**IMPLEMENTAÇÃO E PORTABILIDADE DE UM SISTEMA OPERACIONAL DE TEMPO REAL (*RTOS*) NA PLACA *ATMEL SAMR21 XPLAINED PRO*® USANDO REDES *OPENTHREAD®***

**Santa Maria, 10 de dezembro de 2023**

Filipe Alexandre Moraes Eismann [[1]](#footnote-0)

Victor Fidelis dos Reis Santiago [[2]](#footnote-1)

Kalidsa Buzzatti de Oliveira[[3]](#footnote-2)

Matheus Venturini Bortoluzzi[[4]](#footnote-3)

Anthony Perotti Souza[[5]](#footnote-4)

***Resumo:*** *Este trabalho teve como principal objetivo realizar a portabilidade da CLI OpenThread em um sistema operacional de tempo real (RTOS), como o FreeRTOS, em um microcontrolador Atmel SAMR21 Xplained Pro. Foi possível criar uma rede a partir de uma CLI OpenThread usando a portabilidade já existente para a placa SAMR21, além de também executar com RTOS na plataforma Linux. Também foi realizada a tentativa de portar a plataforma OpenThread com o RTOS para a placa e a implementação de sensor de luz TEMT6000 e de temperatura AT30TSE75X disponível na placa Atmel I/O1 Xplained Pro.*

*Palavras-chave: Portabilidade, OpenThread, RTOS, Microchip, SAMR21, iluminância, temperatura.*

# INTRODUÇÃO

Como trabalho final da disciplina de Projeto de Sistemas Embarcados, foi proposta a portabilidade e criação de uma rede OpenThread em um sistema operacional de tempo real (RTOS) na placa Atmel SAMR21 Xplained Pro e a implementação de sensores.

OpenThread é uma plataforma de código aberto que oferece uma implementação do padrão de rede sem fio Thread. Thread é um protocolo de rede de baixa potência e com auto-organização, projetado para conectar dispositivos na Internet das Coisas (IoT).

Um Sistema Operacional de Tempo Real, ou RTOS (Real-Time Operating System), é um sistema operacional projetado para atender a requisitos temporais rigorosos em sistemas embarcados e aplicações em tempo real. Diferentemente dos sistemas operacionais convencionais, que geralmente se concentram em oferecer um ambiente de execução eficiente e amigável para o usuário, um RTOS prioriza a execução de tarefas dentro de prazos específicos. Exemplos de RTOSs incluem FreeRTOS, QNX, VxWorks e RTEMS. Esses sistemas operacionais são amplamente utilizados em aplicações como sistemas embarcados em automóveis, aviões, equipamentos médicos, sistemas de controle industrial e dispositivos eletrônicos de consumo. Nesse trabalho, foi utilizado o FreeRTOS.

A placa Atmel SAMR21 Xplained Pro oferece um ambiente de desenvolvimento conveniente para projetos IoT (Internet das Coisas) e aplicações de rede de sensores sem fio. Ela possui uma variedade de recursos, incluindo conectores de expansão, interfaces de depuração, botões, LEDs e um módulo de comunicação sem fio baseado no protocolo Zigbee, Thread ou outros padrões 802.15.4.

O sensor implementado neste trabalho foi o Atmel I/O1 Xplained Pro, esse sensor foi projetado para oferecer diversas funcionalidades, para as placas Xplained Pro MCU, como, por exemplo, nessa placa possui um cartão microSD, que é utilizado para gravar e ler informações de ou para o sensor, permitindo o armazenamento de dados como configurações, registros ou qualquer informação relevante para o funcionamento do sensor ou do sistema ao qual está conectado. Esses cartões são úteis para armazenar grandes quantidades de dados, como logs, medições ou arquivos de configuração, fornecendo uma forma portátil e de fácil acesso aos dados registrados pelo sensor.

O atmel I/O1 possui também, como funcionalidade, o sensor de luminosidade, basicamente é um fototransistor que possui um resistor em seu coletor (TEMT6000), que é sensível à intensidade luminosa do ambiente, produzindo uma resposta linear. Isto significa que a saída elétrica irá variar proporcionalmente à quantidade de luz incidente, pois a luminosidade interfere diretamente na corrente que passa pelo transistor e gera uma queda de tensão no resistor. Além de sensores de temperatura (AT30TS75A-SS8M-T), dispositivo de alta precisão projetado para medir a temperatura ambiente com uma resolução de 0,5ºC com 9 bits e 0,00625º C com 12 bits, indicando que o sensor consegue identificar variações extremamente pequenas na temperatura.

# DESENVOLVIMENTO

# Portabilidade

Foi criada uma máquina virtual Linux (Lubuntu 23.10) para desenvolvimento em ambientes equivalentes entre os alunos. Como estudo inicial, foi compilada e executada a CLI do OpenThread no microcontrolador, além da criação de uma rede com 1 *commissioner* e 2 *joiners*, com sucesso. Foi testada a comunicação através da utilização de comandos ping entre um comissioner e um joiner, além do envio de uma mensagem dos joiner para o comissioner, que ocorreu corretamente. Houveram algumas dificuldades devido à dependências de pacotes, o que levou inicialmente à necessidade de usar um script *Python* para acessar a porta USB da placa a fim de realizar a gravação da imagem do OpenThread no microcontrolador. Também houveram dificuldades para testar se o OpenThread havia sido corretamente instalado nas placas pois a ferramenta *screen* e *PuTTY* não funcionaram corretamente no Linux, sendo necessário utilizar o SO Windows 10 para fazer a verificação. Apesar disso, foi compilada com sucesso a CLI do OpenThread em uma máquina virtual Linux, porém, a intercomunicação entre instâncias do OpenThread não foi possível devido à limitação do sistema de virtualização (VirtualBox).

Após os testes realizados com as versões anteriores, foi iniciado o processo de portabilidade da plataforma para o microchip da Atmel. No processo de criação dos arquivos, foram necessárias diversas alterações nos scripts CMake, o que gerou diversos problemas e conflitos de caminhos no momento da construção. Também houveram problemas devido ao espaço pequeno de disco disponível para as máquinas virtuais utilizadas para o desenvolvimento no laboratório.

