Projecto, síntese e implementação em FPGA de um sistema motorizado

Daniel Caetano

***Resumo* - Este artigo descreve a implementação em FPGA de um sistema provido de motores controlado por instruções recebidas do computador hospedeiro via USB, assim como a implementação em C++ da consola destinada ao seu controlo. São descritas as funções básicas de controlo de movimento, nomeadamente controlo da velocidade, da direcção do movimento e do ângulo numa curva. As instruções são introduzidas no teclado pelo acto de premir a tecla específica para uma determinada acção, sendo transmitidas para a FPGA em tempo real.**

***Abstract* - This paper describes the implementation in FPGA of a motor-provided system controlled by instructions received from the host computer via USB, as well as the implementation in C++ of a console designed to control the system. Basic movement control functions are described, such as speed, direction and curvature angle control. By pressing a specific key in the keyboard, the correspondent instruction is sent to the FPGA in real time.**

I. INTRODUÇÃO

Em anos recentes, a tecnologia FPGA (Field-programmable gate array) tem verificado grandes avanços quer em termos tecnológicos quer em aplicabilidade, estando cada vez mais em pé de igualdade com a tecnologia ASIC (Application-specific integrated circuit). Visto este facto, é cada vez mais necessária a formação de profissionais especializados nesta área, sendo a oferta de emprego neste ramo da electrónica cada vez maior. Torna-se então necessário fomentar o interesse por parte dos alunos, quer com a demonstração de aplicações desta tecnologia quer com propostas de projectos que sejam apelativos e ao mesmo tempo didáticos. Neste artigo é então proposto um projecto que, sendo relativamente simples, aborda alguns dos elementos fundamentais na implementação de sistemas reconfiguráveis.

O projecto consiste na implementação de um sistema provido de motores controlado por instruções recebidas do computador hospedeiro via USB. Pretende-se que sejam implementadas funções básicas de controlo de movimento, e que as instruções sejam lidas de um teclado e transmitidas em tempo real.

Este trabalho foi desenvolvido no âmbito de uma Bolsa de Integração na Investigação oferecida pelo IEETA, sob a orientação de Prof. Iouliia Skliarova [1].

II. TRAÇOS GERAIS DO SISTEMA

O sistema foi implementado na placa *Nexys-2* [2], que contém uma FPGA da família Spartan-3E. Foi utilizado o software *ISE 11.1* [3] para síntese e implementação a partir do código VHDL (VHSIC Hardware Description Language) e *Microsoft Visual Studio 2008* [4] para compilação do código C++, usado para criar o interface utilizador-PC e a estrutura de transmissão de instruções. A Fig-1 ilustra a utilização do software mencionado.

Foram criadas funções básicas de controlo de movimento, nomeadamente controlo da velocidade, da direcção do movimento e do ângulo descrito numa curva. As instruções são introduzidas no teclado pelo acto de premir a tecla específica para uma determinada acção, sendo transmitidas para a FPGA em tempo real.

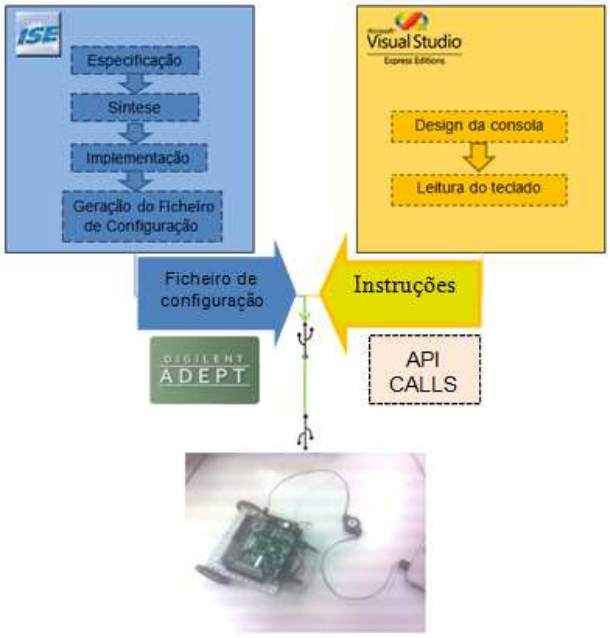


Fig. 1 - Diagrama que ilustra as ferramentas utilizadas na concepção do sistema.

