並行同期EFSM群を用いたバックボーン向けネットワークモニタ回路の実装と評価

桐村 昌行† 高本 佳史† 安本 慶一†† 中田 明夫† 東野 輝夫†

†：大阪大学大学院基礎工学研究科情報数理系

††：滋賀大学経済学部情報管理学科

近年, インターネットや高速ネットワークの発展に伴い，DoS攻撃やフラッディングなどの悪質な行為が増加している． このため, 高速ネットワーク上の大量のトラフィックをリアルタイムで監視するネットワークモニタの必要性が高まってきている． 高速回線上のネットワークモニタを実装する方法として， 処理の高速化やモニタ項目の頻繁な修正などに対処できるよう， FPGAなどの再構成可能なハードウェア回路で実装することが考えられる．本研究では、並列処理やパイプライン処理による高速化を考慮して，複数モジュールの並行動作と協調及び時間制約を扱うことができる時間制約付き並行EFSM群としてネットワークモニタをモデル化し，その形式記述からFPGA回路を自動生成する手法を提案すると共に，異なるモニタリング項目毎にFPGA回路を合成し, その実装実験の結果について報告する．

Design and Implementation of Network Monitoring Circuits for High Speed Networks

Masayuki Kirimura†, Yoshifumi Takamoto†, Keiichi Yasumoto††, Akio Nakata† and Teruo Higashino†

†: Dept. Info. and Math. Sci., Osaka Univ., Toyonaka Osaka., Japan

††: Dept. Info. Proc. and Man., Shiga Univ., Hikone Shiga., Japan

Recently, with the progress of the Internet and high-speed networks, vicious acts for attacking networks such as DoS attack and flooding are increased. For detection of such attacks in high-speed networks, we need real-time network monitors which can handle millions of IP packets per second. Since the monitoring items and the number of monitoring packets often change, it is needed that we can modify the developed network monitors easily and/or add facilities for detecting new types of attacks. One way to develop such high-speed network monitors is that we use reconfigurable circuits such as FPGAs and adopt speed-up techniques such as parallel processing and pipe-line processing. In this paper, we define a concurrent model called concurrent synchronous EFSMs to specify such network monitors and propose a technique to synthesize a FPGA circuit from a specification of a network monitor. We also show some experimental results obtained by using the developed synthesis tool where we have synthesized different types of FPGA circuits depending on the monitoring items and packets.

1 まえがき

近年のインターネットの普及は目覚しいものがある．それに伴い, インターネット上で交わされる情報は日々重要性を増している．現在のインターネット関連市場の拡大からも, この傾向は今後ますます強まっていくことが予想される [1, 6, 10]. そのため, ハッカーなどによる悪意のある行為からホストを守る, ネットワークセキュリティの重要性が増している．そういった行為を検出するソフトウェアやハードウェアもいくつかあり [7, 8, 9], 文献 [7] では, FDDIネットワーク上においてリアルタイムで検出可能なネットワークモニタが提案されている．また, 文献 [8, 9] では分散型のDoS攻撃の検出について提案されている．その他にもさまざまな検出ツールが市販されている．中でもリアルタイムネットワークモニタは，（1）攻撃をリアルタイムに検出できる,（2）侵入者に侵入検知システムの存在を隠蔽できる,（3）現在のモニタリングに影響を与えることなく, 比較的簡単に新しい型の攻撃を検出する機能を追加できる，などの理由から近年注目されている．

従来このようなネットワークモニタは，モニタ項目の頻繁な変更などの理由のため，ソフトウェアとして実装される場合が多かった．しかし, 高速ネットワークの発展に伴い, ギガビットイーサネットのような広帯域ネットワークがますます一般的になってきており, こういった高速ネットワーク上で悪質

な行為をリアルタイムで検出することを考慮した場合, ネットワークモニタ自身の処理の高速化が必要となる [5]. いくつかのネットワークモニタはハードウェアとして実装されているが, 高速ネットワーク上でのモニタリングの一つの適用例にすぎない [2, 3]. 一方, モニタリング項目の追加や変更を容易に出来る, 柔軟性のあるネットワークモニタが必要とされている. こういった柔軟性を実現する一つの方法として, ネットワークモニタをFPGA などの再構成可能なハードウェア回路で実装することが挙げられる. また, 並列処理やパイプライン処理を行なうことによってネットワークモニタ処理の高速化が考えられる.

我々の研究グループでは, 従来から複数のモジュールの並行動作と協調を扱うことができる並行同期EFSM群を用いてシステムの仕様をモデル化し, その形式記述からハードウェア回路を自動生成する手法を提案してきた [12, 13, 15]. 本稿ではその手法を利用し, 新たに, ネットワークモニタの処理をモデル化した形式記述からその仕様に相当するネットワークモニタの回路を自動生成する手法を提案する. その際, 処理の高速化のため, 並列処理やパイプライン処理を考慮すると共に, 実装の変更を容易にするため, ハードウェア回路として再構成可能なFPGA回路を自動合成する.

本手法を用いることのメリットとして, ネットワークモニタのデザイナーはパイプライン処理のようなハードウェア技術に関する知識を必要とすることはなく, ネットワークモニタの動作をEFSMを用いて記述し, それらのEFSM間における同期処理について記述するだけで簡単に設計できる点があげられる. デザイナーはモニタリングしたい項目をEFSMで記述し, あらかじめ我々がライブラリ化しているネットワークのモニタリングに必要な基本機能をいくつか抽出し, それらを組み合わせるだけで環境に応じたネットワークモニタを設計することができる.

本稿では, モニタリングの基本機能の中で最も基本的なものと考えられる, (1) IP フラッディング [1, 6] しているパケットのIPアドレスを検出するモジュールと, (2) SYN フラッド [1, 6] を検出するモジュール, について述べる. ここでIP フラッディングとは, あるホスト (サーバ) に対して短時間に大量のパケットが送りつけられることを指し, 他の利用者がそのホストからのネットワークサービスを受けることの妨げとなる. またSYN フラッドとは, 2端末間のTCP ベース通信を利用した攻撃方法で, ある端末の通信ポートを塞ぎ通信を妨害する. これら二つのモジュールはライブラリとして登録しており, デザイナーはこれらをプリミティブとしてネットワークモニタに組み込むことができる. またIP フラッディングを検出するためには, ネットワーク攻撃の種類やモニタリングするネットワークの帯域によって適切な閾値を決定したり, 1秒間あたりネットワークモニタを通過するパケットの数がその閾値を越えるかどうかを調べる必要がある. このようなDoS攻撃が実際に起こっているかどうかを判断する統計データについてはさまざまな論文で報告されている [4, 5, 10]. 我々はこれらのデータを用いて, IP フラッディングにおける適切な閾値

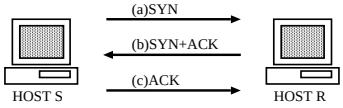

図 1: 通常の2端末間通信

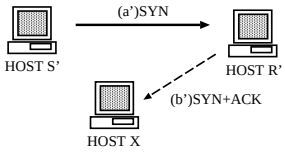

図 2: SYN フラッド

を決定しており, ライブラリに対してこれらのデータを引数として与える. この閾値を用いて提案するネットワークモニタが毎秒100万パケット以上のIP パケットを処理し, IP フラッディングをリアルタイムに検出するために, 自動合成によって得られたFPGA回路における必要な並行プロセス数を決定している. これらの理由によりFPGA回路を自動合成することを考えている.

