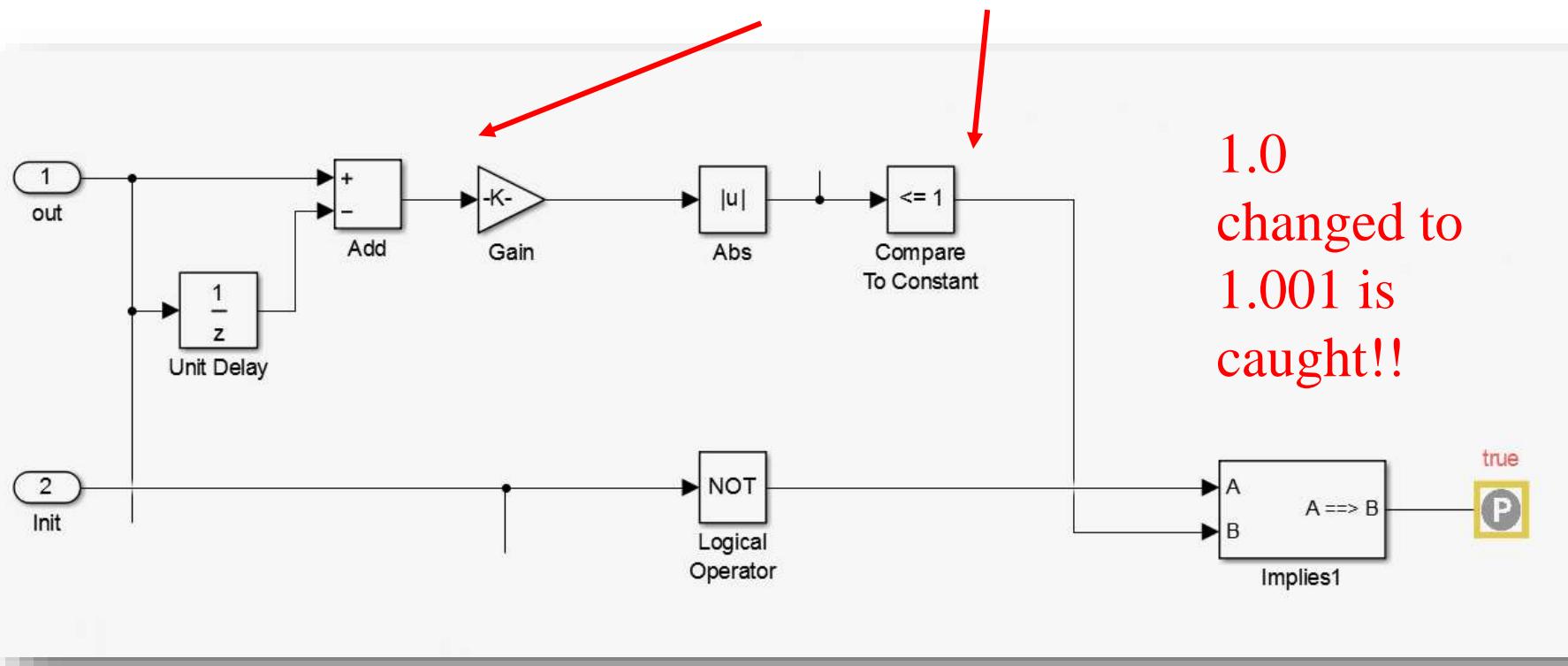
The output rate should be limited to 1

Rate Limiter - Sensitivity

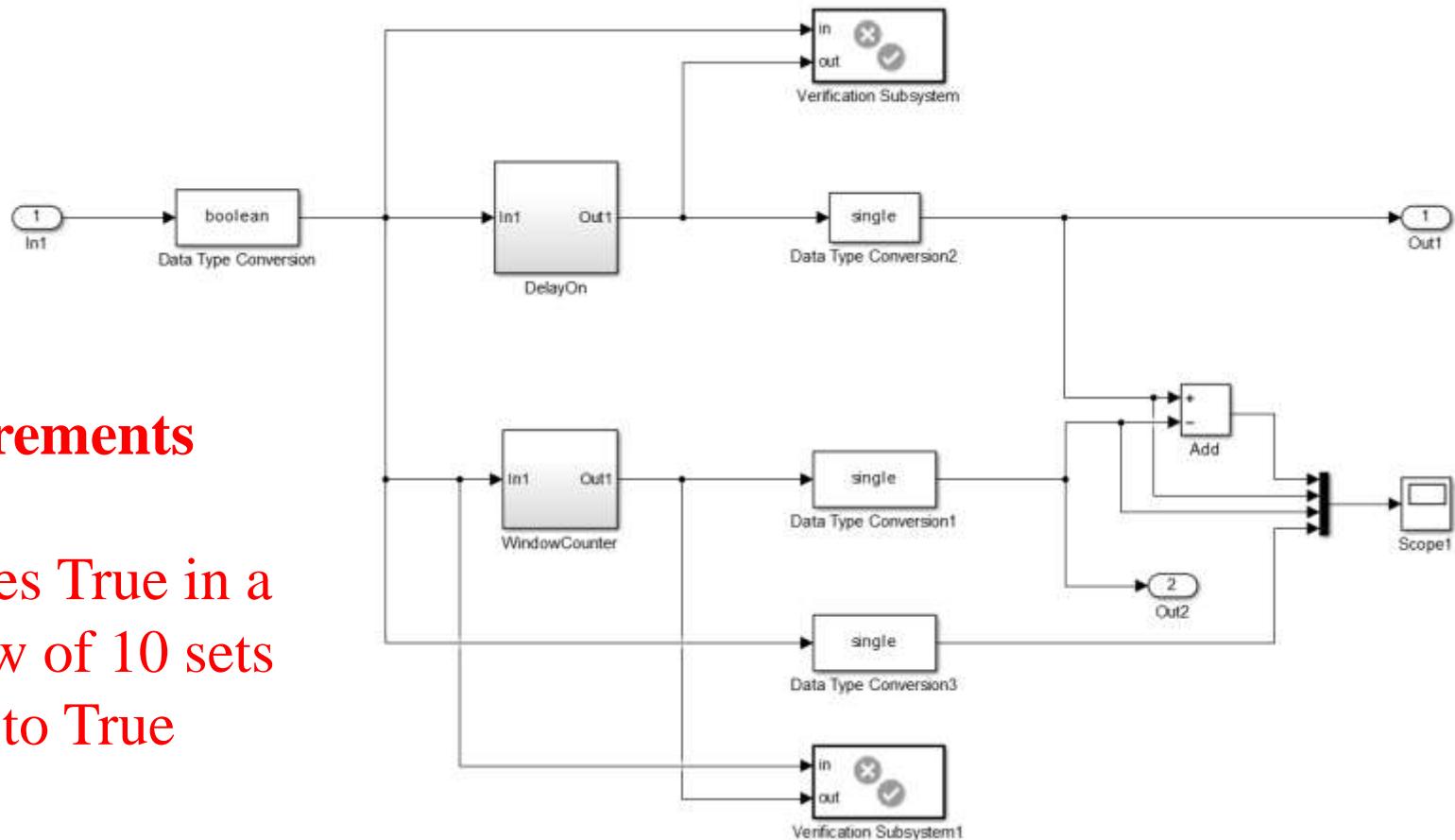


Rate Limiter – Requirement Model

If NOT k=1 (init) implies that rate ≤ 1.0



Window Counter Vs Delay On



Requirements

3 frames True in a window of 10 sets output to True

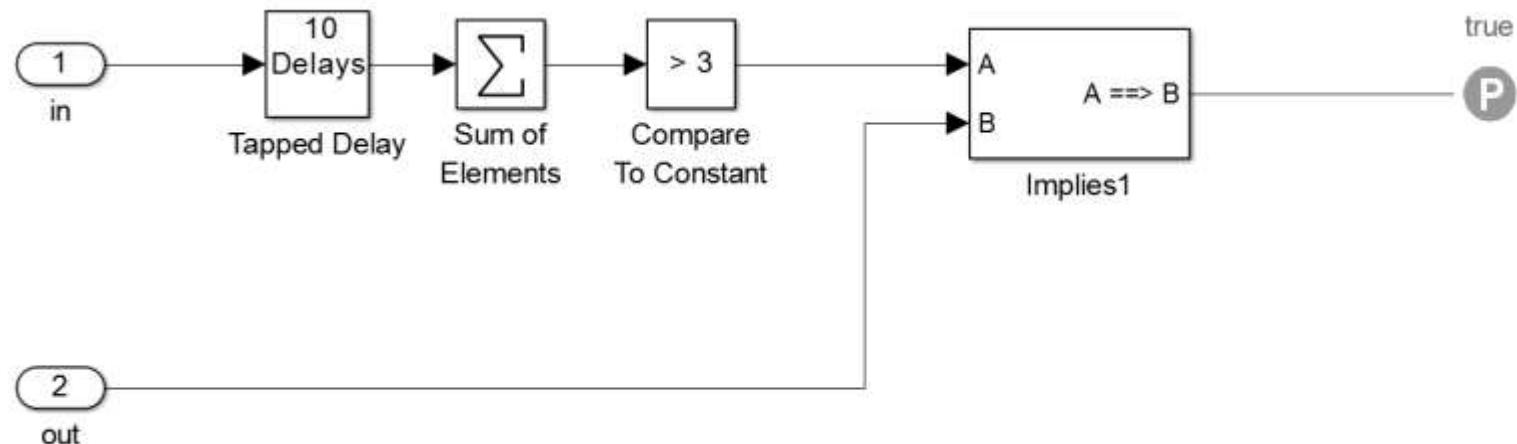
Requirement Model

Requirements

If $(I_k + I_{k-1} + \dots + I_{k-9}) > 3$ then $O_k = \text{TRUE}$

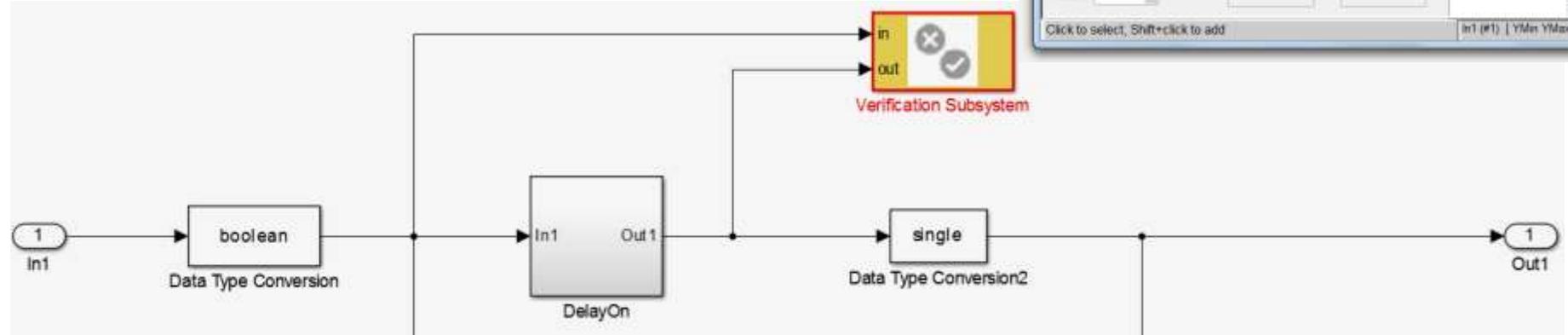
Else

$O_k = \text{FALSE}$



SLDV Falsifies

The Delay On block does not behave like the windows counter. A counter example test case is generated which proves the Delay On is **NOT Window Counter!**

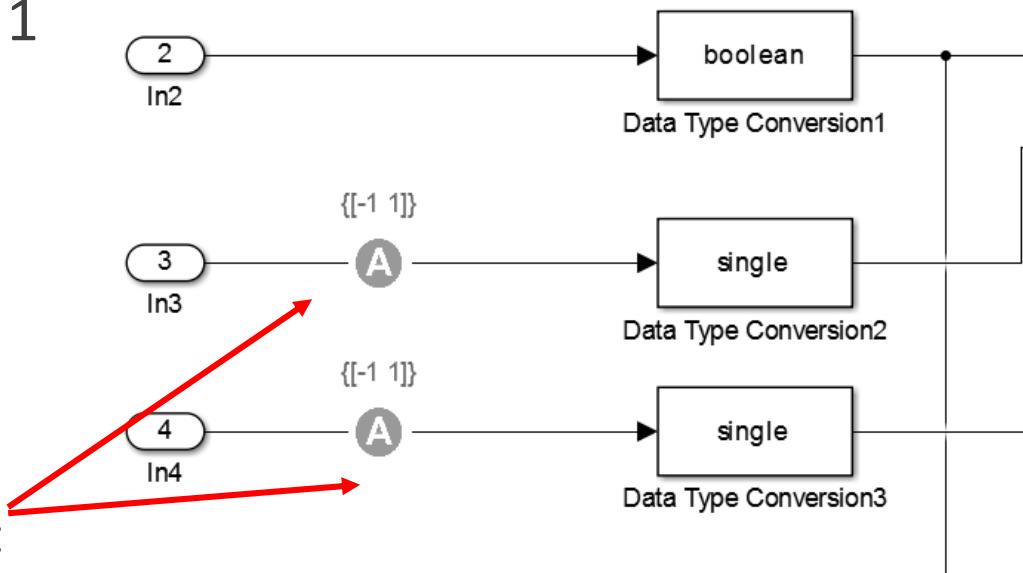


Transient Free Switch

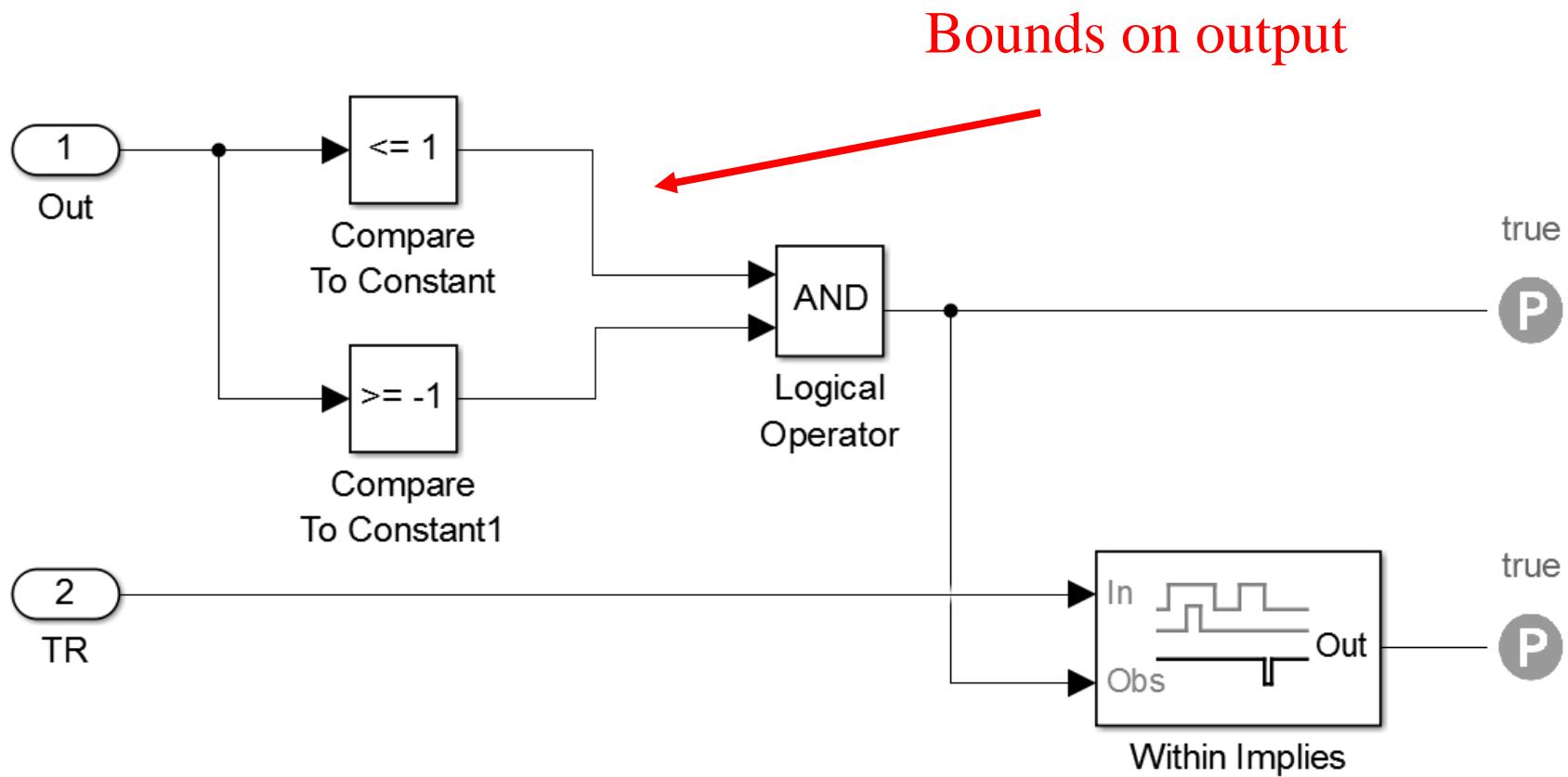
- The Transient free showed a problem where even if the inputs were bounded between 0 and 1 the output had become negative causing a problem in flight.

