

Case study in design analysis

Automotive Electrical Systems
Neal Snooke, Aberystwyth

Plan

- Design analysis techniques we have automated
 - Modelling architecture outline
- Performing analysis (user view)
- Building models and simulation
 - How to break it (bad modelling) !
- Modelling and the design lifecycle
- Other ideas

Electrical design analysis

- Component-based ontology
- Qualitative
- Performs simulation **and result abstraction**
- Number of applications:
 - Failure mode and effects analysis
 - Sneak circuit analysis
 - Design verification
 - Diagnosis (Fault trees...)

FMEA

- To answer the question
 - When a component fails what will the effects be ?
- Risk priority analysis (RPN)
 - If we know the reliability of the components what failures carry the most risk?
 - Can potentially serious risks be designed out?
- Manual FMEA is a very tedious task...

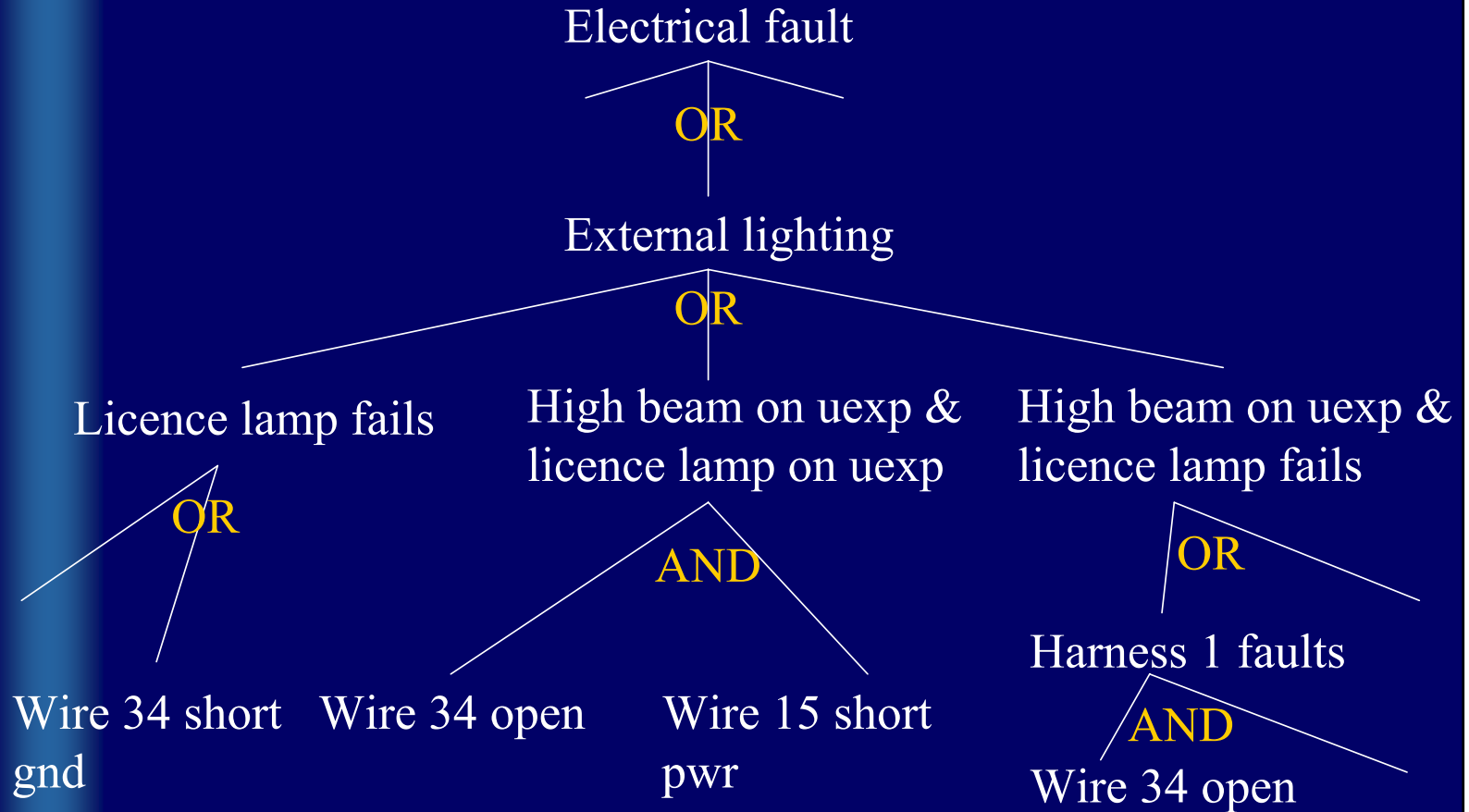
Design Verification/ Sneak Circuit Analysis

- Does the system behave as intended?
- A specification is needed to answer this
 - Followed by a comparison with actual or simulated behaviour
 - Assuming it is feasible to test all behavioural states. (software...)
- Difficulty is matching level of detail in specification and simulation

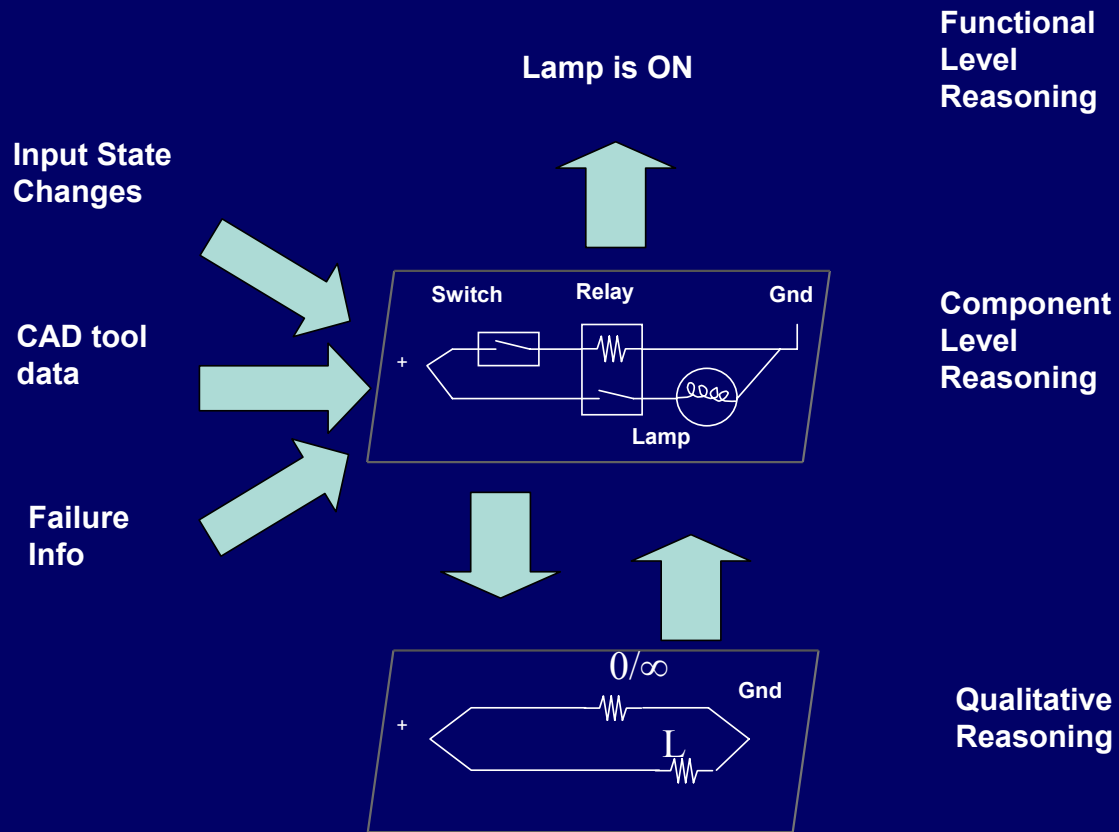
Fault Tree Analysis

- What are the causes of a specific event (faulty operation)
- More than one event or component fault may need to occur to cause a top level fault
- Logic gates are used to describe the relationship between fault and causes
- Can be derived from a *consistent* multiple fault FMEA

Partial fault tree



Modelling Architecture



Using the tool

Using AutoSteve - Simulation

- User can change input settings
 - Activate switches and sensors
 - See what activity occurs in the circuit
 - Functions are reported

Logical Cable

MGC File Edit Cable Setup AutoSteve Miscellaneous Libraries Check Report View Help

Sel: 0 (W|DAE) (Airbag_Circuit | schematic | sheet1) (C:/MentorSchematics/Airbags/Airbag_Circuit/default | /) (3.1809, -6.0414)

Logical Cable#1 Airbag_Circuit sheet1 (Design Context)

icable add route


SESSION ADD/ROUTE
 TEXT DRAW
 CABLE CONNECTOR
 CLS DIAG. HIER.
 VAROPTS MISC






DELETE UNDO
 MOVE COPY
 UNSELECT ALL SET SELECT FILTER

ROUTE SELECTED CHOOSE SYMBOL
 ADD WIRE ADD BUS/BUNDLE
 FLIP ROTATE

F1 Select Area Any Select Vertex Reopen Select	F2 Unselect All Unselect Area Move	F3 Add Wire Add Bus/Bundle Copy	F4 Popup Menu Reselect	F5 Place Symbol Add Property Check Sheet	F6 Set Grid Snap Connect All	F7 Sel Txt & Move Chg Text Value Open Up	F8 View Area View All Open Down	F9 Setup Session	F10 Pulldown Menu	F11 Read File Edit File	F12 Pop Window Close Window
---	---	--	------------------------------	---	------------------------------------	---	--	---------------------	----------------------	-------------------------------	-----------------------------------

Start Logical Cable Logical Cable SimulationTool untitled - Paint 7:38 PM

 at start


-  IGNITION_SWITCH_D.setting = Off Start Ign_on Auxiliary
-  HORN_SWITCH.position = open closed
-  MAIN_CRASH_SENSOR.impact = not_detected detected
-  SIDE_SENSOR_PASSENGER_G180.impact = not_detected detected
-  SIDE_SENSOR_DRIVER.impact = not_detected detected


Simulate **Reset**


Step	Description	Results


Labels Colour Components **View Result**


Event **Failure Modes** Simulation Displays


 at start

 IGNITION_SWITCH_D.setting = Off Start Ign_on Auxiliary

 HORN_SWITCH.position = open closed

 MAIN_CRASH_SENSOR.impact = not_detected detected

 SIDE_SENSOR_PASSENGER_G180.impact = not_detected detected

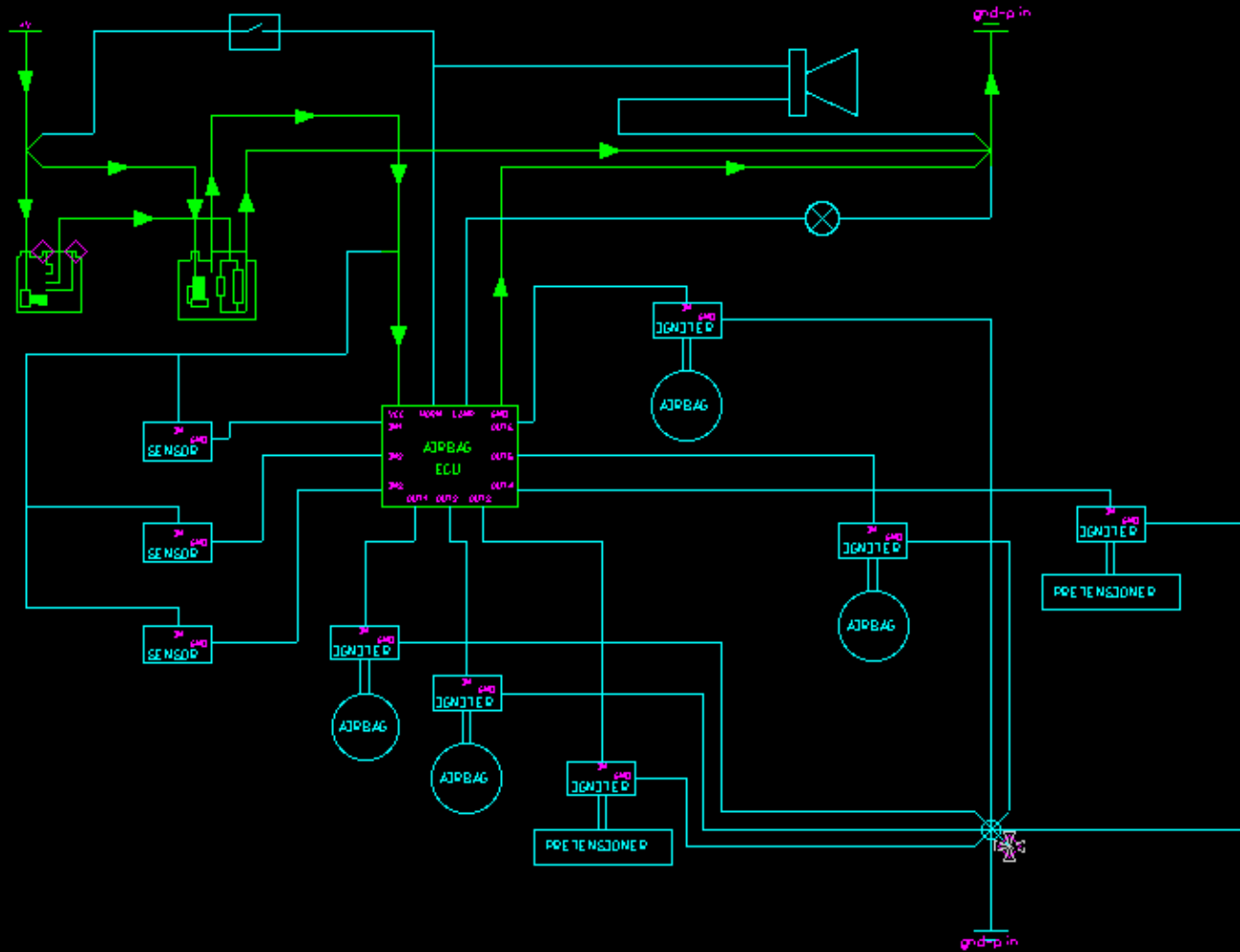
 SIDE_SENSOR_DRIVER.impact = not_detected detected

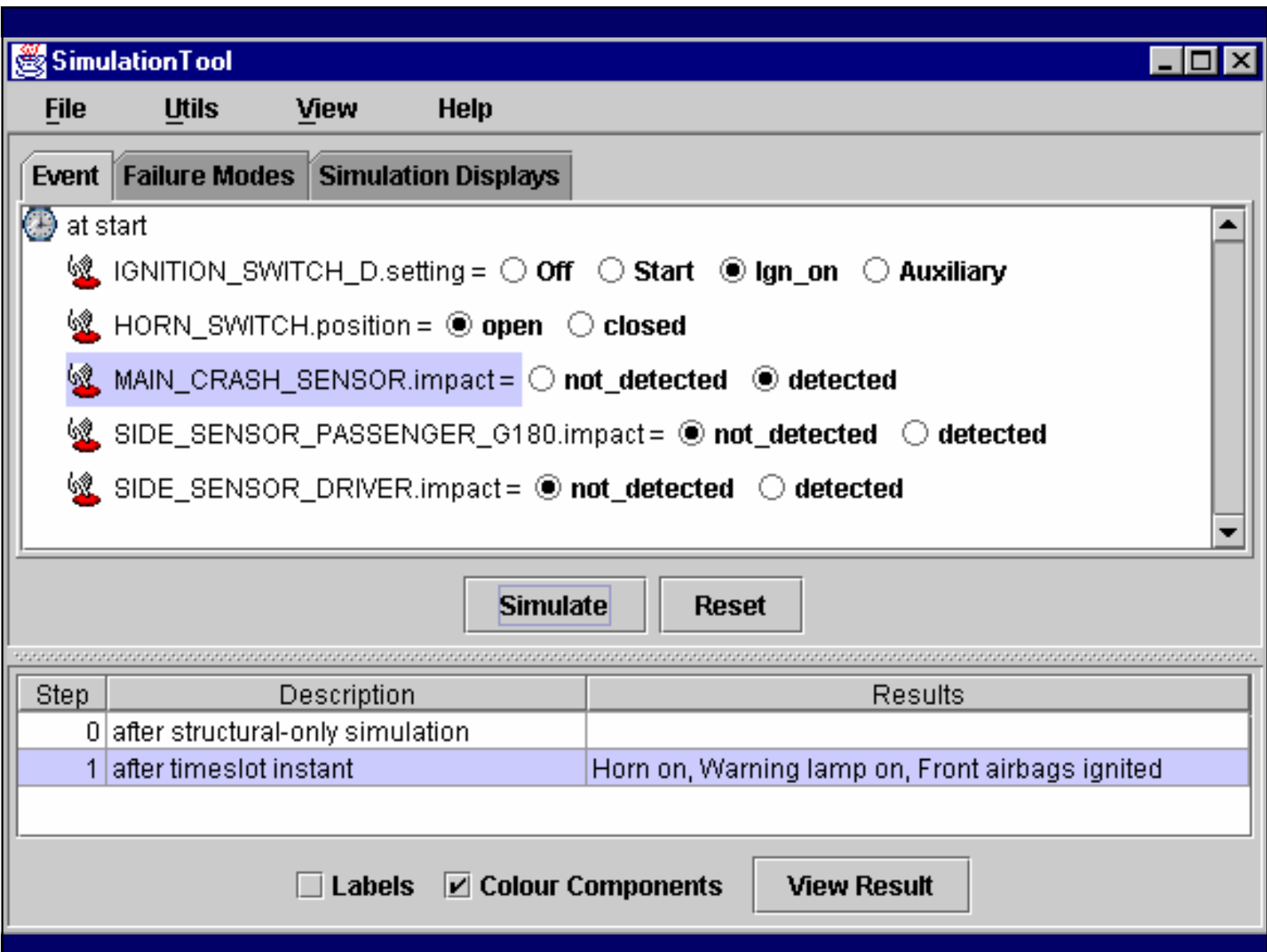
Simulate **Reset**

Step	Description	Results
0	at start	
1	after structural-only simulation	
2	after timeslot instant	

