プリンタ開発へのZIPC適用事例紹介

セイコーエプソン株式会社
CS・品質保証室　品質技術開発部
主任　江津　英幸

１．はじめに

今回この執筆を行うにあたって最初に断っておく必要がある。私がZIPCと最初に出会ったのは約３年前である。それまではメカ系の業務を行っており、ファームウェアという言葉すら知らないに等しかった。なのでこれを読んでいる多くの方にはいまさらと思われる内容も多いと思うが、組込みシステム開発に少し特異な立場から関わっているものの意見として参考にしていただければ幸いである。

当時私は、3DCADやCAEといったメカ関係のツールを用いた設計プロセス改革に従事していた。弊社の主力商品であるプリンタ、プロジェクタなどの設計を効率的に行い、コスト削減、納期短縮といった課題を達成するためである。このための方法の一つとして仮想メカシミュレーションの技術確立を担当することとなった。仮想メカシミュレーションとは、PC上で定義した仮想のメカ体を使ってメカの制御系を評価する技術である。当初はメカ設計側の課題として取り組み始めた案件であった。ZIPCとの出会いはこのときである。ZIPCは仮想メカと接続するしくみやPC上でのシミュレーションを行う機能を持っていた。これらの機能を評価しメカ設計者、ファーム設計者と議論を重ねていく中で、最初のステップとしてファーム設計の効率化という観点からまずはファーム設計にZIPCを導入していこうという結論に至った。

対象としたプリンタは、様々な業界の製品同様、多機能化・高機能化が進んでいるが、製品設計にかけられる時間は短縮傾向にある。また製品設計のなかでファーム設計はメカ設計の後工程に位置するためファーム設計の納期遅れが製品全体の開発日程に影響を及ぼしかねない。このような状況を打破し、効率的な設計を行えるのではないかという期待からZIPCの導入に踏み切ることとなった。

本稿ではZIPC適用にあたっての活動について、プロセス改革の推進者としての観点から報告する。

２．製品開発へのZIPCの適用

2.1．メカ制御用ファームウェアにおける階層

今回ZIPCの適用対象としたプリンタはメカ制御が必要な製品である。メカ制御を行うファームウェアは、様々なレベルのイベントの処理が必要となる。ユーザー操作によるイベント、モーター・センサーなど機器内部で発生するイベント、ユーザーの安全を確保するための例外発生イベントなど数え上げればきりがない。これらを効率的に扱うため設計段階では様々な工夫を行っている。

図１にメカ制御という観点からファームウェアを見た場合の階層を示す。上から、製品としての仕様を実現するためのアプリケーション層、製品の機能を満たすために一連の動作（プリンタでいえば印刷、クリーニングなど）を行うメカシーケンス制御層、モータなど個々のアクチュエータの動き方（印字ヘッドの速度、位置決め等）を制御するためのアクチュエータ制御層、エレキ/ファームの境界部分を効率的に設計するためのエレキ-ファーム協調設計層である。これらの階層はそれぞれの役割が異なるため、インターフェース部分さえ決定することができれば独立した設計プロセス及び検証プロセスを構築することが可能であると考えている。それゆえプロセス改革推進の立場からはそれぞれの階層に対して最適な設計プロセスを構築していく必要がある。

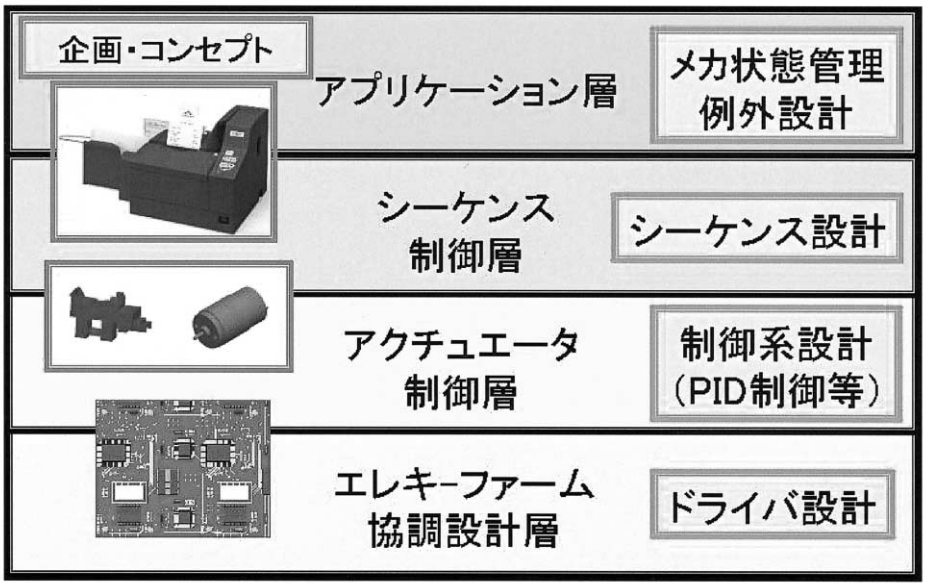

図1　メカ制御用ファームウェアの階層

2.2．ZIPC適用状況

今回対象とする製品は、銀行向けシステムに組み込むためのプリンタである。この製品はこれまで開発したプリンタの中でもっとも多くの機能を搭載している。それゆえ搭載するファームウェアもかなり複雑になることが予想されていた。

