28 de Junho de 2013

COMUNICAÇÃO POR RADIO FREQUÊNCIA PARA CONTROLADORES LÓGICOS PROGRAMÁVEIS (CLP)

Carlos Magno Catharino Olsson Valle

PUC RIO

COMUNICAÇÃO POR RADIO FREQUÊNCIA PARA CONTROLADORES LÓGICOS PROGRAMÁVEIS (CLP)

Aluno(s): Carlos Magno Catharino Olsson Valle

Orientador(es): Moisés Henrique Szwarcman

Trabalho apresentado com requisito parcial à conclusão do curso de Engenharia de Controle e Automação na Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil

PUC RIO

Agradecimentos

Agradeço a todos que me auxiliaram na confecção deste trabalho, em especial à minha família, que sempre esteve ao meu lado em todos os momentos da vida e acredito que sempre estará.

Resumo

Neste trabalho de conclusão de curso pretende-se adaptar, de modo genérico e a um custo relativamente baixo, CLPs a comunicar-se por meio de módulos de radio frequência de modo a reduzir a quantidade de fios necessária para sua instalação e comunicação. Pretende-se também implementar um exemplo funcional onde o CLP “Micrologix 1100” é utilizado para controlar um robô paralelo composto por quatro atuadores pneumáticos que direcionam um painel de células fotovoltaicas de modo a obter o melhor desempenho em relação a posição do sol.

Palavras-chave: CLP; Wireless; Célula Fotovoltaica; Robô Paralelo

RADIO FREQUENCY COMMUNICATION FOR A PROGRAMABLE LOGIC COMTROLLER (PLC)

Abstract

In this work we intend to adapt, in a generic way and relatively low cost, PLCs to communicate via radio frequency modules to reduce the amount of wiring required for installation and communication. We also intend to implement a practical example where the PLC "Micrologix 1100" is used to control a parallel robot consists of four pneumatic actuators that drive a panel of photovoltaic cells in order to obtain the best performance over the sun's position.

Keywords: PLC; Wireless; Photovoltaic Cell; Parallel Robot

Sumário

Introdução 8

Objetivo 9

1. Sistemas De Comunicação Por Radio Frequência 10

1.1. Fundamentos 10

1.2. Espectro eletromagnético 11

1.3. Radiofrequências 12

1.4. Bandas de radiofrequência 12

2. Definição das Ferramentas de Trabalho 14

2.1. Controlador Lógico Programável (CLP) 14

2.1.1. Protocolo RS-232 15

2.2. Módulo de radiofrequência 16

2.2.1. NRF24L01+ 16

2.2.2. Protocolo SPI (*Serial Peripherereal Interface*) 19

2.2.3. Microcontrolador 20

3. Implementação Pratica da Comunicação Serial 21

4. Aplicação da Comunicação Serial 25

4.1. Robôs paralelos 25

4.2. Programas e linguagens utilizados 26

4.3. Implementação do projeto 27

4.3.1. Circuito 27

4.3.2. Rendimento dos painéis solares 31

5. Conclusão 32

6. Referências 33

7. Apêndices 34

7.1. Código do microcontrolador utilizado para comunicação simples 34

7.1.1. Microcontrolador que recebe dados do computador 34

7.1.2. Microcontrolador que recebe dados do CLP 41

7.2. Código do microcontrolador utilizado na comunicação *full-duplex* 47

7.2.1. Microcontrolador que recebe dados do computador 47

7.2.2. Microcontrolador que recebe dados do CLP 54

Introdução

A automação, tanto industrial como residencial, tem evoluído a passos largos, oferecendo cada vez mais precisão e confiabilidade. Mas vemos que, apesar disso, ainda existe a necessidade de um sistema de comunicação por fio, dos computadores aos controladores. Esta necessidade eleva o custo de instalação dos equipamentos, já que se torna um problema a disposição dos fios no ambiente e geralmente é necessário realizar adaptações no layout da planta de fabricação para ocultá-los.

Com sistemas de comunicação sem fio, as plantas de produção poderiam mudar de modo a permitir que o espaço seja otimizado, o que potencialmente possibilita o aumento da produção e consequentemente maior rendimento financeiro à empresa. As automações residenciais poderiam ser feitas sem a necessidade de incomodas obras estéticas e/ou passagem de dutos, tornando o custo total do projeto muito mais baixo e incentivando o fomento do mercado.

Se levarmos em conta que com a comunicação sem fio os controladores tendem a ficar muito mais próximos dos sistemas atuados o passo seguinte seria eliminar a fonte de alimentação externa do sistema como um todo. Se este problema fosse resolvido, seria possível ter módulos totalmente fechados de automações possibilitando o desenvolvimento e a aplicação de novos conceitos de fabricação.

Neste trabalho apresentaremos uma aplicação da comunicação sem fio em uma situação de automação autossuficiente a base de energia solar.

Objetivo

Este trabalho tem como objetivo estudar e implementar um sistema capaz de controlar e monitorar um CLP remotamente, por meio de um circuito em cada extremidade (neste caso um computador e um CLP) capaz de receber e transmitir dados a partir de um módulo de radio frequência. Pretende-se também que o sistema de comunicação possua o menor custo possível, possibilitando ser aplicado em larga escala nas automações industriais ou residenciais.

1. Sistemas De Comunicação Por Radio Frequência

1.1. Fundamentos

As ondas de radio ou radiofrequências são campos eletromagnéticos utilizados nas comunicações sem fio. Como essas ondas levam energia de um ponto ao outro, isso permite a comunicação sem a necessidade de fios, como nas transmissões de televisão, radio e celulares.

Radiofrequência são sinais que se propagam por um condutor cabeado, normalmente cobre, e são irradiados no ar através de uma antena. Uma antena converte um sinal do meio cabeado em um sinal *wireless* (sem fio) e vice-versa. Os sinais irradiados no ar livre, em forma de ondas eletromagnéticas, propagam-se em linha reta e em todas as direções [1].

Uma onda é uma perturbação ou variação que transfere energia progressivamente de ponto a ponto num meio e pode sofrer uma deformação elástica ou uma variação de pressão, intensidade elétrica ou magnética, potencia elétrica, ou temperatura.

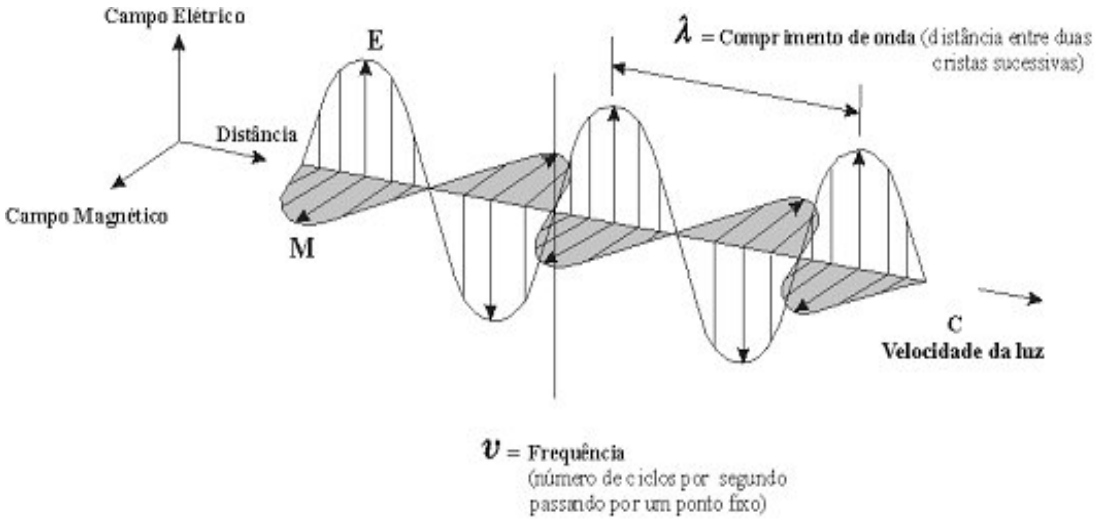

Figura 1 - Demonstração gráfica de uma onda eletromagnética

A radiofrequência utiliza ondas eletromagnéticas (Figura 1) que não requerem meio material para se propagar, ou seja, podem se propagar no vácuo, ar, água e alguns sólidos.

1.2. Espectro eletromagnético

O espectro eletromagnético é o intervalo completo da radiação eletromagnética (ondas de rádio, micro-ondas, infravermelho, a luz visível, ultravioleta, os raios x, até à radiação gama), ou seja, a distribuição da intensidade da radiação eletromagnética em relação ao seu comprimento de onda ou frequência [1].

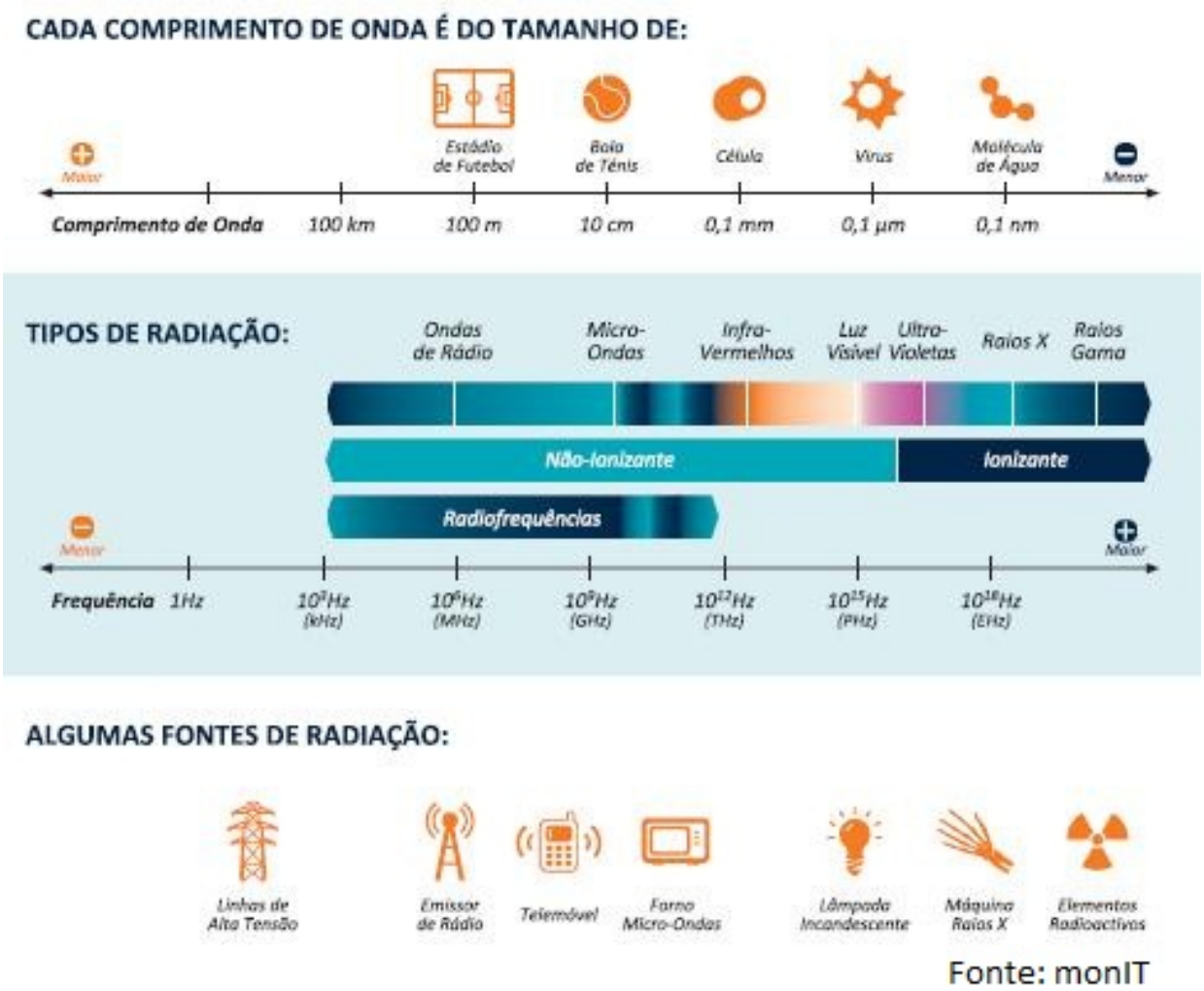

Figura 2 - Espectro eletromagnético

1.3. Radiofrequências

A parte de radiofrequência do espectro eletromagnético ocupa as frequências entre os 3 kHz e os 300 GHz.
As diferentes bandas de radiofrequências (frequências admissíveis 3 kHz a 300 GHz) são constituídas por diferentes intervalos de frequência[1], isto é, diferentes larguras de banda. Estes intervalos são representados na Figura 3.

