

RELATÓRIO FINAL DE PROJETO PARA CONTROLE DE TEMPERATURA DE CONDICIONADOR DE AR

Santa Maria, 26 de junho de 2018

Giuliano Bohn¹

Keli Tauana Prass Ruppenthal²

Victor Dallagnol Bento³

Resumo: O relatório em questão trará a análise do projeto final da disciplina de Projeto de Sistemas Embarcados. O intuito do projeto é desenvolver um controlador de temperatura. A ideia é que o controlador seja usado para verificar a temperatura ambiente e ajustar a temperatura do condicionador de ar conforme necessário.

Palavras-chave: *SAMD21, sensor de temperatura externo, protothreads, EEPROM, Display.*

1. Introdução

Como a base desta disciplina é o estudo de microcontroladores (neste caso o **ATSAMD21J18A**), devemos primeiramente entender a sua base de funcionamento para, então, explicar a forma como ele foi utilizado no projeto em questão. Como as aplicações são diversas, trataremos com mais detalhes aqui apenas do princípio de funcionamento deste projeto de condicionador de ar.

Primeiramente, pode-se dizer que um microcontrolador é um dispositivo que mistura software com hardware. Através de programação (C ou Assembly, geralmente), é possível controlar um hardware para fazer funções específicas de uma maneira fantasticamente simples, flexível e poderosa.

Numa abordagem mais técnica, podemos dizer que um microcontrolador é um circuito integrado, assim como um microprocessador. Porém, este CI não faz algo específico (como os CI's normais), mas é um de uso geral, já que pode ser programado para fazer uma tarefa pré-determinada.

A SAM D21, que será utilizada para este projeto, possui 32 e 64 pacotes de pinos com mais de 256 KB de memória Flash, 32 KB de SRAM, opera em uma frequência máxima de 48 MHz e possui um alcance de 2.46 Coremark/MHz. Ela foi projetada para uma migração simples e intuitiva com módulos periféricos idênticos, código hexadecimal compatível, endereço de portas idêntico e pinos compatíveis aos caminhos de migração entre todos os dispositivos produzidos. Todos eles, incluem periféricos inteligentes e flexíveis, *Atmel Event System* para sinalização inter- periférica e suporte para botão com

¹ Acadêmico(a) do curso de Engenharia de Computação da Universidade Federal de Santa Maria - UFSM, matrícula: 201322332, email: giuliano@compactjrl.com

² Acadêmico(a) do curso de Engenharia de Computação da Universidade Federal de Santa Maria - UFSM, matrícula: 201520603, email: kelitauana@gmail.com

³ Acadêmico(a) do curso de Engenharia de Computação da Universidade Federal de Santa Maria - UFSM, matrícula: 201520835, email: victor.bento@ecomp.ufsm.br

capacidade de toque, usuários de interfaces de controle deslizante e movimentos rotacionais. Para mais informações sobre a placa, deve-se consultar o *Datasheet*.

