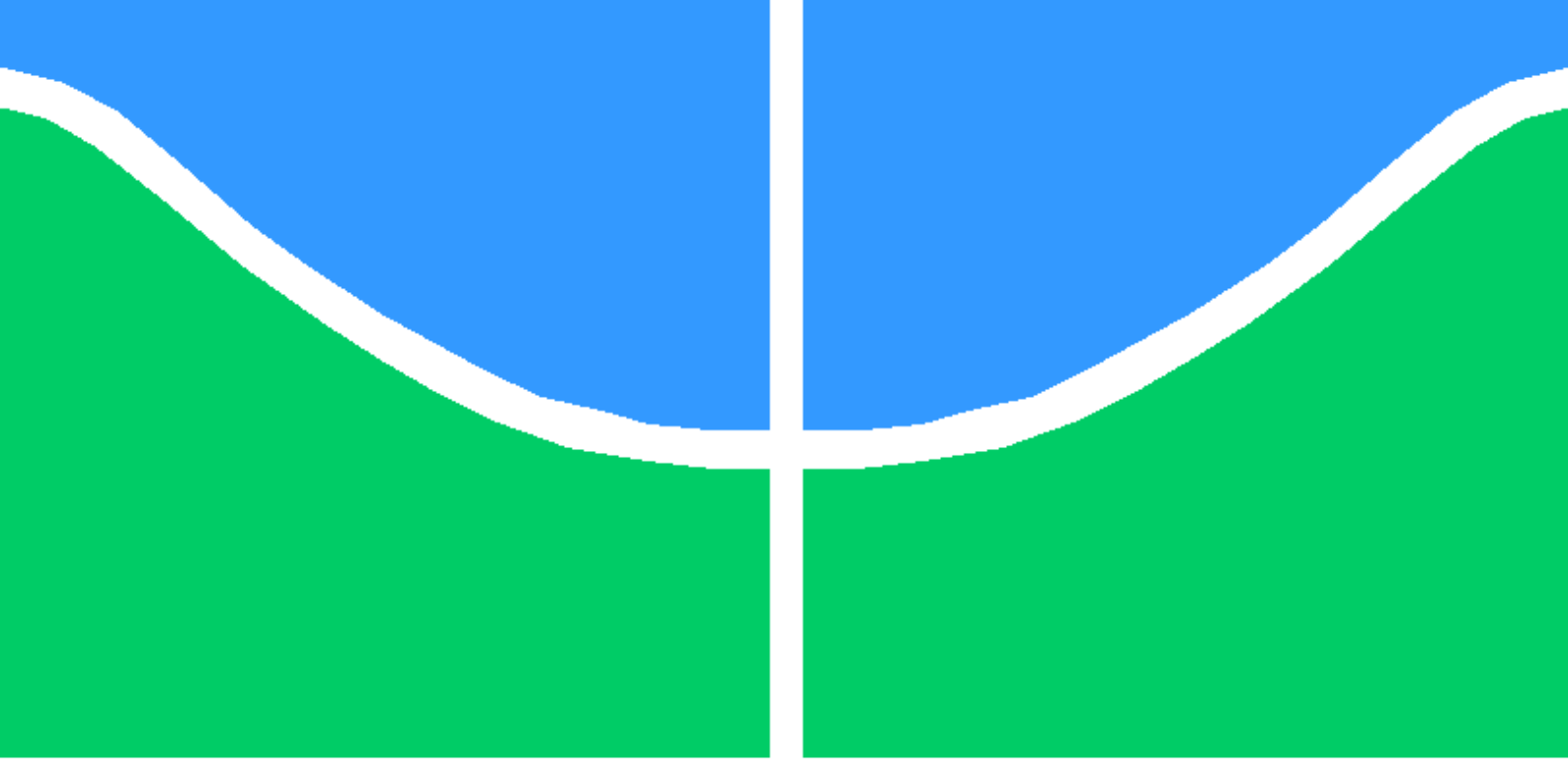

TRABALHO DE GRADUAÇÃO

Gerenciamento de um *Switch Unmanaged* utilizando um controlador ARM7TDMI

Tiago Cavalcante de Rezende

Luiz Raphael Vasconcelos Santos

Brasília, Dezembro de 2009

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

Gerenciamento de um *Switch Unmanaged* utilizando um controlador ARM7TDMI

Tiago Cavalcante de Rezende

Luiz Raphael Vasconcelos Santos

Relatório submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Engenheiro Eletricista

Banca Examinadora

Ricardo Zelenovsky, Doutor, PUC-RJ, ENE/UnB _____
Orientador

Alexandre Ricardo Soares Romariz, Ph.D., Univ. of Colorado at Boulder, ENE/UnB _____

Elber Lopoes da Silva Jr., Mestre, Univ. de Brasília, EMBRATEL _____
Examinador externo

Dedicatórias

Este trabalho é dedicado a todos aqueles que nunca mediram esforços para me incentivar a sempre melhorar.

Luiz Raphael Vasconcelos Santos

Dedico minha monografia àquelas pessoas que acreditaram e apoiaram o meu trabalho. Em especial à minha família e aos meus amigos que conferiram a mim um apoio incondicional..

Tiago Cavalcante de Rezende

Agradecimentos

A minha família por me dar subsídios para que eu me tornasse o que sou. Em especial ao meu pai por me mostrar a importância de persistir; a minha mãe pela paciência e companheirismo; ao meu irmão pela ajuda nos momentos difíceis e a minha avó Célia por nunca duvidar da minha capacidade.
A equipe da ENETEC que desenvolveu este projeto junto comigo: Caio, Vinicius Amaral e Luiz Antonio.
Ao professor Ricardo Queiros por ceder-nos um espaço para que desenvolvêssemos o projeto.
Ao professor Ricardo Zelenovisky pela boa vontade e presteza dispensada ao projeto.
A Helena pela compreensão nos momentos de ausência, pela clareza nos momentos de dúvida e pela alegria divida comigo em todos os momentos.
Ao Murilo pela grande força dispensada a mim antes mesmo de eu começar o meu curso. Sendo fundamental em muitas de minhas escolhas, dentre elas pelo curso de engenharia.
Aos meus amigos: Francisco, Matheus e Rafael, que dividiram comigo, não só infindáveis madrugadas de estudo como também os momentos de procrastinação do mesmo. Sem eles este curso, com toda certeza, seria muito mais complicado.
Em especial ao co-autor deste projeto, Luiz Raphael, um amigo que esteve presente nos momentos decisivos da minha vida nos últimos anos, dispensando a mim um apoio incondicional e uma generosidade surpreendente.

Tiago Cavalcante de Rezende

Primeiramente, à minha família que sempre acreditou em mim, especialmente meus pais pelo amor e apoio incondicionais, mesmo nas horas mais difíceis e meus irmãos pela compreensão, paciência e incentivo constantes.
Um agradecimento especial ao Tiago Rezende, co-autor desse projeto, um verdadeiro amigo.
À Helena Mian, por ser sempre uma grande amiga, em todos os momentos.
Ao Prof. Ricardo Zelenovsky, pela boa vontade e presteza dispensada ao projeto.
Aos amigos e companheiros do projeto Vinícius Amaral, Rafael, Caio Nishiyama, Luis Antonio Barbosa.
Ao Prof. Ricardo Queiroz pela compreensão e por ceder o espaço no GPDS para o desenvolvimento desse projeto.
Finalmente e não menos importante, a todos os amigos e colegas de curso, que de alguma forma me ajudaram a chegar onde estou.

Luiz Raphael Vasconcelos Santos

RESUMO

O presente trabalho trata da implementação do gerenciamento de um *Switch Unmanaged* utilizando um microcontrolador ARM. Para tanto, apresenta-se os conceitos básicos envolvidos no funcionamento de um *Switch*, assim como as especificações e detalhamento da operação do ASIC utilizado no *Switch*. Além disso, o manuscrito faz uma abordagem sobre os microcontroladores que utilizam a arquitetura ARM7TDMI e sobre o ambiente envolvido em sua programação.

ABSTRACT

The scope of the present paper is the implementation of an interface that allows the management of an Unmanaged Switch using an ARM Microcontroller. For so, the basic concepts regarding the operational aspects of a Switch are presented, as well as the specifications and operation of the ASIC used in the Switch. Furthermore, the manuscript considers the microcontrollers that uses the ARM7TDMI architecture and the environment involved in its programming.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Gregos

Ω	Ohm	[V/A]

Siglas

AC	*Alternating Current*
ADC	*Analogic-to-Digital Converter*
ARM	*Advanced RISC Machine*
ASIC	*Application-specific integrated circuit*
CPU	*Central Processing Unit*
DC	*Direct Current*
FIFO	*First In, First Out*
kbps	*Kilo bits per second*
LAN	*Local Area Network*
MAC	*Media Access Control*
MDC	*Management Data Clock*
MDIO	*Management Data Input/Output*
MII	*Media Independent Interface*
Mbps	*Mega bits per second*
OSI	*Open Systems Interconnection*
PHY	*Physical Address*
PIO	*Programmed Input/Output*
RISC	*Reduced Instruction Set Computer*
RMII	*Reduced Media Independent Interface*
VLAN	*Virtual Local Area Network*
SMI	*Serial Management Interface*
SPI	*Serial Peripheral Interface*
WAN	*Wide Area Network*

1 INTRODUÇÃO

1.1 ESTADO DA ARTE E DEFINIÇÃO DO PROBLEMA

O contexto econômico atual impõe que cada vez mais as empresas tenham o acesso amplo à informação e às ferramentas adequadas para compartilhá-la.

A era da informação foi percebida em primeira instância pelas grandes corporações, contudo não foi limitada a elas apenas. De acordo com um estudo realizado pela Mcon Consultoria Empresarial em 1998, cerca de 90% das pequenas e médias empresas já possuíam computadores. [3]

No entanto, os computadores por si não eram capazes de intercambiar informações, havia a necessidade de interconexão dos dispositivos. Esta solução veio com a criação da rede de computadores.

As redes de computadores podem assumir proporções que vão de pequenos perímetros, como uma casa ou um prédio, até proporções globais, envolvendo alguns países ou até continentes, a exemplo da rede Internet.

As redes limitadas geograficamente ou redes locais (LAN) utilizam normalmente uma estrutura formada por um ou mais *Switches Ethernet* ligados entre si operando na camada dois do modelo OSI.[4] Caso haja necessidade de conectividade com a rede Internet, geralmente uma das portas do *Switch* é ligada a um modem ou a um roteador que dê acesso a um provedor do serviço. [5]

Em virtude da estrutura simples das redes locais, normalmente não existem gargalos nas transmissões. As taxas de *throughput* entre os *hosts* de uma mesma LAN apresentam valores nominais altos, sendo limitados pela capacidade da placa de rede de cada usuário ou pelo dispositivo de interconexão, normalmente *Hubs* ou *Switches*. [6]

O principal avanço do *Switch* frente ao *Hub* é o fato de ele conseguir seccionar os *hosts* em diferentes domínios de colisão, fazendo com que os usuários consigam transmitir dados simultaneamente sem que haja descarte de *frames* por colisão. Com isso, os *Switches* conseguem oferecer a taxa de transmissão total a todos os *hosts*. No caso de *Switches* que operam no padrão Ethernet, estes valores são da ordem de dezenas a milhares de Mbps.[7]

Embora a característica principal dos *Switches* seja a de oferecer conectividade entre os hosts de uma rede, alguns dispositivos oferecem funcionalidades que vão um pouco além disso. Estes *Switches* são

conhecidos como *Switches* gerenciáveis e normalmente são requeridos quando surge a necessidade de um maior controle sobre a rede, por isso apresentam um custo de aquisição maior.

Em uma pesquisa feita em lojas de Brasília levantou-se o custo de um *switch* não gerenciável de 16 portas, modelo Enh916p-nwy, produzido pela ENCORE por volta de R$ 89,99. Já um *Switch* gerenciável, com as mesmas características, foi cotado a um custo mínimo de R$ 859,90.

1.2 A MOTIVAÇÃO DO PROJETO

Este projeto surgiu da demanda de uma empresa chamada MUX por um dispositivo que apresentasse as funcionalidades de um *Switch* gerenciável mas que tivesse um custo final próximo ao custo de um *Switch* não-gerenciável.

A MUX atualmente presta o serviço de internet a um condomínio com 224 apartamentos, divididos em 4 blocos. Cada bloco tem 4 andares com 14 apartamentos em cada andar.

O acesso a internet é recebido do provedor por meio de uma fibra, instalada em um CPD no prédio principal, e é distribuído aos usuários finais por meio de um *Switch* instalado em cada andar.

Hoje utilizam-se *Switches* não gerenciáveis, modelo ENH916-NWY do fabricante ENCORE. Como estes *Switches* não são gerenciáveis alguns problemas de operação foram observados, dentre eles:

- A banda disponibilizada a cada *Switch* não é utilizada de forma inteligente;
- a segurança da rede como um todo passa a ser responsabilidade de cada usuário;
- existe DHCP concorrente;
- não é possível desabilitar o serviço em uma porta específica;
- não é possível verificar o *status* de cada porta;
- não existe um *reset* automatizado (*watchdog*).

