UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
ENGENHARIA DE CONTROLE E AUTOMAÇÃO

MILENA CHERUBINI JUSTI - 243702

ANÁLISE DO USO DE RTOS NA IMPLEMENTAÇÃO DE CONTROLE SUPERVISÓRIO

Porto Alegre
2020

MILENA CHERUBINI JUSTI - 243702

ANÁLISE DO USO DE RTOS NA IMPLEMENTAÇÃO DE CONTROLE SUPERVISÓRIO

Trabalho de Conclusão de Curso (TCC-CCA) apresentado à COMGRAD-CCA da Universidade Federal do Rio Grande do Sul como parte dos requisitos para a obtenção do título de *Bacharel em Eng. de Controle e Automação* .

ORIENTADOR:

Prof. Dr. Marcelo Götz

Porto Alegre
2020

MILENA CHERUBINI JUSTI - 243702

ANÁLISE DO USO DE RTOS NA IMPLEMENTAÇÃO DE CONTROLE SUPERVISÓRIO

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção dos créditos da Disciplina de TCC do curso *Engenharia de Controle e Automação* e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____
Prof. Dr. Marcelo Götz , UFRGS
Doutor pela Universität Paderborn – Paderborn, Alemanha e pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Banca Examinadora:

Prof. Dr. Marcelo Götz , UFRGS
Doutor pela Universität Paderborn – Paderborn, Alemanha e pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Prof. Dr. Ivan Müller , UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Prof. Dr. Renato Ventura Bayan Henriques , UFRGS
Doutor pela Universidade Federal de Minas Gerais – Belo Horizonte, Brasil

_____
Marcelo Götz
Coordenador de Curso
Engenharia de Controle e Automação

Porto Alegre, novembro de 2020.

DEDICATÓRIA

Dedico este trabalho aos meus pais, que sempre me apoiaram nas minhas escolhas.

AGRADECIMENTOS

Aos meus pais e meu irmão pelo apoio, suporte, paciência e carinho durante os vários anos de faculdade.

Aos meus amigos e colegas do curso, que possibilitaram ótimos momentos na universidade e tornaram o estudo mais enriquecedor.

Aos meus professores, principalmente ao meu orientador, que contribuíram muito para meu desenvolvimento profissional.

À Universidade Federal do Rio Grande do Sul pela oportunidade de realização de estudos.

Às minhas amigas, aos meus amigos e à minha família, que de alguma forma se fizeram presentes durante a escrita deste trabalho em meio à pandemia de coronavírus.

À minha melhor amiga, Bruna de Paula Monarin, pela revisão deste trabalho.

RESUMO

Com o avanço tecnológico, sistemas a eventos discretos estão cada vez mais presentes. Para controle destes sistemas, são fomentadas pesquisas a respeito da implementação da teoria de controle supervisório em microcontroladores. Apesar de avanços na área, ainda existem lacunas entre o desenvolvimento acadêmico e a implementação prática dos supervisórios, as quais fazem com que estes não sejam amplamente empregados no meio industrial. Neste contexto, este trabalho faz análise do aspecto temporal relativo a execução do controle supervisório em um microcontrolador com escalonador de tempo real. A abordagem desenvolvida utiliza tarefas periódicas e assíncronas e uma arquitetura de controle própria para implementação dos supervisórios. Além disso, é implementado o rastreio da execução do escalonador para coleta de dados. Os resultados obtidos com uma planta simulada evidenciam a validade do trabalho desenvolvido e contribuem para sanar questões relativas ao funcionamento das estruturas internas do controlador.

Palavras-chave: TCS, SED, Controle Supervisório, Microcontrolador, RTOS.

ABSTRACT

Due to technological progress, the presence of discrete event systems is increasing. In order to control these systems, research on the implementation of supervisory control theory in microcontrollers is being promoted. Despite of improvements in this area, there are still some gaps between academic development and the practical implementation of the supervisors, what implies that they are not widely used in the industry yet. In this context, this work analyzes the temporal aspect related to the execution of the supervisory control in a microcontroller with a real-time scheduler. The developed approach uses periodic and asynchronous tasks and its own control architecture for the implementation of the supervisors. In addition, this work implements the scheduler execution data tracking. The results obtained with a simulated manufacturing factory show the validity of the work developed and contribute to address issues related to the functioning of the internal structures of the controller.

Keywords: SCT, DES, Supervisory Control, Microcontroller, RTOS.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

RTOS	*Real-Time Operating System*
TCS	Teoria de Controle Supervisório
SED	Sistemas de Eventos Discretos
SF1	*Stack Feeder* 1
DS1	*Distribuction System* 1
PM1	*Process Machine* 1
XS1	*Exit Slider* 1
XS2	*Exit Slider* 2

1 INTRODUÇÃO

Nas últimas décadas, o avanço da tecnologia empregada em sensores, poder computacional e protocolos de comunicação contribuiu para o rápido crescimento de sistemas dinâmicos com sinais como: clique do usuário em uma interface computacional, borda de subida de um alarme, início ou fim de um processo, percepção de mudança de estado de um sensor, recebimento de mensagens, entre outros. Estes sinais tem como característica serem instantâneos e assíncronos e são chamados de eventos.

Decorrente da crescente eclosão de sistemas orientados a eventos, principalmente em fábricas de manufatura, a modelagem e o controle destes sistemas tornam-se cada vez mais necessárias. Para suprir esta demanda, os sistemas a eventos discretos são alvo de pesquisa acadêmica, sendo a teoria de controle supervisório a principal ferramenta para garantir o comportamento desejado destes sistemas.

Apesar dos já comprovados resultados que podem ser obtidos pelos supervisórios, há lacunas entre as pesquisas acadêmicas e a implementação prática dos supervisórios. Parte da existência dessas lacunas decorre da maioria dos sistemas fabris fazer o controle de seus processos através de CLPs, que devido a estrutura de suas linguagens, não permite que os supervisórios sejam programados de maneira robusta e mantendo todas suas características.

Como resposta a este problema, é fomentada a pesquisa com o uso de microcontroladores para implementação prática dos supervisórios. Mesmo que o uso destes seja mais vantajoso que dos CLPs, algumas questões ainda precisam ser respondidas para difusão da teoria de controle supervisório a nível industrial. Estas questões são relacionadas à robustez da implementação e garantia de preservação das características dos supervisórios abstratos quando codificados em um microcontrolador.

A conversão de um autômato supervisório para um código a ser implementado em microcontroladores já foi abordada por Pereira (2014), Carvalho (2015) e Carvalho et al. (2019). Dessa forma, este trabalho tem foco em outro tópico relevante à implementação prática de supervisórios em microcontroladores: o aspecto temporal. Sendo assim, utilizando-se um escalonador de tempo real, o principal objetivo deste trabalho é a análise temporal da execução do controle supervisório em um microcontrolador, preenchendo lacunas na literatura específica relativas a este tema.

A fim de embasar o desenvolvimento do controlador desenvolvido, a discussão deste trabalho inicia com uma breve descrição de sistemas a eventos discretos, sua representação em autômatos e uma visão geral da teoria de controle supervisório no Capítulo 2. Ainda neste capítulo, revisou-se quais são as principais fontes de problemas da implementação prática de supervisórios e soluções já existentes para estes.

No Capítulo 3 são abordados os materiais necessários tanto para desenvolvimento do controlador quanto para execução do teste que o valida. Estes materiais são um conjunto de *hardware* e *software*, que integrados possibilitaram o funcionamento do controle

supervisório em um microcontrolador e a análise de sua performance e características.

A elaboração do desenvolvimento em si é abordado no Capítulo 4. Neste capítulo propõe-se uma arquitetura de controle em conjunto com uma estrutura de tarefas e bibliotecas para implementação de um supervisório modular genérico em um microcontrolador. Também neste capítulo, é apresentado um *layout* de fábrica a ser controlado e o desenvolvimento específico de um supervisório para esta planta em questão.

A validação do controlador foi realizada através do teste do supervisório desenvolvido agindo sobre uma planta de manufatura simulada. Os resultados obtidos neste teste são exibidos e discutidos no Capítulo 5. Os resultados incluem o comportamento interno das estruturas do controlador durante o teste e a comparação da ação de controle deste com a de outro controlador obtido em um *software* com credibilidade já atestada.

Por fim, o Capítulo 6 encerra o trabalho com a apresentação de conclusões que podem ser feitas a partir deste. Ainda, são sugeridos trabalhos futuros que abrangem pontos não explorados neste trabalho a fim de complementação do estudo realizado através de testes em condições distintas e expansão da estrutura de controle.

2 REVISÃO BIBLIOGRÁFICA

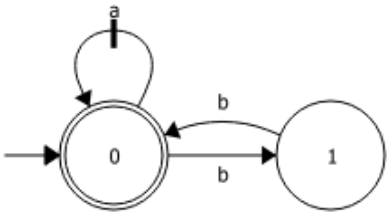
Neste capítulo é realizada uma breve revisão a respeito de sistema a eventos discretos na Seção 2.1 e da teoria de supervisórios na Seção 2.2. Após, na Seção 2.3 são abordados os principais problemas relativos a implementação de supervisórios em microcontroladores reconhecidos pelos autores da área de controle de sistemas a eventos discretos.

2.1 Sistemas a Eventos Discretos

Um Sistema a Eventos Discretos (SED) possui espaço de estados discreto e seu estado é definido pelo histórico de eventos ocorridos. Sendo assim, o evento é a variável independente e é o que causa as transições de estado. Segundo Cassandras e Lafortune (2008a), um SED é modelado através de sua linguagem gerada e sua linguagem marcada. A linguagem gerada é o conjunto de todas as sequências de eventos que podem ocorrer fisicamente no SED em questão. Enquanto a linguagem marcada é um subconjunto desta última que engloba todas as sequências de eventos que representam tarefas completas.

Os autômatos são uma maneira de representar as linguagens através de um grafo direcionado, onde são representados os estados em círculo e as transições em arco com o evento em questão indicado. A Figura 1 mostra um autômato com dois estados. No estado *0*, a ocorrência do evento *a* resulta no próprio estado *0* e a ocorrência do evento *b*, no estado *1*. No estado *1*, o evento *a* não pode ocorrer e, caso ocorra o evento *b*, o estado resultante é o estado *0*. Nota-se que a transição do evento *a* possui um traço, este símbolo representa um evento controlável e sua omissão, um evento não controlável. Também, o estado *0* é representado por um círculo duplo, o que significa que o mesmo é marcado.

Figura 1: *Representação de um SED por um autômato.*



Fonte: Autor

Os autômatos podem ser manipulados e pode-se realizar operações com eles. Existem operações que permitem combinar dois ou mais autômatos que representam subsistemas para obtenção de um modelo completo. Estas operações são associativas e podem ser de produto ou de composição paralela. Na operação produto (símbolo $\times$), apenas os eventos pertencentes a intersecção do alfabeto dos autômatos sendo operados farão parte

do alfabeto do autômato resultante. Já na operação paralela (símbolo $\|$), acrescentam-se a estes os eventos presentes em apenas um dos autômatos (CASSANDRAS; LAFORTUNE, 2008a).

De acordo com Cassandras e Lafortune (2008a), também é possível realizar operações em um único autômato, dentre estas as operações *acessível*, *co-acessível* e *Trim*. Dado um autômato A, a operação *acessível(A)* retira de A os estados inacessíveis, ou seja, aqueles não podem ser acessados por nenhuma cadeia de eventos. A operação *co-acessível(A)* retira de A os estados a partir dos quais não é possível chegar em nenhum estado marcado. Finalmente, a operação *Trim(A)* é equivalente a realização de ambas as operações anteriores no autômato A.

2.2 Teoria de Controle Supervisório

A Teoria de Controle Supervisório (TCS) diz respeito ao cálculo de supervisórios – controladores – para SED. Para aplicação da TCS, o sistema a ser controlado, aqui chamado de planta, é descrito por um autômato G, que representa o comportamento livre do sistema. Por exemplo, em uma fábrica de manufatura as máquinas A e B iniciam um certo processamento sempre que recebem uma peça.

Então, a este sistema são impostas especificações de funcionamento, descritas por um autômato H_s, que restringem o comportamento da planta a fim de satisfazer algum requisito de controle. Por exemplo, as máquinas A e B não podem entrar em funcionamento simultaneamente. A partir de G e H_s é o obtido o autômato $H = H_s \parallel G$, que é um supervisório e apresenta o comportamento da planta atuando com a restrição.

Existem duas características desejáveis a um supervisório: que este seja controlável e não bloqueante. A controlabilidade é a propriedade que determina que após a execução de uma cadeia de eventos s em G, todos os eventos não controláveis previstos de ocorrer em G sejam não desabilitados pelo supervisório após a execução da mesma cadeia s. Já a propriedade de não bloqueio determina que a partir de qualquer cadeia s é possível que o sistema chegue em um estado marcado (CASSANDRAS; LAFORTUNE, 2008b).