Assim, foi feita uma tentativa de portar manualmente os arquivos da camada de abstração de software da placa por meio da portabilidade já existente ².

# Sensor

# I/O1 Xplained Pro - Sensor de luz ambiente (TEMT6000)

Como mencionado anteriormente, o sensor de luz ambiente consiste em um fototransistor que possui um resistor no seu coletor, a corrente que passa por este transistor irá variar de acordo com a luminosidade do local. A luminosidade provoca uma queda de tensão no resistor do coletor e essa queda de tensão que deverá ser lida e convertida pelo microcontrolador.

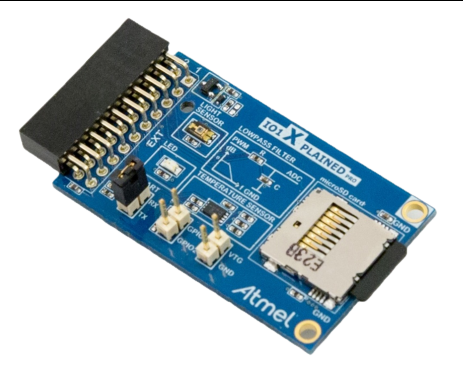


Figura 1 - Atmel I/O1 Xplained Pro

A tensão de operação da placa varia de 3 a 3,6V e a corrente máxima é de 10mA, como apresentado na Figura 2, e o conector de alimentação, do Atmel SAMR21, utilizado para conectar a placa do sensor possui como tensão limite de 3,3V. Então, neste trabalho será aplicada esse valor de tensão para os cálculos.

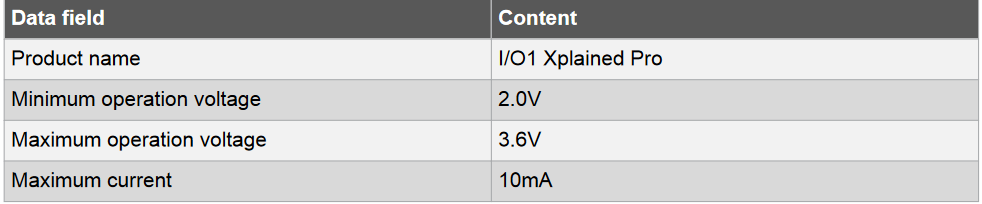


Figura 2 - Características elétricas do I/O1 Xplained Pro

Como é informado no Guia do Usuário do Atmel I/O1 Xplained Pro, as linhas de seleção de endereço passam por resistores de 100kΩ, logo utiliza-se esse valor para realizar os cálculos de corrente no sensor. Com base nessas informações, foi desenvolvida uma função denominada readLightSensor() que irá realizar os cálculos para obter a luminosidade do ambiente na unidade lux. Geralmente, sensores de luz não fornecem leituras diretamente relacionadas à iluminância, porém a relação entre a corrente e a tensão, gerada pelo sensor, é diretamente proporcional à intensidade de luz incidente na área do sensor, utiliza-se estes parâmetros para obter a iluminância em lux.

A implementação para o sensor I/O1 se iniciou através do Atmel Start, onde foi realizado o download dos arquivos direcionados a placa SAMD21, Figura 3, para desenvolver o sensor de temperatura e o de luz, e a adaptação dos códigos para a placa utilizada neste trabalho, SAMR21.

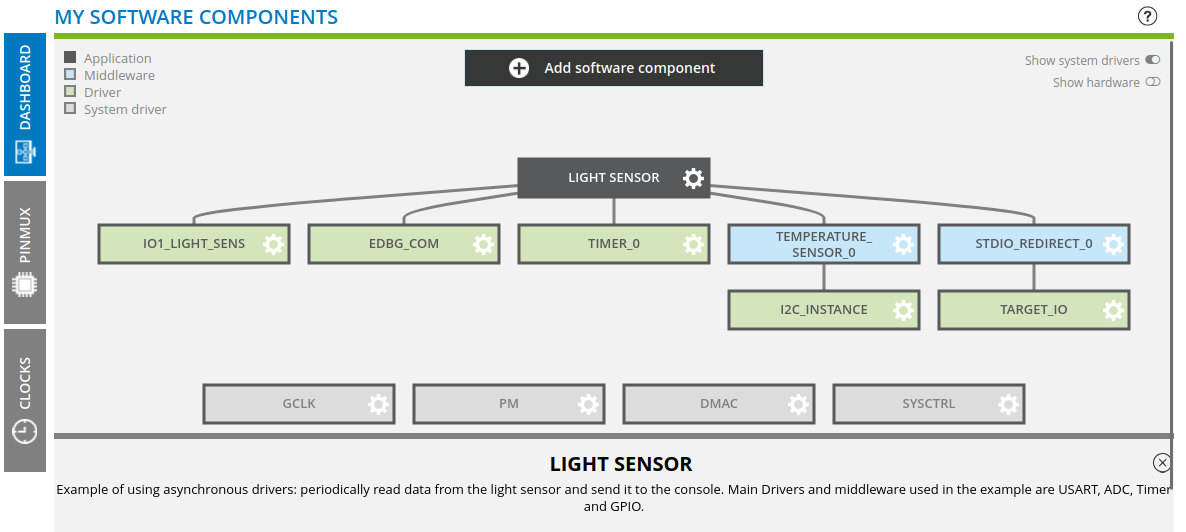


Figura 3 - Ambiente IDE SAMD21

Através dos códigos obtidos, desenvolvemos o driver para a placa I/O1, denominado IO1\_drivers.c, onde consta a função readLightSensor() e readTemperatureSensor(), Figura 5, que são as funções de interesse, onde através da readVoltageSensor() é realizada a conversão analógica-digital pela função adc\_async\_start\_conversion no canal do sensor de luz e armazena o valor digitalizado utilizando a função adc\_async\_read\_channel na variável IO\_SENSOR\_VALUE, após realiza-se a conversão deste valor obtido para uma tensão utilizando o valor da tensão da placa, de 3.3V em VCC\_TARGET e dividindo o produto por 255, ou seja, é considerado uma resolução de 8 bits, com valores variando de 0 a 255. A fórmula utilizada é:

Após obter o valor aproximado de tensão que é passado como parâmentro para a função readCurrentSensor() que realiza o cálculo do valor da corrente, em micro âmpere, que passa pelo sensor pela fórmula abaixo, onde é aplicada a lei de ohm usando o valor de 100kΩ do resistor da linha de seleção.

