

Documentação de Projeto – Parte 2

Design, Estudo da Plataforma

Projeto: Sistema de controle de temperatura veicular

Autores: Felipe Avelino Pantoja Rêgo

Versão: 16-Nov-2022

Parte 2a - Design

1 Introdução

Nesse documento vamos continuar a discussão sobre projeto final para disciplina de Sistemas Embarcados, aprofundando um pouco mais a solução apresentada no primeiro documento, em questão de design e estudo da plataforma.

O projeto selecionado foi o de controle de temperatura veicular, através do kit de desenvolvimento Renesas SK-S7G2.

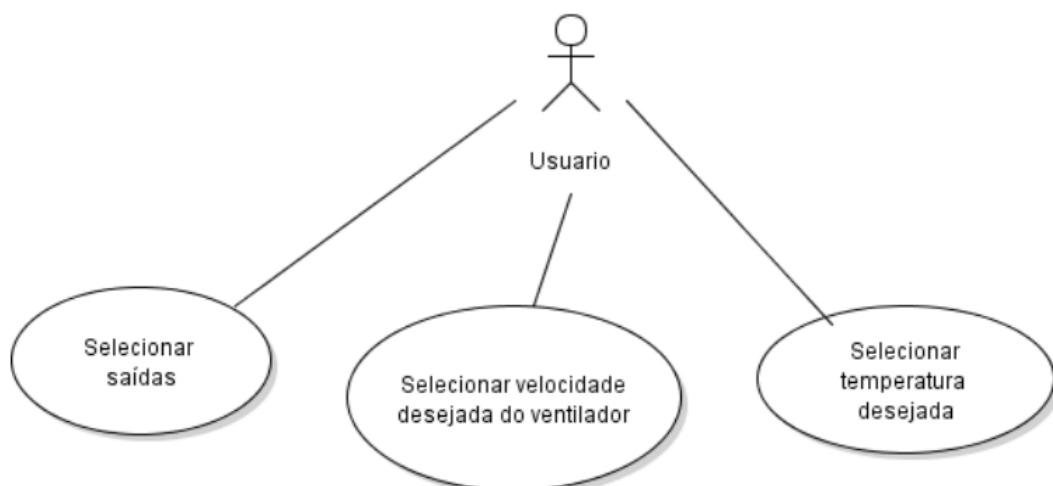
Relembrando um pouco, iremos desenvolver um sistema que estará no microcontrolador que estará conectado no carro e receberá entradas através interface com o usuário e controlará a temperatura do veículo atuando nos seguintes componentes: um ventilador, três saídas de ar, uma válvula de ar frio e uma com ar quente.

Para tal utilizaremos conhecimentos vistos em sala de aula, além dos manuais do kit de desenvolvimento.

2 Arquitetura Funcional

Como as soluções foram colocadas no documento passado, aqui vamos relembrá-las.

Diagrama de casos de uso do projeto:



De forma bem sucinta, o sistema tem que fazer as seguintes funcionalidades:

- Controle de temperatura de 16°C até 30°C, variando de meio em meio grau.
- Controle de velocidade de um ventilador interno, variando em 5 velocidades
- Controle de 3 saídas de ar, mas com 5 estados possíveis
- Controlar abertura de válvulas de ar frio e quente

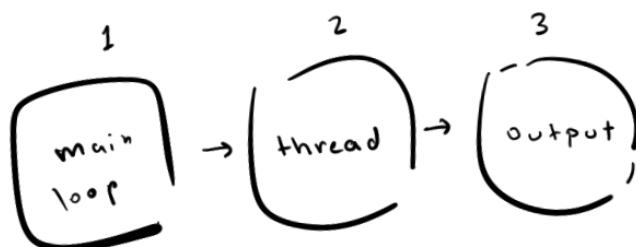
Projeto – Sistema de controle de temperatura veicular

Também existem alguns requisitos para se levar em consideração:

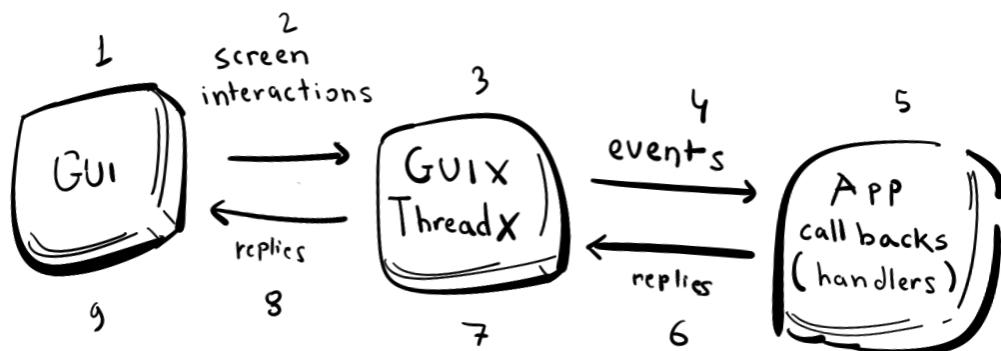
- 5 segundos após o último ajuste, o sistema faz as alterações necessárias para atender à demanda.
- As temperaturas apresentadas pelo sensor devem variar de 10°C a 50°C.
- Se quisermos esfriar o sistema, a temperatura da mistura de ar das válvulas tem que ser 2°C a menos da temperatura desejada.
- Se quisermos aquecer o sistema, a temperatura da mistura de ar das válvulas tem que ser 2°C a mais da temperatura desejada.

Relembrando os requisitos dos stakeholders, podemos começar a discutir melhor como se dará a implementação do projeto.

Em um programa tradicional sem interações com o usuário funcionaria da seguinte forma:

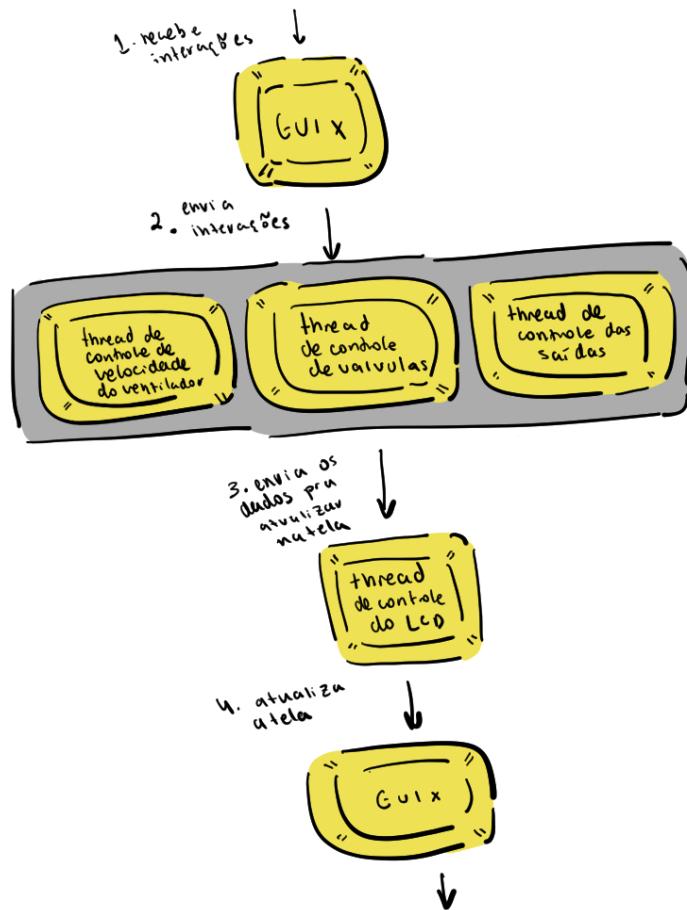


Já um programa orientado a eventos, como GUIX junto com ThreadX, funcionaria da seguinte forma:

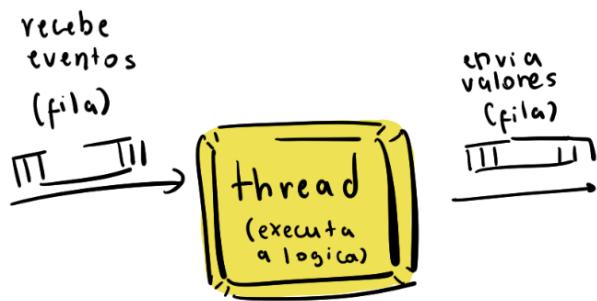


Então é necessário adaptar o nosso código pro que faríamos em um código sem interação com o usuário.

Projeto – Sistema de controle de temperatura veicular



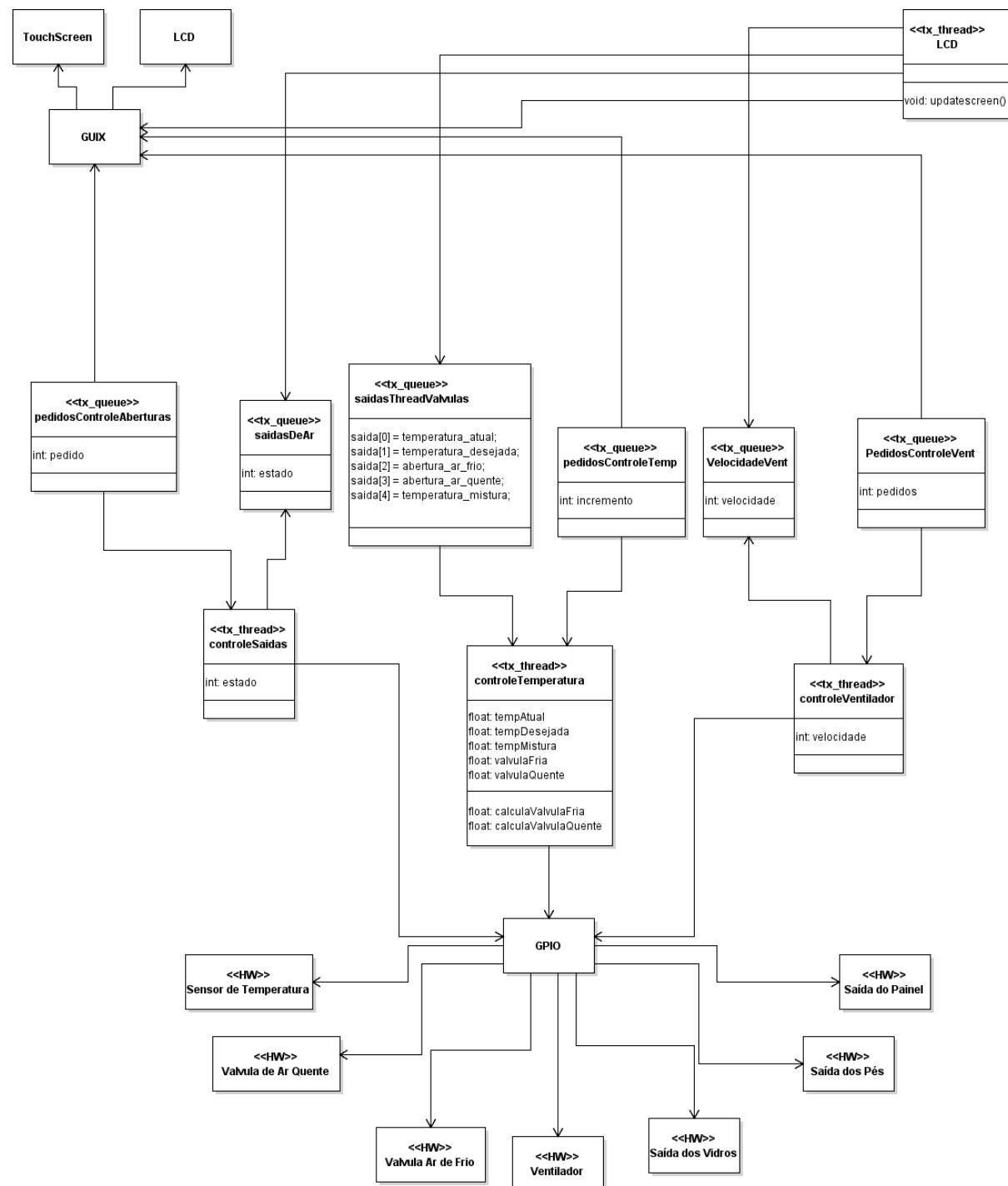
Basicamente seria como se ao invés de invocar os métodos da uma classe, nós mandamos valores para a respectiva **fila de mensagens de pedidos** de cada thread, dependendo do evento e um momento antes de atualizarmos novamente a tela, nós recebemos as atualizações dos valores, através de **outra fila de mensagens**.



Assim uma maneira de se fazer é cada thread (dependendo do caso algumas threads poderão ter mais) ter uma fila de entrada de eventos e uma fila de saída com valores após alguma lógica ser feita. Basicamente **teríamos apenas 4 threads rodando**, a do controle das **válvulas** (que vai controlar de fato a temperatura do carro), do controle de velocidade do **ventilador**, do controle de **saídas de ar** (lembrando que essa ditará o estado do sistema também, dizendo se ele está ligado ou desligado) e uma thread apenas pra **atualizar o LCD** (implementado usando o código fornecido para lab6 pra Renesas).

