freescale™

Best Practices in Verification Planning

Benjamin Ehlers

Presented by

Paul Carzola – Senior Architect

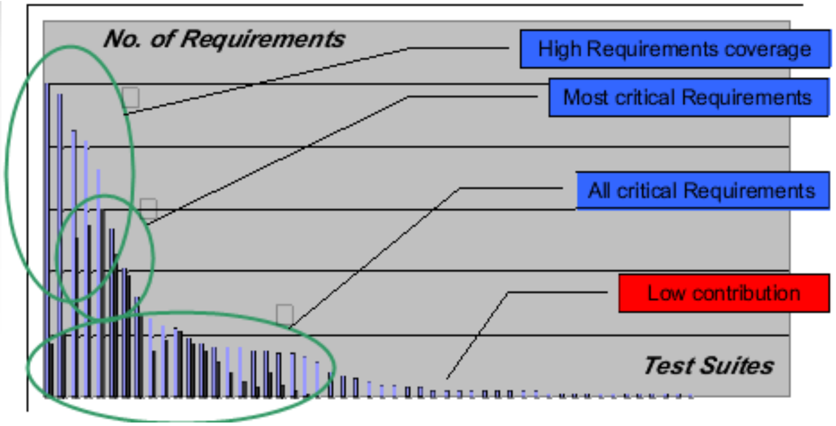
Cadence Design Systems

Feb 2013

Discussion Topics:

- Verification planning problems
- Planning challenges to address
- Notion of Executable Verification Plan
- Recommended verification solution
- Example verification planning flow
- Value Analysis

About this Paper



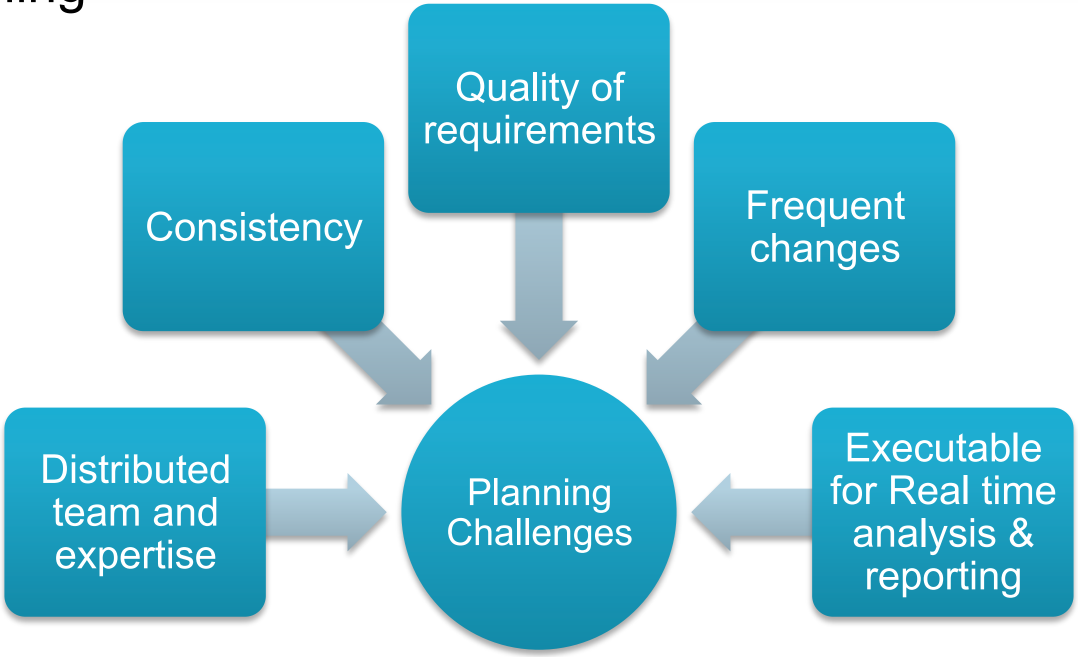
- **Failure to Plan Means Planning to Fail**
- Verification planning, the most crucial step
  - Improves quality
  - Improves efficiency
- Methodology based on actual experiences on an SoC
- Includes strategy, planning, changes and closure
- Begins with culture of Metric Driven Verification
- Follows a well organized process
- Enhanced with the executable verification plan

Verification Planning Problems:

- Verification planning as an afterthought
- Lack of organized or standardized flow
- Absence of verification methodologies
- Lack of process for changing and evolving requirements
- Fear of verification planning costs
- Limitations in Technology

Planning Challenges to Address:

- Large scale distributed planning approach:
  - large team of engineers
  - variety of individual styles, priorities, experience
- Enable and maintain Consistency/Coherency
- Quality of design documentation and design requirements
- Frequent design changes
- Needs to support concept of Executable Verification Plan


Quality of requirements

Consistency

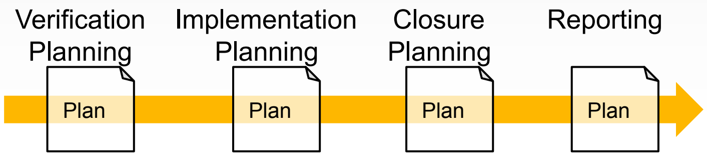
Frequent changes

Distributed team and expertise

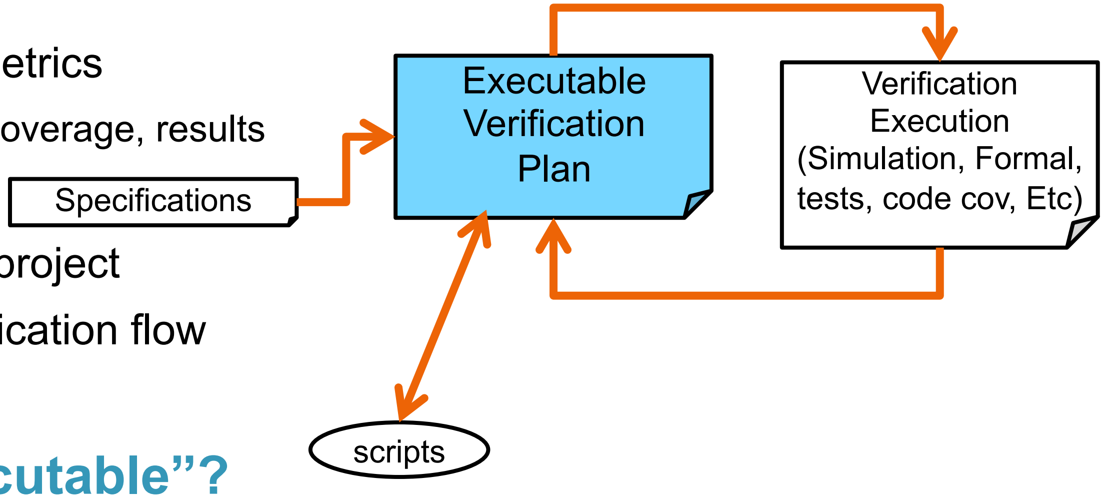
Planning Challenges

Executable for Real time analysis & reporting

Codified and Executable Verification Plan

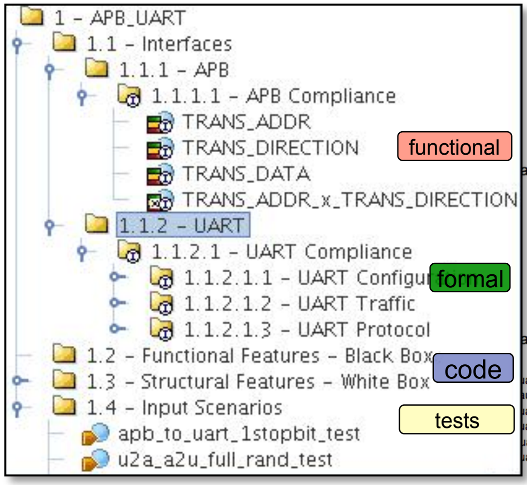


- Single point of source across entire verification flow
- Defines the scope of verification
- Describes all key design features
- Supports link to the verification metrics
  - testcases, checkers, monitors, coverage, results
- Link to specifications
- Planning for the total verification project
  - complete and coherent verification flow
- Flexible for scripting
- **What makes a plan "executable"?**
  - **The notion to define and refine your verification strategy**
  - **The notion of collecting all run time results to make the plan dynamic and alive**

