**Escola Politécnica da Universidade de São Paulo**


**PCS3645 - Laboratório Digital II**

**Planejamento 3**

**Alunos**: Henrique Ramos de Godoy NUSP: 13683192
Marcelo Takayama Russo NUSP: 13680164
Victor Pedreira Santos Pepe NUSP: 13679565

**Professor**: Edson Toshimi Midorikawa

**Bancada** B1

São Paulo, 25 de Setembro de 2024

# Sumário

## Trena Digital Com Saída Serial

Para o experimento 4, foi solicitado o desenvolvimento de uma Trena Digital com Saída Serial. Diferente das últimas atividades, não haverá novos componentes (tirando um Multiplexador 4x1, para escolha de dados de saída do hexadecimal para a serial), de forma que o principal objetivo será reunir os componentes já construídos em um projeto maior, ou seja, a Trena Digital.

### 1. Projeto e Simulação do Circuito

Como já introduzido anteriormente, não haverá novos componentes, de forma que não será necessário ter uma parte introdutória descrevendo o funcionamento de cada entrada/saída. Nesse caso, o maior trabalho será reunir os componentes criados nos últimos experimentos no módulo da *trena_digital*.

Para a codificação em verilog, reutilizou-se os códigos disponibilizados nas outras aulas, criando apenas o módulo *trena_digital*, seu fluxo de dados e unidade de controle, definindo-se fios internos que permitissem a ligação entre os componentes e módulos pré-existentes. Não serão dados muitos detalhes da montagem, uma vez que foi necessário mais um trabalho operacional de conectar os módulos.

```
wire [11:0] s_medida;
wire s_trigger, s_medida_pronto, s_envio_pronto;
wire [1:0] seletor_mux;
wire [6:0] dados_ascii, s_unidade, s_dezena, s_centena;

// Circuito de interface com sensor
interface_hcsr04 INT (
    .clock    (clock    ),
    .reset    (reset    ),
    .medir    (medir    ),
    .echo     (echo     ),
    .trigger  (s_trigger),
    .medida   (s_medida ),
    .pronto   (s_medida_pronto   ),
    .db_estado()
);

tx_serial_701 SER(
    .clock(clock),
    .reset(reset),
    .partida(transmitir),
    .dados_ascii(dados_ascii),
    .saida_serial(saida_serial),
    .pronto(s_envio_pronto),
    .db_clock(),
    .db_tick(),
    .db_partida(),
    .db_saida_serial(),
    .db_estado()
);
```

*Figura 1: conexão dos componentes de sensor e serial no FD*

## 1.1 Análise do RTL Viewer

Como foram feitas muitas conexões, a parte de testes e análises se faz muito importante para garantir que não se deixou passar nenhum erro. Nesse sentido, o primeiro teste realizado foi com o RTL Viewer.

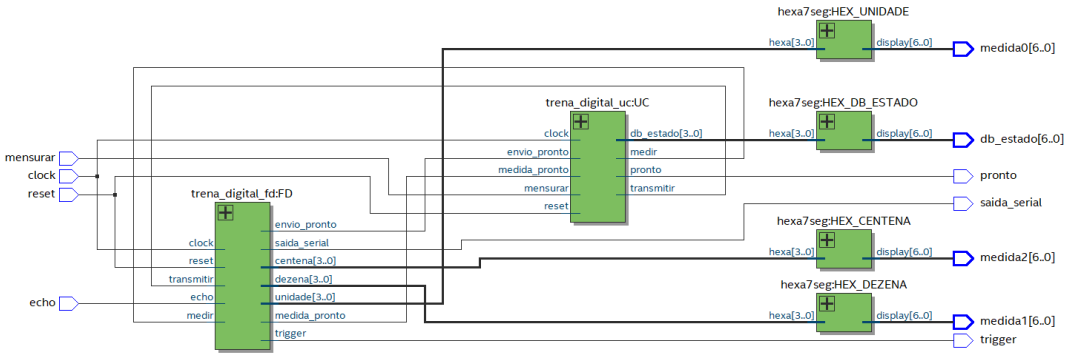


*Figura 2: RTL Viewer do Sistema Completo*

Em níveis de desempenho funcional, é difícil analisar por meio do RTL Viewer se há algum erro. Essa primeira testagem faz uma verificação se não foi deixado nenhum fio em aberto ou desconectado, além de permitir uma primeira noção se a lógica implementada está condizente com a esperada. Apesar disso, é possível

verificar que os componentes, entradas e saídas estão corretas, com *mensurar*, *clock*, *reset* e *echo* como inputs e o *pronto*, *estado* e as *medidas* como outputs.