III. ESPECIFICAÇÕES

A. *Interface utilizador-PC*

Ao correr o programa criado para o interface temos uma consola (Fig-2) a correr na linha de comandos que verifica se as teclas válidas foram premidas no teclado. As teclas destinadas ao controlo do movimento são (ver Fig-3):

D - curva à direita
A - curva à esquerda
W - aumentar velocidade
S - diminuir velocidade
I - aumentar ângulo de curvatura
J - diminuir ângulo de curvatura
R – reset

Cada vez que o programa identifica que foi premida uma tecla válida, o byte correspondente é seleccionado e a transmissão efectuada, no caso do incremento/decremento de velocidade ou ângulo. No caso de ser premida uma tecla de direcção (A ou D), o programa envia o byte correspondente ao inicio de curva, enviando depois o correspondente ao final de curva quando detectar que a tecla foi solta. Ao premir a tecla reset (R) as variáveis *speed* e *angle* voltam ao valor inicial 0, e o circuito de controlo volta ao estado inicial.

B. *Transmissão de instruções*

A estrutura de transmissão de dados é baseada num código de demonstração disponibilizado pela Digilent [5], [6]. Esta estrutura cria um barramento bidireccional de 8 bits e linhas de *handshaking*. No entanto, neste sistema apenas é usada a direcção PC-FPGA. É possível endereçar até 256 registos de 8 bits, mas neste caso apenas 1 registo é utilizado para escrita do byte correspondente à tecla premida.

A Digilent disponibiliza também um conjunto de funções de API (Applications Programming Interface) que trabalham sobre a estrutura criada. Estas funções permitem escrita e leitura de registos criados na FPGA a partir de programas escritos em C ou C++.

C. *Processamento de instruções*

Na FPGA temos um conjunto de processos paralelos que analisam e executam as instruções recebidas, ilustrados na Fig-4. Dois dos processos controlam uma FSM (Finite State Machine [7]) que contém três estados, correspondentes ao movimento em frente, à esquerda e à direita. Um outro processo, *speed/angle updater*, actualiza as variáveis *speed* e *angle* independentemente da máquina de estados.

Fig-2 Consola criada em C++ para interface Utilizador-PC, com indicação da função correspondente à tecla premida.

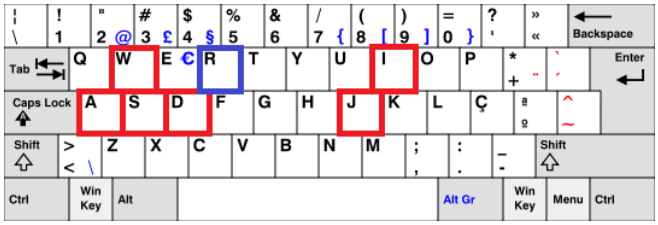


Fig-3 Teclas destinadas ao controlo do movimento

Como se verifica na Fig-5, a máquina permanece num estado não-inicial apenas enquanto uma das teclas direccionais está a ser premida, voltando ao estado inicial quando esta é solta. É também impossível saltar de um estado não-inicial directamente para outro não-inicial.

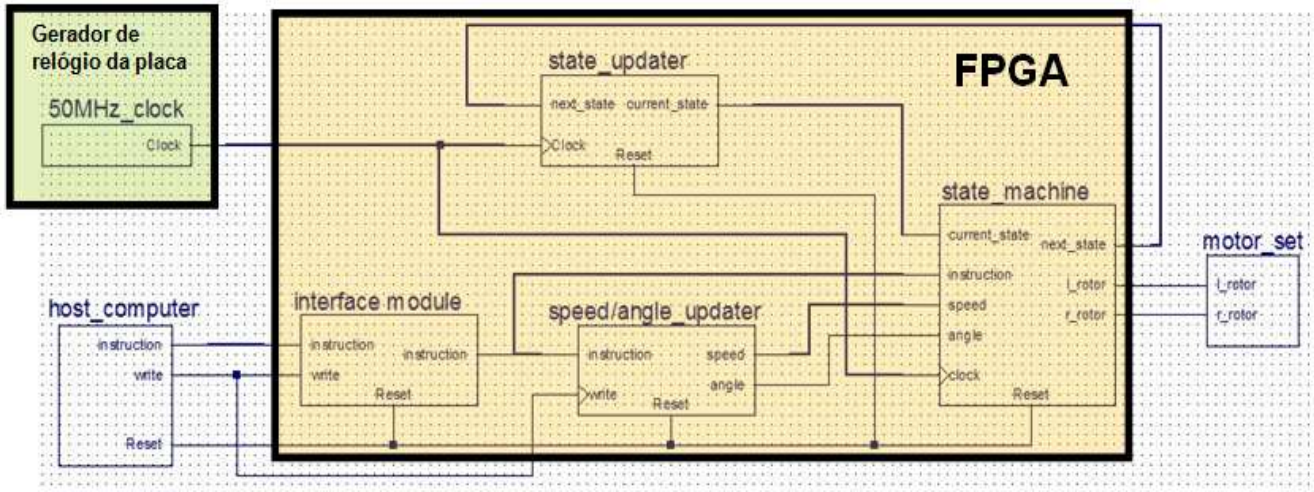
Ao contrário dos processos que regem a FSM, o processo que actualiza as variáveis *speed* e *angle* não é sensível ao sinal de relógio, mas sim ao sinal *write* da estrutura de transmissão de dados. Cada vez que um byte é recebido pela FPGA, este processo compara-o com o byte recebido anteriormente e detecta transições ascendentes de algum dos sinais. Caso alguma transição ascendente seja detectada, e o incremento/decremento da variável correspondente não a faça exceder os limites especificados ($0 \leq speed \leq 10$, $0 \leq angle \leq 6$), a variável é então actualizada conforme o exemplo seguinte.

