# Using model-based design to implement the motor control logic of a fully electric downhole flow-control valve



### Michel Gardes, Schlumberger MATLAB Energy Conference 2021, November 16-17

# Agenda

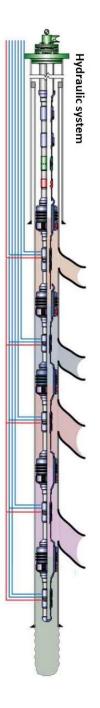
- Project presentation
- Simulink model
- Motor control logic
- Test harness
- Code generation
- CPU load concern
- Improvements in the model and in the generated code
- Results
- Conclusion

# **Project presentation**

Downhole flow control valves

- Control the flow of a producing zone
- High flow-rates: up to 60 000 bbl / day
- Hydraulic technology
  - Installation complexity: 1 hydraulic line per valve + 1 optional return line
    - Limits the number of valves in a well (max 5)
    - Several feedthroughs, potential leakage paths
  - Indexing values: fully closing requires fully opening first
- Electric technology
  - Simpler installation: 1 single electrical line for all the valves in the well
    - Only one feedthrough running through the packers
    - Increased reliability
  - Versatility: the valve can be actuated from any position to any position







## Electric downhole flow-control valve

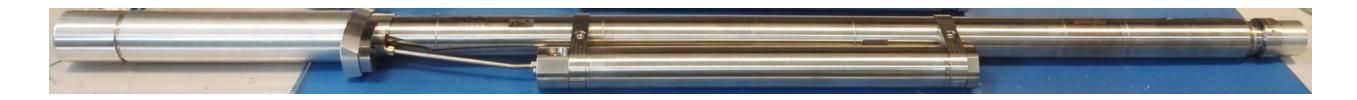
Motor control electronics must control and drive an Electro-Mechanical Actuator (EMA):

- 3-phase Permanent Magnet Synchronous Motor (PMSM)
- Gear box
- Roller screw



Position measurement: resolver mounted on the motor shaft

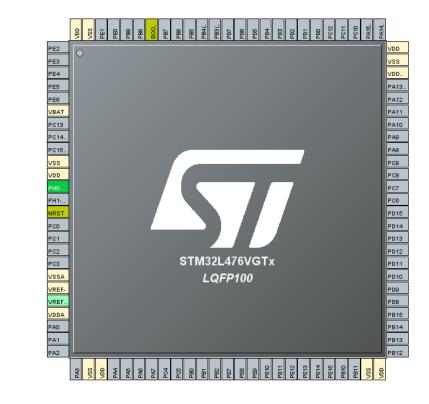
- The EMA must be able to apply the maximal force at the very beginning of an actuation ۲
- Sensored control more suitable for applications requiring high torque at zero / low speed





STM32L476VG from ST microelectronics (ARM Cortex-M4 processor): most powerful microprocessor qualified for high-temperature permanent applications

- Low power
- Advanced-control timers dedicated to motor control applications
  - Complementary PWM outputs with programmable dead-time
  - Break inputs for safety purposes
- Dual-mode ADC (analog-to-digital converter) operation for simultaneous acquisition of phase current values
- Floating Point Unit for higher performance



# Reasons for using model-based design

Late availability of the required motor (custom development for high temperature applications)

- Need to develop and test the control algorithm before receiving the motor Different motor models to be tested
- Flexibility to adapt and tune the control algorithm for different motor types MBD enables more effective troubleshooting
- The objective is to troubleshoot with the simulation instead of analyzing the failure on the hardware

Ability to simulate corner cases

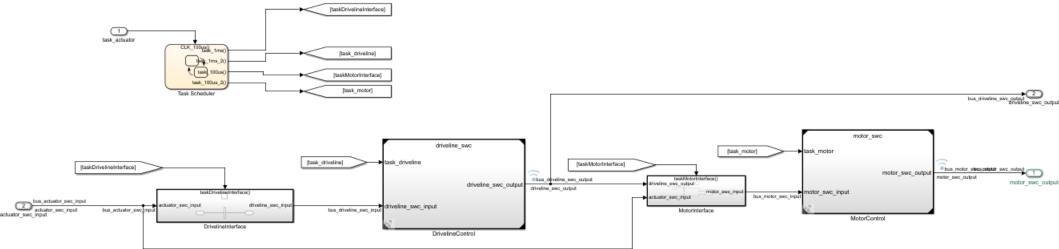
### Motor control method

Field Oriented Control (FOC)