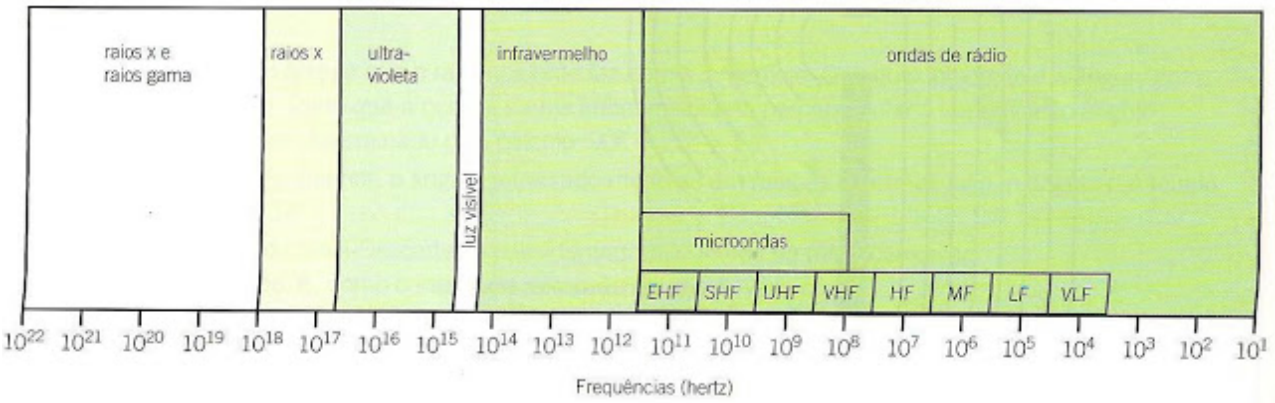

Figura 3 - Intervalos de Frequência das bandas de radiofrequência

1.4. Bandas de radiofrequência

NOME	SIGLA	BANDA DE FREQUÊNCIA	COMPRIMENTO DE ONDA	UTILIZAÇÕES
Frequências extra-baixas	ELF	30 Hz-3 kHz	10 000-1 000 000 m	Comunicação com submarinos.
Frequências muito baixas	VLF	3-30 kHz	100 000-10 000 m	Navegação e fins militares.
Frequências baixas	LF	30-300 kHz	10 000-1000 m	Navegação e estações de rádio internacionais.
Frequências médias	MF	300-3000 kHz	1000-100 m	Estações de rádio nacionais.
Frequências altas	HF	3-30 MHz	100-10 m	Estações de rádio e radiotelefone.
Frequências muito altas	VHF	30-300 MHz	10-1 m	Estações de rádio em FM e estações de televisão.
Frequências ultra-altas	UHF	300-3000 MHz	1-0,1 m	Estações de televisão, telemóveis e controlo aéreo por radar.
Frequências superaltas	SHF	3-30 GHz	0,1-0,01 m	Telemóveis, radar, satélites de comunicação e GPS.
Frequências extra-altas	EHF	30-300GHz	0,01-0,001 m	Estações espaciais.

Figura 4 - Bandas de radiofrequência

As principais propriedades das diversas bandas(apresentadas na Figura 3), assim como os fenômenos ondulatórios a que estão sujeitas durante a sua propagação e transporte de informação, são as seguintes:

- As ondas de rádio de médias e baixas frequências ELF a MF, possuem grandes comprimentos de onda, sendo por isso designadas por ondas médias e longas. Elas são as que melhor se difratam na atmosfera, contornando facilmente obstáculos de grandes dimensões, e acompanhando a curvatura terrestre até alguns milhares de quilômetros. O motivo desse comportamento é porque são pouco absorvidas pelo ar e são refletidas pela estratosfera. À medida que a frequência aumenta, diminui a capacidade de transmitir para distâncias muito longas ao nível da superfície terrestre. Desta forma, as ondas MF são as ondas utilizadas nas estações nacionais, permitindo difundir o som com maior qualidade, embora com menor alcance; já as LF são utilizadas nas estações de rádio que transmitem a nível mundial e as ELF, que têm frequências extra baixas, são utilizadas, por exemplo, nas comunicações entre submarinos, porque são as únicas capazes de penetrar em profundidade no mar[1].

- As Ondas de Rádio de elevadas frequências, HF e VHF, possuem pequenos comprimentos de ondas, geralmente sofrendo múltiplas reflexões na ionosfera e na superfície terrestre, não conseguem acompanhar a curvatura da Terra. Desta forma, elas são utilizadas em comunicações que não exigem grande alcance. No entanto, por serem geralmente moduladas em frequência (FM), são muito utilizadas quando é exigida a alta qualidade de som e imagem[1].

- As Ondas Eletromagnéticas com frequências ultraelevadas, UHF, superaltas, SHF e extra-altas, EHF, como as micro-ondas, são pouco absorvidas e/ou refletidas na atmosfera. Como praticamente não sofrem difração, propagam-se em linha reta. Uma vez que podem atravessar a ionosfera, são utilizadas nas comunicações via satélite, não sendo esta a única razão. Dado que a frequência a transmitir é muito elevada, a largura de banda tem de ser elevada, o que implica ondas portadoras de elevada frequência[1].

2. Definição das Ferramentas de Trabalho

2.1. Controlador Lógico Programável (CLP)

O CLP é muito utilizado em automações industriais, sua função geralmente é de controlar e monitorar uma automação completa ou parcialmente. Para isso, ele dispõe tanto de portas que funcionam como entradas e saídas de sinais digitais quanto de portas de sinais analógicos de modo que o sistema seja capaz de trabalhar com praticamente todos os tipos de sensores voltaicos.

Para que todas estas portas tenham funcionalidade, o CLP dispõe de um circuito que contem um ou mais microcontroladores programáveis. Geralmente, os microcontroladores são programáveis com codificação hexadecimal (Linguagem de Máquina) do programa a ser implementado. Tal codificação é possível devido a um processo de tradução de uma linguagem de nível superior para uma de nível inferior. A Figura 5 ilustra em detalhe o processo,onde a compilação é o nome utilizado para esta tradução da linguagem de nível superior para a de nível inferior.

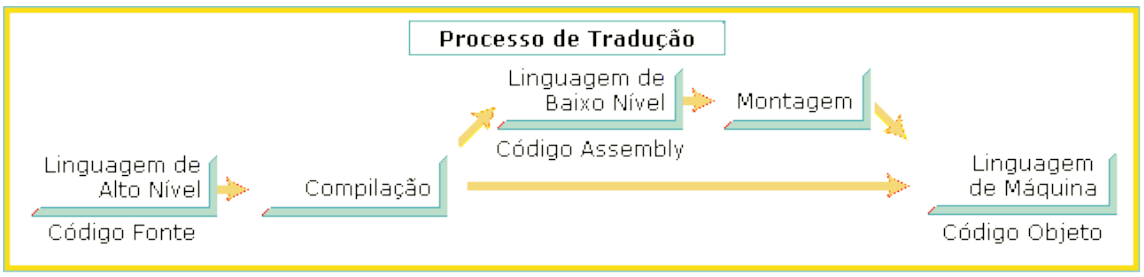

Figura 5 - Processo de Tradução

Na aplicação desenvolvida neste projeto, será utilizado o CLP "Micrologix 1100" fornecido pela faculdade. Ele possui um modelo básico com dez portas de entrada digital, seis portas de saída digital que consistem basicamente de ativações de relés por parte do CLP. Ao ativar o relé, o sinal que as saídas “digitais” transmitem pode consistir de sinais analógicos, considerando que a tensão que esta presente no canal comum do relé é determinado pelo usuário. Estão presentes também dois canais de entrada analógica e uma fonte de tensão de 24V que é fornecida pelo próprio CLP.

O modelo também trabalha com uma placa de expansão que adiciona ao CLP mais duas entradas analógicas de tensão e duas saídas analógicas que podem ser de tensão ou corrente.

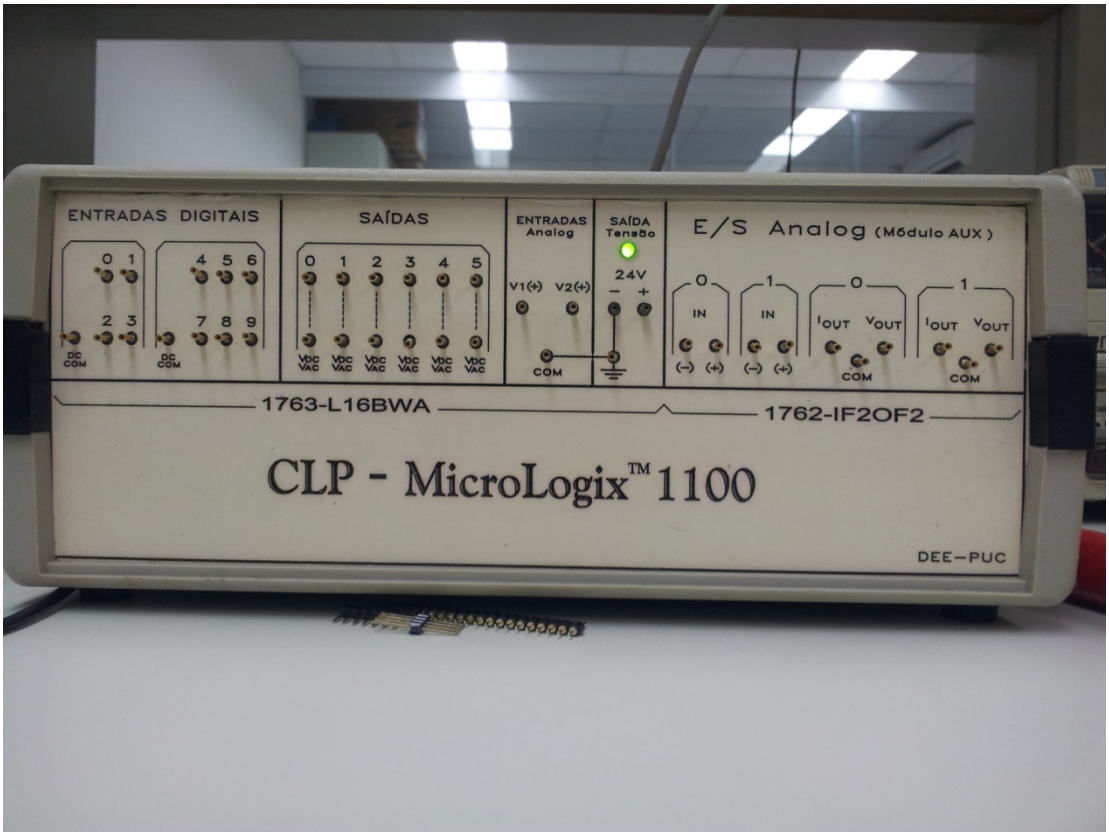

Figura 6 - Painel frontal do CLP

O CLP é alimentado com uma tensão variável de 110V e em relação a comunicação é adaptado a trabalhar com o protocolo de comunicação serial tipo RS-232C cuja aplicação nos processos industriais ainda é abundante então serve bem para exemplificar o CLP padrão.

2.1.1. Protocolo RS-232C

Apesar deste protocolo ser antigo, ainda é bastante utilizado na indústria, principalmente em periféricos e modems. Ele é um basicamente um protocolo para comunicação serial assíncrona que utiliza os bits de START e STOP como referências para a sincronização. Quando a comunicação esta inativa, ela mantêm nível lógico um conhecido como MARK. Assim que um conjunto de bits entra na linha de transmissão um bit de START prepara o receptor mudando o nível lógico da linha para baixo, conhecido como SPACE. Depois disso os bits a serem transmitidos podem ser enviados e o receptor estará esperando por eles. Ao final da transmissão pode ser enviado, dependendo da configuração do sistema, um ou mais bits de paridade que confirmam a integridade do sinal transmitido. Finalmente são enviados um ou dois bits MARK de STOP para sinalizar o fim do sinal enviado. A existência do STOP se deve ao fato de, caso sejam enviadas duas ou mais sequências de bytes, seria possível identificar onde começa e onde termina cada um, pois existiriam alguns bits de MARK entre eles.

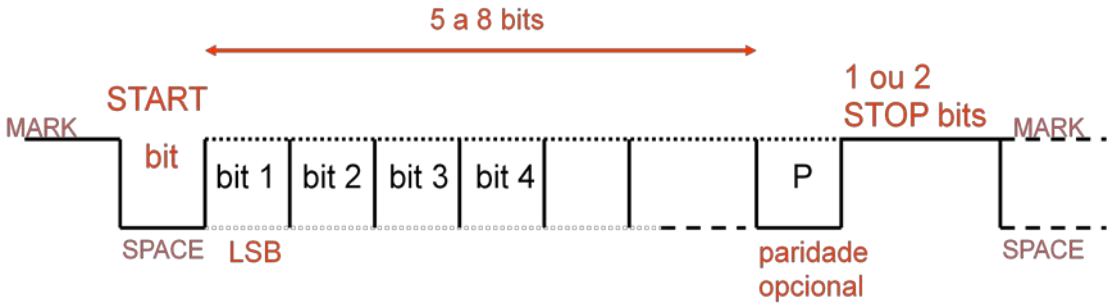

Figura 7 - demonstração do protocolo utilizado no RS-232

2.2. Módulo de radiofrequência

Inicialmente, é necessário estudar em quais condições o sistema normalmente operaria. As automações industriais tendem a operar com sistemas observados praticamente em tempo real. o que nos leva a concluir que a rede de comunicação tem que suportar um tráfego relativamente alto e com extrema confiabilidade.

Em geral, uma fábrica não possui extensões de terra muito grandes, portanto, é importante que utilizemos um sistema que consiga cobrir toda ou boa parte da planta industrial.

Entretanto, é importante ressaltar que quanto maior a frequência melhor a qualidade da comunicação, pois seria possível reafirmar a fidedignidade do sinal por meio de confirmações antes mesmo que o bit enviado mude de valor.

