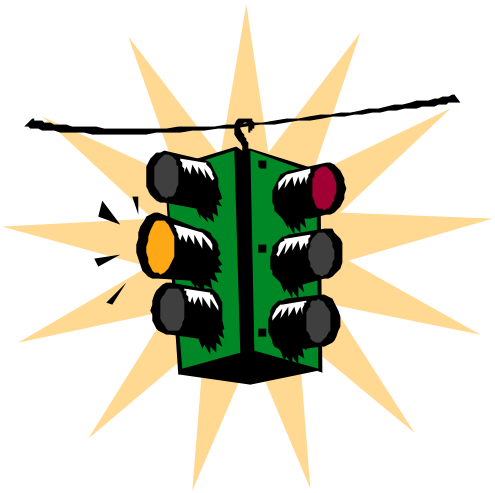
Controlador de Interrupções com Prioridades Rotativas



Julho de 2001

Trabalho realizado por:
Filipe Moreira
Ricardo Almeida

Controlador de Interrupções com Prioridades Rotativas

Introdução

Este relatório reporta-se ao trabalho efectuado por Filipe Moreira e Ricardo Almeida no âmbito da disciplina Arquitecturas e Projecto de Computadores integrante no Mestrado em Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto. O trabalho consistiu em desenvolver, em ferramentas FPGA, um controlador de prioridades rotativo.

Um controlador de prioridades rotativo atribui uma prioridade diferente aquando da ocorrência de interrupções simultâneas; basicamente se uma interrupção for atendida num dado instante e caso ocorra de seguida um novo pedido para essa interrupção simultaneamente com um pedido para outra interrupção, então a interrupção atendida anteriormente terá uma menor prioridade que a outra interrupção. Isto faz com que, por exemplo, no caso de uma interrupção, que tenha uma dada prioridade, estar a fazer pedidos de atendimento consecutivamente, qualquer outra interrupção com prioridade mais baixa nunca será atendida. Por outras palavras, é dada a prioridade menor à interrupção que está a ser executada, após se completar o seu tratamento.

Descrição do trabalho

O trabalho consistiu em implementar que obedecesse aos requisitos próprios de um controlador de interrupções; optou-se por elaborar quatro blocos básicos – *ireq*, *buffer*, *tabprior*, *controlo* -, descritos de seguida. O código, feito em *Verilog*, de cada um deste s quatro blocos encontra-se em anexo, bem como um esquema de todo o sistema com as suas interligações. De referir que nesse esquema encontra-se um bloco – *relogio* – que não é descrito, pois a sua inclusão só foi necessária para efeitos de simulação do circuito em *Xilinx*, ou seja, em termos de um circuito real este bloco não existirá.

ireq

Este bloco tem como finalidade enviar um sinal de pedido de interrupção sempre que haja um pedido de atendimento por parte do dispositivo ligado a este bloco. Contudo, pretendeu-se que só chegue um pedido de atendimento de cada vez ao processador; para tal, desde que o pedido é registado, é actuado um *disable* para os pedidos, caso estes ocorram, fazendo com que eventuais novos pedidos sejam ignorados; só voltarão a ser considerados após o *enable* para os pedidos após o processador atender o pedido feito inicialmente.

Para implementar este bloco, recorreu-se ao código, mostrado em anexo e que se passa a explicar:

Entradas do bloco:

reset – quando ocorre uma transição positiva deste sinal, a saída é colocada a zero e manter-se-á assim enquanto esta linha não for 0

int – quando ocorre uma transição positiva deste sinal, e desde que *reset* esteja a 0, a saída é colocada a 1

Saídas do bloco:

ireq – é a saída do bloco; é colocada a 1 sempre que ocorra um pedido de interrupção (indicado pela transição positiva de *int*) desde que *reset* esteja a 0; quando *reset* passa a 1, esta saída é colocada a 0.

buffer

Este bloco tem como finalidade armazenar num vector – *buffer* – quais as interrupções que pretendem ser atendidas e enviar um sinal de pedido de atendimento de interrupção, sempre que haja uma interrupção para ser atendida. Igualmente, sempre que uma interrupção é atendida – sinalizado por *iack* -, é colocado a 0 a posição respectiva dessa interrupção no *buffer* e é verificado se ainda existe qualquer outro pedido de interrupção pendente no *buffer*. Há, ainda, a possibilidade de pôr o *buffer* que contém as interrupções pendentes completamente a 0 (corresponde a limpar todos os pedidos de interrupção pendentes).

