2009－SE－163 (14)
2009／3／18

FPGAとソフトウェアにおける協調動作の整合性に関する評価手法の提案

加藤 淳†, 狼 嘉彰†

電子機器システムを開発する場合，システムをFPGAやMPU上で動作するソフトウェアといった構成要素に分割し，FPGAやソフトウェアが連携し協調動作することでシステムが満たすべき機能を実現する．しかし，FPGAとソフトウェアにおける協調動作の不整合は，それらを組み合わせたシステム試験まで検出されないことが多い．本稿では，FPGAとソフトウェアが搭載された電子機器システム開発の初期フェーズであるアーキテクチャ設計段階において，FPGA 要求仕様やソフトウェア要求仕様を基に，協調動作の整合性について評価する手法を提案する．提案手法を試行した結果，FPGA とソフトウェアの協調動作に関して，異常時における処理の抜けなどを検出することができた．

An Approach for Verifying Correctness of Co-operations between FPGA and Software in Electronic System

Atsushi KATOH†, Yoshiaki OHKAMI†

Electronic systems are decomposed into processing elements such as FPGA, Software on MPU and satisfied their functions with co-operations between FPGA and Software, when electronic systems are developed. In-correctness of co-operations between FPGA and Software can't generally be detected until a system test where FPGA and Software are composed. This paper proposes an approach for verifying the correctness of co-operations between FPGA and software in electronic system based on their requirement specifications at architectural design phase. Missing of handlings under abnormal conditions could be detected concerned with co-operations between FPGA and software by attempting proposed an approach.

1. はじめに

近年，LSIや組込みソフトウェアが搭載された電子機器システムの市場規模は拡大し，自動車，携帯電話，医療機器，人工衛星，ロケットなど社会の隅々にまで浸透している[1][2]．また，電子機器システムにおいて，開発の自由度および開発期間の短さに加え，集積度および性能面の向上から，FPGA（Field Programmable Gate Array）の採用が急速に進んでいる[3]．その反面,放射線照射装置の不具合による事故など，電子機器システムの不具合は，社会システムに大きな影響を与えるまでに至っている[4][5]．信頼性の高い電子機器システムを実現することは，安全な社会システムを実現する上で重要な課題となっている．

電子機器システムを開発する場合，システム・エンジニアリングに基づき，システムを構成要素に分割し各構成要素が連携し協調動作することでシステムが満たすべき機能を実現する[6]．そのため,電子機器システムの構成要素であるFPGAやMPU上のソフトウェアによる協調動作に不整合が生じる場合,電子機器システムの信頼性に大きな影響を与える．図 1に，電子機器システム開発における FPGAとソフトウェアに関する協調動作の不整合を示す．電子機器システムを開発する場合，まず，ユーザの要求に対し要求分析を行い，システムに要求される機能・性能などをシステム仕様として明確化する．次に，システム仕様を機能に分割・具体化する機能設計を行い，システムを構成するFPGA やソフトウェアという要素に対し各機能を割り付ける

†慶應義塾大学大学院システムデザイン・マネジメント研究科
Keio University Graduate School of System Design and Management

物理設計を行う. そして, FPGA 要求仕様書, ソフトウェア要求仕様書, インタフェース仕様書という形で構成要素の仕様および構成要素間のインタフェースを明確化する. これをアーキテクチャ設計という[6]. ここで, アーキテクチャ設計が不十分で機能の具体化に不備がある場合, FPGA 要求仕様書やソフトウェア要求仕様書に定義される機能仕様が曖昧なことがある. 各要求仕様書において機能仕様が曖昧な場合, それらを基に開発を行う開発者の思い込みや勘違いなどから, 開発プロダクトがシステムとして意図しないものとなる可能性がある. また, FPGA とソフトウェアの協調動作についても同様に, FPGA 要求仕様書やソフトウェア要求仕様書に定義される機能仕様が曖昧な場合, それぞれの開発プロダクト間の協調動作が整合しない可能性がある.

FPGA とソフトウェアの協調動作に関する不整合について, それぞれの開発工程にて発見されることは少なく, FPGA とソフトウェアを組み合わせたシステム試験で発見されることが多い. その理由のひとつとして, FPGA およびソフトウェアの開発が独立して進められ, それぞれの開発者間のコミュニケーションが少ない場合が多いことが考えられる. 協調動作の不整合によっては, アーキテクチャ設計への手戻りが発生することがあり, その場合, 開発コストの増加や開発遅延を引き起こす. また, アーキテクチャ設計には戻らず, FPGA とソフトウェアの協調動作の不整合をそのまま受け入れる場合, システムにとって何らかの制約が生じる可能性がある. 従って, FPGA とソフトウェアとの協調動作については, 電子機器システム開発の初期フェーズであるアーキテクチャ設計にて, 各開発工程のインプットとなる FPGA 要求仕様やソフトウェア要求仕様から検出する必要がある.

