

# Conteúdo

<b>1</b>	<b>Introdução .....</b>	<b>1</b>
1.1	Computação Reconfigurável . . . . .	4
1.2	Motivação . . . . .	6
1.3	Propostas de Aplicações . . . . .	6
1.3.1	Redes Neurais . . . . .	6
1.3.2	Controle Adaptativo . . . . .	8
1.3.3	Computação Genérica . . . . .	9
1.3.4	Outros . . . . .	10
1.4	Objetivos . . . . .	10
1.4.1	Objetivos Gerais . . . . .	10
1.4.2	Objetivos Específicos . . . . .	10
1.5	Metodologia . . . . .	10
1.6	Material e Ferramentas . . . . .	10

# Capítulo 1

## Introdução

*Este capítulo contextualiza o tema do trabalho, apresentando a motivação histórica seu desenvolvimento, uma motivação tecnológica, seus objetivos e metodologia.*

O mundo atual é controlado quase que completamente por sistemas digitais. As informações obtidas pelos sensores são digitalizadas antes de serem tratadas. Tal processo de digitalização é importante, visto que elimina os ruídos intrínsecos ao processamento analógico (??).

O primeiro computador de computação genérica surgiu por volta da década de 40. Sua invenção iniciou a terceira revolução industrial, conhecida como revolução da informação ou revolução técnico-científico-informacional (??). Os computadores dessa época liam e executavam instruções de forma linear, em um modelo conhecido como sequencial ou temporal.

Nos anos que se seguiram, a substituição das válvulas por transistores de silício ajudaram a reduzir o tamanho dos computadores de metros a centímetros quadrados. Tal mudança permitiu um aumento na popularidade destes dispositivos para o uso pessoal, efeito que impulsionou a indústria de produção de processadores (??). As empresas da época começaram então a guerra de miniaturizações de transistores, marcada pelo célebre artigo de Gordon E. Moore, cofundador da Intel, que dizia que o número de transistores dentro de um processador duplicaria aproximadamente a cada 2 anos (??). A partir de 1970, a lei foi adaptada para a duplicação a cada 18 meses (??). A figura 1.1 apresenta uma visualização da lei de Moore nos anos que se seguiram.

Com a integração de mais componentes dentro do processador, conjuntos de instruções cada vez mais complexas foram desenvolvidas. Estas instruções surgiram para acelerar a computação de funções de níveis mais altos. A integração também reduziu a potência dissipada por transistor, permitindo que as frequências de operação dos computadores fosse aumentada (??).

Com o aumento da complexidade das instruções, passou-se a adotar duas nomenclaturas diferentes para processadores: *Reduced Instruction Set Computer* (RISC) e *Complex Instruction Set Computer* (CISC) (??). A arquitetura RISC possui um conjunto pequeno e muito otimizado de funções, comandos exclusivos para acesso a memória (arquitetura *load/store*) e uma média de uma instrução completada por ciclo, quando desconsidera-se as instruções de acesso a memória. A arquitetura CISC possui várias funções para tarefas

