Fighting fires and saving lives with MATLAB

Travis F. Collins, PhD
Analog Devices, Inc.

MATLAB EXPO 2021

Outline

- Problem statement
  - Existing problems of house fires
  - Update to UL 217/UL 268 smoke detector standards
  - Optical sensor technology
- Smoke Detector Design Methodology
  - Data collection campaign
  - Test driven development with MATLAB Unit Test Framework
  - Leveraging Parallel Computing Toolbox
  - Generating Embedded Code
  - MATLAB project as a product
- Summary

Fire Detection – Saving Lives

3 out of 5 Deaths

Properties without working smoke alarms

23% of deaths

Smoke Alarms present but disabled due to false alarms

83% Less Time

To escape a fire than in 1970's due to advances in synthetic building materials

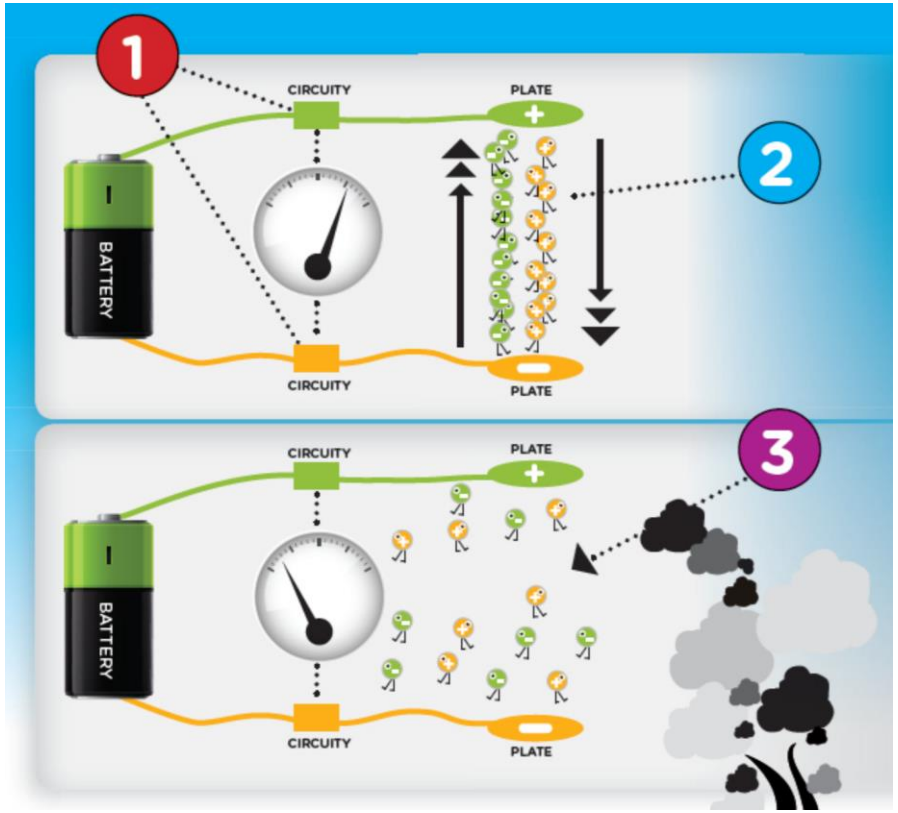
Driving increasing regulatory requirement for more reliable smoke detection

Major Smoke Detector Regulations

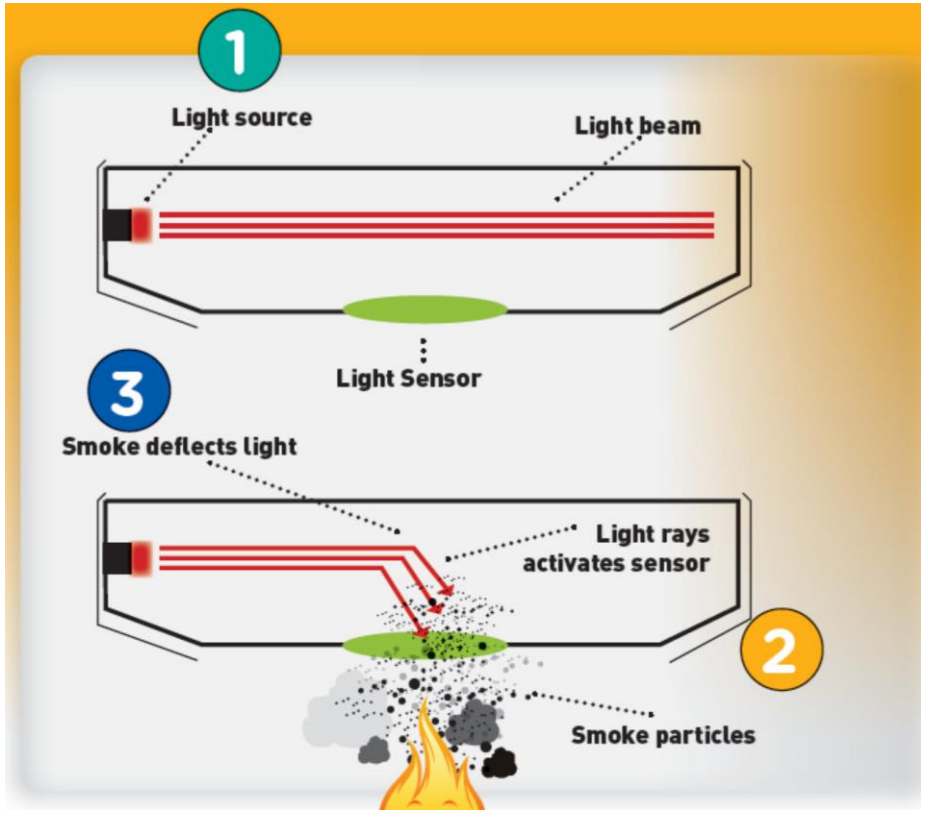
- US and Canadian
  - UL 268 - Smoke Detectors for Fire Alarm Systems
    - 7th edition - in effect 30th June 2021
  - UL 217 - Smoke Alarms
    - 8th edition - in effect 30th June 2021
  - ➢ ***Updates to flaming polyurethane and cooking nuisance (hamburger) test***
- European
  - EN 14604 - Smoke alarm devices (2006)
  - BS EN 54 - Fire detection and fire alarm systems (2015)
    - Part 29: Multi-sensor fire detectors — Point detectors using a combination of smoke and heat sensors
- International
  - ISO 7240 - Fire detection and alarm systems (2018)
    - Part 7: Point-type smoke detectors using scattered light, transmitted light or ionization
    - Chinese standard for point-type smoke detectors follows 2003 edition of this standard