Com a implantação de um sistema capaz de gerenciar estes *Switches* a MUX irá aumentar a eficiência do serviço prestado. Além disto, conseguirá executar uma assistência remota, evitando o deslocamento de um técnico ao local sempre que haja interrupção no serviço.

Em escala de produção adequada, o gerenciamento do *Switch* deverá acrescer seu custo em aproximadamente US$ 8,00. Considerando um dólar equivalente a R$ 2,00 e uma taxa de imposto de 100 % para importação e de 35 % sobre a venda, o custo final para a MUX deverá ser de, aproximadamente, R$ 43,20.

Como o preço inicialmente pago por cada *Switch* é de R$ 89,99 e o preço da implantação do sistema de gerenciamento é de R$ 43,20, o custo final de cada *Switch* semi-gerenciável será de R$ 113,19. Levando-se em conta que o custo de aquisição de um *Switch* gerenciável seria de 859,90 o projeto representará uma economia de R$ 746,71 por dispositivo, somando R$ 11.947,36 em todo o projeto.

O valor total do projeto, envolvendo os cabos e os ativos de rede, é por volta de R$ 80.000,00. Por isso, a implantação destes dispositivos representará uma economia de 14%.

Comprovada a robustez e a funcionalidade deste dispositivo no projeto piloto, a MUX pretende estender a utilização do *Switch* semi-gerenciável a outros clientes. Sendo utilizado, num primeiro instante, pela própria MUX para conseguir prestar seus serviços a preços mais competitivos. Após isto será estudada a viabilidade comercial para a venda do produto final.

1.3 OBJETIVOS DO PROJETO

O presente projeto tem por objetivo o desenvolvimento de um *firmware* escrito em linguagem C destinado a um microcontrolador ARM.

Este *firmware* deve fazer com que o microcontrolador estabeleça comunicação e faça controle de um *Switch* não-gerenciável. Com isso espera-se programar neste dispositivo funções características de um *Switch* gerencial, tais como ativação ou desativação de cada porta, estabelecimento de VLANs, dimensionamento da largura de banda cada porta e bloqueio de cada porta para um MAC especifico.

Além disso, deverá ser desenvolvida uma plataforma WEB para PC que proporcione uma leitura do *status* do *Switch*, assim como a alteração de alguns de seus parâmetros.

1.4 APRESENTAÇÃO DO MANUSCRITO

No capítulo 2 será apresentado uma descrição geral do dispositivo *Switch*, assim como as características e funcionalidades do *Switch* utilizado no projeto. Concluída esta parte o capítulo 3 tratará da caracterização da arquitetura ARM e da placa de desenvolvimento utilizada.

O capítulo 4 fará uma breve explanação sobre o ambiente de desenvolvimento utilizado para construção do código do firmware, assim como os aplicativos para conexão e transferência de dados entre o PC e o microcontrolador.

Os capítulos 5 e 6 tratam de como o projeto foi desenvolvido e dos resultados obtidos, respectivamente. Finalmente o capítulo 7 apresenta uma conclusão geral do trabalho assim como as perspectivas futuras.

2 SWITCH

2.1 CARACTERIZAÇÃO DO DISPOSITIVO

Os *Switches* são dispositivos comutadores utilizados para reencaminhar *frames* entre os diversos nós de uma rede. Usualmente, os *Switches* operam na camada dois do modelo OSI, ou camada física, utilizando os endereços MAC de cada dispositivo como forma de endereçamento.[4]

Os *Switches* uniram as funções das *Bridges* com as funções dos *Hubs*, porém com um desempenho muito maior. Tradicionalmente, os *Switches* eram utilizados para cascatear *Hubs*, ampliando a capacidade da rede local e conectando-os aos servidores de maior porte. São máquinas com grande capacidade de comutação interna, e que garantem velocidade de linha em cada uma de suas interfaces.

Os *Switches* são comumente dispositivos com várias portas RJ-45 de 10, 100 ou 1000 Mbps. Nestas portas podemos conectar uma única estação ou um *Hub* com várias estações conectadas à ele.

Embora possível a ligação direta de uma estação na porta do *Switch*, isto somente era feito para os servidores de maior tráfego, ficando as estações ligadas através de *Hubs*, mais baratos que os *Switches*.

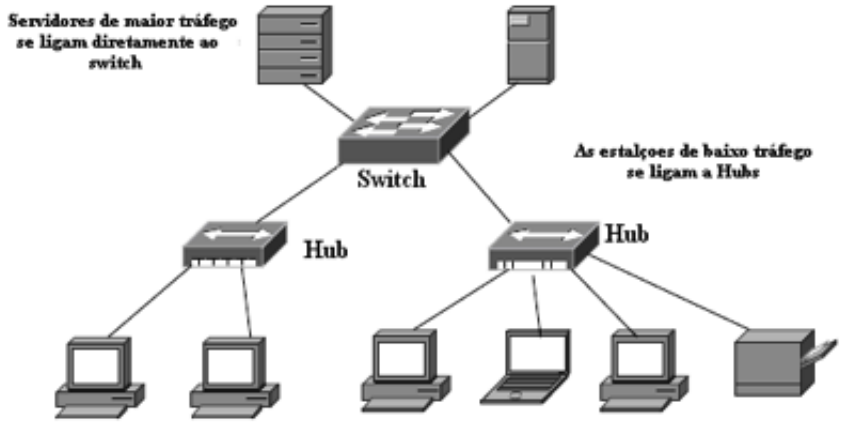

Figura 2.1: Arquitetura Clássica Para uso de *Switches* e *Hubs*.

No início, os *Switches* eram elementos de rede mais caros e, portanto, somente os servidores de maior tráfego eram ligados diretamente ao *Switch*, garantindo uma banda de 100 Mbps, enquanto que *hubs* eram conectados a estes *Switches*, reservando 100 Mbps para todas as estações (arquitetura clássica), como mostrado na Figura 2.1.

Hoje, entretanto, um *Switch* de 24 portas *Fast Ethernet* e 02 portas *GigaEthernet*, custa pouco, porém

com um desempenho de 100 Mbps por estação. Um desempenho muito superior ao de um *Hub* comum.

A utilização de *Switches* aumenta significativamente o desempenho das redes locais, através da eliminação completa das colisões. Esta operação de utilizar *Switches* no lugar dos *hubs* é chamada de *Ethernet Switching*. [8]

Pelo fato deste uso do *Switch* realizar também uma segmentação do barramento ao nível de uma única estação, esta operação também é conhecida por micro-segmentação.

Com o barateamento dos *Switches* hoje é comum verificarmos redes Ethernet onde as estações são diretamente conectadas ao *Switch*. Normalmente, as Bridges são dispositivos com poucas portas que visam a segmentação dos barramentos. Com os *Switches* e o conceito de micro-segmentação, isto é, cada estação da rede conecta a uma porta do *Switch*, como mostrado na Figura 2.2 as colisões foram totalmente eliminadas, garantindo-se uma taxa de transmissão igual à velocidade da porta.

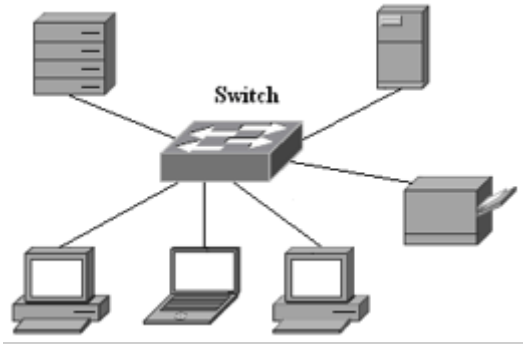

Figura 2.2: Arquitetura Micro-segmentada.

Ao se analisarem estes dispositivos levando-se em conta sua gerência é possível dividi-los em duas classes de equipamentos: os *Unmanaged Switches* e os *Managed Switches*.

Os *Switches* pertencentes à primeira classe, geralmente, são dispositivos do tipo *plug and play*, não apresentando uma interface de opções ou controle. Em virtude da simplificação destes dispositivos eles são oferecidos a preços mais acessíveis que os *Managed Switches*, sendo largamente encontrados em residências e pequenos escritórios.

Já os *Managed Switches* apresentam pelo menos uma maneira de controle de sua operação. Este controle pode ser feito via uma porta serial ou via comandos TELNET. Alguns modelos mais simples oferecem o controle via interface web.

2.2 *SWITCH* UTILIZADO

2.2.1 Caracterização

Utilizou-se neste projeto o *Switch* ENH908-NWY, do fabricante ENCORE. Este modelo é um *Switch* do tipo *plug and play*, que não necessita de instalação prévia ou de qualquer tipo de configuração para que entre em operação. Ele é composto de oito portas *Fast Ethernet* (10/100 Mbps) do tipo RJ-45 que funcionam de acordo com as especificações IEEE802.3, IEEE802.3u, e IEEE802.3x.

O ENH908-NWY opera no modo *Store-and-foward*. Neste modo cada pacote recebido pelo dispositivo é analisado através de uma checagem CRC (*cyclic redundancy check*) para verificação da existência de possíveis erros. Caso o pacote contenha algum erro é descartado, por outro lado se o pacote está integro ele é mandado ao nó seguinte da rede, baseado em seu endereço MAC de destino.

Em sua carcaça externa, o ENH908-NWY possui, além das oito portas *Fast Ethernet*, apenas um *plug* de alimentação e os LEDs para indicação de sua atividade. Como é ilustrado na Figura 2.3.

Figura 2.3: O Encore *Switch* ENH908-NWY

Em condições normais, este *Switch* opera em modo de corrente contínua, de acordo com as seguintes especificações:

Tabela 2.1: Características elétricas do ENH908-NWY

Parâmetro	**Valor mínimo**	**Valor típico**	**Valor máximo**	**Unidade**	**Condições**
Tensão de alimentação	1.80	1.95	2.05	V	Todas as portas desconectadas
Consumo de Potência		1.35		W	100 Mbps Tensão=1.95V

No seu interior o ENH908-NWY utiliza o ASIC (*application-specific integrated circuit*) IP178C. Este

ASIC não é exclusivo do modelo do *Switch* em questão, sendo utilizado também em alguns *Switches* do tipo *managed*. Por isso, apresenta diversas funcionalidades que não são totalmente exploradas pelo ENH908-NWY.

A tabela seguinte explora algumas das funcionalidades do IP178C, assim como os endereços de seus pinos de controle:

Tabela 2.2: Funcionalidades do *Switch* [1]

Função	**IP178C**	
EEPROM	24C01A	
SCA (Smart Cable Analysis)	Sim	
UPDATE_R4_EN	Não	
8 TP + Porta MII (*Switch* de 9 portas)	8 TP + Porta MII (*Switch* de 9 portas)	
	porta MII desabilitada (pin 53 EXTMII_EN=0)	porta MII habilitada (pin 53 EXTMII_EN=1)
Pinos de LED	*Link, Speed, Duplex*	*Link, Speed*
Link quality LED	Pino 73	Ligado de padrão
VLAN_ON	Pino 79	Desligado de padrão
Filter reserved address option	Pino 78	Desligado de padrão
Broadcast frame option	Pino 77	Desligado de padrão
Aging option	Pino 77	Ligado de padrão
Flow control option	Pino 75	Ligado de padrão
Max packet length option	Pino 101	Desligado de padrão
MII port speed/duplex	Não	*Fixed 100 Mbps full*
RMII/MII option	Não	Pino 72
MII MAC mode/ PHY mode	Não	Pino 104
MII register, MDC/MDIO	Não	Sim
Built in regulator	2.5v → 1.95V	3.3V → 1.95V

O ENH908-NWY pode operar tanto no modo Fast Ehernet 10 Mbps quanto no modo Fast Ethernet 100 Mbps. Esta configuração pode ser determinada por meio de alguns ajuste em pinos específicos do ASIC, com resistores de *pull up/down* ou ainda através da interface SMI. No entanto, ele opera em modo padrão utilizando a função de Auto-Negociação.

A Auto-Negociação é uma funcionalidade do padrão Ethernet que permite que os dispositivos divulguem a sua capacidade de transmissão uns para os outros, através de sinais FLP (*Fast Link Pulse*). Com isso, os dispositivos conseguem ajustar individualmente uma configuração para o modo que maximize a sua operação. Ou seja, se o ENH908-NWY é conectado a um *host* que possui uma placa de rede que opere no padrão Fast Ehernet 10 Mbps ele irá ajustar sua operação o padrão de 10 Mbps. Caso seja conectado a uma placa de rede operando no padrão Fast Ehernet 100 Mbps irá configurar sua operação para o mesmo modo.

[1] Os valores padrão podem ser alterados por meio da porta SMI.

