ホワイトペーパー

自動車制御システムのモデルベースデザイン

チームで車両用高性能緊急ブレーキシステム (AEBS) を開発している場面を思い浮かべてみてください。衝突の危険が差し迫っている場合、AEBS は音声アラームでドライバーに警告します。ドライバーが反応しない場合は、警告ブレーキをかけます。ドライバーがブレーキをかけても、衝突の回避に必要な力が足りなければ、システムが追加で必要なブレーキ力を計算して適用します。設計を始める前に、たとえば次のような主な疑問点を解消しておくことが望ましいと考えられます。

- ブレーキのサイズはどれくらいか 。
- 要件が変更されたらどうなるか。
- 望まれる性能を確保するためには、どのような設計の最適化が可能か。
- リスクを最小化した上で、どのように設計の完全なテストができるか。

産業用ロボット、風力タービン、生産用機械、自律車両、掘削機、または電気サーボドライブのどの制御を開発する場合であっても、チームが手作業でコードを書き、ドキュメントベースで要件を記録しているとすれば、これらの疑問を解消する方法は、試行錯誤または物理プロトタイプでのテストしかありません。そして要件が 1 つでも変更されたら、システム全体を再コーディングして再構築しなければならなくなり、プロジェクトが数日、または数週間遅延することになります。

手書きのコードとドキュメントの代わりに MATLAB® と Simulink® によるモデルベースデザインを使用すれば、システム モデル (緊急ブレーキシステムのモデルは、ブレーキ、車両運動、環境、および制御で構成されます。) を作成することができます。どの段階でもモデルをシミュレーションして、リスクや遅延なしで、高価なハードウェアを用いることなく、システム挙動を即座に確認し、複数の what-if シナリオをテストすることができます

このホワイトペーパーではモデルベースデザインを紹介し、導入のためのベストプラクティスを説明します。実際の例を用いて、業界の垣根を超えた複数のチームがモデルベースデザインを採用し、開発期間を短縮し、コンポーネント統合に関わる問題を最小化して、より高品質な製品の提供を実現した方法を見ていきましょう。

モデルベースデザインとは?

モデルベースデザインを理解する最良の方法は、実例を見ることです。

ある自動車エンジニアのチームが、乗用車のエンジン制御ユニット (ECU) を構築する場合を見てみましょう。チームはモデルベースデザインを用いるため、システム要件からのアーキテクチャモデルの構築から作業を始めます。このケースでは、システムは 4 気筒エンジンです。シミュレーション/設計モデルはここで派生します。この高レベル、低忠実度のモデルには、ECU およびプラントで実行される制御ソフトウェアの一部が含まれます。このケースでは、エンジンと動作環境です。

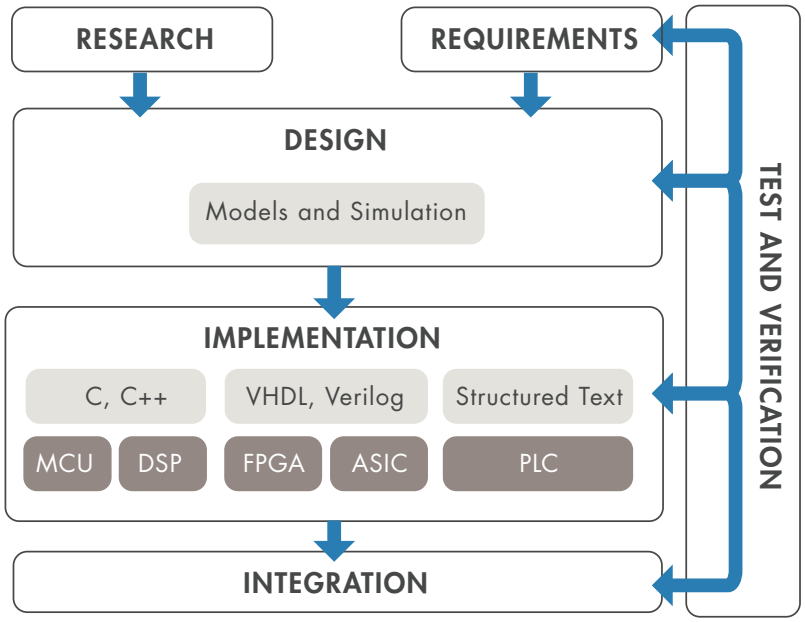
チームは、この高レベルのモデルをさまざまなシナリオでシミュレーションし、システムが正しく表現され入力信号に適切に応答していることを検証することで、最初のシステムテストおよび統合テストを行います。