A partir de H, obtém-se o controlador ótimo $SupC(L_{am})$, que é o componente supremo de L_{am} (linguagem marcada admissível de H) conhecido como a máxima linguagem controlável não bloqueante, ou linguagem marcada menos restritiva possível e controlável, que está contida em L_{am}. O cálculo de $SupC(L_{am})$ é realizado iterativamente: se L_{am} não for controlável, deletam-se de H os estados que violem a condição de controlabilidade e aplica-se a operação *Trim(H)* até que seja obtido um autômato que não viole a condição de controlabilidade e que seja não bloqueante (CASSANDRAS; LAFORTUNE, 2008b).

Devido a todas operações de composição envolvidas no cálculo do supervisório $SupC(L_{am})$, este pode facilmente resultar em um autômato com número elevado de estados e transições. Por este motivo, sua implementação prática pode ser desfavorável devido ao uso excessivo de memória e poder computacional necessário para seu cálculo. Como alternativa, é possível modularizar o supervisório em partes cuja composição resultante tenha comportamento equivalente ao $SupC(L_{am})$, porém sem ser necessário calculá-lo.

Uma das abordagens para isto é a de supervisórios modulares locais, proposta por Queiroz e Cury (2002). A metodologia para obtenção dos supervisórios modulares locais inicia com a identificação de n subsistemas assíncronos, ou seja, sistemas cuja intersecção de seus alfabetos é nula. Estes subsistemas, aqui denominados por G_i com alfabeto E_i, modelam os autômatos de parte da planta e a composição de todos estes resulta na planta completa a ser controlada.

O sistema também é composto por especificações $H_{s,j}$ – com alfabeto E_j – que restringem o livre comportamento da planta. Para obtenção das plantas locais, deve-se analisar o alfabeto de cada especificação e comparar com o alfabeto das plantas, de maneira que uma planta local é dada pela composição paralela entre todas as plantas que possuem eventos em comum com a especificação em questão. Sendo assim, Queiroz e Cury (2002) define uma planta local por:

$$G_j^{local} = \| \; G_i \quad | \quad i \in \{1,...,n\} \quad e \quad E_i \cap E_j \neq \emptyset.$$

Ao sistema local, formado pela planta local e sua especificação, aplica-se a operação de composição paralela para obtenção de $H_j = H_{s,j} \parallel G_j$. Então, tem-se que $L_{am,j} = \mathscr{L}_m(H_j)$ e R_j como a representação por autômato de $SupC(L_{am,j})$, que possui as propriedades de controlabilidade e não bloqueio.

A ação de controle resultante, ou supervisório resultante $SupC_m(s)$, é obtida pela intersecção das ações de controle de cada supervisório modular local:

$$SupC_m(s) = SupC_1(s) \cap SupC_2(s) \cap ... \cap SupC_n(s).$$

Assim, após obtenção dos supervisórios modulares locais, ainda é necessário verificar se o supervisório resultante é não bloqueante. Esta verificação é calculada pela seguinte expressão:

$$(R_1 \parallel R_2 \parallel ... \parallel R_n) \stackrel{?}{=} Trim(R_1 \parallel R_2 \parallel ... \parallel R_n),$$

com R sendo o autômato que representa $SupC(L_{am,j})$. O resultado positivo desta equação significa que o sistema resultante é não bloqueante (QUEIROZ; CURY, 2002).

2.3 Implementação de Supervisórios em Microcontroladores

Apesar da aceitação da TCS no meio acadêmico, o uso da TCS em cenários industriais ainda é escasso. Isso se deve principalmente aos problemas decorrentes da implementação física do supervisório após a obtenção de seu modelo abstrato. Segundo Fabian e Hellgren (1998), as principais classes de problemas relacionados à implementação física da TCS são:

- **Sinais e eventos.** A TCS considera que eventos ocorrem assincronamente em instâncias de tempo discretos, enquanto geralmente as aplicações reais atualizam a leitura dos sinais da planta periodicamente. Se dois ou mais sinais mudam de valor no mesmo intervalo entre duas leituras, não é possível identificar a ordem de ocorrência destes.

- **Causalidade.** A TCS espera que a planta gere espontaneamente todos os eventos e o supervisório apenas os desabilite dinamicamente quando necessário, o que não ocorre numa planta real. A implementação real precisa de algo – geralmente o controlador – que gere os eventos controláveis, deixando a cargo da planta apenas a geração dos eventos não controláveis. Este problema é estritamente intrínseco a TCS.

- **Escolha.** Este problema está relacionado ao requisito dos supervisórios serem minimamente restritivos, o que faz com a planta a ser controlada tenha máxima liberdade. Isto implica que o controlador pode calcular mais de um evento controlável não desabilitado e apenas um deles deve ser escolhido para envio a planta. A

implementação do método de escolha de eventos pode levar um supervisório não bloqueante a ser bloqueante ou a uma situação de *deadlock* ou *livelock*.

- **Sincronização inexata.** Devido ao controlador ler periodicamente os sinais da planta, é possível que uma mudança de sinal não controlável ainda não detectada pelo controlador invalide a ação de controle calculada. Assim, após a detecção do evento não controlável, o supervisório pode estar num estado incorreto ou inválido.

Em Lopes et al. (2012) são recomendadas algumas soluções para os problemas listados, entre elas as seguintes:

- **Sinais e eventos.** Uso de interrupções para leitura dos sinais de entrada e armazenar os eventos em um *buffer*. O período de leitura é determinado pelo *designer* da aplicação.
- **Causalidade.** Implementação de uma estrutura de controle que forneça uma interface entre a planta e o supervisório, gerando os eventos a partir dos sinais lidos.
- **Escolha.** Geração de números aleatórios para seleção de eventos a partir de uma entrada externa, uma vez que microcontroladores são determinísticos e apenas capazes de gerar números pseudoaleatórios.
- **Sincronização inexata.** Verificação do estado do *buffer* que armazena os eventos não controláveis. Se este está cheio, o cálculo da ação de controle deve ser refeito antes de sua aplicação na planta.

3 MATERIAIS

A planta deste trabalho foi simulada no *software* FlexFact. O controlador foi inicialmente projetado no *software* DESTool e, então, testado em conjunto com a planta simulada. Após, o controlador projetado no DESTool foi implementado em um microcontrolador Arduino Mega – com adição de um *shield* com interface Ethernet – rodando o *kernel* FreeRTOS. O microcontrolador tem comunicação com o FlexFact estabelecida através do protocolo Modbus TCP/IP. As seções a seguir abordam brevemente estes materiais.

3.1 FlexFact

O FlexFact é uma ferramenta de simulação de sistemas de manufatura flexíveis desenvolvida pela Friedrich-Alexander-Universität Erlangen-Nürnberg. O simulador imita o comportamento dinâmico de componentes eletromecânicos – como esteiras, máquinas de processo, distribuidores, pilha de alimentadores, entre outros – e o projeto do *layout* da fábrica é facilmente configurável. O controle dos componentes é dado através de sinais virtuais externos ao simulador, os quais assumem valores binários e compõem a imagem do processo (MOOR, 2020b).

O FlexFact fornece duas interfaces para acessar a imagem do processo: Modbus TCP/IP e simplenet (para uso em conjunto com ferramentas do mesmo desenvolvedor). Para a interface Modbus TCP/IP (utilizada neste trabalho), o simulador assume a função de servidor e o controlador de cliente, decidindo quais sinais devem ser lidos e seus valores a serem enviados para comandar a planta (MOOR, 2020b).

3.2 DESTool

Segundo Moor (2020a), o DESTool é uma ferramenta para a síntese e análise de SEDs e é implementado como uma interface gráfica para a biblioteca de sistemas de eventos discretos libFAUDES. Por sua vez, a biblioteca libFAUDES implementa estruturas de dados e algoritmos para autômatos finitos e linguagens regulares e é também desenvolvida pela Friedrich-Alexander-Universität Erlangen-Nürnberg. Além disso, o DESTool possui interface para conexão com o FlexFact através do protocolo simplenet e, nesta configuração, atua como controlador da planta simulada.

3.3 Arduino Mega e *Shield* Ethernet

Arduino é uma plataforma eletrônica criada no Ivrea Interaction Design Institute como uma ferramenta de prototipagem rápida. As placas Arduino, assim como seu

software, são totalmente de código aberto, o que permite o uso da plataforma em conjunto com contribuições de terceiros. Estas contribuições podem se dar na forma de *shields*, bibliotecas e até outras placas microcontroladas baseadas na plataforma (ARDUINO, 2018).

Neste trabalho foi utilizado o Arduino Mega 2560, uma placa projetada com o microcontrolador ATmega2560. Para comunicação entre o Arduino e a planta simulada, necessita-se de *hardware* adicional que possua interface Ethernet. O *hardware* extra escolhido é o Arduino Ethernet Shield W5100, o qual é facilmente acoplado sobre a placa. Além disso, o FreeRTOS tem portabilidade com o Arduino Mega 2560, podendo ser incluído ao projeto na forma de uma biblioteca.

3.4 Modbus TCP/IP

Modbus é um protocolo de mensagens correspondente a camada de aplicação do modelo OSI. Ele fornece comunicação cliente-servidor entre dispositivos conectados em diferentes tipos de barramentos ou redes. A troca de mensagens é do tipo solicitação-resposta – onde a solicitação é feita pelo cliente e a resposta é dada pelo servidor – e oferece serviços especificados por códigos de função pré-definidos (MODBUS..., 2012).

Neste trabalho, o protocolo Modbus é encapsulado em uma mensagem TCP/IP sobre Ethernet. Ainda neste contexto, a planta simulada no FlexFact atua como servidor e o microcontrolador como cliente. Apenas são utilizadas as funções de leitura e escrita de *coils* (registrador definido pelo protocolo para armazenamento de dados com tamanho de 1 *bit*), cujos endereços de memória são definidos automaticamente pelo FlexFact.

3.5 FreeRTOS

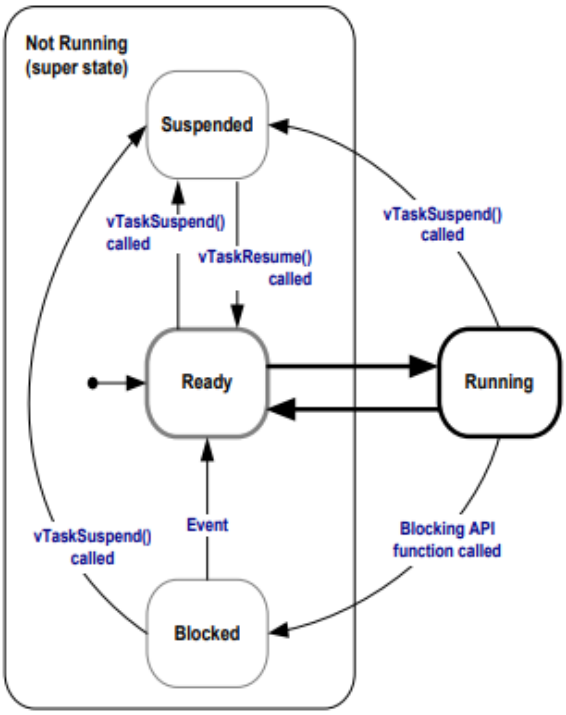
O FreeRTOS, desenvolvido e mantido pela Real Time Engineers Ltd., é um *software* RTOS completamente gratuito distribuído sob a licença de código aberto do MIT. O FreeRTOS é pensado para aplicações de tempo real embarcadas baseadas em microcontroladores ou pequenos microprocessadores, uma classe de aplicação que normalmente abrange uma combinação de requisitos *hard* e *soft* de tempo real (FREERTOS, 2020).

O FreeRTOS é fornecido como uma coleção de arquivos de código fonte em linguagem C. Alguns destes possuem macros que podem ser setadas de acordo com a necessidade da aplicação em questão. Estas macros agregam flexibilidade e portabilidade ao FreeRTOS, o que faz com ele seja compatível com mais de vinte diferentes compiladores e mais de trinta diferentes arquiteturas de processadores, incluindo o ATmega2560.

O FreeRTOS é um *kernel* de tempo real (ou escalonador de tempo real) que permite que a aplicação desenvolvida seja organizada como um conjunto de *threads* de execução independentes, aqui chamadas de tarefas. Em um processador que possui apenas um núcleo, como no caso do Arduino Mega 2560, apenas uma única tarefa pode estar em execução por vez. O escalonador decide qual tarefa deve ser executada examinando a prioridade atribuída a cada tarefa pelo *designer* da aplicação.

As tarefas criadas neste trabalho são orientadas a eventos, ou seja, elas são executadas apenas após o disparo de algum evento. Uma tarefa orientada a eventos pode estar em 4 estados: Pronto (*Ready*), Executando (*Running*), Bloqueado (*Blocked*) e Suspenso (*Suspended*), como é ilustrado na Figura 2. Somente tarefas no estado Executando estão em execução, enquanto as demais tarefas encontram-se no super-estado Não Executando (*Not Running*). As funções em azul com o prefixo "*vTask*" e "*Blocking API function called*" podem apenas ser chamadas pela tarefa que ocupa o estado Executando.