評価のために, まずネットワークモニタの仕様を並行同期EFSM群でモデル化し, 文献 [12, 13, 15] の手法を用いてハードウェア記述言語VHDLの記述に自動変換した. その記述をSYNOPSYS社の論理合成ツール [16] を用いてRT レベルのFPGA レベル回路に論理合成した. 実装実験の結果, 適切な面積でかつ, 毎秒100万パケット以上処理できる回路を合成できることを確認した.

2 ネットワークのセキュリティの侵害と対象とする検出項目

2.1 サービス妨害

インターネットで用いられる技術の多くは幅広いサービスの要求に応えるように設計されている. しかし, 本来システムが想定していなかった使い方をされた場合に, サービス不能や能力低下の状態に陥る場合がある. このような状態を意図的に引き起こす行為を**サービス妨害攻撃（Denial of Service attack）**と呼ぶ. このような攻撃の代表的な例として, 悪意のある者がセキュリティ対策が不完全なサーバに大量のパケットを送り, そのサーバに負荷を掛けパフォーマンスを低下させることが考えられる. このデータの洪水のことを**IP フラッディング**という. これらのパケットは複数のサーバを経由して送られるため, 犯人を特定することが難しい.

2.2 SYN フラッド

通常2端末間における TCP ベースの通信は図1のように, まず, (a) 送信者ホストSがSYN パケットを通信相手ホストRに送信し, 通信依頼をすることから始まる. 次に (b) ホストRがSYN+ACK パケットをホストSに送信し, 通信の許可をする. 最後に (c) ホストSがACK パケットをホストRに送信することによって受理し, 通信が開始される. ただし, 次のような例がある. 図2のように, まず

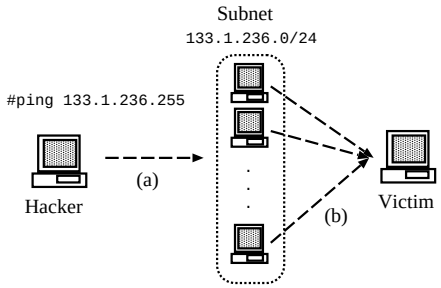

図 3: Smurf

(a') 悪意のある送信ホスト S' がホスト R' に SYN パケットを送る. その際, 架空のホスト X の IP アドレス名義で送信をする．そのため, (b') ホスト R' は SYN+ACK パケットをホスト X に送信し, ACK パケットが返ってくるのを待つことになる．この間ホスト R' は一定期間, ACK パケットがいつ届いても通信が開始できるように通信ポートを確保している．この間にホスト S' が同様の SYN パケットをホスト R' に送り続けると, ホスト R' の通信ポートはすべて埋められてしまい, 他のホストとの通信ができなくなる．このような攻撃を**SYN フラッド**という．そのため, SYN フラッドを検出するためには, SYN+ACK パケットと ACK パケットの対応を監視する必要がある．2 端末間の通信において第三者が介在していないかどうかを調べる必要がある．

2.3 Smurf

複数のホストを使ったフラッディングとして**Smurf** と呼ばれる攻撃があげられる．この攻撃は攻撃者が攻撃対象に対して ICMP(Internet Control Message Protocol) echo (ping) をブロードキャストすることによって行なわれる．具体的には，図 3のように，攻撃者 (Attacker) はまず,(a) 攻撃したい相手 (victim) になりすまし，あるサブネット (図 3中では 133.1.236.0/24 に属するホスト群) に対して ICMP echo (ping) を行う．これにより特定のホストではなく，サブネットに対して IP レベルのブロードキャストを実行する．次に, (b) エコーリクエストを受け取ったサブネット上のすべてのホストが一斉にその応答を返す．送信元 IP アドレスは被害者 (victim) になりすまされているため，リクエスト応答はすべて被害者 (victim) に送られる．これを連続的に実行すると，結果的には被害者 (victim) に対して大量の ping 応答パケットが送られ, 被害者 (victim) ホストのパフォーマンスが低下する．これが Smurf である．この Smurf を検出するためには, ICMP エコーリクエスト, ICMP ブロードキャストのトラフィック量を監視する必要がある．

2.4 ネットワークモニタのパケット監視

ネットワークモニタは，あるネットワーク上に流れる多量のパケットを監視し，ネットワーク中にどのようなパケットがどの程度流れているかを監視するシステムのことである．ネットワークモニタは図 4のように LAN などの回線と並列して配置されることが多く，パケット情報を取得する**ヘッダ情報取得部**と得られた情報を解析する**ネットワーク観測部**で構成されている．提案するネットワークモニタではバックボーンでのパケット監視を目的としており, 大量に流れるパケットから攻撃を受けている (している) ホストの IP アドレスを素早く割り出すために, 高速なハードウェアでの実装を行なっている．なお, ヘッダ情報取得部については Gb/s 以上のネットワークに対応している既存のものを利用することを前提とし, 取得したヘッダ部には, そのパケットの宛先や送信元 IP アドレス, パケットの種類などの記された bit 列が得られるものとする. [2, 3]. 図 5は，図 4中のネットワーク観測部の内部の基本構成である．ネットワーク観測部は図 5のように複数のモジュールを並列に設置し, 並行に処理を進める．各モジュールはモニタに必要な基本機能（**プリミティブ**）を表している．