No fluxo de dados, temos a seguinte organização:

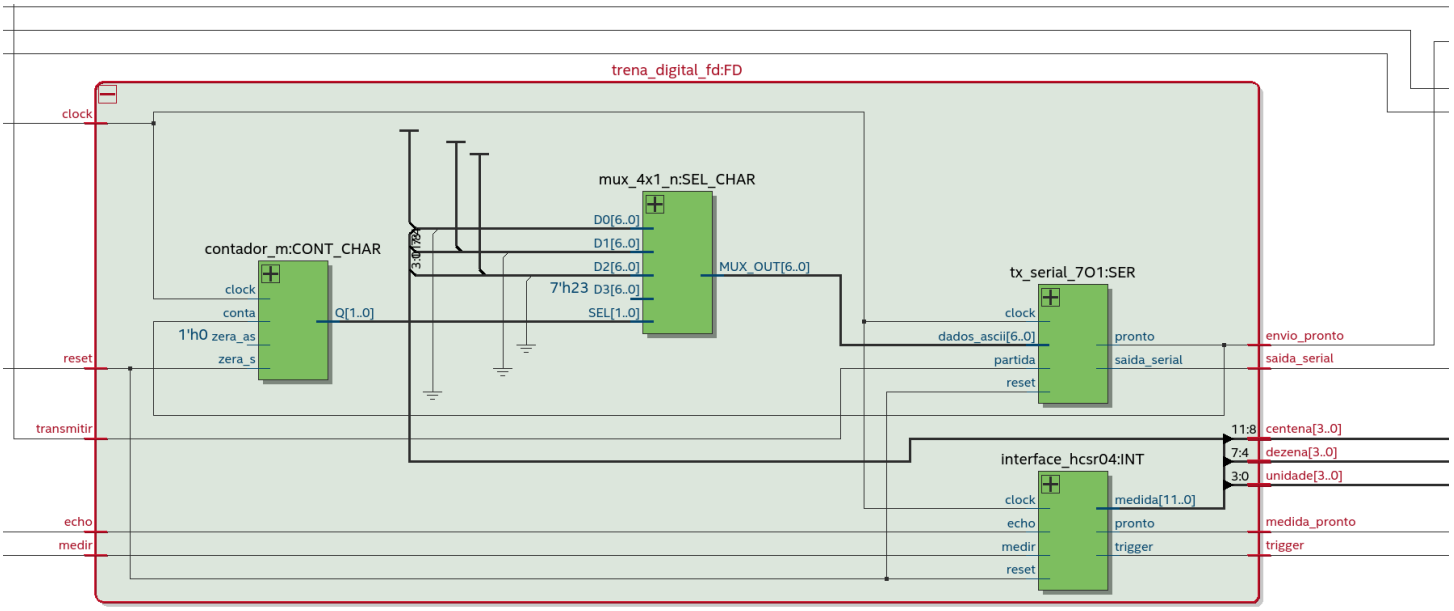


*Figura 3: Estados do Sistema pelo State Machine Viewer*

Aqui, há os 2 módulos desenvolvidos nas experiências anteriores: o transmissor serial e o interface com o sensor ultrassônico. Observe que as saídas da interface - os quais são a centena, a dezena e unidade da distância aferida (em centímetros), com 4 bits cada - são 3 das entradas do MUX. Além disso, para converter algarismos de 4 bits em caracteres ASCII, é preciso acrescentar 011 nos bits mais significativos. Por exemplo, o número 4 (0100), na tabela ASCII, é representado como 0110100. Logo, essas entradas são concatenadas com 011 na esquerda. Além disso, o caractere que indica o final da transmissão é o # (0100011 em ASCII), o qual está conectado na entrada D4 do MUX.