Outro ponto importante é o custo do módulo. Devido ao objetivo de utilização em larga escala, a substituição de cada cabo de comunicação requer uma avaliação de custo benefício na escolha do módulo, pois cada real a menos tem impacto no projeto. Depois de algum tempo de pesquisa chegamos a um bom candidato para a tarefa.

2.2.1. NRF24L01+

Este módulo se mostrou uma boa opção, pois cumpre todos os requisitos que havíamos especificado. Dentre suas características vale ressaltar:

- Alcance operacional radial de cem metros, para o modelo com antena impressa (mais barato), e de um quilômetro para o modelo com antena exterior (mais caro, mas pode ser necessário em alguns pontos da automação) [2]. Para fins práticos, adquirimos o modelo com antena impressa, pois a área de cobertura nos testes não requer tanta potência. No entanto, em aplicações industriais dependendo da disposição da automação seja melhor utilizar o outro modelo.

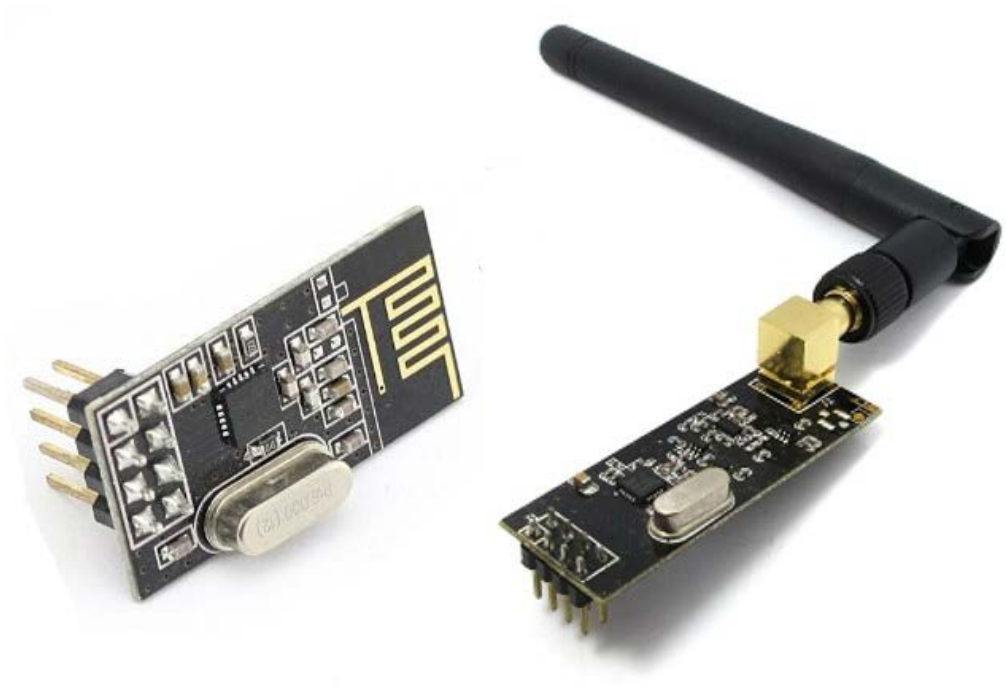

Figura 8 - Os dois modelos comercializados do módulo

- O módulo é um *transceiver* [2], ou seja, é capaz de enviar e receber dados no mesmo chip. Isto possibilita que a implementação tenha menos custo, pois seria necessário adquirir um número menor de módulos por unidade, no nosso caso CLP, a ser comunicada na rede.

- Possibilidade de operar em múltiplos canais [2]. Desta forma, seria possível que muitas comunicações sejam feitas sem causar interferência umas nas outras.

- O modo receptor possui a capacidade de ouvir até seis transmissores simultaneamente. Isto significa que podemos unificar a comunicação em uma configuração de teia de modo que qualquer modulo integrante da rede possa se fornecer ou adquirir dados de qualquer outro membro [3].

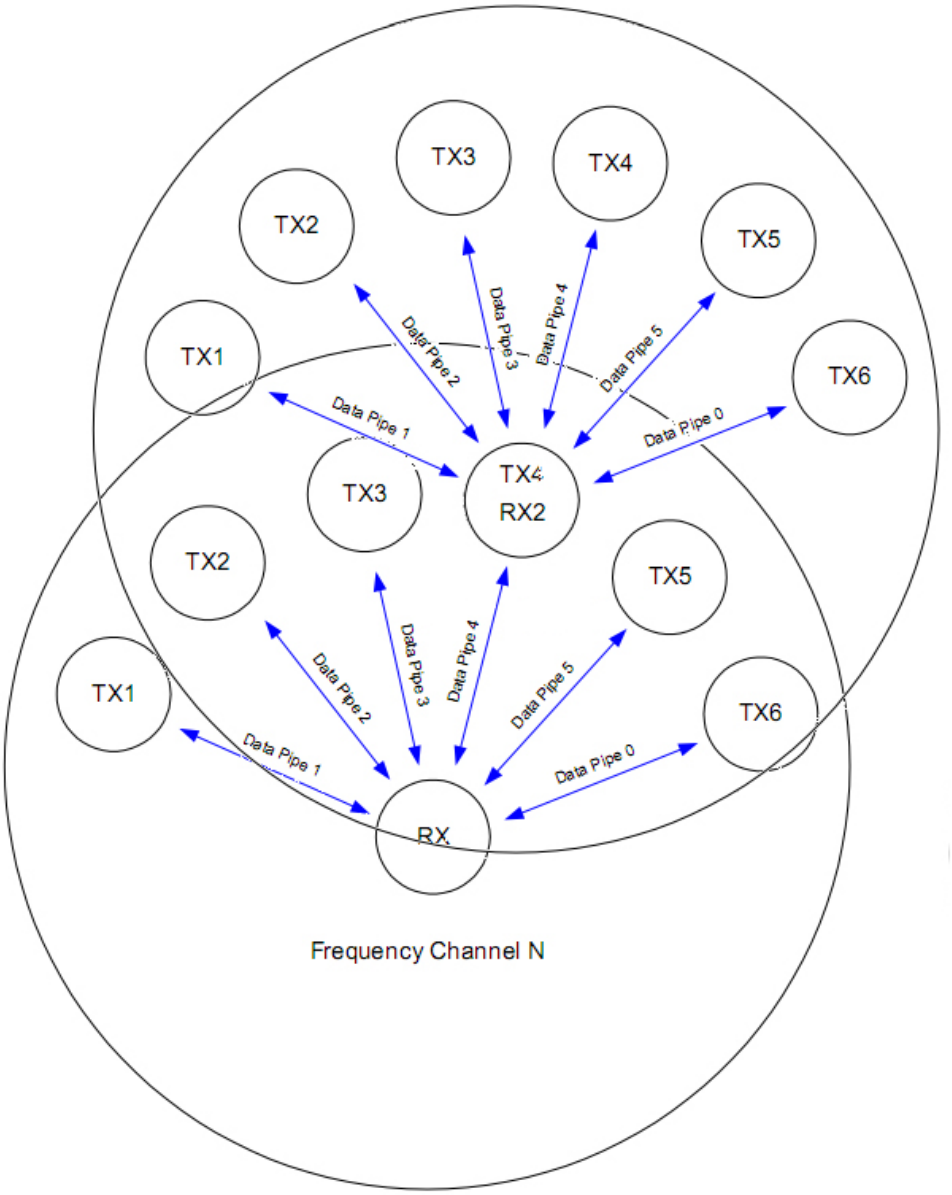

Figura 9 - Exemplo de teia de comunicação

- Pode ser configurado a retornar confirmação de recebimento bem como reenvio caso algum dado seja perdido [2]. Com possibilidade de até quinze reenvios, o sistema se torna extremamente confiável.

- Volume de dados e velocidade de transferência elevados: o módulo é capaz de transmitir até 32 bytes por vez a velocidades de 256Kbps, 1Mbps e 2Mbps [2], o que possibilita que o envio e a concretização do recebimento aconteçam em tempo de execução.

- E finalmente o mais importante, é barato: No Brasil pode ser encontrado a cerca de doze reais e no exterior a cerca de três dólares.

O módulo possui oito pinos, e se comunica com o microcontrolador pelo protocolo SPI

2.2.2. Protocolo SPI (*Serial Periphereral Interface*)

Protocolo que permite a comunicação serial de um dispositivo com diversos outros componentes, formando uma rede. Se baseia no conceito de MASTER (que envia) e SLAVE (que recebe) onde o MASTER pode enviar dados para diversos SLAVES. Ele, basicamente, possui quatro fios que correspondem ao *clock*, aos dados enviados e aos dados recebidos sendo que por eles trafegam quatro tipos de sinal:

• SCK: *Serial clock* (definido pelo MASTER)

• MISO: *Master Input, Slave Output* (oriundo do SLAVE)

• MOSI: *Master Output, Slave Input* (oriundo do MASTER)

• SS: *Slave Select* (ativo em nível lógico baixo, oriundo do MASTER)

Outra característica importante é que ele é *full-duplex*, ou seja, pode receber e enviar informações ao mesmo tempo através da conexão.

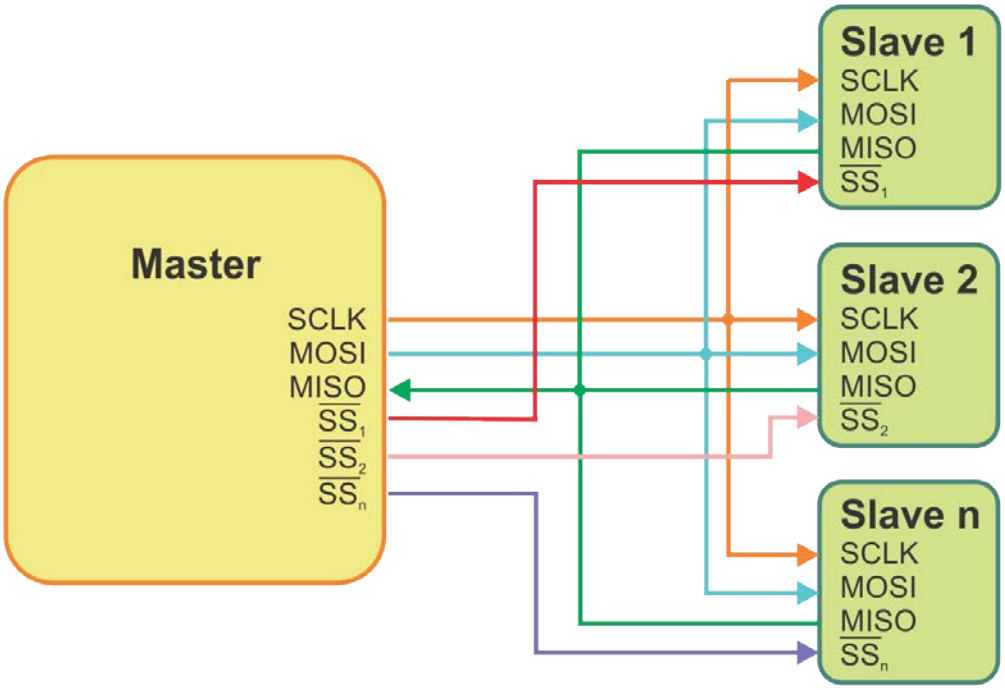

Figura 10 - Exemplo de implementação do protocolo

Para que toda esta comunicação seja estabelecida, necessitamos de um microcontrolador não apenas barato, mas que seja pequeno o suficiente para que em conjunto com a antena seja de fácil encaixe em espaços restritos.

2.2.3. Microcontrolador

Inicialmente, planejávamos utilizar o PIC18F452, mas devido a problemas que serão posteriormente detalhados, passamos a utilizar circuitos integrados. Como o objetivo é obter um sistema barato e pequeno, tivemos que abranger mais e cogitar outros fabricantes. Com isso, acabamos encontrando os circuitos integrados da ATmel, em particular o Arduino mini, que se mostrou barato (25 reais no Brasil e no exterior entre 4 e 8 dólares) em relação aos circuitos integrados vendidos pelos concorrentes. Ele possuí uma solução de tamanho ideal para a aplicação, já que seu circuito integrado era de tamanho similar ao chip de radio frequência, podendo assim facilitar na disposição das peças no possível produto final.