モデルに詳細を追加し、仕様と比較してシステムレベルの挙動を継続的にテストして検証します。システムが大規模で複雑な場合、エンジニアは個別のコンポーネントを単独で開発およびテストし、さらにシステム全体のシミュレーションでそれらを頻繁にテストすることができます。

最後に、チームはシステムの詳細モデルとその動作環境を構築します。このモデルにはシステムに関する蓄積された知識が含まれています (IP)。エンジニアは、ソフトウェアのテストと検証を行う制御アルゴリズムのモデルからコードを自動的に生成します。また、ハードウェアインザループ テストに従い、チームは生成されたコードを量産ハードウェアにダウンロードし、実際の車両でテストを行います。

このシナリオから分かるように、モデルベースデザインは従来の開発ワークフローと同じ要素を使用する一方で、主に次の 2 つの違いがあります。

- 時間がかかるうえに間違いの発生しやすいワークフロー上の手順 (たとえばコード生成) の多くが自動化されている。
- 要件の記録から設計、実装、およびテストに至るまで、システム モデルが開発の中心となっている。



モデルベースデザインのワークフロー

要件の記録および管理

要件がドキュメントで記録される従来のワークフローでは、引き継ぎにより誤りや遅延が生じることがあります。設計ドキュメントや要件を作成するエンジニアと、システムを設計するエンジニアが違う人物であることはよくあることです。要件が「丸投げ」されることもあります。これは、2 つのチーム間に明確なまたは一貫したコミュニケーションがないという意味です。

モデルベースデザインでは、Simulink モデル内で要件の作成、解析、管理を行うことができます。また、カスタム属性を使用したリッチテキストの要件を作成し、それを設計、コード、およびテストにリンクさせることができます。要件は、要件管理ツールのような外部ソースからインポートして同期することもできます。設計にリンクされている要件が変更されれば、自動的に通知が送られます。その結果、変更の影響を直接受ける設計またはテストを特定して、それに対応した適切なアクションをとることができます。また、システムとソフトウェアのコンポーネントに関するアーキテクチャと構成を定義、解析、および指定することができます。

ケーススタディ: Iveco

「当社のシステムエンジニアとソフトウェアエンジニアは、Simulink モデル上で直接共同作業しています。この方法は要件の誤解を生じないため、開発のスピードが増します。モデルが正しいと確信すれば、このモデルから生成するコードによって、さらに時間を短縮することができ、実装エラーもありません。」

— Demetrio Cortese, Iveco

Ivecoの大型車

Iveco は、中南米での中型車から大型車の市場機会を活用するため、9 速および 16 速のトランスミッションを搭載した車両用のシフトレンジ制御システムの設計、実装、テスト、納品を約 6 週間で行わなければなりませんでした。この厳しい納期では、ソフトウェア開発のスケジュールを圧縮する必要があり、仕様や実装のエラーに対応する時間は全くありませんでした。

プロジェクトの時間的制約のため、チームは、PLC を含む既存のハードウェア構成を使用することにしました。しかし、ソフトウェア エンジニアは、PLC 用のストラクチャード テキストを書いた経験がありませんでした。実装エラーと開発期間の延長を回避するためには、Iveco はストラクチャード テキストを自動的に生成する必要がありました。

Iveco の従来のアプローチでは、システムエンジニアが要件と仕様を定義し、それをソフトウェア エンジニアに引き渡していましたが、このプロジェクトの短いスケジュールで、この手順を踏むことは不可能でした。そこで、システムエンジニアとソフトウェアエンジニアは共同で作業し、Simulink でシステムの仮モデルを開発しました。

ソフトウェア　エンジニアは、制約、データ型、組み込みテスト、および診断を追加して、モデルの調整とカスタマイズを行いました。モデルのシミュレーションを行い、設計の整合性を検証して、オーバーフロー状態や未実行のブロック、その他の潜在的な問題を特定しました。

