Universidade Federal Rural do Semi-Árido
Programa de Pós-graduação em Sistemas de Comunicação e Automação

REUBER REGIS DE MELO

MÓDULOS FUNCIONAIS DE SOFTWARE PARA MICROCONTROLADORES ARM

MOSSORÓ-RN

2013

REUBER REGIS DE MELO

MÓDULOS FUNCIONAIS DE SOFTWARE PARA MICROCONTROLADORES ARM

Dissertação de mestrado acadêmico apresentada ao Programa de Pós-Graduação em Sistemas de Comunicação e Automação, como requisito para a obtenção do título de Mestre em Sistemas de Comunicação e Automação.

Orientador(a): Prof. Dr. Elmer Rolando Llanos Villarreal

Universidade Federal Rural do Semi-Árido - UFERSA

Co-orientador(a): Profª. Dra. Danielle Simone S. Casillo

Universidade Federal Rural do Semi-Árido - UFERSA

MOSSORÓ-RN

2013

Ficha catalográfica preparada pelo setor de classificação e catalogação da Biblioteca "Orlando Teixeira" da UFERSA

M517m Melo, Reuber Regis de.

Módulos funcionais de software para microcontroladores ARM. / Reuber Regis de Melo. -- Mossoró, 2013.
101f.: il.

Orientador: Prof. Dr. Elmer Rolando Llanos
Co-orientadora: Profª. Dra. Danielle Simone da Silva Casillo

Dissertação (Mestrado em Sistema de Comunicação e Automação. Área de concentração em Sistema de Comunicação e Automação) – Universidade Federal Rural do Semi-Árido. Pró-Reitoria de Pós-Graduação.

1. Automação. 2. Sistema de controle. 3.Sistemas embarcados. 4. Microcontroladores ARM. I. Título.

CDD: 004.16

Bibliotecária: Vanessa Christiane Alves de Souza
CRB-15/452

REUBER REGIS DE MELO

MÓDULOS FUNCIONAIS DE SOFTWARE PARA MICROCONTROLADORES ARM

Dissertação de mestrado acadêmico apresentada ao Programa de Pós-Graduação em Sistemas de Comunicação e Automação, como requisito para a obtenção do título de Mestre em Sistemas de Comunicação e Automação.

Aprovado em 30 de Agosto de 2013

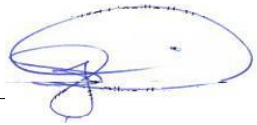
BANCA EXAMINADORA

Prof. Dr. Elmer Rolando Llanos Villarreal

Universidade Federal Rural do Semi-Árido - UFERSA

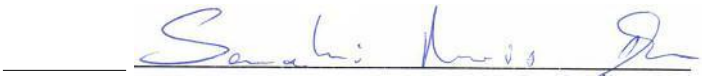
Prof[a]. Dra. Danielle Simone S. Casillo

Co–orientadora e Membro Interno - UFERSA



Prof. Dr. José Patrocínio da Silva

Membro Interno - UFRN



Prof. Dr. Samaherni Morais Dias

Membro Externo - UFRN

Aos meus pais, Bernardino e Valdiza.

Aos meus irmãos, Renato e Rodnei.

Resumo

As Unidades Microcontroladas (MCUs, do inglês *Microcontroller Units*) são componentes utilizados em uma variedade de dispositivos, que vão desde uso doméstico a uso industrial. Na automação industrial as MCUs estão presentes em equipamentos responsáveis pelo controle de motores, acionamento de válvulas, medições entre outras funções. Atualmente, as MCUs baseadas na tecnologia *Advanced RISC Machine* (ARM) estão se destacando devido a sua maior capacidade de desempenho, menor consumo de energia e baixo custo. Essas MCUs ARM® apresentam composição de software constituída por três elementos: um sistema operacional embarcado, *drivers* de dispositivos, e aplicações em gerais. Diante desta composição de software das MCUs ARM, este trabalho apresenta uma solução denominada de Módulos Funcionais para ARM (MoFA), com o objetivo de oferecer uma forma alternativa no desenvolvimento em automação e controle. Estes módulos são aplicações de software que podem ser desenvolvidas para diferentes sistemas operacionais embarcados em MCUs ARM, tais como Android™, distribuições de Linux embarcado e Windows CE. O MoFA possui basicamente quatro módulos: o módulo de algoritmo de controle para desenvolvimento e execução de rotinas de controle de um sistema de automação, o módulo de armazenamento de dados para guardar informações necessárias do processo da automação, o módulo de comunicação para acesso remoto a MCU ARM e o módulo de entradas e saídas digitais e analógicas para manipulação das portas I/O da MCU. Para validar o MoFA foram implementadas duas aplicações: um sistema supervisório para painéis fotovoltaicos e um sistema de controle PI de nível de líquido. Os resultados obtidos para cada aplicação são apresentados ao final deste trabalho. Para o desenvolvimento destas duas aplicações foi utilizado o microcontrolador ARM do Kit Tiny6410 da *Friendly* ARM. O trabalho foi realizado no laboratório do Grupo de Estudo e Desenvolvimento em Energia e Automação (GEDEA) da UFERSA.

Palavras-chave: Automação, Sistemas de controle, Sistemas embarcado, Microcontroladores ARM.

Abstract

The Microcontrollers Units (MCUs) are components used in a variety of devices, ranging from household to industrial use. In industrial automation MCUs are present in the equipment responsible for motor control, valve actuation, measurements and other functions. Currently, the MCUs based on Advanced RISC Machine (ARM) technology are highlighted due to its higher capacity performance, lower power consumption and low cost. These MCUs ARM feature software composition consist of three elements: an embedded operating system, device drivers and applications in general. Given this composition software of ARM MCUs, this paper presents a solution called Functional Modules for ARM (MoFA), with the aim of offering an alternative way in the development of automation and control. These modules are software applications that can be developed for different operating systems on embedded ARM MCUs, such as Android^{TM}, embedded Linux distributions and Windows CE. The MoFA has basically four modules: module of control algorithmic for developing and execution of control routines of an automation system, the data storage module to store information needed of the process automation, communication module for remote access to ARM MCU and the analog and digital inputs and outputs module for manipulation of I/O ports of MCU. To validate the MoFA were implemented two applications: a supervisory system for photovoltaic panels and a PI control system liquid level. The results for each application are presented at the end of this work. For the development of these two applications was used the ARM Microcontroller of Tiny6410 Kit of Friendly ARM. The study was conducted at the laboratory of GEDEA of UFERSA.

Keywords: Automation, Control systems, Embedded systems, ARM microcontrolles.

Agradecimentos

Quero agradecer primeiramente a Deus por propiciar a oportunidade em realizar este mestrado, e ter iluminado meu caminho durante todo esse percurso de mestrado. Não foi fácil.

Quero agradecer meu pai, Bernardino, por ter me dado toda força não somente neste mestrado, mas por todo tempo de estudante que passei na UFERSA. Meu pai serviu como exemplo de dignidade, humildade e simplicidade para minha vida. Suas mãos calejadas de um trabalho pesado me trazem os bons valores de um homem batalhador, de que apesar de todas as dificuldades da vida, o seu suor sempre representará a sua escolha para vence essas dificuldades, jamais passar por cima de ninguém para conseguir as coisas.

Quero agradecer minha mãe, Maria Valdiza, uma mulher de fibra, que em todo esse tempo foi a minha conselheira, a minha médica, a minha psicóloga. Este mestrado representa a luta incansável da minha mãe para oferecer a educação como um maior bem que se pode deixar para eu e meus irmãos. Obrigado mãe por ter cuidado de mim e por ter me encorajado a enfrentar este mestrado.

Quero agradecer aos meus irmãos, Renato e Rodnei. Eles dois são meus exemplos de pessoas vencedoras, e apesar de tudo dignas. Obrigado pelo apoio e por toda ajuda que precisei, serei grato sempre.

Quero agradecer aos meus orientadores, Elmer e Danielle, por ter me guiado e estado sempre do meu lado. Obrigado pelo tratamento humano que tiveram comigo, sou grato por isso.

Agradeço a CAPES pela bolsa concedida durante estes dois anos de mestrado.

Quero agradecer aos Professores Dr. José Patrocínio e Dr. Samaherni, pela participação em minha banca e as contribuições que nela propuseram.

Quero agradecer aos grandes amigos que fiz neste mestrado. Agradeço ao Egmidio pela sua expressiva ajuda em minha dissertação, sem a ajuda e força dele não teria saído. Agradeço a Egmidio também por ter cedido sua casa para os momentos de desopilação,

que incluíam as “generalizações”, enfim todas as “generalizações”. Agradeço ao Cláudio, este amigo sempre presente, por todas nossas conversas, ajuda, participações em festas. Agradeço a Felipe pelas as aventuras e motivação, mesmo de longe, ele dava aquela força pelo skype. Agradeço a todos meus amigos da nossa turma, Cassandra, Thomas, Clayton, Marcelo e Thiago, pelos estudos e trabalhos realizados juntos.

Agradeço aos meus amigos mais recentes do PPGSCA, Emmanuel, Ádller, Tárcisio, Adelson, por todas nossas conversas, ajuda no GEDEA e brincadeiras em nossas reuniões.

Agradeço a todos que compõem o prédio CITED, a Lívia pela paciência, ao pessoal da limpeza, Leiliane e Jucileide pelo os cafezinhos.

Agradeço a Camila pelo carinho e atenção comigo todo esse tempo, e é claro pelas refeições deliciosas que realizei em sua casa. Agradeço a toda família de Camila também pela força.

Agradeço a todos aqueles que não foram citados, mas que de certa forma contribuíram para eu realizar esta conquista.

"A marca do homem imaturo é que ele quer morrer nobremente por uma causa, enquanto a marca do homem maduro é querer viver modestamente por uma."

J. D. Salinger

Sumário

Lista de Tabelas

Lista de Figuras

Lista de abreviaturas e siglas

MCUs – *Microcontrollers Units*

ARM – *Advanced RISC Machine*

MoFA – Módulos Funcionais para ARM

GEDEA – Grupo de Estudo e Desenvolvimento em Energia e Automação

CLPs – Controladores Lógicos Programáveis

SCADA – *Supervisory Control and Data Acquisition*

CPU – *Central Process Unit*

UML – *Unified Modeling Language*

IHM – Interface Homem-Máquina

AIC – Automação Industrial e Controle

DCS – *Distributed Control System*

MIS – *Management Information System*

CEP – Controle Estatístico de Processos

PI – Proporcional Integral

PD – Proporcional Derivativo

PID – Proporcional Integral Derivativo

ANN – *Artificial Neural Network*

RISC – *Reduced Instruction Set Computer*

CISC – *Complex Instruction Set Computer*

ADC – *Analog-to-digital converter*

RS-232 – *Recommended Standard* 232

I^2C –*Inter-Integrated Circuit*

USB – *Universal Serial Bus*

PWM – *Pulse Width Modulation*

ACORN – *Acorn Computer Group*

RTOSs – *Real–Time Operating System*

LAN – *Local Area Network*

SOE – Sistema Operacional Embarcado

MAC – Módulo de Algoritmo de Controle

MAD – Módulo de Armazenamento de Dados

MC – Módulo de Comunicação

MED – Módulo de Entradas Digitais

MSD – Módulo de Saídas Digitais

MEA – Módulo de Entradas Analógicas

MSA – Módulos de Saídas Analógicas

RDIOC – Região de Dados I/O Compartilhados

ADT – *Android Developer Tools*

CITED – Centro Integrado de Inovação Tecnológica do Semiárido

1 Introdução

Os componentes de um sistema de automação evoluíram constantemente com o passam dos anos, desde os primeiros sistemas baseados em controle automático mecanizado (como as primeiras linhas de montagem do século XX) até os sistemas baseados nas tecnologias atuais como a microeletrônica (NEVES et al., 2007).

Em Balieiro (2008) é discutido que com o avanço da microeletrônica o chão de fábrica passou a ter mais integração com o ambiente corporativo, impulsionando fabricantes de Controladores Lógicos Programáveis (CLPs) a expandir as soluções para automação, não restringindo apenas a sistemas Supervisórios de Controle e Aquisição de Dados (SCADA, do inglês *Supervisory Control and Data Acquisition*) .

Nessa evolução da automação os equipamentos deixaram de ser mecanizados e passaram a ter componentes eletromecânicos e dispositivos microcontrolados. Os dispositivos de campo da automação industrial passaram a ter uma Unidade Central de Processamento (CPU, do inglês *Central Process Unit*), possibilitando não apenas medir e atuar no processo, mas também armazenar históricos, executar algoritmos de controle e se autoconfigurar.

Os sistemas embarcados passaram a ter um domínio muito amplo em termos de aplicabilidade, as aplicações industriais discutidas anteriormente é uma delas. Com isso, houve também melhorias nas metodologias e ferramentas para desenvolvimento (ZURAWSKI, 2009). Estas melhorias estão associadas ao uso de modelos de computação para desenvolvimento de sistemas embarcados, exemplo disso seriam as linguagens para modelagem de sistemas, como por exemplo, a *Unified Modeling Language* (UML). Sisbot (2011) trabalha esta questão da engenharia de software em sistemas de automação industrial.

De acordo com Wang e Tan (2006) o desenvolvimento de sistemas modernos de automação industrial tem seguido duas tendências. A primeira é a utilização de equipamentos industriais de controle e medição com as seguintes características: miniaturizados, portáteis e universalizados. A outra direção é o desenvolvimento para sistemas de controle distribuído.

Outra linha que precisa ser considerado são as tecnologias das MCUs, elas definem que nível de complexidade um sistema embarcado pode ter, devido a sua capacidade de processamento e custo. Nesse sentido MCUs ARM estão se destacando no mundo dos dispositivos digitais, e seu uso em dispositivos de automação não poderia ser diferente. Em Sudheer et al. (2013), Krivic et al. (2012), Franc e Šafarič (2010), Anish (2012), Yang et al. (2010) são mostrados algumas aplicações utilizando MCUs ARM, algumas delas para sistemas de controle, outras para Interface Homem-Máquina (IHM) . Ruimei e Mei (2010) e Li (2010) discutem o uso das MCUs ARM no desenvolvimento de sistemas de controle distribuído para automação industrial. Eles destacam que essas MCUs permitem agregar mais funcionalidades aos sistemas de automação.

1.1 Justificativa

As tecnologias usadas para desenvolver soluções em automação industrial são compostas por equipamentos, sejam sensores, atuadores, CLP, transdutores, entre outros, que em quase sua totalidade são equipamentos importados. Os altos custos desses equipamentos, por vezes inviabilizam o serviço de automação, já que em processos industriais qualquer investimento é ponto estratégico da gestão administrativa e em muitos casos os resultados em nível de investimentos financeiros em automação não justificam a implantação.

Essa questão custo-benefício é muito presente nas pequenas e médias empresas, que apesar de entender que modernizar suas máquinas representa em longo prazo algo fundamental para seu crescimento, o capital de investimento para essa modernização tem que ser viável para aquelas empresas. A automação tem um custo dividido em equipamentos e serviço de desenvolvimento, que são os programas, supervisórios, redes, entre outros. Em um sistema de automação industrial e controle, a programação dos controladores muitas vezes está associada a uma forma específica do fabricante. Com isso, é preciso que uma empresa se adeque ao tipo de controlador e sua plataforma de desenvolvimento, fazendo-se necessário o treinamento de funcionários para aquele tipo específico de CLP.

No mercado atual existem várias soluções proprietárias para sistemas de automação, controle e supervisão, mais conhecidos na área como sistemas SCADA. Podem-se citar algumas soluções e seus devidos fabricantes: WinCC – Siemens, RS View32 – Rockwell, Genesis32 – Iconics, Monitor Pro – Schneider Electric, Indusoft web Studio – Indusoft,

LabView – National Instruments, Elipse SCADA – Elipse (SISBOT, 2011). Todas essas soluções trazem consigo uma tecnologia diferente e proprietária que eleva muito o custo do produto. Para pequenas e médias automações pode não ser interessante.

Existem padrões e modelos para desenvolver automação industrial que estão associados às questões relacionadas a ambientes industriais, sinais elétricos e robustez; de modo que o mercado de equipamentos trabalha com padrões definidos para esse tipo particular de automação. É sabido que os atuais CLPs industriais são equipamentos que estão preparados para receber sensores de padrão industrial e fazer atuar válvulas, solenoides e comando elétricos, dentro de um padrão estabelecido e disseminado durante gerações, por fabricantes de equipamentos voltados a automação industrial.

As MCUs[1] foram originalmente desenvolvidas para atender controles em equipamentos das mais variadas aplicações, sejam dos tipos indústrias, equipamentos domésticos como TV, vídeo games, controle de elevadores, entre outros. No início da programação dos microcontroladores o assembly era a linguagem usual por ser nativa desses dispositivos compactos. Com o surgimento de compiladores para linguagens como C e outras linguagens de alto nível, a programação dos microcontroladores tornou mais amigável e proporcionando melhorias em softwares e ambientes de programação para microcontroladores.

É fato que atualmente os dispositivos de controle vêm utilizando MCUs com maior capacidade de processamento e menor tamanho, o que reflete imediatamente na miniaturização dos dispositivos e no aumento de suas funcionalidades. Um paradigma sobre isso é a nova tendência do mercado, que tem em seus produtos o uso de microcontroladores de baixo consumo de energia e alto poder de processamento. Esses microcontroladores, como por exemplo, os de 32 bits descritos em Pereira (2007), estão presentes na maioria dos equipamentos portáteis vendidos atualmente, incluindo telefones celulares, computadores de mão, MP3 players, videogames, entre outros.

Nesse sentido este trabalho propõe uma solução em software embarcado para microcontroladores ARM direcionados para sistemas de automação. Visto que esses microcontroladores estão em ascensão no mercado e podem oferecer maior gama de funcionalidades.

[1] Os termos MCU e microcontroladores possuem o mesmo significado neste trabalho.

1.2 Objetivo Geral

O objetivo geral do trabalho é a elaboração de uma solução em software embarcado que possibilite a utilização de um microcontrolador ARM em sistemas de automação industrial e controle.

1.2.1 Objetivos Específicos

1. Entender a estrutura de um sistema de sistemas de Automação Industrial e Controle (AIC);
2. Entender a estrutura de hardware e software de um microcontroladores ARM;
3. Propor uma solução em software embarcado para utilização de um microcontrolador ARM em sistemas AIC;
4. Implementar e avaliar aplicações baseadas na solução proposta.

1.3 Organização do Trabalho

No capítulo 2 é apresentada uma breve revisão dos conceitos de sistemas AIC e microcontroladores. É destacada a estrutura de um sistema AIC, e que quais as características funcionais dos níveis que compõem esta estrutura. Também neste capítulo é mostrado uma visão geral de um microcontrolador, e como exemplo de seu uso é descrito sobre o CLP e do que ele é constituído. Outro tema deste capítulo é os microcontroladores ARM e do que ele constitui em termos de hardware e software. Ao final do capítulo são destacados alguns trabalhos pesquisados e que serviram de base para este trabalho.

No capítulo 3 é apresentada a estrutura geral da solução MoFA desenvolvida neste trabalho. É mostrado de forma mais detalhada as funcionalidade do MoFA através da modelagem de casos de uso. Também neste capítulo é descrito sobre os matérias e métodos utilizados nas aplicações e experimentos desenvolvidos.

No capítulo 4 são apresentadas aplicações que foram desenvolvidas baseadas no MoFA. É explicado neste capítulo como o MoFA é utilizado em cada aplicação, detalhando quais casos de uso foram necessários para desenvolver cada sistema.

No capítulo 5 é feito as conclusões, considerações finais e possíveis trabalhos futuros.

2 Conceitos de Sistemas de Automação Industrial e Microcontroladores

A automação industrial pode ser definida como um conjunto de técnicas e de sistemas de produção, principalmente fabris, baseado em máquinas e dispositivos (sensores, atuadores, transdutores, entre outros). Atualmente esses elementos interagem de forma integrada, com capacidade de executar tarefas previamente definidas pelo homem e de controlar sequência de operações sem a intervenção humana. A automação industrial tem como objetivo a produtividade, qualidade e segurança na execução de processo (MARTINS, 2010).

Neste capítulo é descrito a estrutura fundamental de um sistema AIC, assim como os conceitos da arquitetura de um microcontrolador, focando nos microcontroladores ARM. Ao final é mostrado alguns trabalhos relacionados ao uso de microcontroladores ARM em sistemas AIC.

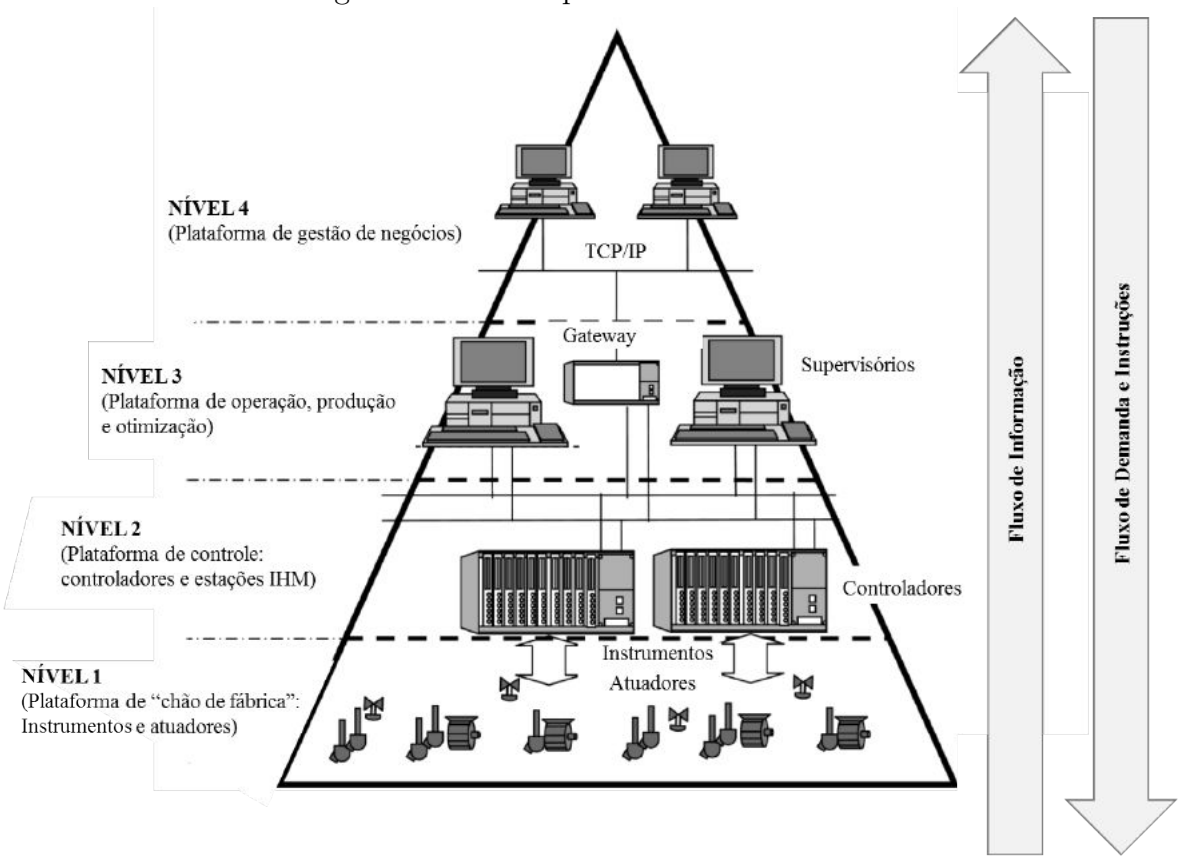
2.1 Sistemas de Automação Industrial e Controle

Sistemas AIC são responsáveis pelo controle em tempo real, monitoramento, relatórios e supervisão do processo de toda produção industrial (SISBOT, 2011). A estrutura fundamental de um sistema AIC é mostrada na Figura 2.1.

