Sistemas Digitais II	Projeto 1		Responsáveis: MT/GDB
PCS3225	2020S2		

PCS3225 – Sistemas Digitais II – Projeto 1 – 2020S2

Multiplicador Binário – Magnitude e Complemento de 2

Responsável: Marco Túlio Carvalho de Andrade

Revisão Técnica, Testbench, Simulações: Glauber de Bona

SUMÁRIO

I-INTRODUÇÃO

Foi disponibilizado para os alunos (material da Aula4 – 02/09/2.020) no e-disciplinas (*Moodle*) um arquivo (multiplicador.zip) contendo o projeto em VHDL de um Multiplicador Binário que calcula a magnitude da multiplicação dos operandos, Multiplicando e Multiplicador, ambos de 4bits, fornecendo o resultado em uma palavra de 8 bits sem sinal. Junto a este exemplo foi fornecido um *Testbench* (multiplicador_tb.vhd). Este *Testbench* permite simular o projeto com a ferramenta GHDL, gerar arquivo de simulação de extensão ".vcd" e finalmente verificar o resultado da simulação da entidade "multiplicador" por meio do gtkwave. Este multiplicador, como se sabe, utiliza o algoritmo de somas sucessivas para obter o resultado da multiplicação. O multiplicando é somado a si mesmo um número de vezes correspondente a magnitude do multiplicador.

Os arquivos VHDL (*.vhd) do multiplicador binário, que foram disponibilizados, são:

- multiplicador.vhd;
- multiplicador_tb.vhd;
- multiplicador_uc.vhd;
- multiplicador_fd.vhd;
- fa_1bit.vhd;
- fa_4bit.vhd;
- fa_8bit.vhd;
- mux4_2to1.vhd;
- reg4.vhd;
- reg8.vhd;
- zero_detector.vhd.

A seguir são apresentadas as entidades do multiplicador (extraída do arquivo VHDL multiplicador.vhd), da unidade de controle (UC) do multiplicador (extraída do arquivo VHDL multiplicador_uc.vhd) e do fluxo de dados (FD) do multiplicador (extraído do arquivo VHDL multiplicador_fd.vhd):

```
entity multiplicador is
  port (
    Clock:      in  bit;
    Reset:      in  bit;
    Start:      in  bit;
    Va,Vb:      in  bit_vector(3 downto 0);
    Vresult:    out bit_vector(7 downto 0);
    Ready:      out bit
  );
end entity;
```

Pode-se verificar que existem três entradas do tipo "*bit*" (*clock*, *reset* e *start*) e duas entradas do tipo "*bit_vector(3 downto 0)*" (*Va* e *Vb*).

```
entity multiplicador_uc is
  port (
    clock:          in bit;
    reset:          in bit;
    start:          in  bit;
```

```
   Zrb:            in  bit;
   RSTa,CEa:         out bit;
   RSTb,CEb:         out bit;
   RSTr,CEr:        out bit;
   DCb:            out bit;
   ready:           out bit
  );
end entity;

entity multiplicador_fd is
  port (
   clock:       in  bit;
   Va,Vb:       in  bit_vector(3 downto 0);
   RSTa,CEa:    in  bit;
   RSTb,CEb:    in  bit;
   RSTr,CEr:    in  bit;
   DCb:         in  bit;
   Zrb:         out bit;
   Vresult:     out bit_vector(7 downto 0)
  );
end entity;
```

A análise destas entidades e dos arquivos *.vhd fornecidos permite extrair um diagrama de blocos dos sinais que conectam a UC ao FD, internamente no multiplicador (Figura I.1).

