ARQUITETURA MULTIPROCESSADA E RECONFIGURÁVEL PARA A SÍNTESE DE REDES DE PETRI EM *HARDWARE*

Tiago de Oliveira[*]
tiagoooli@yahoo.com.br

Norian Marranghello[†]
norian@ibilce.unesp.br

[*]Av. Brasil Norte, 364
PPGEE/FEIS/UNESP
Ilha Solteira, São Paulo, Brasil

[†]Rua Cristóvão Colombo, 2265
DCCE/IBILCE/UNESP
São José do Rio Preto, São Paulo, Brasil

ABSTRACT

A multiprocessed reconfigurable architecture aimed at physical implementation of Petri Nets has been developed in VHDL, and mapped onto an FPGA. Conventionally, Petri nets are described with a hardware description language at the register transfer level, and a high level synthesis process is used to generate the boolean functions and the state transition tables needed to map them onto an FPGA (Morris et al., 2000) (Soto and Pereira, 2001). The proposed architecture has reconfigurable logic blocks especially developed for the implementation of the places and transitions of the Petri net, so, neither it is necessary to translate the Petri net model to obtain descriptions at intermediate abstraction levels, nor a synthesis process to map the net onto the architecture is in order. Petri net models that can tell tokens apart as well as Petri nets with timed transitions can be implemented. The architecture comprises a reconfigurable array of processors, and a dynamic communication system connecting the processors of the array. Each processor of the array represents the behavior of a Petri net transition, and the communication system has a number of routers to direct data packets among the processors. The proposed architecture was validated on a 10,570 logic elements FPGA with Petri Nets of up to 9 transitions and 36 places, achieving 15.4ns latency and 17.12GB/s throughput with 64.58MHz clock frequency.

KEYWORDS: Reconfigurable Architecture, FPGA, Systems Synthesis, Petri Nets.

RESUMO

Uma arquitetura reconfigurável e multiprocessada para a implementação física de Redes de Petri foi desenvolvida em VHDL e mapeada sobre um FPGA. Convencionalmente, as Redes de Petri são transformadas em uma linguagem de descrição de *hardware* no nível de transferências entre registradores e um processo de síntese de alto nível é utilizado para gerar as funções booleanas e tabelas de transição de estado para que se possa, finalmente, mapeá-las num FPGA (Morris et al., 2000) (Soto and Pereira, 2001). A arquitetura proposta possui blocos lógicos reconfiguráveis desenvolvidos exclusivamente para a implementação dos lugares e das transições da rede, não sendo necessária a descrição da rede em níveis de abstração intermediários e nem a utilização de um processo de síntese para realizar o mapeamento da rede na arquitetura. A arquitetura permite o mapeamento de modelos de Redes de Petri com diferenciação entre as marcas e associação de tempo no disparo das transições, sendo composta por um arranjo de processadores reconfiguráveis, cada um dos quais representando o comportamento de uma transição da Rede de Petri a ser mapeada e por um sistema de comunicação, implementado por um conjunto de roteadores que

Artigo submetido em 31/01/2007 (Id:830)
Revisado em 10/09/2008 e em 01/12/2008
Aceito sob recomendação do Editor Associado Prof. José Reinaldo Silva

são capazes de enviar pacotes de dados de um processador reconfigurável a outro. A arquitetura proposta foi validada num FPGA de 10.570 elementos lógicos com uma topologia que permitiu a implementação de Redes de Petri de até 9 transições e 36 lugares, atingindo uma latência de 15,4ns e uma vazão de até 17,12GB/s com uma freqüência de operação de 64,58MHz.

PALAVRAS-CHAVE: Arquitetura Reconfigurável, FPGA, Síntese de Sistemas, Redes de Petri.

1 INTRODUÇÃO

A constante evolução tecnológica tem possibilitado a implementação de funções progressivamente mais complexas e a complexidade dos sistemas digitais atuais exige que eles sejam analisados e verificados detalhadamente, no intuito de minimizar os erros de implementação. Os erros de projeto de um produto no mercado atual podem ser extremamente custosos não apenas em termos financeiros, mas também afeta a imagem do fabricante. Outrossim, é importante automatizar o processo de síntese desses sistemas para viabilizar a inserção dos produtos resultantes no mercado o mais rapidamente possível, pois o tempo de desenvolvimento é crítico para o sucesso do produto.

Uma metodologia de projeto deve prover ferramentas para a modelagem de sistemas num nível elevado de abstração, além de possibilitar a verificação, a validação e a implementação de sistemas cada vez mais complexos. Dentro deste contexto, a Rede de Petri (Wang, 1998) (Murata, 1989) tem se mostrado uma linguagem formal poderosa para especificar e modelar o comportamento algorítmico de sistemas paralelos síncronos e assíncronos num nível de abstração bem elevado e além disso, como ferramenta matemática, possui uma grande quantidade de propriedades e métodos de análise que auxiliam o projetista na análise e interpretação do comportamento e da estrutura do modelo especificado visando eliminar os erros de projeto.

Este artigo aborda o desenvolvimento de uma arquitetura reconfigurável e multiprocessada que pode ser utilizada para a implementação física de sistemas de controle modelados por meio de Redes de Petri. A arquitetura reconfigurável e multiprocessada proposta foi desenvolvida e mapeada sobre um FPGA (*Field-Programmable Gate Array*). Com a implementação desta arquitetura proposta sobre o FPGA, o projetista poderá utilizar uma plataforma de trabalho para implementar Redes de Petri em *hardware*. Inicialmente, o sistema a ser implementado deverá ser descrito graficamente por meio de uma Rede de Petri utilizando esta plataforma de trabalho. Após a sua descrição, a plataforma realizará alguns cálculos básicos para a verificação de erros na estrutura da rede e na sintaxe utilizada, para evitar, por exemplo, que um mesmo arco ligue duas transições ou dois lugares da rede, o que configura um erro no modelo descrito. Além disso, a plataforma executará alguns processos intermediários para permitir o mapeamento tecnológico na arquitetura proposta, ou seja, os componentes constituintes da arquitetura são programados para permitir o mapeamento do modelo de Rede de Petri a ser implementado. Vale a pena ressaltar, que o FPGA é configurado apenas uma vez para implementar a arquitetura proposta. Como a arquitetura proposta também é reconfigurável, o modelo de Rede de Petri a ser implementado é diretamente mapeado nos componentes dessa arquitetura. Portanto, o modelo de Rede de Petri a ser implementado é configurado sobre a arquitetura proposta e não sobre o FPGA.

