Sistemas Embarcados

- Código: CSW41
- Professor: Douglas Paulo Bertrand Renaux
- Email: douglasrenaux@professores.utfpr.edu.br

- Aluno: Luka Takeshi Kato
- Email: luka@alunos.utfpr.edu.br

Laboratório 2 - Jogo de Tempo de Reação

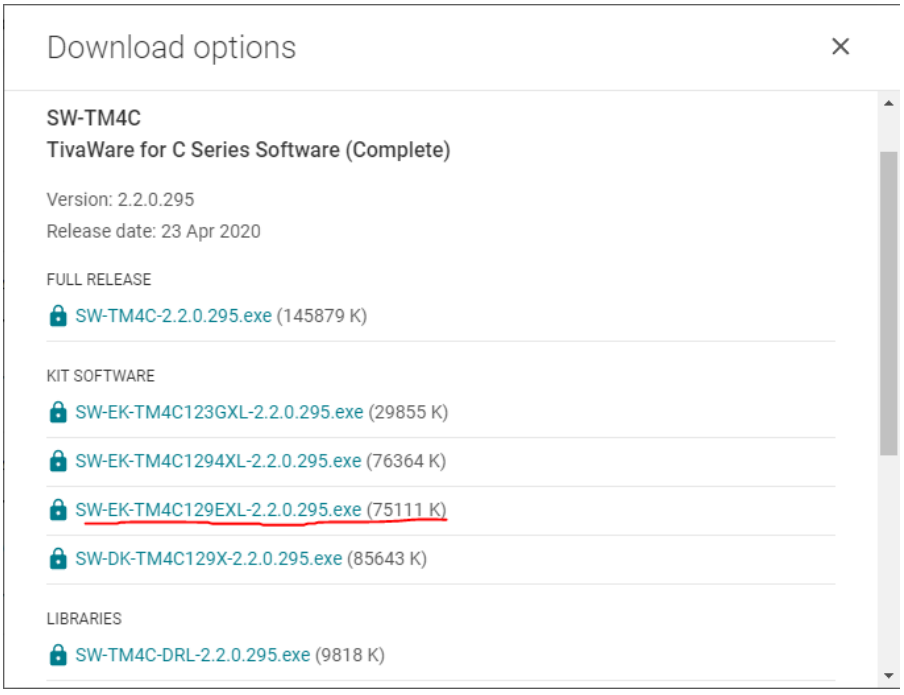
Objetivo

A partir da biblioteca TivaWare, desenvolver uma aplicação embarcada que faz uso de interrupções e interage com dispositivos de I/O (botões e leds).