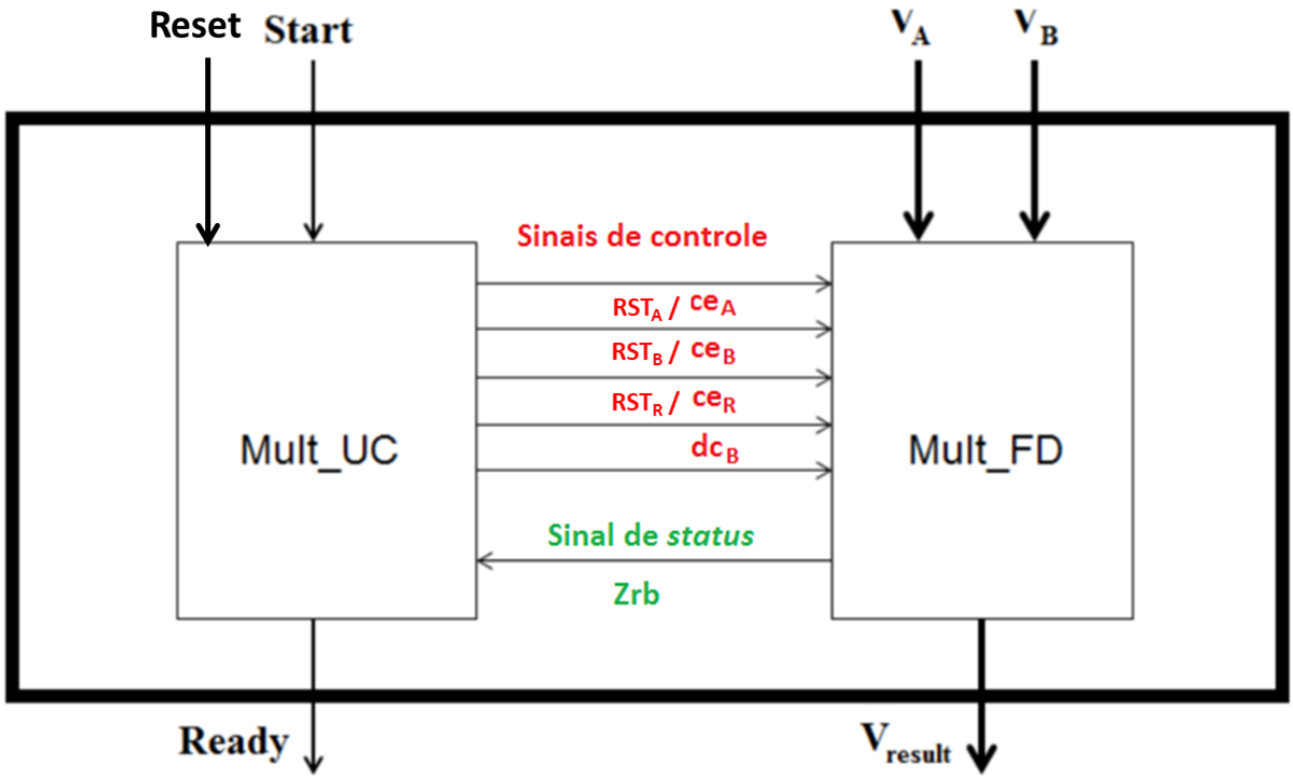

Figura I.1 – Diagrama de blocos das conexões entre a UC e o FD.

Uma análise um pouco mais detalhada, destas entidades e dos arquivos *.vhd de todos os componentes envolvidos permite também extrair um diagrama de blocos do fluxo de dados (FD), com seus sinais de entrada, saída, e os sinais que conectam seus componentes internos (Figura I.2).

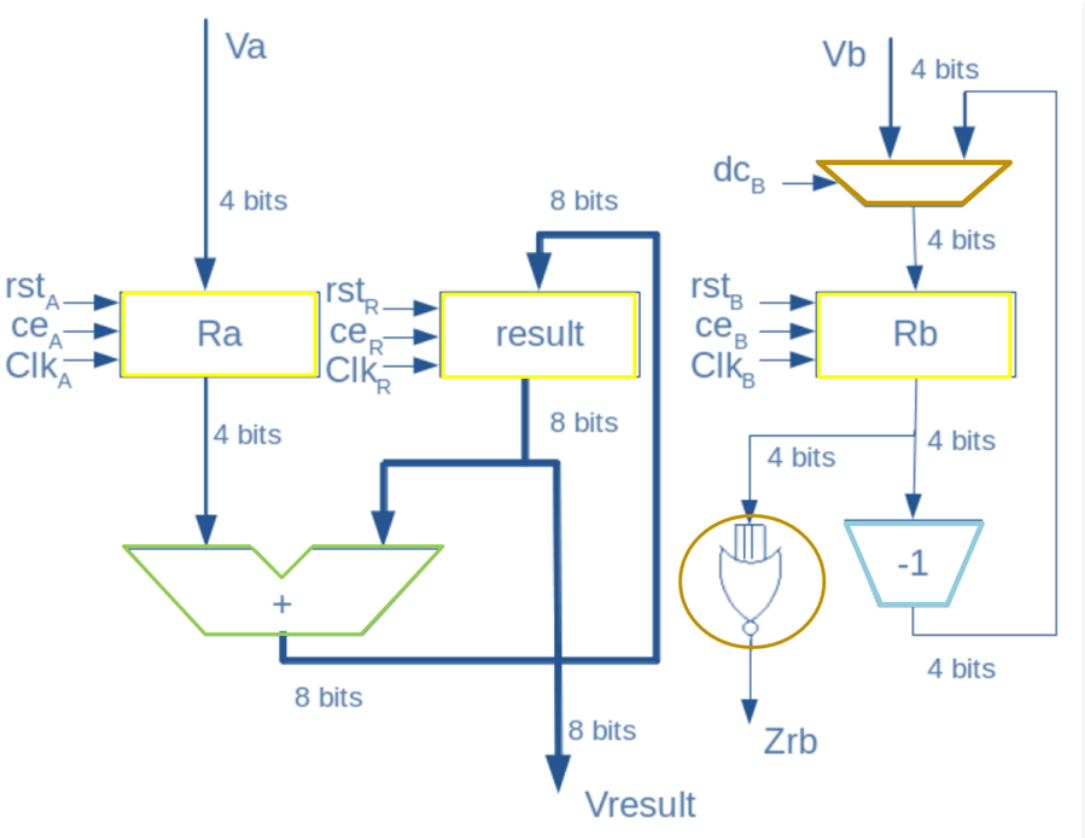

Figura I.2 – Diagrama de blocos das entradas, saídas e conexões internas do FD do multiplicador.

É importante fazer um esforço para entender o funcionamento do fluxo de dados no tocante a execução do algoritmo de somas sucessivas para o cálculo do resultado da multiplicação. Isto será útil para resolver o problema que se vai propor.

