CENTRO DE TECNOLOGIA - UFSM

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]
Victor Fidelis dos Reis Santiago [2]
Kalidsa Buzzatti de Oliveira[3]
Matheus Venturini Bortoluzzi[4]
Anthony Perotti Souza[5]

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.

[1] Acadêmico do curso de Engenharia de Computação da Universidade Federal De Santa Maria - UFSM, matrícula: 2021520042, e-mail: filipe.eismann@ecomp.ufsm.br

[2] Acadêmico do curso de Engenharia de Computação da Universidade Federal De Santa Maria - UFSM, matrícula: 201920852, e-mail: victorfdrsanti@gmail.com

[3] Acadêmica do curso de Engenharia de Computação da Universidade Federal De Santa Maria - UFSM, matrícula: 2018520184, e-mail: kalidsa.oliveira@ecomp.ufsm.br

[4] Acadêmico do curso de Engenharia de Computação da Universidade Federal De Santa Maria - UFSM, matrícula: 202020181, e-mail: matheus.venturini@acad.ufsm.br

[5] Acadêmico do curso de Engenharia de Computação da Universidade Federal De Santa Maria - UFSM, matrícula: 202020815, e-mail: anthony.souza@acad.ufsm.br

1. 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.

2. DESENVOLVIMENTO

2.1. 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].

2.2. Sensor

2.2.1. 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.

Data field	Content
Product name	I/O1 Xplained Pro
Minimum operation voltage	2.0V
Maximum operation voltage	3.6V
Maximum current	10mA

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.

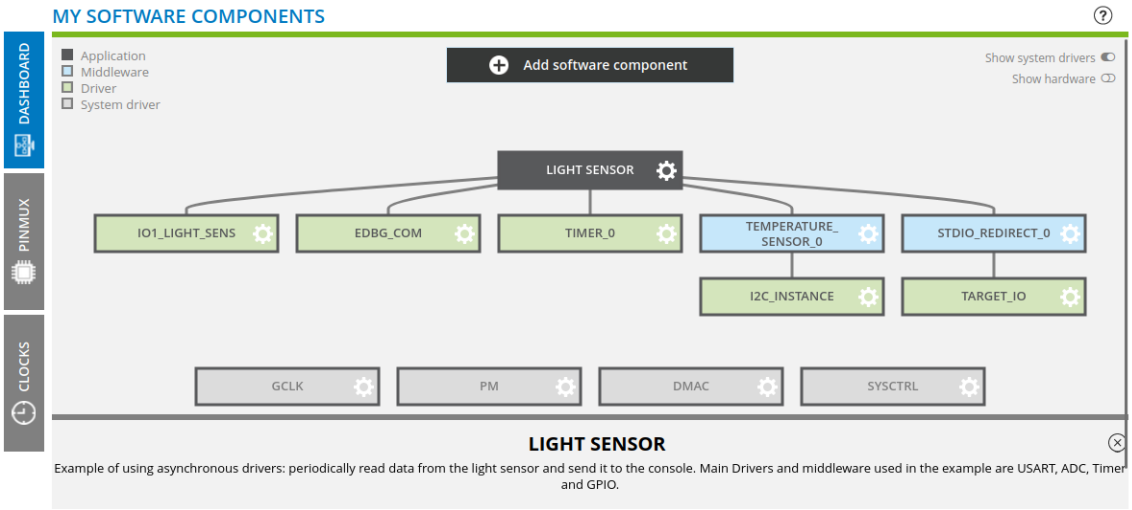

Figura 3 - Ambiente IDE SAMD21

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:

$$IO\_SENSOR\_VOLTAGE = \frac{IO\_SENSOR\_VALUE * VCC\_TARGET}{255}$$

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.

$$IO\_SENSOR\_CURRENT = \frac{(VCC\_TARGET - IO\_SENSOR\_VOLTAGE)}{100000}$$

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

$$Luminosidade = \frac{Ic*2*lux}{Ipce\ \mu A}$$

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.

