Alan Wassyng

A Safety-Critical Software Approach to Hard Real-Time Applications

Safety-Critical Software

- Software required for a safety system
- Action initiated (or not initiated) by the software that can lead directly to loss of life
- In the Canadian Nuclear Industry, software that has a safety role is kept separate from control software

2

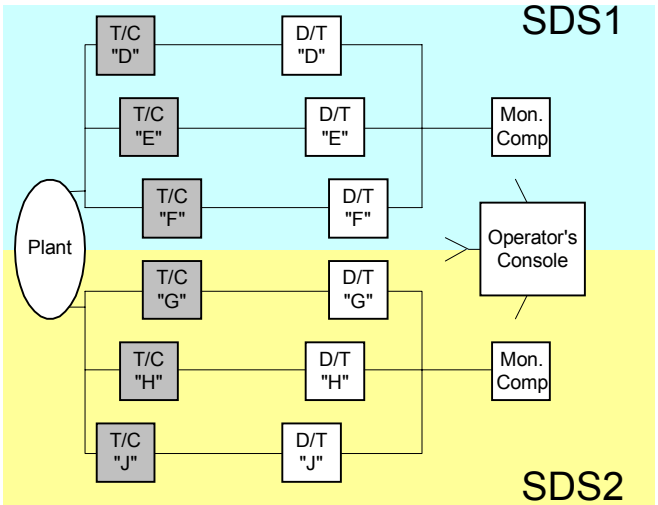
Safety-Critical Software

- The methodology we are going to talk about is applicable to a wide variety of applications:
  - Nuclear Generating Station shutdown systems (the example we will discuss)
  - Embedded applications in auto and avionics industries
  - Embedded applications in software controlled electric motors
  - Weapons systems ...

3

Safety-Critical Software

- NGS Shutdown System



T/C = Trip Computer

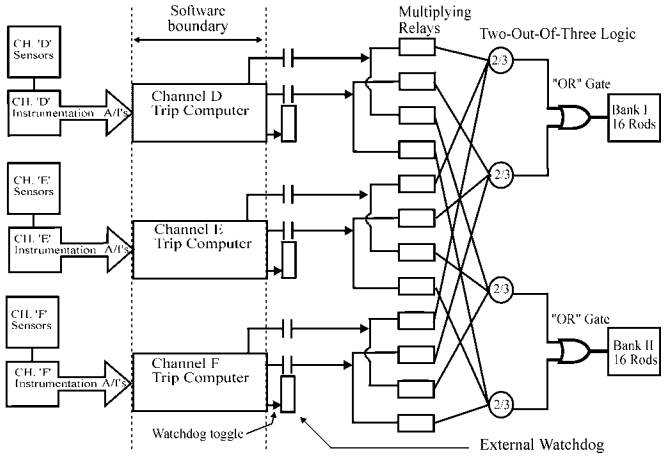
D/T = Display/Test Computer

Mon. Comp = Monitor Computer

4

Safety-Critical Software

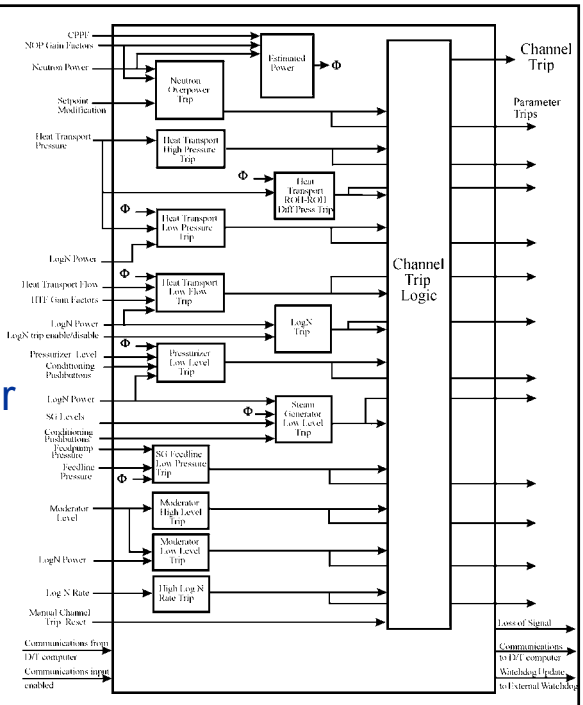
- Target System - SDS1



5

Safety-Critical Software

- An SDS1 Trip Computer



6

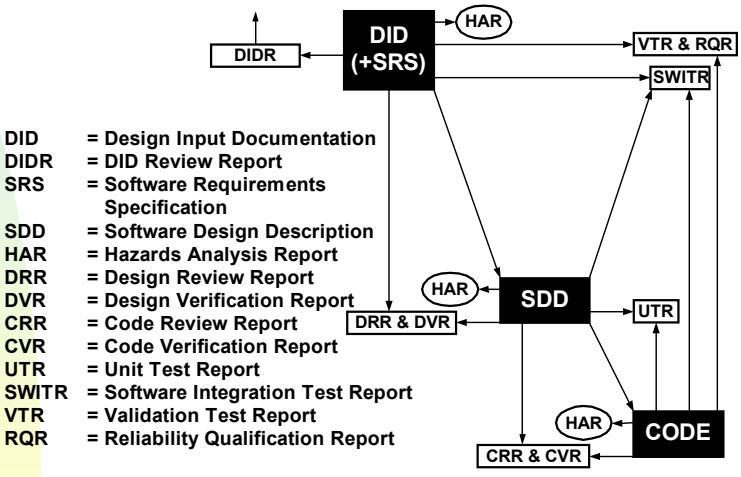
Safety-Critical Software

- SDS1 & SDS2: Real-Time Monitoring & Shutdown
  - These computer systems have hard deadlines in which they have to detect potential accident scenarios.
  - They also have hard deadlines in which they have to initiate alarms, and, if necessary, initiate shutdown of the reactor.

7

Safety-Critical Software

- Methodology - SDS1



DID = Design Input Documentation
DIDR = DID Review Report
SRS = Software Requirements Specification
SDD = Software Design Description
HAR = Hazards Analysis Report
DRR = Design Review Report
DVR = Design Verification Report
CRR = Code Review Report
CVR = Code Verification Report
UTR = Unit Test Report
SWITR = Software Integration Test Report
VTR = Validation Test Report
RQR = Reliability Qualification Report

8

Safety-Critical Software

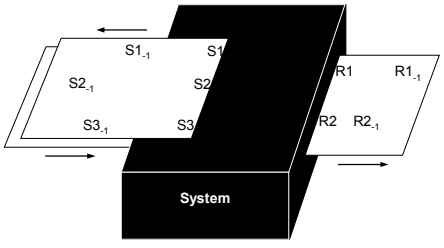
- SDS1
  - **T**rip **C**omputer **D**esign **R**equirements
  - **T**rip **C**omputer **D**esign **D**escription
  - **S**oftware **D**esign **D**escription
  - **S**oftware **D**esign **R**eview - (nd)
  - **S**oftware **D**esign **V**erification
  - **FORTRAN C**ode
  - **T**esting - not going to discuss (nd)
  - **C**ode **R**eview - (nd)
  - **C**ode **V**erification

9

Safety-Critical Software

TCDR

- **T**rip **C**omputer **D**esign **R**equirements (Mills' Black-Box)



Mills, H.D.: Stepwise refinement and verification in box-structured systems. Computer 21 (1988) 23-36

$R = f (S, S_h)$

S is the set of stimuli, S_h the set of stimulus history, R the set of responses

10

TCDR

Safety-Critical Software

- TCDR
  - Context diagrams
  - Stimuli & Responses
  - Constants
  - Main function tables
  - Natural language expressions
  - Tolerances, PTRs and TRs
  - Anticipated changes
  - Changes from previous freezes

11

TCDR

Safety-Critical Software

- We expect that all responses are described in terms of stimuli and stimuli history only.
- It is sometimes advantageous to allow response history to appear in functional descriptions.
- In deterministic systems, response history is always a representation of stimuli history.

12

TCDR

Safety-Critical Software

- We refer to stimuli as monitored variables, and responses as controlled variables.
- We prefix identifiers by a suitable character followed by _ to help identify the role of the identifier.
- m_ name is a monitored variable, c_name is a controlled variable, k_name is a constant, etc.

13

TCDR

Safety-Critical Software

- m_name represents value of the current instance of m_name.
- m_name_{-1} represents value of the previous instance of m_name.
- If m_name is time-continuous, there's an arbitrarily small time, δt, between m_name and m_name_{-1}.
- t_{now} represents current time.

14

TCDR

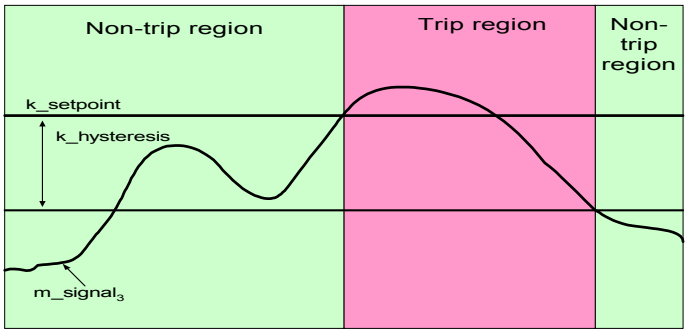
Safety-Critical Software

- If m_name is time-discrete, time between m_name and m_name_{-1} is t(m_name) - $t(m\_name_{-1})$. In general, t(var) returns time stamp of the instance of var.
- In a real system it will not be possible to represent $R = f(S,S_h)$ by a single function or function table.