Atualmente, já são utilizados FPGAs para implementar fisicamente uma Rede de Petri (Soto and Pereira, 2001) (Morris et al., 2000). Porém, como um FPGA contém basicamente pequenos blocos lógicos, tabelas de programação e barramentos programáveis, a Rede de Petri a ser implementada em FPGA normalmente é transformada em uma linguagem de descrição de *hardware* e na seqüência realiza-se a síntese de alto nível para que se possa programar as funções booleanas e tabelas de transição de estados nos recursos do FPGA. Convencionalmente, o processo de síntese gera descrições RTL (*Register Transfer Language*) a partir de uma descrição do sistema em alto nível, e então definem-se equações booleanas e tabelas de transição de estados para que se possa finalmente realizar o mapeamento tecnológico, que consiste na alocação das equações booleanas e tabelas de transição de estados nos recursos disponíveis do FPGA.

A arquitetura que está sendo aqui proposta permite um mapeamento tecnológico diretamente no nível comportamental ou sistêmico, como mostrado na figura 1. Desta forma, não é necessário transformar o modelo de Rede de Petri em descrições com níveis de abstração intermediários e nem realizar um custoso processo de síntese de alto nível para mapear a Rede de Petri no FPGA. Num processo de síntese de alto nível convencional, a transformação de uma especificação comportamental numa descrição funcional é um processo longo, sendo decomposto numa sucessão de tarefas, tais como ordenação e alocação de recursos de *hardware*, geração de grafos e árvores, verificação de dependência entre operações e cobertura e associação bibartite de grafos (Strum et al., 1999).

A arquitetura proposta é composta de blocos lógicos reconfiguráveis exclusivamente desenvolvidos para a implementação dos estados (lugares) e das ações (transições) de uma Rede de Petri. Assim, a Rede de Petri pode ser diretamente mapeada na arquitetura, sem a necessidade de se utilizar um processo de síntese de alto nível para descrever o sistema por meio de equações booleanas e tabelas de transição de esta-

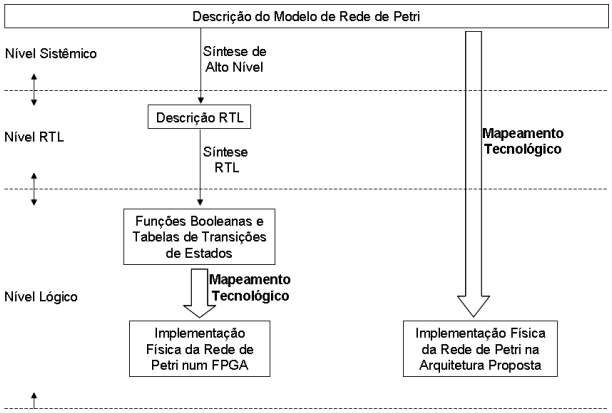

Figura 1: Processo de mapeamento tecnológico proposto

dos.

Comparativamente com o processo de síntese de alto nível, o processo de mapeamento que está sendo proposto pode reduzir significativamente a quantidade de tarefas a serem executadas na obtenção de uma arquitetura configurada para executar o comportamento algorítmico de uma Rede de Petri. A redução no número de tarefas pode tornar o processo de síntese menos complexo e evitar a ocorrência de alguns problemas algorítmicos que podem ocorrer durante o processo de síntese de alto nível convencional(Strum et al., 1999). Dificuldade de paralelizar operações executadas em diferentes módulos arquiteturais durante a síntese hierárquica do circuito, e a possibilidade das unidades de controle e memória consumirem uma grande quantidade de recursos, podem fazer com que a execução do processo de síntese de alto nível leve um tempo excessivamente longo, ou mesmo torne inviável a obtenção de uma descrição RTL a partir do comportamento algorítmico do sistema (Strum et al., 1999).

2 IMPLEMENTAÇÕES FÍSICAS EXISTENTES DE SISTEMAS MODELADOS EM REDES DE PETRI

A implementação em *hardware* pode ser subdividida em realizações indiretas e diretas (Gomes, 1999). As realizações indiretas têm por base a tradução da Rede de Petri numa representação intermediária, como por exemplo um grafo de estados, o qual será posteriormente sintetizado em equações lógicas (Gomes, 1999). A grande desvantagem desse tipo de realização indireta é a grande quantidade de memória necessária para a implementação, visto que o grafo de estados associado a uma Rede de Petri pode se tornar bastante extenso.

Por sua vez, as realizações diretas de Redes de Petri em *hardware* baseiam-se numa tradução onde os elementos da rede, isto é, estados e ações, são implementados por meio de componentes digitais pré-estabelecidos que emulam tais elementos (Gomes, 1999). Para tanto, pode-se utilizar um processo de síntese que realize a conversão de uma descrição do sistema em Redes de Petri para uma descrição cujo nível de detalhamento viabilize uma implementação. As linguagens mais populares para realizar este refinamento a um nível RTL e de portas lógicas são VERILOG e VHDL (Rokyta et al., 2000).

De modo geral, a partir de uma descrição de uma Rede de Petri de alto nível, passa-se por um processo de obtenção de uma Rede de Petri de nível menos elevado, a qual por sua vez, é utilizada para a geração do código VHDL, sendo este último necessário como entrada para um processo de síntese que poderá transformar a especificação num conjunto de portas lógicas devidamente interconectadas (Rokyta et al., 2000).

Para o processo de mapeamento tecnológico tem-se utilizado em larga escala FPGAs, visto que a arquitetura de um único dispositivo FPGA possui milhares de células lógicas programáveis que estão prontamente disponíveis e a baixo custo (Soto and Pereira, 2001).

A simulação de uma Rede de Petri em *hardware* reduz significativamente o tempo de processamento se comparado ao de uma simulação em processadores seqüênciais convencionais devido ao alto grau de paralelismo existente na avaliação da habilitação de todas as ações da rede, na seleção de uma ação para disparar, e na determinação do próximo estado da rede quando ocorrer o disparo da ação selecionada. Para o propósito de simulação de uma Rede de Petri, foi desenvolvido na Universidade de Canberra o processador reconfigurável Achilles (Morris et al., 2000), o qual define uma placa eletrônica que permite integrar vários FPGAs em pilhas, possibilitando a simulação de um número elevado de elementos da rede.

Uma técnica de projeto utilizada para otimizar o mapeamento de Redes de Petri em FPGAs, ou seja, permitir que um maior número de elementos da Rede de Petri sejam incluídos num único FPGA, define um bloco lógico contendo apenas um lugar e uma transição interligados (Soto and Pereira, 2001). Realizando o mapeamento desta estrutura para um único bloco lógico do FPGA pode-se otimizar a quantidade de elementos da Rede de Petri. Outra forma de otimização consiste em codificar as marcações da rede através de um número reduzido de bits.

Outra forma de implementação de Redes de Petri em *hardware* baseia-se no gerenciamento de um conjunto de processos paralelos através da implementação de um controlador definido por meio de uma Rede de Petri (Anzai et al., 1993) (Kamakura et al., 1997). Neste caso, a Rede de Petri é armazenada no controlador em forma de tabelas, as quais se

baseiam na estrutura da rede. A cada lugar da Rede de Petri atribui-se um determinado trabalho, o qual deve ser realizado pelos processos paralelos quando uma nova marca for produzida.