本稿では, 電子機器システムの構成要素である FPGA とソフトウェアに関する協調動作の整合性について着目した. そして, 電子機器システム開発の初期フェーズであるアーキテクチャ設計段階において, FPGA やソフトウェアの要求仕様を基に協調動作の整合性を評価する手法を提案した. また, SD メモリ・カードとのインタフェースを担う FPGA およびソフトウェアの各要求仕様を対象に本手法を試行した. その結果, FPGA とソフトウェアの協調動作に関して, 異常時における処理の抜けなどを検出することができた. 提案手法に対する考察の結果, 協調動作が満たすべき性質の抽出は, 評価者の経験やスキルに依存する課題なども識別することができた.

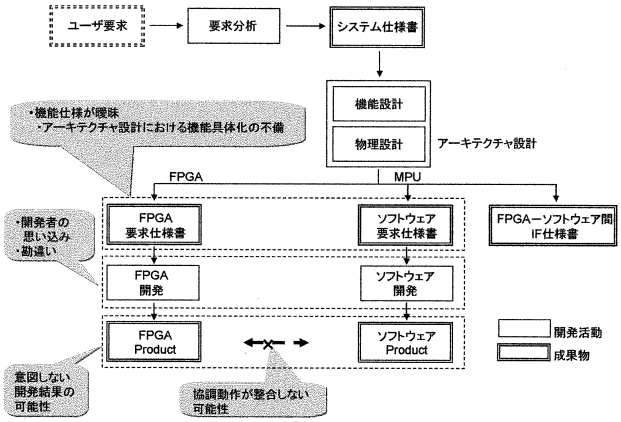

図 1 FPGA とソフトウェアにおける協調動作の不整合

2. 関連研究

2.1. FPGA とソフトウェアにおける協調動作評価手法の課題

FPGA とソフトウェアにおける協調動作の不整合については, FPGA とソフトウェアを組み合わせたシステム試験にて検出される場合が多い. 電子機器システム開発において, システム試験に至るまでに FPGA とソフトウェアの協調動作の整合性を網羅的に評価する手法が確立されていないためである. また, システム試験終了後においても, 電子機器システムに協調動作の不整合が潜在してしまう可能性もある.

電子機器システムのアーキテクチャ設計過程において, 開発者や有識者の経験やスキルに基づくFPGA要求仕様書やソフトウェア要求仕様書のレビューが実施される. この手法は, 比較的短時間で協調動作に関する広範な評価が可能であるが, 人手による評価につき複雑な協調動作の不整合については検出が困難で, また, 網羅的な評価は不可能である. 近年, アーキテクチャ設計においては, EDA (Electronic Design Automation) ツールを用いた FPGA とソフトウェアの協調動作に関するシミュレーションが適用され始めている[7][8]. これらの手法は, 協調動作に関する高速なシミュレーションが可能で, 開発の早期に性能を含めたシステム試験相当の評価を実施できる場合がある. しかし, ツールによって使用できる FPGA デバイスや RTOS に制約があるなど汎用性が低く, また, 現状, ツール自体が非常に高価であるため, 広く普及するまでには至っていない. 更に, シミュレーション技術にて協調動作を評価するため, 試験ケースとして想定しない事象については確認ができず, 網羅的な評価は不可能である.

FPGA およびソフトウェア開発過程においても, EDA ツールを用いた FPGA とソフトウェアの協調動作に関するシミュレーションが実施される場合がある. この手法では, FPGA の HDL (Hardware Description Language) コードや

ソフトウェアのCやC++などのソースコードを用いた協調動作のシミュレーションが可能だが，FPGAとソフトウェアの動作クロックの違いからシミュレーションに多くの時間を要する．また，FPGAとソフトウェアの開発が独立して進められる場合が多いため，HDL コードおよびソースコードを用いた評価自体の実施が困難な場合があり，広く普及するまでには至っていない．更に，シミュレーション技術にて協調動作を評価するため，網羅的な評価は不可能である．

FPGA とソフトウェアを組み合わせたシステム試験においては，実プロダクトを用いて協調動作を評価することが可能だが，FPGA とソフトウェアの協調動作の不整合を発見した場合，その改修に大きなコストを要する[9]．また，シミュレーションと同様に，試験ケースとして想定しない事象については確認できず，網羅的な評価は不可能である．そのため，システム試験終了後においても，協調動作の不整合が潜在する可能性がある．

