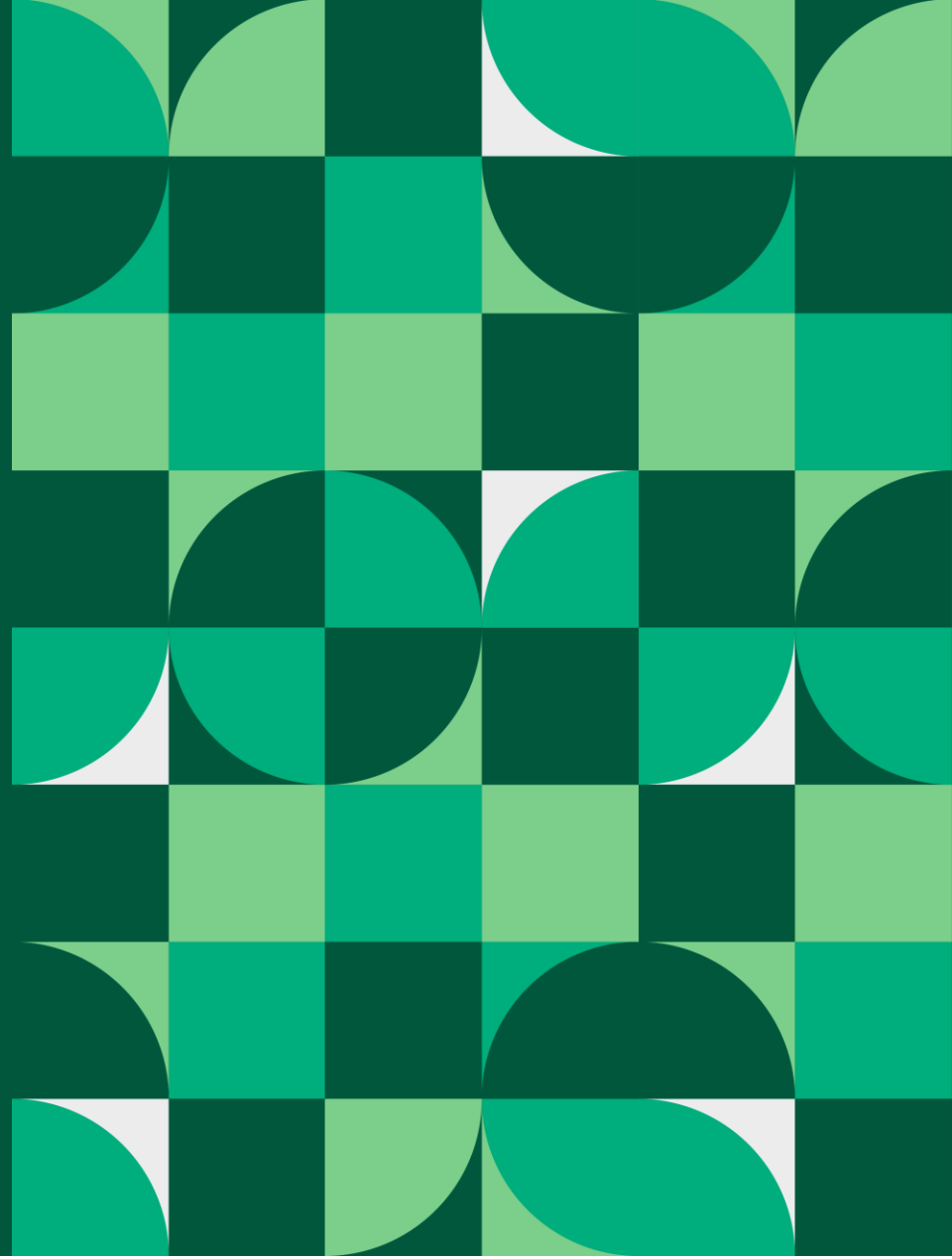
Design for reuse

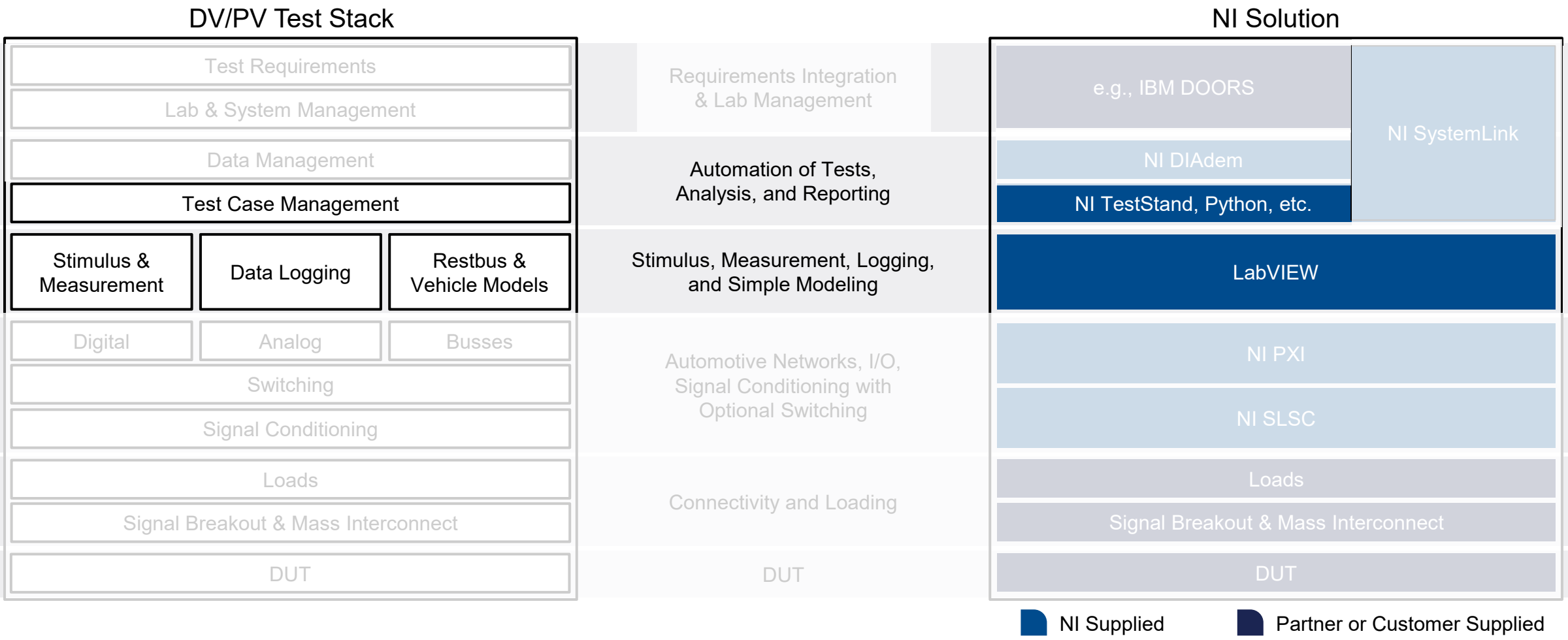
Daniel Eaton

Chief Field Applications Engineer

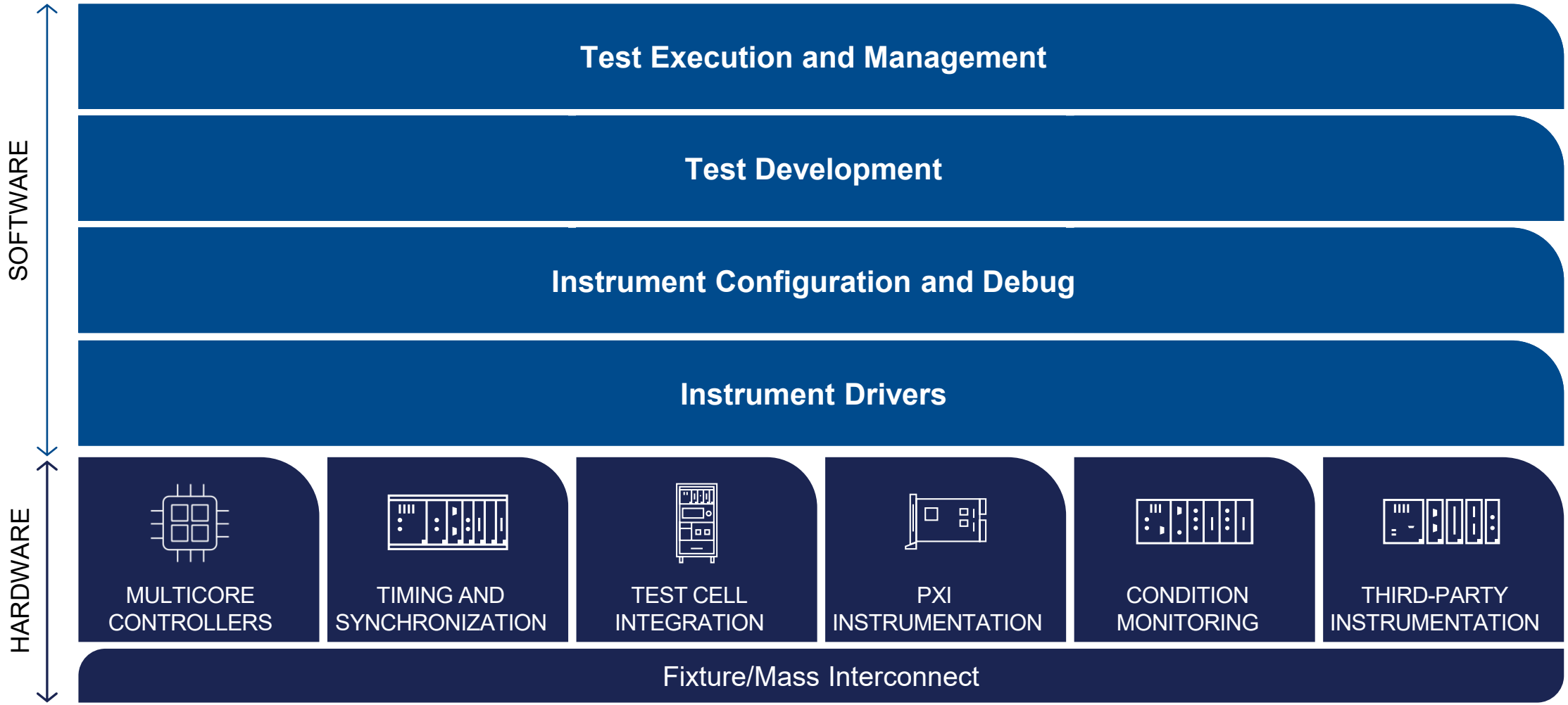


EMERSON

Hardware Validation Solution Overview

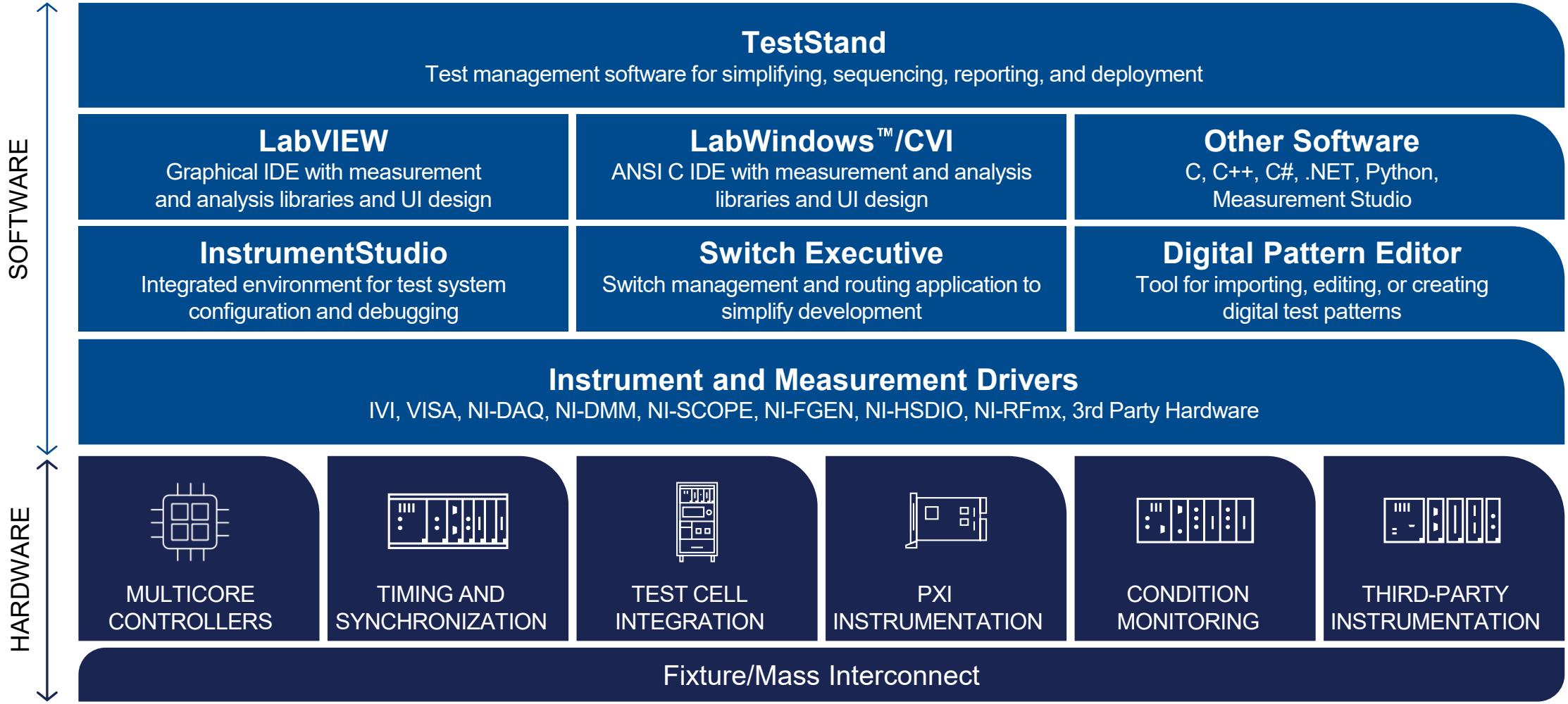


Architecture of an Automated Test System



EMERSON | NI

Architecture of an Automated Test System



TestStand

A test executive software that accelerates system development and deployment for engineers in validation and production.

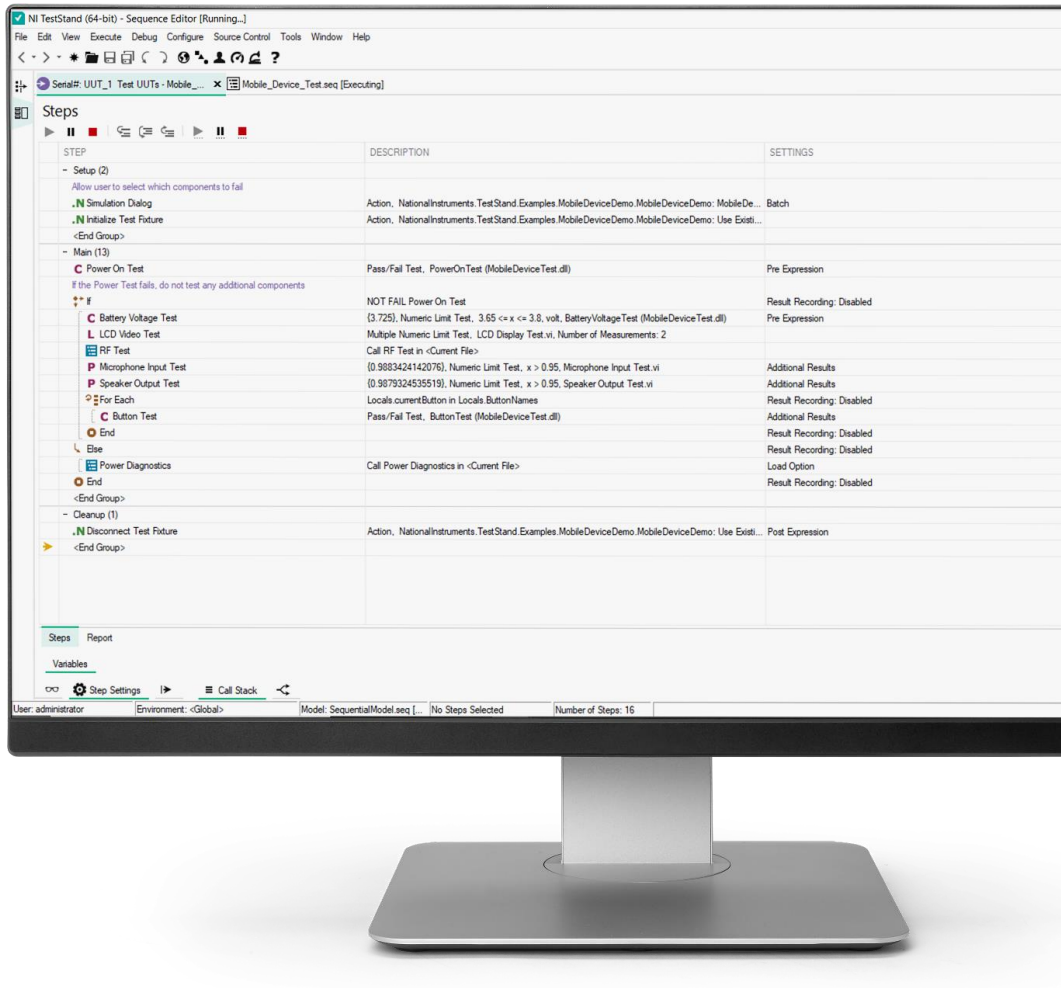
Automate Your System
Create, execute, and debug test sequences using an interactive environment