Typical Smoke Detector Technology Today

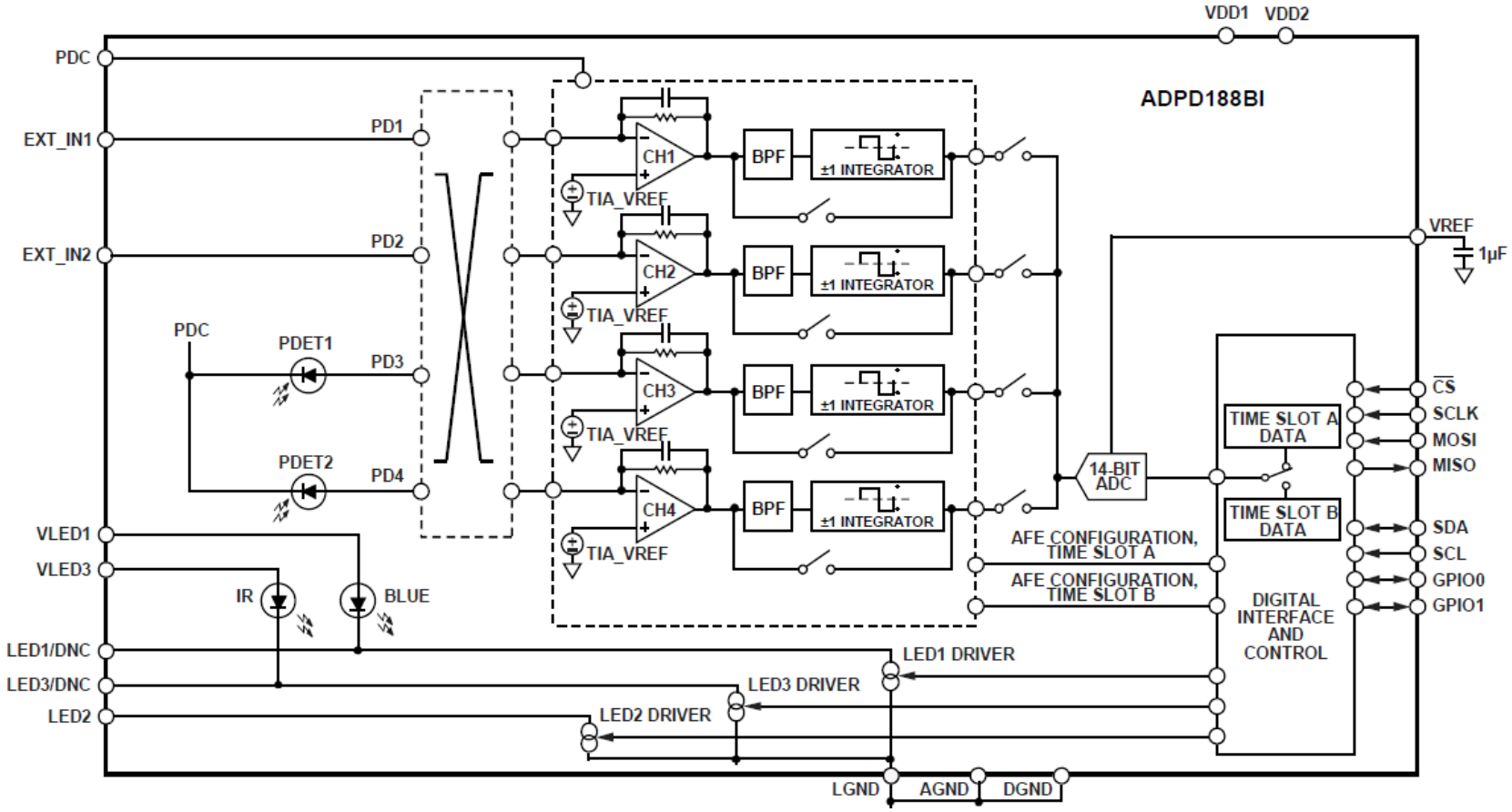
Ionization – Flaming Fires

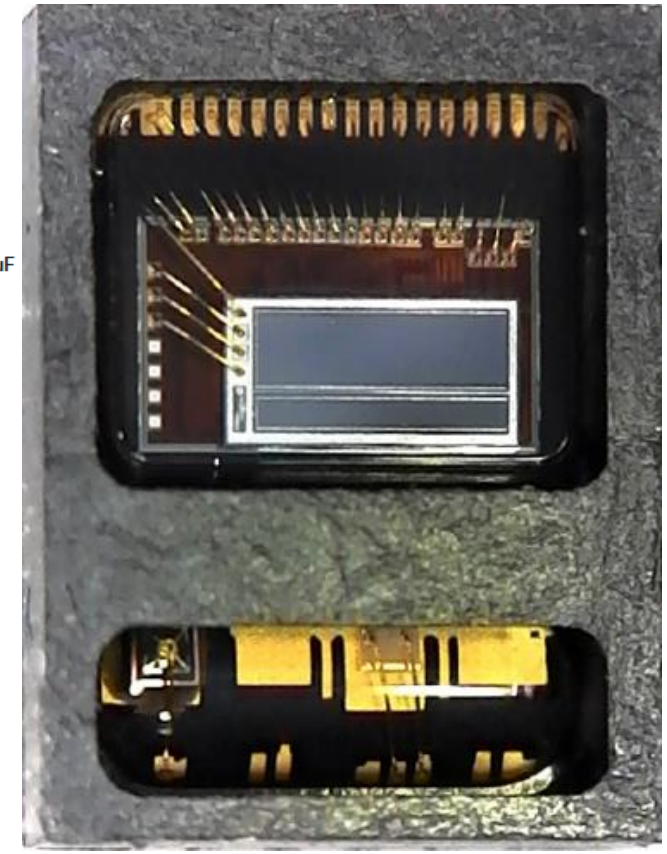


Photoelectric – Smoldering Fires



ADPD188BI Integrated Smoke Sensor



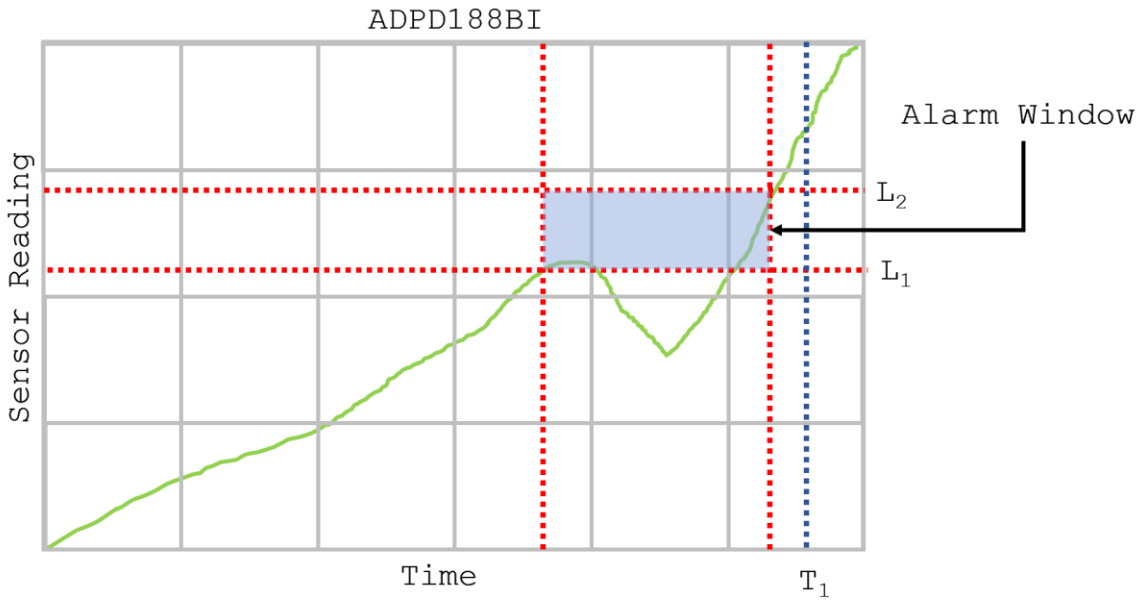


Photodiode and ADPD1080 AFE

Blue and IR LEDs

UL 217/UL 268 Motivations & Specification

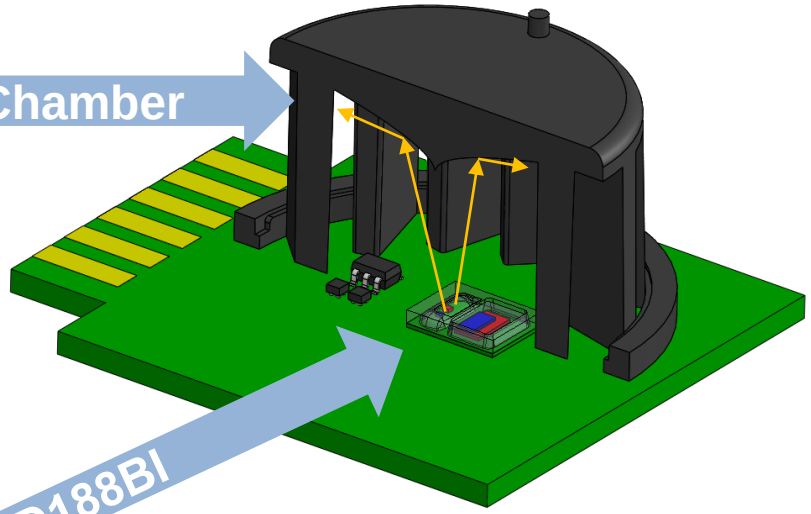
- UL 217/UL 268 Specification to cut down false alarms
  - Smoke vs non-Smoke
  - Real alarm vs Nuisance alarm
- **Research and understand how the ADPD188BI responds to smoke events**

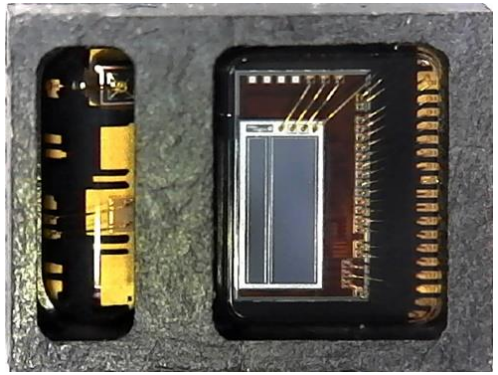


Fire Source	Alarm Time Spec.	Alarm Obscuration Spec.
Smoldering Wood	-----	before 10%/foot
Smoldering PU	-----	before 12%/foot
Nuisance	-----	NOT before 1.5%/foot
Paper	Less than 4 minutes from test start	-----
Flaming PU	Less than 4 minutes from test start	before 5%/foot
Flaming Wood	Less than 4 minutes from test start	-----
Nuisance+PU	Less than 4 minutes from test start	NOT before 1.5%/foot

