Processadores Assíncronos

Cássia Nunes
Helen Peters
Iúri Chaer

Arquitetura de Computadores
Programa de Residência em Tecnologia Google
{cassiasn, helenpeters, iuri.chaer}@gmail.com

Resumo

Circuitos assíncronos são sistemas cujo funcionamento é orientado a eventos. Nestes sistemas, diferentes áreas de um chip podem funcionar em velocidades diversas. Os dados são enviados e recebidos conforme a necessidade.

A maioria dos sistemas hoje em dia é síncrona, um modo de funcionamento que deve a sua popularidade à simplicidade de projeto e implementação. No entanto, ele é extremamente ineficiente quando comparado ao seu contraparte.

1 Introdução

Grande parte dos projetos lógicos se baseia em um modelo que considera o tempo uma grandeza discreta. Esta modelagem simplifica a lógica do projeto. No entanto, à medida que o tamanho e a complexidade dos sistemas síncronos aumentam, a freqüência do *clock* começa a causar problemas. Dentre eles, podemos citar *clock skews* (em um sistema complexo, o sinal de *clock* alcança diferentes pontos de um circuito em instantes de tempo distintos devido aos atrasos resultantes dos diversos comprimentos de fio), distribuição do *clock* (é necessária uma quantidade substancial de área de silício para distribuir o *clock* pelo sistema) e consumo de energia.

Sistemas assíncronos são uma alternativa para contornar estas limitações. Esses sistemas não apresentam um sinal de *clock* que sincroniza todas as operações realizadas pelo sistema. Tipicamente, sistemas assíncronos são compostos por unidades que se comunicam entre si através de protocolos *handshake*. A ausência de *clock* elimina o problema de *clock skew*, da distribuição do *clock*, e, em geral, a utilização de energia é reduzida, já que porções do sistema não utilizadas podem ser mantidas em modo de baixo consumo.

Sistemas assíncronos permitem que cada bloco funcional possa ser otimizado sem a preocupação de sincronismo com um *clock* global. Por causa disso, a performance do sistema reflete a performance média dos seus componentes individuais, diferente dos sistemas síncronos, nos quais a performance é limitada pelo componente de pior performance do conjunto. Por outro lado, no caso de sistemas assíncronos a complexidade do projeto aumenta significativamente, e por isso pode ser necessário utilizar mais hardware para implementar o sistema. Por estes motivos, existem poucos processadores assíncronos implementados [2, 8, 4, 3, 9, 5].

2 Conceitos Básicos

O sinal de *clock* é gerado por um cristal oscilador que pauta todo o funcionamento de um sistema síncrono. Ele controla o tempo de cada um dos eventos em cada um dos circuitos de todos os chips que compõem o sistema. O exemplo mais pervasivo que temos no dia a dia moderno são os microprocessadores.

Cada vez que um impulso elétrico percorre um circuito, parte da sua energia se perde na forma de calor. Por causa de propriedades como a indutância e a capacitâcia, quanto maior a freqüência de operação desse circuito, mais energia ele consome e mais calor é gerado. Considerando que o sinal de *clock* está sempre ativo, mesmo que um microprocessadores não tenha nenhum serviço util para realizar, os impulsos elétricos continuam sendo enviados - o processador consome energia e se aquece desnecessariamente.

Além disso, o sinal de *clock* limita o desempenho de sistemas síncronos, já que a necessidade de manter to-

dos os componente sincronizados faz com que as partes mais rápidas do conjunto tenham de esperar as mais lentas terminarem seus ciclos de processamento.

Esses problemas tendem a se acentuar conforme os processadores atingem freqüências de operação cada vez mais altas.

Uma alternativa aos processadores síncronos são os processadores assíncronos, ou sem *clock*, baseados na operação por eventos, fazendo com que diferentes bits de um chip operem com velocidades diversas, enviando e recebendo dados quando for apropriado, como apresentado na figura 1.

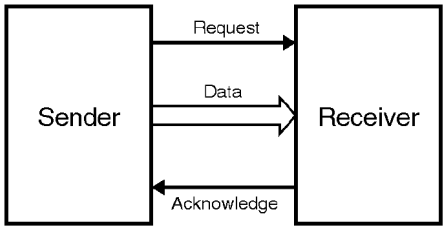

Figura 1. Operação baseada em eventos.

Existem alguns conceitos fundamentais para o entendimento de circuitos assíncronos, a saber: o modelo de temporização, o modo de operação e a sinalização, [7].