2.2. モデル検査を適用した協調動作評価手法

FPGA とソフトウェアの協調動作の整合性について，システム試験に至るまでに網羅的に評価する手法として，形式手法[10]のひとつであるモデル検査を適用した手法が提案されている．モデル検査とは，システムの有限の状態空間を網羅的に調べ，与えられた性質が成り立つか否かを検証する技術である[11]．FPGA およびソフトウェアの協調動作に対しモデル検査を適用することで，モデル化した範囲および与えられた性質においては，網羅的な検証が可能となる．ただし，提案されている手法は，アーキテクチャ設計の段階ではなく，FPGA 開発過程やソフトウェア開発過程での適用を想定した手法である．

まず，FPGA 開発過程やソフトウェア開発過程で生成される成果物を基に，モデルを自動生成し協調動作を評価する手法が提案されている．[12]では，FPGA などのハードウェアの HDL コードおよびソフトウェアの C コードを対象として，その間の通信を抽象化しデータパスを持つ FSM (Finite State Machine)を自動で生成しプロパティ検証する方法が提案されている．[13]では，HDL コードのひとつである verilog-HDL とソフトウェアの設計記述である Executable UML (Unified Modeling Language)を対象として，それらをモデル検査ツールの入力形式に自動変換しプロパティ検証する手法が提案されている．

また，FPGA およびソフトウェアにおける正確な処理時間などを算出し，それを基に時間的な制約を満たすかどうかを検証する手法が提案されている．[14]では，Linear Hybrid Automaton (LHA)を適用して，ハードウェアとソフトウェアのそれぞれ異なった動作 clock を基に，協調動作に関する時間的な制約を検証する手法を提案している．[15]では，ハードウェアやソフトウェアなどの構成要素が担う処理の処理時間および処理するデータ量に着目し，それぞれの処理を組み合わせて実現するタスクが時間的な制約(デッドライン)を満たすかどうかについて，LHA を適用して検証する手法が提案されている．

FPGA とソフトウェアに関する協調動作の不整合について，その多くはシステム試験まで発見されず，システム試験においても見逃されてしまう可能性があること，電子機器システム開発の早期において FPGA とソフトウェアにおける協調動作の不整合を検出する手法は提案されていないことがわかった．電子機器システム開発におけるアーキテクチャ設計の段階において，FPGA とソフトウェアの協調動作の整合性を網羅的に検証する手法が求められている．よって，FPGA とソフトウェアの協調動作について，アーキテクチャ設計のアウトプットであり，各開発工程の入力文書となる FPGA 要求仕様およびソフトウェア要求仕様を基に，協調動作の整合性を評価する手法を提案する．

3. 協調動作に関する評価手法

3.1. 提案手法

本提案手法は，電子機器システムにおけるアーキテクチャ設計の結果である FPGA 要求仕様書およびソフトウェア要求仕様書に定義される協調動作を対象に，その整合性を評価する．提案手法における FPGA とソフトウェアに関する協調動作の整合性とは，FPGA とソフトウェア間におけるトランザクションレベル[16]の振る舞いがお互いに合致していることを意味する．また，本手法においては，協調動作を網羅的に評価するために，形式手法のひとつであるモデル検査を適用する．FPGA とソフトウェアの協調動作については，リアルタイム性(イベントに対し一定時間内に処理が完了する性質)を考慮する必要がある．そのため，適用するモデル検査技術としては，時間オートマトンである必要がある．

図 2に，評価フローを示す．以下の番号と図 2の番号は対応している．

① FPGA とソフトウェアとのインタフェース仕様などを基に，FPGA 要求仕様およびソフトウェア要求仕様からそれぞれとインタフェースを有する機能仕様を抽出する．

② モデル検査技術を用いて，FPGA 要求仕様およびソフトウェア要求仕様から抽出した機能仕様をモデル化し，FPGA 仕様モデルおよびソフトウェア仕様モデルを作成する．機能仕様をモデル化する際，条件やエラー処理の不足など協調動作に関する機能仕様が曖昧な箇所を発見できることがある．その場合は，モデル中にそれらの仕様を追加し，それぞれのモデルを成熟させる．

③ FPGA 仕様モデルおよびソフトウェア仕様モデルをマージすることで協調モデルを作成する．それぞれのモデルをマージする際，FPGA とソフトウェアとの間のインタラクションに関する仕様が曖昧な箇所を発見できることがある．その場合は，モデル中にそれらの仕様を追加し，協調モデルを成熟させる．

④ システム仕様から，協調動作が満たすべき性質を抽出する．この際，システム仕様から FPGA 要求仕

様もしくはソフトウェア要求仕様に引き継がれていない仕様を発見できることがある. その場合は, モデル中にそれらの仕様を追加し, 協調モデルを成熟させる. また, システム仕様から協調動作が満たすべき性質を抽出できない場合は, FPGA 要求仕様およびソフトウェア要求仕様を基に性質を抽出する.