Smoke Detector Evaluation System

Amplification Chamber





ADPD188BI

EVAL-CN0537-ARDZ

CN0537 UL 217/UL 268 Test Datasets

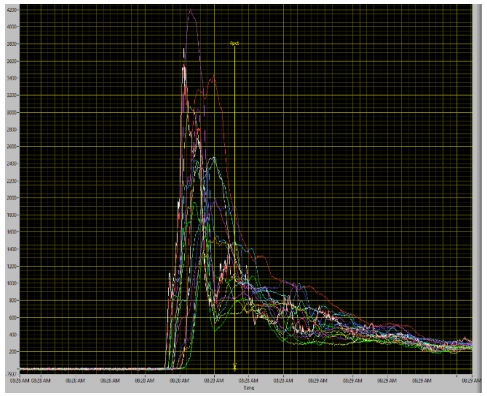
- Data collected at Occupational Safety and Health Administration(*OSHA*) recognized testing facilities Intertek and Underwriters Laboratory(UL)

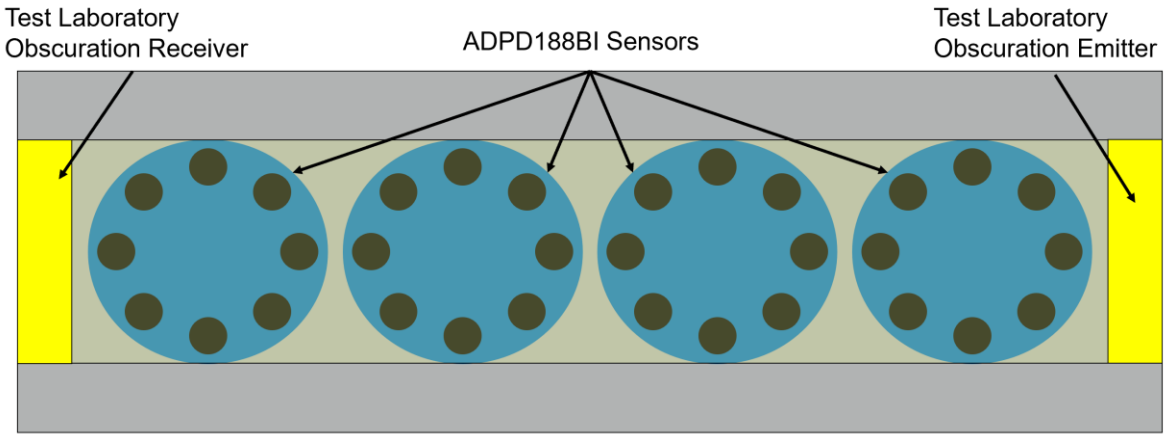
- EVAL-CN0537-DATA Datasets contain:
  - Reference obscuration/humidity/CO2 levels of UL 217/UL 268 test scenarios
  - High sample rate sensor data from multiple ADPD188BI parts across all tests
  - 1000+ unique part specific datasets

- Test data covers UL 217/UL 268 specific tests relating to smoke sensing performance

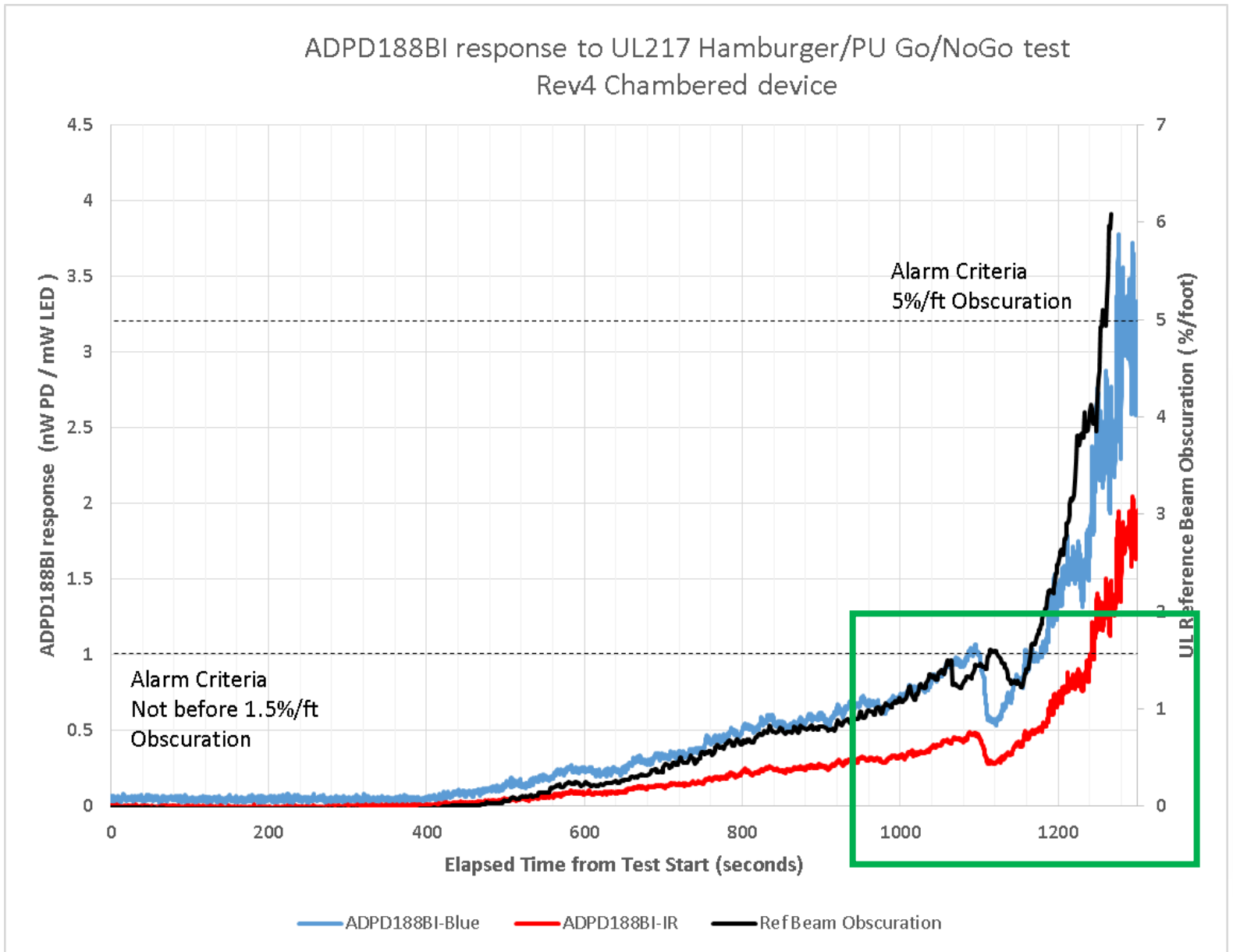
- Data can be used to further refine algorithms, create custom algorithms, or complement existing test harness