II-PROBLEMA A SER RESOLVIDO

O problema que se quer resolver é:

- Dado um multiplicador binário (tal qual o descrito na Introdução, Parte I) que calcula a magnitude da multiplicação de dois operandos de 4 bits (multiplicando e multiplicador), onde estes 4bits representam a magnitude destes operandos (sem sinal), introduzir um sinal externo de controle ("***signed_mult***") de modo que quando este estiver em nível um ("**1**") realize a multiplicação com sinal, considerando os operandos na notação complemento de 2. Quando este sinal estiver em nível zero ("**0**") ele deve operar como um multiplicador de magnitude.

Premissas que **devem** ser adotadas e/ou consideradas:

- Ao inicializar o multiplicador por meio do sinal ***Reset*** (com duração de 1 período de ***clock***) este último será sucedido (após 1 período de ***clock***) pelo sinal de ***Start*** (com duração de 1 período de ***clock***), o que, por sua vez, vai disparar o cálculo da multiplicação (Figura II.1). Em outras palavras, o ***Reset*** é assíncrono (preponderante sobre o ***clock***) e leva a um estado inicial onde, com ***Start*** = 1 dá-se início à operação.
- Um novo pulso de ***Start*** inicia uma nova operação, com os valores de entrada (***Va*** e ***Vb***) que houver. O pulso de ***Start*** não provoca mudanças nas entradas, mas será precedido por elas no ***testbench*** que será usado para avaliação das notas no ***Moodle***. Neste último vamos atualizar os valores das entradas ***Va***, ***Vb***, ***Signed_mul*** e daí ativaremos ***Start*** (Figura II.2).

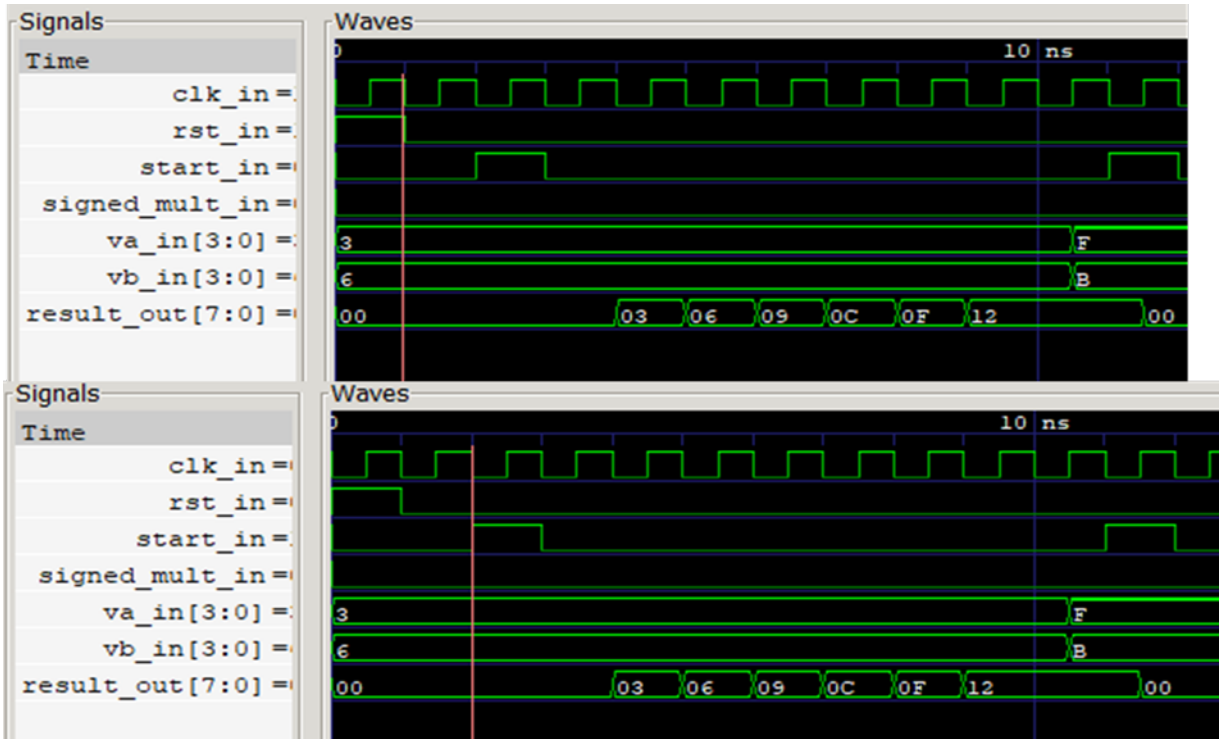

Figura II.1 – Diagrama de temporização – Sinais ***Reset*** e ***Start***.