⑤ 協調モデルに対し, 協調動作が満たすべき性質を与え, モデル検査を実施する. モデル検査の結果, 協調動作が満たすべき性質を協調モデルが満足しない場合, 検査結果を分析する. 分析の結果, モデル化時のミスなどモデルの不備を発見した場合, 協調モデルを修正し, 再度, モデル検査を実施する. 協調動作が満たすべき性質を協調モデルが満足せず, また, モデルに不備を発見できない場合は, 協調動作の仕様に不整合が存在することを意味する. その場合, 協調動作に関する仕様を再検討して協調モデルに反映し, 再度, モデル検査を実施する.

⑥ 協調動作が満たすべき性質をモデルが満足した場合, 協調モデルから評価の過程で追加もしくは見直したモデル箇所を抽出する. そして, 抽出したモデル箇所をFPGA機能仕様およびソフトウェア機能仕様に変換し, FPGA 要求仕様書, ソフトウェア要求仕様書, 場合によってはFPGA-ソフトウェア間インタフェース仕様書にフィードバックする.

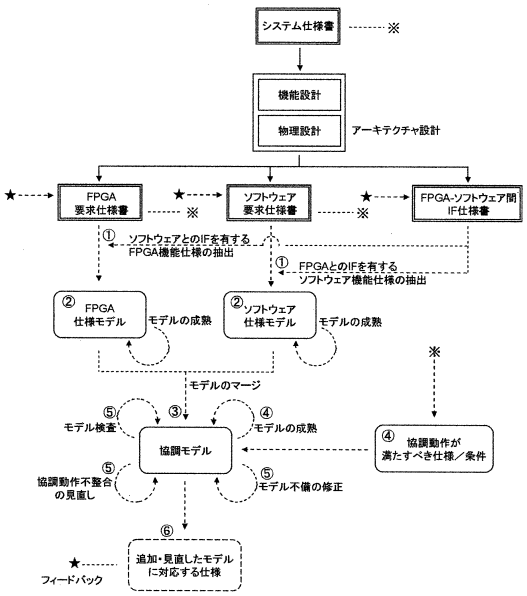

図 2 システム構成要素協調動作の整合性に関する評価手法

3.2. 提案手法の実装

本提案手法を実装するにあたり, 本稿ではモデル検査技術として産業界でも利用実績が多い UPPAAL を採用した[17]. UPPAAL はシステムの状態遷移を有向グラフでモデル化し, 評価対象が満たすべき性質を時相論理式で定義する. そして, 性質が満たされているかを網羅的に検証する. UPPAAL の特徴は, GUI を用いて直感的に状態遷移をモデル化でき, 並行動作するプロセスをモデル化することができる. また, クロックの定義により, システムのリアルタイム性を評価することもできる.

表 1に, FPGA とソフトウェアの協調動作について, 本実装にて評価する整合性を定義する.

表 1 協調動作整合性の定義

No.	協調動作整合性の定義
1	メッセージ送受信の順序が他構成要素と合致していること
2	メッセージ送受信に関連する処理の順序が他構成要素と合致していること
3	メッセージ送受信のタイミングが他構成要素と合致していること
4	メッセージ送受信に関連する処理の開始・終了のタイミングおよび待ち時間が他構成要素と合致していること
5	必要なメッセージ送受信および処理に抜けが無いこと

4. 適用事例

FPGA とソフトウェアにおける協調動作の整合性に関する評価手法の有効性を評価するために, SD メモリ・カードにアクセスする FPGA 要求仕様およびソフトウェア要求仕様を対象に本手法を試行した. 試行の結果, FPGA 要求仕様およびソフトウェア要求仕様から, FPGA とソフトウェアにおける協調動作の不整合を識別することができた. 以下, 適用事例の詳細を示す.

4.1. 試行対象

本提案手法の試行にあたり, [18]を対象とした. 試行対象とした理由について, システム仕様は存在しないものの, SD メモリ・カードとのインタフェースを担う FPGA の要求仕様, FPGA とインタフェースすることにより SD メモリ・カードにアクセスするソフトウェアの要求仕様および FPGA とソフトウェアのインタフェース仕様であるレジスタ・マップが公開されているためである. また, FPGA と SD カード・メモリの通信プロトコルである SPI モードについては, Malti Media Card と互換性があるため, 技術情報の入手性が高いと判断したためである.