- This is modelled in SLDV as

If the inputs are bounded between -1 and 1 the output should be bounded between -1 and 1

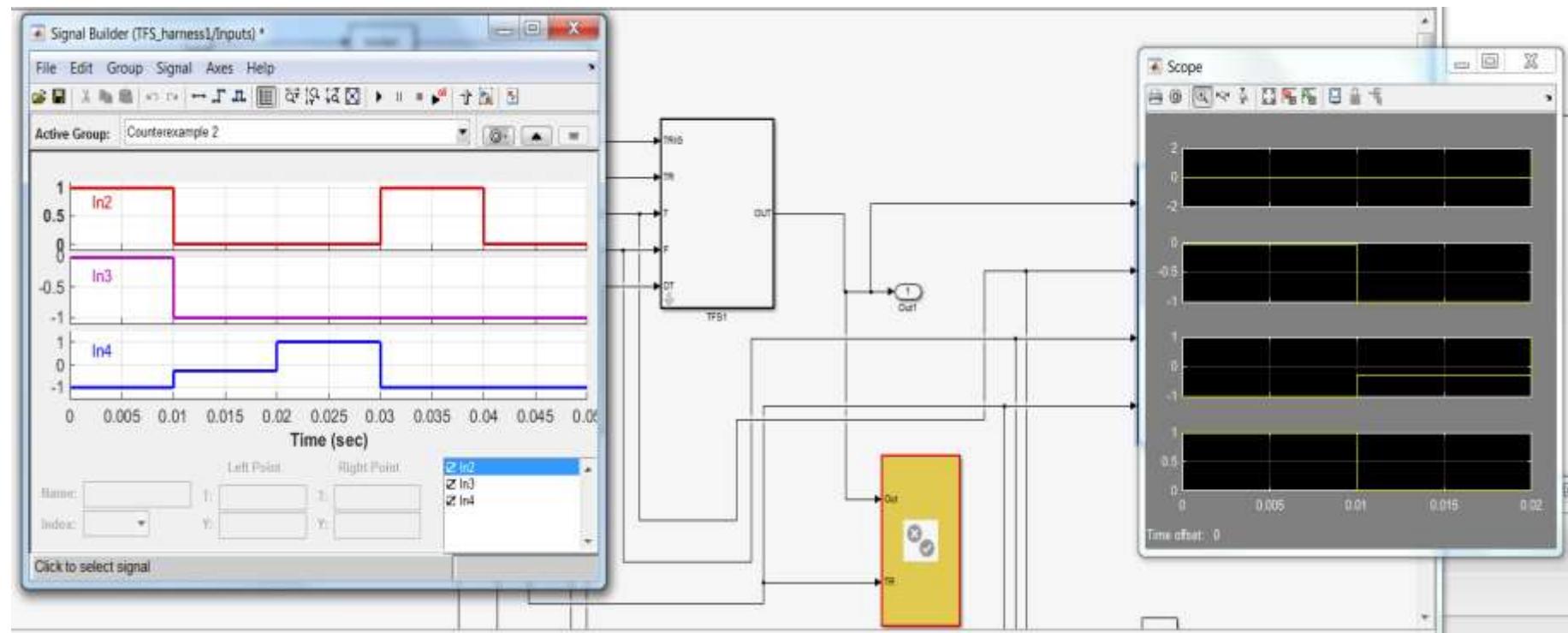


TFS Requirement



Simulation

The test case makes the output go greater than 1.0!



SLDV Scalability

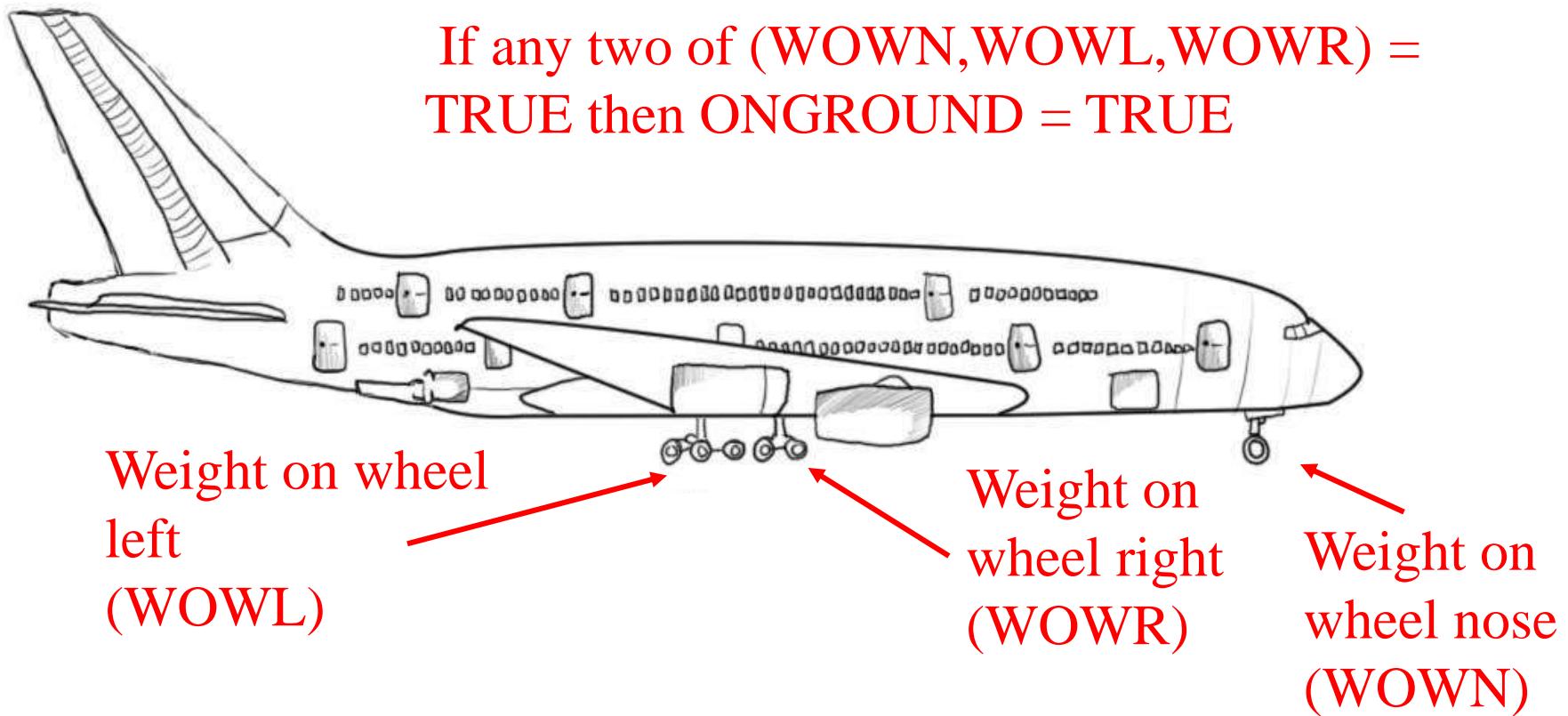
- Simulink design verifier has brought out all the errors found in the flight control blocks defined earlier
 - The process is very easy to implement with just the few blocks provided for SLDV.
 - All the example blocks are available on the Mathworks website in case anyone wants to explore further.
 - The ease of implementation is brought out by the fact that engineering first year students could implement this easily.
-
- **How does SLDV scale up ?**

On Ground Problem

- Aircraft systems require an accurate sense of the aircraft being on ground or in air. This signal is used to reset system, clear failures and start critical system in air.
- A measure of the aircraft on ground is from the sensors on the aircraft landing gear. Weight on Wheel (WOW) proximity switches are provided on the wheel supports to indicate the On Ground status.
- Aircrafts normally have two main wheel units (left and right) and the nose wheel unit. If any two of the three switches indicate WOW then the aircraft is considered on ground.
- As this is a very critical component alternate measure of on ground is taken from the aircraft airdata and radio altimeter systems.

On Ground Requirement

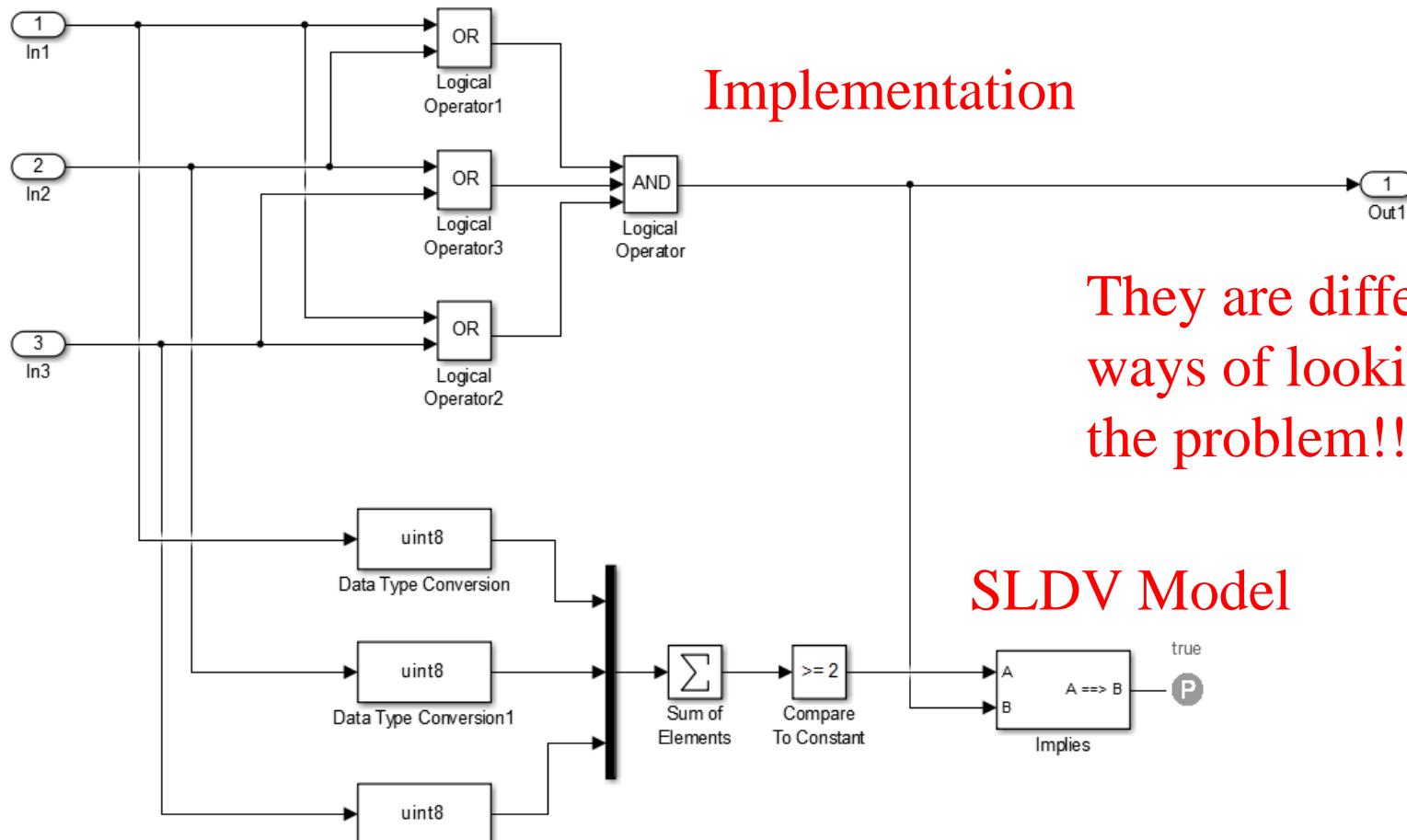
If any two of (WOWN,WOWL,WOWR) = TRUE then ONGROUND = TRUE



Alternate On Ground

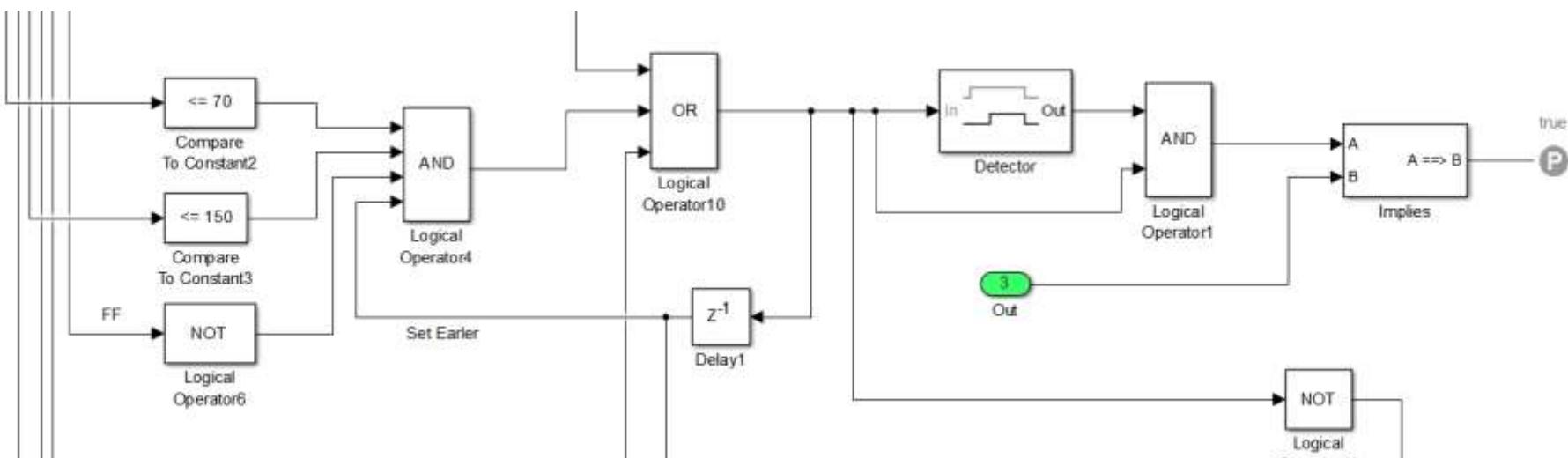
- CAS is calibrated airspeed in knots or the speed from the airdata sensor and RADALT is the radio altimeter sensor value in feet
- If $CAS < 60$ AND $RADALT < 100$ AND for 20 frames then $ONGROUND = TRUE$
- If $CAS > 70$ OR $RADALT > 150$ AND for 40 frames then $ONGROUND = FALSE$
- If first frame AND $CAS \leq 65$ AND $RADALT \leq 125$ then $ONGROUND = TRUE$

SLDV On Ground WOW

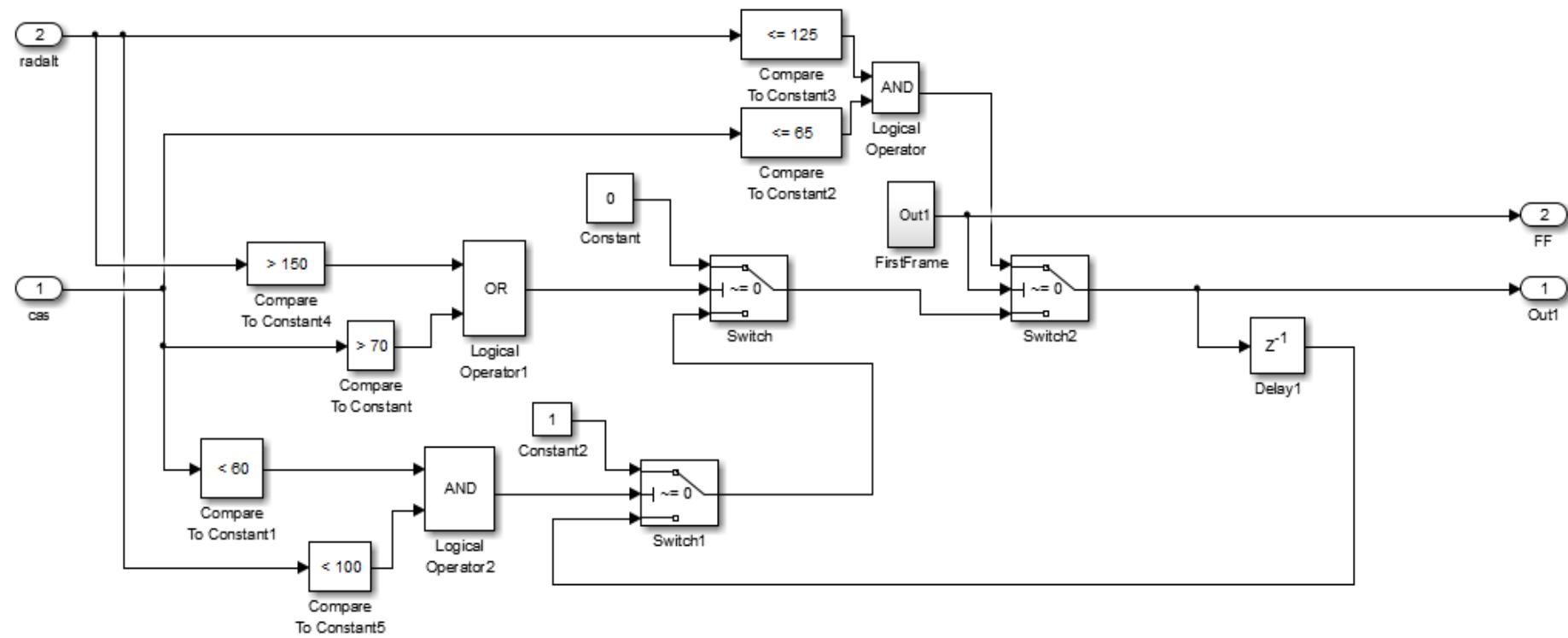


SLDV Alternate On Ground

This implementation for SLDV looks at the True condition CAS ≤ 70 AND RADALT ≤ 150 AND NOT (First Frame) AND Previous True to define ON GROUND = TRUE. This is checked for N Frames using the detector block. **Out** is the signal from the implementation model.