2.1 Modelo de Temporização

Circuitos assíncronos são classificados de acordo como o seu comportamento em relação aos atrasos do circuito. Se um circuito funciona corretamente sem considerar os atrasos das portas lógicas e os atrasos no fio, este circuito é conhecido como *delay-insensitive* (insensível a atrasos). Uma forma mais restrita deste tipo de circuito é conhecida como *speed-independent* (independente de velocidade). Esta forma permite atrasos variados em portas lógicas, mas assume atraso igual a zero nas interconexões (i.e. todos os fios de interconexão são equi-potenciais). Finalmente, se um circuito funciona apenas quando os atrasos estão abaixo de algum valor pré-definido, este circuito é conhecido como *bounded-delay* (com limite de atraso).

2.2 Modo de Operação

Circuitos assíncronos podem operar em dois modos. O primeiro modo é o fundamental. Ele assume que nenhuma modificação pode ser realizada na entrada até que todas as saídas tenham sido estabelecidas. O segundo, *input/output* (entrada/saída), permite mudanças nas entradas enquanto o circuito assíncrono ainda está gerando as saídas.

2.3 Sinalização

A comunicação entre dois elementos num sistema assíncrono pode possuir duas fases (*two-phase*) ou quatro fases (*four-phase*) de operação, e um bit de informação pode ser transportado em um único fio ou em um par de fios (conhecido como *dual-rail encoding*).

Two-phase Numa comunicação em duas fases, a informação é transmitida por uma única transição ou mudança no nível de tensão do fio. A figura 2b mostra um exemplo de comunicação em duas fases. O remetente inicia a comunicação, realizando uma transição no fio de requisição (*request wire*); o receptor responde realizando uma transição no fio de reconhecimento (*acknowledge wire*), completando as duas fases de comunicação. O nível de tensão dos fios não contém nenhuma informação, apenas as transições são importantes. Transições de subida e de descida são equivalentes. Não existe nenhum estágio intermediário de recuperação – se a primeira requisição resultou numa transição de '0' para '1', a nova comunicação começa com a transição '1' para '0'.

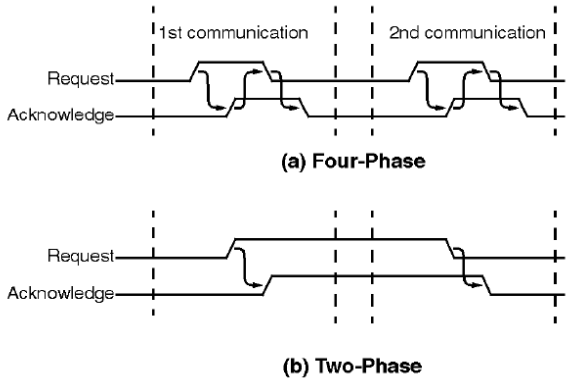

Figura 2. Esquema de sinalização em duas e quatro fases.

Four-phase Em comunicações em quatro fases, duas fases são de comunicação ativa, enquanto as outras duas têm o propósito de voltar ao estado inicial pré-definido. A figura 2a apresenta um exemplo de comunicação em quatro fases no qual todos os níveis são iniciados com um estado lógico igual a '0'. A comunicação é iniciada pelo remetente: ele altera o estado

do fio de requisição de '0' para '1' para indicar que está ativo. O receptor responde mudando o estado do fio de reconhecimento para '1' também. O receptor observa a mudança, que indica que a comunicação foi realizada com sucesso, e então modifica o estado do fio de requisição de '1' para '0' para indicar que ele não está mais ativo. O receptor completa a quarta fase da operação quando modifica o estado do fio de reconhecimento de volta para '0', para indicar que ele também se tornou inativo. Ao final das quatro fases de uma única comunicação, os níveis de tensão nos fios terão retornado ao valor inicial (o que não ocorre com duas fases de comunicação).

Single-rail encoding Um circuito *single-rail* codifica a informação de maneira convencional. É necessário um fio para cada bit de informação. Se a informação é um dado, então uma codificação típica usaria o nível alto de tensão (Vdd) como correspondente ao valor '1' e um nível baixo de tensão (Vss) para representar o valor lógico '0'.

Dual-rail encoding Um circuito *dual-rail* requer dois fios para codificar cada bit de informação. Um fio representa o valor lógico '0' e o outro representa o valor lógico '1'. Em qualquer comunicação, um evento ocorre em um dos dois fios. Não existe nenhum evento que possa ocorrer nos dois fios ao mesmo tempo durante uma única comunicação.