3 Arquitetura Física

Após a explicação da sessão anterior, aqui está uma versão mais detalhada da arquitetura, mostrando o que é thread e o que é fila:



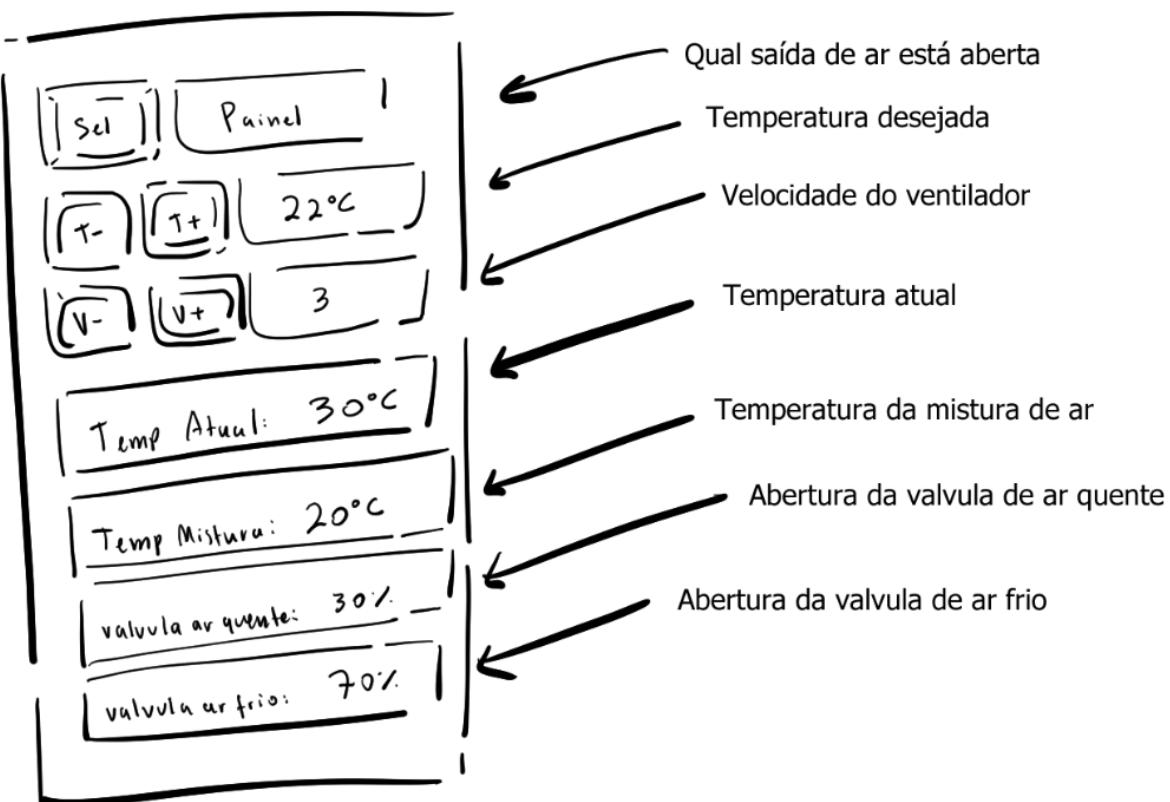
Alguns detalhes ficaram de fora, como por exemplo o GUIX precisa do DTC e do ELC para ler os dados do touchscreen que são mandados por I2C, etc. Detalhes como esses podem ser vistos quando criamos a nossa própria pilha no SSP.

SSP cria sozinho as filas, na pasta “synergy_gen”, nós só precisamos fazer os sends e receives nos nossos códigos das threads. Idealmente cada variável que trabalharemos terá sua própria fila, mas nem porém nem todas as filas necessitam de uma thread só para elas. Além disso, o SSP também gera códigos facilmente de threads.

Também incorporando fila de pedido de alteração para cada variável. Exceto pro caso da thread de controle de válvulas, como ela lidará com várias saídas, é mais interessante usar uma fila que envia um vetor de floats.

4 Interface com o Usuário

Abaixo um esboço de como seria a interface, que será feita no GUIX Studio.



Desta forma, todas as informações solicitadas são mostradas na tela como estados dos atuadores e a temperatura da mistura de ar quente e frio.

Além disso serão apresentados os botões para operação do sistema:

- Botões T+ e T-.
- Botões V+ e V-.
- Botão Sel (que servirá para ligar ou desligar o sistema além de selecionar a saída de ar).

5 Mapeamento da Arquitetura Funcional à Arquitetura Física

Normalmente nesse tipo de tabela as linhas são elementos da arquitetura física e as colunas são elementos da arquitetura funcional, só trocamos as linhas e as colunas para economizar um pouco de espaço.

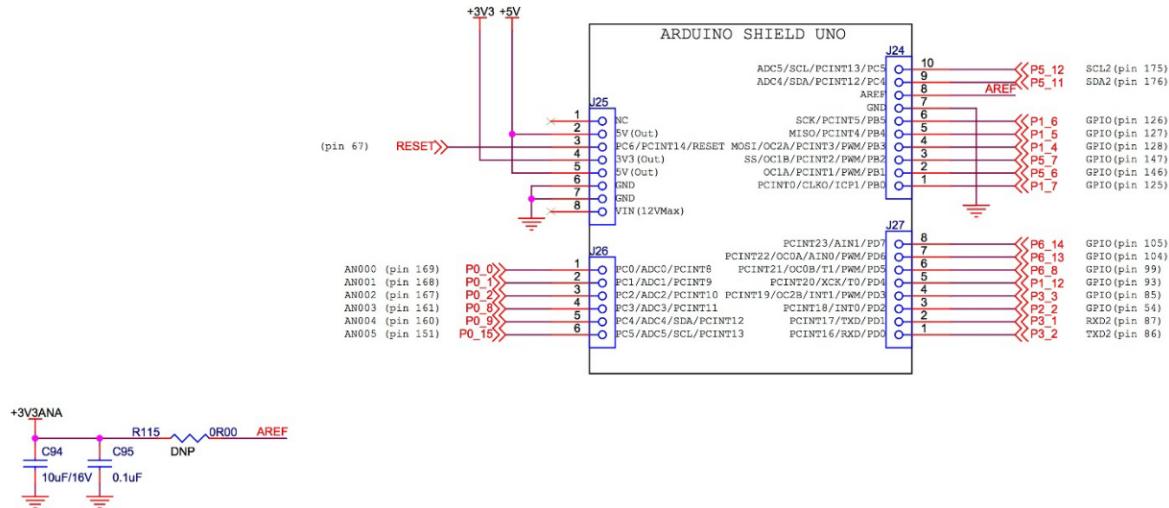
| | GPIO | GUI X | Thread X | Válvulas de ar quente | Válvulas de ar frio | Saídas de Ar dos Painel | Saídas de Ar dos Vidros | Saídas de Ar dos Pés | Ventilador | Sensor de temperatura |
|--------------------------------------------------|------|-------|----------|-----------------------|---------------------|-------------------------|-------------------------|----------------------|------------|-----------------------|
| Medir temperatura | X | | | | | | | | | X |
| Escolher a temperatura desejada | | X | X | | | | | | | |
| Escolher a velocidade desejada para o ventilador | | X | X | | | | | | | |
| Alterar a velocidade do ventilador | X | | X | | | | | | X | |
| Escolher quais saídas de ar abrir ou fechar | | X | X | | | X | X | X | | |
| Abrir ou fechar saídas de ar | X | | X | | | X | | | | |
| Abrir ou fechar válvulas | X | | X | X | X | X | | | | |