エンジニアは、PLC と実際のトランスミッションを使用したラボテストをリアルタイムで実施し、迅速にモデルの調整とコードの再生成を行い、管理システムが機能と性能の要件を満たすまでテストを繰り返しました。

設計

従来のアプローチでは、設計のアイデアはどれも物理プロトタイプでコーディングしてテストする必要がありました。その結果、各テストの反復によりプロジェクトの開発期間とコストが増大するため、調査できる設計のアイデアとシナリオの数は限られていました。

モデルベースデザインでは、事実上、アイデアを無制限に検討することができます。要件、システムコンポーネント、IP、およびテストシナリオのすべてがモデルに含まれています。また、モデルをシミュレーションすることができるため、高価なハードウェアを構築する前に、設計上の問題点と疑問点を調査することができます。素早く複数の設計アイデアを評価し、トレードオフを調べ、それぞれの設計変更がシステムにどう影響するかを確認することができます。

ケーススタディ: Ather Energy

Ather 450 インテリジェント電気スクーター。

「有望なアイデアは沢山ありましたが、小規模なスタートアップ企業のため、それぞれのテストを行うプロトタイプ構築のための時間、費用、および人員が不足していました。モデルベースデザインにより、シミュレーションを通じて最良のアイデアを特定および検証し、より短い期間でフル機能のスクーターの生産を実現できました。」

— Shivaram N.V., Ather Energy

バンガロールを走る車両の 70% を占める、500 万台以上の 2 輪スクーターのほとんどはガソリンを動力としているため、高レベルの騒音公害と CO_2 排出を引き起こしています。よりクリーンな代替手段を求める声に応えるため、スタートアップ企業の Ather Energy はインドで初めてのインテリジェント電気スクーターを開発しました。時速 0 km から 40 km へ 4 秒以下で加速可能なうえに、Ather 450 は時速 80 km の最高速度を誇り、1 回の充電で最大 75 km まで走行できます。

450 のような製品は市場でもまだ数少なかったため、チームは多数の未知の出来事に遭遇しました。スクーターと主要コンポーネントで構成されるプラント モデルを構築した後、シミュレーションを実行して走行および使用シナリオを評価しました。たとえば、傾斜、複数乗員、極端な温度、ほぼ消耗した状態のバッテリーなどで 450 を操作したりしました。

シミュレーション結果を用いて情報に基づく設計のトレードオフを作成しました。結果として、たとえば、バッテリー容量を増加させると走行距離が向上するが、コストとサイズも増大することに加えて、スクーターの重心を変化させてしまうことなどが分かりました。彼らは、コスト、サイズ、および温度の制約を満たしたうえで、ターゲットとする加速と走行距離の要件を満たすモーターとバッテリーの設定を特定するまで設計を改良しました。

スクーターそのものの設計に加え、エンジニアはバッテリー充電、温度管理、およびその他の主要機能の組み込み制御アルゴリズムも開発しました。詳細なコンポーネントデータが無かったため、チームは経験的アプローチによりバッテリーセルをモデル化しました。彼らはさまざまな温度と充電状態レベルでバッテリーをテストし、測定した入力と出力データを使用してセルの温度特性と熱特性のブラックボックス モデルを作成しました。

次に、彼らはバッテリー充電、電力制御、および温度制御のアルゴリズムを開発しました。閉ループ シミュレーションをプラント モデルで実行し、制御設計を検証しました。彼らはコントローラー モデルからコードを生成し、スクーターの ARM® Cortex® プロセッサと充電ステーションの TI C2000™ マイクロコントローラーに展開しました。

Ather 450 はすでに量産体制に入り、バンガロールで販売を開始しています。充電ステーションもバンガロールに 31 箇所、チェンナイに 7 箇所整備されています。

コード生成

従来のワークフローでは、組み込みのコードはシステムモデルからまたはゼロから手書きする必要がありました。ソフトウェア エンジニアは制御システムエンジニアが書いた仕様に基づいて制御アルゴリズムを書きます。仕様を書く、手作業でアルゴリズムをコーディングする、手書きのコードをデバッグする、というプロセスの各手順は、時間がかかりエラーが発生しやすいものになる可能性があります。