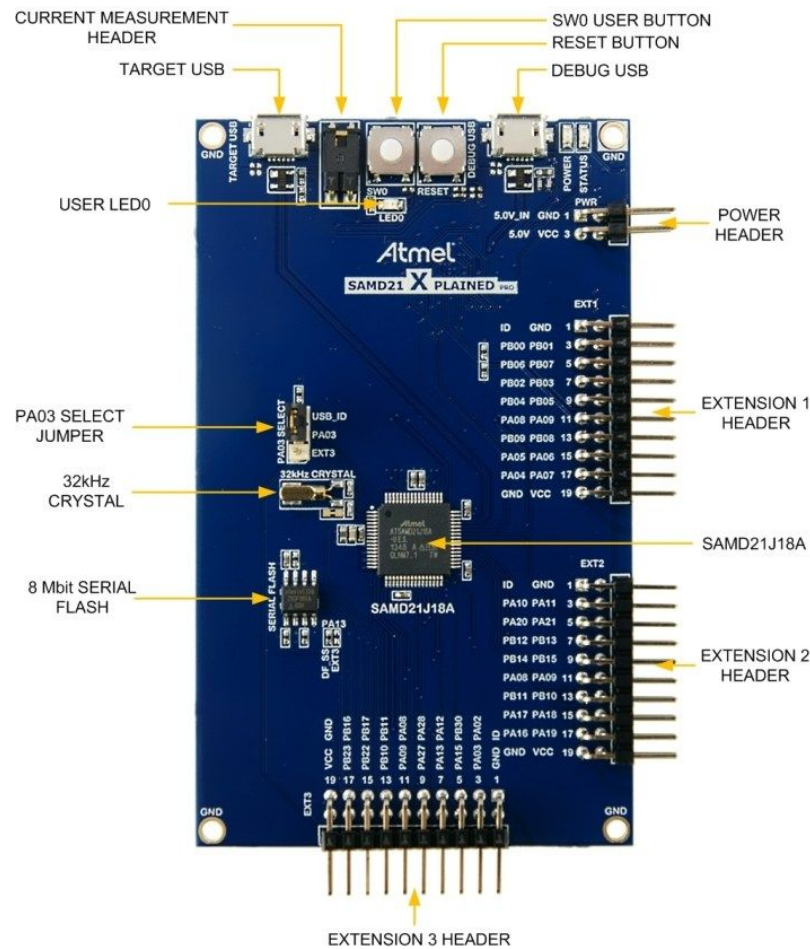


Figura 1: SAMD21J18A microcontroller.

2. Materiais Utilizados

Para este projeto foram necessários os seguintes materiais:

- Microcontrolador SAMD21J18A Xplained Pro;
- Memória EEPROM (armazenamento de dados).
- Display OLED1 Xplained Pro (saída de dados);
- Sensor de temperatura externo I/O1 Xplained Pro (entrada de dados);

O código foi desenvolvido na plataforma *Atmel Studio*, e posteriormente adicionado ao repositório do GitHub (https://github.com/da3mons/Embedded_Systems_Design).

3. Metodologia

3.1 Sensor de Temperatura (entrada de dados)

Primeiramente, seguindo como referência alguns projetos feitos durante o semestre, foi feito o código para leitura da temperatura ambiente. Contudo, o sensor testado durante as aulas foi o interno, enquanto que o do projeto final, é um sensor externo. Essa opção foi

escolhida devido ao fato do sensor externo ser mais preciso do que o interno, uma vez que o último opera dentro da placa, e acaba se aquecendo enquanto executa o processamento.

Inicialmente, inseriu-se todos módulos que viriam a ser utilizados pelas funções de entrada e saída, sensores, USART, entre outros. A **Figura 2** demonstra os módulos adicionados.