Os dispositivos de campo, como instrumentos de medição e atuadores, constituem o nível 1. Esses devem ser cuidadosamente posicionados em conformidade com os requisitos do processo. A quantidade exata dos dispositivos de campo é determinada pela amplitude e profundidade do plano de automação.

O nível 2 é a segunda plataforma onde o controle real, ações de monitoramento, bloqueio e alarme estão sendo realizados. Este nível é composto basicamente por equipamentos de controle, como CLPs, e estações de IHM, sendo estes equipamentos estruturados atualmente em Sistemas de Controle Distribuído (DCS, do inglês *Distributed Control System*).

Figura 2.1: Hierarquia de um sistema AIC



Fonte: (SISBOT, 2011).

O Nível 3 integra a plataforma de controle do nivel 2 para os sistemas de informação, tais como sistemas especialistas, Sistemas de Informações Gerênciais (MIS, do inglês *Management Information System*) e Controle Estatístico de Processos (CEP). Uma vez que essas tarefas requerem considerável poder computacional e tem que lidar com uma grande quantidade de dados, no nível 3 computadores são equipados com processadores rápidos e grande capacidade de armazenamento.

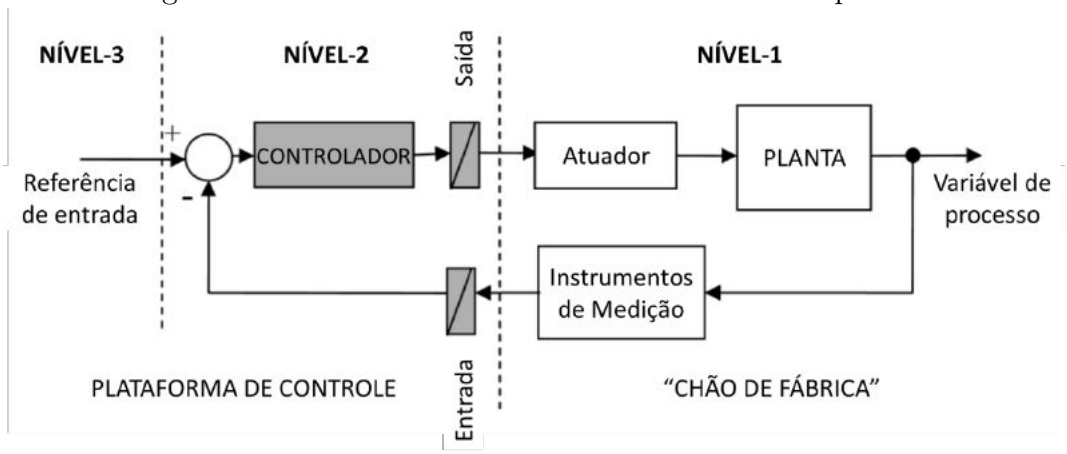
Os sistemas corporativos estão presentes no nível 4, geralmente referido como o "nível de negócios", onde contabilidade de custos, vendas, compras, gestão de ativos e tomada de decisão das tarefas do processo são executadas. Este nível não é diretamente uma parte da automação do processo de sistema de controle, mas está ligado ao mesmo, permitindo que os fluxos de informação e de instruções para os níveis subsequentes de acordo com as respectivas escalas de tempo real válidas.

Todos os níveis de um sistema AIC, excluindo parcialmente Nível 1, são plataformas digitais e sistemas de automação de processos são projetados para atender as necessida-

des específicas de engenheiros de controle, que, por definição, não são assumidos como especialistas em informática.

A Figura 2.2 ilustra um ciclo de controle em malha fechada e seus componentes. Os controladores comerciais vêm com compensadores padrão, como Proporcional Integral (PI), Proporcional Derivativo (PD), Proporcional Integral Derivativo (PID), e *lead-lag* (OGATA; YANG, 2011). Algoritmos de controle avançados, tais como PID *auto-tuning*, *Fuzzy Logic*, e Redes Neurais Artificiais (ANN, do inglês *Artificial Neural Network*), já estão sendo usados em alguns equipamentos de controle comerciais, e mais fabricantes devem estar no mercado em breve a oferecer algoritmos de controle avançados. A maioria dos fabricantes de CLP e DCS oferecem ferramentas com blocos funcionais, Ladder, ou códigos mnemônicos de programação para os engenheiros de controle, com o intuito de oferecer uma solução mais amigável para desenvolvimento em controle.

Figura 2.2: Controle em malha fechada e seus Componentes



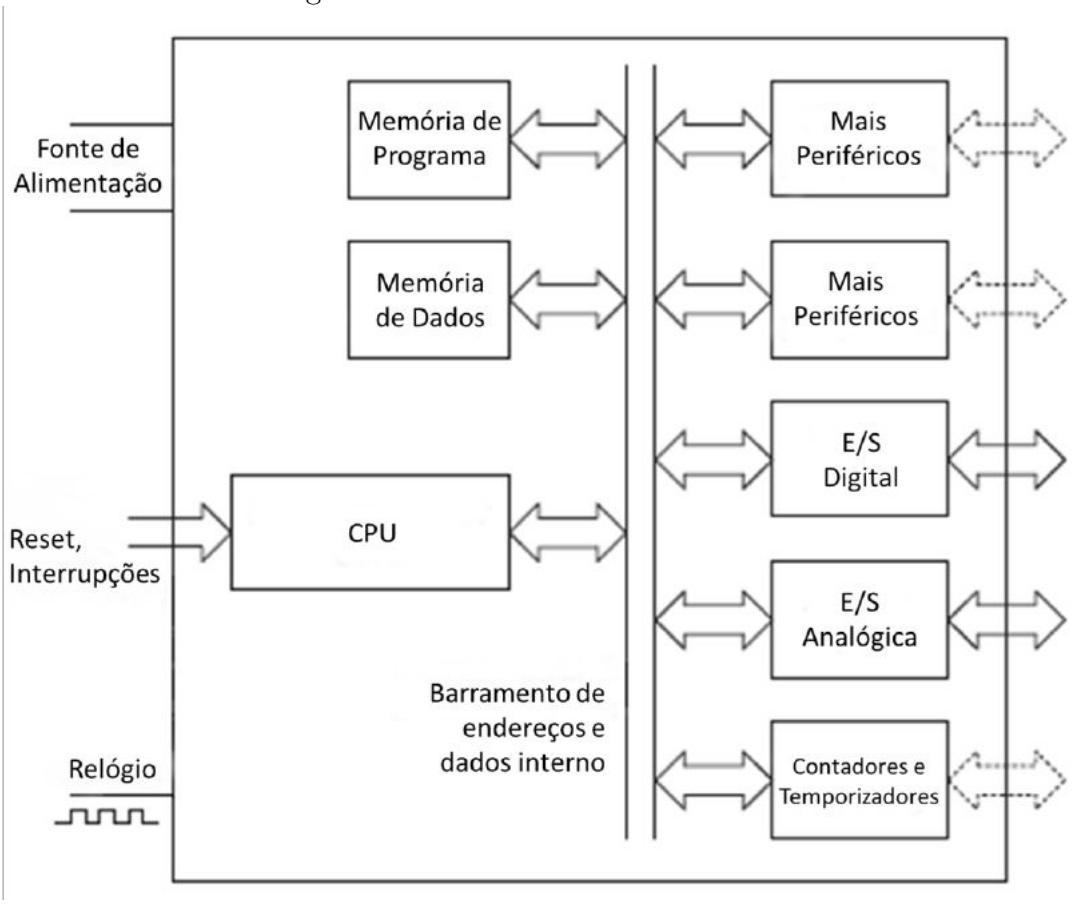
Fonte: (SISBOT, 2011).

2.2 Microcontroladores

Um microcontrolador pode ser considerado como um microcomputador construído sobre um único circuito integrado ou chip. Ele pode ser encontrado na indústria automotiva, sistemas de comunicação, instrumentação eletrônica, equipamentos hospitalares, aplicações e equipamentos industriais, eletrodomésticos, brinquedos, entre outros. O microcontrolador combina os recursos fundamentais disponíveis em um microcomputador, CPU, memória e recursos de entradas/saídas. A Figura 2.3 mostra o diagrama de bloco

de um microcontrolador genérico.

Figura 2.3: Microcontrolador Genérico



Fonte: (WILMSHURST, 2010).

Segundo Barrett e Pack (2006), a CPU do microcontrolador é um circuito sequencial complexo, cuja principal função é executar os programas que estão armazenados dentro da memória de programa. Logo, os programas são instruções responsáveis por realizar determinada tarefa. A arquitetura de hardware do microcontrolador está intimamente relacionada com o tipo do conjunto de instrução. Existem dois tipos desses conjuntos, o *Reduced Instruction Set Computer* (RISC) e *Complex Instruction Set Computer* (CISC), onde o primeiro tipo torna a estrutura da CPU menos complexa que a do segundo (VALDES-PEREZ; PALLAS-ARENY, 2009).

Os subsistemas dentro de um microcontrolador são conectados através de diferentes barramentos. Por exemplo, o barramento de endereço conecta a CPU e o sistema de memória. O barramento de dados é usado para trafegar dados paralelos entre diferentes subsistemas dentro do microcontrolador. A largura deste barramento de dados pode ser

de 4, 8, 16 ou 32 bits.

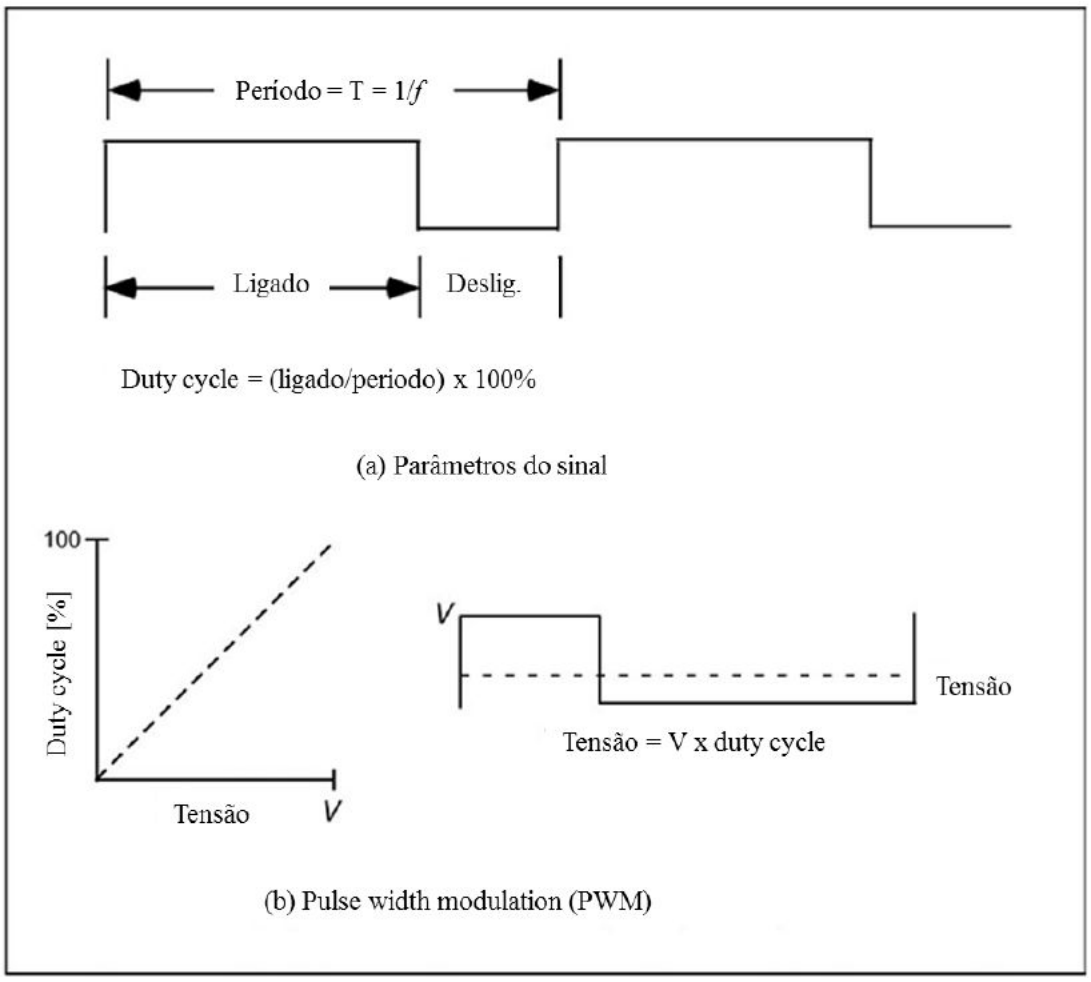
O microcontrolador possui uma série de portas que prover o seu acesso ao "mundo" externo, sendo organizadas como entradas e saídas da forma mostrada na Figura 2.3. As portas são usadas como entradas e saídas de sinais digitais, outras possuem funções alternativas, tais como Conversor Analógico Digital (ADC, do inglês *Analog-to-digital converter*), comunicação serial, e interface de rede. Algumas portas podem ser multiplexadas para resolver a insuficiência de pinos externos (BARRETT; PACK, 2006).

O subsistema ADC converte sinais analógicos em representações binárias adequadas para utilização pelo microcontrolador, tendo resolução de 8 a 10 bits mais comumente utilizados, de acordo com Barrett e Pack (2006). As portas de comunicação serial podem ser de diferentes tecnologias, como *Recommended Standard* 232 (RS-232), *Inter-Integrated Circuit* (I^2C), *Universal Serial Bus* (USB) e *Ethernet*.

A maioria dos microcontroladores também possui um subsistema para temporização. Algumas terminologias associadas a esse subsistema são:

- **Frequência:** A frequência do sinal é número de ciclos por segundo completado por um sinal repetitivo. A unidade de medida é expressa em Hz.
- **Período:** O período é o tempo em segundo necessário para completar um ciclo em sinal repetitivo.
- ***Duty Cycle**:* O ciclo de trabalho indica a porcentagem do tempo no qual o sinal esta no estado ligado. Ver Figura 2.4(a).
- ***Pulse Width Modulation*** (PWM) : Sinais de PWM são frequentemente utilizadas para controlar a velocidade do motor. O sinal digital PWM digital é convertido para um valor de tensão ideal para o funcionamento do motor. Na figura 2.4(b) é mostrado como a variação da tensões pode ser regulada de acordo com o ajuste do *duty cycle*.

Figura 2.4: Conceitos de temporização no microcontrolador



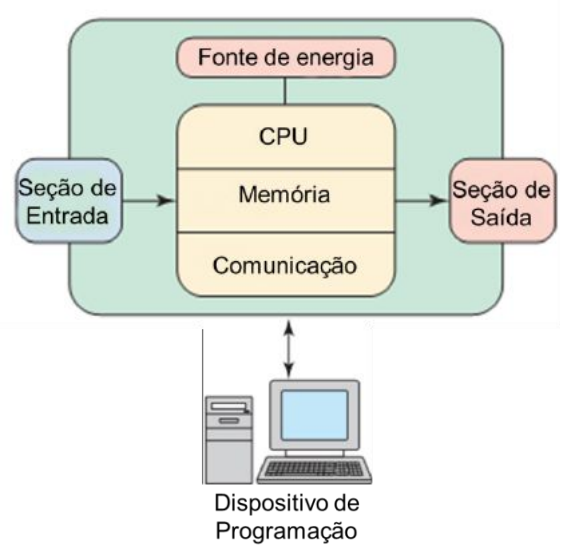
Fonte: (BARRETT; PACK, 2006).

2.2.1 Controlador Lógico Programável

Como foi abordado anteriormente, os microcontroladores podem ser encontrados em aplicações e equipamentos industriais. Um exemplo de equipamento é o CLP, localizado no nível 2 da estrutura de um sistema AIC. O CLP é uma forma especial de controlador baseado em microprocessador que utiliza a memória programável para armazenar instruções e para implementar funções tais como lógica, temporização, contagem e aritmética, podendo, portanto, controlar máquinas e processos. Esse equipamento é projetado para ser operado por pessoas, geralmente engenheiros, que não necessitem de um grande conhecimento sobre computadores e linguagens de programação. Sua programação é realizada utilizando métodos simples, muitas vezes em formas gráficas ou com sintaxes textuais simplórias (BOLTON, 2009). Um CLP típico pode ser dividido em partes, como ilustrado na Figura 2.5. Estas partes são: a CPU, seção de entrada e saída (I/O), memória,

comunicação e dispositivo de programação.

Figura 2.5: Partes típicas de um CLP.



Fonte: (PETRUZELLA, 2011).

A CPU é a unidade que contém um microprocessador, sendo este, responsável por controlar toda atividade do CLP. A programação do CLP é realizada em linguagens normatizadas na IEC 61131-3 (JOHN; TIEGELKAMP, 2010). O programa de um CLP é executado como parte de um processo repetitivo, iniciado pela leitura dos valores das entradas e em seguida a execução do programa da aplicação. Uma vez que a execução do programa tenha sido concluída, a CPU executa tarefas interna de diagnóstico e comunicação. Em seguida o estado de todas as saídas é atualizado, esse processo repetitivo

Figura 2.6: Ciclo de Scan do CLP.

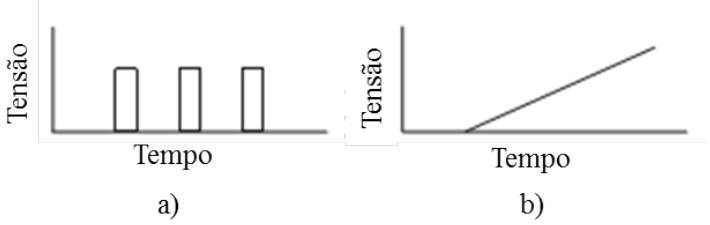
Fonte: (PETRUZELLA, 2011).

pode ser visualizado na Figura 2.6, e denominado de ciclo de *scan* do CLP. A fonte de energia é necessário para converter a voltagem alternada da rede em voltagem contínua, geralmente 5V, para alimentar o processador e os circuitos das seções de entrada e saída.

O dispositivo de programação é usado para desenvolver o programa de controle necessário, e em seguida, transferir para a unidade de memória do CLP. Na unidade de memória é onde o programa contendo as ações de controle exercido pelo microprocessador fica armazenado. Os dados referentes às entradas a serem processados também são armazenados na unidade de memória.

As seções de entrada e saída são onde o processador recebe e envia informações a dispositivos externos. Exemplos de dispositivos externos de entrada seriam os sensores, e dispositivos de saídas, os atuadores, como válvulas, motores, entre outros. Esses dispositivos podem ser classificados pelo tipo de sinal com que eles trabalham, podendo ser digital ou analógico mostrado na Figura 2.7.

Figura 2.7: Sinais: (a) Digital e (b) Analógico.



Fonte: (BOLTON, 2009).

O sinal digital é aquele que possui apenas dois estados, o ligado e desligado. Em dispositivos analógicos o é sinal proporcional ao tamanho da variável a ser monitorada, geralmente um sinal em tensão ou corrente. Por exemplo, um sensor de temperatura pode emitir uma tensão proporcional ao valor real de temperatura.

2.2.2 Microcontrolador ARM

Os microcontroladores ARM possuem CPUs de 32 bits que utilizam a filosofia RISC. Existem várias versões de microprocessadores ARM presentes naquelas CPUs, cada uma voltada para um nicho específico do mercado, podendo ter extensões que permitem executar instruções específicas, por exemplos *byte code* java (PEREIRA, 2007).

Essas importantes CPUs surgiram no início da década de 80, resultantes do projeto inovador de um bem-sucedido fabricante britânico de computadores chamado *Acorn Computer Group* (ACORN). A ACORN, assim como seus concorrentes na época (Atari, Apple, Commodore, Sinclair etc.), fabricava microcomputadores que utilizavam microprocessadores de 8 bits (no caso da Acorn, o 6502). Os novos projetos que a ACORN estudava necessitavam de um microprocessador rápido, de programação simples e construção barata. Após pesquisas e análises das opções disponíveis no mercado, decidiu iniciar eles mesmos o projeto de um microprocessador totalmente novo, empregando uma arquitetura de 32 bits (PEREIRA, 2007).

Os principais conceitos por trás da arquitetura ARM são a simplicidade, baixo custo, pequeno consumo e modularidade: as CPUs ARM foram projetados para serem simples, baratas e integradas aos mais diferentes periféricos, atendendo a diversos perfis de aplicação.

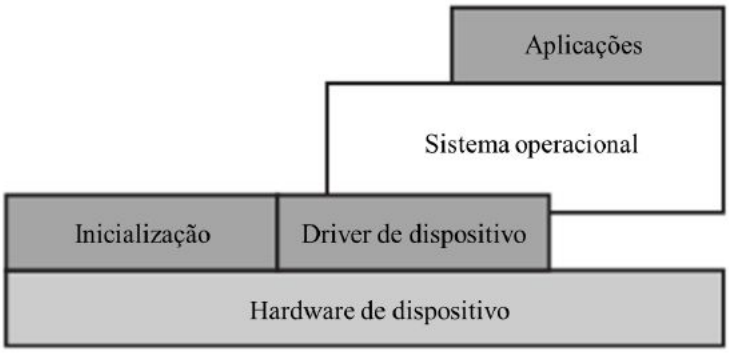
Um aspecto interessante é que, atualmente, a ARM Limited não fabrica *chips*, apenas licencia o uso de suas CPUs (a sua propriedade intelectual) por outros fabricantes de microcontroladores e microprocessadores (como Cirrus Logic, STMicroelectronics, Freescale, NXP (antiga Philips Semiconductors), Texas Instruments, Atmel, Conexant, Analog Devices, OKI, Nitendo, Fujitsu, Intel, VLSI, Samsung, Sharp etc.).

A estrutura de hardware de um microcontrolador ARM segue a algo semelhante mostrado na Figura 2.3. A CPU do microcontrolador é baseado em um microprocessador ARM, podendo mudar de um microcontrolador para outro porque diferentes versões de processadores estão disponíveis para atender as características operacionais desejadas (SLOSS et al., 2004). A versões da CPU ARM estão divididas em 8 tipos que dessa forma resultaram na criação de diversas famílias. Neste trabalho foi utilizado um microcontrolador com um processador ARM da família ARM 11 e versão 6 da CPU (ARM, 2012).

Um sistema embarcado baseado no microcontrolador ARM necessita de quatro componentes típicos de software, como mostrado na Figura 2.8. O software de inicialização é o primeiro código executado uma placa de microcontrolador ARM. Ele realiza configurações mínimas do hardware antes do sistema operacional assumir o controle deste hardware.

O sistema operacional fornece uma infraestrutura para controlar aplicativos e gerenciar os recursos de hardware, tais como memórias, interrupções, entre outros. Segundo

Figura 2.8: Camadas de abstração de software executados em hardware.



Fonte:(SLOSS et al., 2004).

Sloss et al. (2004), processadores ARM suporta mais de 50 sistemas operacionais. Eles podem ser divididos em duas categorias: Sistemas Operacionais de Tempo Real (RTOSs, do inglês *Real–Time Operating System*) e sistemas operacionais de usos gerais.

Os RTOSs estão voltados à execução de processos de controle em que o tempo de resposta a eventos é crítico. Como exemplo destaca-se: IAR–PowerPac da IAR, RL–ARM da Keil, OSEK, entre outros. Em sistemas operacionais de uso geral à execução de múltiplas aplicações em que os tempos de resposta a eventos não são críticos. Como exemplo destes sistemas destaca-se: Windows–CE®, *Embedded* Linux, μCLinux, PalmOS, Symbian, Android™, entre outros (PEREIRA, 2007). Neste trabalho foi utilizado o sistema Android™ para desenvolvimento das aplicações mostradas no Capítulo 4. O Android™não é um sistema operacional de tempo real, mas essa discussão já vem sendo explorada em alguns trabalhos como Maia et al. (2010), Perneel et al. (2012), Mongia e Madisetti (2010), Nagata e Yamaguchi (2012).

Os drivers como o terceiro componente de software mostrado na Figura 2.8, fornecem uma interface de software consistente para acesso ao hardware do microcontrolador e aos periféricos acoplados a ele. Em Venkateswaran (2008) e Love (2010) pode ser visto como é a estrutura de um driver para sistema operacional Linux.