2.3 INTERFACE MII/RMII

O ASIC IP178C apresenta uma interface MII/RMII além das oito portas que são utilizadas no ENH908-NWY. [1]

Uma interface MII (*Media Independent Interface*) é uma interface de conexão entre um controlador de MAC *Fast Ethernet* (i.e. 100Mb/s) e um dispositivo de camada física (PHY). No caso do IP178C, esta porta pode ser utilizada para que o *Switch* trabalhe em modos operacionais diferentes do *default*. Uma delas, por exemplo, é a operação como um roteador de oito portas.

Para se habilitar a interface MII/RMII deve-se ajustar o pino 53 EXTMII_EN em nível alto. Isto pode ser feito por meio de resistores de *pull up/down*, conforme ilustrado nas figuras 2.4 e 2.5.

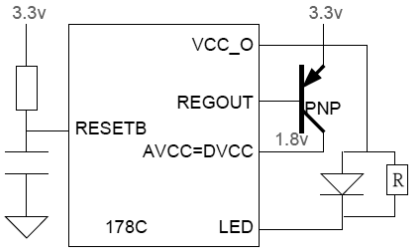

Figura 2.4: Esquemático do funcionamento interno do pino 53 para EXTMII_EN=1.[1][2]

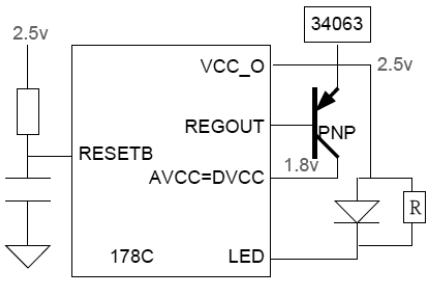

Figura 2.5: Esquemático do funcionamento interno do pino 53 para EXTMII_EN=0[1][3]

Em modo de operação normal tem-se o pino 53 EXTMII_EN = 0, neste caso a porta MII/RMII está desativada e o IP178C funciona como um *Switch* de oito portas, conforme ilustrado na figura 2.6.

[2]R é um resistor de pull-up de 92.6 kΩ e, portanto, deve ser conectado a VCC_O.

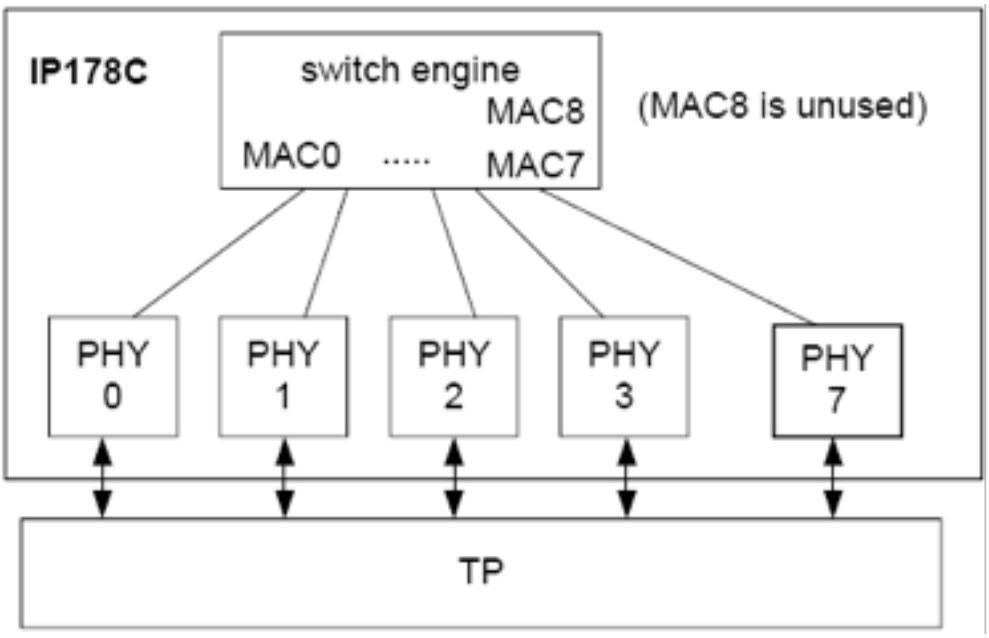

Figura 2.6: Esquemático do *Switch* operando com 8 portas.[1]

Com o pino 53 EXTMII_EN = 1, temos a porta MII/RMII ativa e o IP178C pode operar tanto como um *Switch* de 9 portas quanto um roteador de 8 portas.

Operando como um *Switch* de 8+1 portas a nona porta do *Switch* é conectada ao PHY através da interface MII/RMII. Como o IP178C não acessa o registrador MII do PHY externo a interface MII/RMII trabalha em modo MAC *full duplex*. Neste modo o processador opera segundo o esquema mostrado na figura 2.7.

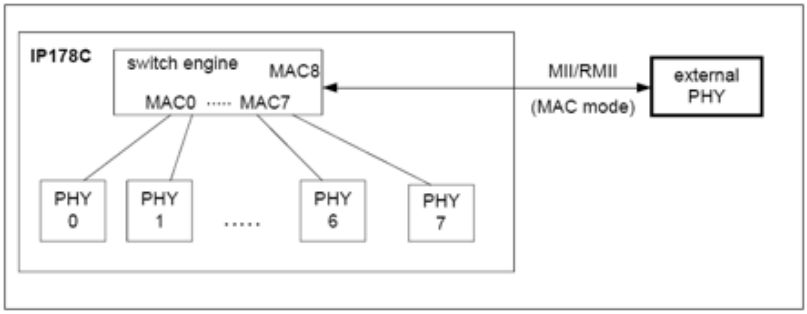

Figura 2.7: Esquemático do *Switch* operando com 9 portas em modo MAC.[1]

Considerando-se ainda o pino 53 em estado 1, o ENH908-NWY pode operar como um roteador. Neste caso o IP178C é conectado a um CPU através da interface MII/RMII, funcionando como um roteador de 8 portas. Para esta configuração a interface MII/RMII deve ser configurada em modo PHY, *full duplex*. Neste caso o roteador será composto de sete portas LAN e uma porta WAN.

Sendo assim, ele irá rotear um pacote direto para a porta WAN sempre que o endereço de destino não pertencer ao seu domínio, conforme ilustrado na figura 2.8

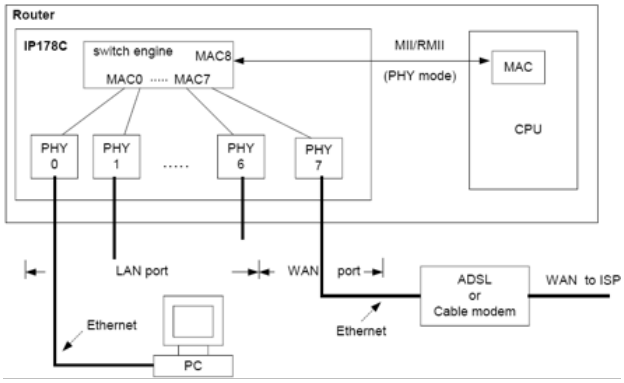

Figura 2.8: Esquemático do *Switch* operando como um roteador com 9 portas em modo PHY.[1]

2.4 INTERFACE SMI

A SMI (*Serial Management Interface*) é uma interface serial constituída de dois pinos: o MDC e o MDIO. Ela faz a conexão entre um dispositivo de controle e dispositivos do tipo PHY, permitindo o acesso e a modificação dos registradores internos de dispositivos PHY, assim como a leitura dos status do PHY.

O MDC (*Management Data Clock*) é uma interface que proporciona um *Clock* assíncrono. Já o MDIO (*Management Data Input/Output*) é uma interface bidirecional capaz de acessar 32 registradores. Ao contrário do MDC, o MDIO permite a transmissão de dados em ambos os sentidos, como ilustrado na figura 2.9.

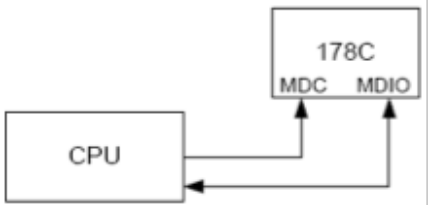

Figura 2.9: Esquemático da Interface SMI.[1]

No IP178C consegue-se acesso aos registradores do MII através da interface SMI, com conjunto MDC e MDIO. Esta interface é acessada por meio dos pinos 103 e 102. [1]

As estruturas dos *frames* que constituem as instruções enviadas pelo MDIO devem ter a informação mostrada na tabela 2.3.

Tabela 2.3: Formato de um quadro para leitura e escrita

Frame Format	**<Idle><start><op code><PHY address><Registers address><turnaround><data><idle>**
Read Operation	<Idle><01><10><A4A3A2A1A0><R4R3R2R1R0><Z0> <b15 b14 b13 b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1b0><Idle>
Write Operation	<Idle><01><01><A4A3A2A1A0><R4R3R2R1R0><10> <b15 b14 b13 b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0><Idle>

Abaixo, nas figuras 2.10 2.11 tem-se exemplificada a execução de um comando de escrita e leitura endereçado ao registrador 0 do PHY 1.

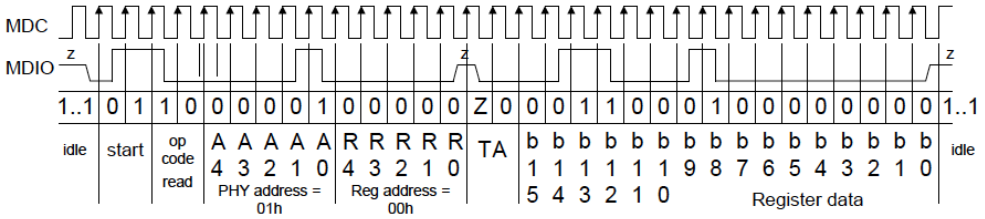

Figura 2.10: Exemplo de um comando de escrita do SMI.[1]

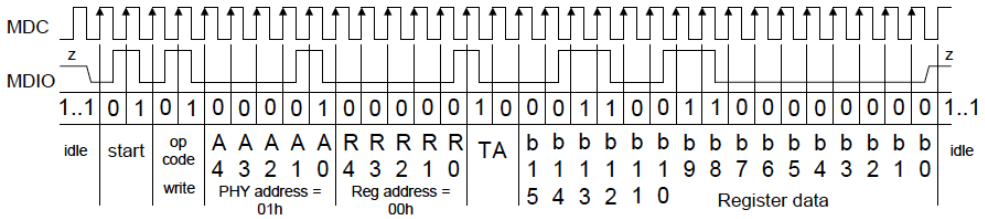

Figura 2.11: Exemplo de um comando de leitura do SMI.[1]

Os processos de leitura e escrita devem ser desenhados levando-se em conta os intervalos mínimos de atraso expressos a seguir, para que se tenha uma segurança maior na interpretação do comando.

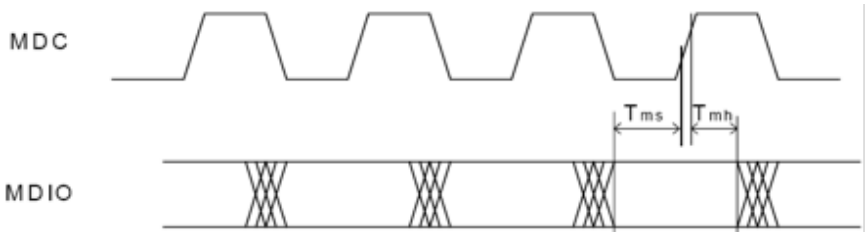

Figura 2.12: Ciclo de escrita.[1]

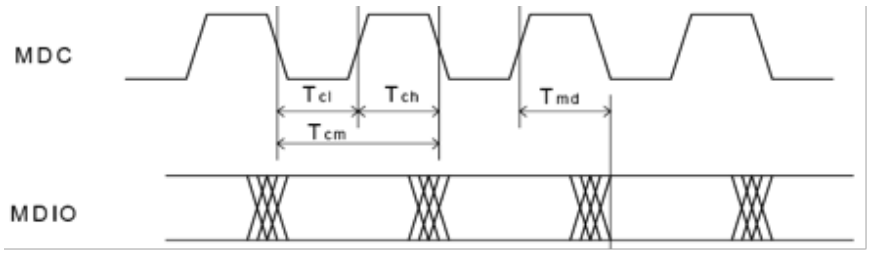

Figura 2.13: Ciclo de leitura.[1]

Tabela 2.4: Parâmetros de tempo do MDC e MDIO

Symbol	Descrição	Min.	Tip.	Max.	Unidade
T_{ch}	MDC *High Time*	40	-	-	ns
T_{cl}	MDC *Low Time*	40	-	-	ns
T_{cm}	MDC *Period*	80	-	-	ns
T_{md}	MDIO *output delay*	-	-	5	ns
T_{mh}	MDIO *setup time*	10	-	-	ns
T_{ms}	MDIO *hold time*	10	-	-	ns

As tensões mínimas para se sensibilizar o IP178C, assim como a máxima permitida para o processo de leitura e escrita estão expressos abaixo, juntamente com o *clock* típico para esta operação.