Alternate On Ground Implementation

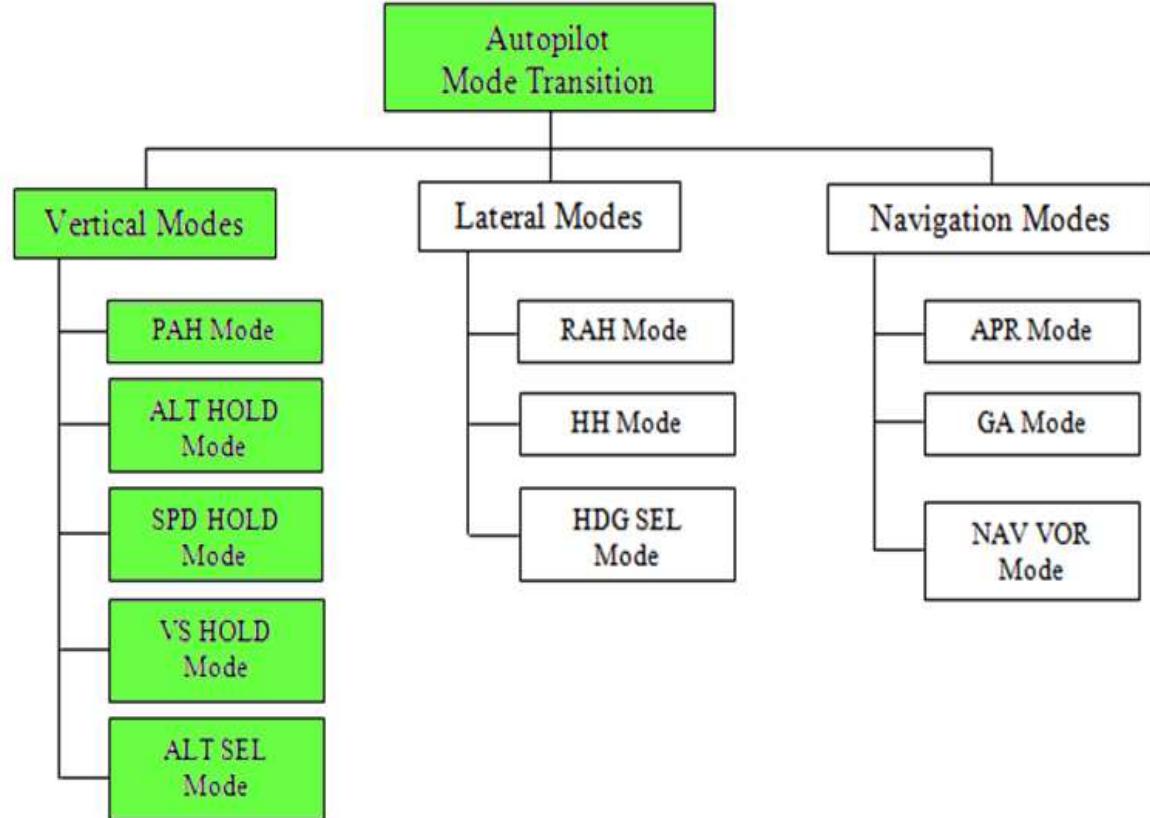


Tips

The implementation model and the Simulink Design Verifier model are two different models developed independently. This will ensure the correctness of the implementation if the independence is preserved. SLDV may not be as complicated as the implementation. This however SHOULD be reviewed for correctness.

Autopilot Mode Transition

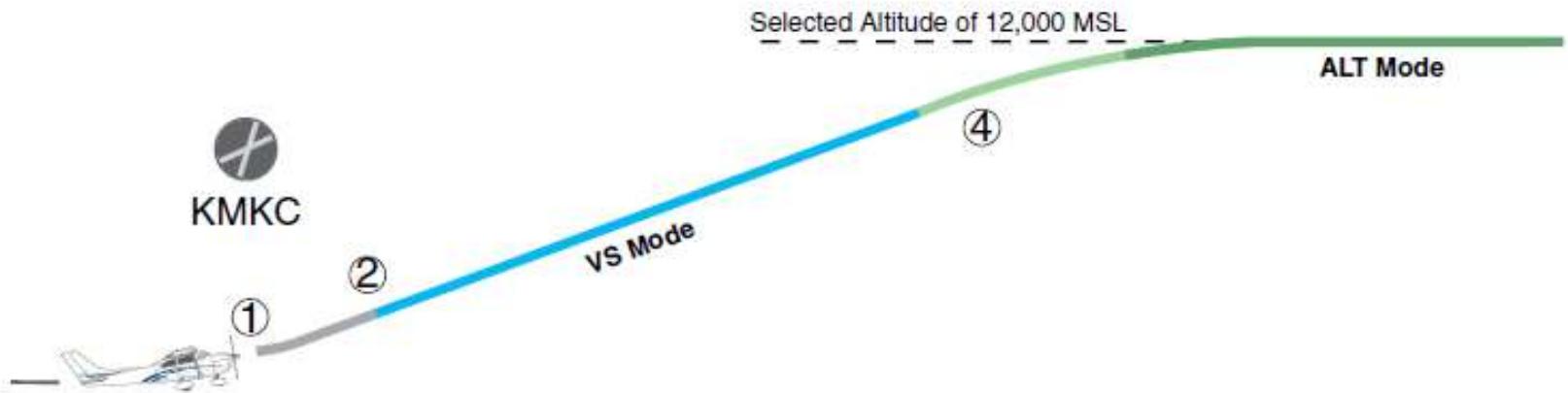
We consider only the vertical modes for the problem



Autopilot Vertical Modes

- PAH mode – This is the basic autopilot mode in vertical axis. This mode holds the current pitch angle.
- Altitude Hold mode (ALT) – This mode holds the aircraft at the current altitude reference.
- Speed Hold mode (SPD) – This mode maintains the present airspeed.
- Vertical Speed mode (VS) – This mode is used to automatically maintain the aircraft at a selected vertical speed (climb rate) reference.
- Altitude Select mode (ALT SEL) – This mode captures the Selected Altitude. The 3 phases are, Arming, Capture and Hold

Reference



A Methodology to Design a Validated Mode Transition Logic

[http://link.springer.com/chapter/10.1007/978-81-322-2141-8 24](http://link.springer.com/chapter/10.1007/978-81-322-2141-8_24)

This models are available at

<http://www.mathworks.com/matlabcentral/profile/authors/5987424-natasha-jeppu>

Transition Table

States	Sl. No.	Modes	Buttons				Software Triggers			
			01	02	03	04	05	06	07	08
			AP	SPD	VS	ALT	ALTS	ALTCAP	ALTCPDN	APFAIL
Vertical	01	DIS(Vertical)	02	00	00	00	00	00	00	00
	02	PAH	01	03	04	05	00	06	00	01
	03	SPD HOLD	01	02	04	05	00	06	00	01
	04	VS	01	03	02	05	00	06	00	01
	05	ALT HOLD	01	03	04	02	00	00	00	01
	06	ALTS CAP	01	00	00	05	00	00	05	01
AP	01	AP ON	02	00	00	00	00	00	00	02
	02	AP OFF	01	00	00	00	00	00	00	00
ALT SEL	01	ALTS OFF	00	00	00	00	02	00	00	00
	02	ALTS ARM	01	00	00	01	01	03	00	01
	03	ALTSEL CAP	01	00	00	01	00	00	01	01

Condition Table

States	Sl. No.	Modes	Buttons				Software Triggers			
			01	02	03	04	05	06	07	08
			AP	SPD	VS	ALT	ALTS	ALTCAP	ALTCPDN	APFAIL
Vertical	01	DIS(Vertical)	01	00	00	00	00	00	00	00
	02	PAH	02	03	04	05	00	00	00	02
	03	SPD HOLD	02	00	04	05	00	00	00	02
	04	VS	02	03	00	05	00	00	00	02
	05	ALT HOLD	02	03	04	00	00	00	00	02
	06	ALTS CAP	02	00	00	05	00	00	00	02
AP	01	AP ON	00	00	00	00	00	00	00	00
	02	AP OFF	01	00	00	00	00	00	00	00
ALT SEL	01	ALTS OFF	00	00	00	00	00	00	00	00
	02	ALTS ARM	02	00	00	05	00	00	00	02
	03	ALTSEL CAP	02	00	00	05	00	00	00	02

Table Description

- There are three independent modes
 - Vertical mode
 - AP mode
 - ALTSEL mode
- Each row indicates the current state in each mode.
- The columns indicates the trigger.
- The corresponding cell defined by the state (row) and trigger (col) in the condition table gives the condition to be satisfied for a transition to take place.
- The corresponding cell in the transition table indicates the new state to which the autopilot will transit if the condition is satisfied.

Example

Consider DIS state (state 1) in Vertical mode and trigger 1 that is AP trigger.

States	Sl. No.	Modes	01	02	01
01	DIS(Vertical)		01	AP	
02	PAH			02	01

Transition table

Trigger

Current State

Condition table

States	Sl. No.	Modes	01	02	01
01	DIS(Vertical)		01	AP	
02	PAH			02	01

Condition 1 to be satisfied

Example

- First, look at the cell corresponding to vertical state 1 i.e. DIS and trigger AP in the condition table. The value in the cell indicates the condition to be satisfied for the transition to take place (in this case condition 1).
- If this condition is satisfied then the corresponding cell in the transition is looked at. The value in the cell indicates the state of transition within the given mode.
- In this case state 2 indicates PAH state in vertical mode.

Example - Condition

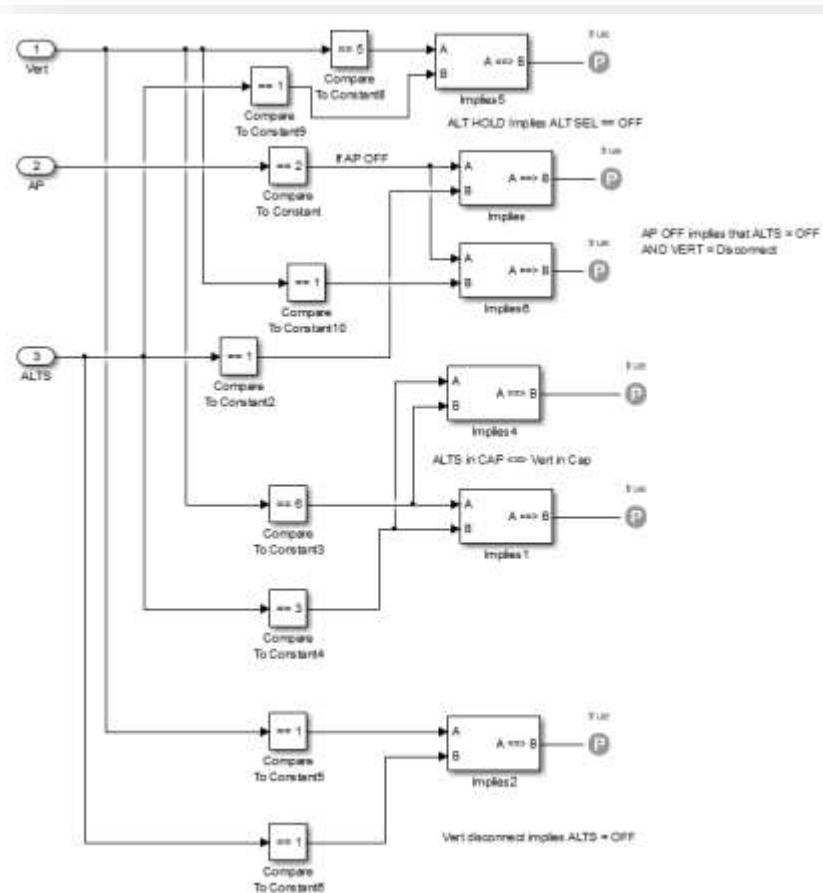
- C1: No AP inhibit conditions exist.
 - C2: AP ON
 - C3: CAS is greater than 120 and less than Vmo (knots)
 - C4: $|VS| > 50 \text{ ft/min}$.
 - C5: $|VS|$ is less than 500 ft/min
-
- Note :Condition 2 has a dependency on the Autopilot State.

Trigger Not Possible

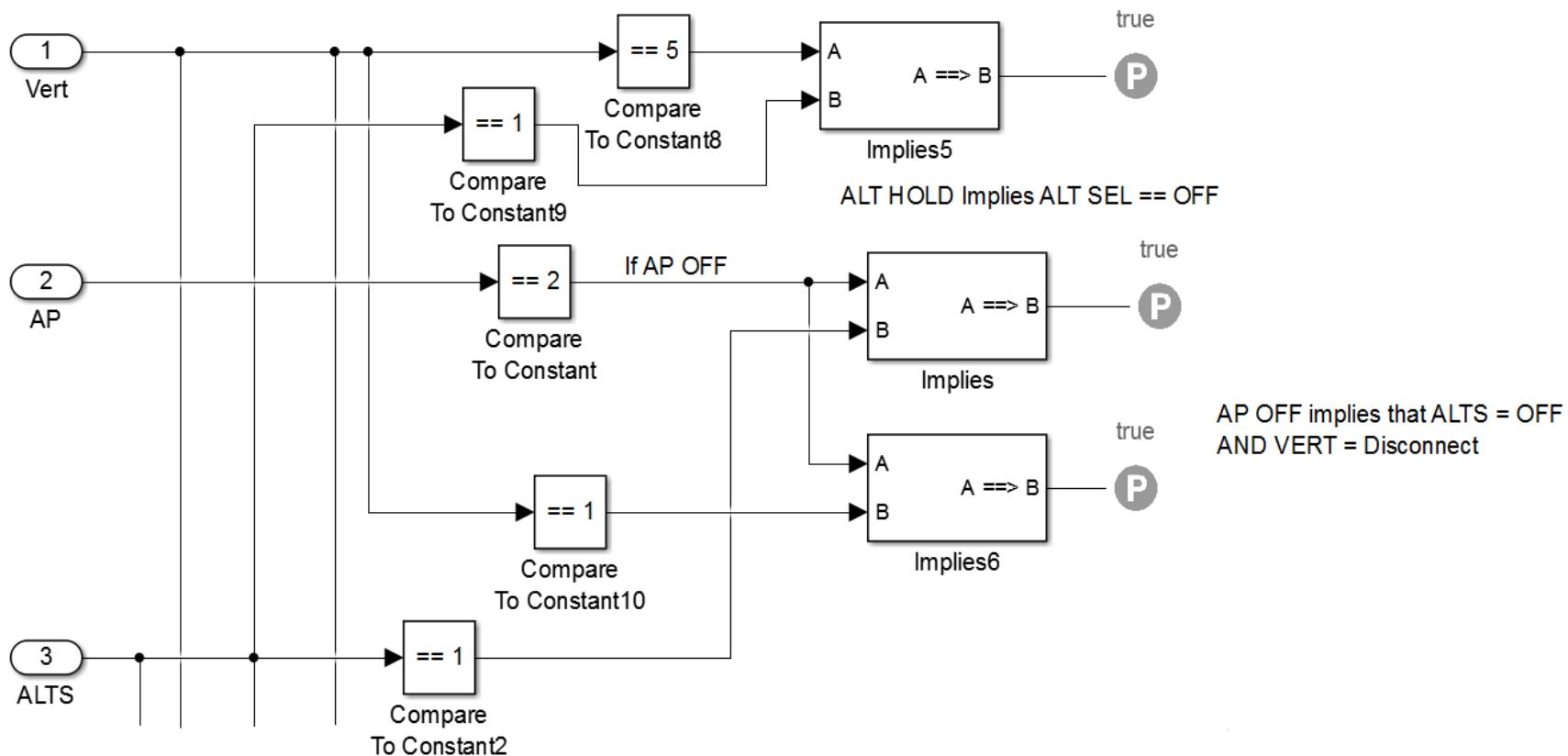
- Some of the software triggers have dependency on the State. This has to be considered. (SLDV brought out some these initial design flaws)
- if AP is OFF software triggers ALTS, ALTCAP, ALTCAPDN and APFAIL are not possible
- if ATLSEL mode is not in ARM state then ALTCAP trigger is not possible
- if VERTICAL state not in ALTCAP mode ALTCAPDN trigger is not possible
- if VERTICAL state in ALTHOLD mode then ALTS trigger is not possible

Assertions

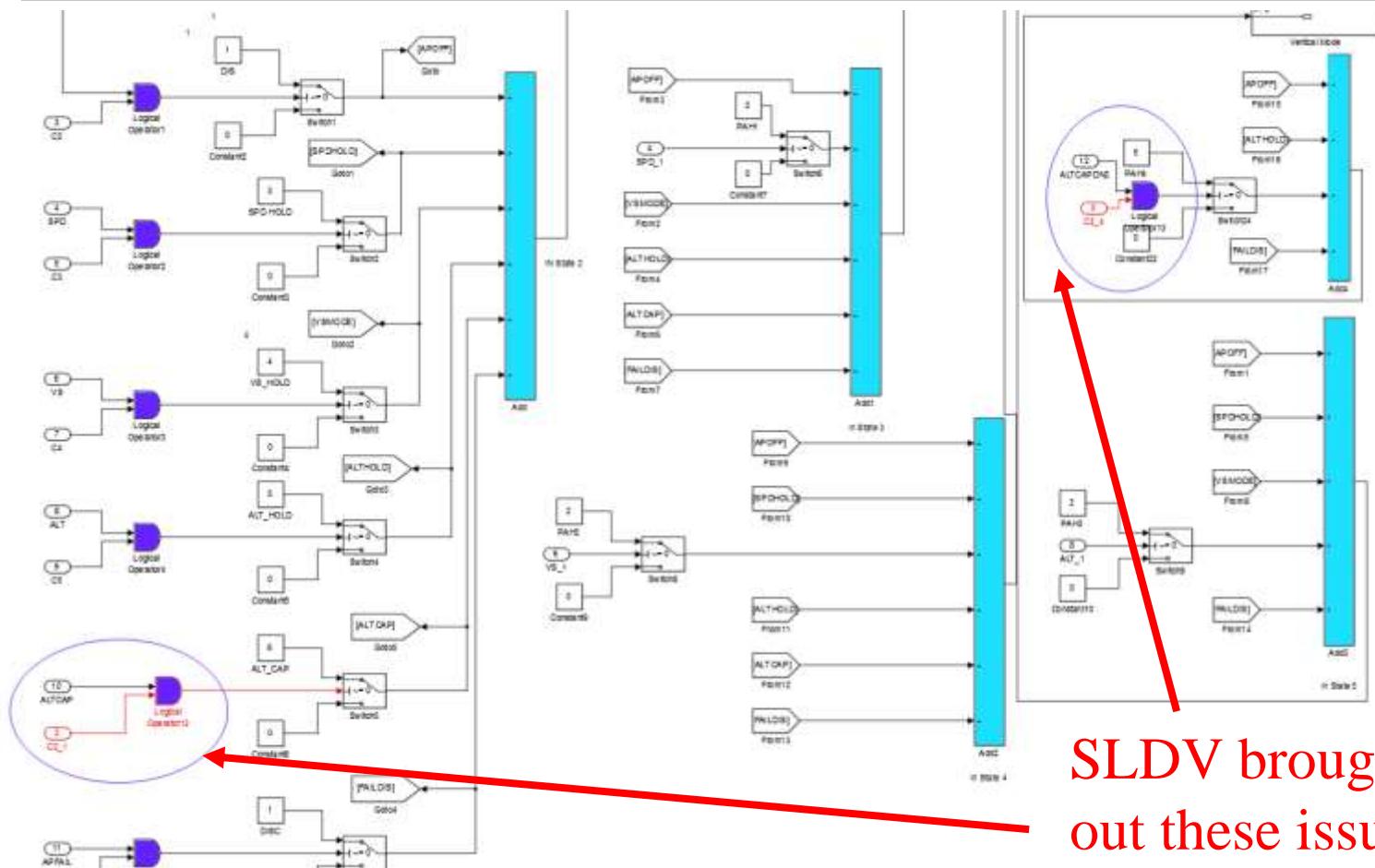
- If Vert = ALTCAP then ALTSEL = ALTSEL CAP
- If Vert = ALT HOLD then ALTSEL = OFF
- If Vert = DIS then AP = OFF
- If Vert = DIS then ALTSEL = OFF



