■省リソース環境でのオブジェクト指向開発

組込みシステムのソフトウェア開発ではコストの制約などから、まだまだアセンブリ言語やC言語が主流の分野があります。
使用するプロセッサに対応したC++のクロスコンパイラが存在しないこともあります。
メモリも贅沢に使えません。数Kバイトとか数100バイト程度のメモリでヒープやスタックを贅沢に使えるはずもありません。必然的にいろんなところから参照、更新する1ビット単位のフラグや変数を大域的に使用してしまいがちです。
RTOSを使用する文化も予算もない場合、μITRONやμT-Kernelなども採用されません。

このような恵まれない貧弱な環境のことをここでは「省リソース」と呼びましょう。

さて、組込みシステムの開発でオブジェクト指向設計が本当に必要な分野は、実はこのような省リソース環境での開発だと言っても過言ではありません。
ハードウェアの変更を含むモデルチェンジに短期間かつ高品質に対応することがこの場合のオブジェクト指向設計の目的です。

旧来型の「組込み」はチームで開発することも少なく、再利用性を考えて設計する時間もありません。
ハードウェアのコストダウンでチップだけを変更して製品仕様が基本的に前のモデルとほとんど変わらず、ソフトウェアの変更工数を少ししか確保できない経験をお持ちの方も少なくないでしょう。

このような開発で実際にやることは前のモデルのソースプログラムをコピーして手直しする「作業」です。ポートのアサインがほとんど変わっていないから手直しは少し。ハード屋さんはこんなことを考えます。そのポートの先に仕様が変わったチップが繋がっていても「そこだけ」変更すれば済むと思い込んでいます。しかし、そのような柔軟な作り方をしていませんから変更は「そこだけ」では済みません。修正個所が広範囲に及ぶことが少なくありません。

では、どうやったら修正個所を少なくすることが出来るでしょうか。
それはオブジェクト指向設計で実装する部分を少しずつ増やしていくことです。
「そこだけ」の対象毎に開発するわけです。
チップやプロセッサに依存する処理を抽象化してアプリケーションと分離して実装すれば、ハードウェアの構成が変わっても局所的な対応が可能になります。ポートの割付や論理に依存しない抽象化を行えば修正個所も限定できます。

オブジェクト指向というとUMLで設計してC++やJAVAなどで実装しなければならないという呪縛があるとすれば、その固定観念を捨て去ることが必要です。そのような固定観念では最初からオブジェクト指向設計など出来るはずもありません。

難解な概念を駆使して設計する必要はほとんど不要です。

- 抽象化
- カプセル化

・ 情報隠蔽

この程度の着目点で十分です。

抽象化するためには、まずブロック図を書きましょう。
・ ハードウェアに依存する部分
・ アプリケーションとしての機能
・ メインループの考え方
・ タイマ処理の考え方
・ 割り込み処理の考え方
・ 状態遷移の考え方
・ タイミングシーケンス制御の考え方
・ ユーザインターフェースの機能
・ コアな機能と拡張機能の分離

次にブロック図で分けたブロックはカプセルという呼び方に換えます。カプセルはデータと処理をひとまとめにして抽象化するという概念ですが、データと処理の記述を一つのソースファイルにまとめることにより実現します。大きな一つのソースファイルになっても気にしないことです。そのようなデメリットよりもカプセルとして独立しているということの方がここでは重要です。データを隠蔽するために外部参照は禁止し、最低限のインターフェースを持つ関数やサブルーチン（マクロだって良い）を用意し、それらを他のカプセルとのインターフェースとします。公開用のインターフェースに対する他のカプセル向けの「公開用ヘッダファイル」も用意します。

なお、RTOS のタスクとカプセルは似ているようで違います。RTOS でのタスク分割は並列処理の視点で行いますがカプセルは機能単位で分割します。省リソース環境（で十分な機器）での並列処理は通信処理、ユーザインターフェース処理、タイマ処理程度しかありません。本当に必要な並列処理は割り込み処理で実装しますが、割り込み処理と通常の処理が関連している場合は一つのソースファイルにまとめてそれぞれがカプセルの構成要素となるようにします。