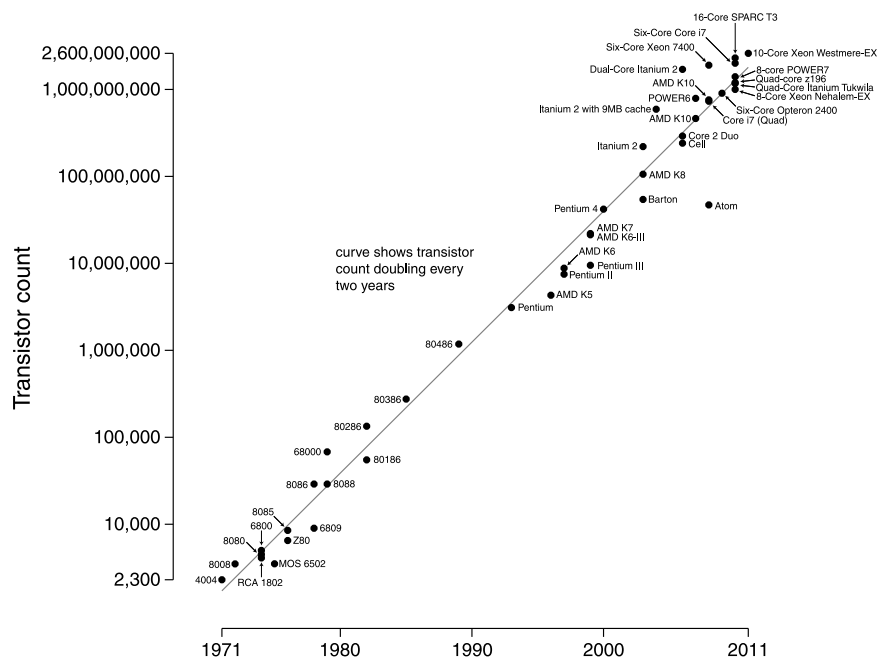


Figura 1.1: Visualização da Lei de Moore. Eixos em escala logarítmica. Extraído de (??).

mais específicas, que por vezes demandam vários ciclos de relógio, e funções que realizam operações com informações lendo e/ou salvando direto na/para a memória. A arquitetura RISC, que possui em seu portfólio dispositivos como ARM, IBM PowerPC, Sun SPARC e MIPS, dentre outros, é muito mais utilizada nos dias de hoje. A figura 1.2 mostra um pouco da história desta arquitetura. Até empresas como a Intel, que ficaram populares com seus processadores CISC, tem se curvado a arquitetura RISC devido a seu uso mais eficiente de potência. Eles vem utilizando uma arquitetura conhecida como núcleo de RISC (*RISC core*), onde as instruções são recebidas em formato CISC e decodificadas para uma arquitetura interna RISC.

Por volta dos anos 2000, a potência dissipada em cada transistor, proporcional a frequência de operação, havia atingido o limite suportado pelo microprocessador. Por causa disso, o crescimento desenfreado da frequência teve que ser repensado. Começou-se então o desenvolvimento de microprocessadores *multicore*, que aumentam a vazão de instruções (*throughput*) sem modificar o tempo de resposta, que corresponde ao tempo de processamento médio de uma instrução. Em meados de 2006, todas as grandes companhias já possuíam produtos com esta arquitetura (??).

Os microprocessadores com vários núcleos (*multicore*) abriram espaço para a chegada de processadores com muitos núcleos (*manycore*). Estes microprocessadores são projetados para placas gráficas e, apesar de possuírem centenas de núcleos, estes núcleos são simplificados (??). Em geral, eles são capazes de realizar apenas algumas poucas operações, mas abrem caminho para paradigmas de programação que transformem a computação concorrente em computação paralela (??).

Mesmo trabalhando com um ou vários núcleos de processamento, o modelo de computação atual ainda é dito temporal ou sequencial uma vez que blocos de instruções são executados em seu devido instante de tempo de forma sequencial, conceito destacado pela atomicidade estudada em programação paralela (??).

Do ponto de vista da programação, os primeiros computadores apresentavam programas que não podiam ser alterados. Parte desta limitação era justificada pela programação utilizando-se cartões, mas nas

