Relatório - Sistemas Embarcados

EQUIPE DE ABSTRAÇÃO DE CAMADA DE HARDWARE (HAL)

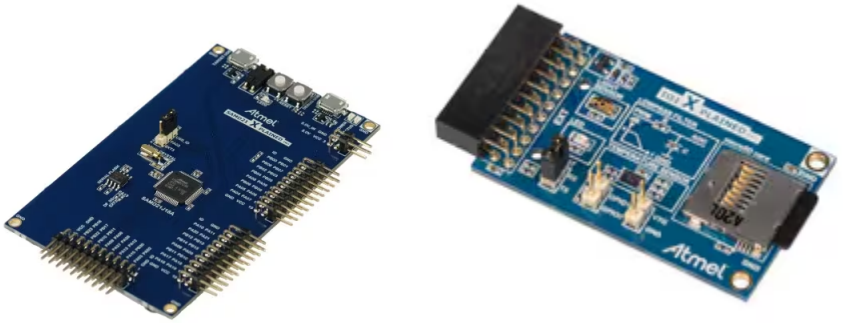
UFSM - Santa Maria - 2024

Este trabalho envolve o desenvolvimento de uma aplicação de Internet das Coisas (IoT) utilizando o Zephyr RTOS em conjunto com o OpenThread, com o objetivo de implementar a comunicação sem fio em uma rede mesh. Além disso, o projeto requer a portabilidade da aplicação para a plataforma de desenvolvimento SAM D21/R21 Xplained Pro, utilizando o Zephyr RTOS para gerenciar os recursos de hardware e garantir a integração eficiente dos componentes do sistema IoT.

Disciplina ministrada por Carlos Henrique Barriquello

RESUMO:

Este trabalho teve como objetivo desenvolver a camada HAL (Hardware Abstraction Layer) para a placa Atmel SAM D21 Xplained Pro, visando facilitar a comunicação com a extensão I/O1 Xplained Pro. Embora tenhamos alcançado sucesso na configuração do ambiente de desenvolvimento e na implementação de várias funções de hardware, enfrentamos dificuldades em alguns âmbitos específicos. Como resultado, a camada HAL criada ainda apresenta áreas que necessitam de melhorias para uma integração totalmente otimizada.



OBJETIVOS:

No desenvolvimento deste trabalho, optamos por portar apenas alguns componentes da extensão I/O1 Xplained Pro, visando uma integração mais eficiente e focada. Os componentes selecionados foram o sensor de temperatura (AT30TSE75X), o sensor de luz (TEMT6000) e o LED. Implementamos suas respectivas comunicações utilizando os protocolos I2C e GPIO, garantindo que a camada HAL criada pudesse gerenciar essas interfaces de forma eficaz. Essa escolha permitiu concentrar nossos esforços em garantir uma funcionalidade robusta e confiável para esses elementos, simplificando o processo de desenvolvimento e testes, ao mesmo tempo que proporciona uma base sólida para futuras expansões.

PLANEJAMENTO:

O planejamento iniciou com o estudo e reconhecimento do que seria necessário para conseguirmos executar o projeto através da leitura dos Datasheets de cada hardware utilizado no projeto. Logo em seguida, viria a configuração do ambiente de desenvolvimento, para que pudéssemos ter eficácia no projeto. Viria então a criação da camada de abstração e implementação dos códigos e seus respectivos testes. Após a validação dos códigos através dos testes, seria iniciada a documentação do que foi abstraído.

MÉTODOS:

Estudo, Engenharia reversa, compilação e criação de material.

RESULTADOS DO ESTUDO E RECONHECIMENTO:

1. ***Pinagem:***

1.1 Tipos de Pinos

*1. **GPIO (General Purpose Input/Output):** São pinos que podem ser configurados como entrada ou saída. Eles são versáteis e usados para interagir com dispositivos externos, como LEDs, botões e sensores.*
*2. **Pinos de Alimentação:** Fornecem tensão ao microcontrolador (VCC e GND). É crucial conectar esses pinos corretamente para garantir o funcionamento do microcontrolador.*
*3. **Pinos Analógicos:** Permitem a leitura de sinais analógicos através de um conversor ADC (Analog-to-Digital Converter), transformando valores analógicos em digitais.*
*4. **Pinos Digitais:** Utilizados para leitura e escrita de sinais digitais (alto ou baixo, 1 ou 0).*
*5. **Pinos de Comunicação**: Incluem pinos para I2C, SPI, UART e outras interfaces de comunicação.*

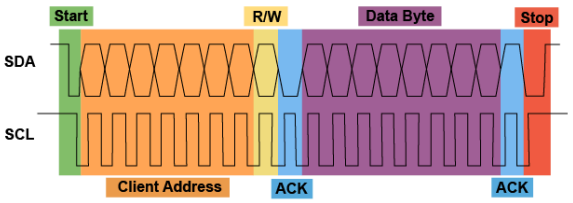
1.2 Configuração:

*1. **Direção:** Configurar o pino como entrada ou saída. Pinos de entrada são usados para ler sinais, enquanto pinos de saída são usados para enviar sinais.*
*2. **Estado Inicial:** Definir o estado inicial dos pinos de saída (alto ou baixo).*
*3. **Pull-up/Down:** Usados para definir um estado de alta ou baixa impedância quando o pino está desconectado. Resistores de pull-up conectam o pino à tensão positiva, enquanto resistores de pull-down conectam o pino ao terra.*

2. ***Protocolos de comunicação:***

***2.1 - ADC (Analog to Digital Converter) -** é um conversor que transforma sinais analógicos (contínuos) em sinais digitais (discretos). Em um microcontrolador, o ADC permite que o sistema interprete sinais analógicos provenientes de sensores como temperatura, luz, som, entre outros, convertendo-os em valores digitais que o microcontrolador pode processar.*

***2.2 - I2C (Inter Integrated Communication) -** é um protocolo de comunicação serial síncrono utilizado para permitir a comunicação entre múltiplos dispositivos em um mesmo barramento. É amplamente usado em sistemas embarcados devido à sua simplicidade e eficiência. O I2C utiliza duas linhas principais: SCL (Serial Clock Line) e SDA (Serial Data Line), que transportam o clock e os dados, respectivamente.*



2.3 - PWM (Pulse Width Modulation) - *é uma técnica utilizada para controlar a intensidade de LEDs (e outros dispositivos) variando a largura do pulso em um sinal digital. Essa técnica permite ajustar a luminosidade do LED, resultando em um efeito de escurecimento suave.*

2.4 - Comunicação Serial - *é um método de transmissão de dados em que os bits de dados são enviados sequencialmente, um após o outro, ao longo de um único canal ou fio. É uma das formas mais comuns de comunicação entre dispositivos em sistemas embarcados. A comunicação serial é utilizada em diversos protocolos, como UART, SPI, e I2C.*

OBJETIVO PRÁTICO: *abstrair o hardware necessário para que seja possível a implementação do seguinte programa:*

- *Através do sensor de luminosidade, controlar o LED_0 da placa (on/off)*
- *Através da do sensor de temperatura (parametrizado) controlar a intensidade do LED presente na extensão (PWM).*

INICIANDO O PROJETO: *Infelizmente, não houve uma boa adaptação ao ambiente de desenvolvimento Zephyr RTOS, o que ocasionou a busca por uma plataforma a qual houvesse maior familiaridade e suporte ao hardware utilizado na disciplina.*

Foi decidido, então, a utilização Microchip Studio, uma plataforma de desenvolvimento oferecida pela própria Microchip (distribuidora das placas Atmel SAMD21 Xplained Pro, utilizada na disciplina), que oferece uma abordagem mais didática, além de exemplos de implementação que ajudam no entendimento de possíveis soluções.

Com a utilização dessa plataforma, também é possível fazer uso do ASF Wizard, o qual importa dravers e protocolos de comunicação seguros (robustos) para que a camada HAL seja implementada com maior consistência.

INSTALANDO BIBLIOTECAS: *Foram adicionadas ao projeto, então as bibliotecas necessárias para o suporte dos periféricos que auxiliariam no bom desenvolvimento da camada de abstração:*

1 Driver ADC*: Para configurar e utilizar o conversor analógico-digital (ADC) do microcontrolador.*

2 Driver I2C*: Para comunicação com o sensor AT30TSE75X via protocolo I2C.*

3 Driver Syscalls*: Para chamadas de sistema, como configuração de clock e inicialização do sistema.*

4 Driver GPIO*: Para configurar e controlar os pinos GPIO (General Purpose Input/Output).*

5 Driver SERCOM (Serial Communication)*: Para configurar e utilizar a comunicação serial UART, se necessário.*

6 AT30TSE75X - *Para inicializar, configurar e executar funcionalidades básicas do periférico.*

Implementando Software: *essa parte do projeto trouxe inúmeros desafios para o grupo, pois foi um conhecimento adquirido sem auxílio ou supervisão, o que facilita o entendimento errado dos conceitos necessários.*

Sensor De Luminosidade

Pela facilidade de implementação, iniciamos com a implementação do sensor de Luminosidade com o protocolo de comunicação ADC. Estudando passo a passo de como

portar esse tipo de hardware, conseguimos chegar a um código que entrega o valor da luminosidade (get_luminosidade()) em um valor inteiro de 0 ~4096.