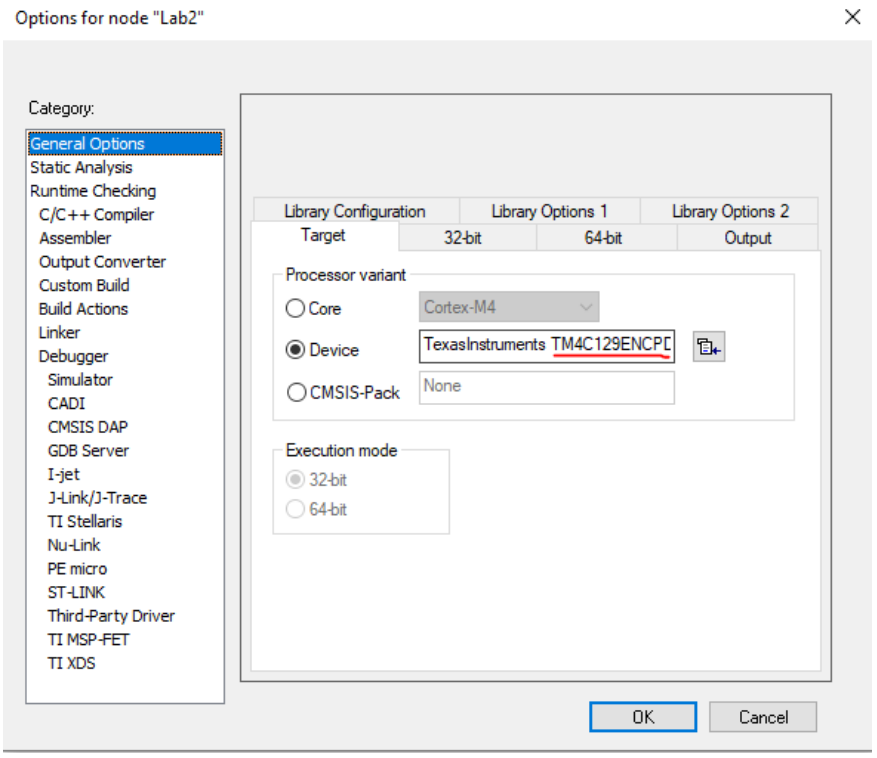
Introdução

1. Primeiro passo, no IAR foi criado o projeto do laboratório 2.

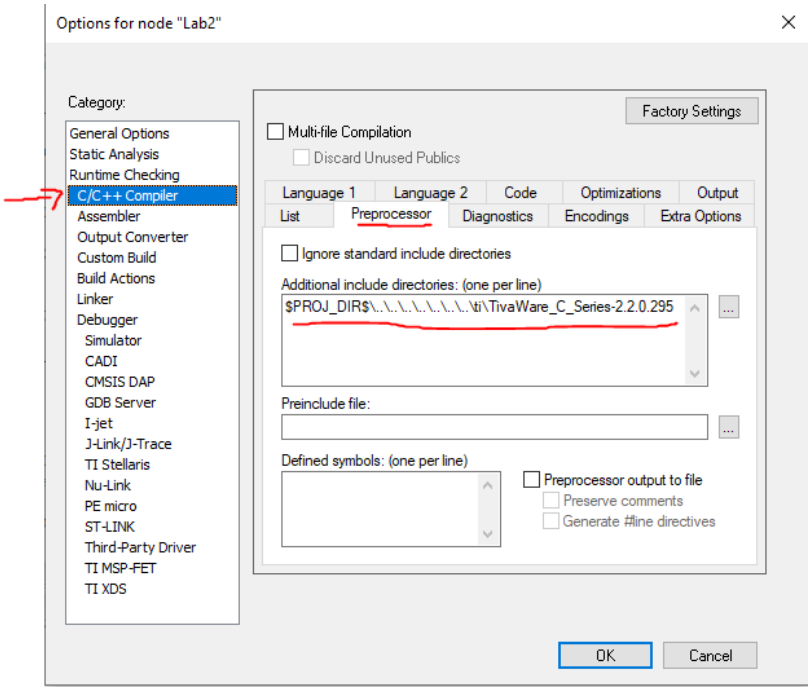
2. Obteve-se o TivaWare versão 2.2.0.295, o datasheet do processador e o manual do usuário.
OBS: Kit utilizado EK-TM4C129EXL
link: https://www.ti.com/tool/EK-TM4C129EXL



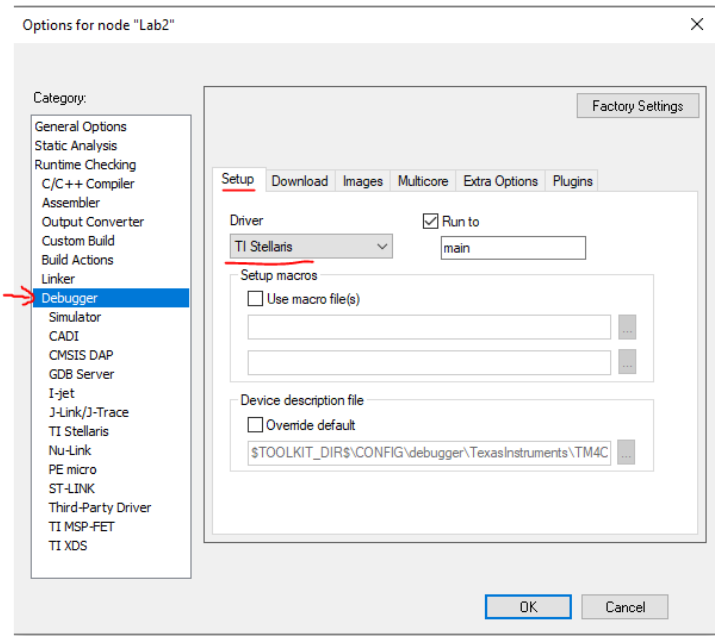
3. Com isso o projeto foi configurado para utilizar o kit, nas opções do projeto criado, foi selecionado o dispositivo.