- High dynamic performance
- Full torque at zero speed



Developed in MATLAB / Simulink in collaboration with MathWorks



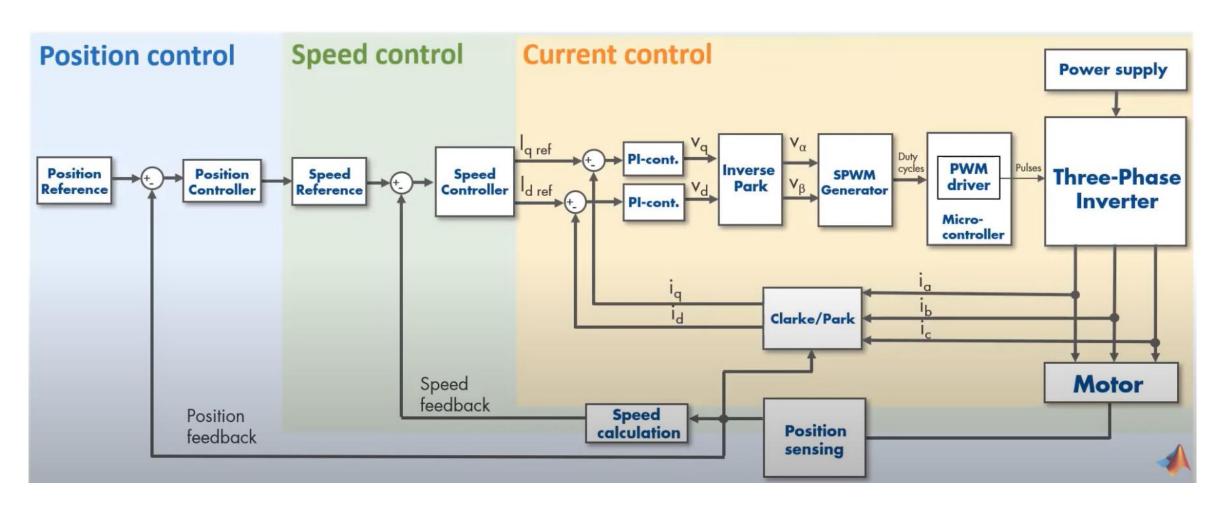
# **Control loops**

- **Position control loop** ٠
  - Position setpoint •
  - Default operating mode •

