Vol.2011-SLDM-149 No.32
Vol.2011-EMB-20 No.32
2011/3/19

トランザクション識別子を伴うバスプロトコル間の変換器自動生成手法

酒井皓太[†1]　垣内洋介[†1]　浜口清治[†1]

ハードウェア開発において，設計資産の再利用をより柔軟に行うための手法に，オンチップ・バスプロトコル変換器の自動合成がある．本研究では，多くの現行バスプロトコル規格が持つトランザクション識別子付きの転送機能を利用して，識別子を伴ってデータをバッファリングする変換器の生成手法を提案する．また，変換器のバッファ容量をパラメータ化し，パラメータを変更したシミュレーションによってバッファ容量を評価する．

Automated converter generation for bus protocols with transaction identifiers

Kota Sakai,[†1] Yosuke Kakiuchi[†1]
and Kiyoharu Hamaguchi[†1]

Automatic synthesis of an on-chip bus protocol converter makes reuse of intellectual properties more flexible in the hardware development process. In this paper, we propose a method to generate a converter which buffers data with transaction identifiers with which many current protocols have a function to transfer. Moreover, we parameterize the buffer size of our converter to estimate its appropriate size using simulations with different parameters.

1. 背　景

システム LSI 設計の高度化により，既設計部品を IP コア (Intellectural Property Core) として再利用することによって設計コストの削減を図ることが望まれるようになった．

IP 再利用設計では，モジュール間のデータ通信は共有のバスを通して行われる．そのため，オンチップ・バスプロトコルと呼ばれるバスの通信規格に依存して，採用できる IP コアは限定される．

これに対して，異なるプロトコルに適合した IP コア同士を，プロトコル変換器を通して接続するというアプローチが考えられる．しかし，該当する 2 つのプロトコルの組み合わせに対応したプロトコル変換器が存在しないときは，新規に変換器を設計しなければならず，設計コスト削減という目的は果たされない可能性が高い．この問題を解決するための手法として，2 つのプロトコル記述の組み合わせから，プロトコル変換器を自動的に生成するという手法が研究されている．

バッファ共有型のプロトコル変換器の自動生成手法では，生成される変換器は内部に容量や入出力ビット幅がパラメータ化されたバッファを持つ．そのため，データバス幅の比に依存しないという特徴の他に，変換器が通信のボトルネックとならないように，バッファの容量を調整できるという利点も持つ．しかし，転送の誤り情報が正しく伝達できないという問題があった．その原因に，アウト・オブ・オーダー転送への非対応が挙げられる．

アウト・オブ・オーダー転送とは，転送要求とそれに対する応答の順序が一致しない転送方法のことである．これに対して，要求と応答の順序が一致する転送方法をイン・オーダー転送と呼ぶ．イン・オーダー転送では，要求と応答をその順序関係により対応付けることができるが，アウト・オブ・オーダー転送では，それぞれの要求と応答に，それらを対応付けるための識別子を付与する必要がある．これをトランザクション識別子と呼ぶ．既存手法では，トランザクション識別子をバッファリングしていないため，アウト・オブ・オーダー転送に対応できない．

本研究では，変換器内部のバッファに関して，生成されるバッファにトランザクション識別子を記憶する機能を追加することで，アウト・オブ・オーダー転送に対応した．それによって，正しく誤り情報が伝達できる変換器の自動生成を可能にした．また，自動生成される変換器のバッファ容量をシミュレーションによって見積もる方法を，実験を交えて議論する．

2. 関 連 研 究

プロトコル変換器の自動生成手法では，プロトコルの仕様を 2 つ受け取り，その 2 つのプロトコルの間で動作する変換器を生成する．Passerone ら[1)2)] は，積オートマトンの探索を行うことで，データの遅延を最小限に抑えた変換器の生成を可能にした．渡辺ら[3)]，石川ら[4)] は，この手法を拡張し，1 回のアドレシングで複数回の転送を行うバースト転送や，要求と応答を並行して行うパイプライン処理などの機能にも対応可能な生成手法を提案した．

積オートマトンを用いた手法とは異なり，変換対象プロトコルと共通プロトコルの部分変換器を入力として与え，2 つの部分変換器を組み合わせて変換器を得るという手法がある．部分変換器はライブラリとして蓄積されたものの中から再利用されるため，この手法はライブラリ方式と呼ばれる．ライブラリ方式の一手法として，プロトコル間のデータの受け渡しをバッファを通して行う，バッファ共有型の変換器生成手法がある．バッファ共有型の