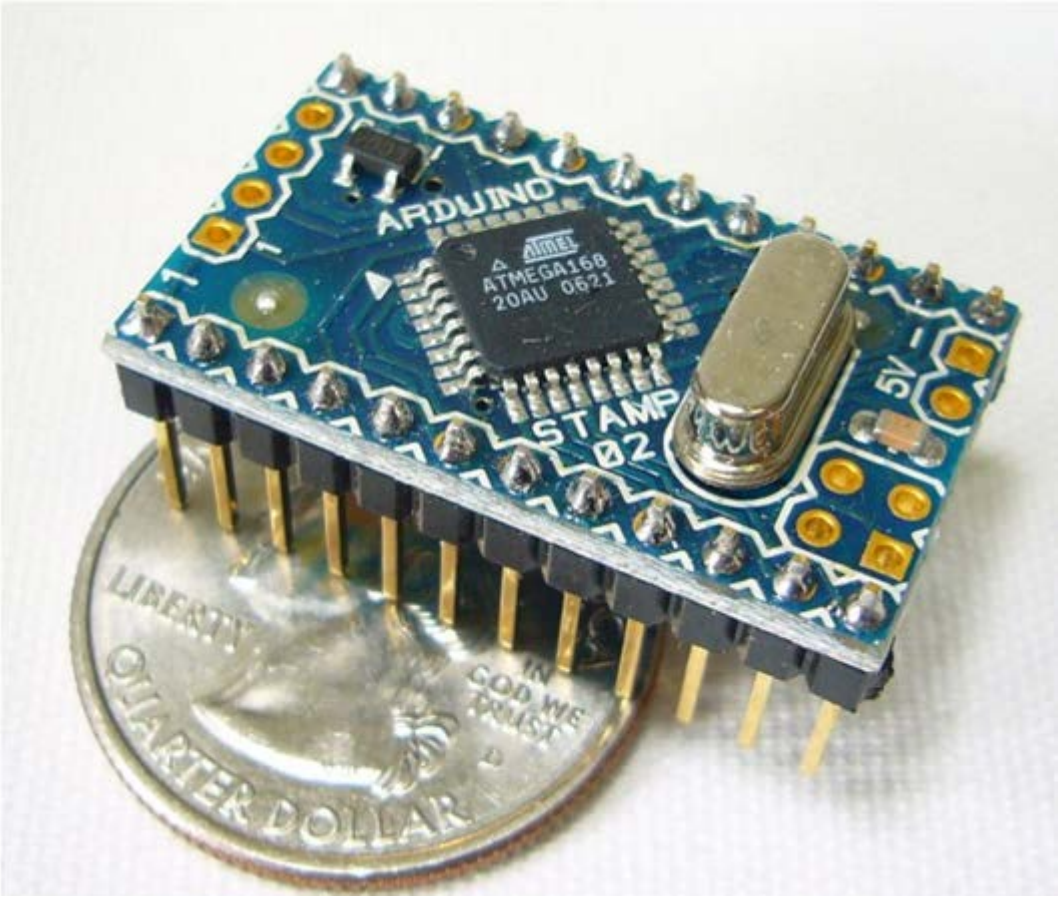

Figura 11 - Arduino mini

Vale ressaltar que, para fins práticos, utilizamos o Arduino UNO, pois eu já possuía dois circuitos desse modelo. Mas, como a programação, bem como o compilador, que os Arduinos utilizam é unificado podemos utilizar o mesmo programa nos dois circuitos. Por isso, na implementação, não houve a necessidade de comprar a solução indicada, mas em caso de aplicação real é desejável que utilize o circuito indicado.

3. Implementação Pratica da Comunicação Serial

Inicialmente utilizamos um microcontrolador tipo PIC, modelo PIC18F452, que possuía uma porta SPI e uma porta serial como requisitávamos. Após um mês de desenvolvimento conseguimos implementar a primeira comunicação do SPI com o chip de radio frequência, pois a biblioteca de SPI do compilador que utilizávamos (MikroC) acusava erro de compilação quando tentávamos utilizá-la. Desta forma, houve a necessidade de desenvolver uma biblioteca própria. Mas, devido a algum problema desconhecido no sistema, a conexão não era estável e após alguns segundos conectado o sinal de *clock* do SPI parava e a conexão era perdida. Suspeitamos da possibilidade do sistema estar instável por algum problema de mal contato e, com isso, passamos a utilizar uma placa integrada compatível com PIC disponibilizada pela faculdade.

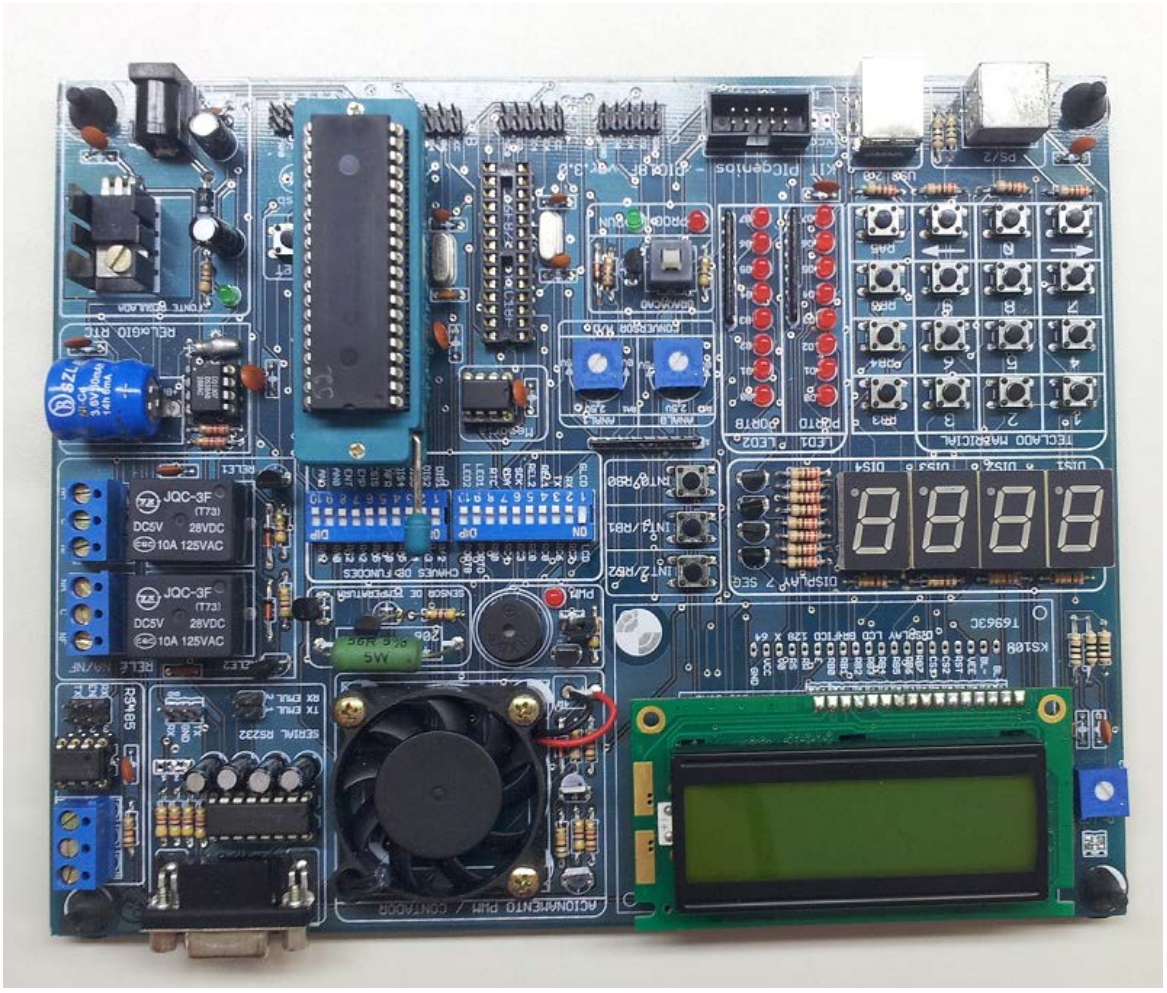

Figura 12 - Placa utilizada nos testes com PIC

Com esta placa conseguimos manter a conexão, então percebemos que para conseguir trabalhar de modo a eliminar problemas de estabilidade na conexão deveríamos aplicar ao máximo o uso de circuitos integrados e sistemas soldados.

Depois disso pesquisamos o circuito integrado que seria melhor para a aplicação real. Como descrito anteriormente, escolhemos o Arduino mini, mas utilizamos o Arduino UNO. Ambos possuem

comunicação Serial e SPI, também podemos ressaltar que utilizam o mesmo microcontrolador, o Atmega328, o que explica o fato de terem as mesmas conexões.

Antes de fazermos qualquer comunicação é necessário rebaixarmos a tensão lógica padrão do RS-232C (cerca de 12V) para a tensão de trabalho do chip de radio frequência (em teoria, 3.3V). No entanto, segundo o fabricante, ele suporta até 5.5V no pino de dados. Logo, utilizaremos 5V.

Para fazer este rebaixamento utilizamos um chip MAX232C que serve especificamente para isso. O sinal rebaixado vai para a porta de aquisição Serial do Arduino (RX) para depois ser retransmitido para a porta de saída do SPI (MOSI no MASTER). Isto é necessário, pois, caso ocorra algum problema com a transmissão, o dado fica armazenado na memória do microcontrolador e pode ser retransmitido caso necessário.

A mesma montagem é necessária no outro lado da transmissão para que o sinal transmitido seja recebido pelo chip de radio frequência e enviado para ser registrado na memória do Arduino pela porta de entrada do SPI (MISO no MASTER). O dado adquirido é então retransmitido para a porta Serial de saída (TX) que enviará o dado para outro MAX232C, e então, elevará a tensão para os 12V padrões do RS-232C. Enfim, o sinal chega ao receptor. Para o retorno do sinal utilizamos o mesmo processo só que desta vez estamos do lado que anteriormente era o receptor e que agora é o transmissor.

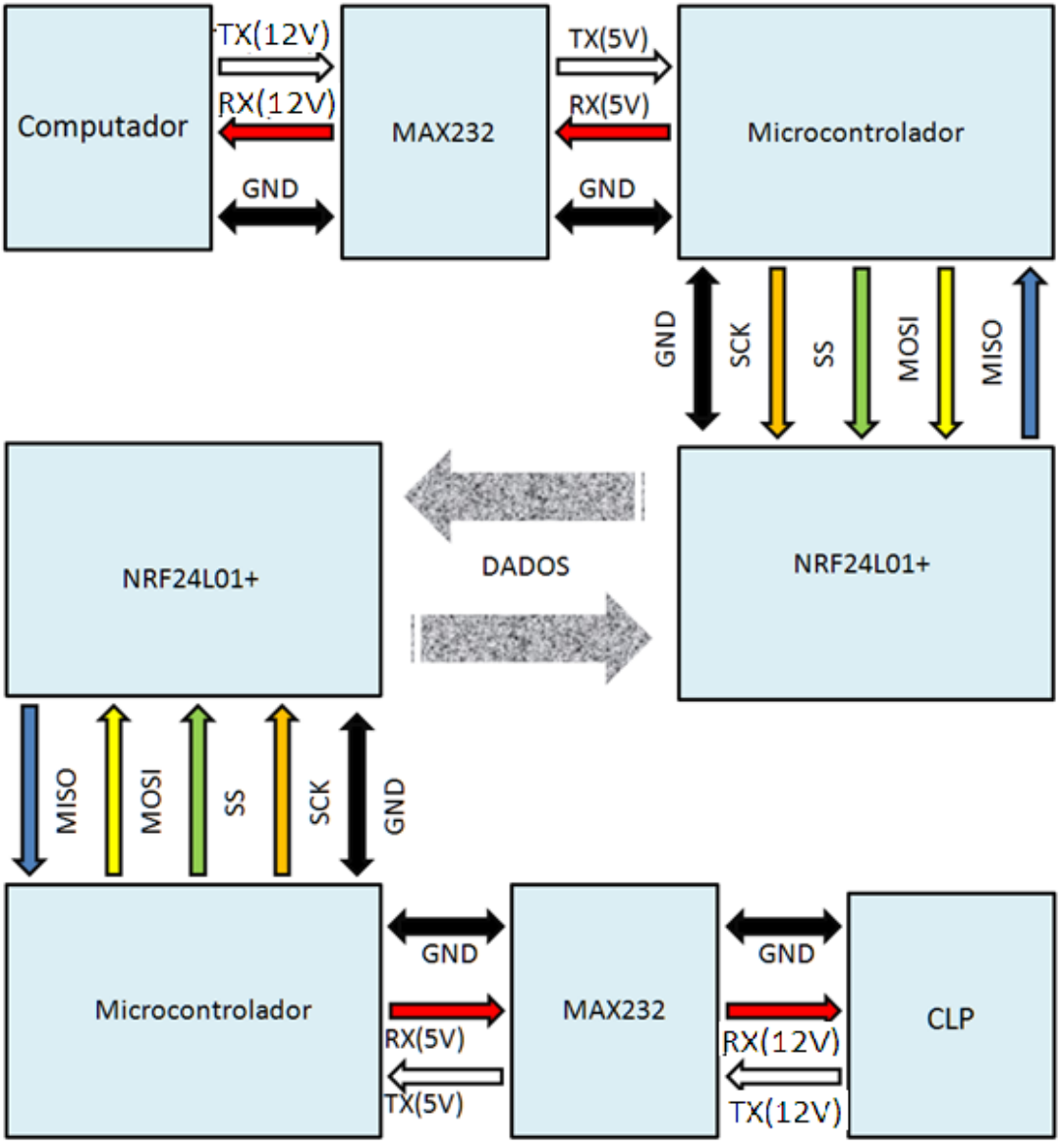

Figura 13 - Representação do circuito que foi implementado

Após alguns meses de desenvolvimento do programa que seria utilizado no microcontrolador conseguimos nossa primeira comunicação, onde o computador reconheceu o CLP como conectado ao sistema. Devido ao fato do modulo não ser *full-duplex* ele não consegue acompanhar a frequência de comunicação mais elevada do modo de gravação de programas. Entretanto, podemos utilizar esta configuração em aplicações que exigem menos demanda de dados, como em monitoramento de sensores e acionamentos, ambos muito requisitados em qualquer automação.

Para que a comunicação possa ser estabelecida, optamos por adicionar mais um módulo de comunicação por radio frequência em cada microcontrolador, de modo que passam a existir módulos dedicados a receber e a transmitir independentemente, configurando assim uma comunicação *full-duplex*. Na figura 14 podemos ver como ficou o novo circuito.