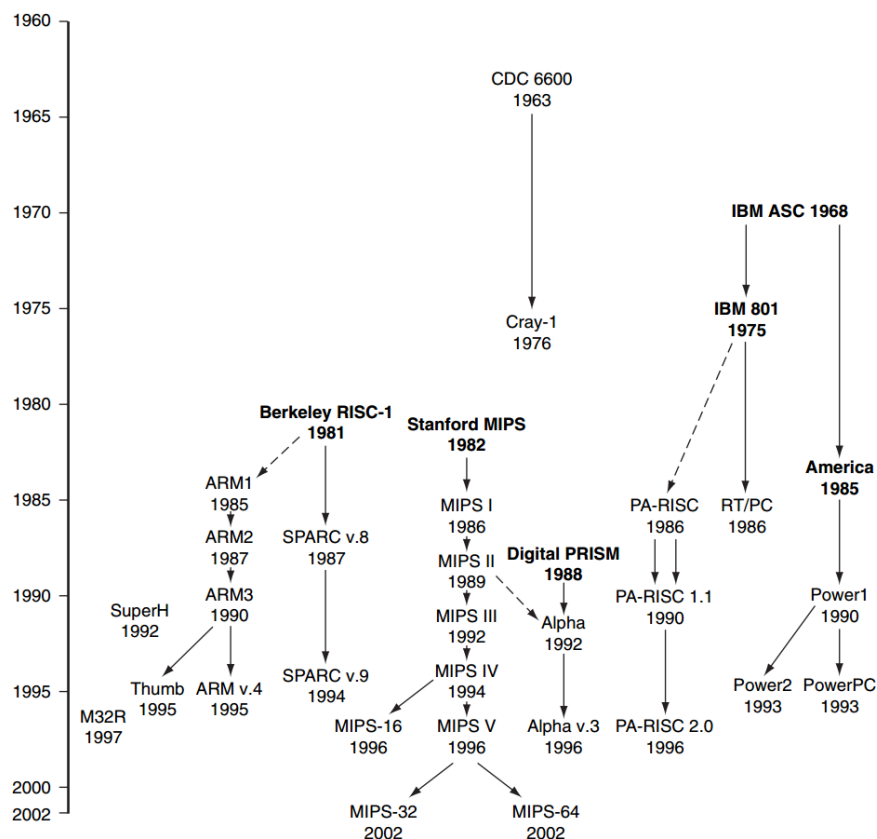


Figura 1.2: Linha do tempo das arquiteturas RISC, extraído de (??). Em negrito estão as iniciativas de pesquisa, em contraste às comerciais.

primeiras gerações de computadores com memórias eletrônicas o mesmo sistema foi utilizado.

A arquitetura de von Neumann, utilizada na primeira geração de computadores eletrônicos, era constituída de uma única unidade de memória, uma unidade de processamento e um canal de comunicação. Esta arquitetura possui tanto uma vantagem tremenda, a capacidade de modificação de programas em tempo de execução, quanto uma falha crucial, conhecida por gargalo de von Neumann. A vantagem aparece uma vez que, como não há distinção entre memória de programa e dados, uma instrução pode sobrescrever um endereço de memória marcado como programa. O problema diz respeito às restrições impostas pelo canal de comunicação, que permitia que apenas uma palavra, seja de programa ou de dados, fosse mandada para a unidade de processamento e de volta (??). Este problema se agrava a medida que o processador fica mais rápido que a memória, uma vez que o tempo de espera, em ciclos de relógios, para a obtenção da informação aumenta. Para solucionar o problema do gargalo de von Neumann, a arquitetura Harvard foi proposta.

A arquitetura Harvard original propunha que a memória de programa e a memória de dados fossem fisicamente separadas e possuíssem cada uma seu próprio canal de comunicação com o processador (??). Essa modificação acelera a execução de certos programas, visto que programa e dados podem ser carregados das suas respectivas memórias simultaneamente. Uma pequena alteração na arquitetura Harvard, conhecida de arquitetura Harvard modificada, permitia que mais de um canal de comunicação ligasse a uma memória tanto de programa quanto de dados (??). Essas informações eram divididas em memórias temporárias (*cache*) específicas para o programa e para dados, formando assim uma arquitetura Harvard

original. Essa modificação combina os benefícios da arquitetura de von Neumann, ou seja, a modificação de programas em tempo de execução, e da arquitetura Harvard original, ou seja, o tempo de acesso reduzido.

Atualmente, nossos modernos computadores multiprocessados utilizam a arquitetura Harvard modificada com diversos níveis de memória *cache* (??), sejam eles dedicados ou compartilhados entre os vários processadores. A sua capacidade de processamento atinge níveis extraordinários, ultrapassando 20 GFlops em computadores comuns (??) e 54 PFlops em supercomputadores (??). Apesar disso, a arquitetura Harvard original ainda é muito usada em microcontroladores e processadores digitais de sinal (*Digital Signal Processors* ou DSPs).

## 1.1 Computação Reconfigurável

A computação reconfigurável foi proposta por volta de 1960 por Gerald Estrin para resolver problemas que não podiam ser resolvidos pela computação da época (??). Estrin propôs um microprocessador composto de uma parte fixa e uma parte variável, onde a parte variável seria usada para programar funcionamentos específicos para serem usados em determinados períodos de tempo. Por volta da década de 1990, porém, o primeiro microprocessador híbrido comercial foi desenvolvido (??), trazendo esta tecnologia à tona.

