Sistemas Embarcados

Periféricos no Contiki: ADC, PWM e Low-Power Mode

Prof. Guilherme de S. Peron

peron@utfpr.edu.br

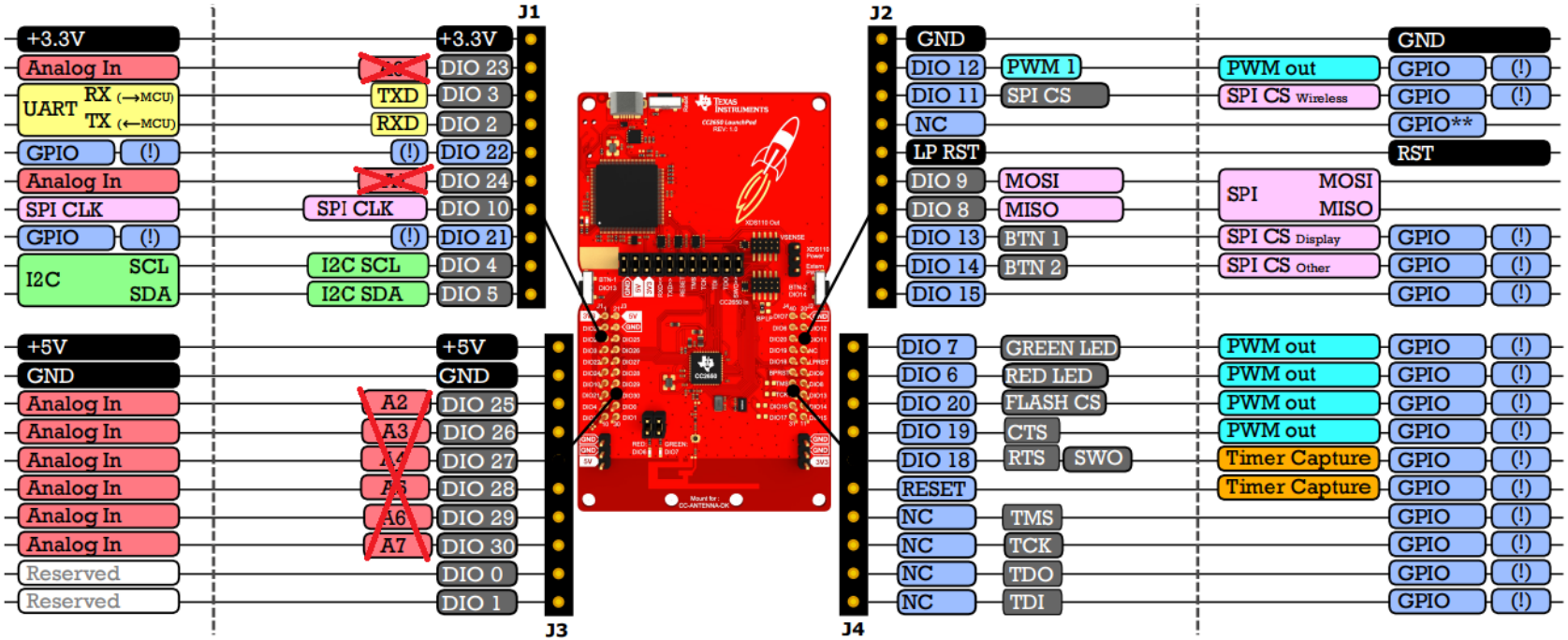
Curso de Especialização em Internet das Coisas (CEIoT)

25 de Agosto de 2018

Recapitulando GPIO

GPIO no Contiki

- Pinos de DIO (Digital I/O) disponíveis:



GPIO no Contiki

- Endereçamento dos pinos disponível em: contiki/platform/srf06-cc26xx/launchpad/cc2650/board.h

```
#define BOARD_IOID_DIO12 IOID_12
#define BOARD_IOID_DIO15 IOID_15
#define BOARD_IOID_DIO21 IOID_21
#define BOARD_IOID_DIO22 IOID_22
#define BOARD_IOID_DIO23 IOID_23
#define BOARD_IOID_DIO24 IOID_24
#define BOARD_IOID_DIO25 IOID_25
#define BOARD_IOID_DIO26 IOID_26
#define BOARD_IOID_DIO27 IOID_27
#define BOARD_IOID_DIO28 IOID_28
#define BOARD_IOID_DIO29 IOID_29
#define BOARD_IOID_DIO30 IOID_30
```

