

## Sumário

### **Conteúdo**

1. Introdução .....	3
2. O Refrigerador .....	4
2.1. Dados Técnicos .....	4
2.2. Funcionamento do refrigerador Cycle Defrost .....	5
3. O Sistema “ON-OFF” .....	7
4. Testes Iniciais.....	7
4.1. Mapeamento da Temperatura Interna do Refrigerador .....	7
4.2. Teste de Ciclo em Vazio.....	7
4.3. Picos de Corrente .....	17
4.4. Ensaio de Consumo e Cargas Térmicas.....	21
5. Inversor de Frequência .....	27
5.1. Introdução .....	27
5.2. Blocos Componentes do Inversor de Frequência.....	28
5.2.1. CPU – Unidade Central de Processamento .....	28
5.2.2. IHM Interface Homem/Máquina .....	29
5.2.3. Interfaces .....	29
5.2.4. Bloco de Potência .....	29
5.2.5. Chaveamento .....	30
5.3. Modulação por Largura de Pulso.....	34
5.4. Classificação dos inversores de frequência.....	38
5.4.1. Região de Enfraquecimento.....	39
5.4.2. Características do Controle Escalar .....	40
6. <i>Hardware e Software</i> .....	40
6.1. <i>Hardware e Seu Desenvolvimento</i> .....	40

6.1.1. Fontes .....	40
6.1.2. Transdutor de Temperatura.....	45
6.1.3. INTERFACE COM USUÁRIO (IHM).....	48
6.1.4. Microcontrolador (MCU) .....	49
6.1.5. INVERSOR DE FREQUÊNCIA .....	50
6.1.6. Desenvolvimento da Placa de Circuito Impresso (PCI) .....	55
6.2. <i>Software</i> e Seu Desenvolvimento .....	58
6.2.1. Descrição dos Arquivos.....	58
6.2.2. Detalhamento das Principais Funções .....	62
7. Testes de validação de <i>Hardware</i> .....	79
8. Levantamento de Custo do <i>Hardware</i> .....	83
9. Conclusão.....	85
REFERÊNCIAS.....	86
ANEXO A – FLUXOGRAMA PWM.....	88
ANEXO B – HARDWARE 1 .....	89
ANEXO C – HARDWARE 2 .....	90
ANEXO D – CIRCUITO CPU.....	91
ANEXO E – OSBDM (Open Source BDM). ....	92

## 1. Introdução

Os refrigeradores domésticos convencionais apresentam um sistema de funcionamento singularmente simples, onde uma temperatura de operação é selecionada e um termostato é responsável por desativar o circuito elétrico de alimentação do motor do compressor, no momento em que essa temperatura é atingida internamente. Após o aumento da temperatura interna do refrigerador, novamente o compressor é acionado refrigerando seu interior até a temperatura selecionada, gerando um ciclo.

Apesar de esse sistema ser funcional, o mesmo apresenta algumas características negativas. Entre elas está o alto consumo de energia elétrica proveniente dos picos de corrente oriundos do acionamento do compressor.

Esse Trabalho de Conclusão de Curso propõe implementar um sistema de controle mais eficiente para o funcionamento do refrigerador através de um hardware e um software desenvolvidos para minimizar os picos de corrente do compressor, controlando sua partida e determinando sua velocidade de rotação pelo monitoramento da temperatura interna do refrigerador.

Em seguida através de testes, pretende-se comprovar o objetivo do projeto: verificar se há redução no consumo de energia com o novo sistema empregado.

Atualmente, apenas os refrigeradores de alto custo possuem esse sistema de controle. Entretanto ele será introduzido no refrigerador *Cycle Defrost*, um modelo de refrigerador mais simples e de baixo custo.

## 2. O Refrigerador

### 2.1. Dados Técnicos

Motivo da Escolha pelo Produto: Simples estrutura física sendo composto de apenas uma porta e de menor potência.

O refrigerador utilizado para esse projeto possui as seguintes especificações:

Tabela1 – Dados gerais do refrigerador *Cycle Defrost*.

Dados Gerais		
Modelo		RUCT280
Fabricante		Mabe Hortolândia Eletrodoméstico Ltda
Marca		Continental
Categoria		Refrigerador
Sistema de degelo		Manual
Agente de expansão de espuma		Ciclo pentano
Compressor	Marca	Embraco
	Modelo	EMX20CLC
	Capacidade (BTU/h)	262
Fluido Refrigerante	Tipo	R600a
	Quantidade (g)	30
	Classificação do congelador	1 estrela
Volume interno	Refrigerador	223
	Compartimento 1 estrela	29
	Total	252
Consumo de energia (KWh/Mês)		15,8
Classe de eficiência energética		A

Fonte: Documento interno Mabe (PET).

Ilustração 1 – Visão geral do refrigerador.



Fonte: produção do próprio autor.

## 2.2. Funcionamento do refrigerador Cycle Defrost

O refrigerador *Cycle Defrost* apresenta um funcionamento simples onde o seu compressor possui duas funções, sendo elas: comprimir e succionar o gás.

No primeiro instante o compressor eleva a pressão do gás refrigerante o que ocasiona o aumento da temperatura do gás. Tal ação é necessária para que quando o gás refrigerante chegue à válvula de expansão o mesmo esteja com uma pressão elevada.

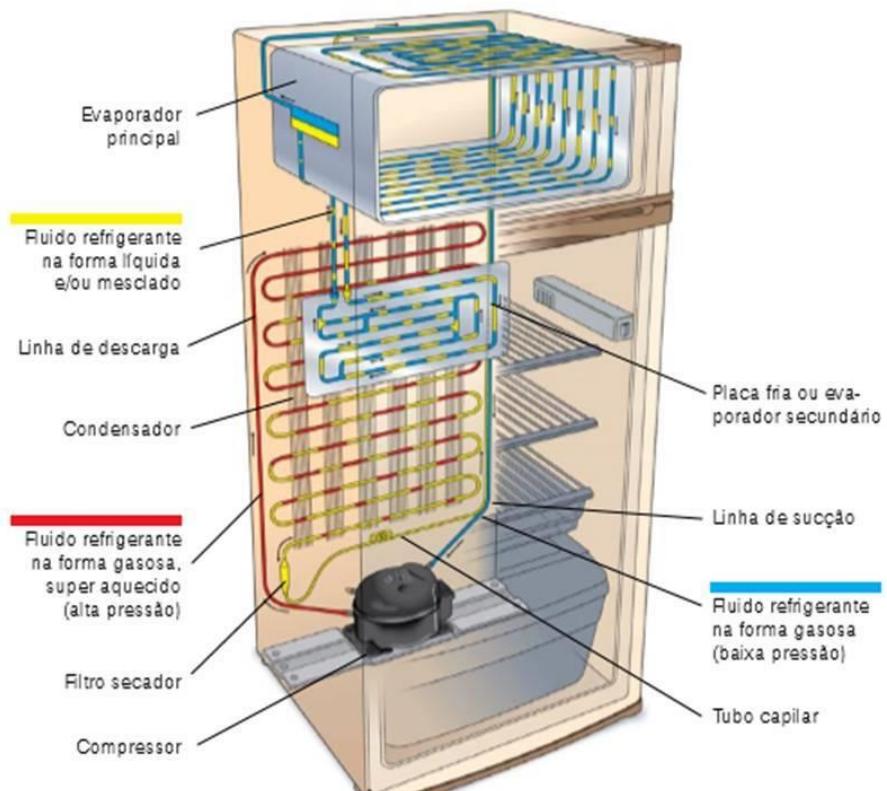
Após essa compressão o gás passa para o condensador que tem a finalidade de trocar calor com o meio ambiente e consequentemente diminuir sua pressão. Depois dessa troca de calor o gás passa através de um filtro secador que retira a umidade do sistema hermético e em seguida vai para a válvula de expansão (tubo capilar – nome dado ao fato do tubo ter o diâmetro interno do tamanho de um fio de cabelo humano),

para diminuir bruscamente sua pressão e consequentemente sua temperatura instantânea de evaporação (Stoecker & Saiz Jabardo, 1994).

Depois desse processo de expansão o gás desloca-se para o evaporador primário, onde é “aproveitado o máximo frio” e isto posto se obtém temperaturas negativas. É importante ressaltar que o valor dessa temperatura depende diretamente do sistema hermético e do compressor utilizado.

Como a maior energia já foi utilizada no processo de expansão no evaporador primário, o gás refrigerante encaminha-se para o evaporador secundário, pois nessa região não é necessário gerar mais frio e sim somente aproveitar o restante de energia no sistema. Após esse processo o gás é succionado novamente pelo compressor e o processo volta a se repetir.

Ilustração 2 – Funcionamento do sistema hermético do refrigerador *Cycle Defrost*.



Fonte: Documento interno MABE.

### 3. O Sistema “ON-OFF”

Atualmente os refrigeradores convencionais possuem um sistema de controle de temperatura chamado “*ON-OFF*”. Esse controle consiste em um botão de regulagem externo (para esse modelo de refrigerador, pois em alguns modelos pode ser interno), que permite selecionar a temperatura de operação do sistema. Um termostato é responsável por desativar o circuito de alimentação do motor do compressor no exato momento em que essa temperatura é atingida.

Em seguida o refrigerador começa a perder temperatura em seu interior por efeito da absorção de energia do ambiente por calor e o termostato, por sua vez, aciona novamente o compressor ocasionando a refrigeração do gabinete, gerando um ciclo.

## 4. Testes Iniciais

### 4.1. Mapeamento da Temperatura Interna do Refrigerador

Inicialmente foi realizado o mapeamento das temperaturas internas do refrigerador através de termopares (tipo “T”), estrategicamente distribuídos em alguns pontos de sua superfície interna. O objetivo desse ensaio é de compreender a resultante das temperaturas do refrigerador em regime permanente.

Através da análise de todos os resultados obtidos dos testes poderá então, definir a posição do sensor de temperatura que fará a função do termostato.

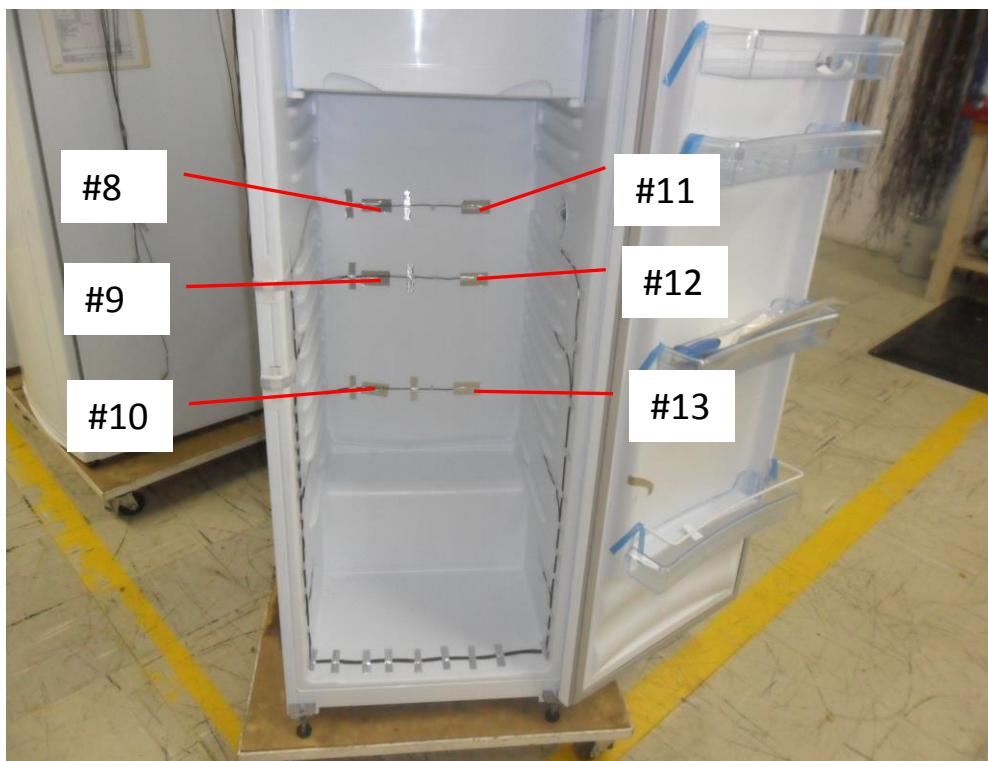
### 4.2. Teste de Ciclo em Vazio

O Teste de Ciclo em Vazio é utilizado com a finalidade de determinar a faixa total de funcionamento do produto e juntamente a porcentagem de marcha do compressor. Em conjunto, através do mesmo irá se obter a curva da temperatura de cada termopar.

Esse teste é realizado com o refrigerador desprovido de carga térmica e para realização foram utilizados 20 termopares, cada qual com sua posição definida e nomenclatura. Abaixo segue a descrição de como o teste foi realizado com o termostato regulado nas posições máxima, média e mínima. Para esse ensaio a temperatura

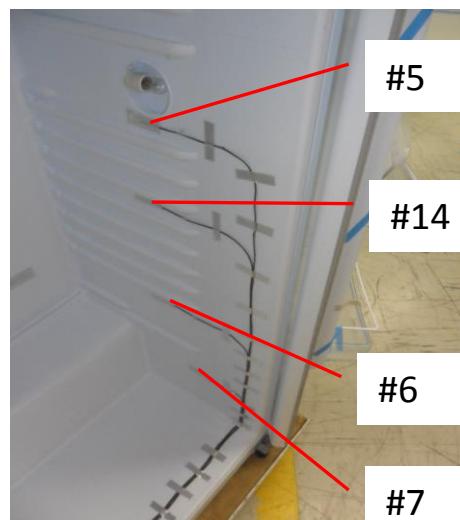
ambiente foi de 32°C como base de estudo referente ao teste de consumo conforme norma ISO8561:1995.

Ilustração 3 – Visão frontal do posicionamento dos termopares.



Fonte: produção do próprio autor.

Ilustração 4 – Visão lateral do posicionamento dos termopares.



Fonte: produção do próprio autor.

Ilustração 5 – Visão lateral do posicionamento dos termopares.



Fonte: produção do próprio autor.

Ilustração 6 – Visão frontal do posicionamento dos termopares.



Fonte: produção do próprio autor.

Tabela 2 – Nomenclatura dos Termopares.

Termopares		
Numero	Nomenclatura	Descrição
1	TP_E_S	Parede esquerda superior
2	TP_E_M1	Parede esquerda meio1
3	TP_E_M2	parede esquerda meio2
4	TP_E_I	Parede esquerda inferior
5	TP_D_S	Parede direita superior
6	TP_D_M2	Parede direita meio2
7	TP_D_I	Parede direita inferior
8	TP_T_E_S	Parede traseira esquerda superior
9	TP_T_E_M	Parede traseira esquerda meio
10	TP_T_E_I	Parede traseira esquerda inferior
11	TP_T_E_S	Parede traseira direita superior
12	TP_T_E_M	Parede traseira direita meio
13	TP_T_E_I	Parede traseira direita inferior
14	TP_D_M1	Parede direita meio1
15	TV	Centro geométrico do freezer
16	TKF1	Centro geométrico da primeira prateleira
17	TKF2	Centro geométrico da segunda prateleira
18	TKF3	Centro geométrico da terceira prateleira
19	TCM	Centro geométrico da gaveta de legumes
20	TR	Sensor externo na porta

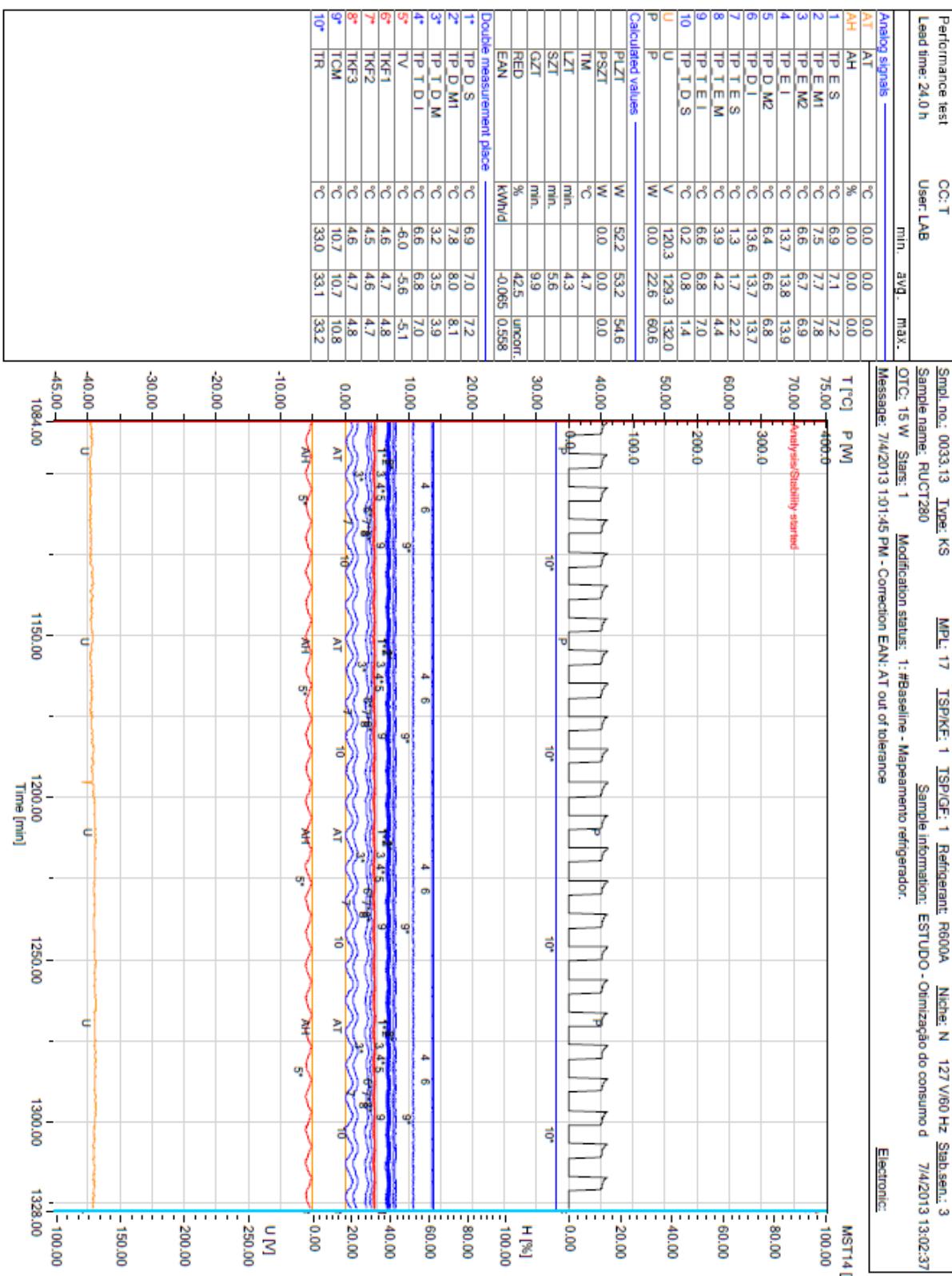
Conforme norma  
ISO 8561:1995

Fonte: produção do próprio autor.

Após a distribuição dos termopares ao longo da superfície interna do refrigerador, o mesmo foi submetido a vários testes para “mapear” as temperaturas de seu interior. O primeiro teste foi realizado parametrizando o termostato na posição mínima, o segundo na posição média e o terceiro na posição máxima.

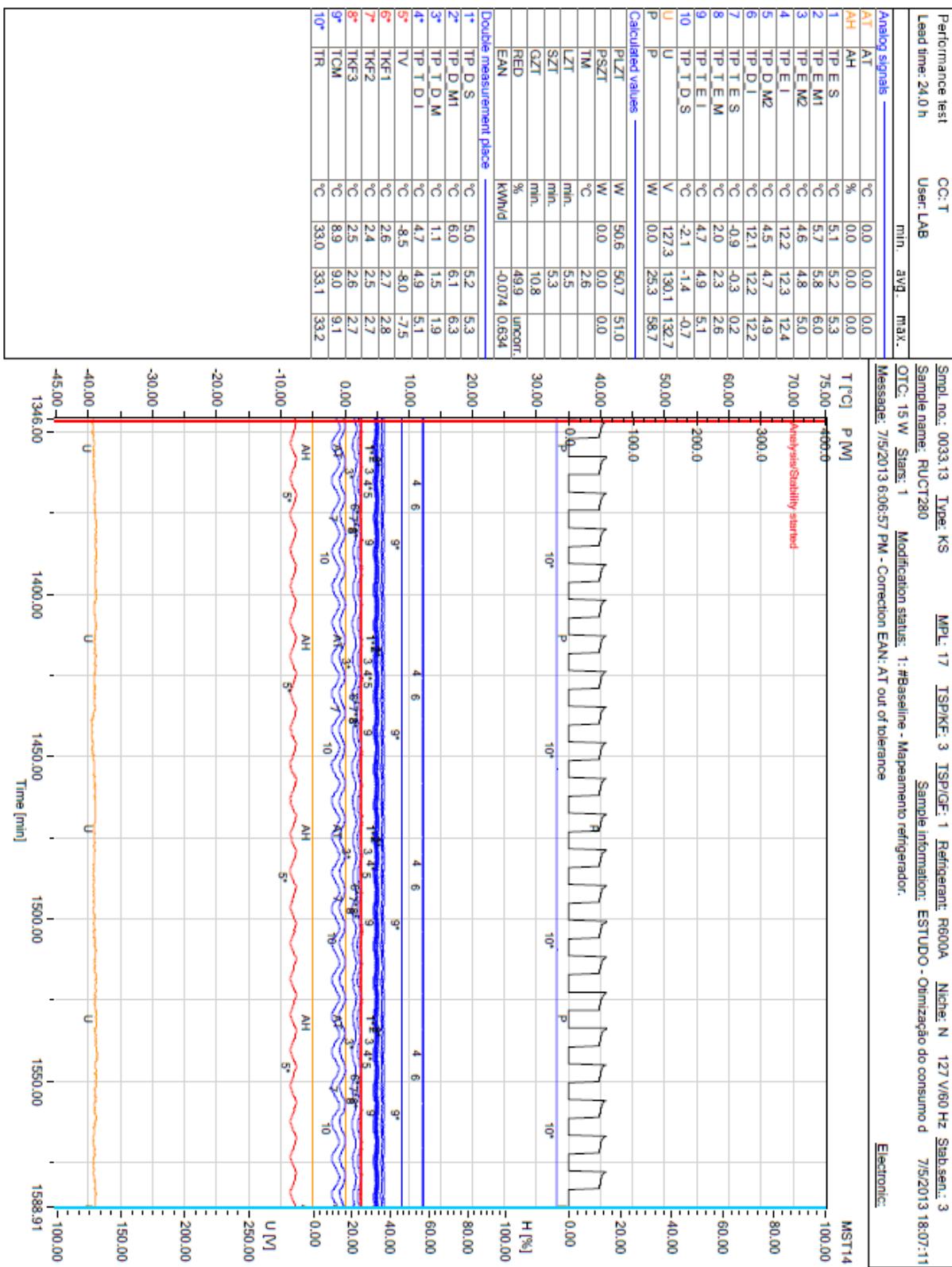
Os resultados desses testes foram obtidos através do software SIGMADATA.

Ilustração 7 - Funcionamento do produto com a posição mínima do termostato.



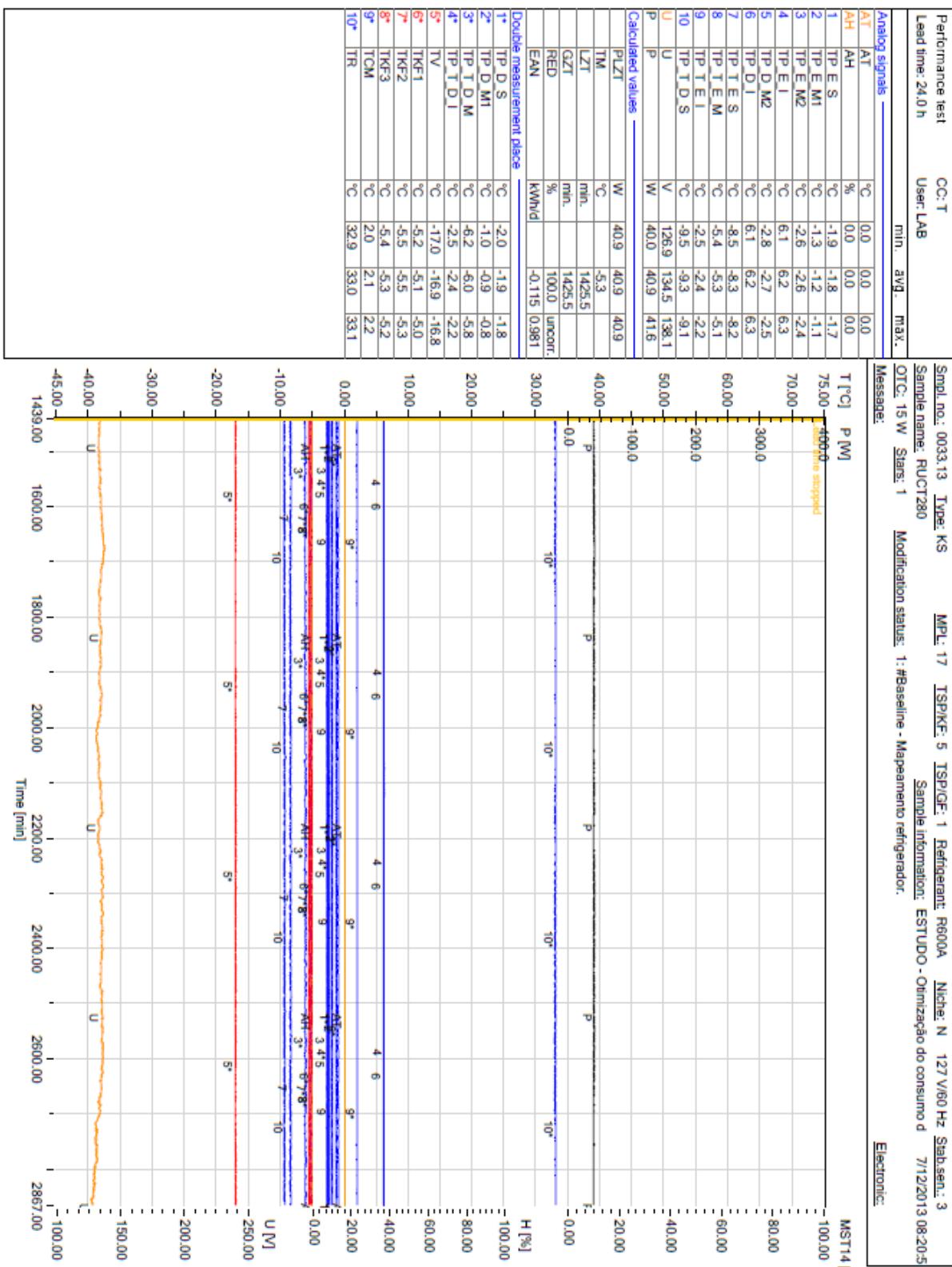
Fonte: Software Sigmadata.