4.2. 評価手法の試行

4.2.1. 協調動作に関する FPGA 要求仕様およびソフトウェア要求仕様の抽出

FPGAとソフトウェアとのインタフェース仕様であるレジスタ・マップを基に，FPGA 要求仕様およびソフトウェア要求仕様からそれぞれとインタフェースを有する機能仕様を抽出した．以下に，ソフトウェアとの協調動作に関連するFPGAの機能仕様を示す．

協調動作に関するFPGA機能仕様

- FPGA は，データ無しのコマンドに関するオペレーションもしくは Read オペレーションを行う．
- 電源投入後，FPGA は IDLE ステートにいる．オペレーション開始要求を受けると，CMND ステートに移行する．
- CMND ステートにて，Status レジスタ bit0 (Busy) を'1'，Status レジスタ bit1 (Resp) を'0'に設定する．そして，6 バイトのコマンドを送信後，RESP ステートで response を待つ．
- RESP ステートにて response を受信した場合，データ無しの場合は Status レジスタ bit0 (Busy) を'0'，Status レジスタ bit1 (Resp) を'1'に設定し，TRND ステートへ移行する．R1b レスポンスの場合は，Status レジスタ bit1 (Resp) を'1'に設定し，WBSY ステートに移行する．データ・リードの場合は，Status レジスタ bit1 (Resp) を'1'に設定し，WDRD ステートに移行する．
- WDRD ステートにて，"11111110" (Start Block) を待つ．Start Block を受信した場合，DTRD ステートに移行する．
- DTRD ステートにて，Start Block に引き続き 512 バイトの Data Block を受け取る．Data Block を受信後，Status レジスタ bit0 (Busy) を'0'に設定する．
- CRC1 ステートにて CRC1 を読み捨て，CRC2 ステートに移行する．
- CRC2 ステートにて CRC2 を読み捨て，TRND ステートへ移行する．
- WBSY ステートにて，SD メモリ・カードからの Busy Token を監視する． Busy Token の終了後，Status レジスタ bit1 (Resp) を'0'に設定し，TRND ステートに移行する．
- TRND ステートにてダミークロック期間を取り，IDLE ステートに移行する．

以下に，FPGA との協調動作に関連するソフトウェアの機能仕様を示す．

協調動作に関するソフトウェア機能仕様

- ソフトウェアは，データ無しのコマンドに関するオペレーションもしくは Read オペレーションを通し，SD メモリ・カードへアクセスする．
- Config レジスタを設定する．response のタイムアウトを許可するために，Config レジスタ bit5 (タイムアウト許可) に'1'をセットする．
- Command FIFO レジスタを設定する．送信する 48bit のコマンドを上位バイトから順に 1 バイトずつ書き込む．
- Control レジスタを設定する．Control レジスタ bit2～0 (Response 長-1) に期待する response の長さ-1，response が R1b を期待する場合は bit3 (R1b) に'1'，リード・データを期待する場合は bit4 (リード・データあり) に'1'をセットする．
- Control レジスタの bit6 に'0'をセットすることでオペレーションを開始する．ソフトウェアは，Status レジスタ bit0 (Busy) のポーリングにより，オペレーション終了を待つ．

以下に，インタフェース仕様であるレジスタ・マップを示す．

レジスタ・マップ (関連 bit のみ抜粋)

名称	内容	
Control (W)	bit6	'1'=オペレーション開始
	bit4	'1'=リード・データ有り， '0'=リード・データ無し
	bit3	R1b
	bit2..0	Response 長-1 (8bit 単位)
Status (R)	bit1	Resp (レスポンス受信済み)
	bit0	Busy (動作中)
Config (R/W)	bit5	タイムアウト許可 ('1'=許可)
Command FIFO (W)	SD メモリ・カードに送出するコマンドを書き込む．1 回のオペレーションに必要な 6 バイトすべて書き込む．FIFO の内容がそのまま出力される．	
Response FIFO (R)	response で返された内容が入る．	

4.2.2. FPGA 仕様モデルおよびソフトウェア仕様モデルの作成

UPPAAL を用いて，FPGA 要求仕様およびソフトウェア要求仕様から抽出した機能仕様をモデル化し，FPGA 仕様モデルおよびソフトウェア仕様モデルを作成した．図 3 に FPGA 仕様モデル，図 4にソフトウェア仕様モデルを示す．FPGA およびソフトウェアの機能仕様としてステート・マシンが定義されている場合は，それを参考にモデルを作成することもできる．