Inicialização dos GPIO

- No Contiki, é necessário configurar os pinos GPIO que serão utilizados:
  `http://dev.ti.com/tirex/content/simplelink_cc13x0_sdk_1_40_00_10/docs/driverlib_cc13xx_cc26xx/cc26x0/driverlib/group__peripheral__group.html`
- Inicialização dos pinos DIO27 e DIO30 como saídas:
  `IOCPinTypeGpioOutput(IOID_27);`
  `IOCPinTypeGpioOutput(IOID_30);`
- Inicialização dos pinos DIO29 como entrada:
  `IOCPinTypeGpioInput(IOID_29);`
- Para alterar o estado dos pinos existe uma série de funções disponíveis, por exemplo: `GPIO_clearDio()` e `GPIO_setDio()`
- Para ler o estado dos pinos: `GPIO_readDio()`

Exemplo de Contador de 2 bits

```
#include "contiki.h"
#include "sys/etimer.h"
#include "sys/ctimer.h"

#include "dev/leds.h"
#include "dev/watchdog.h"
#include "dev/adc-sensor.h"

#include "random.h"
#include "button-sensor.h"
#include "board-peripherals.h"
#include "lib/sensors.h"
#include "ti-lib.h"

#include <stdio.h>
#include <stdint.h>"

static struct etimer et_gpio;

PROCESS(gpio_process, "GPIO process");
AUTOSTART_PROCESSES(&gpio_process)
```

Exemplo de Contador de 2 bits

```
PROCESS_THREAD(gpio_process, ev, data)
{
    static int8_t counter = 0;
    PROCESS_BEGIN();
    etimer_set(&et_gpio, 1*CLOCK_SECOND);

    IOCPinTypeGpioOutput(IOID_27);
    IOCPinTypeGpioOutput(IOID_30);
    GPIO_clearMultiDio(1<<IOID_27 | 1<<IOID_30);

    while(1) {
        PROCESS_WAIT_EVENT();
        if(ev == PROCESS_EVENT_TIMER) {
            counter++;
            if(counter & 0x1)
                GPIO_setDio(IOID_27);
            else
                GPIO_clearDio(IOID_27);

            if(counter & 0x2)
                GPIO_setDio(IOID_30);
            else
                GPIO_clearDio(IOID_30);

            etimer_reset(&et_gpio);
        }
    }
    PROCESS_END();

}
```

ADC – Analog-to-Digital Converter

ADC no Contiki

- O ADC é um interface de sensor ativa do Contiki. Seu código de operação pode ser encontrado em: `contiki/cpu/cc26xx-cc13xx/dev/adc-sensor.c`

- Repare que ao final do código, a seguinte estrutura é definida:

```
SENSORS_SENSOR(adc_sensor, ADC_SENSOR, value, config, status)
```

em que `ADC_SENSOR` endereça o ADC que iremos utilizar.

ADC no Contiki

- Por padrão, o ADC não está ativado no código do Contiki.
  Devemos ativá-lo no código que controla os sensores da lauchpad:
  `/contiki/platform/srf06-cc26xx/launchpad/launchpad-sensors.c`

- Incluir:

```
#include "dev/adc-sensor.h"
```

- Adicionar o `adc_sensor` na estrutura:

```
SENSORS(...)
```

ADC no Contiki

- Para acessar o ADC, necessitamos das seguintes instruções dentro da thread que irá trabalhar com o ADC:

```
static struct sensors_sensor *sensor

sensor = sensors_find(ADC_SENSOR);
```

- Além disso, serão necessários os seguintes includes para trabalhar com o ADC:

```
#include "dev/adc-sensor.h"

#include "lib/sensors.h"
```

ADC no Contiki

- A numeração das portas analógicas está errada no folheto que vem na caixa do kit. A correta está no *User Guide* do CC2650:

I/O Pin Mapping www.ti.com

11.8 I/O Pin Mapping

Table 11-2 shows the I/O pin mapping for different package types.