15

TCDR

Safety-Critical Software



$f\_sensortrip_i$, i=1,..,4

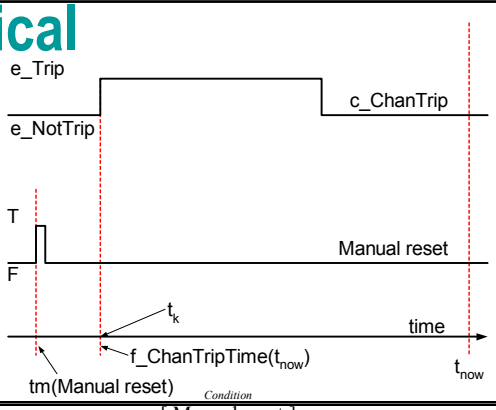
{For each i = 1,..,4}

Condition	*Result* **$f\_sensortrip_i$**
$k\_setpoint \le m\_signal_i$ *{i^{th} signal is now in the trip region}*	e_Trip
$k\_setpoint - k\_hysteresis < m\_signal_i < k\_setpoint$ *{i^{th} signal is now in the deadband region}*	No Change
$m\_signal_i \le k\_setpoint - k\_hysteresis$ *{i^{th} signal is now in the non-trip region}*	e_NotTrip

16

Safety-Critical Software

TCDR



Condition		Result: f_ChanTripTime
[Manual reset]		NT
NOT [Manual reset]	[There is a time t_k, $t_{manualreset} < t_k \le t_{now}$ where $c\_ChanTrip(t_k) = e\_Trip$] & NOT [There is a time t_i, $t_{manualreset} < t_i < t_k$ where $c\_ChanTrip(t_j) = e\_Trip$] *{t_k is the earliest time at which the channel was tripped after the most recent manual reset}*	t_k
	[NOT ($c\_ChanTrip(t) = e\_Trip$) for all t, $t_{manualreset} < t \le t_{now}$] *{No channel trip has occurred since the most recent manual reset}*	NT

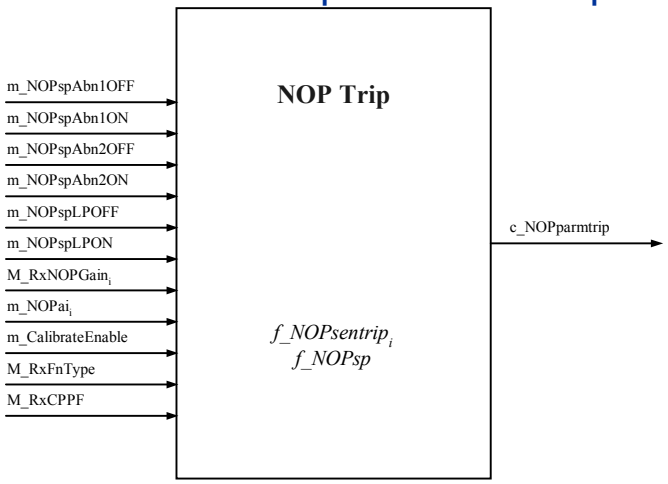
where $t_{manualreset}$ = tm([Manual reset]), and NT is some representation (in ms) to indicate that there was no channel trip in the interval concerned.

tm(cond) is most recent time cond changed from False to True

Safety-Critical Software

TCDR

- Examine one "path" as example



TCDR

Safety-Critical Software

- The following table evaluates the current value of the Neutron Overpower (NOP) setpoint. It relies heavily on a number of "natural language expressions".

19

TCDR

Safety-Critical Software

- Neutron Overpower Setpoint

Condition	*Result* **f_NOPsp**
NOP Low Power setpoint is requested	k_NOPLPsp
NOP Low Power setpoint is cancelled & NOP Abnormal 2 setpoint is requested	k_NOPAbn2sp
NOP Low Power setpoint is cancelled & NOP Abnormal 2 setpoint is cancelled & NOP Abnormal 1 setpoint is requested	k_NOPAbn1sp
NOP Low Power setpoint is cancelled & NOP Abnormal 2 setpoint is cancelled & NOP Abnormal 1 setpoint is cancelled	k_NOPnormsp

20

TCDR

Safety-Critical Software

- Example: NOP Abnormal 1 setpoint is requested or cancelled

Condition	*Result* **NOP Abnormal 1 setpoint is requested or cancelled**
(m_NOPspAbn1ON = e_NotPressed) & (m_NOPspAbn1OFF = e_NotPressed)	No Change
(m_NOPspAbn1ON = e_NotPressed) & (m_NOPspAbn1OFF = e_Pressed)	cancelled
(m_NOPspAbn1ON = e_Pressed) & (m_NOPspAbn1OFF = e_NotPressed)	requested
(m_NOPspAbn1ON = e_Pressed) & (m_NOPspAbn1OFF = e_Pressed)	requested

21

TCDR

Safety-Critical Software

- Already saw NOP setpoint, now NOP sensor trips

{For each i = 1,..,18}

Condition	*Result* **$f\_NOPsentrip_i$**
f_NOPsp ≤ Calibrated i^{th} NOP signal *{Calibrated NOP $signal_i$ is now in the trip region}*	e_Trip
f_NOPsp - k_NOPhys < Calibrated i^{th} NOP signal < f_NOPsp *{Calibrated NOP $signal_i$ is now in the deadband region}*	$(f\_NOPsentrip_i)_{-1}$
Calibrated i^{th} NOP signal ≤ f_NOPsp - k_NOPhys *{Calibrated NOP $signal_i$ is now in the non-trip region}*	e_NotTrip

22

TCDR

Safety-Critical Software

- And NOP parameter trip

Condition	Result c_NOPparmtrip
Any (i ∈ 1,..,18) ($f\_NOPsentrip_i$ = e_Trip) *{Any NOP sensor is tripped}*	e_Trip
All (i = 1,..,18) ($f\_NOPsentrip_i$ = e_NotTrip) *{All NOP sensors are not tripped}*	e_NotTrip

23

TCDR

Safety-Critical Software

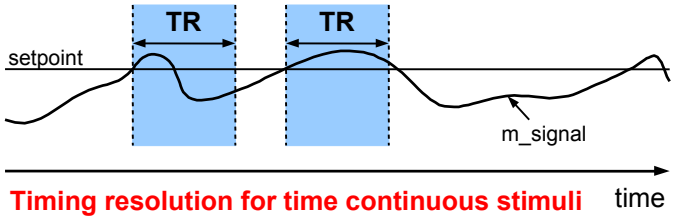
- Tolerances & Timing Requirements
  - Black-box describes behaviour of an idealised system - impossible to meet.
  - Need tolerances in general.
  - Need timing tolerances in particular.
    - Timing Resolutions (TRs)
    - Performance Timing Requirements (PTRs)

24

TCDR

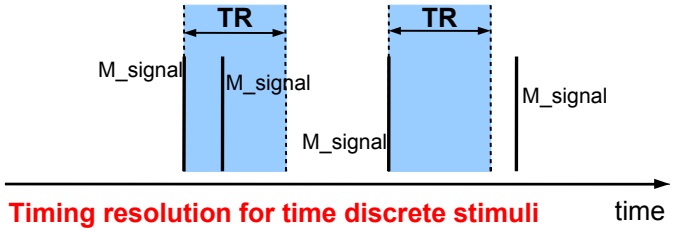
Safety-Critical Software

An event that lasts for TR or longer must be detected



Timing resolution for time continuous stimuli

Two events at least TR apart must both be detected



Timing resolution for time discrete stimuli

25

TCDR

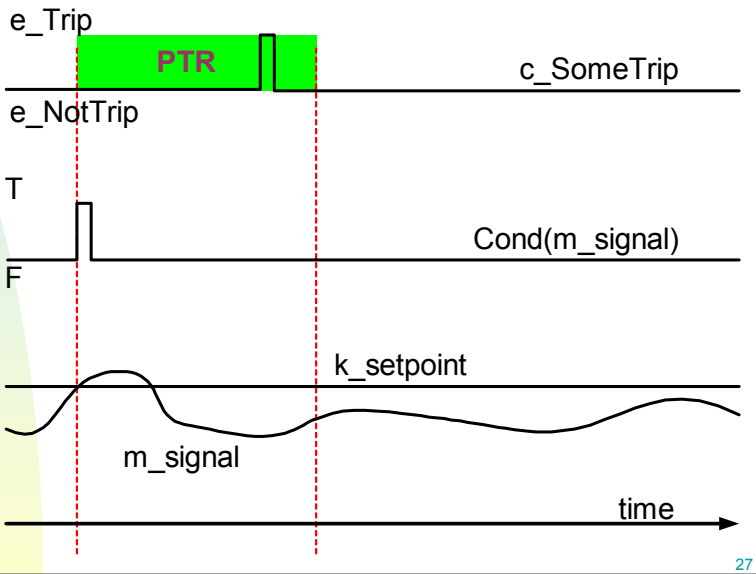
Safety-Critical Software

- PTRs
  - The PTR for a stimulus/response pair specifies an allowable processing delay.
  - Each response must have a PTR specified for it. The PTR applies to the response and the particular stimuli on which the response behaviour depends. The PTR is measured from the time that the stimulus on which the response behaviour depends crosses the system boundary until the time that the associated response crosses the system boundary.

26

Safety-Critical Software

TCDR



27

Safety-Critical Software

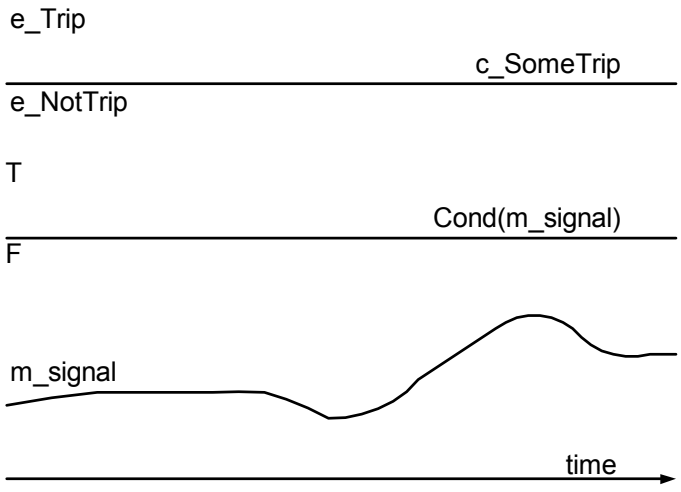
TCDR

- PTRs
  - The PTR for the pair c-m is meaningless if c does not change its value in response to a change in the value of m.
  - The time sequence of responses of a controlled variable, c, cannot be altered by consideration of the PTRs for each c-m_i pair.

28

TCDR

Safety-Critical Software

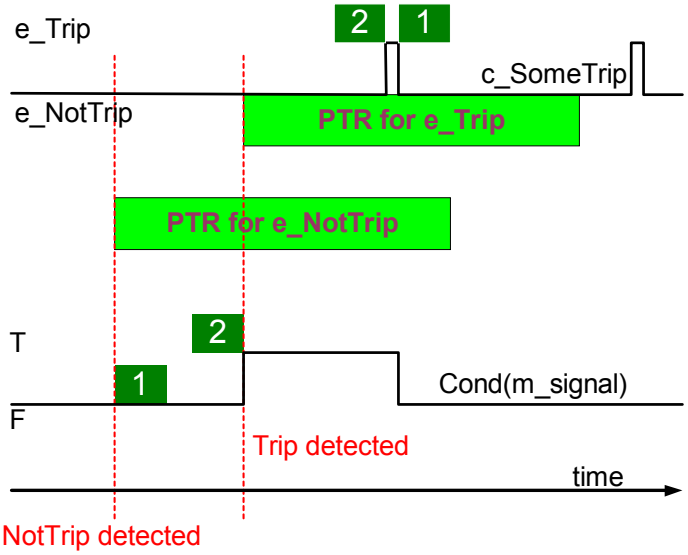


If m does not cause a change in c, PTR has no black-box meaning

29

TCDR

Safety-Critical Software



Cannot allow this!