Leverage Existing Code
Use code from LabVIEW, Python, C/C++, or .NET

Test More, Faster
Use native parallel execution to reduce test time and functionality for advanced tasks such as sweeping and looping

Keep Track of Results
Track units, automatically generate reports, and store results to local or network databases

Deploy To Your Testers
Scale your operations by deploying sequences to numerous test stations with custom or pre-built operator interfaces



EMERSON

LabVIEW

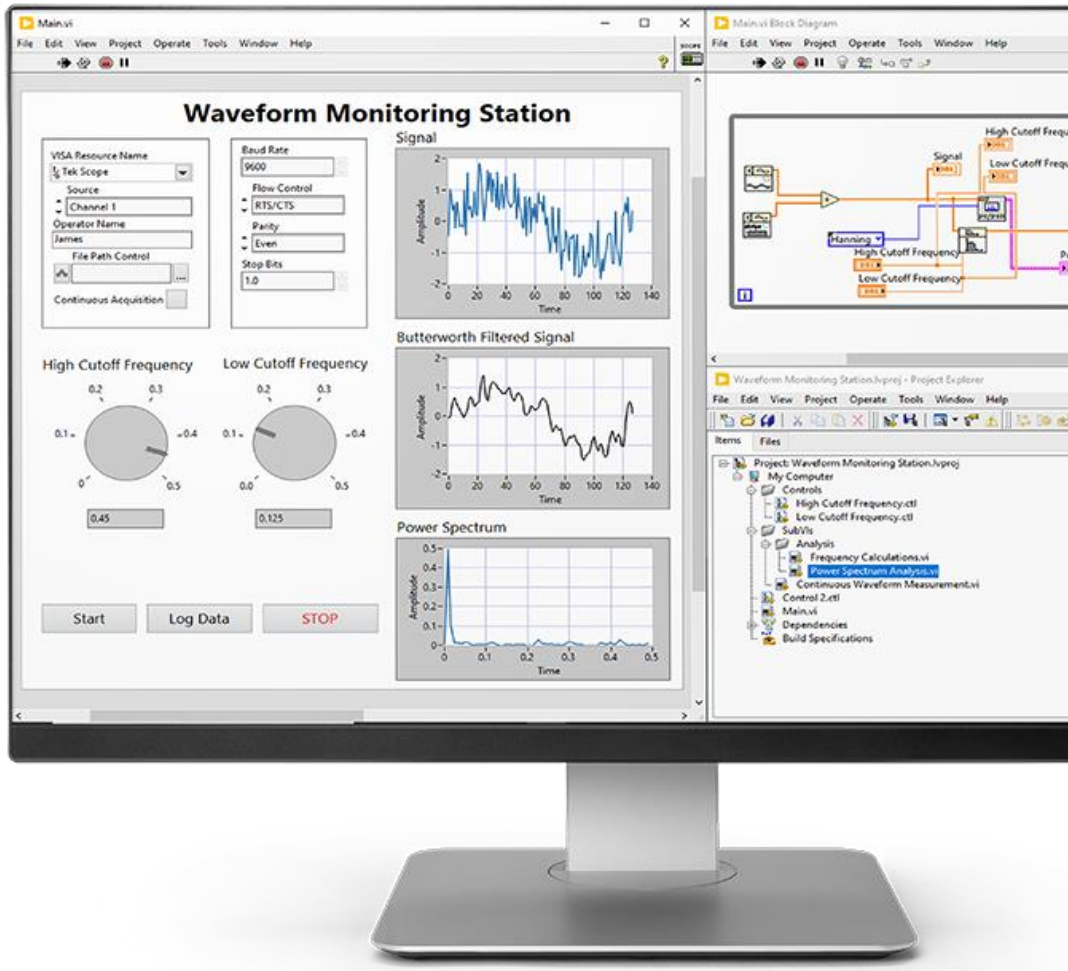
A graphical programming environment engineers use to develop automated research, validation, and production test systems.

Create Professional User Interfaces
View data and control your test system via an interactive UI built from drag-and-drop UI elements

Integrate All Your Instruments
Acquire data from and control any instrument with 1000s of device drivers and industry-standard protocols

Program Like You Think
Save development time by creating and visualizing applications using data flow programming.

Use Other Code
Leverage other and existing code written in Python, C/C++, MATLAB®, and .NET



EMERSON

North Star: Save That Money…

- By maximizing test steps / sequences /software re-use for:
  - Any Rack
  - Any DUT
  - Any number of DUTs

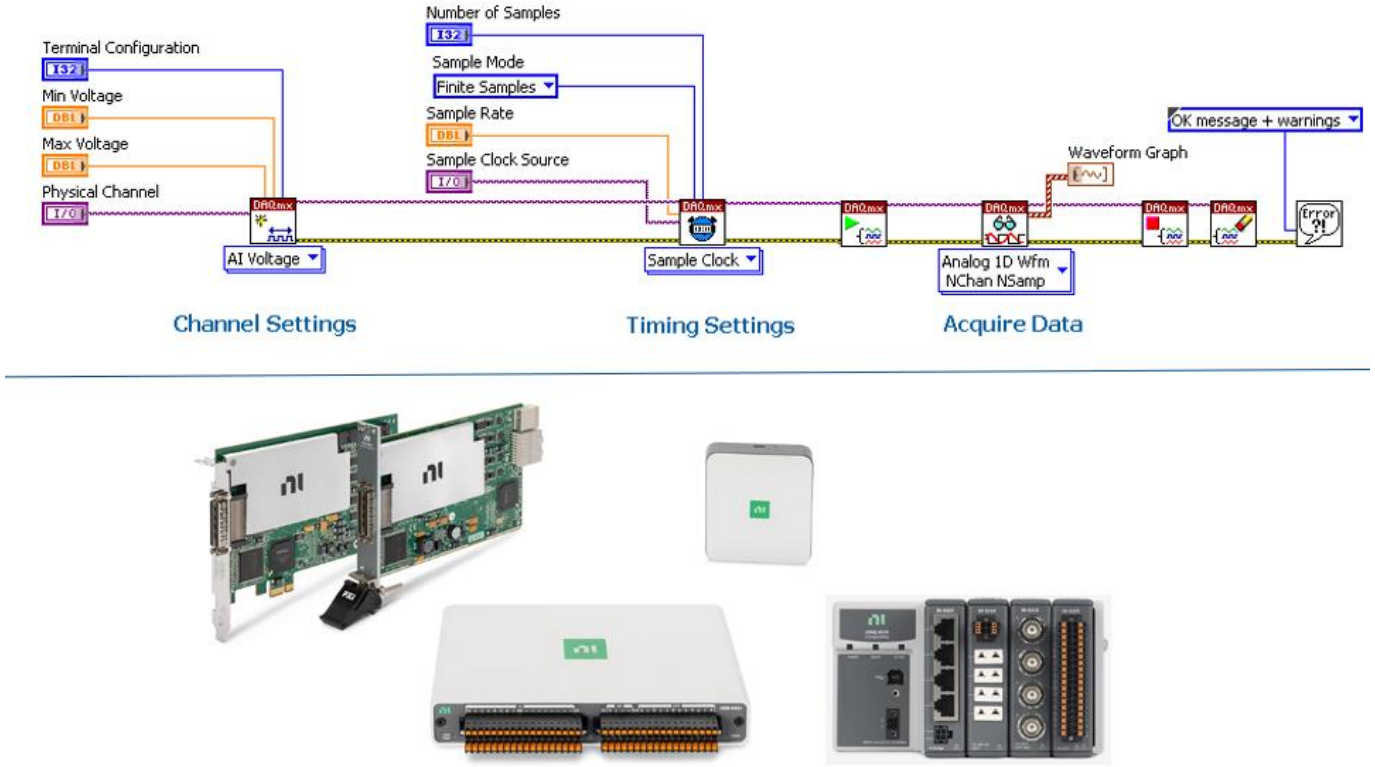
How: Any Rack

Use software to do

- Hardware Abstraction
- Pin Abstraction

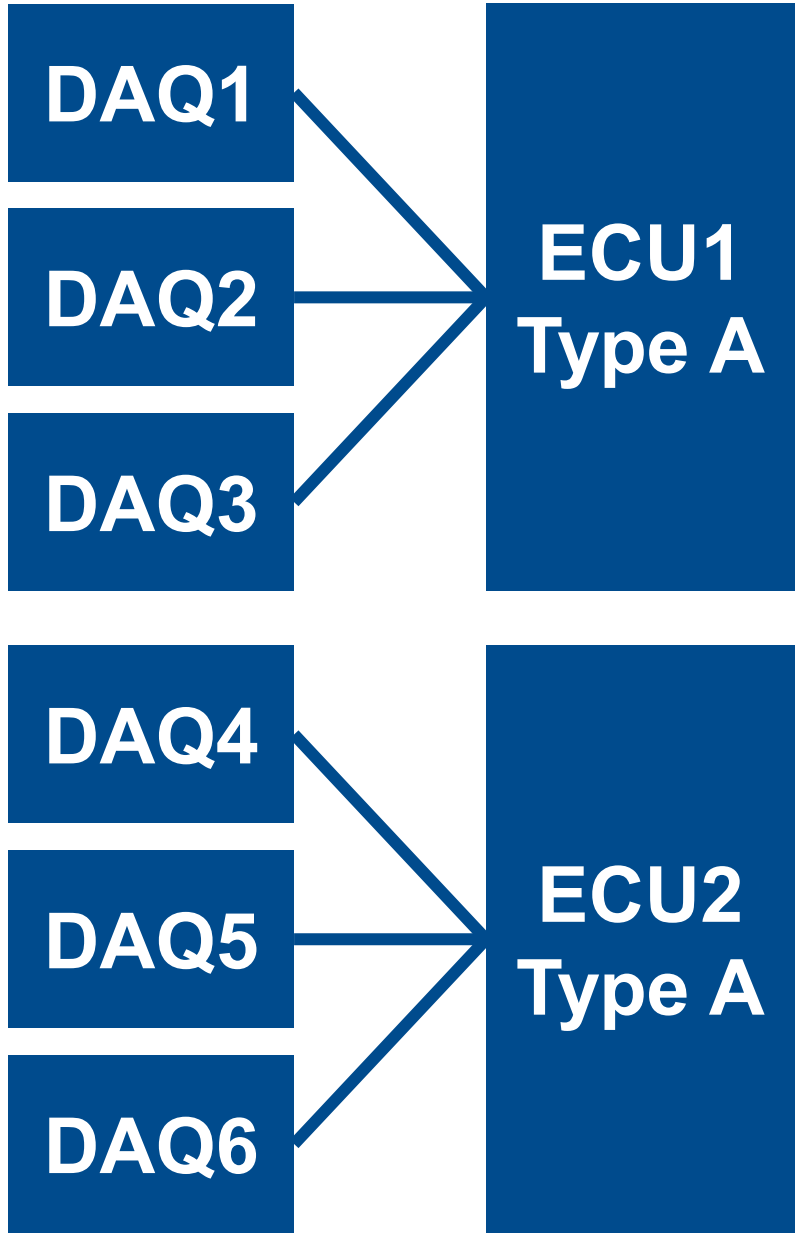
Hardware Abstraction

- Multiple Devices → One API (Software Interface)
- Pick rack specific capabilities → One API (Software Interface)
- NI APIs are HAL for NI devices
  - NI DAQmx: Multi-purpose DAQ
  - NI XNET: Automotive Buses
  - NI Switch: Switches
  - Etc
- Third-party hardware requires a higher-level, user defined HAL



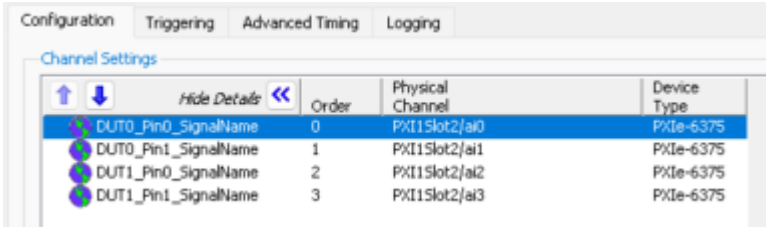
Pin Abstraction

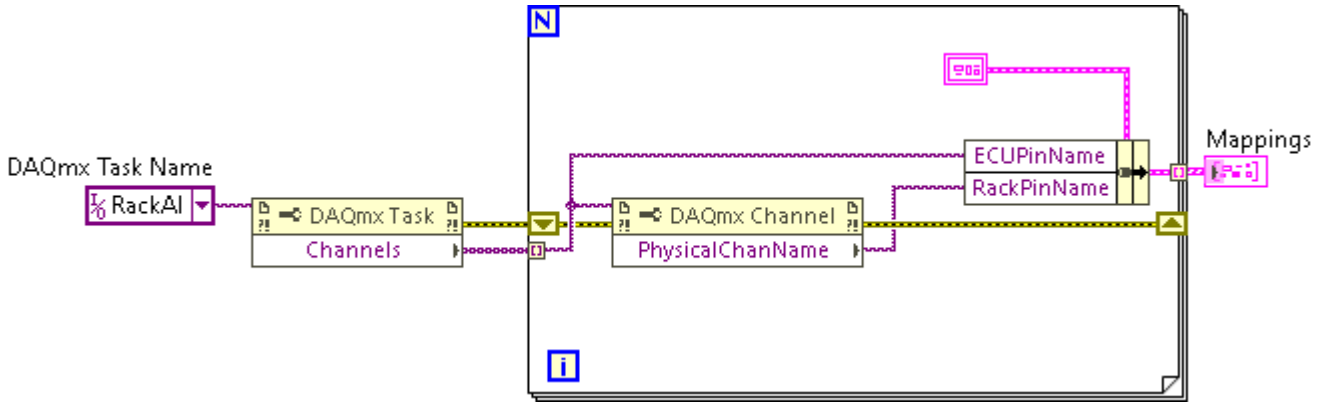
- Abstract rack hardware from sequences/steps
- Scenario: 3 Devices → 1 DUT
- Which option is better re-use?
  - DAQ1/AI1 or ECU.Pin1
  - DAQ2/DO1 or ECU.Pin2
  - DAQ3/AO1 or ECU.Pin3
- Should test sequences care about devices or your ECU?
- Scenario: 6 Devices → 2 DUTs
- Approach: Map ECU Pin → Rack specific pin



Pin Abstraction: Mapping Examples

- DAQmx Global Channels + Naming Convention





Pin Abstraction: DAQmx Global Channels

Pros

- Simple
- GUI Based Configuration
- Naturally include channel properties (voltage levels, etc)
- Exportable
- Allows for naming scheme
- Task required regardless

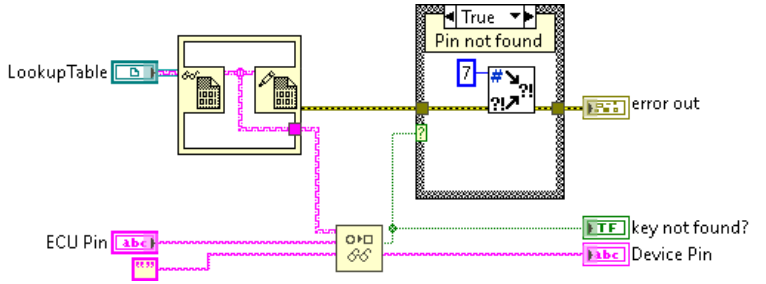
Cons

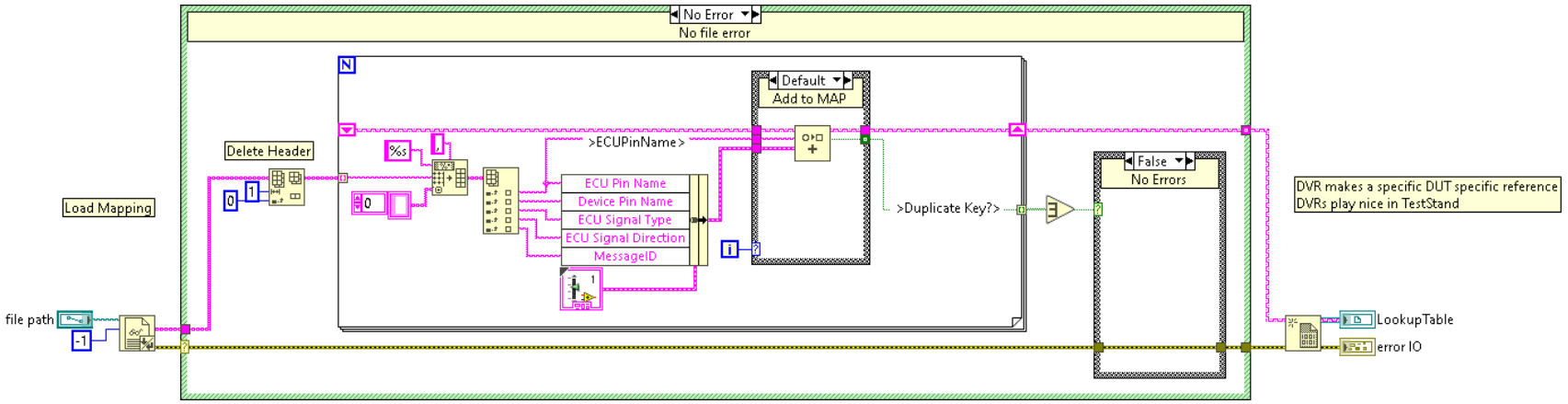
- GUI based configuration tedious for larger channel groups
- Specific to NI hardware
- Does not allow “custom parameters”
- Starts to break down in multi-up, multi-device systems

