APPLICATION OF AUTOMATIC CODE GENERATION FOR RAPID AND EFFICIENT MOTOR CONTROL DEVELOPMENT

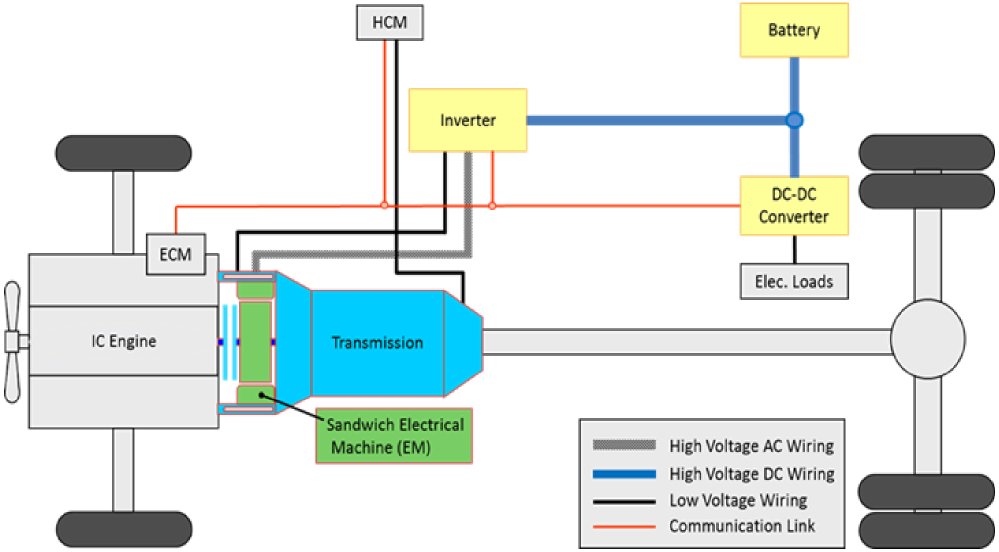
Edward Kelly, James Walters, Cahya Harianto, and Tanto Sugiarto

DELPHI
Innovation for the Real World

Hybrid Vehicle Motor Controls Overview

Requirements dictate fast algorithm execution

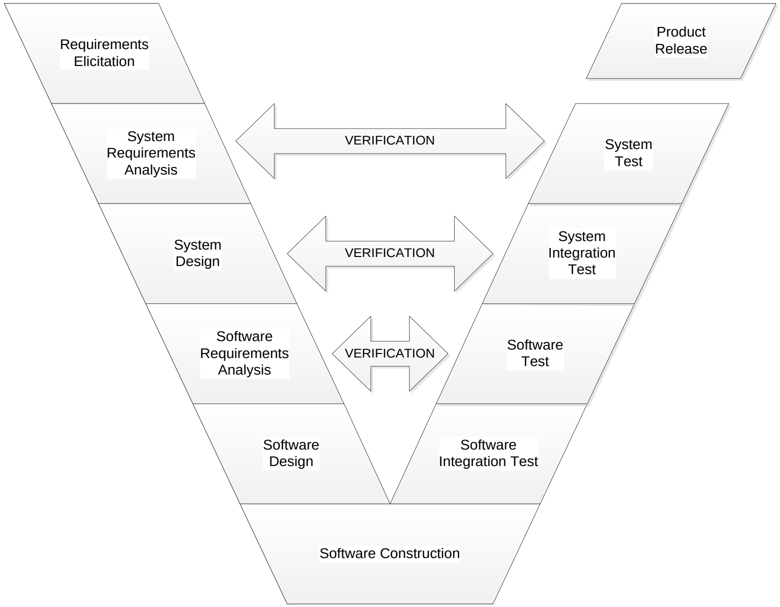
- Torque, speed, voltage and fault reaction modes
  - 1000+ Hz fundamental frequency
  - 500+ Hz current regulator bandwidth
  - 10 kHz PWM rates for DC voltage ripple
- 80 – 100 uSec control loops are common