As aplicações desenvolvidas sobre o sistema operacional é dedicada a realizar tarefas específicas de um processo. Já o sistema operacional controla todos processos que estão sendo executados, assim varias aplicações podem ser realizadas ao mesmo tempo.

Em particular é preciso ser destacado neste trabalho o crescimento que houve com o sistema operacional Linux. Desde o seu surgimento em 1991 o Linux vem ganhando

espaço em aplicações mais diversas, e em sistemas embarcados o seu uso tem ganhado muito apreso devido a sua robustez, flexibilidade, custo, comunidade de desenvolvimento e grande número de fornecedores. Isto implica na possibilidade de agregar muitas formas de realizar tarefas em sistemas embarcados (KARIM et al., 2008). Em MCUs ARM é muito comum encontrar sistemas embarcados baseados em Linux.

2.3 Trabalhos Relacionados

Para o desenvolvimento deste trabalho além do estudo sobre os conceitos já descritos até agora foi também realizado uma pesquisa sobre publicações de trabalhos que tivessem como abordagem o uso de microcontroladores ARM em sistemas AIC. Nas subseções 2.3.1, 2.3.2 e 2.3.3 a seguir foram descritos sobre esses trabalhos pesquisados focando apenas no que é relevante para este trabalho.

2.3.1 Projeto de um sistema embarcado de controle e aquisição de dados baseado em ARM para uma rede LAN industrial

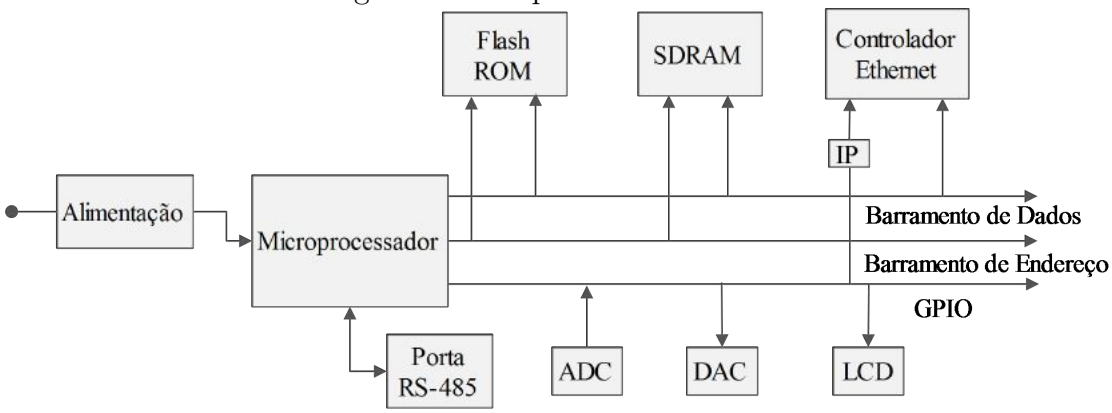
Um primeiro trabalho pesquisado e estudado foi de Li (2010), onde foi projetado um sistema embarcado baseado em um microcontrolador ARM com uma interface Ethernet, com o objetivo de realizar controle e aquisição de dados em uma rede LAN industrial. Li (2010) explica que desde que surgiram as MCUs sua utilização tem sido feito com sucesso, mas que MCUs convencionais de 8–16 bits possui certas deficiências, tais como baixa velocidade e pouca memória. Essas deficiências fazem com que tarefas de controle mais complexas não sejam realizadas nesses tipos de MCUs. No entanto o uso de MCUs baseados em microprocessadores de 32 bits está em grande ascensão, desempenhando um grande papel em sistemas de controle e aquisição de dados atualmente.

Outro conceito que Li (2010) discute e explora é o uso da tecnologia de redes de computadores, especialmente a Ethernet, na área da automação industrial. Segundo ele é uma solução que resolve alguns problemas dessa área, como dificuldade de comunicação entre equipamentos de fabricantes diferentes e custos elevados dos mesmos.

Uma visão geral da arquitetura do sistema proposto por Li (2010) pode ser visto na Figura 2.9, onde o microprocessador utilizado em sua MCU foi o S3C44BOX da família

ARM7 que é um produto da Samsung. Esse é um microprocessador RISC de 32bits com clock de 66MHz que oferece uma relação de custo-benefício em microcontroladores de alto desempenho para aplicações em gerais.

Figura 2.9: Arquitetura do sistema.

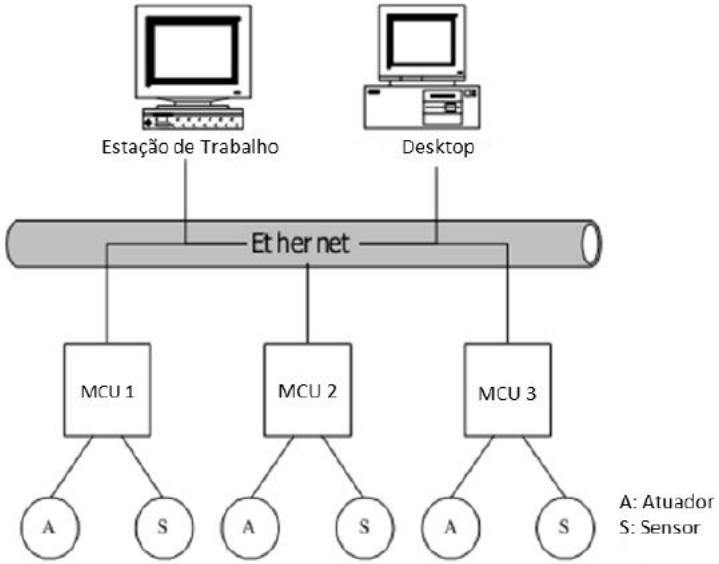


Fonte: (LI, 2010).

Na base desse sistema embarcado dois tipo de sistemas operacionais pode ser portado, o μC/OS-II e o μC-Linux. Segundo Li (2010) com sistemas operacionais embarcados tarefas de controle mais complexas podem ser realizadas no microcontrolador, podendo assim otimizar o desempenho daquelas tarefas. O sistema embarcado utiliza memórias flash para instalação do sistema operacional e aplicações do usuário assim como armazenamento de dados úteis das aplicações. A memória SDRAM é utilizada para execução do sistema embarcado como um todo, que inclui sistema operacional e aplicações do usuário. O conversor ADC é utilizado para aquisição de dados e o conversor DAC para sinal de saída de controle. Os conversores ADC, DAC e o LCD estão ligadas nas portas GPIOs do microcontrolador. Para comunicação com dispositivos externos como PCs ou estações de trabalho, pode ser utilizada a porta serial RS–485 ou a interface Ethernet.

A tecnologia de rede Ethernet é uma solução que vem sendo usada atualmente em "nível de dispositivo", onde ocorre a rápida transferência de dados entre as MCUs e sensores/atuadores. A Figura 2.10 mostra a estrutura do sistema de automação utilizando essa solução em uma rede local (LAN, do inglês *Local Area Network*) industrial. Tal estrutura faz com que o acesso aos controladores ocorram de forma direta, ou seja, as camadas superiores de supervisão e gestão da automação acessam diretamente a camada mais baixa da automação onde ficam os controladores, diferenciando da estrutura convencional de automação que foi visto na Seção 2.1.

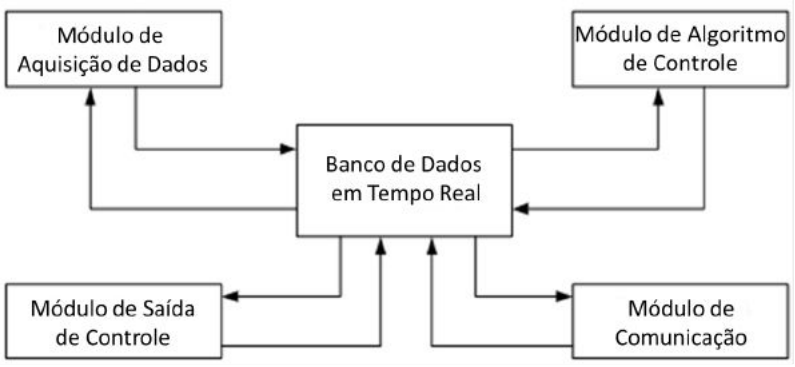
Figura 2.10: Sistema de controle e aquisição de dados utilizando uma Ethernet LAN.



Fonte: (LI, 2010).

Li (2010) dividi as funcionalidades do sistema embarcado em módulos e retrata o fluxo de dados entre esses módulos no diagrama de blocos mostrado na Figura 2.11. O banco de dados em tempo real é uma área comum de dados a todos os módulos, podendo estes acessar essa área diretamente. O módulo de aquisição de dados é responsável por obter dados via conversor ADC, e o módulo de saída de controle pode envia dados para o conversor DAC. O módulo de algoritmo de controle realiza a lógica do controle, por exemplo,

Figura 2.11: Fluxo de dados no sistema embarcado.



Fonte: (LI, 2010).

a lógica de um controlador PID, Fuzzy, entre outros. O módulo de comunicação permite o acesso de um sistema externo ao dispositivo, por exemplo, um sistema supervisório. Esse acesso pode ser realizado através da interface Ethernet ou porta serial RS–485.

Para finalizar Li (2010) conclui dizendo que seu projeto apresenta-se como um novo método em controle e aquisição de dados na forma distribuída. Ele explica que seu projeto permite desenvolvimento rápido e eficiente de uma aplicação para automação. Outro fator importante que ele discute é que tarefas que seriam realizadas em camadas superiores da automação tais como supervisão e gestão, podem ser realizadas no nível dos controladores, já que a plataforma de hardware e software desse sistema embarcado é bem mais robusta do que das MCUs convencionais. Esse fator reduz o grau de controle centralizado e enfatiza mais no conceito de controle e aquisição de dados na forma distribuída, aumentando a confiabilidade e reduzindo riscos no sistema de automação. Outro aspecto defendido no trabalho é que esse sistema embarcado é de baixo custo e compacto, sendo útil em aplicações industriais.

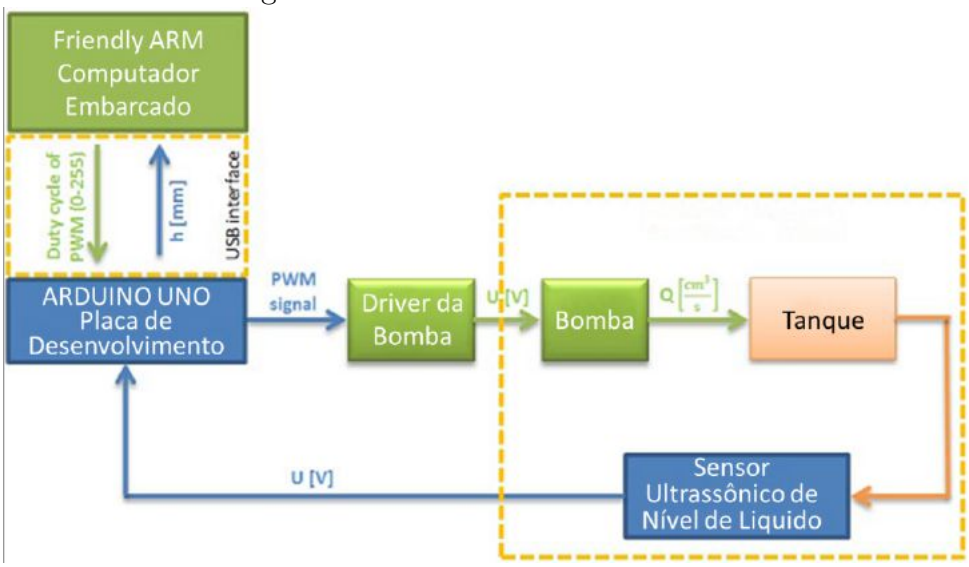
2.3.2 Projeto e implementação de um controlador Fuzzy em um computador embarcado para controle de nível de água

O artigo de Krivic et al. (2012) trata de uma das tarefas mais encontradas em processos industriais, o controle de nível de água, onde frequentemente um controlador PID é usado. No artigo foi implementado um controlador PID e Fuzzy em um computador embarcado, a *Friendly* ARM CoreWind (2012), para controle de uma planta didática de nível de tanque, a *Festo Didactic* DD 3100, e comparado as respostas desses controladores. A estrutura do sistema de controle pode ser visto na Figura 2.12.

O aplicativo desenvolvido na *Friendly* ARM permite ao usuário definir o nível desejado da água e selecionar o tipo de controlador (PID ou Fuzzy). O aplicativo exibe também o nível medido em milímetros. Uma placa Arduino UNO de desenvolvimento é usada para aquisição de dados e envio de sinal de controle para planta. A *Friendly* ARM faz comunicação com o Arduino via interface USB. Um sinal de PWM é usado para o controle da velocidade da bomba. A medição do nível de água é realizada utilizando um sensor ultrassônico e enviado o sinal produzido pelo sensor para porta analógica do Arduino.

Os controladores foram projetados pela primeira vez usando MATLAB e testados usando o modelo no Simulink com base no modelo matemático do tanque. Funções que representassem os controladores foram criadas em linguagem de programação C e embarcadas na *Friendly* ARM. Para tornar mais amigável o uso desses controladores uma

Figura 2.12: Estrutura do Sistema.



Fonte:(KRIVIC et al., 2012).

aplicação GUI simples foi projetada e criada usado o QT *Designer* para *Friendly* ARM Mini 2440. A aplicação GUI é mostrada na Figura 2.13.

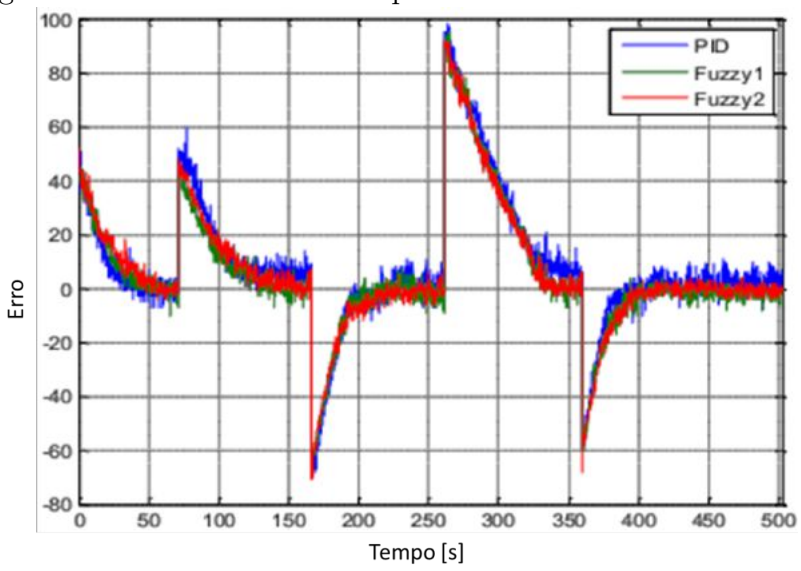
Figura 2.13: Aplicação GUI para Friendly ARM Mini 2440.

Fonte:(KRIVIC et al., 2012).

Todos os dados de medição são colocados em um arquivo e para demonstração da eficiência e comparação os controladores foram submetidos a mesma situação de controle. Na Figura 2.14 é ilustrada a resposta dos erros de controle para os controladores embarcados na *Friendly* ARM.

O interesse no estudo do trabalho de Krivic et al. (2012) não foi a modelagem dos controladores desenvolvidos, mas sim a aplicabilidade da estrutura do sistema utilizando a tecnologia ARM para implementação de tais controladores. O uso do ARM demonstra

Figura 2.14: Erros de controle para os diferentes controladores.



Fonte:(KRIVIC et al., 2012).

que é uma solução viável e aceitável para embarcar algoritmos de controle e ser facilmente integrado em processos industriais.

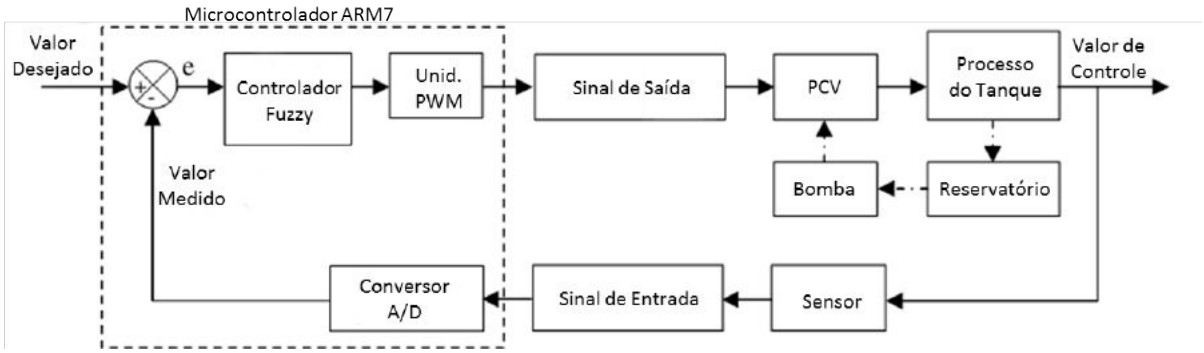
2.3.3 Controlador Fuzzy para sistema de controle de nível de líquido baseado no microcontrolador ARM7

No artigo de Sudheer et al. (2013) é relatado mais uma aplicação de um microcontrolador baseado em ARM para implementação de sistemas de controle. O controlador Fuzzy é implementado em linguagem C e embarcado no ARM para controlar o nível de um tanque a um valor desejado. O desempenho é comparado com um PID convencional. Sudheer et al. (2013) explica que esta abordagem utilizando um microcontrolador ARM reduz custos e espaço no sistema físico.

O diagrama de blocos do sistema de controle de nível de líquido é ilustrado na Figura 2.15. O tanque considerado para controle tem dimensões $100cmX20cmX20cm$ e a medida do nível é realizada por um sensor de pressão, o SXO5DN, onde o sinal gerando pelo sensor é enviado a porta analógica do microcontrolador. O fluxo de água da bomba para o tanque é controlado por uma válvula pneumática (PCV), que por sua vez é controlada pelo microcontrolador através de uma porta PWM.

O algoritmo completo para aquisição de dados, medição, visualização e controle de

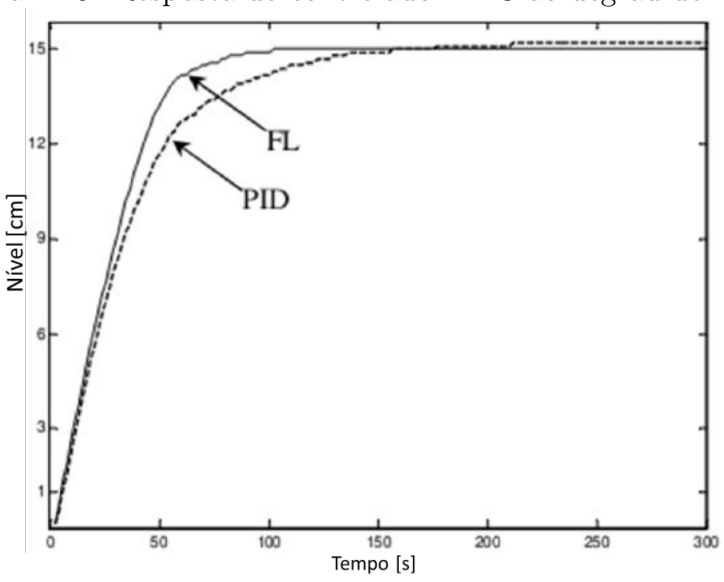
Figura 2.15: Diagrama de blocos do sistema de controle de nível.



Fonte:(SUDHEER et al., 2013).

nível de líquido é desenvolvido na linguagem C no ambiente de desenvolvimento μVision 4.0. A resposta do sistema a um degrau de 15cm é visualizado na Figura 2.16. Como pode ser visto o controlador Fuzzy tem uma melhor resposta obtendo menor tempo de subida e menor tempo de acomodação.

Figura 2.16: Resposta do controlador FLC ao degrau de 15cm.



Fonte:(SUDHEER et al., 2013).

Neste trabalho proposto Sudheer et al. (2013) o que se pode notar é que diferentemente do trabalho Krivic et al. (2012) visto anteriormente, os recursos de portas PWM e portas analógicas são usadas diretamente do próprio microcontrolador que controla a planta de nível de líquido.

2.4 Conclusão

Neste capítulo foi descrito uma breve revisão sobre conceitos de automação industrial e microcontroladores. É notável que os sistemas de automação tenham evoluído nos últimos tempos. Seus equipamentos tornaram-se cada vez mais modernos com o progresso da computação e comunicação. Um exemplo disso são as MCUs de 32–bits que utilizam como base a tecnologia ARM. Alguns trabalhos foram descritos e falam do uso dessas MCUs em sistemas de automação, e o potencial que elas podem oferecer em tais sistemas. No próximo capitulo é mostrado uma solução de software embarcado em um microcontrolador baseado na tecnologia ARM, com a finalidade de uso em sistemas de automação e controle.

3 Metodologia

Este capítulo discute e apresenta uma solução de software embarcado que ao decorrer deste trabalho é chamada MoFA. Essa solução busca alternativas no desenvolvimento de sistemas de automação e controle utilizando um microcontrolador baseado na tecnologia ARM. Em primeira estância é demostrado as funcionalidades que o MoFA apresenta utilizando a modelagem de casos de uso. Em seguida são descritos os materiais e métodos usados nas aplicações e experimentos desenvolvidos neste trabalho.

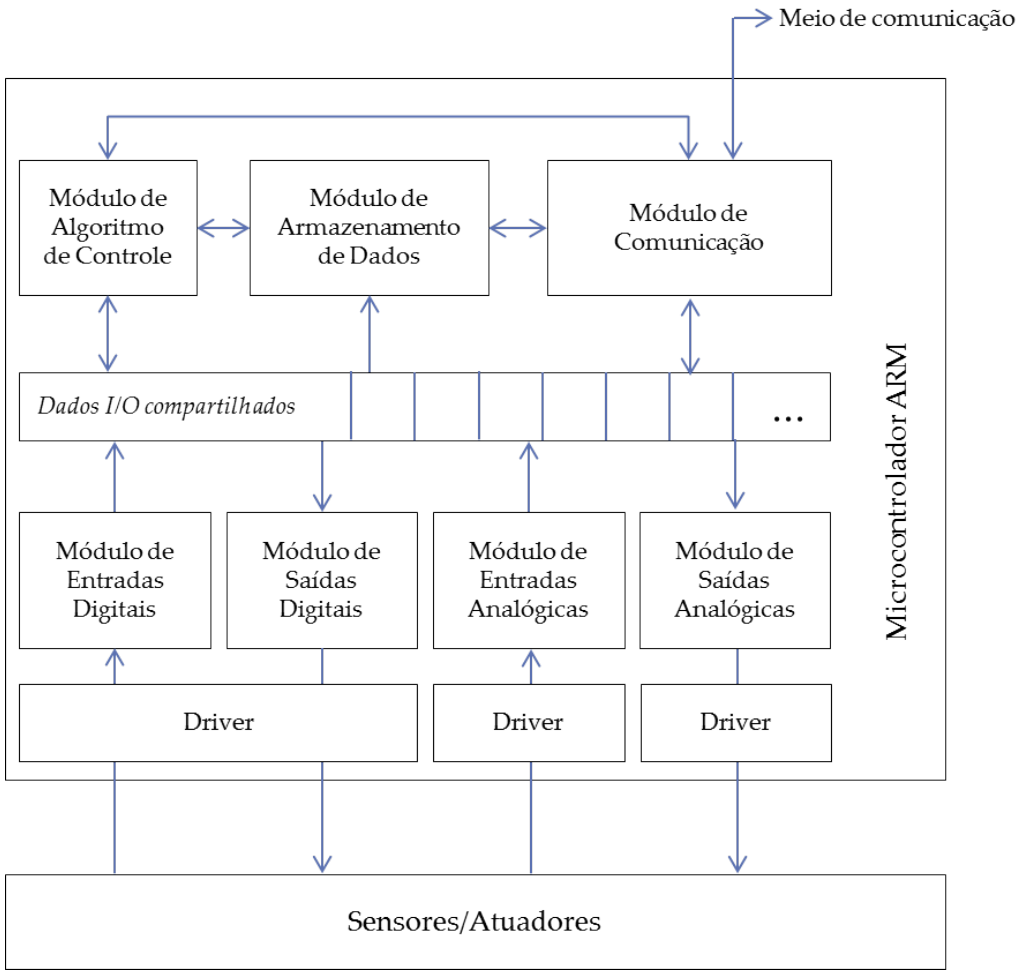
3.1 Modelo da solução MoFA

Com a motivação de uma solução alternativa em software para automação e controle, é apresentado neste capítulo o MoFA como sistema embarcado para MCUs ARM. A princípio o MoFA dividi as funcionalidades do microcontrolador ARM em aplicações de software que possam ser útil para automação e controle, deixando-as assim, em nível de operação. Foi determinado que o MoFA seja aplicações de software que trabalham em conjunto em uma MCU ARM.