A tecnologia inventada por Estrin, também conhecida como estrutura *Fixed Plus Variable* (F+V), cuja representação está mostrada na figura 1.3, trouxe à tona um novo paradigma de processamento de dados (??). O motivo para tal é o fato de que a interação entre as unidades de processamento e os dados mudou completamente. O que antes se conhecia por modelo temporal de computação foi deixado de lado para, nesta nova arquitetura, se tornar um modelo espacial. Em outras palavras, os dados não eram direcionados um a um para uma unidade central de processamento, mas processados continuamente em um sistema distribuído no espaço (??). Tal sistema distribuído é composto de células lógicas e suas conexões, ambas reprogramáveis, ajudando a se alcançar uma eficiência similar a presente em ASICs e flexível como a computação genérica.

Ao contrário da estrutura F+V proposta por Estrin, a maioria dos sistemas reconfiguráveis atuais possuem apenas a parte reconfigurável. Apesar de sistemas reconfiguráveis de alta performance possuírem componentes fixos como processadores e unidades de processamento gráficos (GPUs) (??), a sua ausência reduz o custo de projeto e a flexibilidade do projeto final.

Os sistemas reconfiguráveis atuais utilizam de três meios principais de programação: *Static Random-Access Memory* (SRAM), *Antifuse* e memórias não-voláteis. Usando SRAM, o resultado da síntese é armazenado nas células desta memória e controlam o estado dos transistores das células lógicas. No caso de células compostas de tabelas de busca (*look-up tables* ou LUTs), a SRAM armazena os dados dessas células. Outra segunda tecnologia de programação, o *antifuse*, faz uso de uma conexão com impedância variável, onde através do uso de altas voltagens pode-se modificar a resistência de uma via. Esse processo de programação é irreversível. As memórias não voláteis, como EPROM, EEPROM e FLASH, usam transistores especiais com uma ponte flutuante. Quando a ponte possui carga, o transistor pode ser controlado pela ponte de seleção, que permanece carregada até quando desligada. Estas técnicas permitem a resis-

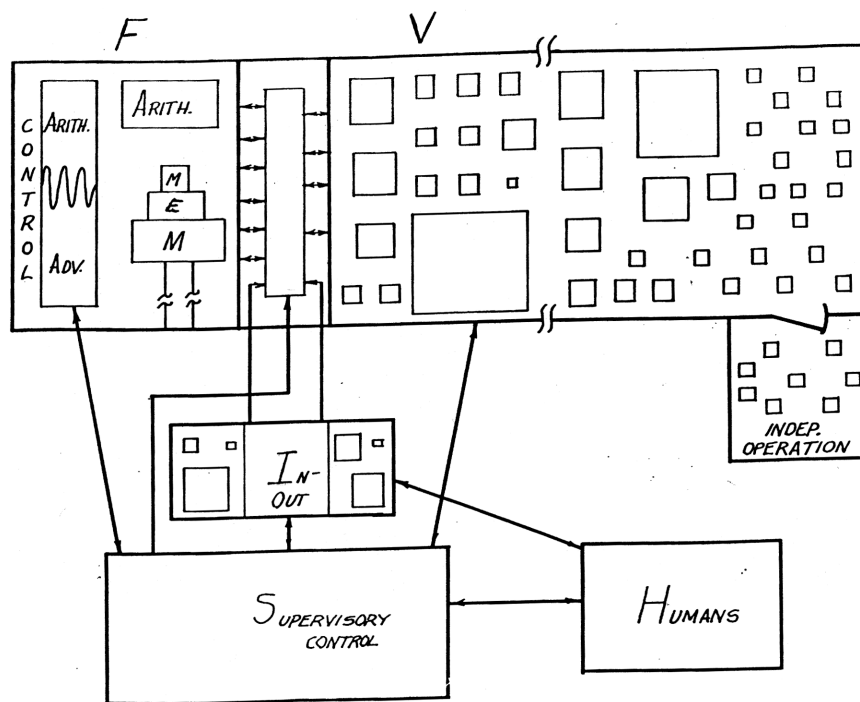


Figura 1.3: Esquemático com o F+V, extraído de (??).

tência da *antifuse* e a reprogramabilidade da SRAM, sendo apenas mais complexa e demorada para ser programada (??).

