



Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica
e de Computação



EA871 - Laboratório de Programação Básica de Sistemas Digitais

Projeto Final: Alarme despertador

Autores: Fernando Teodoro de Cillo e Rafael Silva Cirino

RA:

197029

223730

Campinas
22 de Julho de 2022

1 Introdução

O intuito deste experimento é integrar os conhecimentos adquiridos nos experimentos anteriores, realizando um projeto autoral utilizando o microcontrolador MKL25Z128 e os conceitos de programação estudados em sala de aula.

Para isso, foi concebido um alarme com horário programável, que faz tocar uma sirene assim que chega o horário determinado.

2 Especificações

2.1 Diagrama de Estados

O primeiro passo do projeto é determinar o funcionamento da máquina de estados que descreve o alarme. Assim, é essencial descrever os estados e fazer um diagrama, que pode ser visto na figura 1.

2.1.1 Descrição dos estados

Inicialização do sistema

CONFIG_RELOGIO: Estado de transição para o usuário introduzir a hora atual, libera acesso ao terminal

HORA_ATUAL: Usuário introduz a hora atual no terminal

INICIO: Apresenta no LCD o horário atualizado em tempo real

CONFIG_ALARME: Ao pressionar IRQA5 entra no estado para o, usuário introduz no terminal a hora do alarme

CONFIRMACAO: ao pressionar IRQA5 confirma que o alarme configurado está correto e volta para início, se pressionar IRQA12 cancela alarme e volta para início

ALARME_VERIFICA: Apresenta por 5 segundos na tela o alarme configurado atualmente

ALARME_INTERRUPT: Horário do alarme atingido, Mensagem no LCD de aviso por 10 segundos (tempo de soneca)

MENSAGEM: Mensagem no LCD e led RGB piscando durante 10s

2.1.2 Diagrama de estados

O diagrama de estados que descreve o funcionamento do pseudo-código pode ser visto na figura 1.