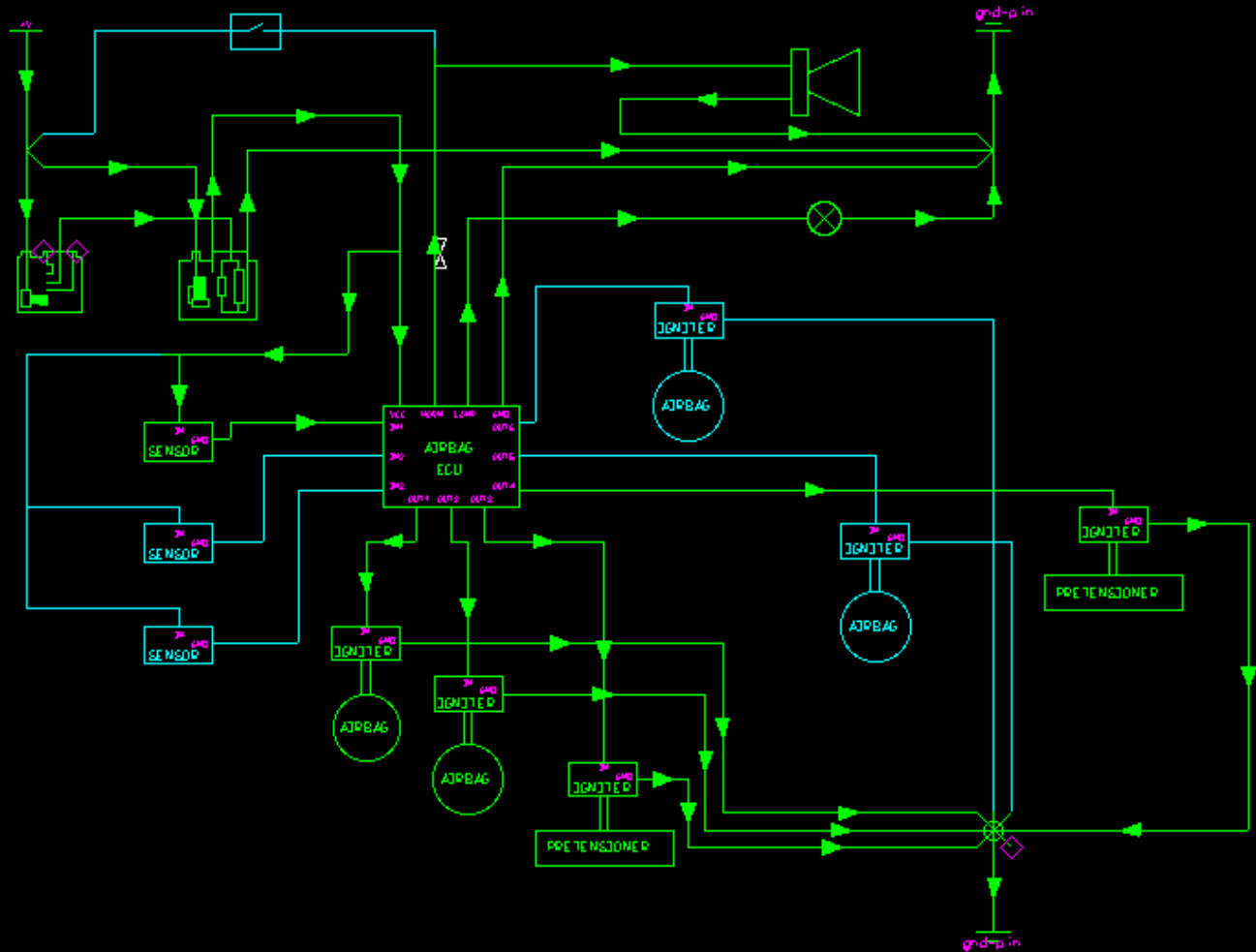
Labels Colour Components **View Result**

Logical Cable#1 Airbag Circuit sheet1 (Design Context) &





Logical Cable#1 Airbag_Circuit sheet1 (Design Context) &



Using AutoSteve

– Failure Simulation









- User can impose failures on components
 - Observe changed activity
- Failure behaviour of failure is part of the model of the component

SimulationTool [_] [□] [×]




File **U**tils **V**iew **H**elp

Event **Failure Modes** **Simulation Displays**

Components

-  WIRE18
-  WIRE21
-  WIRE14
-  WIRE28
-  WIRE29
-  WIRE7
-  WIRE4
-  WIRE12

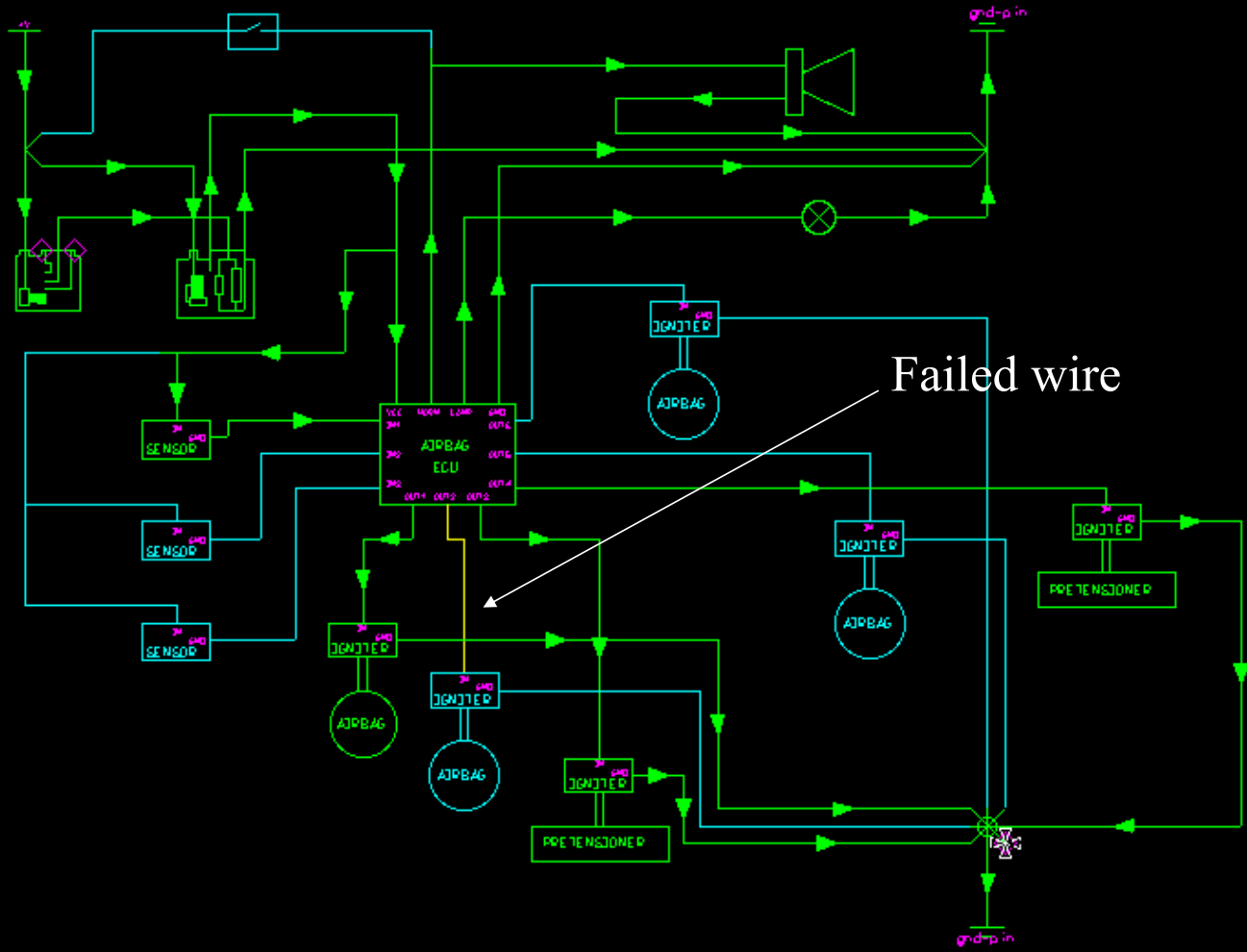
Current Settings

Failure Mode	Setting
 open circuit	<input checked="" type="radio"/>
 short to battery	<input type="radio"/>
 short to ground	<input type="radio"/>

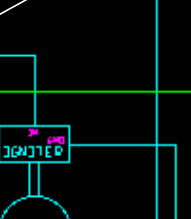
Simulate
Reset

Step	Description	Results
0	at start	
1	after structural-only simulation	
2	after timeslot instant	
3	after timeslot instant	Horn on, Warning lamp on, Driver front airbag ignited

Labels
 Colour Components
View Result



Failed wire



Effect of firing main crash sensor when ignition on

Step	Description	Results
0	at start	
1	after structural-only simulation	
2	after timeslot instant	
3	after timeslot instant	Horn on, Warning lamp on, Front airbags ignited

Effect if wire 29 has failed open circuit

Step	Description	Results
0	at start	
1	after structural-only simulation	
2	after timeslot instant	
3	after timeslot instant	Horn on, Warning lamp on, Driver front airbag ignited

This provides the basis for diagnosis & FMEA generation...

Using AutoSteve - FMEA

- FMEA - failure mode and effects analysis
- Investigate implications of every possible component fault on the overall system
- Calculate the RPN from the product of:
 - severity
 - ability to detect
 - likelihood of occurrenceof each possible fault

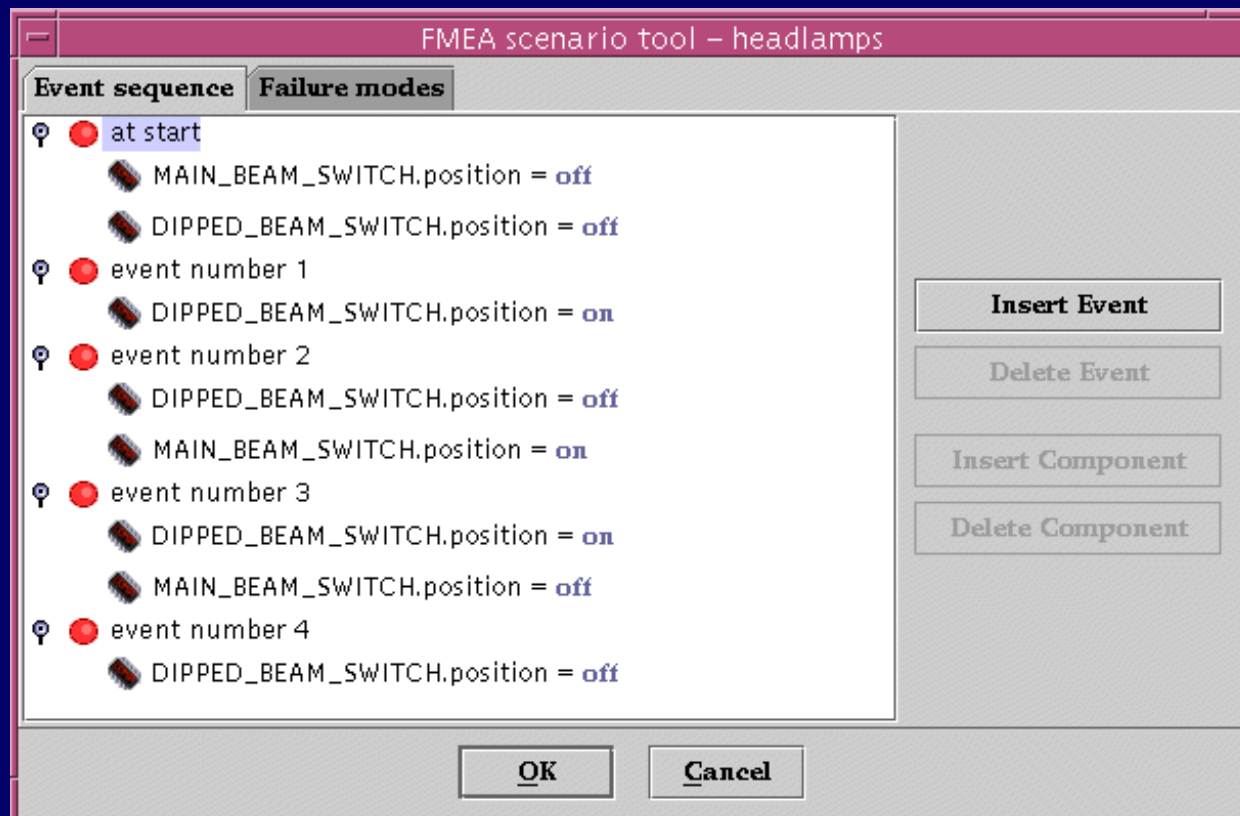
FMEA is based on simulation

- Simulate behaviour for correct circuit
- Abstract the results (functions)
- Test all possible faults for each component
- For each component fault:
 - make a faulty version of the circuit
 - simulate circuit behaviour / abstract the results
 - Compare with correct behaviour
 - Functional differences are what is significant

Set up FMEA scenario

- Decide on set of switch changes needed to test the circuit
- It will usually involve AT LEAST turning all switches/sensors on and off
- Possible to change several switches at once
- Decide which kinds of failures to try
 - multiple failures

Scenario screen



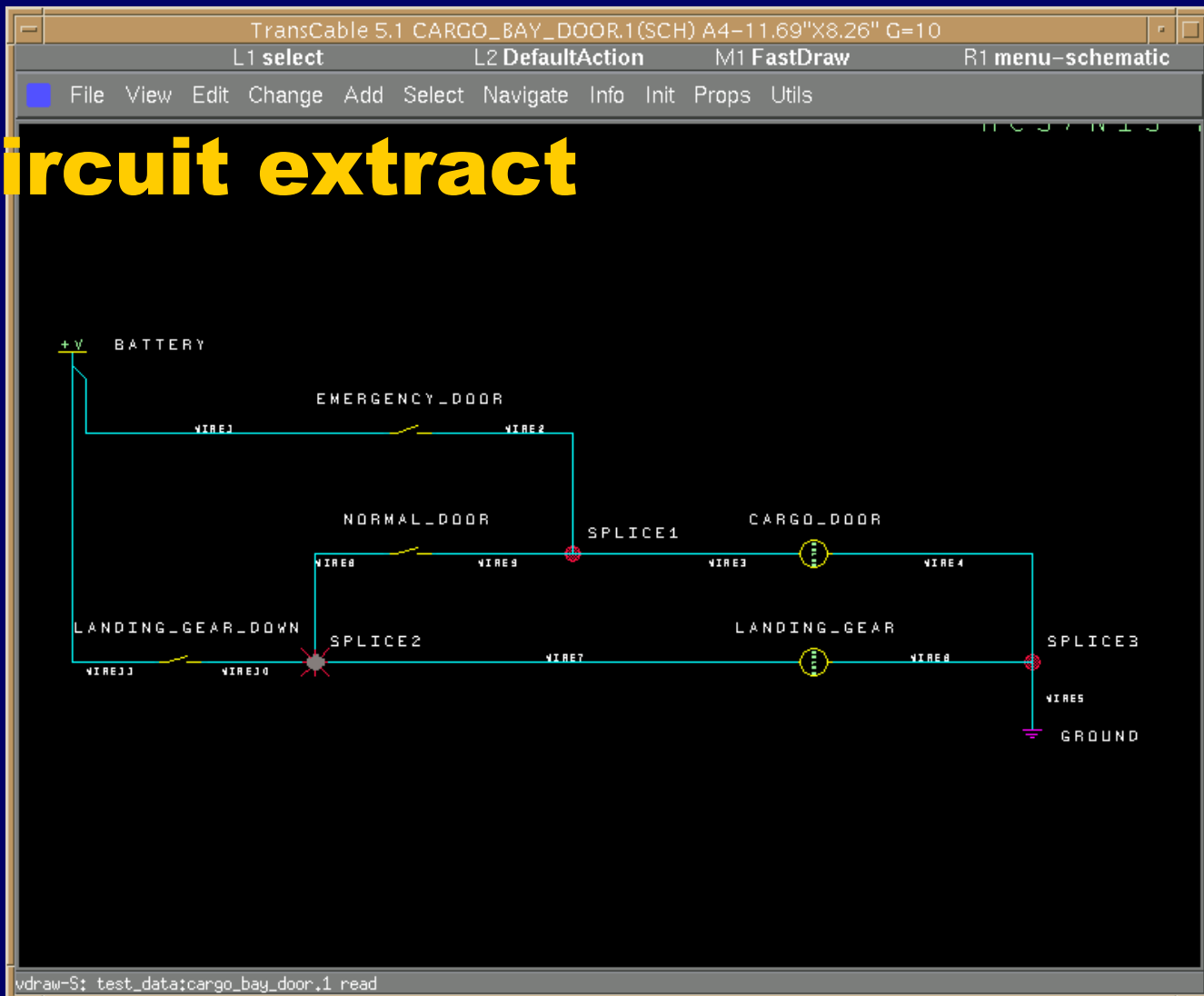
Item/Fn	Name	Failure	Potential Failure Mode	Potential Failure Effect	Sev	Potential Failure C
(113)	SPLICE2	fractured	Regardless of any event change, the "Horn on", "Front airbags ignited", "Warning lamp on", "Driver side airbag ignited" and "Side airbags ignited" functions were never achieved.	Driver can no longer alert other drivers to his/her presence with the horn. The driver and passenger will have severely reduced airbag crash protection. Driver will not know if the airbags are working. The driver will have reduced airbag crash protection. The driver and passenger will have reduced airbag crash protection.	10	The component SPLICE: failure fractured.
(117)	PASSENGER_AIRBAG_IGNITER_N135	Fails to ignite	Regardless of any event change, the "Front airbags ignited" function was never achieved.	The driver and passenger will have severely reduced airbag crash protection.	10	The component PASSENGER_AIRBAG_I has failure fails to ignite.
(118)	DRIVER_AIRBAG_IGNITER_N95	Fails to ignite	Regardless of any event change, the "Front airbags ignited" function was never achieved.	The driver and passenger will have severely reduced airbag crash protection.	10	The component DRIVER_AIRBAG_IGNIT has failure fails to ignite.
(119)	SPLICE5	fractured	When setting on IGNITION_SWITCH_D was set to Ign_on and impact on MAIN_CRASH_SENSOR was set to detected (4) the "Horn on" function was not achieved. Also, when setting on IGNITION_SWITCH_D was set to Ign_on and impact on SIDE_SENSOR_DRIVER was set to detected (6) the "Horn on" function was not achieved. Further, when setting on IGNITION_SWITCH_D was set to Start and impact on SIDE_SENSOR_PASSENGER_G180 was set to detected (8) the "Horn on" function was not achieved. Finally, regardless of any event change, the "Front airbags ignited", "Warning lamp on", "Driver side airbag	The driver and passenger will have severely reduced airbag crash protection. Driver will not know if the airbags are working. The driver will have reduced airbag crash protection. The driver and passenger will have reduced airbag crash protection. Driver can no longer alert other drivers to his/her presence with the horn.	10	The component SPLICE: failure fractured.

