UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Nome: Leandro Ponsano Corimbaba **RA:** 239084
Arthur Cantarela **RA:** 135017

Disciplina: EA871 - Laboratório de Programação Básica de Sistemas Digitais
Docente: Denis F.

Relatório - Projeto Final

1. Descrição Geral

O projeto é um jogo chamado *Dino*, que consiste de um dinossauro em movimento constante em uma direção e que deve desviar de obstáculos (cactos) ao longo do caminho. O jogo termina quando o dinossauro colide com um cacto. Quanto maior a distância percorrida, maior a pontuação do usuário, que também é incrementada conforme se aumenta a velocidade do dinossauro.

2. Periféricos Utilizados

O projeto faz uso de 4 periféricos:

- Display de LCD, onde são renderizados o dinossauro e os obstáculos, além da pontuação do usuário e mensagens de início/fim de jogo;
- Botoeiras, que fazem o controle da movimentação do dinossauro;
- Potenciômetro, que controla o nível de dificuldade do jogo, ou seja, altera a velocidade do dinossauro;
- LEDs vermelhos conectados ao Latch, que indica o progresso e a distância percorrida no jogo.

3. Diagrama de Componentes

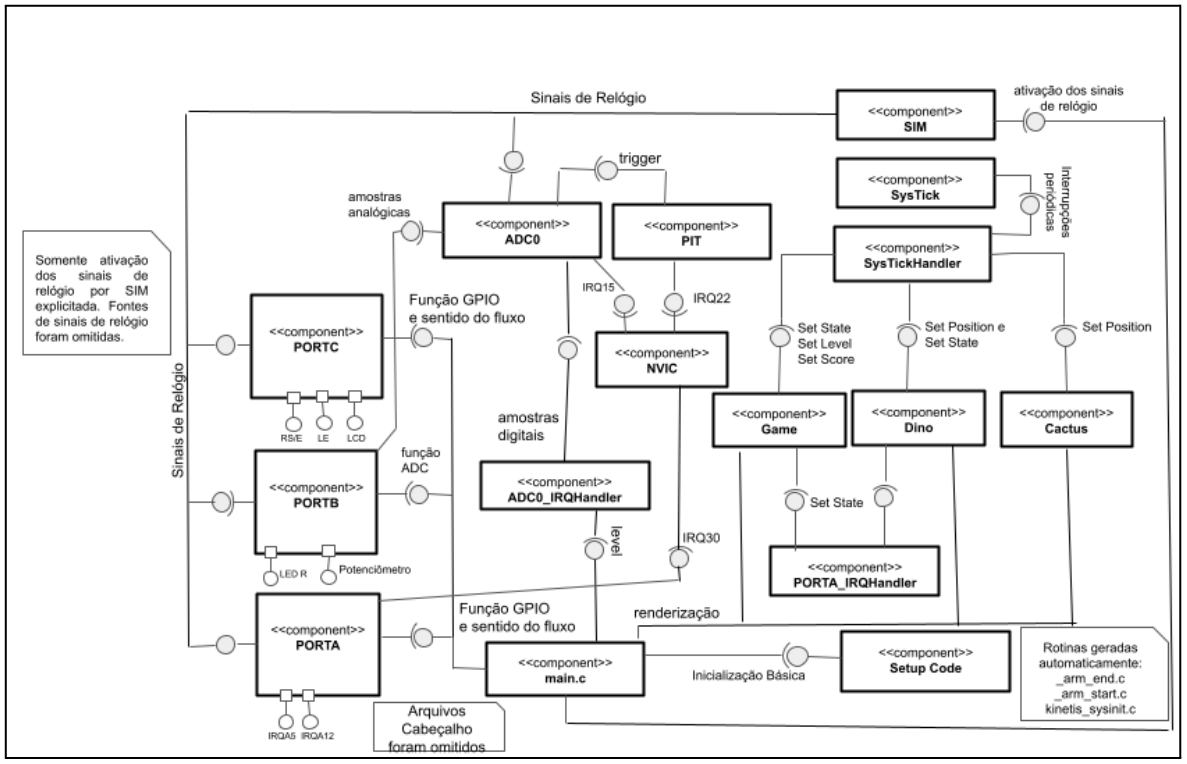


Figura 1: Diagrama de Componentes UML do projeto dino.

4. Máquina de Estados

Há quatro possíveis estados para o jogo:

- *RESETED*: “tela de início” do jogo, mostrando a mensagem “START” e aguardando um acionamento da botoeira IRQA5 pelo usuário;
- RUNNING: estado principal, quando o jogo está correndo. Acionamentos da botoeira IRQA12 executam um salto do dinossauro (passagem do estado *GROUND* para *JUMPING*);
- PAUSED: jogo pausado, desencadeado por um acionamento de IRQA5, com a qual também se retorna ao estado principal;
- GAMEOVER: quando o usuário é derrotado, aguarda um novo acionamento de IRQA5 para resetar o jogo.