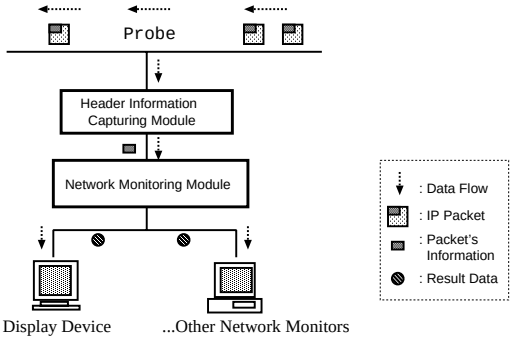

図 4: ネットワークモニタの概観

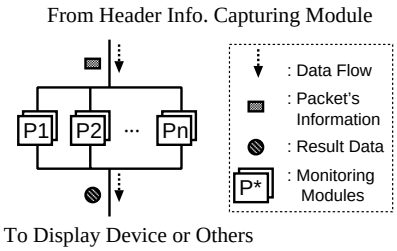

図 5: Network Monitoring Module

3 並行同期 EFSM 群による回路の記述と実装の概要

本稿では，システムを並行に動作する複数の拡張有限状態機械 (EFSM) 群とその間の同期指定でモデル化し (**並行同期** *EFSM* **群**と呼ぶ)，対応する回路記述言語 VHDL へと変換することで，回路を生成する [12, 15].

3.1 並行同期 EFSM 群

並行同期 EFSM 群では，各 EFSM は有限個のレジスタ (変数) を持ち，各遷移 (イベントと呼ぶ) では，ゲートと呼ばれるサービスアクセスポイント (SAP) を介して，入力値をレジスタに取り込んだり，外部に出力することができる．また，**ガード**と呼ばれる論理式が各イベントの実行条件として設定できる．イベントの実行によるデータの入力・

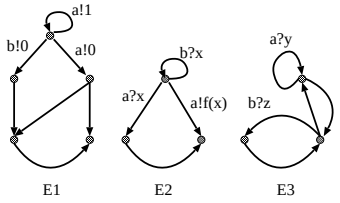

図 6: 並行EFSM群の例

出力は，それぞれ"$a?x : t[f(x)]$"，$b!Eq$のように表す．ここで，"a"や"b"は入出力用のゲートの名前であり，"$?x : t$"は入力値の変数x(型はt)への代入，$[f(x)]$はガード，"$!Eq$"は式Eqの計算結果の出力を表す式である．なお，複数データの同時入出力("$a?x!10$"など)も記述できる．

並行同期EFSM群では，与えられたEFSM群の任意のサブセットが，あるゲートのイベントを同時に実行することによって，そのゲートを介してデータ交換を行わせることが可能である(**マルチランデブ**[14]と呼ばれる)．

EFSM間でどのイベントを同期実行するかは，LOTOS[14]の並列オペレータを用いて以下のように指定する．

$$S ::= S \ |[gate\_list]| \ S \quad | \quad S \ ||| \ S \quad | \quad (S) \quad | \quad E$$

ここで，EはEFSMの名前であり，$gate_list$にはオペレータの両側で同期させたいイベントのゲートの並びを指定する．また，$|||$は，オペレータの両側でイベントを同期させる必要が無いことを指定する($gate_list = \emptyset$に相当)．

また，一対のEFSM間での同期だけでなく，$E_1|[a]|(E_2|[a]|E_3)$のように多数のEFSM間での同期の指定も可能である．これは，マルチキャスト通信のような1対多や多対多通信, 同一リソースへ複数EFSMが同時にアクセスする際の排他制御などを記述する際に有用である．

図6の$E_1|[a,b]|(E_2|[a]|E_3)$では，aのゲート名を持つイベントにおいて, E_1,E_2,E_3全てのEFSM間で同時に実行されなければならない．例えば，E_1, E_2, E_3の各イベント$a!0$, $a?x$, $a?y$が同期実行される場合, イベント$a!0$による出力値0がイベント$a?x$の変数xと, イベント$a?y$の変数yに受け渡される．(ただし, 出力値0と変数x, yの型が一致する必要がある)．また, E_1, E_2, E_3の各イベント$a!1$, $a!f(x)$, $a?y$が同期実行される場合, 関数$f(x)$の値は出力値1と等しくなければならない．

イベントにおいて, E_1がE_2とE_3のどちらか一方と同時に実行されなければならない．例えば, 図6の$E_1|[a,b]|(E_2|[a]|E_3)$では，bのゲート名を持つイベントにおいて, E_1のイベント$b!0$が実行される場合，その出力値0がE_2のイベント$b?x$の変数x, あるいは, E_3のイベント$b?z$の変数zに受け渡される.(ただし, 出力値0と変数x, zの型が一致する必要がある)．

3.2 マルチランデブ制御のための情報

	E_1	E_2	E_3
p_1	$(a!0,$	$a?x$	$a?y)$
p_2	$(a!0,$	$a!f(x)$	$a?y)$
p_3	$(a!1,$	$a?x$	$a?y)$
p_4	$(a!1,$	$a!f(x)$	$a?y)$
p_5	$(b!0$	$b?x)$	
p_6	$(b!0$		$b?z)$

表 1: Possible instances

	E_1	E_2	E_3
r_1	$(a!0,$	$a?x$,$a!f(x),$	$a?y)$
r_2	$(a!1,$	$a?x$,$a!f(x),$	$a?y)$
r_3	$(b!0,$	$b?x)$	
r_4	$(b!0,$		$b?z)$

表 2: マルチランデブ表

並行同期EFSM群では, イベントの同期を$|[a,b]|$オペレータなどを用いて指定するため, その実行時には同期イベントの組合せとその同期条件を動的に判定し決定しなければならない．ハードウェア実装においてこのような動的な計算はパフォーマンスの低下を招く．そこで, ハードウェア実装を容易にするために, (1) 同期するEFSMの名前の組，(2) それらの間で実行されるイベントの組，(3) それらのイベントの実行のため成立しなければならないガード式，に関する情報をあらかじめ静的に全て求める．例えば, 図6でのマルチランデブを考えてみる．まず, 全ての同期イベントの同期実行可能な組合せをあらかじめ求める．実際に求めたものを表1に示す．E_1, E_2, E_3全てのEFSMで同期指定されているイベントaの同期実行可能な組合せはp_1からp_4, E_1とE_2かE_3のどちらか一方と同期指定されているイベントbの同期実行可能な組合せはp_5とp_6である．

これらの同期実行可能な組合せの数は一般的に多くなってしまうため, 複数の組合せを1つにした**マルチランデブ情報**と呼ばれる組にまとめる．各マルチランデブ情報は(1) 各EFSM間で同期実行されるイベントの組と(2) その同期実行条件の2つからなる．ここでは, それぞれの組合せが同期条件とガード式を満たす組合せのみを抽出している．これら全てのマルチランデブ情報をまとめたものを**マルチランデブ表**と呼び, 図 6におけるマルチランデブ表は表2のようになる．