Tabela 2.5: Parâmetros de tensão do MDC e MDIO

Parameter	Sym.	Min.	Typ.	Max.	Unit	Condtions
Input Low Voltage	VIL			0.8	V	
Input High Voltage	VIH	2.0			V	
Ouput Low Voltage	VOL			0.4	V	IOH=4mA, VCC_O_x=3.3V
Ouput High Voltage	VOH	2.4		5	V	IOL=4mA, VCC_O_x=3.3V

Tabela 2.6: Parâmetro de frequência do MDC e MDIO

Parameter	Sym.	Min.	Typ.	Max.	Unit	Conditions
Frequency			25		MHz	
Frequency Tolerance		-50		+50	PPM	

2.5 FUNÇÕES DISPONÍVEIS

2.5.1 Controle de Fluxo

O controle de fluxo (*Flow Control*) é o controle da taxa de transferência entre dois *hosts* da rede, para evitar que o *host* transmissor envie mais bits que o *host* receptor possa processar.

Para tanto, o IP178C indefere a transmissão em uma determinada porta assim que o número de pacotes na fila (*queueing*) exceda o limiar estabelecido. Isto é feito através do envio de um frame que comprime ou congela a transmissão na porta em questão.

O controle de fluxo pode ser ativado por meio de um *pull-up* no pino 75 X_EN ou acessando o PHY 30 registrador 1.10.

Esta função é desativada por dois ou três segundos sempre que o COS está ativo e o IP178C recebe um pacote de alta prioridade. Isto é feito para garantir que os pacotes de alta prioridade tenham uma banda alocada.

2.5.2 Broadcast Storm Protection

O *broadcast* storm acontece ao se ter uma mensagem enviada ao endereço de acesso FF-FF-FF-FF-FF-FF. Quando isso ocorre, a mensagem é replicada em todas as portas do *Switch*. Estas mensagens geram mais respostas, que vão se multiplicando em um efeito bola de neve, fazendo com que a capacidade de transmissão do *Switch* fique extremamente debilitada.

Ao se ativar o *Broadcast Storm Protection* o IP178C descarta todos os pacotes recebidos de *broadcast* que estão abaixo de um limiar de tempo definido no registrador 31.9[15:14], podendo ser de 10 ms no caso de uma transmissão de 100 Mbps ou 100 ms no caso de uma transmissão de 10 Mbps.

O controle pode ser ativado programando-se o registrador 30.1[6] ou por meio de um *pull up* no pino 91 BF_STM_EN.

Este problema pode ser evitado caso o protocolo *spanning tree* esteja ativo.

2.5.3 Port Locking

Com esta função ativa o IP178C restringe o tráfego de uma determinada porta a um endereço especifico de MAC. Com isso, qualquer dispositivo que transmita com um MAC diferente do ajustado tem os seus

pacotes descartados.

Cada porta do controlador pode ser configurada separadamente programando o registrador 30.31[8:0]. Ao se ativar esse registrador, a porta em questão irá travar o seu primeiro endereço MAC. Para o *reset* do bloqueio de porta deve-se enviar o código 0x55AA para o registrador 30.0 do MII.

Ao se utilizar a função de *port locking* o usuário deverá desabilitar a função de *aging*. Isto pode ser feito programando-se o registrador 30.1[3:2], ou com um *pull low* no pino 76 *AGING*.

2.5.4 Port Base VLAN

VLAN (*Virtual Local Network*) é uma rede que segmenta os domínios de uma rede local de maneira lógica. Com isso, em um mesmo *Switch* podem ser configuradas várias VLANs. Com a implementação da VLAN consegue-se segmentar o domínio de *broadcast* da rede e, além disso, pode-se também limitar a utilização de dispositivos, restringindo o acesso somente à LAN virtual.

Com esta função ativa, o IP178C limita a comunicação de uma determinada porta somente as portas pertencentes sua VLAN. Ou seja, os *frames* advindos de um *host* pertencente a uma VLAN não são encaminhados para dispositivos fora grupo.

O processador permite que cada uma de suas portas pertença a mais de uma VLAN. A programação de cada porta é feita através dos registradores do MII 31.0[8:0] até 31.8[8:0].

O IP178C também traz uma configuração padrão de VLAN que não necessita de intervenção no registradores do MII. Esta configuração pode ser feita aplicando-se *pull up/down* no pino 79 VLAN_ON em conjunto com o pino 53 EXTMII_EN. Os arranjos possíveis estão indicadas na tabela 2.7

Tabela 2.7: Configurações de VLAN *Default* utilizando o pino 53. [4]

VLAN_ON	**EXTMII_EN**	**Configuration**
0	X	*Function disabled*
1	0	VLAN *Groups*: (P0, P7), (P1, P7), (P2, P7), (P3, P7), (P4, P7), (P5, P7), (P6, P7)
1	1	VLAN *groups*: (P0, MII), (P1, MII), (P2, MII), (P3, MII), (P4, MII), (P5, MII), (P6, MII), (P7, MII)

[4]MII indica a porta MII, P0 a P7 indicam as portas zero a sete..

2.5.5 CoS

Class of Service (CoS), é uma técnica de classificação e priorização de tráfego utilizada para otimizar a transmissão em redes de dados. Esta classificação é feita levando-se em conta o tipo e aplicação à qual o pacote pertence, por exemplo voz, vídeo, email, http, etc.

Este protocolo opera na camada de enlace dos dados, utilizando três bits do cabeçalho do quadro *ethernet* para classificá-lo de acordo com sua prioridade, indicando para isto um valor de 0 a 7.

O IP178C suporta dois tipos de CoS, um modelo de CoS por porta e outro por frame. Para a priorização do tráfego o IP178C utiliza um mecanismo de duas filas. A determinação da banda alocada para cada fila é feita através do registrador 30.1[15].

Quando se opta pela priorização por porta, um pacote recebido em uma porta de alta prioridade é tratado como um quadro de alta prioridade. Para isto deve-se ativar o bit correspondente ao registrador do MII da porta em questão indo do 31.0[9] até o 31.8[9].

Ao se ativar o modo de prioridade por quadro, o IP178C examina os bits específicos do *tag* da VLAN do pacote tratado e marca como um quadro de alta prioridade caso o valor encontrado para o COS seja maior do que 3. Cada porta do *Switch* pode ter um lista de prioridade diferente, a sua configuração deve ser feita através dos registradores 31.0[10] até 31.8[10].

A marcação do campo de prioridade do quadro VLAN deverá ser feita utilizando-se os bits 13,14 e 15 do tag.

Além disso, o IP178C pode determinar a prioridade de um frame checando no campo DiffServ de pacote Ipv4. Para isso basta ativar o registrador 31.30[13] DIFFSERV_EN.

2.5.6 Controle de largura banda

O IP178C apresenta um mecanismo de controle de banda para redes que tem uma taxa limitada. Com isso consegue-se um gerenciamento da banda disponível na rede local, podendo-se alocar facilmente as velocidades de *uplink* e *downlink* de cada uma de suas portas.

A velocidade de tráfego pode ser ajustada em uma faixa que vai de 128 kbps a 8 Mbps através da programação dos registradores 31.26 até 31.29.

3 CARACTERIZAÇÃO DA PLACA DE DESENVOLVIMENTO

3.1 A EVOLUÇÃO DA ARQUITETURA ARM

Utilizou-se neste projeto um kit de desenvolvimento modelo SAM7EX256 do fabricante Olimex. Este kit conta com um microcontrolador baseado em arquitetura ARM, o AT91SAM7X256.

O termo ARM atualmente é um acrônimo de *Advanced RISC Machine*. Porém, em meados dos anos 80 era tido como uma abreviação de *Acorn RISC Machine*, dando referência à empresa criadora da arquitetura, a Acorn.

A Acorn criou esta arquitetura entre 1983 e 1985, para sua própria utilização. No fim dos anos 80, a Apple e a VLSI se juntaram à Acorn para desenvolvimento de núcleos de processamento ARM, fundando a *Advanced RISC Machine Ltd*. Atualmente, a ARM não fabrica Chips, ela licencia a sua propriedade intelectual para que outros fabricantes de microprocessadores e microcontroladores produzam. Dentre os fabricantes licenciados para utilização da arquitetura ARM encontram-se: Alcatel, Atmel, Broadcom, Cirrus Logic, Digital Equipment Corporation, Freescale, Intel (por meio da DEC), LG, Marvell Technology Group, NEC, NVIDIA, NXP (antes Philips), Oki, Qualcomm, Samsung, Sharp, ST Microelectronics, Symbios Logic, Texas Instruments, VLSI Technology, Yamaha e ZiiLABS.

Originalmente essa arquitetura era desenvolvida para utilização em computadores pessoais. No entanto, este mercado foi dominado pelos processadores da família x86, utilizados pela IBM. Com isso, os processadores ARM acabaram ganhando espaço naquelas aplicações que exigiam um alto nível de desempenho em sistemas embarcados, juntamente com um consumo de potencia eficiente.

Surgiram diversas versões da arquitetura ARM ao longo de seu desenvolvimento. Essas versões resultaram na criação de algumas famílias, sendo cada família subsequente uma evolução da anterior. A tabela 3.1 ilustra algumas das famílias da arquitetura ARM.

Tabela 3.1: Algumas famílias da série ARM

	Description	ISA	*Process*	*Voltage*	*Area* mm^2	*Power mW*	*Clock* / MHz	Mips / MHz
ARM7TDMI	*Core*	V4T	0.18u	1.8 V	0.53	< 0.25	60 - 110	0.9
ARM7TDMI-S	*Synthesizable Core*	V4T	0.18u	1.8 V	< 0.8	< 0.4	> 50	0.9
ARM9TDMI	*Core*	V4T	0.18u	1.8 V	1.1	< 0.3	167 - 220	1.1
ARM920T	*Macrocell* 16+16kB Cache	V4T	0.18u	1.8 V	11.8	< 0.9	140 - 200	1.05
ARM940T	*Macrocell* 8+8 Cache	V4T	0.18u	1.8 V	4.2	< 0.85	140 - 170	1.05

3.2 ARM7TDMI

3.2.1 Especificações

A arquitetura ARM é baseada na tecnologia RISC (*reduced instruction set computer*). Optou-se por este modelo, pois o conjunto de instruções RISC, assim como sua decodificação, é muito mais simples do que aquele utilizado no modelo CISC, permitindo uma taxa maior na leitura das instruções e uma macrocélula pequena e de baixo custo.

Para este projeto utilizou-se a CPU AT91SAM7X256. Este modelo pertence à família ARM7TDMI, cujas principais características estão resumidas na tabela 3.2

Tabela 3.2: Funcionalidades do *Switch*

Função	RISC de 32 bits	
Instruções	ARM	32 bits (alta performance)
	Thumb	16 bits (alta densidade de código)
Pipeline	três estágios: fetch, decode, execute	
Velocidade de processamento	2000 MIPS (*million of instructions per second*)	
Frequência	115 MHz	
Área	0,59 mm^2	

O ARM7TDMI utiliza o pipeline para incrementar a velocidade do fluxo de instruções para o processador. Com isso, várias operações são executadas simultaneamente fazendo com que os sistemas de processamento e de memória operem continuamente.

Esta família já utiliza a versão ARM4T da arquitetura ARM e conta com 3 estágios em seu pipeline, sendo eles o *Fetch*, o *Decode* e o *Execute*.

- Estágio *Fetch*: as instruções são buscadas da memória.
- Estágio *Decode*: os registradores utilizados pela instrução são decodificados
- Estágio *Execute*: leem-se os registradores do banco de registradores, fazem-se as operações na ULA e escrevem-se os registradores de volta no banco de registradores.

Em modo de operação normal esses três estágios são executados simultaneamente. Ou seja, quando uma instrução esta sendo executada sua sucessora esta sendo decodificada e uma terceira instrução está sendo buscada na memória.

O ARM7TDMI pode operar com dois tipos de conjuntos de instruções: o modo ARM, com um conjunto de instruções de 32 bits, e o modo *Thumb*, que conta com um conjunto de instruções de 16 bits.

As instruções do código no modo *Thumb* são um subconjunto das instruções do modo ARM comprimidas para 16 bits, em razão disso ele apresenta-se como um conjunto de instruções mais condensado, ocupando em média 30% a menos de memória.

No entanto, antes de serem executadas, as instruções do modo *thumb* têm de ser descomprimidas. Esta operação é feita em tempo real, o que faz com que o processador tome mais tempo para executar uma mesma tarefa e perca em desempenho.

Para edição do código em Linguagem C devem-se utilizar os identificadores _arm e _thumb para que o compilador gere o código ARM ou *Thumb* para determinada função.

Os microcontroladores ARM7TDMI podem operar em sete modos distintos, sendo eles:

- Modo *User* (USR): é o modo de execução padrão do ARM, sendo utilizado na maioria dos programas de aplicações.
- Modo *Fast Interrupt* (FIQ): é um modo que suporta transferências de dados.
- Modo *Interrupt* (IRQ): é um modo utilizado para o tratamento de interrupções comuns.
- Modo Supervisor (SVC): é um modo de operação protegido para o sistema.
- Modo *Abort* (ABT): é um modo utilizado para implementar memória virtual ou proteção de memória. Pode ser executado após a interrupção de busca antecipada de um dado ou instrução.