30

Safety-Critical Software

TCDR

Example for c_NOPparmtrip

Table 2.4.1 - Timing Requirements

Controlled Variable	Governing Variables	PTR	TR	Reference
c_NOPparmtrip	m_NOPai_i, i=1,..,18	160 ms	Default (Already more restrictive than required to meet seal-in)	[38], Table 1
	m_NOPspAbn1OFF m_NOPspAbn1ON m_NOPspAbn2OFF m_NOPspAbn2ON m_NOPspLPOFF m_NOPspLPON	850 ms	350 ms	[14], 1
	m_CalibrateEnable	N/A	1000 ms	No direct effect
	M_RxFnType	2000 ms	Default	-[3]
	$M\_RxNOPGain_i$, i=1,..,18	N/A	2000 ms	No direct effect

31

Safety-Critical Software

TCDR

- Anticipated Changes
  - Information Hiding is a software design paradigm that was introduced by Parnas in a famous paper in the early 1970s.
  - The original version of Information Hiding used anticipated design changes to drive the software decomposition.
  - It turns out that requirement changes are an even greater source of "secrets".

Parnas, D.: On the criteria to be used in decomposing systems into modules. Communications of the ACM December (1972) 1053-1058

32

TCDR

Safety-Critical Software

Table 9.1-1 - Anticipated Changes

Id	Anticipated Change
AC-1	Provisions shall be made in the software coding to give all power dependent setpoints the ability to handle arbitrary setpoint functions (instead of the current step functions). As a minimum requirement, facilities to accommodate a setpoint value for each 1% power change shall be provided up to an upper limit of 110% Full Power (FP).
AC-2	Ranges as specified in the table that defines values of constants, see [27], shall be pre-verified so that the application can use any trip setpoint in the relevant range.
AC-3	New parameter trips may be added.
AC-4	The algorithm and number of detectors used for the estimated power calculation may change from the current specification.
AC-5	Individual deadbands may be revised, HTLF in particular.
AC-6	The processing time for the HTLF analog inputs may be reduced by 100 ms to make provisions to reallocate delay external to the trip computer, see [47], 2.2.2.

33

TCDD

Safety-Critical Software

- **T**rip **C**omputer **D**esign **D**escription
  - Uses TCDR as a basis
  - Adds design information, e.g. pushbutton debouncing
  - Model changes to a Finite State Machine with an arbitrarily small clock-tick
  - SRS contained within TCDD

34

TCDD

Safety-Critical Software

- Finite State Machine (FSM) Model
  - C(t) - set of controlled vars at time t
  - M(t) - set of monitored vars at time t
  - S(t) - set of state vars at time t
  - t_0 - time of initialisation
  - $S(t_0)$ must be known

$C(t_k) = REQ(M(t_k), S(t_k))$

$S(t_{k+1}) = NS(M(t_k), S(t_k)),\ k=0,1,2,3,...$

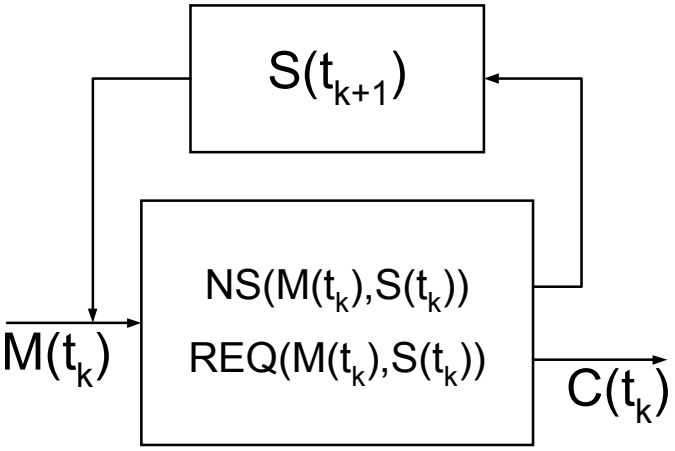
35

TCDD

Safety-Critical Software

- FSM Model

We have very simple states in the TCDD. If f_name is a function, f_name_{-1} is the value of f_name at the previous clock tick. It is a state variable.



36

TCDD

Safety-Critical Software

- TCDD
  - Context diagrams
  - Monitored & Controlled Variables
  - Constants
  - Main function tables
  - Natural language expressions
  - M-I mappings, transfer events
  - C-O mappings, transfer events
  - Tolerances, PTRs and TRs
  - Anticipated changes
  - Changes from previous freezes

37

TCDD

Safety-Critical Software

- We can see how debouncing pushbuttons affects the behaviour specified in the TCDR.
- In particular, “NOP Abnormal 1 setpoint is requested or cancelled” is respecified as follows.

38

TCDD

Safety-Critical Software

- Debounce a pushbutton
  - Can specify more than 1 output

Condition	*Result* f_NOPspAbn1OFF	f_StuckNOPspAbn1OFF
m_NOPspAbn1OFF = e_NotPressed	e_pbNotDebounced	False
[m_NOPspAbn1OFF = e_Pressed] & NOT [(m_NOPspAbn1OFF = e_Pressed) Held for k_Debounce]	e_pbNotDebounced	False
[(m_NOPspAbn1OFF = e_Pressed) Held for k_Debounce] & NOT [(m_NOPspAbn1OFF = e_Pressed) Held for k_pbStuck]	e_pbDebounced	False
(m_NOPspAbn1OFF = e_Pressed) Held for k_pbStuck	e_pbStuck	True

39

TCDD

Safety-Critical Software

- So, NOP Abnormal 1 setpoint is requested or cancelled is now defined in terms of f_NOPspAbn1ON/OFF

Condition	*Result* NOP Abnormal 1 setpoint is requested or cancelled
f_NOPspAbn1ON = e_pbStuck OR f_NOPspAbn1OFF = e_pbStuck	requested
f_NOPspAbn1ON = e_pbNotDebounced & f_NOPspAbn1OFF = e_pbNotDebounced	No Change
f_NOPspAbn1ON = e_pbNotDebounced & f_NOPspAbn1OFF = e_pbDebounced	cancelled
f_NOPspAbn1ON = e_pbDebounced & f_NOPspAbn1OFF = e_pbNotDebounced	requested
f_NOPspAbn1ON = e_pbDebounced & f_NOPspAbn1OFF = e_pbDebounced	requested

40

Safety-Critical Software

TCDD

TCDR Version

Condition	Result: NOP Abnormal 1 setpoint is requested or cancelled
(m_NOPspAbn1ON = e_NotPressed) & (m_NOPspAbn1OFF = e_NotPressed)	No Change
(m_NOPspAbn1ON = e_NotPressed) & (m_NOPspAbn1OFF = e_Pressed)	cancelled
(m_NOPspAbn1ON = e_Pressed) & (m_NOPspAbn1OFF = e_NotPressed)	requested
(m_NOPspAbn1ON = e_Pressed) & (m_NOPspAbn1OFF = e_Pressed)	requested

TCDD Version

Condition	Result: NOP Abnormal 1 setpoint is requested or cancelled
f_NOPspAbn1ON = e_pbStuck OR f_NOPspAbn1OFF = e_pbStuck	requested
f_NOPspAbn1ON = e_pbNotDebounced & f_NOPspAbn1OFF = e_pbNotDebounced	No Change
f_NOPspAbn1ON = e_pbNotDebounced & f_NOPspAbn1OFF = e_pbDebounced	cancelled
f_NOPspAbn1ON = e_pbDebounced & f_NOPspAbn1OFF = e_pbNotDebounced	requested
f_NOPspAbn1ON = e_pbDebounced & f_NOPspAbn1OFF = e_pbDebounced	requested

41

Safety-Critical Software

TCDD

Modified version of TCDR specification

Condition		Result: f_ChanTripTime	FirstTrip
[Manual reset]		NT	True
NOT [Manual reset]	[c_ChanTrip = e_Trip] & [$FirstTrip_{-1}$] *{There is currently a channel trip and this is the first channel trip since a manual reset}*	t_{now}	False
	NOT [c_ChanTrip = e_Trip] OR NOT [$FirstTrip_{-1}$] *{There is not currently a channel trip or this is not the first channel trip since a manual reset}*	$f\_ChanTripTime_{-1}$	$FirstTrip_{-1}$

where NT is chosen to be zero.

Notes:

1) The storage capacity required for this item is one word (16 bits). The value contained in this item may "wraparound" if required, i.e. if the value exceeds the storage capacity by a value n, then the new value becomes n-1.

2) Since NT is chosen to be zero, a channel trip at the time of initialization, or at wraparound, cannot be differentiated from the case where there is no channel trip.

42

Safety-Critical Software

TCDD

TCDR version

Condition		Result: f_ChanTripTime
[Manual reset]		NT
NOT [Manual reset]	[There is a time t_k, $t_{manualreset} < t_k \le t_{now}$ where c_ChanTrip(t_k) = e_Trip] & NOT [There is a time t_i, $t_{manualreset} < t_i < t_k$ where c_ChanTrip(t_i) = e_Trip] *{t_k is the earliest time at which the channel was tripped after the most recent manual reset}*	t_k
	[NOT (c_ChanTrip(t) = e_Trip) for all t, $t_{manualreset} < t \le t_{now}$] *{No channel trip has occurred since the most recent manual reset}*	NT

where $t_{manualreset}$ = tm([Manual reset]), and NT is some representation (in ms) to indicate that there was no channel trip in the interval concerned.

TCDD version

Condition		Result: f_ChanTripTime	Result: FirstTrip
[Manual reset]		NT	True
NOT [Manual reset]	[c_ChanTrip = e_Trip] & [$FirstTrip_{-1}$] *{There is currently a channel trip and this is the first channel trip since a manual reset}*	t_{now}	False
	NOT [c_ChanTrip = e_Trip] OR NOT [$FirstTrip_{-1}$] *{There is not currently a channel trip or this is not the first channel trip since a manual reset}*	$f\_ChanTripTime_{-1}$	$FirstTrip_{-1}$

where NT is chosen to be zero.

43

TCDD

Safety-Critical Software

- We need to examine functional timing requirements in more detail.
- As an example, consider a sensor trip that depends on a sensor (signal) being in a trip state for a sustained period of time.

44

Safety-Critical Software

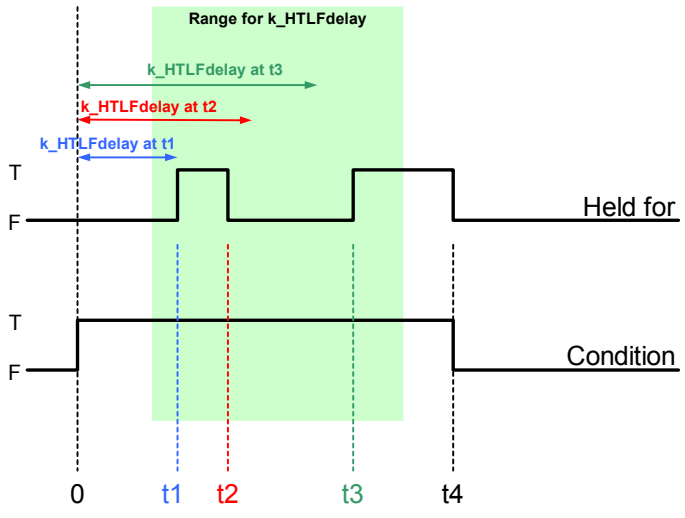
$f\_HTLFsentrip_i$, i=1,..,4
{For each i = 1,..,4}

Condition		Result $f\_HTLFsentrip_i$
$t_{now} \geq$ k_HTLFdelay	(f_HTLFimmsentrip$_i$ = e_Trip) Held for (k_HTLFdelay) *{Immediate sensor trip active for at least past k_HTLFdelay time period}*	e_Trip
	NOT [(f_HTLFimmsentrip$_i$ = e_Trip) Held for (k_HTLFdelay)] *{Immediate sensor trip not active for at least one instant in past k_HTLFdelay time period}*	e_NotTrip
$t_{now} <$ k_HTLFdelay	(f_HTLFimmsentrip$_i$ = e_Trip) Held for (t_{now}) *{Immediate sensor trip active since initialization}*	e_Trip
	NOT [(f_HTLFimmsentrip$_i$ = e_Trip) Held for (t_{now})] *{Immediate sensor trip not active for at least one instant since initialization}*	e_NotTrip

- Since k_HTLFdelay has a tolerance, how do we cope with the fact that, from clock-tick to clock-tick, k_HTLFdelay may have different values at different times?