Records 0 to 19 of 139 Ready

Using Autosteve - Sneak Circuit Analysis

- Unexpected functionality
- Usually caused by unintentional reverse current flow
- Can be detected if 'sneak expressions' are added
- Example:
Transport Aircraft Cargo Bay doors

Circuit extract



Defining Function Inputs

The screenshot shows the 'cargo_bay_door - SubsystemEditor' window. The interface includes a menu bar (File, Edit, View, Tools, Help), a toolbar with icons for file operations, and a 'Function Model' section with a path field containing '/rcs/test/cargo_door'. Below this are three tabs: 'Function expressions' (selected), 'Sneak expressions', and 'Saber settings'. The 'Function Label' dropdown is set to 'cargo_door_open'. The 'Sneak Expression' field contains the following logic: `(LANDING_GEAR_DOWN.property.position = closed and NORMAL_DOOR.property.position = closed) or EMERGENCY_DOOR.property.position = closed`. To the right of the expression is a 'Component' dropdown menu showing a tree view with 'other', 'EMERGENCY_DOOR', and 'LANDING_GEAR_DOWN'. At the bottom, the 'Valid simulation types' section has three checkboxes: 'Qualitative' (checked), 'Quantitative', and 'Common'. A status bar at the very bottom shows a red warning icon and the text 'Ready.'

File Edit View Tools Help

Function Model

/rcs/test/cargo_door

Function expressions Sneak expressions Saber settings

Function Label

cargo_door_open

Sneak Expression

```
(LANDING_GEAR_DOWN.property.position = closed  
and NORMAL_DOOR.property.position = closed) or  
EMERGENCY_DOOR.property.position = closed
```

Component

- other
 - EMERGENCY_DOOR
 - LANDING_GEAR_DOWN

Valid simulation types

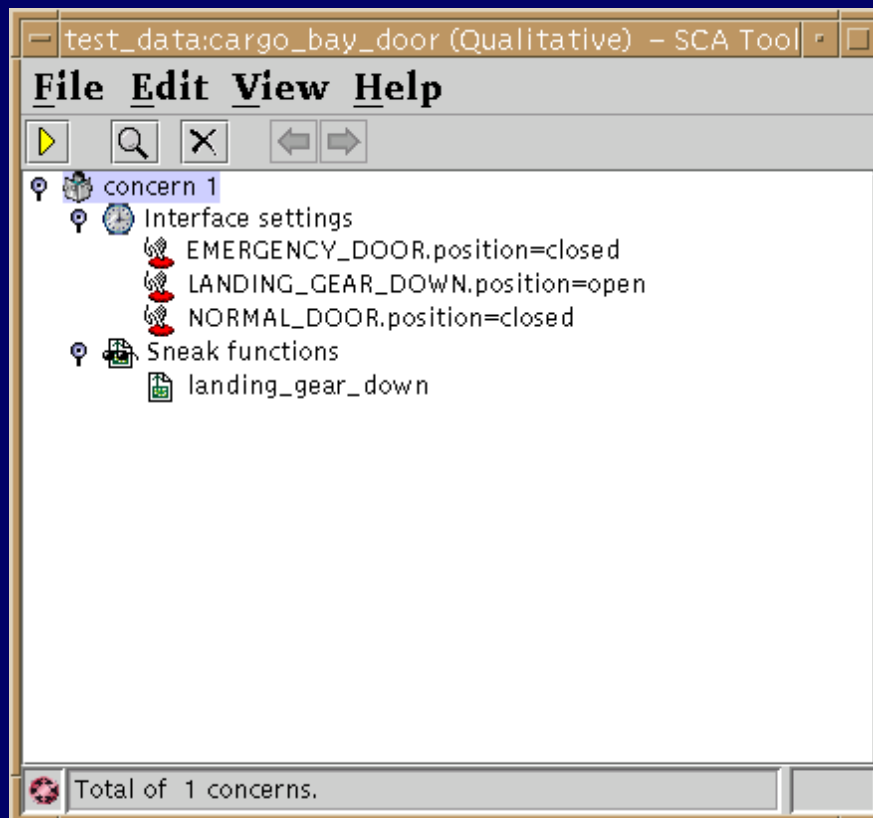
Qualitative Quantitative Common

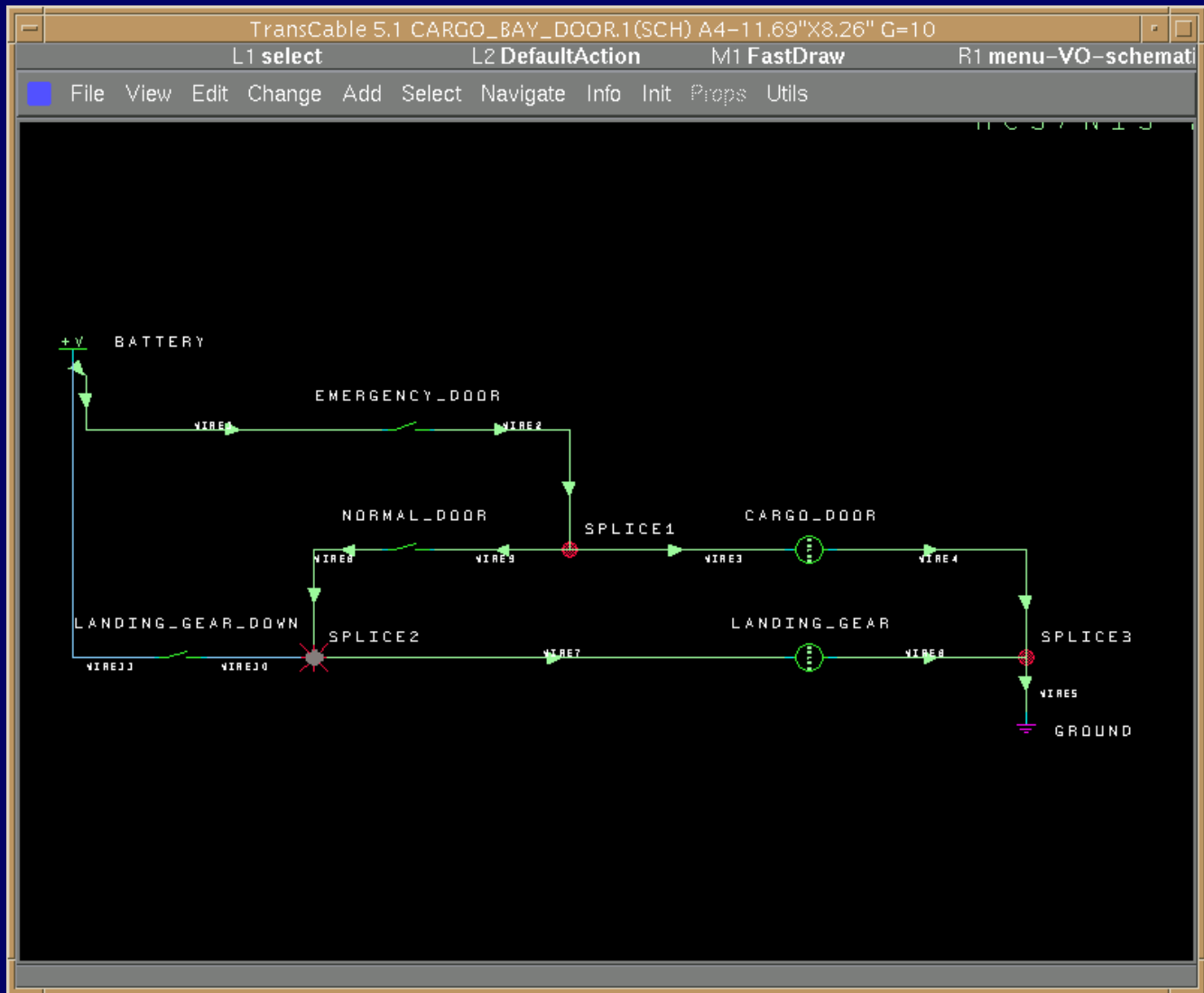
Ready.

The Analysis

- Circuit is simulated for all input combinations (switch positions).
- Active functions are abstracted from the simulation.
- These are compared with the expected functions derived from the inputs.
- Any discrepancies are reported.
- A form of design verification.

Results





Modelling

What does a component definition look like?

● Terminals:

- Inputs / outputs for the component
- Ports where other components can be connected

● Internal topology of component:

- Described in terms of links between terminals
- Can include logical resistors
- Resistance value can change depending on the state

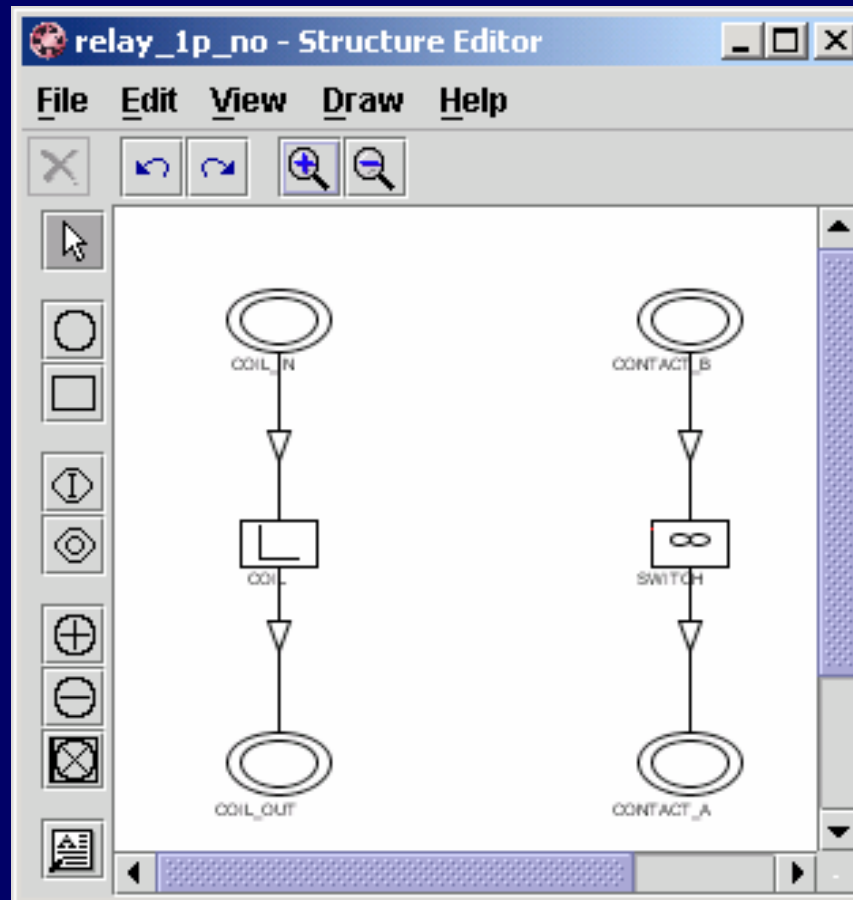
● Dependencies / state chart

- Define changing values of internal resistors of a component as the state of component changes

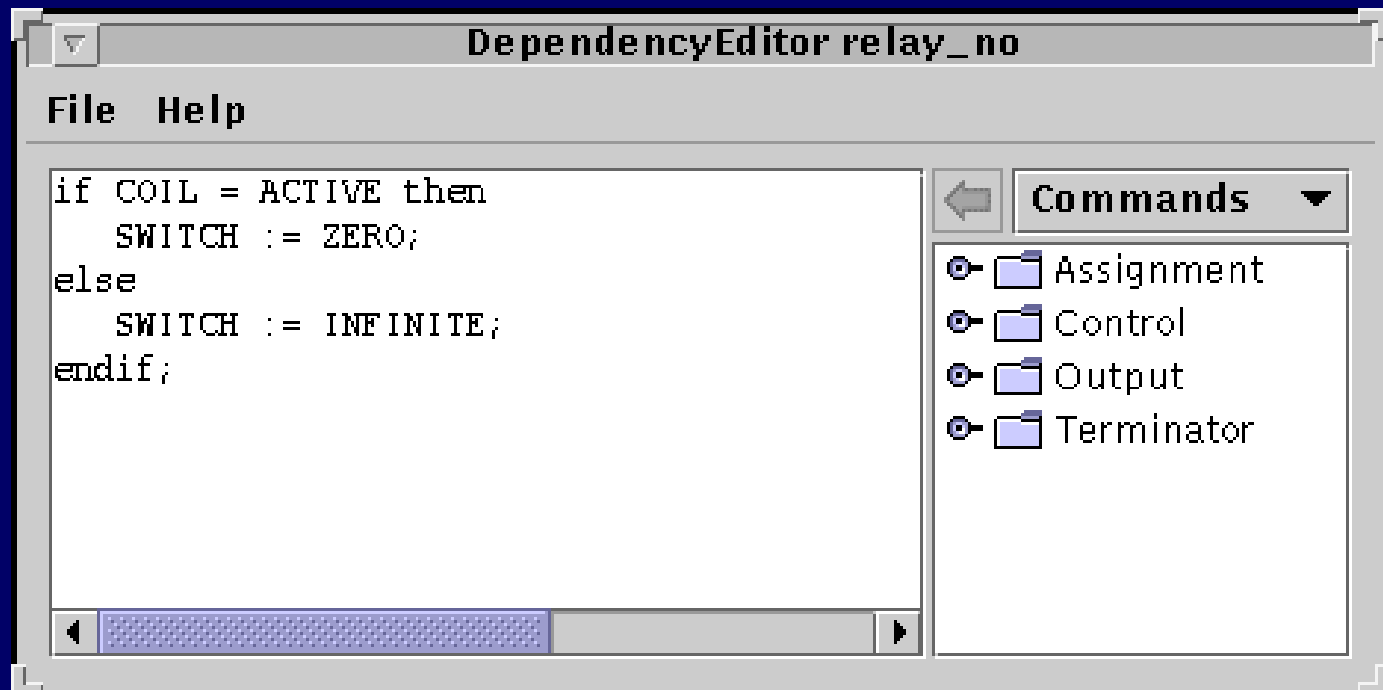
Steps in building a component

- Build component **structure**
- Build component **behaviour**
 - Either as dependencies
 - Or as state chart
- Add any **failure modes** needed

Structure of a relay component

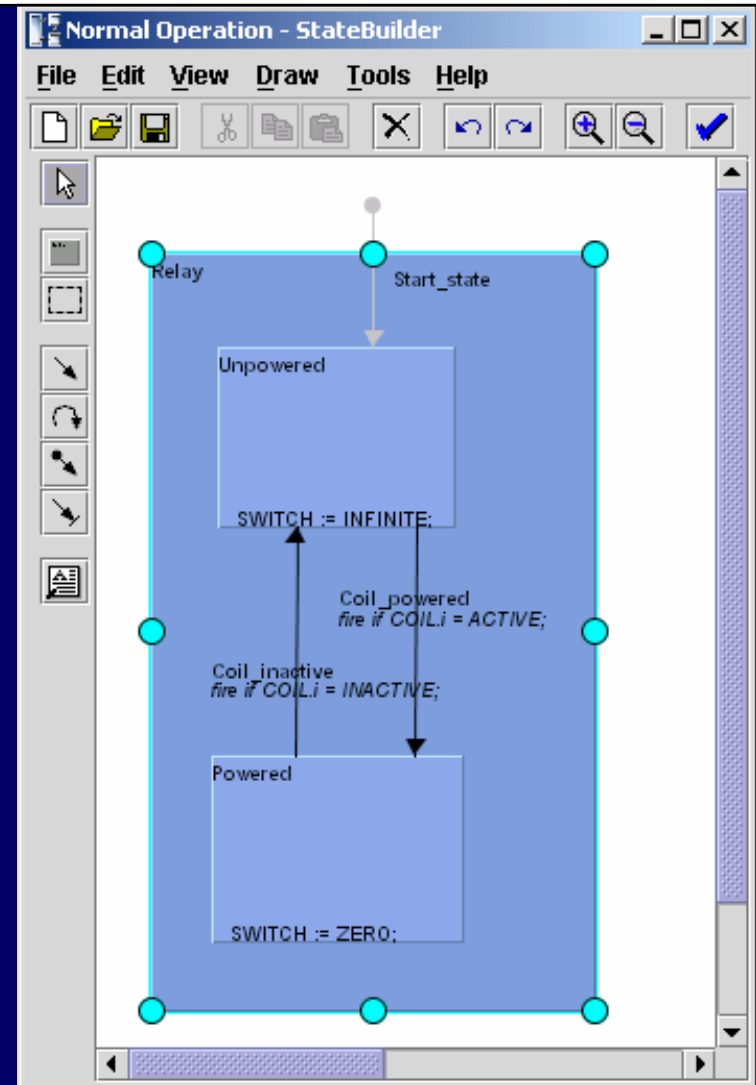


Component behaviour - for relay



Alternative behaviour

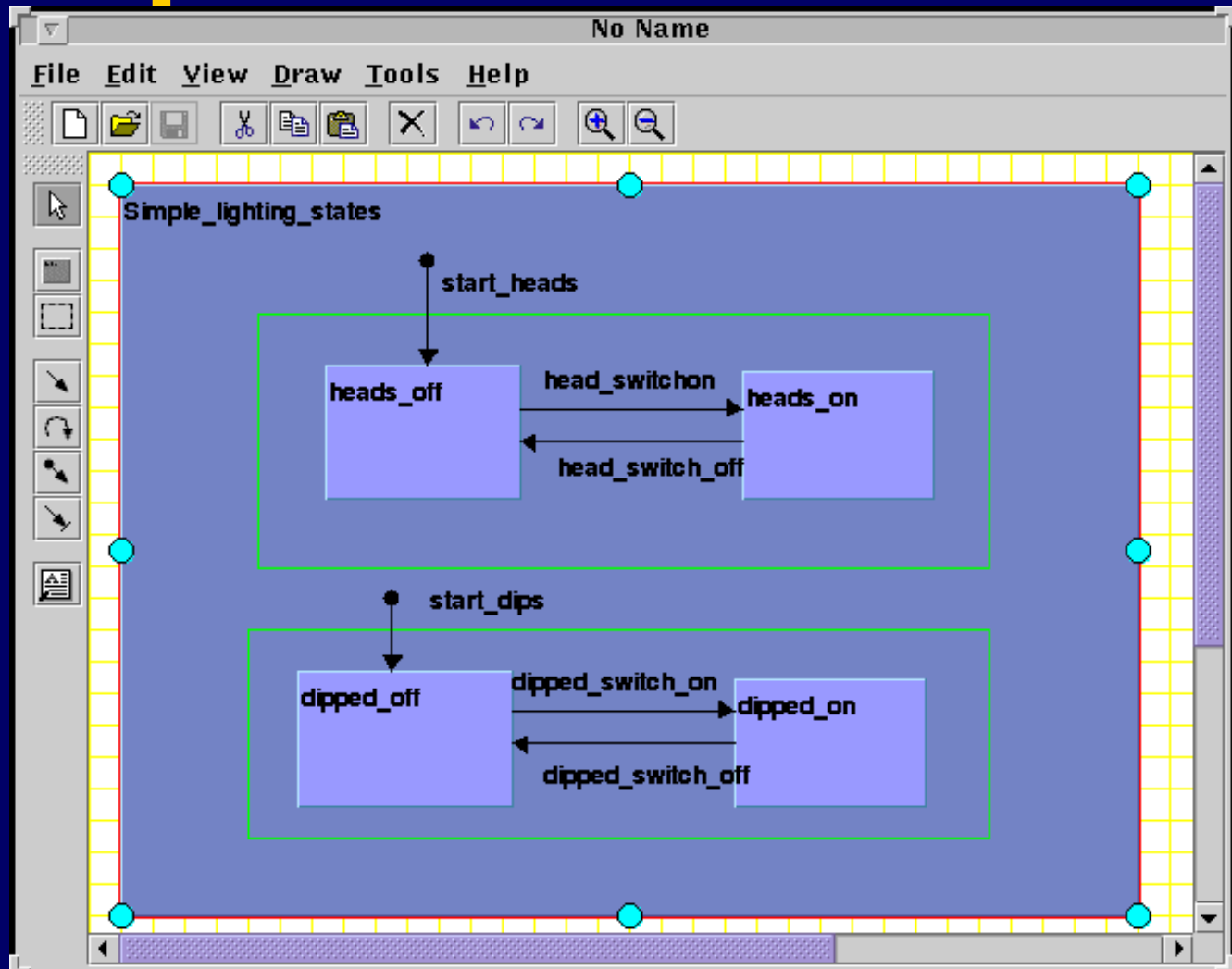
- Provides for explicit state within component models
- Allows complex models of ECU (software)
- Allows introduction of time
 - Eg "fire after mS"