- Modo *System* (SYS): é um modo privilegiado para a operação do sistema.
- Modo *Undefined* (UND): é um modo ativado quando uma instrução indefinida é executada.

O núcleo de processamento do ARM7TDM trabalha segundo o diagrama apresentado na figura 3.1:

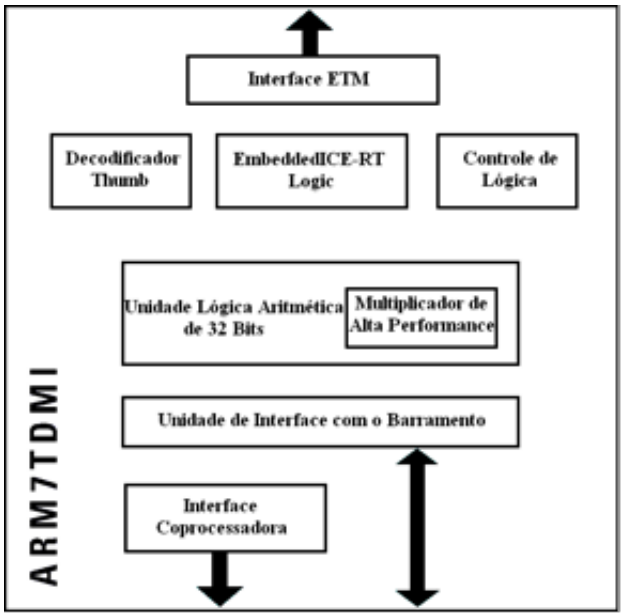

Figura 3.1: Núcleo de processamento do ARM7TDMI.

3.3 MICROCONTROLADOR

3.3.1 Caracterização

Microcontrolador é um dispositivo contido em um único circuito integrado capaz de executar as tarefas de um computador. É composto de uma unidade de processamento, uma pequena quantidade de memória do tipo flash ou RAM e funções de entrada/saída (I/O) analógicas ou digitais. Estes dispositivos são desenhados tendo como fim aplicações dedicadas e relativamente simples.

Neste projeto utilizou-se o microcontrolador AT91SAM7X256 da ATMEL. Este dispositivo é baseado em uma arquitetura RISC ARM7TDMI de 32 bits, contendo 256 kbytes de memória flash de alta performance e 64 kbyte de memória SRAM. [9]

A rotina de instruções a serem executadas pelo microcontrolador deve ser programada em sua memória Flash. Essa programação pode ser feita com uma interface paralela ou utilizando um dispositivo J-Link.

O código de instruções implementado no AT91SAM7X256 pode ser protegido por meio de bits de

travamento contidos na própria memória. Com isso, evita-se que o código seja exposto ou que se escreva outra rotina de instruções sobre a rotina já carregada em memória.

O AT91SAM7X256 apresenta uma estrutura de *clock* com três osciladores: um oscilador RC de baixo consumo que opera na faixa de 22 kHz a 42 kHz, um oscilador principal que opera entre 3 MHz e 20 MHz e oscilador PLL que opera na faixa de 80 MHz a 200 MHz. Esses osciladores são arranjados de acordo com a estrutura mostrada na figura 3.2.

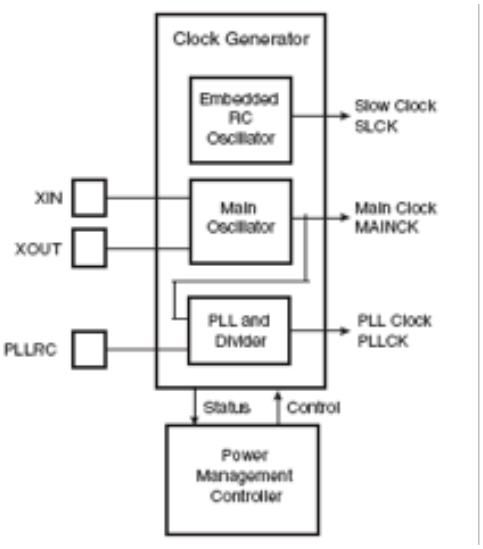

Figura 3.2: Estrutura de *clock* do ARM.[2]

O microcontrolador utilizado pode operar a uma taxa de até 55 MHz, conseguindo desenvolver até 0.9 MIPS/MHz. Uma das características determinantes na escolha do AT91SAM7X256 foi sua vasta gama de funções periféricas, dentre elas destacam-se:

- Porta USB 2.0 *full-speed* integrada, utilizando FIFOs de 328 bytes para os *endpoints*;
- Porta *Ethernet* MAC 10/100 base T;
- Suporte a J-Link para *debug*;
- Unidade Serial de *debug*;
- *Timer* de intervalos periódicos com contador de intervalos de 12 bits;
- *Timer* de tempo real, que conta em segundos;
- *Timer/Counter* de 16 bits com três canais independentes;
- Duas portas USART com taxas independentes;

- Interface SPI;
- Conversor ADC de 8 canais com resolução de 10 bits;
- Unidades de Entrada/Saída (I/O) tolerantes a 5 V;
- Quatro linhas de Entrada/Saída (I/O), com *FAN-OUT* de até 16 mA cada;
- Dois controladores Paralelos de Entrada/Saída (PIO).

3.3.2 Watchdog Timer

O *Watchdog Timer* é um dispositivo que dispara um *reset* ao sistema principal do microcontrolador caso identifique alguma falha em sua operação. Ele é utilizado para prevenir que o sistema trave ao ficar preso em um *loop* infinito.

O *Watchdog Timer* é construído com base em um contador regressivo de 12 bits. Este contador é alimentado com o valor defalut 0xFFF, definido no campo WV do registrador WDT_MR. Como a operação do *Watchdog timer* é orientada pelo valor de *clock* do *Slow Clock* (32,768 kHz) divido por 128, o período máximo que o *watchdog* pode atingir é de 16 segundos. [2]

Em um modo de operação normal, o sistema reinicia o *Watchdog* em intervalos regulares antes que o contador seja zerado, ajustando o bit WDRSTT do Registrador de Controle WDT_CR para 1. O contador do *Watchdog* é então imediatamente recarregado do WDT_MR e reiniciado, e o divisor do *Slow Clock* de 128 é resetado e reiniciado.

Para se desabilitar o *Watchdog Timer* deve-se ajustar o valor do registrador WDDIS para 1.

Uma outra maneira de se desabilitar o *Watchdog* é colocando-se o processador em modo de *debug* ou em modo *idle*.

3.4 PLACA DE DESENVOLVIMENTO

3.4.1 Especificações

Para facilitar a utilização do microcontrolador necessitou-se de uma placa de desenvolvimento. Neste ponto analisaram-se alguns modelos disponíveis e optou-se pela SAM7-EX256 do fabricante Olimex. Isto se deu em primeiro lugar pelo fato da placa adotar a arquitetura ARM7TDMI já assimilada pelo grupo

desenvolvedor. Além disso, levou-se em conta o seu nível de processamento, que supre a necessidade do projeto, de alguns periféricos que facilitavam a operação e o seu custo de aquisição.

A SAM7-EX256 disponibiliza para seus usuários as funcionalidades expressas a seguir:

- Microcontrolador: AT91SAM7X256 16/32 bit ARM7TDMI com 256 kBytes *Program Flash*, 64 kBytes RAM, CAN, USB 2.0, Ethernet 10/100, RTT, 10 bit ADC 384 ksps, 2x UARTs, TWI (I2C), 2x SPI, 3x 32bit TIMERS, 4x PWM, SSC, WDT, operação de até 55 MHz;
- Conector JTAG padrão com pinagem ARM 2x10 para programação/*debug* com ARM-JTAG;
- Tela de LCD 128x128 da Nokia 6610, com *display* colorido de 12 bits com *back light*;
- Ethernet 10/100 PHY com KS8721BL;
- Conector USB;
- Dois canais RS232 interface com *drivers*;
- Conector SD/MMC *card*;
- *Joystick* com 4 direções e *push action*;
- Dois botões;
- Entrada e saída de Áudio para microfones e fones de ouvido;
- Auto-falantes integrados com potenciômetro para controle de volume;
- Trimpot conectado ao ADC;
- Thermistor conectado ao ADC;
- Regulador de tensão *on-board* de 3.3V com até 800 mA de corrente;
- Única fonte de alimentação requerida: 6V AC ou DC, pode ser conseguido por meio de uma porta USB;
- LED para indicação de estado;
- Capacitor para filtrar a alimentação;
- Circuito e botão de *RESET*;

- *Socket* com cristal de 18.432 MHz;
- *Headers* de extensão para todas as portas do microcontrolador.

4 AMBIENTE DE DESENVOLVIMENTO

4.1 INTRODUÇÃO

Para um sistema de desenvolvimento C/C++ completo necessita-se de um servidor GDB (*GNU Debugger*), de um pacote de utilitários para *Cross Compilation* ARM e de um IDE (*Integrated Development Environment*).

Esse projeto utiliza um microcontrolador AT91SAM7X256, o que torna inviável a instalação de um sistema de desenvolvimento para compilarmos o código das instruções a serem seguidas utilizando o próprio microcontrolador, pois ele tem uma capacidade de processamento limitada, uma memória flash de apenas 256 kb e memória RAM de 64 kb. Portanto há a necessidade do uso de um *Cross Compiler*.

4.2 CROSS COMPILER

Um *Cross Compiler* é um compilador com capacidade de criar programas executáveis para uma plataforma diferente daquela em que o código foi compilado. Com isso, elimina-se a necessidade de compilação na máquina hospedeira, tornando a prática muito útil a sistemas embarcados, como o utilizado neste projeto, a sistemas executados em diferentes sistemas operacionais ou em diferentes versões de um sistema operacional, etc. [10] [11]

4.3 YAGARTO

O sistema de desenvolvimento utilizado baseia-se na IDE chamada Eclipse e é composto por um pacote de utilitários chamado YAGARTO (*Yet Another GNU-ARM Toolchain*), que tem três princípios básicos: não utiliza Cygwin, funciona bem com o Eclipse e é gratuito. O YAGARTO é dividido em três pacotes:[12]

- A plataforma Eclipse, o eclipse CDT e o *plugin CDT* para o *debugger GDB* embarcado;
- Uma interface de *debug JTAG*, que pode ser tanto o *J-Link GDB Server* ou o *Open On-Chip Debugger*;
- Binutils, Newlib, compilador GCC - que são binários e bibliotecas necessárias para a *Cross Compi-*

ling, tais como make, sh, rm, cp, mkdir etc, e o debugger Insight.

4.3.1 Eclipse

O Eclipse é um ambiente que engloba vários aplicativos necessários para o desenvolvimento C/C++. A partir dele é possível acessar os utilitários do YAGARTO e também o *J-Link GDB Server*, que funciona conectando-se a um emulador - no caso, o SAM-ICE - utilizando uma conexão TCP/IP. A Zylin, uma empresa norueguesa de consultoria fez umas modificações no CDT do Eclipse para Windows, e criou um plugin que facilita o *debug* embarcado utilizando o GDB. Com ele, poderá ser feito um *debug* embarcado, ou seja, realizar um *debug* enquanto o programa desejado roda dentro do microcontrolador.

4.3.2 JTAG

JTAG é um acrônimo para *Joint Test Action Group*, um nome utilizado para o padrão IEEE 1149.1. Inicialmente o JTAG foi projetado para testar circuitos impressos, porém, hoje em dia ele também é utilizado: para conectar-se a um circuito para testes, para *debug* de sistemas embarcados - sendo essa uma conveniente maneira alternativa de conectar-se ao sistema - ou para gravar um *firmware* num dispositivo. Uma de suas funções mais úteis é a de *debug*, neste modo um adaptador JTAG utiliza o JTAG como um mecanismo de transporte para acessar módulos de *debug* dentro do microcontrolador, esses módulos permitem ao desenvolvedor controlar o *debug* de um software de um sistema embarcado durante sua execução. O J-Link é um emulador de JTAG projetado para núcleos ARM7/ARM9, ele conecta um computador rodando Windows via USB e possui um conector JTAG de 20 pinos que é compatível com o conector padrão de 20 pinos definido pelo ARM. Nesse projeto foi utilizado o SAM-ICE, uma versão OEM do J-Link, vendida pela ATMEL.

Figura 4.1: SAM-ICE.

4.3.3 SAM-BA

Iremos utilizar também o SAM-BA (*SAM Boot Assistant*), um software feito pela ATMEL que pode ser usado para facilmente programar dispositivos Atmel AT91SAM. Pode-se usá-lo para conectar-se na placa de desenvolvimento por meio de um JTAG, pela porta USB ou por RS232. No caso, o SAM-BA irá se conectar na placa de desenvolvimento utilizando o SAM-ICE.

4.4 INSTALAÇÃO

Para a instalação e configuração do ambiente de desenvolvimento, é recomendado que se inicie pelo GDB *Server*, o SAM-BA, o pacote de utilitários YAGARTO e então o Eclipse.