Esse MUX é utilizado para selecionar qual caractere ASCII será transmitido, e o seu seletor é a saída *Q* do *contador_m*. A ordem de transmissão é: centena, dezena, unidade e #. Inicialmente, como o contador está zerado (note que a entrada *reset* zera o contador), a centena é transmitida. Assim que essa transmissão serial é encerrada, a

saída *pronto* do *tx_serial_7o1* fica em nível lógico alto. Note que esse fio está conectado na entrada *conta* do *contador_m.*

Dessa forma, sempre que a transmissão de um caractere chega ao fim, o contador é incrementado e - consequentemente - a saída do MUX é alterada, visto que a entrada seletora muda. Dessa forma, todos os 4 caracteres são percorridos e transmitidos sequencialmente.

## 1.2 Análise de Estados com o State Machine Viewer

Para a unidade de controle, houve um trabalho maior, uma vez que foi necessário definir todo o diagrama de estados do projeto, tendo apenas como base os experimentos anteriores. Dessa forma, foram definidos 12 estados suficientes para garantir o total funcionamento do sistema.

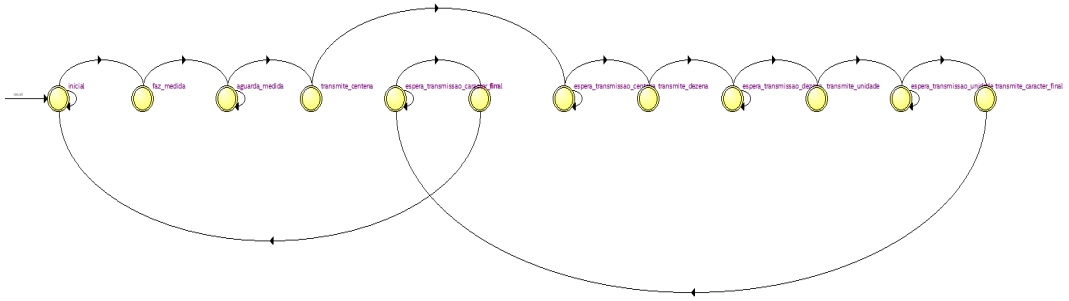


*Figura 3: Estados do Sistema pelo State Machine Viewer*

Como, pela imagem, é difícil entender os estados, vale passar por todos, descrevendo como é feita a passagem de estados e o que ocorre em cada um deles.

**inicial**: Entra-se nesse estado quando o *reset* é ativo ou quando se chega ao estado **fim**. Esse é o estado default do projeto, não tendo uma função a não ser esperar até que o possa ser iniciada a medida.

**faz_medida**: Entra-se nesse estado quando *mensurar* é ativo. Esse é o estado responsável por dar início à contagem, uma vez que ativa o sinal *medir*, que será

interpretado pela *interface_hcsr04,* resetando a contagem e enviando o trigger para começar a medição do sensor de distância.

**aguarda_medida**: Entra-se nesse estado logo após o **faz_medida**. Esse estado é intermediário e representa o tempo até que seja realizada a medição de distância pelo sensor. Esse estado é passado quando o sinal *medida_pronto* - recebido pela FD - é ativado, indicando que a distância já foi calculada e se pode iniciar a transmissão serial.

**transmite_centena**: como dito anteriormente, com o final da medição, inicia-se a transmissão serial. A partir de agora, a descrição do funcionamento será bastante similar, visto que há 2 estados para transmissão de cada hexadecimal. Como a transmissão é feita de maneira sequencial, inicia-se pela centena, de forma que este estado é responsável por enviar o sinal *transmitir* para a FD, indicando que pode ser enviado o hexadecimal da centena pela comunicação serial. A diferenciação de qual grandeza deve ser enviada é definida pelo multiplexador 4x1 implementado para esse projeto.

**espera_transmissao_centena**: estado intermediário bastante similar ao **aguarda_medida**. No entanto, nesse caso, espera-se até que a transmissão serial 7O1 seja realizada para que o próximo dado possa ser enviado. Esse estado é passado quando o sinal *envio_pronto* - enviado pela FD - é recebido pela UC, indicando que pode dar sequência à comunicação serial.