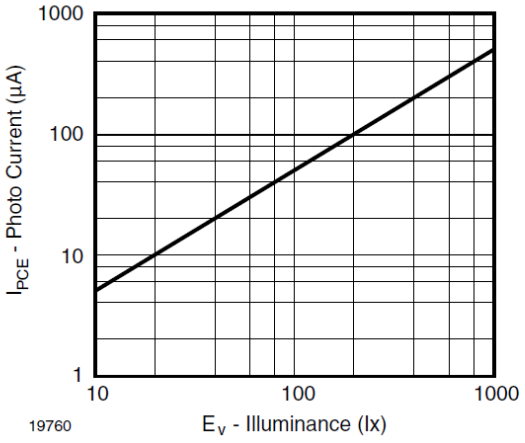

Figura 4 - Relação entre corrente do fototransistor e iluminância

```
 * Le o valor digital do sensor de luz após passar pelo ADC e calcula e entrega a iluminancia com base nas caracteristicas elétricas do sensor
 */
float readVoltageSensor(void){
    uint8_t IO_SENSOR_VALUE;  //Armazena o valor lido do sensor de luz
    float IO_SENSOR_VOLTAGE; //Armazena a tensão medida pelo sensor
    /* Faz a conversão AD do sensor de luz*/
    adc_async_start_conversion(&IO_SENSOR_ADC);
    while(!conversion_done){}
    adc_async_read_channel(&IO_SENSOR_ADC, 0, &IO_SENSOR_VALUE, 1);

    /* Faz a definição dos valores de tensão lidos do sensor a partir dos dados quantizados do ADC*/
    IO_SENSOR_VOLTAGE = IO_SENSOR_VALUE * VCC_TARGET / 255;
    return IO_SENSOR_VOLTAGE;
}
float readCurrentSensor(float voltage){
    float IO_SENSOR_CURRENT;
    /* Calcula a corrente com base na diferença entre a tensão de referência e a tensão medida,
     considerando a relação entre a corrente do fototransistor e a iluminância*/
    IO_SENSOR_CURRENT = (VCC_TARGET - voltage)/100000;
    return IO_SENSOR_CURRENT ;
}
float readLightSensor(float current){
    float IO_ILUMINANCE;    //Representa a iluminância calculada com base nos valores medidos

    IO_ILUMINANCE = (current * 2 *10)/0.000001;
    return IO_ILUMINANCE;
}

uint16_t readTemperatureSensor(void){
    return (uint16_t)temperature_sensor_read(AT30TSE75X);
}
```

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.

```c
#include <IO1_drivers.h>

/**
 * Faz a conversão analógico digital do sensor de luz e avisa quando terminar
 */
volatile bool conversion_done = false;

static void lightSensor_ADC_conversion_callback(const struct adc_async_descriptor *const descr, const uint8_t channel) {
    conversion_done = true;
}
void IO_SENSOR_INIT(void){
    /* Inicialização parâmetros sensor de luz */
    adc_async_register_callback(&IO_SENSOR_ADC, 0, ADC_ASYNC_CONVERT_CB, lightSensor_ADC_conversion_callback);
    adc_async_enable_channel(&IO_SENSOR_ADC, 0);
    adc_async_start_conversion(&IO_SENSOR_ADC);

    /* Inicialização parâmetros USART */
    usart_sync_get_io_descriptor(&TARGETIO, &TARGETIO_DEBUG);

    /* Inicialização parâmetros do sensor de temperatura */
    i2c_m_sync_enable(&I2C_INSTANCE);
    AT30TSE75X = at30tse75x_construct(&AT30TSE75X_descr.parent, &I2C_INSTANCE, CONF_AT30TSE75X_RESOLUTION);
}

/**
 * Escreve 1 byte na UART de debug
 */
void sendByteToUART(uint8_t byte_to_send){
    io_write(TARGETIO_DEBUG, &byte_to_send, 1);
}
```