Quando se detecta um sinal de *reset* colocam-se todos os bits do *buffer* a 0.

Quando ocorre um *iack* começa-se por colocar a posição da interrupção já atendida no *buffer* a 0 - a posição da interrupção é dada pelo seu número que é especi9ficado por *iaddr* -, bem como se coloca a 0 o sinal *int* que serve para avisar o processador de que há uma interrupção pendente.

Quando ocorre um pedido de interrupção proveniente de um dos blocos ***ireq*** então a posição respectiva do *buffer* é colocada a 1.

tabprior

Este bloco tem como finalidade a especificação de qual a prioridade de cada interrupção e fazer a rotação das prioridades.

Para tal, recorreu-se a oito conjuntos de 3 flip-flop's (*ff0* a *ff7*), que indicam qual a prioridade com peso 7 (a que está colocada em *ff7*), qual a prioridade com peso 6 (colocada em *ff6*) e por aí adiante até à prioridade com peso 0 (colocada em *ff0*).

Quando se reinicia o sistema ou quando se reinicializa as prioridades, é ordenada a ordem de prioridades nos flip-flop's: em *ff0* coloca-se o número correspondente à interrupção 7 (a que terá menor prioridade), em *ff1* coloca-se o número correspondente à interrupção 7 e por aí for até se colocar em *ff7* o número 0 (a interrupção será a que terá maior prioridade).

Quando se recebe um *iack*, faz-se uma rotação dos números das interrupções nos flip-flpo's *ffx*, colocando em *ff7* o número da interrupção imediatamente a seguir à que foi atendida – especificada por *iaddr* – e por aí sucessivamente até se colocar em *ff0* o número correspondente à interrupção atendida (que passará a ter a menor prioridade).

controlo

Este bloco tem como finalidade enviar ao processador um pedido de interrupção, especificar qual a interrupção que está a fazer o pedido e fazer o *disable* do bloco ***ireq*** para a interrupção que está ser atendida.

Quando se recebe um sinal *iack*, começa-se por desactivar o pedido de interrupção ao processador – dado pela linha *int* -, e por reactivar o bloco ***ireq*** da interrupção que acabou de ser atendida. De seguida verifica-se que existe algum pedido de interrupção pendente; em caso afirmativo, envia um sinal de *int* ao processador e o número da interrupção – *iaddr* – que requer atendimento com maior prioridade.

Quando se recebe um sinal *inputireq*, verifica-se qual das interrupções te maior prioridade e é enviado um sinal *int* e o número dessa interrupção – *iaddr* – ao processador; além disso desactiva-se o bloco ***ireq*** – através de *reseti* – da interrupção que será atendida.

Para determinar qual a interrupção com maior prioridade recorre-se ao seguinte algoritmo: em primeiro lugar verifica-se se a posição *ff7* do *buffer* que contém as interrupções pendentes está a 1; em caso afirmativo, então esse será a interrupção que será atendida e seu número é dado por *ff7*; em caso negativo, verifica-se se a posição *ff6* do *buffer* que contém as interrupções pendentes está a 1; em caso afirmativo, então esse será a interrupção que será atendida e seu número é dado por *ff6* e por aí sucessivamente. A diferença entre a ocorrência de um *iack* ou de um *inputireq* está e que no primeiro caso é necessário verificar se a posição dada por *ff0* está a 0 ou a 1, enquanto que no segundo caso, se ocorrer um *inputireq* e se não for qualquer uma das outras interrupções enumeradas entre *ff7* e *ff1*, então certamente será a enumerada em *ff0* - pois alguma teve de ser.