与えられた並行同期EFSM群から，マルチランデブ表を自動的に抽出する方法は，文献[13]に詳細が記述されているため，ここでは省略する．

3.3 回路化の概要

我々が文献[12, 13, 15]で提案している回路合成法では，最終的な回路を以下の部分から構成する．

(1) 同クロックで動作する各EFSMを実現する順序回路
(2) マルチランデブ制御部

図6の仕様に対して生成される回路の構成を図7に示す．

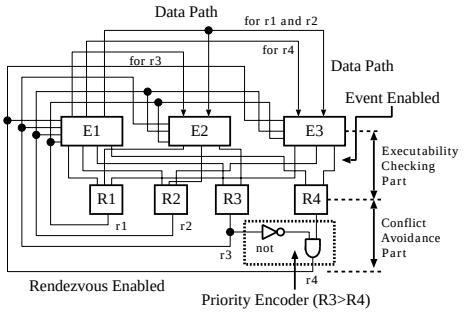

図 7: ハードウェアの全体構成

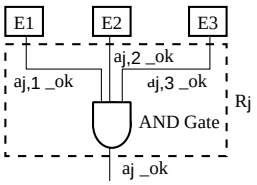

図 8: 同期判定回路の構成

各EFSMを実現する順序回路は，現在の状態を保持する状態レジスタ，クロック値を保持するカウンタレジスタを持ち，各クロック周期ごとに，現状態から実行可能なイベントのうち一つを実行し，指定された次の状態に移る (実行可能なイベントが存在しない場合には現状態で待機する)．EFSM中で用いられている各変数は，レジスタとして実現され，変数へのデータ入力を行うイベントの実行によって，各レジスタの値は随時更新される．

マルチランデブ制御部は，

（1） 各クロック周期ごとに各同期イベント組が実行可能であるかどうかの判定を行う同期判定部

（2） 排他的にしか実行できない複数の同期イベント組が同時に実行可能となった場合に実行させる同期イベント組の選択を行う競合回避部

から構成している [13, 15]．

(1) の同期判定部では，ランデブ情報ごとに同期判定回路を設ける (図7中の $R_1 - R_4$)．同期判定回路は，ランデブ情報に属する同期イベントを実行するすべてのEFSMから，その同期イベントが現状態から実行可能であるかどうかを示す信号 (Event Enabled) を受け取る．すべてのEFSMからの出力が真ならば，その同期イベント組は実行可能となる．その場合は，同期判定回路から各EFSMに対して同期が可能であることを示す信号 (Rendezvous Enabled) を送る．この判定回路は，n 入力 (n は同期を行うEFSMの個数) のAND回路で実現できる．

例えば，図6では表2より，イベント a に関する同期判定回路は図8 のようになる． E_1 のイベント $a!1$ が実行可能になると E_1 の順序回路から各 R_j $(j = 1..4)$ に対してランデブ可能信号 $a_{j,1}$_ok の値が1になり，各ゲート R_j のANDゲートに送られる． 同様に E_2，E_3 においてイベント $a?x$，$a?y$ が実行されると，各ランデブ可能信号 $a_{j,2}$_ok，$a_{j,3}$_ok が1になり，各ゲート R_j のANDゲートに送られる (図7の “Event Enabled” 信号)． これにより，R_j のANDゲートの入力値がすべて1となるため，その出力信号 a_j_ok の値が1になる (図7の “Rendezvous Enabled” 信号)．そして，この信号が各EFSM E_1, E_2, E_3 に送られイベント a が実行される．このようにして表2の r_1 に相当する同期組合せが実行される．

(2) の競合回避部では，互いに競合する複数のイベントが同時に実行可能となった場合に，いずれか一つを選択するための回路である．選択の方法はいくつか考えられるが，本手法では，あらかじめ設定しておいた優先度に従って選択を行うこととした．この選択を行う回路は，プライオリティエンコーダを用いて実現できる．図6では表2より，同期イベント b に対して r_3 と r_4 の二つの同期組合せがある．例えば，r_3 の組合せに対して r_4 の組合せよりも優先度を上げたい場合，図7のように競合回避部を設置すればよい．

また，同期イベントの実行可能性判定に，他のEFSMのイベントの出力値を必要とする場合に備えるため (ガード式の判定に他のEFSMの出力値が必要な場合など)，各同期イベント組の属するEFSM間にデータ転送用パスを設置した．データ転送用パスは，複数の同期イベント組で同時に同期判定される可能性を考慮して，基本的にランデブ情報ごとに設置するが，ある条件を満たすランデブ情報をグループ化し，パスを共用させることで，効率化を計るなどの工夫も考えられる．詳細については文献 [11, 15] で述べている．

3.4 回路合成ツール

我々は，与えられた動作仕様をハードウェア回路化するツールとして，仕様記述言語LOTOS[14]で書かれた動作仕様を，中間ファイル (EFSMとその間のマルチランデブ表) に変換し，レジスタ転送レベルのVHDL記述を生成するためのツールを開発している [12, 13, 15]．また，生成されたVHDL記述はSYNOPSYS社の論理合成ツールF-PGA Express[16] を用いることによってFPGAレベルのハードウェア回路に実装可能である．また，このツールによって，生成する回路のサイズやクロック周波数などもあらかじめ知ることができる．

4 DoS攻撃検出回路の概要

本稿で取り扱うDoS攻撃検出回路は，インターネットバックボーンのような100万パケット/秒以上のトラフィックの高速ネットワーク上のトラフィックを監視しDoS攻撃を検出するための回路である [5]．従来のモニタリング方法では，ネットワーク上にどのようなIPパケットのトラフィックがあったかのログをハードディスク等の高速メディアに記録し，その結果を解析してフラッディングIPアドレスを特定する [4]．この方式では，詳細な監視，分析が可能な反面，高速ネットワークの監視をリアルタイムで行うのは容易ではない．また，そのためのコストも大きくなる．