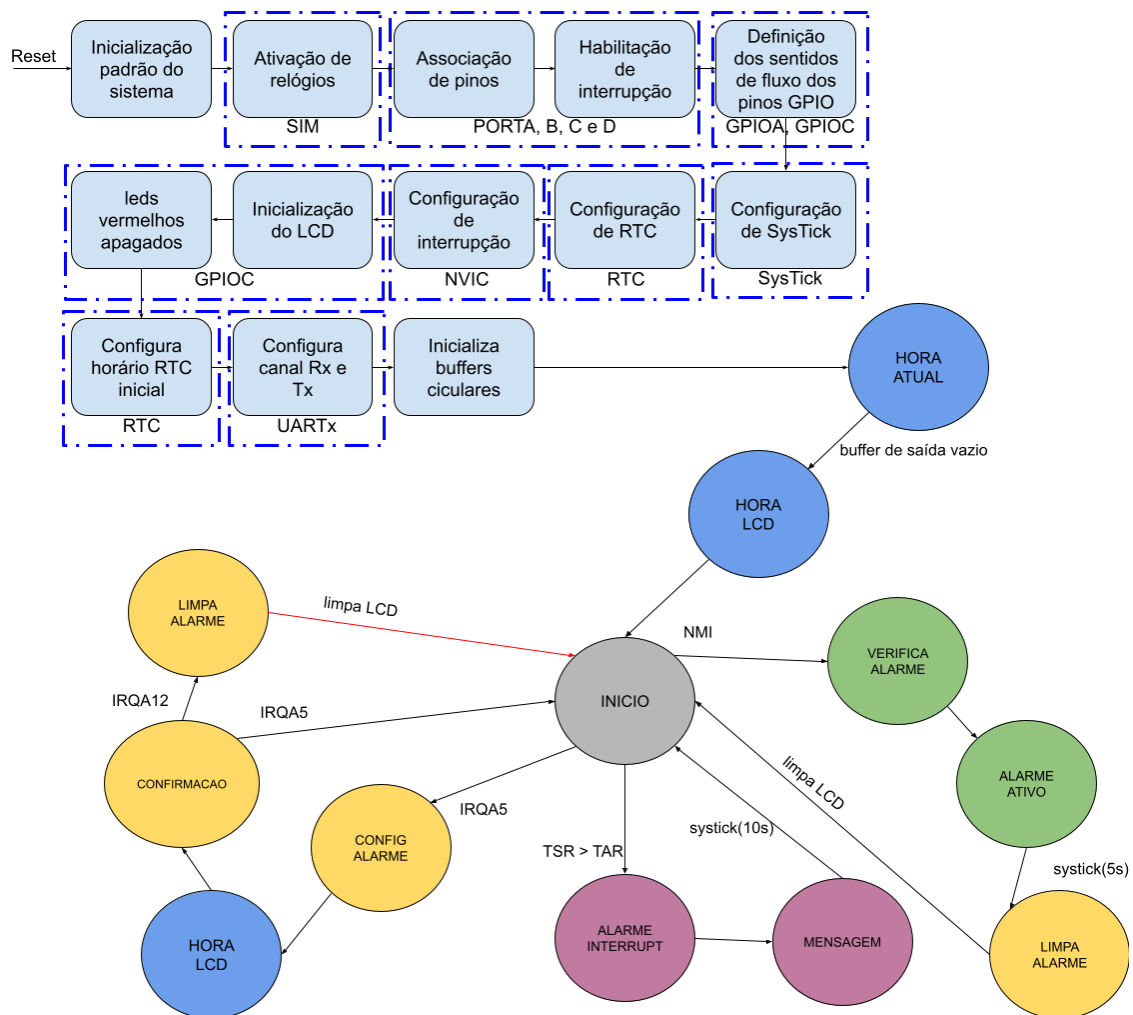


Figura 1: Máquina de estados do projeto alarme

2.2 Lista de módulos e componentes utilizados

- GPIO;
- SIM;
- LED RGB;
- Display LCD;
- RTC;
- SysTick;
- UART;
- Botões NMI, IRQA5 e IRQA12;
- Sirene Deton PPA 120 Db preta, 200 mA, 12 V DC;
- Fonte de 12 V.

```

int main(void){
    config_uart();
    init_modules();
    segundos_atual = config_hora();

    RTClpo_setTime (segundos_atual);
    ISR_carregaHorario();

    //segundos_atual = 0;
    atualizaHorarioLCD (segundos_atual, estado);
    segundos_anterior = segundos_atual;

    ISR_EscreveEstado (INICIO);

    TPM_initPTB0EPWM(65525, 0b101, 0, ON);
    for(;;){
        estado = ISR_LeEstado();

        switch (estado) {
            case INICIO:
                RTClpo_getTime(&segundos_anterior);
                atualizaHorarioLCD (segundos_anterior, estado);

                TPM_atualizaDutyCycleEPWM(1, 0, 0);
                break;
            case CONFIG_ALARME:
                segundos_alarme = config_hora();
                GPIO_escreveStringLCD (0x01, "          ");
                atualizaHorarioLCD (segundos_alarme, estado);

                RTC_ativaAlarmIRQ ();
                RTClpo_setAlarm (segundos_alarme);

                ISR_EscreveEstado (CONFIRMACAO);
                break;
            case VERIFICA_ALARME:
                GPIO_escreveStringLCD (0x02, "Alarme config");
                atualizaHorarioLCD (segundos_alarme, estado);

                ISR_EscreveEstado (ALARME_ATIVO);
                break;
            case LIMPA_ALARME:
                GPIO_escreveStringLCD (0x01, "          ");
                GPIO_escreveStringLCD (0x42, "          ");

                ISR_EscreveEstado (INICIO);
                break;
            case ALARME_INTERRUPT:
                GPIO_escreveStringLCD (0x02, "ALARME ALARME");

                ISR_EscreveEstado (MENSAGEM);
                break;
            case MENSAGEM:
                TPM_atualizaDutyCycleEPWM(1, 0, 50);
                break;
            case ALARME_ATIVO:
            case CONFIRMACAO:
            case HORA_ATUAL:
                break;
        }
    }

    return 0;
}

```

Figura 2: Função main()

3 Projeto

3.1 Função main()

A função `main()`, mostrada na figura 2, serve para configurar os módulos utilizados e inicializar as variáveis, além de entrar no estado inicial da máquina de estados (estado INICIO). É possível ver as semelhanças comparando o código com a máquina de estados da figura 1.

3.2 LED RGB

O led RGB acende na cor vermelho quando ocorre uma interrupção, como mostrado na figura 3. A configuração deste módulo segue o roteiro 4.