Figura 2: *Máquina de estados do FreeRTOS.*



Fonte: Barry (2016a)

Uma tarefa está no estado Bloqueado quando está aguardando por um evento. Este evento pode ser: temporal, como no caso de tarefas periódicas; recebimento de dados em uma fila (estrutura abordada a seguir); liberação de um *mutex* (denominado semáforo no FreeRTOS); notificação vinda de outra tarefa; ou uma combinação destes (BARRY, 2016a).

Tarefas no estado Suspenso não estão disponíveis para o escalonador e somente existem transições neste estado pelas chamadas explicitas das funções *vTaskSuspended* e *vTaskResume*. Geralmente transições com este estado são utilizadas como tratamento de interrupções de *hardware* (BARRY, 2016a).

Tarefas no estado Pronto estão prontas para entrar em execução, entretanto ainda estão no aguardo de serem chamadas pelo escalonador. O escalonador sempre irá escolher a tarefa com maior prioridade para estar no estado Executando. Quando há mais de uma tarefa com a prioridade mais alta, o escalonador escolherá a que está em Pronto há mais tempo e, caso configurado, haverá revesamento de execução a cada *time slice* (intervalo de tempo periódico determinado com base no *clock* do microcontrolador) (BARRY, 2016a).

Deve haver sempre uma tarefa no estado Executando, para isso o FreeRTOS automaticamente cria a tarefa Idle quando o escalonador é inicializado. A tarefa Idle está sempre habilitada para execução, ou seja, não pode entrar em bloqueio nem ser suspensa. Esta tarefa é configurada com a mais baixa prioridade possível, de maneira que ocorra transição imediata sempre que houver outra tarefa com maior prioridade disponível para execução. Além disso, a tarefa Idle é responsável por limpar os recursos do *kernel* ocupados por uma tarefa após a mesma ser deletada (BARRY, 2016a).

3.5.1 Filas

Uma fila é um instrumento que possibilita a comunicação entre tarefas. Dentre os tipos de filas disponíveis no FreeRTOS, utilizou-se as do tipo *First In First Out*, onde dados

são escritos por uma tarefa no final da fila e lidos por outra tarefa no início da mesma. O FreeRTOS implementa o sistema de "fila por cópia", onde os dados são copiados *byte* a *byte* para a fila ao invés de passar um ponteiro para o dado que deseja-se transmitir.

Quando uma tarefa tenta ler dados de uma fila é especificado um tempo de bloqueio. Este parâmetro define o tempo máximo em que uma tarefa estará no estado Bloqueado até que haja algum dado disponível para leitura. Quando um novo dado chega na fila ou o tempo de bloqueio é expirado, uma tarefa no estado Bloqueado é imediatamente movida para o estado Pronto (BARRY, 2016b).

As filas tem tamanho limitado e definido pelo usuário, isso significa que a memória reservada para uma determinada fila pelo *kernel* pode estar completamente preenchida. Analogamente ao processo de leitura da fila, na escrita de dados também pode ser especificado um tempo de bloqueio para aguardar a liberação de memória da fila (BARRY, 2016b).

Vale mencionar que o tempo de bloqueio pode ser configurado pela macro *portMAX_DELAY*, que define que uma tarefa esperará no estado Bloqueado por tempo indeterminado. Ou seja, este parâmetro previne transições para o estado Pronto devido a expiração do tempo de bloqueio. Na implementação do controlador deste trabalho todas as filas utilizam a macro *portMAX_DELAY*.

3.5.2 Semáforos

Semáforos binários são a implementação do FreeRTOS de *mutexes* para compartilhamento de recursos entre duas ou mais tarefas. Analogamente ao caso das filas, os semáforos possuem tempo de bloqueio a ser especificado. Quando uma tarefa tenta "pegar" um semáforo que já está com outra tarefa, a primeira vai para o estado de bloqueio e transiciona para o estado Pronto apenas quando a segunda libera o acesso ao semáforo ou o tempo de bloqueio expira (BARRY, 2016c).

3.5.3 Rastreio do Escalonador

O FreeRTOS possui recursos que possibilitam analisar a execução da aplicação. Dentre as funções já programadas do *kernel*, é possível gerar estatísticas a respeito do tempo que cada tarefa esteve no estado Executando desde o início da execução da aplicação e, além disso, pode-se obter uma captura do estado atual de cada tarefa. Estas funções devem ser usadas apenas durante o processo de desenvolvimento da aplicação, pois sua execução suspende o escalonador por tempo demasiado longo, uma vez que utilizam recursos como imprimir informações na tela. Também por este motivo, os dados obtidos com estas funções podem ser consideravelmente diferentes do que seria observado caso elas não estivessem sendo executadas (BARRY, 2016d).

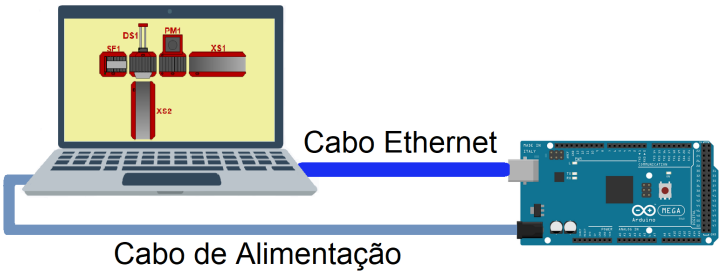
Uma das maneiras de contornar este problema das funções de monitoramento definidas pelo FreeRTOS é através de macros de rastreamento. Estas macros são chamadas pelo escalonador em momentos específicos, como por exemplo: incremento do contador de *ticks*; troca de contexto (transição de estado das tarefas); novo envio bem sucedido para uma fila; tentativa bem sucedida de "pegar" um semáforo; bloqueio de tarefa devido a fila vazia, fila cheia, atraso (tarefas temporais/periódicas) e semáforo indisponível.

Por padrão, as macros são definidas como funções vazias. É possível definir funções para apenas as macros que deseja-se utilizar. Neste trabalho, para rastreamento da execução da estrutura de controle, foram definidas funções para as macros chamadas durante as trocas de contexto do escalonador.

4 DESENVOLVIMENTO DO TRABALHO

Foi desenvolvido um controlador baseado na TCS em um microcontrolador Arduino para uma planta de manufatura simulada no *software* FlexFact, que por sua vez estava executando em um computador com o sistema operacional Windows. A Figura 3 exibe o esquemático do sistema utilizado para desenvolvimento do controlador.

Figura 3: *Esquemático do sistema utilizado para desenvolvimento do controlador.*



Fonte: Autor

O desenvolvimento do trabalho foi divido em partes. A primeira delas aborda a implementação de um controlador baseado na TCS em um microcontrolador de maneira genérica e independente da planta. A segunda parte descreve a planta utilizada para teste do controlador. Finalmente, a última parte envolve a modelagem dos autômatos no DESTool e obtenção dos supervisórios.

4.1 Controlador Baseado na TCS em um Microcontrolador

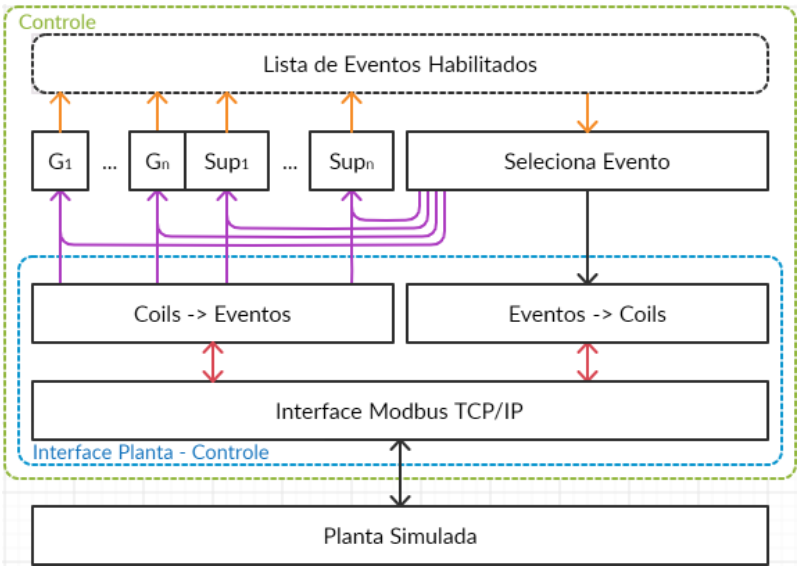
O desenvolvimento do controlador baseado na TCS em um microcontrolador utiliza a arquitetura de controle abordada na Seção 4.1.1. Esta arquitetura foi implementada através de tarefas e bibliotecas, as quais são detalhadas nas seções 4.1.2 e 4.1.3, respectivamente. Após, a Seção 4.1.4 aborda uma visão geral do código desenvolvido e, por fim, ilustra o comportamento previsto das estruturas internas do controlador.

4.1.1 Arquitetura do Controlador

Para implementação em um microcontrolador de um controlador baseado na TCS foi desenvolvida a arquitetura de *software* exibida na Figura 4.

Os sinais referentes aos sensores e atuadores eletromecânicos que compõem a planta simulada no FlexFact são comunicados ao microcontrolador através do protocolo Modbus TCP/IP. Conforme abordado na Seção 3.4, o valor binários de cada sinal é armazenado

Figura 4: *Arquitetura da implementação do controlador em um micro-controlador.*



Fonte: Autor

em um endereço *coil* e é informado ao controlador apenas após este dado ser solicitado pelo mesmo. O controlador baseado na TCS é calculado a partir de eventos; sendo assim, é necessário criar uma interface entre a planta e o controlador que "traduza" *coils* para eventos e vice-versa. Esta interface, denominada Interface Planta-Controle, é exibida na Figura 4 contornada em azul.

A Interface Planta-Controle solicita a leitura dos *coils*, verifica a partir dos valores obtidos se houve a ocorrência de algum evento e, em caso positivo, qual deles. Analogamente, após o calculo do controlador é necessário enviar o evento controlável selecionado para a planta simulada. Para este processo, a interface converte o evento selecionado para o endereço de *coil* correspondente e o valor que este deve assumir e, então, estes são enviados do microcontrolador para a planta simulada.

As conversões de *coils* para eventos e de eventos para *coils* são realizadas por duas estruturas distintas: a tarefa Lê Coils e a tarefa Escreve Coils, respectivamente. Estas tarefas serão abordadas na Seção 4.1.2. Conforme a Figura 4, ambas as estruturas utilizam a interface Modbus TCP/IP do microcontrolador e para que este acesso seja corretamente sincronizado é utilizado o recurso semáforo do FreeRTOS.

Na estrutura de controle desenvolvida, tanto os autômatos que modelam o comportamento das máquinas quanto os supervisórios são tratados da mesma maneira. Todos são implementados como tarefas independentes que instanciam a mesma função-tarefa, a qual é tratada na Seção 4.1.2. Estas tarefas utilizam a estrutura de filas do FreeRTOS para recebimento dos eventos ocorridos.

Na Figura 4 as tarefas dos autômatos são ilustradas por G_1, G_n, Sup_1 e Sup_n e as filas estão representadas pelas setas em roxo. Cada tarefa possui uma fila independente e as estruturas "Coils -> Eventos" e "Seleciona Evento" verificam se o evento em questão pertence ao alfabeto do autômato antes de envia-lo para sua respectiva fila. A estrutura "Coils -> Eventos" é responsável por enviar os eventos não controláveis para os autômatos e a "Seleciona Evento", os eventos controláveis.

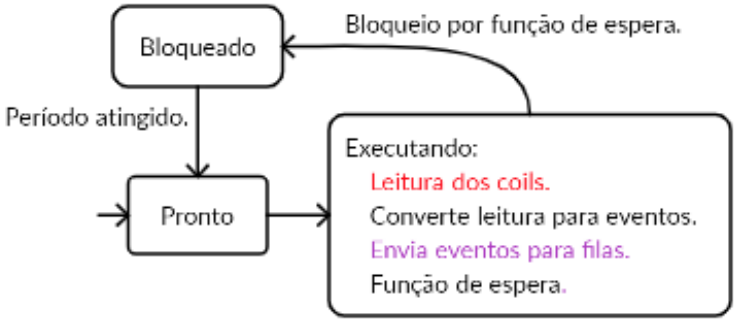
As tarefas dos autômatos escrevem o estado dos eventos controláveis pertencentes ao seu alfabeto num vetor, representado pelo elemento "Lista de Eventos Habilitados"

na Figura 4. A estrutura “Seleciona Evento” acessa essa lista para escolher um evento não desabilitado por todos os autômatos para ser enviado a planta simulada. O vetor de eventos é acessado pelos autômatos e pelo “Seleciona Evento”, sendo assim um recurso compartilhado. O acesso a ele é então sincronizado pelo uso de um semáforo.