そこで，本検出回路ではFPGAなどを用いた回路化による高速化の他にパイプライン処理を実装

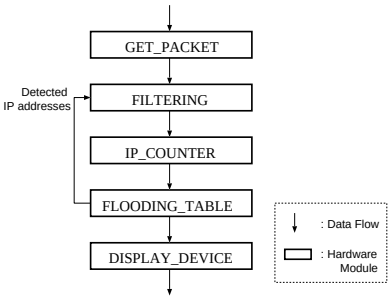

図 9: フラッディング検出モジュールの全体構成

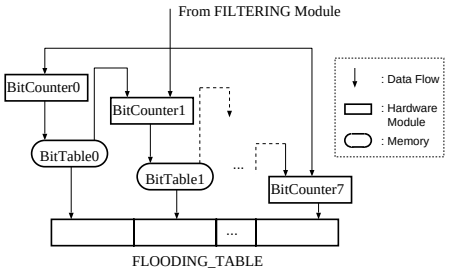

図 10: IP_COUNTER モジュールの構成

することにより，リアルタイム処理に対応することを考える．

また，モニタしている回線を一定時間に通過するパケット数を IP アドレス毎に保持しておくためには，2^{32}(address)*32(bit/address) ビットのメモリ領域が必要となる．ハッシングなどの方法でこの量を減らすことは可能であるが, 本質的には膨大なメモリ領域を必要とすることには変わりがない．

本稿では，構成する FPGA 回路の規模を小さくし，クロック周波数を大きくするため，できるだけメモリーを消費しない (FPGA 自身に格納可能なレジスタのみを使用し，外部メモリーを使用しない) 構成法を考えることにする．

以下では, 2.2 節で述べたプリミティブのうち，以下の 2 つのモジュールの構成と概要について述べる．

（1） IP フラッディング検出回路
（2） SYN フラッド検出回路

4.1 IP フラッディング検出モジュール

2.3 節で述べた Smurf に代表されるような DoS 攻撃は主にある特定のアドレス宛に大量のパケットを送信されることによって行なわれる．そこで, 本検出モジュールではあるネットワーク上においてフラッディングを受けているホストの IP アドレスを特定することを主目的とする．この際, フラッディングを引き起こしているパケットの送信元アドレスも報告する．一般的に送信元アドレスは spoofing によって偽られている場合が多いが, 踏台になっているセキュリティ対策が不完全なホストの検出にも役立つと考えられるためである．

4.1.1 IP フラッディング検出回路の構成

提案する回路の全体構成は図 9のようになる．この回路の入力は図 4のヘッダ情報取得部より得られたパケットヘッダ情報である．

まず GET_PACKET モジュールによってこのヘッダ情報からそのパケットの送信元 (宛先)IP アドレスを取得する．FILTERING モジュールでは, すでにフラッディングを起こしていると判定された一定個数の IP アドレスを保持しておき, 保持しているアドレスと同じアドレスを持つパケットをフラッディング候補から外す. それ以外の IP アドレスの場合は次の IP_COUNTER モジュールにデータを渡す．IP_COUNTER モジュールでは, FILTERING モジュールを通過した 32 ビットの IP アドレスを同一ビット数のいくつかのビット列 (**パーティション**) に分割する．ここでは各パーティションのビット数を**パーティションサイズ**と呼ぶ．各パーティションの区切り方はさまざまで, パーティションサイズが 4bit ならパーティションの数は 8 つであり, 各パーティションのビットパターンは 16 通りになる．各パーティションではビットパターンの数に応じて, 同数のカウンタを並列に設置し, そのビットパターンと一致するパケットの数をカウントする．このカウンタがある一定時間においてあらかじめ設定している閾値を越えればそのビットパターンを被害者 (攻撃者) の IP アドレスの一部であるとみなす．全てのパーティションにおいてビットパターンが決定されるまで同様の処理を繰り返す．このようにして被害者 (攻撃者) の IP アドレスを特定する．なお, この過去に検出された複数のフラッディング IP アドレスを格納しているのが図中の FLOODING_TABLE モジュールである．

分割によるメリットはカウンタの許容範囲を狭くすることによるメモリ領域の削減とそれによる処理の高速化である．ただし, フラッディングしている IP アドレスを先頭から特定していく方法では, IP フラッディングを検出できない場合もあるが (あるドメインからの通信トラフィックの総量が多い場合など), 特定するパーティションの順番を入れ換えることで, ある程度対処できる．

IP_COUNTER モジュールの詳細を図 10に示す．図 10 は 4 ビットずつ 8 つのパーティションに分割している例であり, 並行してカウント処理を行なう 8 つのカウントモジュール BitCounter0-BitCounter7 と検出された IP アドレスを格納する 8 つのメモリ BitTable0-BitTable7 からなる．

まず，一番上位の 4 ビットを担当している BitCounter0 において, ある一定期間に最も頻出したビットパターンをフラッディングを引き起こしているアドレスの一部とみなし, カウント処理と閾値を越えているかどうかの判定によって検出する．

検出されたビットパターンはメモリ BitTable0 に保存しておく．次に BitCounter1 は上位 4bit が BitTable0 の値に一致したアドレスのみカウント処理する．そして同様に頻出したビットパターンを検出し, その値を BitTable1 に保存する．

以上の処理を順に BitCounter7 まで行なう．このように上位から順番に 4bit ずつ特定し, BitTable0 から BitTable7 のビットパターンを結合することによって, 最終的にフラッディングしている IP アドレスを特定する．

なお，これら 8 つの BitCounter と 8 つの BitTable を図 10のように配置することにより，IP ア

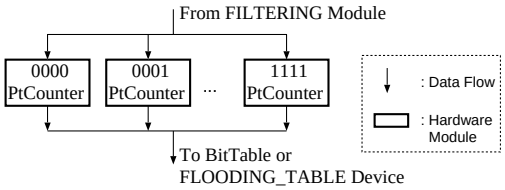

図 11: BitCounter モジュールの構成

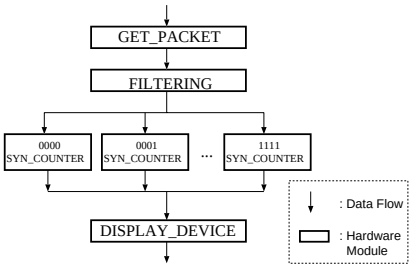

図 12: SYN フラッド検出モジュールの全体構成

ドレスがパイプライン的に処理されるため，1秒間あたりの処理パケット数も多くできる．

4.1.2 BitCounter の処理

パーティションサイズを4bitにしてIPアドレスを分割した場合，0000から1111の2^4個のPtCounterから成る．なお図中の矢印はマルチランデブによる同期を表す(FILTERINGモジュールとは, すべてのPtCounterが同期し, 下位のBitTableとは1つのPtCounterが排他的に同期する)．PtCounterの構成は簡単で, 入力されたビットパターンが自分のビットパターンに一致すればカウンタの値を1増やす．この処理をある一定期間繰返し, ある閾値に一番最初に到達したビットパターンをBitCounterの出力とする．