45

Safety-Critical Software

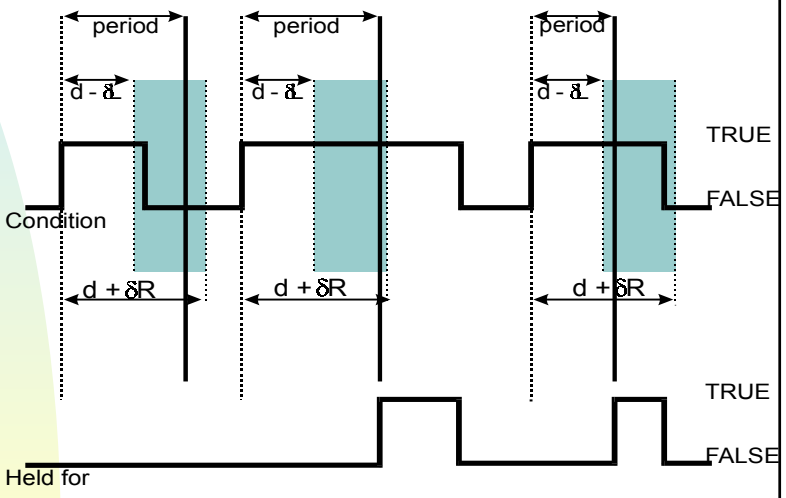


Allowing tolerances on timing constants can cause significant problems

46

Safety-Critical Software

TCDD



47

Safety-Critical Software

(Condition) Held for (d) This function (Held for) checks to see whether "Condition" is true and has been held true continuously for the immediate past "d" time period.

Let period be given by the duration, d, and tolerances $-\delta L$, $+\delta R$, $0 \le \delta L < d$, $0 \le \delta R$. Then period must be restricted as follows:

Condition	*Result* **period**	**Event_start_time**
(Condition = True) & (Condition$_{-1}$ = False)	Any value in [d-δL, d+δR]	t_{now}
(Condition = False) OR (Condition$_{-1}$ = True)	No Change	No Change

where at time of initialization, period is any value in [d-δL, d+δR], and Event_start_time$_{-1}$ = 0.

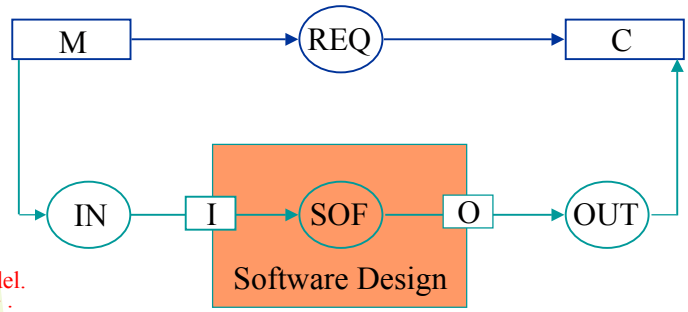
Condition		*Result* **(Condition) Held for (d)**
Condition = True	t_{now} – Event_start_time ≥ period	True
	t_{now} – Event_start_time < period	False
Condition = False		False

48

TCDD

Safety-Critical Software

- TCDD



Parnas' 4 variable model.
Parnas, D.L., Madey, J.: Functional documents for computer systems. Science of Computer Programming 25 (1995) 41-61

C = REQ(M*), and related to the software through I = IN(M) and C = OUT(O)

49

TCDD

Safety-Critical Software

- The TCDD specifies "transfer events" (what the software must do to trigger getting a software input, I, or emitting a software output, O).
- The TCDD also specifies the "M-I mappings", and "C-O mappings" (what the hardware does).

50

TCDD

Safety-Critical Software

- The TCDD specifies a modified list of timing requirements that takes into account the design aspects added in the TCDD.
- The TCDD also lists a (potentially modified) list of anticipated changes.

51

TCDD

Safety-Critical Software

Example for c_NOPparmtrip

Table 2.2.2 - Timing Requirements

Controlled Variable	Governing Variables	PTR	TR	Reference
c_NOPparmtrip	m_NOPai$_i$, i=1,..,18	160 ms	Default (Already more restrictive than required to meet seal-in)	TCDR
	m_NOPspAbn1OFF m_NOPspAbn1ON m_NOPspAbn2OFF m_NOPspAbn2ON m_NOPspLPOFF m_NOPspLPON	850 ms (Held for k_Debounce) / 500 ms	350 ms	TCDR and [13], #1 and [23]
	m_CalibrateEnable	N/A	1000 ms	TCDR
	M_RxFnType	2000 ms	Default	TCDR
	M_RxNOPGain$_i$, i=1,..,18	N/A	2000 ms	TCDR

52

Safety-Critical Software

TCDD

Example function definition in TCDD

2.3.1 Steam Generator Low Level Sensor Trip

Determines whether there is a Steam Generator low level sensor trip, which is used to determine whether there is an associated parameter trip.

2.3.1.1 Inputs/Natural Language Expressions

Input	NL Expression	Reference
f_SGLLsp	-	2.3.3.4
$m\_SGLevel_i$, i=1,..,4	-	Table 2.6.1-1

2.3.1.2 Initial Value

Name	Initial Value	Reference
$(f\_SGLLsentrip_i)_{-1}$, i=1,..,4	e_Trip	TCDR

2.3.1.3 Output Units/Type

Output	Type
$f\_SGLLsentrip_i$, i=1,..,4	y_trip

2.3.1.4 $f\_SGLLsentrip_i$, i=1,..,4

{For each i = 1,..,4}

Condition	*Result* **$f\_SGLLsentrip_i$**
$f\_SGLLsp \geq m\_SGLevel_i$ *{SG level$_i$ signal is now in the trip region}*	e_Trip
$f\_SGLLsp + k\_SGLLhys > m\_SGLevel_i > f\_SGLLsp$ *{SG level$_i$ signal is now in the deadband region}*	$(f\_SGLLsentrip_i)_{-1}$
$m\_SGLevel_i \geq f\_SGLLsp + k\_SGLLhys$ *{SG level$_i$ signal is now in the non-trip region}*	e_NotTrip

53

Safety-Critical Software

SDD

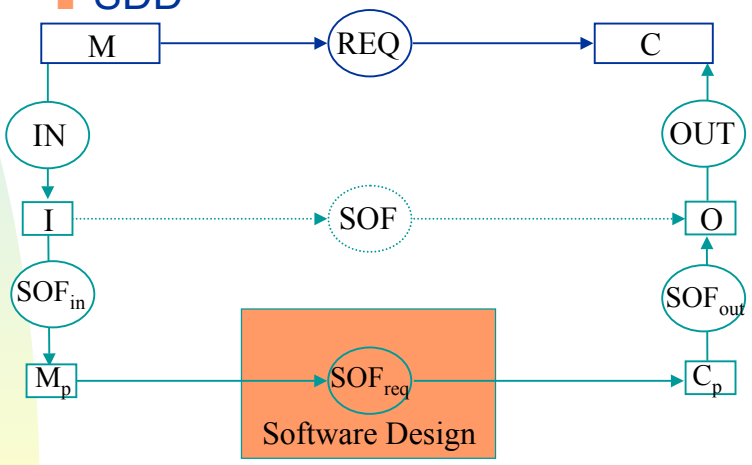
- **S**oftware **D**esign **D**escription
  - Augmented anticipated changes
  - Module guide
  - Alternate views
  - Module designs
  - Derived timing requirements
  - Design notes
  - Supplementary function tables / mappings to TCDD

54

Safety-Critical Software

SDD

- SDD



55

Safety-Critical Software

SDD

- Some crucial definitions
  - Software module
  - Access program
  - Processing unit

56

SDD

Safety-Critical Software

- Software module
  - A module is a conceptual design unit in the decomposition of software functionality. Modules can be developed in a hierarchical structure where one module can "contain" other modules. This leads to a tree structure of modules.
  - The top-level modules divide the system into a few broadly defined areas.

57

SDD

Safety-Critical Software

- Software module - continued
  - Each top-level module should have a general responsibility (role and purpose) related to encapsulating (hiding) a design decision (a secret).
  - The functionality enclosed within such a module should contribute to that overall responsibility.
  - Each of the top-level modules can be further decomposed into smaller modules.

58

SDD

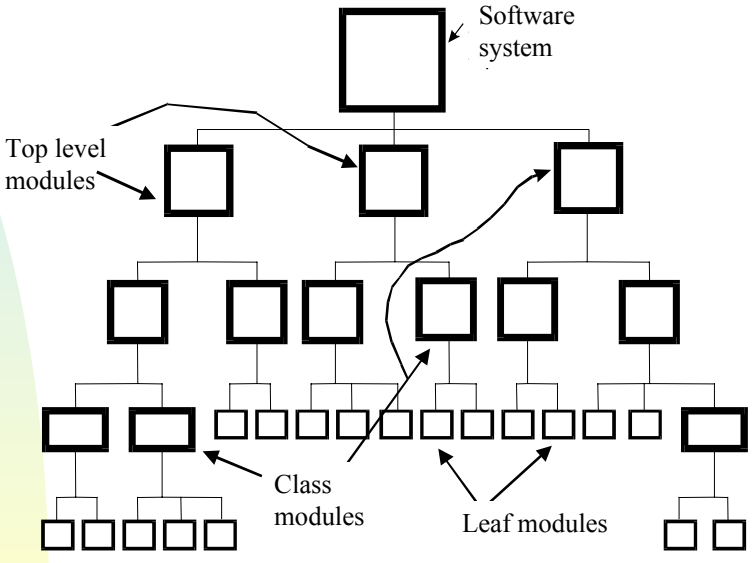
Safety-Critical Software

- Software module - continued
  - Each of these smaller modules has its own responsibility and secret, but the responsibilities are more focused.
  - Modules that require no further refinement are termed “leaf modules” (in keeping with the tree structure). The leaf modules represent the final decomposition of the software system.