**transmite_dezena e transmite_unidade**: mesma funcionalidade do **transmite_centena**, mas agora focado na transmissão do dado da dezena e unidade.

**espera_transmissao_dezena e espera_transmissao_centena**: mesma funcionalidade do **espera_transmissao_unidade** , mas agora focado na transmissão do dado da dezena e unidade.

**transmite_caractere_final**: mesma funcionalidade do **transmite_centena**, mas - por definição da aula - deve ser transmitido o dado #. Esse dado representa o 23 na tabela ASCII que será enviado, delimitando o final da medição.

**espera_transmissao_caractere_final**: mesma funcionalidade do **espera_transmissao_unidade** , mas agora focado na transmissão do #, ou seja, o caractere final.

**fim**: estado final que é ativo quando se encerra a transmissão serial. Esse estado envia para a FD o sinal ativo *pronto*, indicando para os componentes o encerramento do sistema.

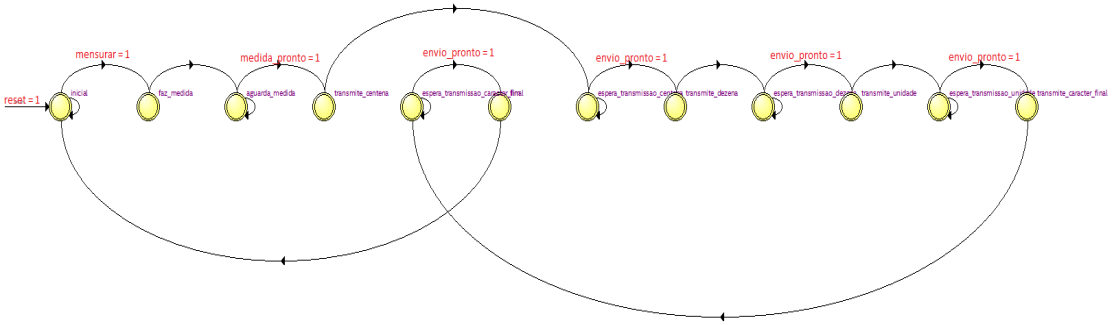


*Figura 4: State Machine Viewer com transições de estado inclusas*

## 1.3 Plano de Testes

Para a realização dos testes, serão utilizados como base os testbenchs anteriores. No caso, utilizou-se o testbench do Sensor de Distância, com os valores de distância a serem medidos. A ideia aqui será testar se esses valores estão sendo devidamente convertidos em Hexa e enviados pela Comunicação para serem interpretados e exibidos no experimento no laboratório. Dessa forma, elaborou-se a seguinte bateria de testes.

### Teste 1: 100 centímetros

Nesse primeiro teste, buscou-se testar o funcionamento básico do componente, verificando se a contagem estava sendo realizada de maneira correta, bem como se os tempos para essa contagem estavam de acordo e a conversão para da contagem para dígitos estava correta. Por isso, testou-se a contagem até 100 centímetros, passando pela conversão para os 3 hexadecimais. Para testar o funcionamento da transmissão, espera-se que sejam enviados pela transmissão 31 (hexadecimal para 1), seguido por 30 (hexa para 0) e 30 novamente, finalizando com 23 (hexa para #).

| CASO | Sinais Esperados | Sinais da Simulação |
|---|---|---|
| **1 (100cm)** | medida0 = 30<br>medida1 = 30<br>medida2 = 31<br>dados_ascii = 31 -> 30 -> 30 -> 23<br>largura_trigger = 10µs | medida0 = 30<br>medida1 = 30<br>medida2 = 31<br>dados_ascii = 31 -> 30 -> 30 -> 23<br>largura_trigger = 10µs |