4.1.2 Tarefas Implementadas

A implementação da arquitetura de controle é composta por três tipos de tarefas: Lê Coils, Autômatos e Escreve Coils. O fluxograma da primeira delas é exibido na Figura 5.

Figura 5: *Fluxograma de execução da tarefa Lê Coils.*

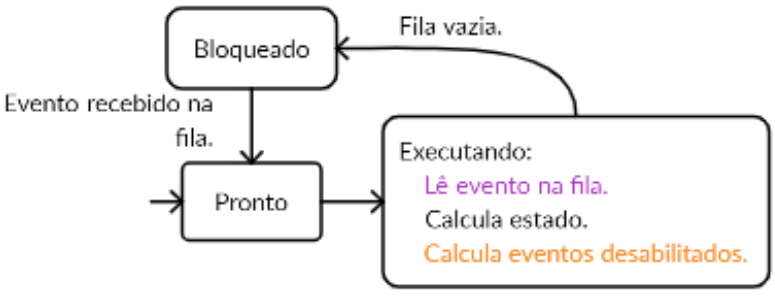


Fonte: Autor

A tarefa Lê Coils tem sua execução periódica e é a tarefa cuja prioridade é a mais alta de todas, possuindo valor igual a 3. Por este motivo, sempre que o período de espera é atingido e a tarefa transiciona do estado Bloqueado para o estado Pronto, o escalonador do FreeRTOS imediatamente coloca a tarefa Lê Coils em estado Executando. O ciclo de execução desta tarefa consiste na leitura de todos os endereços de *coils* correspondentes aos sinais da planta simulada; verificação de ocorrência de eventos a partir desta leitura; envio dos eventos não controláveis detectados para os autômatos cujo alfabeto contém o evento em questão.

A implementação dos autômatos das plantas locais e dos supervisórios se da pela instância da mesma função-tarefa. Estas tarefas tem seu fluxograma exibido na Figura 6.

Figura 6: *Fluxograma de execução das tarefas dos autômatos.*



Fonte: Autor

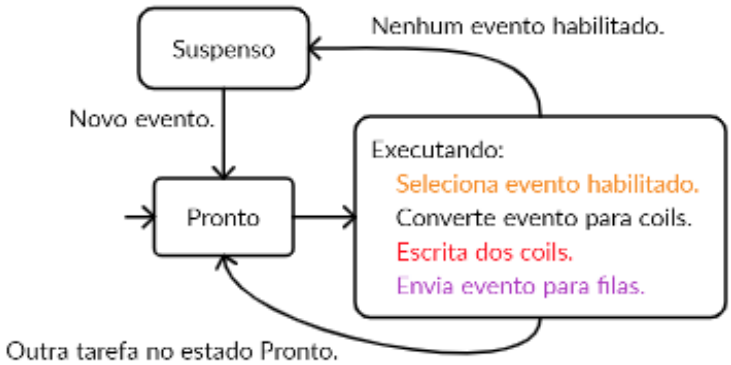
As tarefas dos autômatos possuem prioridade com valou igual a 2, são aperiódicas e executam apenas quando há algum evento que deva ser processado por elas. Os eventos são recebidos através de uma fila para cada tarefa e, portanto, a tarefa está em estado Bloqueado quando sua fila está vazia e, ao recebimento de um novo evento, a tarefa transiciona para o estado Pronto. Geralmente um evento pertencerá ao alfabeto de mais de um autômato logo,

mais de uma tarefa se encontrará no estado Pronto. Devido a todas as tarefas dos autômatos possuírem a mesma prioridade, o escalonador poderá escolher a ordem de execução delas e, como elas são independentes entre si, isto não afeta o cálculo do controlador.

Durante o ciclo de execução da tarefa de um autômato, o primeiro evento da fila é lido e, juntamente com a informação do estado atual do autômato, o próximo estado é calculado. Obtido o novo estado, verifica-se quais eventos controláveis estão habilitados e quais estão desabilitados e esta informação é escrita no vetor que armazena o status de cada evento. Terminado um ciclo, verificam-se se há outro evento na fila; em caso positivo, o ciclo descrito é repetido; em caso negativo, a tarefa transiciona para o estado Bloqueado.

A última tarefa é a Escreve Coils. O fluxograma correspondente a sua implementação é exibido na Figura 7. A Escreve Coils possui a menor prioridade entre as tarefas, com valor igual a 1. Sendo assim, esta somente será escolhida pelo escalonador do FreeRTOS para estar no estado Executando quando todas as demais tarefas estiverem no estado Bloqueado. Este cenário ocorre quando a tarefa Lê Coils já encaminhou os eventos não controláveis para os autômatos e estes, por sua vez, já atualizaram o vetor com os eventos controláveis habilitados e desabilitados.

Figura 7: *Fluxograma de execução da tarefa Escreve Coils.*



Fonte: Autor

A tarefa Escreve Coils engloba as estruturas “Seleciona Evento” e “Eventos -> Coils” da Figura 4. A seleção do evento controlável a ser enviado para a planta ocorre da seguinte maneira: na etapa de *setup* do controlador é lido o valor de um registrador ADC do microcontrolador que está desconectado; o valor lido é utilizado como semente de uma função geradora de números aleatórios; a função aleatória é executada a cada ciclo da tarefa Escreve Coils para determinar por qual posição do vetor de eventos controláveis habilitados e desabilitados será iniciada a varredura até encontrar um evento que seja não desabilitado por todos os autômatos.

Após a seleção do evento controlável, este é convertido para seu endereço e valor de *coil* correspondentes e é enviado para a planta simulada e para as filas dos autômatos que possuem este evento em seu alfabeto. Ao enviar o evento para as filas, as tarefas dos autômatos passarão a estar no estado Pronto e, então, devido a sua prioridade ser superior a da tarefa Escreve Coils, o escalonador transicionará a Escreve Coils para o estado Pronto e colocará os autômatos em execução.

Quando não houver nenhum autômato em execução, a tarefa Escreve Coils voltará a executar. Se houver mais algum evento que pode ser enviado para a planta, o processo descrito é repetido. Caso contrário, a tarefa Escreve Coils chama a função *vTaskSuspended* colocando a si própria no estado Suspenso. A tarefa deixa de ocupar este estado, indo

para o estado Pronto novamente, quando a tarefa Lê Coils detecta um novo evento e explicitamente chama a função *vTasResume* com o *handle* da tarefa Escreve Coils.

4.1.3 Bibliotecas

Para implementação do controlador baseado na TCS foram desenvolvidas 3 bibliotecas em linguagem C. A biblioteca FlexFactTCC define macros referentes ao número total de eventos, número de eventos controláveis e número de autômatos (plantas locais e supervisórios). Além disso, esta biblioteca define o tipo de variável "flexfact_events" – um *enum* com todos os eventos possíveis de ocorrer na planta simulada – e os autômatos.

Os autômatos são definidos em forma de vetor conforme uma adaptação do método Memory Safe proposto por Lopes et al. (2011). Este vetor é composto por sub-vetores que representam cada estado do autômato. O primeiro elemento do sub-vetor indica o número total de transições a partir daquele estado (n). Após, o sub-vetor possui $2n$ posições que descrevem cada transição daquele estado; uma indicando o evento e a outra o estado resultante da transição. A Figura 8 exemplifica a implementação do autômato da Figura 14 (abordada posteriormente) no controlador. Os demais autômatos foram definidos analogamente.

Figura 8: *Descrição do autômato da Figura 14.*

```
const flexfact_events H1r_automato[] = {
    3,  pm1_bmp,0,  xs1_wpar,1, pm1_psn,2,  //estado 0
    3,  xs1_wplv,0, pm1_bmp,1,  pm1_psn,3,  //estado 1
    2,  pm1_bmp,0,  xs1_wpar,3,             //estado 2
    1,  xs1_wplv,2                          //estado 3
};
```

Fonte: Autor

Ademais, a biblioteca FlexFactTCC é responsável pela especificação das funções que convertem endereços e valores de *coils* para eventos e vice-versa. Sendo assim, esta biblioteca é exclusiva para controle da planta simulada neste trabalho e todas as características da planta e do controlador desenvolvido no DESTool são inteiramente cobertas por ela. Isso significa que caso deseja-se controlar uma planta com *layout* diferente, será necessário escrever outra biblioteca análoga à FlexFactTCC, mas não será necessário alterar outros parâmetros do código.

Outra biblioteca desenvolvida é a TCSlib, a qual implementa funcionalidades referentes a qualquer autômato, não dependendo da planta nem do controlador escolhidos. Esta biblioteca define a *struct* "automato" e, também, funções para cálculo do estado atual e eventos controláveis desabilitados e não desabilitados por um autômato. A TCSlib utiliza parâmetros da FlexFactTCC, como o tipo flexfact_events e as macros mencionadas.

A última biblioteca é a FreeRTOSTrace, que através das macros mencionadas na Seção 3.5.3 faz o rastreamento da execução das tarefas do controlador. Para saber quando cada tarefa está em execução utilizou-se a macro *vTaskSwitchedIn*, chamada pelo escalonador do FreeRTOS quando uma nova tarefa é transicionada para o estado Executando. A implementação da função para esta macro armazena em um vetor o *handle* da tarefa que entra em execução e, em outro vetor, o instante que isto ocorreu.

Conforme mencionado, as macros de rastreamento suspendem a execução do escalonador e, por isto, devem ser de execução rápida. Além disso, como o ATmega2560 possui apenas um núcleo de processamento, apenas com a macro *vTaskSwitchedIn* é possível

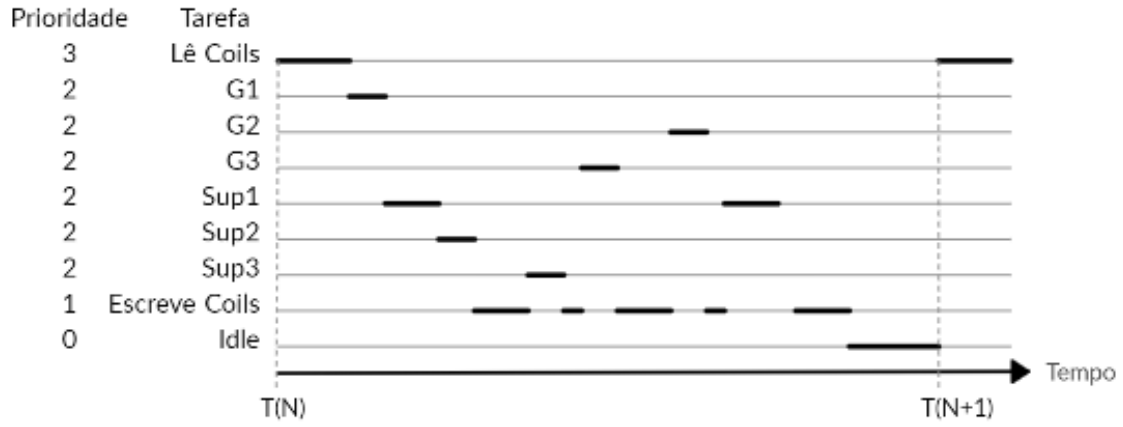
obter informações relativas a quando e por quanto tempo cada tarefa esteve em execução. Ainda, foi definida uma macro que possibilita ativar ou desativar o modo de rastreamento, o qual deve ser utilizado apenas na etapa de desenvolvimento do controlador.

4.1.4 Visão Geral do Código

O arquivo principal do código do microcontrolador inclui as bibliotecas necessárias, define as tarefas e realiza o *setup* do controlador. O *setup* do controlador corresponde à configuração e inicialização da interface Modbus TCP/IP sobre Ethernet; à criação das filas, semáforos e tarefas com os parâmetros definidos na biblioteca FlexFactTCC; e à inicialização da função geradora de números aleatórios e interface serial do microcontrolador. Após o *setup*, o escalonador do FreeRTOS é inciado e as tarefas entram em execução.

O diagrama da Figura 9 exemplifica a execução das tarefas de um controlador com 3 autômatos de plantas locais e 3 autômatos de supervisórios. Cada ciclo do escalonador inicia com a execução da tarefa Lê Coils. Esta tarefa envia os eventos não controláveis ocorridos para as filas dos autômatos. Na Figura 9 é ilustrada a ocorrência de um evento presente no alfabeto dos autômatos G_1, Sup_1 e Sup_2. Os demais autômatos apresentam sua fila vazia e encontram-se no estado Bloqueado. Durante a execução dos autômatos G_1, Sup_1 e Sup_2, os eventos em suas filas são processados e estes também passam para o estado Bloqueado. Então, a tarefa com maior prioridade no estado Executando é a Escreve Coils.

Figura 9: *Sequência de execução das tarefas do controlador.*



Fonte: Autor

A tarefa Escreve Coils seleciona um evento controlável não desabilitado, envia este para a planta simulada e para as filas dos autômatos que possuem este evento em seu alfabeto, no caso: os autômatos Sup_3 e G_3. Nota-se que ao receber um evento em sua fila, uma tarefa de autômato sai do estado Bloqueado e, devido a sua maior prioridade em relação a tarefa Escreve Coils, ela transiciona para o estado Executando.