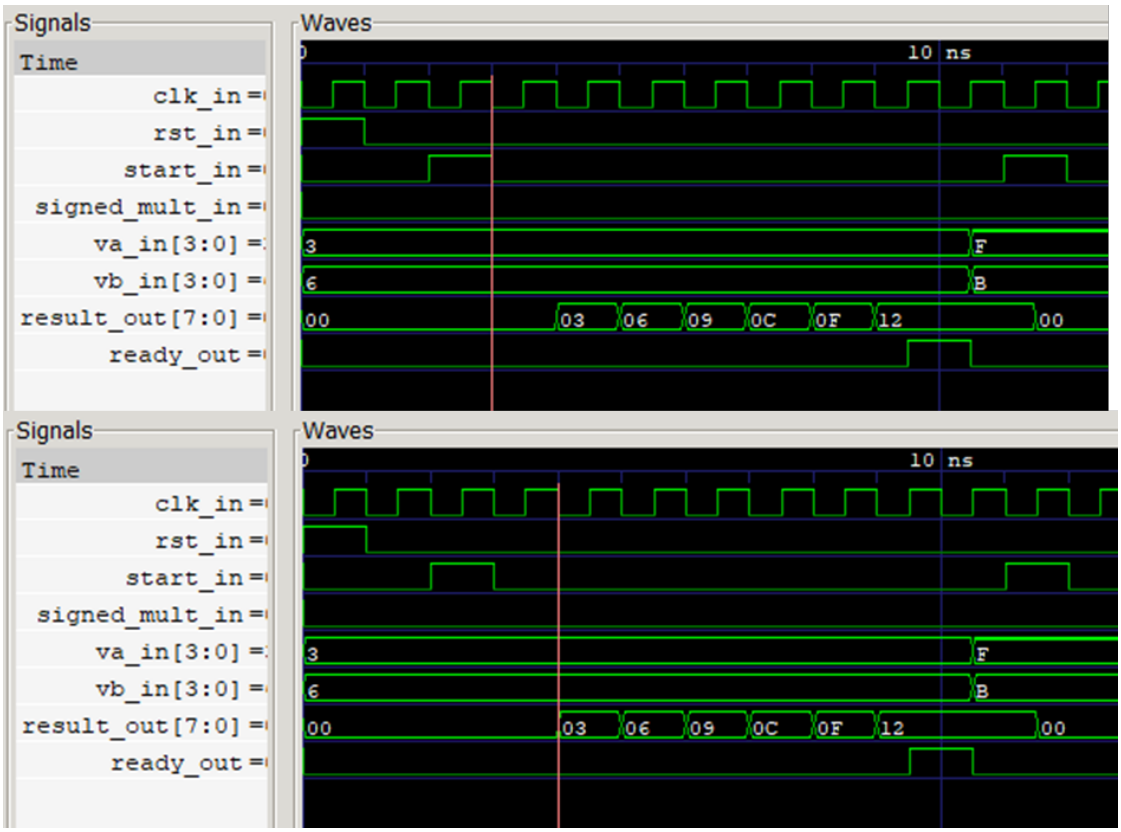

Figura II.2 – Sinais ***Signed_mult*** e ***Start***, operandos ***Va*** e ***Vb***.