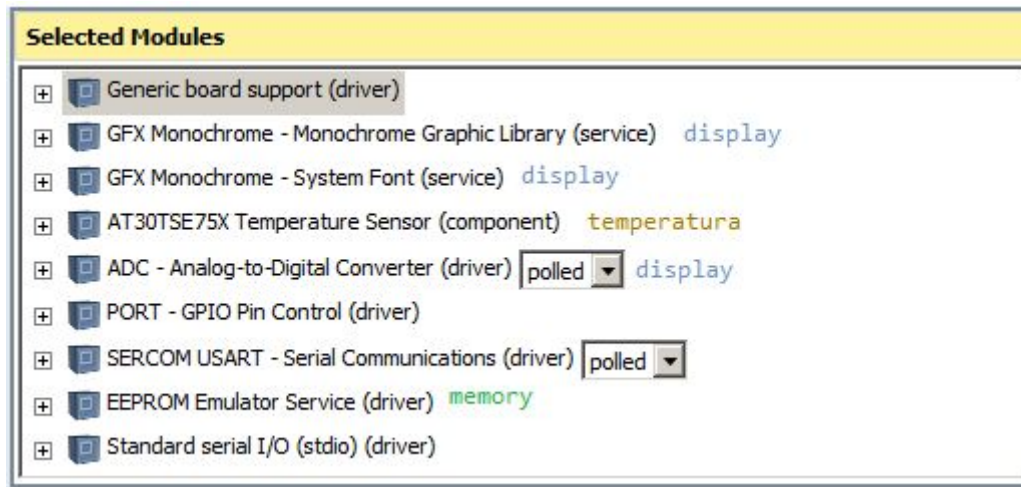
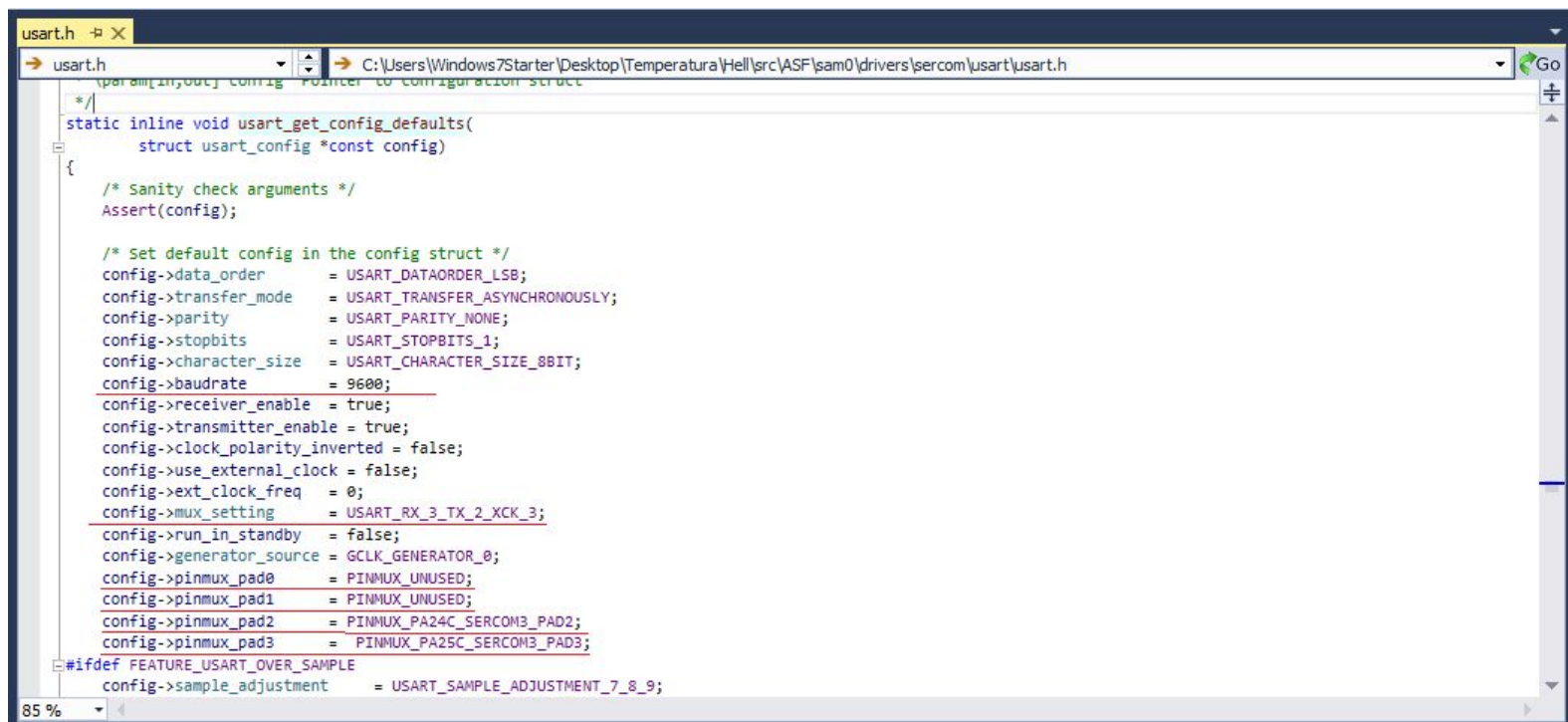


Figura 2: Módulos adicionados para o funcionamento do sistema embarcado.



Figura 3: Sensor de temperatura externo I/O1 Xplained Pro.