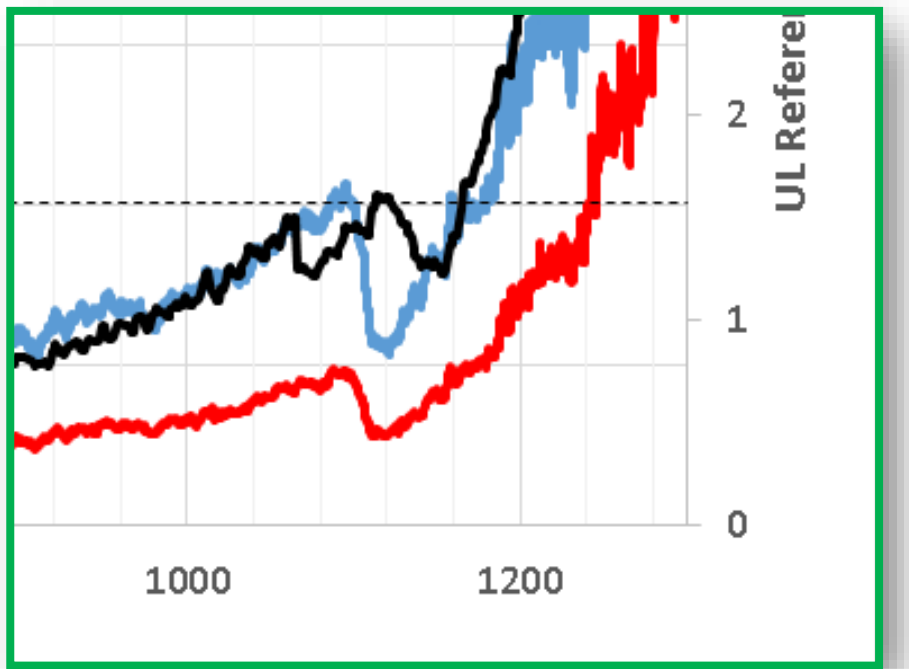
UL Section	UL 8th Edition Test
42	Sensitivity
51.2	Paper Fire
51.3	Wood Fire
51.4	Flaming PolyUrethane
52	Smoldering Smoke
53	Smoldering PolyUrethane
54	Cooking Nuisance
	UL 9th Edition Test
54	Go/No Go Cooking Nuisance





UL217/UL 268 Nuisance Test





Design Methodology: Test driven development

CN0537-Product

Feasibility Study

Requirements Definition

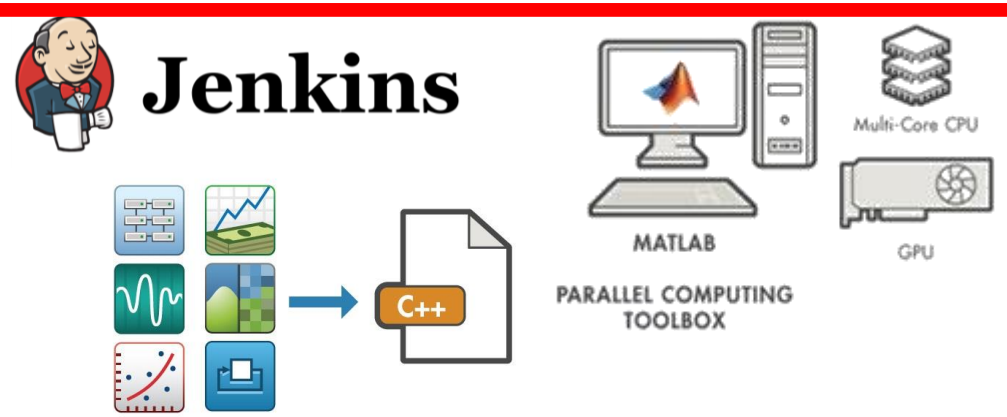
Algorithm Design

Algorithm Verification

Embedded Target Elaboration

Embedded Verification

MATLAB Unittest Definitions



MATLAB Unit Tests

Mapping requirements to code and testing to requirements

- Took requirements from UL 217/UL 268 standard and implemented them directly in code as **matlab.unittest.constraints**
- Leveraged test parameterization for data source test case generation
- Easily describe data tested, checks performed, and extend analysis

```
methods(Test)
    function testConfiguration(testCase, ...
            testIndx, a, b, c, d, e, f, g)
        import matlab.unittest.constraints.IsLessThan
        import matlab.unittest.constraints.IsGreaterThan

        % Extract test data
        testCase.index = testIndx;
        testCase.test_mode = mode;
        x = testCase.smoke_data(testIndx).b;
        t = testCase.smoke_data(testIndx).t1;

        %% Pass through algorithm
        [cs_time, alarm_time_indx] = smoke_detector(x, t, ...
            a, b, c, d, e, f, g); %#ok<*ASGLU>

        %% Validate
        switch testCase.smoke_data(test_index).test_info.smoke_source
            case {'Flaming PU'}
                testCase.assertThat(cs_time, IsGreaterThan(0), 'alg_obsc_min');
                testCase.assertThat(cs_time, IsLessThan(5), 'alg_obsc_max');
                testCase.assertThat(alarm_time_indx, IsLessThan(240), 'alg_atime');
            case {'Paper', 'Wood Fire'}
                testCase.assertThat(cs_time, IsGreaterThan(-Inf), 'alg_obsc_min');
                testCase.assertThat(cs_time, IsLessThan(Inf), 'alg_obsc_max');
                testCase.assertThat(alarm_time_indx, IsLessThan(240), 'alg_atime');
        end
    end
```