6 Arquitetura do Hardware

O hardware utilizado será a placa Renasas SK-S7G2, temos um display Lcd ligado via interface SCI e o touch screen via I2C. Estes dois componentes nos darão os botões e a visualização necessária dos componentes, além disso temos também um sensor de temperatura que no caso será substituído por um potenciômetro. As saídas de ar, válvulas e ventilador não serão implementados.

O SK-S7G2 também inclui um Shield de Arduino, expandindo as funcionalidades da placa. A interface com as portas J24, J25, J26 e J27.

Projeto – Sistema de controle de temperatura veicular



Retirado da Pagina 11 da documentação do Renesas Synergy Software Package:

<https://docs.rs-online.com/d3d3/0900766b81513ff0.pdf>

Table 13: Pin connections

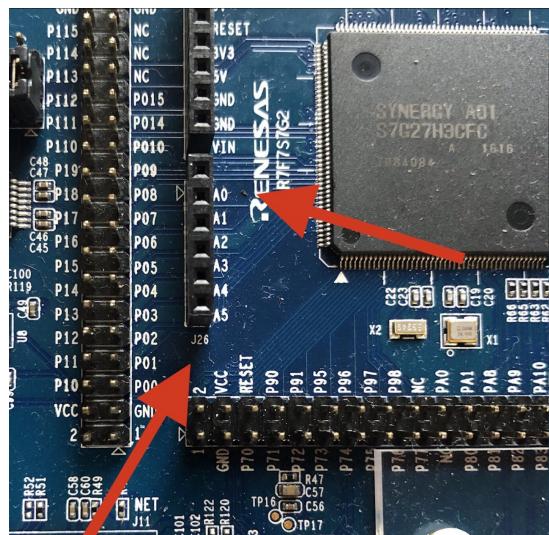
| S7G2 Pin | Peripheral | Signal | SK-S7G2 connector |
|----------|-------------------|--------|-------------------|
| P000 | Arduino Shield | AN000 | J26 (1) |
| P001 | Arduino Shield | AN001 | J26 (2) |
| P002 | Arduino Shield | AN002 | J26 (3) |
| P004 | LCD (Touchscreen) | IRQ9# | n/a |
| P005 | User Button | IRQ10# | S5 |
| P006 | User Button | IRQ11# | S4 |
| P007 | Arduino Shield | n/a | J24 (1) |
| ----- | ----- | ----- | ----- |

Podemos dizer que a porta J26 possui os 6 pinos A0 ao A6, os quais se configurados funcionam como ADC.

Na página 20 da documentação do Renesas Synergy Software Package mostra as conexões com os pinos:

<https://docs.rs-online.com/d3d3/0900766b81513ff0.pdf>

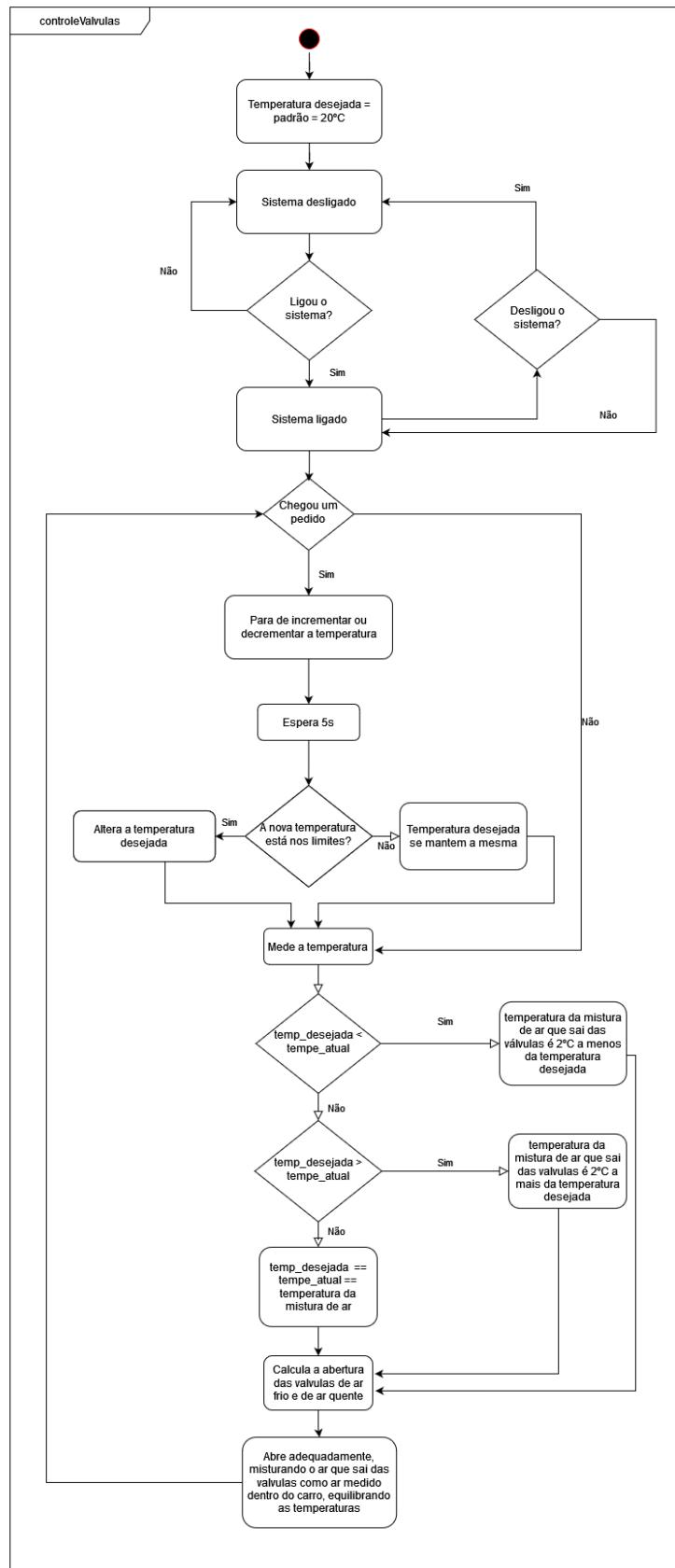
Projeto – Sistema de controle de temperatura veicular