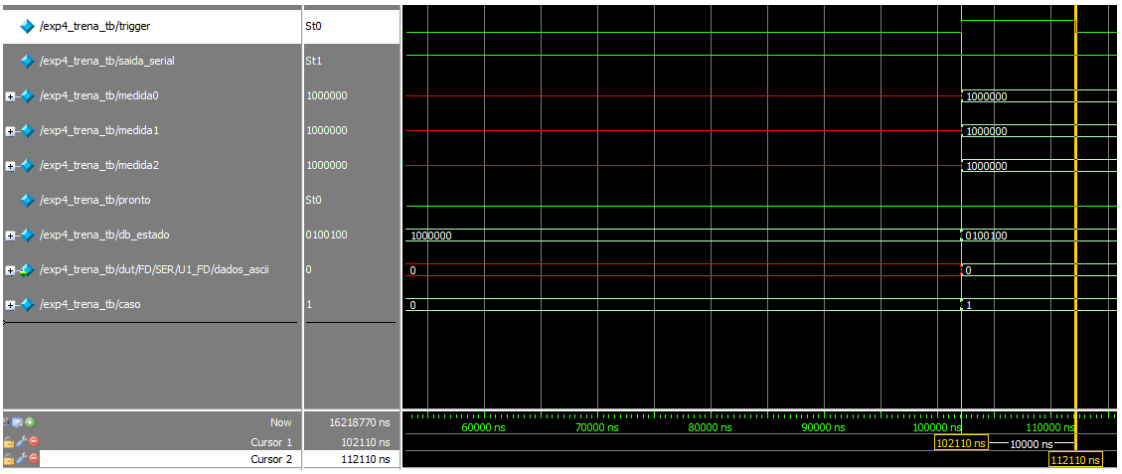


*Figura 5: verificação do tamanho do Trigger para ativação do Sensor*

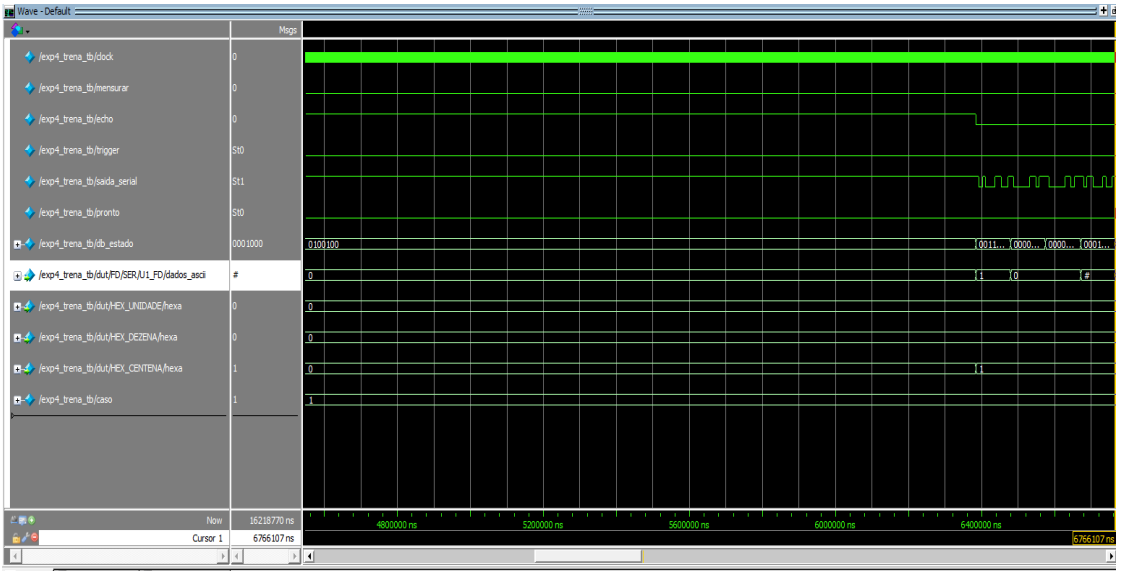


*Figura 6: verificação do Sinais de Saída para o Caso 1*

Como visto na imagem, pode-se perceber que os dados transmitidos pela comunicação serial se dão na ordem 1 -> 0 -> 0 -> #, ou seja Centena -> Dezena -> Unidade -> caractere_final. Outro ponto importante de se verificar foi o tamanho do pulso de trigger de 10µs, uma vez que o componente do sensor será ativado apenas se receber esse intervalo de pulso.