Para obter a iluminância, na função readLightSensor(), aplica-se a seguinte equação:

Através do datasheet do sensor TEMT6000, é especificada uma correlação proporcional entre corrente e luminância, para cada 2μA de corrente correlaciona-se com 1 lx de iluminância. Logo, como o alcance deste sensor é de 10 a 1000 lx, para os cálculos foi escolhido o valor de 10 lx multiplicado por 2, para 1μA. Na implementação do readTemperatureSensor() todo o condicionamento do sinal é realizado pelos componentes automaticamente através do código adquirido pelo Atmel Start.

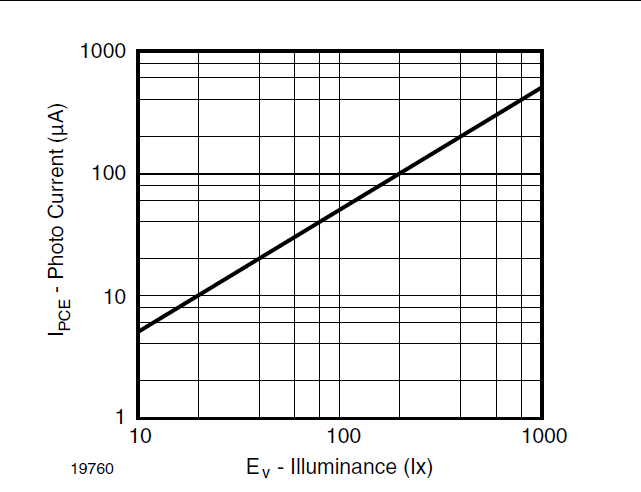


Figura 4 - Relação entre corrente do fototransistor e iluminância

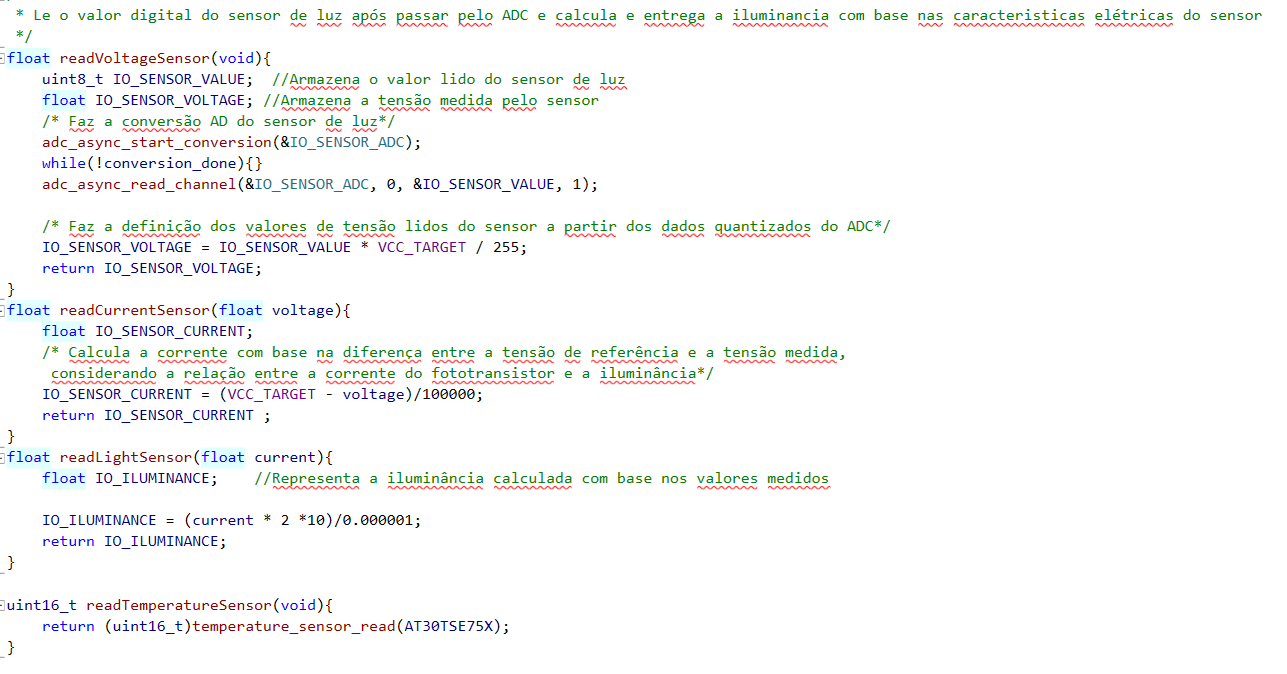


Figura 5 - Funções para obter a tensão, corrente e iluminância no sensor de TEMT6000 e temperatura do AT30TXE75X.

Anteriormente, para o mesmo código de IO1\_driver.c, é definida uma função denominada lightSensor\_ADC\_conversion\_callback que é acionada quando a conversão analógico-digital do sensor de luz for concluída, definindo conversion\_done como true. Após ocorre a inicialização dos parâmetros do sensor de luz, pela função IO\_SENSOR\_INIT, através da função adc\_async\_register\_callback, que indica que quando a conversão analógico-digital do canal 0 do ADC for concluída a função lightSensor\_ADC\_conversion\_callback será chamada, após é habilitado o canal 0 do ADC para a leitura do sensor de luz, adc\_async\_start\_conversion inicia a conversão, a inicialização dos parâmetros de comunicação USART (Transmissão Serial Universal Assíncrona/Síncrona) que permite a comunicação do Atmel SAMR21 com a placa I/O1 Xplained Pro no canal TARGETIO\_DEBUG, sendo o descritor de entrada e saída. E por último I2C, que habilita o periférico de comunicação na instância I2C para configurar os parâmetros necessários para o sensor de temperatura.

Já a função sendByteUART() serve para enviar informações específicas ou byte de dados para a UART de debug, ou seja, Essa UART é designada para enviar mensagens de status, mensagens de erro, informações de depuração e outros dados úteis que ajudam os desenvolvedores a entender o comportamento do sistema ou a solucionar problemas.