Time is represented in the state based behaviour using an order of magnitude representation

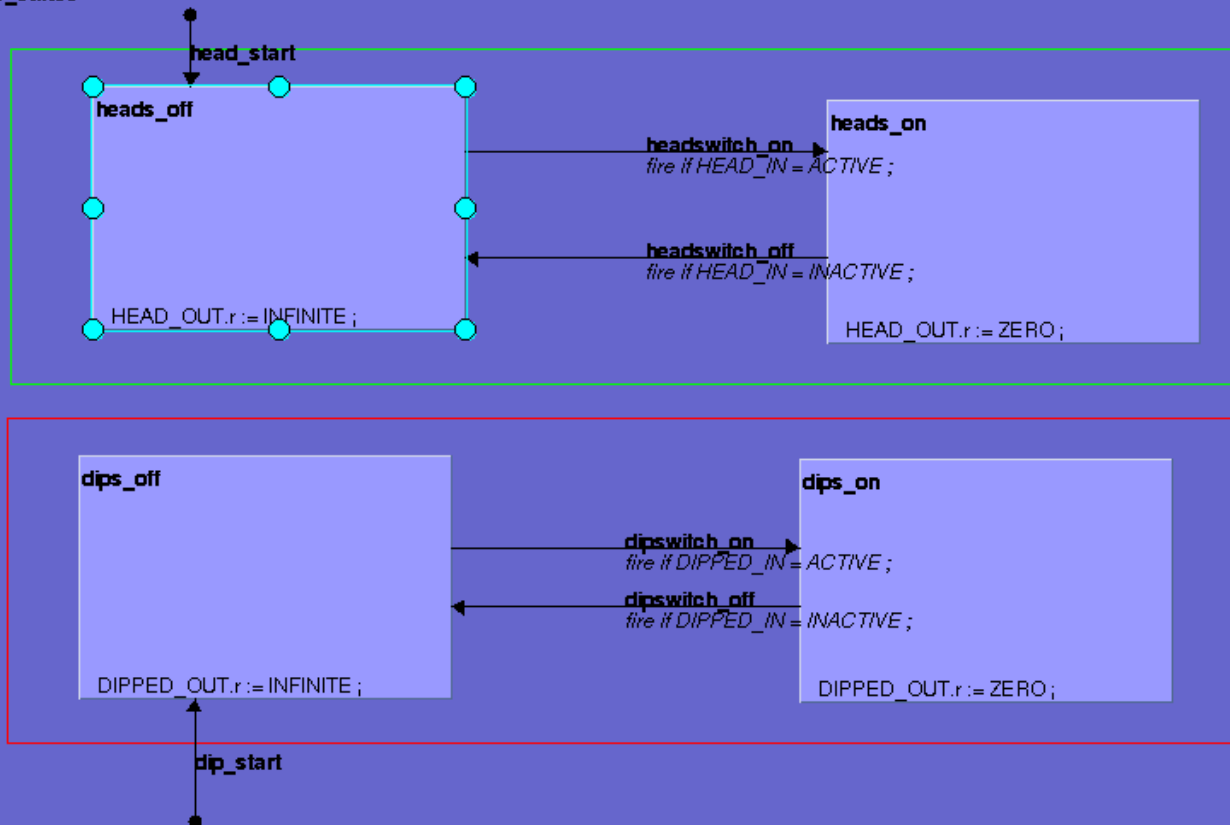
- For automotive systems useful levels have been defined:
 - Instant
 - μS (ECU operations)
 - mS (relay switching)
 - S (user interaction)
 - hour (battery discharge)
- The state based simulation has knowledge of all components and orders events according to timeslot.
- Events within a timeslot are considered concurrent.

Simple ECU state chart



Linking to structure

lights_states



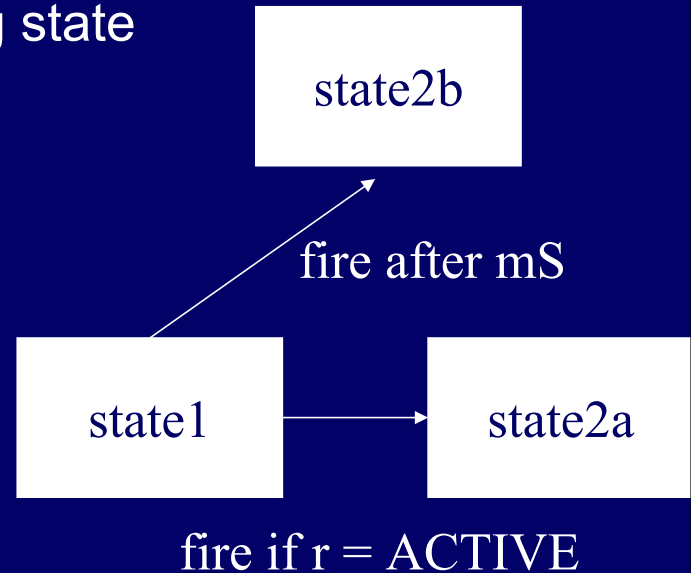
Events and actions

States

- action expressions
 - Executed when entering state

Events

- condition expressions
 - fire if
 - fire after
 - fire after if
- action expressions



Conditional Expressions

- Can be used to interrogate:
 - structure
 - resistance values
 - current flows
 - behaviours
 - interface properties
 - recent events (how was this state achieved?)
 - component variables (local values)
- example event condition
 - fire if motor.i=FORWARD;

Action Expressions

● Can be used to modify

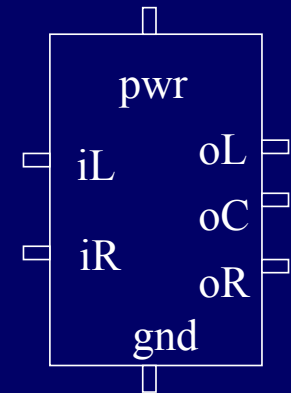
- structure
 - resistance values
- behaviour
 - interface properties
 - component variables

● example

- lockSwitch.r:=INFINITE
- contact1:=ZERO (notice implied resistance...)

Exercise:

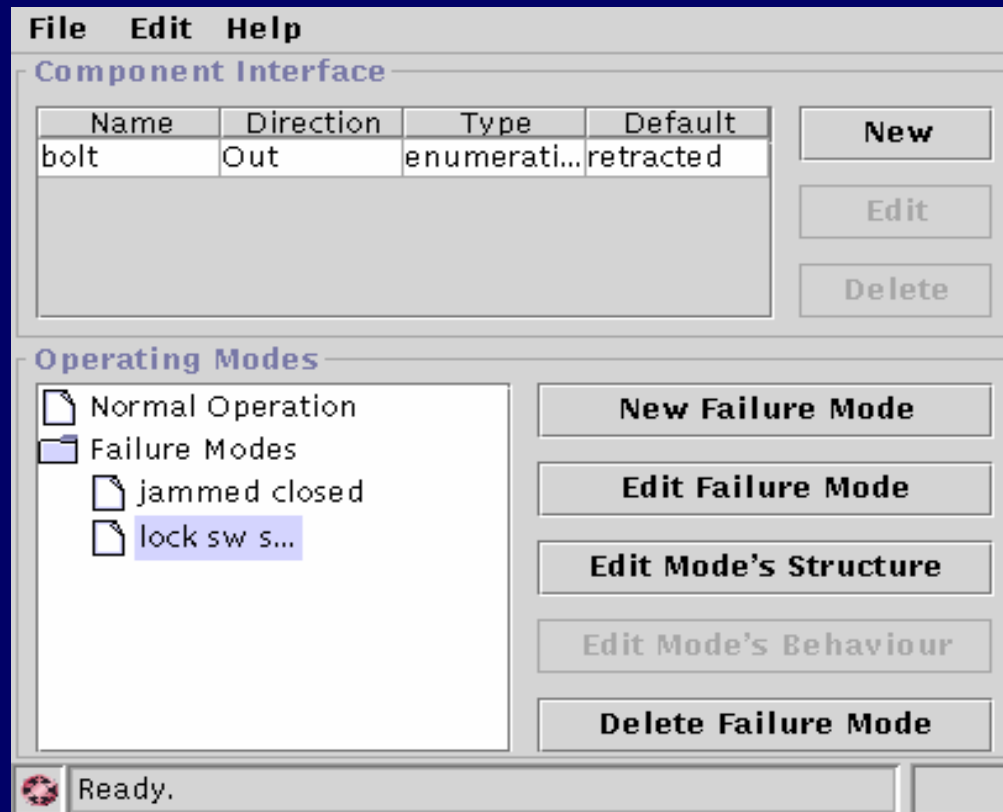
- Draw a state chart and structure for a simple Electronic Control Unit with:
 - 2 inputs (inputL and inputR)
 - 3 outputs (outputL, outputC, outputR)
- With behaviour:
 - Signal at inputL creates a path to gnd from outputL (open circuit otherwise)
 - Similarly for inputR
 - Signals at both inputs creates a path to gnd for outputc



Defining Failure modes

- Component behaviour will change when component fails
- Name the possible failure modes
- Define component behaviour for each failure mode
 - Usually by modification of non failure structure and/or behaviour

Failure Modes



Failure modes

- Decide on failure modes (experience)
- Change structure values (if needed)
- Either new state chart or none
- If new state chart, then often easiest to copy old one - we will make doing this easier in a future version
- None means NO BEHAVIOUR

Are failure models necessary ?

- Provides more specific information about system behaviour in failure situations.
- Works well where a fault causes additional behaviour.
- Prevents implausible explanations
- More accurate fault identification
- Provides the starting point for automated FMEA (no symptoms, unlike diagnosis)

Summary of components

- Independent behaviours can be defined for each component type
- Can use complex state-based representation to model sequential events, memory and 'qualitative time'
- Can model normal operation of mechanical or hydraulic interactions within an assembly
 - Eg relay, ECU.

Simulation Outline

- Generate list of all components in circuit and connections
- Set all components in default state
- Take input changes from user
- Calculate circuit changes (details on next slide)
- Abstract the results

Calculating circuit changes

- Analyse circuit topology to identify current flow
 - Depends on current state of each component
 - Uses component descriptions and “netlist”
 - CIRQ (next slide)
- Make any circuit changes necessary
 - e.g. relays may close, ECUs may change state
 - Use timing rules given for state charts
- Repeat until circuit reaches equilibrium or is circling through identical states

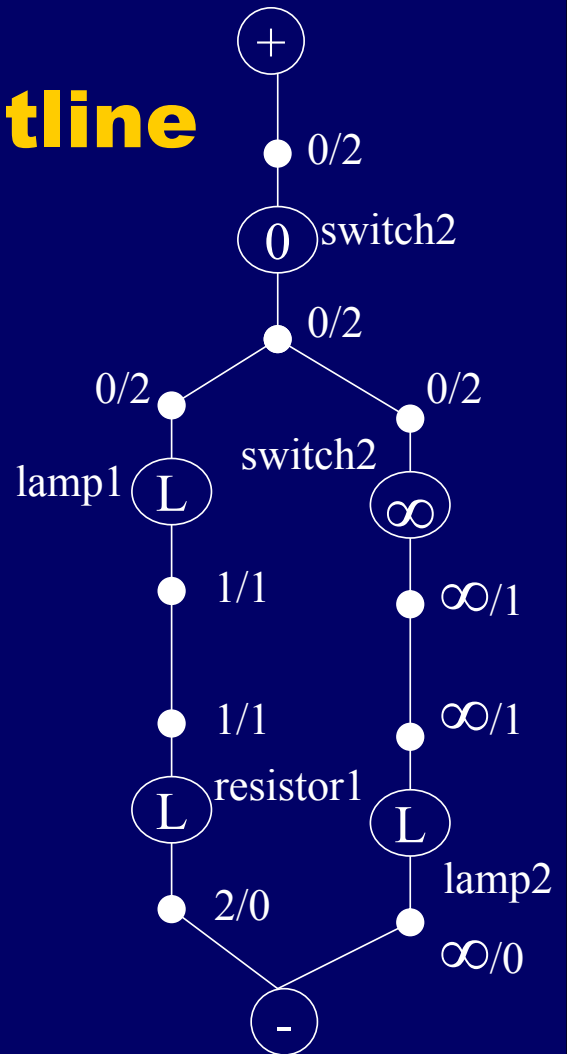


CIRQ – the network analyser

- Calculates activity in a qualitative resistance network
- Based on Dijkstra's shortest path algorithm
 - Stage 1 label all terminals with load count from power/ground (f/r)
 - Stage 2 find activity using depth first traversal

Path finding outline

- Terminal f/r values including ∞ are INACTIVE
- Short paths have the same non infinite f/r values at both ends
 - Components on other branches between these points are INACTIVE



Function (behaviour interpretation)

We have a global circuit simulator
and local non electrical behaviour
but how to give answers in terms
meaningful to the engineer?

Abstracting the results

- Result of simulation is state of each component at each point in time - too much detail
- Want to know significant operation of circuit - *functions* allow you to abstract this
- Functions recognise characteristic overall behaviour of the system

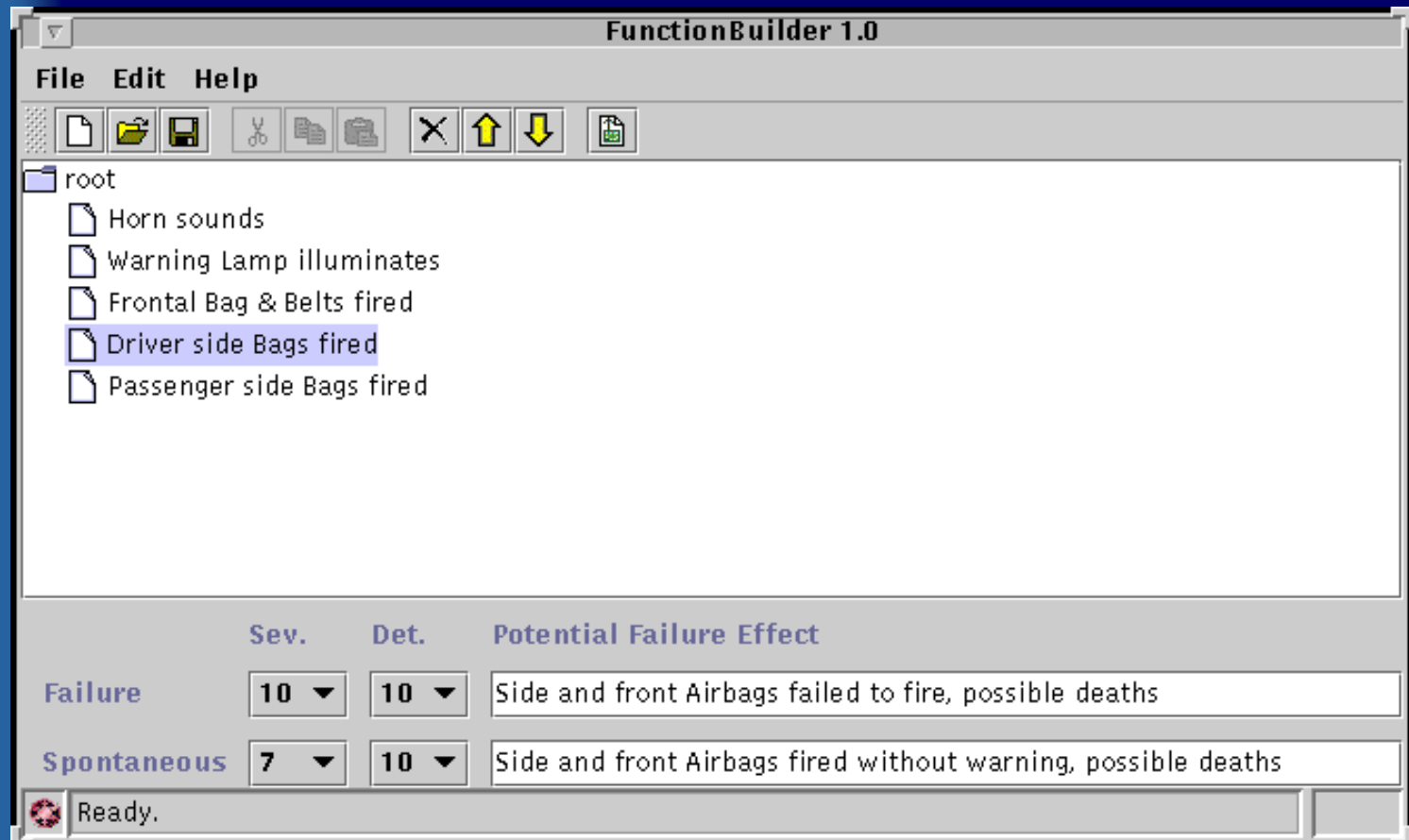
Function Labels

- Form the system level model
- Function labels can be reused between versions of a system
- Must be linked to the structure (system outputs)

Engineers decide on functions

- What does the circuit achieve?
- For example, airbags:
 - Horn sounds
 - Drivers front airbag and belt fires
 - Warning lamp illuminates
 - etc
- These functions and associated RPN values must be defined (they can be reused)