この閾値はモニタが観測するネットワークの環境によって変化させることができる．また, 与えられた閾値を越えるビットパターンが出現しない場合, BitCounter モジュールはフラッディングが起こっていないものと判断し，適当な間隔でPtCounterのカウンタ値を同時にリセットして最初から再度特定を行う．

4.2 SYN フラッド検出モジュール

4.2.1 SYN Flood Detection Circuit

SYNフラッド検出モジュールの全体構成を図12に示す．4.1節で述べたフラッディング検出モジュールと同様，まず, GET_PACKETモジュールで図4のヘッダ情報取得部より得られたパケット情報を取得し, 必要な情報を抽出する．

ここで抽出する情報は,
（1） 送信元IPアドレス
（2） 宛先IPアドレス
（3） TCPヘッダの制御用フラグビット列
の3つである．送信元, 宛先IPアドレスは共に32bitのビット列である．(3)のフラグビット列は6bitから成り, このビット列によってパケットの種類を表している．この中にはSYNパケット, SYN+ACK

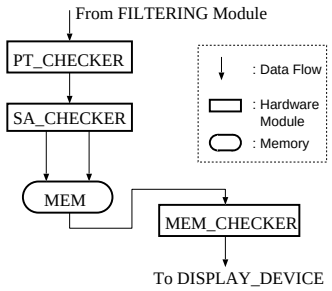

図 13: SYN_COUNTER モジュール

パケット, ACKパケットを区別する(SYN,ACK)の二つのフラグビットを含む．このビット組が(1,0)ならSYNパケット, (1,1)ならSYN+ACKパケット, (0,1)ならACKパケットを表している．

図12のFILTERINGモジュールによって，これらの情報から(SYN,ACK)ビット組を調べ，SYN+ACKパケットとACKパケットのみを抽出する．さらに, FILTERINGモジュールはSYNフラッド被害を受けているホストを検出するために, SYN+ACKパケットならば宛先アドレスを，ACKパケットなら送信元アドレスを抽出する．そして例えば被害者のアドレスのうち4bitのパターンを特定したい場合, 0000から1111のそれぞれのビットパターンに相当するカウンタモジュール(SYN_COUNTER)2^4個にアドレス情報を渡し, ビットパターンの出現回数をカウントする．このSYN_COUNTERモジュールについて次に詳しく述べる．また, 抽出するビットパターンだが, 本手法では32ビットのIPアドレスのうち, ランダムにそのビット列を抽出している．このビットパターンを取得するアルゴリズムは簡単に変更可能であり, ネットワークモニタを設置するネットワークの環境に応じて即座に変更することが可能である．

4.2.2 SYN_COUNTER の処理

IPアドレスを受信した各SYN_COUNTERモジュールは図13 のような構成をしており, 以下の作業を行なう．なお, 図中の矢印は，イベント同期によるデータの受渡しを表す．

まず, PT_CHECKERモジュールによってIPアドレスから指定のビット列を取得し, 各カウンタモジュールが担当するビットパターンと一致しているかどうかを判断する．パターンが一致していれば, 次にSA_CHECKERモジュールに処理を渡す．このモジュールは(SYN,ACK)のビット組に注目し, SYN+ACKパケットかACKパケットかを調べる．もしSYN+ACKパケットなら，メモリMEMの値を1増やす．また，ACKパケットなら，MEMの値を1減らす．これにより，SYN+ACKパケットとACKパケットの対応は,

$$MEM = \#(SYN + ACK) - \#(ACK)$$

のようになる(ただし#(x)はパケットxの数を表す)．SYNフラッドの場合，ACKパケットが返っ

Component Type	IP-FLOOD	SYN-FLOOD
Num. of EFSMs	156	67
Time for Synthesis(sec)	1048	123
Max. Clock Freq.(MHz)	12.45	12.45
Size(gate)	14181	3392

表 3: 各検出回路の論理合成結果

Name	Bit	EFSMs	Area (gates)	Clocks (MHz)
SYN2	2	19	843	13.37
SYN3	3	35	1700	13.37
SYN4	4	67	3392	12.45
SYN5	5	131	7062	11.51
SYN6	6	259	14680	10.55
SYN7	7	515	30848	9.63
SYN8	8	1027	64598	8.67

表 4: bit 抽出数による評価

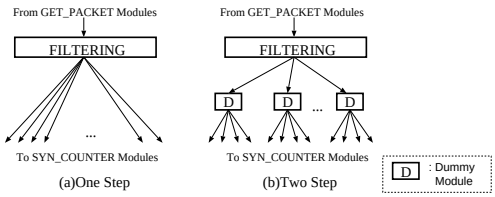

図 14: カウンタ処理の分散

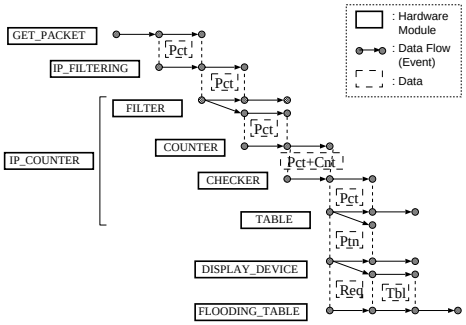

図 15: IP フラッディング回路の一連の処理

てこないので，この MEM の値は増加する．そして, MEM の値がある閾値を越えればそのビットパターンを持つ IP アドレスが SYN フラッドの被害を受けているアドレスとみなす．この判定を行なうのが MEM_CHECKER モジュールである．判定後, MEM_CHECKER モジュールはそのビットパターンを図 12 中のディスプレイデバイスに渡し, 各カウンタモジュールの MEM の値をリセットする．

なお, このカウンタメモリの閾値は環境によって変化させることが可能である．また, SYN+ACK パケットと ACK パケットがそれぞれ違う経路を通過する場合, つまり一方のパケットがネットワークモニタ監視している回線以外を通過する場合は, 本モジュールは SYN フラッドの検出はできない．しかし, 設置場所を工夫するなどの方法で, ほとんどの場合は対処できると思われる．

5 評価

5.1 実装結果

本研究では，上述のネットワークモニタの評価を行なうため，IP フラッディング検出回路 (IP-FLOOD) と SYN フラッド検出回路 (SYN-FLOOD) の二つの回路を論理合成した．表 3は二つの回路の合成結果を表し, IP カウンタのパーティションサイズはいずれも 4bit である．

IP-FLOOD の論理合成にかかった時間は約 17 分で，得られた回路のゲート数は約 14,000gate, そのクロック周波数は約 12.5MHz であった．また SYN-FLOOD の論理合成にかかった時間は約 2 分で，回路の面積は約 3,300gate，クロック周波数は約 12.5MHz であった．