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.

```c
/**
 * Converte um número float em string com a precisão informada
 * É usada para poder printar valor float em tela usando o sprintf
 * O compilador usado não aceita float no sprintf
 */
void floatToString(float num, char* str, int precision) {
    int i = 0;
    int integralPart = (int)num;

    /* Converte a parte inteira para string */
    do {
        str[i++] = integralPart % 10 + '0';
        integralPart /= 10;
    } while (integralPart > 0);

    /* Inverte a string da parte inteira */
    int j;
    int len = i;
    for (j = 0; j < len / 2; j++) {
        char temp = str[j];
        str[j] = str[len - j - 1];
        str[len - j - 1] = temp;
    }

    /* Adiciona ponto decimal */
    str[i++] = '.';

    /* Converte a parte fracionária para string */
    float fractionalPart = num - (int)num;
    int k;
    for (k = 0; k < precision; k++) {
        fractionalPart *= 10;
        int digit = (int)fractionalPart;
        str[i++] = digit + '0';
        fractionalPart -= digit;
    }

    /* Adiciona caractere de término */
    str[i] = '\0';
}
```

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.

```c
#ifndef _IO1_drivers
#define _IO1_drivers

#include "driver_init.h"
#include <stdio.h>
#include <at30tse75x.h>
#include <temperature_sensor.h>
#include <at30tse75x_config.h>

#define VCC_TARGET 3.3 // Tensão VCC da placa de R21 usada como referencia
#define CONF_AT30TSE75X_RESOLUTION 2

struct temperature_sensor *AT30TSE75X;
struct io_descriptor* TARGETIO_DEBUG;

static struct at30tse75x AT30TSE75X_descr;

void IO_SENSOR_INIT(void);
void sendByteToUART(uint8_t byte_to_send);
float readVoltageSensor(void);
float readCurrentSensor(float voltage);
float readLightSensor(float current);
void IO1X_LED_ON(void);
void IO1X_LED_OFF(void);
void floatToString(float num, char* str, int precision);


#endif
```

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.

```
 */
#include <atmel_start.h>
#include <stdio.h>
#include <IO1_drivers.h> // Inclui os arquivos de função dos sensores da placa IO1X Plained

int main(void)
{
    /* Initializes MCU, drivers and middleware */
    atmel_start_init();
    IO_SENSOR_INIT();
    char message[15]; // Mensagem a ser enviada pela serial para o terminal

    while (1) {

            //Liga LED da placa de expansão
            IO1X_LED_ON();
             // Leitura e envio da iluminância calculada
            float voltageSensor = readVoltageSensor();
            char voltage_str[20];
            floatToString(voltageSensor, voltage_str, 4);
            sprintf(message, "Tensão do sensor: %s V\r\n",voltage_str);
            printf(message);

            float currentSensor = readCurrentSensor(voltageSensor);
            char current_str[20];
            floatToString(currentSensor, current_str, 8);
            sprintf(message, "Corrente no sensor: %s âmperes\r\n", current_str);
            printf(message);

            float light_sensor = readLightSensor(currentSensor); // Iluminância medida pelo sensor de luz
            delay_ms(100);
            char iluminance_str[20];
            floatToString(light_sensor, iluminance_str, 4);

            sprintf(message, "Iluminancia: %s lux\r\n\n", iluminance_str);
            printf(message);

            // Leitura e envio da temperatura
            uint16_t temperature = readTemperatureSensor(); // Temperatura medida pelo sensor
            sprintf(message, "Temperatura: %d C\r\n", temperature);
            printf(message);

    }
}
```

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.