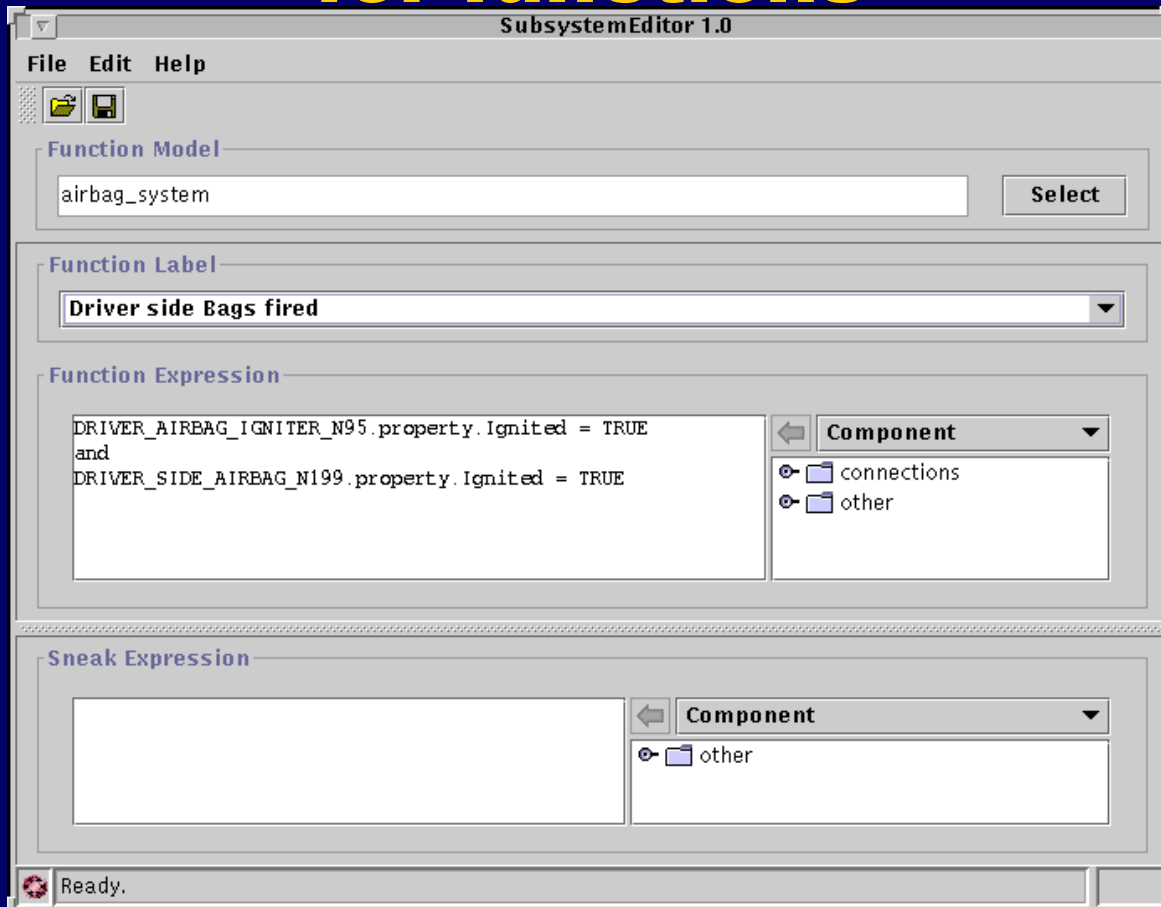
Setting up circuit functions

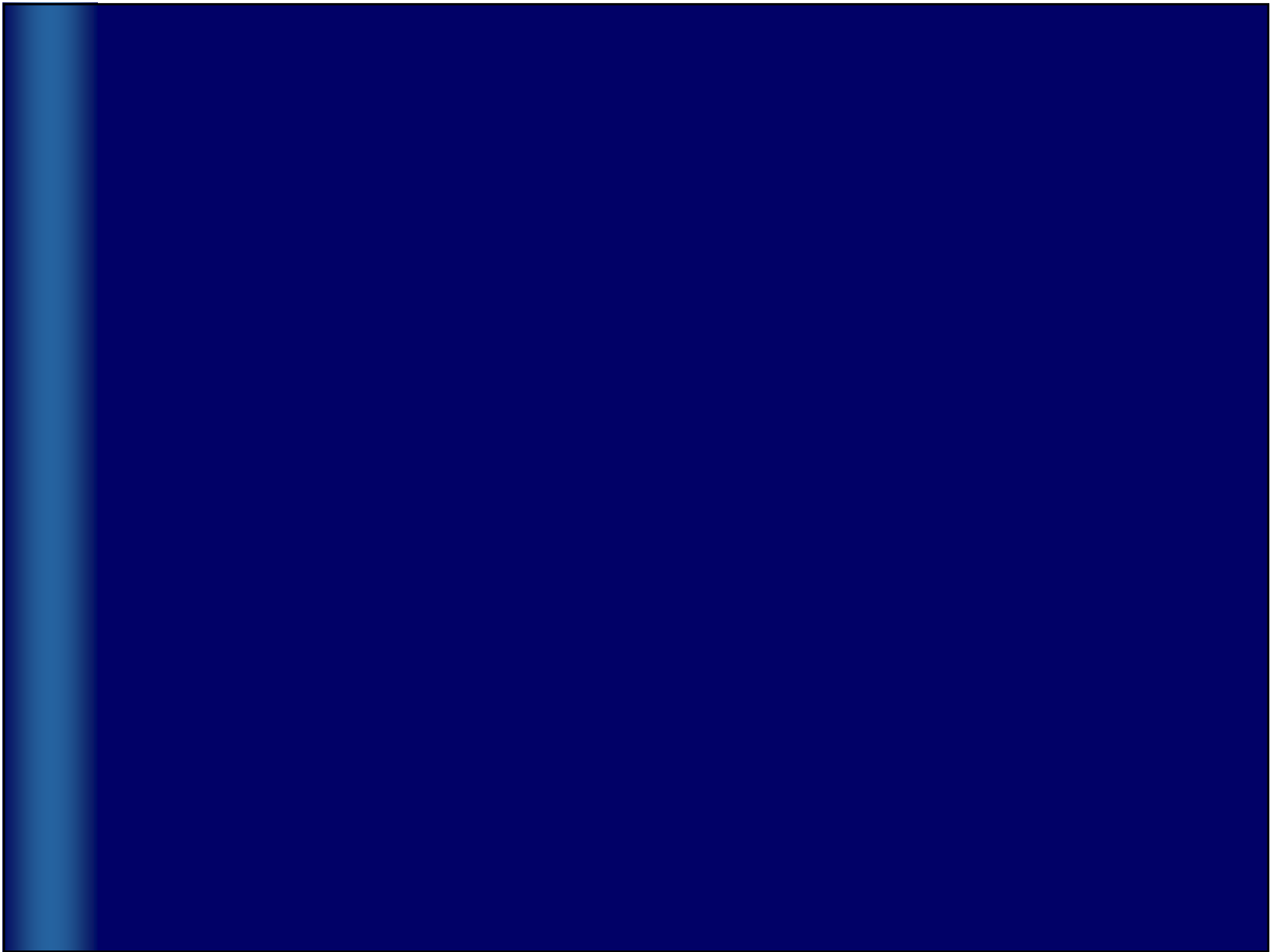


Link functions to circuit

- Decide how you would tell that the function was occurring if you had access to all electrical readings
- Example, central doorlocking:
 - tell DOORS LOCKING by current flowing through the doorlock motors in the correct direction
- Declare circuit activity for each function

Declaring circuit activity for functions





The architecture works because:

- Model building requires little effort.
- Can be done early in the design cycle.
- No physical prototypes are needed.
- Components are reusable and the library is small.
- Works when component details are vague.
- Function labels are reusable at system level
 - Link to structure requires only state identification

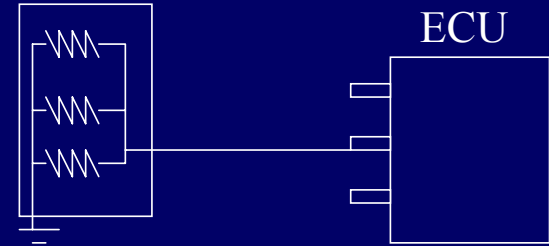
However drawbacks are:

- Results cannot always predict what will happen.
 - Does a fuse blow?
 - Which direction does a motor situated on a bridge circuit turn?
 - Does a lamp actually light up to required brightness?
- These questions can be answered as more data becomes available:
 - Multiple resistor levels
 - resistor values
 - SPICE/Saber

Other Issues

- "properties"

3 position switch



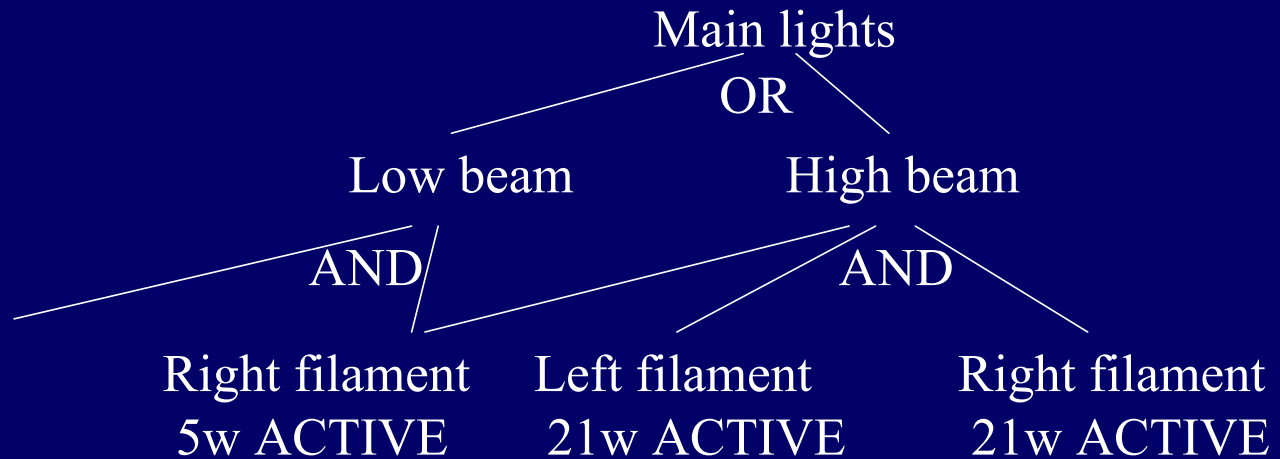
- Behaviour relying on voltage values
 - Eg. voltage level input signal to ECU
 - Implementation allowed the behavioural levels to communicate directly
 - Resulting in ineffectual failure modes unless the electrical status is correctly checked prior to allowing the communication.
 - Components no longer independently reusable

Issues -functions

- Sometimes it is important that a behaviour does *not* happen
 - High beam headlamp might be defined
 - "left high filament ACTIVE and right high filament ACTIVE"
 - So a failure causing only one lamp to remain on in the off state will generate
 - "all functions to be achieved as expected"

Hierarchical functions

- Can report an unexpected "left filament (high beam - main lights)" sub function



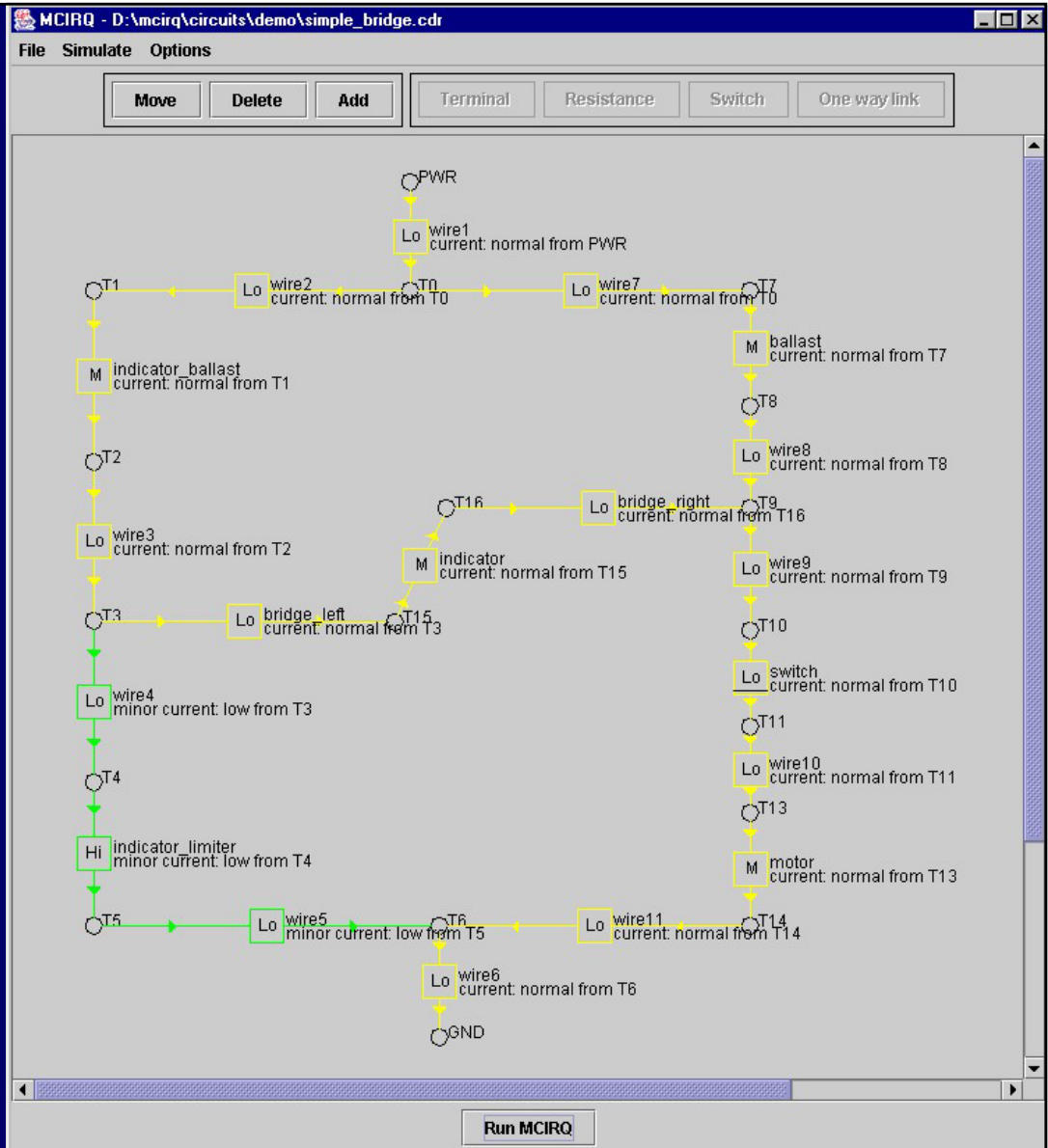
Challenges for electrical design analysis

- Design analysis for much larger systems
- Whole lifecycle design analysis
 - How can we target more specific questions (next slide)
- Making software more explicit
- New applications
 - Education and tutoring
 - Design of monitoring and recovery software
 - FMEA of fault mitigating behaviour

Additional Resistor levels

- 5 resistor levels can distinguish:
 - Information flow (ECU signals)
 - Activation of relays
 - Power flow for motors and lamps etc
- The **lowest reasoning level** is replaced with a new simulator ("MCIRQ" [Lee])
 - Any number of resistors in series has the qualitative value of the largest.
 - Resistances have to be modified in affected components (eg ECU's and Power devices)

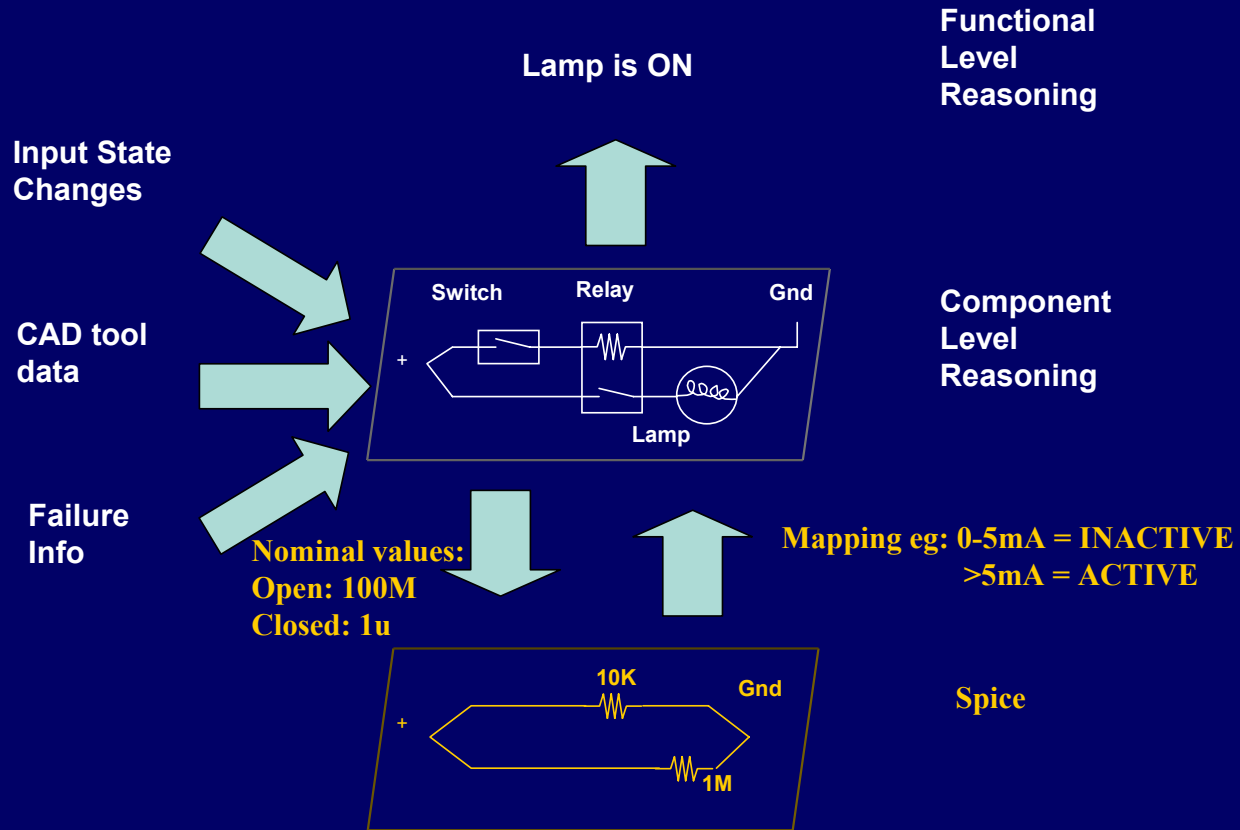
MCIRQ



Resistor values

- Wire lengths and gauges come once the spatial design is known.
- Supplier part numbers are also eventually known.
- The **lowest level** is again replaced (by SPICE)
 - The results are mapped back to the qualitative levels and the component state based behaviours work as before.
 - The mapping is done at the component level.
 - Hence "active" can be different ranges of current.