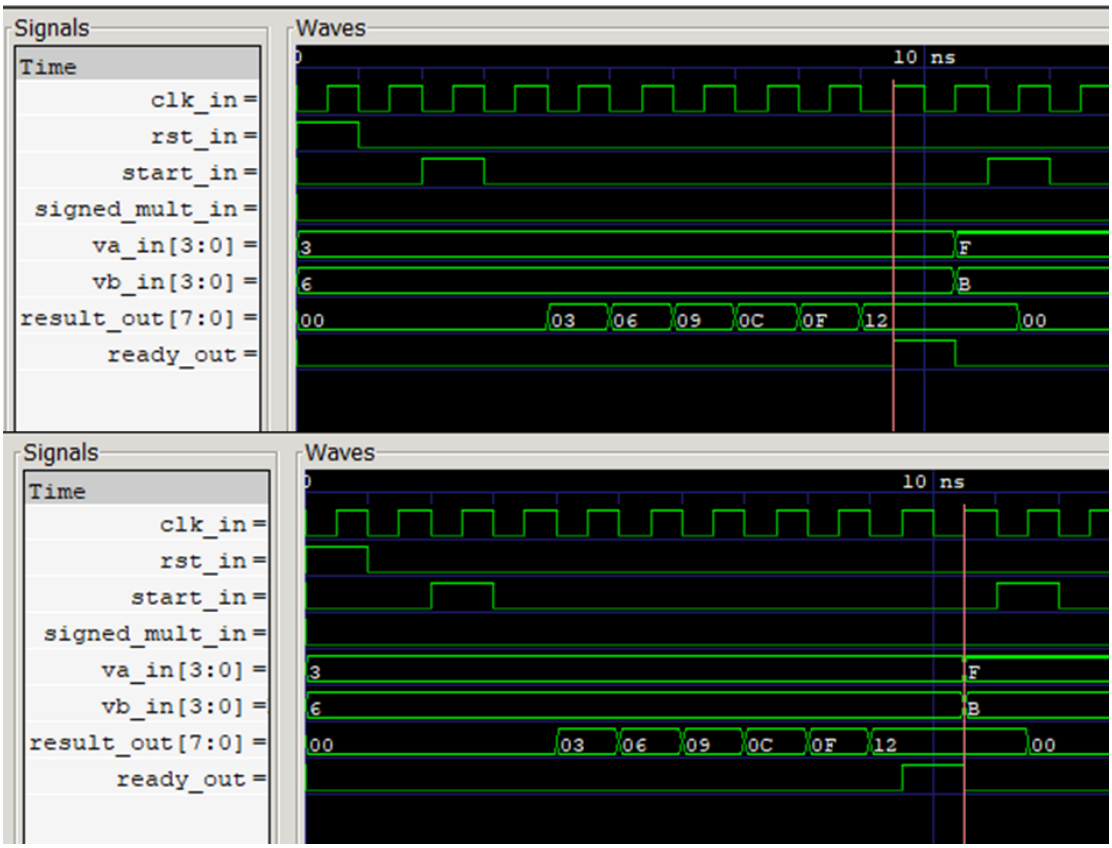

Figura II.3 – Sinais ***Signed_mult*** e ***Start***, operandos ***Va*** e ***Vb*** e sinal ***Ready***.

- Cada operação de multiplicação será concluída quando a UC gerar o sinal ***Ready***. Enquanto o sinal ***Ready*** não é gerado os valores lógicos de ***Signed_mult***, ***Va*** e ***Vb*** serão mantidos constantes (Figura II.3). O sinal de ***Reset*** será ativado apenas uma vez. Um novo sinal de ***Start*** provocará uma atualização dos valores lógicos dos sinais ***Signed_mult***, ***Va*** e ***Vb***, que por sua vez serão mantidos estáveis até que novo ciclo se repita (Figura II.4).
- A unidade do controle do novo multiplicador deve ser a mesma que a unidade de controle do multiplicador da Parte I, com a mesma entidade (mesmos sinais de entrada e saída, mesmos sinais de controle e de condição).

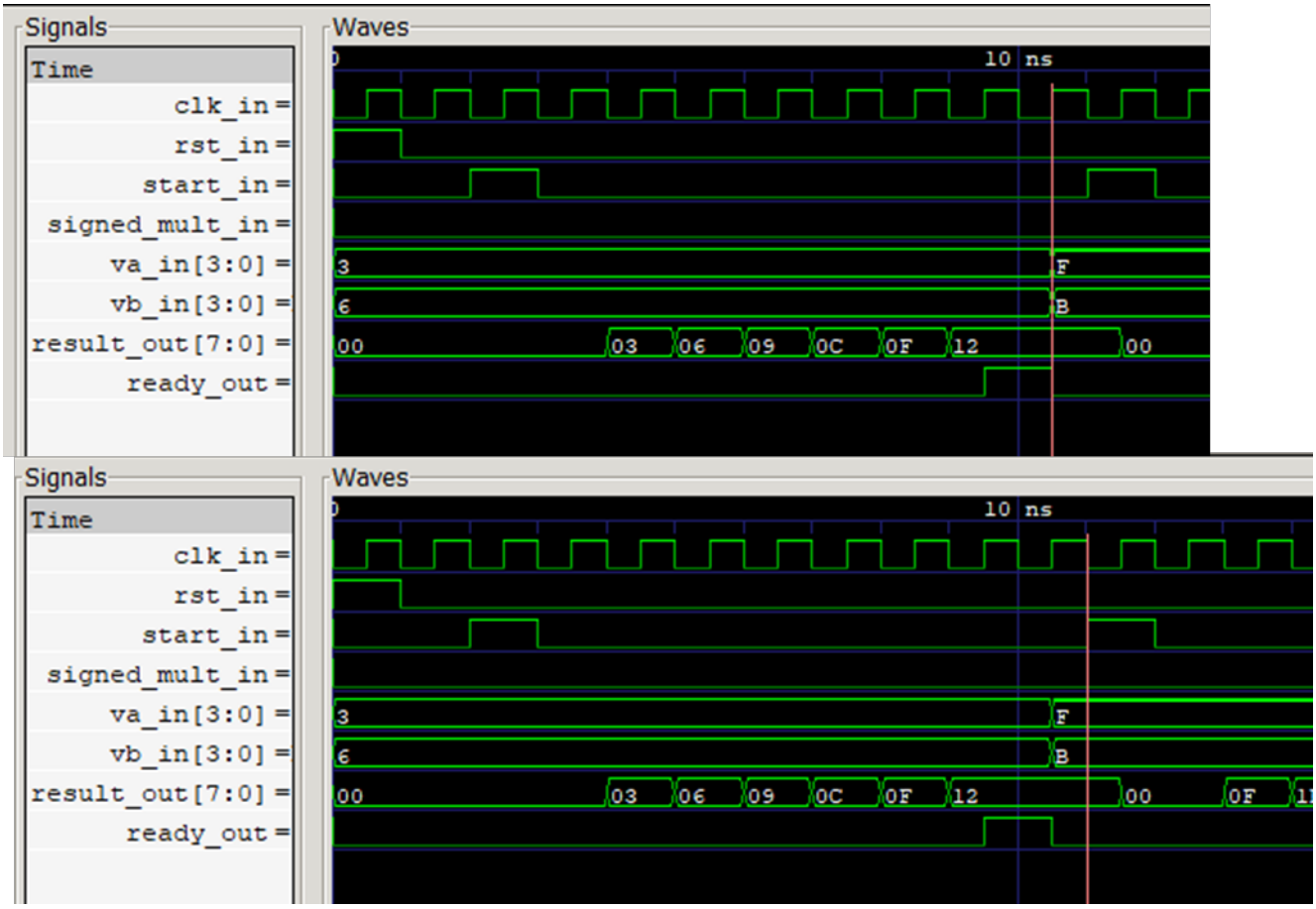

Figura II.4 – Atualização dos valores dos operandos ***Va*** e ***Vb*** e novo sinal ***Start***.

