Data: 19/03/2022
Descrição: Documentação do laboratório 2 de Sistemas Embarcados
Professor: Douglas Paulo Bertrand Renaux
Alunos: Hadryan Salles, Luciano Bonzatto Junior

Especificação do programa:

1. O jogo do tempo de resposta deve utilizar um Led, um *push-button* e um console
2. A operação do jogo, depois de ligado, é:
    a. Esperar 1 segundo
    b. Liga o Led
    c. Começa a contar o *timer*
    d. Espera o jogador apertar o *push-button*
    e. Para o timer, salvando o tempo percorrido
    f. Desliga o led
    g. Guarda o tempo de reação em uma variável
    h. Exibe o tempo de reação no console
    i. Volta para a etapa *a*
3. Descartar tempos de reação acima de 3 segundos

Estudo do Hardware:

1. Periféricos presentes:

Na prática foi utilizado apenas dois periféricos mas na placa possuem vários outros como o display LCD, Ethernet e bluetooth.

2. *Push-buttons*:

Na placa existem dois push-buttons de uso geral (S4 e S5) que são ligados nos pinos 6 e 5 do port 0 do GPIO, sendo que cada um deles pode acionar as interrupções IRQ11 e IRQ10, respectivamente.

3. LEDs:

Na placa existem três LEDs que são controlados pelo GPIO sendo acionados com nivel lógico 0 e desligados com nível lógico 1.

LED1: led verde ligado ao pino P6.0
LED2: led vermelho ligado ao pino P6.1
LED3: led amarelo ligado ao pino P6.2

4. *Clock*:

O circuito de *clock* pode ser configurado de maneira a selecionar diferentes fontes de *clock* e assim definir diferentes frequências:

Table 9.2 Clock generation circuit specifications (internal clock) (1/3)

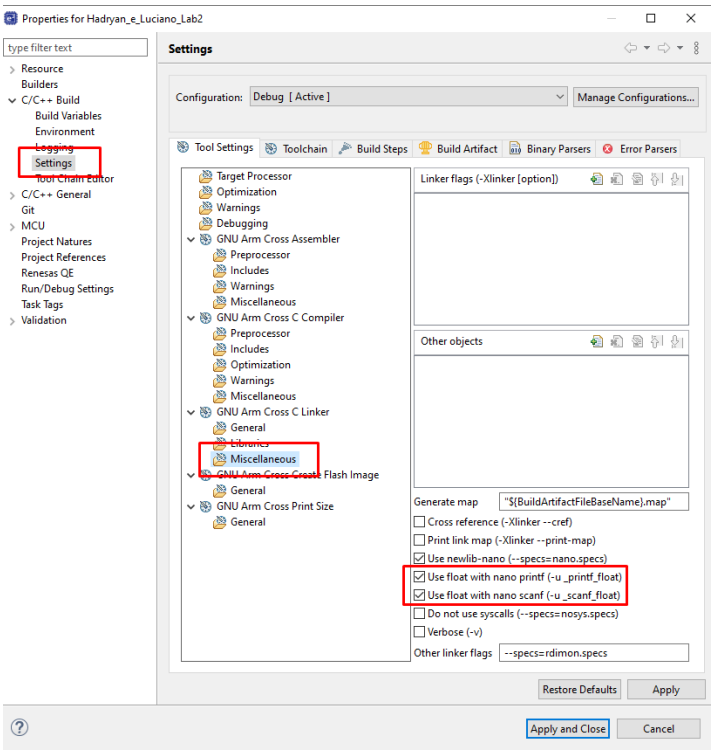
Item	Clock Source	Clock Supply	Specification
System clock (ICLK)	MOSC/SOSC/HOCO/MOCO/ LOCO/PLL	CPU, DTC, DMAC, ROM, RAM	Up to 240 MHz Division ratio: 1/2/4/8/16/32/64
Peripheral module clock A (PCLKA)	MOSC/SOSC/HOCO/MOCO/ LOCO/PLL	Peripheral module (ETHERC, EDMAC, USB2.0 HS, QSPI, SPI, SCIF, TSIP, Graphics LCD, SDHI, CRC, JPEG Engine, DRW, IrDA, GPT Bus-clock, Standby SRAM)	Up to 120 MHz Division ratio: 1/2/4/8/16/32/64
Peripheral module clock B (PCLKB)	MOSC/SOSC/HOCO/MOCO/ LOCO/PLL	Peripheral module (WDT, IWDT, RTC, IIC, SSI, SRC, DOC, CAC, CAN, ADC12, DAC12, POEG, AMI, TSN, SCI)	Up to 60 MHz Division ratio: 1/2/4/8/16/32/64
Peripheral module clock C (PCLKC)	MOSC/SOSC/HOCO/MOCO/ LOCO/PLL	Peripheral module (ADC unit 0, unit 1 (only HM))	Up to 60 MHz Division ratio: 1/2/4/8/16/32/64
Peripheral module clock D (PCLKD)	MOSC/SOSC/HOCO/MOCO/ LOCO/PLL	Peripheral module (GPT Count-clock)	Up to 120 MHz Division ratio: 1/2/4/8/16/32/64
FlashIF clock (FCLK)	MOSC/SOSC/HOCO/MOCO/ LOCO/PLL	FlashIF	4 MHz to 60 MHz (P/E) Up to 60 MHz (Read) *1 Division ratio: 1/2/4/8/16/32/64