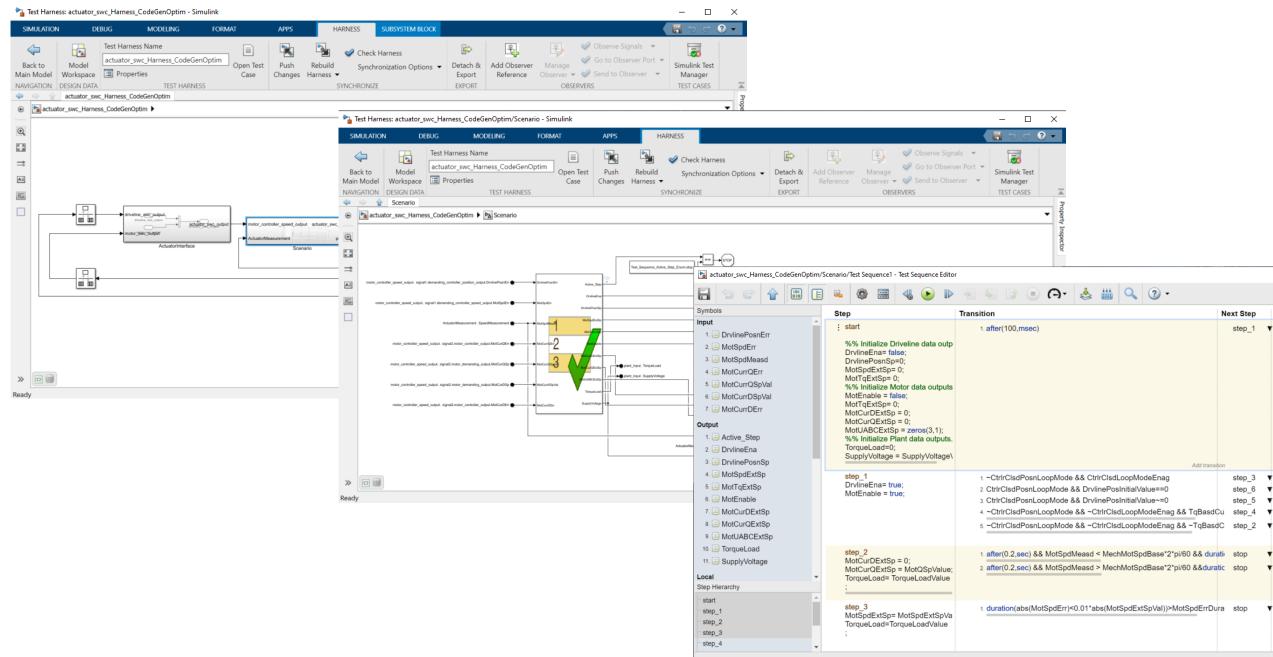
### Speed control loop

• Speed OR torque setpoint

#### Current control loop PI controller for maximal Iq • PI controller for Id = 0•



### **Test Harness**



### - 0 × Next Step Description step\_1 Add transitio step\_3 🔻 step 6 🔻 step 5 🔻

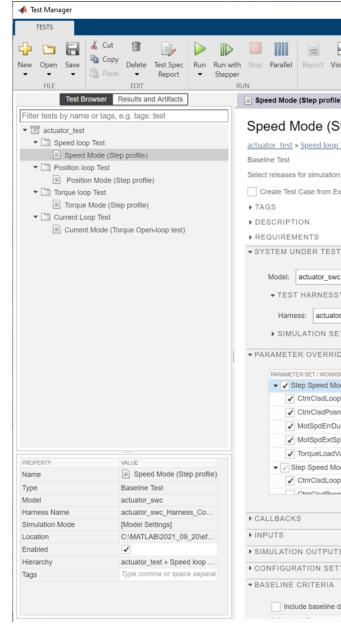
# Simulink Test Manager

New approach for the team

Much easier to launch tests and keep track of results

Erratic behaviour when exploring results





			-		×
sualize High	light <i></i> Expor	Preference	Help		Ā
	Start Page 🗙	LIVINOIVIL	NT RESOURCES		
					*
step pro	file)		~	Enabled	
Test » Speed	Mode (Step pr	ofile)			
1: Current 🕶	-				
xternal File					
۲*				?	
			_		
0					
*					
or_swc_Harne	ss_CodeC 💌	C. 🐿			
TTINGS OV	ERRIDES				I.
DES*				?	
SPACE VARIABLE	OVERRIDE VA	SOURCE	MODEL ELEM	+	
de accelera				<u>_</u>	
pModeEnag	true	DD_intern	actuator		
nLoopMode	false (1x1	DD_intern	driveline		
urationEval	10 (1x1 Si	DD_test_d	actuator		
pVal	100 (1x1 S	DD_test_d	actuator		
/al	0.1	DD_test_d	driveline		
de decelera					
pModeEnag	true (1x1 S		actuator	-	
ebol/laco.la	Add - C	Refresh 🚚 E	xport 💼 Dele	# ete	
	-			?	
				?	
S				?	
	DDIDES			?	
TINGS OVE	RRIDES				
				?	
data in test res	sult				*

# Code generation

C code generation with MATLAB Embedded Coder

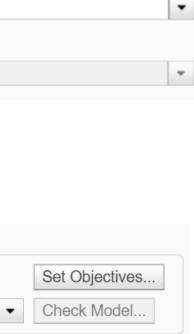
• Control algorithm only

### Optimized for the STM32 microprocessor

Hardware board: STM32 Nucleo L476RG			
Code Generation system target file: ert.tlc			
Device vendor: ARM Compatible	Ŧ	Device type:	ARM Cortex