†1 大阪大学
Osaka University

Vol.2011-SLDM-149 No.32
Vol.2011-EMB-20 No.32
2011/3/19

変換器では，双方のプロトコル間でのデータの受け渡しは共有バッファを介して行われ，ライブラリモジュールは直接データの処理を行わず，バッファのデータの出し入れのみを行う．従って，ライブラリモジュールを作り直すことなく双方のデータバス幅の変更に対応でき，データバスの幅が異なる場合でも，バッファの入出力の幅を変えることで，データを分割／結合して受け渡すことができる．この特徴を利用して，Yun ら[5] は，バス幅が異なる場合はバッファ共有型の手法を，そうでない場合は積オートマトン型の手法を，自動的に選択して変換器を生成する手法を提案した．彼らのバッファ共有型の変換器生成手法は，IPC(Interface Protocol Component) ベースの手法と呼ばれ，この手法では，変換対象プロトコルのバス幅に応じて，前述のような入出力の幅が異なるの FIFO キューとしてバッファを自動生成することで，バス幅及びクロック周波数の異なるバスの間で動作する変換器を生成している．

このように，受け渡すデータを自在に分割したり結合したりできるのが IPC ベース生成手法の最大のメリットであるが，アウト・オブ・オーダーのトランザクション実行機能を持たないプロトコルでは，転送エラーを正常に伝達できない．その例を図 1 に示す．この例では，マスターから変換器への 2 回目の転送データが誤りを含んでいる．変換器はそのデータを蓄え，次の転送開始のために転送成功の信号をマスターに返すが，その後，マスターからの 4 回分のデータをまとめてスレーブに渡したときにスレーブが誤りを検出し，直ちに転送失敗の信号を変換器に送るが，マスターは既に 3 回目までのデータの転送には成功していると認識しているため，変換器がマスターに 2 回目のデータのエラーを伝える方法がない．このように，イン・オーダー転送では，変換器はエラーを正しく取り扱うことができないため，アウト・オブ・オーダー機能を持つプロトコルを選択する必要がある．しかし，既存の IPC ベース手法では FIFO バッファを用いているため，アウト・オブ・オーダーに対応できない．よって，今までの自動生成の手法では，エラーを取り扱うことができなかった．

3. IPC ベース変換器自動生成手法の拡張

3.1 概　　要

IPC ベース手法は，生成されたバッファと IPC スレーブ・IPC マスターと呼ばれるライブラリモジュールを接続して変換器とする．IPC スレーブは特定のプロトコルのマスターモジュールと通信するスレーブモジュールとして動作し，そのプロトコルのデータバスの値を変換器のバッファに出し入れするライブラリモジュールである．IPC マスターも同様に変換器のバッファを操作し，特定のプロトコルのスレーブモジュールと通信するマスターモジュールとして動作する．

既存の IPC ベース手法では，バッファは入力データのビット幅と出力データのビット幅が異なる FIFO キューとして実装される．FIFO キューの入出力ビット幅は変換対象プロトコルのデータバス幅によって決まり，入力ビット幅が出力ビット幅より広ければ入力を分割して格納し，出力ビット幅が入力ビット幅より広ければ格納データを結合して出力する．