Feito isso, foram definidas as structs para configurar a USART e definições de variáveis globais. Na *main*, o sistema é inicializado pela função `system_init()`; . A configuração da USART como default para a leitura dos pinos é dada pela função `usart_get_config_defaults(&usart_conf)`; . Ainda, deve-se iniciar a comunicação com o periférico de temperatura para que o valor desta possa ser lido de forma correta. A função `stdio_serial_init(&usart_instance, EDBG_CDC_MODULE, &usart_conf)`; é responsável por isso. A habilitação da USART se dá pela função `usart_enable(&usart_instance)`; . A inicialização do periférico (já configurado com `thigt` e `tlow`) é feita pela função `at30tse_init()`; . Após isso, o laço *while* irá armazenar o valor de N temperatura medidas com o sensor e seu valor médio será calculado posteriormente.

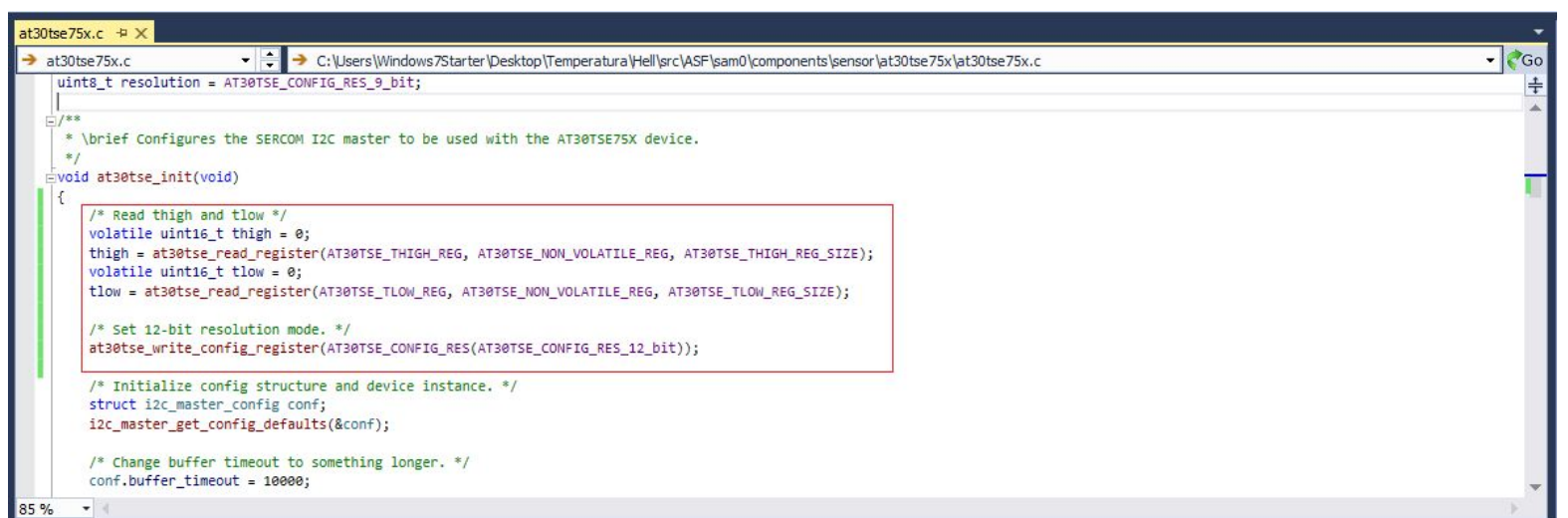


```
usart.h
static inline void usart_get_config_defaults(
    struct usart_config *const config)
{
    /* Sanity check arguments */
    Assert(config);

    /* Set default config in the config struct */
    config->data_order      = USART_DATAORDER_LSB;
    config->transfer_mode    = USART_TRANSFER_ASYNCHRONOUSLY;
    config->parity           = USART_PARITY_NONE;
    config->stopbits         = USART_STOPBITS_1;
    config->character_size    = USART_CHARACTER_SIZE_8BIT;
    config->baudrate          = 9600;
    config->receiver_enable  = true;
    config->transmitter_enable = true;
    config->clock_polarity_inverted = false;
    config->use_external_clock = false;
    config->ext_clock_freq    = 0;
    config->mux_setting       = USART_RX_3_TX_2_XCK_3;
    config->run_in_standby    = false;
    config->generator_source = GCLK_GENERATOR_0;
    config->pinmux_pad0       = PINMUX_UNUSED;
    config->pinmux_pad1       = PINMUX_UNUSED;
    config->pinmux_pad2       = PINMUX_PA24C_SERCOM3_PAD2;
    config->pinmux_pad3       = PINMUX_PA25C_SERCOM3_PAD3;
}

#ifdef FEATURE_USART_OVER_SAMPLE
config->sample_adjustment = USART_SAMPLE_ADJUSTMENT_7_8_9;
```