Existem várias combinações de duas/quatro fases e *single/dual rail* que podem ser utilizados. O uso de quatro fases com *dual-rail* é popular em projetos de circuitos *delay-insensitive*.

3 Revisão Bibliográfica

Vários estudos sobre processadores assíncronos foram realizados nas últimas décadas. A seguir alguns trabalhos são apresentados.

Em 1989, Martin, Burns e Lee [5] apresentaram o primeiro projeto completo de um microprocessador assíncrono. O controle e o *data path* foram projetados separadamente, e em seguida combinados de maneira mecânica. O *pipeline* foi encarado como um problema de programação concorrente. As fases do *pipeline* foram: *fetch*, *decode* e *execute*. Começando com um programa seqüencial para o processador, a concorrência foi introduzida através de séries de transformações de execução da instrução, mas elas não eram mecânicas nem únicas. O projetista decidia como decompor um programa em vários programas concorrentes. Apesar da simplicidade da arquitetura projetada e dos resultados modestos, este primeiro projeto foi muito importante por introduzir conceitos de operação interessantes, utilizar idéias de processamento tipo *pipeline*, e pelo grupo de trabalho reduzido envolvido.

Paver investiga, em sua tese de doutorado [7], a possibilidade da construção comercial de circuitos complexos utilizando projeto assíncrono. Projetos assíncronos oferecem soluções mais econômicas para vários problemas que ocorrem em projetos síncronos. As principais áreas beneficiadas são a sincronização global, a performance, e o consumo de energia. Os benefícios estão diretamente ligados à ausência de um *clock* global que coordene o funcionamento de todas as unidades do circuito. O processador AMULET1 foi projetado e implementado utilizando técnicas de projeto assíncronas, e foi o primeiro a ser compatível em código com outro já existente, o microprocessador ARM6. O AMULET1 demonstrou que é possível implementar uma arquitetura RISC comercial complexa usando circuitos assíncronos. O projeto resultante apresenta soluções para muitos problemas associados a processadores RISC modernos, como suporte para exceções exatas e compatibilidade entre conjuntos de instruções invertidas e operações em *pipeline*. O processador assíncrono AMULET1 não apresentou nenhuma vantagem significante quando comparado com o processador síncrono ARM6. Contudo, o AMULET1 foi o primeiro processador assíncrono projetado utilizando a técnica de *Micropipeline*, enquanto o seu contraparte ARM6 faz parte da quarta geração de processadores síncronos e era um líder mundial em eficiência de energia e área de *die*. O AMULET1 é muito flexível e ajusta automaticamente sua performance a mudanças de temperatura e tensão, aumentando o consumo apenas quando existe algum trabalho útil a ser realizado. Existe um espaço considerável para melhoria da eficiência do uso de energia e aumento da performance. Contudo, a penalidade de área para o controle lógico assíncrono é um obstáculo difícil de superar.

O trabalho proposto por Arvind e Mullins [1] apresenta uma arquitetura superescalar assíncrona baseada na técnica de composição de instruções (*instruction compounding*). Esta técnica define grupos de instruções dependentes, selecionando grupos (*compounds*) cujo grafo resultante seja um grafo dirigido acíclico (DAG). Dentro destes grupos resultados podem ser adiantados entre instruções sucessivas, permitindo um adiantamento dinâmico de dados baseados em informações locais, enquanto mantém as vantagens de se utilizar uma arquitetura assíncrona. Foram realizados experimentos que comparam a arquitetura proposta (utilizando *compounds* dinâmicos, definidos em tempo de

execução, e estáticos, definidos em tempo de compilação) com uma arquitetura superescalar síncrona. Os resultados mostram que a performance pode ser melhorada no caso assíncrono por reduzir tempo gasto com sincronização em tempo de execução e por explorar paralelismo de instruções de forma mais eficiente.