59

SDD

Safety-Critical Software



60

SDD

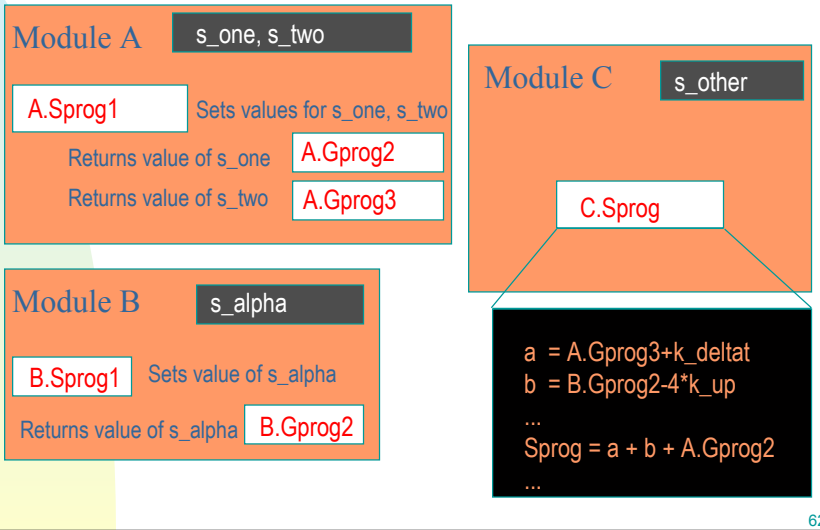
Safety-Critical Software

- Access programs
  - Leaf modules communicate with each other through the use of externally visible access programs. The access programs belonging to a particular leaf module (A) allow other leaf modules to deliver data that the module (A) needs, and to examine the values of data items contained within the module (A). This is the only approved method for one leaf module to access the contents of another leaf module.

61

SDD

Safety-Critical Software



62

SDD

Safety-Critical Software

- Processing unit
  - A processing unit is a group of access programs that is considered to be a single executable unit. The required behaviour of the application is achieved by executing all the processing units defined for the application according to the control structure defined in the scheduler

63

SDD

Safety-Critical Software

- We use design attributes to drive the software design process
  - General design attributes
  - System structure attributes
  - Decomposition attributes
  - Module interface design attributes
  - Module detailed design attributes

64

SDD

Safety-Critical Software

- Decomposition attributes
  - Group 1 - (during and after decomposition)
    - Coverage, Information hiding, Responsibility, Locatability, Hardware isolation
  - Group 2 - (after interfaces)
    - Module cohesion, Focus, Maintainability, Localization of data
  - Group 3 - (after detailed design)
    - Independence, Uniqueness of data

65

SDD

Safety-Critical Software

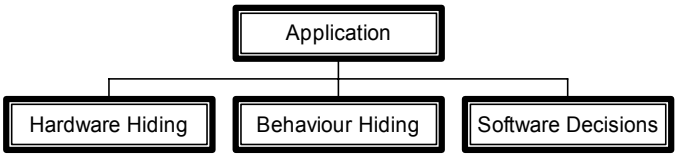
- Information hiding attribute
  - For any change that has been identified as likely, the functional behaviour directly affected by that change should be isolated in a single module.

66

SDD

Safety-Critical Software

- Top-level decomposition



67

SDD

Safety-Critical Software

- Repeated decomposition
  - Each top-level module can be examined to see how it can be further decomposed. The decomposed modules represent classes of modules and should share some distinguishing characteristics in terms of their responsibilities and the secrets they hide from the rest of the system. Typical classes at this level contain sets of: Function drivers; and Device interface modules.

68

SDD

Safety-Critical Software

- Function drivers
  - Modules whose sole purpose is to produce system outputs.
- Device interface modules
  - Modules that provide virtual interfaces to the hardware, completely hiding the characteristics of a peripheral device from the software application.

69

SDD

Safety-Critical Software

- Each class module can be further decomposed until the leaf modules represent the final decomposition of the system.
- To implement this process requires heuristics to aid in further decomposing a module, and criteria for stopping any further decomposition.

70

SDD

Safety-Critical Software

- Decomposing modules
  - Isolate components likely to change
  - Protect data from unintentional/ unwarranted modification by encapsulating it
  - Look for groups of history dependent entities for possible encapsulation.
  - Encapsulate interrupt and timer processing.
  - Encapsulate algorithms specified in the TCDD.
  - Encapsulate scheduling policies.

71

SDD

Safety-Critical Software

- Stopping further decomposition of a module
  - If the module were to be further decomposed, the secret encapsulated within a newly created module would not be worth hiding.
  - Further decomposition would require excessive communication between the new modules.
  - The module is already simple enough to be implemented easily in code.

72

Safety-Critical Software

SDD

Module Guide extract

Table 5-4 - Module Reference Table (Behaviour Hiding)

ID	Name	Module Secrets	Module Responsibility	TCDD Ref
1	BehaviourHiding	All aspects of behaviour of the system that 1) are externally visible, 2) can be implemented independently of changes in hardware (which do not affect the capability of the hardware).	Implements the rules that produce the required external behaviour of the SDS1 Trip Computer.	-
1.1	**ChannelTrip**	The rules for determining the channel trip system outputs, the channel trip time since the last manual reset, the manual reset request status, and the validity of the channel trip sealed-in status word. (C21)	Determines all the channel trip related statuses. Provides current channel trip status to drive the channel trip output, channel trip time since the last manual reset, status of the manual reset request and validity of the channel trip sealed-in status word.	'Any parameter tripped', 'Channel trip sealed-in', 'Consistency check fatal error', 'Manual reset request', c_ChanTrip, f_ChanTripTime, sw_SealinWord, Initial Value, Initialization Requirements.
1.2	ParamTrip (P)	The rules for determining parameter trip system outputs. (C3, C7)	A parameterized module which implements parameter trip detection logic for trip parameters. Provides the current parameter trip status to drive the parameter trip output.	N/A
1.2.1	**NPParTrip**	The rules for determining the Neutron Overpower parameter trip system output.	Instantiation of ParamTrip module for Neutron Overpower trip.	c_NOPparmtrip, Initial Value, Initialization Requirements.
…	**…**	…	…	…
1.3	SensorTrip (P)	The rules for determining sensor trips for each trip parameter. (C5, C6)	A parameterized module which implements the sensor trip determination rules. Provides the current sensor trip status. Provides facilities to track/reset the status of sensor trip occurrence indication.	N/A
1.3.1	**NPSnrTrip**	The rules for determining Neutron Overpower sensor trips.	Instantiation of SensorTrip module for Neutron Overpower trip.	$f\_NOPsentrip_i$, i=1,..,18, Initial Value, Initialization Requirements, $f\_NOPsentripst_i$, i=1,..,18*.
…	**…**	…	…	…

73

Safety-Critical Software

SDD

Module Dependencies

- Need to know what data is required by which module and which module owns the data
- How do we know this?
  - We have complete functional descriptions of behaviour in the TCDD
  - A module's responsibility is represented by functional behaviour as described in the TCDD

74

SDD

Safety-Critical Software

- Create and refine access programs
  - Default set of access program always exists
    - Perform processing (update state data, produce a system output)
    - Initialization program
    - Programs that provide information about the module's state data to other modules
  - Can refine these access programs in a (sort of) systematic way

75

SDD

Safety-Critical Software

- Criteria while refining access programs
  - Access programs must not reveal secret of the module
  - Each program's interface is as simple as possible
  - Each program services its users in a way convenient to the users
  - Each parameter required by every user
  - Make choices based on simplifying timing
  - Parameters of a single access program should all move data in a single direction (does not apply to control variables)

76

SDD

Safety-Critical Software

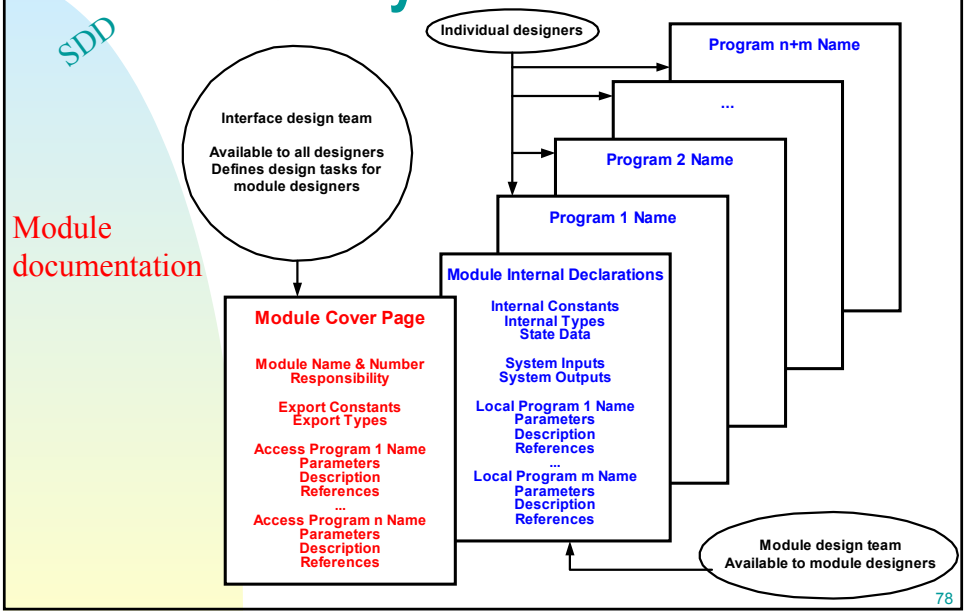
- Once all the modules and module interfaces are decided we can continue with the detailed design of each module.

77

SDD

Safety-Critical Software

Module documentation



78

SDD

Safety-Critical Software

n.m MODULE Watchdog (1.10)

Determines the watchdog system output.

	Name	Value	Type
Constants:	(None)		

	Name	Definition
Types:	(None)	

Access Programs:

EWDOG
Updates the state of the watchdog timer DO.
References: c_Watchdog, 'Watchdog test active'*.

IWDOG
Initializes all the Watchdog module internal states and sets the initial watchdog output.
References: Initial Value, Initialization Requirements.

SWDOG
NCPARM: t_boolean - in
Signals to Watchdog module that a valid watchdog test request is received if NCPARM = $TRUE. Note that NCPARM is a "Conditional Output Call Argument"; calling the program with NCPARM = $FALSE has no effects on the module.
References: 'Watchdog test active'*.