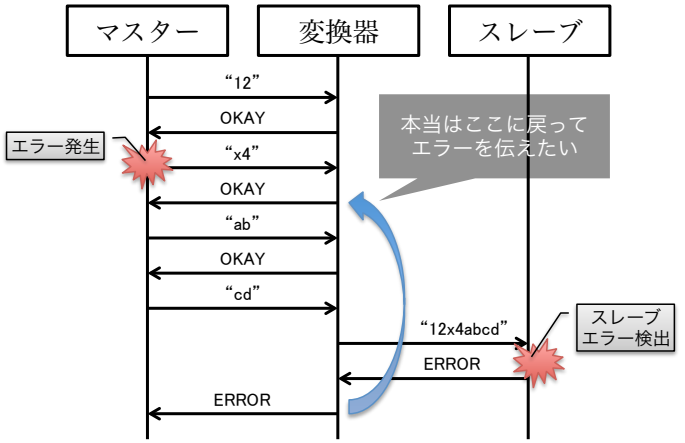

図 1　エラー伝達の失敗例

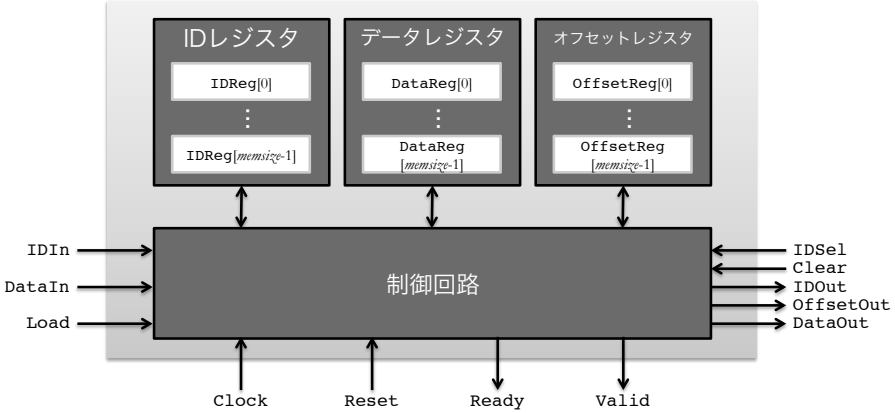

図 2　TDB の回路構造

本手法では，アウト・オブ・オーダー転送に対応した IPC ベース変換器の自動生成を可能にするために，トランザクション識別子によってタグ付けされたデータを扱うバッファを生成する．このバッファをタグ付きデータバッファ(Tagged Data Buffer: TDB) と呼ぶ．TDB の構造を図 2 に示す．入力側ビット幅が m ビット，出力側ビット幅が n ビットであ

Vol.2011-SLDM-149 No.32
Vol.2011-EMB-20 No.32
2011/3/19

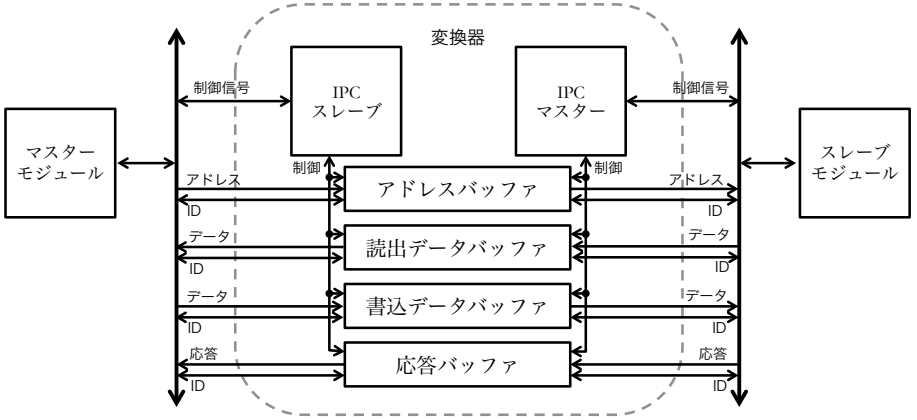

図 3 拡張 IPC ベース変換器の構造

るような TDB を m/nTDB と書くと，マスターモジュールと通信する側のデータバス幅が k ビット，スレーブモジュールと通信する側のデータバス幅が l ビットのとき，書き込みデータバッファは k/lTDB，読み出しデータバッファは l/kTDB として実装される．

バッファを TDB として実装することにより，拡張 IPC ベース変換器はトランザクション識別子を用いたアウト・オブ・オーダー転送に対応することができる．拡張 IPC ベース変換器の構造を図 3 に示す．図中の 4 つのバッファがすべて TDB として生成される．

3.2 バッファの機能拡張

TDB 生成アルゴリズムはパラメータ (*memsize*, *data-width*, *input-multiplier*, *output-multiplier*) を入力し，(*data-width*× *input-multiplier*/ *data-width*× *output-multiplier*) TDB を生成する．*memsize* は TDB の記憶容量を決めるパラメータである．*data-width* は TDB が扱うデータ 1 個分のビット幅を示し，TDB は *input-multiplier* 個のデータを同時に入力し，*output-multiplier* 個のデータを同時に出力する．

TDB は各 1 個の ID レジスタ，オフセットレジスタ，データレジスタの組を記憶の基本単位とする．これをセルと呼び，TDB の記憶領域は *memsize* 個のセルからなる構造となっている．それぞれのセルについて，データレジスタの値がビット幅 *data-width* のデータを表し，ID レジスタの値がそのデータの持つトランザクション識別子を表す．オフセットレジスタは同じ識別子を持つデータがバッファに格納された順序（早い順に $1, 2, \ldots$）を記憶するために用いられる．オフセット値 0 はそのセルが空であることを示す．