### Teste 2: 75,29 centímetros

Nesse teste, a ideia foi testar o arredondamento para baixo do contador, uma vez que temos 29 como valor decimal, sendo que 29 < 50, que é a condição de arredondamento. Como a ideia da contagem se baseia na meia contagem do contador, uma vez que qualquer valor decimal acima de 50 já é considerado, é conveniente esperar que não houve a contagem justamente porque o contador de centímetros é interrompido antes de chegar à meia contagem. Isso pode ser visto na imagem a seguir, onde a distância exibida fica em 75cm, como esperado. Além disso, foi importante verificar se esses sinais estão sendo transmitidos corretamente.

| CASO | Sinais Esperados | Sinais da Simulação |
|---|---|---|
| **2 (100,29cm)** | medida0 = 30<br>medida1 = 30<br>medida2 = 31<br>dados_ascii = 31 -> 30 -> 30 -> 23 | medida0 = 30<br>medida1 = 30<br>medida2 = 31<br>dados_ascii = 31 -> 30 -> 30 -> 23 |

### Teste 3: 54,79 centímetros

Agora, nesse teste, a ideia foi testar o arredondamento para cima do contador, uma vez que temos 79 como valor decimal, sendo que 79 > 50, que é a condição de arredondamento. Como a ideia da contagem se baseia na meia contagem do contador, uma vez que qualquer valor decimal acima de 50 já é considerado, é conveniente esperar que houve a contagem justamente porque o contador de centímetros é interrompido antes de chegar à meia contagem. Isso pode ser visto na imagem a seguir, onde a distância exibida fica em 55cm, como esperado.

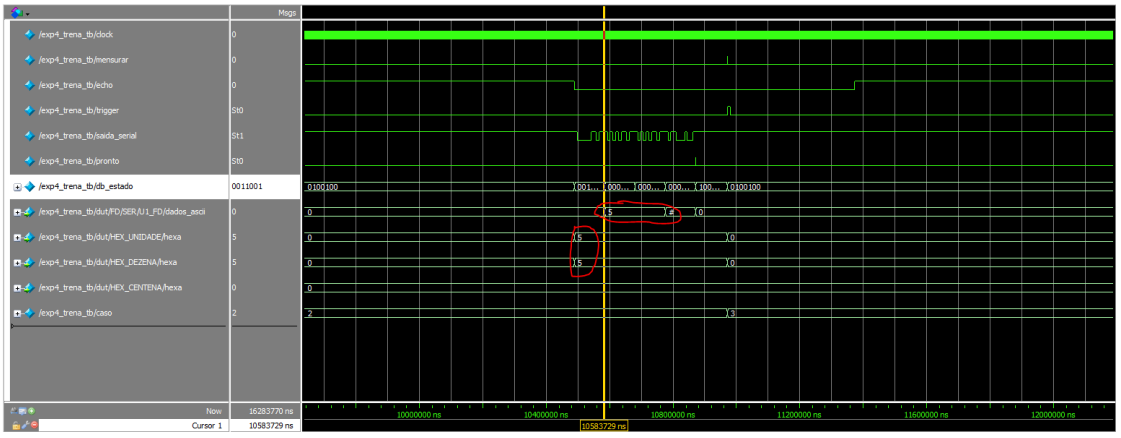


*Figura 7: verificação do Sinais de Saída para o Caso 3*

| CASO | Sinais Esperados | Sinais da Simulação |
|---|---|---|
| **3 (54,79cm)** | medida0 = 35<br>medida1 = 35<br>medida2 = 30<br>dados_ascii = 35 -> 35 -> 30 -> 23 | medida0 = 35<br>medida1 = 35<br>medida2 = 30<br>dados_ascii = 35 -> 35 -> 30 -> 23 |

## 1.4 Plano de Execução Experimental

Com os testes e simulações concluídos, a ideia é criar um plano de testes para ser realizado na bancada, otimizando o tempo de testes uma vez que já teremos isso pronto apenas para ser testado. O Plano Experimental a ser utilizado será realizado sob o formato do fill in the blanks, modelo bastante explorado na matéria.

| CASO | Valores Esperados | Valores Encontrados |
|---|---|---|
| **100 cm** | HEX0 = 0<br>HEX1 = 0<br>HEX2 = 1 | HEX0 =<br>HEX1 =<br>HEX2 = |
| **75,29 cm** | HEX0 = 5<br>HEX1 = 7<br>HEX2 = 0 | HEX0 =<br>HEX1 =<br>HEX2 = |
| **54,79 cm** | HEX0 = 5<br>HEX1 = 5<br>HEX2 = 0 | HEX0 =<br>HEX1 =<br>HEX2 = |

Para auxiliar no processo de designação dos pinos, utilizamos as seguintes imagens, que servirão também como Plano de Montagem Experimental.