次に, SYN-FLOOD においてパーティションサイズの増加に伴う回路面積とクロック周波数の変化を表 4に示す．抽出するビット数が一つづつ増えるたびにカウンタの数は 2 倍になるので，それに伴い回路面積が 2 倍程度に増加していることが分かる．なお，全てのカウンタの演算を並列で処理しているため，処理速度はそれほど落ちていない．クロック周波数が減っているのはマルチランデブ制御部で制御する同期 EFSM(SYN_COUNTER) の数が増加しているためと考えられる．

また, SYN-FLOOD では図 14(a) のように FILTERING モジュールがすべての SYN_COUNTER と同期をとることによりパケット情報を送信している．fan-out の制限から，同期する SYN_COUNTER の数が多くなると，一度に全部の SYN_COUNTER にデータ転送できないので, 論理合成ツールが自動で fan-out の制限を満たすように中継用の論理ゲートを挿入し，1 クロックで処理しようとするため, FILTERING モジュールのクロック周波数が小さくなってしまう．

そこで, 図 14(b) のように同期データ転送を 2 段 (2 クロック) に分割し, FILTERING モジュールは fan-out の制限を満たす範囲で中継用のダミーモジュール D に 1 クロック目で同期データ転送を行い, 各ダミーモジュール D が SYN_COUNTER と 2 クロック目で同期データ転送させるように修正することにより，回路のクロック周波数の減少を防ぐことができる．このように修正した回路の回路面積とクロック周波数を表 5に示す．表中の項目 Steps は図 14における段数, Dummy はダミーモジュールの数, つまり FILTERING モジュールが直接同期するモジュールの数を指す．また, Branches は各ダミーモジュール (1 段の場合は FILTERING モジュール) から SYN_COUNTER への分岐, つまり同期すべきカウンタの数を表している．回路面積については, 段数の増加, ダミーモジュールの数に応じて若干の増加がみられるのが分かる．また, ダミーモジュー

Name	Steps	Dummy	Branches	Area(gates)	Clocks(MHz)
SYN0-64	1	0	64	7062	11.51
SYN2-32	2	2	32	7171	12.45
SYN4-16	2	4	16	7410	14.90

表 5: 5bit 抽出での段数と分岐数による評価

ルを増やし, 2 段処理を行なうと 1 段で処理するよりも 1〜3MHz の速度向上が見られる.

5.2 合成回路の処理能力

各モジュールの最大クロック周波数と最大クロック周期は以上のような結果が得られたが, 実際に 1 秒間にどれくらいの量のパケットを処理出来るかを考察してみる. 今回作成した IP フラッディング検出モジュールでのパケットの流れと各サブモジュール (EFSM) の関係を図 15に示す. 各サブモジュールは左側の四角で囲んだものであり, 右側の図は EFSM を表す. EFSM 間の丸で囲んだものは転送するデータを表し, Pct はパケット情報, Cnt はカウンタ情報, Ptn はビットパターンを表すビット列, Req は外部からのリクエスト信号, Tbl はフラッディング IP アドレスの集合を表す.

図 15から分かる通り, それぞれのサブモジュールでの一連の処理を表す遷移 (イベント系列) は高々3 つの遷移 (イベント) からなる. 各イベント系列はそのイベント系列の全ての遷移を実行後に初期状態に戻る. 各サブモジュールは独立に動作しており, パイプライン処理可能である. 原則 1 つの遷移を実行するために必要なクロック数は 1 クロックとしている. よって 1 秒間に処理できるパケット数は全体のクロック周波数が 12.45MHz で, 遷移数がたかだか 3 遷移なので, 12.45(MHz) / 3 = 415 万パケット/s の処理が可能である. ただし, 実際にはパケット分割処理や, カウント処理などにおいて付加的な作業が追加される場合もある. しかしそのような場合, 例えば, イベント系列全体で 10 遷移かかるような場合でも, 12.45(MHz) / 10 > 100 万パケット/s の処理が可能である.

5.3 閾値の妥当性

次に閾値の妥当性について考察してみる. IP フラッディング検出モジュールも SYN フラッド検出モジュールも取得したパケットの IP アドレスの一部からそのビットパターンに応じて各々のカウンタモジュールにパケット情報を渡し, カウント処理を行っている. カウンタモジュールは担当するビットパターンのパケット数がある値 (閾値) を越えた段階でフラッディングとみなす. そこで 100 万パケット/s のトラフィックにおいてカウンタ 1 個あたり処理しなければならないパケット数とその閾値の関係について考察する.

フラッディング検出モジュールで, 例えば, 通常 100 万パケット/s のトラフィックで 8 万パケット/s のフラッディングを検出したい場合を考える. この場合, IP アドレスをパーティションサイズ 4 ビットで 8 個パーティションに区切り, 各パーティションごとに IP_COUNTER モジュールでビットパターンをカウントする場合, それぞれの IP_COUNTER に 100 万/16 = 6.3 万パケット/s の入力がある. 入力される IP アドレスにはビットパターンの偏りがあると考えられる. いま, IP_COUNTER のアドレスに最大で 50%の変動があると仮定すると, 各 IP_COUNTER は 6.3 万± 50% = 3.1 万〜9.5 万パケット/s の入力があると考えられる. この場合, IP_COUNTER 中の各 BitCounter の閾値を 10.5 万/s に設定する. そうすると, ある特定の IP アドレスが 8 万パケット/s 増えた場合, その IP アドレスのパケット総数は, 少なくとも (3.1 万 +8 万) > 10.5 万となるため, IP アドレスのビットパターンの偏りに関わらず必ず 10.5 万/s を越えるので, 4 ビットづつ 8 個の IP_COUNTER で検出できる.

一方, 例えば 8 ビットずつ 4 個の IP_COUNTER を利用すると, 1 個の IP_COUNTER あたり 100 万 /256 = 3,906 パケット/s となるので, 閾値を 1 万程度に設定すると, ビットパターンの偏りに関わらず必ずフラッディングを検出できる. ただし, その場合, IP_COUNTER 中の BitCounter の総数は 256*4 = 1024 個となり, 回路サイズが BitCounter の総数に応じて増大する.

次に SYN フラッド検出モジュールについて考える. インターネットバックボーンを流れる SYN+ACK,ACK パケットの量は最大で 5%程度であることが分かっている [10]. つまり上記のようなトラフィックを考える場合, SYN フラッド検出モジュールは最大で 5 万パケット/s の処理を行なわなければならない. また SYN フラッドは攻撃したいホストに対して 500 パケット/s 程度送信すればよいことが分かっている [10].