TDB の入力は `Clock`，`Reset`，`IDIn`，`DataIn`，`Load`，`IDSel`，`Clear` であり，それぞれクロック，リセット信号，ID 入力，データ入力，ロード信号，出力用 ID 選択，クリア信号を意味する．TDB の出力は `Ready`，`Valid`，`IDOut`，`OffsetOut`，`DataOut` であり，それぞれデータ入力可能を示す信号，データ出力有効を示す信号，ID 出力，オフセット出力，データ出力を意味する．`IDOut` は TDB に格納された識別子から 1 個選んで出力し，`OffsetOut` は `IDSel` で指定された識別子を持つセルのオフセットの最大値を出力する．入力から出力とレジスタ操作を決定するための TDB の動作仕様を以下に示す．

3.2.1 状態の遷移

データバッファはクロック立ち上がりのタイミングでレジスタの値を変化させる．$\texttt{IDReg}[i]$ の次状態を $\texttt{NextIDReg}[i]$，$\texttt{OffsetReg}[i]$ の次状態を $\texttt{NextOffsetReg}[i]$，$\texttt{DataReg}[i]$ の次状態を $\texttt{NextDataReg}[i]$ とすると，$\texttt{Clock} = 0$ から $\texttt{Clock} = 1$ になった瞬間に，$0 \le i \le \mathit{memsize}$ なる全ての i について，次の操作が行われる．

$$\begin{aligned}
\texttt{IDReg}[i] &\leftarrow (\texttt{Reset} = 1)\,?\,\texttt{NextIDReg}[i] : 0 \\
\texttt{OffsetReg}[i] &\leftarrow (\texttt{Reset} = 1)\,?\,\texttt{NextOffsetReg}[i] : 0 \\
\texttt{DataReg}[i] &\leftarrow (\texttt{Reset} = 1)\,?\,\texttt{NextDataReg}[i] : 0
\end{aligned}$$

3.2.2 最大オフセット値の検索と `OffsetOut` 出力の決定

`OffsetOut` と `CurrentOffset` の値は，次の式で決まる．ここで，`CurrentOffset` は `IDIn` の値を持つセルの最大のオフセット値（ただし，セルが存在しない場合は 0）を示す．このとき，データ入力の際に格納するオフセット値は，$(\texttt{CurrentOffset}+1), \ldots, (\texttt{CurrentOffset} + \mathit{input\text{-}multiplier})$ となる．

$$\begin{aligned}
\texttt{OffsetOut} &= \mathrm{MaxOffset}(\texttt{IDSel}) \\
\texttt{CurrentOffset} &= \mathrm{MaxOffset}(\texttt{IDIn}) \\
\text{ここで } \mathrm{MaxOffset}(n) &= \max(\{m | \exists i.\texttt{IDReg}[i] = n \wedge \texttt{OffsetReg}[i] = m\} \cup \{0\})
\end{aligned}$$

$\mathrm{MaxOffset}(n)$ は，ID レジスタの値が n に一致するようなセルの最大オフセット値を返す．

3.2.3 入力位置の決定

入力データを格納する *input-multiplier* 個のセルを決定する必要がある．決定したセルの添字を格納するための，要素数 *input-multiplier* の配列 `InputPointer` を新たに用意すると，$i = 1, \ldots, \mathit{memsize}$ で，`InputPointer` は次の式を満たす．

$$\begin{aligned}
&(\texttt{InputPointer}[i] \in S) \wedge (\forall j \le |S|.\texttt{InputPointer}[i] \neq \texttt{InputPointer}[j]) && (i \le |S|) \\
&\texttt{InputPointer}[i] = -1 && (i > |S|) \\
&\text{ここで } S = \{k | \texttt{OffsetReg}[k] = 0\}
\end{aligned}$$

すなわち，`InputPointer` の要素には，空のセルを示す添字 ($\in S$) が重複しないように選ばれる．$\texttt{InputPointer}[i] = -1$ は，空のセルが不足していて，添字の値が決定できなかったことを示す．つまり，`InputPointer` のいずれかの要素が -1 であれば，データ入力の操作は実行できず，このとき $\texttt{Ready} = 0$ となる．

Vol.2011-SLDM-149 No.32
Vol.2011-EMB-20 No.32
2011/3/19

3.2.4 出力位置の決定

DataOut の出力，及び内容を削除する *output-multiplier* 個のセルを決定する必要がある．決定したセルの添字を格納するための，要素数 *output-multiplier* の配列 OutputPointer を新たに用意すると，$a = 1, \ldots,$ *output-multiplier* で，OutputPointer は次の式で決まる．

$$\texttt{OutputPointer}[a] = \max(\{i | \texttt{IDReg}[i] = \texttt{IDSel}, \texttt{OffsetReg}[i] = a\} \cup \{-1\})$$