As interconexões entre células lógicas, que influenciam diretamente as células em si, podem ser de cinco tipos: ilha, linha, mar-de-portas, hierárquico e estruturas unidimensionais (??). A arquitetura do tipo ilha consiste em células lógicas conectadas umas as outras através de caixas de conexões e de roteamento. Nesta arquitetura, a célula lógica está cercada por trilhas de conexões, o que explica o nome. A arquitetura do tipo linha consiste em várias linhas divididas em quantidades variadas de segmentos. As conexões são então realizadas usando-se linhas verticais através de blocos lógicos especiais. A arquitetura do tipo mar-de-portas consiste de blocos lógicos que cobrem todo o espaço do dispositivo e são conectados aos seus vizinhos diretos. Em geral este tipo de conexão é mais rápido. A arquitetura do tipo hierárquico agrupa as células lógicas em *clusters*, e agrupa estes em *clusters* de mais alto nível, formando de fato um sistema hierárquico. O último tipo de arquitetura, unidimensional, surge como uma tentativa de simplificar o roteamento complexo dos sistemas bidimensionais apresentados anteriormente. Nele, restrições de alocação e roteamento são impostas para reduzir o número de possibilidades. O problema deste tipo de arquitetura é que se não houverem recursos de roteamento suficientes, o roteamento fica mais complexo que nas arquiteturas bidimensionais.

As arquiteturas reconfiguráveis podem ser classificadas em três tipos segundo a granularidade. A granularidade diz respeito a quantidade de informação mínima que pode ser passada de uma célula lógica para outra. Ela separa as arquiteturas reconfiguráveis em três categorias: granularidade fina, grossa e híbrida. Nas arquiteturas com granularidade fina, como os *Field-Programmable Gate Arrays*, um único *bit* pode ser transferido de uma célula a outra, permitindo assim um maior controle sobre os dados. Nas arquiteturas com granularidade grossa, os *bits* são agrupados em palavras de tamanhos fixos, reduzindo assim o espaço

gasto com roteamento e melhorando a roteabilidade (??). No último tipo de arquiteturas, a híbrida, parte das conexões são grossas e partes são finas, combinando os benefícios das duas classes.

## 1.2 Motivação

O capítulo de conclusão deveria incluir uma seção de propostas de projetos implementáveis ou otimizáveis através da autorreconfiguração. Seguem a seguir as modificações necessárias.

## 1.3 Propostas de Aplicações

O objetivo deste projeto desde o início foi o desenvolvimento de um sistema capaz de realização de autorreconfiguração, o que consitui o ápice tecnológico da reconfiguração dinâmica parcial. Tendo-se compreendido o fluxo de desenvolvimento para o uso desta tecnologia, propõe-se agora algumas possíveis aplicações.

### 1.3.1 Redes Neurais

Redes neurais são uma ferramenta de inteligência artificial largamente utilizada nas áreas de *Data Mining* e sistemas que necessitem de algum tipo de aprendizado. Elas utilizam de redes de elementos extremamente simples, como mostrado na figura 1.4, para criar uma função altamente não-linear com um comportamento esperado. Estes elementos podem ser matematicamente representados por somatórios ponderados afunilados por funções não-lineares. Os pesos utilizados nestes somatórios são os elementos a serem adaptados de forma a modificar o comportamento da estrutura. Elas podem ser utilizadas tanto aprendizados supervisionados, onde funciona como um poderoso aproximador de funções, quanto em aprendizados não-supervisionados, onde traduz um sistema hiperdimensional em bidimensional.

#### 1.3.1.1 Topologias

As redes neurais podem possuir várias camadas de neurônios, que são as unidades básicas de processamento, e várias entradas e saídas. Quanto mais camadas são acrescentadas ao sistema, mais instável fica seu treinamento e custosa sua computação. Em geral utilizam-se apenas duas ou três camadas com um número arbitrário de neurônios em cada uma. Não existe um algoritmo que determine qual seria a topologia ideal de uma rede para um certo modelo. Tal decisão normalmente é feita com base em experiência e tentativas e erros.