Ilustração 8 - Funcionamento do produto com a posição média do termostato.



Fonte: Software Sigmadata.

Ilustração 9 - Funcionamento do produto com a posição média do termostato.



Fonte: Software Sigmadata.

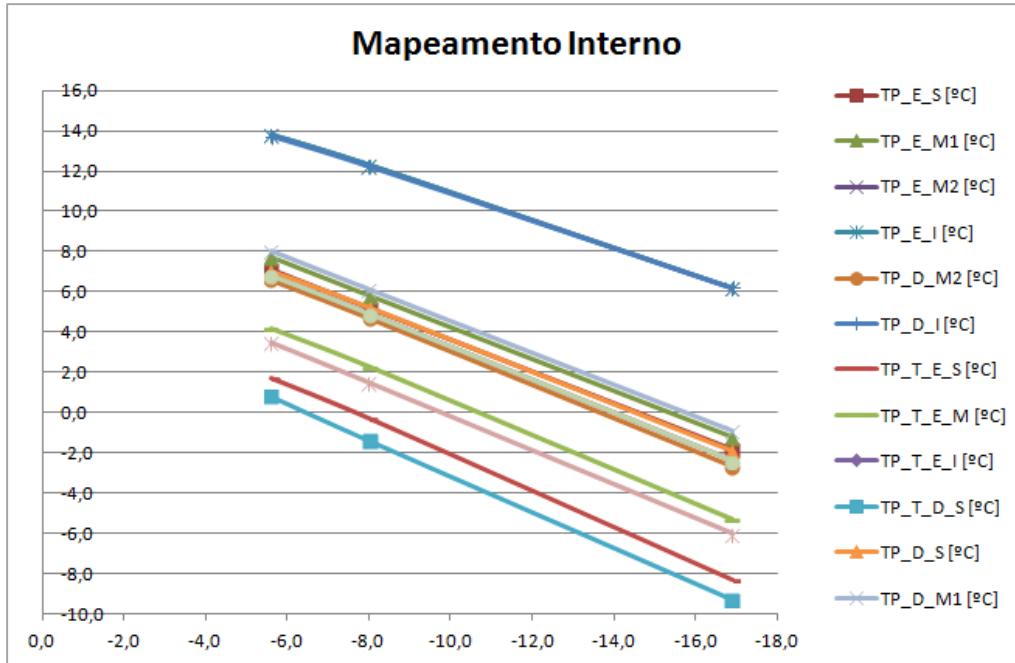
Devido aos resultados obtidos em todo o *range* de funcionamento do termostato, foi possível fazer uma curva das temperaturas em cada ponto mapeado em relação da temperatura do freezer e também escolher uma posição para o sensor temperatura que substituirá o termostato.

Tabela 3 – Mapeamento das temperaturas para cada termopar.

<b><i>Baseline</i></b>				
<b>Numero</b>	<b>Nomenclaturas</b>	<b>Posição Termostato</b>		
		<b>Máximo</b>	<b>Médio</b>	<b>Mínimo</b>
#15	TV [°C]	-16,9	-8,0	-5,6
#16	TKF1 [°C]	-5,1	2,7	4,7
#17	TKF2 [°C]	-5,5	2,5	4,6
#18	TKF3 [°C]	-5,3	2,6	4,7
--	TM [°C]	-5,3	2,6	4,7
#19	TCM [°C]	2,1	9,0	10,7
#1	TP_E_S [°C]	-1,8	5,2	7,1
#2	TP_E_M1 [°C]	-1,2	5,8	7,7
#3	TP_E_M2 [°C]	-2,5	4,8	6,7
#4	TP_E_I [°C]	6,2	12,3	13,8
#6	TP_D_M2 [°C]	-2,7	4,7	6,6
#7	TP_D_I [°C]	6,2	12,2	13,7
#8	TP_T_E_S [°C]	-8,3	-0,3	1,7
#9	TP_T_E_M [°C]	-5,3	2,3	4,2
#10	TP_T_E_I [°C]	-2,4	4,9	6,8
#11	TP_T_D_S [°C]	-9,3	-1,4	0,8
#5	TP_D_S [°C]	-1,9	5,2	7,0
#14	TP_D_M1 [°C]	-0,9	6,1	8,0
#12	TP_T_D_M [°C]	-6,0	1,5	3,5
#13	TP_T_D_I [°C]	-2,4	4,9	6,8
--	Potência [W]	40,8	25,3	22,6
--	Run Time [%]	100,0	49,9	42,4
--	EAN [kWh/d]	0,979	0,634	0,557

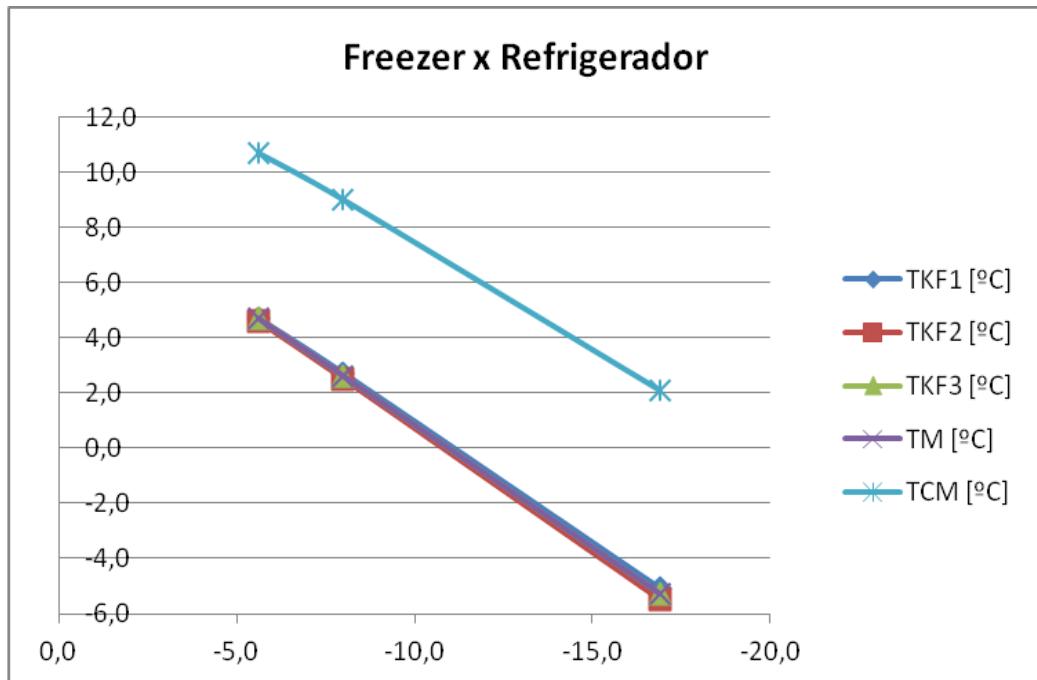
Fonte: *Software Sigmadata*.

Ilustração 10 – Mapeamento Interno.



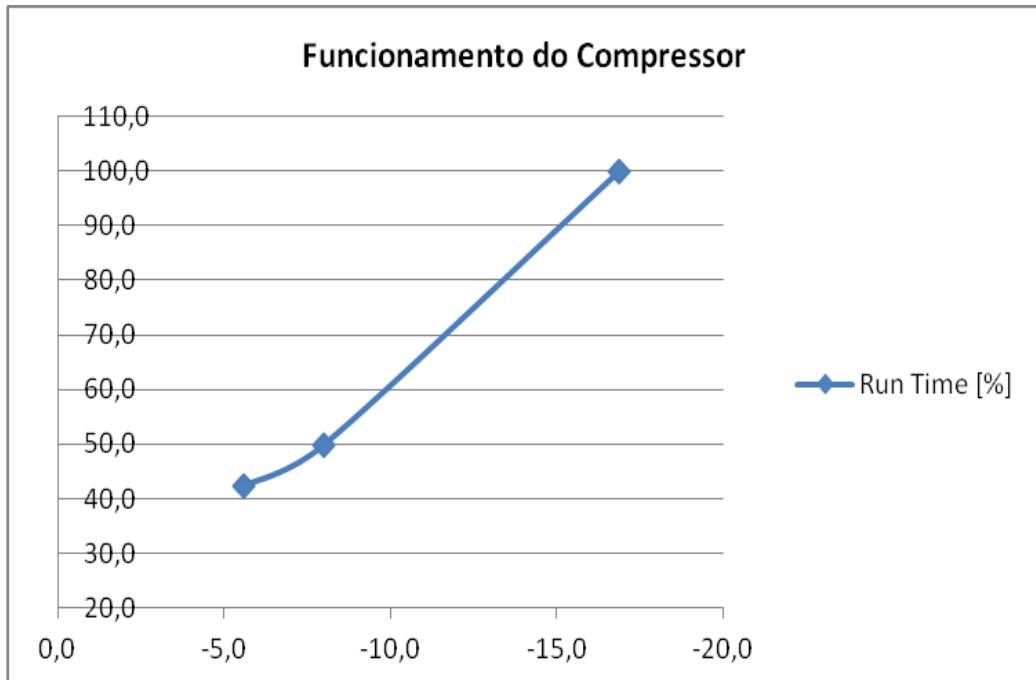
Fonte: *Software Sigmadata*.

Ilustração 11 – Faixa de Trabalho.



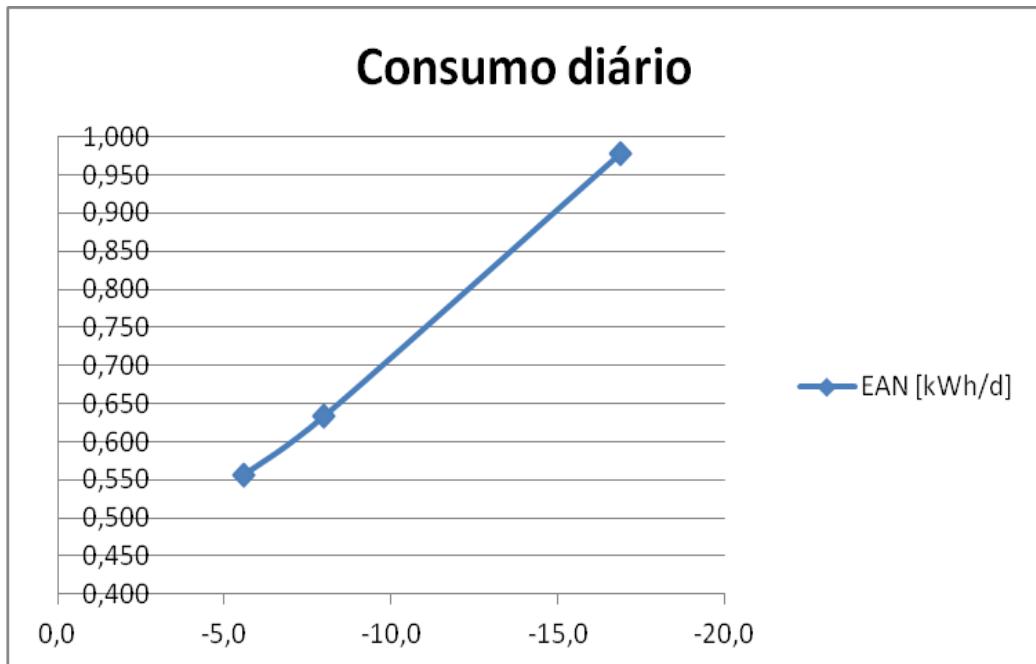
Fonte: *Software Sigmadata*.

Ilustração 12 – Porcentagem de funcionamento do compressor relacionado a cada posição do termostato.



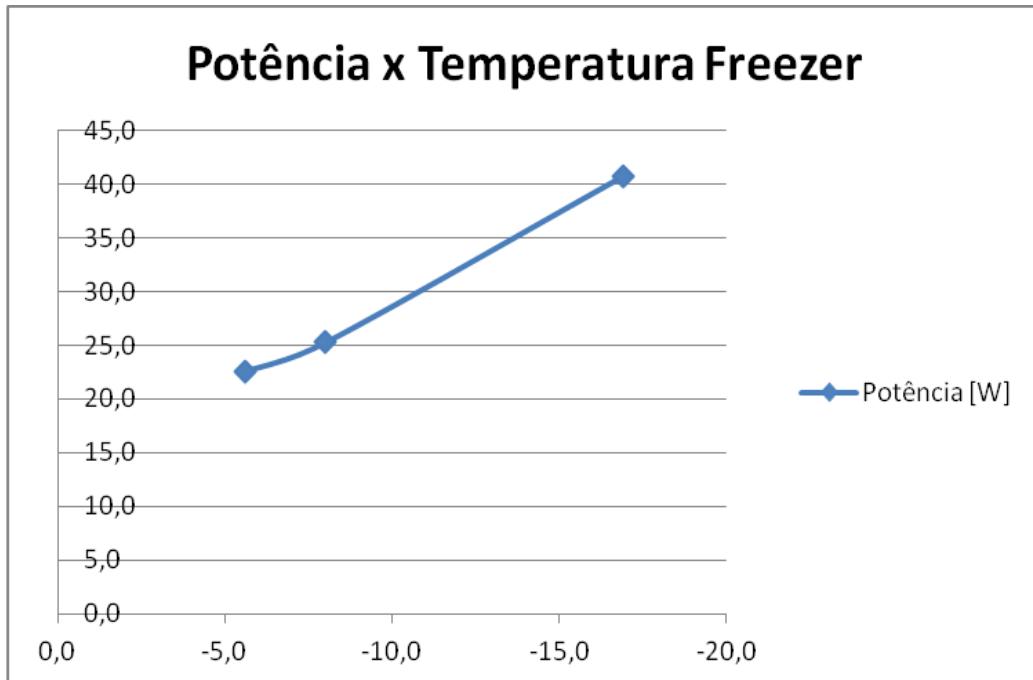
Fonte: *Software Sigmadata*.

Ilustração 13 – Consumo diário do produto.



Fonte: *Software Sigmadata*.

Ilustração 14 – Potência do compressor relacionado a cada posição do termostato.



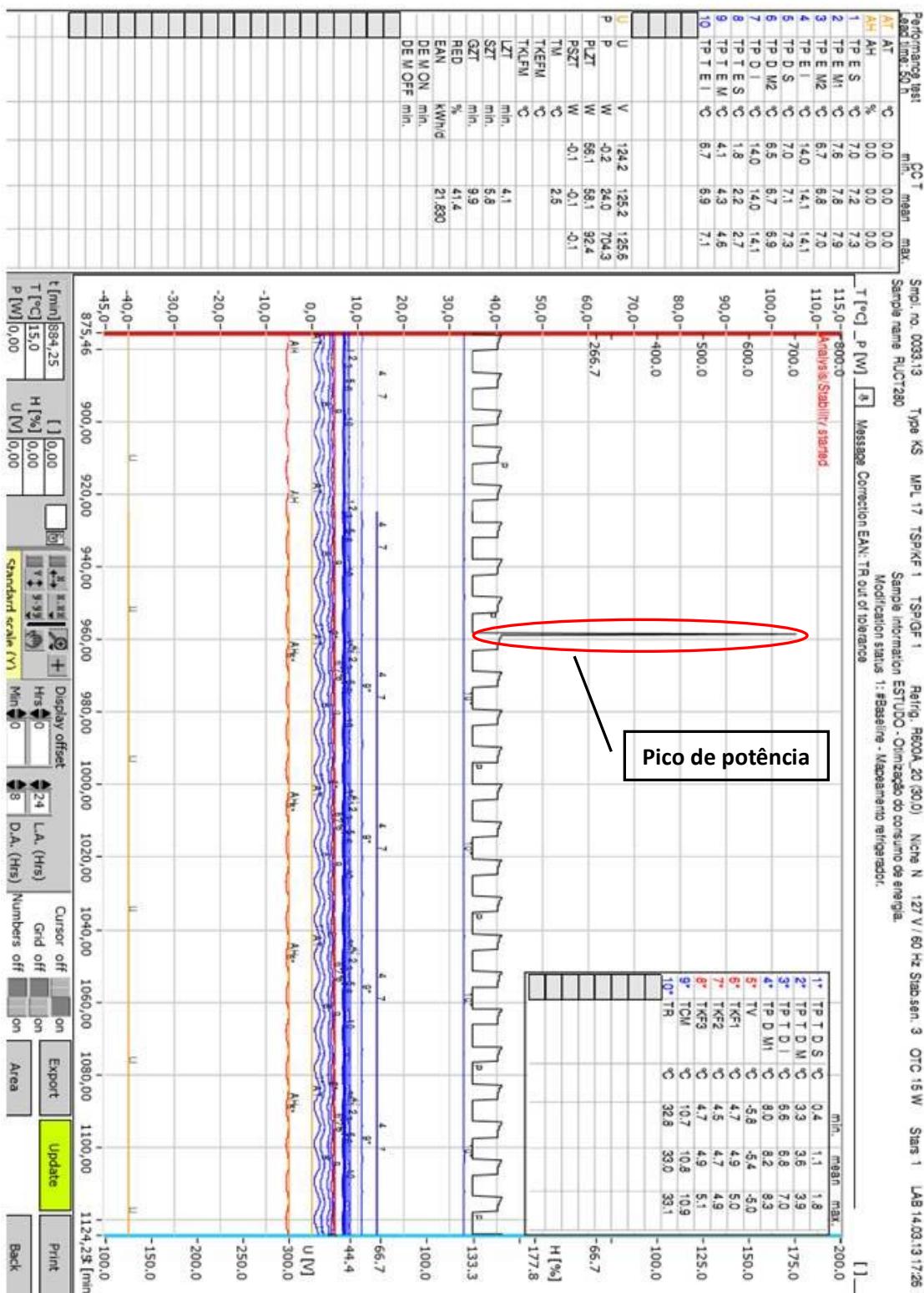
Fonte: *Software Sigmadata*.

Através dos gráficos, a posição do sensor de temperatura escolhida foi a de número #13 (TP\_T\_D\_I), por não ser o ponto mais frio do refrigerador que poderia ter como consequência deixar o refrigerador com temperaturas muito elevadas, e também por não ser o ponto mais quente que teria como consequência o inverso, deixar as temperaturas restantes negativas.

#### 4.3. Picos de Corrente

O gráfico abaixo representa um dos principais problemas relacionados ao sistema ON-OFF, os picos de corrente, devido à partida direta do compressor (Franchi, 2010).

Ilustração 15 – Funcionamento do refrigerador com a posição mínima do termostato.



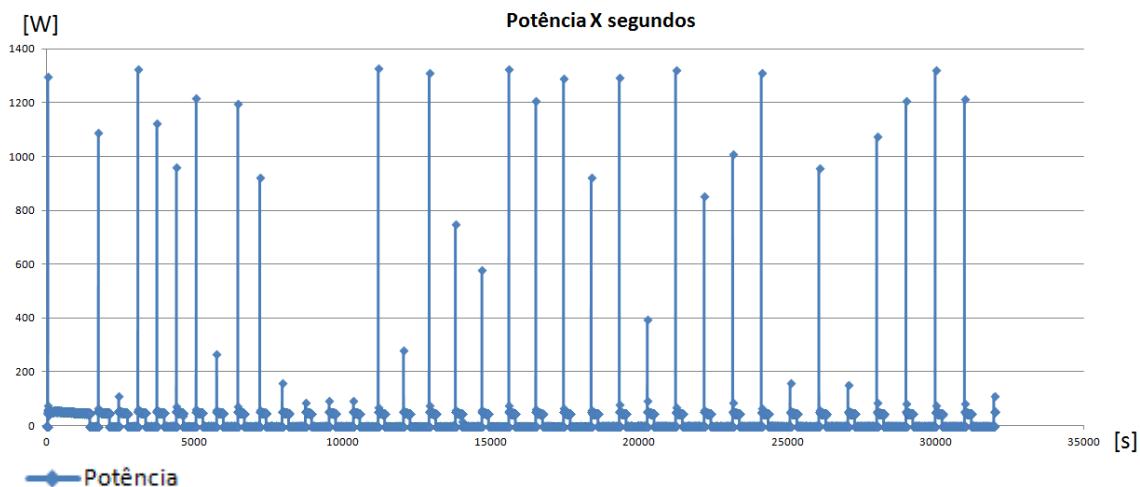
Fonte: Software Sigmadata.

Com o intuito de conhecer melhor o funcionamento do produto, outro sistema de aquisição de dados foi utilizado para realização dos testes, denominado FLUKE.

Esse sistema não necessita especificamente de uma temperatura ambiente controlada e estabilizada, pois somente será estudado em que frequência ocorre os picos de corrente de partida do compressor. Através dele é possível obter um número de amostras em uma faixa de tempo. O sistema anterior de aquisição de dados (SIGMADATA) era atualizado a cada 25s enquanto o sistema FLUKE permite ao usuário determinar o tempo em que o mesmo será atualizado. Para esse ensaio foi determinado o tempo de atualização de 1s.

O refrigerador foi submetido a esse sistema em aproximadamente 24h, isso gerou 60416 amostras que foram convertidas em três gráficos:

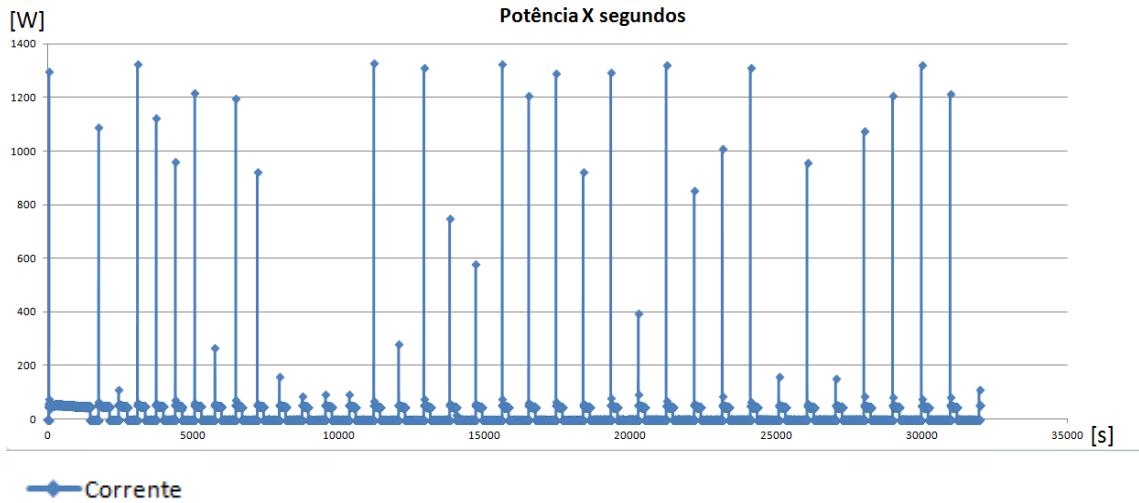
Ilustração 16 – Potência em função do tempo.



Fonte: *Software Fluke*.

Através do gráfico acima é possível concluir que os picos de corrente são gerados frequentemente ocasionando os picos de potência.

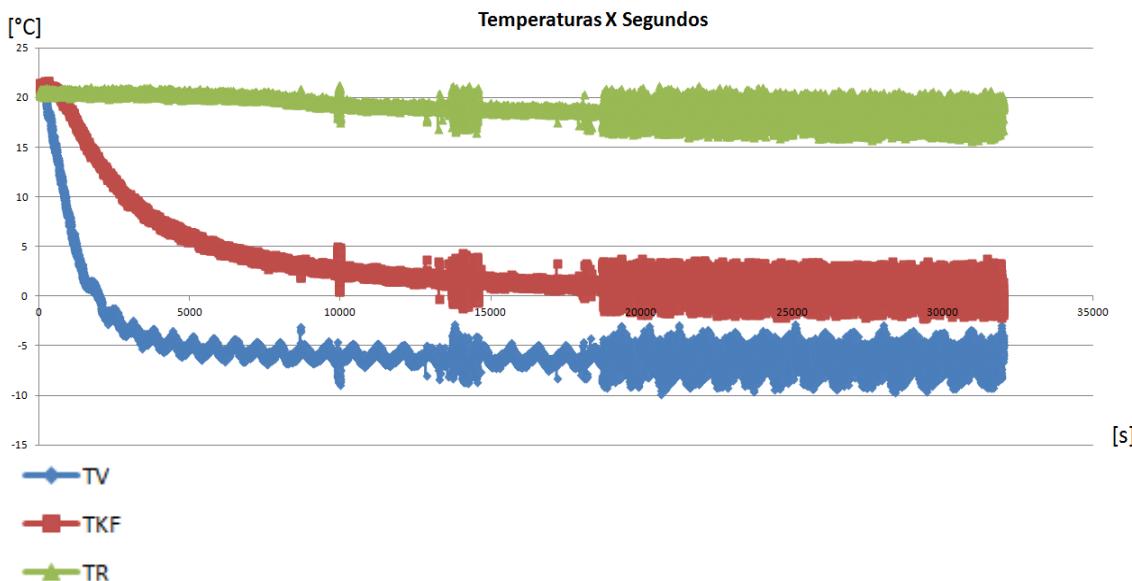
Ilustração 17 – Corrente em relação ao tempo.



Fonte: *Software Fluke*.

Através do gráfico acima é possível determinar o valor dos picos de corrente. A corrente nominal do compressor é aproximadamente 500mA porém, o valor da corrente de pico é de aproximadamente 10A.

Ilustração 18 – Temperatura em relação ao tempo.



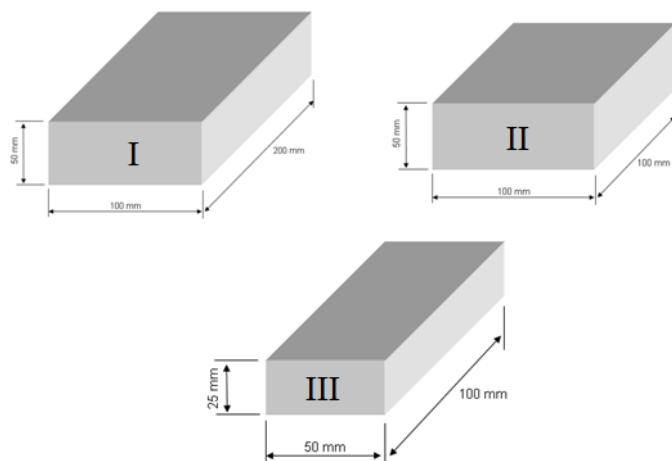
Fonte: *Software Fluke*.