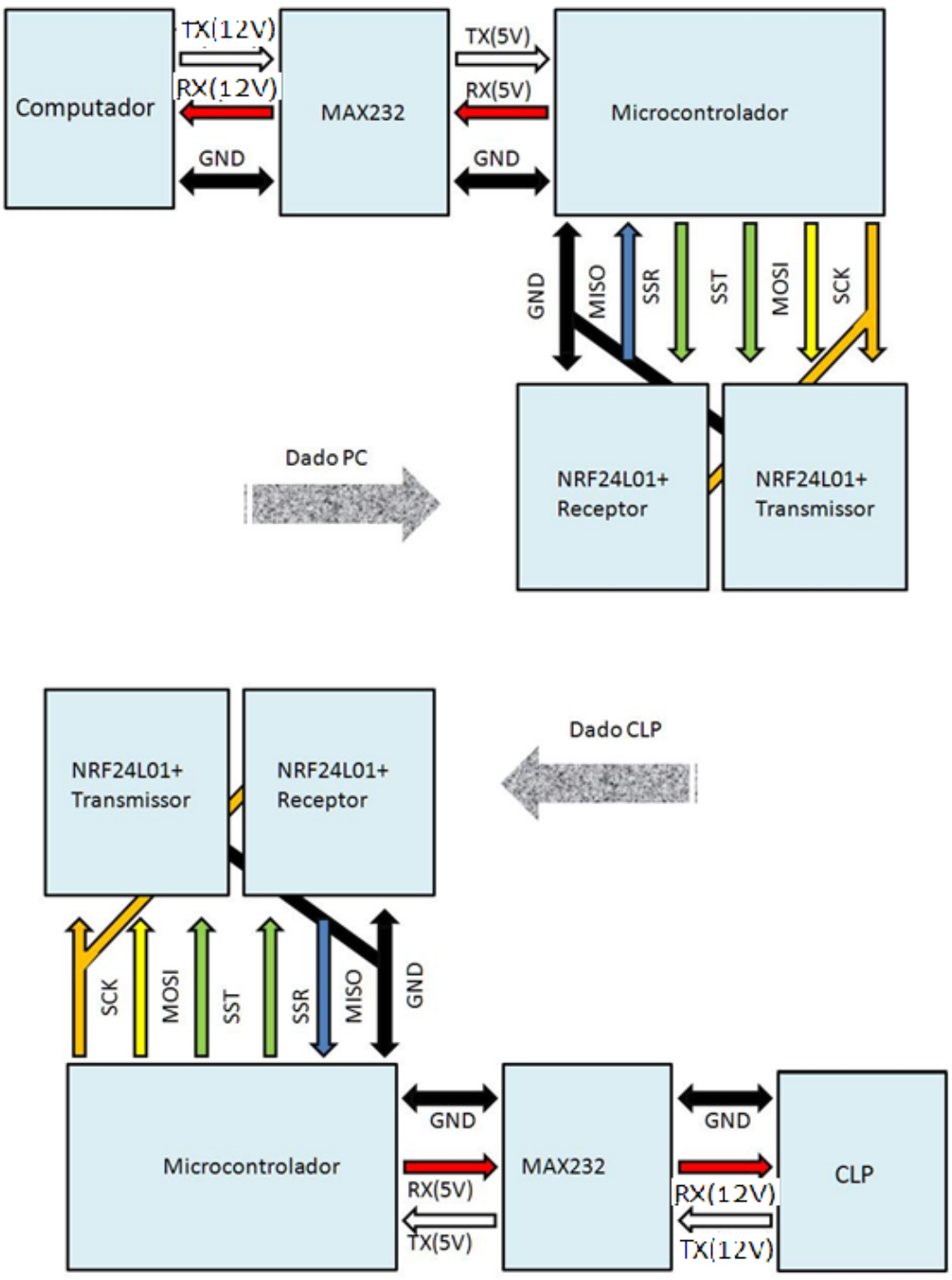

Figura 14 - Representação do novo circuito implementado

Com o sistema pronto, o próximo passo é adaptar o programa para operar neste novo modo. Como inicialmente precisamos configurar o modo de operação em cada NRF24L01+, utilizamos o circuito anterior, pois nele o módulo está apto a receber dados do MASTER (conectado a MISO e MOSI ao mesmo tempo). O MASTER configura os registradores internos dos módulos para trabalhar como desejamos, um receptor e outro transmissor.

Com os módulos configurados, podemos utilizar para este novo circuito.

Este novo sistema possibilitou que mantivéssemos a comunicação em modo *full-duplex*, mas, devido ao tempo exigido pelo módulo para efetuar a transmissão dos dados (aproximadamente cinco milissegundos), observamos um pequeno aumento no tempo que a transferência do arquivo deveria ser feita. Este problema ainda não tem efeitos significativos nas possíveis aplicações industriais ou residenciais, mas em redes mais amplas (com múltiplas comunicações seriais simultâneas) pode ocasionar falhas na comunicação pelo excesso de tempo necessário para obter resposta.

Após alguns testes, observamos também que a conexão estabelecida entre os módulos é de baixa qualidade e é perdida com certa frequência. Por exigir mais tempo dedicado tentando restabelecer a conexão, a eficácia do sistema como um todo se reduz, pois no tempo em que esta operação é efetivada qualquer dado que deveria ser recebido ou enviado é perdido.

Entretanto, vale ressaltar que foram efetuadas algumas comunicações bem sucedidas de modo que o sistema é apto a operar como necessitamos.

4. Aplicação da Comunicação Serial

Nos tópicos seguintes apresentaremos uma aplicação prática ideal para este tipo de sistema, um CLP remoto que controla um robô paralelo programado a direcionar um painel solar em direção ao sol.

Inicialmente precisamos definir o que são robôs paralelos e qual modelo utilizaremos.

4.1. Robôs paralelos

São sistemas mecânicos que usam diversos atuadores lineares em paralelo para controlar a posição do efetuador e suas derivadas.

As suas aplicações mais conhecidas são:

- Simuladores de voo e de automóveis;
- Alinhamento de estruturas ópticas;
- Montagem de placas circuito impresso;
- Micro manipuladores na extremidade de manipuladores lineares;
- Processos de fresagem de alta velocidade e precisão.

O grande diferencial dos robôs paralelos é que, em vez de conectar vários elos em série, existem vários elos paralelos conectados que conectam a base ao efetuador.

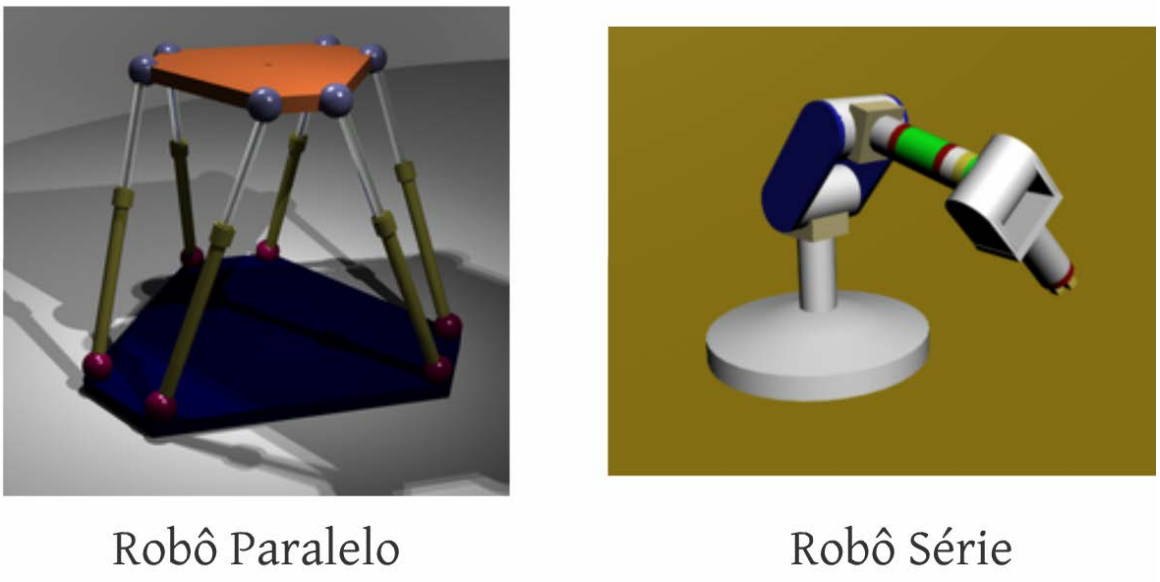

Figura 15 - Tipos de robôs

Os robôs paralelos são conhecidos por serem rígidos e robustos em relação a movimentos indesejados. Os erros de um atuador linear são compensados pelos outros (em oposição aos robôs em série, nos

quais os erros são acumulados), fazendo com que a estrutura se torne mais rígida quanto maior o número de atuadores lineares (juntas prismáticas).

Como a força é distribuída entre diversos elos paralelos e só há um "estágio de elos" antes do efetuador, o peso do robô e seu momento de inércia são reduzidos, possibilitando a realização de tarefas precisas em alta velocidade (não são necessárias reduções de grande ordem para alcançar a estabilidade e compensar o momento de inércia do braço).

O braço não necessita de juntas rotacionais, podendo ser sólido, o que torna a máquina menos precisa e mais sujeita a erros de histerese.

A grande desvantagem é que sua área de atuação é limitada. Seus atuadores podem colidir e as restrições nos seus fins de curso se acumulam, fazendo com que a distância máxima entre seus pontos extremos de atuação seja, normalmente, curta.

Para nossa aplicação utilizaremos um simulador vertical de movimentos. O simulador de movimentos é um sistema automático, composto por quatro atuadores pneumáticos independentes entre si com um sistema de controle [4]. Originalmente ele é utilizado para simular perfis de estrada [4], mas para nossa aplicação servirá para determinar em que direção um painel solar acoplado as extremidades dos atuadores deve inclinar.

A atuação de cada cilindro pneumático é determinada por duas válvulas, uma associada para cada sentido de atuação. Desse modo, podemos ter quatro estados de acionamento em cada atuador:

- 00: onde não há atuação e, portanto, ambas saídas e entradas de ar estão fechadas, o que faz com que o sistema trave na posição em que está.
- 01: onde há pressão no sentido de descida do atuador, o forçando a ficar retraído.
- 10: onde há pressão no sentido de subida do atuador, o forçando a ficar totalmente estendido.
- 11: onde há atuação em ambos os sentidos, forçando o atuador a ficar próximo da metade de sua atuação totalmente estendida.

Para determinar quando e qual atuador deve atuar precisamos determinar algum método de sensoriamento para que saibamos em qual estado o sistema se encontra. Para isso, utilizamos sensores de luminosidade que irão detectar a luminosidade no ambiente e sinalizará ao CLP uma atuação direcionando a plataforma superior em direção à luminosidade de modo que os sensores tenham medida diferencial mínima, ou seja, o desvio padrão de todos deverá ser o menor possível. Esta plataforma possui quatro sensores de luminosidade (um em cada extremidade).

4.2. Programas e linguagens utilizados

É importante ressaltar que foram utilizados dois programas para que este projeto fosse implementado: O LabVIEW, onde os dados dos sensores foram processados e o RSLogix 500 fornecido pela Rockwell Automation, fabricante do CLP, cuja função é programar o CLP utilizando a linguagem Ladder.

O LabVIEW (acrónimo para *Laboratory Virtual Instrument Engineering Workbench*) é uma linguagem de programação gráfica originária da National Instruments.

Os principais campos de aplicação do LabVIEW são a realização de medições e a automação. A programação é feita de acordo com o modelo de fluxo de dados, o que oferece a esta linguagem vantagens para a aquisição de dados e para a sua manipulação.

A linguagem Ladder é um auxílio gráfico para programação de Controladores Lógicos Programáveis (CLPs) no qual as funções lógicas são representadas através de contatos e bobinas, de modo análogo a um esquema elétrico com os contatos dos transdutores e atuadores.

4.3. Implementação do projeto

Através de um aquisitor de dados (que no caso foi por um módulo externo, mas poderia ser feito pelo CLP), é feita a aquisição da tensão em cada um dos sensores. O sensor tem comportamento transitivo, sendo a luminosidade correspondente à tensão de base, isto é, quanto maior a luminosidade, maior a tensão no emissor.

Através do LabVIEW, processamos o dado aquisitado de cada sensor de modo a determinar quando devemos ativar cada atuador e em que posição ele deve parar. Nele também convertemos a atuação para um número inteiro composto por bits, cuja função é auxiliar a programação do CLP.

O CLP recebe o dado já processado pelo LabVIEW, como mencionado anteriormente, e atua diretamente na válvula, sendo que são 8 válvulas a serem atuadas e o CLP Micrologix 1100A possui apenas 6 saídas digitais. Por isso, utilizamos as duas saídas analógicas, oferecidas pelo módulo auxiliar acoplado ao CLP, como saídas digitais. Para isso, incluímos no circuito dois relés, onde acionamos VCC com nível lógico alto na saída analógica (qualquer nível maior que zero) e GND com nível lógico baixo na saída analógica (zero propriamente).