A estrutura da Rede de Petri também pode ser implementada para a realização de análises. Nestes casos, a implementação da Rede de Petri em *hardware* pode ser considerada como um co-processador específico para acelerar o processo de obtenção da análise da Rede de Petri. Em (Csertán et al., 1997) descreve-se um simulador de Rede de Petri para a geração do grafo de alcançabilidade de uma Rede de Petri qualquer. Em (Bundell, 1997) propôs-se uma arquitetura na qual um co-processador integrado com um ambiente de trabalho seria utilizado para acelerar a execução de simulações de Redes de Petri.

Neste contexto, a arquitetura que está sendo proposta apresenta algumas inovações e aperfeiçoamentos, como explicados nos parágrafos a seguir.

Uma das inovações é a possibilidade de mapeamento tecnológico no nível sistêmico, pois a arquitetura possui blocos lógicos desenvolvidos especificamente para implementar os estados e as ações da Rede de Petri, não sendo necessário, portanto, a descrição da rede em níveis de abstração intermediários, como VHDL ou VERILOG e nem a utilização de um processo de síntese para descrever o código gerado em funções booleanas e tabelas de transição de estados, o que ocorre atualmente para a implementação de Redes de Petri em FPGAs.

Além disso, as arquiteturas comentadas acima utilizam Redes de Petri classificadas de baixo nível (Redes de Petri ordinárias e generalizadas), como as redes lugar/transição (Reisig, 1992) e elementares (Thiagarajan, 1987). A arquitetura proposta possibilita o mapeamento direto de uma rede de alto nível, no caso, as Redes de Petri coloridas de arcos constantes. Uma Rede de Petri Colorida de arcos constantes (Reisig, 1992) (Jensen, 1997) é constituída por: lugares, transições e arcos, semelhante a uma Rede de Petri lugar/transição; tipos ou cores que individualizam as marcas da rede; uma marcação inicial que identifica, para cada lugar, as quantidades e os tipos de marcas disponíveis; e um rótulo em cada arco indicando as quantidades e os tipos de marcas que devem ser adicionados ou removidos em cada lugar no disparo das transições. A vantagem dessas redes em relação às redes de baixo nível refere-se, do ponto de vista do projetista, a uma maior facilidade gráfica para a modelagem do sistema (Jensen, 1997).

As arquiteturas mencionadas acima são utilizadas para a simulação ou análise comportamental/estrutural da Rede de Petri, desta forma, essas arquiteturas permitem que apenas um conjunto pré-estabelecido de transições sejam simultaneamente disparadas (Morris et al., 2000) (Csertán et al., 1997). A arquitetura proposta permite a implementação (e não só a simulação) da Rede de Petri em sua forma mais original, ou seja, todas as transições habilitadas podem ser disparadas simultaneamente.

Outra inovação da arquitetura proposta refere-se à possibilidade de mapeamento direto de algumas extensões das Redes de Petri, permitindo ao projetista a implementação de Redes de Petri com um poder de modelagem maior (Wang, 1998). A arquitetura permite, atualmente, o mapeamento de Redes de Petri T-temporizadas (Wang, 1998), autônomas e sincronizadas (David and Alla, 1992). Porém a arquitetura poderá ser adaptada para processar Redes de Petri temporais (Wang, 1998) e até mesmo redes estocásticas (Marsan, 1990) (Wang, 1998), aumentado em muito o poder de modelagem dos sistemas, pois essas Redes de Petri não podem ser convertidas em redes de baixo nível atemporais (Wang, 1998).

3 ARQUITETURA PROPOSTA

A arquitetura é composta por um arranjo de processadores e por um sistema de comunicação. Cada processador do arranjo representa o comportamento de uma transição da Rede de Petri a ser mapeada e os seus respectivos lugares de entrada. O sistema de comunicação é implementado por um conjunto de roteadores. A figura 2 mostra o diagrama de blocos da arquitetura proposta, onde R indica um roteador.

O arranjo de processadores é dedicado à execução das transições da Rede de Petri. Cada transição da Rede de Petri é mapeada a um processador e o sistema de comunicação é usado para conectar os processadores do arranjo. O sistema

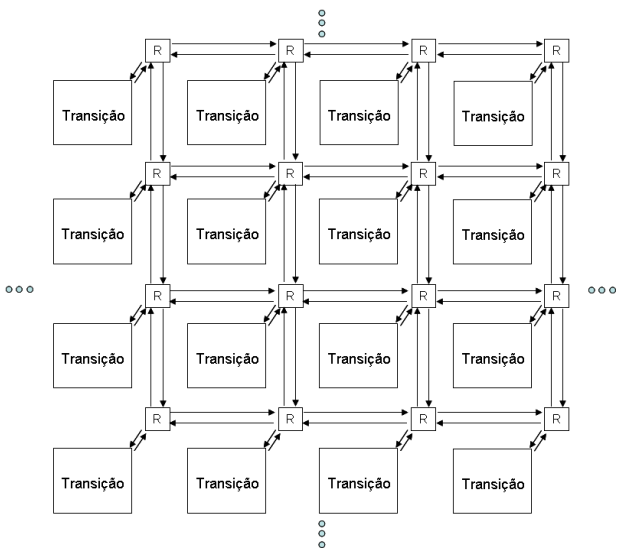

Figura 2: Arquitetura proposta

de comunicação é composto por um grupo de roteadores distribuídos em uma topologia de malha bidimensional. Desta forma, o sistema pode ser configurado para imitar a estrutura da Rede de Petri. O arranjo de processadores da arquitetura realiza um processamento paralelo e cada processador é responsável pelo disparo da transição mapeada. Caso a transição mapeada tiver marcas suficientes para o disparo, o processador correspondente será responsável pela produção de pacotes de dados que indicam a geração de novas marcas, configurando-se assim, o disparo da transição mapeada. Esses pacotes de dados são transportados pelo sistema de comunicação para outros processadores do arranjo de tal forma a informar o disparo da transição correspondente. Cada processador possui um grupo de registradores responsável pelo armazenamento das marcas geradas. Um comparador de marcas, implementado em cada processador do arranjo, é usado para ler simultaneamente o conteúdo de todos os registradores e indicar a presença (ou ausência) de marcas suficientes para o disparo da transição correspondente.