### Objective: execution efficiency = lowest CPU load

Code generation objectives
Prioritized objectives: Execution efficiency
Check model before generating code: Off



## **CPU load concern**

Preliminary analysis based on data from ST, TI, Renesas

• Estimation: 1000 to 1300 cycles for sensored FOC at 10 kHz

PIL (Processor in the Loop) tests with Simulink

- Simulink report: 60% at 80 MHz, i.e. 4800 cycles at 10 kHz
- Lab measurements: 61.6% at 77.4 MHz, i.e. 4772 cycles at 10 kHz
  - With compiler optimization for speed

Unsatisfactory results

Lower CPU frequency targeted

Post-generation code enhancements for maximal performance









## Model parameters and signals

Simulink parameters and signals defined as auto are converted to 64-bit values in the generated code

Must be defined as *single* when using a 32-bit FPU 

Name	Status	Value	DataType	Dimensions	Complexity	Min	Max	Unit	StorageClass	
bus_motor_demanding_output										DD_i
MechMotSpdBase	[]		single	[0 0]	real	[]	[]	rpm	Auto	DD_i
MotCurDExtSpFiltd			single	-1	auto	[]	[]	А	Auto	DD_i
MotCurDSp			single	-1	auto	[]	[]	А	Auto	DD_i
🐻 MotCurDSpMax	[]		single	[0 0]	real	[]	[]	А	Auto	DD_i
MotCurDSpMin	[]		single	[0 0]	real	[]	[]	А	Auto	DD_i
MotCurDSpTqBasd			single	-1	auto	[]	[]	А	Auto	DD_i
MotCurQExtSpFiltd			single	-1	auto	[]	[]	А	Auto	DD_i
MotCurQSp			single	-1	auto	[]	[]	А	Auto	DD_i
MotCurQSpMax	[]		single	[0 0]	real	[]	[]	А	Auto	DD_i
MotCurQSpMin	[]		single	[0 0]	real	[]	[]	А	Auto	DD_i
MotCurQSpTqBasd			single	-1	auto	[]	[]	А	Auto	DD_i
🐻 MotCurSpMax	[]		single	[0 0]	real	[]	[]	А	Auto	DD_i
motor_demanding_output			Bus: bus_motor_demanding_output	ıt -1	auto	[]	[]		Auto	DD_i
MotTqToMotCur	[]		single	[0 0]	real	[]	[]	A/N/m	Auto	DD_i
🖁 PermanentFlxLnkg	[]		single	[0 0]	real	[]	[]	Wb	Auto	DD_i
T_CURRENT_DEMANDING	0.0001		single	[1 1]	real	[]	[]		Auto	DD_i
🐻 TqBasdCurSpEnag	[]		boolean	[0 0]	real	[]	[]		Model default	DD_i

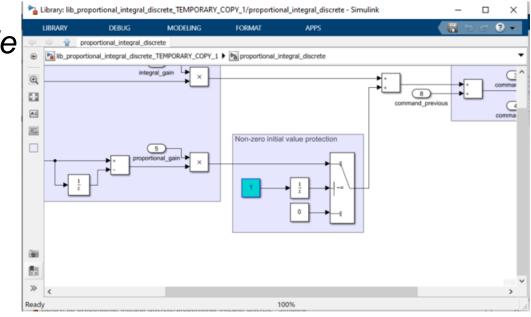
<

interna interna interna interna \_interna interna interna \_interna interna interna interna interna interna interna interna interna interna interna