SLDV Assertions



Design Model



SLDV brought
out these issues

Up/Down Counter as SLDV

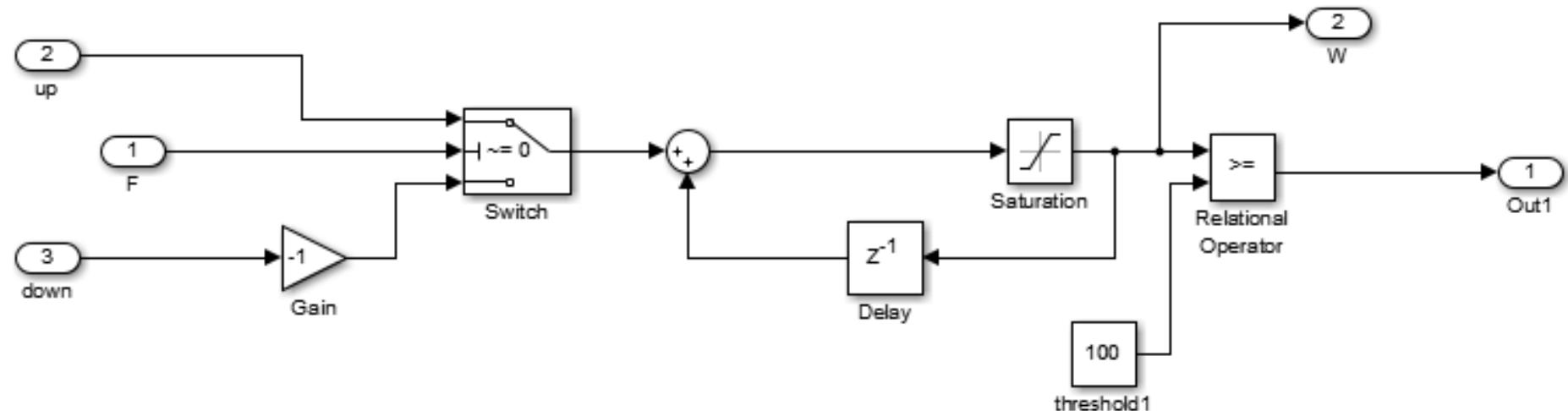
- The numeric error in the up down counter is revisited as an SLDV problem. The error was an implementation of the count variable as an unsigned integer.
 - The upper limit UB is 300, the trigger threshold Th2 is 100
 - This is modelled as a basic library block and validated for different values of up count and down count.
-
-
-
- **SLDV can be used in this manner to validate all your basic library blocks. Just imagine the power of a set of such formally proved blocks and their corresponding C code in your design.**

Counter Implementation

Up down counter design

$\text{Th2} = 100$

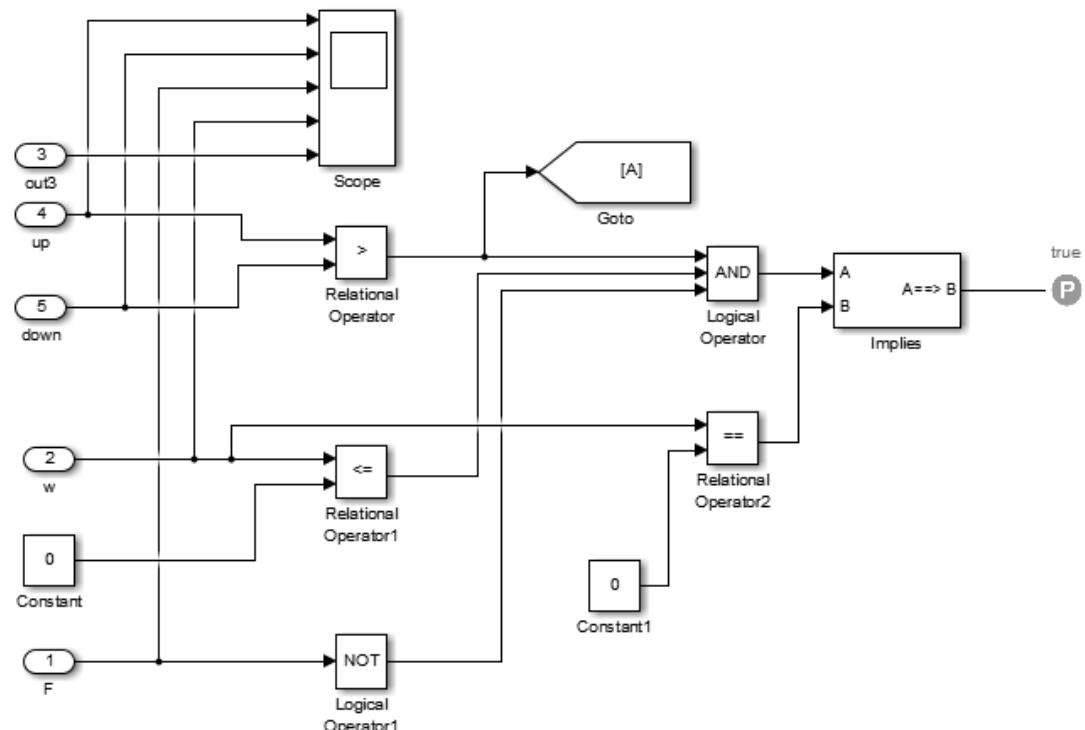
$\text{UB} = 300$



Property 1- Design

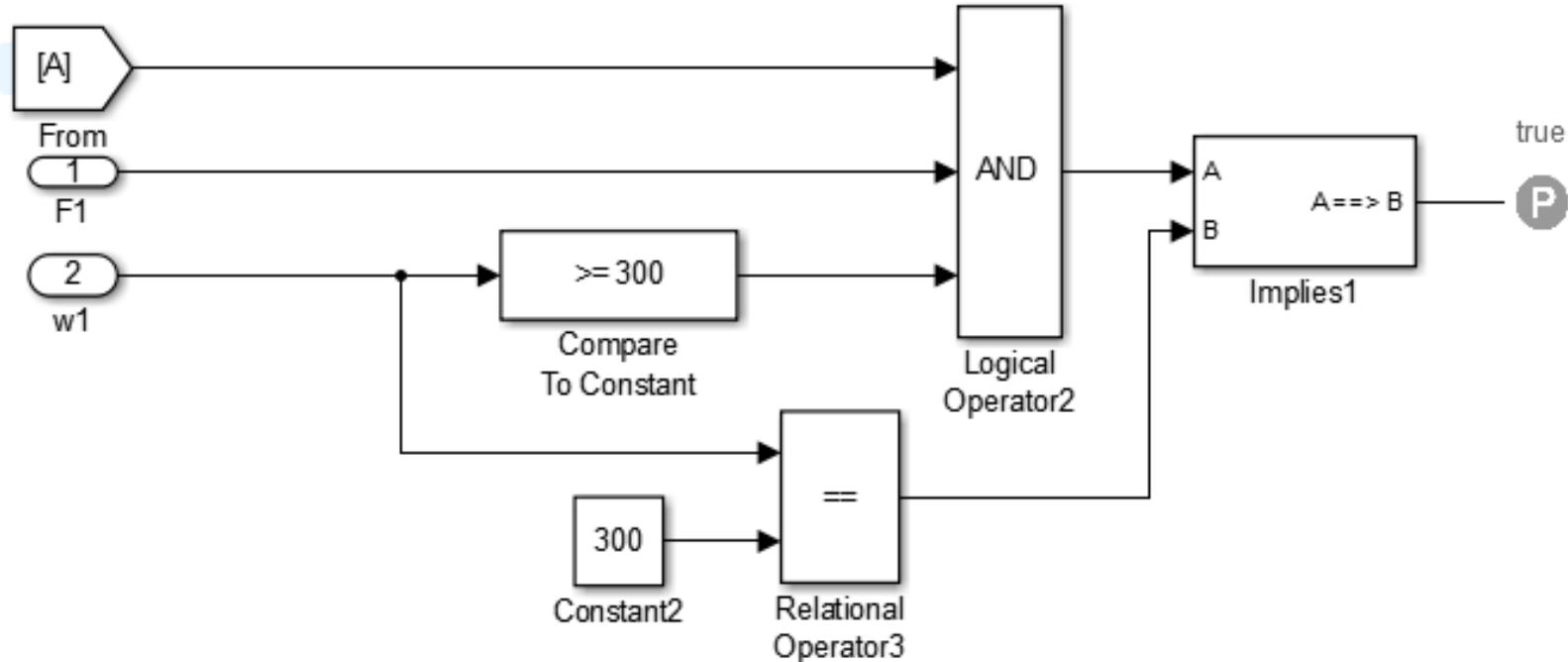
Base property: $U > D$

$\text{if } (w \leq 0 \text{ and } \text{Inp} = 0) \Rightarrow w = 0$



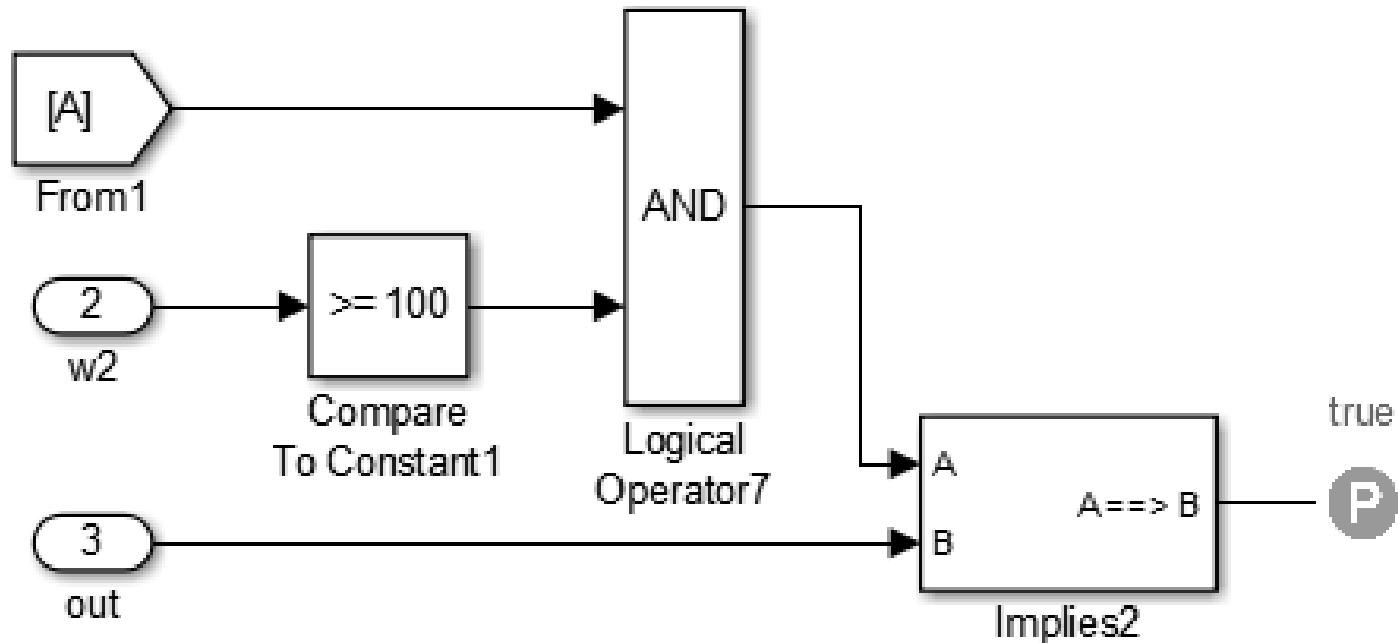
Property 2 - Design

if ($w \geq UB$ and $Inp = 1$) $\Rightarrow w = UB$



Property 3 - Design

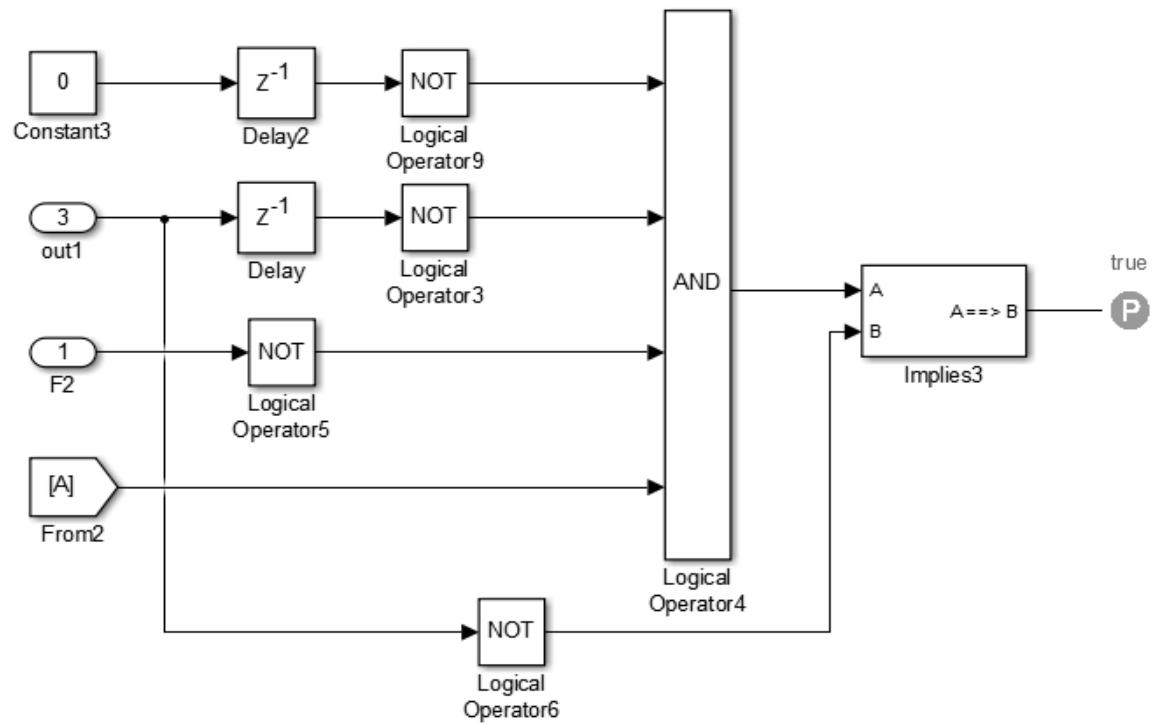
if ($w \geq Th2$) => output is TRUE.



Property 4 - Behavior

if (not (first frame) AND previous value of output = FALSE
AND Inp = 0) => output = FALSE

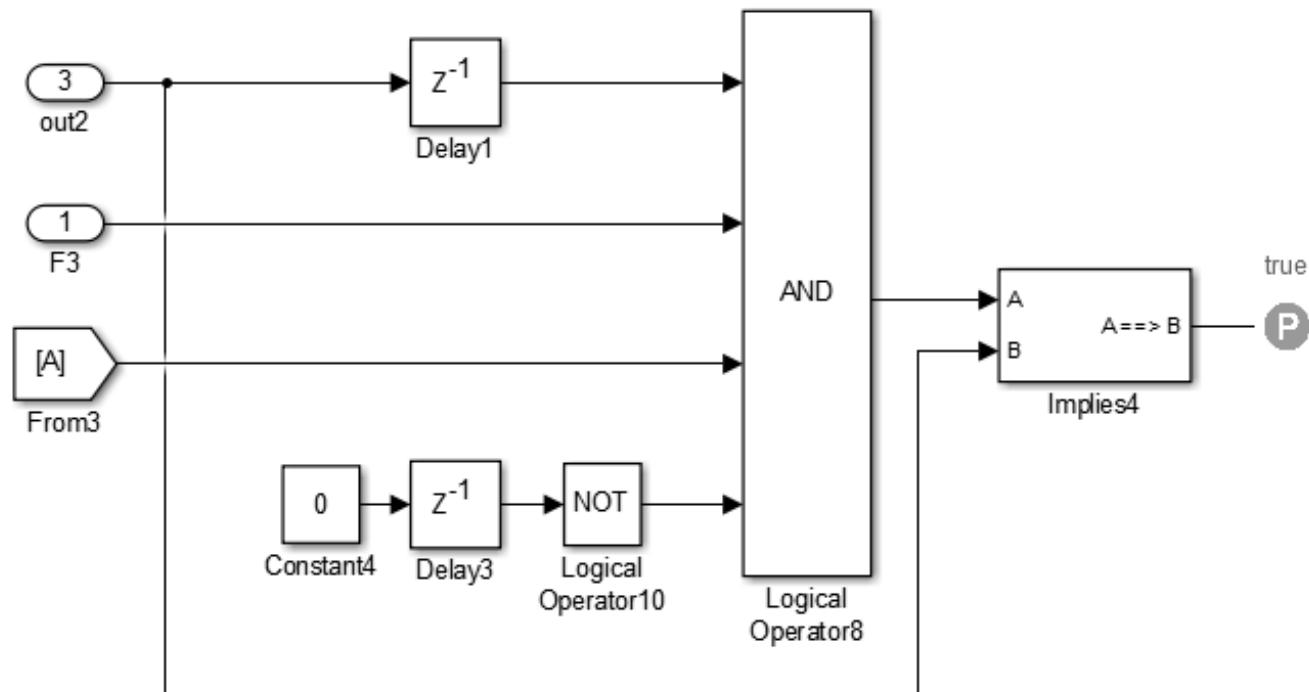
This is a subtle behavioral requirement which may not be specified.