Após o processamento do evento controlável, novamente a tarefa Escreve Coils passa para o estado Executando. Mais uma vez é selecionado um evento controlável, o qual é enviado para a planta e para as filas correspondentes. Na próxima execução da tarefa Escreve Coils verifica-se que não há nenhum evento controlável não desabilitado e esta tarefa transiciona para o estado Suspenso. Então, a tarefa Idle executa até passar o período definido da tarefa Lê Coils.

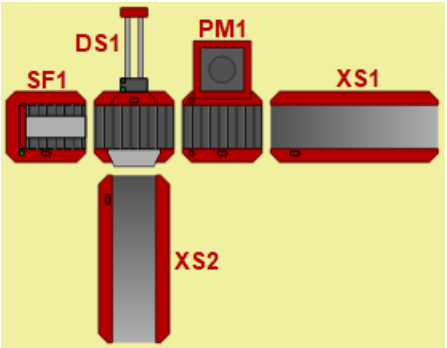
A determinação do período de leitura dos *coils* é limitada por dois fatores. Caso seja definido um ciclo com período curto o suficiente para que a tarefa Escreve Coils não seja

executada, o controlador não consegue atuar na planta. Ao mesmo tempo, um período demasiado grande pode ocasionar que os eventos não sejam identificados na ordem correta. Neste trabalho, o período de leitura foi escolhido empiricamente a partir de experimentos realizados onde pode-se verificar o comportamento do controlador.

4.2 Planta Simulada

O *layout* da planta controlada neste trabalho, composto por 4 diferentes tipos de componentes, é exibido na Figura 10. O componente mais a esquerda, denominado pelo FlexFact como *Stack Feeder* 1 (SF1), é por onde se dá a entrada das peças na linha de produção. Na sequência, o elemento *Distribuction System* 1 (DS1) encaminha a peça para a máquina à sua esquerda – *Process Machine* 1 (PM1) – ou para baixo – *Exit Slider* 2 (XS2). A PM1 realiza algum processamento na peça e, finalmente, a encaminha para o *Exit Slider* 1 (XS1).

Figura 10: *Layout da planta simulada no FlexFact.*



Fonte: Autor

Os componentes da planta simulada são equipados com sensores e atuadores. Todos possuem sensores que indicam a chegada e a saída de uma peça; as máquinas possuem motores para movimentar suas esteiras; e, ainda, cada componente possui sinais específicos abordados na modelagem dos autômatos (Seção 4.3).

4.3 Modelagem dos Autômatos

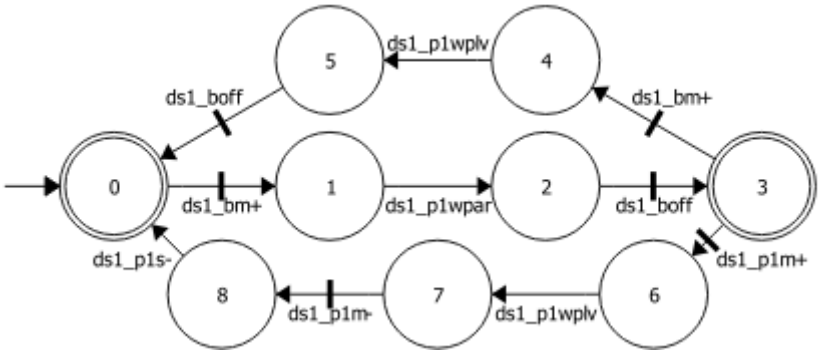
Os autômatos da planta simulada foram desenvolvidos no *software* DESTool. Inicialmente, para cada componente do *layout* da fábrica foi escrito um autômato descrevendo seu comportamento independente dos demais componentes. Após, foram modeladas as restrições impostas devido a interação dos componentes entre si. Por fim, a partir das plantas locais e das restrições, foram gerados os supervisórios ótimos cuja linguagem é a máxima linguagem controlável não bloqueante .

4.3.1 Modelagem dos Autômatos das Máquinas

Os autômatos da máquinas correspondem ao funcionamento de cada componente da planta simulada. O autômato modelado para o SF1 descreve que este está inicialmente desligado; quando seu sensor detecta a chegada de uma peça, o motor da esteira é ligado pra que a peça seja conduzida ao DS1; após a partida da peça, o SF1 volta para sua posição inicial e aguarda a próxima peça.

O autômato do DS1 pode ser visualizado na Figura 11. No estado inicial, o DS1 está com o motor de sua esteira desligada e seu pistão em repouso. Após, o motor da esteira entra em funcionamento (evento *ds1_bm+*), o que eventualmente causará que uma peça chegue ao DS1 e seja detectada por seu sensor, disparando o evento *ds1_p1wpar*. Então, o motor da esteira é desligado (*ds1_boff*) e o autômato está em seu estado 3. Neste estado existem dois eventos controláveis habilitados: *ds1_p1m+* – evento que liga o motor do pistão para empurrar a peça ao XS2 – e *ds1_bm+* – evento para ligar o motor da esteira, conduzindo a peça para a PM1.

Figura 11: *Autômato da máquina DS1.*



Fonte: Autor

No estado 3, caso ocorra o evento *ds1_bm+*, aguarda-se a peça deixar o DS1 (indicado pela ocorrência do evento *ds1_wplv*) para que o motor da esteira seja desligado e o autômato retorne ao seu estado inicial. Caso ocorra o evento *ds1_pm+*, após a peça deixar o DS1, é inciado o retorno do pistão para sua posição inicial com o evento controlável *ds1_pm-*. Quando o pistão atinge esta posição, o autômato está em seu estado inicial.

Na Figura 11, os estados 0 e 3 são marcados. Estes estados representam o DS1 com sua esteira desligada e o pistão em sua posição inicial. No estado 0, o DS1 está sem nenhuma peça e no estado 3, há uma peça posicionada neste elemento. Nos demais autômatos das máquinas, os estados marcados representam situações análogas a estas.

O estado inicial do autômato da máquina PM1 representa a esteira desligada e o processo em espera. Após, a esteira é ligada até que detecte-se a chegada de uma peça, quando a esteira é desligada e a máquina posiciona o processador sobre a peça. Nesta posição, a peça é processada e depois a máquina retorna o processador para a posição de espera. Finalmente, o motor da esteira é novamente ligado para enviar a peça para o XS1.

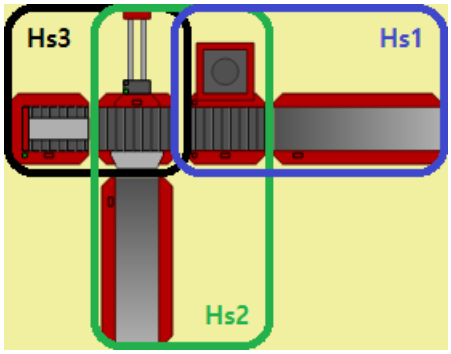
Os autômatos do XS1 e XS2 possuem apenas dois estados, ambos marcados. O estado inicial representa o componente com pelo menos um lugar disponível para armazenamento de alguma peça, enquanto o outro estado representa o componente sem capacidade de receber mais nenhuma peça. Estes autômatos não contém nenhum evento controlável em seu alfabeto e o usuário pode remover as peças que ocupam o XS1 e o XS2 manualmente.

4.3.2 Modelagem das Restrições

Os modelos das máquinas já explanados têm o objetivo de representar o comportamento individual de um componente independentemente dos demais. Para regular a interação de uma máquina com a outra foram modeladas restrições. Sendo assim, as restrições são responsáveis por limitar o comportamento de um componente em função do estado de outro(s) componente(s) para garantir o funcionamento da linha de produção.

As restrições desenvolvidas neste trabalho têm como objetivo identificar quando um componente está apto a receber uma nova peça e, então, utiliza-se esta informação para determinar se o componente anterior no *layout* pode ou não enviar esta peça adiante. A Figura 12 mostra os elementos envolvidos em cada restrição.

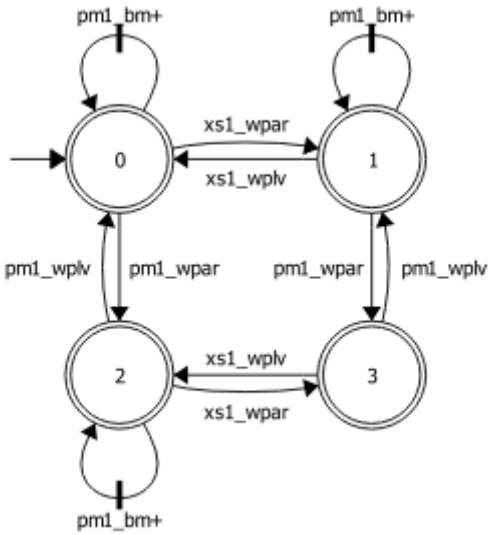
Figura 12: *Restrições sobre a planta simulada.*



Fonte: Autor

A restrição Hs1 modela quando o evento de ligar o motor da esteira da PM1 (*pm1_bm+*) deve estar desabilitado. Para tanto, são utilizados os eventos que indicam a chegada – *pm1_wpar* e *xs1_wpar* – e a partida de uma peça – *pm1_wplv* e *xs1_wplv* – nos elementos PM1 e XS1. A Figura 13 mostra o autômato da restrição Hs1, onde observa-se que o evento *pm1_bm+* está desabilitado apenas quando o XS1 está cheio e já há uma peça na PM1.

Figura 13: *Modelagem da restrição Hs1.*



Fonte: Autor

A segunda restrição, Hs2, corresponde a interação entre os elementos DS1, PM1 e XS2. O autômato de Hs2 estabelece quando a esteira de DS1 não pode ser ligada e, também, quando seu pistão não pode sair de sua posição inicial para empurrar a peça para XS2. Analogamente ao caso anterior, o motor da esteira não pode ser acionado quando já houver uma peça na PM1 e também em DS1. Além disso, o pistão não pode ser acionado quando XS2 já tiver atingido completamente sua capacidade de armazenamento.

Assim como na modelagem da planta local DS1, optou-se que a restrição Hs2 não definisse preferência ou critério de ordem de ocorrência dos eventos *ds1_bm+* e *ds1_p1m+*. Sendo assim, poderá haver situações em que ambos estes eventos estarão não desabilitados e um deles será escolhido pelo DESTool ou pela tarefa Escreve Coils para ser enviado para a planta. Conforme detalhado na Seção 4.1.2, a seleção do evento a ser enviado para a planta pelo controlador desenvolvido é dada através de uma função aleatória.

A terceira e última restrição imposta a planta é a Hs3, a qual é responsável por determinar a não habilitação do evento que liga o motor da esteira do componente SF1. Esta restrição determina que o SF1 apenas pode enviar uma peça para o DS1 caso este não contenha nenhuma peça e esteja com seu pistão retraído na posição inicial.

4.3.3 Geração dos Supervisórios

O *software* DESTool permite que sejam realizadas operações com os autômatos das plantas locais e das restrições e, a partir disto, foram obtidos os supervisórios modulares locais. A primeira operação realizada calcula a composição paralela entre os autômatos das máquinas envolvidos em uma restrição. Como cada máquina é assíncrona em relação as demais, esta operação não impõe restrições ao livre comportamento dos componentes. Esta etapa resultou nos autômatos:

$$G1 = PM1 \parallel XS1, \quad G2 = DS1 \parallel PM1 \parallel XS2 \quad e \quad G3 = SF1 \parallel DS1.$$

Após, foram aplicadas as restrições, gerando os autômatos

$$H1 = G1 \parallel Hs1, \quad H2 = G2 \parallel Hs2 \quad e \quad H3 = G3 \parallel Hs3,$$

que apresentam comportamento restrito. Nesta etapa foram realizadas as operações *isNonBlockin* e *isControllable* com os autômatos H1, H2 e H3 e todos obtiveram resultado positivo. Isto se deve ao processo de modelagem das restrições atentar a não desabilitar nenhum evento não controlável. Vale mencionar que devido a estas duas características, estes autômatos já correspondem aos supervisórios locais ótimos que apresentam linguagem minimamente restritiva controlável e não bloqueante.

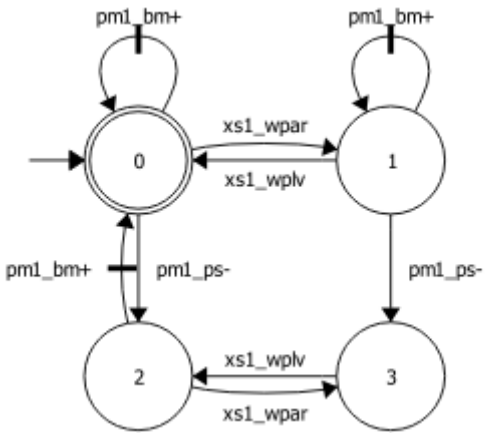
Os autômatos H1, H2 e H3, mesmo em suas realizações mínimas, apresentam elevado número de transições. Então, foi aplicada a operação *SupReduce*, que gera autômatos com o menor número possível de estados e transições que respeite a relação $G \parallel H = G \parallel H_r$. A Figura 14 exibe o autômato $H1_r$ obtido por este método. Os demais autômatos, $H2_r$ e $H3_r$, apesar de significativa redução de estados, ainda são maiores que os autômatos das restrições Hs2 e Hs3 impostas.