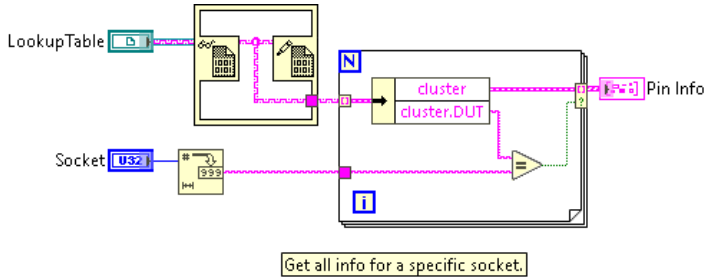
Pin Abstraction: Mapping Examples

- Mapping File & Lookup

	A	B	C	D	E	F
1	DUT	ECU Pin Name	Device Pin Name	ECU Signal Type	ECU Signal Direction	MessageID
2	0	ECU.Connector0.Pin0	Device1/AO0	Voltage	Input	0
3	0	ECU.Connector0.Pin1	Device1/AO1	Voltage	Input	1
4	0	ECU.Connector0.Pin2	Device1/AO2	Voltage	Input	2
5	0	ECU.Connector0.Pin3	Device1/AO3	Voltage	Input	3
6	0	ECU.Connector0.Pin4	Device1/AO4	Voltage	Input	4
7	0	ECU.Connector0.Pin5	Device1/AO5	Voltage	Input	5







Pin Abstraction: Mapping File & Lookup

Pros

- Better for larger channel groups
- Not specific to NI hardware
- Allows for custom parameters
- Holds up under multi-up, multi-device systems
- Allows multiple lookup options (e.g. All pins by DUT, All pins by type, etc)
- Exportable
- Allows for naming scheme

Cons

- Requires front end development investment
- Slight duplication around tasks
- More work to validate config

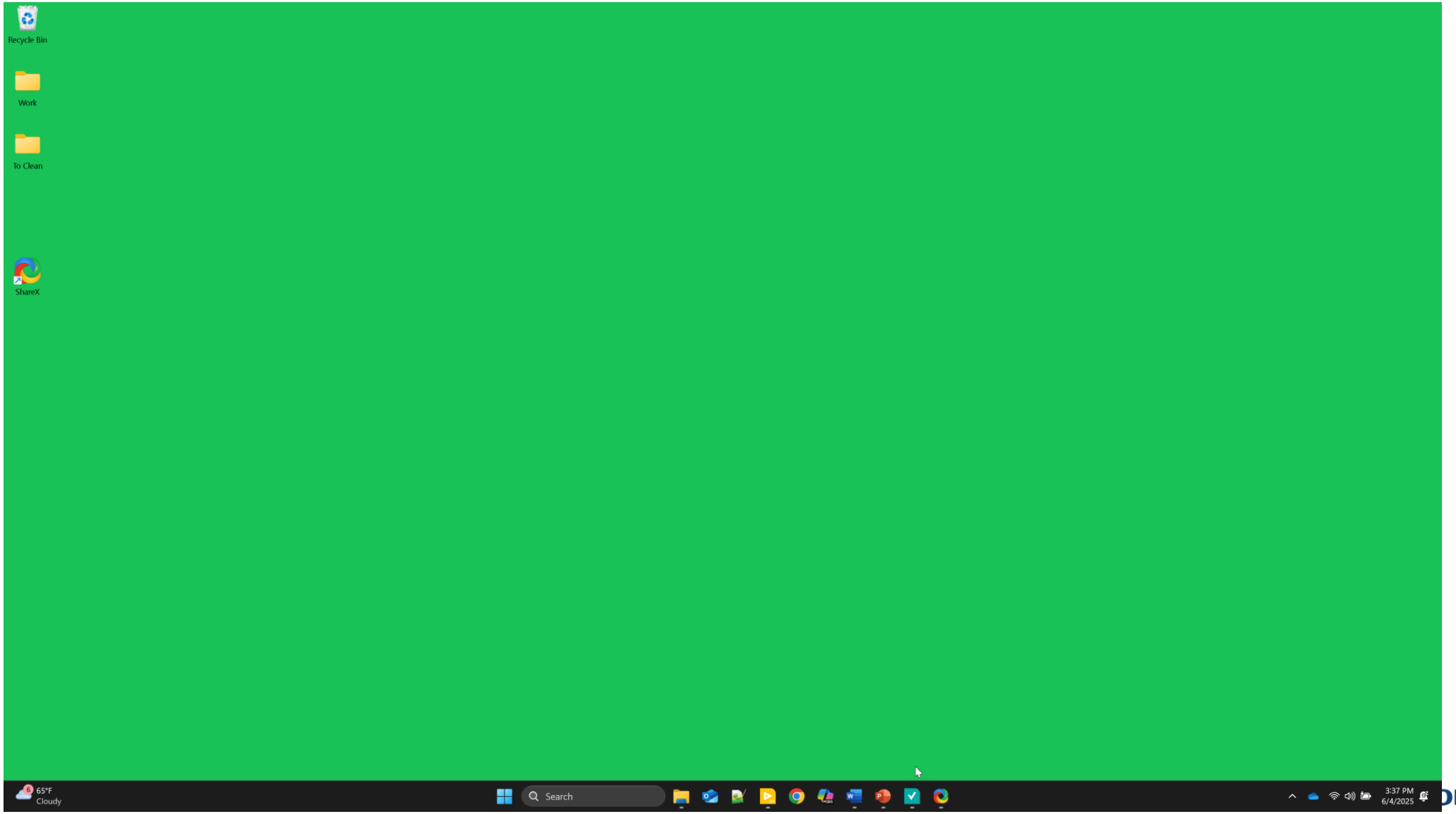
Pin Abstraction: Buses Apply Too

- CAN
- Automotive Ethernet
- Etc

	A	B	C	D	E	F
1	NAME	ECU Port	Tester Port	Baud Rate	FD Baud Rate	Termination
2	DUT0.ECU.RADAR.CAN_FD	0	CAN4	2.17433E+11	3.60453E+16	0
3	DUT0.ECU.HUD_CAN	1	CAN2	500000	0	0
4	DUT0.ECU.CAN3_SPARE	2	CAN3	500000	0	0
5	DUT0.ECU.HS_CAN_FD	3	CAN1	2.17433E+11	3.60453E+16	1

	A	B	C	D	E	F
1	NAME	IP Address	Location	Network	Rate	MAC
2	DUT0.ECU.ITM	10.10.10.101	ENET1	SW1	100	00:80:2F:30:64:11
3	DUT0.ECU.LIDAR	10.10.10.102	ENET2	SW1	1000	00:80:2F:30:64:12
4	DUT0.ECU.RADAR3	10.10.10.103	ENET3	SW1	100	00:80:2F:30:64:13
5	DUT0.ECU.RADAR2	10.10.10.104	ENET4	SW1	100	00:80:2F:30:64:14
6	DUT0.ECU.RADAR1	10.10.10.105	ENET5	SW1	100	00:80:2F:30:83:E4
7	DUT0.ECU.ECG	10.10.10.108	ENET8	SW1	100	00:80:2F:30:83:E7
8	DUT0.ECU.RADAR5	10.10.10.109	ENET9	SW1	100	00:80:2F:30:85:40
9	DUT0.ECU.RADAR4	10.10.10.110	ENET10	SW1	100	00:80:2F:30:85:41
10	DUT0.ECU.FFLIDAR	10.10.10.112	ENET12	SW1	1000	00:80:2F:30:85:43

Demo: Creating a Pin Map



Summary: Creating a Pin Map

- Used a config file
- Used a logic naming scheme that enables multi-up
- Had custom parameters
- Had multi-up, multi-device systems
- Allowed multiple lookup options (e.g. All pins by DUT, All pins by type, etc)

- Pin Abstraction: Converts rack IO → ECU Pin Measurements

- Sequences/Steps don’t care about rack configuration; makes it re-usable

EMERSON

Summary: Hardware & Pin Abstraction

- Hardware Abstraction: Provides one API for varying rack functionality
  - Use NI Drivers or write your own
- Pin Abstraction: Converts rack IO → ECU Pin Measurements
  - Use a mapping config file and look up table
- Use a logic naming scheme that enables multi-up
  - E.g. DUT#.ECU.Connector.Pin.MeasurementType
- Sequences/Steps reference ECU Pins; nice for the user
- Sequences/Steps don’t care about rack configuration; makes it re-usable

Any DUT

- Goal: One test step → Any ECU

- Built on-top of abstraction
- Configuration files
- Step parameterization
- For Loops

- At the step level
- Assumes common ECU communication approach

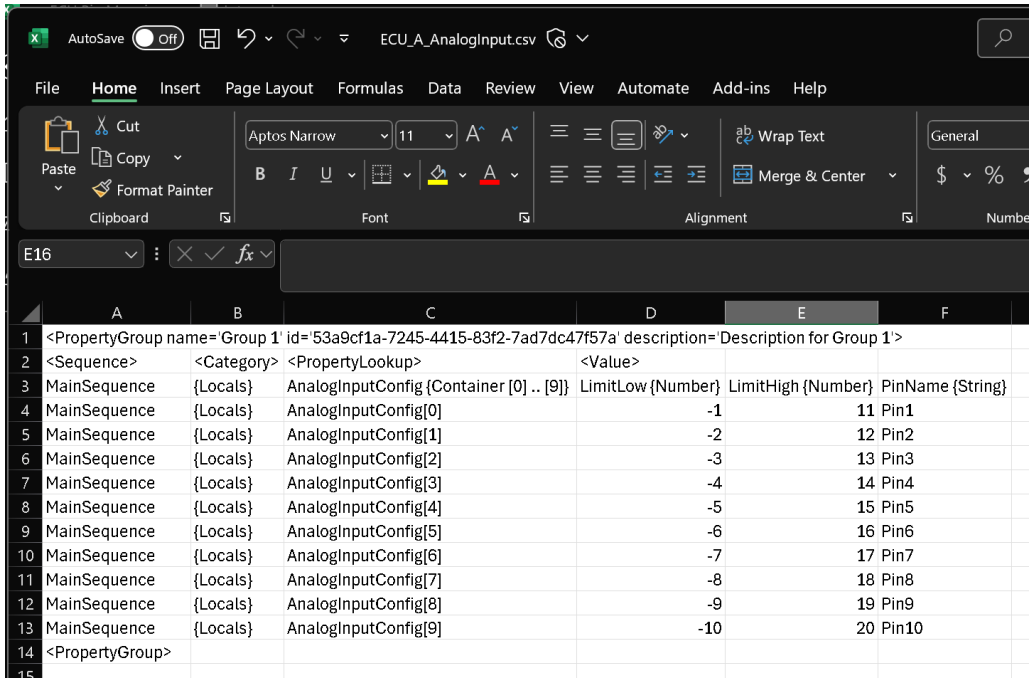
Abstraction

- See previous…
- Required to remove test rack specifics from sequences

Configuration Files

- ECU Limits
- DUT Control Messages / IDs

- Option 1: Use NI TestStand property loader
- Option 2: Use your own file format and parsing functions



	A	B	C	D	E	F
1	<PropertyGroup name='Group 1' id='53a9cf1a-7245-4415-83f2-7ad7dc47f57a' description='Description for Group 1'>					
2	<Sequence>	<Category>	<PropertyLookup>	<Value>		
3	MainSequence	{Locals}	AnalogInputConfig {Container [0] .. [9]}	LimitLow {Number}	LimitHigh {Number}	PinName {String}
4	MainSequence	{Locals}	AnalogInputConfig[0]	-1	11	Pin1
5	MainSequence	{Locals}	AnalogInputConfig[1]	-2	12	Pin2
6	MainSequence	{Locals}	AnalogInputConfig[2]	-3	13	Pin3
7	MainSequence	{Locals}	AnalogInputConfig[3]	-4	14	Pin4
8	MainSequence	{Locals}	AnalogInputConfig[4]	-5	15	Pin5
9	MainSequence	{Locals}	AnalogInputConfig[5]	-6	16	Pin6
10	MainSequence	{Locals}	AnalogInputConfig[6]	-7	17	Pin7
11	MainSequence	{Locals}	AnalogInputConfig[7]	-8	18	Pin8
12	MainSequence	{Locals}	AnalogInputConfig[8]	-9	19	Pin9
13	MainSequence	{Locals}	AnalogInputConfig[9]	-10	20	Pin10
14	<PropertyGroup>					

EMERSON

Step Parameterization

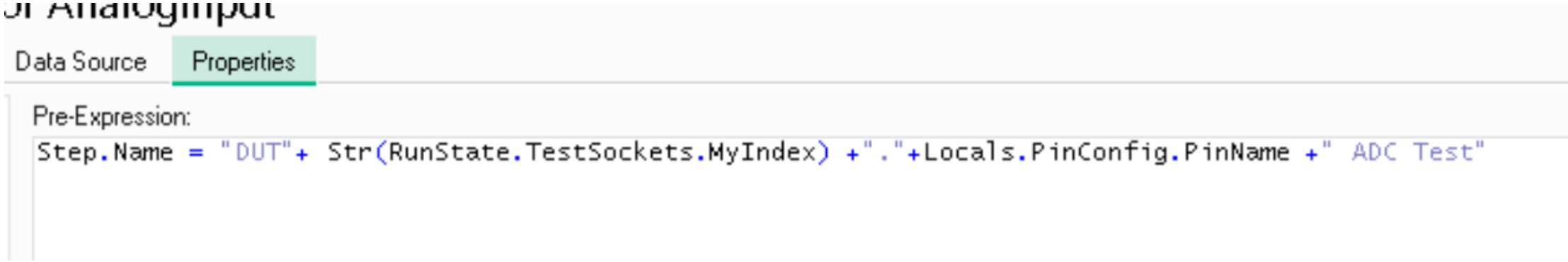
- Build step for common functions
  - Send message & measure pin
  - Generate pin & read message
  - Bus bandwidth
  - Internal chipset check
  - Typical chipsets
  - etc
- Create parameters for test specific settings
  - Pins
  - Messages
  - Limits

For Loops

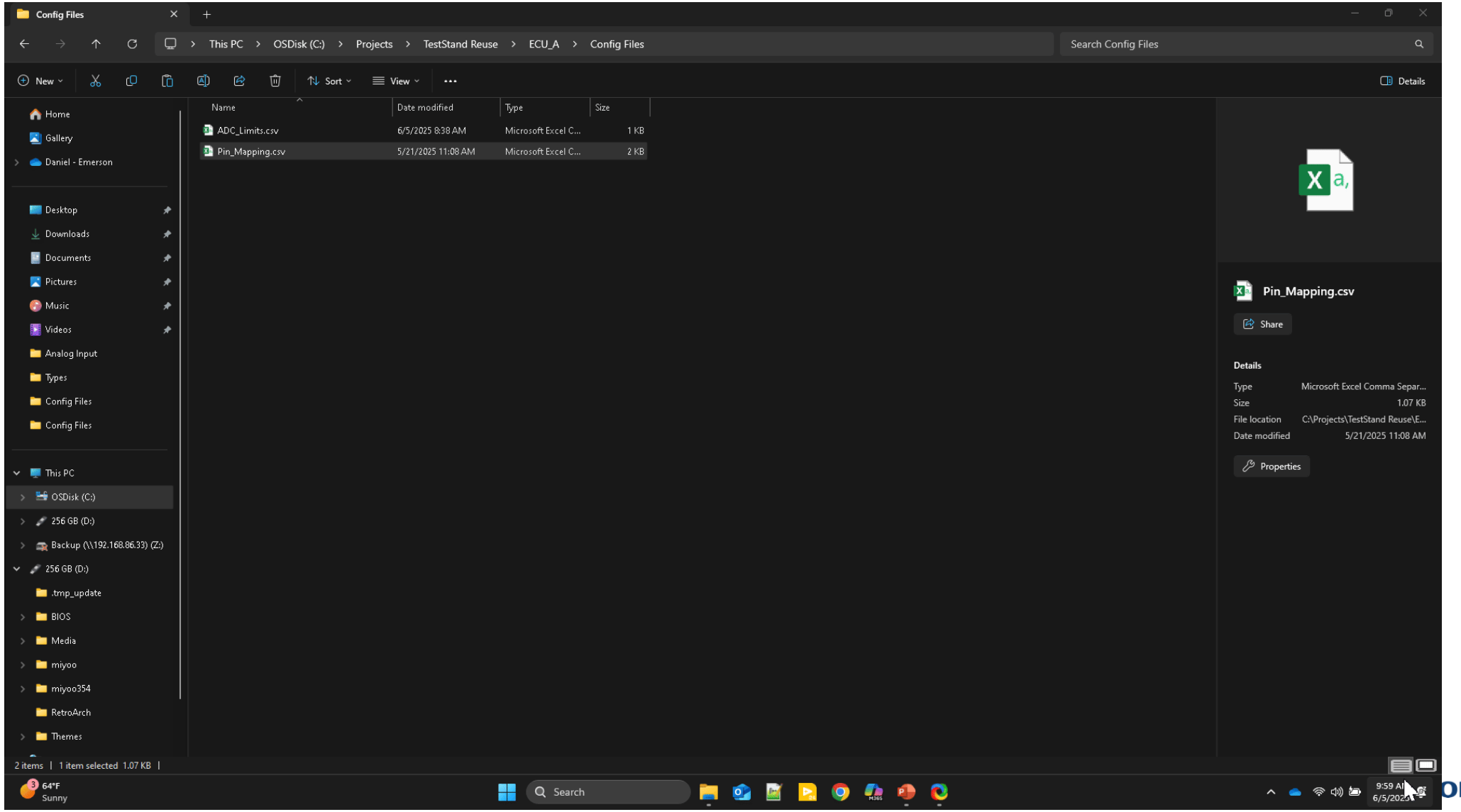
- Abstracts the number of iterations
  - Number of pins
  - Number of tests
- Arrays are your friends
- Pass array element to local variable
- Capture iteration