Extending Unit Tests

Unit tests are not just pass fail

- Added diagnostic reporting for performance analysis
  - Done through custom plugins
- Further extending with MATLAB Report Generation
- MATLAB CI control and generated artifacts easily integrate with Jenkins MATLAB plugin

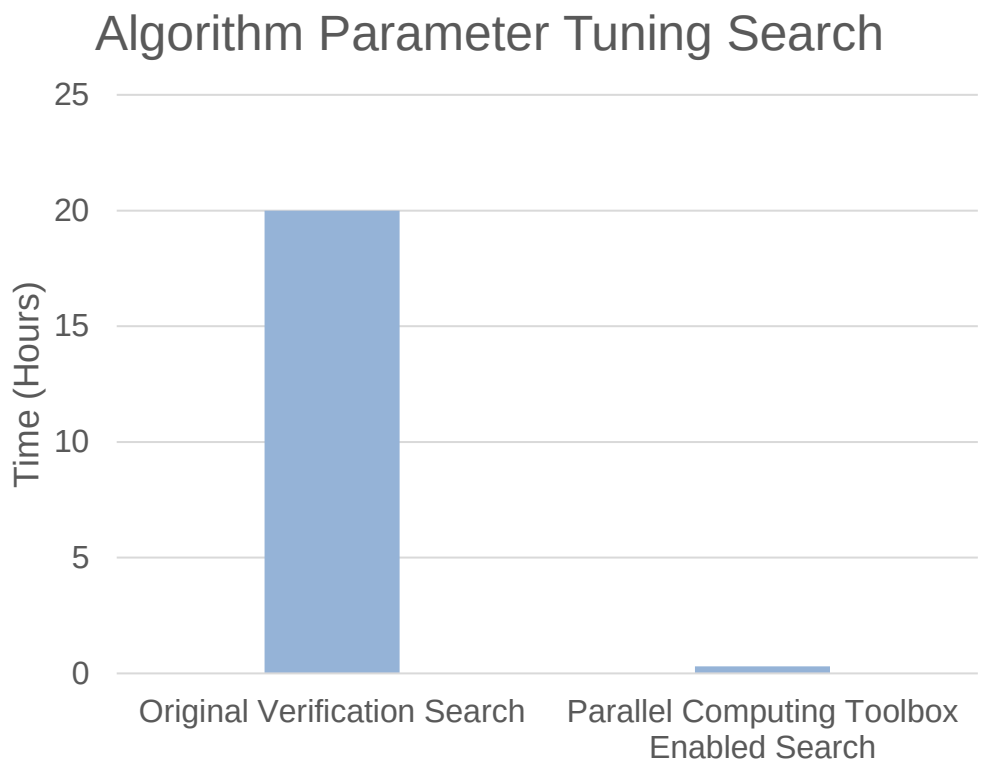
```
%% Custom Unit Test Plugin
classdef details_recording_plugin < matlab.unittest.plugins.TestRunnerPlugin
    %% Define Fields To Collect
    properties (Constant,Access = private)
        ObsObscLvlField = 'alg_alarm_time_obsc_lvl';
        TargetObscLvlField = 'spec_alarm_time_obsc_lvl_range';
        ObsAlarmTimeField = 'alg_alarm_time';
        TargetObsAlarmTimeField = 'spec_alarm_time';
        TestInfoField = 'test_info';
        ModeField = 'mode'
    end


    methods (Access = private)
        %% Collect Assertion Results
        function reactToAssertion(plugin,evd,resultDetails)
            if strcmp(evd.TestDiagnostic, 'alg_obsc_min')
                resultDetails.append(plugin.TargetObscLvlField,{evd.Constraint.FloorValue})
            elseif strcmp(evd.TestDiagnostic, 'alg_obsc_max')
                resultDetails.append(plugin.ObsObscLvlField,{evd.ActualValue})
                resultDetails.append(plugin.TargetObscLvlField,{evd.Constraint.CeilingValue})
            elseif strcmp(evd.TestDiagnostic, 'alg_atime')
                resultDetails.append(plugin.ObsAlarmTimeField,{evd.ActualValue})
                resultDetails.append(plugin.TargetObsAlarmTimeField,{evd.Constraint.CeilingValue})
                resultDetails.append(plugin.TestInfoField,{evd.Source.smoke_data(evd.Source.index).test_info})
                resultDetails.append(plugin.ModeField,{evd.Source.test_mode})
            end
        end
    end
end
```

Algorithm Tuning and Exploration

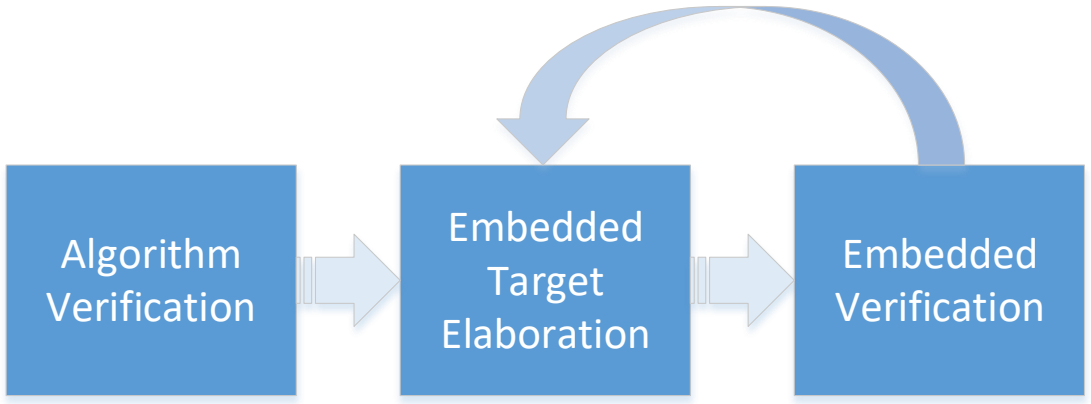
Set tuning parameters → Simulate all 1000+ scenarios → Validate alarm conditions