Property 5- Behavior

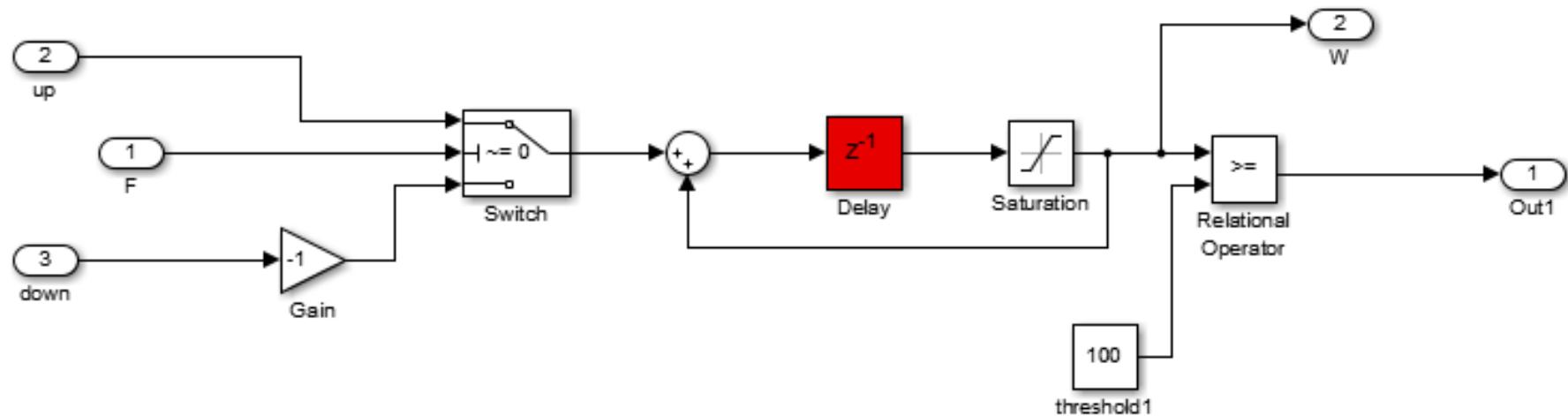
if not (FF) AND previous value of output is TRUE AND Inp = 1
implies output = TRUE

This is a subtle behavioral requirement which may not be specified.



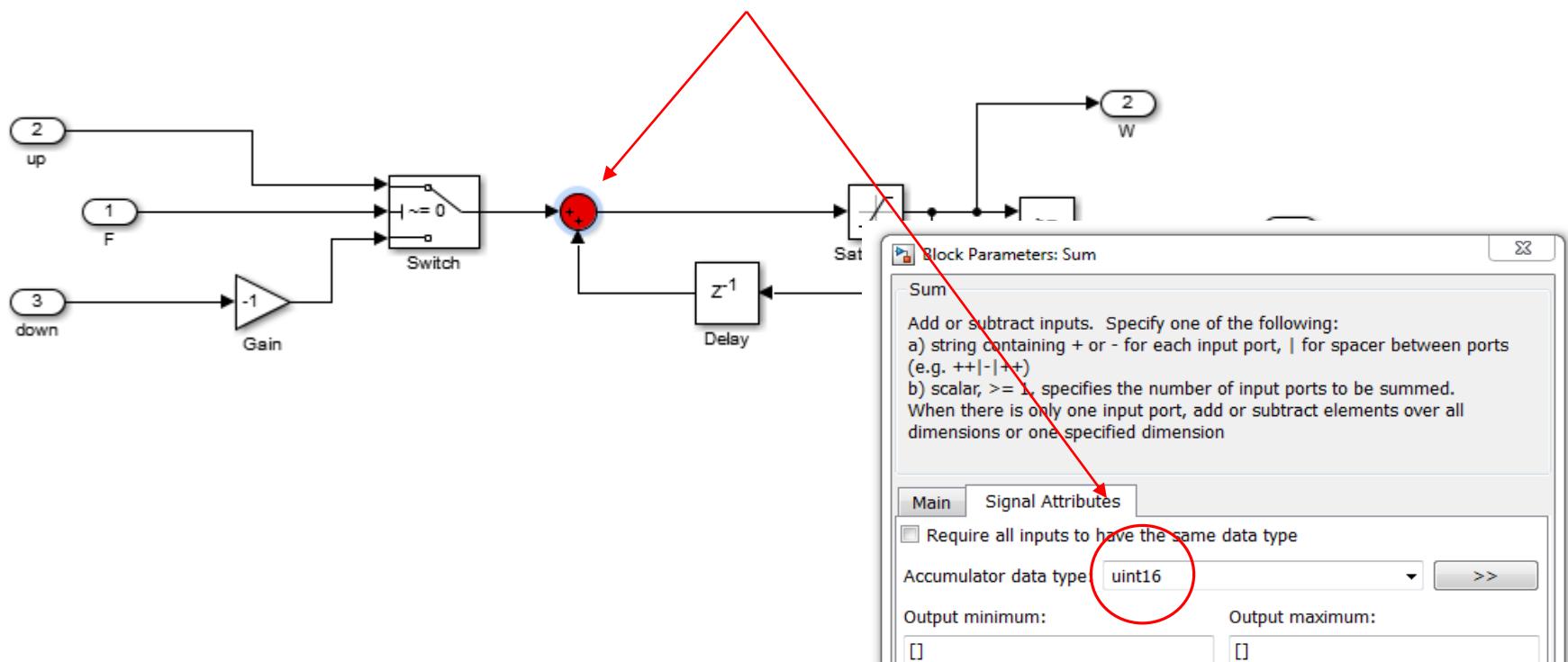
Initial Design Error

The initial design we had made was wrong

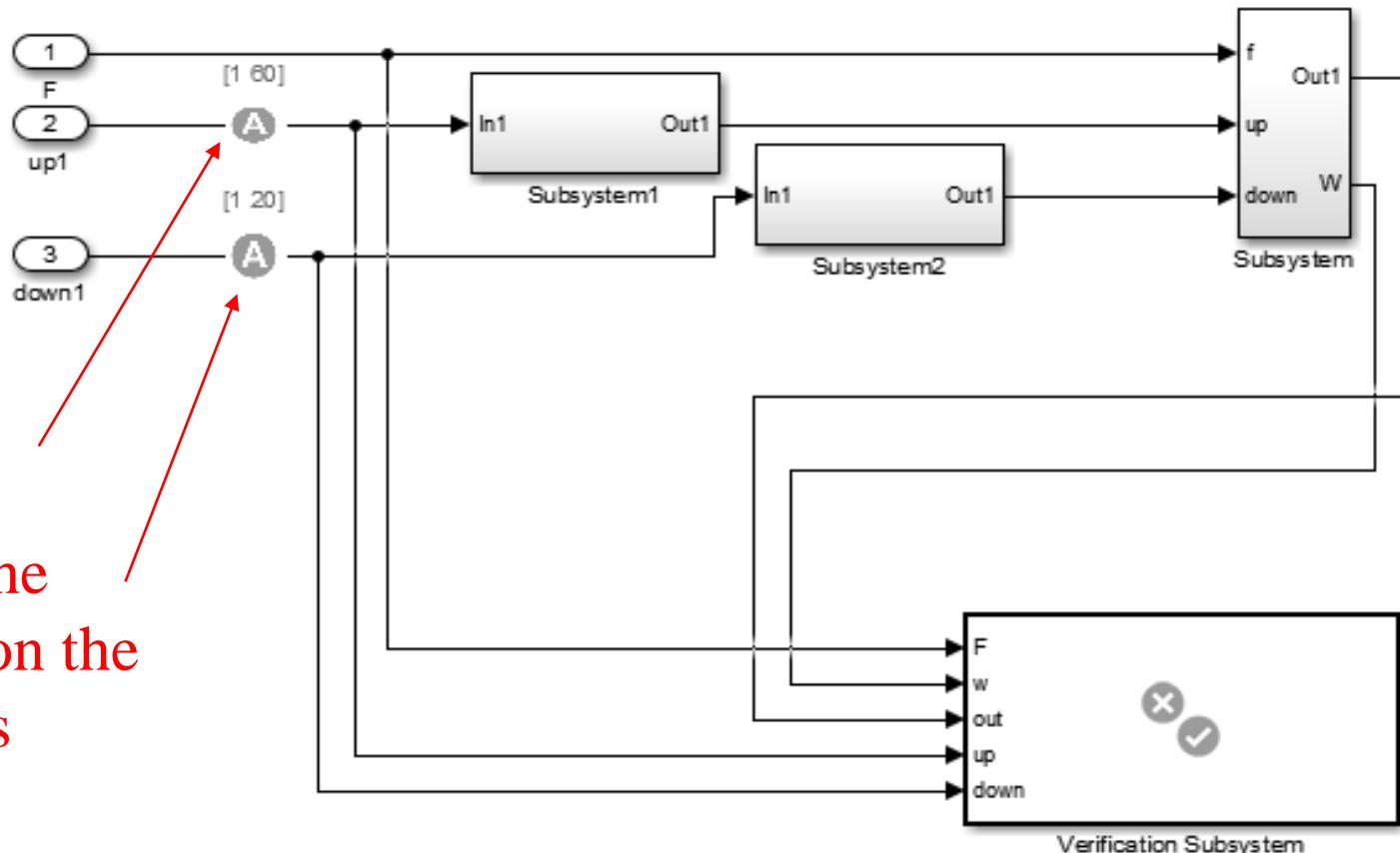


Error in Flight Control

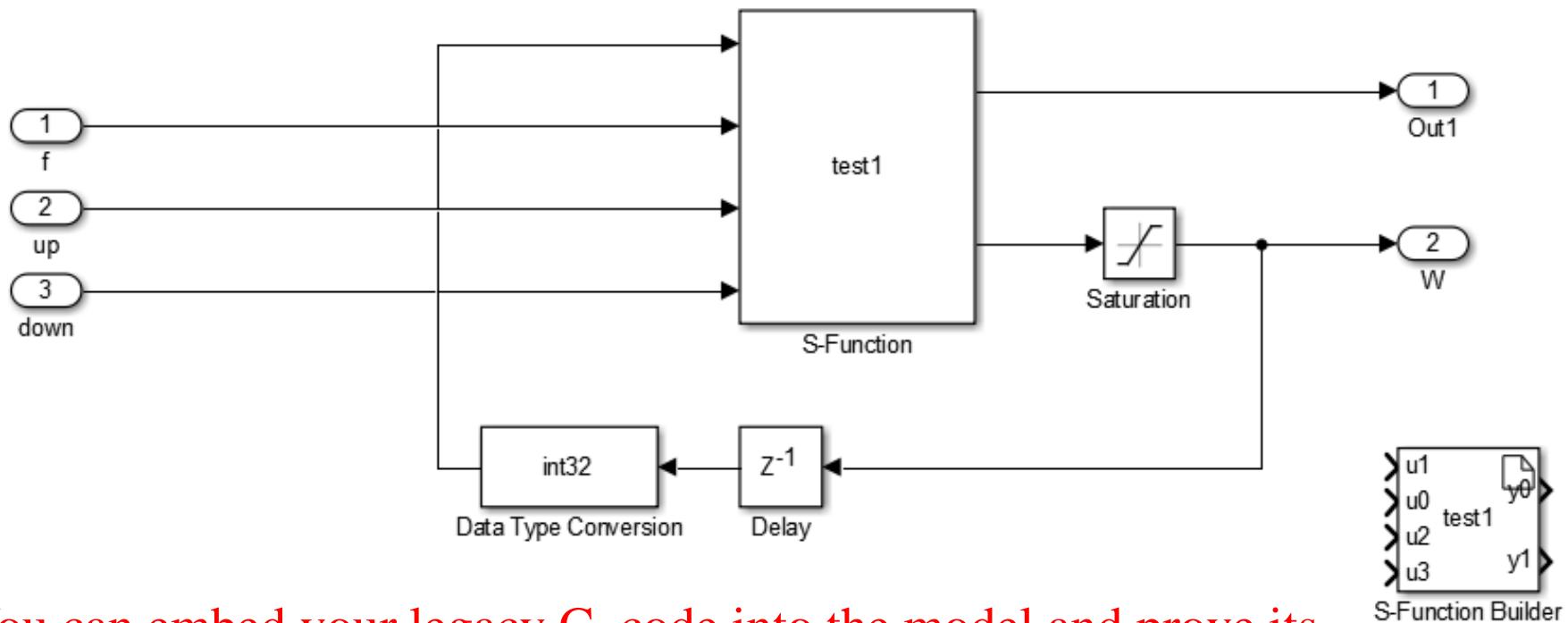
In one of the flight control code there was an unsigned integer error which is replicated in Simulink



SLDV Setup



Legacy S Function in SLDV

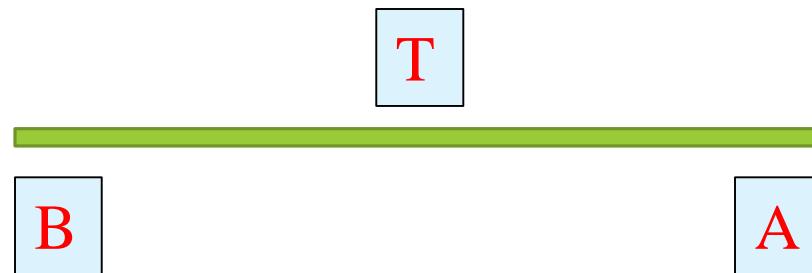


You can embed your legacy C code into the model and prove its correctness. This brought out the UINT16 error in the counter!

Design error

This is a design error in an aerospace system camouflaged as a railway problem. The error was found during the Verification and Validation phase.

The train can be AT_A or AT_B or IN_BET (in between) A and B or INDETER (indeterminate state if there is an error). It can be only one state at a time.



Requirements

The system shall set AT_A TRUE if sensor A is TRUE

The system shall set AT_B TRUE if sensor B is TRUE

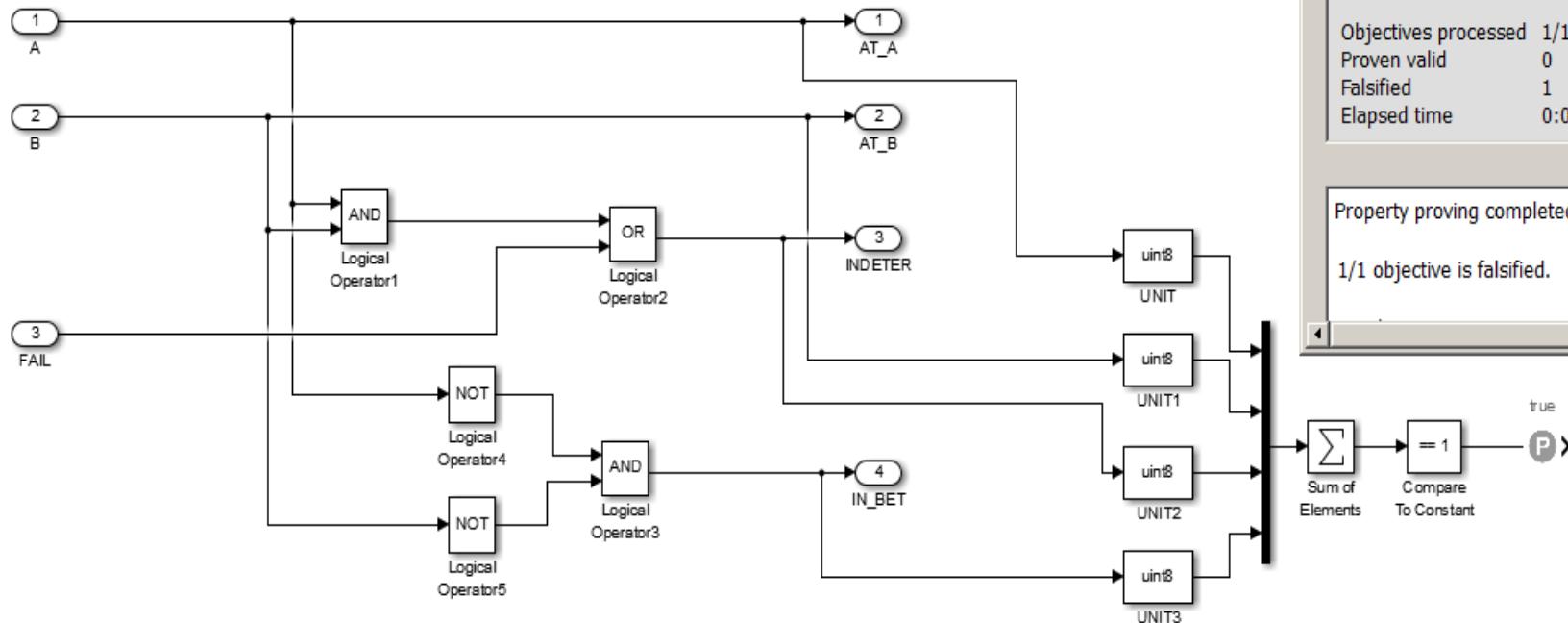
The system shall set IN_BET TRUE if sensor A is FALSE and sensor B is FALSE

The system shall set INDETER TRUE if (sensor A is TRUE AND sensor B is TRUE) or FAIL is TRUE

These are the text requirements provided for the design. The Formal assertion was only one of the states shall be TRUE at a time.