3.1 Sistema de Roteamento

O roteador possui cinco portas de comunicação nomeadas de Norte (N), Sul (S), Leste (L), Oeste (O) e uma para a transição (T). Uma porta de comunicação consiste de dois barramentos unidirecionais ponto-a-ponto entre dois roteadores, ou entre um roteador e um processador do arranjo. Cada porta de comunicação possui três diferentes sinais, sendo: PAP (pedido de armazenamento de pacote), PACA (indica o armazenamento de pacote) e PACOTE (sinaliza os bits representantes do pacote que está sendo enviado). As portas N, S, L e O são responsáveis pela comunicação entre os roteadores vizinhos e a porta T é responsável pela transferência de pacotes entre um processador do arranjo e o sistema de comunicação, representado pelo conjunto de roteadores. O arranjo de processadores é responsável, além de todo o processamento de dados do sistema, por injetar pacotes na rede e definir o seu endereço de destino de tal forma que a rede de comunicação possa se encarregar de enviar os pacotes ao destino definido.

O pacote enviado pelos roteadores da arquitetura proposta é subdividido em apenas dois campos, quais sejam: cabeçalho e carga útil. O cabeçalho inclui as variacões nos eixos x e y (Δx e Δy), e as direções leste-oeste ou oeste-leste, e norte-sul ou sul-norte. A carga útil é efetivamente a informação que deve ser entregue ao processador do arranjo. As variações nos eixos x e y constituem o endereço de destino do pacote e as direções indicam a trajetória. Desta forma, por exemplo, se o cabeçalho de um pacote possuir uma variação no eixo X de 2, $\Delta x = 2$, uma variação no eixo Y de 3, $\Delta y = 3$, e indicar como direções o trajeto norte-sul e oeste-leste, significa que o pacote deverá percorrer três roteadores no eixo X, direção oeste-leste e depois percorrer quatro roteadores no eixo Y, direção norte-sul. Através da técnica de roteamento X-Y (Culler et al., 1997), o roteador decrementa uma posição de Δx até que $\Delta x = 0$. Depois, decrementa-se uma posição de Δy até que $\Delta y = 0$. Quando $\Delta x = 0$ e $\Delta y = 0$ o pacote chegou ao seu destino e o roteador deve entregá-lo ao processador nele conectado retirando-se, com isso, o pacote da rede de comunicação.

A arquitetura proposta pode ser considerada uma Rede-em-*Chip*, a qual possui uma estrutura de comunicação que possibilita o projeto de sistemas escaláveis e reutilizáveis, sendo uma alternativa às arquiteturas que vêm sendo utilizadas nos *Systems-on-Chips* atuais, como barramentos e canais dedicados (Benini and Micheli, 2002) (Dally and Towles, 2001). A rede-em-*chip* é baseada nos fundamentos já consolidados das redes de computadores e pode ser definida como um conjunto de roteadores interligados por canais ponto-a-ponto cujo modelo de comunicação é, tipicamente, o de troca de mensagens, sendo que a comunicação entre unidades de processamento, as quais são acopladas aos roteadores, é feita pelo envio e recebimento de mensagens de requisição e de resposta (Zeferino, 2003) (Dally and Towles, 2001) (Liu et al., 2004).

Contudo, modelos básicos de roteadores (Guerrier and Greiner, 2000) (Zeferino et al., 2002) (Pande et al., 2003) citados na literatura de Redes-em-*Chip* subdividem o pacote em um conjunto de quadros, e estes são transmitidos pela rede. Como a idéia em Redes-em-*Chip* é realizar um sistema de comunicação capaz de se adaptar a diferentes unidades de processamento num sistema heterogêneo, tais unidades de processamento podem enviar pacotes de tamanhos variados e assim, a subdivisão de pacotes em quadros se torna necessária. Os modelos de Redes de Petri que podem ser implementados na arquitetura proposta realizam uma comunicação de granularidade fina, ou seja, as informações necessárias para determinar o disparo de uma transição são enviadas em um único pacote. Assim, utilizando um sistema de comunicação de granularidade fina, não é necessário subdividir os pacotes em quadros reduzindo-se, assim, a lógica que controla o fluxo de pacotes em cada roteador.

Levando em consideração a granularidade fina da Rede de Petri e um sistema multiprocessado homogêneo, foi elaborada a estrutura deste roteador, mostrada num diagrama de blocos da figura 3. O roteador é capaz de atender vários pedidos de armazenamento de pacotes simultaneamente, isto é, os pacotes provenientes dos roteadores vizinhos e do processador do arranjo correspondente podem ser armazenados simultaneamente em um banco de registradores implementado internamente no roteador. Um pedido de armazenamento de pacote só não será atendido num único ciclo de relógio em dois casos: no primeiro caso, o registrador pertencente ao canal de comunicação aonde se deve armazenar o pacote está

ocupado com outro pacote proveniente de um pedido atendido anteriormente; e no segundo caso, há, no mesmo ciclo de relógio, outro pedido com prioridade superior para o armazenamento de pacote num mesmo registrador. Neste segundo caso, o pedido cuja prioridade momentânea é superior será atendido primeiro, postergando o atendimento dos outros pedidos para os próximos ciclos de relógio.

Internamente, o roteador possui cinco decodificadores de prioridade, um para cada porta de comunicação existente; três decrementadores para o eixo X, para o tratamento de pacotes provenientes do leste, do oeste e do processador nele acoplado; cinco decrementadores para o eixo Y, para o processamento de pacotes provenientes do norte, do sul, do processador nele acoplado, do leste com destino ao norte ou ao sul, do oeste com destino ao norte ou ao sul; cinco seletores de pacote, cinco *flip-flops* e cinco registradores, um para cada porta de comunicação; além da lógica de roteamento. A ordem de prioridade para o atendimento às portas de comunicação é variável e foi implementado o *round-robin*.

3.2 Sistema de Processamento

Cada processador da arquitetura, como mostrado na figura 4, possui dois bancos de registradores e uma memória. Um dos bancos possui quatro registradores de 18 bits cada. Neste banco são armazenados os pacotes de dados provenientes do sistema de roteamento, mais precisamente, armazena-se neste banco as marcas geradas devido ao disparo de outras transições da arquitetura. Um processador do arranjo só enviará as marcas geradas para as transições que estiverem diretamente ligadas aos lugares de saída da transição que está sendo disparada. Cada registrador é responsável pelo armazenamento de um tipo diferente de marca. Assim sendo, cada processador poderá armazenar até quatro tipos diferentes de marcas e cada tipo poderá ter de 0 até $2^{18} - 1$ marcas. Ao receber um pacote de dados contendo a informação do tipo e da quantidade de marcas geradas, o processador do arranjo irá armazenar a soma entre o valor atualmente armazenado com a quantidade informada no pacote de dados.

O segundo banco representa quatro registradores de 18 bits cada. Este banco armazena o valor do peso dos arcos pré-definidos pelo modelo da Rede de Petri que está sendo mapeada. Este banco é utilizado para que se possa comparar a quantidade de marcas geradas nos respectivos lugares de entrada com a quantidade de marcas necessárias para habilitar a transição. Unidades digitais de subtração são capazes de identificar quando as marcas geradas são suficientes para a habilitação e o disparo da transição. Quando uma transição estiver habilitada, ela é disparada, isto é, a unidade de controle de envio de pacotes se encarrega de armazenar os valores subtraídos no banco de registrados de marcas geradas, consumindo as marcas dos lugares de entrada da transição.