**Table 3-6 Pin Assignment of Clock Inputs**

| Signal Name | FPGA Pin No. | Description |
|---|---|---|
| CLOCK_50 | PIN_M9 | 50 MHz clock input(Bank 3B) |
| CLOCK2_50 | PIN_H13 | 50 MHz clock input(Bank 7A) |
| CLOCK3_50 | PIN_E10 | 50 MHz clock input(Bank 8A) |
| CLOCK4_50 | PIN_V15 | 50 MHz clock input(Bank 4A) |

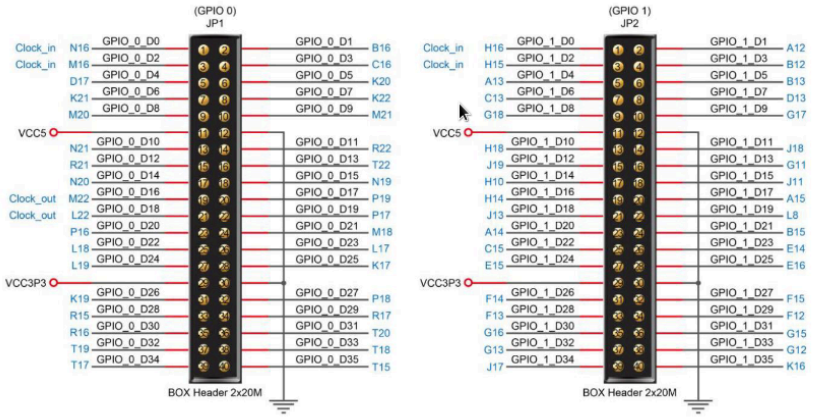


**Figure 3-12 I/O distribution of the expansion headers**

*Figura 8: Relação entre GPIO e pinos da FPGA*

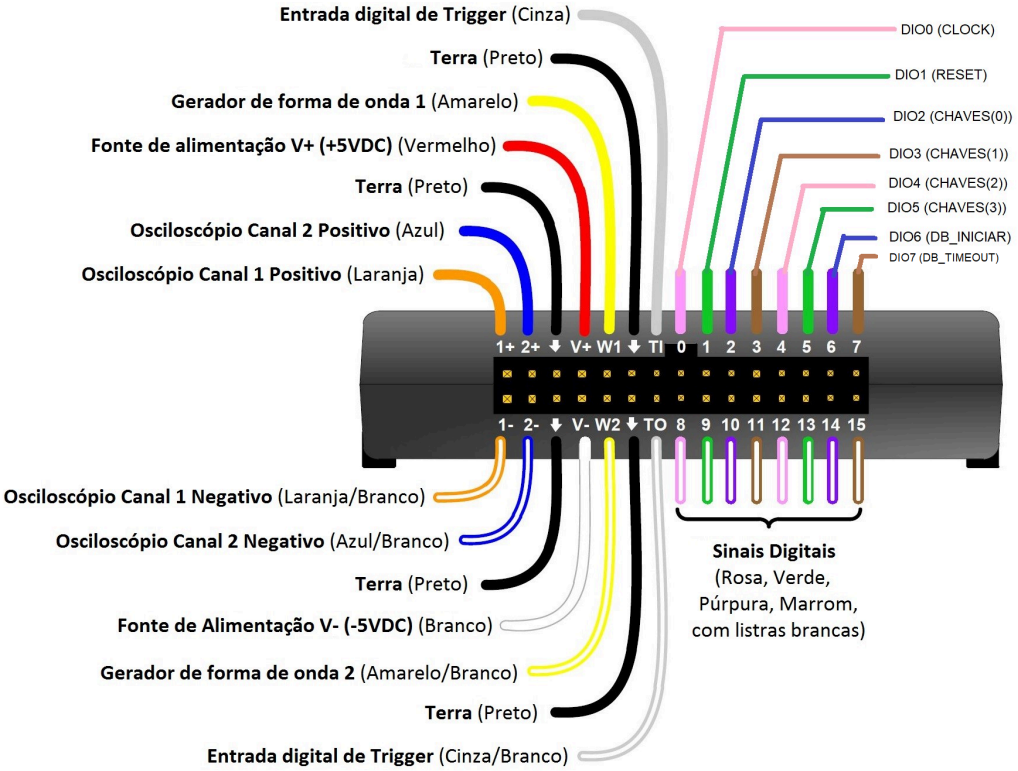


*Figura 9: entradas e saídas da GPIO*

Por fim, foi feita a seguinte pinagem:

| **Sinal** | **Pino da DE0-CV** | **Pino da FPGA** | **Analog Discovery** |
|---|---|---|---|
| clock | clock CLK_50 | M9 | - |
| reset | chave SW0 | U13 | - |
| mensurar | botão KEY0 | U7 | - |
| medida | displays HEX0 a HEX2 | U21 a AB21 | - |
| saida_serial | pino GPIO_0_D1 | B16 | - |
| trigger | pino GPIO_1_D1 | A12 | - |
| echo | pino GPIO_1_D3 | B12 | - |
| pronto | led LEDR0 | AA2 | - |
| db_mensurar | led LEDR1 | AA1 | - |
| db_saida_serial | pino GPIO_0_D35 | T15 | Protocol – DIO7 |
| db_trigger | pino GPIO_1_D33 | G12 | Scope – CH1+ |
| db_echo | pino GPIO_1_D35 | K16 | Scope – CH2+ |
| db_estado | display HEX5 | N9 a W19 | - |

*Figura 10: Tabela com pinagem da FPGA e Analog Discovery*

| Node Name | Direction | Location |
|---|---|---|
| clock | Input | PIN_M9 |