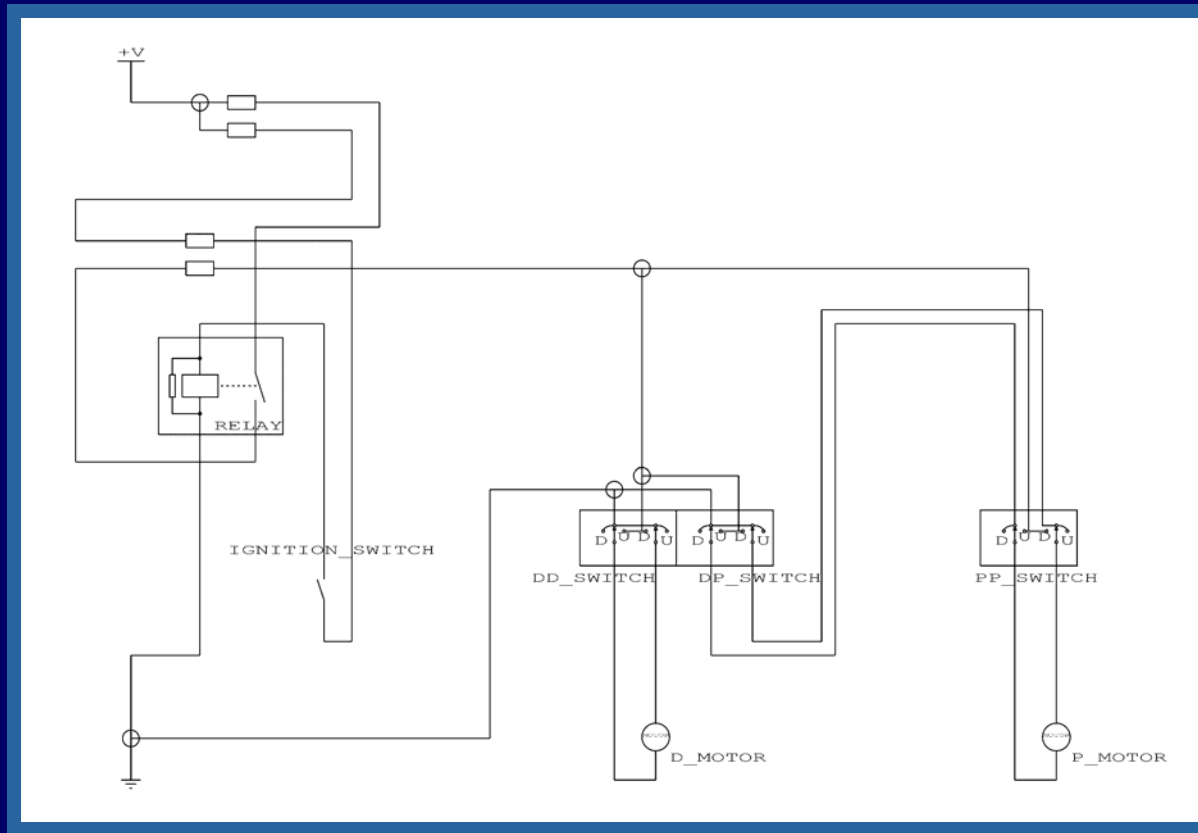
Modified Architecture



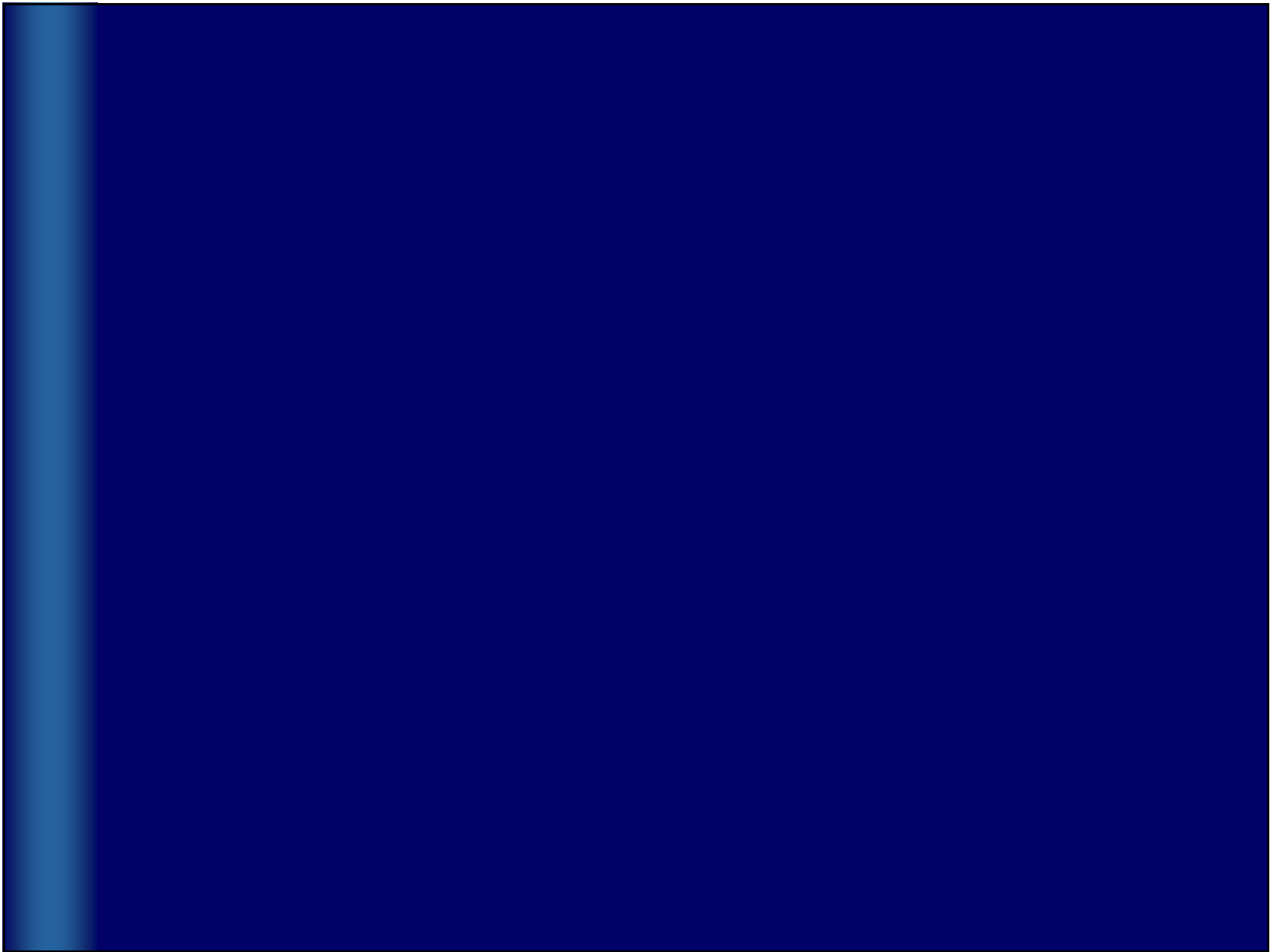
Full numerical simulation

- Replace lowest **two** levels
 - Mapping of results to qualitative levels allows generation of a meaningful FMEA report
 - Requires a complete set of detailed models
 - Appropriate to resolve specific problems or systems:
 - Eg motor inrush/stall current problems
 - Temperature related issues etc

Eg: Power Windows

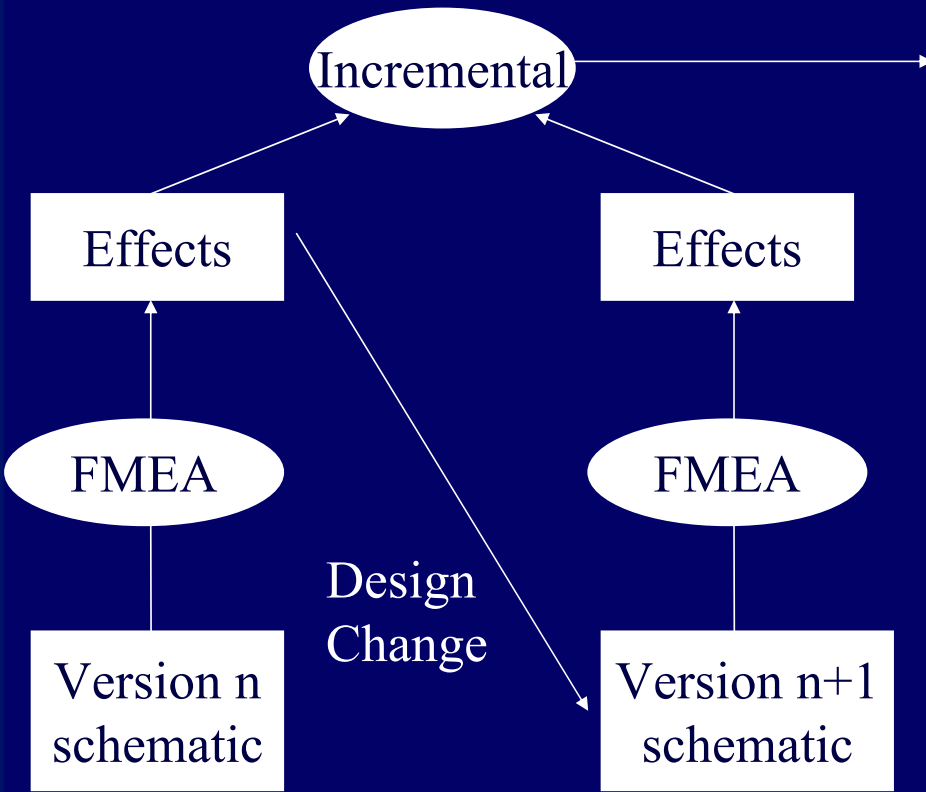


Test	3-leveled qualitative	Multi-leveled	Simple numerical	Complex numerical
Failure mode effects analysis (FMEA)	√	√	√	√
Sneak circuit analysis (SCA)	√	√	√	√
Design verification	√	√	√	√
Resolve current on bridges	x	√	√	√
No power current flow through ignition switch	x	√	√	√
Voltage drop across motor under constant load	x	x	√	√
Correct fuse blow under short circuit condition	x	x	√	√
Motor voltage balanced under normal operation	x	x	√	√
No fuse blow under motor inrush	x	x	x	√
No fuse blow under stall current for 5 seconds	x	x	x	√



Incremental analysis

- Incremental FMEA has proved useful because it only reports effects that have changed due to a design modification.
 - Consistency of report makes it possible
 - Only answers that have changed need be considered by an engineer.



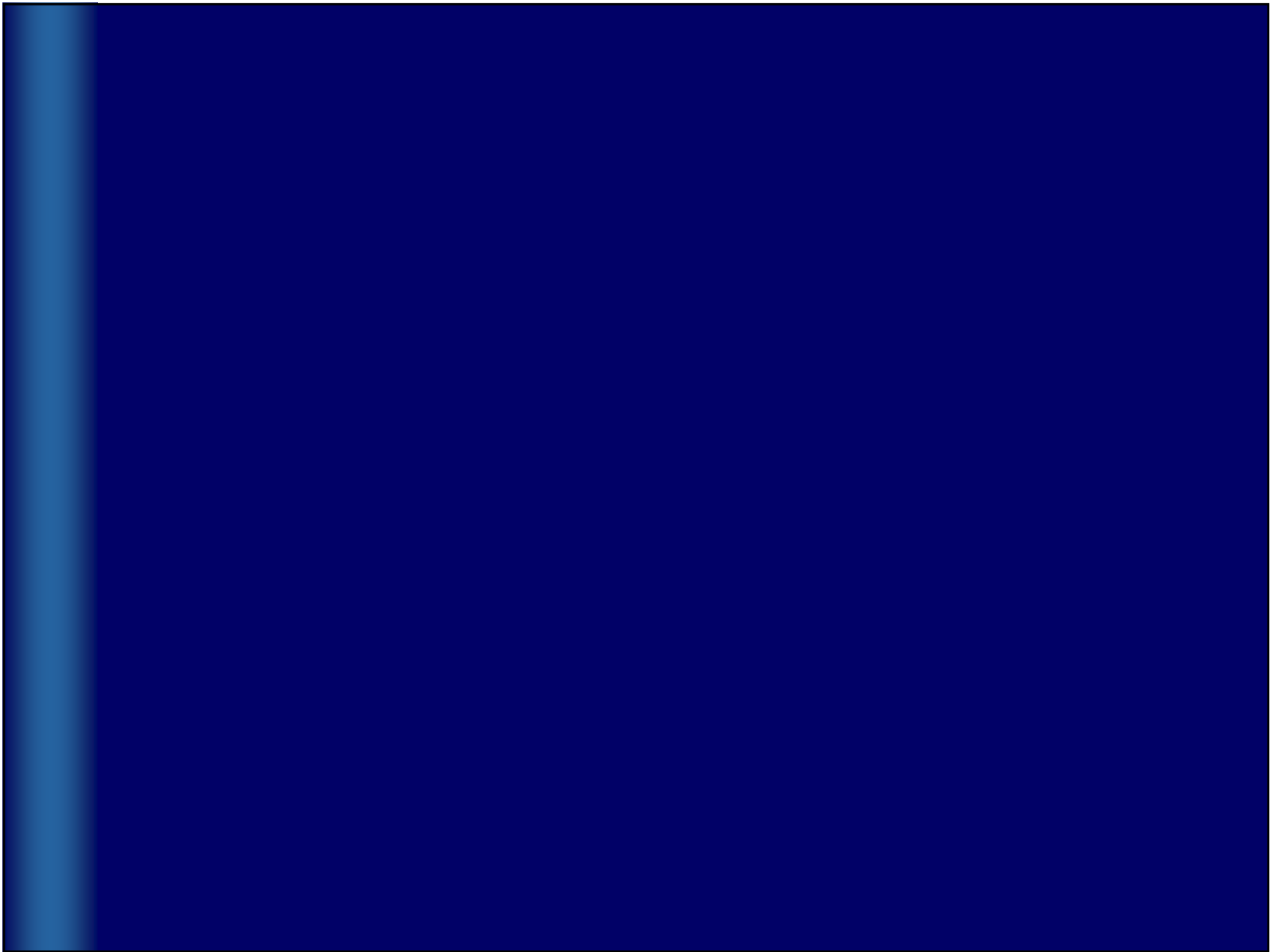
Effects:
New-component failures:
 failed function effects
 unexpected functions
Existing component failures
 new failed function effects
 new unexpected functions
 previously failed
 previously unexpected

Lifecycle incremental analysis

- Automated incremental can also be used for each new analysis level
 - Vague answers such as wire short to ground might cause fuse blow can be made specific - the fuse will or won't blow.
 - "incorrect" answers such as leakage current lighting a lamp will reflect the real situation not enough power to light lamp

Summary

- Electrical systems domain is component based and models are well understood.
- Separation of the qualitative from the component level reasoning allow the QR to be replaced
- Functional Labels work with numerical and qualitative reasoning



Larger systems: Improving the simulation efficiency

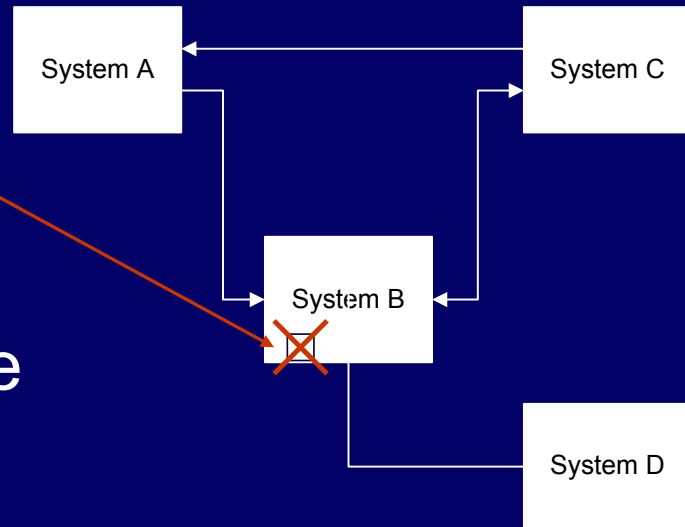
- Automated abstraction of subsystems into black box models'.
- The objective is to generate the simplest unambiguous 'black box' with only **system significant states**.
 - I.e. minimal set of states providing correct behaviour at system interface. Simulation gives the value of all component states, and variables, all resistive values, and electrical parameters.
- A simple mapping between component and system state does not usually exist

The problem:

System Interactions

- Usually only one system contains a fault.
- An **envisionment** describes a complete (sub) system behaviour in detail...

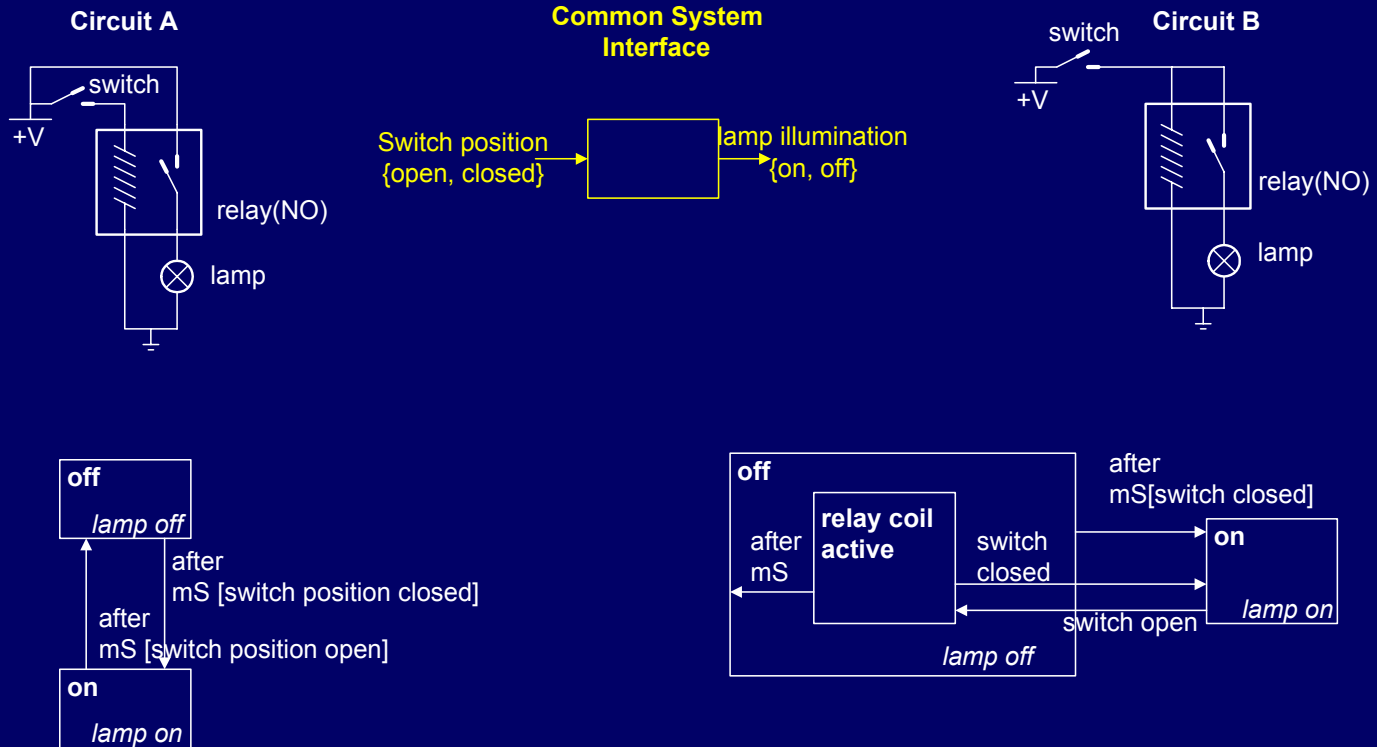
Logical interactions between systems on a shared bus



The Objective

- Generate an unambiguous abstraction of behaviour at the system I/O level.
- No simple mapping between component and system state:
 - System behaviour from electrical simulator
 - Partially used component behaviours.
 - Components form a system of operation!
 - Feedback.
 - Some component state directly/ indirectly reflects the system environment.