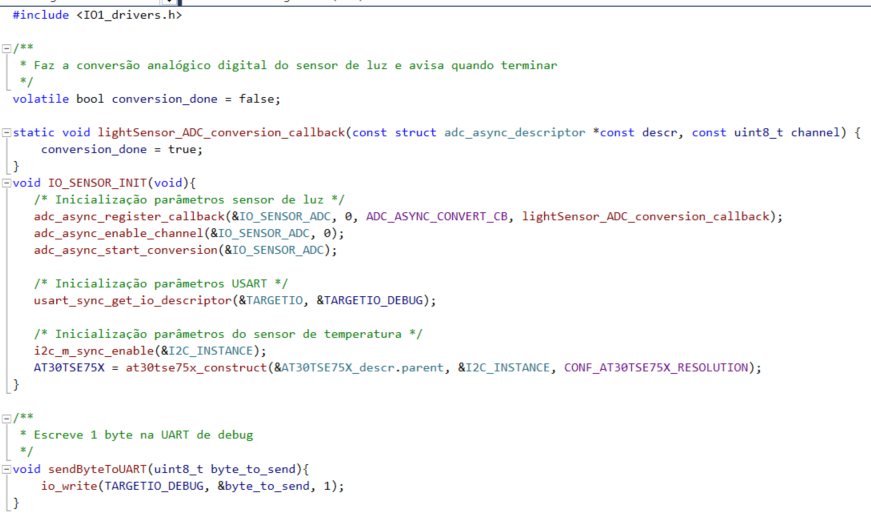


Figura 6 - Inicialização ADC, USART e I2C.

Ainda em IO1\_driver.c é implementada uma função que serve para converter um número float em string com a precisão informada no terceiro parâmetro, essa função é utilizada para apresentar os valores float na tela para o usuário usando o sprintf na main, pois o compilador não aceita float no sprintf.

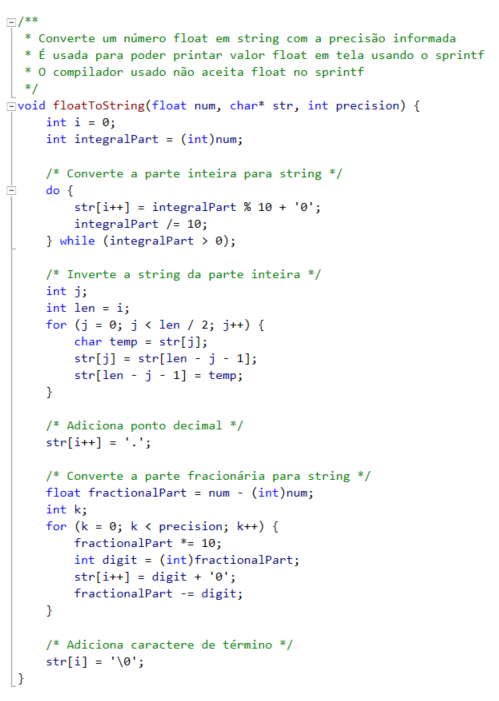


Figura 7 - Função de conversão de float em String

O cabeçalho (\_IO1\_drivers) é um arquivo de cabeçalho em C, usado para definir as declarações de funções, constantes, estruturas e configurações necessárias para o funcionamento de drivers e dispositivos específicos. Nesse arquivo foi definida a variável VCC\_TARGET que carrega o valor de 3.3V e as estruturas necessárias para o funcionamento do sensor de temperatura AT30TSE75X.

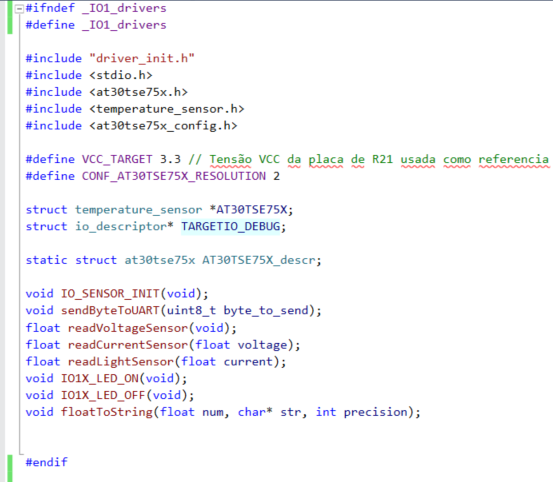
****

Figura 8 - Código para IO1\_drivers.h

Além do IO1\_drivers.c e IO1\_drivers.h, foi desenvolvida também a função principal main() onde são iniciados o microcontrolador, os drivers, dos periféricos, entre outros, pela função fornecida, através do framework Atmel Start, atmel\_start\_init e a inicialização do sensor Atmel I/O1 pelo IO\_SENSOR\_INIT. No while é chamada a função que liga o LED do I/O1e após é feita a leitura da tensão, corrente, iluminância e temperatura obtidos pelo sensor. Cada valor é apresentado na tela para o usuário.

Outros arquivos importantes são os at30tse75x e temperature\_sensor, obtidos pelo Atmel Start e configurado do SAMD21 para o SAMR21, nos arquivos driver\_init, que fazem as configurações e inicializações necessárias para que o sensor I/O1 se conecte com o SAMR21 e funcione, foram realizadas pequenas alterações para a mudança de placa.