すなわち，OutputPointer の要素には，ID が IDSel に一致し，かつオフセット値が *output-multiplier* 以下であるようなセルの添字が選ばれる．$\texttt{OutputPointer}[a] = -1$ は，オフセット値が a に一致するセルが存在せず，添字の値が決定できなかったことを示す．つまり，OutputPointer のいずれかの要素が -1 であれば，正しい DataOut の値は出力されず，データ削除の操作も実行できないことを意味し，このとき $\texttt{Valid} = 0$ となる．

3.2.5 DataOut 出力の決定

3.2.4 項より，DataOut の出力値は次の式で決まる．ただし，$\texttt{wire}[m:n]$ は，信号線 wire の m ビット目から n ビット目までの切り出しを意味する．これは以降の項でも同様とする．

$$\texttt{DataOut}[\textit{data-width} \times a - 1 : \textit{data-width} \times (a-1)] = \begin{cases} \texttt{DataReg}[\texttt{OutputPointer}[a]] & (\texttt{OutputPointer}[a] \neq -1) \\ 0 & (\texttt{OutputPointer}[a] = -1) \end{cases}$$

つまり，DataOut には *output-multiplier* 個のデータを結合して出力する．

3.2.6 IDOut 出力の決定

IDOut の出力値は，バッファに格納された ID の値のうちのいずれかであり，入力とは無関係に決まる．つまり，次の式を満たしていればよい．

$$\texttt{IDOut} \in \{n | \exists i.\texttt{IDReg}[i] = n \wedge \texttt{OffsetReg}[i] > 0\}$$

3.2.7 ID レジスタの次状態の決定

ID レジスタに格納される値は IDIn で指定された値である．よって，$i = 1, \ldots,$ *memsize* で，NextIDReg は次の式で決まる．

$$\texttt{NextIDReg}[i] = \begin{cases} \texttt{IDIn} & (\exists a.\texttt{InputPointer}[a] = i \wedge \texttt{Load}) \\ \texttt{IDReg}[i] & (\text{otherwise}) \end{cases}$$

3.2.8 データレジスタの次状態の決定

DataIn の値は *input-multiplier* 個に分割してデータレジスタに格納される．よって，DataIn を幅 *data-width* に切り分けた配列 DataInArray を用意すると，$i = 1, \ldots,$ *memsize* で，NextDataReg は次の式で決まる．

$$\texttt{NextDataReg}[i] = \begin{cases} \texttt{DataInArray}[a] & (\exists a.\texttt{InputPointer}[a] = i \wedge \texttt{Load}) \\ \texttt{DataReg}[i] & (\text{otherwise}) \end{cases}$$

ここで $\texttt{DataInArray}[a] = \texttt{DataIn}[\textit{data-width} \times a - 1 : \textit{data-width} \times (a-1)]$

3.2.9 オフセットレジスタの次状態の決定

オフセットレジスタの値は様々な条件によって変化する．場合分けを以下に列挙する．

- *input-multiplier* 個のセルに新しくデータが格納されるとき，オフセットの値は (CurrentOffset+1), ..., (CurrentOffset+*input-multiplier*) となる (3.2.2 項を参照)．
  - ただし，$\texttt{IDIn} = \texttt{IDSel}$ かつ $\texttt{Clear} = 1$ のとき，格納される ID と同じ ID のデータが同時に *output-multiplier* 個削除されるため，オフセット値を *output-multiplier* 減らす．
- $\texttt{Clear} = 1$ のとき，ID が IDSel に一致するセルのオフセット値を減らす．
  - オフセット値が *output-multiplier* より大きければ，値を *output-multiplier* 減らす．
  - オフセット値が *output-multiplier* 以下であれば，値を 0 にする．よって，そのセルは取り出されることになり，空になる．

従って，$i = 1, \ldots,$ *memsize* で，NextOffsetReg の値は次のような式で決まる．

$$\texttt{NextOffsetReg}[i] = \begin{cases} \texttt{CurrentOffset} + a \\ \quad (\exists a.\texttt{InputPointer}[a] = i \wedge \texttt{Load} \wedge \neg(\texttt{Clear} \wedge \texttt{IDSel} = \texttt{IDIn})) \\ \texttt{CurrentOffset} + a - \textit{output-multiplier} \\ \quad (\exists a.\texttt{InputPointer}[a] = i \wedge \texttt{Load} \wedge \texttt{Clear} \wedge \texttt{IDSel} = \texttt{IDIn}) \\ \texttt{OffsetReg}[i] - \textit{output-multiplier} \\ \quad (\texttt{IDReg}[i] = IDSel \wedge \texttt{OffsetReg}[i] > \textit{output-multiplier} \wedge \texttt{Clear}) \\ 0 \ (\texttt{IDReg}[i] = IDSel \wedge \texttt{OffsetReg}[i] \leq \textit{output-multiplier} \wedge \texttt{Clear}) \\ \texttt{OffsetReg}[i] \ (\text{otherwise}) \end{cases}$$