#### 4.4. Ensaio de Consumo e Cargas Térmicas

Para realização do ensaio de consumo é necessário preencher completamente o *freezer* do produto com cargas térmicas, porém é necessário atenção a alguns requesitos como respeitar o espaçamento mínimo de 15mm e o máximo de 25mm (ISO8561,1995).

Os pacotes térmicos são em formas de paralelogramos com três tamanhos e pesos diferentes (0,125kg, 0,5kg e 1kg) e composição que simulam as características da estrutura da carne bovina magra.

Ilustração 19 – Pacotes térmicos com suas respectivas dimensões.



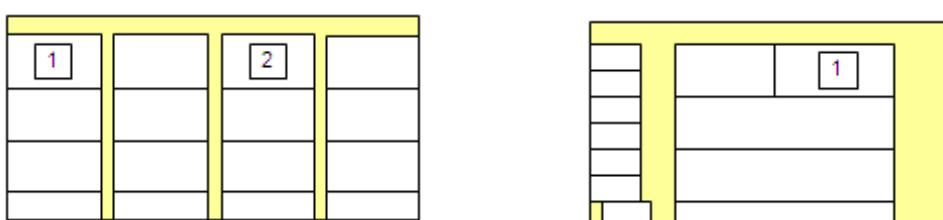
Fonte: Norma ISO8187.

O pacote do tipo I pesa 0,125kg;

O pacote do tipo II pesa 0,5kg;

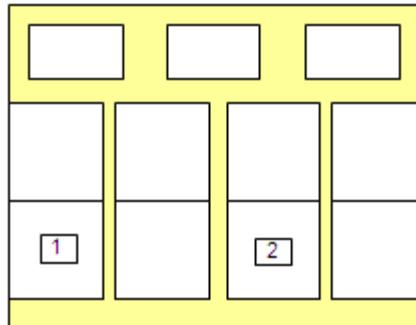
O pacote do tipo III pesa 1,0kg.

Ilustração 20 – Visão frontal (à esquerda) e visão lateral (à direita).



Fonte: Documento interno MABE.

Ilustração 21 – Visão superior



Fonte: Documento interno MABE.

Tabela 4 – Quantidade de Pacotes.

Quantidade de Pacotes	
1Kg	10 unidades
0,5Kg	2 unidades
0,125Kg	37 unidades
M	2 unidades
Total	16,625Kg

Fonte: Documento interno MABE.

Nota-se que no plano de carga térmica apresentado na figura acima há dois pacotes numerados (#1, #2). Estes pacotes são denominados de pacotes “M” e os mesmos devem ser pacotes do tipo II, de 0,5kg, com um termopar alocado em seu centro geométrico (ISO8561,1995).

Ilustração 22 – Pacote “M” com termopar alocado em seu centro geométrico.



Fonte: Documento interno MABE.

Os pacotes térmicos do tipo “M” são colocados obrigatoriamente nos pontos mais quentes do freezer, tais locais são determinados por outros ensaios normativos (ISO8561,1995).

Ilustração 23 – Disponibilidade das cargas térmicas.



Fonte: produção do próprio autor.

O ensaio consiste em determinar o quanto de energia elétrica (Wh) o refrigerador consome ao longo do período de 24h e a partir daí, determina-se o valor obtido para o período de um mês.

As temperaturas necessárias que pacotes “M” devem atingir são determinadas pelo número de estrelas que o produto é classificado (para tal classificação de estrelas novamente se refere à norma ISO8561: 1995).

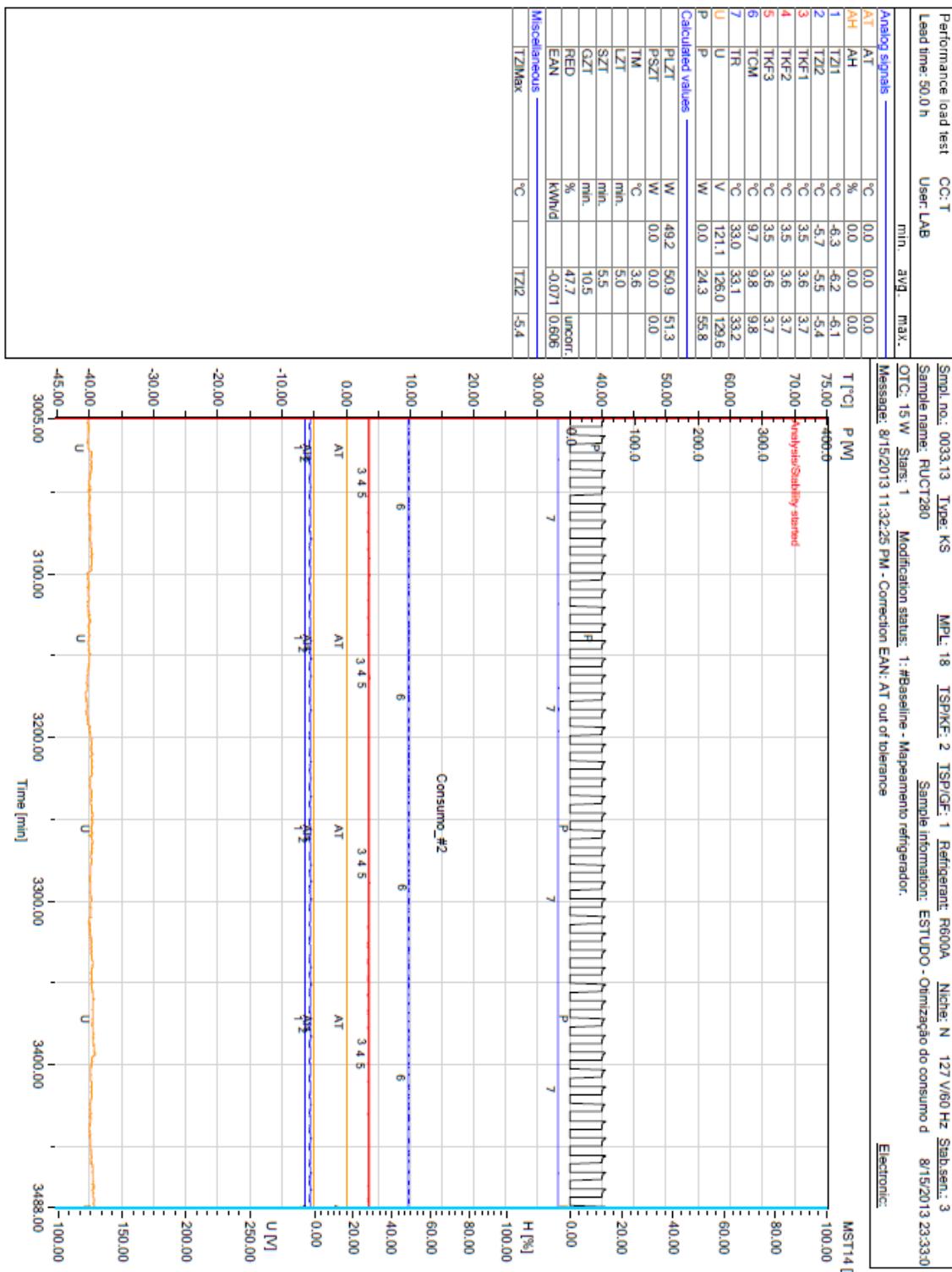
Conforme especificado na tabela 5 o número de estrelas desse produto é de uma estrela. Por tal motivo a temperatura necessária é de -6,0[°C] no pacote mais quente e no refrigerador a temperatura deve ficar entre 0,0 e 5,0 [°C] (ISO8561,1995). Porém como

é muito difícil atingir tal temperatura com tanta precisão, o ensaio precisa ser realizado em duas etapas.

A primeira etapa consiste gerar uma variação de temperatura de +2°C tanto no freezer como no refrigerador e na segunda parte é realizado o inverso da primeira, ou seja, gerando uma variação de -2[°C] no freezer e no refrigerador. Esse ensaio não precisar ser especificamente nessa ordem.

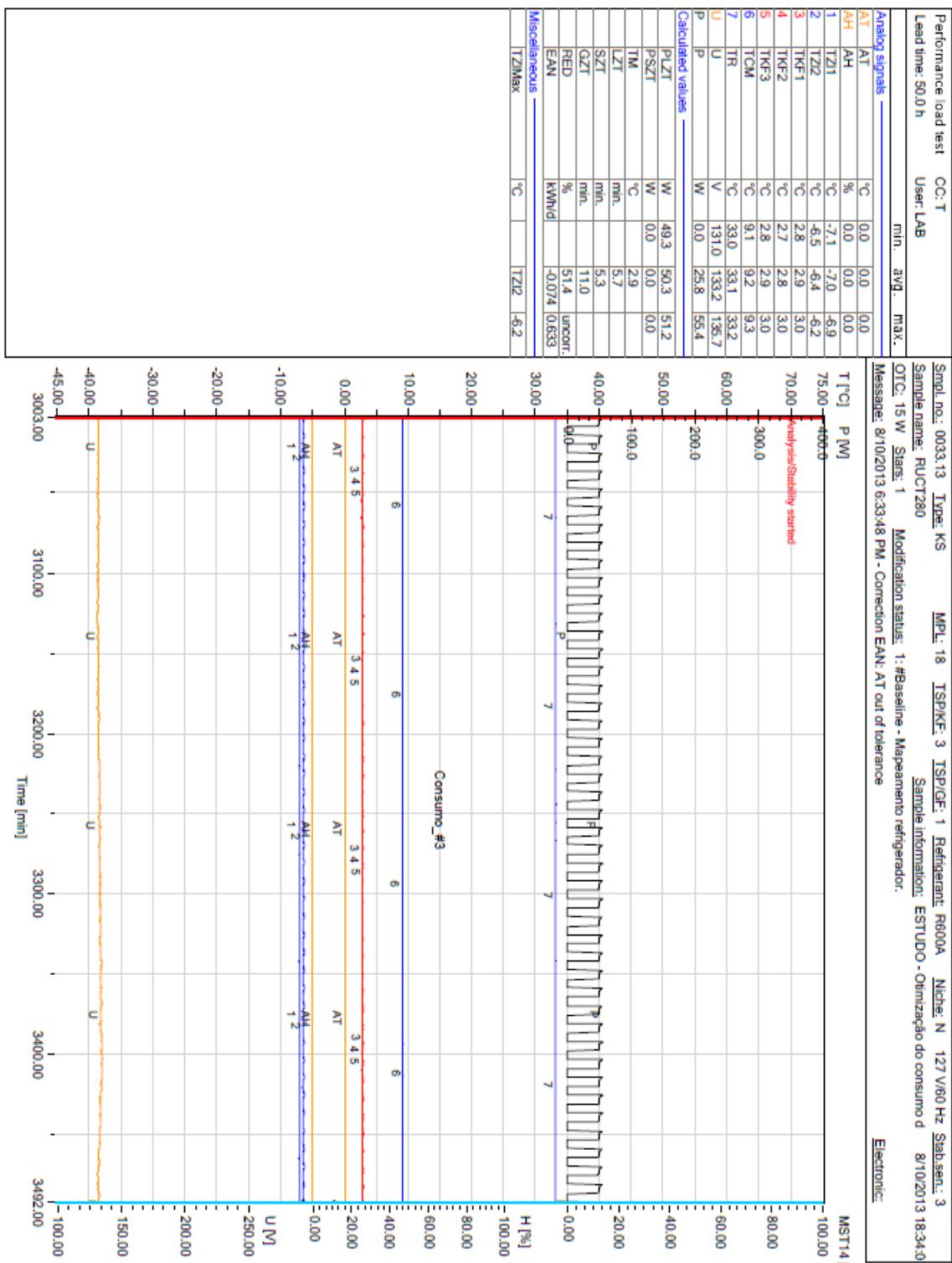
Para a realização do ensaio de consumo a câmara climática controlada deve estar a 32[°C] devido ao clima tropical do Brasil (ISO8561,1995).

Ilustração 24 – Ensaio de consumo realizado com o termostato na posição média.



Fonte: Software Sigmadata.

Ilustração 25 – Ensaio de consumo realizado com o termostato na posição máxima.



Fonte: Software Sigmadata.

Segue resultado do consumo obtido pelo produto *baseline*:

Tabela 5 – Consumo de energia do refrigerado *Cycle Defrost*.

<b><u>CONSUMO DE ENERGIA</u></b>		<b>MODELO: RUCT280</b>		<b>LAB ID. 0033.13</b>
Tensão / Freq.:	127V / 60Hz	Consumo de energia declarado na ENCE (kWh/mês)	15,8	
Compressor:	EMX20CLC	Faixa de eficiência energética declarada:	A	
Isolamento	Ciclopentano	Límite de consumo para faixa declarada (kWh/mês)	24,2	
Temp. ambiente (°C):	32			
Modificação (MS):	1			
Observações		Estudo de controle do compressor EMX20CLC (Baseline) - Gerson - Agosto/13		
volume dos compartimentos	★★★	litros	0,0	
	★★	litros	0,0	
	*	litros	29,0	
	refrigerado	litros	223,0	
frost free (s / n)	-	n		
	volume ajustado	litros	263,89	
categoria	refrigerador		1	
	combinado		2	
	combinado frost free		3	
	congelador vertical		4	
	congelador vertical frost free		5	
	congelador horizontal		6	
consumo padrão		kWh / mês	28,2	
dados de ensaio	controles	# 2	# 3	WR
	C (Wh/24h)	606	633	WLT
	TM (°C)	3,6	2,9	
	TZI max (°C)	-5,4	-6,2	-6,0
	pontos de verificação		-3,5	3,1
Consumo		18,8 kWh / mês	<b>NÃO CONFORME</b>	
Índice de eficiência		0,665	* considerando uma tolerância de + 4%	
Classe de consumo		A	classe de consumo real	A

Fonte: Documento interno empresa MABE.

## 5. Inversor de Frequência

Para fazer alterar a rotação do motor do compressor, o *hardware* possui um inversor de frequência simplificado projetado apenas para esse intuito, sem as opções de mudanças de parâmetros presentes nos inversores industriais.

### 5.1. Introdução

O inversor de frequência trata-se de um equipamento desenvolvido para aplicações em vários setores industriais. Ele é capaz de produzir tensão e frequência ajustáveis com a intenção de controlar a velocidade do motor de indução, o que torna o inversor versátil e dinâmico.

Ilustração 26 – O inversor de frequência.



Fonte: tecmaf.com.br

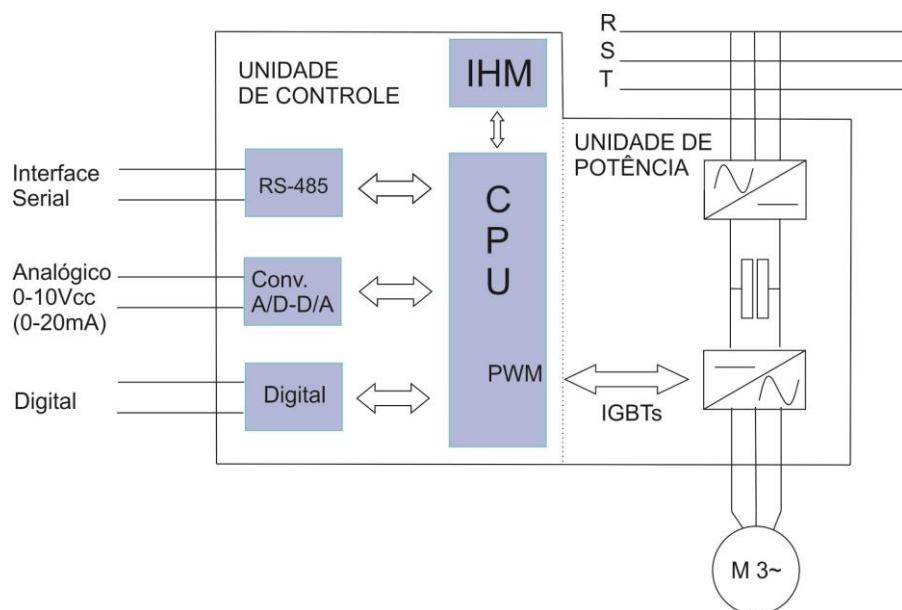
## 5.2. Blocos Componentes do Inversor de Frequência

O inversor de frequência é constituído por blocos componentes onde cada um desempenha sua função.

### 5.2.1. CPU – Unidade Central de Processamento

Pode ser formada por um microprocessador ou um microcontrolador onde estão mantidas todas as informações e parâmetros do sistema. É nesse bloco em que é realizada a função essencial para o funcionamento do inversor de frequência: fornecimento dos pulsos de disparo para os *IGBTs*.

Ilustração 27 – Blocos componentes do inversor.



Fonte: produção do próprio autor.

### 5.2.2. IHM Interface Homem/Máquina

Consiste em um *display* onde é possível verificar-se grandezas do motor e o que está ocorrendo com o inversor de frequência. Trata-se de um conjunto de teclas onde o usuário pode parametrizar o inversor de acordo com a sua aplicação.

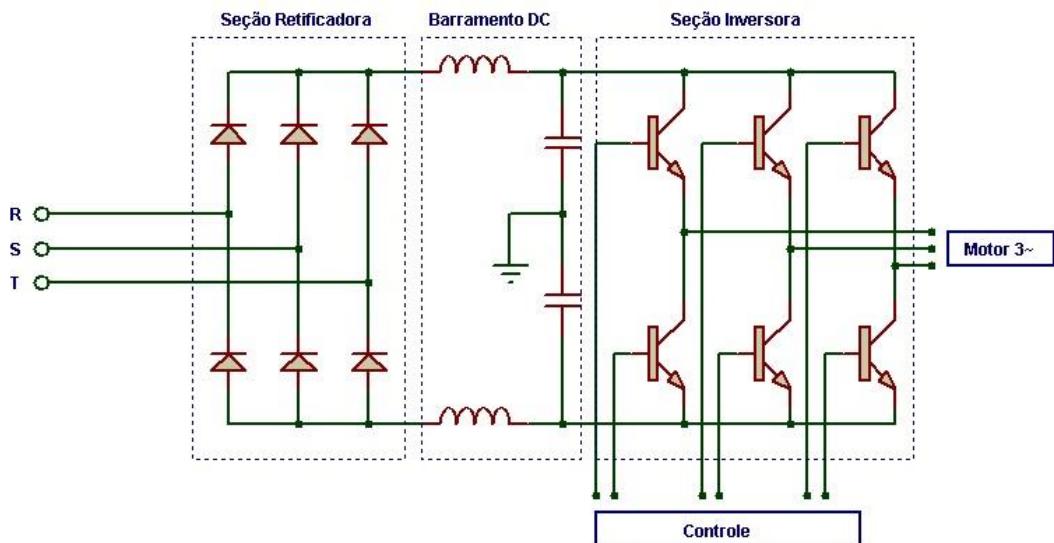
### 5.2.3 Interfaces

Na interface analógica o inversor utiliza sinais de tensão contínua analógicos para fazer com que o motor de indução desempenhe uma determinada rotação. Esse valor de rotação é alterado pelo inversor quando o sinal de tensão muda seguindo uma relação de proporcionalidade. Também o inversor pode utilizar interfaces digitais e seriais.

### 5.2.4. Bloco de Potência

Em tese o bloco de potência é constituído por uma seção retificadora, uma seção inversora e um barramento DC onde unidos, são responsáveis por converter corrente alternada em corrente contínua e por fim em corrente alternada novamente. A seção retificadora através do barramento DC alimenta a seção inversora como demonstra a figura a seguir:

Ilustração 28 – Seção retificadora.



Fonte: produção do próprio autor.

**Círcuito Retificador ou Conversor:** Transforma o sinal alternado oriundo da rede de alimentação cuja frequência é 60 Hz, em contínua através da retificação feita por diodos distribuídos em forma de uma ponte trifásica. A tensão de saída por sua vez não é exatamente contínua, pois possui um determinado “*ripple*” que, com o auxílio do barramento DC será minimizado.

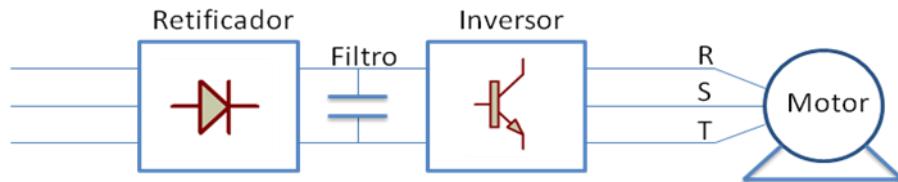
**Barramento DC:** Ou também chamado de *Link DC* é responsável por minimizar as ondulações do sinal retificado, porém não completamente. Nessa etapa existem capacitores e indutores, onde um diminuirá as ondulações de tensão e o outro as ondulações de corrente, respectivamente (Moro Franchi, 2013).

**Círcuito Inversor:** Transforma a tensão contínua proveniente do *filtro* em um sinal alternado. Esse sinal possui tensão e frequência inconstantes.

A inversão do sinal é realizada através do chaveamento de transistores de potência (*IGBTs*) realizado de uma maneira lógica cuja sua coerência é concedida pelo circuito de controle (*CPU*).

Associando as duas seções obtém-se a estrutura do inversor de frequência.

Ilustração 29 – Estrutura do inversor.

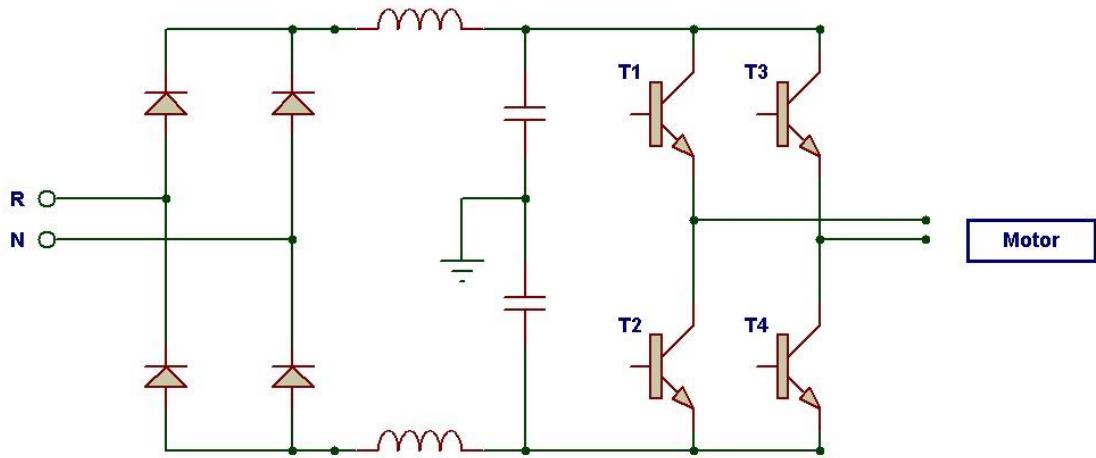


Fonte: produção do próprio autor.

### 5.2.5. Chaveamento

Para melhor compreensão do chaveamento realizado na seção inversora é utilizado o circuito de um inversor de frequência monofásico.

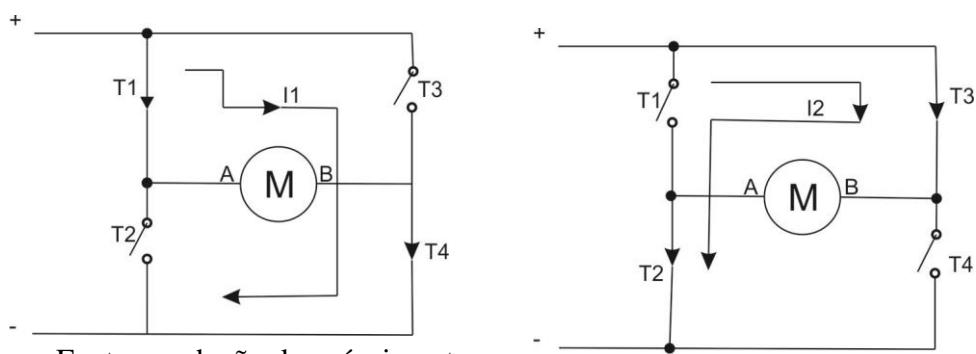
Ilustração 30 – Circuito do inversor de frequência monofásico.



Fonte: produção do próprio autor.

A lógica realiza o chaveamento dos transistores sempre em pares gerando a mudança no sentido da corrente que passa pelo motor monofásico.

Ilustração 31 – Alteração do sentido da corrente devido ao chaveamento dos transistores.



Fonte: produção do próprio autor.

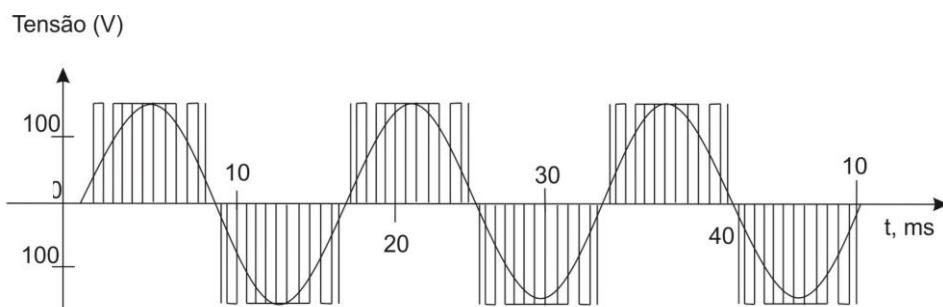
Nesse caso os transistores T1 e T4 estão fechados enquanto T2 e T3 estão abertos, logo a corrente circula de A para B.

Inversamente a situação acima, a corrente circula de B para A comutando os transistores T2 e T3.

Consequente a esse chaveamento tanto a corrente quanto a tensão que é fornecida ao motor tonam-se alternadas novamente. O que vai interferir proporcionalmente na rotação do motor é a frequência de chaveamento parametrizada pelo inversor (Moro Franchi, 2013).

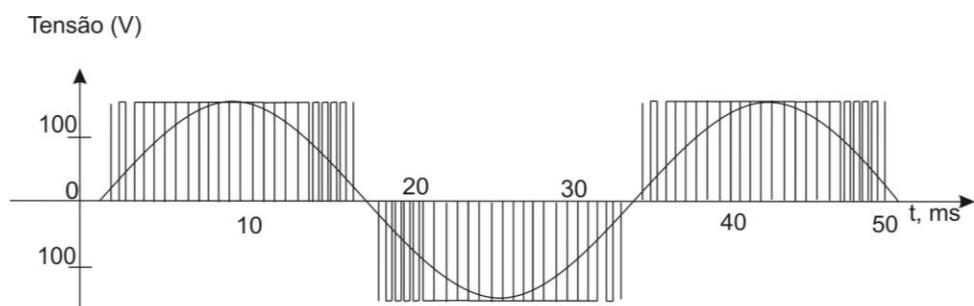
Abaixo seguem dois gráficos que representam o sinal de saída do inversor sendo o primeiro uma forma de onda de 60Hz, 120V e o segundo, 30Hz, 120V.