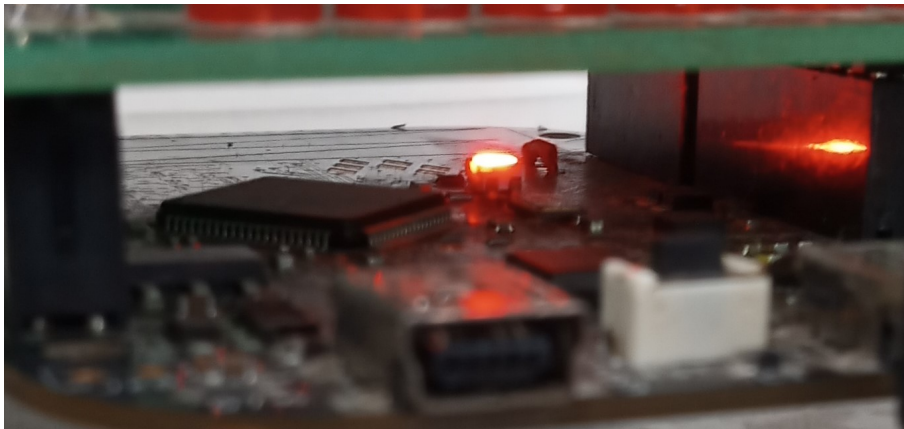


Figura 3: Led RGB aceso na cor vermelho (ON,OFF,OFF) no estado ALARME_INTERRUPT

3.3 Display LCD

Seguindo o que foi feito no roteiro 8, a figura 4 mostra a função `atualizaHorarioLCD`, que serve para manter o horário atualizado com o temporizador RTC mas sem o cursor piscar incessantemente. As mensagens exibidas no LCD em cada estado podem ser vistas na figura 5.

```
void atualizaHorarioLCD (uint32_t segundos, estado_type estado){
    char str_hhddss[8];
    ttoa(segundos, str_hhddss);

    switch(estado){
        case CONFIG_ALARME:
        case HORA_LCD:
        case INICIO:
            GPIO_escreveStringLCD (0x04, str_hhddss);
            break;
        case VERIFICA_ALARME:
            GPIO_escreveStringLCD (0x44, str_hhddss);
    }
}
```

Figura 4: Função `atualizaHorarioLCD`



(a) Estado INICIO



(b) Estado CONFIG_ALARM



(c) Estado VERIFICA_ALARM



(d) Estado ALARME_INTERRUPT

Figura 5: Mensagens exibidas no display LCD

3.4 Temporizadores

Os temporizadores utilizados foram o RTC, que faz a atualização do tempo no microcontrolador e no LCD relativos ao valor inicializado pelo usuário, e o SysTick, que é utilizado para interrupções e contadores de tempo para realizar a troca de estados no tempo correto, seguindo os conceitos desenvolvidos no roteiro 9.

A função `config_hora`, da figura 6a, envia *strings* para o LCD, com a intenção de explicar ao usuário o que fazer, e recebe *strings* no formato HH/MM/SS nos estados HORA_ATUAL e CONFIG_ALARM. Já a função `str_time` converte uma *string* no formato HH/MM/SS para um valor em segundos, possibilitando que os módulos RTC e SysTick operem com ele. Este código pode ser visto na figura 6b.

3.5 UART

Para configurar o teclado do computador como entrada serial, é utilizado o módulo UART, cujas configurações devem ser definidas na inicialização do projeto e são mostradas na figura 7a. O *baud rate*, o número de *bits* de dados e de *bits* de parada são essenciais para que a comunicação ocorra corretamente, como visto no roteiro 11. A porta serial depende da máquina que está rodando o *IDE CodeWarrior*. Para configurar corretamente o módulo UART, é necessário saber o *baud rate* (38400), a frequência do processador (20971520 Hz), setar o divisor de frequência do barramento e habilitar as interrupções, como mostrado na figura 7b. A figura 8 mostra o tratamento de interrupções da UART, que somente libera o terminal para escrita no teclado nos estados HORA_ATUAL e CONFIG_ALARM, em que é pedido para que o usuário digite através de uma mensagem no terminal.

```

int config_hora(){
    uint32_t segundos;

    estado = ISR_LeEstado();
    char s_entrada[8];

    if (estado == HORA_ATUAL){
        ISR_EnviaString("Digite a hora atual no seguinte formato: 00 00 00 \n\r");
    } else if (estado == CONFIG_ALARME){
        GPIO_escreveStringLCD (0x02, "Insira alarme");
        ISR_EnviaString("Digite o alarme no seguinte formato: 00 00 00 \n\r");
    }

    estado = ISR_LeEstado();
    while ((estado == HORA_ATUAL) || (estado == CONFIG_ALARME)){
        estado = ISR_LeEstado();
        if (estado == HORA_LCD){
            ISR_extraiString(s_entrada);
            ISR_EnviaString("\n\r");

            str_time (s_entrada, &segundos);
        }
    }

    return segundos;
}