Cover page for module Watchdog

79

SDD

Safety-Critical Software

n.m.1 MODULE Watchdog Internal Declaration

	Name	Value/Origin	Type
Constants:	KWDDLY	1000	t_integer

	Name	Definition/Origin
Types:	t_boolean	Global
	t_integer	Global
	t_MsecTimerID	Timer
	t_PBId	DigitalInput
	t_PBStat	DigitalInput
	t_PosInt	Global
	t_TimerOp	Timer
	t_WDogStat	DigitalOutput

	Name	Type
State Data:	WDGST	t_boolean
	WDGTST	t_boolean

Module-wide internal declarations

80

Safety-Critical Software

SDD

Detailed module design for an example access program

n.m.1.1 ACCESS PROGRAM EWDOG

	Name	Ext_value	Type	Origin
Inputs:	l_CalEn	GPBKS($PBCAL)	t_PBStat	DigitalInput
	l_TREQD	GCMSEC($CWDG)	t_PosInt	Timer

	Name	Ext_value	Type	Origin
Updates:	WDGST	-	t_boolean	State
	WDGTST	-	t_boolean	State

	Name	Ext_value	Type	Origin
Outputs:	l_WdgClk	SCMSEC($CWDG, l_WdgClk)	t_TimerOp	Timer
	l_WdgDO	SDOWDG(l_WdgDO)	t_WDogStat	DigitalOutput

Range check assertion 1: (l_CalEn = $DBNC) OR (l_CalEn = $NDBNC)

Modes:

l_InTest	0 < l_TREQD < KWDDLY
l_NoTest	l_TREQD = 0
l_TstEnd	l_TREQD >= KWDDLY

VCT: EWDOG

	WDGTST = $FALSE		NOT (WDGTST = $FALSE)			
	WDGST = $FALSE	NOT (WDGST = $FALSE)	l_CalEn = $NDBNC	NOT (l_CalEn = $NDBNC)		
				l_NoTest	l_InTest	l_TstEnd
l_WdgClk	$CRSET	$CRSET	$CRSET	$CSTRT	$CNC	$CRSET
l_WdgDO	$WDON	$WDOFF	$WDNC	$WDNC	$WDNC	$WDNC
WDGST	$TRUE	$FALSE	No Change	No Change	No Change	No Change
WDGTST	No Change	No Change	$FALSE	No Change	No Change	$FALSE

81

Safety-Critical Software

SDD

- Since the decomposition of behaviour in the SDD is almost guaranteed to be different from the decomposition in the TCDD, it is likely that just a portion of a TCDD function may be implemented in an access program, or that a composition of TCDD functions may be implemented in an access program.
- This presents a couple of important problems.

82

SDD

Safety-Critical Software

- We use the TCDD functions to specify the black-box behaviour of an access program, and so if the access program does not implement a single, complete TCDD function, this black-box behaviour is difficult to specify.
- It is difficult to verify the SDD behaviour against the TCDD behaviour when the function topologies of the two are different.

83

SDD

Safety-Critical Software

- The way we overcome these difficulties is by use of “supplementary function tables” (SFTs). The idea here is that we imagine a pseudo requirements specification in which the function topology will exactly match that in the SDD. If such a pseudo requirements specification were to exist, then verifying the SDD against the TCDD could be performed in two steps.

84

Safety-Critical Software

SDD

- Verify the SDD against the pseudo requirements specification.
- Verify the pseudo requirement specification against the TCDD.

85

Safety-Critical Software

SDD

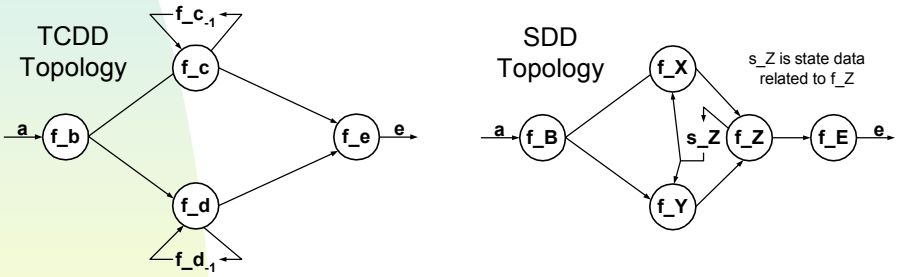
- The SFTs are developed during the forward going process, by the software designers themselves, but are not considered “proved”.
- They are available during the software verification to aid the verification team in the mathematical verification of the software.

86

Safety-Critical Software

SDD

- The following data flow diagrams illustrate the concept of STFs.

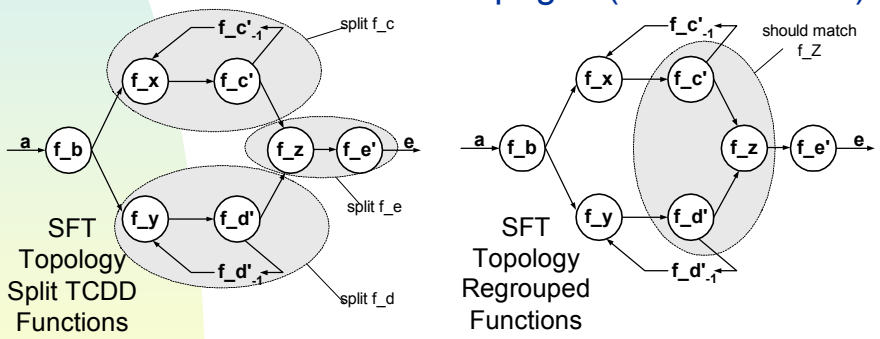


87

Safety-Critical Software

SDD

- We can use the SFTs, f_x, f_c', f_y, f_d', f_z and f_e' as references on module cover pages (and in the MG).



88

SDD

Safety-Critical Software

- Safety-net (a couple of examples)
  - Safe state (if there is one) is always placed in the right-most column of a function table. This convention can be used to produce code that puts the software into a safe state even in the face of hardware malfunctions.
  - Sequence checks - uses a baton passing mechanism to check that programs are invoked in the correct sequence, and no program has been "skipped".

89

SDD

Safety-Critical Software

- Scheduler
  - Optimal: a simple infinite loop that just invokes each processing unit in the required sequence.
  - Reality: if the hardware cannot support the optimal strategy, we have to try alternative strategies - e.g.
    - Loop frames - LF1, LF2, LF3, LF4
    - Each LF invokes programs in a way that will satisfy PTRs, and takes advantage of more relaxed PTRs to schedule those programs less frequently.

90

SDD

Safety-Critical Software

- Processing Unit (PU) Example
  - PU16 {NOP trip fns}

ECALNP()	Calibrated NOP sensors
ESPNP()	NOP setpoint
ESTNP()	NOP sensor trips
EPTNP()	NOP parm trip

91

SDD

Safety-Critical Software

- Example of use of loop frames
  - Straight line version:

136 ms

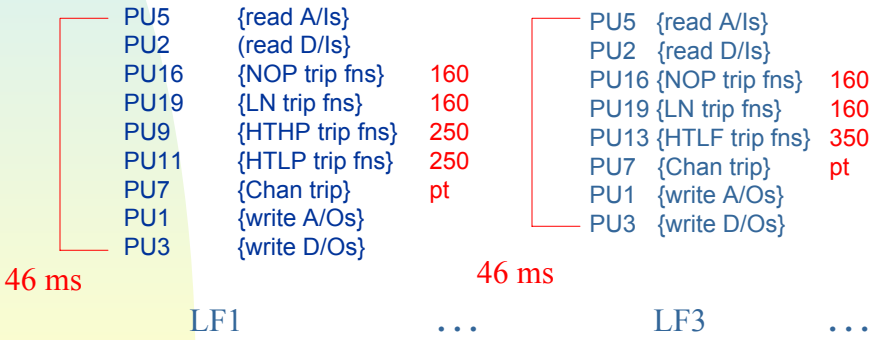
PU5	{read A/Is}	
PU2	(read D/Is}	
PU16	{NOP trip fns}	160
PU13	{HTLF trip fns}	350
. . .	. . .	
PU19	{LN trip fns}	160
PU9	{HTHP trip fns}	250
PU11	{HTLP trip fns}	250
PU7	{Chan trip}	pt
PU1	{write A/Os}	
PU3	{write D/Os}	

92

Safety-Critical Software

SDD

- Example of use of loop frames
  - Loop frame version:



93

Safety-Critical Software

SDV

- SDD Verification
  - Verify SDD against pseudo-TCDD - have same data flow topology.
  - Verify pseudo-TCDD against TCDD - only need to verify those blocks that are different.
  - Concentrate for now on verifying SDD against pseudo-TCDD.

94

Safety-Critical Software

SDV

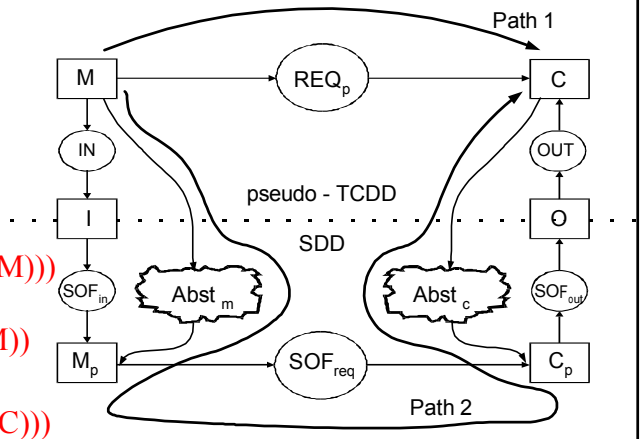
- Proof obligation

Result via path 1 must equal result via path 2.

$REQ_p(M) = Abst_c^{-1}(SOF_{req}(Abst_m(M)))$

$Abst_m(M) = SOF_{in}(IN(M))$

$C = OUT(SOF_{out}(Abst_c(C)))$



95

Safety-Critical Software

SDV

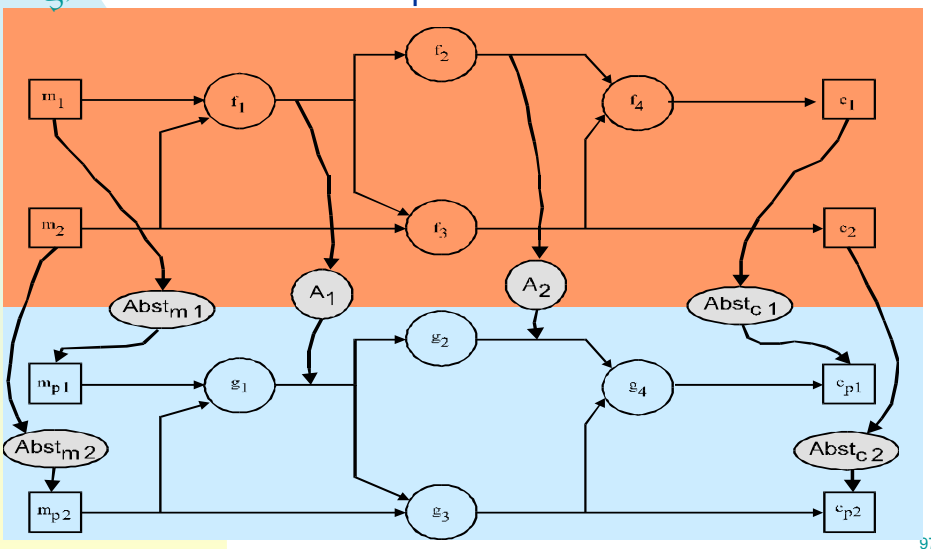
- Proof obligation for SDD vs pseudo-TCDD
  - $REQ_p(M) = Abst_c^{-1}(SOF_{req}(Abst_m(M)))$
  - Can prove that we can do the proof piece-wise in blocks. Each block is identified as having equivalent inputs and outputs in the SDD and pseudo-TCDD. (Can make use of SFTs)