Ilustração 32 – Sinal de saída do inversor 60Hz/120V.



Fonte: produção do próprio autor.

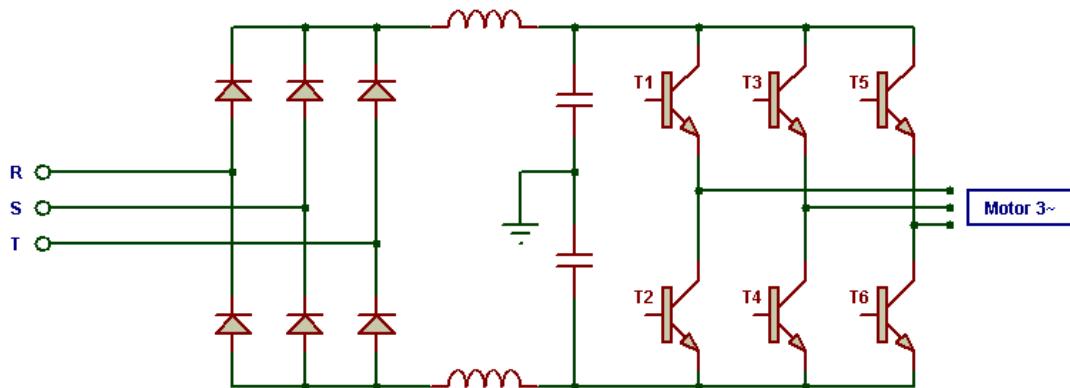
Ilustração 33 – Sinal de saída do inversor 30Hz/120V.



Fonte: produção do próprio autor.

Seguindo o mesmo princípio do sistema monofásico temos o circuito trifásico do inversor de frequência.

Ilustração 34 – Circuito trifásico do inversor de frequência.



Fonte: *Software Proteus*.

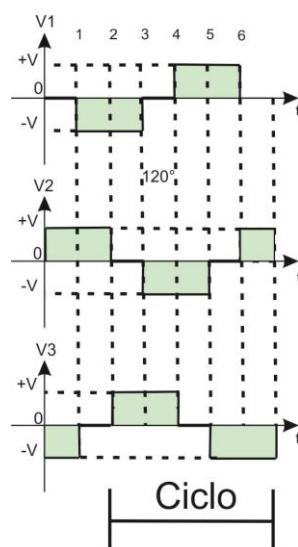
Seguindo a lógica de chaveamento dos *IGBTs* demonstrada na tabela abaixo é gerado uma tensão de saída alternada com fases  $120^\circ$  defasadas entre si.

Tabela 6 – Lógica de chaveamento.

1º Instante	2º Instante	3º Instante	4º Instante	5º Instante	6º Instante
T1, T2, T3	T2, T3, T4	T3, T4, T5	T4, T5, T6	T5, T6, T1	T6, T1, T2

Fonte: produção do próprio autor.

Ilustração 35 – Forma da onda de saída do inversor.



Fonte: produção do próprio autor.

Em relação à modulação da frequência pode-se dizer o seguinte: a velocidade de chaveamento dos *IGBTs* alteram a frequência que será entregue ao motor de indução. Se o chaveamento for acelerado o tempo para completar um ciclo será menor, consequentemente a frequência será elevada, caso contrário a frequência será diminuída. Logo, sendo a frequência variável, a velocidade do motor será modificada.

Tomando como exemplo pode citar-se a seguinte situação: Um motor possui quatro polos e é ligado a um inversor que fornece 60Hz. Sua velocidade nominal será:

$$\omega = \frac{120 \cdot f}{n^{\circ} \text{ polos}} = \frac{120 \cdot 60}{4} = 1800 \text{ [RPM]}$$

Para obter esse valor de frequência é necessário que o tempo de ciclo seja 16,67ms pois:

$$f = \frac{1}{T} = \frac{1}{16,67 \cdot 10^{-3}} = 60 \text{ Hz}$$

Se o tempo de ciclo for reduzido para 4,165ms a frequência será de 240Hz consequentemente passará para 7200 [RPM].

Apesar de haver essa possibilidade de alterar a frequência de chaveamento dos transistores deve-se verificar antes se o motor é apto para receber tal alteração. Os inversores geralmente trabalham em uma faixa entre 20Hz e 300Hz, se um motor trabalhar em 300Hz não sendo preparado para isso terá sua vida útil reduzida.

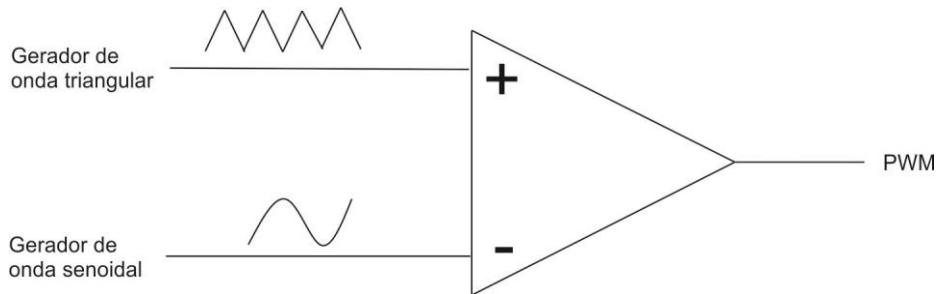
Vale ressaltar que o inversor de frequência utilizado nesse projeto é monofásico, logo não há necessidade de apresentar maiores características em relação ao inversor trifásico.

### **5.3. Modulação por Largura de Pulso**

Conhecido por sua abreviação (*PWM*), a modulação por largura de pulso trata-se de um método de chaveamento dos transistores mais utilizado nos inversores de frequência atuais. A partir desse método é possível chavear os transistores de saída com o intento de obter uma tensão de saída CA sintetizada de frequência variável.

Basicamente esse método consiste na comparação de dois sinais de tensão, uma com onda triangular e outra com onda senoidal.

Ilustração 36 – Sinal senoidal e triangular.

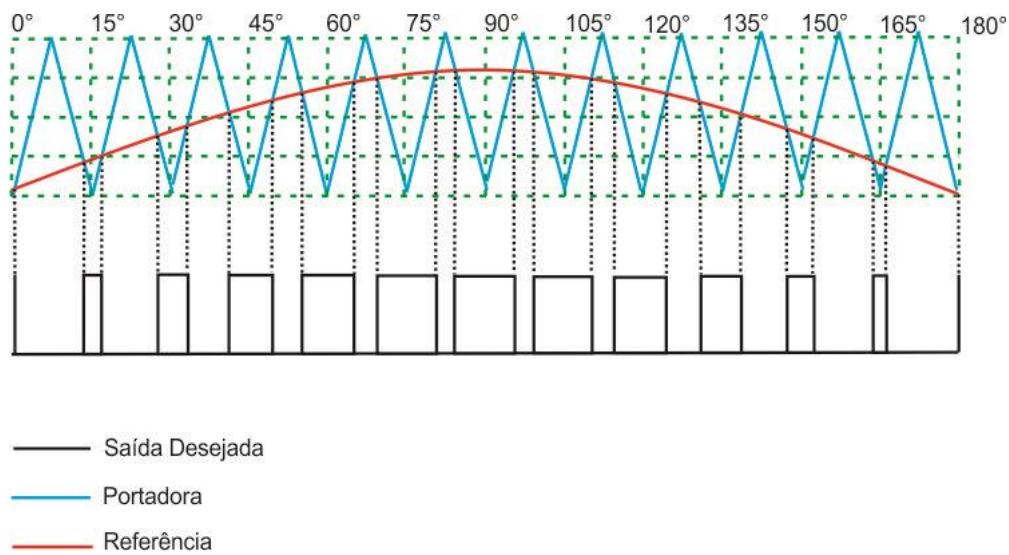


Fonte: produção do próprio autor.

O sinal senoidal é chamado *sinal de referência*. Esse sinal é considerado como a “imagem de tensão de saída desejada”. É com ele que o sinal de saída do inversor é comparado. O sinal triangular é chamado *sinal de portadora*. Sua frequência deve ser dez vezes superior a frequência do sinal de referência para obter-se um sinal de saída do inversor de qualidade.

A modulação é realizada através de um circuito que compara a referência e a portadora variando sua largura de pulso de acordo com a amplitude do sinal de referência.

Ilustração 37 – Comparaçao da referência com a portadora.

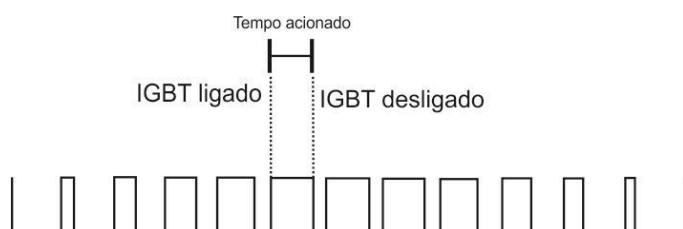


Fonte: produção do próprio autor.

Analizando a figura acima observa-se que quanto mais a referência se aproxima do pico o pulso torna-se maior. Caso contrário a largura do pulso é menor quando o sinal se afastar do pico. A frequência pode ser alterada a qualquer momento.

Com o chaveamento dos *IGBTs* é possível controlar o valor da corrente que o inversor concederá ao motor pelo tempo em que o *IGBT* permanece ligado. De inicio ao ser acionado por um curto período de tempo é passado pelo *IGBT* apenas uma parcela da corrente, a partir de então esse tempo aumenta gradativamente juntamente com a corrente até que a corrente nominal do motor seja atingida. Após esse processo o *IGBT* é acionado em tempos progressivamente menores para diminuir a corrente.

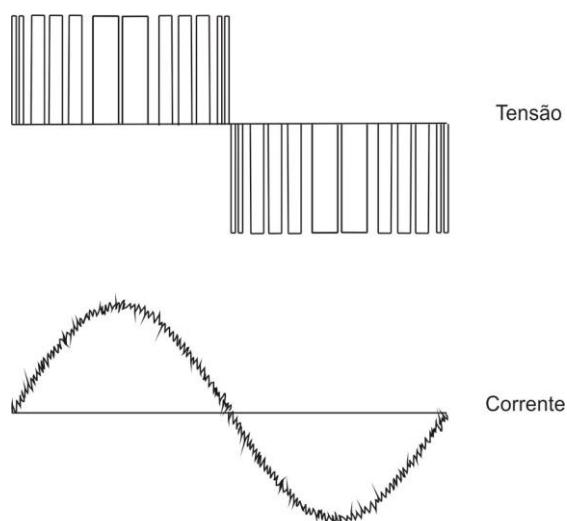
Ilustração 38 – Acionamento dos *IGBTs*.



Fonte: produção do próprio autor.

Deste modo as formas de onda de saída do inversor de frequência em tensão e em corrente adquirão o seguinte formato:

Ilustração 39 – Forma da onda de saída de tensão e corrente.

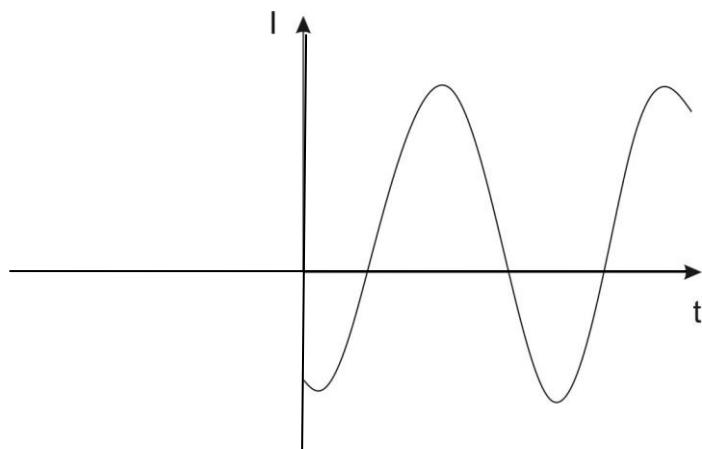


Fonte: produção do próprio autor.

A tensão CC passa pelo processo de modulação para resultar uma tensão e frequência que podem ser variáveis. Para obter tensão e corrente pequenas os *IGBTs* são acionados em tempos curtos e inversamente em tempos prolongados quando é desejado uma frequencia de saída alta que só é possível obte-la com uma alta tensão.

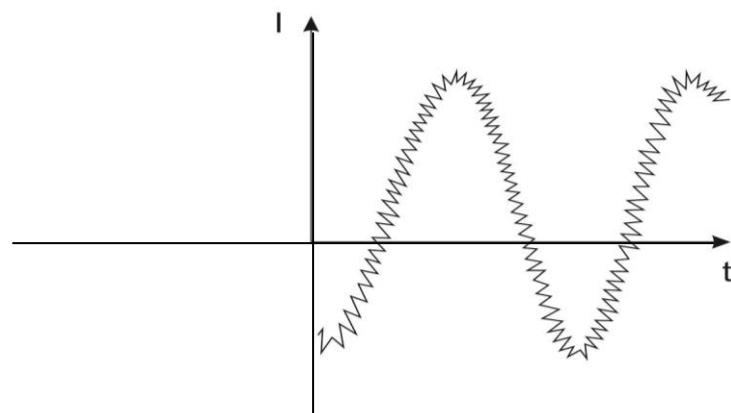
Conforme a frequencia de chaveamento é elevada, a forma de onda de saída será mais aproximada de uma senóide perfeita porém com o ruído também elevado proporcionalmente.

Ilustração 40 – Frequênciа de chaveamento alta.



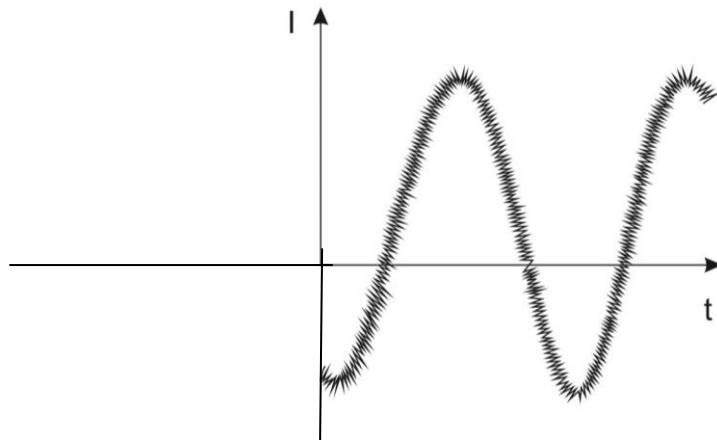
Fonte: produção do próprio autor.

Ilustração 41 – Frequênciа de chaveamento baixa.



Fonte: produção do próprio autor.

Ilustração 42 – Frequência de chaveamento média.



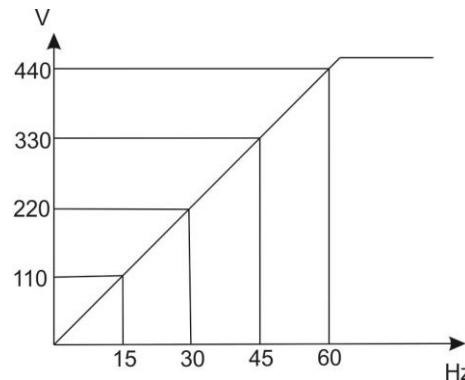
Fonte: produção do próprio autor.

#### 5.4. Classificação dos inversores de frequência

**Inversor de Controle Escalar:** A diferença entre os tipos de inversores está relacionada em como o torque é controlado. No caso do *Inversor de controle escalar* o torque é controlado através da relação Tensão/Frequência ou Voltz/Hertz constante. Nesse tipo de controle o fluxo torna-se aproximadamente constante. São empregados em sistemas simples cujo não há a necessidade de respostas rápidas de torque e velocidade.

A velocidade é elevada proporcionalmente com a frequência até atingir a velocidade nominal do motor, pois se a tensão continuar aumentando juntamente com a frequência, o enrolamento do motor pode ser avariado (Moro Franchi, 2013).

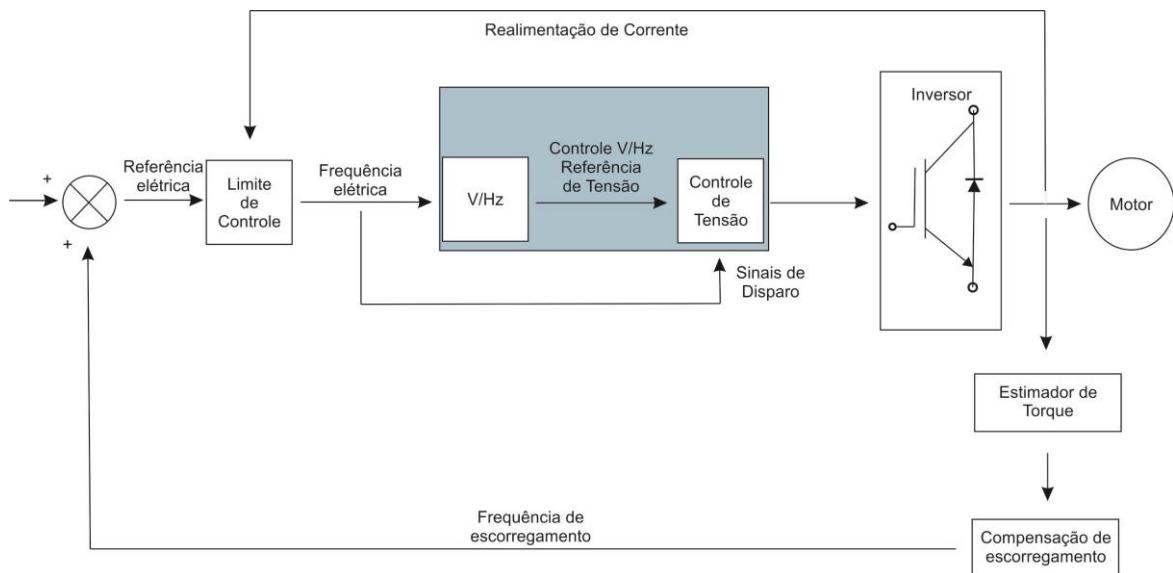
Ilustração 43 – Relação V/F.



Fonte: produção do próprio autor.

Delimitados em sistemas cujo objetivo é o controle de velocidade do motor sem o controle de torque desenvolvido, os conversores de controle escalar utilizam o método voltz/hertz assumindo a velocidade como referência por meio de uma fonte externa variando a tensão e a frequência. Geralmente a frequência é operada entre 10Hz e 60Hz.

Ilustração 44 – Diagrama de blocos para o controle escalar.

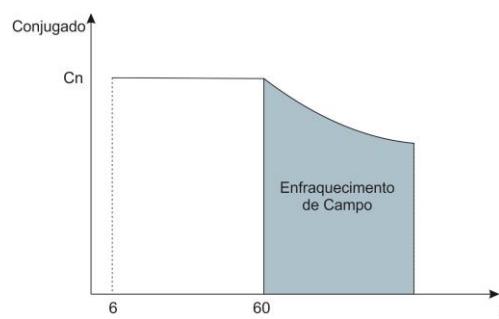


Fonte: produção do próprio autor.

#### 5.4.1. Região de Enfraquecimento

Trata-se da região em que o conjugado do motor começa a decair após manter-se constante até a frequência nominal ser atingida. Isso ocorre pois a partir desse instante a corrente de magnetização do motor cai juntamente com o fluxo magnético no entreferro.

Ilustração 45 – Região de enfraquecimento.



Fonte: produção do próprio autor.

#### **5.4.2. Características do Controle Escalar**

- Controle feito em malha aberta;
- Tempos de aceleração e desaceleração podem ser ajustados pelo operador;
- Precisão de velocidade em função do escorregamento do motor que varia em função da carga;
- Utilizado em sistemas onde não há necessidade de alta precisão no controle de velocidade;
- Baixo custo em relação ao vetorial;
- Não recomendado para motores de baixa rotação;

Entre as classificações dos tipos de inversores pode-se citar ainda o inversor de frequência vetorial, pois, como o inversor utilizado nesse projeto trata-se de um tipo escalar, os vetoriais não serão abrangidos.

## **6. Hardware e Software**

Através dos testes realizados foi possível obter o conhecimento geral do produto, tanto a parte hermética quanto a parte funcional do compressor e consequentemente será necessário o desenvolvimento de um novo tipo de controle do produto para atingir o objetivo geral do TCC. Esse novo controle consiste em projetar uma placa microcontrolada através de um *software* embarcado.

### **6.1. Hardware e Seu Desenvolvimento**

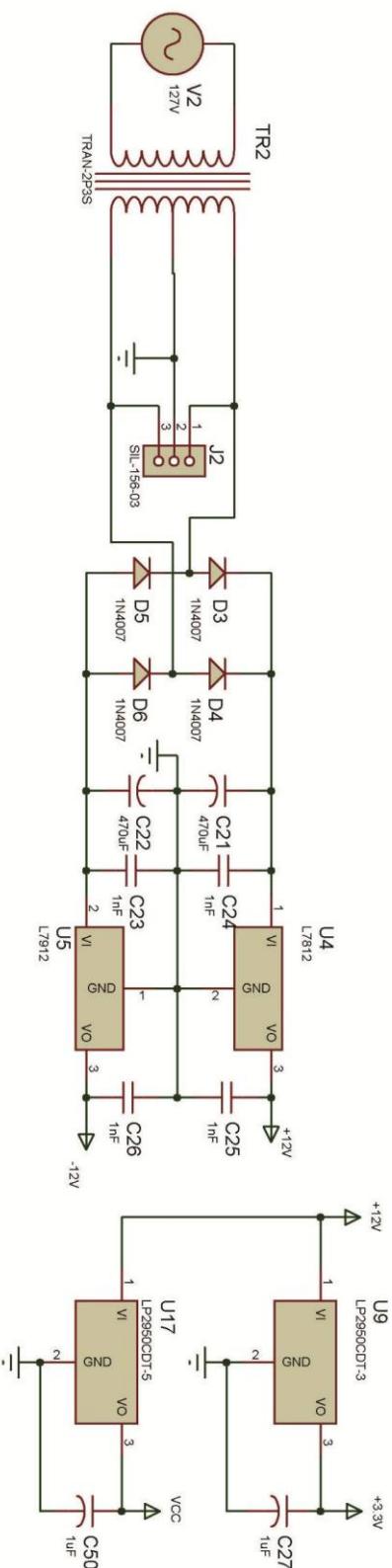
O *hardware* desenvolvido tem como objetivo substituir o controle atualmente empregado ao refrigerador por um sistema de controle mais eficiente. Ele é composto por vários componentes que devidamente associados, compõem a sua estrutura. O hardware desenvolvido consiste nos seguintes componentes:

#### **6.1.1. Fontes**

O *hardware* é composto por três fontes de alimentação responsáveis por alimentar os circuitos digital, analógico e de potência. São necessárias as três fontes no circuito elétrico para obter uma isolação galvânica entre os circuitos de potência e o circuito digital. Abaixo segue a descrição de cada uma dessas fontes:

**Fonte de Alimentação dos Circuitos Digital & Analógico:** Esse projeto trata-se de uma fonte linear e simétrica que possui um transformador “abaixador de tensão” 12V+12V de 500mA. O próximo estágio desse projeto é o circuito retificador composto por 4 diodos 1N4007 esboçados na configuração em “Ponte”. Em seguida o capacitor é utilizado para reduzir a Tensão de *Ripple*. A fonte possui dois reguladores de tensão, sendo o primeiro o CI (Circuito Integrado) 7812 de +12V e o segundo o CI 7912 para tensão negativa de -12V. Além desses existem mais reguladores de tensão espalhados pelo circuito, entre eles temos o regulador 78L33 de +3,3V para alimentar o microcontrolador (MCU) e o OP-AMP (Amplificador Operacional) LT1013, o regulador LP2950 +5V e por ultimo o regulador 7905 -5V para o Amplificador de Instrumentação AD623. Essa fonte tem como objetivo fornecer as devidas tensões para o MCU, Interface do Usuário (IHM) e o transdutor de temperatura.

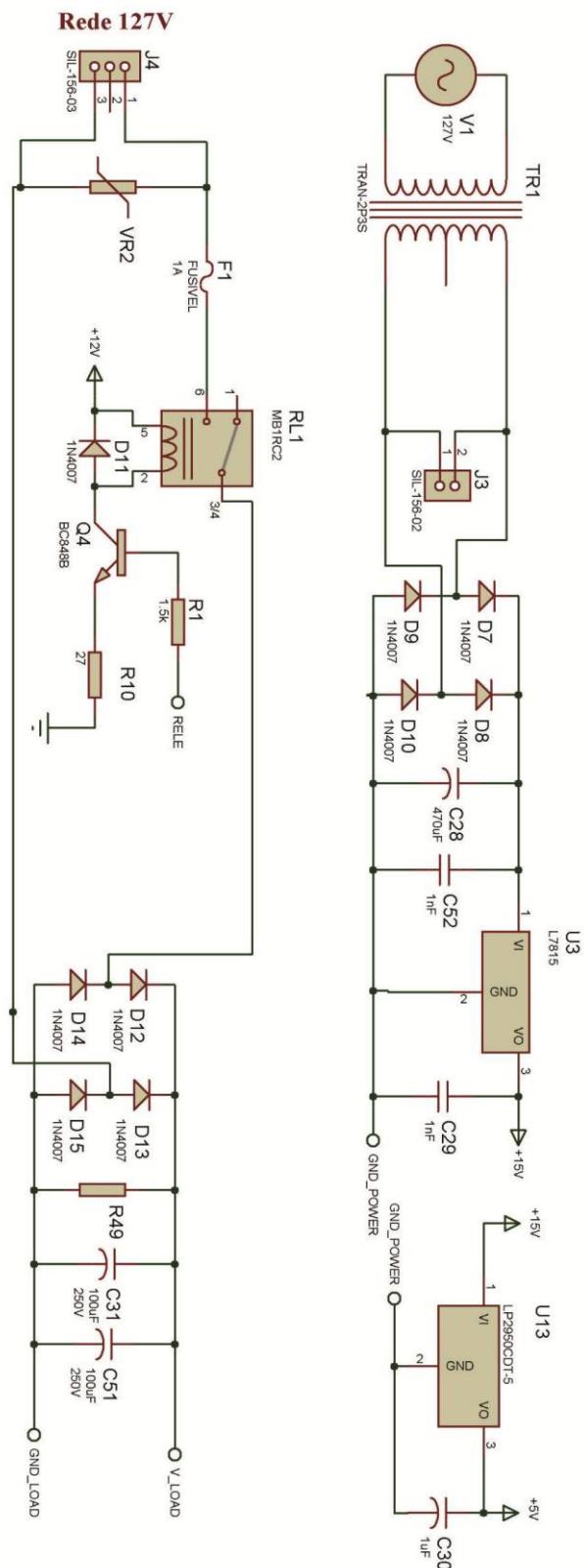
Ilustração 46 – Fonte de alimentação do circuito digital e analógico.