Podemos ver na foto os 6 pinos correspondentes da porta J26 citada anteriormente, **vamos utilizar o potenciômetro no pino A0 desta porta.**

7 Design Detalhado

Diagrama de estados do controle de temperatura feito pelas válvulas:

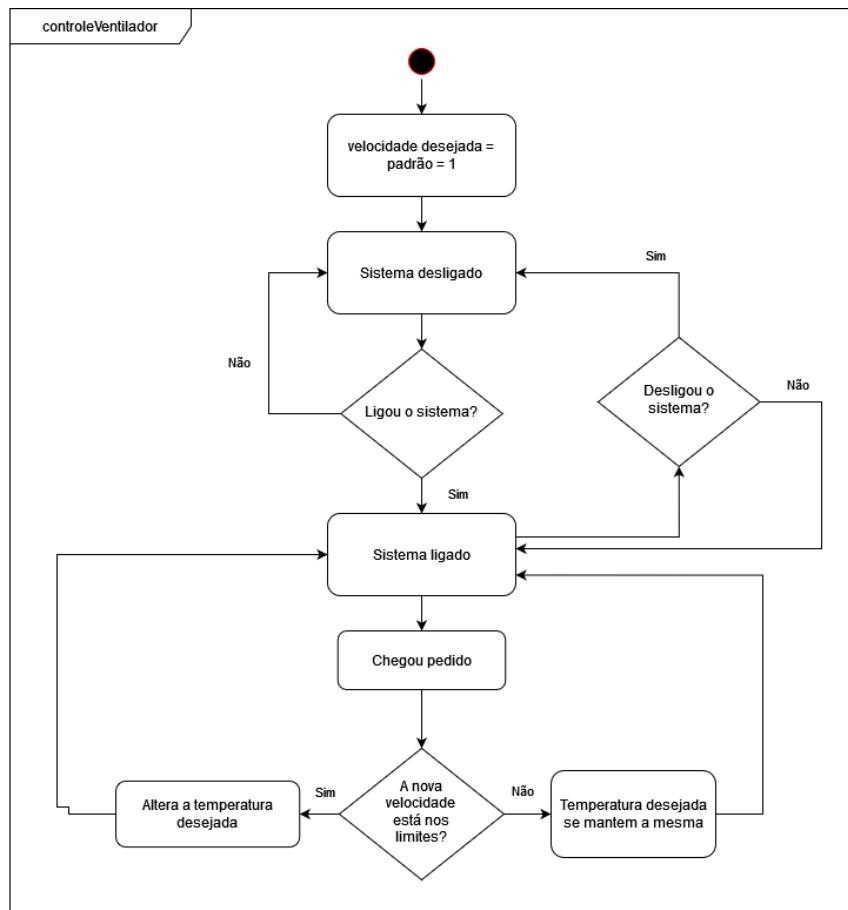


Projeto – Sistema de controle de temperatura veicular

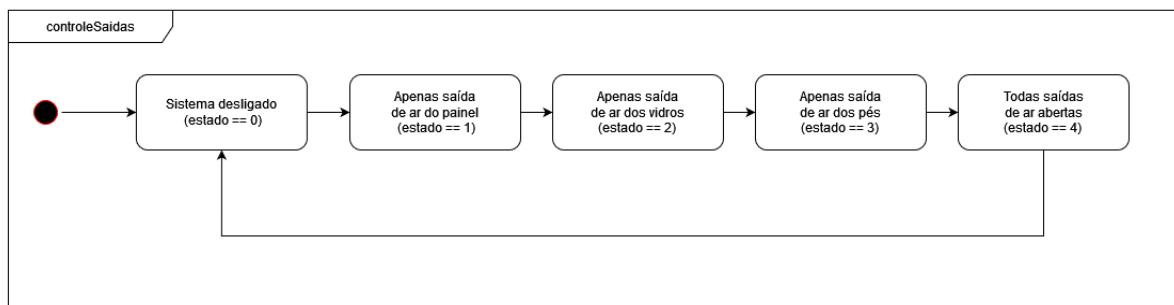
No caso também, como a thread de controle de válvulas retorna muitos valores, ao invés de criar seis filas de mensagem só pra uma thread, podemos ter ainda duas filas de mensagens, uma de pedidos (**entrada**) e uma fila de vetor de float (**saída**) que terá: temperatura atual, temperatura desejada, temperatura da mistura de ar, abertura da válvula de ar e abertura da válvula de ar quente. Assim, todas as threads terão apenas duas filas, mantendo a uniformidade.

No caso, quando a placa é ligada, ela não liga imediatamente as saídas de ar e todo o sistema. Ela espera até que o usuário aperte o botão “sel” a primeira vez para que possa abrir as válvulas, ligar o ventilador e abrir as saídas de ar.

Diagrama controle de velocidade do ventilador:

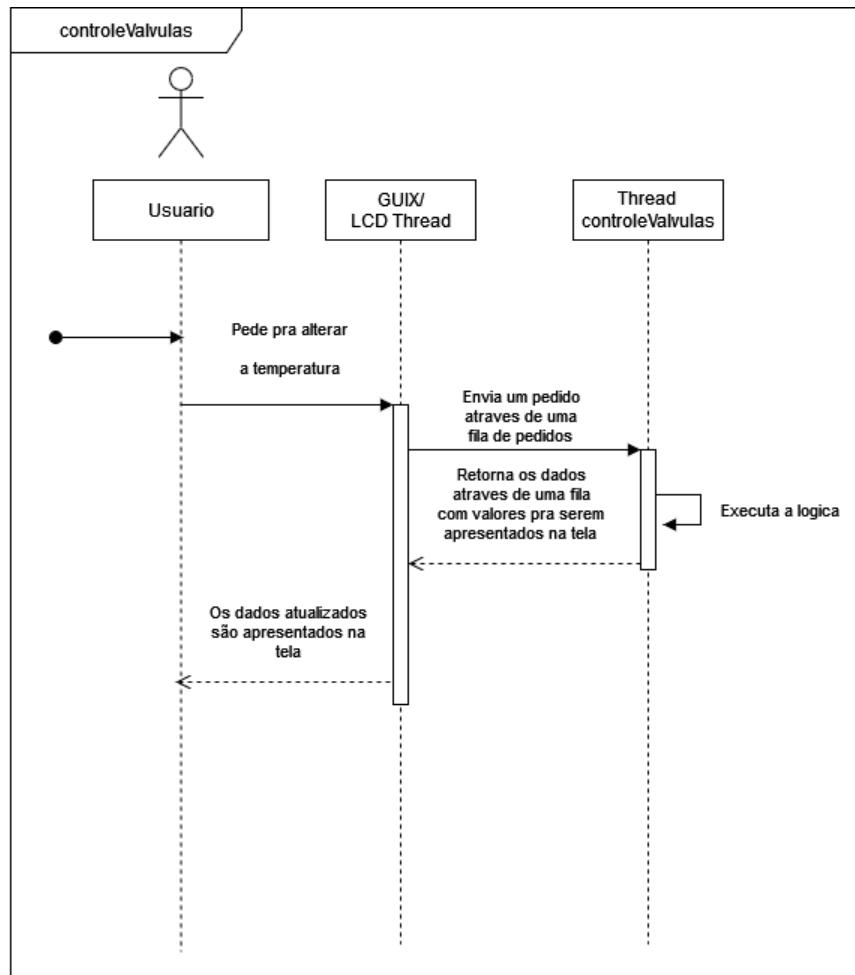


E o diagrama controle das saídas do carro, que também são os estados do sistema. Esse diagrama mostra quando o usuário de fato liga ou desliga o sistema:



Projeto – Sistema de controle de temperatura veicular

Alternativamente, podemos ver um exemplo do caso de um usuário tentando alterar a temperatura através de um diagrama de sequência:



Apesar de ser o exemplo hipotético da alteração da temperatura, ela se aplica também à velocidade do ventilador e também para troca entre as saídas de ar.

Parte 2b – Estudo da Plataforma

1 Manipulação da temperatura

Para o funcionamento ser da forma que foi solicitado, será necessário levar em consideração três temperaturas:

- **A temperatura desejada**, fornecida pelo usuário através da interface do sistema.
- **A temperatura atual**, fornecida pelo sensor de temperatura.
- **A temperatura da mistura de ar** das válvulas de ar quente e frio, que usaremos para calcular a abertura de cada válvula.

Assim se dará o funcionamento:

- Se a temperatura desejada for menor que a temperatura atual, desejamos resfriar o sistema. Portanto, a temperatura da mistura de ar deve estar 2°C a menos que a temperatura desejada. A temperatura atual irá diminuir com o tempo até atingir a temperatura desejada.
- Se a temperatura desejada for maior que a temperatura atual, desejamos aquecer o sistema. Portanto, a temperatura da mistura de ar deve estar 2°C acima que a temperatura desejada. A temperatura atual irá aumentar com o tempo até atingir a temperatura desejada.
- Quando o sistema estiver no equilíbrio, a temperatura atual será igual a desejada. E portanto, a temperatura da mistura também será igual a temperatura desejada.

Para o controle de temperatura, que calculo fazer e como determinar uma abertura adequada para cada válvula a fim de se ter uma mistura de ar com a temperatura desejada, existiam mais de uma solução para este problema.

Como por exemplo usando a Equação de Clapeyron, também conhecida como a lei geral dos gases ideais ou a Lei de Lei de Dalton da soma parcial das pressões parciais.

A solução selecionada foi a Equação fundamental da calorimetria por fornecer uma solução adequada e ser simples de calcular.

$$Q = m \cdot c \cdot \Delta T$$

$$Q_f + Q_q = 0$$

$$m_f \cdot c \cdot (T_m - T_f) + m_q \cdot c \cdot (T_m - T_q) = 0$$

$$m_f \cdot (T_m - T_f) + m_q \cdot (T_m - T_q) = 0$$

$$T_m = \frac{m_f T_f + m_q T_q}{m_f + m_q}$$

Sendo,

T_m : A **temperatura da mistura de ar**, vai depender se queremos resfriar ou aquecer, dependendo do caso ela será 2°C a menos ou a mais da temperatura desejada.

T_f : A temperatura da válvula de ar frio, que é de 15°C.

T_q : A temperatura da válvula de ar quente, que é de 40°C.

A fim de se saber a abertura necessária é fazemos uma média ponderada das temperaturas.

Como é uma equação que temos duas incógnitas (m_f e m_q), vamos arbitrar o valor de um e achar o outro, para assim gerar uma função facilmente. Vamos arbitrar que $m_q = 5$.

Assim achamos a massa de ar frio:

$$mf = \frac{-m_q \cdot (T_m - 40)}{(T_m - 15)} = \frac{-5 \cdot (T_m - 40)}{(T_m - 15)}$$

Que vamos usar pra achar a abertura da válvula de ar frio:

$$\text{Abertura}_{\text{válvula de ar frio}} = \left(\frac{m_f}{m_f + m_q} \right) \cdot 100$$

E a abertura da válvula de ar quente:

$$\text{Abertura}_{\text{válvula de ar quente}} = \left(\frac{m_q}{m_f + m_q} \right) \cdot 100$$

Lembrando que o controle se dará na faixa de 16°C a 30°C.

Lembrando também que para o caso especial da thread de controle de temperatura/controle das válvulas, nós usaremos um vetor de float.