3.3 変換器の生成

本手法で変換器生成に必要な入力は変換対象プロトコルに対応する IPC スレーブと IPC マスターと変換対象プロトコルのバス幅，及び生成される変換器のバッファ容量である．入力されたバッファ容量の値とプロトコルのバス幅に応じて TDB のパラメータ *memsize*, *data-width* が決まり，残りのパラメータはプロトコル間のデータバス幅の比によって決まる．(マスターモジュール側:スレーブモジュール側) のデータバス幅の比が $m:n$ のとき，アドレスバッファ・書き込みデータバッファとして生成される TDB は *input-multiplier* $= m$, *output-multiplier* $= n$，読み出しデータバッファ・応答バッファとして生成される TDB は *input-multiplier* $= n$，*output-multiplier* $= m$ となる．

IPC はこれらのバッファを操作するコンポーネントとして実装される．IPC スレーブはマスターモジュールと通信し，各バッファの出力を参照しながらアドレス／書き込みデータバッファへの値の格納と読み出しデータ／応答バッファからの値の取り出しを行う．IPC マスターはスレーブモジュールと通信し，IPC スレーブと同じようにアドレス／書き込みデータバッファへからの値の取り出しと読み出しデータ／応答バッファへの値の格納を行う．入

Vol.2011-SLDM-149 No.32
Vol.2011-EMB-20 No.32
2011/3/19

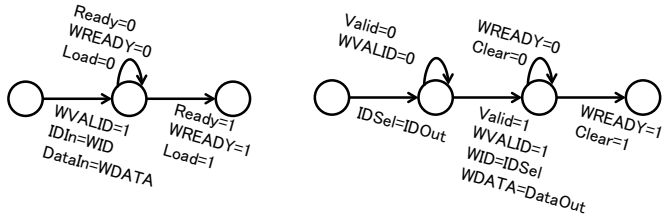

図 4 （左）バッファのデータ格納の流れ／（右）バッファのデータ取り出しの流れ

力された IPC と生成されたバッファを組み合わせて変換器とする．例として，AXI 規格に準拠したマスター／スレーブモジュールと IPC スレーブ／マスターが通信して，書き込みデータバッファに格納／取り出し操作を行うときの信号の遷移を図 4 に示す．図中の W で始まる信号が AXI バスの信号である．

4. 回路規模の評価

4.1 拡張 IPC ベース変換器の回路規模評価

2 つの変換器で回路規模の評価を行った．1 つは AMBA[6] AXI 規格のマスターと AMBA AHB 規格のスレーブの通信を取り持つ変換器であり，AXI 側のデータバス幅を 32 ビット，AHB 側のデータバス幅を 16 ビットに設定した．もう 1 つは AMBA AXI 規格のマスターと OCP[7] 規格のスレーブの通信を取り持つ変換器であり，AXI 側のデータバス幅を 16 ビット，OCP 側のデータバス幅を 32 ビットに設定した．これらの変換器について，バッファの記憶容量（TDB セル数）を変えて論理合成を行い，使用リソース量を評価した．実装先として想定した FPGA デバイスは，Altera Stratix IV である．図 5 にその結果を示す．図中の LUT は FPGA 中の論理エレメント数である．

図 5 より，変換器の回路面積は概ねバッファ容量に比例することがわかる．また，セル数 64 では，Stratix IV の利用可能 LUT 数 (58,080) の半数以上を消費している．これは，バッファ機能拡張によりレジスタを制御する組み合わせ回路が FIFO に比べて複雑になったためと考えられる．

4.2 考　　察

論理合成によって，変換器の回路規模はバッファの容量にほぼ比例することがわかった．提案手法による変換器はバッファ容量の大小にかかわらずエラーを正常に伝達可能であるが，バッファ容量が増えるとサイズが大きくなり，実装が難しくなるため，容量を適切に設定して運用する必要がある．

バッファ容量は変換器の回路面積とのトレードオフとなっており，最適なバッファ容量は接続 IP のトランザクション発生頻度や処理速度によって変化する．変換器にはプロセッサ