A ideia é que a partir do modelo do MoFA sejam construídos aplicações em dois sentidos, uma no sentido de aplicações do tipo ferramentas, que facilite o uso desse microcontrolador por profissionais que não possuem conhecimentos mais avançado de programação, e o outro sentido é a construção de aplicações específicas para um sistema AIC, baseado em linguagens mais avançadas como C/C++ e Java. No capítulo 4 é mostrado duas aplicações específicas desenvolvida em Java e C para o sistema operacional Android™ embarcado em um microcontrolador ARM. O MoFA possui uma estrutura básica para automação, como entradas e saídas, algoritmos de controle e comunicação, como podem ser visto na Figura 3.1. Cada módulo do MoFA é descrito nas subseções a seguir.

É importante ressaltar que o modelo do sistema MoFA, tem o objetivo de desenvolvimento para diferentes sistemas operacionais embarcados em microcontroladores ARM, tais como Android™, Linux ou Windows CE. Não se pretende neste trabalho mostrar

Figura 3.1: Diagrama de blocos da estrutura Modular das Funcionalidades do Microcontrolador ARM.



Fonte: Autoria Própria.

qual o melhor sistema operacional, ou a melhor linguagem de programação, mas sim a essência do MoFA como alternativa em desenvolvimento de sistemas AIC utilizando a tecnologia ARM.

3.1.1 Tabela de atores

Para definir as funcionalidades do MoFA foi adotada a modelagem de casos de uso de acordo com Bezerra (2007) e Pressman (2011). O primeiro passo foi determinar os atores presentes na arquitetura geral do MoFA. Um ator segundo Bezerra (2007) é o elemento que interage com um sistema, normalmente pode ser pessoas, outros sistemas ou equipamentos. Os atores definidos para o MoFA estão identificados na Tabela 3.1.

Tabela 3.1: Atores definidos para o MoFA

Ator	Definição
Sistema Operacional Embarcado (SOE)	Gerencia os recursos da MCU ARM.
Módulo de Algoritmo de Controle (MAC)	Processa algoritmos de controle desenvolvido.
Módulo de Armazenamento de Dados (MAD)	Gerencia banco de dados ou sistema de arquivo.
Módulo de Comunicação (MC)	Gerencia portas de comunicação e cria conexões de rede.
Módulo de Entradas Digitais (MED)	Realiza a leitura das entradas digitais.
Módulo de Saídas Digitais (MSD)	Atualiza as portas de saídas digitais.
Módulo de Entradas Analógicas (MEA)	Realiza a leitura das entradas analógicas.
Módulos de Saídas Analógicas (MSA)	Atualiza as portas de saídas analógicas.
Usuário	Opera o MAC, MAD, MC, MED, MSD, MEA, MSA.

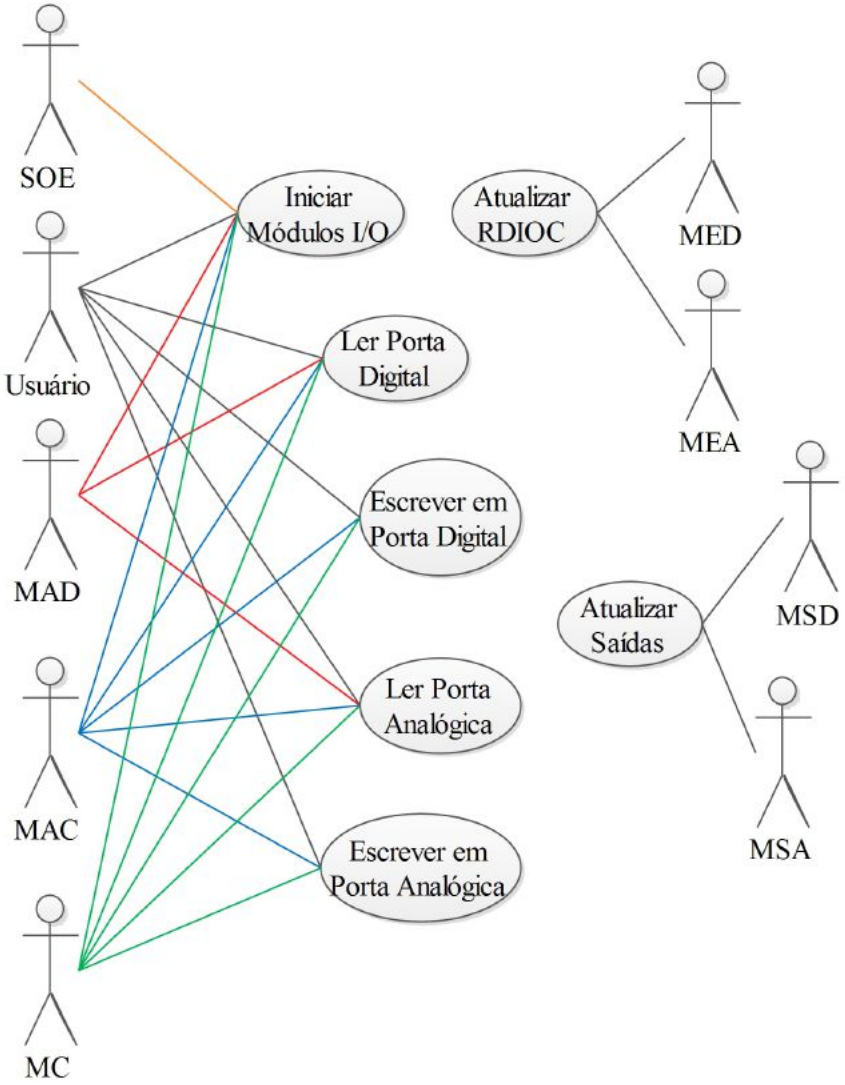
3.1.2 Módulos de entradas e saídas

Os módulos de entradas e saídas, identificados como MED, MSD, MEA e MSA, tem a função de abstrair o acesso às portas de entradas e saídas, tanto analógicas e digitais do microcontrolador ARM. Esse módulo utiliza uma Região de Dados I/O Compartilhados (RDIOC) possibilitando o acesso de outras aplicações a essa região. Existem alguns mecanismos que permitem a criação de um RDIOC, exemplo seria a *shared memory* (YAGHMOUR, 2013).

Neste trabalho, o objetivo é que as portas de entradas e saídas sejam representadas por variáveis na RDIOC. Assim, os módulos de algoritmo de controle, armazenamento de dados e comunicação podem acessar essas variáveis para ler, atualizar ou armazenar seus valores, sem precisar acessar diretamente as portas. Isso permite que o desenvolvimento dos outros módulos fique apenas no nível de aplicação, ou seja, não precisa manipular ou criar drivers novamente para acessar as portas do microcontrolador, apenas manipular as variáveis da RDIOC. Isso permite que o desenvolvimento dos outros módulos fique apenas no nível de aplicação, ou seja, não precisa manipular ou criar drivers novamente para acessar as portas do microcontrolador, apenas manipular as variáveis da RDIOC.

Para entender melhor as funcionalidades destes módulos a Figura 3.2 mostra os casos de uso inerentes a eles.

Figura 3.2: Funcionalidade dos módulos de entradas e saídas.



Fonte: Autoria Própria.

- **Caso de uso: Iniciar Módulos I/O**

Descrição:	Este caso de uso começa quando um dos atores primários inicia o MED, MEA, MSD e MSA. O objetivo deste caso de uso é abrir os drivers de acesso as portas I/O da MCU ARM.
Atores Primários:	Usuário, SOE, MAC, MAD, MC.
Pré-condições:	

Fluxo Principal	
Descrição:	**Iniciar módulos I/O**
Detalhe:	1 – Sistema verifica se existem drivers das portas I/O. 2 – Sistema verifica se drivers das portas I/O estão funcionando corretamente
Fluxos Alternativos	
Descrição:	
Detalhe:	
Fluxo de exceção:	
Descrição:	**Driver não existe ou não está respondendo**
Detalhe:	1 – O sistema informa que drivers não estão respondendo ou não existem.
Regras de negócio	

- **Caso de uso: Atualizar RDIOC**

Descrição:	Este caso de uso começa quando o MED e MEA são inicializados e entra em um ciclo infinito.
Atores Primário:	MED, MEA
Fluxo Principal	
Descrição:	
Detalhe:	1 – Os atores primários obtém através dos driver as leituras referente as entradas digitais. 2 – Os atores primários gravam os valores das leituras na RDIOC.
Fluxos Alternativos	
Descrição:	
Detalhe:	
Fluxo de exceção:	
Descrição:	**Erro ao gravar em região de dados compartilhados e no driver**

Detalhe:	1 – Sistema informa se houve erro na gravação em região de dados compartilhados. 2 – Sistema informa que houve falha no uso do driver das portas.
Regras de negócio	

- **Caso de uso: Atualizar Saídas**

Descrição:	Este caso de uso começa quando MSD e MSA são inicializados e entram em um ciclo infinito.
Atores Primários:	MSD e MSA.
Fluxo Principal	
Descrição:	**Atualizar saídas digitais e analógicas**
Detalhe:	1 – Atores primários buscam valores referentes as portas de saídas na RDIOC 2 – Atores primários enviam os valores buscados para as portas reais da MCU ARM através dos drivers.
Fluxos Alternativos	
Descrição:	
Detalhe:	
Fluxo de exceção:	
Descrição:	**Erro de leitura na região de dados compartilhados e no driver**
Detalhe:	1 – Sistema informa se houve erro de leitura na região de dados compartilhados. 2 – Sistema informa que houve falha no uso do driver das portas.
Regras de negócio	

- **Caso de uso: Ler Porta Digital**

Descrição:	Este caso de uso começa quando um dos atores primários solicita ler um valor referente a uma porta digital da MCU ARM. O objetivo desse caso de uso é buscar no RDIOC o valor da variável associado à porta que se quer ler.
Atores Primários:	Usuário, MAD, MAC, MC.
Pré-condições:	O MED está em pleno funcionamento.
Fluxo Principal	
Descrição:	**Ler porta digital.**
Detalhe:	1 – Ator primário define qual porta digital deseja acessar. 2 – Ator primário solicita ao sistema o valor da porta desejada. 3 – Sistema busca na RDIOC a região da porta associada à solicitação. 4 – Sistema retorna valor ao ator primário.
Fluxos Alternativos	
Descrição:	
Detalhe:	
Fluxo de exceção:	
Descrição:	**Erro de leitura na região de dados compartilhados.**
Detalhe:	1 – Sistema informa se houve erro de leitura na região de dados compartilhados.
Regras de negócio	

- **Caso de uso: Escrever em Porta Digital**

Descrição:	Este caso de uso começa quando um dos atores primários solicita escrever um valor em uma porta digital da MCU ARM.

Atores Primários:	Usuário, MAC, MC.
Pré-condições:	MSD está em pleno funcionamento.
Fluxo Principal	
Descrição:	**Escrever em porta digital.**
Detalhe:	1 – Ator primário define qual porta digital deseja acessar. 2 – Ator primário envia ao sistema valor a ser escrito em porta digital desejado. 3 – Sistema busca na RDIOC a região associada à porta desejada pelo ator primário. 4 – Sistema escreve na RDIOC valor desejado.
Fluxos Alternativos	
Descrição:	
Detalhe:	
Fluxo de exceção:	
Descrição:	**Erro de escrita na região de dados compartilhados.**
Detalhe:	1 – Sistema informa se houve erro de escrita na região de dados compartilhados.
Regras de negócio	

- **Caso de uso: Ler Porta Analógica**

Descrição:	Este caso de uso começa quando um dos atores primários solicita ler um valor referente a uma porta analógica da MCU ARM. O objetivo desse caso de uso é buscar no RDIOC o valor da variável associado à porta que se quer ler.
Atores Primários:	Usuário, MAD, MAC, MC.
Pré-condições:	MEA está em pleno funcionamento.
Fluxo Principal	

Descrição:	**Ler porta analógica.**
Detalhe:	1 – Ator primário define qual porta analógica deseja acessar. 2 – Ator primário solicita ao sistema o valor da porta desejada. 3 – Sistema busca na RDIOC a região da porta associada à solicitação. 4 – Sistema retorna valor ao ator primário.
Fluxos Alternativos	
Descrição:	
Detalhe:	
Fluxo de exceção:	
Descrição:	**Erro de leitura na região de dados compartilhados.**
Detalhe:	1 – Sistema informa se houve erro de leitura na região de dados compartilhados.
Regras de negócio	

- **Caso de uso: Escrever em Porta Analógica**

Descrição:	Este caso de uso começa quando um dos atores primários solicita escrever um valor em uma porta analógica da MCU ARM.
Atores Primários:	Usuário, MAC, MC
Pré-condições:	MSA está em pleno funcionamento.
Fluxo Principal	
Descrição:	**Escrever em porta analógica.**

Detalhe:	1 – Ator primário define qual porta analógica deseja acessar. 2 – Ator primário envia ao sistema valor a ser escrito em porta analógica desejada. 3 – Sistema busca na RDIOC a região associada à porta desejada pelo ator primário. 4 – Sistema escreve na RDIOC valor desejado.
Fluxos Alternativos	
Descrição:	
Detalhe:	
Fluxo de exceção:	
Descrição:	**Erro de leitura na região de dados compartilhados.**
Detalhe:	1 – Sistema informa se houve erro de leitura na região de dados compartilhados.
Regras de negócio	

Os módulos de entradas e saídas digitais acessam as portas digitais do microcontrolador. O módulo de entrada analógica acessa a porta ADC. O módulo de saída analógica acessa portas do tipo PWM. Como foi visto na Seção 2.2.2 quatro componentes de software são necessários para utilização de um sistema embarcado em microcontrolador ARM. Um desses componentes são os *drives* de dispositivos, responsáveis pela interface de acesso ao hardware.

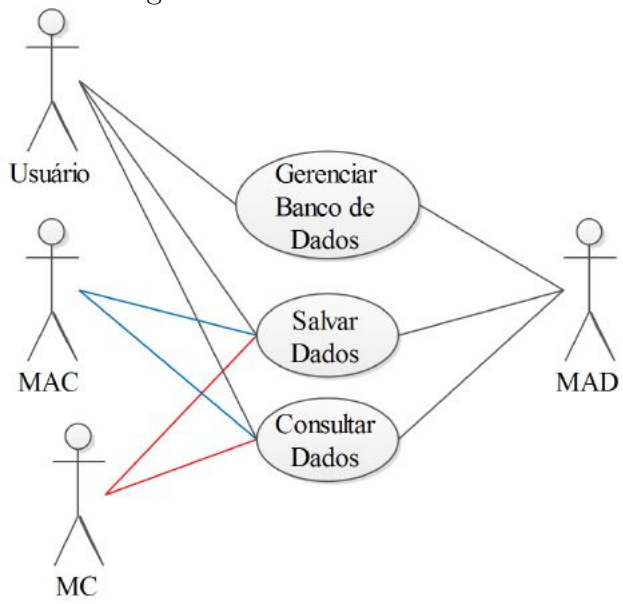
Nestes módulos é fundamental a utilização dos *driver* para o acesso das portas digitais, ADC e PWM. Se esses *drivers* já não forem disponíveis é preciso construi-los para o funcionamento destes módulos. Qualquer alteração na configuração do hardware, *drivers* adequados deveram ser instalados e assim toda a estrutura do MoFA deste trabalho terá funcionalidade. No Apêndice B é apresentado o ambiente utilizado neste trabalho para desenvolver drivers para sistemas operacionais embarcados.

3.1.3 Módulo de armazenamento de dados

Na automação industrial, é muito importante guardar dados do processo para análise e tomada de decisões. Geralmente na estrutura de um sistema AIC quem realiza essas análises e decisões são os níveis 3 e 4, como pode ser visto na seção 2.1. Quebrando um pouco do paradigma da estrutura de um sistema AIC, o modelo do MoFA possui um módulo de armazenamento de dados no nível de controlador, já que MCUs de ARM podem prover recursos computacionais para isso.

Este módulo permite operações inerentes a armazenamento e manipulação de dados sobre o microcontrolador. O armazenamento pode ser realizado na forma de arquivos e banco de dados. Como pode ser visto na Figura 3.1 o fluxo de dados proposto para este módulo é bidirecional em relação ao módulo de algoritmo de controle, bidirecional em relação ao módulo de comunicação e em uma única direção em relação ao RDIOC. Lembrando que alguns sistemas operacionais embarcados dão suporte a banco de dados, é o caso do sistema Android™, cujo banco de dados suportado é o SQLite (MEIER, 2012). Nas aplicações desenvolvidas neste trabalho utilizou a forma de armazenamento de arquivo. Para entender as funcionalidades deste módulo no MoFA, a Figura 3.3 define os casos de uso para este módulo.

Figura 3.3: Diagrama de casos de uso referente ao MAD.



Fonte: Autoria Própria.

- **Caso de uso: Gerenciar Banco de Dados**

Descrição:	Este caso de uso começa quando um usuário usa o MAD para manipular bancos de dados na MCU ARM. O objetivo deste caso de uso é efetuar as operações relacionadas à criação, alteração e exclusão de banco de dados.
Atores Primários:	Usuário.
Atores Secundários:	MAD.
Fluxo Principal	
Descrição:	**Criar banco de dados.**
Detalhe:	1 – Ator insere informações sobre o novo banco de dados. 2 – MAD verifica se informações já foram cadastrados anteriormente. 3 – MAD salvar o novo banco. 4 – MAD lista bancos já cadastrados.
Fluxos Alternativos	
Descrição:	**Alterar banco de dados.**
Detalhe:	1 – Usuário seleciona banco de dados cadastrado anteriormente. 2 – Usuário altera informações sobre o banco de dados. 3 – MAD verifica se informações já foram cadastradas anteriormente. 4 – MAD atualiza as informações. 5 – MAD lista bancos já cadastrados.
Fluxo de exceção:	
Descrição:	**Excluir banco de dados.**

Detalhe:	1 – Usuário seleciona banco cadastrado anteriormente. 2 – Usuário efetua exclusão. 3 – MAD verifica se informações já foram cadastradas anteriormente. 4 – MAD exclui banco. 5 – MAD lista bancos já cadastrados.
Regras de negócio	
Ação Restritiva:	Caso algum banco esteja sendo utilizado pelo MAC ou MC, este banco não poderá ser excluído.

- **Caso de uso: Salvar Dados**

Descrição:	Este caso de uso começa quando um ator primário usa MAD para salvar dados em um banco de dados ou arquivo.
Atores Primários:	Usuário, MAC, MC
Atores Secundários:	MAD
Pré-condições:	Pelo menos um banco de dados precisa está cadastrado ou existir um arquivo para armazenamento.
Fluxo Principal	
Descrição:	**Salvar dados.**
Detalhe:	1 – Ator primário escolhe modo de armazenamento, podendo ser por banco de dados ou arquivo. 2 – Ator solicita conexão a um banco de dados ou abertura de um arquivo para escrita. 3 – MAD cria a conexão com banco ou abre o arquivo para escrita. 4 – Ator primário envia dados para o MAD. 5 – MAD salva estes dados no modo escolhido em 1.
Fluxos Alternativos	
Descrição:	

Detalhe:	
Fluxo de exceção:	
Descrição:	**Erro ao salvar dados.**
Detalhe:	1 – MAD informa que houve erro ao salvar arquivo.
Regras de negócio	

- **Caso de uso: Consultar Dados**

Descrição:	Este caso de uso começa quando um ator primário usa o MAD para realizar uma consulta de dados salvos na MCU ARM.
Atores Primários:	Usuário, MAC, MC
Atores Secundários:	MAD
Pré-condições:	Pelo menos um banco de dados precisa está cadastrado ou existir um arquivo de armazenamento.
Fluxo Principal	
Descrição:	**Consultar dados.**
Detalhe:	1 – Ator primário escolhe modo de armazenamento, podendo ser por banco de dados ou arquivo. 2 – Ator solicita conexão a um banco de dados ou abertura de um arquivo para leitura. 3 – MAD cria a conexão com banco ou abre o arquivo para leitura. 4 – MAD realiza consulta ao banco ou leitura de arquivo.
Fluxos Alternativos	
Descrição:	
Detalhe:	
Fluxo de exceção:	
Descrição:	**Erro ao consultar dados.**
Detalhe:	1 – MAD informa que houve erro ao consultar dados.

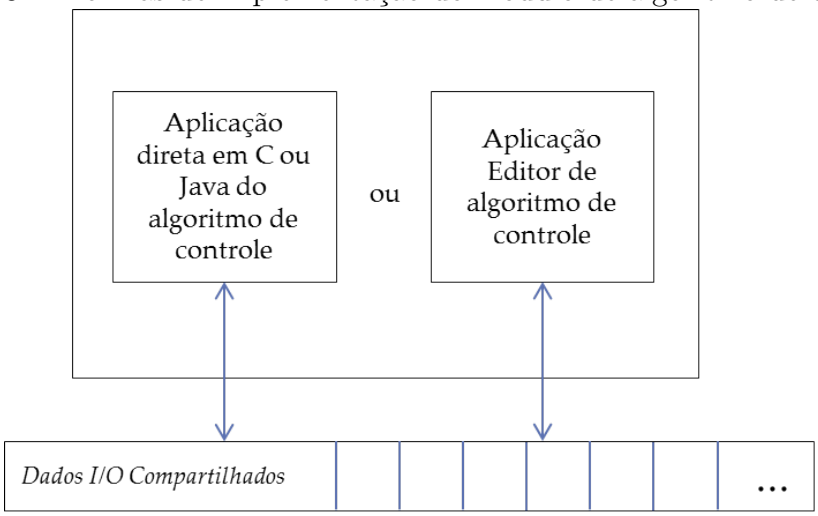
Regras de negócio	

3.1.4 Módulo de algoritmo de controle

O módulo de algoritmo de controle tem a funcionalidade de organizar logicamente tarefas de controle. Esse módulo pode se comunicar com todos os outros módulos da estrutura.

Este módulo possui duas formas de implementação: a primeira é o desenvolvimento de uma aplicação direta, onde o software é desenvolvido para uma aplicação específica de controle e supervisão. A segunda forma é o desenvolvimento de uma aplicação como uma ferramenta, ou seja, um software que permite ao usuário criar em alto nível suas próprias regras de controle. Um exemplo de implementação utilizando a segunda forma seria um software para desenvolvimento em linguagem Ladder ou qualquer outra linguagem especificada na norma IEC 61131–3. Assim, um usuário que domina essas linguagens para se trabalhar com CLP, também poderia utilizar o microcontrolador com o MoFA. Nos experimentos deste trabalho utilizou-se a primeira forma para implementação deste módulo.

Figura 3.4: Formas de implementação do módulo de algoritmo de controle.