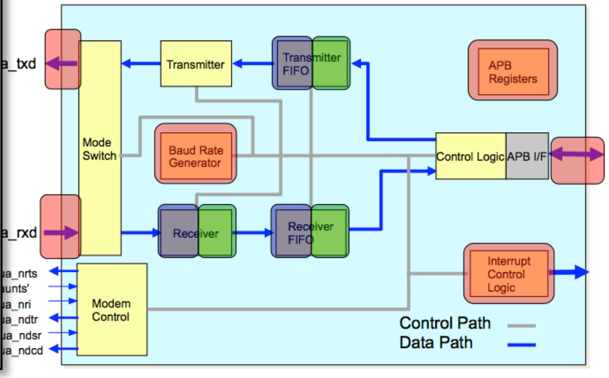

Executable Verification Plan
Verification Execution (Simulation, Formal, tests, code cov, Etc)
Specifications
scripts

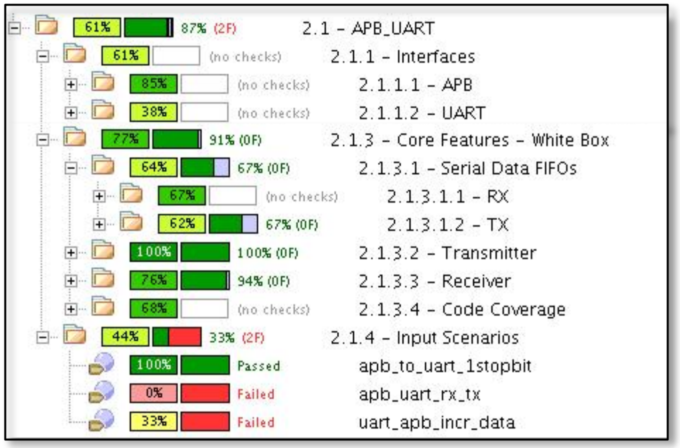
How Executable Plan Recovers Cost of Up-Front Planning

- Testbench design begins in planning
  - Planned coverage, checks, tests cases provide guidance on testbench code and tests cases that need to be written
- The final verification results and completions criteria are built into the executable plan



```
1 – APB_UART
  1.1 – Interfaces
    1.1.1 – APB
      1.1.1.1 – APB Compliance
        TRANS_ADDR
        TRANS_DIRECTION
        TRANS_DATA
        TRANS_ADDR_x_TRANS_DIRECTION
    1.1.2 – UART
      1.1.2.1 – UART Compliance
        1.1.2.1.1 – UART Configu
        1.1.2.1.2 – UART Traffic
        1.1.2.1.3 – UART Protocol
  1.2 – Functional Features – Black Box
  1.3 – Structural Features – White Box
  1.4 – Input Scenarios
    apb_to_uart_1stopbit_test
    u2a_a2u_full_rand_test
```





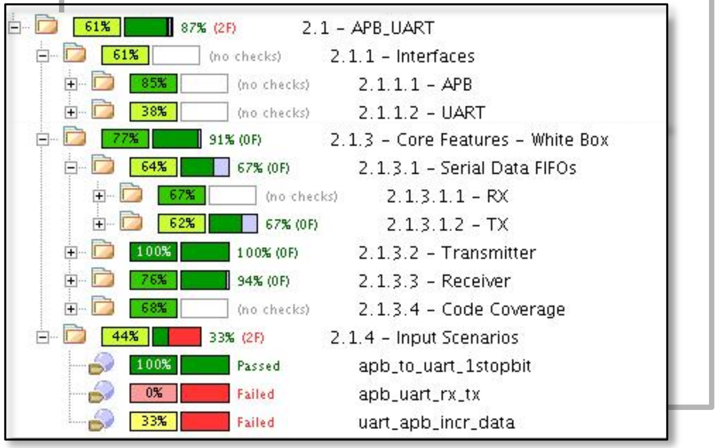
```
2.1 – APB_UART                          61%  87% (2F)
  2.1.1 – Interfaces                    61%  (no checks)
    2.1.1.1 – APB                       85%  (no checks)
    2.1.1.2 – UART                      38%  (no checks)
  2.1.3 – Core Features – White Box     77%  91% (0F)
    2.1.3.1 – Serial Data FIFOs         64%  67% (0F)
      2.1.3.1.1 – RX                    67%  (no checks)
      2.1.3.1.2 – TX                    62%  67% (0F)
    2.1.3.2 – Transmitter               100% 100% (0F)
    2.1.3.3 – Receiver                  76%  94% (0F)
    2.1.3.4 – Code Coverage             68%  (no checks)
  2.1.4 – Input Scenarios               44%  33% (2F)
    apb_to_uart_1stopbit                100% Passed
    apb_uart_rx_tx                      0%   Failed
    uart_apb_incr_data                  33%  Failed
```

Executable Plan: Linking all the key elements

Requirements Objectives

Test Scenarios

Heterogeneous Verification Tools (ie Formal)



```
⊟ 61%  87% (2F)          2.1 - APB_UART
  ⊟ 61%  (no checks)       2.1.1 - Interfaces
    ⊞ 85%  (no checks)       2.1.1.1 - APB
    ⊞ 38%  (no checks)       2.1.1.2 - UART
  ⊟ 77%  91% (0F)          2.1.3 - Core Features - White Box
    ⊟ 64%  67% (0F)          2.1.3.1 - Serial Data FIFOs
      ⊞ 67%  (no checks)       2.1.3.1.1 - RX
      ⊞ 62%  67% (0F)          2.1.3.1.2 - TX
    ⊞ 100%  100% (0F)        2.1.3.2 - Transmitter
    ⊞ 76%  94% (0F)          2.1.3.3 - Receiver
    ⊞ 68%  (no checks)       2.1.3.4 - Code Coverage
  ⊟ 44%  33% (2F)          2.1.4 - Input Scenarios
       100%  Passed          apb_to_uart_1stopbit
       0%    Failed          apb_uart_rx_tx
       33%   Failed          uart_apb_incr_data
```

Distributed Team and Plans

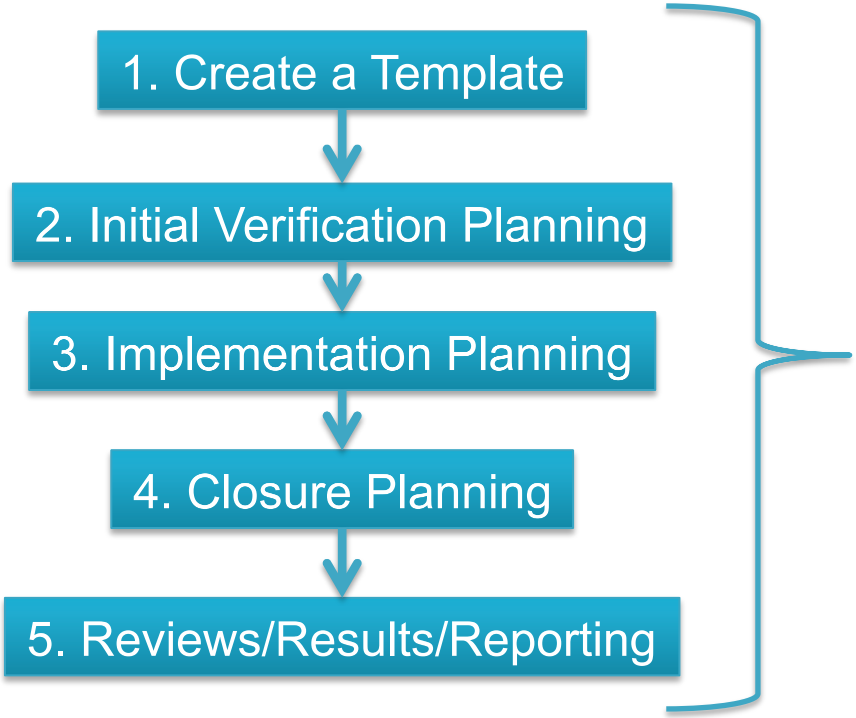
Specification

Error Conditions After Execution

The verification plan becomes the anchor to connect teams and technologies together