Fonte: *Software Proteus*.

**Fonte de Alimentação do Circuito de Potência:** Esta fonte é dividida em duas, sendo a primeira fonte responsável por alimentar a parte de potência do inversor de frequência. Nesse circuito não há um transformador em sua entrada logo a tensão da rede é retificada diretamente, o que proporcionará em sua saída uma tensão de aproximadamente +180V. Na entrada de seu circuito existe uma chave para ligar e desligar a alimentação proveniente da rede, composta por um relé e seu respectivo drive de acionamento pelo transistor bipolar NPN BC548, um diodo 1N4007, e resistores. O sinal de comando é enviado pelo microcontrolador. O retificador utilizado nessa fonte também é projetado na configuração em “ponte” composta por 4 diodos (1N4007) e um capacitor para reduzir a tensão de *Ripple*. A segunda fonte de alimentação do circuito de potência tem a função de alimentar o CI FSB50250S e os acopladores ópticos. Essa fonte é linear, fornecendo uma tensão de +15V e +5V. Em sua entrada há um transformador “abaixador de tensão” 12V+12V de 500mA. Neste circuito não é utilizado o “*Center-Tap*” do transformador, com isso a tensão de saída do mesmo passa a ser de +24V. O estágio seguinte é o circuito retificador composto por diodos em ponte, capacitores e reguladores de tensão sendo o primeiro 7815 +15V e o segundo LP2950.

Ilustração 47 – Fonte de alimentação do circuito de potência.



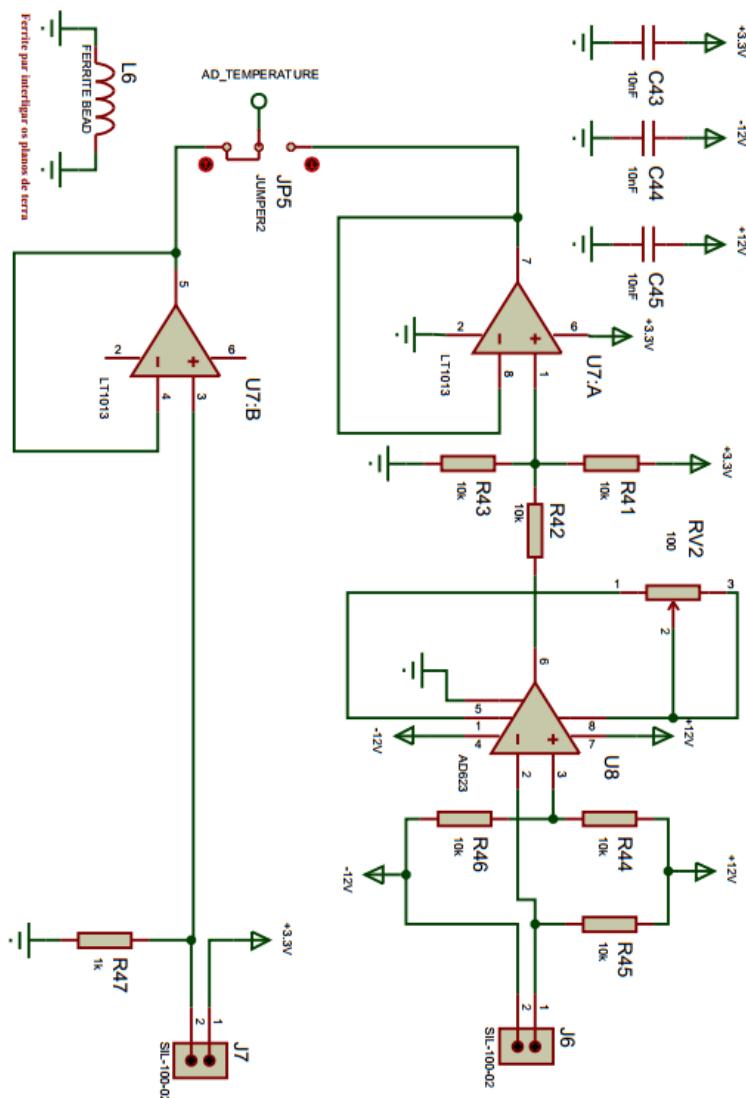
Fonte: Software Proteus.

### 6.1.2. Transdutor de Temperatura

Este circuito tem como função básica converter temperatura em tensão elétrica numa faixa de tensão suportada pelo canal de ADC do microcontrolador.

De inicio o seguinte circuito para aquisição de temperatura foi utilizado no projeto, porém, o mesmo apresentou instabilidade na leitura feita pelo ADC consequentemente prejudicando o valor da temperatura aferida.

Ilustração 48 – Circuito transdutor de temperatura.



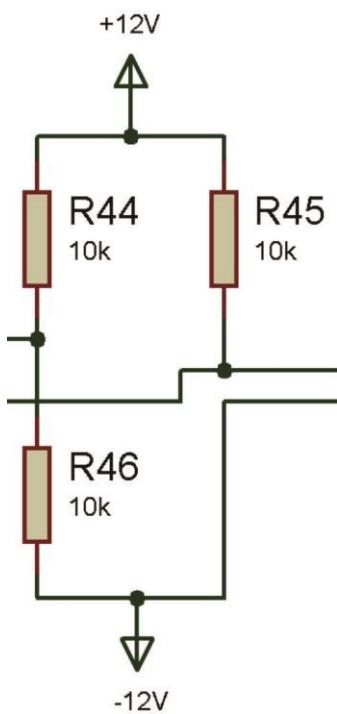
Fonte: *Software Proteus*.

O circuito transdutor de temperatura é composto por quatro blocos sendo eles:

**SENSOR DE TEMPERATURA:** o sensor utilizado é do tipo termistor (*NTC - Negative Temperature Coeficient*). Esse sensor atua como uma resistência elétrica que pode ser alterada através da influência térmica, isto é, apresenta um valor de resistência elétrica para cada valor de temperatura absoluta. No caso do sensor tipo *NTC*, sua resistência diminui conforme o aumento da temperatura.

**PONTE DE WHEATSTONE:** É um instrumento capaz de medir a resistência elétrica com grande precisão, porém, com um transdutor adequado é possível medir outras grandezas físicas. O circuito da ponte de *Wheatstone* divide-se em dois divisores de tensão, onde três resistências são de valores conhecidos e a quarta resistência é o termistor. Segue imagem ilustrativa do circuito:

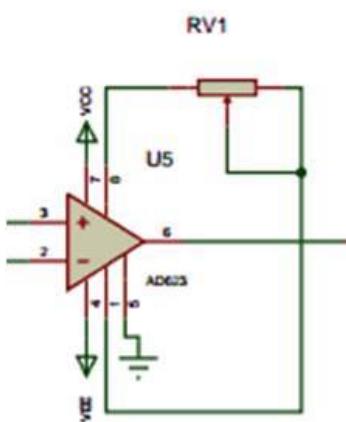
Ilustração 49 – Ponte de *Wheatstone* empregada no circuito do transdutor de temperatura.



Fonte: *Software Proteus*.

**AMPLIFICADOR DE INSTRUMENTAÇÃO:** É uma configuração composta por três amplificadores operacionais e alguns resistores. Esse CI possui impedância elevada em sua entrada e reduzida em sua saída. Essa configuração permite maior ganho em *dB* em relação ao OP-AMP tradicionais. O amplificador de instrumentação utilizado é o circuito integrado AD623 da *Analog Devices*, que contem os três OP-AMP e os resistores, tudo em um único encapsulamento e seu ganho é feito através de um resistor externo ( $R_g$ ).

Ilustração 50 – Amplificador de instrumentação.

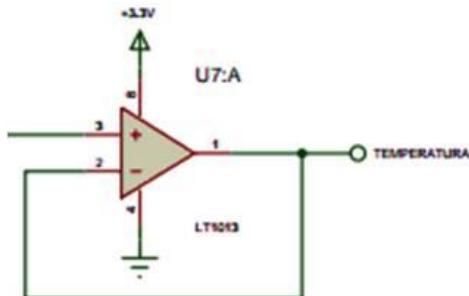


Fonte: *Software Proteus*.

**AMPLIFICADOR OPERACIONAL – BUFFER:** O *buffer* é o circuito de tensão que não amplifica e não inverte fase ou polaridade, assim permitindo o “casamento” da impedância entre o transdutor de temperatura, que possui uma alta impedância de saída, com o ADC do microcontrolador, que por sua vez, tem baixa resistência de entrada. O *buffer* eleva a impedância reduzindo as perdas no sinal. O sensor de temperatura NTC por se tratar de uma resistência elétrica compõe um dos resistores da ponte de *Wheatstone*, que de acordo com a variação de sua resistência gera uma pequena diferença de potencial (DDP).

Essa DDP é amplificada pelo AD623 e o seu ganho é definido através de um potenciômetro de ajuste. O estágio seguinte é do *buffer* que realiza o “casamento” de impedância e do ajuste de tensão de entrada do ADC do MCU. Após a conversão do sinal do transdutor o software se encarrega de calcular a temperatura através de um algoritmo.

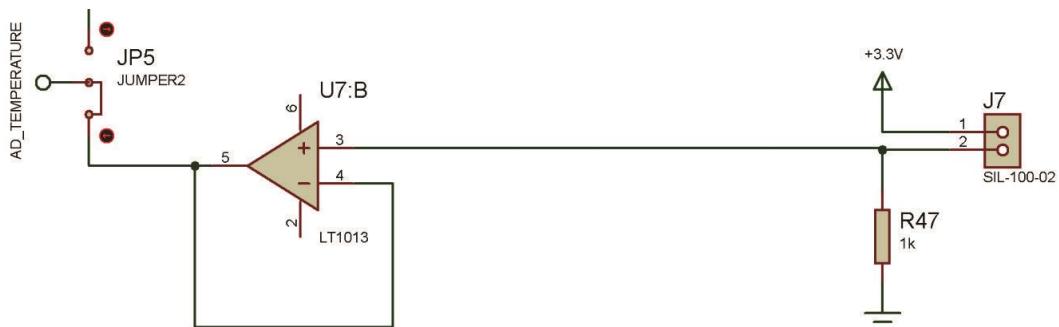
Ilustração 51 – Circuito do *buffer*.



Fonte: *Software Proteus*.

Então se optou por um circuito mais simples constituído por divisor resistivo em um buffer para o “casamento” de impedâncias como mostra a figura a seguir.

Ilustração 52 – Divisor resistivo.



Fonte: *Software Proteus*.

Ambos os circuitos estão presentes no *hardware* e é possível utilizar um ou o outro através de um *jumper* que conecta o circuito selecionado com o ADC.

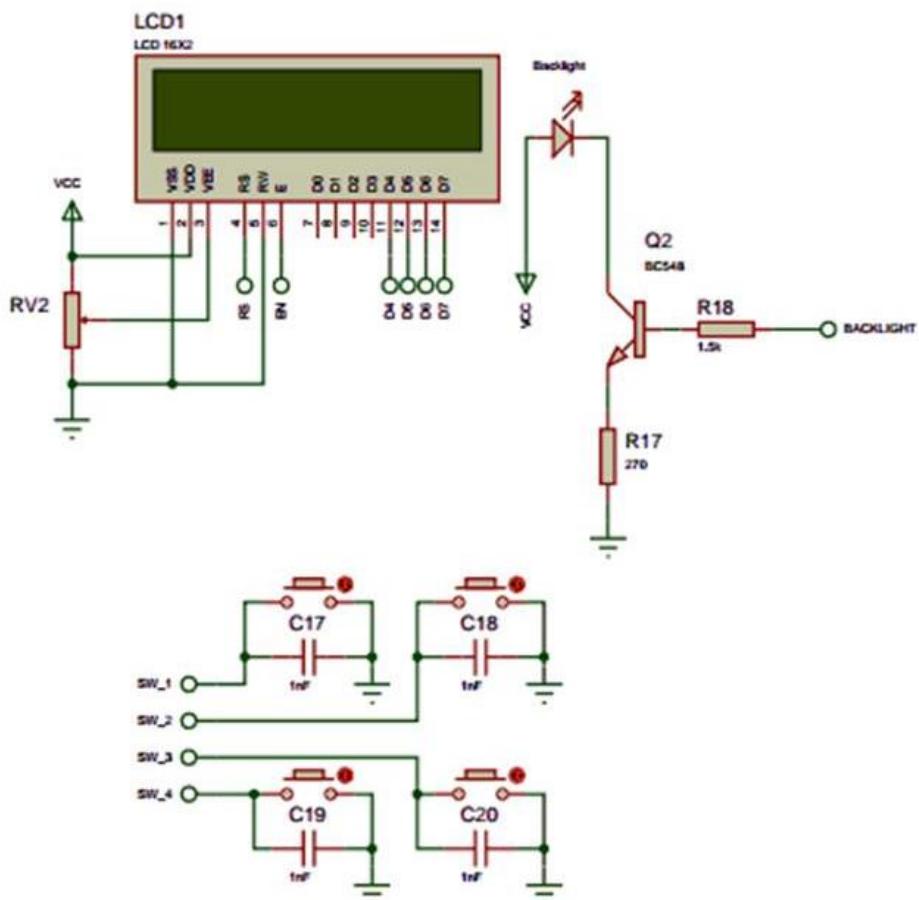
### 6.1.3. INTERFACE COM USUÁRIO (IHM)

A interface é composta por um *display* de LCD alfanuméricico de duas linhas por dezesseis caracteres e um conjunto de *push-button*. Esse circuito tem como finalidade

interagir com o usuário, possibilitando que o mesmo possa visualizar informações do sistema ou configurá-lo.

O MCU é responsável por realizar a escrita e controle do *display* através de um algoritmo seguindo as especificações no *datasheet*. Os *push-button* são conectados diretamente nas portas digitais do MCU que são configurados para serem entradas.

Ilustração 53 – Circuito IHM.



Fonte: *Software Proteus*.

#### 6.1.4. Microcontrolador (MCU)

O MCU é responsável por gerenciar todo o funcionamento do sistema através de um *software* embarcado.

O hardware desenvolvido para esse projeto possui dois microcontroladores sendo um destinado para a aplicação e outro que faz o processo de gravação e o *debugger* (depurar) do

software seguindo as instruções do *datasheet* e *Deferece Design*. O micro controlador adotado é MCF51MM256 da *Freescale* que possui as seguintes características:

- Núcleo *CodeFire* V1 com 50MHz;
- Memória Flash de 256 KB e 32 KB de SRAM;
- 4 Timer PWM;
- 2 UART (Serial), 2 SPI(Serial Peripheral Interface), 1 I<sup>2</sup>C (Inter-Integrated Circuit);
- USB - Device/Host/On-The-Go;
- ADC de 16 bit;
- DAC de 12 bit.

A imagem do circuito da CPU segue em ANEXO.

O segundo microcontrolador que é utilizado no circuito de gravação e depuração do software é o MC9S08JM60CLD da empresa *Freescale*, sendo o mesmo um circuito aberto e o firmware é oferecido pelo próprio fabricante em conjunto com a PEMICRO. Esse conjunto é nomeado de OSBDM (*Open Source BDM*).

OSBDM (Open Source BDM) é o conjunto formado por *hardware, software e firmware* que permite comunicações e depurações de diferentes microcontroladores *Freescale*. Esse projeto "*open source*" desenvolvido é suportado pela comunidade P&E. A *Microcomputer Systems* (PEMICRO) é uns dos principais contribuintes para o projeto contribuindo com *drivers*, correções e suporte ao código-fonte. A figura ilustrativa do OSBDM segue em ANEXO.

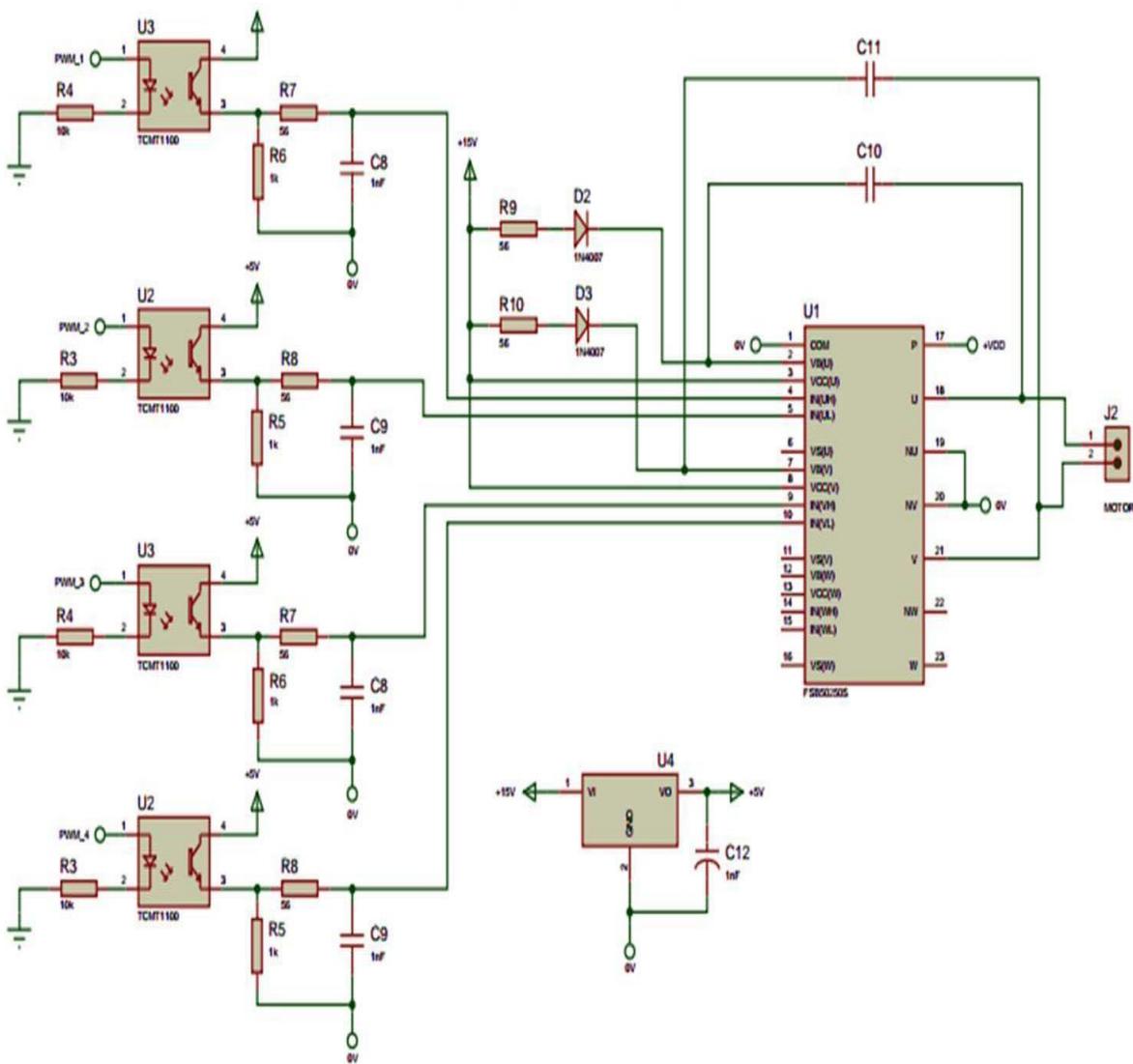
#### 6.1.5. INVERSOR DE FREQUÊNCIA

Inicialmente o projeto seria feito com "DRIVER" de potencia da empresa *Fairchild Semiconductor* que é solução de inversor compacta em aplicações de acionamento de motores de pequenas potências. Trata-se de um circuito integrado com seis *MOSFET* e possui como características técnicas:

- Tensão de carga 500V RMS;
- Tensão de Isolamento de 1500V RMS;
- Interface lógica de 3,3V a 5V;
- Corrente nominal de 1A.

Além do circuito do CI FSB50250 possuir acopladores ópticos interfaciando a entrada lógica do mesmo com o microcontrolador, realiza a isolação entre o circuito digital com o de potência. O controle do inversor é feito pelo MCU utilizando *PWM (Pulse-Width Modulation - Modulação por Largura de Pulso)*. O circuito montado é sugerido segundo o *datasheet*.

Ilustração 54 – Circuito do inversor de frequência.



Fonte: Software Proteus.

Porém o mesmo não apresentou um resultado satisfatório o que impossibilitou o seu uso. Assim se fez necessário pesquisar novos componentes com o objetivo de atender as especificações do projeto, onde foram encontrados dois modelos: IR3101 e IRAM10UP60A, ambos são da empresa *International Rectifier*.

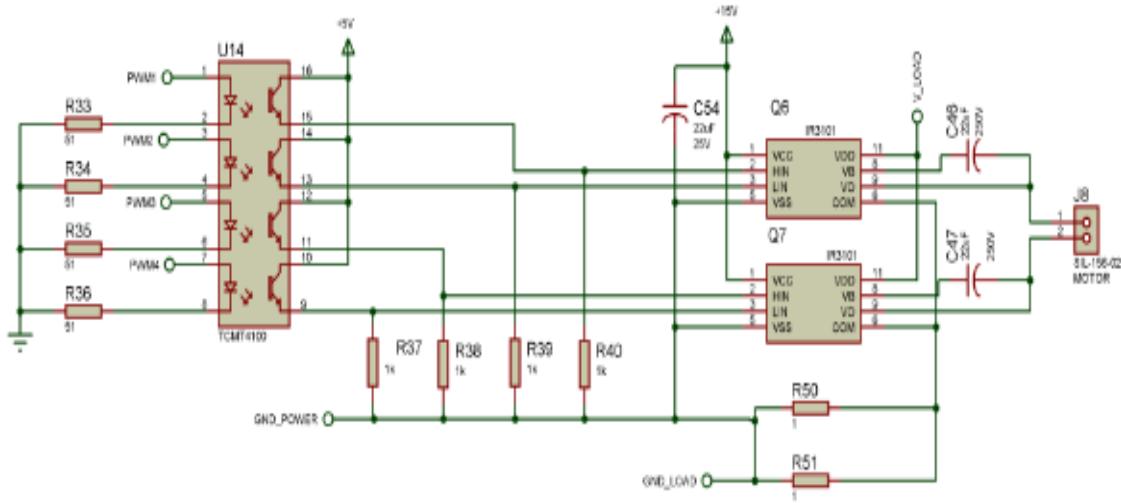
O modelo IR3101 é um modulo de potencia composto por dois transistores MOSFET em ponte e um circuito lógico de controle que possui as seguintes características:

- Tensão de carga 500V RMS;
- Corrente nominal 1,6A;

- Tensão da interface lógica de 5V;
- Tensão de isolamento de 1500 V RMS.

O circuito montado com esse componente segue as instruções do *datasheet* e o *Reference Design*.

Ilustração 55 – Circuito do inversor de frequência com IR3101.

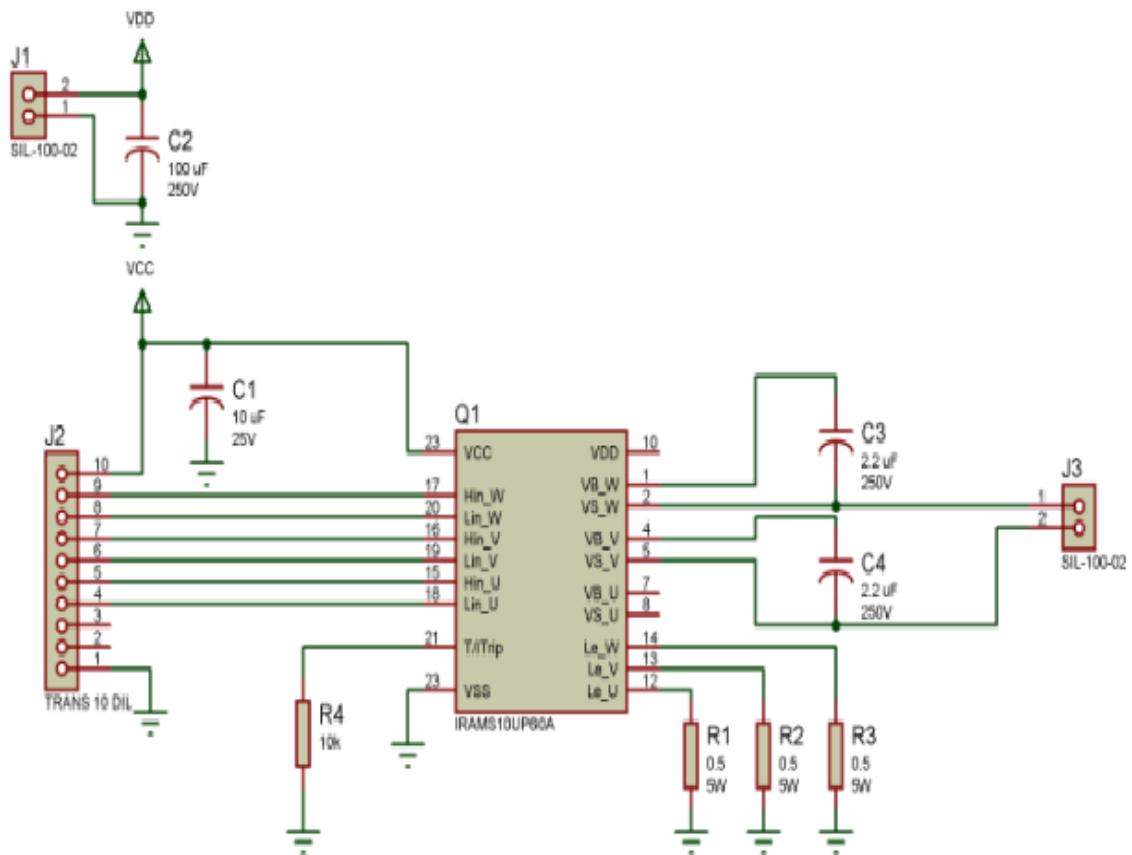


Fonte: *Software Proteus*.

Com esse componente é necessário a utilização de dois CI's para atender os requisitos do projeto. Os componentes IRAMS10UP60 são módulos de potência que possuem seis transistores IGBT e bloco lógico de controle, o mesmo permite a configuração de inversor trifásico com apenas um CI com as seguintes características:

- Tensão de carga 600V RMS;
- Corrente nominal 10A;
- Tensão da interface lógica de 5V;
- Tensão de isolamento de 2000 V RMS.

Ilustração 56 – Circuito do inversor de frequência com IRAMS10UP60.