Conclusões

A realização deste trabalho permitiu-nos uma maior familiarização com as ferramentas FPGA *Verilog* e *Xilinx* no seguimento do que foi aprendido na disciplina de Projecto de Circuitos de Sistemas Digitais leccionada no 1º semestre.

Permitiu, também, desenvolver um raciocínio específico para operar com este tipo de ferramentas, o que para nós foi relativamente difícil devido à nossa habituação ao raciocínio para programar em linguagens de mais alto nível.

Por outro lado, permitiu uma melhor compreensão do funcionamento das interrupções.

Todo o código foi desenvolvido de raiz, bem como a arquitectura por nós elaborada e desenvolvida.

Bibliografia

- Manual de utilização do *Verilog*

- Manual de utilização do *Xilinx*

- Apontamentos das aulas de Projecto de Circuitos de Sistemas Digitais do 1º semestre do ramo de Informática Industrial do Mestrado em Engenharia Electrotécnica e de Computadores leccionado na Faculdade de Engenharia da Universidade do Porto

- Apontamentos da disciplina Sistemas Digitais II do curso de Engenharia Informática leccionado na Escola Superior de Tecnologia e Gestão do Instituo Politécnico de Bragança

- **Computer Organization and Architecture**
  Designing For Performance
  William Stallings
  Prentice Hall International Editions

- **Microcomputers and Microprocessors**
  The 8080, 8085, and Z-80 programming, interfacing and troubleshooting
  John Uffenbeck
  Prentice Hall International Editions

- **Microprocessors and Interfacing**
  Programming and Hardware
  Douglas V. Hall
  McGraw-Hill International Editions

Anexos

Bloco ireq

```
module ireq(int,reset,ireq);
input int, reset;
output ireq;
reg ireq;

always @ (posedge reset or posedge int)
begin
  if (reset)
    ireq = 0;
  else
    ireq = 1;
end

endmodule
```

Bloco buffer

```
module buffer(ira, irb, irc, ird, ire, irf, irg, irh, iack, iaddr, reset, ireq, buffer);
input ira, irb, irc, ird, ire, irf, irg, irh, iack, reset;
input [2:0] iaddr;
output [7:0] buffer;
output ireq;

reg [7:0] buffer;
reg [2:0] i;
reg ireq, haint;

always @ (iack or ira or irb or irc or ird or ire or irf or irg or irh or reset)
begin
  if (reset)
  begin
    buffer[0] = 0;
    buffer[1] = 0;
    buffer[2] = 0;
    buffer[3] = 0;
    buffer[4] = 0;
    buffer[5] = 0;
    buffer[6] = 0;
    buffer[7] = 0;
  end           // fim do reset
  else          // não foi reset
  begin
  if (iack)     // foi iack
  begin
    i = iaddr;
    buffer[i] = 0;
    ireq = 0;
  end

  else          // não foi um iack
  begin
    if (ira)
    begin
      buffer[0] = 1;
    end
    if (irb)
    begin
      buffer[1] = 1;
    end
    if (irc)
    begin
      buffer[2] = 1;
    end
    if (ird)
    begin
```

```
      buffer[3] = 1;
    end
    if (ire)
    begin
      buffer[4] = 1;
    end
    if (irf)
    begin
      buffer[5] = 1;
    end
    if (irg)
    begin
      buffer[6] = 1;
    end
    if (irh)
    begin
      buffer[7] = 1;
    end
    ireq = 1;
  end           // fim do ir
  end           // fim do iack ou ir
end             // fim do always @ …

endmodule
```