Como deve ser adotado que a unidade de controle (UC) será a mesma que a do multiplicador da Parte I deste enunciado, isto define que o único bloco que deve ser alterado, redesenhado, reprojetado, é o fluxo de dados (FD).

A seguir mostra-se como devem ser definidas as entidades do **multiplicador** a ser projetado:

- **Multiplicador** – multiplicador binário em magnitude e complemento de 2 (deve estar desta forma no arquivo VHDL multiplicador.vhd);

```
entity multiplicador is
  port (
    Clock:       in  bit;
--! Entrada adicional para multiplicação com sinal
-------------------------------------------------------
    signed_mult:  in  bit;
-------------------------------------------------------
    Reset:       in  bit;
    Start:       in  bit;
    Va,Vb:       in  bit_vector(3 downto 0);
    Vresult:     out bit_vector(7 downto 0);
    Ready:       out bit
  );
end entity;
```

- **UC** – unidade de controle do multiplicador (deve estar desta forma no VHDL multiplicador_uc.vhd). Pode-se notar que a entidade é a mesma que a utilizada no multiplicador da Parte I, sem alterações em seus sinais de entrada e de saída;

```
entity multiplicador_uc is
  port (
    clock:           in bit;
    reset:           in bit;
    start:           in  bit;
    Zrb:             in  bit;
    RSTa,CEa:        out bit;
    RSTb,CEb:        out bit;
    RSTr,CEr:        out bit;
    DCb:             out bit;
    ready:           out bit
  );
end entity;
```

- **FD** – fluxo de dados do multiplicador (deve estar desta forma no arquivo VHDL multiplicador_fd.vhd):

```
entity multiplicador_fd is
  port (
    clock:       in  bit;
--! Entrada adicional para multiplicação com sinal
-------------------------------------------------------
    sig_mult_fd:  in  bit;
-------------------------------------------------------
    Va,Vb:       in  bit_vector(3 downto 0);
```

```
   RSTa,CEa:    in  bit;
   RSTb,CEb:    in  bit;
   RSTr,CEr:    in  bit;
   DCb:         in  bit;
   Zrb:         out bit;
   Vresult:     out bit_vector(7 downto 0)
  );
end entity;
```

A análise destas novas entidades permite extrair um diagrama de blocos dos sinais que conectam a UC ao FD, internamente no multiplicador (Figura II.5).