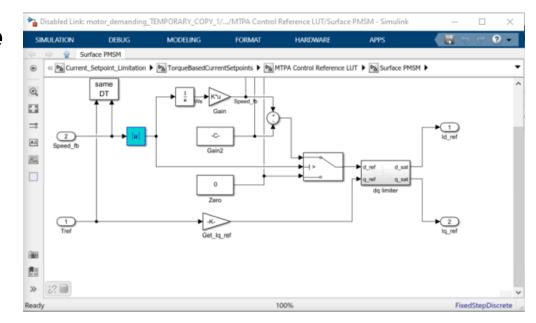
#### >

## Simulink blocks – constant and abs

The data type of the constant block output must be set to single



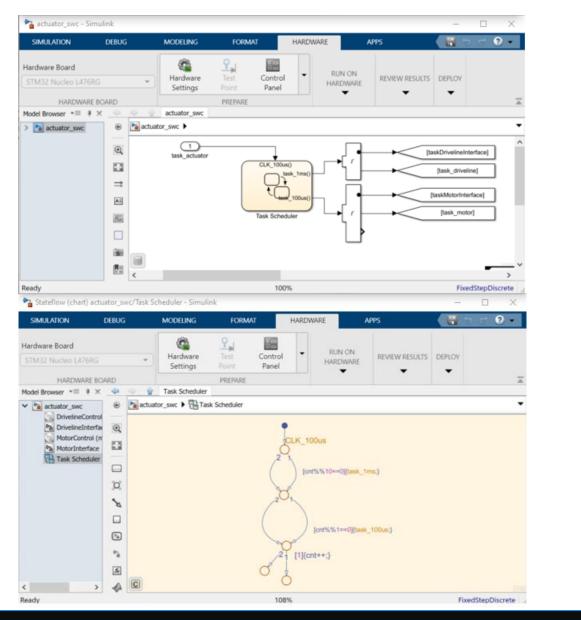
The abs block output data type must be set to single

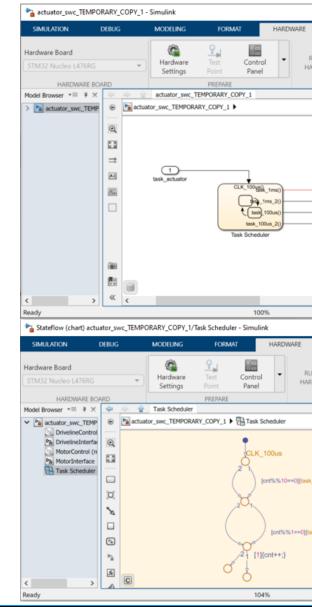


# Simulink blocks – function call split



### Function call splits are not 32-bit compatible and must be replaced by as many individual function calls as required







		-			×
A	NPPS ·		9 G	?	•
RUN ON ARDWARE	REVIEW RESULTS	5 DEPLO	Y		I
					•
	[tasi	kDrivelineIn	terface]		^
	$\rightarrow$	(task_drive)	ine]		
		skMotorinte [task_mot		]	
			٢	_	~
		Fixe	edStepl		_
		Fix	edStepl	Discre	_
АР	PS			Discre	te 🔮
IN ON	PS REVIEW RESULTS	-		Discre	te 🔮
JN ON		-		Discre	te /
IN ON IDWARE ▼	REVIEW RESULTS	-		Discre	te /

### C standards: C89/90 versus C99

May have considerable impact on the implementation of floating-point operations

<b>Q</b> Search			
Solver Data Import/Export Math and Data Types Diagnostics Hardware Implementation Model Referencing	Transport layer:       tcpip         MEX-file arguments: <empty>         Static memory allocation</empty>	✓ MEX-file name: ext_comm	
Simulation Target Code Generation Optimization Report Comments Identifiers Custom Code Interface Code Style Verification Templates Code Placement Data Type Replacement Coverage	<ul> <li>Advanced parameters</li> <li>Standard math library:         <ul> <li>non-inlined S-functions</li> <li>Maximum word length:</li> <li>Multiword type definitions:</li> <li>Buffer size of dynamically-sized string (bytes):</li> <li>Classic call interface</li> <li>Use dynamic memory allocation for model initi</li> <li>Single output/update function</li> <li>Terminate function required</li> <li>Combine signal/state structures</li> <li>Generate separate internal data per entry-poir</li> <li>MAT-file logging</li> </ul> </li> </ul>		•