O sistema de sensoriamento é alimentado com 5V de modo que a aquisição dos dados varia de 0V a 5V. Um valor de 0V significa que não há luminosidade no ambiente. Analogamente, um valor de 5V indica luminosidade total. É importante ressaltar que o sensor recebe estímulos luminosos de todas as direções. Logo, utilizamos um invólucro cilíndrico de cor preta com o intuito de captar apenas a luminosidade oriunda de uma determinada direção.

4.3.1. Circuito

Circuito de Aquisição de Dados

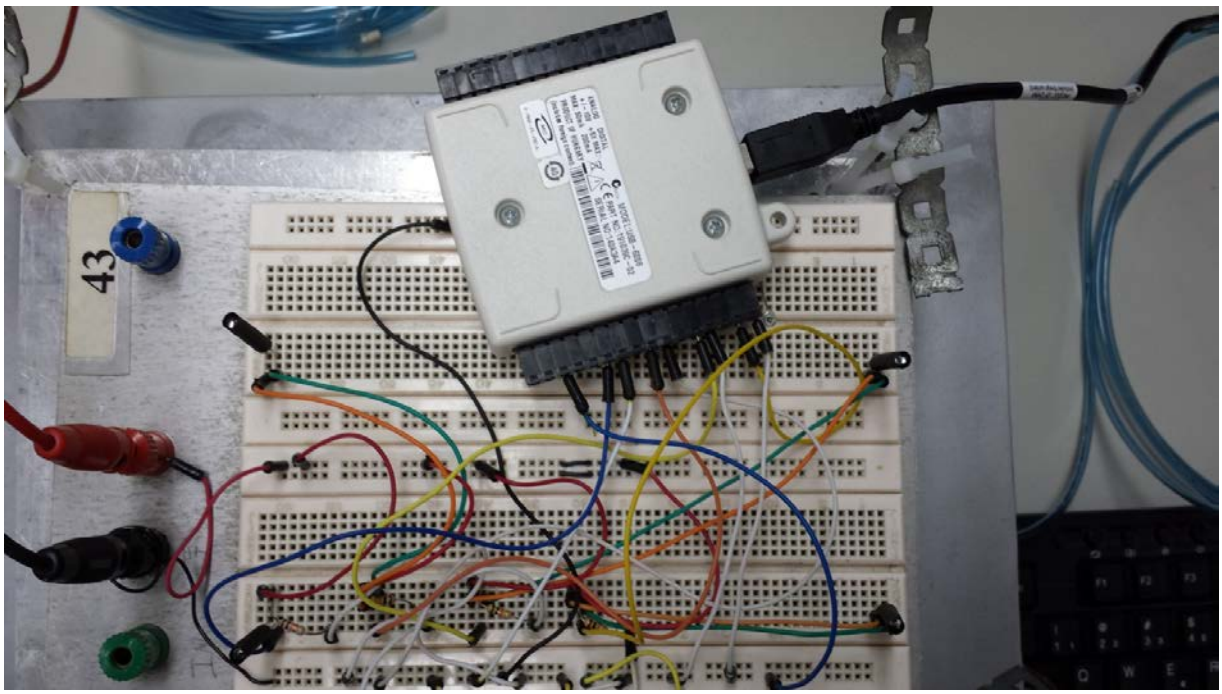

Figura 16 - Circuito que realiza a aquisição de dados

A Figura 16 acima mostra o circuito que realiza a aquisição de dados. Os quatro sensores de luminosidade encontram-se nas extremidades da *protoboard* e estão cobertos parcialmente por um isolante preto, para obter uma luminosidade direcionada. O posicionamento destes sensores está de acordo com a localização dos atuadores, isto é, o sensor superior esquerdo está relacionado com o atuador 4 (atuador superior esquerdo), o sensor superior direto com o atuador 2 (atuador superior direito), o sensor inferior esquerdo com o atuador 3 (atuador inferior esquerdo) e o inferior direito com o atuador 1 (atuador inferior direito).

Circuito de Acionamento das Válvulas

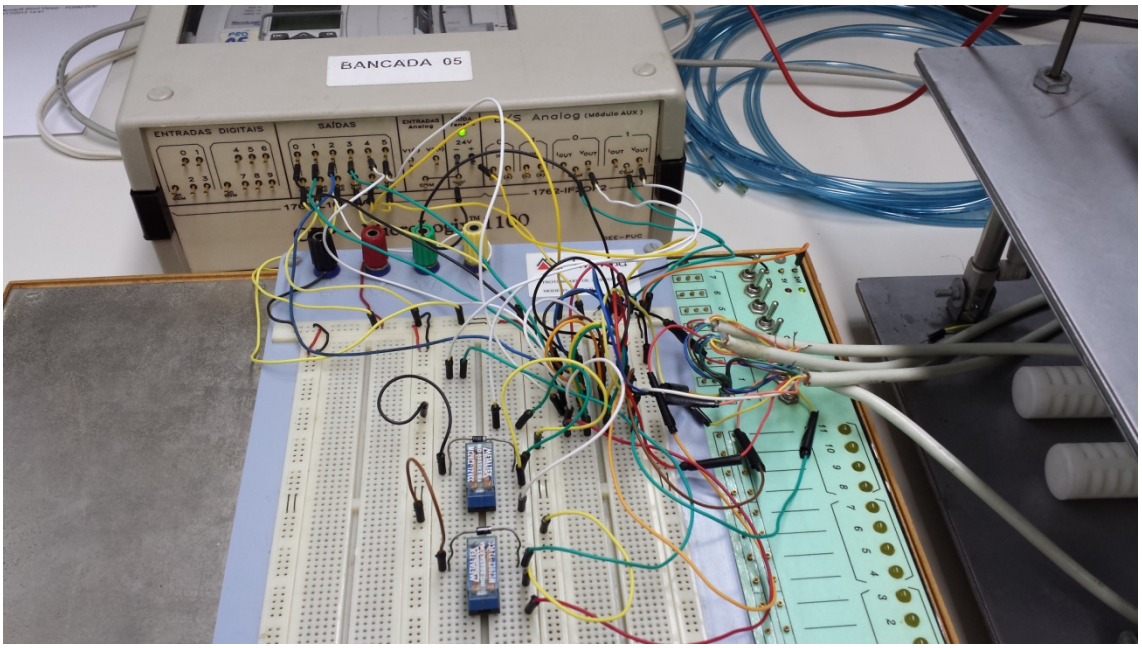

Figura 17 - Circuito de acionamento das válvulas

Este circuito tem como função ativar ou desativar as válvulas que controlam a entrada e saída de ar nos atuadores. Para isso, utilizamos os seis relés que estão presentes no CLP e também as duas saídas analógicas do modulo acoplado ao CLP. Estas últimas, convertidas a trabalhar como saídas digitais com a utilização de dois relés externos adicionados ao sistema. Podemos ver em detalhes essa conversão na Figura abaixo

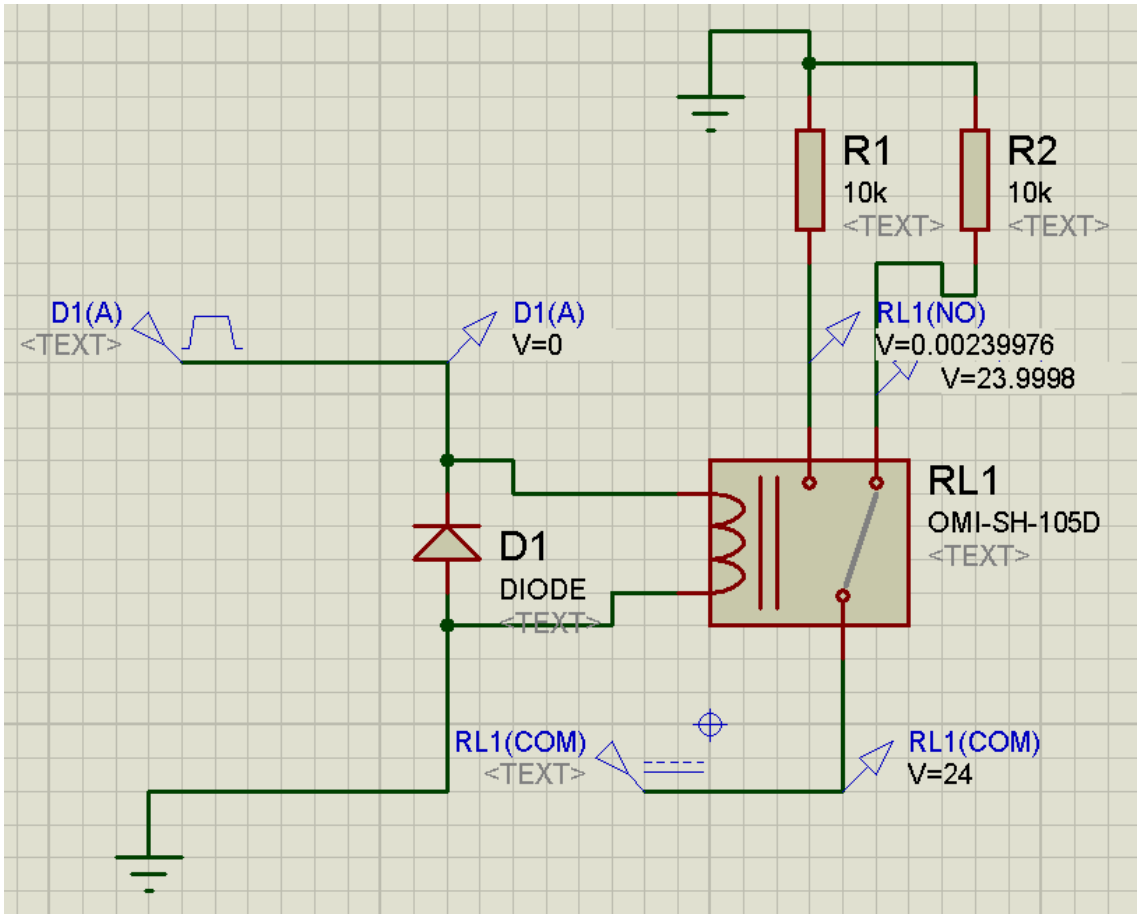

Figura 18 - Esquemático da conversão analógico digital

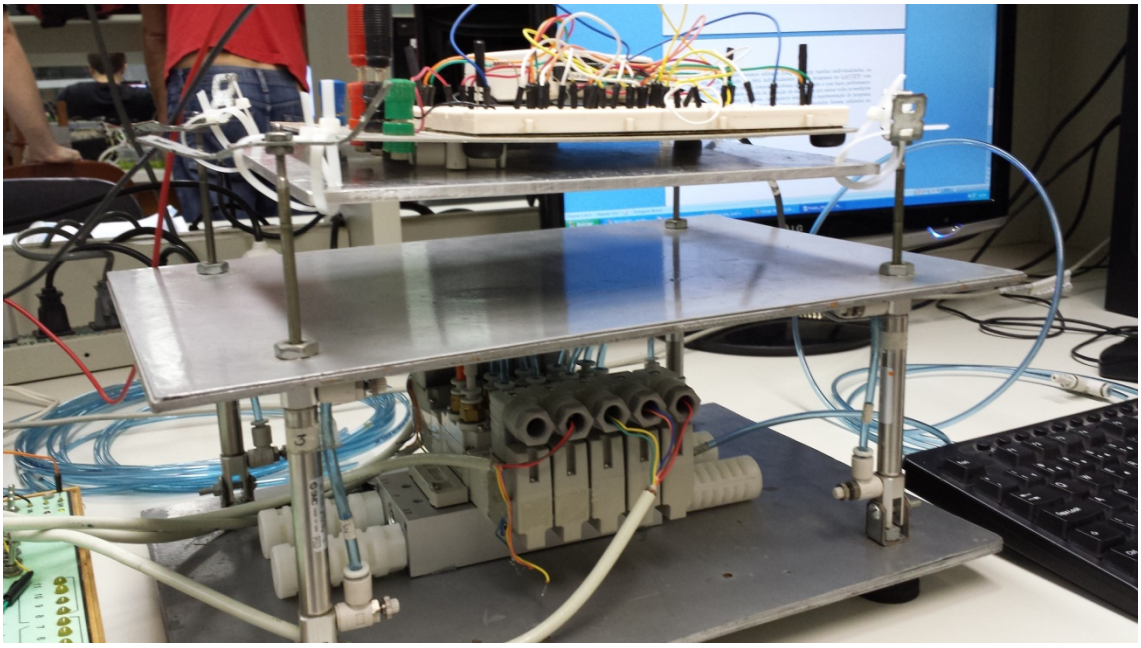

Figura 19 - Imagem do atuador paralelo. Na parte inferior é possível ver as válvulas de atuação