A estrutura do controlador executa os autômatos que modelam as plantas locais e os supervisórios independentemente. A ação de controle é o envio para a planta de um evento controlável não desabilitado simultaneamente por todos os autômatos, ou seja, é a intersecção destes autômatos. Utilizando a propriedade da TCS a seguir, tem-se que:

$$\begin{aligned} \mathscr{L}(H) \cap \mathscr{L}(G) &= \mathscr{L}(H \parallel G) \\ &= \mathscr{L}(Hs \parallel G \parallel G) \\ &= \mathscr{L}(Hs \parallel G) \\ &= \mathscr{L}(Hs) \cap L(G), \end{aligned}$$

o que mostra que pode-se utilizar Hs2 e Hs3 no controlador ao invés de H2 e H3 para simplificação da escrita dos autômatos mantendo as características de H2 e H3.

Figura 14: *Supervisório $H1_r$ gerado.*



Fonte: Autor

Finalmente, foi verificado que a ação conjunta dos supervisórios é não bloqueante a partir da averiguação da veracidade da equação:

$$H1 \parallel H2 \parallel H3 = Trim(H1 \parallel H2 \parallel H3).$$

Assim, os autômatos utilizados para implementação do controlador são: SF1, DS1, PM1, $H1_r$, Hs2 e Hs3. Os autômatos XS1 e XS2 são necessários para cálculo dos supervisórios, mas como possuem apenas eventos não controláveis, não é necessário incluí-los na implementação do controlador.

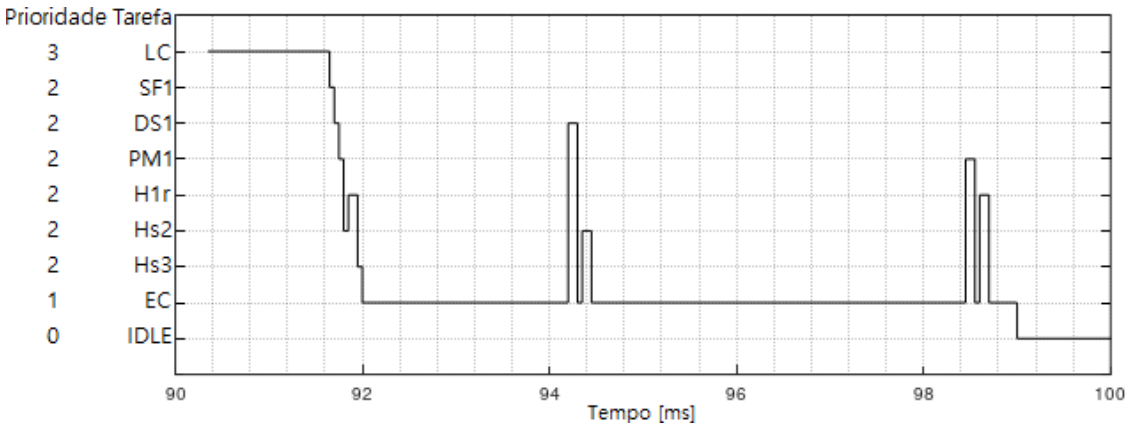
5 RESULTADOS E DISCUSSÕES

O teste do controlador se deu através de sua execução junto ao *layout* de fábrica da Figura 10 no FlexFact. Utilizando-se a funcionalidade de rastreamento do FreeRTOS abordada na Seção 3.5.3, foram obtidos dados a respeito da execução do escalonador do FreeRTOS, os quais são discutidos na Seção 5.1. Após, na Seção 5.2 é realizada uma análise a respeito do uso de memória do controlador desenvolvido. Então, a Seção 5.3 compara a ação de controle dos controladores no Arduino e no DESTool. Finalmente, na Seção 5.4, comentam-se os resultados obtidos da execução da estrutura de controle com outro *layout* de fábrica no FlexFact.

5.1 Rastreamento do Teste

O controlador implementado no microcontrolador Arduino Mega 2560 foi testado em conjunto com a planta simulada no FlexFact exibida na Figura 10. Ao iniciar a simulação, a planta encontrava-se sem nenhuma peça, os motores das esteiras estavam desligados, o pistão do DS1 estava em sua posição inicial, assim como o processador da PM1. A Figura 15 exibe a sequência inicial de execução do escalonador do FreeRTOS.

Figura 15: *Diagrama do escalonador 1.*



Fonte: Autor

Na Figura 15 é mostrado que o escalonador inicia o revezamento das tarefas cerca de 90,4 ms após inicialização do microcontrolador. Este tempo foi utilizado pelo microcontrolador para *boot* completo do sistema, carregamento do programa e *setup* do controlador, o qual inclui a criação das tarefas e das filas, estabelecimento da comunicação Modbus TCP/IP sobre Ethernet e inicialização das variáveis globais e da função aleatória.

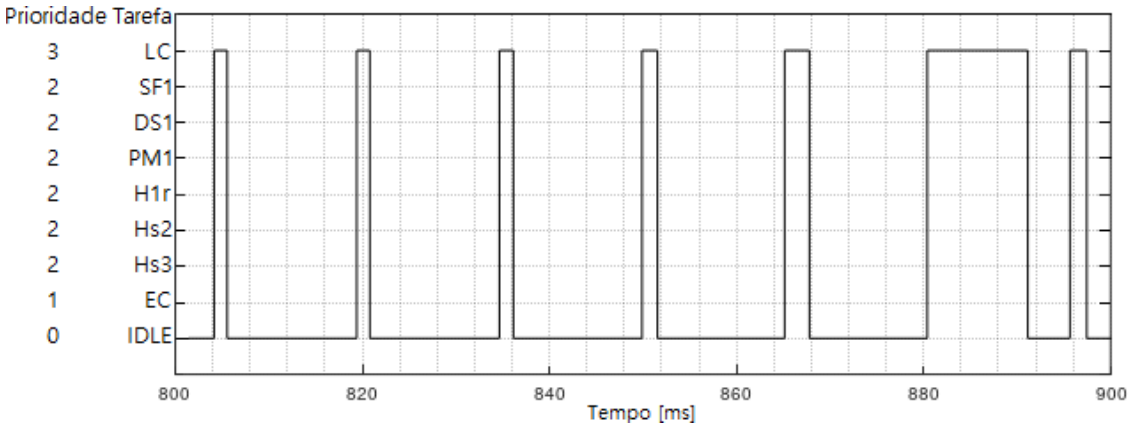
Após criação das tarefas, todas elas encontram-se em estado Pronto. Então, a tarefa Lê Coils (LC) é escolhida pelo escalonador para entrar em execução por possuir a prioridade com valor mais alto. Esta tarefa lê os endereços de *coils* da planta e transiciona para o estado Bloqueado até o período de leitura ser atingido. Assim, as tarefas dos autômatos – com prioridade igual a 2 – entram em execução uma por vez. Inicialmente, como não ocorreu ainda nenhum evento, as tarefas entram em bloqueio assim que identificam que suas filas estão vazias.

Com todas as demais tarefas no estado Bloqueado, a tarefa Escreve Coils (EC) identifica que o motor da esteira do DS1 pode ser ligado e o evento correspondente a esta ação é enviado primeiro para a planta simulada e, em seguida, para o autômato da planta local DS1 e para o autômato da restrição Hs2, onde seus estados atuais e a lista de eventos desabilitados e não desabilitados é atualizada. Após, a tarefa EC identifica outro evento não desabilitado por todos os autômatos, o evento correspondente a ligar o motor da PM1.

Analogamente ao caso do evento anterior, a tarefa EC envia o evento para a planta simulada e para os autômatos que o possuem em seu alfabeto. Após nova atualização dos eventos não desabilitados, a tarefa EC verifica que não há mais nenhuma ação de controle possível e transiciona para o estado Suspenso, fazendo com que a tarefa Idle seja a única habilitada a executar.

Na maioria dos ciclos – definidos como o intervalo entre o início de duas execuções da tarefa LC – não é identificada a ocorrência de nenhum evento na planta e o escalonador tem o comportamento da Figura 16. Devido a requisitos do FreeRTOS, tarefas periódicas tem seu período como múltiplo do *tick*, definido com base no *clock* do microcontrolador. O período deve ser suficientemente pequeno para que dois eventos subsequentes sejam identificados pelo controlador na correta ordem de ocorrência na planta; porém, quanto menor o valor do período, maior é o uso do processador. Empiricamente foi constatado que um período de 1 *tick* era ideal.

Figura 16: *Diagrama do escalonador 2.*

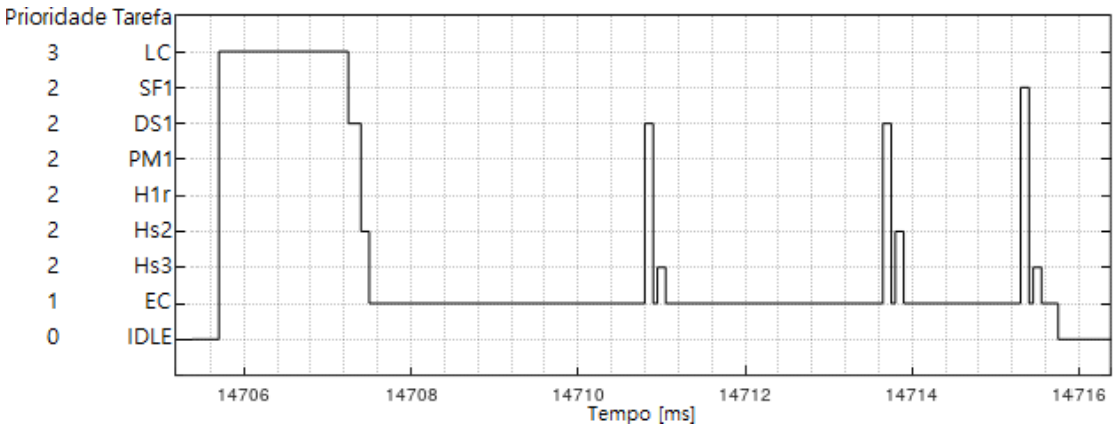


Fonte: Autor

A Figura 16 mostra que o intervalo entre o início da execução de cada ciclo da tarefa LC é constante e igual a 15,25 ms (medido com uma resolução de 0,05 ms). Percebe-se que o tempo de execução da tarefa LC em cada ciclo varia, provavelmente isso se deve ao sistema operacional Windows onde o simulador é executado e carece de uma maior investigação.

Terminada a inicialização, foi inserida uma peça no SF1, a qual foi encaminhada até o DS1. A Figura 17 mostra o ciclo do escalonador em que o evento de chegada da peça no

Figura 17: *Diagrama do escalonador 3.*



Fonte: Autor

DS1 é detectado. A tarefa LC envia este evento para os autômatos que o possuem em seu alfabeto: o da planta local DS1 e o da restrição Hs2. Após execução destas tarefa, a tarefa EC desliga o motor da esteira e este evento controlável é processado pelas tarefas DS1 e Hs3. Em seguida, há dois eventos controláveis não desabilitados (estado 3 da Figura 11), um deles responsável por enviar a peça do DS1 para a PM1 e o outro para o XS2.

O estado dos eventos controláveis é informado em um vetor e uma função aleatória determina a partir de qual evento (posição do vetor) será realizada a busca por um evento controlável não desabilitado. Sendo assim, para o teste exibido nesta seção, posicionou-se os eventos correspondentes a encaminhar a peça ou para a PM1 ou para o XS2 no vetor de maneira que tivessem a mesma probabilidade de serem selecionados. Por exemplo, se há 12 eventos controláveis, um deles está na primeira posição do vetor e o outro na sétima posição.