Pro Tips

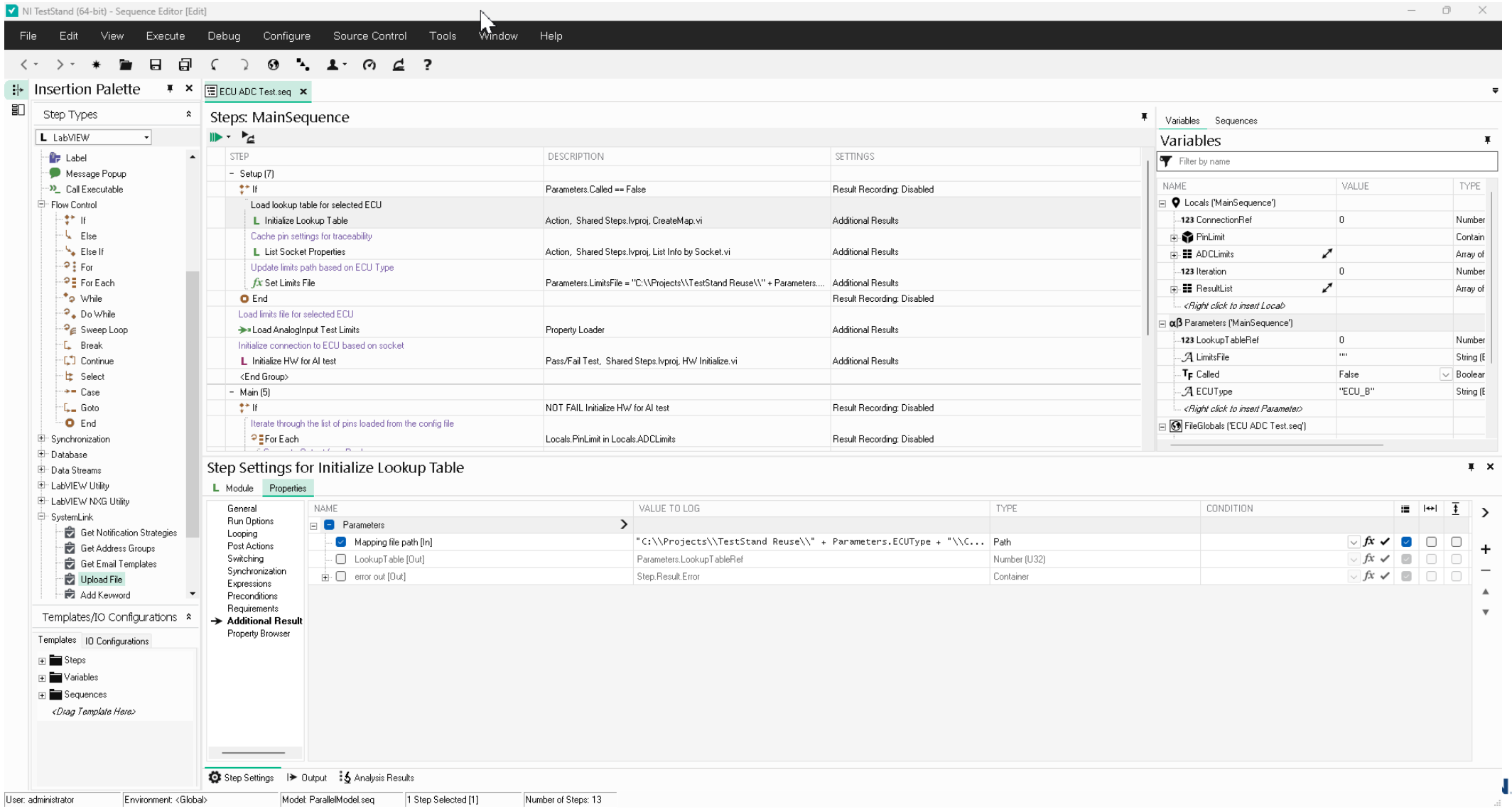
- Change Step Name at Runtime:



DEMO – Different Config & Limits Files



DEMO – Same TestStand Sequence



Summary: Different Configurations; Same Sequence

- Used pin mapping config file to abstract rack specifics
- Used limits config file to abstract away ECU specifics
- Used For Loop to abstract number of pins / tests
- Used ECU name to parameterize path, step names, etc
- Used pin name to parameterize step names
- Used pin name as cross reference between files

- Result: Different ECU & rack configurations → Same TestStand Sequence

Hardware Racks for Any DUT

- Typical: Rack designed per DUT
- Abstraction allows software tests to be re-used
- Hardware can be pulled out and re-used program to program (if allowed)
  - Typically requires additional hardware based on IO count/type

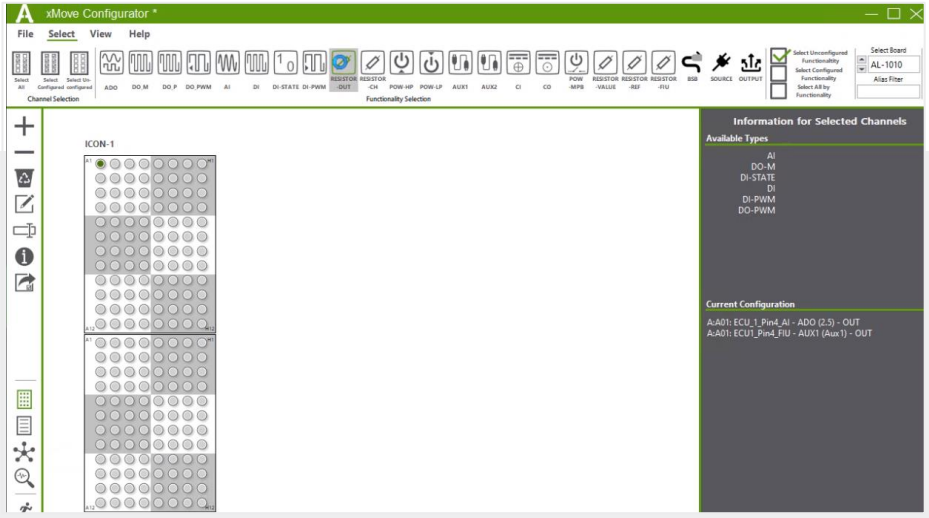
- Aliaro created software-defined hardware that enables direct program to program reuse

Flexible

NI / Aliaro Approach

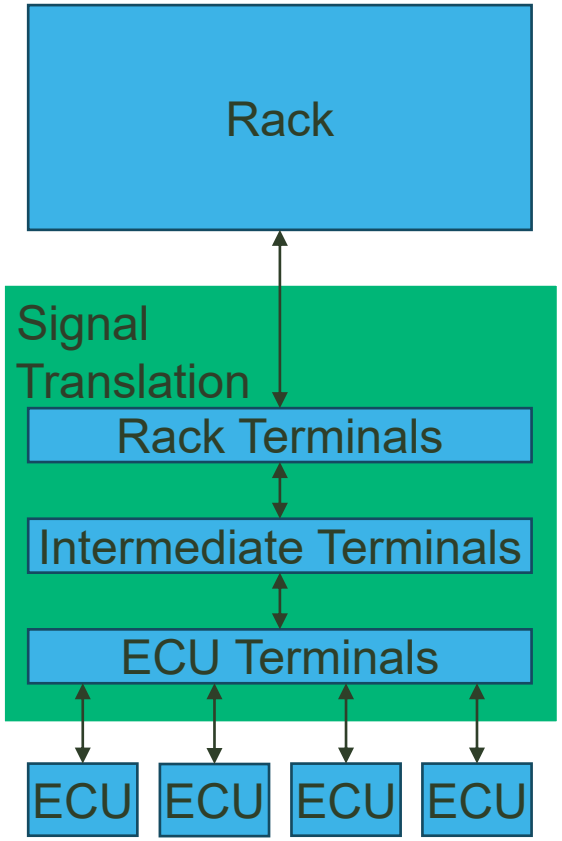
Truly Flexible Pins (Oversimplified)

- Software Selectable w/ appropriate protections
- Can be input, output or both (allows loop back)
  - Closest competition is in or out, not both
- FIU on every pin
- 12 & 32 channel cards based on requirements
  - Closest competition is 10 channels per card

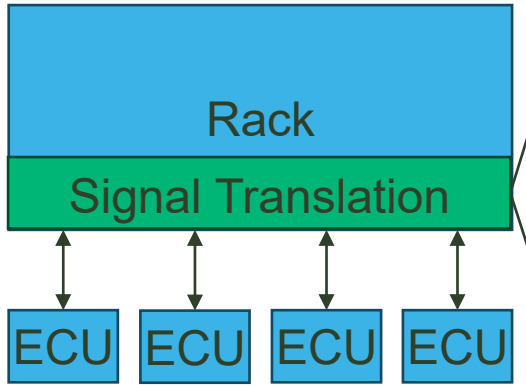


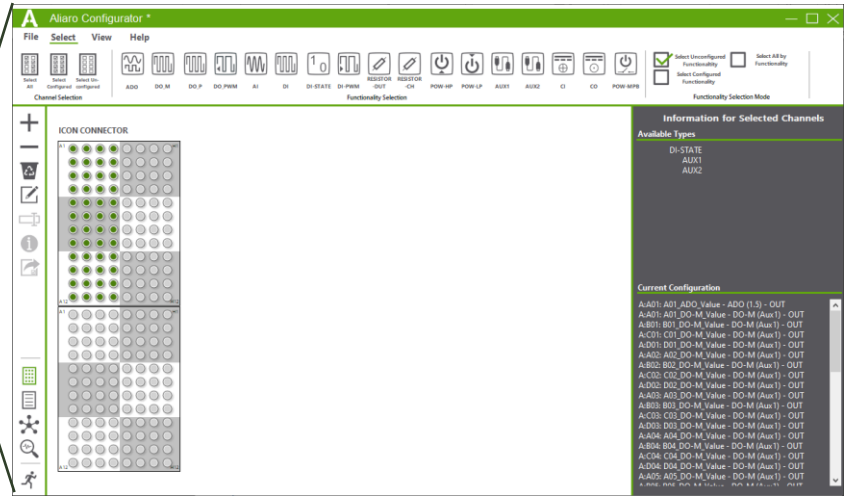
Reduce Commissioning (time to test)

Current State:



Future State:





- Software defined wiring

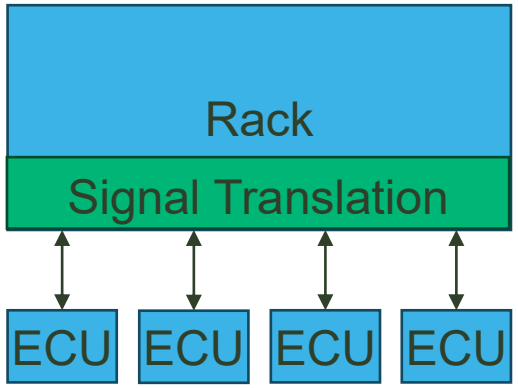
- Reduce wiring 3x to 4x
  (and everything associated with it)
- Frees weeks in project plan
- Frees resources to write scripts

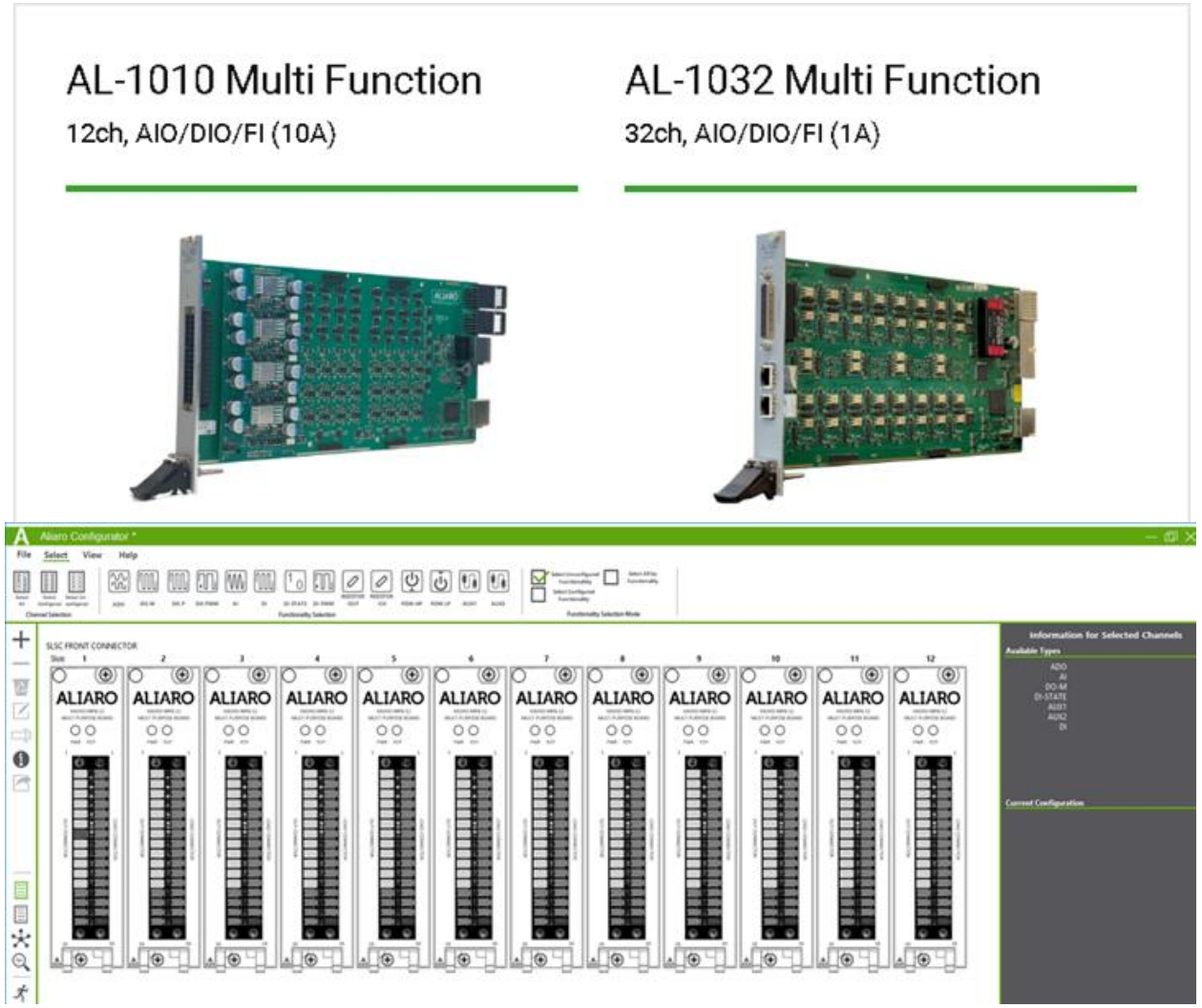
How: Aliaro Software Configurable Pins

Aliaro 1010 & 1032

- AIO, DIO, PWM, FIU **per pin**
- Software configurable

Connect, Configure, Next





Any Number of DUTs

- Goal: One Sequence → Any number of DUTS

- Hardware Design
- Design code for multiple callers
- TestStand Batch or Parallel Mode

Multi-up: Hardware Design

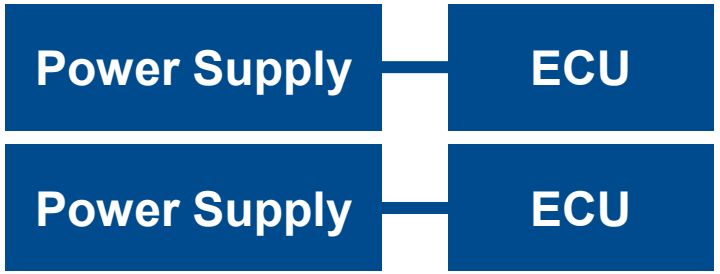
- Ideal: Grouped, Independent Hardware set per DUT
  - Will cost more in hardware
  - Will reduce software development costs
  - Will reduce test times; no shared resources

Example:

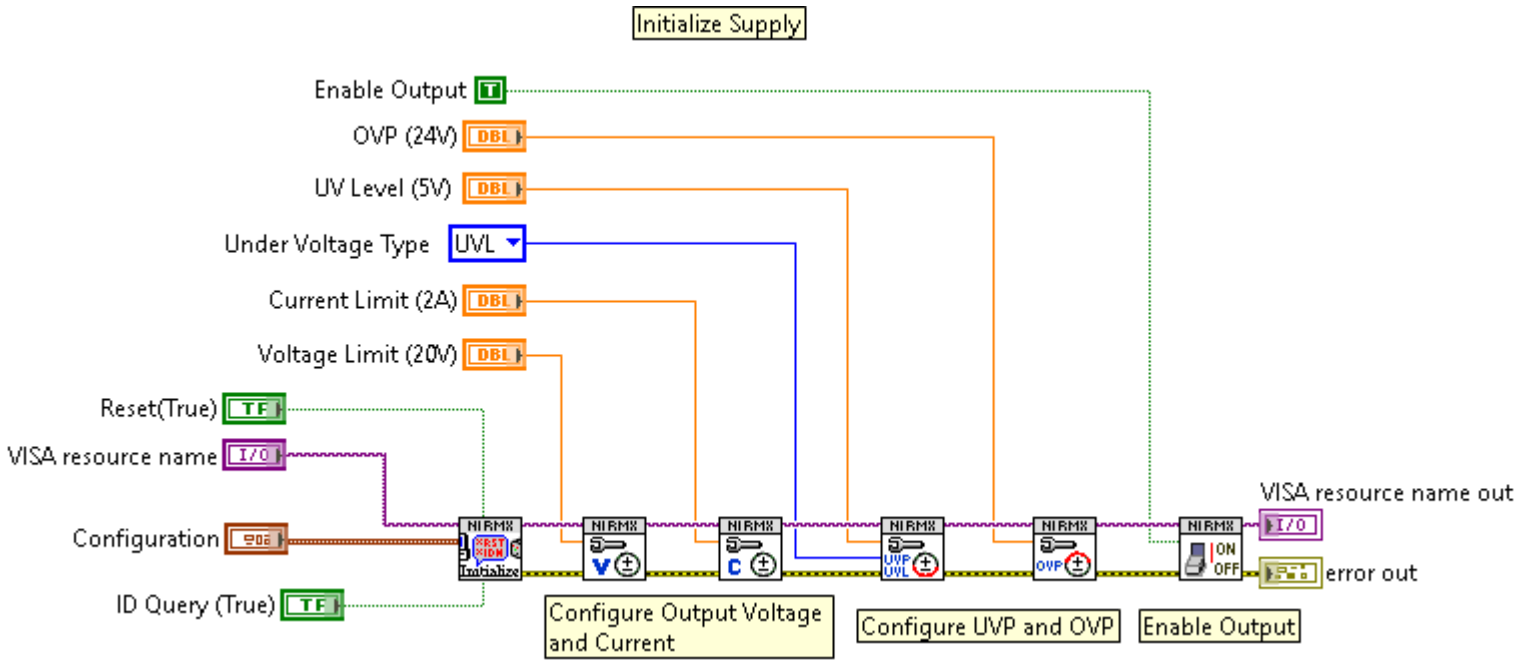
- Power supply per DUT
- Shared power supply for all DUTs

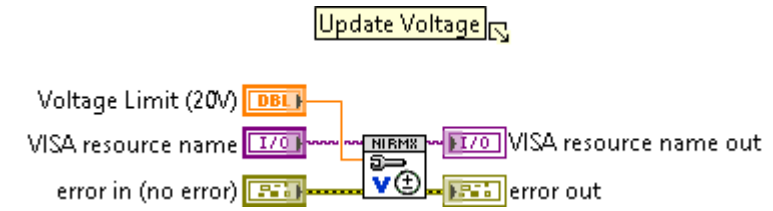
Power Supply Example: Independent (Ideal)

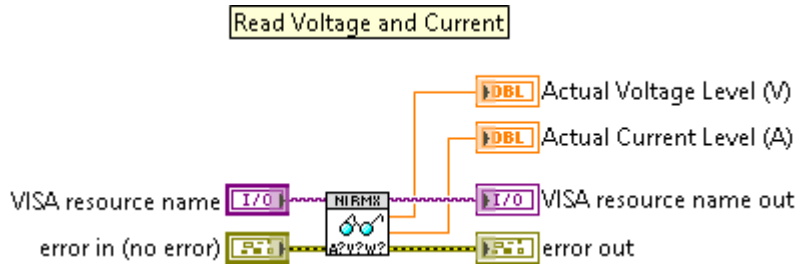
- Power supply per DUT (+ cost)
- Power supply provides per DUT:
  - Enable relay
  - Current Sense
  - Voltage Sense
  - Drop compensation

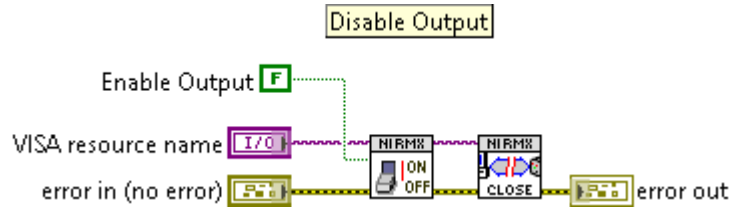


Power Supply Example: Independent (Ideal)









Power Supply Example: Shared (Not Ideal)

Shared power supply (- cost)

Bigger power supply (+ cost)

Relays (+ cost)

Relay Control (+ cost)

Individual current sensors (+ cost)

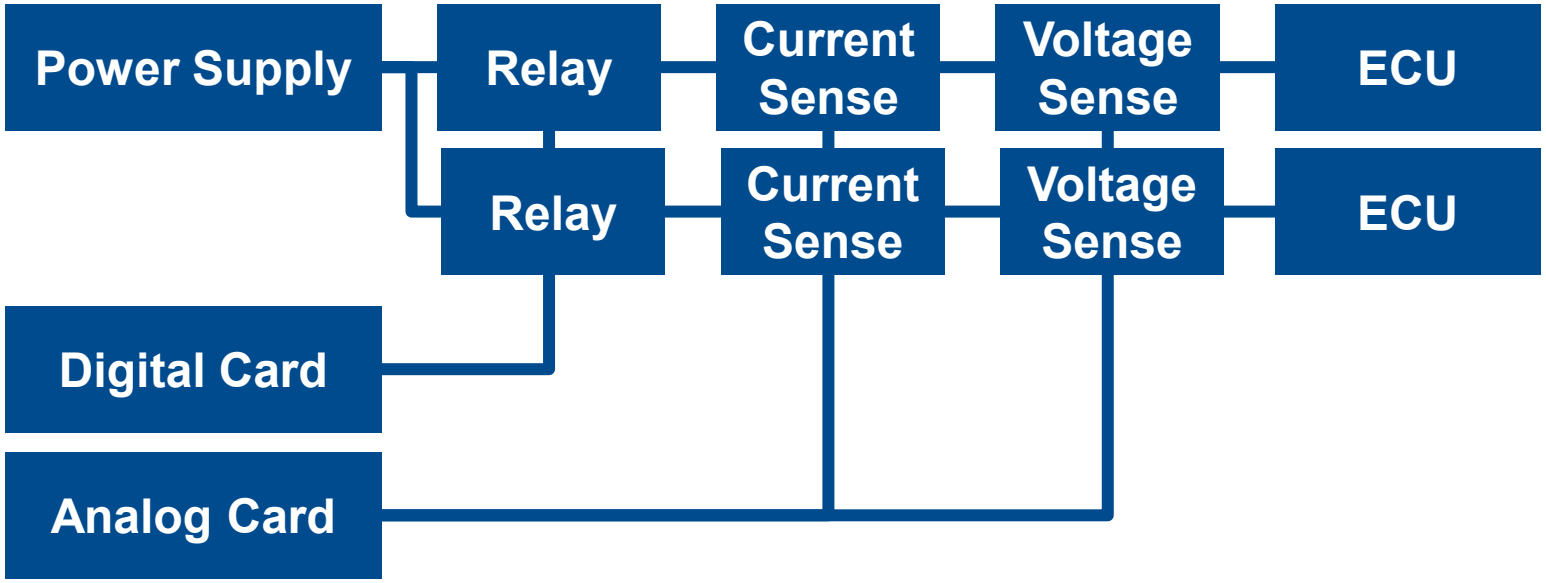
DAQ to measure current/voltage (+ cost)

Additional wiring (+ cost)

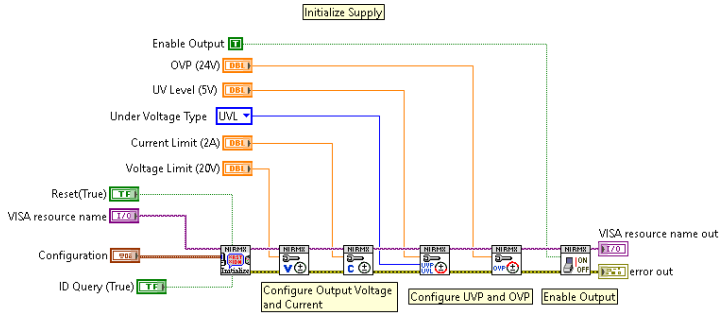
Requires significant software (+ cost)

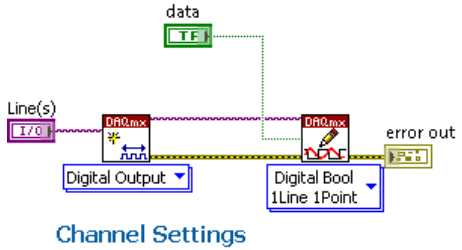
Can’t adjust voltage per DUT

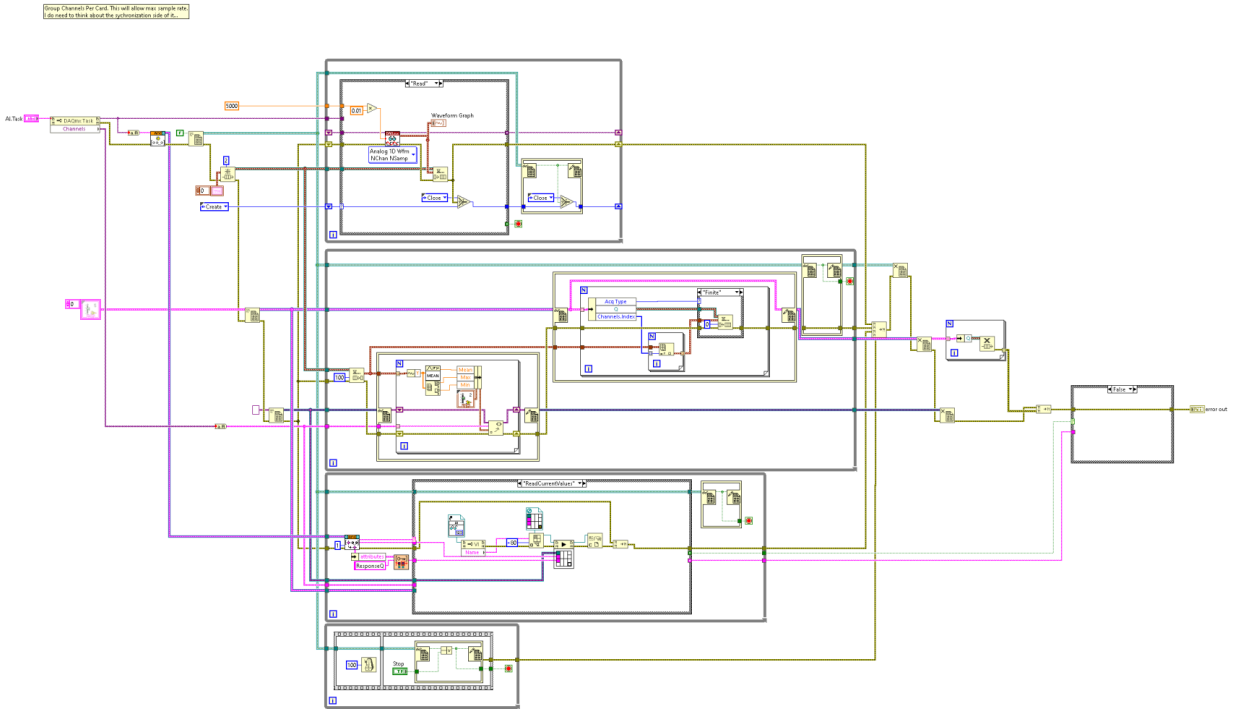
No drop compensation

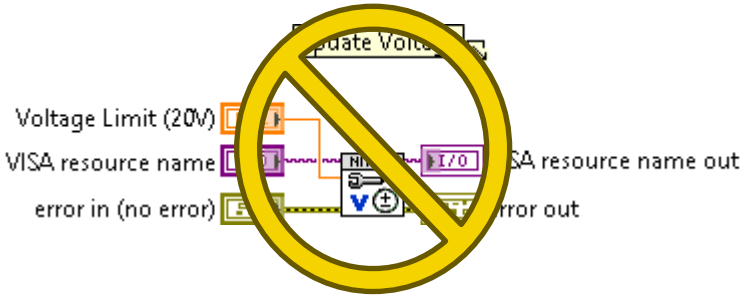


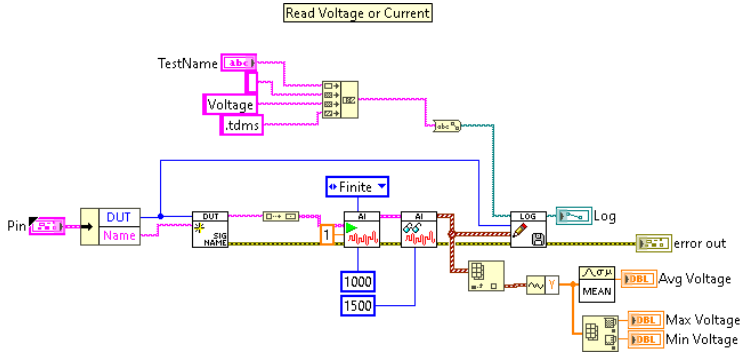
Power Supply Example: Shared (Not Ideal)