```

(a) Função config_hora

```

int str_time (char string[], int *seg){
    char* str;
    const char espaco[1] = " ";

    int i = 0;
    int oper_i[3];

    for(str=strtok(string, espaco); str!=0; str=strtok(0, espaco)){
        if(isdigit(*str) != 0){
            oper_i[i] = atoi(str);
        } else if (i > 2){
            return -1;
        } else{
            return -1;
        }
    }

    i += 1;
}

dhms2s(0, oper_i[0], oper_i[1], oper_i[2], seg);

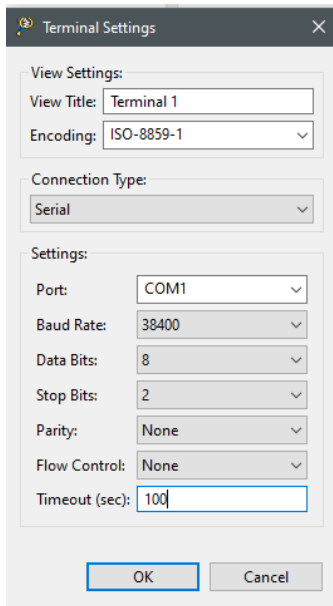
if (i < 3){
    return -1;
}

return 0;
}

```

(b) Função str_time

Figura 6: Funções dos temporizadores



(a) Terminal

```
void config_uart(){
    /*!
     * Seto o divisor de frequencia do barramento
     */
    SIM_setOUTDIV4 (0b001);

    /*!
     * Configurar UART0 com Rx e Tx habilitados: SBR tem que ser aproximado para maior que baud rate setado
     */
    config0.sbr = (uint16_t) (20971520./ (BAUD_RATE * (config0.c4_osr+1)));
    if ((20971520./ (1.0 * BAUD_RATE * (config0.c4_osr+1))) > config0.sbr)
        config0.sbr++;
    UART0_initTerminal (&config0);

    /*!
     * Ativa IRQs
     */
    UART0_ativaIRQTerminal (0);

    // Setup
    ISR_inicializaBC();

    /*!
     * Habilita a interrupcao do Rx do UART0
     */
    UART0_ativaInterruptRxTerminal();
    //UART0_ativaInterruptTxTerminal();

    ISR_EnviaString ("\n\r");
}
```

(b) Configuração do módulo UART

```
void UART0_IRQHandler()
{
    char item;

    estado = ISR_LeEstado();
    if (UART0_S1 & UART0_S1_RDRF_MASK) {
        /*!
         * Interrupcao solicitada pelo canal Rx
         */
        char r;

        r = UART0_D;

        if ((estado != HORA_ATUAL) && (estado != CONFIG_ALARME)) {
            //Liberar o canal para novas recepcoes
            return;
        }

        UART0_D = r; //Ecoar o caractere

        //Adicionar no buffer circular de entrada
        if (r == '\r') {
            //inserir o terminador e avisar o fim de uma string
            BC_push (&bufferE, '\0');
            while (!(UART0_S1 & UART_S1_TDRE_MASK));
            UART0_D = '\r';
            UART0_D = '\n';

            ISR_EscreveEstado (HORA_LCD);
        } else {
            BC_push (&bufferE, r);
        }
    }
}
```

Figura 8: uart irq

3.6 Sirene

Para o alarme sonoro do despertador foi utilizada uma sirene Deton PPA 120 Db preta, 200 mA, 12 V DC, vista na figura 9. A configuração da sirene foi feita por PWM, como no projeto do cooler (roteiro 10), em que o *duty cycle* é setado para 0% quando se quer desligar a sirene e

50% quando se quer ligá-la.



