高 ASIL ソフトウェアの城壁の築き方

ソフトウェア・ディファインド・ビークル（SDV）、ドメイン／ゾーン ECU、大規模 SoC などのニーズの変化や技術の進展により、自動車の中身は「多数の ECU」から、「巨大 SoC にソフトウェアを混載した世界」に急速にシフトしています。
1 チップ上に安全制御も快適機能もクラウド連携も同居するようになり、「ソフトウェアをどうパーティショニングするか」は、ソフトウェア開発の重要なテーマの 1 つになっています。
ソフトウェアの安全設計では、高い ASIL の安全機構を必要なタイミングで確実に動作させることが必達要件となっていますが、設計現場からは、「対策が後回しになってしまう」「膨大な作業がボトルネックになっている」という声をよく聞きます。
今回のメルマガでは、このソフトウェアパーティショニングの効率的なアプローチをご紹介します。

ソフトウェアパーティショニングの効率化

最近の大規模 SoC やドメインコントローラーでは、**1 チップの中に ASIL D～QM が集約されるケース**が当たり前になってきました。
この状況でソフトウェアパーティショニングを考えるときの近道は、
① まず「高 ASIL 側をどう守るか」を設計し
② 次に「QM／低 ASIL をどこまで統制するか」を決める
という“順序”で設計することです。
「高い側を守る」と「低い側からの侵害を減らす」は表裏一体ですが、**スタート地点を高 ASIL 側**に置いた方が、設計もレビューも格段にやりやすくなります。

「高い側を守る」→「低い側からの侵害を減らす」の順序の理由

背景として、近年の E/E アーキテクチャは次のような変化が進んでいます。

- 大規模 SoC／ドメインコントローラー上に
  - パワトレ／シャシーの安全制御（ASIL D/C）
  - ボディ・インフォテインメント系の快適機能（QM）
  - OTA アップデータ、ログ収集、クラウド連携アプリ（多くは QM）が同居する。
- 1 ECU 内のソフトウェアコンポーネント数は**数十～数百のオーダー**になり、機能安全上重要なものはその一部にとどまっています。

このような環境で求められる安全設計は、「影響分離（Freedom from Interference）」です。
ここでは分かりやすく、次の 3 つに整理できます。

1. **空間的分離**：メモリ・アドレス空間・デバイスアクセスの分離
2. **時間的分離**：CPU 時間・スケジューリング・レスポンス時間の分離
3. **論理的分離**：インターフェース・状態遷移・責務の分離

QM／低 ASIL 側の品質をどれだけ上げても、

- 高 ASIL のメモリに自由に書き込める
- CPU を占有して高 ASIL タスクの処理時間を圧迫する
- 安全機構の重要な API が丸見えになっている

のような設計が残れば、影響分離は達成できません。
一方で、**高 ASIL の「城壁」（空間・時間・論理の防御ライン）を先に決めてしまえば**、

- QM／低 ASIL がどこまで壊れても、高 ASIL にはここまでしか影響しない
- だから QM 側のターゲット品質はこのくらいでよい

というふうに、テストレベルやレビューの“メリハリ”を論理的に決められ、過剰品質や手戻りを防げます。
プロジェクト計画やアーキテクチャ設計の視点で言えば、
「コンポーネントごとの品質目標を決める前に、**安全上のリスク構造と分離戦略を設計せよ**」ということになります。
このような考え方が、”**安全設計は重点思考**”と言われる所以です。

「高い側を守る」→「低い側からの侵害を減らす」の具体例

① まず「高 ASIL 側をどう守るか」を設計する

例として、ドメインコントローラー上に次のソフトが同居しているケースを考えます。

- ASIL D：ブレーキ・トルク制御アプリ
- ASIL B：横滑り防止、安定化制御
- QM：UI 連携、ログ、OTA 用クラウド連携アプリ多数

ここで最初にやるべきは、**高 ASIL ソフトウェアを中心に防御ラインを引くこと**です。

- **空間的分離**
  - ハードウェアメモリ保護（MMU/MPU）やハイパーバイザを使い、ASIL D/C のコード・データ・スタックを専用保護領域に配置
  - デバイスアクセス（ブレーキアクチュエータ等）も、高 ASIL 専用ドライバを通す構造にして QM から直接触れないようにする
- **時間的分離**