ZIPCを適用するにあたって当初は仮想メカシミュレーションの推進といった立場からシーケンス制御層への適用を考えていたが、いくつかの課題および設計者からの要望もありアプリケーション層に属するタスクへの適用を行うこととした。このタスクは他タスクから来る様々なイベントに対してメカの状態を管理し、メカ制御タスクにイベントを発行するといった制御の中枢になるタスクである。このタスクの設計にZIPCを適用することとした。

プリンタのファームウェア開発においては、仕様記述の手段として状態遷移図や状態遷移表をすでに用いていたため、設計手法及びZIPCの操作については設計者の呑込みが早かった。設計手法について設計者から「これまでシステムを状態遷移で記述しようとしても図や表が巨大化してしまって全てを表現するのは難しかった。この手法はこれに対する１つの答えになりうる」というような感想をもらっている。

推進側としては以下のような活動を行っている。

- ・仕様レビュー等の設計活動に参加
- ・手法や生成コードの概要といった内容について勉強会の実施
- ・製品用の状態遷移表を作成するにあたってのサンプル作成とアドバイス
- ・ZIPCの不具合検証、ZIPCサポートとの窓口

2.3．実施した設計プロセス

今回実施した設計プロセスを図２に示す。ZIPCは以下の用途で適用している。

1．要求仕様書からもれ・ぬけ・矛盾を発見し、全体の整合性を高めるための状態遷移表の作成
2．設計者の意図通りの動作をするか確認するための動的シミュレーション
3．作成した状態遷移表から自動コード生成（2001ジェネレータ使用）

当初は要求分析段階での適用も考えたが、ZIPC自体要求分析を行うための機能はなく設計意図の決定プロセスなどを盛り込むことができなかったため今回は適用を見合わせている。

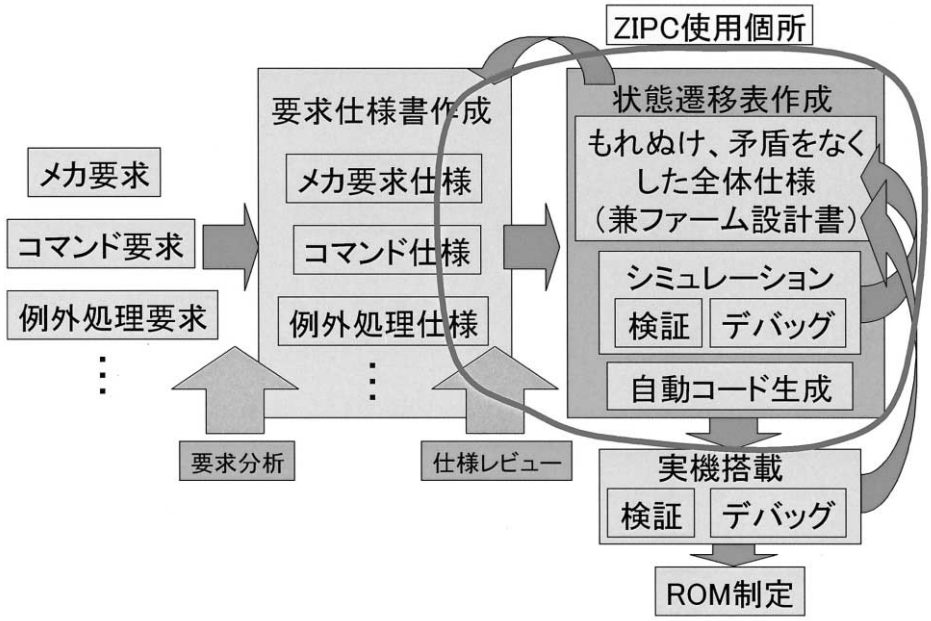

図2　実施した設計プロセス

3．適用結果と効果

3.1．適用結果

今回作成したZIPCデータの内訳を以下に示す。

- 状態遷移表シート数：29
- 総セル数：約3450
- ユーザー定義関数：約200
- 生成コード量：約92Kバイト（コンパイル後）

上記は製品向けに作成したものである。これ以外に検証を行うための環境を構築している。今回は単体のタスクを対象としていたため、ATVは使用せずに独自に検証環境の構築を行った。

コード生成に使用したジェネレータはZIPC2001ジェネレータで、FULL型に固定して行っている。生成されたコード量は前機種の同一タスクの約1.3倍となっている。当初のベンダー情報では、ZIPC初期トライアルで約1.7～2倍程度のコード生成量になるといわれていた。これを見越して、今回のプロジェクトでは前機種の約1.4倍程度を目標に活動を行った。結果としては、ZIPCの記述等で苦労することなく目標をクリアすることが出来た。これについては今回追加された機能がある一方で仕様を見直すことで単純化できた部分もあり、単純に前機種と比較することは難しいと考えている。