このため, 例えば, SYN_COUNTER モジュールを 7bit にしてカウント処理を行う場合, カウンタ数は 128 個であり, 1 カウンタあたり処理しなければならないパケット数は 5 万パケット / 128= 391 パケット (平均値) となる. 上記同様, IP アドレスに最大で 50%の変動があると仮定すると, 各カウンタは 391 ± 50% = 200〜600 パケット/s の入力があると考えられる. この場合, カウンタの閾値を 650 程度に設定すると, あるホストに対して 500 パケット/s の攻撃用の SYN+ACK パケットが送られた場合でも, そのことを検出できる. 各 SYN_COUNTER モジュールを 8bit に設定すると, 250 パケット/s 程度の攻撃用 SYN+ACK パケットの検出に利用できる. 表 4に SYN_COUNTER モジュールのビット幅と回路面積, 回路スピードの関係について記載する.

5.4 回路構成の変更への対処

前節ではパーティションサイズの変化による合成回路のサイズと処理能力について述べた. パーティションサイズの変更が必要な場合, EFSM の数もそれに応じて変更しなければならない.

文献 [12, 13] では,LOTOS のあるサブクラスから並行同期 EFSM 群への変換法を提案しており, その

サブクラスでは,動的にプロセスを生成するような仕様を記述できる. ただし,生成される最大プロセス数はガードで制限されなければならない. (例えばガード式 $[x < C]- > P$ で表せる. ただし, C は定数).

本サブクラスを用いることで, IP フラッディング検出回路において, 何ビットずつアドレスを特定するか, AND 回路の fan-in を少なくするため, 取得した IP アドレスのカウンタ処理を行なう EFSM への分配を何段で行なうかなどの値をパラメタ化し, 動的に必要なプロセスを生成するような仕様を記述し, 並行同期 EFSM 群モデルにおける固定数の EFSM 群に変換できる. 以上の方法により, 仕様中のパラメタを変更するだけで, 最終的な回路構成を変更できるため, 様々なネットワーク環境に適したモニタ回路を容易に合成することができる.

6 まとめ

本稿では, DoS 攻撃を検出する回路の提案と実装法について説明した. また, 本手法を用いて生成された FPGA 回路は毎秒 100 万パケット以上のパケットを処理し, DoS 攻撃を検出することが可能であり, 面積・速度共に実用上問題ないことを確認した.

また, この DoS 攻撃検出回路は大量のパケットをリアルタイムに監視をし, そのモニタリングの処理速度はギガビットイーサネットなどの高速ネットワーク上でも処理可能な速度である. そのため, この手法を用いて同様の高速ネットワーク上で動作するネットワークモニタのプリミティブを表す FPGA 回路を作成することも可能である.

我々はさまざまな種類の DoS 攻撃を検出するための基本モジュールの開発を行なっている. このような基本モジュールを我々の自動合成ツールに組み込むことによって, さまざまなタイプの DoS 攻撃を検出することができると考えられる. 本稿で紹介したネットワークモニタを実際にインターネットバックボーンに接続し, パケットをモニタリングすることが今後の課題である.

参考文献

[1] A. S. Tanenbaum: "Computer Networks, Third Edition ". Prentice-Hall Inc. (1996).

[2] S. Yagi, T. Ogura, T. Kawano, M. Maruyama and N. Takahashi: "METAMONITOR: An Adaptive Network-traffic Monitor", Journal of Information Processing Society of Japan, Vol.41, No.2, pp. 444-451 (2000) (in Japanese).

[3] Z. D. Ditta, J. R. Cox Jr and G. M. Parulkar: "Design of the APIC: A High Performance ATM Host-Network Interface Chip", Proc. of IEEE INFOCOM'95, pp. 179-187 (1995).

[4] J. Apisdorf, K. Claffy and K. Thompson: "OC3MON:Flexible, Affordable, High-Performance Statistics Collection", Proc. of INET'97 (1997), http://www.isoc.org/isoc/whatis/conferences/inet/97/proceedings/F1/F1_2.HTM

[5] K. Claffy, G. J. Miller and K. Thompson: "The Nature of the Beast Recent Traffic Measurements from an Internet Backbone", Proc. of INET'98 (1998), http://www.caida.org/outreach/papers/Inet98/

[6] L. Garber: "Denial-of-Service Attacks Rip the Internet", Proc. of IEEE Computer, pp. 12-17 (2000).

[7] V. Paxson: "Bro: A System for Detecting Network Intruders in Real-Time", Computer Networks, Vol. 31, No.23-24, pp. 2435-2463 (1999).

[8] K. Park and H. Kee: "On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets", Proc. of ACM SIGCOMM2001, pp. 15-26 (2001).

[9] Glenn Mansfield et. al: "Towards Trapping Wily Intruders in the Large", Computer Networks, Vol. 34, pp. 659-670 (2000).

[10] D. Moore, G. M. Voelker and S. Savage: "Inferring Internet Denial-of-Service Activity", USENIX Security Symposium (2001).

[11] A. Kitajima, K. Yasumoto, T. Higashino and K. Taniguchi: "A Method to Convert Concurrent EFSMs with Multi-Rendezvous into Synchronous Sequential Circuit", IEICE Trans. on Fundamentals, Vol. E81-A, No. 4, pp. 566 – 575 (1998).

[12] K. Yasumoto, A. Kitajima, T. Higashino and K. Taniguchi: "Hardware Synthesis from Protocol Specifications in LOTOS", Proc. of Joint Intl. Conf. on 11th Formal Description Techniques and 18th Protocol Specification, Testing, and Verification (FORTE/PSTV'98), pp. 405-420 (1998).

[13] H. Katagiri, M. Kirimura, K. Yasumoto, T. Higashino and K. Taniguchi: "Hardware Implementation of Concurrent Periodic EFSMs", Proc. of Joint Intl. Conf. on 13th Formal Description Techniques and 20th Protocol Specification, Testing, and Verification (FORTE/PSTV'2000), pp. 285-300 (2000).

[14] ISO : "Information Processing System, Open Systems Interconnection, LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behavior", ISO 8807 (1989).

[15] H. Katagiri, K. Yasumoto, A. Kitajima, T. Higashino and K. Taniguchi: "Hardware Implementation of Communication Protocols Modeled by Concurrent EFSMs with Multi-Way Synchronization", 37th IEEE/ACM Design Automation Conference (DAC'2000), pp. 762-767 (2000).

[16] SYNOPSYS, Inc. : http://www.synopsys.com