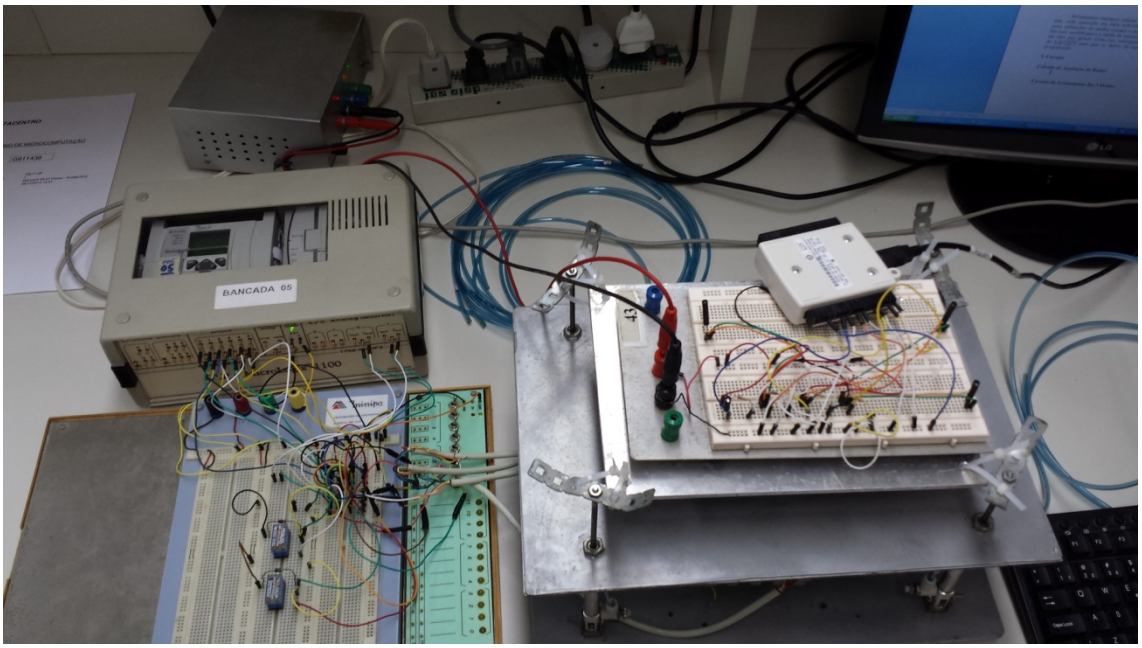

Figura 20 - Imagem do projeto por completo

4.3.2. Rendimento dos painéis solares

Geralmente os painéis solares são instalados em uma posição fixa onde ocorre mais incidência solar. Esta configuração impede que o sistema tenha rendimentos altos, pois, durante a manhã e o fim da tarde, os painéis não estão posicionados de modo a obter maior radiação incidente. Seu rendimento pico geralmente é à tarde, quando geralmente o painel está corretamente direcionado para o sol e a incidência do mesmo é mais elevada.

Em um sistema móvel podemos garantir que o painel está sempre direcionado para o sol de modo a permanecer com rendimentos similares ao de pico do sistema fixo. Mesmo considerando os gastos energéticos do sistema responsável por esse direcionamento ainda teremos uma grande margem para superar a configuração fixa[5], de modo que ainda podemos adicionar outros módulos de automação que sejam dependentes da energia gerada pelos painéis.

A Figura abaixo demonstra o rendimento encontrado em uma aplicação teste onde uma solução parecida obteve resultados similares ao que esperávamos.

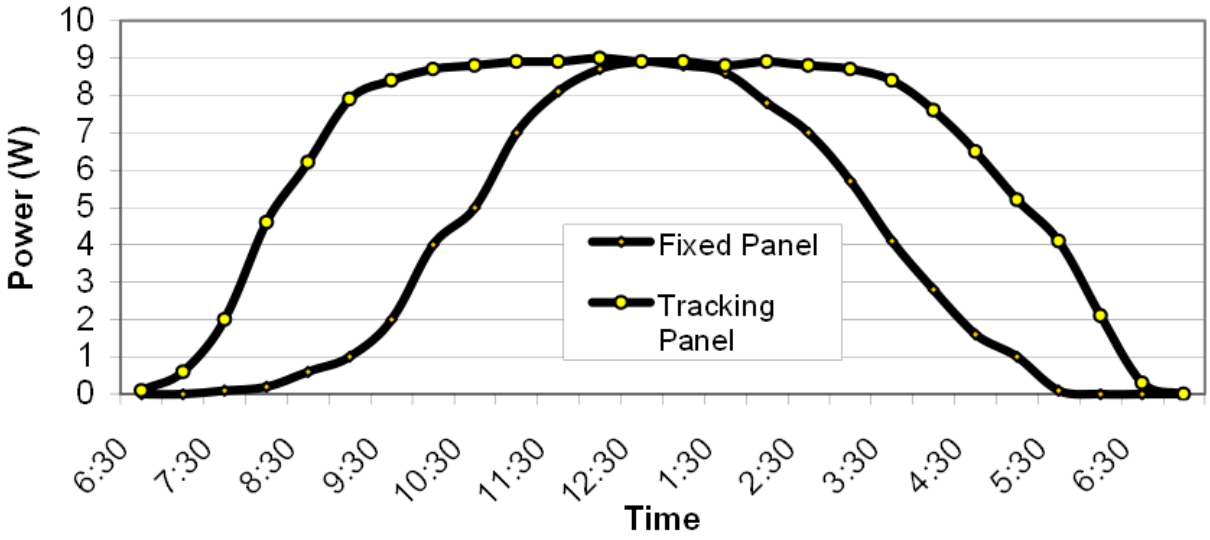

Figura 21 - Aplicação pratica de uma solução similar

5. Conclusão

Observando os resultados obtidos podemos concluir que a aplicabilidade da solução de baixo custo obtida para a comunicação Serial sem fio ainda não está pronta para aplicações industriais, onde há maior exigência em relação à velocidade e a confiabilidade na transferência de dados. No entanto, para aplicações residenciais, onde geralmente necessita-se uma taxa de transferência menor, está solução possui potencial para ser aplicada.

Em relação a aplicação, podemos concluir que sua utilização possibilita que novos conceitos de automação sejam explorados, considerando que, sem a necessidade de alimentação externa, uma automação pode se tornar independente de infra estrutura, possibilitando que locais remotos possam ser explorados como possível espaço industrial.

Se aliarmos a essa independência energética um sistema de comunicação sem fio confiável e de longo alcance, podemos monitorar e modificar automações remotamente, configurando uma celular remota de produção.

Figura 22 - Exemplo bem sucedido de aplicações futuras

6. Referências

[1] RAPPAPORT ,T. S., “Wireless Communications: Principles and Practice (2nd Edition)”, Prentice Hall, USA, 2002.

[2] NORDIC SEMICONDUCTORS, “nRF24L01+, Single Chip 2.4GHz Transceiver, Product Specification v1.0”, Nordic Semiconductors, Norway, 2008.

[3] NORDIC SEMICONDUCTORS, “Introduction to wireless networks”, Nordic Semiconductors, Norway, 2004.

[4] ALLAN,N. A.,” Desenvolvimento de simuladores de movimento em escala com atuação pneumática”, PUC-Rio, Brasil, 2009

[5] RIZK J. e CHAIKO Y.,”Solar Tracking System: More Efficient Use of Solar Panels”, World Academy of Science, Engineering and Technology, 17, Australia, 2008.

7. Apêndices

7.1. Código do microcontrolador utilizado para comunicação simples

7.1.1. Microcontrolador que recebe dados do computador

```
#include <SPI.h>
#include <SoftwareSerial.h>
#include "nRF24L01.h"
#include "RF24.h"

RF24 radio(8,7);
SoftwareSerial mySerial(5, 6); // RX, TX
// Radio pipe addresses for the 2 nodes to communicate.
const uint64_t pipes[2] = {
  0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL };

//for Serial input
String inputString = "";         // a string to hold incoming data
String outputString = "";         // a string to hold incoming data
boolean stringComplete = false;  // whether the string is complete

//NRF Packages
byte SendPackage[32];
byte ReceivePackage[32];
boolean sending=0;
```

```
void setup(void)
{
  //
  // Print preamble
  //

  mySerial.begin(19200);
  //Serial.begin(115200);
  //mySerial.begin(9600);
  //Serial.begin(9600);
  radio.begin();
  // optionally, increase the delay between retries & # of retries
  radio.setDataRate(RF24_2MBPS);
  radio.setPALevel(RF24_PA_MAX);
  radio.setChannel(85);
  // delay 1ms, 3 retries
  // 0 - 250us, 15 - 4000us
  radio.setRetries(0,0);
  radio.setPayloadSize(32);
  radio.openWritingPipe(pipes[1]);
  radio.openReadingPipe(1,pipes[0]);
  radio.startListening();
  radio.printDetails();
}

void loop(void)
```

```
{
 NRFreceive();
 Serialreceive();
 inputString = "T:";
 inputString += "S:";
 while (mySerial.available())
 {
  byte inByte = (byte)mySerial.read();
  //Serial.println(inByte);
  if(inByte < 100)
  {
   inputString += "0";
   if(inByte < 10)
   {
    inputString += "0";
   }
  }
  inputString += inByte;
  stringComplete = true;
  unsigned long started_waiting_at2 = millis();
  bool timeout2 = false;
  while(!mySerial.available()  && ! timeout2)
  {
   if (millis() - started_waiting_at2 > 1 )
   {
    timeout2 = true;
   }
```

```
    }
   }
   inputString += "Gu:";
  }

byte NRFsend(String NRFPack = ""){
   if (!NRFPack.startsWith("S:")) {
    NRFPack = "S:" + NRFPack;
   }
   //Serial.println(NRFPack);
   //Serial.println("send");
   NRFPack.getBytes(SendPackage, 32);
   radio.stopListening();
   radio.openWritingPipe(pipes[0]);
   radio.openReadingPipe(1,pipes[1]);
   bool ok = false;
   while(!ok)
   {
    ok = radio.write(SendPackage,sizeof(SendPackage));
   }
   radio.startListening();
   unsigned long started_waiting_at = millis();
   bool timeout = false;
   while (! timeout )// ! radio.available() &&
   {
    if(radio.available())
    {
```

```
    bool done = false;
    while (!done)
    {
      done = radio.read( &ReceivePackage, sizeof(ReceivePackage) );
    }
    if(ReceivePackage==SendPackage)
    {
      timeout = true;
    }
    else
    {
      NRFsend(NRFPack);
    }
  }
  if (millis() - started_waiting_at > 250 )
  {
    timeout = true;
  }
 }
 if ( timeout )
 {
  //Serial.println("NRFerror");
  NRFsend(NRFPack);
 }
 radio.openWritingPipe(pipes[0]);
 radio.openReadingPipe(1,pipes[1]);
}
```

```
void NRFreceive(){
  if ( radio.available() )
  {
    bool done = false;
    while (!done)
    {
      done = radio.read( &ReceivePackage, sizeof(ReceivePackage) );
    }
    radio.stopListening();
    inputString = ((char *)ReceivePackage);
    stringComplete = true;
    bool ok = false;
    while(!ok)
  {
    ok = radio.write(&ReceivePackage, sizeof(ReceivePackage));
  }
    radio.startListening();
  }
}


void Serialreceive(){
  if (stringComplete) {
    if (inputString.startsWith("T:")) {
      int i = 2;
      //Serial.println(inputString);
      for(i=2;i<inputString.length()-2;i+=28)
```

```
  {
    NRFsend(inputString.substring(i,i+28));
  }
}
else if (inputString.startsWith("S:")) {
  outputString += inputString.substring(2);
  inputString = "";
  if(outputString.endsWith("Gu:"))
  {
    //Serial.println(outputString);
    byte outputByte[(outputString.length()-3)/3];
    int y = 0,x = 0;
    //String = outputString.substring(0,outputString.length()-3);
    //mySerial.print(outputString.substring(0,outputString.length()-3));
    //Serial.print(outputString.substring(0,outputString.length()-3));
    for(y = 0;y<outputString.length()-3;y+=3)
    {
      outputByte[x] = outputString.substring(y,y+3).toInt();
      //mySerial.write(outputString.substring(y,y+3).toInt());
      //Serial.println(outputString.substring(y,y+3).toInt());
      x += 1;
    }
    mySerial.write(outputByte,sizeof(outputByte));
    //Serial.write(outputByte,sizeof(outputByte));
    //Serial.println();
    outputString = "";
  }
```

```
    }
    //inputString = "";
    stringComplete = false;
  }
}
```