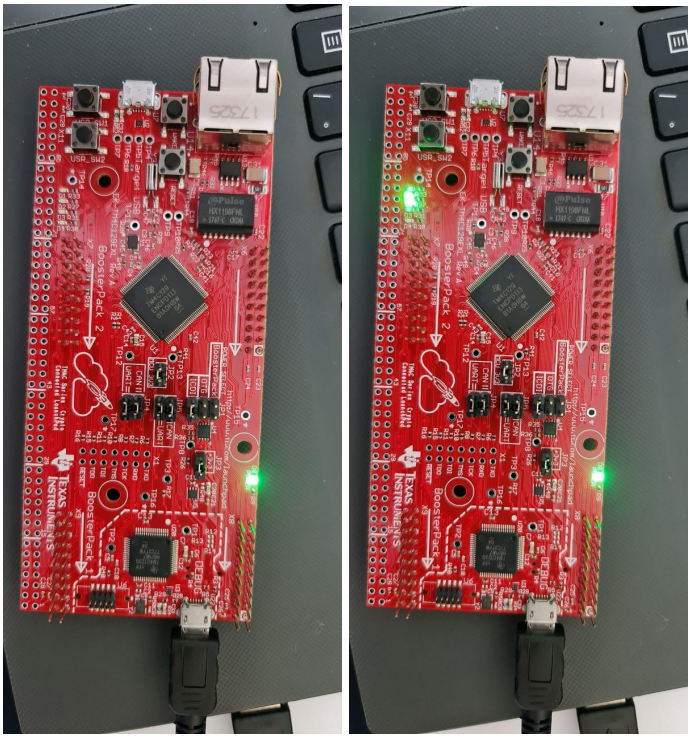
4. Em **C/C++ Compiler**, na aba **Preprocessor** foi incluído o diretório da TivaWare que foi instalado.



5. E para finalizar a configuração, em **Debugger**, na aba **Setup**, foi escolhido o driver **TI Stellaris**.



6. Com isso, foram importados os arquivos disponibilizados pelo professor, que usa como base um programa para fazer um LED da placa piscar de tempo em tempo.



7. O exemplo original do **blinky** utiliza um loop para gerar a temporização da piscada do LED. A variável **ui32Loop** é declarada como **volatile** dentro de **main()**, pois esse modificador de acesso é utilizado para indicar ao compilador que a variável especificada pode ser alterada por eventos externos, isto é, o conteúdo da variável pode ser alterado de forma não explícita. Por exemplo, uma rotina de interrupção pode alterar o valor de uma variável utilizada em outro procedimento.

8. Para fazer isso devemos primeiro inicializar o sistema com um clock conhecido. Para isso utilizamos a função **SysCtlClockFreqSet()**, então para configurar o clock para 120 MHz, podemos utilizar a função da seguinte maneira:

```
// Run from the PLL at 120 MHz.
g_ui32SysClock = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
                                     SYSCTL_OSC_MAIN |
                                     SYSCTL_USE_PLL |
                                     SYSCTL_CFG_VCO_240), 120000000);
```

9. Depois disso, utilizamos uma função para configurar o **Systick** para ele chamar uma interrupção depois de uma certa contagem de clocks.

```
int setupSystick(void) {

  // Disables the SysTick counter.
  SysTickDisable();
  //uint32_t ui32Value;

  //Sets the period of the SysTick counter.
  SysTickPeriodSet(12000000);

  SysTickIntRegister(sysTickHandler);
  SysTickIntEnable();

  // Enables the SysTick counter
  SysTickEnable();

  //ui32Value = SysTickValueGet;
  return 0;
}
```

10. Como o clock é de 120 MHz, quando utilizamos a função **SysTickPeriodSet(contagem_de_clocks)** passamos a quantidade de clocks que queremos para acontecer uma interrupção, porém segundo a documentação (SW-TM4C-DRL-UG-2.2.0.295), o número máximo de clocks que o Systick conta é de 16.777.216, ou seja, temos que fracionar em várias contagens de tempo, se quisermos contar 1 segundo. Por isso a interrupção faz um aumento em uma flag, e a cada 10 adições temos 1 segundo.

```
void sysTickHandler(void) {
  flag++;
}
```

11. E na **main()** é feito a lógica para o LED da placa piscar, a cada 3 segundos.

```
setupSystick();

while(1) {

  if(flag == 30) {
    GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_0, GPIO_PIN_0);
  }

  if(flag == 60) {
    GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_0, 0x0);
    flag = 0;
  }

}
```