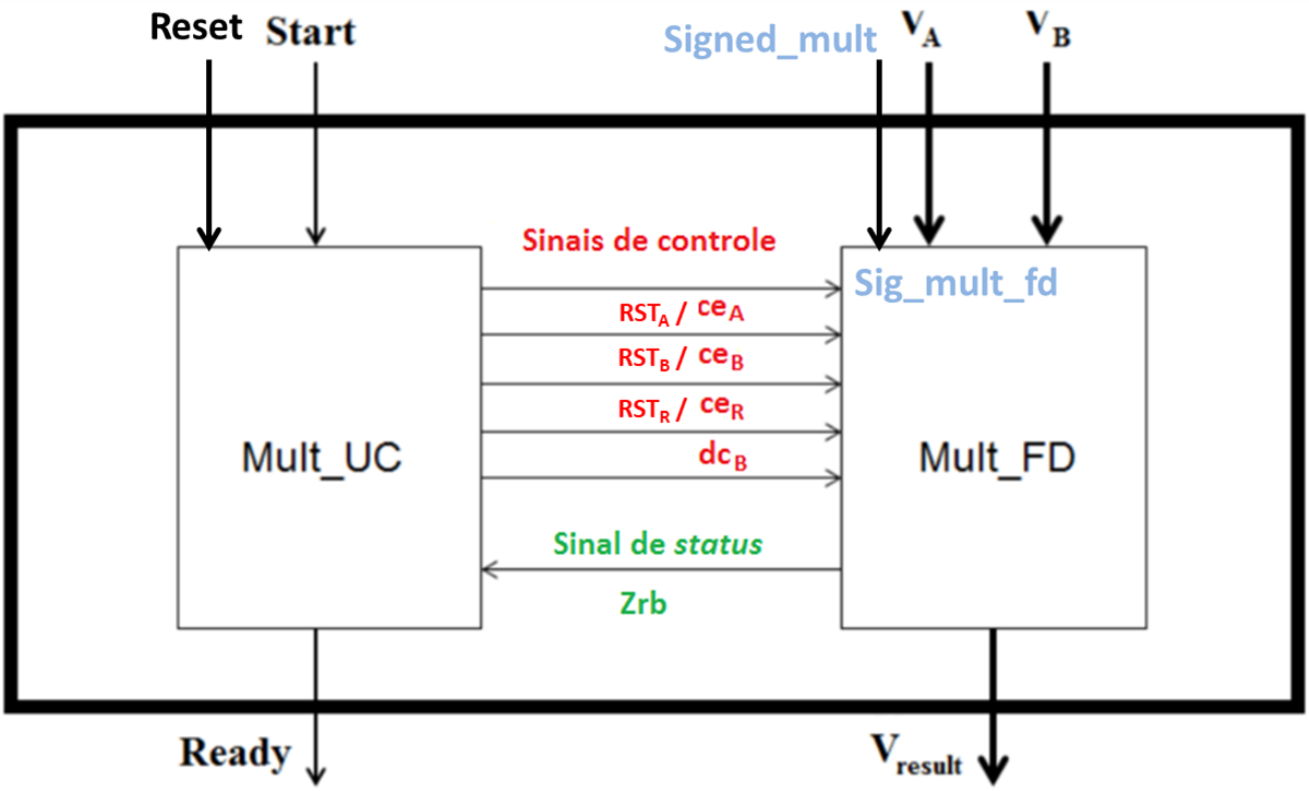

Figura II.5 – Diagrama de blocos das conexões entre a UC e o FD.

III-PROPOSTA DE OBTENÇÃO DA SOLUÇÃO E ENTREGAS

A entrega deste projeto consiste em:

- Fazer o projeto do **fluxo de dados** do multiplicador que contemple operações em magnitude ou em sinal na notação complemento de dois. A seleção entre operar em magnitude ou em sinal será feita pelo valor lógico da variável de entrada do FD "**sig_mult_fd**", que será uma entrada extra com relação ao multiplicador da parte I (Introdução).

Lá no ensino fundamental, quando fazíamos mentalmente uma operação de multiplicação com operandos com sinal tínhamos um procedimento para isto.

Fazíamos a operação de multiplicação com os módulos (magnitudes) dos operandos.

Verificávamos os sinais dos operandos e para sinais iguais dos operandos aplicávamos o sinal positivo ao resultado.

Para sinais diferentes dos operandos aplicávamos o sinal negativo ao resultado.

Devido ao fato de já dispor de um componente que calcula a multiplicação em magnitude um encaminhamento possível é o de utilizar o FD do multiplicador da parte I (Introdução) e introduzir um bloco que faria o que segue (ver Figura III.1):

- Verifica o valor lógico de "**sig_mult_fd**":
  - Se "**sig_mult_fd = 0**" – Nada a fazer, apenas encaminhar os valores de **Va** e **Vb** para o cálculo da multiplicação em magnitude;
  - Se "**sig_mult_fd = 1**":
    - Se **Va** e/ou **Vb** forem negativos, calcular a magnitude destes para encaminhar o cálculo do valor da multiplicação em magnitude;
    - Sintetizar um bloco detector de sinais diferentes:
      - Se os sinais de **Va** e de **Vb** forem iguais – Não há nada a fazer com o resultado da saída, que já terá o resultado positivo (com 0s à esquerda);
      - Se os sinais de **Va** e de **Vb** forem diferentes – Calcular a representação negativa do resultado da saída, que deve ter resultado negativo (com 1s à esquerda), dado que os sinais dos operandos eram diferentes.

Outro encaminhamento possível é reprojetar o fluxo de dados inteiro.

Você tem que tomar uma decisão de Engenharia!

A decisão é sua!