Table 11-2. CC26x0 and CC13x0 Family Pin Mapping

Package Type								Sensor Controller		Drive Strength	JTAG
7 × 7 QFN (RGZ)		5 × 5 QFN (RHB)		WCSP (YFV)		4 × 4 QFN (RSM)		Analog Capable	AUX I/O		
Pin	DIO	Pin	DIO	Pin	DIO	Pin	DIO				
43	30	27	14					yes	0	2 mA / 4 mA	
42	29	26	13	B3	13			yes	1	2 mA / 4 mA	
41	28	25	12	D4	12			yes	2	2 mA / 4 mA	
40	27	24	11	B2	11	26	9	yes	3	2 mA / 4 mA	
39	26	22	9	A1	9	25	8	yes	4	2 mA / 4 mA	
38	25	23	10	C2	10	24	7	yes	5	2 mA / 4 mA	
37	24	21	8	D3	8	23	6	yes	6	2 mA / 4 mA	
36	23	20	7	D2	7	22	5	yes	7	2 mA / 4 mA	
32	22									2 mA / 4 mA	
31	21									2 mA / 4 mA	
30	20									2 mA / 4 mA	
29	19									2 mA / 4 mA	
28	18									2 mA / 4 mA	

Passo a Passo do ADC – a cada loop

1) Ativar ADC: `SENSORS_ACTIVATE(*sensor);`

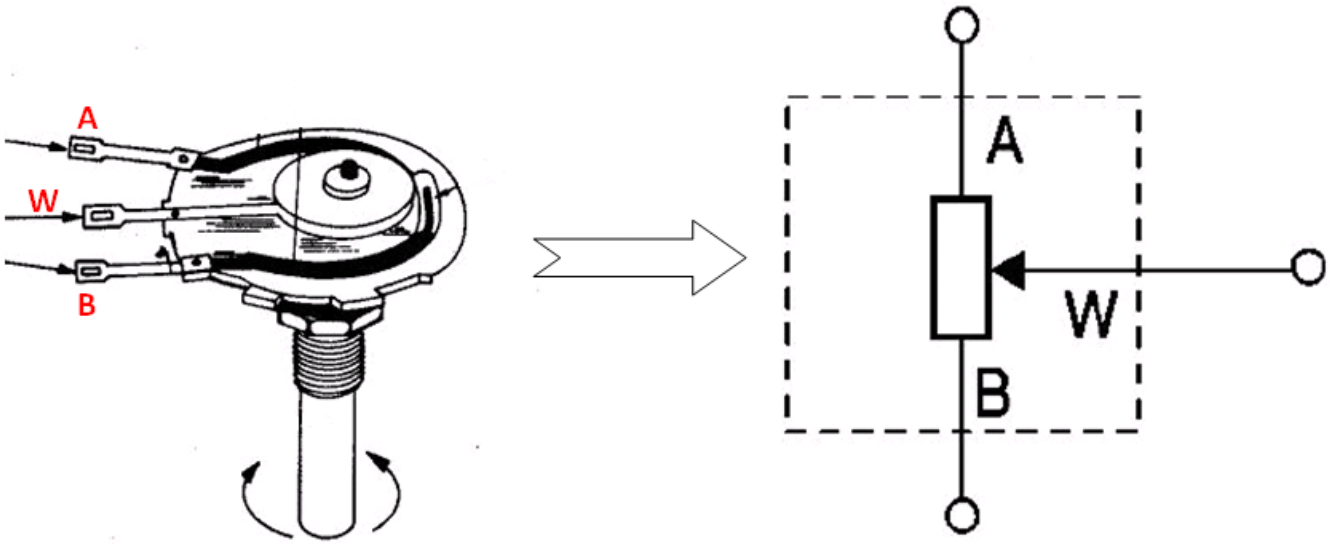
2) Configurar canal a ser utilizado:
   `configure(ADC_SENSOR_SET_CHANNEL, ADC_COMPB_IN_AUXIOX);`
   `X` é a porta analógica a ser utilizada $\rightsquigarrow$ escolher alguma

