Raptor

Model-Based Control System Development Platform
Development Platform for Full System Lifecycle

- Control system development tool suite for taking a project from start to finish including:
  - Concept, ECU & Component Specification
  - Design & Software Development using Model-Based Development
  - Hardware / Wiring Development
  - System Integration and Calibration
  - Validation
  - Production
  - Service
Raptor Control Development System

- The Raptor development system allows customers to bring new products and features to market quickly
  - Fast development cycle times
  - High flexibility
  - Customer ownership of control strategy IP
- Comprises a suite of controller products, industry-standard software tools and application engineering expertise
- End-to-end suite of tools for full product lifecycle
Raptor Overview

Vehicle, Industrial & Power Generation Controls
Prototype on Production Hardware

- Achieve both cost and time savings using the same hardware through the whole system lifecycle
- Avoid extra iteration cycles needed to convert (shoe-horn) prototype design to production hardware
- Harness / physical design can run parallel to controls development
Raptor Overview

- Model-based controls development environment for simulation, rapid prototyping, and production controls development
- Built upon MATLAB, Simulink, and Simulink Coder
- Raptor Build Engine handles the setup and configuration of all processor resources and includes XCP support for secure calibration of the ECU
- Allows the same software to be targeted/compiled to multiple ECUs with minimal changes to the model
Why Model-Based Development?

• Shortened V-Cycle for Model-Based Software Development

• Leverage proven design elements & code-generation to create software vs one-off hand created elements & source code

• Proven ECU hardware reduces H/W verification efforts
Why Model-Based Development?

• Traditional Development Process
  • ‘Paper’ handoffs, opportunity for misunderstanding from person to person
  • Opportunity for variability in tool-chain from ECU to ECU, project to project
Why Model-Based Development?

Automated Model-to-Executable Build Engine
Why Model-Based Development?

- Consistent application development interface across ECU hardware
  - Software portability across Raptor hardware platforms (libraries, software subsystems)
  - Leverage training & infrastructure across products / platforms
Built on latest MathWorks Toolset

- Targets 2012a+ Simulink, Simulink Coder, Embedded Coder
  - Improved code efficiency over prior releases
  - Code generation report enhancements
- Possible support of Model Reference for enhanced software modularity & build efficiency
  - Incremental code generation for top-level models
Raptor Architecture

Application Layer
- Sensor Software Component
- Application Software Component
- Actuator Software Component
- Model-Based System Architecture

Raptor Build Engine

ECU SDK Layer
- Operating System / Scheduler
- Services
- Communications
- ECU Abstraction
- Complex Drivers
- uC Abstraction

Rugged ECU Hardware

New Eagle Confidential
Raptor Build Engine

• Designed to maximize flexibility when mapping Raptor Simulink Model-Based interface blocks onto the target hardware (SDK)
• Raptor block parameters are data-driven from ECU target description file (XML)
• Allows efficient module integration
Comprehensive Set of Application Development Features

- Timer / Trigger blocks
- Target Definition
  - Stack(s), Heap Definition
  - RAM, FLASH, EEPROM Configuration
  - Build output artifact configuration
- Industry-standard calibration protocol (XCP)
  - Measurement points
  - Calibration points
  - 1D, 1D interpolation table lookups
  - 2D, 2D interpolation table lookups
  - Scalar, Enum, Array calibration tables
  - User configurable read/write access level per parameter
  - Absolute / relative signal overrides
  - ASAP2 file generation

- Analog input
- Digital input
- Frequency input
- PWM output
  - Current feedback
- Model-based CAN bus interface
  - Configurable baud rate / bit timing
  - Raw transmit / receive interface
  - DBC message transmit /receive interface
  - Automated parsing / message generation
- Fault Management (Standard and OBD)
- Efficient Simulink Coder generated application algorithm code integration
- Build engine hooks for target-specific post-build executable file manipulations
In addition to targeting rugged control hardware, Raptor also targets embedded displays

- Automated Bitmap code generation
- Exposes user input (buttons) via standard trigger mechanism
- PC based simulation of application created during Simulink build
  - CAN input to simulator via Kvaser Cable
  - I/O input via system
  - Allows rapid system display development & iteration
- VEECAN320
  - 2 CAN buses
  - 2 digital inputs
  - 7 analog inputs
  - 4 relay outputs
  - 1 frequency input
  - USB port

USB Port is exposed via file-based blockset in Raptor which allows data logging of system data
Easy to use documentation

- Integrated with Simulink Help
- Block by block documentation
Raptor J1939 Library

- Easily add J1939 transport protocol to an application
- Designed to meet OBD-HD compliance for an engine application
- Supports DTC based DM messages via seamless integration with the Raptor OBD fault manager
Industry Standard Tools Support

• Raptor exposes XCP, industry standard calibration protocol
• Automated A2L file generation during model build
• Efficient data transfer offers improved logging throughput vs proprietary protocols
• Customer-specific security option available to protect your intellectual property
• Most common calibration tools supported
New Eagle Calibration Tool

- Utilizes industry standard XCP calibration protocol
- Lower-cost option provides many of the features of the higher-end tools
  - Data-efficient logging
- Also offers integrated ECU reflash
- Support customer-specific security
- Lower startup costs/training if you are not already using or requiring higher-end tools
New Eagle Automated Testing Framework (NEAT)

- Target-in-the-Loop Framework
- Provides automated script driven testing of Control Algorithm & State Machines
- Repeatable tests for validation and regression
- Supports XCP and raw CAN for facilitating a wide variety of test scenarios
- Built-in support for macros
- Automated PDF report generation
New Eagle Automated Testing Framework (NEAT)

- Built-in Support for a coordinator ECU to provide plant model/environment
- Graphical Test Creation
- Intuitive ‘Test-Runner’ interface
- Allows traceability from requirements to validation tests
Automated End-of-Line Configuration/Unit Configuration Manifest

- NEAT serves as a programming utility for production end-of-line, can be integrated into OEM assembly operations
- Provides script-based, automated end-of-line configuration of parameters, with confirmation
- PDF report for each unit with configuration settings, and identifying values (VIN, Serial Number etc..)
Cost-Effective Tools for Service Fleet Deployment

• Supports the use of heavy-duty industry-standard RP1210 adaptor(s) commonly found at service shops today
• Also supports low-cost Kvaser CAN hardware
• Graphical interface for technicians to perform troubleshooting & field software upgrade packages
• Software-configurable access levels
  – Software Developer
  – Calibrator
  – Dealer/Service Technician
  – Consumer
• Supports customer-specific security to protect your application from tampering