### C89/90

### **C99**

/* End of Switch: ' <s6>/Switch2' */</s6>	/* End of Switch: ' <s6>/Switch2' */</s6>
) <b>else</b> {	} <b>else</b> {
/* Abs: ' <s8>/Abs' */</s8>	/* Abs: ' <s8>/Abs' */</s8>
<pre>rtb_Abs = (real32_T)fabs((real_T)rtu_motor_swc_input-&gt;MechMotSpdMeasd);</pre>	<pre>rtb_Abs = fabsf(rtu_motor_swc_input-&gt;MechMotSpdMeasd);</pre>

<pre>if (rtb_convert_pu &lt; 0.0F) {     /* Outputs for IfAction SubSystem: '<s7>/If Action SubSystem' incorporates:         ActionPort: '<s9>/Action Port'         //</s9></s7></pre>	<pre>if (rtb_convert_pu &lt; 0.0F) (     /* Outputs for IfAction SubSystem: '<s7>/If Action Subsystem'     ActionPort: '<s9>/Action Port'     */</s9></s7></pre>
<pre>rtb_convert_pu -= (real32_T)(intl6_T)(real32_T)floor((real_T)rtb_convert_pu);</pre>	<pre>rtb_convert_pu -= floorf(rtb_convert_pu);</pre>



### C89/90

### **C99**

<pre>rtb_Add3 = rtb_MotUABCSpRaw[0]; } else {    rtb_Add3 = rtb_MotUABCSpRaw[1]; }</pre>	
<pre>} /* MinMax: '<s16>/Max' */ if (rtb_MotUABCSpRaw[0] &gt; rtb_MotUABCSpRaw[1]) {     rtb_MotEDObsrvd = rtb_MotUABCSpRaw[0]; } else {     whetEDObsrvd = rtb_MotUABCSpRaw[0]; }</s16></pre>	
<pre>rtb_MotEDObsrvd = rtb_MotUABCSpRaw[1]; } /* MinMax: '<s16>/Max1' */ if (rtb_Add3 &gt;= rtb_MotUABCSpRaw[2]) {    rtb_Add3 = rtb_MotUABCSpRaw[2]; </s16></pre>	
<pre>} /* MinMax: '<s16>/Max' */ if (rtb_MotEDObsrvd &lt;= rtb_MotUABCSpRaw[2]) {    rtb_MotEDObsrvd = rtb_MotUABCSpRaw[2]; }</s16></pre>	
<pre>/* Product: '<s16>/Product' incorporates:     * Constant: '<s16>/Constant'     * MinMax: '<s16>/Max'     * MinMax: '<s16>/Max1'     * Sum: '<s16>/Add'     */</s16></s16></s16></s16></s16></pre>	<pre>/* Product: '<s16>/Product' incorporates:     * Constant: '<s16>/Constant'     * MinMax: '<s16>/Max'     * MinMax: '<s16>/Max1'     * Sum: '<s16>/Add'     */</s16></s16></s16></s16></s16></pre>
rtb_Add3 = (rtb_MotEDObsrvd + rtb_Add3) * 0.5F;	<pre>rtb_Add3 = (fmaxf(fmaxf(rtb_MotUABCSpRaw[0], rtb_MotUABCSpRaw[1]),</pre>
<pre>rtb_MotUABCSpRaw[0] -= rtb_Add3; rtb_MotUABCSpRaw[1] -= rtb_Add3; rtb_MotUABCSpRaw[2] -= rtb_Add3;</pre>	<pre>rtb_MotUABCSpRaw[0] -= rtb_Add3; rtb_MotUABCSpRaw[1] -= rtb_Add3; rtb_MotUABCSpRaw[2] -= rtb_Add3;</pre>

v[0],

Generation of code for both standards: C89/90 and C99

Best CPU load is achieved with a blend of C89/90 and C99



CPU load of generated code for the control logic after improvement

4772 cycles  $\rightarrow$  1498 cycles

No compiler optimization

Lower clock frequency



Implementing Model-Based Design with the help of MathWorks consulting team allowed us to achieve challenging deadlines

Code generation is very useful, but might require manual intervention for optimal performance

## Acknowledgements

Wided Zine, MathWorks Zakaria Mahi, MathWorks Denis Heliot, Schlumberger

### Q&A