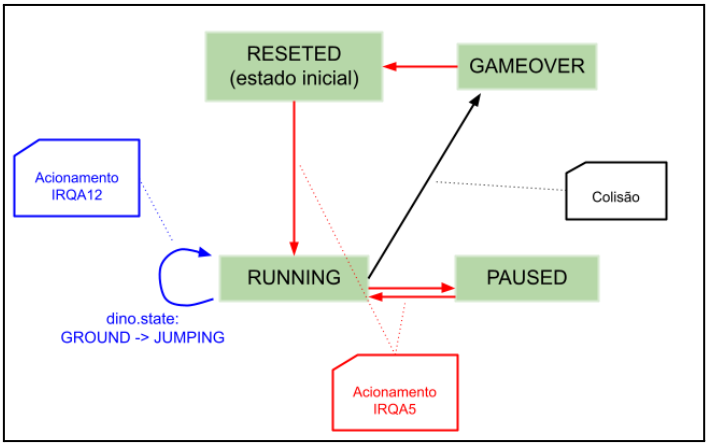


Figura 2: Diagrama da Máquina de Estados do Game.

5. Lógica e Fluxo de Execução

A função principal (*main.c*) consiste de duas partes:

I. inicialização e configuração dos módulos (SIM, PIT, ADC0);
II. laço infinito de execução.

O laço principal também pode ser dividido em dois:

I. renderização dos objetos no LCD;
II. atualização do nível de dificuldade conforme amostra obtida do ADC.

Os objetos (Dinossauro, Jogo e Cacto) são definidos nos arquivos *dino.h*, *game.h* e *cactus.h*, e as funções que manipulam suas variáveis internas, bem como sua renderização, se encontram nos respectivos arquivos *.c*.

No arquivo *ISR.c* ocorre o controle de estados do jogo. São 4 tratadores de interrupção presentes:

I. *SysTick_Handler*: chamado uma vez a cada X segundos, é responsável por atualizar a posição do cacto, verificar se houve colisão, solicitar movimentação do dinossauro e, por fim, incrementar o progresso e a pontuação do jogador (conforme nível de dificuldade);
II. *PORTA_IRQHandler*: chamada a cada aperto de botoeira, responsável pelas mudanças de estado explicitadas no item 4. Também habilita ou desabilita as interrupções do SysTick conforme estado;
III. *PIT_IRQHandler*: chamado a cada 180 milissegundos (aproximadamente), envia um trigger por software para o módulo ADC0;
IV. *ADC0_IRQHandler*: chamado a cada trigger recebido pelo PIT, armazena a amostra digital em sua variável de controle *valor*, usada pela *main* via função *ISR_LeValorAnostrado()*.

Por fim, os módulos *GPIO_switches.h, GPIO_latch_lcd.h, timers.h, TPM.h* e *util.h* foram desenvolvidos em experimentos anteriores e provém funções auxiliares e de configuração dos módulos mencionados.

6. Testes Conduzidos

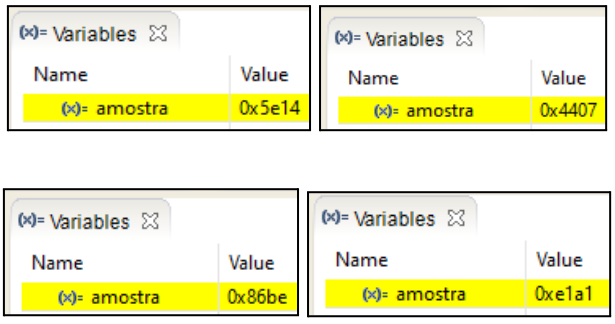
I. Período dos Temporizadores: utilizando as funções do módulo *GPIO_H5.c* (experimento 7) e conectando o analisador lógico ao *header* H5 do *shield* FEEC-871, foram medidos os seguintes períodos dos temporizadores utilizados:

A. PIT: 90 * 2 = 180 ms

B. SysTick: aproximadamente 125 ms. Salienta-se que sua forma de onda se apresentou peculiar, por razões que não puderam ser reconhecidas.

Figura 3: Captura de Tela do analisador Lógico no momento do teste.

II. Amostragem do ADC: com o jogo pausado e inserindo um *breakpoint* logo após a chamada da função *ISR_LeValorAmostrado* (em *main.c*), podemos monitorar a amostragem lendo o valor da variável *amostra*. Uma sequência de 4 valores lidos, com a chave do potenciômetro sendo girada a cada breakpoint, foi a seguinte:



Figuras 4 - 7: Capturas de Tela da aba Variables durante sessão de Debug no CodeWarrior.