モデルベースデザインでは、手で数千行のコードを書く代わりに、モデルから直接コードを生成します。またモデルはソフトウェア エンジニアと制御システムエンジニアの橋渡しをします。生成したコードはラピッド プロトタイピングまたは量産用に使用できます。

ラピッド プロトタイピングでは、ハードウェア上でアルゴリズムをリアルタイムでテストする高速かつ安価な方法を提供します。そのため、従来数週間要していた設計の繰り返し作業を数分で実行することができます。プロトタイプのハードウェアまたは量産用 ECU を使用することができます。同じラピッド プロトタイピング ハードウェアと設計モデルを用いて、ハードウェアインザループ テストとその他のテストおよび検証作業を実施し、量産前にハードウェアとソフトウェアの設計を検証できます。

量産向けコード生成により、モデルが量産組み込みシステムに実装される実際のコードに変換されます。生成されたコードは、特定のプロセッサ アーキテクチャに対して最適化するか、手書きのレガシーコードと統合することができます。

ケーススタディ: Ponsse

Ponsse の山林ハーベスター Scorpion。

「Simulink モデルからエラーのないコードを生成する機能により、制御エンジニアはアルゴリズム設計に集中し、ソフトウェア エンジニアはファームウェア層のプログラミングに集中することができました。その結果、開発期間が短縮され、品質が向上し、コストを削減することができました。」

— Juha Inberg, Ponsse

Ponsse Scorpion は、険しい森林地帯での作業を想定して設計された 8 輪の山林ハーベスターです。本機の特徴的なフレームは、12% も回転するジョイントで連結した 3 つのセグメントで構成されています。このフレームにより、前後の車輪セグメントが地形の変化に合わせて調整され、センターセグメント上のキャビンを水平に保つことができます。

以前のプロジェクトでは、Ponsse の制御エンジニアが MATLAB でアルゴリズムを開発してデバッグし、ソフトウェア エンジニアがアルゴリズムを手作業で C 言語に変換していましたが、プロジェクトを重ねるにつれて制御アルゴリズムの複雑さが増大し、このアプローチを継続することが困難になりました。C コード生成に人的エラーが発生するリスクが高まるとともに、アルゴリズムの初期設計からハードウェア上での検証までに長い時間がかかっていました。Ponsse は、この間隔を短縮しつつコーディングエラーを最小限に抑え、全体的な開発期間を短縮したいと考えていました。

エンジニアは、加速度計やジャイロスコープからの入力を処理し、油圧バルブを作動させて Scorpion のセンターフレームを水平に保つ制御モデルを開発し、そのモデルからリアルタイムのアプリケーションを生成して Speedgoat ターゲット コンピューター ハードウェアに展開する、プロトタイプのコントローラーを構築しました。

チームはこのリアルタイム プロトタイプを使用して、実際の Scorpion のハードウェアでテストを実施しました。その結果に基づいて、更新されたプロトタイプを再生成して再度テストを実施する前に、制御モデルにわずかな変更を加えました。その後、Scorpion の ECU 向けモデルから C コードを生成しました。

生成したコードをファームウェアや ECU 用の他の低水準インターフェイス コードと統合して、最初はサードパーティ製シミュレーターでテストし、その後実際に Scorpion ハーベスターでテストを実施しました。

この Scorpion プロジェクトの成功を受け、Ponsse のエンジニアは、Scorpion 制御設計からフィルターとモデルコンポーネントを再利用するモデルベースデザインを活用して、Ponsse 製品ラインの他のハーベスター用の組み込みコントローラーを開発しています。

テストと検証

従来の開発ワークフローでは、テストと検証は通常プロセスの終盤に発生するため、設計とコーディングの段階で生じたエラーを特定して修正することは困難です。

モデルベースデザインでは、テストと検証は要件と仕様のモデル化から、設計、コード生成、および統合まで、開発サイクル全体を通して継続的に発生します。モデルで要件を作成し、設計、テスト、およびコードへとトレースすることができます。形式的手法は設計が要件を満たしていることの証明に役立ちます。レポートとアーティファクトを作成して、ソフトウェアを機能安全規格に準拠させることができます。