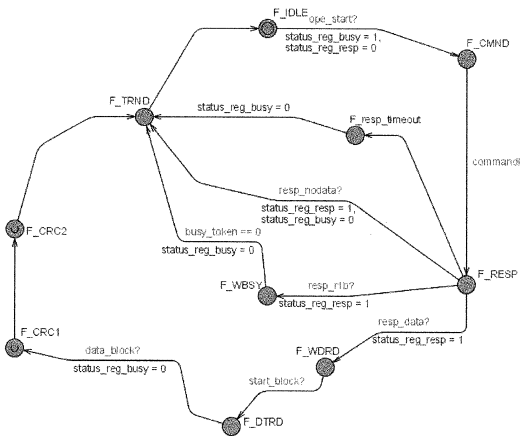

図 3 FPGA 仕様モデル

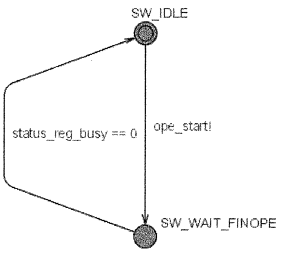

図 4 ソフトウェア仕様モデル

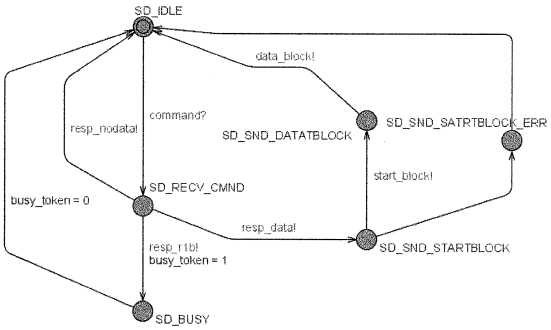

図 5 SD メモリ・カード仕様モデル

4.2.3. 協調モデルの作成

FPGA 仕様モデルおよびソフトウェア仕様モデルをマージし，協調モデルを作成した．FPGA としては SD メモリ・カードにアクセスするため，SD メモリ・カードとの協調動作についても考慮する必要がある．従って，図 5に示す SD メモリ・カードの仕様モデルを作成した．FPGA および SD カード間の同期的なインタフェース，ソフトウェアおよび FPGA 間の同期的なインタフェースについてはチャネルを用いた．また，レジスタおよび SD カードが送信する Busy Token については，bool 型変数を用いた．モデル検査技術として UPPAAL を採用した場合，モデルとして複数のプロセスを定義できる．そのため，モデルをマージする場合は，FPGA 仕様モデル，ソフトウェア仕様モデル，および SD メモリ仕様モデルの変数をすり合わせるだけでよい．

ここで，協調モデル作成の段階において，FPGA として，WBSY ステートにて Busy Token が終了しない場合の仕様が明確化されていないことが判明した．従って，SD メモリ・カードからの Busy Token が一定時間経っても終了しない場合はタイムアウトし，TRND ステートに遷移するパスをモデル中に追加した．この際，Busy Token が一定時間経っても終了しないという異常状態をソフトウェアが検知できない仕様になっていることも判明した．これについては，Status レジスタに本エラー内容に対応する bit を準備し，ソフトウェアがそのレジスタを参照することで対処できると考える．また，FPGA として，WDRD ステートにて Start Block を受信できない場合の仕様が明確化されていないことも判明した．従って，SD メモリ・カードからの Start Block が一定時間経っても受信できない場合はタイムアウトし，TRND ステートに遷移するパスをモデルに追加した．この際，Start Block が一定時間経っても受信できないという異常状態をソフトウェアが検知できない仕様になっていることも判明した．これについては，Status レジスタに本エラー内容に対応する bit を準備し，ソフトウェアがそのレジスタを参照することで対処できると考える．図 6に，協調モデルのうち仕様を追加した FPGA 仕様部分を示す．太字矢印および四角で囲った部分が追加した箇所である．

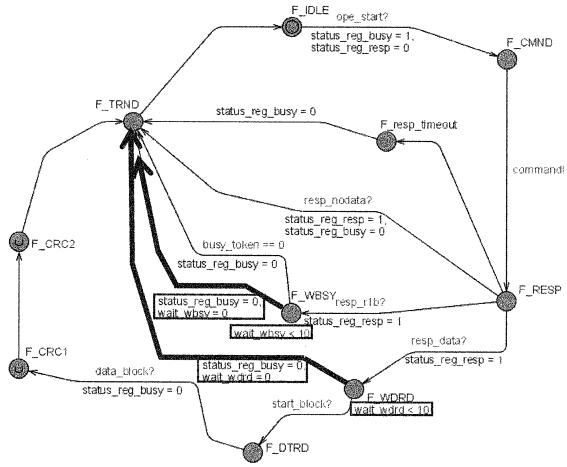

図 6 協調モデル（モデル化時仕様追加 FPGA 仕様部分）

4.2.4. 協調動作が満たす性質の抽出

本手法の試行対象については，FPGA 要求仕様およびソフトウェア要求仕様の上位仕様となるシステム仕様が存在しなかった．従って，試行においては，FPGA の仕様から以下の性質を抽出し時相論理式を作成した．