- Easy integration with testing framework
  - Test parameterization allows for simple multiplicative dimensioning
- Went from 20 hours to 20 minutes
- Mindsets change when you don't have to wait hours and hours for a simulation
- **Switch from waiting to exploring**

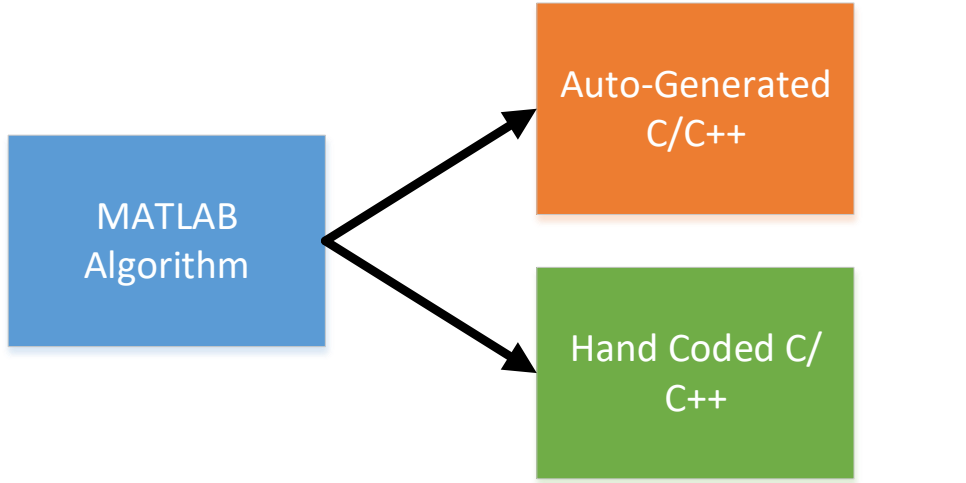
```
classdef UL217Tests < matlab.unittest.TestCase
    properties (TestParameter)
        testIndx = enumerateDataSet()
        implementation = {'matlab', 'c', 'python'};
        a = {1};
        b = {45};
        c = {4};
        d = {128,123,11};
        e = {0.1};
        f = {0.2};
        g = {1,2,3};
    end

    methods(Test)
        function test_UL_and_Intertek_Configuration(testCase, ...
                testIndx, implementation, a, b, c, d, e, f, g)
```

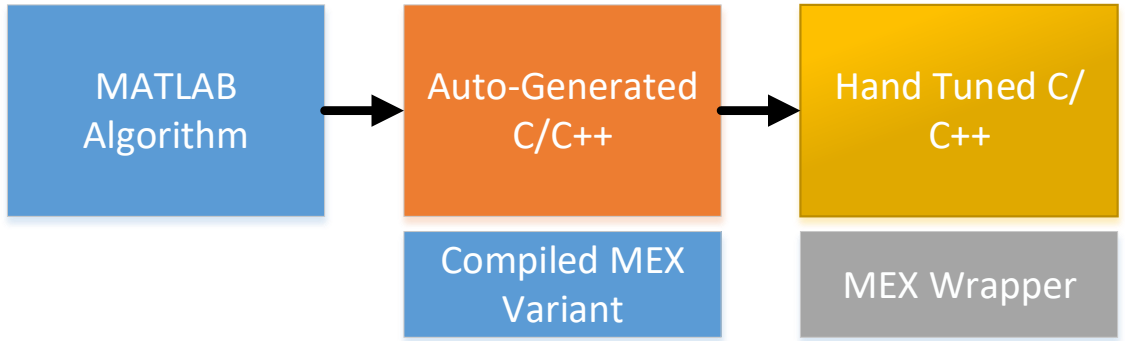


Generating Embedded Code



- Port exact algorithm to embedded hardware without functionality or precision loss
- Minimize mathematical operations to reduce power requirements on hardware
- Connect driver API and requirements to algorithmic code
- Clear communication and validation between driver (embedded) engineers and algorithmic engineers
- Parallel Computing Toolbox also allows for parallel runs with mex'ed code