Jogo de Tempo de Reação

1. Da mesma maneira que foi feita na introdução fazemos para esse laboratório. Inicializamos o 'port J' para podermos usar o botão.

```
//--------------------------------port J--------------------------------------

// Enable clock on port J
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);

// Wait until the port is ready to use
while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOJ)){}

// Disable the interrupt of pin 0
GPIOIntDisable(GPIO_PORTJ_AHB_BASE,GPIO_PIN_0);

// Register a function to call
GPIOIntRegister(GPIO_PORTJ_AHB_BASE, portJHandler);

//Configure: PortJ | Pin 0 | Strength 2 mA | Weak pull-up enabled
GPIOPadConfigSet(
  GPIO_PORTJ_AHB_BASE,
  GPIO_PIN_0,
  GPIO_STRENGTH_2MA,
  GPIO_PIN_TYPE_STD_WPU
);
```

```
//Define the pin as input
GPIOPinTypeGPIOInput(GPIO_PORTJ_BASE, GPIO_PIN_1);

// Make pin 0 rising edge triggered interrupts.
GPIOIntTypeSet(GPIO_PORTJ_AHB_BASE, GPIO_PIN_0, GPIO_LOW_LEVEL);

// Enable the pin interrupts.
GPIOIntEnable(GPIO_PORTJ_BASE, GPIO_PIN_0);
```

2. São declaradas **Flags** para a sinalização para o programa.

```
int flag = 0; // milisseconds
int portJflag = 0;
int startFlag = 0;
```

Sendo que:

- **flag**: Tempo em milissegundos do contador;
- **portJflag**: Indicador de acionamento do botão;
- **startFlag**: Indicador de início do jogo.

3. Da mesma forma que fazemos com o **Systick**, também criamos uma interrupção para o aperto do botão, onde ela configura **portJflag** como 1.

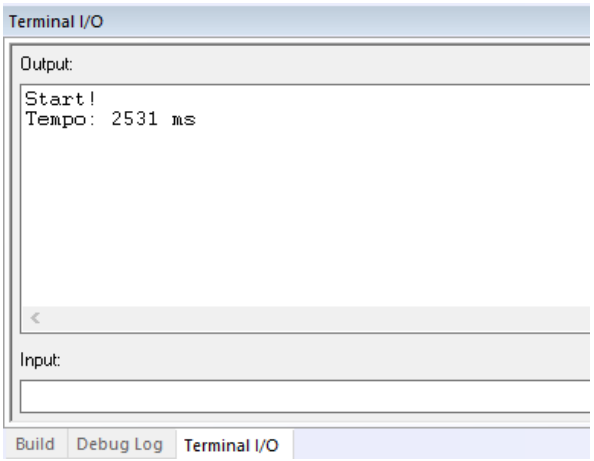
```
void portJHandler(void) {
  portJflag = 1;
}
```

4. Quando o programa é iniciado, depois de todas as configurações feitas, após 1 segundo o LED acende e é impresso na tela: `Start!`, para indicar que o jogo iniciou.

```
if((flag == 1000) && (startFlag == 0)) {
  printf("Start!\n");
  GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_0, 1);
  flag = 0;
  startFlag = 1;
}
```

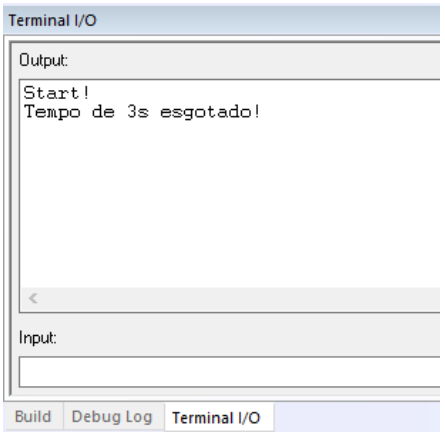
5. Quando o usuário aperta o botão, depois de iniciar o jogo, o programa é finalizado, o tempo aparece no terminal de IO e o LED apaga.

```
} else if (portJflag == 1) {

  GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_0, 0);
  SysTickDisable();
  GPIOIntDisable(GPIO_PORTJ_AHB_BASE,GPIO_PIN_0);
  printf("Tempo: %i ms\n", flag);
  portJflag = 0;
  return 0;

}
```



6. E quando passa de 3 segundos sem apertar o botão o programa finaliza, dizendo que o tempo foi esgotado e o LED apaga.

```
if(flag == 3000) {

  GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_0, 0);
  printf("Tempo de 3s esgotado!\n");
  flag = 0;
  return 0;
```



Referências

- Tiva™ TM4C1294NCPDT Microcontroller - DATA SHEET (Funcionamento do clock da placa);

- SW-TM4C-DRL-UG-2.2.0.295 - TivaWare™ Peripheral Driver Library (Systick API Functions);

- TM4C Series TM4C129E Crypto Connected EK-TM4C129EXL - User's Guide (Verificação dos pinos para o botão e para o LED).