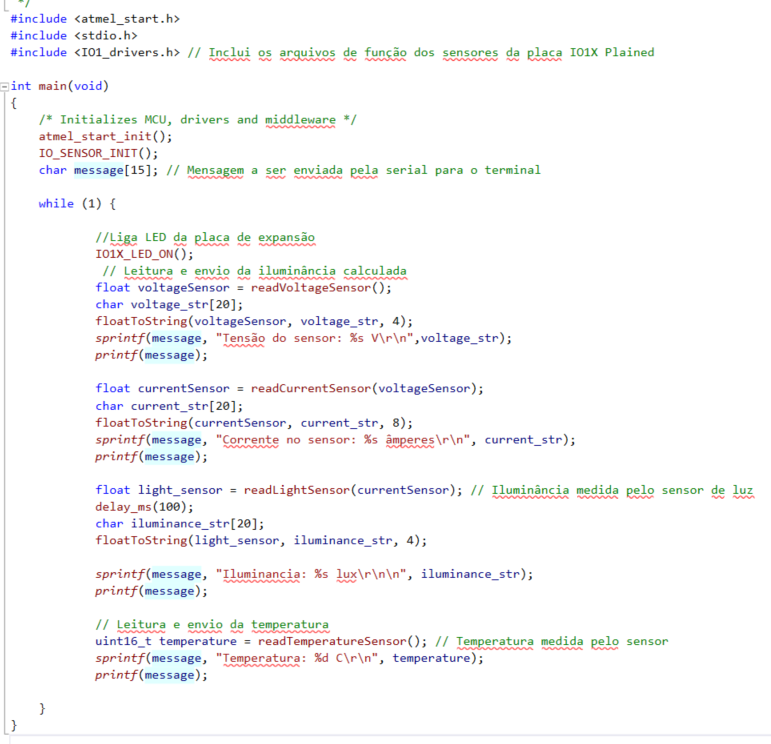
****

Figura 9 - Código para a main.c

Para a conexão do Atmel I/O1 com o SAM R21, utilizou-se o cabeçalho de extensão (Extension Headers) EXT1, Figura 10, que oferece acesso a E/S do microcontrolador para expandir as funcionalidades da placa. É configurado para o pino PA06, como entrada de sinal analógico do sensor de iluminância, para o uso com o ADC, que conecta ao pino 3 da placa I/O1, Figura 11, os pinos PA04 e PA05 para o uso do USART, nos pinos 13 e 14, respectivamente da placa I/O1 e os pinos PA16, sendo a linha de dados serial, e PA17, linha de clock serial, para o uso com I2C que necessita de dois pinos para realizar a transferência de dados.

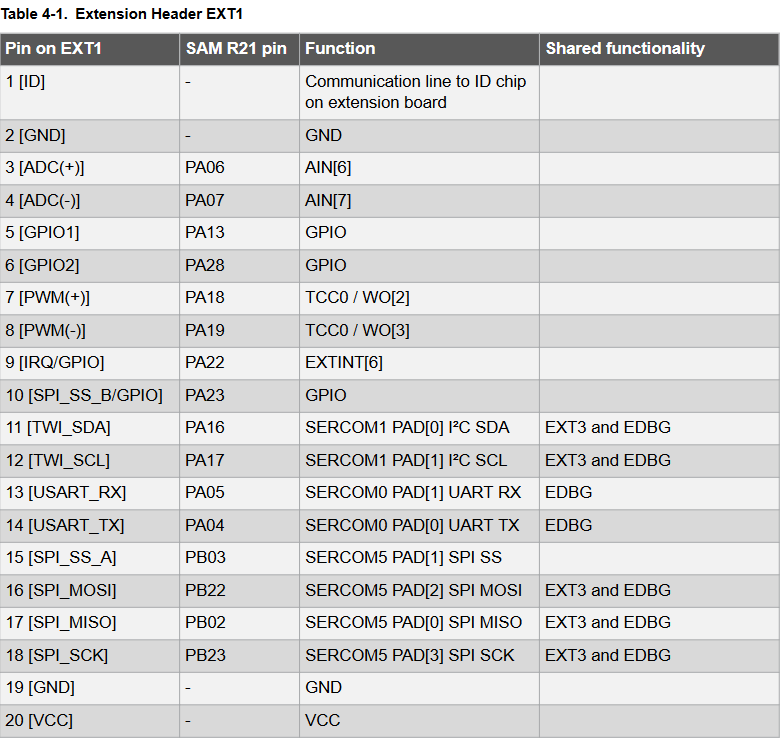
****

Figura 10 - Pinagem do cabeçalho de extensão da placa SAM R21

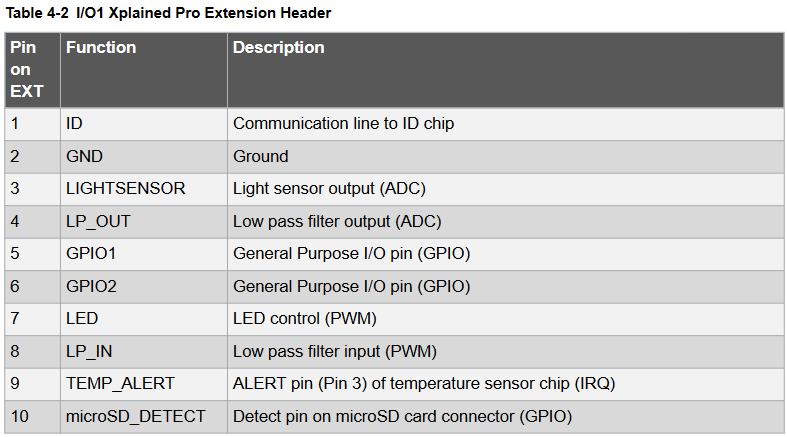
****

Figura 11 - Pinagem do cabeçalho de extensão da placa I/O1

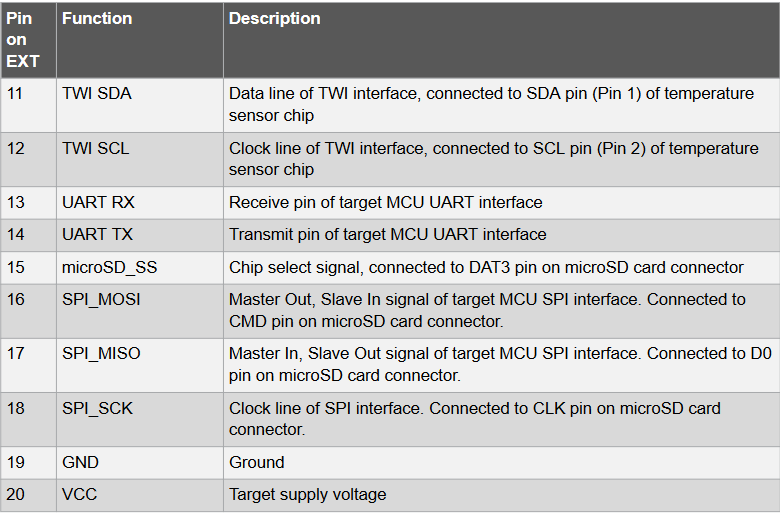


Figura 12 - Continuação da tabela de pinagem do cabeçalho de extensão da placa I/O1

# Documentação

O arquivo contém os seguintes passos e códigos:

# Documentação da portabilidade:

A documentação da portabilidade foi criada em formato passo a passo em um arquivo markdown nomeado por “readme\_trabalho.md” contendo as etapas realizadas para o início da portabilidade da placa SAM-R21 para ot-rtos openthread.