Figura 4: Modificações na função *usart.h*.



```
at30tse75x.c
uint8_t resolution = AT30TSE_CONFIG_RES_9_bit;

/**
 * \brief Configures the SERCOM I2C master to be used with the AT30TSE75X device.
 */
void at30tse_init(void)
{
    /* Read thigh and tlow */
    volatile uint16_t thigh = 0;
    thigh = at30tse_read_register(AT30TSE_THIGH_REG, AT30TSE_NON_VOLATILE_REG, AT30TSE_THIGH_REG_SIZE);
    volatile uint16_t tlow = 0;
    tlow = at30tse_read_register(AT30TSE_TLOW_REG, AT30TSE_NON_VOLATILE_REG, AT30TSE_TLOW_REG_SIZE);

    /* Set 12-bit resolution mode. */
    at30tse_write_config_register(AT30TSE_CONFIG_RES(AT30TSE_CONFIG_RES_12_bit));

    /* Initialize config structure and device instance. */
    struct i2c_master_config conf;
    i2c_master_get_config_defaults(&conf);

    /* Change buffer timeout to something longer. */
    conf.buffer_timeout = 10000;
```

Figura 5: Modificações na função *at30tse75x.c*.

3.2 Protothreads (organização do código e otimização das tarefas)

Para este projeto foram criadas duas protothreads. Uma *sender* (responsável por enviar o valor da média da temperatura lida da memória) e outra *receiver* (responsável por ler este dado e chamar a função específica que irá setar o ar condicionado e mostrar no display). Na função *main* a variável ACK é atualizada quando a temperatura for gravada na memória. Com isso, a protothread *sender* lê este valor, salvando-o em uma variável auxiliar. É feita a atualização de uma flag SET, fazendo com que a protothread *receiver* “receba” o dado, e faça as devidas comparações. Elas foram inicializadas pela função `PT_INIT(&pt_sender)` e pela função `PT_INIT(&pt_receiver)`, e invocadas pelas

funções `sender(&pt_sender)` e `receiver(&pt_receiver)`. Ambas as funções executadas pelas protothreads estão na **Figura 6**.

```

/*
PT_THREAD(sender(struct pt *pt))
{
    PT_BEGIN(pt);
        PT_WAIT_UNTIL(pt, ack == 1);
        eeprom_emulator_read_page(0, aux);
        set = 1;
        ack = 0;
    PT_END(pt);
}

/*
* ProtoThread para comparar a temperatura
* Espera flag que informa que temperatura já foi lida da memória
*/
PT_THREAD(receiver(struct pt *pt))
{
    PT_BEGIN(pt);
        PT_WAIT_UNTIL(pt, set == 1);
        set = 0;
        if (final_temp <= 15)
        {
            temp_low();
        }
        else if (final_temp >= 26)
        {
            temp_high();
        }
        else
        {
            temp_default();
        }
    PT_END(pt);
}

```

Figura 6: Estruturas das protothreads utilizadas.

3.3 Memória (armazenamento e leitura de dados)

A configuração da memória é dada pela função `configure_eeprom(void)`. Após a média da temperatura ser calculada, o valor é gravado na memória através da função `eeprom_emulator_write_page(0, temp_lida);`. Quando a protothread *sender* for ler este dado da memória para enviar para a *receiver*, utilizará a função `eeprom_emulator_read_page(0, aux);`. Dessa forma, o dado lido estará na variável **aux**, e poderá ser acessada pela *receiver*. Neste projeto, foi utilizada apenas a página zero da memória.