Existem topologias três topologias básicas de redes neurais: a *feedforward*, a recorrente e a atrasada. A primeira possui neurônios que entregam seus resultados apenas para camadas mais a frente. A segunda, mais complexa, além de entregar seus resultados para as camadas mais a frente, realimentam camadas anteriores. A última utiliza de elementos especiais para atrasar o uso de certas entradas ou resultados.

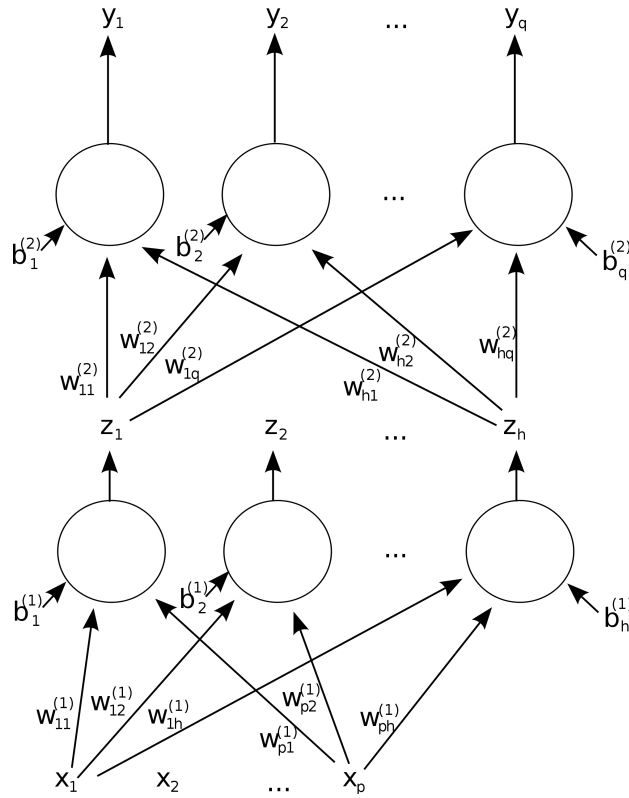


Figura 1.4: Modelo de rede neural com " $p$ " entradas, " $q$ " saídas e duas camadas.

### 1.3.1.2 Formas de funcionamento

As redes neurais podem ser usadas de diversas formas diferentes: através de treinamento *online*, onde o sistema continuamente recebe informações e treina os pesos dos seus neurônios; através de treinamento em lotes, onde o sistema acumula um lote de informações e treina o sistema lote a lote; através de treinamento *offline*. Cada um dos usos listados tem seu uso específico e suas vantagens e desvantagens. O treinamento *online* tem a vantagem de se adaptar mais frequentemente ao sistema que tenta modelar, mas precisa de muito mais processamento/potência para manter o sistema treinado. O treinamento em lotes tem a vantagem de poder ter a frequência de atualização dos pesos modificada de acordo com as necessidades. Sendo assim, a relação taxa de atualização/potência pode ser ajustada. O treinamento *offline* tem a vantagem de considerar um conjunto potencialmente representativo do sistema modelado além da vantagem de ser a forma que precisa de menos processamento, visto que a atualização dos pesos acontece apenas uma vez. A desvantagem dessa forma é a falta de capacidade de adaptação frente a sistemas variáveis no tempo.

### 1.3.1.3 Algoritmos de Treinamento

Além das formas de funcionamento das redes neurais, outra característica que a define grandemente é o algoritmo de treinamento utilizado. Dentre os mais populares existem o *backpropagation* e o de Levenberg-Marquardt. O algoritmo de *backpropagation* é o mais difundido devido a sua facilidade de entendimento e implementação. Já o algoritmo de Levenberg-Marquardt vê o ajuste de pesos de uma rede neural como um problema de otimização.



### 1.3.1.4 Propostas

A computação reconfigurável, mais especificamente a reconfiguração dinâmica parcial, pode ser usada para modificar a topologia da rede neural. Ela pode ser programada para, quando o erro ultrapassar um certo limite máximo, aumentar ou diminuir o número de neurônios numa certa camada. Os desafios referentes a essa proposta são apenas relacionados a implementação de um sistema com reconfiguração dinâmica parcial.