A memória armazena até oito palavras de 32 bits cada. Nesta memória se encontram os pacotes que devem ser enviados às transições que possuem lugares de entrada e saída em comum com a transição que está sendo disparada. Quando ocorrer o disparo de uma transição, o processador correspondente deve enviar os pacotes de dados para a rede de comunicação informando as marcas que devem ser removidas dos lugares de entrada e as marcas que devem ser criadas nos lugares de saída. Deve-se enviar um pacote contendo 12 bits para o endereço de destino do pacote, dois bit para o tipo de marca que deve ser gerada/subtraída e 20 bits para a quantidade de marca que está sendo removida/criada. O sistema de comunicação realiza o roteamento do pacote baseado no endereço de destino e entrega apenas a carga útil (tipo e a quantidade de marcas) para o processador do arranjo, o qual realizará a atualização das marcas no seu banco de registradores.

As dimensões inicialmente adotadas para o sistema de comunicação e de processamento se baseiam no fato de que não é muito comum encontrar modelos práticos de Redes de Petri onde a maioria dos lugares da rede seja conectada com uma quantidade elevada de transições. Normalmente, a maioria dos lugares de uma Rede de Petri é ligada, no máximo, com uma ou duas transições, exceto para lugares que representam recursos em comum e pontos de sincronismo da rede (Soto and Pereira, 2001). A utilização, por exemplo, de um banco de registradores capaz de armazenar uma grande quantidade de lugares aumentaria muito a lógica necessária para implementar os processadores do arranjo e além disso, modelos típicos de Redes de Petri poderiam sub-utilizar muito os recursos disponíveis da arquitetura.

Contudo, a arquitetura proposta foi descrita em VHDL utilizando códigos parametrizáveis. Desta forma, é possível aumentar ou diminuir a dimensão dos pacotes, a quantidade de roteadores do sistema de comunicação, a quantidade de registradores, o número de marcas e tipos de uma marca para adaptar a arquitetura proposta às necessidades do usuário.

Apesar de cada processador do arranjo possuir um banco de registradores capaz de armazenar apenas quatro tipos diferentes de marcas de entrada, o que é suficiente para mapear a maioria das transições de uma Rede de Petri convencional, a arquitetura pode ser configurada para possibilitar o mapeamento de transições que possuam mais do que quatro tipos diferentes de marcas de entrada. Para isso, faz-se uso de transições intermediárias denominadas transições de convergência, como mostrado na figura 5.

Cada transição de convergência do nível N dispara após receber uma determinada quantia de marcas. O disparo de quatro transições de convergência do nível N irá disparar uma transição de convergência de um nível imediatamente inferior (nível N-1) e assim sucessivamente até que se atinja a