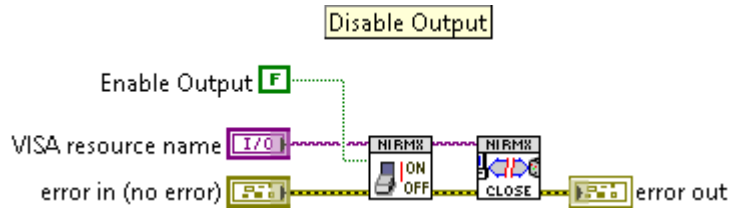






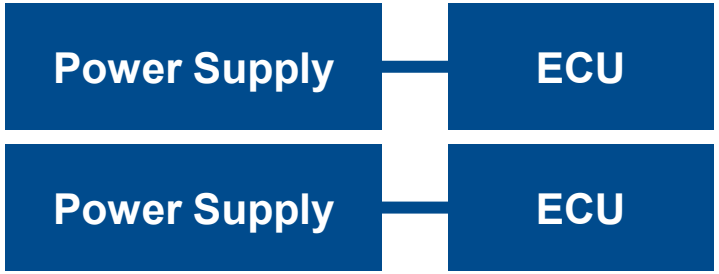


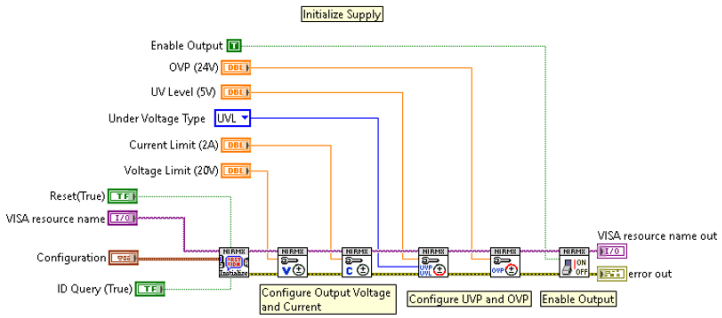


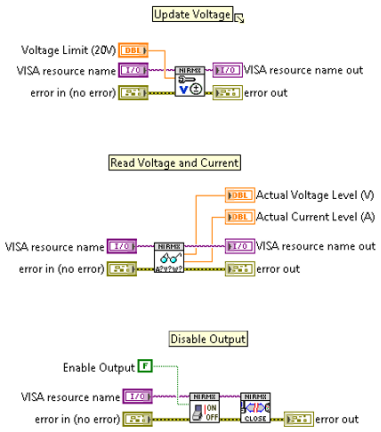


Compared: Cost – Up Front Vs Total System

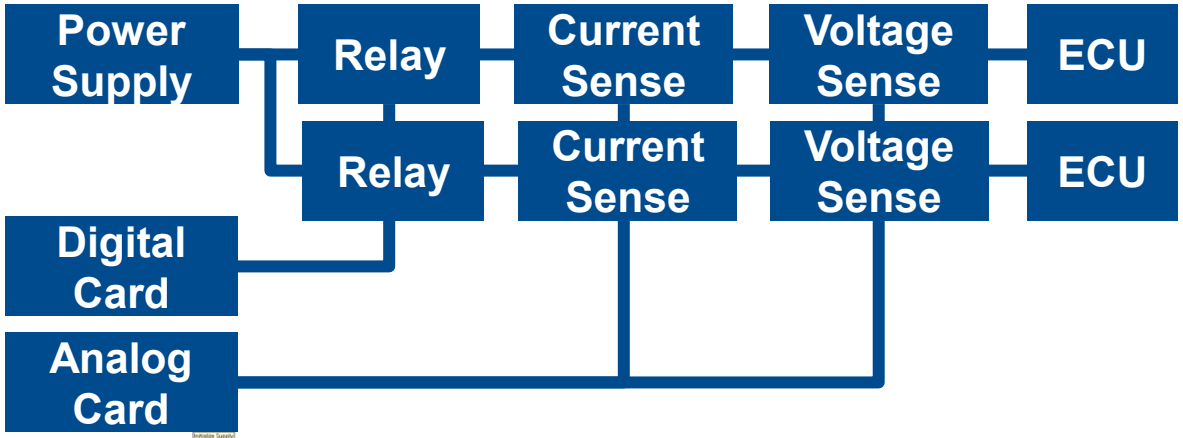
Upfront: $$; System: $$$

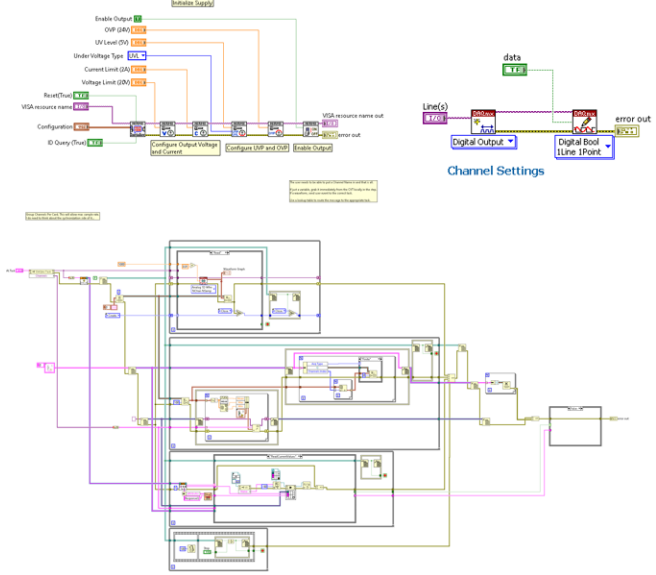


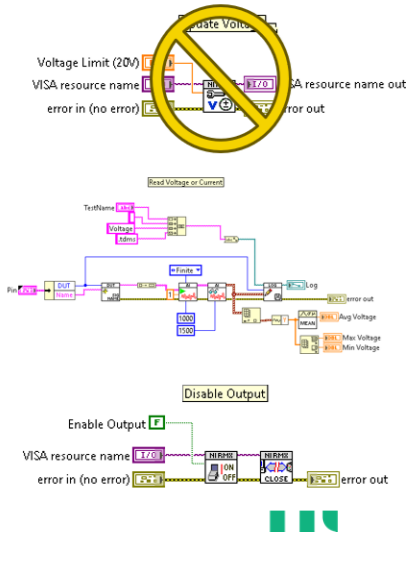




Upfront: $; System: $$$$$

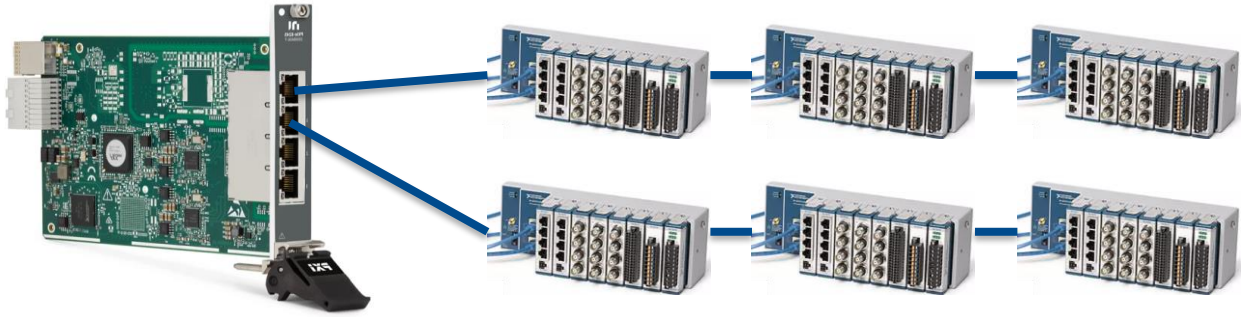




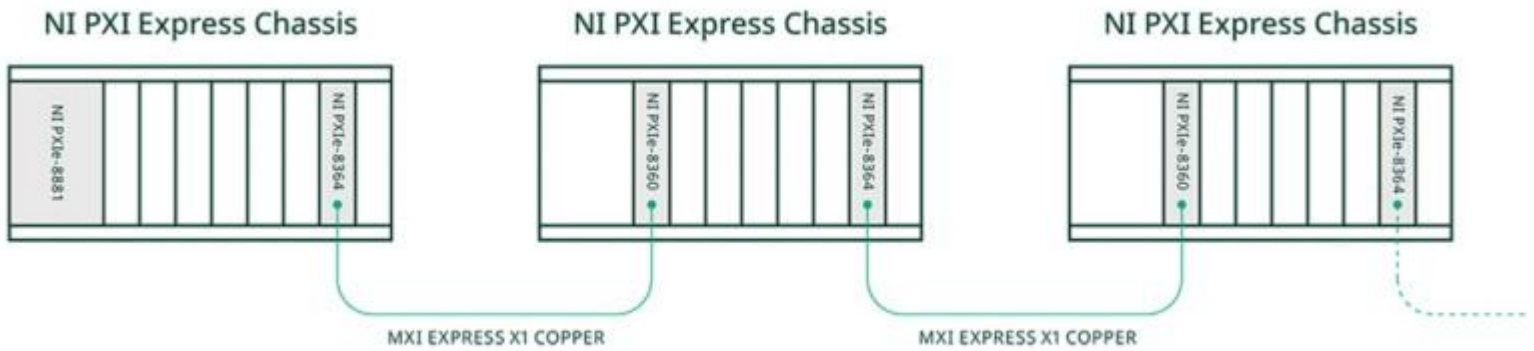


Multi-up: Design Hardware

- cDAQ Ethernet String per DUT

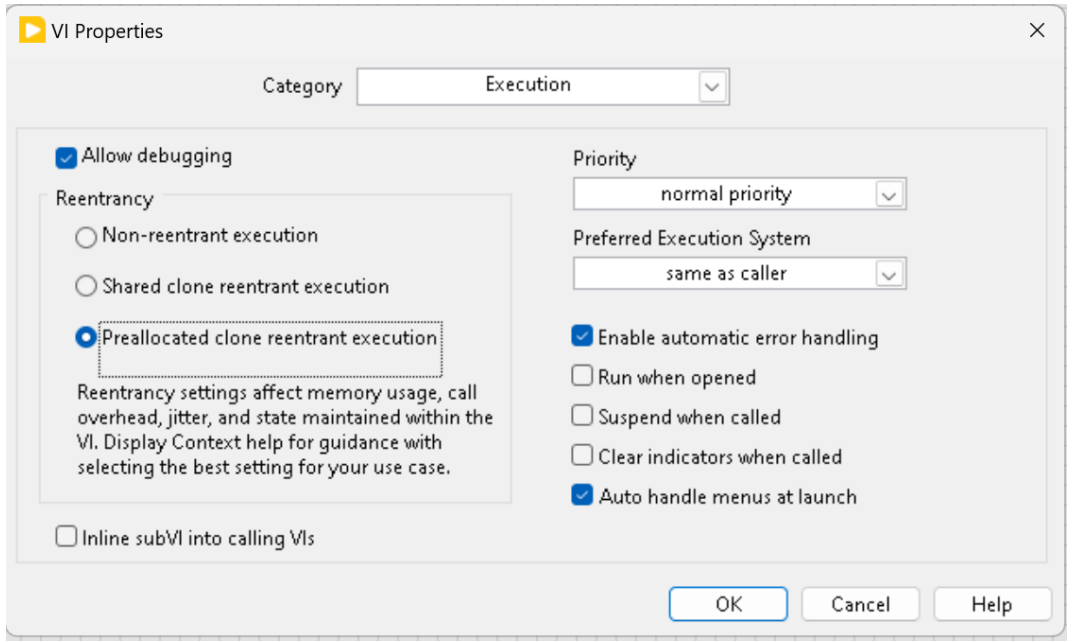


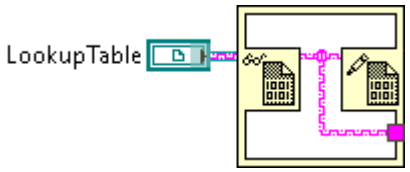
- PXI Chassis Expansion

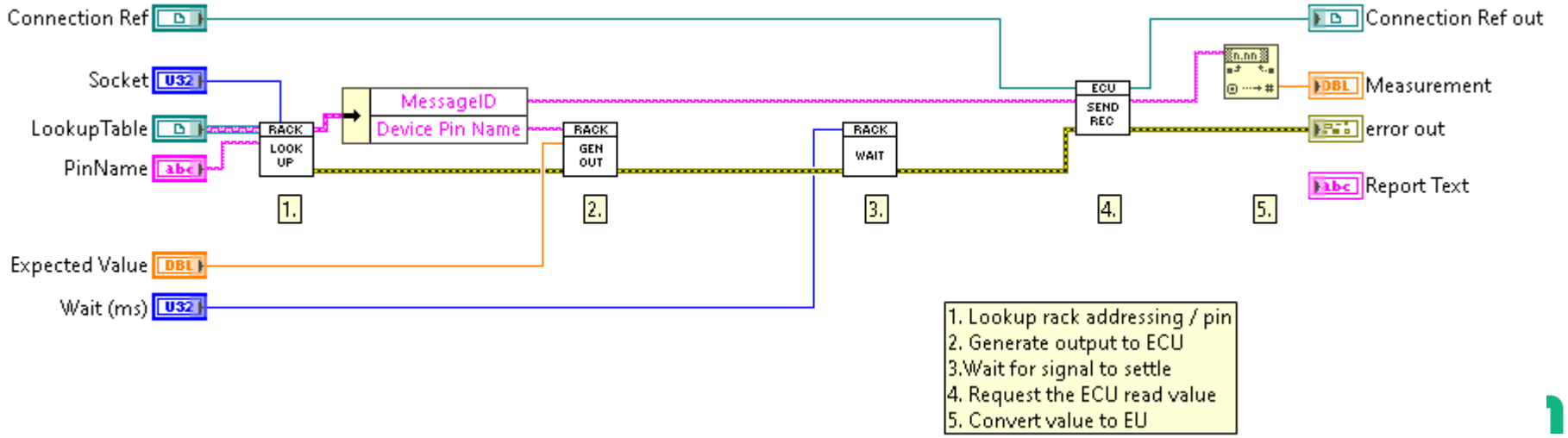


Multi-up: Software Considerations

- Re-entrant (single source file → each call is uniquely instantiated)
- Separate memory per DUT
  - Use DVRs or named queues, not (functional) globals
  - Pass unique references
- Abstract shared resources with async engine and APIs
- Parameterize functions with Socket ID

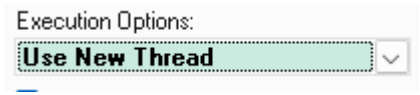


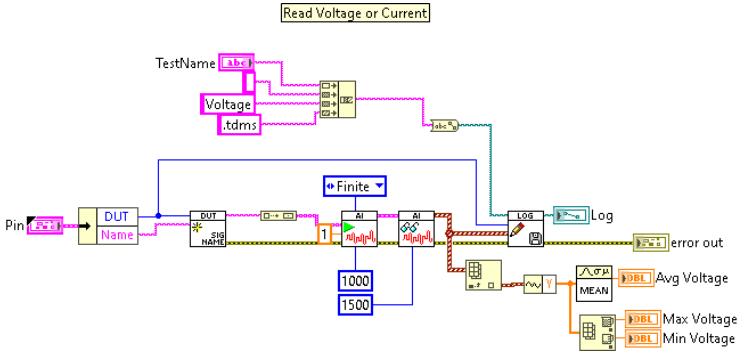


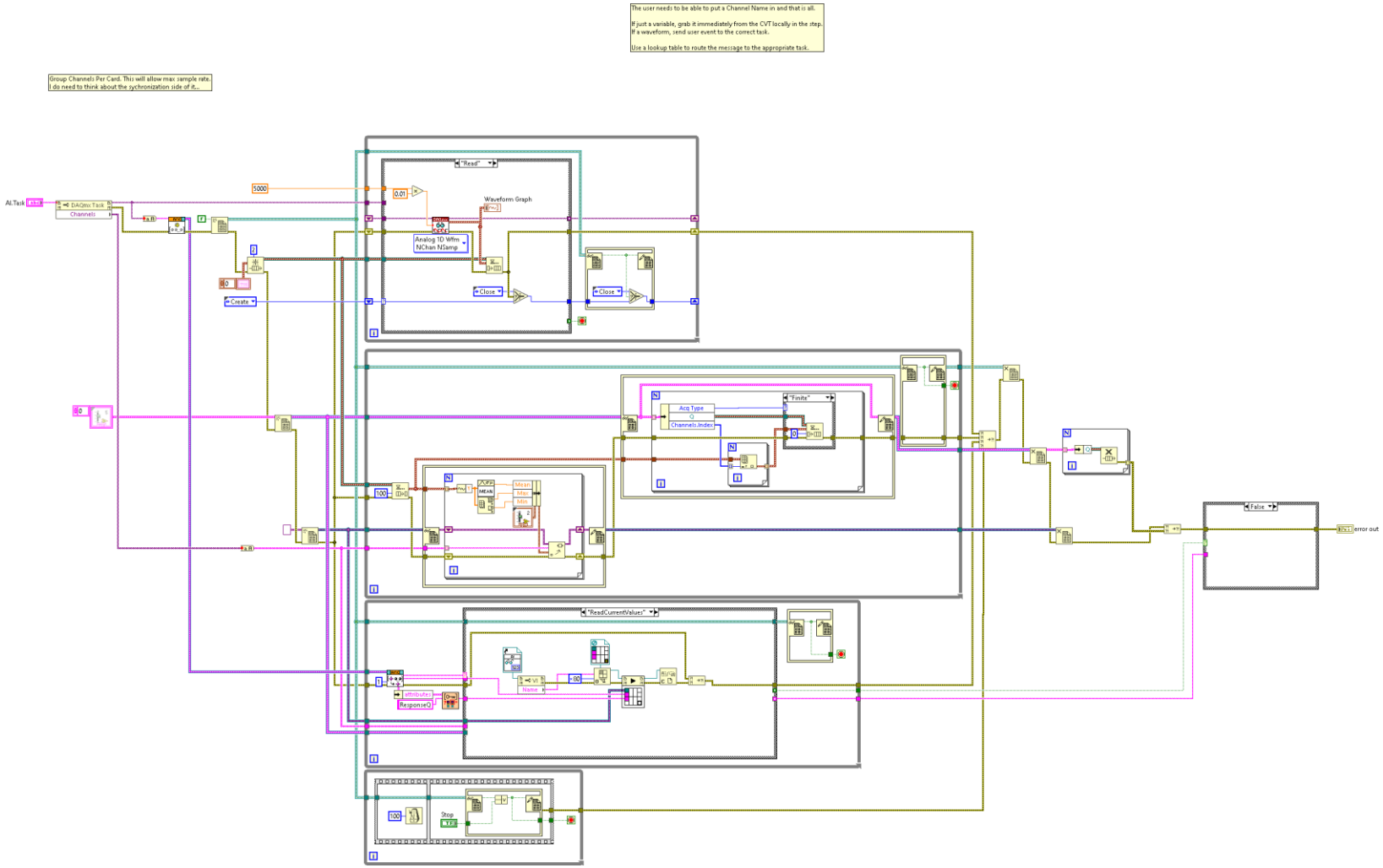


Multi-up: Shared Resources

- Build asynchronous main engine
- Launch as new thread
- Serve data to test steps via API