Em [6] os autores propõem um processador assíncrono para redes de sensores sem fio (RSSF). Nestas redes a energia é a um fator crítico, já que os elementos da rede apresentam uma fonte de alimentação limitada cuja recarga, em geral, é difícil ou mesmo impraticável. As tarefas de sensoriamento, comunicação e processamento de dados são as responsáveis pelo consumo de energia nos nós, sendo que tipicamente a energia consumida com tarefas de comunicação (transmissão e recepção de dados) é maior do que a consumida pelas outras duas tarefas. Grande parte dos trabalhos propostos para RSSFs tentam amenizar o consumo de energia através da diminuição do tráfego de informações na rede através de um aumento da computação realizada localmente em cada nó. Por este motivo, diminuir a quantidade de energia consumida com processamento passa a ser uma tarefa cada vez mais importante. Neste trabalho os autores alcançam uma redução de consumo de energia significativa utilizando um processador assíncrono comparado com o caso de um processador síncrono. O processador assíncrono projetado minimiza o consumo de energia por realizar gerenciamento de energia através de mudança dinâmica de tensão e freqüência, e por trabalhar com uma tensão muito próxima do limite do processador.

4 Sistemas Síncronos x Assíncronos

A seguir são discutidos três aspectos de projeto em sistemas síncronos e assíncronos: a organização do pipeline, *instruction issue* e o tratamento de instruções de desvio[10].

4.1 Organização de pipeline

Organizações em pipeline tipicamente incluem um ou mais estágios para realizar o *fetch*, *decode*, execução, acesso a memória e *write back*. Há instruções em um pipeline não passam por todos seus estágios, como por exemplo instruções que não fazem acesso a memória e instruções que realizam desvios. No caso de desvios, assim que é determinado se o desvio é tomado ou não, pode ser necessário parar o pipeline por um estágio. Pipelines em arquiteturas síncronas eliminam essas *bolhas* adicionando hardware para realizar adiantamento de dados (*data forwarding*) a fim de manter a taxa de processamento. O pipeline em arquiteturas assíncronas, por outro lado, depende de *bolhas* para manter essa taxa, já que o estágio estará desocupado.

A fim de melhorar a performance em pipelines, em geral, são criados pipelines mais longos, mas com estágios mais simples. Isto traz impactos em sistemas síncronos, já que o aumento do número de estágios aumenta a carga capacitiva do *clock* global, aumentando também o *clock skew*. Além disso, estágios pequenos permitem que o ciclo de *clock* seja mais rápido o que também tem impacto no *clock skew*. Arquiteturas assíncronas, por outro lado, tiram vantagem da performance alcançada pelo aumento do número de estágios do pipeline sem os problemas relativos a *clock skew*.

4.2 Instruction issue

Em geral, as instruções são processadas em uma ordem determinada: *fetch*, acesso aos operandos, execução, acesso à memória e *write back*.

Uma instrução que necessita de adiantamento de dados (*data forwarding*) pode ler um operando antes que o resultado seja escrito, causando riscos (*hazards*) to tipo *read-after-write* (leitura seguindo escrita, conhecidos como RAW). Além disso, se forem permitidas que instruções sejam completadas fora de ordem riscos do tipo *write-after-read* (escrita seguindo leitura, ou WAR) e *write-after-write* (escrita seguindo escrita, ou WAW) também são possíveis. Caso as instruções sempre sejam completadas em ordem, riscos de dados do tipo WAR e WAW são evitados, já que os operandos são sempre lidos ou escritos antes que as instruções que a sucedem executem escrita de dados.

Processadores síncronos com instruções completadas em ordem evitam riscos do tipo RAW através da inserção de instruções NOP (*no operations* – operações improdutivas) ou parando (*stall*) a execução da instrução até que o risco tenha passado.

Processadores assíncronos, por outro lado, eliminam riscos do tipo RAW atribuindo um *flag* no registrador de destino de uma instrução assim que ela é decodificada. As instruções seguintes que dependem deste resultado serão interrompidas durante o acesso ao operando. Isto ocorre devido a natureza dos protocolos de comunicação de sistemas assíncronos, já que é necessário receber uma confirmação (*acknowledgement*) antes que a instrução possa continuar.

4.3 Tratamento de desvios

Instruções de desvio podem alterar o fluxo do programa. Os processadores, em geral, tentam prever se o desvio será ou não tomado. Um erro nesta decisão implica em uma penalidade que corresponde ao número

de ciclos necessários para que o processador se recupere da previsão errada do desvio. As alternativas, nesse caso, são: travar o pipeline até que o endereço do desvio seja conhecido, assumir que o desvio será tomado (*predict taken*), assumir que o desvio não será tomado (*predict not-taken*), escalonar código para uso de *delay-slots*, ou ainda, utilizar algum outro algoritmo de previsão de desvios.