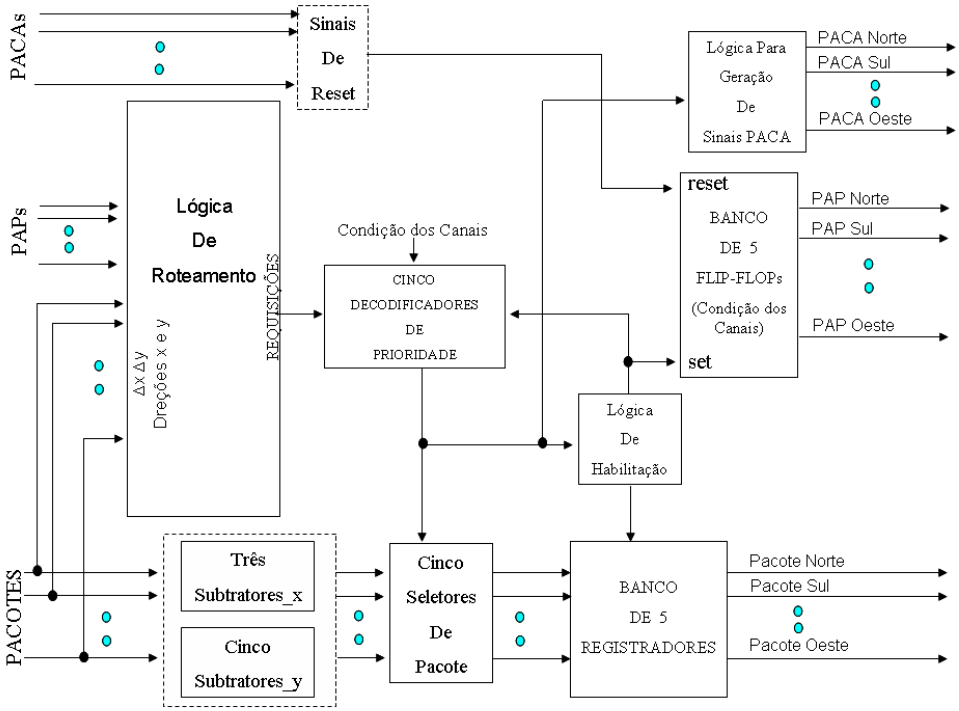

Figura 3: Diagrama de blocos do roteador

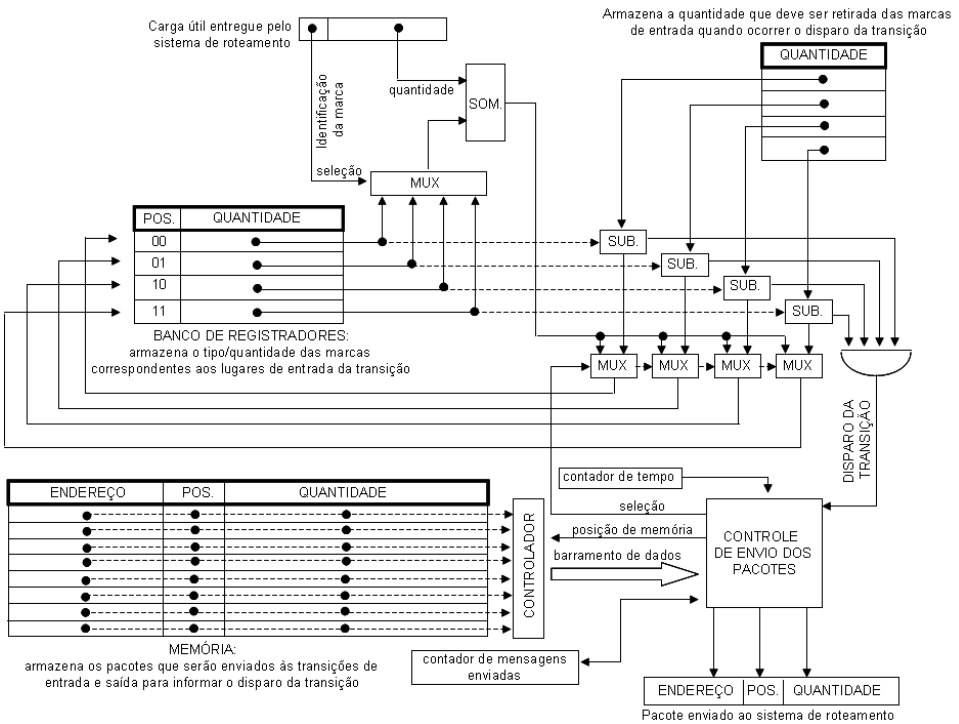

Figura 4: Diagrama de blocos do processador da arquitetura

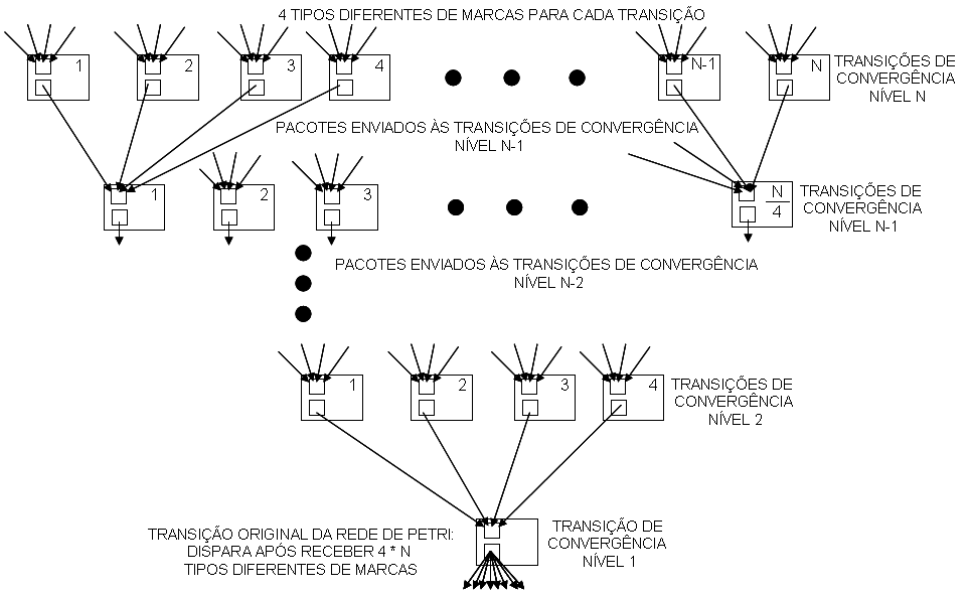

Figura 5: Transições de convergência para possibilitar o mapeamento de um número maior de tipos diferentes de marcas

última transição de convergência de nível 1. Desta forma, a transição de convergência de nível 1, na realidade, só irá disparar após o recebimento de $4 * N$ tipos diferentes de marcas nas suas quantidades pré-estabelecidas.

Originalmente, a arquitetura também teria uma restrição quanto ao número de pacotes que cada processador do arranjo poderia enviar informando o disparo da transição correspondente, visto que a memória capaz de armazenar estes pacotes possui oito palavras de 32 bits cada. Apesar de oito pacotes serem suficientes para permitir o mapeamento da maioria das transições da Rede de Petri, a arquitetura pode ser configurada para possibilitar o mapeamento de transições que precisem enviar um número maior de pacotes para o sistema de roteamento. Para isso, faz-se uso de transições intermediárias denomeadas transições escravas, como mostrado na figura 6.

A transição principal, denominada transição mestre, quando tiver para cada tipo marcas suficientes para disparar, deve enviar oito pacotes para as transições escravas de nível 1. A transição escrava do nível 1 ao receber um pacote de dados da transição mestre deverá disparar enviando outros oito pacotes para as transições escravas de nível imediatamente inferior (nível 2) e assim sucessivamente até que se atinja um nível que possibilite o envio da quantidade necessária de pacotes para as transições originais da Rede de Petri que foram mapeadas na arquitetura. Assim, a transição mestre juntamente com as transições escravas de nível 1 até N são capazes de enviar até 8^N pacotes.

No processo de configuração do modelo de Rede de Petri a ser implementado, cada processador do arranjo deve ser informado sobre o tipo de transição que está sendo mapeado. Os quatro tipos possíveis são: transições convencionais, que implementam as transições originais da Rede de Petri, transições mestres, transições de convergência e as transições escravas. Os processadores do arranjo que forem configurados como sendo transições mestres, de convergência ou escravas atuam de forma diferenciada no sistema de processamento da arquitetura para impedir que a dinâmica da Rede de Petri seja alterada na execução da rede. Monitores e semáforos foram implementados nos processadores do arranjo para fazer com que todas as transições intermediárias co-relacionadas trabalhem de forma atômica, conjunta e sincronizada, de forma a produzir um resultado único. Assim, uma transição original da Rede de Petri pode estar fisicamente implementada em diversos processadores do arranjo, mas logicamente, os processadores trabalham em conjunto. Em outras palavras, as transições mestres, de convergência ou escravas não são transições das Redes de Petri, de fato. Elas são artifícios implementados na arquitetura, e que não existem nos modelos de Redes de Petri, para permitir a implementação de uma Rede de Petri genérica. Elas podem ser consideradas módulos de *hardware* que possibilitam a expansão das transições, de forma semelhante ao que se faz para definir multiplexadores genéricos de N:1, utilizando apenas multiplexadores de 2:1.

4 RESULTADOS

Na literatura científica existem alguns projetos de roteadores, como o ParIS (Zeferino et al., 2004), OS4RS (Bartic

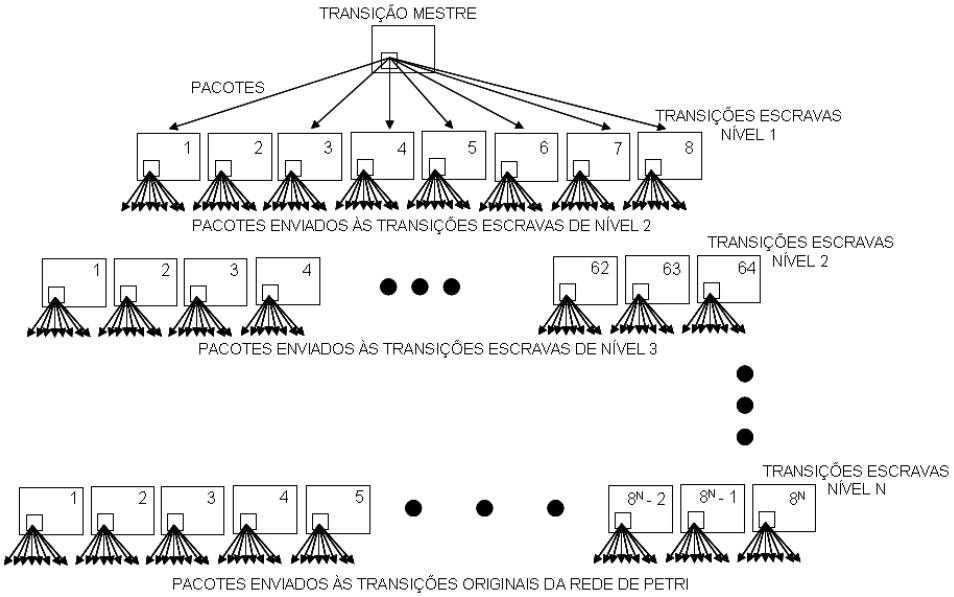

Figura 6: Transições escravas para possibilitar o envio de um número maior de pacotes informando o disparo de um transição