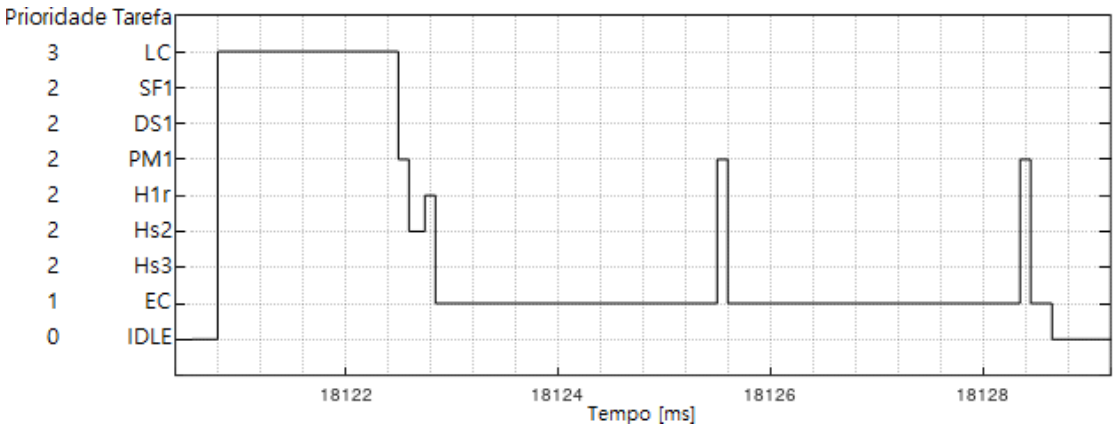
Na Figura 17 é selecionado o evento correspondente a enviar a peça para a PM1 e, na próxima atualização, como o DS1 está sendo desocupado o motor da esteira do SF1 é ligada para encaminhar a segunda peça para o DS1. O teste foi repetido diversas vezes e notou-se que aproximadamente metade das caixas foram enviadas para o XS2 e a outra metade para a PM1, conforme o desejado.

Também realizaram-se testes alterando a posição destes eventos no vetor de estado dos eventos controláveis e confirmou-se que a posição relativa entre eles no vetor afeta diretamente a probabilidade de um evento ser selecionado pelo controlador. Logo, pode-se utilizar a posição do evento para definir sua prioridade mesmo que a modelagem dos autômatos não estabeleça nenhum critério de prioridade entre os eventos.

A Figura 18 mostra o escalonador no ciclo em que a chegada da peça na PM1 é detectada. Nesta situação a esteira deste componente é parada e incia-se o deslocamento do processador para sua posição de processo. Nota-se que estes dois eventos controláveis dizem respeito apenas ao autômato da planta local da PM1 e que, portanto, nenhuma tarefa correspondente a outro autômato é envolvido nestas etapas.

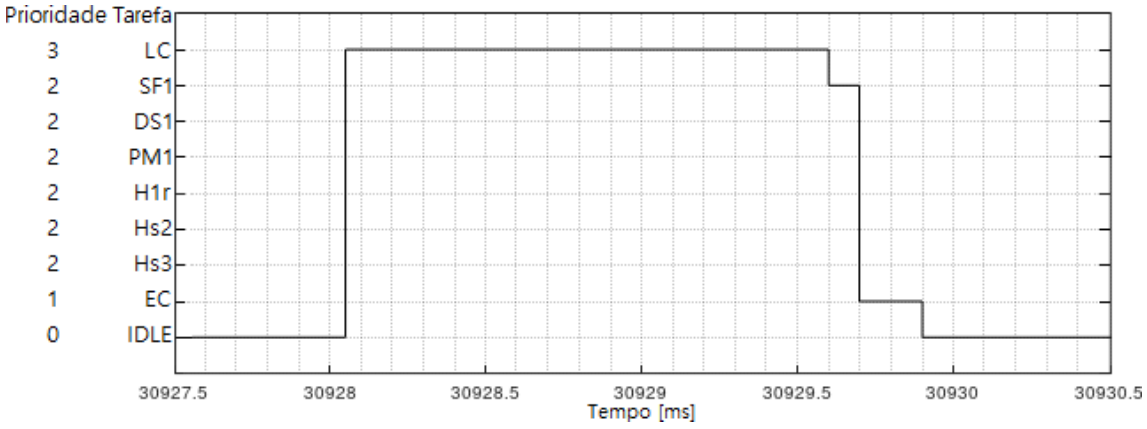
Em sequência no teste, a segunda peça foi encaminhada pelo DS1 para o XS2 e enquanto o pistão estava retornando para sua posição inicial, uma nova peça chegou no SF1. Esta situação é exibida na Figura 19, onde percebe-se que como o DS1 não está apto para receber nenhuma peça, o motor da esteira do SF1 não pode ser ligado e a tarefa EC não identifica nenhum evento controlável não desabilitado, fazendo com que neste ciclo, apesar de ser detectada a ocorrência de um evento na planta, não haja nenhuma ação do

Figura 18: *Diagrama do escalonador 4.*



Fonte: Autor

Figura 19: *Diagrama do escalonador 5.*



Fonte: Autor

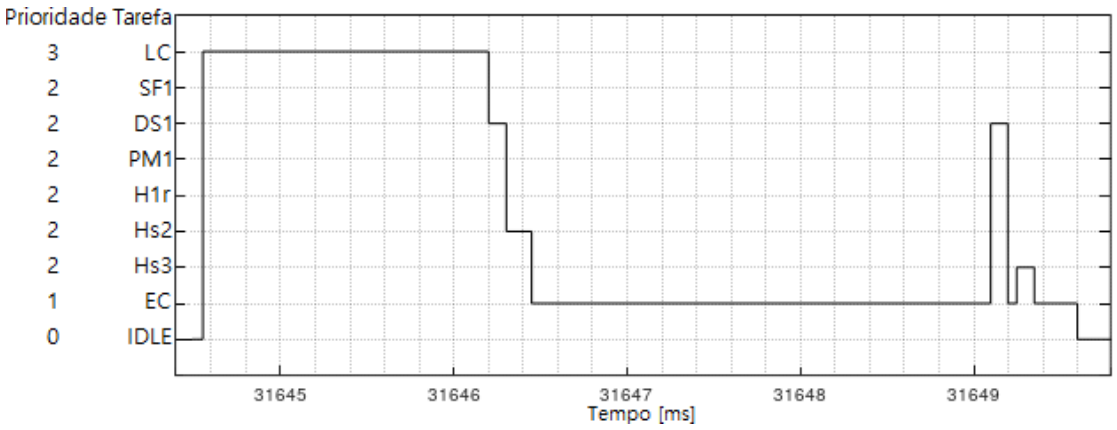
controlador.

Avançando ainda mais no teste, o XS1 e o XS2 atingiram sua capacidade máxima de armazenamento e a PM1 já estava com uma peça. A Figura 20 mostra o escalonador quando, nesta situação, chega uma nova peça no DS1. A tarefa EC verifica que, após o envio do evento de desligar a esteira do DS1, não há nenhum outro evento controlável não desabilitado. Assim, pode-se comparar este ciclo do escalonador com o ciclo exibido na Figura 17 – uma vez que ambos detectam a ocorrência do evento de chegada de uma peça no DS1 – para verificar que controladores gerados a partir da TCS consideram todo o histórico de eventos para definir sua ação de controle e não apenas o último evento ocorrido.

Os dados da execução do escalonador foram obtidos a partir das macros de rastreamento que são chamadas toda vez que ocorre troca de contexto da tarefa no estado Executando. A função implementada obtém o *handle* da tarefa, a partir desta informação, salva em um vetor o nome (ponteiro de `char`) da tarefa e, em outro vetor, salva o valor do registrador de um dos *Timers* do microcontrolador. Ao final do preenchimento do vetor, os dados salvos são exportados para o computador através da comunicação serial do microcontrolador. Vale mencionar que o tempo de execução da macro utilizada está embutido nos dados aqui apresentados.

As figuras com os diagramas do escalonador do FreeRTOS mostram que as tarefas LC

Figura 20: *Diagrama do escalonador 6.*



Fonte: Autor

e EC possuem tempo de execução significativamente alto em comparação com as tarefas dos autômatos. Um dos fatores responsáveis por este comportamento é a comunicação com a planta através do protocolo Modbus TCP/IP sobre Ethernet, principalmente em relação a tarefa LC que necessita realizar a leitura de todos os endereços de *coils* da planta. Sendo assim, para plantas maiores é esperado que o tempo de execução desta tarefa aumente proporcionalmente ao aumento do número de sinais da planta a ser controlada.

Outro fator que contribui para o elevado tempo de execução da tarefa EC é o tempo de processamento empregado na função aleatória. Apesar destas duas tarefas terem tempo de execução consideravelmente elevado, o escalonador ainda passa a maior parte do tempo executando a tarefa Idle. Sendo assim, não foram realizadas otimizações em relação ao tempos de execução das tarefas, mas pra outra planta isto pode ser necessário. Uma solução é implementar um método alternativo para seleção dos eventos controláveis e realizar a leitura apenas dos endereços de *coils* correspondentes a eventos não controláveis.

Ainda em relação as figuras dos diagramas do escalonador, nota-se que a estrutura de controle implementada permite que em um único ciclo diversos eventos controláveis sejam executados, o que propicia rapidez à ação de controle. Em uma planta real, pode ser necessário esperar um determinado intervalo de tempo para enviar dois eventos controláveis consecutivos para uma mesma máquina devido a efeitos não considerados na simulação, como a inércia dos componentes, por exemplo. Nestes casos, para utilização do controlador desenvolvido é necessário incluir este intervalo de tempo no modelo dos autômatos.

5.2 Uso de Memória do Microcontrolador

O microcontrolador fornece duas informações a respeito do uso de sua memória: uso da memória de programa e uso de memória dinâmica. A memória de programa é não volátil e é onde são armazenadas as instruções de código. O controlador completo desenvolvido utiliza 33348 *bytes* (1 *byte* = 8 *bits*), correspondendo a 13% da memória disponível para esta finalidade.

Foram realizados testes removendo-se instâncias da tarefa de autômatos – ou seja, diminuindo o número de autômatos do controlador – e, então, obteve-se que para cada instância de tarefa de autômato são necessários aproximadamente 40 *bytes* de memória de programa. Esta memória é ocupada pelas instruções de criação da tarefa e pela instância da função dos autômatos.

Na criação de uma tarefa é informado o espaço de memória destinado a armazenar os dados relativos a tarefa em questão. Este espaço é alocado dinamicamente durante a execução do controlador quando a instrução de criação da tarefa é executada. As tarefas dos autômatos, como são instâncias da mesma função-tarefa, ocupam o mesmo tamanho na memória, que equivale a aproximadamente 415 *bytes*, correspondendo a 5% da memória dinâmica.

Além de uma instância de tarefa, para cada autômato também é necessário reservar memória para o vetor que descreve o autômato e para sua fila, cujo espaço ocupado varia com o seu tamanho definido pelo usuário. O caso ideal do controlador consiste na execução das tarefas dos autômatos a cada novo evento, sendo assim, idealmente o tamanho da fila poderia ser unitário. Entretanto, assumindo que em um determinado ciclo possam ocorrer erros de comunicação com a planta, causando a execução prolongada da tarefa LC, uma fila com tamanho maior, como 5 elementos, é mais adequada.

A partir dos dados obtidos, percebe-se que o número de tarefas instanciadas influencia diretamente o uso da memória. Quanto menor o número de tarefas, menos memória é necessária no microcontrolador. Devido a propriedade $\mathscr{L}(A_1) \cap \mathscr{L}(A_2) = \mathscr{L}(A_1 \parallel A_2)$ dos autômatos e a ação de controle do controlador implementado corresponder a intersecção dos autômatos; a fim de reduzir o número de tarefas no controlador, pode-se realizar a operação de composição paralela para diminuir o número de autômatos.

Os ônus da redução do número de autômatos são a geração de autômatos possivelmente com números de estados e transições elevados e redução da modularidade do controlador. O microcontrolador utilizado possui apenas um núcleo de processamento, então a perda de modularidade não é significativa neste caso. Entretanto, em outros microcontroladores com maior número de núcleos de processamento, as tarefas dos autômatos – por serem independentes – podem ser executadas em paralelo. Nesta situação, a perda de modularidade é mais significativa.

5.3 Comparação com Execução no DESTool

Foram realizados ensaios da planta simulada com o controlador executando no DESTool para validação do controlador desenvolvido no microcontrolador Arduino Mega 2560 através de comparação entre estes. Com ambos os controladores, os testes foram realizados utilizando os mesmos autômatos, porém a comunicação planta-controlador se deu através do protocolo Modbus no caso do Arduino e do simplenet no caso do DESTool.

O teste realizado iniciou-se com o XS1 apenas com 1 espaço livre e o XS2, com 2. Após estabelecimento da comunicação com o controlador, adicionaram-se peças no SF1. A primeira peça foi encaminhada pelo DS1 para o XS2. Já segunda peça seguiu para o caminho contendo a PM1 e o XS1. A terceira, novamente foi encaminhada para o XS2, atingindo a capacidade máxima de armazenamento deste. A quarta peça seguiu para a PM1 e, após outra peça chegar no DS1, todos os elementos da planta simulada estavam completamente ocupados por peças.

A Tabela 1 exibe a sequência de eventos – controláveis e não controláveis – na planta simulada. Nesta tabela, em vermelho são destacados os eventos com diferente ordem de ocorrência entre os controladores. Nas duas primeiras colunas da Tabela 1, a diferença da ordem dos eventos corresponde a situações onde mais de um evento controlável encontrava-se não desabilitado e cada controlador escolheu uma ordem diferente para enviar estes eventos para a planta.