Outra possibilidade é a modificação do algoritmo de treinamento com base em seus custos computacionais e erros. Quando um algoritmo de baixo custo computacional começar a retornar um erro acima de certo limite máximo, ele seria trocado pelo algoritmo imediatamente mais custoso. Os desafios dessa proposta são, além dos relacionados a implementação de um sistema com reconfiguração dinâmica parcial, a implementação em *hardware* de algoritmos de treinamentos de redes neurais.

Uma outra possibilidade seria a de implementar uma autorreconfiguração não-planejada no sistema, onde o número de neurônios e a forma de suas conexões não seriam programados previamente em configurações específicas, como na primeira proposta, mas determinados em tempo de execução. De forma um pouco mais clara, apenas o comportamento e *bitstreams* referentes aos elementos básicos/neurônios seriam programados. Suas conexões e pesos seriam determinados por um sistema controlador presente na lógica estática do sistema reconfigurável. A dificuldade de implementação desse projeto o torna inviável para um trabalho deste porte.

### 1.3.2 Controle Adaptativo

O controle adaptativo tradicional tem como principal objetivo o controle de sistemas incertos ou com características variantes no tempo. Um controlador adaptativo tem a forma mostrada na figura 1.5. Este sistema implementa uma lógica de estimação e correção de parâmetros segundo a estabilidade de Lyapunov.

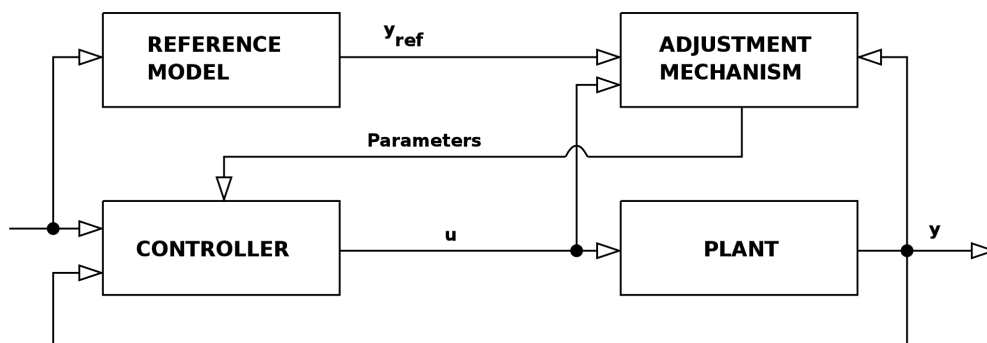


Figura 1.5: Modelo de um controlador adaptativo do tipo MRAC.

#### 1.3.2.1 Propostas

Utilizando o conceito básico deste tipo de controle e a reconfiguração dinâmica, é possível construir um controlador cuja forma é mudada segundo os requisitos dos sistema, i.e. hora seria um controlador

As dificuldades deste projeto dizem respeito a necessidade de construção prévia de um processador genérico funcional, a inclusão de uma memória para armazenamento de configurações, o desenvolvimento de cada configuração parcial para este componente e o desenvolvimento de um controlador para a troca de configurações. É provável que o circuito responsável pela reconfiguração possa ganhar um estágio dedicado a ele, devido a sua complexidade e ao tempo de programação. Para se contornar o problema

do tempo de reconfiguração, pode-se utilizar diversas ALUs, onde apenas uma estaria ativa no ciclo em execução. Sendo assim, outros sistemas de controle podem ser necessários.

Uma outra possibilidade de implementação, menos eficiente, é a reconfiguração do sistema estágio-a-estágio. Ou seja, quando o dispositivo fosse ligado, apenas o primeiro estágio estaria programado. Esta implementação não é recomendada, uma vez que transforma um processador com *pipeline* em um processador uniclo.

### **1.3.4 Outros**

De forma geral, qualquer sistema multiplexado ou que possua diversas etapas de processamento, pode ser implementado utilizando-se de autorreconfiguração. Deve-se, porém, para sua utilização em aplicações reais, verificar se o *overhead* de tempo introduzido é balanceado pelos ganhos de performance e potência.

## **1.4 Objetivos**

### **1.4.1 Objetivos Gerais**

Estudar a

### **1.4.2 Objetivos Específicos**

## **1.5 Metodologia**

## **1.6 Material e Ferramentas**

## **Todo list**