96

Safety-Critical Software

SDV

- Example: TCDD vs SDD



97

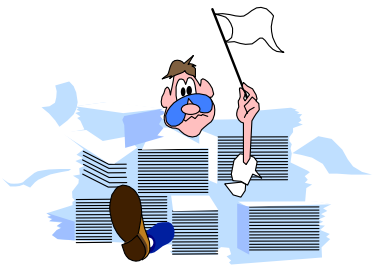
Safety-Critical Software

SDV

- Proof obligation for example

$Abst_{c\,1}^{-1}(g_4(g_2(g_1(Abst_{m\,1}(m_1), Abst_{m\,2}(m_2))), g_3(g_1(Abst_{m\,1}(m_1), Abst_{m\,2}(m_2)), Abst_{m\,2}(m_2)))) = f_4(f_2(f_1(m_1, m_2)), f_3(f_1(m_1, m_2), m_2))$ ... (1)

$Abst_{c\,2}^{-1}(g_3(g_1(Abst_{m\,1}(m_1), Abst_{m\,2}(m_2)), Abst_{m\,2}(m_2))) = f_3(f_1(m_1, m_2), m_2)$ ... (2)



98

Safety-Critical Software

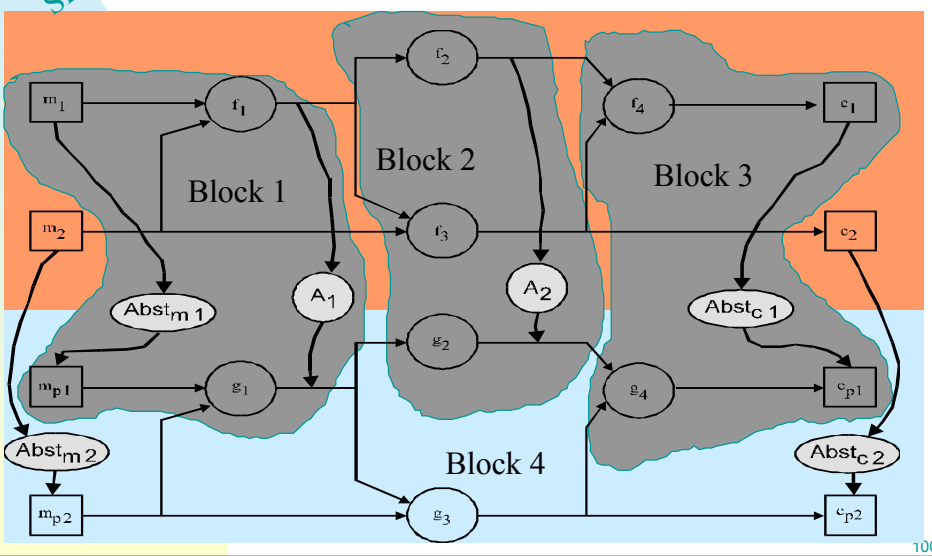
SDV

- Piece-wise verification
  - Possible block boundaries exist on each set of matching inputs and outputs (matching in pseudo-TCDD and SDD)

99

Safety-Critical Software

SDV



100

Safety-Critical Software

SDV

Block 1

- $A_1(f_1(m_1, m_2) = g_1(Abst_{m\,1}(m_1), Abst_{m\,2}(m_2))$ ... (3)

Block 2

- $A_2(f_2(f_1^*)) = g_2(A_1(f_1^*)$ ... (4)

Block 3

- $Abst_{c\,1}(f_4(f_2^*, f_3^*)) = g_4(A_2(f_2^*), Abst_{c\,2}(f_3^*))$ ... (5)

Block 4

- $Abst_{c\,2}(f_3(f_1^*, m_2)) = g_3(A_1(f_1^*), Abst_{m\,2}(m_2))$ ... (6)

- If we now assume that (3) through (6) are verified to be true, then we can proceed by:
  Substituting for $A_2(f_2(f_1^*))$ from (4) into (5) - gives (1), and substituting for $A_1(f_1(m_1, m_2)$ from (3) into (6) - gives (2)
- So, example demonstrates piece-wise success - if topology is the same.

101

Safety-Critical Software

SDV

- Typically, in a specific block, we compose the SDD tables and manipulate them until we can show that they are equal to the composed pseudo-TCDD tables.
- There has also been considerable working done in automating these proofs using theorem provers like PVS.

Lawford, M., McDougall, J., Froebel, P., Moum, G.: Practical application of functional and relational methods for the specification and verification of safety critical software. In Rus, T., ed.: Proc. of AMAST 2000, Iowa City, Iowa, USA, May 2000. Volume 1816 of LNCS., Springer (2000) 73-88.

102

SDV

Safety-Critical Software

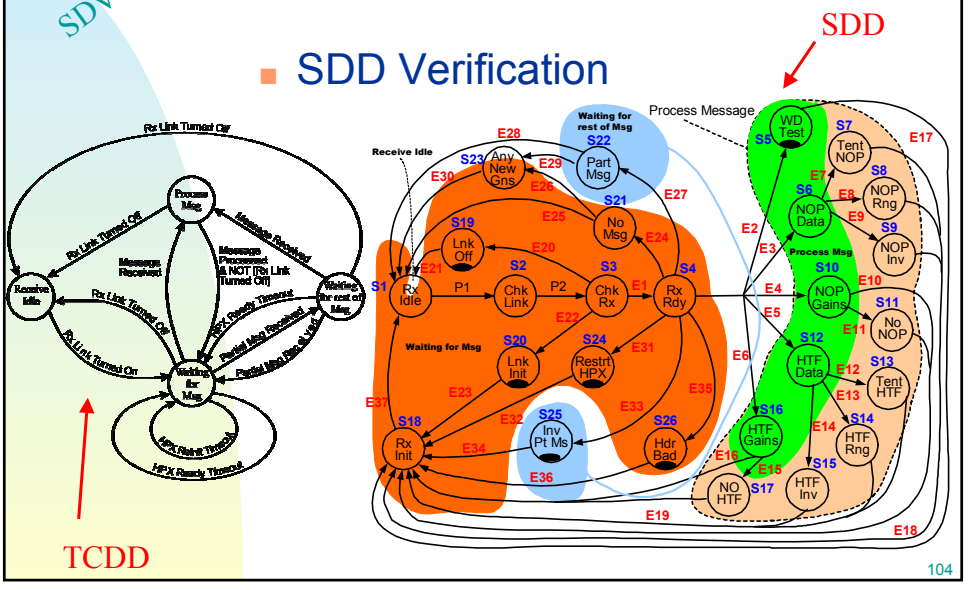
- Design heuristic helps verification: Get-Process-Set.
- There are often cases where timing considerations considerably complicate the verification process.
- There are also some cases where software design decisions complicate the verification.

103

SDV

Safety-Critical Software

- SDD Verification



104

SDV

Safety-Critical Software

- Currently, without more advanced automated verification tools, my estimate is that SDD verification accounts for about 30% of the total effort.

105

Code

Safety-Critical Software

- Coding
  - Coding performed from detailed design in the SDD.
  - Traceable to the SDD.
  - Safe even in the face of hardware malfunctions.
  - Coding from function tables produces well structured, easily verified, code.

106

Code

Safety-Critical Software

- FORTRAN 66 Code
  - Use labelled common to construct modules.
    - COMMON /NAMEX/ list of module exported contants
    - COMMON /NAMEV/ list of module state variables
    - COMMON /NAMEK/ list of module internal constants
    - Any program can include any number of ___X common statements, but only three common statements with the NAME of the module
  - Challenge to describe safe and readable control structures.
  - Comments refer to the SDD.

107

Safety-Critical Software

Code

Example extract for EWDOG

Part 1

```
C     < SDD Input: l_CalEn = DigitalInput.GPBKS($PBCAL) >
      LLCALE = GPBKS($PBCAL)
C     < SDD Input: l_TREQD = Timer.GCMSEC($CWDG) >
      LLTREQ = GCMSEC($CWDG)
C
C     Range check on <l_CalEn> and <l_TREQD>.
      IF (((LLCALE .NE. $DBNC) .AND. (LLCALE .NE. $NDBNC)) .OR.
     +    (LLTREQ .LT. 0))
C        ErrorHdler.SFAT($FERNG, KPLN)
     +   CALL SFAT($FERNG, KPLN)
C
C --- < VCT EWDOG > ----------------------------------------------------
      IF (WDGTST .NE. $FALSE) GO TO 20000
         IF (WDGST .NE. $FALSE) GO TO 12000
            WDGST = $TRUE
C           < SDD Output call: Timer.SCMSEC($CWDG, l_WdgClk) >
            CALL SCMSEC($CWDG, $CRSET)
C           < SDD Output call: DigitalOutput.SDOWDG(l_WdgDO) >
            CALL SDOWDG($WDON)
            GO TO 29999
12000    CONTINUE
C        < NOT(WDGST = $FALSE) >
            WDGST = $FALSE
C           < SDD Output call: Timer.SCMSEC($CWDG, l_WdgClk) >
            CALL SCMSEC($CWDG, $CRSET)
C           < SDD Output call: DigitalOutput.SDOWDG(l_WdgDO) >
            CALL SDOWDG($WDOFF)
            GO TO 29999
20000 CONTINUE
C     < NOT(WDGTST = $FALSE) >
         IF (LLCALE .NE. $NDBNC) GO TO 22000
            WDGTST = $FALSE
C           < SDD Output call: Timer.SCMSEC($CWDG, l_WdgClk) >
            CALL SCMSEC($CWDG, $CRSET)
C           < SDD Conditional Output Call:
C                                DigitalOutput.SDOWDG(l_WdgDO) >
C           CALL SDOWDG($WDNC)
            GO TO 29999
```

108

Safety-Critical Software

Code

Example extract for EWDOG

Part 2

```
22000    CONTINUE
C        < NOT(LLCALE = $NDBNC) >
            IF (LLTREQ .NE. 0) GO TO 22200
C           < l_NoTest >
C              < SDD Output call: Timer.SCMSEC($CWDG, l_WdgClk) >
               CALL SCMSEC($CWDG, $CSTRT)
C              < SDD Conditional Output Call:
C                                DigitalOutput.SDOWDG(l_WdgDO) >
C              CALL SDOWDG($WDNC)
               GO TO 29999
22200       CONTINUE
C           Redundant test of 0 < l_TREQD is not included for
C           performance reasons.
            IF (LLTREQ .GE. KWDDLY) GO TO 22300
C           < l_InTest >
C              < SDD Conditional Output Call:
C                                Timer.SCMSEC($CWDG, l_WdgClk) >
C              CALL SCMSEC($CWDG, $CNC)
C              < SDD Conditional Output Call:
C                                DigitalOutput.SDOWDG(l_WdgDO) >
C              CALL SDOWDG($WDNC)
               GO TO 29999
22300       CONTINUE
C           < l_TstEnd >
               WDGTST = $FALSE
C              < SDD Output call: Timer.SCMSEC($CWDG, l_WdgClk) >
               CALL SCMSEC($CWDG, $CRSET)
C              < SDD Conditional Output Call:
C                                DigitalOutput.SDOWDG(l_WdgDO) >
C              CALL SDOWDG($WDNC)
29999 CONTINUE
C --- < END VCT EWDOG > ------------------------------------------------
```

109

Safety-Critical Software

Code

- Safety-net (again)
  - Use full word for logical/boolean values, not bits
    - 16-bit word: False = 0, True <> 0
    - 65,535 ways to be True, 1 to be False
    - Make sure True represents safe state, then even if there is a RAM error (even some other hardware errors), there is a 65000 to 1 chance that the software will set the safe state

110

CVR

Safety-Critical Software

- Code Verification
  - Typically straight forward and relatively easy.
  - If code comment indicates VCT (Vertical Condition Table) then analyse code to produce VCT. Afterwards compare with SDD.
  - If code comment indicates Algorithm (pseudo-code) then analyse code to compare with pseudo-code.