Best Practice Planning Methodology

- Successfully used at Freescale on a large SoC project
- With a widely distributed verification team

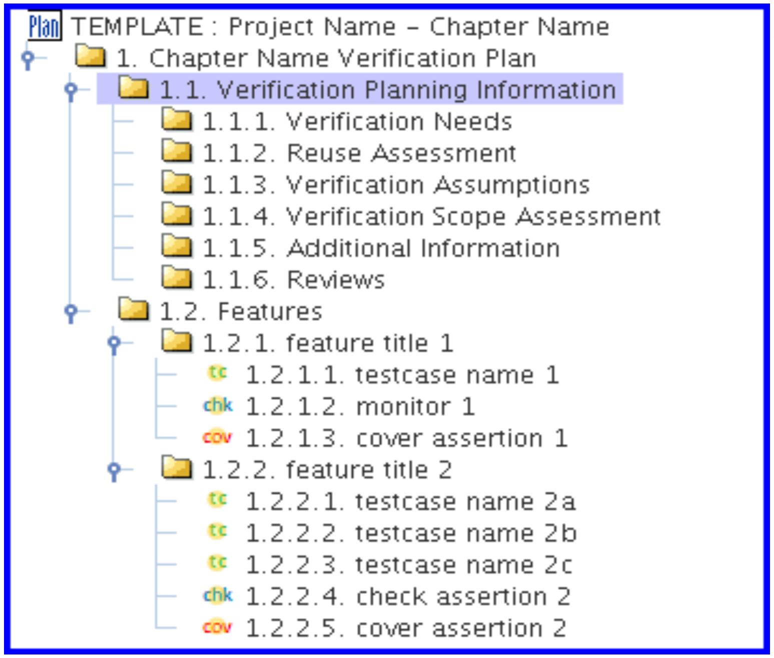


1. Create a Template
2. Initial Verification Planning
3. Implementation Planning
4. Closure Planning
5. Reviews/Results/Reporting

All very important aspects to the Planning Methodology

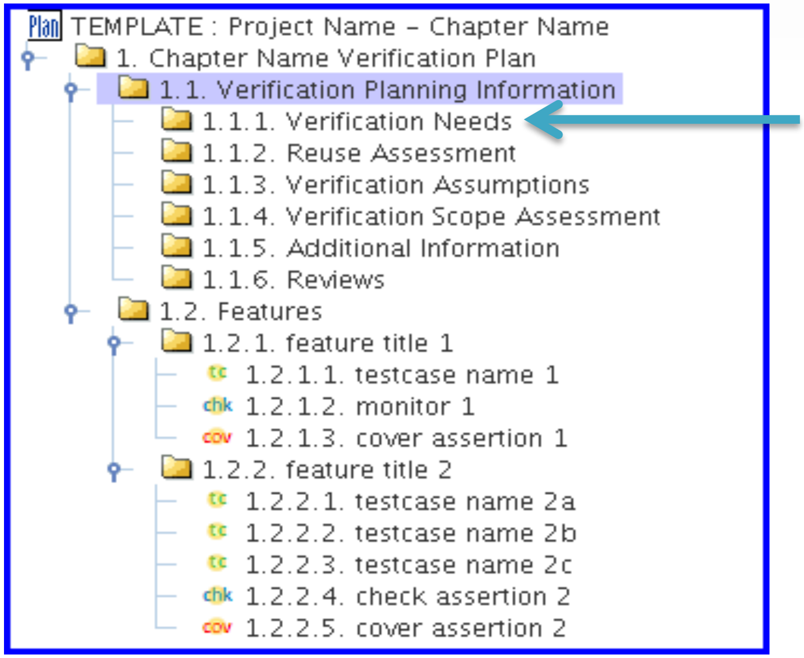
1. Create a Template:

- Enable and help ensure Consistency and Coherency from the beginning
- Provides a prescriptive approach to follow
- May seem trivial, unnecessary...
  - VITAL STEP
- Make sure template supports merging/importing multiple plans
  - distributed verification approach
- Include detailed explanation descriptions



- Plan TEMPLATE : Project Name – Chapter Name
  - 1. Chapter Name Verification Plan
    - 1.1. Verification Planning Information
      - 1.1.1. Verification Needs
      - 1.1.2. Reuse Assessment
      - 1.1.3. Verification Assumptions
      - 1.1.4. Verification Scope Assessment
      - 1.1.5. Additional Information
      - 1.1.6. Reviews
    - 1.2. Features
      - 1.2.1. feature title 1
        - tc 1.2.1.1. testcase name 1
        - chk 1.2.1.2. monitor 1
        - cov 1.2.1.3. cover assertion 1
      - 1.2.2. feature title 2
        - tc 1.2.2.1. testcase name 2a
        - tc 1.2.2.2. testcase name 2b
        - tc 1.2.2.3. testcase name 2c
        - chk 1.2.2.4. check assertion 2
        - cov 1.2.2.5. cover assertion 2

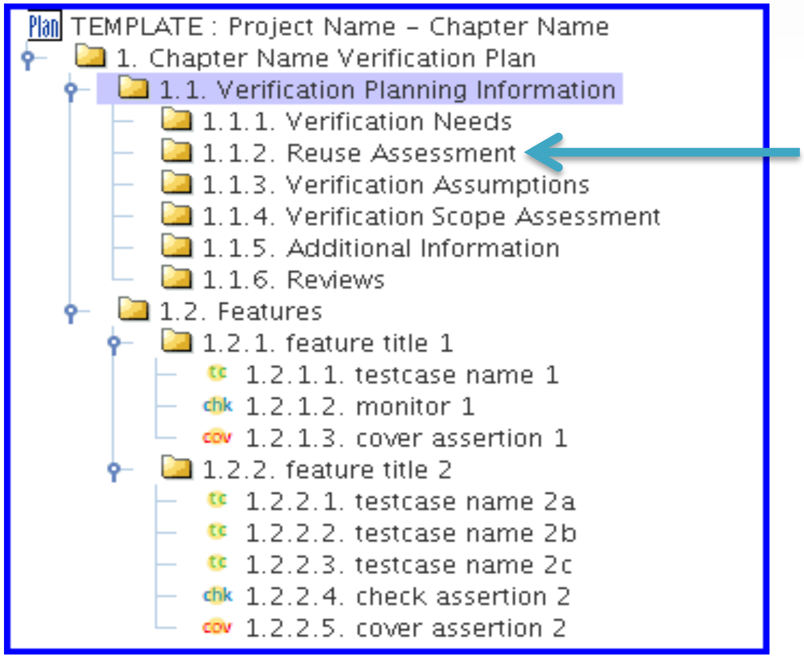
2. Initial Verification Planning: Verification Needs