5 IMPLEMENTAÇÃO

5.1 INTRODUÇÃO

Um dos desafios para o desenvolvimento desse projeto foi a necessidade de uma interface capaz de fazer uma ligação entre o microcontrolador e o usuário, pois é preciso que a comunicação com o *Switch* seja transparente para o usuário e também que essa interface seja de fácil acesso e de simples manuseio. Necessita-se também que o sistema operacional seja de tempo real, ou seja, um sistema multitarefa cujo tempo de resposta de um estímulo externo possa ser determinado previamente. Uma maneira simples e eficiente de se acessar uma interface de gerenciamento é utilizando um navegador WEB, portanto o desafio tornou-se obter um sistema operacional compacto - pois a memória flash do microcontrolador é de somente 256kb - e com uma pilha TCP/IP para que seja implementado um micro-servidor HTTP que abrigaria uma página para servir como interface com o usuário.[10]

5.2 O FIRMWARE UTILIZADO: FREERTOS

O sistema operacional utilizado é o FreeRTOS (*Free Real-Time Operational System*), ele é de código aberto com um mini *Real Time Kernel* com suporte para 23 arquiteturas diferentes de microcontroladores.

Umas das características do FreeRTOS é que ele foi escrito predominantemente em C de maneira muito simples e compacta. O núcleo do *kernel* está contido somente em 3 arquivos C; suporta rotinas e co-rotinas; não há restrições quanto ao número de tarefas e de prioridades a serem utilizadas e nem restrições quanto às prioridades, ou seja, duas tarefas podem ter a mesma prioridade; possui uma gama de recursos para comunicação e sincronização entre tarefas ou entre tarefas e interrupções; também há a possibilidade de cross-compilação, permitindo utilizar uma máquina Windows para o desenvolvimento. O FreeRTOS também possui como código exemplo uma pilha TCP/IP com um servidor http implementado. Utilizou-se esse servidor para hospedar o ambiente de gerenciamento do *Switch*. [13]

Uma grande dificuldade que houve ao iniciar-se o trabalho com o FreeRTOS na placa de desenvolvimento foi que a placa reiniciava-se automaticamente a aproximadamente cada 4s sem motivo aparente. Afim de solucionar o problema, tentou-se primeiramente rodar um *debug* no FreeRTOS instalado no microcontrolador utilizando o *J-Link GDB Server* em conjunto com o SAM-ICE, pois uma de nossas suspeitas

era de que havia algum conflito de um periférico na placa com o FreeRTOS que fazia com que o *WatchDog* reiniciasse a placa. Porém não conseguimos fazer funcionar o *debug*, pois para executá-lo diretamente do microcontrolador era necessária a compra de uma licença para operar o SAM-ICE o que não foi possível devido ao limitado orçamento do projeto. Pesquisando em fóruns na internet, viu-se que o problema era que a versão do FreeRTOS disponibilizada no sítio do FreeRTOS estava com uma ID do PHY para uma outra placa de desenvolvimento diferente da utilizada no projeto. Portando, para resolver o problema bastou modificar no arquivo "FreeRTOS\Demo\ARM7_AT91SAM7X256_Eclipse\RTOSDemo\SrcAtmel\mii" as linhas de ID do PHY para os seguintes valores:

```
/* PHY ID */
#define MII_DM9161_ID 0x00221610
#define MII_AM79C875_ID 0x00221610
#define MII_KS8721_ID 0x00221610
#define AT91C_PHY_ADDR 1
```

5.3 FUNCIONALIDADES

Com o FreeRTOS operacional, passou-se para a implementação das funcionalidades que eram necessárias para o gerenciamento do *Switch*. O projeto prevê as seguintes funcionalidades: a verificação do status de uma porta do *Switch* (se há um *link* entre um computador e o *Switch*); gerenciamento da largura disponível para cada porta; criação de uma VLAN, para que cada usuário possa ter acesso somente à porta em que o servidor está ligada, e não ao computador dos outros usuários, como podemos ver na figura 5.1; possibilidade de desligar uma porta, cortando o acesso à internet de algum usuário remotamente. [1]

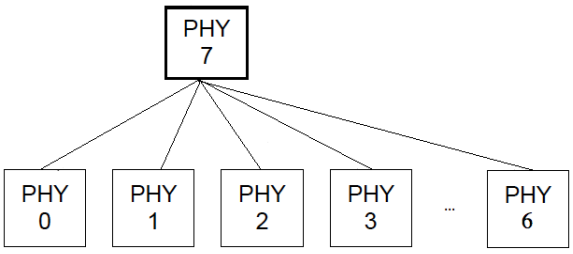

Figura 5.1: Diagrama do funcionamento da VLAN.

O ambiente de gerenciamento do *Switch* inicialmente se baseava em duas áreas de ligação com o mesmo, uma que verifica o status dos LEDs por meio da tensão lida nos terminais dos LEDs de status das

portas do *Switch* e outra que utiliza uma interface de comunicação SMI.

5.3.1 Status das Portas

Na figura 5.2, podemos ver um esquemático dos LEDs de status das portas. A fim de verificar o seu nível de tensão, soldou-se um fio no terminal LINK_LED, o fio foi ligado a um cabo plano conectado na porta EXT de 20 pinos da placa de desenvolvimento como pode ser visto no Anexo I.1. Os pinos de 7 a 14 da porta EXT (PB18, PB21, PB22, PB23, PB27, PB28, PB29, PB30) foram usados para verificação do status dos LEDs das portas 1 a 8. [2]

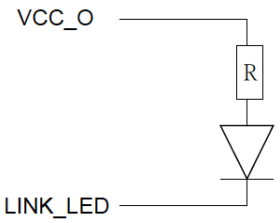

Figura 5.2: Esquemático do LED de status.[1]

O ambiente de gerenciamento verifica o status do LED com a função ler_estado_ext( ) que recebe como entrada o número do pino que é uma variável do tipo portLONG e um caractere para especificar se o pino é PA ou PB e retorna como saída um valor booleano verdadeiro ou falso, ou seja, ligado ou desligado. Essa função é chamada a cada meio segundo por uma outra função chamada “verifica_estado_led” que atualiza o status das portas mostrado na tela.

Porém, após uma análise detalhada do *datasheet* do *Switch*, viu-se que era possível acessar o status das portas por meio de um registrador específico do PHY de cada porta. O funcionamento da verificação continua o mesmo, porém não há mais a necessidade da função ler_estado_ext(), que foi substituída por uma função de leitura de dados pela interface SMI.

5.3.2 Interface SMI

O próprio microcontrolador já possui uma interface SMI implementada, porém ela já é utilizada para comunicação do PHY da placa de desenvolvimento com o microcontrolador, portanto foi necessária a criação de um código que emule uma interface SMI, com um *clock* (MDC) e funções para transferência de dados (MDIO). Os pinos escolhidos foram o PA3 e PA4 da porta EXT da placa de desenvolvimento. Não foi prevista no projeto do *Switch* uma maneira de acessar facilmente o MDIO e o MDC do *Switch*,

que estão nos pinos 102 e 103, respectivamente, conforme pode ser observado no Anexo I.3, portanto foi necessário fazer uma solda do ASIC do *Switch* puxando dois fios até as portas de solda de uma EEPROM que não será utilizada no projeto. [2]

5.3.2.1 Funcionamento

A interface SMI criada por software consiste de duas funções, uma de leitura do MDIO e outra de escrita. Um exemplo de como deve ser o sinal de escrita e leitura utilizando as portas MDC e MDIO está descrito na figuras 2.10 e 2.11. Todo comando utilizando o MDC e o MDIO deve ter pelo menos 33 ciclos de clock, ou seja, há um ciclo adicional do MDC em conjunto com o MDIO enviando um bit 1 no término de cada comando. [1]

Na parte de envio de dados pela MDIO, temos a função de escrita "smi_escrita(int endereco_PHY, int endereco_registrador, int dados)", que recebe como entrada o endereço do PHY, o registrador a ser acessado e os dados a serem enviados. Em seguida, essas entradas são colocadas numa variável do tipo *long* e é feito um laço com 32 iterações que manda os bits da variável por meio da porta MDIO. Simultaneamente, é feito um período do *clock* do MDC a cada iteração.

A função de leitura, chamada "smi_leitura(int endereco_PHY, int endereco_registrador)" recebe duas entradas: o PHY e o registradores a serem acessados, e funciona em duas etapas, como podemos ver na figura 2.11. A função inicia-se enviando dentro de um loop os 14 bits iniciais pela porta MDIO, então se envia um comando para o microcontrolador afim de mudar o status da MDIO para alta impedância, tornando-a assim uma porta de entrada e, novamente dentro de um *loop*, os valores lidos são salvos numa variável. Ao término do recebimento desses 16 bits a função inverte novamente o status da porta MDIO para saída e envia o bit 1 em conjunto com um último ciclo do MDIO.

5.3.3 Controle de Largura de Banda

O controle de largura de banda funciona enviando um comando aos registradores MII 31.26 ao 31.29 para mudar a configuração das portas de 0 a 7. Cada registrador permite alterar a largura de banda de duas portas, tanto no *upload* quanto no *download*. A tabela 5.1 mostra como devem ser escritos os 16 bits de dados para cada registrador afim de controlar a largura de banda.

Tabela 5.1: Configuração de controle de largura de banda.

000: sem limite	100: 1 Mbps
001: 128 kbps	101: 2 Mbps
010: 256 kbps	110: 4 Mbps
011: 512 kbps	111: 8 Mbps

Tomando como exemplo o registrador 31.26, que controla a largura de banda de *upload* e *download* da porta 0 e da porta 1. Os bits [0:2] do registrador correspondem ao *upload* da porta 0, enquanto os bits [6:4] correspondem o *download* da porta 0. E os bits [10:8] correspondem ao *download* da porta 1 e os bits [14:12] ao *upload* da porta 1.

Portanto, seguindo o exemplo do registrador 26, caso se deseje limitar a largura de banda de download da porta 0 para 128 kbps e a largura de banda de *upload* da porta 1 para 256 kbps deve-se ajustar os bits correspondentes para 001 e 010 respectivamente, e o resto deverá ser preenchido com 0. Logo, a configuração desejada será 0010000000010000.

Como há uma possibilidade de haver uma configuração de largura de banda prévia, deve-se fazer uma máscara do dado a ser enviado com o dado atual. Logo, deve-se primeiramente ler qual a configuração atual do registrador 31.26 utilizando a função smi_leitura(31,26) e então fazer a montagem do comando a ser enviado: [14]

```
dado = (smi_leitura(31,26) & 0x2010);
smi_escrita(31,26,dado);
```

Devemos colocar o valor do dado binário encontrado em hexadecimal ou decimal, isso se deve ao fato de que em C os valores devem ser escritos somente em hexadecimal ou decimal. Caso não haja uma configuração anterior no registrador, o "e" lógico efetuado irá apenas copiar o dado desejado.[14]

5.3.4 Criação de VLANs

A implementação da VLAN consiste em ajustar o bit correspondendo ao *VLAN look up table* das portas que se deseja configurar, que são os bits [8:0] no PHY 31 nos registradores [1:8], correspondendo às portas 0 até a MII_EXT. Por exemplo, caso se deseje que a porta 0 seja o servidor, e as outras 7 portas clientes, a configuração para cada porta é:

Tabela 5.2: Configuração do VLAN para a porta 0 como servidor. [1]

	Bit 0	**Bit 1**	**Bit 2**	**Bit 3**	**Bit 4**	**Bit 5**	**Bit 6**	**Bit 7**	**Bit 8**
Registrador 0(Porta 0)	X	1	1	1	1	1	1	1	1
Registrador 1(Porta 1)	X	0	0	0	0	0	0	0	0
Registrador 2(Porta 2)	X	0	0	0	0	0	0	0	0
Registrador 3(Porta 3)	X	0	0	0	0	0	0	0	0
Registrador 4(Porta 4)	X	0	0	0	0	0	0	0	0
Registrador 5(Porta 5)	X	0	0	0	0	0	0	0	0
Registrador 6(Porta 6)	X	0	0	0	0	0	0	0	0
Registrador 7(Porta 7)	X	0	0	0	0	0	0	0	0
Registrador 8(Porta MII_EXT)	1	0	0	0	0	0	0	0	0

Caso se queira que outra porta esteja nessa VLAN, basta trocar o 0 pelo 1 no registrador e no bit correspondente à porta.

5.4 DESABILITAR UMA PORTA

Para desabilitar uma porta, deve-se acessar o PHY da porta que se deseja desabilitar e enviar o valor 1 para o bit 11 do registrador 0. Para isso, assim como no ajuste de largura de banda, deve-se antes de enviar o comando afim de fazer uma máscara com a configuração atual do registrador e o dado que se deseja enviar. Por exemplo, caso se deseje desabilitar a porta 1 (PHY 1), o código deve ser como se segue:

```
dado = (smi_leitura(1,0) & 0x0100);
smi_escrita(1,0,dado);
```

5.4.1 Status das Portas por SMI

A verificação do status das portas por meio da interface SMI consiste na leitura do PHY correspondente à porta e o bit 2 do registrador 1. Portanto, basta fazer uma máscara do bit 2 com o valor recebido de cada porta, como se segue:

```
status = (smi_leitura(PHY,0) & 0x0004);
```

[1] X corresponde a um *Don't Care*.