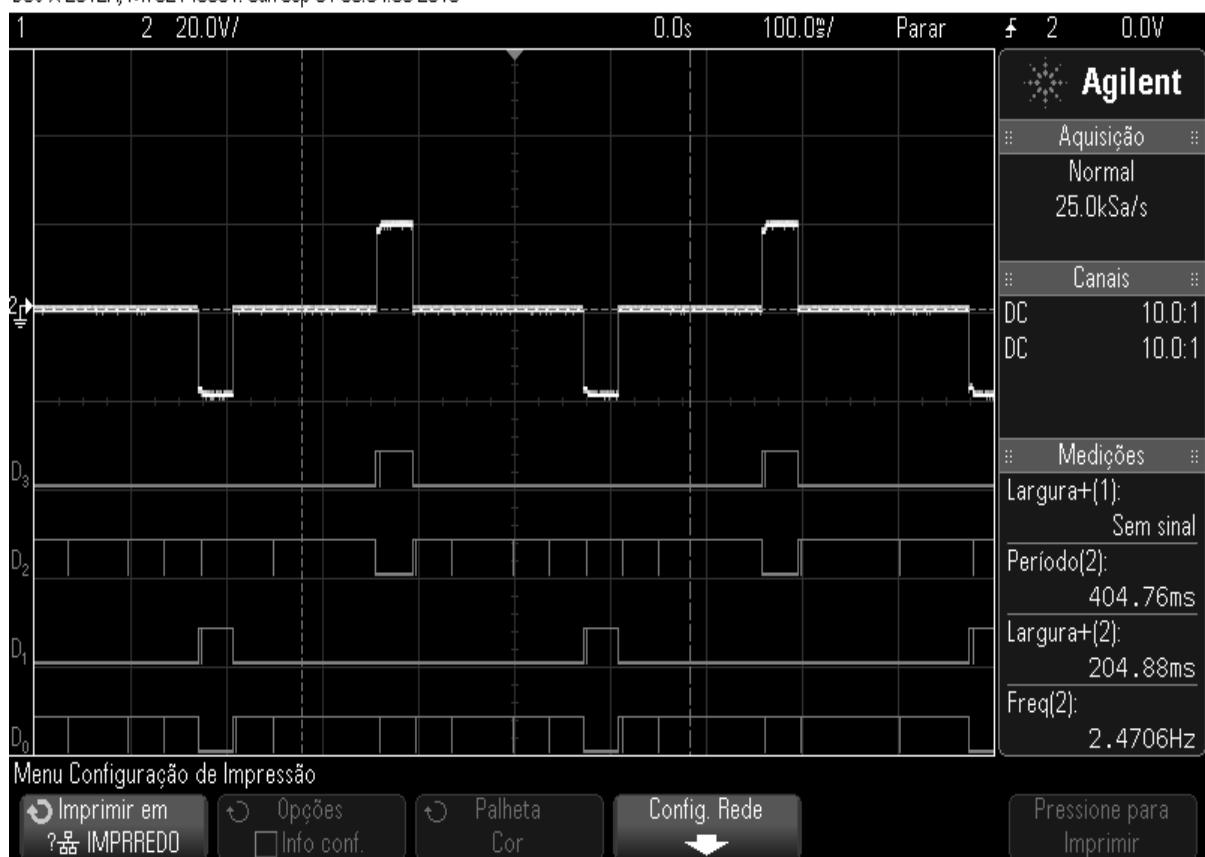


Fonte: *Software Proteus*.

O primeiro componente a ser montado e testado foi o IR3101 e obteve um bom desempenho possibilitando o desenvolvimento do software de controle. A seguir temos a imagem capturada do osciloscópio com os sinais de entrada de controle gerada pelo MCU e o sinal de saída do inversor de frequência.

Ilustração 57 – Sinal de entrada representada no osciloscópio.

DSO-X 2012A, MY52143031: Sun Sep 01 03:04:08 2013



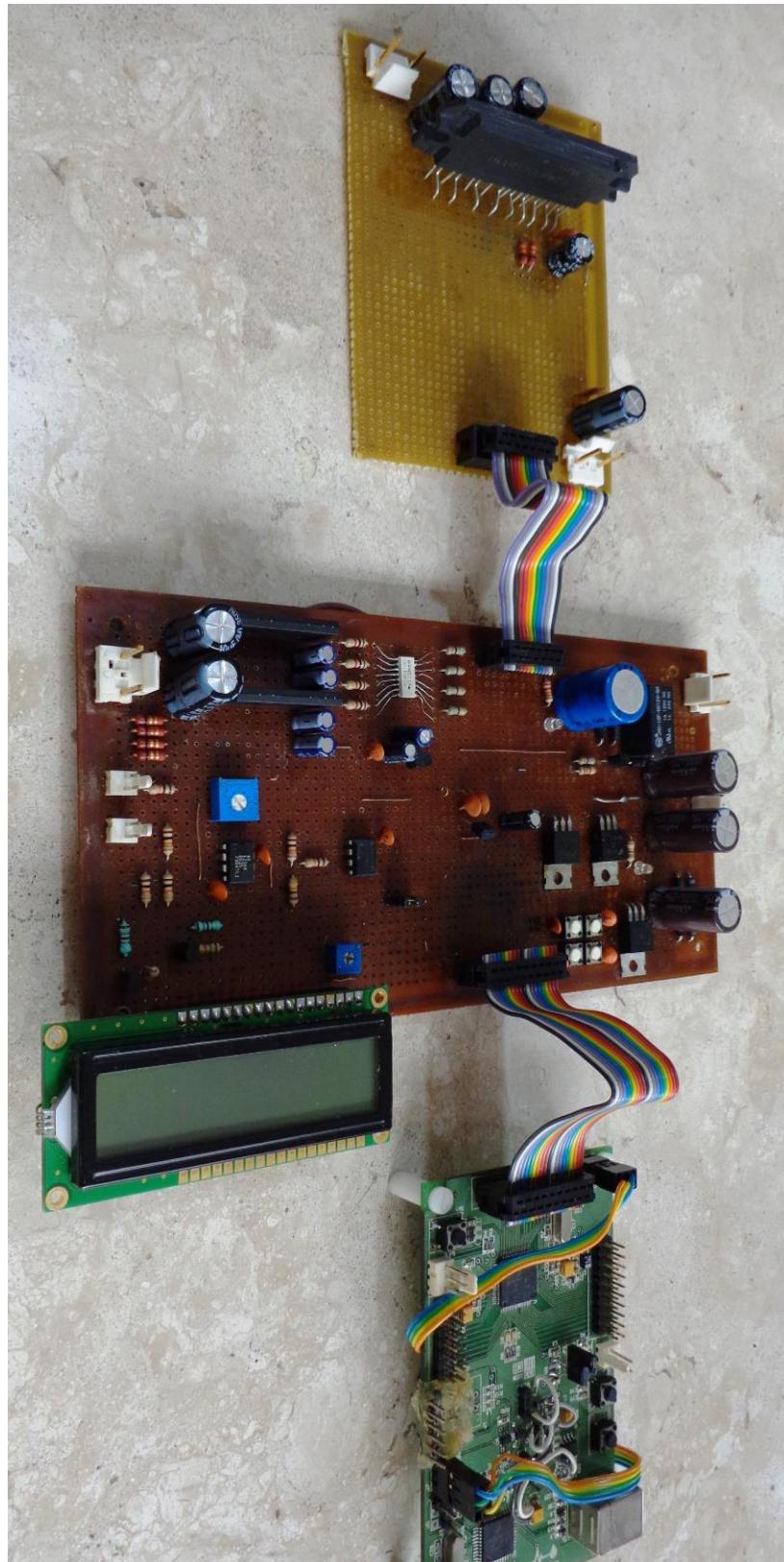
Fonte: Imagem obtida do osciloscópio.

Porém ao realizar os testes com a potência elevada alguns *Drives* apresentaram avarias. Como não havia peças para reposição se fez necessário a montagem do CI IRAMS10UP60.

#### 6.1.6. Desenvolvimento da Placa de Circuito Impresso (PCI)

A montagem do protótipo foi realizada utilizando o padrão de fenolite. Esse tipo de montagem acelera o processo de prototipagem do projeto, porém trazem diversos pontos negativos como surgimento de ruídos, mau contato (solda fria) e surgimento de problemas devido à compatibilidade eletromagnética (EMC). Para evitar esses problemas na utilização da placa padrão, foi desenvolvido *layout* de placa de circuito impresso seguindo as boas práticas de projeto de *circuito & layout*.

Ilustração 58 – *Hardware* montado.

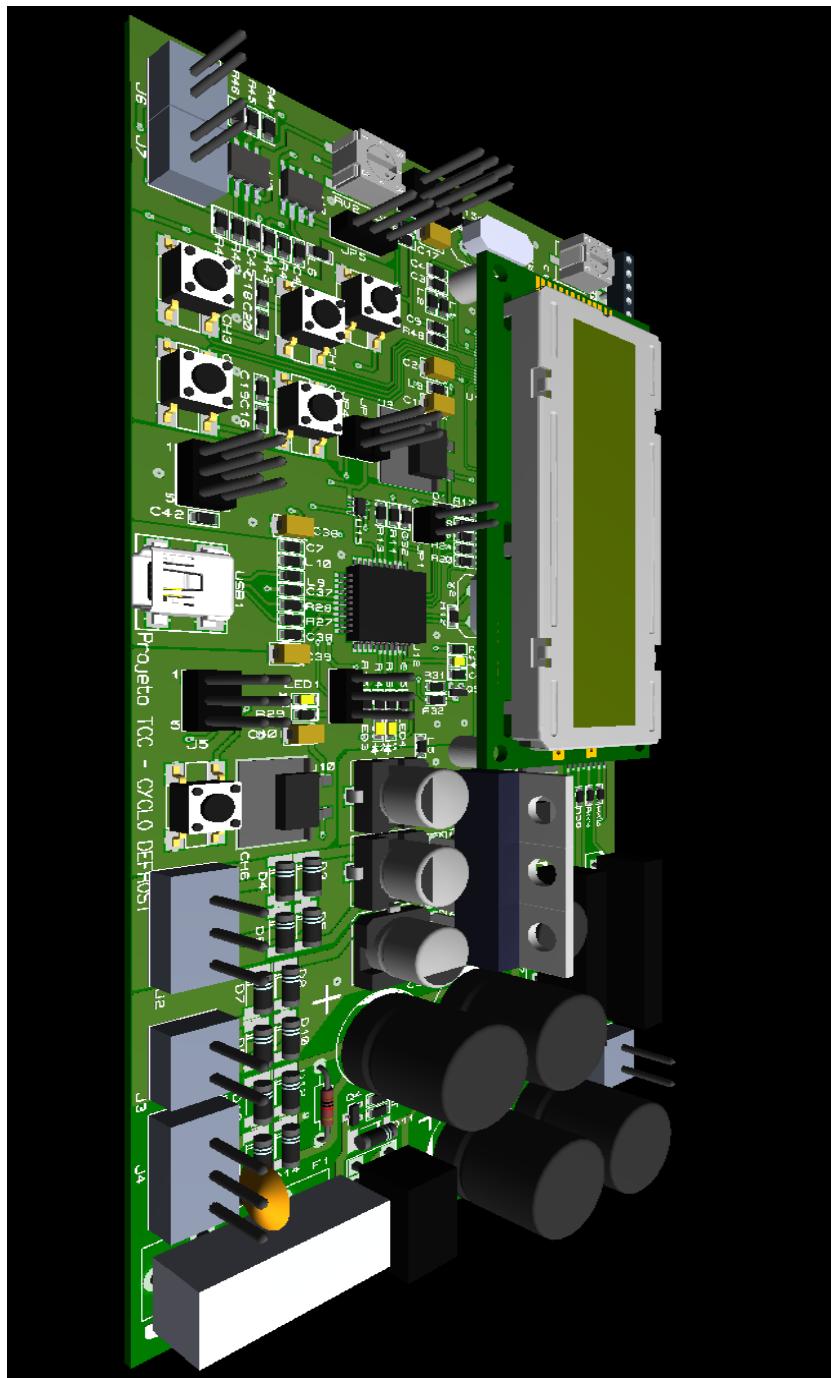


Fonte: produção do próprio autor.

O *layout* da PCI foi desenvolvido deixando os grupos de potência, analógico e digital em áreas bem separadas para evitar qualquer tipo de interferência entre eles.

A PCI não chegou a ser confeccionada devido ao alto custo.

Ilustração 59 – Placa de Circuito Impresso.



Fonte: produção do próprio autor.

## 6.2. Software e Seu Desenvolvimento

O *software* foi desenvolvido com os seguintes objetivos: controlar o inversor de frequência através dos pulsos PWM, calcular a temperatura por intermédio da conversão ADC e controlar a Interface Homem/Máquina (IHM).

Ele foi estruturado obedecendo as práticas de desenvolvimento de *firmware* com o intuito de minimizar os erros e *bugs*. O método adotado foi estruturá-lo em módulos onde cada um é responsável por uma determinada tarefa. O mesmo é dotado de "Flags" que são sinais que são utilizado para sinalizar um evento. Esses Flags podem receber apenas dois valores "*TRUE*" e "*FALSE*".

O *software* foi construído partindo do princípio de não ter nenhuma rotina de atraso (*delay*), para não ocorrer perca do processamento do microcontrolador e consequentemente, erros no controle da frequência do inversor. Para isso foi utilizado o método de Máquina de Estados Finitos (FSM - do inglês *Finite State Machine*), onde o conceito é concebido como uma máquina abstrata, que deve estar em um dos seus finitos estado.

A ferramenta utilizada para desenvolver o *firmware* é *CodeWarrior*, um ambiente integrado (IDE). Essa ferramenta é fornecida pela *Freescale*, empresa fabricante do microcontrolador utilizado no projeto. A mesma possui a opção para compilação e *debuggar* do *software*. A versão utilizada é a 10.3 sendo a mesma baseada no *software* Eclipse.

### 6.2.1. Descrição dos Arquivos

O *software* está divido em diversos arquivos. Cada um deles atua como se fosse uma biblioteca do projeto. Além do arquivo "*main.c*" que é o arquivo principal e possui a função "*void main(void)*", que contém a rotina com o "*loop infinito*", o projeto contém os seguintes arquivos:

Ilustração 60 – Arquivos contidos no *software*.

```
*****
*
* Projeto: Trabalho de Conclusão de Curso - OTIMIZAÇÃO DE UM REFRIGERADOR CYCLO DEFROST
* Autor: Evandro Teixeira, Gerson Stein, Guilherme Martino
*
*****
#include <hidef.h>           /* for EnableInterrupts macro */
#include "derivative.h"       /* include peripheral declarations */
#include <stdio.h>
#include <math.h>
#include "define.h"
#include "externa.h"
#include "aplicacao.h"
#include "Timer.h"
#include "LCD.h"
#include "types.h"
#include "Controle_LCD.h"
/*****
#ifndef __cplusplus
extern "C"
#endif
*****
```

Fonte: produção do próprio autor.

- "hidef.h" e "derivative.h": são gerados pela própria IDE e relacionados ao microcontrolador utilizado.
- "MCUinit.c" e "MCUinit.h": gerado pelo próprio compilador, esse arquivo contem as funções "\_\_initialize\_hardware" , "MCU\_init", vetores de inicialização da interrupções e as rotinas das interrupções.
- "stdio.h" e "math.h": são bibliotecas nativas da linguagem C.

Os arquivos descritos abaixo foram criados especificamente para atender os requisitos do projeto.

- "define.h": possui todas a Macros do projeto. Esse arquivo torna o *firmware* mais eficiente e torna o código fonte mais legível.
- "externa.h": possui a função de interligar a variáveis globais entre os arquivos do projeto.
- "aplicacao.h" e "aplicacao.c": possuem todas as funções com as rotinas do projeto.

- "Timer.h" e "Timer.c": possui a função "delay\_us" que é utilizada na biblioteca do display LCD.
- "LCD.h" e "LCD.c": contém a biblioteca do display de LCD alfanumérico 16x2.
- "types.h": possui algumas definições para permitir a portabilidade do código gerado em linguagem C.
- "Controle\_LCD.h" e "Controle\_LCD.c": arquivos responsáveis pelas telas que são exibidas no *display*.

Ilustração 61 - Trecho do código fonte do arquivo “define.h”.

```
/*
Definições Macro do software
*****
#define PARADO          1
#define READ_CH          2
#define TEMPERATURA      3
#define DECISION         4
#define MOTOR_CONTROL    5
#define DISPLAY          6
#define PARTIDA_MOTOR    2
#define MOTOR_ON          3
#define MOTOR_OFF         4
#define ON                1
#define OFF               0
#define RED               1
#define BLUE              2
#define GREEN             3
#define Linha_1            1
#define Linha_2            2
// Lista de erro
#define ERRO_DISPLAY       1
#define CH_1               PTCD_PTCD1
#define CH_2               PTCD_PTCD5
#define CH_3               PTCD_PTCD7
#define CH_4               PTCD_PTCD3
//Botões de teste prototipo
#define SW_1                PTAD_PTAD6
#define BACKLIGHT_BLUE      PTDD_PTDD6
#define BACKLIGHT_RED        PTDD_PTDD7
#define HIN_1                PTCD_PTCD6 //teste
#define LIN_1                PTCD_PTCD0
#define HIN_2                PTCD_PTCD4
#define LIN_2                PTDD_PTDD2 //TESTE
#define DTFE

```

Fonte: produção do próprio autor.

Ilustração 62 – Trecho do código fonte do arquivo “externa.h”.

```
*****  
#include "types.h"  
//#include "externa.h"  
extern uint8      motor_on;  
extern uint8      motor_off;  
extern uint8      controle_estado;  
extern uint16     t_OFF;      // variavel que controla o tempo off do PWM  
extern uint16     t_ON;       // variavel que controla o tempo on do PWM  
//extern uint16    t_vcc;      // variavel que controla qtd de pulso para determinar tensão RMS  
extern uint32     number;  
extern uint8      channel;  
extern uint16     read_adc;  
extern uint16     resultado;  
extern int        tensao_adc;  
extern float      valor_freq;  
extern float      periodo;  
extern char       LCD_LINHA_1[17];//      =      "  
extern char       LCD_LINHA_2[17];//      =      "  
extern uint8      erro;  
extern uint16     timer;  
extern uint16     temp_mV;  
extern float      temp;  
extern uint16     posicao_termo;  
extern uint8      estado_LCD;  
extern uint8      backlight;  
extern int16      temperature;  
extern int        converte_mili_volt(int dado);  
extern bool       Flag_ch1;    // Indica o estado da chave 1 - SETUP  
extern bool       Flag_ch2;    // Indica o estado da chave 2 - INCREMENTO +  
extern bool       Flag_ch3;    // Indica o estado da chave 3 - INCREMENTO -  
extern bool       Flag_ch4;    // Indica o estado da chave 4 - ENTER  
extern bool       Flag_Overflow_TPM2;  
extern bool       Flag_Overflow_TDM1.
```

Fonte: produção do próprio autor.

Ilustração 63 – Trecho do código fonte do arquivo “Timer.c”.

```
/*  
 * Timer.c  
 */  
#include <hidef.h>          /* for EnableInterrupts macro */  
#include "derivative.h"       /* include peripheral declarations */  
#include "externa.h"  
#include "Timer.h"  
*****  
Função Delay com interrupção de 2 micro segundos  
*****  
void delay_us(unsigned long int t)  
{  
    number = t;  
    while(Flag_Overflow_TPM2 == 0)  
    {}  
    Flag_Overflow_TPM2      = FALSE;  
}  
*****
```

Fonte: produção do próprio autor.

Ilustração 64 – Código fonte do arquivo “types.h”.

```
*****
* Este arquivo permite a portabilidade do código gerado em linguagem C.
* O porte para uma nova plataforma será feito preenchendo os tipos listados
* abaixo.
*****
* Proteção para inclusão recursiva
*****
#ifndef TYPES_H
#define TYPES_H
*****
* Definições de tipos para portabilidade.
*****
/* Tipos numéricos sinalizados */
typedef signed char    int8;   /* Inteiro sinalizado de 8 bits */
typedef signed int      int16;  /* Inteiro sinalizado de 16 bits */
typedef signed long     int32;  /* Inteiro sinalizado de 32 bits */
/* Tipos numéricos não-sinalizados */
typedef unsigned char   uint8;  /* Inteiro não-sinalizado de 8 bits */
typedef unsigned int    uint16; /* Inteiro não-sinalizado de 16 bits */
typedef unsigned long   uint32; /* Inteiro não-sinalizado de 32 bits */
/* Tipos booleanos */
typedef unsigned char   bool;
*****
* Definições padrão
*****
#ifndef FALSE
#define FALSE   ((bool) 0)
#endif
#ifndef TRUE
#define TRUE    ((bool) 1)
#endif
#ifndef NULL
#define NULL   (void *) 0
#endif
```

Fonte: produção do próprio autor.

### 6.2.2. Detalhamento das Principais Funções

**Arquivos MCUunit:** para inicialização dos periféricos do microcontrolador foi utilizada a ferramenta "*Device Initialization*", a mesma se encontra presente na própria IDE "*CodeWarrior*".

O "Device Initialization" é uma ferramenta de inicialização dos periféricos do microcontrolador onde configura-se o micro através de uma interface gráfica. Após o ajuste dos parâmetros o mesmo gera os códigos de inicialização do microcontrolador que são salvos nos arquivos MCUInti.c e MCUInt.h.

Nesses arquivos encontram-se as funções "initialize\_hardware" sendo o algoritmo de inicialização necessária para o funcionamento do MCU, a função "MCUinit" que recebe as rotinas de inicialização de cada periférico configurado.

Além das duas funções citadas esse arquivo contém os vetores de inicialização das interrupções utilizadas e as rotinas de cada interrupção.

**Rotina de interrupção ADC:** o algoritmo contido nessa interrupção recebe o valor do canal que realiza a leitura do ADC e repassa o valor para a variável "read\_adc".

Ilustração 65 – Código fonte da interrupção do ADC.

```
/*
** =====
**     Interrupt handler : isrVadc
**
**     Description :
**         User interrupt service routine.
**     Parameters  : None
**     Returns      : Nothing
** =====
*/
_interrupt void isrVadc(void)
{
    /* Write your interrupt code here ... */
    while(ADCSC1A_COCOA != 1) {}
    resultado = ADCRA;
    if(channel == 0x06)
    {
        read_adc = resultado;
    }
}
/* end of isrVadc */
```

Fonte: produção do próprio autor.

**Rotina de interrupção TPM1 e TPM2:** os periféricos TPM são "timers" do microcontrolador. No projeto os *timers* são utilizado para ser a base tempo dos contadores presentes no projeto.

Ilustração 66 – Código fonte presente no TPM1 que é utilizado pela função "Controle\_PWM".

```
/*
** =====
**      Interrupt handler : isrVtpm1ovf
**
**      Description :
**          User interrupt service routine.
**      Parameters : None
**      Returns     : Nothing
** =====
*/
_interrupt void isrVtpm1ovf(void)
{
    /* Write your interrupt code here ... */
    TPM1SC;
    TPM1SC_TOF = 0;
    //inicia a contagem
    if(Flag_Start_TPM1 == TRUE)
    {
        contador_TPM1++;
        if(contador_TPM1 >= timer)
        {
            contador_TPM1      = FALSE;
            Flag_Overflow_TPM1 = TRUE;
            Flag_Start_TPM1    = FALSE;
        }
    }
}
/* end of isrVtpm1ovf */
```

Fonte: produção do próprio autor.

Ilustração 67 – Código fonte do contador utilizado pela função "delay\_us" que esta presente no TPM1.

```
/*
** =====
**      Interrupt handler : isrVtpm2ovf
**
**      Description :
**          User interrupt service routine.
**      Parameters : None
**      Returns    : Nothing
** =====
*/
_interrupt void isrVtpm2ovf(void)
{
    /* Write your interrupt code here ... */
    /* Write your interrupt code here ... */

    TPM2SC;
    TPM2SC_TOF = 0;
    //contador da rotina da função delay
    contador_TPM2++;
    if(contador_TPM2 >= number)
    {
        contador_TPM2      = FALSE;
        Flag_Overflow_TPM2 = TRUE;
    }
}
```

Fonte: produção do próprio autor.

Ilustração 68 – Código fonte do contadores utilizado pela rotina de " Debounce" e "Atualiza\_LCD" que esta presente no TPM1.

```
//rotina que verifica se as chaves foram precionada
read_ch();
//contador da rotina que faz o debounce das chaves
if(Flag_Start_Contador_CH == TRUE)
{
    contador_CH++;
    if(contador_CH >= 50000)//100 mile segundos
    {
        Flag_Start_Contador_CH = FALSE;
        contador_CH = FALSE;
        Flag_Overflow_CH = TRUE;
    }
}
//Contador da rotina de controle LCD
if(Flag_Start_Contador_LCD == TRUE)
{
    contador_LCD++;
    if(contador_LCD >= 1000000)//50000000// 10 segundos
    {
        Flag_Overflow_LCD      = TRUE;
        Flag_Start_Contador_LCD = FALSE;
        contador_LCD          = 0;
    }
}
else if(Flag_Start_Contador_LCD |= TRUE)
{
    contador_LCD = 0;
}
```

Fonte: produção do próprio autor.

Ilustração 69 – Código fonte dos contadores para geração do PWM.

```

' //controle do inversor
// É necessário que essa rotina execute no tempo da interrupção
if(Flag_Start_PWM == TRUE)
{
    Controle_PWM(ON);
}

//Contador da rotina de partida do motor
if(Flag_Start_Contador_Motor == TRUE)
{
    contador_motor++;
    if(contador_motor >= 100)//valor a definir
    {
        contador_motor      = FALSE;
        Flag_Overflow_Motor = TRUE;
        Flag_Start_Contador_Motor = FALSE;
    }
}

// Contador da rotina de repouso do motor
if(Flag_Start_Repouso == TRUE)
{
    contador_repouso++;
    if(contador_repouso >= 1000000) //100 segundos - 10000000
    {
        contador_repouso      = FALSE;
        Flag_Start_Repouso     = FALSE;
        Flag_Overflow_Repouso  = TRUE;
    }
}

/* end of isrVtpm2ovf */

```

Fonte: produção do próprio autor.

**Função "main":** essa é a função principal do projeto sendo responsável por inicializar os periféricos do MCU, a biblioteca de inicialização do *display* do LCD, a tela de apresentação do *display* do LCD e a rotina de loop infinito. A rotina de loop infinito possui o algoritmo que gerencia principalmente o momento em que o equipamento é ligado, pois são necessário cinco ciclos para o cálculo da temperatura. Após esse cálculo o sistema é introduzido na máquina de estado que gerencia o sistema.