- Plan TEMPLATE : Project Name - Chapter Name
  - 1. Chapter Name Verification Plan
    - 1.1. Verification Planning Information
      - 1.1.1. Verification Needs
      - 1.1.2. Reuse Assessment
      - 1.1.3. Verification Assumptions
      - 1.1.4. Verification Scope Assessment
      - 1.1.5. Additional Information
      - 1.1.6. Reviews
    - 1.2. Features
      - 1.2.1. feature title 1
        - tc 1.2.1.1. testcase name 1
        - chk 1.2.1.2. monitor 1
        - cov 1.2.1.3. cover assertion 1
      - 1.2.2. feature title 2
        - tc 1.2.2.1. testcase name 2a
        - tc 1.2.2.2. testcase name 2b
        - tc 1.2.2.3. testcase name 2c
        - chk 1.2.2.4. check assertion 2
        - cov 1.2.2.5. cover assertion 2

- Design documents
- Tools
- Models behavior or functional
- Design blocks that cannot be modeled
- Protocols used
- Testbench components
- Purchase vs Make components

2. Initial Verification Planning: Reuse Assessment

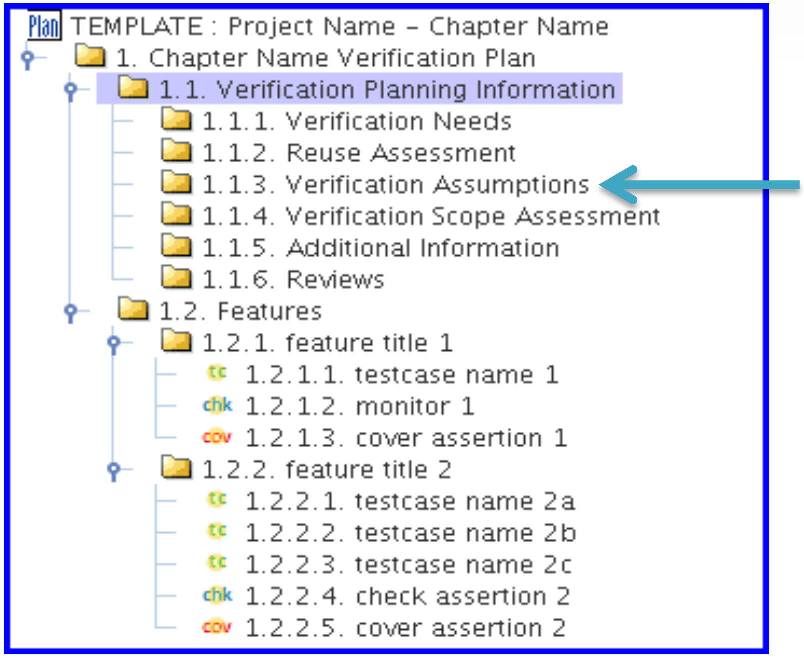


```
Plan TEMPLATE : Project Name - Chapter Name
  1. Chapter Name Verification Plan
    1.1. Verification Planning Information
      1.1.1. Verification Needs
      1.1.2. Reuse Assessment
      1.1.3. Verification Assumptions
      1.1.4. Verification Scope Assessment
      1.1.5. Additional Information
      1.1.6. Reviews
    1.2. Features
      1.2.1. feature title 1
        tc  1.2.1.1. testcase name 1
        chk 1.2.1.2. monitor 1
        cov 1.2.1.3. cover assertion 1
      1.2.2. feature title 2
        tc  1.2.2.1. testcase name 2a
        tc  1.2.2.2. testcase name 2b
        tc  1.2.2.3. testcase name 2c
        chk 1.2.2.4. check assertion 2
        cov 1.2.2.5. cover assertion 2
```

- Determine how much can be reused from previous projects
  - Test benches, checks, tests, VIP, etc…
  - Don’t under scope
- Team reuse and dynamics
  - What teams work well together vs new teams.
- Determine what needs to be developed
- Review existing documentation
- What designers are available for questions

freescale™

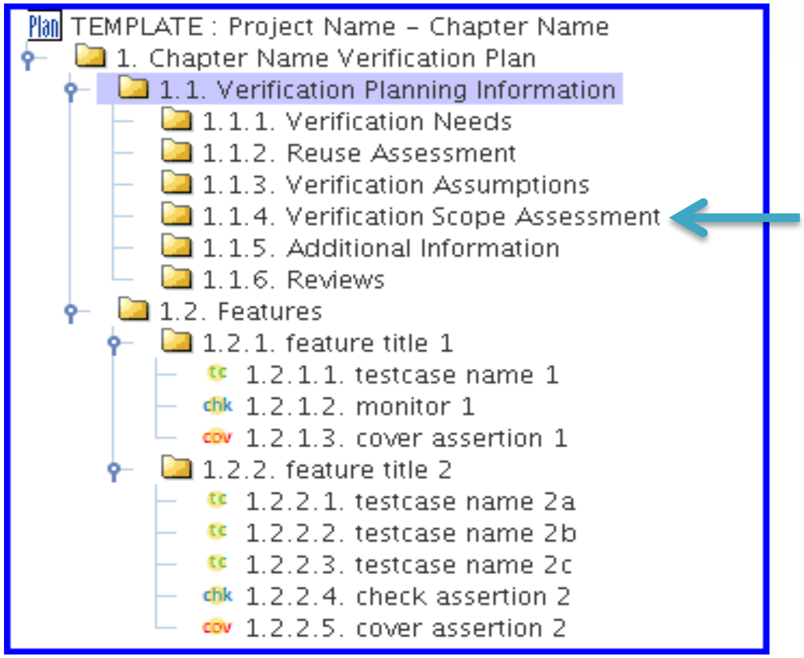
2. Initial Verification Planning: Verification Assumptions



- Plan TEMPLATE : Project Name – Chapter Name
  - 1. Chapter Name Verification Plan
    - 1.1. Verification Planning Information
      - 1.1.1. Verification Needs
      - 1.1.2. Reuse Assessment
      - 1.1.3. Verification Assumptions
      - 1.1.4. Verification Scope Assessment
      - 1.1.5. Additional Information
      - 1.1.6. Reviews
    - 1.2. Features
      - 1.2.1. feature title 1
        - tc 1.2.1.1. testcase name 1
        - chk 1.2.1.2. monitor 1
        - cov 1.2.1.3. cover assertion 1
      - 1.2.2. feature title 2
        - tc 1.2.2.1. testcase name 2a
        - tc 1.2.2.2. testcase name 2b
        - tc 1.2.2.3. testcase name 2c
        - chk 1.2.2.4. check assertion 2
        - cov 1.2.2.5. cover assertion 2

- Assumptions are all too often left unspoken leading to design features being misinterpreted
- Document and review any assumptions
- Review with designer early
- Have face-to-face meetings
- Document all findings
- E.g.
  - Feature X cannot be verified in digital, needs Analog Mixed Signal
  - System level signal needs to be modeled during SoC verification
- Checklist can be useful here
  - Some captured during post mortem time
  - Use action item system
- **Make this a committed contract so there are no surprises**