Software Development Process

General Approach

- Responsibilities
  - Systems: Analyze, derive and specify
  - Software: Implement
  - Software/Systems/Validation: Verify
- Time consuming and error prone
  - Requirements formation, implementation and verification are too dispersed
  - Decoupling of domain knowledge from implementation



Automatic Code Generation

Benefits

- Linking of simulation, code development and testing
  - Common in 5 – 10 mSec task rates
  - Improves testability.
- Implementation responsibility transitions to domain experts
- Potential for time savings
  - Faster verification of implementation
  - Production hardware can be used for design and detailed problem solving

Challenges

- Creating and maintaining easily understood environment and models
- Identifying preferred implementations

Goal

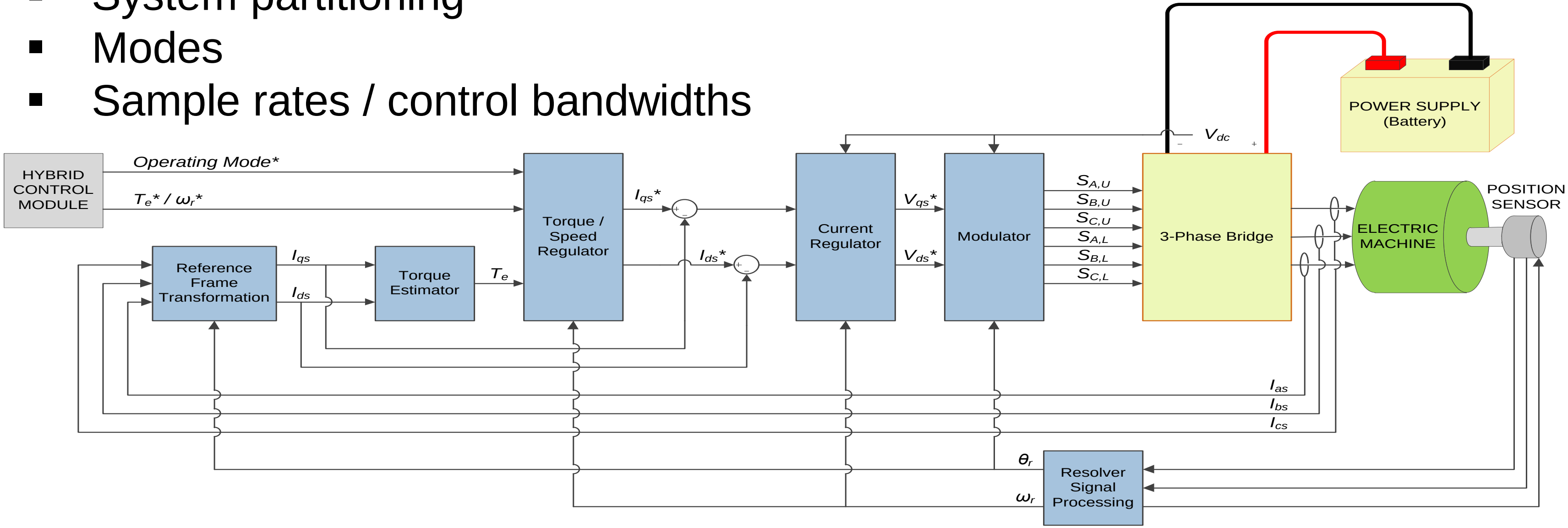
Develop automatic code generation process for time critical tasks

- Requirements focused
  - To support Automotive SPICE
- Directly apply system expertise to implementation
  - Create path for high level simulation models to software
- Shorten time between design, implementation and verification steps
  - Reduce development time
- Create easy to understand implementations that can be shared among teams
- Closely match hand-code throughput efficiency

Process: Requirements Derivation / Partitioning Phase

High level design

- Verify overall requirements are met
- Establish derived requirements
  - Architecture
  - System partitioning
  - Modes
  - Sample rates / control bandwidths