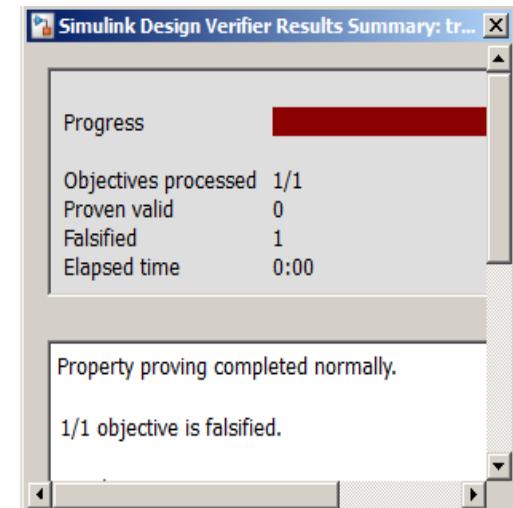
System model

Assertion: sum of all four Booleans should be 1
(only one true)



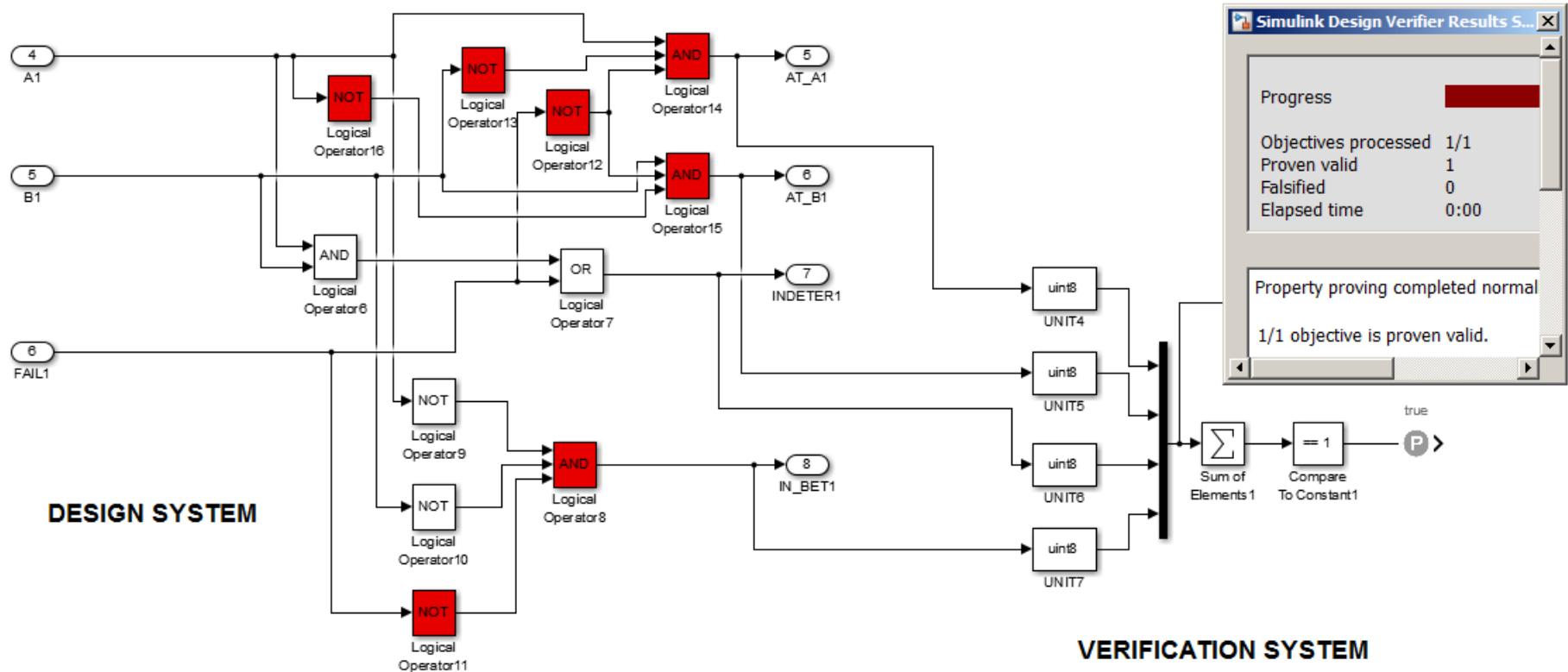
DESIGN SYSTEM

VERIFICATION SYSTEM



Corrected model

The design is proven valid



Correct Requirements

The system shall set AT_A TRUE if sensor A is TRUE AND (NOT FAIL) AND sensor B is FALSE

The system shall set AT_B TRUE if sensor B is TRUE AND (NOT FAIL) AND sensor A is FALSE

The system shall set IN_BET TRUE if sensor A is FALSE AND sensor B is FALSE AND (NOT FAIL)

The system shall set INDETER TRUE if (sensor A is TRUE AND sensor B is TRUE) or FAIL is TRUE

Using SLDV one can look at the design and requirements very early much before the coding and verification and come up with design issues.

Design Insights

- The SLDV model of the alternate On Ground brought out many errors in the initial implementation.
- The initial design had a hysteresis for the CAS and RADALT separately. This was then used to find the Alternate On Ground signal.
- This was erroneous and required a rethinking. SLDV brings out these sort of issues very early in the design phase adding to the correctness of the design.
- Surveys indicate that the errors are more often in the requirements rather than the implementation. SLDV helps in this.
- Brainstorming with the team on what exactly is the requirement is an additional positive input of using SLDV. Issues like what should happen in the first frame is a good discussion point.

Lessons Learnt

- The errors found earlier were due to extensive testing. SLDV is an equivalent to this extensive test activity.
- It is scalable. From simple blocks to collection of blocks that define a specific system requirement – SLDV delivers.
- SLDV can take a large time sometimes if the states are large. This would be true of a random test case also perhaps. If SLDV can say after 2 days that there are no errors then there is a very good chance that there are no design errors. The autopilot case is a good example as all combination of states, triggers and conditions have been exercised and there were no errors. This is full factorial test activity. SLDV brought out the same!

Lesson Learnt

- Formal methods are a way forward for this industry. Formal methods need to be used very early in the design stage and not at the verification stage. This has to be enforced by an independent design validation team like the independent verification team.
- In the Indian LCA program the control law design was validated by an independent team of academics. This sort of independence is a must (though perhaps not necessarily with the academia). A mathematical model of the system behavior and formal validation will go a long way in ensuring safety.
- In case of the autopilot the assertions can be easily coded as a safety check into the system itself. Assertions as part of the system design may be the future of ensuring safety and reliability in systems.

Tips

Make your assertion based on the required behavior. Let it not be for each and every design requirement. It is okay to make it for each and every requirement but it is a lot of work. A top level abstraction – that subtle behavioral nuance is what one should look for. It is difficult but not impossible.

Tips

Implementing code in Matlab or C language into a formal model is a very powerful way of validating your design and code. All library C code can be validated along with their models in this manner. A formally proven model and code can be used across programs.

Tips

You can validate your models and code with mutants or by deliberate injection of error into the model. This can serve two objectives. It serves as a way to qualify your formal tool. It serves as a benchmark – I can validate my design against such and such errors. You can look at some mutants here.

<http://www.mathworks.com/matlabcentral/profile/authors/59874-24-natasha-jeppu>

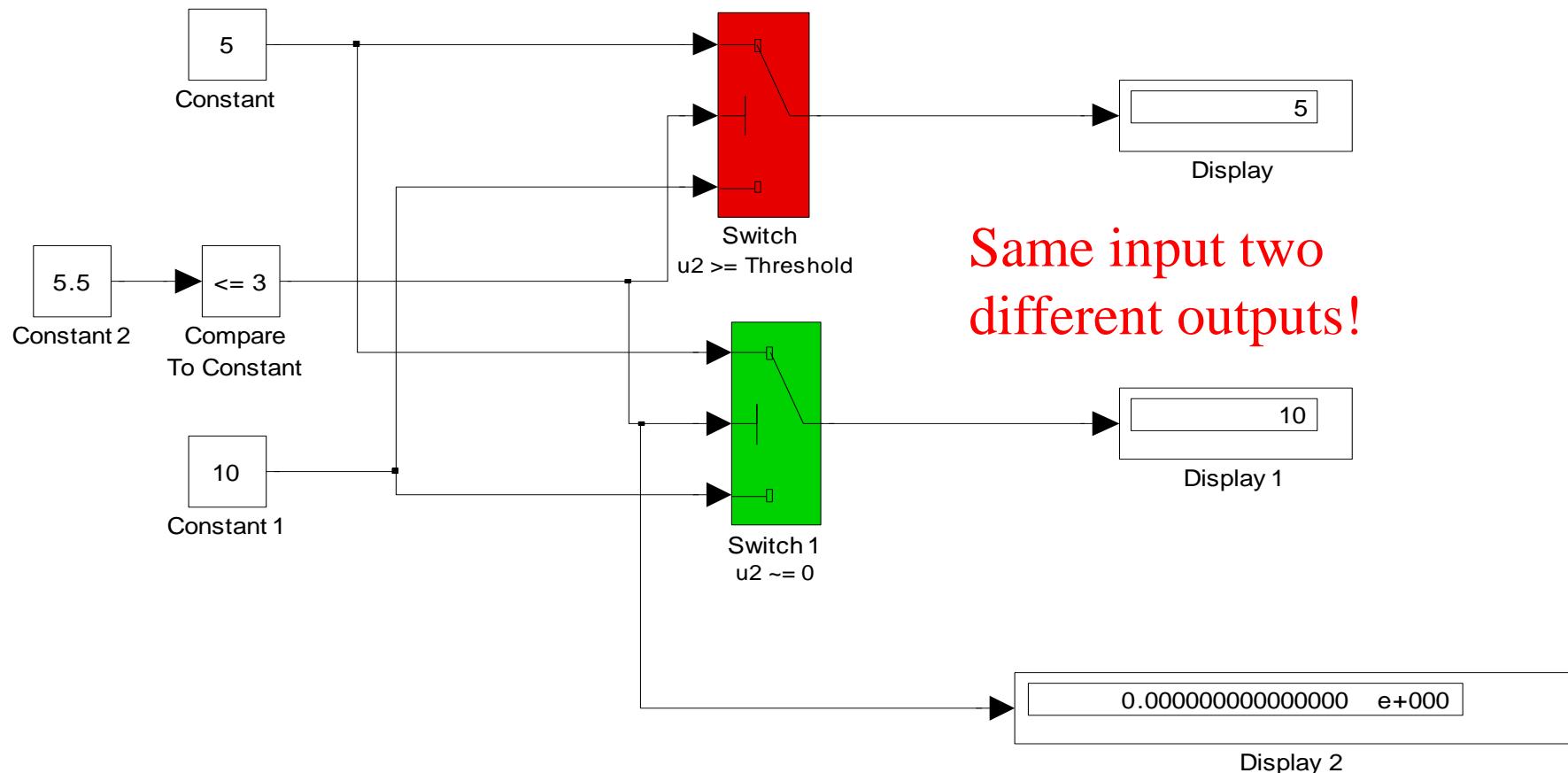
Modelling Guidelines



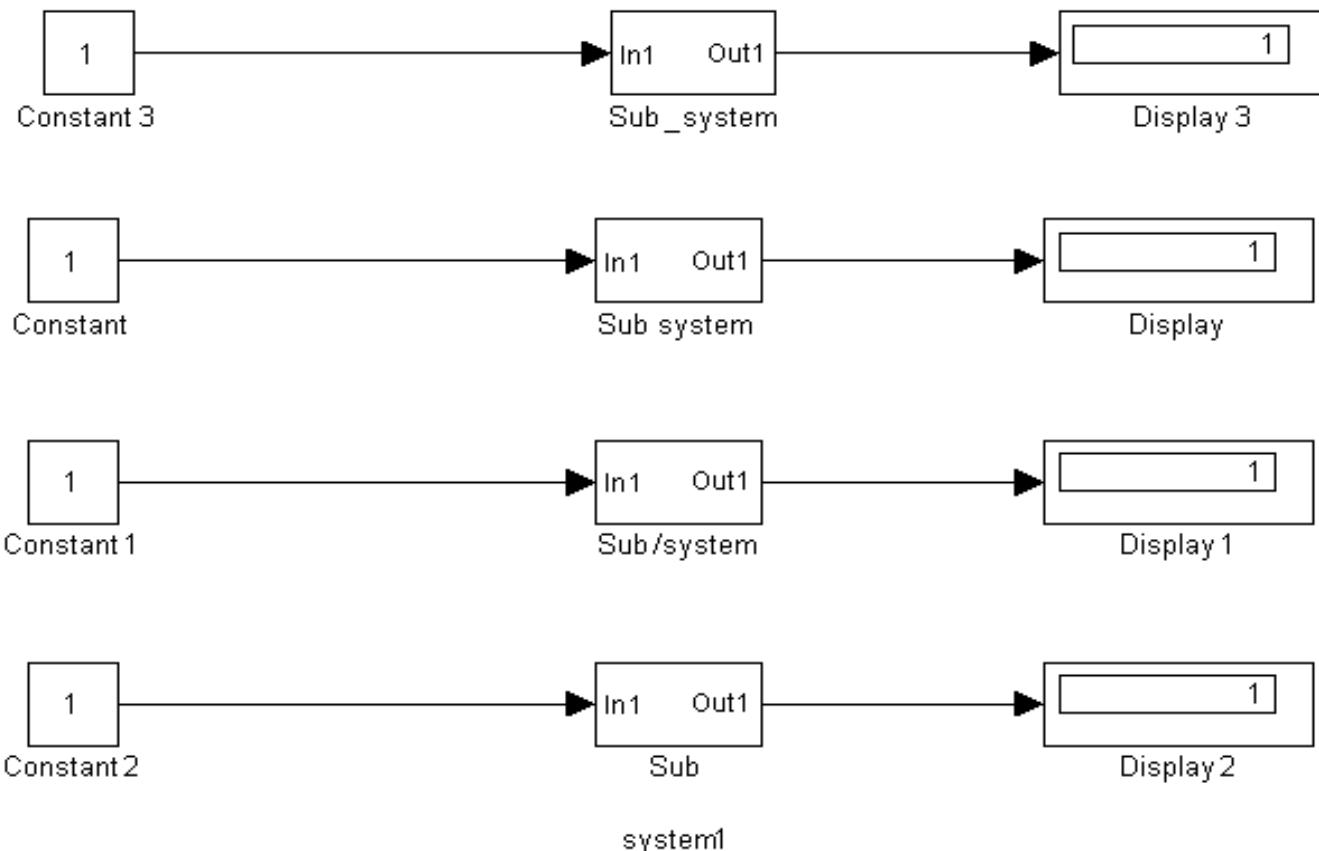
Follow guidelines

- While developing models always follow the guidelines. These may look like cosmetics but there are small issues which crop up due to the errors we make by not following guidelines.
- Two major mistakes that we made was to use `> Threshold` in switches and to name subsystems with spaces and carriage returns. We have suffered for these while automating and developing tools. Switches behave differently with “`> Threshold`” and “`~=0`”
- Mathworks has published these guidelines and there are checkers available to see if you have done so.

≈ 0 in switches



Model naming



Each model
is named
differently.
See how
they are
addressed
in Matlab
in the next
slide,.

Model naming

logsout =

Simulink.ModelDataLogs (NamingStd):

Name	Elements	Simulink Class
('Sub system1')	1	SubsysDataLogs
('Sub system')	1	SubsysDataLogs
('Sub/system')	1	SubsysDataLogs
Sub_system	1	SubsysDataLogs

In case you want to get the names for an automated processing you get into problems. You have to take care of the spaces, the "/". You have to use (). It complicates the process. A simple “_” works best!

Mathworks Guidelines

MathWorks® Products Solutions Academia Support Community Events Company

Contact Us How To Buy Log In

Documentation

Search R2016a Documentation Documentation ▾

CONTENTS Close

◀ All Products

◀ Simulink

Getting Started with Simulink

Modeling

Simulation

Performance

Component-Based Modeling

Modeling Guidelines

MAAB Control Algorithm Modeling

High-Integrity System Modeling

Code Generation

Large-Scale Modeling

Complex Logic

Physical Modeling

Signal Processing

Model Upgrades

Block Creation

Supported Hardware

Modeling Guidelines

Application-specific guidelines for model architecture, design, and configuration

Modeling guidelines help you develop models and generate code using Model-Based Design with MathWorks® products. Applying these guidelines can improve the consistency, clarity, and readability of your models. The guidelines also help you to identify model settings, blocks, and block parameters that affect simulation behavior or code generation.

Frequently Viewed Topics

[Model Advisor Checks for High-Integrity Modeling Guidelines](#) [Modeling Guidelines for Stateflow Charts](#)

[Model Advisor Checks for MAAB Guidelines](#) [Componentization Guidelines](#)

MAAB Control Algorithm Modeling
MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB®, Simulink®, and Stateflow®

High-Integrity System Modeling
Modeling guidelines for high-integrity systems

Code Generation
Guidelines and factors to consider for code generation

Large-Scale Modeling
Model architecture for large models and multi-user development teams

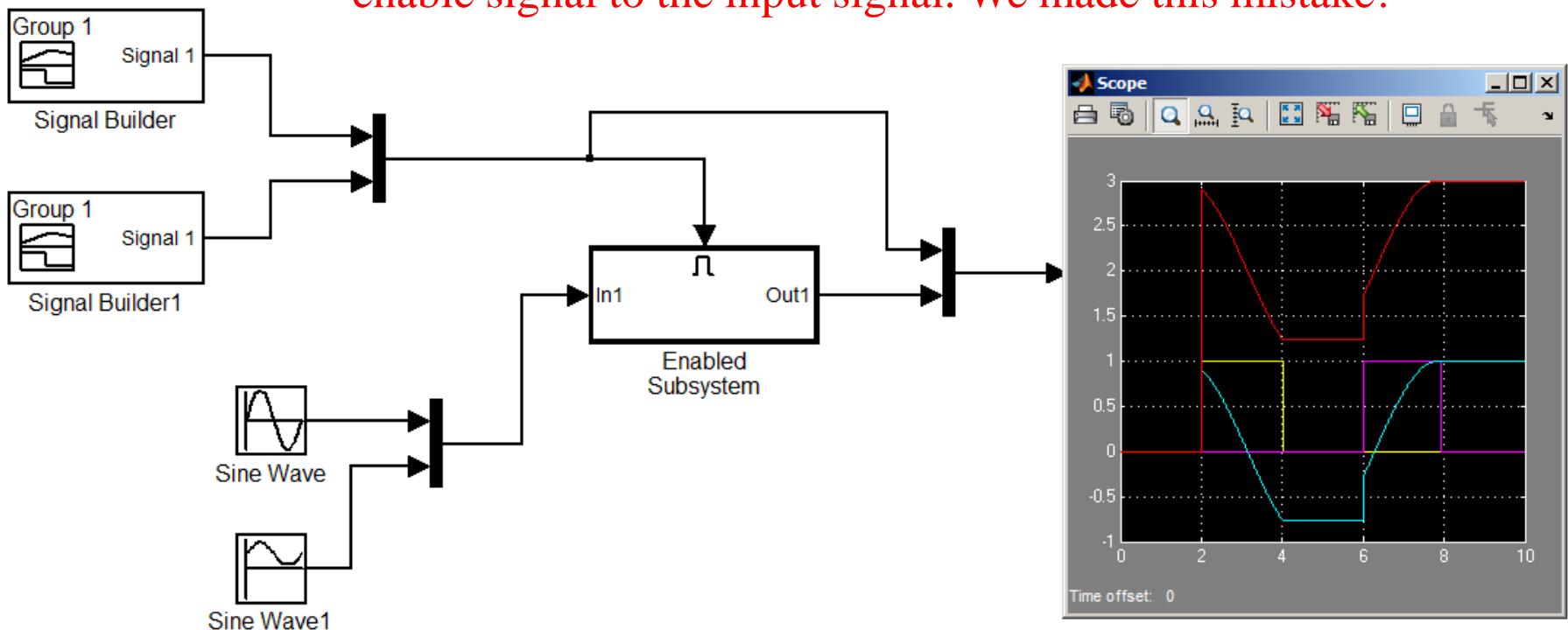
Complex Logic
Guidelines for modeling complex logic using state diagrams

Modelling guidelines help a lot!