Código:

```
static uint16_t get_luminosidade(void)
{
	uint16_t luz_val;
	status_code_genare_t adc_status;
	adc_start_conversion(&adc_instance);
	do {
		adc_status = adc_read(&adc_instance, &luz_val);
	} while (adc_status == STATUS_BUSY);
	return LIGHT_SENSOR_RESOLUTION - luz_val;
}
```

Tivemos bons resultados com esse protocolo de comunicação, o que ocasionou o recebimento de uma variável bem comportada e de fácil parametrização.

Função	Retorno	Valores
get_luminosidade();	16 bits (inteiro +)	0 ~ 4096
get_luminosidade_per();	float (+)	0 ~100

A função get_luminosidade_per() é a parametrização de valor de luminosidade para porcentagem, ou seja, a simples divisão do inteiro por LIGHT_SENSOR_RESOLUTION e multiplicação por LIGHT_SENSOR_RESOLUTION.

Sensor De Temperatura (At30tse75x) (Problema)

Nesta etapa do projeto, enfrentamos a maior dificuldade: o protocolo de comunicação I2C, que nos levou a estagnação do projeto. Mesmo com o estudo do material, além da procura por diversos vídeos, não fomos capazes de implementar esse protocolo de comunicação. Algumas ideias que tivemos de implementação foram paradas frente aos obstáculos, principalmente de sincronia e comunicação serial. Cremos que, se não houvesse esse obstáculo, que talvez seria facilitado pelo uso do Zephyr e suas bibliotecas de drivers livres, teríamos completado o planejamento a tempo de fazer a melhor integração com a equipe de SOFTWARE do projeto.

RESULTADOS: *O projeto de desenvolvimento da camada HAL teve como resultado uma significativa aquisição de conhecimento sobre como operar nessa área, além de entender os inúmeros benefícios de uma implementação robusta e eficiente de uma camada de abstração, mesmo que por caso reverso. Houveram grandes dificuldades, porém resultados, em nossa visão, satisfatórios para um primeiro contato com esse tipo de desenvolvimento.*

Infelizmente, o planejamento inicial da equipe não foi cumprido, por inúmeras razões, além da falta de tempo devido ao ritmo acelerado do semestre. Porém, essa primeira implementação exitosa do sensor de luminosidade (que já demorou tempo suficiente), nos torna otimistas ao fazer novas tentativas.

Como resultado principal, temos o código abstraído de leitura da leitura do sensor de luminosidade, que implica o bom entendimento da comunicação ADC, a utilização e a interpretação dos sinais nos pinos da placa.

Código Abstraído:

```
/** UART module for debug. */
static struct usart_module cdc_uart_module;


/* Light sensor module. */
struct adc_module adc_instance;



static uint16_t get_luminosidade(void)
{
	uint16_t luz_val;
	status_code_genare_t adc_status;
	adc_start_conversion(&adc_instance);
	do {
		adc_status = adc_read(&adc_instance, &luz_val);
	} while (adc_status == STATUS_BUSY);
	return LIGHT_SENSOR_RESOLUTION - luz_val;
}


static void configure_console(void)
{
	struct usart_config usart_conf;

	usart_get_config_defaults(&usart_conf);
	usart_conf.mux_setting = EDBG_CDC_SERCOM_MUX_SETTING;
	usart_conf.pinmux_pad0 = EDBG_CDC_SERCOM_PINMUX_PAD0;
	usart_conf.pinmux_pad1 = EDBG_CDC_SERCOM_PINMUX_PAD1;
	usart_conf.pinmux_pad2 = EDBG_CDC_SERCOM_PINMUX_PAD2;
	usart_conf.pinmux_pad3 = EDBG_CDC_SERCOM_PINMUX_PAD3;
	usart_conf.baudrate    = 115200;

	stdio_serial_init(&cdc_uart_module, EDBG_CDC_MODULE, &usart_conf);
	usart_enable(&cdc_uart_module);
}


static void configure_adc(void)
{
	struct adc_config config_adc;
	adc_get_config_defaults(&config_adc);
	config_adc.gain_factor     = ADC_GAIN_FACTOR_DIV2;
	config_adc.clock_prescaler = ADC_CLOCK_PRESCALER_DIV32;
	config_adc.reference       = ADC_REFERENCE_INTVCC1;
	config_adc.positive_input  = ADC_POSITIVE_INPUT_PIN18;
	config_adc.resolution      = ADC_RESOLUTION_12BIT;
	config_adc.freerunning     = true;
	config_adc.left_adjust     = false;
	adc_init(&adc_instance, ADC, &config_adc);
	adc_enable(&adc_instance);
}


static void init_board(void)
{
	system_init();

	configure_console();

	configure_adc();
}
```

Resultado da abstração:

Função	Retorno	Valores
get_luminosidade();	16 bits (inteiro +)	0 ~ 4096
get_luminosidade_per();	float (+)	0 ~100