Figura 9: Sirene Deton PPA 120 Db

3.7 Interrupção

O alarme do despertador foi configurado por interrupção (roteiro 7). Quando o tempo em TSR é maior que o tempo em TAR, ocorre a interrupção: há o disparo da sirene (*duty cycle*=50%), o display LCD mostra a mensagem "ALARME ALARME" e o led RGB acende em vermelho. Depois de 3 segundos, a subrotina retorna para o fluxo normal do código, o *duty cycle* da sirene volta para 0, o display LCD exibe o horário atual e o led RGB se apaga.

```
void PORTA_IRQHandler (void){
    GPIO_leSwitchesISF(&valor);

    switch(estado){
        case INICIO:
            if (valor == 0x2) {
                ISR_EscreveEstado (CONFIG_ALARME);
            } else if (valor == 0x1) {
                SysTick_ativaInterrupt ();
                ISR_EscreveEstado (VERIFICA_ALARME);
            }
            break;
        case CONFIRMACAO:
            if (valor == 0x100) {
                ISR_EscreveEstado (LIMPA_ALARME);
            }
            break;
    }

    //SysTick_ativaInterrupt ();
    PORTA_ISFR |= 0x1030;
}
```

Figura 10: Rotina de serviços da interrupção

A figura 10 mostra a rotina de interrupção da PORT A e a figura 11 mostra a interrupção do SysTick.

```
void SysTick_Handler(void) {
    ISR_LeHorario(&segundos);

    if((contador > 8) && (estado == ALARME_ATIVO)){
        SysTick_desativaInterrupt ();

        contador = 0;
        ISR_EscreveEstado (LIMPA_ALARME);
    } else if ((contador > 8) && (estado == MENSAGEM)){
        SysTick_desativaInterrupt ();
        //TPM_atualizaDutyCycleEPWM(1, 0, 0);

        contador = 0;
        ISR_EscreveEstado (LIMPA_ALARME);
    } else if ((contador > (seg_alarm * 2)) && (estado == INICIO)){
        SysTick_desativaInterrupt ();
        //TPM_atualizaDutyCycleEPWM(1, 0, 50);

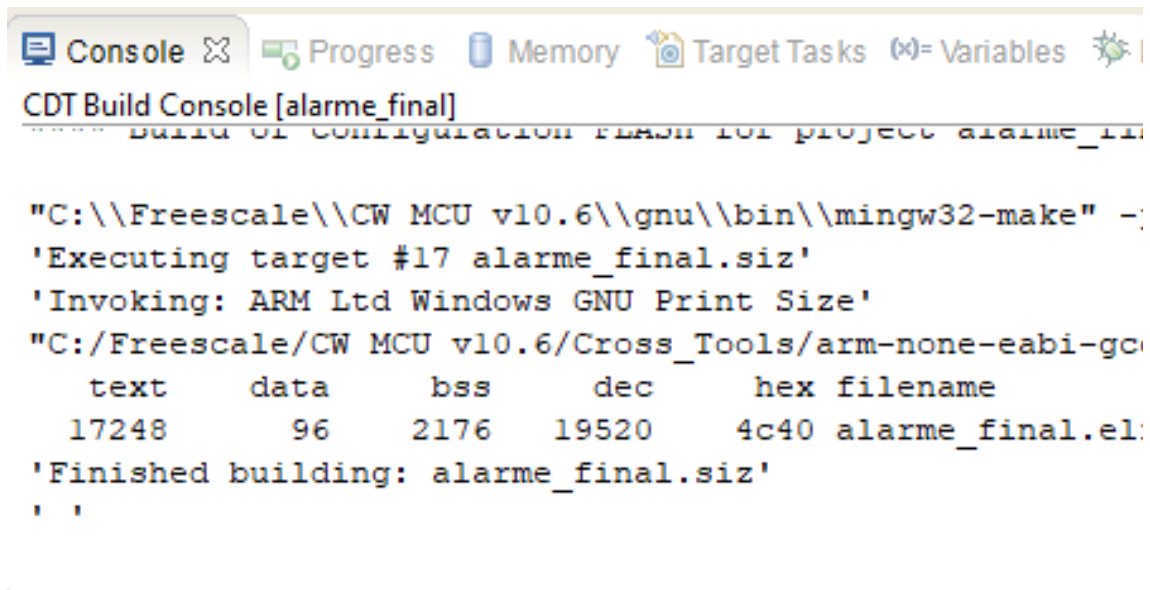
        contador = 0;
        seg_alarm = 0;
        ISR_EscreveEstado (ALARME_INTERRUPT);
        SysTick_ativaInterrupt ();
    }
    contador++;
}

void RTC_Alarm_IRQHandler (void){
    if (estado == INICIO){
        RTC_desativaAlarmIRQ ();
        SysTick_ativaInterrupt ();
        //ISR_EscreveEstado (ALARME_INTERRUPT);
    }
}
```