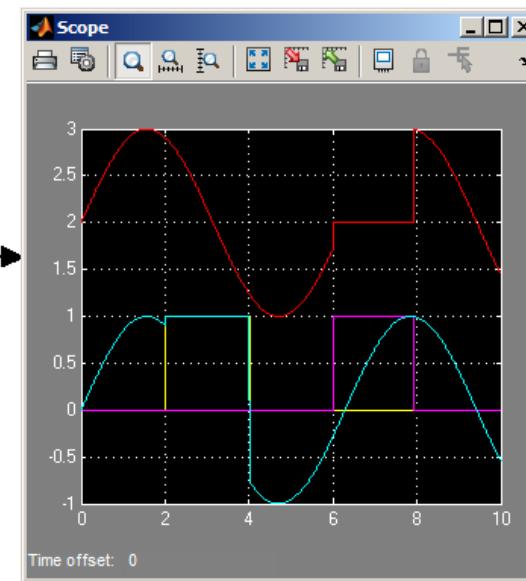
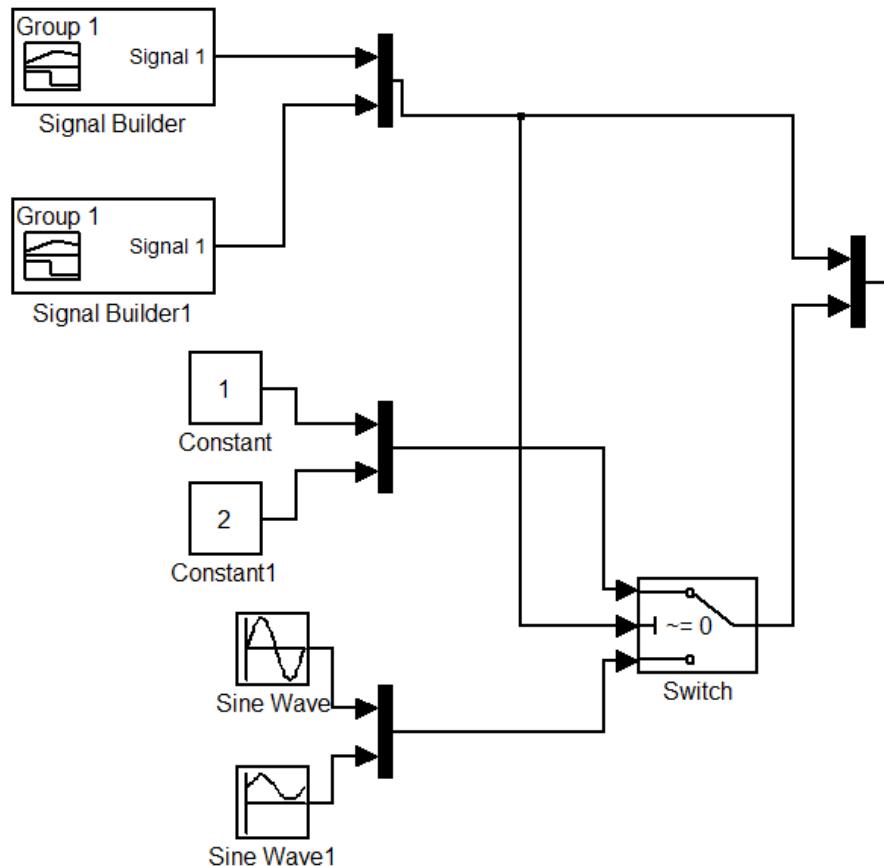
<http://in.mathworks.com/help/simulink/design-guidelines.html?requestedDomain=www.mathworks.com>

Enable Subsystem

The enable subsystem responds to any enable input > 0 .
Unlike other blocks like switches there is no association of enable signal to the input signal. We made this mistake!



Switch with Mux



Each switching Boolean input in switch is associated with the signal

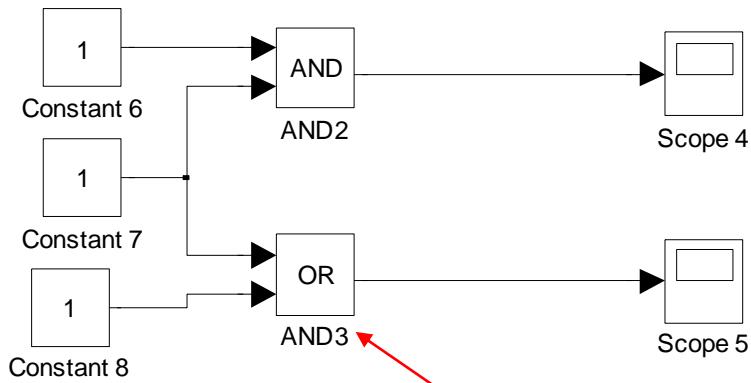
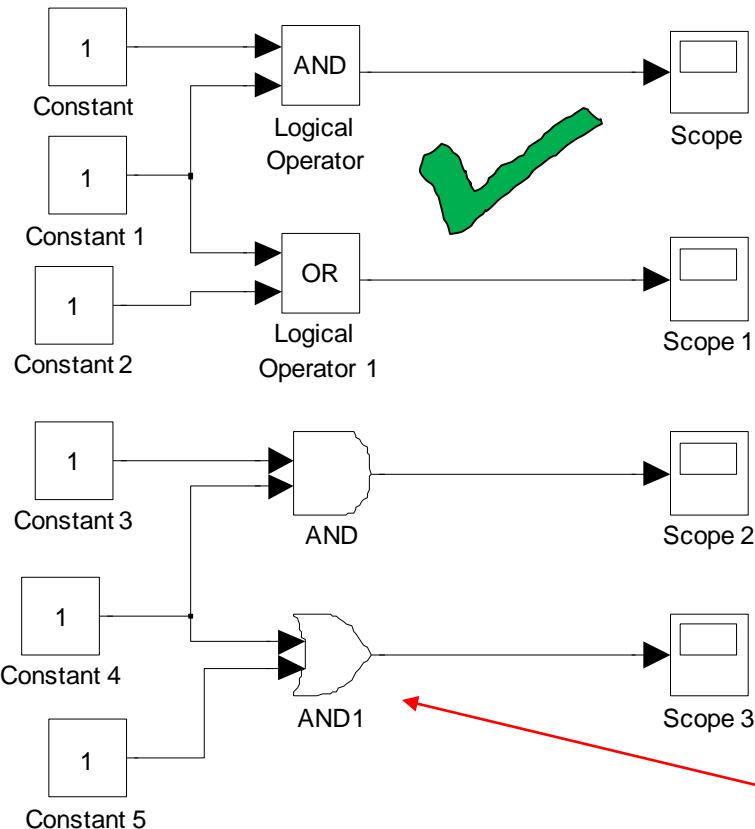
Copy Paste Blocks

- We have a tendency to copy paste blocks. This is made very easy in Simulink. Press Click block, Press [CTRL], drag. You have clones of the block for use in the model.
- This leads to problems. Simulink automatically names the blocks by adding one. AND block dragged twice will create AND1 and AND2 blocks.
- We have the freedom to click on these blocks and convert the AND block to an OR block. But do we change the name to OR.

Normally we do not.

- This can lead to readability problems.

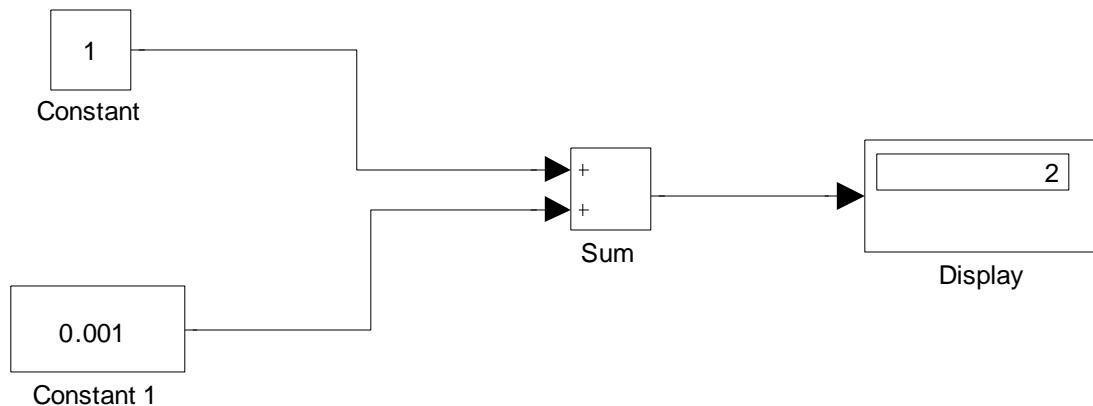
Copy Paste Blocks



The other blocks can create problems during reviews. What is displayed in side is correct. But it is possible to change what is displayed. Ideally the icon shape parameter should be set to “distinctive”.

Add Block

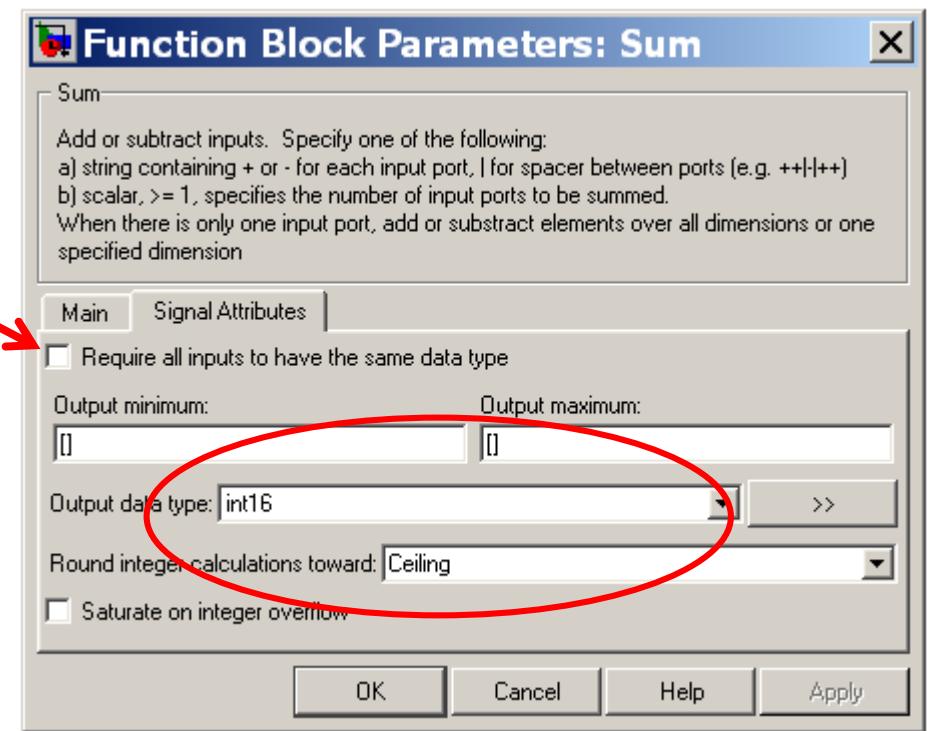
We found this issue because of copy paste. The add block internal was changed for some test to int16



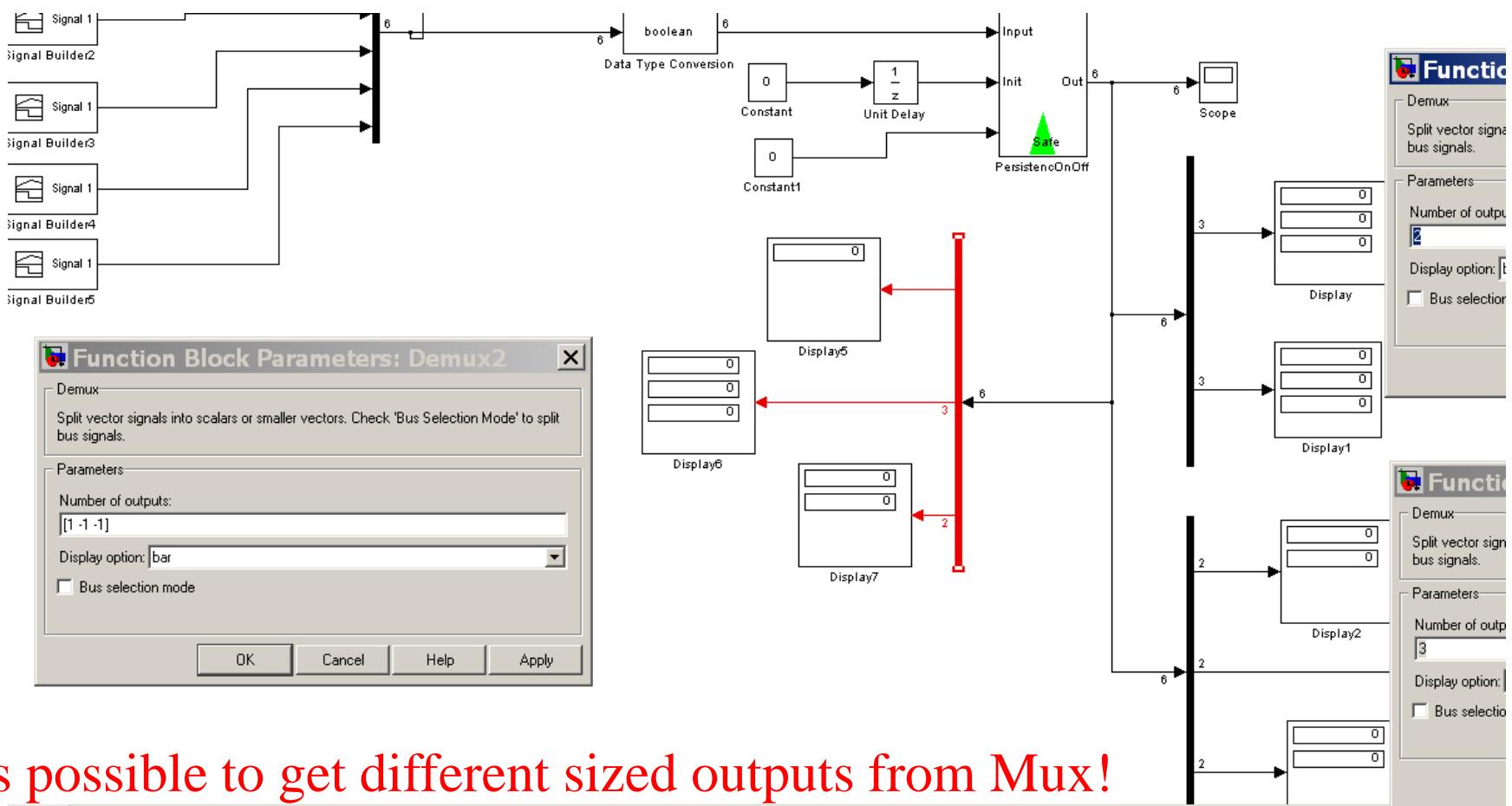
$$1.0 + 0.001 = 2$$

Add Block

It is a good practice to click require all inputs to have the same data type check box. The qualified blocks have this set.