6 RESULTADOS

As modificações conferidas ao hardware foram limitada a duas intervenções. Em primeiro lugar efetuou-se uma solda de um fio em cada pino de LED do *Switch* para que fosse aferido o status de cada porta. Feito isto, soldou-se um fio a cada um dos dois pinos que davam acesso a interface SMI do *Switch* para que pudesse ser feito a ligação desta interface com o microconlador. Essas alterações não comprometeram o desempenho do *Switch*, que ficou como pode-se ver na figura 6.1

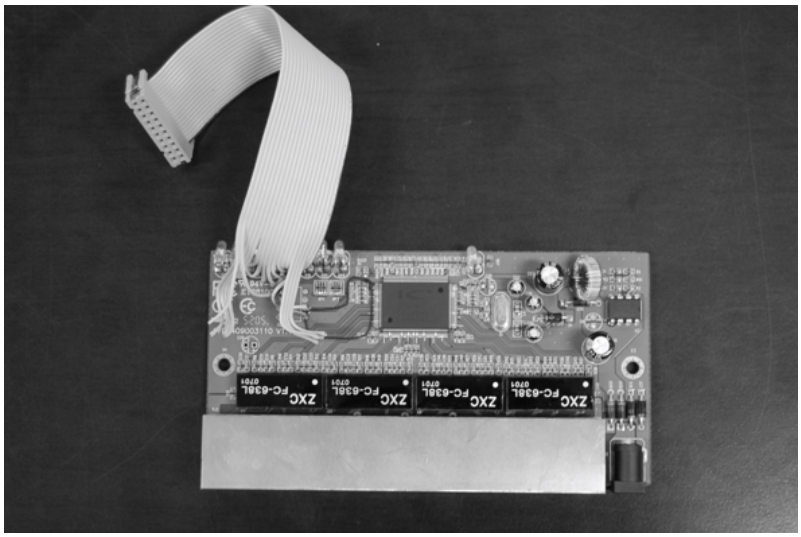

Figura 6.1: *Switch* após as modificações.

Inicialmente conseguiu-se fazer com que o FreeRTOS fosse instalado corretamente no microcontrolador. Com o ajuste do IP deste, no arquivo “FreeRTOSConfig.h” para 169.254.10.10, foi possível acessar o *webserver* de demonstração, que será modificado afim de abrigar a interface de gerenciamento do *Switch*. A figura 6.2 mostra a tela de estatísticas do *webserver* após a instalação.

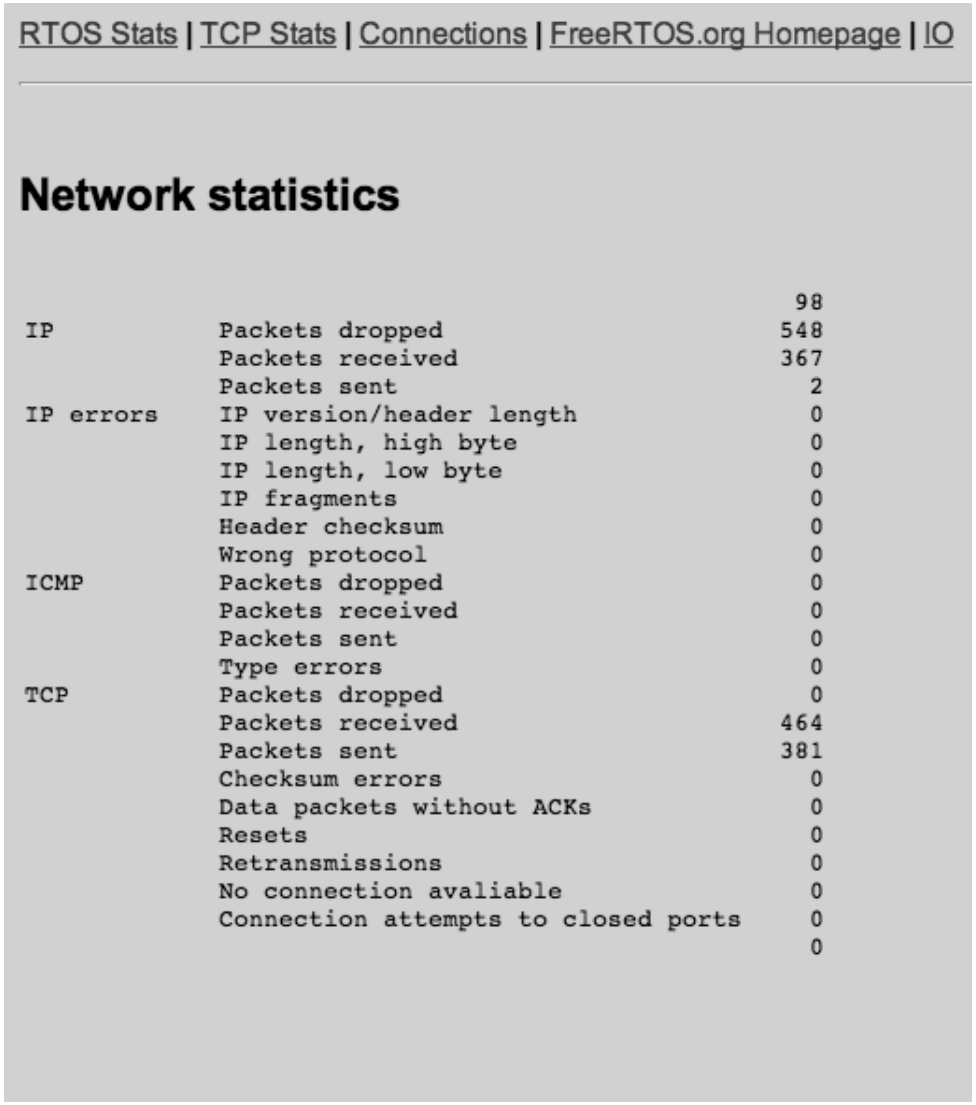

Figura 6.2: Tela de estatísticas do FreeRTOS.

Então, com o intuito de conferir se as soldas para verificar os níveis de tensão dos LEDs estavam corretas, modificou-se a página de IO, que anteriormente servia para modificar o status dos LEDs da placa de desenvolvimento, para mostrar o status de cada porta: a modificação consiste numa tabela com 8 colunas com cada coluna correspondendo ao status uma porta. Caso uma porta estivesse inativa (LED correspondente da porta apagado) a coluna seria preenchida com a cor vermelha, caso contrário com a cor verde. Esse resultado pode ser visto na figura 6.3.

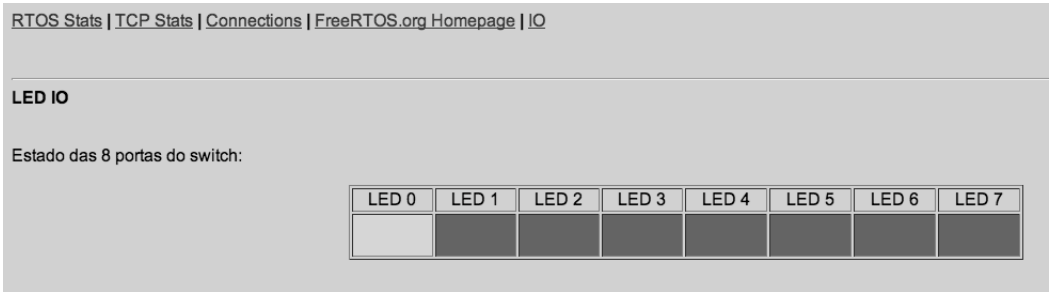

Figura 6.3: Tela de status dos LEDs.

Com o status dos LEDs funcionando, o próximo passo foi um aprofundamento dos estudos do funcionamento da interface SMI e a criação de um código para acessar a SMI do *Switch*. Feito isso, foram feitos testes para verificar o funcionamento de cada função que o *Switch* deveria executar. Primeiramente, testou-se o controle de largura de banda, limitando a velocidade das portas 6 e 7 para 128 kbps. O resultado pode ser visto na figura 6.4 em que temos o mesmo computador ligado no *Switch* sem o controle de largura de banda e com o controle de largura de banda ajustado para 128kbps.

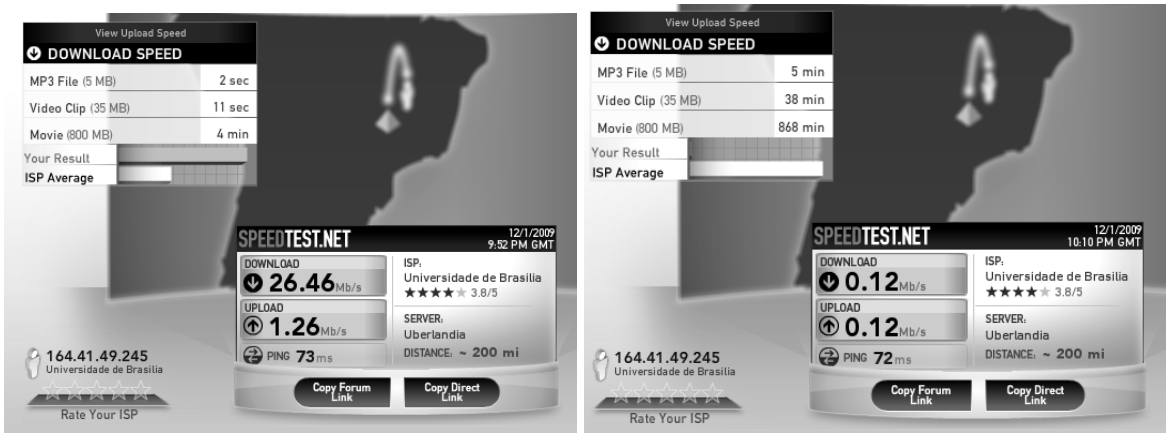

Figura 6.4: Teste do controle de largura de banda.

Encerrado o teste com o controle de largura de banda, concluiu-se que a comunicação da interface SMI funcionava corretamente e então testou-se a configuração do VLAN, que deveria enviar um comando para 7 das 8 portas e então verificou-se que tudo ocorreu como o esperado: as 7 portas escolhidas tinham acesso somente à porta do servidor e o servidor tinha acesso a todas as outras 7 portas.

Em seguida, após um estudo mais aprofundado do *datasheet* do *Switch*, verificou-se que é possível verificar o status das portas acessando registradores específicos no PHY de cada porta, como foi explicado no capítulo 5, testou-se então como ficaria a tabela feita anteriormente com essa nova configuração e a conclusão é que não houve mudança, o que facilitará na produção do produto final, pois serão necessárias menos soldas no *Switch*. A partir daí testou-se outras funções mais simples, tais como um *soft reset* do

Switch e desabilitar portas, todas com sucesso.

7 CONCLUSÕES

Este documento sintetiza o trabalho realizado ao longo do último ano com o intuito de se estabelecer o gerenciamento de um *Switch Unmanaged* por meio de um microcontrolador.

Muito tempo foi gasto para que os desenvolvedores se familiarizassem com o ambiente de desenvolvimento e padronizassem um modelo para os códigos. Isto porque havia conjunto extenso de aplicativos que eram utilizados para o estabelecimento da comunicação entre o computador e o microcontrolador e, além disto, foram testados alguns *firmwares* para o microcontrolador até se decidisse pela utilização do FreeRTOS.

A partir do momento em que foi estabelecida a comunicação com o microcontrolador passou-se a pesquisar as características do *Switch* e seu funcionamento. Com isso, foi possível determinar quais melhorias poderiam ser implementadas e o qual o melhor modo para que isso fosse feito.

Finalmente foi possível programar, por meio do Eclipse, as funções de: desativação de porta; controle de banda e estabelecimento de VLANs. Com isso foi possível implementar, com sucesso, gerenciamento a operação do *Switch*.

Como continuação deste projeto, poderá ser desenvolvida uma interface WEB que permita a configuração direta de cada funcionalidade. Além disto, poderá ser implementado um *WatchDog* no microcontrolador que seja capaz de monitorar a operação do *Switch* e efetuar o seu *Reset*, quando necessário. Também deve-se implementar uma maneira de atualizar o *firmware* do microcontrolador por meio do *webserver*. Por fim, deverá ser estudado um novo *layout* para a placa do *Switch* fazendo com que ela seja capaz de comportar o microcontrolador e tenha um acabamento adequado às necessidades do mercado utilizando um modelo de *Switch* de 16 portas.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] IP178C/IP178C LF/IP178CH/IP178CH LF Datasheet. [S.l.], 2008.

[2] AT 91 ARM Thumb-based Microcontrollers. [S.l.], 2009.

[3] D'ERCOLE, R. Dinheiro a conta-gotas. Pequenas empresas grandes negocios, n. 110, ano X.