ケーススタディ: Lear

Lear のハードウェアインザループ テスト。

「BCMプロジェクトでは、Simulink で実行可能な機能モデルを使用して仮想的な統合とテストを行うことで、実装前に要件に関する問題を 95% 以上特定することができました。モデルベースデザインを適用する前はわずか 30% の問題しか特定できなかったことを考えると、これは大きな進歩です。」

— Jason Bauman, Lear

自動車メーカーは、サプライヤーに対して、ECU ソフトウェアにより多くの機能を搭載するように要求しています。自動車のエレクトロニクス システムとパワーシステム系統は次第に複雑さを増してきているため、明確で完全に記述された、矛盾のない要件が必要となっています。しかし、従来の手作業でコードを作成するワークフローでは、あいまいな要求や矛盾する要求が開発プロセスの終盤になって見つかることが多く、納期に間に合わなかったり、予算を超過したりする結果を引き起こしています。Lear Corporation のエンジニアは、モデルベースデザインを適用してボディ制御エレクトロニクス システムを開発、検証、実装することで、これらの問題に対応しています。

エンジニアは、顧客からの要件を分析して、システム全体を室内灯や外部のライト、バッテリー管理、発進制御などのコンポーネントに分割しました。その後、機能的挙動モデルやプラントモデルを開発し、機能テストやユニットテストを実施しました。エンジニアは、モデルカバレッジを解析し、判定カバレッジと改良条件/判定カバレッジ (MC/DC) を含め、満足のいくモデルカバレッジが達成されるまで、テストケース、設計、要件の調整を繰り返しました。

このようにして、チームはほぼ 400 個のユニットモデルを検証した後、C コードを生成し、このコードをソフトウェアインザループ (SIL) テストによって検証しました。このテストでは、ユニットモデルテストのために生成されたテストケースが再利用されました。

Lear のエンジニアは、各ユニットモデルに対して生成したコードを 20 ～ 30 個の機能レベルのコンポーネントに統合した後、さらにこれらのモデルを完全なシステムモデルに統合しました。チームは、顧客の立会いのもとでコンポーネントと完全なモデルのシミュレーションを行い、元の設計仕様であいまいだった点を明確にしました。

その後、MATLAB スクリプトを使用して、ハードウェアインザループ (HIL) テストと車両ベーステストに備えた、テストケースのテストベクターへの変換が自動化されました。チーム全体で約 70 万行のコードを生成し、開発サイクルを通じてテストケースを再利用することで、全体的な開発期間が短縮されました。

モデルベースデザインの始め方

モデルベースデザインに移行する利点を理解する一方で、それにまつわる組織的、ロジスティクス的、および技術的なリスクと課題の発生を懸念する声がチーム内からあがるかもしれません。このセクションでは、モデルベースデザインの導入を検討中のエンジニアリング チームからよく寄せられる質問にお答えし、チームの移行に役立ったヒントとベストプラクティスを紹介します。

Q. モデルベースデザインの導入はエンジニアリングの役割にどのように影響しますか?
A. モデルベースデザインは、制御設計とソフトウェア アーキテクチャにおけるエンジニアリングの専門知識を置き換えるものではありません。モデルベースデザインにより、制御エンジニアの役割は紙での要件を提供することから、モデルとコードの形式で実行可能な要件を提供することへと広がります。ソフトウェア エンジニアがアプリケーション ソフトウェアをコーディングする時間は短縮され、アーキテクチャのモデリング、OS、デバイスドライバー、その他のプラットフォームのソフトウェアのコーディング、およびシステム統合の実施に費やす時間は増えることになります。制御エンジニアとソフトウェア エンジニアは、ともに開発プロセスの初期段階からシステムレベルの設計に関与することになります。

Q. 既存のコードはどうなりますか?
A. 設計の一部となります。つまり、システム モデルに最初からモデリングされたコンポーネントと、レガシー コンポーネントの両方を含めることができます。そのため、システム シミュレーション、検証、およびコード生成を継続しながら、レガシー コンポーネントを段階的に導入することができます。