A diferença na ordem dos eventos no início da terceira coluna se deve à diferença na

Tabela 1: *Comparação da ordem de ocorrência dos eventos com controlador no DESTool e no microcontrolador.*

DESTool	Arduino	DESTool	Arduino	DESTool	Arduino
ds1_bm+	ds1_bm+	ds1_boff	ds1_boff	sf1_fdhome	pm1_ps-
pm1_bm+	pm1_bm+	sf1_fdon	ds1_bm+	sf1_fdoff	pm1_bm+
sf1_wpar	sf1_wpar	ds1_bm+	sf1_fdon	pm1_ps-	sf1_fdhome
sf1_fdon	sf1_fdon	sf1_wplv	sf1_wplv	sf1_wpar	sf1_fdoff
sf1_wplv	sf1_wplv	sf1_fdhome	sf1_fdhome	pm1_bm+	sf1_wpar
sf1_fdhome	sf1_fdhome	sf1_fdoff	sf1_fdoff	pm1_wplv	pm1_wplv
sf1_fdoff	sf1_fdoff	sf1_wpar	sf1_wpar	pm1_boff	pm1_boff
sf1_wpar	sf1_wpar	pm1_wpar	pm1_wpar	pm1_bm+	pm1_bm+
ds1_p1wpar	ds1_p1wpar	pm1_boff	pm1_boff	ds1_p1wpar	ds1_p1wpar
ds1_boff	ds1_boff	pm1_pm+	pm1_pm+	ds1_boff	ds1_boff
ds1_p1m+	ds1_p1m+	ds1_p1wpar	ds1_p1wpar	ds1_bm+	ds1_bm+
ds1_p1wplv	ds1_p1wplv	ds1_boff	ds1_boff	ds1_p1wplv	ds1_p1wplv
ds1_p1m-	ds1_p1m-	ds1_p1m+	ds1_p1m+	sf1_fdon	ds1_bm+
xs2_wpar	xs2_wpar	pm1_ps+	pm1_ps+	ds1_bm+	sf1_fdon
xs2_wplv	xs2_wplv	pm1_mrqu	pm1_mrqu	sf1_wplv	sf1_wplv
ds1_p1s-	ds1_p1s-	pm1_mon	pm1_mon	xs1_wpar	xs1_wpar
sf1_fdon	ds1_bm+	ds1_p1wplv	ds1_p1wplv	sf1_fdhome	sf1_fdhome
ds1_bm+	sf1_fdon	ds1_p1m-	ds1_p1m-	sf1_fdoff	sf1_fdoff
sf1_wplv	sf1_wplv	xs2_wpar	xs2_wpar	sf1_wpar	sf1_wpar
sf1_fdhome	sf1_fdhome	ds1_p1s-	ds1_p1s-	pm1_wpar	pm1_wpar
sf1_fdoff	sf1_fdoff	sf1_fdon	ds1_bm+	pm1_boff	pm1_boff
sf1_wpar	sf1_wpar	ds1_bm+	sf1_fdon	ds1_p1wpar	pm1_pm+
ds1_p1wpar	ds1_p1wpar	sf1_wplv	sf1_wplv	ds1_boff	ds1_p1wpar
ds1_boff	ds1_boff	pm1_mack	pm1_mack	pm1_pm+	ds1_boff
ds1_bm+	ds1_bm+	pm1_moff	pm1_moff	pm1_ps+	pm1_ps+
ds1_p1wplv	ds1_p1wplv	pm1_pm-	pm1_pm-	pm1_mrqu	pm1_mrqu

Fonte: Autor.

ordem de ocorrência dos eventos não controláveis *sf1_fdhome* e *pm1_ps-*. Observa-se que, por este motivo, os eventos controláveis *sf1_fdoff* e *pm1_bm+* também ocorrem em ordem diferente. Essa diferença se deve a algum fator da simulação e não invalida a performance do controlador implementado no microcontrolador.

Após, na terceira coluna, novamente repete-se a situação de diferença de escolha entre dois eventos controláveis não desabilitados das duas colunas anteriores. Ainda, no fim desta coluna, o evento *pm1_pm+* ocorre depois no controlador desenvolvido no DESTool. A sequência *pm1_wpar-pm1_boff-pm1_pm+* no Arduino é a analisada na Figura 18, onde percebe-se que *pm1_boff* e *pm1_pm+* ocorrem no mesmo ciclo do controlador. No DESTool, apenas um evento é enviado por ciclo de leitura dos sinais da planta, este pode ser o motivo do envio posterior do evento *pm1_pm+* pelo controlador no DESTool.

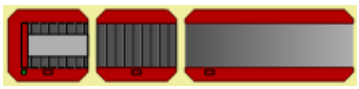
A comparação da sequência de eventos da Tabela 1 permite afirmar que a execução do controlador na estrutura desenvolvida no microcontrolador apresentou resultados equivalentes ao controlador desenvolvido no DESTool. Assim, valida-se o controlador desenvolvido neste trabalho.

5.4 Teste com Outra Planta

Finalmente, para exploração da característica de flexibilidade da estrutura de controle desenvolvida no microcontrolador, foi realizado um teste com uma planta com outro

layout no FlexFact. Para a nova planta, conforme Figura 21, a biblioteca FlexFactTCC foi substituída pela FlexFactTCC2. Neste novo arquivo, foram definidas as conversões de eventos para endereços de *coils* com seus valores correspondentes e vice-versa.

Figura 21: *Segundo layout de planta para teste do controlador.*



Fonte: Autor

Também, foram definidas as macros de: números de eventos controláveis e não controláveis; número de autômatos; número total de endereços de *coils* e outras particularidades. Finalmente, na biblioteca FlexFactTCC2 foram definidos os vetores que representam os autômatos e a lista de eventos presentes na planta.

Semelhante ao caso da planta anterior, as plantas locais descrevem o comportamento independente dos componentes da planta e as restrições foram modeladas para determinar quando o motor da esteira de um componente pode ser ligado. Para tanto, considerou-se se o componente subsequente está apto a receber uma nova peça e se o próprio componente possui espaço livre.

Executou-se o controlador com a nova planta simulada e observou-se que o comportamento da planta se deu conforme o esperado. Foi possível observar que os componentes executaram de acordo com os modelos dos autômatos das plantas locais e respeitando as restrições impostas. Sendo assim, foi validada a característica de flexibilidade da estrutura de controle.

6 CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho, foi implementado um supervisório em um microcontrolador com um escalonador de tempo real. Para tanto, propôs-se uma arquitetura para o sistema de controle em conjunto com uma estrutura de tarefas com diferentes prioridades. Os testes realizados com o controlador permitiram que o mesmo fosse validado através da análise de diagramas do escalonador e da comparação da ação de controle gerada com a de um controlador utilizando os mesmos autômatos no DESTool, *software* com credibilidade já atestada.

Ademais, a partir deste trabalho, pode-se dizer que algumas lacunas da literatura foram preenchidas em questão de utilização de escalonadores de tempo real em aplicações de supervisórios em sistemas microcontrolados.

Os testes realizados mostraram que a estrutura de controle desenvolvida possui flexibilidade, uma vez que para cada novo *layout* de fábrica basta alterar informações relativas aos novos eventos e autômatos em um único arquivo. Assim, a implementação do controlador torna-se rápida e simples e as características de execução do escalonador são mantidas.

A escrita dos autômatos em forma de vetor permite menor uso de memória e também maior facilidade de interpretação deste em comparação com a escrita do autômato em diagrama linguagem Ladder em CLPs. Sendo assim, é mais fácil que o desenvolvedor identifique erros de implementação do autômato a partir do método adotado neste trabalho.

A escolha de utilização do FreeRTOS mostrou-se vantajosa dado que sua licença é gratuita para pesquisa e para uso comercial e seu desempenho é bastante satisfatório. O uso das macros de rastreamento do FreeRTOS possibilitaram que fossem obtidas informações a respeito da execução das tarefas. Esta ferramenta foi muito útil tanto para desenvolvimento do controlador quanto para análise de execução deste com a planta simulada.

Para trabalhos futuros, sugere-se implementar o controlador desenvolvido em um microcontrolador com mais de um núcleo de processamento. Isso permitirá avaliar mais profundamente as vantagens e desvantagens da modularização do controlador em função do número de tarefas de autômatos.

Ainda a respeito da modularização, pode-se separar a tarefa de leitura de *coils* em mais de uma tarefa, onde cada uma destas é responsável pela leitura de um subconjunto de eventos da planta. Dessa maneira, pode ser utilizado um período de leitura diferente em cada tarefa de acordo com o intervalo mínimo de ocorrência de eventos subsequentes do subconjunto em questão. Assim, é reduzido o tempo total de leitura dos *coils*, resultando em menor tempo de processamento para esta finalidade.

Dado um cenário de uma fábrica complexa com diversos componentes, pode ser necessário descentralizar o controlador devido a limitações de memória ou capacidade de processamento paralelo de um único microcontrolador. Assim, outra sugestão é a descentralização do controlador, de maneira que o processamento seja realizado por uma rede de microcontroladores.

REFERÊNCIAS

ARDUINO. *What is Arduino?* 2018. Disponível em: <https://www.arduino.cc/en/Guide/Introduction>. Acesso em: 27 set. 2020.

BARRY, R. Mastering the FreeRTOS™ Real Time Kernel, A Hands-On Tutorial Guide. In: [s.l.]: Real Time Engineers Ltd., 2016a. Task Management, p. 44–100. Disponível em: <https://www.freertos.org/wp-content/uploads/2018/07/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf>. Acesso em: 26 set. 2020.

BARRY, R. Mastering the FreeRTOS™ Real Time Kernel, A Hands-On Tutorial Guide. In: [s.l.]: Real Time Engineers Ltd., 2016b. Queue Management, p. 101–146. Disponível em: <https://www.freertos.org/wp-content/uploads/2018/07/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf>. Acesso em: 28 set. 2020.

BARRY, R. Mastering the FreeRTOS™ Real Time Kernel, A Hands-On Tutorial Guide. In: [s.l.]: Real Time Engineers Ltd., 2016c. Resource Management, p. 233–264. Disponível em: <https://www.freertos.org/wp-content/uploads/2018/07/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf>. Acesso em: 28 set. 2020.

BARRY, R. Mastering the FreeRTOS™ Real Time Kernel, A Hands-On Tutorial Guide. In: [s.l.]: Real Time Engineers Ltd., 2016d. Developer Support, p. 328–354. Disponível em: <https://www.freertos.org/wp-content/uploads/2018/07/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf>. Acesso em: 28 set. 2020.

CARVALHO, A. S. *Estudo da utilização do Matlab/Simulink Aplicado ao Controle de sistemas a Eventos Discretos*. 2015. Monografia (TCC em Engenharia de Controle e Automação) – UFRGS.

CARVALHO, A. et al. Matlab Simulink Stateflow for Supervisory Control Simulation and Automated Code Generation. *XIV Simpósio Brasileiro de Automação Inteligente (SBAI)*, 2019.

CASSANDRAS, C. G.; LAFORTUNE, S. Introduction to Discrete Event Systems. In: [s.l.]: Springer Science+Business Media, LLC, 2008a. Languages and Automata.

CASSANDRAS, C. G.; LAFORTUNE, S. Introduction to Discrete Event Systems. In: [s.l.]: Springer Science+Business Media, LLC, 2008b. Supervisory Control.

FABIAN, M.; HELLGREN, A. PLC-based Implementation of Supervisory Control for Discrete Event Systems. *Proc. of the 37th IEEE Conference on Decision and Control*, 1998.

FREERTOS. *The FreeRTOS^{TM} Kernel*. Disponível em: <https://www.freertos.org/RTOS.html>. Acesso em: 28 set. 2020.

LOPES, Y. K. et al. Local Modular Supervisory Implementation in Microcontroller. *9th International Conference of Modeling, Optimization and Simulation - MOSIM'12*, 2012.

LOPES, Y. K. et al. Proposta de Implementação de Controle Supervisório em Microcontroladores. *X Simpósio Brasileiro de Automação Inteligente (SBAI)*, 2011.

MODBUS ORGANIZATION. *Modbus Application Protocol Specification*. V1.1b3. [S.l.], 2012. Disponível em: <https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf>. Acesso em: 27 set. 2020.

MOOR, T. *DESTool*. Lehrstuhl für Regelungstechnik der Friedrich-Alexander-Universität Erlangen-Nürnberg. Disponível em: <https://fgdes.tf.fau.de/destool/index.html>. Acesso em: 27 set. 2020.

MOOR, T. *Manufacturing System Simulator*. Lehrstuhl für Regelungstechnik der Friedrich-Alexander-Universität Erlangen-Nürnberg. Disponível em: <https://fgdes.tf.fau.de/flexfact.html>. Acesso em: 27 set. 2020.

PEREIRA, F. A. *Infraestrutura para implementação em linguagem C de controladores supervisórios para plantas de manufatura virtuais.* 2014. Monografia (TCC em Engenharia de Controle e Automação) – UFRGS.

QUEIROZ, M. H. D.; CURY, J. E. R. Controle Supervisório Modular de Sistemas de Manufatura. *Sba: Controle & Automação*, 2002.