De forma geral, processadores síncronos e assíncronos tratam desvios de maneira similar e com uma complexidade comparável utilizando alguma das técnicas citadas.

Em processadores síncronos, o atraso que uma instrução de desvio gera é fixo e conhecido. O processador apenas invalida os estágios de pipeline que contêm instruções decorrentes de um desvio cuja previsão foi errada. Em sistemas assíncronos, os processadores não tem conhecimento do número de instruções inseridas no pipeline após uma previsão errada, e por isso não é possível distingüir entre instruções válidas e inválidas. Uma das técnicas utilizadas para solucionar este problema é incluir um dado em cada instrução representando uma cor. Todas as instruções no pipeline têm a mesma cor até que uma instrução de *branch* seja encontrada. A partir deste momento o bit de cor é trocado e as instruções seguintes têm uma cor diferente. Se um branch é previsto de forma errada, todas instruções da cor errada são removidas do pipeline.

4.4 Vantagens e Desvantagens de Sistemas Assíncronos

Dentre as vantagens de se utilizar sistemas computacionais assíncronos podemos citar:

Performance: Por se tratar de um sistema baseado em eventos, não há *clock* e, portanto, o que limita a velocidade é o tempo necessário para propagar a informação por seus circuitos. Os componentes internos do processador trabalham isoladamente à sua capacidade máxima, sem serem retardados por componentes mais lentos, como mostrado na figura 3.

Menor consumo de energia: O ambiente baseado em transições por eventos não apresenta a necessidade de distribuição do sinal do *clock*, o que é extremamente complexo nos chips atuais; nem apresenta *jitter* (variações indesejáveis do *clock*). Disto advém que o consumo de energia numa arquitetura com *clock* é maior, porque mesmo as partes inativas dos circuitos têm que responder a cada ciclo do mesmo, o que não ocorre em sistemas assíncronos. Além disso, processadores assíncronos permitem economia de energia pois o chip só funciona quando há dados para processar.

Menor emissão eletromagnética: Os chips com *clock* produzem emissões eletromagnéticas em freqüências múltiplas do oscilador (harmônicas), que podem causar rádio-interferência.

Robustez: Processadores assíncronos só ativam circuitos necessários para tratar dados, deixando prontos para atender outras demandas os circuitos que estão livres. Além disso, chips assíncronos dissipam menos energia e apresentam picos de tensão menores e com menos freqüência. Por estes motivos, estes chips têm menor probabilidade de falhar por problemas relacionados ao aquecimento e são mais robustos.

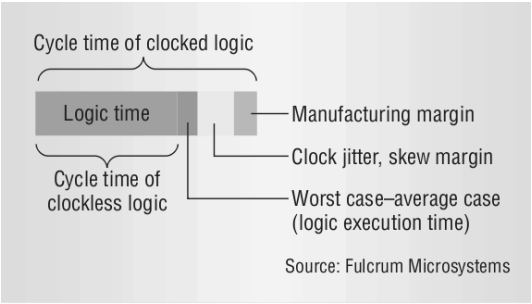

Figura 3. Tempo de ciclo para lógica assíncrona e síncrona

Apesar de apresentar vantagens no que diz respeito à performance, consumo de energia e emissão eletromagnética, a complexidade de projetar sistemas assincronos é muito maior que a de projetar sistemas síncronos.

Dentre as desvantagens de se utilizar sistemas computacionais assíncronos podemos citar:

Complexidade do projeto: A ausência do sinal de *clock* remove a restrição da sincronia dos elementos do chip, aumentando a complexidade das decisões que o projetista precisa tomar. Passa a ser necessário, por exemplo, planejar a forma como os dados trafegarão dentro do chip, adicionar *buffers* para que dados não sejam perdidos pela diferença de velocidade entre os componentes, e isso torna o projeto do processador e testes do sistema muito mais complexos.

Atrasos desiguais: Em sistemas síncronos todo o dispositivo aguarda um tempo suficientemente longo, depois das mudanças externas na entrada,

para que todos os flip-flops alcancem um estado estável antes de aplicar um novo pulso de *clock*. No entanto, para sistemas assíncronos, são necessárias técnicas de projeto mais complexas, a fim de eliminar problemas resultantes de atrasos desiguais através dos vários caminhos do sistema.