111

Timing

Safety-Critical Software

- Relationship between TR and PTR
  - Usually the TR for a given m-c pair is dictated by physical characteristics of m. However, nuclear safety analyses make few asumptions about the TR.
  - TR < PTR is sometimes all we can assume [t_s is the sample interval, t_p the processing time, then $t_s+t_p \leq$ PTR. Since $t_p > 0$ and TR is an upper bound for t_s, TR < PTR]

112

Safety-Critical Software

Timing

TRs and PTRs - Basics

real world

times at which inputs are sampled

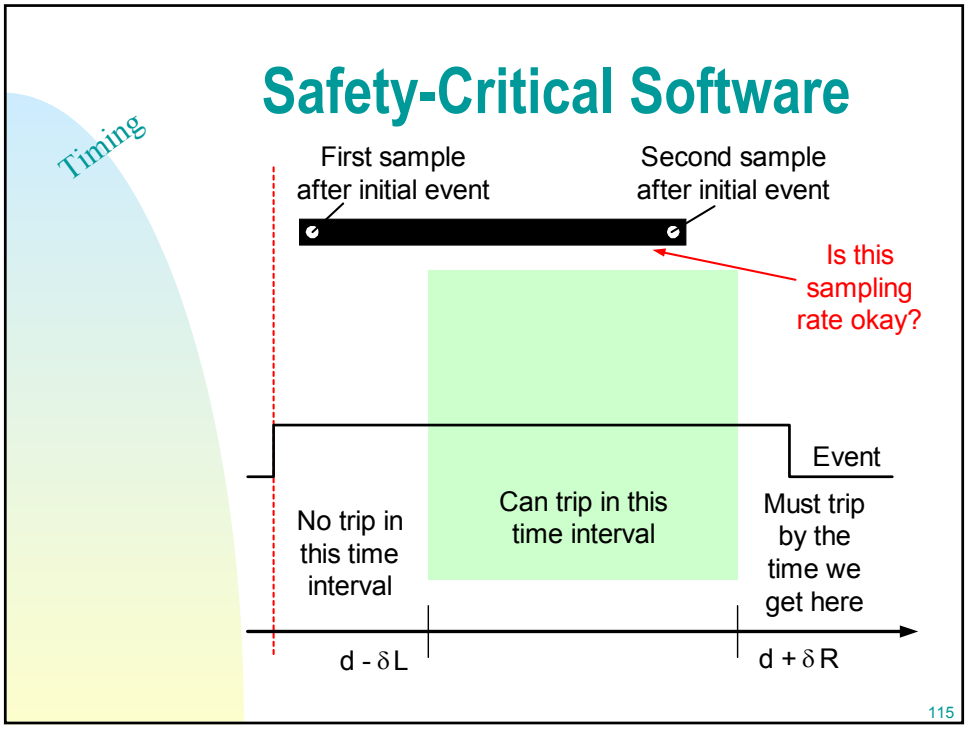
113

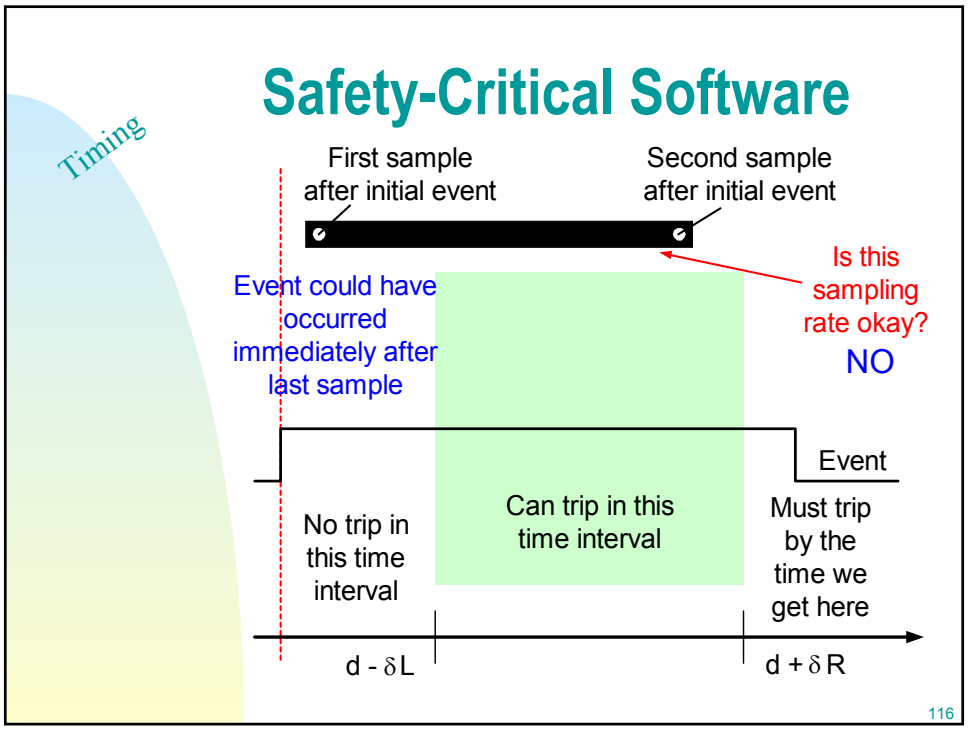
Timing

Safety-Critical Software

- Relationship between TR and functional timing requirements
  - What if we have a sustained event?
    Example: To filter out the effect of a noisy signal we may specify that a sensor trip should be sustained for 400 ± 50 ms before it can cause a parameter trip. This means that the implementation must guarantee that if the sensor trip is sustained for less than 350 ms, the parameter trip must not occur. Similarly, if the sensor trip is sustained for 450 ms, the parameter trip must be recorded. (It can be issued some time later, as long as it is within the PTR for that c-m pair.)

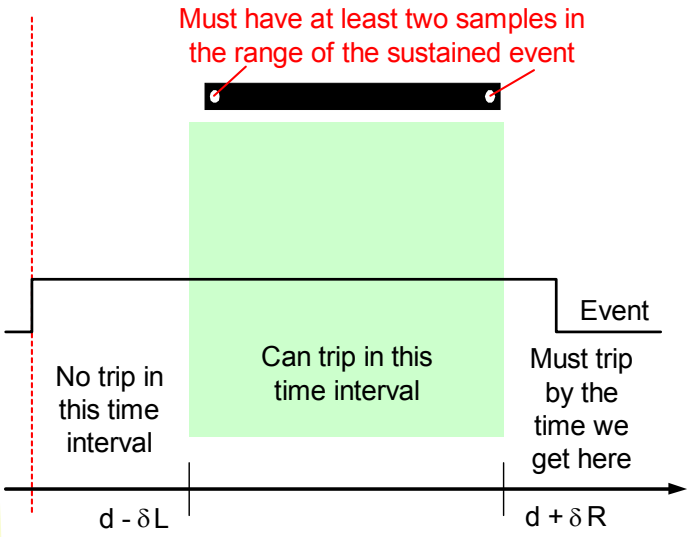
114


Safety-Critical Software
Timing
First sample after initial event
Second sample after initial event
Is this sampling rate okay?
Event
No trip in this time interval
Can trip in this time interval
Must trip by the time we get here
d - δL
d + δR
115


Safety-Critical Software
Timing
First sample after initial event
Second sample after initial event
Event could have occurred immediately after last sample
Is this sampling rate okay?
NO
Event
No trip in this time interval
Can trip in this time interval
Must trip by the time we get here
d - δL
d + δR
116

Timing

Safety-Critical Software



117

Timing

Safety-Critical Software

- Why do we need two samples in the range?
  - If we have only one, and it is not on a boundary, then:
    - Assume t_s is constant
    - Let that sample be the n^{th} sample, including the detection of the event
    - We know that at least $(n-1)t_s$ has elapsed since the actual event
    - But, actually, nt_s may have elapsed
    - However, nt_s will be greater than $d+\delta R$, since only one sample in range

118

Timing

Safety-Critical Software

- TR ≤ (δL+δR)/2 will suffice but is overly restrictive
- If ½(δL+δR) < t_s < (δL+δR), there has to be a sample point in the interval [d-δL, d+δR] at which we make the decision concerning the parameter trip, and this point must be such that the previous sample point also lies within [d-δL, d+δR]. Mathematically what we are looking for is described by:

  Choose t_s such that

  $d-\delta L \le (n-1)t_s$ (left most sample in the interval)

  $nt_s \le d+\delta R$ (right most sample in the interval)

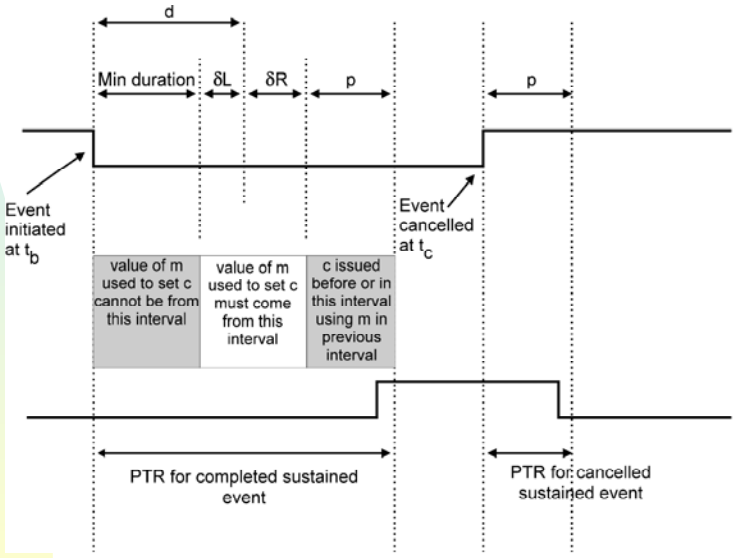
  where n is an integer.

k_HTLFdelay
= 400 ± 50 ms
= [d-δL, d+δR]

119

Timing

Safety-Critical Software



PTRs for sustained events

120

Timing

Safety-Critical Software

- "Seal-in" presents similar but different difficulties.
  - Seal-in is measured from the time a controlled variable changed from e_NotTrip to e_Trip (if the channel D/O is opened for 125-200 ms then the channel trip D/O is not allowed to close until a manual reset is perfomed)
  - Turns out to be slightly easier because it is measured from a known time, so only one sample in the range is required
- Would be very useful to find a more general analysis

121