Table 4-1. Extension Header EXT1

Pin on EXT1	SAM R21 pin	Function	Shared functionality
1 [ID]	-	Communication line to ID chip on extension board	
2 [GND]	-	GND	
3 [ADC(+)]	PA06	AIN[6]	
4 [ADC(-)]	PA07	AIN[7]	
5 [GPIO1]	PA13	GPIO	
6 [GPIO2]	PA28	GPIO	
7 [PWM(+)]	PA18	TCC0 / WO[2]	
8 [PWM(-)]	PA19	TCC0 / WO[3]	
9 [IRQ/GPIO]	PA22	EXTINT[6]	
10 [SPI_SS_B/GPIO]	PA23	GPIO	
11 [TWI_SDA]	PA16	SERCOM1 PAD[0] I²C SDA	EXT3 and EDBG
12 [TWI_SCL]	PA17	SERCOM1 PAD[1] I²C SCL	EXT3 and EDBG
13 [USART_RX]	PA05	SERCOM0 PAD[1] UART RX	EDBG
14 [USART_TX]	PA04	SERCOM0 PAD[0] UART TX	EDBG
15 [SPI_SS_A]	PB03	SERCOM5 PAD[1] SPI SS	
16 [SPI_MOSI]	PB22	SERCOM5 PAD[2] SPI MOSI	EXT3 and EDBG
17 [SPI_MISO]	PB02	SERCOM5 PAD[0] SPI MISO	EXT3 and EDBG
18 [SPI_SCK]	PB23	SERCOM5 PAD[3] SPI SCK	EXT3 and EDBG
19 [GND]	-	GND	
20 [VCC]	-	VCC	

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

Table 4-2 I/O1 Xplained Pro Extension Header

Pin on EXT	Function	Description
1	ID	Communication line to ID chip
2	GND	Ground
3	LIGHTSENSOR	Light sensor output (ADC)
4	LP_OUT	Low pass filter output (ADC)
5	GPIO1	General Purpose I/O pin (GPIO)
6	GPIO2	General Purpose I/O pin (GPIO)
7	LED	LED control (PWM)
8	LP_IN	Low pass filter input (PWM)
9	TEMP_ALERT	ALERT pin (Pin 3) of temperature sensor chip (IRQ)
10	microSD_DETECT	Detect pin on microSD card connector (GPIO)

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

Pin on EXT	Function	Description
11	TWI SDA	Data line of TWI interface, connected to SDA pin (Pin 1) of temperature sensor chip
12	TWI SCL	Clock line of TWI interface, connected to SCL pin (Pin 2) of temperature sensor chip
13	UART RX	Receive pin of target MCU UART interface
14	UART TX	Transmit pin of target MCU UART interface
15	microSD_SS	Chip select signal, connected to DAT3 pin on microSD card connector
16	SPI_MOSI	Master Out, Slave In signal of target MCU SPI interface. Connected to CMD pin on microSD card connector.
17	SPI_MISO	Master In, Slave Out signal of target MCU SPI interface. Connected to D0 pin on microSD card connector.
18	SPI_SCK	Clock line of SPI interface. Connected to CLK pin on microSD card connector.
19	GND	Ground
20	VCC	Target supply voltage

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

3. Documentação

O arquivo contém os seguintes passos e códigos:

3.1. 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.

3.1.1. OpenThread Network Simulator (OTNS):

3.1.1.1. Instalação do Go:

O Go pode ser instalado pelo link

https://golang.org/dl/

3.1.1.2. Openthread network simulator:

Siga até o passo 3 do

https://openthread.io/codelabs/openthread-network-simulator#0

3.1.1.3. 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.

3.1.1.4. Após a clonagem do repositório OTNS:

Rodar o seguinte script na pasta "script":

$ *install-dep*

$ install

3.1.1.5. Abrindo o browser com o simulador:

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

3.2. Portabilidade Ot-Rtos para placa SAMR21

3.2.1. Compilação e Flash do OpenThread para a Placa Atmel SAMR21

3.2.1.1. Baixar GNU Toolchain:

$ cd <path-to-ot-samr21>

$ ./script/bootstrap

3.2.1.2. Build:

$ cd <path-to-ot-samr21>

$ ./script/build

3.2.1.3. 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

3.2.1.4. Formar uma Rede Thread no Thread Border Router:

Siga a partir do Passo 4 -
openthread.io/codelabs/esp-openthread-hardware#2

3.2.1.5. 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.

3.2.1.6. Compilação do OT-RTOS na Placa Nordic nRF52840:

$ git submodule update --init

$ mkdir build && cd build

$ cmake ..
-DCMAKE_TOOLCHAIN_FILE=../cmake/arm-none-eabi.cma
ke -DPLATFORM_NAME=nrf52

$ make -j12

Isso construirá a aplicação de teste CLI em
build/ot_cli_nrf52840.hex.