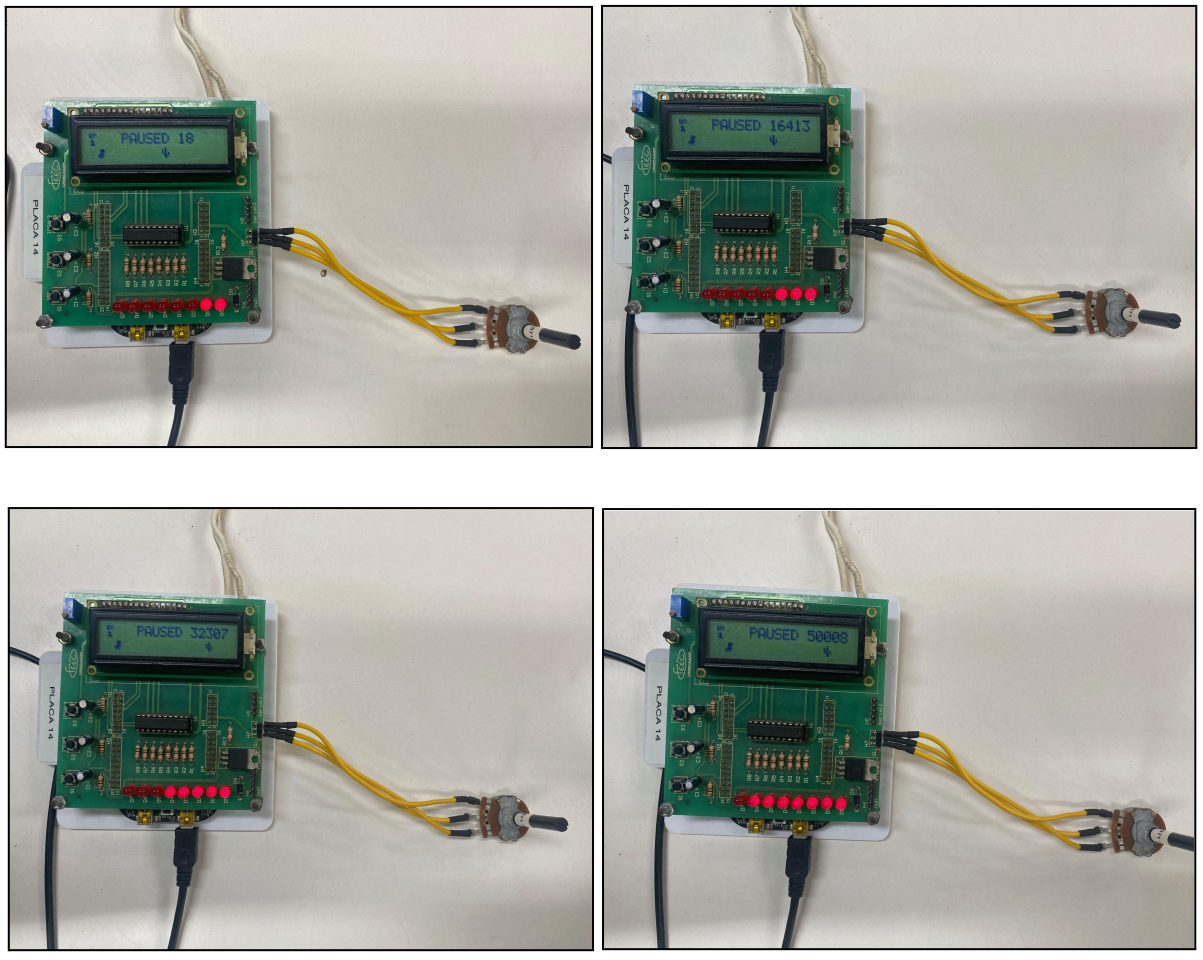
III. Byte de Progresso: o byte de progresso é transmitido ao Latch e mostrado nos LEDs vermelhos da placa. Ele incrementa à medida que a pontuação do jogador se aproxima de 65530 (quantidade máxima representável por um *uint16_t*, tipo da variável que armazena o *score*). Cada vez que o jogador atinge esse máximo, a pontuação é zerada e diz-se que o dinossauro completou uma volta (*turn*), que acende permanentemente o LED mais à direita.

Por exemplo, se o usuário completou 2 voltas e tem 33000 pontos (aproximadamente metade do valor máximo), o byte enviado para o Latch seria 0b00011111 (2 voltas = 2 bits menos significativos acesos; 50% de progresso = metade dos bits restantes acesos).

Vale ressaltar que a quantidade de bytes acesos pode diferir um pouco da porcentagem real, o que se deve à aproximações feitas pelo sistema e pela limitação de armazenar um progresso fracionário em uma variável do tipo *uint8_t.*

Podemos verificar o byte enviado ao Latch inicializando diferentes valores nas variáveis internas de *game.c*. Foram testadas 4 combinações:

A. *game.score* = 0 (0%) e *game.turns* = 1
B. *game.score* = 16400 (25%) e *game.turns* = 0
C. *game.score* = 32300 (50%) e *game.turns* = 2
D. *game.score* = 50000 (80%) e *game.turns* = 1



Figuras 8 - 11: Fotos da placa com o jogo em execução e pausado, para diferentes valores de score *e* turns.