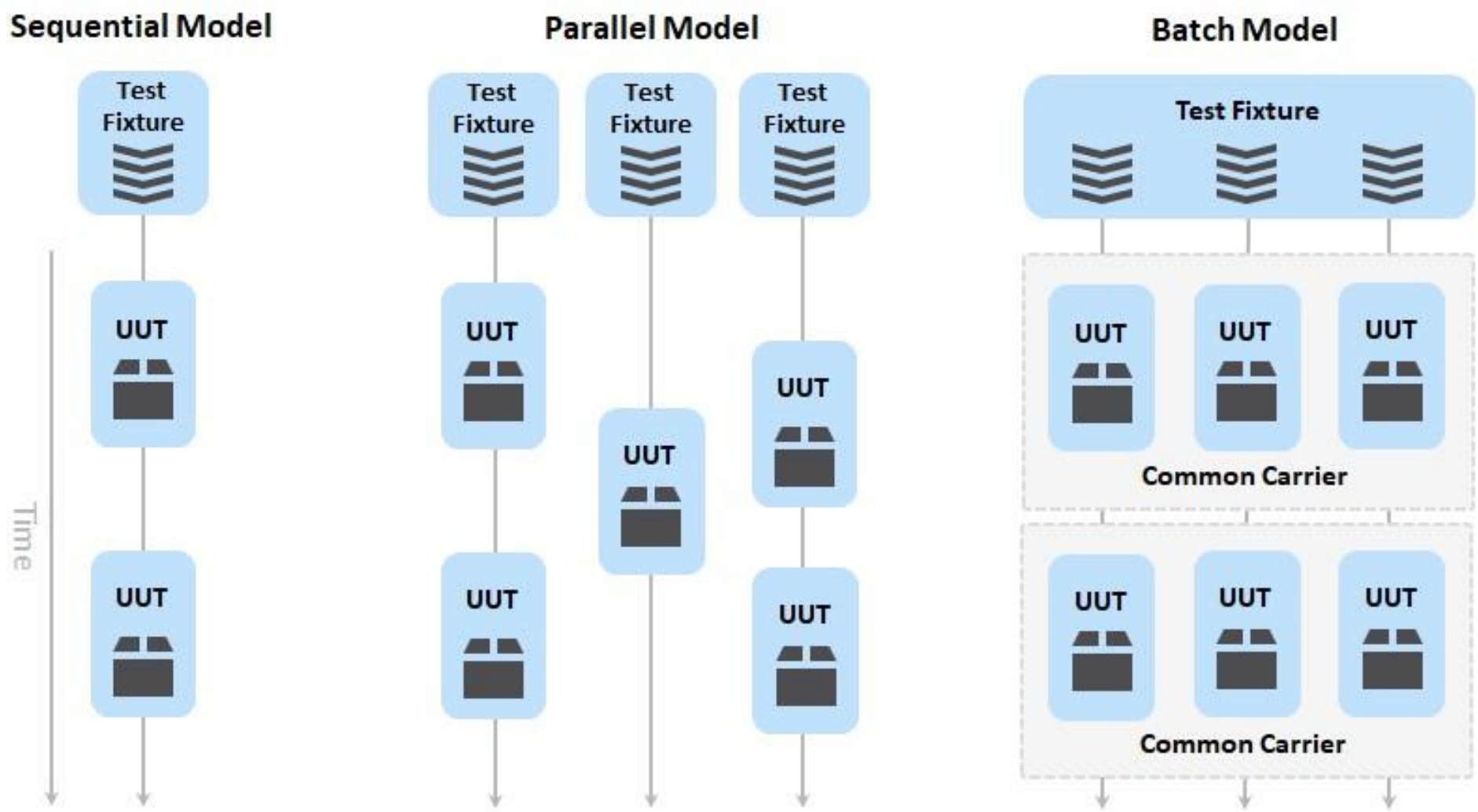




Multi-up: TestStand Process Model

- Set Process Model: Sequential vs Batch vs Parallel Mode
- Set Number of Sockets
- Get Socket ID

TestStand Process Model Options



TestStand Process Model Options

Sequential

- One DUT and one fixture

Batch

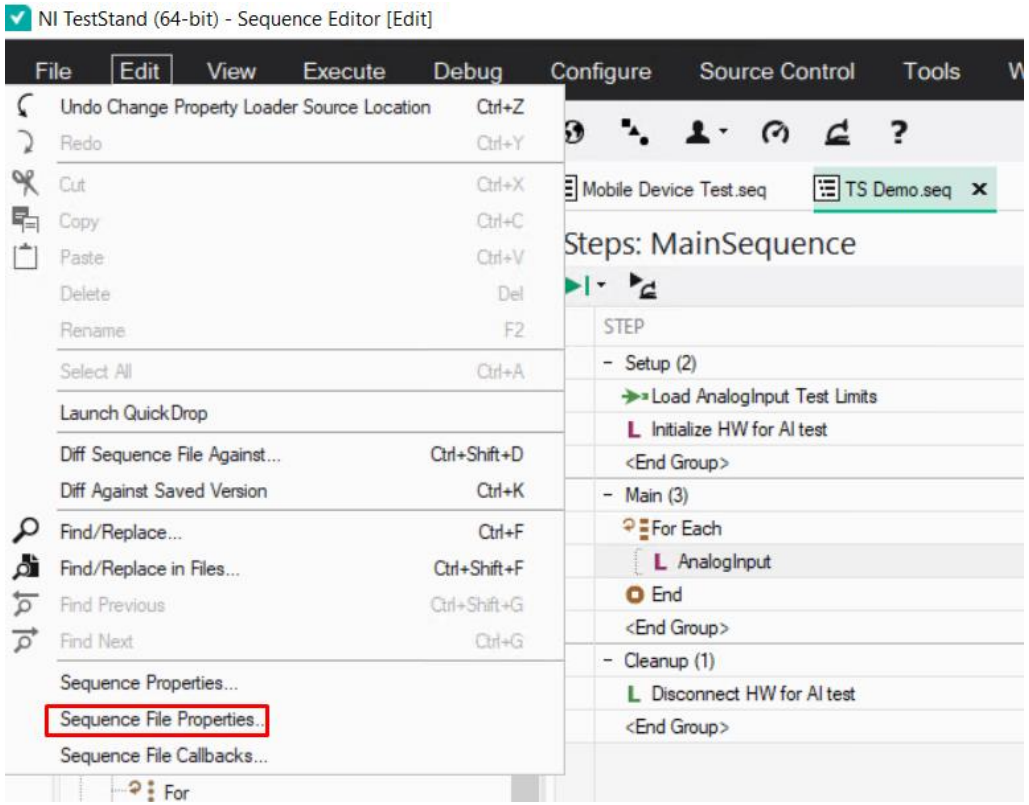
- Multiple DUTs and multiple fixtures
- Acts together
- DUT fails → Abort all
- Does not require hardware independence

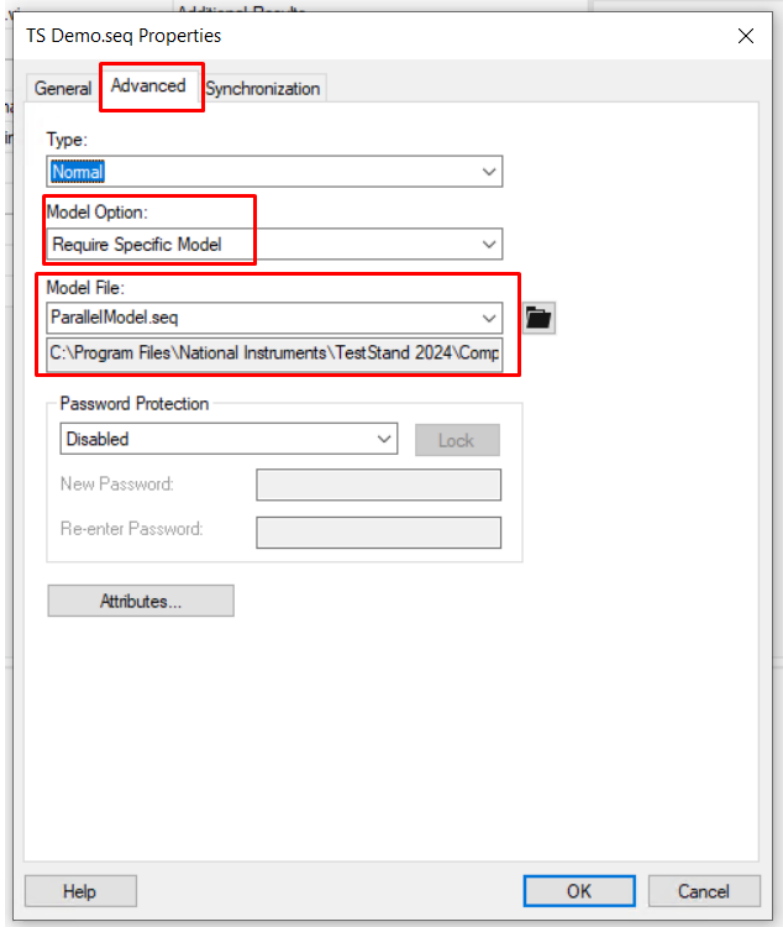
Parallel

- Multiple DUTs and multiple fixtures
- Acts independently
- DUT fails → Abort failed DUT
- Requires hardware independence

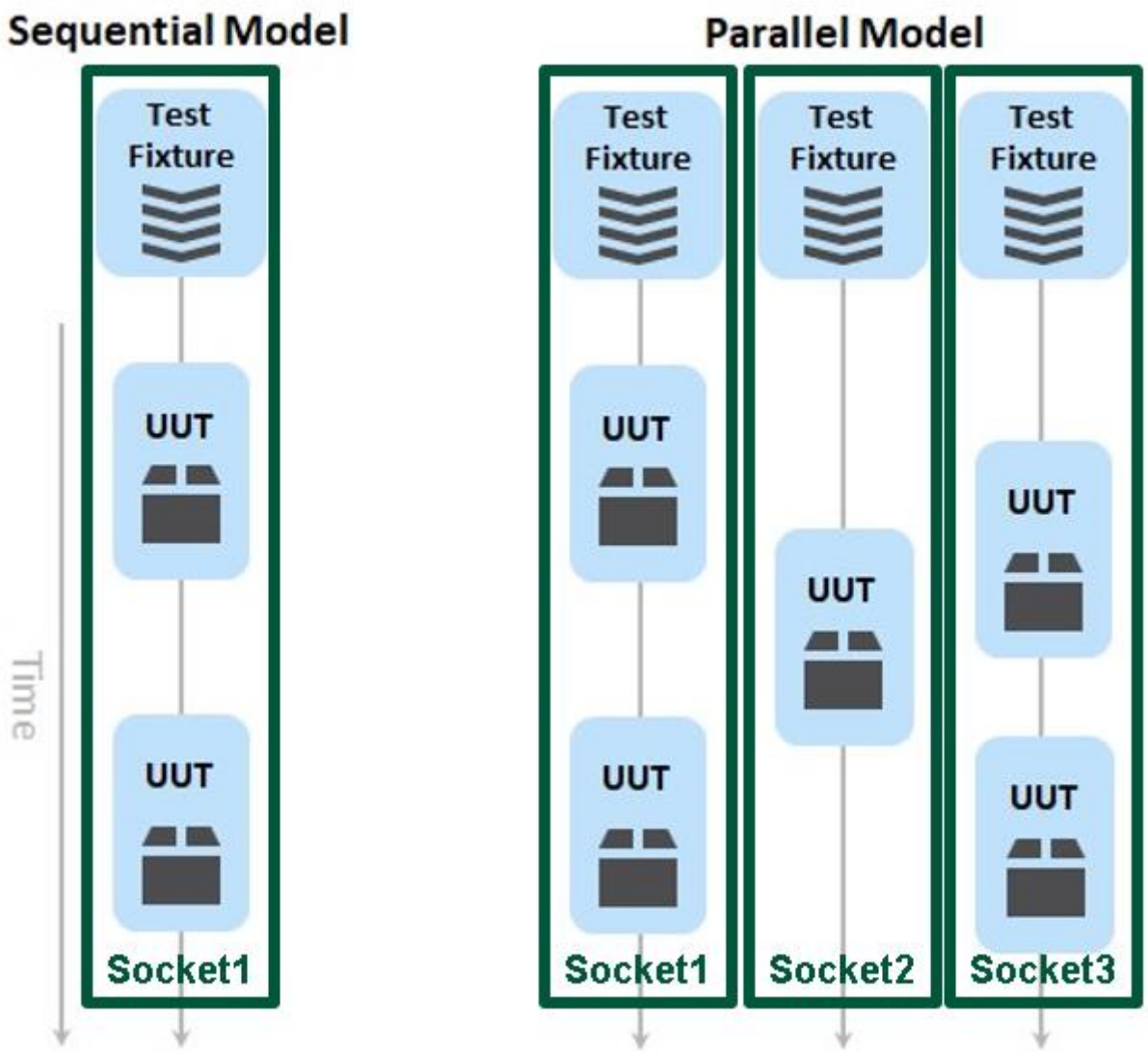
Specify a Process Model for TestStand Sequence File

- https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z000000P7NrSAK&l=en-US



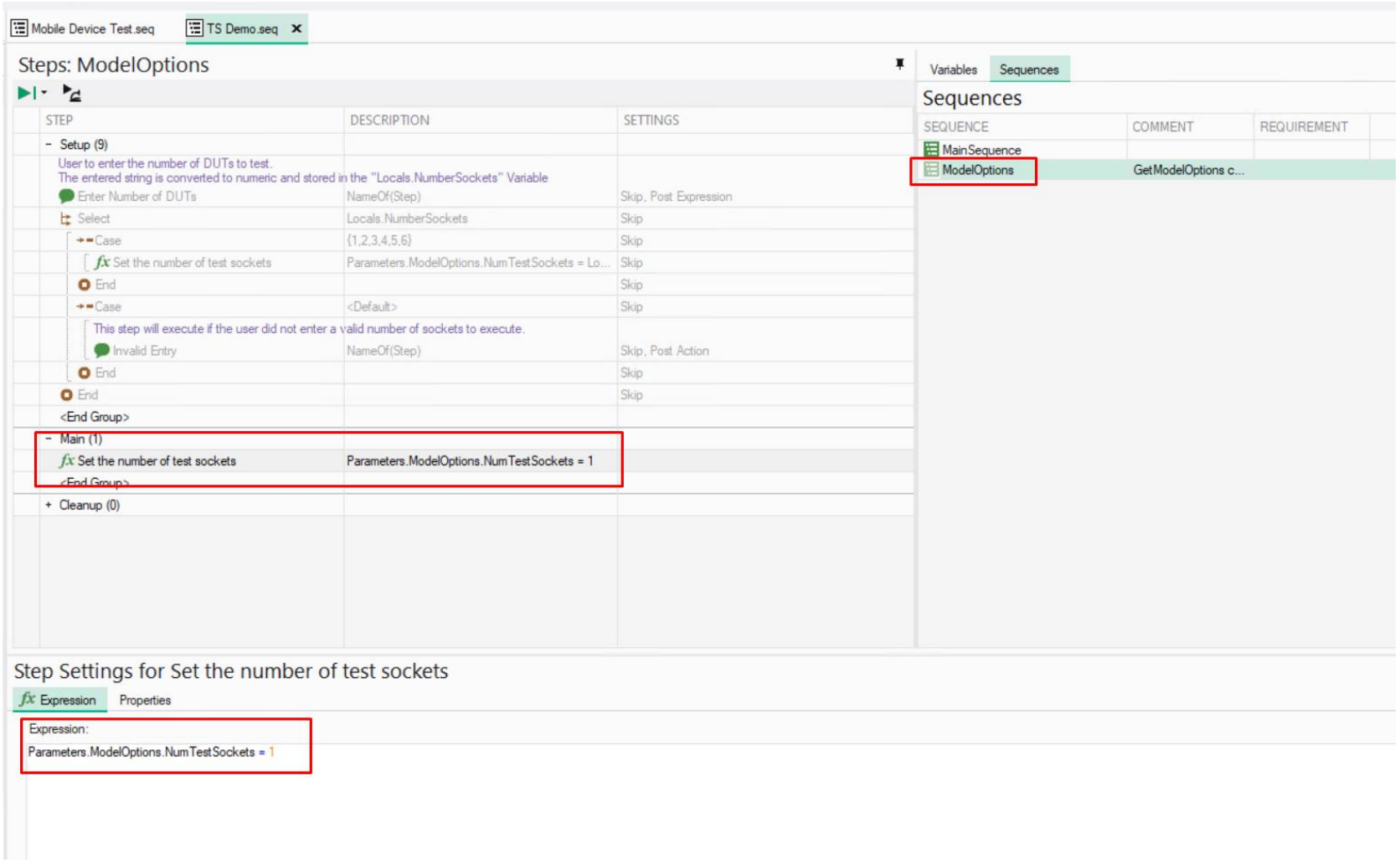


TestStand Sockets



Programmatically Set the Number of Test Sockets in TestStand

- https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z000000P7p7SAC&l=en-US