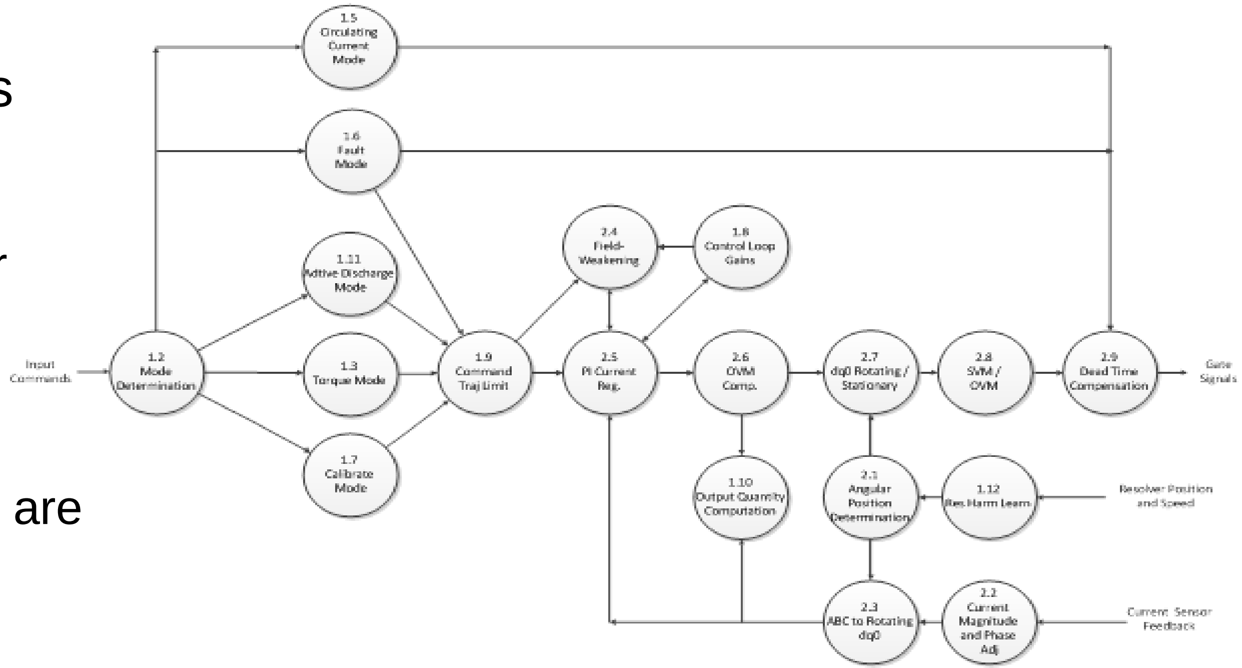


DELPHI

Process: Implementation Phase

Functional Modules

- Testable requirements
  - Inputs / outputs
  - Functionality
  - Execution rate / order
- Model development
  - Best practices
- Documentation
  - Model / requirements are not sufficient
- Test vectors
  - Simulation
  - Requirements verification



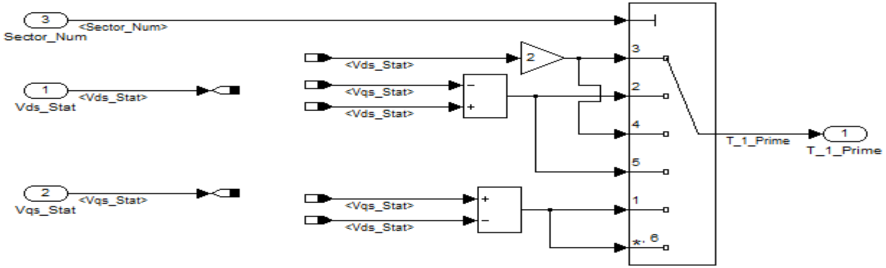
Process: Implementation Phase (Modelling)

Simulation tools offer numerous options for implementation

- Not all approaches will code with equal efficiency
  - Tools have optimization settings
- Consistency of implementation among modules is important for ‘read-ability’
  - Key for sharing among teams
- Peer review process is important to ensure efficient code
  - Systems: Implementation meets requirements
  - Software: Optimization and problem resolution
    - Detailed review of code
    - Identification of best practices