3.4 Display (saída de dados)

O display é a parte que fará a conexão com o usuário. O modelo utilizado no projeto está na **Figura 7**. Sua função será mostrar três possibilidades:

1. Caso a temperatura estiver abaixo de 15°C, o Display irá mostrar o valor da temperatura ambiente e o valor para o qual o ar condicionado deve ser setado (23°C);

2. Caso a temperatura estiver acima de 26°C, o Display irá mostrar o valor da temperatura ambiente e o valor para o qual o ar condicionado deve ser setado (20°C);
3. Como default, caso a temperatura esteja entre estes valores (15°C e 26°C), o Display irá mostrar a temperatura ambiente e desligará o aparelho de ar condicionado.

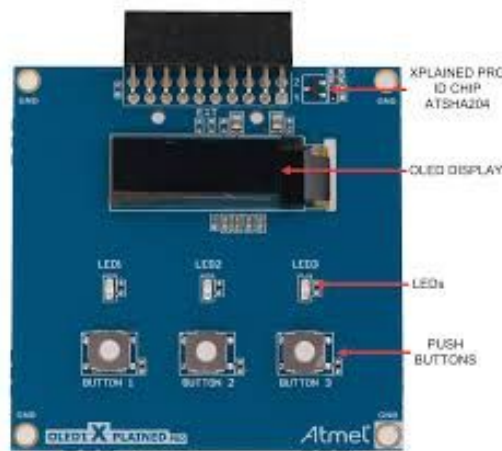


Figura 7: Display OLED1 Xplained Pro .

Primeiramente, inicializou-se o Display pela função `gfx_mono_init()`; . Feito isso, quando a protothread *receiver* chamar uma das três funções do aparelho de ar condicionado referidas anteriormente, a inicialização do Display é feita novamente para que ele seja resetado e sua leitura não seja prejudicada. As possibilidades de escrita, bem como o ajuste dos eixos x e y são feitas na função `gfx_mono_draw_string("string que será escrita", eixo x, eixo y, &sysfont)`; . Dessa forma, o usuário estará apto a ler os dados medidos pelo sensor. Nota-se ainda que na função *main* a temperatura média (*int*) é convertida para *char* através da função `itoa((int)final_temp, temp_lida, 10)`; , que salva em **temp_lida** o valor convertido. Isso serve para possibilitar a leitura do dado pelo Display, uma vez que ele apenas lê caracteres.

4 Conclusão

Levando-se em consideração o tempo dedicado ao entendimento e montagem dos módulos, o trabalho foi realizado dentro do prazo, porém com alguns imprevistos. Na especificação do trabalho, enviada anteriormente, era especificado que usaríamos a placa SAM R21, memória Flash e sensor de temperatura interno. Alterou-se estes componentes para placa SAM D21, memória EEPROM e sensor de temperatura externo, devido ao fato de existirem mais materiais e exemplos disponíveis para tais componentes.

Ademais, os valores coletados estiveram muito próximos do real, pois o ambiente em que o sensor foi testado era climatizado e a temperatura encontrada pelo sensor era a mesma ou próxima a que marcava o aparelho do local. Esta análise valida os dados obtidos pelo projeto em questão.

Por fim, pode-se dizer que o uso de microcontroladores em geral possui uma vasta área de utilidades e aplicações. O projeto aqui referido pode ser visto como algo simples, mas que sintetiza os aspectos principais de um sistema embarcado (leitura, armazenamento e saída de dados) e que foi fundamental para o entendimento do real

funcionamento deste tipo de sistema. Embora existam equipamentos condicionadores de ar muito mais eficazes e inovadores do que este, o intuito de projetar e assimilar o que foi aprendido em sala de aula com um sistema que será visto em funcionamento na prática, abre inúmeras oportunidades de aperfeiçoar os equipamentos já existentes e contribuir para a inovação tecnológica.

Referências

[1] “*User Guide OLED1 Xplained Pro.*” acessado em: 15/06/2018. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42077-OLED1-Xplained-Pro_User-Guide.pdf>

[2] “*User Guide I/O1 Xplained Pro.*” acessado em: 14/06/2018. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42078-IO1-Xplained-Pro_User-Guide.pdf>

[3] “*User Guide SAM D21 Xplained Pro.*” acessado em: 11/06/2018. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42220-SAMD21-Xplained-Pro_User-Guide.pdf>