2 GUIX

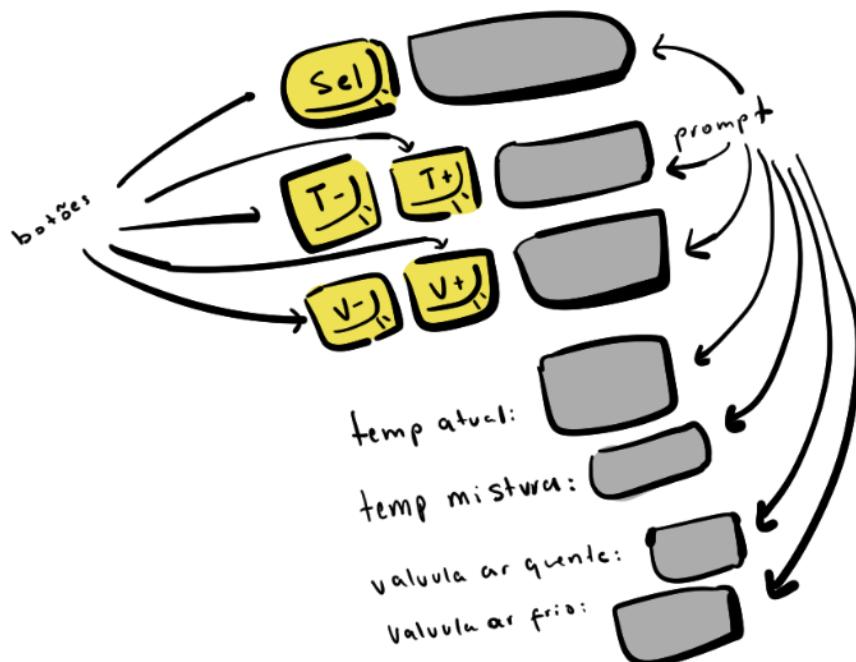
Grande parte do projeto se dará nessa parte, pois é aqui que é implementada a interface com o usuário. Utilizando a GUI do Azure GUIX, que será feita no programa GUIX Studio, fornecido pela Microsoft, criaremos a interface com o usuário e escolhemos os eventos dos botões para que possamos trabalhar com eles depois nos nossos códigos no ThreadX.

Lembrando que apresentaremos na tela, além dos botões, a temperatura atual, a temperatura desejada e também as aberturas das válvulas assim a fim de podermos ter uma noção do que está acontecendo no sistema.

Projeto – Sistema de controle de temperatura veicular

| Event | Description |
|----------------------------|----------------------------------------------|
| GX_EVENT_REDRAW | |
| GX_EVENT_SHOW | |
| GX_EVENT_HORIZONTAL_SCROLL | |
| GX_EVENT_VERTICAL_SCROLL | |
| GX_EVENT_TIMER | |
| GX_EVENT_PEN_DOWN | touch press or mouse button click |
| GX_EVENT_PEN_UP | touch released or mouse button released |
| GX_EVENT_PEN_MOVE | mouse pointer moved to a new location |
| GX_EVENT_PEN_DRAG | pen or mouse drag |
| GX_EVENT_KEY_DOWN | keyboard: key pressed |
| GX_EVENT_KEY_UP | keyboard: key released |
| GX_EVENT_TOGGLE_ON | checkbox buttons: changed to checked state |
| GX_EVENT_TOGGLE_OFF | checkbox buttons: changed to unchecked state |
| <u>GX_EVENT_CLICKED</u> | |

Também sobre os eventos, é possível fazer uma aplicação bem simples utilizando apenas o GX_EVENT_CLICKED.



No GUIX Studio é preciso saber o que será prompt e o que será botão, os botões em amarelo serão text buttons e as caixas em cinza serão os prompts que atualizaremos os valores ao decorrer da execução do programa.

Através dos **eventos do clique dos botões**, nós enviaremos solicitações para as threads, através de filas de mensagens. Através da thread do controle do LCD, **periodicamente** nós invocamos **um método** para atualizar a tela, neste método nós recebemos o retorno dos valores das threads, através de outras filas de mensagens. Essas informações nós colocaremos nos prompts.

3 ThreadX

A fim de podermos tirar vantagem do GUIX, além de criar threads, fazer as leituras das temperaturas, controlar os componentes e criar estruturas de controle adequadas será necessário utilizar algum tipo de RTOS, para isso utilizaremos o ThreadX, que tem suporte fornecido pela Renesas em sua plataforma de desenvolvimento, o e2studio.

Além disso, utilizamos várias threads, tentaremos desacoplar algumas funções de um loop principal, assim tornando elas em tarefas mais simples individualmente. Além disso, para estabelecer uma comunicação robusta faremos algumas filas pra enviar dados entre elas, como já foi mostrado nas seções anteriores

Vamos utilizar **Filas** (queues) porque são de longe a opção mais fácil. Memory Pool e Memory Block já são mais difíceis pelo simples fato de você ter que dar um endereço inicial durante a sua criação. Queues são criadas automaticamente pelo SSP e não precisamos nos preocupar com isso.

É possível enviar valores em float diretamente pela fila. A seguir um exemplo de como enviaremos a temperatura para a fila:

```
...
float temperatura_atual;
float saidas[5];

while (1)
{
    status = g_adc0.p_api->read(g_adc0.p_ctrl, ADC_REG_CHANNEL_0,
&adc_val);
    temperatura_atual = 10 + (40 * (float) adc_val/1024) ;
    saida[0] = temperatura_atual;

    tx_queue_send(&temp_atual_queue,  saida, TX_NO_WAIT);
...
}
```

Recebendo a temperatura atual da fila nas demais threads:

```
...
float estadoThreadValvulas[5];

while (1)
{
    tx_queue_receive(&temp_atual_queue, estadoThreadValvulas, 10);
...
}
```

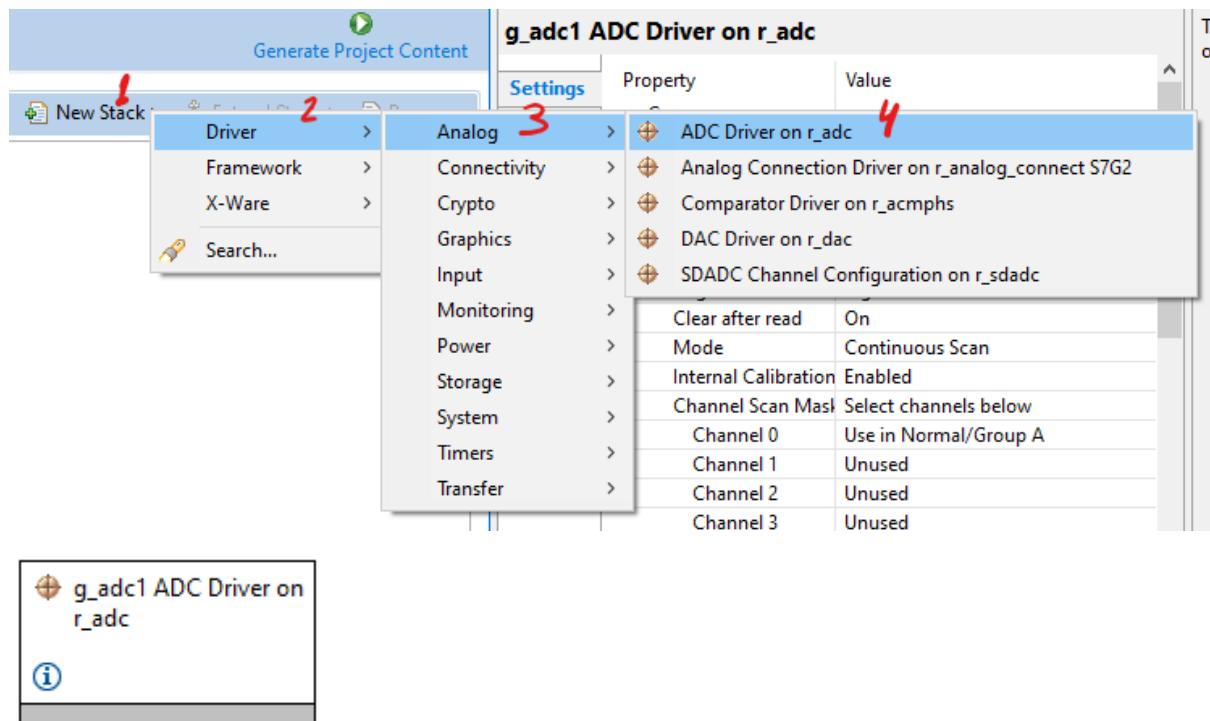
Assim idealmente cada variável crítica será transmitida através de uma fila.