2. Initial Verification Planning: Verification Scope Assessment



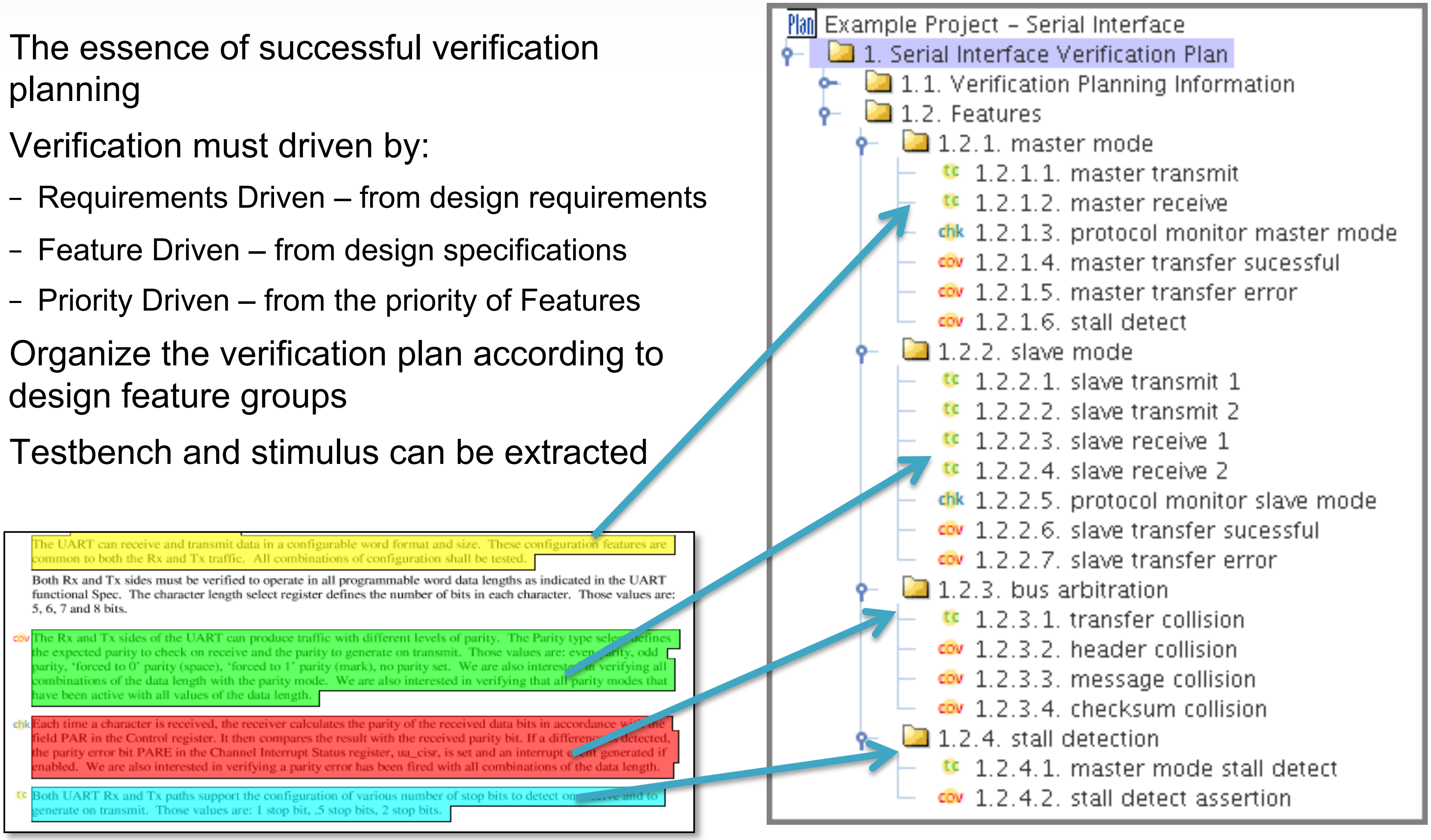
- Plan TEMPLATE : Project Name – Chapter Name
  - 1. Chapter Name Verification Plan
    - 1.1. Verification Planning Information
      - 1.1.1. Verification Needs
      - 1.1.2. Reuse Assessment
      - 1.1.3. Verification Assumptions
      - 1.1.4. Verification Scope Assessment
      - 1.1.5. Additional Information
      - 1.1.6. Reviews
    - 1.2. Features
      - 1.2.1. feature title 1
        - tc 1.2.1.1. testcase name 1
        - chk 1.2.1.2. monitor 1
        - cov 1.2.1.3. cover assertion 1
      - 1.2.2. feature title 2
        - tc 1.2.2.1. testcase name 2a
        - tc 1.2.2.2. testcase name 2b
        - tc 1.2.2.3. testcase name 2c
        - chk 1.2.2.4. check assertion 2
        - cov 1.2.2.5. cover assertion 2

- Consider all the available capabilities available to you: directed test, constrained random, formal, checks, coverage properties, code coverage, etc…
- Methodology doc for how to choose.
- Iterative process
- Take inputs from reuse assessment
- Understand the cost to create new test benches
- Determine the skill set of the verification engineers

freescale™

3. Implementation Planning: Plan Creation

- The essence of successful verification planning
- Verification must driven by:
  - Requirements Driven – from design requirements
  - Feature Driven – from design specifications
  - Priority Driven – from the priority of Features
- Organize the verification plan according to design feature groups
- Testbench and stimulus can be extracted



The UART can receive and transmit data in a configurable word format and size. These configuration features are common to both the Rx and Tx traffic. All combinations of configuration shall be tested.

Both Rx and Tx sides must be verified to operate in all programmable word data lengths as indicated in the UART functional Spec. The character length select register defines the number of bits in each character. Those values are: 5, 6, 7 and 8 bits.

cov The Rx and Tx sides of the UART can produce traffic with different levels of parity. The Parity type select defines the expected parity to check on receive and the parity to generate on transmit. Those values are: even parity, odd parity, 'forced to 0' parity (space), 'forced to 1' parity (mark), no parity set. We are also interested in verifying all combinations of the data length with the parity mode. We are also interested in verifying that all parity modes that have been active with all values of the data length.

chk Each time a character is received, the receiver calculates the parity of the received data bits in accordance with the field PAR in the Control register. It then compares the result with the received parity bit. If a difference is detected, the parity error bit PARE in the Channel Interrupt Status register, ua_cisr, is set and an interrupt event generated if enabled. We are also interested in verifying a parity error has been fired with all combinations of the data length.

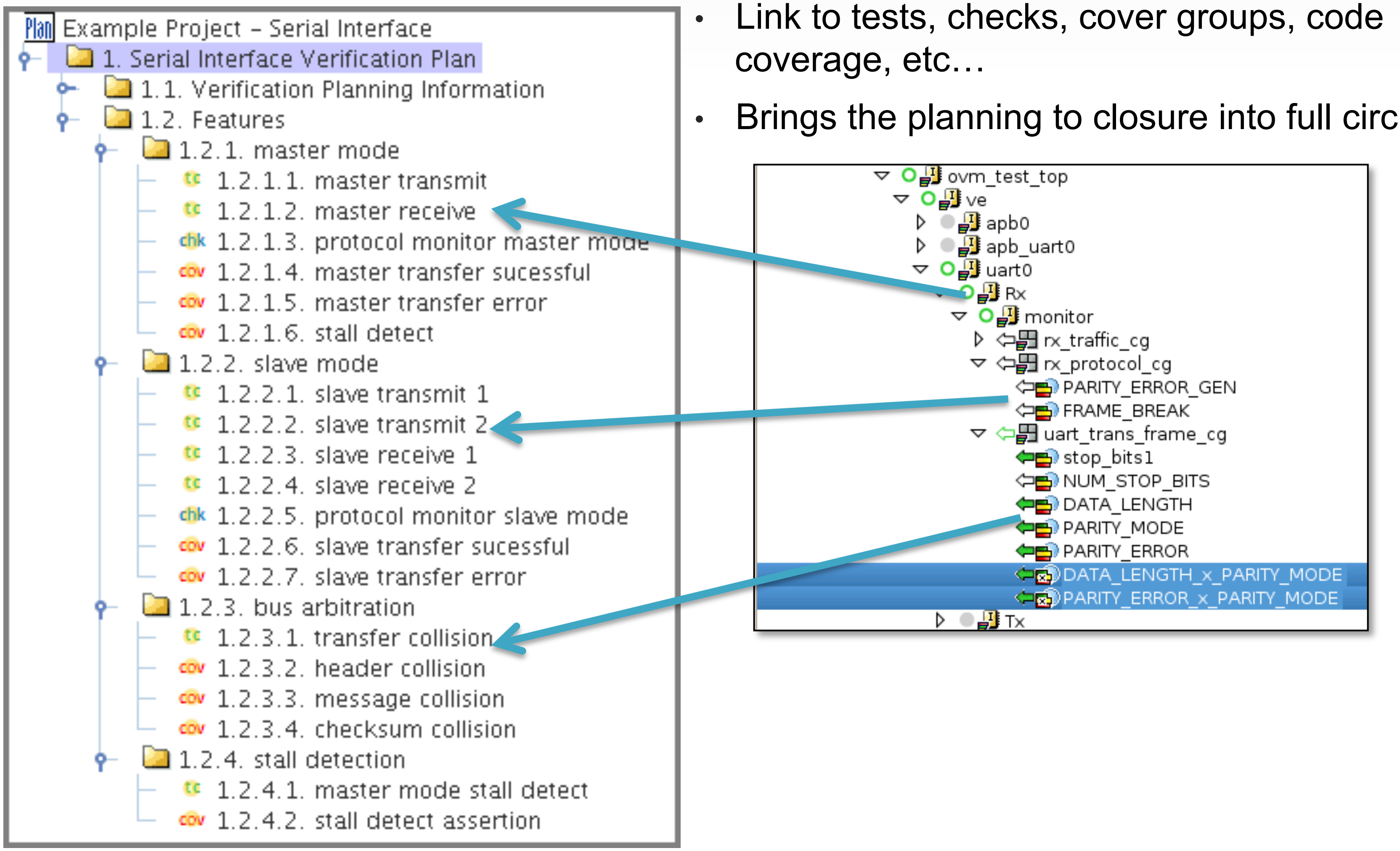
tc Both UART Rx and Tx paths support the configuration of various number of stop bits to detect on receive and to generate on transmit. Those values are: 1 stop bit, .5 stop bits, 2 stop bits.