[4] DAY, J.; ZIMMERMANN, H. The OSI reference model. *Proceedings of the IEEE*, v. 71, n. 12, p. 1334–1340, 1983.

[5] SWITCH/REPEATER. [S.l.]: National Institute of Standards and Technology. `http://ieee1588.nist.gov/switch.htm`.

[6] HUTCHINSON, D. *Local area network architectures*. [S.l.]: Addison-Wesley, 1989.

[7] HEIN, M.; GRIFFITHS, D. *Switching Technology in the Local Network*. [S.l.]: Itp New Media, 2000.

[8] METCALFE, R.; BOGGS, D. Ethernet: Distributed packet switching for local computer networks. ACM New York, NY, USA, 1976.

[9] ARM7TDMI Technical Reference Manual. [S.l.], 2001.

[10] BARR, M.; MASSA, A. *Programming embedded systems: with C and GNU development tools*. [S.l.]: O'Reilly Media, Inc., 2006.

[11] SLOSS, A. N.; SYMES, D.; WRIGHT, C. *ARM system developer's guide: designing and optimizing system software*. [S.l.]: Elsevier, 2000.

[12] LYNCH, J. Using Open Source Tools for AT91SAM7S Cross Development Revision. Citeseer.

[13] BARRY, R. *FreeRTOS Reference Manual*. [S.l.], 2009.

[14] KERNIGHAN, B. W.; RITCHIE, D. M. *C Programming Language*. [S.l.]: Prentice Hall PTR; 2 edition, 1988.

ANEXOS

I. DIAGRAMAS ESQUEMÁTICOS

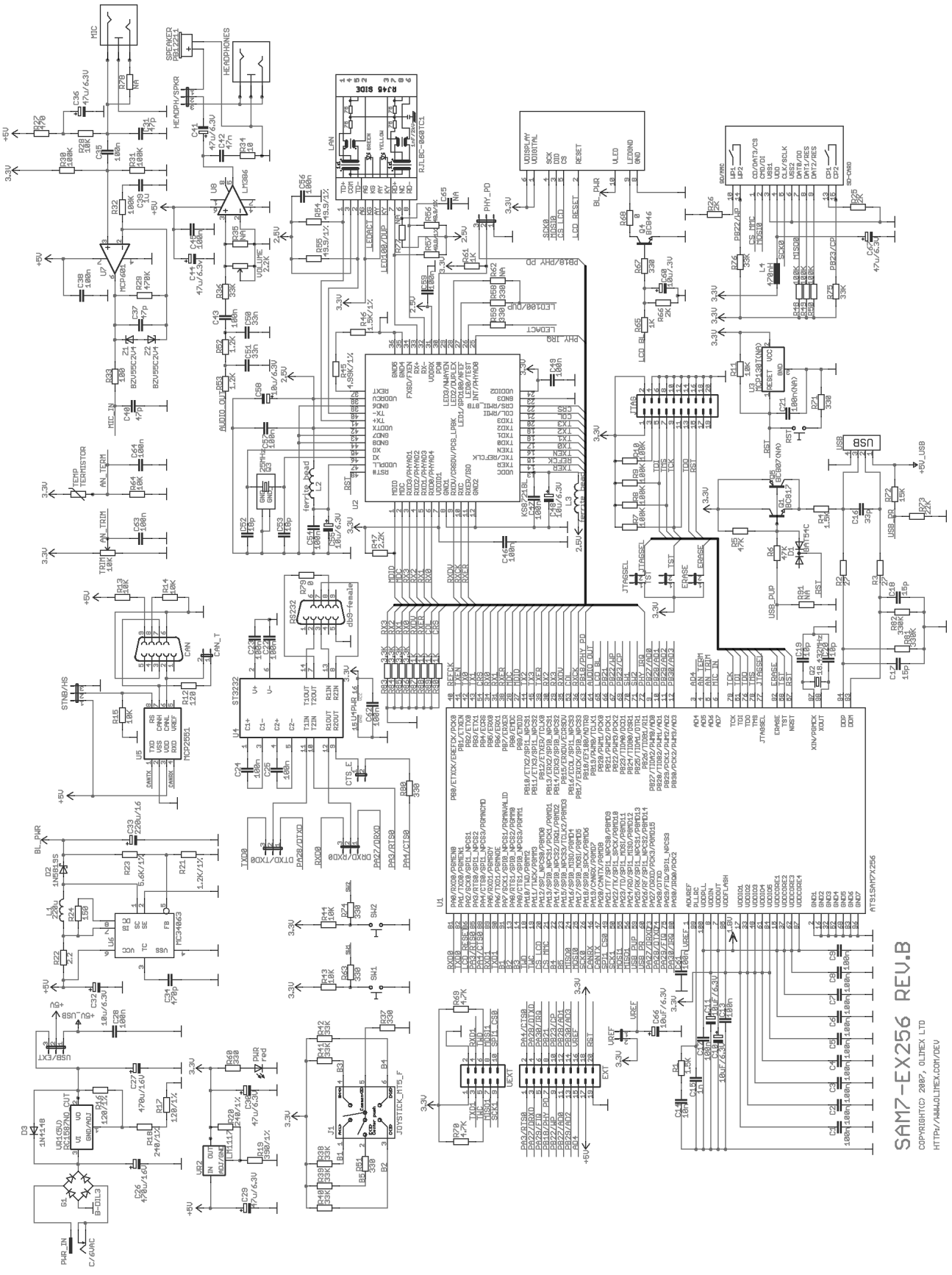

Figura I.1: Esquemático da placa de desenvolvimento.

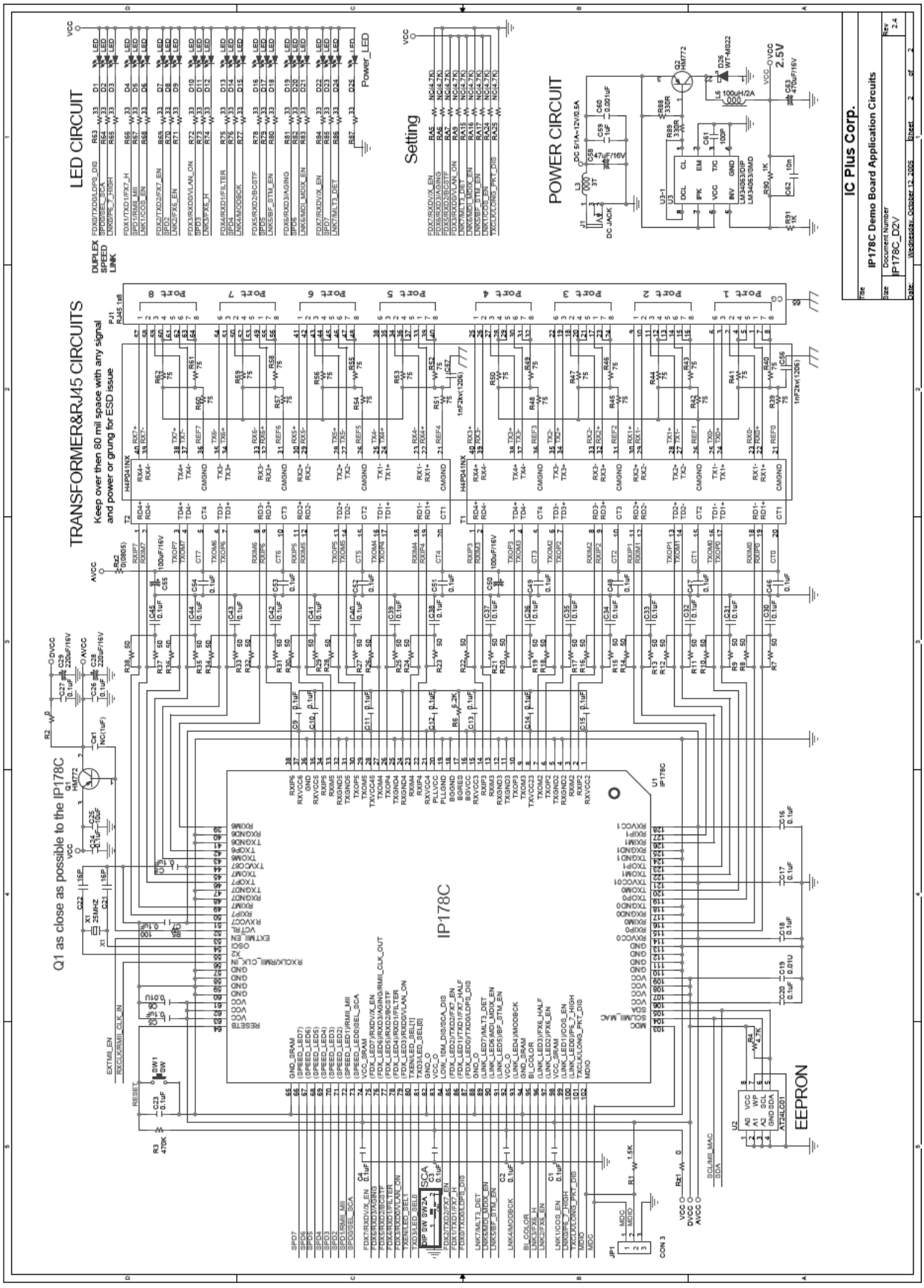

Figura I.2: Esquemático do Switch.

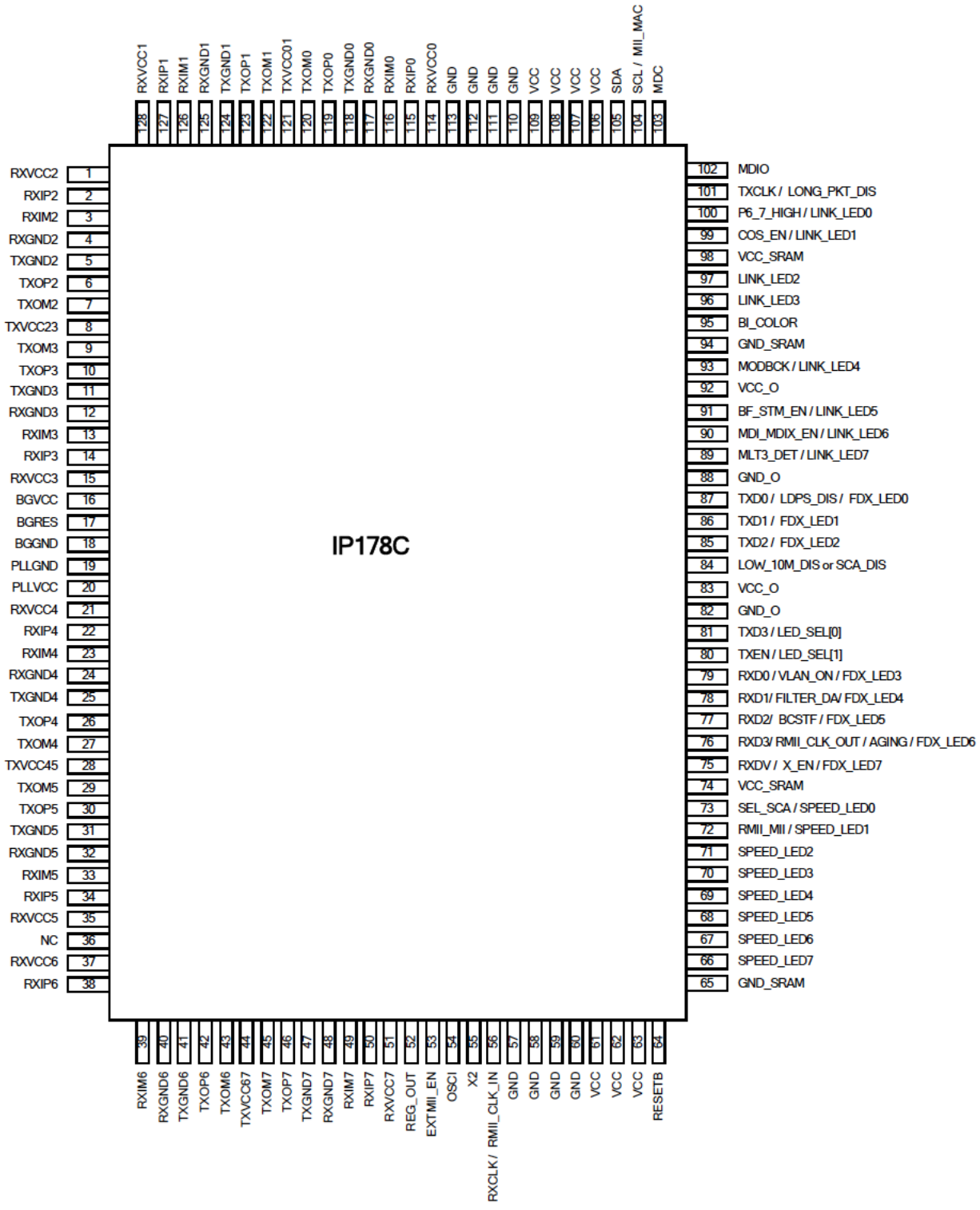

Figura I.3: Diagrama de pinos do Switch.