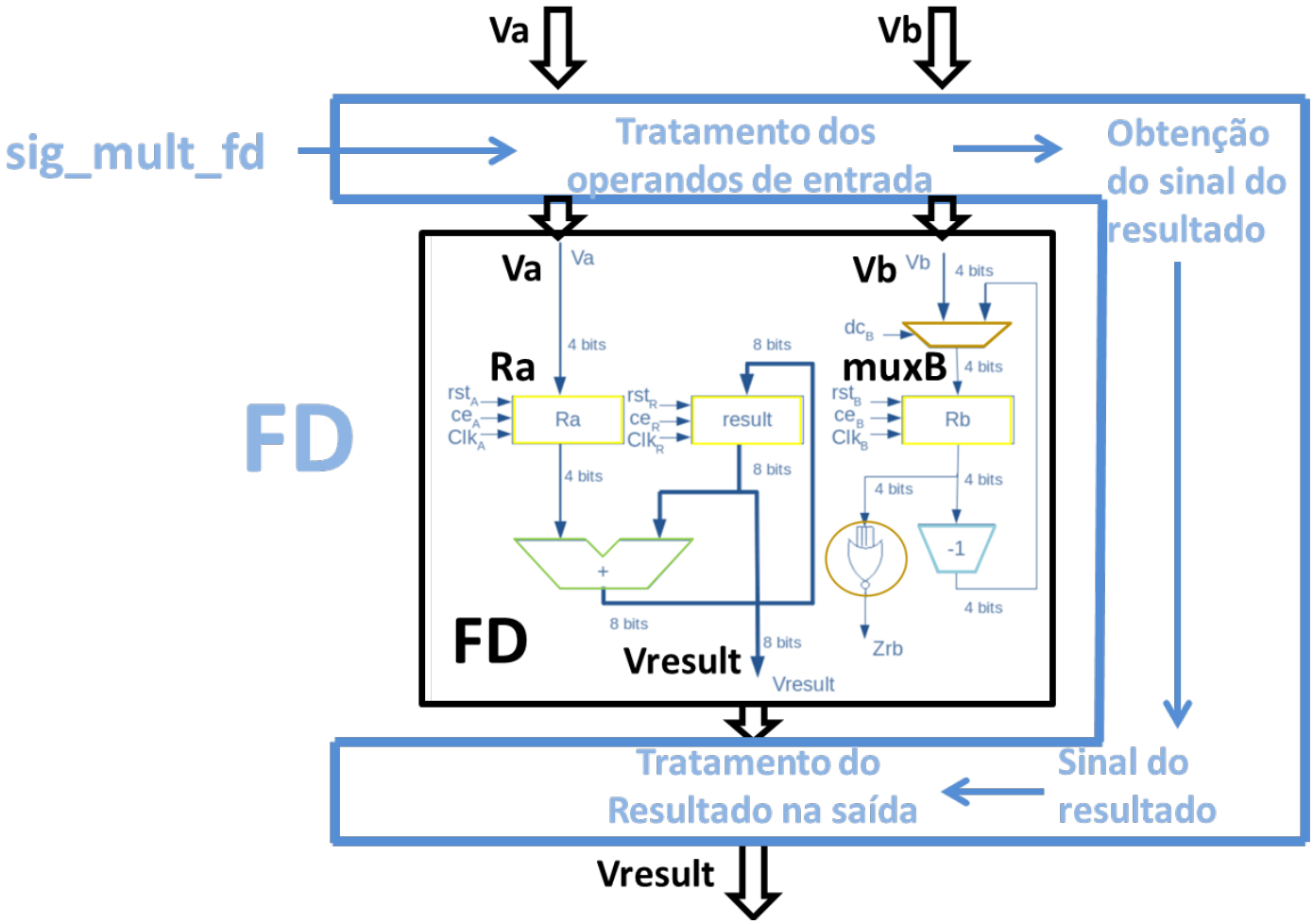

Figura III.1 – Diagrama de blocos das entradas, saídas e conexões internas do FD do multiplicador.

Quanto ao formato para **submissão ao Judge** cabem algumas considerações:

- O aluno deve submeter apenas um **<arquivo.vhd>**, que é aquele que contenha o **FD** e **todos seus componentes**, mas que **não contenha a UC nem o multiplicador como um todo**.
- A biblioteca **numeric_bit** pode ser utilizada para a soma ou não. Lembrar que já temos disponíveis os somadores completos (de 4 e 8 bits) no projeto do multiplicador original.
- O **FD** projetado deve realizar a **multiplicação com somas sucessivas**, e todos os produtos parciais devem aparecer na saída ***Vresult*** já com o sinal do produto final. Por exemplo, -3x4 deve passar por 0, -3, -6, -9 e -12, enquanto 4x-3 deve passar por 0, -4, -8 e -12. Note que estes produtos parciais dependem de qual operando será o **multiplicando** (**Va**) e de qual será o multiplicador (**Vb**).
- O **produto parcial** é **condição necessária para acertar o caso de teste**. Se o circuito **acertar o produto final** sem passar pelo **produto parcial**, será **considerado como errado**.
- Haverá um limite de tempo, de 50 ciclos de ***clock*** por multiplicação, que será adotado no ***testbench*** do ***judge***.

IV- INSTRUÇÕES PARA SUBMISSÃO DO PROJETO NO e-DISCIPLINAS

Há um link específico no e-Disciplinas para submissão deste projeto. Acesse-o somente quando estiver confortável para enviar sua solução. Em cada atividade, você pode enviar apenas um único arquivo codificado em UTF-8 (*). O nome do arquivo não importa, mas sim a descrição VHDL que está dentro. A entidade da síntese (implementação) do **Fluxo de Dados** do **multiplicador** que o aluno submeterá precisa estar, obrigatoriamente, de acordo com o que foi definido/especificado nas Partes I, II e III do enunciado e deve ser idêntica na sua solução ou o juiz não irá processar seu arquivo.

Quando acessar o link no e-Disciplinas, o navegador abrirá uma janela para envio do arquivo. Selecione-o e envie para o juiz. Jamais recarregue a página de submissão pois seu navegador pode enviar o arquivo novamente, o que vai ser considerado pelo juiz como um novo envio e pode prejudicar sua nota final. Caso desista do envio, simplesmente feche a janela.

Depois do envio, a página carregará automaticamente o resultado do juiz (**), quando você poderá fechar a janela. A nota dada pelo juiz é somente para a submissão que acabou de fazer. Esta atividade **permite até 5 submissões** e a nota será a maior nota dentre todas as suas submissões. Sua nota na atividade poderá ser vista no e-Disciplinas e pode diferir da nota dada pelo juiz.

Não use a biblioteca std_logic_1164, a textio e qualquer outra biblioteca não padronizada para minimizar possível fonte de problemas. Também se certifique que não imprime nada na saída da simulação.

Atenção: não atualize a página de envio e não envie a partir de conexões instáveis (e.g. móveis) para evitar que seu arquivo chegue corrompido no juiz.

(*) Qualquer editor de código moderno suporta UTF-8 (e.g. Atom, Sublime, Notepad++, etc).

(**) Pode demorar alguns segundos até o juiz processar seu arquivo.