3.2.1.7. Configurar o RCP Joiner:

$ cd ~/src

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

$ cd ot-nrf528xx

$ script/build nrf52840 USB_trans

3.2.1.8. Compilação do código para a placa SAMR21:

$ git submodule update --init

```
$ mkdir build && cd build
$ cmake ..
-DCMAKE_TOOLCHAIN_FILE=../cmake/arm-none-eabi.cma
ke -DPLATFORM_NAME=samr21
$ make -j12
```

3.2.1.9. 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
${CMAKE_CURRENT_SOURCE_DIR}/third_party/openthre
ad/repo/microship/include/samr21x18a.ld)
  set_target_properties(ot_cli_${PLATFORM_NAME}
PROPERTIES LINK_FLAGS "-T
```

```
${CMAKE_CURRENT_SOURCE_DIR}/third_party/openthre
ad/repo/microship/include/samr21x18a.ld -lc -lnosys -lm
-lstdc++")
   set_target_properties(ot_demo_101 PROPERTIES
LINK_DEPENDS
${CMAKE_CURRENT_SOURCE_DIR}/third_party/openthre
ad/repo/microship/include/samr21x18a.ld)
   set_target_properties(ot_demo_101 PROPERTIES
LINK_FLAGS "-T
${CMAKE_CURRENT_SOURCE_DIR}/third_party/openthre
ad/repo/microship/include/samr21x18a.ld -lc -lnosys -lm
-lstdc++")
   #build hex file
   add_custom_command(OUTPUT
ot_cli_${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()
```

3.2.1.10. Adicionar o FreeRTOSConfig.h:

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

3.2.1.11. Adicionar o Arquivo cli.h:

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

3.3. Documentação Sensor de Luzes:

A documentação do sensor de luzes seguiu os seguintes passos:

3.3.1. Comentários de funções @brief e comentários de parâmetros de funções com @param

```
/**
 * @brief Desliga o LED da placa de expansão IO1X.
 */
void IO1X_LED_OFF(void);

/**
 * @brief Converte um número float em string com a precisão informada.
 *
 * @param num Número a ser convertido.
 * @param str String resultante da conversão.
 * @param precision Precisão da conversão.
 */
void floatToString(float num, char* str, int precision);
```

Figura 13 - Comentários de funções e parâmetros

3.3.2. Comentários de estruturas e definições no formato "/**<comentário */"

```
#define VCC_TARGET 3.3 /**< Tensão VCC da placa de R21 usada como referencia */
#define CONF_AT30TSE75X_RESOLUTION 2 /**< Resolução configurada para o sensor de temperatura */

struct temperature_sensor *AT30TSE75X; /**< Descritor para o sensor de temperatura AT30TSE75X */
struct io_descriptor* TARGETIO_DEBUG; /**< Descritor para a interface de comunicação USART de debug */

static struct at30tse75x AT30TSE75X_descr; /**< Descritor do sensor de temperatura AT30TSE75X */
```

Figura 14 - Comentários de estruturas

3.3.3. Comentários de cabeçalhos de arquivos @file

```
/** @file
 * @brief Funções relacionadas ao sensor de luz e suas operações.
 */