5. *Timers*:

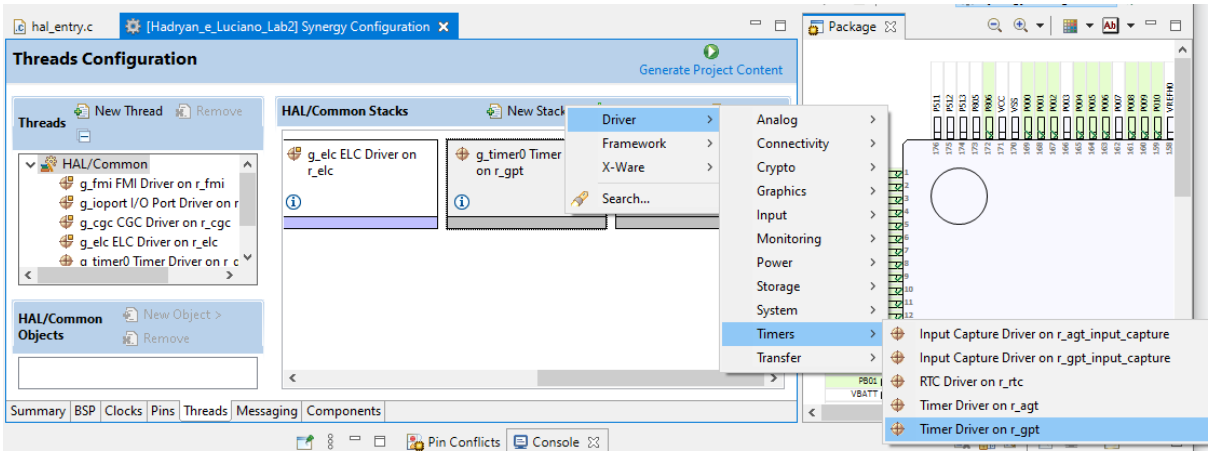
Os *timers* da placa utilizam o *General PWM Timer* que são alimentados com o PCLKA de 120 MHz, de acordo com a tabela do *clock*. Existem 14 canais de timer sendo 4 melhorados e de alta resolução, 4 melhorados e mais 6 convencionais.

Estudo do *software*:

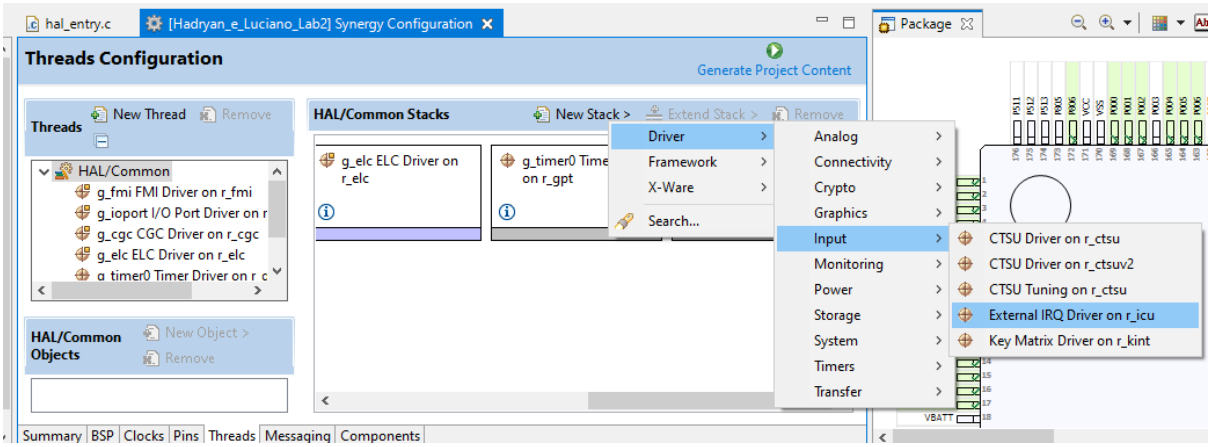
1. Para imprimir *floats* no console de depuração:



2. Para criar uma *thread* para o *timer*:



3. Para criar uma *thread* para o *switch button*:



4. Funções da API relacionadas ao *timer*: (fonte:https://www.renesas.com/us/en/document/apn/gpt-hal-module-guide-application-project?language=en)

Function Name	Example API Call and Description
.open	`g_timer0.p_api->open(g_timer0.p_ctrl, g_timer0.p_cfg)` Initial configuration.
.stop	`g_timer0.p_api->stop(g_timer0.p_ctrl)` Stop the counter.
.start	`g_timer0.p_api->start(g_timer0.p_ctrl)` Start the counter.
.reset	`g_timer0.p_api->reset(g_timer0.p_ctrl)` Reset the counter to the initial value.
.counterGet	`g_timer0.p_api->counterGet(g_timer0.p_ctrl, &value)` Get current counter value and store it in provided pointer value.
.periodSet	`g_timer0.p_api->periodSet(g_timer0.p_ctrl, period, unit)` Set the time until the timer expires.
.dutyCycleSet	`g_timer0.p_api->dutyCycleSet(g_timer0.p_ctrl, period, unit, pin)` Sets the time until the duty cycle expires.
.infoGet	`g_timer0.p_api->infoGet(g_timer0.p_ctrl, &info)` Get the time until the timer expires in clock counts and store it in provided pointer info.
.close	`g_timer0.p_api->close(g_timer0.p_ctrl)` Allows driver to be reconfigured and may reduce power consumption.
.versionGet	`g_timer0.p_api->versionGet(&version)` Get version and store it in provided pointer version.

Note: For details on operation and definitions for the function data structures, typedefs, defines, API data

5. Assinatura da função *callback* chamada pela *thread* do *timer*:

```
void f(timer_callback_args_t *);
```

6. Assinatura da função *callback* chamada pela *thread* do *switch button*:

```
void f(external_irq_callback_args_t *);
```

7. Para ajustar a interrupção o *timer*:
    a. utilizar a função *open* da API para configurar o *timer*
    b. utilizar a função *start* da API para iniciar o contador do *timer*
    c. utilizar a função *reset* da API para resetar o contador do *timer*
    d. utilizar a função *periodSet* da API para ajustar o periodo do *timer*, de acordo com a tabela:

Table 3 Timer period calculation

Timer Units	Formula
TIMER_UNIT_PERIOD_NSEC	Counts = (period * clk_freq_hz) / 1000000000
TIMER_UNIT_PERIOD_USEC	Counts = (period * clk_freq_hz) / 1000000
TIMER_UNIT_PERIOD_MSEC	Counts = (period * clk_freq_hz) / 1000
TIMER_UNIT_PERIOD_SEC	Counts = (period * clk_freq_hz)
TIMER_UNIT_FREQUENCY_HZ	Counts = (clk_freq_hz) / period
TIMER_UNIT_FREQUENCY_KHZ	Counts = (clk_freq_hz) / 1000 * period

8. Para ajustar a interrupção do *push-button*:
    a. utilizar a função *open* da API para configurar o *push-button*

Design

1. *Push-button* utilizando IRQ *driver* em r_icu no canal 11 (IRQ 11), conectado ao S4 com prioridade 5.
2. Led utilizado: Led1
3. *Timer Driver* em r_gpt no canal 8 configurado com prioridade 4 no modo *One Shot.*