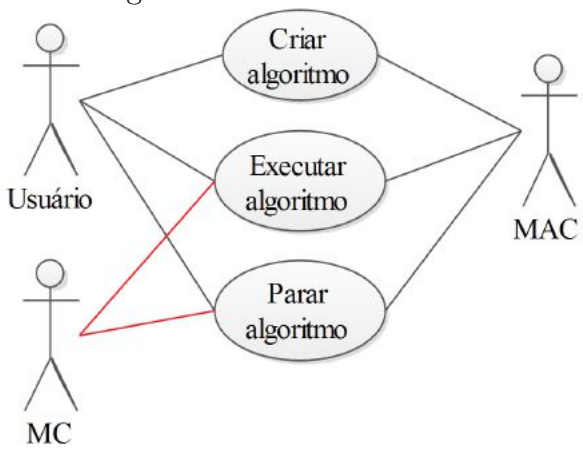


Fonte: Autoria Própria.

Na figura 3.4 são mostradas as duas formas propostas para a implementação do módulo

de algoritmo de controle. A primeira pode ser uma aplicação direta do algoritmo de controle desenvolvido sobre o sistema operacional embarcado. A segunda é uma aplicação também desenvolvida para tal sistema, mas com a tarefa de ferramenta de desenvolvimento, onde o usuário em alto nível pode de forma fácil e prática organizar seu algoritmo de controle. Um exemplo, como já foi descrito, seria um editor Ladder, cujo código do usuário ao ser compilado e executado, proporciona mudanças sobre as variáveis da RDIOC.

Figura 3.5: Diagrama de casos de uso referente ao MAC.



Fonte: Autoria Própria.

Os casos de uso que determinam as funcionalidades presente neste módulo são visualizados na Figura 3.5. O detalhamento de cada caso de uso é apresentado nos pontos a seguir.

- **Caso de uso: Criar Algoritmo**

Descrição:	Este caso de uso começa quando um usuário usa o MAC para criar rotinas de controle. O objetivo deste caso de uso é permitir ao usuário criar, editar e excluir rotinas de controle para serem executadas na MCU ARM.
Atores Primários:	Usuário.
Atores Secundários:	MAC.
Fluxo Principal	
Descrição:	**Criar algoritmo.**

Detalhe:	1 – Ator primário seleciona criar novo algoritmo de controle. 2 – Ator monta as rotinas do algoritmo. 3 – MAC verificar se rotina esta correta. 4 – Ator solicita ao MAC salvar algoritmo. 5 – MAC salva o algoritmo. 6 – MAC lista algoritmos cadastrados.
Fluxos Alternativos	
Descrição:	**Alterar algoritmo.**
Detalhe:	1 – Ator seleciona algoritmo cadastrado anteriormente. 2 – Ator altera rotinas e efetua a atualização. 3 – MAC atualiza o algoritmo. 4 – MAC lista algoritmos cadastrados.
Fluxo de exceção:	
Descrição:	**Excluir algoritmo.**
Detalhe:	1 – Ator seleciona algoritmo cadastrado anteriormente. 2 – Ator efetua a exclusão. 3 – MAC exclui algoritmo. 4 – MAC lista algoritmos já cadastrados.
Regras de negócio	
Interface:	A interface tem que ser amigável para o usuário criar as rotinas de controle, podendo explorar o desenvolvimento em linguagens definidas pela IEC 61131-3, tais como Ladder ou blocos funcionais.
Ação Restritiva:	Caso algum algoritmo esteja sendo executado, não será possível exclui este algoritmo.

- **Caso de uso: Executar Algoritmo**

Descrição:	Este caso de uso começa quando um dos atores primários usa o MAC para executar um algoritmo de controle já criado anteriormente ou já desenvolvido como aplicação direta.
Atores Primários:	Usuário, MC.
Atores Secundários:	MAC.
Fluxo Principal	
Descrição:	**Executar algoritmo.**
Detalhe:	1 – Ator primário seleciona algoritmo cadastrado. 2 – Ator primário solicita execução do algoritmo. 3 – MAC executa algoritmo em um ciclo infinito, até que o ator solicite a parada do algoritmo.
Fluxos Alternativos	
Descrição:	
Detalhe:	
Fluxo de exceção:	
Descrição:	
Detalhe:	
Regras de negócio	

- **Caso de uso: Parar Algoritmo**

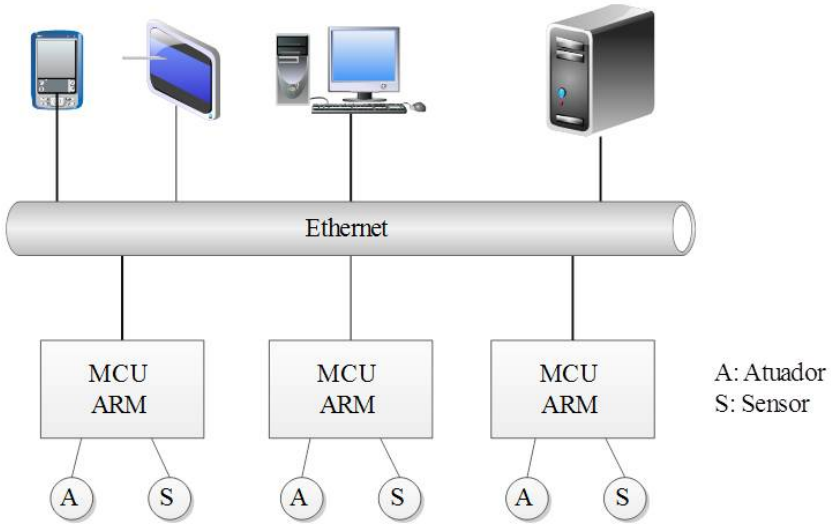
Descrição:	Este caso de uso começa quando um dos atores primários usa o MAC para encerrar a execução de algoritmo de controle na MCU ARM.
Atores Primários:	Usuário, MC
Atores Secundários:	MAC
Fluxo Principal	
Descrição:	**Parar algoritmo de controle.**

Detalhe:	1 – Ator primário solicita ao MAC a parada do algoritmo em execução na MCU ARM. 2 – MAC encerra execução do algoritmo. 3 – MAC lista algoritmos cadastrados.
Fluxos Alternativos	
Descrição:	
Detalhe:	
Fluxo de exceção:	
Descrição:	
Detalhe:	
Regras de negócio	

3.1.5 Módulo de comunicação

Este módulo permite escolher o meio físico pelo qual os dados do sistema de automação podem trafegar. É definida para o MoFA uma interface de porta serial RS–232/485 e Ethernet como meio físico, sendo que esta ultima é considerado o dispositivo ARM em uma rede local LAN. Essas duas interfaces são suportados pela maioria dos microcontroladores

Figura 3.6: Rede Ethernet LAN para acesso aos microcontroladores ARM.

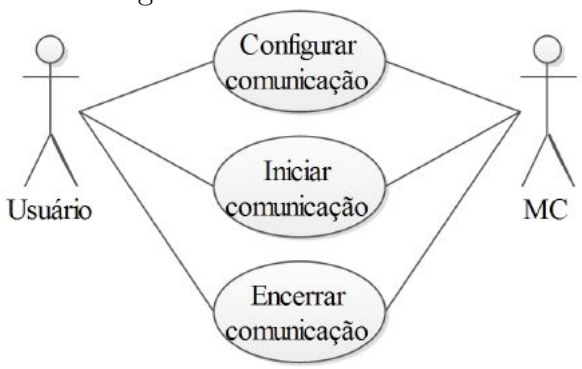


Fonte: Autoria Própria.

ARM. A estrutura básica de acesso via Ethernet ao microcontrolador ARM é mostrado na Figura 3.6. A partir desse módulo pode-se ter acesso de forma remota as variáveis da RDIOC, dados em arquivos ou banco de dados, e o algoritmo de controle.

Com este módulo é possível que outros dispositivos de outras camadas da automação acessem o microcontrolador de forma direta, quebrando o paradigma da pirâmide de um sistema AIC. Para utilização deste módulo, três casos de uso foram determinados. A Figura 3.7 ilustra estes casos de uso.

Figura 3.7: Diagrama de casos de uso referente ao MC.



Fonte: Autoria Própria.

- **Caso de uso: Configurar Comunicação**

Descrição:	Este caso de uso começa quando o usuário usa o MC para configurar uma conexão de rede.
Atores Primários:	Usuário
Atores Secundários:	MC
Fluxo Principal	
Descrição:	**Configurar comunicação.**
Detalhe:	1 - Ator seleciona um tipo de conexão. 2 – Ator informa dados da conexão. 3 - MC verifica se os dados estão corretos. 4 – MC salva os dados da configuração.
Fluxos Alternativos	
Descrição:	
Detalhe:	

Fluxo de exceção:	
Descrição:	
Detalhe:	
Regras de negócio	

- **Caso de uso: Iniciar Comunicação**

Descrição:	Este caso de uso começa quando o usuário usa o MC par iniciar a comunicação com outro dispositivo. O objetivo deste caso de uso é permitir a conexão da MCU ARM a outro dispositivo para troca de informação ou execução de alguma tarefa na MCU, remotamente.
Atores Primários:	Usuário
Atores Secundários:	MC
Fluxo Principal	
Descrição:	**Iniciar comunicação.**
Detalhe:	1 – Ator seleciona o tipo de conexão. 2 – MC verifica a configuração da conexão selecionada. 3 – MC inicia a conexão da comunicação.
Fluxos Alternativos	
Descrição:	
Detalhe:	
Fluxo de exceção:	
Descrição:	**Erro de conexão.**
Detalhe:	1 – O MC informa que não foi possível iniciar comunicação.
Regras de negócio	

Fato Estrutura:	O tempo de comunicação precisa está de acordo com o projeto de automação que se deseja fazer. Caso precise estabelecer tempos críticos da aplicação, o tipo de padrão, e conexão de rede precisa está de acordo com estas características.

- **Caso de uso: Encerrar Comunicação**

Descrição:	Este caso de uso começa quando o usuário usa o MC para encerrar uma conexão estabelecida anteriormente.
Atores Primários:	Usuário
Atores Secundários:	MC
Fluxo Principal	
Descrição:	**Encerrar comunicação.**
Detalhe:	1 – Ator solicita ao MC o encerramento da comunicação. 2 – MC encerra a conexão da comunicação.
Fluxos Alternativos	
Descrição:	
Detalhe:	
Fluxo de exceção:	
Descrição:	
Detalhe:	
Regras de negócio	

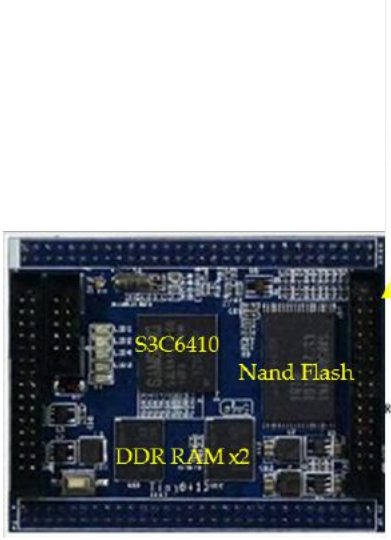
3.2 Materias e métodos

Esta seção descreve as ferramentas utilizadas para implementação e validação da abordagem proposta.

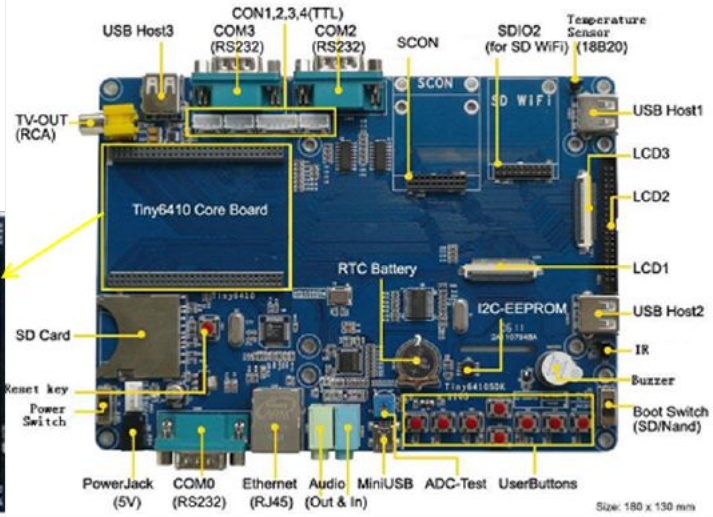
3.2.1 Kit de desenvolvimento Tiny6410

Para mostrar a aplicabilidade do MoFA em um microcontrolador ARM, foi utilizado o kit de desenvolvimento Tiny6410, fabricado por uma empresa chinesa, a *Friendly* ARM. O kit possui um módulo principal com um microprocessador Samsung S3C6410 ARM11 de 533MHz, 256 de memória RAM e 2GB de memória NAND flash. Este módulo principal, mostrado na Figura 3.8(a), é conectado a uma placa-mãe. No mercado atual além deste

Figura 3.8: Kit Tiny6410.



(a) Módulo Principal



(b) Placa-Mãe

(c) Placa Completa

Fonte: (FRIENDLYARM, 2012).

Kit também existe outras plataformas de desenvolvimento para ARM, tais como *Raspberry PI*, *BeagleBoardxM*, *Panda Board ES*, entre outras.

O kit Tiny6410 é de baixo custo e tem a possibilidade de desenvolvimento em plata-

formas livres, além de oferecer vários recursos em sua placa mãe como pode ser visto na 3.8(b) tais como: três portas RS232, três portas USB, LCD touchscreen resistivo, driver de cartão SD, porta Ethernet, botões, entre outros. A 3.8(c) mostra a placa completa do kit Tiny6410, onde é composta pelo módulo principal conectado a placa-mãe, sendo este módulo a CPU do microcontrolador ARM.

O kit possui dois DVDs que acompanham manuais em inglês para uso geral da placa e auxílio na instalação dos sistemas operacionais embarcados, assim como ferramentas para desenvolvimento e imagens já criadas de sistemas operacionais. Quatro sistemas operacionais estão disponíveis no kit: Windows CE, Android™, Linux Qtopia e Ubuntu.

A instalação do sistema operacional no kit Tiny6410 pode ser realizada via cabo usb ou pelo cartão SD. Nesta instalação também é possível escolher o local da instalação e execução do sistema, podendo ser feita a partir da memória Nand flash do módulo principal ou do cartão SD. No DVD do kit é disponibilizado um manual com passos para instalação dos sistemas operacionais (FRIENDLYARM, 2012). No Apêndice A é demonstrado como foi instalado o sistema operacional Android™ na placa Tiny6410.

3.2.2 Ferramentas de desenvolvimento

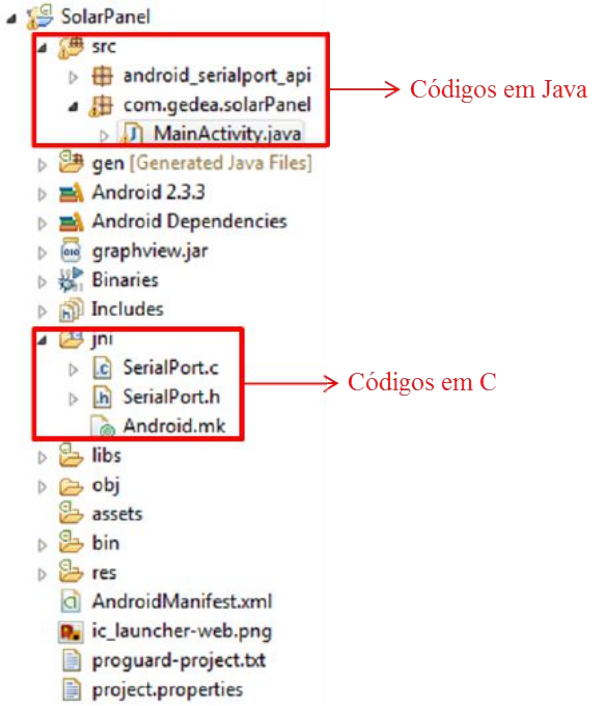
Para desenvolvimento das aplicações embarcadas deste trabalho foram ultilizados pacotes de ferramentas instaladas em um notebook com sistema operacional Linux Ubuntu 12.04. Um primeiro pacote de ferramenta foi utilizado para configuração e compilação do kernel do Linux, e um segundo pacote para desenvolvimento de aplicações Android. No Apêndice B é descrito os passos que foram realizados para a compilação e configuração do Kernel do Linux.

Em sistemas operacionais embarcados, o acesso ao hardware é realizado via *drivers*. Estes *drivers* ficam instalados no kernel do sistema operacional, em uma pasta chamada /dev. No kit Tiny6410, os sistemas operacionais disponíveis já incluem alguns *drivers* criados. Para este trabalho foram utilizados três *drivers*, os quais já eram inclusos no kernel dos sistemas operacionais e permitiam o acesso a uma porta ADC, porta serial e uma porta do tipo PWM. No driver da porta PWM foram necessários fazer alterações para que se pudessem executar mudanças na largura do pulso do PWM, denominado *dutycycle*. O *driver* original permitia apenas mudanças na frequência. No Apêndice C é

mostrado o código do driver PWM com as alterações realizadas.

O pacote de ferramentas utilizado para desenvolvimento de aplicações Android foi o *Android Developer Tools* (ADT) bundle. Neste pacote inclui componentes essenciais do Android SDK, e uma versão do ambiente de desenvolviento Eclipse com plugin ADT (GOOGLE, 2013b). Além do Android SKD, também utilizado o Android NDK para programação de código nativo escrito em C/C++(GOOGLE, 2013a). Na Figura 3.9 é visualizada a estrutura de um projeto de aplicação Android no ambiente de desenvolvimento Eclipse.

Figura 3.9: Estrutura de um projeto Android no ambiente Eclipse.



Fonte: Autoria Própria.

As aplicações Android desenvolvidas neste trabalho foi implementada em linguagem Java, e outra parte em linguagem C. No ambiente Eclipse, os pacotes de códigos fonte java são localizados no diretório src, mostrado na Figura 3.9. Já os códigos fontes C são desenvolvidos no diretório jni.

3.3 Conclusões

Foi apresentada neste capítulo a solução MoFA estruturada em módulos funcionais de software, visando assim, ser útil no desenvolvimento de sistemas de automação e controle.

No MoFA cada módulo é responsável por uma função característica no microcontrolador ARM. Em cada módulo foi mostrado os casos de uso referentes as suas funcionalidades na solução MoFA. No próximo capitulo é apresentado duas aplicações e experimentos que foram desenvolvidos baseados no MoFA.

4 Resultados

Com o objetivo de mostrar a utilização do MoFA em uma MCU ARM, foram realizadas duas atividades experimentais com a placa Tiny6410. A primeira delas foi um sistema de supervisão e aquisição de dados para painéis fotovoltaicos e a segunda um sistema de controle de nível de água. Nas seções a seguir serão descritos como foram realizados tais experimentos.

4.1 Sistema de Supervisão e Aquisição de Dados para Painéis Fotovoltaicos

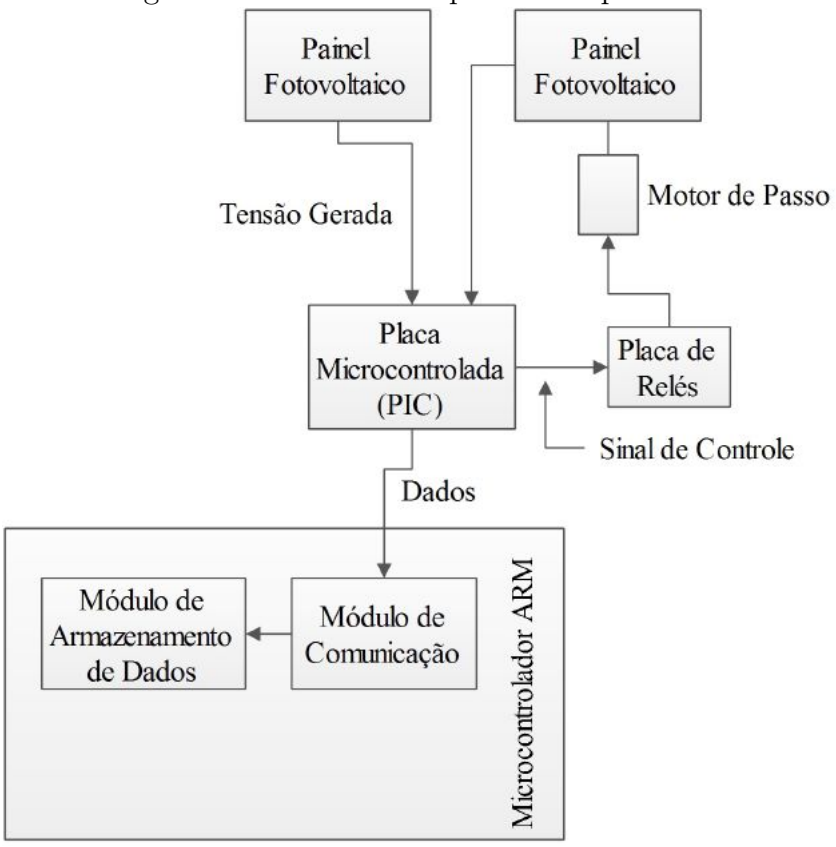
O primeiro experimento realizado foi o desenvolvimento de um sistema de supervisão e aquisição de dados na placa Tiny6410, com o interesse em analisar e avaliar dados sobre a eficiência na geração de energia elétrica por dois módulos fotovoltaicos instalados na cidade de Mossoró/RN, dos quais um é acoplado a um sistema do tipo de posicionamento estático, e o outro a um sistema com rastreamento solar em um único eixo. Vale ressaltar que o interesse deste trabalho é enfatizar o uso do MoFA em um microcontrolador ARM da placa Tiny6410 no experimento.

4.1.1 Desenvolvimento do Sistema

Para o desenvolvimento do sistema supervisório foi utilizado dois módulos funcionais do modelo do MoFA, o módulo de comunicação e o módulo de armazenamento de dados. Para entender melhor como o MoFA se encaixa nesta aplicação é mostrado uma visão geral do sistema na Figura 4.1.

Como pode ser visto na Figura 4.1 o módulo de comunicação recebe os dados enviados pela placa microcontrolada PIC. Essa comunicação é realizada via porta RS–232 de ambas as placas de microcontroladores, a placa com o microcontrolador PIC e a placa Tiny6410 com o microcontrolador ARM. A placa Tiny6410 é constituída de um sistema operacional Android instalado da forma descrita no Apêndice A. O software supervisório desenvolvido

Figura 4.1: Sistema completo do experimento.

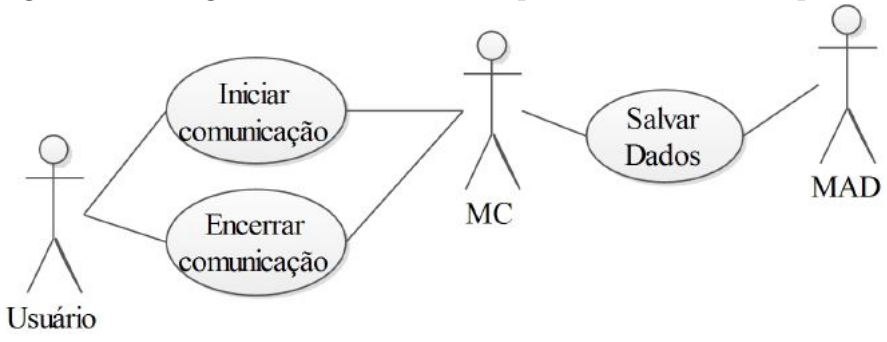


Fonte: Autoria Própria.

para o Android foi implementado no ambiente Eclipse.

A estrutura geral mostrada na Figura 4.1 foi o ponto de partida para desenvolvimento do sistema supervisório. O primeiro passo foi determinar qual dos casos de uso seria preciso para desenvolvimento deste sistema. Diante do cenário da Figura 4.1 foi determinado o relacionamento entre os atores Usuário, MC e MAD. Os casos de uso associados a esses relacionamentos são mostrados na Figura 4.2. O detalhamento destes casos de uso pode ser visto no Capítulo 3.