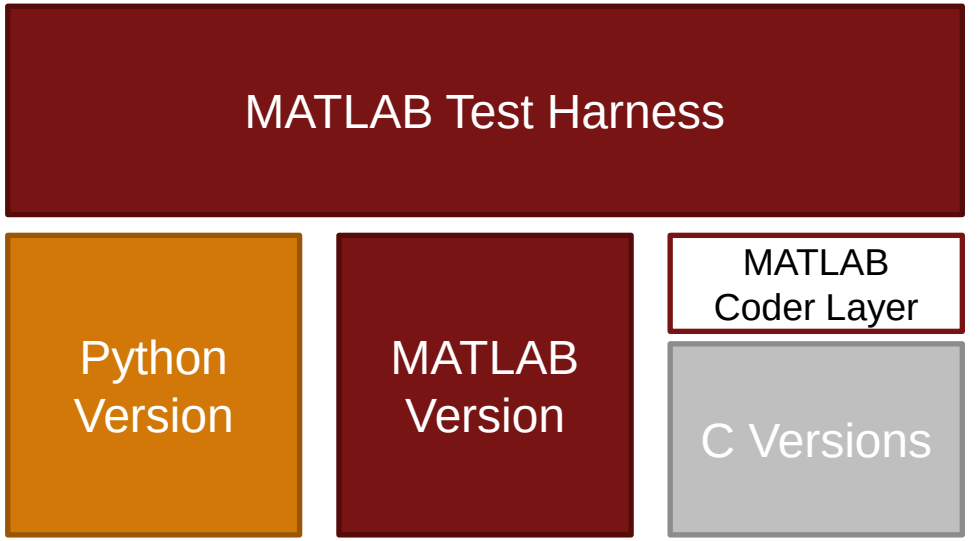




Software Testing harness

- Fully integrated testing harness across implementations include MATLAB/Python/C
  - Allows for direct comparison as the algorithm is moved embedded
- The harness itself is written in MATLAB and utilizes MATLAB Coder to interface with C and MATLAB's standard python integration
- Testing harness generates plots to help visualize the margins with which the research fire tests comply with UL 217/UL 268

```
%% Pass through algorithm
switch implementation
    case 'matlab'
        [cs_time, alarm_time_indx] = smoke_detector(x, t, ...
            a, b, c, d, e, f, g);
    case 'c'
        [cs_time, alarm_time_indx] = smoke_detector_c_wrapper_mex(x, t, ...
            a, b, c, d, e, f, g);
    case 'python'
        pa = py.python.smoke;
        [cs_time, alarm_time_indx] = pa.smoke_detector(x, t, ...
            a, b, c, d, e, f, g);
end
```



UL 217/UL 268 Testing Results

- Tested and verified UL 217 (8th and 9th Ed.) and UL 268 (7th Ed.) smoke detection algorithm (.c code)
- Complete documentation can be found on wiki.analog.com
  - Note – Test and verified with the current ADPD188BI, smoke chamber, and algorithm.
- **Able to validate ahead of test facility by feeding data into test harness between tests**

Smoke Alarms [UL 217:2015 Ed.8+R:23Nov2016]

SECTION 1

SUMMARY

Intertek wishes to inform you that we have completed the research UL217 8th Ed performance testing on your EVAL-CN0537-ALGO Smoke sensor. The following list of tests were performed and resulted in a passing result.

TEST	UL 217 8th	Result
Directionality	43	Pass
Sensitivity	42	Pass
UL – Paper Fire	51.2	Pass
UL – Wood Fire	51.3	Pass
UL – Flaming polyurethane Foam Test	51.4	Pass
UL – Smoldering Smoke Test	52	Pass
UL – Smoldering Polyurethane Foam Test	53	Pass
UL – Cooking Nuisance Smoke Test	54	Pass
UL - Go/No Go Flaming Polyurethane Foam Test	54	Pass
Velocity-Sensitivity Test	44	Pass
Variable Ambient (0 & 49c)	62	Pass
Humidity	63	Pass

CN0537 Reference Design Offerings

Algorithm

- Software
  - CN0537 Source Code including UL 217/UL 268 Detection Algorithm (.c)
  - **MATLAB UL 217/UL 268 Projects**
- Data
  - UL 217/UL 268 Test Dataset Files
- Documentation
  - Algorithm Documentation
  - Test Datasets User Guide
  - MATLAB User Guide
- Support
  - 10 hours of phone support
  - Additional paid support available if required

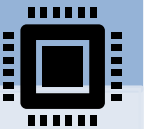
EVAL-CN0537-ALGO

Data

- Data
  - UL 217/UL 268 Test Datasets Files
- Software
  - CN0537 Source Code (excl. detection algorithm)
- Documentation
  - Test Datasets User Guide

EVAL-CN0537-DATA

Hardware



- Hardware
  - CN0537 Reference Design
  - ADICUP3029 Microcontroller Board
- Software
  - Embedded UL 217/UL 268 algorithm (.hex)
  - ADPD188BI no-OS driver
- Documentation
  - Circuit Note
  - CN0537 User Guide
  - UL 217/UL 268 Test Results (Intertek)

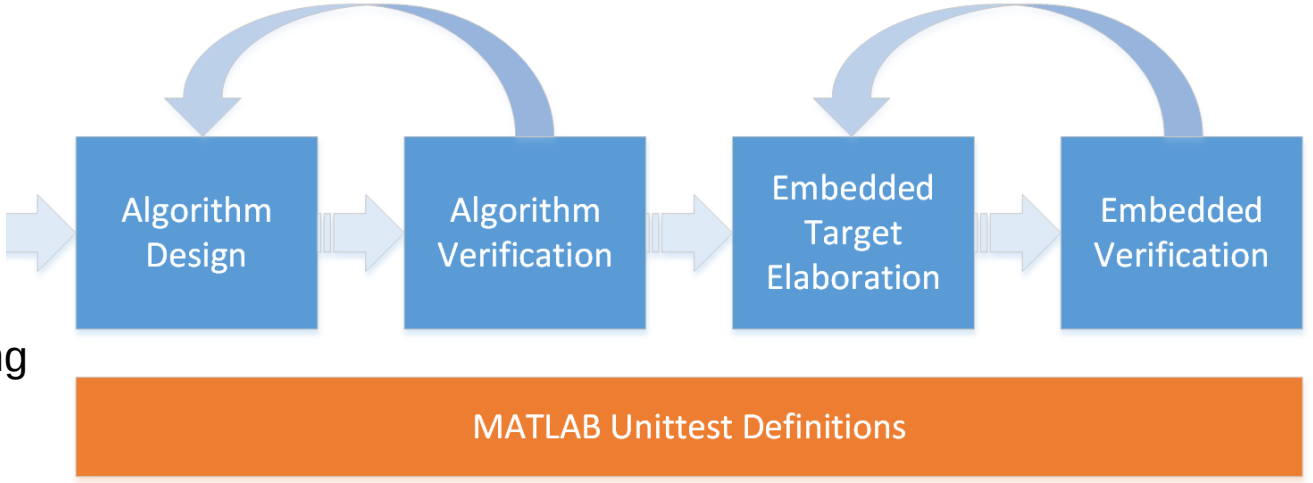
EVAL-CN0537-ARDZ
EVAL-ADICUP3029

Summary

Model-Based Design aka

Test Driven Development

- MATLAB frameworks and advanced tools provide:
  - Consistent validation environment from simulation to production
  - Spend time solving problems rather than waiting for simulations
  - Delivering MATLAB code as a product allows engineering customers to move quickly and tune for their unique cases
- MATLAB User Story available at MathWorks.com
- **Please visit our booth for more information**



MATLAB EXPO 2021

Thank you