Ferramentas de desenvolvimento Um grande problema enfrentado pelos projetistas de sistemas assíncronos é a carência de ferramentas de desenvolvimento, que existem em abundância no mundo dos chips síncronos, sendo necessário desenvolver ferramentas próprias ou adaptar ferramentas desenvolvidas para sistemas síncronos. Ambas tarefas tem um custo monetário alto e necessitam de um tempo de desenvolvimento significativo.

5 Conclusão

Atualmente o esforço de pesquisa e indústria com relação a sistemas assíncronos trata de arquiteturas híbridas, nas quais sistemas síncronos são constituídos por ilhas de componentes assíncronos sincronizados por um *clock* principal que troca seu sinal quando dados passam entre seções. Esta abordagem adiciona os benefícios do projeto assíncrono aos chips síncronos.

Uma outra vertente se move em direção a projetos assíncronos com componentes localmente síncronos. Ilhas síncronas operariam com sinais de *clock* independentes, e utilizariam protocolos *handshake* para a comunicação através de buffers assíncronos. A principal motivação para esta estratégia é o fato de que distribuir o sinal do *clock* por todo processador está se tornando cada vez mais difícil, e por isso, realizar essa distribuição em componentes menores que se comunicam de forma assíncrona traz benefícios no projeto e desenvolvimento de circuitos.

Sistemas sem *clock* também surgem como campo promissor em sistemas embutidos, onde chips assíncronos teriam a vantagem de consumir menos energia e, graças a isso, permitir o desenvolvimento de aparelhos menores e mais leves, já que seria possível diminuir o tamanho da bateria mantendo a mesma autonomia.

Dado que chips assíncronos não têm *clock* e cada circuito só é ligado quando utilizado, processadores assíncronos utilizam menos energia que processadores síncronos. Em particular, chips assíncronos economizam energia em aplicações de áudio, vídeo e aplicações de fluxo intensivo de dados. Em sistemas síncronos, aplicações deste tipo desperdiçam ciclos de processamento quando não há áudio, ou os quadros do vídeo mudam pouco, já que é necessária pouca lógica para correção de erro. Durante este tempo inativo, processadores assíncronos economizam energia.

Dentre os desafios em aberto das arquiteturas assíncronas está a integração entre soluções síncronas e assíncronas. A interface entre arquiteturas síncronas e assíncronas pode causar problemas particularmente em sistemas de memória e barramentos, já que em processadores assíncronos as instruções são completadas fora da marca do *clock*. Componentes com *clock* necessitam que dados válidos estejam estáveis a cada borda do *clock*, diferente de componentes assíncronos. Isto torna necessária a implementação de circuitos para alinhar informações assíncronas e síncronas com o *clock* do sistema síncrono.

Referências

[1] D. K. Arvind and R. D. Mullins. A fully asynchronous superscalar architecture. In *IEEE PACT*, pages 17–22, 1999.

[2] E. Brunvand. The nsr processor. *Proceeding of the Twenty-Sixth Hawaii International Conference on System Sciences*, January 1993.

[3] K.-R. Cho, K. Okura, and K. Asada. Design of a 32-bit fully asynchronous microprocessor (FAM). *Proceedings of the 35th Midwest Symposium on Circuits and Systems, IEEE Press*, August 1992.

[4] M. Dean. Strip: A self-timed risc processor. Technical Report CSL-TR-92-543, Stanford University, Stanford, California, 1992.

[5] A. J. Martin, S. M. Burns, T. K. Lee, D. Borkovic, and P. J. Hazewindus. The design of an asynchronous microprocessor. *SIGARCH Comput. Archit. News*, 17(4):99–110, 1989.

[6] L. Necchi, L. Lavagno, D. Pandini, and L. Vanzago. An ultra-low energy asynchronous processor for wireless sensor setworks. *Proceedings of the 12th IEEE International Symposium on Asynchronous Circuits and Systems*, March 2006.

[7] N. C. Paver. *The Design and Implementation of an Asynchronous Microprocessor*. PhD thesis, Department os Computer Science, University of Manchester, Manchester, UK, 1994.

[8] R. Sproull, I. Sutherland, and C. Molnar. Counterflow pipeline processor architecture. Technical Report SMLI TR-94-25, Sun Microsystems Laboratories, Mountain View, California, 1994.

[9] J. Tierno, A. Martin, D. Borkovic, and T. K. Lee. A 100-mips gaas asynchronous microprocessor. *IEEE Design and Test of Computers*, 11(2):43–49, 1997.

[10] T. Werner and V. Akella. Asynchronous processor survey. *Computer*, 30(11):67–76, November 1997.