et al., 2003) e o Hot-Potato switch (Nilsson et al., 2003) mostrados na tabela 1. Os roteadores podem ser especificados levando em consideração os seguintes fatores: algoritmo de roteamento, tamanho do canal, número de bits para endereçamento, controle de fluxo, memorização e arbitragem.

O roteador ParIS possui o algoritmo de roteamento X-Y, um canal de comunicação de 32 bits, um endereçamento de 8 bits, um controle de fluxo baseado na técnica *Handshake*, uma memorização de pacotes na entrada e realiza uma arbitragem *Round-Robin*. A implementação deste roteador possui 698 LUTs (*Look Up Tables*) e 113 *flip-flops*, além de uma RAM utilizada para o armazenamento de pacotes no processo de contenção da rede.

O roteador OS4RS possui um algoritmo de roteamento baseado em tabelas, um canal de comunicação de dados de 16 bits, um controle de fluxo baseado na técnica *Handshake*, uma memorização de pacotes na saída e realiza uma arbitragem *Round-Robin*. A implementação deste roteador possui 17500 portas lógicas, além de uma RAM utilizada para o armazenamento de pacotes.

O roteador Hot-Potato possui um algoritmo de roteamento baseado na técnica *hot-potato*, um canal de comunicação de 127 bits, um endereçamento de 8 bits, uma memorização *Buffer-less* e realiza uma arbitragem utilizando contadores e valores de *stress* (*Hop-counter* / *Stress value*). A implementação deste roteador possui 13964 portas lógicas.

O roteador que está sendo aqui proposto possui o algoritmo de roteamento X-Y, um canal de comunicação de 32 bits, um endereçamento de 8 bits, um controle de fluxo baseado na técnica *Handshake*, uma memorização de pacotes na saída e realiza uma arbitragem *Round-Robin*. O roteador foi projetado utilizando a linguagem VHDL e mapeado no FPGA EP1S10F780C5, da família STRATIX. A implementação do roteador proposto possui 464 LUTs e 169 *flip-flops*, totalizando 2203 portas lógicas, os pacotes são armazenados em registradores internos não necessitando de uma estrutura maior de memorização como uma RAM, por exemplo. A latência do roteador é de aproximadamente 6,8ns, podendo atingir uma vazão de até 5,48GB/s.

Deve-se observar que cada um desses roteadores tem um propósito diferente, sendo destinados a aplicações específicas, o que implica em implementações e soluções de projeto distintas. Além disso, foram utilizados diferentes processos para a realização da síntese de cada roteador. Contudo, pode-se notar que a arquitetura proposta para o roteador apresenta uma quantidade significativamente pequena de portas lógicas. Isso se deve à aplicação, no caso Redes de Petri, possuir um sistema de comunicação de granularidade fina que combinada com a homogeneidade do arranjo de processadores da transição permitiu a redução no controle de fluxo e na estrutura de memorização.

No processo de prototipação e validação da arquitetura proposta foi utilizada uma rede-em-*chip* 3x3, o que permitiu a implementação de até 9 transições de uma Rede de Petri. A arquitetura foi mapeada no FPGA EP1S10F780C5, também da família STRATIX. De 10570 elementos lógicos disponíveis, usou-se 4.052 (38%), e de 427 pinos, utilizou-se 80

Tabela 1: Especificações de alguns roteadores

	ParIS	OS4RS	Hot-Potato switch	Roteador Proposto
Algoritmo de Roteamento	X–Y	Tabelas de roteamento	Hot–potato	X–Y
Tamanho do canal	32 bits	16 bits de dados	127 bits	32 bits
Endereçamento	8 bits	X	8 bits	8 bits
Controle de Fluxo	Handshake	Handshake	X	Handshake
Arbitragem	Round-Robin	Round-Robin	Hop-counter/stress value	Round-Robin
Memorização	Entrada	Saída	Buffer-less	Saída
Síntese	FPGA-Altera APEX II	FPGA-Xilinx Virtex2Pro	Synopsys lsi10K	FPGA-Altera STRATIX
Portas lógicas equivalentes	X	17500 (sem RAM)	13964	2203
LUTs / FFs	698 / 113 (sem RAM)	X	X	464 / 169

(18%). A latência de toda a arquitetura é de 15,4ns e com uma freqüência de operação de 64,58MHz a vazão pode alcançar até 17,12GB/s. De acordo com as quantidades de elementos lógicos de cada unidade de processamento e de cada roteador, pode-se mapear, neste FPGA, até 23 transições e em FPGAs maiores, como o EP1S80 com 79.000 elementos lógicos, também da família da STRATIX, pode-se mapear uma Rede de Petri com cerca de 150 transições.

No processo de prototipação e validação da arquitetura proposta, foi escolhida a menor quantidade possível de elementos de processamento e de roteadores para facilitar o processo de correção de erros de projeto, isto é, com uma pequena quantidade de elementos o tempo para identificar e localizar os erros de projeto no sistema é menor. No entanto, vale ressaltar, que a dimensão escolhida foi suficiente para comprovar e validar todas as características e funcionalidades da arquitetura proposta. Para cada modelo de Rede de Petri testado foram gerados, manualmente, em torno de 120 pacotes ou 4000 bits de configuração. A utilização de uma rede-em-*chip* maior adicionaria um tempo excessivamente longo para a geração dos vetores de teste, dificultaria o processo de identificação e localização dos erros de projeto e, ainda, não acrescentaria nenhuma funcionalidade no processo de validação da arquitetura proposta.

5 CONCLUSÃO

A arquitetura desenvolvida permite a implementação física de sistemas de controle descritos por meio de Redes de Petri coloridas de arcos constantes e T-temporizadas. A arquitetura foi implementada numa rede-em-*chip* contendo nove unidades de processamentos e nove roteadores, o que permitiu o mapeamento de modelos de Redes de Petri com até nove transições temporizadas. A arquitetura foi validada num FPGA de 10570 elementos lógicos e utilizou em torno de 38% de recursos disponíveis do FPGA. A implementação da arquitetura em FPGAs maiores com, por exemplo, 79.000 elementos lógicos, permite o mapeamento de modelos de Redes de Petri contendo até 150 transições.