* + 1. **OpenThread Network Simulator (OTNS):**
       1. **Instalação do Go:**

O Go pode ser instalado pelo link

https://golang.org/dl/

* + - 1. **Openthread network simulator:**

Siga até o passo 3 do https://openthread.io/codelabs/openthread-network-simulator#0

* + - 1. **Criar openthread para OTNS:**

A partir do passo 3 do <https://openthread.io/codelabs/openthread-network-simulator?hl=pt-br#2>

Para compilar o Openthread, utilizar, também a flag “-DOT\_OTNS=1” para que o OTNS detecte a instalação do Openthread na máquina.

Comando completo:

*$ ./script/cmake-build simulation -DOT\_OTNS=ON -DOT\_SIMULATION\_VIRTUAL\_TIME=ON -DOT\_SIMULATION\_VIRTUAL\_TIME\_UART=ON -DOT\_SIMULATION\_MAX\_NETWORK\_SIZE=999*.

* + - 1. **Após a clonagem do repositório OTNS:**

Rodar o seguinte script na pasta “script”:

$ *install-dep*

*$ install*

* + - 1. **Abrindo o browser com o simulador:**

Rode o OTNS no terminal da pasta /src/openthread/build/simulation/examples/apps/cli

# Portabilidade Ot-Rtos para placa SAMR21

# Compilação e Flash do OpenThread para a Placa Atmel SAMR21

* + - 1. **Baixar GNU Toolchain:**

$ cd <path-to-ot-samr21>

$ ./script/bootstrap

* + - 1. **Build:**

$ cd <path-to-ot-samr21>

$ ./script/build

* + - 1. **Flash Binary:**

Após o build bem-sucedido:

$ openocd -f board/atmel\_samr21\_xplained\_pro.cfg

Em outro terminal:

$ cd <path-to-ot-samr21>/build/bin

$ arm-none-eabi-gdb ot-cli-ftd

$ (gdb) target remote 127.0.0.1:3333

$ (gdb) load

$ (gdb) monitor reset

$ (gdb) c

* + - 1. **Formar uma Rede Thread no Thread Border Router:**

Siga a partir do Passo 4 - [openthread.io/codelabs/esp-openthread-hardware#2](https://openthread.io/codelabs/esp-openthread-hardware?hl=pt-br#3)

* + - 1. **Linux Simulation:**

$ git submodule update --init

$ mkdir build && cd build

$ cmake .. -DPLATFORM\_NAME=linux

$ make -j12

Isso compilará a aplicação de teste CLI em build/ot\_cli\_linux.

* + - 1. **Compilação do OT-RTOS na Placa Nordic nRF52840:**

$ git submodule update --init

$ mkdir build && cd build

$ cmake .. -DCMAKE\_TOOLCHAIN\_FILE=../cmake/arm-none-eabi.cmake -DPLATFORM\_NAME=nrf52

$ make -j12

Isso construirá a aplicação de teste CLI em build/ot\_cli\_nrf52840.hex.

* + - 1. **Configurar o RCP Joiner:**

$ cd ~/src

$ git clone --recursive https://github.com/openthread/ot-nrf528xx.git

$ cd ot-nrf528xx

$ script/build nrf52840 USB\_trans

* + - 1. **Compilação do código para a placa SAMR21:**

$ git submodule update --init

$ mkdir build && cd build

$ cmake .. -DCMAKE\_TOOLCHAIN\_FILE=../cmake/arm-none-eabi.cmake -DPLATFORM\_NAME=samr21

$ make -j12

* + - 1. **Modificação no CMakeLists.txt da Raíz do OT-RTOS:**

No arquivo CMakelists.text, substitua o bloco condicional para a plataforma Nordic pelo seguinte para a plataforma SAMR21:

if (${PLATFORM\_NAME} STREQUAL samr21)

add\_executable(ot\_demo\_101

${SRC\_DIR}/apps/cli/main.c

${SRC\_DIR}/core/io\_redirect.c

)

target\_link\_libraries(ot\_demo\_101

PUBLIC

demo\_101

)

target\_compile\_definitions(ot\_demo\_101

PUBLIC

\_\_HEAP\_SIZE=8192

\_\_STACK\_SIZE=8192

)

target\_compile\_definitions(ot\_cli\_${PLATFORM\_NAME}

PUBLIC

\_\_HEAP\_SIZE=8192

\_\_STACK\_SIZE=8192

)

#special link script

set\_target\_properties(ot\_cli\_${PLATFORM\_NAME} PROPERTIES LINK\_DEPENDS ${CMAKE\_CURRENT\_SOURCE\_DIR}/third\_party/openthread/repo/microship/include/samr21x18a.ld)

set\_target\_properties(ot\_cli\_${PLATFORM\_NAME} PROPERTIES LINK\_FLAGS "-T ${CMAKE\_CURRENT\_SOURCE\_DIR}/third\_party/openthread/repo/microship/include/samr21x18a.ld -lc -lnosys -lm -lstdc++")

set\_target\_properties(ot\_demo\_101 PROPERTIES LINK\_DEPENDS ${CMAKE\_CURRENT\_SOURCE\_DIR}/third\_party/openthread/repo/microship/include/samr21x18a.ld)

set\_target\_properties(ot\_demo\_101 PROPERTIES LINK\_FLAGS "-T ${CMAKE\_CURRENT\_SOURCE\_DIR}/third\_party/openthread/repo/microship/include/samr21x18a.ld -lc -lnosys -lm -lstdc++")

#build hex file

add\_custom\_command(OUTPUT ot\_cli\_${PLATFORM\_NAME}.hex

COMMAND arm-none-eabi-objcopy -O ihex ot\_cli\_${PLATFORM\_NAME} ot\_cli\_${PLATFORM\_NAME}.hex

DEPENDS ot\_cli\_${PLATFORM\_NAME}

)

add\_custom\_target(ot\_cli\_nrf52\_hex ALL DEPENDS ot\_cli\_${PLATFORM\_NAME}.hex)

add\_custom\_command(OUTPUT ot\_demo\_101.hex

COMMAND arm-none-eabi-objcopy -O ihex ot\_demo\_101 ot\_demo\_101.hex

DEPENDS ot\_demo\_101

)

add\_custom\_target(ot\_demo\_101\_hex ALL DEPENDS ot\_demo\_101.hex)

endif()

* + - 1. **Adicionar o FreeRTOSConfig.h:**

Adicione o arquivo FreeRTOSConfig.h da pasta "freertos\_portable" do OT-RTOS a pasta "third-party".