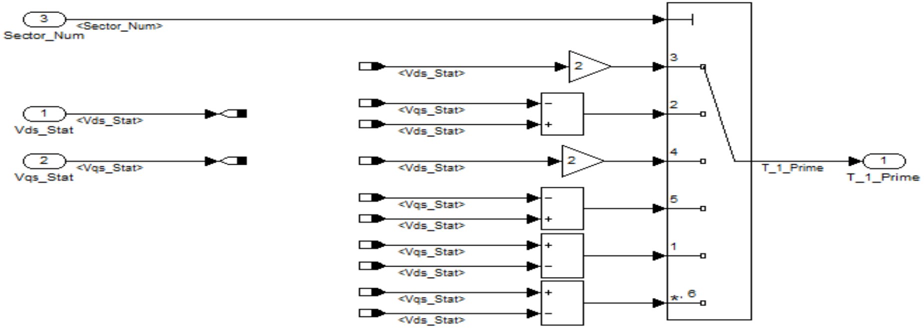
Process: Implementation Phase (Example 1)

Inefficient Model / Code:




```
real32_T rtb_Gain3;
real32_T rtb_Add2;
rtb_Add2 = Vds_Stat - Vqs_Stat;
T_1_Prime = Vqs_Stat - Vds_Stat;
rtb_Gain3 = 2.0F * Vds_Stat;
switch (Sector_Num) {
 case 3:
  T_1_Prime = rtb_Gain3;
  break;

 case 2:
  T_1_Prime = rtb_Add2;
  break;

 case 4:
  T_1_Prime = rtb_Gain3;
  break;

 case 5:
  T_1_Prime = rtb_Add2;
  break;

 case 1:
  break;
}
```

Efficient Model / Code:




```
switch (Sector_Num) {
 case 3:
  T_1_Prime = 2.0F * Vds_Stat;
  break;

 case 2:
  T_1_Prime = Vds_Stat - Vqs_Stat;
  break;

 case 4:
  T_1_Prime = 2.0F * Vds_Stat;
  break;

 case 5:
  T_1_Prime = Vds_Stat - Vqs_Stat;
  break;

 case 1:
  T_1_Prime = Vqs_Stat - Vds_Stat;
  break;

 default:
  T_1_Prime = Vqs_Stat - Vds_Stat;
  break;
}
```

DELPHI

Process: Implementation Phase (Example 2)

Inefficient Embedded MATLAB code:

```
floor_index = uint32(unfloor_index);
```

Generated code:

```
/*  ================================================================== */
/* '<S3>:1:46' */
tmp = unfloor_index;
if ((unfloor_index < 8.388608E+6F) && (unfloor_index > -8.388608E+6F)) {
  tmp = (unfloor_index < 0.0F) ? ceilf(unfloor_index - 0.5F) : floorf
    (unfloor_index + 0.5F);
}

floor_index = (uint32_T)tmp;
```

Process: Implementation Phase (Example 2)

Inefficient Embedded MATLAB code:

```
floor_index = uint32(unfloor_index);
```

Generated code:

```
/*  ================================================================== */
/* '<S3>:1:46' */
tmp = unfloor_index;
if ((unfloor_index < 8.388608E+6F) && (unfloor_index > -8.388608E+6F)) {
  tmp = (unfloor_index < 0.0F) ? ceilf(unfloor_index - 0.5F) : floorf
    (unfloor_index + 0.5F);
}

floor_index = (uint32_T)tmp;
```

Process: Implementation Phase (Example 2)

Efficient Embedded MATLAB Code:

```
        floor_index = coder.ceval('SingleToInteger32',unfloor_index);
```

Hand-coded CustomFuntion.h:

```
#ifndef CustomFunction_H
#define CustomFunction_H

#include "rtwtypes.h"

#define SingleToInteger32(u)   ( (int32_T) u )
```