Bloco tabprior

```
module tabprior(reset,iack,iaddr,ff0,ff1,ff2,ff3,ff4,ff5,ff6,ff7);
input reset, iack;
input [2:0] iaddr;
output [2:0] ff0, ff1, ff2, ff3, ff4, ff5, ff6, ff7;
reg [2:0] ff0, ff1, ff2, ff3, ff4, ff5, ff6, ff7;

// ff0 tem o número da interrupção com menos prioridade e ff7 com mais prioridade
always @ (posedge reset or posedge iack) begin
  if (reset)
  begin
    ff0 = 7;
    ff1 = 6;
    ff2 = 5;
    ff3 = 4;
    ff4 = 3;
    ff5 = 2;
    ff6 = 1;
    ff7 = 0;
  end
  else
  begin
    ff7 = iaddr + 1;
    ff6 = iaddr + 2;
    ff5 = iaddr + 3;
    ff4 = iaddr + 4;
    ff3 = iaddr + 5;
    ff2 = iaddr + 6;
    ff1 = iaddr + 7;
    ff0 = iaddr;
  end
end
endmodule
```

Bloco controlo

```
module controlo(iack, inputireq, buffer, ff7, ff6, ff5, ff4, ff3, ff2, ff1, ff0, iaddr, int,
reset);
input iack, inputireq;
input [7:0] buffer;
input [2:0] ff0, ff1, ff2, ff3, ff4, ff5, ff6, ff7;
output int;
output [7:0] reset;
output [2:0] iaddr;

reg [2:0] iaddr;
reg int;
reg [7:0]reset;

always @ (inputireq or iack)
begin
  if (iack)
  begin
    int = 0;
    reset[iaddr] = 0;
    if (buffer[ff7] == 1)
    begin
      iaddr = ff7;
      reset[iaddr] = 1;
      int = 1;
    end
    else
    begin
      if (buffer[ff6] == 1)
      begin
        iaddr = ff6;
        reset[iaddr] = 1;
        int = 1;
      end
      else
      begin
        if (buffer[ff5] == 1)
        begin
          iaddr = ff5;
          reset[iaddr] = 1;
          int = 1;
        end
        else
        begin
          if (buffer[ff4] == 1)
          begin
            iaddr = ff4;
            reset[iaddr] = 1;
            int = 1;
```

```
        end
        else
          begin
            if (buffer[ff3] == 1)
            begin
              iaddr = ff3;
              reset[iaddr] = 1;
              int = 1;
            end
            else
            begin
              if (buffer[ff2] == 1)
              begin
                iaddr = ff2;
                reset[iaddr] = 1;
                int = 1;
              end
              else
              begin
                if (buffer[ff1] == 1)
                begin
                  iaddr = ff1;
                  reset[iaddr] = 1;
                  int = 1;
                end
                else
                  if (buffer[ff0] == 1)
                  begin
                    iaddr = ff0;
                    reset[iaddr] = 1;
                    int = 1;
                  end
              end
            end
          end
        end
    end
  end
end

// fim do iack
else    // foi um inputreq

begin
if (buffer[ff7] == 1)
begin
  iaddr = ff7;
  reset[iaddr] = 1;
end
else
```

```
begin
  if (buffer[ff6] == 1)
  begin
    iaddr = ff6;
    reset[iaddr] = 1;
  end
  else
  begin
    if (buffer[ff5] == 1)
    begin
      iaddr = ff5;
      reset[iaddr] = 1;
    end
    else
      begin
        if (buffer[ff4] == 1)
        begin
          iaddr = ff4;
          reset[iaddr] = 1;
        end
        else
        begin
          if (buffer[ff3] == 1)
          begin
            iaddr = ff3;
            reset[iaddr] = 1;
          end
          else
          begin
            if (buffer[ff2] == 1)
            begin
              iaddr = ff2;
              reset[iaddr] = 1;
            end
            else
            begin
              if (buffer[ff1] == 1)
              begin
                iaddr = ff1;
                reset[iaddr] = 1;
              end
              else
                begin               // certamente foi uma delas
                  iaddr = ff0;
                  reset[iaddr] = 1;
                end
            end
          end
        end
      end
```

```
    end
  end
  int = 1;
  end
end

endmodule
```

Esquema do circuito