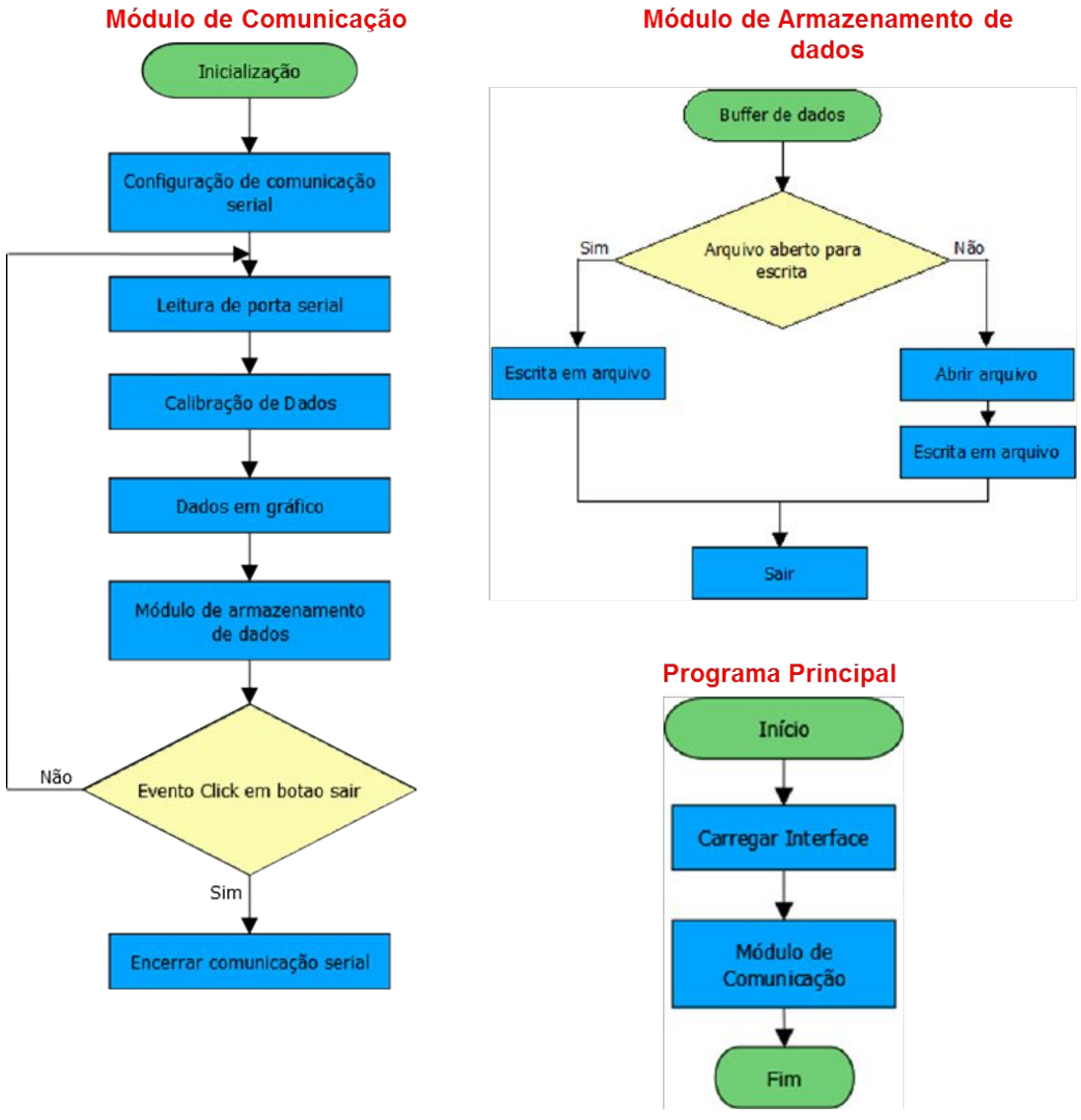
Figura 4.2: Diagrama de casos de uso para o sistema de supervisão.



Estes casos de uso da Figura 4.2 determinam as funcionalidades necessárias para criação do sistema supervisório. O relacionamento do ator Usuário e MC representa a interação de um Usuário real e a interface desenvolvida para a MCU ARM. O relacionamento do ator MC e MAD representa a solicitação do armazenamento de dados em um arquivo.

Na Figura 4.3 é possível visualizar melhor como acontecem as interações entre os módulos. Em primeira instância ao ser iniciada o programa principal a interface é carregada para que o usuário interaja com o sistema. O usuário ao iniciar a supervisão do sistema, a comunicação serial da MCU ARM com o PIC é configurada e estabelecida. Logo em seguida a leitura da porta serial começa a ser feita, e os dados recebidos são ca-

Figura 4.3: Fluxogramas do programa supervisório.



librados. Posteriormente estes dados são plotados em um gráfico na interface do usuário e passados para o módulo de armazenamento de dados para serem salvos. O módulo

de armazenamento realiza a abertura de um arquivo para salvar os dados recebidos pelo módulo de comunicação. Essa ação é realizada até o usuário solicitar o encerramento da comunicação a partir de um evento click no botão sair da interface.

A interface do supervisório foi criada na IDE eclipse, utilizando o plugin ADT. Para plotagem de gráficos foi utilizado o pacote *graphview* para Android™ disponível em (GEHRING, 2013). A interface do supervisório é visualizada na Figura 4.4, onde os dados foram recebidos pela aplicação Android, a qual apresentou em gráficos os valores de tensão lidos de ambos os painéis fotovoltaicos. Estes valores foram, então, armazenados por meio de arquivo de texto, em um cartão SD.

Figura 4.4: Sistema supervisório em Android.

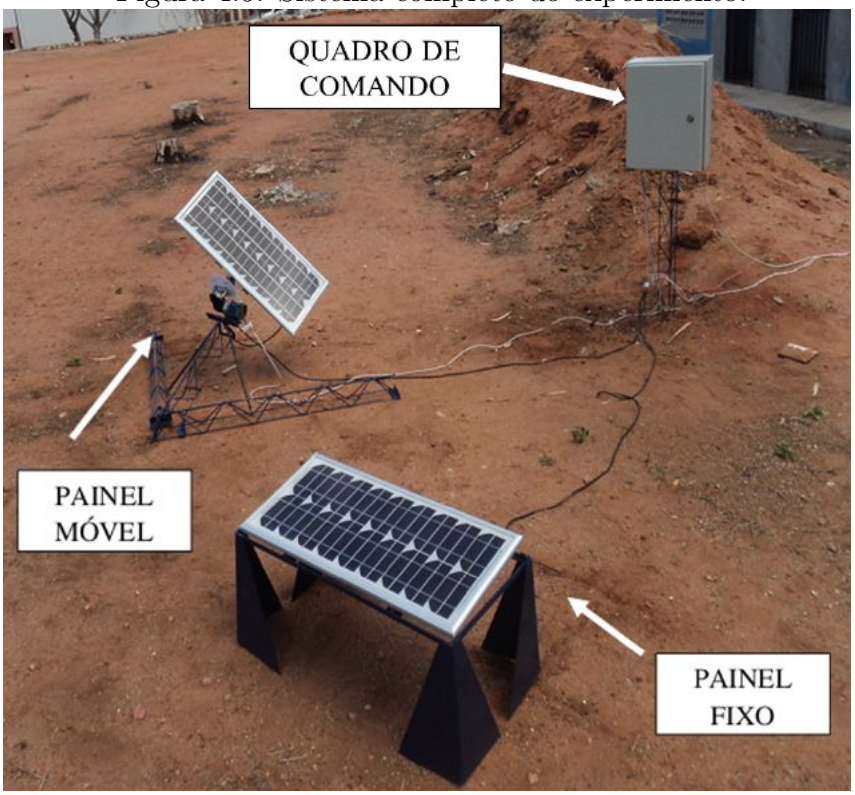
4.1.2 Montagem do experimento

O material necessário à realização do experimento constitui-se principalmente de: cargas resistivas, dois módulos fotovoltaicos idênticos, dois suportes para os módulos fotovoltaicos, um motor de passo com seu respectivo driver, uma bateria, uma placa microcontrolada (PIC), a placa Tiny6410 já configurada de acordo com a seção 3.2, um quadro para instalação do sistema de comando, um suporte para o quadro de comando, fusíveis para proteção do sistema, e fiação.

Foi acoplado a cada painel fotovoltaico um resistor de 15Ω para funcionar como cargas de teste. . Os dois painéis solares escolhidos para a realização deste trabalho são idênticos e pertencem ao Laboratório de Energia da UFERSA. Vale lembrar que estão

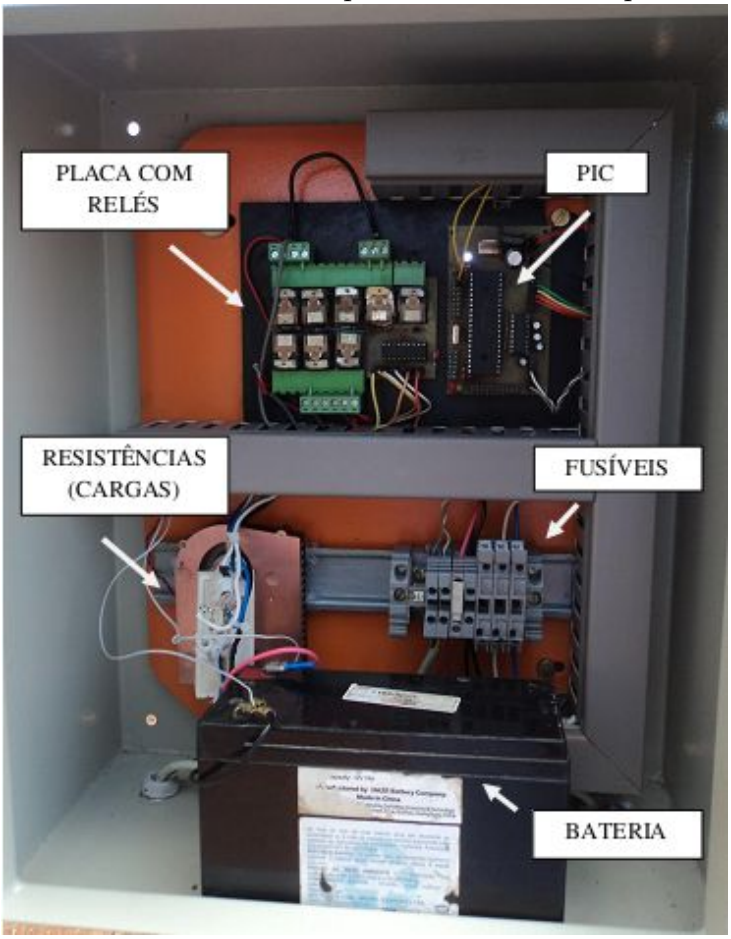
sendo utilizados dois painéis, devido à análise que será realizada em dois tipos de sistemas, um estático e o outro móvel. Os módulos possuem pequenas dimensões e baixa potência, na faixa de $20W$, com tensão máxima de $16.8V$, o que fornece uma maior praticidade ao desenvolvimento da prática. A Figura 4.5 mostra o posicionamento dos módulos fotovoltaicos com suas devidas estruturas de suporte, e o quadro de comando. A estrutura que segura o painel móvel é acoplada ao eixo de um motor de passo, responsável pela movimentação da placa durante o período do experimento.

Figura 4.5: Sistema completo do experimento.



No quadro de comando estão os componentes responsáveis pela coleta dos dados de geração de energia das placas, controle de movimentação da placa móvel e comunicação com o supervisório. Na Figura 4.6 é mostrado como os componentes estão dispostos no quadro de comando. A placa com o microcontrolador PIC tem as tarefas de coleta de dados de geração via portas analógicas, acionamento da placa de relés para movimentação do motor de passo com a placa acoplada a ele, assim como o envio de dados via porta serial para o sistema de supervisão. Para alimentação dos componentes no quadro de comando foi utilizada uma bateria de $12V$, e para proteção de tal sistema a utilização de fusíveis. Para coleta dos dados de tensão dos painéis foi preciso utilizar um divisor de tensão, visto que a tensão gerada pelos painéis eram superior a $5V$, limite este provido pelas portas analógicas do PIC.

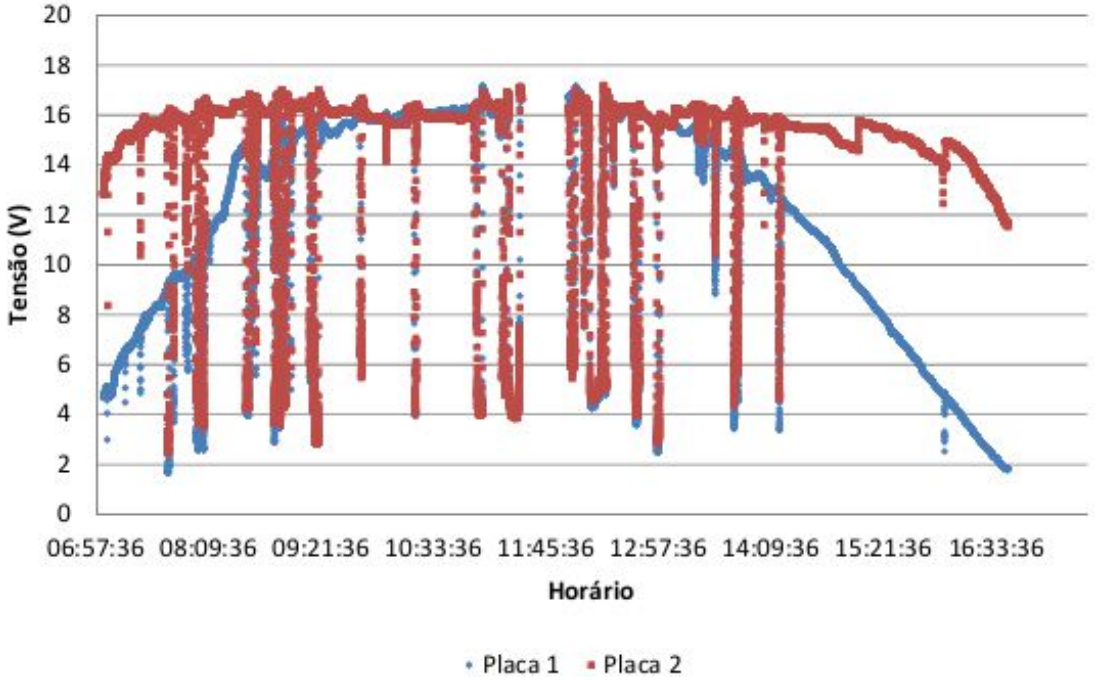
Figura 4.6: Posicionamento dos componentes dentro do quadro de comando.



4.1.3 Aquisição de Dados

Após alguns testes foi iniciada a etapa de levantamento de dados com os sistemas fotovoltaicos fixo e móvel. A periodicidade de coleta dos dados de tensão dos painéis foi de 1 segundo, ao longo de 12 horas em um dia, o que resulta num total de 43200 leituras de tensão para cada painel, totalizando 86400 leituras. No entanto, serão apresentados neste trabalho, apenas os valores obtidos entre os horários das 07 : 01 : 54h às 17 : 12 : 31h, já que até as 7 : 00h os painéis encontravam-se sombreados, e após as 17 : 00h a claridade do dia já se encontrava bastante reduzida. A Figura 4.7 mostra o gráfico plotado no *software* Excel®, e referem-se, respectivamente, aos valores de tensão para cada painel fotovoltaico. Esses dados foram extraídos do arquivo texto criado no cartão SD pelo o sistema de supervisão.

Figura 4.7: Valores de tensão de saída dos painéis fotovoltaicos fixo e móvel.



4.2 Sistema de Controle PI em Tanque de Nível

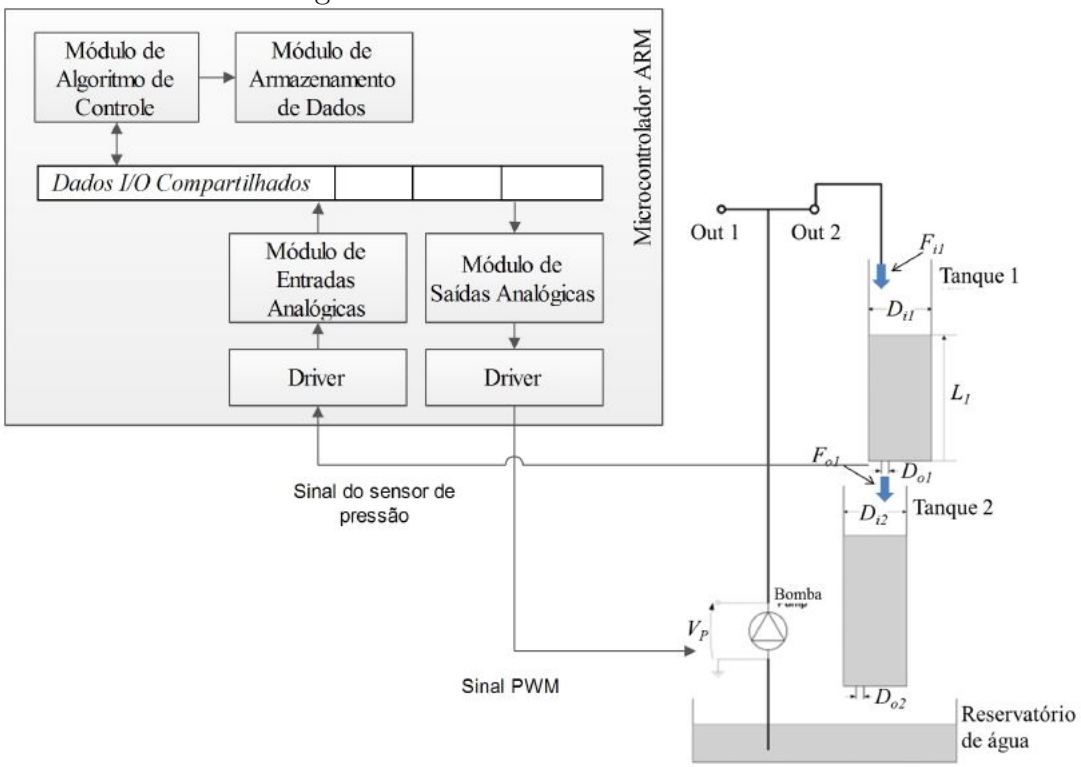
O segundo experimento realizado neste trabalho foi um sistema de controle de nível de água. Como planta a ser controlada, foi utilizado o sistema experimental de tanques acoplados de fabricação Quanser®, e como MCU foi utilizado a placa Tiny6410. Foi desenvolvida uma aplicação Android™ para realização do algoritmo de controle e gravação da resposta do sistema a diferentes *setpoints*. A seguir é descrito como foi desenvolvido o sistema baseado no MoFA e os materiais usados neste experimento

4.2.1 Desenvolvimento do sistema

Seguindo a mesma ideia de desenvolvimento realizado no experimento anterior, o projeto do sistema de controle embarcado para a MCU ARM foi baseado no modelo do MoFA. A Figura 4.8 mostra quais módulos da estrutura do MoFA foram utilizados para o desenvolvimento do sistema. Como pode ser visto os módulos utilizados foram: módulo de algoritmo de controle, módulo de armazenamento de dados, módulos de entradas analógicas e módulo de saídas analógicas.

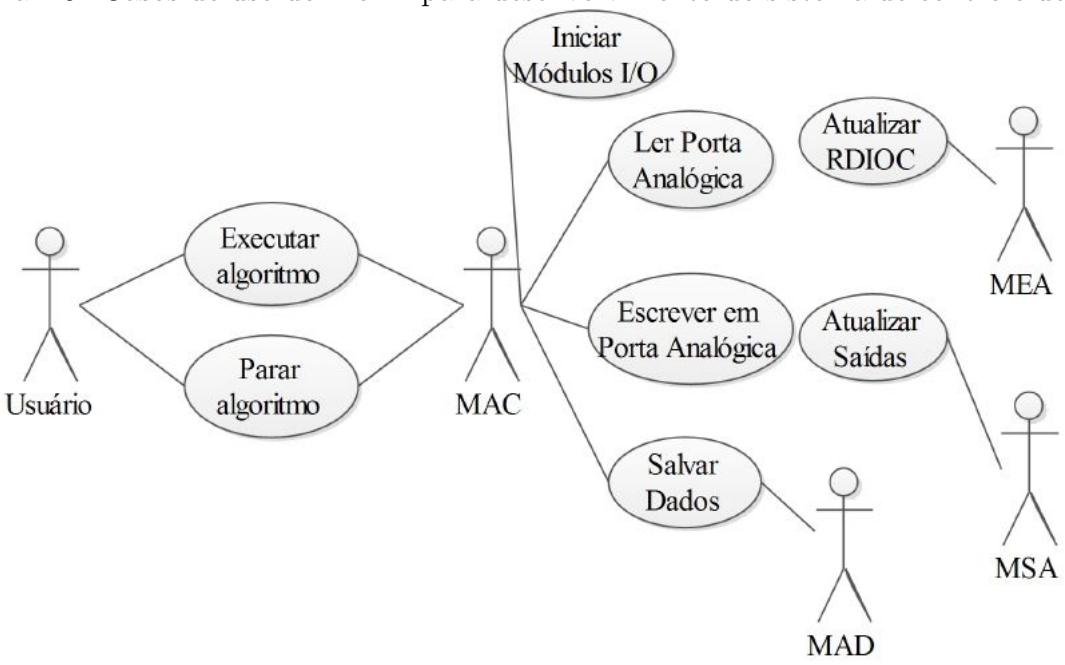
De acordo com essa estrutura da Figura 4.8, foi determinado os casos de uso do MoFA necessários para o desenvolvimento do sistema. Estes casos de uso podem ser

Figura 4.8: Sistema baseado no MoFA.



Fonte: Adaptado de Quanser (2013a).

Figura 4.9: Casos de uso do MoFA para desenvolvimento do sistema de controle de nível.

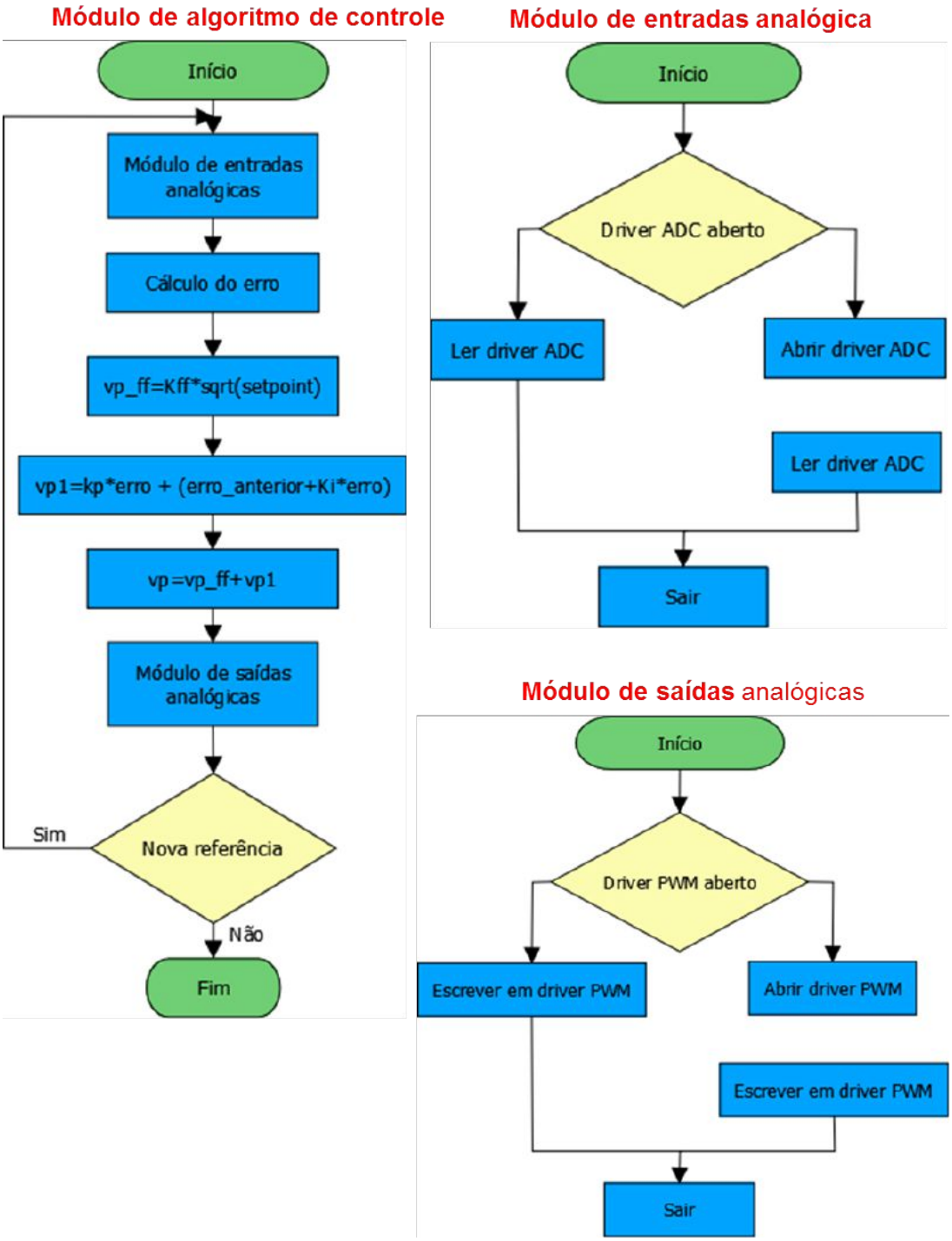


visto na Figura 4.9. Os relacionamentos do Usuário e o MAC representam a interação do usuário real com a interface do sistema de controle de nível na MCU ARM. Os casos de uso “Iniciar Módulos I/O”, “Ler Porta Analógica”, “Escrever em Porta Analógica” representam as funcionalidades de inicialização dos drivers das portas ADC e PWM da

MCU, acesso ao sensor de pressão da planta e o sinal de controle que é enviado para a bomba. O relacionamento do MAC com o MAD representa o armazenamento das informações de nível do tanque durante a execução do controle.