3) O valor lido pelo ADC (em microvolts) estará disponível na estrutura `sensor->value(ADC_SENSOR_VALUE);`

4) Desativar ADC: `SENSORS_DEACTIVATE`

Atividade 1

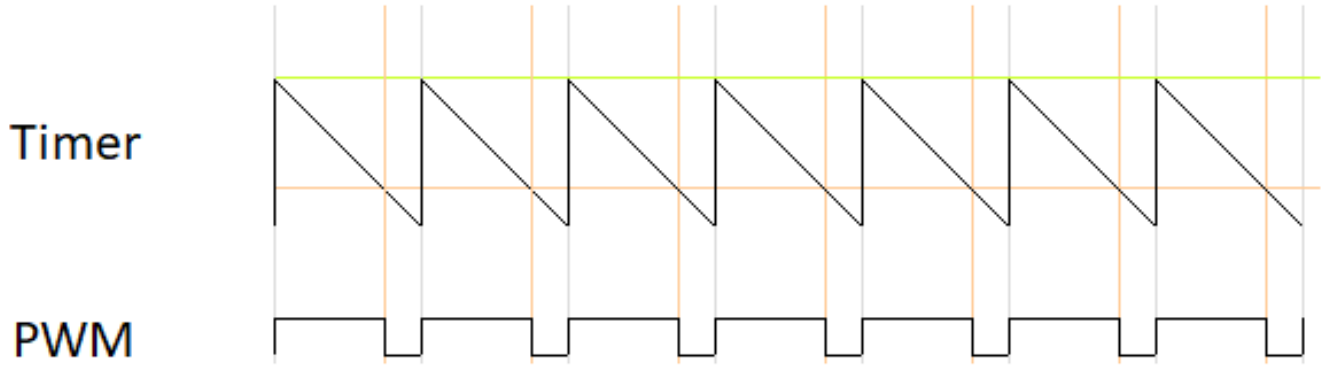
- Usar o ADC para ler a tensão em um potenciômetro. Crie um contador que lê o ADC periodicamente a cada alguns segundos. A cada evento de contagem do timer, realize os 4 passos descritos no slide anterior.



PWM – Pulse Width Modulation

PWM no Contiki

- O PWM no CC2650 usa o Timer A para funcionar



- O valores alto e de comparação são definidos pelas seguintes funções:

```
ti_lib_timer_load_set();

ti_lib_timer_match_set();
```

Função de Inicialização

```
int16_t pwminit(int32_t freq)
{
    uint32_t load = 0;

    ti_lib_ioc_pin_type_gpio_output(IOID_21);
    leds_off(LEDS_RED);

    /* Enable GPT0 clocks under active mode */
    ti_lib_prcm_peripheral_run_enable(PRCM_PERIPH_TIMER0);
    ti_lib_prcm_load_set();
    while(!ti_lib_prcm_load_get());

    /* Drive the I/O ID with GPT0 / Timer A */
    ti_lib_ioc_port_configure_set(IOID_21, IOC_PORT_MCU_PORT_EVENT0,
                                  IOC_STD_OUTPUT);

   /* GPT0 / Timer A: PWM, Interrupt Enable */
    ti_lib_timer_configure(GPT0_BASE,
    TIMER_CFG_SPLIT_PAIR | TIMER_CFG_A_PWM | TIMER_CFG_B_PWM);
```

Função de Inicialização

```
    /* Stop the timers */
    ti_lib_timer_disable(GPT0_BASE, TIMER_A);
    ti_lib_timer_disable(GPT0_BASE, TIMER_B);

    if(freq > 0) {
        load = (GET_MCU_CLOCK / freq);

        ti_lib_timer_load_set(GPT0_BASE, TIMER_A, load);
        ti_lib_timer_match_set(GPT0_BASE, TIMER_A, load-1);

        /* Start */
        ti_lib_timer_enable(GPT0_BASE, TIMER_A);

    }
    return load;
}
```