Plan Example Project – Serial Interface

- 1. Serial Interface Verification Plan
  - 1.1. Verification Planning Information
  - 1.2. Features
    - 1.2.1. master mode
      - tc 1.2.1.1. master transmit
      - tc 1.2.1.2. master receive
      - chk 1.2.1.3. protocol monitor master mode
      - cov 1.2.1.4. master transfer sucessful
      - cov 1.2.1.5. master transfer error
      - cov 1.2.1.6. stall detect
    - 1.2.2. slave mode
      - tc 1.2.2.1. slave transmit 1
      - tc 1.2.2.2. slave transmit 2
      - tc 1.2.2.3. slave receive 1
      - tc 1.2.2.4. slave receive 2
      - chk 1.2.2.5. protocol monitor slave mode
      - cov 1.2.2.6. slave transfer sucessful
      - cov 1.2.2.7. slave transfer error
    - 1.2.3. bus arbitration
      - tc 1.2.3.1. transfer collision
      - cov 1.2.3.2. header collision
      - cov 1.2.3.3. message collision
      - cov 1.2.3.4. checksum collision
    - 1.2.4. stall detection
      - tc 1.2.4.1. master mode stall detect
      - cov 1.2.4.2. stall detect assertion

3. Implementation Planning: Correlated Results

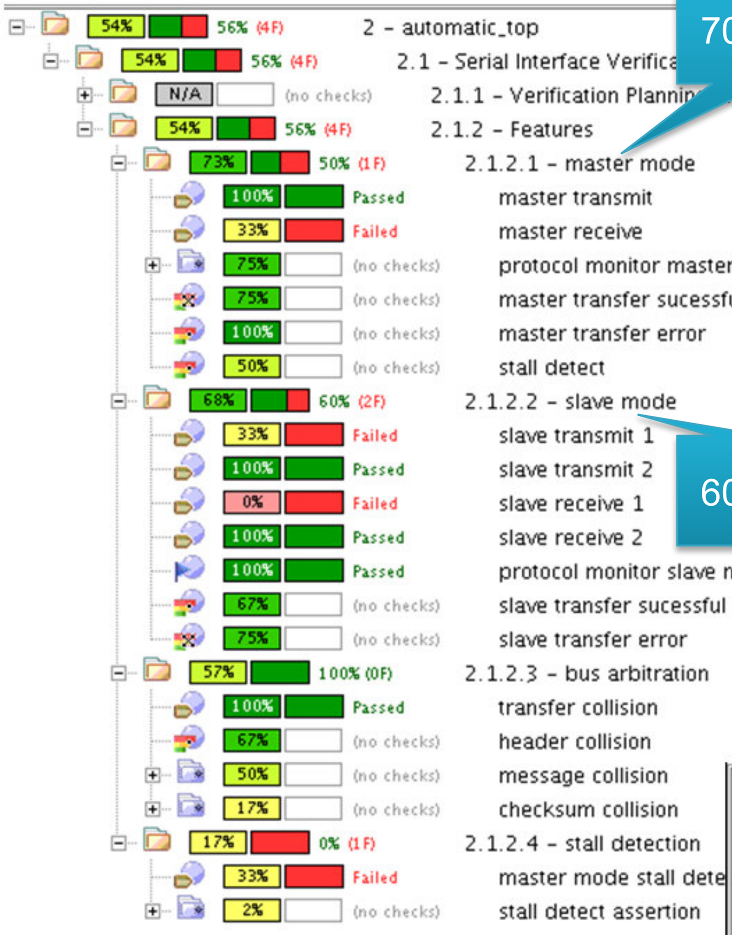
- Link to tests, checks, cover groups, code coverage, etc…
- Brings the planning to closure into full circle



```
Plan Example Project – Serial Interface
  1. Serial Interface Verification Plan
    1.1. Verification Planning Information
    1.2. Features
      1.2.1. master mode
        tc  1.2.1.1. master transmit
        tc  1.2.1.2. master receive
        chk 1.2.1.3. protocol monitor master mode
        cov 1.2.1.4. master transfer sucessful
        cov 1.2.1.5. master transfer error
        cov 1.2.1.6. stall detect
      1.2.2. slave mode
        tc  1.2.2.1. slave transmit 1
        tc  1.2.2.2. slave transmit 2
        tc  1.2.2.3. slave receive 1
        tc  1.2.2.4. slave receive 2
        chk 1.2.2.5. protocol monitor slave mode
        cov 1.2.2.6. slave transfer sucessful
        cov 1.2.2.7. slave transfer error
      1.2.3. bus arbitration
        tc  1.2.3.1. transfer collision
        cov 1.2.3.2. header collision
        cov 1.2.3.3. message collision
        cov 1.2.3.4. checksum collision
      1.2.4. stall detection
        tc  1.2.4.1. master mode stall detect
        cov 1.2.4.2. stall detect assertion
```

```
ovm_test_top
  ve
    apb0
    apb_uart0
    uart0
      Rx
        monitor
          rx_traffic_cg
          rx_protocol_cg
            PARITY_ERROR_GEN
            FRAME_BREAK
          uart_trans_frame_cg
            stop_bits1
            NUM_STOP_BITS
            DATA_LENGTH
            PARITY_MODE
            PARITY_ERROR
            DATA_LENGTH_x_PARITY_MODE
            PARITY_ERROR_x_PARITY_MODE
      Tx
```

4. Closure Planning

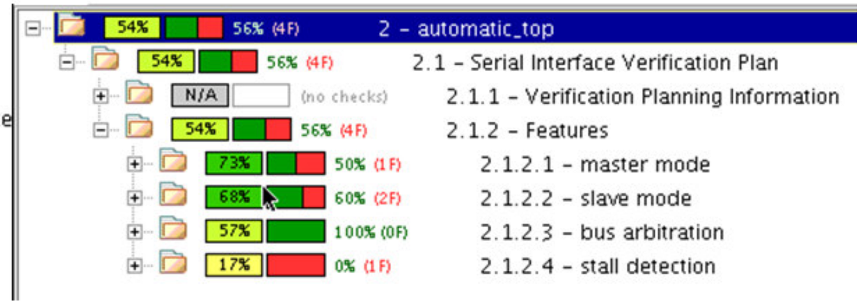


54%	56% (4F)	2 - automatic_top
54%	56% (4F)	2.1 - Serial Interface Verifica
N/A	(no checks)	2.1.1 - Verification Plannin ormation
54%	56% (4F)	2.1.2 - Features
73%	50% (1F)	2.1.2.1 - master mode
100%	Passed	master transmit
33%	Failed	master receive
75%	(no checks)	protocol monitor master mode
75%	(no checks)	master transfer sucessful
100%	(no checks)	master transfer error
50%	(no checks)	stall detect
68%	60% (2F)	2.1.2.2 - slave mode
33%	Failed	slave transmit 1
100%	Passed	slave transmit 2
0%	Failed	slave receive 1
100%	Passed	slave receive 2
100%	Passed	protocol monitor slave mode
67%	(no checks)	slave transfer sucessful
75%	(no checks)	slave transfer error
57%	100% (0F)	2.1.2.3 - bus arbitration
100%	Passed	transfer collision
67%	(no checks)	header collision
50%	(no checks)	message collision
17%	(no checks)	checksum collision
17%	0% (1F)	2.1.2.4 - stall detection
33%	Failed	master mode stall dete
2%	(no checks)	stall detect assertion