Ilustração 70 – Código fonte da função “main”.

```

void main(void)
{
    MCU_init(); //Inicializa perifericos do MCU
    //Inicializa função do display de LCD
    lcd_ini(display_8x5|2_linhas,display_ligado|cursor_desligado|cursor_fixo); //Função Display LCD
    //Habilita tela inicial do Display
    Tela_inicial();
    RELE = OFF; //Desliga Rele
    Flag_Motor_ON = FALSE; //Inicia o motor no estado desligado
    //Inicia o Loop Infinito
    for(;;)
    {
        //antes de iniciar o controle é feita medida temperatura
        if(controle_inicial < 6)
        {
            // Realiza aquisição e calculo da tensão do ADC
            Tensao_RMS();
            // Realiza o calculo da temperatura
            temperatura(temp_mV);
            // Set Flag para sair da rotina de repouso
            Flag_Overflow_Repouso = TRUE;
            // Incremente variavel de controle
            controle_inicial++;
        }
        else
        {
            //inicia rotina de controle do display
            Atualiza_LCD(); //Maq. de Estado Display LCD
            //Inicia aplicação/controle
            Application_Control(); //Maq. de Estado Aplicação
        }
    }
} ****

```

Fonte: produção do próprio autor.

**Rotina de IHM:** o IHM é composto por um *display LCD* alfanumérico 16x2 com *backlight* RGB e quatro chaves *push-buttons*. O algoritmo da interface/homem máquina é feito pelas funções "atualiza\_LCD" , "tela\_inicial" e "read\_ch".

**Função "Atualiza\_LCD":** além da tela de apresentação o sistema possui uma tela que apresenta a temperatura, uma tela de "*Setup*" onde é possível configurar a precisão da

refrigeração e outra que configura a cor de fundo do *display (backligh)*. O *setup* é acessado através dos *push-buttons* do hardware.

Essa função possui uma rotina de máquina de estado que é responsável pelo gerenciamento das telas do display e tratamento dos sinais provenientes dos push-buttons. De maneira geral cada estado da máquina executa a seguinte tarefa: escreve no display, realiza a checagem das teclas e dispara a contagem de tempo de cada tela.

Ilustração 71 – Código fonte da função “Atualiza\_LCD”.

```
//primeira tela lcd
case 1:
    sprintf(LCD_LINHA_1," PROJETO TCC " );
    sprintf(LCD_LINHA_2," CYCLO DEFROST ");
    lcd_pos_xy(1,1);
    lcd_escreve_string(LCD_LINHA_1);
    lcd_pos_xy(1,2);
    lcd_escreve_string(LCD_LINHA_2);
    //SET flag para iniciar contagem
    Flag_Start_Contador_LCD = TRUE;
    //Se ouver o estouro na contagem muda a msg do lcd
    if(Flag_Overflow_LCD == TRUE)
    {
        Flag_Overflow_LCD      = FALSE;
        estado_LCD             = 2; //muda para segunda tela LCD
    }
    //Se a chave um for precionada muda para SETUP
    //Zera contador_lcd
    if(Flag_ch1 == TRUE)
    {
        Flag_Start_Contador_LCD = FALSE;
        Flag_ch1                = FALSE;
        estado_LCD              = 3;
    }
    //Caso as outras chaves for precionada não deve ocorrer nenhuma alterção
    //zeras Flag das chaves
    if((Flag_ch2 == TRUE) || (Flag_ch3 == TRUE) || (Flag_ch4 == TRUE))
    {
        Flag_ch2                = FALSE;
        Flag_ch3                = FALSE;
        Flag_ch4                = FALSE;
        Flag_Start_Contador_LCD = FALSE;
    }
break;
```

Fonte: produção do próprio autor.

**Função "Tela\_inicial":** essa função é a primeira tela a ser apresentada quando é ligado.

Ilustração 72 – Código fonte da função “Tela\_inicial”.

```
void Tela_inicial(void)
{
    sprintf(LCD_LINHA_1," PROJETO TCC ");
    sprintf(LCD_LINHA_2," CYCLO DEFROST ");
    lcd_pos_xy(1,1);
    lcd_escreve_string(LCD_LINHA_1);
    lcd_pos_xy(1,2);
    lcd_escreve_string(LCD_LINHA_2);
    //Liga backlight com fundo azul
    BACKLIGHT_BLUE = ON;
}
```

Fonte: produção do próprio autor.

**Função "read\_ch":** essa rotina verifica de maneira periódica se alguns dos *push-buttons* foram pressionados. Caso alguma chave foi pressionada é realizado o algoritmo de "Debounce" para validar se a mesma foi acionada. O algoritmo de "Debounce" consiste em toda vez que o bit do registrador onde estão ligado os "*push-buttons*" mudarem de nível lógico (1 para 0) é habilitado um "Flag" que inicia um contador. Ao final da contagem é habilitado outro "Flag" e em seguida é verificado se não houve alteração no valor do registrador. Caso o registrador esteja em nível zero é habilitado o "Flag" que sinaliza que foi pressionado um botão na placa.

Ilustração 73 – Código fonte da função “push-bottons”.

```
//rotina que verifica de as chaves foram precionada
read_ch();
//contador da rotina que faz o debounce das chaves
if(Flag_Start_Contador_CH == TRUE)
{
    contador_CH++;
    if(contador_CH >= 50000)//100 mile segundos
    {
        Flag_Start_Contador_CH = FALSE;
        contador_CH = FALSE;
        Flag_Overflow_CH = TRUE;
    }
}
```

Fonte: produção do próprio autor.

Ilustração 74 – Código fonte da função “read\_ch”.

```

void read_ch(void)
{
    //testa chave 1
    if(CH_1 == 0)
    {
        LED_1 = ON;
        Flag_Start_Contador_CH = TRUE;
        if((Flag_Overflow_CH == TRUE) && (CH_1 == 0))
        {
            Flag_ch1          = TRUE;
            Flag_Overflow_CH = FALSE;
            LED_1 = OFF;
            teste_ch++;
        }
    }
    //testa chave 2
    else if(CH_2 == 0)
    {
        Flag_Start_Contador_CH = TRUE;
        if((Flag_Overflow_CH == TRUE) && (CH_2 == 0))
        {
            Flag_ch2          = TRUE;
            Flag_Overflow_CH = FALSE;
            teste_ch2++;
        }
    }
    //testa chave 3
    else if(CH_3 == 0)
    {
        Flag_Start_Contador_CH = TRUE;
        if((Flag_Overflow_CH == TRUE) && (CH_3 == 0))
        {
            Flag_ch3          = TRUE;
            Flag_Overflow_CH = FALSE;
        }
    }
}

```

Fonte: produção do próprio autor.

**Funções de leitura do ADC e Temperatura:** O algoritmo que realiza a leitura do ADC é dividido em diversas funções sendo elas: "channel\_ADC", "isrVadc", "converte\_mili\_volt" e "Tensao\_RMS".

- A função "channel\_ADC" repassa para o registrador do MUC qual o canal de ADC deve ser feita a leitura;
- A função "isrVadc" faz a leitura do ADC e repassa o valor da conversão para a variável "read\_adc".

- A função "converte\_mili\_volt" recebe o valor da conversão do ADC e realiza o cálculo matemático para converter em mili-volt.
- A função "Tensao\_RMS" realiza o cálculo de tensão RMS da leitura do ADC. Essa rotina possui uma máquina de estado, contendo seis estados, sendo os cinco primeiros responsáveis por realizar a amostragem da tensão do ADC, e o ultimo realiza a média da leitura e em seguida o cálculo da tensão RMS.

**Função "temperatura":** realiza o calculo da temperatura recebendo a variável "read\_adc" e em seguida calcula a temperatura usando uma expressão matemática.

Ilustração 75 – Trecho do código fonte com as funções "channel\_ADC", "converte\_mili\_volt", "temperatura".

```
/*
***** Função que de escolha do canal ADC *****
*****
int channel_ADC(uint8 adc)
{
    ADCSC1A_ADCHA = adc;
    channel = adc;
    return 1;
}
*****
***** Função que converte unidade do ADC para milivolts *****
*****
int converte_mili_volt(int dado)
{
    return (uint16) (((float)dado/4095)*3300) ;
}
*****
* Função Converte Tensão em Temperatura
*****
void temperatura(uint16 t)
{
    //return (uint16) ((float) (0.0307*t)-50);
    temp = (float) ((0.0307*t)-45);
    /* if(t<=2100)
       | temp = (float) ((0.0332*t)-50.343);
    else if((t>2100) || (t<=2290))
       | temp = (float) ((0.0281*t)-10.20);
    else if(t > 2290)
       | temp = (float) ((0.0644*t)-160.2); */
    temperature = (int16)temp;
}
```

Fonte: produção do próprio autor.

Ilustração 76 – trecho de código fonte "Tensao\_RMS".

```

channel_ADC(0x06);
x1          =(uint16)converte_mili_volt(read_adc);
calc_rms    = 2;
break;
case 2:
//Realiza a segunda amostra de tensão
channel_ADC(0x06);
x2          = (uint16)converte_mili_volt(read_adc);
calc_rms    = 3;
break;
case 3:
//realiza a terceira amostra de tensão
channel_ADC(0x06);
x3          = (uint16)converte_mili_volt(read_adc);
calc_rms    = 4;
break;
case 4:
//realiza a quarta amostra de tensão
channel_ADC(0x06);
x4          = (uint16)converte_mili_volt(read_adc);
calc_rms    = 5;
break;
case 5:
//Realiza a quinta amostra de tensão
channel_ADC(0x06);
x5          = (uint16)converte_mili_volt(read_adc);
calc_rms    = 6;
break;
case 6:
//Calcula a tensão RMS
temp_mV     = (uint16)((x1+x2+x3+x4+x5)/5);
temp_mV     = (uint16)sqrt((temp_mV*temp_mV));
calc_rms    = 1;
break.

```

Fonte: produção do próprio autor.

**Funções que compõem a rotina de geração do PWM:** o PWM adotado no projeto é "Modulação por Largura de Pulso Único". Esse método consiste em apenas um pulso por semiciclo tendo como parâmetro configurável a frequência, pois a tensão é dada através da largura do pulso. Para realizar esse controle são utilizadas as funções "Controle\_PWM" e "Tempo\_PWM".

- **Função "Tempo\_PWM":** possui a tarefa de realizar os cálculos das variáveis de controle do *PWM*. A função "Tempo\_PWM" recebe os parâmetros de frequência e tensão retornando as variáveis "t\_ON", "t\_OFF" e "safe\_time".

A variável "t\_ON" recebe o tempo de duração dos pulsos em microssegundos.

A variável "t\_OFF" recebe o tempo que o controle mantém em nível lógico zero e esse valor é dado em microssegundos.

A variável "safe\_time" recebe o tempo entre pulsos e esse valor é dado em milissegundos.

Primeiramente é calculado o valor do período e dividido por dois para determinar o tempo do semiciclo. Vinte e cinco por cento do valor desse resultado é destinado a variável "safe\_time" e o restante é destinado para o tempo *ON* e o tempo *OFF* do pulso.

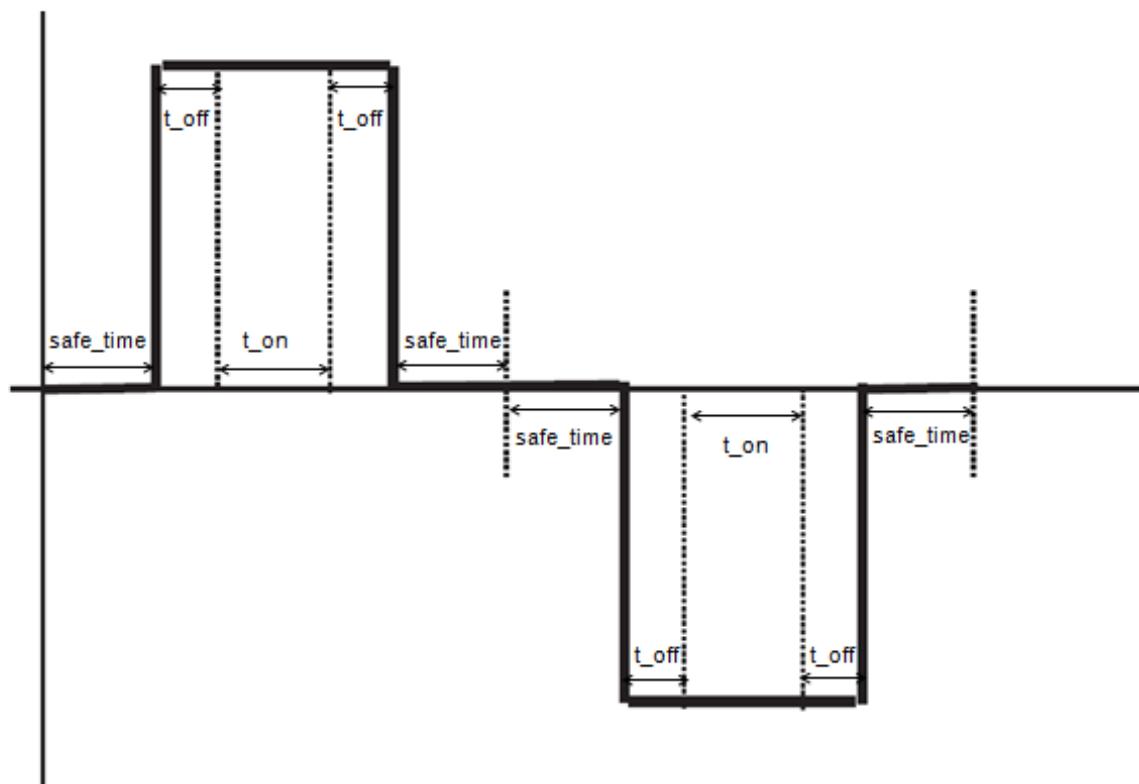
Ilustração 77 – Código fonte da função "Tempo\_PWM".

```
void Tempo_PWM(uint8 tensao, uint8 freq)
{
    float constante = 0.0;
    uint32 periodo = 0;
    //uint16 periodo_us = 0;
    //float constante2 = 0.0;
    //calcula pi(semi-ciclo da senoide)
    //periodo = (float)((1/valor_freq)/2);
    //periodo = (float)((float)(1/freq))/2;
    //calculo do periodo T = 1/f
    //é utilizado 1000000 converter em micro segundos
    periodo = (uint32)(1000000/freq); //1000000
    periodo = (uint32)(periodo/2);
    //Converte o valor para micro segundos
    periodo_us = (uint32)(periodo * 1000000);
    //calculo da constante de multiplicação
    constante = (float)((float)tensao/100);
    //constante2 = (float)(1 - constante);
    //Desconta 2 milisegundos para o tempo segurança
    //esse tempo é importante, pois é para evitar de colocar os transistor
    //de potencia em curto (25%)
    safe_time = (uint16)((float)periodo * 0.25)/2;
    periodo = ((uint16)periodo - safe_time);
    //calculo do tempo ON
    t_ON = (uint16)(periodo * constante);
    //calculo do tempo OFF
    t_OFF = (uint16)((periodo - t_ON)/2);
    //é necessário dividir por dois a interrupção é de micro segundos
    safe_time = (uint16)(safe_time/2);
    t_ON = (uint16)(t_ON/2);
    t_OFF = (uint16)(t_OFF/2);
}
```

Fonte: produção do próprio autor.

- **Função "Controle\_PWM":** essa função contém uma máquina de estado que controla a geração do PWM. O primeiro estagio é chamada a função "Tempo\_PWM" que repassa os parâmetros de frequência e tensão. Os próximos estados são para gerar os "safe\_time", "t\_OFF" e "t\_ON" atuando de maneira semelhante. Eles repassam a variável com o tempo, habilitam o "Flag" para iniciar o contador aciona os bits de controle (HIN\_1, LIN\_1, HIN\_2, LIN\_2).

Ilustração 78 – Forma de onda de Saída.



Fonte: produção do próprio autor.

Ilustração 79 – Trecho do código fonte da função "Controle\_PWM".

```

void Controle_PWM(uint8 estado)
{
    //state_PWM = estado;
    if(estado == TRUE)
    {
        switch(state_PWM)
        {
            case 1:
                //Tempo_PWM(Tensao,Frequencia);
                Tempo_PWM(target_tensao,target_freq);
                state_PWM = 2;
                break;
                //Inicia a geração dos pulsos
                //estado "safe_time"
            case 2:
                //repassa valor da variavel safe_timer para iniciar a contagem
                timer = safe_time;
                //SET contador
                Flag_Start TPM1 = TRUE;
                //SET os BIT de controle do diver IR3101
                //Drive 1 : Alta Impedancia
                //Drive 2 : Alta Impedancia
                //safe_transition();
                HIN_1 = OFF;
                LIN_1 = ON;
                HIN_2 = OFF;
                LIN_2 = ON;
                //RELE = ON;
                if(Flag_Overflow TPM1 == TRUE)
                {
                    Flag_Overflow TPM1 = FALSE;
                    state_PWM = 3;
                }
        }
    }
}

```

Fonte: produção do próprio autor.

O fluxograma da lógica da geração do *PWM* está presente no *Anexo A*.

**Algoritmo de gerenciamento do sistema:** para gerenciar o sistema foram criadas as funções "PID" e "Application\_Control" . A função "PID" recebe os parâmetros que o usuário configura no "SETUP" e retorna os valores de frequência e tensão que o projeto deve executar. A função "Application\_Control" tem como objetivo coordenar todo sistema. O algoritmo possui uma máquina de estado que gerencia o acesso do periféricos do sistema leitura do ADC, o calculo da tensão, a conversão da temperatura, o algoritmo de PID e repassa os parâmetros do *PWM*.

Ilustração 80 – Trecho do código fonte da função "PID".

```
void PID(uint16 position, float temperature)
{
    uint16 temperatura = 0;
    //converte a temperatura para numero intero e adiciona um off set de 3
    temperatura = (uint16)(3+temperature);
    //Maq. de Estado controle PID
    switch(position)
    {
        case 1:
            // se temperatura for menor do que -3°C
            // desliga compressor
            if(temperatura <= 0)
            {
                RELE          = OFF;
                Flag_Motor_ON = FALSE;
                // RESET Flag para ligar o motor
                Flag_Start_Motor = FALSE;
                // SET a Flag para contar o tempo de repouso
                Flag_Start_Repous = TRUE;
                //frequencia      = 140;
                //p_tensao        = 100;
            }
            else if(temperatura > 0 && temperatura <= 1)
            {
                frequencia      = 114;
                p_tensao        = 80;
                // SET Flag para ligar o motor
                Flag_Start_Motor = TRUE;
            }
            else if(temperatura > 1 && temperatura <= 2)
            {
                frequencia      = 114;
                p_tensao        = 82;
                // SET Flag para ligar o motor
            }
    }
}
```

Fonte: produção do próprio autor.

Ilustração 81 – Trecho do código fonte.

```
void Application_Control(void)
{
    switch(controle_estado)
    {
        case PARADO:
            Tensao_RMS();
            controle_estado = TEMPERATURA;
            break;
        case TEMPERATURA:
            temperatura(temp_mV);
            controle_estado = READ_CH;
            break;
        case READ_CH:
            //key_processes();
            //as ações a respeito das leituras da chave estão função Controle_LCD
            controle_estado = DECISION;
            break;
        case DECISION:
            //if(temp target_temperature)
            PID(posicao_termo,temp);
            controle_estado = MOTOR_CONTROL;
            break;
        case MOTOR_CONTROL:
            // Para ser ligado o motor é necessário três condições
            if((Flag_Motor_ON == FALSE) && (Flag_Start_Motor == TRUE) )//&& (Flag_O
            {
                // rotina de partida do motor
                motor_starting();
                RELE = ON;
                Flag_Overflow_Reposo = FALSE;
            }
            pass_parameters();
            //Controle_PWM(ON);
            Flag_Start_Motor = TRUE;
    }
}
```

Fonte: produção do próprio autor.

## 7. Testes de validação de Hardware

O circuito com o microcontrolador utilizado no desenvolvimento do protótipo foi disponibilizado pela empresa *Biosensor*.

Os primeiros testes realizados com o hardware foram conforme a montada da placa. Primeiramente foram montados e testados os circuitos dos retificadores, e não apresentaram nenhum tipo de problema em seu funcionamento. Em seguida foram montados os circuitos IHM, LCD e *push-bottom*. Esses circuitos foram testados em conjunto com a placa do microcontrolador.

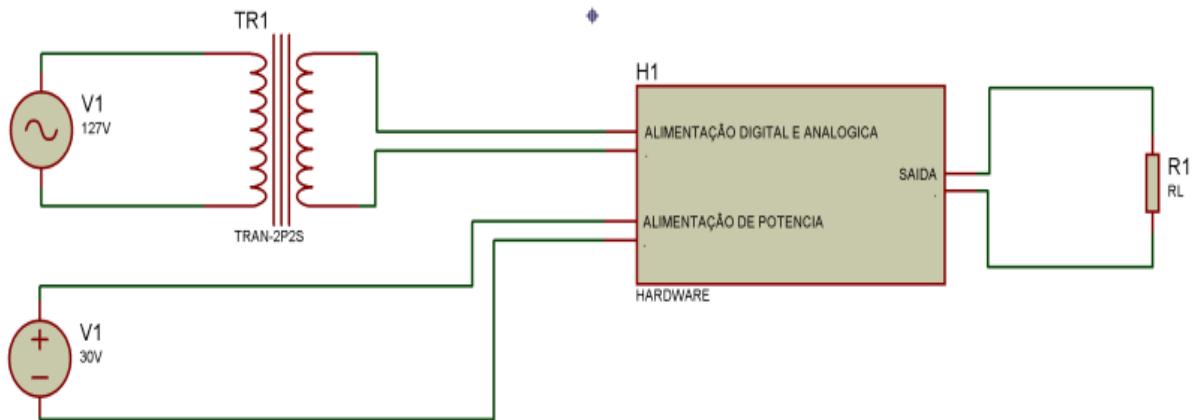
O circuito do transdutor de temperatura inicialmente foi montado com ponte de *Winston*, amplificador instrumental e circuito do *buffer*. Mas ao realizar os testes em conjunto com o MCU, a leitura do canal do ADC do microcontrolador era instável, então foi adotado outro circuito para realizar a medida da temperatura. Esse novo circuito possui uma configuração mais simples em relação ao anterior, pois consiste em apenas em um divisor de tensão, que tem a finalidade de variar a sua tensão de saída conforme a temperatura. Após essa modificação verificou-se um resultado com maior estabilidade na leitura do ADC comparada a configuração anterior.

O circuito do inversor é composto basicamente por acopladores ópticos e por *driver* de potência. Foram montados os acopladores e o *drive* de potência respectivamente. O CI utilizado foi FSB50250, porém o mesmo não funcionou adequadamente, consequentemente foram avaliados outros componentes para substituí-lo. Foram escolhidos dois modelos IR3101 e IRAM10UP60A.

O primeiro componente a ser montado e testado foi IR3101. Os testes realizados inicialmente foram com baixa potência utilizando uma fonte de bancada para prevenir a queima de componente.

A fonte de bancada era conectada ao circuito do retificador que alimenta o circuito de potência que por sua vez alimenta o inversor. Na saída ao invés de ligar o motor era utilizado um resistor como carga. Essa configuração foi a qual obteve o melhor resultado, possibilitando dar continuidade do desenvolvimento do software embarcado.

Ilustração 82 – Montagem do *hardware*.



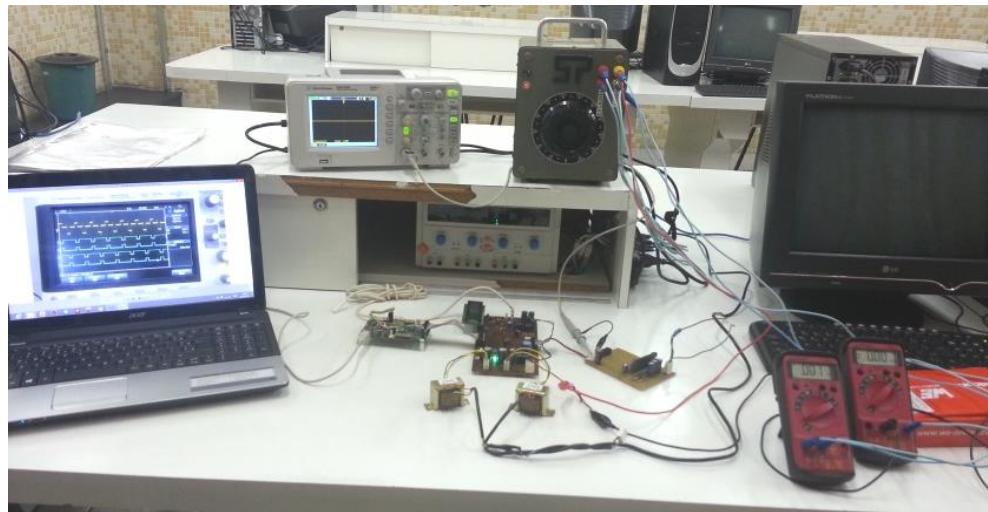
Fonte: *Software Proteus*.

Após iniciarem os testes que necessitavam de maior demanda de potência um dos componentes começou a falhar levando a sua queima.

Os testes com potência elevada foram realizados após a troca da fonte de bancada por um Variac que possibilitava o controle manual da tensão. Nesse teste ao atingir o valor de 60V no Variac o *driver* IR3101 começou a apresentar alguns defeitos levando, consequentemente, a sua queima.

Como não havia outro componente para substituição imediata, foram iniciados os testes com a segunda opção de CI o *driver* IRAM10UP60A. Com esse componente foi possível realizar as correções de software e hardware que levaram o CI anterior a queima, porém foi necessário a montagem de uma placa aparte devido a esse componente. Em paralelo a montagem do *driver* do IRAM foram comprados mais algumas unidades do componente IR3101. Após todas as modificações necessárias, os mesmos testes foram replicados.

Ilustração 83 – Circuito montado com IRAM.



Fonte: produção do próprio autor.

O *driver* IRAM10UP60A também apresentou alguns avarias quando submetido aos testes com maior potência, consequentemente levando-o a sua queima. Como medida corretiva foi trocado novamente o componente IRAM pelo IR3101

Ilustração 84 – Circuito montado com IR3101.



Fonte: produção do próprio autor.

Com essa substituição foi possível fornecer na saída do inversor tensão de aproximadamente 110V RMS com frequências ajustáveis. Porém o *driver* apresentou aquecimento e ruídos nos sinais de PWM. No intuito de tentar corrigir o problema, foram feitas algumas alterações no *software* que eliminaram os ruídos, mas levou a queima do componente.

Ilustração 85 – Sinal de saída após queima do componente.



Fonte: produção do próprio autor.

Atualmente foi feita a troca do CI danificado e realizado novas correções no *software* para replicar os mesmos testes citados acima.

Como próximo passo após todas as correções no *hardware* e *software* o mesmo deverá ser testado no refrigerador para avaliar o real desempenho para poder comparar seouve alguma melhoria em relação ao modelo atual.