Thread de PWM

```
PROCESS_THREAD(pwm_process, ev, data)
{
    static int16_t current_duty = 0;
    static int16_t loadvalue;

    PROCESS_BEGIN();

    loadvalue = pwminit(5000);

    while(1) {
        PROCESS_WAIT_EVENT();
        ...



    }
    PROCESS_END();
}
```

Thread de PWM

- Um novo duty cycle é feito da forma inversa:

```
ti_lib_timer_match_set(GPTO_BASE, TIMER_A, loadvalue - ticks);
```

- em que `ticks` é uma parcela da contagem dos clocks em relação ao duty cycle desejado:

```
ticks = (current_duty * loadvalue) / 100;
```

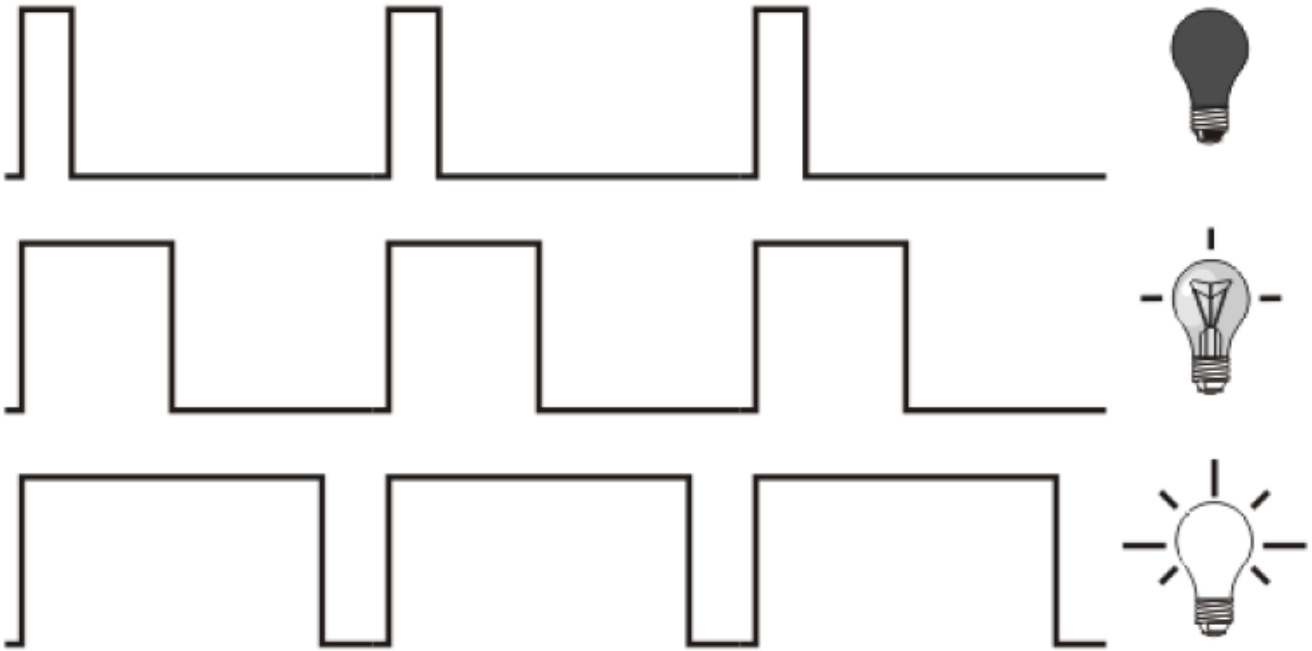
Thread de PWM

- Incluir as libs:

```
#include “ti-lib.h”
#include “sys/etimer.h”
#include “sys/ctimer.h”
```

Atividade 2

- Ligar um LED na saída do PWM e observar a variação de luminosidade com o duty cycle do PWM. Para alterar o duty cycle, configure os dois botões do kit para aumentar ou diminuir a luminosidade em 10% a cada vez que o botão é pressionado.



Atividade 3

- Por que o PWM não funciona corretamente no CC2650?