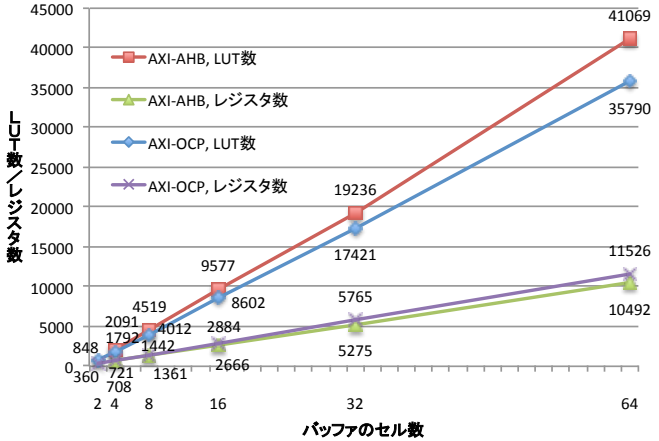

図 5 変換器の使用リソース量の評価

などの複雑な振る舞いをする IP コアが接続されることが想定されるため，画一的な数学的モデルを用いてバッファ容量を見積もることは難しい．そこで次章では，IP コアの動作モデルを用いたシミュレーションの結果をバッファ容量決定の目安とすることを考えた．

5. シミュレーションによるバッファ容量の見積もり

5.1 実 験 内 容

回路面積の評価に用いた AXI-AHB 変換器と AXI-OCP 変換器を RTL シミュレータ上で動かし，バッファ使用量の変動を計測した．シミュレーションを行うために，図 6 に示すような，変換器のテストベンチとして各プロトコルに従った動作を行う擬似的な IP コアを作成した．マスターとして働くモジュールは状態空間 $\{\mathrm{I}, \mathrm{B}, \mathrm{L}, \mathrm{H}\}$ を持ち，各状態間を確率的に遷移する．各状態での動作の内容は，トランザクション要求を発生させない (**I**dle)，連続的にトランザクション要求を発生させる (**B**urst)，低確率でトランザクション要求を発生させる (**L**ow)，高確率でトランザクション要求を発生させる (**H**igh) となっている．状態 B，L，H でのトランザクション発生間隔は指数分布に従うものとした．すなわち，トランザクションが発生したタイミングから数えて「x クロック後に次のトランザクションが発生している確率」は，確率分布関数 $F(x;\lambda) = 1 - e^{-\lambda x}$ で与えられる．パラメータ λ は状態 B のとき 1.0，状態 L のとき 0.1，状態 H のとき 0.5 に設定した．スレーブとして働くモジュールは状態空間 $\{\mathrm{O}, \mathrm{U}, \mathrm{E}\}$ を持ち，各状態間を確率的に遷移する．各状態での動作の内容は，トランザクション要求に正常応答する (**O**kay)，トランザクション要求に応答しない (B**u**sy)，トランザクション要求にエラー応答する (**E**rror) となる．

Vol.2011-SLDM-149 No.32
Vol.2011-EMB-20 No.32
2011/3/19

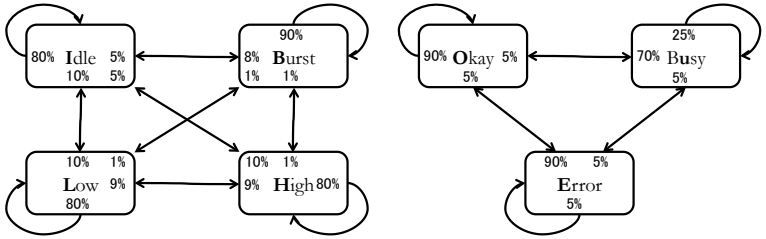

図 6 テストベンチの状態遷移図

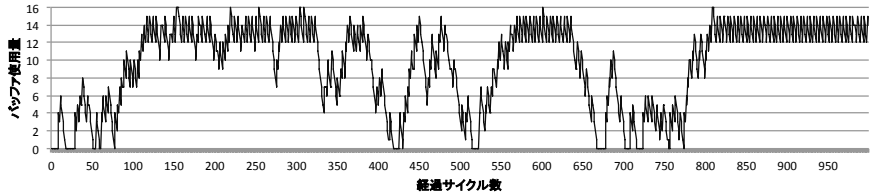

図 7 実験 1 の結果

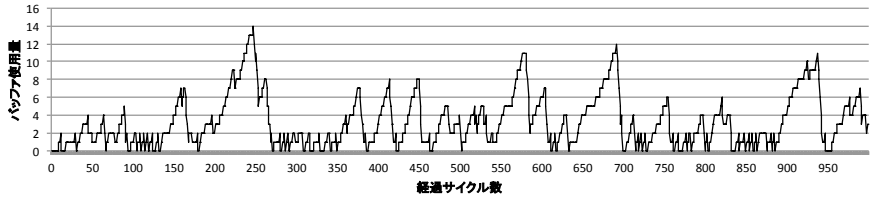

図 8 実験 2 の結果

各モジュールの状態空間の遷移確率行列をパラメータ化し，シミュレーション実験を行った．以降，状態 Q から Q' への遷移確率パラメータを $P_{QQ'}$ と書く．