7.1.2. Microcontrolador que recebe dados do CLP

```
#include <SPI.h>
#include <SoftwareSerial.h>
#include "nRF24L01.h"
#include "RF24.h"

RF24 radio(8,7);
SoftwareSerial mySerial(5, 6); // RX, TX
// Radio pipe addresses for the 2 nodes to communicate.
const uint64_t pipes[2] = {
  0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL };

//for Serial input
String inputString = "";         // a string to hold incoming data
String outputString = "";         // a string to hold incoming data
boolean stringComplete = false;  // whether the string is complete

//NRF Packages
byte SendPackage[32];
byte ReceivePackage[32];
```

```
boolean sending=0;


void setup(void)
{
  //
  // Print preamble
  //


  mySerial.begin(19200);
  //Serial.begin(115200);
  //mySerial.begin(9600);
  //Serial.begin(9600);
  radio.begin();
  // optionally, increase the delay between retries & # of retries
  radio.setDataRate(RF24_2MBPS);
  radio.setPALevel(RF24_PA_MAX);
  radio.setChannel(85);
  // delay 1ms, 3 retries
  // 0 - 250us, 15 - 4000us
  radio.setRetries(0,0);
  radio.setPayloadSize(32);
  radio.openWritingPipe(pipes[0]);
  radio.openReadingPipe(1,pipes[1]);
  radio.startListening();
  radio.printDetails();
}
```

```
void loop(void)
{
 NRFreceive();
 Serialreceive();
 inputString = "T:";
 inputString += "S:";
 while (mySerial.available())
 {
  byte inByte = (byte)mySerial.read();
  //Serial.println(inByte);
  if(inByte < 100)
  {
   inputString += "0";
   if(inByte < 10)
   {
    inputString += "0";
   }
  }
  inputString += inByte;
  stringComplete = true;
  unsigned long started_waiting_at2 = millis();
  bool timeout2 = false;
  while(!mySerial.available()  && ! timeout2)
  {
   if (millis() - started_waiting_at2 > 1 )
   {
    timeout2 = true;
```

```
    }
   }
  }
  inputString += "Gu:";
 }

byte NRFsend(String NRFPack = ""){
  if (!NRFPack.startsWith("S:")) {
    NRFPack = "S:" + NRFPack;
  }
  //Serial.println(NRFPack);
  //Serial.println("send");
  NRFPack.getBytes(SendPackage, 32);
  radio.stopListening();
  radio.openWritingPipe(pipes[0]);
  radio.openReadingPipe(1,pipes[1]);
  bool ok = false;
  while(!ok)
  {
    ok = radio.write(SendPackage,sizeof(SendPackage));
  }
  radio.startListening();
  unsigned long started_waiting_at = millis();
  bool timeout = false;
  while ( ! radio.available() && ! timeout )
  {
    if (millis() - started_waiting_at > 250 )
```

```
    {
      timeout = true;
    }
  }
  if ( timeout )
  {
    //Serial.println("NRFerror");
    NRFsend(NRFPack);
  }
  radio.openWritingPipe(pipes[0]);
  radio.openReadingPipe(1,pipes[1]);
}


void NRFreceive(){
  if ( radio.available() )
  {
    bool done = false;
    while (!done)
    {
      done = radio.read( &ReceivePackage, sizeof(ReceivePackage) );
    }
    radio.stopListening();
    inputString = ((char *)ReceivePackage);
    stringComplete = true;
    radio.write( "1", 1 );
    radio.startListening();
  }
```

```
}


void Serialreceive(){
  if (stringComplete) {
    if (inputString.startsWith("T:")) {
      int i = 2;
      //Serial.println(inputString);
      for(i=2;i<inputString.length()-2;i+=28)
      {
        NRFsend(inputString.substring(i,i+28));
      }
    }
    else if (inputString.startsWith("S:")) {
      outputString += inputString.substring(2);
      inputString = "";
      if(outputString.endsWith("Gu:"))
      {
        //Serial.println(outputString);
        byte outputByte[(outputString.length()-3)/3];
        int y = 0,x = 0;
        //String = outputString.substring(0,outputString.length()-3);
        //mySerial.print(outputString.substring(0,outputString.length()-3));
        //Serial.print(outputString.substring(0,outputString.length()-3));
        for(y = 0;y<outputString.length()-3;y+=3)
        {
          outputByte[x] = outputString.substring(y,y+3).toInt();
          //mySerial.write(outputString.substring(y,y+3).toInt());
```

```
        //Serial.println(outputString.substring(y,y+3).toInt());
        x += 1;
      }
      mySerial.write(outputByte,sizeof(outputByte));
      //Serial.write(outputByte,sizeof(outputByte));
      //Serial.println();
      outputString = "";
    }
  }
  //inputString = "";
  stringComplete = false;
 }
}
```

7.2. Código do microcontrolador utilizado na comunicação *full-duplex*

7.2.1. Microcontrolador que recebe dados do computador

```
#include <SPI.h>
#include <SoftwareSerial.h>
#include "nRF24L01.h"
#include "RF24.h"

RF24 radio(8,7);
RF24 radio2(4,3);
SoftwareSerial mySerial(5, 6); // RX, TX
```

```
// Radio pipe addresses for the 2 nodes to communicate.
const uint64_t pipes[2] = {
  0xF0F0F0F0E1LL, 0xE0E0E0E0D2LL };


//for Serial input
String inputString = "";         // a string to hold incoming data
String outputString = "";         // a string to hold incoming data
boolean stringComplete = false;  // whether the string is complete


//NRF Packages
byte SendPackage[32];
byte ReceivePackage[32];
boolean sending=0;


void setup(void)
{
  //
  // Print preamble
  //


  mySerial.begin(19200);
  Serial.begin(115200);
  //mySerial.begin(9600);
  //Serial.begin(9600);
  radio.begin();
  radio2.begin();
  // optionally, increase the delay between retries & # of retries
```

```
  radio.setDataRate(RF24_2MBPS);
  radio.setPALevel(RF24_PA_MAX);
  radio.setChannel(85);
  radio2.setDataRate(RF24_2MBPS);
  radio2.setPALevel(RF24_PA_MAX);
  radio2.setChannel(115);
  // delay 1ms, 3 retries
  // 0 - 250us, 15 - 4000us
  radio.setRetries(0,0);
  radio.setPayloadSize(32);
  radio2.setRetries(0,0);
  radio2.setPayloadSize(32);
  radio.openWritingPipe(pipes[1]);
  radio2.openReadingPipe(1,pipes[0]);
  radio.stopListening();
  radio.printDetails();
  radio2.startListening();
  radio2.printDetails();
}

void loop(void)
{
  NRFreceive();
  Serialreceive();
  inputString = "T:";
  inputString += "S:";
  while (mySerial.available())
```

```
  {
    byte inByte = (byte)mySerial.read();
    //Serial.println(inByte);
    if(inByte < 100)
    {
      inputString += "0";
      if(inByte < 10)
      {
        inputString += "0";
      }
    }
    inputString += inByte;
    stringComplete = true;
    unsigned long started_waiting_at2 = millis();
    bool timeout2 = false;
    while(!mySerial.available()  && ! timeout2)
    {
      if (millis() - started_waiting_at2 > 1 )
      {
        timeout2 = true;
      }
    }
  }
  inputString += "Gu:";
}

byte NRFsend(String NRFPack = ""){
```

```
if (!NRFPack.startsWith("S:")) {
  NRFPack = "S:" + NRFPack;
}
//Serial.println(NRFPack);
//Serial.println("send");
NRFPack.getBytes(SendPackage, 32);
//radio.stopListening();
//radio.openWritingPipe(pipes[1]);
//radio.openReadingPipe(1,pipes[0]);
bool ok = false;
while(!ok)
{
  ok = radio.write(SendPackage,sizeof(SendPackage));
}
//radio.startListening();
unsigned long started_waiting_at = millis();
bool timeout = false;
while ( ! radio2.available() && ! timeout )
{
  if (millis() - started_waiting_at > 250 )
  {
    timeout = true;
  }
}
if ( timeout )
{
  //Serial.println("NRFerror");
```

```
    NRFsend(NRFPack);
  }
  //radio.openWritingPipe(pipes[1]);
  //radio.openReadingPipe(1,pipes[0]);
}

void NRFreceive(){
  if ( radio2.available() )
  {
    bool done = false;
    while (!done)
    {
      done = radio2.read( &ReceivePackage, sizeof(ReceivePackage) );
    }
    //radio.stopListening();
    inputString = ((char *)ReceivePackage);
    stringComplete = true;
    radio.write( "1", 1 );
    //radio.startListening();
  }
}

void Serialreceive(){
  if (stringComplete) {
    if (inputString.startsWith("T:")) {
      int i = 2;
      Serial.println(inputString);
```

```
  for(i=2;i<inputString.length()-2;i+=28)
  {
    NRFsend(inputString.substring(i,i+28));
  }
}
else if (inputString.startsWith("S:")) {
  outputString += inputString.substring(2);
  inputString = "";
  if(outputString.endsWith("Gu:"))
  {
    //Serial.println(outputString);
    byte outputByte[(outputString.length()-3)/3];
    int y = 0,x = 0;
    //String = outputString.substring(0,outputString.length()-3);
    //mySerial.print(outputString.substring(0,outputString.length()-3));
    //Serial.print(outputString.substring(0,outputString.length()-3));
    for(y = 0;y<outputString.length()-3;y+=3)
    {
      outputByte[x] = outputString.substring(y,y+3).toInt();
      //mySerial.write(outputString.substring(y,y+3).toInt());
      //Serial.println(outputString.substring(y,y+3).toInt());
      x += 1;
    }
    mySerial.write(outputByte,sizeof(outputByte));
    Serial.write(outputByte,sizeof(outputByte));
    Serial.println();
    outputString = "";
```

```
    }
   }
   //inputString = "";
   stringComplete = false;
  }
}
```

7.2.2. Microcontrolador que recebe dados do CLP

```
#include <SPI.h>
#include <SoftwareSerial.h>
#include "nRF24L01.h"
#include "RF24.h"

RF24 radio(8,7);
RF24 radio2(4,3);
SoftwareSerial mySerial(5, 6); // RX, TX
// Radio pipe addresses for the 2 nodes to communicate.
const uint64_t pipes[2] = {
  0xF0F0F0F0E1LL, 0xE0E0E0E0D2LL };

//for Serial input
String inputString = "";         // a string to hold incoming data
String outputString = "";         // a string to hold incoming data
boolean stringComplete = false;  // whether the string is complete
```

```
//NRF Packages
byte SendPackage[32];
byte ReceivePackage[32];
boolean sending=0;


void setup(void)
{
 //
 // Print preamble
 //


 mySerial.begin(19200);
 Serial.begin(115200);
 //mySerial.begin(9600);
 //Serial.begin(9600);
 radio.begin();
 radio2.begin();
 // optionally, increase the delay between retries & # of retries
 radio.setDataRate(RF24_2MBPS);
 radio.setPALevel(RF24_PA_MAX);
 radio.setChannel(85);
 radio2.setDataRate(RF24_2MBPS);
 radio2.setPALevel(RF24_PA_MAX);
 radio2.setChannel(115);
 // delay 1ms, 3 retries
 // 0 - 250us, 15 - 4000us
```

```
  radio.setRetries(0,0);
  radio.setPayloadSize(32);
  radio2.setRetries(0,0);
  radio2.setPayloadSize(32);
  radio2.openWritingPipe(pipes[0]);
  radio.openReadingPipe(1,pipes[1]);
  radio.startListening();
  radio.printDetails();
  radio2.stopListening();
  radio2.printDetails();
}

void loop(void)
{
  NRFreceive();
  Serialreceive();
  inputString = "T:";
  inputString += "S:";
  while (mySerial.available())
  {
    byte inByte = (byte)mySerial.read();
    Serial.println(inByte);
    if(inByte < 100)
    {
      inputString += "0";
      if(inByte < 10)
      {
```

```
      inputString += "0";
     }
    }
    inputString += inByte;
    stringComplete = true;
    unsigned long started_waiting_at2 = millis();
    bool timeout2 = false;
    while(!mySerial.available()  && ! timeout2)
    {
     if (millis() - started_waiting_at2 > 1 )
     {
      timeout2 = true;
     }
    }
  }
  inputString += "Gu:";
  Serial.println(inputString);
}

byte NRFsend(String NRFPack = ""){
  Serial.println("NRFsend");
  if (!NRFPack.startsWith("S:")) {
    NRFPack = "S:" + NRFPack;
  }
  Serial.println(NRFPack);
  Serial.println("send");
  NRFPack.getBytes(SendPackage, 32);
```

```
  //radio.stopListening();
  //radio.openWritingPipe(pipes[0]);
  //radio.openReadingPipe(1,pipes[1]);
  bool ok = false;
  while(!ok)
  {
    ok = radio2.write(SendPackage,sizeof(SendPackage));
  }
  //radio.startListening();
  unsigned long started_waiting_at = millis();
  bool timeout = false;
  while ( ! radio.available() && ! timeout )
  {
    if (millis() - started_waiting_at > 250 )
    {
      timeout = true;
    }
  }
  if ( timeout )
  {
    //Serial.println("NRFerror");
    NRFsend(NRFPack);
  }
  //radio.openWritingPipe(pipes[0]);
  //radio.openReadingPipe(1,pipes[1]);
}
```

```
void NRFreceive(){
  Serial.println("NRFreceive");
  if ( radio.available() )
  {
    bool done = false;
    while (!done)
    {
      done = radio.read( &ReceivePackage, sizeof(ReceivePackage) );
    }
    //radio.stopListening();
    inputString = ((char *)ReceivePackage);
    stringComplete = true;
    radio2.write( "1", 1 );
    //radio.startListening();
  }
}


void Serialreceive(){
  if (stringComplete) {
    if (inputString.startsWith("T:")) {
      int i = 2;
      Serial.println(inputString);
      for(i=2;i<inputString.length()-2;i+=28)
      {
        NRFsend(inputString.substring(i,i+28));
      }
    }
```

```
else if (inputString.startsWith("S:")) {
  outputString += inputString.substring(2);
  inputString = "";
  if(outputString.endsWith("Gu:"))
  {
    //Serial.println(outputString);
    byte outputByte[(outputString.length()-3)/3];
    int y = 0,x = 0;
    //String = outputString.substring(0,outputString.length()-3);
    //mySerial.print(outputString.substring(0,outputString.length()-3));
    //Serial.print(outputString.substring(0,outputString.length()-3));
    for(y = 0;y<outputString.length()-3;y+=3)
    {
      outputByte[x] = outputString.substring(y,y+3).toInt();
      //mySerial.write(outputString.substring(y,y+3).toInt());
      //Serial.println(outputString.substring(y,y+3).toInt());
      x += 1;
    }
    mySerial.write(outputByte,sizeof(outputByte));
    Serial.write(outputByte,sizeof(outputByte));
    Serial.println();
    outputString = "";
  }
}
//inputString = "";
stringComplete = false;  }}
```