Figura 11: Definição das funções da interrupção

3.8 Print size

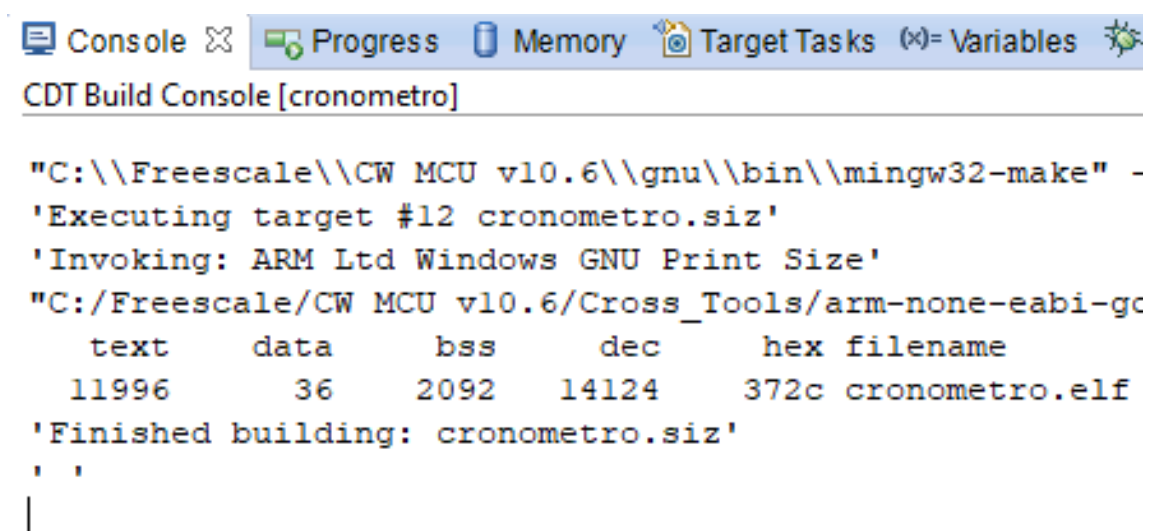
Na figura 12 temos o *print size* obtido no projeto final, comparando com a figura 13, referente ao experimento 13, podemos perceber que há um aumento nos três valores: `.txt` corresponde ao tamanho das instruções e dados constantes armazenados na memória flash, `.data` se refere a valores com inicialização armazenada na memória RAM e `.bss` variáveis sem inicialização armazenadas na memória RAM. Em detalhe, podemos comentar o grande aumento no valor para `.txt`. Como esperado, o projeto final foi o maior projeto realizado na disciplina, porque utilizou conceitos e partes do código de diversos roteiros anteriores, além de algumas funções e blocos de código novos.



```
CDT Build Console [alarme_final]
----- Build of configuration FLASH for project alarme_final -----

"C:\Freescale\CW MCU v10.6\gnu\bin\mingw32-make" -;
'Executing target #17 alarme_final.siz'
'Invoking: ARM Ltd Windows GNU Print Size'
"C:/Freescale/CW MCU v10.6/Cross_Tools/arm-none-eabi-gc
    text    data    bss    dec    hex filename
    17248     96    2176   19520    4c40 alarme_final.elf
'Finished building: alarme_final.siz'
, ,
```

Figura 12: Print size do projeto alarme_final



```
CDT Build Console [cronometro]
----- Build of configuration FLASH for project cronometro -----

"C:\Freescale\CW MCU v10.6\gnu\bin\mingw32-make" -;
'Executing target #12 cronometro.siz'
'Invoking: ARM Ltd Windows GNU Print Size'
"C:/Freescale/CW MCU v10.6/Cross_Tools/arm-none-eabi-gc
    text    data    bss    dec    hex filename
    11996     36    2092   14124    372c cronometro.elf
'Finished building: cronometro.siz'
, ,
```

Figura 13: *Print size* do projeto cronometro