Os módulos de entradas e saídas analógicas foram responsáveis pelo acesso aos drivers das portas físicas do microcontrolador. Esses drives são aqueles obtidos pela configuração e compilação do kernel Linux, como descrito no Apêndice B. Um dos drives permite

Figura 4.10: Lógica do programa do sistema de controle de tanque de nível.



o acesso a uma porta ADC para realização da leitura do sensor de nível da planta de controle. O outro driver é responsável pelo acesso a uma porta do tipo PWM para envio de sinal de controla a bomba.

O módulo de entrada analógica realiza a leitura da porta ADC e atualiza a região de dados I/O compartilhado com o valor dessa leitura para que, dessa forma, o módulo de algoritmo de controle, possa acessar esse valor. Esta funcionalidade é representada pelo caso de uso "Atualizar RDIOC". O caso de uso "Atualizar Saídas" representa a funcionalidade que o módulo de saída analógica realiza ao atualizar a porta PWM de acordo com o valor presente na região de dados I/O compartilhado, na qual é atualizado pelo módulo de algoritmo de controle. Na figura 4.10 é visualizada a lógica de programação do sistema de controle.

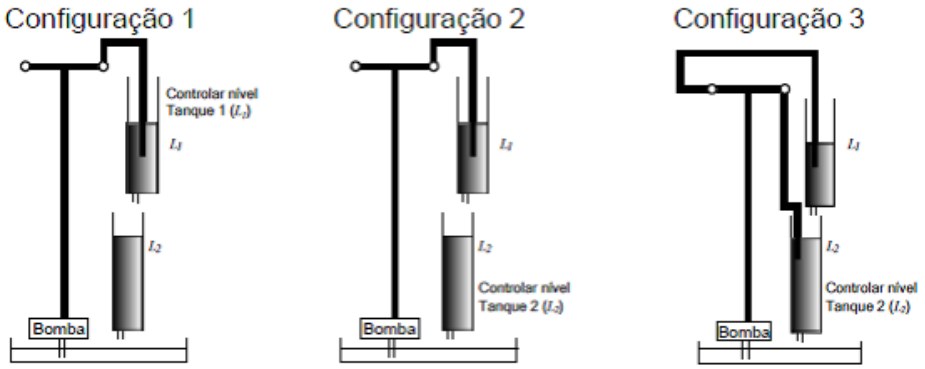
O módulo de armazenamento de dados foi responsável por salvar em arquivo texto os valores da resposta do sistema mostrado na Subseção 4.2.4. Semelhante ao experimento anterior esse arquivo esta localizado no cartão SD inserido na Tiny6410.

4.2.2 Montagem do Experimento

Os materiais utilizados na realização do experimento foram: planta didática, a placa Tiny6410, resistores e transistor para confecção de um divisor de tensão e driver PWM, módulo de potencia VoltPaq-X1, cabos para comunicação, fonte de tensão. Os pontos a seguir detalham como esses componentes foram utilizado no experimento.

- Planta didática Quanser de tanques acoplados – Nesse sistema, dois tanques e uma bomba podem ser dispostos em três configurações de forma a simular diferentes sistemas de controle de nível. Para esta aplicação foi utilizada a configuração 1, mostrada na Figura 4.11, na qual a bomba alimenta o tanque 1 com o objetivo de controlar o nível deste tanque. O tamanho do orifício de vazão do tanque utilizado foi $0,3175cm$. Na base do tanque possui um sensor de pressão para emitir o sinal do nível em tensão $0-5V$. A bomba da planta suporta uma tensão de até $23V$ por curtos períodos de tempo e de $12V$ de forma ininterrupta (QUANSER, 2013b).

Figura 4.11: Estrutura da planta de tanques acoplados.



Fonte: (FONSECA, 2010).

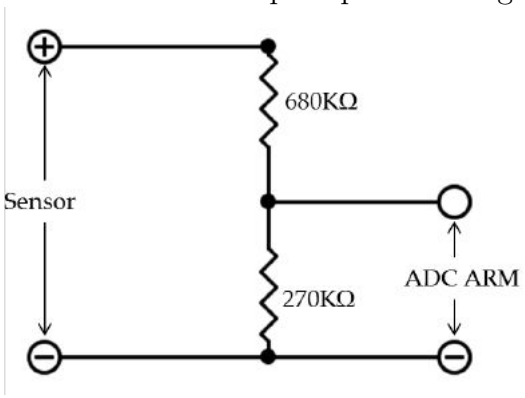
- Módulo de potência VoltPaq-X1 – Este módulo é responsável por receber os sinais dos sensores de pressão presente nos tanques e emitir um ganho de tensão $3x$ para bomba da planta. O módulo é mostrado na Figura 4.12.

Figura 4.12: Módulo de potencia VoltPaq-X1.

Fonte: (QUANSER, 2013c).

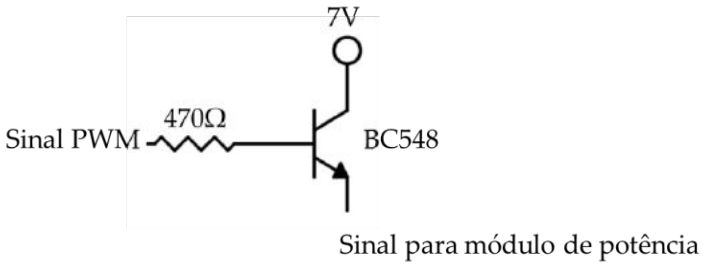
- Divisor de tensão – A porta analógica do microcontrolador ARM da placa Tiny6410 trabalha com sinais de tensão que variam entre $0-3.3V$, sendo que o sinal dos sensores da planta didática possui os sinais entre $0-5V$. De forma a compatibilizar esses sinais dos sensores com a entrada analógica do microcontrolador, foi estabelecido o circuito divisor de tensão da Figura 4.13.

Figura 4.13: Divisor de tensão para porta analógica do ARM.



- Driver PWM – Para o sinal de controle da bomba foi utilizado uma porta PWM do microcontrolador ARM. A Figura 4.14 mostra o driver utilizado para modularização do sinal de tensão de $0 - 7V$ ao módulo de potência. No módulo de potência esse sinal tinha um ganho de $3x$ para alimentar a bomba da planta.

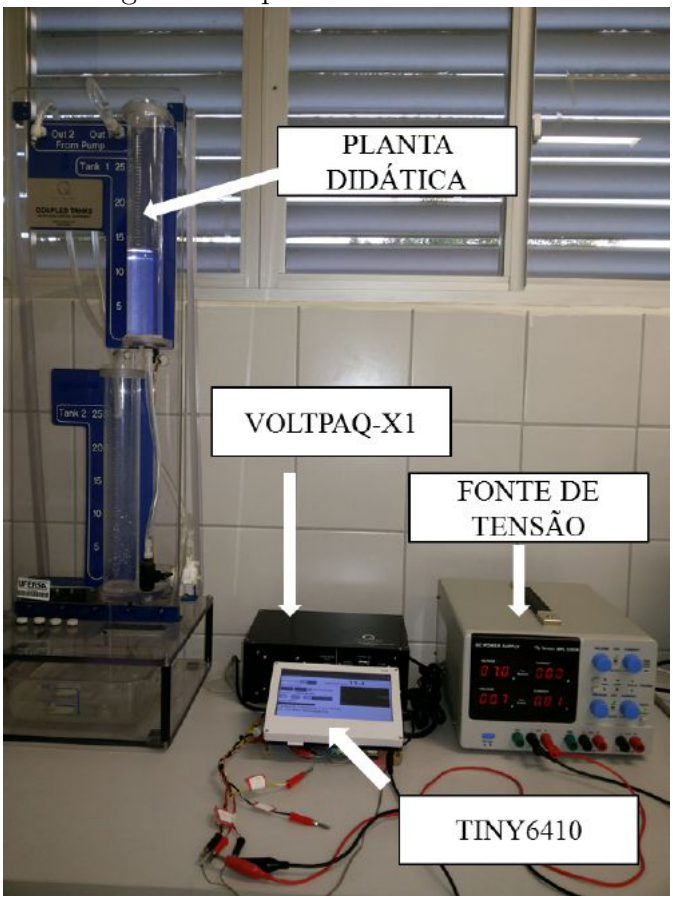
Figura 4.14: Driver PWM.



Este experimento foi realizado no laboratório do GEDEA localizado no prédio do Centro Integrado de Inovação Tecnológica do Semiárido (CITED) da UFERSA. A estrutura de montagem do experimento é mostrada na Figura 4.15. As ligações entre o módulo de potência e a placa Tiny6410 foram feitas através de cabos. A fonte de tensão fez a alimentação do driver PWM para envio de sinal para bomba. A planta didática, como já mencionado anteriormente foi configurada de modo a realizar o controle do tanque 1.

Antes da realização do experimento foram realizados calibrações do sensor do tanque 1. Essa calibração foi realizada ajustando os potenciômetros de offset e ganho do sensor. Estes potenciômetros ficam localizados na base da planta. O primeiro ajuste foi do offset com o tanque vazio para zerar a leitura feita pela aplicação android na placa Tiny6410. Depois foi ajustado o ganho com o tanque no nível de $15cm$.

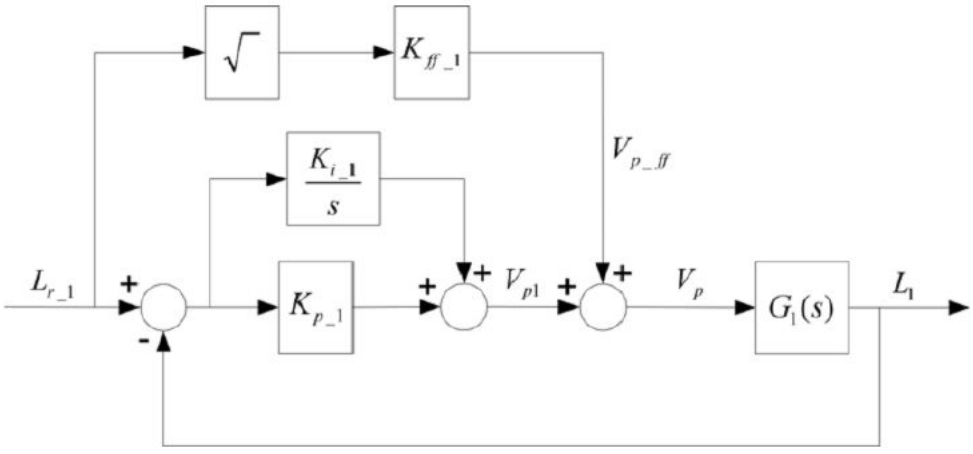
Figura 4.15: Montagem do experimento de controle de nível de tanque.



4.2.3 Modelo do Controlador PI

O controle de malha fechada utilizado neste experimento é mostrado na Figura 4.16, onde V_{p1} é o valor de voltagem da bomba gerado pelo controlador PI, e o V_{pff} é o valor de ação *feedforward*. Os valores de K_{i_1}, K_{p_1} e K_{ff_1} são calculados de acordo com o modelo

Figura 4.16: Loop do controle PI mais ação *feedforward* para nível de água.



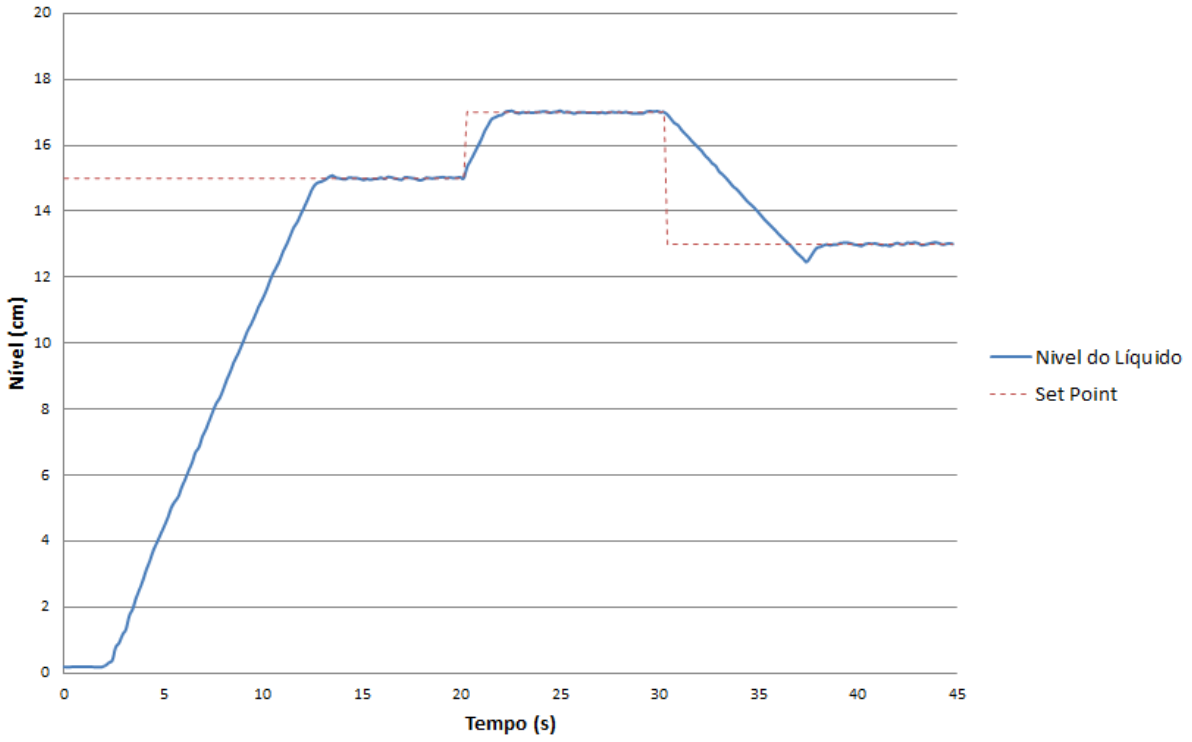
Fonte: (QUANSER, 2013a).

proposto pela Quanser (2013a). Não é o foco deste trabalho desenvolver o equacionamento do controlador, mas sim implementar o modelo pronto da Quanser® para ser embarcado na placa Tiny6410.

4.2.4 Resposta do Sistema

Após a montagem do sistema de controle de tanque de nível, foi realizado testes para análise da resposta do sistema. Foram aplicados *setpoints* de 15cm, 17cm e 13cm em um intervalo de 45s, como pode ser visto na Figura 4.17

Figura 4.17: Gráfico de resposta do sistema de controle PI.

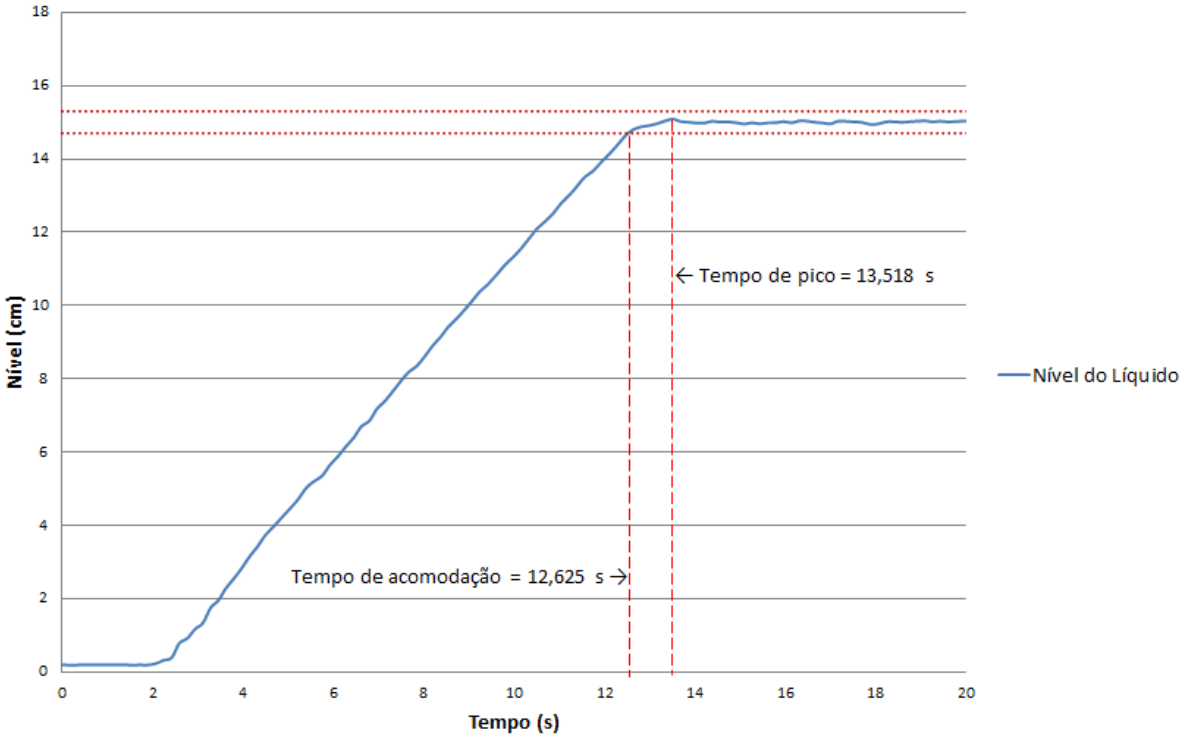


Para os primeiros 15s foram colhidos as variáveis de tempo de atraso (t_d), tempo de subida (t_r), tempo de pico (t_p), máximo sobre-sinal (M_r) e Tempo de acomodação (T_s). Em Ogata e Yang (2011) pode ser encontrado mais sobre a teoria do controle e o significado destes tempos. Na Figura 4.18 é visualizado os detalhes dos primeiros 15s de controle do tanque. Os valores encontrados para o controle PI foram:

- Tempo de atraso: $t_d = 7,151s$;
- Tempo de subida: $t_r = 13,338s$;

- Tempo de pico: $t_p = 13,518s$;
- Máximo sobre-sinal: $M_r = 0,571\%$;
- Tempo de acomodação: $T_s = 12,625s$ a creitério de 2%.

Figura 4.18: Gráfico de resposta ao degrau.



4.3 Conclusão

Neste capítulo foi mostrado duas atividades experimentais realizadas com a placa Tiny6410, com o intuito de mostrar a utilidade da solução MoFA em uma MCU ARM. Estas atividades tiveram o processo de execução dividido em três etapas principais: desenvolvimento do sistema de software embarcado para a MCU ARM, montagem do experimento e os resultados experimentais. Com a solução MoFA, na etapa de desenvolvimento do sistema embarcado foi determinado as funcionalidades necessárias para utilização da MCU ARM em cada atividade. O MoFA foi utilizado como base para o desenvolvimento dos sistemas, cada um com um propósito diferente, uma interface diferente, mas possuíam a mesma estrutura funcional determinada pelo MoFA.

5 Conclusões e trabalhos futuros

Neste trabalho foi apresentado alguns conceitos básicos da estrutura de um sistema AIC, destacando o nível 2, o nível dos controladores. Este nível foi o alvo de pesquisa deste trabalho, focando no estudo dos microcontroladores ARM e seu uso em sistemas AIC. Alguns trabalhos estudados expõe a ideia de que esses microcontroladores são bem mais eficientes do que microcontroladores convencionais. Eles podem oferece mais funcionalidades aos sistemas de automação, como comunicações Ethernet, interface touch screen e um maior poder de processamento, podendo realizar tarefas de controle mais complexas.

Com o intuito de explorar as funcionalidades dos microcontroladores ARM e elevar essas funcionalidades em nível de operação, foi apresentado neste trabalho o MoFA. O MoFA é uma solução que prover estruturar as funcionalidades daquele microcontrolador em módulos de software. Esses módulos seguem uma estrutura básica a ser útil a um sistema AIC. Tal estrutura constitui de um módulo de entradas e saídas digitais e analógicas, módulo de algoritmo de controle, módulo de comunicação e módulo de armazenamento de dados.

A grande questão aqui é que o MoFA sirva para a construção tanto de aplicações que trabalhe como ferramenta, facilitando o uso dos recursos de um microcontrolador ARM, como aplicações diretas para sistemas AIC utilizando programações mais avançadas, como C/C++ ou Java.

Foi apresentado também neste trabalho duas aplicações específicas realizadas com base no MoFA. Um sistema supervisório para painéis fotovoltaicos e um sistema de controle PI para controle de nível de tanque. A primeira aplicação se mostrou bem eficiente durante a coleta dos dados, mostrando em gráficos os valores lidos sobre as tensões geradas pelas placas solares. Na segunda aplicação o sistema também se mostrou eficiente e apresentando resultados bons, praticamente sem sobressinal. Nas duas aplicações desenvolvidas o processo de desenvolvimento do software embarcado foi baseado no MoFA, determinando quais casos de uso seriam necessários para atender as funcionalidades exigidas por cada aplicação.

A pesquisa deste trabalho se fez com a parceria da empresa de automação industrial, a EGM Tecnologia, que acreditou na ideia do uso de microcontroladores ARM em sistemas AIC e pretende elaborar uma placa dedicada a placa Tiny6410 para uso de padrões de sinais industriais. Esta mesma empresa já vendeu também uma solução em controle de fonte de plasma para um projeto de pesquisa, onde a base dessa solução está o uso do modelo do MoFA na construção da aplicação de controle da fonte. Foi utilizado também a placa Tiny6410 para esta solução.

Para as aplicações computacionais, o MoFA não se limita a uma tecnologia Tyni6410. A abordagem sugere aplicações a serem utilizáveis em qualquer plataforma e em qualquer caráter, seja aberta ou não, comercial ou livre, Android, Linux e Windows. Para isso é necessária a utilização de drives já disponíveis ou a criação de drivers inerente as mais variadas tecnologias e upgrade que possam existir.

Como sugestões de futuros trabalhos podem ser destacadas:

1. O desenvolvimento de aplicações na forma de ferramentas para a placa Tiny6410 ou outra plataforma ARM. Deixando as funcionalidades da placa em nível de operação. A ideia é que o usuário crie suas próprias funcionalidades em alto nível através das ferramentas desenvolvidas baseadas no modelo do MoFA.