No processo de validação, cada processador do arranjo pôde ser configurado para mapear transições com diferentes quantidades de arcos de entrada e saída e trabalhar com diferentes tipos de marcas. Restrições relacionadas ao tamanho da memória e ao banco de registradores foram resolvidos utilizando as transições escravas e convergentes que expandem a capacidade de mapeamento dos modelos de transições de uma Rede de Petri, sem realizar alterações na dinâmica da rede, pois os processadores do arranjo que implementam essas transições intermediárias trabalham em conjunto e em sincronismo para garantir a coerência dos dados na execução da Rede de Petri descrita.

REFERÊNCIAS

Anzai, F., Kawahara, N., Takei, T., Watanabe, T., Murakoshi, H., Kondo, T. and Dohi, Y. (1993). Hardware implementation of a multiprocessor system controlled by petri nets, *Proceedings of the IECON'93, 1993 International Conference on Industrial Electronics, Control and Instrumentation*, Vol. 1, Piscataway, NJ, IEEE, Hawaii, USA, pp. 121–126.

Bartic, T., Mignolet, J.-Y., Nollet, V., Marescaux, T., Verkest, D., Vernalde, S. and Lauwereins, R. (2003). Highly scalable network on chip for reconfigurable systems, *Proceedings of International Symposium on System-on-Chip*, pp. 79–82.

Benini, L. and Micheli, G. D. (2002). Networks on chips: A new soc paradigm, *Computer* **35**(1): 70–78.

Bundell, G. A. (1997). An fpga implementation of the petri net firing algorithm, *The 1997 Australasian Conference on Parallel and Real-Time Systems*, Springer-Verlag, Singapore, Newcastle, pp. 434–445.

Csertán, G., Majzik, I., Pataricza, A., Allmaier, S. C. and Hohl, W. (1997). Hardware accelerators for petri net

analysis, *In Proceedings of Distributed and Parallel Systems, DAPSYS98*, Institut für angewandte Informatik und Informationssysteme, Universität Wien, Budapest, Hungary, pp. 99–104.

Culler, D. E., Gupta, A. and Singh, J. P. (1997). *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Dally, W. J. and Towles, B. (2001). Route packets, not wires: on-chip interconnection networks, *DAC'01: Proceedings of the 38th conference on Design automation*, ACM Press, New York, NY, USA, pp. 684–689.

David, R. and Alla, H. (1992). *Petri Nets and Grafcet: Tools for Modelling Discrete Event Systems*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Gomes, L. (1999). *Redes de Petri e Sistemas Digitais: uma introdução*, 1.2 edn, FCT-UNL, Universidade Nova de Lisboa, Portugal.

Guerrier, P. and Greiner, A. (2000). A generic architecture for on-chip packet-switched interconnections, *DATE '00: Proceedings of the conference on Design, automation and test in Europe*, ACM Press, New York, NY, USA, pp. 250–256.

Jensen, K. (1997). *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, second edn, SPRINGER.

Kamakura, T., Shimoda, T., Dohi, Y. and Murakoshi, H. (1997). Implementation of a large petri net by a group of petri net controller, *Proceedings of the IECON'93, The 23rd International Conference on Industrial Electronics, Control and Instrumentation*, Vol. 3, Piscataway, NJ, IEEE, New Orleans, Louisiana, USA, pp. 1210–1215.

Liu, J., Zheng, L.-R. and Tenhunen, H. (2004). Interconnect intellectual property for network-on-chip (noc), *J. Syst. Archit.* **50**(2-3): 65–79.

Marsan, M. A. (1990). Stochastic petri nets: an elementary introduction, *Advances in Petri Nets 1989, covers the 9th European Workshop on Applications and Theory in Petri Nets-selected papers*, Springer-Verlag, London, UK, pp. 1–29.

Morris, J., Bundell, G. A. and Tham, S. (2000). A scalable re-configurable processor, *IEEE Proceedings of the 5th Australian Computer Architecture Conference*, IEEE Computer Society, Canberra, Australia, pp. 64–73.

Murata, T. (1989). Petri nets, *Proceedings...*, Proceedings of the IEEE, IEEE Computer Society Press, Canberra, pp. 541–580.

Nilsson, E., Millberg, M., Oberg, J. and Jantsch, A. (2003). Load distribution with the proximity congestion awareness in a network on chip, *DATE '03: Proceedings of the conference on Design, Automation and Test in Europe*, IEEE Computer Society, Washington, DC, USA, p. 11126.

Pande, P. P., Grecu, C., Ivanov, A. and Saleh, R. (2003). Design of a switch for network on chip applications, *Proceedings of ISCAS, IEEE International Symposium on Circuits and Systems*, Vol. V, ISCAS, Bangkok, Thailand, pp. 217–220.

Reisig, W. (1992). *A Primer in Petri Net Design*, Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Rokyta, P., Fengler, W. and Hummel, T. (2000). Electronic system design automation using high level petri nets, *Hardware Design and Petri Nets* pp. 193–204.

Soto, E. and Pereira, M. (2001). Implementing a petri net specification in a fpga using vhdl, *The International Workshop on Discrete-Event System Design DES-Des'01*, Oficyna Wydaw, Zielona Góra, Poland.

Strum, M., Chau, W. J. and do Vale Neto, J. V. (1999). Síntese de alto nível: Plana e hierárquica, *in* A. G. Rozo (ed.), *Sistemas Digitales – Elementos para un Diseño a Alto Nivel*, Uniandes, Santafé de Bogotá, chapter V, pp. 181–279.

Thiagarajan, P. S. (1987). Elementary net systems, *in* W. Brauer, W. Reisig and G. Rozenberg (eds), *Advances in Petri Nets 1986–PartI: Central Models and Their Properties*, Vol. 254 of *Lecture Notes in Computer Science*, Springer Verlag, Berlin, pp. 26–59.

Wang, J. (1998). *Timed Petri Nets: Theory and Application*, Kluwer Academic, Boston.

Zeferino, C. A. (2003). Introdução às redes-em-chip, *Technical report*, Universidade do Vale do Itajaí – CTTMar/CSED, Itajaí, Santa Catarina.

Zeferino, C. A., Kreutz, M. E., Carro, L. and Susin, A. A. (2002). A study on communication issues for systems-on-chip, *SBCCI '02: Proceedings of the 15th symposium on Integrated circuits and systems design*, IEEE Computer Society, Washington, DC, USA, p. 121.

Zeferino, C. A., Santo, F. G. M. E. and Susin, A. A. (2004). Paris: a parameterizable interconnect switch for networks-on-chip, *SBCCI '04: Proceedings of the 17th symposium on Integrated circuits and system design*, ACM Press, New York, NY, USA, pp. 204–209.