70% Goal

60% Goal

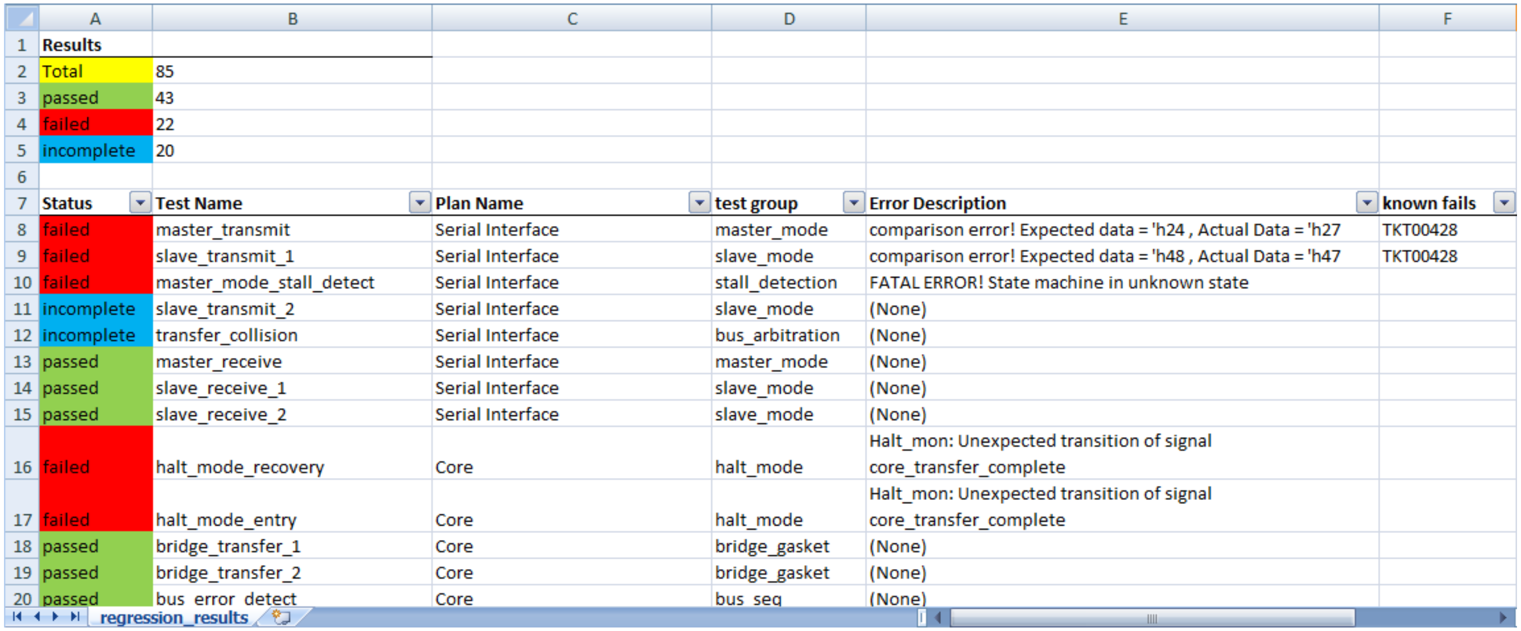
- Determine the signoff and completion criteria
- Don’t wait until the end
- Verification Completion Criteria must be based on metrics
  - Coverage metrics:
  - Executable verification plan includes direct coverage correlation



54%	56% (4F)	2 - automatic_top
54%	56% (4F)	2.1 - Serial Interface Verification Plan
N/A	(no checks)	2.1.1 - Verification Planning Information
54%	56% (4F)	2.1.2 - Features
73%	50% (1F)	2.1.2.1 - master mode
68%	60% (2F)	2.1.2.2 - slave mode
57%	100% (0F)	2.1.2.3 - bus arbitration
17%	0% (1F)	2.1.2.4 - stall detection

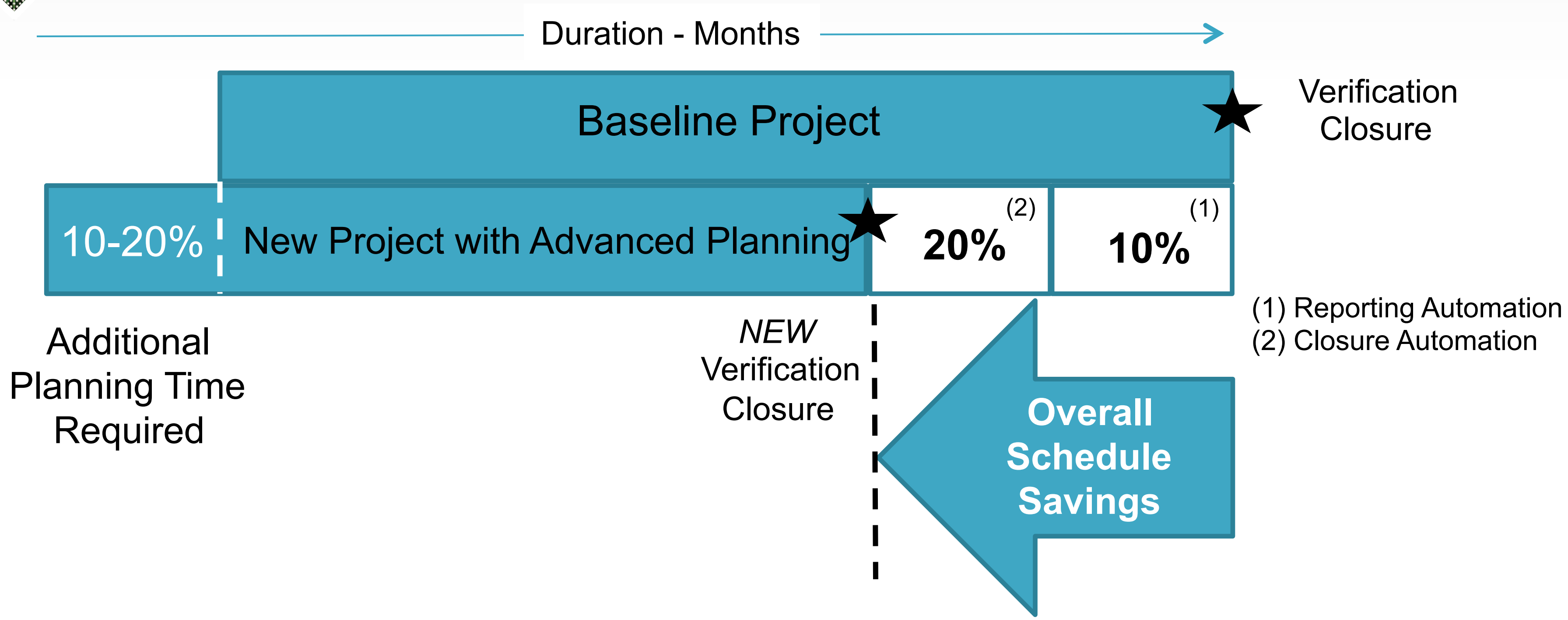
5. Plan for Reviews/Results/Reporting

- Plan for key milestone reviews and get commitment up front
  - If not, they don't get done
- Need a consistent review flow and formats
- Executable verification plan
  - includes direct results correlation
- Report merge and roll-up