抽出した性質および時相論理

”すべてのパスにおいて必ずいつかは TRND ステートに遷移する”

A<>P_FPGA_SD_ACCESS.F_TRND

4.2.5. モデル検査の実施

協調モデルに対し抽出した性質を与え，モデル検査を実施した．その結果，与えた性質を満足しない反例が検出された．反例を分析したところ，FPGA として，response のタイムアウトが発生した後で response を受信した場合の仕様が考慮されていないことが判明した．従って，

response のタイムアウトが発生した後でも response が受信できる様にモデルを修正した. 図 7に, 協調モデルのうち仕様を見直した FPGA 仕様部分を示す. 太字矢印および四角で囲った部分が修正した箇所である.

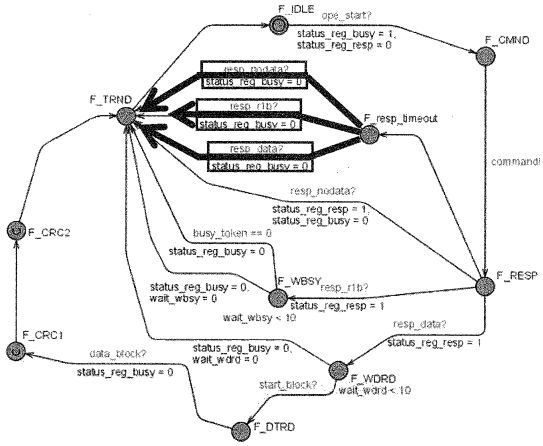

図 7 協調モデル(検査時仕様見直し FPGA 仕様部分)

4.2.6. 各要求仕様に対するフィードバック

図 7に示すモデルの修正を実施し, 再度, モデル検査を実施したところ, 反例が出力されなくなった. 評価の過程で追加もしくは見直したモデル箇所を協調モデルから抽出し, FPGA 機能仕様, ソフトウェア機能仕様およびインタフェース仕様に変換した. 以下に, 変換結果を示す. それぞれの変換結果については, 各 FPGA 要求仕様, ソフトウェア要求仕様, および FPGA-ソフトウェア間インタフェース仕様にフィードバックする.

FPGA 機能仕様

- Busy Token が一定時間経っても終了しない場合はタイムアウトし, TRND ステートに遷移する.
- SD メモリ・カードからの Start Block が一定時間経っても受信できない場合はタイムアウトし, TRND ステートに遷移する.
- response のタイムアウトが発生した後に response を受信した場合, response を一旦受信し, 破棄する.

ソフトウェア機能仕様

- オペレーション終了後, Status レジスタ bit2 (Busy Token エラー) および Status レジスタ bit3 (Start Block エラー) を確認する

FPGA-ソフトウェア間インタフェース仕様

- Status レジスタ bit2 : Busy Token エラー='1'
- Status レジスタ bit3 : Start Block エラー='1'

4.3. 試行結果

FPGA とソフトウェアにおける協調動作の整合性に関する評価手法を試行した結果, FPGA として WBSY ステートにて Busy Token が終了しない場合の仕様が明確化されていないこと, WDRD ステートにて Start Block を受信できない場合の仕様が明確化されていないこと, また, その際の Status レジスタの定義やソフトウェアの仕様に不足があることを識別した. これらは, 表 1 No.5 "必要なメッセージ送受信および処理に抜けが無いこと" という整合条件に該当しない不整合である. また, FPGA にて response のタイムアウトが発生した後で response を受信した場合の仕様が考慮されていないことも識別できた. これは, 表 1 No.3 "メッセージ送受信のタイミングが他構成要素と合致していること" という整合条件に該当しない不整合である. 従って, 本提案手法は, FPGA とソフトウェアにおける協調動作の整合性を評価する手法として有効であるといえる.

5. 考察

今回の試行により, 本提案手法は, FPGA とソフトウェアにおける協調動作の整合性に関する評価手法として有効であることが確認できた. また, 試行環境として CPU : Intel Core2 1.06Ghz およびメモリ : 0.99GB を搭載する一般的な PC を使用したが, すべてのモデル検査が 1s 以内で終了し, 状態爆発は発生しなかった. この理由として, モデル化の対象が FPGA 要求仕様およびソフトウェア要求仕様で定義されるトランザクションレベルの協調動作であること, モデルの変数としてチャネルや bool 変数を使用していること, FPGA のモデル化の範囲が SD カードアクセスという 1 機能モジュールに相当していることが考えられる. よって, 本提案手法におけるモデル化の範囲としては, FPGA の 1 機能モジュールおよびそれに対応するソフトウェアとの協調動作の整合性とするのが状態爆発を引き起こさない実用的な範囲である可能性がある.