novo valor do sinal antigo valor do sinal

```
if(signal_vect1(3)='1' and signal_vect2(3)='0'
and (speed<10)) then
        speed<=speed +1;
end if;
```

D. *Execução das instruções*

Após definidas as variáveis *speed* e *angle*, assim como determinada a direcção em que se pretende que o movimento seja efectuado, falta então enviar essa informação aos motores. A velocidade de cada motor é controlada por PWM (*Pulse-Width Modulation*) tendo-se agrupado cada 10 pulsos de 50 MHz (frequência de



relógio da placa) num pulso de 5MHz, sendo então possível controlar cada porção de 10% do seu *duty-cycle*.

```
fwd_d_c:=duty_cycle_table(speed);
```

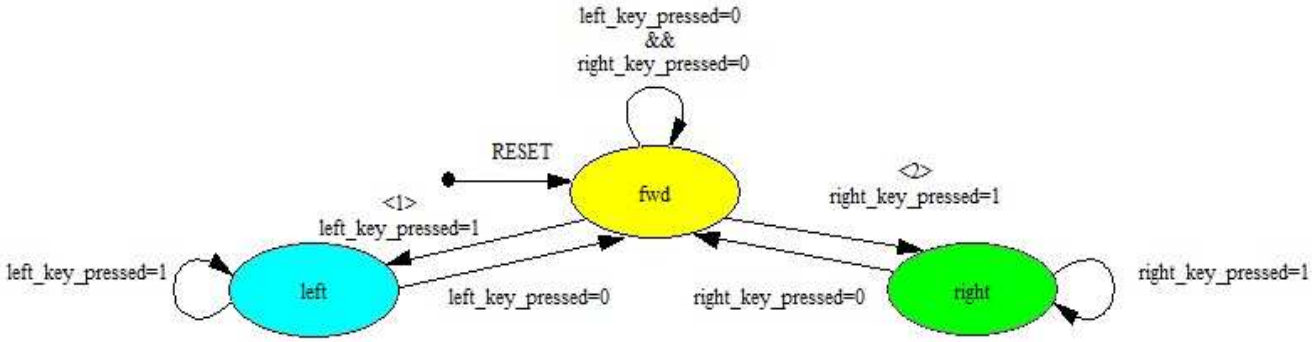


Fig-5 Diagrama de estados da FSM implementada em FPGA.

Fig 4 Diagrama de blocos representando os processos paralelos criados na FPGA

Como se escolheu dividir o período em porções de 10%, é então possível obter-se 10 velocidades distintas. O código seguinte ilustra os 10 *duty-cycles* possíveis do sinal a ser transmitido aos motores.

```
constant d_c_table:d_c_vector:=(0=>"0000000000",
                                1=>"1000000000",
                                2=>"1100000000",
                                3=>"1110000000",
                                4=>"1111000000",
                                5=>"1111100000",
                                6=>"1111110000",
                                7=>"1111111000",
                                8=>"1111111100",
                                9=>"1111111110",
                               10=>"1111111111");
```

Cada '1' em cada *d_c_vector* corresponde a 10% de período. No caso `sample_d_c_vector:=d_c_table(4)`, por exemplo, temos um *duty-cycle* de 40%.

No caso da FSM se encontrar no estado *fwd* e, consequentemente, a direcção do movimento ser em frente, pretende-se que os dois motores funcionem com o mesmo *duty-cycle*. Esse *duty-cycle* é então escolhido de acordo com a variável *speed*.