Q. 推奨されているモデルベースデザインの導入方法はありますか?
A. 新しいアプローチや設計ツールを試す際には、常にリスクとなる要素が伴います。成功するチームはモデルベースデザインを段階的に導入し、プロジェクトを遅延させることなく、プロジェクトに役立つ焦点を絞った手順を行うことで、このリスクを軽減してきました。どのような規模の企業でも、小さなグループのレベルからモデルベースデザインの導入を始めます。通常、1 つのプロジェクトから始め、短期で成功させて、それを基礎とします。経験を積んだ後、モデルベースデザインを部門レベルに展開し、モデルがグループの組み込みシステム開発の中心となるようにします。

次の**4 つのベストプラクティス**は多くのチームを成功に導いています。

- **プロジェクトの小さい部分で実験する。**はじめて導入する際は、組み込みシステムの新しい分野を選び、ソフトウェア挙動のモデルを構築し、モデルからコードを生成することを推奨します。チームメンバーは新しいツールと手法の学習に投資する時間を最小限にして、小規模な変更を行うことができます。この結果で、モデルベースデザインの次の主な利点を実際に確認することができます。
    - 高品質コードを自動的に生成できる。
    - コードがモデルの挙動と一致する。
    - モデルをシミュレーションすることで、アルゴリズムの欠陥をより簡単に発見できる。また、デスクトップで C コードをテストするよりも、詳細を確認しやすい。

- **システムレベル シミュレーションを追加して最初のモデリングにおける成功を基礎とする。**前のセクションで述べたように、システム シミュレーションを用いて要件を検証し、設計の疑問点を調査し、早期にテストおよび検証を行うことができます。システム モデルは高忠実度である必要はありません。インターフェイスの信号が正しい単位で正しいチャネルに接続され、システムの動的挙動を取得していることを確認できるだけの情報があれば十分です。シミュレーション結果により、プラントとコントローラーがどのように動作するかを早期に確認することができます。

- **モデルを使用して特定の設計上の問題を解決する。**チームは、プラント、環境、およびアルゴリズムのモデルをフルスケールで開発することなく、希望する機能を利用することができます。たとえば、チームは動作に使用されるソレノイドのパラメーターを選択する必要があるとします。この場合、ソレノイドの周りに概念的な「制御ボリューム」を描く単純なモデルを、それの駆動力と、それが動作する環境を含めて開発することができます。チームはさまざまな極限の運用条件をテストし、方程式を算出することなく基本的なパラメーターを算出できます。このモデルはその後、異なる設計の問題に、または将来のプロジェクトで使用するために保存しておくことができます。
- **モデルベースデザインのコア要素から始める。**モデルベースデザインの導入当初に得られる利点として、コンポーネント モデルやシステム モデルの作成、シミュレーションを使用した設計のテストと検証、プロトタイプ作成とテストのための C コードの自動的生成などの機能を挙げることができます。その後、高度なツールや実践を検討し、モデリング ガイドライン、コンプライアンス チェックの自動化、要件のトレーサビリティ、およびソフトウェアビルドの自動化を導入することができます。

ケーススタディ: Danfoss

Danfoss 社の VLT® AutomationDrive FC302。

「我々は、新しいエンジニアを配置し、新しい設計プロセスを採用したにも関わらず、モデルベースデザインを用いて最初のソーラーインバーターのプロジェクトをスケジュールどおりに完了しました。我々の 2 つめのプロジェクトでは、開発期間は実に 10 %から 15% 短縮されました。」

— Jens Godbersen, Danfoss

増大する製品需要に応えるため、Danfoss のパワー エレクトロニクス グループは新しいエンジニアを雇用し、それまでは手作業でのコーディングに頼っていた組み込みソフトウェア開発プロセスを見直しました。従来の開発プロセスと手作業でのコーディングでは、エラーがハードウェア プロトタイプと認定テストまで検知されないまま残っていました。

Danfoss は新しいプロセスが必要であることは理解していましたが、モデルベースデザインを採用すると納期遵守が危うくなるのではと懸念していました。チームを軌道に乗せるには時間がかかります。また、新製品であるソーラーインバーターの作業も既に始まっていました。モデルベースデザインプロジェクトの納期に影響を与えること無く、モデルベースデザインを開発中に導入する必要がありました。

MathWorks のコンサルタントと作業することで、Danfoss は最初にモデルベースデザインの採用を確実に成功させる計画を作りました。Danfoss のエンジニアは、MathWorks のエンジニアが主催する Simulink、Stateflow®、および Embedded Coder® に関するオンサイトのトレーニングコースに参加しました。