今回の試行を通し, 本提案手法には課題があることもわかった. まず, モデル検査を実施する場合, システム仕様から協調動作が満たすべき性質を抽出するが, どのような性質を抽出するかについては, 明確な基準はなく, 評価者の経験やスキルに依存する. 評価者によるばらつきがない Systematic な方法を検討する必要がある. 次に, モデル化時に仕様の抜けを識別した場合, モデルに対し仕様を追加しているが, 仕様の抜けに気づくかどうかは評価者に大きく依存する. この課題についても評価者に依存しない Systematic を検討する必要がある.

6. まとめ

本稿では, 電子機器システム開発の初期フェーズであるアーキテクチャ設計段階において, FPGA やソフトウェアの要求仕様を基に協調動作の整合性を評価する手法を提案した. そして, SD メモリ・カードにアクセスする FPGA

要求仕様およびソフトウェア要求仕様を対象に本手法を試行した．提案手法の試行を通して，手法の有効性を確認することができた．しかし，試行を通して，協調動作が満たすべき性質の抽出は，評価者の経験やスキルに依存するなどの課題を識別することができた．今後，評価者によるばらつきがないSystematicな方法を検討する必要がある．

参考文献

[1] 社団法人電子情報技術産業協会半導体部会，“WSTS 2008 年秋季半導体市場予測について”，2008

[2] 経済産業省商務情報製作局情報政策ユニット情報処理振興課，”2008年版組込みソフトウェア産業実態調査－経営者および事業責任者向け調査－，2008”

[3] 宮崎仁，”FPGA の特徴と最新動向を理解する”，CQ出版社，Design Wave Magazine 2008 年 9 月号，pp.52-58，2008

[4] Nancy G. Leveson, “Medical Devices: The Therac-25 Story”, SAFEWARE: SYSTEM SAFETY AND COMPUTERS, pp.515-553, 1995

[5] 清水久二，”アリアン 5 の爆発事故とソフトウェア安全性に関する国際規格”，安全工学会，安全工学誌 Vol.41，No.1，pp.39-42，2002

[6] “IEEE Standard for Application and Management of the Systems Engineering Process”, IEEE 1220-2005, 2005

[7] 中川敬三，木下常雄，竹田修，辻政信，山本徹也，”宇宙用電子機器設計支援システムの基本設計結果”，宇宙航空研究開発機構研究開発資料，RM-04-023，2005

[8] “CoWare Platform Architect: SystemC Platform Capture and Architecture Analysis for Platform-driven ESL design”

[9] B. W. Boehm, “Verifying and Validating Software Requirement and Design Specifications”, IEEE Software Jan., 1984

[10] 中島震，“ソフトウェア工学の道具としての形式手法”，NII Technical Report, NII-2007-007J, 2007

[11] E.M.Clarke, O.Grumberg, D.E.Long,, “Model Checking”, Springer-Verlag Nato ASI Series F, Vol.152, 1996

[12] 西原祐，松本剛史，小松聡，藤田昌宏，”状態繊維表現への変換に基づくハードウェア/ソフトウェア協調設計の形式的検証手法”，電子情報通信学会論文誌 D，Vol.J89-D，No.4，pp.651-659，2006

[13] Fei Xie, Xiaoyu Song, Haera Chung, Ranajoy Nandi, “Translation-Based Co-Verification”, Proceedings of the 2nd ACM/IEEE International Conference on Formal Methods and Models for Co-Design, pp.101-120, 2005

[14] Pao-Ann Hsiung, “Hardware-Software Coverification of Concurrent Embeded Real-time Systems”, 11th Euromicro Conference on Real-Time Systems, pp.216-223, 1999

[15] Jih-Ming FU, Trong-Yen LEE, Pao-Ann Hsiung, Sao-Jie Chen, “Hardware-Software Timing Coverification of Distributed Embedded Systems”, IEICE Transactions on Information and Systems, Vol. E83-D, No. 9, pp. 1731-1740, 2000

[16] 藤田昌宏，”システムLSI設計工学”，オーム社，2006

[17] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, Wang Yi, “UPPAAL - a Tool Suite for Automatic Verification of Real-Time Systems”, In Proceeding of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems, 1995

[18] 河野崇，”ディジタル・オーディオ・プレーヤの製作”，CQ出版社，Design Wave Magazine, 2007年7月号，pp.95-105，2007