No caso da FSM se encontrar num estado de curva (*right* ou *left*), um dos motores terá de trabalhar com um *duty-cycle* inferior ao outro. É assim definida uma variável dependente de *speed* e *angle* correspondente ao *duty-cycle* do motor do lado para o qual é efectuada a curva.

```
turn_d_c:=duty_cycle_table(speed-angle);
```

No caso da variável angle possuir valor maior ou igual à variável speed, é então atribuído o valor `turn_d_c:=duty_cycle_table(0)`, ficando o motor parado enquanto a curva é efectuada.

A cada ciclo de relógio é necessário seleccionar um dos bits do *d_c_vector* para enviar a cada motor. Para tal efeito é usado um contador simples ilustrado no exemplo seguinte.

```
if rising_Edge(clkMain) then
(…)
       case current_state is
      (…)
```

```
        when right=>l_rotor(0)<= fwd_d_c(ind);
                    r_rotor(0)<= turn_d_c(ind);
                    if right_key_pressed='1' then
                        next_state<=right;
                    else
                        next_state<=fwd;
                    end if;
        end case;

        if ind=10 then
                ind:=0;
        else
                ind:= ind+1;
        end if;
end if;
```

O exemplo anterior mostra a parte do código que é relevante durante uma curva à direita. Quando neste estado, as variáveis *speed* e *angle* podem ser actualizadas, mas tal não influenciará o movimento, pois *fwd_d_c* e *turn_d_c* apenas são actualizadas no estado *fwd*. O diagrama temporal corresponde a este exemplo para o caso de speed=7 e angle=5.

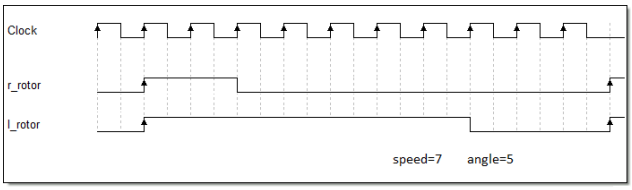


Fig-6 Diagrama temporal para o caso da curva à direita

IV. CONCLUSÕES

Todo o sistema foi testado e está a funcionar correctamente, tendo-se verificado que todas as instruções produzem o resultado esperado.

De um ponto de vista pedagógico, a construção deste sistema pode ser visto como um bom ponto de partida para a aprendizagem das funcionalidades básicas da linguagem VHDL, assim como a implementação de circuitos de controlo e comunicação com outros dispositivos, podendo ser integrada como trabalho prático em cadeiras como SDR [8], LDH [9] ou CR [10].

A vantagem de um projecto desta natureza em relação aos tradicionais projectos propostos nas cadeiras referidas é a maior facilidade de observação dos resultados, assim como a possibilidade de, a partir da base aqui construída, os alunos poderem desenvolver livremente novas funcionalidades para o sistema (controlo wireless, sensores de obstáculos, etc.).

A criação de trabalhos práticos mais apelativos e de um ambiente de competição (através do desenvolvimento livre de novas funcionalidades) é um passo fundamental para atraír mais alunos para este ramo em expansão da electrónica.

V. REFERÊNCIAS

[1] Página pessoal da Prof. Iouliia Skliarova: http://www.ieeta.pt/~iouliia/

[2] Digilent Nexys-2 Board Reference Manual: http://digilentinc.com/Data/Products/NEXYS2/Nexys2_rm.pdf

[3] Xilinx ISE Software manuals: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/manuals.pdf

[4] Visual Studio, Visual Studio Developer Center: http://msdn.microsoft.com/en-us/library/52f3sw5c.aspx

[5] Digilent Parallel Interface Module Reference Manual: http://www.digilentinc.com/Data/Products/ADEPT/DpimRef%20programmers%20manual.pdf

[6] Digilent Port Communications Programmers Reference Manual: http://www.digilentinc.com/Data/Products/ADEPT/DPCUTIL%20Programmers%20%20Reference%20Manual.pdf

[7] John F. Wakerly, Digital Design, Principles and Practices, 4th ed., 2005, Prentice Hall

[8] Página da disciplina *Sistemas Digitais Reconfiguráveis*: http://www.ieeta.pt/~skl/SDR_V.html

[9] Página da disciplina *Linguagens de Descrição de Hardware*: http://www.ieeta.pt/~skl/LDH.html

[10] Página da disciplina *Computação Reconfigurável*: http://www.ieeta.pt/~skl/CR_V.html