- 高 ASIL タスクに最優先の優先度と CPU マージンを割り当て
- QM タスクにはタイムスライス制限、ウォッチドッグ監視、オーバラン検出を導入
- 過負荷時は「QM パーティションだけを止める／再起動する」ポリシーを定義

- **論理的分離**
  - 高 ASIL が受け付けるコマンド・パラメータは、レンジチェック・整合性チェック・状態遷移チェックを通さないと採用しない
  - 安全関連の決定（トルクリミット、フェイルセーフモード移行など）は、高 ASIL の内部ロジックだけで完結させる

この時点で、「城壁の内側」に何を置き、「外側からどのような形でしかアクセスさせないか」が見えてきます。

② 次に「QM／低 ASIL をどこまで統制するか」を決める

次に、**大量のソフトウェアコンポーネントをどう整理するか**を考えます。

- コンポーネント群を、以下の観点でグルーピングします。
  - 安全関連（ASIL D/C/B）
  - 安全に間接影響する QM（診断、ログ、OTA など）
  - 安全とほぼ無関係な QM（UI、情報提供等）
- グループごとに、次の設計を決めていきます。
  - どのパーティション（空間）にまとめるか
  - どの優先度帯／タイムスライス（時間）を与えるか
  - どの API セットだけを見せるか（論理）

例えば、

- OTA 更新・ログ収集など「事故や乗員へのリスクに関連しない機能」は、ひとつの QM パーティションにまとめる。
  - 高 ASIL に近づけないメモリマップ
  - 低めの優先度
  - 制限された API のみ
- 逆に、診断経由で安全パラメータに間接アクセスし得る QM ソフトの扱いを決める。
  - 高 ASIL のゲートウェイを経由する構造
  - インタフェースレベルのテストやフェイルセーフ動作を確認

これにより、コンポーネント数が多くても、
「このパーティションは、壊れても高 ASIL にはここまでの影響」
「必要なコーディングルール・レビュー・テストはこのレベル」
という**筋の通ったルールセット**を定義できます。

まとめ

ソフトウェアパーティショニングで迷ったときは、次の順番を推奨します。

1. **高 ASIL の「城壁」を設計する**
    - 空間的分離
    - 時間的分離
    - 論理的分離
2. **その城壁の外側に、QM／低 ASIL の多数コンポーネントの統制を決める**
    - どのパーティションに束ねるか
    - どの品質レベル・テストレベルを要求するか。

この順番を守るだけで、

- 「どこまでやれば十分か？」
- 「QM をどこまで厳しく見るべきか？」

といった日々のお悩みが、かなりスッキリしてくるはずです。
そして何より、このテーマは、アーキテクトやリードエンジニアだけでなく、現場のソフトウェアエンジニア一人ひとりが設計レビューや実装段階で関わることができます。
「分離戦略から逆算した設計・テストの考え方」をチーム内で共有できれば、SDV 時代の巨大 SoC 開発でも、自分たちのコードが"どこまで高 ASIL を守れているか"の重点思考の開発を進められるようになるはずです。

機能安全実装ワークショップでは、このようなソフトウェアパーティショニングの考え方を、事例演習を通して学んでいただけます。
受講者の方々からも、
「ソフトウェアパーティショニングの技術に関して、新たな気づきを得ることができた」
「ソフトウェアパーティショニング設計時の保護対象に対する安全分析の演習が、実際の開発で役立つ」
といったお声をいただいています。
また、この他にもシステム領域やハードウェア領域の設計を進める上でのヒントとなるコースも用意しています。
是非、機能安全対応の充実化や効率化にご活用いただければ幸いです。

【関連トレーニングのリンク】

トレーニング情報：機能安全実装ワークショップ

※次回募集の案内をお待ちください。

※プライベートトレーニングのご要望などございましたら、「HP お問い合わせ」までお願いいたします。

過去のメルマガ：ソフトウェアパーティショニング設計の実践アプローチ

2025/11/26 吉川 初芽