3.2．適用効果

今回の適用において一番大きな効果があったのは不具合件数である。実機搭載後の不具合件数で比較すると前機種の約10分の1以下に抑えることが出来た。これについては、機能追加や仕様変更など単純に比較できない部分もあるが、ZIPCを導入したことの効果が最も大きいと考えられる。

今回のプロセスでは、最初に要求仕様書を作成しこの仕様書をもとに状態遷移表の作成を行っている。この課程で要求仕様書に書ききれなかったもれ・ぬけ・矛盾を発見することが出来、要求仕様書にフィードバックすることが出来た。状態遷移表はこの形式で記述しセルを埋めるだけでも効果があった。

さらにこの状態遷移表に対して動的シミュレーションを実行することで効果的に問題を発見することができた。これまで試作機ができる

までデバッグが難しい状態であったものが、試作機レスの状態でタスク単体のデバッグが出来たことは大変有効であった。シミュレーションを行うにあたっては、VIPを使って外部の環境と連動することでより効率的な検証を行うことが出来た。

作成した状態遷移表はファーム仕様書兼コード生成用の状態遷移表として後工程まで利用した。こうすることで仕様書とコードの一元化が可能となった。仕様変更が発生した場合も状態遷移表レベルで変更することで効率よく作業を行うことが出来た。

これらが効果的に作用することで実機搭載時の不具合件数を減少することが出来たと考えている。

もう一つの効果としては納期についてである。今回対象としたタスクは他タスクの仕様が固まった後でないと仕様が固まらない性質のものであったため、他タスクに比べて要求仕様の作成にかなりの時間をかけている。これに加えてZIPCの習得時間が必要であった。このような状態であったが他タスクに比べても最終的な納期の遅れは無かった。これはZIPCを使うことで仕様の検証とコードの作成が同時進行で出来たおかげであると考えている。

これらの成果に関して設計者からは「不具合件数に関しては感覚的に雲泥の差がある」という感想をもらっており、プロジェクトリーダーからは「プロジェクト後半での問題の発生が少なくすばやく収束できた」という感想をもらうことが出来た。

4．まとめ

プリンタのファームウェア開発にZIPCを適用した結果、不具合件数、納期などで明確な成果を出すことが出来た。これらの成果に基づき次機種以降もZIPCを適用しており、適用範囲を広げている。

4.1．今後の課題

今回はアプリケーション層のタスクを対象にZIPCの導入と設計プロセスの再構築を行った。このなかでこの階層でのファームウェア設計に関するノウハウを多少なりとも構築することが出来た。今後はこの中で得られた成果を他部署や他事業部に水平展開していきたい。また他の設計階層においても同様にその階層にあった最適なプロセスを考え構築していきたい。

また当初の目的であった仮想メカシミュレーションについては、企画⇒メカ設計⇒ファーム設計という流れの中でもっと広く捉えていく必要性を感じている。例えば例外ということを考えるとファーム設計段階でいくらもれ・ぬけが防げるといっても、そもそもその前の仕様段階でもれているものは入れ込むことはできない。設計上流で、いかにもれ・ぬけのない仕様を作っていけるか、メカ、ファーム、エレキの各設計者がいかに上流で協業していけるか。これが今後の一番大きな課題になると考えている。

4.2．要望

今回のプロジェクトを通して様々な要望があった。これらはZIPC Ver.9.0で多くを対応していただいている。さらにZIPCを使いやすくしていくために以下を要望したい。

チェッカー

チェッカーは状態遷移表を記述していく上で大変有効な機能である。これをさらに有効なツールにしていくためにチェック品質の向上をお願いしたい。また、表示される内容について現状では全ての内容が一覧で表示されるが、一度にたくさん表示されるとどうしても1つずつ確認しにくくなってしまう。優先度が高いものから絞って表示されるなど工夫していただけるともっと使いやすくなる。

マニュアル

それぞれの機能についてどのような意図でその機能があるか、どのような使い方を想定しているかということまで記述していただきたい。現状のマニュアルではこれらの記述が少ないため、ユーザーが試行錯誤で使ってみてできるかどうか判断する場合が多い。これだと事前に想定していない使い方をしてしまい、バージョンアップ等で使用できなくなりユーザー側から苦情が出ることがある。

検証機能（ATV）

ATVについては今回のプロジェクトでは使用

していない。現状のATVはシステムの検証用であって今回のようにタスク単体のみ作成したものの検証には向いていないという印象を受けている。実際の検証の順序はタスク単体⇒システムとなる。タスク単体で有効な検証が行えるような機能があれば今後は使っていきたい。

ソフトウェア品質

ソフトウェアにバグはつきものであるが、少し多すぎる気がする。品質管理の体制は改善していただいているようなので今後もよりいっそうの努力を続けていただきたい。

５．最後に

この場を借りて、今回このような機会を与えていただくとともに、こちらから出す様々な難題に迅速に対応していただいているキャッツ株式会社殿、今回の適用で社内的にはほとんど実績のなかったZIPCの導入を快く受け入れてくれた事業部の方々、このツールを導入するにあたって根気よく見守り、協力してくれた部内の方々に感謝するとともに、お礼を述べさせていただく。