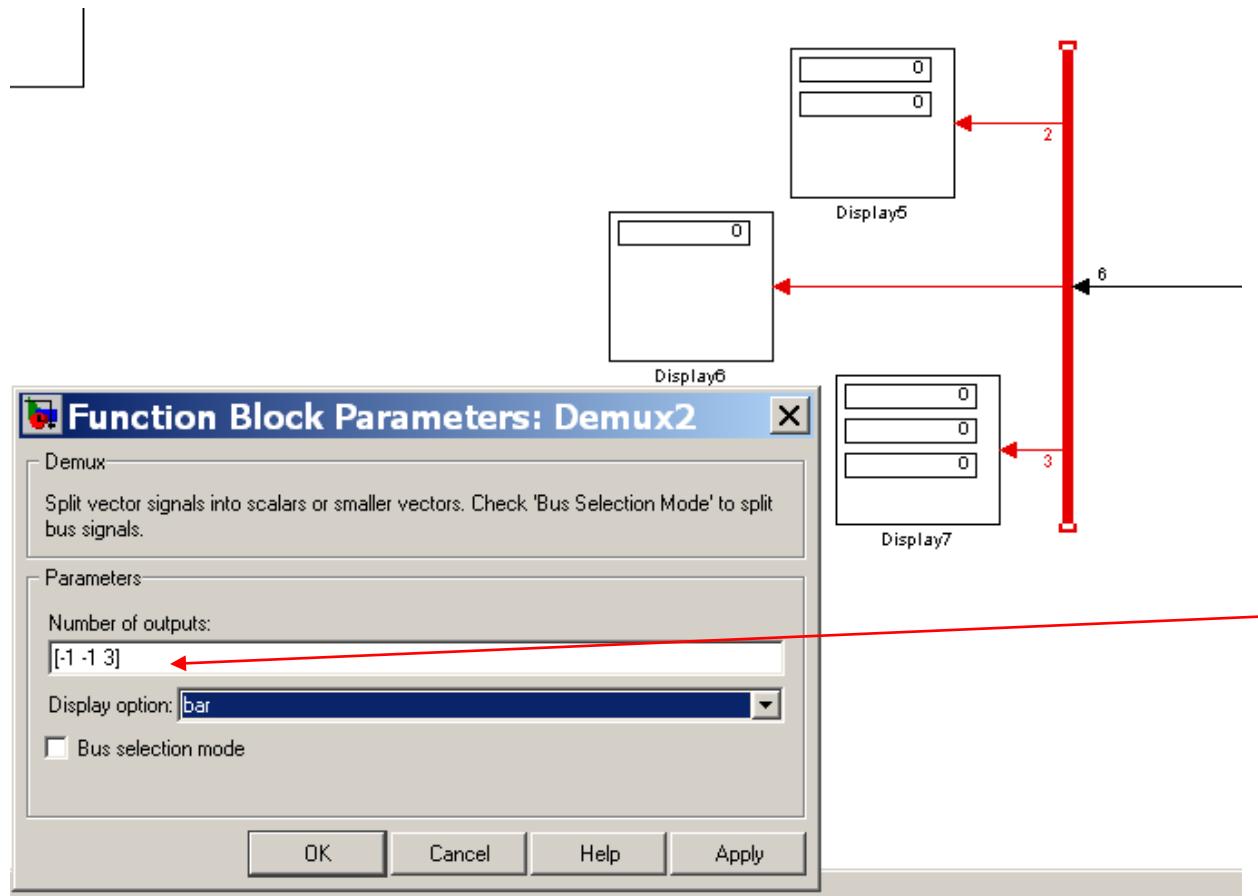


Mux Issues



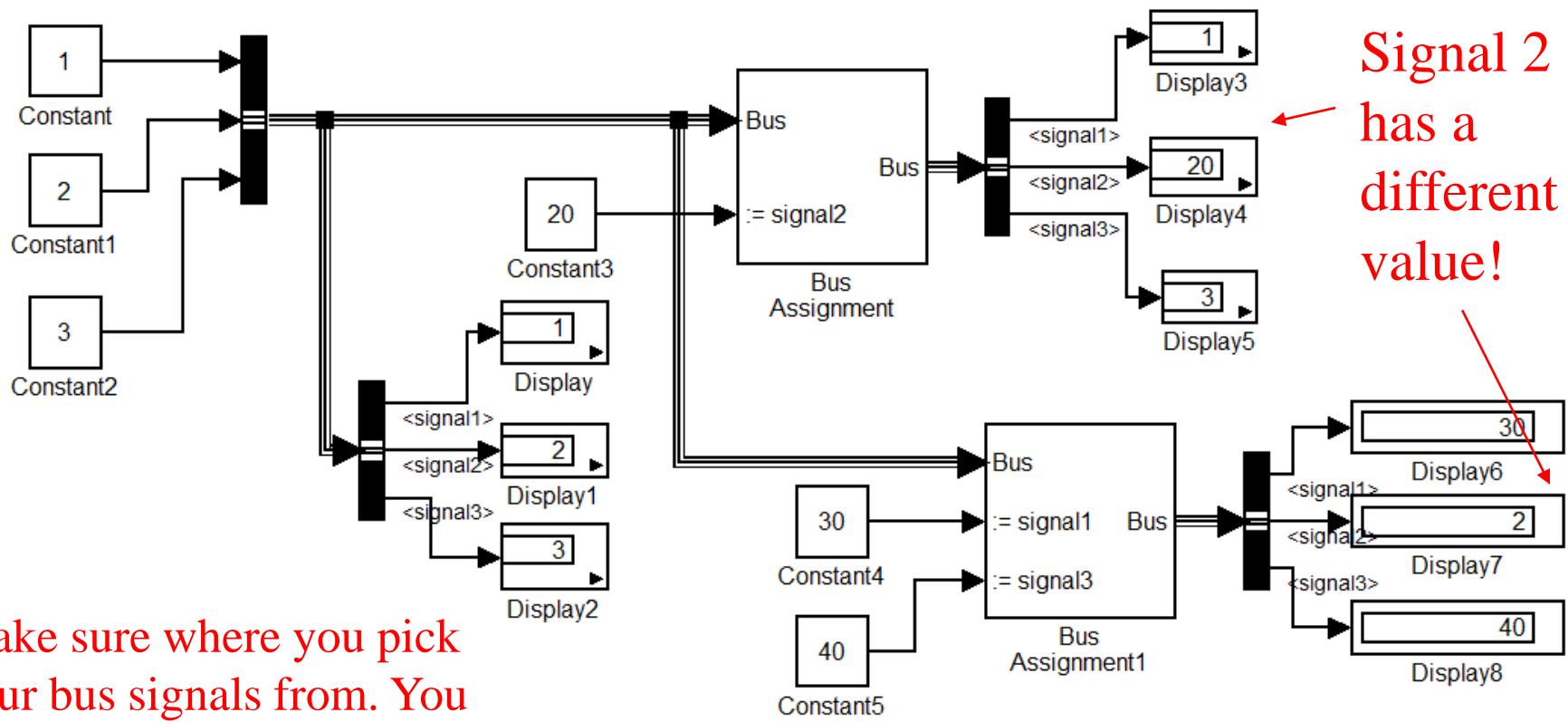
It is possible to get different sized outputs from Mux!

Mux Issues



Setting number of outputs to -1 lets Simulink decide what to give you. It is better to know beforehand what you are in for! Set display port signal dimensions to see the number.

Bus issues

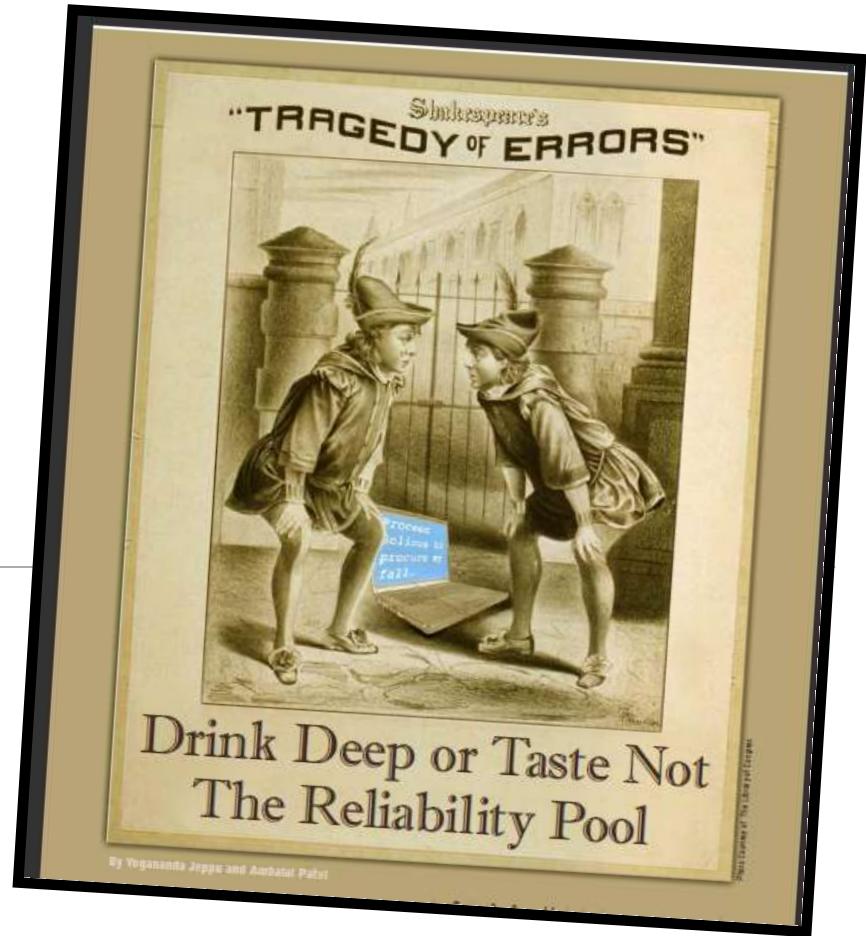


Long variable names

```
variable_name_is.pretty_long_123456789012345678901234567890123 =  
100  
  
>> variable_name_is.pretty_long_1234567890123456789012345678901234567890=200  
Warning: 'variable_name_is.pretty_long_1234567890123456789012345678901234567890'  
exceeds the MATLAB maximum name length of 63 characters and will be truncated to  
'variable_name_is.pretty_long_123456789012345678901234567890123'.  
  
variable_name_is.pretty_long_123456789012345678901234567890123 =  
200
```

We have had problems with very long variables names and the truncation that is automatically done. Limit name length!

Best Practices



To Err is Human

- We have found that a good threshold is to use the formula
- If the $|Output|$ signal is > 1.0 then divide the error by the signal
- If it is ≤ 1.0 then take the computed error itself
- We used a threshold of 0.0002 for the pass/fail and found it to be adequate for our processor and precision used
- This has been reported in open literature so feel free to use it!

Testing Tantras

- Automate the complete process from DAY 1 – test generation, test execution, download, analysis, reporting
- Analyze every case in the first build – Painful but essential. This gives you an insight into the working
- Analyze failed cases and as you have the code, do a debug to some level – do not send error reports (test case could be wrong!) [Pssst... We face it regularly]
- Have a configuration control mechanism for test cases, reports, open/closed PRs
- Develop a front end for the test activity eases the whole process

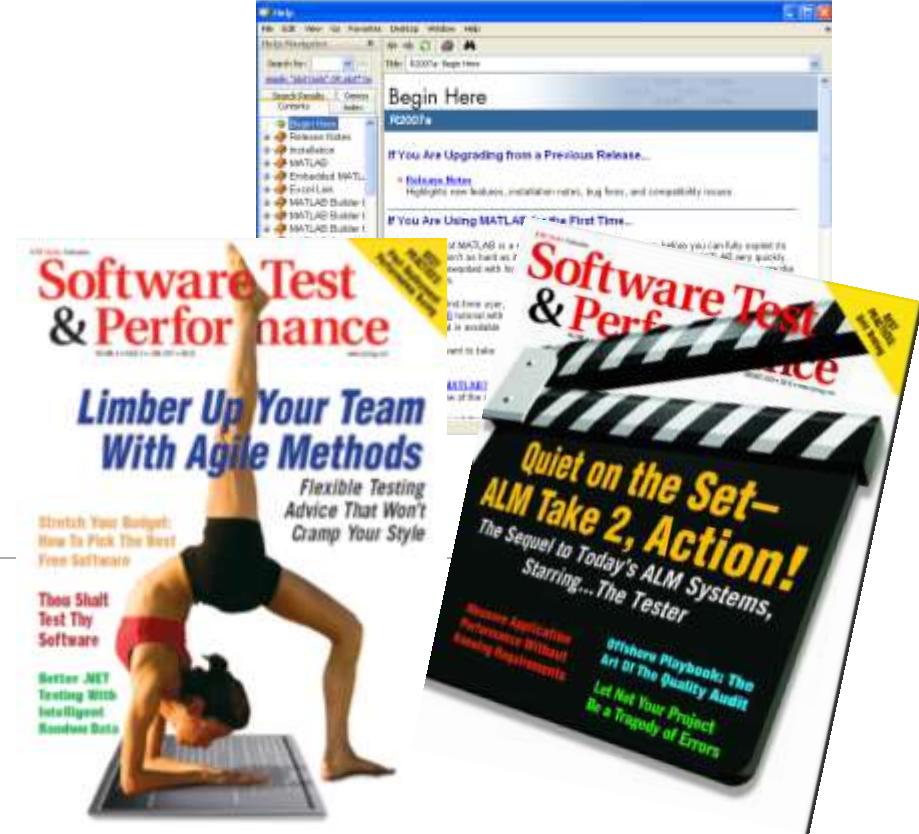
Testing Mantras

- Eyeball the Requirements and the Model. If allowed look at the Model and Code (Make the tests based on the Model). This first step will bring out lot of errors. **Preserve Independence.**
- Errors, like the bugs, are found at the same place (behind the sink!). Try to search there first. You will get a lead on the development guys. Smart Testing!
- It is very useful if you have a systems guy close by. Lot of issues get solved across the partition
- Have tap out points in the model and code. They are extremely useful in debugging especially at system level

Last Words

- Children are born true scientists. They spontaneously experiment and experience and experience again. They select, combine and test, seeking to find order in their experiences: “Which is the mostest? Which is the leastest?” They smell, taste, bite and touch-test for hardness, softness, springiness, roughness, smoothness, coldness, warmness: they heft, shake, punch, squeeze, push, crush, rub and try to pull things apart. – R. Buckminster Fuller
- Let us experiment with Model Based Testing – there is so much to experience here!

References



References

- RTCA, 1992, "Software Considerations in Airborne Systems and Equipment", DO-178B, Requirements and Technical Concepts for Aviation, Inc.
- International Electrotechnical Commission, IEC 61508, "Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems", draft 61508-2 Ed 1.0, 1998
- UK Ministry of Defense. Defense Standard 00-55: "Requirements for Safety Related Software in Defense Equipment", Issue 2, 1997
- UK Ministry of Defense. Defense Standard 00-56: "Safety Management Requirements for Defense Systems", Issue 2, 1996
- FAA System Safety Handbook, Appendix C: Related Readings in Aviation System Safety, December 30, 2000

References

- YV Jeppu, CH Harichoudary, Wg Cdr BB Misra, "Testing of Real Time Control System: A Cost Effective Approach" SAAT 2000, Advances in Aerospace Technologies, Hyderabad, India
- Y V Jeppu, Dr K Karunakar, P S Subramanyam , "A New Test Methodology to Validate and Verify the Control Law on the Digital Flight Control Computer" 3rd Annual International Software Testing Conference 2001, Bangalore, India
- YV Jeppu, K Karunakar, PS Subramanyam, "Flight Clearance of Safety Critical Software using Non Real Time Testing", American Institute of Aeronautics and Astronautics, ATIO, 2002, AIAA-2002-5821
- YV. Jeppu, K Karunakar and P.S. Subramanyam, "Testing Safety Critical Ada Code Using Non Real Time Testing", Reliable Software Technologies ADA-Europe 2003, edited by Jean-Pierre Rosen and A Strohmeier, Lecture Notes in Computer Science, 2655, pp 382-393.
- S.K. Giri, Atit Mishra, YV Jeppu, K Karunakar, "A Randomized Test Approach to Testing Safety Critical Code" presented as a poster session at the International Seminar on "100 Years Since 1st Powered Flight and Advances in Aerospace Sciences", Dec 2003.
- Sukant K. Giri, Atit Mishra, Yogananda V. Jeppu and Kundapur Karunakar, "A Randomized Test Approach to Testing Safety Critical Ada Code", Reliable Software Technologies, Ada-Europe-2004, edited by Albert Lliamosi and Alfred Strohmeier, Lecture Notes in Computer Science, 3063, pp 190-199.

References

- Rajalakshmi K, Jeppu Y V, Karunakar K, "Ensuring software quality -experiences of testing Tejas airdata software". *Defence Science Journal* 2006, 56(1), pp13-19.
- Yogananda V. Jeppu, K. Karunakar, Prakash R Apte "Optimized Test Case Generation Using Taguchi Design of Experiments", 7th AIAA Aviation Technology, Integration and Operations Conference (ATIO), September 2007 (accepted for publication)
- Rohit Jain, Srikanth Gampa, Yogananda Jeppu, "Automatic Flight Control System For The Saras Aircraft" HTSL Technical Symposium, Bangalore, India, December 2008
- Yogananda Jeppu, "Automatic Testing of Simulink Blocks using Orthogonal Arrays" 2009 Engineering Conference, Moog Inc, 26 May 2009
- YV Jeppu, "The Tantras and Mantras of Testing", *Software Test and Performance Magazine*, Sep 2005, pp 39-43
- Yogananda Jeppu, "Thou Shalt Experiment With Thy Software", *Software Test and Performance Magazine*, June 2007
- Sukant K. Giri, Atit Mishra, Yogananda V. Jeppu and Kundapur Karunakar "Stress Testing Control Law Code using Randomised NRT Testing" 43rd American Institute of Aeronautics and Astronautics, Aerospace Sciences Meeting and Exhibit, 10 - 13 Jan 2005 - Reno, Nevada, AIAA 2005-1253
- Yogananda Jeppu and Ambalal Patel, "Let Not Your Project Become a Tragedy of Errors", *Software Test & Performance magazine*, January 2008

References

- System Safety Handbook http://www.faa.gov/library/manuals/aviation/risk_management/ss_handbook/
- Hazard Analysis http://en.wikipedia.org/wiki/Hazard_analysis
- Jeppu, Y., "Flight Control Software: The Mistakes We Made and the Lessons We Learnt," *Software, IEEE*, vol.PP, no.99, pp.1,1, 0, doi: 10.1109/MS.2013.42
- Atit Mishra, Manjunatha Rao, Chethan CU, Vanishree Rao, Yogananda Jeppu, and Nagaraj Murthy. 2013. An auto-review tool for model-based testing of safety-critical systems. In *Proceedings of the 2013 International Workshop on Joining AcadeMiA and Industry Contributions to testing Automation (JAMAICA 2013)*. ACM, New York, NY, USA, pp 47-52.

K. Samatha, Shreesha Chokkadi, Jeppu Yogananda, A Genetic Algorithm Approach for Test Case Optimization of Safety Critical Control, *Procedia Engineering*, Volume 38, 2012, Pages 647-654,

Cu, C.; Jeppu, Y.; Hariram, S.; Murthy, N.N.; Apte, P.R., "A new input-output based model coverage paradigm for control blocks," *Aerospace Conference, 2011 IEEE*, vol., no., pp.1,12, 5-12 March 2011

A Benchmark Problem for Model Based Control System Tests – 001

<http://www.mathworks.fr/matlabcentral/fileexchange/28952-a-benchmark-problem-for-model-based-control-system-tests-001>

References

MC/DC Test Case Generator, <http://www.mathworks.com/matlabcentral/fileexchange/37953-mcdc-test-case-generator>

A Benchmark Problem for Model Based Control System Tests – 002

<http://www.mathworks.in/matlabcentral/fileexchange/37973-a-benchmark-problem-for-model-based-control-system-tests-002>

<http://www.mathworks.in/matlabcentral/fileexchange/39720-safety-critical-control-elements-examples>

<http://www.mathworks.in/matlabcentral/fileexchange/41838-benchmark-problem-02-matlab-code>

Natasha Jeppu, Exploring Design Verifier, <http://in.mathworks.com/matlabcentral/fileexchange/48858-exploring-design-verifier>

Natasha Jeppu, Exploring Simulink Design Verifier - 2,

<http://in.mathworks.com/matlabcentral/fileexchange/51567-exploring-simulink-design-verifier-2>

Natasha Jeppu, Exploring Simulink Design Verifier 03,

<http://in.mathworks.com/matlabcentral/fileexchange/54945-exploring-simulink-design-verifier-03>

Yogananda Jeppu

yvjeppu@gmail.com

<http://in.linkedin.com/in/yoganandajeppu>



A “control system bug” which walked into our office one day.