チームは次に、手作業でコーディングされた既存のソフトウェア コンポーネントを再構築して、パイロット プロジェクトを完成させました。パイロット版として、彼らはモデル化、シミュレーション、およびコード生成という、モデルベースデザインの 3 つのコア機能に焦点を合わせることにしました。パイロット プロジェクトを完成させた後、チームは新しいソーラーインバーターの開発で完全にモデルベースデザインに移行しました。

MathWorks のコンサルタントは、毎週電話で導入のための最善の方法についてアドバイスしたり、モデルの早期バージョンに関するフィードバックを提供したり、またモデルを最大限に再利用して生成コードの性能を改善するため、業界のベスト プラクティスをチームに適用する支援を行いました。

チームはスケジュールどおりに開発を完了しました。また、準備段階でチームが広範囲のシミュレーションを行っていたため、テストと認定の活動はスムーズに進みました。

新しいワークフローは成功し、会社全体でモデルベースデザインに関わるエンジニアの数が増加しました。また、将来のプロジェクトで再利用できるモデルのライブラリとナレッジベースも構築されました。

まとめ

要件の記録から設計、実装、およびテストに至るまで、システムモデルが開発の中心になります。これがモデルベースデザインの本質です。このシステムモデルでは、以下を行うことができます。

- 設計を要件に直接リンク
- 共有の設計環境で共同作業
- 複数の what-if シナリオをシミュレーション
- システムレベルでの性能の最適化
- 組み込みソフトウェアのコード、レポート、およびドキュメンテーションを自動的に生成
- 早期テストにより、エラーを早期に検知

「3 年前、SAIC Motor には組み込み制御ソフトウェアを開発した豊富な経験はありませんでした。そのため、効率的で実績のある開発手法のモデルベースデザインを選択しました。この手法のおかげで、エンジニアのチームが、非常に複雑な HCU 制御ロジックを開発することができ、プロジェクトをスケジュールよりも早く完了することができました。」

— Jun Zhu, SAIC Motor Corporation

モデルベースデザインのツール

基本製品

MATLAB

データの解析、アルゴリズムの開発、および数学的モデルの作成

Simulink

組み込みシステムのモデル化とシミュレーション

要件の記録および管理

Simulink Requirements™

要件の作成、管理、およびモデル、生成コード、テストケースへのトレース

System Composer™

システムおよびソフトウェア アーキテクチャの設計と解析

設計

Simulink Control Design™

モデルの線形化と制御システムの設計

Stateflow®

ステートマシンおよびフローチャートを使用した判定ロジックのモデル化とシミュレーション

Simscape™

マルチドメイン物理システムのモデル化およびシミュレーション

コード生成

Simulink Coder™

Simulink モデルと Stateflow モデルからの C コードと C++ コードの生成

Embedded Coder®

組込みシステム用に最適化された C コードと C++ コードの生成

HDL Coder™

FPGA と ASIC 設計用 VHDL コードと Verilog コードの生成

テストと検証

Simulink Test™

シミュレーションベースのテストの開発、管理、実行

Simulink Check™

スタイルガイドラインやモデリング標準への準拠性を検証

Simulink Coverage™

モデルおよび生成コードのテストカバレッジを測定

Simulink Real-Time™

リアルタイム アプリケーションのビルド、実行およびテスト

Polyspace® 製品

重大なランタイムエラーがないことを証明

関連情報

mathworks.com には、モデルベースデザインの短期間での学習に役立つ幅広いリソースが用意されています。以下のリソースから始めることをおすすめします。

インタラクティブ チュートリアル

MATLAB 入門

Simulink 入門

Stateflow 入門

Web セミナー

新規ユーザー向け Simulink (36:05)

Simulinkによる制御設計入門～PIDのチューニングから最新例題まで～ (31:45)

制御システム設計のスピードと範囲を飛躍的に発展 (51:03)

ソーラーインバーターをモデル化し、シミュレーションして、コードを生成 (45:00)

オンサイトまたは自己学習形式のトレーニングコース

MATLAB 基礎

Simulink 基礎

MATLAB と Simulink による制御設計

その他のリソース

技術コンサルティング サービス