| db_echo | Output | PIN_K16 |
| db_estado[6] | Output | PIN_W19 |
| db_estado[5] | Output | PIN_C2 |
| db_estado[4] | Output | PIN_C1 |
| db_estado[3] | Output | PIN_P14 |
| db_estado[2] | Output | PIN_T14 |
| db_estado[1] | Output | PIN_M8 |
| db_estado[0] | Output | PIN_N9 |
| db_mensurar | Output | PIN_AA1 |
| db_saida_serial | Output | PIN_T15 |
| db_trigger | Output | PIN_G12 |
| echo | Input | PIN_B12 |
| medida0[6] | Output | PIN_AA22 |
| medida0[5] | Output | PIN_Y21 |
| medida0[4] | Output | PIN_Y22 |
| medida0[3] | Output | PIN_W21 |
| medida0[2] | Output | PIN_W22 |
| medida0[1] | Output | PIN_V21 |
| medida0[0] | Output | PIN_U21 |
| medida1[6] | Output | PIN_U22 |
| medida1[5] | Output | PIN_AA17 |
| medida1[4] | Output | PIN_AB18 |
| medida1[3] | Output | PIN_AA18 |
| medida1[2] | Output | PIN_AA19 |
| medida1[1] | Output | PIN_AB20 |
| medida1[0] | Output | PIN_AA20 |
| medida2[6] | Output | PIN_AB21 |
| medida2[5] | Output | PIN_AB22 |
| medida2[4] | Output | PIN_V14 |
| medida2[3] | Output | PIN_Y14 |
| medida2[2] | Output | PIN_AA10 |
| medida2[1] | Output | PIN_AB17 |
| medida2[0] | Output | PIN_Y19 |
| mensurar | Input | PIN_U7 |
| pronto | Output | PIN_AA2 |
| reset | Input | PIN_U13 |
| saida_serial | Output | PIN_B16 |
| trigger | Output | PIN_A12 |
| <<new node>> | | |

*Figura 11: Tabela com pinagem removida do Pin Planner*