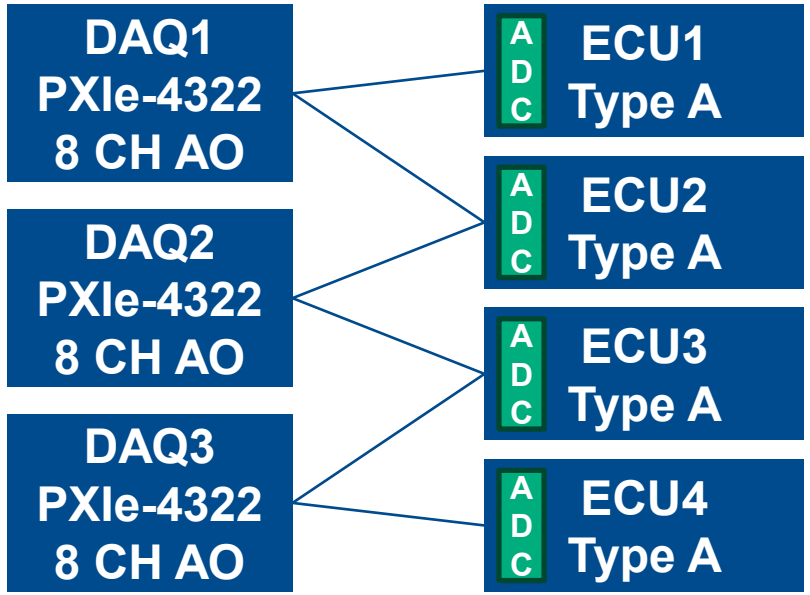


Programmatically Get Socket ID

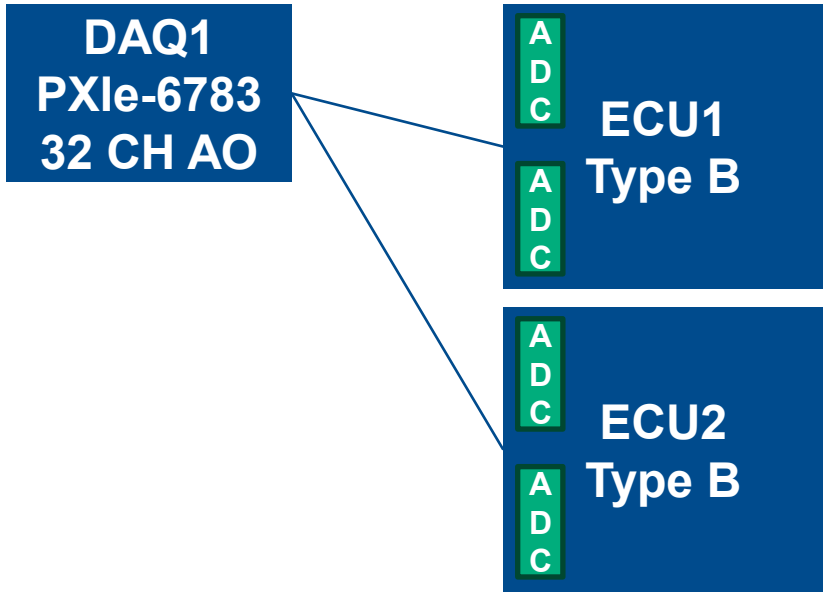
- Use TestStand Variable: RunState.TestSockets.MyIndex

PARAMETER NAME	TYPE	IN/OUT	LOG	DEFAULT	VALUE
Socket	Number (U32)	in	☐	☐	RunState.TestSockets.MyIndex

Demo: Putting it all together



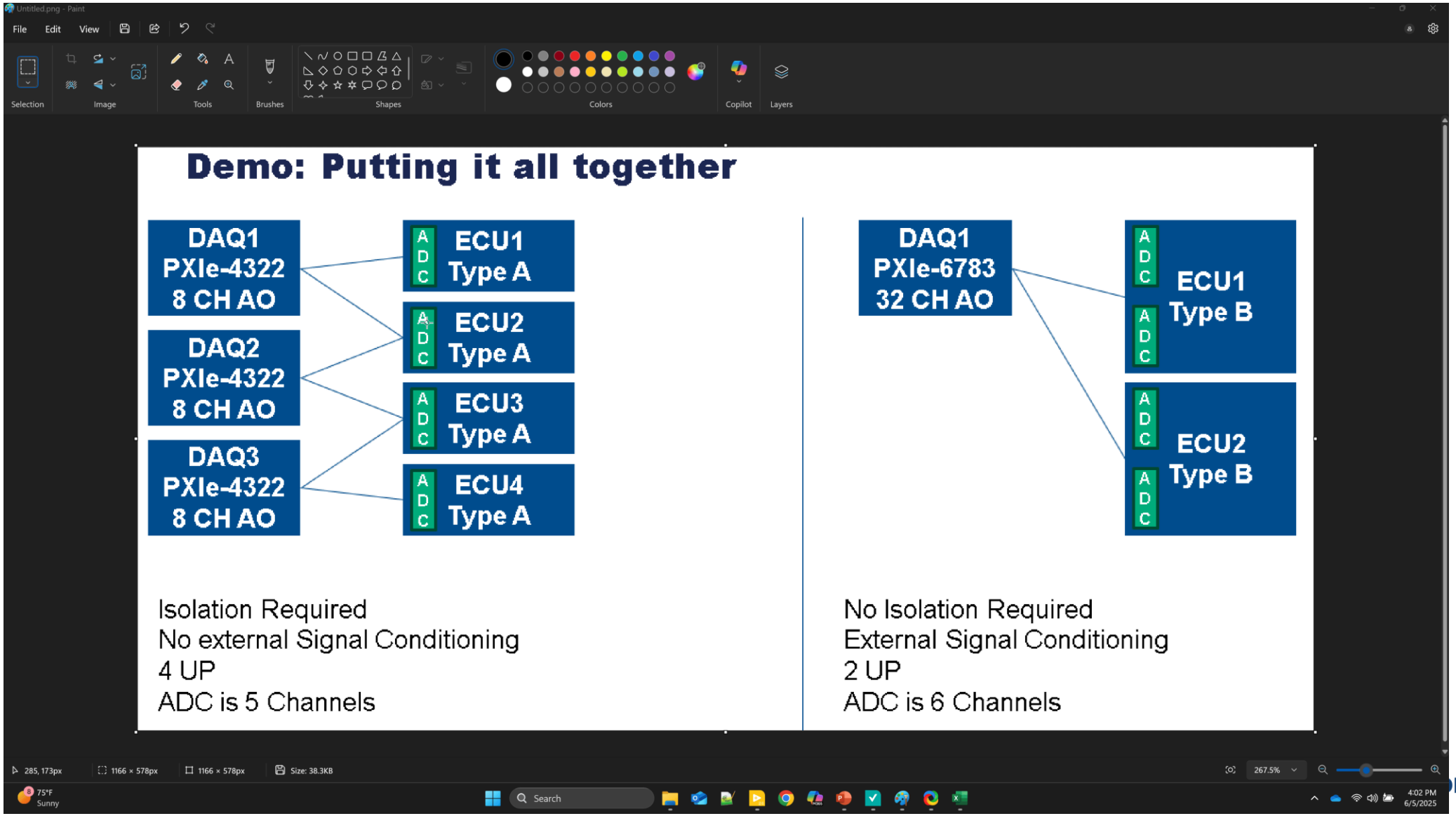
Isolation Required
No external Signal Conditioning
4 UP
ADC is 5 Channels



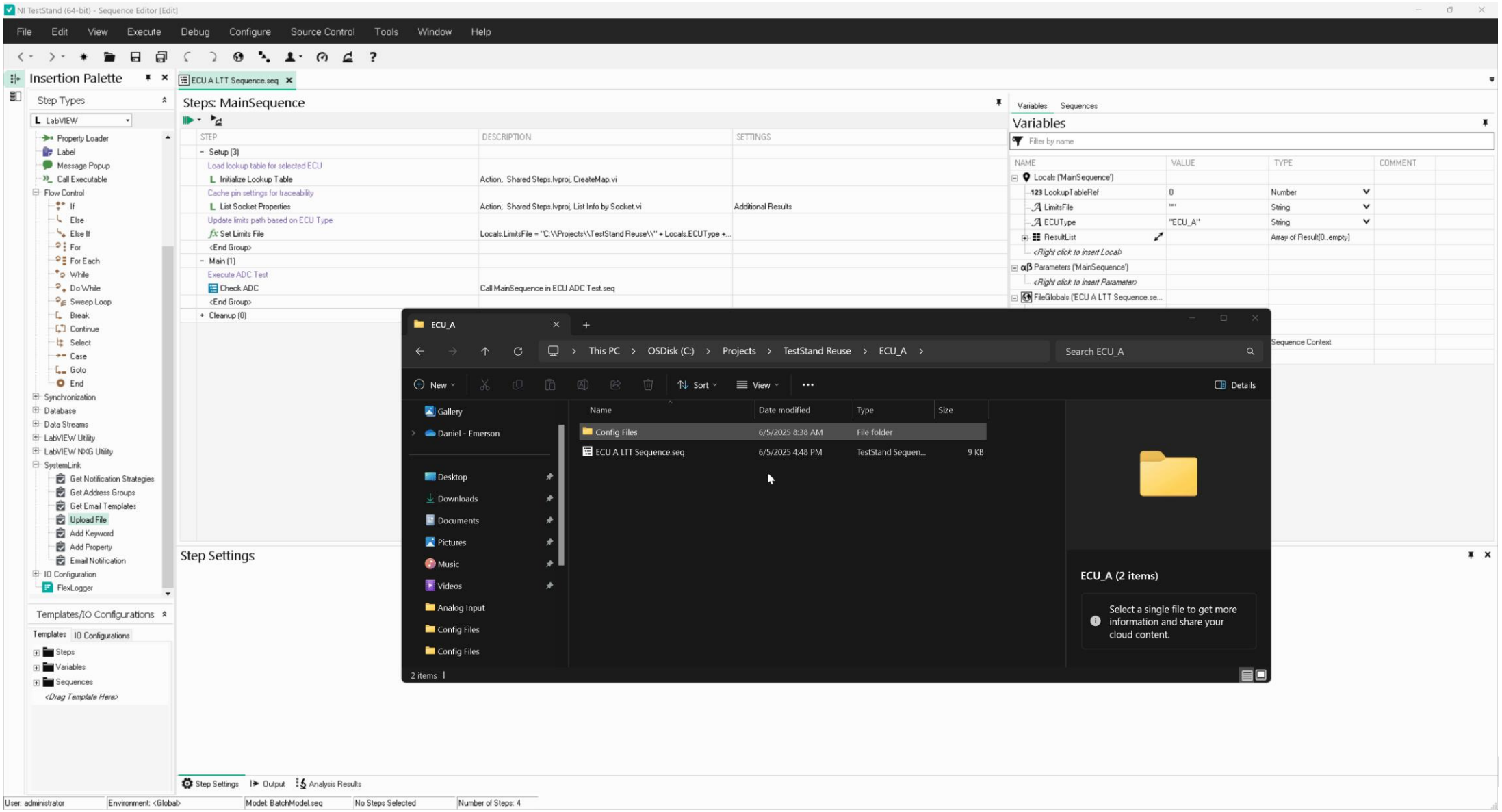
No Isolation Required
External Signal Conditioning
2 UP
ADC is 6 Channels

EMERSON

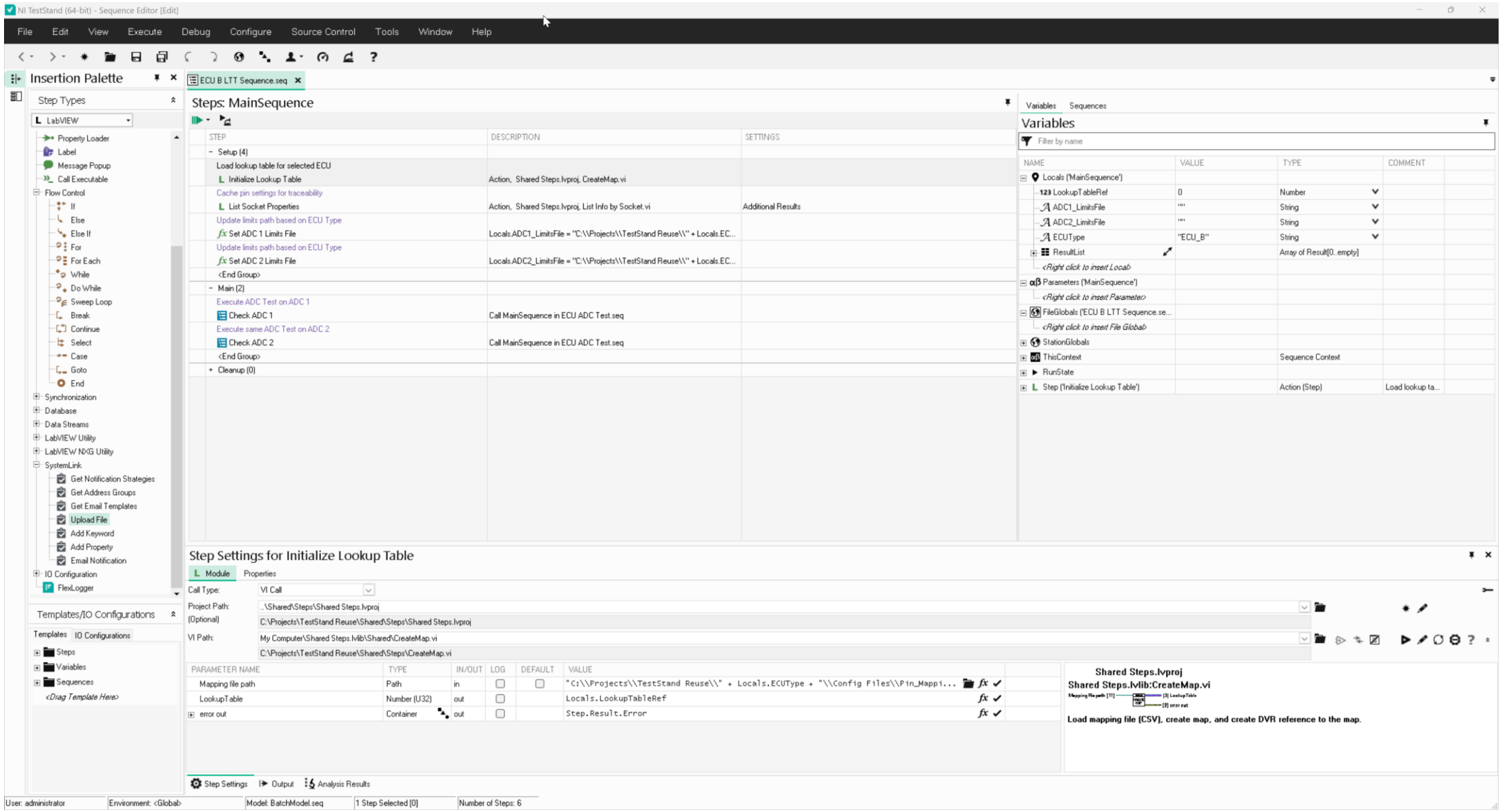
Setup: Putting it all together



Demo: ECU A – Putting it together



Demo: ECU B – Putting it together



Summary: Putting it all together

- Used DAQmx for HAL
- Used file mapping for pin abstraction & command IDs per ECU Type
- Config file for limits per ECU type
- ECU Type sets number of sockets
- Executing in Batch mode
- Single Sequence for ADC test
  - Worked for both ECUs
  - Worked for one or any number of ADCs
  - ADC test step renamed each iteration to specific pin under test

North Star: Save That Money…

- By maximizing test steps / sequences re-use for:
  - Any Rack → Abstraction
  - Any DUT → Hardware and Test Software Design
  - Any number of DUTs → Test Sequencer Design

Two Paths to Success

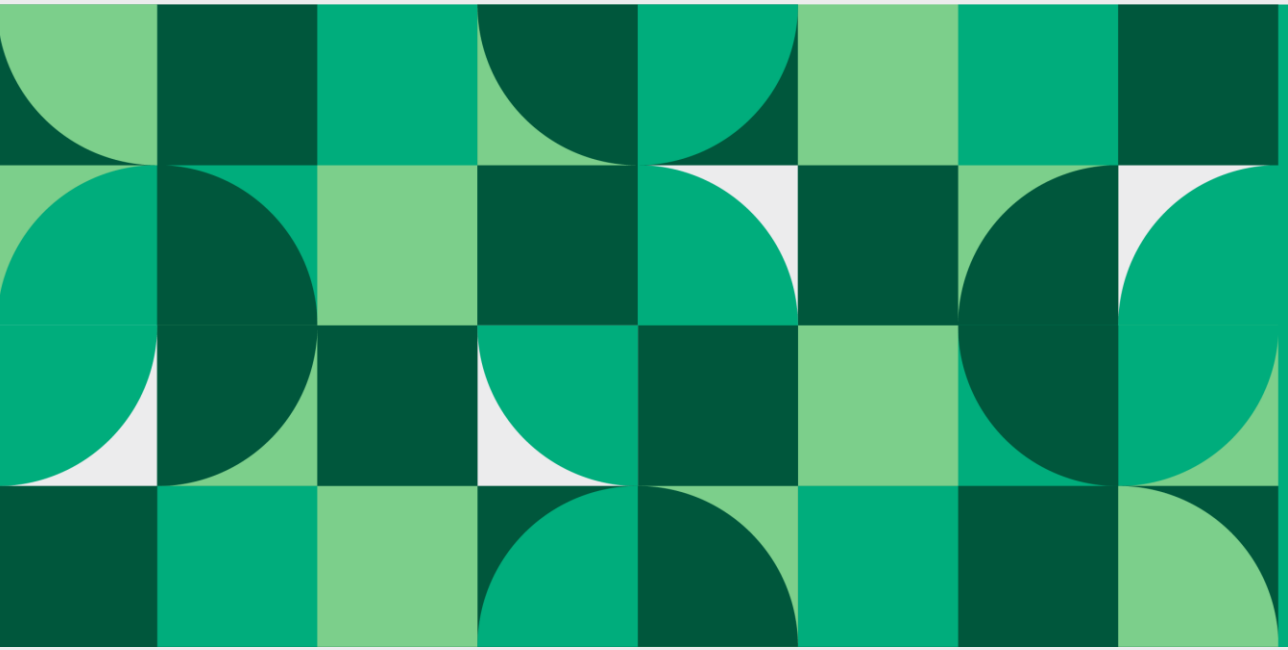
DIY

- This is always a conversation
- Include us…We can help

Want to outsource?

- This is always a conversation
- Include us…We can help
- We can point you to partners who can help more

Thank you!!



NI is now part of Emerson.