Illustration: system models for two simple circuits

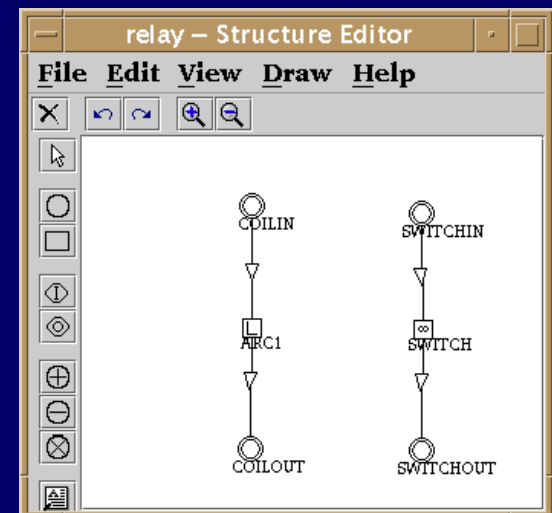
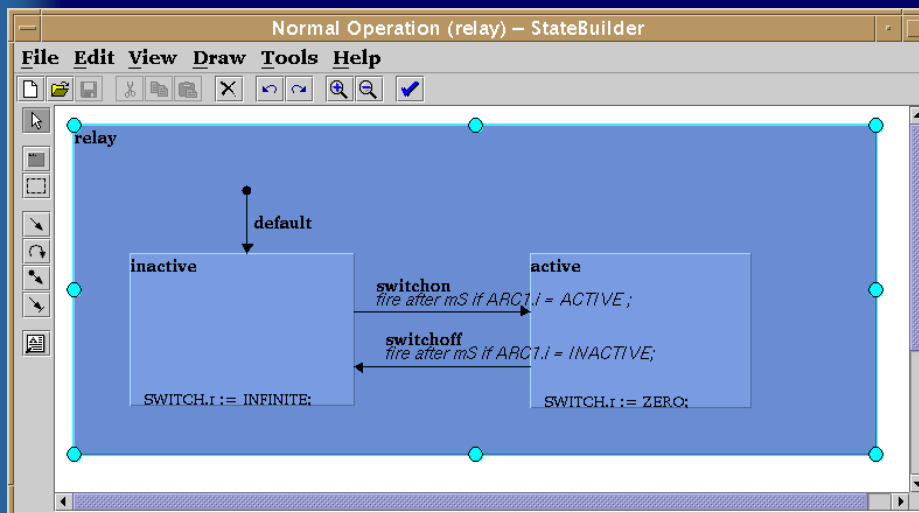
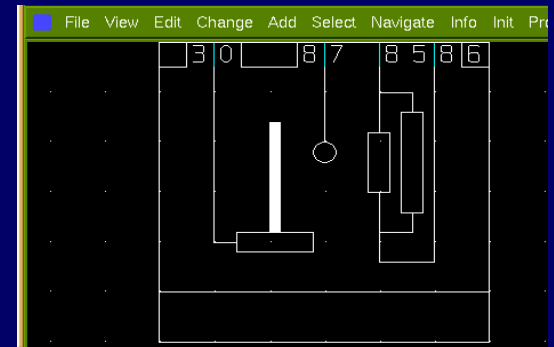


Example Component

Relay


- Contains transient internal state

Behaviour



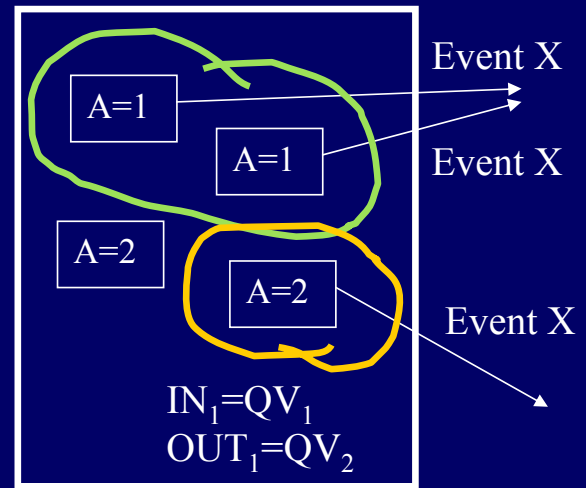
An Approach

(Informal description)

- Group **envisionment** states by system environment to produce *system states*
- Group also 'insignificant' states. 
- Resolve any ambiguous events:
 - Find significant distinguishing factors.
 - Subdivide system state groups.
- Repeat last step until an unambiguous model is produced.
- Identify concurrency in the model.

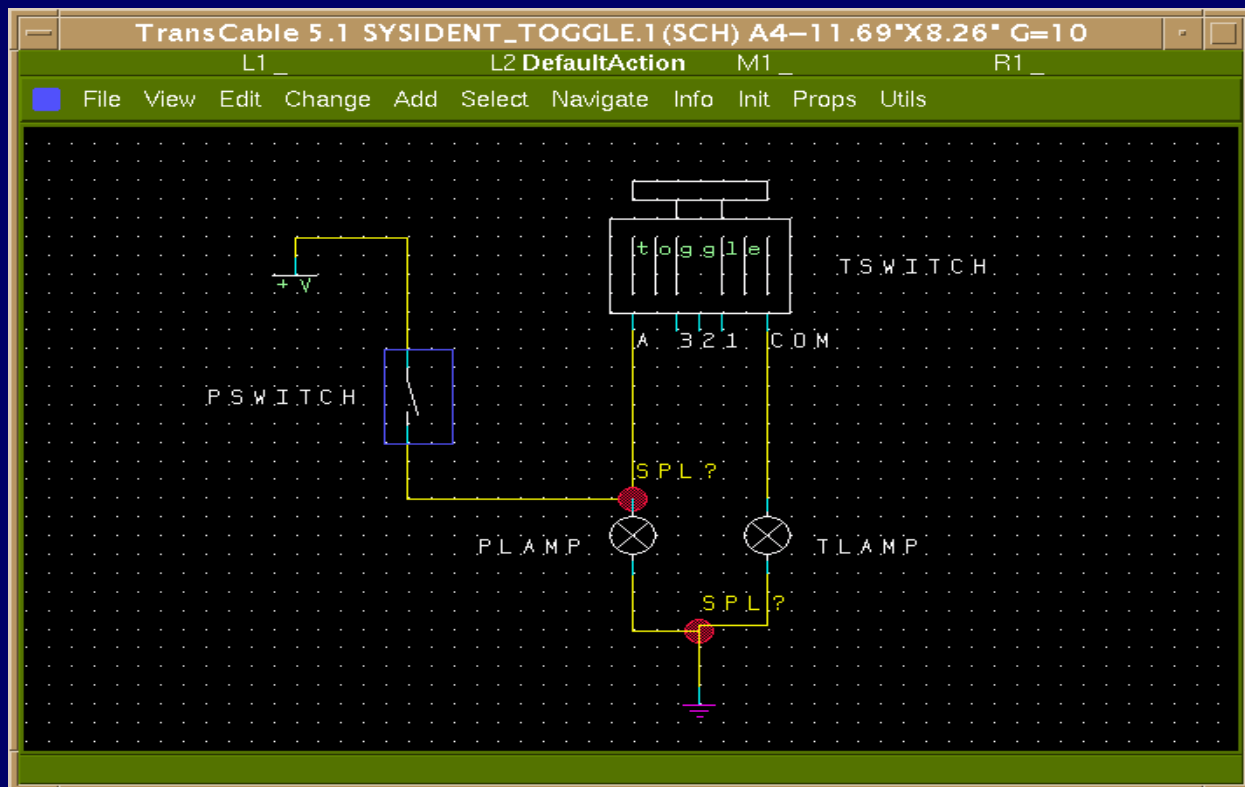
Finding state groups

- Possible distinguishing factors (component variables) are found for every ambiguous *pair* of system transitions.
- Factors selected using best first search based on distinguishing ability.
- Sole factors often exist.



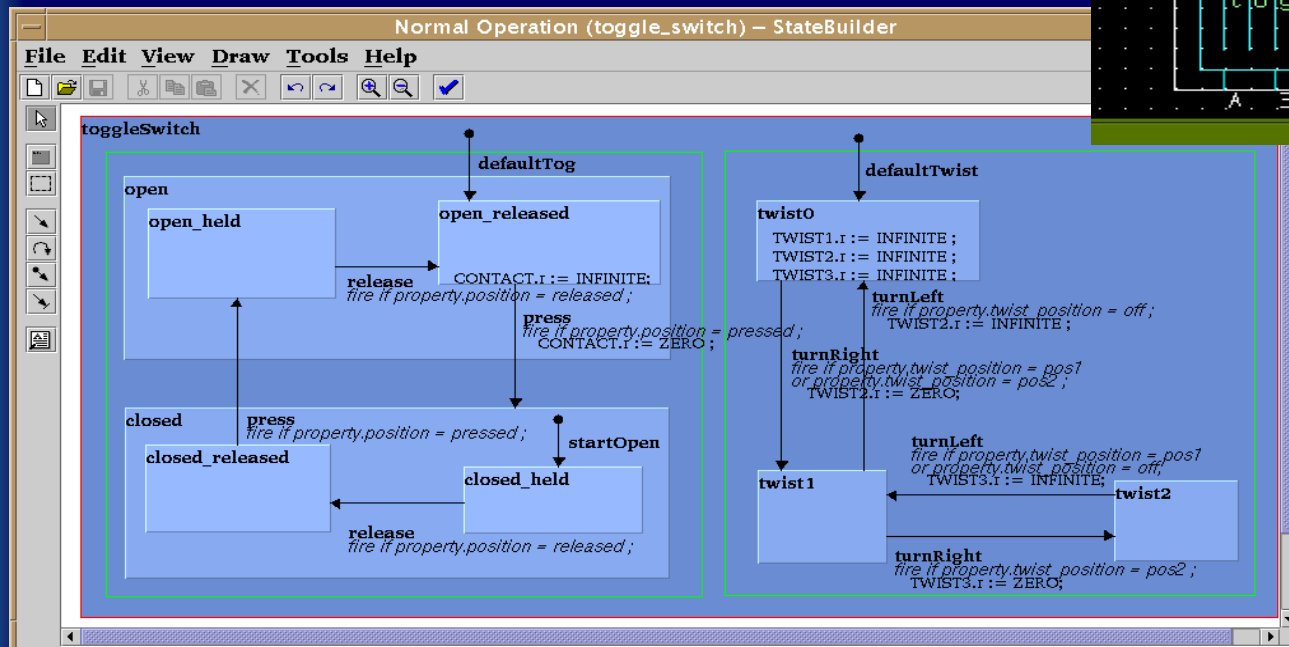
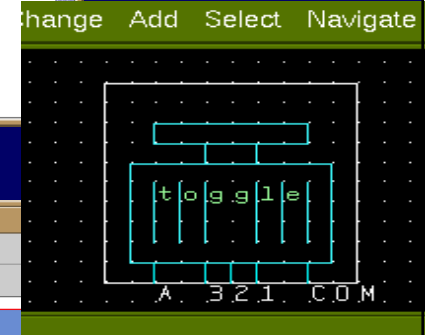
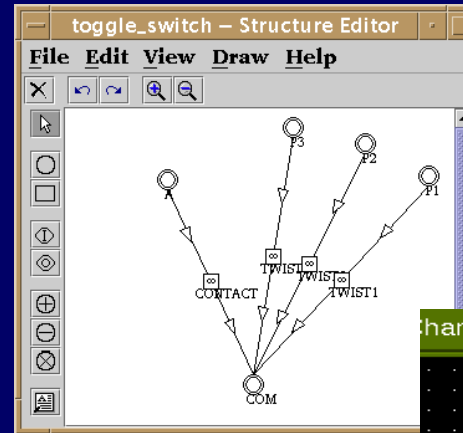
Simple example circuit

- Toggle switch circuit



Toggle switch

- Contains internal mechanical memory
- Mechanical inputs
 - Twist (3 position)
 - press, release

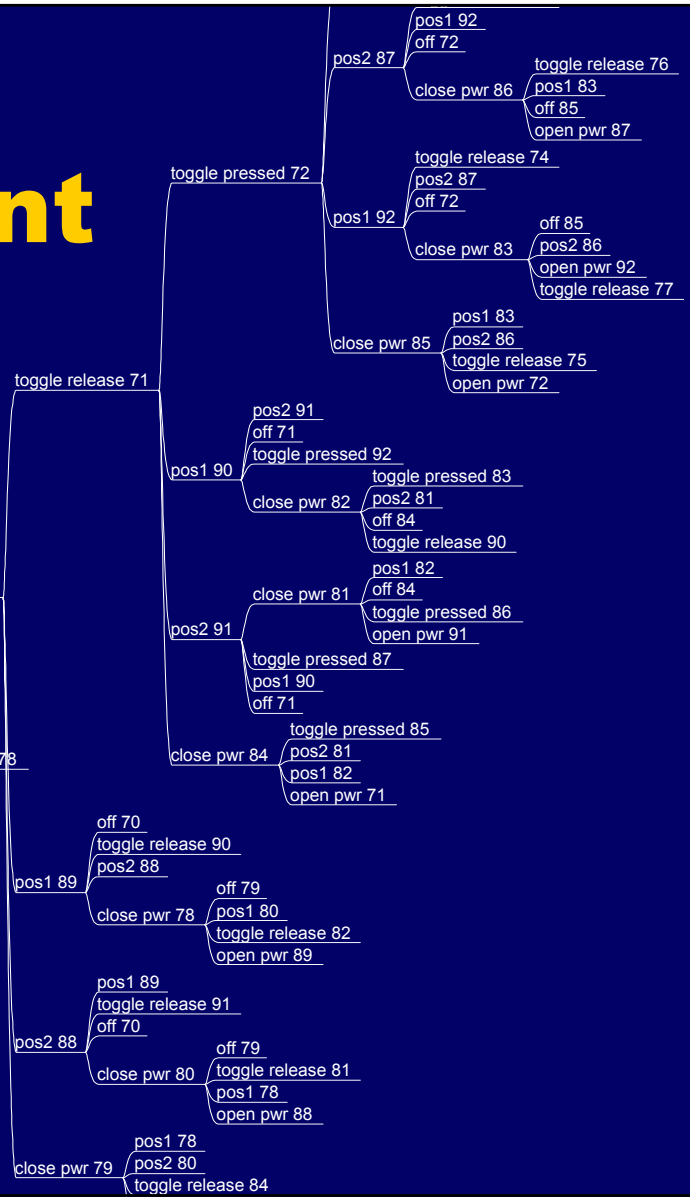
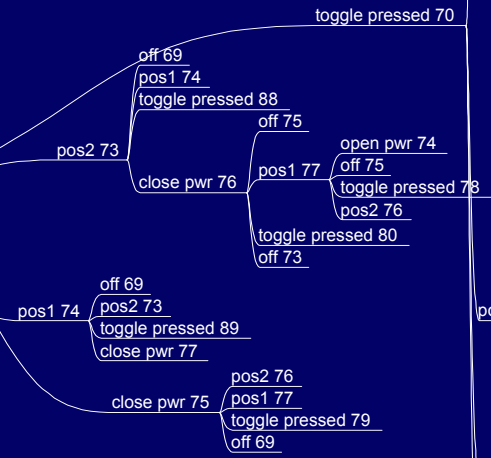


Envisionment

Events:

- Toggle pressed
- Toggle released
- Open pwr
- Close pwr
- Off
- Pos1
- Pos2

Toggle switch default 69



24 states total
96 transitions

External states

EXTERNAL STATES

ID	PLAMP. Light	PSWITCH. position	TLAMP. Light	TSWITCH. position	TSWITCH. twist_position
1	TRUE	closed	TRUE	pressed	off
2	TRUE	closed	TRUE	pressed	pos1
3	TRUE	closed	FALSE	released	pos1
4	TRUE	closed	FALSE	released	pos2
5	TRUE	closed	FALSE	released	off
6	FALSE	open	FALSE	released	pos1
7	FALSE	open	FALSE	released	pos2
8	FALSE	open	FALSE	pressed	off
9	FALSE	open	FALSE	released	off
10	FALSE	open	FALSE	pressed	pos1
11	FALSE	open	FALSE	pressed	pos2
12	TRUE	closed	TRUE	pressed	pos2
13	TRUE	closed	TRUE	released	off
14	TRUE	closed	TRUE	released	pos1
15	TRUE	closed	TRUE	released	pos2