	A	B	C	D	E	F
1	**Results**					
2	Total	85				
3	passed	43				
4	failed	22				
5	incomplete	20				
6						
7	**Status**	**Test Name**	**Plan Name**	**test group**	**Error Description**	**known fails**
8	failed	master_transmit	Serial Interface	master_mode	comparison error! Expected data = 'h24 , Actual Data = 'h27	TKT00428
9	failed	slave_transmit_1	Serial Interface	slave_mode	comparison error! Expected data = 'h48 , Actual Data = 'h47	TKT00428
10	failed	master_mode_stall_detect	Serial Interface	stall_detection	FATAL ERROR! State machine in unknown state	
11	incomplete	slave_transmit_2	Serial Interface	slave_mode	(None)	
12	incomplete	transfer_collision	Serial Interface	bus_arbitration	(None)	
13	passed	master_receive	Serial Interface	master_mode	(None)	
14	passed	slave_receive_1	Serial Interface	slave_mode	(None)	
15	passed	slave_receive_2	Serial Interface	slave_mode	(None)	
16	failed	halt_mode_recovery	Core	halt_mode	Halt_mon: Unexpected transition of signal core_transfer_complete	
17	failed	halt_mode_entry	Core	halt_mode	Halt_mon: Unexpected transition of signal core_transfer_complete	
18	passed	bridge_transfer_1	Core	bridge_gasket	(None)	
19	passed	bridge_transfer_2	Core	bridge_gasket	(None)	
20	passed	bus_error_detect	Core	bus_seq	(None)	

Formalized Planning: Value



Duration - Months

Baseline Project

Verification Closure

10-20%

New Project with Advanced Planning

20% (2)

10% (1)

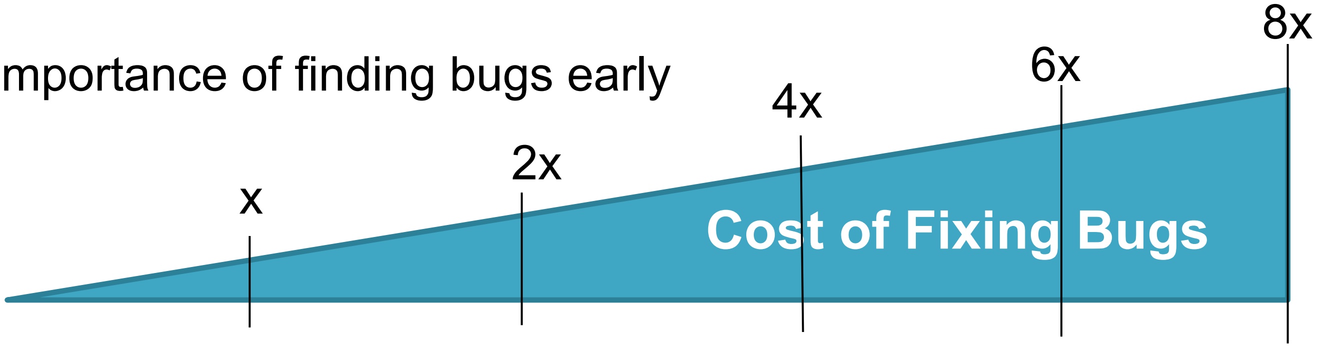
(1) Reporting Automation
(2) Closure Automation

Additional Planning Time Required

NEW Verification Closure

Overall Schedule Savings

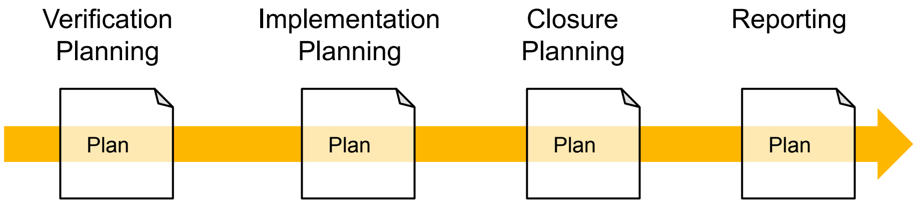
Importance of finding bugs early

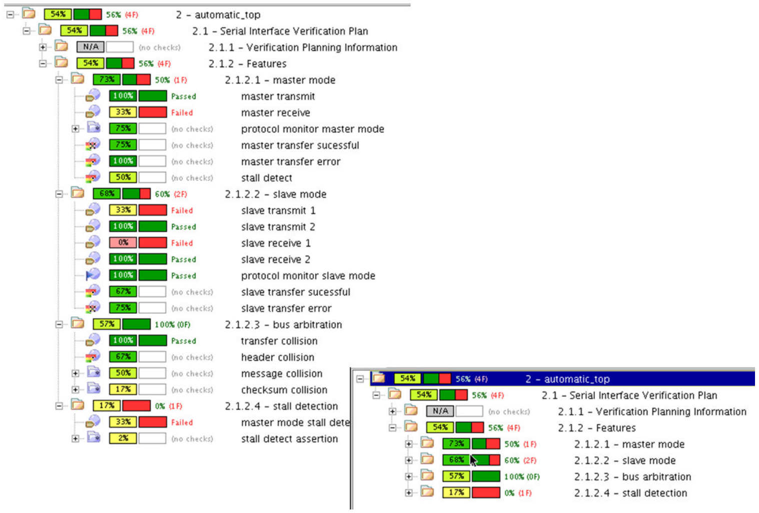


Cost of Fixing Bugs

Summary

- Verification Planning is not simply a task that is done once and forgotten
- It is a living, breathing and executable methodology saving time and resources
- There is a cost but with a tremendous set of benefits
- Analysis and reporting is automated
- Saved 10%-40% in Freescale projects





```
54%  56% (4F)     2 - automatic_top
  54%  56% (4F)     2.1 - Serial Interface Verification Plan
    N/A  (no checks)     2.1.1 - Verification Planning Information
    54%  56% (4F)     2.1.2 - Features
      73%  50% (1F)     2.1.2.1 - master mode
        100%  Passed        master transmit
        33%   Failed        master receive
        75%   (no checks)   protocol monitor master mode
        75%   (no checks)   master transfer sucessful
        100%  (no checks)   master transfer error
        50%   (no checks)   stall detect
      68%  60% (2F)     2.1.2.2 - slave mode
        33%   Failed        slave transmit 1
        100%  Passed        slave transmit 2
        0%    Failed        slave receive 1
        100%  Passed        slave receive 2
        100%  Passed        protocol monitor slave mode
        67%   (no checks)   slave transfer sucessful
        75%   (no checks)   slave transfer error
      57%  100% (0F)    2.1.2.3 - bus arbitration
        100%  Passed        transfer collision
        67%   (no checks)   header collision
        50%   (no checks)   message collision
        17%   (no checks)   checksum collision
      17%  0% (1F)      2.1.2.4 - stall detection
        33%   Failed        master mode stall dete
        2%    (no checks)   stall detect assertion
```

```
54%  56% (4F)     2 - automatic_top
  54%  56% (4F)     2.1 - Serial Interface Verification Plan
    N/A  (no checks)     2.1.1 - Verification Planning Information
    54%  56% (4F)     2.1.2 - Features
      73%  50% (1F)      2.1.2.1 - master mode
      68%  60% (2F)      2.1.2.2 - slave mode
      57%  100% (0F)     2.1.2.3 - bus arbitration
      17%  0% (1F)       2.1.2.4 - stall detection
```

Q/A

- Questions?

freescale™