/**
 * @brief Flag que indica se a conversão analógico-digital do sensor de luz foi concluída.
 */
volatile bool conversion_done = false;
```

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:

3.3.4. Configurações de parâmetros para criação da documentação

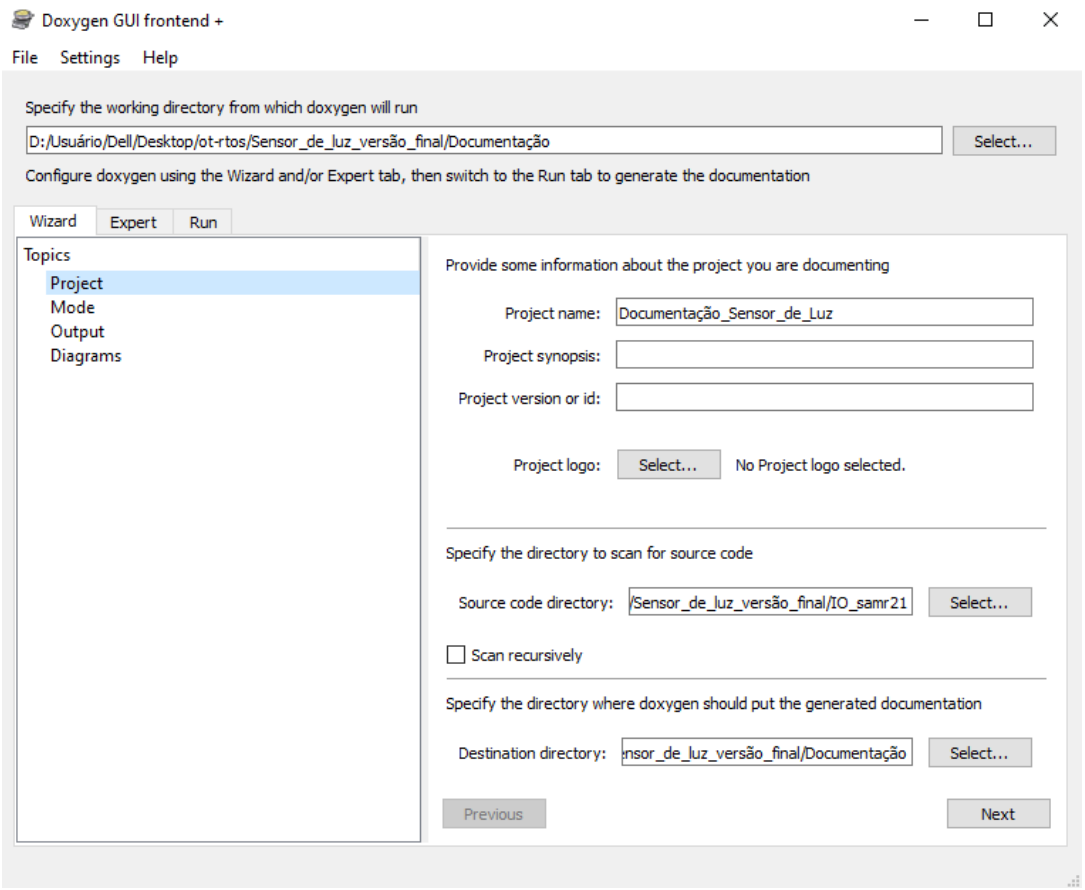

Figura 16 - Parâmetros do “doxywizard”

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

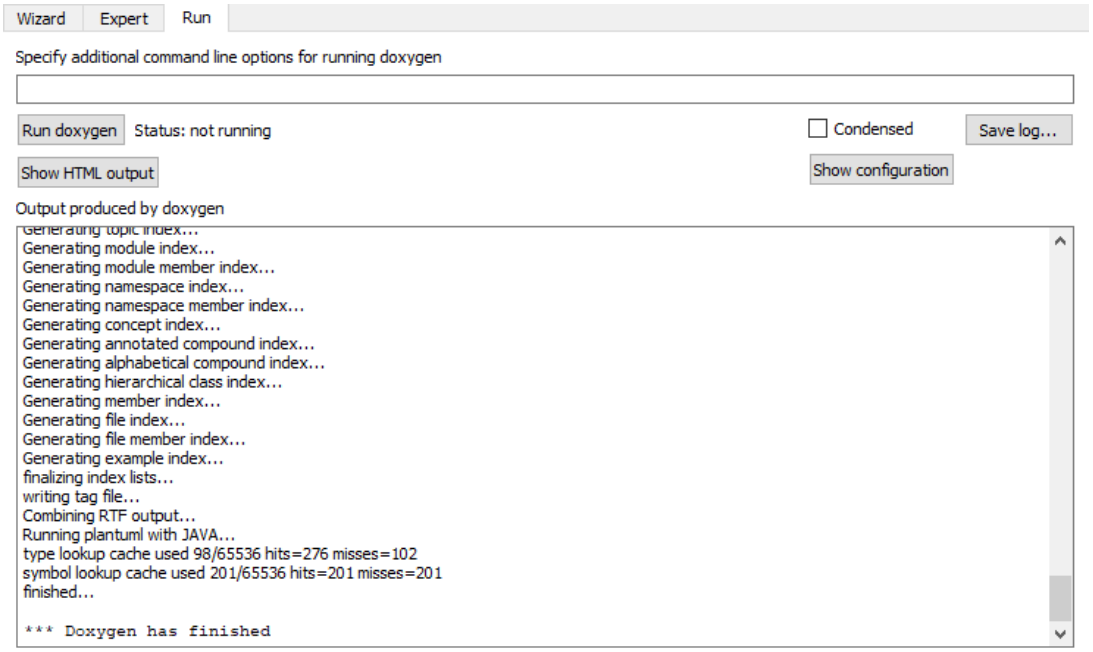

Figura 17 - Geração do html pelo doxywizard

3.3.6. 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

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

programados