The 3 grouped states

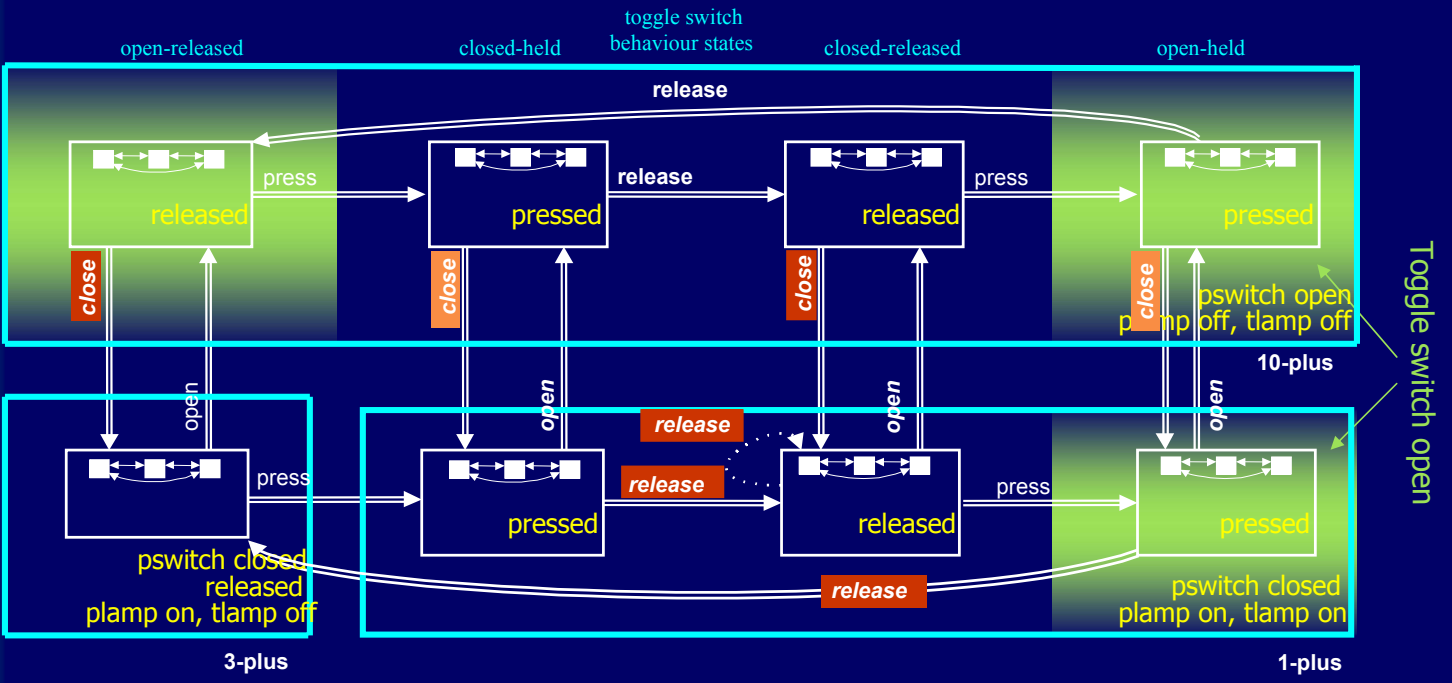
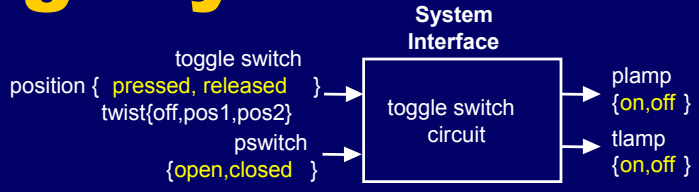
(those with environment determined transitions)

GROUPED STATES

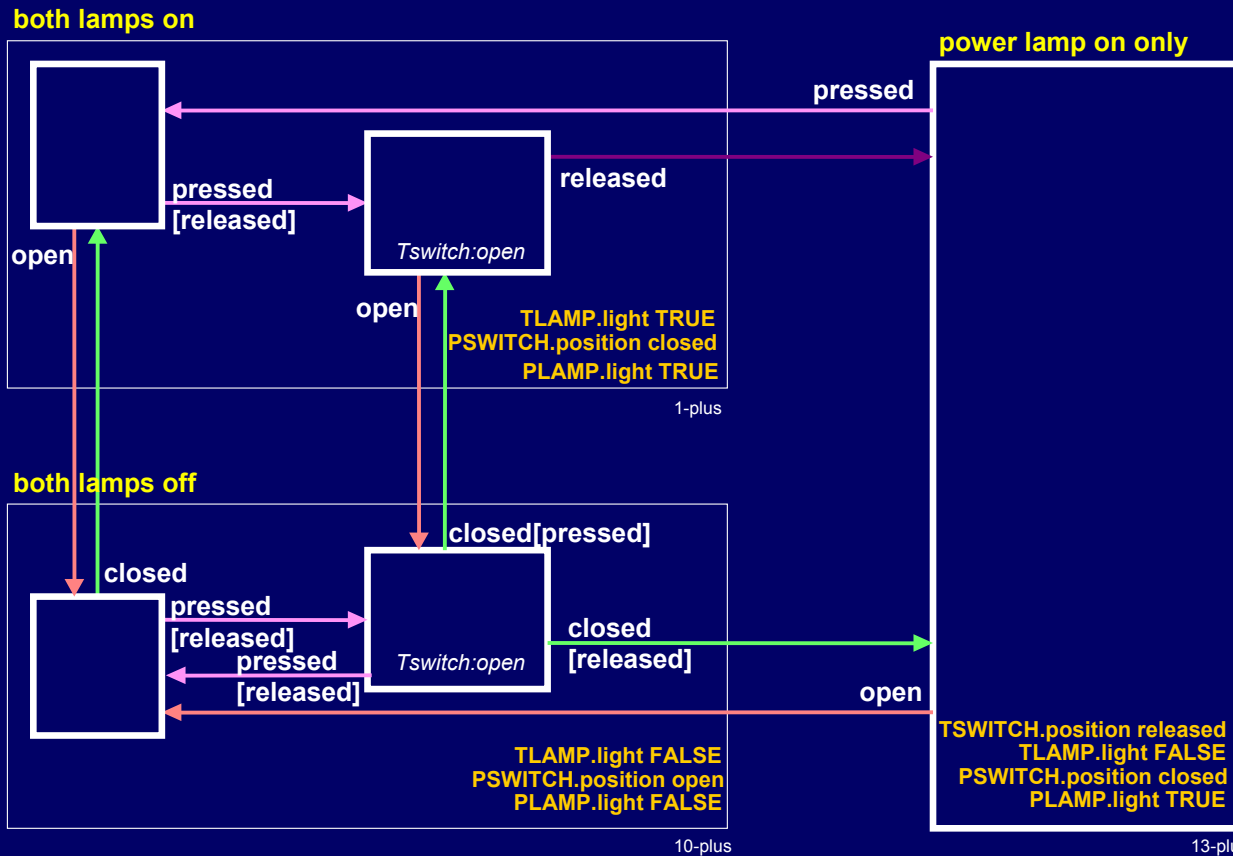
ID	PLAMP. Light	PSWITCH. position	TLAMP. Light	TSWITCH. position	TSWITCH. twist_position
1-plus	TRUE	closed	TRUE	pressed	off
1-plus				released	pos1
1-plus					pos2
10-plus	FALSE	open	FALSE	pressed	off
10-plus				released	pos1
10-plus					pos2
3-plus	TRUE	closed	FALSE	released	off
3-plus					pos2
3-plus					pos1

Ambiguity

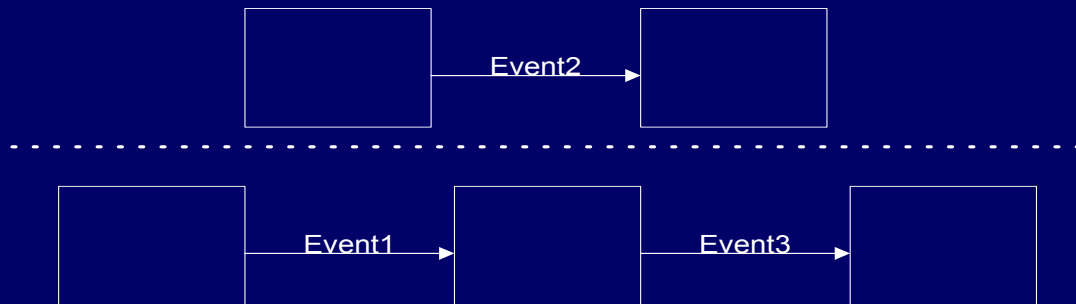
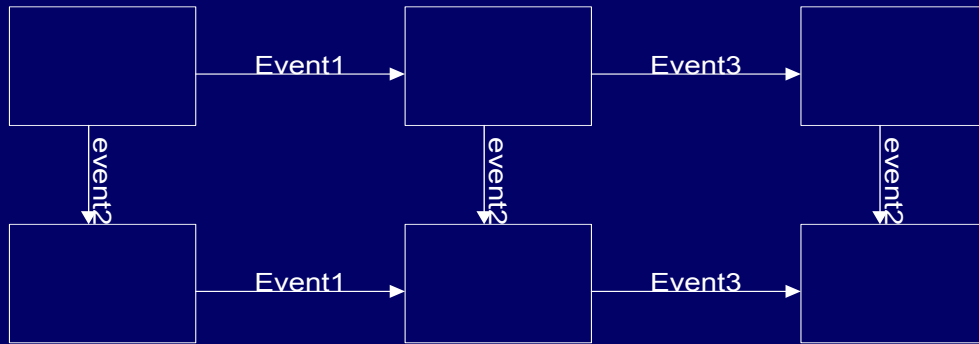
- Ambiguous events
- Implicit events



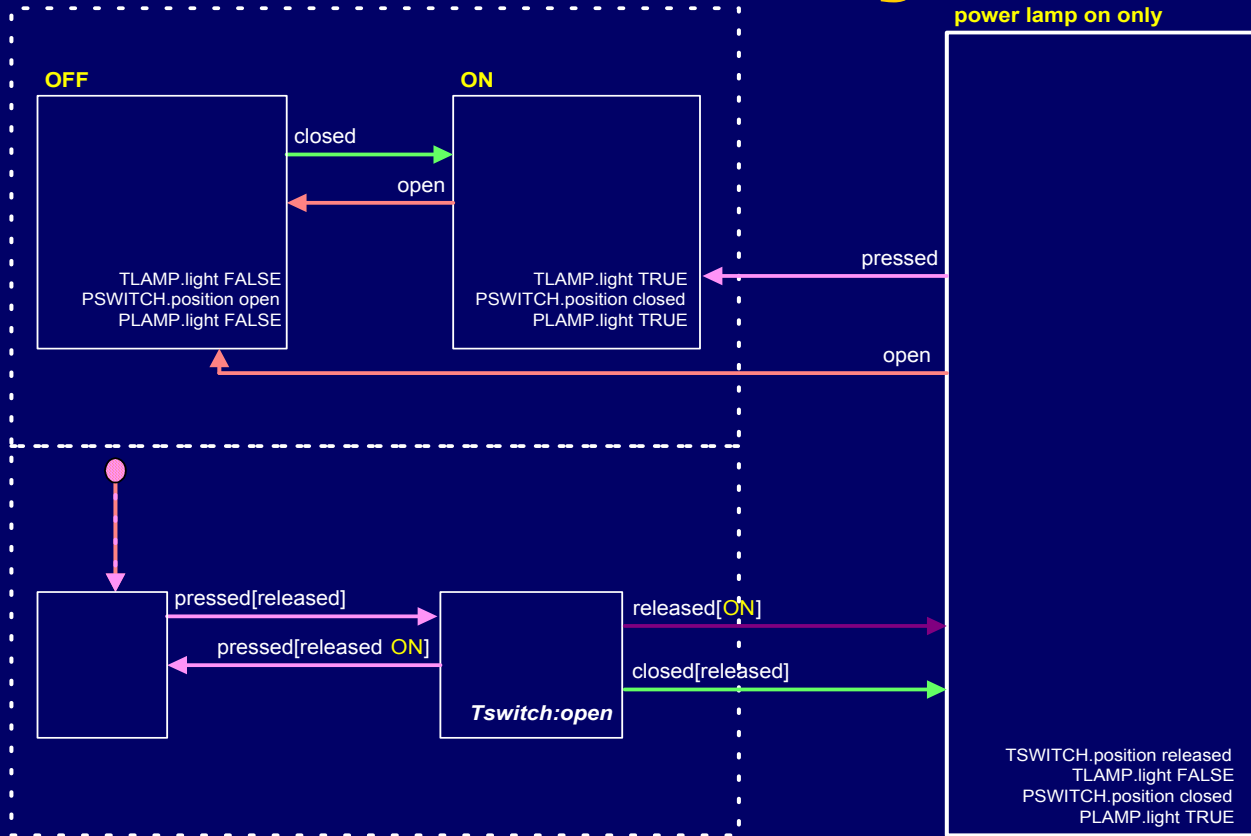
Final Model



Concurrency

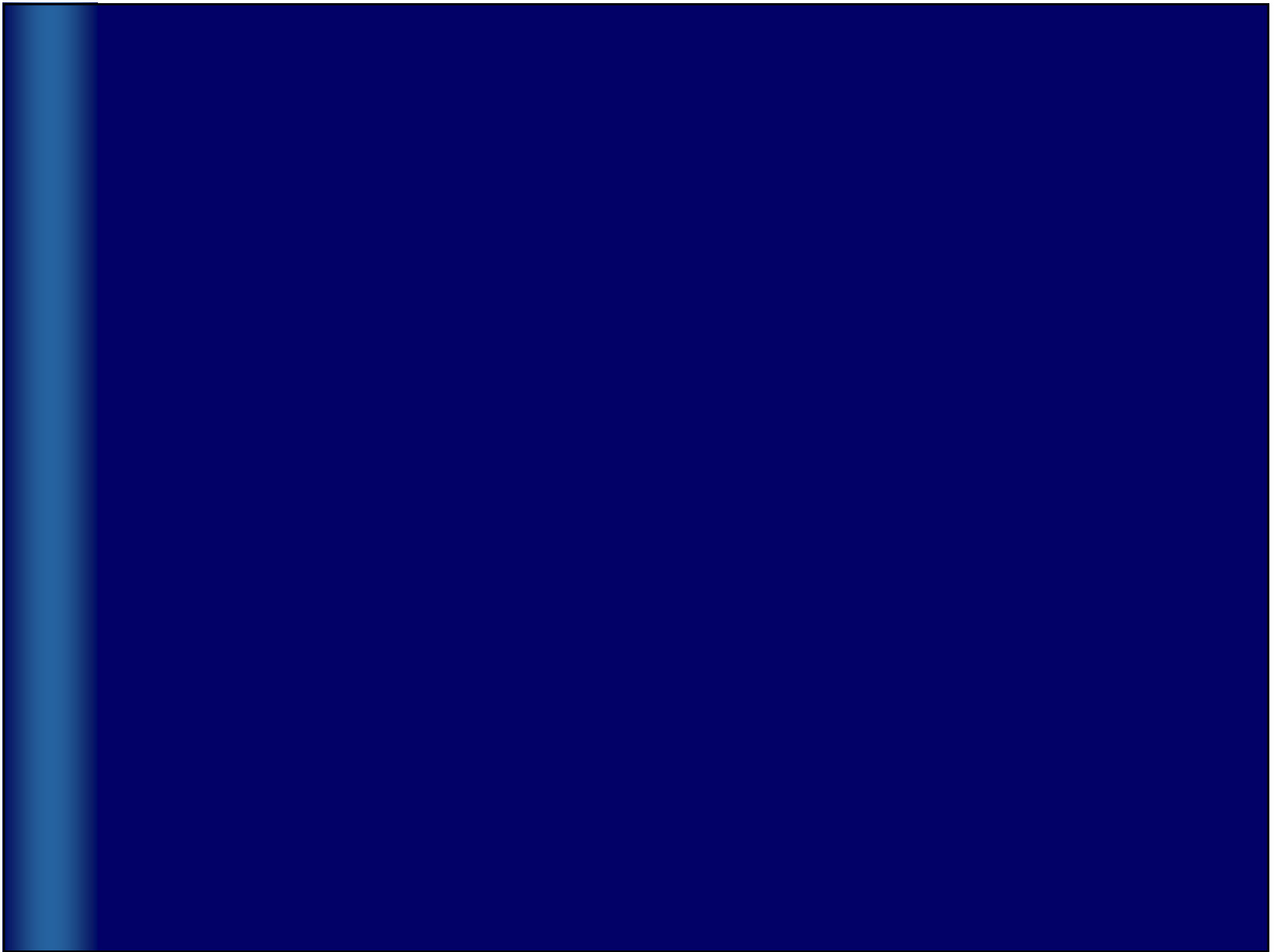


Concurrency



Automated Abstraction

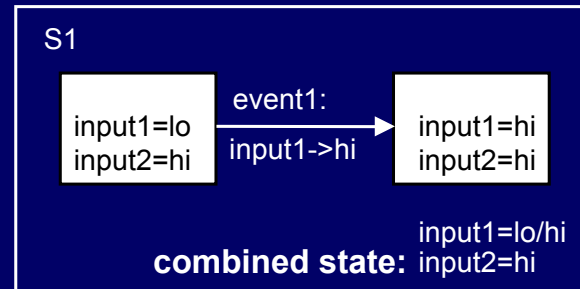
- Good for systems with moderate amounts of internal state (envisionment must be tractable).
- Simulation efficiency gains for SCA and multi system FMEA.
- Possibly highlights system behaviour artefacts to engineers (with visualisation)



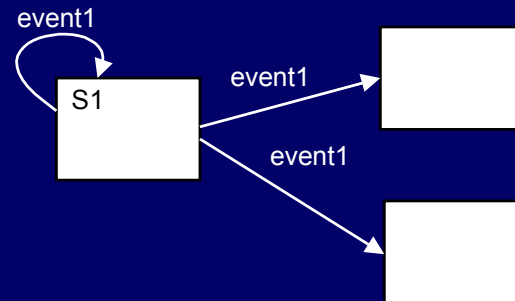
Combining States

- Insignificant state changes

'insignificant' states



- Ambiguous state changes



Selecting distinguishing factors

AMBIGUOUS TRANSITION TABLE

Possible factors are:

- 1 TSWITCH.open_held
- 2 TSWITCH.open
- 3 TSWITCH.closed_held
- 4 TSWITCH.closed
- 5 TSWITCH.open_released
- 6 TSWITCH.CONTACT.r:ZERO
- 7 TSWITCH.closed_released

AMBIGUOUS TRANSITIONS	1234567
{72_to_69 70_to_71}	XXXX
{72_to_85 70_to_79}	XXXX
{70_to_79 72_to_85}	XXXX
{88_to_80 87_to_86}	XXXX
{87_to_73 88_to_91}	XXXX
{87_to_86 88_to_80}	XXXX
{92_to_74 89_to_90}	XXXX
{92_to_83 89_to_78}	XXXX
{89_to_78 92_to_83}	XXXX
{74_to_77 90_to_82}	X XXXX
{90_to_82 74_to_77}	X XXXX
{71_to_84 69_to_75}	X XXXX
{69_to_75 71_to_84}	X XXXX
{73_to_76 91_to_81}	X XXXX
{91_to_81 73_to_76}	X XXXX