* + - 1. **Adicionar o Arquivo cli.h:**

Inclua o arquivo cli.h da pasta "openthread" do Openthread à pasta "third-party".

* 1. **Documentação Sensor de Luzes:**

A documentação do sensor de luzes seguiu os seguintes passos:

* + 1. **Comentários de funções @brief e comentários de parâmetros de funções com @param**

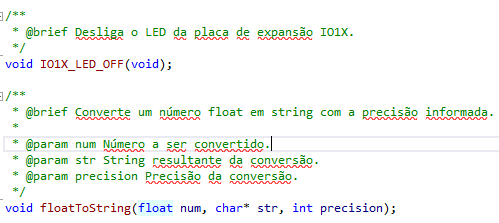


Figura 13 - Comentários de funções e parâmetros

* + 1. **Comentários de estruturas e definições no formato “/\*\*<comentário \*/”**

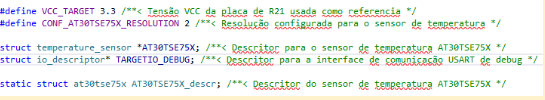


Figura 14 - Comentários de estruturas

* + 1. **Comentários de cabeçalhos de arquivos @file**

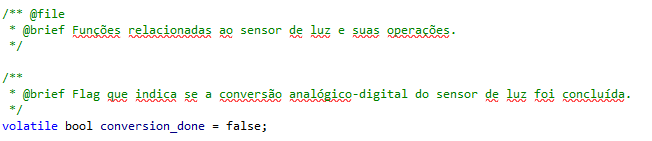


Figura 15 - Comentários de cabeçalho de arquivos

Após os comentários, foi utilizado o programa “Doxywizard” para gerar os arquivos html e pdf da documentação do sensor de luzes:

* + 1. **Configurações de parâmetros para criação da documentação**

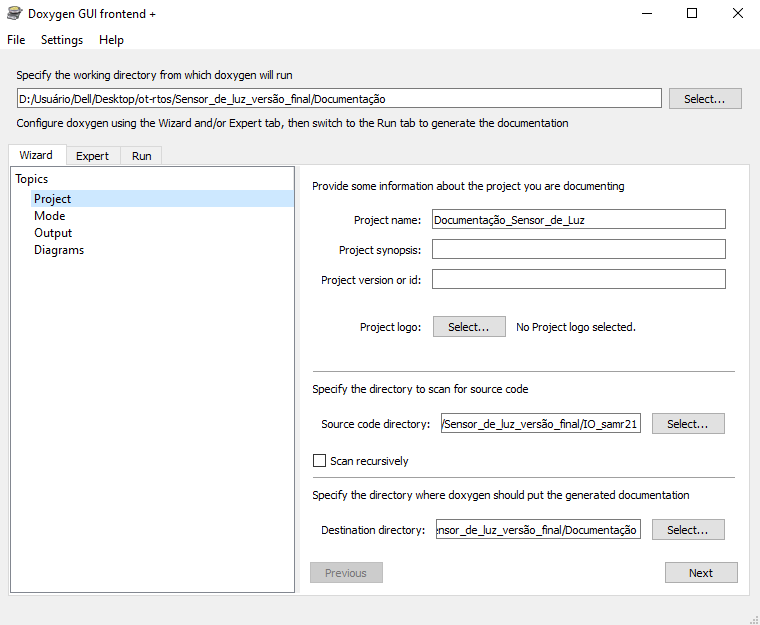
****

Figura 16 - Parâmetros do “doxywizard”

* + 1. **Programa “Doxywizard” gerando html a partir dos arquivos em “.c” encontrados na pasta selecionada**

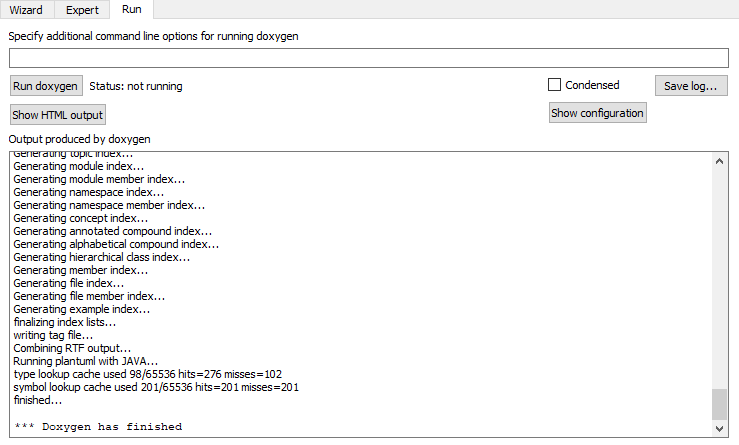
****

Figura 17 - Geração do html pelo doxywizard

* + 1. **Html gerado pelo programa contendo todos os arquivos encontrados com programação “.c” juntamente com seus devidos comentários anteriormente expostos**

****

Figura 18 - Html gerado pelo doxywizard

* + 1. **Criação do pdf a partir de um arquivo selecionado, no caso deste trabalho foram selecionados apenas os arquivos modificados e programados**