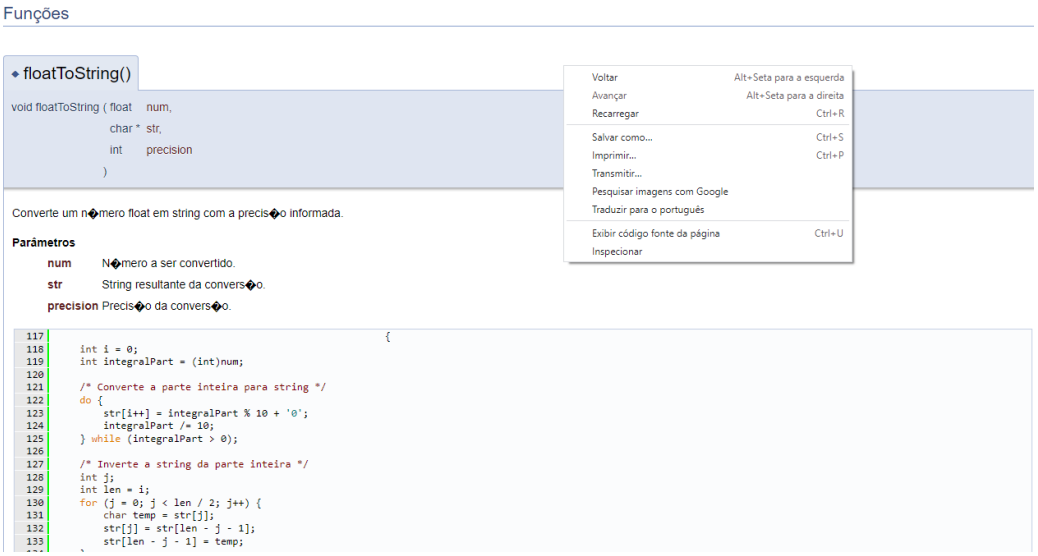

Figura 19 - Visualização de arquivo no html gerado pelo doxywizard

3.3.8. 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”.

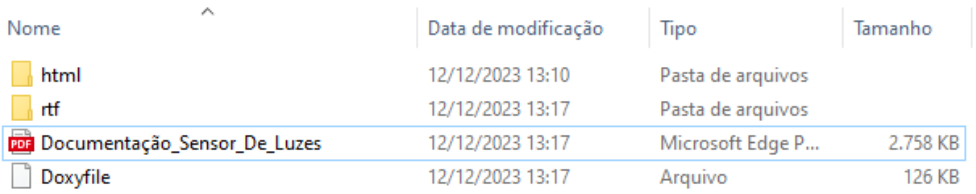

Figura 20 - Pasta contendo arquivos gerados da documentação do sensor de luzes

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. Futuramente, os códigos do IO_samr21, desenvolvidos para a placa I/O1 Xplained Pro, precisarão ser adaptados para operar no ot-samr21 e no ot-rtos.

4. 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
10. Generate documentation from source code - https://www.doxygen.nl/