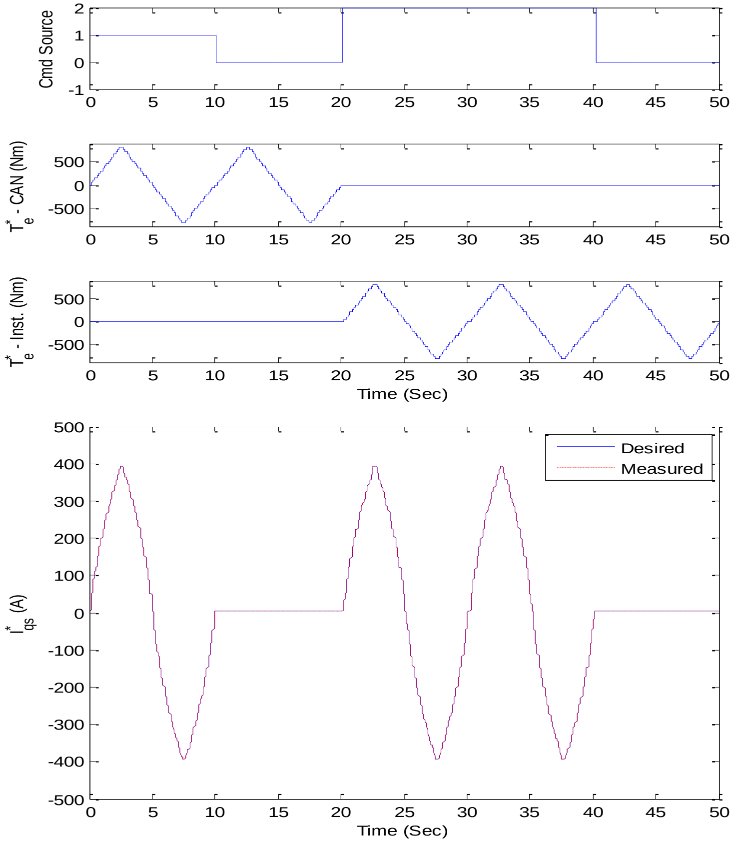
Generated code:

```
/* '<S3>:1:46' */
floor_index = SingleToInteger32(unfloor_index);
```

Process: Verification Phase

Test to verify requirements are met

- Module test vectors
  - Verify functionality
    - Inputs / internal variables / outputs
- Full model test vectors
  - Simulation environment
  - Hardware in the loop
    - Correct compiler
    - Simulate virtual load in processor or test bench



DELPHI

Evaluation of Process

Verified auto generated software was dynamometer tested to evaluate performance

- Comparison was made to mature hand-code
- Equivalent motor control functionality
- Slight penalty in 100 uSec task throughput
  - 1.54 uSec

Task / Module	Throughput (uSec)	
	Model	Hand-Code
Current Magnitude and Phase Process	1.42	1.31
ABC to dq0 Frame Transformation	0.76	0.52
Resolver Harmonic Learn	0.48	0.22
Angle Position Determination	0.93	0.84
PI-Current Regulator	7.62	7.51
Torque Mode	4.82	4.72
dq0 Rotating to Stationary Frame Transformation	0.94	0.82
Complete 100 uSec Task	**65.37**	**63.83**

DELPHI

Summary

Structured automatic code generation can be applied to time critical tasks

- A process is required to ensure efficient implementations
- New roles
  - Software: responsible for auto-coding environment, determining best practices, peer reviewing implementations and detailed problem solving
  - System: responsible for forming requirements, creating implementations that demonstrably meet requirements (test vectors) and following identified best practices
- Requirements and models are not sufficient to document implementation
- Automatic code generation should be viewed as a tool to link simulation, implementation and verification testing
  - Concurrent activities speed the software development process

Acknowledgement of Support

Acknowledgement: “This material is based upon work supported by the Department of Energy under Award Number DE-FC26-07NT43121.”

DELPHI

DELPHI

Innovation for the Real World