4 Leitura do sensor

Para a leitura da temperatura, nós temos um conversor A/D como foi apresentado anteriormente. No caso da plataforma da Renesas, ela fornece ferramentas para utilizarmos rapidamente os conversores, no próprio software do e2studio.

Projeto – Sistema de controle de temperatura veicular

Para isso basta selecionarmos a thread em que queremos fazer a leitura e criarmos uma nova stack.



Com o stack criado nós podemos configurar de maneira adequada para o nosso projeto:

| g_adc1 ADC Driver on r_adc | | |
|----------------------------|----------------------|-----------------------|
| Settings | Property | Value |
| API Info | Common | |
| | Parameter Checkin | Enabled |
| | Module g_adc1 ADC | |
| | Name | g_adc1 |
| | Unit | 0 |
| | Resolution | 10-Bit |
| | Alignment | Right |
| | Clear after read | On |
| | Mode | Continuous Scan |
| | Internal Calibration | Enabled |
| | Channel Scan Mask | Select channels below |
| | Channel 0 | Use in Normal/Group A |
| | Channel 1 | Unused |

A nossa solução para o controle de temperatura não vai utilizar a temperatura atual, apenas iremos mostrar no display.

Projeto – Sistema de controle de temperatura veicular

A tabela abaixo mostra os métodos implementados para a leitura do conversor A/D, que usaremos para ler a temperatura, que no caso será substituído pelo potenciômetro a fim de mudarmos o estado do sistema.

ADC HAL Module API Summary

| Function Name | Example API Call and Description |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| open | <code>g_adc.p_api->open(g_adc.p_ctrl, g_adc.p_cfg);</code> Initialize ADC Unit; apply power, set the operational mode, trigger sources, interrupt priority, and configurations common to all channels and sensors. |
| scanCfg | <code>g_adc.p_api->scanCfg(g_adc.p_ctrl, g_adc.p_channel_cfg);</code> Configure the scan including the channels, groups and scan triggers to be used for the unit that was initialized in the open call. |
| scanStart | <code>g_adc.p_api->scanStart(g_adc.p_ctrl);</code> Start the scan (in case of a software trigger), or enable the hardware trigger. |
| scanStop | <code>g_adc.p_api->scanStop(g_adc.p_ctrl);</code> Stop the ADC scan (in case of a software trigger), or disable the hardware trigger. |
| scanStatusGet | <code>g_adc.p_api->scanStatusGet(g_adc.p_ctrl);</code> Check scan status. |
| read | <code>g_adc.p_api->read(g_adc.p_ctrl, ADC_REG_CHANNEL_13, &adc_data);</code> Read ADC conversion result(s). |
| sampleStateCountSet | <code>g_adc.p_api->sampleStateCountSet(g_adc.p_ctrl, &adc_sample);</code> Set the sample state count for the specified channel. |
| close | <code>g_adc.p_api->close(g_adc.p_ctrl);</code> Close the specified ADC unit by ending any scan in progress, disabling interrupts, and removing power to the specified A/D unit. |
| infoGet | <code>g_adc.p_api->infoGet(g_adc.p_ctrl, &adc_info);</code> Return the ADC data register address of the first (lowest number) channel and the total number of bytes to be read for the DTC/DMAC to read the conversion results of all configured channels. |
| versionGet | <code>g_adc.p_api->versionGet(&version);</code> Retrieve the API version with the version pointer. |

Na página 696 da documentação do Renesas Synergy Software Package mostra os exemplos de como chamar as funções do Conversor Analogico/Digital:

<https://synergygallery.renesas.com/media/products/1/412/en-US/r11um0161eu0120-synergy-sspv220.pdf>

5 Timer One Shot

Utilizaremos também um timer OneShot de 5s, que será resetado e acionado a cada vez que o usuário fizer um input e o callback será algo para aplicar as alterações (obviamente se a temperatura desejada se enquadrar nos limites impostos pelos stakeholders). Assim cumpriremos todas as exigências do projeto.

Projeto – Sistema de controle de temperatura veicular

Configuramos da seguinte maneira e o e2studio criará pra nós o código do timer, bastando apenas iniciá-lo.

| g_timer2 Timer Driver on r_gpt | |
|--------------------------------|-----------------------------------------|
| Settings | API Info |
| Property | Value |
| Common | |
| Parameter Checkin | Default (BSP) |
| Module g_timer2 Time | |
| Name | g_timer2 |
| Channel | 2 |
| Mode | One Shot |
| Duty Cycle Range (| Shortest: 2 PCLK, Longest: (Period -... |
| Period Value | 5 |
| Period Unit | Seconds |
| Duty Cycle Value | 50 |
| Duty Cycle Unit | Unit Raw Counts |
| Auto Start | True |
| GTIOCA Output En | False |
| GTIOCA Stop Level | Pin Level Low |
| GTIOCB Output En | False |
| GTIOCB Stop Level | Pin Level Low |
| Callback | timer2Callback |
| Overflow Interrupt | Priority 2 |

Abaixo alguns métodos que permitem nós configurarmos o timer. Basicamente, nós queremos dar um reset e um start após cada solicitação e no callback nós criamos alguma lógica que confirma as alterações.

Table 1 GPT HAL Module API Summary

| 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. |

Tabela retirada da página 3 do documento GPT HAL Module Guide:

<https://www.renesas.com/us/en/document/apn/gpt-hal-module-guide-application-project>