5.2 実験結果

まず，AXI-AHB プロトコル変換器を用いて，図 6 の遷移確率でシミュレーションを行い，各サイクルでのアドレスバッファ使用量を測定した．結果を図 7 に示す． 図 7 の折れ線はバッファ使用量の遷移を示す．100~300 サイクル付近等の，折れ線が 12~16 の間を行き来している部分は，バッファ容量不足によりトランザクションが渋滞している状態である．

次に，AXI-OCP 変換器を用いてシミュレーションを行った．その結果を図 8 に示す． 4.1 節の設定の通り，この実験では，スレーブ側のデータバス幅がマスター側より広い．そのため，転送速度はスレーブ側のほうが速くなる．そこで，P_{UU} の値を大きく取る $(P_{\mathrm{UO}} = 9\%, P_{\mathrm{UU}} = 90\%, P_{\mathrm{UE}} = 1\%)$ ことで，スレーブモジュールが Busy 状態から抜け出しにくいような設定にした．その他の遷移確率は図 6 と同様である．図 8 は図 7 と同様，折れ線は各サイクルでのバッファ使用量の遷移を表す．

5.3 実験結果の考察

図 7，図 8 のように，シミュレーションによってバッファの使用状況を視覚化すると，変換器のバッファ容量見積もりのよい判断材料になることがわかる．例えば，実験 1 の設定通りに変換器が運用された場合，バッファの容量が不足して転送遅延が大きくなることが予想される．低遅延のシステムが求められるならば，バッファ容量をより大きく取ることが望ましい．また，実験 2 の結果からは，トランザクションの渋滞は観測されなかった．よって，回路面積の節約を図りたい場合，バッファ容量を小さくすることも考えられる．トランザクションの発生頻度や処理速度は IP の動作に左右されるので，この見積もりが完全に正確とは言えないが，設計段階である程度のバスシミュレーションを行えるのは有用と考える．

6. おわりに

IPC ベース手法を始めとする，内部に容量可変のバッファを持つ変換器を自動生成する手法は，任意の異なるデータバス幅を持つプロトコル間の変換器が生成可能であるという特徴を持つ．しかし，これらの手法では，アウト・オブ・オーダー転送に対応しない限り，エラーの転送が正しく行えないという問題があった．本研究では，IPC ベースの変換器生成手法を拡張し，生成されるバッファにトランザクション識別子を管理する機能を追加することにより，アウト・オブ・オーダー転送に対応した変換器を生成する手法を提案した．提案手法を用いた変換器はバッファ容量を変更しても正常にエラー伝達が可能である．

また，バッファ容量は伝送遅延とのトレードオフになる．最適なバッファ容量を数学的モデル等で定量的に見積もるのは，実際に変換器が運用される状況の多様性を考えると困難である．そこで，本研究では，提案手法により生成した変換器を用いて，バッファ容量をシミュレーションによって評価した．そして，シミュレーションによるバッファ使用状況の視覚化がバッファ容量決定の目安となることを実験によって示した．

参 考 文 献

1) Roberto Passerone, James A. Rowson, Alberto Sangiovanni-Vincentelli. Automatic Synthesis of Interfaces between incompatible Protocols. *DAC'98, San Francisco, California*
2) Roberto Passerone, Luca de Alfaro, Thomas A. Henzinger, Alberto L. Sangiovanni-Vincentelli. Convertibility Verication and Converter Synthesis: Two Faces of the Same Coin. *Intl. Conf. on Computer Aided Design (ICCAD),* 2002
3) Shota Watanabe, Kenshu Seto, Yuji Ishikawa, Satoshi Komatsu, Masahiro Fujita. Protocol Transducer Synthesis using Divide and Conquer Approach. *ASPDAC'07*
4) 石川悠司, 小松聡, 藤田昌宏, "積グラフ探索を利用した実用的なプロトコル変換器の自動合成と検証," 電子情報通信学会技術研究報告, Vol.107, No.506, pp.1-6, 2008 年 3 月
5) ChangRyul Yun, DongSoo Kang, YoungHwan Bae, HanJin Cho, KyoungSon Jhang. Automatic Interface Synthesis based on the Classificaion of Interface Protocols of IPs. *ASPDAC'08*
6) AMBA Specification (Rev 2.0) `http://www.arm.com/products/solutions/AMBA_Spec.html`
7) OCP-IP `http://www.ocpip.org/`