2. Desenvolvimento de um sistema de controle de tanque de nível usando um controlador PI-Fuzzy.

3. Desenvolvimento de dispositivos de controle que tenham em sua base de software embarcado o MoFA.

4. Neste ano a ARM lançou uma nova arquitetura de seus microprocessadores, o ARMv8-R, direcionados para aplicações industriais e automotivas (ARM, 2013). Poderíamos ter, portanto um CLP que agregasse tanto um sistema operacional de propósito geral, como Android, Linux embarcado, Windows Ce, proporcionando novas experiências na programação desses equipamentos, com interfaces mais amigáveis oferecidas por aqueles sistemas operacionais, mas ao mesmo tempo ter o desempenho de um sistema operacional de tempo real.

Referências Bibliográficas

ANISH, M. Internal model control of pressure process using arm microcontroller. In: IEEE. *Computing, Electronics and Electrical Technologies (ICCEET), 2012 International Conference on*. [S.l.], 2012. p. 351–353.

ARM. *ARM11 Processor Family*. 2012. Disponível em: <`http://www.arm.com/products/processors/classic/arm11/index.php`>. Acesso em: fev. 2012.

ARM. *ARMv8-R Architecture*. 2013. Disponível em: <`http://www.arm.com/products/processors/instruction-set-architectures/armv8-r-architecture.php`>. Acesso em: out. 2013.

BALIEIRO, R. L. *Desenvolvimento de uma ferramenta computacional para aquisição via Internet de dados de dispositivo de campo em ambientes fieldbus*. 173 p. Dissertação (Mestrado) — Escola de Engenharia de São Carlos, USP, 2008.

BARRETT, S. F.; PACK, D. J. Microcontrollers fundamentals for engineers and scientists. *Synthesis Lectures on Digital Circuits and Systems*, Morgan & Claypool Publishers, v. 1, n. 1, p. 1–124, 2006.

BEZERRA, E. *Princípios de Análise e Projeto de Sistemas com UML*. [S.l.]: Elsevier, 2007.

BOLTON, W. *Programmable logic controllers*. [S.l.]: Access Online via Elsevier, 2009.

COREWIND. *Friendly ARM*. 2012. Disponível em: <`www.friendlyarm.net`>. Acesso em: fev. 2012.

FONSECA, D. G. V. da. *Desenvolvimento de um Software para a Sintonia de Malhas de Controle em Processos Industriais*. 51 f. Monografia (Graduação) — Departamento de Engenharia de Computação e Automação, Universidade Federal do Rio Grande do Norte, Rio Grande do Norte, 2010.

FRANC, H.; ŠAFARIČ, R. Arm-cortex microcontroller fuzzy position control on an automatic door test-bed. In: IEEE. *Robotics in Alpe-Adria-Danube Region (RAAD), 2010 IEEE 19th International Workshop on*. [S.l.], 2010. p. 417–422.

FRIENDLYARM. *User's Guide to Tiny6410 System Installation*. 2012. Disponível em: <`http://www.friendlyarm.net/products/tiny6410`>. Acesso em: fev. 2012.

GEHRING, J. *GraphView Library*. 2013. Disponível em: <`http://www.jjoe64.com/p/graphview-library.html`>. Acesso em: abr. 2013.

GOOGLE. *Android NDK*. 2013. Disponível em: <`http://developer.android.com/tools/sdk/ndk/index.html`>. Acesso em: jan. 2013.

GOOGLE. *Android SDK*. 2013. Disponível em: <`http://developer.android.com/sdk/index.html`>. Acesso em: jan. 2013.

JOHN, K. H.; TIEGELKAMP, M. *IEC 61131-3: programming industrial automation systems: concepts and programming languages, requirements for programming systems, decision-making aids*. [S.l.]: Springer, 2010.

KARIM, Y.; JON, M.; GILAD, B.; PHILIPPE, G. *Building embedded linux systems*. [S.l.]: O'Reilly Media Publishers, Sebastopol, 2008.

KRIVIC, S.; HUJDUR, M.; MRZIC, A.; KONJICIJA, S. Design and implementation of fuzzy controller on embedded computer for water level control. In: IEEE. *MIPRO, 2012 Proceedings of the 35th International Convention*. [S.l.], 2012. p. 1747–1751.

LI, G.-p. Design of an embedded control and acquisition system for industrial local area networks based on arm. In: IEEE. *Computer Science and Education (ICCSE), 2010 5th International Conference on*. [S.l.], 2010. p. 35–39.

LOVE, R. *Linux kernel development*. [S.l.]: Pearson Education, 2010.

MAIA, C.; NOGUEIRA, L.; PINHO, L. M. Evaluating android os for embedded real-time systems. In: *Proceedings of the 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications, Brussels, Belgium*. [S.l.: s.n.], 2010. p. 63–70.

MARTINS, M. R. A. *Integração Sistêmica em middleware*. 233 p. Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais, USP, 2010.

MEIER, R. *Professional Android 4 application development*. [S.l.]: John Wiley & Sons, 2012.

MONGIA, B. S.; MADISETTI, V. K. Reliable real-time applications on android os. *IEEE Electrical and Computer Engineering Electrical and Computer Engineering*, 2010.

NAGATA, K.; YAMAGUCHI, S. An android application launch analyzing system. In: IEEE. *Computing Technology and Information Management (ICCM), 2012 8th International Conference on*. [S.l.], 2012. v. 1, p. 76–81.

NEVES, C.; DUARTE, L.; VIANA, N.; FERREIRA, V. Os dez maiores desafios da automação industrial: As perspectivas para o futuro. In: *II Congresso de Pesquisa e Inovação da Rede Norte Nordeste de Educação Tecnológica, João Pessoa, Paraíba, Brasil*. [S.l.: s.n.], 2007.

OGATA, K.; YANG, Y. *Modern control engineering*. [S.l.]: Prentice-Hall Englewood Cliffs, 2011.

PEREIRA, F. Tecnologia arm-microcontroladores de 32 bits. *São Paulo: Editora Érica*, 2007.

PERNEEL, L.; FAYYAD-KAZAN, H.; TIMMERMAN, M. Can android be used for real-time purposes? In: IEEE. *Computer Systems and Industrial Informatics (ICCSII), 2012 International Conference on*. [S.l.], 2012. p. 1–6.

PETRUZELLA, F. D. *Programmable logic controllers*. [S.l.]: Tata McGraw-Hill Education, 2011.

PRESSMAN, R. S. *Engenharia de software*. [S.l.]: McGraw Hill Brasil, 2011.

QUANSER. *Coupled Tanks – Student Manual*. 2013. Disponível em: <`http://www.quanser.com`>. Acesso em: abr. 2013.

QUANSER. *Coupled Tanks – User Manual*. 2013. Disponível em: <`www.quanser.com`>. Acesso em: abr. 2013.

QUANSER. *VoltPAQX1 – User Manual*. 2013. Disponível em: <`http://www.quanser.com`>. Acesso em: abr. 2013.

RUIMEI, Z.; MEI, W. Design of arm-based embedded ethernet interface. In: IEEE. *Computer Engineering and Technology (ICCET), 2010 2nd International conference on*. [S.l.], 2010. v. 4, p. V4–268.

SISBOT, S. Execution and evaluation of complex industrial automation and control projects using the systems engineering approach. *Systems Engineering*, Wiley Online Library, v. 14, n. 2, p. 193–207, 2011.

SLOSS, A.; SYMES, D.; WRIGHT, C. *ARM System Developer's Guide: Designing and Optimizing System Software*. [S.l.]: Morgan Kaufmann, 2004.

SUDHEER, L. S.; IMMANUEL, J. B.; PARVATHI, C. Arm7 microcontroller based fuzzy logic controller for liquid level control system. *International Journal of Electronics*, 2013.

VALDES-PEREZ, F. E.; PALLAS-ARENY, R. *Microcontrollers: fundamentals and applications with PIC*. [S.l.]: CRC Press, 2009.

VENKATESWARAN, S. *Essential Linux device drivers*. [S.l.]: Prentice Hall Press, 2008.

WANG, L.; TAN, K. C. *Modern industrial automation software design*. [S.l.]: Wiley. com, 2006.

WILMSHURST, T. Designing embedded systems with pic microcontrollers: Principles and applications. 2010.

YAGHMOUR, K. *Embedded Android: Porting, Extending, and Customizing*. [S.l.]: O'Reilly, 2013.

YANG, H.; KEJIAN, L.; QIZHONG, C. Design of arm-based human-machine interface of plastic injection blow molding machine. In: IEEE. *Computer Application and System Modeling (ICCASM), 2010 International Conference on*. [S.l.], 2010. v. 1, p. V1–449.

ZURAWSKI, R. Embedded systems handbook: Embedded systems design and verification. CRC Press, Inc., 2009.

A Instalação do Sistema Operacional Android™ na Placa Tiny6410

A instalação do sistema operacional no kit Tiny6410 pode ser realizada via cabo usb ou pelo cartão SD. Nesta instalação também é possível escolher o local da instalação e execução do sistema, podendo ser feita a partir da memória Nand flash do módulo principal ou do cartão SD. No DVD do kit é disponibilizado um manual com passos para instalação dos sistemas operacionais.

Neste trabalho foi instalado o sistema operacional Android™ na memória *Nand flash* via cartão SD. O cartão foi preparado com software *SD-Flasher* disponível no DVD do Kit para fazer o boot na placa Tiny6410 pelo cartão. A raiz do cartão é composta de um diretório de nome imagens, onde dentro do mesmo contém outro diretório com nome Android e dois arquivos, um de nome FriendlyARM.ini e um binário com nome superboot-6410.bin responsáveis por boot pelo cartão. No arquivo FriendlyARM.ini está a configuração para instalação do sistema. A configuração utilizada neste trabalho é mostrada na Figura A.1 e essa estrutura de arquivo já vem disponível no material do kit também.

Figura A.1: Configurações do arquivo FrendlyARM.ini.

```
#This line cannot be removed. by FriendlyARM(www.arm9.net)

CheckOneButton=No
Action=install
OS= android

VerifyNandWrite=No

StatusType = Beeper| LED

################### Android ###################
Android-BootLoader = superboot-6410.bin
Android-Kernel = Android/azImage_s70
Android-CommandLine = root=ubi0:FriendlyARM-root ubi.mtd=2
rootfstype=ubifs  init=/linuxrc console=ttySAC0,115200
androidboot.console=s3c2410_serial0

Android-RootFs-InstallImage = Android/rootfs_android-mlc2.ubi
Android-RootFs-RunImage = Android/rootfs_android.ext3
```

Para instalação do sistema são necessários três arquivos: o arquivo de bootloader, do Kernel e os arquivos de sistema. Como pode ser visto na Figura A.1 os respectivos arquivos são: superboot-6410.bin, azImage_s70 e o rootfs_android-mlc2.ubi, sendo que os

dois últimos estão localizados no diretório Android do cartão. A placa-mãe possui uma chave que alterna o modo de boot do sistema, que pode ser pelo cartão SD ou pela *Nand flash*. Com o cartão pronto para instalação e inserido no *driver* de cartão, o sistema foi instalado automaticamente ao iniciar a placa com o modo de *boot* pelo cartão. Ao final desses passos a placa ficou pronta para uso e desenvolvimento de aplicações embarcadas no sistema operacional Android™.

B Configuração e Compilação do Kernel Linux e Driver

Para fazer alterações nesse driver utilizou-se um cross-compilador toolschain da Friendly ARM disponível no DVD do kit. Este cross-compilador é responsável por compilar códigos em C/C++ para a arquitetura ARM. Para instalação dessa ferramenta de compilação foi utilizada a seguinte linha de comando em terminal do Ubuntu:

```
$ tar xvzf armlinuxgcc4.5.1v6vfp20101103.tgz C /
```

Após a realização do comando um diretório /opt/FriendlyARM/toolschain /4.5.1/ foi criado. Para finalização da instalação foi configurada a variável de ambiente do sistema operacional Ubuntu. Esta configuração foi feita acrescentando a seguinte linha no arquivo /root/.bashrc do sistema Ubuntu:

```
export PATH=$PATH:/opt/FriendlyARM/toolschain/4.5.1/bin
```

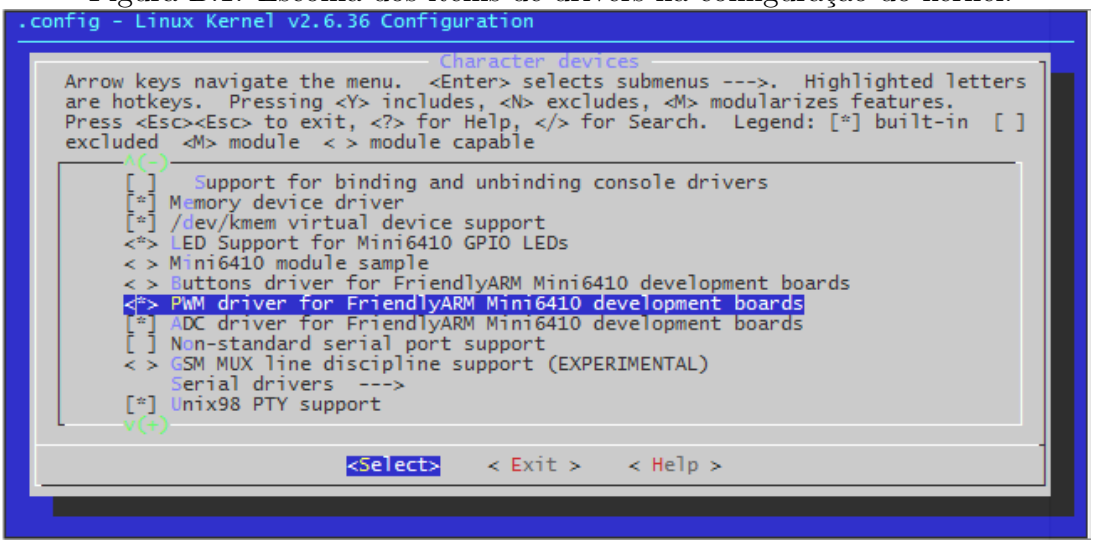
Com o ambiente de cross-compilação pronto para ser usado, o kernel do Linux já pode ser configurado e compilado. O código do Kernel foi descompactado no seguinte diretório de trabalho, /opt/FriendlyARM/mini6410/android/linux-2.6.36-android. A primeira configuração feita foi a da resolução do LCD do Kit tiny6410. Dentro do diretório linux-2.6.36-android já possuem vários arquivos de configurações de LCD. Neste trabalho foi utilizado o arquivo config_android_s70 para configuração de um LCD de 7 polegadas. Para essa configuração foi utilizado a seguinte linha de comando no diretório linux-2.6.36-android:

```
cp config_android_s70 .config
```

Com o comando "make menuconfig" no diretório linux-2.6.36-android é possível visualizar a configuração do kernel na forma gráfica, como mostrada na Figura B.1. Alguns drives disponíveis no kernel para placa Tiny6410 são listados no menu *Character devices*.

Através desse menu é possível selecionar drivers para serem compilados e instalado no kernel. Foram selecionados para este trabalho os *drivers* para acesso de portas seriais, ADC e PWM.

Figura B.1: Escolha dos items de drivers na configuração do kernel.



Antes de realizar a compilação dos *drivers*, foram feitas as alterações necessárias no código do driver PWM. As alterações foram realizadas com o editor gedit do Ubuntu. O código fonte desse *driver* encontra-se no diretório /opt/FriendlyARM/mini6410/android/linux-2.6.36-ndroid/drives/char/mini6410_pwm.c. O drive é escrito em linguagem C e detalhado no Apêndice C deste trabalho. Após essas alterações, com o comando "make modules" também no diretório linux-2.6.36-android, todos os drives selecionados na configuração são compilados. Em seguida com o comando "makezImage" a imagem do kernel é criada no diretório /opt/FriendlyARM/mini6410/android/linux-2.6.36-android/arch/arm/boot e pronta para ser utilizada na instalação do sistema operacional Android™, como já foi descrito no Apêndice A.

C Driver PWM

```
1  #include <linux/module.h>
2  #include <linux/kernel.h>
3  #include <linux/fs.h>
4  #include <linux/init.h>
5  #include <linux/delay.h>
6  #include <linux/poll.h>
7  #include <asm/irq.h>
8  #include <asm/io.h>
9  #include <linux/interrupt.h>
10 #include <asm/uaccess.h>
11 #include <mach/hardware.h>
12 #include <plat/regs-timer.h>
13 #include <mach/regs-irq.h>
14 #include <asm/mach/time.h>
15 #include <linux/clk.h>
16 #include <linux/cdev.h>
17 #include <linux/device.h>
18 #include <linux/miscdevice.h>
19
20 #include <mach/map.h>
21 #include <mach/regs-clock.h>
22 #include <mach/regs-gpio.h>
23
24 #include <plat/gpio-cfg.h>
25 #include <mach/gpio-bank-e.h>
26 #include <mach/gpio-bank-f.h>
27 #include <mach/gpio-bank-k.h>
28
29 #define DEVICE_NAME     "pwm"
30
31 #define PWM_IOCTL_SET_DUTY_CYCLE        3
```

```
32 #define PWM_IOCTL_SET_FREQ              1
33 #define PWM_IOCTL_STOP                  0
34
35 static struct semaphore lock;
36
37 /* freq:  pclk/50/16/65536 ~ pclk/50/16
38   * if pclk = 50MHz, freq is 1Hz to 62500Hz
39   * human ear : 20Hz~ 20000Hz
40   */
41
42 unsigned long freq=1000;
43 unsigned long dutyC=1;
44
45 static void PWM_Set_Freq(void)
46 {
47         unsigned long tcon;
48         unsigned long tcnt;
49         unsigned long tcfg1;
50         unsigned long tcfg0;
51
52         struct clk *clk_p;
53         unsigned long pclk;
54
55         unsigned tmp;
56
57         tmp = readl(S3C64XX_GPFCON);
58         tmp &= ~(0x3U << 28);
59         tmp |=  (0x2U << 28);
60         writel(tmp, S3C64XX_GPFCON);
61
62         tcon = __raw_readl(S3C_TCON);
63         tcfg1 = __raw_readl(S3C_TCFG1);
64         tcfg0 = __raw_readl(S3C_TCFG0);
65
66         //prescaler = 50
```

```
67          tcfg0 &= ~S3C_TCFG_PRESCALER0_MASK;
68          tcfg0 |= (50 - 1);
69
70          //mux = 1/16
71          tcfg1 &= ~S3C_TCFG1_MUX0_MASK;
72          tcfg1 |= S3C_TCFG1_MUX0_DIV16;
73
74          __raw_writel(tcfg1, S3C_TCFG1);
75          __raw_writel(tcfg0, S3C_TCFG0);
76
77          clk_p = clk_get(NULL, "pclk");
78          pclk  = clk_get_rate(clk_p);
79          tcnt  = (pclk/50/16)/freq;// Codigo Original
80
81          //__raw_writel(tcnt, S3C_TCNTB(0));
82          //__raw_writel(tcnt/2, S3C_TCMPB(0));
83
84          __raw_writel(tcnt, S3C_TCNTB(0));
85          __raw_writel((dutyC*tcnt)/100, S3C_TCMPB(0));
86
87          tcon &= ~0x1f;
88          tcon |= 0xb; //disable deadzone,
89                       //auto-reload, inv-off,
90                       //update TCNTB0&TCMPB0, start timer 0
91
92          __raw_writel(tcon, S3C_TCON);
93
94          tcon &= ~2;//clear manual update bit
95          __raw_writel(tcon, S3C_TCON);
96  }
97
98  void PWM_Stop( void )
99  {
100         unsigned tmp;
101         tmp = readl(S3C64XX_GPFCON);
```

```
102         tmp &= ~(0x3U << 28);
103         writel(tmp, S3C64XX_GPFCON);
104 }
105
106 static int s3c64xx_pwm_open(struct inode *inode, struct file *
    file)
107 {
108         if (!down_trylock(&lock))
109                 return 0;
110         else
111                 return -EBUSY;
112 }
113
114 static int s3c64xx_pwm_close(struct inode *inode, struct file *
    file)
115 {
116         up(&lock);
117         return 0;
118 }
119
120 static long
121 s3c64xx_pwm_ioctl(struct file *filep,
122     unsigned int cmd, unsigned long arg)
123 {
124         switch (cmd) {
125                 case PWM_IOCTL_SET_FREQ:
126                         if (arg == 0)
127                                 return -EINVAL;
128                         freq = arg;
129                         PWM_Set_Freq();
130                         break;
131                 case PWM_IOCTL_SET_DUTY_CYCLE:
132                         if (arg == 0)
133                                 return -EINVAL;
134                         dutyC = arg;
```

```
135                         PWM_Set_Freq();
136                         break;
137                 case PWM_IOCTL_STOP:
138                 default:
139                         PWM_Stop();
140                         break;
141         }
142         return 0;
143 }
144
145 static struct file_operations dev_fops = {
146     .owner                      = THIS_MODULE,
147     .open                       = s3c64xx_pwm_open,
148     .release            = s3c64xx_pwm_close,
149     .unlocked_ioctl     = s3c64xx_pwm_ioctl,
150 };
151
152 static struct miscdevice misc = {
153         .minor = MISC_DYNAMIC_MINOR,
154         .name = DEVICE_NAME,
155         .fops = &dev_fops,
156 };
157
158 static int __init dev_init(void)
159 {
160         int ret;
161         init_MUTEX(&lock);
162         ret = misc_register(&misc);
163         printk (DEVICE_NAME"\tinitialized\n");
164         return ret;
165 }
166
167 static void __exit dev_exit(void)
168 {
169         misc_deregister(&misc);
```

```
170 }
171
172 module_init(dev_init);
173 module_exit(dev_exit);
174 MODULE_LICENSE("GPL");
175 MODULE_AUTHOR("FriendlyARM Inc.");
176 MODULE_DESCRIPTION("S3C6410 PWM Driver");
```

D Código Java do controlador PI

```
1  package com.gedea.leveltankcontroller;
2
3  public class Controller {
4
5      double erro_a=0.0, g=981, L10, Aol, Do1=0.3175,
6          Atl, Dt1=4.445, Kdc, tau_t1, zeta, omega, Kp, Ki, P, I
               =0.0, PI, Kff;
7
8      public int PI_Controller(double setPoint, double sensorValue)
           {
9
10         L10 = setPoint;
11
12         Atl = (Math.PI*Math.pow(Dt1,2.0))/4.0;
13
14         Aol = (Math.PI*Math.pow(Do1,2.0))/4.0;
15
16         Kdc = (3.3*(Math.sqrt(g*L10)*(Math.sqrt(2.0))))/(Aol*g);
17
18         tau_t1 = (Math.sqrt(g*L10)*Atl*Math.sqrt(2.0))/(Aol*g);
19
20         zeta = (Math.log(11.0/100.0))/
21         (Math.sqrt(Math.pow((Math.log(11.0/100.0)),2.0)+
22         Math.pow(Math.PI,2.0)));
23
24         if(zeta<0) zeta = -1*zeta;//Calcula valor em módulo;
25
26         Kff = (Aol*Math.sqrt(2.0)*Math.sqrt(g))/3.3;
27
28         omega = 4.0/(zeta*5.0);
29
```

```
30          Kp = ((2.0*zeta*omega*tau_t1)-1.0)/Kdc;
31
32          Ki = (Math.pow(omega, 2.0)*tau_t1)/Kdc;
33
34          double erro=setPoint-sensorValue;
35
36          double Vp_ff = Kff*Math.sqrt(setPoint);
37          double Vp1 = Kp*erro + (erro_a + Ki*erro);
38          double Vp = Vp1 + Vp_ff;
39          erro_a = erro;
40
41          int VpARM = (int) (99*Vp)/21;//conversao para sinal PWM
               0-99
42
43          if(Vp>21)//Limitador
44              return 99;
45          else
46              return VpARM;
47
48      }
49
50  }
```