## 8. Levantamento de Custo do *Hardware*

A tabela abaixo representa o valor agregado à produção do *hardware*. O levantamento de custo inicialmente foi feito para um lote de mil unidades com o preço correspondente em dólares.

Tabela 7 - Custo dos componentes do *hardware*.

Levantamento de custo PCI			
Componentes	QTD	CUSTO p/ 1000UN US\$	US\$
Resistores			
1.5K	1	0,005440	0,005440
1M	1	0,005440	0,005440
10K	23	0,005440	0,125120
330R	3	0,005440	0,016320
27R	1	0,005440	0,005440
10M	1	0,005440	0,005440
4.7K	1	0,005440	0,005440
1k	9	0,005440	0,048960
1R	3	0,005440	0,016320
3k3	1	0,005440	0,005440
33R	2	0,005440	0,010880
TRIMMER 10K	1	0,555000	0,555000
TRIMMER 100R	1	0,555000	0,555000
270R	1	0,005440	0,005440
51R	4	0,005440	0,021760
CAPACITORES			
1uF	7	0,115000	0,805000
10uF	4	0,115000	0,460000
100nF	11	0,087000	0,957000
1nF	15	0,087000	1,305000
22pF	2	0,087000	0,174000
470uF	3	0,087000	0,261000
100uF	2	0,087000	0,174000
18pF	2	0,087000	0,174000
10nF	4	0,087000	0,348000
22uF	3	0,087000	0,261000
CIRCUITO INTEGRADO			
MCF51MM256	1	6,560000	6,560000
LP2950	5	0,485000	2,425000
L7815	1	0,240720	0,240720
L7812	1	0,249900	0,249900
L7912	1	0,245280	0,245280

LT1013	1	0,968500	0,968500
AD623	1	1,982800	1,982800
MC9S08JM16CLD	1	2,000000	2,000000
TCMT4100	1	0,970000	0,970000
74LVC1T45	1	0,208000	0,208000
TRANSISTOR			
BC848	4	0,047000	0,188000
MMBT3904	1	0,024720	0,024720
IR3101	2	3,383500	6,767000
1N4007	2	0,026200	0,052400
1N4148	13	0,023180	0,301340
OUTROS			
MB1RC2 - RELE	1	2,200000	2,200000
LCD 16X2	1	7,000000	7,000000
CONECTOR USB	1	1,500000	1,500000
CRISTAL 16MHZ	1	0,560000	0,560000
CRISTAL 4MHZ	1	0,560000	0,560000
FERRITE BEAD	10	0,040000	0,400000
LED 0805	2	0,010000	0,020000
			0,000000
PLACA DE CIRCUITO	1	15,000000	15,000000
	TOTAL		\$ 56,73
Esse levantamento de custo não leva em consideração o custo das montagem das placas.			
O drive de potência utilizado neste documento é o IR3101.			
Fornecedores Cotados - Farnell, Digikey			

Fonte: produção do próprio autor.

## 9. Conclusão

Conclui-se que através dos testes realizados no refrigerador *Cycle Defrost* foi possível determinar o local adequado para o posicionamento do sensor de temperatura que substituirá o termostato.

Comprovou-se também que o inversor de frequência desenvolvido atingiu os requisitos necessários para tornar o projeto viável, tanto relacionado ao seu custo quanto ao seu funcionamento..

Através de testes realizados em laboratório verificou-se que para a validação do *hardware*, tanto os *drivers* IRAMS10UP60 e o IR3101 foram satisfatórios para o controle da velocidade do motor, porém, devido ao baixo custo agregado ao componente, o *driver* escolhido foi o IR3101.

Obteve-se bons resultados em relação ao funcionamento da IHM, do circuito de aquisição de temperatura, como também a interação do microcontrolador com o *software* via OSBDM e o restante do circuito do *hardware*.

Exigiu-se um determinado tempo que foi utilizado entre os estudos, o desenvolvimento e os testes de validação para o refrigerador, o *hardware* e o *software*, porém, o que demandou um maior período de trabalho foram os testes realizados no *hardware* e no desenvolvimento do *software*. Em consequência, justamente devido a falta de tempo hábil, não foi possível realizar os testes finais no refrigerador.

## REFERÊNCIAS

FRANCHI, Claiton Moro. **Acionamentos Elétricos**. 4. Ed. São Paulo: Érica, 2010. 250 p.

FRANCHI, Claiton Moro. **Inversores de Frequência: Teoria e Aplicações**. 2. Ed. São Paulo: Érica, 2013. 192 p.

STOECKER, W. F.; JABARDO, J. M. Saiz. **Refrigeração Industrial**. São Paulo: Edgard Blücher, 1997. 453 p.

RASHID, Muhammad H. **Eletrônica de Potência Circuitos, Dispositivos e Aplicações**. Tradução Carlos Alberto Favato; Revisão Técnica Pertence Júnior. São Paulo: Makron Books, 1999. 820 p.

GRENNING, James W. Test-Driven Development for Embedded C. Dallas, Texas: The Pragmatic Bookshelf, 2011. 360 p.

SISTEMA EMBARCADO LIVRE. Práticas Para a Qualidade – Os 4 Tipos de Módulos da Firmware. Felipe de Andrade Neves lavratti. Disponível em: <http://selivre.wordpress.com/2013/01/12/praticas-para-a-qualidade-os-4-tipos-de-modulos-da-firmware/#more-593>. Acesso em: 27 Mai. 2013.

FARIRCHILD. FSB50250US Motion SPM 5 FRFET Series. Disponível em: <http://www.fairchildsemi.com/ds/FS/FSB50250US.pdf> Acesso em 20 Abr. 2013.

INTERNATIONAL RECTIFIER. IR3101. Disponível em : <http://www.irf.com/product-info/datasheets/data/ir3101.pdf>. Acesso em: 25 Jun. 2013

INTERNATIONAL RECTIFIER. IRAMS10UP60A. Disponível em: <http://www.irf.com/product-info/datasheets/data/irams10up60a.pdf>. Acesso em: 20 Jun. 2013

INTERNATIONAL RECTIFIER. Reference Design IRADK31 revA . Disponível em: <http://www.irf.com/product-info/datasheets/data/irams10up60a.pdf>. Acesso em: 20 Jun. 2013

INTERNATIONAL RECTIFIER. Application Note NA-1044 revA . Disponível em: <http://www.irf.com/technical-info/appnotes/an-1044.pdf>. Acesso em: 22 Jun. 2013

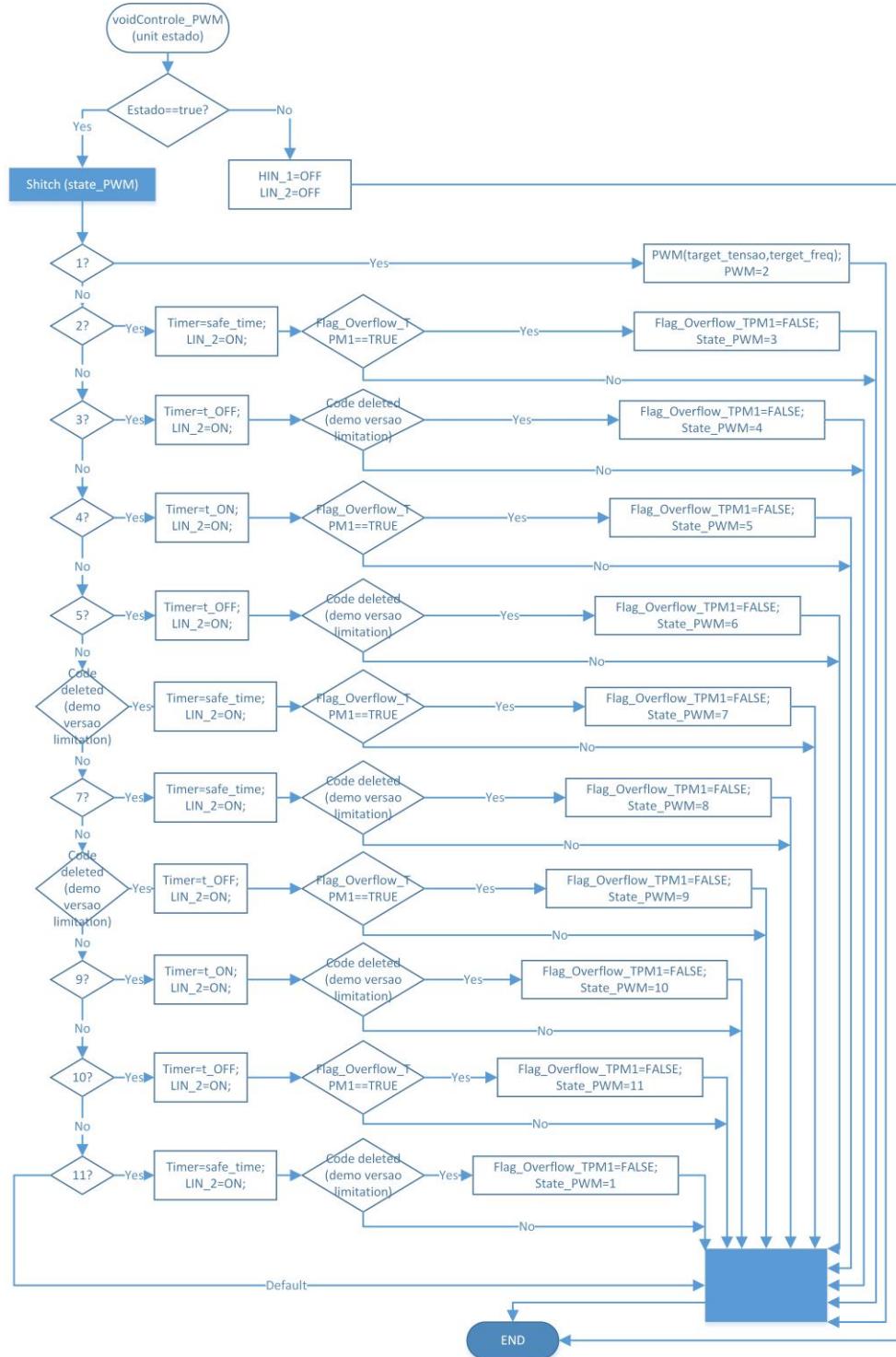
FREESCALE. MCF51MM256 Reference Manual. Disponível em:  
[http://cache.freescale.com/files/32bit/doc/ref\\_manual/MCF51MM256RM.pdf?fpst=1](http://cache.freescale.com/files/32bit/doc/ref_manual/MCF51MM256RM.pdf?fpst=1). Acesso em: 20 Abr. 2013.

FREESCALE. TWR-MCF51MM User Manual. Disponível em:  
[http://cache.freescale.com/files/microcontrollers/doc/user\\_guide/TWR-MCF51MMUM.pdf?fpst=1](http://cache.freescale.com/files/microcontrollers/doc/user_guide/TWR-MCF51MMUM.pdf?fpst=1). Acesso em: 20 Abr. 2013.

PEMICRO. Embedded OSBDM. Disponivel em: <http://www.pemicro.com/osbdm/index.cfm>.  
Acesso em 20 Abr. 2013

## ANEXO A – FLUXOGRAMA PWM

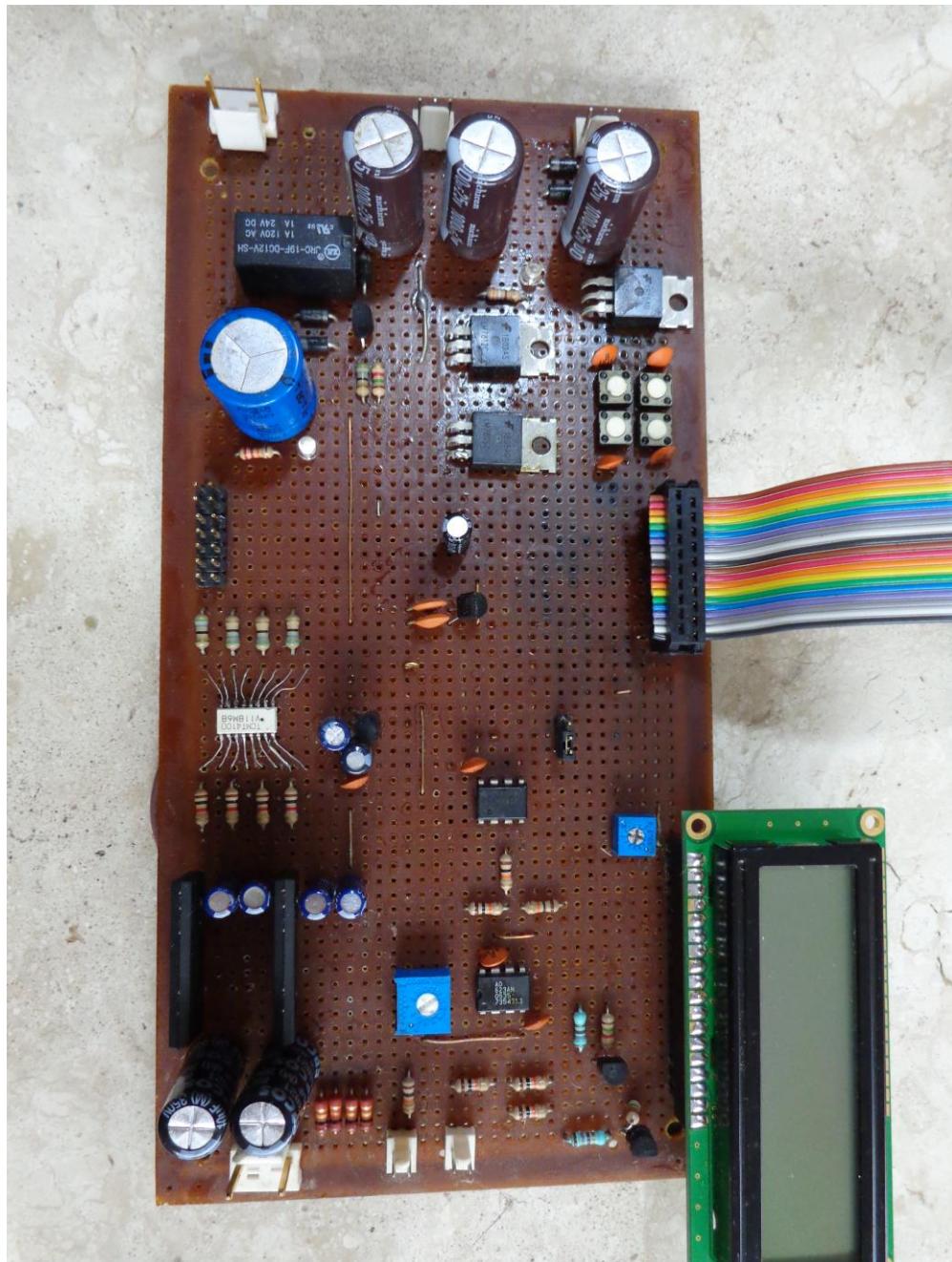
Ilustração 86 – Fluxograma lógica geração PWM.



Fonte: produção do próprio autor.

## ANEXO B – HARDWARE 1

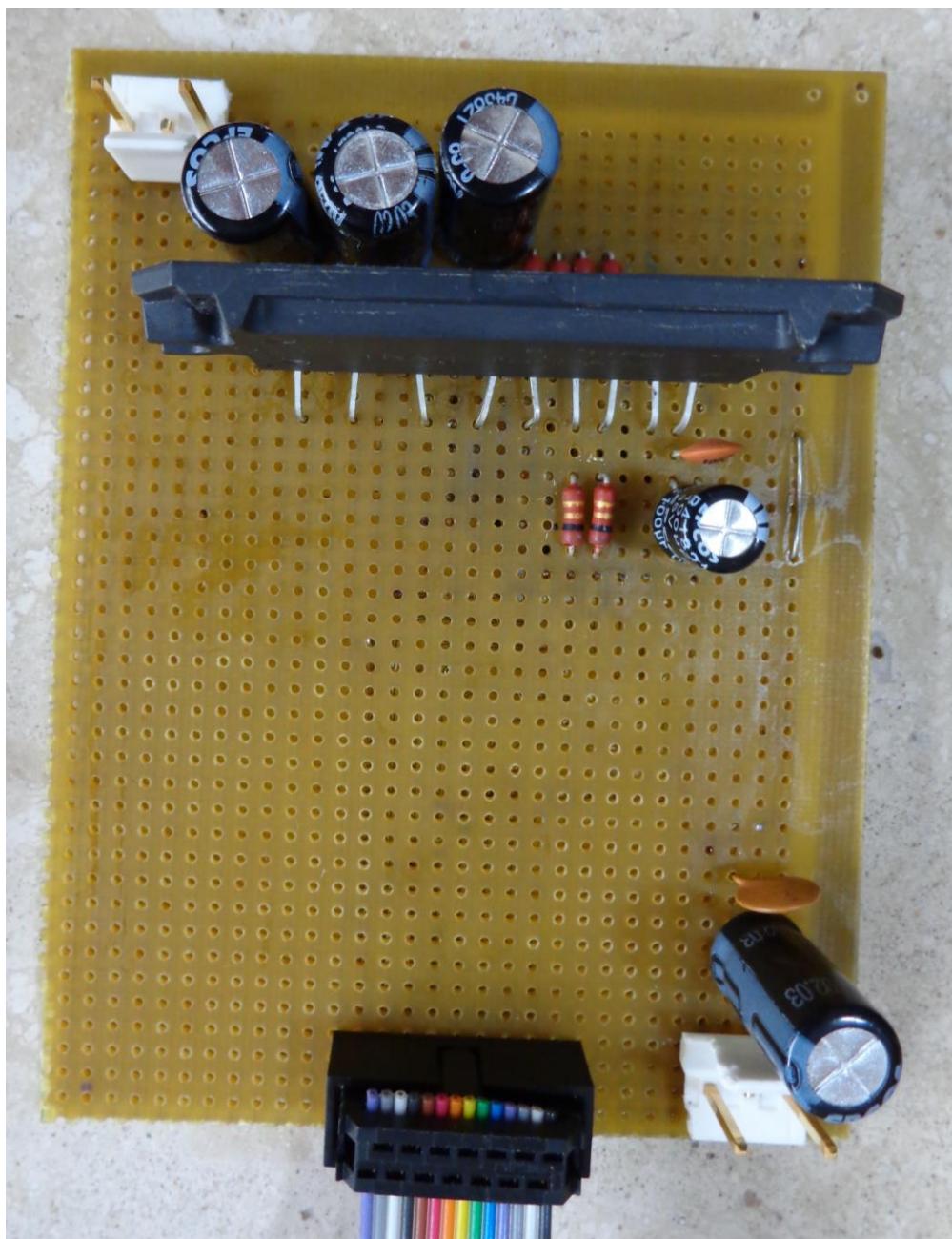
Ilustração 87 – Parte do *Hardware*.



Fonte: produção do próprio autor.

**ANEXO C – HARDWARE 2**

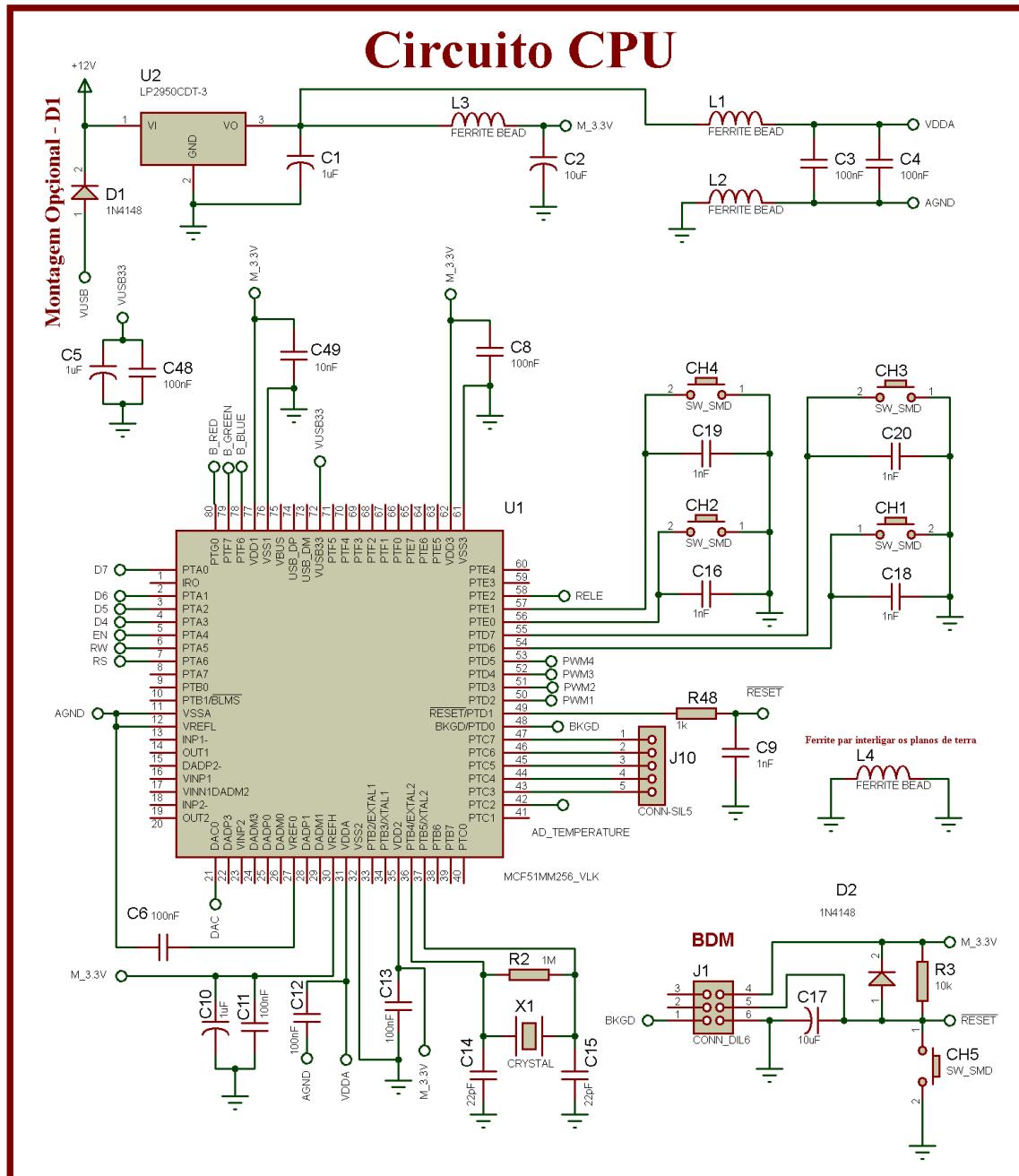
Ilustração 88 – Parte do *Hardware*.



Fonte: produção do próprio autor.

## ANEXO D – CIRCUITO CPU.

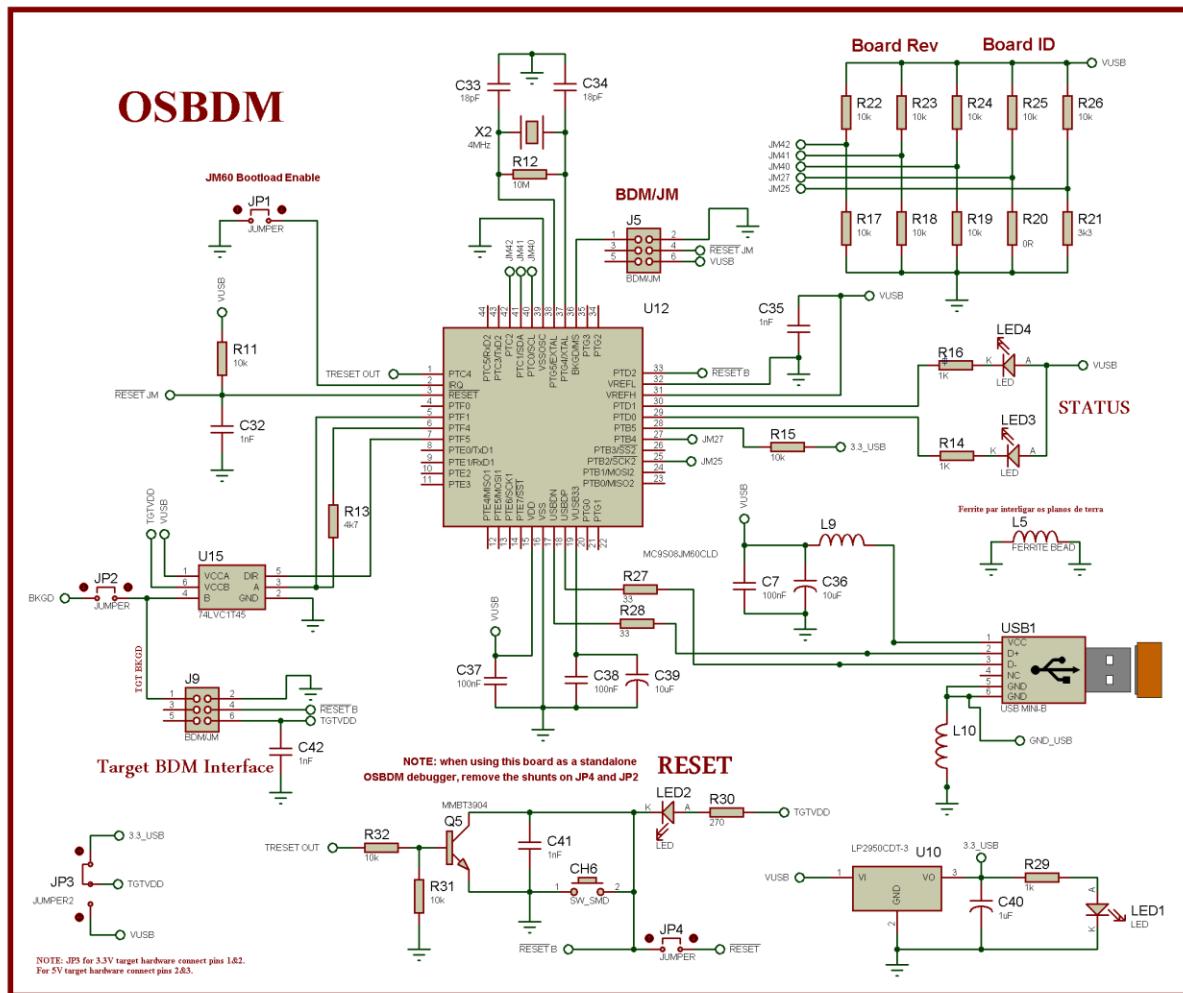
Ilustração 89 – Circuito CPU.



Fonte: *Software Proteus*.

## ANEXO E – OSBDM (Open Source BDM).

Ilustração 90 – OSBDM.



Fonte: *Software Proteus*.