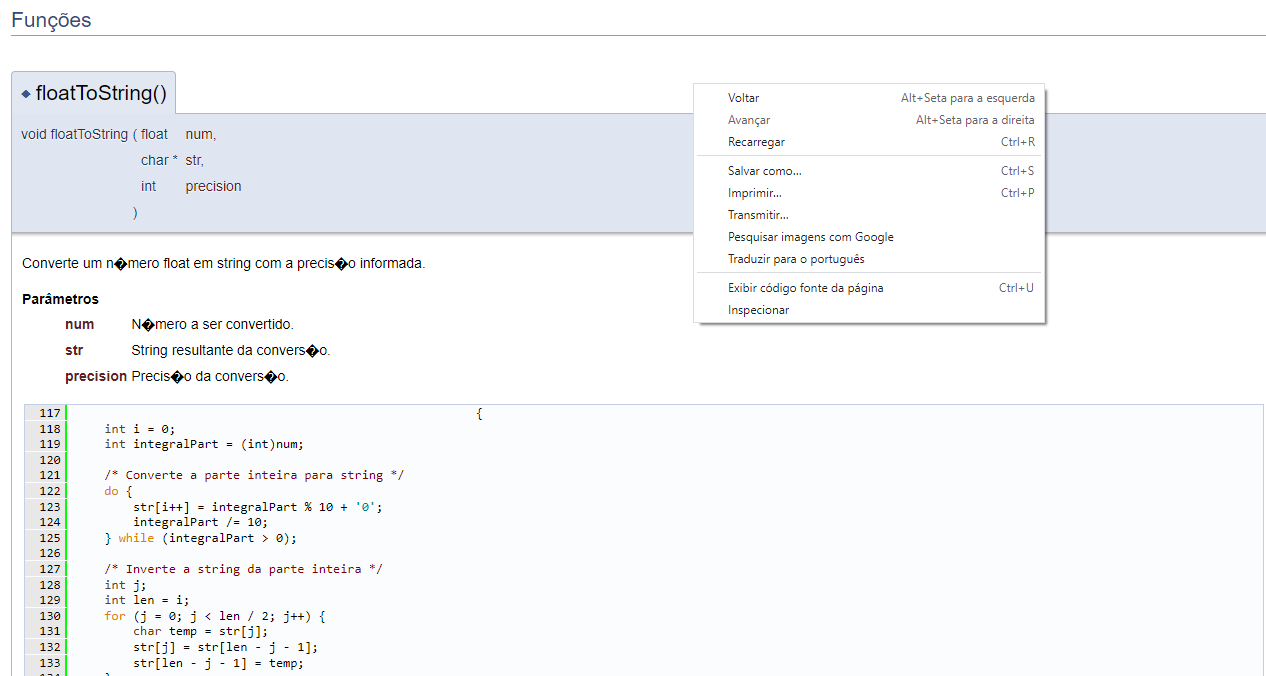
****

Figura 19 - Visualização de arquivo no html gerado pelo doxywizard

* + 1. **Pasta com a documentação após a junção dos pdfs gerados na etapa anterior, junto com as pastas “html”, “rtf” e o arquivo “Doxyfile” gerados pelo programa “Doxywizard”.**

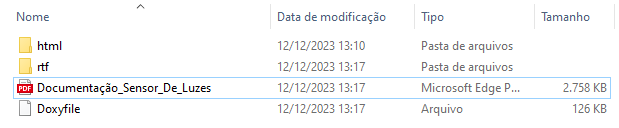
****

Figura 20 - Pasta contendo arquivos gerados da documentação do sensor de luzes

**CONCLUSÃO**

Como estudo principal, foi desenvolvido o ambiente virtual Linux necessário para a compilação e execução da portabilidade, para assim facilitar o desenvolvimento futuro dos projetos envolvendo a placa SAMR21, a plataforma OpenThread e sistemas operacionais de tempo real.

Após as tentativas de portabilidade, não foi possível criar a rede OpenThread usando RTOS, devido a complicações de dependências de pacotes, caminhos divergentes nos arquivos CMake, formas de compilação diferentes e problemas de memória na máquina virtual. Também não foi possível realizar o funcionamento do sensor I/O1 Xplained Pro no trabalho “ot-samr21”, apenas através do Microchip Studio pelo Windows.

# REFERÊNCIAS

1. Repositório OT-RTOS - <https://github.com/ufsm-barriquello/ot-rtos>
2. Repositório OT-SAMR21 - <https://github.com/ufsm-barriquello/ot-samr21>
3. OpenThread - <https://openthread.io/>
4. OpenThread - Build a Thread Network - <https://openthread.io/codelabs/esp-openthread-hardware#4>
5. OpenThread - Porting - <https://openthread.io/guides/porting>
6. FreeRTOS - <https://www.freertos.org/index.html>
7. Atmel SAMR21 Datasheet - <https://ww1.microchip.com/downloads/en/devicedoc/sam-r21_datasheet.pdf>
8. Atmel SAMR21 Xplained Pro User Guide - <https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42243-SAMR21-Xplained-Pro_User-Guide.pdf>
9. Atmel I/O1 Xplained Pro User Guide <https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42078-IO1-Xplained-Pro_User-Guide.pdf>

## Generate documentation from source code -

<https://www.doxygen.nl/>

1. Acadêmico do curso de Engenharia de Computação da Universidade Federal De Santa Maria - UFSM, matrícula: 2021520042, e-mail: filipe.eismann@ecomp.ufsm.br [↑](#footnote-ref-0)
2. Acadêmico do curso de Engenharia de Computação da Universidade Federal De Santa Maria - UFSM, matrícula: 201920852, e-mail: victorfdrsanti@gmail.com [↑](#footnote-ref-1)
3. Acadêmica do curso de Engenharia de Computação da Universidade Federal De Santa Maria - UFSM, matrícula: 2018520184, e-mail: kalidsa.oliveira@ecomp.ufsm.br [↑](#footnote-ref-2)
4. Acadêmico do curso de Engenharia de Computação da Universidade Federal De Santa Maria - UFSM, matrícula: 202020181, e-mail: matheus.venturini@acad.ufsm.br [↑](#footnote-ref-3)
5. Acadêmico do curso de Engenharia de Computação da Universidade Federal De Santa Maria - UFSM, matrícula: 202020815, e-mail: anthony.souza@acad.ufsm.br [↑](#footnote-ref-4)